PSOC™ Control C3 アーキテクチャ

Reference manual

本書について

適用範囲と目的

本書は、インフィニオンの PSOC™ Control C3 シリーズのマイクロコントローラーに関する包括的な情報を提供します。これら 32 ビット シングルコア チップは、リアルタイム制御、拡張センシング、セキュア、低消費電力アプリケーション向けに特別に設計された高性能、低消費電力、セキュアな MCU ファミリーの一部です。

対象者

本書は、インフィニオンの PSOC™ Control C3 シリーズ マイクロコントローラーのアーキテクチャを理解したいハードウェア エンジニア、組み込みソフトウェア開発者、システム アーキテクトを対象としています。

目次

本書について . . . 1
目次 . . . 2
1 はじめに . . . 20
1.1 詳細な特長 . . . 20
1.2 アーキテクチャ . . . 23
1.3 文書の構成と表記法 . . . 24
1.3.1 主要セクション . . . 24
1.3.2 本書の表記法 . . . 24
1.3.2.1 レジスタの表記法 . . . 24
1.3.2.2 数値の表記法 . . . 24
1.3.2.3 測定単位 . . . 24
1.3.2.4 略語 . . . 25
2 CPU サブシステム (CPUSS) . . . 31
2.1 特長 . . . 31
2.2 アーキテクチャ . . . 31
2.2.1 CPUSS . . . 31
2.2.2 アドレスマップとメモリマップ . . . 32
2.2.2.1 待機状態のルックアップテーブル . . . 34
2.3 動作モードと特権レベル . . . 34
2.3.1 セキュリティ状態 . . . 35
2.3.2 動作モード . . . 35
2.3.3 動作状態 . . . 35
2.3.4 特権レベル . . . 35
2.4 命令セット . . . 36
2.5 レジスタの概要 . . . 36
3 バス インフラストラクチャー . . . 41
3.1 SYSCPUSS AHB5 インターコネクト . . . 42
3.1.1 AHB5 バス マトリックス インターコネクト . . . 42
3.1.2 SYSCPU バス . . . 43
3.1.2.1 Cortex® -M33 C-AHB バス . . . 43
3.1.2.2 Cortex® -M33 S-AHB バス . . . 43
3.1.2.3 DMA バス . . . 43
3.2 AHB5 MMIO . . . 43
3.3 ペリフェラル インターコネクト (PERI) . . . 43
3.3.1 AHB バスのエラー処理 . . . 45
3.4 バス マスター ペリフェラル . . . 45
3.4.1 ダイレクトメモリ アクセス (DMA-DW) . . . 45
3.4.2 CryptoLite . . . 45

目次

4 プロセッサ間通信 (IPC) 46
4.1 特長 46
4.2 アーキテクチャ 46
4.2.1 IPC チャネル 46
4.2.2 IPC 割込み 47
4.2.3 IPC チャネルと割込み 48
4.3 ロックの実装 49
4.4 メッセージ パッシング 50
4.5 代表的な使用モデル 52
4.5.1 全二重通信 52
4.5.2 独立したイベント処理による半二重通信 52
4.5.3 共有イベント処理による半二重通信 53
5 フラッシュ 55
5.1 特長 55
5.2 設定 55
5.2.1 ブロック図 55
5.2.2 フラッシュの構造 56
5.2.3 バンクモードとアドレスマッピング 58
5.3 フラッシュコントローラー 58
5.3.1 電力モード 59
5.4 RWW (Read While Write) 対応 59
5.5 ECC 59
5.5.1 ECC 生成/エンコーディング 60
5.5.2 ECC (エラー) インジェクション 60
5.6 セキュリティと保護 61
5.7 API ライブラリ 61
6 SRAM コントローラー 62
6.1 特長 62
6.2 アーキテクチャ 62
6.2.1 AHB5 読み出し転送 62
6.2.2 32 ビット AHB5 書き込み転送 62
6.2.3 AHB5 部分書き込み転送 62
6.2.4 電源パーティション制御 62
6.3 エラー訂正符号 (ECC) 63
6.3.1 ECC エラー インジェクション 63
6.3.2 ECC エラー報告機能 63
6.3.2.1 AHB5 部分書き込み転送 63
6.3.2.2 AHB5 読み出し転送 63
6.3.2.3 訂正可能な ECC エラー報告 64
6.3.2.4 訂正不可能な ECC エラー報告 64

目次

6.4 待機状態 64
7 割込み 65
7.1 特長 65
7.2 アーキテクチャ 65
7.3 割込みと例外操作 66
7.3.1 割込み/例外処理 66
7.3.2 レベル割込み 66
7.3.3 例外ベクタ テーブル 67
7.4 例外ソース 69
7.4.1 リセット例外処理 69
7.4.2 ノンマスカブル割込み例外 69
7.4.3 HardFault 例外 70
7.4.4 MemManage フォールト例外 70
7.4.5 BusFault 例外 70
7.4.6 UsageFault 例外 70
7.4.7 SecureFault 例外 71
7.4.8 監視呼び出し (SVCall) 例外 71
7.4.9 DebugMonitor 例外 71
7.4.10 PendSupervisory (PendSV) 例外 71
7.4.11 システムティック (SysTick) 例外 71
7.5 割込みソース 72
7.6 割込み/例外対象のセキュリティ状態とバンキング 76
7.6.1 割込み対象のセキュリティ状態 77
7.6.2 GPIO のセキュア割込みと非セキュア割込み 78
7.7 割込み/例外の優先順位 78
7.8 割込みのイネーブルとディセーブル 78
7.9 割込み/例外状態 79
7.9.1 保留中の割込み/例外 80
7.10 割込み/例外のスタック使用 81
7.11 割込みと低消費電力モード 81
7.12 割込み/例外 - 初期化/設定 82
8 フォールト監視 83
8.1 特長 83
8.2 アーキテクチャ 83
8.2.1 フォールト レポート 84
8.2.2 信号インターフェース 84
8.2.3 モニター 85
8.2.4 低消費電力動作 85
8.2.5 フォールト構造の使用 86
8.2.6 CPU 例外とフォールト監視 86

目次

8.3 フォールト原因 86
8.4 フォールト エンコーディング 89
8.4.1 PERI マスターインターフェース PPC 違反 89
8.4.2 PERI ペリフェラル PPC PC マスク違反 89
8.4.3 PERI ペリフェラル グループ AHB タイムアウト 90
8.4.4 PERI ペリフェラル グループ AHB 違反 90
8.4.5 RAMC0 MPC 違反 91
8.4.6 AHB ブリッジエラー フォールト 91
8.4.7 PROMC MPC 違反 92
8.4.8 RAMC0_C ECC mmio 92
8.4.9 RAMC0_NC ECC mmio 92
8.4.10 FLASHC MPC フォールト 92
8.4.11 FLASHC バスアクセスフォールト 93
8.4.12 FLASHC main_c ECC mmio フォールト 94
8.4.13 FLASHC main_nc ECC mmio フォールト 94
8.4.14 FLASHC FM CTL フォールト 94
8.4.15 FLASHC FM r_q フォールト 95
8.4.16 FLASHC フラッシュ CTL フォールト 95
8.4.17 FLASHC アドレス ホール フォールト 96
8.4.18 FLASHC ロック フォールト 96
8.4.19 SRSS クロック監視フォールト 96
8.4.20 SRSS 電源管理フォールト 96
8.5 レジスタの概要 97

9 ダイレクトメモリ アクセス (DMA - DW) 98
9.1 特長 98
9.2 アーキテクチャ 99
9.2.1 標準フロー 100
9.3 チャネル 100
9.3.1 チャネルの優先度 100
9.3.2 チャネルのセキュリティ属性 100
9.3.3 チャネル状態 101
9.3.4 チャネル割込み生成 101
9.4 記述子 102
9.4.1 記述子メモリ転送タイプ 103
9.4.1.1 シングル転送 103
9.4.1.2 1D メモリ転送 103
9.4.1.3 2D メモリ転送 104
9.4.1.4 CRC (Cyclic Redundancy Check) 転送 105
9.4.2 記述子サイズ 106
9.4.3 記述子設定 106
9.4.3.1 記述子制御 (DESCR_CTL) 107

目次

9.4.3.2 送信元アドレスと宛先アドレス (DESCR_SRC/DESCR_DST) . . . 108
9.4.3.3 記述子 X ループ制御 (DESCR_X_CTL) . . . 108
9.4.3.4 記述子 Y ループ制御 (DESCR_Y_CTL) . . . 108
9.4.3.5 記述子の次のポインタ (DESCR_NEXT_PTR) . . . 108
9.4.4 記述子転送サイズ . . . 108
9.4.5 記述子チェーン . . . 109
9.5 トリガー . . . 110
9.5.1 入力トリガー . . . 110
9.5.2 保留中のトリガー . . . 111
9.5.3 出力トリガー . . . 111
9.6 性能 . . . 112
9.6.1 シングル転送 . . . 112
9.6.2 1D 転送 . . . 112
9.6.3 2D 転送 . . . 113
9.6.4 CRC 転送 . . . 113
10 トリガー マルチプレクサ (トリガー MUX) . . . 114
10.1 特長 . . . 114
10.2 アーキテクチャ . . . 114
10.3 機能説明 . . . 115
10.4 ソフトウェア トリガー . . . 140
10.5 トリガー機能 . . . 140
10.6 レジスタの概要 . . . 142
11 座標回転デジタルコンピュータ (CORDIC) . . . 144
11.1 特長 . . . 144
11.2 アーキテクチャ . . . 145
11.2.1 ブロック図 . . . 145
11.3 機能説明 . . . 146
11.3.1 動作 . . . 146
11.3.2 正規化された結果データ . . . 147
11.3.3 動作モード . . . 148
11.3.4 収束領域 . . . 149
11.3.5 オーバーフローへの配慮 . . . 149
11.3.6 データフォーマット . . . 150
11.3.7 性能 . . . 150
11.3.8 ルックアップテーブル . . . 150
11.3.9 割込み生成 . . . 152
11.4 電源、クロック、リセット . . . 153
11.4.1 電源 . . . 153
11.4.2 クロック . . . 153
11.4.3 リセット . . . 153

目次

11.5 レジスタの概要 . . . 153
12 プログラムおよびデバッグ インターフェース . . . 155
12.1 特長 . . . 155
12.2 アーキテクチャ . . . 155
12.2.1 デバッグアクセスポート (DAP) . . . 156
12.2.1.1 DAP 電源ドメイン . . . 157
12.2.1.2 DAP セキュリティ . . . 157
12.2.2 ROM テーブル . . . 158
12.2.3 トレース . . . 158
12.2.4 エンベデッド クロストリガー . . . 158
12.3 シリアルワイヤデバッグ (SWD) インターフェース . . . 159
12.4 JTAG インターフェース . . . 159
12.5 MCU のプログラミング . . . 159
12.5.1 SWD ポートの開通 . . . 160
12.5.2 SWD プログラム モードへの移行 . . . 160
12.5.3 SWD プログラム ルーチンの実施 . . . 160
12.6 PSOC™ Control C3 MCU SWD デバッグ インターフェース . . . 160
12.6.1 デバッグ制御およびコンフィギュレーション レジスタ . . . 161
12.6.2 ブレークポイントユニット (BPU) . . . 161
12.6.3 データ ウォッチポイント (DWT) . . . 161
12.6.4 PSOC™ Control C3 デバイスのデバッグ . . . 162
12.7 レジスタの概要 . . . 162
13 暗号化アクセラレータ . . . 163
13.1 特長 . . . 163
13.2 アーキテクチャ . . . 163
13.2.1 ランダムクロック . . . 164
13.2.2 ハードウェア/ソフトウェア インターフェース . . . 164
13.3 ブロック機能 . . . 165
13.3.1 AES-128 . . . 165
13.3.2 SHA-256 . . . 167
13.3.3 ベクトル ユニット (VU) . . . 168
13.3.4 真性乱数生成器 (TRNG) . . . 171
13.3.4.1 固定リング発振器: RO11, RO15, GARO15, FIRO15 . . . 173
13.3.4.2 リング発振器: GARO31 . . . 173
13.3.4.3 リング発振器: FIRO31 . . . 174
13.3.4.4 TRNG 結果 . . . 175
13.3.4.5 TRNG ヘルス モニター . . . 175
13.4 割込み . . . 177
13.4.1 TRNG 割込み . . . 177
13.4.2 エラー割込み . . . 178

目次

13.5 レジスタの概要 . . . 178

14 TrustZone . . . 180
14.1 特長 . . . 180
14.2 アーキテクチャ . . . 180
14.3 メモリ領域 . . . 182
14.4 属性ユニット . . . 183
14.4.1 実装定義属性ユニット (IDAU) . . . 183
14.4.2 セキュア属性ユニット (SAU) . . . 184
14.5 状態の遷移 . . . 186
14.5.1 セキュアから非セキュアへの状態遷移 . . . 186
14.5.2 非セキュアからセキュアへの状態遷移 . . . 186
14.5.3 例外または割込みからの状態遷移 . . . 186

15 保護ユニット . . . 188
15.1 アーキテクチャ . . . 189
15.2 保護コンテキスト . . . 192
15.2.1 特殊な保護コンテキスト . . . 193
15.3 マスター保護属性 . . . 194
15.3.1 マスター セキュリティ コントローラー (MSC) . . . 194
15.4 スレーブ保護属性 . . . 194
15.4.1 メモリ保護ユニット (MPU) . . . 194
15.4.2 メモリ保護コントローラー (MPC) . . . 194
15.4.2.1 ソフトウェアビュー . . . 196
15.4.2.2 MPC MMIO レジスタ保護 . . . 197
15.4.3 ペリフェラル保護コントローラー (PPC) . . . 197
15.4.4 PPC アクセス制御ルール . . . 198
15.4.4.1 セキュリティ アウェア ペリフェラル . . . 200
15.4.4.2 PPC 領域 . . . 200

16 セキュア ブート . . . 210
16.1 基本的な定義 . . . 210
16.2 機能 . . . 211
16.3 アーキテクチャ . . . 211
16.3.1 ポリシーと鍵 . . . 211
16.3.1.1 ポリシー . . . 212
16.3.1.2 鍵 . . . 212
16.3.2 ライフサイクルステージ (LCS) . . . 212
16.3.2.1 IFX_MFG . . . 213
16.3.2.2 開発 LCS . . . 213
16.3.2.3 プロダクション LCS . . . 213
16.3.2.4 RMA . . . 214
16.3.3 ブートシーケンス . . . 214

目次

16.3.3.1 セキュア ブート ... 216
16.3.3.2 保護状態 ... 217
16.3.4 他のシリアル インターフェース ... 217
16.3.5 セキュアなプロビジョニングと再プロビジョニング ... 218
16.3.6 セキュア ブートで使用される保護コンテキスト ... 218
17 電源と電圧監視 ... 219
17.1 機能 ... 219
17.2 アーキテクチャ ... 220
17.3 電源 ... 221
17.3.1 アクティブ レギュレーター ... 221
17.3.2 ディープスリープ レギュレーター ... 221
17.3.3 リテンション レギュレーター ... 222
17.3.4 バックアップドメイン ... 222
17.4 SRSS リファレンスブロック ... 222
17.5 電圧監視 ... 222
17.5.1 パワーオンリセット (POR) ... 222
17.5.2 ブラウンアウト検出 (BOD) ... 222
17.5.3 低電圧検出 (LVD) ... 222
17.6 スリープ制御 ... 223
17.7 レジスタの概要 ... 223
18 デバイス電力モード ... 224
18.1 特長 ... 224
18.2 アーキテクチャ ... 224
18.2.1 電力モード ... 225
18.2.1.1 アクティブとスリープ ... 226
18.2.1.2 ディープスリープ ... 227
18.2.1.3 ディープスリープ RAM ... 227
18.2.1.4 ディープスリープ OFF ... 228
18.2.1.5 ハイバネート ... 228
18.2.2 その他の動作状態 ... 228
18.2.2.1 リセット状態 (XRES) ... 229
18.2.2.2 オフ状態 (OFF) ... 229
18.3 電力モードの遷移 ... 229
18.4 まとめ ... 231
18.5 レジスタの概要 ... 232
19 クロック システム ... 234
19.1 特長 ... 234
19.2 アーキテクチャ ... 234
19.3 クロックソース ... 236
19.3.1 内部メイン発振器 (IMO) ... 236

目次

19.3.2 内部高周波発振器 (IHO) . . . 236
19.3.3 内部低速発振器 (ILO) . . . 236
19.3.4 外部クロック (EXT_CLK) . . . 236
19.3.5 外部水晶発振器 (ECO) . . . 236
19.3.6 時計用水晶発振器 (WCO) . . . 237
19.3.7 ECO プリスケーラ . . . 237
19.4 クロック生成 . . . 237
19.4.1 デジタル位相同期回路 (DPLL) . . . 237
19.4.2 周波数同期回路 (FLL) . . . 240
19.5 クロックツリー . . . 242
19.5.1 クロックパス . . . 243
19.5.2 高周波ルートクロック (CLK_HF) . . . 243
19.5.3 低周波クロック (CLK_LF) . . . 245
19.5.4 バックアップクロック (CLK_BAK) . . . 246
19.5.5 タイマークロック (CLK_TIMER) . . . 246
19.5.6 電源クロック (CLK_PWR) . . . 246
19.6 CLK_HF 分配 . . . 247
19.7 ペリフェラルクロック分周器 . . . 248
19.7.1 分数クロック分周器 . . . 248
19.7.2 ペリフェラル クロック分周器のコンフィギュレーション . . . 248
19.7.3 ペリフェラルクロック分周器グループ . . . 250
19.8 クロック キャリブレーションカウンター . . . 253
19.9 クロック監視機能 (CSV: Clock Supervisor) . . . 254
19.10 レジスタの概要 . . . 256

20 ウォッチドッグ タイマー . . . 259
20.1 機能 . . . 259
20.2 アーキテクチャ . . . 259
20.3 ベーシック ウォッチドッグ タイマー (WDT) . . . 260
20.3.1 概要 . . . 260
20.3.2 ウォッチドッグリセット . . . 262
20.3.3 ウォッチドッグ割込み . . . 263
20.4 マルチカウンター WDT . . . 264
20.4.1 概要 . . . 264
20.4.1.1 MCWDT0_WDT0 および MCWDT0_WDT1 カウンターの動作 . . . 264
20.4.1.2 MCWDT0_WDT2 カウンター動作 . . . 267
20.4.2 MCWDT の有効化と無効化 . . . 269
20.4.3 ウォッチドッグ カスケード オプション . . . 270
20.4.4 MCWDT リセット . . . 273
20.4.5 MCWDT 割込み . . . 273
20.5 リセット原因検出 . . . 274
20.6 レジスタの概要 . . . 274

目次

21 バックアップ . . . 276
21.1 機能 . . . 276
21.2 アーキテクチャ . . . 277
21.3 電源 . . . 277
21.4 クロック . . . 278
21.4.1 外部クロック/正弦波入力による WCO . . . 279
21.4.2 キャリブレーション . . . 279
21.5 リセット . . . 279
21.6 リアルタイム クロック . . . 280
21.6.1 RTC ユーザー レジスタの読み出し . . . 280
21.6.2 RTC ユーザー レジスタへの書き込み . . . 281
21.7 アラーム機能 . . . 281
21.8 バックアップレジスタ . . . 283
21.9 レジスタの概要 . . . 283

22 リセット システム . . . 285
22.1 機能 . . . 285
22.2 リセットソース . . . 285
22.2.1 ブラウンアウトリセット . . . 287
22.2.2 WDT/MCWDT リセット . . . 287
22.2.3 ソフトウェア リセット . . . 287
22.2.4 外部リセット . . . 288
22.2.5 デバッガリセット . . . 288
22.2.6 フォールト検出リセット . . . 288
22.2.7 **高周波クロックエラーリセット** . . . 288
22.2.8 基準クロックエラーリセット . . . 288
22.2.9 ハイバネート ウェイクアップ リセット . . . 288
22.3 リセット ソースの識別 . . . 289
22.4 レジスタの概要 . . . 290

23 I/O サブシステム . . . 291
23.1 特長 . . . 291
23.2 アーキテクチャ . . . 292
23.2.1 I/O セル アーキテクチャ . . . 292
23.2.2 GPIO に代替機能をマッピングする HSIO . . . 293
23.2.3 電源投入時の I/O 状態 . . . 297
23.2.4 低消費電力モードでの動作 . . . 297
23.3 機能説明 . . . 298
23.3.1 デジタル入力バッファ . . . 298
23.3.2 デジタル出力ドライバ . . . 299
23.3.2.1 スルーレート制御 . . . 299
23.3.2.2 ドライブ強度 . . . 300

目次

23.3.2.3 ドライブモード 300
23.3.3 ファームウェアで制御される GPIO 304
23.3.4 割込み 304
23.3.5 アナログ I/O 307
23.3.6 セキュア GPIO 307
23.4 プログラマブル デジタル (スマート I/O) 307
23.4.1 概要 307
23.4.2 ブロック コンポーネント 308
23.4.2.1 クロック (Clock) とリセット (Reset) 309
23.4.2.2 同期化 311
23.4.2.3 ルックアップ テーブル (LUT) 311
23.4.2.4 データユニット (DU) 315
23.4.3 ルーティング 320
23.4.4 動作 320
23.5 レジスタの概要 321

24 タイマー/カウンター/パルス幅変調器 (TCPWM) 324
24.1 特長 324
24.2 アーキテクチャ 325
24.2.1 TCPWM ブロックにおけるカウンターの有効/無効 326
24.2.2 クロック 326
24.2.2.1 クロック プリスケーリング 326
24.2.2.2 カウントイベント 326
24.2.3 トリガー入力 326
24.2.4 グリッチフィルタリング 330
24.2.5 複数のカウンターの同期 331
24.2.6 トリガー出力 332
24.2.7 内部イベント 333
24.2.7.1 アンダーフロー イベント 333
24.2.7.2 オーバーフローイベント 333
24.2.7.3 ターミナル カウントイベント 334
24.2.7.4 cc0_match (cc1_match) イベント 334
24.2.8 割込み 335
24.2.9 デバッグ モード 337
24.2.10 PWM 出力 337
24.2.11 電力モード 339
24.3 カウンターの動作モード 340
24.3.1 タイマーモード 341
24.3.1.1 タイマーモードのカウンターの設定方法 349
24.3.2 キャプチャモード 349
24.3.2.1 キャプチャモードのカウンターの設定方法 353
24.3.3 パルス幅変調 (PWM) モード 354

目次

24.3.3.1 PWM モード機能 . . . 360
24.3.3.2 PWM モード用カウンターの設定方法 . . . 381
24.3.4 デッドタイム付きパルス幅変調 (PWM_DT) モード . . . 382
24.3.4.1 デッドタイムモード付き PWM 用カウンターの設定方法 . . . 385
24.4 ディザリング . . . 386
24.5 モーションインターフェース (MOTIF) . . . 387
24.5.1 概要 . . . 387
24.5.1.1 特長 . . . 388
24.5.1.2 ブロック図 . . . 388
24.5.2 ホールセンサー制御 . . . 389
24.5.3 直交デコーダ制御 . . . 399
24.5.3.1 直交クロックと方向デコード . . . 403
24.5.3.2 インデックス制御 . . . 404
24.5.4 スタンドアロン マルチチャネル モード . . . 405
24.5.5 同期スタート . . . 407
24.5.6 MOTIF の使い方 . . . 407
24.6 高分解能 PWM (HRPWM) . . . 415
24.6.1 概要 . . . 415
24.6.1.1 機能 . . . 415
24.6.1.2 ブロック図 . . . 415
24.6.2 機能説明 . . . 416
24.7 リカバリー . . . 423
24.8 初期化 . . . 424
24.9 ピンの状態 . . . 424
24.10 レジスタの概要 . . . 424
25 コントローラー エリア ネットワーク フレキシブル データ レート (CAN FD) . . . 427
25.1 概要 . . . 427
25.1.1 特長 . . . 427
25.1.2 サポートされていない機能 . . . 428
25.2 アーキテクチャ . . . 428
25.2.1 ブロック図 . . . 428
25.2.2 デュアルクロックソース . . . 428
25.2.3 割込みライン . . . 428
25.3 機能説明 . . . 429
25.3.1 動作モード . . . 429
25.3.1.1 ソフトウェアの初期化 . . . 429
25.3.1.2 通常動作 . . . 430
25.3.1.3 CAN FD 動作 . . . 430
25.3.1.4 伝搬遅延補償 . . . 431
25.3.1.4.1 説明 . . . 432
25.3.1.4.2 伝搬遅延補償測定 . . . 432

目次

25.3.1.5 制限動作モード 433
25.3.1.6 バス モニタリング モード 433
25.3.1.7 自動再送信の無効化 434
25.3.1.8 電源オフ (スリープモード) 434
25.3.1.9 テスト モード 435
25.3.1.9.1 外部ループバックモード 435
25.3.1.9.2 内部ループバックモード 435
25.3.1.10 アプリケーション ウォッチドッグ 436
25.3.2 タイムスタンプ生成 436
25.3.3 タイムアウトカウンター 437
25.3.4 受信処理 437
25.3.4.1 受入フィルタリング 437
25.3.4.2 受信 FIFO 440
25.3.4.2.1 RX FIFO ブロッキング モード 441
25.3.4.2.2 RX FIFO 上書きモード 442
25.3.4.2.3 RX FIFO トップポインタ 442
25.3.4.3 専用 RX バッファ 443
25.3.4.4 CAN サポートのデバッグ 444
25.3.4.4.1 デバッグメッセージのフィルタリング 444
25.3.4.4.2 デバッグ メッセージの処理 444
25.3.5 送信処理 445
25.3.5.1 送信ポーズ (一時停止) 446
25.3.5.2 専用 TX バッファ 446
25.3.5.2.1 専用 TX バッファのアドレス指定 446
25.3.5.3 送信 FIFO 447
25.3.5.4 TX キュー 447
25.3.5.5 混合専用 TX バッファ/TX FIFO 448
25.3.5.6 混合専用 TX バッファ/TX キュー 448
25.3.5.7 送信キャンセル 449
25.3.5.8 送信イベントの処理 449
25.3.6 FIFO アクノリッジ処理 450
25.3.7 CAN ビットタイミングの設定 450
25.3.7.1 CAN ビットタイミング 450
25.3.7.2 CAN ビット レート 452
25.4 メッセージ RAM (MRAM) 454
25.4.1 メッセージ RAM の設定 454
25.4.2 RX バッファと FIFO エレメント 455
25.4.3 TX バッファ エレメント 458
25.4.4 TX イベント FIFO エレメント 461
25.4.5 標準メッセージ ID フィルター エレメント 463
25.4.6 拡張メッセージ ID フィルター エレメント 465

目次

25.4.7 トリガー メモリ エレメント 467
25.4.8 RAM ウォッチドッグ (RWD) 469
25.5 タイマートリガー CAN 469
25.5.1 参照メッセージ 469
25.5.2 TTCAN コンフィギュレーション 470
25.5.2.1 TTCAN タイミング 470
25.5.2.2 メッセージのスケジューリング 472
25.5.2.3 トリガー メモリ 473
25.5.2.3.1 トリガーの種類 473
25.5.2.3.2 ノードのトリガーリストの制限事項 474
25.5.2.3.3 トリガー処理の例 475
25.5.2.3.4 コンフィギュレーション エラーの検出 476
25.5.2.4 TTCAN スケジュールの初期化 476
25.5.2.4.1 タイムスレーブ 477
25.5.2.4.2 ポテンシャル タイム マスター 477
25.5.3 TTCAN ギャップ制御 477
25.5.4 ストップウォッチ 478
25.5.5 ローカルタイム、サイクルタイム、グローバルタイム、外部クロック同期 478
25.5.6 TTCAN エラーレベル 481
25.5.7 TTCAN メッセージ処理 482
25.5.7.1 参照メッセージ 483
25.5.7.2 メッセージ受信 483
25.5.7.3 メッセージ送信 483
25.5.7.3.1 排他的タイムウィンドウでの送信 484
25.5.7.3.2 タイムウィンドウのアービトレーションにおける送信 484
25.5.7.3.3 マージされたアービトレーション タイムウィンドウでの送信 485
25.5.8 TTCAN 割込みとエラー処理 485
25.5.9 レベル 0 485
25.5.9.1 同期 486
25.5.9.2 エラー レベルの処理 487
25.5.9.3 マスタースレーブ関係 487
25.5.10 外部タイムスケジュールとの同期 488
25.6 設定手順 489
25.6.1 一般的なプログラムのフロー 489
25.6.2 クロック停止要求 490
25.6.3 MRAM オフ動作 491
25.6.4 MRAM オン動作 492
25.6.5 M_TTCAN チャネル固有の手順 492
25.6.6 CAN バス設定 494
25.6.7 メッセージ RAM のコンフィギュレーション 494
25.6.8 割込みコンフィギュレーション 497

目次

25.6.9 フレーム コンフィギュレーションの送信 . . . 498
25.6.10 割込み処理 . . . 499
25.7 レジスタの概要 . . . 506
26 シリアル通信ブロック (SCB) . . . 507
26.1 機能 . . . 507
26.2 アーキテクチャ . . . 507
26.2.1 トリガー インターフェース . . . 507
26.2.2 クロック モード . . . 508
26.2.3 Buffer Mode (バッファ モード) . . . 509
26.2.3.1 FIFO モード . . . 509
26.2.3.2 EZ モード . . . 509
26.2.3.3 CMD RESP モード . . . 509
26.2.4 ブロック操作 . . . 510
26.2.4.1 ブロック有効化とリカバリー . . . 510
26.2.4.2 リセット後の初期化フロー . . . 510
26.2.5 電力モード . . . 510
26.3 シリアル ペリフェラル インターフェース (SPI) . . . 511
26.3.1 機能 . . . 511
26.3.2 概要 . . . 511
26.3.3 SPI 動作モード . . . 513
26.3.3.1 Motorola 社製 SPI . . . 513
26.3.3.2 Texas Instruments 社製 SPI . . . 515
26.3.3.3 National Semiconductor 社製 SPI . . . 517
26.3.4 SPI バッファモード . . . 518
26.3.4.1 FIFO モード . . . 519
26.3.4.2 EZSPI モード . . . 519
26.3.4.3 コマンド レスポンス モード . . . 522
26.3.5 クロッキングとオーバーサンプリング . . . 524
26.3.5.1 クロック モード . . . 524
26.3.5.2 SPI マスターを使用してスレーブにクロックを供給する . . . 525
26.3.5.3 オーバーサンプリングとビットレート . . . 526
26.3.6 SPI マスター SELECT 出力タイミング制御 . . . 527
26.3.7 SPI パリティ機能 . . . 530
26.3.8 ループバック . . . 530
26.3.9 SPI の有効化と初期化 . . . 530
26.3.10 高速 SPI マスター/スレーブ サポート . . . 531
26.3.11 I/O パッド接続 . . . 531
26.3.11.1 SPI マスター . . . 531
26.3.11.2 SPI スレーブ . . . 532
26.3.12 レジスタの概要 . . . 534
26.4 UART . . . 534

目次

26.4.1 機能 . . . 534
26.4.2 概要 . . . 535
26.4.3 UART 動作モード . . . 536
26.4.3.1 標準プロトコル . . . 536
26.4.3.1.1 パリティ . . . 537
26.4.3.1.2 スタートスキップ . . . 538
26.4.3.1.3 半二重モード . . . 539
26.4.3.1.4 ブレーク検出 . . . 540
26.4.3.1.5 UART フロー制御 . . . 541
26.4.3.1.6 マルチプロセッサモード . . . 541
26.4.3.2 スマートカード (ISO 7816) . . . 543
26.4.3.3 IrDA . . . 544
26.4.3.4 UART ローカル インターコネクト ネットワーク (LIN) モード . . . 545
26.4.4 クロッキングとオーバーサンプリング . . . 548
26.4.5 UART の有効化と初期化 . . . 548
26.4.6 I/O パッド接続 . . . 549
26.4.6.1 UART モード . . . 549
26.4.6.1.1 標準 UART モード . . . 549
26.4.6.1.2 半二重モード . . . 550
26.4.6.2 スマートカードモード . . . 551
26.4.6.3 IrDA モード . . . 551
26.4.6.4 LIN モード . . . 552
26.4.7 レジスタの概要 . . . 553
26.5 インター インテグレーテッド サーキット (I^2C) . . . 553
26.5.1 機能 . . . 553
26.5.2 概要 . . . 553
26.5.3 用語および定義 . . . 554
26.5.3.1 クロック ストレッチ . . . 555
26.5.3.2 バスアービトレーション . . . 555
26.5.4 I2C 動作モード . . . 555
26.5.4.1 書き込み転送 . . . 556
26.5.4.2 読み出し転送 . . . 557
26.5.5 I2C バッファモード . . . 557
26.5.5.1 FIFO モード . . . 557
26.5.5.2 EZI2C モード . . . 559
26.5.5.3 コマンド レスポンス モード . . . 561
26.5.6 クロッキングとオーバーサンプリング . . . 562
26.5.6.1 グリッチフィルタリング . . . 564
26.5.6.2 オーバーサンプリングとビットレート . . . 565
26.5.7 ループバック . . . 566
26.5.8 I2C の有効化と初期化 . . . 566

目次

26.5.8.1 I2C FIFO モードの設定 567
26.5.8.2 EZ モードと CMD_RESP モードの設定 567
26.5.9 I/O パッド接続 567
26.5.10 レジスタの概要 568
26.6 SCB 割込み 569
26.6.1 SPI 割込み 569
26.6.2 UART 割込み 570
26.6.3 I2C 割込み 572
27 高性能プログラマブル アナログ サブシステム (HPPASS) 574
27.1 特長 574
27.2 アーキテクチャ 576
27.2.1 インフラストラクチャー (INFRA) 577
27.2.1.1 起動 578
27.2.1.2 入力トリガー 581
27.2.2 SAR ADC 582
27.2.2.1 チャネル 583
27.2.2.2 シーケンサー 585
27.2.2.3 マルチプレクサのルーティング制御 587
27.2.2.4 トリガー、サンプリング、変換の優先順位の図解 588
27.2.2.5 エラー状態 591
27.2.2.6 後処理 591
27.2.2.7 結果ハンドリング 594
27.2.2.8 SAR ADC 割込み 594
27.2.2.9 キャリブレーション 595
27.2.3 コンパレーターおよびスロープジェネレーター (CSG) 596
27.2.3.1 10 ビット DAC 597
27.2.3.2 DAC モード 599
27.2.3.3 DAC モードスタート/ストップ 603
27.2.3.4 DAC アップデートトリガー 603
27.2.3.5 後処理 605
27.2.3.6 CSG 割込み 607
27.2.4 自律型コントローラー (AC) 607
27.2.4.1 概要 607
27.2.4.2 ブロックインターフェース 608
27.2.4.3 状態遷移表 608
27.2.4.4 初期化 612
27.2.4.5 例 612
27.2.5 温度センサー 615
27.3 割込みおよびトリガー出力 617
27.4 クロック 618
27.5 レジスタの概要 619

目次

28 低消費電力コンパレーター (LPCOMP) 624
28.1 特長 624
28.2 アーキテクチャ 624
28.2.1 入力コンフィギュレーション 625
28.2.2 出力と割込みの設定 625
28.2.3 電力モードおよび動作速度のコンフィギュレーション 627
28.2.4 ヒステリシス 628
28.2.5 低消費電力モードからウェイクアップ 629
28.2.6 コンパレーター クロック 629
28.2.7 低消費電力コンパレーターの使用 629
28.3 レジスタの概要 630

改訂履歴 631

免責事項 633

1 はじめに

本資料は、産業用アプリケーション向けに設計されたインフィニオンのマイクロコントローラー PSOC™ Control C3 シリーズに関する情報を提供します。これらのマイクロコントローラーは、Arm® Cortex®-M33 CPU と統合されたプログラマブル ペリフェラルを備えています。特にリアルタイム アプリケーション向けに設計され、先進の 40 nm プロセスで製造されています。さらに、さまざまな低消費電力モード、セキュリティ機能、柔軟なクロック システム、豊富な通信ペリフェラル、高度なデジタルおよびアナログ ペリフェラル、業界標準の開発ツールとの互換性を備えています。

1.1 詳細な特長

このデバイスには次のような特長があります。

- CPU サブシステム
  - 最大動作周波数 180 MHz の Arm® Cortex®-M33
  - デジタル シグナル プロセッサ (DSP)、浮動小数点演算ユニット (FPU)、メモリ保護ユニット (MPU)、16 KB I キャッシュ
  - 各 16 チャネルの 2 つのダイレクト メモリ アクセス (DMA) コントローラー
  - セキュリティ
    - プラットフォーム セキュリティ アーキテクチャ レベル 2 (PSA L2) 認定
    - ユーザーコードに制御を渡すまで、実行イメージを段階的に認証
    - 保護されたルーチンの実行専用モードにおけるコードのセキュアな実行
    - イメージ認証と完全性チェック
    - 信頼証明とソフトウェア管理のために、分離されたデバイスのルート オブ トラスト (RoT) を確立する TrustZone フレームワーク
- メモリ
  - ECC を備えたオンチップ フラッシュ
    - 最大 256 KB フラッシュ (RWW 機能付き)、64 KB ROM (ブートコードおよびブートローダ機能用)
    - シリアル インターフェース (UART/I2C/SPI) を介したブート ROM の内蔵デバイス ファームウェア アップグレード (DFU) に対応
  - ECC を備えた SRAM
    - 64 KB ディープスリープ時に利用可能な SRAM
      - SRAM データ パスは、ソフト エラー検出と訂正用のハードウェア メカニズム (ECC) で保護されています。
- クロッキング サブシステム
  - ディープスリープ動作で精度±2%の 8 MHz IMO
  - 48 MHz 内部高周波発振器 (IHO)、±1%の精度
  - 32 kHz 内蔵低周波発振器 (ILO)、±10%の精度
  - 4～35 MHz の外部水晶発振器 (ECO) を使用可能
  - リアルタイム クロック (RTC) に使用可能な 32.768 kHz の外部時計用水晶発振器 (WCO)
  - 外部クロック (EXTCLK)最大周波数: 80 MHz
  - 周波数ロック ループ (FLL) ×1、出力範囲 24～100 MHz
  - 2 つのデジタル位相同期ループ、DPLL#0 と DPLL#1、出力範囲は 25～250 MHz
- 1.71 V～3.6 V で動作
  - 6 つの電力モード (アクティブ、スリープ、ディープスリープ、ディープスリープ-RAM、ディープスリープ-OFF、ハイバネート状態) できめ細かな電力管理を実現

  - 64 KB の SRAM リテンション、LPComp、ディープスリープ SCB を備えた内部電圧レギュレーターを使用し、3.3 V の外部電源で 11 µA のディープスリープモード電流を実現
  - RTC と LPComp を使用した場合のハイバネートモード電流は最大 1000 nA
- 通信周辺
  - シリアル通信ブロック (SCB)
    - 最大 6 つの独立したランタイム再設定可能な SCB。各 SCB はソフトウェアで I2C、SPI、または UART としてマスターまたはスレーブ モードに設定可能
    - 1 つの SCB は、I2C スレーブモードと SPI スレーブモードでのディープスリープ動作とディープスリープからのウェイクアップにも対応
    - 1 つの SCB が最大 50 MHz の高速 SPI に対応
    - SCB は UART の単線半二重モードに対応
  - コントローラー エリア ネットワーク (CAN FD)
    - 単一インスタンスで最大 2 つの CAN FD チャネル、最大 8 Mbps の動作が可能
- 高性能プログラマブル アナログ サブシステム (HPPASS)
  - アナログ-デジタル変換器 (ADC)
    - 12 ビット 12 Msps SAR ADC×1
    - 最大 16 の専用アナログパッド、最大 16 のパラレルサンプルステージに接続可能
    - さらに 2 つの GPIO をアナログ入力として使用可能
    - SAR ADC 内の最大 16 個のサンプル/ホールド (S/H) 回路をピンに直接または AMUX 経由で接続
    - SAR ADC 内の 1 つの S/H 回路はアナログ リファレンスと温度センサーに内部接続されています
    - 16 個の S/H 回路すべてに、1、3、6、12 の入力ゲインを設定可能
    - プログラムされた境界値と ADC の結果を比較するためのデジタル コンパレーター出力
    - デジタルコンパレーター出力はタイマー/カウンター/パルス幅変調器 (TCPWM) に接続可能 (モジュール間の低レイテンシ)
  - アナログ コンパレータ
    - 5 つのアクティブ コンパレーター (ディープスリープ機能なし)、各コンパレーターのリファレンスを生成する 10 ビット DAC 内蔵
    - 各コンパレーターは、ピンをとおして外部リファレンス/スレッショルドに対応
    - アクティブ コンパレーターは、ヒステリシス モードで内蔵 DAC と共に使用可能
    - LPComp の 2 つの追加コンパレーターは、アクティブ/ディープスリープ/ハイバネート モードで利用可能
    - コンパレーター出力は、制御ループ アプリケーション用にピンに接続可能
    - コンパレーター出力は TCPWM に接続可能 (モジュール間の低レイテンシ)
    - トリガー マルチプレクサを介して TCPWM への入力トリガーとして接続された複数のコンパレーター トリガー出力の論理和
- リアルタイム制御ペリフェラル
  - 座標回転デジタルコンピュータ (CORDIC)
    - 円関数 (三角関数)、双曲線関数を解くためのすべての CORDIC 動作モードに対応し、計算を高速化する独立ルックアップテーブルを内蔵
  - タイマー/カウンター/パルス幅変調器 (TCPWM)
    - 16 ビット TCPWM チャネル×16
    - 4 つの 32 ビット TCPWM チャネル PWM 出力用の高分解能 PWM 生成 (HRPWM) をサポートします。
    - 中央揃え、エッジ、疑似ランダム モード
    - コンパレーター ベースのキル シグナルのトリガー

    - デューティ、周期、デッドタイム、出力信号極性、ディザリングのシャドウ更新 (擬似ランダムモード)
    - マルチチャネル コントロール: 8 つの TCPWM チャネルのグループでは、グループ内の 1 つのチャネルが他のチャネルをトリガー可能
    - スマート I/O による複数チャネルの出力を論理的に組み合わせる機能
    - PWM チャネルを HPPASS のトリガーおよび/またはゲート信号として柔軟に使用できるよう、グループ内の専用出力トリガーマルチプレクサを用意
    - 自律 BLDC ブロック整流に対応したホールセンサー インターフェース
    - モーター速度とローター位置をデコードするための直交エンコーダーインターフェース
    - 周期、デューティ、デッドタイム挿入のための HRPWM 機能 (標準分解能 80 ps 以下)
- I/O サブシステム
  - プログラマブル GPIO ピン
    - 最大 66 機能ピン (50 デジタル GPIO、50 GPIO のうち 2 つはアナログ入力に使用可能＋ 16 本の専用アナログ専用入力)
    - プログラム可能な駆動モード、駆動強度およびスルー レート
  - プログラマブル デジタル
    - 最大 7 のスマート I/O 対応ポート (I/O、56 の LUT) により、I/O 信号のブール演算が可能
- 暗号技術
  - 暗号化アクセラレータ
    - ベクター ユニット (VU) とハッシュ関数 (SHA-256) がサポートする対称暗号 (AES-128) と非対称暗号アルゴリズム (RSA と楕円曲線暗号 (ECC)) のハードウェア アクセラレーション
    - 真性乱数生成器 (TRNG)

1.2 アーキテクチャ

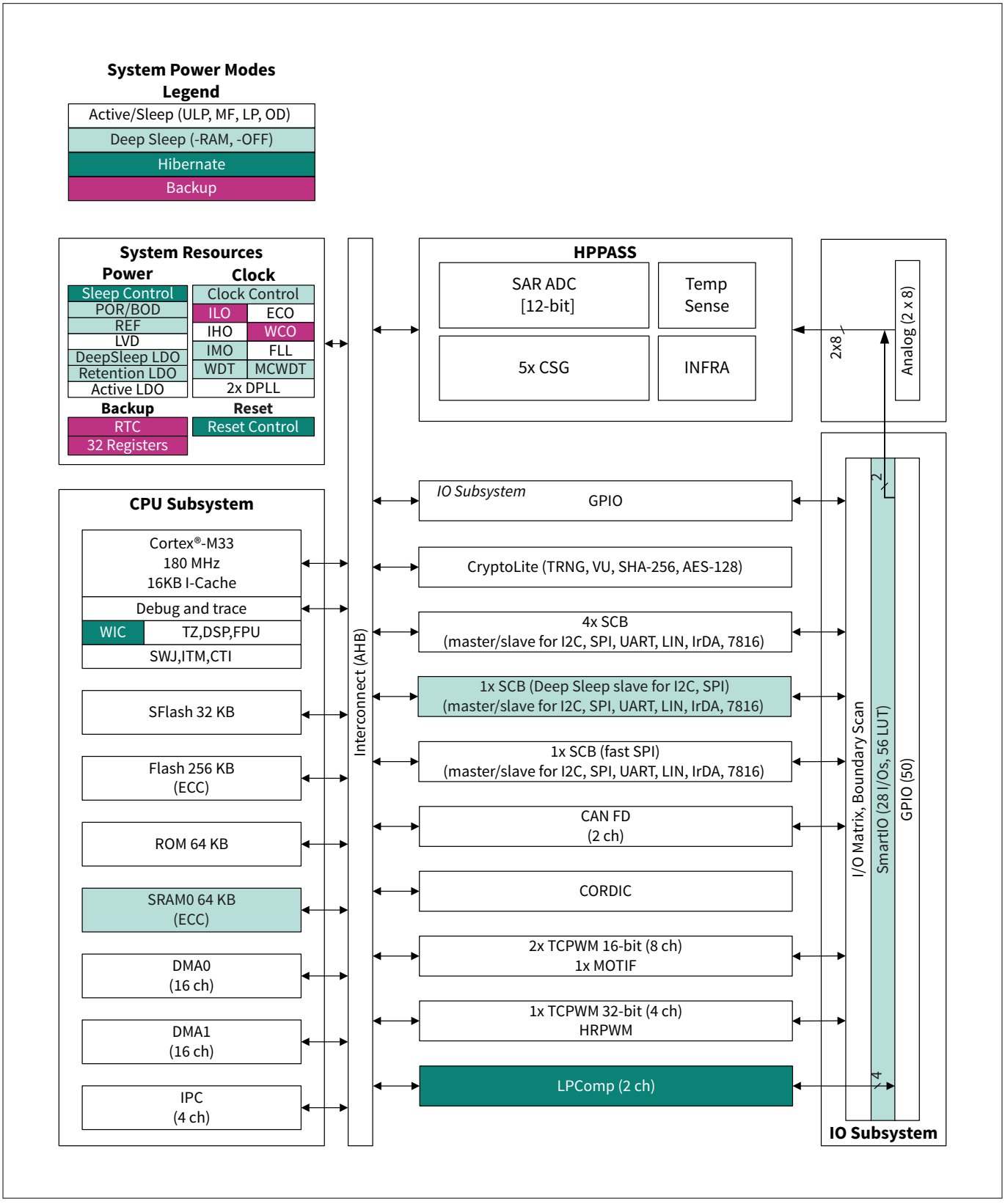


図 1 機能ブロック図

1.3 文書の構成と表記法

1.3.1 主要セクション

このリファレンスマニュアルの情報は以下のように構成されています。

- 章: セクショントピックに特有の事項を個々に説明します。搭載と使用に関する詳細な情報です。
- レジスタの概要: その章に関連するレジスタの概要。
- レジスタ テクニカル リファレンス マニュアル: テクニカル リファレンス マニュアルでまとめられるデバイスのレジスタの詳細な情報を提供します。これは追加資料です。

1.3.2 本書の表記法

1.3.2.1 レジスタの表記法

レジスタの規則については、『レジスタ リファレンス マニュアル』に詳述されています。

1.3.2.2 数値の表記法

16 進数はすべて大文字で表記し、小文字の‘h’を付記しています (例えば「14h」「3Ah」)。C 言語のコーディング規則に基づき、接頭語「0x」を使用して 16 進数を表現している場合もあります。2 進数には小文字の‘b’を付記しています (例:「01010100b」、「01000011b」)。‘h’も‘b’も付いていない数は 10 進数です。

1.3.2.3 測定単位

表 1 に本書で使用する測定単位を示します。

表 1 測定単位

略語	測定単位
bps	ビット/秒
°C	摂氏温度
dB	デシベル
dBm	ディービーエム (1 mW 基準デシベル表現電力比)
fF	フェムト ファラッド (静電容量)
G	ギガ
Hz	ヘルツ
k	キロ、1000
K	キロ, 2^10
KB	1024 バイトまたは 1000 バイト
Kbit	1024 ビット
kHz	キロヘルツ (32.000)
kΩ	キロオーム

(続く)

表 1　　(続き) 測定単位

略語	測定単位
MHz	メガヘルツ
MΩ	メガオーム
µA	マイクロ アンペア
µF	マイクロ ファラド
µs	マイクロ秒
µV	マイクロボルト
µVrms	二乗平均平方根 (Root-Mean-Square) 電圧
mA	ミリアンペア
ms	ミリ秒
mV	ミリボルト
nA	ナノアンペア
ns	ナノ秒
nV	ナノボルト
Ω	オーム
pF	ピコファラド
pp	ピークピーク値
ppm	100 万分の 1
SPS	サンプル数毎秒
σ	シグマ: 標準偏差値を 1 単位とした表記
V	ボルト

1.3.2.4　　略語

表 2 は頭字語とその定義の一覧です。

表 2　　本書で使用する略語

略語	定義
AC	自律型コントローラー
ACK	アクノリッジ
ADC	アナログ-デジタル変換器
AES	Advanced Encryption Standard (高度暗号化標準)
AF	アナログ フィルター
AHB	AMBA 高性能バス、Arm®データ転送バス
AIRCR	アプリケーション割込みおよびリセット制御レジスタ

(続く)

表 2 (続き) 本書で使用する略語

略語	定義
AMBA	アドバンスド マイクロコントローラー バス アーキテクチャ
ARB	アービトレーション
BGR	バンドギャップ基準
BOD	電圧低下検出
CAN	コントローラー エリア ネットワーク
CAN-FD	Controller Area Network with Flexible Data Rate
CC0	Capture/Compare0
CC1	Capture/Compare1
CM-33	Cortex® M33
CORDIC	座標回転デジタルコンピューター
CPU	中央演算処理装置
CPUSS	CPU サブシステム
CSG	コンパレーターとスロープジェネレーター
CSV	クロック監視
CTS	クリアして送信
DAC	デジタル-アナログ コンバーター
DAP	デバッグ アクセスポート
DF	デジタルフィルター
DMA	ダイレクト メモリアクセス
DMAC	ダイレクト メモリアクセス コントローラー (DMAC)
DPLL	デジタル位相同期回路
DSI	デジタル システム インターコネクト
DSP	デジタル信号処理
DU	データユニット
DW	データ線
ECC	Error correcting code (エラー訂正符号)
ECO	外部水晶発振器
Em_EEPROM	エミュレートされた電気的消去書き込み可能な読み出し専用メモリ
EOC	変換の終了
EPPB	外部プライベート ペリフェラル バス
FIFO	first in first out (先入れ先出し)
FIR	有限インパルス応答

(続く)

表 2 (続き) 本書で使用する略語

略語	定義
FLL	周波数同期回路
FM	フラッシュ マクロ
FPU	浮動小数点ユニット
FSM	有限状態機械 (FSM)
GPIO	汎用入出力
HPPASS	高性能プログラマブル アナログ サブシステム
HRPWM	高分解能 PWM
HSIOM	高速 I/O マトリックス
I2C	インター インテグレーテッド サーキット
I-Cache	命令キャッシュ
ICER	割込み有効クリア レジスタ
IDAU	実装定義属性ユニット
IHO	内部高速発振器
ILO	内部低速発振器
IMO	内部主発振器
INFRA	インフラストラクチャー
IOSS	I/O サブシステム
IPC	プロセッサ間通信
IRQ	割込み要求
ISER	割込み有効セット レジスタ
ISR	Interrupt service routine (割込みサービスルーチン)
ITNS	割込みターゲット非セキュア
LCS	ライフサイクルステージ
LDO	低ドロップアウト
LFCLK	低周波クロック
LIN	ローカル相互接続ネットワーク
LP	低消費電力
LPCOMP	低消費電力コンパレータ
LUT	ルックアップテーブル
LVD	低電圧検出
MCU	マイクロコントローラー ユニット
MCWDT	マルチカウンター ウォッチドッグ タイマー
MF	中間周波数

(続く)

表 2 (続き) 本書で使用する略語

略語	定義
MISO	マスターインスレーブアウト
MMIO	メモリマップ I/O
MOSI	マスターアウトスレーブイン
MOTIF	モーション インターフェース
MPC	メモリ保護コントローラー
MPS	マグニチュード プリスケーラー
MPU	メモリ保護ユニット
MSC	マスター セキュリティ コントローラー
MSP	メイン スタック ポインタ
MTQ	最小タイムクォンタム
NACK	ノー アクノリッジ
NMI	ノンマスカブル割込み
NVIC	ネスト型ベクタ割込みコントローラー
NVM	不揮発性メモリ
OD	オーバードライブ
OTP	ワンタイムプログラマブル
OV	オーバーフロー
PC	保護コンテキスト
PDL	ペリフェラル ドライバ ライブラリ
PendSV	システムサービスへの保留可能な要求
PERI	ペリフェラル インターコネクト
PID	保護 ID
PLL	位相同期回路
POR	パワーオン リセット
PPB	プライベート ペリフェラル バス
PPC	ペリフェラル保護コントローラー
PPU	電源ポリシー ユニット
PSP	プロセス スタック ポインタ
PSR	プログラム ステータス レジスタ
PWM	パルス幅変調
PWMDT	パルス幅変調デッドタイム有り
RAM	ランダム アクセスメモリ
RMA	返品承認

(続く)

表 2 (続き) 本書で使用する略語

略語	定義
RTC	リアルタイムクロック
RTS	送信要求
RX	Receive (受信側)
SAR	逐次比較レジスタ
SAU	セキュア属性ユニット
SCB	シリアル コミュニケーション ブロック
SCL	シリアル クロック
SDA	シリアル データ
SFR	特殊機能レジスタ
SHA	セキュア ハッシュ アルゴリズム
SoC	システムオンチップ
SPI	シリアル ペリフェラル インターフェース
SRAM	スタティック ランダム アクセス メモリ
SRAMC	スタティック ランダム アクセス メモリ コントローラー
SRSS	システム リソース サブ システム
SVCall	スーパバイザーコール
SWD	シングル ワイヤ デバッグ
TC	ターミナル カウント
TCPWM	タイマー/カウンター/パルス幅変調器
TQ	タイムクォンタム
TRNG	真性乱数発生器
TX	Transmit (送信側)
UART	ユニバーサル非同期送受信器
UART_MP	UART マルチプロセッサ
ULP	超低消費電力
UN	アンダーフロー
VTOR	ベクタ テーブル オフセット レジスタ
VU	ベクトル単位
WCO	時計用水晶発振回路
WDT	ウォッチドッグ タイマー
WFE	イベントの待機
WFI	Wait for interrupt (割込み待ち命令)
WIC	ウェイクアップ割込みコントローラー

(続く)

表 2 (続き) 本書で使用する略語

略語	定義
WS	待機状態
XRES	外部リセット

2 CPU サブシステム (CPUSS)

CPU サブシステムは、32 ビットの Arm® Cortex®-M33 (CM33) CPU をベースにしています。セキュリティ、制御、通信、短い割込み応答時間、高いコード密度、高い 32 ビット スループットを実現する一方で、厳しいコストと消費電力のバジェットを維持するように設計されています。

ここでは、PSOC™ Control C3 MCU の Arm® Cortex® CPU の概要についてのみ説明します。詳細については、Cortex®-M33 の Arm®ドキュメントセットを参照してください。

2.1 特長

PSOC™ Control C3 MCU Arm® Cortex®-M33 (CM33) CPU には、次のような特長があります。

- 最大クロック周波数 180 MHz (オーバードライブ (OD) モード時)
- 単精度浮動小数点ユニット (FPU) と DSP 拡張機能
- 各セキュリティ状態 (セキュアと非セキュア) ごとに最大 8 つの領域を持つメモリ保護ユニット (MPU)
- CPU コアと統合されたネステッド ベクタード割込みコントローラー (NVIC) により、割込みレイテンシを低減
- ディープスリープ割込みによって CPU をディープスリープ電力モードからウェイクアップさせるウェイクアップ割込みコントローラー (WIC)
- 16 KB 命令キャッシュ (I キャッシュ)、Cortex®-M33 C-AHB バス、4 ウェイ セットアソシアティブ
- プログラマブルなセキュア帰属ユニット (SAU) と固定実装定義帰属ユニット (IDAU) に基づくセキュア/非セキュア エイリアシングの TrustZone-M メモリ マッピングに対応

2.2 アーキテクチャ

CM33 は、32 ビット データ パス、32 ビット レジスタ、32 ビット メモリ インターフェースを備えた 32 ビット プロセッサです。CPU は、Thumb 命令セットのさまざまな命令に対応しています。CM33 CPU はスレッドとハンドラの動作モードに対応しています (動作モードと特権レベルセクションを参照)。さらに、CM33 CPU は、セキュリティ拡張機能に基づいて、セキュアと非セキュアのセキュリティ状態に対応します。

CM33 命令セットには以下が含まれます。

- 符号付きおよび符号なし、32×32→32 ビットおよび 32×32→64 ビット、積和演算、全シングルサイクル
- 2～11 サイクル (値による) の符号付きおよび符号なし 32 ビット除算
- DSP 命令
- 複雑なメモリ ロードとストア アクセス
- 複雑なビット操作

FPU 付き CM33 プロセッサは、FPv5 アーキテクチャで定義された単精度演算に対応しています。これは ANSI/IEEE Std 754-2008 (バイナリ浮動小数点演算の IEEE 標準) に準拠しています。

2.2.1 CPUSS

PSOC™ Control C3 CPUSS は、Cortex®-M33 (CM33) コア上に構築され、最大 180 MHz で動作するセキュリティ、制御、通信用 CPU です。

Arm®CM33 CPU は、Arm® v8-M アーキテクチャをベースに開発された 3 段パイプライン (3PIP) で、高速クロック切り替えが可能な低消費電力ドメインを備えています。CPUSS は Arm® TrustZone-M 互換のセキュリティと保護インフラストラクチャーを実装しています。セキュリティと保護スキームの初期起動を提供します。

- CM33 CPU はセキュリティ拡張機能を実装した TrustZone-M CPU です。セキュリティ拡張の一環として、CM33 は CPU 内部にセキュア アトリビューション ユニット (SAU) を実装しています。TrustZone-M アーキテク

チャは、CPU への外部インターフェースも定義しており、実装定義アトリビューション ユニット (IDAU) に接続できます。

- CPUSS の非 CPU マスターの場合、マスター セキュリティ コントローラー (MSC) が実装定義帰属ユニット (IDAU) とともにインスタンス化され、セキュア/非セキュアの転送を区別します。詳細については、センサー シミュレーションのセクションを参照してください。

CPUSS は、C-AHB バス上には、16 KB の 4 ウェイセット アソシアティブ命令キャッシュがあります。CPUSS は、オンチップ SRAM コントローラーとメモリに対応しています。SRAM から C-AHB バスアクセスを提供しコードを実行します。CPUSS は、命令フェッチと FLASH (NVM) からのデータ アクセス用に 128 ビット AHB5 バス インターフェース、およびエイリアス アドレスでアクセス可能な内蔵 SRAM を備えています。また、システム メモリ マップに存在するメモリ マップド フラッシュ、内部 SRAM にアクセスするための 32 ビット AHB5 バス インターフェースもサポートしています。

CPUSS には、Arm® Q チャネルをサポートする 16 チャネルに対応する 2 つの DMA コントローラーが搭載されています。これらのチャネルは、主に SCB、CORDIC、TCPWM ペリフェラルのサービスに使用されます。CPUSS には、セキュア アプリケーションと非セキュア アプリケーションが通信し、活動を同期させるための 4 つのプロセッサ間通信 (IPC) ハードウェアが搭載されています。

CPUSS は、大半のハードウェア モジュール用に clk_hf0 クロックドメインを持っています。

- clk_hf0: 最大 180 MHz @ 1.2 V。I キャッシュ、AHB バス インフラストラクチャー、DMA、MSC/IDAU/MPC、および単一の同期周波数で動作するすべてのオンチップ メモリ コントローラーなど、CPUSS の大部分で使用されます。

CPUSS は集中型フォールト レポート インフラストラクチャーを実装しており (単一のフォールト割込みハンドラのみが必要)、MPC/PPC 違反、ペリフェラル固有のエラー、ECC エラーなどのメモリコントローラー固有のエラー、タイムアウトエラーに関連するフォールトを捕捉します (すべてを含むものではありません)。

CPUSS では、アクティブ電源ドメインが最初にアクティブまたは初期化されます (デフォルトでは、パワーオン リセット (POR) に続きます)。詳細については、「概要説明」のセクションを参照してください。

2.2.2 アドレスマップとメモリマップ

CPU には固定アドレスマップがあります。CPU のデフォルト メモリ マップは、0x00000000～0xFFFFFFFF の範囲をカバーします。デフォルトのメモリ マップについては、Arm® Cortex®-M33 プロセッサ テクニカル リファレンス マニュアルのメモリ モデルのセクションを参照してください。この 32 ビット (4GB) のアドレス空間は、表 3 に示すように、さらに非セキュアとセキュアのメモリ領域に分割されます。アドレスの Arm® TrustZone ビット[28] で、メモリがセキュアか非セキュアかを定義します。

表 3 PSOC™ Control C3 MCU アドレスとメモリマップ

アクセス タイプ	アドレス範囲	サブ領域名	バンクモード	サイズ	領域名 - インターフェース
非セキュア	0x22000000 – 0x2203FFFF	フラッシュ (NVM)	シングル バンク	256 KB	SRAM 領域 - S バス
セキュア	0x32000000～0x3203FFFF			256 KB	
非セキュア	0x02000000～0x0203FFFF		シングル バンク	256 KB	コード領域 - C バス
セキュア	0x12000000～0x1203FFFF			256 KB	
非セキュア	0x22000000～0x2201FFFF		デュアル バンク (バンク 0)	128 KB	SRAM 領域 - S バス

(続く)

表 3 (続き) PSOC™ Control C3 MCU アドレスとメモリマップ

アクセス タイプ	アドレス範囲	サブ領域名	バンクモード	サイズ	領域名 - インターフェース
	0x22800000～0x2281FFFF		デュアルバンク (バンク 1)	128 KB	
セキュア	0x32000000～0x3201FFFF		デュアルバンク (バンク 0)	128 KB	
	0x32800000～0x3281FFFF		デュアルバンク (バンク 1)	128 KB	
非セキュア	0x02000000～0x0201FFFF		デュアルバンク (バンク 0)	128 KB	コード領域 - C バス
	0x02800000～0x0281FFFF		デュアルバンク (バンク 1)	128 KB	
セキュア	0x12000000～0x1201FFFF		デュアルバンク (バンク 0)	128 KB	
	0x12800000～0x1281FFFF		デュアルバンク (バンク 1)	128 KB	
非セキュア	0x04000000 – 0x0400FFFF	SRAM	該当なし	64 KB	コード領域 - C バス
セキュア	0x14000000 – 0x1400FFFF		該当なし	64 KB	
非セキュア	0x24000000～0x2400FFFF		該当なし	64 KB	SRAM 領域 - S バス
セキュア	0x34000000～0x3400FFFF		該当なし	64 KB	
非セキュア	0x23400000～0x23407FFF	SFLASH	該当なし	32 KB	SRAM 領域 - S バス
セキュア	0x33400000～0x33407FFF		該当なし	32 KB	SRAM 領域 - S バス
非セキュア	0x03400000～0x03407FFF		該当なし	32 KB	コード領域 - C バス
セキュア	0x13400000～0x13407FFF		該当なし	32 KB	コード領域 - C バス
非セキュア	0x00800000 – 0x0080FFFF	予約済み	該当なし	64 KB	ROM 領域
セキュア	0x10800000 – 0x1080FFFF		該当なし	64 KB	
非セキュア	0x42000000～0x43FFFFFF	SYSCPUSS ペリフェラル (SYS MMIO)	該当なし	32 MB	ペリフェラル領域
セキュア	0x52000000～0x53FFFFFF		該当なし	32 MB	

注: *セキュアメモリと非セキュアメモリのペアは同じ物理メモリですが、セキュアか非セキュアかによってマッピングが異なります。*

アドレス空間のギャップは予約されています。これらのギャップにはアクセスしないでください。アクセスした場合、アドレスがどのバスセグメントまたはペリフェラルに割り当てられているかによって、ハードフォールトまたは BUS ERROR になることがあります。PC (保護コンテキスト) 切り替え時のキャッシュのコヒーレンシーは、すべてソフトウェアによって維持されます。CM33 では、フラッシュのアクセス可能なアドレス空間はすべてキャッシュ可能です。

0X4000_0000/0x5000_0000 で指定されるすべてのペリフェラル アドレス空間はキャッシュ不可能です。このアドレス空間からの XIP 実行命令は MemFault を引き起こします。アドレス エイリアシングは、CM33 プロセッサがコード バス経由で SRAM にアクセスする場合にのみ適用され、SRAM メモリからのコード実行を容易にします。

注: *CM33_S_VECTOR_TABLE_BASE、CM33_NS_VECTOR_TABLE_BASE レジスタは、CM33 CPU のセキュア ベクタテーブルと非セキュア ベクタ テーブルの位置を決定します。*

2.2.2.1 待機状態のルックアップテーブル

待機状態のルックアップテーブルは、異なるメモリモジュールの待機状態を示します。

表 4 メモリ待機状態

パラメーター	記号	値			注記またはテスト条件
		Min	Typ	Max	
メモリ待機状態					
待機状態	$M33_{cache}$	-	-	0	
CM33 からの待機状態	$M33_{SRAM}$	-	0	3	ECC 有効時は最大 3
C バス経由でフラッシュからの待機状態 (キャッシュされます)	$M33_{Flash}$	3*	-	9	キャッシュヒット、プリフェッチ、オペコードに依存
S バス経由でフラッシュからの待機状態	$M33_{Flash}$	-	-	9	

注: *待機状態のルックアップテーブルは、すべての電力モード (ULP/MF/LP/OD) に適用可能です。*

注: *[*] キャッシュ ヒット時に $M33_{Flash}$ の最小値が 0 になります。*

2.3 動作モードと特権レベル

CM33 CPU は、セキュア状態と非セキュア状態の両方に対応しています。CPU にはスレッドとハンドラの動作モードがあり、Thumb またはデバッグのいずれかの動作状態で実行できます。さらにプロセッサは、特権モードまたは非特権モードでコードを実行することで、一部のリソースへのアクセスを制限または除外できます。

動作モードと実行モードの詳細については、Arm® v8-M アーキテクチャ リファレンス マニュアルを参照してください。

2.3.1 セキュリティ状態

Arm® v8-M セキュリティ拡張機能がプロセッサに搭載されている場合、プログラマモデルには、セキュアと非セキュアの 2 つの直交するセキュリティ状態が実装されています。CM33 CPU ではセキュリティ拡張機能が有効になっており、常にセキュア状態にリセットされます。

システム制御空間のレジスタは、セキュア状態と非セキュア状態にまたがってバンクされ、非セキュア レジスタビューは、セキュア状態へのエイリアス アドレスで利用できます。

2.3.2 動作モード

各セキュリティ状態に対して、プロセッサはスレッドモードかハンドラモードで動作できます。

- **スレッドモード**: アプリケーション ソフトウェアの実行に使用されます。プロセッサは、リセット時にスレッドモードに入るか、例外の結果としてスレッドモードに戻ります。スレッドモードには特権コードと非特権コードがあります。
- **ハンドラモード**: 例外処理に使用されます。プロセッサは、例外の結果としてハンドラ モードに入ります。ハンドラモードには特権コードのみが含まれます。

プロセッサは、例外が発生するとセキュリティ状態を変更できます。例えば、非セキュア状態からセキュアな例外が発生すると、スレッドモードはセキュアな状態のハンドラモードに入ります。プロセッサはまた、非セキュア状態からセキュアな機能を呼び出したり、セキュア状態から非セキュアな機能を呼び出したりもできます。セキュリティ拡張機能には、非セキュア状態でセキュアなデータにアクセスされないようにするための要件があります。

2.3.3 動作状態

プロセッサは、Thumb 状態またはデバッグ状態で動作できます。

- Thumb 状態とは、16 ビットおよび 32 ビットのハーフワード配列の Thumb 命令を実行する通常の実行状態です。
- デバッグ状態とは、プロセッサが halting debug にあるときの状態です。

2.3.4 特権レベル

- **非特権:** ソフトウェアは、汎用レジスタから特殊レジスタへの移動 (MSR) 命令および特殊レジスタから汎用レジスタへの移動 (MRS) 命令へのアクセスが制限され、プロセッサ状態変更/割込み禁止 (CPSID) 命令およびプロセッサ状態変更/割込み許可 (CPSIE) 命令を使用できません。システムタイマー、NVIC、システムコントロールブロックにはアクセスできません。メモリやペリフェラルへはアクセス制限されています。
- **特権:** ソフトウェアは、すべての命令を実行でき、すべてのリソースにアクセスが可能です。

スレッド モードでは、CONTROL レジスタが、ソフトウェアの実行が特権か非特権かを制御します。ハンドラモードでは、ソフトウェアの実行は常に特権的です。特権ソフトウェアのみが、CONTROL レジスタに書き込んで特権レベルを変更できます。非特権ソフトウェアは、SVC 命令を使って特権ソフトウェアに制御を移せます。

ハンドラモードでは、常に MSP が使用されます。例外エントリーと復帰の仕組みは自動的に CONTROL レジスタを更新します。このレジスタは、MSP レジスタと PSP レジスタのどちらを使用するかによって変化することがあります。

スレッド モードでは、MSR 命令を使用して、CONTROL レジスタのスタック ポインタ ビットを設定します。スタックポインタを変更する場合は、MSR 命令の直後に ISB 命令を使用します。これで ISB 以後の命令に新しいスタックポインタが確保されます。

2.4 命令セット

CM33 CPU は、Arm®v8-M アーキテクチャで定義された 16 ビットの Thumb 命令にすべて対応しています。さらに、このプロセッサは、Thumb-2 テクノロジーに基づく T32 命令セットを実装しており、高いコード密度とプログラム メモリ要件の削減を実現しています。

CM33 の CPU は Arm®v8-M の以下の命令を実装しています。

- すべての基本命令
- メイン拡張機能のすべての命令
- セキュリティ拡張機能が有効な場合、セキュリティ拡張機能内のすべての命令
- DSP 拡張機能のすべての命令 (オプション)
- FPU が有効な場合、浮動小数点拡張のすべての単精度命令と倍精度ロードおよびストア命令

これらの命令の詳細については、Arm® Cortex®- M33 デバイス ジェネリック ユーザー ガイドのセクション 3 - Cortex®-M33 命令セットを参照してください。

プロセッサはカスタム データパス拡張 (CDE) 命令も実装しています。CDE は、コプロセッサ命令空間に 2×3 クラスの命令を導入します。

- 3 つのクラスが汎用レジスタ ファイルを操作
- 3 つのクラスが浮動小数点レジスタ ファイルを操作

プロセッサに実装されている CDE 命令の具体的な情報については、Arm® Cortex®-M33 プロセッサ RM のセクション 9. Arm® custom instructions を参照してください。

2.5 レジスタの概要

CM33 CPU には、以下の 32 ビットコアレジスタがあります。セキュリティ エクステンションが含まれている場合、一部のレジスタはバンクされます。これらのレジスタのセキュア ビューは、CPU がセキュア状態にあるときに利用でき、非セキュア ビューは CPU が非セキュア状態にあるときに利用できます。

表 5 CM33 コアレジスタセットの概要

名称	種類 [1]	必要な特権 [2]	リセット値	説明
R0-R12	RW	どちらか	UNKNOWN	R0～R12 はデータ操作用の 32 ビット汎用レジスタ
MSP (R13)	RW	どちらか	-[3]	R13 はスタック ポインタ レジスタ (SP)。スレッド モードでは、CONTROL レジスタが主スタック ポインタ (MSP) またはプロセス スタック ポインタ (PSP)、どちらを使用するかを示します。 Arm®v8-M セキュリティ拡張が含まれている場合、Cortex®-M33 プロセッサには 2 つの MSP レジスタがあります。 • 非セキュア状態用 MSP_NS • セキュア状態用 MSP_S Arm®v8-M セキュリティ拡張が含まれている場合、Cortex®-M33 プロセッサには 2 つの PSP レジスタがあります。 • 非セキュア状態用 PSP_NS • セキュア状態用 PSP_S
PSP (R13)	RW	どちらか	UNKNOWN	

(続く)

表 5 (続き) CM33 コアレジスタセットの概要

名称	種類 [1]	必要な特権 [2]	リセット値	説明
MSPLIM	RW	特権	0x00000000	スタック リミット レジスタは、MSP レジスタと PSP レジスタがそれぞれ下降できる範囲を制限します。 Arm® v8-M セキュリティ拡張が含まれている場合、Cortex®-M33 プロセッサには 2 つの MSPLIM レジスタがあります。 • 非セキュア状態用 MSPLIM_NS • セキュア状態用 MSPLIM_S Arm® v8-M セキュリティ拡張が含まれている場合、Cortex®-M33 プロセッサには 2 つの PSPLIM レジスタがあります。 • 非セキュア状態用 PSPLIM_NS • セキュア状態用 PSPLIM_S
PSPLIM	RW	特権	0x00000000	
LR (R14)	RW	どちらか	UNKNOWN	リンク レジスタ (LR) はレジスタ R14。サブルーチン、関数呼び出しおよび例外の戻り情報を保存します。
PC (R15)	RW	どちらか	- [3]	プログラム カウンター (PC) はレジスタ R15。現在のプログラム アドレスを保持します。
xPSR	RW	どちらか	- [4]	プログラム ステータス レジスタ (PSR) は以下を含みます。 • アプリケーション プログラム ステータス レジスタ (APSR) • 割込みプログラム ステータス レジスタ (IPSR) • 実行プログラム ステータス レジスタ (EPSR) これらのレジスタは PSR の異なるビューを提供します。
APSR	RW	どちらか	UNKNOWN	APSR は直前の命令を実行後の条件フラグの状態を保持します。
IPSR	RO	特権	0x00000000	IPSR は現在の ISR の例外番号を保持します。
EPSR	RO	特権	- [4]	EPSR には、If-Then (IT) 命令の Thumb 状態のビットと実行ステート ビット、および割込みロード マルチプル命令または割込みストア マルチプル命令の割込み継続可能命令 (ICI) フィールドが含まれます。

(続く)

表 5 (続き) CM33 コアレジスタセットの概要

名称	種類 [1]	必要な特権 [2]	リセット値	説明
PRIMASK	RW	特権	0x00000000	PRIMASK レジスタは優先順位を設定することですべての例外が有効になることを防止します。 Arm®v8-M セキュリティ拡張が含まれる場合、Cortex®-M33 プロセッサには 2 つの PRIMASK レジスタがあります。 • 非セキュア状態用 PRIMASK_NS • セキュア状態用 PRIMASK_S
BASEPRI	RW	特権	0x00000000	BASEPRI レジスタは、例外処理の最小優先度を定義します。 Arm®v8-M セキュリティ拡張が含まれている場合、Cortex®-M33 プロセッサには 2 つの BASEPRI レジスタがあります。 • 非セキュア状態用 BASEPRI_NS • セキュアな状態用 BASEPRI_S
FAULTMASK	RW	特権	0x00000000	FAULTMASK レジスタは、NMI (NON-MASKABLE INTERRUPT) とオプションでセキュアな HardFault を除くすべての例外の発生を防ぎます。 Arm®v8-M セキュリティ拡張が含まれている場合、Cortex®-M33 プロセッサには 2 つの FAULTMASK レジスタがあります。 • 非セキュア状態用 FAULTMASK_NS • セキュア状態用 FAULTMASK_S
CONTROL	RW	特権	0x00000000	CONTROL レジスタは、プロセッサがスレッド モードにあるときに使用されるスタックと、オプションで特権レベルを制御します。 Arm®v8-M セキュリティ拡張が含まれている場合、Cortex®-M33 プロセッサには 2 つの CONTROL レジスタがあります。 • 非セキュア状態用 CONTROL_NS • セキュア状態用 CONTROL_S

1) スレッドモードおよびハンドラモードでプログラム実行中のアクセスの種類について説明します。デバッグ アクセスは異なることがあります。
2) どのエントリーも、特権ソフトウェアおよび非特権ソフトウェアの両方がレジスタにアクセスできることを意味します。
3) リセットハンドラによって、取得された値にソフトリセットします。
4) ビット[24] は T ビットで、リセットベクトルのビット[0] からロードされます。その他のビットはすべて 0 にリセットされます。

CM33 の浮動小数点ユニット (FPU) には以下のレジスタがあります。

- 32 ビットの単精度レジスタ S0～S31。これらのレジスタは、16 本の 64 ビット倍精度レジスタ (D0～D15) としてもアドレス指定できます。
- 4 つの FPU コントロール レジスタとステータス レジスタは以下のとおりです。
  - FPCCR - 浮動小数点コンテキスト制御レジスタ
  - FPCAR - 浮動小数点コンテキスト アドレス レジスタ
  - FPSCR - 浮動小数点状態制御レジスタ
  - FPDSCR - 浮動小数点デフォルト ステータス制御レジスタ

これらのレジスタがどのように使われるかについては、Arm® Cortex®-M33 リファレンス マニュアルを参照してください。

PSR、PRIMASK、CONTROL、FAULTMASK、BASEPRI レジスタにアクセスするためには、MSR 命令と MRS 命令を使用します。以下の表に PSR ビットの割り当て方法を示します。詳細については、Arm® Cortex®-M33 リファレンス マニュアルの複合プログラム ステータス レジスタのセクションを参照してください。

表 6　　CM33 APSR ビット割り当て

ビット	名称	使用法
31	N	負のフラグ
30	Z	ゼロフラグ
29	C	キャリーまたはボロー フラグ
28	V	オーバーフロー フラグ
27	Q	DSP オーバーフローおよび飽和フラグ
26:20	-	予約済み
19:16	GE[3:0]	大小フラグ
15:0	-	予約済み

表 7　　CM33 IPSR ビット割り当て

ビット	名称	使用法
31:9	-	予約済み

(続く)

表 7 (続き) CM33 IPSR ビット割り当て

ビット	名称	使用法
8:0	例外番号	これは現在の例外の数です。 • 0 = スレッド モード • 1 = Reset • 2 = NMI • 3 = HardFault • 4 = MemManage • 5 = BusFault • 6 = UsageFault • 7 = SecureFault • 8～10: 予約済み • 7～10: 予約済み • 11 = SVCall • 12 = DebugMonitor • 13=予約済み • 14 = PendSV • 15 =SysTick • 16 = IRQ0 • …495 = IRQ479

表 8 CM33 EPSR ビット割り当て

ビット	名称	使用法
31:27	-	予約済み、RES0
26:25, 15:10	ICI	割込み継続可能命令ビット
26:25, 15:10	IT	IT 命令の実行状態ビットを示します
26:25, 11:10, 15:12	ECI	ビート ワイズ ベクトル命令の例外継続フラグです。このフィールドは、機内命令のどのビートが完了したかを示します。
24	T	Thumb 状態のビット
23:16	-	

3 バス インフラストラクチャー

システムオンチップ (SoC) のバス インフラストラクチャーは、チップ内のさまざまなコンポーネント間の通信のバックボーンを形成します。これはプロセッサ、メモリ、ペリフェラル、アクセラレータなどの異なる IP ブロックを接続するバスネットワークです。インフラストラクチャーは、データおよび制御信号要に標準化されたインターフェースを提供し、これらのコンポーネント間のシームレスな相互作用とデータ交換を実現します。通常、システムバス、インターコネクト、ペリフェラルバスなど、さまざまな種類のバスで構成され、それぞれが特定の目的を果たしています。アドバンスド ハイパフォーマンス バス (AHB) などのシステム バスや相互接続は、プロセッサ、メモリ、その他の重要なコンポーネント間の広帯域幅通信を容易にします。

PSOC™ Control C3 MCU ファミリーは、複数のバス マスターとスレーブを相互接続する AHB5 ベースのバス インフラストラクチャーで構成される SYSCPUSS を備えています。バス インターコネクトは、ブリッジ、アップサイザー、ダウンサイザーを使ってバス幅の異なる他のインターコネクトと接続することで、複数のマスターからスレーブへのアクセスを可能にし、複数の高速ペリフェラルが同時に動作している場合でも、同時アクセスと効率的な動作を可能にします。図 2 に、この PSOC™ Control C3 デバイス用のバス インフラストラクチャーを示します。

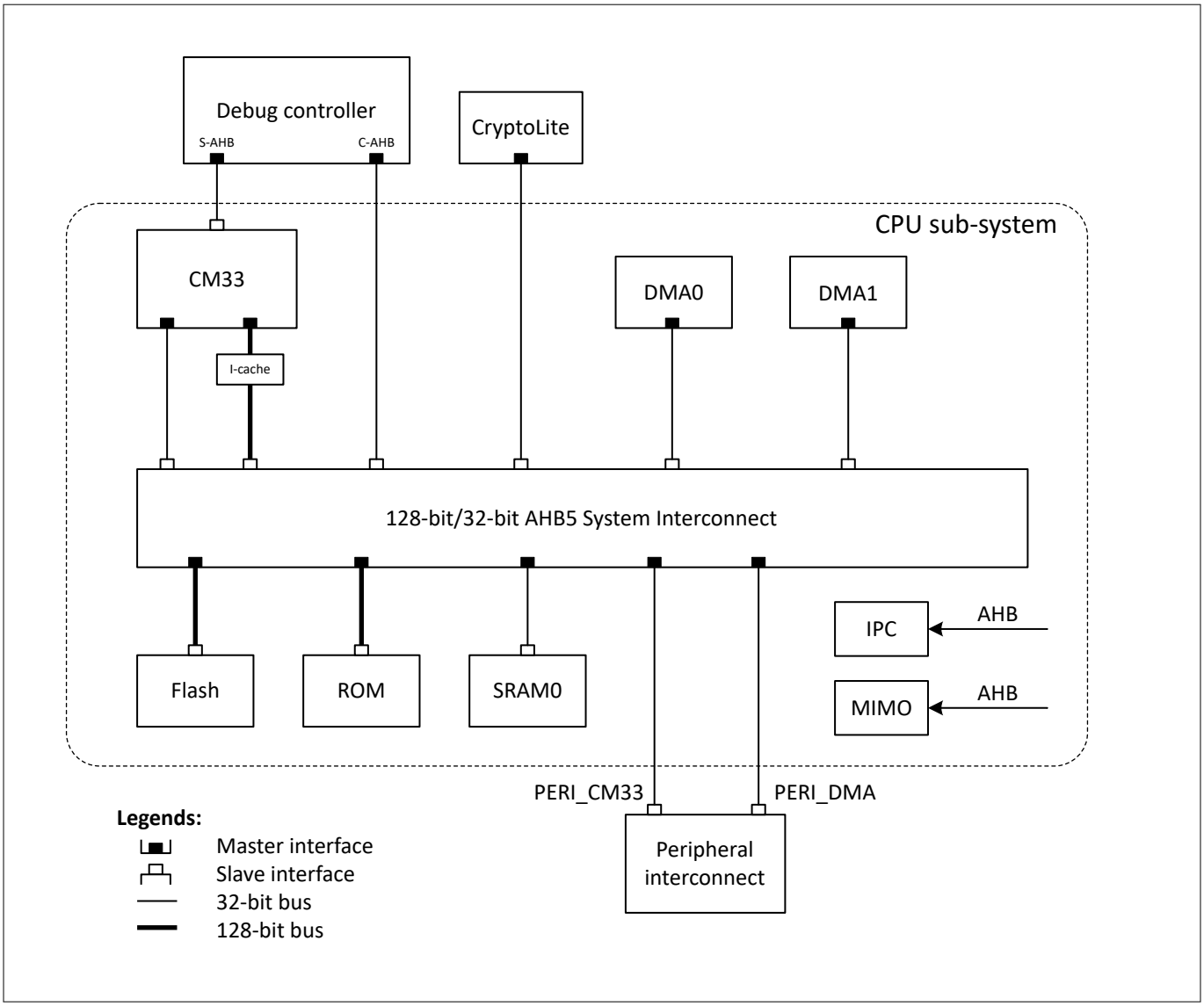


図 2 PSOC™ Control C3 デバイス用バス インフラストラクチャー

3.1 SYSCPUSS AHB5 インターコネクト

SYSCPUSS (CM33 CPU) は、128 ビットと 32 ビットのマルチレイヤー AHB5 バス マトリックス インターコネクトを使用しています。AHB5 バス マスターは、SYSCPUSS 内に存在することもあれば、SYSCPUSS の外部 (SYSCPUSS 外部バス マスター) に存在することもあります。同様に、AHB5 バスのスレーブは、SYSCPUSS の内部 (例えば、オンチップ メモリ コントローラー) に存在することもあれば、SYSCPUSS の外部 (例えば、ペリフェラル インターコネクト (PERI) または外部スレーブ インターフェースに接続されたスレーブ) に存在することもあります。

SYSCPUSS AHB5 バス マトリックスは、以下のマスターとスレーブを相互接続します。

マスター:

- Cortex®-M33 コード AHB (C-AHB) バス、命令キャッシュ (I-Cache) 経由
- Cortex®-M33 スレーブ AHB (S-AHB) バス
- 2 つのダイレクト メモリ アクセス - データワイヤ (DMA0、DMA1)
- 暗号化アクセラレータ (CryptoLite) ×1

スレーブ:

- SYSCPUSS 内部のフラッシュコントローラー
- SYSCPUSS (SRAM0) 内部の SRAM コントローラー×1
- SYSCPUSS 内部の ROM ×1
- CM33 からペリフェラルへの相互接続 (PERI_CM33)
- AHB DMA からペリフェラルへの相互接続 (PERI_DMA) 1 本

3.1.1 AHB5 バス マトリックス インターコネクト

SYSCPUSS は、レイテンシを伴うラウンドロビン方式を採用し、AHB5 インターコネクトのバス アービトレーションを実装しています。

SYSCPUSS AHB5 バス マトリックスは、以下のマスタとスレーブに相互接続します。

表 9 AHB5 バス マトリックス相互接続

バスマスター	**バススレーブ**				
	フラッシュ	ROM	SRAM0	PERI_CM33	PERI_DMA
CM33_C-AHB	**X**	**X**	x	-	
CM33_S-AHB	**x**	-	X	X	-
DMA0	**X**	-	X	-	X
DMA1	**X**	-	X	-	X
CryptoLite	**X**	-	X	-	X

ここで、

「**X**」= 接続、太字は 128 ビットバスを示します

「-」= 未接続

「x」= 接続、エイリアスされたアドレスパス

「X」= コネクション

「**x**」= 接続、エイリアスアドレスパス、太字は 128 ビットバスを示します

注:

- *メモリエイリアシングにより、CM33 の C-AHB バス (I-Cache) はシステム SRAM にアクセスできます。CM33 の S-AHB バスはエイリアスされたアドレス パスを使用し、フラッシュ (C-AHB バスのアドレス空間) に書き込みます。フラッシュエイリアスアドレスからデータを読み出すこともできますが、CM33 の I キャッシュにキャッシュされないため、好ましくありません。*
- *すべての CM33 C-AHB バス アクセスはキャッシュ可能です。*
- *S-AHB バス アドレス空間 (0x20000000 以上) を使用したエイリアス アドレスへの DMA 書き込み*
- *DMA または CM33 S-AHB バスからのエイリアスされたアドレスへの書き込みは、古いデータがキャッシュに残らないように、キャッシュ無効化シーケンスの後に行わなければいけません。*
- *PERI_CM33 および PERI_DMA インターフェースは、ペリフェラル インターコネクト (PERI) を介して接続されたペリフェラルへのアクセスを提供します。*

3.1.2 SYSCPU バス

3.1.2.1 Cortex®-M33 C-AHB バス

このバスは、Cortex®-M33 コアの C-AHB バスを、命令キャッシュを介してバス マトリックスに接続します。命令フェッチと、コード領域にマッピングされた内部および外部メモリへのデータ アクセスに使用されます。このバスのターゲットはフラッシュ、内蔵 SRAM、ROM です。

3.1.2.2 Cortex®-M33 S-AHB バス

このバスは Cortex®-M33 コアの S-AHB バスをバス マトリックスに接続します。このバスは、コアがペリフェラルや SRAM 領域にあるデータにアクセスするために使用されます。このバスのターゲットは、エイリアスされたフラッシュ、内蔵 SRAM、および PERI_CM33 を介して接続されたペリフェラルです。

3.1.2.3 DMA バス

このバスは、DMA の 32 ビット AHB マスター インターフェースにバス マトリックスを接続し、CM33 CPU とクロックを共有します。このバスのターゲットは、フラッシュ、内蔵 SRAM、および PERI_DMA を介して接続されたペリフェラルです。

3.2 AHB5 MMIO

CM33、CryptoLite、DMA0/1、CPUSS、およびその他すべてのペリフェラル コンフィギュレーション用のメモリ マップド I/O レジスタは、AHB5 MMIO バス インターフェースを介してアクセスできます。

3.3 ペリフェラル インターコネクト (PERI)

ペリフェラル インターコネクト (PERI) は、AHB5 バス マトリックス インフラストラクチャーを実装し、AHB マスターをペリフェラル デバイスのグループに接続します。これにより、SYSCPUSS の内部/外部マスターは、低消費電力ドメインで動作するペリフェラルとインターフェースできます。これらの周辺デバイスには、PERI、PCLK、外部周辺デバイスのすべてのメモリ マップド レジスタ (MMIO レジスタ)、および周辺アドレス空間にマップされた同期および非同期ペリフェラルが含まれます。

- 0x4200_0000～0x43FF_FFFF (非セキュアアドレスマップ)
- 0x5200_0000～0x53FF_FFFF (セキュアアドレスマップ)

PSOC™ Control C3 MCU ファミリーでは、PERI インスタンスは 1 つです。以下の表に示すように、PERI は 6 つのペリフェラルグループで構成され、対応するペリフェラルグループに存在するスレーブペリフェラルを SYSCPUSS や他の AHB マスターに接続します。

表 10　　PERI グループ

ペリフェラル グループ	スレーブ ペリフェラル	ペリフェラル タイプ	説明
PERI グループ 0	ペリフェラル インターコネクト MMIO レジスタ	内部	I/O サブシステムなしの SYSCPUSS プラットフォーム コンポーネント
	ペリフェラル保護コントロール (PPC)		
	PERI_PCLK レジスタ		
	APB スレーブグループ-0、SRAM0、PROM、フラッシュ、MXCM33-0、DW-0/1、SYSCPUSS、IPC、フォールト構造 MMIO レジスタ		
	SRSS コア	ペリフェラル	
	SRSS 電力モード		
	SRSS バックアップ		
	暗号アクセラレータ (CryptoLite)		
	座標回転デジタルコンピュータ (CORDIC)		
	デバッグ レジスタ		
PERI グループ 1	高速 I/O マトリックス (HSIOM)	ペリフェラル	非同期ペリフェラル
	GPIO		
	スマート I/O		
	低消費電力コンパレーター (LPComp)		
PERI グループ 2	DFT	ペリフェラル	
	eFuse		
PERI グループ 3	コントローラー エリア ネットワーク フレキシブル データレート (CAN FD)	ペリフェラル	非同期ペリフェラル
	シリアル通信ブロック (SPI/UART/I2C) (SCB[0])		
	シリアル通信ブロック (SPI/UART/I2C) (SCB[1:4])		
PERI グループ 4	タイマー/カウンター/PWM (TCPWM)	ペリフェラル	非同期ペリフェラル

(続く)

表 10 (続き) PERI グループ

ペリフェラル グループ	スレーブ ペリフェラル	ペリフェラル タイプ	説明
	高性能プログラマブル アナログ サブシステム (HPPASS)		
PERI グループ 5	シリアル通信ブロック (SPI/UART/I2C) (SCB[5])	ペリフェラル	高速 SPI

ペリフェラルグループ内のペリフェラルスレーブの AHB バスインターフェースは、すべて同じ周波数の「clk_group[i]」で動作します。消費電力を削減するために、SL_CTL[i].ENABLED[j]を使用してスレーブを無効にできます。

トリガー信号の多重化は、トリガーコンポーネントによって行われます。詳細については、トリガー マルチプレクサ (トリガー MUX) を参照してください。

3.3.1 AHB バスのエラー処理

バス インフラストラクチャーでは、以下のようなバス エラーが発生する可能性があります。

1. マスター インターフェースで検出された AHB エラー レスポンス
2. ペリフェラル デコーダーからの AHB エラーレスポンス (ペリフェラル デコーダーがいずれかのシナリオでエラーレスポンスを返します。SL_*レジスタを使用することにより、各スレーブへのアクセスが無効/リセット (これらのレジスタの詳細については、PSOC™ Control C3 MCU レジスタ リファレンス マニュアルを参照してください)、またはマッピングされていないスレーブ アドレスへのアクセス、または各ペリフェラル自体からのエラー応答)
3. 予想よりも多くのサイクルにおいて、ペリフェラルから AHB トランザクションに対して応答がありません。

PERI は、これらのエラーを報告するためにフォールト インターフェースを使用します。

3.4 バス マスター ペリフェラル

3.4.1 ダイレクトメモリ アクセス (DMA-DW)

DMA0/1 は、CPU サブシステムの AHB マスターとして使用できます。DMA-DW は、データサイズが小さく、トランザクション DMA に最適です。トランザクション DMA は通常、ADC から RAM など、ペリフェラル間でバイトを送信するために使用されます。DMA-DW は 32 ビット AHB バスを使用します。

3.4.2 CryptoLite

CryptoLite は、8/16/32 ビット AHB-Lite 転送をサポートする AHB-Lite マスター インターフェースを提供します。AHB-Lite マスター インターフェースは、サブシステムがシステム メモリ (例えばフラッシュや SRAM メモリ) からオペランド データにアクセスできるようにします。メモリ バッファ アクセスは 8/16/32 ビット アクセスが可能です。

4 プロセッサ間通信 (IPC)

プロセッサ間通信 (IPC) は、セキュア アプリケーションと非セキュア アプリケーションが通信を行い、同期するための機能を提供します。IPC ハードウェアは、2 つのレジスタ構造を使って実装されています。

- IPC チャネル: プロセッサ間の通信と同期は、この構造を使って行われます。
- IPC 割込み: 各割込み構造は、任意の IPC チャネルの「notify」または「release」イベントによってトリガーされる割込みラインを設定します。

図 3 に示すように、チャネルと割込みの構造は独立しており、互いに相関はありません。これにより、代表的な使用モデルで示したように、さまざまなインターフェースのモデルを構築できます。

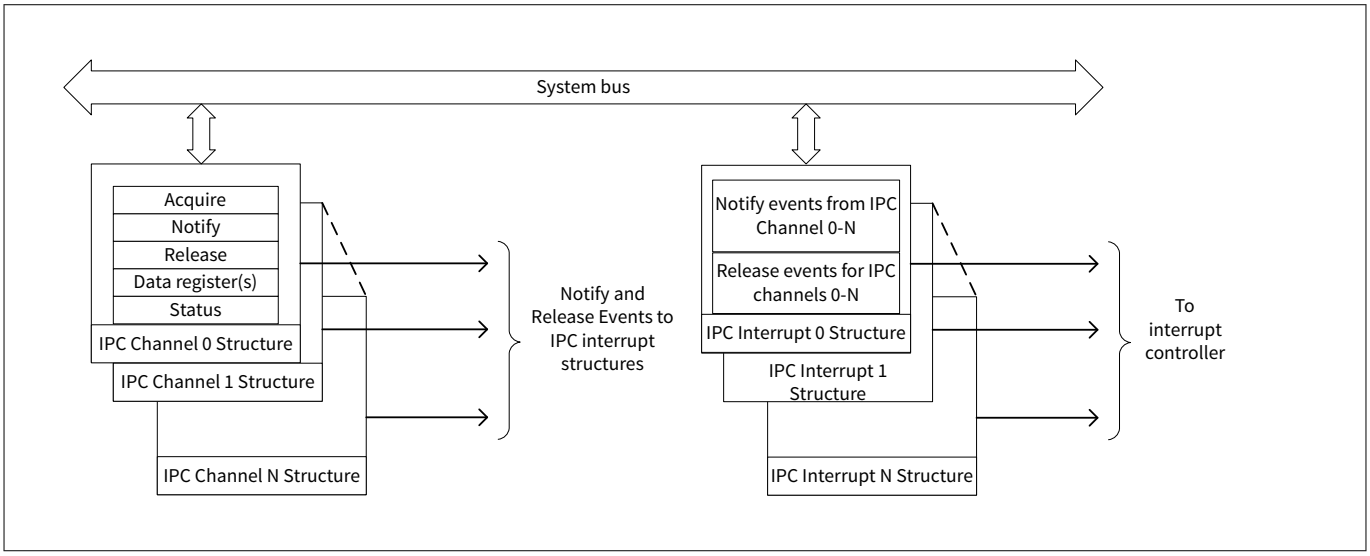


図 3 IPC レジスタ アーキテクチャ

4.1 特長

以下に IPC の特長を示します。

- セキュア アプリケーションと非セキュア アプリケーション間の相互排除と同期のためのロックの実装
- セキュア アプリケーションと非セキュア アプリケーションの間でメッセージの送信が可能
- 最大 16 の通信用チャネルに対応
- 最大 16 の割込みに対応し、チャネルからの notify または release イベントを使用してトリガー可能

4.2 アーキテクチャ

4.2.1 IPC チャネル

IPC チャネルは、図 4 に示すように、6 つのハードウェア レジスタとして実装されます。IPC チャネル レジスタは、システム内のセキュア アプリケーションと非セキュア アプリケーションの両方からアクセス可能です。

- IPC_STRUCTx_ACQUIRE: このレジスタは IPC のロック機能を決定します。IPC チャネルのロックは、このレジスタの読み出しによって取得できます。SUCCESS フィールドが値‘1’を返す場合、読み出しがロックを獲得したことを意味します。SUCCESS フィールドが値'0'を返す場合、読み出し操作がロックの獲得に失敗したことを意味します。P、NS、PC、MS フィールドは、以前にロックを成功裏に取得したトランザクションのアクセス属性を表し、現在のアクセスはこれらのフィールドに影響を与えません。

1 回の読み出しアクセスが 2 つの機能を果たすことに注意してください。

- ロックの獲得の試行
- 獲得試行の結果を返す (SUCCESS フィールド)

この 2 つの関数のアトミック性は、互いに先取りする可能性のある複数のタスクを持つ CPU では不可欠です。このレジスタは取得されると、IPC_STRUCTx_RELEASE レジスタに任意の値を書き込むことで解放されます。レジスタがすでに取得状態にある場合、そのレジスタを再度読み出そうとしても取得できません。

- IPC_STRUCTx_NOTIFY: このレジスタは、IPC 通知イベントを生成するために使用されます。このレジスタの各ビットは IPC 割込み構造に対応します。IPC チャネルから生成された通知イベントは、任意または複数の割込み構造をトリガーできます。
- IPC_STRUCTx_RELEASE: このフィールドは、IPC 割込み構造をアンロックし、解除イベントをトリガーしますが、ロックが取得されたとき (LOCK_STATUS.ACQUIRED が'1') のみです。IPC チャネルによって生成されたリリース イベントは、任意または複数の割込み構造をトリガーできます。IPC チャネルのみをリリースして割込みを発生させないようにするためには、IPC リリース レジスタにゼロを書き込みます。割込みのマスキングは、IPC_INTR_STRUCTx_INTR_MASK レジスタでも可能です。
- IPC_STRUCTx_DATA0 および IPC_STRUCTx_DATA1: これらのレジスタはデータを保持するためのもので、どちらも 32 ビットサイズです。これらのレジスタは、チャネルの共有データ メモリとみなせます。通常、これらのレジスタは、セキュア アプリケーションと非セキュア アプリケーションの間で通信する必要のあるメッセージを保持します。メッセージが合計 64 ビット サイズより大きい場合、これらのレジスタの一方または両方にポインタを置きます。
- IPC_STRUCTx_LOCK_STATUS: このレジスタは、IPC チャネルの瞬時ロック ステータスを提供します。チャネルが獲得された場合 (LOCK_STATUS.ACQUIRED が'1')、それぞれのビット フィールドに保護コンテキスト、セキュリティ、その他の情報が表示されます。ただし、ロックが獲得されていない場合 (LOCK_STATUS.ACQUIRED が'0')、フィールド P、NS、PC、MS の値は無効です。ロック ステータスの読み出しは即時のステータスを示すだけで、次のサイクルで変化する可能性があります。

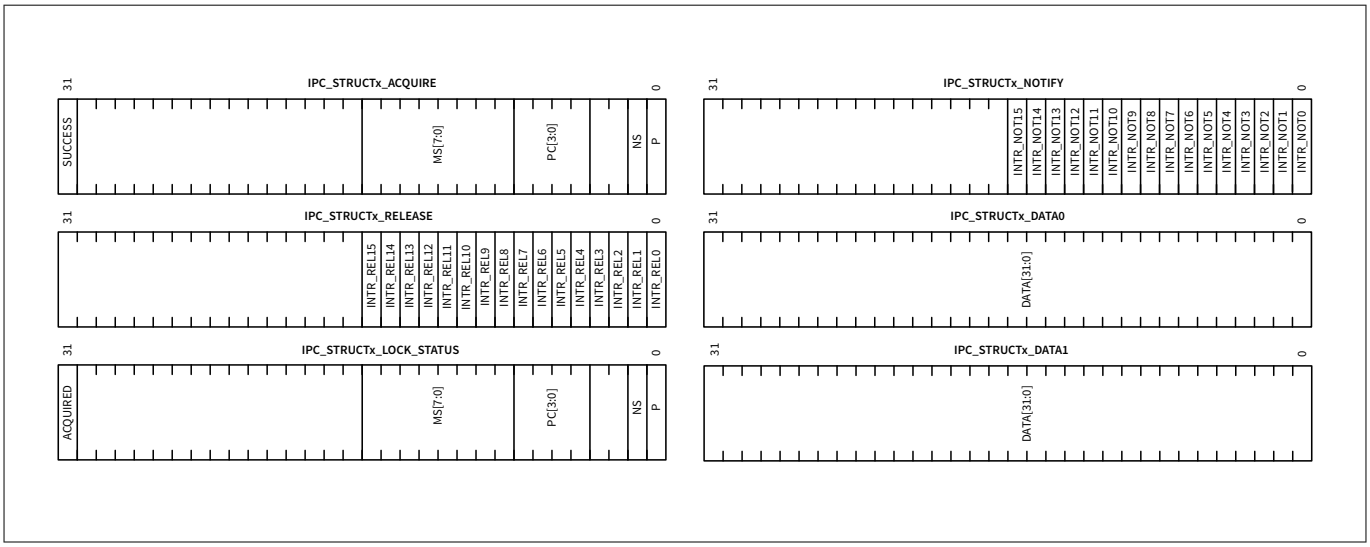


図 4　　IPC チャネル構造

4.2.2　　IPC 割込み

システムの各 IPC 割込みラインは、対応する IPC 割込み構造を持っています。IPC 割込みは、システム内のどの IPC チャネルからの通知またはリリースイベントによってもトリガーされます。IPC 割込みレジスタを使って、これらのイベントのソースをマスクできます。図 5 に IPC 割込み構造のレジスタを示します。

- IPC_INTR_STRUCTx_INTR: このレジスタは、割込みソースの瞬時ステータスを提供します。このレジスタには 16 の通知イベント ビットと 16 の解除イベント ビットがあることに注意してください。これらは、16 の IPC チャネルに対応した通知イベントと解除イベントです。IPC チャネル 0 で通知イベントがトリガーされると、対応す

る Notify 0 ビットが割込みレジスタでアクティブになります。ビットに‘1’を書き込むと、割込みはクリアされます。

- IPC_INTR_STRUCTx_INTR_MASK: このレジスタのビットは割込みソースをマスクします。マスクが有効になっている割込みソースのみが、割込みをトリガーできます。
- IPC_INTR_STRUCTx_INTR_SET: このレジスタに‘1’を書き込むと、割込みが設定されます。
- IPC_INTR_STRUCTx_INTR_MASKED: このレジスタは、マスクされた後の割込みの瞬時値を提供します。このレジスタの値は IPC_INTR_STRUCTx_INTR と IPC_INTR_STRUCTx_INTR_MASK です。

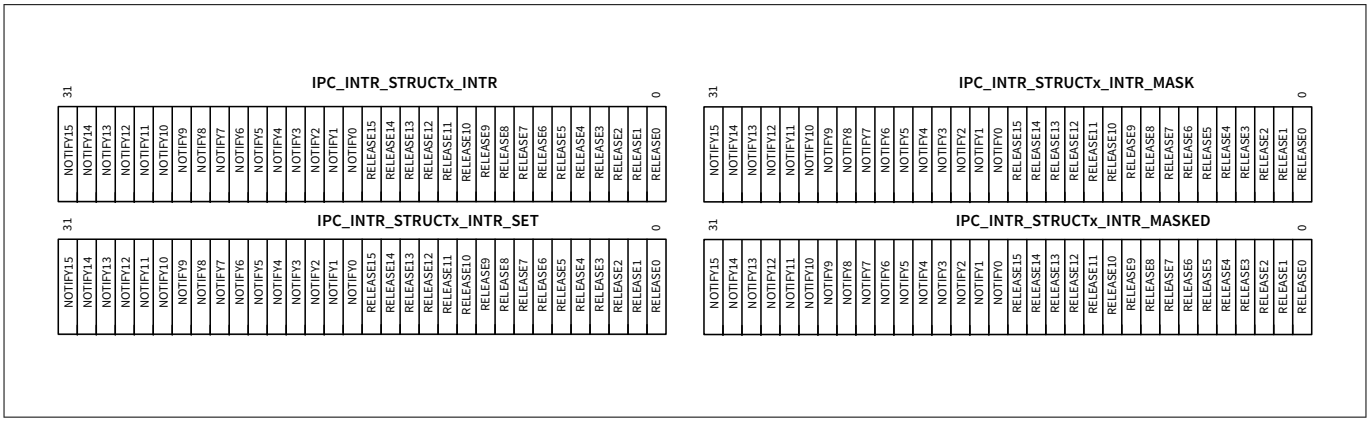


図 5　　IPC 割込み構造

4.2.3　　IPC チャネルと割込み

IPC ブロックには、関連する IPC 割込みのセットがあります。各 IPC 割込みレジスタ構造は、IPC 割込みラインに対応しています。この割込みは、システム内の任意のプロセッサで割込みをトリガーします。プロセッサの割込みルーティングは、デバイス アーキテクチャに依存します。

各 IPC チャネルは解除レジスタと通知レジスタを持ち、IPC 割込みのいずれかにイベントを発生させられます。図 6 に、この IPC チャネルと IPC 割込み構造の関係を示します。

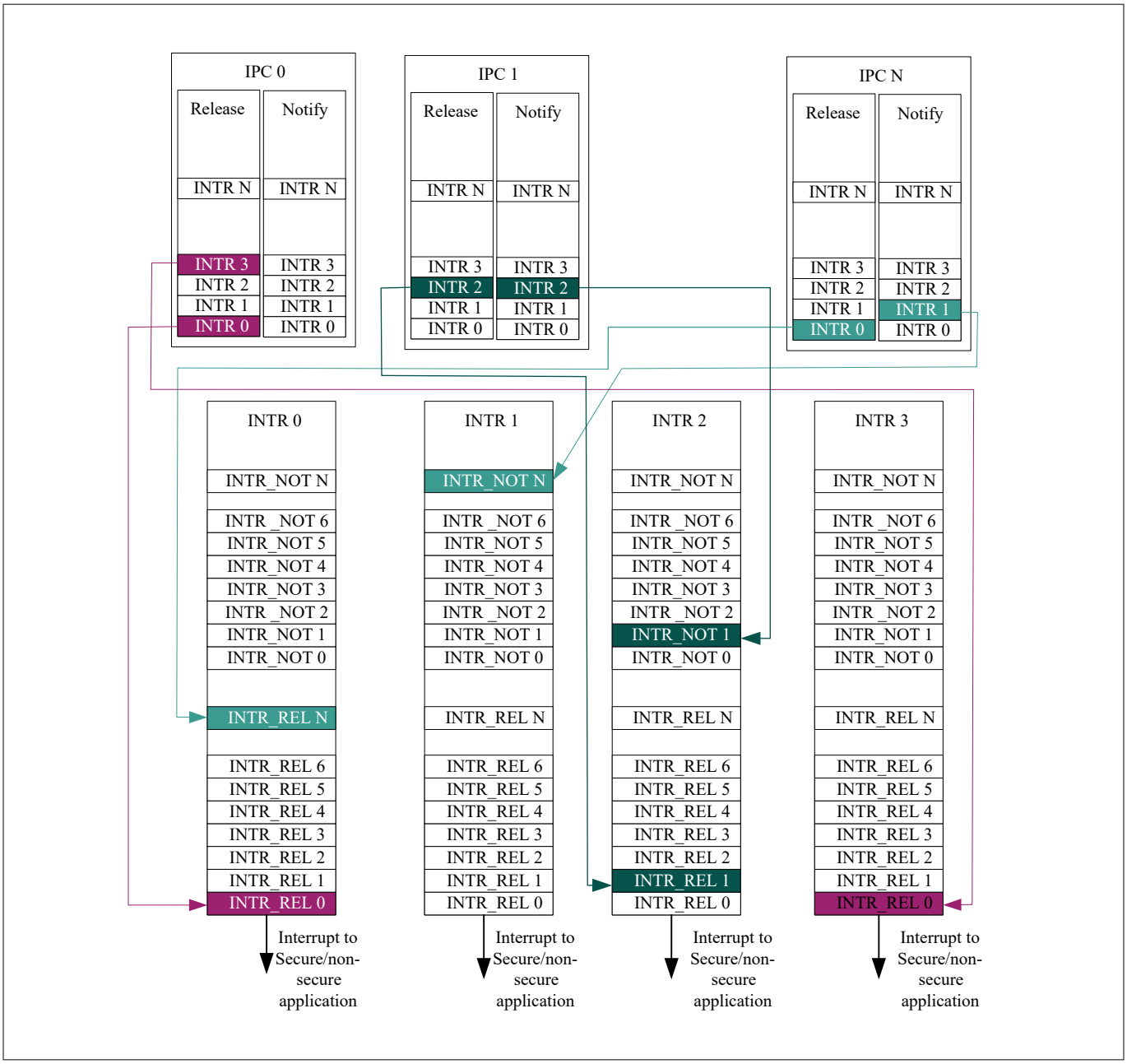


図 6　　IPC チャネルと割込み

4.3　　ロックの実装

IPC チャネルはロックの実装に使用できます。ロックは通常、セキュア アプリケーションと非セキュア アプリケーションの間で使用され、共有リソースへの何らかの形の相互排他的アクセスを可能にします。セキュア アプリケーションと非セキュア アプリケーションがリソースを共有する場合、セキュア アプリケーションまたは非セキュア アプリケーションは IPC チャネルを獲得および解放できます。そのため、セキュア アプリケーションも非セキュア アプリケーションも、IPC チャネルをロックとして想定できます。このコードのセマンティクスは、共有リソースへのアクセスは、チャネルのセキュア アプリケーションと非セキュア アプリケーションの所有権によってゲートされるというものです。そのため、セキュアまたは非セキュアのアプリケーションは、共有リソースにアクセスする前に IPC チャネルを取得する必要があります。

セキュアまたは非セキュアなアプリケーションが IPC チャネルを取得できなかった場合、共有リソースがロックされたことを意味します。IPC チャネルは、どのアプリケーションがチャネルを獲得または解放するのか強制しない

ことに注意してください。セキュアまたは非セキュアのアプリケーションは、IPC チャネルを獲得または解放できます。

4.4 メッセージ パッシング

IPC チャネルは、セキュア アプリケーションと非セキュア アプリケーションの間のメッセージ通信に使用できます。この使用例では、チャネルは割込み構造と組み合わせて使用されます。IPC チャネルは、データ レジスタへのアクセスをロックするために使われます。IPC チャネルは送信者によって取得され、メッセージの入力に使われます。受信者はメッセージを読み、チャネルを解放します。したがって、送信者がデータをチャネルに送信し、受信者がそれを読み出すまでの間、チャネルは他のすべてのタスクのアクセスに対してロックされます。送信側は、受信側の IPC 割込みの notify イベントを使用して送信操作を示します。受信側はこの割込みに反応し、データ レジスタからデータを読み出します。受信が完了すると、受信側はチャネルを解放し、送信側の IPC 割込みに対して解放イベントを生成することもできます。チャネルをロックしても、ハードウェアのデータ レジスタへのアクセスは制限されません。これは、ソフトウェアによって強制されるべきセマンティックです。

図 7 に、送信者 (非セキュア アプリケーション) が受信者 (セキュア アプリケーション) にデータを送信する例を示します。IPC 割込み A は、非セキュア アプリケーションに割り込むように設定されています。IPC 割込み B は、セキュアなアプリケーションに割り込むように設定されています。

1. 送信側は、IPC_STRUCTx_ACQUIRE レジスタの読み出しによって、IPC チャネルの獲得を試みます。チャネルを獲得した場合、送信側はデータ送信のためのチャネルの所有権を持ちます。これは、チャネルとそれに対応する IPC_STRUCTx_LOCK_STATUS レジスタのステータスも変更します。チャネルが獲得されていない場合、送信側はチャネルが獲得可能になるまで待つ必要があります。これは、IPC チャネルの IPC_STRUCTx_LOCK_STATUS レジスタをポーリングすることで可能です。
2. IPC チャネルが獲得されると、送信側は通信のためにチャネルを制御し、IPC_STRUCTx_DATA0 と IPC_STRUCTx_DATA1 レジスタにデータを配置します。
3. メッセージが IPC チャネルに置かれているので、送信側は受信側の割込みラインに notify イベントを生成します。これは、IPC チャネルの IPC_STRUCTx_NOTIFY レジスタの対応するビットを設定することで実現します。このイベントは IPC 割込み B (IPC_INTR_STRUCTx_INTR) で通知イベントを生成します。IPC 割込み B のマスクビット (IPC_INTR_STRUCTx_INTR_MASK [31:16]) を設定することで、IPC チャネルの通知イベントが有効になると、受信側に割込みが発生します。
4. IPC 割込み B を受信すると、受信側は IPC_INTR_STRUCTx_INTR_MASKED レジスタをポーリングして、どの IPC チャネルが notify ベントをトリガーしたのかを知ることができます。これを元に、受信側は読み出すチャネルを特定し、IPC チャネルの IPC_STRUCTx_DATA0 および IPC_STRUCTx_DATA1 レジスタから読み出します。これで受信者は送信者から送られたデータを受信したことになります。他のプロセスが使用できるように、チャネルを解放する必要があります。
5. 受信側はチャネルを解放します。また、オプションで、送信側の IPC 割込み A にリリースイベントを発生させることもできます。

リリース割込みを受信すると、送信側はアプリケーションの要求に基づいてイベントに対処できます。さらに送信するためにチャネルを再取得しようとするか、送信が完了したため他のタスクに移せます。

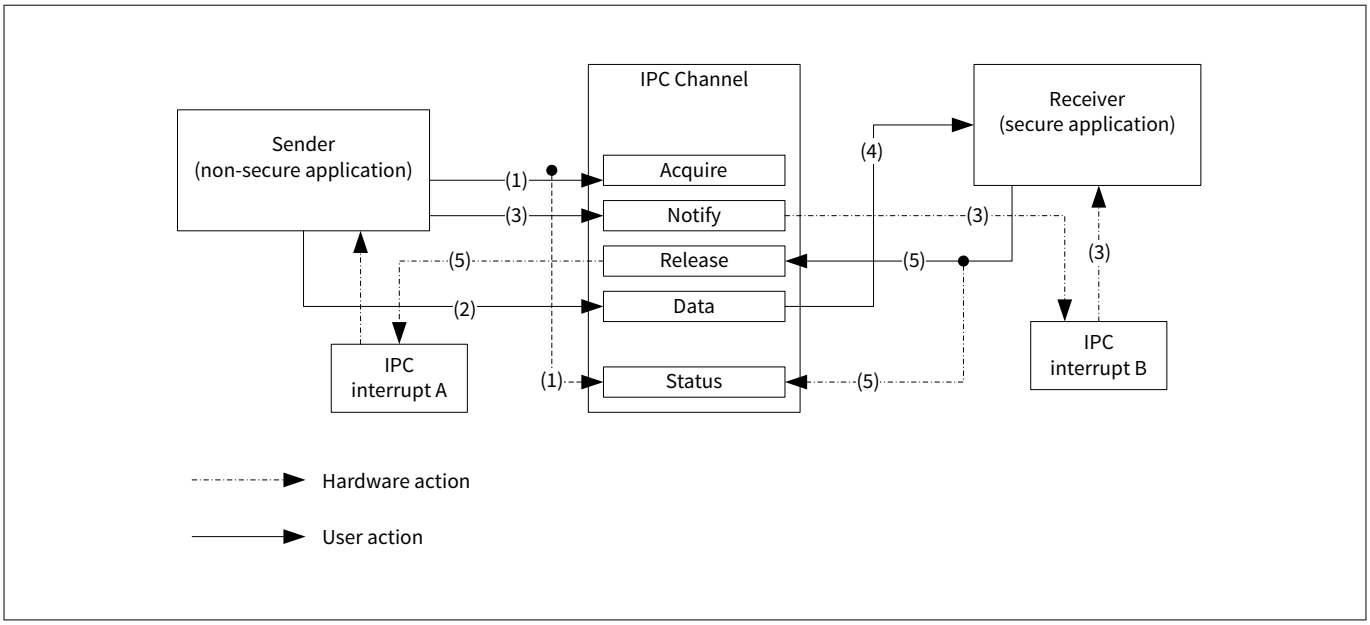


図 7　　IPC を使ったメッセージ送信

前の例では、送信されるデータのサイズは 64 ビットしかありませんでした。さらに大きなメッセージはポインタとして送信できます。送信側は、大容量のメッセージ構造体をメモリ上に確保し、データ レジスタへのポインタを渡せます。使い方は図 8 を参照してください。ユーザー コードは、メッセージ読み出し処理の同期を実装しなければいけないことに注意してください。

- この実装では、受信側がメッセージ パケット内のデータをすべて使用し、メッセージ パケットの書き換えができるまで、チャネルを停止できます。これは、IPC チャネル数が限られているため、他のプロセス間通信を止めてしまうので意味がありません
- 受信側は、メッセージ パケットへのポインタを受け取るとすぐにチャネルを解放できます。メッセージパケットに同期ロジックをフラグとして実装し、送信側は書き込み完了時にセットし、受信側は読み出し完了時にクリアします。

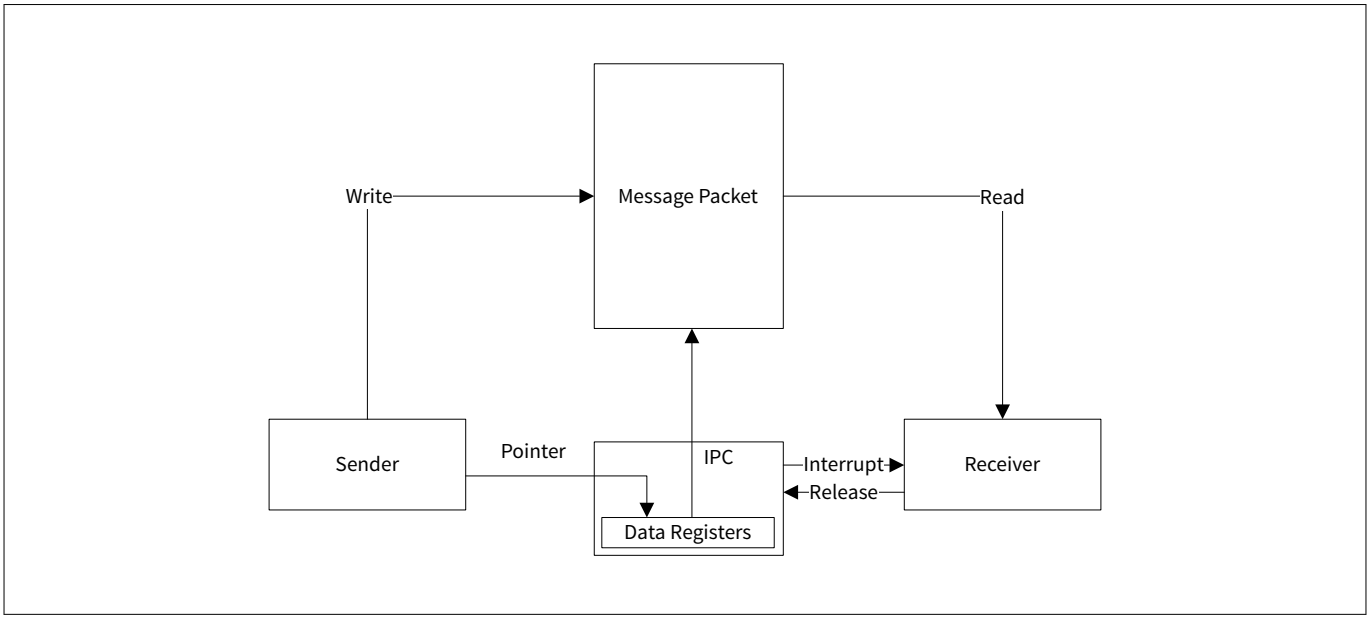


図 8　　大容量メッセージ通信

4.5 代表的な使用モデル

PSOC™ Control C3 IPC のユニークなチャネルと割込みアーキテクチャは、セキュアなアプリケーションと非セキュアなアプリケーション間の通信用にさまざまな使用モデルに対応します。ここに挙げるのは、そのいくつかの例です。IPC アーキテクチャを使用して実現が可能な通信モデルは、この文書に記載されているものがすべてではありませんので、注意してください。また、本文書はハイレベルな使用モデルのみについて記載しており、通信におけるデータ管理の詳細については記載していないことに注意してください。データ管理については、特定のアプリケーションのユースケースにしたがって決めなければいけません。

4.5.1 全二重通信

この使用モデルでは、セキュア アプリケーションと非セキュア アプリケーションの間で、通信の方向に応じて IPC チャネルが使用されます。この例では、IPC チャネル X は非セキュア アプリケーションからセキュア アプリケーションへのデータ通信専用で、IPC チャネル Y はセキュア アプリケーションから非セキュア アプリケーションへのデータ通信専用です。IPC 割込み X は、セキュア アプリケーションのイベントを通知します。したがって、その割込み出力は、セキュア アプリケーションのシステム割込みに接続されています。イベントは、システム バス経由で IPC 割込みレジスタ構造に書き込むことでトリガーされます。同様に、IPC 割込み Y は非セキュア アプリケーション専用です。

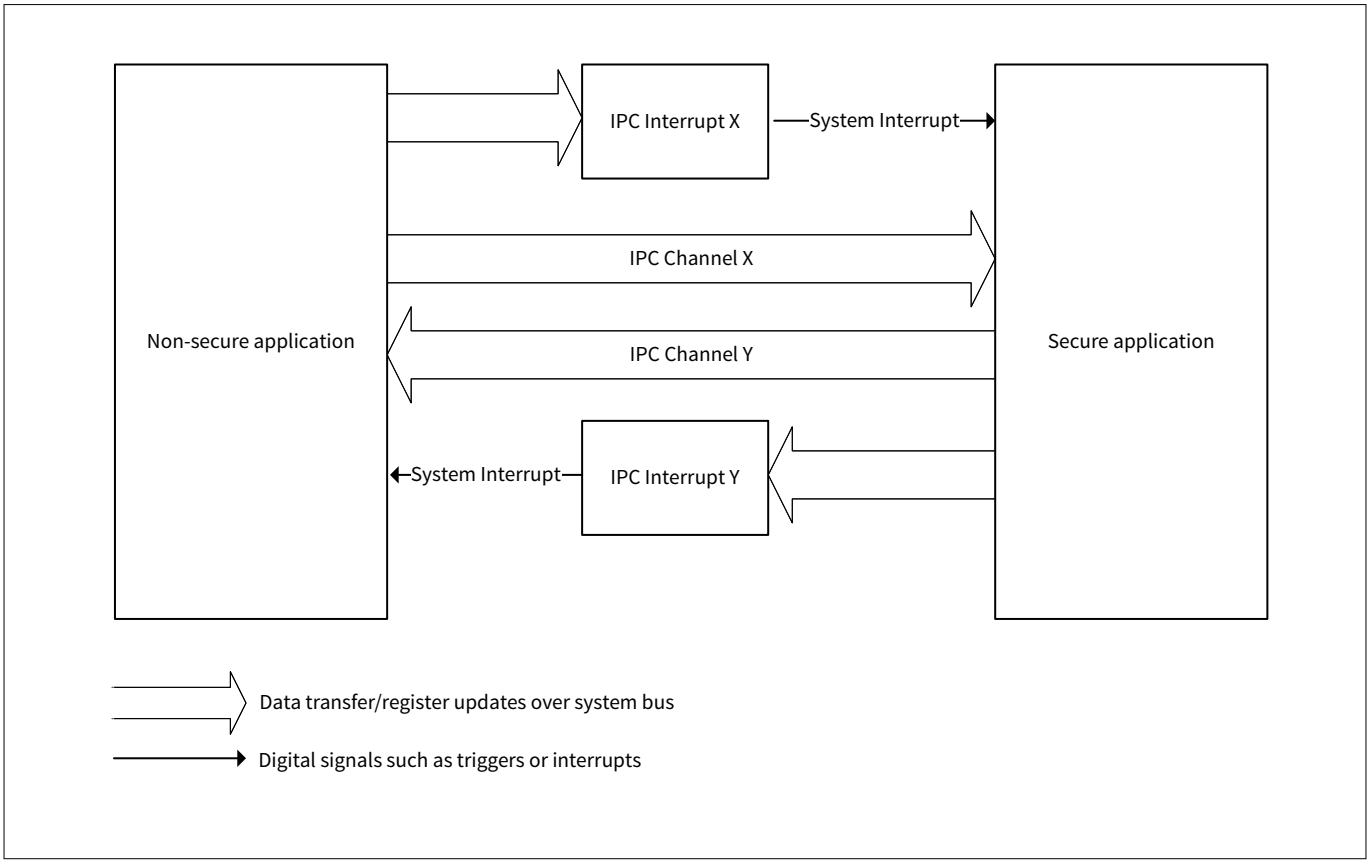


図 9 セキュア アプリケーションと非セキュア アプリケーション間の全二重 IPC

4.5.2 独立したイベント処理による半二重通信

この場合、1 つの IPC チャネルのみを使用して、セキュア アプリケーションと非セキュア アプリケーション間の転送を設定します。つまり、一度にデータ転送を制御するのは片側のみということです。1 つの共有 IPC チャネルの競合を避けるために、チャネルのロック レジスタを使用する必要があります。2 つの IPC 割込み構造が使用されるため、2 つの独立したイベントがサポートされます。図 10 に、このモードを示します。

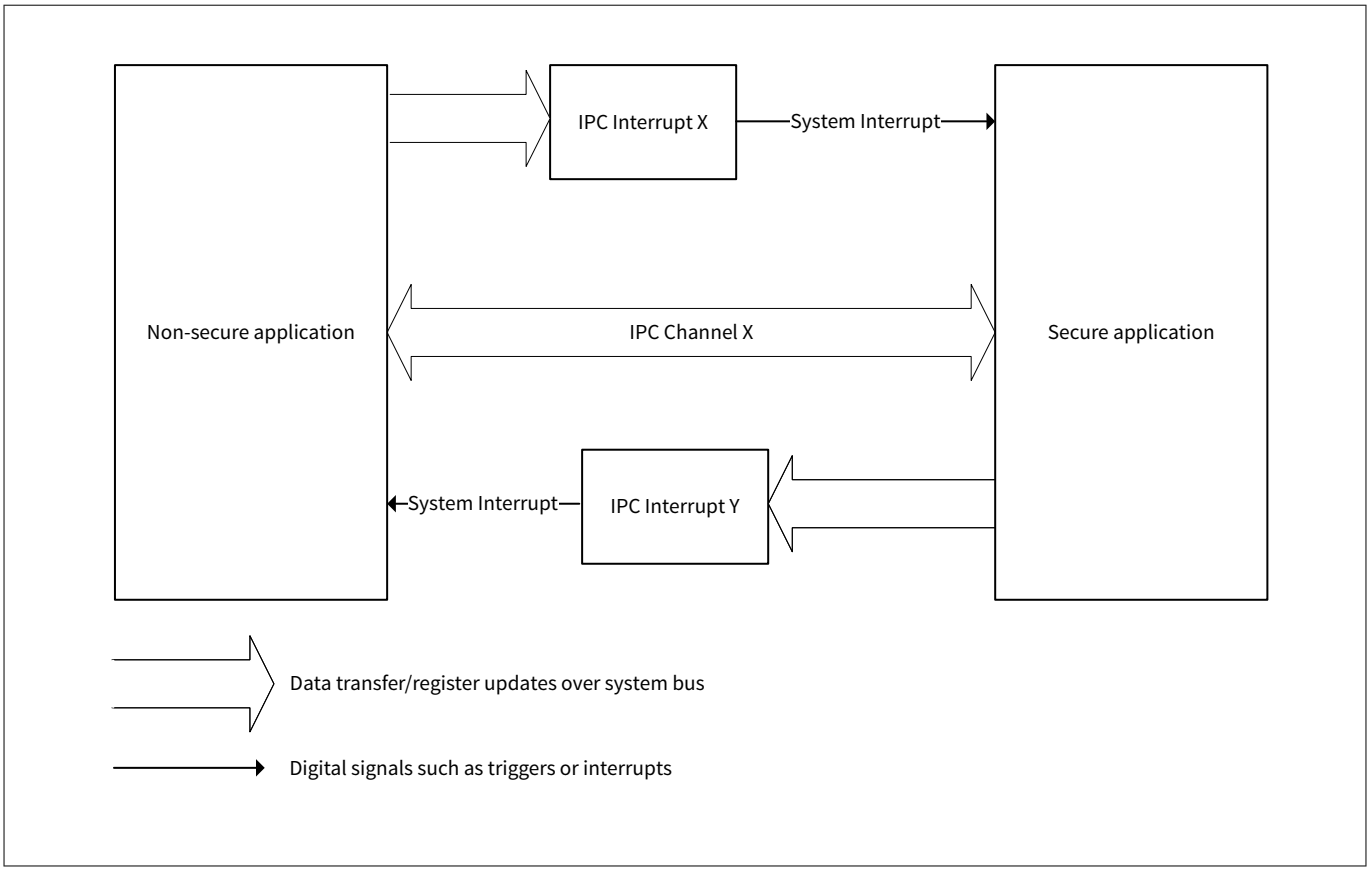


図 10　　独立したイベント処理による半二重通信

4.5.3　　共有イベント処理による半二重通信

このモデルでは、IPC チャネルと割込みの両方が、セキュア アプリケーションと非セキュア アプリケーションの間で共有されます。割込みも共有されるため、割込みレジスタへのアクセスはチャネルの IPC ロックを使用して管理する必要があります。図 11 に示すように、IPC 割込みはセキュア アプリケーションでも非セキュア アプリケーションでも割込みをトリガーするよう設定されています。したがって、セキュアまたは非セキュア アプリケーション割込みは、どのアプリケーションが IPC を制御しているかをチェックし、メッセージとイベントがそのアプリケーションのものであるかどうかを判断するロジックを ISR に持つ必要があります。

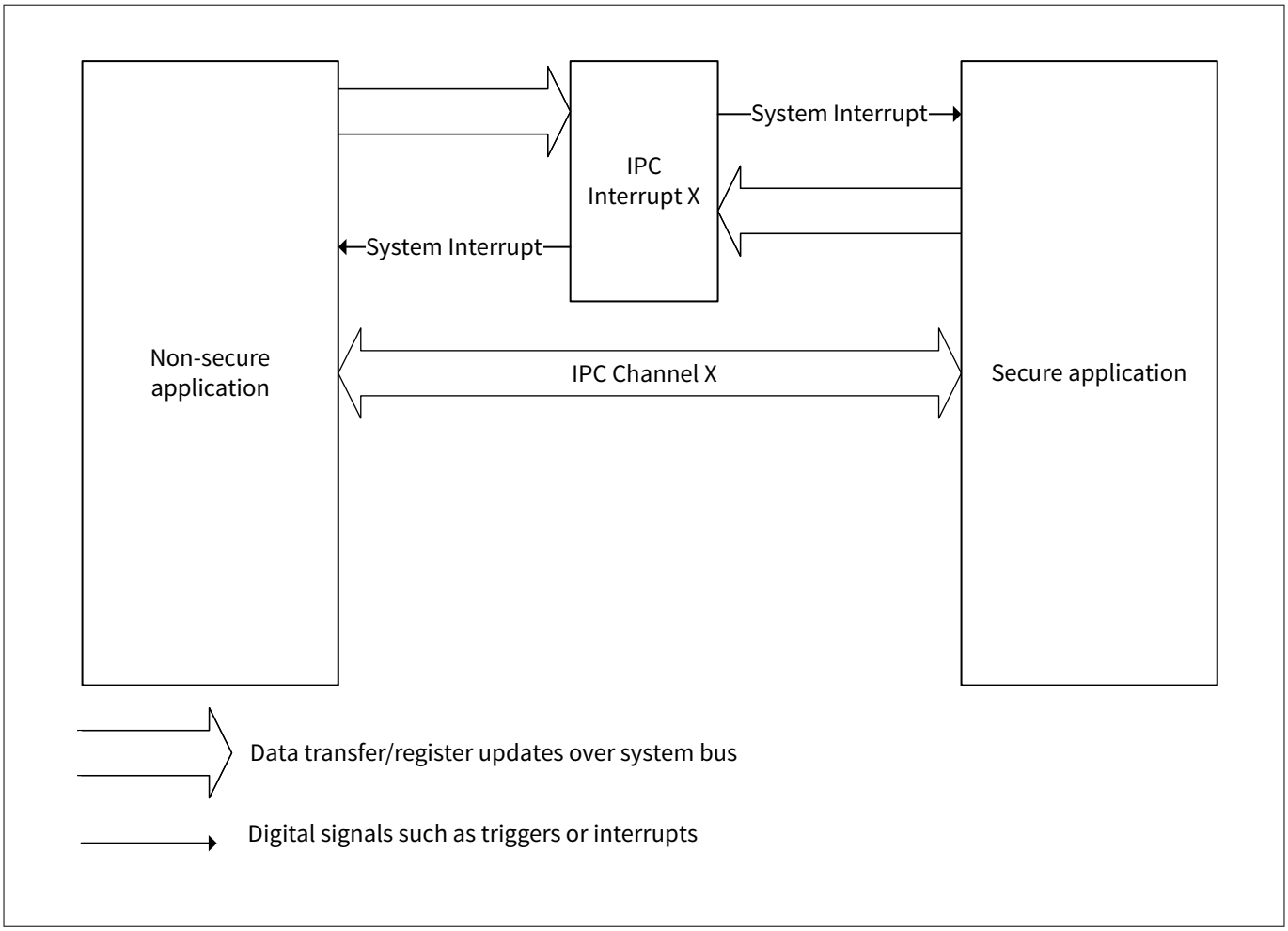


図 11　　共有イベント処理による半二重通信

5 フラッシュ

PSOC™ Control C3 フラッシュは、高いバルク プログラム性能を提供し、超低消費電力動作に対応しています。フラッシュは通常、デバイスの電源がオフのときに CPU の命令やデータを保存するために使用されます。フラッシュへの書き込みは可能ですが、そのプロセスは SRAM よりはるかに遅く、制限も多いです。ここでは、フラッシュ メモリの構造と機能について説明します。

5.1 特長

フラッシュには次のような特長があります。

- 最大 256 KB のフラッシュ メモリ
- 512 バイトの行サイズ; 最小プログラマブル ユニット
- 1 ビット エラー訂正および 2 ビット エラー検出 (SECDED) 用 9 ビット ECC
- Read While Write 機能に対応
- MPC (memory protection controller) を使用したフラッシュメモリ保護
- プログラム可能な待機状態数に対応
- 低消費電力モード管理用 Arm® Q チャネル LPI
- 10～20 年間の保持期間
- 100k プログラム サイクルの耐久性

5.2 設定

5.2.1 ブロック図

ここでは、FLASHC 内の主なコンポーネントについて説明します。

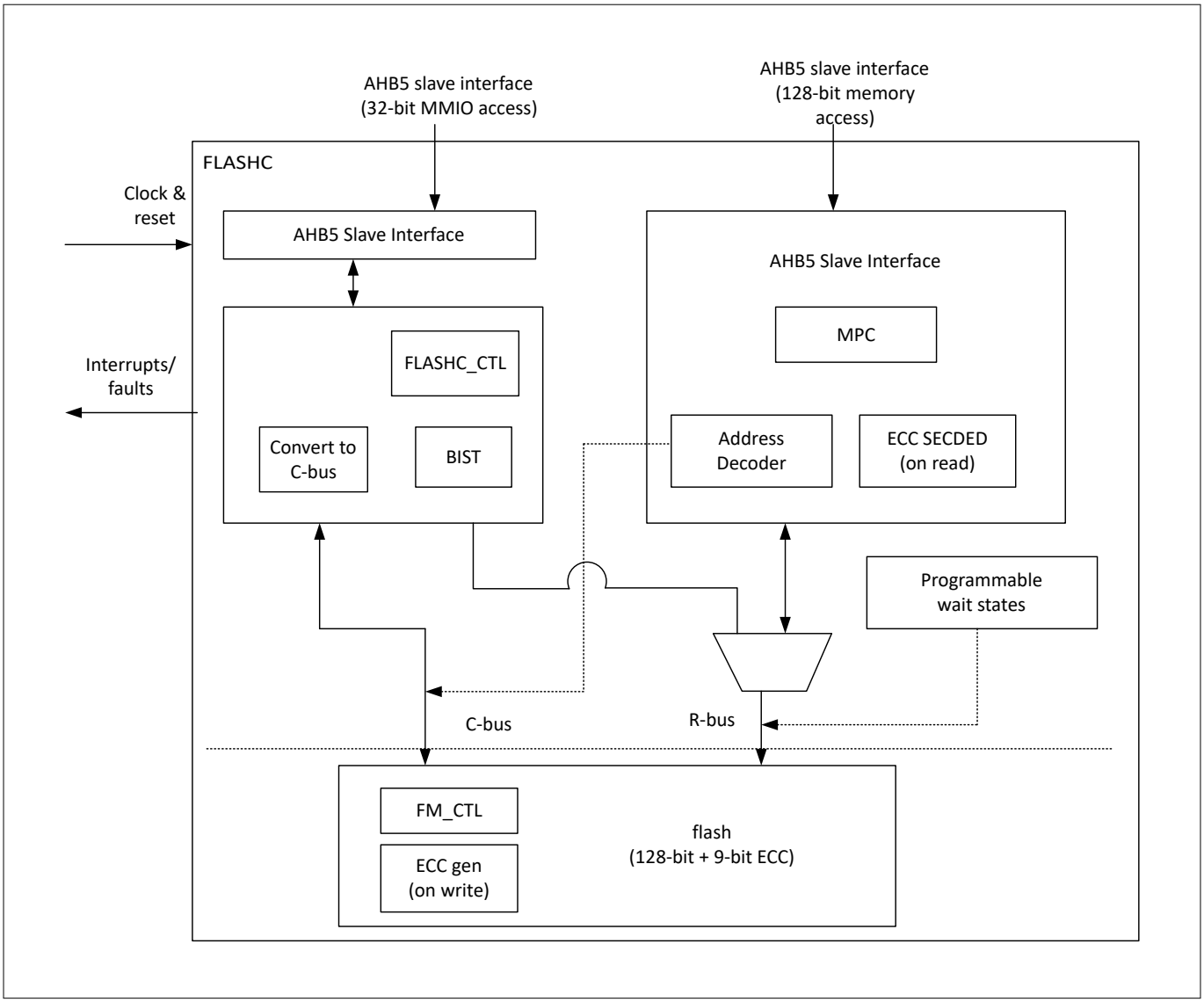


図 12　　FLASHC ハイレベル ブロック図

5.2.2　　フラッシュの構造

下図にフラッシュマクロ (FM) の構造を示します。

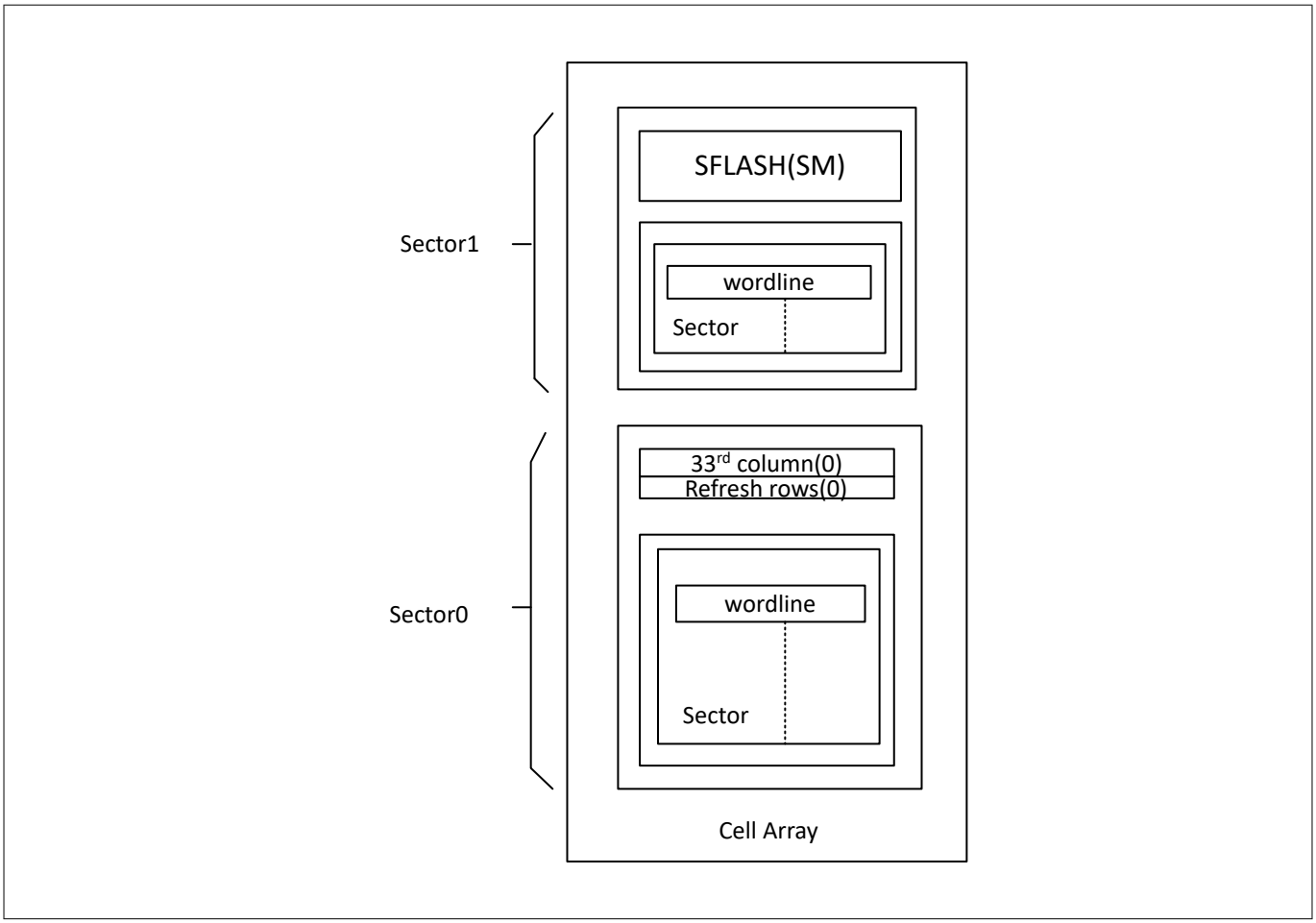


図 13　　フラッシュの構造

- ワードの読み出し: フラッシュ マクロ (FM) にはワードの読み出しサイズがあり、各読み出しコマンドで読み出して、返します。各ワードは 128 データビットで、9 ビットの誤り訂正符号 (ECC) 機能を搭載しています。
- Row (行): Row は FM のワードラインに関連付けられており、その全体のみを書き込み/消去できます。32 ユニットのリード ワードで構成されています。各ワードラインのサイズは 512 バイトです。
- セクタ: セクタは 256 行で構成されます。各セクタのサイズは 128 KB です。
- 33 列目: FM では、同じセクタ内の他の行に消去/書き込みサイクルが発生すると、選択されていない行にプログラム妨害が発生し、メモリセルの耐久性/保持性が低下します。33 列目は、これらのカウント値をとおしてセクタ内の最も古い行と最も若い行を追跡するカウンターとして使用されます。33 列目のデータは、リフレッシュ操作を開始するために使用されます。
- リフレッシュ行: これらの行は各セクタにあり、リフレッシュ操作にのみ対応する追加ページです。リフレッシュ操作は、行内のオリジナル データを消去、破壊してからプログラムを書き換えます。このシーケンス中に中断があると、重要なデータ行が破損または消去された状態で残る可能性があります。したがって、リフレッシュされるページのデータは、リフレッシュ行にコピーされます。その後、リフレッシュされる行は消去され、リフレッシュされる行と同じデータと新しいカウンター値で再プログラムされます。SFLASH はこのセクタと共有されているため、セクタ 1 でのリフレッシュは許可されていません。リフレッシュ行へのアクセスは、リカバリー プロセスの破損を防ぐため、保護コンテキスト (PC) を 0、1、または 2 に設定する必要があります。リフレッシュのテスト、必要に応じてフラッシュ リフレッシュ ルーチンを開始するのは、OEM コードが行います。

***注**:　　EEPROM エミュレーション メモリは物理セクタ 0 に割り当てなければいけません。*

5.2.3 バンクモードとアドレスマッピング

このフラッシュ メモリ コントローラーは、デュアル バンク モード機能を備えています。デュアルバンクモードを使用する場合、フラッシュメモリ領域は 2 つのハーフバンクに分割されます。1 つはバンク 0 と呼ばれ、もう 1 つはバンク 1 と呼ばれます。

フラッシュメモリは、同じフラッシュ上で書き込みを行いながら読み出しも行えます。つまり、一方の論理バンクに書き込みを行い、もう一方の論理バンクから読み出しを行えます。書き込みと読み出しを同じ論理バンクから行うとエラーになります。

メインフラッシュ領域はデュアルバンクモードに対応しています。ユーザーは FLASH_CTL.BANK_MODE でこのモードを選択できます。

このモードは、フラッシュ メモリ内のソフトウェア イメージのファームウェア アップデートに対応するためのものです。

これらのアドレス マップは、以下のように設定することでバンクスワップに対応できます。

- シングルバンクモードを設定する場合、メインフラッシュ領域全体が 1 つの連続したアドレス領域としてマップされます。
- フラッシュメモリ領域は 2 面に分割され、各半分は別々のアドレス領域として表示されます。さらに、実行するプログラムを入れ替えて、同一領域のファームウェアアップデートに対応できます。

- マッピング A を選択すると、バンク 0 が下段に、バンク 1 が上段に表示されます。

- マッピング B を選択すると、バンク 0 が上段に、バンク 1 が下段に表示されます。

フラッシュバンクを切り替える際は、以下のルールに従わなければいけません。

- バンクを正しく切り替えるには、BANK_MODE LSB を変更する必要があります。BANK_MODE LSB を変更せずに BANK_MAPPING を変更すると、エラーが発生することがあります。

フラッシュ アドレスのマッピングについては表 11 を参照してください。

5.3 フラッシュコントローラー

フラッシュ コントローラーは、以下の FM プラットフォーム コントローラー機能を提供します。

- 128 ビット CODE インターコネクトに接続された MPC 保護付き 128 ビット AHB5 ホスト インターフェース上の FM から読み出してください
- 同じ 128 ビット AHB5 ホスト インターフェースまたは 32 ビット AHB5 MMIO インターフェース上の FM への書き込みシーケンス開始。制御とステータス チェックのための連続的なシーケンス進行は、PPC 保護機能付き周辺インターコネクトに接続された 32 ビット AHB5 MMIO インターフェースによって実現されます。
- FM には、フル R バス、フル C バス、RWW (Read While Write) モードの 3 つの動作モードがあります。
  - FLASHC は、AHB5 トランザクションを FM データ読み出し用の R バスインターフェースに変換します。
  - 128 ビット AHB5 ホスト インターフェースにおける FM データ読み出しの待機状態 (FLASH_CTL.RBUS_WS)。
- FLASHC の MMIO および FM の読み出し、書き込み、制御/設定用の FM MMIO アクセス用の 32 ビット AHB5 MMIO インターフェース:
  - 32 ビット ワード アラインド アクセスのみ
  - PERI 内のペリフェラル保護コントローラー (PPC) を使用したペリフェラル保護
  - FLASHC は、AHB5 MMIO トランザクションを FM シーケンス進行用の C バスインターフェースに変換します。
- フォールト インターフェースは、CPUSS 中央フォールト レポート インフラストラクチャーへの接続に対応しています。
- 設定可能な FM のシングルまたはデュアルバンクモード

5.3.1 電力モード

PSOC™ Control C3 は、OFF、HIBERNATE、DEEPSLEEP_OFF、DEEPSLEEP_RAM、DEEPSLEEP、ACTIVE などのシステム電力モードに対応しています。次の表に、システムおよびフラッシュ電力モードのマッピングを示します。

表 11 フラッシュ電力モード マッピング

システム電力モード	FM 電力モード	FM のウェイクアップ時間	条件
ハイバネート	ハイバネート	< 10 us	全アナログ回路オフ
DEEPSLEEP_OFF	ハイバネート	< 10 us	全アナログ回路オフ
DEEPSLEEP_RAM	ハイバネート	< 10 us	全アナログ回路オフ
DEEPSLEEP	ハイバネート	< 10 us	全アナログ回路オフ
SLEEP	スタンバイ	-	全アナログ回路オン、クロックなし
ACTIVE	アクティブ	-	全アナログ回路オン、クロックあり

5.4 RWW (Read While Write) 対応

PSOC™ Control C3 MCU は、1 つのセクタに対する読み出し動作と、別の領域に対する書き込みや消去に対応しています。FM には 2 つのセクタがあり、それぞれ 128 KB の大きさです。RWW 機能はセクタ間で利用可能で、アプリケーションは、あるセクタで書き込みまたは消去動作が進行している間に、別のセクタから読み出し、または実行できます。ただし、コードの実行または読み出しが所定のセクタ (例えばセクタ 0) の最後の 16 バイトにあり、フラッシュの書き込み/消去動作が次のセクタ (セクタ 1) にある場合、キャッシュ プリフェッチが有効になっていると、RWW 違反が発生することがあります。これは、同じセクタで書き込みが行われている間に、プリフェッチがセクタ 1 の一部である次の 16 バイトのデータをフェッチするためです。フォールトにつながるため、ファームウェアの設計時に考慮する必要があります。ファームウェアは、セクタの最後の 16 バイトが決してアクセスされないように、各セクタの最後の 16 バイトにデッド コードまたは定数データを配置するように設計でき、またはフラッシュの書き込みまたは消去中にキャッシュのプリフェッチを無効にできます。

RWW FM は、Read-bus による読み出し動作、Control-bus による読み出しまたは書き込み動作、または Read-bus による読み出し動作と Control-bus による書き込み動作を行う RWW モードとして使用できます。Read-bus と Control-bus は、別々のセクタアドレスレジスタと行アドレスレジスタとプリデコーダーを持ち、それらの出力はセクタレベルで多重化されています。書き込み動作中、同じセクタでの同時読み出しはできません。

RWW 中、書き込みセクタに対して以下の操作が可能です。

- 行の消去またはプログラム
- セクタ消去またはプリプログラム

RWW 中、以下は使用できません。

- 書き込みセクタでのデータ読み出し
- 一括消去またはプリプログラム
- テスト モード

5.5 ECC

フラッシュ コントローラーはエラー訂正符号 (ECC) に対応しています。FM は、128 ビットのワードごとに 9 ビットを追加して使用し、1 ビットエラー訂正 (SEC) と 2 ビットエラー検出 (DED) を実行します。ECC により、内部読み出し幅は 128 ビットから 137 ビットに拡張されています。

ECC ブロックは、不正な ECC 値を挿入することでテストできます。詳細は、ECC (エラー) インジェクションを参照してください。

5.5.1 ECC 生成/エンコーディング

ECC エンコーダーは FM に、ECC デコーダーは CPUSS 内のフラッシュ コントローラーに配置されています。フル C バスまたは RWW モードでは、FM ハードウェア生成 ECC または FM に書き込まれたソフトウェア指定 ECC のいずれかが、書き込みシーケンスで FM C バス ECC 生成/エンコーディングに従うようにソフトウェア設定されます。

- FM ANA_CTL0.ECC_ENC_DIS == 1'b0 のとき、ECC 機能付き FM で ECC エンコーダーが有効になっていることを意味します。PL ECC ビットは自動的にロードされます。
- FM ANA_CTL0 の場合、ECC_ENC_DIS == 1'b1 は、ECC エンコーダーが無効であることを意味します。FM は FM_PL_ECC.DATA9 の ECC パリティを使用します。

5.5.2 ECC (エラー) インジェクション

ECC インジェクションは、レジスタ ECC_INJ_EN.ECC_INJ_ENABLE で有効または無効にできます。フル R バス モードまたは RWW モードでは、ECC_INJ_EN.ECC_INJ_ENABLE に基づいて ECC パリティを挿入し、フラッシュ マクロから供給される ECC パリティを置き換えられます。ECC_CTL.WORD_ADDR および ECC_CTL.PARITY レジスタは、挿入されるアドレスとパリティを指定します。ECC_CTL.PARITY は、ECC_INJ_EN.ECC_INJ_PC と ECC_CTL.WORD_ADDR の両方が 128 ビット インターフェース上の AHB5 読み出しトランザクションと一致すると挿入されます。

フル R バスモードまたは RWW モードでは、ECC パリティ (FM によって生成される ECC パリティを置き換える) を、読み出し中の ECC 訂正またはデコードロジックの前に挿入できます。フラッシュ MMIO の ECC_INJ_EN/ ECC_INJ_CTL を有効にし、128 ビット ロング ワードのプログラム カウンターとアドレスに基づいて 9 ビット ECC を挿入するよう設定できます。

ECC_INJ_EN.ECC_INJ_ENABLE が LOW の場合、ECC_INJ_EN.ECC_INJ_PC は、ECC_INJ_ENABLE を HIGH にイネーブル/ コンフィギュレーションしたマスターからプログラム カウンター (PC) を継承します。

- ECC_INJ_ENABLE が HIGH のとき、ECC_INJ_PC で指定された PC のみが ECC_INJ_EN (ENABLE、ECC_ERROR、および ECC_INJ_PC ビット) にアクセス (読み出し/書き込み) できます。
- PC 違反はバスエラーとなり、追加のフォールト/割込みは発生しません。唯一の例外は PC0 で、PC0 はいつでも ECC_INJ_EN にアクセス (読み出し/書き込み) して、ECC_INJ_EN の潜在的なデッドロックを解除できます。

ECC_INJ_CTL.WORD_ADDR は挿入するアドレス (モジュール内部オフセット アドレスに基づく) を、ECC_INJ_CTL.PARITY は挿入するパリティを指定します。

- ECC_INJ_ENABLE が HIGH のとき、ECC_INJ_PC で指定された PC のみが ECC_INJ_CTL にアクセス (読み出し/書き込み) できます。
- PC 違反はバスエラーとなり、追加のフォールト/割込みは発生しません。

PARITY は、ECC_INJ_PC と WORD_ADDR の両方が 128 ビット インターフェー ス上の AHB 読み出 しトランザクションにー致すると挿入されます。

- ECC_INJ_CTL.PARITY は、FM からの ECC パリティを置き換え、ECC 訂正/復号ロジックによって使用されます。
- ECC_INJ_EN.ECC_ERROR:
  - LOW (1'b0): 挿入された ECC が回復不可能なエラーを引き起こさない場合 (ECC エラー ≦ 1)
  - HIGH (1'b1): 挿入された ECC が回復不可能なエラーを引き起こした場合 (ECC エラー ≧ 2)。AHB 読み出しトランザクションがバスエラーになります。フォールト/割込みがトリガーされることはありません。

ECC インジェクションは MPC 保護の対象となります。各 PC は、割り当てられたアドレスブロック/範囲にのみ ECC パリティの挿入が許されます。

ECC 挿入を無効にするためには、ECC_INJ_PC で指定された PC が ECC_INJ_EN をクリアします (ECC_INJ_ENABLE ビットと ECC_ERROR ビットをリセット)。PC0 はいつでも ECC_INJ_EN もクリアできます。
ECC_INJ_EN がクリアされている間、FLASHC ハードウェアは ECC_INJ_CTL を自律的にクリアします。

5.6 セキュリティと保護

FLASHC の 128 ビット AHB5 ホスト インターフェースは MPC で保護されています。この MPC は、Arm® SIE-200 MPC と同様に、セキュア/非セキュア ブロックベースの保護を実装しています。しかし、各保護コンテキスト (PC) に対してもブロック ベースの保護を提供するように機能が拡張されています。
MPC の詳細については、メモリ保護コントローラー (MPC)を参照してください。

***注**: フラッシュへの書き込みはセキュアなアプリケーションからのみ可能です。*

***注**: FLASHC API はセキュアなアプリケーションからのみ呼び出せます。*

5.7 API ライブラリ

ペリフェラル ドライバ ライブラリ (PDL) は、低消費電力モード設定、FM セクタ書き込み保護、Read While Write、FM 上での読み出し/書き込み操作へのユーザー アクセスを提供します。詳細については、Peripheral Driver Library API Reference Manual を参照してください。

6 SRAM コントローラー

ここでは、PSOC™ Control C3 MCU の SRAM コントローラーとその機能、アーキテクチャ、待機状態について説明します。SRAM コントローラーは、CPU が PSOC™ Control C3 マイコンの SRAM の一部を読み出し/書き込みできるようにします。

PSOC™ Control C3 MCU では、SRAM のサイズは 64 KB です。SRAM インスタンスとその開始アドレスの詳細については、CPU サブシステムの アドレスマップとメモリマップ セクションを参照してください。

6.1 特長

SRAM コントローラーには以下の特長があります。

- 8、16、32 ビットアクセスに対応
- Arm® SIE-200 から排他的アクセス モニター (EAM) を使用して、排他的アクセスに対応
- MPC (メモリ保護コントローラー) を使用した SRAM メモリ保護
- Arm® PPU (パワー ポリシー ユニット) を使用した SRAM 電力モード制御
- SRAM コントローラーは、SRAM のエラー訂正コード (ECC) に対応

仕様の詳細についてはデバイス データシートを参照してください。

6.2 アーキテクチャ

SRAM コントローラーは、AHB5 インフラストラクチャーに接続する AHB5 インターフェースを持っています。AHB5 転送は、すべての SRAM アクセスの起点となります。SRAM コントローラーは、AHB5 転送を以下の 3 種類に区別します。

- AHB5 読み出し転送
- 32 ビット AHB5 書き込み転送
- 8 ビットおよび 16 ビットの AHB5 書き込み転送 (partial AHB5 write transfer とも呼ばれます)

6.2.1 AHB5 読み出し転送

AHB5 は読み出し転送を行い、読み出し転送は SRAM 読み出しアクセスに変換されます。

6.2.2 32 ビット AHB5 書き込み転送

32 ビット AHB5 書き込み転送は、SRAM 書き込みアクセスに変換されます。

6.2.3 AHB5 部分書き込み転送

部分書き込み (8 ビットおよび 16 ビット) は、書き込み時に関連するマスクを適用することにより、読み出し、変更、書き込みを通じてサポートされます。

6.2.4 電源パーティション制御

PWR_MACRO_CTL レジスタは、各 SRAM 電源パーティションの電源状態を独立に制御します。PWR_MACRO_CTL.OFF[31:0]の各ビットは、各 SRAM 電源パーティションの個別状態を表します。特定の電源パーティションに対応するビットを有効にすると、パーティションはオフになり、無効にすると、それぞれのパーティションはオンになります。PWR_MACRO_CTL レジスタの内容が変更された後、STATUS.PWR_DONE が 1 になり、SRAM モードの移行が正常に完了することを確認することが必須です。各電源パーティションには、(RAM[ ]_MACRO_NR/RAM[ ]_PWR_GROUP_NR) 数のベース SRAM ユニットが含まれます。PSOC™ Control C3

MCU では、RAMC0_MACRO_NR = 8 および RAMC0_PWR_GROUP_NR = 8 により、各電源パーティションに 1 マクロを持つ 8 つの電源パーティションが作成されます。

PWR_MACRO_CTL_LOCK レジスタは、この (PWR_MACRO_CTL_LOCK[1:0]) フィールドがゼロでない時、読み出し/書き込みアクセスを禁止することによって、PWR_MACRO_CTL レジスタへの偶発的な書き込みを制御します。

注: OFF になったパーティションはデータ内容を失います。DEEPSLEEP-OFF、DEEPSLEEP-RAM、DEEPSLEEP システム電力モードでは、このパーティションも同様です。オンになっているパーティションは、アクティブ電力モードで動作可能です。このパーティションは、DEEPSLEEP-RAM および DEEPSLEEP システム電力モードではデータ内容を保持し、DEEPSLEEP-OFF システム電力モードでは内容を失います。

6.3 エラー訂正符号 (ECC)

SRAM コントローラーは、SRAM のエラー訂正符号 (ECC) に対応しています。

7 ビットの SECDED パリティは 32 ビットのデータワードを 1 つのエンティティとしてカバーし、データ内の 4 つの 8 ビットバイトには専用のパリティはありません。したがって、AHB5 の 8 ビットまたは 16 ビットの部分書き込みバス転送には、SRAM 読み出しアクセスと SRAM 書き込みアクセスが必要です。

- 32 ビット AHB5 書き込みバス転送では、SRAM 書き込みアクセスが 1 回で済みます。
- 8 ビットおよび 16 ビットの AHB5 書き込みバス転送では、不足するデータ バイトを取得するために、SRAM 書き込みアクセスの前に SRAM 読み出しアクセスを追加する必要があります。

1 ビットより多いエラーである回復不可能なエラーが検出された場合、AHB5 バス転送の結果として使用されるデータ ワードは不正ですが、AHB5 バス エラーは発生しません。その代わり、エラーは CPUSS のフォールトレポート インフラストラクチャーを通じて伝えられます。

6.3.1 ECC エラー インジェクション

エラー インジェクション メカニズムは、ECC_MATCH で定義されたアドレスに 32 ビット データ ワードを書き込むことでサポートされます。ECC_MATCH.WORD_ADDR で定義されているアドレスからの読み出しは、ECC_CTL.PARITY の値が予想される ECC 値と比較して 1、2、またはそれ以上のエラーがある場合に、修正可能または修正不能なフォールト表示をトリガーします。

6.3.2 ECC エラー報告機能

ECC シンドローム ロジックは、回復可能なエラーと回復不可能なエラーを CPUSS フォールト レポート インフラストラクチャーに報告します。ECC シンドローム ロジックは回復可能なエラーを訂正し、1 ビットまでのエラーは回復可能です。

6.3.2.1 AHB5 部分書き込み転送

SRAM 読み出しを含む部分書き込み転送中に報告されたエラーは以下のとおりです。

- **訂正可能エラー (1 ビットエラー):** 訂正されたこの 32 ビットのデータ ワードは、適切なマスクを持つ書き込みデータとマージされ、書き込みバッファに格納されます。将来の書き込みバッファ リクエストにより、マージされた書き込みデータで SRAM 書き込みアクセスが行われます。
- **マルチビットエラー:** エラーは CPUSS フォールト レポート インフラストラクチャーを通じて伝達されます。

6.3.2.2 AHB5 読み出し転送

AHB 読み出し転送中、以下のようなシナリオが考えられます。

- **訂正可能エラー (1 ビットエラー):** 修正されたデータは AHB5 インターフェースに戻されます。ECC 自動訂正機能が有効な場合、訂正されたデータも書き込みバッファに格納されます。将来の書き込みバッファ リクエストにより、修正されたデータで SRAM 書き込みアクセスが行われます。
- **マルチビットエラー:** 破損データ (マルチビットエラーデータ) が hrdata[31:0]で AHB マスターに返されます。

6.3.2.3 訂正可能な ECC エラー報告

1 ビット訂正可能な ECC エラーに対して自動訂正機能が有効な場合、部分書き込み時の訂正データおよび結合データは、対応する AHB アドレスと共に WRITE-BUFFER に格納されます。この AHB アドレスのデータが WRITE_BUFFER から SRAM マクロへ書き戻されるまで、この AHB アドレスへの今後のアクセスはすべて、データが WRITE_BUFFER から読み出される結果となります。

6.3.2.4 訂正不可能な ECC エラー報告

回復不可能なエラーが検出された場合、このエラーはフォールト レポート構造を通じてのみ報告され、AHB バス エラーは発生しません。さらに、AHB5 バス転送の結果として使用されるデータ ワードは正しくありません。

6.4 待機状態

AHB5 リクエストしかないため、待機状態はありません。ECC が有効な場合、読み出し動作には最大 3 つの待機状態があります。

7 割込み

7.1 特長

PSOC™ Control C3 は以下の割込み機能に対応します。

- 140 のシステム割込みに対応
  - 4 つの割込みを CPU のノンマスカブル割込み (NMI) にマッピングできます。
- すべての割込みソースは、CPU スリープ電力モードから CPU をウェイクアップできます。
  - 30 の割込みソースが、Cortex®-M33 を CPU ディープスリープ電力モードからウェイクアップできます。
- CPU コアに統合されたネスト型ベクタ割込みコントローラー (NVIC) により、低割込みレイテンシを実現
- CPU のディープスリープ電力モードで割込み検出 (CPU ウェイクアップ) を可能にするウェイクアップ割込みコントローラー (WIC)
- ベクタ テーブルは、揮発性メモリまたは不揮発性メモリに置けます。
- Cortex®-M33 のセキュア状態と非セキュア状態用の別々のベクタ テーブル。
- 各割込みに対して設定可能な優先度レベル、Cortex®-M33 では 8 レベル
- レベルトリガー方式の割込み信号

7.2 アーキテクチャ

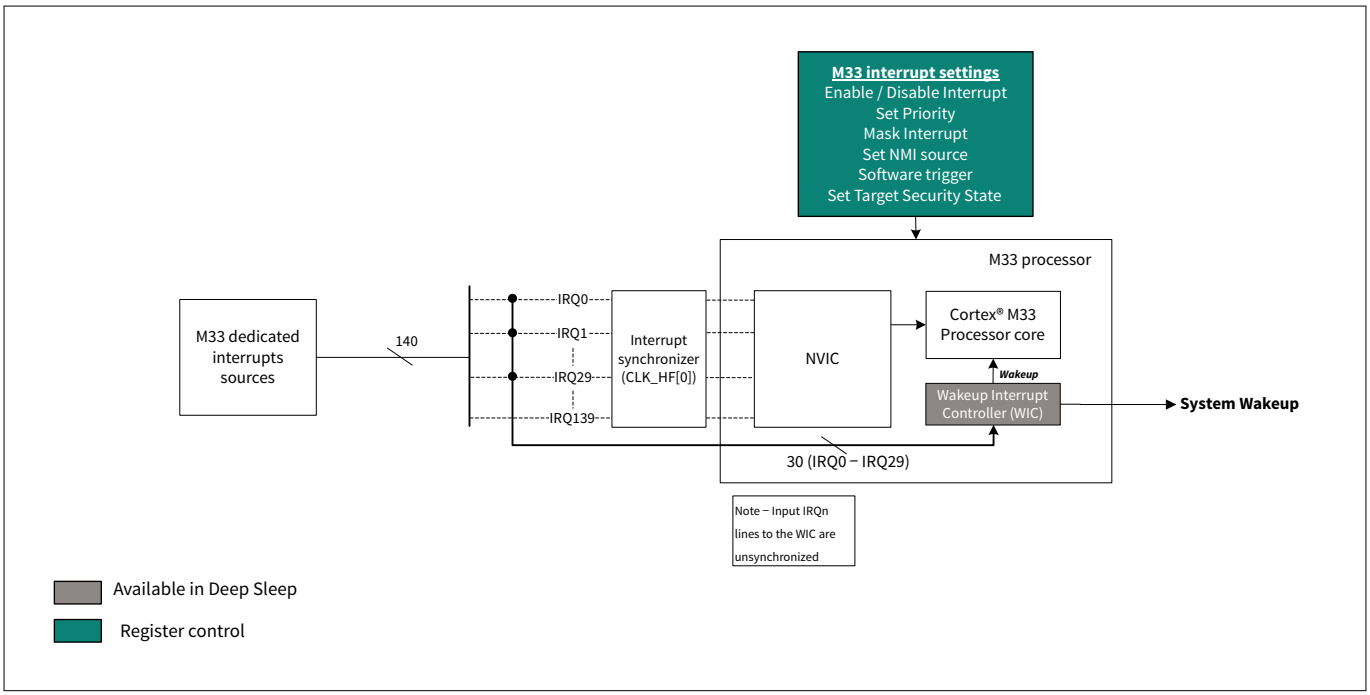


図 14 PSOC™ Control C3 割込みアーキテクチャ ブロック図

図 14 に PSOC™ Control C3 の割込みアーキテクチャを示します。Cortex®-M33 には、NVIC の IRQn に直接接続されたシステム割込みソースがあります。NVIC は、個々の割込み IRQ の有効化/無効化、優先順位解決、CPU コアとの通信を処理します。その他の例外 (NMI やハードフォールトなど) は、CPU 外部ペリフェラルによって生成される割込みとは異なり、CPU コアによって生成されるイベントの一部のため、図 14 には表示されません。

NVIC に加えて、PSOC™ Control C3 は CPU 用のウェイクアップ割込みコントローラー (WIC) と、割込みを CPU の CLK_HF0 ドメインに同期させる割込み同期ブロックに対応しています (同期用に CLK_HF0 サイクルの遅延を 2 回追加)。WIC は、CPU のディープスリープ電力モードでディープスリープ割込みを検出します。

7.3 割込みと例外操作

7.3.1 割込み/例外処理

割込みまたは例外イベントがトリガーされると、以下の一連のイベントが発生します。

- 前提は、すべての割込み信号と例外信号が最初は Low (アイドルまたは非アクティブ状態) で、プロセッサがメインコードを実行することです。アサートされたままの割込み信号は、割込みまたは例外が CPU によって処理されるように有効になっていれば、NVIC によって登録されます。信号は保留状態となり、CPU によって処理されるのを待ちます。
- NVIC からの信号を検出すると、CPU は CPU レジスタの内容をスタックにプッシュして現在のコンテキストを保存します。
- CPU はまた、NVIC からトリガーされた割込みの例外番号を受け取ります。すべての割込みおよび例外には、個別の例外番号が与えられています。この例外番号を使用することで CPU は具体的な例外ハンドラ アドレスをベクタ テーブルからフェッチします。各割込みは、セキュアまたは非セキュア ベクタ テーブルを対象とするように設定できます。
- CPU はこのアドレスへ分岐し、あとに続く例外ハンドラを実行します。
- 例外ハンドラが終了すると、CPU レジスタはスタックからポップし、元の状態に復元されます。CPU はメインコードの実行を再開します。

NVIC は、他の割込み処理を実行中に割込み要求を受信するか、複数の割込みを同時に受信した場合、これらすべての割込みの優先度を評価し、最も高い優先度を持つ割込みの例外番号を CPU に通知します。そのため、優先度の高い割込みは、いつでも優先度の低い ISR の実行をブロックできます。Armv8-M の CPU はテールチェーンにも対応しています。つまり、例外ハンドラの完了時に、次の保留中の例外がスタックされているすべての例外よりも優先順位が高い場合、スタックポップ操作はスキップされ、制御は新しい例外ハンドラに移されます。例外は割込みと同じように処理されます。各例外イベントには一意の例外番号があり、CPU が適切な例外ハンドラを実行するために使用されます。

セキュリティは重要です。セキュアなコードの実行中は、非セキュア割込みをマスクすることを推奨します。これは、セキュアなコードの実行中に非セキュア割込みがトリガーされた場合、非セキュア割込みがセキュアなコードの保護コンテキストを継承してしまう可能性があるためです。

7.3.2 レベル割込み

PSOC™ Control C3 は割込みライン (IRQn) 上のレベル信号のみに対応します。

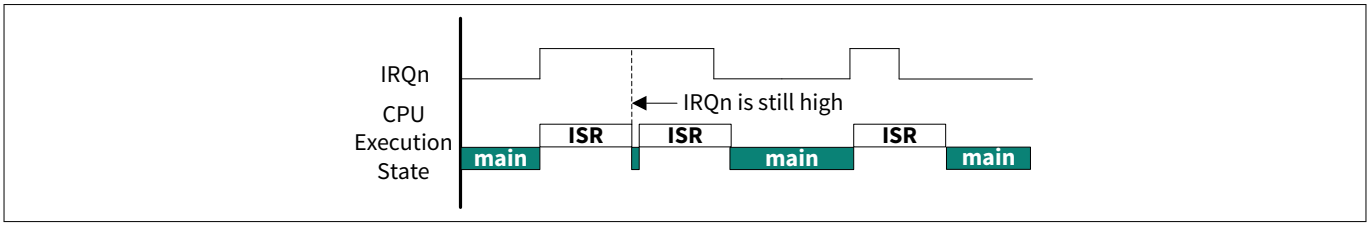


図 15 レベル割込み

注: パルス割込みはこのデバイスではサポートされていません。

図 15 に、レベル割込みがどのように処理されるかを示します。割込み信号が初期状態では非アクティブ (ロジック Low) であると仮定すると、レベ ルの処理は以下の一連のフローになります。

- レベル割込みの場合、割込み信号は NVIC が割込み要求を登録するまでアサートされたままです。これは割込みが保留状態にあり、CPU が割込み要求を処理していないことを意味します。

- その後、NVIC は例外番号を割込み要求信号と共に CPU に送信します。CPU が ISR を実行し始めると、割込みの保留状態はクリアされます。
- ISR 完了後も割込み信号が High のままであれば、保留となり、再度 ISR が実行されます。図 15 はレベルトリガー割込みの場合を示しており、割込み信号がハイである限り ISR は実行されます。

7.3.3 例外ベクタ テーブル

PSOC™ Control C3 は、Cortex®-M33 セキュアステートと Cortex®-M33 非セキュア状態のすべての例外ハンドラのエントリーポイントアドレスを格納する 2 つの例外ベクタ テーブルを持っています。CPU は例外番号に基づいて適切なアドレスをフェッチします。

表 12 Cortex®-M33 セキュア例外ベクタ

例外番号	セキュア例外	例外優先度	ベクタ アドレス
-	初期スタック ポインタの値	-	Start_Address =CM33_S_VECTOR_TABLE _BASE または CM33 の VTOR_S
1	リセット	-4	スタートアドレス + 0x0004
2	セキュア ノンマスカブル割込み (NMI_S)	-2	スタートアドレス + 0x0008
3	セキュア HardFault	(-1 または-3)	スタートアドレス + 0x000C
4	セキュア メモリ管理	設定可能 (0～7)	スタートアドレス + 0x0010
5	セキュアバス フォールト	設定可能 (0～7)	スタートアドレス + 0x0014
6	セキュア Usage Fault	設定可能 (0～7)	スタートアドレス + 0x0018
7	SecureFault	設定可能 (0～7)	スタートアドレス + 0x001C
8～10	予約済み	-	...
11	セキュア監視呼び出し (SVCall_S)	設定可能 (0～7)	スタートアドレス + 0x002C
12	セキュア デバッグ モニター	設定可能 (0～7)	スタートアドレス + 0x0030
13	予約済み	-	...
14	セキュア保留スーパーバイザリー (PendSV_S)	設定可能 (0～7)	スタートアドレス + 0x0038
15	セキュア Sys Tick タイマー (SysTick_S)	設定可能 (0～7)	スタートアドレス + 0x003C
16	セキュア IRQ(0)	設定可能 (0～7)	スタートアドレス + 0x0040
...	...	...	...
154	セキュア IRQ(138)	設定可能 (0～7)	スタートアドレス + 0x0228
155	セキュア IRQ (139)	設定可能 (0～7)	スタートアドレス + 0x022C

注: スタートアドレスは、リセット時に CM33_S_VECTOR_TABLE_BASE となり、CM33 の Armv8-M VTOR_S レジスタを更新することにより、後でファームウェアによって変更できます。
セキュア HardFault は、Armv8-M AIRCR レジスタの BFHFNMINS ビットに基づいて優先順位を変更できます。

表 13 Cortex®-M33 非セキュア例外ベクタ

例外番号	セキュア例外	例外優先度	ベクタ アドレス
-	初期スタック ポインタの値	-	開始アドレス =CM33_S_VECTOR_TABLE_BASE または CM33 の VTOR_NS
1	リセット	-4	スタートアドレス + 0x0004
2	ノンマスカブル割込み (NMI_NS)	-2	スタートアドレス + 0x0008
3	HardFault	(-1 または-3)	スタートアドレス + 0x000C
4	メモリ管理	設定可能 (0～7)	スタートアドレス + 0x0010
5	バス フォールト	設定可能 (0～7)	スタートアドレス + 0x0014
6	Usage Fault	設定可能 (0～7)	スタートアドレス + 0x0018
7	SecureFault	設定可能 (0～7)	スタートアドレス + 0x001C
8～10	予約済み	-	...
11	監視呼び出し (SVCall_NS)	設定可能 (0～7)	スタートアドレス + 0x002C
12	デバッグ モニター	設定可能 (0～7)	スタートアドレス + 0x0030
13	予約済み	-	...
14	保留スーパーバイザリー (PendSV_NS)	設定可能 (0～7)	スタートアドレス + 0x0038
15	Sys Tick タイマー (SysTick_NS)	設定可能 (0～7)	スタートアドレス + 0x003C
16	IRQ(0)	設定可能 (0～7)	スタートアドレス + 0x0040
...	...	...	...
154	IRQ(138)	設定可能 (0～7)	スタートアドレス + 0x0228
155	IRQ(139)	設定可能 (0～7)	スタートアドレス + 0x022C

注: スタートアドレスは、リセット時に CM33_NS_VECTOR_TABLE_BASE となり、CM33 の Armv8-M VTOR_NS レジスタを更新することにより、後でファームウェアによって変更できます。

各ベクタ テーブルの最初のワード (4 バイト) は、例外番号ゼロとしてマークされません。例外テーブルの最初のワードは、メイン スタック ポインタ (MSP) の値をデバイスリ セットで初期化するために使用されるためです。例外とみなされません。Armv8-M システム コントロール ブロック レジスタ (SCB) の一部として存在するベクタ テーブル オフセット レジスタ (VTOR) は、ベース アドレス (0x0000) からのベクタ テーブル オフセットを設定します。VTOR 値は、揮発性メモリおよび不揮発性メモリに置くことができるベクタ テーブルの位置を決定します。VTOR レジスタは特権 CPU モードでのみ更新可能であることに注意してください。SRAM にベクタ テーブルを移動させる利点は、SRAM ベクタ テーブルの内容を変更することにより例外ハンドラのアドレスを動的に変更できることです。しかし不揮発性フラッシュ メモリのベクタ テーブルを、不揮発性フラッシュ メモリ書き込みによって変更する必要が

あります。Cortex®-M33 では、例外テーブルは 1024 バイト アラインでなければいけないことに注意してください。例外ソース (例外番号 1～15) については 例外ソース セクションで説明しています。予約済みとしてマークされている例外は、使用されてはいませんが、ベクタ テーブルに予約アドレスを持っています。割込み発生源 (例外番号 16 以上) については、割込みソースセクションで説明しています。

7.4 例外ソース

ここでは表 12 および表 13 に記載されている (例外番号 1～15) 例外ソースについて説明します。ここでは、「バンキング」と呼ばれる TrustZone のコンセプトについても説明します。バンクについての詳細は、割込み/例外対象のセキュリティ状態とバンキングを参照してください。

7.4.1 リセット例外処理

デバイスのリセットは例外として扱われます。最も優先度の高い例外である–4 の優先度によって常に有効にされています。

デバイス リセットは、パワー オン リセット (POR)、XRES ピンの外部リセット信号、ウォッチドッグ リセットなど、複数の理由で発生することがあります。PSOC™ Control C3 の場合、デバイスリセット後、Cortex®-M33 コアが利用可能になり、セキュアブートプロセスが始まります。Cortex®-M33 セキュア リセット例外が発生します。その後、Cortex®-M33 リセットハンドラが Cortex®-M33 ブートコードを開始します。Cortex®-M33 のブートコードが正常に完了すると、Cortex®-M33 はユーザー アプリケーション コードに入ります。

7.4.2 ノンマスカブル割込み例外

ノンマスカブル割込み (NMI) は常に有効で、優先順位は固定で-2 です。

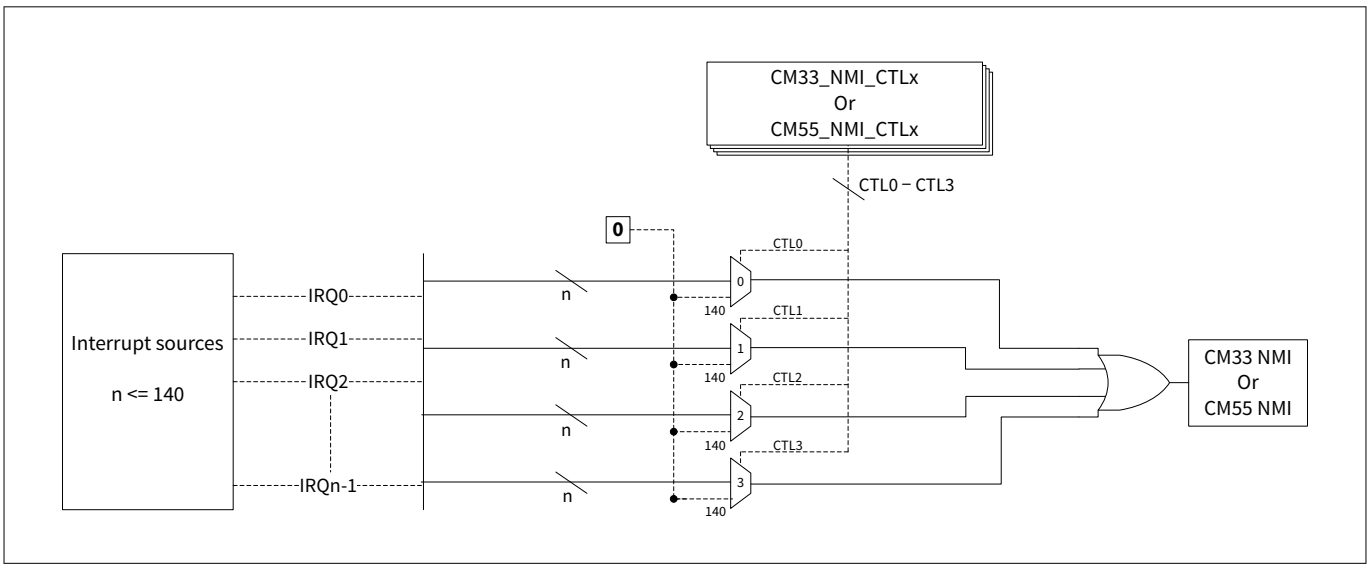


図 16 NMI トリガー

Cortex®-M33 は、CPU がアクセスできる割込み (IRQ0-IRQn) のうち最大 4 つから NMI 例外をトリガーするオプションを提供します。割込みによってトリガーされた NMI 例外は、アクティブな例外ベクタ テーブルが指す NMI ハンドラを実行します。CM33_NMI_CTLx レジスタは、ハードウェアから NMI をトリガーする割込みソースを選択します。つまり、レジスタが 1023 に設定されている場合、そのレジスタはいかなる割込みソースも NMI にマッピングしません。このようなレジスタは 4 つあり、それぞれが 1 つの割込みベクタを NMI にマッピングできます。NMI は、4 つの割込みのいずれかがトリガーされたときにトリガーされます。つまり、割込みは論理和 (OR) がとられます。

CPU の場合、例外はセキュリティ状態間でバンクされません。

- Armv8-M アプリケーション割込みおよびリセット制御レジスタ (AIRCR) の BFHFNMINS ビットが 0 に設定されている場合、NMI 例外はセキュア NMI ハンドラを対象とします。
- Armv8-M の AIRCR レジスタの BFHFNMINS ビットが 1 に設定されている場合、NMI 例外は非セキュア NMI ハンドラを対象とします。

7.4.3 HardFault 例外

PSOC™ Control C3 Armv8-M CPU は HardFault 例外に対応しています。HardFault は、通常または例外処理中のエラーが原因で発生する常に有効な例外です。HardFault は-1 の優先度を持ち、優先度が設定できる他の例外よりも高い優先度を持っています。HardFault 例外は、異なる種類のフォールト条件 (未定義命令の実行や無効なメモリ アドレスへのアクセスなど) のためのキャッチ オール例外です。CPU は、HardFault 例外ハンドラにフォールト ステータス情報を提供していません。ただしソフトウェアがフォールト状態から回復する能力を有している場合、ハンドラに例外からの復帰と実行の継続ができるようにしています。

CPU の場合、AIRCR レジスタの BFHFNMINS によって、例外がバンクされるかどうかが決まります。

- Armv8-M AIRCR レジスタの BFHFNMINS ビットを 0 に設定すると、優先度 -1 で Secure HardFault のみが有効になります。その後、Secure HardFault は、現在の実行を先取りできないすべてのフォールトを処理します。このシナリオでは、例外はバンクされません。
- Armv8-M AIRCR レジスタの BFHFNMINS ビットが 1 に設定されている場合、セキュアおよび非セキュア HardFaults が有効になります。セキュアな HardFault の優先度は-3 で、非セキュア HardFault の優先度は-1 です。使用される HardFault ハンドラは、フォールトがセキュアな状態と非セキュアな状態のどちらを対象としているかによって異なります。このシナリオでは、例外はバンクされます。

7.4.4 MemManage フォールト例外

MemManage フォールトは、メモリ保護関連のフォールトが原因で発生する例外です。固定メモリ保護制約は、命令とデータの両方のメモリトランザクションに対してこのフォールトを決定します。このフォールトは、常に、実行不可 (XN) メモリ領域への命令アクセスを中止するために使用されます。例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

例外は、セキュリティ状態間でバンクされます。

7.4.5 BusFault 例外

BusFault は、命令またはデータ メモリトランザクションのメモリ関連フォールトが原因で発生する例外です。これは、メモリシステム内のバスで検出されたエラーによる可能性があります。例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

CPU の場合、例外はセキュリティ状態間でバンクされません。

- ARMv8-M AIRCR レジスタの BFHFNMINS ビットが 0 に設定されている場合、BusFault 例外は Secure BusFault ハンドラを対象とします。
- ARMv8-M AIRCR レジスタの BFHFNMINS ビットが 1 に設定されている場合、BusFault 例外は Non-Secure BusFault ハンドラを対象とします。

7.4.6 UsageFault 例外

UsageFault は、命令実行に関連するフォールトが原因で発生する例外です。これには、以下のものが含まれます。

- 未定義の命令
- 不正なアンアライインド アクセス
- 命令実行時の無効な状態
- 例外リターン時のエラー

コアがこれらを報告するように設定されている場合、以下のような Usage Fault が発生することがあります。

- ワードおよびハーフワードメモリアクセスのアンアライインド アドレス
- ゼロ除算

例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

例外は、セキュリティ状態間でバンクされます。

7.4.7 SecureFault 例外

SecureFault は、TrustZone として知られるセキュリティ拡張機能を搭載した Armv8-M CPU の追加例外です。SecureFault は、実行されるさまざまなセキュリティチェックによってトリガーされる可能性があります。例えば、非セキュアコードから、有効なエントリーポイントとしてマークされていないセキュアコード内のアドレスにジャンプすると、SecureFault がトリガーされます。SecureFault を処理するときは注意し、ハンドラが不注意にセキュリティの脆弱性をもたらすことがないようにしてください。SecureFault は常にセキュアな状態をターゲットとします。例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

7.4.8 監視呼び出し (SVCall) 例外

監視呼び出し (SVCall) は CPU が SVC 命令をアプリケーション コードの一部として実行することにより発生する、常に有効な例外です。アプリケーション ソフトウェアは SVC 命令を使用して、オペレーティングシステムへの呼び出しを行い、サービスを提供させます。これは監視呼び出しとして知られています。SVC 命令はアプリケーションがシステムへの特権アクセスを要する監視コールを発行できるようにします。例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

例外は、セキュリティ状態間でバンクされます。

7.4.9 DebugMonitor 例外

Armv8-M CPU は DebugMonitor 例外に対応しています。DebugMonitor は、デバッグの停止が無効で、デバッグ監視が有効な場合にのみ使用できます。これは、Armv8-M Debug Halting Control and Status Register (DHCSR) で C_DEBUGEN ビットを disabled に設定し、Armv8-M Debug Exception and Monitor Control Register (DEMCR) で MON_EN ビットを enabled に設定することで実行できます。DebugMonitor 例外は、同じコア上で動作する専用のデバッグ監視ソフトウェアによって処理されます。デバッグイベントは、DebugMonitor 例外の優先順位が現在の例外の優先順位よりも高い場合にのみ、DebugMonitor 例外を発生させます。例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

この例外は、セキュリティ状態間ではバンクされません。Armv8-M DEMCR レジスタの Secure Debug Monitor Enable (SDME) ビットが 1 に設定されている場合、DebugMonitor はセキュアハンドラをターゲットにします。SDME が 0 に設定されている場合、DebugMonitor の例外は非セキュアハンドラを対象とします。

7.4.10 PendSupervisory (PendSV) 例外

Armv8-M の CPU は PendSV 例外に対応しています。PendSV は SVCall に類似する監視コールの例外です。通常ソフトウェアで生成されます。PendSV は常に有効で、その優先度を設定できます。例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

例外は、セキュリティ状態間でバンクされます。

7.4.11 システムティック (SysTick) 例外

SysTick は、RTOS ティック タイマー、高速アラーム タイマー、単純なカウンターなどの様々な時間維持の目的のための簡単な 24 ビット デクリメントのカウンターを提供しています。SysTick タイマーは、カウント値がゼロになったときに割込みを発生させるように設定でき、これは SysTick 例外と呼ばれます。例外の優先順位は 0 (最高) から 7 (最低) まで設定可能です。

例外は、セキュリティ状態間でバンクされます。つまり、2 つの SysTick タイマーがあり、各セキュリティ状態に 1 つずつあります。

7.5 割込みソース

PSOC™ Control C3 はペリフェラルからの 140 の割込みに対応します。表 14 に各割込みのソースを示します。これらのシステム割込みは、Cortex®-M33 NVIC に直接マッピングされます。

割込みには、TCPWM、SCB、ウォッチドッグ、ADC などのオンチップ ペリフェラルからの標準割込みが含まれます。生成された割込みは、通常は異なるペリフェラルとの接続状態との論理 OR です。ペリフェラル ステータス レジスタを ISR で読み出し、割込みを生成した条件を検出します。これらの割込みはレベル割込みです。割込みをデアサートするためには、ISR 内で適切な割込みレジスタをクリアする必要があります。通常、レジスタをクリアするためには'1'を書き込む必要があります。ステータスレジスタが ISR でクリアされない場合、割込みラインは常にアサートされた状態となり、ISR は継続的に実行されます。

Cortex®-M33 CPU では、すべての割込みはセキュリティ状態間でバンクされません。

表 14 からわかるように、30 本の割込み (IRQ0～IRQ29) が Cortex®-M33 を CPU のディープスリープ電力モードからウェイクアップさせます。すべての割込みは、CPU スリープ電力モードから CPU をウェイクアップさせられます。

表 14 PSOC™ Control C3 Cortex®-M33 割込みソースの一覧

Cortex®®-M33 の割込み	Cortex®®-M33 例外番号	CPU ディープスリープ対応	割込みソース
NMI	-2	-	以下の 140 の割込みソースのうち 4 つの論理和
IRQ0	16	有り	GPIO ポート非セキュア#0
IRQ1	17	有り	GPIO ポート非セキュア#1
IRQ2	18	有り	GPIO ポート非セキュア#2
IRQ3	19	有り	GPIO ポート非セキュア#3
IRQ4	20	有り	GPIO ポート非セキュア#4
IRQ5	21	有り	GPIO ポート非セキュア#5
IRQ6	22	有り	GPIO ポート非セキュア#6
IRQ7	23	有り	GPIO ポート非セキュア#7
IRQ8	24	有り	GPIO ポート非セキュア#8
IRQ9	25	有り	GPIO ポート非セキュア#9
IRQ10	26	有り	GPIO ポートセキュア#0
IRQ11	27	有り	GPIO ポートセキュア#1
IRQ12	28	有り	GPIO ポートセキュア#2
IRQ13	29	有り	GPIO ポートセキュア#3
IRQ14	30	有り	GPIO ポートセキュア#4
IRQ15	31	有り	GPIO ポートセキュア#5
IRQ16	32	有り	GPIO ポートセキュア#6
IRQ17	33	有り	GPIO ポートセキュア#7

(続く)

表 14 (続き) PSOC™ Control C3 Cortex®-M33 割込みソースの一覧

Cortex®®-M33 の割込み	Cortex®®-M33 例外番号	CPU ディープスリープ対応	割込みソース
IRQ18	34	有り	GPIO ポートセキュア#8
IRQ19	35	有り	GPIO ポートセキュア#9
IRQ20	36	有り	GPIO 電源検出
IRQ21	37	有り	GPIO 非セキュア - 全ポート
IRQ22	38	有り	GPIO セキュリティ、全ポート
IRQ23	39	有り	SCB#0
IRQ24	40	有り	MCWDT
IRQ25	41	有り	バックアップドメイン
IRQ26	42	有り	プロセス間通信 #0
IRQ27	43	有り	プロセス間通信 #1
IRQ28	44	有り	WDT
IRQ29	45	有り	LPCOMP
IRQ30	46	-	SRSS (LVD, CLKCAL)
IRQ31	47	-	SCB#1
IRQ32	48	-	SCB#2
IRQ33	49	-	SCB#3
IRQ34	50	-	SCB#4
IRQ35	51	-	SCB#5
IRQ36	52	-	フラッシュ マクロ
IRQ37	53	-	CPUSS DMA#0, Channel#0
IRQ38	54	-	CPUSS DMA#0, Channel#1
IRQ39	55	-	CPUSS DMA#0, Channel#2
IRQ40	56	-	CPUSS DMA#0, Channel#3
IRQ41	57	-	CPUSS DMA#0, Channel#4
IRQ42	58	-	CPUSS DMA#0, Channel#5
IRQ43	59	-	CPUSS DMA#0, Channel#6
IRQ44	60	-	CPUSS DMA#0, Channel#7
IRQ45	61	-	CPUSS DMA#0, Channel#8
IRQ46	62	-	CPUSS DMA#0, Channel#9
IRQ47	63	-	CPUSS DMA#0, Channel#10
IRQ48	64	-	CPUSS DMA#0, Channel#11
IRQ49	65	-	CPUSS DMA#0, Channel#12

(続く)

表 14 (続き) PSOC™ Control C3 Cortex®-M33 割込みソースの一覧

Cortex®®-M33 の割込み	Cortex®®-M33 例外番号	CPU ディープスリープ対応	割込みソース
IRQ50	66	-	CPUSS DMA#0, Channel#13
IRQ51	67	-	CPUSS DMA#0, Channel#14
IRQ52	68	-	CPUSS DMA#0, Channel#15
IRQ53	69	-	CPUSS DMA#1, Channel#0
IRQ54	70	-	CPUSS DMA#1, Channel#1
IRQ55	71	-	CPUSS DMA#1, Channel#2
IRQ56	72	-	CPUSS DMA#1, Channel#3
IRQ57	73	-	CPUSS DMA#1, Channel#4
IRQ58	74	-	CPUSS DMA#1, Channel#5
IRQ59	75	-	CPUSS DMA#1, Channel#6
IRQ60	76	-	CPUSS DMA#1, Channel#7
IRQ61	77	-	CPUSS DMA#1, Channel#8
IRQ62	78	-	CPUSS DMA#1, Channel#9
IRQ63	79	-	CPUSS DMA#1, Channel#10
IRQ64	80	-	CPUSS DMA#1, Channel#11
IRQ65	81	-	CPUSS DMA#1, Channel#12
IRQ66	82	-	CPUSS DMA#1, Channel#13
IRQ67	83	-	CPUSS DMA#1, Channel#14
IRQ68	84	-	CPUSS DMA#1, Channel#15
IRQ69	85	-	SRAM#0 PPU
IRQ70	86	-	Cortex®-M33 浮動小数点ユニット
IRQ71	87	-	Cortex®-M33 CTI #0
IRQ72	88	-	Cortex®-M33 CTI #1
IRQ73	89	-	CPUSS フォールト
IRQ74	90	-	CPUSS PPU
IRQ75	91	-	CPUSS マスターセキュリティコントローラー
IRQ76	92	-	TCPWM#0、グループ 0、カウンター#0
IRQ77	93	-	TCPWM#0、グループ 0、カウンター#1
IRQ78	94	-	TCPWM#0、グループ 0、カウンター#2
IRQ79	95	-	TCPWM#0、グループ 0、カウンター#3
IRQ80	96	-	TCPWM#0、グループ 1、カウンター#256
IRQ81	97	-	TCPWM#0、グループ 1、カウンター#257

(続く)

表 14 (続き) PSOC™ Control C3 Cortex®-M33 割込みソースの一覧

Cortex®®-M33 の割込み	Cortex®®-M33 例外番号	CPU ディープスリープ対応	割込みソース
IRQ82	98	-	TCPWM#0、グループ 1、カウンター#258
IRQ83	99	-	TCPWM#0、グループ 1、カウンター#259
IRQ84	100	-	TCPWM#0、グループ 1、カウンター#260
IRQ85	101	-	TCPWM#0、グループ 1、カウンター#261
IRQ86	102	-	TCPWM#0、グループ 1、カウンター#262
IRQ87	103	-	TCPWM#0、グループ 1、カウンター#263
IRQ88	104	-	TCPWM#0、グループ 2、カウンター#512
IRQ89	105	-	TCPWM#0、グループ 2、カウンター#513
IRQ90	106	-	TCPWM#0、グループ 2、カウンター#514
IRQ91	107	-	TCPWM#0、グループ 2、カウンター#515
IRQ92	108	-	TCPWM#0、グループ 2、カウンター#516
IRQ93	109	-	TCPWM#0、グループ 2、カウンター#517
IRQ94	110	-	TCPWM#0、グループ 2、カウンター#518
IRQ95	111	-	TCPWM#0、グループ 2、カウンター#519
IRQ96	112	-	SRSS メイン PPU
IRQ97	113	-	Crypto
IRQ98	114	-	TRNG
IRQ99	115	-	CAN#0, Interrupt#0, Channel#0
IRQ100	116	-	CAN#0, Interrupt#1, Channel#0
IRQ101	117	-	CAN#0, Interrupt#0, Channel#1
IRQ102	118	-	CAN#0, Interrupt#1, Channel#1
IRQ103	119	-	CAN#0、固定割込み#0
IRQ104	120	-	CAN#0、固定割込み#1
IRQ105	121	-	CORDIC
IRQ106	122	-	TCPWM #0, MOTIF #1 SR0
IRQ107	123	-	TCPWM #0, MOTIF #1 SR1
IRQ108	124	-	HPPASS (AC、エラー条件) の組み合わせ
IRQ109	125	-	SAR エントリーと FIR 結果の組み合わせ
IRQ110	126	-	個別 SAR エントリー結果 [0]
IRQ111	127	-	個別 SAR エントリー結果 [1]
IRQ112	128	-	個別 SAR エントリー結果 [2]
IRQ113	129	-	個別 SAR エントリー結果 [3]

(続く)

表 14 (続き) PSOC™ Control C3 Cortex®-M33 割込みソースの一覧

Cortex®®-M33 の割込み	Cortex®®-M33 例外番号	CPU ディープスリープ対応	割込みソース
IRQ114	130	-	個別 SAR エントリー結果 [4]
IRQ115	131	-	個別 SAR エントリー結果 [5]
IRQ116	132	-	個別 SAR エントリー結果 [6]
IRQ117	133	-	個別 SAR エントリー結果 [7]
IRQ118	134	-	FIR 結果 [0]
IRQ119	135	-	FIR 結果 [1]
IRQ120	136	-	SAR レンジの組み合わせ
IRQ121	137	-	個別 SAR 範囲 [0]
IRQ122	138	-	個別 SAR 範囲 [1]
IRQ123	139	-	個別 SAR 範囲 [2]
IRQ124	140	-	個別 SAR 範囲 [3]
IRQ125	141	-	個別 SAR 範囲 [4]
IRQ126	142	-	個別 SAR 範囲 [5]
IRQ127	143	-	個別 SAR 範囲 [6]
IRQ128	144	-	個別 SAR 範囲 [7]
IRQ129	145	-	CSG DAC の組み合わせ
IRQ130	146	-	個別 CSG DAC [0]
IRQ131	147	-	個別 CSG DAC [1]
IRQ132	148	-	個別 CSG DAC [2]
IRQ133	149	-	個別 CSG DAC [3]
IRQ134	150	-	個別 CSG DAC [4]
IRQ135	151	-	予約済み
IRQ136	152	-	予約済み
IRQ137	153	-	予約済み
IRQ138	154	-	CSG CMP の組み合わせ
IRQ139	155	-	FIFO の組み合わせ

7.6 割込み/例外対象のセキュリティ状態とバンキング

Cortex®-M33 CPU では、例外と割込みはセキュアまたは非セキュア ISR ハンドラをターゲットにできます。例外がバンクされていると言われる場合、これは例外のリソースが重複しており、セキュアバージョンと非セキュアバージョンの両方があることを意味します。例外がバンクされていないと言われる場合、これは例外がセキュアまたは非セキュアハンドラのみをターゲットにするように設定できることを意味します。

以下のものはバンク例外に含まれます。

- バンク有効ビット、保留ビット、アクティブ ビット

- バンクされた SHPRn.PRI フィールド
- バンクされた例外ベクタ
- 各セキュリティ状態に対応するハンドラ

表 15　　バンク例外

例外	バンク
リセット	無し
HardFault	有り
NMI	無し
MemManagment Fault	有り
BusFault	無し
UsageFault	有り
SecureFault	無し
SVCall	有り
DebugMonitor	無し
PendSV	有り
SysTick	有り
IRQ(0)	無し
…	…
IRQ(n)	無し

***注**:　　HardFault は、BFHFNMINS ビットが 1 に設定されているときのバンク例外です。BFHFNMINS ビットが 0 にセットされると、すべての HardFaults はセキュアハンドラに送られます。*

BusFault、NMI、および HardFault は、Armv8-M Application Interrupt and Reset Control Register (AIRCR) の BFHFNMINS ビットを使用して設定できます。BFHFNMINS が 0 に設定されると、BusFault、NMI、および HardFault はそれぞれのセキュア ハンドラをターゲットにします。BFHFNMIS が 1 に設定されると、BusFault と NMI はそれぞれの非セキュア ハンドラを対象とし、HardFault はセキュア ハンドラと非セキュア ハンドラの両方を持つバンク例外となります。

SecureFault は常にセキュアハンドラを対象とします。

Armv8-M DEMCR レジスタの Secure Debug Monitor Enable (SDME) ビットが 1 に設定されている場合、DebugMonitor はセキュアハンドラをターゲットにします。SDME が 0 に設定されている場合、DebugMonitor の例外は非セキュアハンドラを対象とします。

7.6.1　　割込み対象のセキュリティ状態

Cortex®-M33 CPU では、各割込みはセキュアまたは非セキュア状態をターゲットに設定できます。その結果、割込みは、セキュア ベクタ テーブルの割込みサービス ルーチン (ISR) または非セキュア ベクタ テーブルの ISR のいずれかをサービスするように設定できます。Armv8-M アーキテクチャでは、割込みが使用する特定のベクタ テーブルを設定するに使用される 16 の割込みターゲット非セキュアレジスタ (ITNSn) があります。各 ITNSn レジスタは 32 ビットです。ここで、m はレジスタのビット番号、n はレジスタのインスタンスです。ビット m を 0 に設定するとセキュア状態の割込みターゲットが設定され、ビット m を 1 に設定すると非セキュア状態の割込みターゲットが設定されます。

7.6.2 GPIO のセキュア割込みと非セキュア割込み

CPU については、各 GPIO ピンをセキュアまたは非セキュアに設定できます。1 つの GPIO ポートには、セキュア ピンと非セキュア ピンを組み合わせられます。その結果、各 GPIO ポートに対して、セキュア GPIO 割込み用と非セキュア GPIO 割込み用の 2 つの割込みソースが NVIC に入ることになります。表 14 に、これら 2 つの割込みソースを示します。ピンのセキュリティ状態を設定すると、トリガーされる GPIO ポートの割込みソースも設定されます。その結果、セキュア GPIO ピンはセキュア GPIO ポート割込みのみをトリガーでき、非セキュア GPIO ピンは非セキュア GPIO ポート割込みのみをトリガーできます。

7.7 割込み/例外の優先順位

CPU によって処理される必要がある複数の例外がある時、例外の優先度は例外アービトレーションに有用です。リセット、NMI および HardFault 以外のすべての例外に優先度を割り当てられます。リセット、NMI および HardFault 例外はそれぞれ–3、–4、–2 の固定された優先度を持っています。CPU の HardFaults の優先順位は-1 および-3 です。TrustZone を使用した HardFault 構成の詳細については、HardFault 例外セクションを参照してください。優先度番号が小さい方がより高い優先度を表します。よってリセット、NMI、HardFault の例外は最上位優先度になります。他の例外は優先度 0～7 を割り当てられます。

Armv8-M の CPU はネストされた例外に対応しており、優先順位の高い例外が現在アクティブな例外ハンドラを妨害 (割込み) できます。後から発生した例外の優先度がアクティブな例外と同じであれば、このプリエンプションは発生しません。CPU は優先度の高い例外を処理した後、優先度の低い例外ハンドラの実行を再開します。8 つの優先レベルがサポートされているため、最大 8 つのネストされた例外が可能です。CPU は同じ優先度の複数の例外要求を受信した場合、最小の例外番号のものが最初に処理されます。

例外番号 4～15 の優先順位は、3 つのシステム ハンドラ優先順位レジスタ (SHPR1、SHPR2、SHPR3) に書き込むことで設定できます。各 SHPRn には、4 つの例外の優先順位 PRI_N が格納されます。各優先順位番号は例外番号に対応しています。SHPR1 には、例外 PRI_4～PRI_7 の優先度が格納されます。SHPR2 には、例外 PRI_8～PRI_11 の優先度が格納されます。SHPR3 には、例外 PRI_12～PRI_15 の優先度が格納されます。

割込みの優先度は、対応する割込み優先度レジスタ (NVIC_IPRn) への書き込みによって設定できます。表 16 で示すように、これは 124 の 32 ビットレジスタのグループで、レジスタ 1 つは 4 つの割込みの優先度を格納します。各フィールドに対応する割込み番号は 4n + m で表され、m はレジスタのフィールド番号、n はレジスタのインスタンスです。

表 16 NVIC_IPRn の割込み優先度レジスタ ビット定義

ビット	名称	説明
7:5	PRI_N0	割込み番号 4N + 0 の優先度
15:13	PRI_N1	割込み番号 4N + 1 の優先度
23:21	PRI_N2	割込み番号 4N + 2 の優先度
31:29	PRI_N3	割込み番号 4N + 3 の優先度

7.8 割込みのイネーブルとディセーブル

Armv8-M CPU の NVIC は、ソフトウェアで割込みを個別に有効/無効にするレジスタを提供します。割込みが有効になっていない場合、NVIC はその割込みライン上の割込み要求を処理しません。Armv8-M の割込みセット イネーブル レジスタ (ISERn) と割込みクリア イネーブル レジスタ (ICERn) はそれぞれの割込みを有効/無効にするために使用されます。Armv8-M アーキテクチャには、32 ビット ISER/ICER レジスタのインスタンスが 16 あります。各ビットに対応する割込み番号は 32n+m で表され、m はレジスタのビット番号、n はレジスタのインスタンスです。これらのレジスタは、割込みのイネーブル ステータスを確認するためにソフトウェアで読み出すこともできます。割込み/例外の有効/無効は、これら 2 つのレジスタのレジスタ アクセス プロパティを示します。これらのレジスタにゼロを書き込んでも影響がないことに注意してください。CPU の場合、ISER と ICER はセキュリティ状態間でバンクされません。

表 17 割込みイネーブル/ディセーブル レジスタ

レジスタ	動作	ビット値	注記
割込みセット イネーブル レジスタ (ISERn)	書き込み	1	割込みを有効
		0	無効 (No effect)
	読み出し	1	割込みが有効
		0	割込みが無効
	書き込み	1	割込みを無効
		0	無効 (No effect)
	読み出し	1	割込みが有効
		0	割込みが無効

ISER および ICER レジスタは割込みにのみ適用されます。これらのレジスタは例外番号 1～15 を有効にするためには利用できません。

Armv8-M SHCSR レジスタは、SecureFault、UsageFault、BusFault、MemFault の有効/無効を設定できます。SHCSR レジスタには、SECUREFAULTENA、USGFAULTENA、BUSFAULTENA、MEMFAULTENA ビットがあり、これらのビットを設定すると例外が有効になり、クリアすると例外が無効になります。CPU の場合、レジスタはビット単位でセキュリティ状態の間でバンクされます。SECUREFAULTENA ビットと BUSFAULTENA ビットはセキュリティ状態間でバンクされません。USGFAULTENA ビットと MEMFAULTENA ビットはセキュリティ状態間でバンクされます。

SysTick は、Armv8-M SysTick Control and Status Register (SYST_CSR) の ENABLE ビットで設定できます。CPU の場合、このレジスタはセキュリティ状態間でバンクされます。

DebugMonitor は、Armv8-M の DEMCR (Debug Exception and Monitor Control Register) の MON_EN ビットで有効にできます。DebugMonitor が機能するためには、ARMv8-M の DHCSR (Debug Halting Control and Status Register) で C_DEBUGEN ビットを無効にする必要があることに注意してください。CPU の場合、レジスタはセキュリティ状態間でバンクされません。

HardFault、NMI、SVCall、PendSV は常に有効であるため、例外有効ビットはないことに注意してください。

Armv8-M の特殊用途マスクレジスタ (PRIMASK、FAULTMASK、BASEPRI) を使用して、割込みと例外を無効にすることもできます。有効にされているかどうかに関わらず、PRIMASK レジスタは、優先度を設定できる例外をマスクするためのグローバル有効レジスタとして使用されます。優先度を設定できる例外は表 14 に記載されているリセット、NMI、HardFault 以外のすべての例外を含みます。FAULTMASK レジスタは、NMI 例外を除くすべてのフォールトと割込みを無効にするために使用できます。BASEPRI レジスタは、優先度レベルに基づいて割込みを禁止するために使用できます。BASEPRI レジスタの値よりも優先度が低い (高優先度) 割込みのみがトリガーできます。PRIMASK、FAULTMASK、または BASEPRI が設定されている場合、マスクされた例外のいずれも CPU は処理できません。マスクされた例外は、マスクがクリアされるまで保留状態のままです。CPU の場合、レジスタはセキュリティ状態の間でバンク化されます。

7.9 割込み/例外状態

各例外は次のいずれかの状態にあります。

表 18 割込み/例外状態

例外状態	意味
非アクティブ	例外は非アクティブで、保留中ではありません例外が無効になっているか、有効な例外がトリガーされていません

(続く)

表 18 (続き) 割込み/例外状態

例外状態	意味
保留中	例外要求が CPU/NVIC によって受信されており、例外が CPU による処理を待機中
アクティブ	CPU で処理されていますが、例外ハンドラの実行が完了していない例外です。優先度の高い例外は、優先度の低い例外の実行に割り込むことが可能。この場合両方の例外がアクティブ状態になります
アクティブおよび保留	例外がプロセッサによって処理されていますが、その例外ハンドラの実行中に、同じソースからの保留中の要求があります。

割込み制御状態レジスタ (ICSR) はさまざまな例外状態を説明するステータス ビットが含まれています。

- ICSR の VECTACTIVE ビット ([8:0]) は現在実行中の例外番号を格納します。CPU が例外ハンドラを (CPU はスレッド モードにある) を実行していない場合、この値はゼロです。VECTACTIVE ビット フィールドが、アクティブ例外番号を格納するために使用される割込みプログラム ステータス レジスタ (IPSR) のビット[8:0]の値と同じであることに注意してください。
- ICSR の VECTPENDING ビット ([20:12]) は最上位優先度の保留中の例外の例外番号を格納します。保留中の例外が存在しない場合、この値はゼロです。
- ICSR 内の ISRPENDING ビット (ビット 22) は、NVIC 生成割込みが保留状態にあるかどうかを示します。

7.9.1 保留中の割込み/例外

周辺装置が NVIC へ割込み要求信号を生成するまたは例外イベントが発生する時、対応する例外は保留状態に入ります。CPU は対応する例外ハンドラ ルーチンの実行を開始すると、例外が保留状態からアクティブ状態へ変更されます。割込みの保留状態を設定またはクリアするための異なるレジスタ ビットを提供することにより、NVIC は 20 個の割込みラインのソフトウェア保留を可能にします。割込み保留セットレジスタ (ISPRn) と割込み保留クリアレジスタ (ICPRn) は割込みラインの保留状態をセットまたはクリアするために使用されます。32 ビット ISPR/ICPR レジスタが 16 個あり、各ビットが割込みに対応します。各ビットに対応する割込み番号は 32n ＋ m で表され、m はレジスタのビット番号、n はレジスタのインスタンスです。表 19 に、この 2 つのレジスタのレジスタ アクセス プロパティを示します。これらのレジスタにゼロを書き込んでも影響がないことに注意してください。CPU の場合、ISPR と ICPR はセキュリティ状態間でバンクされません。

表 19 割込み設定保留/クリア保留レジスタ

レジスタ	動作	ビット値	注記
割込み保留セット レジスタ (ISPRn)	書き込み	1	割込みを保留状態にする
		0	無効 (No effect)
	読み出し	1	割込みが保留中
		0	割込みが保留中でない
割込み保留クリア レジスタ (ICPRn)	書き込み	1	保留中の割込みをクリア
		0	無効 (No effect)
	読み出し	1	割込みが保留中
		0	割込みが保留中でない

既にセットされているビットに保留ビットを設定しても結果は 1 つのみの ISR の実行です。対応する割込みが有効になっているかどうかに関わらず、保留ビットの更新は可能です。割込みが有効にされていない場合、割込みラインが ISER レジスタに書き込んで有効化されるまで、保留状態に移行しません。ISPR レジスタと ICPR レジス

タは、ペリフェラルの割込みにのみ使用されることに注意してください。これらのレジスタは例外番号 1～15 までを保留するためには利用できません。

例外 1～15 の保留ビットは、Armv8-M の SHCSR、ICSR、および DEMCR レジスタにあります。

SHCSR には、HARDFAULTPENDED、SECUREFAULTPENDED、SVCALLPENDED、BUSFAULTPENDED、MEMFAULTPENDED、および USGFAULTPENDED ビットがあり、これらはそれぞれ同名の例外に対応する保留ビットです。これらの例外に対する明確な保留レジスタはありません。これらのビットを書き込み/クリアすることで、例外の保留状態がセット/クリアされます。

ICSR は PENDNMISET、PENDSVSET、PENDSTSET ビットを持ち、1 つ書き込むと NMI、PendSV、SysTick 例外を保留状態に設定します。ICSR には、PENDNMICLR、PENDSVCLR、PENDSTCLR ビットもあり、これらのビットを書き込むと、NMI、PendSV、SysTick の例外がクリアされます。

DEMCR レジスタには MON_PEND ビットがあり、DebugMonitor の保留状態を設定またはクリアするために使用されます。

7.10 割込み/例外のスタック使用

CPU はメイン コード (スレッド モードで) を実行し、例外要求が発生すると、CPU は汎用レジスタの状態をスタックに格納します。次に対応する例外ハンドラの実行 (ハンドラ モード) を開始します。CPU は 8 つの 32 ビットの内部レジスタをスタックにプッシュします。これらのレジスタはプログラム ステータス レジスタ (PSR)、戻りアドレス、リンク レジスタ (LR または R14)、R12、R3、R2、R1 および R0 です。TrustZone が搭載されていない Armv8-M CPU には、メイン スタック ポインタ (MSP) およびプロセス スタック ポインタ (PSP)の 2 つのスタックポインタがあります。TrustZone が有効な Armv8-M CPU は、セキュアと非セキュアの状態を分けるために、MSP_NS、MSP_S、PSP_NS、PSP_S の 4 つのスタックポインタがあります。一度にアクティブにできるのは 1 つのスタック ポインタのみです。スレッド モードでは、制御レジスタのアクティブ スタック ポインタ ビットが現在のアクティブ スタック ポインタを定義するために使用されます。ハンドラ モードでは、MSP は常にスタック ポインタとして使用されます。スタック ポインタは常に下向きに進み、最後にプッシュされたデータを持つアドレスを示します。

CPU がスレッド モードにあり、例外要求が発生した場合、CPU は汎用レジスタの内容を格納するために制御レジスタで定義されたスタック ポインタを使用します。スタック プッシュの動作後、例外ハンドラの実行をするために、CPU はハンドラ モードに入ります。現在の例外の実行中に別の優先度の高い例外が発生すると、CPU がハンドラ モード中であるため、MSP はスタック プッシュ/ポップ動作に使用されます。詳細については CPU サブシステム (CPUSS) セクションを参照してください。

7.11 割込みと低消費電力モード

特定のペリフェラルの割込み要求が発生すると、CPU は低消費電力モードから起動します。ウェイクアップ割込みコントローラー (WIC) ブロックは、1 つまたは複数のウェイクアップ ソースが割込み信号を生成する時、ウェイクアップ信号を生成し、デバイスがアクティブ モードに移行します。アクティブ モードに入った後、周辺割込みの ISR が実行されます。

CPU で実行される割込み待機命令 (WFI) またはイベント待機命令 (WFE) は、CPU スリープモードおよび CPU ディープスリープモードへの遷移をトリガーします。WFI 命令も WFE 命令も割込みでウェイクアップできます。しかし、WFE は、CPU の例外マスクレジスタ (PRIMASK) で割込みをマスク解除する必要があります。さらに、WFE 命令は、イベントビットの状態に基づいて CPU をスリープ状態にし、通常は他の CPU から送られるイベント信号によってウェイクアップします。WFI は PRIMASK のマスク解除を必要とせず、NVIC または WIC にマスクされた保留中の割込みから CPU をウェイクアップできます。ただし、WFI は他の CPU からのイベント信号で CPU をウェイクアップさせることはできません。異なる低消費電力モードに入るシーケンスの詳細は、デバイスの電力モードを参照してください。

チップ低消費電力モードは固定機能割込みソースの 2 つの分類があります。

- CPU アクティブモード、CPU スリープモード、CPU ディープスリープモードで使用可能な割込みソース
- CPU アクティブモードと CPU スリープモードでのみ使用可能な割込みソース

ディープスリープに対応した割込みについては、割込みソースを参照してください。

WFE 命令を使用する場合、スリープ/ディープスリープモードに入ったり出たりするためには、必ず WFE 命令を 2 回呼び出してください。この動作は、Armv8-M アーキテクチャのイベントレジスタ実装に由来します。

Armv8-M アーキテクチャのリファレンス マニュアル (セクション B2.1.1) によると、以下のようになります。

- リセットはイベント レジスタをクリアします。
- WFE ウェイクアップイベント、または例外リターン命令の実行は、イベントレジスタを設定します。
- WFE 命令は、イベントレジスタをクリアします。
- ソフトウェアは、イベント レジスタの値を直接読み出し/書き込みすることはできません。

したがって、最初の WFE 命令は、CPU をスリープさせ、2 番目の WFE 命令は、WFE ウェイクアップ後にイベントレジスタをクリアし、イベントレジスタをセットします。そのため、次の WFE ではコアをスリープ状態にします。

7.12 割込み/例外 - 初期化/設定

ここでは、PSOC™ Control C3 での例外の初期化とコンフィギュレーションに関連のあるさまざまなステップを説明します。

1. 例外ベクタ テーブルの場所の設定: 例外を使用する最初のステップは、揮発性または不揮発性メモリに、必要に応じてベクタ テーブルの場所を設定することです。この設定は例外ベクタ テーブルセクションで説明したように行います。アプリケーションでベクタ アドレスを動的に変更する必要がある場合は、ベクタ テーブルを揮発性メモリに置く必要があります。テーブルがフラッシュに位置している場合には、フラッシュ書き込み動作でベクタ テーブルの内容を変更する必要があります。

   注: セキュアなアプリケーションのみがフラッシュ書き込みを実行できます。

2. 個々の例外設定: 次のステップは、先のセクションで説明したように、アプリケーションで必要とされる個々の例外を設定する方法です。
   a. 例外または割込みソースを設定: これには、割込み発生条件の設定も含まれます。レジスタ コンフィギュレーションは要求された具体的な例外に依存します。各ペリフェラルがサポートする割込み設定の詳細については、各ペリフェラルのセクションを参照してください。
   b. Cortex®-M33 CPU の場合、割込みターゲット セキュリティ状態に記述されているように、割込みソースがトリガーされたときにセキュアまたは非セキュア ベクタ テーブルを使用するように NVIC を設定します。
   c. 例外ハンドラ関数を定義し、その関数のアドレスを例外ベクタ テーブルに書き込みます。表 14 に、例外ベクタ テーブルのフォーマットを示します。テーブルの適切な例外番号エントリーに例外ハンドラのアドレスを書き込みます。
   d. 割込み/例外の優先順位で説明したように例外の優先度を設定します。
   e. 割込み/例外の有効と無効で説明されているように、例外を有効にします。

8 フォールト監視

フォールト監視では、デバイス内で発生したさまざまなフォールトを監視し、報告されたフォールトに基づいてアクションを実行できます。フォールトを報告するだけでなく、PSOC™ Control C3 MCU のフォールト構造は、フォールト原因からのデータを記録し、オプションでソフトリセットを実行するメカニズムを提供します。

PSOC™ Control C3 MCU ファミリーは、デバイス内で発生したフォールトを監視する 1 つの集中型フォールト レポート/監視構造に対応しています。フォールト レポート構造は、最大 96 のソースからのフォールトをモニターし、レポートできます。PSOC™ Control C3 MCU は 30 のフォールト原因に対応しています。

8.1 特長

PSOC™ Control C3 MCU のフォールト レポート構造は以下に対応しています。

- 保護ユニット違反、ペリフェラル固有のエラー、メモリコントローラー固有のエラーなどの監視
- フォールト情報を記録する 4 つの 32 ビット データ レジスタ
- フォールト情報を保持したまま、フォールト検出時にソフトリセット
- フォールト検出時の割込み
- DW へのトリガー出力でフォールトデータを転送
- 外部フォールト処理用ピンへのフォールト検出出力

8.2 アーキテクチャ

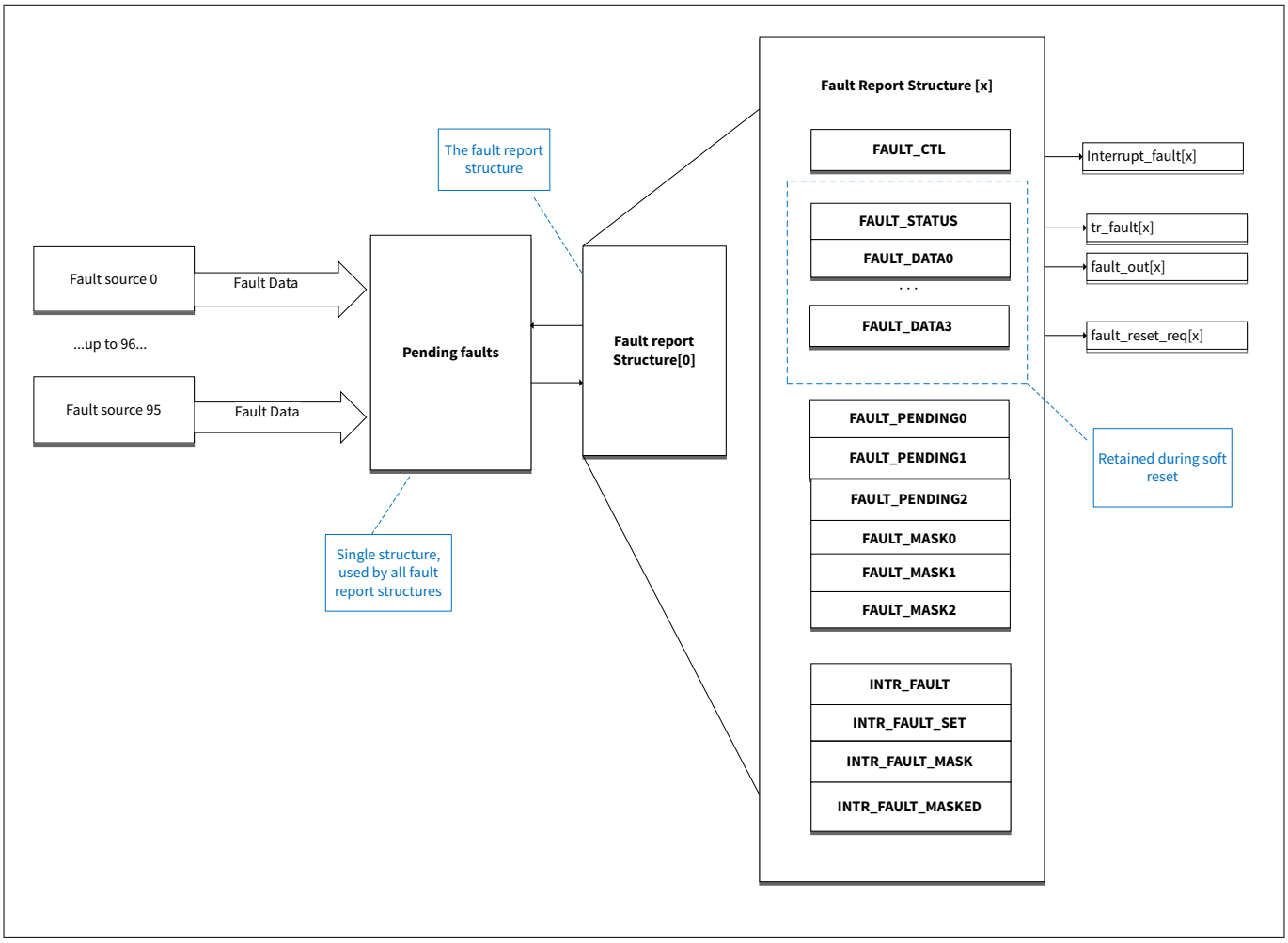


図 17 フォールト レポート アーキテクチャ

PSOC™ Control C3 MCU ファミリーは、集中型フォールト レポート アーキテクチャを採用しているため、システム全体にわたって一貫したフォールト処理を行うことができ、ソフトウェア開発を簡素化できます。フォールト レポート ハードウェアは、すべてのフォールトに対応する単一のフォールト割込みハンドラのみが必要となるよう設定できます。フォールト レポート構造は、1 セットの MMIO レジスタから、フォールト原因とフォールト固有の追加情報を提供します。フォールト原因やフォールト情報を繰り返し検索する必要はありません。保留中の全フォールト情報は単一の MMIO レジスタセットから読み出せます。

フォールト構造は、MPC/PPC 保護違反、ペリフェラル固有エラー、メモリコントローラー固有エラー、ECC エラー、タイムアウトエラーに関連するフォールトをキャプチャします。

フォールト構造は、表 20 にリストされているように、1 つ以上のフォールトを捕捉するような設定も可能です。フォールト構造が特定のフォールトを捕捉するよう設定されている場合、発生したフォールトは保留中のフォールトとして記録されます。フォールト構造が他のすべてのフォールトの処理を終了した場合、または他に保留中のフォールトがない場合、フォールトデータはフォールト構造レジスタにキャプチャされます。さらに、成功したキャプチャは割込みをトリガーし、アプリケーションの要求に応じて Cortex®-M33 で処理できます。

フォールト構造は、フォールトをキャプチャするだけでなく、オプションとして、フォールト情報を保持したままソフトリセットを実行することもできます。このリセットにより SRSS_RES_CAUSE レジスタの RESET_ACT_FAULT がリセットされることになります。

8.2.1 フォールト レポート

PSOC™ Control C3 MCU ファミリーは、1 つのフォールト レポート構造に対応しています。コントロール レジスタとステータス レジスタの専用セットを備えています。フォールト レポート構造は単一のフォールトをキャプチャします。キャプチャされたフォールト情報は以下のとおりです。

- フォールトがキャプチャされていることを示すフォールト有効ビット (FAULT_STRUCT0_STATUS.VALID)。このビットは、フォールトがキャプチャされるたびにセットされます。このビットは、フォールト情報を処理した後にクリアする必要があります。このビットが'0'のときのみ、新しいフォールトがキャプチャされます。
- 表 20 に示すような 7 ビットのインデックスで、フォールト原因を特定します (FAULT_STRUCT0_STATUS.IDX)。
- フォールトの詳細を説明する追加フォールト情報 (FAULT_STRUCT0_DATA0 through FAULT_STRUCT0_DATA3)。この追加情報は、フォールト発生源固有のものです。データ フィールドのエンコーディングは フォールト エンコーディング セクションにあります。
- FAULT_STRUCT0_STATUS は、ソフトウェアから書き込むことができ、フォールトをトリガーします

8.2.2 信号インターフェース

捕捉されたフォールト情報に加えて、フォールト レポート構造は、キャプチャされたフォールトをシステムの他の部分に通知するためのシグナリング インターフェースに対応していします。フォールト レポート構造のインターフェースは、以下に対応しています。

- フォールト割込み (interrupt_fault[0])。FAULT_STRUCT0_INTR、FAULT_STRUCT0_INTR_SET、FAULT_STRUCT0_INTR_MASK、FAULT_STRUCT0_INTR_MASKED レジスタを使用して、フォールト構造の割込みを監視、設定、マスクします。割込みが設定されている状態で FAULT_STRUCT0_STATUS.VALID が設定されると、割込み要求は NVIC に送られます。単一の割込み原因のみが存在しているのは、フォールトが検出されたことを示します。フォールト レポート構造は、割込みハンドラで読み出して、フォールトを推測できます。
- DMA トリガー (tr_fault[0])。FAULT_STRUCT0_CTL.TR_EN を設定することでトリガーが有効になります。トリガーを有効にすると、FAULT_STRUCT0_STATUS.VALID を設定した時に DMA トリガーが起動します。トリガーは DMA コントローラーに接続でき、DMA コントローラーはキャプチャしたフォールト情報をフォールト レポート構造からメモリに転送し、FAULT_STRUCT0_STATUS.VALID フィールドをクリアできます。故障解析には、ウォーム/ソフト リセット時に保持されるメモリ ロケーションを推奨します。
- チップ出力信号 (fault_out[0])。FAULT_STRUCT0_CTL.OUT_EN の設定により出力信号が有効になります。出力信号を有効にすると、FAULT_STRUCT0_STATUS.VALID を設定したときにフォールト構造は出力信号を

生成します。この信号は、HSIOM を通じてデバイスの外にルーティングでき、回復不可能なフォールトをオフチップ コンポーネントに伝達するために使用できます (場合によっては、チップのリセットにつながります)。

- フォールトリセット要求 (fault_reset_req[0])。FAULT_STRUCT0_CTL.RESET_REQ_EN を設定することでリセット機能が有効になります。リセット機能を有効にすると、FAULT_STRUCT0_STATUS.VALID がソフトリセットをトリガーします。リセットは SRSS_RES_CAUSE レジスタの RESET_ACT_FAULT としてキャプチャされます。FAULT_STRUCT0 STATUS および DATA レジスタは、このリセットによって保持されます。

4 種類の信号インターフェースが用意されています。各インターフェースには、それぞれ独自のイネーブル機能があります。FAULT_STRUCT0_STATUS.VALID が設定されると、各有効 インターフェースがアクティブになります。

デバイスは単一の fault_reset_req 信号を持つため、図 18 に示すように、フォールト構造からの個々の fault_reset_req[x]信号は 1 つの fault_reset_req 信号に統合されます。

注: x は、1 つの中心構造を共有する異なるフォールトレポート構造を意味します。

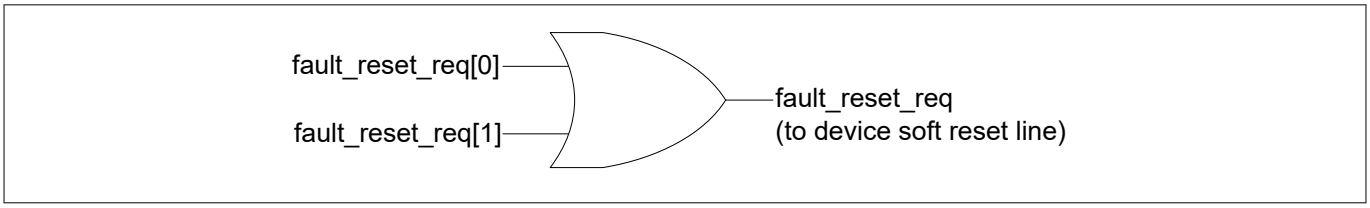


図 18 フォールト リセット

8.2.3 モニター

フォールト レポート構造は、システム内のすべての保留中のフォールトを追跡します。FAULT_STRUCT0_PENDING0、FAULT_STRUCT0_PENDING1、FAULT_STRUCT0_PENDING2 レジスタは、保留中のフォールト原因を反映します。このレジスタは、最大 96 のフォールト原因に対してそれぞれ 1 つの保留ビットを提供します。FAULT_STRUCT0_PENDINGx レジスタは、フォールト レポート構造内でミラーされます。フォールト原因の番号付けは FAULT_STRUCT0_STATUS.IDX の番号付けに従います (詳細は、表 20 を参照)。

保留中のフォールト原因は、フォールト構造によって捕捉されません。フォールトがフォールト構造によってキャプチャされると、関連する保留ビットは‘0’にクリアされます。

フォールト レポート構造は、フォールトを選択的に捕捉します。FAULT_STRUCT0_MASK0、FAULT_STRUCT0_MASK1、FAULT_STRUCT0_MASK2 は、フォールト構造によって捕捉される保留中のフォールト原因を反映します。こうしたフォールトは「有効」フォールトと呼ばれます。フォールト構造は、それ自身の FAULT_STRUCT0_MASK レジスタ セットを含みます。

FAULT_STRUCT0_STATUS.VALID が‘0’の場合、フォールト構造は「有効」フォールトのみをキャプチャします。フォールトがキャプチャされると、ハードウェアは FAULT_STRUCT0_STATUS.VALID を‘1’に設定します。さらに、ハードウェアは関連する保留ビットを‘0’にクリアします。フォールト構造が処理されるとき、ファームウェアまたは DMA 転送は FAULT_STRUCT0_STATUS.VALID を‘0’にクリアする必要があります。フォールト キャプチャは FAULT_STRUCT0_INTR.FAULT ビットを考慮しないことに注意してください。

8.2.4 低消費電力動作

フォールトレポート構造の機能は、CPU アクティブおよび CPU スリープ電力モードでのみ利用可能です。フォールト原因とフォールト レポート構造間のインターフェースは、CPU ディープスリープ電力モードでリセットされます。フォールト レポート構造はアクティブな機能であるため、保留中のフォールト (FAULT_STRUCT0_PENDING レジスタ) は、CPU ディープスリープ電力モードに移行しても保持されません。フォールト構造のレジスタは、リセットドメインとその保持能力に基づいて、以下のように分割できます。

- アクティブ リセットドメイン: FAULT_STRUCT0_PENDING、FAULT_STRUCT0_INTR、FAULT_STRUCT0_INTR_SET、および FAULT_STRUCT0_INTR_MASKED レジスタ。これらのレジスタは、CPU ディープスリープ電力モードでは保持されません。

- ディープスリープのリセット領域: FAULT_STRUCT0_CTL, FAULT_STRUCT0_MASK, および FAULT_STRUCT0_INTR_MASK レジスタこれらのレジスタは CPU ディープスリープ電力モードでも保持されますが、システム リセットはこれらのレジスタをデフォルト状態にリセットします。
- ソフトリセット領域: FAULT_STRUCT0_STATUS と FAULT_STRUCT0_DATA レジスタこれらのレジスタは、ソフトリセット (SRSS_RES_CAUSE レジスタで検出可能) によって保持されます。しかし、XRES/POR/BOD のようなハード リセットはレジスタをリセットします。

8.2.5 フォールト構造の使用

以下の手順に従って、フォールト構造を設定し、使用してください。

- 表 20 からシステムで監視すべきフォールトを特定します。
- 割込みによるファームウェアのフォールト処理の場合:
  1. FAULT_STRUCT0_INTR_MASK.FAULT ビットを設定します。
  2. FAULT_STRUCT0_INTR.FAULT を設定し、保留中の割込みをクリアします。
  3. 適切な ISER レジスタを設定して、CPU の FAULT 割込みを有効にします。割込みセクションを参照してください。
- DW によるフォールト処理の場合:
  1. FAULT_STRUCT0_CTL.TR_EN ビットを設定します
  2. tr_fault[0]信号をトリガー入力 DMA コントローラーに送信します
  3. FAULT_STRUCT0_STATUS と FAULT_STRUCT0_DATA レジスタをメモリに転送し、転送完了後に FAULT_STRUCT0_STATUS レジスタに‘0’を書き戻すように DMA コントローラーを設定し、有効にします。
- デバイス外でのフォールト処理の場合:
  1. FAULT_STRUCT0_CTL.OUT_EN ビットを設定します
  2. fault_out[0]信号を HSIOM を介してピンにルーティングします
  3. 信号をデバイス外で使用し、外部リセット、パワーサイクル、またはフォールト情報のログを生成します。
- FAULT_STRUCT0_CTL.RESET_REQ_EN ビットは、構造のフォールトを検出するためにソフトリセットが必要な場合に設定します。
- FAULT_STRUCT0_STATUS.VALID レジスタをクリアして、キャプチャされたフォールトをクリアします。
- フォールト構造で捕捉する必要があるフォールトについては、FAULT_STRUCT0_MASK レジスタにフォールト インデックス ビットを設定します。

8.2.6 CPU 例外とフォールト監視

表 20 で捕捉されるいくつかのフォールトは、バスエラーまたは CPU 例外 (バス/使用/メモリ/ハードフォールト) にもなります。フォールトは 2 つの方法で通知できます。

- フォールトが発生したバス転送のマスターへのバスエラーとして。その結果、CPU にバス、使用状況、メモリ、ハードフォールトの例外が発生します。
- フォールト レポート構造のフォールトとして。このフォールトは、システム内のどのプロセッサにもフォールト割込みとして伝えられます。これにより、フォールトが発生したバス転送のマスターではないプロセッサでフォールトを処理できます。フォールトの発生した転送のマスターが応答不能または信頼に足らないようなフォールトに有効です。

8.3 フォールト原因

フォールト原因は、デバイス ファミリーによって異なることがあります。表 20 に、PSOC™ Control C3 MCU で利用可能なフォールト原因のリストを示します。

表 20 フォールト原因

フォールト指数	ソース	説明	フォールト エンコーディング
0	peri.peri_ms0_ppc_vio	マスターインターフェースのフォールト インターフェース。PPC 違反 (セキュア/非セキュアアクセスミスマッチ、特権/ユーザーアクセスミスマッチ、PC アクセスミスマッチ) およびバスエラーを報告するために使用されます。	PERI マスター インターフェース PPC 違反。
1	peri.peri_ms1_ppc_vio	マスター 1 インターフェースのフォールト インターフェース。PPC 違反 (セキュア/非セキュアアクセスミスマッチ、特権/ユーザーアクセスミスマッチ、PC アクセスミスマッチ) およびバスエラーを報告するために使用されます。	PERI マスター インターフェース PPC 違反。
2	peri.peri_ppc_pc_mask_vio	PPC MMIO アクセス違反のフォールト インターフェース。このフォールト インターフェースは、ロックされた PC が PC_MASK レジスタに書き込もうとしたときに、PPC 違反を報告するために使用されます。	PERI ペリフェラル PPC PC マスク違反。
3	peri.peri_gp1_timeout_vio	ペリフェラル グループ 1 の AHB タイムアウトのフォールト インターフェース	PERI ペリフェラル グループ AHB タイムアウト。
4	peri.peri_gp2_timeout_vio	ペリフェラル グループ 2 の AHB タイムアウトのフォールト インターフェース	PERI ペリフェラル グループ AHB タイムアウト。
5	peri.peri_gp3_timeout_vio	ペリフェラル グループ 3 の AHB タイムアウトのフォールト インターフェース	PERI ペリフェラル グループ AHB タイムアウト。
6	peri.peri_gp4_timeout_vio	ペリフェラル グループ 4 の AHB タイムアウトのフォールト インターフェース	PERI ペリフェラル グループ AHB タイムアウト。
7	peri.peri_gp5_timeout_vio	ペリフェラル グループ 5 の AHB タイムアウトのフォールト インターフェース	PERI ペリフェラル グループ AHB タイムアウト。
8	peri.peri_gp0_ahb_vio	ペリフェラル グループ 0 の AHB ERROR のフォールト インターフェース	PERI ペリフェラル グループ AHB 違反。

(続く)

表 20 (続き) フォールト原因

フォールト指数	ソース	説明	フォールト エンコーディング
9	peri.peri_gp1_ahb_vio	ペリフェラル グループ 1 の AHB ERROR のフォールト インターフェース	PERI ペリフェラル グループ AHB 違反。
10	peri.peri_gp2_ahb_vio	ペリフェラル グループ 2 の AHB ERROR のフォールト インターフェース	PERI ペリフェラル グループ AHB 違反。
11	peri.peri_gp3_ahb_vio	ペリフェラル グループ 3 の AHB ERROR のフォールト インターフェース	PERI ペリフェラル グループ AHB 違反。
12	peri.peri_gp4_ahb_vio	ペリフェラル グループ 4 の AHB ERROR のフォールト インターフェース	PERI ペリフェラル グループ AHB 違反。
13	peri.peri_gp5_ahb_vio	ペリフェラル グループ 5 の AHB ERROR のフォールト インターフェース	PERI ペリフェラル グループ AHB 違反。
14	cpuss.ramc0_mpc_fault_mmio	RAMC-0 MPC フォールト	RAMC0 MPC 違反。
15	cpuss.exp_ms_ahb_error_mmio	EXP ブリッジの AHB エラーにより発生したフォールト	AHB ブリッジ エラー フォールト
16	cpuss.promc_mpc_fault_mmio	PROMC MPC フォールト	PROMC の MPC 違反。
17	cpuss.ramc0_c_ecc_mmio	RAMC-0 訂正可能な ECC フォールト	RAMC0_C ECC mmio。
18	cpuss.ramc0_nc_ecc_mmio	RAMC-0 修正不可能な ECC フォールト	RAMC0_NC ECC mmio。
19	cpuss.flashc_mpc_fault	FLASHC MPC フォールト	FLASCH 製 MPC の不具合。
20	cpuss.flashc_bus_access_fault	FLASHC バスエラー	FLASHC バスアクセス フォールト
21	cpuss.flashc_main_c_ecc_mmio	FLASHC ECC 修正可能フォールト	FLASHC メイン_c ECC mmio フォールト
22	cpuss.flashc_main_nc_ecc_mmio	FLASHC ECC の訂正不可能なフォールト	FLASHC メイン_nc ECC mmio フォールト。
23	cpuss.flashc_fm_ctl_fault	FLASHC FM CTRL フォールト	FLASHC FM CTL フォールト。
24	cpuss.flashc_fm_r_q_fault	FLASHC FM r_q フォールト	FLASHC FM r_q フォールト。
25	cpuss.flashc_flash_ctl_fault	FLASHC FLASH CTL フォールト	FLASHC フラッシュ CTL フォールト。
26	cpuss.flashc_addr_hole_fault	FLASHC ADDR HOLE フォールト	FLASHC アドレス ホール フォールト。
27	cpuss.flashc_lock_fault	FLASHC ロック フォールト	FLASHC ロック フォールト

(続く)

表 20 (続き) フォールト原因

フォールト指数	ソース	説明	フォールト エンコーディング
28	srss.fault_csv	CSV フォールト インターフェース	SRSS クロックスーパーバイザー フォールト
29	srss.fault_ssv	SSV フォールト インターフェース	SRSS 電源管理フォールト

8.4 フォールト エンコーディング

ここでは、各フォールトのデータフィールドのすべてのエンコーディングが含まれています。

HMASTER, HWRITE, HPROT など、"H"で始まるデータフィールドの詳細については、Arm® AHB protocol specification を参照してください。

8.4.1 PERI マスターインターフェース PPC 違反

このフォールト インターフェースは、PPC 違反やバス エラーを報告するために使用されます。

データフィールド「違反の種類」は、複数の違反が同時に発生した場合、最も優先度の高い違反のみを記録します。

「違反のタイプ」が AHB エラーである場合、周辺領域インデックス値は無効です。

表 21 PERI マスターインターフェース PPU 違反データフィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[9:0]: 周辺部指数 [15:12]: マスター インターフェース トランザクションの保護コンテキスト [18:16]: 違反の種類 • 0: 違反なし • 1: セキュア/非セキュアアクセスの不一致 (優先順位 2) • 2: 特権/ユーザーアクセスの不一致 (優先順位 3) • 3: PC アクセスの不一致 (優先順位 1) • 4: 予約済み • 5: AHB エラー (優先順位 4) • 6, 7: 予約済み [31:24]: HMASTER

8.4.2 PERI ペリフェラル PPC PC マスク違反

このフォールト インターフェースは、ロックされた PC が PC_MASK レジスタに書き込もうとしたときに PPC 違反を報告するために使用されます。

表 22 ペリフェラル PPC PC マスク違反データフィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[9:0]: 周辺部指数 [11:10]: 予約済み [15:12]: マスター インターフェーストランザクションの保護コンテキスト [18:16]: 違反の種類 • 4: ロックされた PC から PC_MASK レジスタへの書き込み • 0、1、2、3、5、6、7: 予約済み [23:19]: 予約済み [31:24]: HMASTER

8.4.3 PERI ペリフェラル グループ AHB タイムアウト

このフォールト インターフェースは、各ペリフェラル グループにおけるタイムアウトエラーの報告に使用されます。各ペリフェラル グループには、独自のフォールト インターフェースがあります。

表 23 ペリフェラル グループ AHB タイムアウト データ フィールド

DATA0
[4:0]: ペリフェラル グループスレーブ番号 [31:5]: 予約済み

8.4.4 PERI ペリフェラル グループ AHB 違反

このフォールトインターフェースは、ペリフェラルグループデコーダー (ブリッジに面する側) のバスエラーを報告するために使用されます。
各ペリフェラル グループには、独自のフォールト インターフェースがあります。

表 24 ペリフェラル グループ AHB 違反データ フィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[0]: 特権モード • 0: ユーザーモード • 1= 特権モジュール [1]: HNONSEC [2]: HWRITE [3]: 予約済み [7:4]: マスター インターフェーストランザクションの保護コンテキスト [23:8]: HMASTER [31:30]: エラータイプ • 0: AHB エラー • 1, 2, 3: 予約済み

8.4.5 RAMC0 MPC 違反

このフォールトは、アクセス違反があり、MPC_CFG.RESPONSE が設定されているときにトリガーされます。アクセス違反には、読み出し不可能なメモリからの読み出し、書き込み不可能なメモリへの書き込み、セキュアなメモリへの非セキュア アクセスが含まれます。

MPC_CFG.RESPONSE が設定されず、アクセス違反が発生した場合、フォールトは生成されません。読み出しはゼロで、書き込みは無視されます。

表 25 MPC 違反のデータフィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[5:0]: HMASTER [15:6]: 予約済み [16]: HNONSEC [17]: 違反ブロックのセキュアまたは非セキュア構成 • 0: セキュア • 1: 非セキュア [18]: HWRITE [23:19]: 予約済み [27:24]: マスター インターフェース トランザクションの保護コンテキスト [29:28]: 未使用 [30]: セキュリティアクセス違反 [31]: PC アクセス違反

8.4.6 AHB ブリッジエラー フォールト

このフォールト インターフェースは、AHB ブリッジの AHB エラーを報告するために使用されます。

表 26 AHB ブリッジ エラー データ フィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[0]: HPROT [1]: HNONSEC [2]: HWRITE [3]: 予約済み [7:4]: マスター インターフェース トランザクションの保護コンテキスト [15:8]: HMASTER [31:16]: 予約済み

8.4.7 PROMC MPC 違反

表 27 PROMC MPC 違反のデータフィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[5:0]: MASTER [15:6]: 予約済み [16]: NONSEC [17]: 予約済み [18]: WRITE [23:19]: 予約済み [27:24]: マスター インターフェース トランザクションの保護コンテキスト [29:28]: 未使用 [30]: セキュリティアクセス違反 [31]: PC アクセス違反

8.4.8 RAMC0_C ECC mmio

このフォールト インターフェースは、訂正可能な RAMC0 ECC エラー検出フォールトを報告するために使用されます。このフォールトは、MPC アクセス フォールトと共に発生した場合、マスクされます。

表 28 RAMC0_C ECC mmio データ フィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[6:0]: SYNDROME

8.4.9 RAMC0_NC ECC mmio

このフォールト インターフェースは、訂正不能な RAMC0 ECC エラー検出フォールトを報告するために使用されます。このフォールトは、MPC アクセス フォールトと共に発生した場合、マスクされます。

表 29 RAMC0_NC ECC mmio データ フィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[6:0]: SYNDROME

8.4.10 FLASHC MPC フォールト

これは FLASHC MPC の違反フォールトインターフェースです。

表 30 FLASHC MPC 違反のデータフィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[15:0] MASTER [16] 非セキュア取引 [17] 違反したブロックのセキュアまたは非セキュアなコンフィギュレーション [18] WR • 1: 書き込み • 0: 読み出し [23:19] 予約済み [27:24] アクセス違反のトランザクション PC 29～28: 予約済み [30] セキュリティアクセス違反 [31] PC アクセス違反

8.4.11 FLASHC バスアクセスフォールト

このフォールト インターフェースは、以下のフォールトを報告するために使用されます。

- FM レジスタ FM_ADDR/FM_REFRESH_ADDR への C バスアクセス (Full R-bus mod) 時
- フル C バス モードでの R バス アクセス
- フル C バス/RWW モードで、ENFORCE_PC_LOCK = 0 のとき、128 ビット AHB5 ホスト インターフェースで AHB5 書き込みトランザクションを実行
- 128 ビット AHB5 バスによるフル R バスモードで FM_ADDR レジスタへの書き込み

表 31 FLASHC バスアクセスデータフィールド

DATA0	DATA1
[31:0]: 違反したアドレス	[0] フル R バス モード [1] フル C バス モード [2] RWW モード [3] ENFORCE_PC_LOCK [4] どの AHB5 インターフェースからのトランザクションかを示します [6:5] フォールトの種類 • 00: フル R バス モードでの 128 ビット AHB5 ホスト インターフェースからの C バス アクセス/書き込み • 01: フル C バス モードでの R バス アクセス • 10: フル C バス/RWW モードで、ENFORCE_PC_LOCK = 0 のとき、128 ビット AHB5 ホスト インターフェースで AHB5 書き込みトランザクション • 11: 予約済み 31～7: 予約済み

8.4.12 FLASHC main_c ECC mmio フォールト

このフォールト インターフェースは、パラメーター ECC_PRESENT が High で、ECC 回復可能エラー カウントが閾値カウンター (ECC_LOGIR.ECCTHRESHOLD) を超えている (ECC (エラー) 挿入が原因ではない) 場合にフォールトを報告するために使用されます。

表 32 flashc_main_c_ecc_mmio データフィールド

DATA0	DATA1
[31:0] 違反アドレス	[8:0] シンドローム 31～9: 予約済み

8.4.13 FLASHC main_nc ECC mmio フォールト

このフォールト インターフェースは、回復不可能な ECC エラー (ECC (エラー) 注入によるものではない) が発生した場合にフォールトを報告するために使用されます。

表 33 flashc_main_nc_ecc_mmio データフィールド

DATA0	DATA1
[31:0] 違反アドレス	[8:0] シンドローム 31～9: 予約済み

8.4.14 FLASHC FM CTL フォールト

フォールト インターフェース:

- 無効モード (IF_SEL = 1、WR_EN = 1) で FLASH_MACRO_CTL レジスタに書き込み
- FLASH_MACRO_CTL.IF_SEL と FLASH_MACRO_CTL.WR_EN の同時更新
- FLASH_CTL.ENFORCE_PC_LOCK が High のとき、TEST_CTL レジスタに 0 以外の値を書き込んでください

表 34 FLASHC FM CTL データフィールド

DATA0	DATA1
[31:0] 違反アドレス	[0] FLASH_MACRO_CTL.IF_SEL に書き込む値 [1] FLASH_MACRO_CTL.WR_EN に書き込む値 [2] ENFORCE_PC_LOCK [4:3] フォールトの種類 • 00: FLASH_MACRO_CTL.IF_SEL および FLASH_MACRO_CTL.WR_EN 1 を使用して FLASH_MACRO_CTL レジスタに書き込んでください • 01: FLASH_MACRO_CTL.IF_SEL と FLASH_MACRO_CTL.WR_EN の同時更新 • 10: FLASH_CTL.ENFORCE_PC_LOCK が High の時に TEST_CTL レジスタに 0 以外の値を書き込んでください • 11: 予約済み 31～5: 予約済み

8.4.15 FLASHC FM r_q フォールト

R バス アドレス競合のフォールトを報告するためのフォールト インターフェースです (FM からの r_q_error)。読み出しアドレスが C バスのアドレスと競合します。

表 35 FLASHC FM r_q データフィールド

DATA0	DATA1
[4:0] Word address [log2(FLASHC_ROWS_PER_SECTOR-1)+4:5]行アドレス [FM_ADDR_WIDTH-4:FM_ADDR_WIDTH-4-log2(FLASHC_SECTOR_M-1)+1]セクタアドレス [FM_ADDR_WIDTH-3] axa (01 は SFLASH 用) [FM_ADDR_WIDTH-2] bxa (01 はリフレッシュ行用) [FM_ADDR_WIDTH-1] cxa (01 は 33 列目用) [32-FM_ADDR_WIDTH] 予約済み	7～0: 予約済み [MASTER_WIDTH+8-1:8] マスター ID [31:MASTER_WIDTH+8] 予約済み

8.4.16 FLASHC フラッシュ CTL フォールト

フォールト インターフェース:

- FLASH_CTL.BLOCK_SECTOR_OPERATIONS が High のとき、FLASH_MACRO_CTL.FM_MODE[3:0]をモード 9、12、13 に設定
- FLASH_CTL.BLOCK_SUBSECTOR_OPERATIONS が High のとき、FLASH_MACRO_CTL.FM_MODE[3:0]をモード 6 と 8 に設定
- FLASH_CTL.BLOCK_BULK_OPERATIONS が High のとき、FLASH_MACRO_CTL.FM_MODE[3:0]をモード 10 と 15 に設定
- FLASH_CTL.ENFORCE_PC_LOCK が High のとき、FLASH_MACRO_CTL.FM_MODE[3:0]をモード 3 に設定

表 36 FLASHC フラッシュ CTL データフィールド

DATA0

[3:0] FM_MODE[3:0]

[4] ENFORCE_PC_LOCK

[5] BLOCK_SECTOR_OPERATIONS

[6] BLOCK_SUBSECTOR_OPERATIONS

[7] BLOCK_BULK_OPERATIONS

[9:8] フォールトの種類

- 00: FLASH_CTL.BLOCK_SECTOR_OPERATIONS high は、SW による FLASH_MACRO_CTL.FM_MODE[3:0] のモード 9、12、および 13 に設定することを禁止
- 01: FLASH_CTL.BLOCK_SUBSECTOR_OPERATIONS high 設定時は、SW による FLASH_MACRO_CTL.FM_MODE[3:0] のモード 6 および 8 に設定することを禁止
- 10: FLASH_CTL.BLOCK_BULK_OPERATIONS high は、SW による FLASH_MACRO_CTL.FM_MODE[3:0] のモード 10 および 15 に設定することを禁止
- 11: FLASH_CTL.ENFORCE_PC_LOCK が High のとき、FLASH_MACRO_CTL.FM_MODE[3:0]をモード 3 に設定

31～10: 予約済み

8.4.17 FLASHC アドレス ホール フォールト

128 ビットの AHB5 インターフェースを経由して、傷や FM アドレスホールを狙った不正なアドレスへのトランザクションが発生した場合にフォールトを報告するフォールト インターフェースです。

表 37 FLASHC アドレスホールのデータフィールド

DATA0
[31:0] 違反アドレス

8.4.18 FLASHC ロック フォールト

非取得 PC または非取得 HMASTER_ID を持つ取得 PC が、FLASH_LOCK.LOCKED の書き込み/解除を行おうとした際のフォールトを検出するためのフォールトインターフェース。

表 38 FLASHC ロック データ フィールド

DATA0
[0] FLASH_LOCK.LOCKED
[4:1] トランザクション PC
[8:5] 取得 PC
[MASTER_WIDTH+8:9] 違反転送のマスター ID
[2*MASTER_WIDTH+8:MASTER_WIDTH+9] 取得マスター ID
[31:2*MASTER_WIDTH+9] 予約済み

8.4.19 SRSS クロック監視フォールト

このフォールトは、SRSS がいずれかのクロックのエラーを検出したときに生成されます。

表 39 SRSS クロック監視 (CSV) データフィールド

DATA0
[6:0]: CSV_HF
[24]: CSV_REF
[25]: CSV_LF
[26]: CSV_ILO

8.4.20 SRSS 電源管理フォールト

このフォールトは、SRSS がいずれかの電源のエラーを検出したときに発生します。

表 40 SRSS 電源管理 (SSV) データフィールド

DATA0
[0]: BOD_VDDA
[16]: HVLVD1
[17]: HVLVD2

8.5 レジスタの概要

表 41 フォールトレジスタの概要

レジスタ	機能
FAULT_STRUCT0_CTL	フォールト コントロール レジスタ
FAULT_STRUCT0_STATUS	フォールト ステータス レジスタ
FAULT_STRUCT0_DATA	フォールト データレジスタがフォールト情報を取り込みます
FAULT_STRUCT0_PENDING0	fault pending0 レジスタは、保留中の (キャプチャされていない) フォールト原因を指定します。
FAULT_STRUCT0_PENDING1	fault pending1 レジスタは、保留中の (キャプチャされていない) フォールト原因を指定します。
FAULT_STRUCT0_PENDING2	fault pending2 レジスタは、保留中の (キャプチャされていない) フォールト原因を指定します。
FAULT_STRUCT0_MASK0	fault mask0 レジスタは、フォールト原因の「有効」を指定します。
FAULT_STRUCT0_MASK1	fault mask1 レジスタは、フォールト原因の「有効」を指定します。
FAULT_STRUCT0_MASK2	fault mask2 レジスタは、フォールト原因の「有効」を指定します。
FAULT_STRUCT0_INT	フォールト割込み原因レジスタ
FAULT_STRUCT0_INT_SET	フォールト割込みセット レジスタ
FAULT_STRUCT0_INT_MASK	フォールト割込みマスク レジスタ
FAULT_STRUCT0_INT_MASKED	フォールト割込みマスクド レジスタ

9 ダイレクトメモリ アクセス (DMA - DW)

ダイレクト メモリ アクセス (DMA) は、32 ビット アドバンスド ハイパフォーマンス バス (AHB) を使用して、メモリ、ペリフェラル、レジスタ間でデータを転送します。こうした転送は、CPU から独立して行われます。DMA は、複数の独立したデータ転送を実行するよう設定できます。各データ転送はチャネルによって管理されます。DMA には 16 のチャネルがあります。デバイスの詳細については、MCU のデータシートを参照してください。

CPU には DMA コントローラーが搭載されており、メモリ、ペリフェラル、レジスタとのデータ転送に使用できます。これにより、PWM のようなペリフェラルの自律的で確定的な制御や、ADC のようなペリフェラルからメモリへの大容量データ転送が可能になります。

DMA コントローラーは、M33 ドメインのバス マスターです。

- 16 チャネルの 2 つの DMA。バス マスター アクセスをアービトレーションする全チャネル用のシングル転送エンジンを備えています。この DMA は、M33 とクロックを共有する 32 ビット AHB バスを使用します。DMA は小さなデータ転送に最適で、トランザクション DMA は通常、メモリからペリフェラル、またはペリフェラルからメモリへのデータ転送に使用されます。

***注**: DMA は、トリガー マルチプレクサ (トリガー MUX) の章におけるレジスタ名とトリガーの両方で、データワイヤ (DW) として参照されます。*

9.1 特長

DMA コントローラーには以下の特長があります。

- 16 データチャネルに対応
- チャネルごとに 4 段階の優先度
- 記述子はメモリ上に定義され、それぞれのチャネルに参照されます。
- 記述子のシングル、1D、2D 転送モードに対応
- 設定可能な送信、受信アドレス増分
- ソース側とデスティネーション側で 8 ビット、16 ビット、32 ビットのデータ幅
- 各チャネルの入力トリガー動作を設定可能
- 各チャネルで設定可能な割込み発生
- 各チャネルに設定可能な出力トリガー生成
- 記述子は、より複雑な転送を行うために、他の記述子に連鎖できます。
- インターフェース巡回冗長検査 (CRC)

9.2 アーキテクチャ

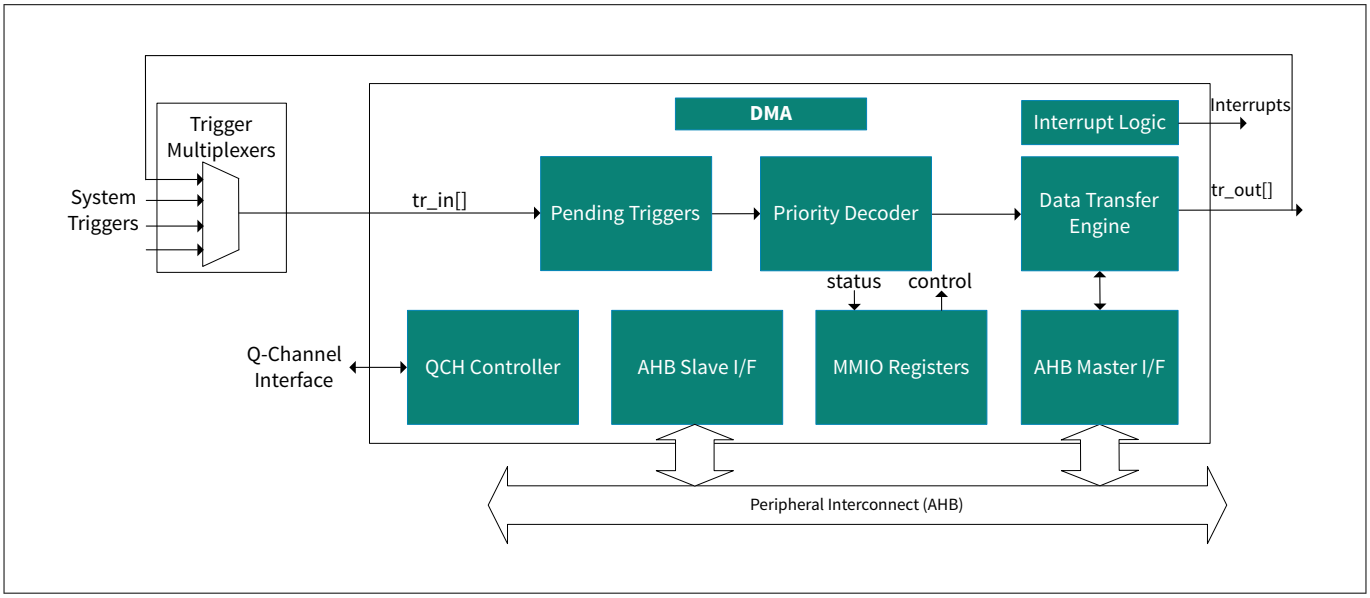


図 19 DMA のアーキテクチャ

データ転送は入力トリガーによって開始されます。このトリガーは、転送元、転送先、CPU SW、または他の SoC コンポーネントから発生することがあります。トリガーはアクティブ/スリープ機能を提供しますが、CPU/システム ディープスリープおよびハイバネート電力モードでは利用できません。

転送がトリガーされると、DMA はチャネル記述子によって設定されたとおりに転送を完了します。チャネルは、1D または 2D のメモリコピーを実行するように設定できます。転送が終了すると、記述子は新しい記述子を指すことによって新しい転送を開始するか、完了割込みを発生できます。

DMA は以下のブロックに分かれています。

- 保留中のトリガー: トリガー保留ブロックは、保留中ビットの形態でローカルにトリガーを保存することで、起動されたトリガーを監視します。データ転送エンジンは 1 回に 1 つのチャネルにしか対応できません。複数のチャネル トリガーが同時に起動されることがある状況で不可欠な機能です。
- 優先度デコーダー: 優先度デコーダーは起動されたトリガーで最高の優先度チャネルを判定します。プライオリティグループ内では、トリガーはラウンドロビン方式でデコードされます。
- データ転送エンジン: 転送元の位置から転送先の位置までデータを転送します。アイドル時には、データ転送エンジンは最優先でアクティブ化されたチャネルを受け入れる準備ができています。また、メモリからチャネル記述子を読み出す役割もあります。
- AHB スレーブ I/F: メイン CPU がコントローラー MMIO のコントロール/ステータス レジスタにアクセスできる AHB バス スレーブ
- AHB マスター I/F: AHB バス マスターは、コントローラーが転送元および転送先に対して AHB データ転送の開始を行い、メモリから記述子を読み出せるようにします。
- 割込みロジック: 割込み回路は各チャネルの割込みステータスを含みます。
- MMIO レジスタ: 各チャネルの記述子を格納
- QCH (Q チャネル) コントローラー: デバイスの電源状態を変更する前に、パワーポリシーユニット (PPU) が DMA の現在の状態をチェックできるようにします。

9.2.1 標準フロー

典型的な DMA データ転送は次のような手順で行われます。

1. メイン CPU は (メモリ内の) 記述子チェーンをプログラムし、それを特定のチャネルに関連付けます。さらに、チャネル レジスタをプログラムし、チャネルに必要な属性を設定します。また、MMIO レジスタをプログラムし、チャネルの特定のシステム トリガーを選択します。
2. チャネルのシステムトリガーがアクティブになります。
3. 優先順位デコードは、最も優先順位の高い保留チャネルを決定します。
4. データ転送エンジンは、アクティブ化されたチャネルを受け入れ、チャネル識別子を使用して、マスター I/F を使用してメモリからチャネルの記述子構造をロードします。記述子構造体は、チャネルのデータ転送を指定します。
5. データ転送エンジンは、マスター I/F を用いて、転送元からデータをロードします。
6. データ転送エンジンは、マスター I/F を用いて、転送先にデータを保存します。転送されるデータ量は TR_IN_TYPE に依存します。
7. データ転送エンジンは、アクティブ記述子レジスタとチャネル構造レジスタを更新し、データ転送を反映します。メモリ上の記述子は更新されません。
8. 出力トリガー発生と割込み発生は、それぞれ TR_OUT_TYPE と INTR_TYPE で決定されます。DW はエラーが発生した場合、エラー割込みを発生できます。エラーが発生した場合、チャネルは無効になります。

9.3 チャネル

DMA コントローラーは、チャネルによって管理される複数の独立したデータ転送に対応します。各チャネルは、DMA コントローラーの外部にあるトリガーマルチプレクサを介して、特定のシステムトリガーに接続します。

9.3.1 チャネルの優先度

1 度に 1 つの DMA チャネルのみがデータを転送できる可能性があります。各チャネルは、複数のチャネルがトリガーされた場合に、どのチャネルがバス アクセスを受けるかを決定するために、対応する優先順位を持っています。

各チャネルに 0 から 3 までの優先度 (CHi_CTL.PRIO) が割り当てられます。最も高い優先度が 0 で、最も低い優先度が 3 です。同じプライオリティを持つチャネルは優先度のグループを構成します。優先度デコードは、最も優先順位の高い保留チャネルを決定します。このチャネルは以下のように決定されます。

- 保留中のチャネルを持つ最優先グループが最初に特定されます。
- この優先順位グループ内では、ラウンドロビン アービトレーションが適用されます。ラウンドロビン アービトレーション (優先度グループ内) では、(優先度グループ内で) 下位のチャネルインデックスに最も高い優先度を与えます。

各チャネルは、CHi_CTL.PREEMPTABLE を‘1’に設定することで、プリエンプト可能としてマークできます。優先順位の高いチャネルが保留状態になると、1D 転送の間にプリエンプト可能のマークが付いたチャネルに割り込みます。優先順位の高いチャネル転送が完了すると、プリエンプト可能のマークが付けられたチャネルは、コントローラーのステートマシンによって再アクティブ化されます。

9.3.2 チャネルのセキュリティ属性

各チャネルには 3 つのアクセス制御属性があり、メモリ保護コントローラー (MPC) とペリフェラル保護コントローラー (PPC) がアクセス制御に使用します。以下の 3 つのフィールドは、CHi_CTL レジスタを構成する MMIO 書き込みトランザクションのアクセス属性から継承されます。

- 「特権モード」(CHi_CTL.P) 属性は、特権またはユーザーとして継承できます。
- 「非セキュア」(CHi_CTL.NS) 属性は、セキュアまたは非セキュアとして継承できます。
- 「PC」(CHi_CTL.PC) は、書き込みトランザクションから保護コンテキストを継承します。

9.3.3 チャネル状態

任意の時点で、アクティブにデータ転送を行なっているチャネルはせいぜい 1 つです。このチャネルはアクティブチャネルと呼ばれます。チャネルは、以下の表に示す 4 つのチャネル状態のいずれかになります。

表 42 チャネル状態

チャネル状態	レジスタ	説明
無効	CHi_CTL.ENABLED = 0	チャネルは DW_CHi_CTL.ENABLED を 0 に設定することで無効化されています。この状態では、チャネル トリガーは無視されます。アクティブ チャネルがソフトウェアによって無効化された場合、チャネルの状態 (MMIO レジスタまたは記述子によって反映される転送の現在の「位置」) について仮定を行うべきではありません。ソフトウェアによるチャネルの再有効化は、新しい記述子を準備し、チャネルを再設定する必要があります。
ブロック	CHi_CTL.ENABLED = 1 CHi_STATUS.PENDING = 0	チャネルは有効で、データ転送を開始するトリガーを待っています。
保留中	CHi_CTL.ENABLED = 1 CHi_STATUS.PENDING = 1	チャネルが有効で、アクティブトリガーを受信しました。この状態では、チャネルはデータ転送を開始する準備ができていますが、スケジュールされるのを待っています。
アクティブ	CHi_CTL.ENABLED = 1 CHi_STATUS.PENDING = 1 DW_STATUS.CH_IDX = アクティブ チャネル DW_STATUS.ACTIVE = 1	チャネルは有効で、アクティブトリガーを受信し、スケジューリングされています。アクティブにデータ転送を行っています。保留中のチャネルが複数ある場合、最も優先順位の高いチャネルがスケジューリングされます。

チャネルがアクティブ状態になると、DW_STATUS.STATE を使用してより詳細な情報を得られます。

- Val 0: デフォルト/非アクティブ状態
- Val 1: メモリからチャネル記述子をロードします
- Val 2: ソースロケーションからデータ エレメントをロードします
- Val 3: データ エレメントを保存先に保存します
- Val 4: CRC 機能 (DESCR_CTL.DESCR_TYPE が CRC に設定されている場合のみ使用されます)
- Val 5: アクティブな制御情報 (送信元アドレスと宛先アドレス) を更新し、トリガーの非アクティブ化を待ちます。
- Val 6: エラー

9.3.4 チャネル割込み生成

各 DMA チャネルには、それに関連した割込みラインがあります。DMA では、転送完了時やエラー発生時に割込みを発生できます。これらの割込みは、転送終了時に CPU をウェイクアップさせたり、CPU が DMA エラーを修正する必要があることを通知したりします。各 DMA チャネルには、割込みの制御/処理用に 6 つのレジスタがあります。転送の種類の詳細については、記述子メモリ転送タイプを参照してください。

DESCR_CTL.INTR_TYPE は DMA チャネルによって生成される完了割込み (CHi_STATUS.INTR_CAUSE は COMPLETION に設定) を選択します。

- 設定#0: 割込みはシングル転送の後に発生します。
- 設定#1: 割込みはシングル 1D 転送の後に発生します。記述子タイプが「single」の場合、割込みは 1 回の転送の後に発生します。記述子タイプが「1D」、「CRC」、「2D」の場合、割込みは 1D 転送の実行後に発生します。
- 設定#2: 現在の記述子の実行後に割込みが発生します。現在の記述子の DESCR_NEXT_PTR.ADDR の値に依存しません。
- 設定#3: 現在の記述子の実行後に割込みが発生します。現在の記述子の DESCR_NEXT_PTR.ADDR の値は “0”でなければいけません。

CHi_STATUS.INTR_CAUSE は、CHi_INTR.CH が‘1’に設定されているときの割込み発生源を指定します。CHi_STATUS.INTR_CAUSE では以下の割込みを発生できます。

- 値 0: 割込み信号を検出しない (NO_INTR)
- 値 1: 記述子 (COMPLETION) の DESCR_CTL.INTR_TYPE フィールドに基づいて設定された転送完了時の割込み
- 値 2: ソースバスエラー (SRC_BUS_ERROR)
- 値 3: デスティネーション バス エラー (DST_BUS_ERROR)
- 値 4: ソースアドレスの不整合 (SRC_MISAL)
- 値 5: 宛先アドレスのずれ (DST_MISAL)
- 値 6: 現在の記述子ポインタが NULL (CURR_PTR_NULL)
- 値 7: アクティブチャネルが無効状態 (ACTIVE_CH_DISABLED)
- 値 8: 記述子 バス エラー (DESCR_BUS_ERROR)

CHi_INTR は、割込み (CHi_STATUS.INTR_CAUSE で指定) が検出されるとハードウェアによって‘1’に設定されます。レジスタは‘1’を書き込むことでクリアされます。

CHi_INTR_SET を‘1’に設定すると、対応する CHi_INTR フィールドが設定されます。

CHi_INTR_MASK は、指定された DMA チャネルの割込みマスクを設定するために使用されます。

CHi_INTR_MASKED を読み出すと、CHi_INTR と CHi_INTR_MASK の間のビットごとの結合が得られます。

エラー関連の割込み原因 (CHi_STATUS.INTR_CAUSE 2～8) の場合、ハードウェアはチャネルを無効にします (HW は CHi_CTL.ENABLED を‘0’に設定します)。この割込みは、チャネルが再び有効になる前にクリアされなければいけません。

DMA コントローラーは CPU アクティブ電力モード ペリフェラルです。そのため、CPU/System ディープスリープ電力モードでは INTR と INTR_SET は保持されません。INTR_MASK は保持されます。

9.4 記述子

チャネル内の送信元と宛先間のデータ転送は、記述子と呼ばれる 32 ビット ワードの集まりを使って設定されます。記述子は最大 6 つの 32 ビット ワードの集合で、4 つの異なるメモリ転送タイプをサポートするように設定できます。

記述子ポインタは DMA チャネル レジスタで指定されます。CHi_CURR_PTR は、現在の記述子のアドレスを格納し、現在の記述子の実行後に DESCR_NEXT_PTR に更新されます。DMA コントローラーは記述子を変更せず、読み出し専用として扱います。

各チャネルには、X_IDX と Y_IDX のビット フィールドを含む CHi_IDX レジスタが含まれます。これら 2 つのフィールドは、転送が完了するときの現在の記述子の X と Y のインデックスを含みます。これらは CHi_CURR_PTR が更新されたときにゼロにセットされ、X と Y の各ループの後にインクリメントされます。

9.4.1 記述子メモリ転送タイプ

DMA は、シングル、1D、2D、CRC (巡回冗長検査) の 4 種類のメモリ転送に対応します。メモリ転送タイプは DESCR_CTL.DESCR_TYPE で設定されますが、指定された転送タイプをサポートするために最大 6 つの設定された 32 ビット ワードが必要となることがあります。以下のセクションでは、3 つのメモリ転送タイプについて説明します。

9.4.1.1 シングル転送

記述子設定は、単一のデータ エレメントを実行します。データ エレメント サイズは DESCR_CTL.DATA_SIZE で 8 ビット、16 ビット、32 ビットに設定できます。シングル転送には、4 つの 32 ビット ワードのコンフィギュレーションが必要です。以下の擬似コードは、シングル転送の機能を説明するものです。

必要な設定: DESCR_CTL、DESCR_SRC、DESCR_DST、DESCR_NEXT_PTR。

```
// DST_ADDR is a pointer to an object of type defined by DESCR_DST.DST_ADDR
            // SRC_ADDR is a pointer to an object of type defined by DESCR_SRC.SRC_ADDR
            // t_DATA_SIZE is the type associated with the DESCR_CTL.DATA_SIZE
            DST_ADDR[0] = (t_DATA_SIZE) SRC_ADDR[0];
```

下図に、シングル転送でメモリ内の 32 ビットを移動する方法を示します。

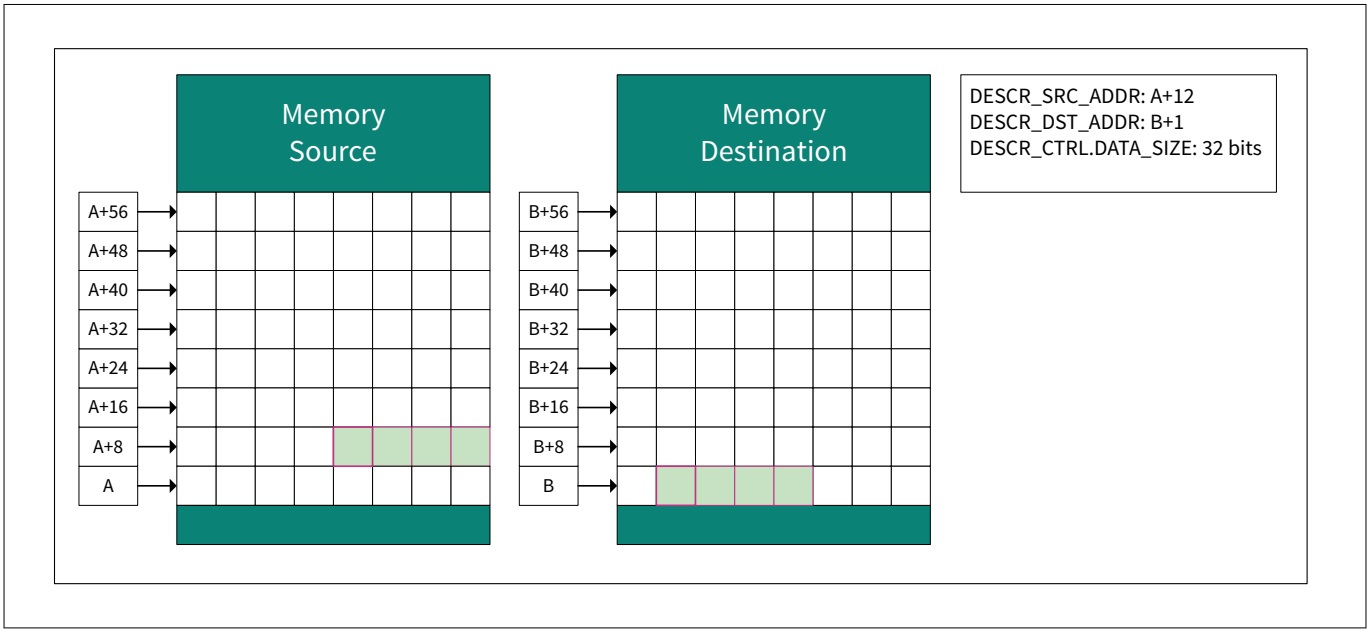


図 20 シングル転送

9.4.1.2 1D メモリ転送

記述子設定は 1 次元の「for loop」を実行し、チャネルは X_COUNT+1 バイトを DESCR_SRC から DESCR_DST にコピーします。1D メモリ転送は、5 つの 32 ビットワードの設定を必要とします。以下の擬似コードは、1D メモリ転送の機能を説明するものです。

必要な設定: DESCR_CTL、DESCR_SRC、DESCR_DST、DESCR_X_CTL、DESCR_NEXT_PTR。

```
// DST_ADDR is a pointer to an object of type defined by DESCR_DST.DST_ADDR
            // DST_X_INCR specifies the source increment between single transfer defined by
DESCR_X_CTL.DST_X_INCR
            // SRC_ADDR is a pointer to an object of type defined by DESCR_SRC.SRC_ADDR
            // SRC_X_INCR specifies the source increment between single transfer defined by
DESCR_X_CTL.SRC_X_INCR
            // t_DATA_SIZE is the type associated with the DESCR_CTL.DATA_SIZE
            // X_COUNT is the number of single transfers defined by DESCR_X_CTL.X_COUNT
            for (X_IDX = 0; X_IDX <= X_COUNT; X_ID++){
            DST_ADDR[X_IDX * DST_X_INCR] = (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR};
            }
```

次の図に、1D メモリ転送を使用してメモリ内の 3 つの 32 ビット データ エレメントを移動する方法を示します。

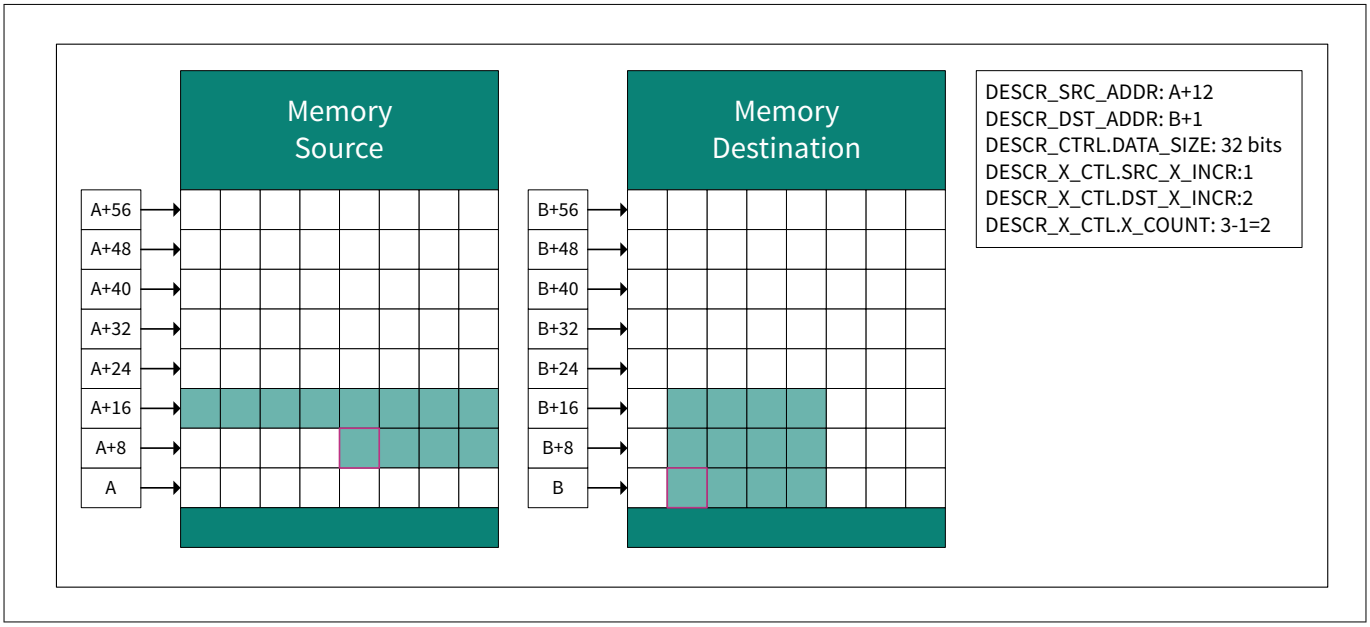


図 21　　1D メモリ転送

9.4.1.3　2D メモリ転送

記述子の設定は 2 次元の「for loop」を実行し、チャネルは (Y_COUNT+1)* (X_COUNT+1) バイトを DESCR_SRC から DESCR_DST にコピーします。2D メモリ転送は、6 つの 32 ビットワードの設定を必要とします。以下の擬似コードは、2D メモリ転送の機能を説明するものです。

必要な設定: DESCR_CTL, DESCR_SRC, DESCR_DST, DESCR_X_CTL, DESCR_Y_CTL, and DESCR_NEXT_PTR.

```
// DST_ADDR is a pointer to an object of type defined by DESCR_DST.DST_ADDR
            // SRC_ADDR is a pointer to an object of type defined by DESCR_SRC.SRC_ADDR
            // SRC_X_INCR/DST_X_INCR specifies the source and destination increment between
single transfers defined by DESCR_X_CTL
            // SRC_Y_INCR/DST_Y_INCR specifies the source and destination increment between 1D
transfers defined by DESCR_Y_CTL
            // Y_COUNT is the number of 1D transfers defined by DESCR_Y_CTL.Y_COUNT
            // t_DATA_SIZE is the type associated with the DESCR_CTL.DATA_SIZE
            // X_COUNT is the number of single transfers defined by DESCR_X_CTL.X_COUNT
            for (Y_IDX = 0; Y_IDX <= Y_COUNT; Y_IDX++) {
            for (X_IDX = 0; X_IDX <= X_COUNT; X_IDX++) {
            DST_ADDR[X_IDX * DST_X_INCR + Y_IDX * DST_Y_INCR] = (t_DATA_SIZE) SRC_ADDR[X_IDX *
SRC_X_INCR + Y_IDX * SRC_Y_INCR];
            }
            }
```

次の図に、2D メモリ転送を使用して、メモリ内の 6 つの 32 ビット データ エレメントを移動する方法を示します。

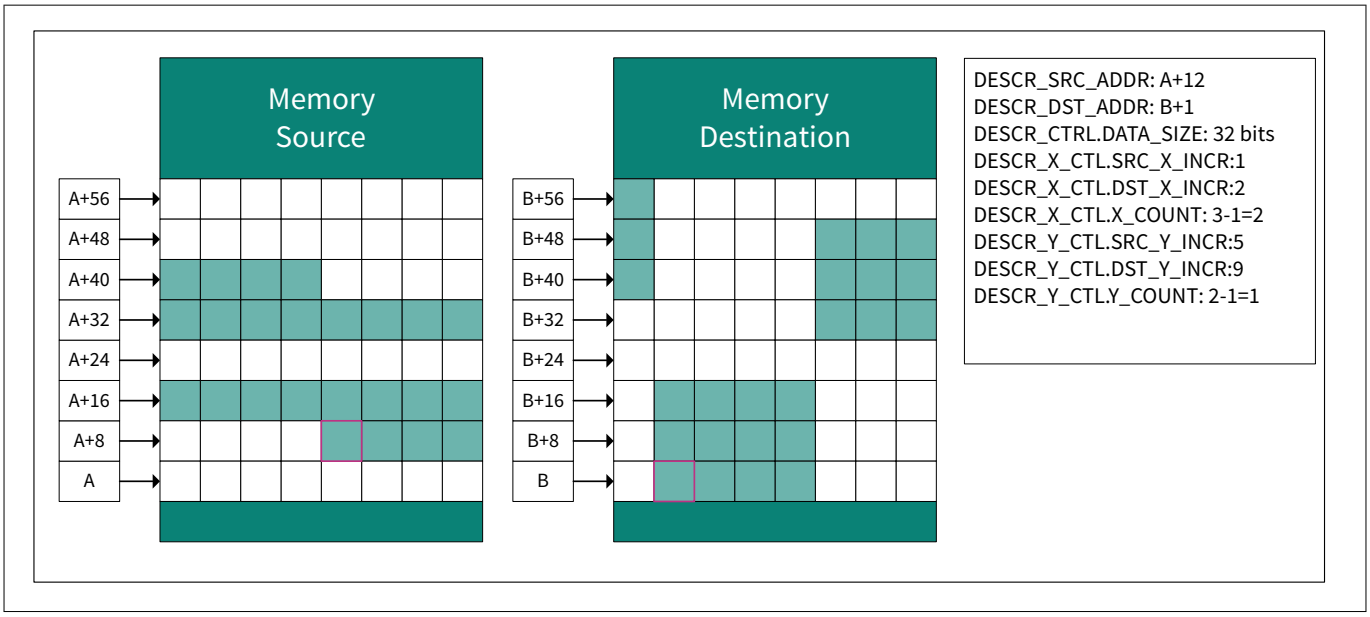


図 22　　2D メモリ転送

9.4.1.4　CRC (Cyclic Redundancy Check) 転送

この記述子設定は、1D 転送と同様に 1 次元の「for loop」を実行します。ただし、転送元データは転送先に転送されません。CRC は、ソースデータにわたって計算されます。CRC 転送には 5 つの 32 ビット ワードの設定が必要です。CRC 設定は、すべての DW チャネルで共有されるレジスタ群を通じて提供され、DW チャネルは時間的に互いに排他的な CRC 機能を使用することが前提となっています。以下の疑似コードは、CRC 転送の機能を説明するものです。

必要な設定: DESCR_CTL、DESCR_SRC、DESCR_DST、DESCR_X_CTL、DESRC_NEXT_PTR。

必要なレジスタ: CRC_CTL、CRC_DATA_CTL、CRC_POL_CTL、CRC_LFSR_CTL、CRC_REM_CTL、CRC_REM_RESULT。

```
// DST_ADDR is a pointer to an address location where the calculated CRC is stored.
            // SRC_ADDR is a pointer to an object of type defined by DESCR_SRC
            // t_DATA_SIZE is the type associated with the DATA_SIZE
            CRC_STATE = CRC_LFSR_CTL;
            for (X_IDX = 0; X_IDX <= X_COUNT; X_IDX++) {
            Update_CRC (CRC_STATE, (t_DATA_SIZE) SRC_ADDR[X_IDX * SRC_X_INCR];
            }
            DST_ADDR = CRC_STATE;
```

9.4.2 記述子サイズ

記述子のサイズはそのタイプに依存します。関連するパラメーターのみが保存されます。例えば、単一の転送記述子はパラメーター DESCR_X_CTL と DESCR_Y_CTL を含みません。チャネルがトリガーされると、記述子はメモリからチャネルのアクティブ記述子レジスタ (CHi_ACT_DESCR_CTL、CHi_ACT_DESCR_SRC、CHi_ACT_DESCR_DST、CHi_ACT_DESCR_X_CTL、CHi_ACT_DESCR_Y_CTL、CHi_ACT_DESCR_NEXT_PTR) に取り出されます。DMA は指定された DESCR_TYPE に必要な 32 ビット記述子ワードのみを取り出します。

9.4.3 記述子設定

チャネル記述子を設定する 32 ビット ワードは 6 つあります。すべてのコンフィギュレーションが 6 つの 32 ビット ワードすべてを必要とするわけではありません。メモリから記述子を読み出す場合、必要な記述子サイズのみが読み出されます。つまり、1 回の転送でメモリから読み出されるのは 4 つの 32 ビット ワードのみです。図 19 に記述子のレイアウトを示します。

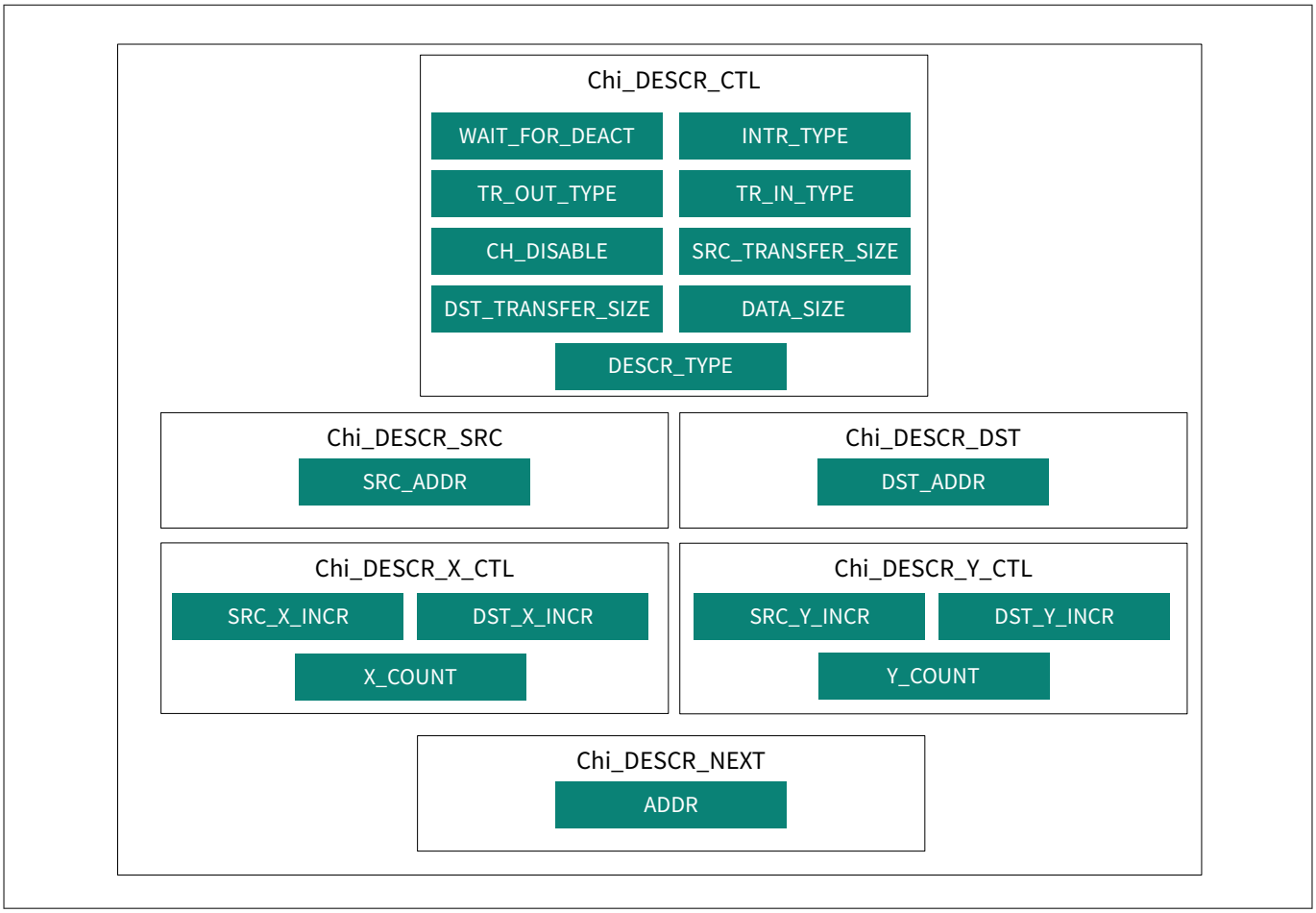


図 23　　記述子のレイアウト

9.4.3.1　記述子制御 (DESCR_CTL)

この記述子制御レジスタは、すべての転送タイプで必要とされる 7 つの異なる制御に分かれています。

- WAIT_FOR_DEACT: DMA コントローラーが、現在のトリガーに対応するデータ転送を完了した後、入力トリガーが非アクティブになるのを待つかどうかを指定します。このフィールドは、トリガーエージェントがトリガーを解除するために十分な時間を与えるために、レベル感応型トリガーに使用されます。指定できるウェイトは、ウェイトなし (パルス感応トリガー)、最大 4 サイクル、最大 16 サイクル、または無期限です。詳細については、トリガー マルチプレクサ (トリガー MUX) の章を参照してください。
- INTR_TYPE: これは、完了割込みが発生するタイミングを定義します。割込みは CHi_INTR で発生します。詳細については、チャネル割込み生成の章を参照してください。
- TR_OUT_TYPE: 出力トリガーが生成されるタイミングを定義します。
- TR_IN_TYPE: 入力トリガーは、データ転送を開始し、TR_IN_TYPE は、トリガーのアクションを定義します。
- CH_DISABLE: 現在の記述子の処理完了後にチャネルが無効化されるかどうかを指定します。これは、DESCR_NEXT_PTR の値とは独立しています。
- SRC_TRANSFER_SIZE: ソースロケーションへのバス転送サイズを指定します。DATA_SIZE で指定されたデータ サイズ、または 32 ビットに設定できます。
- DST_TRANSFER_SIZE: デスティネーション ロケーションへのバス転送サイズを指定します。DATA_SIZE で指定されたデータ サイズ、または 32 ビットに設定できます。

- DATA_SIZE: SRC_TRANSFER_SIZE または DST_TRANSFER_SIZE が DATA_SIZE を使用する場合、データ エレメント サイズを指定します。これによりデータ エレメント サイズは、8 ビット、16 ビット、32 ビットのいずれかに指定されます。
- DESCR_TYPE: 記述子の転送タイプを指定します。DMA は、シングル、1D、2D、または CRC 転送用に設定できます。詳細については記述子メモリ転送タイプを参照してください。

9.4.3.2 送信元アドレスと宛先アドレス (DESCR_SRC/DESCR_DST)

送信元と宛先のアドレス: 送信元と宛先のアドレスは記述子の各レジスタに設定されます。これらのアドレスは転送用に、送信元と宛先の開始位置アドレスを指定します。記述子が 1 つのエレメントを転送するように設定される場合、このフィールドはデータ エレメントの転送元/転送先のアドレスそのものとなります。記述子がインクリメンタルモードの転送元アドレスあるいは転送先アドレスまたはその両方で複数のエレメントを転送するように設定される場合、このフィールドは転送される最初のエレメントのアドレスを保持します。

転送タイプが CRC に設定されている場合、このフィールドは DW_CRC_LSFR_CTL レジスタのアドレスでプログラムされなければいけません。計算された CRC LFSR の状態は、AHB インターフェースでこのアドレスに書き込まれます。

9.4.3.3 記述子 X ループ制御 (DESCR_X_CTL)

DESCR_CTL.DESCR_TYPE が 1D または 2D 転送用に設定されている場合に使用されます。

- SRC_X_INCR: 各 X ループ反復のソース アドレスのインクリメントを指定します (SRC_TRANSFER_SIZE の倍数)。このフィールドは、[-2048, 2047]の範囲の符号付き数値です。このフィールドが"0"の場合、送信元アドレスはインクリメントされません。これは RX FIFO 構造からの読み出しに便利です。
- DST_X_INCR: X ループの反復ごとに、宛先アドレスのインクリメントを指定します (DST_TRANSFER_SIZE の倍数)。このフィールドは、[-2048, 2047]の範囲の符号付き数値です。このフィールドが"0"の場合、宛先アドレスはインクリメントされません。これは TX FIFO 構造への書き込みに便利です。DESCR_TYPE が CRC に設定されている場合、このフィールドは使用されません。
- X_COUNT:「X ループ」の反復回数 (マイナス 1) (X_COUNT+1 は 1D 転送のシングル転送回数)。このフィールドは[0, 255]の範囲の符号なし数値で、1 から 256 までの反復を表します。

9.4.3.4 記述子 Y ループ制御 (DESCR_Y_CTL)

DESCR_CTL.DESCR_TYPE が 2D 転送用に設定されている場合に使用されます。

- SRC_Y_INCR: SRC_TRANSFER_SIZE の倍数で) 各 Y ループ反復のソース アドレスのインクリメントを指定します。このフィールドは、[-2048, 2047]の範囲の符号付き数値です。
- DST_Y_INCR: 各 Y ループ反復の宛先アドレスのインクリメントを指定します (DST_TRANSFER_SIZE の倍数)。このフィールドは、[-2048, 2047]の範囲の符号付き数値です。
- Y_COUNT:「Y ループ」の反復回数 (マイナス 1) (X_COUNT+1) * (Y_COUNT+1) は、2D 転送におけるシングル転送の回数。このフィールドは[0, 255]の範囲の符号なし数値で、1 から 256 までの反復を表します。

9.4.3.5 記述子の次のポインタ (DESCR_NEXT_PTR)

記述子リストの次の記述子のアドレス。このフィールドが"0"のとき、この記述子は記述子リストの最後の記述子です。

9.4.4 記述子転送サイズ

転送用の転送ワード幅は記述子のパラメーターの転送/データ サイズにより設定することが可能です。この設定は転送元の転送サイズ、転送先の転送サイズおよびデータサイズにわかれます。データ サイズ パラメーター

(DESCR_CTL.DATA_SIZE) は転送のためのバス幅を設定します。DESCR_CTL.SCR_TRANSFER_SIZE および DESCR_CTL.DST_TRANSFER_SIZE により設定された転送元と転送先のサイズは、DATA_SIZE の値または 32 ビットの値を有することができます。DESCR_CTL.DATA_SIZE は 32 ビット、16 ビットまたは 8 ビットに設定できます。

DMA の転送元と転送先の転送サイズは、移動しなければいけないデータの幅に関係なく、転送元と転送先のアドレス可能な幅と一致する必要があります。DATA_SIZE パラメーターは、実際のデータの幅に対応します。例えば、16 ビット PWM が DMA データの転送先として使用される場合、DST_TRANSFER_SIZE は PWM レジスタの幅に合わせて 32 ビットに設定されなければいけません。なぜならば、TCPWM ブロック (およびほとんどのペリフェラル) のペリフェラル レジスタ幅は常に 32 ビットだからです。しかしこの例では、16 ビット PWM が 2 バイトのデータしか使用しないため、転送先の DATA_SIZE が 16 ビットに設定されている可能性があります。

表 43 は、転送サイズの可能な組み合わせをまとめたものです。

表 43　　データ転送の幅

DATA_SIZE	SCR_TRANSFER_SIZE	DST_TRANSFER_SIZE	通常の使用法	説明
8 ビット	8 ビット	8 ビット	メモリ→メモリ	データの操作なし
8 ビット	32 ビット	8 ビット	周辺デバイス→メモリ	転送元より送信される上位の 24 ビットは切り捨て
8 ビット	8 ビット	32 ビット	メモリ→周辺デバイス	転送先で上位の 24 ビットに 0 を挿入
8 ビット	32 ビット	32 ビット	周辺デバイス→周辺デバイス	転送元より送信される上位の 24 ビットが切り捨てられ、その部分に 0 を挿入
16 ビット	16 ビット	16 ビット	メモリ→メモリ	データの操作なし
16 ビット	32 ビット	16 ビット	周辺デバイス→メモリ	転送元より送信される上位の 16 ビットは切り捨て
16 ビット	16 ビット	32 ビット	メモリ→周辺デバイス	転送先で上位の 16 ビットに 0 を挿入
16 ビット	32 ビット	32 ビット	周辺デバイス→周辺デバイス	転送元より送信される上位の 16 ビットが切り捨てられ、その部分に 0 を挿入
32 ビット	32 ビット	32 ビット	周辺デバイス→周辺デバイス	データの操作なし

9.4.5　記述子チェーン

記述子が連鎖している場合、それは記述子リストとして知られています。DESCR_NEXT_PTR フィールドは、チェーンの次の記述子へのポインタを含みます。このポインタを 8 バイトの倍数に揃えます。DMA チャネルは、記述子の実行が完了すると、現在の記述子に指定されている次の記述子を実行します。チェーンの最後の記述子で、DESCR_NEXT_PTR は“0” (NULL ポインタ) に設定されます。この場合、エラーが発生するか、チャネルまたはコントローラーが SW によって無効化されるまで、実行は無期限に継続されます。循環リストは、それ自身を指す記述子を含みます。

9.5 トリガー

各 DMA チャネルには、1 つの入力トリガーと 1 つの出力トリガーがあります。これらのトリガーは、データ転送を開始したり、転送終了時に他のペリフェラルに信号を送るために使用されます。入力トリガーと出力トリガーの両方が記述子ごとに設定されます。

9.5.1 入力トリガー

入力トリガーによりデータ転送が開始されます。このトリガーは、転送元、転送先、CPU SW、または他のペリフェラルから発生することがあります。トリガーは CPU アクティブ/スリープ機能を提供し、CPU/システム ディープスリープおよびハイバネート電力モードでは利用できません。

各チャネルは、DMA コントローラーの外部にあるトリガーマルチプレクサを介して、特定のシステムトリガーに接続します。トリガー マルチプレクサ ブロックは、トリガー信号を指定されたデスティネーションにルーティングします。DMA はそのような目的地のひとつです。トリガー マルチプレクサ ブロックの詳細については、トリガー マルチプレクサ (トリガー MUX) の章を参照してください。

DMA は、TR_IN_TYPE フィールドと WAIT_FOR_DEACT フィールドを使用して、トリガー入力で何が起こるかを制御します。これらのフィールドは共に DESCR_CTL レジスタの一部です。

TR_IN_TYPE: 各 DMA チャネルの入力トリガーは、DESCR_CTL.TR_IN_TYPE を使用して設定します。表 44 は異なるトリガータイプについて説明しています。

表 44 入力トリガー タイプ

トリガータイプ	説明
タイプ 0	トリガーは、シングル転送を実行します。記述子のタイプは、シングル、1D、または 2D です。
タイプ 1	トリガーは、シングル 1D 転送を実行します。記述子タイプが「single」の場合、トリガーはシングル転送を実行します。記述子タイプが「1D」または「2D」の場合、トリガーは 1D、または 2D 転送を実行します。
タイプ 2	トリガーは、現在の記述子を実行します。
タイプ 3	トリガーは、現在の記述子の実行をもたらし、(別の入力トリガーを必要とせずに) 次の記述子の情報を用いて次の記述子の実行を継続します。

WAIT_FOR_DEACT: DMA には、入力トリガーを停止するためのコントロールがあります。TR_IN_TYPE に基づく DMA 転送が完了すると、データ転送エンジンはトリガー解除の状態をチェックします。第 2 のトリガーによるデータ転送は、第 1 のトリガーの停止後にのみ開始されます。DESCR_CTL.WAIT_FOR_DEACT パラメーターは、トリガー信号がいつ非活性化されたとみなされるかを決定します。最初の DMA 転送は、トリガーがアクティブになった時にアクティブになりますが、転送はトリガーが非アクティブになるまで完了したとはみなされません。このフィールドは、コントローラーのデータ転送を、トリガーを生成したエージェントと同期させるために使用されます。異なるトリガー無効化は 表 45 にリストされています。

表 45 トリガー停止タイプ

非アクティブ化タイプ	説明
タイプ 0	パルストリガー: 非アクティブ化を待ちません。トリガーが検出されると、転送が開始されます。転送完了後、トリガーがまだアクティブであれば、それは別のトリガーとみなされ、後続の転送が即座に開始されます。

(続く)

表 45 (続き) トリガー停止タイプ

非アクティブ化タイプ	説明
タイプ 1	レベルセンシティブ: 非活性化とみなされる転送の後、4 クロックの低速サイクルを待ちます。トリガーが検出されると、転送が開始されます。転送完了後、トリガーがまだアクティブであれば、4 サイクル待った後、別のトリガーとみなされます。そして、その後の転送が開始されます。トリガーに対応する転送は、4 つの追加サイクルが終了した時点で初めて完了したとみなされます。トリガー出力イベントも、この遅延に基づいて影響を受けます。このパラメーターは、各トリガー トランザクションに 4 サイクルの遅延を与え、スループットに影響を与えます。
タイプ 2	レベルセンシティブ: 非アクティブ化とみなされる転送の後、16 スロー クロック サイクルを待ちます。トリガーが検出されると、転送が開始されます。転送が完了した後、トリガーがまだアクティブであれば、16 サイクル待った後、別のトリガーとみなされます。そして、その後の転送が開始されます。トリガーに対応する転送は、16 の追加サイクルが終了した時点で初めて完了したとみなされます。トリガー出力イベントも、この遅延に基づいて影響を受けます。このパラメーターは、各トリガー トランザクションに 16 サイクルの遅延を与え、スループットに影響を与えます。
タイプ 3	パルストリガー: 無期限に非アクティブになるのを待ちます。トリガー信号が非アクティブになった後、DMA 転送が開始されます。次の転送は、トリガーが Low になり、再び High になった場合にのみ開始されます。トリガー信号がアクティブのままであるか、2 つのトランザクション間でゼロに遷移しない場合、単に DMA チャネルをストールさせるだけです。

DMA 転送はソフトウェアでトリガーすることもできます。CHi_TR_CMD に‘1’を書き込むとトリガーが発生し、チャネルに保留中のトリガーが設定されます。チャネルが有効になっていなければいけません。

9.5.2 保留中のトリガー

有効化されたチャネルがトリガーされた (CHi_STATUS.PEDNING が 1 に設定された) とき、チャネルは保留中とみなされます。トリガーされると、DMA コントローラーは、他のチャネルが保留状態でなければ、転送を開始します。

保留状態のチャネルが複数ある場合、DMA は優先度デコードを使用して、最も優先度の高い保留チャネルを決定します。このチャネルは以下のように決定されます。

1. まず、保留中のチャネルを持つ最優先グループが特定されます。
2. 次に、この優先順位グループ内では、(優先順位グループ内で) 下位のチャネル インデックスに最も高い優先順位を与えるラウンドロビン アービトレーションが適用されます。チャネルの優先度は、CHi_CTL.PRIO で設定されます。

9.5.3 出力トリガー

各チャネルには出力トリガーがあり、少なくとも 2 クロックの低速サイクルの間 High となります。このトリガーは、データ転送の完了時に生成されます。システムレベルでは、これらの出力トリガーをトリガー マルチプレクサ コンポーネントに接続できます。トリガー マルチプレクサ ブロックは、DMA ブロックの外側にあります (トリガー マルチプレクサ ブロックの項を参照)。この接続により、DMA コントローラー出力トリガーを DMA コントローラー入力トリガーに接続できます。言い換えれば、あるチャネルで転送が完了すると、別のチャネルが活性化したり、同じチャネルが再活性化することすらあります。

TR_OUT_TYPE: DMA チャネル出力トリガーは、CHi_DESCR_CTL.TR_OUT_TYPE を用いて設定されます。次の表に異なるトリガーの種類を示します。

表 46 出力トリガーの種類

トリガータイプ	説明
タイプ 0	出力トリガーはシングル転送の後に生成されます。
タイプ 1	1D 転送が 1 回実行された後に、出力トリガーが生成されます。記述子タイプが「single」の場合、出力トリガーは 1 回の転送の後に生成されます。記述子タイプが「1D」、「CRC」、「2D」の場合、出力トリガーは 1D 転送の実行後に生成されます。
タイプ 2	現在の記述子の実行後に出力トリガーが発生します。
タイプ 3	出力トリガーは、記述子リストの実行後 (現在の記述子の実行後、かつ現在の記述子 DESCR_NEXT_PTR.ADDR が“0”) に生成されます。

9.6 性能

DMA 転送のクロックサイクル推定値を計算できます。この推定値は DESCR_CTL.DESCR_TYPE で指定された転送タイプに依存します。

9.6.1 シングル転送

DMA コントローラーは、記述子のロードやデータ エレメントのロード/ストアを行うために、バス インフラストラクチャー上で待機状態がなければ、1 回のデータ転送に 14 クロックサイクル (clk) を要します。以下はクロックサイクルの内訳です。

- トリガーの同期に 2 サイクル
- 検出、優先順位デコード、チャネル保留のための 1 サイクル
- コントローラーのステートマシンが保留中のチャネルを検出し、開始するまでの 1 サイクル
- 記述子チャネル制御情報が DW チャネル コンフィギュレーション メモリから DW 制御フリップフロップにロードされるための 2 サイクル
- 記述子をロードするための 4 サイクル (DESCR_CTL、DESCR_SRC、DESCR_DST)
- 次のポインタをロードするための 1 サイクル (DESCT_NEXT_PTR)
- 送信元からデータをロードし、送信先でデータを保存するための 3 サイクル

9.6.2 1D 転送

以下は、n 個のデータ エレメントの 1 次元転送を完了するためのサイクル数の計算式です。

サイクル数 = 12 + n*3 + m、ここで m は、記述子/データのロード/ストア用に DW コントローラーで見られる待機状態の総数です。

以下は、上記の式における 12 固定サイクルの内訳です。

- トリガーの同期に 2 サイクル
- 検出、優先順位デコード、チャネル保留のための 1 サイクル
- コントローラーのステートマシンが保留中のチャネルを検出し、開始するまでの 1 サイクル
- 記述子チャネル制御情報が DW チャネル コンフィギュレーション メモリから DW 制御フリップフロップにロードされるための 2 サイクル
- 記述子をロードするための 4 サイクル (DESCR_CTL、DESCR_SRC、DESCR_DST)
- 記述子 X 制御 (DESCR_X_CTL) をロードするための 1 サイクル
- 次のポインタをロードするための 1 サイクル (DESCT_NEXT_PTR)

ソースからデスティネーションへの各データ エレメントの転送には 3 サイクルかかり、式中では n*3 で表されます。

9.6.3 2D 転送

以下は、n 個のデータ エレメントの 2 次元転送を完了するためのサイクル数の計算式です。

サイクル数 = 13 + n*3 + m、ここで m は、記述子/データのロード/ストア用に DW コントローラーで見られる待機状態の総数です。

以下は、上記の式における 13 固定サイクルの内訳です。

- トリガーの同期に 2 サイクル
- 検出、優先順位デコード、チャネル保留のための 1 サイクル
- コントローラーのステートマシンが保留中のチャネルを検出し、開始するまでの 1 サイクル
- 記述子チャネル制御情報が DW チャネル コンフィギュレーション メモリから DW 制御フリップフロップにロードされるための 2 サイクル
- 記述子をロードするための 4 サイクル (DESCR_CTL、DESCR_SRC、DESCR_DST)
- 記述子 X 制御 (DESCR_X_CTL) をロードするための 1 サイクル
- 記述子 Y 制御 (DESCR_Y_CTL) をロードするための 1 サイクル
- 次のポインタをロードするための 1 サイクル (DESCT_NEXT_PTR)

ソースからデスティネーションへの各データ エレメントの転送には 3 サイクルかかり、式中では n*3 で表されます。

9.6.4 CRC 転送

以下は、n 個のデータ ワードの CRC 計算を完了するためのサイクル数の計算式です。

サイクル数 = 17 + n*9 + m、ここで m は、記述子/データのロード/ストアのために DW コントローラーで見られる待機状態の総数です。

各データ ワードの CRC 計算には‘9’クロック サイクルがかかり、これは式中では n*9 で表されます (CRC 記述子設定がワード転送を使用すると仮定)。余分な 17 サイクルには、トリガーの同期、検出、記述子のロード、および計算された CRC を送信先に戻す格納が含まれます。

10 トリガー マルチプレクサ (トリガー MUX)

この MCU デバイスのすべてのペリフェラルは、トリガー信号を使用して相互接続されています。トリガー信号は、ペリフェラルがイベントの発生や状態遷移を示すために使用します。

これらのトリガーは、他のペリフェラルのアクションを実行または開始するために使用されます。トリガーマルチプレクサは、ユーザーがトリガーをソースペリフェラルからデスティネーションにルーティングさせるのをサポートします。トリガーはあるペリフェラルによって生成され、別のペリフェラルによって消費されます。割込みとは異なり、トリガーはペリフェラルと Arm® CPU コアの間ではなく、ペリフェラル間の同期に使われます。

10.1 特長

トリガー マルチプレクサには次のような特長があります。

- あるペリフェラルから別のペリフェラルへ任意のトリガー信号を接続可能
- マルチプレクサ ベースのトリガー グループと 1 対 1 のトリガー グループを提供
- ブロック内の任意の信号をトリガーできるソフトウェア トリガーに対応
- 反転やエッジ/レベル検出などのトリガー操作機能を持つトリガー マルチプレクサをハードウェアで設定する機能
- デバッグモードでトリガーをブロックする機能

10.2 アーキテクチャ

トリガーは、(ソフトウェアベースの同期とは対照的に) ハードウェアでペリフェラルの機能を同期させるために使用されます。ペリフェラル DMA は、あるアドレス位置から別のアドレス位置へのデータ エレメントの転送を開始するためにトリガーを使用します。例えば、「ADC 変換完了」イベントは、ADC モジュールから SRAM メモリ位置へ ADC サンプルの転送を開始します。

トリガーは、FIFO レベルなどの状態やアクションの完了などのイベントを示すために、周辺ブロックによって生成されるデジタル信号です。これらのトリガー信号は、通常、他の周辺ブロックにおける他のアクションのイニシエーターとして機能します。例として、3 チャネルをサンプリングする ADC 周辺ブロックがあります。変換が完了すると、トリガー信号が生成され、ADC データをメモリ バッファに転送する DMA チャネルがトリガーされます。図 24 に、この例を示します。

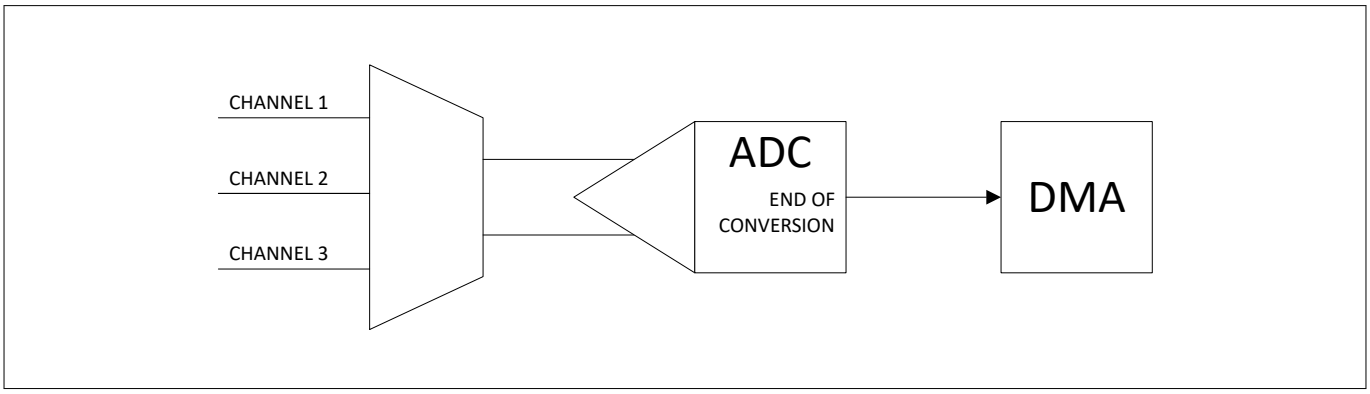


図 24 トリガー信号例

PSOC™ Control C3 MCU には複数のペリフェラル ブロックがあり、各ブロックはシステムの実装に応じて、トリガー信号によって他のブロックと接続できます。これをサポートするため、可能性のあるソースからデスティネーションにトリガー信号をルーティングするために使用される一連のマルチプレクサであるハードウェアがあります。このハードウェアは、トリガー マルチプレクサ ブロックと呼ばれます。トリガー マルチプレクサは、MCU 内のどのペリフェラル ブロックから発せられるトリガー信号にも接続でき、それを他のペリフェラルにルーティングして、宛先のペリフェラル ブロックの動作を開始させたり、影響を与えたりできます。

トリガーには 2 種類あります。

- ハイアクティブ、レベル センシティブ トリガー: このタイプは通常、ペリフェラルの状態を反映します。例えば、tr_fifo_empty は FIFO が空であることを示します。FIFO が空である限り、トリガーは'1'のままです。このタイプは、プロデューサがトリガーを非アクティブにするためにトリガーとなる、消費側によるアクションが必要です。例えば、tr_fifo_empty は、関連する FIFO にデータ エレメントを書き込むことで非活性化されます。このトリガー タイプは、どのクロックでも発生させられます。
- 立ち上りエッジ、パルストリガー: このタイプは通常、イベントの発生を反映します。例えば、tr_adc_done は SAR ADC がサンプルを変換したことを示します。このトリガータイプは、周辺システムインターフェースクロックで生成されます。このトリガーは、周辺システムインターフェースクロックの 2 サイクルのパルスの間、'1'のままで、自ら'0'に戻ります。

トリガーの消費側が即座にトリガーに反応できない場合、トリガーが発生したことを保持しておく必要があります。トリガーの消費側が複数のトリガーに連続して直面した場合、複数のトリガーを保持しなければいけないか、トリガーを無視することにする場合があります。

トリガーは、プロデューサーペリフェラルと消費側ペリフェラル間の非同期信号として扱う必要があり、消費側ペリフェラルは入力トリガーを同期化する必要があります。さらに、パルストリガーの場合、消費側ペリフェラルは立ち上りエッジ検出を行い、トリガーの発生を保持できます。

抽象度の高いレベルでは、トリガーは、プロデューサと消費側ペリフェラル間のワイヤ接続です。しかし、より詳細なレベルでは、いくつかの処理段階が区別されます。

一般に、トリガー入力信号は、周辺動作の完了や周辺イベントを示します。一般的に、トリガー出力信号は周辺動作を開始させます。

ペリフェラルの一部として統合されたマルチプレクサ コンポーネントではなく、(ペリフェラル グループの一部として) 標準的なマルチプレクサ コンポーネントを採用したことには、次のような理由があります。

- 標準的なマルチプレクサ コンポーネントは、一貫したユーザー インターフェースを強制します。特定のトリガー出力信号 (周辺入力信号) に対するトリガー入力信号の選択は、すべてのコンポーネントで同じです。

- 出力トリガー信号の起動をソフトウェアで制御するなど、マルチプレクサ コンポーネントが提供する追加機能は、すべてのコンポーネントに利益をもたらします。

トリガーモジュールは多重化機能を提供します。トリガー出力信号を使用するペリフェラルにおいて、以下の機能のいずれかを実行することが要求される場合があります。

- トリガー信号の周辺クロックドメインへの同期
- トリガー信号のエッジ検出
- トリガー信号の保存/保持

トリガーコンポーネントは、複数のトリガーグループで構成されます。トリガーグループには 2 つのタイプがあります。

- 1 対 1 ベースのコネクティビティ グループ: このグループタイプは、ペリフェラル入力トリガーを特定のペリフェラル出力トリガーに接続します。
- マルチプレクサベースの接続グループ: このタイプは、周辺入力トリガーを複数の周辺出力トリガーに接続します。選択はソフトウェアの制御下 (PERI_TR_GR_TR_CTL.TR_SEL) に置かれています。

PSOC™ Control C3 MCU 用のトリガー コンポーネントは、15 個のマルチプレクサ ベースのトリガー グループと 3 個の 1 対 1 トリガー グループを提供します。

10.3 機能説明

トリガーコンポーネントは、トリガー信号を多重化します。トリガー入力ソース信号は、通常、周辺出力信号です。トリガー出力先信号は、通常、周辺入力信号です。

トリガー入力ソース信号の例:

- TCPWM 出力信号: 例えば、カウンターが事前にプログラムされた制限値に達した場合
- ADC 出力信号: 例えば、ADC 変換が完了した場合

- I/O 入力信号
- 転送の完了を示す DMA コントローラーの出力信号

トリガー出力先信号の例は以下のとおりです。

- TCPWM 入力信号: 例えば「start a counter」
- ADC 入力信号: 例えば「start an ADC conversion」
- 入出力信号
- 転送を開始するための DMA コントローラー入力信号

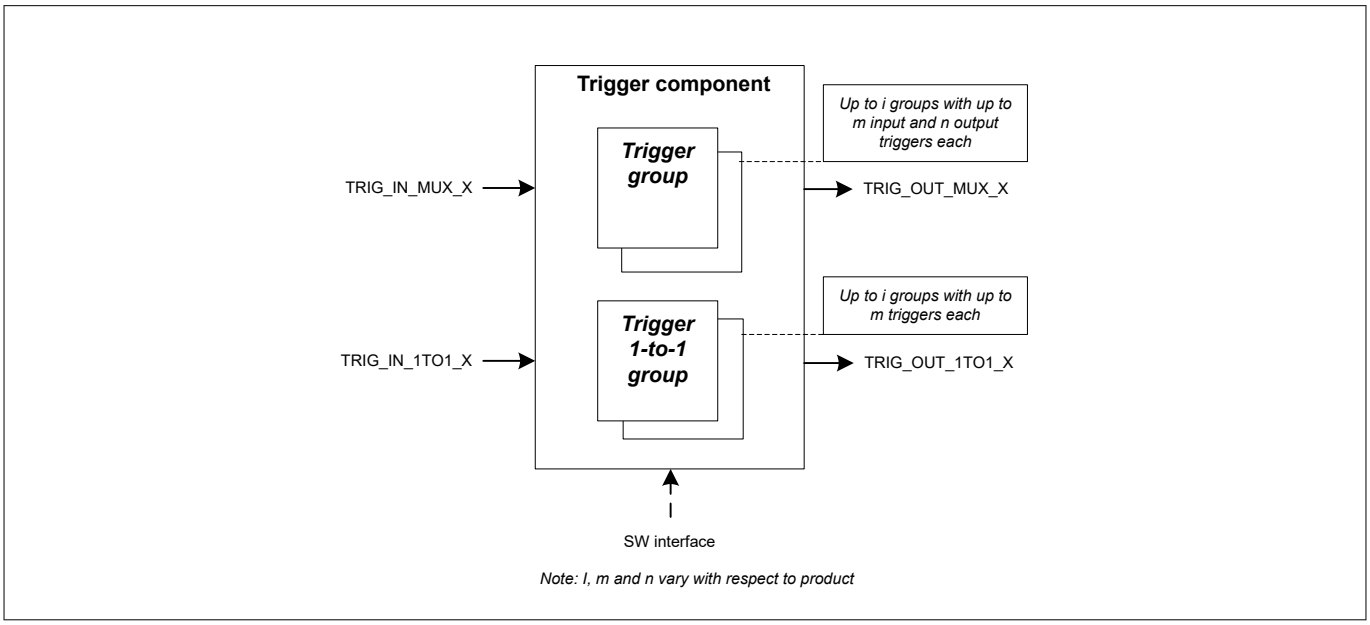


図 25　　トリガー設定のパラメーター

各グループは、特定のペリフェラルのトリガー入力に関連付けられています。図 25 に概要を示します。

トリガーデスティネーションペリフェラルは、トリガーソース ペリフェラル コンポーネント TRIG_IN_MUX_x と TRIG_IN_1TO1_x に接続されます。トリガーソースペリフェラルは、トリガーデスティネーション ペリフェラルコンポーネント TRIG_OUT_MUX_x と TRIG_OUT_1TO1_x に接続されています。PERI のトリガーコンポーネントが提供する機能と、ペリフェラルが提供する機能を区別することが重要です。

トリガーコンポーネントは、以下の機能を提供する (TRIG_IN_MUX_x と TRIG_IN_1TO1_x):

- マルチプレクサベースの接続グループでは、入力トリガー TRIG_IN_MUX_x は、各出力トリガー TRIG_OUT_MUX_x に対して選択されます。グループ[i]内のすべての出力トリガーは同じ入力トリガーを共有していることに注意してください。1 対 1 ベースのグループの場合、入力トリガー TRIG_IN_1TO1_x は、出力トリガー TRIG_OUT_1TO1_x に接続されます。
- トリガー起動はソフトウェア制御。このコントロールにより、特定のトリガーを作動できます。レベル感応型トリガーの場合、トリガーの起動は完全にソフトウェア制御の下にあります。エッジセンシティブトリガーの場合、トリガーは CLK_PERI の High/'1'サイクルで 2 回起動します。
- デバッグモード (通常、CPU の停止時) では、トリガー伝搬をブロックできます。これにより、デバッグ停止中にトリガー消費側ペリフェラルが入力トリガーを取得しないようにできます。デバッグモードは、CPUSS の CTI トリガー出力に接続されたレベルトリガー入力 x_DEBUG_FREEZE_TR_IN1 によって示されます。
- 出力トリガークロックドメインに転送するパルストリガーを可能にするために、ハードウェアエッジ検出が提供されます。このハードウェアは、出力トリガークロックドメインよりも高いクロック周波数で動作する入力トリガーをサポートするために、非同期エッジ検出を実行します。この機能は、パルストリガーを対象としています (レベルトリガーは通常、エッジ検出機能をバイパスします)。

- ハードウェアトリガー信号のレベル反転
- ほとんどのトリガー マルチプレクサは、すべての出力信号が共通のペリフェラルに接続されています。操作ロジックは、そのペリフェラルのクロックに接続されます。しかし、デバッグ マルチプレクサには、多くのクロックドメインの出力があります。これは、ほとんどのデスティネーションがレベルセンシティブな信号、I/O、またはクロック操作を必要としないその他のデスティネーションであるためです。

ペリフェラルは以下の機能を提供することがあります。

- 出力トリガーの同期 TRIG_OUT_MUX_x と TRIG_OUT_1TO1_x
- 同期出力トリガーのエッジ検出
- トリガー信号の保存/保持

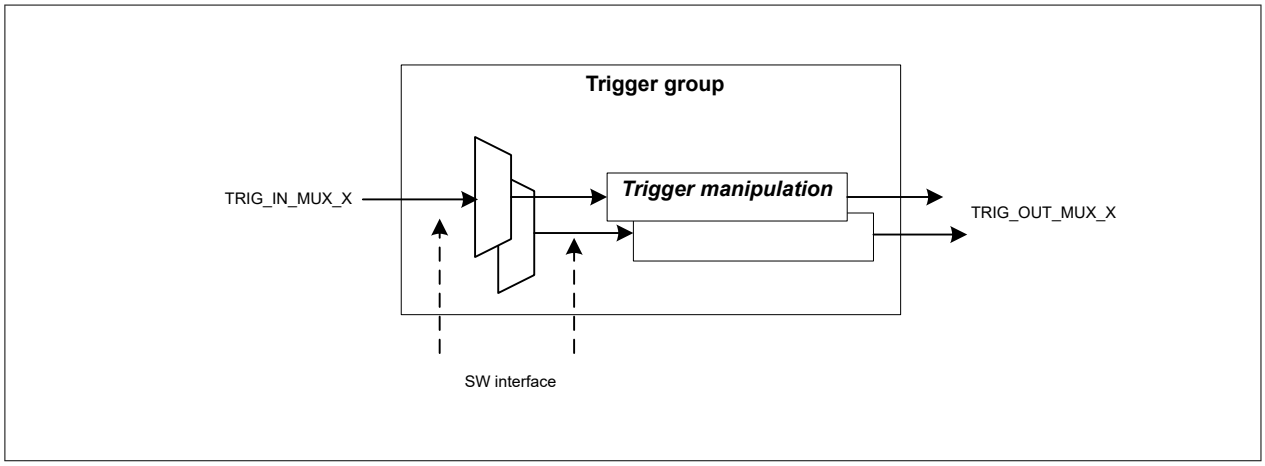


図 26　　トリガーグループ

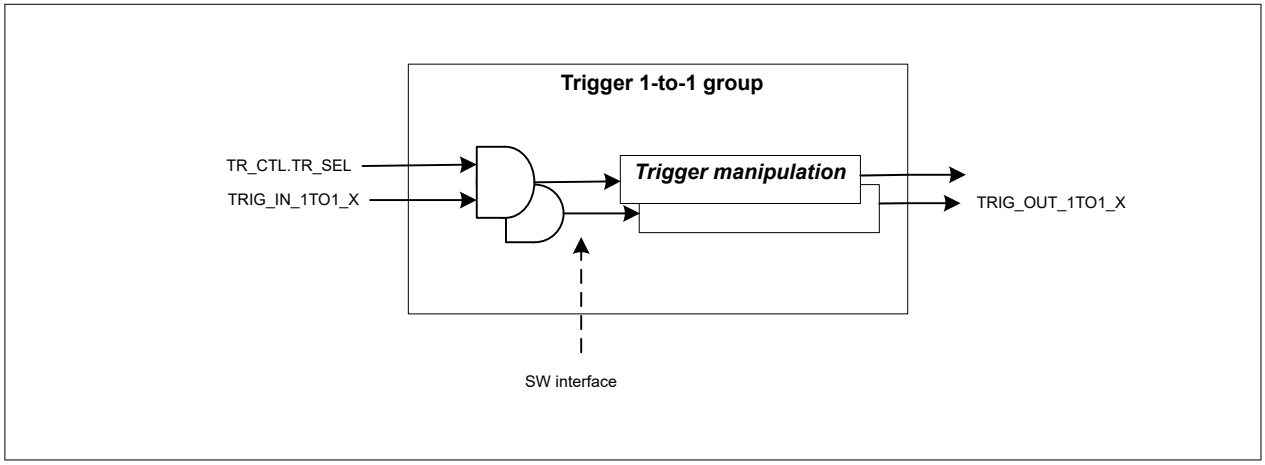


図 27　　トリガー 1 対 1 グループ

注: *1 対 1 グループには、入力トリガーを無効にする AND ゲート機能があります。*

表 47 に PSOC™ Control C3 のトリガー マルチプレクサの接続の概要を示します。

表 47　　トリガーマルチプレクサ接続の概要

入力	トリガー マルチプレクサ (MUX)	出力
IOSS[0:49]	MOTIF トリガー	TCPWM.motif_in[0:7]
HPPASS tr_level_out[0:7]		

(続く)

表 47 (続き) トリガーマルチプレクサ接続の概要

入力	トリガー マルチプレクサ (MUX)	出力
8x16 ビット TCPWM[GP1 カウンター out0, out1]		
HPPASS[all]	TCPWM0 トリガー	TCPWM.tr_all[0:3]
LPCOMP[comp0, comp1]		
SCB[all]	TCPWM1 トリガー	TCPWM.tr_all[4:15]
CPUSS.tr_fault[0]		
DW0, DW1[all]		
CANFD.tmp_rtp_out		
TCPWM[out1 のみ]		
IOSS[0:49]		
TCPWM.tr_motif_out[0:7]		
IOSS[0:49]	TCPWM2 トリガー	TCPWM.tr_one[トリガー 3 (20×カウンター)]
TCPWM[out0 のみ]	TCPWM3 トリガー	TCPWM.tr_one[トリガー 4 (20×カウンター)]
TCPWM[out1 のみ]	HPPASS トリガー	HPPASS.tr_muxed_in[all]
IOSS[0:49]		
TCPWM tr_motif_out[0:7]		
TCPWM[all out0]	PDMA0	DW0 - 16 チャネル
IOSS[0:49]		
SCB[6x3] (すべて)		
HPPASS[all]		
LPCOMP[comp0, comp1]		
CORDIC.done_mxcordic		
CRYPTOLITE.tr_trng_bitstream		
CANFD[all]		
CPUSS.tr_fault[0]		
DEBUG600.CTI_out[0:1]		
PDMA0 と同じ「入力」	PDMA1	DW1 - 16 チャネル
DW0[0-3]	HSIOM0 トリガー	IOSS.peri_tr_io_output_out[0:49]
DW0[all]		
DW1[all]		
CPUSS.CTI_out[0:1]		

(続く)

表 47 (続き) トリガーマルチプレクサ接続の概要

入力	トリガー マルチプレクサ (MUX)	出力
SCB[all]		
CANFD.tmp_rtp_out		
TCPWM[all out1]		
TCPWM tr_line[0:19]		
TCPWM tr_line_compl[0:19]		
TCPWM tr_motif_out[0:7]		
HPPASS[all]		
LPCOMP[comp0, comp1]		
CORDIC.done_mxcordic		
CRYPTOLITE.tr_trng_bitstream		
IOSS.peri_tr_io_input_in[0:49]		
TCPWM[line][0:3]	HSIOM1 トリガー	IOSS.peri_tr_io_output_out [50:57]
TCPWM[line_compl][0:3]		
TCPWM[line][256:259]	HSIOM2 トリガー	IOSS.peri_tr_io_output_out [58:65]
TCPWM[line_compl][256:259]		
TCPWM[line][512:515]	HSIOM3 トリガー	IOSS.peri_tr_io_output_out[66:73]
TCPWM[line_compl][512:515]		
DW0[all]	CPUSS_CTI	CPUSS.CTI_in[0:1]
DW1[all]		SRSS.debug_freeze_mcwdt
DEBUG600.CTI_out[0:1]		
CPUSS.tr_fault[0]		
TCPWM[all out0, out1]		
IOSS[all pads]		
SCB[all]		
CANFD.tmp_rtp_out[0:1]		
HPPASS[all]		
LPCOMP[comp0, comp1]		
CORDIC.done_mxcordic		
CORDIC.start_mxcordic		
TCPWM tr_motif_out[0:7]		
DEBUG600.CTI_out[0:1]	PERI_FREEZE	TCPWM.debug_freeze
CPUSS.tr_fault[0]		PERI.dbg_freeze
CANFD.tmp_rpt_out[0:1]	CANTT	CANFD.evt_swt_in[0:1]

注: DMA は、このセクションのレジスタ名とトリガーの両方で、データワイヤ(DW) として参照されます。

PSOC™ Control C3 MCU の詳細なトリガーマルチプレクサの接続は、以下の表を参照してください。

表 48 PERI トリガー ソース グループ 0 (PDMA0 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:4	PDMA0_TR_OUT[0:3]	P-DMA0 を PDMA0 にチェーンできるようにします。チャネル 0〜3 はチェーン用。
5	TCPWM0_GRP0_OUT0_[0]	PDMA1 への 32 ビット TCPWM カウンター
6	TCPWM0_GRP0_OUT0_[1]	
7	TCPWM0_GRP0_OUT0_[2]	
8	TCPWM0_GRP0_OUT0_[3]	
9	TCPWM0_GRP1_OUT0_[0]	PDMA1 への 16 ビット TCPWM カウンター
10	TCPWM0_GRP1_OUT0_[1]	
11	TCPWM0_GRP1_OUT0_[2]	
12	TCPWM0_GRP1_OUT0_[3]	
13	TCPWM0_GRP1_OUT0_[4]	
14	TCPWM0_GRP1_OUT0_[5]	
15	TCPWM0_GRP1_OUT0_[6]	
16	TCPWM0_GRP1_OUT0_[7]	
17	TCPWM0_GRP2_OUT0_[0]	
18	TCPWM0_GRP2_OUT0_[1]	
19	TCPWM0_GRP2_OUT0_[2]	
20	TCPWM0_GRP2_OUT0_[3]	
21	TCPWM0_GRP2_OUT0_[4]	
22	TCPWM0_GRP2_OUT0_[5]	
23	TCPWM0_GRP2_OUT0_[6]	
24	TCPWM0_GRP2_OUT0_[7]	
25:74	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)
75	SCB_I2C_SCL[0]	
76	SCB_TX[0]	SCB TX/RX DMA リクエスト
77	SCB_RX[0]	
78	SCB_I2C_SCL[1]	

(続く)

表 48 (続き) PERI トリガー ソース グループ 0 (PDMA0 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
79	SCB_TX[1]	
80	SCB_RX[1]	
81	SCB_I2C_SCL[2]	
82	SCB_TX[2]	
83	SCB_RX[2]	
84	SCB_I2C_SCL[3]	
85	SCB_TX[3]	
86	SCB_RX[3]	
87	SCB_I2C_SCL[4]	
88	SCB_TX[4]	
89	SCB_RX[4]	
90	SCB_I2C_SCL[5]	
91	SCB_TX[5]	
92	SCB_RX[5]	
93:94	CTI_TR_OUT[0:1]	CPUSS デバッグイベント
95	MXCORDIC_DONE	MXCORDIC がトリガーを完了
96	CANFD_TT_TR_OUT[0]	CAN TT 同期出力
97	CANFD_TT_TR_OUT[1]	CAN TT 同期出力
98	CAN_DBG0	
99	CAN_DBG1	
100	CAN_FIFO0_CH0	
101	CAN_FIFO0_CH1	
102	CAN_FIFO1_CH0	
103	CAN_FIFO1_CH1	
104	LPCOMP_DSI_COMP0	LPCOMP コンパレーター#0
105	LPCOMP_DSI_COMP1	LPCOMP コンパレーター#1
106	CPUSS_TR_FAULT[0]	
107	TRNG_BITSTREAM	TRNG_BITSTREAM
108:111	PASS_FIFO[0:3]	FIFO レベル用 PERI へのトリガー
112:119	PASS_PULSE[0:7]	HPPASS パルストリガー用 PERI へのトリガー
120:127	PASS_LEVEL[0:7]	レンジ レベルの PERI へのトリガー

表 49 PERI トリガー宛先グループ 0 (PDMA0 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:15	PDMA0_TR_IN[0:15]	P-DMA0 へのトリガー

表 50 PERI トリガー ソース グループ 1 (PDMA1 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:4	PDMA1_TR_OUT[0:3]	P-DMA1 を PDMA1 にチェーンできるようにします。チャネル 0～3 はチェーン用。
5	TCPWM0_GRP0_OUT0_[0]	PDMA1 への 32 ビット TCPWM カウンター
6	TCPWM0_GRP0_OUT0_[1]	
7	TCPWM0_GRP0_OUT0_[2]	
8	TCPWM0_GRP0_OUT0_[3]	
9	TCPWM0_GRP1_OUT0_[0]	PDMA1 への 16 ビット TCPWM カウンター
10	TCPWM0_GRP1_OUT0_[1]	
11	TCPWM0_GRP1_OUT0_[2]	
12	TCPWM0_GRP1_OUT0_[3]	
13	TCPWM0_GRP1_OUT0_[4]	
14	TCPWM0_GRP1_OUT0_[5]	
15	TCPWM0_GRP1_OUT0_[6]	
16	TCPWM0_GRP1_OUT0_[7]	
17	TCPWM0_GRP2_OUT0_[0]	
18	TCPWM0_GRP2_OUT0_[1]	
19	TCPWM0_GRP2_OUT0_[2]	
20	TCPWM0_GRP2_OUT0_[3]	
21	TCPWM0_GRP2_OUT0_[4]	
22	TCPWM0_GRP2_OUT0_[5]	
23	TCPWM0_GRP2_OUT0_[6]	
24	TCPWM0_GRP2_OUT0_[7]	
25:74	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)
75	SCB_I2C_SCL[0]	
76	SCB_TX[0]	SCB TX/RX DMA リクエスト
77	SCB_RX[0]	

(続く)

表 50 (続き) PERI トリガー ソース グループ 1 (PDMA1 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
78	SCB_I2C_SCL[1]	
79	SCB_TX[1]	
80	SCB_RX[1]	
81	SCB_I2C_SCL[2]	
82	SCB_TX[2]	
83	SCB_RX[2]	
84	SCB_I2C_SCL[3]	
85	SCB_TX[3]	
86	SCB_RX[3]	
87	SCB_I2C_SCL[4]	
88	SCB_TX[4]	
89	SCB_RX[4]	
90	SCB_I2C_SCL[5]	
91	SCB_TX[5]	
92	SCB_RX[5]	
93:94	CTI_TR_OUT[0:1]	CPUSS デバッグイベント
95	MXCORDIC_DONE	MXCORDIC がトリガーを完了
96	CANFD_TT_TR_OUT[0]	CAN TT 同期出力
97	CANFD_TT_TR_OUT[1]	CAN TT 同期出力
98	CAN_DBG0	
99	CAN_DBG1	
100	CAN_FIFO0_CH0	
101	CAN_FIFO0_CH1	
102	CAN_FIFO1_CH0	
103	CAN_FIFO1_CH1	
104	LPCOMP_DSI_COMP0	LPCOMP コンパレーター#0
105	LPCOMP_DSI_COMP1	LPCOMP コンパレーター#1
106	CPUSS_TR_FAULT[0]	
107	TRNG_BITSTREAM	TRNG_BITSTREAM
108:111	PASS_FIFO[0:3]	FIFO レベル用 PERI へのトリガー
112:119	PASS_PULSE[0:7]	HPPASS パルストリガー用 PERI へのトリガー
120:127	PASS_LEVEL[0:7]	レンジ レベルの PERI へのトリガー

表 51 PERI トリガー宛先グループ 1 (PDMA1 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:15	PDMA1_TR_IN[0:15]	P-DMA1 へのトリガー

表 52 PERI トリガー ソース グループ 2 (HSIOM0 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:16	PDMA0_TR_OUT[0:15]	P-DMA0 tr_out
17:32	PDMA1_TR_OUT[0:15]	P-DMA1 tr_out
33	TCPWM0_GRP0_OUT1_[0]	PDMA1 への 32 ビット TCPWM カウンター
34	TCPWM0_GRP0_OUT1_[1]	
35	TCPWM0_GRP0_OUT1_[2]	
36	TCPWM0_GRP0_OUT1_[3]	
37	TCPWM0_GRP1_OUT1_[0]	PDMA1 への 16 ビット TCPWM カウンター
38	TCPWM0_GRP1_OUT1_[1]	
39	TCPWM0_GRP1_OUT1_[2]	
40	TCPWM0_GRP1_OUT1_[3]	
41	TCPWM0_GRP1_OUT1_[4]	
42	TCPWM0_GRP1_OUT1_[5]	
43	TCPWM0_GRP1_OUT1_[6]	
44	TCPWM0_GRP1_OUT1_[7]	
45	TCPWM0_GRP2_OUT1_[0]	
46	TCPWM0_GRP2_OUT1_[1]	
47	TCPWM0_GRP2_OUT1_[2]	
48	TCPWM0_GRP2_OUT1_[3]	
49	TCPWM0_GRP2_OUT1_[4]	
50	TCPWM0_GRP2_OUT1_[5]	
51	TCPWM0_GRP2_OUT1_[6]	
52	TCPWM0_GRP2_OUT1_[7]	
53:60	TCPWM0_MOTIF_TR_OUT[0:7]	MOTIF トリガーグループ#1 MOTIF #0
61:64	TCPWM0_GRP0_LINE_[0:3]	TCPWM トリガーライン
65:68	TCPWM0_GRP0_LINE_COMPL_[0:3]	TCPWM トリガー line_compl
69:76	TCPWM0_GRP1_LINE_[256:263]	TCPWM トリガーライン
77:84	TCPWM0_GRP1_LINE_COMPL_[256:263]	TCPWM トリガー line_compl

(続く)

表 52 (続き) PERI トリガー ソース グループ 2 (HSIOM0 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
85:92	TCPWM0_GRP2_LINE_[512:519]	TCPWM トリガーライン
93:100	TCPWM0_GRP2_LINE_COMPL_[512:519]	TCPWM トリガー line_compl
101:150	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)
151	SCB_I2C_SCL[0]	
152	SCB_TX[0]	SCB TX/RX DMA リクエスト
153	SCB_RX[0]	
154	SCB_I2C_SCL[1]	
155	SCB_TX[1]	
156	SCB_RX[1]	
157	SCB_I2C_SCL[2]	
158	SCB_TX[2]	
159	SCB_RX[2]	
160	SCB_I2C_SCL[3]	
161	SCB_TX[3]	
162	SCB_RX[3]	
163	SCB_I2C_SCL[4]	
164	SCB_TX[4]	
165	SCB_RX[4]	
166	SCB_I2C_SCL[5]	
167	SCB_TX[5]	
168	SCB_RX[5]	
169:170	CTI_TR_OUT[0:1]	デバッグ イベント
171	MXCORDIC_DONE	MXCORDIC がトリガーを完了
172	CANFD_TT_TR_OUT[0]	CAN TT 同期出力
173	CANFD_TT_TR_OUT[1]	CAN TT 同期出力
174	LPCOMP_DSI_COMP0	LPCOMP コンパレーター#0
175	LPCOMP_DSI_COMP1	LPCOMP コンパレーター#1
176	CPUSS_TR_FAULT[0]	
177	TRNG_BITSTREAM	TRNG_BITSTREAM
178:181	PASS_FIFO[0:3]	FIFO レベル用 PERI へのトリガー
182:189	PASS_PULSE[0:7]	HPPASS パルス トリガー用 PERI へのトリガー

(続く)

表 52 (続き) PERI トリガー ソース グループ 2 (HSIOM0 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
190:197	PASS_LEVEL[0:7]	レンジ レベルの PERI へのトリガー

表 53 PERI トリガー宛先グループ 2 (HSIOM0 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:49	HSIOM_TR_IO_OUTPUT[0:49]	HSIOM へのトリガー

表 54 PERI トリガー ソース グループ 3 (HSIOM1 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:4	TCPWM0_GRP0_LINE[0:3]	
5:8	TCPWM0_GRP0_LINE_COMPL[0:3]	

表 55 PERI トリガー宛先グループ 3 (HSIOM1 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:7	HSIOM_TR_IO_OUTPUT[50:57]	TCPWM GRP0 から HSIOM へのトリガー

表 56 PERI トリガー ソース グループ 4 (HSIOM2 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:4	TCPWM0_GRP1_LINE[0:3]	
5:8	TCPWM0_GRP1_LINE_COMPL[0:3]	

表 57 PERI トリガー宛先グループ 4 (HSIOM2 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:7	HSIOM_TR_IO_OUTPUT[58:65]	TCPWM GRP0 から HSIOM へのトリガー

表 58 PERI トリガー ソース グループ 5 (HSIOM3 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:4	TCPWM0_GRP2_LINE[0:3]	
5:8	TCPWM0_GRP2_LINE_COMPL[0:3]	

表 59 PERI トリガー宛先グループ 5 (HSIOM3 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:7	HSIOM_TR_IO_OUTPUT[66:73]	TCPWM GRP0 から HSIOM へのトリガー

表 60 PERI トリガーソースグループ 6 (CPUSS_CTI Trigger MUX)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:16	PDMA0_TR_OUT[0:15]	P-DMA0 tr_out
17:32	PDMA1_TR_OUT[0:15]	P-DMA0 tr_out
33	TCPWM0_GRP0_OUT0_[0]	32 ビット TCPWM カウンター
34	TCPWM0_GRP0_OUT1_[0]	
35	TCPWM0_GRP0_OUT0_[1]	
36	TCPWM0_GRP0_OUT1_[1]	
37	TCPWM0_GRP0_OUT0_[2]	
38	TCPWM0_GRP0_OUT1_[2]	
39	TCPWM0_GRP0_OUT0_[3]	
40	TCPWM0_GRP0_OUT1_[3]	
41	TCPWM0_GRP1_OUT0_[0]	16 ビット TCPWM カウンター
42	TCPWM0_GRP1_OUT1_[0]	
43	TCPWM0_GRP1_OUT0_[1]	
44	TCPWM0_GRP1_OUT1_[1]	
45	TCPWM0_GRP1_OUT0_[2]	
46	TCPWM0_GRP1_OUT1_[2]	
47	TCPWM0_GRP1_OUT0_[3]	
48	TCPWM0_GRP1_OUT1_[3]	
49	TCPWM0_GRP1_OUT0_[4]	
50	TCPWM0_GRP1_OUT1_[4]	
51	TCPWM0_GRP1_OUT0_[5]	
52	TCPWM0_GRP1_OUT1_[5]	
53	TCPWM0_GRP1_OUT0_[6]	
54	TCPWM0_GRP1_OUT1_[6]	
55	TCPWM0_GRP1_OUT0_[7]	
56	TCPWM0_GRP1_OUT1_[7]	
57	TCPWM0_GRP2_OUT0_[0]	
58	TCPWM0_GRP2_OUT1_[0]	
59	TCPWM0_GRP2_OUT0_[1]	
60	TCPWM0_GRP2_OUT1_[1]	
61	TCPWM0_GRP2_OUT0_[2]	
62	TCPWM0_GRP2_OUT1_[2]	

(続く)

表 60 (続き) PERI トリガーソースグループ 6 (CPUSS_CTI Trigger MUX)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
63	TCPWM0_GRP2_OUT0_[3]	
64	TCPWM0_GRP2_OUT1_[3]	
65	TCPWM0_GRP2_OUT0_[4]	
66	TCPWM0_GRP2_OUT1_[4]	
67	TCPWM0_GRP2_OUT0_[5]	
68	TCPWM0_GRP2_OUT1_[5]	
69	TCPWM0_GRP2_OUT0_[6]	
70	TCPWM0_GRP2_OUT1_[6]	
71	TCPWM0_GRP2_OUT0_[7]	
72	TCPWM0_GRP2_OUT1_[7]	
73:80	TCPWM0_MOTIF_TR_OUT[0:7]	MOTIF トリガーグループ#1 MOTIF #0
81:130	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)
131	SCB_I2C_SCL[0]	
132	SCB_TX[0]	SCB TX/RX DMA リクエスト
133	SCB_RX[0]	
134	SCB_I2C_SCL[1]	
135	SCB_TX[1]	
136	SCB_RX[1]	
137	SCB_I2C_SCL[2]	
138	SCB_TX[2]	
139	SCB_RX[2]	
140	SCB_I2C_SCL[3]	
141	SCB_TX[3]	
142	SCB_RX[3]	
143	SCB_I2C_SCL[4]	
144	SCB_TX[4]	
145	SCB_RX[4]	
146	SCB_I2C_SCL[5]	
147	SCB_TX[5]	
148	SCB_RX[5]	
149:150	CTI_TR_OUT[0:1]	デバッグ イベント
151	MXCORDIC_DONE	MXCORDIC がトリガーを完了

(続く)

表 60 (続き) PERI トリガーソースグループ 6 (CPUSS_CTI Trigger MUX)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
152	CANFD_TT_TR_OUT[0]	CAN TT 同期出力
153	CANFD_TT_TR_OUT[1]	CAN TT 同期出力
154	LPCOMP_DSI_COMP0	LPCOMP コンパレーター#0
155	LPCOMP_DSI_COMP1	LPCOMP コンパレーター#1
156	CPUSS_TR_FAULT[0]	
157:160	PASS_FIFO[0:3]	FIFO レベル用 PERI へのトリガー
161:168	PASS_PULSE[0:7]	HPPASS パルストリガー用 PERI へのトリガー
169:176	PASS_LEVEL[0:7]	レンジ レベルの PERI へのトリガー

表 61 PERI トリガー宛先グループ 6 (CPUSS_CTI トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:1	CPUSS_CTI_TR_IN[0:1]	debug600.cti へのトリガー

表 62 PERI トリガーソースグループ 7 (CANTT トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1	CAN_TT_TR_OUT[0]	CAN TT 同期出力
2	CAN_TT_TR_OUT[1]	CAN TT 同期出力

表 63 PERI トリガー宛先グループ 7 (CANTT トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0	CAN_TT_TR_IN[0]	CAN TT 同期入力
1	CAN_TT_TR_IN[1]	CAN TT 同期入力

表 64 PERI トリガーソースグループ 8 (PERI_FREEZE トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1	CPUSS_TR_FAULT[0]	
2:3	CTI_TR_OUT[0:1]	デバッグ イベント

表 65 PERI トリガー宛先グループ 8 (PERI_FREEZE トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0	PERI_DEBUG_FREEZE_TR_IN	動作フリーズのトリガー
1	TCPWM_DEBUG_FREEZE_TR_IN	

表 66 PERI トリガー ソース グループ 9 (TCPWM0 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1	LPCOMP_DSI_COMP0	LPCOMP コンパレーター#0
2	LPCOMP_DSI_COMP1	LPCOMP コンパレーター#1
3:6	PASS_FIFO[0:3]	FIFO レベル用 PERI へのトリガー
7:14	PASS_PULSE[0:7]	HPPASS パルス トリガー用 PERI へのトリガー
15:22	PASS_LEVEL[0:7]	レンジ レベルの PERI へのトリガー

表 67 PERI トリガー宛先グループ 9 (TCPWM0 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:3	TCPWM0_TR_IN[0:3]	TCPWM0 へのトリガー

表 68 PERI トリガー ソース グループ 10 (TCPWM1 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:16	PDMA0_TR_OUT[0:15]	P-DMA0 tr_out
17:32	PDMA1_TR_OUT[0:15]	P-DMA1 tr_out
33	TCPWM0_GRP0_OUT1_[0]	PDMA1 への 32 ビット TCPWM カウンター
34	TCPWM0_GRP0_OUT1_[1]	
35	TCPWM0_GRP0_OUT1_[2]	
36	TCPWM0_GRP0_OUT1_[3]	
37	TCPWM0_GRP1_OUT1_[0]	PDMA1 への 16 ビット TCPWM カウンター
38	TCPWM0_GRP1_OUT1_[1]	
39	TCPWM0_GRP1_OUT1_[2]	
40	TCPWM0_GRP1_OUT1_[3]	
41	TCPWM0_GRP1_OUT1_[4]	
42	TCPWM0_GRP1_OUT1_[5]	
43	TCPWM0_GRP1_OUT1_[6]	
44	TCPWM0_GRP1_OUT1_[7]	
45	TCPWM0_GRP2_OUT1_[0]	
46	TCPWM0_GRP2_OUT1_[1]	
47	TCPWM0_GRP2_OUT1_[2]	
48	TCPWM0_GRP2_OUT1_[3]	

(続く)

表 68 (続き) PERI トリガー ソース グループ 10 (TCPWM1 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
49	TCPWM0_GRP2_OUT1_[4]	
50	TCPWM0_GRP2_OUT1_[5]	
51	TCPWM0_GRP2_OUT1_[6]	
52	TCPWM0_GRP2_OUT1_[7]	
53:60	TCPWM0_MOTIF_TR_OUT[0:7]	MOTIF トリガーグループ#1 MOTIF #0
61:110	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)
111	SCB_I2C_SCL[0]	
112	SCB_TX[0]	SCB TX/RX DMA リクエスト
113	SCB_RX[0]	
114	SCB_I2C_SCL[1]	
115	SCB_TX[1]	
116	SCB_RX[1]	
117	SCB_I2C_SCL[2]	
118	SCB_TX[2]	
119	SCB_RX[2]	
120	SCB_I2C_SCL[3]	
121	SCB_TX[3]	
122	SCB_RX[3]	
123	SCB_I2C_SCL[4]	
124	SCB_TX[4]	
125	SCB_RX[4]	
126	SCB_I2C_SCL[5]	
127	SCB_TX[5]	
128	SCB_RX[5]	
129	CANFD_TT_TR_OUT[0]	CAN TT 同期出力
130	CANFD_TT_TR_OUT[1]	CAN TT 同期出力
131	CPUSS_TR_FAULT[0]	

表 69 PERI トリガー宛先グループ 10 (TCPWM1 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:11	TCPWM0_TR_IN[4:15]	TCPWM0 へのトリガー

表 70　　PERI トリガー ソース グループ 11 (TCPWM2 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1:50	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)

表 71　　PERI トリガー宛先グループ 11 (TCPWM2 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0	TCPWM0_GRP0_CNT0_TR3_IN	グループ[0]、カウンター[0]、トリガー[3]
1	TCPWM0_GRP0_CNT1_TR3_IN	グループ[0]、カウンター[1]、トリガー[3]
2	TCPWM0_GRP0_CNT2_TR3_IN	グループ[0]、カウンター[2]、トリガー[3]
3	TCPWM0_GRP0_CNT3_TR3_IN	グループ[0]、カウンター[3]、トリガー[3]
4	TCPWM0_GRP1_CNT0_TR3_IN	グループ[1]、カウンター[0]、トリガー[3]
5	TCPWM0_GRP1_CNT1_TR3_IN	グループ[1]、カウンター[1]、トリガー[3]
6	TCPWM0_GRP1_CNT2_TR3_IN	グループ[1]、カウンター[2]、トリガー[3]
7	TCPWM0_GRP1_CNT3_TR3_IN	グループ[1]、カウンター[3]、トリガー[3]
8	TCPWM0_GRP1_CNT4_TR3_IN	グループ[1]、カウンター[4]、トリガー[3]
9	TCPWM0_GRP1_CNT5_TR3_IN	グループ[1]、カウンター[5]、トリガー[3]
10	TCPWM0_GRP1_CNT6_TR3_IN	グループ[1]、カウンター[6]、トリガー[3]
11	TCPWM0_GRP1_CNT7_TR3_IN	グループ[1]、カウンター[7]、トリガー[3]
12	TCPWM0_GRP2_CNT0_TR3_IN	グループ[2]、カウンター[0]、トリガー[3]
13	TCPWM0_GRP2_CNT1_TR3_IN	グループ[2]、カウンター[1]、トリガー[3]
14	TCPWM0_GRP2_CNT2_TR3_IN	グループ[2]、カウンター[2]、トリガー[3]
15	TCPWM0_GRP2_CNT3_TR3_IN	グループ[2]、カウンター[3]、トリガー[3]

(続く)

表 71 (続き) PERI トリガー宛先グループ 11 (TCPWM2 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
16	TCPWM0_GRP2_CNT4_TR3_IN	グループ[2]、カウンター[4]、トリガー[3]
17	TCPWM0_GRP2_CNT5_TR3_IN	グループ[2]、カウンター[5]、トリガー[3]
18	TCPWM0_GRP2_CNT6_TR3_IN	グループ[2]、カウンター[6]、トリガー[3]
19	TCPWM0_GRP2_CNT7_TR3_IN	グループ[2]、カウンター[7]、トリガー[3]

表 72 PERI トリガー ソース グループ 12 (TCPWM3 トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1	TCPWM0_GRP0_OUT0_[0]	32 ビット TCPWM カウンター
2	TCPWM0_GRP0_OUT0_[1]	
3	TCPWM0_GRP0_OUT0_[2]	
4	TCPWM0_GRP0_OUT0_[3]	
5	TCPWM0_GRP1_OUT0_[0]	16 ビット TCPWM カウンター
6	TCPWM0_GRP1_OUT0_[1]	
7	TCPWM0_GRP1_OUT0_[2]	
8	TCPWM0_GRP1_OUT0_[3]	
9	TCPWM0_GRP1_OUT0_[4]	
10	TCPWM0_GRP1_OUT0_[5]	
11	TCPWM0_GRP1_OUT0_[6]	
12	TCPWM0_GRP1_OUT0_[7]	
13	TCPWM0_GRP2_OUT0_[0]	
14	TCPWM0_GRP2_OUT0_[1]	
15	TCPWM0_GRP2_OUT0_[2]	
16	TCPWM0_GRP2_OUT0_[3]	
17	TCPWM0_GRP2_OUT0_[4]	
18	TCPWM0_GRP2_OUT0_[5]	
19	TCPWM0_GRP2_OUT0_[6]	
20	TCPWM0_GRP2_OUT0_[7]	

表 73 PERI トリガー宛先グループ 12 (TCPWM3 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0	TCPWM0_GRP0_CNT0_TR4_IN	グループ[0]、カウンター[0]、トリガー[4]
1	TCPWM0_GRP0_CNT1_TR4_IN	グループ[0]、カウンター[1]、トリガー[4]
2	TCPWM0_GRP0_CNT2_TR4_IN	グループ[0]、カウンター[2]、トリガー[4]
3	TCPWM0_GRP0_CNT3_TR4_IN	グループ[0]、カウンター[3]、トリガー[4]
4	TCPWM0_GRP1_CNT0_TR4_IN	グループ[1]、カウンター[0]、トリガー[4]
5	TCPWM0_GRP1_CNT1_TR4_IN	グループ[1]、カウンター[1]、トリガー[4]
6	TCPWM0_GRP1_CNT2_TR4_IN	グループ[1]、カウンター[2]、トリガー[4]
7	TCPWM0_GRP1_CNT3_TR4_IN	グループ[1]、カウンター[3]、トリガー[4]
8	TCPWM0_GRP1_CNT4_TR4_IN	グループ[1]、カウンター[4]、トリガー[4]
9	TCPWM0_GRP1_CNT5_TR4_IN	グループ[1]、カウンター[5]、トリガー[4]
10	TCPWM0_GRP1_CNT6_TR4_IN	グループ[1]、カウンター[6]、トリガー[4]
11	TCPWM0_GRP1_CNT7_TR4_IN	グループ[1]、カウンター[7]、トリガー[4]
12	TCPWM0_GRP2_CNT0_TR4_IN	グループ[2]、カウンター[0]、トリガー[4]
13	TCPWM0_GRP2_CNT1_TR4_IN	グループ[2]、カウンター[1]、トリガー[4]
14	TCPWM0_GRP2_CNT2_TR4_IN	グループ[2]、カウンター[2]、トリガー[4]
15	TCPWM0_GRP2_CNT3_TR4_IN	グループ[2]、カウンター[3]、トリガー[4]
16	TCPWM0_GRP2_CNT4_TR4_IN	グループ[2]、カウンター[4]、トリガー[4]
17	TCPWM0_GRP2_CNT5_TR4_IN	グループ[2]、カウンター[5]、トリガー[4]
18	TCPWM0_GRP2_CNT6_TR4_IN	グループ[2]、カウンター[6]、トリガー[4]

(続く)

表 73 (続き) PERI トリガー宛先グループ 12 (TCPWM3 トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
19	TCPWM0_GRP2_CNT7_TR4_IN	グループ[2]、カウンター[7]、トリガー[4]

表 74 PERI トリガーソースグループ 13 (MOTIF トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1	TCPWM0_GRP0_OUT0_[0]	
2	TCPWM0_GRP0_OUT1_[0]	
3	TCPWM0_GRP0_OUT0_[1]	
4	TCPWM0_GRP0_OUT1_[1]	
5	TCPWM0_GRP0_OUT0_[2]	
6	TCPWM0_GRP0_OUT1_[2]	
7	TCPWM0_GRP0_OUT0_[3]	
8	TCPWM0_GRP0_OUT1_[3]	
9	TCPWM0_GRP1_OUT0_[0]	16 ビット TCPWM グループ 1 カウンター
10	TCPWM0_GRP1_OUT1_[0]	
11	TCPWM0_GRP1_OUT0_[1]	
12	TCPWM0_GRP1_OUT1_[1]	
13	TCPWM0_GRP1_OUT0_[2]	
14	TCPWM0_GRP1_OUT1_[2]	
15	TCPWM0_GRP1_OUT0_[3]	
16	TCPWM0_GRP1_OUT1_[3]	
17	TCPWM0_GRP1_OUT0_[4]	
18	TCPWM0_GRP1_OUT1_[4]	
19	TCPWM0_GRP1_OUT0_[5]	
20	TCPWM0_GRP1_OUT1_[5]	
21	TCPWM0_GRP1_OUT0_[6]	
22	TCPWM0_GRP1_OUT1_[6]	
23	TCPWM0_GRP1_OUT0_[7]	
24	TCPWM0_GRP1_OUT1_[7]	
25	TCPWM0_GRP2_OUT0_[0]	
26	TCPWM0_GRP2_OUT1_[0]	
27	TCPWM0_GRP2_OUT0_[1]	

(続く)

表 74 (続き) PERI トリガーソースグループ 13 (MOTIF トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
28	TCPWM0_GRP2_OUT1_[1]	
29	TCPWM0_GRP2_OUT0_[2]	
30	TCPWM0_GRP2_OUT1_[2]	
31	TCPWM0_GRP2_OUT0_[3]	
32	TCPWM0_GRP2_OUT1_[3]	
33	TCPWM0_GRP2_OUT0_[4]	
34	TCPWM0_GRP2_OUT1_[4]	
35	TCPWM0_GRP2_OUT0_[5]	
36	TCPWM0_GRP2_OUT1_[5]	
37	TCPWM0_GRP2_OUT0_[6]	
38	TCPWM0_GRP2_OUT1_[6]	
39	TCPWM0_GRP2_OUT0_[7]	
40	TCPWM0_GRP2_OUT1_[7]	
41:90	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)
91:98	HPPASS_TR_LEVEL[0:7]	HPPASS CSG トリガー出力

表 75 PERI トリガー宛先グループ 13 (MOTIF トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:7	TCPWM0_MOTIF_TR_IN[0:7]	MOTIF トリガーグループ#1 MOTIF #0

表 76 PERI トリガーソースグループ 14 (HPPASS トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
0	ZERO	
1	TCPWM0_GRP0_OUT1_[0]	32 ビット TCPWM カウンター
2	TCPWM0_GRP0_OUT1_[1]	
3	TCPWM0_GRP0_OUT1_[2]	
4	TCPWM0_GRP0_OUT1_[3]	
5	TCPWM0_GRP1_OUT1_[0]	16 ビット TCPWM カウンター
6	TCPWM0_GRP1_OUT1_[1]	
7	TCPWM0_GRP1_OUT1_[2]	
8	TCPWM0_GRP1_OUT1_[3]	
9	TCPWM0_GRP1_OUT1_[4]	
10	TCPWM0_GRP1_OUT1_[5]	

(続く)

表 76 (続き) PERI トリガーソースグループ 14 (HPPASS トリガーマルチプレクサ)

トリガーマルチプレクサ入力	トリガー ソース ラベル	説明
11	TCPWM0_GRP1_OUT1_[6]	
12	TCPWM0_GRP1_OUT1_[7]	
13	TCPWM0_GRP2_OUT1_[0]	
14	TCPWM0_GRP2_OUT1_[1]	
15	TCPWM0_GRP2_OUT1_[2]	
16	TCPWM0_GRP2_OUT1_[3]	
17	TCPWM0_GRP2_OUT1_[4]	
18	TCPWM0_GRP2_OUT1_[5]	
19	TCPWM0_GRP2_OUT1_[6]	
20	TCPWM0_GRP2_OUT1_[7]	
21:28	TCPWM0_MOTIF_TR_OUT[0:7]	MOTIF トリガーグループ#1 MOTIF #0
29:78	HSIOM_TR_OUT[0:49]	HSIOM のピンに接続されたトリガー (peri.tr_in)

表 77 PERI トリガー宛先グループ 14 (HPPASS トリガーマルチプレクサ)

トリガーマルチプレクサ出力	トリガー宛先ラベル	説明
0:7	PASS_TR_A_IN[0:7]	PASS トリガー マルチプレクサ入力

PSOC™ Control C3 MCU のトリガー 1 対 1 接続は、以下の表で確認できます。

表 78 PERI トリガー 1 対 1 グループ 0 (CAN DW0 トリガー - DW から CAN へ戻る)

入力	ラベル	説明
0	PDMA0_TR_OUT[0]_ACK_TO_CAN_0	CAN DW アクノリッジ
1	PDMA0_TR_OUT[0]_ACK_TO_CAN_1	CAN DW アクノリッジ

表 79 PERI トリガー 1 対 1 グループ 1 (HPPASS CSG トリガー)

トリガーマルチプレクサ入力	トリガーマルチプレクサラベル	説明
0	HPPASS_CSG_TO_TCPWM0_GRP0_CNT0_TR2_IN	HPPASS→TCPWM グループ[0]、カウンター[0]、トリガー[2]
1	HPPASS_CSG_TO_TCPWM0_GRP1_CNT0_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[0]、トリガー[2]
2	HPPASS_CSG_TO_TCPWM0_GRP1_CNT1_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[1]、トリガー[2]

(続く)

表 79 (続き) PERI トリガー 1 対 1 グループ 1 (HPPASS CSG トリガー)

トリガーマルチプレクサ入力	トリガーマルチプレクサラベル	説明
3	HPPASS_CSG_TO_TCPWM0_GRP1_CNT2_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[2]、トリガー[2]
4	HPPASS_CSG_TO_TCPWM0_GRP0_CNT1_TR2_IN	HPPASS→TCPWM グループ[0]、カウンター[1]、トリガー[2]
5	HPPASS_CSG_TO_TCPWM0_GRP1_CNT3_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[3]、トリガー[2]
6	HPPASS_CSG_TO_TCPWM0_GRP1_CNT4_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[4]、トリガー[2]
7	HPPASS_CSG_TO_TCPWM0_GRP1_CNT5_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[5]、トリガー[2]
8	HPPASS_CSG_TO_TCPWM0_GRP0_CNT2_TR2_IN	HPPASS→TCPWM グループ[0]、カウンター[2]、トリガー[2]
9	HPPASS_CSG_TO_TCPWM0_GRP1_CNT6_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[6]、トリガー[2]
10	HPPASS_CSG_TO_TCPWM0_GRP1_CNT7_TR2_IN	HPPASS→TCPWM グループ[1]、カウンター[7]、トリガー[2]
11	HPPASS_CSG_TO_TCPWM0_GRP2_CNT0_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[0]、トリガー[2]
12	HPPASS_CSG_TO_TCPWM0_GRP0_CNT3_TR2_IN	HPPASS→TCPWM グループ[0]、カウンター[3]、トリガー[2]
13	HPPASS_CSG_TO_TCPWM0_GRP2_CNT1_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[1]、トリガー[2]
14	HPPASS_CSG_TO_TCPWM0_GRP2_CNT2_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[2]、トリガー[2]
15	HPPASS_CSG_TO_TCPWM0_GRP2_CNT3_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[3]、トリガー[2]

(続く)

表 79 (続き) PERI トリガー 1 対 1 グループ 1 (HPPASS CSG トリガー)

トリガーマルチプレクサ入力	トリガーマルチプレクサラベル	説明
16	HPPASS_CSG_TO_TCPWM0_GRP2_CNT4_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[4]、トリガー[2]
17	HPPASS_CSG_TO_TCPWM0_GRP2_CNT5_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[5]、トリガー[2]
18	HPPASS_CSG_TO_TCPWM0_GRP2_CNT6_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[6]、トリガー[2]
19	HPPASS_CSG_TO_TCPWM0_GRP2_CNT7_TR2_IN	HPPASS→TCPWM グループ[2]、カウンター[7]、トリガー[2]

表 80 PERI トリガー 1 対 1 グループ 2 (HPPASS トリガー)

トリガーマルチプレクサ入力	トリガーマルチプレクサラベル	説明
0	TCPWM0_GRP0_CNT2_OUT1_TO_HPPASS_TR_IN[0]	TCPWM グループ[0]、カウンター[2]、OUT1～HPPASS
1	TCPWM0_GRP0_CNT3_OUT1_TO_HPPASS_TR_IN[1]	TCPWM グループ[0]、カウンター[3]、OUT1～HPPASS
2	TCPWM0_GRP1_CNT5_OUT1_TO_HPPASS_TR_IN[2]	TCPWM グループ[1]、カウンター[5]、OUT1～HPPASS
3	TCPWM0_GRP1_CNT6_OUT1_TO_HPPASS_TR_IN[3]	TCPWM グループ[1]、カウンター[6]、OUT1～HPPASS
4	TCPWM0_GRP1_CNT7_OUT1_TO_HPPASS_TR_IN[4]	TCPWM グループ[1]、カウンター[7]、OUT1～HPPASS
5	TCPWM0_GRP2_CNT5_OUT1_TO_HPPASS_TR_IN[5]	TCPWM グループ[2]、カウンター[5]、OUT1～HPPASS
6	TCPWM0_GRP2_CNT6_OUT1_TO_HPPASS_TR_IN[6]	TCPWM グループ[2]、カウンター[6]、OUT1～HPPASS
7	TCPWM0_GRP2_CNT7_OUT1_TO_HPPASS_TR_IN[7]	TCPWM グループ[2]、カウンター[7]、OUT1～HPPASS

10.4 ソフトウェア トリガー

トリガー マルチプレクサへのすべての入出力信号は、ソフトウェアによってトリガーできます。これは、PERI_TR_CMD レジスタに書き込むことでできます。このレジスタにより、対応する信号を周辺クロック サイクル数分トリガーできます。

10.5 トリガー機能

図 28 に、マルチプレクサのトリガーグループの概要を示します。

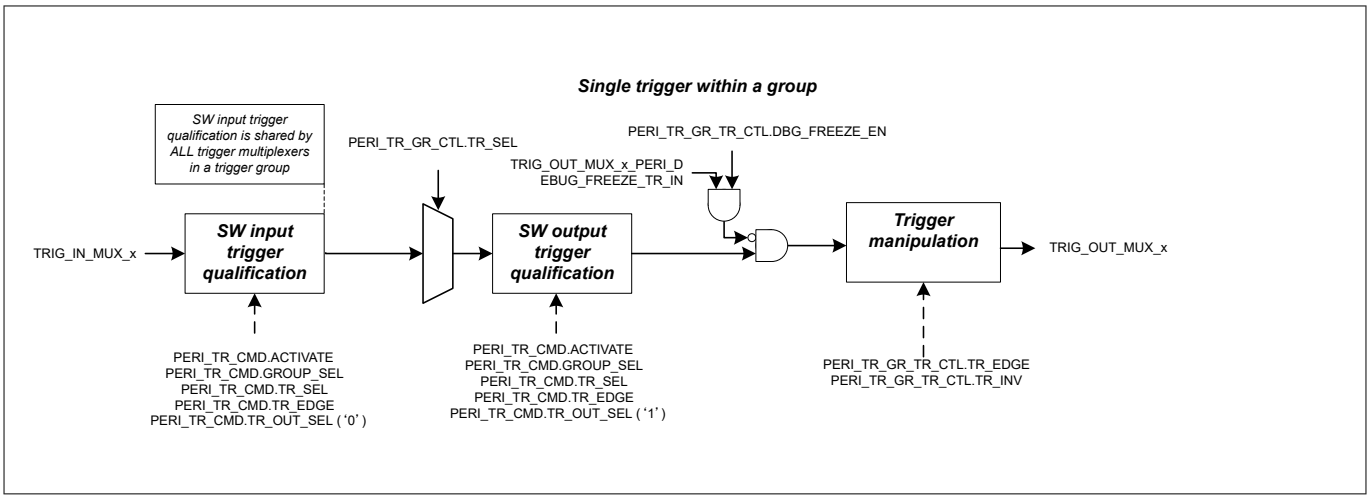


図 28 マルチプレクサ トリガーグループ

グループ トリガー 設定:

PERI[Peri Group Number]_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_SEL = 接続する入力トリガー

PERI[Peri Group Number]_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_INV = トリガーを反転させるか否か

PERI[Peri Group Number]_TR_GR[Group Number].PERI_TR_GR_TR_CTL[Trigger Number].TR_EDGE = エッジまたはレベルセンシティブトリガー

ソフトウェアによるグループ トリガーの起動

PERI[Peri Group Number]_TR_CMD.TR_SEL = Trigger Number 入力

PERI[Peri Group Number]_TR_CMD.GROUP_SEL = Trigger Number Group_Nr

PERI[Peri Group Number]_TR_CMD.TR_EDGE = エッジまたはレベルセンシティブトリガー

PERI[Peri Group Number]_TR_CMD.OUT_SEL = 出力トリガーは 1、入力トリガーは 0

PERI[Peri Group Number]_TR_CMD.ACTIVATE = 1 は、設定されたトリガーの起動を意味します。

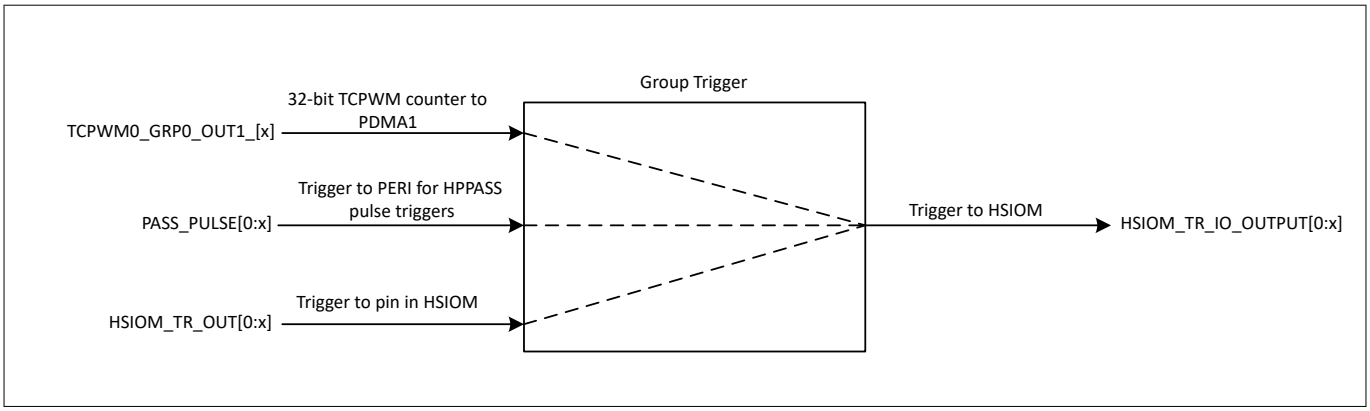


図 29　　グループトリガーの例

図 30 に、1 対 1 グループの概要を示します。

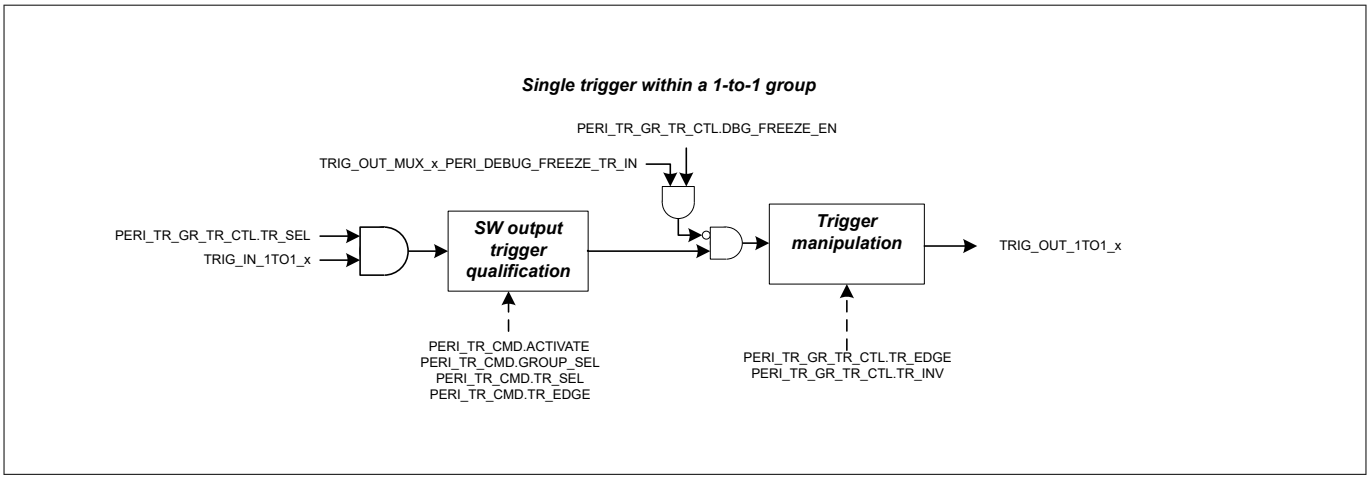


図 30　　1 対 1 トリガーグループ

1 対 1 トリガー設定:

PERI[Peri Group Number]_TR_1TO1_GR[Group Number]. PERITR_1TO1_GR_TR_CTL[Trigger Number].TR_SEL = True (入力トリガー) または False (一定信号レベル'0')。

PERI[Peri Group Number]_TR_1TO1_GR[Group Number]. PERI_TR_1TO1_GR_TR_CTL[Trigger Number].TR_INV = 出力トリガーを反転させるかどうか。

PERI[Peri Group Number]_TR_1TO1_GR[Group Number]. PERI_TR_1TO1_GR_TR_CTL[Trigger Number].TR_EDGE = エッジまたはレベル感応トリガー

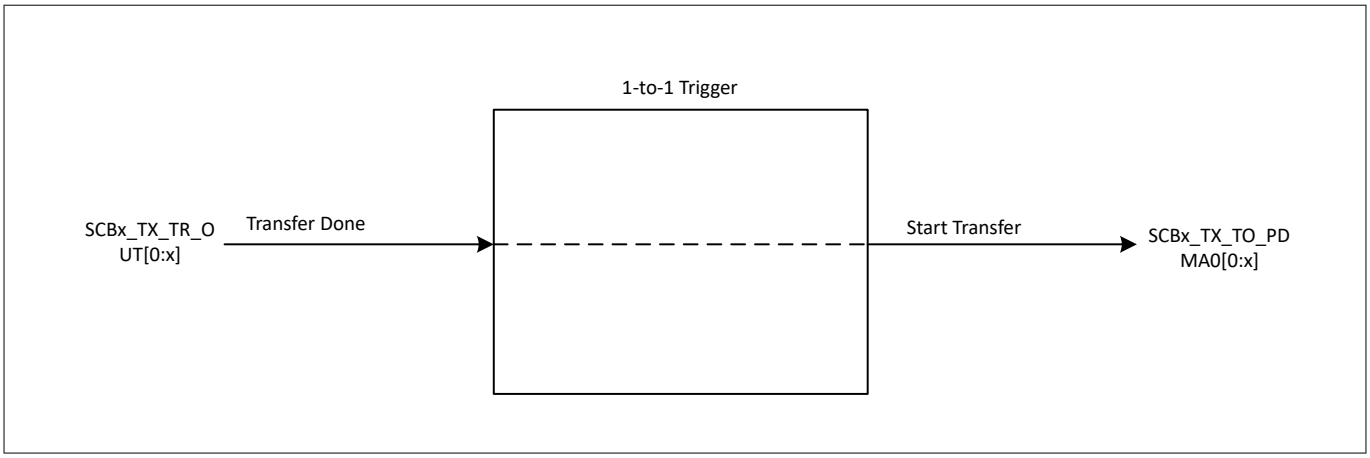


図 31　　1 対 1 トリガーの例

前述のように、トリガー操作ロジックは非同期エッジ検出ロジックを提供します。トリガー操作の詳細については、図 32 を参照してください。

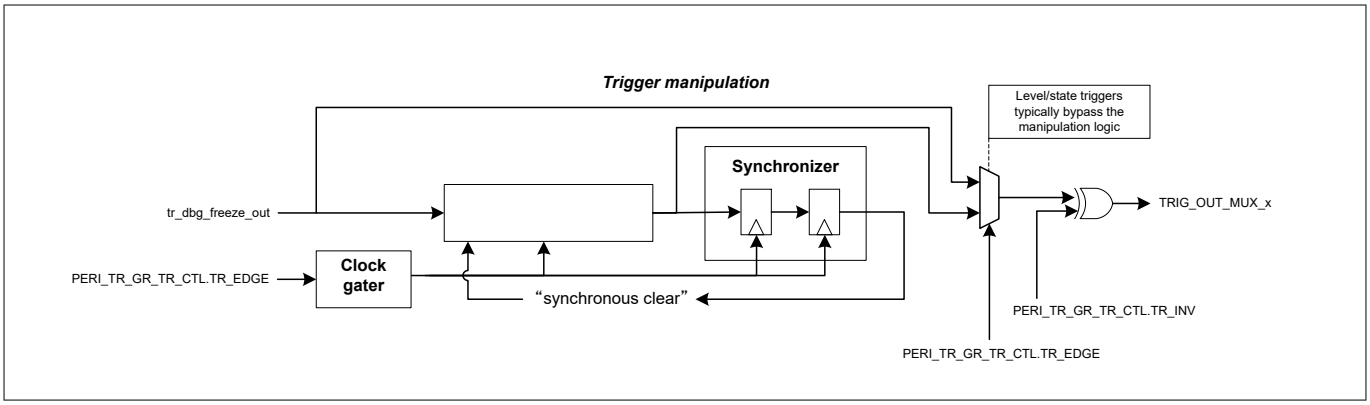


図 32　　トリガー操作

さらに、トリガーを生成するために使用できる I/O (TRIG_IN[0:x]) があります。これらの入力は TCPWM、SAR ADC、PERI、CPU-CTI のトリガーに使用できます。デバイス固有のトリガーマルチプレクサの割り当てについては、デバイスのデータシートを参照してください。

注:　両方のトリガーが同時に有効になった場合、衝突につながる可能性のあるトリガーの組み合わせが複数存在する可能性があります。

10.6　レジスタの概要

表 81　　トリガー マルチプレクサ レジスタ

記号	名称	説明
PERI[i]_TR_CMD	トリガー コマンド レジスタ	制御により、トリガー マルチプレクサ構造の特定の入力トリガーまたは出力トリガーをソフトウェアで起動できます。

(続く)

表 81 (続き) トリガー マルチプレクサ レジスタ

PERI[i]_TR_GR[x]_TR_CTL[y]	トリガー グループ コントロール レジスタ	グループトリガーアクションを制御し、ペリフェラル入力トリガーを複数のペリフェラル出力トリガーに接続します。 また、信号を反転させたり、出力信号をレベルセンシティブトリガーとして扱うか、エッジセンシティブトリガーとして扱うかを指定することもできます。 出力トリガーがデバッグモードでブロックされるかどうかを指定します。Tr_dbg_freeze (bit 12) は、High に設定されると、出力トリガーの発生をブロックします。 レジスタフォーマットにおいて、「i」はペリグループ、「x」はトリガーグループ、「y」はマルチプレクサからの出力トリガーライン番号です。
PERI[i]_TR_1TO1_GR[x]_TR_CTL[y]	トリガー 1 対 1 グループ コントロール レジスタ	1 対 1 のトリガー動作を制御し、周辺入力トリガーを特定の周辺出力トリガーに接続します。 また、信号を反転させるトリガー操作や、出力信号をレベルセンシティブトリガーとして扱うか、エッジセンシティブトリガーとして扱うかを指定することもできます。 出力トリガーがデバッグモードでブロックされるかどうかを指定します。Tr_dbg_freeze (bit 12) は、High に設定されると、出力トリガーの発生をブロックします。 レジスタフォーマットにおいて、「i」はペリグループ、「x」はトリガーグループ、「y」はマルチプレクサからの出力トリガーライン番号です。

11 座標回転デジタルコンピュータ (CORDIC)

11.1 特長

CORDIC は、三角関数、双曲関数、複素関数、および関連関数のハードウェア計算で CPU をサポートする周辺モジュールです。CPU は、ペリフェラル AHB バスを介して CORDIC にアクセスできます。動作の完了は、ステータスビット、割込み要求ライン、またはトリガー完了ラインを介して通知されます。

CORDIC アルゴリズムは有用な収束方法です。ユークリッド平面のみでなく、双曲面でもベクトル回転を行えます。CORDIC アルゴリズムは反復プロセスであり、切り捨て誤差が内在します。CORDIC コプロセッサでは、1 回の計算に 28 回の反復を使用し、カーネルのデータ幅を 32 ビット以上とすることで、より高い精度が得られます。このアルゴリズムを使用する主な利点は、ソフトウェアに比べて計算速度が速く、精度が高いことです。一般化された CORDIC アルゴリズムには、以下の CORDIC 方程式があります。

表 82 CORDIC アルゴリズム方程式

方程式	ラベル
$x_{n+1} = x_n - m \cdot d_n y_n 2^{-n}$	Eq.1
$y_{n+1} = y_n + d_n x_n 2^{-n}$	Eq.2
$z_{n+1} = z_n - d_n e_n$	Eq.3

m ファクターはベクトルの回転を制御し、円関数と双曲線関数の角度のセットを選択します。n ファクターは、CORDIC 計算の繰り返し回数です。

m = 1: $e_n = \text{atan}(2^{-n})$ の円形モード

m = -1: $e_n = \text{atanh}(2^{-n})$ の双曲線関数モード

CORDIC 機能一覧:

- 動作モード
  - 円関数 (三角関数) と双曲線関数を解くためのすべての CORDIC 動作モードに対応
  - 全動作モード用のルックアップテーブル (LUT) を内蔵
- サーキュラー ベクタリング モード: 角度と大きさを解くために、X と Y の初期データの値を$[-2^{23}, (2^{23}-1)]$ の全範囲までサポートするように拡張
- 円形回転モード: 三角法を解くための$[-\pi, ((2^{23}-1)/2^{23})\pi]$ 範囲の角度を表す、$[-2^{23}, (2^{23}-1)]$の全範囲までの初期 Z データの値のサポートを拡張
- 24 ビットのアクセス可能なデータ幅
  - 32 ビットのカーネル データ幅に、X と Y にそれぞれ 2 つの符号拡張ビットと 1 つのオーバーフロービットを加えたもの
  - 29 ビットのカーネル データ幅に 1 符号拡張ビットと Z 用の 1 オーバーフロー ビットを加えたもの
  - 新しい計算のために MXCORDIC_1_0_KEEP レジスタの最後の値を保持する KEEP ビット
- 1 回の計算で 28 回の反復 (双曲線関数モードでは 27 回): 1 クロック サイクルあたり 4 回の反復。その結果、1 回の計算あたり 7 クロック サイクルと、データ バイトの書き込みと読み出しアクセスにかかる時間を除いた、スタート (ST) ビット セットから計算終了フラグ セットまでの 3 クロック サイクルが必要となります。
- 2 の補数データ処理
  - X と Y のデータは、一般的に整数または有理数として受け入れられます。
  - 唯一の例外: X 結果データ、符号なし結果をユーザーが選択可能

- 切り捨て誤差
  - CORDIC の計算結果は、LSB の切り捨てによって近似値を返す場合があります。
  - CORDIC 計算結果データの制度は高く、特にサーキュラー (三角関数) モード時に高くなります。
- 割込み
  - 計算完了時
  - 割込みイネーブルと対応フラグ

11.2 アーキテクチャ

11.2.1 ブロック図

CORDIC ブロック図は、システム インターフェース、MMIO、および CORDIC カーネルで構成されます。

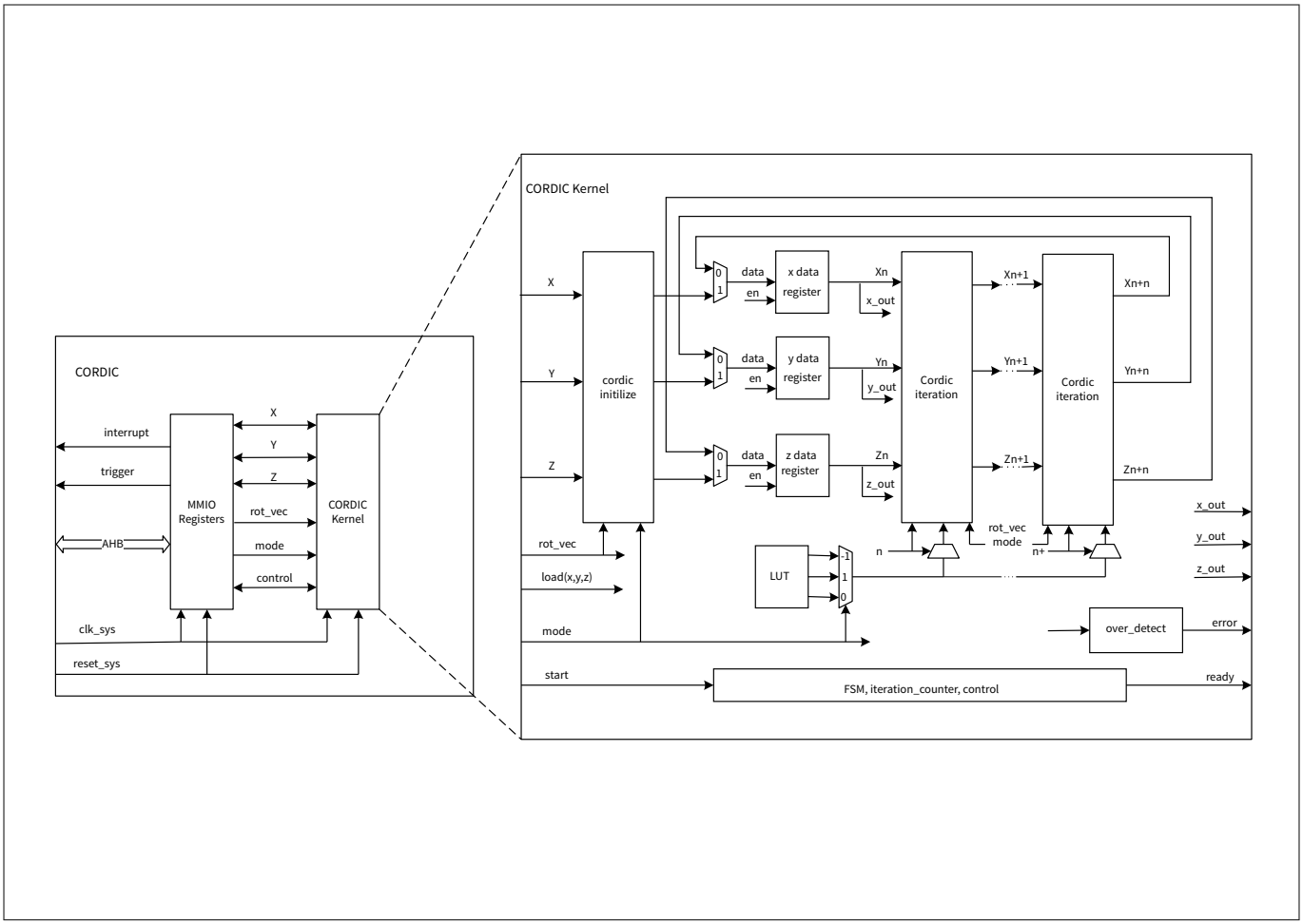


図 33 CORDIC ブロック図

システムインターフェース: CORDIC には、CPU と通信するための割込みインターフェースとトリガーインターフェースがあります。通常、ジョブの完了や実行中のエラーを知らせるために、CPU に割込み要求を送ります。DW にトリガーリクエストを送り、ジョブの完了を知らせます。DW は、CORDIC の設定および起動、結果の読み出しに使用できます。

MMIO: MMIO ブロックには、すべてのコントロール、ステータス、データ レジスタが含まれます。MMIO は AHB スレーブ インターフェースによってアクセスされます。

CORDIC カーネル: 反復 CORDIC アルゴリズムを実装しています。X、Y、Z データ レジスタは、MMIO インターフェースを介して初期値が設定されます。計算が開始されると、これらのレジスタには CORDIC_ITERATION ブロックで算出された中間結果が格納されます。CORDIC_ITERATION ブロックは、組み合わせシフターおよび加算器を使用して CORDIC アルゴリズムの 1 つの反復を実行します。4 つの CORDIC_ITERATION が順番にインスタンス化されます。7 クロック サイクル後、最終結果 (28 反復) が X、Y、Z データ レジスタで利用可能になります。シーケンスは有限状態機械 (FSM) と反復カウンターによって制御されます。ルックアップテーブル (LUT) は、円曲線および双曲線関数モードでの CORDIC アルゴリズムに必要な定数を証明します。

11.3 機能説明

ここでは、このブロックの機能について説明します。

11.3.1 動作

CORDIC コプロセッサは、回転モードまたはベクトルモードのいずれかで円関数 (三角関数) および双曲線関数に使用できます。モードは、MXCORDIC_1_0_CON コントロール レジスタを介してソフトウェアで選択可能です。カーネル データ レジスタの初期化は、レジスタ MXCORDIC_1_0_KEEP の各 KEEPx ビットをクリアすることで有効になります。MXCORDIC_1_0_CON.ST_MODE = 1 の場合、MXCORDIC_1_0_START_CMD.ST に'1'を書き込むとで新しい計算を開始します。そうでなければ、デフォルトでは、MXCORDIC_1_0_CON.ST_MODE = 0 の場合、レジスタ MXCORDIC_1_0_CORDZ への書き込みアクセス後に新しい計算が始まります。各計算には、MXCORDIC_1_0_CON.N_ITER*4 回の CORDIC 反復が必要です。ビット MXCORDIC_1_0_STAT.BSY は、ビジー ステータスを示すために計算中にセットされます。計算が終わると、ハードウェアはそれをクリアします。ビット MXCORDIC_1_0_START_CMD.ST は、新規 CORDIC 計算が開始された直後にクリアされます。

KEEP が 0 の場合はデータレジスタから、KEEP が 1 の場合は結果レジスタから初期データがロードされます)。CORDIC 計算を開始する第 1 段階として、初期データがデータレジスタ CORDx から内部カーネルデータレジスタへロードされます。計算中、カーネル データ レジスタは常に最新の中間データを保持します。計算が完了すると、結果データは結果レジスタ CORRx にロードされます。

データレジスタ CORDx はシャドーレジスタとして機能し、進行中の計算に影響を与えることなく書き込めます。値は、MXCORDIC_1_0_START_CMD.ST ビットが有効に設定されている場合、または X データ レジスタ CORDX への書き込みアクセス後に MXCORDIC_1_0_CON.ST_MODE = 0 の場合にのみカーネルデータ レジスタへ転送されます (対応データの KEEP ビットが設定されていない場合)。結果データは、計算終了時に結果レジスタ CORRx から読み出せます (BSY はもはやアクティブではありません)。結果データは、次の計算を開始する前に読み出さなければいけません。MXCORDIC_1_0_STAT.BSY がセットされている間 (そしてカーネルがまだ計算を実行している間)、結果レジスタ CORRx への読み出しアクセスは、MXCORDIC_1_0_STAT.BSY がリセットされるまでカーネルにバス待機を発行させます。これにより、確実に結果レジスタ CORRx への読み出しアクセスが有効な結果を返します。

各計算が終了すると、MXCORDIC_1_0_STAT.BSY は 0 に戻ります。CORDIC の計算終了を示す MXCORDIC_1_0_INTR.CDEOC ビットが設定されます。MXCORDIC_1_0_INTR_MASK.CDEOC = 1 で割込みが有効になると、割込み要求信号が有効になります。MXCORDIC_1_0_INTR.CDEOC フラグは、MXCORDIC_1_0_INTR.CDEOC = 1 を書き込むことにより、ソフトウェアから手動でクリアする必要があります。X,Y,Z の結果データは内部的にチェックされ、データがオーバーフローした場合には MXCORDIC_1_0_INTR.CDERR ビットが設定されます。MXCORDIC_1_0_INTR_MASK.CDERR = 1 を書き込み、割込みが有効になると、割込み要求信号がアクティブになります。MXCORDIC_1_0_INTR.CDERR フラグは、MXCORDIC_1_0_INTR.CDERR = 1 を書き込むことにより、ソフトウェアから手動でクリアする必要があります。

各計算の終了時、ハードウェアトリガーは 2 クロックサイクルの間、‘1’に設定されます。この信号を使用して DW をトリガーし、CORDIC の結果を読み出せます。

計算中 (MXCORDIC_1_0_CON.ST_MODE = 1 の場合) に MXCORDIC_1_0_STAT.BSY がセットされている状態で MXCORDIC_1_0_START_CMD.ST ビットをセットしても効果はありません。新しい計算を開始するためには、MXCORDIC_1_0_START_CMD.ST ビットを、MXCORDIC_1_0_STAT.BSY がアクティブでなくなった後に、再度セットす

る必要があります。同様に、実行中の計算 (MXCORDIC_1_0_STAT.BSY で示される) 中に動作モードを変更しても影響はありません。

自動開始モード (MXCORDIC_1_0_CON.ST_MODE = 0) では、MXCORDIC_1_0_STAT.BSY が設定されている間に CORDX への書き込みアクセスが発生した場合、MXCORDIC_1_0_CON.ST_MODE = 1 の場合のように、新たな計算は開始されません。

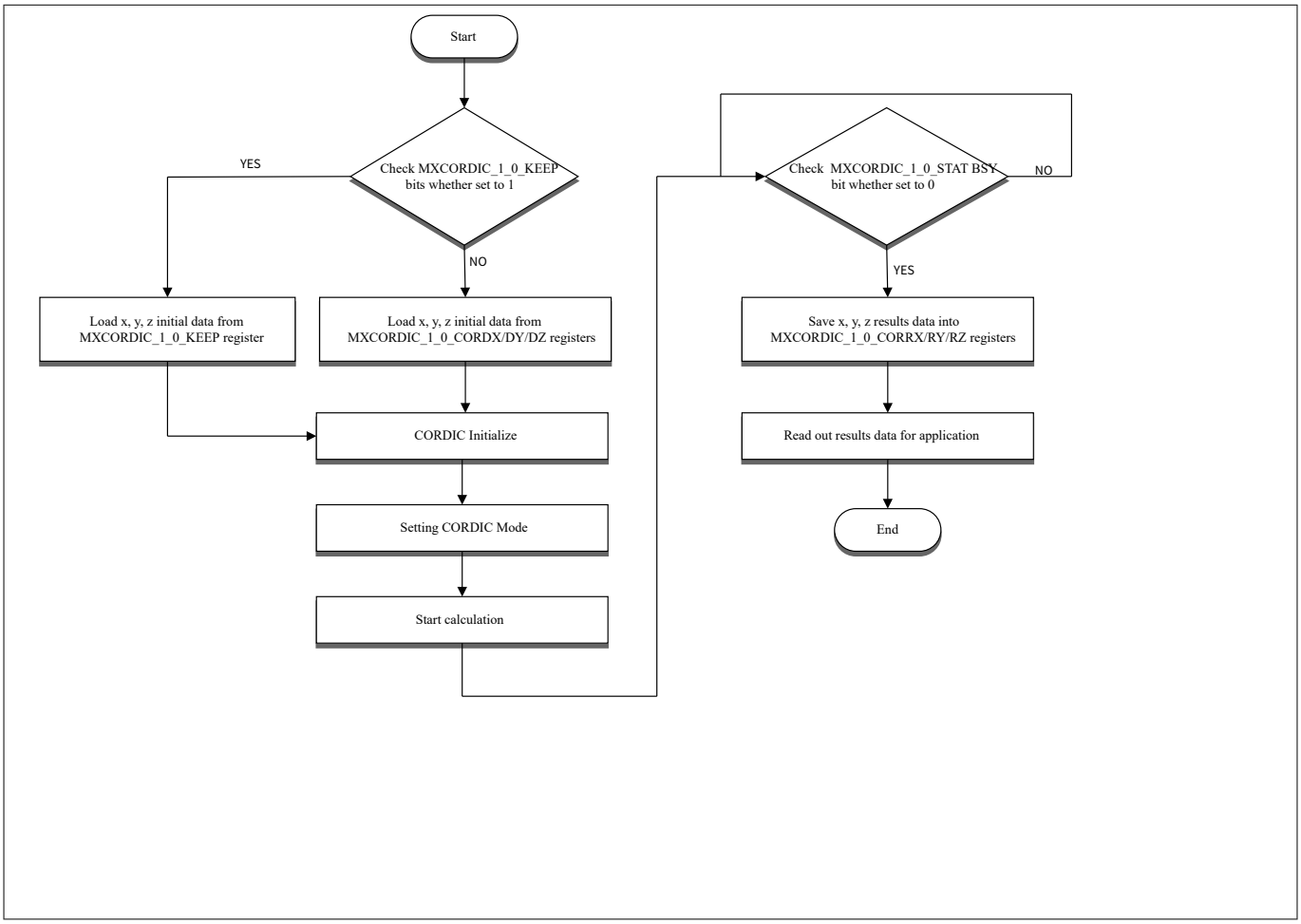


図 34　　CORDIC 動作

11.3.2　　正規化された結果データ

すべての動作モードで、CORDIC コプロセッサは、下式に示すように X と Y を正規化した結果データを返します。

X または Y 結果データ = (CORDIC 計算データ)/MPS

***注**:　　MPS はマグニチュードプレスケーラーを表します。*

一方、Z 結果データの解釈は異なり、これも使用する CORDIC 関数に依存します。

サーキュラー (三角)関数および **双曲線** 関数の場合、アクセス可能な Z の結果データは正規化された整数値です; 範囲 $[-\pi, ((2^{23}-1)/2^{23})\pi]$ の角度は $[-2^{23}, (2^{23}-1)]$ で表されます。CORDIC コプロセッサは、Z データがこのスケーリングで解釈されることを想定しています。

入力 Z 初期データ = 実 Z 初期値 (ラジアン単位)·0x800000/π

実 Z 結果値 (ラジアン単位) = Z 結果データ· π/0x800000

CORDIC の計算データには、回転またはベクトル化による固有のゲイン係数 K が含まれます。下表に示すように、値 K は各 CORDIC 関数ごとに異なります。

表 83 近似 CORDIC ゲイン K

関数	近似ゲイン K
円回転	1.646760258121
双曲線関数	0.828159360960

11.3.3 動作モード

次の図に、CORDIC コプロセッサの動作モードの概要を示します。この表では、X、Y、Z が初期データを表しています。

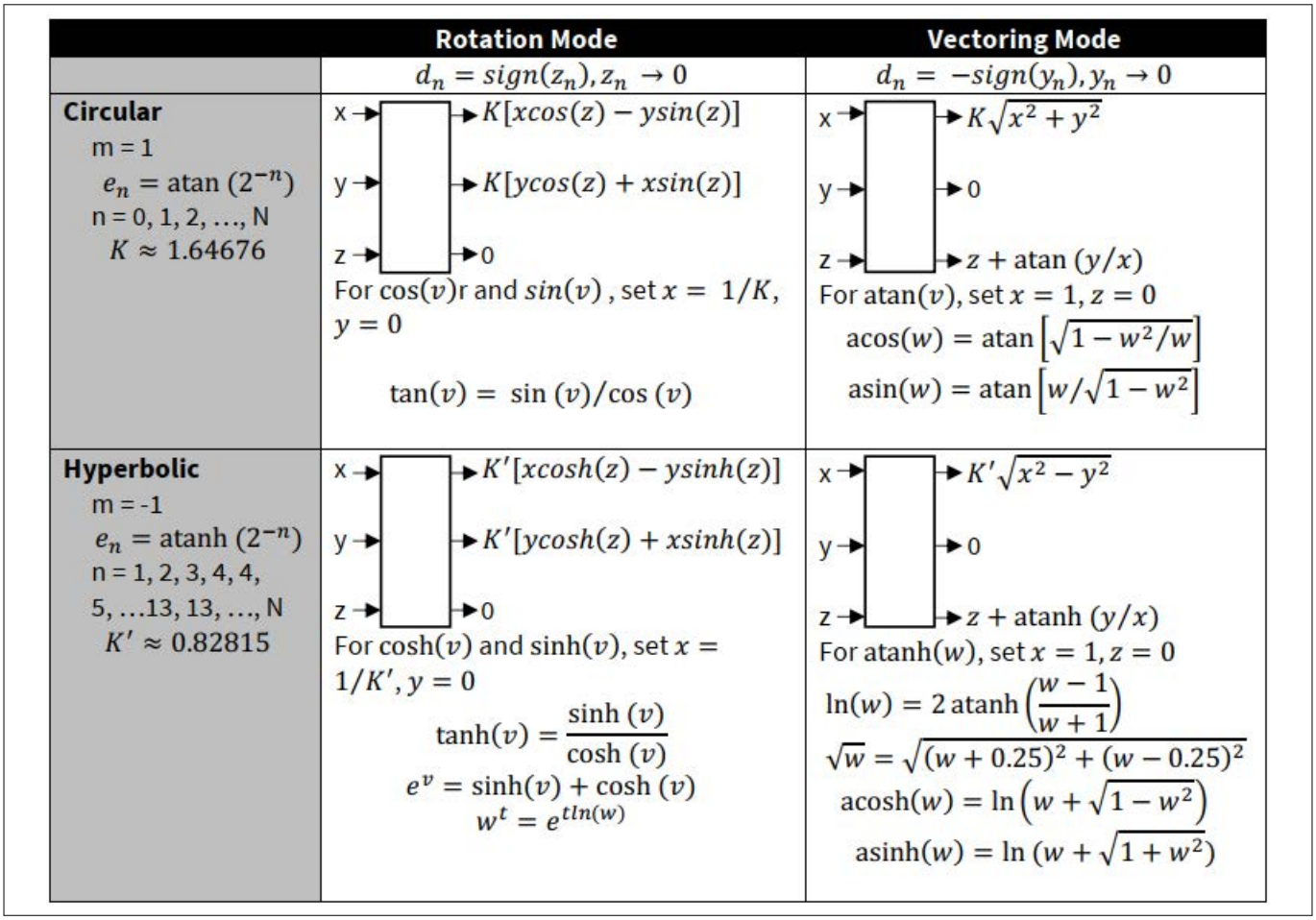


図 35 CORDIC 動作モード

使用上の注意:

- それぞれの関数を解く際には、結果の収束を保証するために、収束領域内の意味のある初期値で CORDIC データ (X、Y、Z) を初期化します。表内の「有用な領域」は、CORDIC アルゴリズムに対してサポートされている収束領域を網羅しており、関数の非有意範囲は除外されています。
- 関数の制限を考慮する必要があります。例えば、atan (Y/X) の初期値 X = 0 を設定しても意味がありません。このような関数の制限に違反すると、CORDIC の結果データが支離滅裂になることがあります。
- すべてのデータ入力は 2 の補数として処理され、扱われます。唯一の例外は、X 結果データ (のみ) を符号なし値として読み出すためのユーザーオプションです。
- 結果データが常に正で初期データより大きい唯一のケースは、循環ベクトル モードの X 結果データ (のみ) です。したがって、ユーザーは MSB ビットを符号ビットではなくデータビットとして使用することを選択することがあります。X_USIGN = 1 に設定することにより、X 結果データは符号なしデータとして処理されます。

- 円関数と双曲線関数では、対応する LUT が固定されているため、Z データは常に符号付き整数 Q (1,28) として扱われ、Q (1,23) としてアクセスできます。LUT には、n = 0, 1, 2, ..., 27 に対して atan(2^{-n})、n = 1, 2, ..., 27 に対して atanh(2^{-n}) のスケーリングされた整数値 Q(1,28) が格納されています。これにより角度 [-π, ((2^{23}-1)/2^{23})π] の範囲が整数値 [-1,(2^{23}-1)/2^{23}] で表現されます。したがって、Z データは、これらの CORDIC 関数の角度[-π, ((2^{23}-1)/2^{23})π]を表すように制限されます (収束領域を考慮しません)。この範囲外の Z の計算値はオーバーフローエラーとなります。
- MPS 設定は Z データには影響しません。オーバーフローや不正確な結果データを防ぐため、Z 初期データを適切に初期化します。
- CORDIC コプロセッサは、ユーザーが MPS > 1 を適切に設定すると、X データおよび Y データの内部オーバーフローは発生せず、読み出し結果データは完全なままになります。ただし、このような場合、MPS を高く設定するほど、LSB ビットが失われるため、結果データの解像度が低下することに注意してください。
- 双曲線回転モードは、初期 Y データをゼロに設定しなければいけないため、結果精度の点で限界があります。言い換えれば、CORDIC コプロセッサは、1 回の計算で cosh (Z)+/-sinh (Z) の正確な結果を返すことはできません。

11.3.4 収束領域

ここでは、回転またはベクトル化モードの CORDIC 関数の種類によって異なる CORDIC 収束の固有の限界に関する理論については説明しません。結果データの収束のためには、使用される動作モードによって、初期データの大きさや値、対応する有用なデータ形式に制限があります。CORDIC の結果データの収束に関しては、一般的に以下のことが適用されます。

回転モード: Z のデータは 0 に収束する必要があります。初期の Z データは、$\Sigma d_n \cdot e_n$ 以下でなければならず、e_n は反復回数の n に対して、常に減少します。つまり、|Z|≦LUT の和です。円関数では、これは |Z|≦整数値が 1.74 ラジアンを表すことを意味します。双曲線関数では、|Z|≦整数値は 1.11 ラジアンを表します。

ベクタリング モード: Y データは 0 に収束する必要があります。X と Y の初期値は、対応する LUT に依存する Z 関数によって制限されます。円関数では、| atan (Y/X) | ≦1.74 ラジアンです。双曲線関数では、| atanh (Y/X) | ≦1.11 ラジアンとなります。ベクタリング モードでは、X > 0 であることが追加要件となります。

円形回転モード: 角度[-π, ((2^{23}-1)/2^{23})π]を表す Z 入力[-2^{23}, (2^{23}-1)]の全範囲がサポートされています。オーバーフローを考慮する以外は、X と Y の初期入力に制限はありませんが、これは MPS 設定で克服できます。

サーキュラー ベクタリング モード: X と Y の全入力範囲[-2^{23}, (2^{23}-1)]がサポートされ、Z の初期値は、Z の結果データのオーバーフローを防ぐために | Z | ≦π/2 を満たす必要があります。

CORDIC コプロセッサの動作モードは通常、これらの収束限界にホールドされますが、円回転モードおよび円ベクトルモードには例外があり、これらは広範な入力をサポートするために追加の前処理および後処理ロジックを使用します。

注: また、結果データの意味 (例えば、ゼロで割ることは意味がない) など、関数の制限についても考慮する必要があります。各主要関数の図 35 内に含まれる「有用領域」は、CORDIC 収束と関数の有用範囲の両方をカバーしようとするものです。

注: 入力値は収束領域内に収まることがありますが、これは CORDIC 結果データの一定レベルの精度を保証するものではありません。

11.3.5 オーバーフローへの配慮

収束領域の他に、結果データのオーバーフローを防ぐために、入力データの大きさの制限を考慮します。

すべての動作モードで、CORDIC コプロセッサはデータのオーバーフローを同様に処理します。X および Y データのオーバーフローを防止するためには、MPS ビットを正しく設定してください。このビットの値は、CORDIC コプロセッサの動作モードとアプリケーションデータに基づいて部分的に決まります。

MPS 設定は Z データには影響しません。円関数と双曲線関数の場合、Z の値が[-π, ((2^23-1)/2^23)π] の範囲外であると、Z データのオーバーフローエラーが発生します。カーネルデータ Z は、範囲[-π, ((2^23-1)/2^23)π]を範囲[-2^23, (2^23-1)]にスケーリングした値を持つので、Z データの書き込まれた値と読み出された値は常に正規化されていることに注意してください。Z が実数値である一次関数では、Z の大きさは 4 ビットを超えてはいけません。

CORDIC コプロセッサは、2 の補数形式でデータ X、Y、および Z の (初期) 入力を受け付けます。結果データも 2 の補数形式です。唯一の例外は、サーキュラーベクタリングモードの X 結果データです。X 結果データのデフォルト データ フォーマットは 2 の補数ですが、CON.X_USIGN = 1 ビットにより、X 結果データを符号なし値として読み出すよう選択できます。このオプションは、MSB ビットがデータビットとなるため、(MPS 設定と合わせて) X 結果データのオーバーフローを防げます。CON.X_USIGN = 1 に設定することは、サーキュラー ベクタリング モードで動作する場合にのみ有効であることに注意してください。

11.3.6 データフォーマット

CORDIC コプロセッサは、2 の補数形式の初期データ (24 ビット) X、Y、Z 入力を受け付けます。結果データ (24 ビット) も 2 の補数形式です。唯一の例外は、サーキュラーベクタリングモードの X 結果データです。X 結果データのデフォルト データ フォーマットは 2 の補数ですが、CON.X_USIGN = 1 ビットにより、X 結果データを符号なし値として読み出すよう選択できます。このオプションは、MSB ビットがデータビットとなるため、(MPS 設定と合わせて) X 結果データのオーバーフローを防げます。CON.X_USIGN = 1 に設定することは、サーキュラー ベクタリング モードで動作する場合にのみ有効であることに注意してください。

***注**: X、Y、Z 初期データレジスタ(MXCORDIC_1_0_CORDX、MXCORDIC_1_0_CORDY、MXCORDIC_1_0_CORDZ) と結果データレジスタ(MXCORDIC_1_0_CORRX、MXCORDIC_1_0_CORRY、MXCORDIC_1_0_CORRZ) はすべてビット[31:8]を使用してデータを保存し、ビット[7:0]は予約されています。*

一般に、X と Y の入力データは整数または有理数 (分数) です。ただし、いずれの計算においても、X と Y のデータ形式は一致していなければいけません。また、分数の場合、X と Y は小数点以下のビット数が同じでなければいけません。Z データは常に、円関数または双曲線関数の正規化係数に基づく整数として扱われます。

データの正規化の詳細については、正規化された結果データ セクションを参照してください。

11.3.7 性能

CORDIC の計算時間は、精度が上がるにつれて直線的に増加します。精度を上げるには反復回数を増やす必要があり、そのためにはデータ パラメーターの幅を広げる必要があります。CORDIC コプロセッサは、データシフトにバレルシフタを使用します。1 回の計算で 28 回の反復を固定回数行う場合、計算開始から EVFR.CDEOC フラグがセットされるまでの合計時間は 28/4+3 = 10 カーネルクロックサイクルとなります。EVFR.CDERR フラグは、EVFR.CDEOC が発生してから 1 カーネル クロック サイクルの間のみ有効であることに注意してください。この 1 回の完全な計算のタイミングは、追加のデータ処理を伴うモードに適用されます。

***注**: 双曲線関数モードでは、反復は n = 1 から始まり、収束するためには反復 n = 4 と 13 を繰り返さなければいけません。その結果、n = 28 を除いて、合計 n = 27 の反復が実行されます。*

***注**: 上記のタイミングは、ソフトウェアによる初期データのロードと、6 つのデータレジスタへの最終結果データの読み出しにかかる時間を除いたものです。*

11.3.8 ルックアップテーブル

ここでは、CORDIC コプロセッサカーネルで使用されるルックアップテーブルについて説明します。LUT は 29 ビット幅です。atan および atanh LUT の各エントリーは、1 符号ビット (MSB) と、それに続く 28 ビットの小数部 Q (1,28) に分割されます。

Z データは実際の角度を正規化したものです。内部スケーリングは、$[-\pi, ((2^{23}-1)/2^{23})\pi]$が$[-2^{28}, (2^{28}-1)]$と等価になるようなものです。アドレス指定時に 24 ビットのデータがデータ バスに転送されるため、最下位 (LSB) 5 ビットは切り捨てられます。ユーザーから見ると、角度$[-\pi, ((2^{23}-1)/2^{23})\pi]$は、したがって、範囲$[-2^{23}, (2^{23}-1)]$で表されます。

表 84　サーキュラー (三角関数) モード ルックアップ テーブル

反復回数 n	16 進数のスケール付き atan (2^{-n})	反復回数 n	16 進数のスケール付き atan (2^{-n})
0	4000000	14	145F
1	25C80A4	15	A30
2	13F670B	16	518
3	A2223B	17	28C
4	5161A8	18	146
5	28BAFC	19	A3
6	145EC4	20	51
7	A2F8B	21	29
8	517CA	22	14
9	28BE6	23	A
10	145F3	24	5
11	A2FA	25	3
12	517D	26	1
13	28BE	27	1

双曲線関数モードでは、反復 n = 4 と n = 13 が繰り返され、反復 n = 0 と n = 27 はスキップされます。サーキュラー (三角関数) モードと同じクロック数で収束させるため、n = 27 はスキップされます。

表 85　双曲線関数モードルックアップテーブル

反復回数 n	16 進数のスケール付き atanh (2^{-n})	反復回数 n	16 進数のスケール付き atanh (2^{-n})
0	-	14	145F
1	2CC2F12	15	A30
2	14D01AC	16	518
3	A3D4E0	17	28C
4	5197FC	18	146
5	28C1C7	19	A3
6	145F9D	20	51
7	A2FA6	21	29
8	517CE	22	14
9	28BE6	23	A

(続く)

表 85 (続き) 双曲線関数モードルックアップテーブル

反復回数 n	16 進数のスケール付き atanh (2^{-n})	反復回数 n	16 進数のスケール付き atanh (2^{-n})
10	145F3	24	5
11	A2FA	25	3
12	517D	26	1
13	28BE	27	-

11.3.9 割込み生成

CORDIC 割込み要求信号は、下図のように生成されます。

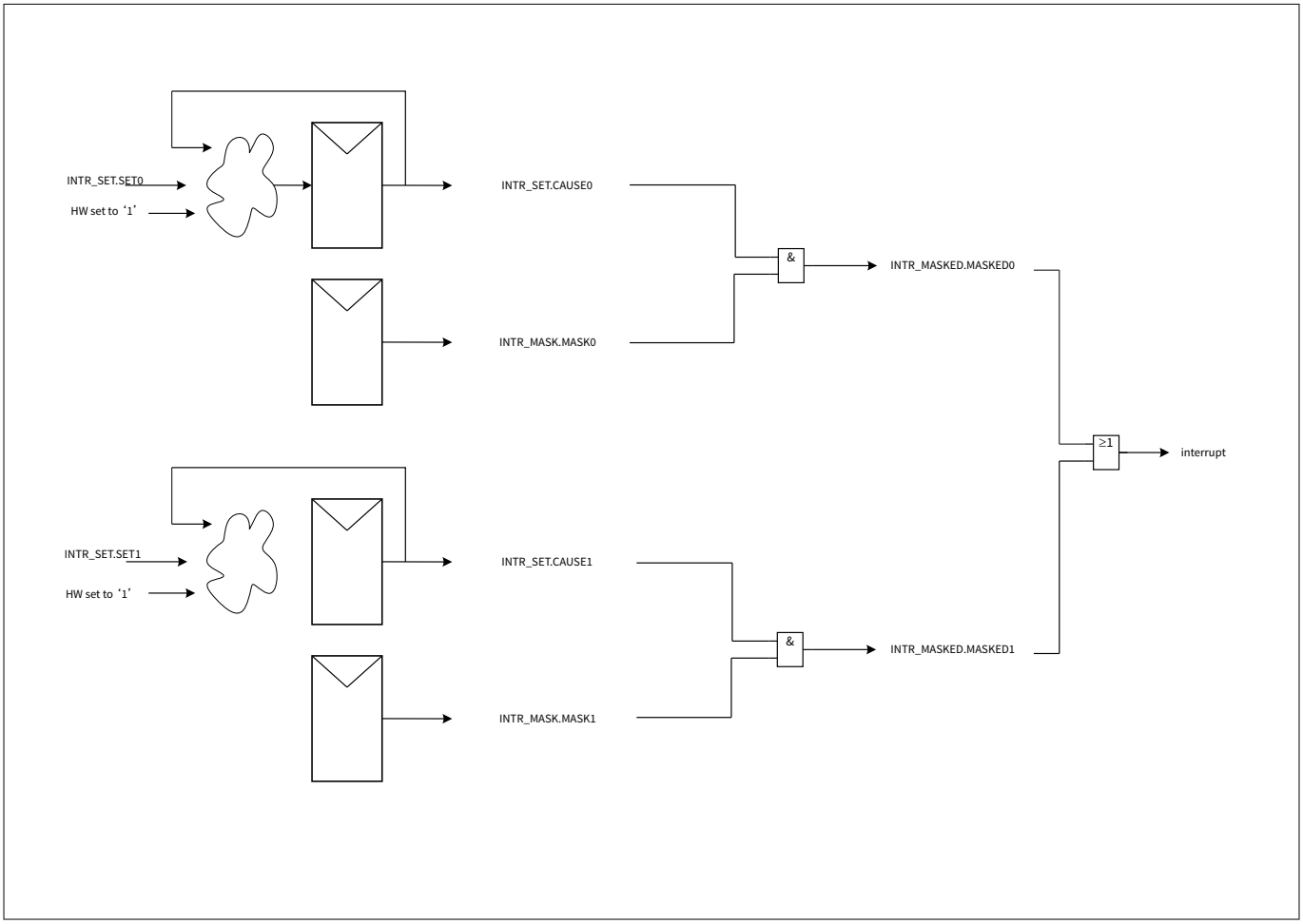


図 36 CORDIC 割込み生成

ハードウェアは、イベント信号が検出されると、MXCORDIC_1_0_INTR レジスタの割込みフラグ ビットを 1 に設定できます。MXCORDIC_1_0_INTR.CDEOC は計算終了イベントフラグです。MXCORDIC_1_0_INTR.CDERR は計算エラーイベントフラグです。ソフトウェアは MXCORDIC_1_0_INT_SET レジスタの割込み発生ビットを 1 に設定できます。MXCORDIC_1_0_INTR_MASK レジスタは、MXCORDIC_1_0_INT 内の同位置割込み発生ビットの割込みマスクビットを保持できます。同位置割込み原因ビットは、そのマスクビットが'1'の場合にのみ割込みを発生させます。MXCORDIC_1_0_INTR_MASKED レジスタは、MXCORDIC_1_0_INT_INTR レジスタと MXCORDIC_1_0_INTR_MASK レジスタの論理 AND です。ユーザーは、MXCORDIC_1_0_INTR_MASKED レジスタを介して、割込み要求信号を直接読み出せます。

注: PSOC™ Control C3 デバイスでは、CORDIC ハードウェアによる計算が高速化するため、ノンブロッキング関数 API を使用しても速度の利点は得られない場合があります。ユーザーはブロック機能を直接使用することを推奨します。

11.4 電源、クロック、リセット

ここでは、このブロックで使用される電源、クロック、リセットの概要について説明します。

11.4.1 電源

CORDIC はアクティブ電力モードのみ使用できます。システムがディープスリープ モードにある場合、CORDIC が配置されている電源ドメインはパワーダウンされます。CORDIC 内部のフロップはすべて、保持されずにリセットされます。CORDIC には、電源関連のインターフェース (P チャネルまたは Q チャネル) はありません。CORDIC IP は、内部電源ドメインやパワースイッチを実装していません。

11.4.2 クロック

CORDIC は、1 つのクロック入力 (*clk_sys*) で動作します。このクロックは m33syscpuss クロックに同期しています。クロックは、SL_CTL.ENABLED MMIO によって (MXSPERI 内で) 集中的にクロックゲートを行えます。

11.4.3 リセット

CORDIC には、1 つの非同期リセット信号があります。リセットは非同期にアサートできますが、システム クロックに同期してデアサートする必要があります。このモジュールの内部にはリセット同期はありません。リセットの同期は MXSPERI 内部で行われます。

11.5 レジスタの概要

表 86 CORDIC レジスタの概要

レジスタ	機能
MXCORDIC_1_0_CTL	モジュール有効/無効レジスタ
MXCORDIC_1_0_ID	モジュール ID レジスタ
MXCORDIC_1_0_INTR	割込み原因レジスタ
MXCORDIC_1_0_INTR_SET	割込みセットレジスタ
MXCORDIC_1_0_INTR_MASK	マスクされた割込みレジスタ
MXCORDIC_1_0_INTR_MASKED	割込みマスクドレジスタ
MXCORDIC_1_0_KEEP	CORDIC キープレジスタ
MXCORDIC_1_0_CON	CORDIC 制御レジスタ
MXCORDIC_1_0_CORDX	CORDIC X データレジスタ
MXCORDIC_1_0_CORDY	CORDIC Y データレジスタ
MXCORDIC_1_0_CORDZ	CORDIC Z データレジスタ
MXCORDIC_1_0_CORRX	CORDIC X 結果レジスタ

(続く)

表 86 (続き) CORDIC レジスタの概要

レジスタ	機能
MXCORDIC_1_0_CORRY	CORDIC Y 結果レジスタ
MXCORDIC_1_0_CORRZ	CORDIC Z 結果レジスタ
MXCORDIC_1_0_STAT	CORDIC ステータスレジスタ
MXCORDIC_1_0_START_CMD	CORDIC 開始コマンド レジスタ

12 プログラムおよびデバッグ インターフェース

PSOC™ Control C3 MCU のプログラムおよびデバッグのインターフェースは、外部デバイスがプログラミングとデバッグを実行するための通信ゲートウェイを提供します。外部デバイスは、インフィニオンが提供するプログラマとデバッガー、またはプログラミングとデバッグをサポートするサードパーティ製デバイスです。シリアル ワイヤ デバッグ (SWD) または JTAG インターフェースは、外部デバイスと PSOC™ Control C3 MCU の通信プロトコルとして使用できます。

12.1 特長

プログラムおよびデバッグ インターフェースには以下の特長があります。

- JTAG または SWD インターフェースを使用したプログラムおよびデバッグに対応
- CM33 は、8 ビットの Instrumentation Trace Macrocell (ITM) と 4 ビットの Embedded Trace Macrocell (ETM) のトレースに対応しています
- クロス トリガーリング インターフェース (CTI) とクロス トリガーリング マトリックス (CTM) に対応しています
- CM33 は、8 つのハードウェアブレークポイントと 4 つのウォッチポイントに対応しています
- デバッグ期間中にシステム内のすべてのメモリとレジスタ (コアが動作中または停止した時の Cortex®-CM33 レジスタ バンクも含む) への読み出し/書き込みアクセスが可能です

12.2 アーキテクチャ

図 37 は PSOC™ Control C3 MCU のプログラムとデバッグ インターフェースのブロック図です。デバッグ アクセス ポート (DAP) は、プログラムおよびデバッグ インターフェースとして機能します。外部プログラマまたはデバッガ (ホストとも呼ばれる) は、シリアル ワイヤ デバッグ (SWD) またはジョイント テスト アクション グループ (JTAG) インターフェースのどちらかを使用して、PSOC™ Control C3 MCU (ターゲット) の DAP (デバッグアクセスポイント) と通信します。物理デバッグポートピンは、高速 I/O マトリックス (HSIOM) をとおして DAP と通信します。HSIOM の詳細については I/O サブシステムを参照してください。

デバッグ インフラストラクチャーは以下のグループに分かれています。

- DAP (デバッグホストがターゲットに接続するためのピンインターフェースを提供)
- Cortex®-M33 コアのデバッグコンポーネント
- その他のデバッグ インフラストラクチャー (CTM、TPIU、システム ROM テーブルを含む)

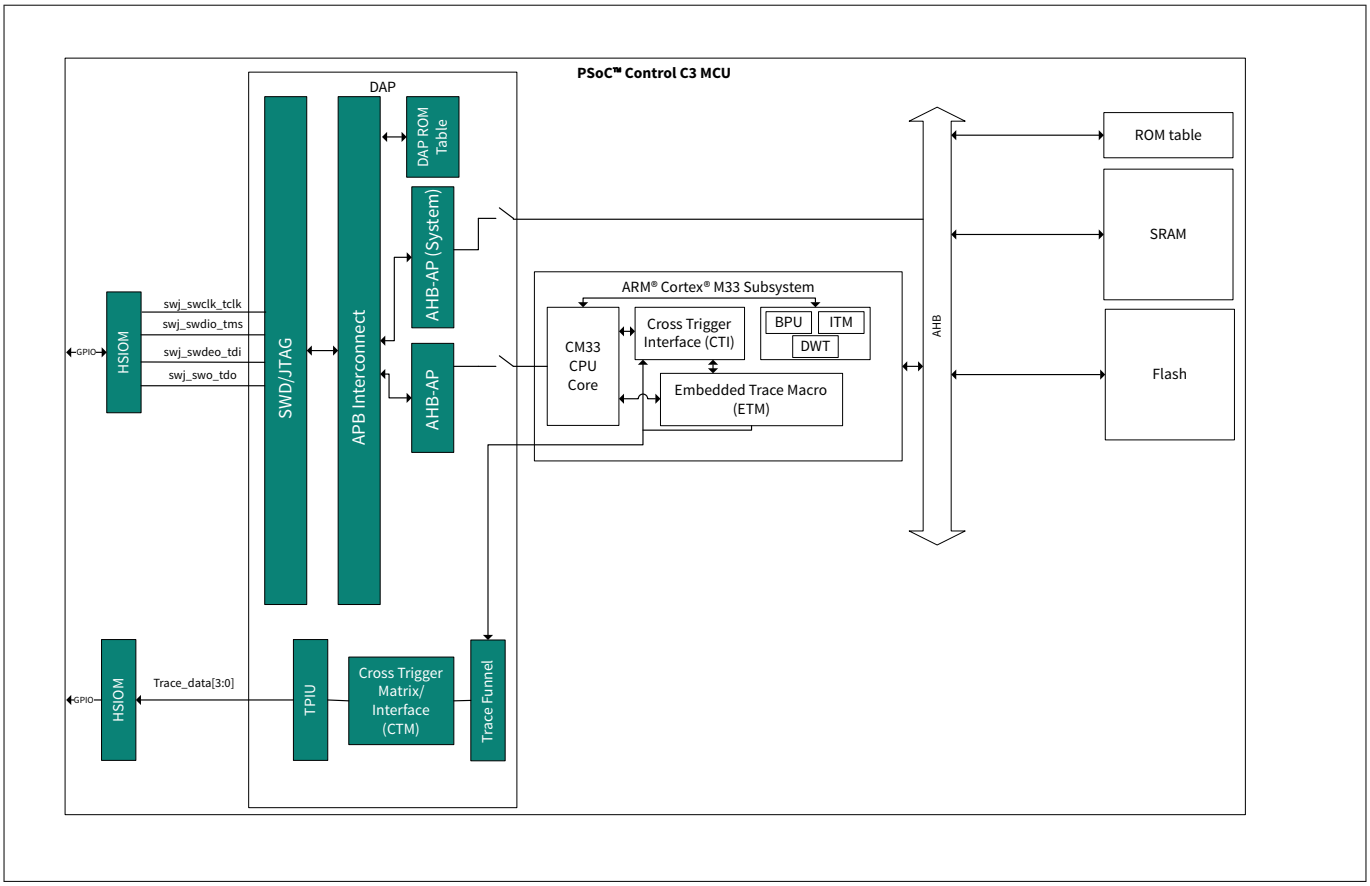


図 37　　プログラミングおよびデバッグ アーキテクチャ

DAP は、Arm®特有の AHB インターフェース (advanced high-performance bus) を使用してコアと通信します。外部ホストは、DAP を通じてデバイス全体を効果的に制御し、プログラミングとデバッグの両方を実行できます。

以下は、さまざまなデバッグとトレースのコンポーネントです。

- デバッグ コンポーネント
  - デバッグ制御とアクセス用の JTAG と SWD
- トレース ソース コンポーネント
- トレース シンク コンポーネント
- トレースポート インターフェース ユニット (TPIU): トレース情報をチップから外部トレースポート アナライザに転送します。
- クロストリガー コンポーネント
  - クロストリガー インターフェース (CTI)
  - クロストリガー マトリックス (CTM)

12.2.1　　デバッグアクセスポート (DAP)

DAP は、SWD/JTAG インターフェース (SWJ) の組み合わせで構成されます。SWJ には、JTAG インターフェース (リセット後のデフォルト) か SWD インターフェースかを決定する SWD リスナーが含まれています。JTAG インターフェースと SWD インターフェースはピンを共有しているため、相互に排他的です。

デバッグ ポート (DP) は、APBIC (Advanced Peripheral Bus Interconnect) インフラストラクチャーを使用してデバッグ コンポーネントと通信し、2 つの AHB アクセス ポートの 1 つに接続します。

- Cortex® -M33 AP は AHB デバッグポートに直接接続し、Cortex® -M33 の内部デバッグコンポーネントにアクセスできます。これにより、Cortex® -M33 AHB マスター インターフェースを介してシステムの残りの部分にアク

セスできるようになります。AP は、Cortex® -M33 上で動作するアプリケーションと同じビューをデバッグホストに提供します。これには、Cortex® -M33 サブシステム内の他のデバッグコンポーネントの MMIO へのアクセスも含まれます。これらのデバッグコンポーネントは、Cortex® -M33 CPU からアクセスできますが、他の AP や Cortex® -M33 コアからはアクセスできません。

- System-AP は、AHB mux を通じて、ユーザー SRAM やフラッシュなどのシステムの残りの部分にアクセスできます。これにより、他の方法ではアクセスできないシステム ROM テーブルにアクセスできるようになります。システム ROM テーブルはチップ ID を提供しますが、それ以外は空です。

デバッグおよびアクセス ポート レジスタはどちらも Arm® IP で、Arm® Debug Interface Architecture Specification ADIv6.0 に記載されています。

***注**:　　DAP は Cortex® M33 と非同期で動作します。*

12.2.1.1 DAP 電源ドメイン

デバッガ全体は、Cortex® -M33 サブシステムと同じ電源ドメインを使用します。ほとんどすべてのデバッグ コンポーネントは、アクティブ電源ドメインの一部です。唯一の例外は SWD で、SWCLKTCK と SWDIOTMS の 2 つのピンを使用し、ディープスリープ時に動作します。これにより、SWD デバッグホストは、ディープスリープ中、アプリケーション実行中、またはパワーダウン中に接続できます。

デバッガがチップに接続された後、操作を行う前にチップをアクティブ状態にする必要があります。このため、SWD/JTAG-DP には、CDBGPWRUPREQ および CSYSPWRUPREQ の 2 つの電源要求ビットを持つレジスタ (DP_CTL_STAT) があります。これらのビットは、それぞれデバッグ電源とシステム電源を要求します。デバッグ セッションの間、セットされた状態のままでなければいけません。

JTAG ピンはアクティブ モードでのみ動作します。つまり、システムがディープスリープにあるときは、JTAG デバッグと JTAG バウンダリー スキャンは利用できません。JTAG 機能は、チップのパワーオンリセット後にのみ利用可能です。

操作の前にチップをアクティブ状態にします。デバッガが接続されると、CM33 はアクティブ電源ドメインに切り替わります。他のすべてのリソースは、認証されたソフトウェアまたはデバッグ用ソフトウェアによってアクティブ状態にされなければいけません。

12.2.1.2 DAP セキュリティ

DAP ピンはポリシーファイルによって制御されます。標準的なポリシー ファイルを持つ工場出荷時のデバイスは、SWD または JTAG インターフェースを設定なしで接続できるように DAP ピンを設定します。セキュアポリシーファイルを使用する場合は、DAP ピンを設定し、セキュアポリシーファイルで有効にします。セキュアポリシーファイルの詳細については、セキュア ブートセクションを参照してください。

Cortex®-M33 には独自の CPUSS_AP_CTL レジスタがあり、侵襲性/非侵襲性デバッグやセキュアな特権侵襲性/非侵襲性デバッグなどのデバッグ機能を設定できます。これらのデバッグ機能の設定は、セキュア エンクレーブがフラッシュの保護領域に AP_CTL レジスタを設定するために使用するポリシー ファイルで行います。SYS_AP は Cortex® M33 の AP_CTL レジスタによって制御され、ポリシー ファイルで AP_CTL.SYS_DISABLE を‘00’に設定することで有効になります。

DISABLE で終わるすべての AP_CTL ポリシーフィールド名は 2 ビットで、‘00’は有効、‘01’は無効、‘1x’は永久に無効です。セキュア/非セキュアデバッグ機能を有効にするためには、AP_CTL.CM33_0_DISABLE を‘00’に設定する必要があります。

次の表に、Cortex®-M33 の非セキュアデバッグ用ポリシーファイルのレジスタ設定を示します。

表 87　　非セキュアなデバッグ

デバッグ モード	AP_CTL.CM33_0_DBG_DISABLE	AP_CTL.CM33_0_NID_DISABLE
侵襲性	00	01
非侵襲性	01	00

セキュアなデバッグ機能については、Cortex® -M33 のポリシーファイルに以下を設定してください。

表 88　　Cortex®-M33 セキュアデバッグ

デバッグ モード	AP_CTL.CM33_0_SPID_DISABLE	AP_CTL.CM33_0_SPNID_DISABLE	AP_CTL.CM33_0_SECURE_DISABLE
セキュア侵襲性	00	01	00
セキュア非侵襲性	01	00	00

注:　セキュア侵襲性デバッグを有効にするためには、CM33_0_DBG_DISABLE を‘00’に設定します。セキュア非侵襲デバッグを有効にするためには、CM33_0_NID_DISABLE を‘00’に設定します。

12.2.2　ROM テーブル

CoreSight アーキテクチャは、検出可能なデバッグ トポロジーに対応しています。つまり、デバッグ ホストはチップの身元を知らなくても、チップ上で利用可能なすべてのデバッグ コンポーネントを見つけられます。チップ アイデンティティとデバッグ トポロジーを検出するためには、CoreSight ROM テーブルを使用します。

ROM テーブルはツリー構造になっています。各 AP には、その AP のルート ROM テーブルのベースを指す 32 ビットのアドレスを含むレジスタがあります。デバッガーのルート ROM テーブルは「Dap ROM Table」で、異なる AP を指しています。Coresight DP コンポーネントには、DAP ROM テーブルを指す BASEPTR というレジスタがあります。PSOC™ Control C3 MCU には、このようなルート ROM テーブルが 3 つあります (DAP、M33、System)。

各 ROM テーブルには 32 ビットのエントリーがあり、次のレベルの ROM テーブルのベースか、リーフ デバッグ コンポーネントのベースを指すアドレス ポインターがあります。各 ROM テーブルには、メーカー、部品番号、メジャー リビジョン番号、マイナー リビジョン番号を特定するための JEDEC 準拠の識別子を保持する ID レジスタのセットも含まれています。PSOC™ Control C3 MCU のすべての ROM テーブルで、これらの ID は同じです。各 ROM テーブルと CoreSight 対応コンポーネントには、コンポーネント識別レジスタも含まれています。

12.2.3　トレース

Cortex® -M33 には、ITM (Instrumentation Trace Macrocell) と ETM (Embedded Trace Macrocell) があります。これらのコンポーネントは、CPU からのプログラム実行フローをキャプチャし、ATB (Advanced Trace Bus) 上にトレース出力を生成します。これらのトレース出力は、複数のトレースストリームを単一のバスに結合するコアサイトトレースファネル (CSTF) に渡されます。

デバッガには 4 ビットのトレースポート インターフェース ユニット (TPIU) が内蔵されています。これは、外部トレースポートアナライザー (TPA) でトレースをキャプチャできるように、IOSS インターフェースを介して外部ピンを駆動するために使用されます。詳細については、Arm® Debug Interface Architecture Specification ADIv6.0 を参照してください。

12.2.4　エンベデッド クロストリガー

Arm® CoreSight には、異なるデバッグコンポーネント間でイベントをトリガー (送受信) する機能を提供する Embedded Cross Triggering (ECT) が搭載されています。トリガーイベントの使用例:

- CPU の起動または停止
- トレース バッファがフル、フルでない、またはその他のイベントに基づいて、命令トレースを開始または停止します。

CoreSight は、Cross-Trigger Interface (CTI) と Cross Trigger Matrix (CTM) という 2 つの主要コンポーネントを使用して ECT に対応しています。CTI は他のデバッグ コンポーネントとのインターフェースを持ち、必要に応じてトリガーを送ったり同期させたりします。CTM は複数の CTI を接続し、ある CTI から別の CTI にイベントを伝達できます。

デバッグ インフラストラクチャーは、主にイベントを TPIU とシステムトリガーに接続する CTI を提供します。

注: CPU には、CPU コアとデバッグ コンポーネント間のイベントを接続する内部 CTI があり、CPU コアと CPU 内のデバッグ コンポーネント (ETM、DWT など) 間のイベントを接続する内部 CTI があります。これら 2 つの CTI は CTM を介して接続され、CTM はこれら 2 つの CTI 間でイベントを送受信できます。

12.3 シリアルワイヤデバッグ (SWD) インターフェース

PSOC™ Control C3 MCU は、SWD インターフェースを介したプログラムおよびデバッグに対応しています。SWD プロトコルは、パケットベースのシリアル トランザクション プロトコルです。端子レベルでは双方向データ信号 (SWDIO) と一方向クロック信号 (SWDCK) を使用します。クロック ラインは常にホスト プログラマが駆動し、データ ラインはホストまたはターゲットのいずれかが駆動します。完全なデータ転送 (1 つの SWD パケット) は 3 つのフェーズで構成されます。

- ホスト パケット要求段階–ホストは PSOC™ Control C3 ターゲットに要求を発行します。
- ターゲット ACK 応答段階– PSOC™ Control C3 ターゲットはホストに確認応答を送信します。
- データ転送段階– 転送方向に応じてホストまたはターゲットがバスにデータを書き込みます。

SWD 実装の詳細については、Arm® Debug Interface Architecture Specification ADIv6.0 を参照してください。

12.4 JTAG インターフェース

IC の高ピン密度化に対応するため、ジョイント テスト アクション グループ (JTAG) は、別のテスト インターフェースを介して IC 上のピンを制御 (およびその値を読み出し) して回路基板をテストする方法を提案しました。後に IEEE 規格 1149.1-2001 として正式化されたこのソリューションは、IC の全ピンに渡って配線されたシリアル シフト レジスタの概念に基づいています。各ピンの回路は、バウンダリー スキャン セルと呼ばれる多目的素子で補われています。PSOC™ Control C3 MCU では、ほとんどの GPIO ポート ピンにバウンダリー スキャン セルが関連付けられています (I/O サブシステムセクションの GPIO ブロック図を参照)。バウンダリー スキャン セル内の値を制御するために使用されるインターフェースは、テスト アクセス ポート (TAP) と呼ばれ、一般に JTAG インターフェースとして知られています。テスト データ イン (TDI)、テスト データ アウト (TDO)、テスト モード セレクト (TMS) の 3 つの信号で構成されています。他の信号をクロックするクロック信号 (TCK) も含まれ、オプションの入力信号であるデバッグ TAP リセット (TRST) も存在します。TDI、TMS、TCK はすべてデバイスへの入力で、TDO はデバイスからの出力です。JTAG インターフェースの詳細については、Arm® Debug Interface Architecture Specification ADIv6.0 を参照してください。

12.5 MCU のプログラミング

PSOC™ Control C3 は下記のシーケンスでプログラムされます。

1. PSOC™ Control C3 MCU で SWD ポートを開通させます。
2. プログラム モードに移行します。
3. シリコン ID 確認、フラッシュ プログラム、フラッシュ検証、チェックサム検証などといったデバイス プログラム ルーチンを実行します。

12.5.1 SWD ポートの開通

外部リセット (XRES) ピンをアサートしてデバイスをリセットすると、DP はデフォルトで JTAG 動作になります。SWD プロトコルに切り替えるには、XRES を使ってデバイスをリセットし、リセット状態にします。ホストは以下の SWD 接続シーケンスを送信する必要があります。

1. SWDIO を、SWCLK サイクル 5 回以上 HIGH にします。これにより、JTAG TAP ステートマシンはテストロジックリセット状態になり、IDCODE 命令が選択されます。
2. SWDIO に 31 ビットのシーケンス (0x2EEEEEE6、MSB first) を書き込むことにより、デバイスを JTAG から休止状態に変更します。デバイスはこれにより休止状態になりました。
3. SWDIO を SWCLK サイクル 8 回以上 HIGH にします。これにより、デバイスがセレクション アラート シーケンスの途中でないことが確実になります。
4. SWDIO で、128 ビットのセレクション アラート シーケンス (0x49CF9046 A9B4A161 97F5BBC7 45703D98、MSB first) を送信します。
5. SWCLK 4 サイクルの間、SWDIO を LOW にします。
6. SWDIO で、必要なアクティベーション コード シーケンス (0b0101_1000、MSB first) を送信します。
7. SWDIO を少なくとも 50 SWCLK サイクル HIGH にすることにより、ターゲットを既知の状態になります。これにより、SWD インターフェースがライン リセット状態になります。デバイスはこれにより SWD コンフィギュレーションになりました。PSOC™ Control C3 MCU は JTAG から休止状態へ、そして休止状態から SWD へ移動するために休止操作を使用します。

この操作の詳細については、Arm® Debug Interface Architecture Specification ADIv6.0 を参照してください。

12.5.2 SWD プログラム モードへの移行

SWD ポートの取得後、ホストは特定のタイムウィンドウにデバイス プログラミング モードに入る必要があります。これはテスト モード制御レジスタ (MODE レジスタ) 内の TEST_MODE ビット (ビット 31) を設定することにより行われます。デバイス プログラミング モードに入る前に、必ずデバッグ ポートを設定してください。

12.5.3 SWD プログラム ルーチンの実施

デバイスがプログラミング モードになると、外部プログラマは SWD パケット シーケンスの送信を開始し、フラッシュ消去、フラッシュ プログラム、チェックサム検証などのプログラミング動作を実行できます。

12.6 PSOC™ Control C3 MCU SWD デバッグ インターフェース

Cortex®-M33 DAP デバッグ機能は、侵襲性デバッグおよび非侵襲性デバッグの 2 種類に分けられています。侵襲性デバッグには、プログラムのホールトとステップ実行、ブレークポイントおよびデータ ウォッチポイントがあります。非侵襲性デバッグには、命令アドレス プロファイリング、そしてフラッシュ メモリ、SRAM および他のペリフェラル レジスタを含むデバイス メモリ アクセスがあります。

DAP は以下の 3 つの主なデバッグ サブシステムを備えています。

- デバッグ制御およびコンフィギュレーション レジスタ
- ブレークポイント ユニット (BPU) –ブレークポイント機能を提供します
- デバッグ ウォッチポイント (DWT) –ウォッチポイント機能を提供します

デバッグ アーキテクチャの詳細情報については Armv8-M アーキテクチャ リファレンス マニュアルを参照してください。

注: BPU は、Arm® Debug Interface Architecture Specification ADIv6.0 の Flashpoint and Breakpoint (FPB) ユニットの一部です。

12.6.1 デバッグ制御およびコンフィギュレーション レジスタ

デバッグ制御およびコンフィギュレーション レジスタは、ファームウェア デバッグを実行するために使用されます。レジスタとその主な機能は次のとおりです (これらのレジスタの完全なビットレベルの定義については、Armv8-M アーキテクチャのリファレンスマニュアルを参照してください)。

- **デバッグ ホールト制御およびステータス レジスタ (DHCSR):** このレジスタには、デバッグの有効化、CPU の停止、単ステップ操作を実行するための制御ビットがあります。また、プロセッサのデバッグ状態用のステータス ビットも含まれます。
- **デバッグ フォールト ステータス レジスタ (DFSR):** このレジスタには、デバッグイベントが発生する理由について説明します。CPU の停止、ブレークポイントイベント、ウォッチポイントイベントによって引き起こされるデバッグイベントについても含まれます。
- **デバッグ コア レジスタ データ レジスタ (DCRDR):** このレジスタは、書き込みまたは読み出しのデータを格納するために使用されます。
- **デバッグ例外およびモニター制御レジスタ (DEMCR):** このレジスタは、グローバル デバッグ ウォッチポイント (DWT) ブロック有効、リセット ベクタ キャッチ、ハードフォールト例外キャッチに対応したイネーブルビットを含みます。

12.6.2 ブレークポイントユニット (BPU)

BPU は命令のフェッチ段階でブレークポイント機能を提供します。PSOC™ Control C3 MCU の Cortex® -M33 は、4 つのハードウェア ブレークポイントに対応しています。ハードウェア ブレークポイントに加え、BKPT 命令を使用することで、任意の数のソフトウェア ブレークポイントを作成できます。BPU には以下の 2 種類のレジスタがあります。

- ブレークポイント制御レジスタ (FPB_CTRL) は BPU を有効にし、デバッグ システムでサポートされているハードウェア ブレークポイントの数を格納するために使用されます。
- 各ハードウェア ブレークポイントには、ブレークポイント比較レジスタ (FPB_COMPx) が 1 つずつあります。ブレークポイントのイネーブルビットとコンペアアドレス値が含まれます。一般的な使用例としては、命令フェッチアドレスがブレークポイントの比較アドレスと一致する場合、ブレークポイント イベントが生成され、プロセッサが一時停止します。

12.6.3 データ ウォッチポイント (DWT)

DWT はデータ アドレス アクセスまたはプログラム カウンター (PC) 命令アドレスでウォッチポイント サポートを提供しています。

DWT は、Cortex® -M33 向けに 2 つのウォッチポイントをサポートします。また PC サンプル レジスタを使用して外部プログラム カウンター サンプリングも提供しています。このサンプル レジスタは、プログラム カウンターの非侵襲性のおおまかなプロファイリングに使用されます。

- ウォッチポイント比較 (DWT_COMPx) レジスタは、ウォッチポイント コンパレーターがウォッチポイント イベントの生成に使用する比較値を格納します。各ウォッチポイントに対して関連する DWT_COMPx レジスタを持っています。
- ウォッチポイント機能 (DWT_FUNCTIONx) レジスタは、ウォッチポイント イベントのトリガー条件を格納します。イベントはプログラム カウンター ウォッチポイント イベントまたはデータ アドレス読み出し/書き込みアクセス ウォッチポイント イベントです。また関連するウォッチポイント イベントが発生するとステータス ビットを設定します。
- ウォッチポイント コンパレーター PC サンプル レジスタ (DWT_PCSR) は、プログラム カウンターの現在の値を格納します。このレジスタはプログラム カウンター レジスタの非侵襲性のおおまかなプロファイリングに使用されます。

12.6.4 PSOC™ Control C3 デバイスのデバッグ

ホストは、デバッグ コントロールおよびコンフィギュレーション レジスタ、BPU レジスタ、DWT レジスタにアクセスすることにより、ターゲットをデバッグします。すべてのレジスタは SWD インターフェースを介してアクセスされます。Cortex® -M33 では、DAP が SWD パケットを DAP-AHB インターフェースを通じて適切なレジスタアクセスに変換します。

デバイスをデバッグする最初の手順は SWD ポートを取得することです。開通シーケンスは、SWD ポートの開通セクションに記載されています。正しい M33 DAP SWDID がターゲット デバイスから読み出された時、SWD ポートが開通します。SWD インターフェースでデバッグを行う場合、対応する端子を他の目的に使用しないでください。I/O サブシステムセクションを参照し、SWD ポートピンの設定方法を確認してください。これにより、SWD ポートピンを SWD インターフェース専用に使用するか、LCD や GPIO などの他の機能に使用するかを設定できます。デバッグが必要な場合は、SWD ポートピンを他の目的に使用しないでください。

プログラム サポートのみが必要な場合は、SWD ピンを他の目的にも使用できます。SWD ポートが開通した時、外部デバッガはデバッグを有効にするために、DHCSR レジスタでの C_DEBUGEN ビットを設定します。そして、デバッグ システムに適切なレジスタを書き込むことで、ステップ実行、ホールト、ブレークポイント コンフィギュレーションおよびウォッチポイント コンフィギュレーション等といった異なるデバッグ動作を実行します。

12.7 レジスタの概要

表 89 デバッグ機能レジスタ

名称	説明
CPUSS_AP_CTL	デバッグ機能の設定

13 暗号化アクセラレータ

暗号アクセラレータ ブロックは、暗号関数のハードウェア実装とアクセラレーションを行います。ハードウェアでの実装は、同等のファームウェア実装よりも時間とエネルギーがかかりません。さらに、このブロックは真性乱数生成 (TRNG) 機能を提供します。PSOC™ Control C3 MCU の暗号アクセラレータ ブロック名は CryptoLite です。

13.1 特長

以下に暗号アクセラレータの特長を示します。

- Advanced Encryption Standard (AES)-128 ブロック暗号機能を高速化し、128 ビットの固定キーによる 128 ビットブロックの暗号化機能のみをサポートします。
- ハッシュ機能を高速化するセキュアハッシュアルゴリズム (SHA) -256
- 真性乱数生成器 (TRNG)
- RSA や楕円曲線暗号 (ECC) などの非対称鍵暗号をサポートするベクトル ユニット (VU)。

13.2 アーキテクチャ

下図は CryptoLite ブロックの概要です。

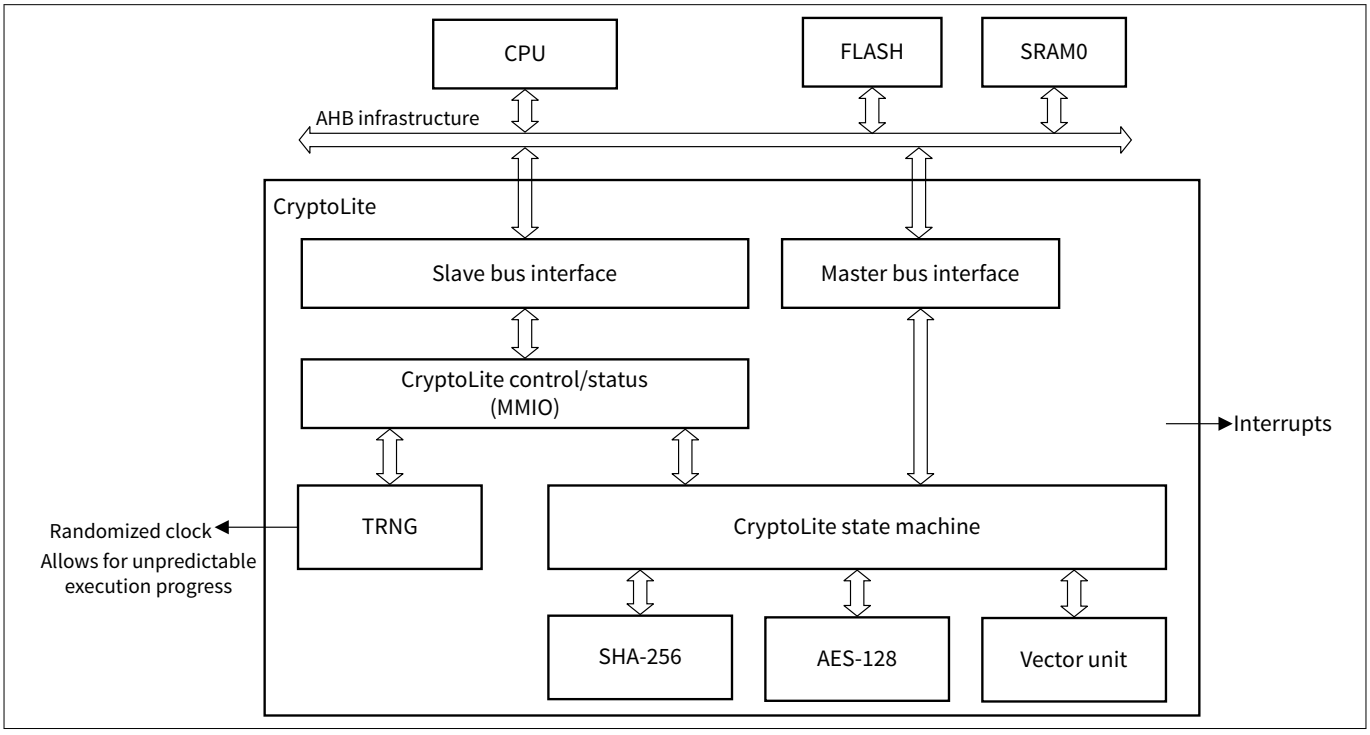


図 38 CryptoLite ブロック図

CryptoLite ブロックには以下のインターフェースがあります。

- AHB スレーブ インターフェース: 外部バス マスターは、このインターフェースを使用してブロック MMIO レジスタにアクセスします
- AHB マスター インターフェース: ブロックはこのインターフェースを使用して、メモリ (FLASH と SRAM) 内のハードウェア機能データ構造とオペランドにアクセスします。
- 2 つの割込み: 1 つは、通常動作用の割込み (例: TRNG 割込み)。もう1つは、異常/エラー動作用の割込み (例: AHB バス インターフェスのバス エラー検出)。
- ランダムクロック: TRNG デジタル化アナログ信号 (DAS) のビットストリームシーケンスです。
- クロックおよびリセット信号インターフェースは、システム リソース サブシステム (SRSS) に接続されます。

CryptoLite ブロックは、最小限の暗号関数のハードウェア アクセラレーションを行います。この機能は 4 つのコンポーネントによって実装されています。

- AES-128 コンポーネント: ブロック暗号の機能を高速化します。このコンポーネントは次の機能に対応します。
  - 128 ビット キーによる 128 ビット ブロックの順方向暗号化
- SHA-256 コンポーネント: ハッシュ機能を高速化します。このコンポーネントは次の機能に対応します。
  - 512 ビットのメッセージチャンクのメッセージスケジュール計算
  - 512 ビットのメッセージチャンクの処理
- ベクトル ユニット (VU) コンポーネント: 非対称鍵暗号 (例: RSA や ECC) を高速化します。このコンポーネントは次の機能に対応します。
  - 大きな整数の乗算、加算など
- TRNG コンポーネントリング発振器をベースにした TRNG には、ハードウェアヘルスモニターが搭載されています。このコンポーネントは、最大 6 つのリング発振器を使用して真性乱数を生成します。

このコンポーネントは、SHA-256 と VU ハードウェアアクセラレーションにより、セキュアブートのデジタル署名認証 (DSA または ECDSA) などを高速化します。デジタル署名は、メッセージの認証、完全性、否認防止を提供します。

13.2.1 ランダムクロック

CryptoLite は、インターフェース信号「*tr_trng_bitstream*」で TRNG デジタル化アナログ信号 (DAS) ビットストリームシーケンスを公開します。このシーケンスは、機能のタイミングをランダムにするために使用されます。この信号は、トリガー マルチプレクサを使用して GPIO や DMA にルーティングできます。詳細についてはトリガー マルチプレクサ (トリガー MUX) の章を参照してください。

tr_trng_bitstream 信号のデフォルト値は、発振器が有効でない場合、一定の‘0’に設定されます。

TRNG 機能は、繰り返し回数 (RC) または適応プロポーション (AP) のエラー検出時にモニターによって停止されることがあります。TRNG 機能が停止した場合、DAS ビットストリームとランダム化クロック信号の両方が一定の‘0’に設定されます。「停止機能」が有効でない場合、DAS ビットストリームとランダム化クロックはランダムなビットシーケンスを生成し続けます。詳細については真性乱数生成器 (TRNG)を参照してください。

13.2.2 ハードウェア/ソフトウェア インターフェース

SHA-256、AES-128、および VU コンポーネントは、同様のハードウェア/ソフトウェア インターフェースを持っています。以下のコンポーネントには、メモリ上のデータ構造へのポインタを提供する専用の MMIO レジスタがあります。

- SHA-256 コンポーネントは CRYPTOLITE_SHA_DESCR ポインタレジスタを使用します。
- AES-128 コンポーネントは CRYPTOLITE_AES_DESCR ポインタレジスタを使用します。
- VU コンポーネントは CRYPTOLITE_VU_DESCR ポインタレジスタを使用します。

これらの MMIO レジスタの 1 つへの書き込みは、ハードウェア機能を開始します。ブロックはメモリからデータ構造を読み出し、データ構造は目的のハードウェア関数とそのオペランドを指定します。データ構造はハードウェアの機能に特化しています。ブロックは指定されたハードウェア機能を実行します。最初から最後まで、ハードウェアの機能はビジー状態と考えられます。

- ハードウェア機能 (SHA-256、AES-128、VU) がビジー状態である間、CryptoLite ブロックは新しい MMIO レジスタ書き込み転送を受け入れられません。新しい MMIO レジスタの書き込み転送が待機状態になり、それ以外の (読み出し/書き込み) MMIO レジスタ転送はブロックされます。
- ハードウェア機能がビジー状態で、CryptoLite ブロックがブロックされていない間、ブロックは新しい MMIO レジスタ読み出し転送を受け入れられます。例えば、読み出し転送は CRYPTOLITE_STATUS レジスタを読み出すかもしれません。CRYPTOLITE_STATUS レジスタは、CryptoLite ブロックがビジー状態かどうかを示します。

上記の動作には、次のような意味があります。

- 1 つのハードウェア機能 (SHA-256、AES-128、VU) のみが、同時にビジー/アクティブになれます。これは、ハードウェアリソースを共有する実装として望ましいです。
- データ構造ポインタ MMIO レジスタは、ハードウェア機能がビジー状態の間はソフトウェアで上書きできません。
- ソフトウェアは、ハードウェア機能がビジー状態でない間に、次のデータ構造をメモリ上に準備し、そのポインタをデータ構造ポインタ MMIO レジスタに書き込めます。次のデータ構造のポインタの書き込み転送は暗号ブロックをブロックします。これにより、データ構造ポインタ MMIO レジスタが上書きされるのを防げます。

  ***注**:* *ソフトウェアは、メモリ上の現在の (ビジーな) データ構造と次のデータ構造が重ならないようにする必要があります。*

***注**:* *TRNG とハードウェア機能 (SHA-256、AES-128、VU) を同時にアクティブにできます。しかし、TRNG MMIO レジスタ書き込み転送は、ハードウェア機能がビジー状態の時に保留されます。したがって、ハードウェア機能を使用する前に、TRNG 機能をセットアップ/開始することを推奨します。TRNG MMIO レジスタ読み出し転送 (CRYPTOLITE_INTR_TRNG および CRYPTOLITE_TRNG_RESULT) は、ハードウェア機能がビジー状態のときに可能です。*

前述したように、メモリ上のデータ構造は、ハードウェア関数とそのオペランドを指定します。オペランドは通常、メモリ上の 8 ビットバイトまたは 32 ビットワード列へのポインタです。メモリ上のこれらのシーケンスは、秘密鍵 (AES-128)、メッセージチャンク (SHA-256)、または長い符号なし整数 (VU) を表します。図 39 に、記述子、データ構造、オペランドの高レベルブロック図を示します。

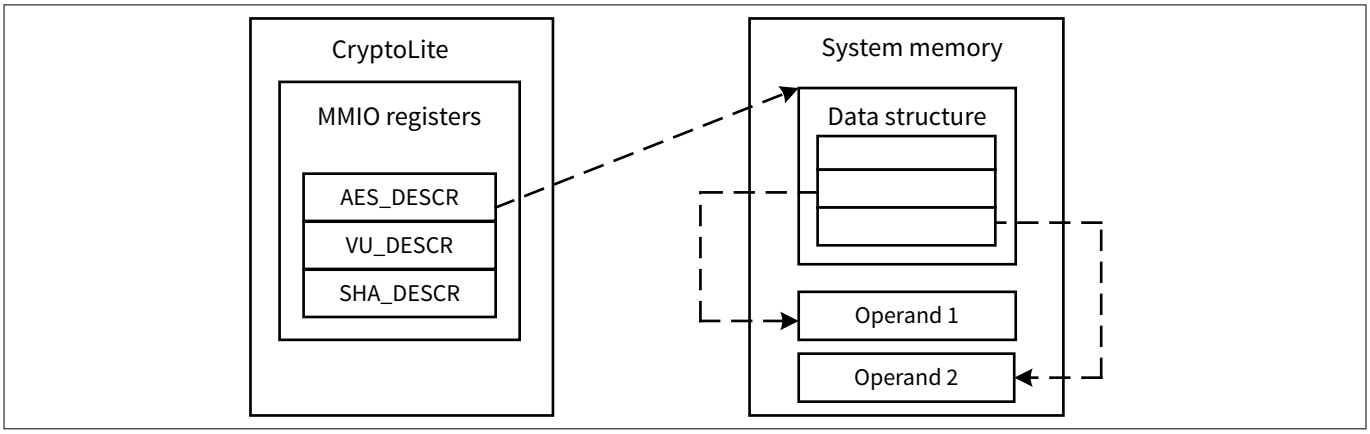


図 39　　CryptoLite ハードウェア/ソフトウェア インターフェース

ほとんどの暗号関数は、CPU ソフトウェアとハードウェア アクセラレーションを組み合わせて実装されています。

ハードウェア機能 (SHA-256、AES-128、VU) は、AHB マスター インターフェースを使用してメモリにアクセスします。バス エラーが報告され (CRYPTOLITE_INTR_ERROR.BUS_ERROR)、AHB バス エラーでハードウェア機能が終了します。

13.3　　ブロック機能

13.3.1　　AES-128

AES-128 ブロック暗号関数は、AES ブロック暗号ファミリーのメンバーです。128 ビットのブロックを 128 ビットの鍵で暗号化します。

ブロック暗号の動作モードは、AES-128 ブロック暗号機能を特定の設定で使用し、メッセージの機密性と真正性を提供します。CryptoLite コンポーネントは、AES-128 ブロック暗号機能のハードウェアアクセラレーションを提供

します。特定のブロック暗号の動作モードは、ソフトウェアとハードウェアアクセラレーションを使って実装されます。通常、メッセージは複数の 128 ビットブロックに分割されます。一般的な動作モードは以下のとおりです。

- 電子コードブック (ECB)
- 暗号ブロック連鎖 (CBC)
- 暗号フィードバック (CFB)
- 出力フィードバック (OFB)
- カウンター (CTR)

AES-128 ブロック暗号関数は 10 ラウンドを実行します。各ラウンドは内部の暗号状態を更新します。各ラウンドには専用の 128 ビットのラウンド鍵があり、これは提供された 128 ビットの鍵から導き出されます。

CryptoLite AES-128 ブロック暗号機能のハードウェアアクセラレーションは、単一のハードウェア機能で構成されます。

- **Encrypt**: 128 ビットのブロック P (平文) を 128 ビットの SK (秘密鍵) で暗号化し、128 ビットのブロック C (暗号文) にします。

CryptoLite ブロックの CRYPTOLITE_AES_DESCR MMIO レジスタへの書き込みは、ECB モードでの 128 ビット AES 暗号化を開始します。CRYPTOLITE_AES_DESCR レジスタは、メモリ上の 3 つの 32 ビットワード構造へのポインタを想定しています。ポインターは 32 ビットのワードアラインでなければいけません。CryptoLite コンポーネントは、ハードウェア機能を指定する 3 つの制御語を取得します。

- **Word 0**: 16 * 8 ビットの秘密鍵 SK へのポインタ。秘密鍵はハードウェア関数入力
- **Word 1**: 16*8 ビットブロック P へのポインタ。ブロックはハードウェア機能入力
- **Word 2**: 16*8 ビットブロック C へのポインタ。ブロックはハードウェア機能出力

次の図に、AES-128 のデータフローを示します。

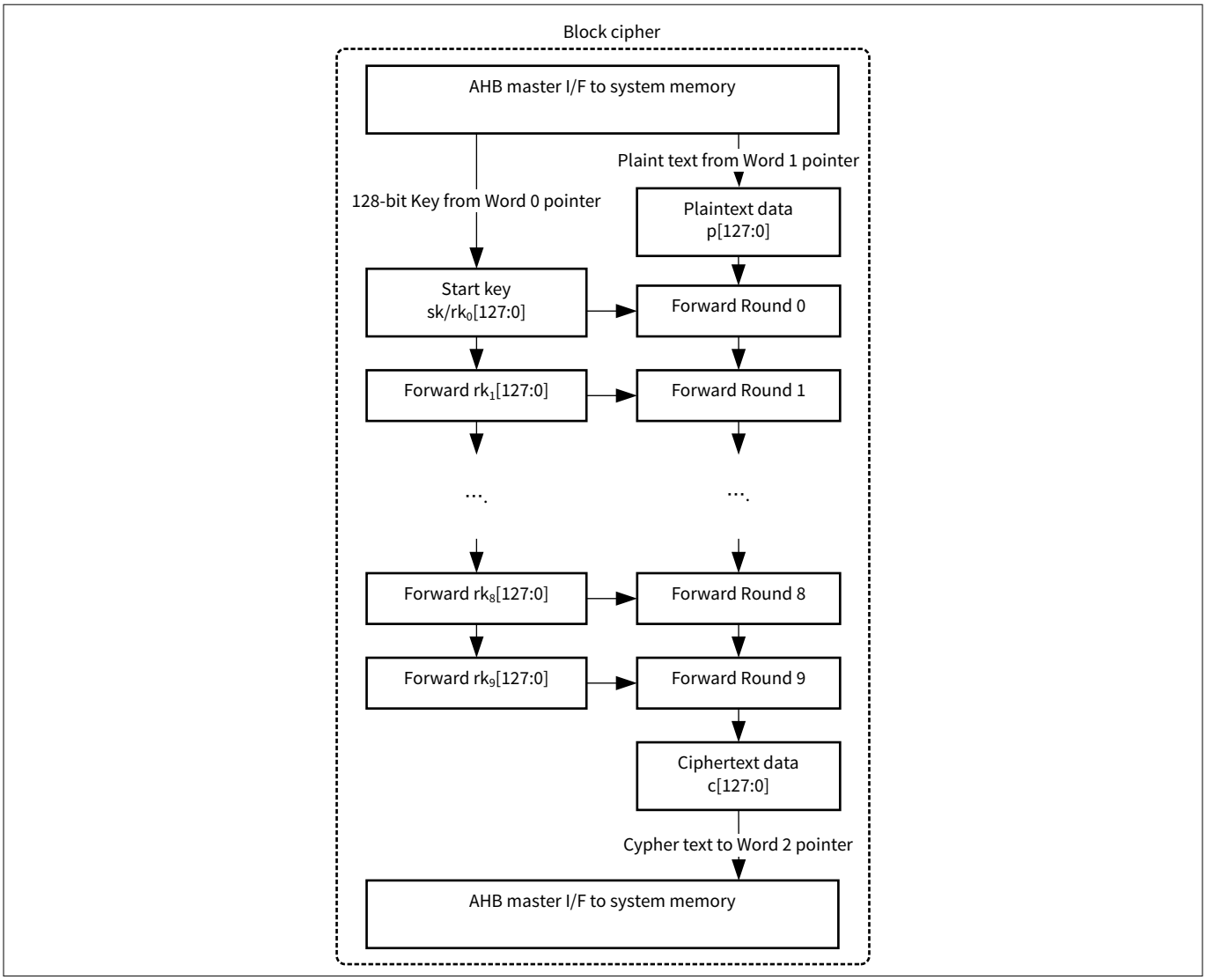


図 40　　AES-128 データフロー

注: AES-128 ブロック暗号機能は 8 ビットバイトで動作します。その結果、ハードウェアの観点から特別なエンディアンを考慮する必要はありません。

13.3.2　SHA-256

SHA-256 ハッシュ関数は SHA-2 ハッシュファミリーのメンバーです。メッセージのハッシュ値 (メッセージ ダイジェストとも呼ばれる) を計算するために、メッセージは複数の 512 ビットのメッセージ チャンクに分割されます。SHA-256 ハッシュ関数は一つのメッセージチャンクを処理します。

SHA-256 ハッシュ関数には、内部に 256 ビット (8 つの 32 ビットワード) のハッシュ状態 H[j]があります (j = 0, ..., 7)。最初のメッセージ チャンクで、SHA-256 であらかじめ定義された値で開始ハッシュ状態が初期化されます。連続するメッセージチャンクごとに、開始ハッシュ状態は前のメッセージチャンクの終了ハッシュ状態となります。最終メッセージチャンクの終了ハッシュ状態はメッセージのハッシュ値です。

SHA-256 ハッシュ関数は圧縮関数の処理を 64 回行います。圧縮関数は処理のたびに内部のハッシュ状態を更新します。各ラウンドには、専用の 32 ビットのラウンド定数 K[i]と専用の 32 ビットのメッセージスケジュール値 W[i]があります。

ラウンド定数は SHA-256 であらかじめ定義された値で、固定です。

メッセージ スケジュール値は 512 ビットのメッセージ チャンクから生成されます。

CryptoLite SHA-256 ハッシュ関数ハードウェアアクセラレーションは、2 つのハードウェア関数で構成されます。

- メッセージスケジュール機能: 512 ビットのメッセージ チャンク M から 64 * 32 ビットのメッセージ スケジュール値 W[i] (with = 0, ..., 63) を作成します。
- プロセス機能: SHA-256 は、圧縮関数を 64 ラウンド実行することにより、256 ビットのハッシュ状態を処理します。プロセスの入力は、256 ビットのハッシュ状態と、(メッセージ スケジュール関数によって作成された) メッセージ スケジュール W です。プロセスの出力は、更新された 256 ビットのハッシュ状態です。ラウンド定数 K は、明示的なプロセス入力ではなく、暗黙的なプロセス入力であることに注意してください (プロセス関数にハードワイヤーされています)。

1 つのメッセージチャンクを処理するために、1 つのメッセージスケジュールハードウェア関数が実行され、続いて 1 つのプロセスハードウェア関数が実行されます。

MMIO レジスタ CRYPTOLITE_SHA_DESCR への書き込みによりハードウェアの機能が開始されます。CRYPTOLITE_SHA_DESCR レジスタは、メモリ上の 3 つの 32 ビットワード構造へのポインタを想定しています。

***注**:* *ポインターは 32 ビットのワードアラインでなければいけません。*

CryptoLite コンポーネントは、ハードウェア機能を指定する 3 つの制御語を取得します。最初のワード (Word 0) は一般的なもので、他の 2 つのワード (Word 1 と 2) はハードウェア機能固有のものです。

- **Word 0** ビット位置 28 は、ハードウェア機能を指定するために使用されます。
  - WORD0[28]が'0'の場合、メッセージ スケジュール ハードウェア機能が実行されます。
  - WORD0[28]が'1'の場合、プロセスハードウェア機能が実行されます。

メッセージ スケジュール関数の場合、Word1 と Word2 は以下のように定義されます。

- **Word 1**: 16*32 ビット (512 ビット) のメッセージ チャンク M へのポインタ。メッセージ チャンクはハードウェア関数の入力です。ポインタは 32 ビットワード整列でなければいけません。
- **Word 2**: 64*32 ビット (2048 ビット) のメッセージ スケジュール配列 W へのポインタ。メッセージ スケジュールはハードウェア関数の出力です。ポインタは 32 ビットワード整列でなければいけません。

プロセス関数の場合、Word1 と Word2 は以下のように定義されます。

- **Word 1**: 8 * 32 ビット (256 ビット) ハッシュ状態 H へのポインタ。ハッシュ状態は、ハードウェア関数の入力で、ハードウェア関数の出力でもあります (入力は出力によって上書きされます)。ポインタは 32 ビットワード整列でなければいけません。
- **Word 2**: 64*32 ビット (2048 ビット) のメッセージ スケジュール配列 W へのポインタ。メッセージ スケジュールはハードウェア関数の入力です。ポインタは 32 ビットワード整列でなければいけません。

***注**:* *SHA-256 ハッシュ関数は 32 ビットワードで動作し、メモリ上のこれらのワードのビッグエンディアン メモリレイアウトを想定しています。PSOC™ Control C3 MCU はリトルエンディアン メモリ レイアウトを採用しています。これには次のような意味があります。*

- *メッセージ チャンクの 8 ビット バイトがメッセージ スケジュールの 32 ビット ワード W[i] を構築するために使われる時、リトルエンディアンからビッグ エンディアンへの変換 ("LE2BE") がハードウェアによって処理されます。*
- *最終的なハッシュ値の 32 ビットワード H[j] がメッセージダイジェストとして使用される場合 (メッセージダイジェストがバイトストリームとみなされると仮定)、ソフトウェアによってビッグエンディアンからリトルエンディアンへの変換 ("BE2LE") が必要となります。*

13.3.3 ベクトル ユニット (VU)

ベクトル ユニット (VU) は、RSA や楕円曲線暗号 (ECC) などの非対称鍵暗号に有用な長整数演算に対応します。

演算には、1 つのデスティネーション オペランド (DST) と、1 つまたは 2 つのソース オペランド (SRC0 と SRC1) があります。各オペランドのビットサイズは決まっています。このビット サイズは 32 ビットのワード倍数で指定されます。x の値は、(x+1)*32 ビットのビットサイズを指定します。

VU は以下の長整数演算に対応しています。

- MUL: この演算は通常の長整数乗算演算です。
  - DST[dst_bit_size-1:0]=SRC0[src0_bit_size-1:0] * SRC1[src1_bit_size-1:0]。
- XMUL: この演算は、キャリーレスの長整数乗算演算です。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] xmul SRC1[src1_bit_size-1:0]
- ADD: この演算は長整数加算演算です。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] + SRC1[src1_bit_size-1:0]
- SUB: この演算は長整数の減算演算です。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] - SRC1[src1_bit_size-1:0]
- XOR: 演算。この演算は長整数の排他的論理和演算です。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] ^ SRC1[src1_bit_size-1:0]
- MOV: この操作は長整数を移動/コピーします。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0]
- LSR1: この操作は、長整数を 1 ビット右にシフトします。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] >> 1
- LSL1: 長整数を 1 ビット左にシフトします。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] << 1
- LSR: この操作は、長整数を特定のビット位置のみ右にシフトします。
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] >> "ビットシフト量"
- COND_SUB: この操作は条件付き長整数の減算です。
  - DST[dst_bit_size-1:0] = (SRC0[src0_bit_size-1:0] ≧ SRC1[src1_bit_size-1:0]) ? (SRC0[src0_bit_size-1:0] - SRC1[src1_bit_size-1:0]) : SRC0[src0_bit_size-1:0]

  ***注**:* *演算の「Greater than Equal」演算子はソース ビット サイズ (デスティネーション ビット サイズではない) を使用し、演算の「減算」演算子はデスティネーション ビット サイズを使用します。*

CRYPTOLITE_VU_DESCR レジスタへの書き込むことで VU 動作が開始されます。CRYPTOLITE_VU_DESCR レジスタは、メモリ上の 4 つの 32 ビットワード構造へのポインタを想定しています。ポインターは 32 ビットのワードアラインでなければいけません。CryptoLite コンポーネントは、ハードウェア機能を指定する 4 つの制御語を取得します。最初のワード (Word 0) は一般的なもので、他の 3 つのワード (Word 1、2、3) はハードウェア機能固有のものです。

- **Word 0**:
  - ビット位置 31:28 で動作を指定

表 90　　ベクトルユニットのオペコード

オペコード	動作
0_H	乗算 (MUL)
1_H	加算 (ADD)
2_H	減算 (SUB)
3_H	排他的論理和 (XOR)
4_H	バイナリ乗算 (XMUL)
5_H	論理右シフト 1 (LSR1)
6_H	論理左シフト 1 (LSL1)

(続く)

表 90 (続き) ベクトルユニットのオペコード

オペコード	動作
7_H	論理右シフト (LSR)
8_H	条件付き減算 (COND_SUB)
9_H	転送 (MOV)
A_H～F_H	予約済み

- ビット位置 27:25 は予約済み
- ビット位置 24:16 は、デスティネーション オペランドのワード長 (マイナス 1) を指定します。9 ビットのワード長により、最大 512 個の 32 ビット ワード (16384 ビット) のデスティネーション オペランドが可能
- ビット位置 15:8 は、ソース オペランド 1 (存在する場合) のワード長 (マイナス 1) を指定します。
- ビット位置 7: 0 は、ソース オペランド 0 (存在する場合) のワード長 (マイナス 1) を指定します。8 ビットのワード長により、最大 256 個の 32 ビット ワード (8192 ビット) のソース オペランドが可能

注: *予約済みオペコード (>9) の場合、ハードウェア状態に変化はなく、数サイクルのみ STATUS.BUSY がハイになることに注意してください。これは、CryptoLite ハードウェアが MMIO 書き込みによって開始された時点で VU オペコードが未定義かどうかわからないので、CryptoLite は記述子の位置からオペコードをフェッチしてアイドルになるまでビジー状態になるためです (未定義のオペコードが検出されると、STATUS.BUSY は '0' にクリアされます)。*

MUL、XMUL、ADD、SUB、XOR、COND_SUB 演算では、Word1、Word2、Word3 は次のように定義されます。

- **Word 1**: ソース オペランド 0 の Word 0 (最下位ワード) へのポインタ。ポインタは 32 ビットワード整列でなければいけません。
- **Word 2**: ソース オペランド 1 の Word 0 (最下位ワード) へのポインタ。ポインタは 32 ビットワード整列でなければいけません。
- **Word 3**: 出力オペランドの Word 0 (最下位ワード) へのポインタ。ポインタは 32 ビットワード整列でなければいけません。

LSR1、LSL1、および MOV 演算では、Word1、Word2、および Word3 は以下のように定義されます。

- **Word 1**: ソース オペランドの Word 0 (最下位ワード) へのポインタ。ポインタは 32 ビットワード整列でなければいけません。
- **Word 2**: 未使用
- **Word 3**: 出力オペランドの Word 0 (最下位ワード) へのポインタ。ポインタは 32 ビットワード整列でなければいけません。

LSR オペレーションでは、Word1、Word2、Word3 は次のように定義されます。

- **Word 1**: ソース オペランドの Word 0 (最下位ワード) へのポインタ。ポインタは 32 ビットワード整列でなければいけません。
- **Word 2**: シフト量。ビット位置 12:0 でシフト量を指定します
- **Word 3**: 出力オペランドの Word 0 (最下位ワード) へのポインタ。ポインタは 32 ビットワード整列でなければいけません。

オペランド サイズに関する注意:

- ソース オペランドとデスティネーション オペランドには専用のサイズ指定があります。デスティネーション オペランドはソース オペランドより大きいことがあります。通常、これは乗算操作に便利です。
- デスティネーション オペランドのサイズが演算結果のサイズより小さい場合、演算結果の上位ビットは無視されます。宛先オペランドサイズが演算結果サイズより大きい場合、演算結果'0'が拡張されます。

通常、ソース オペランドとデスティネーション オペランドは異なるメモリ位置を占めます。しかし、メモリ割り当てを制限するためには、次のような方法が有効なことがあります。

- 2 つのソース オペランドに同じメモリ位置を使用します。これは何の制限もなく行えます。例えば、ADD 演算の 2 つのソース オペランドは同じ値で、ソース オペランドは 2 倍になります。
- ソース オペランドとデスティネーション オペランドに同じメモリ位置を使用します。これには動作に依存する制約があります。具体的には、以下の操作により、ソース オペランドとデスティネーション オペランドに同じメモリ ロケーションを使用できます。
  - ADD、SUB、XOR、MOV、LSR1、LSL1 演算

  以下の操作では、ソース オペランドとデスティネーション オペランドに同じメモリ ロケーションの使用はできません。
  - MUL、XMUL、LSR、COND_SUB 演算

***注**:* *VU は 32 ビットのワード倍数で動作し、メモリ内のこれらのワードのリトルエンディアン メモリレイアウトを想定しています。PSOC™ Control C3 MCU はリトルエンディアン メモリレイアウトを採用しています。そのため、エンディアン変換は必要ありません。*

13.3.4 真性乱数生成器 (TRNG)

真性乱数生成コンポーネント (TRNG) は、真性乱数を生成します。生成される数値のビットサイズは 32 ビットに固定されています。

TRNG は、物理的なノイズ源を提供するために最大 6 つのリング発振器に依存しています。リング発振器は、リングを形成するためにフィードバックループで接続された一連のインバーターで構成されます。インバーター遅延の温度感受性のため、リングの発振信号にジッターが発生します。ジッター発振された信号はサンプリングされ、「デジタル化されたアナログ信号」(DAS) を生成します。これはすべてのマルチリング発振器で行われます。

エントロピーを高め、DAS ビットの偏りを減らすために、DAS ビットはさらに後処理されます。後処理には 2 つのステップがあります。

- エントロピーを増加させるためのオプションのリダクション ステップ (最大 6 つのリング発振器 DAS ビットと 1 つまたは複数の DAS ビット期間にわたり)。
- ‘0’または‘1’のバイアスを減らすためのオプション「フォン ノイマン補正」ステップ。

この補正ステップでは、前のステップで生成されたリダクションビットのペアを処理します。2 つの縮小ビット r0 と r1 (r0 は r1 の前に生成) が与えられた場合、補正ステップは以下のように定義されます。

{r0, r1} = {0, 0}: ビットは生成されません

{r0, r1} = {0, 1}: ‘0’ビットが生成される (ビット r0)

{r0, r1} = {1, 0}: ‘1’ビットが生成される (ビット r0)

{r0, r1} = {1, 1}: ビットは生成されません

つまり、補正ステップでは、‘0’から‘1’、または‘1’から‘0’への遷移時にのみビットが生成されます。ランダムな入力ビット列の場合、補正ステップは入力ビット列の 4 分の 1 に近い頻度の出力ビット列を生成することに注意してください (入力削減ビットは重複しないペアで処理され、ペアエンコーディングの半分のみが出力ビットになります)。

後処理により、真のランダムビットサンプルとみなされるビットサンプルが生成されます。真のランダム ビット サンプルをレジスタにシフトし、最大 32 ビットのランダム値を提供します。

高いスイッチング アクティビティの結果、リング発振器はかなりの電力を消費します。そのため、TRNG 機能が無効化されると、リングを「破壊」しスイッチングを防ぎます。TRNG 機能が有効な場合、リング発振器は当初予測可能な動作をします。しかし、時間の経過とともに、限りなく小さな環境 (温度) の変化が、この予測可能な挙動からの逸脱を増大させます。

- 初期遅延の間、リング発振器は信頼できる物理ノイズ源ではありません。
- 初期遅延の後、同じリング発振器が異なる発振挙動を示し、信頼性の高い物理ノイズ源を発生させます。

したがって、DAS ビットは初期化期間 (CRYPTOLITE_TRNG_CTL0.INIT_DELAY) 中、ドロップされる可能性があります。

以下の図に、TRNG コンポーネントの概要を示します。

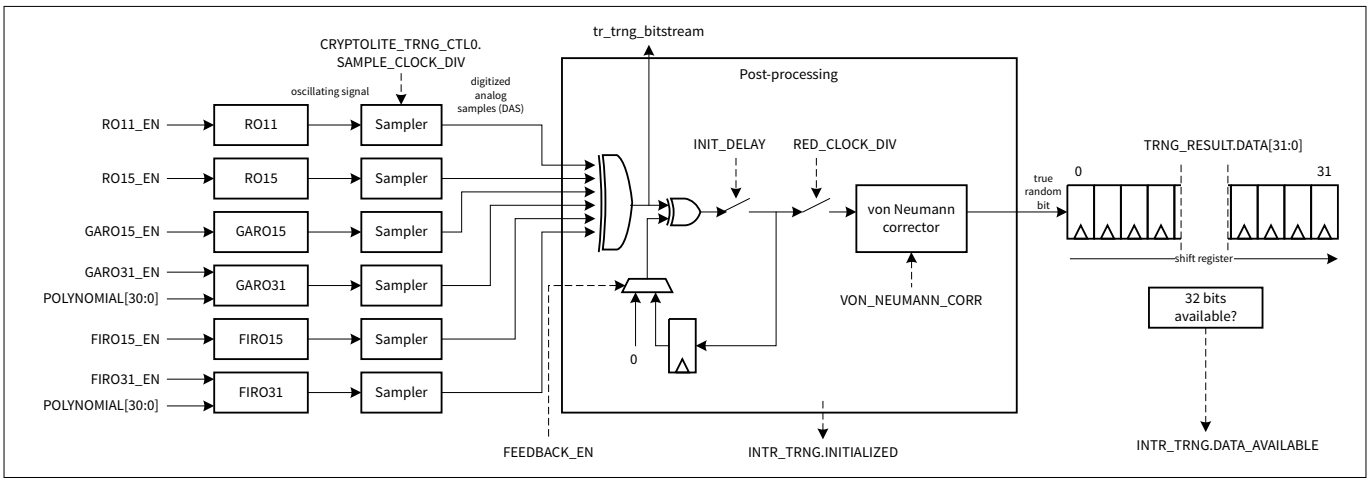


図 41　　TRNG の概要

CRYPTOLITE_TRNG_CTL0 レジスタの FEEDBACK_EN ビットフィールドはフィードバックを制御します。'0'の場合、フィードバックはありません。フィードバックが有効になっていると、「フォン ノイマン」コレクターの効果が低下する可能性があることが判明したため、この機能が存在します。

「サンプラー」ロジックは、発振信号をデジタル化します。以下の図に、機能を示します。

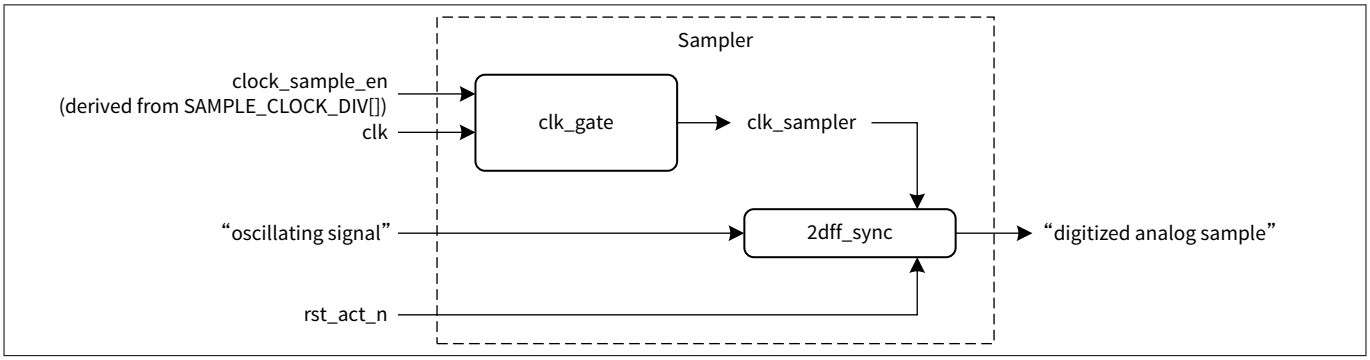


図 42　　サンプラー ロジック

***注**:　　リング発振器が無効化されると、同期ロジックはリセットされます。したがって、リング発振器は、後処理のリダクションステップに定数'0'を寄与します。*

前述のとおり、TRNG は最大 6 つのリング発振器に依存しています。

- RO11: 11 のインバーターで構成される固定リング発振器
- RO15: 15 のインバーターで構成される固定リング発振器
- GARO15: 15 のインバーターで構成される固定ガロア ベースのリング発振器
- GARO31: 最大 31 のインバーターからなる柔軟なガロア ベースのリング発振器。最大 31 次までのプログラム可能な多項式により、発振器のフィードバックに柔軟性がもたらされます。
- FIRO15: 15 のインバーターからなる固定フィボナッチ ベースのリング発振器
- FIRO31: 最大 31 のインバーターからなる、柔軟なフィボナッチ ベースのリング発信器。最大 31 次までのプログラム可能な多項式により、発振器のフィードバックに柔軟性がもたらされます。

各リング発振器は有効または無効にできます。無効化すると、リングは「壊れて」スイッチングを防ぎます。
XXX_EN イネーブル信号は CRYPTOLITE_TRNG_CTL1 レジスタに由来します。

13.3.4.1 固定リング発振器: RO11, RO15, GARO15, FIRO15

次の図は、固定リング発振器の回路図です。

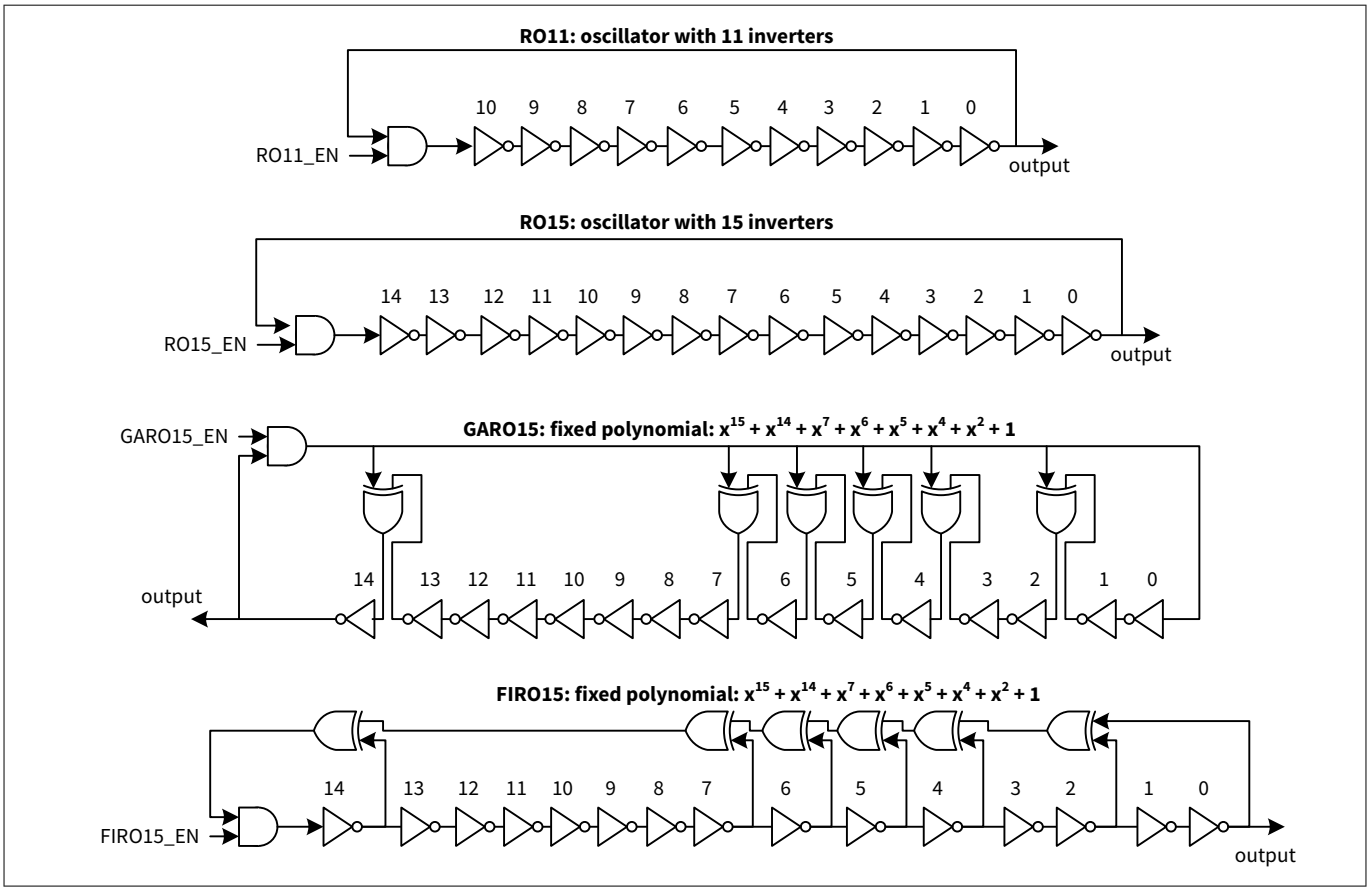


図 43 4 つの固定リング発振器: RO11, RO15, GARO15, および FIRO15

13.3.4.2 リング発振器: GARO31

柔軟なガロア ベースのリング発振器は、発振器のフィードバックを指定するためにプログラム可能な多項式に依存しています。これにより、1、3、5、...、31 のインバーターのリングが可能になります (発振信号を生成するためには奇数が必要)。下図はガロア ベースのリング発振器の概要です。

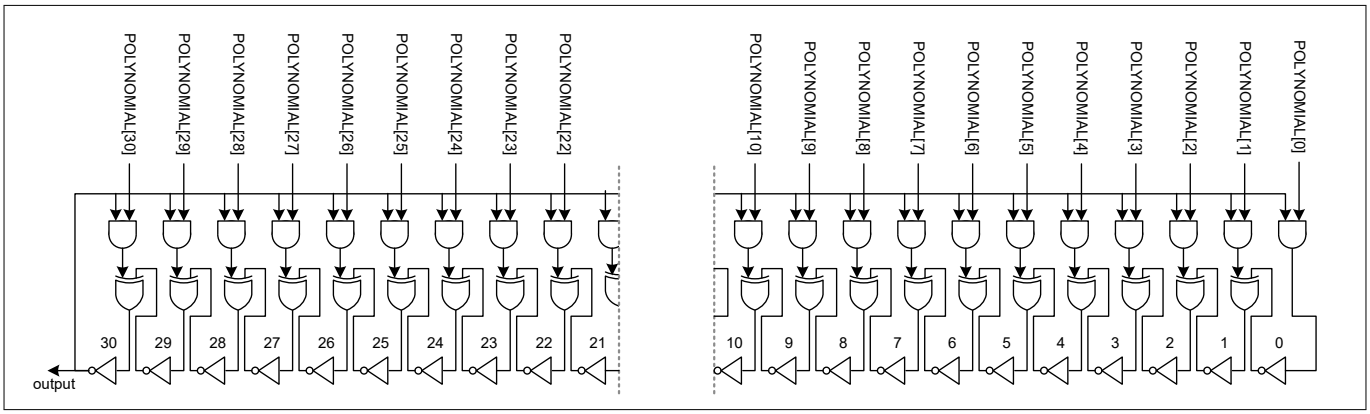


図 44 柔軟なガロア ベースのリング発振器: GARO31

リング発振器が無効になると (CRYPTOLITE_TRNG_CTL1 レジスタの GARO31_EN ビットが‘0’)、多項式は強制的に“0”になり、下図に示すようにリングは切断されます。

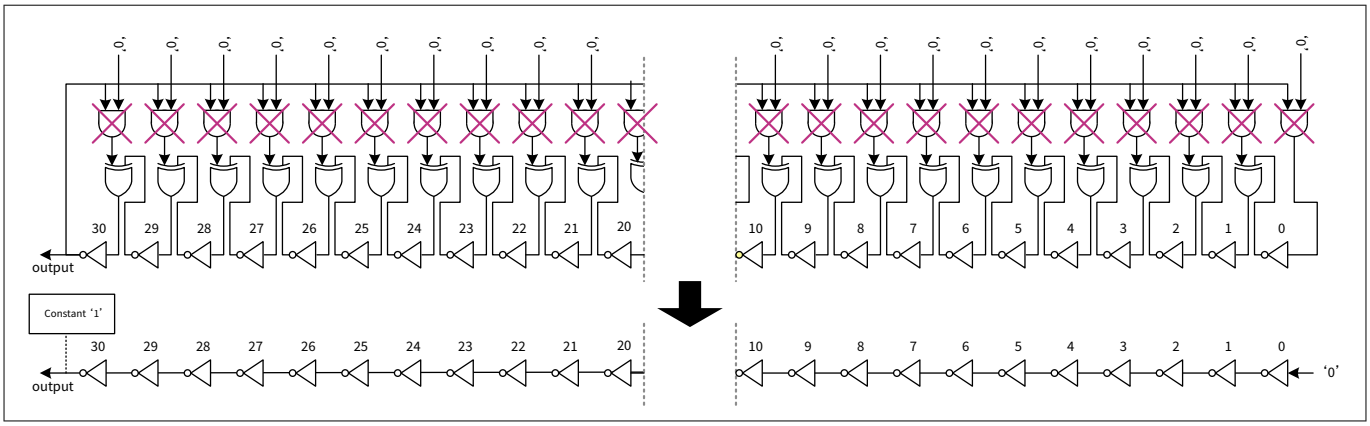


図 45　　GARO31 停止

プログラム可能な多項式は、発振器のフィードバックを指定します。次の図に、2 つの例を示します。

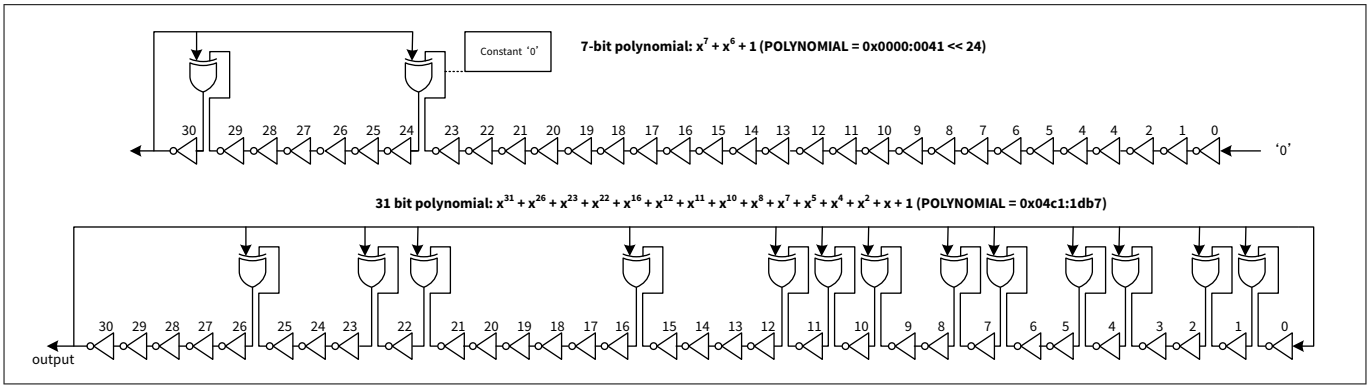


図 46　　GARO31 の例

13.3.4.3　　リング発振器: FIRO31

フレキシブルなフィボナッチ ベースのリング発振器は、発振器のフィードバックを指定するためにプログラム可能な多項式に依存しています。これにより、1、3、5、...、31 のインバーターのリングが可能になります (発振信号を生成するためには奇数が必要)。下図は、フィボナッチ数列を用いたリング発振器の概要です。

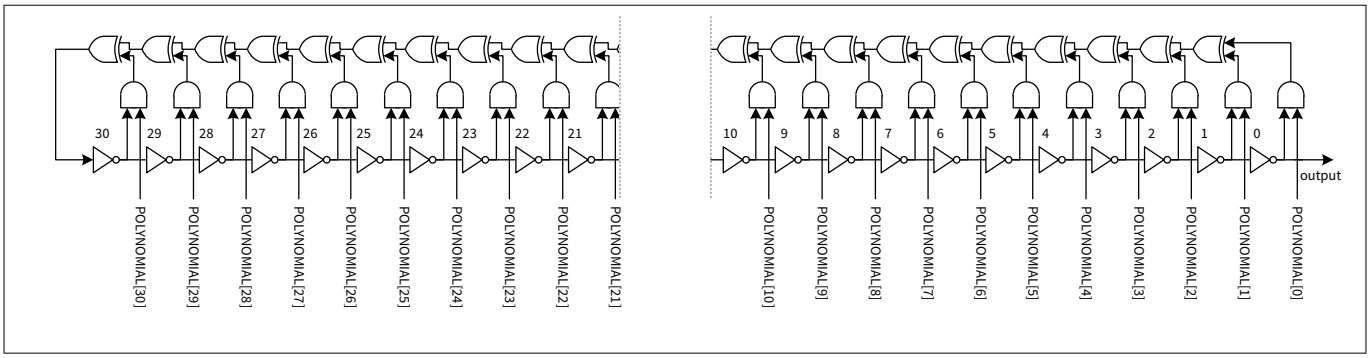


図 47　　柔軟なフィボナッチベースのリング発振器: FIRO31

リング発振器が無効になると (CRYPTOLITE_TRNG_CTL1 レジスタの FIRO31_EN ビットが'0')、多項式は強制的に'0'になり、下図に示すようにリングは切断されます。

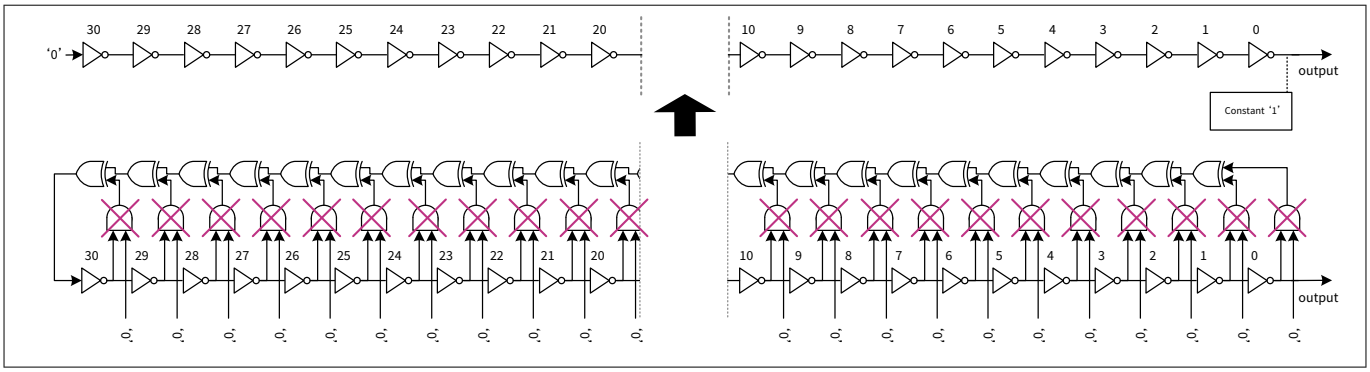


図 48　　FIRO31 停止

プログラム可能な多項式は、発振器のフィードバックを指定します。次の図に、2 つの例を示します。

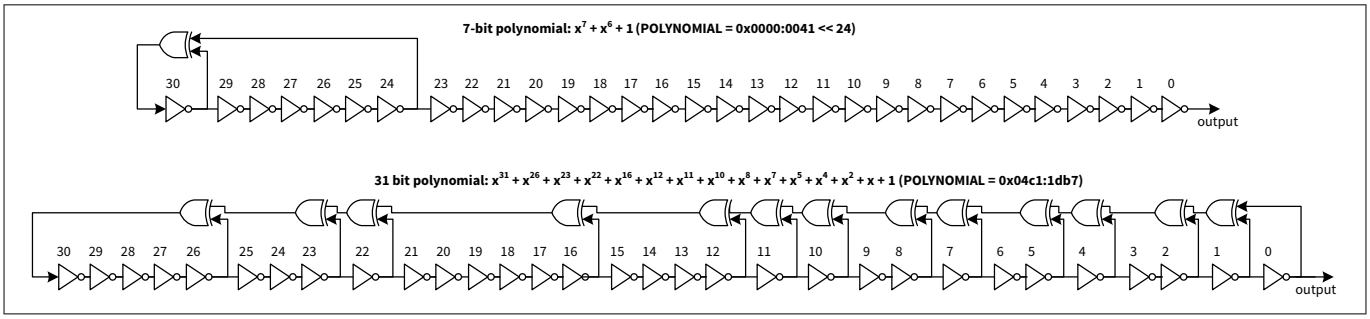


図 49　　FIRO31 の例

13.3.4.4　　TRNG 結果

32 ビットの TRNG 値は CRYPTOLITE_TRNG_RESULT レジスタからの読み出しで提供されます。この読み出しはブロッキング読み出しで、32 ビットが利用できない場合、読み出しはブロックされます。TRNG が有効でない場合、CRYPTOLITE_TRNG_RESULT レジスタからの読み出しは“0”を返します (CRYPTOLITE_TRNG_CTL1 レジスタの全ての XXX_EN フィールドは‘0’)。

13.3.4.5　　TRNG ヘルス モニター

TRNG にはヘルスモニターが内蔵されており、デジタル化されたノイズ源のテストを行い、意図した動作からの逸脱を検出します。例えば、ヘルスモニターは、デジタル化されたアナログサンプルの「縮退フォールト (stuck at)」を検出します。次の図に、ヘルスモニターの機能を示します。

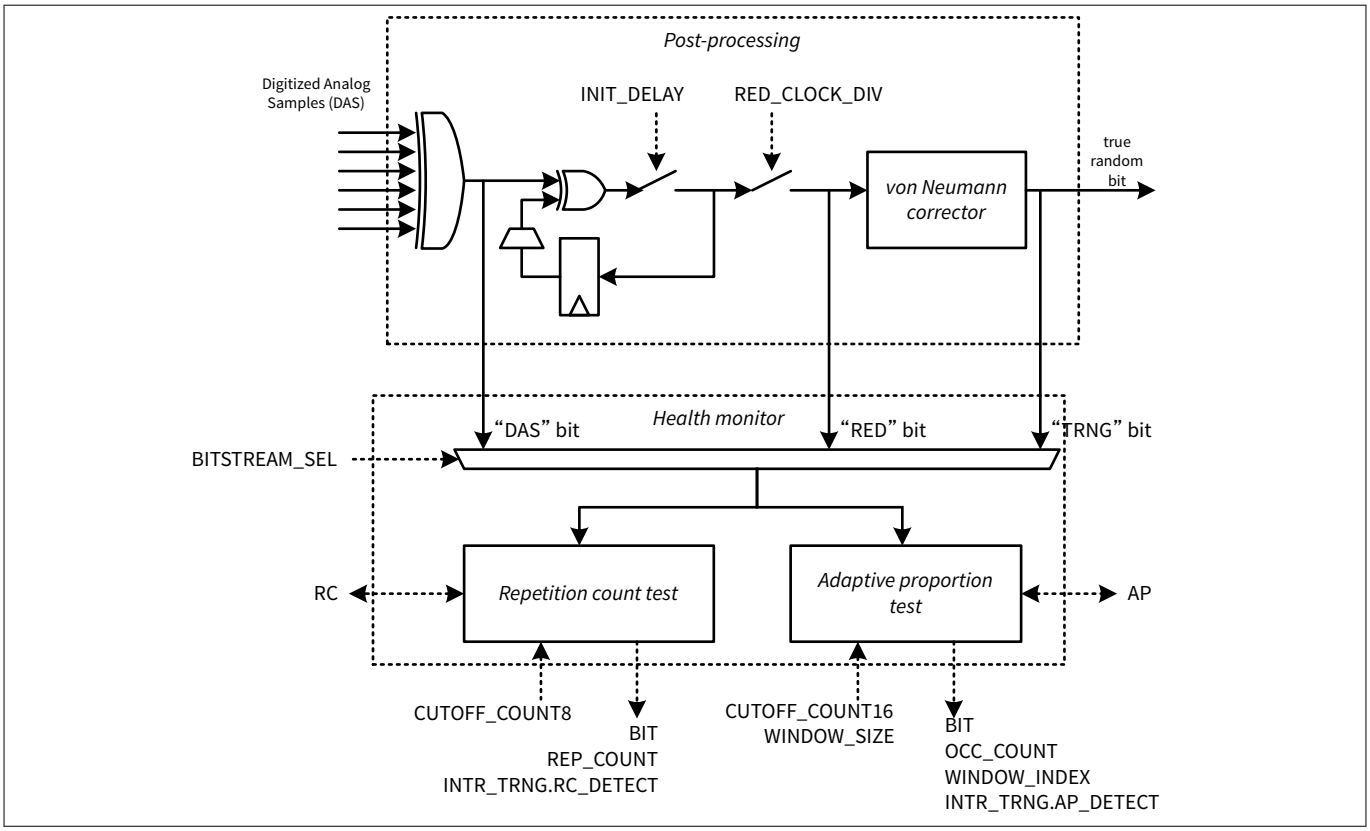


図 50　　TRNG ヘルス モニター

ヘルスモニターは、3 つのデジタル化されたビットストリームのうち 1 つをテストします。

- DAS ビットストリーム: これは、デジタル化されたアナログ サンプルの XOR です。
- RED ビットストリーム: これがリダクションビットのビットストリームです。各リダクション ビットは、複数の DAS ビットにわたって計算される可能性があることに注意してください。
- TR ビットストリーム: これが真のランダムビットのビットストリームです (「フォン ノイマン分解」ステップの後)。

ヘルスモニターは 2 つの異なるテストを行います。

- **繰返し回数試験**

  この試験は、ビットストリーム中に同じビット値 (‘0’または‘1’) を繰り返しチェックします。検出は、特定のアクティブビット値 (ステータスフィールド BIT で指定) が事前にプログラムされたビット数 (CRYPTOLITE_RNG_MON_RC_CTL レジスタの制御フィールド CUTOFF_COUNT8[7:0]で指定) 繰り返されたことを示します。このテストでは、アクティブ ビット値の繰り返し回数を保持するためにカウンターを使用します (CRYPTOLITE_TRNG_MON_RC_STATUS1 レジスタのステータス フィールド REP_COUNT[7:0]で指定)。

  テストが開始され (CRYPTOLITE_TRNG_MON_CTL レジスタの RC ビットフィールドで指定)、ビットストリームの値の変化が観測された場合、アクティブビット値 BIT は新しいビット値に設定され、繰り返しカウンター REP_COUNT[] は"1"に設定されます。ビットストリームの値に変化がない場合、繰り返しカウンター REP_COUNT[]は"1"インクリメントされます。

  検出により繰り返し回数テストを停止し (ハードウェアは CRYPTOLITE_TRNG_MON_CTL レジスタの RC ビットを‘0’に設定)、関連する割込みステータス フィールドを‘1’に設定し、ハードウェアがステータス フィールドを変更しないようにします。テストが停止すると、REP_COUNT[]は CUTOFF_COUNT8[]と等しくなります。

  CRYPTOLITE_TRNG_CTL0 レジスタの STOP_ON_RC_DETECT ビットが‘1’に設定されている場合、検出により TRNG 機能が停止します (ハードウェアにより CRYPTOLITE_TRNG_CTL1 レジスタのすべての XXX_EN フィールドが‘0’に設定)。

- **適応プロポーションテスト**

このテストは、ビットストリームにおいて特定のビット値 (‘0’または‘1’) が不均衡に出現しているかどうかをチェックします。検出は、特定のビット ウィンドウ サイズ (CRYPTOLITE_TRNG_MON_AP_CTL レジスタの制御フィールド WINDOW_SIZE[31:16]で指定) のビット シーケンスにおいて、特定のアクティブ ビット値 (ステータス フィールド BIT で指定) が事前にプログラムされた回数 (CRYPTOLITE_TRNG_MON_AP_CTL レジスタの制御フィールド CUTOFF_COUNT16[15:0]で指定) 発生したことを示します。このテストは、現在のウィンドウのインデックスを保持するカウンター (CRYPTOLITE_TRNG_MON_AP_STATUS1 レジスタの WINDOW_INDEX[15:0]で指定) と、アクティブ ビット値の発生回数を保持するカウンター (CRYPTOLITE_TRNG_MON_AP_STATUS1 レジスタのステータス フィールド OCC_COUNT[31:16]で指定) を使用します。

テストが開始されると (CRYPTOLITE_TRNG_MON_CTL レジスタの AP ビットによって指定)、ビットストリームは特定のウィンドウサイズのビットシーケンスに分割されます。ビット列の最初のビットでは、アクティブ ビット値 BIT が最初のビット値に設定され、カウンター WINDOW_INDEX が「0」に設定され、カウンター OCC_COUNT が“1”に設定されます。ビット列の他のすべてのビットに対して、カウンター WINDOW_INDEX は“1”インクリメントされます。新しいビット値がアクティブ ビット値 BIT と等しい場合、カウンター OCC_COUNT[]は“1”インクリメントされます。アクティブ ビット値 BIT は、ビット列の最初のビットにのみ設定されることに注意してください。

検出により適応プロポーションテストを停止し (ハードウェアは CRYPTOLITE_TRNG_MON_CTL レジスタの AP ビットを‘0’に設定)、関連する割込みステータス フィールドを‘1’に設定し、ハードウェアがステータス フィールドを変更しないようにします。テストが停止されると、OCC_COUNT[]は CUTOFF_COUNT16[]と等しくなり、WINDOW_INDEX は検出が発生したビット列インデックスを識別します。

CRYPTOLITE_TRNG_CTL0 レジスタの STOP_ON_AP_DETECT ビットが‘1’に設定されている場合、検出により TRNG 機能が停止します (ハードウェアにより CRYPTOLITE_TRNG_CTL1 レジスタのすべての XXX_EN フィールドが‘0’に設定)。

13.4 割込み

CryptoLite は 2 つの割込みを発生させます。

- TRNG 割込み
- エラー割込み

13.4.1 TRNG 割込み

この割込みは TRNG の動作を報告するために使用され、以下のシナリオでこの割込みが発生します。

- TRNG の初期化
- TRNG データは CRYPTOLITE_TRNG_RESULT レジスタで利用可能
- TRNG モニターが「繰り返し回数」エラーを検出
- TRNG モニターが「適応プロポーション」エラーを検出

TRNG 割込みには、CRYPTOLITE_INTR_TRNG_MASK レジスタに割込みマスクが設定されています。割込みマスクを Low にすることで、対応する割込みソースが無視されます。TRNG 割込みは、割込みマスクビットが High で、対応する割込みソースが保留中の場合に発生します。

TRNG 割込みをサービスするとき、割込みサービス ルーチン (ISR) は CRYPTOLITE_INTR_TRNG レジスタの対応する割込みビットに‘1’を書き込むことで、割込みソースをクリアできます。

CRYPTOLITE_INTR_TRNG_MASKED レジスタは、割込みソースと割込みマスクの論理積です。このレジスタは、ファームウェアが割込み要因を特定するために便利な方法を提供します。

検証やデバッグのために、設定されたビット (CRYPTOLITE_INTR_TRNG_SET レジスタの DATA_AVAILABLE など) が各割込みのトリガーとして使用されます。これによりファームウェアは実際にイベントが発生しなくても割込みを生成できます。

13.4.2 エラー割込み

この割込みは、AHB マスター インターフェース上の AHB バス エラーに対して生成されます。

エラー割込みは、CRYPTOLITE_INTER_ERROR_MASK レジスタの BUS_ERROR ビットを設定することで有効になり、CRYPTOLITE_INTR_ERROR_MASKED レジスタを読み出すことで、CPU 割込みコントローラーに転送される割込みマスク状態を知ることができます。

エラー割込みをサービスするとき、割込みサービス ルーチン (ISR) は CRYPTOLITE_INTR_ERROR レジスタの BUS_ERROR ビットに"1"を書き込むことによって割込みをクリアできます。

CRYPTOLITE_INTR_ERROR_SET レジスタの BUS_ERROR ビットは、ファームウェア デバッグ用の割込みをアサートするために使用できます。

注: 暗号ブロックはエラーを検出すると、AES、SHA、または VU の機能を終了します。エラーはスティッキー (粘着性) です。そのためソフトウエアは、個々の操作の後にエラーをチェックするのではなく、一連の操作の後にエラーをチェックできます。

13.5 レジスタの概要

表 91 CryptoLite レジスタの概要

レジスタ	機能
CRYPTOLITE_CTL	CryptoLite 制御レジスタ
CRYPTOLITE_STATUS	CryptoLite ステータス レジスタ
CRYPTOLITE_AES_DESCR	AES 記述子ポインタレジスタ
CRYPTOLITE_VU_DESCR	VU 記述子ポインタレジスタ
CRYPTOLITE_SHA_DESCR	SHA 記述子ポインタレジスタ
CRYPTOLITE_INTR_ERROR	CryptoLite エラー割込みレジスタ
CRYPTOLITE_INTR_ERROR_SET	CryptoLite エラー割込みセット レジスタ
CRYPTOLITE_INTR_ERROR_MASK	CryptoLite エラー割込みマスク レジスタ
CRYPTOLITE_INTR_ERROR_MASKED	CryptoLite マスク エラー 割込み出力レジスタ
CRYPTOLITE_TRNG_CTL0	TRNG 制御 0 レジスタ
CRYPTOLITE_TRNG_CTL1	TRNG 制御 1 レジスタ
CRYPTOLITE_TRNG_STATUS	TRNG ステータスレジスタ
CRYPTOLITE_TRNG_RESULT	TRNG 結果レジスタ
CRYPTOLITE_TRNG_GARO_CTL	TRNG ガロア ベース リング発振器 (GARO) 制御レジスタ
CRYPTOLITE_TRNG_FIRO_CTL	TRNG フィボナッチ ベース リング発振器 (FIRO) 制御レジスタ
CRYPTOLITE_TRNG_MON_CTL	TRNG ヘルスモニター制御レジスタ
CRYPTOLITE_TRNG_MON_RC_CTL	TRNG ヘルスモニター繰り返し回数テスト制御レジスタ
CRYPTOLITE_TRNG_MON_RC_STATUS0	TRNG ヘルスモニター繰り返し回数テストステータス 0 レジスタ
CRYPTOLITE_TRNG_MON_RC_STATUS1	TRNG ヘルスモニター繰り返し回数テストステータス 1 レジスタ
CRYPTOLITE_TRNG_MON_AP_CTL	TRNG ヘルスモニター適応プロポーションテスト制御レジスタ

(続く)

表 91 (続き) CryptoLite レジスタの概要

レジスタ	機能
CRYPTOLITE_TRNG_MON_AP_STATUS0	TRNG ヘルスモニター適応プロポーションテストステータス 0 レジスタ
CRYPTOLITE_TRNG_MON_AP_STATUS1	TRNG ヘルスモニター適応プロポーションテストステータス 0 レジスタ
CRYPTOLITE_INTR_TRNG	TRNG 割込みレジスタ
CRYPTOLITE_INTR_TRNG_SET	TRNG 割込みセット レジスタ
CRYPTOLITE_INTR_TRNG_MASK	TRNG 割込み要求マスク レジスタ
CRYPTOLITE_INTR_TRNG_MASKED	TRNG マスクされた割込み出力レジスタ

14 TrustZone

14.1 特長

- Arm® TrustZone 対応 Cortex®-M33 CPU
- セキュアな CPU 状態と非セキュアな CPU 状態
- 属性ユニット - セキュア属性ユニット (SAU) および実装定義属性ユニット (IDAU)
- メモリマップのセキュアなエイリアシングと非セキュアなエイリアシング
- セキュア割込みと非セキュア割込み
- セキュア ペリフェラルと非セキュア ペリフェラル
- セキュア デバッグと非セキュア デバッグ

14.2 アーキテクチャ

Arm® TrustZone テクノロジーは、Armv8-M 用に設計されたオプションのセキュリティ拡張機能です。このセキュリティ拡張機能は、PSOC™ Control C3 デバイスで有効化され、セキュリティが向上します。

TrustZone は、リソース (メモリとペリフェラル) をセキュアな世界と非セキュアな世界に分割することで、システム全体のセキュリティを実装します。TrustZone は、CPU にセキュアと非セキュアの 2 つの状態を採用しています。これらのセキュリティ状態は、既存のスレッド モードとハンドラ モードと直交しているため、4 つのモードになります。セキュアハンドラ、セキュアスレッド、非セキュアハンドラ、非セキュアスレッドの各モードです。PSOC™ Control C3 デバイスでは、TrustZone の拡張機能が有効になっているため、M33 CPU はセキュア状態で起動します。セキュア状態では、ソフトウェアはセキュアなリソースと非セキュアなリソースの両方にアクセスできますが、非セキュア状態では、ソフトウェアは非セキュアなリソースにしかアクセスできません。これにより、同じ CPU 上でセキュアなソフトウェアと非セキュアなソフトウェアを、メモリマップに基づく 2 つの世界間のリソースの分離とともに実行できます。メモリ マップの分割は、プログラマブルなセキュア属性ユニット (SAU) と固定の実装定義属性ユニット (IDAU) によって実現されます。

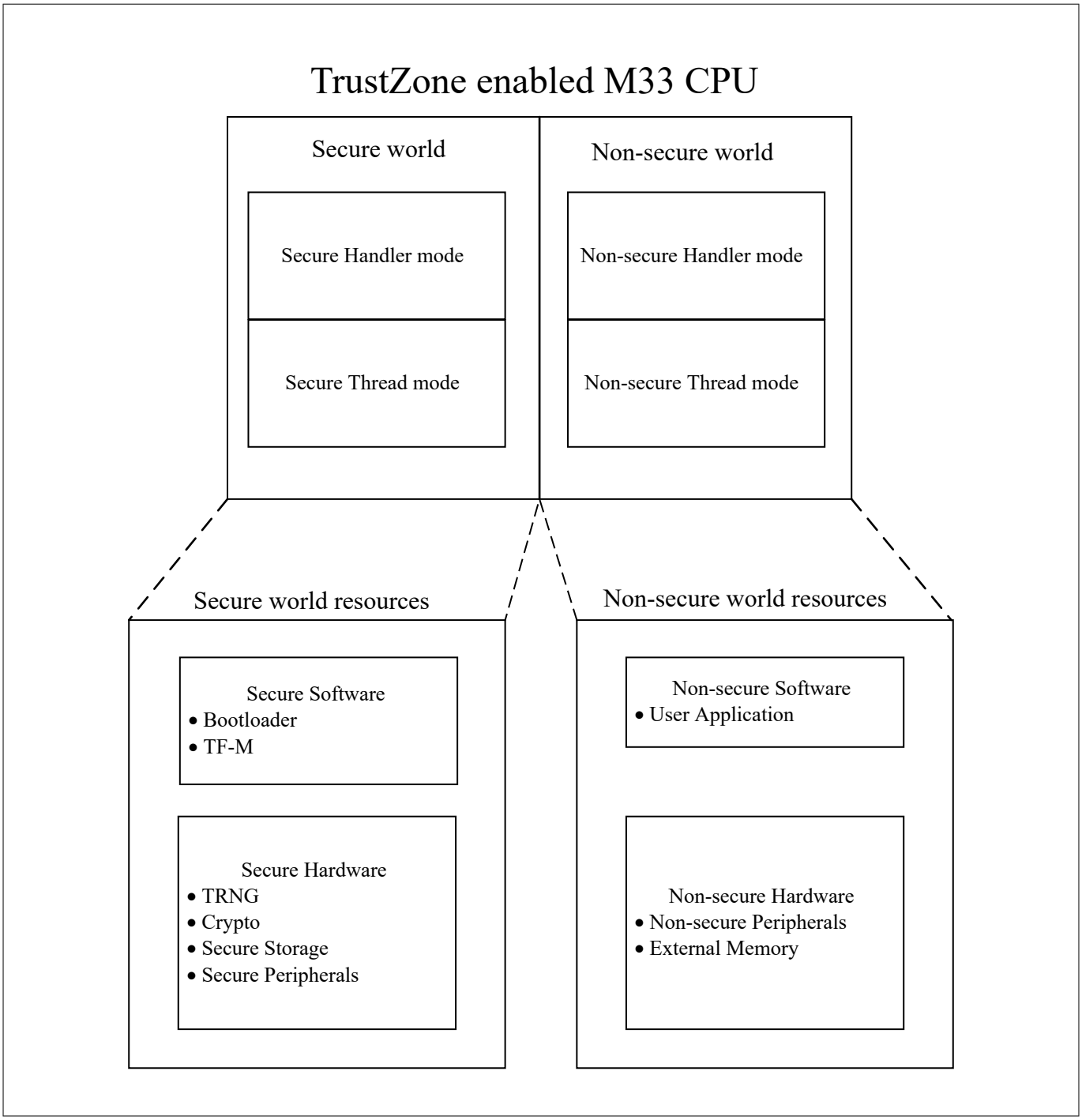


図 51　　Arm® TrustZone 対応システムの概要

以下の主要な内部リソースは、セキュア状態と非セキュア状態の間でバンクされます。

- スタック ポインタ
- マスクされた割込みレジスタ
- MPU
- SysTick タイマー
- VTOR
- システム制御ブロックのその他のレジスタ

一部のレジスタはビットごとにバンクされています。M33 CPU は、セキュアなソフトウェア用と非セキュアなソフトウェア用に別々のベクターテーブルがあります。セキュリティ拡張機能を有効にすると、新しい SecureFault 例外がシステムに追加されます。

14.3 メモリ領域

完全な 4GB のアドレス空間は、セキュア領域と非セキュア領域に分割されています。つまり、1 つのメモリ位置がアドレスマップの 2 つの異なる位置、1 つはセキュア領域、もう 1 つは非セキュア領域にマッピングされます。すべてのメモリとペリフェラルはエイリアスされ、両方の領域に配置されます。

アドレス空間は 3 つの領域に分割されます。

- セキュア (S):
  - セキュア アドレスは、セキュア ソフトウェアによってのみアクセス可能なメモリやペリフェラルへのアクセスに使用されます。
  - セキュア状態の CPU は、セキュア領域からの命令しか実行できません
  - セキュア アドレスのデータはセキュア ソフトウェアからアクセス可能
- 非セキュア コーラブル (NSC):
  - セキュア ゲートウェイ (SG) 命令を保持する特別なタイプのセキュア領域で、非セキュア状態からセキュア状態への状態遷移に対応しています。
- 非セキュア (Non-Secure:NS):
  - 非セキュア アドレスは、セキュア ソフトウェアと非セキュア ソフトウェアの両方からアクセス可能なメモリやペリフェラルへのアクセスに使用されます。
  - 非セキュア状態の CPU は、非セキュア領域の命令のみを実行します。
  - 非セキュアアドレスにあるデータは、セキュアソフトウェアからも非セキュアソフトウェアからもアクセス可能です。

属性ユニット、SAU、IDAU は、メモリ位置にセキュリティ属性を割り当て、メモリ空間を分割します。セキュリティ属性チェックに起因するアクセス違反はすべて、SecureFault 例外を生成します。

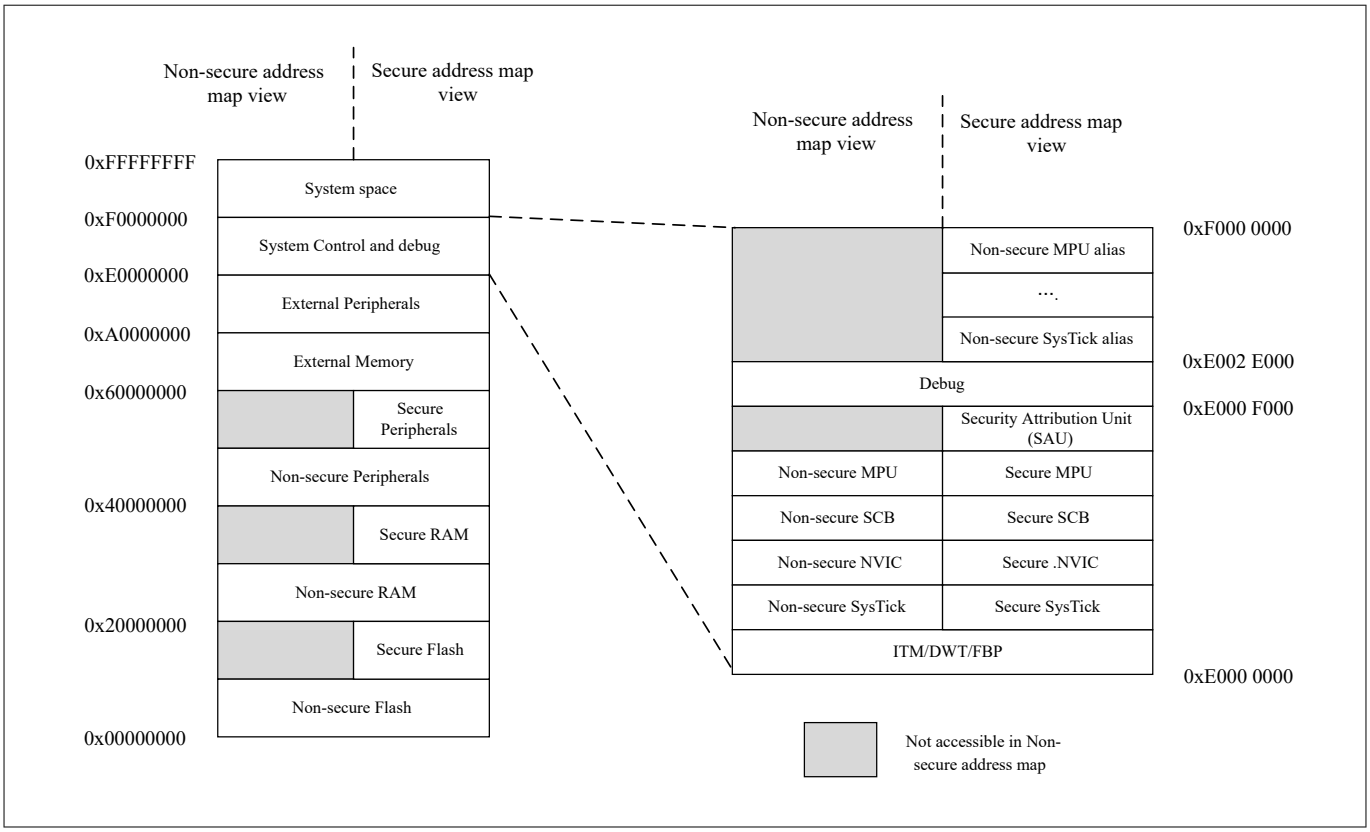


図 52　　TrustZone 対応システムのメモリ領域

14.4　　属性ユニット

メモリ空間は、セキュア属性ユニット (SAU) と実装定義属性ユニット (IDAU) で分割されます。SAU と IDAU は連動して、アドレスのセキュリティ属性を決定します。この 2 つのうち、セキュリティレベルの高い方が適用されます。HIGH から LOW のセキュリティレベルは以下のとおりです。セキュア (S) ＞非セキュア コーラブル (NSC) ＞非セキュア (NS)

表 92 は結果のセキュリティ属性を要約したものです。

表 92　　結果のセキュリティレベル

SAU	IDAU	結果のセキュリティ属性
S	X	S
X	S	S
NSC	NSC または NS	NSC
NSC または NS	NSC	NSC
NS	NS	NS

***注**:　　X=Don't Care*

14.4.1　　実装定義属性ユニット (IDAU)

- IDAU は、メモリ空間を非セキュア領域、非セキュア呼び出し可能領域、セキュア領域に分割できるアドレスエイリアス方式を提供します。

- IDAU は固定実装のため、アドレス エイリアス方式は設定できません。
- IDAU はアドレス ビット 28 を使って、セキュア メモリ領域と非セキュア メモリ領域を分離します。
- 専用周辺バス (PPB) 領域と外部専用周辺バス (EPPB) 領域はセキュリティ属性検査の対象外です。
- IDAU アドレスマップ パーティショニングは、より高いセキュリティレベルの SAU コンフィギュレーションでのみオーバーライド可能です。
- IDAU 実装は、M33 CPU およびすべての非 CPU マスターの AHB バスに外部接続されます。
- IDAU は、テストターゲット (TT) 命令に必要な、すべてのアドレスの領域番号を返します。

表 93 に、固定 IDAU 実装によって定義されたメモリ空間のパーティショニングを示します。

表 93　　IDAU メモリマップのパーティショニング

領域	開始アドレス	終了アドレス	サイズ	セキュリティ属性	地域番号
コード NS	0x0000 0000	0x0FFF FFFF	256 MB	NS	0x0
コード S	0x1000 0000	0x1FFF FFFF	256 MB	NSC	0x1
RAM NS	0x2000 0000	0x2FFF FFFF	256 MB	NS	0x2
RAM S	0x3000 0000	0x3FFF FFFF	256 MB	NSC	0x3
ペリフェラル NS	0x4000 0000	0x4FFF FFFFF	256 MB	NS	0x4
ペリフェラル S	0x5000 0000	0x5FFF FFFF	256 MB	S	0x5
外部メモリ NS	0x6000 0000	0x6FFF FFFF	256 MB	NS	0x6
外部メモリ S	0x7000 0000	0x7FFF FFFF	256 MB	S	0x7
外部メモリ予約済み	0x8000 0000	0x8FFF FFFF	256 MB	NS	0x8
	0x9000 0000	0x9FFF FFFF	256 MB	S	0x9
外部デバイス予約済み	0xA000 0000	0xAFFF FFFF	256 MB	NS	0xA
	0xB000 0000	0xBFFF FFFF	256 MB	S	0xB
	0xC000 0000	0xCFFF FFFF	256 MB	NS	0xC
	0xD000 0000	0xDFFF FFFF	256 MB	S	0xD
システムアドレス空間	0xE000 0000	0xE00F FFFF	1 MB	Exempt	0xE
	0xE010 0000	0xEFFF FFFF	255 MB	NS	0xE
	0xF000 0000	0xF00F 0000	1 MB	Exempt	0xF
	0xF010 0000	0xFFFF FFFF	256 MB	S	0xF

***注**:　IDAU はアドレスビット 28 を S および NS 表示に使用します。1 ビットは、0: NS, 1: S の 2 つの状態しか示せません。NSC 領域を示すために、IDAU は S 表示を再使用します。IDAU は、アドレス範囲 0x1000_0000 ～0x1FFF_FFFF (CODE S) および 0x3000_0000 ～0x3FFF_FFFF (RAM S) を、S 表示を再利用して NSC としてマークします。*

14.4.2　セキュア属性ユニット (SAU)

- SAU はプログラマブル ユニットで、M33 CPU の一部です
- SAU コンフィギュレーションは M33 CPU にのみ適用可能
- SAU は 8 つのプログラマブル領域に対応
- SAU はリセット時に無効

- SAU が無効な場合、アドレス空間の属性は SAU_CTRL.ALLNS ビット フィールドによって制御
- SAU が有効な場合、8 つのプログラマブル領域は、メモリを NS または NSC としてマークするために使用されます。SAU によってカバーされていないメモリ領域は、セキュアとしてマークされます。
- SAU を設定し、非セキュアソフトウェアの実行を可能にする NS および NSC 領域を作成するのは、セキュアソフトウェアの仕事です。

表 94 に、SAU コンフィギュレーションの例を示します。

表 94　　SAU のコンフィギュレーション例

領域	説明	開始アドレス	終了アドレス	SAU セキュリティ属性	IDAU セキュリティ属性	結果のセキュリティ属性
コード NS	フラッシュ	0x02000000	0x0201FFFF	-	NS	S
		0x02020000	0x0203FFFF	NS		NS
	SRAM	0x04000000	0x04007FFF	-	NS	S
		0x04008000	0x0400FFFF	NS		NS
コード S	フラッシュ	0x12000000	0x1201FEFF	-	NSC	S
		0x1201FF00	0x1201FFFF	NSC		NSC
		0x12020000	0x1203FFFF	-		S
	SRAM	0x14000000	0x1400FFFF	-	NSC	S
RAM NS	フラッシュ	0x22000000	0x2201FFFF	-	NS	S
		0x22020000	0x2203FFFF	NS		NS
	SRAM	0x24000000	0x24007FFF	-	NS	S
		0x24008000	0x2400FFFF	NS		NS
RAM S	フラッシュ	0x32000000	0x3203FFFF	-	NSC	S
	SRAM	0x34000000	0x3400FFFF	-	NSC	S
ペリフェラル NS	ペリフェラル	0x42000000	0x43FFFFFF	NS	NS	NS
ペリフェラル S	ペリフェラル	0x52000000	0x53FFFFFF		S	S
外部メモリ NS	-	0x60000000	0x6FFFFFFF		NS	NS
外部メモリ S	-	0x60000000	0x7FFFFFFF		S	S
外部メモリ予約済み	-	0x80000000	0x8FFFFFFF		NS	NS
		0x90000000	0x9FFFFFFF		S	S
外部デバイス予約済み	-	0xA0000000	0xAFFFFFFF		NS	NS
		0xB0000000	0xBFFFFFFF		S	S
		0xC0000000	0xCFFFFFFF		NS	NS
		0xD0000000	0xDFFFFFFF		S	S
システムアドレス空間	システム制御とデバッグ	0xE0000000	0xE00FFFFF		Exempt	Exempt
		0xE0100000	0xEFFFFFFF		NS	NS
		0xF0000000	0xF00F0000		Exempt	Exempt
		0xF0100000	0xFFFFFFFF		S	S

14.5 状態の遷移

M33 には、セキュリティの状態遷移をサポートするために、3 つの命令が定義されています。

- BXNS: エクスチェンジから非セキュアへの分岐
- BLXNS: 非セキュアへのリンクと交換を伴うブランチ
- SG: セキュア ゲートウェイ

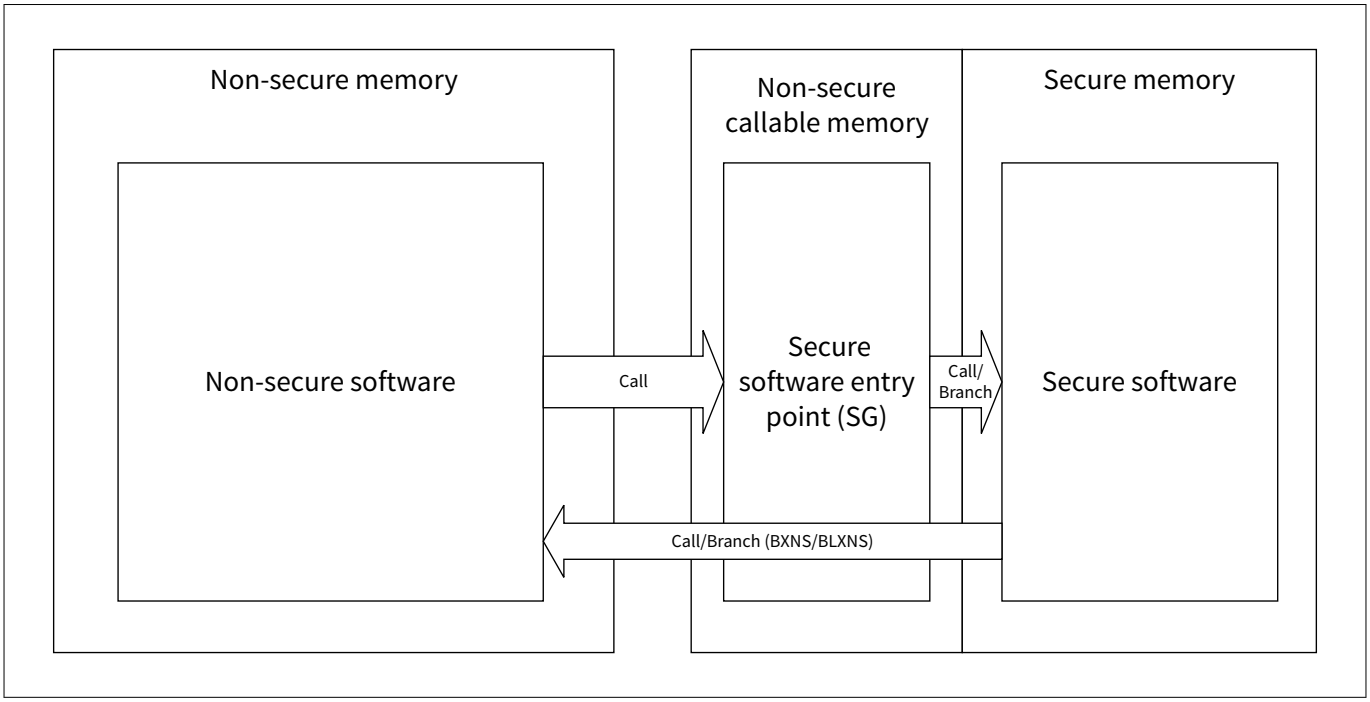


図 53 セキュリティ状態遷移

14.5.1 セキュアから非セキュアへの状態遷移

リセット後、M33 CPU はセキュア状態でスタートします。CPU は、セキュア状態ではセキュア メモリからの命令しか実行できません。セキュアソフトウェアが非セキュアソフトウェアを呼び出す場合、CPU の状態遷移は BXNS または BLXNS 命令から開始できます。BXNS 命令を呼び出すと、状態遷移が実行され、非セキュアなアドレスに分岐します。BXLNS は状態遷移を実行し、非セキュアなアドレスのサブルーチンを呼び出します。

これらの命令は、CPU がセキュア状態のときにのみ実行されます。BXNS/BLXNS は、非セキュア状態から呼び出された場合、未定義です。

14.5.2 非セキュアからセキュアへの状態遷移

非セキュア ソフトウェアはセキュア メモリに直接アクセスできません。状態遷移のために、SG 命令は非セキュア コール可能領域から最初に実行される命令でなければいけません。SG 命令の実行に成功すると、CPU はセキュア状態に移行し、セキュアソフトウェアへのコール/ブランチが実行できるようになります。通常、非セキュアからセキュアへの状態遷移は、セキュア ソフトウェアからサブルーチンを呼び出すために実装されます。

SG 命令は、非セキュア呼び出し可能領域からのみ実行される必要があります。非セキュア領域から実行された場合、命令は NOP として動作します。

14.5.3 例外または割込みからの状態遷移

状態遷移は、例外や割込みによって起こることもあります。例外と割込みは、セキュアまたは非セキュア状態をターゲットに設定できます。

- CPU がセキュア状態にあるときに、非セキュア状態を対象とする例外または割込みがトリガーされると、セキュア状態から非セキュア状態に遷移します。
- CPU が非セキュア状態にあるときに、セキュア状態を対象とする例外または割込みがトリガーされると、非セキュア状態からセキュア状態に遷移します。
- 到着した割込みの状態が現在の CPU の状態と同じであれば、状態遷移は起こりません。

15 保護ユニット

PSOC™ Control C3 MCU は、転送の特定の特性に基づいてバス転送を許可または制限することにより、セキュリティを実施するための保護ユニットを実装しています。保護を決定するルールは、保護ユニットの MMIO レジスタに実装されており、保護アドレス空間と保護属性を定義します。こうした保護属性を評価し、アクセスを制限または許可するハードウェアが保護ユニットです。PSOC™ Control C3 デバイスには、セキュア属性ユニット (SAU)、実装定義属性ユニット (IDAU)、メモリ保護ユニット (MPU)、メモリ保護コントローラー (MPC)、ペリフェラル保護コントローラー (PPC)、マスター セキュリティ コントローラー (MSC) といった異なるタイプの保護ユニットがあります。各タイプの保護 ユニットには、異なる保護領域とその属性を定義するために役立つ MMIO レジスタの明確なセットがあります。

PSOC™ Control C3 MCU プラットフォームは Armv8-M アーキテクチャをベースにしており、TrustZone セキュリティ機能はこのアーキテクチャの一部です。TrustZone 対応システムでは、ソフトウェアはセキュアなドメインと非セキュアなドメインに分けられます。プロセッサは、プログラマブル SAU とデバイス固有のロジック ブロック IDAU を組み合わせて、アドレス空間のパーティショニングを処理します。詳細については、TrustZone を参照してください。

SAU と IDAU は不可欠ですが、セキュリティ重視のアプリケーションにおいて、メモリ空間を多数の領域に分割するためには十分ではない可能性があります。このような場合、アドレス空間をページ単位など、より多くの領域に分割する必要があることもあります。MPC と PPC は、メモリ空間と周辺空間をいくつかの領域への分割を可能にします。これらのユニットは、非セキュア属性と保護コンテキストに基づいて、各メモリとペリフェラル領域のアクセシビリティを制御します。MPU は、CPU の一部であるプログラマブル ユニットで、特権ソフトウェアがメモリ アクセス権限を定義できるようになっています。さらに、セキュリティ拡張機能を持たないバス マスターのために、MSC は TrustZone 機能を追加します。

保護ユニットは、アクセス属性に従って、バス上の転送を許可または制限します。アクセス属性には、次のものがあります。

- Read/Write 属性
- コードアクセスとデータアクセスを区別するための属性実行
- 例えば、OS やカーネルへのアクセスと、タスクやスレッドへのアクセスを区別するための、ユーザーまたは特権属性
- 「セキュア」アクセスと「非セキュア」アクセスを区別するためのセキュアまたは非セキュア属性。
- 異なる保護コンテキストからのアクセスを区別するための保護コンテキスト属性

セキュアなシステムの機能は、以下のコンポーネントに基づいています。

- **バス マスター**: この用語は、アーキテクチャ内のバス マスターを指します。PSOC™ Control C3 デバイスにおいて、バスマスターの例には、Cortex®-M コア、DW、またはテストコントローラーが含まれます。
- **保護ユニット**: メモリ、ペリフェラル、共有リソースをバス マスターから保護するための保護スキームを実装するハードウェア ブロックです。保護ユニットの MMIO レジスタで定義された保護を適用します。
- **保護 MMIO レジスタ**: MMIO レジスタは、メモリ上にレジスタ構造を形成し、各保護ユニットが転送を評価する際のルールを定めます。各保護ユニットはレジスタ構造に関連付けられています。MMIO レジスタの構造は以下のとおりです。
  - ルールが適用されるメモリ領域。メモリ領域は、開始/終了アドレスまたは開始/終了ブロックインデックスで定義できます。
  - 保護属性のセットで、転送が評価される基準となる特性。これらには以下が含まれます。
    - R/W/X
    - ユーザー/特権
    - セキュア/非セキュア
    - 保護コンテキスト

MMIO レジスタで設定可能な保護属性のセットは、関連する保護ユニットによって異なります。

各バス マスターには、バス マスターのアクセス権限を定義するアクセス属性が関連付けられています。バス マスターは、セキュア/非セキュア、保護コンテキスト、ユーザー/特権属性などの属性と関連付けられています。こ

れらは、バス マスターのコンテキストと状態に基づいて変化する動的な属性です。しかし、バス マスターによっては、セキュア/非セキュアやユーザー/特権属性といった属性が固定されており、変更できないものもあります。

保護の属性は 2 箇所に現れます。

- 保護 MMIO レジスタ
- バス マスターの保護属性

保護 ユニットは、これらのマスターのアクセス属性を、メモリ/ペリフェラルのアクセス属性と照らし合わせて評価します。アクセス属性 (バス マスターとメモリ/ペリフェラル) が一致すれば、保護 ユニットは転送を許可し、そうでなければ転送はブロックされます。

保護ユニットは以下に対応しています。

- **セキュリティ要件**: これには、セキュアなメモリやペリフェラルへアクセスし行われる悪意ある攻撃の防止も含まれます。
- **安全要件**: これには、偶発的な (悪意のない) ソフトウェアエラーやランダムなハードウェアエラーの検出も含まれます。安全違反の根本原因を調査するために、故障解析を可能にすることが重要です。

15.1 アーキテクチャ

バスマスターがメモリやペリフェラルにアクセスすると、そのアクセスは保護ユニットによって、アクセスされるメモリやペリフェラルの場所の保護属性と照らし合わせて評価されます。バス マスターの保護属性がメモリまたはペリフェラルの保護属性を満たす場合、保護ユニットはアクセスを許可します。アクセス違反の場合、フォールト コンディションがトリガーされ、バス エラーとなります。MPC および PPC では、エラー応答をバス エラーまたは Read Zero Write Ignore (RAZ/WI) のいずれかに設定できます。

MPU、SAU、IDAU は M33 CPU サブシステムの一部で、4GB のアドレス空間全体を保護します。SAU および IDAU の詳細については、属性ユニットを参照してください。MPC は M33 CPU サブシステムの一部でもあり、不正アクセスからメモリを保護します。ペリフェラル アドレス空間内のペリフェラル保護用に、別の保護ユニット タイプ PPC が用意されています。さらに、マスター セキュリティ コントローラー (MSC) は、TrustZone 機能を追加するセキュリティ拡張機能を持たない非 CPU バス マスター用に実装されています。

- **MPU**: M33 バス マスターには専用の MPU があり、特権ソフトウェアの制御下で CPU の一部として実装されています。通常、MPU のアクセス属性は、RTOS/TF-M SPM のような特権ソフトウェアによって動的に更新されます。
- **MPC**: MPC は、複数のバスマスターを持つシナリオで、保護を実装するよう設計されています。保護コンテキストと呼ばれる概念に基づいて保護を実装し、異なる保護コンテキストを区別し、セキュアなアクセスと非セキュアなアクセスを区別できるようにすることで、マルチマスターシナリオで効果的な保護を提供します。
- **PPC**: PPC は、ペリフェラル保護のためにペリフェラルレジスタ空間に設けられた保護ユニットです。その属性は MPC と似ていますが、周辺空間の保護を目的としている点が異なります。PPC は、異なる保護コンテクストを区別し、セキュアアクセスと非セキュアアクセスを区別し、ユーザーモードアクセスと特権モードアクセスを区別します。特定のペリフェラルに対応する固定アドレス領域の保護を実装しています。
- **MSC**: MSC は、TrustZone 機能を追加するためのセキュリティ拡張属性を提供するためインスタンス化されます。マスターのセキュリティ属性を設定し、バス上で開始されるトランザクションを保護するため、セキュリティ拡張機能のないマスターとペアになります。マスターの例には、DW と Crypto があります。

PSOC™ Control C3 保護アーキテクチャは以下のコンポーネントで構成されています。

図 54 に、PSOC™ Control C3 保護アーキテクチャの概要を示します。

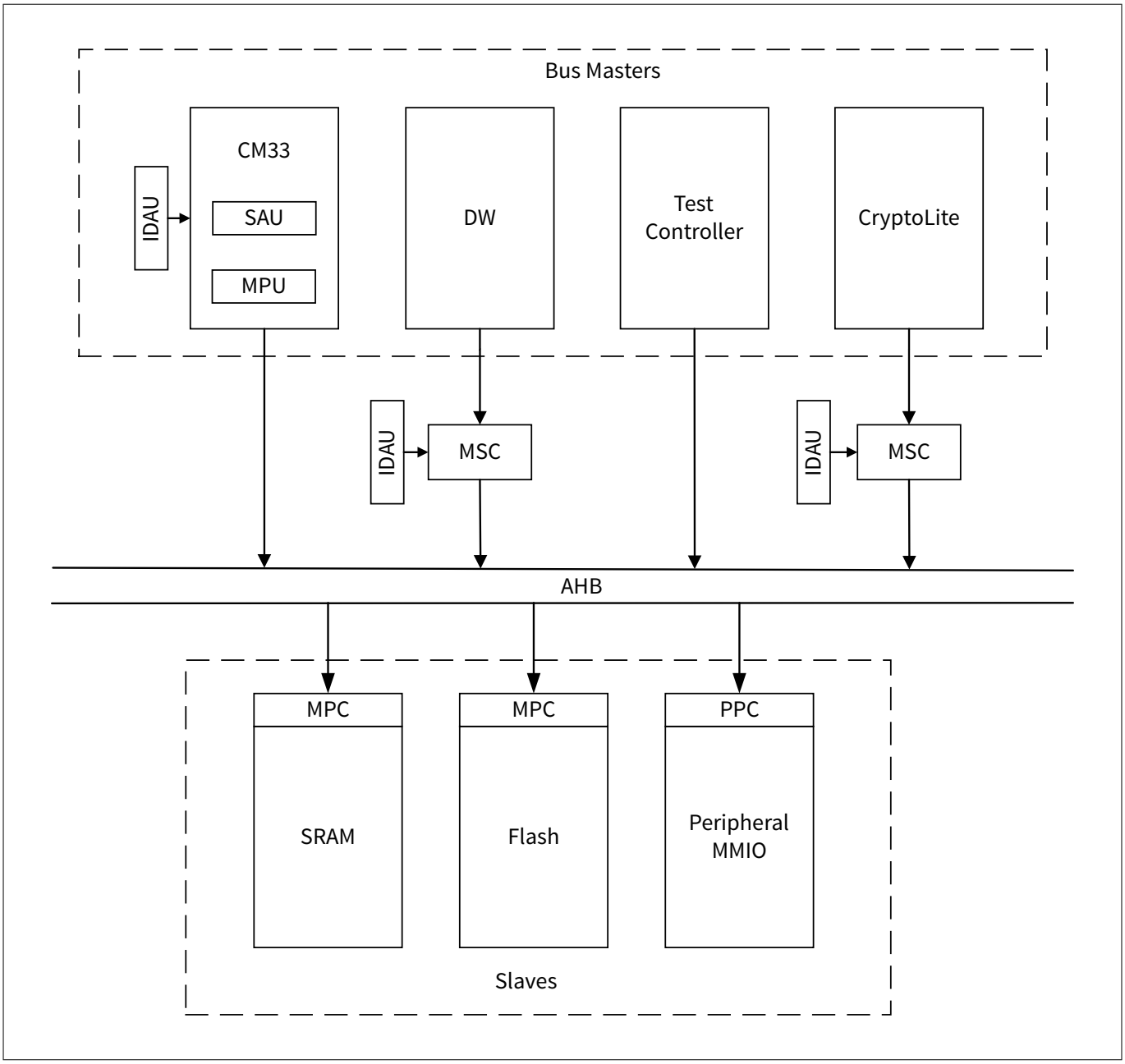


図 54　　保護アーキテクチャ

保護違反は、バスマスターのアクセス属性とメモリまたはペリフェラル領域の保護属性の不一致によって発生します。これらの違反は、故障解析を可能にするようフォールト レポート構造に取り込めます。フォールト レポート構造はエラーの発生を示す割込みを生成できます。これは、違反したバス マスター自身でバス エラーを解決できず、他の CPU バス マスターにバス エラーを解決してもらわなければいけない場合に特に有効です。保護違反はバスエラーとなり、バス転送は意図したターゲットに到達する前に停止します。

図 55 に、PSOC™ Control C3 デバイスの属性チェックの優先順位を示します。

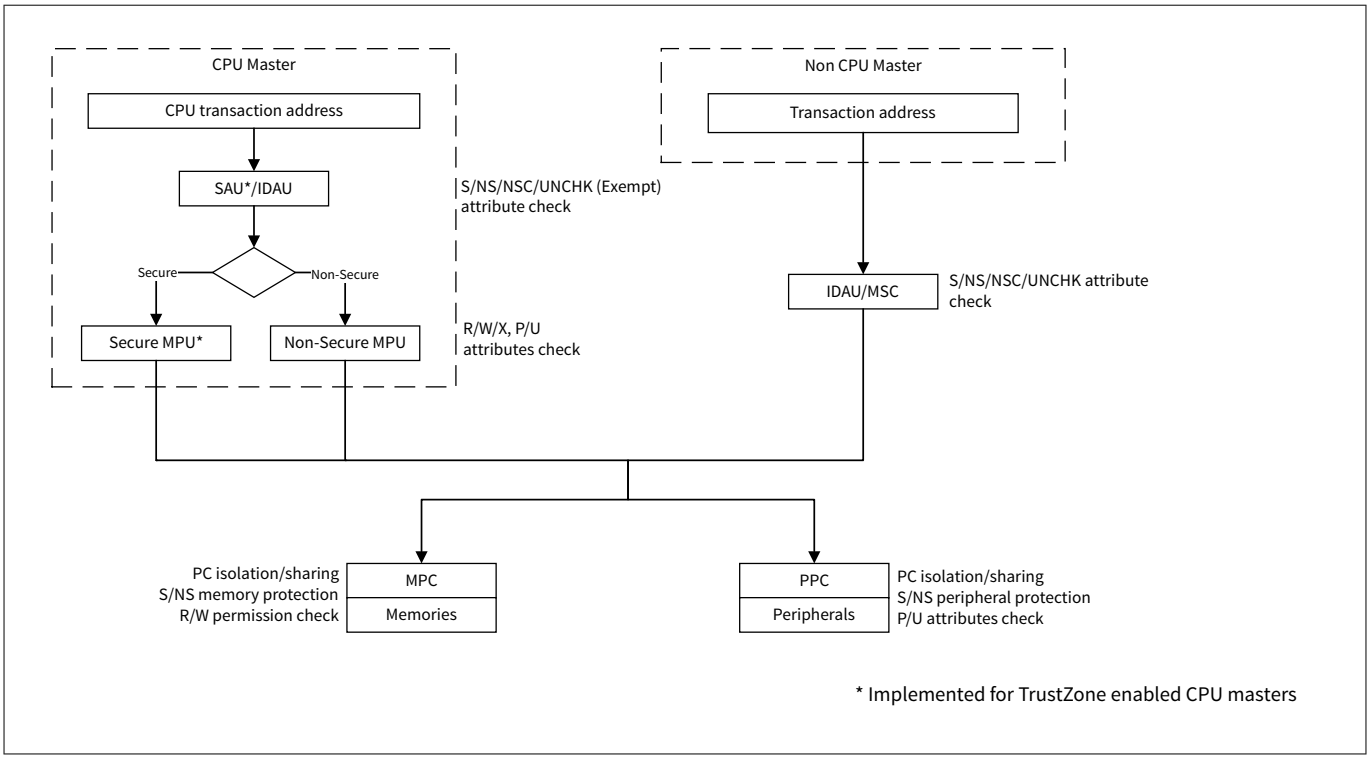


図 55　　属性チェックの優先順位

保護 ユニットの種類は、次の表に示すように、さまざまな使用ケースに対応しています。

表 95　　PSOC™ Control C3 保護ユニット

保護ユニット	用途	インスタンス数
MPU	単一の Arm®コア内のタスク間のメモリを保護するために使用されます。Arm®コアの 1 つにあるタスクは、同じコアにある別のタスクからのアクセスからメモリを保護できます。	M33 CPU: セキュア MPU×1、非セキュア MPU×1、各 MPU に 8 つのプログラマブル領域を搭載
MPC	複数のバス マスター間で共有される (保護コンテキストが異なる) メモリを保護するために使用されます。	2
PPC	これらのユニットは、デバイス定義の周辺メモリ空間を保護します。PPC は固定領域を持ち、事前に定義された周辺アドレス領域 (PPC 領域) を保護するためにのみ使用されます。	1
SAU	セキュアリソースと非セキュアリソースを分離するためのアドレス空間のパーティションに使用されます。	1
IDAU	セキュアリソースと非セキュアリソースを分離するため、アドレス空間の固定パーティショニングを実装します。	3

(続く)

表 95 (続き) PSOC™ Control C3 保護ユニット

保護ユニット	用途	インスタンス数
MSC アダプター	セキュリティ拡張機能のないマスターに TrustZone 機能を追加するために使用します。	2

15.2 保護コンテキスト

保護 コンテキストは、インフィニオン独自の保護 メカニズムで、複数のバス マスター間で共有されるメモリやペリフェラルへのアクセス制限、および同じバス マスターからのアクセス制限を実現します。各バス マスターは保護コンテキスト フィールド MS_PCx_PC.PC[3:0]を持っており、マスターが開始するすべてのバス転送の保護コンテキスト属性として利用されます。MPC と PPC はこの保護コンテキスト属性を利用して、バスマスターとメモリまたはペリフェラル間のバス転送を許可または制限します。

マスターの MS_PCx_PC.PC[3:0] (PC[]) フィールドをプログラミングすることで、バス マスターの保護コンテキストを設定できます。保護コンテキストの変更は、複数のタスク間を移行し、それぞれが異なる保護 コンテキストに対応する可能性があるバス マスターにとって必要です。保護コンテキスト属性はバス転送を有効または制限するため、保護 コンテキスト フィールドの変更は、セキュリティを損なわないように注意深く制御する必要があります。各バス マスターには、MSx_CTL.PC_MASK[7:0] (PC_MASK[]) 保護コンテキスト マスク フィールドがあり、バス マスターに許可される保護コンテキストを識別します。このビット フィールドは、バス マスターに許可される保護コンテキストを指定する「one-hot」フィールドです。複数のバス マスターが保護コンテキストを共有できます。

保護コンテキストフィールド PC[ ]は、アクティブな保護コンテキスト (PC) を示し、その変更は、関連する PC_MASK[ ]値によって制限されます。PC[]は、対応するマスク ビット PC_MASK[i]が'1'のときのみ i に設定できます。

例えば、MS_PC_STRUCT[x].PC[3:0]を‘4’にプログラムしようとした場合:

- PC_MASK[4]が‘1’の場合、PC[ ]は'4'に設定されます。
- PC_MASK[4]が‘0’の場合、PC[ ]は変更されません

PSOC™ Control C3 MCU は 8 つの保護コンテキストをサポートしており、各コンテキストは特定のタスク用にあらかじめ定義されています。表 96 に各タスクの PC の割り当てを示します。

表 96 ファームウェア層への保護コンテキストの割り当て

PC	ファームウェア/ソフトウェア層
0	ROM_BOOT/FLASH_BOOT
1	保護されたファームウェア
2	OEM_BL/TF-M
3	OEM NS アプリ
4	
5	
6	
7	SYS-AP (MXDEBUG600)

表 97 は、デバイス ブート時に設定されたバス マスターの保護属性を指定します。

表 97 マスター セキュリティ属性

マスター名	マスター ID	MSx_CTL 値			MS_PCx_PC 値
		特権属性	NS 属性	PC_MASK	アクティブ (PC)
M33 CPU	0	-	-	PC1, PC2, PC3, PC4, PC5, PC6	PC2
DW	4, 5	Inherited		PC1, PC2, PC3, PC4, PC5, PC6	Inherited
MXCRYPTOLITE	7	Inherited		PC1, PC2, PC3, PC4, PC5, PC6	Inherited
テスト コントローラー	31	-	-	PC7	PC7

独自のアクセス属性を提供するマスターの場合、MSx_CTL レジスタ構造の対応するビット フィールドは実装されません。例えば、M33 CPU の MS0_CTL は、独自の P 属性と NS 属性を提供するため、P ビットフィールドと NS ビットフィールドを持ちません。

注: DW コントローラーおよび暗号アクセラレーターは、チャネルまたはコンポーネントをプログラムしたバス転送のアクセス制御属性を継承します。

15.2.1 特殊な保護コンテキスト

M33 CPU バス マスターのアクティブ PC は、PC[ ]ビット フィールドに書き込むことで変更できます。このフィールドへのアクセスは通常、PC2 セキュア ソフトウェアに制限されており、アクティブ PC は他のソフトウェアから直接変更できません。PC4/5/6 から PC1/2/3 への切り替えが必要な場合、ハンドラ機構を用いなければ実行できません。

8 つの保護コンテキストのうち、PC1、PC2、PC3 は特別なものとして扱われ、これらの特別な PC へのエントリーはハードウェアで制御されます。特殊 PCx (x は特殊 PC 番号を指す) の場合、プログラマブル例外ハンドラ アドレスが MXCM33_CM33_PCx_HANDLER.ADDR[31:0]に提供されます。ビットフィールド MXCM33_CM33_PC_CTL.VALID[x]は、MXCM33_CM33_PCx_HANDLER.ADDR[31:0]で提供される PCx のハンドラが有効かどうかを示します。CPU 例外ハンドラ フェッチが、プログラムされた MXCM33_CM33_PCx_HANDLER.ADDR[31:0]アドレス値と一致するハンドラ アドレスを返した場合、CM33 PC はハードウェアによって PCx に変更されます。PC の MXCM33_CM33_PC_CTL.VALID[x]が設定されていない場合、その PC は通常の PC として扱われます。

OEM アプリケーションから MXCM33_CM33_PCx_HANDLER.ADDR[31:0]および MXCM33_CM33_PC_CTL.VALID[x] ビットフィールドへの書き込みは禁止されています。これらのレジスタは、OEM のプロビジョニング プロセスを通じてのみプログラムできます。OEM ポリシーは、これらのレジスタをプログラミングするための JSON オブジェクトを提供します。

M33 バス マスターの場合、MS_PC0_PC レジスタには、保存された PC を保持する追加の MS_PC0_PC.PC_SAVED[19:16] ビット フィールドが含まれます。現在の PC と保存された PC は 2 つのエントリーのスタックを実装しています。特別な例外ハンドラが入ると、ハードウェアは現在の PC をスタックにプッシュします。例外ハンドラを終了する前に、アクティブ PC とセーブ PC をリストアすることは、例外ハンドラの責任です。

PCx がハードウェアで制御されている場合 (MXCM33_CM33_PC_CTL.VALID[x]ビットを‘1’で表示)、保護コンテキストはソフトウェアで選択できません。この場合、PC_MASK[x]に書き込まれたデータの対応するビットは強制的に‘0’になります。ハンドラによる PC 切り替えは M33 CPU のみに実装されており、他のバスマスターには実装されていないことに注意してください。

15.3 マスター保護属性

保護ユニットの MMIO レジスタにある保護属性は、異なるメモリ領域とそのアクセス属性のルールを確立します。保護ユニットは、バス マスター自身の保護属性を使用し、保護ユニットの MMIO レジスタに設定されたルールに基づいてアクセスを規制します。すべてのバスマスターが、バス転送に関連するすべての保護属性を提供するわけではありません。例えば、バス マスターのいずれにも、実行中のタスクに従って動的に設定されるべきネイティブ保護コンテキストの属性がありません。

システム全体のアクセス制限を確実にするため、不足する属性はバス マスターの MMIO レジスタ フィールドで提供されます。これらのフィールドのいくつかは、ROM ブート プロセス中に設定され、ユーザー コードによる変更はできません。

- PC_MASK[]および PC[]レジスタ フィールドは、保護コンテキスト機能を提供します。
- MSx_CTL.P レジスタフィールドは、独自に属性を持たないマスターに、ユーザー/特権属性を付与します。
- MSx_CTL.NS レジスタ フィールドは、独自にセキュア/非セキュア属性を持たないバスマスターにセキュア/非セキュア属性を付与します。
- execute 属性を持たないマスターの execute 属性は'0'に設定されます。

ROM ブートコードによって設定されるバスマスター保護属性については表 97 を参照してください。

15.3.1 マスター セキュリティ コントローラー (MSC)

AHB5 TrustZone マスター セキュリティ コントローラー (MSC) は、レガシー AHB ライト マスターまたはセキュリティ拡張機能のない AHB5 マスターを AHB5 システムに接続し、TrustZone for ARMv8-M 機能を追加するためのアダプタとして機能します。セキュリティ拡張のないマスターは、MSx_CTL レジスタの一部である NS ビット フィールドをプログラムするか、またはプログラムしたマスターから属性を継承することで、セキュアまたは非セキュアに設定できます。MSC は、セキュリティを意識していないマスターの NS 属性を IDAU に対して評価します。マスターが適切な NS 属性を持っていればアクセスは許可され、そうでなければブロックされます。MSC は、アクセス違反の場合に RZWI を動作させるか、MSC 割込みを発生させるようにプログラムできます。

MSC の詳細については、CoreLink SIE-200 テクニカル リファレンス マニュアル (Arm®社) を参照してください。

15.4 スレーブ保護属性

MPU、MPC、PPC は、マスターとのすべてのトランザクションの前に、メモリおよびペリフェラル スレーブのアクセス属性を評価します。

15.4.1 メモリ保護ユニット (MPU)

ARMv8-M のメモリ保護ユニット (MPU) はプログラマブル ユニットで、CPU の一部です。特権ソフトウェアが、通常のメモリやデバイス メモリに対するメモリ アクセス許可 (R/W/X) やメモリ属性を定義したり、CPU からの命令フェッチやデータ アクセス操作を監視したりできます。PSOC™ Control C3 プラットフォームでは、M33 CPU は 1 つのセキュア MPU と 1 つの非セキュア MPU を搭載し、各 MPU には 8 つのプログラマブル領域があります。MPU がアクセス違反を検出すると、MemManage フォールトが発生します。

アーキテクチャとレジスタの詳細については、Arm®の Armv8-M メモリモデルおよびメモリ保護ユニット (MPU) ユーザーガイドを参照してください。

15.4.2 メモリ保護コントローラー (MPC)

メモリ保護コントローラー (MPC) は AHB バス上にある。SRAM とフラッシュには MPC が搭載されています。

MPC は、以下のパラメーターを提供します。

- ランタイム メモリ分離または分離と共有に基づいて保護コンテキスト (PC) を割り当てます。

- セキュアな世界と非セキュアな世界の間で、ARMv8-M TrustZone-M 準拠のメモリ保護コントローラーとして機能します。
- セキュアなアドレス エイリアスでも、非セキュアなアドレス エイリアスでも、メモリをアクセス可能にし、見えるようにします。
- メモリ空間を固定サイズのブロック/ページに分割し、SRAM ブロックは 2 KB、フラッシュブロックは 2 KB とします。
- すべての PC に対して各ブロックの NS/R/W 属性を設定します。
- セキュアな領域を対象とする非セキュアなアクセスをブロックします。
- 非セキュア領域を対象とするセキュアなアクセスをブロックします。
- 設定可能なエラー応答を提供:
  - MPCx_CFG.RESPONSE = 1'b0: ハードウェアが Read Zero Write Ignore (RZWI) を実行
  - MPCx_CFG.RESPONSE = 1'b1: ハードウェアによるバスエラー

MPC は LUT_SRAM をプログラムする MMIO レジスタで構成されます。LUT_SRAM はセキュリティ属性のルックアップテーブルを含み、トランザクションを許可またはブロックする属性チェック ロジック ブロックに接続されています。

図 56 に MPC ブロックのアーキテクチャを示します。

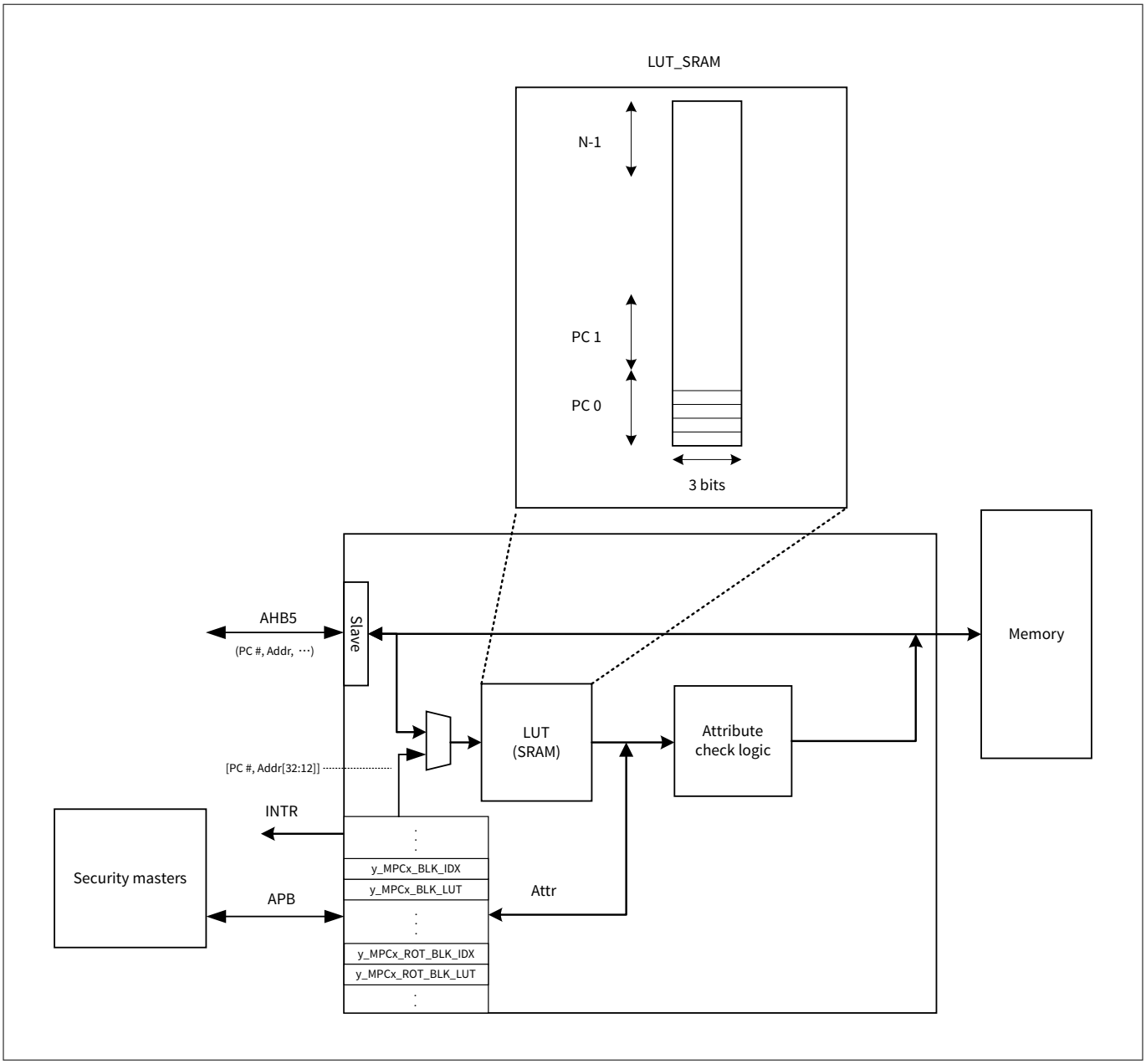


図 56　　MPC のアーキテクチャ

注:　セキュア/非セキュア(NS) 属性は、「セキュア」アクセスと「非セキュア」アクセスを区別し、アクセスされる場所の結果的なセキュリティ属性を反映します。データアクセスにおいて、S 状態の CPU が NS ロケーションにアクセスした場合、そのアクセスは NS とみなされます。同様に、NS 状態の CPU が NS ロケーションにデータアクセスを行った場合、NS とみなされます。同様に、命令アクセスの場合、セキュア/非セキュア(NS) アクセス属性は、アクセスされるメモリ位置のセキュリティ属性を表します。

15.4.2.1　　ソフトウェアビュー

MPC は、すべての PC のメモリブロックごとに 3 ビットの NS/R/W 属性を含む SRAM (SRAM_LUT) で構成されるルックアップテーブルを含みます。各 PC に対して、メモリブロックごとに NSx1、Rx1、Wx1 の属性ビットが予約され、LUT_SRAM に格納されます。LUT_SRAM の総ビット数は 3 * PC_NR * メモリモジュールのブロック数で、サポートされる PC_NR の最大値は 8 です。

MPC コンフィギュレーションは OEM ポリシーの一部で、ブート ROM によって適用されます。OEM アプリケーションで実行時に変更することはできません。すべての NS/R/W 属性は、セキュア ブート ROM ソフトウェアによって設定され、ロックされます。

リセット時、ハードウェア ステート マシンは LUT_SRAM をすべての PC に対して R/W アクセス可能なセキュア メモリに初期化します。LUT_SRAM と同様に、すべてのレジスタは DEEPSLEEP_RAM/OFF 電力モードでは保持されず、ウェイクアップイベントの後に復元される必要があります。

図 57 に MPC のブロック図を示します。

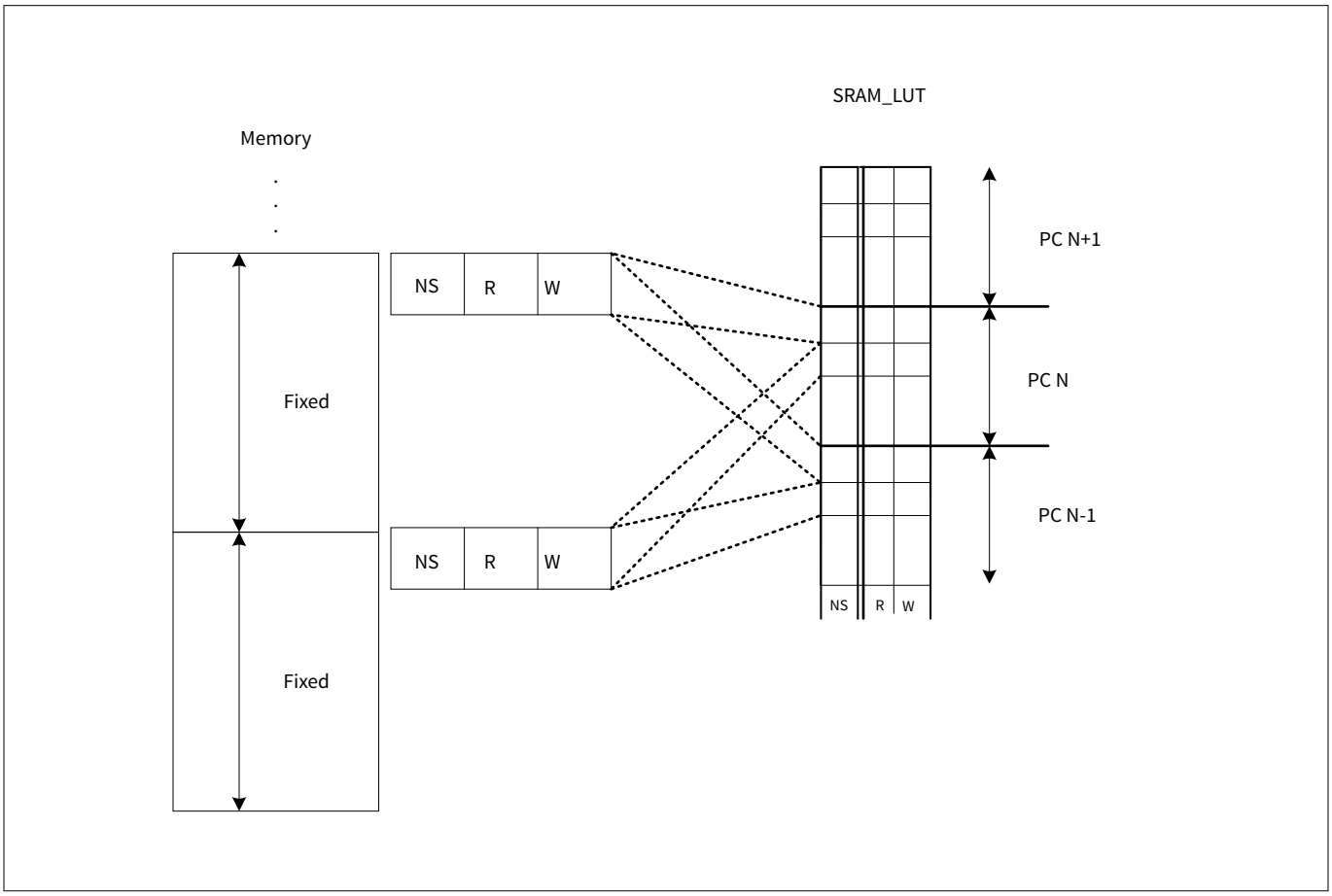


図 57　　MPC ソフトウェアビュー

15.4.2.2　　MPC MMIO レジスタ保護

MPC の MMIO レジスタは PPC によって保護され、信頼されたソフトウェアのみが MPC のコンフィギュレーションに変更を加えられます。x_PPC_MPC_MAIN, x_PPC_MPC_PC, x_PPC_MPC_ROT PPC 領域は MPC MMIO レジスタへのアクセスを制御します。PSOC™ C3 プラットフォームでは、MPC 用の PPC 領域は、MPC コンフィギュレーションの変更を PC0 で実行される ROM コードのみに制限する方法で、ROM によって設定されます。

15.4.3　　ペリフェラル保護コントローラー (PPC)

PPC は、Arm® TrustZone-M のセキュアまたは非セキュア (NS)、特権または非特権 (P)、および保護コンテキスト (PC) 属性に基づく周辺アクセス制御を提供します。バス マスターとペリフェラルの間に位置する PPC は、通常同じセキュリティ許可を必要とするペリフェラルの MMIO レジスタを「領域」にグループ化します。例えば、SCB0 のレジスタは 1 つの領域にまとめられています。各 PPC 領域は PC/NS/P 属性で設定でき、ペリフェラルはアクセス制御の必要性に応じて複数の PPC 領域を持つことができます。PPC は、ペリフェラル領域のセキュリティ属性と

マスターからのトランザクション属性を照合し、セキュリティ違反をチェックします。そのトランザクションにセキュリティ違反がある場合は、ブロックされます。

PPC は、以下のパラメーターを提供します。

- AHB5 バス インターフェースによる TrustZone 互換のペリフェラル保護
- セキュアな領域への非セキュア転送のブロックと、非セキュア領域へのセキュアな転送のブロックは違反とみなされます。
- ペリフェラル領域が特権領域として設定されている場合、非特権転送をブロックします。
- 設定可能なエラー応答:
  - Read Zero Write Ignore (RZWI)
  - バス フォールト

15.4.4 PPC アクセス制御ルール

PPC の設定は、主に以下のレジスタによって制御されます。

表 98 PPC レジスタ

レジスタ名	機能
LOCK_MASK	このレジスタは、各 PC のロック機構を提供します。
PC_MASK	このレジスタはペリフェラル領域にアクセスできる PC を示し、各ペリフェラル領域に対して 1 つの PC_MASK があります。
NS_ATT	このレジスタはペリフェラル領域が非セキュアかセキュアかを示します。各ビットはペリフェラル領域に対応します。
S_P_ATT	このレジスタはペリフェラル領域のセキュア特権または非特権ステータスを示し、各ビットはペリフェラル領域に対応します。
NS_P_ATT	このレジスタはペリフェラル領域の非セキュア特権または非特権ステータスを示し、各ビットはペリフェラル領域に対応します。

以下、PPC の制限を記述する際、"i"は PC 値、"j"はペリフェラル領域を表します。

- リセットまたは電源サイクル後、PC_MASK ビットはデフォルトで High に設定され、S_P_ATT および NS_ATT ビットは Low に設定されます。
- PC_MASK[j].PC_MASK[i] = 0x0 かつ LOCK_MASK[i] = 0 の場合、PC「i」は PC_MASK[j]、NS_ATT[j]、NS_P_ATT[j]、S_P_ATT[j]への読み出し/書き込みが可能です。
- LOCK_MASK[i]は、対応する PC「i」からしか書き込めません。PC「i」によってアクセスされ、PC「i」に関連していない LOCK_MASK ビットは RZWI を返します。
- PC_MASK[j].PC_MASK[i] = 1'b1 の場合、PC「i」はペリフェラル領域「j」に Read または Write アクセスできます。
- (LOCK_MASK[i] & PC_MASK[j].PC_MASK[i]) == 1'b1 の場合:
  - ペリフェラル領域「j」セキュリティ属性は変更できません (NS_P_ATT レジスタは例外です)。これは、対応する LOCK_MASK ビットが 1'b1 に設定されているため、PC「i」が「ロック」されていることを意味します。
  - 「ロック」されている PC「i」は、どの PC_MASK にも書き込めません。

- 「ロック」された PC「i」は、ペリフェラル領域への追加アクセスもペリフェラル領域からの削除もできません。
- さらに、任意の PC「k」に対して、PC_MASK[j].PC_MASK[k]が 1'b1 に設定されている場合、PC「k」は、「ロック」されていない PC_MASK をクリアすること、すなわち、他の PC から周辺領域「j」へのアクセスを削除することのみが可能です。PC「k」は、PC「i」が周辺領域「j」を「ロック」しているため、他の PC に追加アクセスを提供できません。

• LOCK_MASK[i]= = 1'b1 かつ PC_MASK[j].PC_MASK[i] = 0x0 の場合:
  - PC「i」は、上記の要件で規定された「ロック」状態に置かれ、周辺領域「j」への読み出しまたは書き込みアクセスを持っていません。PC「i」はペリフェラル領域「j」のセキュリティ属性を変更できませんが、PC「i」は周辺領域「j」のセキュリティ属性への読み出しアクセス権を持っています。

図 58 に、PPC MMIO レジスタの構造を示します。

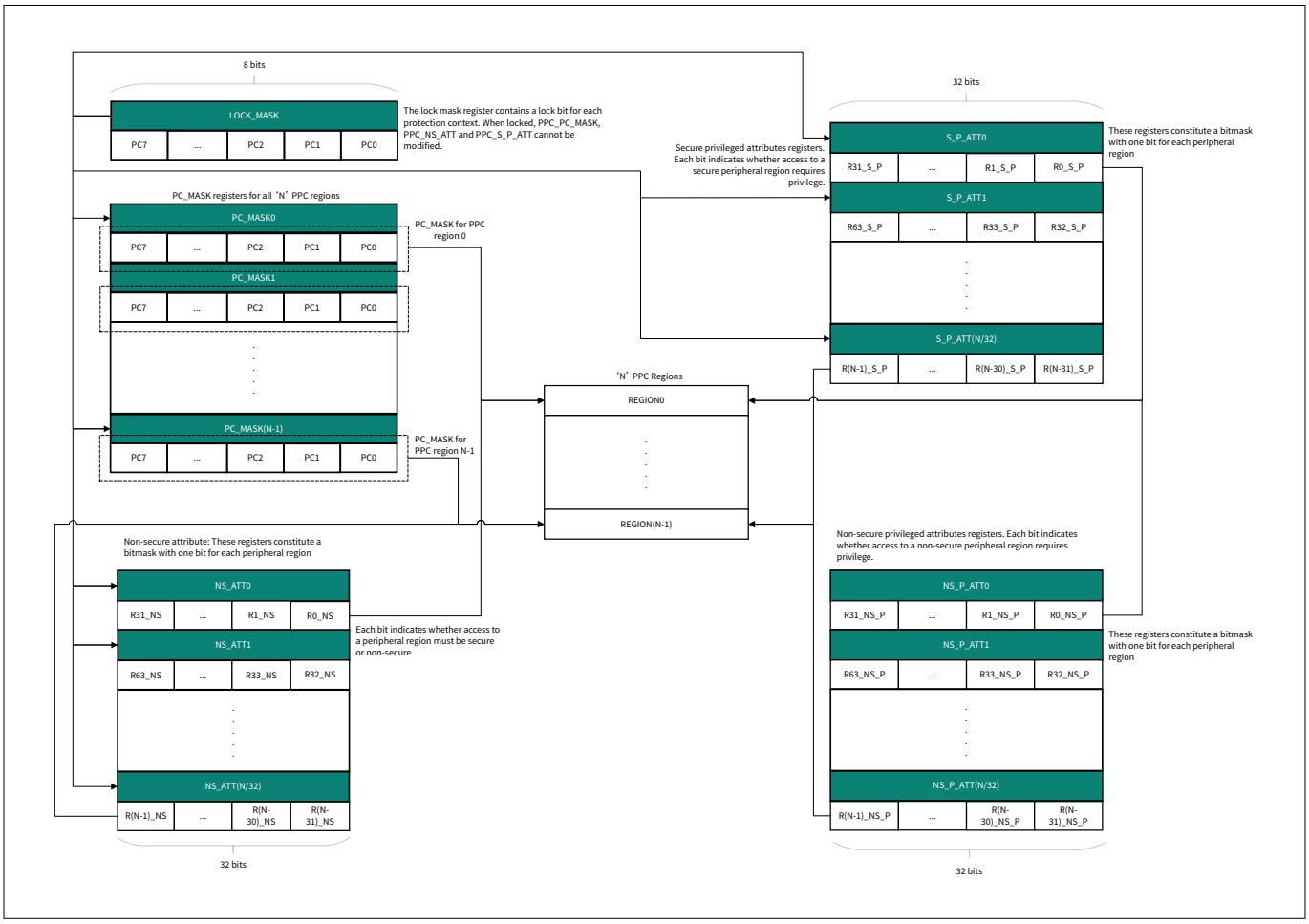


図 58　　PPC MMIO 構造

ROM ブートコードは、PC0 と共有される PPC 領域の一部を設定し、ロックします。PC0 と共有する領域のリストは 表 99 を参照してください。

PPC レジスタの定義とプログラミング モデルについては、PSOC™ Control C3 レジスタ リファレンス マニュアルを参照してください。

15.4.4.1 セキュリティアウェア ペリフェラル

PSOC™ Control C3 では、ペリフェラルはセキュリティアウェアとセキュリティ非アウェアに分類されます。セキュリティ非認識 PPC ペリフェラル領域 (SECURITY_AWARE = 0) は以下の PPC 規則に従います。

- セキュアなトランザクションはセキュアな領域のみにアクセスでき、非セキュアなトランザクションは非セキュアな領域のみにアクセスできます。

セキュリティ対応 PPC ペリフェラル領域 (SECURITY_AWARE = 1) は以下の PPC 規則に従います。

- NS = 0: この領域は、セキュアなトランザクションのみがセキュアなリソースにアクセスできるようにします。
- NS = 1: この領域では、セキュアなトランザクションはセキュアなリソースと非セキュアなリソースの両方にアクセスでき、非セキュアなトランザクションは非セキュアなリソースのみにアクセスできます。したがって、PPC は、他の属性が一致すれば、セキュアなトランザクションも非セキュアなトランザクションも通過できます。

セキュリティ対応ペリフェラルのリストについては、表 99 を参照してください。

15.4.4.2 PPC 領域

PPC はペリフェラルのみでなく、MPC、ペリフェラル インターコネクト、SRSS など、他のブロックの MMIO レジスタも保護します。PPC には全部で 222 の領域があります。ROM ブートコードは NS_ATT、S_P_ATT、NS_P_ATT、PC_MASK レジスタを設定します。

ROM ブートコードは、PPC の全領域の属性を以下のように設定します。

- NS_ATT = 0 - PPC 領域はセキュア
- S_P_ATT= 1 - 4 つの領域を除くすべての領域に設定された特権と非特権アクセス。S_P_ATT は、PPC0_PPC_PPC_SECURE, PPC0_PPC_PPC_NONSECURE, CPUSS_AP, MS_PC0_PRIV の 4 つの領域で 0 に設定されます。
- すべての領域 NS_P_ATT = 1
- PC0 のロックビットを設定します。PC2 ソフトウェアは、他の PC の PPC アクセス許可と属性の設定を担当します。

表 99 に、ブートコードによって設定されたすべての PPC0 領域とその PC_MASK のリストを示します。

表 99 PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
PERI0_MAIN	42000000	4000	0	1, 2, 3, 4, 5, 6, 7
PERI0_GR0_GROUP	42004010	8	0	1, 2, 3, 4, 5, 6, 7
PERI0_GR1_GROUP	42004040	20	0	1, 2, 3, 4, 5, 6, 7
PERI0_GR2_GROUP	42004080	20	0	1, 2, 3, 4, 5, 6, 7
PERI0_GR3_GROUP	420040c0	20	0	1, 2, 3, 4, 5, 6, 7
PERI0_GR4_GROUP	42004100	20	0	1, 2, 3, 4, 5, 6, 7
PERI0_GR5_GROUP	42004150	8	0	1, 2, 3, 4, 5, 6, 7
PERI0_GR0_BOOT	42004020	4	0	0
PERI0_GR1_BOOT	42004060	4	0	0
PERI0_GR2_BOOT	420040a0	4	0	0
PERI0_GR3_BOOT	420040e0	4	0	0
PERI0_GR4_BOOT	42004120	4	0	0

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
PERI0_GR5_BOOT	42004160	4	0	0
PERI0_TR	42008000	8000	0	1, 2, 3, 4, 5, 6, 7
PPC0_PPC_PPC_SECURE	42020000	4000	0	0, 2
PPC0_PPC_PPC_NONSECURE	42024000	4000	0	0, 2
PERI_PCLK0_MAIN	42040000	10000	0	1, 2, 3, 4, 5, 6, 7
CPUSS	42100000	10000	0	1, 2, 3, 4, 5, 6, 7
RAMC0_CM33	42110000	40	0	1, 2, 3, 4, 5, 6, 7
RAMC0_BOOT	42110100	8	0	0
RAMC0_RAM_PWR	42110200	100	0	1, 2, 3, 4, 5, 6, 7
RAMC0_MPC0_PPC_MPC_MAIN	42114000	4	0	0
RAMC0_MPC0_PPC_MPC_PC	42114100	20	0	0
RAMC0_MPC0_PPC_MPC_ROT	42114200	20	0	0
PROMC_CM33	42140000	4	0	1, 2, 3, 4, 5, 6, 7
PROMC_MPC0_PPC_MPC_MAIN	42141000	4	0	0
PROMC_MPC0_PPC_MPC_PC	42141100	20	0	0
PROMC_MPC0_PPC_MPC_ROT	42141200	20	0	0
FLASHC_BOOT	42150000	8	0	2, 3, 4, 5, 6
FLASHC_BOOT1	42150100	20	0	1, 2, 3, 4, 5, 6, 7
FLASHC_MAIN	42150200	10	0	2, 3, 4, 5, 6
FLASHC_DFT	42150400	80	0	0
FLASHC_ECC	42150800	10	0	2, 3, 4, 5, 6
FLASHC_MPC0_PPC_MPC_MAIN	42151000	4	0	0
FLASHC_MPC0_PPC_MPC_PC	42151100	20	0	0
FLASHC_MPC0_PPC_MPC_ROT	42151200	20	0	0

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
FLASHC_FM_CTL_FM_DFT	42152000	4	0	0
FLASHC_FM_CTL_FM_BOOT	42152040	8	0	0
FLASHC_FM_CTL_FM_MAIN	42152800	800	0	2, 3, 4, 5, 6
MXCM33_CM33	42160000	100	0	1, 2, 3, 4, 5, 6, 7
MXCM33_CM33_S	42161000	4	0	1, 2, 3, 4, 5, 6, 7
MXCM33_CM33_NS	42161004	4	0	1, 2, 3, 4, 5, 6, 7
MXCM33_BOOT_PC0	42162000	80	0	0
MXCM33_BOOT_PC1	42162100	4	0	0
MXCM33_BOOT_PC2	42162140	4	0	0
MXCM33_BOOT_PC3	42162180	4	0	0
MXCM33_BOOT	421621c0	4	0	0
MXCM33_CM33_INT	42168000	400	0	1, 2, 3, 4, 5, 6, 7
DW0_DW	42180000	80	0	1, 2, 3, 4, 5, 6, 7
DW1_DW	42190000	80	0	1, 2, 3, 4, 5, 6, 7
DW0_DW_CRC	42180100	80	0	1, 2, 3, 4, 5, 6, 7
DW1_DW_CRC	42190100	80	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT0_CH	42188000	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT1_CH	42188040	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT2_CH	42188080	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT3_CH	421880c0	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT4_CH	42188100	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT5_CH	42188140	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT6_CH	42188180	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT7_CH	421881c0	40	0	1, 2, 3, 4, 5, 6, 7

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
DW0_CH_STRUCT8_CH	42188200	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT9_CH	42188240	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT10_CH	42188280	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT11_CH	421882c0	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT12_CH	42188300	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT13_CH	42188340	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT14_CH	42188380	40	0	1, 2, 3, 4, 5, 6, 7
DW0_CH_STRUCT15_CH	421883c0	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT0_CH	42198000	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT1_CH	42198040	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT2_CH	42198080	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT3_CH	421980c0	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT4_CH	42198100	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT5_CH	42198140	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT6_CH	42198180	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT7_CH	421981c0	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT8_CH	42198200	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT9_CH	42198240	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT10_CH	42198280	40	0	1, 2, 3, 4, 5, 6, 7

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
DW1_CH_STRUCT11_CH	421982c0	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT12_CH	42198300	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT13_CH	42198340	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT14_CH	42198380	40	0	1, 2, 3, 4, 5, 6, 7
DW1_CH_STRUCT15_CH	421983c0	40	0	1, 2, 3, 4, 5, 6, 7
CPUSS_ALL_PC	421c0000	80	0	1, 2, 3, 4, 5, 6, 7
CPUSS_DDFT	421c0080	4	0	0
CPUSS_CM33_S	421c0100	4	0	1, 2, 3, 4, 5, 6, 7
CPUSS_CM33_NS	421c0120	4	0	1, 2, 3, 4, 5, 6, 7
CPUSS_MSC_INT	421c0200	10	0	1, 2, 3, 4, 5, 6, 7
CPUSS_AP	421c1000	4	0	0, 2
CPUSS_BOOT	421c2000	8	0	0
MS0_MAIN	421c4000	4	0	0
MS4_MAIN	421c4040	4	0	0
MS5_MAIN	421c4050	4	0	0
MS7_MAIN	421c4070	4	0	0
MS31_MAIN	421c41f0	4	0	0
MS_PC0_PRIV	421c5000	4	0	0, 1, 2, 3, 4, 5, 6, 7
MS_PC31_PRIV	421c51f0	4	0	1, 2, 3, 4, 5, 6, 7
MS_PC0_PRIV_MIR	421c5004	4	0	1, 2, 3, 4, 5, 6, 7
MS_PC31_PRIV_MIR	421c51f4	4	0	1, 2, 3, 4, 5, 6, 7
MSC_ACG	421c6000	40	0	1, 2, 3, 4, 5, 6, 7
CPUSS_SL_CTL_GROUP	421c8000	8	0	1, 2, 3, 4, 5, 6, 7
IPC_STRUCT0_IPC	421d0000	20	1	1, 2, 3, 4, 5, 6, 7
IPC_STRUCT1_IPC	421d0020	20	1	1, 2, 3, 4, 5, 6, 7
IPC_STRUCT2_IPC	421d0040	20	1	1, 2, 3, 4, 5, 6, 7
IPC_STRUCT3_IPC	421d0060	20	1	1, 2, 3, 4, 5, 6, 7
IPC_INTR_STRUCT0_INTR	421d1000	10	1	1, 2, 3, 4, 5, 6, 7

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
IPC_INTR_STRUCT1_INTR	421d1020	10	1	1, 2, 3, 4, 5, 6, 7
FAULT_STRUCT0_MAIN	421e0000	100	0	1, 2, 3, 4, 5, 6, 7
SRSS_GENERAL	42200000	400	0	1, 2, 3, 4, 5, 6, 7
SRSS_GENERAL2	42200400	40	0	1, 2, 3, 4, 5, 6, 7
SRSS_HIB_DATA	422008a0	10	0	1, 2, 3, 4, 5, 6, 7
SRSS_MAIN	42201000	1000	0	1, 2, 3, 4, 5, 6, 7
SRSS_SECURE	42202000	2000	0	1, 2, 3, 4, 5, 6, 7
RAM_TRIM_SRSS_SRAM	42204000	8	0	1, 2, 3, 4, 5, 6, 7
SRSS_DPLL	42204200	40	0	1, 2, 3, 4, 5, 6, 7
SRSS_WDT	4220c000	10	0	1, 2, 3, 4, 5, 6, 7
MAIN	4220d000	40	0	1, 2, 3, 4, 5, 6, 7
PWRMODE_PWRMODE	42210000	4000	0	1, 2, 3, 4, 5, 6, 7
BACKUP_BACKUP	42220000	100	0	1, 2, 3, 4, 5, 6, 7
BACKUP_B_BREG0	42221000	10	0	1, 2, 3, 4, 5, 6, 7
BACKUP_B_BREG1	42221010	10	0	1, 2, 3, 4, 5, 6, 7
BACKUP_B_BREG2	42221020	20	0	1, 2, 3, 4, 5, 6, 7
BACKUP_B_BREG3	42221080	40	0	1, 2, 3, 4, 5, 6, 7
BACKUP_BACKUP_SECURE	4222ff00	4	0	0
CRYPTOLITE_MAIN	42230000	100	0	1, 2, 3, 4, 5, 6, 7
CRYPTOLITE_TRNG	42230100	100	0	1, 2, 3, 4, 5, 6, 7
MXCORDIC_1_0	42240000	10000	0	1, 2, 3, 4, 5, 6, 7
DEBUG600_DEBUG600	42250000	4	0	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT0_PRT	42400000	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT1_PRT	42400010	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT2_PRT	42400020	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT3_PRT	42400030	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT4_PRT	42400040	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT5_PRT	42400050	8	1	1, 2, 3, 4, 5, 6, 7

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
HSIOM_PRT6_PRT	42400060	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT7_PRT	42400070	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT8_PRT	42400080	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_PRT9_PRT	42400090	8	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT0_SECURE_PRT	42401000	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT1_SECURE_PRT	42401010	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT2_SECURE_PRT	42401020	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT3_SECURE_PRT	42401030	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT4_SECURE_PRT	42401040	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT5_SECURE_PRT	42401050	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT6_SECURE_PRT	42401060	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT7_SECURE_PRT	42401070	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT8_SECURE_PRT	42401080	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_SECURE_PRT9_SECURE_PRT	42401090	4	1	1, 2, 3, 4, 5, 6, 7
HSIOM_AMUX	42402000	10	1	1, 2, 3, 4, 5, 6, 7
HSIOM_MON	42402200	10	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT0_PRT	42410000	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT1_PRT	42410080	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT2_PRT	42410100	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT3_PRT	42410180	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT4_PRT	42410200	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT5_PRT	42410280	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT6_PRT	42410300	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT7_PRT	42410380	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT8_PRT	42410400	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT9_PRT	42410480	40	1	1, 2, 3, 4, 5, 6, 7

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
GPIO_PRT0_CFG	42410040	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT1_CFG	424100c0	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT2_CFG	42410140	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT3_CFG	424101c0	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT4_CFG	42410240	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT5_CFG	424102c0	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT6_CFG	42410340	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT7_CFG	424103c0	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT8_CFG	42410440	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_PRT9_CFG	424104c0	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_SEC_GPIO	42417000	4	1	1, 2, 3, 4, 5, 6, 7
GPIO_GPIO	42418000	40	1	1, 2, 3, 4, 5, 6, 7
GPIO_TEST	42419000	8	1	1, 2, 3, 4, 5, 6, 7
SMARTIO_PRT0_PRT	42420000	100	0	1, 2, 3, 4, 5, 6, 7
SMARTIO_PRT1_PRT	42420100	100	0	1, 2, 3, 4, 5, 6, 7
SMARTIO_PRT2_PRT	42420200	100	0	1, 2, 3, 4, 5, 6, 7
SMARTIO_PRT3_PRT	42420300	100	0	1, 2, 3, 4, 5, 6, 7
SMARTIO_PRT5_PRT	42420500	100	0	1, 2, 3, 4, 5, 6, 7
SMARTIO_PRT6_PRT	42420600	100	0	1, 2, 3, 4, 5, 6, 7
SMARTIO_PRT9_PRT	42420900	100	0	1, 2, 3, 4, 5, 6, 7
LPCOMP	42430000	10000	0	1, 2, 3, 4, 5, 6, 7
DFT	42600000	1000	0	0
EFUSE_CTL1	42610000	4	0	1, 2, 3, 4, 5, 6, 7
EFUSE_CTL2	42610100	80	0	1
EFUSE_CTL3	42610180	4	0	1, 2, 3, 4, 5, 6, 7
EFUSE_DATA_BOOT1	42610800	80	0	0
CANFD0_CH0_CH	42800000	200	0	1, 2, 3, 4, 5, 6, 7
CANFD0_CH1_CH	42800200	200	0	1, 2, 3, 4, 5, 6, 7
CANFD0_MAIN	42801000	40	0	1, 2, 3, 4, 5, 6, 7
CANFD0_BUF	42810000	10000	0	1, 2, 3, 4, 5, 6, 7
SCB0	42820000	10000	0	1, 2, 3, 4, 5, 6, 7
SCB1	42840000	10000	0	1, 2, 3, 4, 5, 6, 7
SCB2	42850000	10000	0	1, 2, 3, 4, 5, 6, 7

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
SCB3	42860000	10000	0	1, 2, 3, 4, 5, 6, 7
SCB4	42870000	10000	0	1, 2, 3, 4, 5, 6, 7
SCB5	42c00000	10000	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP0_CNT0_CNT	42a00000	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP0_CNT1_CNT	42a00100	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP0_CNT2_CNT	42a00200	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP0_CNT3_CNT	42a00300	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT0_CNT	42a10000	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT1_CNT	42a10100	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT2_CNT	42a10200	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT3_CNT	42a10300	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT4_CNT	42a10400	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT5_CNT	42a10500	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT6_CNT	42a10600	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP1_CNT7_CNT	42a10700	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP2_CNT0_CNT	42a20000	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP2_CNT1_CNT	42a20100	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP2_CNT2_CNT	42a20200	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP2_CNT3_CNT	42a20300	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP2_CNT4_CNT	42a20400	100	0	1, 2, 3, 4, 5, 6, 7

(続く)

表 99 (続き) PPC 領域

PPC 領域名	開始アドレス (16 進)	サイズ (16 進)	SECURITY_AWARE	(PCx) によりアクセス可能
TCPWM0_GRP2_CNT5_CNT	42a20500	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP2_CNT6_CNT	42a20600	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_GRP2_CNT7_CNT	42a20700	100	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_TR_ALL_GF_TR_ALL_GF	42a80000	40	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_TR_ALL_SYNC_BYPASS_TR_ALL_SYN_BYPASS	42a90000	4	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_BOOT	42a90800	4	0	1, 2, 3, 4, 5, 6, 7
TCPWM0_MOTIF_GRP1_MOTIF0_MOTIF	42aa4000	200	0	1, 2, 3, 4, 5, 6, 7
MCPASS	42b00000	100000	0	1, 2, 3, 4, 5, 6, 7

***注**: お使いの製品が MOTIF をサポートしていれば、TCPWM0_MOTIF_GRP1_MOTIF0_MOTIF 領域へのアクセスはすべての PC で許可されます。製品が MOTIF をサポートしているかどうかは、製品のデータシートを参照してください。製品が MOTIF をサポートしていない場合、この領域へのアクセスは PC0 に制限されます。*

16 セキュア ブート

セキュア ブートとは、製品のファームウェアを入手し、検証し、起動するプロセスと定義されます。PSOC™ Control C3 MCU の BootROM は ROM_BOOT と FLASH_BOOT で構成されます。リセット後は、書き換え不能な ROM_BOOT が必ず起動し、FLASH_BOOT、インフィニオンおよび OEM のポリシーアセットから成る SFLASH メモリの完全性を検証します。ROM_BOOT が FLASH_BOOT に制御を移した後、事前にプログラムされた FLASH_BOOT が ROM_BOOT の機能を拡張し、次の段階のイメージ (OEM アプリケーション) を認証する前に、必要に応じてフルトリミングを適用します。FLASH_BOOT は、次のステージのアプリケーションまたは Edge Protect ブートローダーを認証して起動します。次の章では、セキュアブートプロセスの各ステージについて詳しく説明します。

16.1 基本的な定義

以下の用語と定義は、本書で使用される用語の理解に役立ちます。これらの用語の多くは、このセクションで詳しく説明します。

- **BootROM**: PSOC™ Control C3 は ROM_BOOT および FLASH_BOOT の 2 つのファームウェア コンポーネントで構成されています。これらのファームウェア コンポーネントを総称して、イミュータブル (書き換え不可能な) BootROM と呼びます。
- **信頼の連鎖 (Chain of Trust: CoT)**: 信頼の基点 (The root of trust) は、ROM に格納されているインフィニオンのコードから開始され、これの変更はできません。信頼の連鎖は、信頼の基点 (RoT) である ROM_BOOT から始まるソフトウェアのブロックを検証することで確立されます。
- **デバッグ アクセスポート (DAP)**: 書き込みおよびデバッグ用の外部デバッガ/プログラマと PSOC™ Control C3 MCU 間のインターフェース。
- **ダイジェスト**: ダイジェストは、アプリケーションのバイナリのようなデータセットにハッシュ機能を適用して計算されます。
- **デジタル署名**: ダイジェスト (データセットのハッシュ) の暗号化。一例に、ユーザーアプリケーションの暗号化されたハッシュがあります。
- **デバイス整合性試験**: プロビジョニングの前に、このデバイスが本物で、既知の状態であることを確認するために使用されるテスト。
- **ハッシュ (Hash):** 所定のデータブロックに対し繰り返し可能な一意の署名を生成する暗号アルゴリズムです。これは不可逆的です。
- **ハードウェアセキュリティモジュール (HSM)**: デジタル キーを保護 管理する物理的なコンピューティング デバイスまたはプログラミング デバイスで、デジタル ソフトウェア署名や暗号化、その他の暗号化機能を実行できます。
- **ハードウェア固有キー (HUK):** すべてのデバイス固有鍵を導き出すための 256 ビットのデバイス ルート鍵
- **プロセス間通信 (IPC):** セキュアメモリと非セキュアメモリ間の通信を促進するために使用される IPC ハードウェア
- **ライフサイクルステージ (LCS):** LCS はデバイスが動作しているセキュリティモード。
- **メモリ保護コントローラー (MPC):** PC によってメモリの一部を分離することが可能
- **マスター セキュリティ コントローラー (MSC):** バス上で開始されたトランザクションの保護に役立ちます
- **不揮発性カウンター (NV カウンター):** ロールバック防止を提供するセキュリティ カウンター
- **OTP**: ワンタイムプログラマブルメモリ
- **保護コンテキスト (Protection Context: PC):** このパラメーターは、各バス マスターに'0'から'7'までのセキュリティ状態 レベルを許可します。バス マスターには PC 値を割り当てることができ、その値は固定されることも、アプリケーション実行中に変更されることもあります。PC はメモリ制限をより正確に適用する方法を提供します。PC の状態は保護ユニットと連動します
- **公開鍵暗号 (Public-Key Cryptography: PKC):** 非対称暗号とも呼ばれ、セキュアなデータ通信のために公開鍵と秘密鍵のペア (または非対称鍵) アルゴリズムを使用する暗号化技術です。セキュアなメッセージま

たはデータブロックにするために使用します。秘密鍵はデータの復号化に使用され、セキュアに保管する必要があります。一方、公開鍵はデータの暗号化に使用され、広く配布できます。
  - **公開鍵 (Public Key):** 公開鍵は共有できますが、変更できないように認証または保護する必要があります。
  - **秘密鍵 (Private Key):** 秘密鍵はセキュアな場所に保管し、公開または窃取されないようにする必要があります。ペアとなる公開鍵を使用して複合化されるデータのブロックを、暗号化するために使用します。
- **ポリシー:** ポリシーは、ポリシーが適用されるデバイス上で何が許可され、何が許可されないかを記述する、定義済みの (名称、値) ペアの集まりです。
- **ペリフェラル保護コントローラー (PPC):** RoT によって割り当てられた設定済みの PC でのみアクセスできるようにペリフェラルを設定できます。
- **プロビジョニング:** 鍵、ポリシー、秘密がデバイスに注入されるプロセス。一旦プロビジョニングされると、関連するポリシーに従って注入されたキーでのみ、デバイスにアクセスまたは変更できます。
- **プラットフォーム セキュリティ アーキテクチャ (PSA):** ファームウェアのセキュリティ仕様とオープンソースのリファレンス実装により、製品にベストプラクティスのセキュリティを組み込むことを支援します
- **RMA:** 返品承認
- **SESIP:** IoT (モノのインターネット) プラットフォームのセキュリティ評価基準
- **SHA-256:** データまたはコードのブロックダイジェストを作成するために使用する暗号化ハッシュアルゴリズムです。データブロックのサイズに関係なく、一意の 256 ビットダイジェストを生成します。

16.2 機能

PSOC™ Control C3 MCU は以下のセキュア ブート機能を提供します。

- 各ブートステージは、次のステージのイメージが起動される前に検証します。したがって、最初のユーザーアプリケーションを起動するまでの信頼の連鎖を提供します。
- あらかじめプログラムされたマルチステージ BootROM (ROM_BOOT と FLASH_BOOT ファームウェアで設定) により、不変のトラスト アンカーを持つハードウェア ルート オブ トラストを実装しています。
- デバイスを製品化するために、従来の開発 LCS から生産用 LCS への移行を行います。
- プロビジョニング時に、必要な OEM のキーやその他のユーザー キーをロードできる、変更不可のキー ストレージ エリアを内蔵。
- デュアルバンク対応。デバイス ユーザー フラッシュは、バンク 0 とバンク 1 と呼ばれる 2 つの等しいサイズのバンク (各 128 KB) に分割されています。この機能は、プロビジョニングによって有効/無効にします。
- プロテクトされたファームウェアに対応。この機能により、お客様はファームウェア IP を保護できます。
- セキュアなデバッグと、インフィニオンからユーザーへのセキュアなプロビジョニングをサポートします。
- デバイスのプロビジョニングとファームウェアのアップグレードは、SWD インターフェースまたは BootROM DFU シリアルインターフェースのいずれかを介して実行できます。

16.3 アーキテクチャ

16.3.1 ポリシーと鍵

16.3.1.1 ポリシー

- **IFX_POLICY:** インフィニオンの工場では、IFX_POLICY が規定されています。これは、あらかじめプログラムされたインフィニオンのアセットと、セキュアにブート可能でアップグレード可能なあらかじめプログラムされたソフトウェアで構成されています。これらは工場でプログラムされ、インフィニオンによってロックされます。
- **PROT_FW_POLICY:** PROT_FW_POLICY は、PROT_FW_ROT_KEYS、MPN_ROT_KEY_REVOCATION、および保護されたファームウェア属性で構成されます。PROT_FW_POLICY は開発 LCS に用意されています。
- **OEM_POLICY:** OEM は開発 LCS の OEM_POLICY を規定します。ポリシーは、デバイスのライフサイクル状態 (LCS)、デバッグ ポリシー、ブート ポリシー、dfu ポリシー、OEM_ROT_KEYS、およびその他のアセットで構成されます。ポリシーは"prov_oem_complete"フラグが"true"に設定されるまで何度でも変更できます。

16.3.1.2 鍵

ここで説明されている鍵は、非対称 ECC 256 ビット ECDSA 鍵ペアです。各ペアには、秘密鍵と公開鍵の両方が含まれます。

- **IFX_ROT_KEY:** IFX_ROT_KEY は、初期製造時にインフィニオンが署名し、インストールしたアセットを検証するために使用される非対称 ECC 256 ビット ECDSA キー ペアです。インフィニオンは、IFX_ROT_PRIVATE_KEY を内部制御 HSM にセキュアに格納し、IFX_ROT_PUBLIC_KEY をデバイス OTP に格納します。RAM アプリケーションは、保存されている IFX_ROT_PRIVATE_KEY を使用して署名され、デバイスに保存されている対応する IFX_ROT_PUBLIC_KEY を使用して検証されます。
- **PROT_FW_ROT_KEY:** PROT_FW_ROT_KEY は、プロテクトされたファームウェアを認証するために使用されます。このキーを取り消すには、PROT_FW_ROT_KEY_REVOCATION を使用します。
- **OEM_ROT_KEY:** OEM_ROT_KEY は、OEM のアプリケーション イメージの認証に使用されます。デフォルトでは、OEM_ROT_KEY0 がイメージの認証に使用されます。必要であれば、OEM はキーを失効させて OEM_ROT_KEY1 を使用できます。OEM は、プロビジョニングを通じて一度のみキーを取り消すことができ、元に戻せません。

16.3.2 ライフサイクルステージ (LCS)

PSOC™ Control C3 MCU は設定可能な不揮発性のライフサイクル ステージを持ち、デバイス セキュリティにおいて重要な役割を果たします。これらのステージは、e ヒューズを飛ばすことによって制御される厳密な一方通行の進行に従っており、一度ステージが完了すると、元に戻せません。例えば、デバイスが開発段階から生産段階に移行すると、開発段階に戻せません。その代わり、OEM の決定に従って、生産にとどまるか、返品承認 (RMA) の段階に進めます。

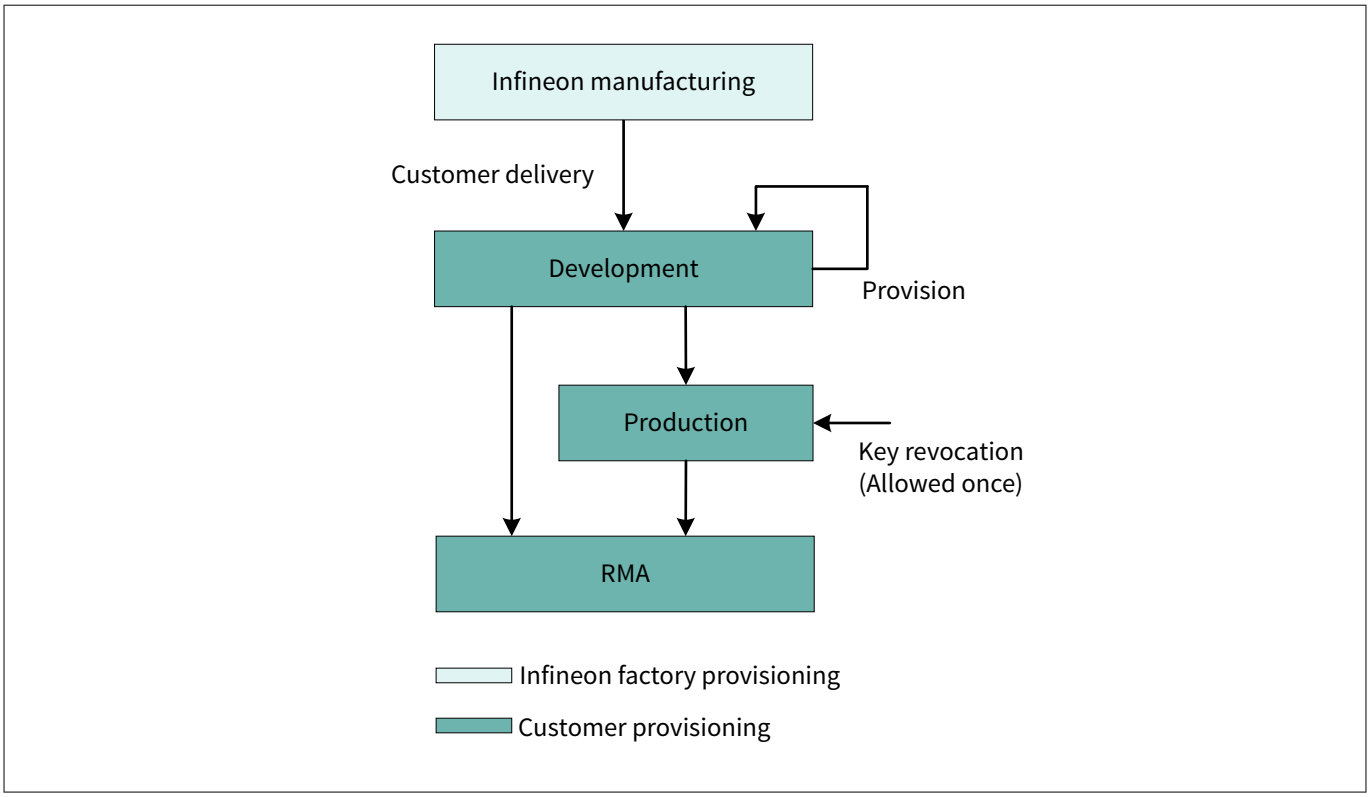


図 59　　ライフサイクルの段階

16.3.2.1　IFX_MFG

インフィニオン ファブでは、製品は IFX_MFG LCS で生産、調整、プログラムされます。必要なテストとトリミングはすべてこの LCS で行われ、インフィニオンのアセットはデバイスにロードされます。デバイスは、インフィニオンの工場から出荷される前に開発 LCS に移行されます。

16.3.2.2　開発 LCS

PSOC™ Control C3 MCU デバイスは、開発 LCS で顧客に出荷されます。デバイスがこの状態になる前に、すべてのトリミングと工場テストが完了しており、デバイスはこの LCS でインフィニオンから出荷されます。顧客は、開発 LCS 中に OEM のアセットをプロビジョニングし、エンド アプリケーションを開発する必要があります。この段階で、デバイスは何度でも再プロビジョニングでき、OEM_ROT_KEY とアセットは OEM のプロビジョニングによって必要に応じて交換できます。

***注**:　　顧客が開発 LCS でデバイスの再プロビジョニングを希望する場合、prov_oem_complete パラメーター値を"true"に設定し、システムの"ap"値を"Permanently Disable"に設定しないでください。詳細については、AN240106 Getting started with PSOC™ Control C3 security を参照してください。*

16.3.2.3　プロダクション LCS

これは、保護されたデバイスの現場での運用 LCS です。このフェーズに進む前に、署名済みの有効なアプリケーション コードがプログラムされ、OEM のキーの HASH がプロビジョニングされている必要があります (OEM_ROT_KEYS)。この LCS では、アプリケーションの認証に成功した場合のみ、デバイスが起動します。いずれかの段階で認証に失敗した場合、次のアプリケーションは起動せず、デバイスのブートプロセスはここで終了します。

プロダクション LCS の移行中、プロビジョニング RAM アプリケーションは以下を実行します。

- 規定 OEM_ROT_KEYS
- すべてのポリシーが認可されたプロダクション キーでプロビジョニングされていることを確認します。
- OTP に SECURE_HASH を書き込みます。16 バイトの SHA-256 SECURE_HASH は、OEM_ROT_KEY と OEM_POLICY のハッシュで構成されます。
- OTP ビットを書き込み、本番状態に不可逆的に移行します。

  本番 LCS に入る前に、デバッグ アクセス制限を定義する必要があります。例えば、デバッグ ポリシーを設定して、この LCS のデバッグを永久に無効に設定できます。代わりに、この LCS でデバッグ機能が必要で、デバイスがセキュアなデバッグ証明書でデバッグできる場合、デバッグ ポリシーを無効に設定することもできます。インフィニオンは、プロダクション LCS ではデバッグ ポートを閉じておくことを推奨します。

16.3.2.4 RMA

顧客がインフィニオンに故障解析 (FA) の実施を要求した場合、デバイスは RMA ライフサイクルに移行できます。デバイスは、OEM_ROT_KEY で署名された証明書を用いて `TransitionToRMA RAMApp` を実行することで、RMA モードに移行できます。お客様は、`RAMApp` を起動し、RMA LCS 状態に移行する前に、デバイス上のすべての OEM 所有アセットを消去する必要があります。デバイスが RMA ライフサイクル ステージに移行すると、デバイスのプロビジョニングやプログラミングはできなくなります。デバイスが RMA モードに移行した後、ブートすると、ユーザー フラッシュ内のコードを実行しようとしません。代わりに、インフィニオンによって署名されたデバッグ ポートを介して `OpenRMA` トークンを待ち、デバイスの内部デバッグと解析をさらに実行できるようにします。

16.3.3 ブートシーケンス

セキュア ブートは、業界固有の標準によって定義された公開鍵暗号セキュリティ プロトコルを使用して、アプリケーション イメージを認証します。ROM_BOOT はハードウェアのトリミングと初期化を行います。FLASH_BOOT の実行にジャンプする前に、FLASH_BOOT を検証します。FLASH_BOOT は ROM_BOOT の機能を拡張し、IFX および OEM のポリシーを適用し、検証し、エッジ プロテクト ブートローダー (オプション) または OEM のアプリケーションを起動します。エッジ プロテクト ブートローダは最初の OEM/カスタマー アプリケーションを認証し、図 60 に示すように起動します。これらの BootROM とエッジ プロテクト ブートローダは、デバイスに格納されたデータとコードを認証するために広範囲にハッシュを使用します。OEM は、表 100 に記すように、OEM のポリシーで定義された BOOT_CFG_ID に基づいて複数のイメージを持てます。

表 100 ブート設定

BOOT_CFG_ID	イメージオプション
シンプルなアプリケーション設定	
BOOT_SIMPLE_APP	• デフォルトでは、デバイスは BOOT_SIMLPE_APP 設定に設定されています。 • ユーザー フラッシュには、署名なし RAW イメージが必要です。 • イメージはフラッシュ アドレスの先頭に配置します。 • イメージは署名されていないため、この設定はプロダクション LCS をサポートしません。

(続く)

表 100 (続き) ブート設定

BOOT_CFG_ID	イメージオプション
デバイスがパワーサイクル/リセットされるたびに、BootROM は OEM のイメージを起動する前に認証します。認証を成功させるためには、イメージは OEM_ROT_KEY で署名され、かつ MCUboot フォーマットでなければいけません。BOOT_ONE_SLOT、BOOT_TWO_SLOTS、BOOT_THREE_SLOTS はセキュア ブートをサポートします。以下は、これらの設定に共通する属性です。 • これらの設定は、開発用およびプロダクション用の LCS をサポートし、イメージは署名され、MCUboot フォーマットをサポートする必要があります。 • セキュアなデバッグと RMA 移行に対応 • 最初のユーザー イメージがエッジ プロテクト ブートローダである、または同等の場合、さらなるユーザー イメージの認証は、エッジ プロテクト ブートローダまたは同等の方法の責任に任されます。	
セキュアブート設定	
BOOT_ONE_SLOT	• BootROM によって認証されるイメージは 1 つのみです。 • "boot_cfg_id"コンフィギュレーションは、BOOT_ONE_SLOT として設定してください。 • "boot_app_layout"と"address"と"size"は、`policy_oem_provisioning.json` ファイルに記述し、コードを実行する前にデバイスをプロビジョニングする必要があります。
BOOT_TWO_SLOTS	• 2 つのイメージをユーザーフラッシュに配置できます。1 つはプライマリ、もう 1 つはセカンダリまたはステージングスロットに配置します。 • セカンダリスロットは、BootROM DFU またはユーザー DFU MW ベースのシリアルインターフェースによるイメージのアップグレードに使用されます。 • イメージの"アドレス"と"サイズ"の両方を、`policy_oem_provisioning.json` ファイルの"boot_app_layout"に記入し、コードを実行する前にデバイスをプロビジョニングする必要があります。
BOOT_THREE_SLOTS	• 3 つのイメージをユーザーフラッシュに置くことができ、すべてのイメージは BootROM によって認証されます。 • 3 つのイメージの"アドレス"と"サイズ"はすべて、`policy_oem_provisioning.json` ファイルの"boot_app_layout"に記入し、コードを実行する前にデバイスをプロビジョニングする必要があります。

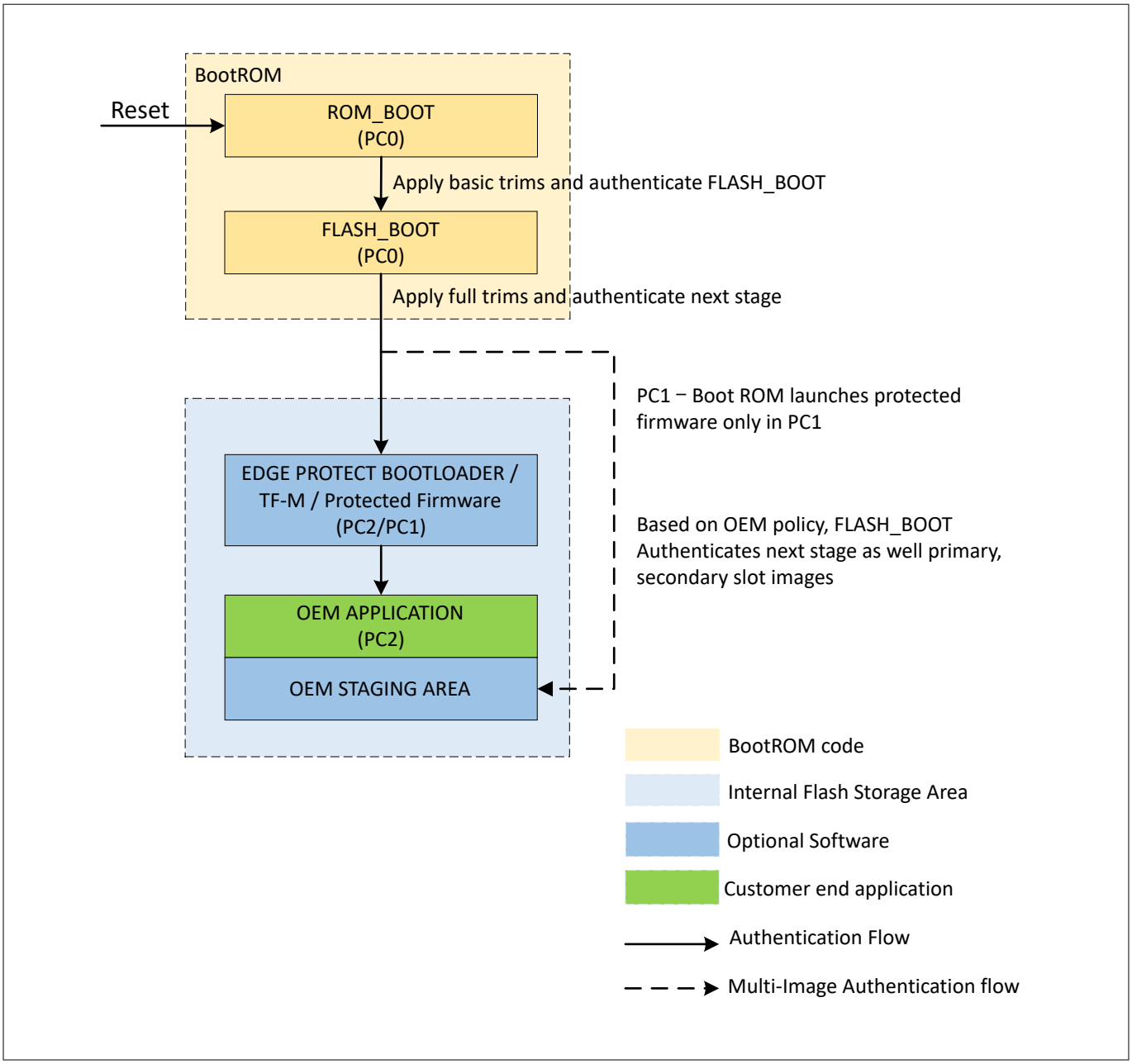


図 60　　セキュア ブートフロー

16.3.3.1　セキュア ブート

BootROM は、安全なシステムを提供するために必要な基本的なセキュリティ機能を提供します。デバイスが開発 LCS でインフィニオンから出荷される時には、ポリシーで「セキュア ブート」が無効化されており、BootROM が検証されずに OEM のアプリケーションをブートします。「セキュアブート」は、OEM のアセットをプロビジョニングし、MCUboot フォーマットで OEM のアプリケーションイメージをプログラミングすることにより、開発 LCS で有効に設定できます。あるいは、デバイスがプロダクション LCS に移行すると、「セキュア ブート」が自動的に有効になります。

ポリシーで「セキュア ブート」が有効になっている場合、BootROM は OEM_ROT_KEY_PUBLIC を使用して最初のアプリケーションを検証し、BootROM と最初の OEM のアプリケーションの間に信頼の連鎖を作成して、最高レベルのセキュリティを提供します。

16.3.3.2 保護状態

ライフサイクルステージは、ワンタイムプログラム (OTP) eFuse ビットの書き込みによって指示される厳密な不可逆的な進行です。ブートが発生するたびに、デバイスの保護設定 (MPC と PPC) がブートプロセスによって適用されます。ハードウェア初期化エラーやインフィニオンによりプロビジョニングされたアセットの破損や不一致検出などの重大なエラーがブートによって発見された場合、デバイスは破損状態に入ります。

セキュア ブートが有効になっていて、最初のユーザー アプリケーション イメージの認証に失敗した場合、デバイスは「デッド ブランチ」状態になります。この状態から回復するためには、開発ライフサイクルの段階で、正しい OEM のアセットでデバイスを再プロビジョニングする必要があります。

デバイスがデバッグ トークンの検証に失敗した場合、そのデバイスはデバッグ ポートへのアクセスを許可しません。デバッグポリシーを設定するためには、OEM のプロビジョニングポリシーのデバッグセクションを変更する必要があります。

16.3.4 他のシリアル インターフェース

BootROM は、製造工程における各種作業 (プロビジョニングや保護されているファームウェアのダウンロード、顧客アプリケーションのダウンロードなど) を実行するための簡易シリアルインターフェースを提供します。これらは、インフィニオン独自の簡易デバイスファームウェア更新 (DFU) プロトコルを介したシリアル通信で行われます。

デフォルトの設定では、デバイスファームウェア更新 (DFU) 機能は OEM ポリシーで有効化されていません。この機能を有効化するためには、ユーザーがシリアル ワイヤ デバッグ (SWD) インターフェースを使用する必要があります。

表 101 に示すように、ポート P2.2 および P2.3 を介して異なるシリアルインターフェースを選択でき、シリアルインターフェース ピン テーブルに従ってホスト (DFU ホストツール) と PSOC™ C3 MCU のターゲット間での通信を確立します。

表 101 DFU シリアル インターフェース選択

DFU シリアル インターフェース選択	P2.3 = Vss	P2.2 = Vdd	P2.3 = High-Z
P2.2 = Vss	シリアル無効	UART	シリアル無効
P2.2 = Vdd	I2C	SPI	シリアル無効
P2.2 = High-Z	シリアル無効	シリアル無効	シリアル無効

***注**: DFU 選択ピン (P2.2、P2.3) をハイまたはローに設定するためには、1 kΩ 以下の外部プルアップ抵抗かプルダウン抵抗を使用してください。*

以下のデフォルト設定が DFU 通信プロトコルに適用されます。

- SPI: モード = Slave, Motorola 社製 00 (CPHA = 0、CPOL = 0)、速度 12 Mbps 以上
- UART: ボーレート = 115200、ビット = 8、ストップビット = 1、RTS/CTS=無効、パリティ=無効
- 速度 = 400 kHz、モード = Slave、7 ビット アドレス、アドレス = 0x35

表 102 シリアル インターフェース ピン

ピン	UART	SPI	I2C
P8.0	-	SS	-
P8.1	Rx	MOSI	SCL
P8.2	-	MISO	-

(続く)

表 102 (続き) シリアル インターフェース ピン

ピン	UART	SPI	I2C
P8.3	Tx	CLK	SDA

BootROM DFU バッファのサイズ制限により、通信に使用される最大パケットサイズが制限されます。

下記の制限が各プロトコルに適用されます。

SPI, UART:

- 入力バッファサイズ (ホスト書き込み) は 512 バイトのペイロード + プロトコルのオーバーヘッド 7 バイト
- 出力バッファサイズ (ホスト読み出し) は 128 バイト + 7 バイト

I2C:

- 入力バッファサイズ (ホスト書き込み) は 256 バイト + 7 バイト
- 出力バッファサイズ (ホスト読み出し) は 128 バイト + 7 バイト

16.3.5 セキュアなプロビジョニングと再プロビジョニング

プロビジョニングとは、認可された鍵、証明書、認証情報、およびファームウェア イメージのセットでデバイスを設定することです。プロビジョニングは、デバイス SRAM から実行されるように構築されたプロビジョニング アプリケーション (RAM アプリケーション) を使用して実行されます。これらのアプリケーションは、インフィニオンと OEM のアセットを OTP、SFLASH にプログラムし、デバイスを開発またはプロダクション LCS に進めます。デバイスは、これらの LCS でユーザー アプリケーションを起動する準備ができます。

開発 LCS では、開発段階で OEM のアセットを何度でも再プロビジョニングできます。プロダクション LCS は、OEM のアセットの再プロビジョニングを許可せず、一度だけ OEM_ROT_KEY キーの失効を許可します。

16.3.6 セキュア ブートで使用される保護コンテキスト

保護コンテキスト (PC) は、ハードウェアの特定のブロックではありません。ただし、MPU、MPC、PPC など、すべての保護ユニットに使用されます。複数の保護コンテキストは、PSOC™ Control C3 デバイス内の異なるセキュリティ レベルを分離するために使用されます。セキュア ブートプロセスでは、図 60 と表 96 に示すように、PC0、PC1、PC2 を使用します。

17 電源と電圧監視

PSOC™ Control C3 MCU は、1.71 V～3.6 V の範囲で単一の安定した電源 VDDD を使用して動作します。さらに、1.4 V～3.6 V の範囲で使用できるオプションの VBACKUP 電源もあります。複数の LDO レギュレーターを統合し、様々な電力モードにおいてデバイス内のブロックに電力を供給します。

このデバイスは、VDDD、VDDA、VDDIOs、VBACKUP などの複数の電源レールに対応しており、アプリケーションがデバイス内の異なるブロックに対して個別の電源を使用できます。例えば、VDDA は ADC や CSG などのアナログ ペリフェラルに電力を供給します。PSOC™ Control C3 MCU ファミリーは、電源監視と故障保護を目的として、パワー オン リセット (POR)、ブラウンアウト検出 (BOD)、低電圧検出 (LVD) 回路に対応しています。

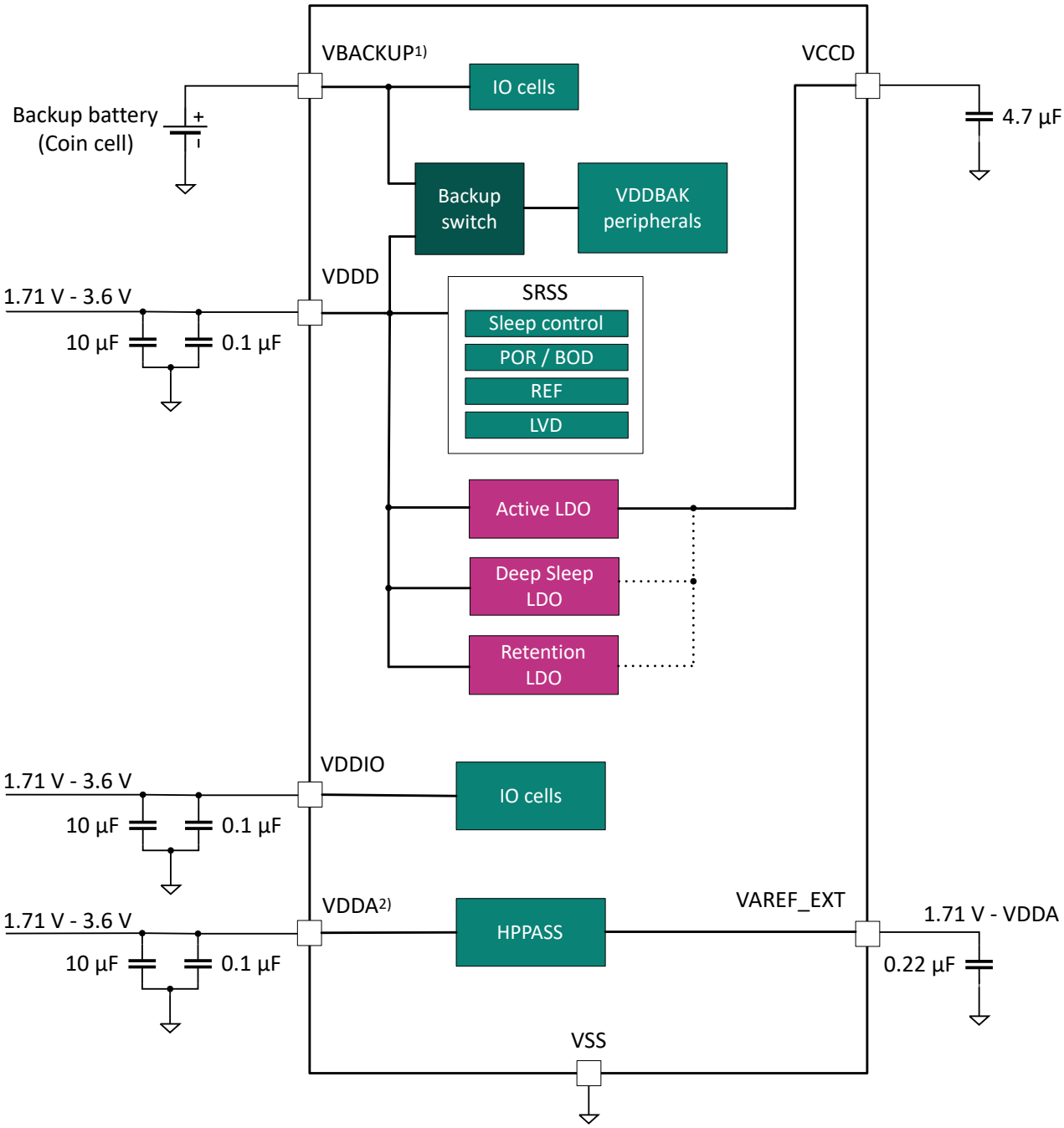
17.1 機能

PSOC™ Control C3 MCU の電源と電圧監視サブシステムは、以下の機能に対応しています。

- 動作電圧範囲
  - VDDD: 1.71 V～3.6 V
  - VBACKUP: 1.4 V～3.6 V
- ユーザーが選択可能なコアロジック動作
  - 超低消費電力 (ULP): コアロジックは 0.9 V で動作
  - 中間周波数 (MF): コアロジックは 1.0 V で動作
  - 低消費電力 (LP): コアロジックは 1.1 V で動作
  - オーバードライブ (OD): コアロジックは 1.2 V で動作
- 複数の独立した電源レール
  - VDDD
  - VDDA
  - 複数の VDDIO
  - VBACKUP
- 複数のオンチップ リニアレギュレーター
  - 1 つのアクティブ LDO
  - 異なる電力モードで動作するペリフェラルに電力を供給する複数の低消費電力レギュレーター
- VDDD と VCCD をモニターする 2 つの POR/BOD 回路
- VDDD を監視する低電圧検出 (LVD) 回路×1

***注**: 使用可能な電源およびユーザーが選択可能なコアロジックの動作電圧については、デバイスのデータシートを参照してください。*

17.2 アーキテクチャ



1) VBACKUP can be connected to a coin cell (1.4 V to 3.6 V) or a supercapacitor, depending on the application. If not used, it should be connected to VDDD.
2) VDDA should be ≥ 2.7 V to run HPPASS at the highest supported clock speed of 240 MHz.

図 61 電源と電圧監視システムのブロック図

図 61 に、すべての電源ピンに単一の安定した電源を使用し、VBACKUP をコイン電池に接続した典型的な電源設定を示します。例えば、アナログ回路を絶縁するために、VDDA を 1.71 V～3.6 V の独立した安定した電源に接続できます。

図 61 に示したレギュレーターと電源ピン/レールは、デバイス内部のさまざまなブロックに電力を供給します。アプリケーションの電源レール/ピンの利用可能性については、選択されたデバイス パッケージによって異なります。詳細は、デバイス データシートを参照してください。

すべてのコア レギュレーターは、VDDD 電源端子から電力を取り込みます。アクティブ LDO によって生成される VCCD 電源は、アクティブ/スリープ モードのすべてのアクティブ ドメインペリフェラルに電力を供給します。ディープスリープ LDO はディープスリープペリフェラルに電力を供給し、リテンション LDO はディープスリープおよびディープスリープ-RAM 電力モードで SRAM のリテンション ロジックに電力を供給します。ハイバネート モードでは、すべての内部レギュレーターがオフになり、LPComp やスリープ コントロールなどのハイバネート ペリフェラルは、VDDD 電源から直接給電されます。

I/O セルは、それらが配置されているポートに応じて、様々な VDDIOx ピンから動作します。VCCD 電源は、コアロジックペリフェラルから I/O セル内のロジックを駆動します。VDDA は HPPASS (SAR ADC および CSG) と I/O セル内のアナログ マルチプレクサ スイッチなどのアナログ ロジックに電力を供給します。どの I/O がどの電源で動作するかについては、デバイスのデータシートを参照してください。

電源レールとレギュレーターに加えて、このデバイスは電源レールをモニターし、可能性のある電源障害から保護するためのオプションを提供します。これには POR/BOD 回路と LVD 回路が含まれます。スリープ コントロール ブロックは、低消費電力モードへの移行と終了に必要な回路を提供します。

17.3 電源

このデバイスには、様々な電力モードでペリフェラルに電力を供給するために、以下のコアレギュレーターが搭載されています。

17.3.1 アクティブ レギュレーター

このデバイスには、アクティブ モードとスリープ モードのペリフェラルに電力を供給するためのリニア LDO レギュレーターが搭載されています。このアクティブ リニア レギュレーターは、ペリフェラルのアクティブ モード動作に必要なコア電圧 (VCCD) を VDDD から生成します。レギュレーターは、コア動作用に 0.9 V、1.0 V、1.1 V、1.2 V[1)] を供給できます。アクティブ レギュレーターの出力電圧を変更する場合、特に高い電圧に変更する場合は、安定するまでに時間がかかります。コンフィギュレーション レジスタに直接書き込んでコア電圧を変更する場合は、この点を考慮する必要があります。PDL 機能には安定化遅延が含まれます。

レギュレーターは、アクティブとスリープ モードで使用可能です。このレギュレーターは 2 つのサブモードを実装しています。

- 大電流モード
- 低電流モード

大電流モードは通常の動作モードです。つまり、アクティブ電力モードでは、デバイスは全能力で動作します。低電流モードでは、レギュレーターから出力される電流とその静止電流が制限されます。このモードは、LPActive、スリープ、および LPSleep 電力モードを実現します。低電流モードは、スリープ モードにおけるリソースの能力と利用可能性に制限を設けます。例えば、LPActive サブモードと LPSleep サブモードの場合、サポートされる最大クロック周波数は制限されます。詳細については、デバイス電力モードを参照してください。このため、電源投入時や、他の動作モードからアクティブ モードに移行する際には、常にアクティブ LDO がオンになります。アクティブおよびスリープ以外の電力モードでは、アクティブ リニア レギュレーターはオフになります。

17.3.2 ディープスリープ レギュレーター

ディープスリープ レギュレーターは、ディープスリープ モードで使用可能なペリフェラルに電力を供給します。このレギュレーターは、低消費電力モードでのデバイスの動作電流を低減するため、低静止電流動作に最適化されています。

1 利用可能なコア電圧については、デバイスのデータシートを参照してください

17.3.3 リテンション レギュレーター

リテンション LDO は、ディープスリープ モード時にリテンション フロップと SRAM コアに低電圧を供給することで、ディープスリープ電流を低減します。電源電圧が低いため、リーク電流を最小限に抑えながら、最低保持電圧の要件を満たせます。

17.3.4 バックアップドメイン

バックアップ ドメインは独立した電源ドメインです。通常動作時、VDDD 電源はバックアップドメイン ペリフェラルに電力を供給します。VDDD 供給がない場合、バックアップ スイッチはノーマル クローズ ポジションになり、VBACKUP 供給入力を RTC などのバックアップ ドメイン ペリフェラルの電源として利用します。バックアップ機能が不要な場合は、コイン電池やスーパーキャパシターなどのバックアップ電源を VBACKUP ピンに接続するか、外部で VDDD に接続できます。

17.4 SRSS リファレンスブロック

SRSS リファレンス ブロックには、0.8 V のバンド ギャップ リファレンス (BGR) 電圧発生器と 250 nA の電流リファレンスの 2 つのインスタンスが含まれます。これらは内部レギュレーター、電圧モニター、PLL などで使用されます。アクティブ/スリープ モード時に使用可能なアクティブ BGR は 20 uA の電流を消費し、ディープスリープ モード時に使用可能なディープスリープ BGR はわずか 0.2 uA の電流しか消費しません。

17.5 電圧監視

PSOC™ Control シリーズ MCU は、複数の電圧監視および電源障害保護オプションを提供します。これには POR、BOD、LVD が含まれます。

17.5.1 パワーオンリセット (POR)

POR 回路は初期電圧上昇時にリセットパルスを提供します。POR 回路は VDDD および VCCD 電圧を監視します。POR 回路は電源投入時に使用され、以後無効になります。POR トリップポイント レベルの詳細については、デバイスのデータシートを参照してください。

17.5.2 ブラウンアウト検出 (BOD)

BOD 回路は、デバイスをリセットすることで、動作中またはデータ保持中の回路を安全ではない電源供給条件から保護します。BOD は VDDD 電圧を監視し、電圧変動が安全動作に必要な最小 VDDD 電圧を下回った場合にリセットを生成します (詳細はデバイスデータシートを参照)。電源が有効なことが再び検出されるまで、システムは RESET から復帰しません。BOD 回路は、システムがクラッシュする前に、アクティブ、スリープ、ディープスリープ-RAM、ディープスリープ オフの各電力モードでのリセットを保証します。

ハイバネートモードではオンチップ BOD に対応していません。ハイバネート中に BOD サポートを必要とするアプリケーションは、外部スーパーバイザ回路を使用する必要があります。BOD 用の設定レジスタはありません。BOD トリップ ポイント、サポートされる供給ランプ レート、およびブラウンアウト検出応答時間の詳細については、デバイスのデータシートを参照してください。

17.5.3 低電圧検出 (LVD)

LVD 回路は外部電源電圧を監視し、デバイスがまだ動作限界内にある間に割込みまたはフォールトを発生させます。ファームウェアは、LVD 割込みを利用して、事前に対策を講じて予防できます。PWR_LVD_CTL レジスタを使用して、LVD のトリップポイントをカスタマイズし調整します。PSOC™ Control C3 MCU は、VDDD レールに 1 つの LVD を内蔵しています。アクティブ/スリープモード中はアクティブで機能します。ディープスリープおよびハイバネ

ート モードでは LVD は使用できません。デフォルトでは、LVD 動作は電力を消費するため、節電のために無効になっています。LVD がリセット ソースとして機能しない代わりに、システム リセットの責任は BOD 回路にあることに注意することが重要です。しかし、必要であれば、ファームウェアは LVD 割込みをとおしてシステム リセットを開始できます。

HVLVD1_TRIPSEL [3:0]ビットフィールドは、サポートされる 16 の電圧閾値の 1 つを選択するために使用されます。閾値の詳細については、PSOC™ Control C3 MCU Register TRM の PWR_LVD_CTL レジスタの説明を参照してください。HVLVD_SRCSEL[6:4]ビットフィールドは、デフォルト値の 0x0 を使用するか、0x1 と 0x4 を使用してそれぞれ AMUXBUSA または AMUXBUSB 経由の VDDD 分岐を選択することで、VDDD モニターのみをサポートします。その他の値は予約されています。LVD を有効に設定するための HVLVD1_EN [7] ビットを設定します。PWR_LVD_CTL レジスタの他の設定ビットを変更する前に、LVD が無効にされていることを確認してください。HVLVD1_EDGE_SEL [17:16]ビットフィールドは、閾値を超えた時にどのエッジ (複数可) をトリガーとして動作するかを選択するために使用されます。これは、無効、立ち上りエッジ、立ち下りエッジ、または立ち上りと立ち下りの両方のエッジに設定できます。

17.6 スリープ制御

スリープ制御には、アクティブ/スリープからディープスリープ、ハイバネートへの電力モード遷移と、アクティブ/スリープモードへのウェイクアップを有効に設定するために必要なロジックと回路が含まれます。コプロセッサやペリフェラルが CPU の介入なしに動作している可能性があります。ディープスリープに入ってもこれらの動作が影響を受けないように、スリープ制御は静止要求と応答スキームを実装しています。ディープスリープ エントリーが要求されると、CPU は PERI に静止要求を送り、PERI はこの要求を関連するペリフェラルに展開します。これらのリクエストは Q チャネルと呼ばれる専用インターフェースで送信されます。いずれかのペリフェラルが要求を拒否した場合、ディープスリープエントリーは中止されます。

ウェイクアップ割込みコントローラー (WIC) は、ディープスリープやハイバネート モードからシステムをウェイクアップさせる入力を検知するために使用されます。WIC はウェイクアップ要求をメインのパワーポリシーユニット (PPU) に転送し、PPU は順番にウェイクアップシーケンスを実行してデバイスをアクティブモードにします。詳細は、デバイス電力モードを参照してください。

17.7 レジスタの概要

表 103 電源と電圧監視レジスタの一覧

レジスタ	機能
PWR_LVD_CTL	高電圧/低電圧検出 (HVLVD) コンフィギュレーション レジスタ
PWR_LVD_STATUS	高電圧/低電圧検出 (HVLVD) ステータス レジスタ
PWR_CTL2	POR/BOD、電圧/電流リファレンス、LDO 制御レジスタ

18 デバイス電力モード

PSOC™ Control C3 デバイスは、アプリケーションの平均消費電力を最小限に抑えることを目的とした複数の電力モードに対応しています。PSOC™ Control C3 は、低消費電力、短い起動時間、利用可能なペリフェラル、利用可能なウェイクアップ ソースの間の最適なバランスを達成するために、6 つの低消費電力モードに対応しています。

ペリフェラル ドライバ ライブラリ (PDL) は、すべてのデバイスの電力モード遷移をサポートしており、PSOC™ Control C3 電源リソースの遷移と設定に推奨される方法です。

18.1 特長

PSOC™ Control C3 電力モードは以下の特長があります。

- ソフトウェアで電力モードを切り換えることにより、アプリケーションの消費電力を最適化できます。
- ディープスリープモードは複数のウェイクアップ ソースと設定可能な SRAM リテンション量に対応
- ハイバネートモードにより、I/O、LPComp、RTC、タイマーアラームからのウェイクアップが可能

さまざまな電力モードでの電力消費は、次の方法で制御されます。

- 各ペリフェラルのクロックを有効もしくは無効に設定
- クロック ソースの電源オン/オフ
- ペリフェラルおよび PSOC™ Control C3 内の部品の電源オン/オフ

18.2 アーキテクチャ

PSOC™ Control C3 MCU は、消費電力の大きい順に以下の電力モードをサポートします。

- **アクティブ/スリープ**: アクティブ モードでは、すべてのペリフェラルと CPU は有効です。スリープ モードでは CPU を除いて、すべてのペリフェラルは有効です。
  - **低消費電力アクティブ (LPACTIVE)/低消費電力スリープ (LPSLEEP)**: これらのモードは、アクティブ/スリープ モード内のレジスタ コンフィギュレーションとして実装された低消費電力プロファイルです。
- **ディープスリープ**: このモードでは、アクティブ ロジックはフリーズしますが、低速機能は使用できます。ほとんどのコンフィギュレーションとメモリ状態は保持されます。高速クロックソースはオフ。ウェイクアップでは、プロセッサを中断した場所に戻ります
- **ディープスリープ RAM**: アクティブ モード ロジックは保持されないため、ウェイクアップ時にプロセッサ リセットが必要となります。システム RAM 全体を保持できます。ウェイクアップによりプロセッサが「ウォーム」リブートします
- **ディープスリープ OFF**: アクティブ モード ロジックおよびメモリは保持されません。ウェイクアップによりプロセッサが「コールド」リブートします
- **ハイバネート**: VDDD がまだ存在する最低消費電力モード。デバイスと I/O はフリーズ状態で、デバイスはウェイクアップ時にリセットされます。

デバイス内のデジタル回路の大部分には、2 つの内蔵リニア電圧レギュレーター (リニアアクティブレギュレーターとディープスリープレギュレーター) が電力を供給します。リニア アクティブ レギュレーターは、アクティブ モードとスリープ モードで使用され、ディープスリープ レギュレーターは、リテンション レギュレーターと NWELL レギュレーターとともに、ディープスリープ、ディープスリープ RAM、ディープスリープ OFF モードで使用されます。

デバイスは複数の電圧動作点をサポートしており、アクティブ モードでの消費電力を最適化するためのダイナミックな電圧スケーリングが可能です。ロジック コアに電力を供給するリニア アクティブ レギュレーターからの電圧 (VCCD) は、システムの最大動作周波数に基づいて調整できます。

パワーマネージメントユニットシステムは、以下のモード選択で動作します。

- **超低消費電力 (ULP) モード**: CPU が 50 MHz で動作している場合 0.9 V 設定
- **中間周波数 (MF) モード**: CPU が 70 MHz で動作している場合 1.0 V 設定

- **低消費電力 (LP) モード**: CPU が 150 MHz で動作している場合 1.1 V 設定
- **オーバードライブ (OD) モード**: CPU が 180 MHz で動作している場合 1.2 V 設定

PSOC™ Control C3 デバイスは、1.71 V から 3.6 V の範囲の単一の外部電源 (VDDD) で動作し、パワー サブシステムによって遷移が管理される 6 つの電力モードを備えています。バックアップ電源 (VBACKUP) は 1.71 V～3.6 V の範囲で動作します。外部 VBACKUP が接続されていない場合は、VDDD 上の電源がバックアップ電源として使用されます。

18.2.1 電力モード

PSOC™ Control C3 デバイスは次の電力モードに対応しています。

表 104 PSOC™ Control C3 MCU 電力モード

電力モード	説明	遷移条件	ウェイクアップ ソース	ウェイクアップのトリガー
アクティブ	• すべてのペリフェラルと CPU が利用可能な基本動作モード • 最小 7 mA の消費電流 (Typ.)	スリープ/ディープスリープモード, ハイバネートリセット, またはその他のリセットからのウェイクアップ	適用なし	適用なし
スリープ	• CPU はスリープモード; 他のすべてのペリフェラルは利用可能 • 最小 4 mA の消費電流 (Typ.)	アクティブモードからのレジスタ書き込み、またはデバッガを介したディープスリープからのウェイクアップ	CPU への割込み	割込み
ディープスリープ	• すべての高周波クロックとペリフェラルはオフになります。 • 低周波クロックと低消費電力アナログおよびデジタル ペリフェラルは、動作用およびウェイクアップソースとして利用できます。 • 最小 7 µA の消費電流 (Typ.)	アクティブ モードでレジスタ書き込みを実施	GPIO 入力、低消費電力コンパレーター、SCB インスタンス、ウォッチドッグ タイマー、RTC アラーム、デバッガー	割込みまたはデバッグ

(続く)

表 104 (続き) PSOC™ Control C3 MCU 電力モード

電力モード	説明	遷移条件	ウェイクアップ ソース	ウェイクアップのトリガー
ディープスリープ RAM	• ディープスリープに似た低消費電力モード。ただし、ほとんどのコンフィギュレーション状態は保持されません。 • ウェイクアップによりプロセッサが再起動します • ソフトウェアは、保持された SRAM コンテンツ (プログラマブル) を使用して、アプリケーションの状態をより迅速に回復し、「ウォーム」リブートを可能にします。	アクティブ モードでレジスタ書き込みを実施	GPIO 割込み、低消費電力コンパレーター、SCB、CTBm、ウォッチドッグ タイマー、および RTC アラーム	ウォームリブート
ディープスリープ OFF	• ディープスリープに似た低消費電力モード。ただし、ほとんどのコンフィギュレーション状態とメモリは保持されません。 • ウェイクアップにより、プロセッサは「コールド」再起動を行います	アクティブ モードでレジスタ書き込みを実施	GPIO 割込み、低消費電力コンパレーター、SCB、CTBm、ウォッチドッグ タイマー、および RTC アラーム	コールドリブート
ハイバネート	• GPIO の状態がフリーズし、すべての高周波クロックとペリフェラルがオフになります。 • 低周波クロックと WCO は機能します • ウェイクアップイベントでデバイスがリセットされます • 最小 300 nA のの消費電流 (Typ.)	アクティブ モードでレジスタ書き込みを実施	WAKEUP ピン、低消費電力コンパレーター、RTC アラーム、ウォッチドッグ タイマ	ハイバネート リセット

18.2.1.1 アクティブとスリープ

アクティブ モード: このモードは、システムで使用されるすべてのデバイス リソースがアクティブになり、クロックが供給される通常の動作電力モードです。このモードでは、CPU はすべてのロジックとともにコードを実行し、メモリ駆動となります。ファームウェアは、アプリケーションと電力要件に基づいて、特定のペリフェラルと電源ドメインの有効化または無効化を決定できます。すべてのペリフェラルはアクティブモードで使用可能です。

スリープ モード: スリープモードでは、CPU クロックはオフになり、スリープに移行するために、CPU はコードの実行を停止します。アクティブ モードで使用可能なすべてのペリフェラルは、スリープ モードでも使用可能です。CPU が必要であれば、クロックゲーティングによって瞬時に起動できます。マスクされていない割込みはすべて、CPU をアクティブ モードにウェイクアップできます。

LPACTIVE/LPSLEEP: これらの低消費電力プロファイルは、アクティブ/スリープ モード内のレジスタ設定として実装され、アクティブまたはスリープ モード時の消費電力を削減します。これらの低消費電力プロファイルは、電流が制限され、クロックが低周波数で動作し、一部の機能が利用できないか制限されることを除けば、アクティブ/スリープに似ています。

- **LPACTIVE**:
  - clk_hf0 (メイン ロジック クロック) のクロックは、ノミナル 8 MHz で動作する IMO として選択されます。
  - 基準電圧バッファ、低電圧検出 (LVD)、その他のクロック ソースなど、オプションのペリフェラルをオフにします。
  - コードはフラッシュから実行されます
  - POR/BOD とリファレンスは低消費電力モードになり、VDDD の最大外部ランプ レートが制限されます。
- **LPSLEEP**: LPSLEEP と LPACTIVE の関係は、スリープとアクティブのような関係です。

18.2.1.2 ディープスリープ

ディープスリープ モードは低消費電力モードで、アクティブ ロジックはフリーズしますが、低速機能は利用可能なままです。ほとんどのコンフィギュレーションとメモリ状態は保持され、GPIO の状態はフリーズします。このモードでは、内部メイン発振器 (IMO)、外部水晶発振器 (ECO)、位相同期ループ (PLL) などの高速クロック ソースがオフになり、CPU、SRAM、高速ロジック、高速ペリフェラルが使用できなくなります。しかし、時計用水晶発振器 (WCO) や内部低速発振器 (ILO) のような低速クロック ソースや、ファームウェアによって設定され有効化されたペリフェラルは、引き続き動作します。ILO は、より節電のためにオフにできます。ウェイクアップするとプロセッサが停止した時の状態に戻ります。

このモードでは、同じプログラムカウンター値から動作を再開でき、再開を可能にする状態が完全に保持されます。スタックや重要なパラメーターも保持可能で、低速非同期または低消費電力のアナログペリフェラルからの割込みによって、CPU はディープスリープモードからウェイクアップできます。ディープスリープからの通常のウェイクアップでは、特定のウェイクアップ割込みでプログラムされた動作に応じて、プロセッサはアクティブまたはスリープのいずれかに戻りますが、ディープスリープからのデバッグ ウェイクアップではスリープに戻ります。

このモードは、プロセッサがコードを実行する必要はありませんが、ペリフェラルがシステムクロック速度より低い速度で機能する必要があるアプリケーションで使用されます。

18.2.1.3 ディープスリープ RAM

アクティブ モードでは、ロジックは保持されず、ウェイクアップ時にプロセッサ リセットが必要となります。しかし、ディープスリープ RAM モードでは、SRAM 全体が保持されます。このソフトウェアは、重要な情報をシステム RAM に保存でき、完全な「コールド」再起動よりも迅速にアプリケーションを以前の状態に戻す「ウォーム」再起動を可能にします。

ディープスリープ モード中は、ロジックは動作可能ですが、SRAM のみが保持されます。ウェイクアップが発生すると、ウォームリブートと呼ばれるリセットが行われます。

注: SRAM 領域 (SRAM サイズから 8 KB を引いたもの) は、ブート動作中にインフィニオンのファームウェアによって使用されます。この領域はユーザーが利用できますが、インフィニオン ブート ファームウェアによって上書きされる可能性があるため、ディープスリープ RAM からの復帰を含め、リセットに伴うデータ保持は保証されません。

18.2.1.4 ディープスリープ OFF

アクティブ モードでは、ロジックとメモリはまったく保持されないため、アプリケーションを以前の状態に戻すためには完全な「コールド」再起動が必要となります。

ディープスリープ中、ロジックは機能しますが、アクティブ ロジックと SRAM の電源はオフになります。ウェイクアップが発生すると、リセット (コールドリブート) になります。

ディープスリープ OFF モードでは、SRAM リテンションは利用できません。

表 105 に、特定のディープスリープ状態後の PSOC™ Control C3 ウェイクアップ動作の概要を示します。

表 105 ディープスリープ状態とウェイクアップ行動

ディープスリープ状態	ウェイクアップ後
ディープスリープ	WFI/WFE からのウェイクアップ
ディープスリープ RAM	ウォームリブート
ディープスリープ OFF	コールドリブート

18.2.1.5 ハイバネート

ハイバネート モードは、デバイスの最低消費電力モードで、外部電源がまだ存在し、XRES がデアサートされている間の休止状態のアプリケーション用に設計されています。このモードでは、アクティブ スリープ レギュレーターとディープスリープ レギュレーターの両方がオフになり、すべてのクロックが停止します。さらに、GPIO のステートはフリーズされ、指定された I/O ピンの遷移や可能なリセット イベントなどの非同期イベントのみがサポートされます。低速コンパレーター電流を犠牲にして、このモードでコンパレーターを使用することも可能です。このモードでは、デバイスをリセットする必要があります。

ハイバネート モードから復帰すると、ブート ROM のコードはチップがその状態になったことを検出し、必要最小限のコードを実行して動作を再開し、適宜、保持されているデータを利用します。このモードでは、GPIO の状態は保持されます。ハイバネート モードからのウェイクアップは、専用ウェイクアップ ピン (P2.0 と P9.0) と低消費電力コンパレーター出力によって容易に行われます。ハイバネート モードでの低消費電力コンパレーター動作には、ウェイクアップ比較用の外部電圧が必要です。さらに、RTC アラームとウォッチドッグ タイマー (WDT) 割込みは、ハイバネート ウェイクアップ信号を生成できます。ハイバネート状態からのウェイクアップソースの設定は、PWR_HIB_WAKE_CTL レジスタを使用して行えます。例えば、PWR_HIB_WAKE_CTL レジスタの HIB_WAKE_RTC を有効に設定することで、ハイバネート状態からの RTC アラームによるウェイクアップを可能にします。

ハイバネートモードの要件を満たせないアプリケーションで誤ってハイバネートモードに入るのを防ぐため、ハイバネートモードを無効に設定するオプションが用意されています。これは、PWR_HIBERNATE レジスタの HIBERNATE_DISABLE ビット[30]を設定し、デバイスのハイバネートモードを無効に設定することで実現できます。このビットは、実行中はライトワンス ビットで、リセット時にのみクリアされることに注意してください。ハイバネートモードに入ると、デバッグ機能が失われ、デバッガが切断されます。

ハイバネートモードとディープスリープ OFF モードの主な違いは、後者ではディープスリープロジックとレギュレーターが動作可能に対し、ハイバネートでは VDDD 供給ロジックとバックアップドメインのみが使用可能です。

18.2.2 その他の動作状態

前のセクションで説明した電力モードに加え、デバイスはリセットと電源オフの 2 つの状態を取りえます。これらの状態は、外部電源と XRES の接続によって決定されます。これらのモードに入るにはファームウェアの動作は必要なく、これらのモードを終了するにも割込みやウェイクアップ イベントは必要ありません。リセットとオフの状態に関する議論が含まれているのは、デバイスが入る可能性のあるすべてのモードと状態をカバーするためです。これらの状態をシステムで利用することで、消費電力をさらに最適化できます。

18.2.2.1 リセット状態 (XRES)

リセットとは、外部リセット (XRES ピンが Low にプルされた状態) が印加されたとき、またはパワーオンリセット/ブラウンアウト検出がアサートされたときのデバイスの状態です。リセットは電力モードとしては機能しません。リセット状態の間、デバイスの全コンポーネントはパワーダウンし、I/O はトライステートされるため、消費電力は最小限に抑えられます。

18.2.2.2 オフ状態 (OFF)

オフ状態は、単に電源が印加されていないデバイスの状態を表します。

18.3 電力モードの遷移

図 62 に、デバイスが取り得る様々な状態と、可能な電力モード遷移経路を示します。

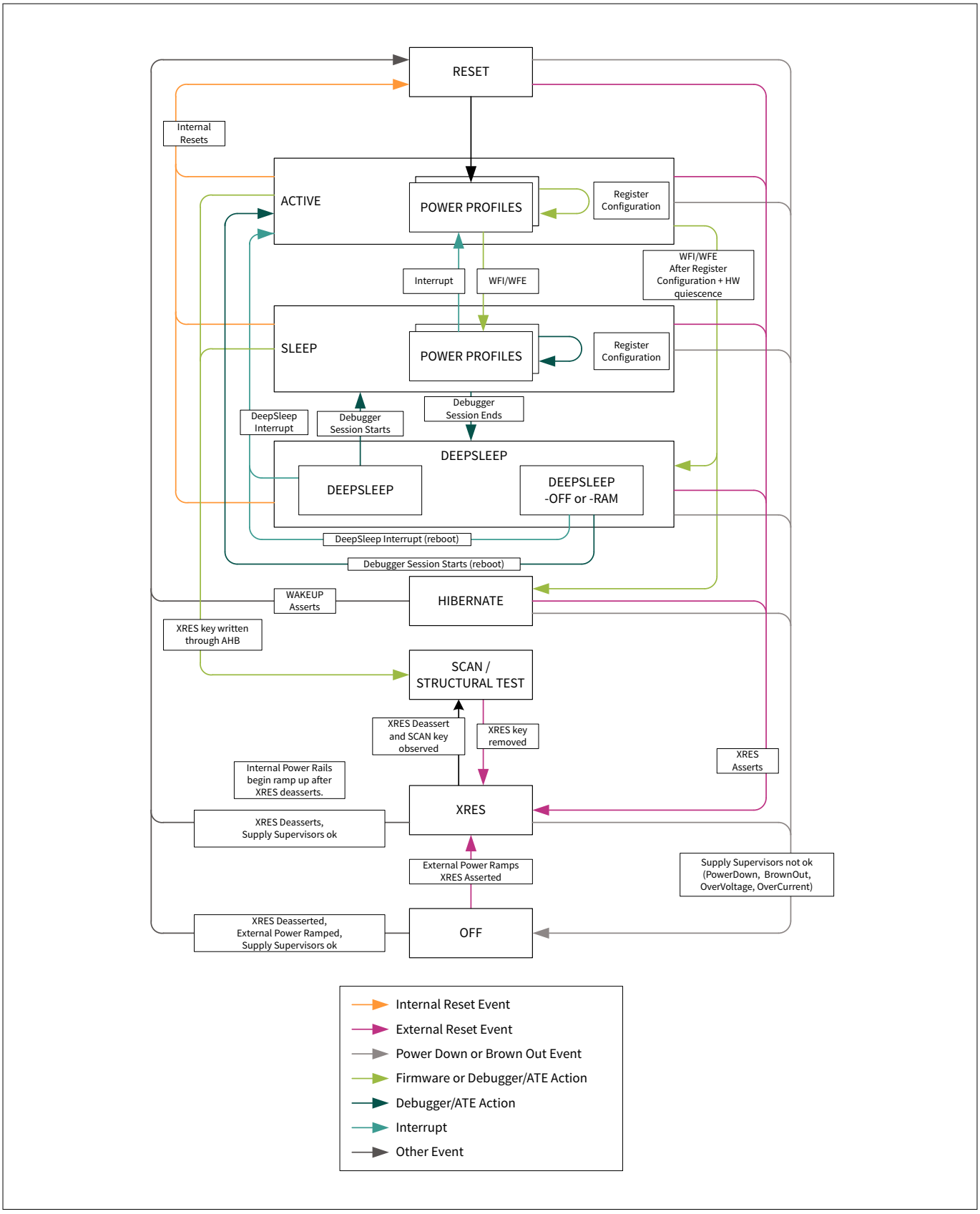


図 62　　電力モードの遷移

18.4 まとめ

表 106 に、特定の電力モードでアクティブになる利用可能なリソースと、それに対応するウェイクアップ パラメーターを示します。

表 106 電力モードとウェイクアップ パラメーター

	アクティブ/スリープ	ディープスリープ	ディープスリープ RAM	ディープスリープ OFF	ハイバネート	オフ
パラメーター						
ウェイクソース [1]	任意の割込み	DS ペリフェラル	DS ペリフェラル	DS ペリフェラル	RTC/HIB ペリフェラル	電源投入
ウェイクアクション	再開	再開	ウォーム ブート	リセット/コールドブート	リセット	リセット
ウェイク時間	1 CPU サイクル	<20µs	ディープスリープ＋ウォームブート	ディープスリープ＋コールドブート	POR＋コールドブート＜1 ms	
リソース						
ECO	オン/オフ	オフ	オフ	オフ	オフ	オフ
IHO	オン	オフ	オフ	オフ	オフ	オフ
IMO	オン	オン/オフ	オフ	オフ	オフ	オフ
ILO	オン/オフ	オン/オフ	オン/オフ	オン/オフ	オン/オフ	オフ
WCO	オン/オフ	オン/オフ	オン/オフ	オン/オフ	オン/オフ	オフ
CPU	オン/スリープ	リテンション	オフ	オフ	オフ	オフ
SRAM	オン	オン	オン/オフ	オフ	オフ	オフ

1) DS (ディープスリープ) および HIB (ハイバネート) 電力モードで使用可能なペリフェラルの一覧については、表 107 を参照してください。

低消費電力モードで利用可能な動作状態は、一般に、アクティブ モードでの能力に比べ、機能やパラメトリック性能が制限されています。さらに、ディープスリープやハイバネートなどの低消費電力モードをサポートしていないブロックは、これらのモードからは CPU をウェイクアップできません。

表 107 は、デバイスの電力モード/状態における様々なデバイス コンポーネントの利用可能性を概説しています。

表 107 異なる電力モードで使用できるリソース

ブロック	電力モード				
	アクティブ	スリープ	ディープスリープ	ハイバネート	バックアップ
CPUSS					
CPU	Y	N	N	N	N
NVIC	Y	Y	N	N	N
WIC	Y	Y	Y	Y	N
FLASH	Y	Y	N	N	N

(続く)

表 107 (続き) 異なる電力モードで使用できるリソース

ブロック	電力モード				
	アクティブ	スリープ	ディープスリープ	ハイバネート	バックアップ
SRAM	Y	Y	Y	N	N
DMA	Y	Y	N	N	N
プログラマブル デジタル					
スマート I/O	Y	Y	Y	N	N
固定機能デジタル					
TCPWM	Y	Y	N	N	N
SCB	Y	Y	Y[1)]	N	N
CAN FD	Y	Y	N	N	N
特殊機能					
CORDIC	Y	N	N	N	N
アナログ					
HPPASS (SAR, CSG)	Y	Y	N	N	N
LPComp	Y	Y	Y	Y	N
I/O					
GPIO	Y	Y	Y	有り [2)]	N
バックアップ					
RTC	Y	Y	Y	Y	Y
レジスタ	Y	Y	Y	Y	Y

1) SCB0 (I2C、SPI) のみ
2) hibernate_wakeup ピン P2.0 および P9.0) のみが動作可能で、デバイスをハイバネートモードからウェイクアップできます。

18.5 レジスタの概要

表 108 レジスタリスト

名称	説明
PWR_CTL	デバイスの電力モードオプションの制御、現在の状態の監視を行います。
PWR_CTL2	
PWR_HIBERNATE	ハイバネートモードの開始/終了に関連するさまざまなオプションを制御します。
PWR_HIB_WAKE_CTL	ハイバネート ウェイクアップ マスク レジスタは、ハイバネート割り当てウェイクアップを設定し、記録をトリガーします。

(続く)

表 108 (続き) レジスタリスト

名称	説明
PWR_HIB_WAKE_CTL2	ハイバネート ウェイクアップ極性レジスタは、対応するハイバネート ウェイクアップ ソースの極性を選択します。
PWR_HIB_WAKE_CAUSE	ハイバネート ウェイクアップ要因レジスタは、マスクされていないハイバネート ウェイクアップ要因を記録し、ハイバネート モード中に保持します。

19 クロック システム

PSOC™ Control C3 MCU ファミリーは、オンチップ オシレーター、フェーズ ロック ループ (PLL)、および複数の外部クロック ソースをサポートする柔軟なクロック オプションを提供します。

19.1 特長

PSOC™ Control C3 MCU クロック システムには以下のリソースが含まれます。

- 3 つの内部クロック ソース:
  - 8 MHz 内部主発振器 (IMO)
  - 48 MHz 内部高速発振器 (IHO)
  - 32 kHz 内部低速発振器 (ILO)
- 3 つの外部クロック ソース:
  - I/O ピンから最大 80 MHz の外部クロック (EXT_CLK) を入力可能
  - 4～35 MHz の外部水晶発振器 (ECO)
  - 32.768 kHz 外部時計用水晶発振器 (WCO)
- 24～100 MHz の出力範囲を持つ周波数ロック ループ (FLL) 1 つ
- 2 つのデジタル位相同期ループ (DPLL0 と DPLL1)、25～250 MHz の出力範囲
- 各クロック ソースには、クロック スーパーバイザー (CSV) があります。

19.2 アーキテクチャ

図 63 に、PSOC™ Control C3 MCU ファミリーのクロックシステムの一般的なビューを示します。

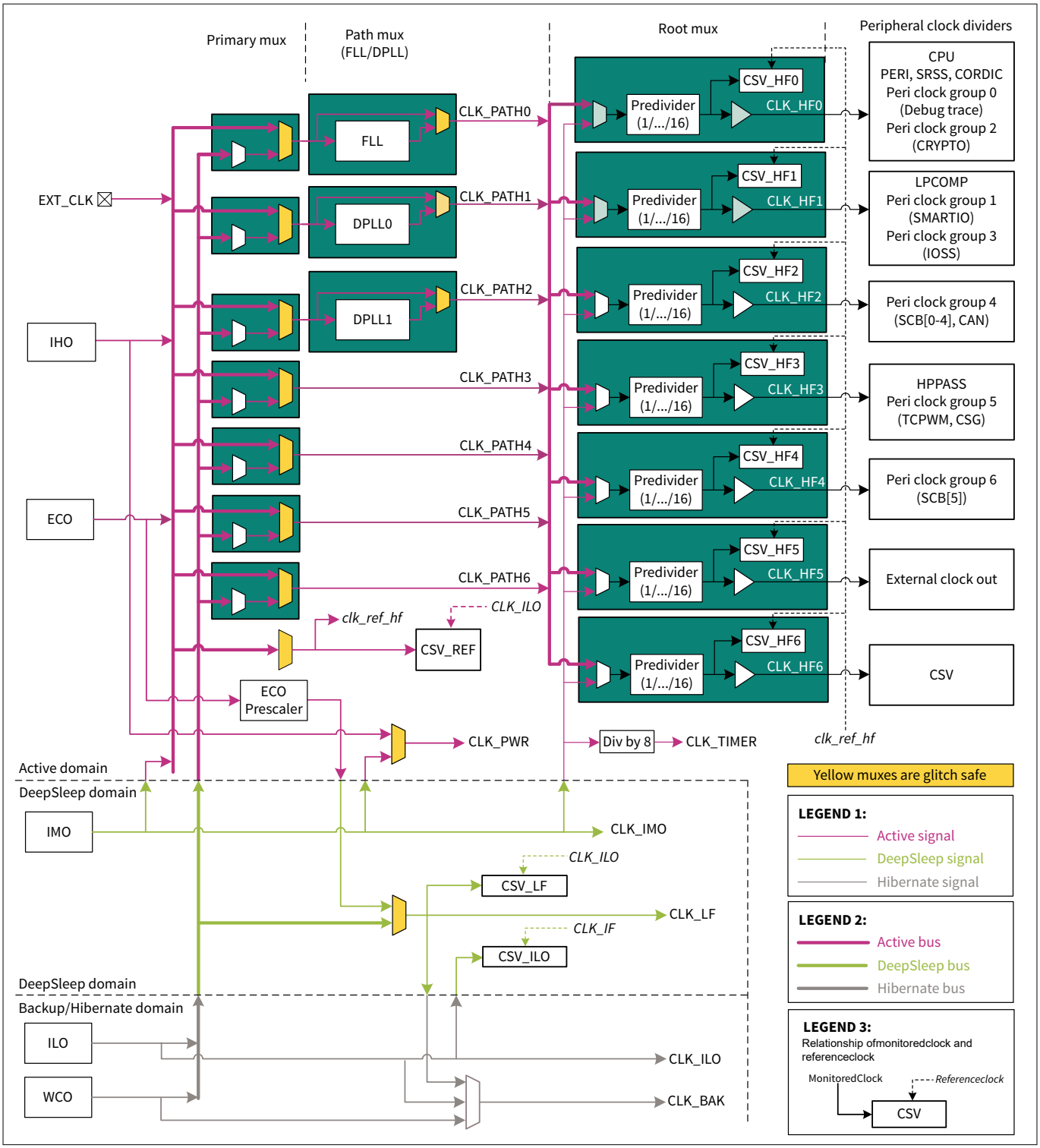


図 63　　クロック システム ブロック図

注:　ペリフェラルクロック分周器の詳細についてはペリフェラルクロック分周器 セクションを参照してください。

19.3 クロックソース

19.3.1 内部メイン発振器 (IMO)

IMO は、高速ウェイクアップと低ジッターの高速内部 (水晶なし) 主発振器で、8 MHz の固定周波数を生成し、許容誤差±2%です。IMO は、パワーオンリセット (POR) やその他のリセット後のデフォルトのクロック ソースです。DPLL または FLL は IMO を使用して広範囲の高周波クロックを生成するか、高周波ルート クロックとして IMO を直接使用します。IMO を利用できるのは、アクティブモード、スリープモード、ディープスリープモードです。

19.3.2 内部高周波発振器 (IHO)

IHO は正確で高速な内部 (水晶振動子なし) 発振器で、外付け部品なしで固定周波数 48 MHz を生成し、誤差±1%です。高速クロックは、IHO と DPLL を使用して導き出せます。IHO はアクティブとスリープ モードで動作します。

19.3.3 内部低速発振器 (ILO)

ILO は外付け部品なしで動作する低速内部発振器で、公称周波数 32 kHz の安定したクロックを出力します。ILO は比較的低消費電力で低精度です。すべての電力モードで利用可能です。ILO がハイバネート モードで、パワー オン リセット (POR) またはブラウンアウト検出 (BOD) を超えてもアクティブを維持する場合は、CLK_ILO_CONFIG レジスタに ILO_BACKUP ビットを設定する必要があります。

ILO は、CLK_LF のクロック ソースとしてだけでなく、バックアップ ドメイン (CLK_BAK) のソースとしても使用できます。CLK_BAK はリアルタイム クロック (RTC) を動作させます。ILO は精度が低いため RTC としては適しませんが、RTC のウェイクアップ アラーム機能を使用することで HIBERNATE のウェイクアップ ソースとして使用できます。この場合、ILO を動作させるには、ハイバネート中に VDDD レールを供給し、CLK_ILO_CONFIG レジスタに ILO_BACKUP ビットを設定する必要があります。

ILO はウォッチドッグタイマー (WDT) のクロック ソースとして使用できます。ILO の有効化および無効化は、CLK_ILO_CONFIG レジスタの ENABLE ビットで行います。ILO が WDT のソースの場合は、有効のままにしてください。WDT が有効な場合、ILO を無効に設定する唯一の方法は、まず WDT_CTL レジスタの WDT_LOCK ビットをクリアし、次に CLK_ILO_CONFIG レジスタの ENABLE ビットをクリアすることです。WDT_LOCK ビットが設定されている場合、ILO を無効化するためのレジスタ書き込みは無視されます。WDT を有効にすると、ILO が自動的に有効になります。

クロック キャリブレーションカウンターの章で説明されているキャリブレーション カウンターは、ECO のような高精度のクロックに対して ILO を測定するために使用できます。この結果をもとに、ILO をどのように調整すべきか決定できます。ILO は CLK_TRIM_ILO_CTL レジスタを使用してトリミングできます。

19.3.4 外部クロック (EXT_CLK)

外部クロックは 0.001～80 MHz 範囲のクロックで、指定された I/O ピンの信号から供給可能です。このクロックは DPLL または FLL のソース クロックとして使用でき、また高周波クロックによって直接使用もできます。

手動でピンを EXT_CLK の入力として設定する場合は、ピンのドライブモードをハイインピーダンス デジタルに設定し、デジタル入力バッファを有効にします。詳細については I/O サブシステムの章を参照してください。EXT_CLK に使用される特定のピンについては、デバイスのデータシートを参照してください。

19.3.5 外部水晶発振器 (ECO)

PSOC™ Control C3 MCU には、外部 4 MHz～35 MHz 水晶振動子を駆動する発振器が内蔵されています。このクロック ソースは発振回路を使って作られています。この回路は、PSOC™ Control C3 MCU の外部水晶振動子ピン

に外部水晶振動子を配置する必要があります。ハードウェアは、ハイバネート、ディープスリープ、および XRES の間、自動的に ECO を無効にします。ECO はアクティブ モードでのみ作動します。

ECO は、CLK_ECO_CONFIG.ECO_EN レジスタのビット フィールドを使用して有効に設定できます。ECO が有効になった後、CLK_ECO_STATUS レジスタをチェックし、準備ができていることを確認します。ECO クロックは CLK_HF と CLK_LF (プリスケール) から選択できます。

19.3.6 時計用水晶発振器 (WCO)

WCO は高精度の 32.768 kHz クロック ソースです。リアル タイム クロック (RTC) のプライマリ クロック ソースです。WCO は CLK_LF と CLK_BAK のソースとしても使用できます。バックアップドメインの BACKUP_CTL レジスタの WCO_EN ビットを設定することで、WCO を有効/無効に設定できます。また、WCO をバイパスし、外部 32.768 kHz クロックを WCO 出力ピンにルーティングすることも可能です。バックアップドメインの BACKUP_CTL レジスタに WCO_BYPASS ビットを設定することで行えます。

WCO は外部クロック/正弦波入力からも動作可能です。これらのモードでは、WCO を有効に設定する前に、BACKUP_CTL レジスタの WCO_BYPASS ビットを設定することにより、WCO をバイパスします。また、WCO_OUT と WCO_IN 信号の GPIO をアナログモードに設定します。

19.3.7 ECO プリスケーラ

ECO プリスケーラは ECO を分周し、CLK_LF クロックで使用可能なクロックを生成します。この機能を利用できるのは、アクティブモードとスリープモードのみです。ディープスリープまたはハイバネートモード中は使用できません。

分周機能には、10 ビットの整数除算器と 8 ビットの分数除算器があります。この機能は CLK_ECO_PRESCALE レジスタを使って設定します。

19.4 クロック生成

PSOC™ Control C3 MCU は完全に統合されたクロックを持ち、クロックを必要とするすべてのブロックにクロックを提供します。異なるクロック ソースをグリッチなしで切り替え、確実に準安定状態がないようにします。

19.4.1 デジタル位相同期回路 (DPLL)

DPLL は PSOC™ Control C3 MCU 用のデジタル PLL で、25～250 MHz の出力クロック周波数をサポートします。PLL は、デジタル フィルターと、出力デューティ サイクル 48～52%の内部 250～500 MHz デジタル制御発振器 (DCO) を使用します。また、電源ノイズを除去し、ジッター性能を向上させる LDO も内蔵しています。

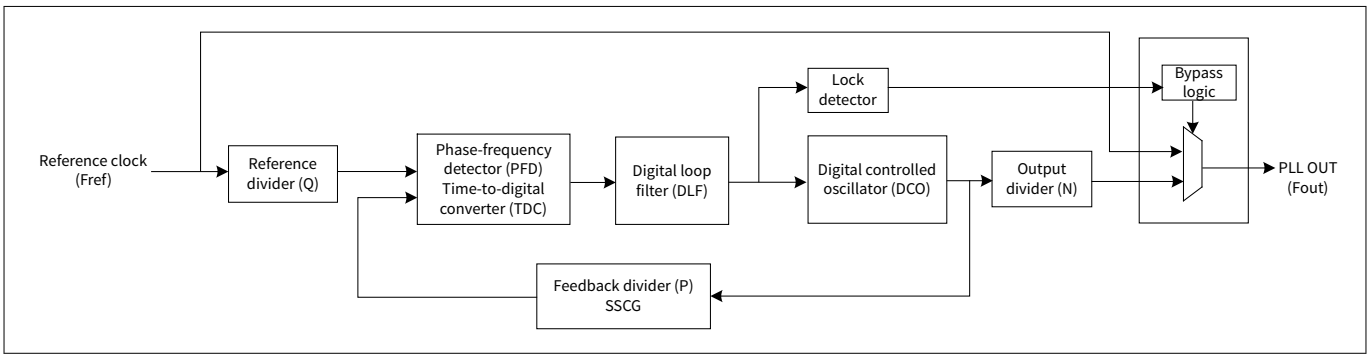


図 64 DPLL ブロック図

図 64 に DPLL のブロック図を示します。時間デジタル変換器 (TDC) ブロックは、基準クロックとフィードバック クロックの差をコードに変換し、デジタル ループ フィルター (DLF) に入力します。DCO にはデジタル コードが入力され、それに応じて出力周波数を生成します。

PLL は、プログラマブルな入出力整数分周器 (バイナリのみではない) とフィードバック分周器 (整数分周器と分数分周器) を使用して、複数の周波数合成コンフィギュレーションを提供します。PLL には 3 つのプログラマブル分周器があります。

- P (製品): この分周器は、DCO 出力と位相周波数検出器 (PFD) 入力間のフィードバック経路にあります。DCO の乗算係数を設定します
- Q (商): この分周器は入力 (基準) クロックと PFD の間にあります。入力クロックを分周してから PFD に渡します。
- N (出力/ポストディバイダ): この分周器は、DCO の出力クロックを受け取り、目的の周波数と PLL の出力が得られるように分周します。

すべての分周器 (P、Q、N) は、入力クロックを整数で分周するようにデジタルでプログラム可能で、フィードバック クロックは分数での分周もサポートしています。PLL は、P、Q、N 分周器の設定と、適切な基準クロックの適用に基づいて、特定の出力周波数を生成します。

PLL が対応している動作モード

- **整数演算モード**: PLL は P、Q、N を設定することで入力周波数の整数倍の変調を生成し、PLL 出力周波数を設定します。

  PLL 出力周波数方程式 (整数モード):

$$F_{out} = F_{ref} \times (P/Q/N) \tag{1}$$

  ***注**: Fref は PLL の入力周波数、つまり PATH_MUX を使って選択された高周波クロック ソースの周波数 (例えば IMO の 8 MHz)。*

- **分数演算モード**: 分数周波数は 24 入力ビットで定義されます。周波数分解能は$1/2^{24}$です。

  PLL 出力周波数方程式 (フラクショナル モード):

$$F_{out} = (F_{ref}/Q) \times (P + Frac_div)/N \tag{2}$$

  ***注**: 端数処理の設定は CLK_DPLL_LPx_CONFIG2 レジスタを使用します。Frac_div は CLK_DPLL_LPx_CONFIG2.FRAC_DIV で設定された値を2^{24}で割った値です。Frac_div = CLK_DPLL_LPx_CONFIG2.FRAC_DIV / 2^{24} です。*

  例を示します。Fref = 8 MHz から生成された非標準周波数 40.456 MHz

  P = 40, Q = 1, N = 8, Frac_div =7650410

  Fout = (8 * (40 + 7650410/16777216)) / 8 = 40.456 MHz

以下の手順で PLL 機能のテストを行ってください。

1. 希望する基準クロック周波数 (Fref) と希望する出力周波数 (Fout) を決定します。以下の制約条件に従って、基準分周器 (Q)、帰還分周器 (P)、出力分周器 (N) を計算します。
   - **a.** PFD 周波数 (周波数制約: 4〜8 MHz)。Fpfd=Fref/Q。この制約を満たす基準分周値は複数存在する可能性があります。
   - **b.** DCO 周波数 (周波数制約: 250〜500 MHz)。DCO = Fpfd × P。異なる REFERENCE_DIV の選択で、この制約を満たす複数のフィードバック分周器の値が存在する可能性があります。
   - **c.** 出力周波数。PLL_OUT = DCO/N。したがって、最も近い 2 つの選択肢の周波数誤差を考慮します。
   - **d.** 用途に応じて最適なデバイダ パラメーターの組み合わせを選択

2. CLK_DPLL_LPx_CONFIG レジスタで分周器設定をプログラムします。分周器の設定と同じレジスタ書き込み中に PLL を有効にしないでください。PLL が有効になっている間は、分周器の設定を変更しないでください。
3. 必要に応じて、CLK_DPLL_LPx_CONFIG2 レジスタで分数分周器を、CLK_DPLL_LPx_CONFIG3 レジスタで SSCG を設定します。
4. PLL を有効に設定します (CLK_PLL_CONFIGx.ENABLE = 1)。PLL 回路が起動するまで、少なくとも 1µs 待ちます。
5. 出力を使用する前に PLL がロックされるまで待ちます。デフォルトでは、PLL 出力は基準クロックにバイパスされ、ロックされると自動的に PLL 出力に切り替わります。この動作は CLK_DPLL_LPx_CONFIG.BYPASS_SEL で変更できます。PLL の状態は CLK_DPLL_LPx_STATUS を読み出すことで確認できます。このレジスタには PLL がロックしたことを示すビットが含まれます。また、PLL がロック状態を失ったかどうかを示すビットも含まれています。

スペクトラム拡散クロック生成 (SSCG)

スペクトラム拡散クロック生成 (SSCG) とは、クロック ソースの狭帯域に含まれる電力を制御された方法で広帯域に拡散させ、基本波と高調波のスペクトル振幅のピークを小さくすることで、クロック ソースからの放射エミッションを低減する方法です。これは、クロック周波数を波形で変調することで実現されます。SSCG のコンフィギュレーションには CLK_DPLL_LPx_CONFIG3 レジスタを使用します。

SSCG には以下のようなパラメーターがあります。

- **変調速度**
  変調率 (MR) とは、クロック ソースの電力が出力クロック周波数周辺の周波数帯域に分配される割合 (Hz) のことです。変調レートはソースクロック周波数よりはるかに低く、オーディオ周波数帯域以上でなければいけません。
- **変調深度**
  変調深度 (偏差とも呼ばれる) とは、変調率で変化しながらクロックが変化する周波数範囲のことです。これはパーセンテージ (%) で指定され、周波数エクスカージョンの帯域幅 (ΔF) とソースクロック周波数の比です。これにより、達成可能なピーク EMI 低減量が決まります。一般に、変調の深さが大きいほど、EMI の低減は大きくなります。しかし、周波数変動に関するシステムの制約を評価しながら変調深度を選択する必要があります。
- **変調タイプ**
  変調タイプ (または拡散モード) は、非変調クロックに対する変調クロックの周波数偏差の関係を指定します。一般的な SSCG では、センター、アップ、ダウンスプレッドが利用可能ですが、この PLL ではダウンスプレッドにのみ対応しています。
  ダウン拡散とは、スペクトラム拡散クロックの最大周波数が、変調されていないクロックの最大周波数と同じになることです。ダウンスプレッドシステムでは、出力クロックは変調レートと変調プロファイルに従って (Fc - ΔF) と Fc の間で変化します。これは、「Fout = Fc - ΔF」と表せます。

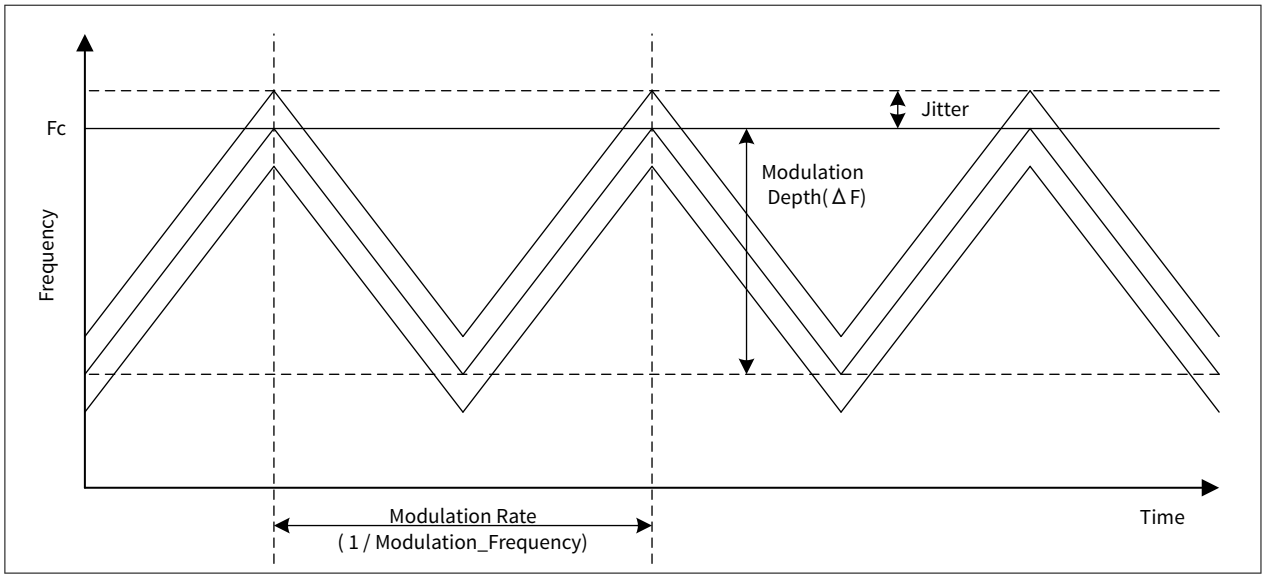


図 65　　ダウンスプレッディングの例

Fout = 変調出力クロック周波数
Fc = ソースまたはキャリア周波数
ΔF = 全周波数偏差 (最小～最大)

19.4.2　　周波数同期回路 (FLL)

PSOC™ Control C3 MCU には周波数同期回路 (FLL) が 1 つあり、CLK_PATH0 上に存在します。FLL は 24 MHz から 100 MHz のクロック出力を生成できます。入力周波数は 0.008～100 MHz で、電流制御発振器 (CCO) の周波数の少なくとも 2.5 倍以下でなければいけません。これにより、IMO を使用してシステムの残りの部分により高いクロック周波数を生成することが可能になります。

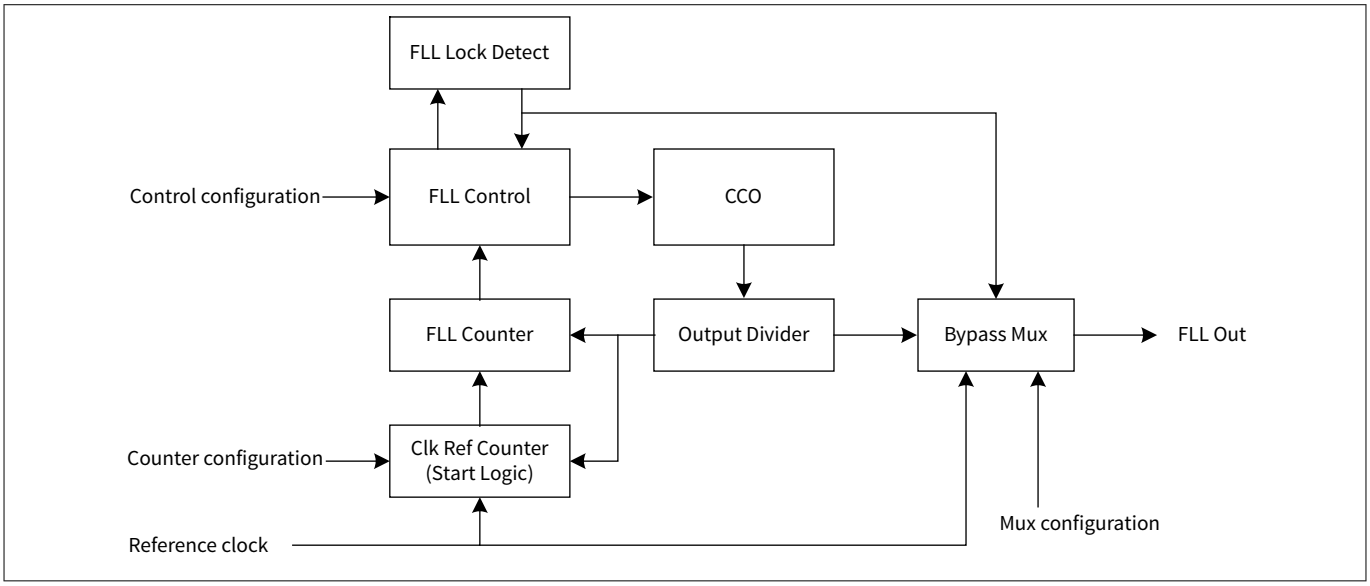


図 66　　FLL ブロック図

FLL は PLL と目的は似ていますが、同等ではなく、いくつかの違いがあります。

- PLL よりも消費電流が少ない
- FLL は位相をロックしません。FLL の心臓部は電流制御発振器 (CCO) です。この CCO の出力周波数は、CCO のトリムを調整することによって制御されます。

- FLL は分周クロック出力により、デューティサイクルの良い最大 100 MHz のクロックを生成可能です。
- FLL 基準クロックには、WCO (32.768 kHz)、IMO (8 MHz)、またはその他の周期的なクロックソースを使用できます。

CCO は 48 MHz から 200 MHz の範囲で安定した周波数を出力できます。この範囲は表 109 で示すように 5 つのサブレンジに分かれています。

表 109　　CCO 周波数範囲

CCO 範囲	0	1	2	3	4
f_{min}	48 MHz	64 MHz	85 MHz	113 MHz	150 MHz
f_{max}	64 MHz	85 MHz	113 MHz	150 MHz	200 MHz

注: CCO の出力には、2 分周を有効に設定するかどうかのオプションがあります。このデバイスの場合、2 分周は常に有効になっていなければいけません。

各レンジ内では、CCO 出力は 9 ビットのトリムフィールドで制御されます。このトリムフィールドは、以下に説明されている制御アルゴリズムに基づき、ハードウェアを介して更新されます。

基準クロックは FLL に提供されなければいけません。この基準クロックは通常 IMO ですが、様々なクロックソースを使用できます。FLL は基準クロックと CCO クロックを比較し、CCO トリムの調整方法を決定します。具体的には、FLL は基準クロックサイクルの指定されたウィンドウ内の CCO クロックサイクル数をカウントします。カウントする基準クロックサイクル数は CLK_FLL_CONFIG2 レジスタの FLL_REF_DIV フィールドによって設定されます。

CCO クロックがカウントされた後、理想値と比較され、誤差が計算されます。理想的な値は CLK_FLL_CONFIG レジスタの FLL_MULT フィールドにプログラムされます。

例として、基準クロックは IMO (8 MHz)、希望する CCO 周波数は 100 MHz、FLL_REF_DIV の値は 146 に設定されます。これは、FLL が基準クロックの 146 クロック周期内にある CCO クロックの数をカウントすることを意味します。基準クロック (IMO) の 1 クロックサイクルには、100/8 = 12.5 クロックサイクルの CCO が存在するはずです。この数値に 146 を掛けると、FLL_MULT の値は 1825 となります。

FLL が 1825 と異なる値をカウントした場合、次にカウントするときに 1825 を達成するように CCO を調整しようとします。これは、CLK_FLL_CONFIG3 にある FLL_LF_IGAIN と FLL_LF_PGAIN で誤差項をスケーリングすることによって行われます。図 67 に、誤差 (err) 項が FLL_LF_IGAIN と FLL_LF_PGAIN に乗算され、現在のトリムと合計され、CCO の新しいトリム値を生成する方法を示します。CLK_FLL_CONFIG4 の CCO_LIMIT フィールドを使用して、トリム調整の上限を設定できます。

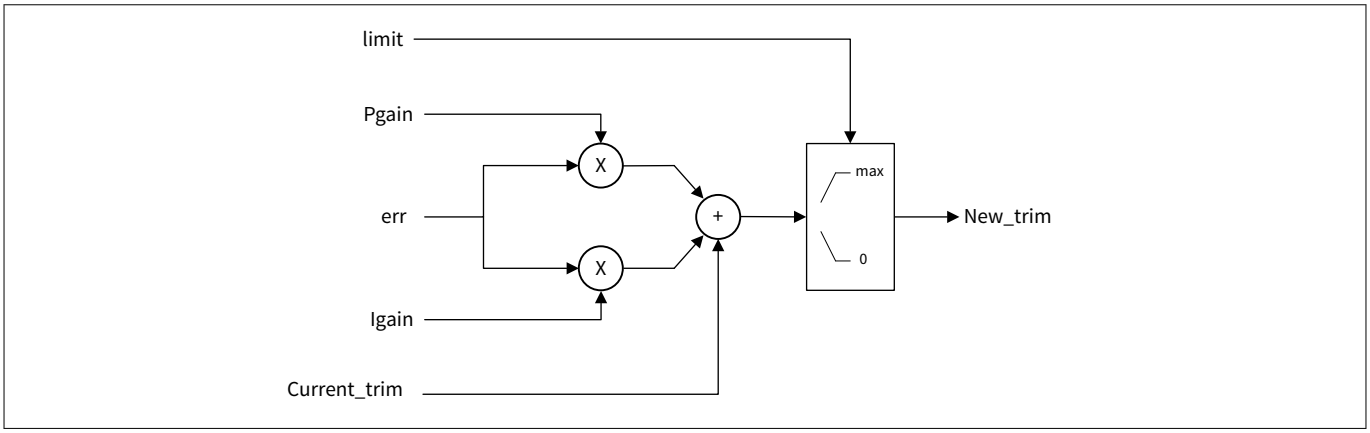


図 67　　FLL 誤差訂正図

FLL は、誤差項と CLK_FLL_CONFIG2 レジスタの LOCK_TOL フィールドを比較することで、「ロック」されているかどうかを判断します。誤差が LOCK_TOL より小さい場合、FLL はロックされたとみなされます。

トリムを調整するたびに、FLL は新たな測定を行う前に一定数の基準クロックを待つようにプログラムできます。待機する基準クロック数は CLK_FLL_CONFIG3 の SETTLING_COUNT フィールドで設定します。FLL が新しいカウントの前に～1 µs 待つように設定します。したがって、8 MHz IMO を基準として使用する場合は、このフィールドを'8'にプログラムする必要があります。

FLL を設定する際には、ロック時間と精度という 2 つの重要な要素を考慮する必要があります。正確さとは、意図した出力周波数に近いことです。これら 2 つの数値は、REF_DIV の値を介して互いに反比例の関係にあります。REF_DIV の値が大きいほど精度が高くなり、REF_DIV の値が小さいほどロック時間が短くなります。

前述の例では、8 MHz の IMO が基準として使用され、希望する FLL 出力は 100 MHz でした。この例では、1 つの基準クロックは 12.5CCO クロックになります。REF_DIV の値が'1'に設定されている場合、FLL_MULT は'13'または'12'に設定されなければいけません。この結果、CCO 出力は 96 MHz または 104 MHz となり、希望する 100 MHz との誤差は 4 パーセントとなります。したがって、これを改善する最善の方法は、REF_DIV を大きくすることです。しかし、REF_DIV が大きいほど、各測定サイクルが長くなり、ロック時間が長くなります。この例では、REF_DIV は 146 に設定されています。これは、各測定サイクルに 146 * (1/8 MHz) = 18.25 µs を要することを意味し、REF_DIV が 1 に設定されている場合は、各測定サイクルに 1 * (1/8 MHz) 0.125 µs を要します。

REF_DIV の値が低い場合のもう一つの問題は、LOCK_TOL の最小値が 1 のため、CCO の出力が±1 の誤差を持つ可能性があることです。REF_DIV = 1、FLL_MULT = 13 の例では、MULT 値は実際には 12、13、14 でもロックされます。これは、FLL の出力が 96 MHz から 112 MHz の間で変動する可能性があることを意味します。

そのため、ロック時間の短縮と、より正確な FLL 出力のどちらかを選択する必要があります。そのための最大の変更点は、REF_DIV の値です。CLK_FLL_STATUS レジスタは FLL の状態をチェックします。

FLL の有効化および無効化

FLL は、ディープスリープの有効化、無効化、およびディープスリープの開始/終了時に、ファームウェアによるシーケンス処理を必要とします。

FLL を有効に設定するためには、以下の手順に従ってください。

1. CLK_FLL_CONFIG4.CCO_ENABLE = 1 を書き込んで CCO を有効にし、CLK_FLL_STATUS.CCO_READY == 1 になるまで待ちます。
2. 基準クロックが安定し、CLK_FLL_CONFIG3.BYPASS_SEL = FLL_REF であることを確認します。
3. FLL_ENABLE = 1 と書き込み、CLK_FLL_STATUS.LOCKED == 1 になるまで待ちます。
4. CLK_FLL_CONFIG3.BYPASS_SEL = FLL_OUT を書き込み、FLL 出力に切り替えます。

FLL を無効に設定するためには、以下の手順に従ってください。

1. プロセッサがクロックパス 0 とは異なるクロックで動作していることを確認します。マルチプレクサが変更された場合は、FLL 出力が完了するまで 4 クロック サイクル待ちます。
2. CLK_FLL_CONFIG3.BYPASS_SEL = FLL_REF と書き込み、書き込みが完了したことを確認するために同じレジスタを読み出します。
3. FLL リファレンス クロック サイクルを少なくとも 6 回待ち、FLL_ENABLE = 0 で無効にします。
4. CLK_FLL_CONFIG4.CCO_ENABLE = 0 を書き込んで CCO を無効にします。

ディープスリープに入る前に、上記の手順で FLL を無効に設定するか、以下の手順でディープスリープの前後に FLL の選択を解除/選択します。ディープスリープに入る前に CLK_FLL_CONFIG3.BYPASS_SEL =FLL_REF を書き込み、FLL が基準クロックを使用するように変更します。DEEPSLEEP ウェイクアップ後、CLK_FLL_STATUS.LOCKED= = 1 になるまで待ち、CLK_FLL_CONFIG3.BYPASS_SEL = FLL_OUT を書き込んで FLL 出力に切り替えます。

注: *BYPASS_SEL フィールドで FLL_AUTO オプションを使用することは推奨されません。*

19.5 クロックツリー

PSOC™ Control C3 の MCU クロックは、図 63 のようにデバイス全体に分散されています。クロックツリーは以下のとおりです。

- パスクロック

- 高周波ルートクロック (CLK_HF)
- 低周波クロック (CLK_LF)
- バックアップクロック (CLK_BAK)
- タイマークロック (CLK_TIMER)
- 電源クロック (CLK_PWR)

19.5.1 クロックパス

PSOC™ Control C3 MCU には 7 つのクロックパスがあります。CLK_PATH0 は FLL、CLK_PATH1 は DPLL0、CLK_PATH2 は DPLL1、CLK_PATH3、CLK_PATH4、CLK_PATH5 と CLK_PATH6 は高周波数ルートクロックへの直接接続です。FLL と DPLL が不要な場合は、バイパスできます。これらのパスは、高周波クロックルートの入力ソースとなります。

各クロック パスには、どのソース クロックがそのパスのクロックとなるかを決定するための Mux (マルチプレクサ) があります。この設定は CLK_PATH_SELECTx レジスタで行われます。

表 110 クロック パス ソースの選択

名称	説明
PATH_MUX[2:0]	CLK_PATHx のソースを選択します。 0: IMO - 内部主発振器 1: EXT_CLK - 外部クロック端子 2: ECO - 外部水晶発振器 3: 予約済み使用禁止 4: DSI_MUX 5: 予約済み使用禁止 6: IHO - 内部高速発振器 7: 予約済み使用禁止

注: もともと選択されていたクロックから切り替えるには、4 サイクルのクロックが必要です。この間、元のクロックを無効にしないでください。

DSI mux は CLK_DSI_SELECTx レジスタによって設定されます。表 111 に DSI mux ソースの選択を示します。

表 111 DSI mux ソースの選択

名称	説明
DSI_MUX[4:0]	DSI_MUXx のソースを選択します。 0～15: 予約済み使用禁止 16: ILO - 内部低速発振器 17: WCO - 時計用水晶発振器 18～31: 予約済み使用禁止

19.5.2 高周波ルートクロック (CLK_HF)

PSOC™ Control C3 MCU には 7 つの高周波ルートクロックがあります (CLK_HF[0～6])。表 112 に示すように、各 CLK_HF には指定のデスティネーションがデバイス上にあります。ペリフェラルクロック グループの詳細についてはペリフェラルクロック分周器グループの章を参照してください。

表 112 CLK_HF デスティネーション

名称	説明
CLK_HF[0]	CPU、PERI、AHB インフラストラクチャー、ペリフェラル クロック グループ 0 (デバッグトレース)、ペリフェラルクロックグループ 2 (CRYPTO)
CLK_HF[1]	LPCOMP、ペリフェラルクロック グループ 1 (SMART IO)、ペリフェラルクロック グループ 3 (IOSS)
CLK_HF[2]	ペリフェラル クロック グループ 4 (CAN、SCB [0〜4])
CLK_HF[3]	HPPASS、ペリフェラル クロック グループ 5 (TCPWM、HPPASS CSG)
CLK_HF[4]	ペリフェラルクロック グループ 6 (SCB[5])
CLK_HF[5]	clk_ext ピンのクロック出力 (出力として使用する場合)
CLK_HF[6]	CSV

各高周波ルートクロックには、そのソースと分周器を決定するための Mux があります。ルート ソースは表 113 に示す CLK_DIRECT_SELECT[i]レジスタの DIRECT_MUX ビットによって IMO または ROOT_MUX となります。

表 113 クロック ルート ダイレクト セレクト レジスタ

名称	説明
DIRECT_MUX[8]	CLK_HF ダイレクト セレクト Mux 設定 0: IMO 1: ROOT_MUX (CLK_ROOT_SELECT レジスタ参照)

ROOT_MUX と predivider の設定は表 114 に示す CLK_ROOT_SELECT[i]レジスタで行います。

表 114 クロック ルート セレクト レジスタ

名称	説明
ROOT_MUX[3 : 0]	CLK_HF 入力クロック パスの選択: 0: CLK_PATH0 を選択 1: CLK_PATH1 を選択 2: CLK_PATH2 を選択 3: CLK_PATH3 を選択 4: CLK_PATH4 を選択 5: CLK_PATH5 を選択 6: CLK_PATH6 を選択 [7 : 15]: 予約済み

(続く)

表 114 (続き) クロック ルート セレクト レジスタ

名称	説明
ROOT_DIV[11 : 8]	クロック ルートの分周器値を選択 0: 分割なし 1: 選択したクロック ソースを 2 分周に設定 2: 選択したクロック ソースを 3 分周に設定 3: 選択したクロック ソースを 4 分周に設定 4: 選択したクロック ソースを 5 分周に設定 5: 選択したクロック ソースを 6 分周に設定 6: 選択したクロック ソースを 7 分周に設定 7: 選択したクロック ソースを 8 分周に設定 8: 選択したクロック ソースを 9 分周に設定 9: 選択したクロック ソースを 10 分周に設定 10: 選択したクロック ソースを 11 分周に設定 11: 選択したクロック ソースを 12 分周に設定 12: 選択したクロック ソースを 13 分周に設定 13: 選択したクロック ソースを 14 分周に設定 14: 選択したクロック ソースを 15 分周に設定 15: 選択したクロック ソースを 16 分周に設定
ENABLE	このクロックルートを有効にします。HFCLK0 を除き、すべてのクロック ルートのデフォルトは無効 (ENABLE = 0) で、HFCLK0 を無効に設定できません。

注: CLK_HF[1 ～6] は有効と無効が可能です。CLK_HF[0] は CPU のソースなので常に有効です。CLK_HF[1 ～6] を有効/無効に設定するためには、CLK_ROOT_SELECT レジスタの ENABLE ビットを設定します。

19.5.3 低周波クロック (CLK_LF)

PSOC™ Control C3 MCU の低周波クロック (CLK_LF) には、ILO、WCO、ECO プリスケーラの 3 つの入力オプションがあります。CLK_LF は、マルチ カウンター ウォッチドッグ タイマー (MCWDT) と RTC のソースです。

CLK_LF のソースは CLK_SELECT レジスタの LFCLK_SEL ビットで設定されます。このフィールドへの書き込みは、WDT_LOCK レジスタを使って WDT をアンロックしない限り無視されます。詳細は、ウォッチドッグ タイマーの章を参照してください。元々選択されていたクロックから切り替わるには、4 サイクルかかります。この間、元のクロックを無効にしないでください。

表 115 LFCLK 入力選択ビット LFCLK_SEL

名称	説明
LFCLK_SEL[2:0]	LFCLK 入力クロックの選択 0: ILO - 内部低速発振器 1: WCO - 時計水晶発振器バックアップ ドメインが存在し、適切に設定されていることが必要 (外部時計用水晶を使用している場合、それも含む) 2～4: 予約済み 5: ECO_PRESCALER - プリスケーリング後の外部水晶発振器。DEEPSLEEP または HIBERNATE モードでは動作しません。ACTIVE/SLEEP モードのみで動作するアプリケーション向け。このオプションは、製品に ECO が存在する場合にのみ有効です。 6: 予約済み

19.5.4 バックアップクロック (CLK_BAK)

バックアップ クロックは、バックアップ ドメイン (特に RTC) のクロックとして使用されます。詳細は、バックアップの章を参照してください。

19.5.5 タイマークロック (CLK_TIMER)

タイマー クロック (CLK_TIMER) は、CPU SYSTICK タイマーのクロック ソースとして使用できます。タイマー クロックは、IMO に由来する 1 MHz (固定) クロックとして標準化されており、アクティブ/スリープ モードで常に動作しています。ディープスリープなどの他のモード中はオフになります。PSOC™ Control C3 MCU には事前分周器がなく、CLK_IMO から 8 分周された 1 MHz の固定周波数のみが供給されます。SRSS 制約は、関連する内部クロックを自動的に作成します。

19.5.6 電源クロック (CLK_PWR)

電源クロック (CLK_PWR) は、Arm® パワーアーキテクチャコンポーネント、Power Police Unit (PPU) および Power Dependency Control Matrix (PDCM) のクロックに使用されます。CLK_PWR は常にアクティブ/スリープ モードで動作しています。ディープスリープモードなど、他のモード中はオフになります。

電源クロックのソースは IMO または IHO のいずれかであり、PWRMODE_CLK_SELECT レジスタの CLK_PWR_MUX ビットで設定されます。分周値はこのレジスタの CLK_PWR_DIV ビットによって設定されます。

表 116 電力モードクロック選択レジスタ

名称	説明
CLK_PWR_DIV[7 : 0]	CLK_PWR は、CLK_PWR_MUX 選択を (CLK_PWR_DIV+1) で分周することによって生成されます。
CLK_PWR_MUX[17 : 16]	電源クロックのソースを選択: 0: IMO 1: IHO 2: 予約済み

19.6 CLK_HF 分配

PSOC™ Control C3 MCU には複数の CLK_HF があり、CPU サブシステム、SRSS、ペリフェラルを接続しています。また、CLK_HF の一部は CSV に接続されます。詳細については高周波ルートクロック (CLK_HF)の章を参照してください。いくつかのペリフェラルは、ペリフェラルのクロック分周器によって CLK_HF に接続されます (詳細は、ペリフェラルクロック分周器の章を参照)。

以下の表は、CLK_HF 分配網の詳細と、異なるアクティブ モード下での CPU とペリフェラルの最大周波数の一覧です。PSOC™ Control C3 MCU は 4 つのアクティブ モードに対応しています。超低消費電力 (ULP) モード (0.9 V)、中周波 (MF) モード (1.0 V)、低消費電力 (LP) モード (1.1 V)、オーバードライバ (OD) モード (1.2 V)。電力モードの詳細については、デバイスの電力モードの章を参照してください。

表 117 クロック分配とアクティブ モードの性能パラメーター

CLK_HF	ペリフェラル	推奨ソース	最大許容クロック周波数 [1]			
			OD (1.2 V)	LP (1.1 V)	MF (1.0 V)	ULP (0.9 V)
CLK_HF0	CPU, PERI, SRSS, CORDIC, CRYPTO	DPPL0	180 MHz	150 MHz	70 MHz	50 MHz
	ペリフェラル クロック グループ 0 (デバッグ トレース)		90 MHz	75 MHz	35 MHz	25 MHz
CLK_HF1	IOSS、LPCOMP、ペリフェラルクロック グループ 1 (SMARTIO)	DPLL0	180 MHz	150 MHz	70 MHz	50 MHz
CLK_HF2	ペリフェラル クロック グループ 4 (CAN、SCB[0:4])	FLL	100 MHz	100 MHz	25 MHz	25 MHz
CLK_HF3	HPPASS[2]、ペリフェラル クロック グループ 5 (TCPWM、HPPASS CSG)	DPLL1	240 MHz	200 MHz	120 MHz	60 MHz
CLK_HF4	ペリフェラル クロック グループ 6 (SCB[5])	DPLL1 (高速 SPI 用)	240 MHz	200 MHz	120 MHz	60 MHz
		FLL (UART/I2C 用)	100 MHz	100 MHz	25 MHz	25 MHz
CLK_HF5	EXT_CLK_OUT	すべて	80 MHz	80 MHz	80 MHz	50 MHz
CLK_HF6	CSV	すべて	8 MHz	8 MHz	8 MHz	8 MHz

1) 対応するクロックソース (DPLL/FLL + 分周器) 後の最大クロック周波数。すべての内部誤差と影響は、これらの周波数でカバーされます。
2) HPPASS の最小許容クロック周波数は 60 MHz

19.7 ペリフェラルクロック分周器

SCB や TCPWM などの PSOC™ Control C3 MCU ペリフェラルはクロックを必要とします。これらのペリフェラルは、ペリフェラルクロック分周器によってのみクロック供給されます。PSOC™ Control C3 MCU は 24 のペリフェラルクロック (PCLK) 分周器を持っています。これらの分周器の出力は、グループ内のどのペリフェラルにもルーティングできます。

クロックを分周させて、アナログ ペリフェラルおよびデジタル ペリフェラル用同期クロックを生成できます。以下の整数および分数クロック分周器が用意されています。

- 13 の 8 ビット整数クロック分周器
- 5 つの 16 ビット整数クロック分周器
- 4 つの 16.5 分数クロック分周器 (16 整数ビット、5 分数ビット)
- 2 つの 24.5 分数クロック分周器 (24 整数ビット、5 分数ビット)

19.7.1 分数クロック分周器

分数クロック分周器により、クロック除数には 0～31/32 の分数があり得ます。例えば 16.5 ビット分数クロック分周器の整数分周値が 3 の場合、その分周器は 48 MHz の CLK_PERI から 16 MHz のクロックを生成します。例えば 16.5 ビット分数クロック分周器の整数分周値が 4 の場合、その分周器は 48 MHz の CLK_PERI から 12 MHz のクロックを生成します。16.5 ビット分数クロック分周器の整数分周値が 3 で、分数分周値が 16 の場合、48 MHz の CLK_PERI から 48/ (3 + 16/32) = 48/3.5 = 13.7 MHz のクロックを生成します。13.7 MHz のクロック周期のすべてが等しいというわけではなく、16 MHz のものもあれば 12 MHz のものもあり、平均すると 13.7 MHz となります。

分数クロック分周器は、高精度のクロックが必要な場合に有用です (例えば、UART/SPI シリアル インターフェース用)。低ジッタークロックが必要な場合、分周器は使用しません。なぜなら、クロック周期は CLK_PERI サイクルの 1 周期分のジッターを持つからです。

19.7.2 ペリフェラル クロック分周器のコンフィギュレーション

DIV_CMD、DIV_8_CTL、DIV_16_CTL、DIV_16_5_CTL、DIV_24_5_CTL、CLOCK_CTL レジスタです。

まず、DIV_8_CTL、DIV_16_CTL、DIV_16_5_CTL、DIV_24_5_CTL レジスタを使用してクロック分周器を設定します。例えば、8 ビット分周器が 8 つあるので、DIV_8_CTL レジスタは 8 つあります。これらのレジスタで、整数分周器の値を設定します。分数分周器の場合は、分数部分も設定します。

分周器が設定されたら、DIV_CMD レジスタを使用して分周器を有効にします。これは、DIV_SEL を有効にしたい分周器の番号に設定し、TYPE_SEL を分周器のタイプに設定することで行います。例えば、0 番目の 16.5 ビット分周器を有効にしたい場合は、DIV_SEL に'0'、TYPE_SEL に'2'を書き込みます。10 番目の 16 ビット分周器を有効にしたい場合は、DIV_SEL に'10'、TYPE_SEL に'1'を書き込みます。詳細については、レジスタリファレンス マニュアルを参照してください。

ペリフェラルを特定の分周器に接続するためには、CLOCK_CTL レジスタを使用します。CLOCK_CTL レジスタは、ペリフェラルクロック分周器グループの章で述べた分周器グループの各 PCLK に対応しています。例えば、グループ 5、出力 3 の PCLK に対して 2 番目の 8 ビット分周器を選択するためには、PERI_PCLK_GR5_CLOCK_CTL3 レジスタで DIV_SEL を'2'に、TYPE_SEL を'1'に設定します。また、'Output'の順序は PERI_PCLK_GRx_CLOCK_CTLy レジスタの y の順序と一致します。

位相整合分周器

特定のユースケースでは、位相整合クロックを生成します。例えば、48 MHz の CLK_PERI から 24 MHz と 12 MHz の 2 つのゲーテッド クロックが生成されるとします。位相整合を考慮しない場合、生成されるゲーテッド クロックは以下のようになります。

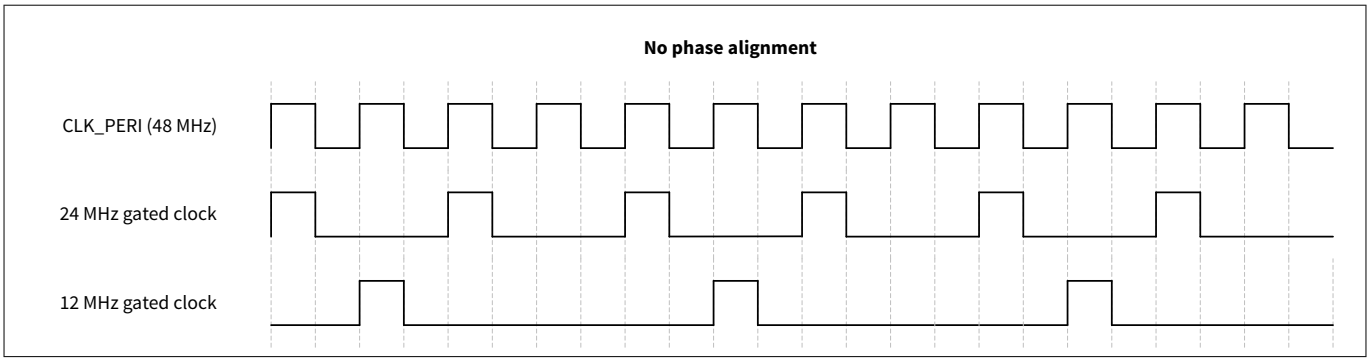


図 68　　非位相整合クロック分周器

これらのクロック信号は、これら 2 つのクロックに実装されるロジック機能によって、許容できる場合とそうでない場合があります。2 つのクロックドメインが互いに通信し、遅いクロックドメイン (12 MHz) がそのクロックの High/'1'パルスが高いクロックドメイン (24 MHz) の High/'1'位相パルスと一致すると仮定する場合、位相のずれは許容できません。これに対処するため、分周器が他の (有効な) クロック分周器の Y と位相整合クロック信号を生成するようにすることが可能です。したがって、(有効な) 分周器 x が 24 MHz クロックの生成に使用される場合、分周器 y は分周器 x に位相を合わせ、12 MHz クロックの生成に使用できます。整合されたクロックは以下のように表示されます。

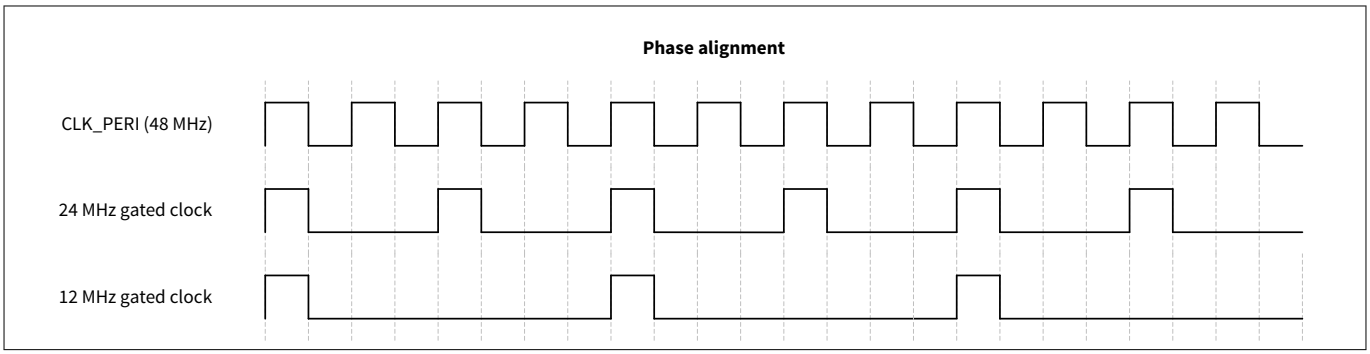


図 69　　位相整合クロック分周器

位相整合は、分周器の分数値に対しても機能します。(有効な) 分周器 x が 38.4 MHz クロックの生成 (1 8/32 で分周) に使用される場合、分周器 y は分周器 x に位相を合わせ、19.2 MHz クロックの生成 (2 16/32 で分周) に使用できます。生成されたゲーテッド クロックは以下のようになります。

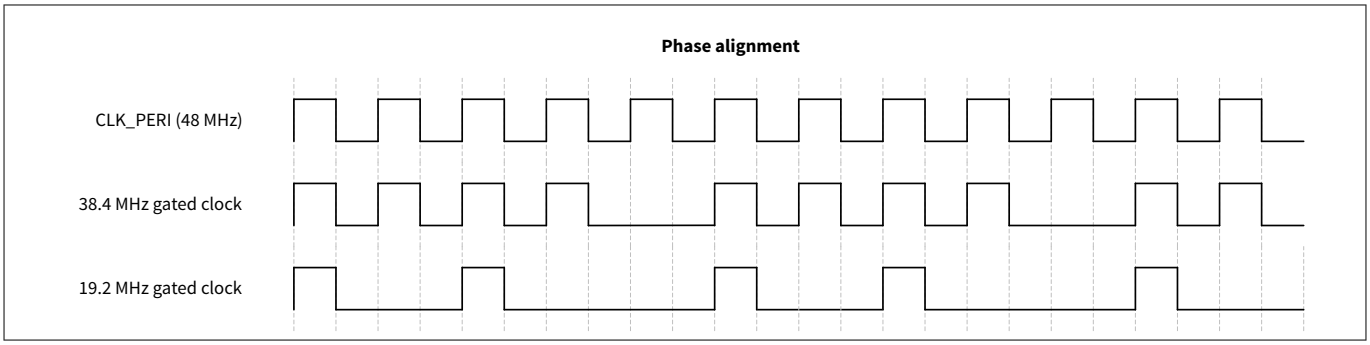


図 70　　位相整合分数除算器

分周器の位相整合には、位相整合先の分周器がすでに有効になっている必要があります。そのためには、特定の順番で分周器を有効に設定する必要があります。

位相整合は、分周カウンターの開始タイミングをハードウェアで制御することで実装されます。分周器が有効になると、分周カウンターは'0'に設定されます。分周カウンターは、それが位相合わせされている分周器が整数カウンター値'0'を持つときにのみ、'0'からプログラムされた整数分周値および分数分周値へのインクリメントを開始します。

分周器とクロック マルチプレクサのコントロール レジスタ フィールドは、すべてディープスリープ電力モード中も保持されることに注意してください。しかし、整数および分数クロック分周器を実装するために使用される分周カウンターはそうではありません。これらのカウンターは、ディープスリープ モード中は'0'に設定されます。したがって、ディープスリープからアクティブモードに移行する際には、設計上、すべての分周器 (およびクロック信号) が有効になり、位相が整合します。

位相整合は、クロックを有効に設定する前に DIV_CMD レジスタの PA_DIV_SEL ビットと PA_DIV _TYPE ビットを設定することで行えます。例えば、4 番目の 8 ビット分周器を 3 番目の 16 ビット分周器に合わせるには、DIV_SEL を'4'、TYPE_SEL を'0'、PA_DIV_SEL を'3'、PA_TYPE_SEL を'1'に設定します。

19.7.3 ペリフェラルクロック分周器グループ

PSOC™ Control C3 MCU には 7 つの PCLK クロック分周グループがあり、各グループの出力は特定のペリフェラルに接続されています。ペリフェラルの完全なリストについては、以下のクロック グループの表を参照してください。

ペリフェラル クロック グループ 0

- ルートクロック: CLK_HF[0]
- 分周器数: 1
- 出力数: 1
- 8 ビット整数分周カウント: 0
- 16 ビット整数分周カウント: 0
- 16.5 ビットの分数分周カウント: 0
- 24.5 ビットの分数分周カウント: 1

表 118 ペリフェラルクロック グループ 0: 分周クロック出力

出力番号	宛先
0	デバッグトレースクロック (debug600.clock_trace_in)

ペリフェラル クロック グループ 1

- ルートクロック: HF[1]
- 分周器数: 1
- 出力数: 7
- 8 ビット整数分周カウント: 1
- 16 ビット整数分周カウント: 0
- 16.5 ビットの分数分周カウント: 0
- 24.5 ビットの分数分周カウント: 0

表 119 ペリフェラル クロック グループ 1: 分周クロック出力

出力番号	宛先
0	SMARTIO (ioss.clock_smartio_pclk_pos_en[0])
1	SMARTIO (ioss.clock_smartio_pclk_pos_en[1])
2	SMARTIO (ioss.clock_smartio_pclk_pos_en[2])
3	SMARTIO (ioss.clock_smartio_pclk_pos_en[3])
4	SMARTIO (ioss.clock_smartio_pclk_pos_en[5])
5	SMARTIO (ioss.clock_smartio_pclk_pos_en[6])
6	SMARTIO (ioss.clock_smartio_pclk_pos_en[9])

ペリフェラル クロック グループ 2

- ルートクロック: HF[0]
- 分周器数: 0
- 出力数: 1
- 8 ビット整数分周カウント: 0
- 16 ビット整数分周カウント: 0
- 16.5 ビット分数分周カウント: 0
- 24.5 ビット分数分周カウント: 0

表 120 ペリフェラル クロック グループ 2: 分周クロック出力

出力番号	宛先
0	CRYPTO (cryptolite.clk_hf)

注: CRYPTO ペリフェラルは CLK_HF[0] に直接接続します。

ペリフェラル クロック グループ 3

- ルートクロック: HF[1]
- 分周器数: 0
- 出力数: 1
- 8 ビット整数分周カウント: 0
- 16 ビット整数分周カウント: 0
- 16.5 ビット分数分周カウント: 0
- 24.5 ビット分数分周カウント: 0

表 121 ペリフェラル クロック グループ 3: 分周クロック出力

出力番号	宛先
0	IOSS (ioss.clk_hf)

注: I/O サブシステムは CLK_HF[1]に直接接続します。

ペリフェラル クロック グループ 4

- ルートクロック: HF[2]
- 分周器数: 7
- 出力数: 7
- 8 ビット整数分周カウント: 2
- 16 ビット整数分周カウント: 1
- 16.5 ビット分数分周カウント: 3
- 24.5 ビット分数分周カウント: 1

表 122 ペリフェラル クロック グループ 4: 分周クロック出力

出力番号	宛先
0	SCB0 (scb[0].clock_scb_en])
1	SCB1 (scb[1].clock_scb_en])

(続く)

表 122 (続き) ペリフェラル クロック グループ 4: 分周クロック出力

出力番号	宛先
2	SCB2 (scb[2].clock_scb_en])
3	SCB3 (scb[3].clock_scb_en])
4	SCB4 (scb[4].clock_scb_en])
5	CAN 0 チャネル 0 (canfd[0].clock_can_en[0])
6	CAN 0 チャネル 1 (canfd[0].clock_can_en[1])

ペリフェラル クロック グループ 5

- ルートクロック: HF[3]
- 分周器数: 14
- 出力数: 22
- 8 ビット整数分周カウント: 10
- 16 ビット整数分周カウント: 4
- 16.5 ビット分数分周カウント: 0
- 24.5 ビット分数分周カウント: 0

表 123 ペリフェラルクロック グループ 5: 分周クロック出力

出力番号	宛先
0	TCPWM グループ 0 32 ビット カウンター 0 (tcpwm[0].clock_counter_en[0])
1	TCPWM グループ 0 32 ビット カウンター 1 (tcpwm[0].clock_counter_en[1])
2	TCPWM グループ 0 32 ビット カウンター 2 (tcpwm[0].clock_counter_en[2])
3	TCPWM グループ 0 32 ビット カウンター 3 (tcpwm[0].clock_counter_en[3])
4	TCPWM グループ 1 16 ビット カウンター 0 (tcpwm[0].clock_counter_en[256])
5	TCPWM グループ 1 16 ビット カウンター 1 (tcpwm[0].clock_counter_en[257])
6	TCPWM グループ 1 16 ビット カウンター 2 (tcpwm[0].clock_counter_en[258])
7	TCPWM グループ 1 16 ビット カウンター 3 (tcpwm[0].clock_counter_en[259])
8	TCPWM グループ 1 16 ビット カウンター 4 (tcpwm[0].clock_counter_en[260])
9	TCPWM グループ 1 16 ビット カウンター 5 (tcpwm[0].clock_counter_en[261])
10	TCPWM グループ 1 16 ビット カウンター 6 (tcpwm[0].clock_counter_en[262])
11	TCPWM グループ 1 16 ビット カウンター 7 (tcpwm[0].clock_counter_en[263])

(続く)

表 123 (続き) ペリフェラルクロック グループ 5: 分周クロック出力

出力番号	宛先
12	TCPWM グループ 2 16 ビット カウンター 0 (tcpwm[0].clock_counter_en[512])
13	TCPWM グループ 2 16 ビット カウンター 1 (tcpwm[0].clock_counter_en[513])
14	TCPWM グループ 2 16 ビット カウンター 2 (tcpwm[0].clock_counter_en[514])
15	TCPWM グループ 2 16 ビット カウンター 3 (tcpwm[0].clock_counter_en[515])
16	TCPWM グループ 2 16 ビット カウンター 4 (tcpwm[0].clock_counter_en[516])
17	TCPWM グループ 2 16 ビット カウンター 5 (tcpwm[0].clock_counter_en[517])
18	TCPWM グループ 2 16 ビット カウンター 6 (tcpwm[0].clock_counter_en[518])
19	TCPWM グループ 2 16 ビット カウンター 7 (tcpwm[0].clock_counter_en[519])
20	TCPWM MOTIF 0 (tcpwm[0].clock_motif_en[32])
21	HPPASS CSG クロック (pass.clock_csg)

ペリフェラル クロック グループ 6

- ルートクロック: CLK_HF[4]
- 分周器数: 1
- 出力数: 1
- 8 ビット整数分周カウント: 0
- 16 ビット整数分周カウント: 0
- 16.5 ビットの分数分周カウント: 1
- 24.5 ビットの分数分周カウント: 0

表 124 ペリフェラルクロック グループ 6: 分周クロック出力

出力番号	宛先
0	SCB5 (scb[5].clock_scb_en)

19.8 クロック キャリブレーションカウンター

PSOC™ Control C3 MCU のクロッキング システムには、ハードウェア キャリブレーション カウンターが内蔵されています。これらのカウンターは、2 つのクロックソースの周波数を互いに比較するために使用します。主な使用例は、ECO のような高精度のクロックを、ILO のような低精度のクロックの測定に使用することです。この測定結果を使って ILO をトリミングします。

以下の 2 つのカウンターがあります。

1. キャリブレーション カウンター 1 はキャリブレーション クロック 1 (一般に高精度クロック) からクロックオフされ、カウントダウンします。
2. キャリブレーション カウンター 2 はキャリブレーション クロック 2 からクロックオフされ、カウントアップします。

キャリブレーション カウンター 1 が 0 になると、キャリブレーション カウンター 2 はカウントアップを停止し、その値を読み出せます。その値から、キャリブレーション用クロック 2 の周波数を以下の式で求めます。

$$Calibration\ clock\ 2\ frequency = \frac{Counter\ 2\ final\ value}{Counter\ 1\ final\ value} \times Calibration\ clock\ 1\ requency \quad (3)$$

例えば、キャリブレーション クロック 1 = 8 MHz、カウンター 1 = 1000、カウンター 2 = 5 の場合

$$Calibration\ clock\ 2\ frequency = \frac{5}{1000} \times 8\ MHz = 40\ KHz \quad (4)$$

キャリブレーション クロック 1 とキャリブレーション クロック 2 は CLK_OUTPUT_FAST レジスタで選択します。すべてのクロック ソースは、この 2 つのクロックのソースとして利用できます。CLK_OUTPUT_SLOW はクロックソースの選択にも使用されます。

キャリブレーション カウンター 1 は CLK_CAL_CNT1 レジスタにプログラムされます。キャリブレーション カウンター 2 は CLK_CAL_CNT2 レジスタで読み出せます。

キャリブレーション カウンター 1 が 0 になると、CLK_CAL_CNT1 レジスタに CAL_COUNTER_DONE ビットが設定されます。

19.9 クロック監視機能 (CSV: Clock Supervisor)

クロック監視 (CSV) 回路は、監視クロックの周波数が許容周波数ウィンドウ内にあるかどうかをチェックします。モニタークロックが停止したり、起動しなかったりすると、低周波として検出されます。

PSOC™ Control C3 MCU クロック システムには、すべての高周波ルート クロック (CLK_HF)、基準クロック (CLK_REF_HF)、低周波クロック (CLK_LF)、内部低速発振器 (ILO) をモニターするための CSV_HF、CSV_REF、CSV_LF、CSV_ILO があります (CSV 接続については図 63 を参照)。本 CSV には、以下の機能があります。

- すべての CLK_HF は CSV を持っています。すべての CLK_HF CSV は同じ基準クロック (CLK_REF_HF) を使用します。基準クロックはアクティブ クロック ソース (レジスタ: CSV_REF_SEL) のいずれかを選択。通常、IMO が選択される (デフォルト)
- ILO は CLK_LF と CLK_REF_HF の両方で監視されます。CLK_LF はディープスリープの監視に必要で、CLK_REF_HF はアクティブ時の精度に必要です。
- CSV_REF は CLK_REF_HF を CLK_ILO でモニターします。
- すべての CSV_HF と CSV_REF がアクティブ ドメインにあります。
- ディープスリープ ドメインには 2 つの CSV (CSV_LF および CSV_ILO) があります。CSV_LF は、選択された CLK_LF クロックを ILO でモニターするために使用されます。CSV_ILO は、CLK_LF で ILO を監視するために使用されます。

CSV 動作

CSV 回路の基本動作原理は以下のとおりです。

- モニタークロックはモニターイベント (Period) を生成し、基準クロックは下限値と上限値を生成します。
- モニターイベントは、下限値または上限値と比較されます。
- モニターイベント≦下限値、またはモニターイベント＞上限値の場合、エラーが報告されます。

注: Period はモニタークロック数で、ターゲットは基準クロック数です。理想的な状況では、両者の期間は同じです。

図 71 に CSV 動作の信号例を示します。

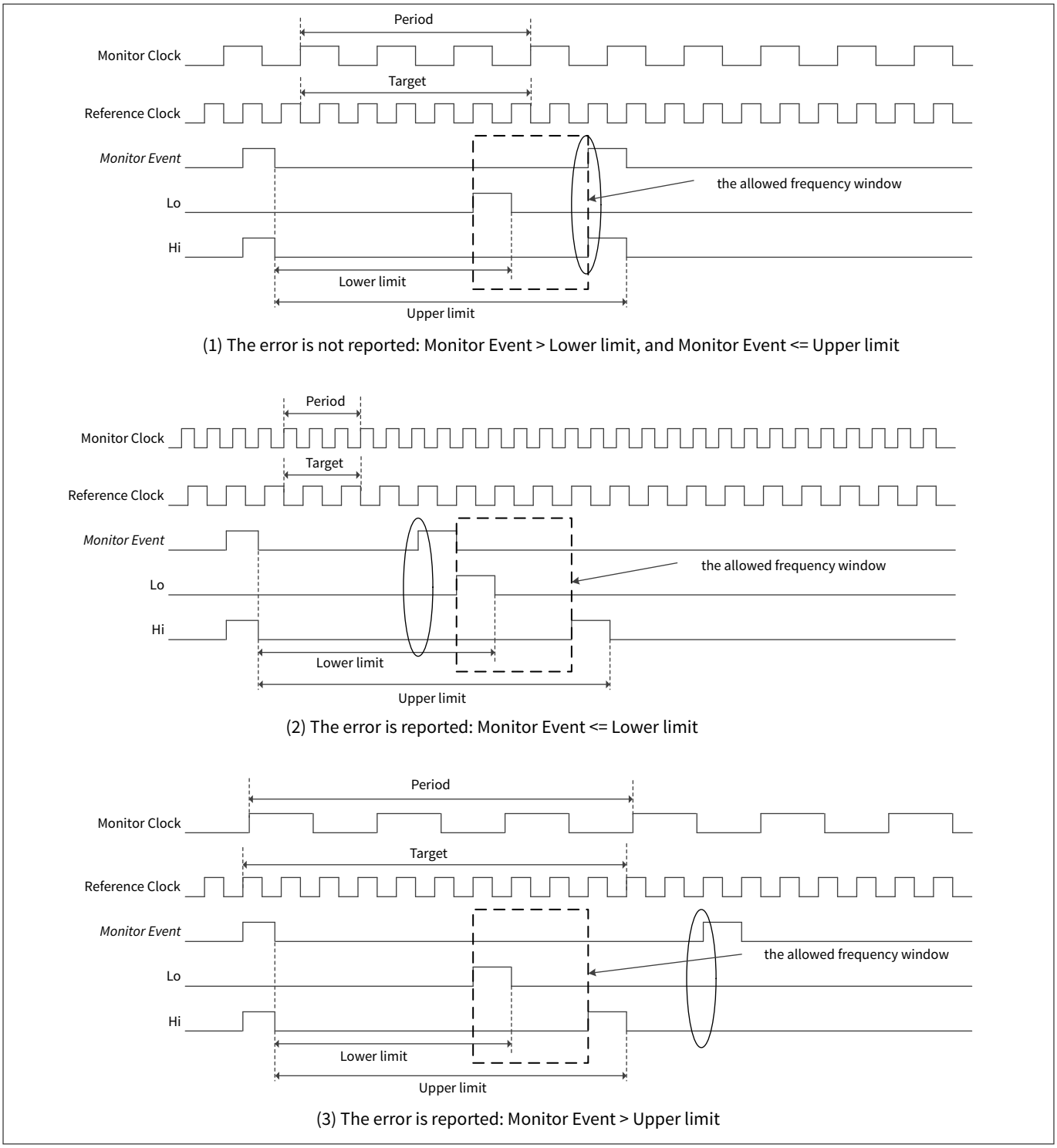


図 71　　CSV 動作の信号例

CSV 運用のポイントは以下のとおりです。

- パラメーターをチェック:
  - 基準クロック周波数と許容誤差 (整数)
  - モニタークロック周波数と必要な許容誤差 (整数%)。

  必要なモニタークロックの許容誤差は、基準クロックの許容誤差と同じか、それ以上でなければいけません。
- 以下の式で「ターゲット」を決定:

最小ターゲット = 200/基準クロック誤差
例えば、誤差が 1%の場合、ターゲットは少なくとも 200 でなければいけません。ターゲットを増やすと CSV の精度とレイテンシが向上します

- 以下の式で「周期」を決定します。
  周期 = ターゲット / (基準クロック周波数/モニタークロック周波数)
- 以下の式で Lower_limit と Upper_limit を決定します。
  - 下限値 = 周期 * ((基準クロック周波数 * (1 - 許容範囲/100)) / (モニタークロック周波数 * (1 + 許容範囲/100)))
  - 上限値 = 周期 * ((基準クロック周波数 * (1 + 許容範囲/100)) / (モニタークロック周波数 * (1 - 許容範囲/100)))
- 2 つのクロックが非同期の場合 (典型的な場合)、モニター イベントの周期は 1 サイクル変動します。
- 周波数ウィンドウは、両クロックの最大クロック許容誤差を考慮する必要があります。
- 下限値は上限値より少なくとも 1 つ小さくなければいけません。
- すべての CSV は初期状態ではオフになっており、有効に設定する前に設定が必要です。
- ディープスリープ モードの間、アクティブ ドメイン CSV は自動的に停止します。
  - ウェイクアップ後、自動的に再起動します
  - 各 CSV には、ソフトウェアでプログラム可能な起動時間があります。WCO の場合、CSV 起動時間は未使用 (STARTUP = 0)。WCO の起動後に CSV を有効に設定 (BACKUP_STATUS.WCO_OK = 1)
- すべてのアクティブ domain CSV は、Reset または Fault レポートを生成できます。
  - フォールト構造は CLK_HF0 で動作するため、CLK_HF0 CSV はリセットを使用する必要があります。
  - 他のすべての CSV は、ソフトウェアがシャットダウンできるように、フォールト レポートを使用する必要があります。
  - フォールト レポートにより割込みが発生
- ディープスリープドメイン CSV は、アクティブおよびディープスリープ中に動作し、フォールトの報告のみが可能です (リセットのオプションはありません)。
  - CSV エラー検出によりシステムが起動し (必要な場合)、フォールト レポートが可能になります。
  - フォールト レポートが割込みになります (CSV からの直接割込みはありません)
- すべての CSV はモニタークロックとは無関係に有効になります。したがって、以下に注意が必要です。
  - ソフトウェアは、モニター クロックを停止または再設定する前に CSV を無効にしてください (誤検出を避けるため)。
  - CSV を適宜再設定し、監視クロックの再起動後に再スタートさせる必要があります。
- CSV_REF_SEL レジスタは、アクティブ ドメインで CSV の基準クロックとして使用するソースを選択します。
  CSV 機能を設定するためのレジスタは以下のとおりです。
  - これらのレジスタは CSV を有効化し、CSV が有効化されたときの動作を設定できます。
    CSV_HF_CSVx_REF_CTL, CSV_REF_CSV_REF_CTL, CSV_LF_CSV_REF_CTL, CSV_ILO_CSV_REF_CTL
  - 上限値と下限値を設定できるレジスタは以下のとおりです。下限値と上限値に-1 を設定します。
    CSV_HF_CSVx_REF_LIMIT, CSV_REF_CSV_REF_LIMIT, CSV_LF_CSV_REF_LIMIT, CSV_ILO_CSV_REF_LIMIT
  - 以下のレジスタで PERIOD (周期) を設定できます。PERIOD (期間) は−1 に設定してください。
    CSV_HF_CSVx_MON_CTL, CSV_REF_CSV_MON_CTL, CSV_LF_CSV_MON_CTL, CSV_ILO_CSV_MON_CTL
- CSV 違反が発生した場合、FAULT_STRUCT_STATUS レジスタの IDX ビットに SRSS_FAULT_CSV が表示されます。CSV フォールト レポートは FAULT_STRUCT_DATA レジスタに記録されます。詳細については、フォールト監視の章を参照してください。

19.10 レジスタの概要

表 125、表 126、表 127、表 128 に、クロックシステムに関連するレジスタを示します。

表 125 PERI のクロック システム レジスタの概要

レジスタ	機能
PERI_PCLK_GRx_DIV_CMD	分周命令レジスタ
PERI_PCLK_GRx_CLOCK_CTLy	分周クロック制御レジスタ
PERI_PCLK_GRx_DIV_8_CTLy	8 ビット分周クロック制御レジスタ
PERI_PCLK_GRx_DIV_16_CTLy	16 ビット分周クロック制御レジスタ
PERI_PCLK_GRx_DIV_16_5_CTLy	16.5 ビット分周クロック制御レジスタ
PERI_PCLK_GRx_DIV_24_5_CTLy	24.5 ビット分周クロック制御レジスタ

表 126 SRSS のクロック システム レジスタの概要

レジスタ	機能
CLK_DSI_SELECTx	クロック DSI 選択レジスタ
CLK_PATH_SELECTx	クロックパス選択レジスタ
CLK_OUTPUT_FAST	高速クロック出力選択レジスタ
CLK_OUTPUT_SLOW	低速クロック出力選択レジスタ
CLK_CAL_CNT1	クロック キャリブレーションカウンター 1 レジスタ
CLK_CAL_CNT2	クロック キャリブレーションカウンター 2 レジスタ
CLK_DIRECT_SELECTx	クロック ルート ダイレクト セレクト レジスタ
CLK_ROOT_SELECTx	クロック ルート セレクト レジスタ
CLK_SELECT	クロック選択レジスタ
CLK_IMO_CONFIG	IMO コンフィギュレーション レジスタ
CLK_ECO_PRESCALE	ECO プリスケーラー コンフィギュレーション レジスタ
CLK_ECO_CONFIG	ECO コンフィギュレーション レジスタ
CLK_ECO_CONFIG2	ECO コンフィギュレーション レジスタ 2
CLK_ECO_STATUS	ECO ステータスレジスタ
CLK_FLL_CONFIG	FLL コンフィギュレーション レジスタ
CLK_FLL_CONFIG1	FLL コンフィギュレーション レジスタ 1
CLK_FLL_CONFIG2	FLL コンフィギュレーション レジスタ 2
CLK_FLL_CONFIG3	FLL コンフィギュレーション レジスタ 3
CLK_FLL_CONFIG4	FLL コンフィギュレーション レジスタ 4
CLK_FLL_STATUS	FLL ステータスレジスタ
CLK_ILO_CONFIG	ILO コンフィギュレーション レジスタ
CLK_IHO_CONFIG	IHO コンフィギュレーション レジスタ
CLK_DPLL_LPx_CONFIG	DPLL コンフィギュレーション レジスタ
CLK_DPLL_LPx_CONFIG2	DPLL コンフィギュレーション レジスタ 2

(続く)

表 126 (続き) SRSS のクロック システム レジスタの概要

レジスタ	機能
CLK_DPLL_LPx_CONFIG3	DPLL コンフィギュレーション レジスタ 3
CLK_DPLL_LPx_CONFIG4	DPLL コンフィギュレーション レジスタ 4
CLK_DPLL_LPx_CONFIG5	DPLL コンフィギュレーション レジスタ 5
CLK_DPLL_LPx_CONFIG6	DPLL コンフィギュレーション レジスタ 6
CLK_DPLL_LPx_CONFIG7	DPLL コンフィギュレーション レジスタ 7
CLK_DPLL_LPx_STATUS	DPLL ステータスレジスタ
CSV_HF_CSVx_REF_CTL	CLK_HFx 用クロック監視リファレンス コントロール レジスタ
CSV_HF_CSVx_REF_LIMIT	CLK_HFx 用クロック監視リファレンス リミット レジスタ
CSV_HF_CSVx_MON_CTL	CLK_HFx 用クロック監視モニター コントロール レジスタ
CSV_REF_SEL	アクティブドメインの CSV 基準クロックを選択
CSV_REF_CSV_REF_CTL	CLK_REF_HF 用クロック監視リファレンス コントロール レジスタ
CSV_REF_CSV_REF_LIMIT	CLK_REF_HF 用クロック監視リファレンス リミット レジスタ
CSV_REF_CSV_MON_CTL	CLK_REF_HF 用クロック監視モニター コントロール レジスタ
CSV_LF_CSV_REF_CTL	CLK_LF 用クロック監視リファレンス コントロール レジスタ
CSV_LF_CSV_REF_LIMIT	CLK_LF 用クロック監視リファレンス リミット レジスタ
CSV_LF_CSV_MON_CTL	CLK_LF 用クロック監視モニター コントロール レジスタ
CSV_ILO_CSV_REF_CTL	CLK_ILO 用クロック監視リファレンス コントロール レジスタ
CSV_ILO_CSV_REF_LIMIT	CLK_ILO 用クロック監視リファレンス リミット レジスタ
CSV_ILO_CSV_MON_CTL	CLK_ILO 用クロック監視モニター コントロール レジスタ

表 127 BACKUP のクロック システム レジスタの概要

レジスタ	機能
BACKUP_CTL	BACKUP 制御レジスタ

表 128 PWRMODE のクロック システム レジスタの概要

レジスタ	機能
PWRMODE_CLK_SELECT	電力モードコンポーネント用クロック選択レジスタ

20 ウォッチドッグ タイマー

ウォッチドッグ タイマー (WDT) は、ファームウェアが予期しない実行パスに入った時、デバイスを自動的にリセットするために使用されます。有効化されている場合、リセットを回避するためにファームウェアで定期的に WDT をクリアする必要があります。放置するとタイマーの時間が経過し、デバイス リセットを発生させます。加えて、WDT は割込み要因もしくは低消費電力モードからのウェイクアップ要因として使用可能です。

PSOC™ Control C3 MCU ファミリーには、1 つのフリーランニング WDT と 1 つのマルチカウンター WDT (MCWDT) が含まれます。ウォッチドッグ システムには 4 つのカウンターがあり、フリーランニング WDT には 32 ビット カウンターが 1 つ、MCWDT には 16 ビット カウンターが 2 つと 32 ビット カウンターが 1 つあります。すべての 16 ビット カウンターと 32 ビット カウンターはウォッチドッグ デバイス リセットを生成できます。4 つのカウンターはすべて、一致イベント時に割込みを発生させられます。

20.1 機能

PSOC™ Control C3 MCU WDT はこれらの機能に対応しています。

32 ビット フリーランニング WDT×1:

- デフォルトでは、ILO を入力クロックソースとします。クロックソースは ILO と CLK_BAK から選択できます。クロックソースの詳細については、クロックシステム セクションを参照してください。
- 設定可能な間隔内にサービスが提供されない場合、デバイスがリセットされます。
- アクティブ、スリープ、ディープスリープ、ハイバネート電力モードで定期的な割込み/ウェイクアップを生成可能。

マルチ カウンター ウォッチドッグ タイマー (MCWDT):

- 設定可能な間隔内にサービスが提供されない場合、デバイスがリセットされます。
- LFCLK (ILO、WCO) を入力クロック ソースとします。
- アクティブ、スリープ、ディープスリープの電力モードで定期的な割込み/ウェイクアップ トリガーを生成可能
- 2 つの 16 ビット カウンターと 1 つの 32 ビット独立カウンター。1 つの 64 ビットまたは 48 ビット (1 つの 16 ビット独立カウンター付き)、または 2 つの 32 ビット カスケード カウンターとして設定可能です。
- 16 ビットカウンターでウィンドウモード動作に対応

20.2 アーキテクチャ

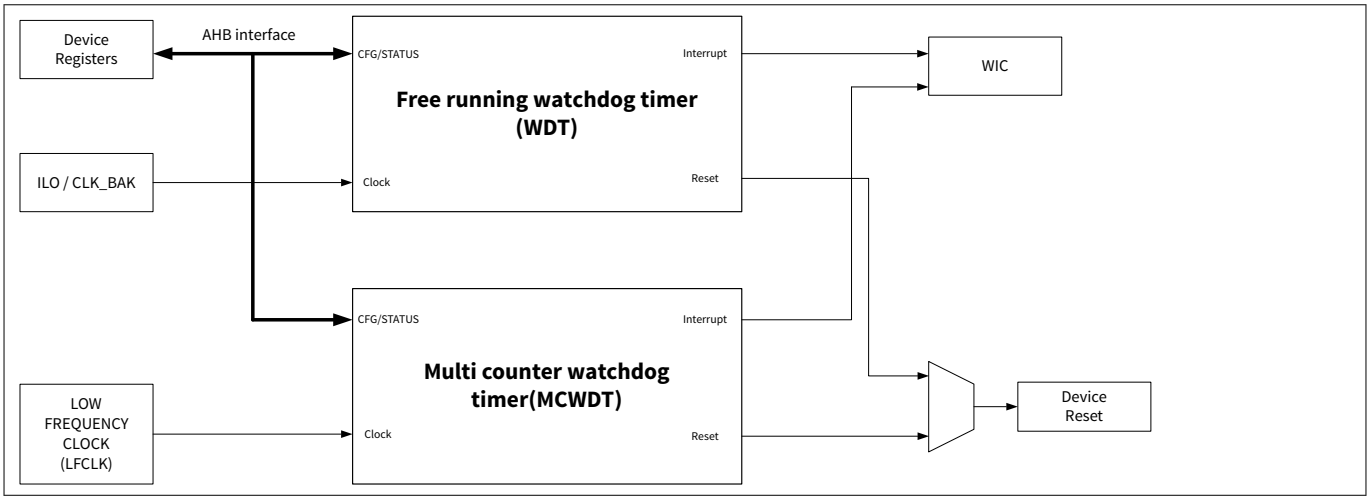


図 72 ウォッチドッグ タイマー ブロック図

20.3 ベーシック ウォッチドッグ タイマー (WDT)

20.3.1 概要

図 73 にフリーランニング WDT の機能概要を示します。WDT は、最大 32 ビットの分解能を有するフリーランニング ラップアラウンド アップカウンターです。カウンターはデフォルトで ILO によってクロックされます。クロック ソースは WDT_CTL.WDT_CLK_SEL ビットを使って CLK_BAK に変更できます。タイマーは、一致時に割込みを発生させ、3 回目のサービスが実行されない割込みでリセットイベントを発生させられます。一致比較に使われるビット数は、図 73 に示すように設定可能です。

注: BAK - 選択された CLK_BAK ソース (存在する場合)。BACKUP_CTL を参照してください。この選択は、安全性やセキュリティをウォッチドッグ タイマーに依存するアプリケーションには推奨されません。CLK_BAK の生成は WDT_LOCK によって保護されておらず、異なるセキュリティ属性を持つ可能性のある別のメモリ領域にあります。

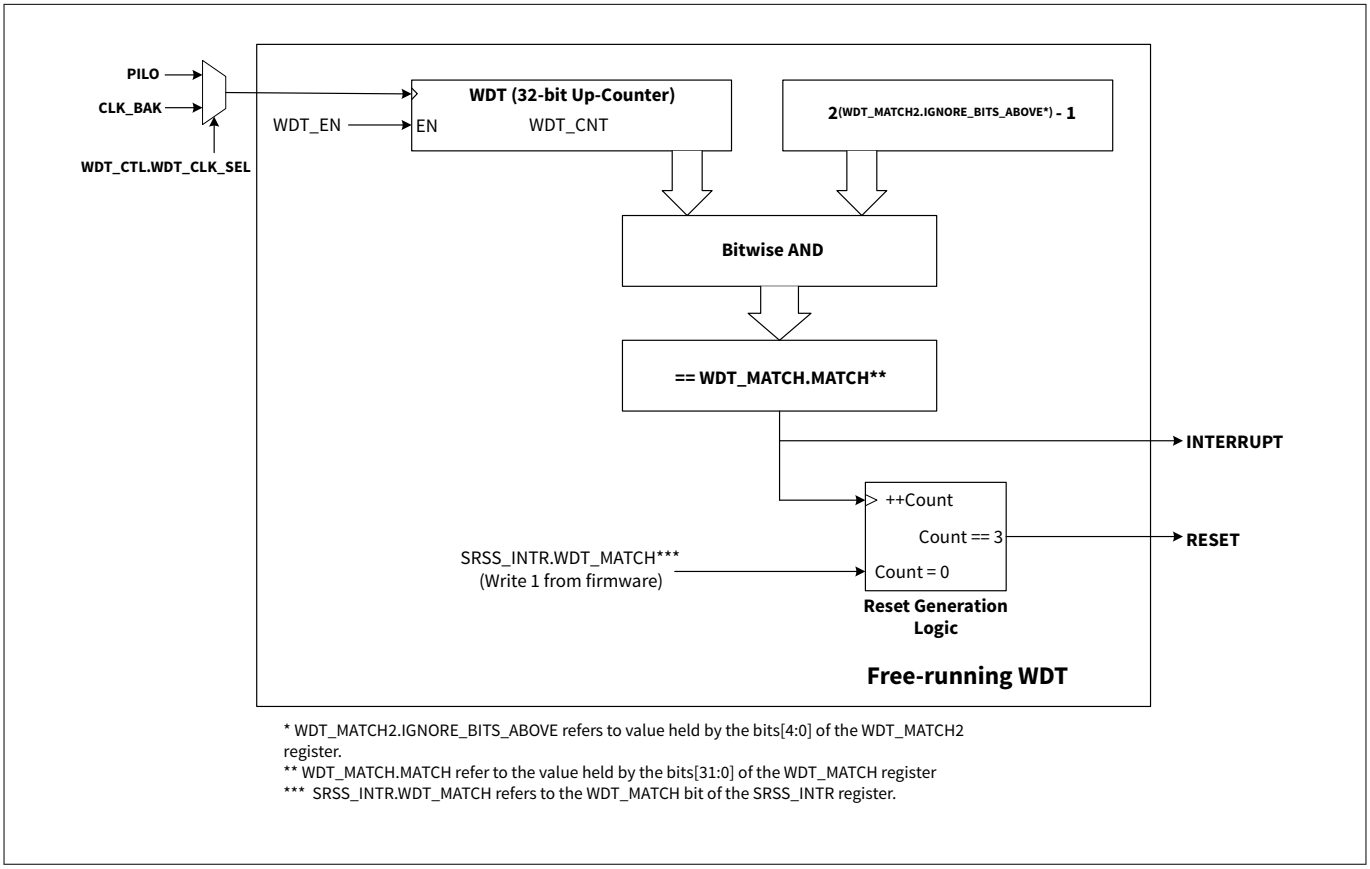


図 73 フリーランニング WDT 機能図

有効に設定すると、WDT は ILO または CLK_BAK の各立ち上りエッジでカウントアップします。カウンター値 (WDT_CNT レジスタ) が WDT_MATCH レジスタの MATCH ビット[31:0]に格納された一致値と等しい時、割込みが生成されます。一致イベントは WDT カウンターをリセットせず、WDT は 32 ビット境界 (0xFFFF FFFF) に達するまでカウントを続け、その時点で 0 に折り返してカウントアップします。一致割込みは、カウンター値が一致値と等しくなるたびに発生します。

WDT 一致割込みが発生すると SRSS_INTR レジスタの WDT_MATCH ビットが設定されます。この割込みは、同じビットに'1'を書き込むことでクリアする必要があります。割込みをクリアするとウォッチドッグがリセットされます。ファームウェアが 2 回連続で割込みをクリアしない場合、3 回目の割込みはデバイスリセットを生成します。

さらに、WDT は比較に使用するビット数を設定するオプションを提供します。IGNORE_BITS_ABOVE (WDT_MATCH2 レジスタのビット[4:0]) はこの目的に使用されます。これらのビットは、32 ビットのカウント値から無視する LSB の数を設定します。例えば、これらのビットの値が 4 に等しいとき、ビット 5 からビット 31 は無視され、ビット 0 からビット 4 が一致とみなされ、WDT カウンターは 5 ビット カウンターと同様の動作をします。WDT_CNT レジスタは 0 から (0xFFFF FFFF) (32 ビット) までカウントします。

WDT は WDT_CTL レジスタの WDT_EN ビット[0]を使用して有効または無効に設定できます。WDT_CTL[5:4]は、ILO (デフォルト) と CLK_BAK から WDT クロックソースを選択する機能を追加します。WDT_CTL レジスタは WDT 設定レジスタをロックする機構を提供します。WDT_LOCK ビット[31:30]は WDT レジスタのロック状態を制御します。これらのビットは特別なビットで、1 回の書き込みでロックを有効に設定でき、ロックを解除するためには 2 回の書き込みが必要です。WDT_LOCK ビットは WDT_EN ビット、WDT_MATCH レジスタ、CLK_ILO_CONFIG レジスタ、CLK_SELECT レジスタの LFCLK_SEL ビット[2:0]を保護します。WDT_LOCK ビットはディープスリープモードとハイバネートモードでは保持されず、ウェイクアップ後にデフォルト状態 (LOCK) にリセットされることに注意してください。したがって、ディープスリープ ウェイクアップまたはハイバネート ウェイクアップ後に WDT_LOCK ビットで保護されたレジスタを更新するためには、レジスタ更新の前に WDT UNLOCK シーケンスを発行します。

表 129 に、WDT を設定し使用するために使用される様々なレジスタとビットフィールドを示します。

表 129　　フリーランニング WDT 設定オプション

レジスタ [Bit_Pos]	Bit_Name	説明
WDT_CTL[0]	WDT_EN	WDT リセットの有効/無効を選択
		0 : WDT リセット無効
		1 : WDT リセット有効
WDT_CTL[5:4]	WDT_CLK_SEL	WDT のソースを選択
		0: ILO
		2 : CLK_BAK - 存在する場合、clk_bak ソースを選択。BACKUP_CTL を参照してください。この選択はセキュリティや安全性の確保にウォッチドッグ タイマーを使用するアプリケーションには推奨されません。clk_bak の生成は WDT_LOCK によって保護されておらず、異なるセキュリティ属性を持つ可能性のある別のメモリ領域にあります。
WDT_CTL[31:30]	WDT_LOCK	ウォッチドッグ コンフィギュレーションおよびクロック関連レジスタへの書き込みアクセスをロックまたはアンロックします。ビットが設定されると、ロックが有効になります。
		0 : 無効
		1 : ビット 0 をクリア
		2 : ビット1をクリア
		3 : ビット 0 と 1 の両方をセット (ロック有効)

(続く)

表 129 (続き) フリーランニング WDT 設定オプション

レジスタ [Bit_Pos]	Bit_Name	説明
		WDT はリセットでロックします。このフィールドはディープスリープまたはハイバネートモードでは保持されないので、これらのモードからのウェイクアップ後に WDT はロックされます。
WDT_CNT[31:0]	COUNTER	WDT カウンターの現在値
WDT_MATCH[31:0]	MATCH	ウォッチドッグ一致イベントを生成する一致値
WDT_MATCH2[4:0]	IGNORE_BITS_ABOVE	MATCH 値と比較するために考慮する LSB の数。このビットフィールドの最小値は 3 です。
SRSS_INTR[0]	WDT_MATCH	WDT 割込み要求。このビットはウォッチドッグ一致イベントが発生するたびにセットされます。このビットに‘1’を書き込むことで、WDT 割込みがクリアされます。ソフトウェアはこのビットに 1 を書き込むことで、ウォッチドッグにフィードできます。
SRSS_INTR_MASK[0]	WDT_MATCH	WDT 割込みのマスク
		0: WDT 割込みがブロックされています
		1: WDT 割込みが CPU に転送されます

20.3.2 ウォッチドッグリセット

ウォッチドッグは通常、ファームウェア/システムのクラッシュやフォールトからデバイスを保護するために使用されます。システム クラッシュからの保護に WDT が使用される時、WDT 割込みと直接関係がないコードから WDT 割込みビットをクリアする必要があります。そうしないと、ファームウェアのメイン関数がクラッシュしたり無限ループに陥ったとしても、WDT 割込みベクタは変わらないままで、周期的に WDT に合わせます。デバッグ プローブが接続されている場合、デバイスのリセットはブロックされ、代わりに割込みが発生することに注意してください。

システムクラッシュに対して WDT を使用する最も安全な方法は、以下のとおりです。

- 1 番長いファームウェアの遅延パスの間でも、ファームウェアが周期中に最低 1 回ウォッチドッグをリセットできるようにウォッチドッグ リセット周期を設定します。
- SRSS_INTR レジスタの WDT_MATCH に‘1’を書き込むことにより、ファームウェア コードの本体で定期的に割込みビットをクリアしてウォッチドッグをリセット (フィード) します。これはウォッチドッグ カウンターをリセットしないことに注意してください。次の 2 つの一致イベントでリセットを引き起こさないように、ウォッチドッグのみにフィードします。

WDT をクラッシュからシステムを保護するためのリセット ソースとして使用する場合、WDT 割込みサービス ルーチン (ISR) でウォッチドッグをリセットすることを推奨しません。したがって、WDT リセット機能と ISR を併用しないでください。

WDT をリセット ソースとして使用するための推奨手順は以下のとおりです。

1. WDT_CTL レジスタの WDT_LOCK ビット[31:30]をクリアすることによって WDT 設定がアンロックされていることを確認してください。ビットをクリアするためには、表 129 で説明したように、各書き込みで 1 ビットをクリアするレジスタへの 2 回の書き込みが必要であることに注意してください。
2. WDT_MATCH2 レジスタに希望の IGNORE_BITS_ABOVE を書き込んで、一致に使用されるカウンター レゾリューションを設定します。
3. WDT_MATCH レジスタに任意の一致値を書き込みます。一致イベントではカウンターはリセットされないので、一致値はウォッチドッグ リセットの期間を制御しません。この値により、最初の割込み間隔のみを制御するオプションが提供されます。その後の連続割込みの周期は、IGNORE_BITS_ABOVE によって定義されます。おおよそのウォッチドッグ時間 (秒) は次の式で求められます。

$$\text{Watchdog reset period} = \text{ILO}_{\text{period}} \times (2 \times 2^{(\text{IGNORE_BITS_ABOVE})} + \text{WDT_MATCH}) \quad (5)$$

4. 保留中の WDT 割込みをクリアするために SRSS_INTR レジスタの WDT_MATCH ビットを設定します。
5. CLK_ILO_CONFIG レジスタの ENABLE ビット[31]を設定して ILO を有効にし ます。
6. WDT_CTL レジスタの WDT_EN ビットを設定して WDT を有効にします。
7. WDT_LOCK ビットに'3'を書き込んで WDT および ILO 設定をロックします。これにより CLK_SELECT レジスタの LFCLK_SEL ビットもロックされます。
8. ファームウェアでは、SRSS_INT レジスタの WDT_MATCH ビットに'1'を書き込んで、ウォッチドッグにフィード (割込みをクリア) します。

20.3.3 ウォッチドッグ割込み

デバイス リセットの生成に加えて、WDT は割込みの生成にも使用できます。フリーランニング WDT を使用して割込みサービスとウォッチドッグ リセットを同時には使用できないことに注意してください。

ウォッチドッグ カウンターはアクティブ モードで CPU へ、またはディープスリープモードでウェイクアップ割込みコントローラー (WIC) へ割込み要求を送信できます。また、WDT はハイバネート電力モードからデバイスを起動できます。仕組みは以下のとおりです。

- CPU アクティブモード: アクティブ電力モードでは、WDT は CPU に対して割込みを通知可能です。CPU は割込み要求を承認し、ISR を実行します。ISR で割込みをクリアします
- CPU スリープまたはディープスリープモード: このモードでは、CPU の電源がオフになるため、WDT からの割込み要求は直接 WIC に送信され、CPU は WIC によりウェイクアップされます。CPU は割込み要求を承認し、ISR を実行します。ISR で割込みをクリアします
- ハイバネートモード: このモードでは、いくつかのペリフェラル (WDT や LPCOMP など) を除くデバイス全体がパワーダウンします。このモードでデバイスをウェイクアップするための割込みは、デバイス リセットとなります。したがって、このモードに関連する割込みサービスルーチンやメカニズムはありません。

  デバイスの電力モードの詳細については、デバイスの電力モードセクションを参照してください。

WDT はそのフリーランニングの性質上、周期的な割込み発生に使用してはいけません。代わりにマルチカウンター WDT (MCWDT) を使って定期的な割込みを発生させます。絶対に必要な場合は、以下の手順に従って WDT を周期割込みジェネレーターとして使用してください。

1. ディープスリープまたはハイバネートウェイクアップ、またはデバイスリセット後の WDT レジスタへの最初の更新の場合、WDT をアンロックします。
2. WDT_MATCH レジスタに希望の IGNORE_BITS_ABOVE を書き込んで、一致に使用するカウンター レゾリューションを設定します。
3. WDT_MATCH レジスタに希望する一致値を書き込みます。
4. 保留中の WDT 割込みをクリアするために SRSS_INTR レジスタの WDT_MATCH ビットを設定します。

5. SRSS_INTR_MASK の WDT_MATCH ビットの設定によって、WDT 割込みを有効にします。
6. 適切な ISER レジスタを設定することにより、CPU への SRSS 割込みを有効に設定します (詳細については割込みセクションを参照)。
7. ISR では WDT をアンロックします。WDT 割込みをクリアし、既存の一致値に希望する一致値を追加します。これにより、カウンターは新しい一致値に達した時、他の定期的な割込みを生成します。

20.4 マルチカウンター WDT

20.4.1 概要

PSOC™ Control C3 MCU には MCWDT ブロックが 1 つあります。MCWDT は 2 つの 16 ビット カウンター (MCWDT0_WDT0 と MCWDT0_WDT1) と 1 つの 32 ビット カウンター (MCWDT0_WDT2) があります。これらのカウンターは、独立して、またはカスケード (最大 64 ビット) で動作するように設定できます。16 ビット カウンターは、割込みを発生させたり、デバイスをリセットできます。32 ビット カウンターは割込みを発生させることしかできません。すべてのカウンターは LFCLK でクロックされます。

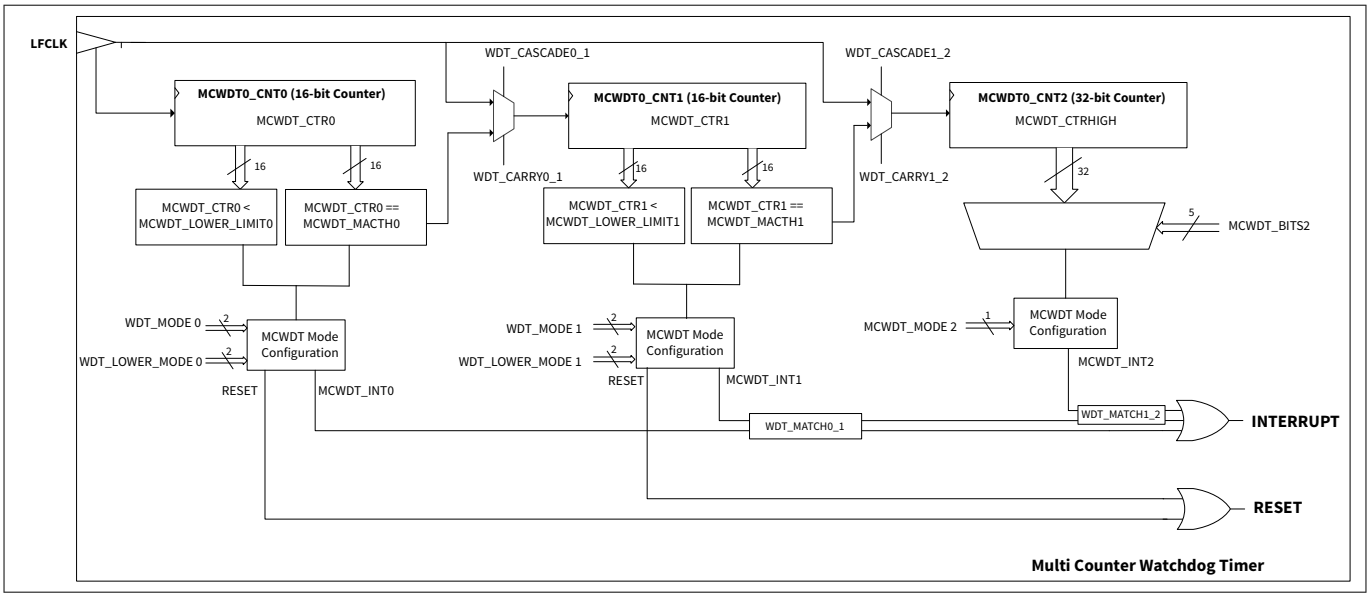


図 74 マルチカウンター WDT 機能図

20.4.1.1 MCWDT0_WDT0 および MCWDT0_WDT1 カウンターの動作

MCWDT0_WDT0 と MCWDT0_WDT1 は 16 ビット アップ カウンターで、16 ビット フリーランニング カウンターまたは任意の 16 ビット周期のカウンターに設定できます。これらのカウンターを使用して、割込みまたはリセットを生成します。

MCWDT0_CNTLOW レジスタの WDT_CTR0 ビット[15:0]と WDT_CTR1 ビット[31:16]は、それぞれ MCWDT0_WDT0 と MCWDT0_WDT1 の現在のカウンター値を保持します。MCWDT0_MATCH レジスタの WDT_MATCH0 ビット[15:0] と WDT_MATCH1 ビット[31:16]はそれぞれ MCWDT0_WDT0 と MCWDT0_WDT1 の一致値を格納します。
MCWDT0_CONFIG レジスタの WDT_MODEx ビットは一致イベント (WDT_MATCHx == WDT_CTRx) に対してウォッチドッグ カウンターが取る動作を設定します。MCWDT0_WDT0/WDT1 カウンターは以下の動作を行います。

- 一致時に割込み (WDT_INTx) をアサートします。
- 一致時にデバイス リセットをアサートします
- 一致時に割込みをアサートし、3 回目の未対応の割込みでデバイスをリセットします。

リセットと割込みを発生させるだけでなく、カウンターをクリアするために一致イベントを設定します。これは、MCWDT0_CONFIG レジスタの WDT_CLEARx ビットを設定することで実現されます。図 75 に MCWDT0_WDT0/WDT1 カウンターの動作を示します。

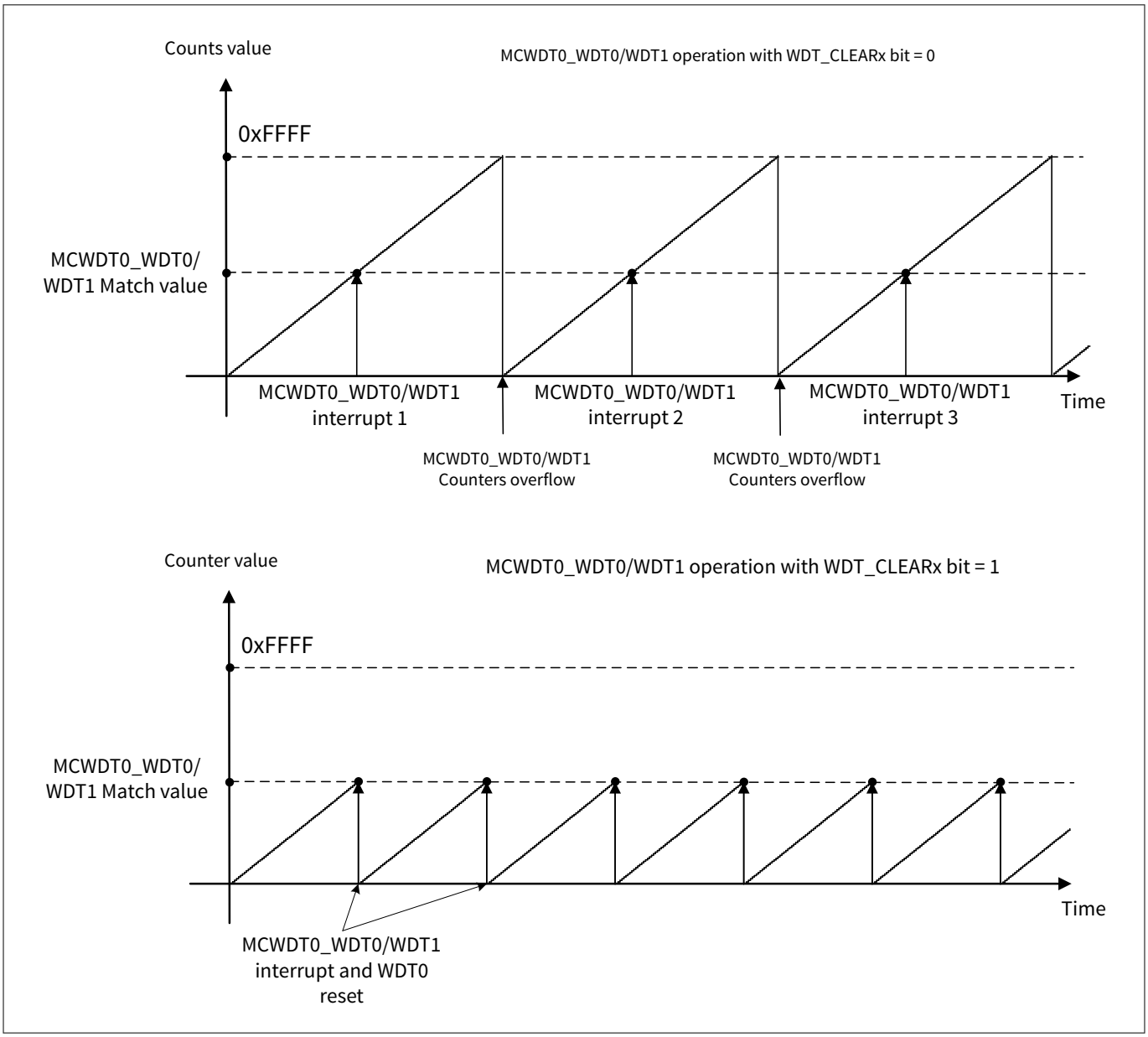


図 75　　MCWDT0_WDT0/WDT1 動作

PSOC™ Control C3 MCU は、図 76 に示すように、MCWDT0_CTR0/1 サブ カウンターのウィンドウ モード動作に対応しています。ウィンドウモードでは、MCWDT0_WDT0 および MCWDT0_WDT1 に対してそれぞれ、MCWDT0_LOWER_LIMIT レジスタの WDT_LOWER_LIMIT0 ビット [15:0] および WDT_LOWER_LIMIT1 ビット [31:16] に初期閾値が格納されます。MCWDT0_CONFIG レジスタの WDT_LOWER_MODE0 ビット[5:4]および WDT_LOWER_MODE1 ビット[13:12]が'2'に設定されている場合に使用でき、カウンターのサービスが早すぎた場合にリセットを生成します。

ウィンドウ機能は、カウンターがオーバーフローして折り返さない場合にのみ機能します。実際問題として、WDT_MODE0/1== RESET のみがウィンドウ動作に有用であることを意味します。その他の WDT_MODE0/1 設定は、ウィンドウ操作ではサポートされません。

ウィンドウ機能は、サブカウンター 0 と 1 で機能します。カスケード接続もサポートされており、この場合、下限レジスタは統合された限度値として扱われます。ウィンドウ機能はサブカウンター 2 ではサポートされていません。

サービス後すぐに、下限値と上限値を動的に変更できる必要があります。これは、元のサービスが許可されたウィンドウに違反しない限り、いかなるアクションも引き起こしてはいけません。

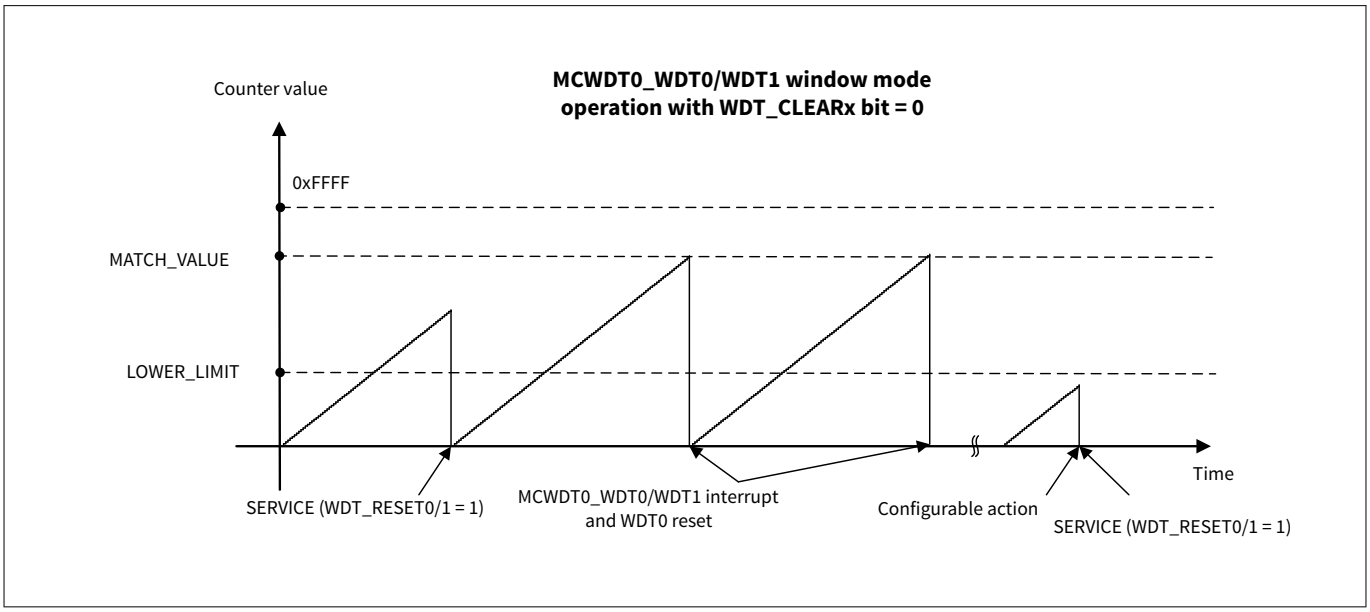


図 76　　MCWDT0_WDT0/WDT1 ウィンドウモード動作

表 130　　MCWDT0_WDT0 および MCWDT0_WDT1 の設定オプション

レジスタ [Bit_Pos]	Bit_Name	説明
MCWDT0_CONFIG[1:0] MCWDT0_CONFIG[9:8]	WDT_MODE0 WDT_MODE1	ウォッチドッグのカウンターアクション値が一致。値が一致 (WDT_CTR0=WDT_MATCH0) した後、次のインクリメントでアクションが実行されます。
		0: 何もしない
		1: 割込みをアサート (WDT_INTx)
		2: デバイスのリセットをアサート
		3: 一致時に割込みをアサート、3 回目の未対応割込み後にデバイスをリセット
MCWDT0_CONFIG[5:4] MCWDT0_CONFIG[13:12]	WDT_LOWER_MODE0 WDT_LOWER_MODE1	ウォッチドッグ カウンター下限値前のサービスに対する動作。 0: 何もしない 1: 割込みをアサート (MCWDT_INTx) 2: WDT リセットをアサート

(続く)

表 130 (続き) MCWDT0_WDT0 および MCWDT0_WDT1 の設定オプション

レジスタ [Bit_Pos]	Bit_Name	説明
MCWDT0_CONFIG[2] MCWDT0_CONFIG[10]	WDT_CLEAR0 WDT_CLEAR1	一致時に MCWDT0_WDT0/WDT1 カウンターをクリアします。言い換えれば、(WDT_MATCHx + 1) は MCWDT0_WDT0/WDT1 カウンターの周期と同様の働きをします。 0: フリーランニング カウンター 1: 一致時に WDT_CTRx ビットをクリアします。
MCWDT0_CNTLOW[15:0] MCWDT0_CNTLOW[31:16]	WDT_CTR0 WDT_CTR1	現在のカウント。ビット[15:0]には、MCWDT0_WDT0 カウンターの現在値が格納されます。ビット[31:16]には、MCWDT0_WDT1 カウンターの現在値が格納されます。
MCWDT0_MATCH[15:0] MCWDT0_MATCH[31:16]	WDT_MATCH0 WDT_MATCH1	一致値 WDT_MATCHx を変更すると有効になるまでに 1.5 CLK_LF サイクルを要します。WDT_MATCHx を変更後、WDT が確実に新しい設定に更新されるために、少なくとも 1 CLK_LF サイクルはディープスリープモードに移行しないでください。
MCWDT0_LOWER_LIMIT[15:0] MCWDT0_LOWER_LIMIT[31:16]	WDT_LOWER_LIMIT0 WDT_LOWER_LIMIT1	カウンター 0 と 1 の下限値を提供します。

20.4.1.2 MCWDT0_WDT2 カウンター動作

MCWDT0_WDT2 は 32 ビットのフリーランニング カウンターで、割込みを発生させるように設定できます。MCWDT0_CNTHIGH レジスタは WDT2 カウンターの現在値を保持します。WDT2 は一致機能に対応していません。ただし、カウンタービットのいずれかがトグルした際に割込みを生成するように設定できます。MCWDT0_CONFIG レジスタの WDT_BITS2 ビット[28:24]は、MCWDT0_WDT2 割込みがアサートされるビットを選択します。MCWDT0_CONFIG レジスタの WDT_MODE2 ビット[16]は、ビットトグル時に割込みをアサートするかどうかを決定します。図 77 に MCWDT0_WDT2 カウンターの動作を示します。

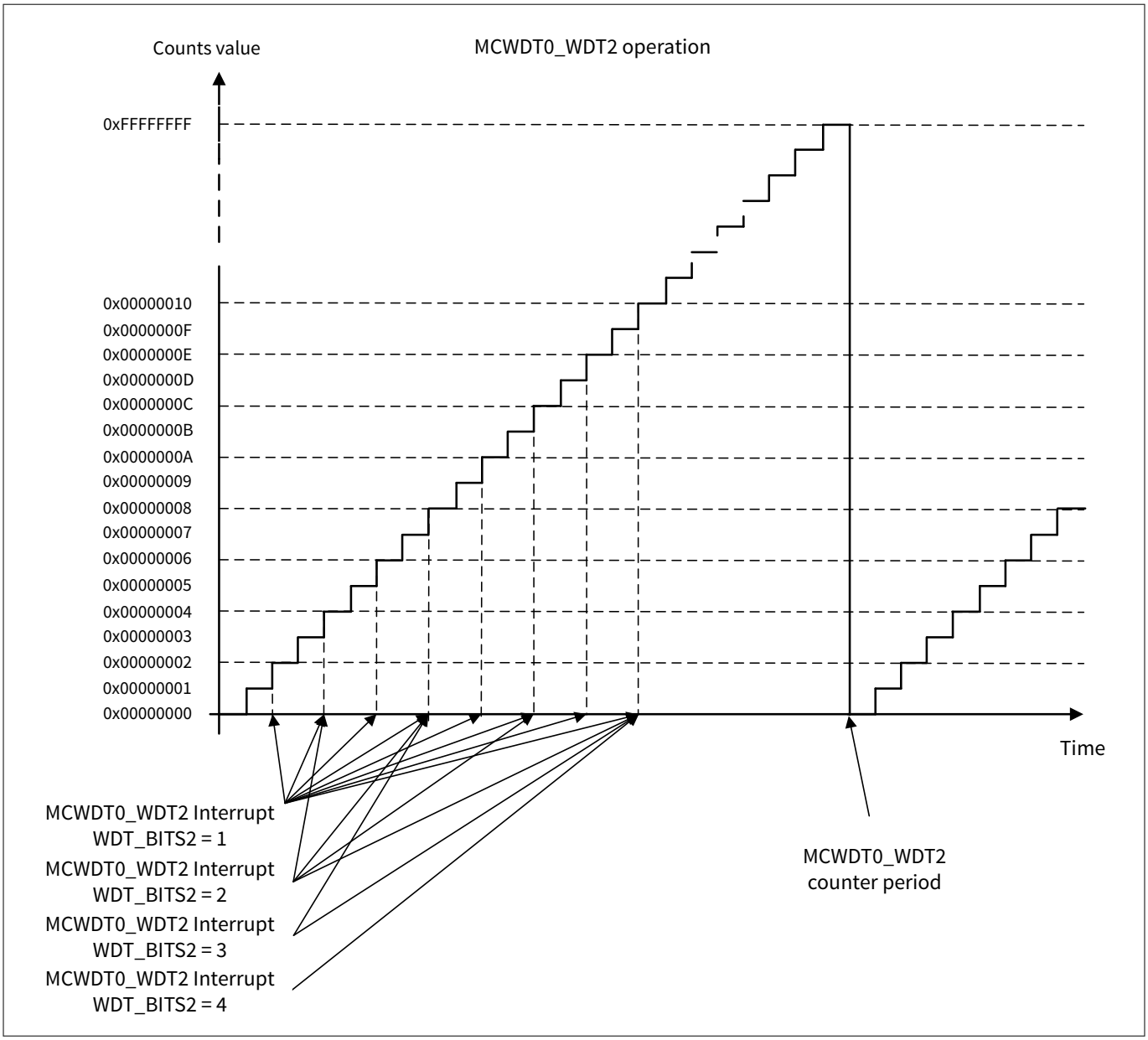


図 77　　MCWDT0_WDT2 動作

表 131　　MCWDT0_WDT2 設定オプション

レジスタ [Bit_Pos]	Bit_Name	説明
MCWDT0_CONFIG[16]	WDT_MODE2	MCWDT0_WDT2 モード 0: フリーランニング カウンター 1: 割込み (WDT_INTx) 要求生成付きフリーランニング カウンター。割込みは MCWDT0_CNTHIGH レジスタが保持するカウンター値の WDT_BITS2 で指定されたビット位置がトグルする時に発生します。

(続く)

表 131 (続き) MCWDT0_WDT2 設定オプション

レジスタ [Bit_Pos]	Bit_Name	説明
MCWDT0_CONFIG[28:24]	WDT_BITS2	MCWDT0_WDT2 割込みアサート監視用ビット 0: MCWDT0_CNTHIGH レジスタのビット[0]がトグルするとアサート (Tick ごとに割込み) …… 31: MCWDT0_CNTHIGH レジスタのビット[31]がトグルしたときにアサート (2^ (31) Tick ごとに割込み)
MCWDT0_CNTHIGH[31:0]	WDT_CTR2	MCWDT0_WDT2 の現在のカウンター値

20.4.2 MCWDT の有効化と無効化

MCWDT カウンターは、MCWDT0_CTL レジスタ内の WDT_ENABLEx ビットを設定することで有効化され、クリアすることで無効になります。MCWDT を有効または無効に設定するためには、最大 2LFCLK サイクルが必要です。したがって WDT_ENABLEx ビットの値はその期間に 1 回より多く変更されてはならず、MCWDT0_CTL レジスタの WDT_ENABLEDx ビットを使用してカウンターの有効/無効状態を監視できます。

MCWDT0_CTL レジスタの WDT_RESETx ビットは、ファームウェアで設定されると、対応する MCWDT0 カウンターをクリアします。ハードウェアは、カウンターのリセット後にビットをクリアします。このオプションは、MCWDT0_WDT0 または MCWDT0_WDT1 が一致イベントでデバイス リセットを生成するように設定されている場合に便利です。このような場合、カウンターが一致値に達すると、デバイスはリセットされます。したがって、WDT_RESET0 または WDT_RESET1 ビットを設定すると、それぞれ MCWDT0_WDT0 または MCWDT0_WDT1 がリセットされ、デバイスのリセットが防止されます。

MCWDT を有効にした後は、MCWDT 設定レジスタ (MCWDT0_CONFIG) および制御レジスタ (MCWDT0_CTL) に書き込みをしないでください。これらのレジスタが誤って書き換えられることを防ぐには、MCWDT0_LOCK レジスタの MCWDT_LOCK ビット[31:30]を設定します。MCWDT の実行中にアプリケーションが WDT_MATCH を更新しなければいけない場合は、MCWDT_LOCK ビットをクリアする必要があります。両方の MCWDT_LOCK ビットをクリアするために 2 つの異なる書き込みが必要です。‘1’を書き込むとビット 0 がクリアされます。‘2’を書き込むとビット 1 がクリアされます。‘3’を書き込むと両方のビットが設定され、‘0’を書き込んでも効果はありません。MCWDT_LOCK ビットは、MCWDT0_CTL、MCWDT0_CONFIG、MCWDT0_MATCH レジスタのみを保護することに注意してください。LFCLK セレクト レジスタはフリーランニング WDT ロック ビットによって保護されます。

表 132 ウォッチドッグ設定オプション

レジスタ [Bit_pos]	Bit_Name	説明
MCWDT0_CTL[0] MCWDT0_CTL[8] MCWDT0_CTL[16]	WDT_ENABLE0 WDT_ENABLE1 WDT_ENABLE2	MCWDT カウンター有効。最大 2× LFCLK サイクルの時間がかかります。 0: カウンター無効 (クロックされません) 1: カウンター有効 (カウントアップ)

(続く)

表 132 (続き) ウォッチドッグ設定オプション

レジスタ [Bit_pos]	Bit_Name	説明
MCWDT0_CTL[1] MCWDT0_CTL[9] MCWDT0_CTL[17]	WDT_ENABLED0 WDT_ENABLED1 WDT_ENABLED2	カウンター x の実際の状態を示します。WDT_ENABLEx ビットを変更した後にこのビットをモニターして、変更の確認を受けます。
MCWDT0_CTL[3] MCWDT0_CTL[11] MCWDT0_CTL[19]	WDT_RESET0 WDT_RESET1 WDT_RESET2	MCWDT カウンター x を 0 にリセットします。リセットが完了すると、ハードウェアはビットをクリアします。 0: ソフトウェア - 対応なし 1: ソフトウェア - カウンターをリセットします
MCWDT0_LOCK[31:30]	MCWDT_LOCK	MCWDT0_CTL、MCWDT0_CONFIG、および MCWDT0_MATCH レジスタへの書き込みアクセスをロックまたはアンロックします。ビットが設定されると、ロックが有効になります。 0: 無効 1: ビット 0 をクリア 2: ビット 1 をクリア 3: ビット 0 と 1 の両方をセット (ロック有効)

20.4.3 ウォッチドッグ カスケード オプション

図 74 に示すカスケード設定は、MCWDT カウンターの範囲、または最大タイムアウト時間を増やす選択肢を提供します。MCWDT0_CONFIG レジスタの WDT_CASCADE0_1 ビット[3]は MCWDT0_WDT0 と MCWDT0_WDT1 をカスケードし、MCWDT0_CONFIG レジスタの WDT_CASCADE1_2 ビット[11]は MCWDT0_WDT1 と MCWDT0_WDT2 をカスケードします。2 つの 16 ビット カウンターをカスケード接続しても 32 ビット カウンターが形成されないことに注意してください。その代わりに 16 ビット分周器出力に対して 16 ビット周期カウンターを取れます。例えば、MCWDT0_WDT0 と MCWDT0_WDT1 をカスケード接続する時、MCWDT0_WDT0 は MCWDT0_WDT1 の分周器として動作して、分周値は MCWDT0_MATCH レジスタの WDT_MATCH0 ビット[15:0]に定義されます。MCWDT0_WDT1 の周期は WDT_MATCH レジスタ内の WDT_MATCH1[31:16]ビットに定義されます。MCWDT0_WDT1 と MCWDT0_WDT2 のカスケード接続でも同じロジックを使用します。

さらに、カスケード接続されたカウンターのキャリー動作は、MCWDT0_CONFIG レジスタの WDT_CARRY0_1 および WDT_CARRY1_2 ビットを使用することで、一致またはロールオーバー時に設定できます。WDT_MATCH0_1 ビットと WDT_MATCH1_2 ビットはカスケード接続されたカウンターの一致動作を定義するために設定できます。一致時割込みは、最初のカウンターのみが一致した場合、またはカスケード接続されたカウンターが同時に一致した場合に発生します。

表 133 にカスケード コンフィギュレーション ビットの概要を示します。

表 133 ウォッチドッグ カスケード オプション

レジスタ [Bit_Pos]	Bit_name	説明
MCWDT0_CONFIG[3]	WDT_CASCADE0_1	カスケード ウォッチドッグ カウンター 0, 1。カウンター 1 は WDT_CTR0=WDT_MATCH0 の後のサイクルでインクリメントします。 0: 独立カウンター 1: カスケード カウンター
MCWDT0_CONFIG[11]	WDT_CASCADE1_2	カスケード ウォッチドッグ カウンター 1, 2。カウンター 2 は WDT_CTR1=WDT_MATCH1 の後のサイクルでインクリメントします。3 つの WDT カウンターすべてをカスケード接続できます。 0: 独立カウンター 1: カスケードカウンター。3 つのカウンターすべてをカスケード接続する場合、WDT_CLEAR1 は 1 でなければいけません。
MCWDT0_CONFIG[6]	WDT_CARRY0_1	WDT_CASCADE0_1 == 1 の場合に適用される動作を実行します。WDT_CASCADE0_1 == 0 の時、このビットは使用されません。 0: カウンター 0 の一致で実行。 1: カウンター 0 のロールオーバーで実行。
MCWDT0_CONFIG[14]	WDT_CARRY1_2	WDT_CASCADE1_2 == 1 の場合に適用される動作を実行します。WDT_CASCADE1_2 == 0 の時、このビットは使用されません。 0: カウンター 1 の一致で実行。 1: カウンター 1 のロールオーバーで実行。
MCWDT0_CONFIG[7]	WDT_MATCH1_0	WDT_CASCADE0_1 == 1 の場合のマッチング動作を指定します。WDT_CASCADE0_1 == 0 のとき、このビットは使用されず、カウンター 1 のみで一致します。 0: カウンター 1 のみ一致 1: カウンター 1 とカウンター 0 の一致時

(続く)

表 133 (続き) ウォッチドッグ カスケード オプション

レジスタ [Bit_Pos]	Bit_name	説明
MCWDT0_CONFIG[15]	WDT_MATCH1_2	WDT_CASCADE1_2 == 1 の場合のマッチング動作を指定します。 WDT_CASCADE1_2 == 0 のとき、このビットは使用されず、カウンター 2 のみで一致します。 0: カウンター 2 のみ一致 1: カウンター 2 とカウンター 1 の一致時

カスケードを使用する場合 (WDT_CASCADE0_1 または WDT_CASCADE1_2 が設定)、カウンターが一致時にクリアされるように設定されている状態でプリスケーラまたは下位カウンターが一致値にある時にカウンターをリセットすると、上位カウンターは 0 のままではなく 1 にインクリメントされます。最初のリセットから約 100 µs 後に上位カウンターに 2 回目のリセットを発行することで、この動作を修正します。2 回目のリセットが必要なのは、プリスケーラ カウンター値がその一致値にある間に 1 回目のリセットが発行された場合のみであることに注意してください。図 78 に、MCWDT0_WDT0 と MCWDT0_WDT1 が第 2 のリセット タイミングとともにカスケード接続された場合の動作を示します。

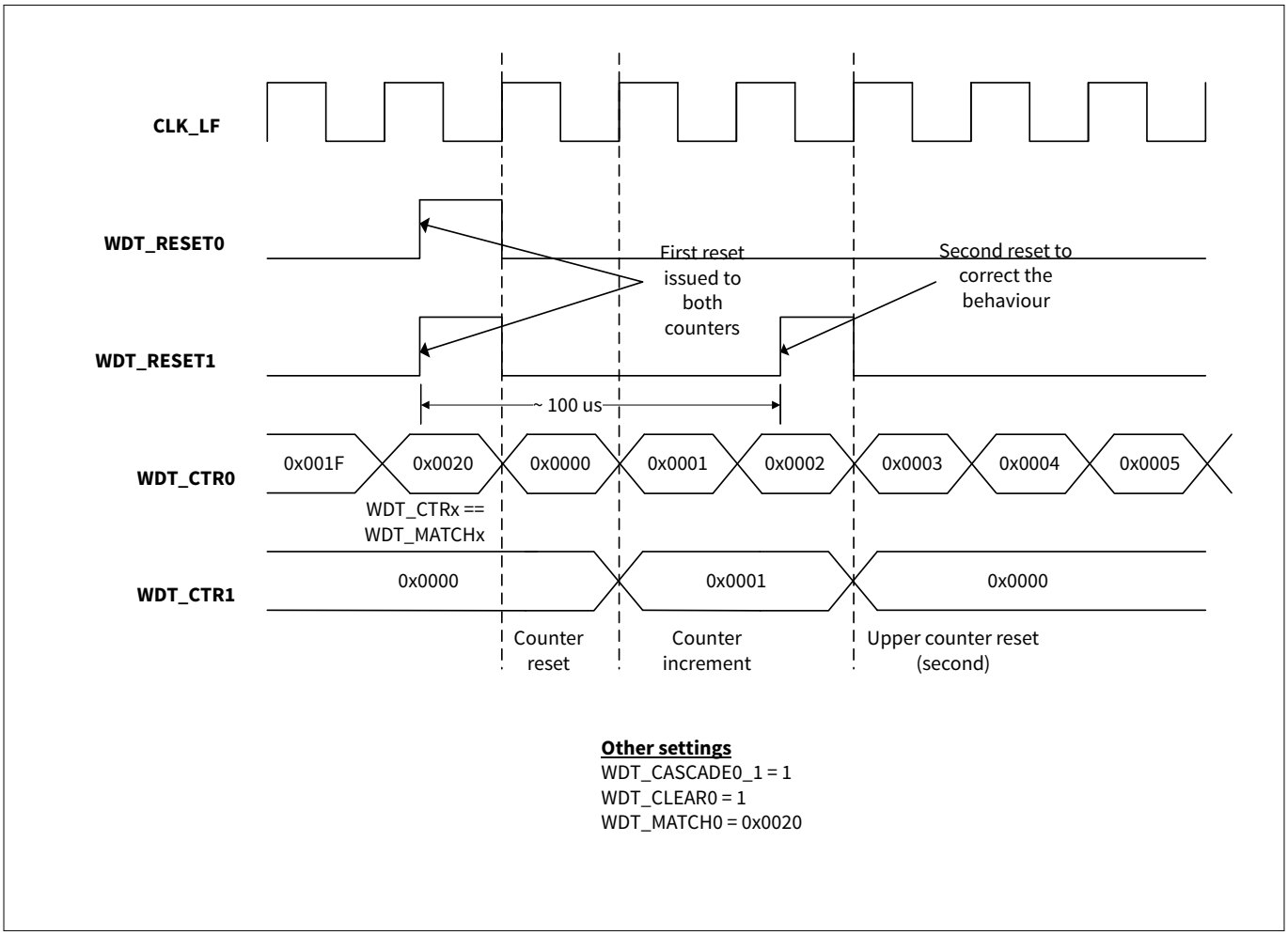


図 78 カスケード モードでの MCWDT リセット動作

さらに、カウンターは以下のカスケード条件において非単調性を示します。

- WDT_CASCADE0_1 が設定されている場合、WDT_CTR1 は WDT_CTR0 = WDT_MATCH0 の後のサイクルをインクリメントしません。
- WDT_CASCADE1_2 が設定されている場合、WDT_CTR2 は WDT_CTR1 = WDT_MATCH1 の後のサイクルをインクリメントしません。
- WDT_CASCADE0_1 と WDT_CASCADE1_2 の両方が設定されている場合、WDT_CTR2 は WDT_CTR1 = WDT_MATCH1 の後のサイクルをインクリメントせず、WDT_CTR1 は WDT_CTR0 = WDT_MATCH0 の後のサイクルをインクリメントしません。

カスケードが有効な時、プリスケーラカウンター (WDT_CTR0 または WDT_CTR1) 値が 0 でない時のみ常に WDT_CTR1 または WDT_CTR2 カウンター値を読んでください。これにより、周期カウンターの一致イベント後に、上位カウンターがインクリメントされます。

20.4.4 MCWDT リセット

MCWDTx_WDT0 と MCWDTx_WDT1 は、フリーランニング WDT リセットと同様のデバイス リセットを生成するように設定できます。デバッグ プローブが接続されている場合、デバイスのリセットはブロックされますが、設定されていれば割込みが発生することに注意してください。MCWDTx ブロックの MCWDTx_WDT0 または MCWDTx_WDT1 カウンターを使用してシステム リセットを生成するためには、以下の手順に従ってください。

1. MCWDTx_CONFIG の WDT_MODEx ビットを使用して、リセットを生成するように MCWDT を設定します。MCWDTx_CONFIG 内の WDT_MODE0 または WDT_MODE1 ビットを‘2’ (一致時リセット) または‘3’ (一致時割込み、3 番目の未処理の割込み時リセット) に設定します。
2. オプションで、MCWDTx_CONFIG レジスタの WDT_CLEAR0 ビットまたは WDT_CLEAR1 ビットを設定し、MCWDTx_WDT0 または MCWDTx_WDT1 の一致イベント発生時に、対応するウォッチドッグカウンターを‘0’にリセットします。そうでなければ、カウンターはフリーランニングとなります。詳細については MCWDT0_WDT0 および MCWDT0_WDT1 カウンターの動作を参照してください。
3. 1 番長いファームウェアの遅延パスの間でも、ファームウェアが周期中に最低 1 回ウォッチドッグをリセットできるようにウォッチドッグ リセット周期を設定します。WDT_MODEx == 2 の場合、一致値はウォッチドッグ周期と同じです。WDT_MODEx == 3 の場合、一致値はウォッチドッグ周期の 3 分の 1 となります。計算された一致値を、MCWDTx_WDT0 または MCWDTx_WDT1 の WDT_MATCH レジスタに書き込みます。オプションで、カスケードを有効にして間隔を広げてください。

   ***注**: WDT_MATCH フィールドの有効値は 1 から 65535 です。*

4. WDT_MODEx == 2 ならば、WDTx カウンターを 0 にリセットするために MCWDTx_CONFIG レジスタの WDT_RESETx ビットを設定します。WDT_MODEx == 3 の場合、MCWDTx_INTR の WDT_INTx ビットを設定し、保留中の割込みをクリアします。
5. MCWDTx_CTL レジスタの WDT_ENABLEx ビットを設定し、WDTx を有効にします。WDT_ENABLEDx ビットが設定されるまで待ちます。
6. MCWDTx_CTL レジスタの MCWDT_LOCK ビットを設定して MCWDTx コンフィギュレーションをロックします。
7. ファームウェアで、ステップ 4 で説明したようにウォッチドッグをフィード (リセット) します。

WDT ISR でウォッチドッグをリセットしないでください。また、システム リセットと割込みを生成するために同じウォッチドッグ カウンターを使用することも推奨しません。例えば、クラッシュからシステムを保護することを目的としてシステム リセットを生成するために MCWDTx_WDT0 を使用する場合、MCWDTx_WDT1 と MCWDTx_WDT2 は定期的に割込みを生成する割込みタイマーとして使用される必要があります。

20.4.5 MCWDT 割込み

割込みを生成するように設定されている場合、MCWDTx_INTR レジスタの WDT_INTx ビットは保留中のウォッチドッグ割込みのステータスを提供します。ファームウェアは WDT_INTx を設定することによって割込みをクリアする必要があります。MCWDTx_INTR_MASK レジスタの WDT_INTx ビットは MCWDTx ブロックの対応する WDTx 割込みを CPU に対してマスクします。

以下の手順に従って WDT を周期割込みジェネレーターとして使用してください。

1. MCWDTx_WDT0/WDT1 の場合は MCWDT_MATCH レジスタに比較値を書き込み、MCWDTx_WDT2 の場合は MCWDTx_CONFIG レジスタに WDT_BITS2 値を書き込みます。

   注: WDT_MATCH フィールドの有効値は 1 から 65535 です。

2. MCWDTx_CONFIG の WDT_MODEx ビットを使用して割込みを生成するように WDTx を設定してください。MCWDTx_WDT0 または MCWDTx_WDT1 の場合、MCWDTx_CONFIG レジスタ内の WDT_MODE0 または WDT_MODE1 ビットを‘1’ (一致時割込み) または‘3’ (一致時割込み、3 番目の未処理の一致時リセット) に設定します。MCWDTx_WDT2 については、MCWDTx_CONFIG レジスタ内の WDT_MODE2 ビットを設定します。
3. MCWDTx_INTR 内の WDT_INT ビットを設定し、保留中の割込みをクリアします。
4. MCWDTx_CONFIG レジスタの WDT_CLEAR0 ビットまたは WDT_CLEAR1 ビットを設定し、MCWDTx_WDT0 または MCWDTx_WDT1 の一致イベント発生時に、対応するウォッチドッグカウンターを‘0’にリセットします。
5. MCWDTx_INTR_MASK レジスタの WDT_INTx ビットを設定することにより、CPU への WDTx 割込みをマスクします。
6. MCWDTx_CTL レジスタの WDT_ENABLEx ビットを設定し、WDTx を有効にします。WDT_ENABLEDx ビットが設定されるまで待ちます。
7. 適切な ISER レジスタを設定することにより、CPU に対する MCWDTx 割込みを有効にします。割込み セクションを参照してください。
8. ISR で、MCWDTx_INTR レジスタの WDT_INTx ビットを設定することによって WDTx 割込みをクリアします。

MCWDT ブロックの 3 つの WDTx カウンターすべてからの割込みは、CPU への 1 つの割込みとしてマッピングされることに注意してください。割込みサービスルーチンでは、割込み元を識別するために MCWDTx_INTR レジスタの WDT_INTx ビットを読み出せます。しかし、MCWDT ブロックには、それぞれ CPU への割込みがあります。割込みの詳細については、割込みセクションを参照してください。MCWDT ブロックは、アクティブ電力モードでは CPU に、スリープおよびディープスリープ電力モードでは WIC に割込み要求を送信できます。フリーランニング WDT に似た働きをします。

20.5 リセット原因検出

RES_CAUSE レジスタの RESET_WDT ビット [0] はフリーランニング WDT が生成するリセットを示します。RES_CAUSE レジスタ内の RESET_MCWDT0 ビットは、MCWDT0 ブロックによって生成されたリセットを示します。このビットはクリアされるか、パワーオン リセット (POR)、電圧低下リセット (BOD) もしくは外部リセット (XRES) が発生するまでセットされたままになります。他のリセットでは、このビットは変更されません。詳細については、TRM のリセット制御の章を参照してください。

20.6 レジスタの概要

表 134 WDT レジスタ

名称	説明
WDT_CTL	ウォッチドッグカウンター制御レジスタ
WDT_CNT	ウォッチドッグカウンター カウント レジスタ
WDT_MATCH	ウォッチドッグカウンター マッチ レジスタ
WDT_MATCH2	ウォッチドッグ カウンター マッチ レジスタ 2
MCWDT0_MCWDT_CNTLOW	マルチカウンター WDT サブカウンター 0/1
MCWDT0_MCWDT_CNTHIGH	マルチカウンター WDT サブカウンター 2

(続く)

表 134 (続き) WDT レジスタ

名称	説明
MCWDT0_MCWDT_MATCH	マルチカウンター WDT カウンター マッチ レジスタ、カウンター 0 および 1 用
MCWDT0_MCWDT_CONFIG	マルチカウンター WDT カウンター設定 (カウンター 2 のビットトグル割込み生成を含みます)
MCWDT0_MCWDT_CTL	マルチカウンター WDT カウンター制御
MCWDT0_MCWDT_LOCK	マルチカウンター WDT カウンター ロック レジスタ
MCWDT0_MCWDT_LOWER_LIMIT	マルチカウンター WDT 下限レジスタ
MCWDT0_MCWDT_INTR	マルチカウンター WDT カウンター割込みレジスタ
MCWDT0_MCWDT_INTR_SET	マルチカウンター WDT カウンター割込み設定レジスタ
MCWDT0_MCWDT_INTR_MASK	マルチカウンター WDT カウンター割込みマスク レジスタ
MCWDT0_MCWDT_INTR_MASKED	マルチカウンター WDT カウンター割込みマスクドレジスタ
CLK_SELECT	クロック選択レジスタ
CLK_ILO_CONFIG	ILO の設定
SRSS_INTR	SRSS 割込みレジスタ
SRSS_INTR_SET	SRSS 割込みセットレジスタ
SRSS_INTR_MASK	SRSS 割込みマスクレジスタ
SRSS_INTR_MASKED	SRSS 割込みマスクドレジスタ
RES_CAUSE	リセット要因監視レジスタ

21 バックアップ

バックアップドメインは、バッテリーやスーパーキャパシターのようなバックアップ電源 (VBACKUP) から供給される独立した電源ドメインを使用して、PSOC™ Control C3 MCU に「常時オン」機能を追加します。32768 Hz の時計用水晶発振器 (WCO) がサポートするアラーム機能付きリアルタイム クロック (RTC) を内蔵しています。

バックアップは電力モードではなく、独自の電源供給を持つ電源ドメインです。デバイスがいずれの電力モード中にあってもアクティブ状態を維持できます。

21.1 機能

- フル機能 RTC
  - 年/月/日, 曜日, 時: 分: 秒に対応したフィールド
  - RTC は二進化十進数 (BCD、Binary-coded decimal) の代わりに 2 進数を使用
  - 12 時制と 24 時制の両方に対応
  - うるう年の自動修正
- 設定可能なアラーム機能
  - 月/日, 曜日, 時: 分: 秒に対応したフィールド
  - 2 つの独立したアラーム
- 32768 Hz WCO、キャリブレーション付き
- バックアップ電源の自動切り替え
- 内蔵スーパーキャパシター充電器
- 32 バックアップレジスタ

21.2 アーキテクチャ

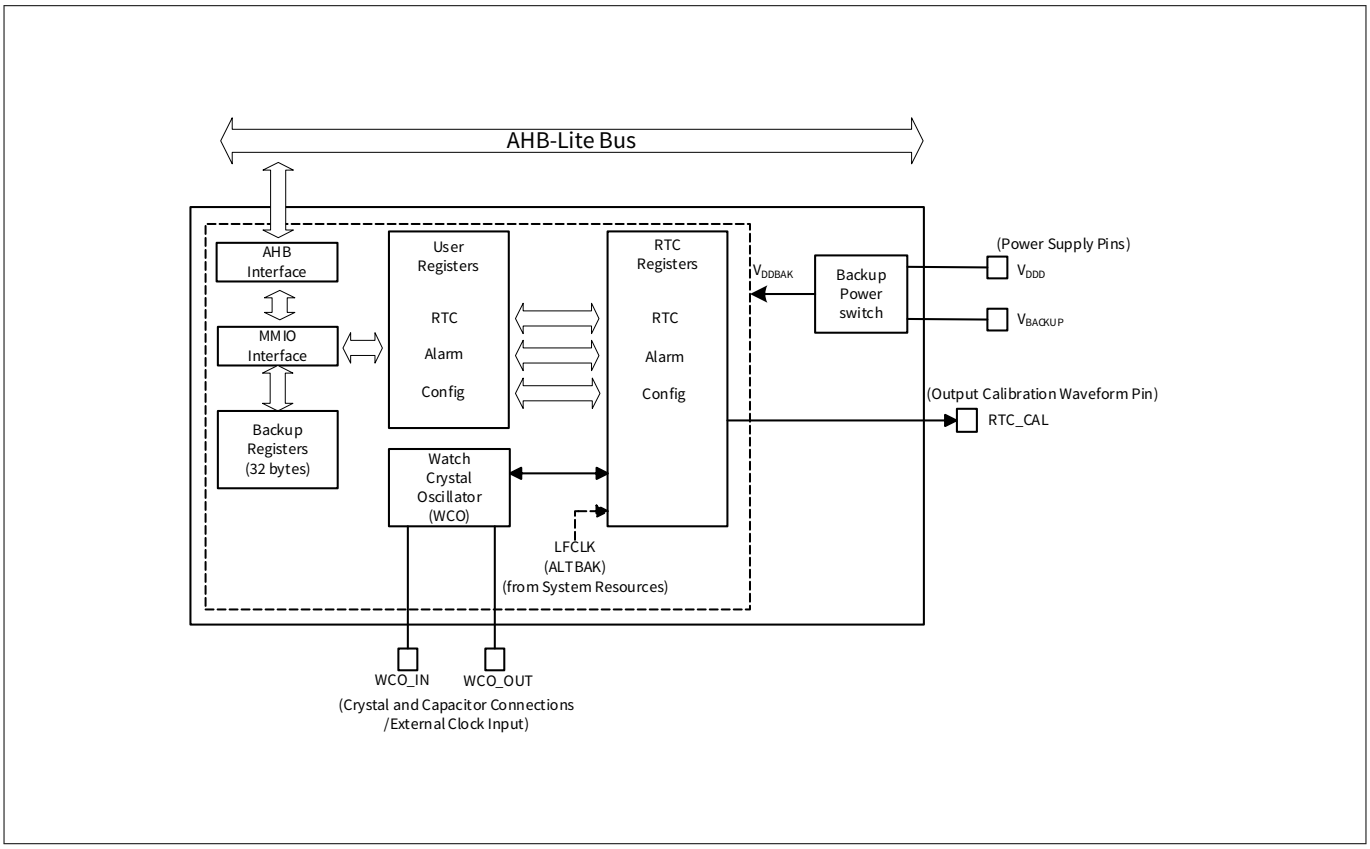


図 79 ブロック図

バックアップ システムには、外部水晶振動子または外部クロック入力の助けを借りて RTC に必要なクロックを生成できる正確な WCO が含まれています。RTC はプログラマブルなアラーム機能を持っており、CPU に割込みを発生させられます。AHB-Lite インターフェースは、Backup ドメインの MMIO レジスタへのファームウェア アクセスを提供します。

自動バックアップ電源スイッチは、バックアップ ドメインのブロックを実行するために必要な VDDBAK 電源として、VDDD (メイン電源) または VBACKUP (バックアップ バッテリ/スーパーキャパシター電源) のいずれかを選択します。

ドメインはまた、バックアップ電源 (VBACKUP) が存在する場合、メイン電源 (VDDD) がオフであってもその内容を保持する 32 バイトのデータを格納するバックアップレジスタを有します。

21.3 電源

バックアップシステム (VDDBAK) への電源は、VDDD (メイン電源) と VBACKUP (バックアップドメイン電源) の間で自動的に切り替わります。VBACKUP は通常、バッテリーまたはスーパーキャパシターから供給される独立電源に接続されます。

電源セレクタースイッチには、VBACKUP 対 VDDD のシーケンス制限はありません。通常動作中に VBACKUP または VDDD のいずれかを取り外しても、バックアップ システムは電力を供給されたままです。

BACKUP_CTL レジスタの VDDBAK_CTL ビット フィールドは電源選択スイッチの動作を制御します。このレジスタの詳細については、レジスタ リファレンス マニュアルを参照してください。別のモード オプションは以下のとおりです。

- VDDBAK_CTL = 0 (デフォルト モード): システム リソース内のブラウンアウト検出が有効で、ブラウンアウト状況が検出されない場合に VDDD を選択します (詳細は、電源と電圧監視の章を参照)。そうでなければ、VDDD と VBACKUP の中で最も高い電源を選択します。
- VDDBAK_CTL = 1、2、または 3: デバッグ用に常に VBACKUP を選択します。

スーパーキャパシターが VBACKUP に接続されている場合、PSOC™ Control C3 MCU は VDDD が利用可能な間にスーパーキャパシターを充電できます。BACKUP_CTL レジスタの EN_CHARGE_KEY ビット フィールドに"3C"を書き込むことにより、スーパーキャパシターの充電を有効に設定できます。この機能はスーパーキャパシターのみを充電するもので、バッテリーを安全に充電できないことに注意してください。VBACKUP がバッテリーに接続されているときは、このキーを書き込まないでください。バッテリーの充電は、外部回路を使って基板レベルで処理されます。

***注**: VDDD と VBACKUP が PCB 上で接続されている場合、Backup ドメインは BACKUP_RESET レジスタの RESET ビット フィールドを使用してファームウェアによってトリガーされる明示的なリセットを必要とすることがあります。このファームウェア リセットは、VBACKUP 電源が以前の電源のランプアップまたはブラウンアウトの際に無効であった場合に必要となります。RES_CAUSE レジスタがリセットの原因として非電力関連のリセットを示す場合、または PSOC™ Control C3 MCU がハイバネート モードからウェイクアップしたばかりで、電源がずっと有効と仮定される場合、バックアップ ドメインをリセットする必要はありません。*

21.4 クロック

RTC は主に 32768 Hz のクロックを 1 秒刻みにスケールダウンして動作します。このクロック信号は、これらの内部ソースのいずれかから供給されます。

- 時計用水晶発振器 (WCO)。これは RTC アプリケーションに適した高精度クロック ジェネレーターで、アプリケーション ボード上に 32768 Hz の外部水晶振動子を実装する必要があります。WCO は、外部クロック/正弦波入力を使用して、水晶振動子なしで動作することもできます。これらの追加の動作モードについては、このセクションの後半で説明します。WCO はバックアップ ドメインから供給されるため、VDDD がなくても動作します。
- 代替バックアップクロック (ALTBAK): このオプションにより、システム リソース サブシステム (SRSS) が生成する LFCLK をバックアップ ドメイン クロックとして使用できます。LFCLK は、すべてのデバイスの電力モードまたは VDDD が取り外されている場合には利用できないことに注意してください。

SRSS によって LFCLK が有効または無効にされると、クロック グリッチがバックアップ システムに伝搬することがあります。さらに、LFCLK の実際のソースによっては、LFCLK は WCO ほど正確ではないかもしれません。このため、LFCLK は RTC アプリケーションには推奨されません。また、WCO をクロック ソースとする場合は、LFCLK を経由せず、直接選択してください。

クロックとキャリブレーションの詳細については、リファレンス マニュアルのクロック システムを参照してください。

RTC クロックソースは、BACKUP_CTL レジスタの CLK_SEL ビットフィールドで選択できます。BACKUP_CTL レジスタの WCO_EN ビットは、WCO を有効または無効に設定するために使用されます。WCO が外部水晶発振子で動作する場合、WCO を有効に設定する前に BACKUP_CTL レジスタの WCO_BYPASS ビットがクリアされていることを確認してください。さらに、BACKUP_CTL の PRESCALER ビットフィールドは 32768 のプリスケーラ値に設定されています。

***注**: WCO_IN 端子と WCO_OUT 端子には、適切な値の外付け水晶振動子と負荷コンデンサを接続する必要があります。コンポーネントの値と電気的接続の詳細については、デバイスのデータシートを参照してください。さらに、GPIO を WCO_OUT と WCO_IN 信号用に設定する必要があります。GPIO の設定については、リファレンス マニュアルの I/O サブシステムの章を参照してください。*

21.4.1 外部クロック/正弦波入力による WCO

WCO は外部クロック/正弦波入力からも動作可能です。これらのモードでは、WCO を有効に設定する前に、BACKUP_CTL レジスタの WCO_BYPASS ビットを設定し、WCO をバイパスする必要があります。また、GPIO を WCO_OUT と WCO_IN 信号用に設定する必要があります (アナログモードの場合)。外部クロック/正弦波入力モード、プリスケーラ設定、電気的接続は以下のとおりです。

- 32768 Hz 外部クロック モード: このモードでは、WCO_IN はフローティングで、WCO_OUT は 32768 Hz の矩形波クロックで外部駆動され、グランドと VDDD の電源レベルをトグルします。この設定では、WCO_OUT ピンは外部クロックのデジタル入力ピンとして機能します。BACKUP_CTL の PRESCALER ビット フィールドは、プリスケーラ値 32768 に設定されています。
- 60 Hz 外部クロックモード: このモードは、60 Hz の AC 主電源からクロックを得るために使用できます。このモードでは、WCO_OUT はフローティングとなり、WCO_IN は 100:1 の容量分圧器をとおして 60 Hz/120 V の主電源から得られるゼロ DC オフセットの外部正弦波で駆動されます。例えば、WCO_IN とグランド間に 220 pF/6 V のコンデンサを接続し、WCO_IN と 60 Hz/120 V 主電源入力間に 2.2 pF//200 V のコンデンサを接続することで、適切な容量分割器が形成されます。BACKUP_CTL の PRESCALER ビット フィールドはプリスケーラ値 60 に設定されています。
- 50 Hz 外部クロックモード: このモードは 60 Hz モードと似ており、50 Hz/220 V の主電源規格に使用されます。先に説明した容量分圧器は、WCO_IN と主電源入力の間に 1 pF/250 V のコンデンサを持つことで、このタイプの電源に合うように修正されています。BACKUP_CTL の PRESCALER ビット フィールドはプリスケーラ値 50 に設定されています。

21.4.2 キャリブレーション

クロック入力の絶対周波数は、BACKUP_CAL_CTL レジスタを使って校正できます。キャリブレーションは、BACKUP_CTL の PRESCALER ビットフィールドが 32768 に設定されているときのみ機能します。

CALIB_VAL は BACKUP_CAL_CTL レジスタの 6 ビット フィールドで、絶対周波数 (固定温度) のキャリブレーション値を保持します。このフィールドの 1 ビットは、クロックカウントから 128 Tick を追加または削除したことになります。したがって、CAL_COMP_PER_MIN がデフォルト値に設定されている場合、各ビットは 1.085 ppm (128/(60*60*32768)) の変化に相当します。正の値 0x01～0x3c (1～60) はパルスを追加し、負の値はパルスを削除するため、CAL_COMP_PER_MIN がデフォルト値に設定されている場合、±65.1 ppm の範囲を与えます (このフィールドの範囲ではなく、毎時 60 分によって制限されます)。CAL_COMP_PER_MIN がデフォルト値に設定されている場合、2*CALIB_VAL の調整が完了するまで、キャリブレーションは 30 秒ごとに 64 目盛りとして適用され、59 分 59 秒から 1 時間ごとに実行されます。

BACKUP_CAL_CTL レジスタの CALIB_SIGN フィールドは、1 秒をカウントするために刻みを加算する (1 秒をカウントするために必要なクロック刻みを少なくする) か、減算する (1 秒をカウントするために必要なクロック刻みを多くする) かを制御します。

21.5 リセット

PSOC™ Control C3 MCU リセット ソースは、バックアップ電源 (VDDBAK) が存在する限り、バックアップ システムをリセットできないパワーオン リセット (POR) やブラウンアウト検出 (BOD) のようなデバイス電源を監視します。さらに、内部および外部リセット: ウォッチドッグ タイマー (WDT) リセットおよび XRES は、バックアップ システムをリセットできません。リセットには以下の条件が必要です。

- すべての電源がバックアップ領域から取り外されていること
- ファームウェアは BACKUP_RESET レジスタの RESET ビットフィールドを使用してバックアップドメインリセットがトリガーされること

21.6 リアルタイム クロック

表 135 RTC フィールド

ビットフィールド名	ビット数	説明
RTC_SEC	6	カレンダー 秒、0～59
RTC_MIN	6	カレンダー 分、0～59
RTC_HOUR	5	カレンダー 時間、12/24 時間モードによる値 24 時間モードの場合、ビット RTC_HOURS[20:16] = 0～23 12 時間モードの場合、ビット RTC_HOURS[20]:0= AM、1= PM、RTC_HOURS[19:16] = 1～12
CTRL_12HR	1	12 時間/24 時間モードを選択します。1=12 hr, 0=24 hr
RTC_DAY	3	カレンダー 曜日、1～7 値の意味の定義はユーザー次第、1=月曜日とすることを推奨します
RTC_DATE	5	カレンダー 日付、1～31 うるう年の自動修正
RTC_MON	4	カレンダー 月、1～12
RTC_YEAR	7	カレンダー 年、0～99

RTC は、AM/PM フラグ付きの 12 時間フォーマットと、"hours"フィールド用の 24 時間フォーマットの両方に対応しています。RTC には「曜日」フィールドもあり、1 から 7 までカウントされます。プログラマーは平日を'1'の値で表すように定義します。

RTC には、日付フィールド (月の日数) の自動的なうるう年補正機能が実装されています。年が 4 で割り切れる場合、2 月 (月 = 2) は 28 日ではなく 29 日となります。西暦が 2100 年 (年フィールドが 99 から 00 へロールオーバーする年) になると、うるう年補正が誤ります (2100 年はうるう年としてフラグが立てられますが実際にはうるう年ではありません)。そのため、ファームウェアが適切な処理を行えるように、割込みが発生します。この割込みはセンチュリー割込みと呼ばれます。

BACKUP_RTC_TIME および BACKUP_RTC_DATE のビット フィールドを含むユーザー レジスタ。詳細については、レジスタ リファレンスの対応するレジスタの説明を参照してください。ユーザー レジスタは高周波のバス クロック ドメインにあり、実際の RTC レジスタは低周波の 32768 Hz クロックで動作するため、RTC レジスタの読み出し/書き込みには特別な注意が必要です。これらは、以下の項で説明します。

21.6.1 RTC ユーザー レジスタの読み出し

読み出しトランザクションを開始するために、ファームウェアは BACKUP_RTC_RW レジスタの READ ビットに'1'を設定します。RTC レジスタはユーザー レジスタにコピーされ、ファームウェアがコヒーレントな RTC 値を安全に読み出せるようにフリーズされます。READ ビットをクリアしてトランザクションを完了します。

以下の場合、READ ビットは設定できません。

- RTC がまだ前の操作でビジー状態のとき (すなわち、BACKUP_STATUS レジスタの RTC_BUSY ビットが設定されています)。
- BACKUP_RTC_RW レジスタの WRITE ビットが設定されているとき

ファームウェアは、READ ビットが設定される前に、上記のビットが設定されていないことを確認します。

21.6.2 RTC ユーザー レジスタへの書き込み

BACKUP_RTC_RW レジスタの WRITE ビットが設定されると、RTC ユーザー レジスタにデータが書き込まれ、そうでなければ RTC ユーザー レジスタは無視されます。すべての RTC 書き込みが完了すると、ファームウェアは RTC 更新を有効に設定するために WRITE ビットをクリアします。WRITE ビットがクリアされた後、ハードウェアは実際の RTC レジスタとのコヒーレンシを確保するために、1 回の WCO クロックエッジですべての新しいデータをコピーします。

以下の場合、WRITE ビットは設定できません。

- RTC がまだ前の操作でビジー状態のとき (すなわち、BACKUP_STATUS レジスタの RTC_BUSY ビットが設定されています)
- BACKUP_RTC_RW レジスタの READ ビットが設定されているとき

ファームウェアは、RTC フィールドに書き込まれた値を更新し、法的に一貫したデータセットを形成します。ハードウェアは書き込まれた値の妥当性をチェックしません。不正な値を書き込むと、RTC の動作が未定義になります。

WRITE ビットが設定された RTC 更新の途中で、ブラウンアウト、リセット、またはディープスリープやハイバネートモードへの移行が発生すると、書き込み動作は完了しません。WRITE ビットはリセットによってクリアされ、RTC 更新はこのビットが WRITE トランザクションによってクリアされたときにのみトリガーされるからです。書き込み動作中 (RTC_BUSY) であれば、データ破損が発生します。それ以外の場合、システムはリセットされるか、ディープスリープまたはハイバネートモードに入ります。

RTC フィールドの 1 つまたはいくつかのみを更新する場合 (例えば、RTC を夏時間 (DST) に合わせて時刻のみ調整)、時間のフィールドのみ更新され、秒と分は更新されずに動作が継続されます。この場合、RTC フィールドごとに「更新」フラグが維持されます。BACKUP_RTC_RW.WRITE ビットがクリアされると、更新されたフィールドが実際の RTC にコピーされます。

21.7 アラーム機能

アラーム機能により、RTC は割込みを発生させることができ、スリープ、ディープスリープ、ハイバネート電力モードからシステムをウェイクアップさせるために使用されます。

アラーム機能は、RTC のフィールドに対応する 6 つのフィールドで構成されています。月/日, 曜日, 時: 分: 秒。各アラーム フィールドには、マッチングを有効に設定するためにセットされるイネーブル ビットがあります。ビットがクリアされると、そのフィールドはマッチングで無視されます。

アラームビットフィールドは以下のとおりです。

表 136 アラーム ビット フィールド

ビットフィールド名	Bit_Pos	Register_name	説明
ALM_SEC	5:0	BACKUP_ALMx_TIME	アラーム秒、0～59
ALM_SEC_EN	7	BACKUP_ALMx_TIME	アラーム秒を有効に設定します。0=無視、1=一致
ALM_MIN	13:8	BACKUP_ALMx_TIME	アラーム分、0～59
ALM_MIN_EN	15	BACKUP_ALMx_TIME	アラーム分を有効に設定します。0=無視、1=一致

(続く)

表 136 (続き) アラーム ビット フィールド

ビットフィールド名	Bit_Pos	Register_name	説明
ALM_HOUR	20:16	BACKUP_ALMx_TIME	アラーム時間、12/24 HR モードによる値 24 HR: [4:0]=0〜23 12 HR: [4]:0=AM, 1=PM, [3:0]=1〜12
ALM_HOUR_EN	23	BACKUP_ALMx_TIME	アラーム時間を有効に設定します。0=無視、1=一致
ALM_DAY	26:24	BACKUP_ALMx_TIME	アラーム曜日、1〜7 値の意味の定義はユーザー次第、1=月曜日を推奨します。
ALM_DAY_EN	31	BACKUP_ALMx_TIME	アラーム曜日を有効に設定します。0=無視、1=一致
ALM_DATE	4:0	BACKUP_ALMx_DATE	アラーム日付、1〜31。 うるう年補正
ALM_DATE_EN	7	BACKUP_ALMx_DATE	アラーム日付を有効に設定します。0=無視、1=一致
ALM_MON	11:8	BACKUP_ALMx_DATE	アラーム月、1〜12
ALM_MON_EN	15	BACKUP_ALMx_DATE	アラーム月を有効に設定します。0=無視、1=一致
ALM_EN	31	BACKUP_ALMx_DATE	アラーム 1 のマスターを有効に設定します。 0: アラーム 1 が無効。日付と時刻のフィールドは無視されます。 1: アラーム 1 が有効。アラームは、新しい日付と時刻が有効なすべての日付と時刻フィールドに一致したときにトリガーされます。日時フィールドがいずれも有効でない場合、このアラームは 1 秒に 1 回作動します。

マスターイネーブル (ALM_EN) が設定されているが、日付と時刻のすべてのアラームフィールドが無効になっている場合、アラーム割込みは 1 秒に 1 回発生します。なお、年のアラーム欄はありません。

PSOC™ Control C3 MCU には 2 つの独立したアラームがあります。詳細については、レジスタ リファレンス マニュアルの BACKUP_ALM1_TIME、BACKUP_ALM1_DATE、BACKUP_ALM2_TIME、BACKUP_ALM2_DATE レジスタを参照してください。

注: *アラーム ユーザー レジスタは、RTC ユーザー レジスタと同様に、リファレンス マニュアルの RTC ユーザー レジスタの読み出しと RTC ユーザー レジスタへの書き込み で説明されているように、読み出し/書き込み操作の前に特定の特別な手順を必要とします。*

RTC が割込み/ウェイクアップ イベントを生成するためには、割込みが適切に設定されていなければいけません。また、ハイバネート モードからデバイスをウェイクアップさせるために RTC 割込みを有効に設定するためには、PWR_HIB_WAKE_CTL レジスタの HIB_WAKE_RTC ビットを設定する必要があります。詳細は、デバイス電力モードの章と割込みの章を参照してください。

BACKUP_INTR_MASK レジスタは、Backup システムからの特定の割込みを無効に設定するために使用されます。

表 137 割込みマスク ビット

ビット名	説明
ALARM1	ALARM1 による割込みのマスクビット
ALARM2	ALARM2 による割込みのマスクビット
CENTURY	センチュリー割込みのマスクビット (年フィールドが 99 から 00 にロールオーバーしたときに割込みが発生する)

21.8 バックアップレジスタ

バックアップ ドメインには 32 本のレジスタ (BACKUP_BREG_SET0～BACKUP_BREG_SET3) があります。これらのレジスタは、それぞれ 1 バイトの重要なユーザー情報/フラグを格納するために使用できます。これらのレジスタは、バックアップ電源 (VBACKUP) が存在する限り、メイン電源 (VDDD) がオフであってもその内容を保持できます。これらのレジスタは、デバイスがハイバネート モードに入ったときに保持しなければいけない情報を格納するためにも使用できます。

21.9 レジスタの概要

表 138 バックアップレジスタ

レジスタ名	説明
BACKUP_CTL	メイン コントロール レジスタ (電源とクロックを含む)
BACKUP_RTC_RW	RTC 読み出し/書き込み制御レジスタ
BACKUP_STATUS	ステータス レジスタ
BACKUP_RTC_TIME	カレンダーの秒、分、時、曜日
BACKUP_RTC_DATE	カレンダーの月日、月、年
BACKUP_ALM1_TIME	アラーム 1 秒、分、時、曜日
BACKUP_ALM1_DATE	アラーム 1 月日と月
BACKUP_ALM2_TIME	アラーム 2 秒、分、時、曜日
BACKUP_ALM2_DATE	アラーム 2 月日と月
BACKUP_INTR	割込み要求レジスタ
BACKUP_INTR_MASK	割込みマスクレジスタ

(続く)

表 138 (続き) バックアップレジスタ

レジスタ名	説明
BACKUP_INTR_MASKED	割込みマスク要求レジスタ
BACKUP_BREG_SET0～BACKUP_BREG_SET3	バックアップレジスタ
BACKUP_RESET	バックアップドメインのリセットレジスタ

22 リセット システム

PSOC™ Control C3 は、電源投入時に誤りのない動作を保証し、ユーザーが供給する外部ハードウェアまたは内部ソフトウェアによるリセット信号に応じてデバイスのリセットを可能にする複数のリセット タイプをサポートします。リセットの範囲は広く、一般的に電源ドメインやグローバル電力モードに沿ったもので、アクティブ モードで終了するリブートを引き起こします。ブロックによっては、それぞれの章で説明されているローカル リセットを持つ場合があります。リセットのアサートは非同期に伝搬され、リセットのデアサートは使用される各クロックドメインに同期されます。さらに、PSOC™ Control C3 には、発生したリセットを報告する機能を持つハードウェアが含まれています。

さらに、フォールトマネージャはフォールトを NMI などの高優先度割込みに変換し、プロセッサがメモリ書き込みの停止や周辺機能を解放するなど、安全状態に戻れるようにします。その後、プロセッサは自身のローカル リセットまたはシステム リセットをトリガーできます。これにより、クロック監視、電源監視、マルチカウンター ウォッチドッグ タイマーなどの安全回路によって発生したフォールトからの回復が可能になります。

22.1 機能

PSOC™ Control C3 MCU には以下のリセット ソースがあります。

- 通常動作中に電源がデバイスの仕様を下回った場合にデバイスをリセットするブラウンアウト検出 (BOD)
- 外部リセット (XRES): 外部入力を使用し、デバイスをリセット
- ウォッチドッグ タイマー (WDT/MCWDT) リセット: ファームウェアの実行がウォッチドッグ タイマーの定期的なサービスに失敗した場合にデバイスをリセット
- ソフトウェア リセット (SYS): 必要に応じてファームウェアを使用してデバイスをリセット
- テストコントローラーまたはデバッガがアサートされたリセット (TC_DEBUG)
- 特定のフォールトが発生した場合にデバイスをリセットするフォールト検出リセット (FAULT_ACT)
- 高周波クロック関連エラーに対するクロック監視トリガー リセット (CSV_HF)
- 基準クロック関連エラーのためのクロック監視トリガー リセット (CSV_REF)
- ハイバネート ウェイクアップ リセット (HIB WAKEUP) により、デバイスをハイバネート低消費電力モードから復帰
- ソフトウェアから XRES と同等のリセットをトリガーするプログラマブル XRES (PXRES)

22.2 リセットソース

以下のセクションでは、PSOC™ Control C3 MCU ファミリーのリセットソースについて説明します。表 139 にはすべてのリセット ソースのリストが含まれています。

表 139 PSOC™ Control C3 MCU リセットソース

リセットソース	リセット条件	電力モードでの利用可能性	測定可否	SRAM リテンション
外部リセット (XRES)	外部リセット ピンがアサート ロー	すべて	不可	不可
ブラウンアウト検出 (BOD)	VDDD、VCCD、バックアップ電源のブラウンアウト検出	オフ、XRES、ハイバネートを除くすべて	不可	不可
WDT リセット	ウォッチドッグ タイマーによるリセット動作	アクティブ, スリープ, ディープスリープ	可能	可能 [1)]

(続く)

表 139 (続き) PSOC™ Control C3 MCU リセットソース

リセットソース	リセット条件	電力モードでの利用可能性	測定可否	SRAM リテンション
MCWDT リセット	マルチカウンター WDT トリガーリセット動作	アクティブ, スリープ, ディープスリープ	可能	可能 [1]
ハイバネート ウェイクアップ (HIB_WAKEUP)	ハイバネートモードから復帰	ハイバネート	可能 (PWR_HIBERNATE.TOKEN フィールド上)	不可
内部システム リセット (SYS)	システム リセット (SYSRESETREQ を使用して CPUSS が内部で生成)	アクティブ, スリープ	可能	可能 [1]
テスト/デバッグ リセット (TC_DEBUG)	テストコントローラーまたはデバッガがリセットをアサート	アクティブ, スリープ	可能	可能 [1]
フォールト検出アクティブロジック (FAULT_ACT)	フォールトマネージャによるリセット	アクティブ, スリープ	可能	可能 [1]
クロック監視 (CSV_HF)	高周波クロック HF0～HF6 の損失または周波数違反。	アクティブ, スリープ	可能	CSV_HF0: 不可 [2] CSV_HF1..6: 可能 [1]
クロック監視 (CSV_REF)	他の高周波クロックソースをモニターするために使用される基準クロックソースの損失または周波数違反。	アクティブ, スリープ	可能	不可
プログラマブル XRES (PXRES)	XRES と同様のスコープを持つ CPU トリガーリセット (RES_PXRES_CTL.PXRES_TRIGGER を使用してトリガーされます)。	アクティブ, スリープ	不可	不可

1) リセットがトリガーされる前に、プロセッサによって処理される順序どおりのシャットダウンを想定しています
2) システム RAM は clk_hf0 によってクロック制御されるため、CSV_HF0 によってトリガーされたリセットは RAM クロックの不安定な発振や、クロック信号への悪影響を引き起こすことがあります。

注: *いずれもバックアップシステムはリセットできません。バックアップドメインがリセットされるのは、すべての電源が取り除かれたとき(コールド スタートとも呼ばれる)、またはファームウェアが BACKUP_RESET レジスタを使用してリセットをトリガーしたときのみです。詳細については、バックアップを参照してください。*

表 140 に、リセット ソースとリセットの影響を受けるデスティネーションのマッピングを示します。'X'は、そのブロックが対応するリセットによって影響を受けるレジスタ/ビットを少なくともいくつか持っていることを示します。

表 140　PSOC™ Control C3 MCU リセット要因分布

リセットタイプ/ソース	リセット要因ビット	FAULT 構造のデータレジスタ	バックアップレジスタ	デバッグ ロジック	CM33
XRES	X	X		X	X
BOD	X	X	X[1)]	X	X
WDT		X		X[2)]	X
MCWDT		X		X[2)]	X
HIB_WAKEUP	X	X		X	X
SYS		X			X
TC_DEBUG		X		X	X
FAULT_ACT		X			X
CSV_HF		X			X
CSV_REF		X			X

1)　バックアップ電源がない場合
2)　リセットソースがディープスリープ中にトリガーされた場合

***注**:　ウォッチドッグ フォールトとクロック監視フォールトは、フォールト処理システムがクラッシュした可能性があるため、特殊なケースです。例えば、クロック監視フォールトは、クロックが故障したときにクロックのグリッチが発生したことを意味するかもしれません。可能性は低いが、この不具合はフォールト処理システムをロックすることがあります。この場合、フォールト記録システムはフォールトを記録しないかもしれませんが、リセットを観察できます。*

22.2.1　ブラウンアウトリセット

ブラウンアウト検出回路は、デバイスのデジタル電源電圧 (VDDD) と内部生成電源電圧 (VCCD) を監視し、それらが電圧閾値を下回るとリセットを生成します。すべてのブラウンアウト検出が解除されるまで、デバイスはリセット状態になります。これは、最初のパワーランプ中にも発生します。詳細については電源と電圧監視の章を参照してください。

22.2.2　WDT/MCWDT リセット

ウォッチドッグ タイマー リセットは、WDT またはマルチ カウンター WDT (MCWDT) が指定された制限時間内にファームウェアによって処理がされない場合にリセットします。ウィンドウ モードでは、MCWDT は指定された時間より早くサービスされるとリセットを生成します。詳細についてはウォッチドッグ タイマーの章を参照してください。

22.2.3　ソフトウェア リセット

ソフトウェア リセットは、ソフトウェアまたは接続されたデバッガがシステム リセットを要求できるようにするメカニズムです。Cortex®-M33 は、RES_SOFT_CTL レジスタの TRIGGER_SOFT ビットに'1'を書き込むことでリセットを要求できます。これは、デバッガ プローブを介して行うこともできます。

RES_SOFT_CTL レジスタを使用したソフトウェア リセットに加えて、この MCU には、Arm® Cortex®-M33 から XRES をトリガーするオプションもあります。XRES 割込みは RES_PXRES_CTL レジスタの PXRES_TRIGGER ビットに'1'を書き込むことでトリガーされます。RES_PXRES_CTL は、XRES と同等のフルスコープ リセットをトリガーするソフトウェア アクセス可能なレジスタです。

22.2.4 外部リセット

外部リセット (XRES) は、外部信号によってトリガーされるリセットで、アサートされると即座にシステム リセットを引き起こします。XRES ピンはアクティブ ローで、ロジック‘1’では何の動作も起こさないのに対し、ロジック‘0’でリセットされます。このピンはデバイス内部でロジック'1'にプルされ、専用ピンとして利用可能です。端子配置の詳細については、デバイス データシートの端子配置のセクションを参照してください。

XRES ピンは、このピンの入力が'0'である限り、デバイスをリセット状態に保持します。ピンが解放される (ロジック'1'に変わる) と、デバイスは通常のブート シーケンスに従います。XRES の論理スレッショルドとその他の電気的特性は、デバイス データシートの電気的仕様セクションに記載されています。XRES はすべての電力モードで使用できますが、バックアップシステムのリセットはできません。

22.2.5 デバッガリセット

デバッガ リセットは、テスト コントローラーまたはデバッガによってアサートされます。SWD/JTAG-DP の DP_CTL_STAT レジスタにはリセットリクエストビット (CDBGRSTREQ) があり、このリクエストはリセットコントローラーに送られ、デバッグリセット信号が生成されます。

デバッガ リセットは、RES_CAUSE レジスタの RESET_TC_DBGRESET ビット フィールドから識別できます。

22.2.6 フォールト検出リセット

PSOC™ Control C3 のフォールト レポート構造は、ユーザー設定可能なフォールトに対してリセットを要求するように設定できます。

ディープスリープ中に発生したフォールトはシステムをウェイクさせませんが、ウェイク時には保留されます。ディープスリープからのウェイクが必要な状態の場合は、WIC への割込みとして設定します。

クロック監視と MCWDT によって生成されたフォールトは、フォールト システムもまた故障している可能性を示すことがあります。これらの回路は、直接リセットを引き起こすように設定できます。

22.2.7 高周波クロックエラーリセット

クロック監視ロジックは、高周波クロックまたは水晶発振子クロックの損失、あるいは高周波クロックエラーによりリセットを開始します。RESET_CSV_HF は、RES_CAUSE2 レジスタの 7 ビット フィールドで、高周波クロックの損失または周波数違反によるリセットを識別するために使用できます。

クロックの詳細については、クロック システムの章を参照してください。

22.2.8 基準クロックエラーリセット

クロック監視ロジックは、他の高周波クロックソースを監視するために使用される基準クロックソースの損失または周波数違反により、リセットを要求します。RES_CAUSE2 レジスタの RESET_CSV_REF フィールドは、他の高周波クロックソースをモニターするために使用される基準クロックソースの損失または周波数違反によって引き起こされるリセットを識別するために使用できます。

クロックの詳細については、クロック システムの章を参照してください。

22.2.9 ハイバネート ウェイクアップ リセット

ハイバネート ウェイクアップ リセットは、ハイバネート ウェイクアップ ソースの 1 つがデバイス リセットを実行し、アクティブ電力モードに戻るときに発生します。ハイバネートモードと利用可能なウェイクアップ ソースの詳細については、デバイス電力モードの章を参照してください。

TOKEN は PWR_HIBERNATE レジスタの 8 ビット フィールドで、ハイバネート ウェイクアップ シーケンスをとおして保持されます。ファームウェアはこのビットフィールドを使用して、ハイバネート ウェイクアップと一般的なリセット イベントを区別できます。

22.3 リセット ソースの識別

デバイスがリセット解除後にリセットの要因を知ることは有益です。リセット要因は RES_CAUSE と RES_CAUSE2 レジスタに記録されます。このレジスタは、いくつかのリセット ソースを示す特定のステータス ビットを持っています。WDT リセット、MCWDT リセット、ソフトウェア リセット、ロジック保護フォールト リセット、デバッガ リセット、クロック監視リセットの発生を記録します。ただし、BOD、XRES、PXRES、ハイバネート ウェイクアップ リセットの発生は記録されません。これらのレジスタのビットは、対応するリセットの発生時に設定され、ファームウェアによってクリアされるか、または XRES や BOD リセットなどの保持が失われるまで、リセット後も設定されたままになります。

ハイバネート ウェイクアップ リセットは、前述のように PWR_HIBERNATE レジスタの TOKEN フィールドを調べることによって検出できます。ただし、XRES で発生する、ハイバネート ウェイクアップ リセットは検出不可です。表 141 に RES_CAUSE と RES_CAUSE2 レジスタのリセット要因を示します。

リセット要因を特定し評価した後、RESET_CAUSE および RESET_CAUSE2 レジスタをクリアする必要があります。この手順は、次のリセットをキャプチャするために必要です。

表 141 リセット ソースを検出するためのリセット要因ビット

レジスタ [bit pos]	ビットフィールド	ビット数	説明
RES_CAUSE[0]	RESET_WDT	1	最後のパワー サイクル以降にハードウェア WDT リセットが発生しました。
RES_CAUSE[1]	RESET_ACT_FAULT	1	フォールト ロギング システムがアクティブ ロジックからのリセットを要求しました。
RES_CAUSE[3]	RESET_TC_DBGRESET	1	テストコントローラーまたはデバッガがリセットをアサートしました。デバッグドメインのリセットのみ。
RES_CAUSE[4]	RESET_SOFT	1	CPU が TRIGGER_SOFT を通じてシステム リセットを要求しました。これは、デバッガ プローブまたはファームウェアから行えます。
RES_CAUSE[5]	RESET_MCWDT0	1	マルチカウンター WDT リセットが最後の電源サイクル以降に発生しました。
RES_CAUSE2[6:0]	RESET_CSV_HF	7	クロック監視ロジックは、高周波クロックの損失または周波数違反によるリセットを要求しました。各ビットインデックス K は HFCLK<K>に対応します。

(続く)

表 141 (続き) リセット ソースを検出するためのリセット要因ビット

レジスタ [bit pos]	ビットフィールド	ビット数	説明
RES_CAUSE2[16]	RESET_CSV_REF	1	クロック監視ロジックは、他の高周波クロックソースを監視するために使用される基準クロックソースの損失または周波数違反によるリセットを要求しました。

詳細については、レジスタのリファレンス マニュアルの RES_CAUSE および RES_CAUSE2 レジスタを参照してください。
これらの方法でリセットの要因を検出できない場合は、記録されていない非保持リセットの可能性があります。これらのリセットは、オンチップ リソースを使用した区別はできません。

22.4 レジスタの概要

表 142 リセット システム レジスター覧

レジスタ	説明
RES_CAUSE	リセット要因監視レジスタ
RES_CAUSE2	リセット要因監視レジスタ 2
PWR_HIBERNATE	ハイバネート電力モード制御レジスタ。ハイバネート ウェイクアップ リセットを検出するために使用できる TOKEN フィールドを含みます。
RES_SOFT_CTL	ソフト リセット トリガー レジスタ
RES_PXRES_CTL	プログラマブル外部リセット (PXRES) の制御レジスタ。

23 I/O サブシステム

ここでは、PSOC™ Control C3 MCU の I/O サブシステム、その機能、アーキテクチャ、動作モード、割込みについて説明します。I/O サブシステムは CPU コアとペリフェラル コンポーネント間のインターフェースを外部に提供します。PSOC™ Control C3 MCU の機能は、ほとんどのピンへの効率的な信号配線を可能にし、回路設計と基板レイアウトを簡素化します。この MCU ファミリーの GPIO ピンはポートにグループ化されており、1 つのポートには最大 8 つの GPIO ピンを持つことができます。

ここでは以下のことを説明します。

- 特長
- アーキテクチャ
- 機能説明
- プログラマブル デジタル (スマート I/O)
- レジスタの概要

23.1 特長

PSOC™ Control C3 GPIO には以下の特長があります。

- アナログ-デジタル入出力機能
- 8 つのドライブ強度モード
- 読み出し用と書き込み用に分かれたポートレジスタ
- 全 GPIO における立ち上りエッジ、立ち下りエッジ、または両エッジでのエッジトリガー型割込み
- スルーレート制御
- 状態をラッチするためのホールドモード (ディープスリープモードで I/O 状態を保持するために使用)
- CMOS、TTL 入力バッファモード選択可能
- スマート I/O は I/O 信号経路にブール関数を実行する能力を提供
- セキュリティを考慮した GPIO コンフィギュレーション

23.2 アーキテクチャ

PSOC™ Control C3 MCU はアナログとデジタルのペリフェラルを備えています。図 80 に、ペリフェラルとピン間の配線の概要を示します。

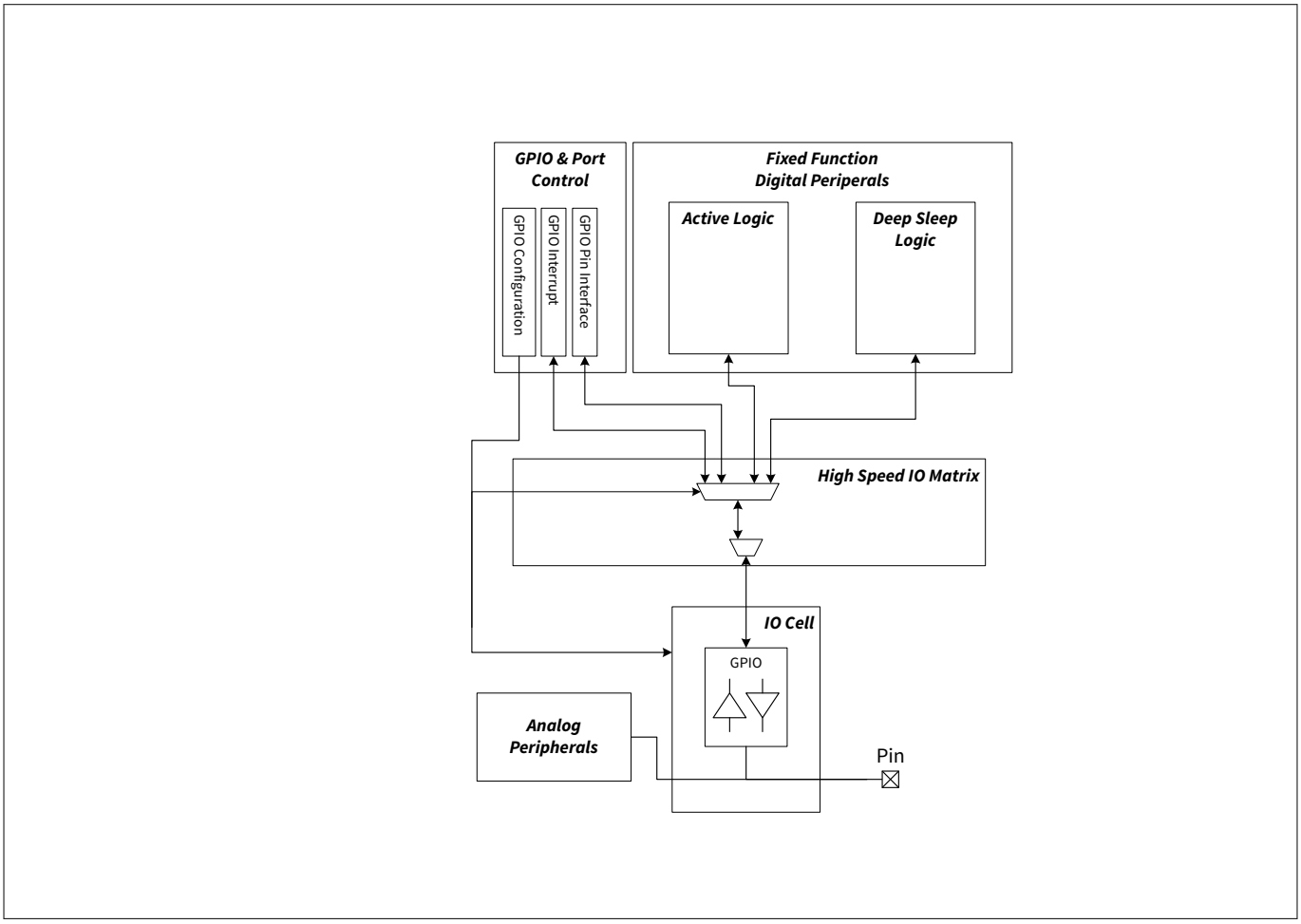


図 80 GPIO インターフェースの概要

GPIO ピンは I/O セルに接続されています。これらのセルは、高入力インピーダンスを提供するデジタル入力用の入力バッファと、デジタル出力信号用のドライバを備えています。また、高性能プログラマブル アナログ サブシステム (HPPASS) セクションで説明されているように、MCU には最大 16 の専用アナログ ピンがあります。

デジタル ペリフェラルは、高速 I/O マトリックス (HSIOM) を介して I/O セルに接続されます。各ピンの HSIOM には、選択されたペリフェラルとピンを接続するためのマルチプレクサが含まれています。さらに HSIOM は、デジタル システム インターコネクト (DSI) とピン間の接続を橋渡しします。

23.2.1 I/O セル アーキテクチャ

図 81 に各 GPIO セルに存在する I/O セル アーキテクチャを示します。デジタル入出力信号用の HSIOM マルチプレクサに接続する入力バッファと出力ドライバで構成されます。アナログ ペリフェラルは、ポイント ツー ポイント接続のためにピンに直接接続します。

GPIO コンポーネントは、レジスタをとおして I/O セル コンフィギュレーション情報を提供します。これらのレジスタはディープスリープ電力モードでは保持されますが、ハイバネート電力モードではデフォルト値にリセットされます。ハイバネート割込み機能を有効に設定するために、I/O セルは、ディープスリープまたはハイバネート電力モードに入るとき、コンフィギュレーション情報を保持またはフリーズします。その結果、コンフィギュレーション信号をアクティブ電源ドメインで配線できます。

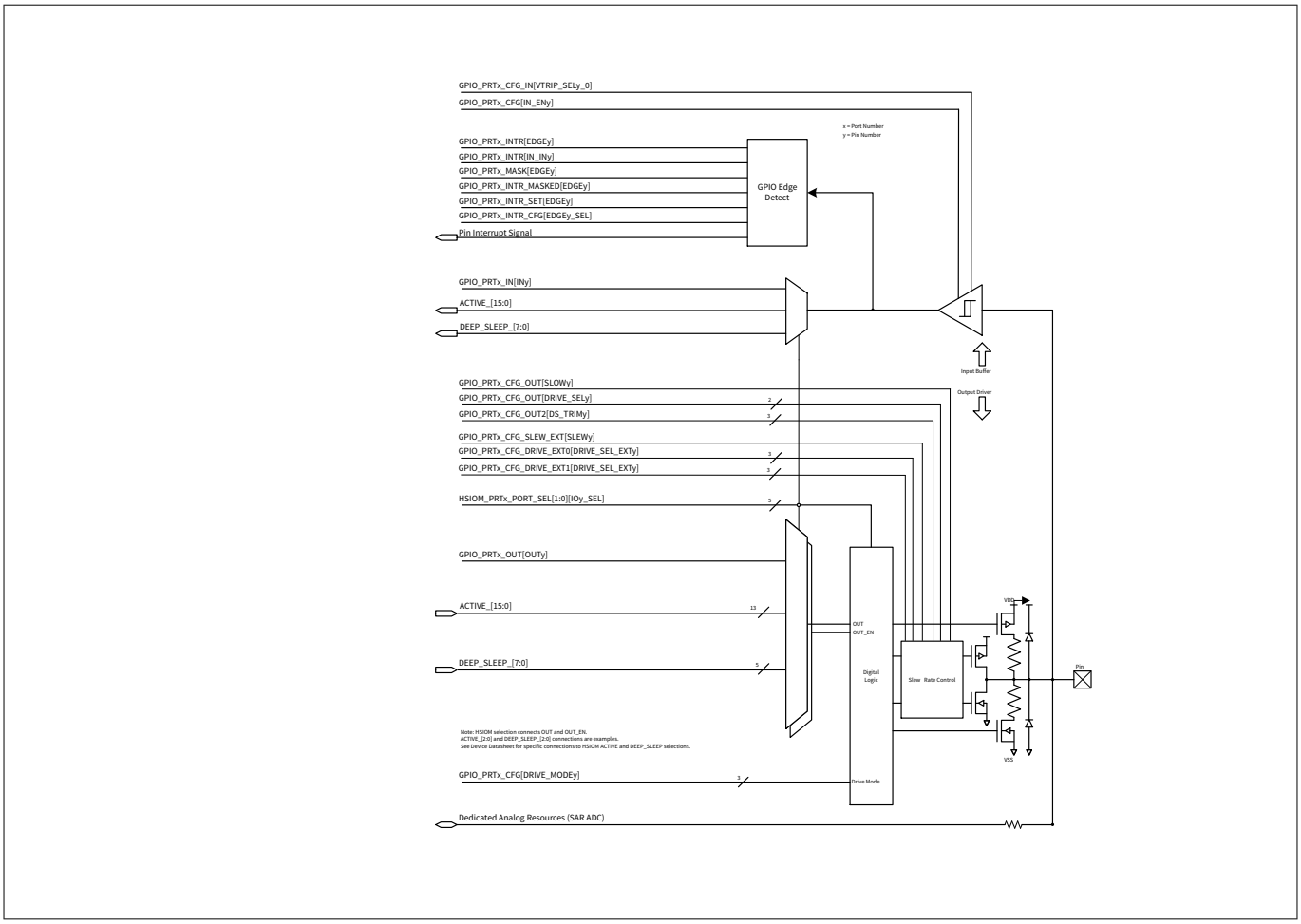


図 81　　GPIO セルアーキテクチャ

HSIOM が I/O セルに機能的に接続する場合、GPIO はコンフィギュレーション情報を提供します。HSIOM が I/O セルにテスト接続 (スキャン、パラレル テスト モード、JTAG) する場合、GPIO コンフィギュレーション情報は無視されます。そして、HSIOM は必要なコンフィギュレーション情報を提供します。

I/O セル コンフィギュレーションには、ドライブモード (プルアップ/プルダウン) やドライブ強度などの情報が含まれます。コンフィギュレーション情報は、特定の I/O パッドに対応したもので、ドライブモード、ドライブ強度、高速スルー制御と低速スルー制御の切り替え、入力バッファモードなどがあります。

I/O ポートの I/O は、I/O 出力信号の制御性と I/O 入力信号の観測性を提供するために、ソフトウェアからアクセス可能です。制御可能性と可観測性を組み合わせることで、ソフトウエアのビットバンギング機能を提供します。

I/O セルの GPIO データ入力とデータ出力/データ出力イネーブル信号は HSIOM 機能接続にあります。特定の接続は HSIOM レジスタ フィールドの制御下にあります。

23.2.2　　GPIO に代替機能をマッピングする HSIO

High-Speed I/O Matrix (HSIOM) は、CPU およびペリフェラルの信号を GPIO とやり取りする高速マルチプレクサです。HSIOM を使用すると、GPIO を複数の機能と共有し、選択したペリフェラルのピン接続を多重化できます。`HSIOM_PRTx_PORT_SEL[1:0]`レジスタは、表 143 にリストされているように、各ピンへの最大 32 の異なる接続から 1 つの選択を可能にします。

表 143 HSIOM 接続

代替機能 5 ビット [4:0]	名称	デジタルドライバ信号ソース		デジタル信号接続先	説明
		OUT	OUT_EN		
0	GPIO	OUT レジスタ	1	IN レジスタ	GPIO_PRTx_OUT レジスタは"出力"を制御します
1	予約済み	-	-	-	-
2	予約済み	-	-	-	-
3	予約済み	-	-	-	-
4	予約済み	-	-	-	-
5	予約済み	-	-	-	-
6	予約済み	-	-	-	-
7	予約済み	-	-	-	-
8	ACT_0	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 0 - 具体的なピン接続についてはデータシートを参照してください。
9	ACT_1	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 1 - 具体的なピン接続についてはデータシートを参照してください。
10	ACT_2	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 2 - 具体的なピン接続についてはデータシートを参照してください。
11	ACT_3	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 3 - 具体的なピン接続についてはデータシートを参照してください。
12	DS_0	ディープスリープ ソース OUT	ディープスリープ ソース OUT_EN	ディープスリープ IN	ディープスリープ機能 0 - 具体的なピン接続についてはデータシートを参照してください。

(続く)

表 143 (続き) HSIOM 接続

代替機能 5 ビット [4:0]	名称	デジタルドライバ信号ソース		デジタル信号接続先	説明
		OUT	OUT_EN		
13	DS_1	ディープスリープ ソース OUT	ディープスリープ ソース OUT_EN	ディープスリープ IN	ディープスリープ機能 1 - 具体的なピン接続についてはデータシートを参照してください。
14	DS_2	ディープスリープ ソース OUT	ディープスリープ ソース OUT_EN	ディープスリープ IN	ディープスリープ機能 2 - 具体的なピン接続についてはデータシートを参照してください。
15	DS_3	ディープスリープ ソース OUT	ディープスリープ ソース OUT_EN	ディープスリープ IN	ディープスリープ機能 3 - 具体的なピン接続についてはデータシートを参照してください。
16	ACT_4	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 4 - 具体的なピン接続についてはデータシートを参照してください。
17	ACT_5	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 5 - 具体的なピン接続についてはデータシートを参照してください。
18	ACT_6	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 6- 具体的なピン接続についてはデータシートを参照してください。
19	ACT_7	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 7 - 具体的なピン接続についてはデータシートを参照してください。
20	ACT_8	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 8 - 具体的なピン接続についてはデータシートを参照してください。

(続く)

表 143 (続き) HSIOM 接続

代替機能 5 ビット [4:0]	名称	デジタルドライバ信号ソース		デジタル信号接続先	説明
		OUT	OUT_EN		
21	ACT_9	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 9 - 具体的なピン接続についてはデータシートを参照してください。
22	ACT_10	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 10 - 具体的なピン接続についてはデータシートを参照してください。
23	ACT_11	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 11 - 具体的なピン接続についてはデータシートを参照してください。
24	ACT_12	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 12 - 具体的なピン接続についてはデータシートを参照してください。
25	ACT_13	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 13 - 具体的なピン接続についてはデータシートを参照してください。
26	ACT_14	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 14 - 具体的なピン接続についてはデータシートを参照してください。
27	ACT_15	アクティブ ソース OUT	アクティブ ソース OUT_EN	アクティブ ソース IN	アクティブ機能 15 - 具体的なピン接続についてはデータシートを参照してください。

(続く)

表 143 (続き) HSIOM 接続

代替機能 5 ビット [4:0]	名称	デジタルドライバ信号ソース		デジタル信号接続先	説明
		OUT	OUT_EN		
28	DS_4	ディープスリープソース OUT	ディープスリープソース OUT_EN	ディープスリープ IN	ディープスリープ機能 4 - 具体的なピン接続についてはデータシートを参照してください。
29	DS_5	ディープスリープソース OUT	ディープスリープソース OUT_EN	ディープスリープ IN	ディープスリープ機能 5 - 具体的なピン接続についてはデータシートを参照してください。
30	DS_6	ディープスリープソース OUT	ディープスリープソース OUT_EN	ディープスリープ IN	ディープスリープ機能 6 - 具体的なピン接続についてはデータシートを参照してください。
31	DS_7	ディープスリープソース OUT	ディープスリープソース OUT_EN	ディープスリープ IN	ディープスリープ機能 7 - 具体的なピン接続についてはデータシートを参照してください。

注: アクティブおよびディープスリープのソースはピンに依存しています。各ピンがサポートする機能の詳細については、デバイス データシートの「ピンアウト」セクションを参照してください。

23.2.3 電源投入時の I/O 状態

電源投入時、すべての GPIO はハイインピーダンス アナログ状態にあり、入力バッファは無効になっています。ランタイム中、関連するレジスタに書き込むことで GPIO を設定できます。ただし、デバッグ アクセス ポート (DAP) 接続をサポートするピン (SWD ライン) は、電源投入時に常に SWD ラインとして有効になります。これらのピンをフローティングのままにしておくと、DAP 接続にプルアップ抵抗やプルダウン抵抗がないため、過電流が流れる危険性があります。これを回避するためには、HSIOM を通じて DAP 接続を汎用的に使用できるように無効化または再設定できますが、これはデバイスが起動してコードを実行し始めた後に限られます。

23.2.4 低消費電力モードでの動作

表 144 に、低消費電力モードでの GPIO の状態を示します。

表 144　低消費電力モードでの動作

低消費電力モード	状態
スリープ	スタンダード GPIO、GPIO-OVT、および SIO ピンはアクティブで、TCPWM や SCB など、スリープ モードで動作可能なほとんどのペリフェラルで駆動できます。入力バッファが有効なため、あらゆる I/O 上の割込みを CPU をウェイクアップするために使用できます
ディープスリープ	ディープスリープ領域のペリフェラルに接続されている GPIO、GPIO-OVT、および SIO ピンは機能します。他のすべてのピンは、最後の出力ドライバの状態と設定を維持します。ピン割込みはすべての I/O で機能し、デバイスのウェイクアップに使用できます。
ハイバネート	ピンの出力状態とコンフィギュレーションはラッチされ、フリーズ状態のままです。ピン割込みは選択された IO でのみ機能し、デバイスのウェイクアップに使用できます。特定のハイバネート ピンの接続性については、デバイスのデータシートを参照してください。

23.3　機能説明

23.3.1　デジタル入力バッファ

デジタル入力バッファは、外部デジタル入力にハイインピーダンスのバッファを提供します。バッファは GPIO_PRTx_CFG.IN_ENy ビット ('x'はポート番号、'y'はピン番号) によって有効または無効になります。

入力バッファは HSIOM に接続され、CPU ポート レジスタと選択されたペリフェラルにルーティングされます。HSIOM ポート セレクト レジスタ (HSIOM_PRTx_PORT_SEL) へ書き込むことにより、ピン接続を選択します。各ピンで利用可能な接続についてはデバイスデータシートを参照してください。ポートピンは、入力としても出力としても同時に使用できます。ピンがアナログ信号のみに接続されている場合、過電流を避けるために入力バッファを無効に設定する必要があります。各ピンの入力バッファのトリップポイントとヒステリシスは、以下のモード用に設定可能です。

- CMOS + I2C
- TTL

GPIO_PRTx_CFG_IN.VTRIP_SELy_0 ビットで CMOS バッファモードと TTL バッファモードを選択します。GPIO_PRTxCFG_IN_AUTOLVL.VTRIP_SELy_1 ビットを使用して、産業用モードを有効または無効にします。

***注**:　産業用モードを有効にする場合は、GPIO_PRTx_CFG_IN モードを CMOS に設定します。図 82 に CMOS とオートモーティブのトリップレベルを示します。*

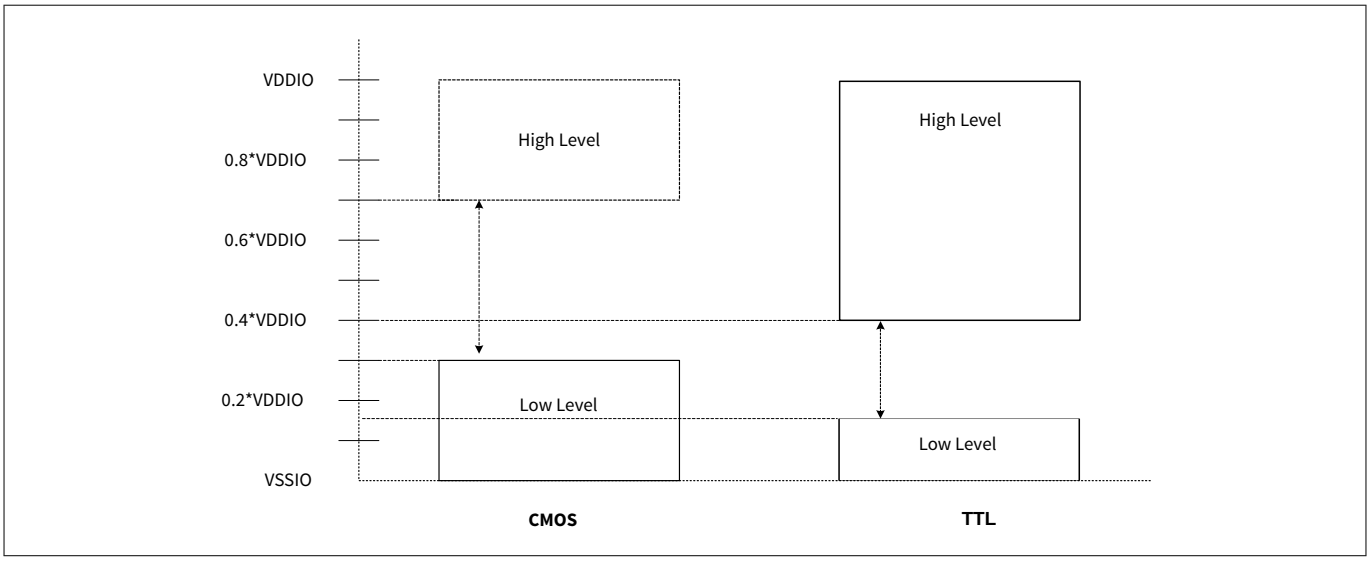


図 82　　入力バッファ モードのトリップ レベル

23.3.2　デジタル出力ドライバ

デジタル出力ドライバはピンをドライブします。デジタル出力信号のさまざまなドライブ モードとスルー レート制御を実現する回路で構成されています。HSIOM は、出力ドライバの制御ソースを選択します。2 つの主要な制御源は、ポートコンフィギュレーションレジスタと固定機能デジタルペリフェラルです。特定の HSIOM 接続は、HSIOM ポート選択レジスタ (HSIOM_PRTx_PORT_SEL) に書き込むことで選択されます。

各 GPIO ピンは、ピンの電圧が I/O 電源電圧を上回らないようにするための ESD ダイオードを備えています。ピンの電圧が I/O 電源電圧 VDDIO/VDDD/VDDA を超えたり、VSSIO/VSSD/VSSA を下回ったりしないようにしてください。GPIO 電圧の絶対最大値と最小値については、デバイスのデータシートを参照してください。

出力ピンに関連付けられた出力データ レジスタ (GPIO_PRTx_OUT) は、ハードウェアのデジタル出力ドライバを有効または無効にします。GPIO ポート以外のペリフェラルは、出力バッファの出力信号と出力イネーブル信号の両方を直接制御します。

23.3.2.1　スルーレート制御

一部の GPIO ピンには、ポート出力コンフィギュレーション レジスタ (GPIO_PRTx_CFG_OUT) の SLOW ビットを使用して設定された、強力なドライバ用の高速および低速出力スルー レート オプションがあります。このビットはデフォルトでクリアされ、ポートは高速のスルーレートで動作します。低速のスルーレートが必要な場合にこのビットをセットします。スルーレートを遅くすると、EMI やクロストークが減少するため、低周波信号やタイミング制約の厳しくない信号に推奨されます。

高速スルー レートに設定されている場合、ドライブ強度は、GPIO_PRTx_CFG_OUT.DRIVE_SELy を使用した以下の 4 つのレベルのいずれかに設定できます。ドライブ強度フィールドは、使用される出力ドライバのアクティブ部分を決定し、出力信号のスルー レートに影響を与える可能性があります。ドライブ強度のオプションは、フルドライブ強度 (デフォルト)、2 分の 1 強度、4 分の 1 強度です。低速スルー レート ビット (SLOW) が設定されているときは、ドライブ強度をフルに設定する必要があります。

表 145 GPIO_STD のドライブセレクト

ドライブセレクト	DRIVE_SEL[0:1]	説明
DRIVE_SEL_THREE	11	1/8 ドライブ強度: GPIO は最大定格仕様の 8 分の 1 の電流で駆動します。
DRIVE_SEL_TWO	10	1/4 ドライブ強度: GPIO は最大定格仕様の 4 分の 1 の電流で駆動します。
DRIVE_SEL_ONE	01	1/2 ドライブ強度: GPIO は最大定格仕様の 2 分の 1 の電流で駆動します
DRIVE_SEL_ZERO	00	フルドライブ強度: GPIO はその最大定格仕様で電流を駆動します。

23.3.2.2 ドライブ強度

I/O のドライブ強度は、VOH と VOL の最小レベルを維持しながら、I/O ピンがドライブおよびシンクできる電流量を指定します。出力が高速スルー レート用に設定されている場合、ポート出力コンフィギュレーション レジスタ (GPIO_PRTx_CFG_OUT) の DRIVE_SEL フィールドを使用して、ドライブ強度を 4 つのレベルのいずれかに設定できます。ドライブ強度フィールドは、使用される出力ドライバのアクティブ部分を決定し、出力信号のスルー レートに影響を与える可能性があります。ドライブ強度のオプションは、フルドライブ強度 (デフォルト)、2 分の 1 強度、4 分の 1 強度、8 分の 1 強度です。低速スルー レート ビット (SLOW) が設定されているときは、ドライブ強度をフルに設定する必要があります。

I/O ポートには、各ポートの各ピンのドライブ幅を設定するために、ポート出力ドライブセレクト拡張レジスタ (GPIO_PRTx_CFG_DRIVE_EXT[0-1]) と呼ばれる 2 つの高度なレジスタが用意されています。

また、各ポートの各ピンのドライブセレクト トリムを制御するためのポート出力コンフィギュレーション レジスタ 2 (GPIO_PRTx_CFG_OUT2) があります。

表 146 GPIO_PRTx_CFG_OUT2 のドライブセレクト トリム

ドライブセレクト	DS_TRIM[0:2]	説明
DEFAULT	000	デフォルト (50Ω)
DS_15OHM	111	15 Ω
DS_20OHM	110	20 Ω
DS_30OHM	101	30 Ω
DS_50OHM	100	50 Ω
DS_60OHM	011	60 Ω
DS_90OHM	010	90 Ω
DS_120OHM	001	150 Ω

23.3.2.3 ドライブモード

各 I/O は、ポート コンフィギュレーション レジスタ (GPIO_PRTx_CFG) の DRIVE_MODE[7:0]フィールドによって、8 つのドライブ モードの 1 つに個別に設定可能です。

表 147 は、GPIO_PRTx_CFG レジスタのドライブモードの一覧です。これは、2 つの異なるコラム (GPIO 機能と固定ペリフェラル機能) で異なるモードの包括的な内訳を提供します。

注: OUT_EN はユーザー設定可能ではなく、その値はピン モードに応じて設定されます。例えば、GPIO モードでは OUT_EN = 1 です。表 143 を参照してください。

表 147 ドライブモード設定

ドライブモード	値	GPIO ポート コンフィギュレーション レジスタ、AMUXBUS				固定機能デジタル ペリフェラル			
		OUT_EN = 1		OUT_EN = 0		OUT_EN = 1		OUT_EN = 0	
		OUT = 1	OUT = 0	OUT = 1	OUT = 0	OUT = 1	OUT = 0	OUT = 1	OUT = 0
ハイインピーダンス	0	High-Z	High-Z	High-Z	High-Z	High-Z	High-Z	High-Z	High-Z
予約済み	1	-	-	-	-	-	-	-	-
抵抗プルアップ	2	弱 1	強 0	High-Z	High-Z	強 1	強 0	弱 1	弱 1
抵抗プルダウン	3	強 1	弱 0	High-Z	High-Z	強 1	強 0	弱 0	弱 0
オープンドレイン、ロードライブ	4	High-Z	強 0	High-Z	High-Z	強 1	強 0	High-Z	High-Z
オープンドレイン、ハイドライブ	5	強 1	High-Z	High-Z	High-Z	強 1	強 0	High-Z	High-Z
ストロング	6	強 1	強 0	High-Z	High-Z	強 1	強 0	High-Z	High-Z
抵抗プルアップまたは抵抗プルダウン	7	弱 1	弱 0	High-Z	High-Z	強 1	強 0	弱 1	弱 0

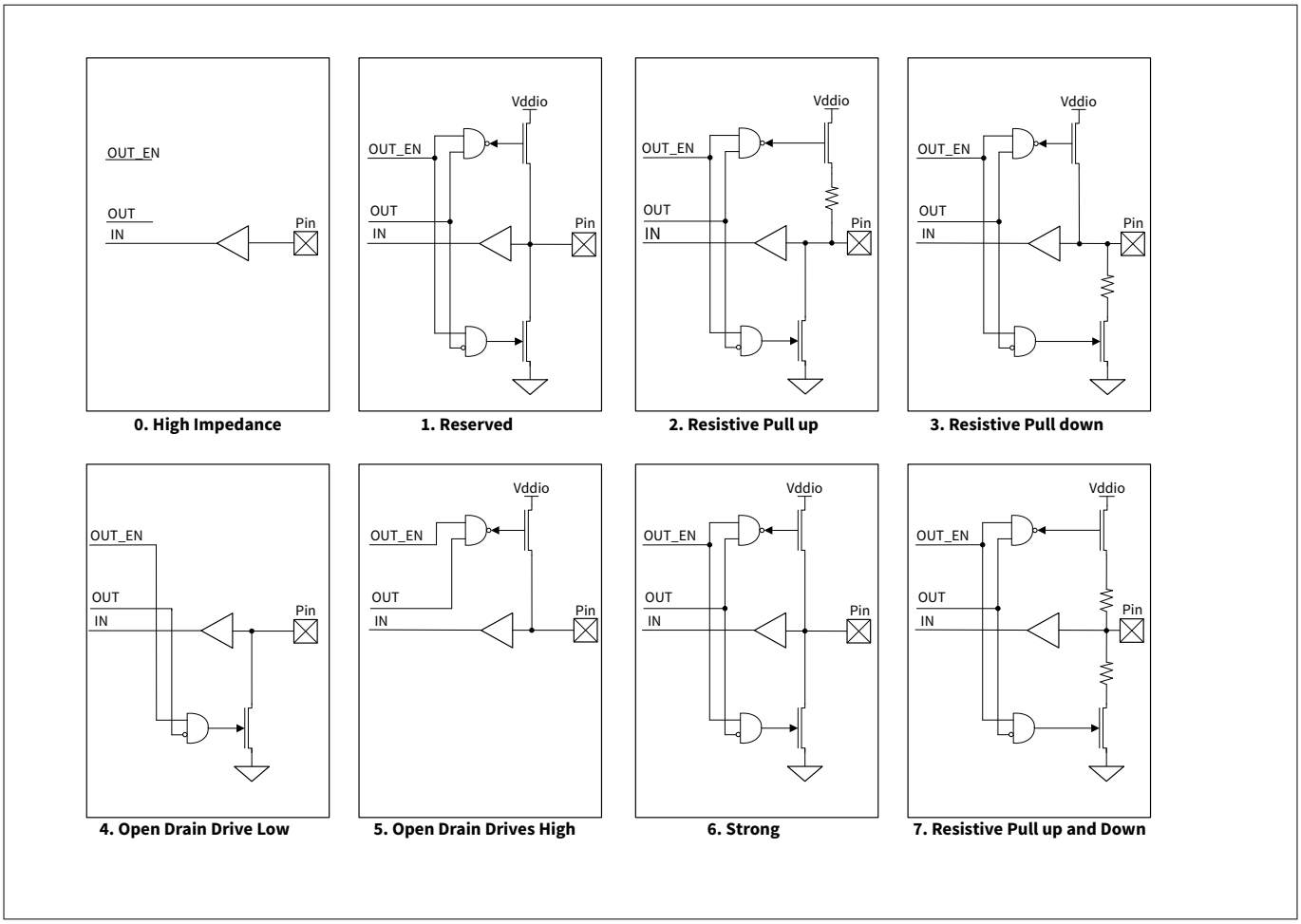


図 83　　GPIO ポート、ドライブモードブロック図

図 84 は、すべての利用可能なドライブモードにおける固定機能ベースのペリフェラルのピンビューを示す出力ドライバ図です。

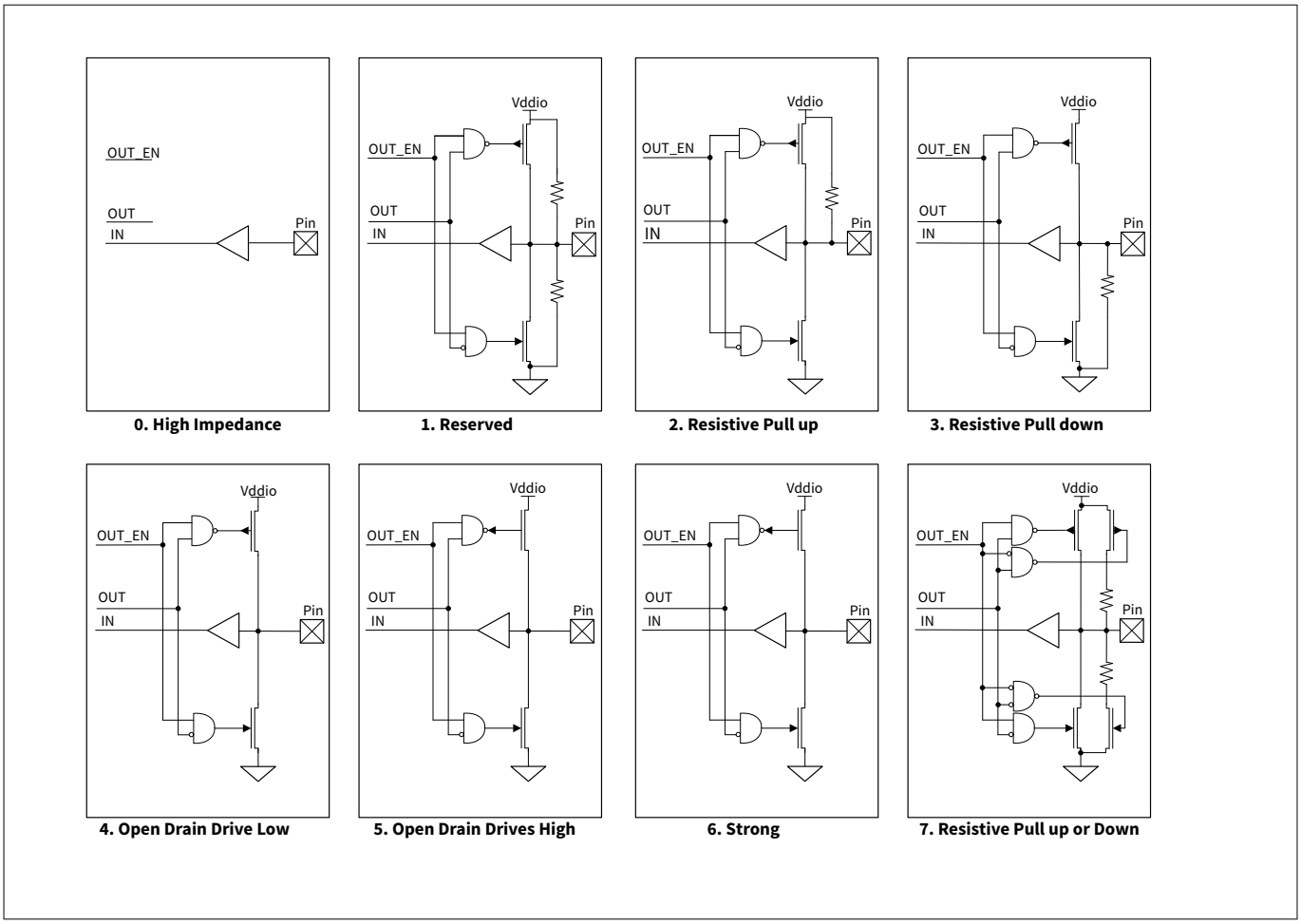


図 84　　固定機能ペリフェラル I/O ドライブ モード ブロック図

1. ハイインピーダンス (0)

   ハイインピーダンス モードは、アナログとデジタル入力に対し推奨される標準のハイインピーダンス (High-Z) 状態です。デジタル信号の場合、入力バッファが有効です。アナログ信号の場合、低消費電力設計では過電流とリーク電流を減らすため入力バッファは通常無効です。デバイス電流を最小にするためには、未使用の GPIO を入力バッファ無効に設定してハイ インピーダンスドライブ モードに設定する必要があります。入力バッファを無効にしたハイ インピーダンスドライブ モードは、デフォルトのピン リセット状態でもあります。

2. 抵抗プルアップモード (2) および抵抗プルダウンモード (3)

   抵抗モードは、データ状態の 1 つに直列抵抗を、他方のデータ状態に強ドライブが提供されます。ピンはこれらのモードにおいて、デジタル入力またはデジタル出力のいずれにも使用できます。抵抗プルアップをする場合は、そのピンのデータ レジスタ ビットに'1'を書き込みます。抵抗プルダウンをする場合は、そのピンのデータ レジスタに'0'を書き込みます。メカニカル スイッチへのインターフェースが、これらのドライブ モードの一般的な用途です。抵抗モードは、MCU をオープンドレイン駆動配線とインターフェースするためにも使用されます。入力がオープンドレイン LOW の場合は抵抗プルアップ、オープンドレイン HIGH の場合は抵抗プルダウンを使用します。

3. オープンドレイン駆動 LOW (4) およびオープンドレイン駆動 HIGH (5)

   オープンドレイン モードでは、データ状態の 1 つに高インピーダンスを、他方のデータ状態に強ドライブが提供されます。ピンはこれらのモードにおいて、デジタル入力またはデジタル出力として使用できます。したがってこのモードは双方向デジタル通信で広く使われています。信号が外部でプルダウンされた場合はオープンドレイン HIGH 駆動モード、外部でプルアップされた場合はオープンドレイン LOW 駆動モード

を使用します。オープンドレイン LOW 駆動モードの一般的な用途は、I2C バス信号ラインを駆動することです。

4. ストロング駆動 (6)

   ストロング駆動モードは、ピンの標準デジタル出力モードです。HIGH 状態と LOW 状態の両方で強い CMOS 出力駆動を提供します。ストロング駆動モードのピンは、通常の状況下で入力として使用しないでください。このモードは多くの場合、デジタル出力信号または外部デバイスを駆動するために使用されます。

5. 抵抗プルアップまたは抵抗プルダウン (7)

   抵抗性プルアップまたはプルダウン モードでは、GPIO はロジック 1 とロジック 0 の両方の出力状態で直列抵抗を持ちます。HIGH データ状態はプルアップになり、一方 LOW データ状態はプルダウンになります。このモードは、ピンが短絡を生じる可能性のある他の信号によって駆動される場合に使用します。

23.3.3 ファームウェアで制御される GPIO

レジスタを使用する標準ファームウェア制御 GPIO の場合は、HSIOM_PRTx_PORT_SEL レジスタで GPIO モードを選択する必要があります。

GPIO_PRTx_OUT レジスタは、GPIO の出力バッファ状態の読み出し/書き込みに使用します。このレジスタへの書き込み動作により、GPIO 出力ドライバ状態を書き込まれた値に変更します。読み出し動作は、このレジスタへ書き込まれた出力データと結果としての出力ドライバの状態に影響を与えます。GPIO ピンの現在のロジック レベルは返しません。GPIO_PRTx_OUT レジスタを使用することで、入力と出力の両方の GPIO を持つポートに対して、リード モディファイ ライトのシーケンスを安全に実行できます。

データ レジスタに加え、3 つのレジスタ (GPIO_PRTx_OUT_SET、GPIO_PRTx_OUT_CLR、GPIO_PRTx_OUT_INV) があり、他のピンに影響を与えることなく、ポート内の特定ピンの出力データをそれぞれセット、クリア、反転できます。これにより、ほとんどのユースケースにおいて、リード モディファイ ライトの操作が不要になります。これらのレジスタのビット フィールドに'1'を書き込むと、それぞれのピンがセット、クリア、または反転され、'0'を書き込んでもピンの状態に影響はありません。

GPIO_PRTx_IN はポート I/O パッド レジスタで、読み出し時に GPIO ピンにおける実際のロジック レベルを提供します。このレジスタへの書き込みは何の効果もありません。

23.3.4 割込み

すべてのポートピンは割込みを生成できます。図 85 に示すように、割込みを発生させるピン信号には 2 つのルーティングの可能性があります。

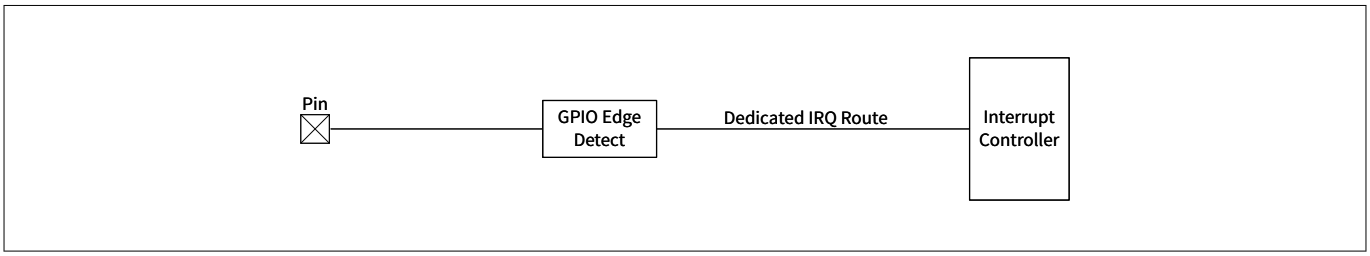


図 85 割込み信号の配線

CPU 割込みコントローラーに直接接続された「GPIO エッジ検出」ブロックを介したピン信号。割込みの発生は HSIOM のコンフィギュレーションに依存しないので、割込みを無効に設定 (GPIO_PRTx_INTR_CFG.EDGEy_SEL = 00) することを検討してください。図 86 に GPIO エッジ検出ブロックのブロック図を示します。

各 GPIO ピンの割込みは、GPIO_PRTx_INTR_CFG レジスタの選択により、GPIO ピンの立ち上りエッジまたは立ち下りエッジの変化で起動できます。I/O ポート内の最大 8 つの GPIO パッドは、GPIO ピン入力信号の立ち上りエッジと立ち下りエッジでそれぞれ'1'に設定される割込み発生フィールドを持っています (GPIO_PRTx_INTR レジスタ)。I/O ポートの GPIO_PRTx_INTR_MASK レジスタは、どの割込みがその割込み出力に伝搬されるかを指定します。こ

の伝搬された GPIO ピン割込みはディープスリープ機能割込みで、図 86 に示すように他の所定の GPIO ポート“i”ピン割込みと組み合わせてポート割込み interrupts_gpio[i]を形成した後、GPIO 入力信号の変化によりディープスリープ電力モードから CPU をウェイクアップできます。

さらに、レジスタ フィールドは、50 ns グリッチ フィルター (GPIO_PRTx_INTR_CFG.FLT_SEL レジスタ) にルーティングされる特定の I/O 入力信号を 1 つ指定します。グリッチ フィルター出力には、専用の検出回路があり、専用の検出制御フィールド (GPIO_PRTx_INTR.FLT_EDGE など) があります。各 I/O ポートには専用の割込みが関連付けられています (I/O ポート i に対しては interrupts_gpio[i])。図 85 に割込み機能を示します。図 86 に GPIO エッジ検出のブロックアーキテクチャを示します。

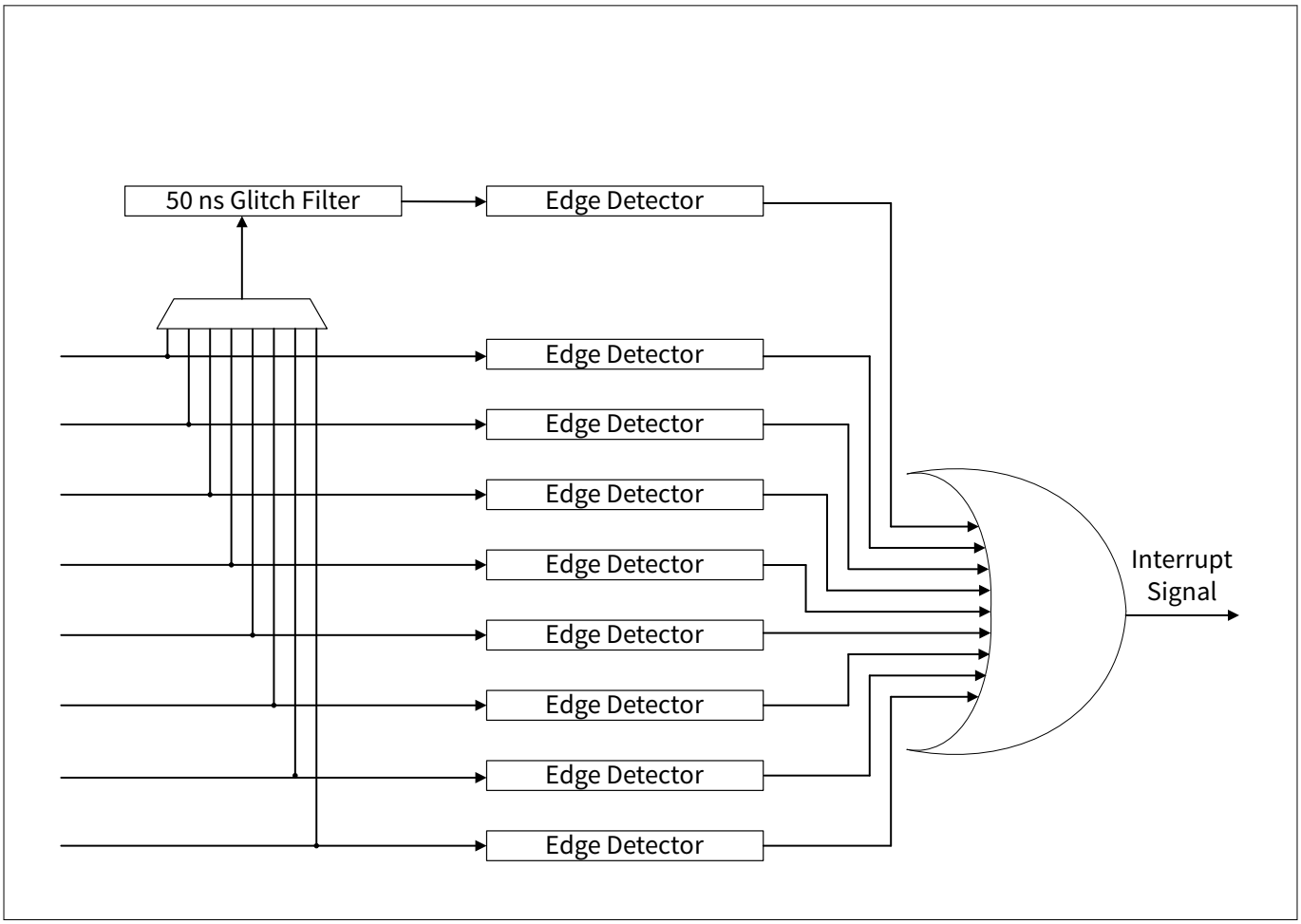


図 86　　GPIO エッジ検出のブロックアーキテクチャ

ソフトウェア ISR は、8+1 割込み要因フィールドを読み出して、割込み起動の要因となった I/O 信号またはグリッチ フィルター信号を特定できます。ISR は割込みを解除するために、割込み要因フィールドをクリアする必要があります。

各ピンにエッジ検出器があります。再設定なしで、立ち上りエッジ、立ち下りエッジ、両方のエッジを検出できます。エッジ検出器は、表 148 に示すように、GPIO_PRTx_INTR_CFG.EDGEy_SEL フィールドに書き込むことで設定します。

表 148　　エッジ検出器の設定

EDGE_SEL	設定
00	割込みが無効
11	両エッジで割込み

(続く)

表 148 (続き) エッジ検出器の設定

EDGE_SEL	設定
10	立ち下りエッジでの割込み
01	立ち上りエッジでの割込み

対応するステータス ビットに'1'を書き込むと、ピン エッジの状態がクリアされます。エッジ状態ステータス ビットをクリアすることが重要です。そうしないと、割込みは単一のトリガーに対して繰り返して発生するか、または複数のトリガーに対して 1 回のみ応答します。それらの関連事項は本章の後半で説明されます。対応するポートでエッジが発生していると同時にポート割込み制御ステータス レジスタが読み出されると、エッジが適切に検出されないことがあります。そのため GPIO 割込みを使用する場合、コードの他の部分ではなく対応する割込みサービス ルーチン内でのみステータス レジスタを読み出すように推奨します。

ファームウェアとデバッグ インターフェースは、GPIO_PRTx_INTR_SET レジスタの対応するビットを設定することで、任意のピンからハードウェア割込みをトリガーできます。

ピンに加えて、各ポートは、独自のエッジ検出器に接続されたグリッチフィルターを提供します。このフィルターは、ポートの 1 つのピンで駆動できます。駆動ピンの選択は GPIO_PRTx_INTR_CFG.FLT_SEL フィールドに 表 149 のように書き込みます。

表 149 グリッチフィルター入力部

FLT_SEL	選択されたピン
000	ピン 0 が選択
001	ピン 1 が選択
010	ピン 2 が選択
011	ピン 3 が選択
100	ピン 4 が選択
101	ピン 5 が選択
110	ピン 6 が選択
111	ピン 7 が選択

ポートピンのエッジが発生した場合、どのピンがエッジの原因となったかを知る必要があります。これは、ポート割込みステータス レジスタ (GPIO_PRTx_INTR) を読み出すことで行います。このレジスタには、どのピンがエッジを検出したかのラッチ情報と、現在のピンのステータスの両方が含まれます。これにより、CPU は 1 回の読み出し操作で両方の情報を読み出せます。このレジスタには、ラッチされたエッジの状態をクリアするという用途もあります。

ピンの対応するビット フィールドに'1'が書き込まれると、GPIO_PRTx_INTR_MASK レジスタは GPIO_PRTx_INTR エッジ検出信号を割込みコントローラーに送信します。続いて GPIO_PRTx_INTR_MASKED レジスタを読み出すことで、割り込みコントローラーへ転送された割り込み信号を生成した特定のピンを特定できます。その後、ポートのマスクされたエッジ検出器出力が OR され、割込みコントローラー (CPU サブシステムの NVIC) にルーティングされます。したがって、各ポートに割込みベクターは 1 つしかありません。

マスクされ OR されたエッジ検出ブロック出力は、割込みソース マルチプレクサにルーティングされ、レベル エッジ検出と立ち上りエッジ検出を選択できます。レベル オプションが選択されると、ポート割込みステータス レジスタ ビットが設定されている間、割込みが繰り返しトリガーされます。立ち上りエッジ検出オプションが選択された場合、ポート割込みステータス レジスタがクリアされなければ、割込みは 1 回だけトリガーされます。したがって、エッジ検出ブロックを使用する場合は、割込みステータス ビットをクリアすることが重要です。

割込み信号が固定機能ルートをとおしてルーティングされる場合、各ポートに専用の割込みベクタがあります。

すべてのポート割込みベクタは、使用可能な割込みベクタ数よりも多くのポートを持つデバイスで使用するために、1 つの割込みベクタに論理和されます。割込みをトリガーしたポートを特定するために、GPIO_INTR_CAUSEx レジスタを読み出せます。ビット位置に'1'がある場合は、対応するポートに保留中の割込みがあることを示します。その後、指示された GPIO_PRTx_INTR レジスタを読み出してピンを特定できます。

GPIO_VDD_ACTIVE レジスタは、外部電源の状態を読み出す機能を提供します。VDDIO 電源、VDDA、VDDD の有無を示します。さらに、電源の状態変化で割込みを発生させるよう、電源割込みを設定することもできます。GPIO_VDD_INTR_MASK レジスタは、CPU への割込み転送をマスクまたは有効にするために使用します。割込みの状態は GPIO_VDD_INTR レジスタで確認できます。

23.3.5 アナログ I/O

低インピーダンスの配線経路を必要とする ECO や WCO 用の LPCOMP や外部水晶ピンなどのアナログ リソースには固定ピンがあります。これらのピンは、AMUXBUS を使用して特定のアナログ ブロックに直接接続します。これらは性能を向上させ、これらのアナログ リソースを使用する際に他のピンよりも優先させます。これらの専用ピンの詳細についてはデバイス データシートを参照してください。GPIO を AMUXBUS に接続するアナログ ピンとして設定するためには、入力バッファを無効にして高インピーダンス アナログ モードに設定し、HSIOM_PORT_SELx レジスタを使用して正しい AMUXBUS にルーティングする必要があります。

23.3.6 セキュア GPIO

セキュア GPIO は IO ピンごとにセキュアなコンフィギュレーションを提供します。すべての IO ピンのデフォルトは非セキュアです。ピン コンフィギュレーションに関連するすべての AHB アクセスは、以下のようにセキュリティ属性チェックの対象です。

1. セキュアトランザクションは、セキュアとして設定されたピンの MMIO レジスタにのみアクセスできます。
2. 非セキュアトランザクションは、非セキュアとして設定されたピンの MMIO レジスタにのみアクセスできます。

上記の規則に違反すると、Read Zero Write Ignore (RZWI) 動作となります。したがって、ソフトウェアは、IO ピンにアクセスするために (HSIOM_PRTx.PORT_SEL0.IOy_SEL を介して) 選択された機能/ペリフェラルが、一致するセキュリティ プロファイルを持っていることを確認する必要があります。

全ポート共通の MMIO レジスタ、およびポートごとに編成されたレジスタは、このチェックから除外されます。これらの MMIO レジスタへのアクセスにセキュリティ チェックが必要な場合は、PPC で保護されたセキュアな領域に置かなければいけません。

23.4 プログラマブル デジタル (スマート I/O)

スマート I/O ブロックはプログラマブルな論理を I/O ポートに追加します。このプログラマブル ロジックは、AND、OR、XOR などのボードレベルのブール ロジック機能をポートに統合します。

スマート I/O ブロックは以下の特長を持っています。

- ボードレベルのブール ロジック機能をポートに統合
- GPIO ポート ピンからの HSIOM 入力信号を前処理する機能
- HSIOM 出力信号を GPIO ポート ピンに対して後処理する機能
- ハイバネートを除くすべてのデバイス電力モードで対応
- I/O パッドに密接に統合し、プログラマブルな最短の信号経路を実現

23.4.1 概要

スマート I/O ブロックは HSIOM と I/O ポート間の信号パスに配置されています。HSIOM は固定機能のペリフェラルおよび CPU から特定のポート ピンへと (逆の場合も同様) 出力信号を多重化します。この信号経路における

Smart I/O ブロックの配置により、図 87 に示すように、ポートピンと HSIOM の間の信号を処理するブリッジとして機能します。

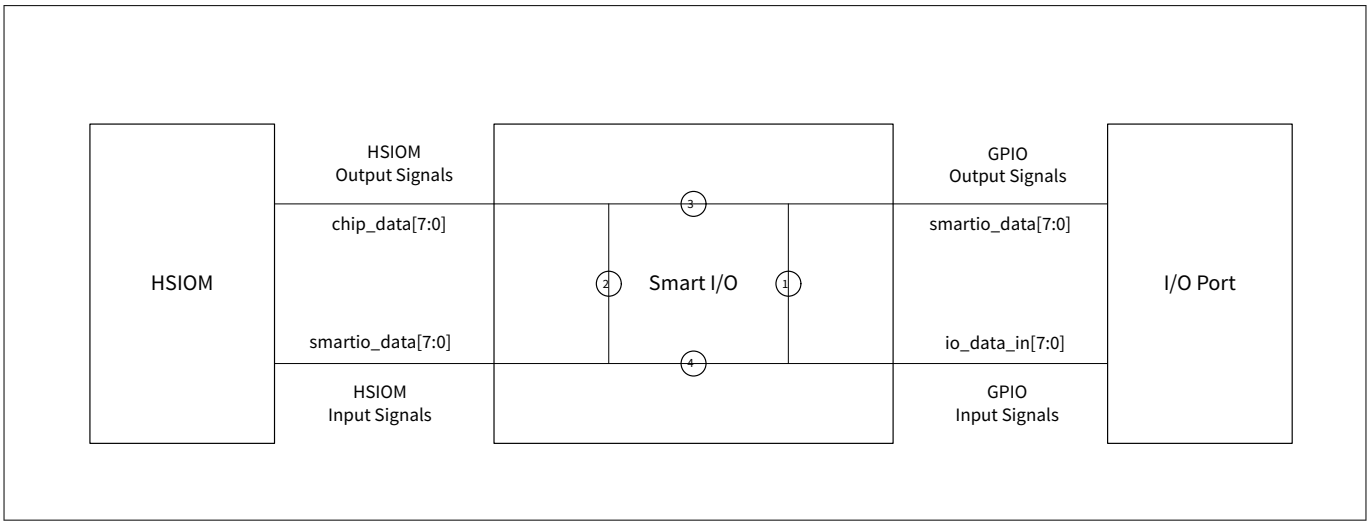


図 87　　スマート I/O インターフェース

図 87 に示すように、スマート I/O ブロックをとおしてサポートされる信号経路は以下のとおりです。

1. I/O ポート信号で直接動作する自己完結型の論理関数を搭載
2. HSIOM 信号で直接動作する自己完結型の論理関数を搭載
3. HSIOM 出力信号で動作し、その出力信号を変更したり、変更された信号を I/O ポートの信号に配線
4. I/O ポート出力信号で動作し、その出力信号を変更したり、変更された信号を HSIOM 入力信号に配線

次の節ではスマート I/O ブロックのコンポーネント、配線およびコンフィギュレーションを詳細に説明します。それらの節においては、GPIO 信号 (io_data) は I/O ポートからの入出力信号を、デバイスまたはチップ (chip_data) 信号は HSIOM からの入出力信号を指します。

23.4.2　　ブロック コンポーネント

スマート I/O の内部論理には以下のコンポーネントが含まれます。

- クロック (Clock) とリセット (Reset)
- 同期化 (Sync)
- 3 入力ルックアップテーブル (LUT)
- データユニット (DU)

図 88 にスマート I/O の内部ルーティングを示します。

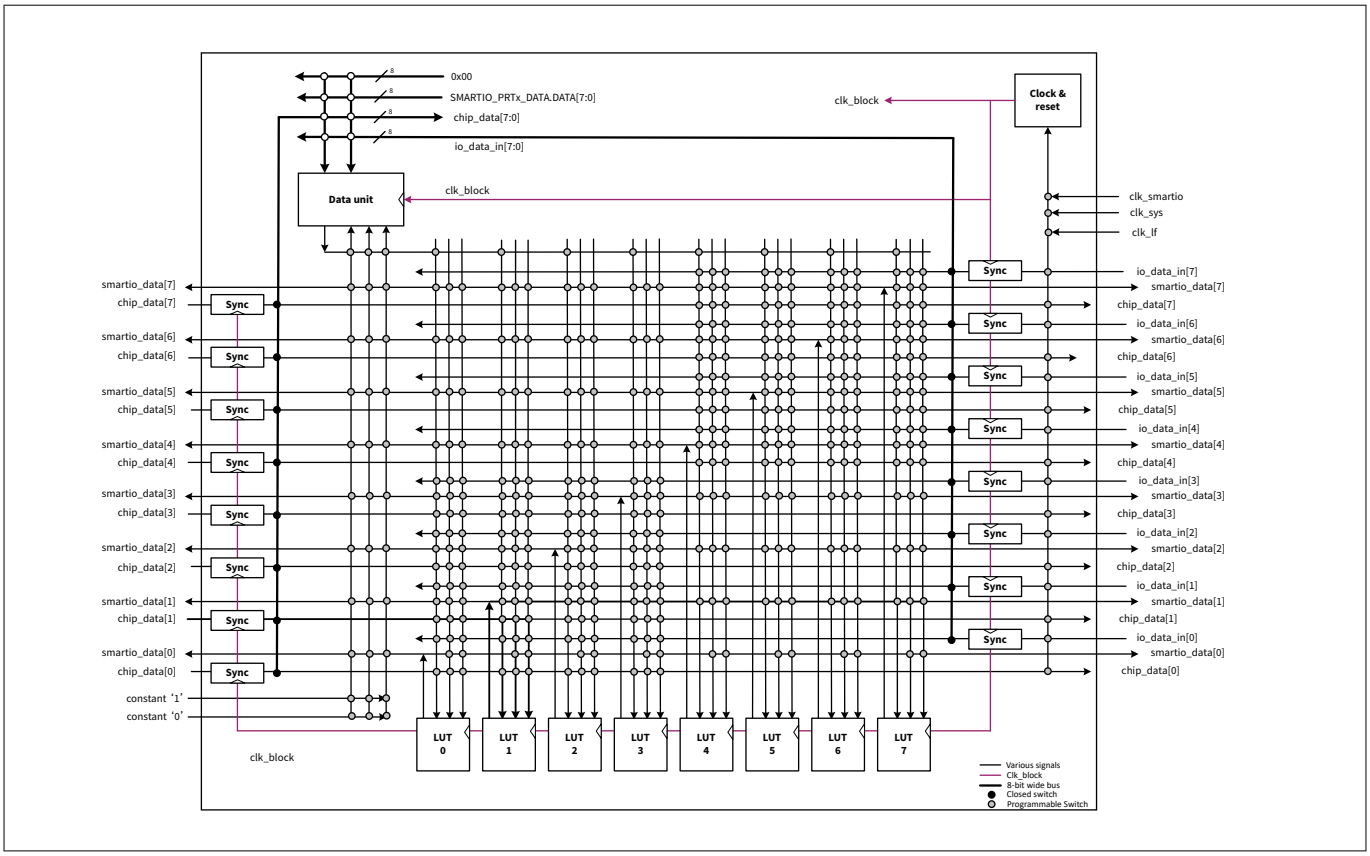


図 88　　スマート I/O 内部ルーティング

23.4.2.1　クロック (Clock) とリセット (Reset)

クロックとリセット コンポーネントはスマート I/O ブロックのクロック (clk_block) およびリセット信号 (rst_block_n) を選択します。単一のクロックとリセット信号が、ブロック内の全コンポーネントに使用されます。クロックとリセットのソースは、SMARTIO_PRTx_CTL レジスタの CLOCK_SRC[4:0]ビットフィールドで決定されます。選択されたクロックは、I/O 入力シンクロナイザー、LUT、およびデータ ユニット コンポーネントを含むブロック コンポーネント内の同期ロジックに使用されます。選択されたリセットは、LUT およびデータ ユニット コンポーネントの同期ロジックを非同期にリセットするために使用されます。

***注**:　　ブロックの同期ロジック用に選択されたクロック(clk_block) は、同じクロックで動作するデバイス内の他の同期ロジックと位相が整合していません。そのため、スマート I/O と他の同期ロジックとの通信は非同期として扱う必要があります。*

以下に選択できるクロックソースを示します。

- GPIO 入力信号「io_data_in[7:0]」。これらのクロック ソースには、関連するリセットはありません。
- HSIOM 出力信号"chip_data[7:0]"。これらのクロック ソースには、関連するリセットはありません。
- Smart I/O クロック (clk_smartio)。周辺クロック分周器を用いてシステムクロック (clk_sys) から生成されます。ペリフェラルクロック分周の詳細については、クロックセクションを参照してください。このクロックはアクティブ電力モードでのみ動作できます。クロックは 2 つの関連するリセット (rst_sys_act_n と rst_sys_dpslp_n) のうち 1 つを持つことができます。

  これらのリセットはブロック同期状態がどの電力モードでリセットされるかを判定します。例えば、rst_sys_act_n リセットはアクティブ電力モードにあるスマート I/O の同期機能向けです。
- 低周波システムクロック (clk_lf)。このブロックはディープスリープ電力モードで動作できます。このクロックには、関連するリセット rst_lf_dpslp_n があります。リセットは、デバイスがハイバネート モードに入るか、POR 状態になると作動します。

ブロックが有効な場合、選択されたクロック (clk_block) および関連リセット (rst_block_n) はファブリック コンポーネントに提供されます。ファブリックが無効になると、ファブリック コンポーネントにクロックが供給されなくなり、リセットが有効になります (LUT およびデータ ユニット コンポーネントはリセット値'0'に設定されます)。

I/O 入力シンクロナイザーは、2 clk_block サイクルの遅延をもたらします (シンクロナイザーが有効の場合)。その結果、最初の 2 サイクルで、ブロックはシンクロナイザー出力からの古いデータにさらされる可能性があります。したがって、最初の 2 クロック サイクルの間、リセットはアクティブになり、ブロックはバイパス モードになります。

表 150　　クロックとリセットのレジスタ制御

レジスタ[BIT_POS]	ビット名	説明
SMARTIO_PRTn_CTL[12:8]	CLOCK_SRC[4:0]	クロック (clk_fabric)/リセット (rst_fabric_n) ソースの選択 0: io_data_in[0]/1 ... 7: io_data_in[7]/1 8: chip_data[0]/1 ... 15: chip_data[7]/1 16: : clk_smartio/ rst_sys_act_n。"clk_smartio"上のアクティブ機能同期ロジックに使用されます。この選択は、PCLK 指定のクロック周波数 ("clock_smartio_pos_en") での同期動作を意図しています。 17: clk_smartio/ rst_sys_dpslp_n。"clk_smartio"上のディープスリープ機能同期ロジックに使用されます ("clk_smartio"はディープスリープおよびハイバネート電力モードでは使用できないことに注意してください)。この選択は PCLK 指定クロック周波数での同期動作を意図しています。 18: 17 と同じ 19: clk_lf/rst_lf_dpslp_n。この選択は"clk_lf"での同期動作を意図しています。 20-30: クロックソースは定数'0'です。低消費電力を確保するため、スマート I/O が無効の場合は、これらのクロックソースのいずれかを選択する必要があります。 31: clk_sys/1。この選択は非同期動作を意図しています。スマート I/O は、選択されたクロックの 2 クロック エッジ後に完全に機能します (リセットが解除されます)。

23.4.2.2 同期化

各 GPIO 入力信号とデバイス入力信号 (HSIOM 入力) は、非同期または同期で使用できます。信号を同期して使用するために、図 89 に示すようなダブル フリップフロップ シンクロナイザーがこれらの信号経路の両方に配置され、信号をスマート I/O クロック (clk_block) に同期させます。SMARTIO_PRTx_SYNC_CTL レジスタの GPIO 入力信号の IO_SYNC_EN[i]、および HSIOM 信号の CHIP_SYNC_EN[i] を設定またはクリアすることで、各ピン/入力の同期を有効または無効にします。ここでは'i'はピン番号を示します。

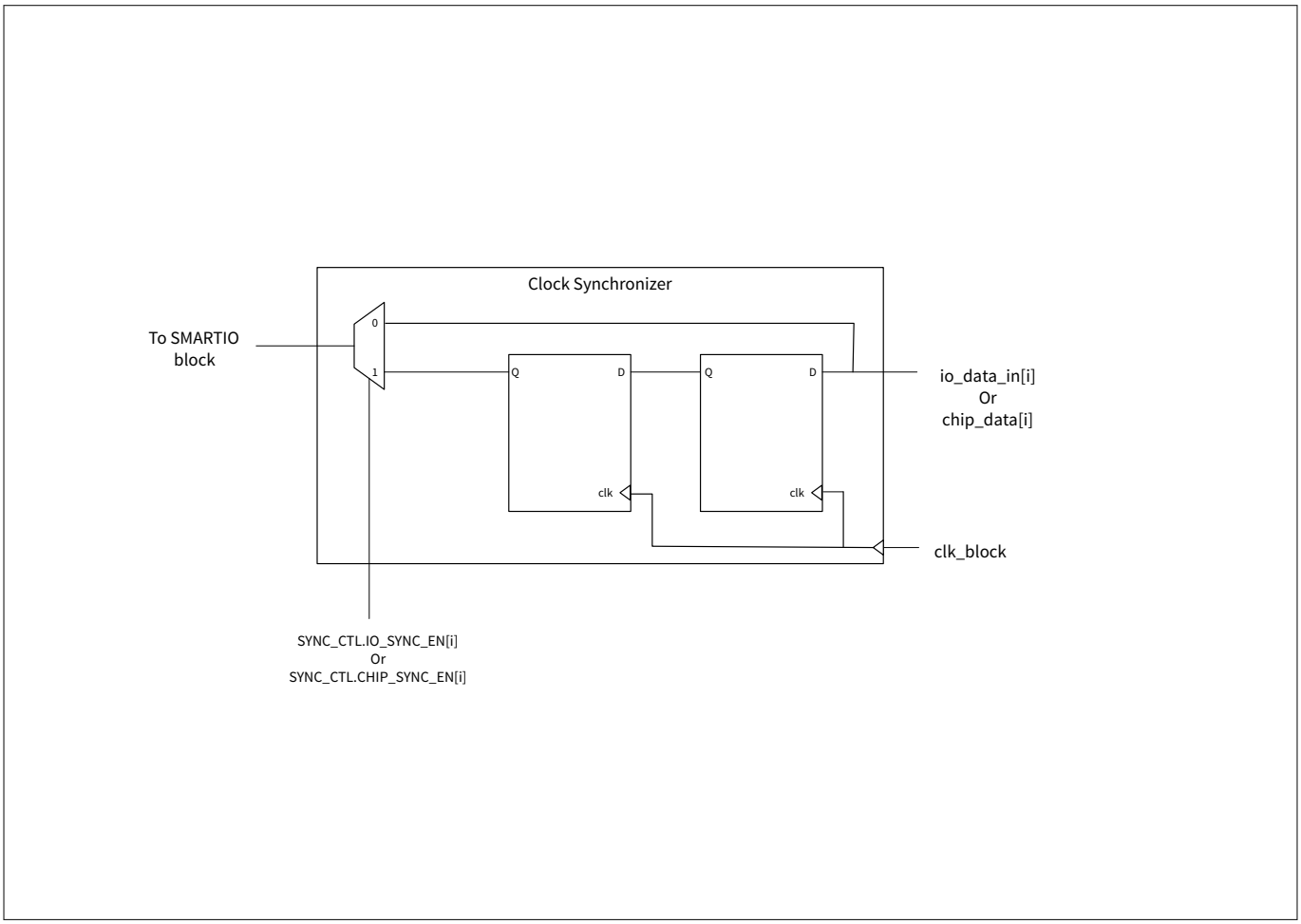


図 89 スマート I/O クロック同期

23.4.2.3 ルックアップ テーブル (LUT)

1 つのスマート I/O ブロックには 8 つのルックアップ テーブル (LUT) コンポーネントが含まれます。LUT コンポーネントは、3 入力 LUT とフリップフロップで構成されます。各 LUT ブロックには 3 つの入力信号があり、SMARTIO_PRTx_LUT_CTLy レジスタ (y は LUT 番号を示します) に設定された構成に基づいて出力を生成します。各 LUT の設定は、SMARTIO_PRTx_LUT_CTLy レジスタの 8 ビットルックアップベクター LUT[7:0] および 2 ビットオペコード OPC[1:0] によって決定されます。8 ビットのベクトルは、3 つの入力信号のルックアップテーブルとして使用されます。2 ビットのオペコードは、フリップフロップの使用法を決定します。図 90 に、異なるオペコードに対する LUT コンフィギュレーションを示します。

SMARTIO_PRTx_LUT_SELy レジスタは、各 LUT に入力される 3 つの入力信号 (tr0_in、tr1_in、tr2_in) を選択します。入力ソースは以下のとおりです。

- データユニット出力
- その他の LUT 出力信号 (tr_out)

- HSIOM 出力信号 (chip_data[7:0])
- GPIO 入力信号 (io_data[7:0])

SMARTIO_PRTx_LUT_SELy レジスタの LUT_TR0_SEL[3:0]ビットは、y 番目の LUT の tr0_in 信号を選択します。同様に、LUT_TR1_SEL[3:0]ビットおよび LUT_TR2_SEL[3:0]ビットはそれぞれ tr1_in 信号と r2_in 信号を選択します。詳細については、表 151 を参照してください。

表 151 LUT レジスタ制御

レジスタ[BIT_POS]	ビット名	説明
SMARTIO_PRTx_LUT_CTLy[7:0]	LUT[7:0]	LUT の設定。LUT オペコード (LUT_OPC), 内部状態および LUT 入力信号 tr0_in, tr1_in, tr2_in に応じて、LUT 設定は LUT 出力信号と次の順序状態の決定に使用します。
SMARTIO_PRTx_LUT_CTLy[9:8]	LUT_OPC[1:0]	LUT のオペコードは、図 90 に示すように LUT の動作を指定します。

(続く)

表 151 (続き) LUT レジスタ制御

レジスタ[BIT_POS]	ビット名	説明
SMARTIO_PRTx_LUT_SELy[3:0]	LUT_TR0_SEL[3:0]	LUT 入力信号“tr0_in”のソース選択 0: データユニット出力 1: LUT 1 出力 2: LUT 2 出力 3: LUT 3 出力 4: LUT 4 出力 5: LUT 5 出力 6: LUT 6 出力 7: LUT 7 出力 8: chip_data [0] (LUT 0, 1, 2, 3 の場合); chip_data[4] (LUT 4, 5, 6, 7 の場合) 9: chip_data [1] (LUT 0, 1, 2, 3 の場合); chip_data[5] (LUT 4, 5, 6, 7 の場合) 10: chip_data [2] (LUT 0, 1, 2, 3 の場合); chip_data[6] (LUT 4, 5, 6, 7 の場合) 11: chip_data [3] (LUT 0, 1, 2, 3 の場合); chip_data[7] (LUT 4, 5, 6, 7 の場合) 12: io_data_in[0] (LUT 0, 1, 2, 3 の場合); io_data[4] (LUT 4, 5, 6, 7 の場合) 13: io_data_in[1] (LUT 0, 1, 2, 3 の場合); io_data[5] (LUT 4, 5, 6, 7 の場合) 14: io_data_in[2] (LUT 0, 1, 2, 3 の場合); io_data[6] (LUT 4, 5, 6, 7 の場合) 15: io_data_in[3] (LUT 0, 1, 2, 3 の場合); io_data[7] (LUT 4, 5, 6, 7 の場合)

(続く)

表 151 (続き) LUT レジスタ制御

レジスタ[BIT_POS]	ビット名	説明
SMARTIO_PRTx_LUT_SELy[11:8]	LUT_TR1_SEL[3:0]	LUT 入力信号“tr1_in”のソース選択 0: LUT 0 出力 1: LUT 1 出力 2: LUT 2 出力 3: LUT 3 出力 4: LUT 4 出力 5: LUT 5 出力 6: LUT 6 出力 7: LUT 7 出力 8: chip_data [0] (LUT 0, 1, 2, 3 の場合); chip_data[4] (LUT 4, 5, 6, 7 の場合) 9: chip_data [1] (LUT 0, 1, 2, 3 の場合); chip_data[5] (LUT 4, 5, 6, 7 の場合) 10: chip_data [2] (LUT 0, 1, 2, 3 の場合); chip_data[6] (LUT 4, 5, 6, 7 の場合) 11: chip_data [3] (LUT 0, 1, 2, 3 の場合); chip_data[7] (LUT 4, 6, 6, 7 の場合) 12: io_data_in[0] (LUT 0, 1, 2, 3 の場合); io_data[4] (LUT 4, 5, 6, 7 の場合) 13: io_data_in[1] (LUT 0, 1, 2, 3 の場合); io_data[5] (LUT 4, 5, 6, 7 の場合) 14: io_data_in[2] (LUT 0, 1, 2, 3 の場合); io_data[6] (LUT 4, 5, 6, 7 の場合) 15: io_data_in[3] (LUT 0, 1, 2, 3 の場合); io_data[7] (LUT 4, 5, 6, 7 の場合)
SMARTIO_PRTx_LUT_SELy[19:16]	LUT_TR2_SEL[3:0]	LUT 入力信号“tr2_in”のソース選択 エンコーディングは LUT_TR1_SEL と同じです。

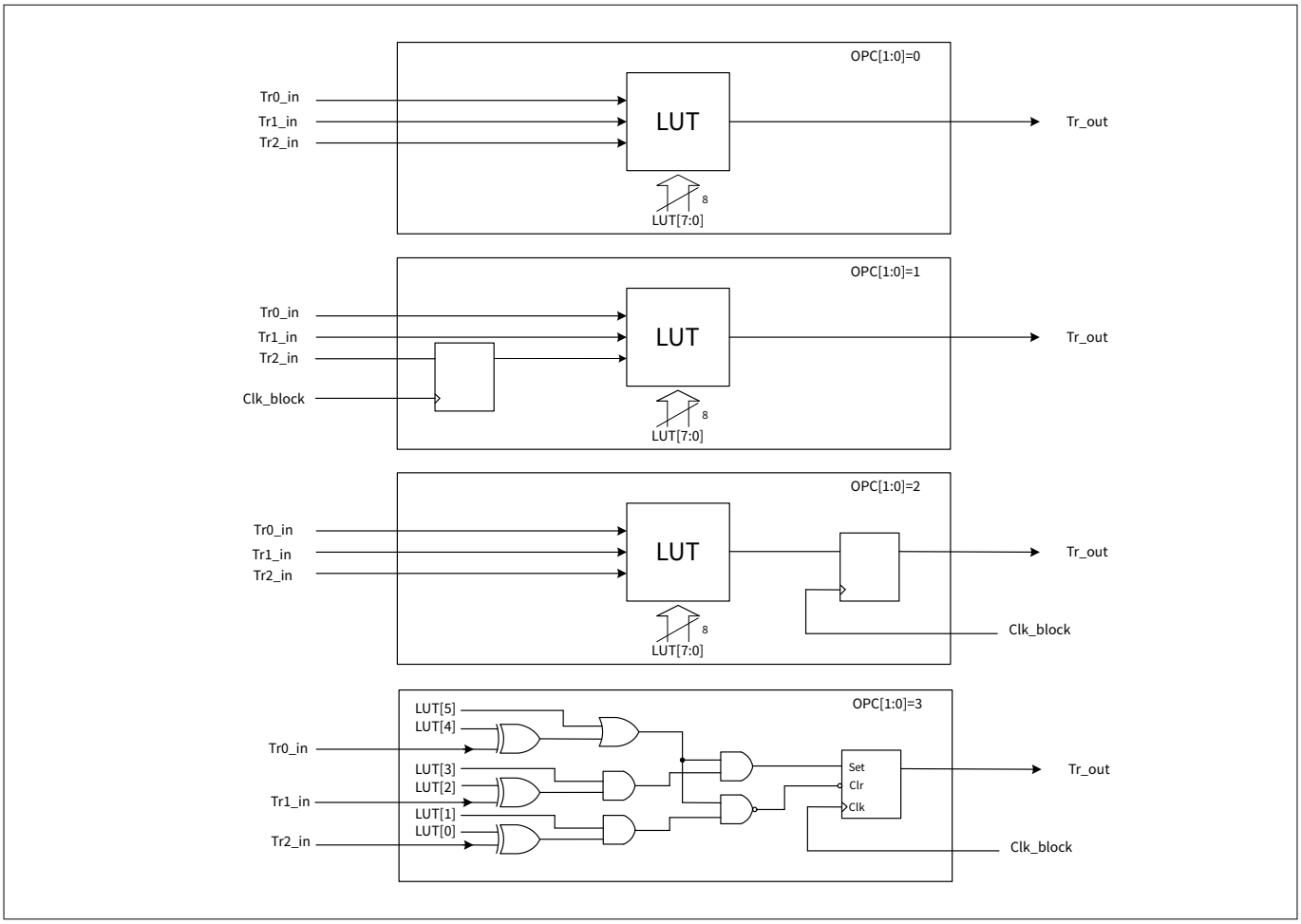


図 90　　スマート I/O LUT の設定

23.4.2.4　データユニット (DU)

各スマート I/O ブロックにはデータ ユニット (DU) コンポーネントが含まれます。DU はシンプルな 8 ビットのデータパスで構成されています。簡単なインクリメント, デクリメント, インクリメント/デクリメント, シフト, および AND/OR の動作を実行できます。DU が実行する動作は、SMARTIO_PRTx_DU_CTL レジスタの 4 ビットのオペコード DU_OPC[3:0]を用いて選択されます。

DU コンポーネントは、LUT コンポーネントと同様に、最大 3 つの入力トリガー信号 (tr0_in、tr1_in、tr2_in) をサポートします。これらの信号は、DU オペコードで定義された動作を開始するために使用されます。さらに、DU には 2 つの 8 ビット データ入力 (data0_in[7:0]および data1_in[7:0]) があり、8 ビットの内部状態 (data[7:0]) の初期化や参照に使用されます。8 ビットのデータ入力ソースは次のように設定されています。

- 定数 0x00
- io_data_in[7:0]
- chip_data_in[7:0]
- SMARTIO_PRTx_DATA レジスタの DATA [7:0]ビット フィールド

トリガー信号は SMARTIO_PRTx_DU_SEL レジスタの DU_TRx_SEL[3:0]ビットフィールドで選択します。SMARTIO_PRTx_DU_SEL レジスタの DUT_DATAx_SEL[1:0]ビットは 8 ビット入力データソースを選択します。DU のサイズ (データパスで使用されるビット数) は、SMARTIO_PRTx_DU_CTL レジスタの DU_SIZE[2:0]ビットで定義されます。レジスタ制御の詳細は、図 91 のブロック図を参照してください。

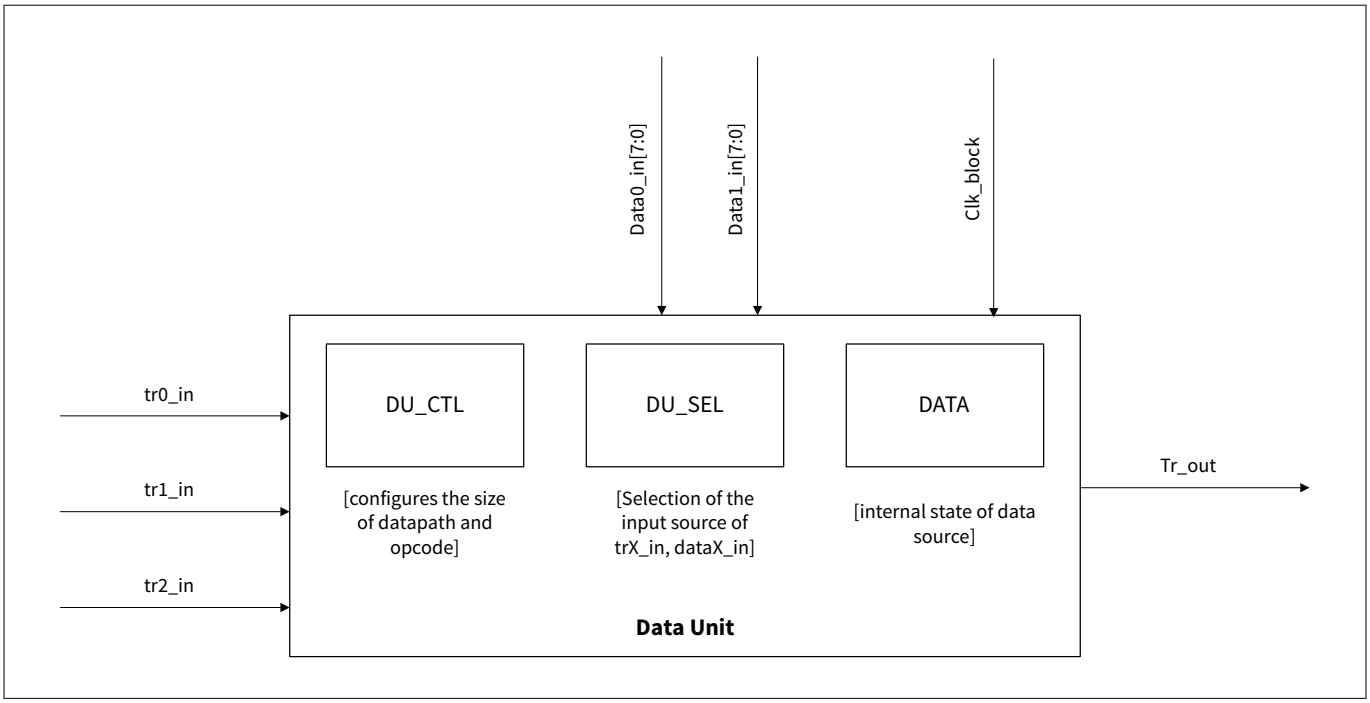


図 91　　データユニット (DU)

レジスタ制御の詳細については表 152 を参照してください。

表 152　　データ ユニット レジスタ制御

レジスタ[BIT_POS]	ビット名	説明
SMARTIO_PRTx_DU_CTL[2:0]	DU_SIZE[2:0]	データユニットのサイズ/幅 (ビット単位) は DU_SIZE+1 です。例えば、DU_SIZE が 7 の場合、幅は 8 ビットです。
SMARTIO_PRTx_DU_CTL[11:8]	DU_OPC[3:0]	データユニットオペコードはデータユニット動作を指定します。 1: INCR 2: DECR 3: INCR_WRAP 4: DECR_WRAP 5: INCR_DECR 6: INCR_DECR_WRAP 7: ROR 8: SHR 9: AND_OR 10: SHR_MAJ3 11: SHR_EQL それ以外: 未定義。

(続く)

表 152 (続き) データ ユニット レジスタ制御

レジスタ[BIT_POS]	ビット名	説明
SMARTIO_PRTx_DU_SEL[3:0]	DU_TR0_SEL[3:0]	データユニット入力データ“tr0_in”のソース選択 0: 定数 '0' 1: 定数 '1' データユニット出力。 10–3: LUT 7-0 出力。 それ以外: 未定義。
SMARTIO_PRTx_DU_SEL[11:8]	DU_TR1_SEL[3:0]	データユニット入力データ“tr1_in”のソース選択エンコーディングは DU_TR0_SEL と同じ。
SMARTIO_PRTx_DU_SEL[19:16]	DU_TR2_SEL[3:0]	データユニット入力データ“tr2_in”のソース選択エンコーディングは DU_TR0_SEL と同じ。
SMARTIO_PRTx_DU_SEL[25:24]	DU_DATA0_SEL[1:0]	データユニット入力データ“data0_in”のソース選択 0: 0x00 1: chip_data[7:0] 2: io_data[7:0] 3: SMARTIO_PRTx_DATA.DATA [7:0] レジスタフィールド
SMARTIO_PRTx_DU_SEL[29:28]	DU_DATA1_SEL[1:0]	データユニット入力データ“data1_in”のソース選択エンコーディングは DU_DATA0_SEL と同じ。
SMARTIO_PRTx_DATA[7:0]	DATA[7:0]	データユニット入力データソース。

DU は、単一の出力トリガー信号 (tr_out) を生成します。内部状態 (du_data[7:0]) はフリップフロップでキャプチャされ、clk_block を必要とします。

以下の擬似コードは、DU オペコードがサポートする様々なデータパス操作について説明しています。"Comb"は、組み合わせ機能、つまり、以前の出力状態とは無関係に動作する機能を表していることに注意

してください。"Reg"は、登録された機能、つまり、入力と以前の出力状態 (フリップフロップを使用して登録された) に対して動作する機能を表します。

```
// The following is shared by all operations.
mask = (2 ^ (DU_SIZE + 1)- 1)
data_eql_data1_in = (data & mask) == (data1_in & mask));
data_eql_0 = (data & mask) == 0);
data_incr = (data + 1) & mask;
data_decr = (data - 1) & mask;
data0_masked = data_in0 & mask;
// INCR operation: increments data by 1 from an initial value (data0) until it reaches a
// final value (data1).
Comb: tr_out = data_eql_data1_in;
Reg: data <= data;
if (tr0_in) data <= data0_masked; //tr0_in is reload signal - loads masked data0
// into data
else if (tr1_in) data <= data_eql_data1_in ? data : data_incr; //increment data until
// it equals data1
// INCR_WRAP operation: operates similar to INCR but instead of stopping at data1, it
// wraps around to data0.
Comb: tr_out = data_eql_data1_in;
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) data <= data_eql_data1_in ? data0_masked : data_incr;
// DECR operation: decrements data from an initial value (data0) until it reaches 0.
Comb: tr_out = data_eql_0;
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) data <= data_eql_0 ? data : data_decr;
// DECR_WRAP operation: works similar to DECR. Instead of stopping at 0, it wraps around
// to data0.
Comb: tr_out = data_eql_0;
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) data <= data_eql_0 ? data0_masked : data_decr;
// INCR_DECR operation: combination of INCR and DECR. Depending on trigger signals
// it either starts incrementing or decrementing. Increment stops at data1 and decrement
// stops at 0.
Comb: tr_out = data_eql_data1_in | data_eql_0;
Reg: data <= data;
if (tr0_in) data <= data0_masked; // Increment operation takes precedence over
// decrement when both signal are available
else if (tr1_in) data <= data_eql_data1_in ? data : data_incr;
else if (tr2_in) data <= data_eql_0 ? data : data_decr;
// INCR_DECR_WRAP operation: same functionality as INCR_DECR with wrap around to data0 on
// reaching the limits.
Comb: tr_out = data_eql_data1_in | data_eql_0;
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) data <= data_eql_data1_in ? data0_masked : data_incr;
else if (tr2_in) data <= data_eql_0 ? data0_masked : data_decr;
// ROR operation: rotates data right and LSb is sent out. The data for rotation is taken
// from data0.
```

```
Comb: tr_out = data[0];
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) {
data <= {
0,
data[7: 1]
} & mask; //Shift right operation
data[du_size] <= data[0]; //Move the data[0] (LSb) to MSb
}
// SHR operation: performs shift register operation. Initial data (data0) is shifted out
// and data on tr2_in is shifted in.
Comb: tr_out = data[0];
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) {
data <= {
0,
data[7: 1]
} & mask; //Shift right operation
data[du_size] <= tr2_in; //tr2_in Shift in operation
}
// SHR_MAJ3 operation: performs the same functionality as SHR. Instead of sending out
// the shifted out value, it sends out a '1' if in the last three samples/shifted-out
// values (data[0]), the signal high in at least two samples. otherwise, sends a '0'.
// This function sends out the majority of the last three samples.
Comb: tr_out = (data == 0x03) |
(data == 0x05) |
(data == 0x06) |
(data == 0x07);
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) {
data <= {
0,
data[7: 1]
} & mask;
data[du_size] <= tr2_in;
}
// SHR_EQL operation: performs the same operation as SHR. Instead of shift-out, the
// output is a comparison result (data0 == data1).
Comb: tr_out = data_eql_data1_in;
Reg: data <= data;
if (tr0_in) data <= data0_masked;
else if (tr1_in) {
data <= {
0,
data[7: 1]
} & mask;
data[du_size] <= tr2_in;
}
// AND_OR operation: ANDs data1 and data0 along with mask; then, ORs all the bits of
// the ANDed output.
```

```
Comb: tr_out = | (data & data1_in & mask);
Reg: data <= data;
if (tr0_in) data <= data0_masked;
```

23.4.3 ルーティング

スマート I/O ブロックは、ブロック内外の信号をルーティングしたり、ブロック内の様々なコンポーネント間にルーティングしたりするために使用される数多くのスイッチを含みます。ルーティングスイッチは PRTGIO_PRTx_LUT_SELy と SMARTIO_PRTx_DU_SEL レジスタで処理されます。詳細については、レジスタ リファレンス マニュアルを参照してください。図 88 にスマート I/O の内部ルーティングを示します。この図では、LUT7～LUT4 は io_data/chip_data[7]～io_data/chip_data[4]で動作することに対して、LUT3～LUT0 は io_data/chip_data[3]～io_data/chip_data[0]で動作することに注意してください。

23.4.4 動作

スマート I/O ブロックは、以下のように設定および操作してください。

1. ブロックを有効に設定する前に、すべてのコンポーネントとルーティングをブロック コンポーネントで説明したように設定する必要があります。
2. コンポーネントとルーティングのコンフィギュレーションに加え、いくつかのブロックレベルの設定は、希望する動作のために正しくコンフィギュレーションされなければいけません。
   a. バイパス制御: スマート I/O 経路は、SMARTIO_PRTx_CTL レジスタの BYPASS[i]ビット フィールドを設定することにより特定の GPIO 信号に対してバイパスされます。BYPASS[7:0]ビットフィールドに'i'ビットが設定されると、i 番目の GPIO 信号は HSIOM 信号経路に直接バイパスされます。これは、スマート I/O 機能が特定の I/O にのみ必要な場合に便利です。
   b. パイプライントリガー モード: LUT 入力マルチプレクサおよび LUT コンポーネント自体には、組み合わせループは含まれません。同様に、データユニットにも組み合わせループは含まれません。ただし 1 つの LUT がほかの LUT またはデータユニットと相互作用するとき、意図しない組合せループが発生する可能性があります。この制限を克服するために、SMARTIO_PRTx_CTL レジスタの PIPELINE_EN ビットフィールドが使用されます。このビットを設定すると、ほかのコンポーネントが分岐する前にすべての超えた出力 (LUT および DU) の値が決定されます。
3. スマート I/O ブロックは希望の機能に応じてコンフィギュレーションされた後、SMARTIO_PRTx_CTL ジスタの ENABLED ビット フィールドを設定することで有効化されます。ENABLED ビット フィールドがクリアされると、スマート I/O ブロックはバイパス モードに入ります。そのモードにおいては、GPIO 信号は直接 HSIOM 信号により制御されます (逆の場合も同様)。スマート I/O ブロックをコンフィギュレーションする必要があります。すなわち、レジスタの更新中のグリッチを防止するために、ブロックを有効化する前にすべてのレジスタ設定を更新する必要があります。

表 153 スマート I/O ブロック制御

レジスタ [BIT_POS]	ビット名	説明
SMARTIO_PRTx_CTL[25]	PIPELINE_EN	パイプラインレジスタの有効化 0: 無効 (レジスタがバイパスされます) 1: 有効

(続く)

表 153 (続き) スマート I/O ブロック制御

レジスタ [BIT_POS]	ビット名	説明
SMARTIO_PRTx_CTL[31]	ENABLED	Smart I/O を有効にする。スマート I/O が完全に設定されている場合のみ'1'に設定してください。 0: 無効 (信号はバイパスされます。BYPASS[7:0]が 0xFF の場合の動作)。無効に設定すると、ブロック (データユニットと LUT) のリセットが有効になります。 ブロックが無効の場合。 - PIPELINE_EN レジスタフィールドは、低消費電力を保証するために'1'に設定してください。 - CLOCK_SRC レジスタ フィールドは、低消費電力を確保するために 20～30 (クロックは定数'0') に設定する必要があります。 1: 有効。有効に設定すると、ブロック リセットが解除され、ブロックが完全に機能するようになるまで、clk_block クロック サイクルを 3 回要します。この動作は、ブロックが完全に機能するときに I/O ピンの入力シンクロナイザーの状態が確実にフラッシュされるようにします。
SMARTIO_PRTx_CTL[7:0]	BYPASS[7:0]	スマート I/O のバイパス、各 I/O ピンに対して 1 ビット: BYPASS[i] は I/O ピン i 用です。ENABLED が'1'の時、このフィールドが使用されます。ENABLED が'0'の場合、このフィールドは使用されず、スマート I/O は常にバイパスされます。 0: バイパスなし (信号経路にスマート I/O があります) 1: バイパス (信号経路にスマート I/O がありません)

23.5 レジスタの概要

表 154 レジスタの概要

レジスタ	名称	説明
GPIO_PRTx_OUT	ポート出力レジスタ	ポートの I/O ピンの出力ドライバ データを読み出し/書き込みします。

(続く)

表 154 (続き) レジスタの概要

レジスタ	名称	説明
GPIO_PRTx_OUT_CLR	ポート出力クリア レジスタ	ポート内の特定の I/O ピンの出力データをクリアします。
GPIO_PRTx_OUT_SET	ポート出力セット レジスタ	ポートの特定の I/O ピンの出力データを設定します。
GPIO_PRTx_OUT_INV	ポート出力反転レジスタ	ポートの特定の I/O ピンの出力データを反転させます。
GPIO_PRTx_IN	ポート入力レジスタ	I/O ピン入力の現在のピン状態を読み出します。
GPIO_PRTx_INTR	ポート割込みステータス レジスタ	現在のピンの割込み状態を読み出します。
GPIO_PRTx_INTR_MASK	ポート割込みマスク レジスタ	ピン割込みを CPU の割込みコントローラーに転送するマスクを設定します。このレジスタは CPU の割込みコントローラーへの割込みの転送をマスクするだけで、INTR レジスタへの割込みのロギングを有効/無効に設定するものではありません。
GPIO_PRTx_INTR_MASKED	ポート割込みマスク ステータス レジスタ	CPU 割込みコントローラーに転送される INTR と INTR_MASK レジスタの AND 値。
GPIO_PRTx_INTR_SET	ポート割込みセット レジスタ	ファームウェアがピン割込みを設定できるようにします。
GPIO_PRTx_INTR_CFG	ポート割込みコンフィギュレーション レジスタ	各ピン割込みのエッジ検出タイプを選択します。
GPIO_PRTx_CFG	ポート コンフィギュレーション レジスタ	各ピンのドライブモードと入力バッファイネーブルを選択します。
GPIO_PRTx_CFG_IN	ポート入力コンフィギュレーション レジスタ	各ピンの入力バッファモード (CMOS または TTL) を設定します。VTRIP_SEL[7:0]_0。
GPIO_PRTx_CFG_OUT	ポート出力コンフィギュレーション レジスタ	ポート出力バッファのドライブ強度を設定します。
GPIO_PRTx_CFG_SLEW_EXT	ポート出力コンフィギュレーション レジスタ	ポート出力バッファのスルーレートを設定します。
HSIOM_PRTx_PORT_SELy	HSIOM ポート選択レジスタ	I/O ピンへのハードウェア ペリフェラル接続を選択します。
GPIO_INTR_CAUSEz	割込みポート原因レジスタ	ポート (0 + z × 32) ～ (31 + z × 32) に対応する割込みステータスを提供します。“z”は 0 から 3 まで。
GPIO_VDD_ACTIVE	外部電源検出レジスタ	外部電源ステータスの提供。

(続く)

表 154 (続き) レジスタの概要

レジスタ	名称	説明
GPIO_VDD_INTR	電源検出割込みレジスタ	このレジスタは、電源のランプ アップまたはランプ ダウンが検出されるたびにセットされます。電源のシーケンスによっては、システム電源投入後に設定されるビットもあります。
GPIO_VDD_INTR_MASK	電源検出割込みマスクレジスタ	すべての電源に対する電源検出割込みを設定します。これは CPU への割込みの転送をマスクするだけで、VDD_INTR レジスタへの割込みのロギングを有効/無効に設定するものではありません。
GPIO_VDD_INTR_MASKED	電源検出割込みマスクド レジスタ	VDD_INTR と VDD_INTR_MASK レジスタの AND 値を含みます。
GPIO_VDD_INTR_SET	電源検出割込みセットレジスタ	ファームウェアまたはデバッガが、対応するビット フィールドに'1'を書き込むことによって、VDD_INTR レジスタの割込みビットを設定できるようにします。読み出すと、VDD_INTR レジスタと同じ値を返します。
GPIO_VDD_PORSW_STATUS	電源スイッチ ステータス	電源スイッチの状態を示します。
GPIO_VDD_PORSW_CTL	電源スイッチ制御レジスタ	GPIO の突入電流制限レベルセレクト、スレッショルドセレクト、レベルセットを設定します。
GPIO_SEC_INTR_CAUSE	セキュア割込みポート原因レジスタ	ポートに対応する割込みステータスを提供します。
SMARTIO_PRTx_CTL	SMARTIO 制御レジスタ	特定のポートの SMARTIO の制御レジスタです。有効化、クロックソース、バイパスなどを制御します。
SMARTIO_PRTx_LUT_SELy	LUT コンポーネント入力選択レジスタ	LUT 入力選択レジスタ。LUT_TR0_SEL、LUT_TR1_SEL、および LUT_TR2_SEL
SMARTIO_PRTx_LUT_CTLy	LUT コンポーネント制御レジスタ	LUT のオペコードを提供します。
SMARTIO_PRTx_DU_SEL	データ ユニット コンポーネント入力選択レジスタ	データ ユニット入力選択レジスタ
SMARTIO_PRTx_DU_CTL	データ ユニット コンポーネント制御レジスタ	データ ユニット制御レジスタ
SMARTIO_PRTx_DATA	データ レジスタ	データユニット入力データソース

24 タイマー/カウンター/パルス幅変調器 (TCPWM)

PSOC™ Control C3 MCU のタイマー/カウンター/パルス幅変調器 (TCPWM) ブロックは、16 ビットまたは 32 ビットの TCPWM、モーション インターフェース (MOTIF) を実装しています。TCPWM には 3 つのカウンター グループがあり、グループ 1 とグループ 2 は最大 8 つのカウンターをサポートし、グループ 0 は最大 4 つのカウンターのみをサポートします。このカウンターは、入力信号の周期とパルス幅の測定 (タイマー)、特定のイベントの発生回数の検出 (カウンター)、および PWM 信号の生成に使用できます。TCPWM ブロックはアクティブとスリープ モードで動作します。

ここでは TCPWM ブロックの機能、実装、動作モードを説明します。

24.1 特長

TCPWM の特長は、以下のとおりです。

- 3 つのカウンターグループ:
  - グループ 1 とグループ 2 は最大 8 つのカウンターに対応。グループ 0 は最大 4 つのカウンターのみに対応
  - 各カウンターは、以下の機能モードのいずれかで実行できます。
    - コンペア機能付きタイマー カウンター
    - キャプチャ付きタイマー カウンター
    - パルス幅変調 (PWM)
    - デッドタイム付き PWM
- 設定可能なモーションインターフェース (MOTIF) ブロック (グループ 1 にのみ含まれる)
- MOTIF モジュールは、直行デコーダー、ホールセンサー、スタンドアロンマルチチャネルモード、およびスタンドアロンマルチチャネル付き直行モードの 4 つの異なるモードで動作できます。
- 32 ビット TCPWM カウンターで高分解能 PWM 生成 (HRPWM) 機能に対応
- HRPWM は PWM と PWM_DT モードのみに限定されます
- HRPWM サポートはグループ 0 にのみ適用されます。
- 16 ビットまたは 32 ビット カウンター、2 つのコンペア/キャプチャ (CC0/CC1)、周期レジスタ
- カウンター コンペア、ピリオド、ライン セル、デッド タイム レジスタはダブル バッファ仕様
- アップ、ダウン、アップ/ダウンカウントモード
- クロック プリスケーリング (1 倍、2 倍、4 倍、8 倍、16 倍、32 倍、64 倍、128 倍)
- すべてのカウンターで使用される汎用トリガーと、各カウンターの特定の 1 対 1 のトリガーでサポートされる設定可能な深さのグリッチフィルター
- すべての HW 入力信号の立ち上りエッジ、立ち下りエッジ、立ち上りエッジと立ち下りエッジの組み合わせの検出、またはパススルーは、カウンター イベントを発生させるために使用できます。
- start、reload、stop、capture0、capture1 の各イベントは、ソフトウェアで生成できます。
- アンダーフロー、オーバーフロー、cc0_match、cc1_match、cc0_cc1_match、LINE_OUT、および TC イベントを示すために、各カウンターに 2 つの出力トリガー信号
- 各カウンターに 2 本の PWM 相補出力ライン。デッドタイム挿入は 16 ビットで、2 つの PWM 出力に対して独立
- クロック プリスケーリングとデッドタイムの独立制御
- グループ 1 およびグループ 2 において、周期およびデューティ サイクルの PWM ディザリングを設定可能
- デバッグ モードでのキル極性によるパッシブ状態への移行機能を備えた、継続/一時停止の設定可能なオプション

- 停止中またはキル動作中の PWM 信号の出力状態を、周期一致に同期して制御します。
- キル動作中の PWM 信号の独立したライン極性設定

24.2 アーキテクチャ

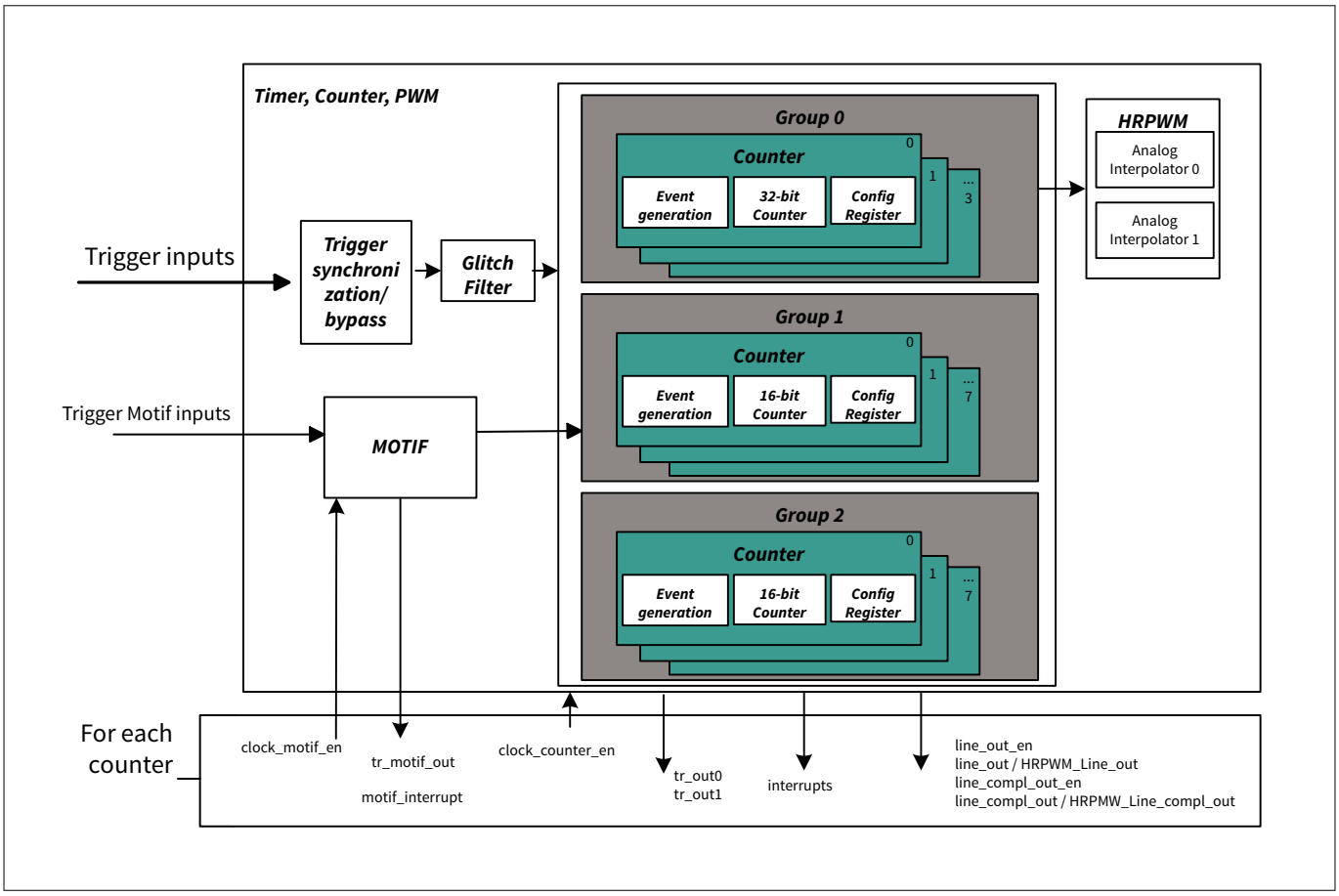


図 92 TCPWM ブロック図

カウンターを制御する 3 つの主要レジスタは以下のとおりです。

- COUNTER レジスタには現在のカウントが入っています。
- PERIOD レジスタは、カウンターの上限値を保持します。カウンターが n サイクルカウントする場合、このフィールドに n-1 をセットします。
- CC0/1 レジスタは、CAPTURE モードでカウンター値をキャプチャするために使用されます。他のすべてのモードでは、カウンター値と比較されます。

各カウンターは 19 の入力トリガー信号と 2 つの定数入力信号 ('0'と'1') を持つことができ、これらはすべて CLK_PERI クロックに同期しています。

TCPWM ブロックには、以下のインターフェースがあります。

- **バス インターフェース:** AHB-5 インターフェースを介してブロックを CPU サブシステムに接続します。
- **入出力信号インターフェース:** 入力トリガー (Reload、Start、Stop、Count、Capture0/1 など) と出力信号 (LINE_OUT、LINE_COMPL_OUT、TR_OUT0、TR_OUT1 など) から構成されます。すべての入力信号は、Reload トリガーがリロード イベントを生成するように、カウンター内のイベントをトリガーするために使用されます。出力信号は内部イベント (アンダーフロー、オーバーフロー、キャプチャ/コンペア) によって生成され、イベントをトリガーするために他のペリフェラルに接続できます。
- **割込み:** ターミナル カウント (TC)、コンペア/キャプチャ CC0_match、またはコンペア/キャプチャ CC1_match イベントに基づいて、各カウンターからの割込み要求信号を提供します。

このブロックに必要なすべてのレジスタの詳細については、レジスタの概要を参照してください。

24.2.1 TCPWM ブロックにおけるカウンターの有効/無効

カウンターは、CTRL レジスタの対応する ENABLE ビットに'1'を書き込むことで有効にでき、同じビットに'0'を書き込むことで無効にできます。

注: カウンターは、有効にする前に設定する必要があります。カウンターを無効にしても、コンフィギュレーション レジスタの値は保持されます。

24.2.2 クロック

TCPWM カウンターはそれ自身のクロックを持つことができます。クロックの唯一の供給源は、クロッキング システムによって生成される設定可能な周辺クロック分周器です。詳細は、クロック システムの章を参照してください。TCPWM 内の特定のカウンター用にクロック分周器を選択するためには、周辺レジスタ空間の CLOCK_CTL レジスタを使用します。ここでは、カウンターのクロックを clk_counter と呼びます。イベント生成は clk_counter で行われます。もう 1 つのクロック、システムクロック (clk_sys) は、出力トリガーのパルス幅に使用されます。システム クロック (clk_sys) は clk_peri に同期していますが、CLOCK_CTL を使って分周できます。

24.2.2.1 クロック プリスケーリング

clk_counter は、1、2、4、8、16、32、64、128 の値で、各カウンター内部でさらに分割できます。この分割はプレスケーリングと呼ばれます。プリスケーリングは PS レジスタの PS_DIV [2:0]フィールドで設定されます。

24.2.2.2 カウントイベント

カウンター機能は、"カウント イベント"信号によってゲートされる"アクティブ カウント"プリスケール クロックで実行されます。カウンターはクロックサイクルのカウントイベント検知により、カウント値が増えたり減ったりします。

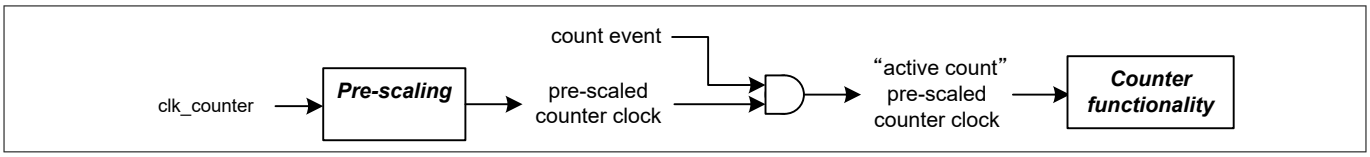


図 93 カウンタークロック生成

すべてのステータスや出力の変化は、"アクティブ カウント"プリスケールされたカウンター クロックでのみ起こります。言い換えると、カウントイベントが非アクティブの場合、カウンター、ステータス、割込み、およびすべての出力は値を変更しません。例えば、カウント イベントがパス スルー モードで、ダウン カウント モードでカウンターが"0"になったときにカウント イベントがローであれば、ターミナル カウント (tc) イベントとアンダーフロー イベントは、カウント イベントがハイになった次のプリスケーリングされたカウンター クロックで生成されます。唯一の例外は即時 Kill モードです。Kill 入力は、"アクティブ カウント"プリスケーリング カウンター クロックに関係なく、PWM 出力を即座に抑制します。

注: すべてのモードではサポートされていません。

24.2.3 トリガー入力

TCPWM ブロックには、19 の Trigger_In 信号と、SCB、DMA、HPPASS などの他のオンチップ リソースから来る定数'0'および'1'信号があります。Trigger_In 信号は、TCPWM ブロック内の全カウンターと共有されます。

2 種類のトリガー信号が同期しており、カウンターがイベントを生成するために使用できます。

- **汎用トリガー:** これらはすべてのカウンターで使用できます。これらのトリガーは、システム内の異なるブロックによって生成され、トリガー インフラストラクチャー (周辺トリガーマルチプレクサ) によって分配されます。詳細についてはトリガー マルチプレクサ (トリガー MUX)の章を参照してください
- **1 対 1 のトリガー:** 各カウンターには個別のセットが存在し、そのカウンターにのみ接続されます。これらのトリガーは、トリガーソース (HPPASS レベルトリガーなど) から関連する TCPWM カウンターへの直接トリガー接続に使用されます。

TR_IN_SEL0 と TR_IN_SEL1 というトリガー mux レジスタを使って、TCPWM ブロックの Trigger_In にルーティングされる信号を設定します。TCPWM ブロックの入力トリガーイベントを選択するすべての可能なマルチプレクサ設定については、表 155 を参照してください。各イベントに対して、2 つの定数トリガー入力が利用可能です。入力トリガー 0 は常に一定の'0'で、入力トリガー 1 は常に一定の'1'です。

各カウンターは、21 のトリガー信号の中から、以下のイベントのソースとなる信号を選択できます。

- Capture0 と Capture1
- Count
- Reload
- Stop/Kill
- Start

***注**:* *PERI0_TR_CMD レジスタは、ソフトウェアからそれぞれ Reload、Stop、Start、Capture0/1 をトリガーするために使用できます。*

表 155 入力トリガーイベントのマルチプレクサ選択

入力トリガー選択レジスタ	ビットフィールド	ビット	説明
TR_IN_SEL0	CAPTURE0_SEL	7:0	21 の入力トリガーのうちの 1 つを Capture0 トリガーとして選択します。PWM、PWM_DT モードでは、このトリガーは、コンペアおよびピリオド レジスタがバッファ対応であれば、その値を切り替えるために使用されます。
	COUNT_SEL	15:8	21 の入力トリガーのうち 1 つを Count トリガーとして選択します。
	RELOAD_SEL	23:16	21 の入力トリガーの 1 つを Reload トリガーとして選択します。
	STOP_SEL	31:24	21 の入力トリガーのうちの 1 つを Stop トリガーとして選択します。PWM、PWM_DT モードでは、これが Kill トリガーとなります。

(続く)

表 155 (続き) 入力トリガーイベントのマルチプレクサ選択

入力トリガー選択レジスタ	ビットフィールド	ビット	説明
TR_IN_SEL1	START_SEL	7:0	21 の入力トリガーの中から 1 つを Start トリガーとして選択します。
	CAPTURE1_SEL	15:8	最大 21 の入力トリガーから 1 つを Capture1 トリガーとして選択します。

以下のセクションでは、各 TCPWM モードと各入力イベントの機能について詳細に説明します。

典型的な操作では、カウンターを初期化してスタートさせるために reload イベントを 1 回使用し、カウンターをストップさせるために stop イベントを使用します。カウンターが停止している場合、start イベントを使用して、停止時のカウンター値を変更せずにカウンターをスタートできます。

ストップ、リロード、スタートの各イベントが重なった場合、以下の優先順位関係が成り立ちます。

- 停止イベントは、リロードイベントよりも高い優先順位です。
- リロード イベントは、スタート イベントよりも高い優先順位です。

その結果、リロードイベントやスタートイベントがストップイベントと重なっても、リロードイベントやスタートイベントは何の影響も及ぼしません。

各トリガー_IN は、カウンターに送られる前に、正エッジ検出器、負エッジ検出器、両エッジ検出器を通過するか、そのままカウンターに送られます。これは TR_IN_EDGE_SEL レジスタで制御されます。

検出された複数のイベントは、以下のように扱われます。

- 立ち上りエッジモードと立ち下りエッジモードでは、複数のイベントが効果的に 1 つのイベントに縮小されます。その結果、イベントが失われることがあります。
- 立ち上り/立ち下りエッジモードでは、偶数個のイベントは検出されず、奇数個のイベントは 1 つのイベントに縮小されます。立ち上り/立ち下りエッジ モードは通常、パルスの幅を決定するためのキャプチャ イベントに使用されるからです。電流機能により、立ち上りと立ち下りの交互パターンが維持されます。

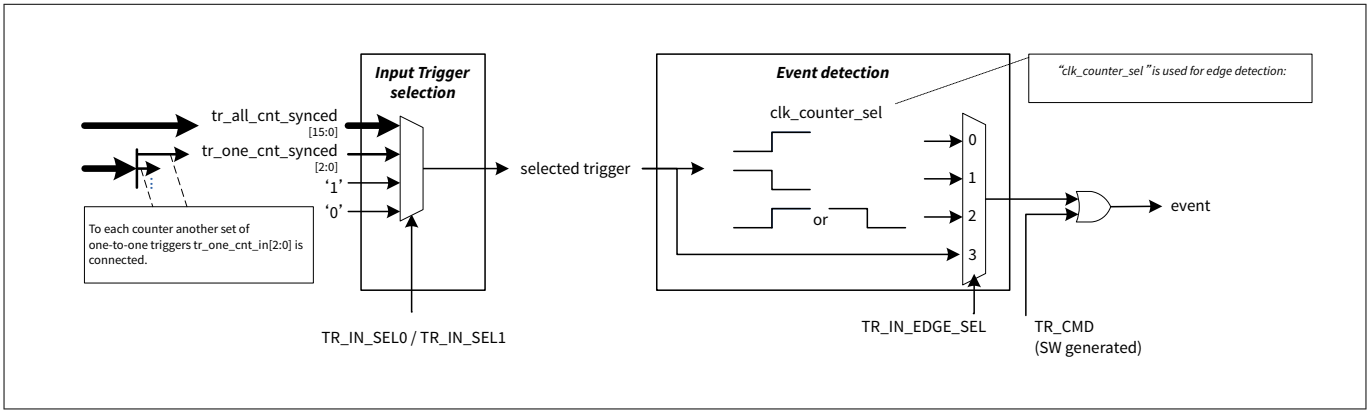


図 94 TCPWM 入力イベント

表 155 によれば、TR_IN_SEL0 と TR_IN_SEL1 レジスタによって、トリガーイベント発生用の専用入力トリガー信号を定義できます。定数信号 (定数'0'と定数'1')、特定の 1 対 1 トリガー入力信号、または汎用入力トリガー信号から選択できます。図 94 に、入力トリガーソースがどのように選択されているかを示します。

1 対 1 入力トリガー (tr_one_cnt_synced) と汎用入力トリガー (tr_all_cnt_synced) の数はデバイス固有です (PSOC™ Control C3 MCU データシートを参照)。

表 156 に、入力トリガーイベント生成のためにマルチプレクサがどのように処理されるべきかを示します。

表 156　　入力トリガー マルチプレクサの処理

入力トリガー選択	入力トリガー	入力トリガーソース
0	定数'0'	定数'0'
1	定数'1'	定数'1'
2	PASS (プログラマブル アナログ サブシステム)、1:1 トリガー mux#0 経由	トリガー入力の章を参照してください。
3	HSIOM	デバイス データシートの「代替ピン機能」セクションを参照してください。
4	HSIOM	デバイス データシートの「代替ピン機能」セクションを参照してください。
5	tr_all_cnt_in[0]	トリガー mux ブロックを参照してください。
...		
20	tr_all_cnt_in[15]	トリガーマルチプレクサブロックを参照

***注**:　　入力トリガーは異なるソースから生成できます。汎用トリガー入力 (tr_all_cnt_in[0] ～ tr_all_cnt_in[15]) は、トリガー マルチプレクサ ブロック(トリガー入力 の章を参照) からのみですが、1 対 1 入力トリガーは、外部 GPIO 入力ピンからも生成できます。*

すべてのトリガー入力は、周辺クロックに同期しています。同じ counter_clock 期間に複数のイベントが発生した場合、1 つ以上のイベントが見逃されることがあります。これは、高周波数イベント (カウンター周波数に近い周波数) や、プリスケーリングされた (分周された) カウンター クロックを使用するタイマー設定で起こりえます。

次の図に、入力トリガーがカウンターに検出されるタイミングを示します。

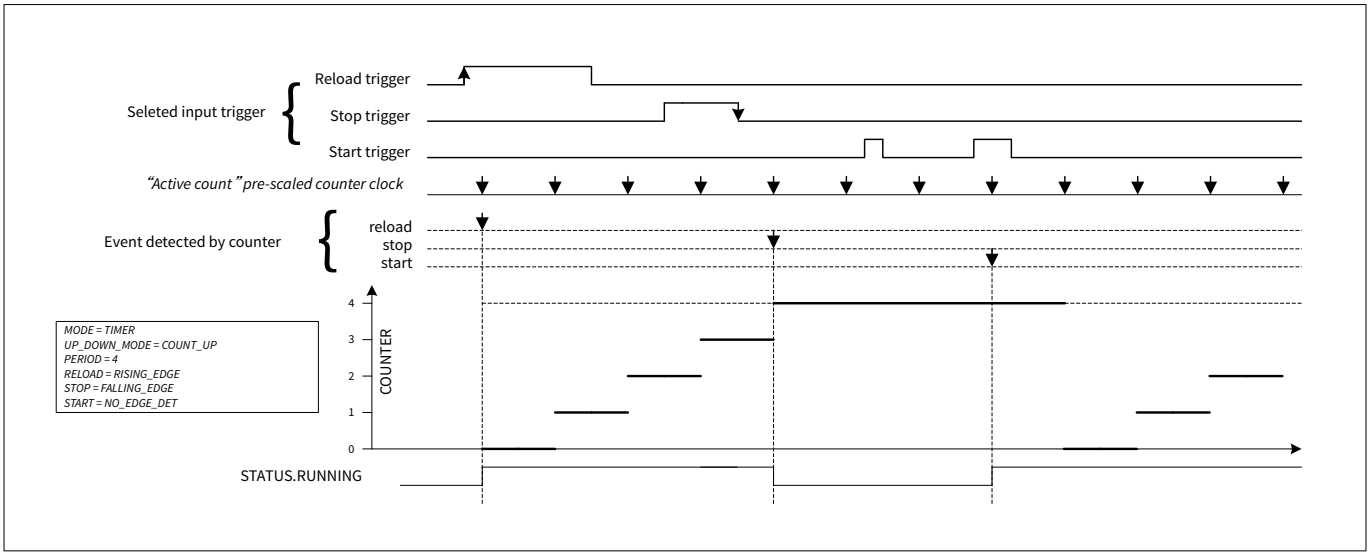


図 95　　「アクティブ カウント」プリスケーリング カウンター クロックによる入力トリガー検出

***注**:　　以下のセクションの図で矢印で示されたイベントは、カウンターによって検出されたイベントを示します。*

通常、カウントイベントは一定の'1'で、プリスケーリングはオフです。この場合、"アクティブ カウント"プリスケーリングされたカウンター クロックは、カウンター クロックと同じです。他のケースでは、エッジ検出は、検出されたイベントが使用される次の"アクティブ カウント"プリスケールされたカウンター クロックの前に、(カウンター クロック上で) 複数のイベントを検出するかもしれません。検出された複数のイベントは、以下のように扱われます。

- 立ち上りエッジモードと立ち下りエッジモードでは、複数のイベントが効果的に 1 つのイベントに縮小されます。その結果、イベントが失われることがあります。
- 立ち上り/立ち下りエッジモードでは、偶数個のイベントは検出されず、奇数個のイベントは 1 つのイベントに縮小されます。立ち上り/立ち下りエッジ モードは通常、パルスの幅を決定するためのキャプチャ イベントに使用されるからです。電流の機能により、立ち上り、立ち下り、立ち上り、立ち下りなどのパターンが交互に維持されます。

パス スルー イベントは CLK_PERI によって保持されません。パス スルー イベントは、それが持続し、カウンター オペレーション クロックによって検出されることができれば、機能に影響を与える。パススルー イベントのパルス幅がカウンター動作クロック サイクルより小さい場合、それは失われることがあります。パススルー検出の結果、複数のカウンター クロックの間アクティブなイベントが発生することがあります。これは、カウンターとそれに関連するトリガー出力の望ましくない動作につながることがあります。パススルー イベント検出は、ほとんどの機能モードではストップ イベントとカウント イベント タイプにのみ使用します。パススルーモードは、PWM/PWM_DT モードのスイッチイベントでも使用できます。

24.2.4 グリッチフィルタリング

グリッチフィルタリングとは、通常ハイレベルかローレベルのデジタル入力信号から不要なパルスを除去する工程です。グリッチ フィルターは、汎用トリガーや特定の 1 対 1 トリガーに挿入できます。グリッチ フィルターは clk_peri で動作します。

グリッチ フィルターは、汎用トリガーに使用されるグリッチ フィルターは ALL_GF レジスタで設定でき、1 対 1 トリガーに使用されるグリッチ フィルターは ONE_GF レジスタで設定できます。ALL_GF レジスタと ONE_GF レジスタは、ハードウェア的には同じ設計で、同じレジスタ コンフィギュレーション設定を持っています。これらは、汎用トリガーや 1 対 1 トリガーに接続されるグリッチ フィルターのレジスタ名を区別するために、異なる名称が付けられています。グリッチ フィルターの各インスタンスには、独立したコンフィギュレーションのためのレジスタがあります。

GF_DEPTH と GFPS_DIV の組み合わせにより、グリッチフィルターの深さを 0～512 の範囲で選択できます。

図 96 に、グリッチ フィルターの入力の継続時間がグリッチ フィルターの深さ以上の場合に、グリッチ フィルターの出力がどのように変化するかを示します。それ以外の場合、出力は現在の値から変化しません。

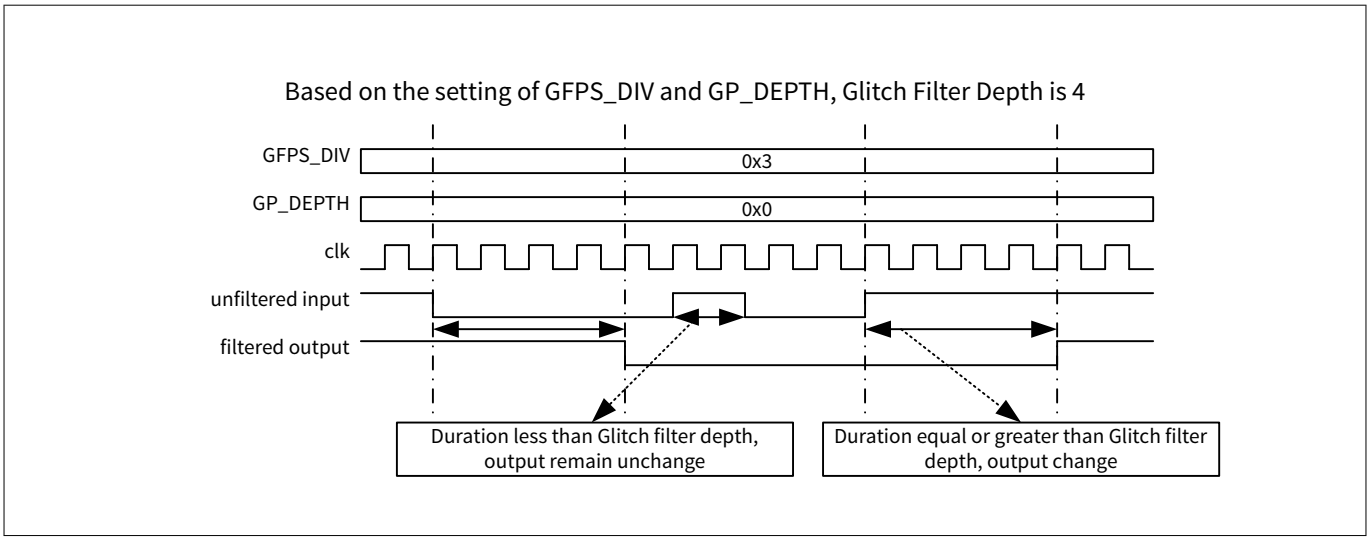


図 96 グリッチフィルターの動作

図 179 に、2 つの異なる TCPWM カウンターグループに対する汎用トリガーと 1 対 1 のトリガーへのグリッチフィルターの接続例を示します。

24.2.5 複数のカウンターの同期

前のセクションでは、ハードウェアベースのイベント生成について説明しました。PERI0_TR_CMD レジスタへの書き込みにより、Reload、Start、Stop、Capture0、Capture1 の各イベントを生成できます。これらはカウンター固有のレジスタで、1 つのカウンターに対してのみソフトウェアベースのイベント発生を許可します。すべてのカウンターで同じトリガー信号を選択し (TR_IN_SEL0 と TR_IN_SEL1 レジスタを使用)、PERI ブロックの PERI0_TR_CMD レジスタでトリガーを生成することで、同期したソフトウェアベースのイベント生成 (複数のカウンターを同期してスタートさせるなど) が可能です。

PERI ブロックの TR_CMD で複数のカウンターを同期スタートさせる例を下図に示します。

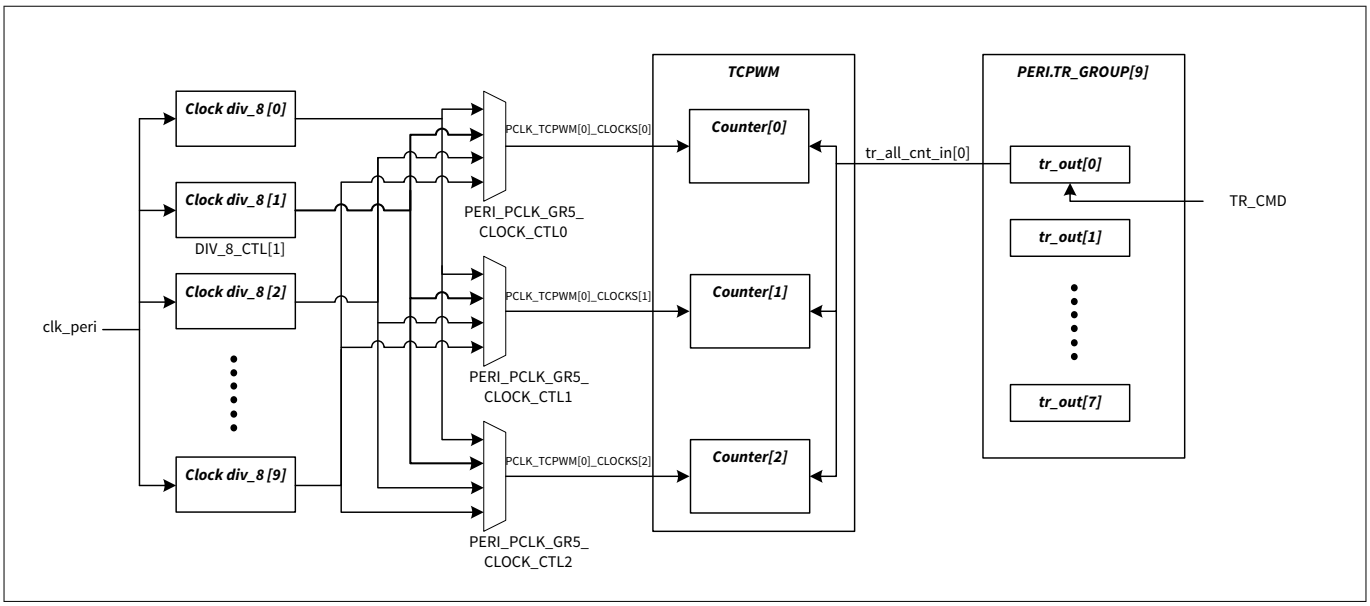


図 97 PERI TR_CMD がカウンターを同期スタート

次の例では、カウンターを同期的に開始するために必要な手順を説明します。

- PERI_PCLK_GR5_CLOCK_CTL0/1/2 を設定し、同じクロック分周器 clock_div_8[1]を選択します。
- DIV_8_CTL[1] と DIV_CMD を設定して TCPWM 用のクロック分周イネーブル信号を生成します。したがって、3 つのカウンターは同じクロック分周イネーブル信号を持ち、言い換えれば、それらのクロックは同期しています。
- 3 つのカウンターを 1 つずつ有効にしてください - カウンターは有効になりますが、スタートまたはリロードイベントが検出されるまで実行されません
- tr_all_cnt_in[0]を選択し、3 つのカウンターのスタートイベントを設定します。これは、tr_all_cnt_in[0]がアサートされると、3 つのカウンターが同期して動作することを意味します。
- TR_CMD を使用して、TR_GROUP[9].TR_OUT[0]にトリガーパルスを生成します。3 つのカウンターはすべて同期して動作します

注: ここに挙げたレジスタはすべて PERI ブロックに属します (クロッキング システムの章を参照)。

- ソフトウェアベースのイベントは、TR_CMD それぞれのビットを'1'に書き込んだ後にセットされ、次の「アクティブ カウント」プリスケール カウンター クロックでハードウェアによってクリアされます。特定のモードにおけるいくつかのイベントについては、TC イベントを持つ次の"アクティブ カウント"プリスケール カウンター クロックでクリアされます。イベント検出設定 (TR_IN_EDGE_SEL) は、ソフトウェアベースのイベントには影響しません。

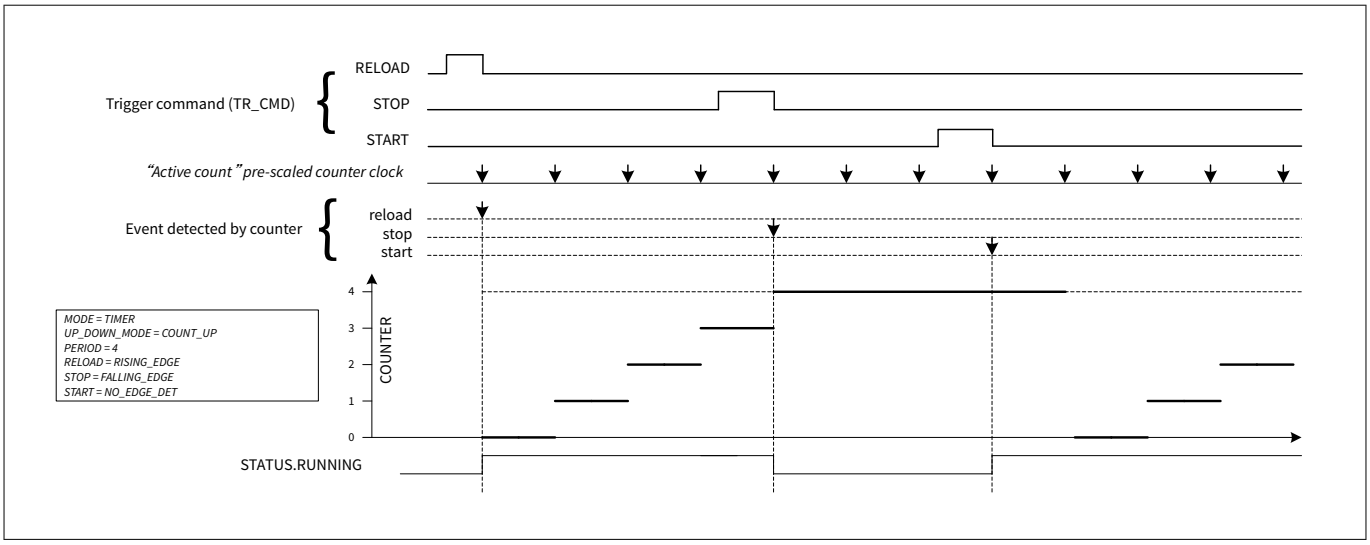


図 98　　アクティブ カウント プリ スケール カウンター クロックによるソフトウェア トリガー コマンド検出

24.2.6　トリガー出力

各カウンターには 2 つのトリガー出力信号 (TR_OUT0 と TR_OUT1) があり、トリガー mux をとおしてデバイス上の他のペリフェラルにルーティングできます。TR_OUT_SEL レジスタのビットフィールド OUT0 は、出力トリガー 0 (TR_OUT0) を生成する内部イベントの 1 つを選択するために使用され、ビットフィールド OUT1 は、TR_OUT1 を生成する内部イベントの 1 つを選択します。また、出力トリガーを無効にすることもできます。

以下は、カウンター トリガー出力信号によって示されるイベントです。

- オーバーフロー (OV): オーバーフローイベントは、アップカウントモードで、COUNTER が PERIOD レジスタと等しく、異なる値に変更されたことを示します。
- アンダーフロー (UN): アンダーフロー イベントは、ダウン カウント モードにおいて、COUNTER が 0 に等しく、異なる値に変化することを示します。
- TC (ターミナル カウント): TC イベントは、カウンターのアンダーフローとオーバーフロー イベントの論理和です。
- CC0/1_MATCH: このイベントは、カウンターが動作中に以下のいずれかの条件が発生したときに発生します。
  - カウンターは比較値と等しくなります。このイベントは、一致が起ころうとしているとき (COUNTER が CC0/1 と等しくなく、CC0/1 に変化したとき)、または一致が起ころうとしていないとき (COUNTER が CC0/1 と等しく、異なる値に変化したとき) に発生します。
  - キャプチャ イベントが発生し、CC0 (CC1) と CC0_BUFF (CC1_BUFF) レジスタが更新された。
- CC0_CC1_MATCH: このイベントは CC0_MATCH または CC1_MATCH が発生したときにトリガーされます。
- LINE_OUT: PWM 出力信号
- DISABLED: 出力トリガーが無効

出力トリガー発生イベントの選択は、TR_OUT_SEL レジスタによって行われます。また、出力トリガーを無効にすることもできます。

***注**:　これらの信号は CLK_SYS の 2 サイクルの間のみハイレベルを維持します。信頼性の高い動作のためには、このトリガーを発生させる条件は、CLK_SYS の最大 1/4 である必要があります。例えば CLK_SYS の周波数が 24 MHz の場合、トリガーが発生する条件は 6 MHz 未満の周波数となります。*

出力トリガーに LINE_OUT を選択した場合、出力トリガーは 2 サイクル パルス生成ロジックをバイパスし、LINE_OUT を直接出力します。

生成されたトリガーには主に 2 つの用途があります。

- DMA データ転送の開始。例えば、アップカウントタイマーを使用した PWM モードでは、オーバーフローを使用して、新しい周期とコンペア値をメモリからカウンターの PERIOD_BUFF と CC0_BUFF レジスタに転送できます。
- プログラマブル デジタル コンポーネントにおける PWM 信号の再構成。パルス幅変調 (PWM) モードに記述されているように、PWM ライン出力信号は cc0_match (cc1_match)、アンダーフロー、オーバーフローの内部イベントから得られます。これらの内部イベントを出力トリガーとして利用できるようにすることで、他のコンポーネントが PWM 信号を再構築し、変更できる可能性があります (前述の周波数制限に注意してください)。

24.2.7 内部イベント

24.2.7.1 アンダーフロー イベント

アンダーフロー イベントは、ダウン カウントにおいて COUNTER がゼロに等しく、異なる値に変化したことを示します。Reload はまた、ある特定のモードにおいてアンダーフロー イベントを発生させます。次の表は、各ファンクション モードのアンダーフロー発生をまとめたものです。

表 157 アンダーフロー発生

モード	UP	DOWN	UPDN1	UPDN2
タイマ	カウンターがデクリメントされ、COUNTER が 0 に等しい状態から変化します。DOWN、UPDN1、UPDN2 モードでの Reload イベント。			
キャプチャ	カウンターがデクリメントされ、COUNTER が 0 に等しい状態から変化します。DOWN、UPDN1、UPDN2 モードでの Reload イベント。			
PWM	カウンターがデクリメントされ、COUNTER が 0 に等しい状態から変化します。DOWN、UPDN1、UPDN2 モードでの Reload イベント。			
PWM_DT	カウンターがデクリメントされ、COUNTER が 0 に等しい状態から変化します。DOWN、UPDN1、UPDN2 モードでの Reload イベント。			

24.2.7.2 オーバーフローイベント

オーバーフローイベントは、アップカウントにおいて COUNTER が PERIOD と等しくなり、異なる値に変化したことを示します。Reload はまた、ある特定のモードでオーバーフローイベントを発生させます。

次の表は、各機能モードのオーバーフロー発生をまとめたものです。

表 158 オーバーフロー発生

モード	UP	DOWN	UPDN1	UPDN2
タイマ	カウンターはインクリメントし、COUNTER が PERIOD と等しい状態から変化します。UP カウントモードでの Reload イベント。			
キャプチャ	カウンターはインクリメントし、COUNTER が PERIOD と等しい状態から変化します。UP カウントモードでの Reload イベント。			
PWM	カウンターはインクリメントし、COUNTER が PERIOD と等しい状態から変化します。UP カウントモードでの Reload イベント。			
PWM_DT	カウンターはインクリメントし、COUNTER が PERIOD と等しい状態から変化します。UP カウントモードでの Reload イベント。			

24.2.7.3 ターミナル カウントイベント

TC イベントは、カウンターのアンダーフローとオーバーフロー イベントの論理和です。例外として、リロードイベントはアンダーフローまたはオーバーフローを発生させるが、TC イベントは発生させません。

次の表は、各機能モードの TC 生成をまとめたものです。

表 159 TC 生成

モード	UP	DOWN	UPDN1	UPDN2
タイマ	オーバーフロー	アンダーフロー	アンダーフロー	オーバーフローとアンダーフローの論理和
キャプチャ	オーバーフロー	アンダーフロー	アンダーフロー	オーバーフローとアンダーフローの論理和
PWM	オーバーフロー	アンダーフロー	アンダーフロー	オーバーフローとアンダーフローの論理和
PWM_DT	オーバーフロー	アンダーフロー	アンダーフロー	オーバーフローとアンダーフローの論理和

24.2.7.4 cc0_match (cc1_match) イベント

cc0_match イベントは、COUNTER が CC0 に等しいことを示します。このイベントは、COUNTER が CC0 に変わろうとするとき、または COUNTER が CC0 と等しく、異なる値に変わろうとするときに発生します。特殊なケースとして、PWM モードでのデューティ サイクルが 0 または 100 パーセントの発生があります。詳細は、パルス幅変調 (PWM) モードを参照。その他の特定の動作モードでは、このイベントは CC0/CC0_BUFF レジスタが更新されたことを示すために使用されます。cc1_match は COUNTER と CC1 の状態に基づいて発生します。その他の動作は cc0_match と同じです。

以下の表は、各機能モードの cc0/1_match 発生についてまとめたものです。

表 160 cc0_match 発生

モード	UP	DOWN	UPDN1	UPDN2
タイマ	カウンターは、COUNTER が CC0 に等しい状態から変化します。			
キャプチャ	Capture0 イベント			
PWM	カウンターは、COUNTER が CC0 に等しい状態に変化します。		COUNT_UPDN1/2: カウンターが COUNTER と CC0 が等しい状態から変化します。 第 2 のコンペア機能がカウンターグループに存在する場合、 CC0_MATCH_DOWN_EN/ CC0_MATCH_UP_EN は cc0_match 発生を有効/無効化します。	

(続く)

表 160 (続き) cc0_match 発生

モード	UP	DOWN	UPDN1	UPDN2
PWM_DT	カウンターは、COUNTER が CC0 に等しい状態に変化します。		COUNT_UPDN1/2: カウンターが COUNTER と CC0 が等しい状態から変化します。第 2 のコンペア機能がカウンター グループに存在する場合、CC0_MATCH_DOWN_EN/ CC0_MATCH_UP_EN は cc0_match 発生を有効/無効にします。	

表 161 cc1_match 発生

モード	UP	DOWN	UPDN1	UPDN2
タイマ	カウンターは、COUNTER が CC1 と等しい状態から変化します。			
キャプチャ	Capture1 イベント			
PWM	カウンターは、COUNTER が CC1 と等しい状態に変化します。		COUNT_UPDN1/2: カウンターが COUNTER と CC1 が等しい状態から変化します。 第 2 のコンペア機能がカウンター グループに存在する場合、CC1_MATCH_DOWN_EN/ CC1_MATCH_UP_EN は cc1_match 発生を有効/無効にします。	
PWM_DT	カウンターは、COUNTER が CC1 と等しい状態に変化します。		COUNT_UPDN1/2: カウンターが COUNTER と CC1 が等しい状態から変化します。 第 2 のコンペア機能がカウンター グループに存在する場合、CC1_MATCH_DOWN_EN/ CC1_MATCH_UP_EN は cc1_match 発生を有効/無効にします。	

24.2.8 割込み

TCPWM ブロックは、各カウンターに対して専用の割込み出力を提供します。割込みはカウンター モードに固有で、ターミナル カウント (TC) または Compare/Capture0/1 (CC0/1) イベントに対して発生させられます。TC は、OV と UN イベントの論理和です。

このブロックでは、表 162 に示すように 4 つのレジスタが割込みの処理に使われています。

表 162 割込みレジスタ

割込みレジスタ	ビット	名称	説明
INTR (割込み要求レジスタ)	0	TC	ターミナル カウント検出時、このビットは'1'に設定されます。クリアするためには'1'を書き込んでください。

(続く)

表 162 (続き) 割込みレジスタ

割込みレジスタ	ビット	名称	説明
	1	CC0_MATCH	カウント値がキャプチャ/比較 0 (CC0) のレジスタ値と一致時、このビットは'1'に設定されます。クリアするためには'1'を書き込んでください。
	2	CC1_MATCH	カウント値がキャプチャ/比較 1 (CC1) のレジスタ値と一致時、このビットは'1'に設定されます。クリアするためには'1'を書き込んでください。
INTR_SET (割込みセット要求レジスタ)	0	TC	割込み要求レジスタの対応するビットのセットは'1'を書き込んでください。読み出す時、このレジスタは割込み要求レジスタステータスを反映
	1	CC0_MATCH	割込み要求レジスタの対応するビットのセットは'1'を書き込んでください。読み出す時、このレジスタは割込み要求レジスタステータスを反映
	2	CC1_MATCH	割込み要求レジスタの対応するビットのセットは'1'を書き込んでください。読み出す時、このレジスタは割込み要求レジスタステータスを反映
INTR_MASK (割込みマスクレジスタ)	0	TC	割込み要求レジスタの対応する TC ビットのマスクビット。
	1	CC0_MATCH	割込み要求レジスタの対応する CC_MATCH0 ビットのマスクビット。
	2	CC1_MATCH	割込み要求レジスタの対応する CC_MATCH1 ビットのマスクビット。
INTR_MASKED (割込みマスク要求レジスタ)	0	TC	対応する TC 要求とマスクビットの論理積
	1	CC0_MATCH	対応する CC_MATCH0 要求とマスクビットの論理積

(続く)

表 162 (続き) 割込みレジスタ

割込みレジスタ	ビット	名称	説明
	2	CC1_MATCH	対応する CC_MATCH1 要求とマスクビットの論理積

24.2.9 デバッグ モード

TCPWM カウンターは、CTRL.DBG_FREEZE_EN ビットと CTRL.DBG_SUS_EN ビットによってデバッグをサポートします。デバッグ状態 (例えば、ブレーク ポイントまで実行した後) でカウンターの動作が継続するか、一時停止するかは、カウンターごとに設定できます。この機能は、TCPWM カウンターを OS タイマーとして使用する場合に特に意図されています。これは、DBG_FREEZE_EN = '1'の間、デバッグ状態に入る際にカウンター クロック"clk_counter"をゲートすることによって実現されます。

この表は、CTRL.DBG_FREEZE_EN ビットと CTRL.DBG_SUS_EN ビットに基づくデバッグモードの動作を表にしたものです。

表 163 デバッグ モード

	DBG_FREEZE_EN	DBG_SUS_EN
カウンター動作継続	0	0
カウンターの動作は直ちに一時停止します	1	0
tc イベントの後、カウンター動作は一時停止し、パッシブ状態になります。受動状態は CTRL.KILL_LINE_POLARITY によって制御されます。	0	1
カウンターの動作は直ちに一時停止し、パッシブ状態になります。受動状態は CTRL.KILL_LINE_POLARITY によって制御されます。STATUS.RUNNING ビットを 0 に設定。	1	1

注: デバッグモードで TCPWM をフリーズさせる場合は、DBG_FREEZE_EN と DBG_SUS_EN のみでなく、シグナル (tr_debug_freeze) も設定する必要があります。tr_debug_freeze は DEBUG600 System Cross Trigger Interface (CTI) のものです。

注: フリーズ機能とサスペンド機能は、デバッグ セッションがアクティブのときのみ有効のため、有効にする前に PWR_CTL レジスタの DEBUG_SESSION ビットでデバッグ セッションがアクティブかどうかを確認してください。

24.2.10 PWM 出力

PWM、PWM_DT 動作モードは 2 つの出力信号を生成します。

- PWM LINE_OUT 出力信号
- コンプリメンタリ PWM LINE_COMPL_OUT 出力信号 (LINE_OUT の反転バージョン)

注: PWM モードと PWM_DT モードでは、TR_PWM_CTRL レジスタ(表 164 参照) を設定することにより、CC0_match、CC1_match、アンダーフロー、オーバーフローの内部イベント条件が LINE_OUT と LINE_COMPL_OUT の駆動に使用されます。

表 164 OV、UN、CC0/1 条件の出力ラインの設定

文字列	ビット	設定値	イベント	内容
CC0_MATCH_MODE 初期設定値 = 3	1:0	0	LINE_OUT を'1'にセット	比較一致 (CC0) イベントの発生時に出力ラインを設定
		1	LINE_OUT を'0'にクリア	
		2	LINE_OUT を反転	
		3	変更なし	
OVERFLOW_MODE 初期設定値 = 3	3:2	0	LINE_OUT を'1'にセット	オーバーフロー (OV) イベントの発生時に出力ラインを設定
		1	LINE_OUT を'0'にクリア	
		2	LINE_OUT を反転	
		3	変更なし	
UNDERFLOW_MODE 初期設定値 = 3	5:4	0	LINE_OUT を'1'にセット	アンダーフロー (UN) イベントの発生時に出力ラインを設定
		1	LINE_OUT を'0'にクリア	
		2	LINE_OUT を反転	
		3	変更なし	
CC1_MATCH_MODE 初期設定値 = 3	7:6	0	LINE_OUT を'1'にセット	比較一致 (CC1) イベントの発生時に出力ラインを設定
		1	LINE_OUT を'0'にクリア	
		2	LINE_OUT を反転	
		3	変更なし	

PWM 出力信号の生成は多段階プロセスです。LINE_OUT と LINE_COMPL_OUT はどちらも PWM 信号線から生成されます。PWM 信号線は、カウンターの TR_PWM_CTRL レジスタで指定された cc0_match、cc1_match、アンダーフロー、オーバーフローの内部イベントの状態に応じて生成されます。各内部イベントに対して、TR_PWM_CTRL レジスタは、そのイベントがどのように出力 LINE_OUT に影響するかを指定します。

- 出力は 0 に設定
- 出力は 1 に設定
- 出力は反転
- 出力に影響はありません

内部 cc0_match イベントが内部アンダーフローまたはオーバーフローイベントの発生と同時に発生した場合、cc0_match は LINE_OUT がアンダーフロー/オーバーフローの設定に従って状態を変化させた後に有効となります。cc1_match は cc0_match の後に有効になります。表 165 はそのメカニズムを示すためにいくつかの例を挙げています。

表 165 LINE_OUT の作成例

一致するケース	オーバーフロー	アンダーフロー	CC0_match	CC1_match	LINE_OUT
CC0_match とオーバーフロー	CLEAR	ドント ケア	INVERT	ドント ケア	1 (SET)
CC0_match とアンダーフロー	ドント ケア	SET	INVERT	ドント ケア	0 (CLEAR)
CC0_match および CC1_match	ドント ケア	ドント ケア	SET	CLEAR	CLEAR
CC0_match と CC1_match とオーバーフロー	INVERT	ドント ケア	INVERT	INVERT	INVERT

24.2.11 電力モード

TCPWM ブロックは、ACTIVE、LPACTIVE、SLEEP、および LPSLEEP モードで動作します。TCPWM ブロックはデジタル コア電源に接続されています。リテンション レジスタは、値を保持する必要がある MMIO レジスタの状態を保持するため、DEEPSLEEP モードでも電力が供給されます。

ACTIVE、LPACTIVE、SLEEP、および LPSLEEP 電力モードでは、clk_peri が提供されます。DEEPSLEEP モードでは clk_peri は動作していません。

Vccact は HIBERNATE モードと DEEPSLEEP モードではカットオフされますが、ACTIVE/LPACTIVE/SLEEP/LPSLEEP モードでは供給されます。

Vccret は DEEPSLEEP モードではリテンション レジスタに供給され、HIBERNATE モードではオフになります。

電源スイッチはブロックに内蔵されていません。

表 166 TCPWM ブロックの電力モード

電力モード	ブロックの状態
ACTIVE	このブロックは、電源オンでクロックが動作するこのモードで完全に動作します。
LPACTIVE	LPACTIVE (LOWPOWERACTIVE) は ACTIVE に似ていますが、電流を減らすためにいくつかのトレードオフが適用されます。
SLEEP	このモードでは CPU はスリープしていますが、機能はオンになっています。カウンターのクロックはすべてオン。
LPSLEEP	LPSLEEP (LOWPOWERSLEEP) は、SLEEP が ACTIVE に関係するのと同じように、LPACTIVE に関係します。
DEEPSLEEP	ブロックへの電源はオンですがバス クロックがないためブロックが動作しません。すべてのリテンション レジスタは、電源 Vccret でもその状態を維持します。
HIBERNATE	このモードでは、ブロック全体の電源がオフになり、clk_peri は提供されません。

24.3 カウンターの動作モード

表 167 に示すようにカウンター ブロックは 6 つの動作モードで機能します。カウンター制御レジスタ (CTRL) の MODE[26:24]フィールドはカウンターを特定の動作モードに設定します。

表 167 動作モード コンフィギュレーション

モード	MODE フィールド[26:24]	説明
タイマ	000	カウント イベントが検出されるカウンター クロック サイクルごとにカウンターは'1'ずつインクリメント/デクリメントされます。コンペア/キャプチャ レジスタは、カウントの比較に使用されます。
キャプチャ	010	カウント イベントが検出されるカウンター クロック サイクルごとにカウンターは'1'ずつインクリメント/デクリメントされます。キャプチャイベント検知時、カウント値はキャプチャレジスタにコピーされます。
PWM	100	パルス幅変調 (PWM) モード。HRPWM 機能は HRPWM_CTRL レジスタをとおしてこのモードでサポートされます。
PWM_DT	101	デッドタイム挿入有りパルス幅変調。HRPWM 機能は HRPWM_CTRL レジスタをとおしてこのモードでサポートされます。

表 168 に示すように TCPWM_CNT_CTRL レジスタの UP_DOWN_MODE[17:16]フィールドを設定することでカウンターをカウントアップ、ダウン、アップ/ダウンに設定できます。

表 168 カウント モード コンフィギュレーション

カウント モード	UP_DOWN_MODE[17:16]	説明
カウントアップ モード	00	周期に達するまでカウンターをインクリメント。ターミナル カウント (TC) 条件は、カウンターが Period 値から変化したときに発生します。
カウントダウン モード	01	0 に達するまでカウンターを Period からデクリメント。TC 条件は、カウンターの値が'0'から変化したときに生成されます。
カウントアップ/ダウン モード 1	10	Period 値に達するまでカウンターをインクリメントし、'0'に達するまでカウンターをデクリメントします。TC 条件は、カウンターの値が'0'から変化したときのみ発生します。

(続く)

表 168 (続き) カウント モード コンフィギュレーション

カウント モード	UP_DOWN_MODE[17:16]	説明
カウントアップ/ダウン モード 2	11	アップ/ダウンカウンティングモード 1 と似ていますが、カウンターが'0'から変化したときと、カウンター値が Period 値から変化したときに TC 条件が発生します。

24.3.1 タイマーモード

タイマーモードは、イベントの所要時間や 2 つのイベント間の時間差を測定するために使用できます。タイマー機能は、0 と PERIOD レジスタに格納された値の間でカウンターをインクリメント/デクリメントします。カウンターが動作しているとき、COUNTER レジスタに格納されたカウント値はコンペア/キャプチャ レジスタ (CC0 および CC1) と比較されます。COUNTER が CC0 に等しいとき、cc0_match イベントが発生し、COUNTER が CC1 に等しいとき、cc1_match イベントが発生します。

タイマー機能は通常、以下のいずれかに使用されます。

- 特定の遅延のタイミング - カウントイベントは一定の'1'
- 特定のイベントの発生をカウントする - イベントを入力トリガーとして接続し、カウントイベントに選択する必要があります。

表 169 タイマーモードトリガー入力の説明

トリガー入力	使用法
Reload	カウンターを初期化し、スタートさせます。動作は UP_DOWN_MODE に依存します。 • COUNT_UP: カウンターは'0'にセットされ、カウント方向は'up'にセットされます。 • COUNT_DOWN: カウンターは PERIOD に設定され、カウント方向は「down」に設定されています。 • COUNT_UPDN1/2: カウンターは'1'にセットされ、カウント方向は'up'にセットされます。 カウンターが動作中または動作していないときに使用できます。
Start	カウンターを開始します。カウンターはハードウェアによって初期化されません。現在のカウンターの値が使用されます。動作は UP_DOWN_MODE に依存します。カウンターが作動していないとき: • COUNT_UP: カウントの方向は'up'に設定されています • COUNT_DOWN: カウントの方向は「down」に設定されています • COUNT_UPDN1/2: カウント方向は変更されません カウンターが動作しているときは、開始イベントは何の効果もなく、カウンターが動作しているときでもしていないときでも使用できることに注意してください。
Stop	カウンターを止めます。

(続く)

表 169 (続き) タイマーモードトリガー入力の説明

トリガー入力	使用法
Count	カウントイベントは、カウンターをインクリメント/デクリメントします。
Capture0	このイベントは、外部アップ/ダウン カウント イベントとして使用されます。
Capture1	未使用

カウンターのインクリメントとデクリメントは、count イベントと clk_counter によって制御されます。典型的な動作は、一定の'1'カウント イベントと、プリスケーリングなしの clk_counter を使用します。例えば、カウンター イベント コンフィギュレーションでは、同期入力トリガーの立ち上りエッジをカウントできます。

表 170 タイマーモード対応機能

サポートされる機能	説明
クロック プリスケーリング	clk_counter をプリスケーリングします。
One shot	カウンターは tc イベントでハードウェアにより停止されます。COUNT_UPDN2 では、アンダーフロー時に tc イベントでカウンターを停止します。
オートリロード CC	CC0 と CC0_BUFF は、cc0_match イベントで交換されます (CTRL.AUTO_RELOAD_CC で指定されている場合、入力イベントは必要ありません)。 CC1 と CC1_BUFF は、cc1_match イベント時に交換されます (CTRL.AUTO_RELOAD_CC で指定された場合)。
アップ/ダウン モード	UP_DOWN_MODE によって指定されます。 • COUNT_UP: カウンターは 0 から PERIOD までカウントします。 • COUNT_DOWN: カウンターは PERIOD から 0 までカウントします。 • COUNT_UPDN1/2: カウンターは 1 から PERIOD までカウントし、0 に戻ります。

表 171 は、トリガー出力と、それらがトリガーされたときの条件をリストアップしています。

表 171 タイマーモードトリガー出力

トリガー出力	説明
cc0_match	カウンターは、COUNTER が CC0 に等しい状態から変化します。
cc1_match	カウンターは、COUNTER が CC1 と等しい状態から変化します。

(続く)

表 171 (続き) タイマーモードトリガー出力

トリガー出力	説明
アンダーフロー (UN)	カウンターがデクリメントされ、COUNTER が 0 に等しい状態から変化します。 COUNT_DOWN、COUNT_UPDN1、または COUNT_UPDN2 モードで、Reload イベントがアンダーフローを生成します。
オーバーフロー (OV)	カウンターはインクリメントし、COUNTER が PERIOD と等しい状態から変化します。 COUNT_UP モードで Reload イベントがオーバーフローを発生させます。
TC	UP_DOWN_MODE によって指定されます。 • COUNT_UP: tc イベントはオーバーフローイベントと同じです • COUNT_DOWN: tc イベントはアンダーフローイベントと同じです • COUNT_UPDN1: tc イベントはアンダーフロー イベントと同じです。 • COUNT_UPDN2: tc イベントは、オーバーフローイベントとアンダーフローイベントの論理和と同じです。 Reload はアンダーフロー/オーバーフローを発生させるが、tc は発生させません。

表 172 タイマーモード PWM 出力

PWM 出力	説明
LINE_OUT	未使用
LINE_COMPL_OUT	未使用

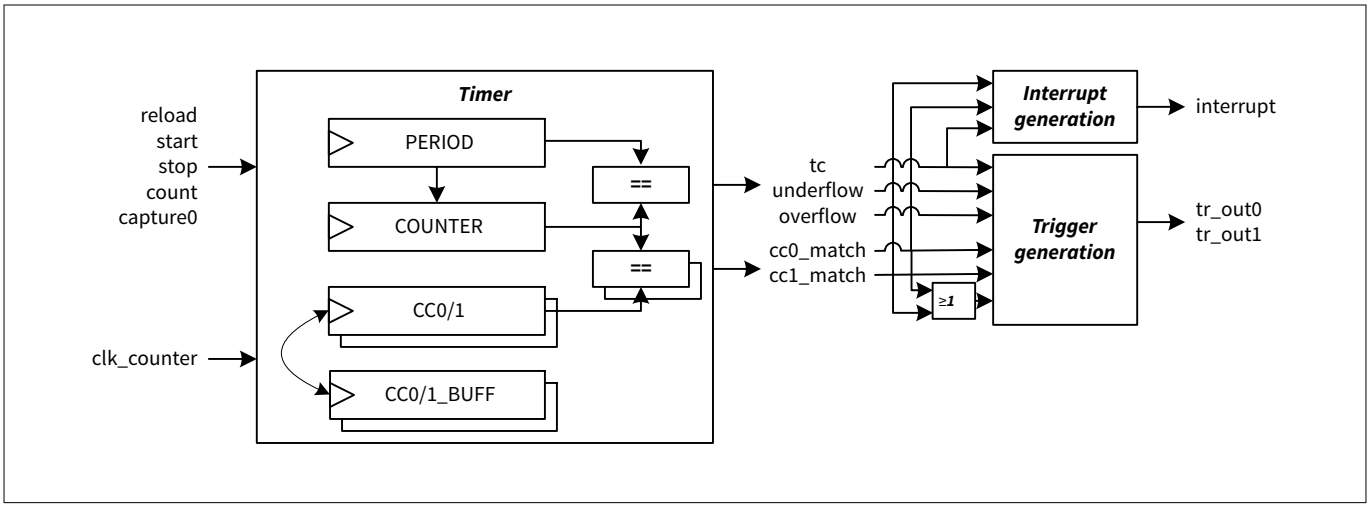


図 99 タイマー機能

注:
- *トリガー tr_out0 と tr_out1 は、それぞれ内部イベント cc0_match、cc1_match、cc0_match と cc1_match の OR 組み合わせ、アンダーフロー、オーバーフロー、tc に基づいて生成されます (選択はレジスタ TR_OUT_SEL によって行われます)。*
- *タイマー機能は、PERIOD*
- *カウンターが動作しているときに COUNTER に書き込むことは推奨されません。*
- *COUNT_UPDN2 モードでは、**ソフトウェアは、COUNTER が"0"** のとき (そして、COUNTER が PERIOD のとき)、TC 条件のための PERIOD のみを変更する必要があります。*

図 100 にアップカウントモードのタイマーを示します。カウンターは (0 に) 初期化され、ソフトウェアベースのリロードイベントで開始されます。

注:
- *カウンターが 4 の状態から変化すると、オーバーフローと tc イベントが発生します。*
- *カウンターが COUNTER が 2 の状態から変化するとき、cc0_match イベント発生 PERIOD は 4 となり、4+1 = 5 clk_counter 周期の有効な繰り返しカウンター パターンとなります。CC0 レジスタは 2 で、cc0_match イベントの条件を設定します。*

以下のシナリオでは、'1'の定数カウント イベントとプリスケーリングなしの clk_counter が使用されます。カウント イベントが'0'でリロード イベントがトリガーされた場合、リロードはカウント イベントが'1'の最初のクロック エッジでのみ登録されます。つまり、カウント イベントが'1'の最初のクロック エッジはカウントに使用されません。リロードに使用します。

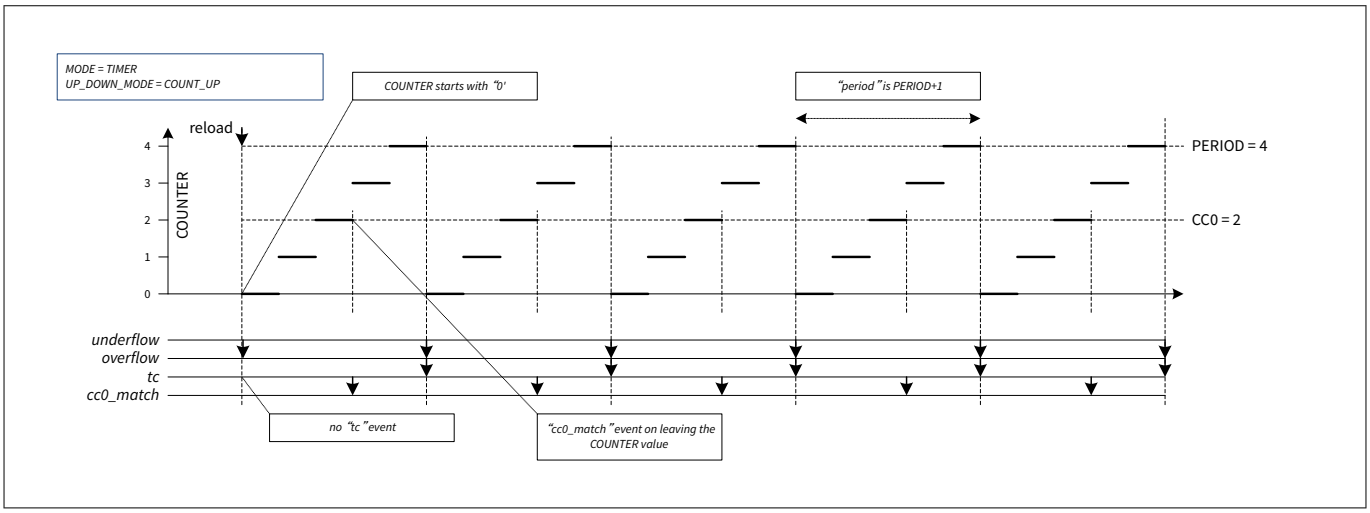


図 100 アップカウントモードのタイマー

図 101 に、ダウンカウントモードのタイマーを示します。カウンターは (PERIOD に) 初期化され、ソフトウェアベースのリロード イベントで開始されます。

注:
- *COUNTER が 0 の状態からカウンターが変化すると、アンダーフローと tc イベントが発生します。*
- *カウンターが COUNTER = 2 の状態から変化すると、cc0_match イベントが発生します。*
- *PERIOD は 4 で、4+1 = 5 クロック周期の有効な繰り返しカウンター パターンとなります。*

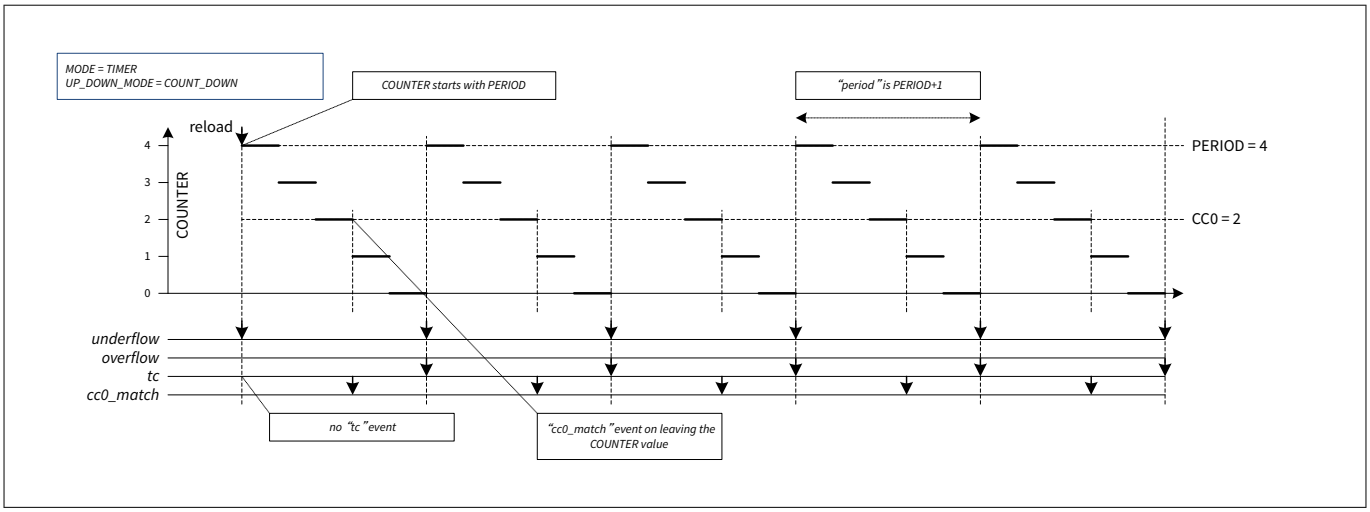


図 101　　ダウンカウントモードのタイマー

図 102 に、アップ/ダウン カウント モード 1 のタイマーを示します。カウンターは (1 に) 初期化され、ソフトウェアベースのリロードイベントで開始されます。

***注**:*
- *カウンターが 4 の状態から変化すると、オーバーフローが発生します。*
- *COUNTER が 0 の状態からカウンターが変化すると、アンダーフローと tc イベントが発生します。*
- *カウンターが COUNTER = 2 の状態から変化すると、cc0_match イベントが発生します。*
- *PERIOD は 4 で、2*4 = 8 クロック周期の有効な繰り返しカウンター パターンとなります。*

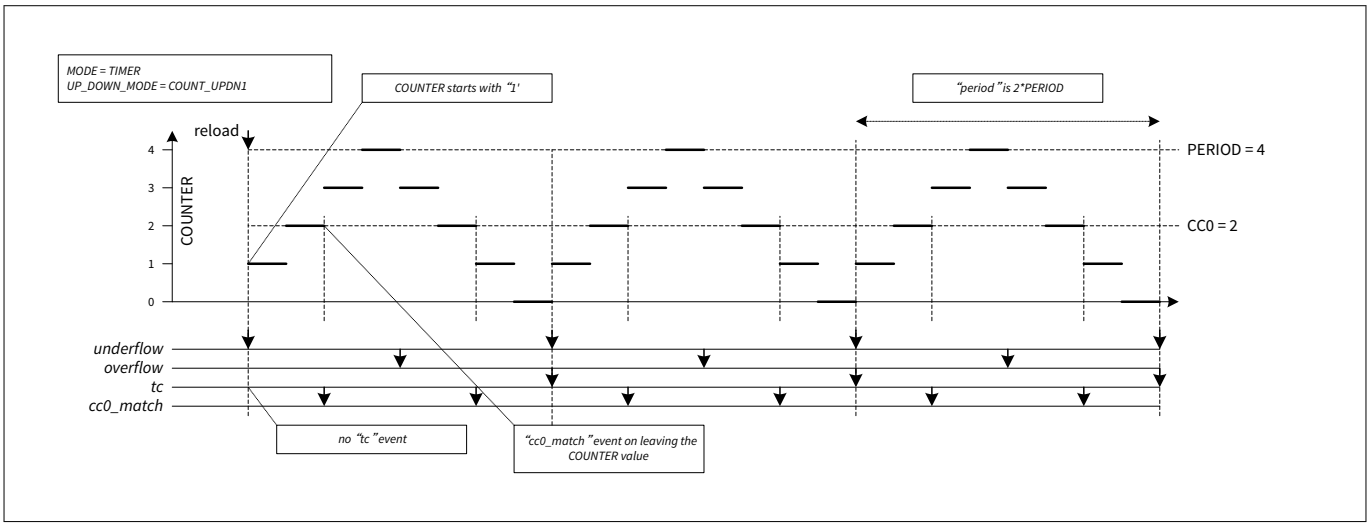


図 102　　タイマーのアップ/ダウン カウント モード 1

COUNT_UPDN2 は、オーバーフローイベントで発生する tc イベントを除いて、COUNT_UPDN1 と同じ動作をします。

図 103 に「ワンショット」動作モードのタイマーを示します。

***注**:*
- *tc イベントでカウンターが停止*
- *COUNT_UPDN2 では、アンダーフロー時に tc イベントでカウンターが停止します。*
- *COUNTER は tc の次の値に留まります。以下の図では、カウンターが停止した後も"0"のままです。*

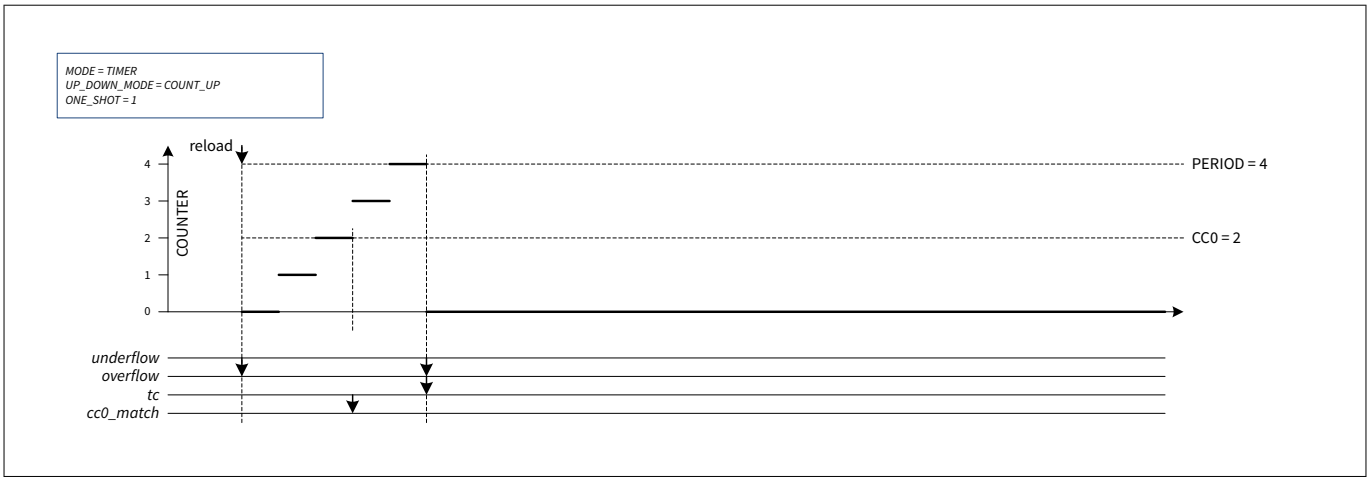


図 103　　ワンショットタイマー

図 104 にクロックのプリスケーリングを示します。対応するプリスケーラ値は“/1”, “/2”, “/4”, “/8”, “/16”, “/32”, “/64”, “/128”です。

***注**:*　　• *カウンターは 1 サイクルおきにのみインクリメントされます。*

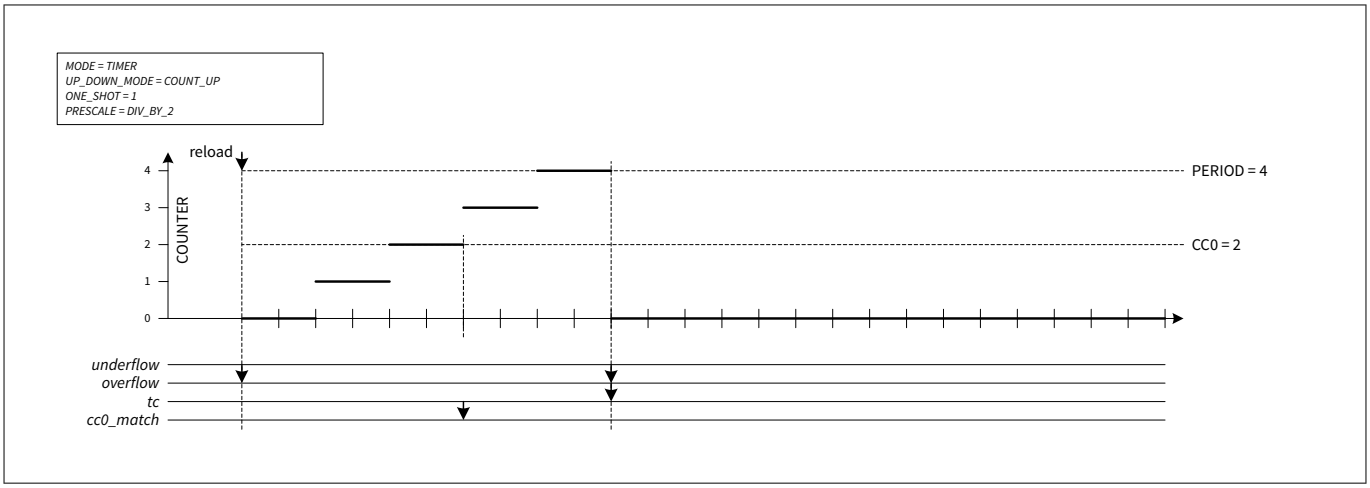


図 104　　タイマークロック プリスケーリング

図 105 に、SWAP_ENABLE = "1"のときに CC0 と CC0_BUFF の両方のレジスタを使うタイマーを示します。

***注**:*　　*CC0 と CC0_BUFF は cc0_match イベントで交換されます。*

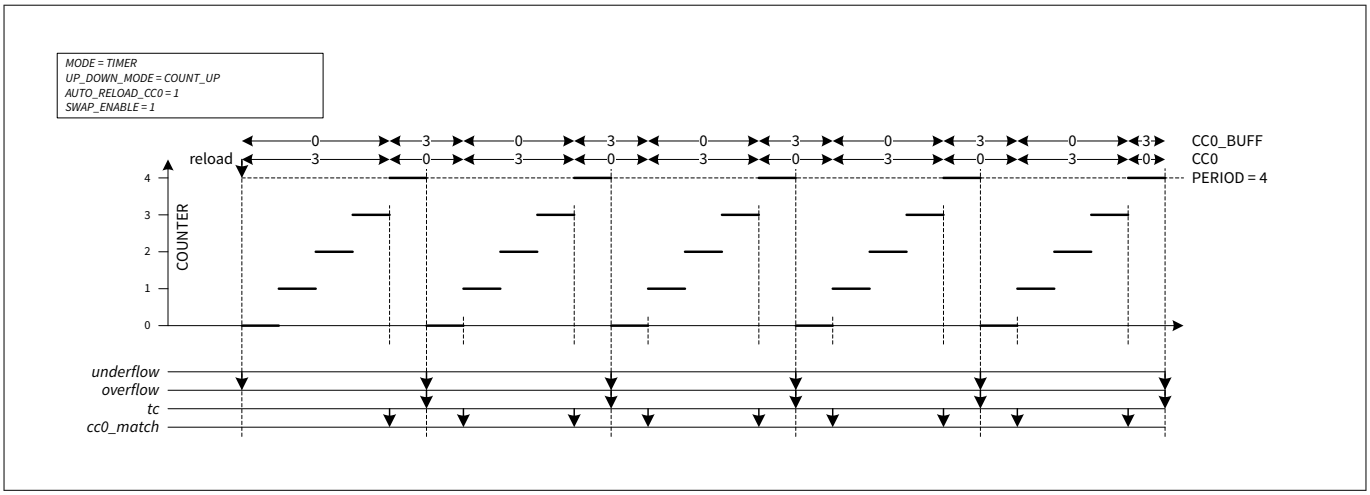


図 105　　SWAP_ENABLE = “1” 時、cc0_match で自動的に切り替わる CC0/CC0_BUFF

CTRL.SWAP_ENABLE が"0"の場合、cc0_match イベントがアサートされると、CC0_BUFF 値は HW によって CC0 に転送されます (交換のトリガーとなる入力イベントは必要ありません)。

図 106 に、SWAP_ENABLE = "0"のときに CC0 と CC0_BUFF の両方のレジスタを使うタイマーを示します。

注: *CC0_BUFF は cc0_match イベントで CC0 に転送されます。*

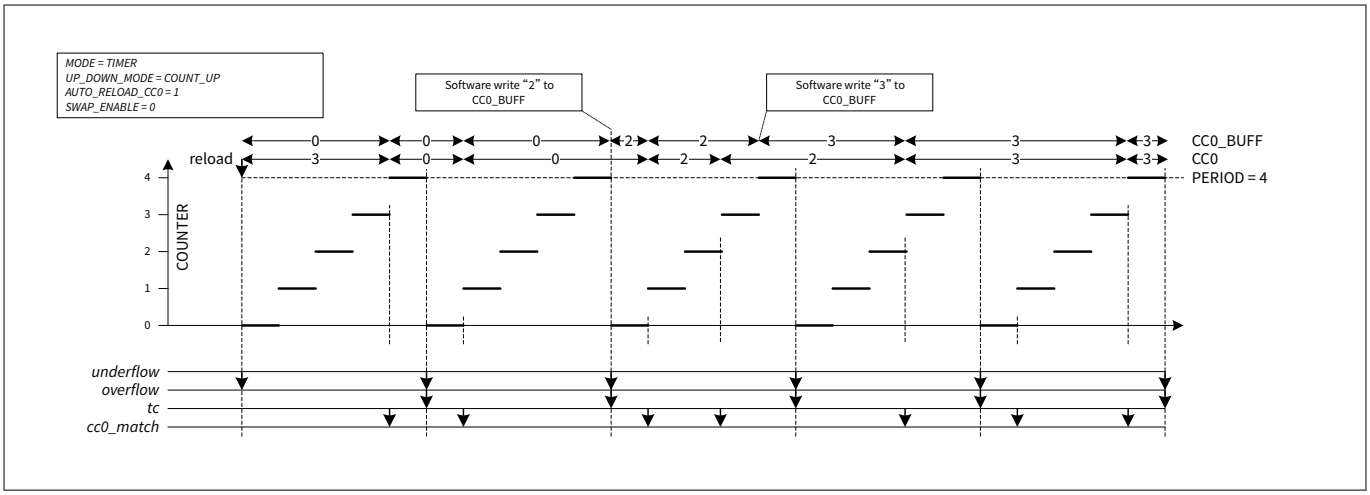


図 106　　SWAP_ENABLE = “0” 時、cc0_match で自動的に CC0 に切り替わる CC0_BUFF

上記の説明は CC1/CC1_BUFF に適用されます。

タイマーモードは、リロード/スタート/ストップ/カウント入力イベントをサポートします。

図 107 に、初期化され、開始され (リロードイベント)、停止され (ストップイベント)、継続/開始される (スタートイベント) カウンターを示します。

注: *実行中でないとき (STATUS.RUNNING) は、カウンターの値は変化しません。*

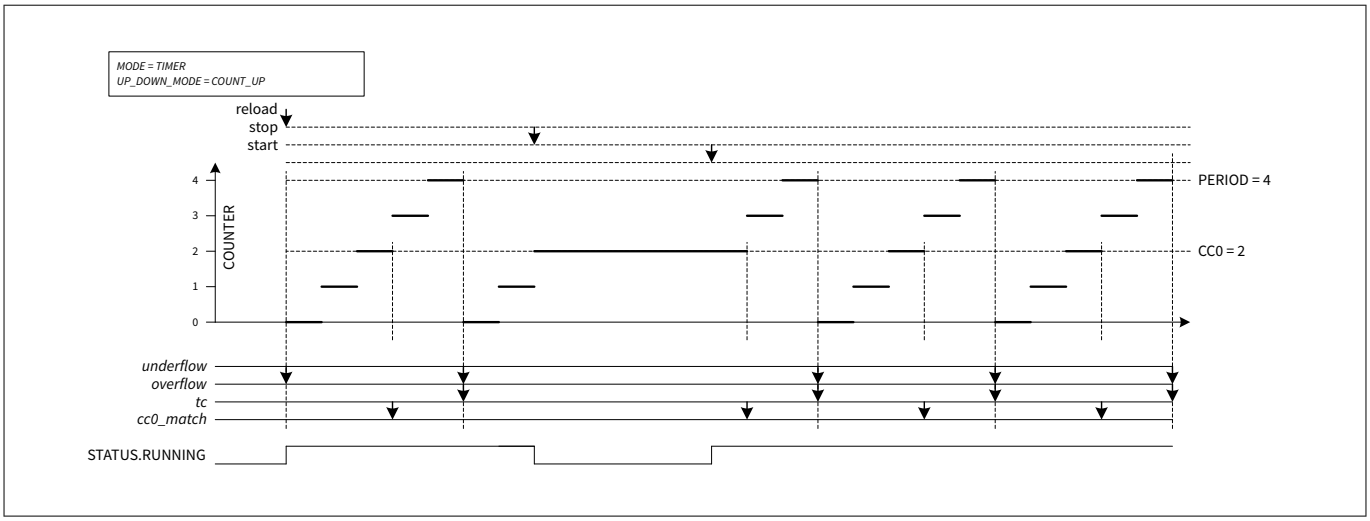


図 107　　カウンター開始/停止/継続

Reload はカウンターが動作しているときに入力できます。以下の説明は、リロードが tc イベント、cc0_match イベント、stop イベントと重なった場合の動作を明確にしたものです。

- 停止イベントはリロードイベントより優先されます
- アンダーフロー、オーバーフロー、tc イベント発生において、Reload イベント発生が「実行中カウンター」機能より優先されます。
- Reload イベントが決定します。
  - アンダーフローイベント生成
  - オーバーフローイベント発生
  - tc イベント発生 (常に'0')
- "ランニングカウンター"機能が決定します。
  - アンダーフローイベント生成
  - オーバーフローイベント発生
  - tc イベント生成
  - cc0_match イベント発生
  - CC0/CC0_BUFF 交換

図 108 は、カウンターが動作している間にロー レベルになるカウント イベントと、カウント イベントがロー レベルのときにアサートされる立ち上りエッジ検出ストップ イベントを示しています (この場合、カウント イベントはパス スルー モードとして使用されます)。COUNTER は、カウント イベントが Low の時にラン アンド ホールド値を停止します。ストップ イベントは、カウント イベントがハイ (「アクティブ カウント」プリスケール カウンター クロック) になるまで、実行ステータスをローに変更しません。

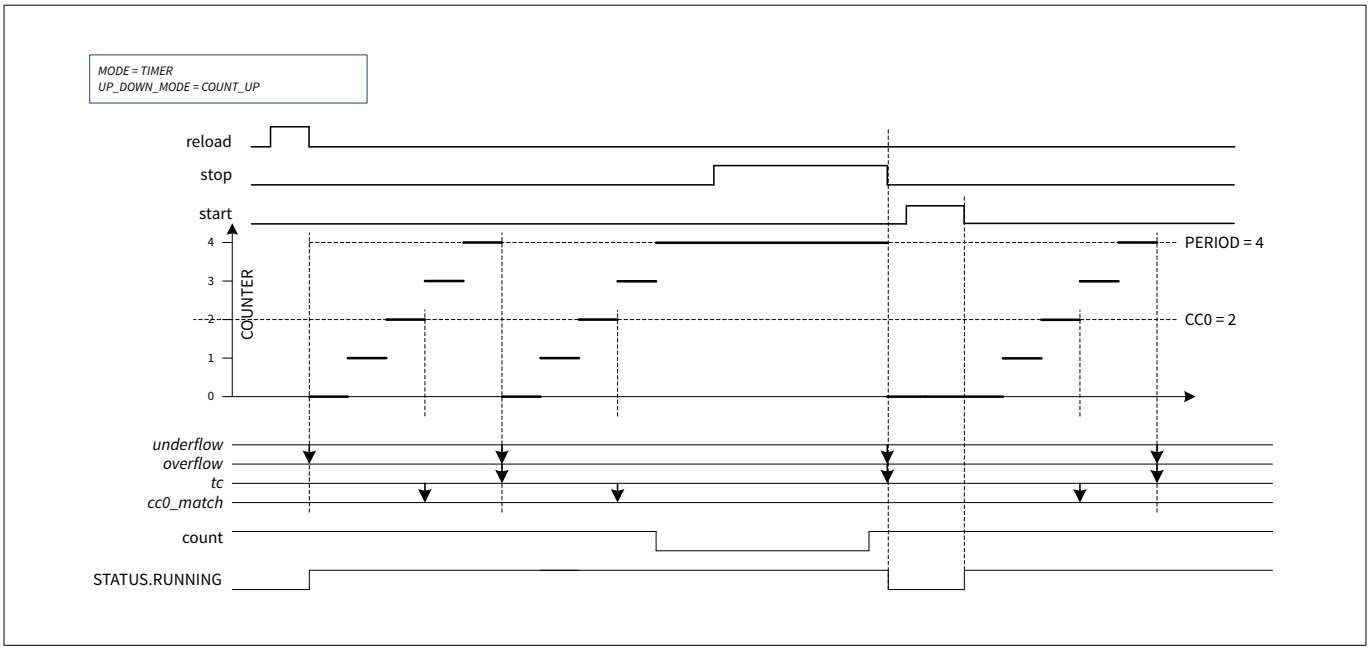


図 108　　カウントイベントの機能

24.3.1.1　タイマーモードのカウンターの設定方法

カウンターをタイマー動作モードに設定する手順と、影響を受けるレジスタビットは以下のとおりです。

- CTRL レジスタの ENABLE ビットに'0'を書き込んでカウンターを無効にします。
- CTRL レジスタの MODE[26:24]フィールドに'000'を書き込んでタイマー モードを選択してください。
- PERIOD レジスタに必要な 16 ビットまたは 32 ビットの周期を設定してください。
- CC0 レジスタに 16 ビットまたは 32 ビットのコンペア値、CC0_BUFF レジスタにバッファ コンペア値を設定してください。
- 必要に応じて、CC 条件ごとに値を切り替えるために、CTRL レジスタの AUTO_RELOAD_CC0 フィールドを設定してください。
- PS レジスタの PS_DIV[2:0]フィールドへの書き込みによりクロックプリスケーリングを設定してください。
- CTRL レジスタの UP_DOWN_MODE[17:16]フィールドにカウント方向を設定してください。
- タイマーは、CTRL レジスタの ONE_SHOT[18]フィールドにそれぞれ 0 または 1 を書き込むことで、連続モードまたはワンショットモードのいずれかで動作するように設定できます。
- TR_IN_SEL0 または TR_IN_SEL1 レジスタを設定して、イベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーを選択します。
- TR_IN_EDGE_SEL レジスタを設定して、イベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーのエッジを選択します。
- 必要に応じて、TC または CC0_MATCH または CC1_MATCH 条件時に割込みを設定します。
- CTRL レジスタの ENABLED ビットに'1'を書き込んでカウンターを有効にします。ハードウェア リロード信号が有効になっていない場合、カウンターを開始するためには、ファームウェア (PERI0_TR_CMD レジスタの RELOAD ビット) をとおしてリロード トリガーを与える必要があります。

24.3.2　キャプチャモード

キャプチャ機能は、0 から PERIOD の間でカウンターをインクリメント/デクリメントします。カウント中、capture0 イベントが観測されます。capture0 イベントがアクティブになると、カウンター値 COUNTER が CC0 にコピーされます (CC0 は CC0_BUFF にコピーされる)。STATUS.CC0_READ_MISS ビットは、CC0 と CC0_BUFF のデータがソフトウ

ェアによって読み出される前に上書きされているかどうかを示します。ソフトウェアが CC0 と CC0_BUFF を読み出す前に、過去のキャプチャイベントが COUNTER からの新しいデータで上書きされた場合、STATUS.CC0_READ_MISS は'1'に設定されます。STATUS.CCO_READ_MISS は、CC0 または CC0_BUFF のいずれかに読み出し操作があると、'0'にクリアされます。

capture1 イベントと STATUS.CC1_READ_MISS ビットが存在します。その場合、capture1 イベントがアクティブになると、カウンター値 COUNTER が CC1 にコピーされます (CC1 は CC1_BUFF にコピーされる)。

キャプチャ機能は、パルスの幅を測定するために使用できます (入力トリガーの 1 つとして接続され、Capture0 または Capture1 イベントとして使用されます)。

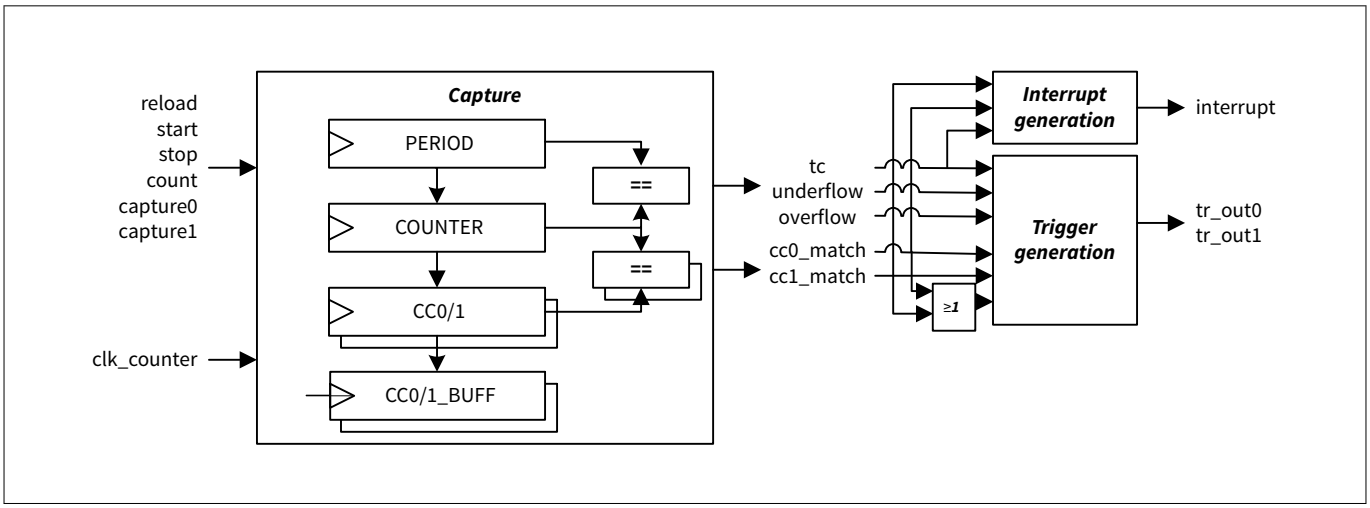


図 109　　キャプチャモード機能図

表 173　　キャプチャモードトリガー入力の説明

生成イベント	使用法
Reload	カウンターを初期化し、スタートさせます。**カウンターの作動中または非作動中に使用できます。** 初期化は UP_DOWN_MODE に依存します。 • COUNT_UP: カウンターは"0"に初期化されます • COUNT_DOWN: カウンターは PERIOD に初期化されます。 COUNT_UPDN1/2: カウンターは"1"に初期化されます
Start	カウンターを開始します (カウンターは HW によって初期化されません)。 カウンターが動作中または動作していないときに使用できます。カウンターが作動しているときは、スタートイベントは何の効果もありません。
ストップ	カウンターを止めます。
カウント	カウント イベントは、アップ/ダウン モードの設定に従ってカウンターをインクリメント/デクリメントします。
Capture0	カウンターの値を CC0 にコピーし、CC0 を CC0_BUFF にコピーします。
Capture1	カウンターの値を CC1 にコピーし、CC1 を CC1_BUFF にコピーします。

表 174 キャプチャのサポート機能

サポートされる機能	説明
クロック プリスケーリング	カウンター クロック"clk_counter"をプリ スケーリングします。
One shot	tc イベントで HW がカウンターを停止。 COUNT_UPDN2 では、アンダーフロー時に tc イベントでカウンターを停止します。
アップ/ダウン モード	UP_DOWN_MODE によって指定されます。 • COUNT_UP: カウンターが 0 から PERIOD までカウントします。 • COUNT_DOWN: カウンターが PERIOD から 0 までカウントします。 COUNT_UPDN1/2: カウンターが 1 から PERIOD までカウントし、0 に戻ります。

表 175 キャプチャーの内部イベント

内部イベント	説明
cc0_match	CC0 は CC0_BUFF にコピーされ、カウンター値は CC0 にコピーされます (cc0_match は capture0 イベントに等しい)。
cc1_match	CC1 は CC1_BUFF にコピーされ、カウンター値は CC1 にコピーされます (cc1_match は capture1 イベントに等しい)。
アンダーフロー	カウンターがデクリメントされ、COUNTER が"0"の状態から変化します。 COUNT_DOWN、COUNT_UPDN1、または COUNT_UPDN2 モードで、Reload イベントがアンダーフローを生成します。
オーバーフロー	カウンターはインクリメントし、COUNTER が PERIOD と等しい状態から変化します。 COUNT_UP モードで Reload イベントがオーバーフローを発生させます。
tc	UP_DOWN_MODE によって指定されます。 • COUNT_UP: tc イベントはオーバーフローイベントと同じです • COUNT_DOWN: tc イベントはアンダーフローイベントと同じです • COUNT_UPDN1: tc イベントはアンダーフロー イベントと同じです。 • COUNT_UPDN2: tc イベントは、オーバーフローイベントとアンダーフローイベントの論理和と同じです。 Reload はアンダーフロー/オーバーフローを発生させるが、tc は発生させません。

表 176　　キャプチャ モード ライン出力

PWM 出力	説明
line_out	未使用
line_compl_out	未使用

CC0/CC0_BUFF に COUNTER 値を取り込む

capture0 イベントが検出されると、COUNTER の値は CC0 レジスタにコピーされ、CC0 レジスタの値は CC0_BUFF レジスタにコピーされます。この機能は、パルスの幅を測定するために使用できます (入力トリガーの 1 つとして接続され、立ち上り/立ち下りエッジ検出モードで Capture0 イベントとして使用されます)。次の図に、COUNT_UP モードでのキャプチャ動作を示します。

注:

- Capture0 イベント検出は、立ち上りエッジ検出を使用します。その結果、Capture0 のイベントは、次の"アクティブ カウント"プリスケーリング カウンター クロックまで保持されます。
- capture0 イベントが発生すると、COUNTER は CC0 にコピーされます。CC0 が CC0_BUFF にコピーされます
- カウンター値がキャプチャされると cc0_match イベントが発生します。
- CC0_READ_MISS は、ソフトウェアが 3 番目のキャプチャ イベントの前に一度も読み出しを行わなかった場合、'1'に設定されます。ソフトウェアが CC0 または CC0_BUFF のいずれかを読み出すと、'0'にクリアされます。

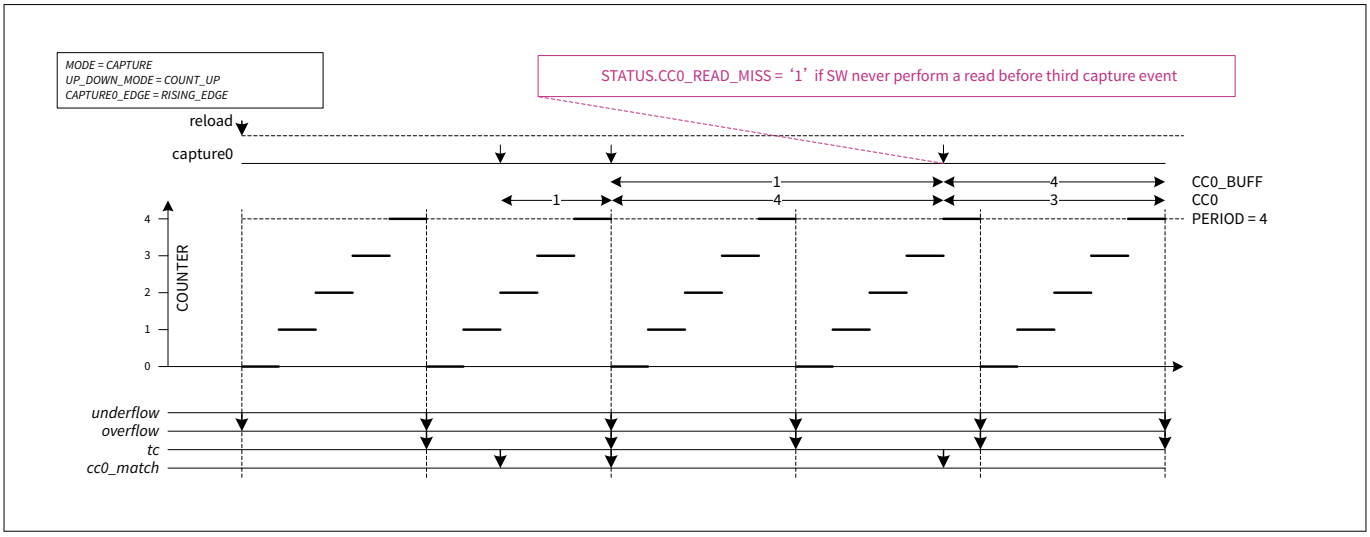


図 110　　CC0/CC0_BUFF が capture0 イベントによって更新

次の"アクティブ カウント"プリスケール カウンター クロックの前に複数のキャプチャ イベントが検出された場合、キャプチャ イベントは以下のように扱われます。

- 立ち上りエッジまたは立ち下りエッジ検出モードでは、複数のイベントが効果的に 1 つのイベントに縮小されます。
- 立ち上り/立ち下りの両エッジ検出モードでは、偶数個のイベントは検出されず、奇数個のイベントは 1 つのイベントに縮小されます

上記の動作は、4 倍のプリスケーラーを使用した以下の図によって説明されます。最初の図は、立ち上りエッジ検出を使用しています。2 つ目の図は、立ち上り/立ち下りのエッジ検出を使用しています。

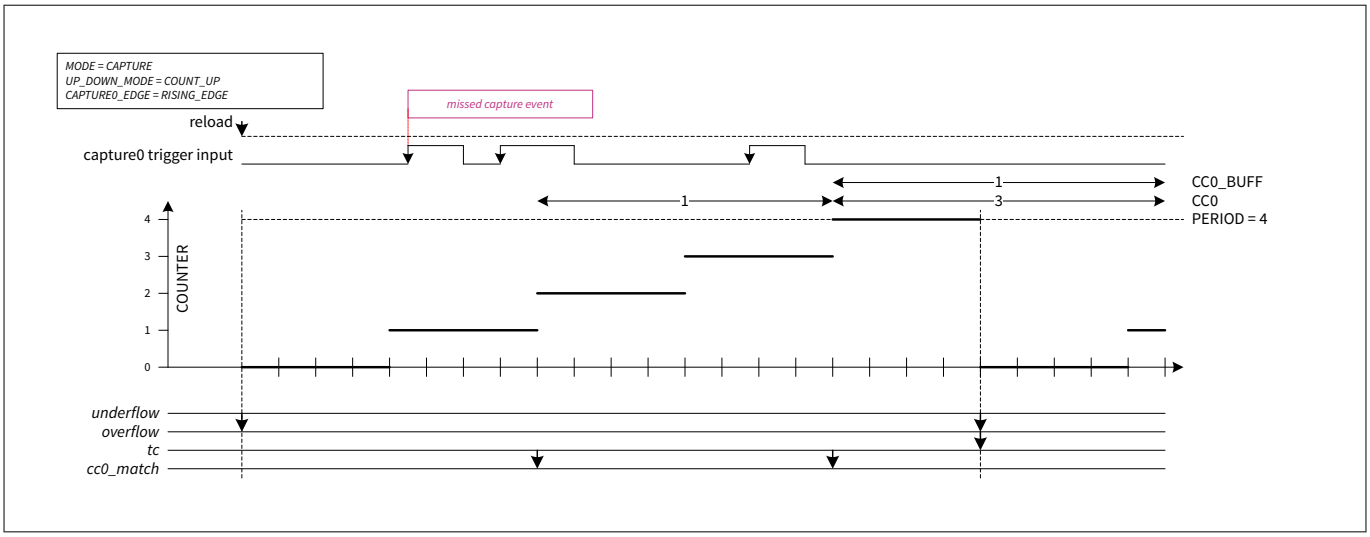


図 111　　複数の立ち上りエッジのキャプチャ

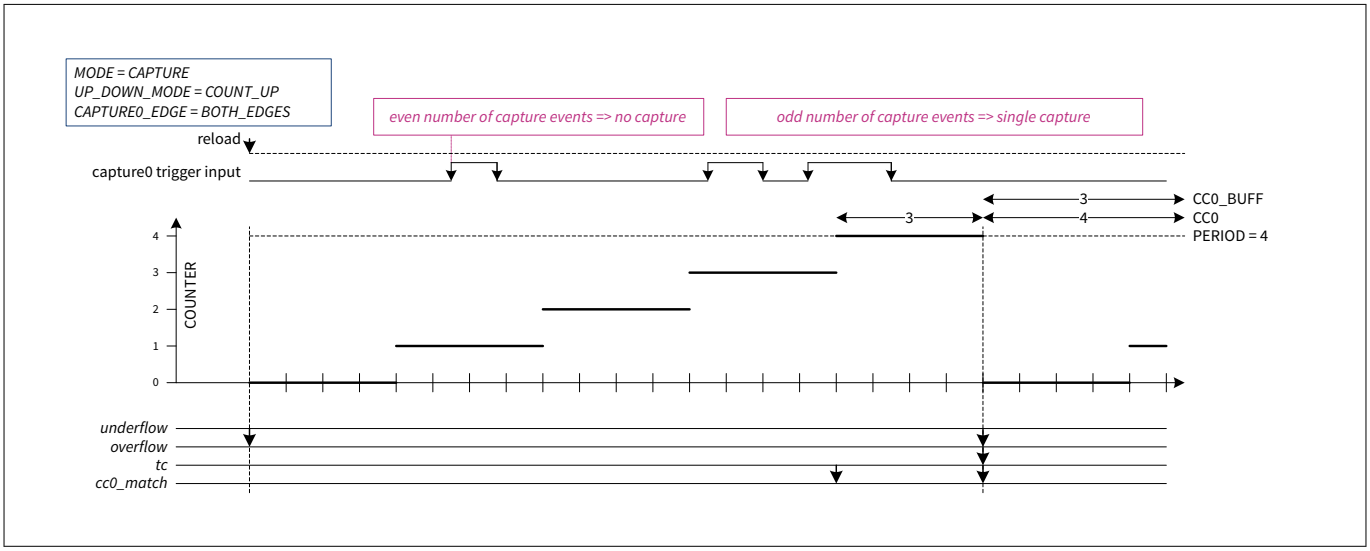


図 112　　複数の立ち上り/立ち下りの両エッジ キャプチャ

24.3.2.1　キャプチャモードのカウンターの設定方法

キャプチャ モード動作用にカウンターを設定する手順と、影響を受けるレジスタビットは以下のとおりです。

- CTRL レジスタの ENABLE ビットに'0'を書き込んでカウンターを無効にします。
- CTRL レジスタの MODE[26:24]フィールドに'010'を書き込んでキャプチャ モードを選択します。
- PERIOD レジスタに必要な 16 ビットまたは 32 ビット周期を設定してください。
- PS レジスタの PS_DIV[2:0]フィールドへの書き込みによりクロックプリスケーリングを設定してください。
- CTRL レジスタの UP_DOWN_MODE[17:16]フィールドにカウント方向を設定してください。
- カウンターは、CTRL レジスタの ONE_SHOT[18]フィールドにそれぞれ 0 または 1 を書き込むことで、連続モードまたはワンショットモードのいずれかで動作するように設定できます。
- TR_IN_SEL0 または TR_IN_SEL1 レジスタを設定して、イベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーを選択します。
- TCPWM_CNT_TR_CTRL0 レジスタを設定してイベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーを選択してください。

- 必要に応じて、TC または CC0_MATCH または CC1_MATCH 条件時に割込みを設定します。
- Enable the counter by writing '1' to the ENABLED bit in the CTRL register. ハードウェアのリロード信号が有効でない場合、カウンターを開始するためにはファームウェア経由で Reload トリガー (PERI0_TR_CMD レジスタの RELOAD ビット) を与える必要があります。

24.3.3 パルス幅変調 (PWM) モード

PWM 機能は 0 と PERIOD の間でカウンターをインクリメント/デクリメントします。カウンターが動作しているとき、カウンター値 COUNTER は CC0 (CC1) と比較されます。COUNTER が CC0 (CC1) に等しいとき、cc0_match (cc1_match) イベントが発生します。さらに、カウンターのオーバーフローとアンダーフローでは、オーバーフローとアンダーフローのイベントが発生します。cc0_match、cc1_match、overflow、underflow の各イベントを組み合わせて、PWM LINE_OUT と LINE_COMPL_OUT 出力信号にパルス幅変調信号を生成します。左揃え、右揃え、中央揃えの PWM 信号を生成できます。非対称 PWM 信号は COUNT_UPDN2 モードを使用して生成できます。現在の PWM 出力レベルを読み出せます。デューティ サイクルが 0%または 100%という特殊なケースもサポートされています。PERIOD_BUFF レジスタはデューティ サイクルの更新に使用され、TC イベントによってアクティブになります。

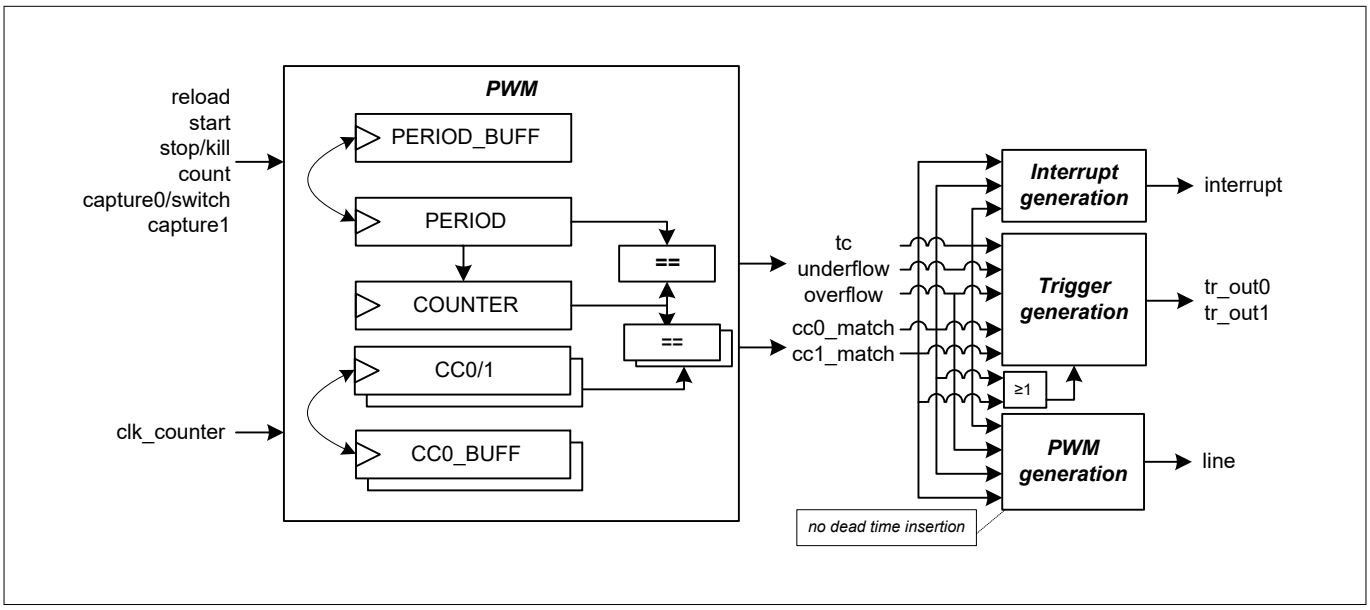


図 113 PWM モード機能

表 177 PWM モードトリガー入力の説明

生成イベント	使用法
リロード	カウンターを初期化し、スタートさせます。**カウンターが動作していないときにのみ使用してください。**初期化は UP_DOWN_MODE に依存します。 • COUNT_UP: カウンターは"0"に初期化されます • COUNT_DOWN: カウンターは PERIOD に初期化されます COUNT_UPDN1/2: カウンターは"1"に初期化されます

(続く)

表 177 (続き) PWM モードトリガー入力の説明

生成イベント	使用法
start	カウンターを開始します (カウンターは HW によって初期化されません)。 カウンターが動作中または動作していないときに使用できます。カウンターが作動しているときは、スタートイベントは何の効果もありません。
ストップ	カウンターを止めます。さまざまなストップ/キルモードが存在します。
カウント	カウント イベントは、アップ/ダウン モードの設定に従ってカウンターをインクリメント/デクリメントします。
Capture0	このイベントはスイッチイベントとして機能します。CTRL.AUTO_RELOAD_CC0/1, CTRL.AUTO_RELOAD_PERIOD, CTRL.AUTO_RELOAD_LINE_SEL で指定された場合) CTRL.SWAP_ENABLE = "1"の時、このイベントがアクティブになると、tc イベントで CC0/CC0_BUFF, CC1/CC1_BUFF, PERIOD/PERIOD_BUFF, LINE_SEL/LINE_SEL_BUFF レジスタが交換されます。 このイベントはスイッチイベントとして機能します。CTRL.AUTO_RELOAD_CC0/1、CTRL.AUTO_RELOAD_PERIOD、CTRL.AUTO_RELOAD_LINE_SEL で指定された場合) CTRL.SWAP_ENABLE = "0"の時、このイベントがアクティブになると CC0_BUFF は CC0 へ、CC1_BUFF は CC1 へ、PERIOD_BUFF と LINE_SEL_BUFF レジスタは PERIOD と LINE_SEL へ tc イベント (CTRL.AUTO_RELOAD_CC0/1、CTRL.AUTO_RELOAD_PERIOD、CTRL.AUTO_RELOAD_LINE_SEL で指定された場合) で転送されます。 切り替えイベントが検出されると次の tc イベントまで維持されます。**スイッチ イベントには、立ち上り、立ち下り、または立ち上り/立ち下りのエッジ モードが必要です。パス スルー モードは、選択されたイベントが定数'0'または'1'でない限り、サポートされていません。カウンターが動作していないときに検出された場合、スイッチ イベントはクリアされ、何の効果もありません。**

(続く)

表 177 (続き) PWM モードトリガー入力の説明

生成イベント	使用法
Capture1 (stop1/kill1)	このイベントは、第 2 のストップ/キルイベントとして機能します。stop0/kill0 イベントと同じ機能があります。両イベントは統合されます。 **注:** PWM に対して 2 つのストップ/キルイベントを持つことは、1 つのストップ/キルイベントに対して PERI トリガー マルチプレクサからの共通トリガー (複数の PWM の同期ストップ/キル動作を可能にする) を選択する一方で、もう 1 つのストップ/キルイベントに対して例えば専用の ADC 範囲外トリガー (例えば、PWM 駆動信号の電流が範囲外の場合に PWM のリアルタイムのハードウェア停止を可能にする) を選択することを可能にします。

表 178 PWM モード対応機能

サポートされる機能	説明
クロック プリスケーリング	カウンター クロック"clk_counter"をプリ スケーリングします。
One shot	tc イベントで HW がカウンターを停止。 COUNT_UPDN2 では、アンダーフロー時に tc イベントでカウンターを停止します。
オートリロード CC	CC0 と CC0_BUFF は、スイッチイベントと tc イベント (CTRL.AUTO_RELOAD_CC0 で指定された場合) で交換されます。 CC1 と CC1_BUFF は、スイッチイベントと tc イベント (CTRL.AUTO_RELOAD_CC1 で指定された場合) で交換されます。
オートリロード PERIOD	PERIOD と PERIOD_BUFF は、switch イベントと tc イベント (CTRL.AUTO_RELOAD_PERIOD で指定された場合) で交換されます。注: COUNT_UPDN2 モードがオーバーフローイベントと一致する tc イベントで PERIOD と PERIOD_BUFF を交換する場合、ソフトウェアは PERIOD と PERIOD_BUFF の値が同じことを確認する必要があります。
オートリロード LINE_SEL	LINE_SEL と LINE_SEL_BUFF は、CTRL.SWAP_ENABLE = "1"の場合、スイッチイベントと tc イベント (CTRL.AUTO_RELOAD_LINE_SEL で指定された場合) で交換されます。 LINE_SEL_BUFF は、CTRL.SWAP_ENABLE = "0"のとき、スイッチイベントおよび tc イベント (CTRL.AUTO_RELOAD_LINE_SEL で指定された場合) で LINE_SEL に転送されます。

(続く)

表 178 (続き) PWM モード対応機能

サポートされる機能	説明
アップ/ダウン モード	UP_DOWN_MODE によって指定されます。 • COUNT_UP: カウンターが 0 から PERIOD までカウントします。 • COUNT_DOWN: カウンターが PERIOD から 0 までカウントします。 • COUNT_UPDN1/2: カウンターが 1 から PERIOD までカウントし、0 に戻ります
Kill Mode (キル モード)	PWM_IMM_KILL, PWM_SYNC_KILL, PWM_STOP_ON_KILL, PWM_TC_SYN_KILL_DT, PWM_SYN_KILL_DT で指定されます。詳細は、以下のセクションを参照してください。

注: PWM モードはデッドタイムの挿入をサポートしません。この機能は、独立した PWM_DT モードによってサポートされています。

表 179 PWM モードトリガー出力の説明

トリガー出力	説明
cc0_match	UP_DOWN_MODE によって指定されます。 • COUNT_UP と COUNT_DOWN: カウンターが COUNTER と CC0 が等しい状態に変化します • COUNT_UPDN1/2: カウンターが COUNTER と CC0 が等しい状態から変化します。CC0_MATCH_UP_EN/CC0_MATCH_DOWN_EN は、カウントがアップ/ダウン カウントの時に cc0_match 生成を有効/無効にします。(CC0 = 0 と CC0=PERIOD は、0/100%デューティサイクルの PWM 生成のための特別なケースになります。
cc1_match	UP_DOWN_MODE によって指定されます。 • COUNT_UP と COUNT_DOWN: カウンターが COUNTER と CC1 が等しい状態に変化します • COUNT_UPDN1/2: カウンターが COUNTER と CC1 が等しい状態から変化します。CC1_MATCH_UP_EN/CC1_MATCH_DOWN_EN は、カウントがアップ/ダウンカウントのときに、cc1_match の生成を有効/無効にします。
アンダーフロー	カウンターがデクリメントされ、COUNTER が"0"の状態から変化します。 COUNT_DOWN、COUNT_UPDN1、または COUNT_UPDN2 モードで、Reload イベントがアンダーフローを生成します。

(続く)

表 179 (続き) PWM モードトリガー出力の説明

トリガー出力	説明
オーバーフロー	カウンターはインクリメントし、COUNTER が PERIOD と等しい状態から変化します。 COUNT_UP モードで Reload イベントがオーバーフローを発生させます。
tc	UP_DOWN_MODE によって指定されます。 • COUNT_UP: tc イベントはオーバーフローイベントと同じです • COUNT_DOWN: tc イベントはアンダーフローイベントと同じです • COUNT_UPDN1: tc イベントはアンダーフロー イベントと同じです。 • COUNT_UPDN2: tc イベントは、オーバーフローイベントとアンダーフローイベントの論理和と同じです。 Reload はアンダーフロー/オーバーフローを発生させるが、tc は発生させません。

表 180 PWM モード内部イベントの説明

PWM 出力	説明
line_out	PWM ライン出力
line_compl_out	コンプリメンタリ PWM ライン出力。

図 114 に、"line_out"と"line_compl_out"の生成過程を示します。一般に、"line_out"出力は PWM 信号"line"の状態を反映し、"line_compl_out"出力は"line"の反転バージョンを反映します。ラインの挙動は、さまざまな機能モードに依存します。それらは、「ライン選択」、「デッドタイム挿入」、「キルファンクション」、「ライン極性」、「キルライン極性」です。

PWM 出力信号は、アンダーフロー、オーバーフロー、cc_match イベントを使用して生成されます。これらのイベントは、それぞれ個別に INVERT、SET、または CLEAR pwm_dt_input に設定できます。

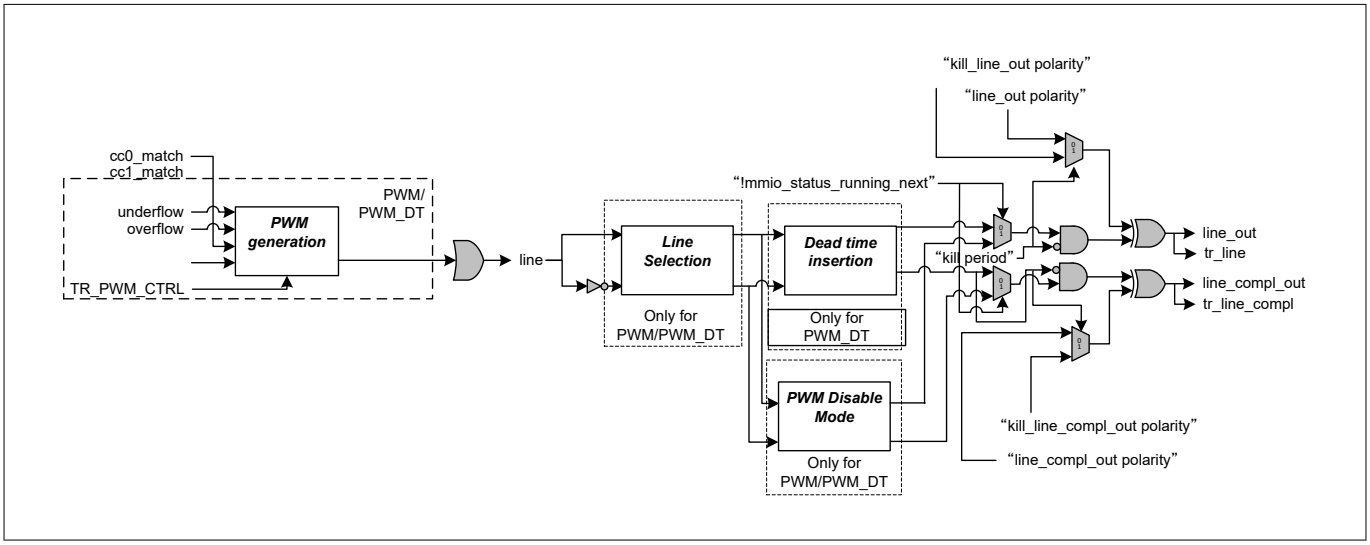


図 114 PWM 出力生成

注: COUNT_UP および COUNT_DOWN モードで発生する cc0_match イベントは、他の TIMER/CAPTURE モードでの発生とは異なります。これは、0 %と 100 %のデューティ サイクルを確実に発生させるためです。

PWM 動作は PERIOD と CC0 レジスタに依存します。ソフトウェアは、PWM の動作に影響を与えることなく、PERIOD_BUFF と CC0_BUFF レジスタを更新できます。switch/capture イベントは、コンペアレジスタとバッファ コンペアレジスタの値を切り替えるために使用できます。また、CTRL.SWAP_ENABLE = "1"の時、Period レジスタとバッファ Period レジスタの値を切り替えます。CTRL.SWAP_ENABLE="0"の時、スイッチ/キャプチャイベントはバッファ コンペアレジスタの値をコンペアレジスタに転送するために使用できます。また、バッファ Period レジスタの値を Period レジスタに転送します。

以下の表は、PWM モードでサポートされるキルモードをまとめたものです。以下の表に定義されていない設定は、サポートされていないか、または推奨されていません。

表 181 PWM のキルモード

Kill Mode (キル モード)	設定	キル動作
No-IMM-Async	PWM_IMM_KILL = 0; PWM_SYNC_KILL = 0; PWM_TC_SYNC_KILL_DT = 0; PWM_SYNC_KILL_DT = 0; PWM_STOP_ON_KILL = 0; STOP_EDGE = NO_EDGE_DET	• キル入力がアクティブになった後、次のアクティブ カウント クロックで PWM 出力が抑制されます。 • PWM 出力抑制は、キル入力が非アクティブになった次のアクティブ カウント クロックで解除されます
IMM-Async	PWM_IMM_KILL = 1; PWM_SYNC_KILL = 0; PWM_TC_SYNC_KILL_DT = 0; PWM_SYNC_KILL_DT = 0; PWM_STOP_ON_KILL = 0; STOP_EDGE = NO_EDGE_DET	• キル入力がアクティブになった直後に PWM 出力が抑制されます • PWM 出力抑制は、キル入力が非アクティブになった次のアクティブ カウント クロックで解除されます
No-IMM-Sync	PWM_IMM_KILL = 0; PWM_SYNC_KILL = 1; PWM_TC_SYNC_KILL_DT = 0; PWM_SYNC_KILL_DT = 0; PWM_STOP_ON_KILL = 0; STOP_EDGE = 立ち上り;	• キル入力がアクティブになった後、次のアクティブ カウント クロックで PWM 出力が抑制されます。 • キル入力が非アクティブになった後、次の tc イベントで PWM 出力抑制が解除されます
IMM-Sync	PWM_IMM_KILL = 1; PWM_SYNC_KILL = 1; PWM_TC_SYNC_KILL_DT = 0; PWM_SYNC_KILL_DT = 0; PWM_STOP_ON_KILL = 0; STOP_EDGE = 立ち上り;	• キル入力がアクティブになった直後に PWM 出力が抑制されます • キル入力が非アクティブになった後、次の tc イベントで PWM 出力抑制が解除されます

(続く)

表 181 (続き) PWM のキルモード

Kill Mode (キル モード)	設定	キル動作
TC-Sync-DT	PWM_IMM_KILL = 0; PWM_SYNC_KILL = 0; PWM_TC_SYNC_KILL_DT = 1 PWM_SYNC_KILL_DT = 0; PWM_STOP_ON_KILL = 0; STOP_EDGE = 立ち上り;	• キル入力がアクティブになった後、次の tc イベントで PWM 出力が抑制されます。デッドタイムが有効な場合、tc イベント＋デッドタイムの後、出力は抑制されます。 • キル入力が非アクティブになった後、次の tc イベントで PWM 出力抑制が解除されます • PWM_DT モードでのみサポートされます。
IMM-Sync-DT	PWM_IMM_KILL = 1; PWM_SYNC_KILL = 0; PWM_TC_SYNC_KILL_DT = 0; PWM_SYNC_KILL_DT = 1; PWM_STOP_ON_KILL = 0; STOP_EDGE = 立ち上り;	• PWM 出力の line_out は即座に抑制されるのに対し、line_compl_out はキル入力がアクティブの場合、デッドタイム後に抑制されます。 • キル入力が非アクティブになった後、次の tc イベントで PWM 出力抑制が解除されます • PWM_DT モードでのみサポートされます。
No-IMM-Stop	PWM_IMM_KILL = 0; PWM_SYNC_KILL = Don’t care; PWM_TC_SYNC_KILL_DT= Don’t care; PWM_SYNC_KILL_DT = Don’t care; PWM_STOP_ON_KILL = 1; STOP_EDGE = RISING_EDGE/ FALLING_EDGE/BOTH_EDGES;	• キル入力がアクティブになった後、次のアクティブ カウント クロックで PWM 出力が抑制されます。 • キル入力が非アクティブの後、カウンターがリスタートすると PWM 出力の抑制が解除されます
IMM-Stop	PWM_IMM_KILL = 1; PWM_SYNC_KILL = Don’t care; PWM_TC_SYNC_KILL_DT = Don’t care; PWM_SYNC_KILL_DT = Don’t care PWM_STOP_ON_KILL = 1; STOP_EDGE = RISING_EDGE/ FALLING_EDGE/BOTH_EDGES;	• キル入力がアクティブになった直後に PWM 出力が抑制されます • キル入力が非アクティブの後、カウンターがリスタートすると PWM 出力の抑制が解除されます

24.3.3.1 PWM モード機能

注: ワンショットモードとクロックプリスケーリングはタイマーモードと同じです。

カウントアップ/ダウン モード

アップ/ダウンカウントモードは、カウンターが動作している間、カウント方向 (インクリメントまたはデクリメント) を制御します。

図 115 に COUNT_UP モードの PWM を示します。カウンターは (0 に) 初期化され、ソフトウェアベースのリロードイベントで開始されます。

注:

- カウンターが 4 の状態から変化すると、オーバーフローと tc イベントが発生します。
- カウンターが COUNTER が 2 と等しい状態に変化すると、cc0_match イベントが発生します。
- PERIOD は 4 で、4+1 = 5 クロック周期の有効な繰り返しカウンター パターンとなります。

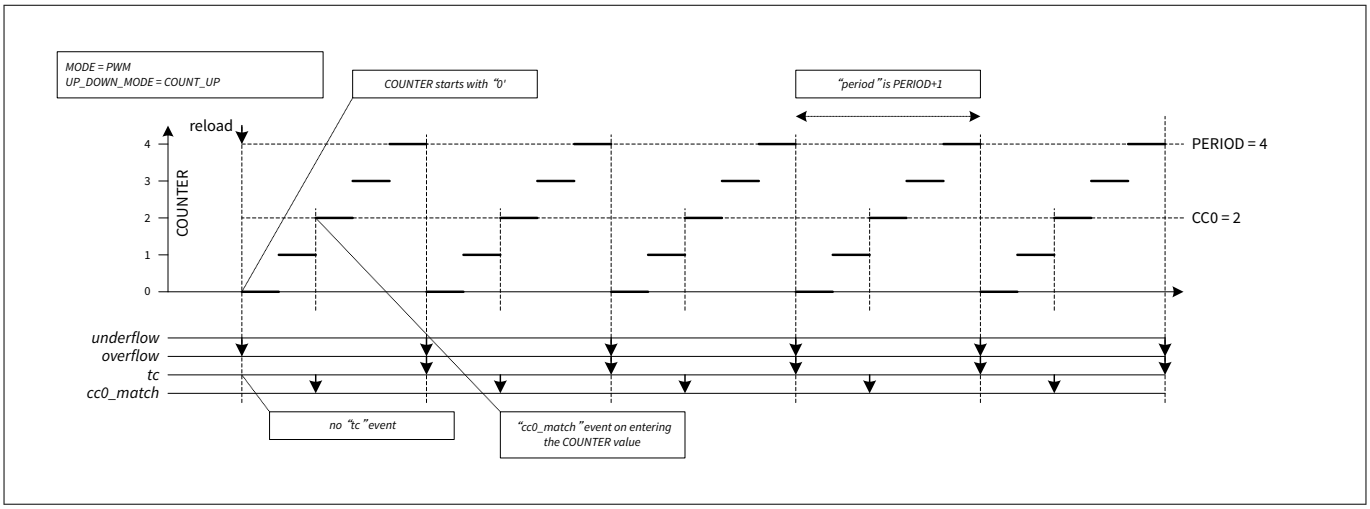


図 115 カウントアップモードの PWM モード

図 116 に、ダウンカウントモードの PWM を示します。カウンターは (PERIOD に) 初期化され、ソフトウェアベースのリロード イベントで開始されます。

注:

- COUNTER が 0 の状態からカウンターが変化すると、アンダーフローと tc イベントが発生します。
- カウンターが COUNTER が 2 と等しい状態に変化すると、cc0_match イベントが発生します。
- PERIOD は 4 で、4+1 = 5 クロック周期の有効な繰り返しカウンター パターンとなります。

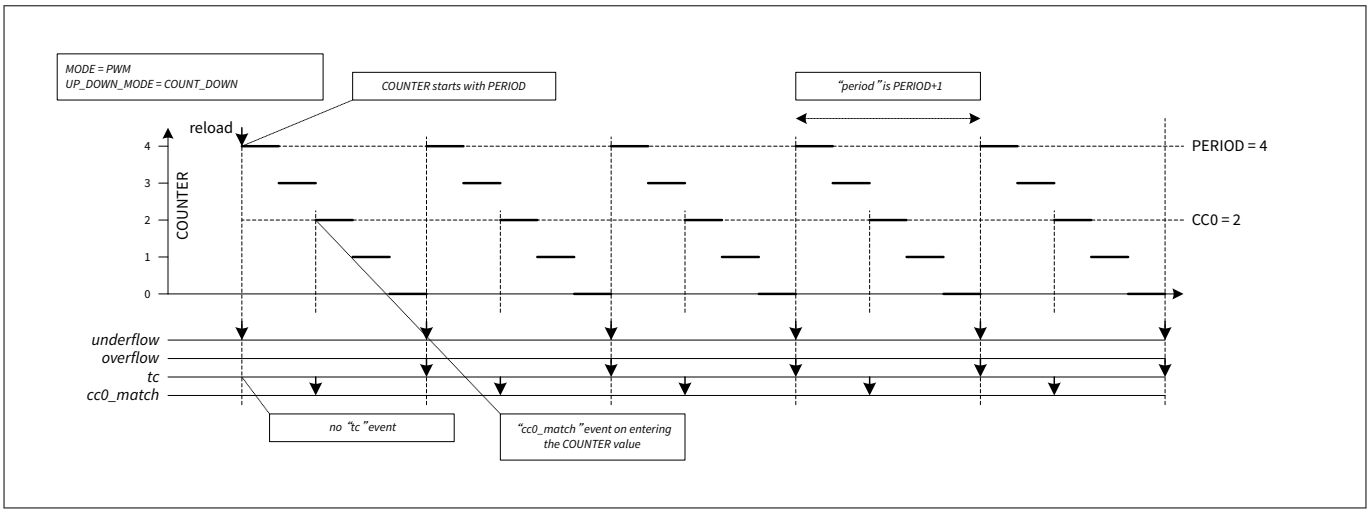


図 116 ダウンカウントモードの PWM

図 117 に、アップ/ダウン カウント モードの PWM を示します。カウンターは (1 に) 初期化され、ソフトウェアベースのリロードイベントで開始されます。

注:

- カウンターが 4 の状態から変化すると、オーバーフローが発生します。
- COUNTER が 0 の状態からカウンターが変化すると、アンダーフローと TC イベントが発生します。
- カウンターが COUNTER = 2 の状態から変化すると、cc0_match イベントが発生します。
  リロードイベント前の実際のカウンター値 COUNTER は使用されないことに注意してください。
  リロード前のカウンター値は 0 とみなされます。
- PERIOD は 4 で、2*4 = 8 クロック周期の有効な繰り返しカウンター パターンとなります。

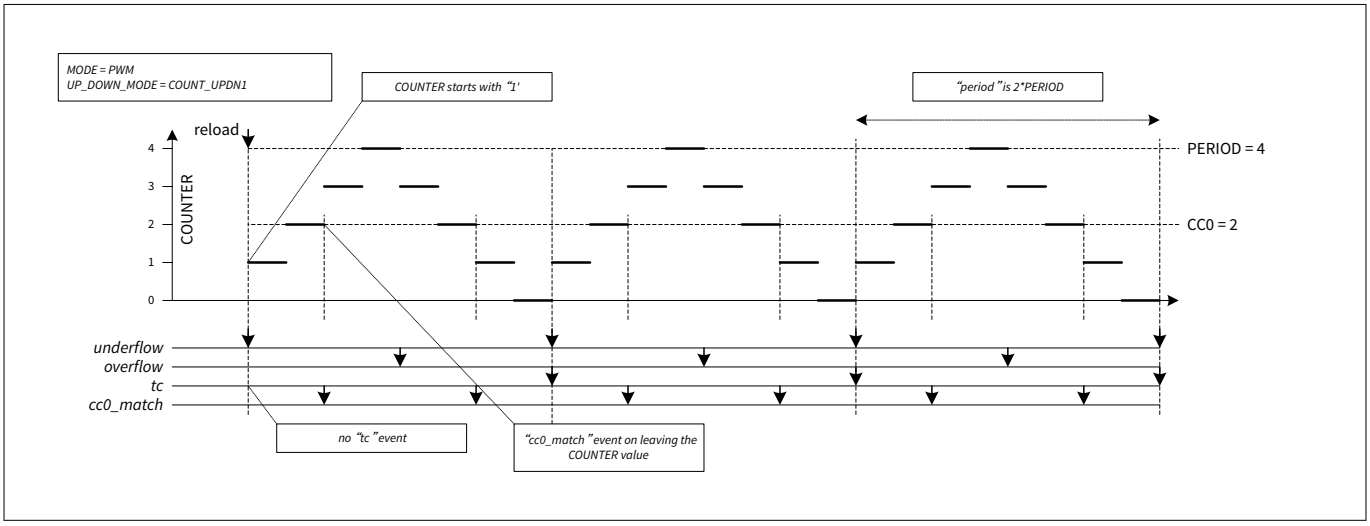


図 117　　アップ/ダウン カウント モードの PWM

スイッチ イベントで CC0 と PERIOD が自動リロード

自動 CC リロードと自動 PERIOD リロードは、ダイナミックな PWM デューティ サイクル変更とカウント周期長変更を提供します。スイッチ アクティビティには、アクティブ スイッチ イベント (capture0) が必要です。

- TC イベントにおいて、スイッチイベントがアクティブで CTRL レジスタの AUTO_RELOAD_CC ビットが'1'に設定されている場合、CC0/1 と CC0/1_BUFF は値を交換します。
- TC イベントにおいて、スイッチイベントがアクティブで CTRL レジスタの AUTO_RELOAD_PERIOD ビットが'1'に設定されている場合、PERIOD と PERIOD_BUFF は値を交換します。
- CC1 と CC1_BUFF は、capture0 がアクティブであれば、TC イベントで値を交換します。

次の図は、COUNT_UP モードと COUNT_DOWN モードの期間値の更新を示し、各スイッチ イベント後に異なる周期時間が発生します。

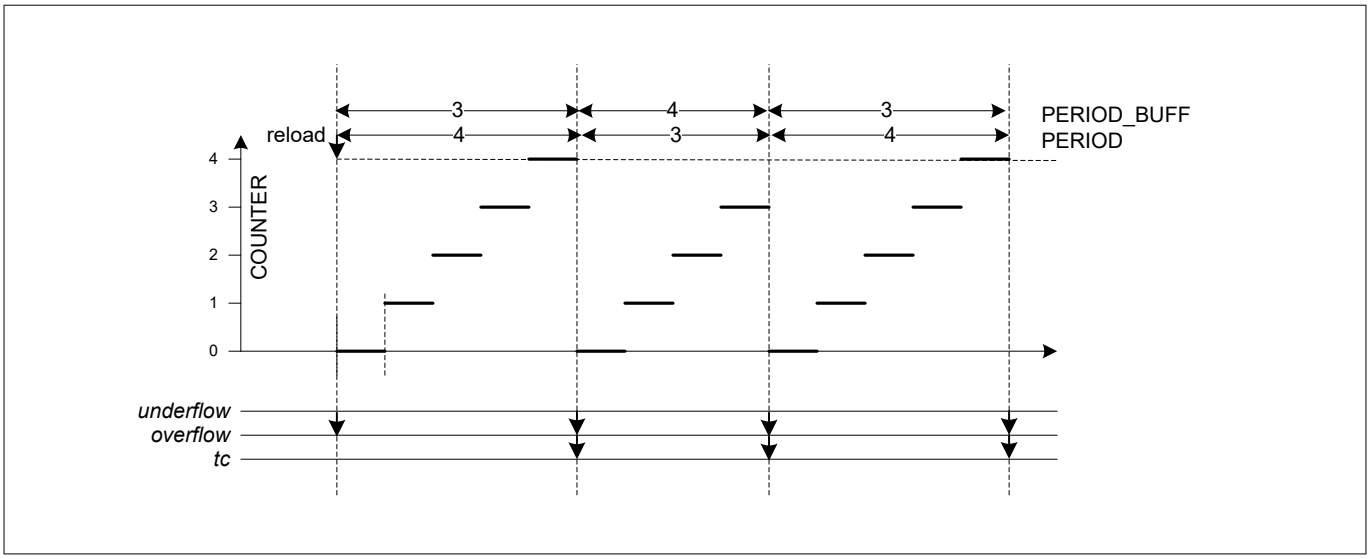


図 118　　スイッチイベントによる COUNT_UP モードでの PERIOD/PERIOD_BUFF 交換

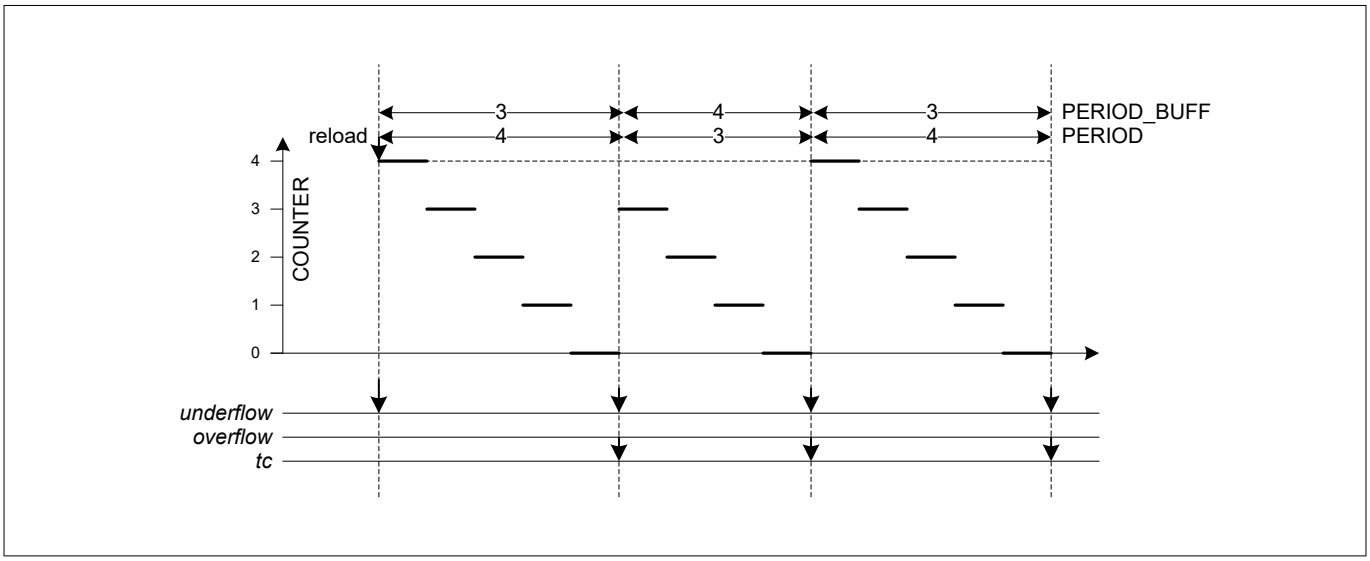


図 119　　スイッチイベントによる COUNT_DOWN モードでの PERIOD/PERIOD_BUFF 交換

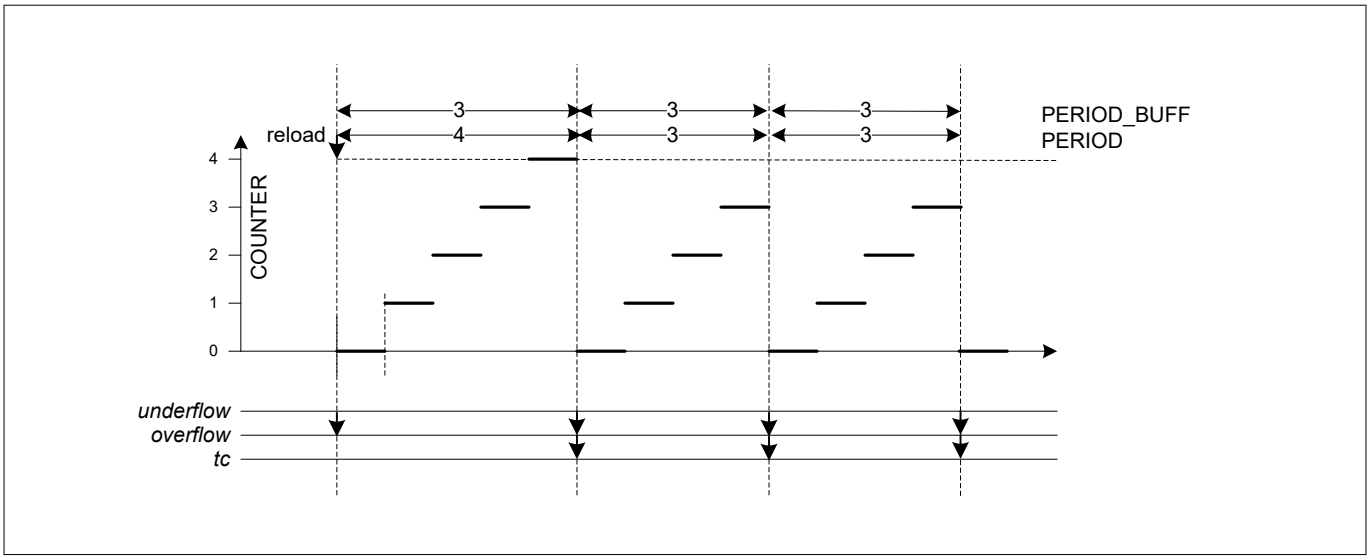


図 120　　COUNT_UP モードでの PERIOD/PERIOD_BUFF 交換 (SWAP_ENABLE="0")

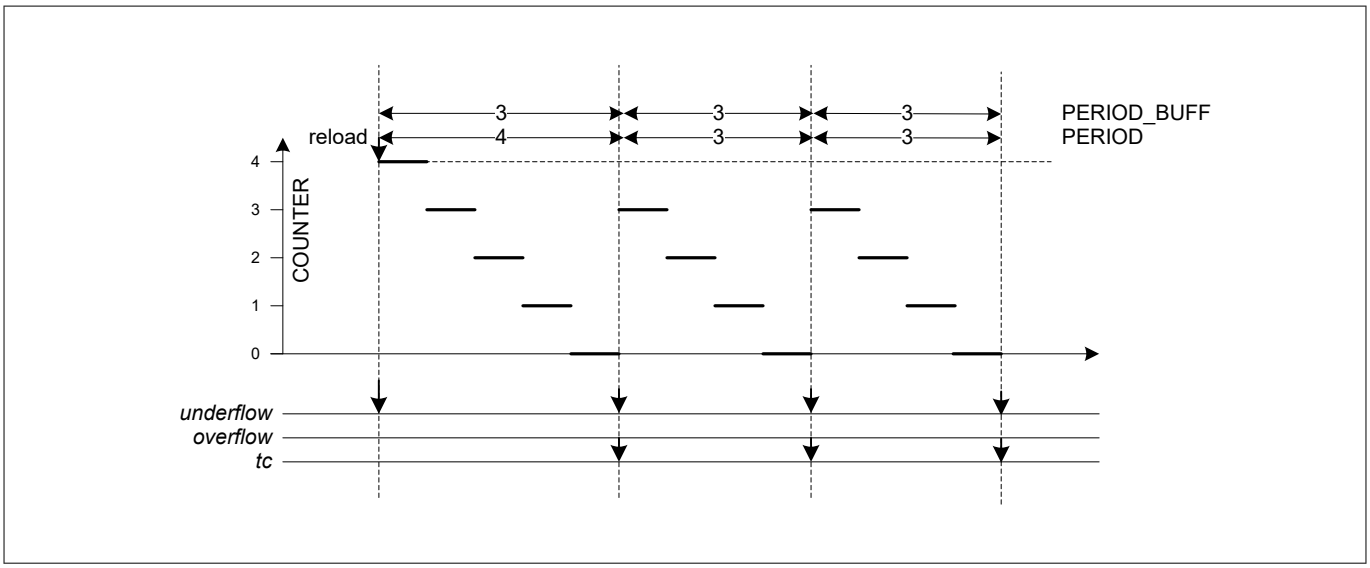


図 121　　COUNT_DOWN モードでの PERIOD/PERIOD_BUFF 交換 (SWAP_ENABLE="0")

保留中のスイッチイベントがアクティブな状態で、次の tc イベントまでにソフトウェアのアップデートが完了しない場合、問題が発生する場合があります。例えば、ソフトウェアが tc イベントの前に PERIOD_BUFF を更新し、tc イベントの後に CC0_BUFF を更新した場合、スイッチングは CC0_BUFF レジスタの更新を反映しません。これを防ぐには、PERIOD_BUFF レジスタと CC0_BUFF レジスタの両方が更新された後、MMIO レジスタの書き込みによってソフトウェアがスイッチ イベントを生成する必要があります。スイッチイベントは、次の tc イベントが発生するまでハードウェアによって保留されます。

前の段落では、ソフトウェアが生成するスイッチ イベントによって、1 つの PWM の CC0/CC0_BUFF と PERIOD/PERIOD_BUFF レジスタの同期更新を取り上げました。モーター制御中、3 つの PWM は一体となって動作し、すべてのピリオド レジスタとコンペア レジスタのペアの更新は同期していなければいけません。3 つの PWM はすべて周期が同期しており、結果として tc イベントも同期しています。3 つの PWM のスイッチ イベントは、3 つの PWM の PERIOD_BUFF と CC0_BUFF レジスタがソフトウェアによって更新された後、同じハードウェア トリガー入力によって生成され、同じ TC で CC0 を同期的に更新します。

CC0/CC0_BUFF オートリロードによる左/右/中央揃え PWM

PWM は、左揃え、右揃え、中央揃えを生成でき、PWM モードでは以下の機能がサポートされています。

- 異なる位相の PWM を生成するためには、アップ/ダウン カウント モードを使用する必要があります。
- ライン状態はアンダーフロー/オーバーフロー/cc0_match/cc1_match 内部イベントごとに変更され、TR_PWM_CTRL レジスタで設定可能

左揃え、右揃え、中央揃えの PWM に必要な設定は以下のとおりです。

- 左揃え:
  - CTRL レジスタの UP_DOWN_MODE [17:16]フィールドに値'0'を書き込んでカウンターの方向を COUNT_UP モードに設定してください。
  - TR_PWM_CTRL レジスタの OVERFLOW_MODE [3:2] フィールドに値'0' (SET) を書き込んで、COUNTER が PERIOD 値に達したら LINE_OUT 信号を'1'に設定してください。
  - TR_PWM_CTRL レジスタの CC0_MATCH_MODE [1:0]フィールドに値'1' (CLEAR) を書き込み、COUNTER が CC0 値と等しい時に LINE_OUT 信号を'0'に設定してください。
- 右揃え:
  - CTRL レジスタの UP_DOWN_MODE [17:16] フィールドに値'1'を書き込んでカウンターの方向を COUNT_DOWN モードに設定。
  - TR_PWM_CTRL レジスタの CC0_MATCH_MODE [1:0]フィールドに値'0' (SET) を書き込んで、COUNTER が CC0 値と等しい時に LINE_OUT 信号を'1'に設定してください。
  - TR_PWM_CTRL レジスタの UNDERFLOW_MODE [5:4]フィールドに値'1' (CLEAR) を書き込んで、COUNTER が'0'になったら LINE_OUT 信号を'0'にしてください。
- 中央揃え:
  - カウンターの方向を COUNT_UPDN1 モードに設定するために CTRL レジスタの UP_DOWN_MODE [17:16] フィールドに値'2'を書き込んでください。
  - TR_PWM_CTRL レジスタの OVERFLOW_MODE [3:2]フィールドに値'0' (SET) を書き込んで、COUNTER が PERIOD 値に達したら LINE_OUT 信号を'1'に設定してください。
  - TR_PWM_CTRL レジスタの UNDERFLOW_MODE [5:4]フィールドに値'1' (CLEAR) を書き込んで、COUNTER が'0'になったら LINE_OUT 信号を'0'にしてください。
  - TR_PWM_CTRL レジスタの CC0_MATCH_MODE [1:0]フィールドに値'2' (INVERT) を書き込んで、COUNTER が CC0 値と等しい時に LINE_OUT 信号を反転させてください。

図 122 に、CC0 値を変えた COUNT_UP モードの PWM を示します。この図に、COUNT_UP モードの PWM を使用して左揃えの PWM を作成する方法も示します。

注:

- tc イベントで CC0 が変更されます (描かれていない CC0_BUFF に)。
- スイッチイベントは常に"1"

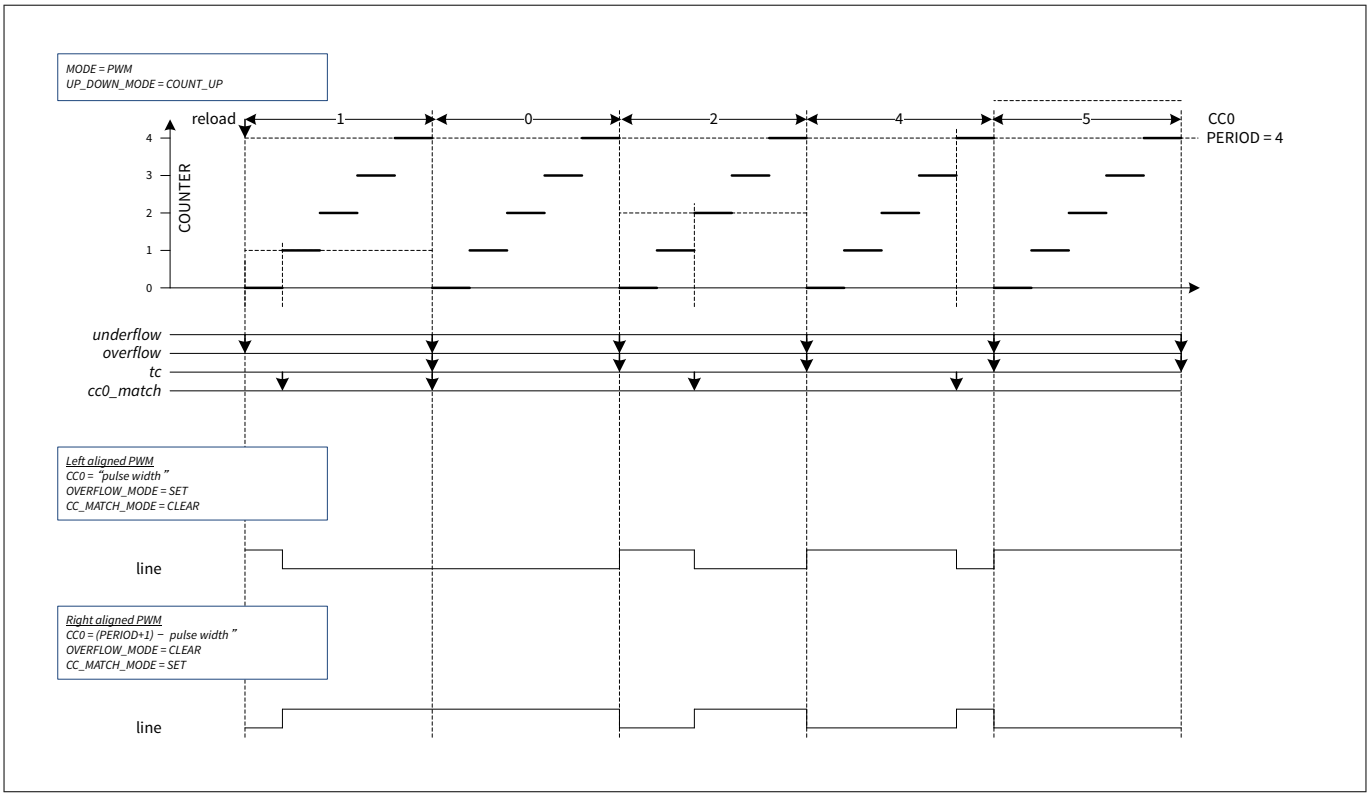


図 122　　左揃え PWM

図 123 に、CC0 の値を変えた COUNT_DOWN モードの PWM を示します。この図に、COUNT_DOWN モードの PWM を使用して、右揃えの PWM を作成する方法も示します。

注:

- TC イベントで CC0 が (描かれていない CC0_BUFF に) 変更されます
- スイッチイベントは常に"1"

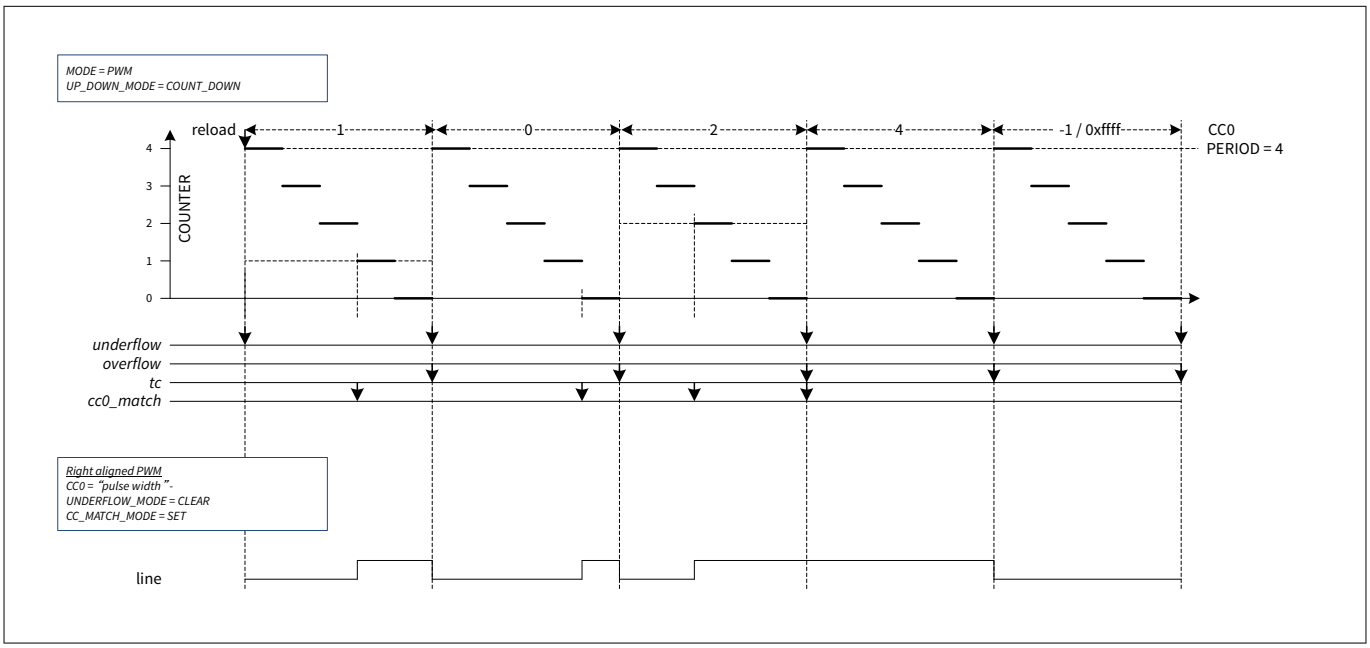


図 123　　右揃え PWM

図 124 に、COUNT_UPDN1 の PWM を CC0 の値を変えて示します。この図に、COUNT_UPDN1 モードの PWM を使用して中央揃え PWM を作成する方法も示します。

注:

- スイッチ イベントは、ハードウェア トリガー 1 によって生成され、これは一定の'1'のため、TC 条件では常にアクティブです。
- TC イベントで CC0 が (描かれていない CC0_BUFF に) 変更されます
- cc0_match は、CC0 が"0"に切り替わるのと同時に、カウント期間の最初に生成されます。cc0_match が生成される特殊なケースとして、「COUNTER が CC0 に等しい」という条件なしに発生するものがあります (CC0 は COUNTER が“0”から“1”に変化するのと同時に“0”に変化します)

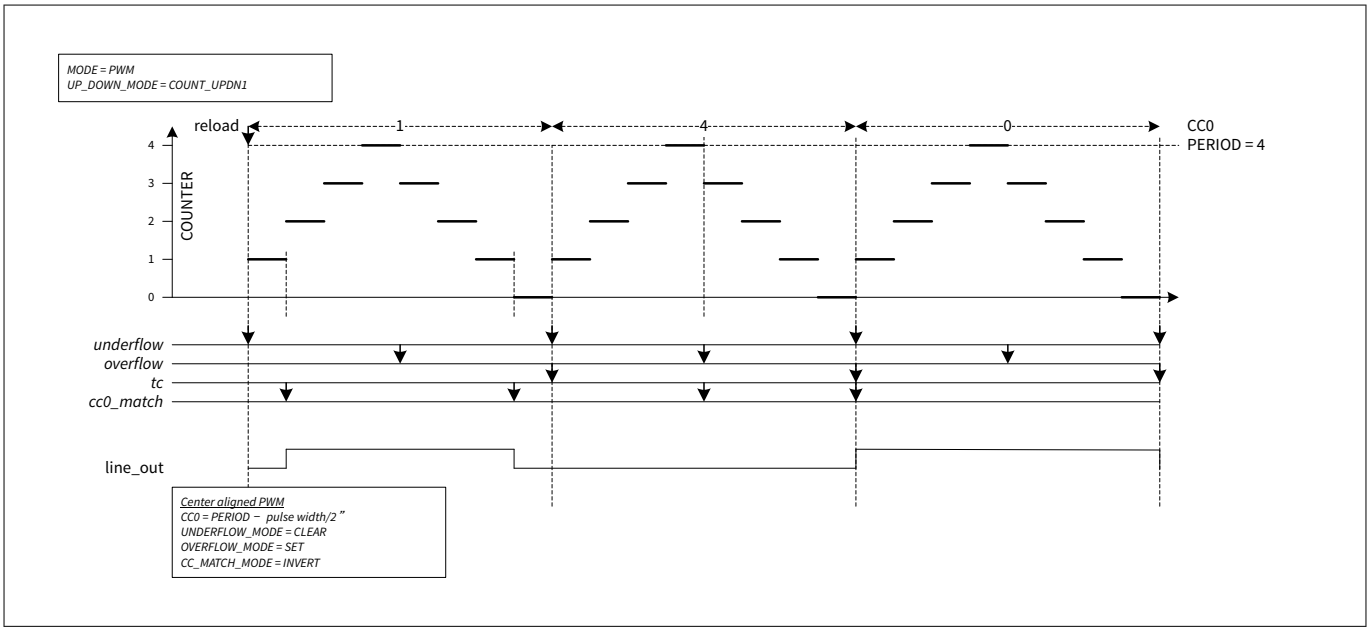


図 124　　中央揃え PWM

図 125 に、CC0 がリロード イベントが来たときに 0 になるという、もう一つのコーナーケースを示します。リロード イベント前の実際のカウンター値は使用されず、リロードイベント前のカウンター値が 0 とみなされます。その結果、リロード イベントでの最初の CC0 値が 0 になると、cc0_match イベントが発生します。

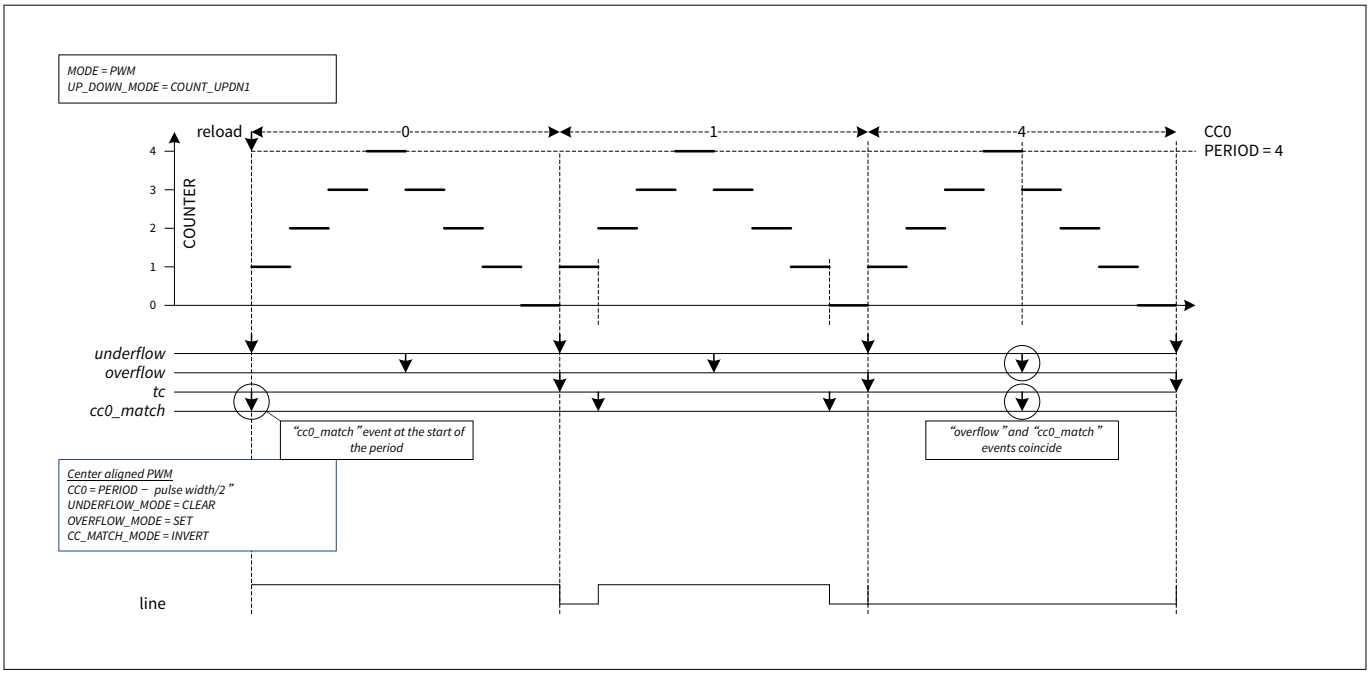


図 125　　リロード後、CC0 = "0"で中央揃え PWM

非対称 PWM

PWM モードは非対称 PWM の生成をサポートします。非対称 PWM の場合、"ライン"パルスは必ずしも周期の中央にあるとは限りません。この機能は、カウントアップ時とカウントダウン時に異なる CC0 値を持つことで実現されます。CC0 と CC0_BUFF の値はオーバーフロー時に交換されます。これにより、生成される「ライン」パルスの非対称性が制限されることに注意してください。

非対称 PWM に必要な設定は以下のとおりです。

- 非対称 PWM: COUNT_UPDN2 モードで"overflow=SET、underflow=CLEAR、cc0_match=INVERT"

非対称 PWM は通常、カウントアップ時とカウントダウン時に同じ周期値を使用します。PERIOD と PERIOD_BUFF が TC イベント (オーバーフローまたはアンダーフロー イベント) で切り替わる場合、以下を確認してください。

- PWM 周期内では、周期値は同じです (PERIOD と PERIOD_BUFF のオーバーフロースイッチは周期値を変更してはいけません、つまり PERIOD_BUFF は PERIOD でなければいけません)。
- PWM 周期の間で、周期値が変化することがあります (PERIOD と PERIOD_BUFF のアンダーフロー スイッチにより、周期値が変化することがあります。つまり、PERIOD_BUFF は PERIOD と異なる場合があります)

図 126 に、COUNT_UPDN2 モードを使って非対称 PWM を生成する方法を示します。

注:

- TC イベントは、アンダーフローとオーバーフローの両方のイベントに対して生成されます。TC イベントは、CC0 と CC0_BUFF の値を交換するために使用されます。

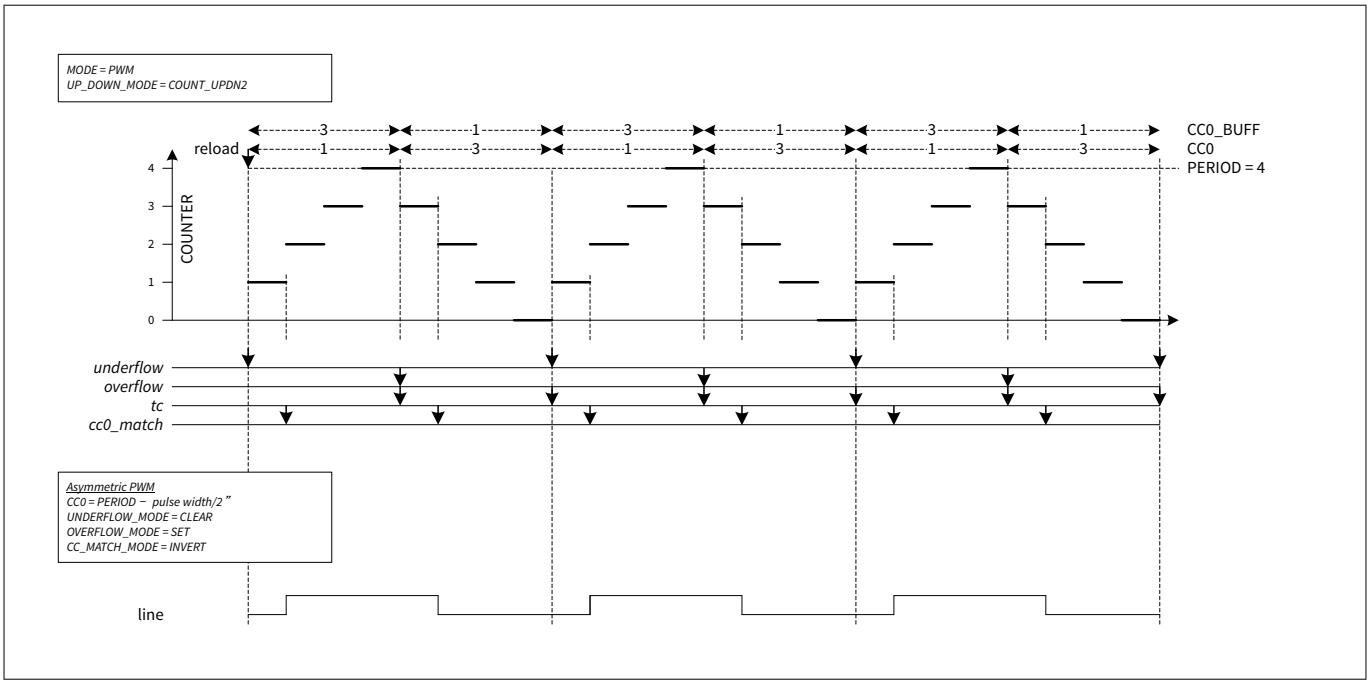


図 126　　非対称 PWM

CC0 の値が"0"でも"PERIOD"でもない場合の機能を波形で示したものです。CC0 の値が"0"または"PERIOD"に等しい場合のコーナーケース条件を以下に示します。

***注**:*

- *アップカウント時、アンダーフロー時の CC0 値が 0 の場合、cc0_match イベントが発生します。*
- *ダウンカウント時、オーバーフローイベント時の CC0 値が PERIOD の場合、cc0_match イベントが生成されます。*
  - *tc イベントは、アンダーフローとオーバーフローの両方のイベントに対して生成されます。tc イベントは CC0 と CC0_BUFF の値を交換するために使用されます。*
- *ソフトウェアは、tc イベントの割込みハンドラで CC0_BUFF と PERIOD_BUFF を更新する (そして、CC0/CC0_BUFF と PERIOD/PERIOD_BUFF の交換からハードウェアが更新した値を上書きします)。*

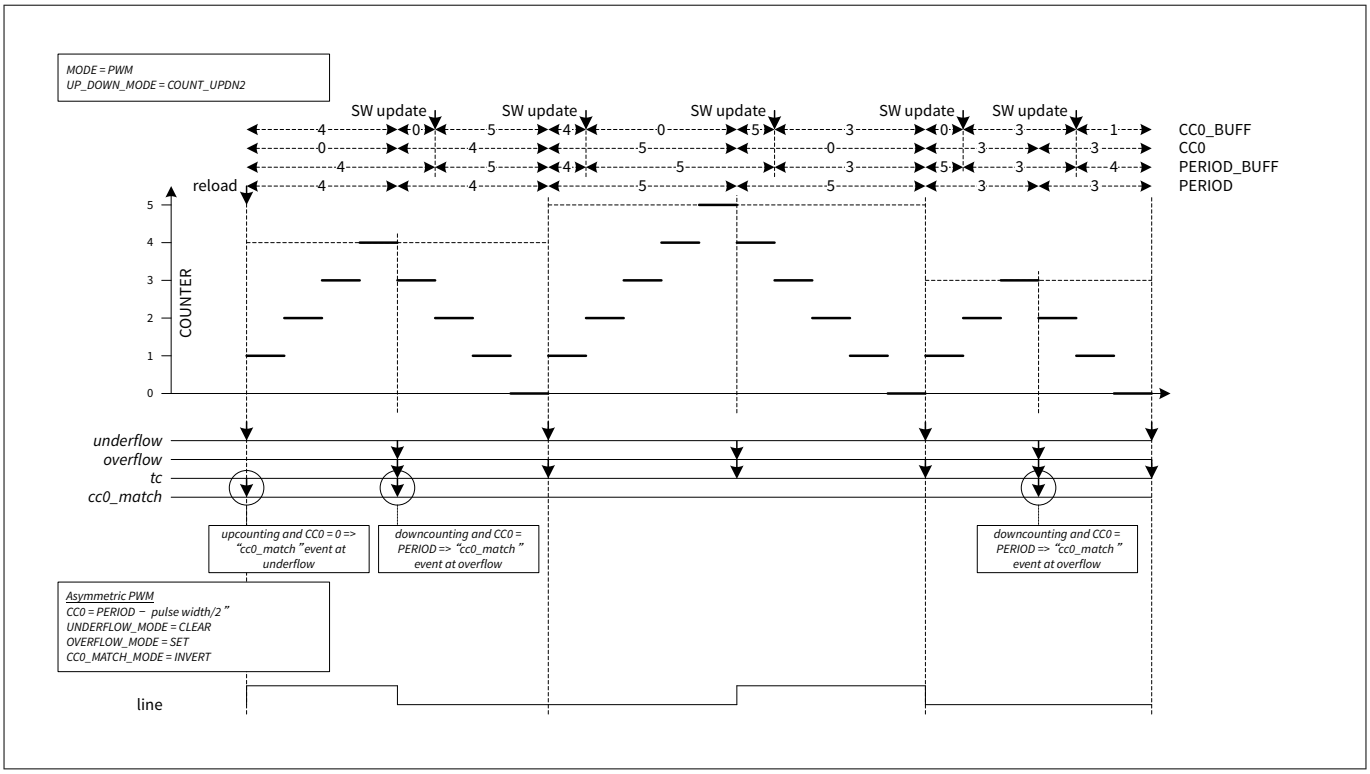


図 127　　CC0 = 0 および CC0=PERIOD デューティ サイクルの非対称 PWM

cc1_match イベントを発生させるレジスタ CC1 と CC1_BUFF を使ったコンペア 1 機能は、cc0_match イベントを発生させるレジスタ CC0 と CC0_BUFF を使ったコンペア機能と同じです。

cc1_match イベントは、PWM 出力信号の生成にも使用できます。PWM 出力信号の生成に cc0_match イベントと cc1_match イベントの両方を使うことで、下図に示すような非対称 PWM を生成する別の方法が得られます。

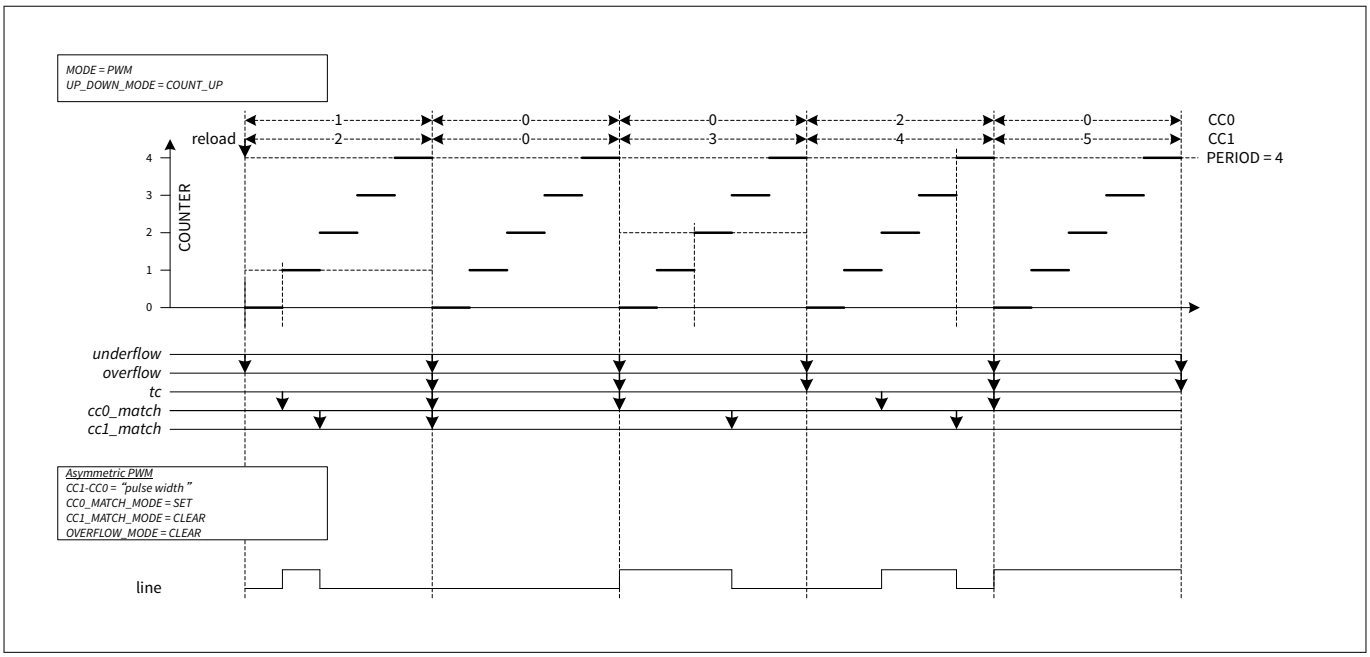


図 128　　COUNT_UP モードでの cc0_match と cc1_match による非対称 PWM

このような非対称 PWM 生成は、COUNT_UPDN2 モードで 1 つのコンペア機能のみを使用するよりも柔軟です。しかし、別の (3 番目の) コンペア機能が必要な場合、例えば ADC をトリガーするために、別の同期動作カウンターを使用する必要があります。

コンペアマッチ 0 とコンペアマッチ 1 のイベント発生は、アップカウントとダウンカウント (COUNT_UPDN1/2 モード時) に対して個別に有効または無効にできます。これにより、COUNT_UPDN1 モードでは、一方のコンペアー致イベントがカウントアップ中のみ PWM 出力を変更し、他方のコンペアー致イベントがカウントダウン中のみ PWM 出力を変更する非対称 PWM 生成が可能になります。これは 図 129 に示されており、ある期間から次の期間へとデューティサイクルの値が増加する時間 (PWM 信号に変調されたサイン波の立ち上り部分) における、モーター制御用の 3 つの中央に並んだ PWM フェーズのうちの 1 つを示します。

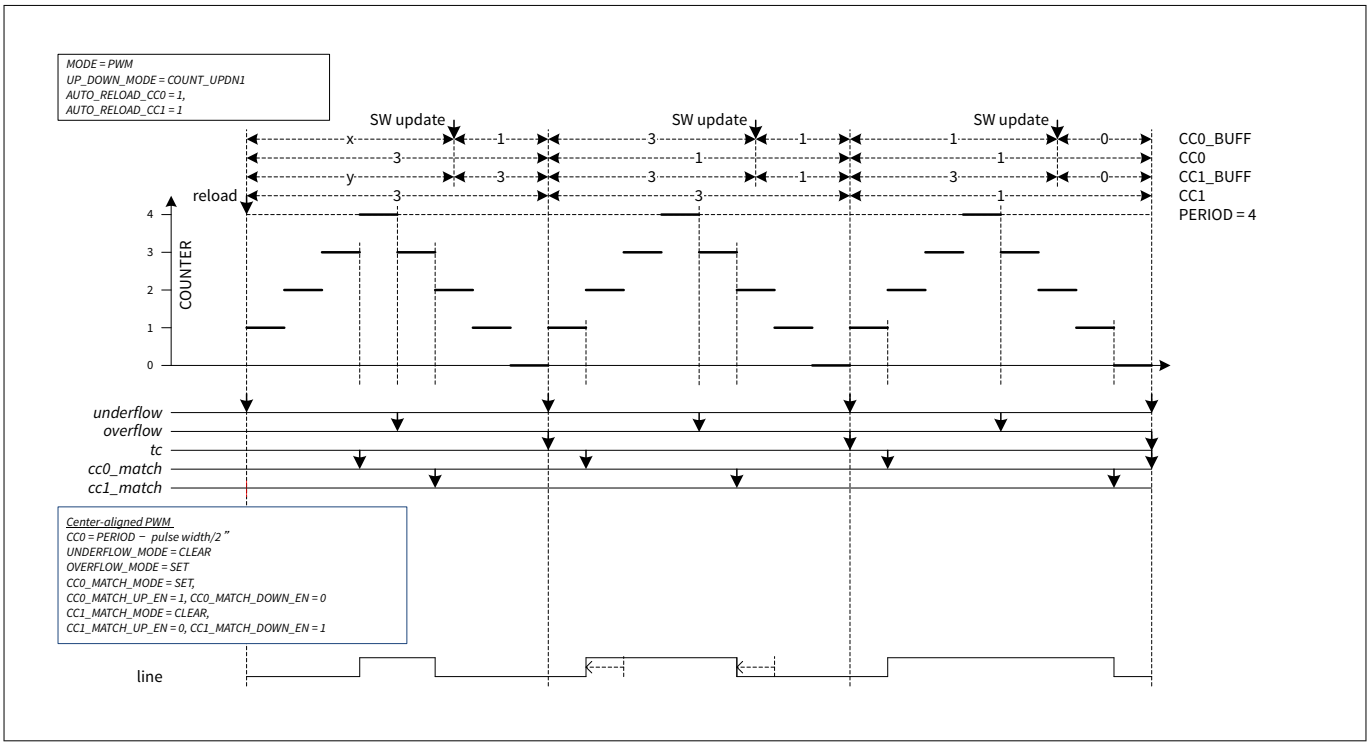


図 129　　COUNT_UPDN1 モードでの cc0_match と cc1_match による非対称 PWM

PMSM (永久磁石同期モーター) のモーター制御において、3 相の電流デューティ サイクル値がそれを許さない場合 (1 つの PWM チャネルがアクティブで、他の 2 つのチャネルがアクティブでないウィンドウが小さすぎる場合)、常に中央揃えされた PWM の代わりに、PWM 信号の位相を一時的にシフトさせ、1 つのシャント電流測定 (2 つのトリガーでの電流測定とソフトウェアでの差分計算) を可能にできます。COUNT_UPDN2 モードで単一の比較機能のみを用いて実現される非対称 PWM と比較して、本ソリューションは 2 つの独立したバッファ コンペア値を使用し、CPU 負荷を低減します (必要な割込みが減少)。すなわち、すべての更新処理を「ADC 完了」割込みサービスルーチン内で実行可能で、新たなデューティサイクル値を計算し、単一シャント電流測定のための一時的な位相シフトを導入します。

第 2 のコンペア機能を使用して非対称 PWM を有効にする方法:

- "cc0_match=SET、cc1_match=CLEAR"の COUNT_UP/DOWN モード。パルス幅 = CC1-CC0
- COUNT_UPDN1/2 モードで「overflow=SET, underflow=CLEAR, cc0_match=SET, cc1_match=CLEAR, CC0_MATCH_UP_EN = 1, CC0_MATCH_DOWN_EN = 0, CC1_MATCH_UP_EN = 0, CC1_MATCH_DOWN_EN = 1」。パルス幅 = 2xPERIOD - CC1 - CC0

Kill Mode (キル モード)

PWM モードには異なるストップ/キルモードがあります。モードは PWM_IMM_KILL、PWM_STOP_ON_KILL、PWM_SYNC_KILL、PWM_TC_SYNC_KILL_DT、PWM_SYNC_KILL_DT で指定されます。

コンフィギュレーション オプション PWM_IMM_KILL は、PWM 出力信号の抑制を開始するタイミングを指定します。

- PWM_IMM_KILL は'1'です。PWM 出力信号 DT_LINE_OUT と DT_LINE_COMPL_OUT は、キルイベントが検出されると即座に抑制されます
- PWM_IMM_KILL は'0'です。PWM 出力信号 DT_LINE_OUT と DT_LINE_COMPL_OUT はキルイベント検出後、次のカウントクロックに同期して抑制されます

以下の図に、両設定を示します。

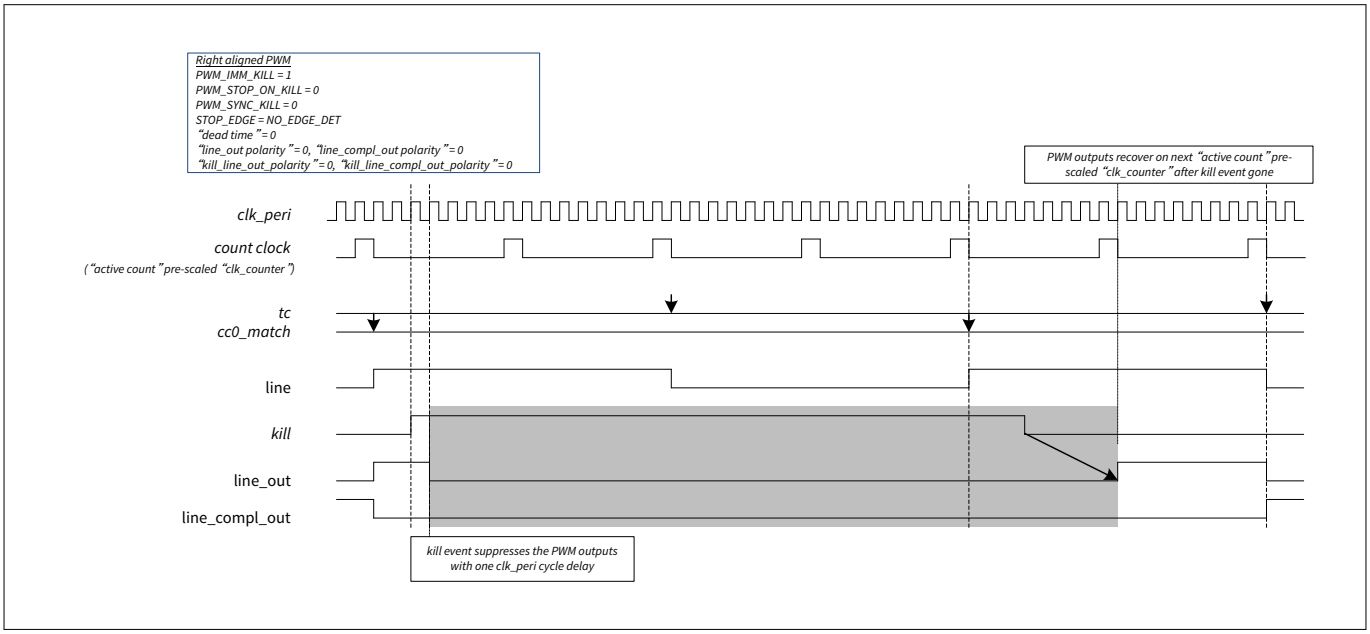


図 130　　ライン出力を即座に抑制 (PWM_IMM_KILL = 1)

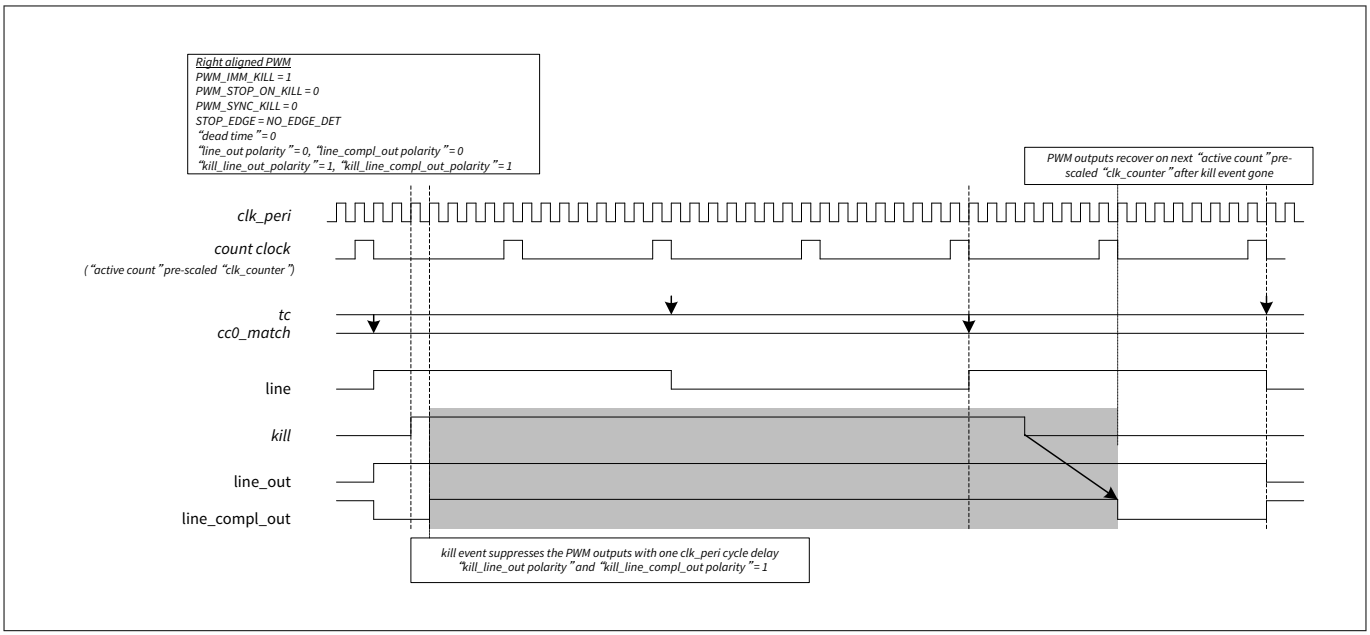


図 131　　"kill_line_out_polarity"と"kill_line_compl_out_polarity" = 1 で、ライン出力を即座に抑制 (PWM_IMM_KILL = 1)

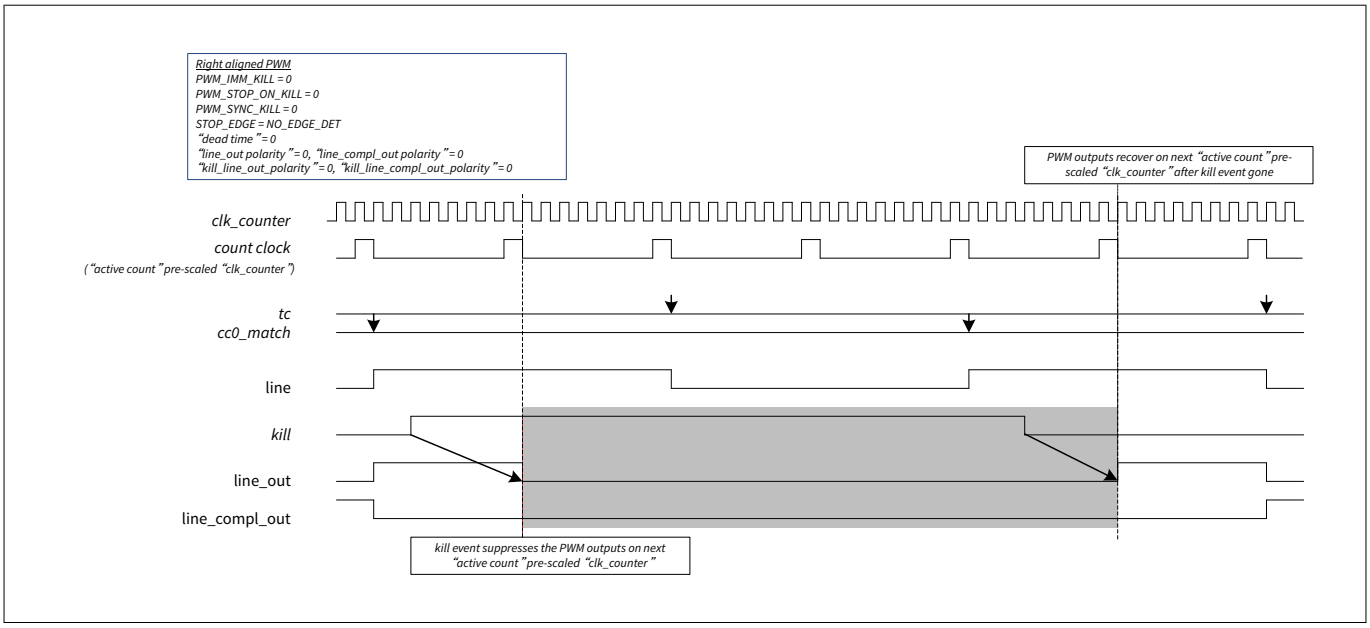


図 132　　カウントクロックによるライン出力の抑制 (PWM_IMM_KILL = 0)

以下の図は、"kill_line_out_polarity"と"kill_line_compl_out_polarity"を 0 と 1 に設定し、前述の 5 つのモードを示したものです。

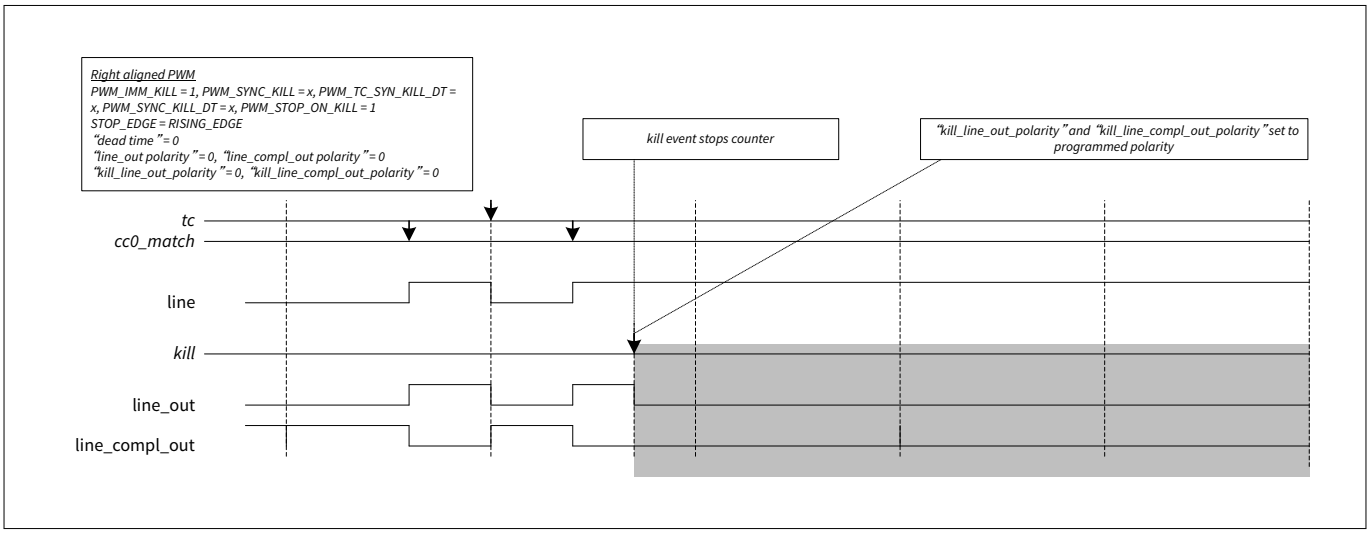


図 133　　キルでストップ

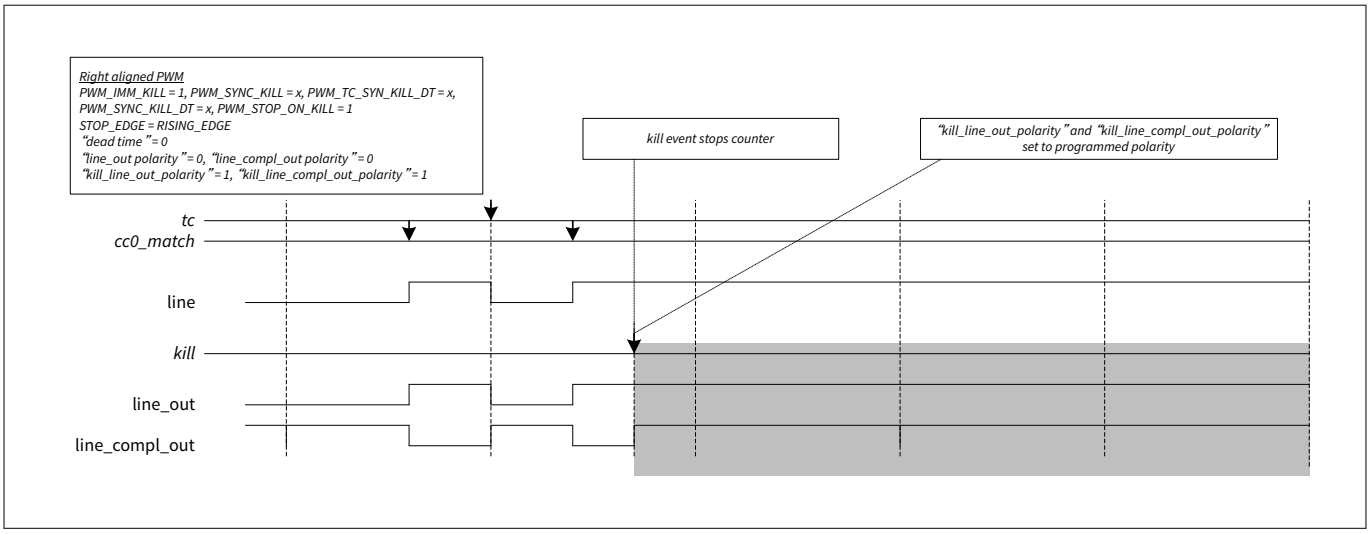


図 134　"kill_line_out_polarity"と"kill_line_compl_out_polarity" = 1 でキル時に停止

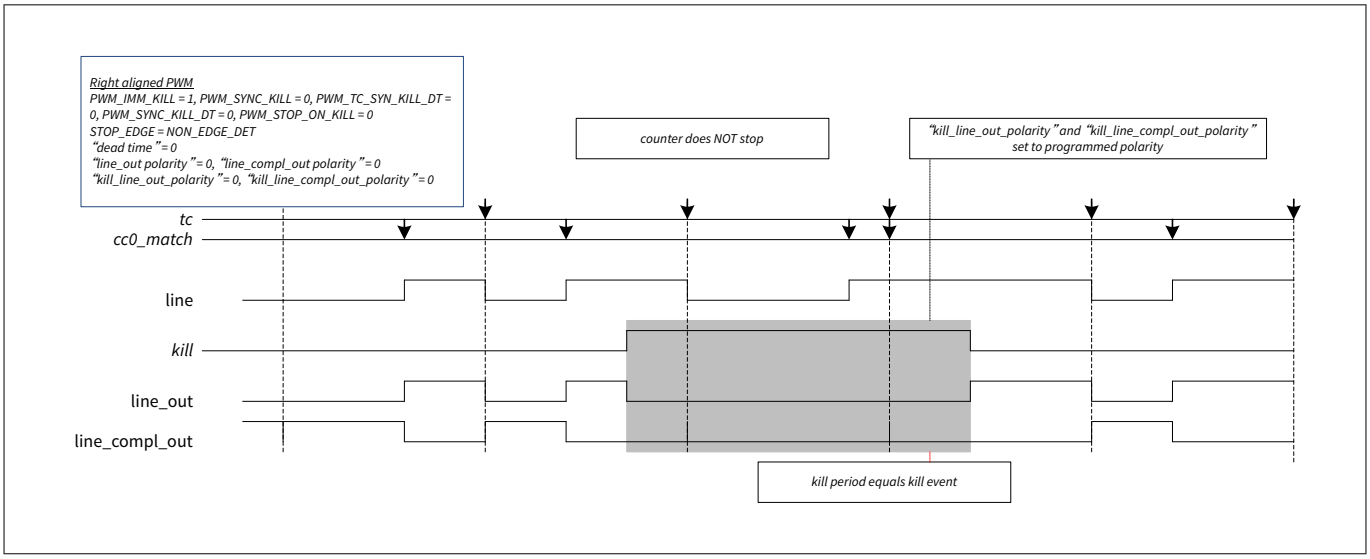


図 135　非同期キル

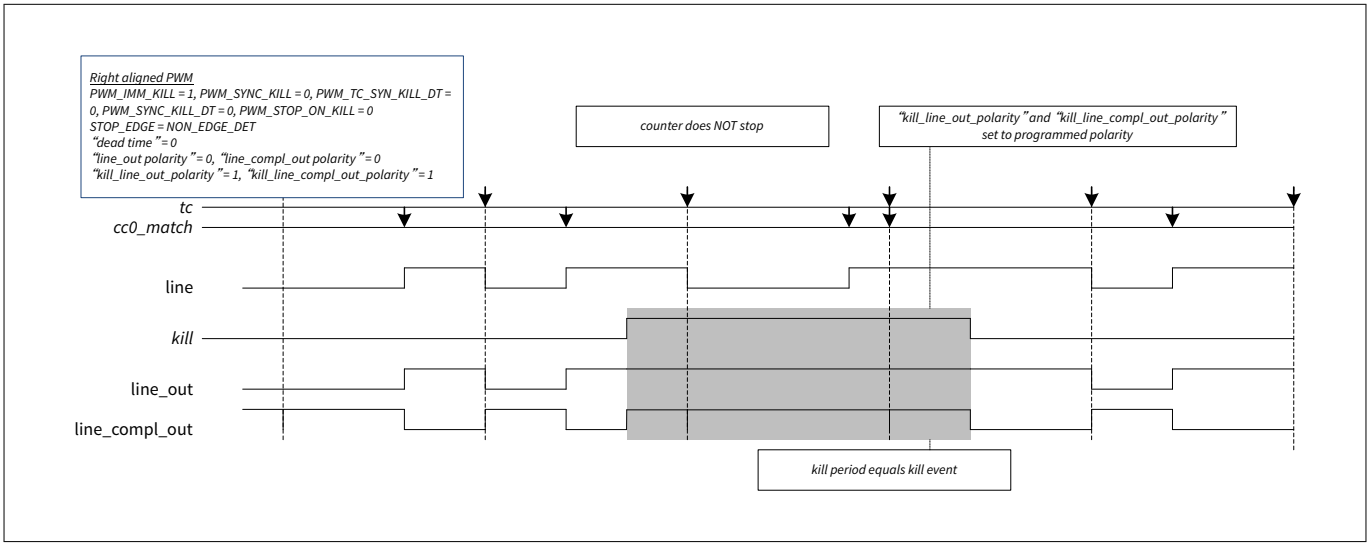


図 136　"kill_line_out_polarity"と"kill_line_compl_out_polarity" = 1 による非同期キル

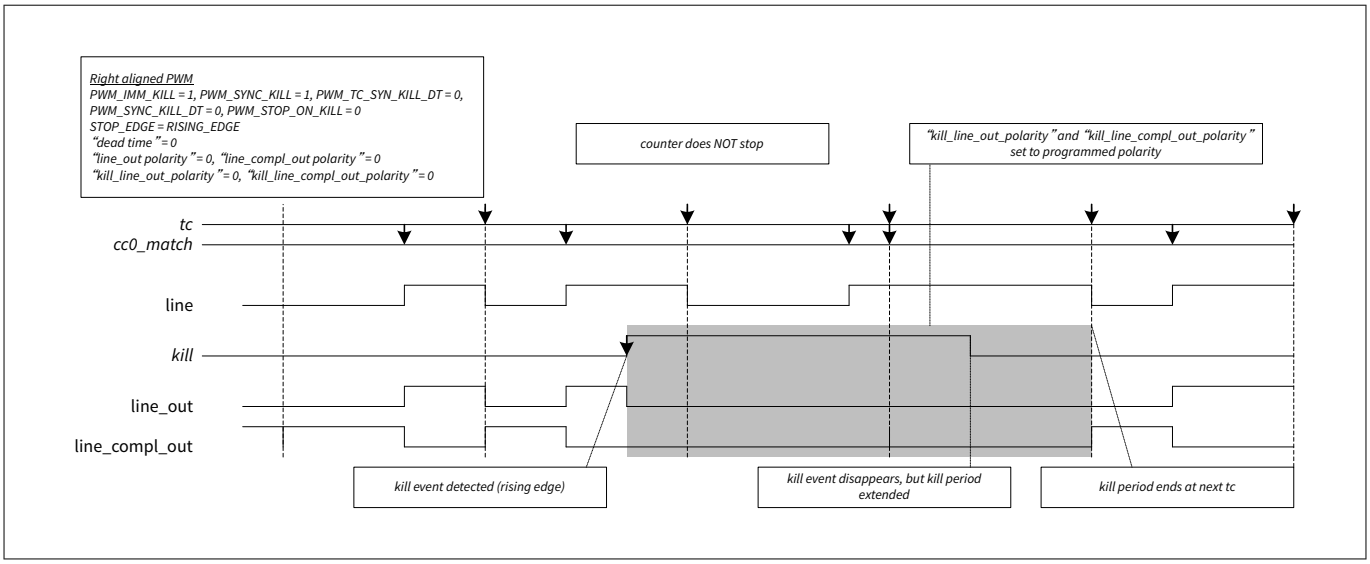


図 137　　同期キル

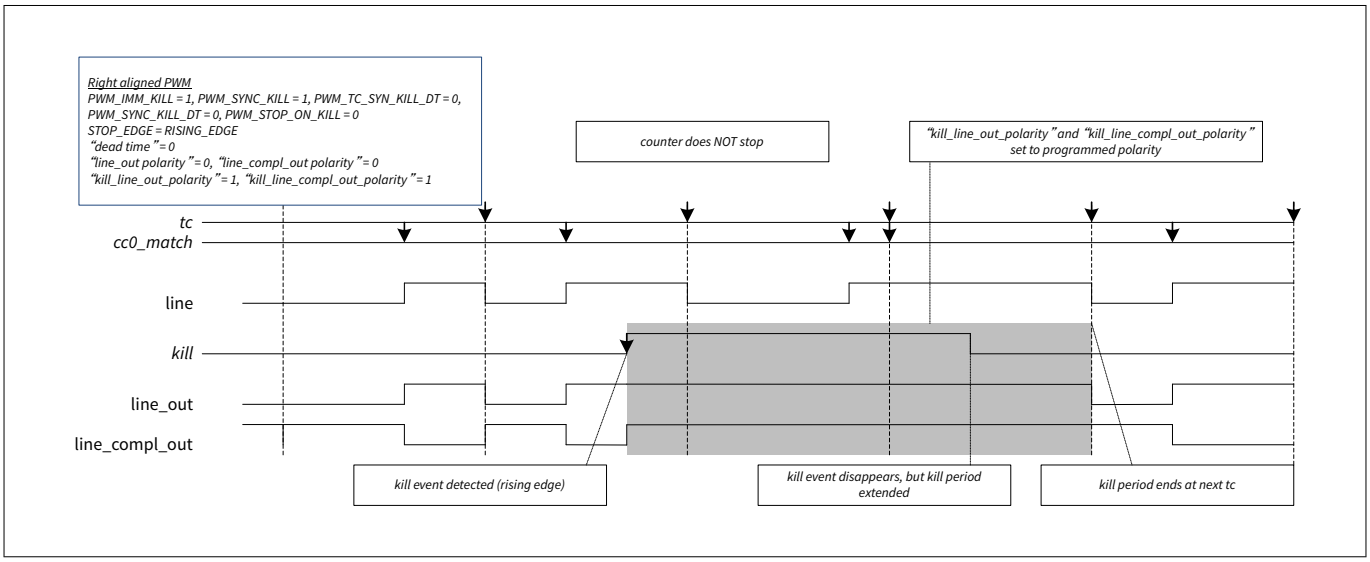


図 138　　"kill_line_out_polarity"と"kill_line_compl_out_polarity" = 1 での同期キル

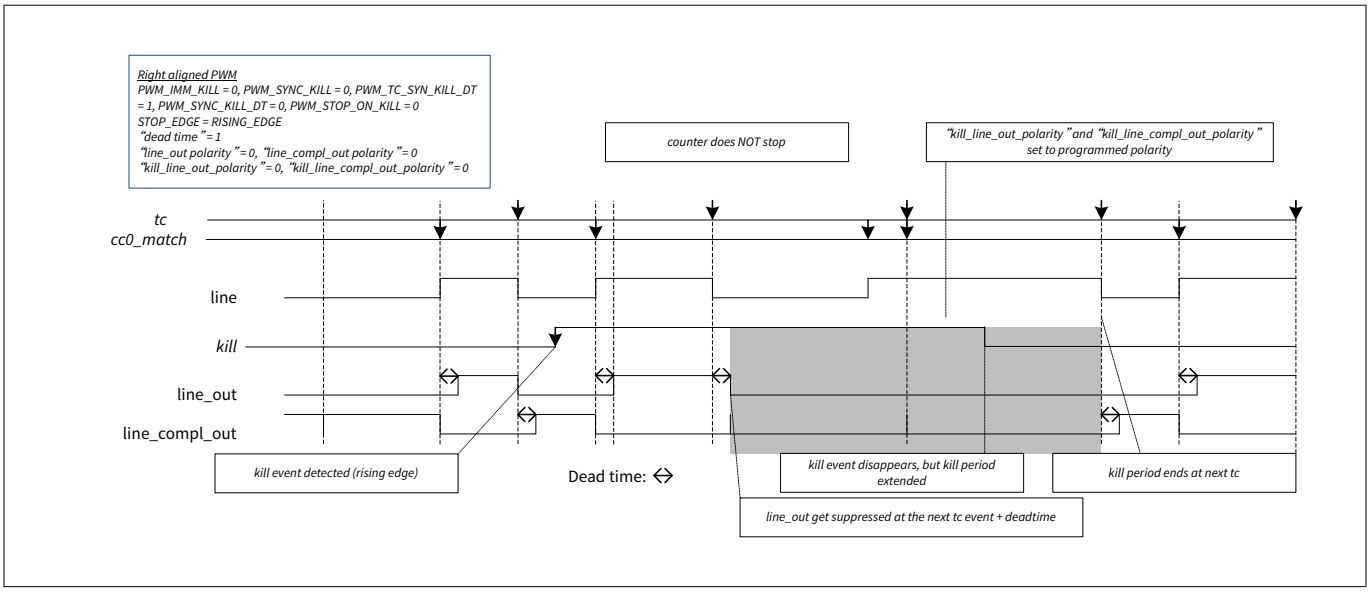


図 139　　TC 同期 DT キル

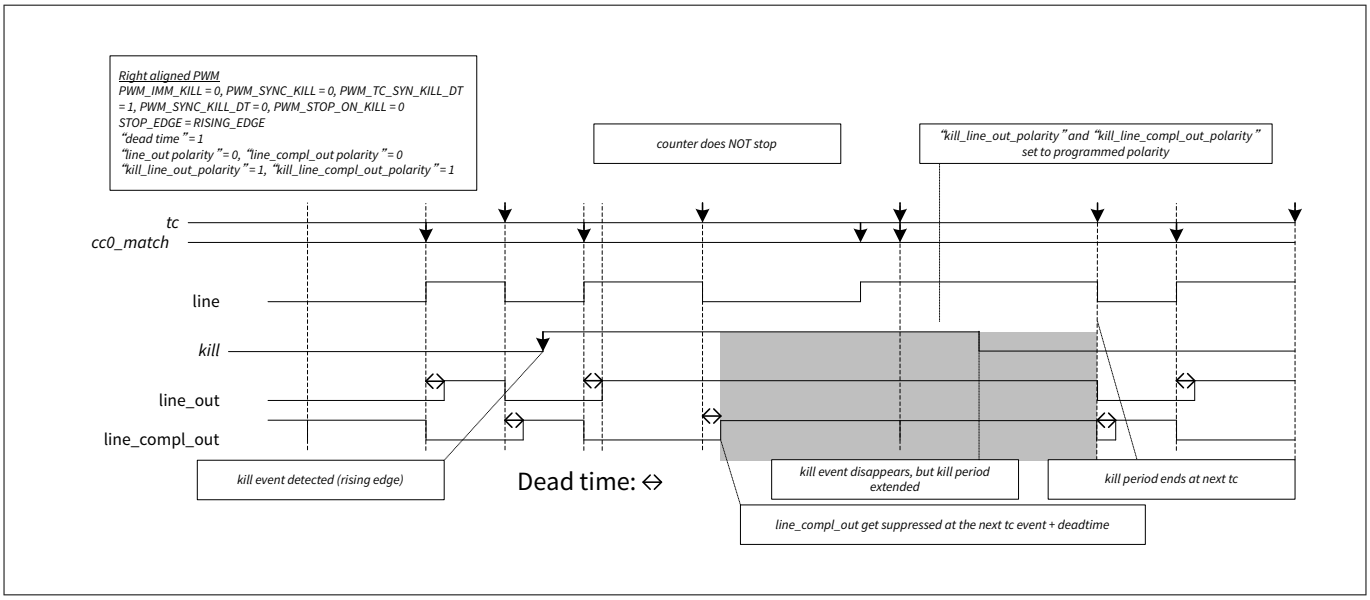


図 140　　"kill_line_out_polarity"と"kill_line_compl_out_polarity" = 1 による TC 同期 DT キル

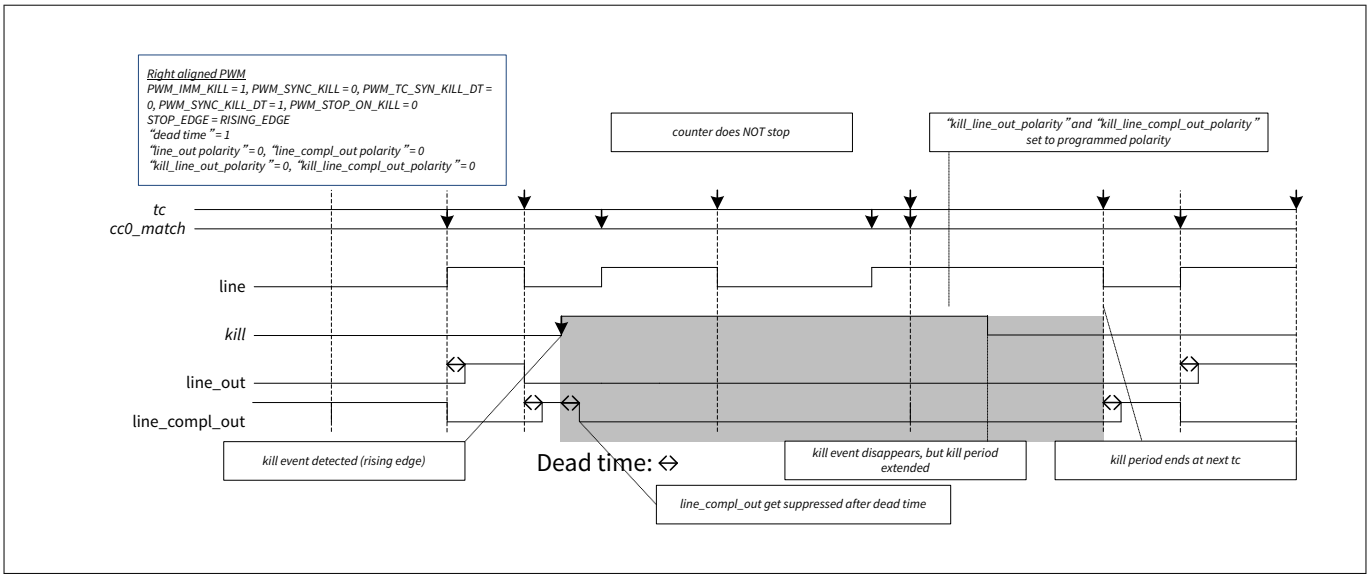


図 141　　同期 DT キル

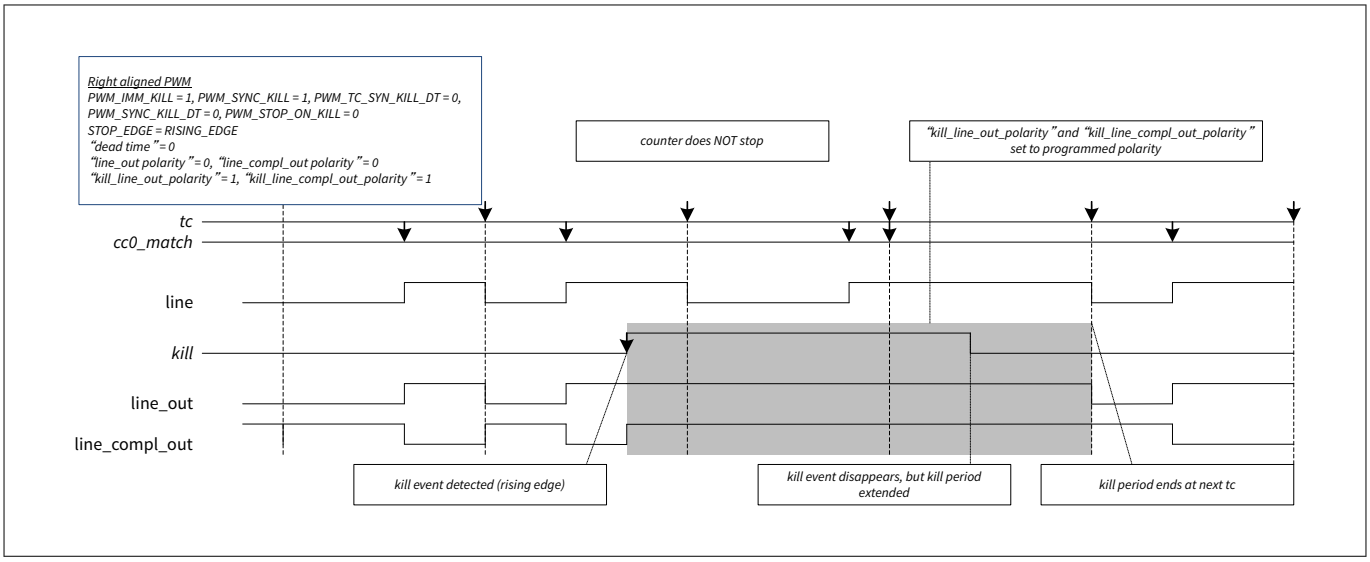


図 142　　"kill_line_out_polarity"と"kill_line_compl_out_polarity" = 1 での同期 DT キル

非同期 kill とは異なり、同期 kill は kill が終了した後、tc イベントでのライン出力の抑制を停止します。

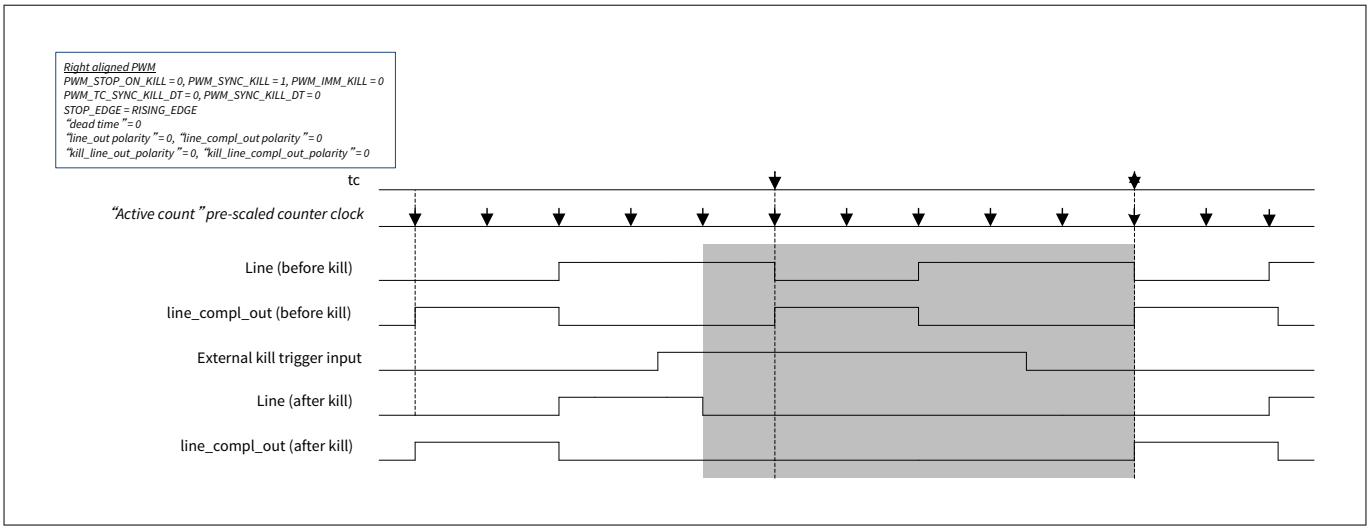


図 143　　同期キルタイミング (PWM_IMM_KILL = 0)

PWM_IMM_KILL の値が何であれ、PWM はキル イベントが非アクティブになった後、次の"アクティブ カウント"プリスケール カウンター クロック (PWM_SYNC_KILL = 0) または次の TC (PWM_SYNC_KILL = 1) で回復します。例外として、PWM_SYNC_KILL="0"、PWM_IMM_KILL="1"の場合、キル イベントのパルス幅が狭すぎて、"アクティブ カウント"プリスケール カウンター クロックでは認識できません。PWM 出力は下図のようにすぐに回復します。

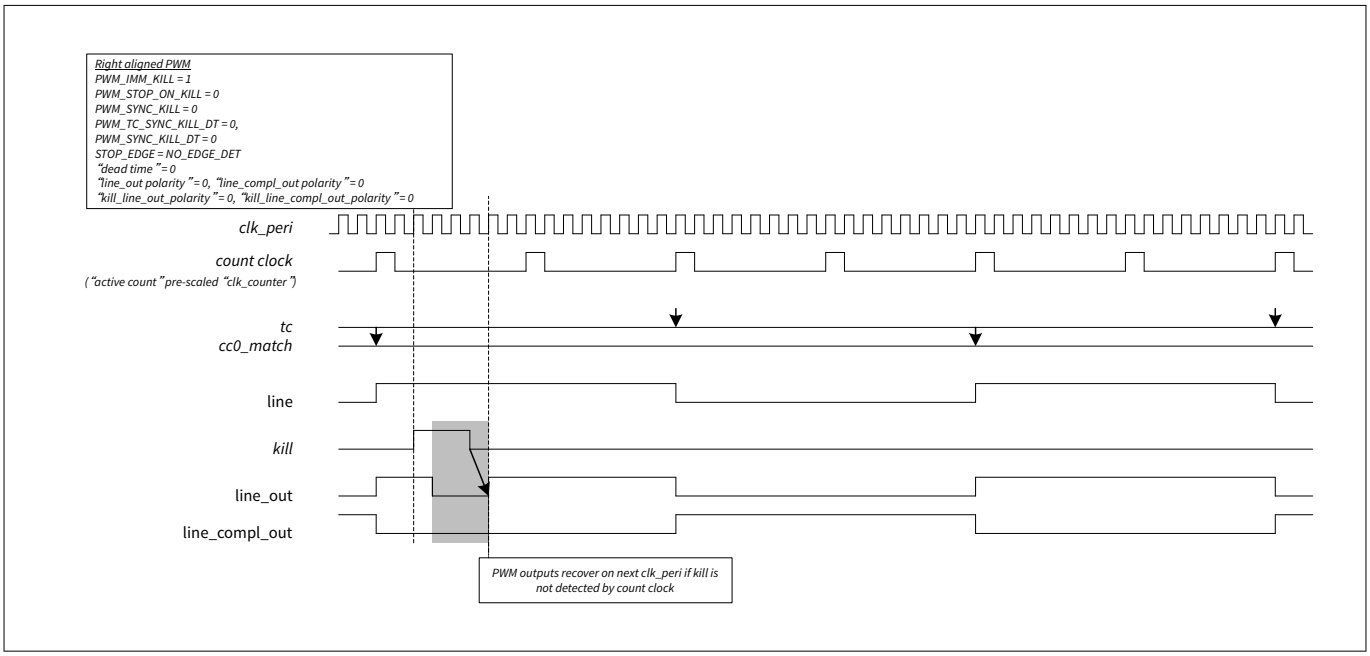


図 144　　カウントクロックでキルが検出されなかった場合、PWM 出力は次の clk_peri で回復

Capture1 イベントは、stop イベントと同様に、kill 入力としても機能します。両イベントは論理和され、同じキルモード設定を共有します。PWM のために 2 つの停止/キルイベントを持つことは、1 つの停止/キルイベントのために PERI トリガーマルチプレクサからの共通トリガー (複数の PWM の同期停止/キル動作を可能にする) を選択する一方で、もう 1 つの停止/キルイベントのために、例えば、専用の ADC 範囲外トリガー (例えば、PWM 駆動信号の電流が範囲外の場合に PWM のリアルタイムのハードウェア停止を可能にする) を選択することを可能にします。

ステッピングモーター制御用 PWM 出力選択

PWM モードでは、TCPWM カウンターはマイクロステッピングを含むステッピング モーター制御に使用できます。したがって、2 つの PWM 出力信号"line_out"と"line_compl_out" (通常の PWM 動作では相補的な一対の PWM

信号) は、出力選択レジスタ (レジスタ LINE_SEL、フィールド OUT_SEL と COMPL_OUT_SEL) を使用して以下のオプションに設定できます。

- コンスタントロー ('0')
- コンスタントハイ ('1')
- PWM 信号"ライン"
- 反転 PWM 信号"ライン"
- 「Z」(ハイ インピーダンス)

フル ステッピングには一定のロー値またはハイ値が使用され、マイクロ ステッピングには PWM 信号が使用されます。オプション"Z"により、TCPWM 出力信号"line_out_en"または"line_compl_out_en"を'0'に設定でき、接続された IO セルは出力を駆動しません。これにより、現在駆動していないコイルの誘導電圧を ADC を介して測定でき、ゼロ点検出に使用されます。

***注**:* *フルステッピングの場合は、TCPWM カウンターを PWM モードに設定します。しかし、PWM 生成は必要ありません。つまり、コンペア機能リソース (CC0/CC1) は、他の目的 (例えば、トリガー生成など) に自由に使用できます。*

ステッピング モーターの各コイルには、個別の TCPWM カウンターが使用されます。これらのカウンターは同期して実行されている (同じリロード/スタート/ストップ/カウントイベントと同じ周期を共有しています)。複数の TCPWM カウンターにまたがる出力信号の同期更新を達成するために、CC0/CC1 または PERIOD レジスタの更新にも同じ二重バッファリングとスイッチング方法が使用されます。そのため、LINE_SEL_BUFF レジスタが使用され、TC イベントでアクティブに保留されているスイッチイベント (CTRL.AUTO_RELOAD_LINE_SEL で指定された場合) で LINE_SEL レジスタと交換されます。下図に、CTRL.SWAP_ENABLE = "1"のとき、2 つの TCPWM カウンターでステッピング モーターの 2 つのコイルを駆動する例を示します。

これは、3 つのフルステップ (電気角 180°、270°、0° (= 360°)) と、PWM を使った 3 つのマイクロステップ (電気角～30°、～60°、～120°) を示します。これは、PWM カウンターの PERIOD ごとに 1 ステップを示す単なる例示です。実際には、マイクロステッピングのステップは PWM 周期に比べてかなり遅いため、PWM デューティ サイクル (CC0) および/または出力セレクト レジスタ (LINE_SEL) は、スイッチ イベントによって変更される前の多くの PWM 周期にわたって安定しています。

ステップのタイム ベースは、タイマー モードの追加 TPCWM カウンター (より遅いカウンター クロックで動作するか、より高い PWM 周期で動作するか、PWM カウンターのオーバーフロー イベントをカウントする) によって実現できます。このカウンターは、CPU が PWM デューティ サイクルおよび/または出力セレクト レジスタを更新し、MXPERI ブロックの PERI0_TR_CMD レジスタを使用して最終的にスイッチ イベントを生成するトリガーとなる割込みを生成できます。

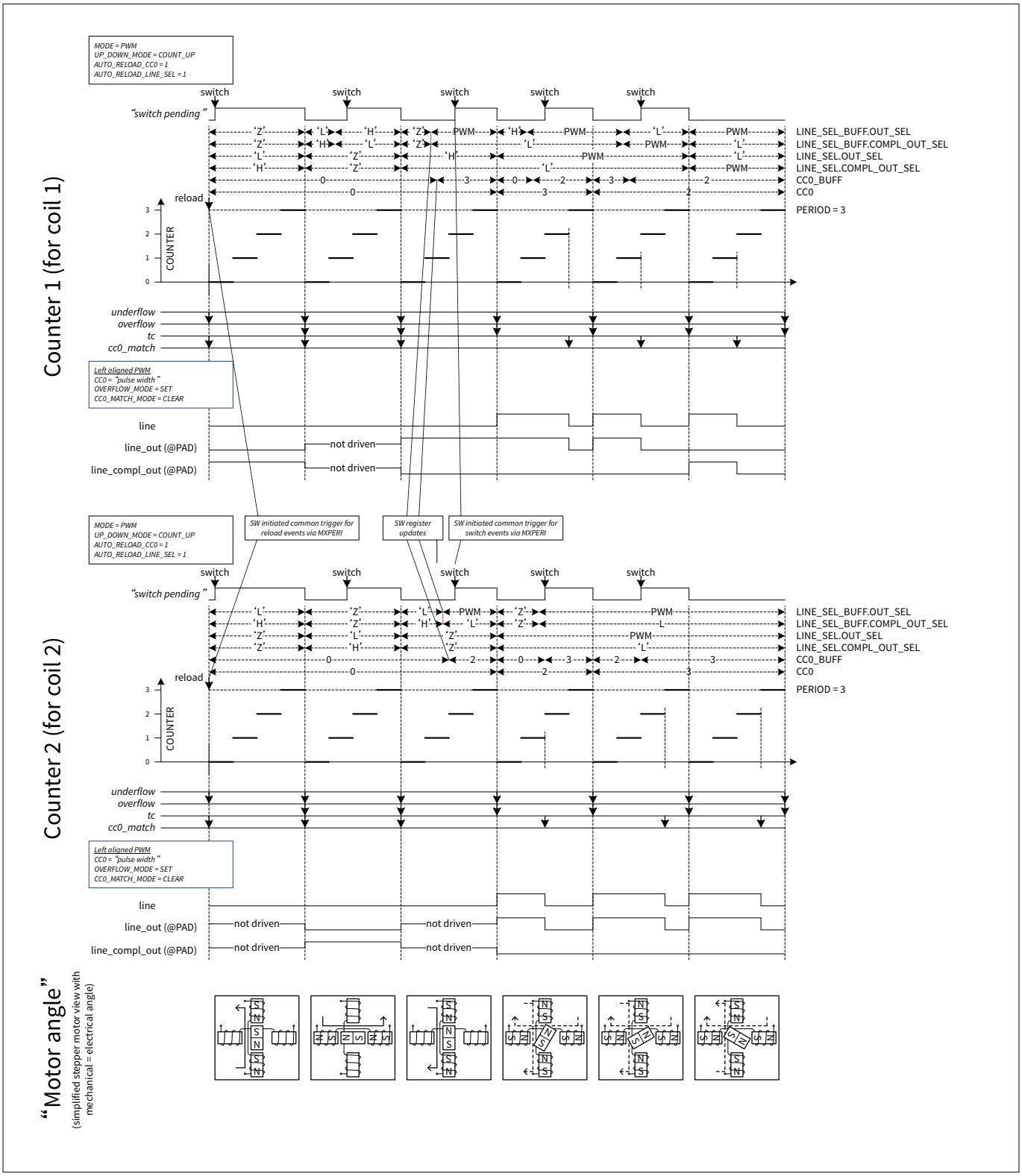


図 145　　ステッピングモーター用 PWM 出力

下図は、CTRL.SWAP_ENABLE = "0"のとき、2 つの TCPWM カウンターでステッピング モーターの 2 つのコイルを駆動する例です。ソフトウェアがステッピング モーター制御用のレジスタをどのように更新できるかを示します。

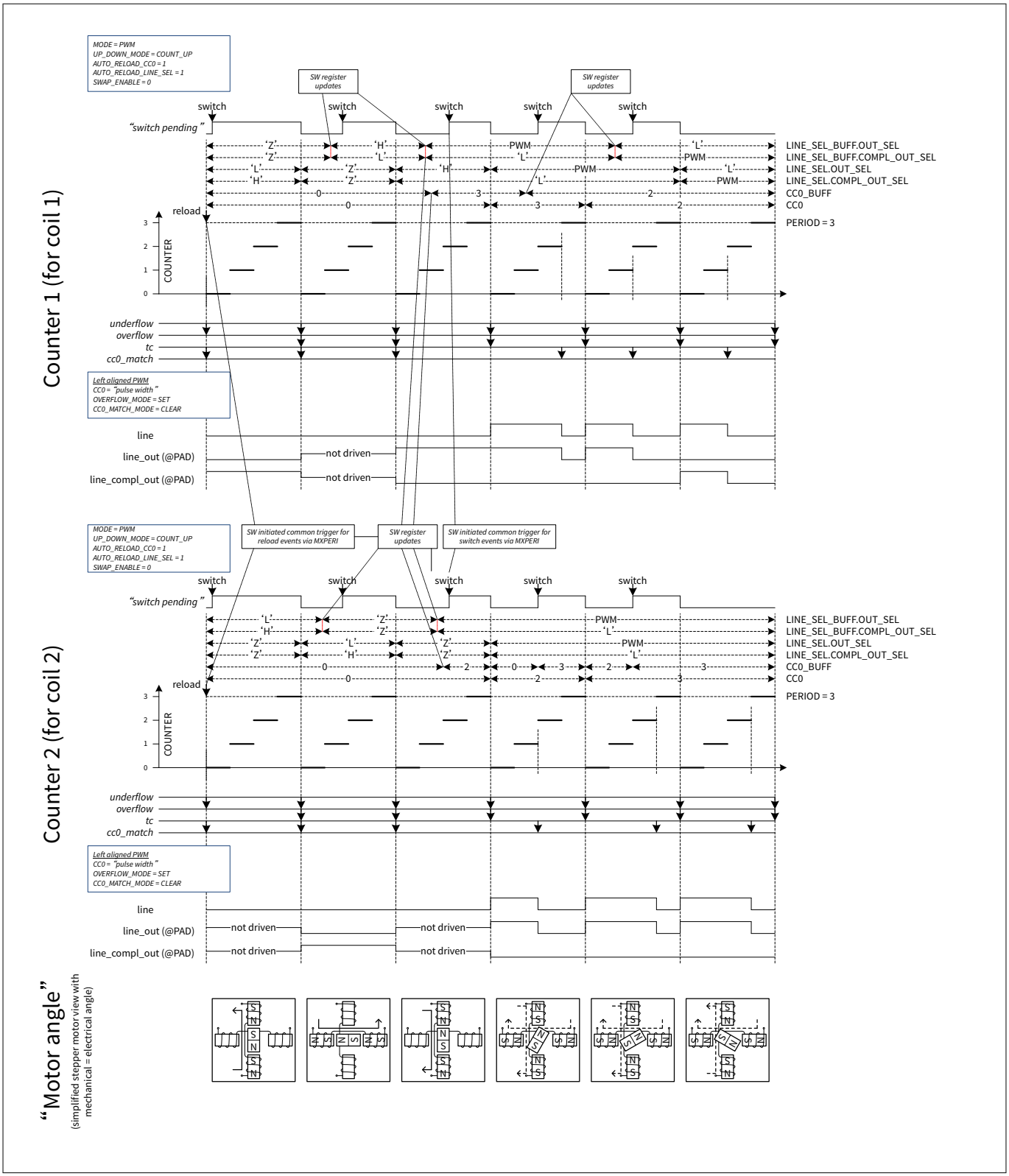


図 146　　ステッピングモーター用 PWM 出力 (SWAP_ENABLE = 0 の場合)

24.3.3.2　PWM モード用カウンターの設定方法

PWM 動作モード用にカウンターを設定する手順と、影響を受けるレジスタビットは以下のとおりです。

- CTRL レジスタの ENABLE ビットに'0'を書き込んでカウンターを無効にします。

- CTRL レジスタの MODE[26:24]フィールドに'100'を書き込むことで PWM モードを選択
- PERIOD レジスタに 16 ビット/32 ビット周期を設定し、必要に応じて PERIOD_BUFF レジスタにバッファ周期値を設定してください。
- CC0/1 レジスタの 16 ビット/32 ビット コンペア値と CC0/1_BUFF レジスタのバッファ コンペア値を、必要に応じて値を切り替えるように設定します。
- CTRL レジスタの UP_DOWN_MODE[17:16]フィールドにカウント方向を設定し、PWM を左揃え、右揃えまたは中央揃えに設定してください。
- CTRL レジスタの PWM_IMM_KILL、PWM_STOP_ON_KILL、および PWM_SYNC_KILL フィールドを必要に応じて設定してください。
- TR_IN_SEL0 または TR_IN_SEL1 レジスタを設定して、イベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーを選択します。
- TR_IN_EDGE_SEL レジスタを設定して、イベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーのエッジを選択します。
- LINE_OUT および LINE_COMPL_OUT は、cc0_match、cc1_match、オーバーフロー、アンダーフローの条件でセット、リセット、または反転するように、TR_PWM_CTRL レジスタによって制御できます。
- 必要であれば、TC または CC0/1 条件時に割込みを設定します。
- CTRL レジスタの ENABLED ビットに'1'を書き込んでカウンターを有効にします。ハードウェアのリロード信号が有効化されていない場合、カウンターを開始するためには、ファームウェア (PERI0_TR_CMD レジスタの RELOAD ビット) を通じてリロードトリガーを提供する必要があります。

24.3.4 デッドタイム付きパルス幅変調 (PWM_DT) モード

PWM_DT 機能は、以下の相違点を除き、PWM 機能と同じです。

- PWM_DT はデッドタイム挿入をサポートします。PWM はデッドタイムの挿入をサポートしません
- PWM_DT はクロック プリスケーリングをサポートします

デッドタイム挿入は、予備 PWM 出力信号「ライン」で動作するステップです。

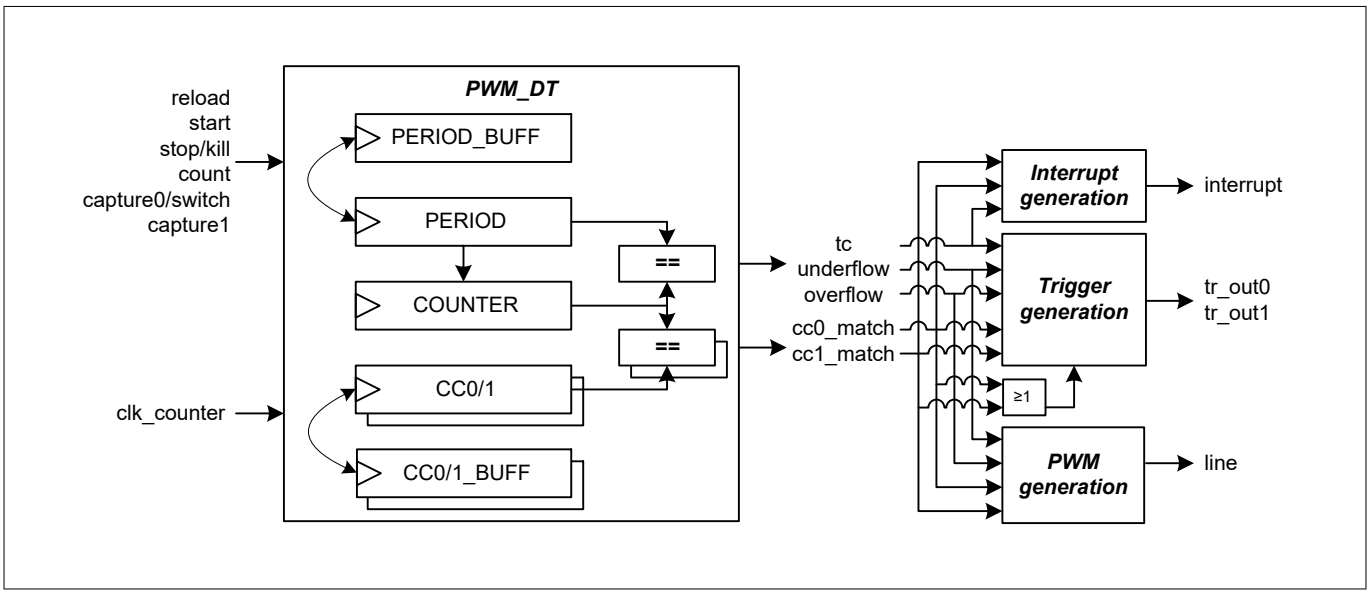


図 147 デッドタイム機能付き PWM

- デッドタイムは 16 ビット値 (最大デッドタイムは 65535 カウンター サイクル)
- PWM 出力信号"line_out"と"line_compl_out"をアクティブにする前に、異なるデッドタイムを使用できます。"line_out"のデッドタイムは、16 ビット レジスタ フィールド DT_LINE_OUT (DT_LINE_OUT_H と

DT_LINE_OUT_L で設定) で指定されます。"line_compl_out"のデッドタイムは、16 ビット レジスタ フィールド DT_LINE_COMPL_OUT で指定されます。

注: 未定義の動作を防ぐため、デッドタイム挿入 (デッドタイム値) が周期値を超えないようにしてください。

下図は、CTRL.SWAP_ENABLE = "1"のとき、PWM 出力信号 line_out と line_compl_out のデッドタイムが異なる例です。

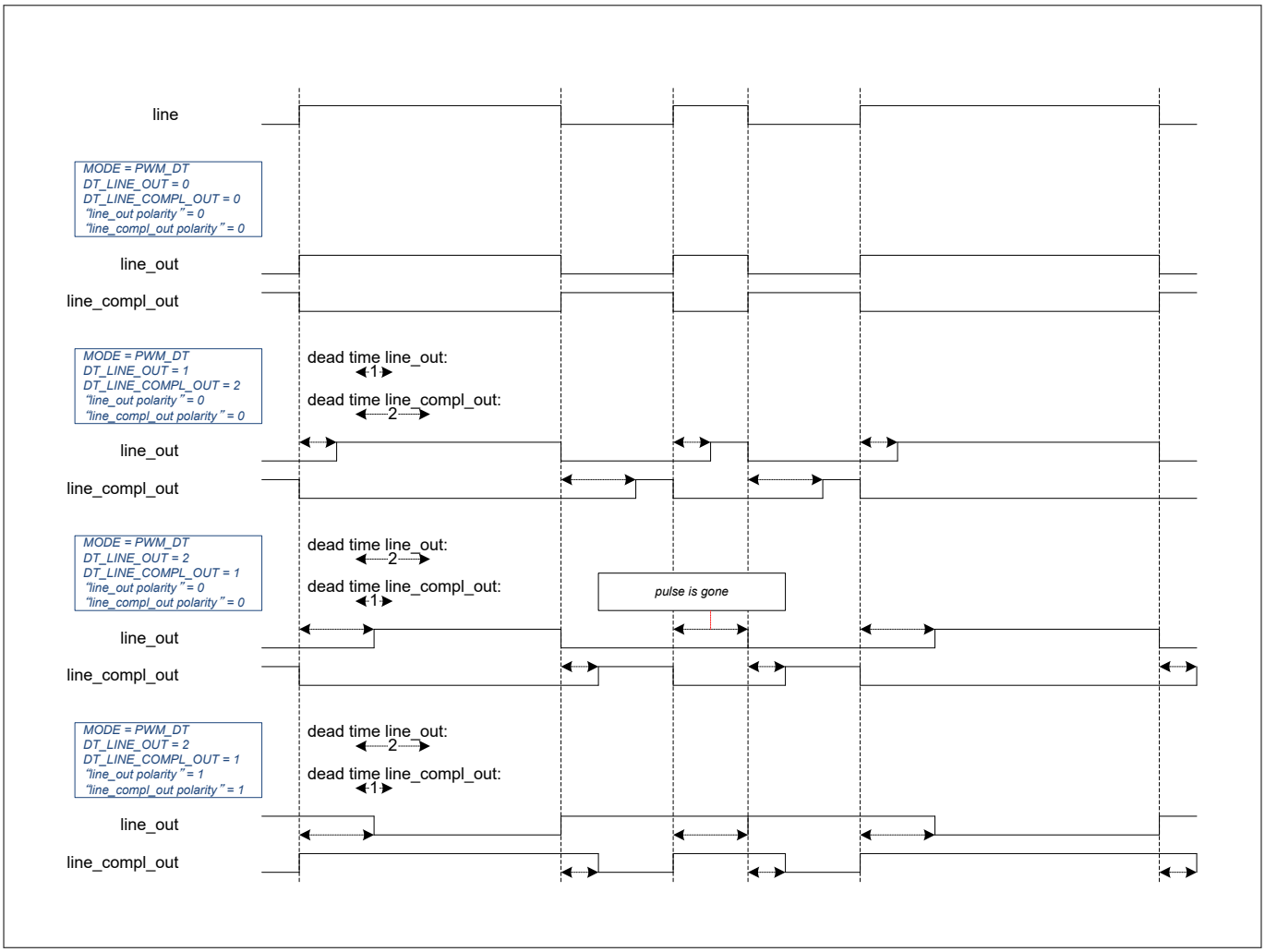


図 148 デッドタイムの異なる"line_out"と"line_compl_out"

以下の図に、CTRL.SWAP_ENABLE = "0"のときに 2 つの TCPWM カウンターを使用してステッピング モーターの 2 つのコイルを駆動する例を示します。PWM 出力信号"line_out"と"line_compl_out"のデッドタイムを組み込むために、ソフトウェアがステッピング モーター制御用レジスタを更新する方法を示します。ソフトウェアは、DT_BUFF レジスタのデッドタイムを更新でき、それは tc イベントで DT レジスタに転送されます。COUNT_UPDN2 モードでは、シャドー転送は TC とアンダーフローが一致する周期で起こります。

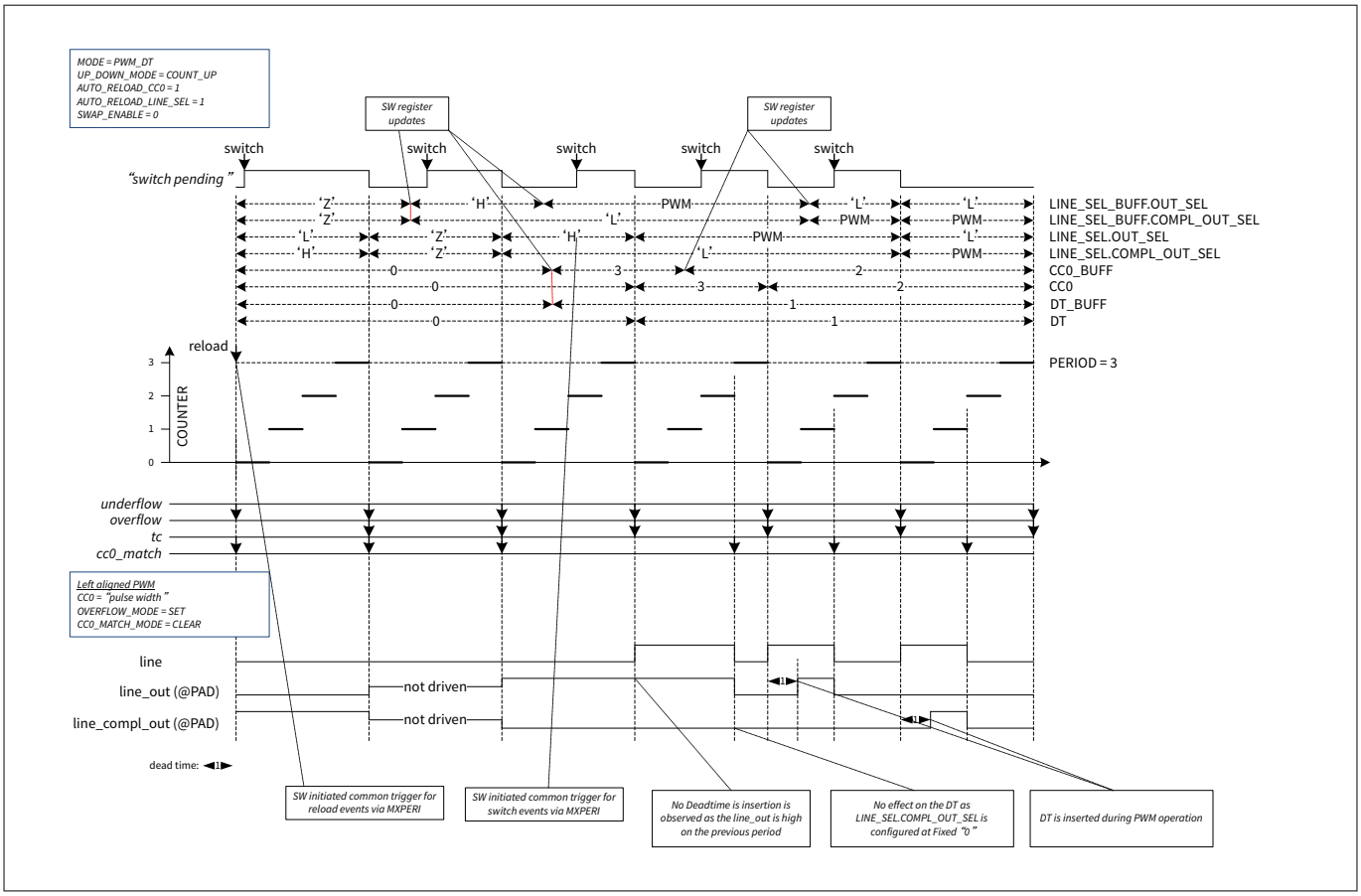


図 149　　PWM_DT 出力ケース 1 シナリオ (SWAP_ENABLE = 0 の場合)

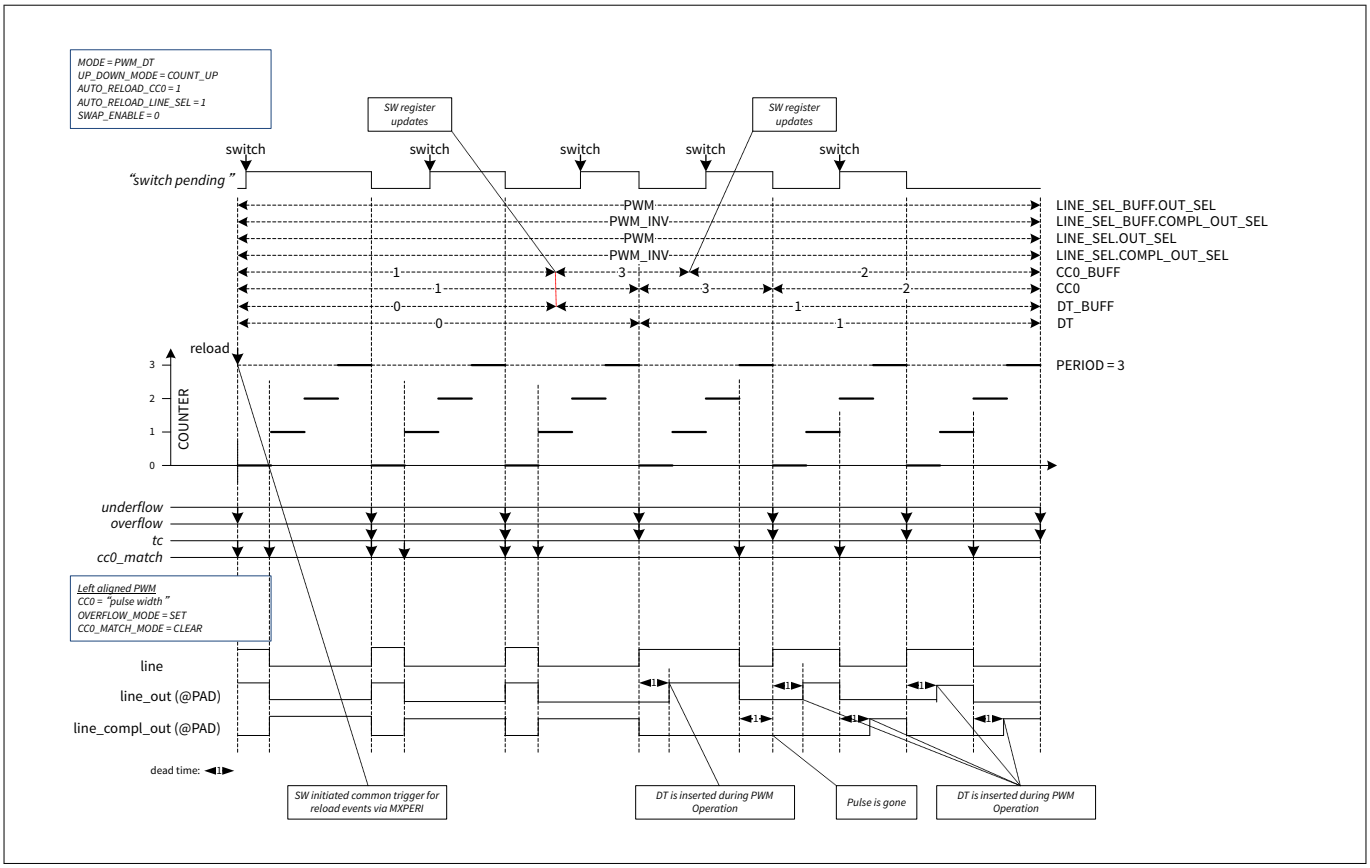


図 150　　PWM_DT 出力ケース 2 シナリオ (SWAP_ENABLE = 0 の場合)

24.3.4.1　デッドタイムモード付き PWM 用カウンターの設定方法

デッドタイム付き PWM 動作モード用にカウンターを設定する手順と、影響を受けるレジスタビットは以下のとおりです。

- CTRL レジスタの ENABLE ビットに'0'を書き込んでカウンターを無効にします。
- CTRL レジスタの MODE[26:24]フィールドに'101'を書き込んでデッドタイム付き PWM モードを選択します。
- DT レジスタの DT_LINE_OUT_L [7:0]フィールドと DT_LINE_OUT_H [15:8]フィールドに書き込んで必要なデッドタイムを設定してください。
- PERIOD レジスタに 16 ビット/32 ビット周期を設定し、必要に応じて PERIOD_BUFF レジスタにバッファ周期値を設定してください。
- CC0/1 レジスタの 16 ビット/32 ビットコンペア値と CC0/1_BUFF レジスタのバッファコンペア値を、必要に応じて値を切り替えるように設定します。
- CTRL レジスタの UP_DOWN_MODE[17:16]フィールドにカウント方向を設定し、PWM を左揃え、右揃えまたは中央揃えに設定してください。
- CTRL レジスタの PWM_IMM_KILL、PWM_STOP_ON_KILL、PWM_SYNC_KILL、PWM_TC_SYNC_KILL、PWM_SYNC_KILL_DT の各フィールドを必要に応じて設定します。
- TR_IN_SEL0 または TR_IN_SEL1 レジスタを設定して、イベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーを選択します。
- TR_IN_EDGE_SEL レジスタを設定して、イベント (Reload、Start、Stop、Capture0/1、Count) を発生させるトリガーのエッジを選択します。
- LINE_OUT および LINE_COMPL_OUT は、cc0_match、cc1_match、オーバーフロー、アンダーフローの条件でセット、リセット、または反転するように、TR_PWM_CTRL レジスタによって制御できます。

- 必要に応じて、TC または CC0/1 コンペア条件で割込みを設定します。
- CTRL レジスタの ENABLED ビットに'1'を書き込んでカウンターを有効にします。ハードウェアのリロード信号が有効化されていない場合、カウンターを開始するためには、ファームウェア (PERI0_TR_CMD レジスタの RELOAD ビット) を通じてリロードトリガーを提供する必要があります。

24.4 ディザリング

ディザリング機能は、グループ 1 とグループ 2 で利用できます。PWM および PWM_DT モードでは、これらのカウンター グループにディザリング機能が追加されます。PWM および PWM_DT モードでは、CTRL.DITHEREN によりディザリングを ON にできます。ディザリングは、周期とデューティ サイクルを独立してオンにできます。

周期ディザリングでは、リニアフィードバックシフトレジスタ (LFSR) から生成された擬似乱数が周期値に加算または減算されます。カウンター値と周期値を比較することで、PWM に似た信号が作られます。生成された信号には、通常の PWM 信号とは異なる周波数/ノイズ特性があります。

デューティ サイクル ディザリングでは、LFSR から生成された擬似乱数が CC0/CC1 の値に加算または減算されます。PWM のような信号のデューティ サイクルは、カウンターの値と CC0/CC1 の値を比較することによって作られます。生成された信号には、通常の PWM 信号とは異なるランダム デューティ サイクル特性があります。

ディザリングの機能は以下のとおりです。

- PLFSR/DLFSR レジスタの値は、CTRL.DITHEREN によってディザリングが有効になる前に、ソフトウェアによって LFSR.PLFSR/LFSR.DLFSR の 0 以外の値 (シード値) に初期化されます。
- LFSR 多項式:
  - PLFSR/DLFSR レジスタの値は 8 ビット LFSR タップ多項式に基づいて変化します。LFSR の長さは 8 ビットで、設計上固定されています。
  - カウンター値 COUNTER が PERIOD に等しい場合、tc イベントが生成されます。COUNT_UPDN1/COUNT_UPDN2 の場合、tc イベントは"PERIOD"が 2*PERIOD (1 つのオーバーフローと 1 つのアンダーフロー条件) の場合に生成されます。
  - カウンター値 COUNTER が CC0 に等しい場合、cc0_match イベントが発生します。
  - カウンター値 COUNTER が CC1 と等しい場合、cc1_match イベントが発生します。
  - ソフトウェアは、LFSR.Limiter ビットをプログラムすることによって、加算または減算される LFSR 値の大きさを制御できます。
  - PLFSR/DLFSR の MSB ビットは、PERIOD/CC0/CC1 から加算/減算される値の極性を決定します。
  - PLFSR/DLFSR の符号付き数値表現では、周期/デューティに付加されるランダム性は-2^ (N-1) から 2^ (N-1)-1 の範囲にあり、N は PLFSR/SLFSR の長さです。設計では、N は 8 です。
  - ソフトウェアが CTRL.DITHEREN でディザリング機能を有効にした場合、line_out と line_comp_out で直ちにディザリングが有効になるわけではありません。ディザリングはピリオド一致イベントで line_out と line_comp_out にのみ導入されます。
  - ディザリング機能は、ピリオド一致イベントとシーケンスのピリオド終了時に無効になります。

注: ディザリング機能を使用する場合は、以下の点に注意してください。

- *LSFR レジスタにリミッター オプションがあり、period/CC0/CC1 に加える値の大きさを制限できます。*
- *PERIOD、CC0、CC1 の値は、リミッター値より大きくなければいけません。*
- *CC0 + MAX (ディザ値) < 周期 + Min (ディザ値)、これはリミッターをとおして制御されます。*

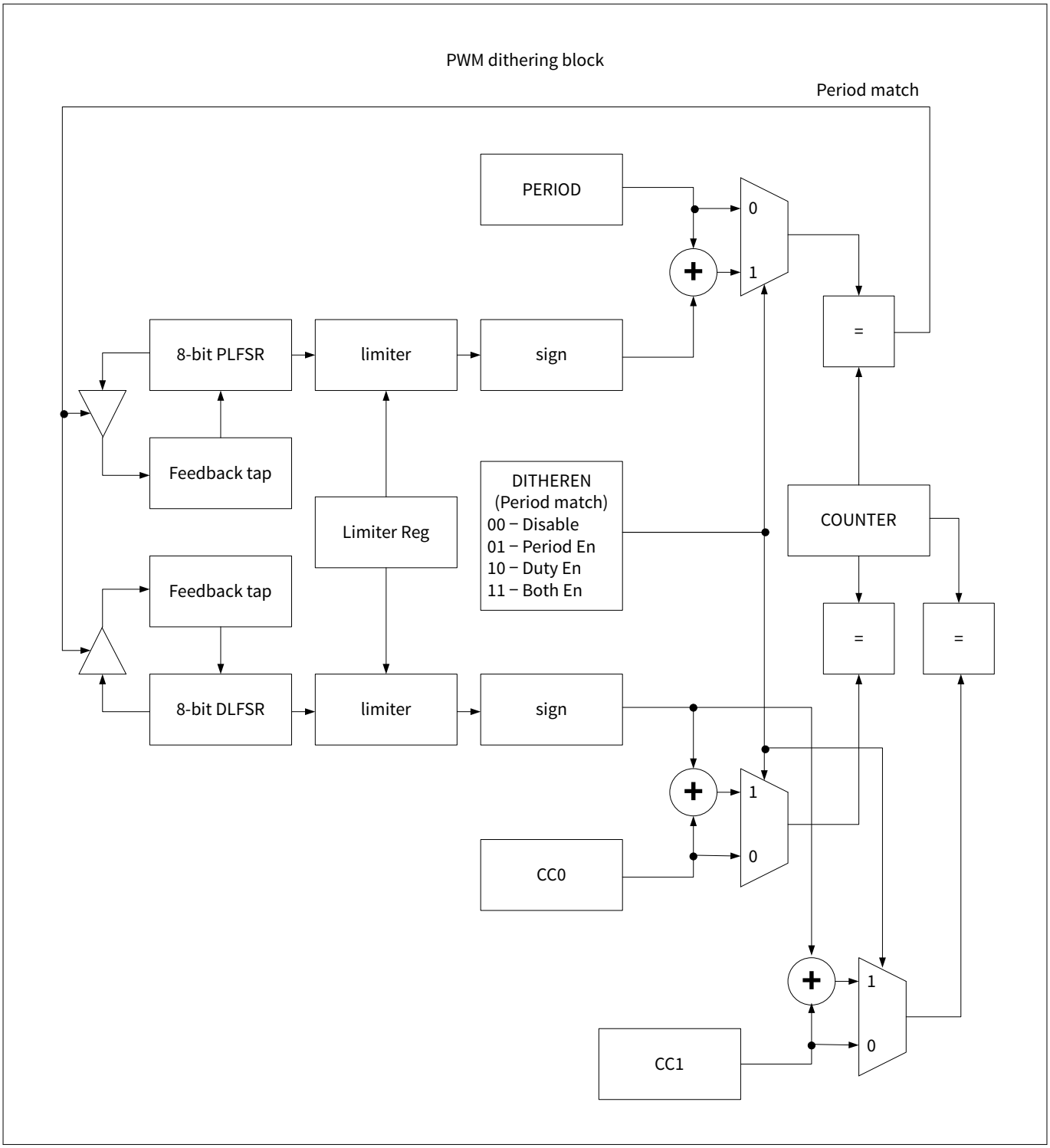


図 151　　PWM ディザリング ブロック

24.5　　モーションインターフェース (MOTIF)

24.5.1　　概要

MOTIF モジュールは 3 つの主要なサブユニットから構成されています。

- 直交デコーダユニット

- ホールセンサー制御ユニット
- マルチチャネル モード ユニット

直交デコーダーユニットは、ロータリーインクリメンタルエンコーダーと連動した位置制御に使用されます。ホールセンサー制御ユニットは、ブラシレス DC モーターの直接制御に使用されます。マルチチャネルモードユニットは、ホールセンサーモードと組み合わせて使用し、必要なモーター制御パターンを出力しますが、単体で使用することも可能で、複数の制御ユニットをシンプルにマルチチャネル制御できます。

24.5.1.1 特長

一般的な特長

- 直交デコーダ
  - 位置測定用インターフェース
  - モーター回転測定用インターフェース
  - 速度測定用インターフェース
  - 位相エラー、モーター回転、方向変更、位相デコードエラーの割込みソース
- ホールセンサー
  - ブラシレス DC モーター制御用のシンプルなビルドイン モード
  - マルチチャネルパターンのシャドーレジスタ
  - PWM 信号とマルチチャネルパターン更新の完全同期化
  - 正しいホールイベント検出、間違ったホールイベント検出のための割込みソース
- マルチチャネルモード
  - ホールセンサーモードによるシンプルな使用
  - スタンドアロン マルチチャネル モード
  - マルチチャネルパターンのシャドーレジスタ

24.5.1.2 ブロック図

MOTIF モジュールは 4 つのモードで動作できます。

- 直交デコーダ
- ホールセンサー
- スタンドアロン マルチチャネル
- スタンドアローン マルチチャネルによる直交モード

これらすべてのモードの制御/測定ループを完成させるには、MOTIF と TCPWM カウンター (複数可) をリンクさせます。各 MOTIF モジュールは専用のチャネルカウンターグループに接続します。図 152 に示すように、各 MOTIF モジュールは、いくつかの入力と出力 (サービスリクエストを含む) を含み、選択された設定 (直交デコーダー、ホールセンサー、またはスタンドアロンマルチチャネル) に応じて、利用可能なモジュール機能にマッピングできます。また、このモジュールには 2 本の専用サービス要求ラインがあります。

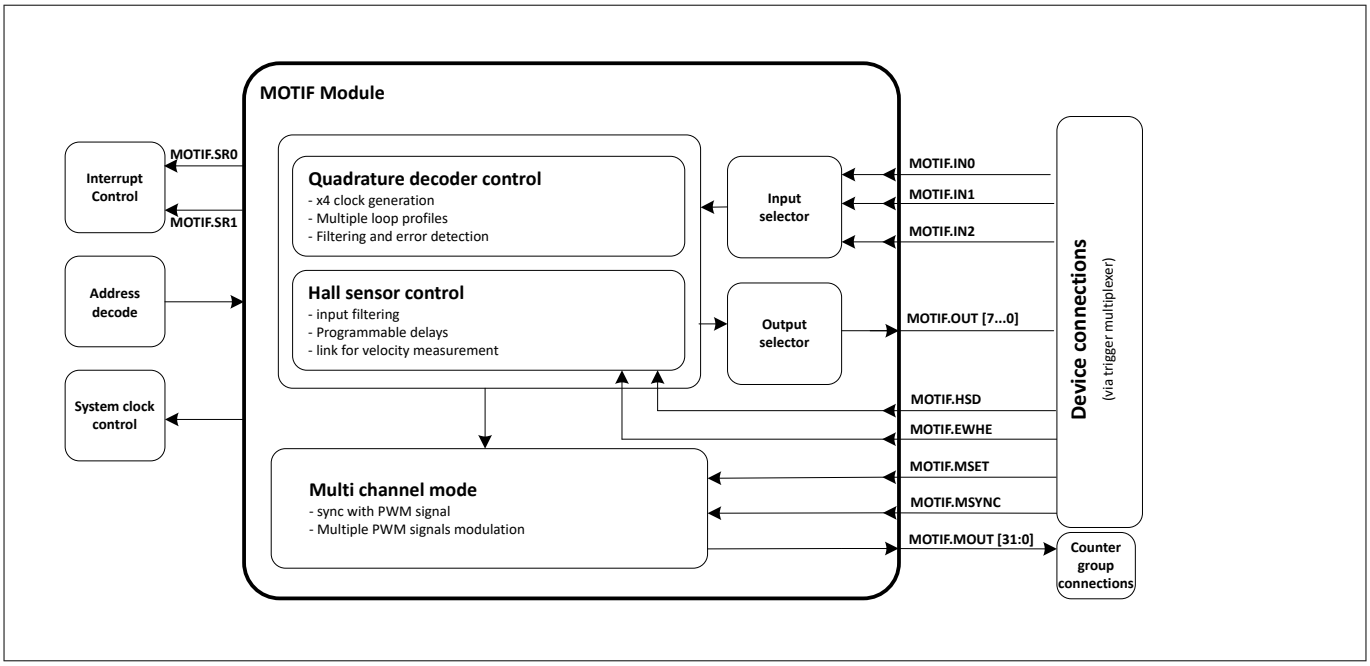


図 152　　MOTIF ブロック図

MOTIF モジュールには、直交デコーダーとホールセンサー制御ユニットの両方で並列に使用される機能セレクタ ユニットが含まれています。このブロックは、各コントロール ユニットに対して、どの入力信号をデコードするかを選択します。出力セレクタでは、モジュールの動作モードに応じて、どの機能を出力に転送するかを選択できます。MOTIF モジュールには、ホールセンサー/マルチチャネルモードでのみ使用される入力のサブセットも含まれています。これらの入力は、トリガー マルチプレクサを介して TCPWM カウンターに接続され、パターン サンプリング、マルチチャネル パターン更新、同期などの間の遅延を制御するために使用されます。

マルチチャネル ユニットには、現在のマルチチャネル パターンを含む 32 の専用出力があり、これらは TCPWM グループ#1 に接続されています。MOTIF は専用のチャネルグループカウンターに接続します。MOTIF コントロールユニットには、ソフトウェアでセット/クリアできる専用のランビットがあります。また、TCPWM カウンターに接続できる同期スタート信号を生成することもできます。

直交デコーダー モードでは、どの信号を先行相とするか、また各信号のアクティブ レベルを定義できます。直交デコーダー モードは、単純なエンコーダーで使用できる外部ソースからクロックと方向信号を直接受信することもできます。マルチチャネルでは、マルチチャネルパターンのシャドウレジスタが用意されており、これらのパラメーターをその場で更新できます。書き込みアクセスは常にシャドウ レジスタをアドレスし、読み出しは常にマルチチャネル パターンの実際の値を返します。

24.5.2　　ホールセンサー制御

ホール センサー モードは大きく 3 つのループに分かれています。

- ホール入力の更新検出
- 検出までの遅延
- 予想されるホールパターンと比較するためのホール入力のサンプリングとマルチチャネルパターンの更新

ホール入力はエッジ検出回路に直接接続されており、これら 3 つの入力のいずれかが変化すると、イベントが発生する-遷移検出- 図 153 を参照してください。このイベントは、エッジ検出とホールセンサーモードの次のステップであるホール入力のサンプリングとの間の遅延を制御するトリガー マルチプレクサを介して、1 つの TCPWM カウンターに接続できます。ホール入力のサンプル トリガーに使用される信号は、立ち上りエッジまたは立ち下りエッジでアクティブにできます (PCONF.SPES)。アプリケーション ソフトウェアは、HIST.HIS フィールド (通常はスタートアップに使用) に 1_B を書き込むことによって、ホール入力のサンプルを発行/要求することもできます。

サンプリングトリガーが感知されると、ホール入力は保持され、現在のパターン HALP.HCP、予想されるホールパターン HALP.HEP、および前のパターン HALP.HPP と比較されます (前のパターン比較は無効にできます)。

エッジ検出回路はホール入力をサンプリングするためのイネーブル信号 PIFHSP を生成し、サンプルロジックは新しい値がキャプチャされるたびにパルス PIFHRDY を生成し、パターン比較ロジック内で使用されます。

HMEC.HDBP ビットフィールドを 1_B に設定することにより、外部サンプルトリガーをバイパスすることもできます。これにより、ホール遷移が検出された場合、フィルタリング ステージの後にホール入力の即時サンプルがトリガーされます。ホール入力も、MOTIF スタートビットが立ち上りエッジを検出するとすぐにサンプリングされます。ソフトウェアは、LPF 出力が有効になってから MOTIF を起動する必要があります。

サンプリングされた値が期待されるホールパターンと一致すると、イベント/パルスが生成され、正しいホールイベントとなり、サンプリングされたパターンが期待されるものと一致したことを示します。同時に、シャドー レジスタにプログラムされた値がロードされます。HALP.HCP[LSB]はホール入力 1 にリンクされ、HALP.HCP[MSB]はホール入力 3 にリンクされます (HALP.HEP レジスタも同様)。

HALP.HCP、HALP.HEP、および HALP.HPP フィールドの更新は、MCMS.STHR に 1_B を書き込むことによってソフトウェアから行えます。

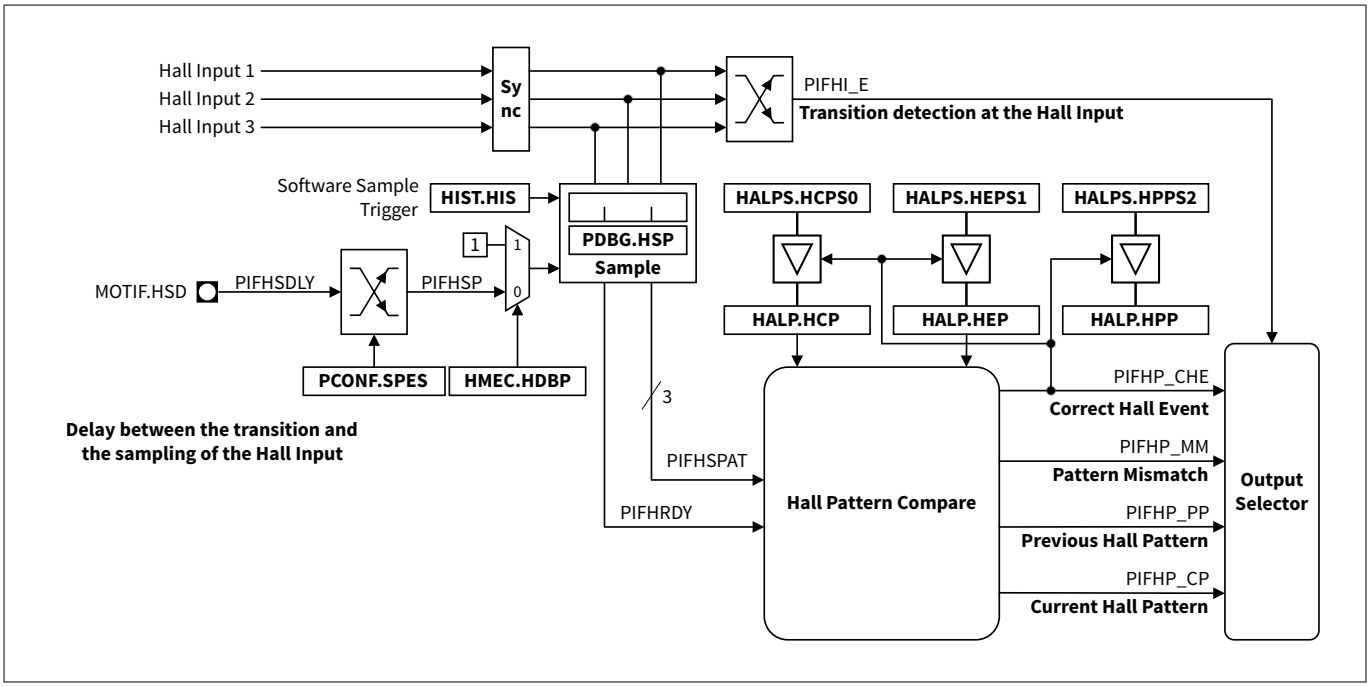


図 153　　ホールセンサー制御図

サンプリングされた値が現在のホールパターンと一致した場合、ライングリッチが発生したとみなされ、何も処置は取られません。現在のホール パターンとの比較の他に、第 3 の比較ルールである前のホール パターンを使用することも可能です。このルールは、HMEC.HPPE ビットフィールドを 1_B に設定することで有効になります。この比較により、サンプリングされたパターンが HALP.HPP でプログラムされた値と一致する場合、間違ったホールイベントは発生しません。これは、モーターが停止位置にある場合に重要で、トルクが小さすぎると前のホールパターンが MOTIF 入力で感知される可能性があります。

サンプリングされた値がどの値 (現在のパターン、予想されるパターン、以前のパターン) とも一致しない場合、重大なエラーが発生したとみなされ、誤ったホールイベント信号が生成されます。サンプリングされたパターンが間違ったホール イベントにつながるたび、または現在のパターンや以前のパターンに一致するたび (これは有効にも無効にもできる)、パターン不一致イベントが生成され、(出力セレクタを介して) モジュール出力ピン図 154 の 1 つに転送できます。

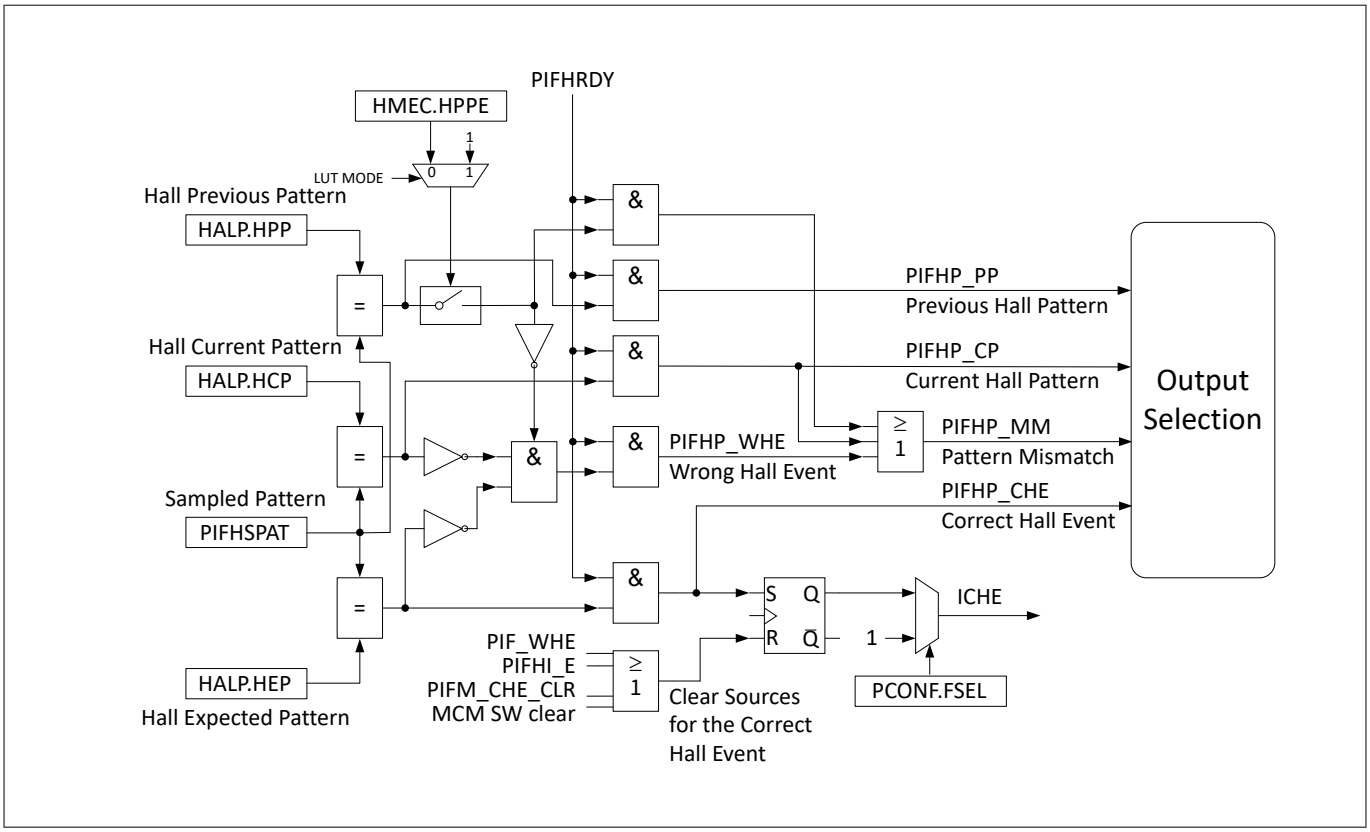


図 154　　ホールセンサー比較ロジック

フォールトホールイベントは、MOTIF モジュールでもデコードできます。このフォールト ホール イベントは、ホール比較ロジックからの間違ったホール イベントとプログラマブル モジュール入力ピン図 155 の選択可能な組み合わせです。

このフォールトホールイベントは、出力セレクタを介して MOTIFx.OUTy モジュールピンの 1 つに転送でき (例えば、PWM ユニットで強制停止動作を実行するため)、また、マルチチャネルパターンをクリアするために使用することもできます - 図 156。

ホールセンサーモードでフォールト イベントが検出された場合、ハードウェアは安全機構として MOTIF を停止する必要があります。この安全機構機能は LUT モードでのみ適用されます。ソフトウェアが動作し、フォールト イベントが検出された後に MOTIF を再設定して再度起動する必要があります。

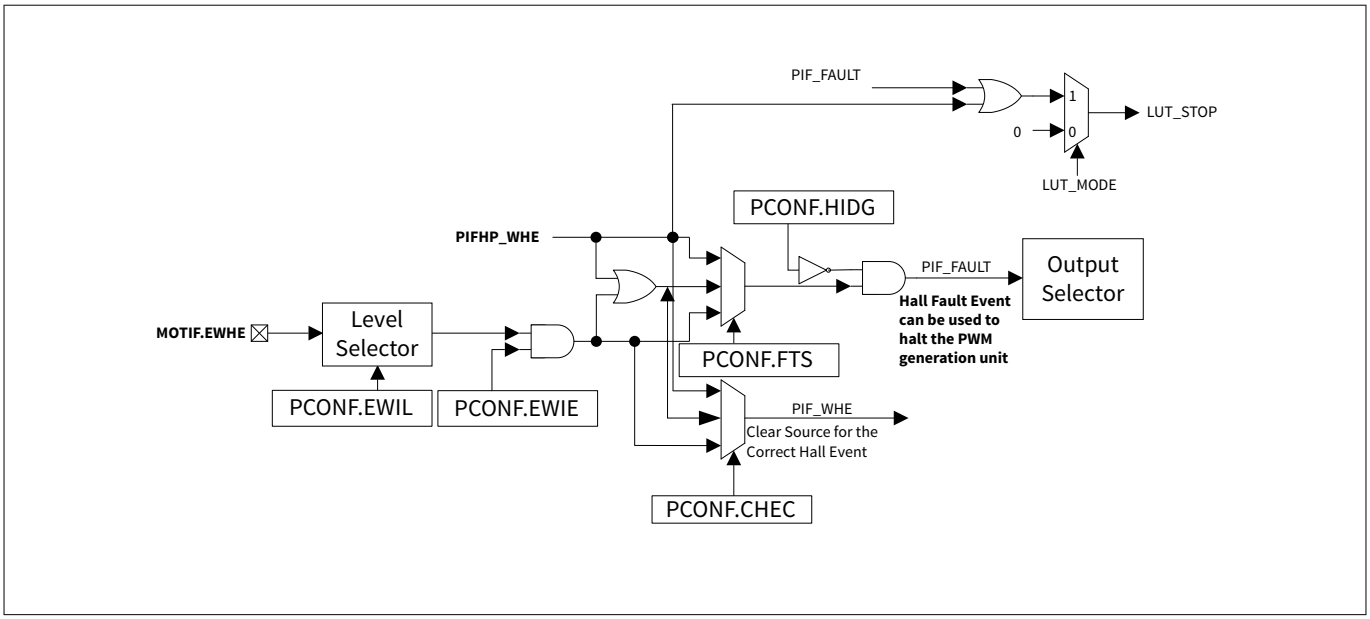


図 155　　ホール フォールト イベント/アイドル ロジック

正しいホールイベントが検出された後、この検出とマルチチャネルパターンの更新の間に遅延を発生できます。図 156 に、マルチチャネルモードの制御ロジックを示します。

マルチチャネルパターンの更新遅延は、TCPWM カウンターによって直接制御できます。遅延が終了したことを示すトリガーは、入力 MOTIFx.MSET にマッピングされます。また、PCONF.MSES でトリガーのアクティブエッジを選択することもできます。この遅延挿入は、MCMEC.MEBP ビットフィールドを 1_B に設定することでバイパスできます。

PCONF.MCUE フィールドは、マルチチャネルパターンの更新を可能にするソースを選択します。1_B に設定されている場合、マルチチャネルパターンは、ソフトウェアが MCMS.MNPS フィールドに 1_B を書き込んだ後にのみ更新できます。

更新遅延 (有効な場合) の後、マルチチャネルパターンはまだ PWM 信号と同期する必要があります。これには、入力 MOTIFx.MSYNC が使用されます。また、PCONF.MSYES でトリガーのアクティブエッジを選択することもできます。この信号で設定されたエッジが検出されると、MCM.MCMP[31]が MOTIFx.MOUT[31]に、MCM.MCMP[0]が MOTIFx.MOUT[0]にリンクされ、MOTIFx.MOUT[31:0]出力で新しいマルチパターンが適用されます。この同期段階は、MCMEC.MSBP を 1_B に設定することでバイパスすることもできます。

マルチチャネル パターン更新要求トリガーは MCMF.MSS レジスタ フィールドに接続され、出力セレクタを介してモジュール ピンに転送できます。このレジスタフィールド (MSS) は、マルチチャネルパターンの更新 (同期信号 PIFMSYNC の受信時に行われる) を有効にします。マルチチャネルパターンは、マルチチャネル PWM 出力の同期更新を行うために TCWPM カウンターと共に使用されます。MCM.MCMP の更新は、MCMS.STMR に 1_B を書き込むことで、ソフトウェアがトリガーできます。

また、ホール フォールト イベントを使用して、事前に設定された値 (MCPF.MCFV) でマルチチャネル パターンをクリア/ロードすることも可能です。このメカニズムは、誤操作が検出され、出力スイッチを制御している PWM タイマーに特定の安全パターンを適用する必要がある場合に使用できます。

32 ビット マルチチャネル パターン出力は、2 つの異なる方法で関連カウンターの PWM 出力に影響を与えられます。

- AND ゲート PWM - PWM を強制的にパッシブにします
- OR ゲート PWM - PWM を強制的にアクティブにします。

各カウンターは、マルチチャネルパターン出力によって駆動される 4 つの信号を受信し、line_out と line_comp_out で変調を実行します。1 つの信号が OR か AND かを指示し、もう 1 つの信号が機能の 2 番目の入力として使われます。

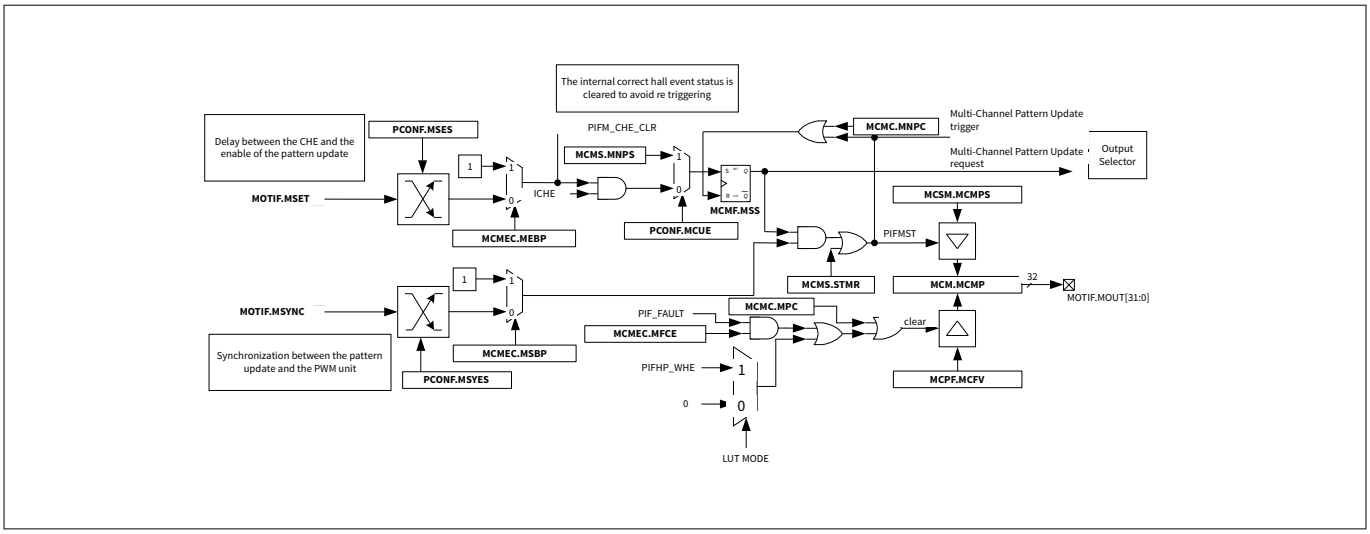


図 156　　マルチチャネル モード図

図 157 に、タイマードメイン内のホール センサー モードの説明で、前述したすべてのステップを示します。ホール入力の遷移が検出されるたびに、信号 (遷移検出) がアサートされます。この信号は、遷移検出とホール入力のサンプリングとの間のタイミング遅延を開始するために使用されます。

このシナリオでは、TCPWM カウンターのステータス出力 (図 157 の TR_OUT0) がタイミング 1 (ホール入力の遷移とサンプリングの間の遅延) を制御するために使用されました。TR_OUT0 信号の立ち上りエッジでホール入力がサンプリングされ、それらが予想されるパターンと一致すれば、信号 (正しいホール イベント) がアサートされます。

同じ TCPWM カウンターにマッピングされた期間一致イベント出力 (TR_OUT1 信号) は、正しいホールイベントとマルチチャネルパターンの更新との間の遅延を制御するために使用されます。このサービス要求は、期間の一致が検出されたときにアサートされます。

もう一つの TCPWM カウンターは、各正しいホールイベント間の時間を測定するために使用されます。このタイマー セルは図 157 のタイミング 2 で表されます (CR はカウンター ブロックのキャプチャ レジスタを象徴します)。

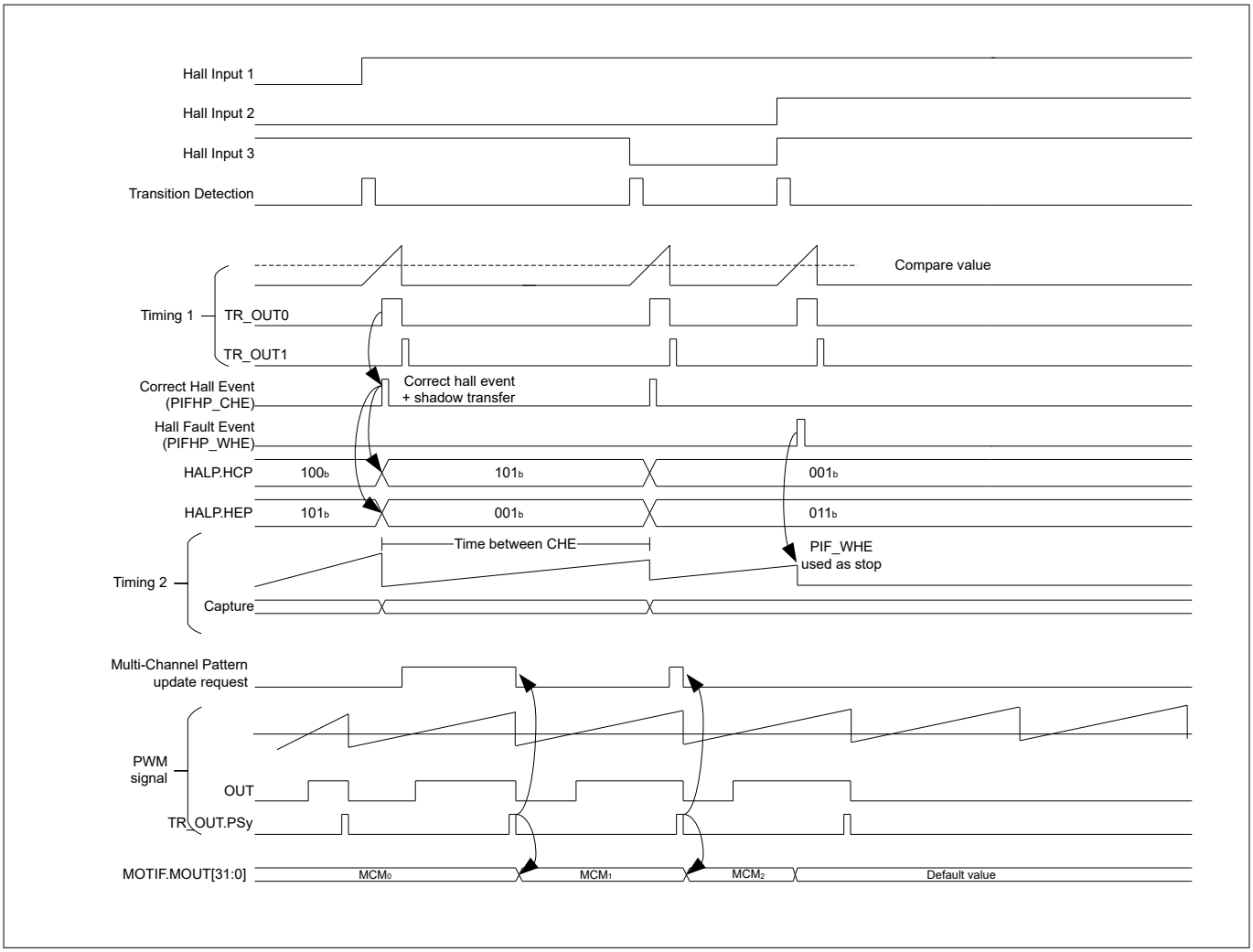


図 157　　ホールセンサータイミング図

LUT サポートによるホールセンサー動作

ホールセンサーは、ルックアップテーブル (LUT) モードで動作できます。このモードは、半自律的な方法で典型的なモーター制御段階の整流の CPU 負荷を軽減することを目的としています。LUT モードは、完全にハードウェアベースのフィードバック制御ループで、ソフトウェアによって支援されます。モーターが同じ方向に動く限り、ハードウェアはマルチチャネル パターン出力、MCM.MCMP を完全に制御します。出力パターンは、ソフトウェアによって事前にロードされた LUT から取得されます。モーターの向きが変わると MOTIF は停止し、ソフトウェアは LUT の内容を再入力して再び MOTIF を起動する必要があります。

各 MOTIF モジュールには 2 つの LUT が含まれています。ホール LUT (HLUT、「入力 LUT」とも呼ばれます) と変調 LUT (MLUT、「出力 LUT」とも呼ばれます)。HLUT はホール入力パターンを含み、HALP レジスタにロードされます。MLUT は出力 PWM 変調パターンを含み、MCM.MCMP レジスタにロードされます。

各 MOTIF には LUT 制御ロジックがあり、LUT の動作を支援するために使用されます。以下のパラメーターはこのアクションに関連付けられています。

- LUT 制御コンフィギュレーション

CLUT.LUTEN -> LUT モードの有効化/無効化

- LUT ステータスおよびデバッグ レジスタ

SLUT.PTR -> 内部ポインタ値の状態

- MLUT (出力 LUT)

MCSM, MCSM1, MCSM2, MCSM3, MCSM4, MCSM5

- HLUT (入力 LUT)

HALPS.HCPS0[2:0], HALPS.HEPS1[2:0], HALPS.HPPS2[2:0], HALPS.HLUT3[2:0], HALPS.HLUT4[2:0], HALPS.HLUT5[2:0]

下図は、LUT 動作モード時の主要ロジックの基本図です。LUT モードが有効な場合、LUT 制御ロジックは HLUT と MLUT からそれぞれ HALP と MCM レジスタにロードされる値を決定します。LUT モードが無効の場合、HALP および MCM レジスタのロードは、ホールセンサー標準動作セクションで説明されているように、固定レジスタ位置から行われます。

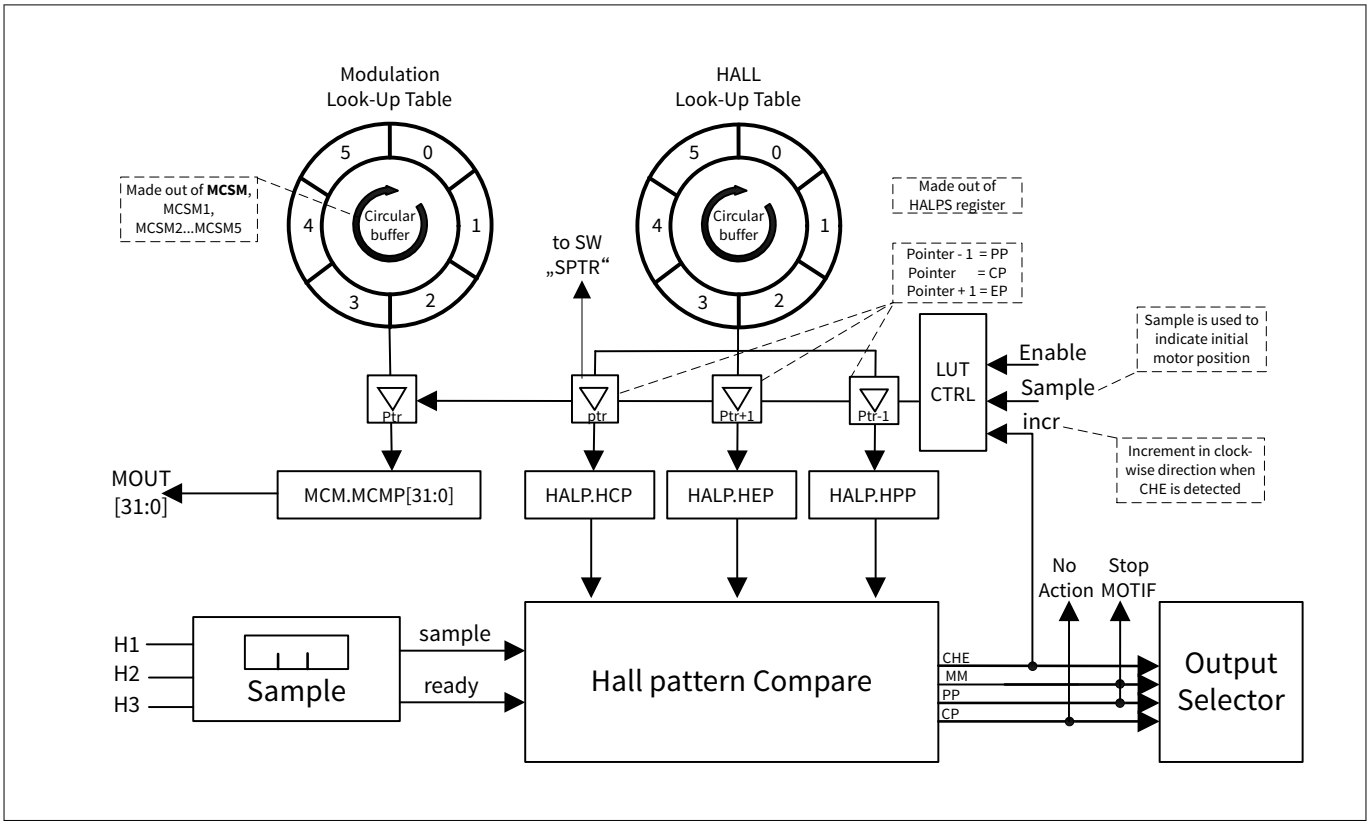


図 158　　LUT サポート付きホールセンサーモード

LUT 動作はハードウェアベースのループですが、以下に説明されているように、いくつかの条件下ではソフトウェアによる支援が必要です。

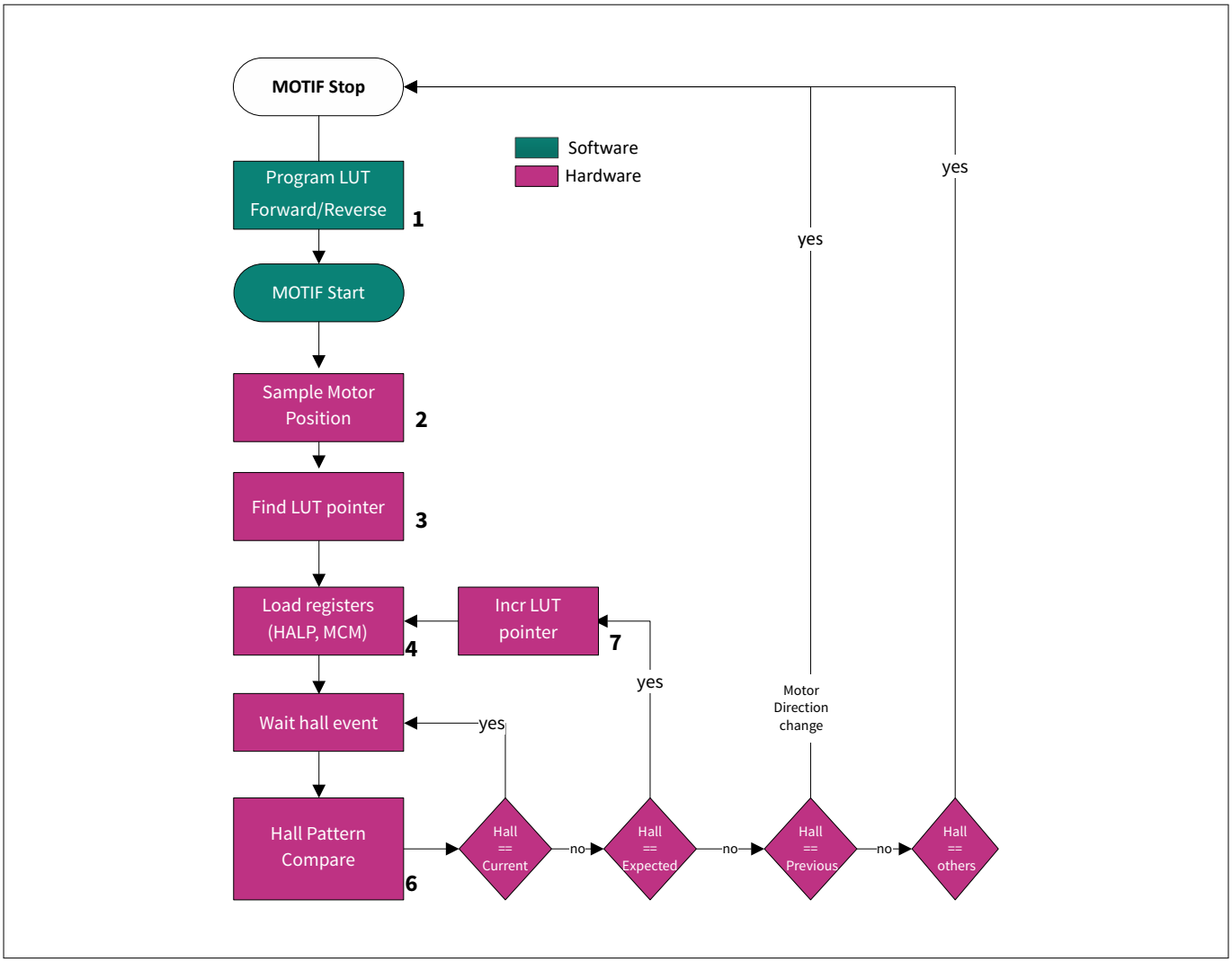


図 159 LUT モード - イベントのシーケンス

ステップ 1: プログラム LUT

MOTIF 停止時にソフトウェアで HLUT と MLUT を書き込んでください。MOTIF を起動した状態で両方の LUT に書き込むと、動作がおかしくなる可能性があります。LUT には、モーターが望む方向に応じて、順方向または逆方向のパターンがロードされます。パターンはモーター位置とは無関係に LUT に格納されます。ハードウェアは、現在のモーター位置を見つけ、適切なホールと変調パターンをロードする責任があります。

HLUT と MLUT はデータ抽出のために同じハードウェア ポインタを共有するため、ソフトウェアは LUT アドレスに対して両者の整合を保つように注意する必要があります。誤ったプログラミングは、誤った行動につながるかもしれません。例えば、現在のモーター位置が HLUT (x) に一致する場合、出力変調は同じインデックスである MLUT (x) に位置する必要があります。ソフトウエアは、それらのアライメントを維持することに気を配らなければいけません。

LUT はサーキュラーバッファとして機能し、0 から 5 までの 6 つのアドレスを時計回り方向にのみ進みます。HLUT は繰り返し値を持つことはできません。6 つの値はすべて一意であることが期待されます。現在のパターンが HLUT (x) に格納されている場合、予想されるパターンは HLUT (x+1) に、前のパターンは HLUT (x-1) に格納されていなければいけません。MLUT も同様です。

ステップ 2: モーターポジションサンプル

HALL 入力のサンプルは、MOTIF モジュールが起動し、HALL モードが有効になるとハードウェアによって自動的に行われ、これは現在のモーター位置を得るために行われます。MOTIF 起動後に LUT モードを有効にすると、

動作がおかしくなることがあります。これはソフトウェアで防がなければいけません。モーター位置のサンプルは、MOTIF モジュールが起動し、LUT モードが有効になってから 2 クロック後に利用できます。

ステップ 3: LUT ポインタを探す

ホールサンプルの後、ハードウェアは現在のモーター位置 PDBG.HSP (STEP 2 から入手可能) を 6 つの HLUT 値すべてと並行して比較します。LUT ポインタは、内容が現在のモーター位置に一致するアドレスに対応します。有効な比較は 6 つのうち 1 つのみと予想されます。なぜなら、HLUT の値はすべて一意であることが期待されているからです。

6 つすべての HALL LUT 値がモーターの現在位置と異なる場合、パターン不一致イベントが生成されます。ハードウェアが MOTIF を止めます。パターン不一致イベントは、変調フォールト安全パターンをロードするためにソフトウェアを使用できます。割込みを発生させられます。現在のパターンに対応する LUT アドレスポインタ"PTR"が更新され、ソフトウェアから SLUT.PTR レジスタを通じてアクセスできます。可能な値は、0、1、2、3、4、および 5 です。無効な値は 6、7 です。この動作は、MOTIF が有効化された後の始動段階 (停止から始動への移行) で一度のみ実行されます。LUT のインデックスを検索し整合させることが目的のため、この段階では LUT ポインタはインクリメントされません。

ステップ 4: ロード レジスタ (HALP、MCM)。

HALP および MCM レジスタを LUT ポインタ インデックスでロードするために、自動シャドウ転送が行われます。正しいホールイベントが検出された後、この検出とマルチチャネルパターンの更新の間に遅延を発生できます。図 156 に、マルチチャネルモードの制御ロジックを示します。

マルチチャネルパターンの更新遅延は、TCPWM カウンターによって直接制御できます。遅延が終了したことを示すトリガーは、入力 MOTIFx.MSET にマッピングされます。また、PCONF.MSES でトリガーのアクティブエッジを選択することもできます。この遅延挿入は、MCMEC.MEBP ビットフィールドを 1_B に設定することでバイパスできます。

PCONF.MCUE フィールドは、マルチチャネルパターンの更新を可能にするソースを選択します。1B に設定すると、ソフトウェアが MCMS.MNPS フィールドに 1B を書き込んだ後にのみ、マルチチャネルパターンを更新できます。

更新遅延 (有効な場合) の後、マルチチャネルパターンはまだ PWM 信号と同期する必要があります。これには、入力 MOTIFx.MSYNC が使用されます。また、PCONF.MSYES でトリガーのアクティブエッジを選択することもできます。この信号で設定されたエッジが検出されると、出力ルックアップテーブルから新しいマルチパターンが MOTIFx.MOUT[31:0]出力に適用されます。この同期段階は、MCMEC.MSBP を 1_B に設定することでバイパスすることもできます。

ステップ 5: ウェイト ホールイベント

サンプリングトリガーが感知されると、ホール入力が保持され、現在のパターン HALP.HCP、予想されるホールパターン HALP.HEP、および前回のパターン HALP.HPP と比較されます。

ステップ 6: ホールパターンの比較

ホール入力==予想されるパターンの場合:

- MCUE = 0 はソフトウェアで設定する必要があります。
- MOTIF は、LUT を更新するソフトウェアなしで自由に動作します。
- CHE が生成され、LUT から次の値が HALP、MCM にロードされます。
- 出力変調 MCM.MCMP = MCSM
- ステップ 7 に進んでください

ホール入力==現在のパターン

- MOTIF は、LUT を更新するソフトウェアなしで自由に動作します。
- 何もしない (信号線上にグリッチが発生したと判断されます)
- 出力変調 MCM.MCMP = 値の保持
- ステップ 5 に戻ってください

ホール入力==前のパターンの場合:

- MOTIF がハードウェアにより停止
- モーターの方向転換を示す PHP がソフトウェアに生成されます
- 出力変調 MCM.MCMP = 値の保持
- ステップ 0 に戻る (MOTIF を停止)

ホール入力!=現在のパターン、期待されるパターン、前のパターン

- MOTIF がハードウェアにより停止
- WHE はソフトウェアに生成されます
- 出力変調 MCM.MCMP = MCPF.MCFV (セーフパターン)
- ステップ 0 に戻る (MOTIF を停止)

ステップ 7: LUT ポインタのインクリメント

正しいホールイベントがデコードされると、LUT ポインタが 1 インクリメントされます。PTR は常に現在のホールパターンを指します。

- PTR = 現在のパターン
- PTR + 1 = 予想されるパターン
- PTR - 1 = 前のパターン

現在のパターンに対応する LUT アドレスポインタ"PTR"が更新され、ソフトウェアから SLUT.PTR レジスタを通じてアクセスできます。可能な値は、0、1、2、3、4、および 5 です。無効な値は 6、7 です。この操作は、正しいホールイベントが検出されるたびに実行されます。

以下の表は、LUT モードに影響または関連する以下のレジスタをまとめたものです。

表 182 LUT 動作のレジスタ関係

レジスタ	説明	LUT モードの使用
PCONF.FTS	LUT モードでは、動作にフォールトホールイベント信号を使用できます。	LUT モードで使用可能
PCONF.CHEC	LUT モードは正しいホールイベントクリア選択を使用できます	LUT モードで使用可能
PCONF.HIDG	LUT モードでは、PCONF.HIDG = 1 のときのみ WHE 信号を使用します。 PCONF.HIDG = 0 の場合、LUT モードは WHE と PIF_FAULT を使用します。	LUT モードで使用可能
PCONF.EWIE	外部の誤ったホールイベントのイネーブル (PCONF.FTS と共に使用) 0 - 外部の誤ったホールイベントエミュレーション信号 (MOTIFx.EWHE) が無効 1 - 外部の誤ったホールイベントエミュレーション信号 (MOTIFx.EWHE) が有効	LUT モードで使用可能

(続く)

表 182 (続き) LUT 動作のレジスタ関係

レジスタ	説明	LUT モードの使用
PCONF.MCUE	PCONF.MCUE フィールドは、マルチチャネルパターンの更新を可能にするソースを選択します。1B に設定すると、ソフトウェアが MCMS.MNPS フィールドに 1B を書き込んだ後にのみ、マルチチャネルパターンを更新できます。 マルチチャネル パターン ソフトウェア アップデート可能 0 - マルチチャネルのパターン更新はハードウェアで制御されます 1 - マルチチャネル パターン アップデートをソフトウェアで制御	LUT モードで使用可能
HMEC.HPPE	LUT モードには影響を与えず、ハードウェアは動作に WHE 信号を使用します。	LUT モードでは使用できません。
MCMEC.MNPS	マルチチャネルパターンは、PCONF.MCUE が 1 に設定されている場合、ソフトウェアが MCMS.MNPS フィールドに 1 を書き込んだ後にのみ更新できます。	LUT モードで使用可能

24.5.3 直交デコーダ制御

直交デコーダー モードは、PCONF.FSEL = 01_B または PCONF.FSEL = 11_B を設定することによって選択されます (この場合、マルチチャネル モードも有効になります)。

直交デコーダー モードでは、2 つの異なるサブセットが利用できます。

- 標準直交モード
- 方向カウントモード

標準モードは、外部ロータリーエンコーダーが 2 つの位相信号を提供し、さらにシャフト 1 回転につき 1 回生成されるインデックス/マーカ信号を提供する場合に使用されます。方向カウント モードは、外部エンコーダーがクロックと方向信号のみを供給する場合に使用します。

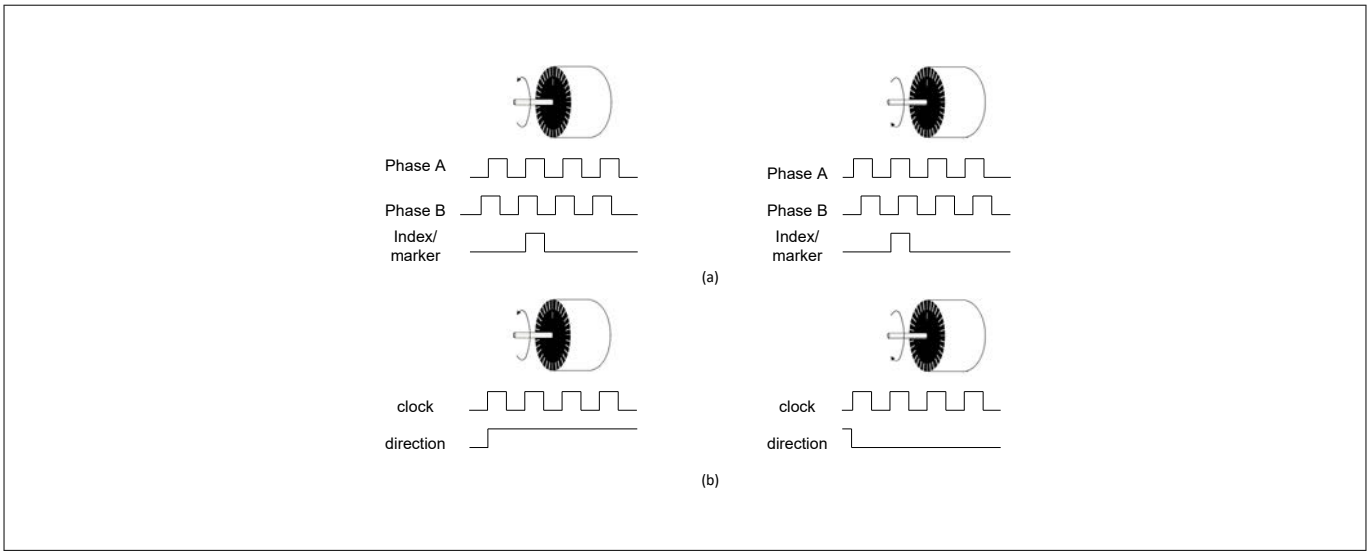


図 160　　ロータリーエンコーダーのタイプ - a) 標準 2 相＋インデックス信号 b) クロック＋方向

標準直交モード

直交デコーダー ユニットは、非常に柔軟な A 相/B 相設定ステージを提供します。通常、時計回りのモーターシャフトの回転では、A 相が B 相に先行するはずですが、それにもかかわらず、図 161 のように各信号の特定のアクティブ状態と同様に、先行する相を設定できます。

A 相信号、B 相信号、およびインデックス信号の選択は、QDC.PACS、QDC.PBDS、および QDC.IDS 制御フィールドを介して行われます。直交デコーダー制御ユニットを構成するブロックは、直交 (Quadrature) クロックとモーター軸の方向をデコードするブロックと、インデックス (モーター回転) 制御を処理するブロックの 2 つに大別されます。

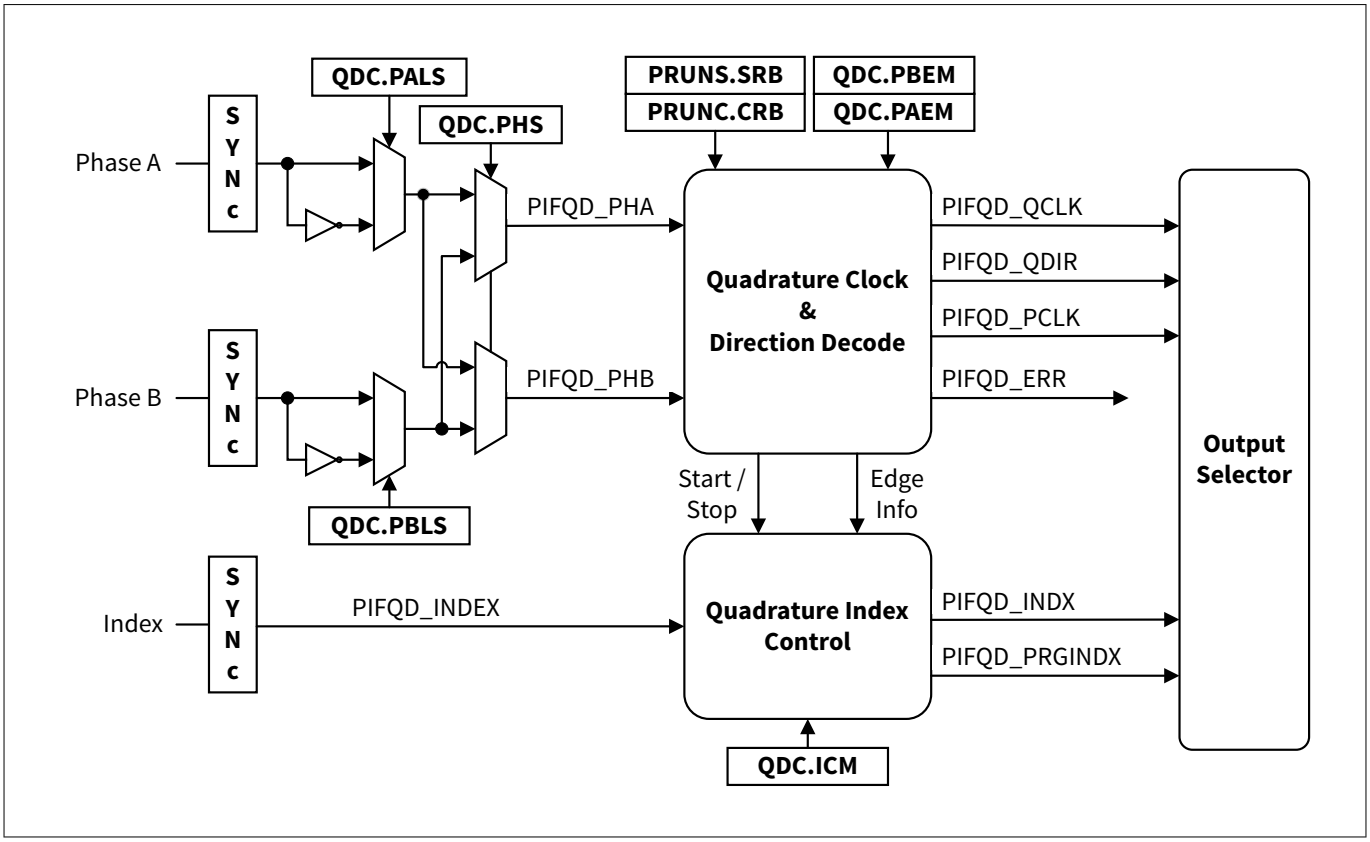


図 161　　直交デコーダー制御の概要

直交クロックは位置測定に使用でき、MOTIFx.OUTy 出力のいずれかに転送できます。このクロックは位相信号の各エッジからデコードできるため、1 位相期間あたり 4 クロックパルスとなります。直交クロック信号のデコードに使用される (A 相信号と B 相信号の) エッジは、アプリケーション ソフトウェアで設定できます。

以下の図に、A 相信号と B 相信号からクロックをデコードするいくつかの例を示します。どのエッジを立ち上げるか、立ち下げるか、両方立ち上げるか、あるいは立ち上げないかの設定は、QDC.PAEM と QDC.PBEM (A 相と B 相に対応) を通じて行われます。QDC.PHS が'1'に設定されている場合、QDC.PAEM は B 相の設定に使用され、QDC.PBEM は A 相の設定に使用されることに注意してください。

次の図に、X1 クロック生成の動作を示します。

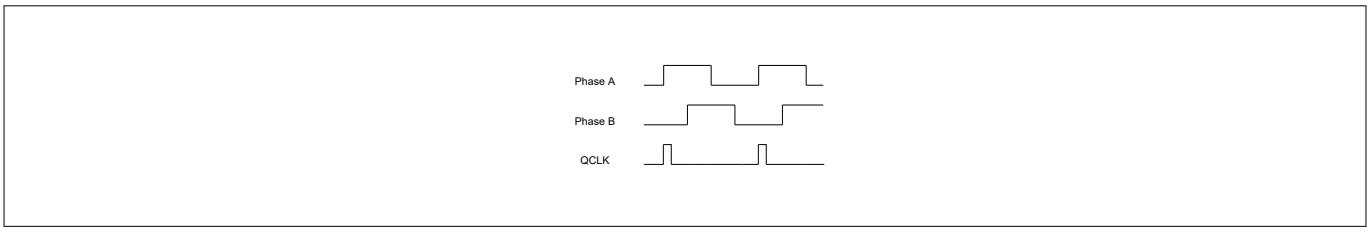


図 162　　X1 クロック生成 PAEM=RISE/PBEM=NONE

次の図に、X2 クロック生成の動作を示します。

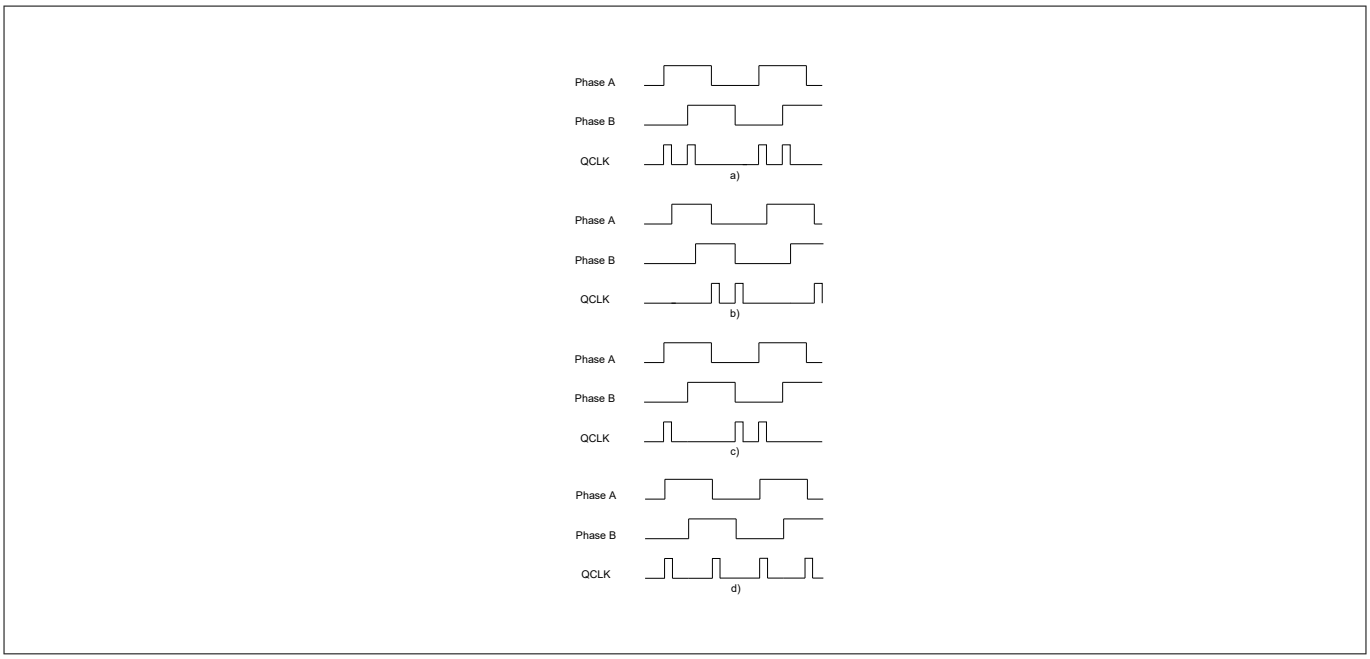


図 163　　X2 クロック生成: a) PAEM=RISE/PBEM=RISE; b) PAEM=FALL/PBEM=FALL; c) PAEM=RISE/PBEM=FALL; d) PAEM=BOTH/PBEM=NONE

次の図に、X4 クロック生成の動作を示します。

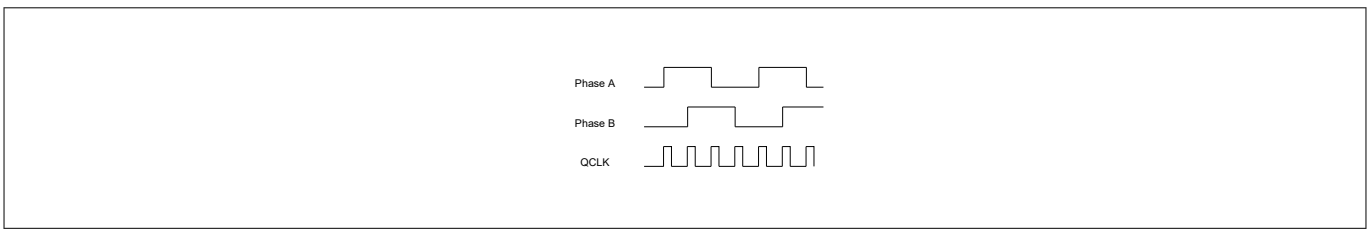


図 164　　X4 クロック生成 PAEM=BOTH/PBEM=BOTH

周期クロックも速度測定動作用に生成され、MOTIFx.OUTy 出力のいずれかに転送できます。このクロックのデコードは QDC.PAEM と QDC.PBEM フィールドの影響も受けます。モーターの回転方向は、A 相信号と B 相信号

(QDIR) からデコードされ、MOTIFx.OUTy 出力のいずれかに転送することもできます。方向信号は、モーターが時計回りに回転しているときに HIGH にアサートされ、反時計回りに回転しているときに LOW にアサートされます。

インデックス コントロール ロジックは、インデックス信号のアサート後の最初のエッジを保持しているため、同じ直交遷移がインデックス イベント動作に使用されます。また、回転インクリメント信号がいつアサートされるべきかを制御できるように、最初のインデックスの方向も保持します。MOTIFx.OUTy 出力には 2 つのインデックスイベント (INDX イベントと PRGINDX イベント) の転送ができます。

誤った位相ペアが検出された場合、フラグに接続されたエラー信号 (有効の場合は割込みも生成可能) を生成することもできます (このイベントは割込みとしてのみ利用可能で、MOTIFx.OUTy 出力に転送できません)。

直交デコーダー制御は、方向とクロックをデコードするために、現在の位相ペアと前の位相ペアの情報を使用します。両相信号はエッジ検出ロジックを通過し、その出力は有効/無効遷移ステージに対して使用されます (図 165 を参照)。PCONF.ENABLE フィールドに 1_B を書き込むことで、デコーダーのステートマシンをリセットできる可能性があります (フラグやリテンション mmio レジスタはリセットできません)。直交デコーダーの現在の状態、前の状態、および現在のインデックス値は、PDBG レジスタを介して読み出せます。

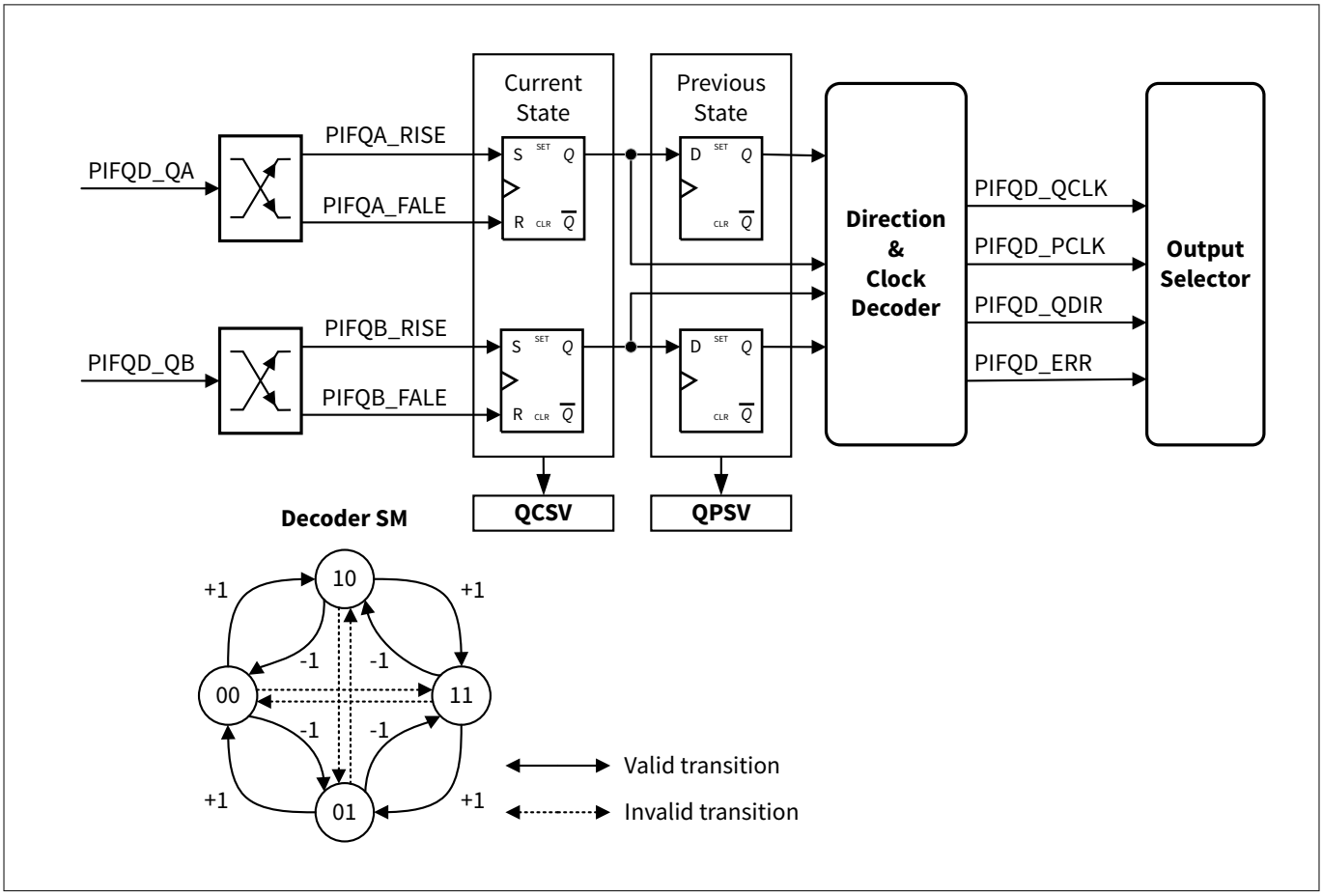


図 165　　直交デコーダの状態

方向カウントモード

位置エンコーダーの中には、出力として位相信号を持たないものがあります。その代わりに、クロックと方向情報を含む 2 つの信号を出力するものがあります。これはダイレクト カウント モードとして知られています。

このタイプの位置エンコーダーを使用する場合、ユーザーは PCONF.QDCM ビットフィールドを 1_B に設定する必要があります (方向カウントモードを有効にします)。この場合、クロック信号と方向信号をどの入力グループ (MOTIFx.INy) に接続するかを選択する必要があります。これは QDC.PACS と QDC.PBDS コントロールフィールドを介して行われます。入力信号 (方向とクロック) は MOTIF モジュールのクロックに同期され、出力セレクタを介して MOTIFx.OUTy 出力のいずれかに転送される前に、有効であればローパスフィルター処理されます。方向入力

は必須ではありません。QDC.PBDS を 00_B に設定することで無効にできます。このモードでは出力信号 PCLK、INDX、PRGINDX、ERR は一定のゼロに設定されます。

24.5.3.1 直交クロックと方向デコード

直交デコーダー ユニットは 2 つのクロックを出力します。1 つのクロックが位置制御に使用されるため、位相信号 QCLK の各エッジ遷移で生成されます。2 番目のクロックは速度測定に使用され、モーターの振動の原因となる非常に遅い回転速度で発生することのある位相信号のグリッチの影響を受けません - PCLK。

モーターの振動によるグリッチは、必ずしも電気的なノイズのグリッチではありません。このタイプの振動グリッチは、入力フィルターでグリッチを抑制できないほど遅いです。図 167 を参照してください。

モーターの方向が変わる (QDIR で遷移する) と PCLK は生成されません。ただし、モーター運転開始時は例外です。PCLK は A 相または B 相の最初の遷移に対して生成されます。

方向信号のデコードは図 165 のルールに従って行われます。図 166 に、A 相信号と B 相信号のすべての有効な遷移を示します。図 168 に、位相信号にグリッチがある場合の 2 つのクロック信号の差を示します。

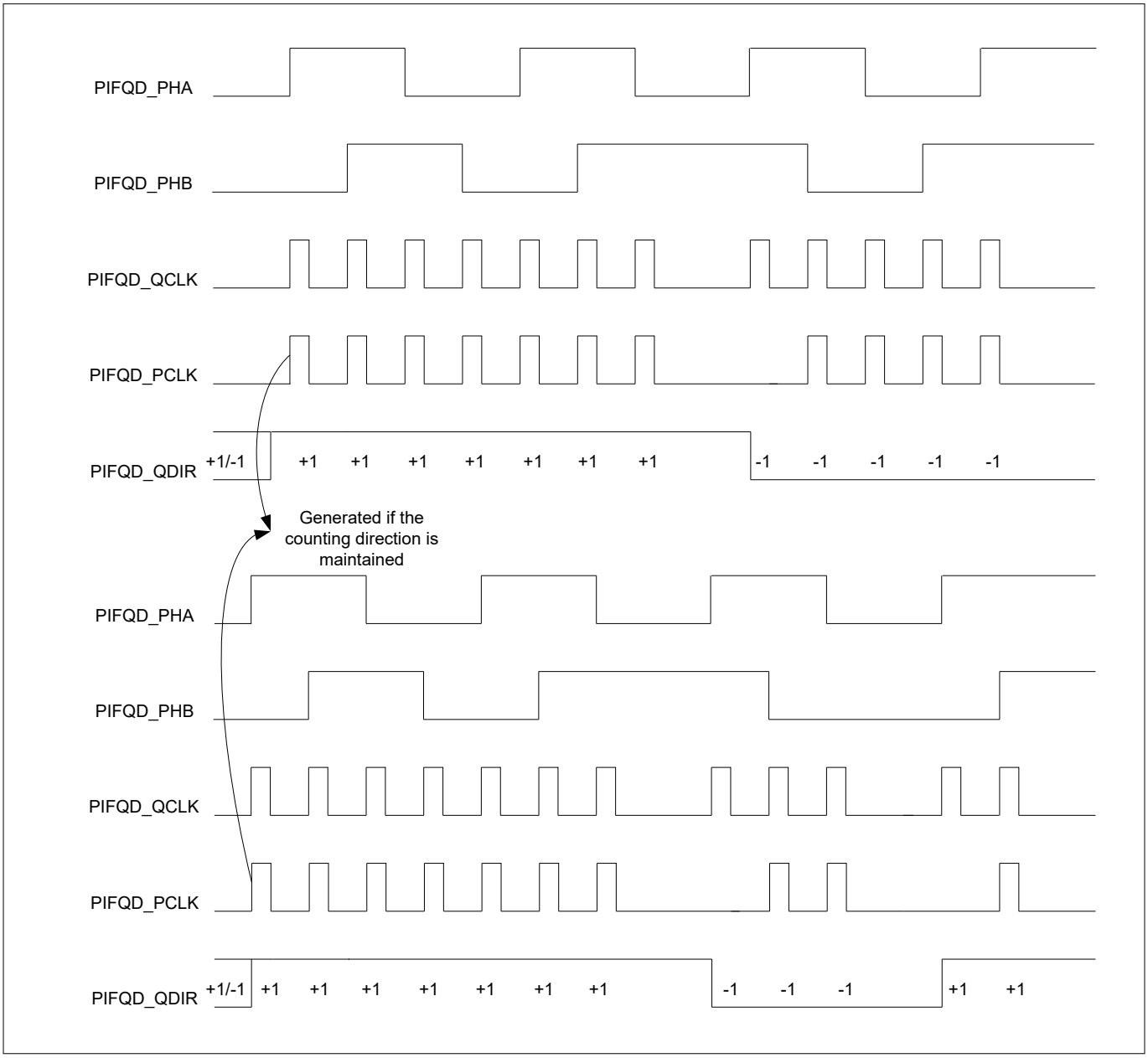


図 166 直交クロックと方向タイミング

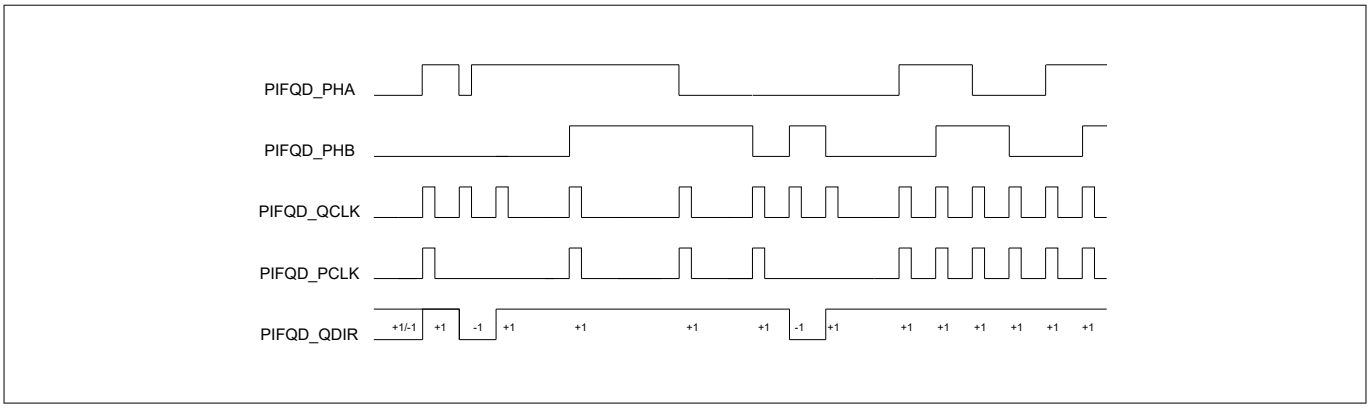


図 167　　ジッター付き直交クロック

24.5.3.2　インデックス制御

インデックス コントロール ロジックには、2 つの異なる出力イベントがあります。1 つは、インデックス信号が検出される (そしてモーターシャフトの回転が同じです) たびにアサートされる-INDX-ので、回転カウンターに使用できます。ソフトウェアは、シャフトの位置のみでなく、発生した総回転数もモニターできます。

もう 1 つのイベント (PRGINDX) のアクティビティは、QDC.ICM フィールドを介してプログラムできます。QDC.ICM で設定された値に応じて、この信号を生成できます。

- すべてのインデックス シグナルの発生において
- 最初のインデックス信号発生時のみ
- 無効 - この出力はアサートされません

これは、すべてのインデックス イベントでカウンターをリセットする必要があるアプリケーションや、最初のインデックス イベントでのみカウンターをリセットする必要があるアプリケーションに便利です。

インデックス コントロール ロジックは、インデックスに続く次の位相エッジを保持するため、生成される信号は常に同じ基準になります。インデックス後の最初の相エッジが B 相信号の立ち上りエッジであれば、次のインデックスイベントでは、方向が維持されていれば B 相信号の立ち上りエッジで、方向が変化していれば B 相信号の立ち下りエッジでインデックス信号が生成されることになります。

***注**:　x2 クロック生成の場合 (例えば、A 相または B 相の立ち上りエッジのみが直交クロックの生成に使用される場合)、インデックスデコーディングは上述と同じ動作に従ってください。インデックス信号パルスのロスがありません。図 168 にインデックス信号の生成タイミング図を示します。この場合、QDC.ICM = 10_B となり、PRGINDX イベントが入力インデックスの出現毎に発生することになります。*

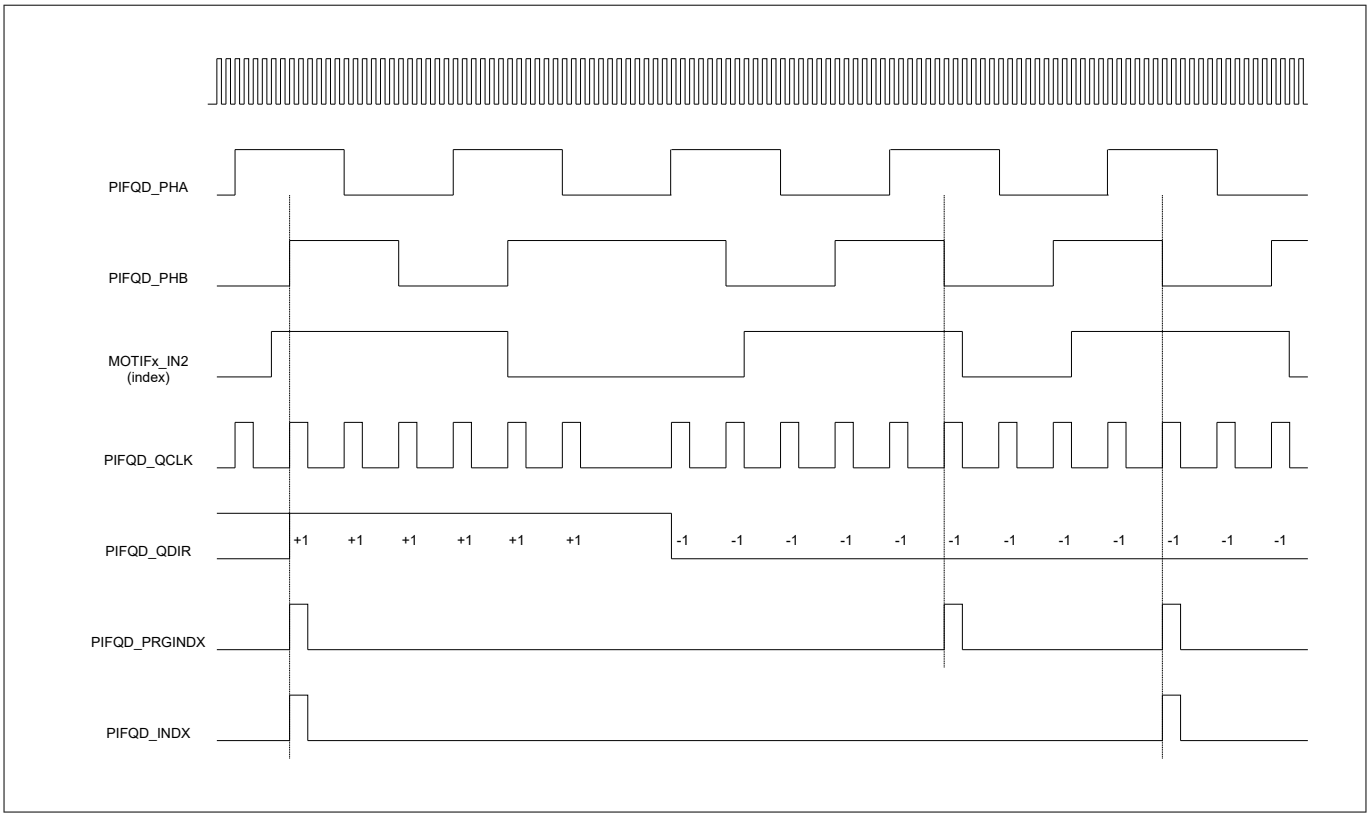


図 168　　インデックス信号のタイミング

24.5.4　スタンドアロン マルチチャネル モード

PCONF.FSEL = 10_B に設定することで、ホールセンサー制御なしでマルチチャネルモードを使用できます。スタンドアロンマルチチャネルモードは、マルチチャネルパターン出力を更新するための 2 つのシャドウ転送メカニズムをサポートする必要があります。

シングルシャドウ転送

シングルシャドウ転送の伝達メカニズムは、ホールセンサー制御の項で説明したものと同じです。パターン更新のトリガーは、MCMS.MNPS に 1_B を書き込むことによってソフトウェアから来ることも、外部信号から来ることもできます。この外部信号は、PCONF.MSETS フィールドで適切な値を選択することにより、PIFMSET 関数にマッピングされなければいけません (図 156)。

また、MCMEC.MSBP を 0_B に設定することで、新しいマルチチャネルパターンの更新と制御信号の同期の有効化ができます-入力信号 MOTIFx.MSYNC はこの機能に使用されます(図 156)。

マルチチャネルパターンは、ソフトウェアが MCMC.MPC ビットフィールドに 1_B を書き込むか、または MOTIFx.EWHE モジュールピン でフォールトが検出されたときに有効になると、MCM.MCMP に MCPF.MCFV レジスタで事前にプログラムされた値でクリア/ロードできます (図 155)。MCPF.MCFV にあらかじめプログラムされた値が MOTIFx.MOUT[31:0]出力に適用されます。

ルックアップテーブル シャドー転送

スタンドアローン マルチチャネル モードは、ルックアップテーブル (出力 LUT) による出力パターン変調の更新をサポートする必要があります。この動作を有効にするためには、LUTEN レジスタを'1'にプログラムする必要があります。

出力 LUT はソフトウェアでプログラム可能で、最大 6 つの異なる変調パターンを格納でき、各パターンは 32 ビットで構成されます。ハードウェアは一度に 1 つの変調パターンを MOTIF.MOUT[31:0]ピンに適用し、これはパターン更新のトリガーを受信するたびに発生します。

下図に、LUT と MOUT ピンがどのようにリンクしているかを示します。

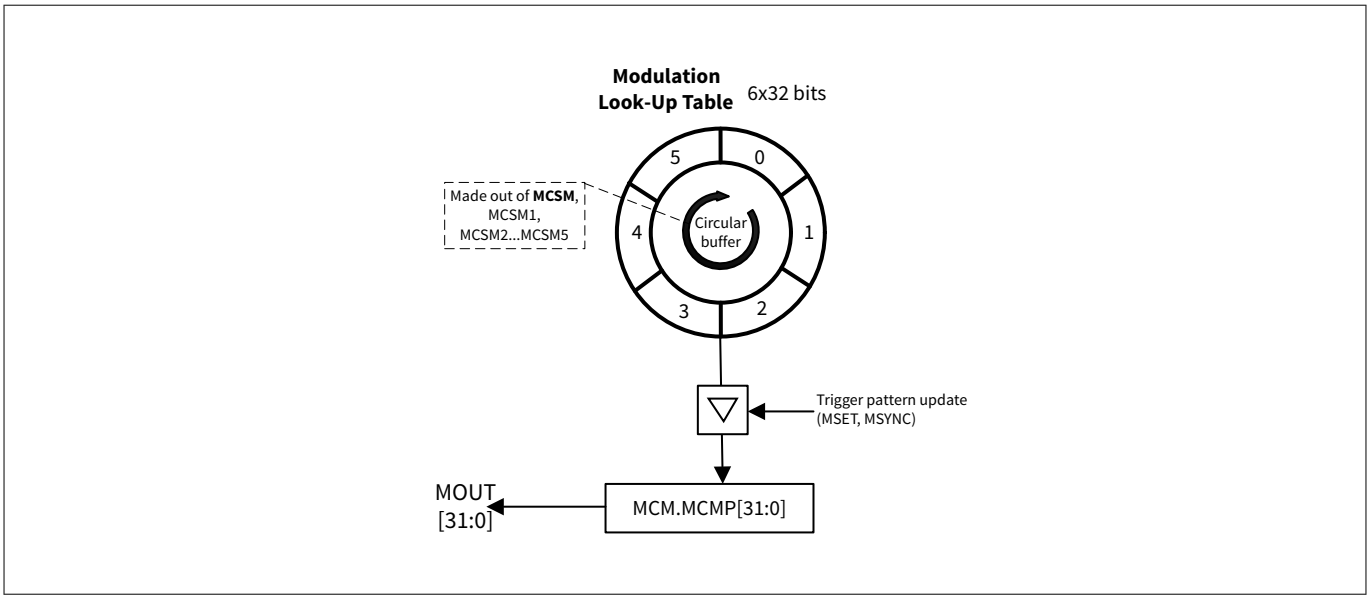


図 169　　LUT 対応スタンドアロン マルチチャネル

パターン更新のトリガーは、MCMS.MNPS に 1_B を書き込むことによってソフトウェアから来ることも、外部信号から来ることもできます。この外部信号は、PCONF.MSETS フィールドで適切な値を選択することで、PIFMSET 関数にマッピングされます。MCMEC.MSBP を 0_B に設定することで、新しいマルチチャネルパターンの更新と制御信号の同期を有効にすることもできます - この機能には入力信号 MOTIFx.MSYNC が使用されます。

ハードウェアは、循環的な方法で LUT コンテンツにアクセスする必要があります。ポジション 0 から始まり 5 まで、そしてまた 0 から繰り返します。パターン更新のトリガーが受信されるたびに、LUT アドレスが 1 ずつインクリメントされます。

LUT アドレスと LUT データレジスタ間のマッピングは以下のとおりです。

- LUT [0]は MCSM に対応
- LUT [1]は MCSM1 に対応
- LUT [2]は MCSM2 に対応
- LUT [3]は MCSM3 に対応
- LUT [4]は MCSM4 に対応
- LUT [5]は MCSM5 に対応

以下のタイミング図に、LUT を有効にしたスタンドアロン マルチチャネルの動作を示します。MSET、MSYNC 状態は LUT アドレスを 1 インクリメントし、LUT[x]の内容を MOUT[31:0]出力に転送するために使用されます。マルチチャネル パターンは、ソフトウェアが MCMC.MPC ビットフィールドに 1_B を書き込むたびに、MCPF.MCFV レジスタで MCM.MCMP にプリプログラムされた値でクリア/ロードできます。MOTIFx.EWHE 入力イベントを受信するたびに MOTIF モジュールは停止し、MCPF.MCFV に事前にプログラムされた値が MOUT[31:0]出力に適用されます。この動作のためにソフトウェアは PCONF.HIDG = 0、PCONF.FTS = 0/2、MCMEC.MFCE = 1 に設定する必要があります。

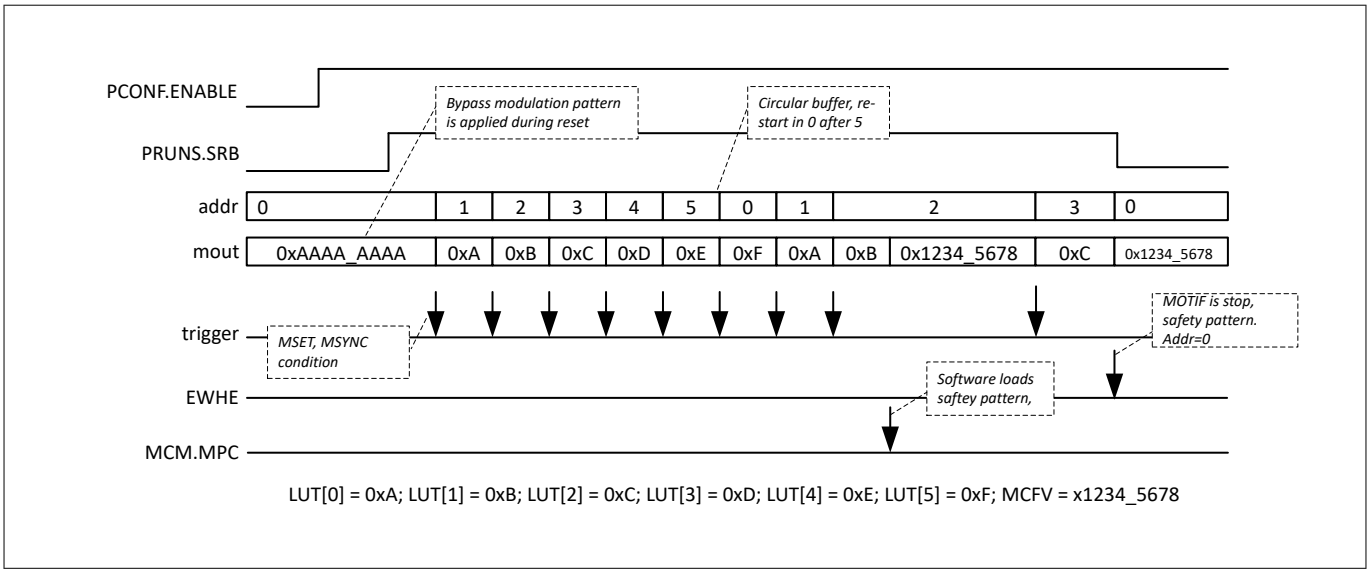


図 170　　LUT をサポートしたタイミング スタンドアロン変調

注:　　シングルシャドウ転送モードでは、MOTIF.MOUT 端子へのシャドウ転送は常に MCSM レジスタから行われます。ソフトウェアは、シャドウ転送後にこのレジスタに新しいパターンを確実に再書き込みするする必要があります。

LUT モードでは、ソフトウェアは出力 LUT を一度のみプログラムし、ハードウェアはアドレス 0 から始まる 6 アドレスをループします。MOTIF が停止すると、プロセスは停止します。

24.5.5　　同期スタート

MOTIF モジュールには同期スタートイベントがあり、出力セレクタを介して MOTIFx.OUTy 出力のいずれかに転送できます。このイベントは、TCPWM カウンターと併用することで、両モジュールの完全な同期スタートが可能です。同期スタートは MOTIF モジュールのラン ビットとリンクしており、ラン ビットが設定されるたびに遷移が生成され、いずれかの出力に転送されます。

同期スタート出力を使用することにより、ソフトウェアは MOTIF カウンターと TCWPM カウンターをスタートさせるために 2 つの独立したアクセスを実行せず、したがって、両方のモジュールが全く同時に動作を開始することが保証されます。

24.5.6　　MOTIF の使い方

MOTIF モジュールは、(カウンター/タイマーを内蔵していないため) 可能な各モードでフルセットの機能を実行するためには、TCPWM カウンターモジュールとリンクする必要があります。

MOTIF を直交デコーダーモードで動作させるには (少なくとも) 5 つのカウンターが必要です。ホールセンサーモードでは、ブラシレス DC モーターを制御するために少なくとも 3 つのカウンターが必要で、また遅延を制御するために (少なくとも) 2 つのカウンターが必要です。スタンドアローン マルチチャネル モードのリンク設定は、ユースケースに大きく依存するため、使用するカウンターの数はユーザーが自由に選択できます。

ホールセンサーモードの使用:

MOTIF のホールセンサーモードを使用する場合、マルチチャネルモードは TCWPM カウンターにマルチチャネルパターンを提供するためにも機能します。カウンター モジュールはマルチチャネル PWM 変調を行います。このため、PWM 信号の信号調整ソースとして MOTIF を選択するよう、LINE_SEL レジスタを介してカウンターを設定する必要があります。

図 171 では、ホールセンサーモードは 2 つのカウンターグループ、合計 5 つのカウンターで使用されます。グループ 1 は、アドバンス モーター コントロール機能 (3 つのカウンター) で設定されています。グループ 0 は、標準

的な汎用カウンター (2 つのカウンター) として設定されます。グループ 0 カウンターは、ホール入力のホール遷移検出と実際のサンプリングとの間の遅延を制御するために使用され、また、正しいホールイベントとマルチチャネルパターン更新イネーブルとの間の遅延を制御するためにも使用されます。

グループ 0 カウンター 0 tr_out0 の立ち上りエッジが最初の遅延の終了トリガーとして使用されます。MOTIF 入力 HSD に接続。グループ 0 カウンター 0 tr_out1 はホール入力サンプリングから変調パターン更新までの遅延時間として使用され、MOTIF 入力 MSET に接続されます。グループ 0 カウンター 0 はシングルショットモードに設定されており、MOTIF からの要求が発生するたびにタイム遅延を再トリガーできます。

グループ 0 カウンター 1 はキャプチャモードで使用され、正しいホールイベント間の時間をキャプチャします (2 つの正しいホールイベント間のモーター速度をこのように保存します)。MOTIF の正しいホールイベントはカウンターのキャプチャトリガーとして使用され、一方、パターン不一致イベントはグループ 0 カウンター 0 とグループ 0 カウンター 1 の停止として使用されます。Capture トリガーと Stop トリガーは、特定のタイマー スライスで、立ち上りエッジでアクティブになるように設定されます。

グループ 1 カウンターはモーターを制御する PWM 信号を生成するために使用されるため、マルチチャネル パターン出力 MOTIFx.MOUT[31:0]はこのユニットにリンクされています。

マルチチャネルループを閉じるには、グループ 1 カウンターの出力を MOTIF モジュールに接続し (tr_out0 期間一致)、マルチチャネルパターンが PWM 期間に同期して更新されるようにする必要があります。すべての MOTIF 入出力イベントはトリガー マルチプレクサを介して接続され、MOTIF 変調出力は TCPWM カウンター グループ 1 へのハードウェア接続を介して内部的に接続されます。

カウンターの設定:

- 2 カウンター グループ
  - G0_CNTx 標準カウンター (2 カウンター)
  - G1_CNTx、高度なモーター制御 (3 カウンター)

G0_CNT0 はホール入力サンプリングの遅延と MOTIF への MSET 信号の生成に使用されます (ワンショットタイマーモード)。

- G0_CNT1 はキャプチャモードで使用します。
- G1_CNT[2:0]は 3 相 PWM (PWM 右揃えモード) の生成に使用されます。

MOTIF の設定

- OUT0 = エッジ検出
- OUT1 = 正しいホールイベント
- OUT2 = パターンの不一致
- OUT3 = 間違ったホールイベント
- OUT4 = スタート

MOTIF <> TCPWM の接続性

- OUT0 は、1 対 1 トリガーを介して G0_CNT0 のリロードに使用されます。
- OUT1 は、1 対 1 トリガーで G0_CNT1 をキャプチャするために使用されます (**TCPWM にはキャプチャ クリア機能がないので、G0_CNT1 をリロードするためにも使用されます**)。
- OUT2 はグループトリガーで G0_CNTx を停止させるために使用されます。
- OUT3 はグループトリガーで G1_CNTx を停止させるために使用されます。
- OUT4 はグループトリガーによる G1_CNTx のリロードに使用されます。
- G0_CNT0 TROUT0 (CC0 一致) は HSD にルーティングされます。
- G0_CNT0 TROUT1 (周期一致) が MSET にルーティングされます。
- G1_CNT0 TROUT0 (周期一致) は MSYNC にルーティングされます。

PWM と MOTIF は、OUT4 経由の RUN ビットを利用することで同期して起動します。

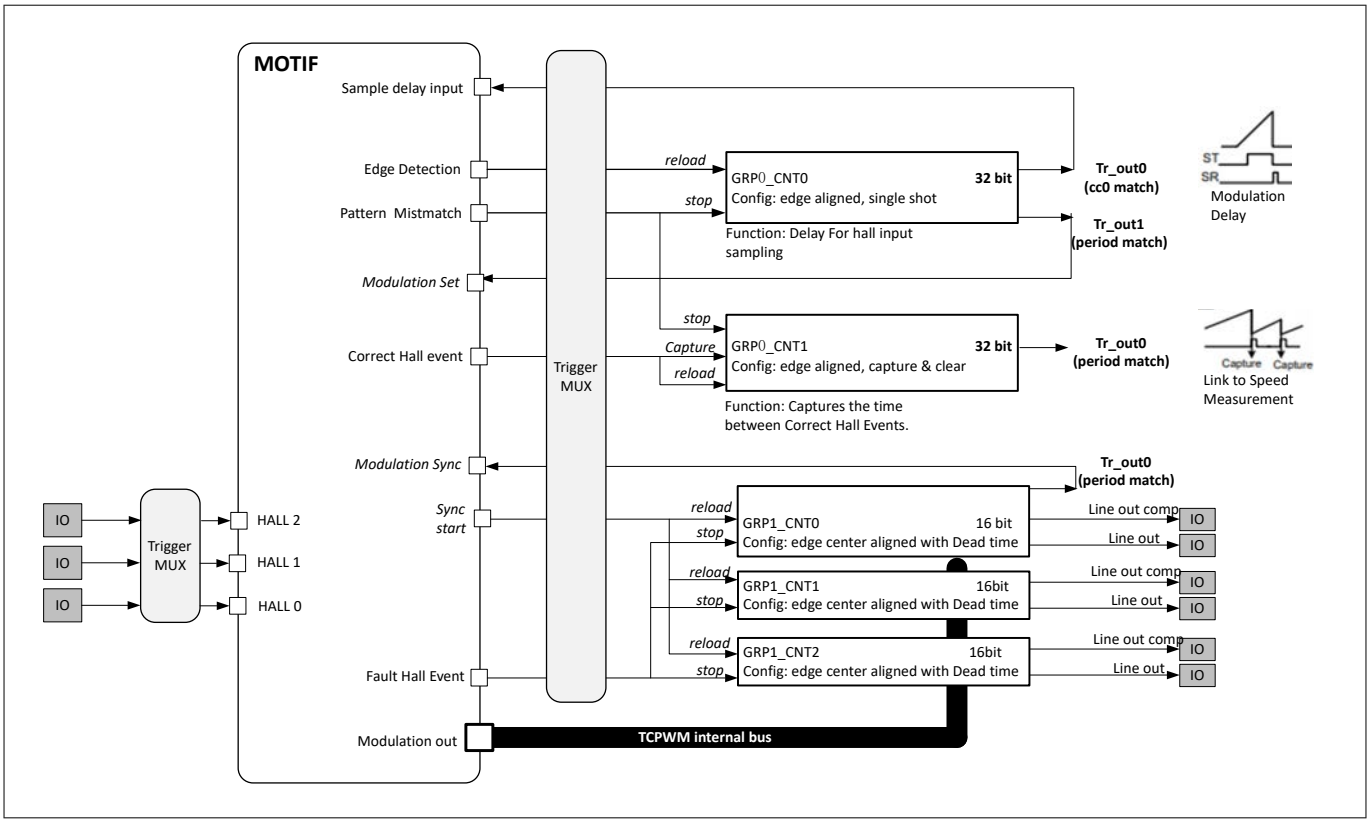


図 171 ホールセンサーモードの使用 - プロファイル 1

直交デコーダモード使用:

直交デコーダー モードは、TCPWM カウンター モジュールと接続することにより、非常に柔軟な方法で使用できます。接続プロファイルは、ユーザーが並行して実行したい機能 (位置制御、回転制御、速度測定) の数によって決まります。

直交デコーダーの使用法 - N ティック間のティックと回転の比較と速度

図 172 では、MOTIF はトリガー マルチプレクサを介して 4 つの TCPWM カウンターとリンクしています。このプロファイルでは、直交デコーダー モードのユーザーは、位置制御 (比較) に使用されるカウンター、回転数カウントに使用されるカウンター、速度測定に使用される集約された 2 つのカウンターを持っています。

カウンター 0 は QCLK と方向イベントに接続されています。これは、QCLK をカウント機能として設定し、カウンター 0 の内部でアップ/ダウン カウント機能として方向を設定することを意味します。このカウンターは、システムの実際の位置を追跡するために使用され、位置が特定の値に達したときに、必要なアクションをトリガーするようにコンペアチャネルを設定できます。

カウンター 1 は INDX イベント (インデックス イベント) に接続されています。このイベントはモーター回転カウンターとして使用されます。コンペアチャネルは、モーターが N 回転するたびに割込みをトリガーするようにプログラムできます。

カウンター 2 とカウンター 3 は速度測定に使用されます。カウンター 2 は PCLK をカウント イベントとして使用します。このスライスのコンペアチャネルは、カウンター 3 のキャプチャイベントのトリガーに使われます。最後のものはモジュール クロックを使用しているため、すべてのキャプチャ イベントで実際のシステム ティックがキャプチャされます。こうすることで、ユーザーは N 個の位相期間間の経過時間を知ることができ、対応する速度プロファイルを計算できます。

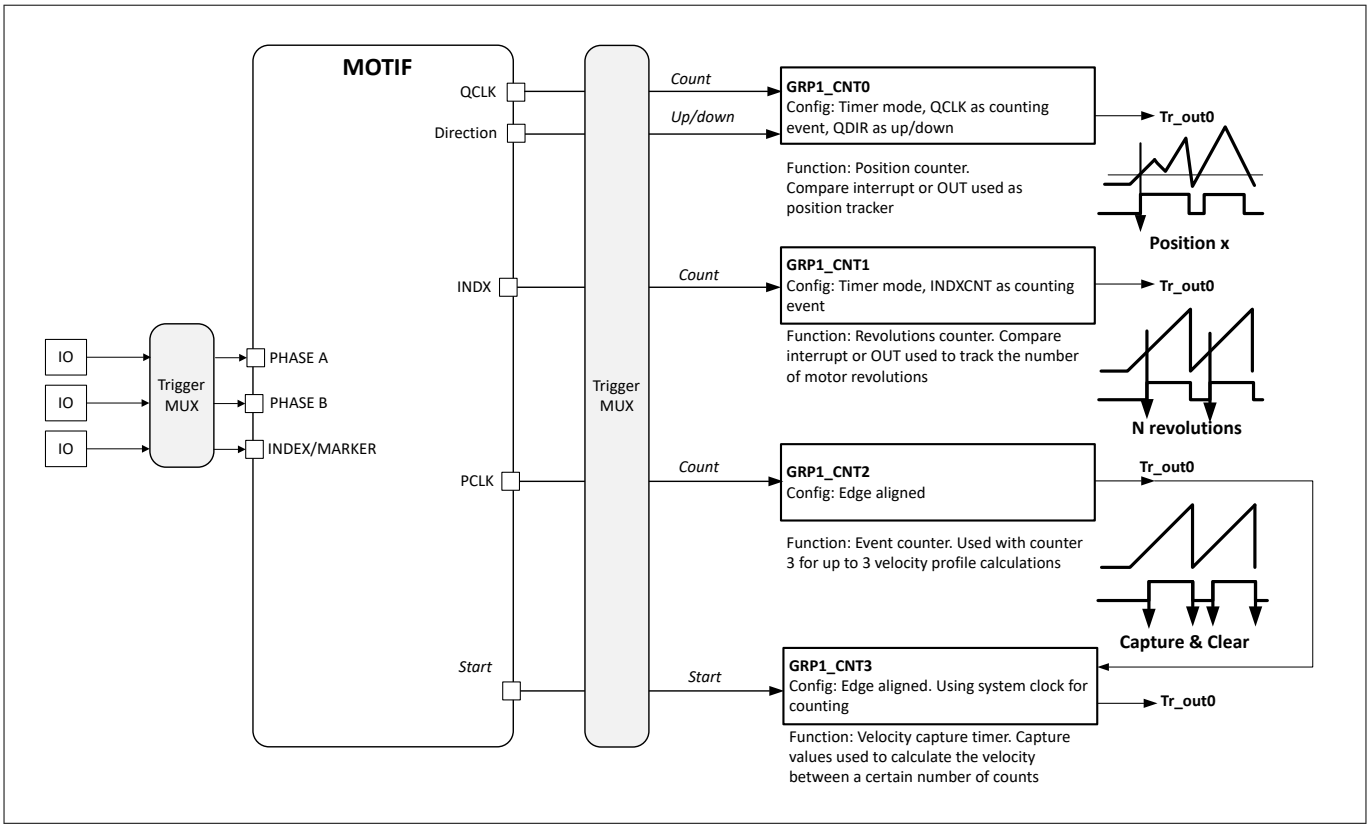


図 172　　直交デコーダー モードの使用 - プロファイル 1

直交デコーダーの使用法 - 拡張ティック比較と N ティック間の速度

下図に示すプロファイルでは、位置制御に 2 つのコンペアチャネルを使用できます。このプロファイルは、モーターが 1 回転する間に複数の作業を行う場合に特に有効です。

カウンター 0 とカウンター 1 は、QCLK イベントをカウント機能として使用し、方向をアップ/ダウン機能として使用しています。各カウンターの比較チャネルは、異なる比較値でプログラムされます。

カウンター 2 とカウンター 3 は、図 172 のプロファイル 1 と同様に使用されます。

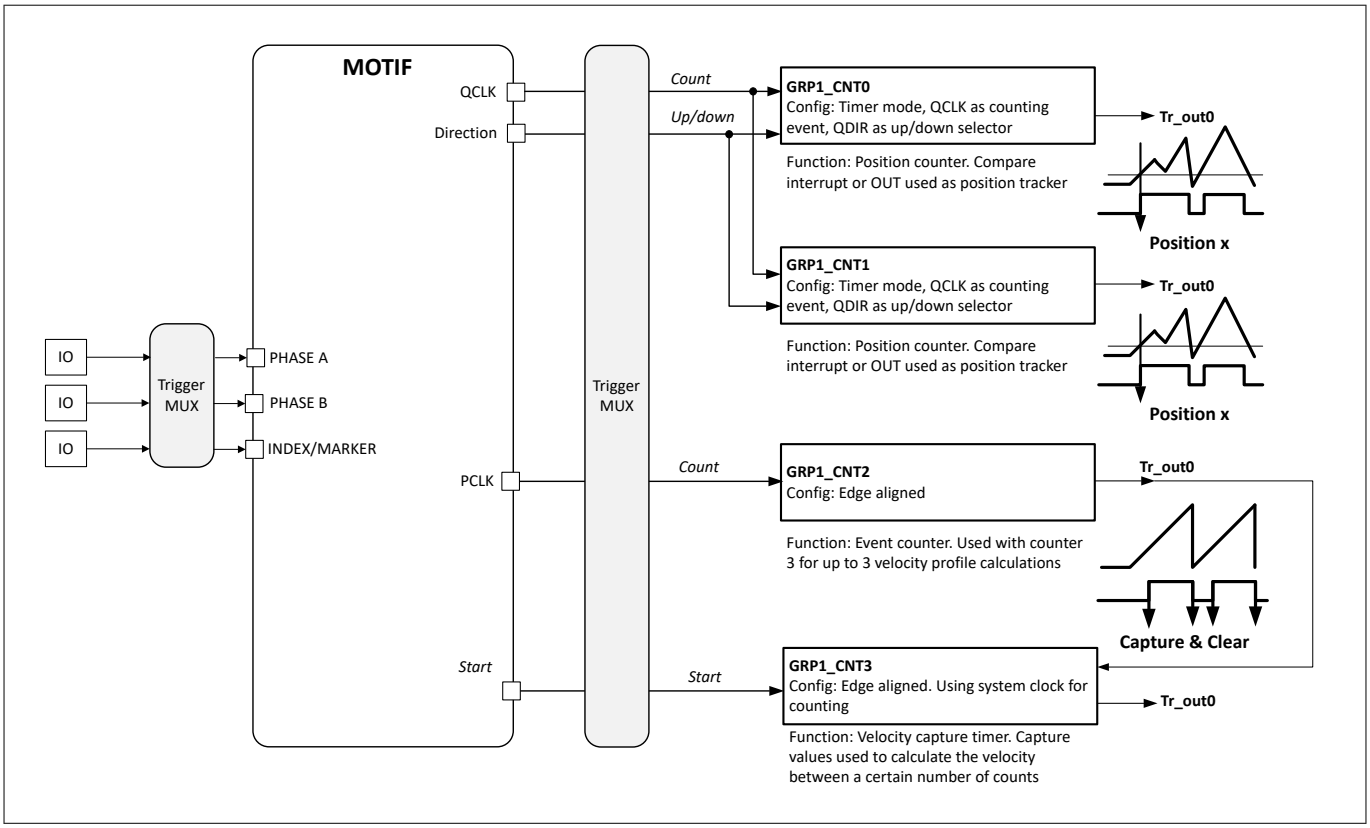


図 173　　直交デコーダー モードの使用 - プロファイル 2

直交デコーダーの使用法 - インデックスクリアと N ティック間の速度によるティックと回転の比較

アプリケーションによっては、インデックス マーカーを位置と速度の制御のための明確な信号として使用すると便利です。PRGINDX イベントは、最初のインデックスマーカーでのみ発生するように設定することも、すべてのマーカーのヒットで発生するように設定することもできます。この PRGINDX イベントは、特定のカウンターに接続され、クリア機能として使用されます。次の図に、PRGINDX イベントをクリア機能として使用したプロファイル 1 の適応を示します。

プロファイル 2 でも同様の手順で、2 つのコンペアチャネルに加え、速度計測と PRGINDX をクリア信号として得られます。

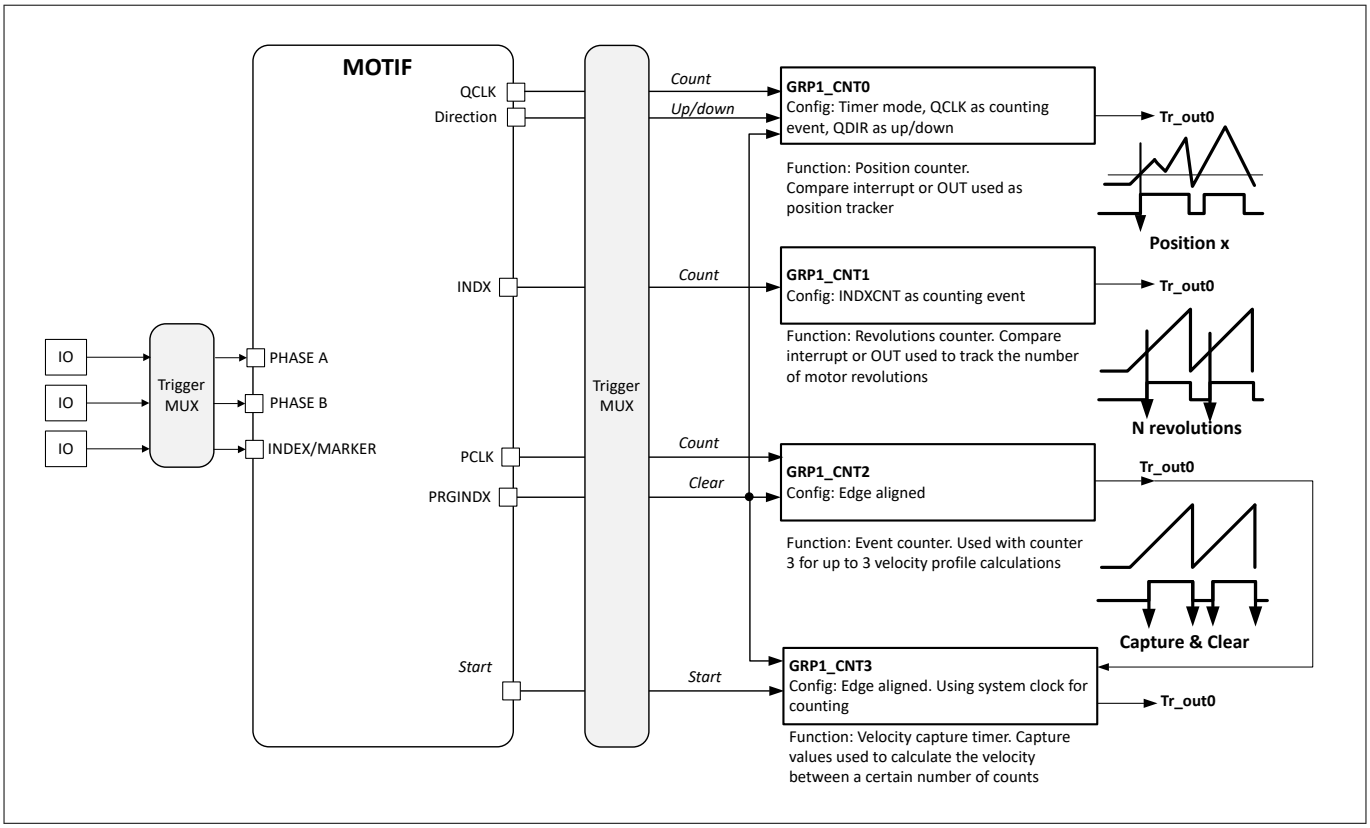


図 174　　直交デコーダー モードの使用 - プロファイル 3

直交デコーダーの使用法 - 低速回転用のティック比較と微小ティック速度

モーターの回転速度が遅い場合、ロータリエンコーダーのティックが発生するまでの時間を捨象することは適さない場合があります。

高速回転システムでは、通常、ティック間の時間は破棄され、最後の ISR から経過したティック数を考慮することで、速度計算を行えます。しかし、回転の遅いシステムでは、ティック数を考慮するだけでは不十分な場合があり(関連する誤差のため)、ソフトウェアは最後のティックから実際の ISR トリガーまでの時間も考慮する必要があります。

以下のタイミング図に、速度計算のための ISR が周期的にトリガーされる低速回転システムを示します。この場合、各 ISR トリガー間のティック数が少ないため、ソフトウェアは経過ティック数のみでなく、最後のティックから ISR 発生までの経過時間も知る必要があります。

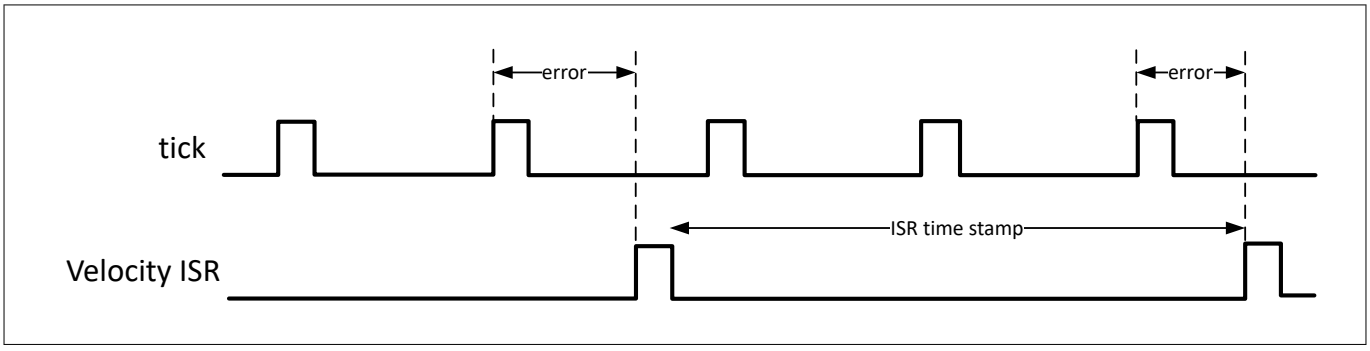


図 175　　低速回転システムの例

MOTIF と 1 つのカウンターモジュールでプロファイルを構築し、この遅い速度計算の落とし穴に陥らない制御ループを実行することは可能です。カウンター 0 は、モーターシャフトの現在位置を監視するために使用されます。低速度計算ループを構築するためには、さらにカウンター 1、カウンター 2、カウンター 3 の 3 つのカウンターが必要です。

カウンター 1 は、各速度 ISR が発生する間に経過するティック数 (PCLK) をカウントします。カウンター 2 は各ティック (PCLK) 発生間の時間をカウントしています。このカウンターはティックごとにクリアされ開始されるため、常に前回のティックと現在のティックの間のタイミング情報を保持します。カウンター 3 は、速度 ISR の周期性を制御しています。ベロシティ ISR がトリガーされるたびに、カウンター 3 はカウンター 2 のキャプチャとカウンター 1 のキャプチャ＆クリアもトリガーします。

このメカニズムにより、ソフトウェアがカウンター 1 とカウンター 2 から取り込まれた値を読み返すたびに、それに基づいて速度を計算できます。

- 最後の ISR からの経過ティック数
- さらに、最後のティックから ISR までの経過時間

したがって、この制御ループは、低速度システム内でモーター速度を計算する非常に正確な方法を提供します。

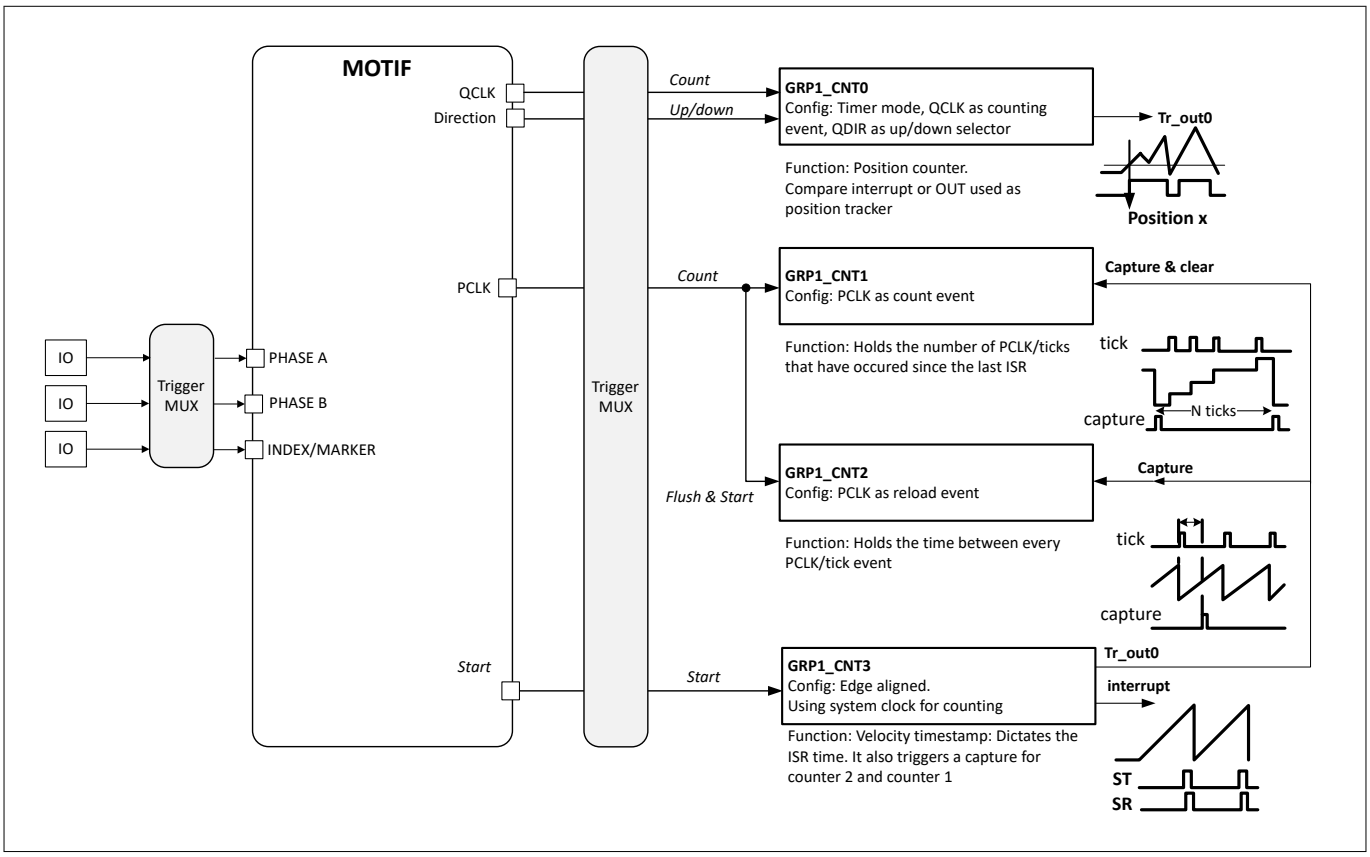


図 176　　直交デコーダー モードの使用 - プロファイル 4

スタンドアロン マルチチャネル モードの使用法

マルチチャネルモードは、ホールセンサーモードと連動することなく MOTIF モジュールで使用できます。これは、完全に同期させる必要がある汎用的な制御ループを実行する場合に特に有効でした。

スタンドアローンのマルチチャネルモードは非常に汎用的で、ユーザーが指定した制御ループに大きく依存します。マルチチャネルモードの一般的なプロファイルは、信号生成を行うために PWM カウンターのセットを使用し、マルチチャネルパターンを更新するためのトリガーとして使用される外部ピンとリンクされた汎用カウンターを使用することです。

図 177 では、汎用カウンターは、外部信号 (例えば外部センサー) の更新からマルチチャネルパターンの更新までの遅延を制御するために使用されます。タイミング遅延が必要ない場合は、この外部信号を MOTIF モジュールに接続することもできるし、MCMS.MNPS フィールドに 1 を書き込んでソフトウェアで制御することもできます。

そして、PWM カウンター tr_out 出力の 1 つを、生成された信号とマルチチャネルパターンの更新の間の同期トリガーとして動作するように選択できます。

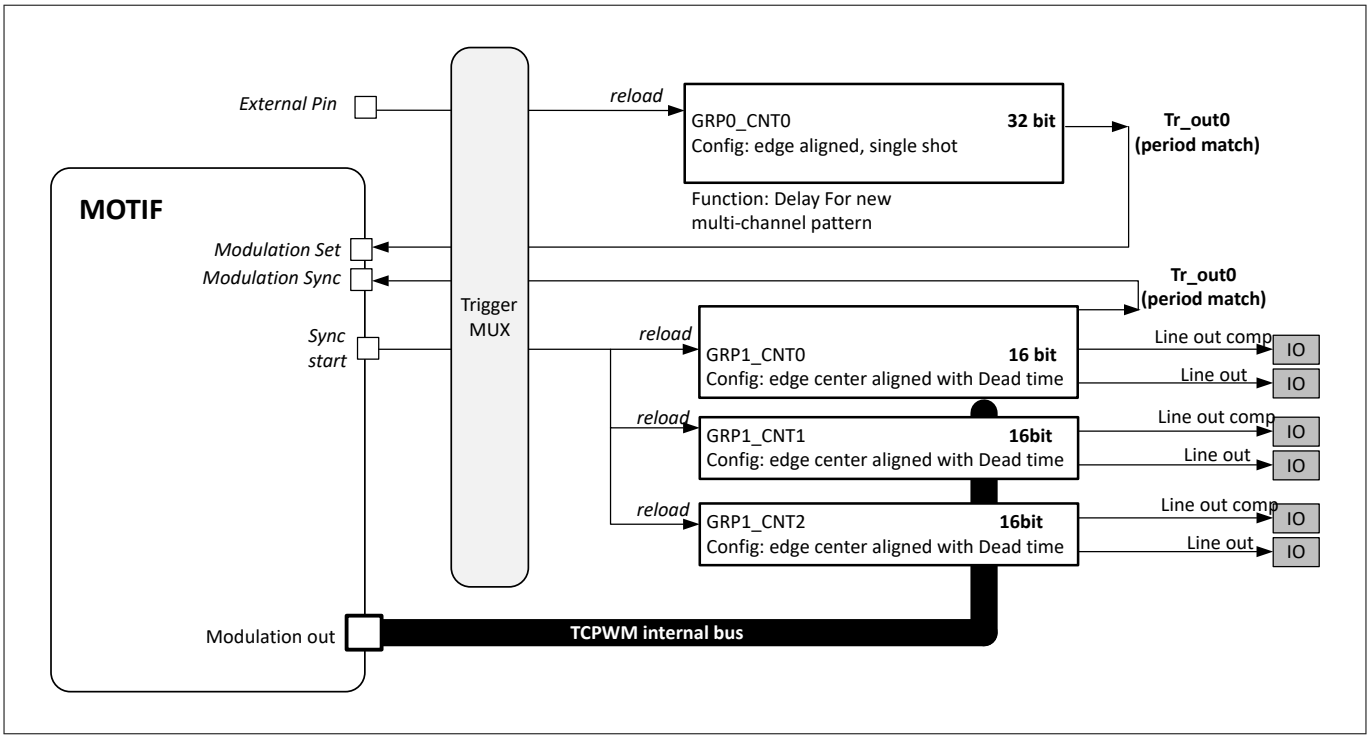


図 177　　スタンドアロン マルチチャネル モードの使用法

MOTIF 変調パターンで制御された PWM 出力

図 178 に、MOTIF がサポートするさまざまなタイプの PWM スイッチング方式を TCPWM カウンターとともに示します。

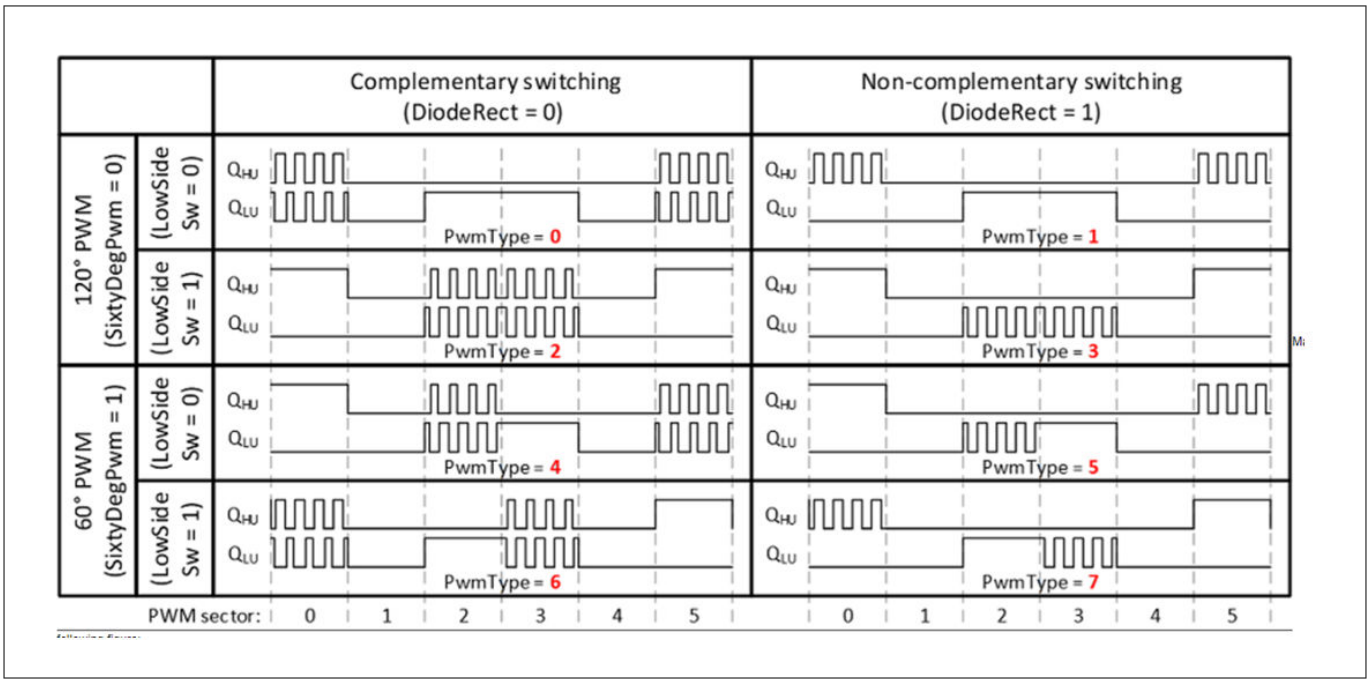


図 178　　MOTIF による PWM スイッチング変調

24.6 高分解能 PWM (HRPWM)

24.6.1 概要

高分解能 PWM (HRPWM) 生成機能は、PWM および PWM_DT モードでのみサポートされます。HRPWM の強化により、電力変換アプリケーションの高分解能制御が可能になります。高解像度生成サポートはグループ 0 にのみ適用されます。

24.6.1.1 機能

- PWM および PWM_DT モードのみ HRPWM に対応
- 32 ビットカウンターで HRPWM 機能に対応
- HRPWM 機能は GROUP #0 でのみ使用可能です。
- 周期、デューティサイクル、デッドタイムを整数および小数機能でサポート
  - 既存の PERIOD、PERIOD_BUFF、CC0、CC0_BUFF、CC1、CC1_BUFF、DT、DT_BUFF レジスタは再利用されます。
  - PERIOD、CC0、CC1、DT レジスタは、整数および小数機能の目的で使用されます。
  - PERIOD_BUFF、CC0_BUFF、CC1_BUFF、DT_BUFF レジスタはシャドウ更新用に使用されます。
- 遅延調整用 6 小数ビットに対応
- 以下の PWM 出力モードをサポート: HRPWM 機能の「右揃え、左揃え、中央揃え、cc0 との非対称、cc0 と cc1 との非対称、cc0 と cc1 との中央揃え非対称」

***注**: HRPWM 機能を持つ各 TCPWM カウンターについて、カウンターを無効にする正しい順序は以下のとおりです。*

- *1) HRPWM_EN ='0'に設定して HRPWM 機能を無効にします。*
- *2) ENABLED ='0'に設定してカウンターを無効にします。*

***注**: HRPWM 機能を有効にする場合は、**PERIOD - CC0 ≧3** を確認してください。そうしないと、PERIOD - CC0 ≦2 の場合、中央揃え PWM モードで HRPWM を有効にし、UPDWN を有効にすると、CC0_match イベントが発生せず、PWM 波形が異常出力されます。*

24.6.1.2 ブロック図

次のブロック図に、カウンターグループに対する HRPWM の接続例を示します。

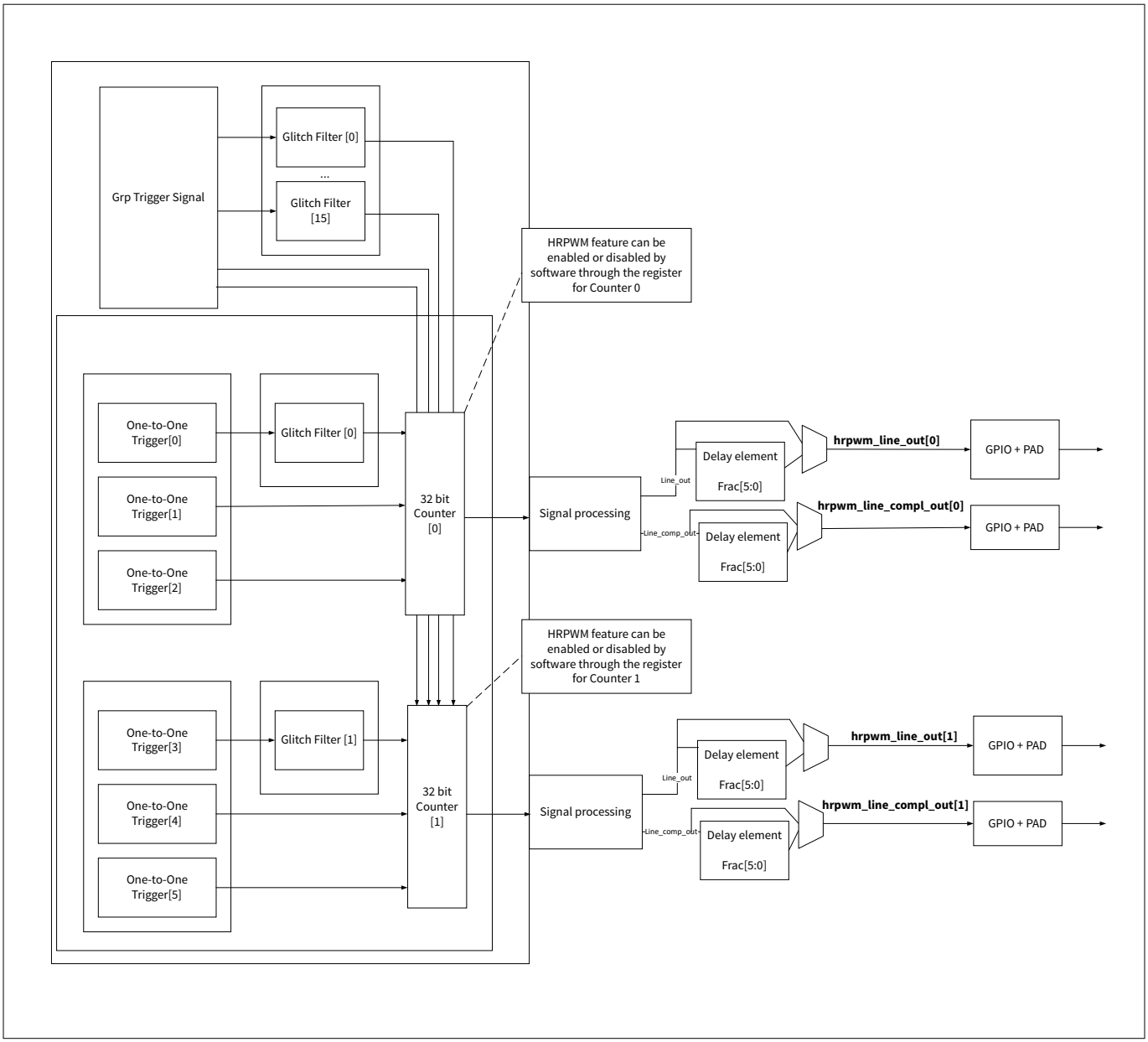


図 179　　HRPWM ブロック図

24.6.2　　機能説明

PWM モードでは、HRPWM 機能が有効な場合、LSB は PERIOD/PERIOD_BUFF、CC0/CC0_BUFF、CC1/CC1_BUFF および DT/DT_BUFF の line_out および line_compl_out 出力を遅延させるための小数ティックまたはマイクロティックに使用されます。

図 180 に、レジスタの整数と小数のパーティションを示します。

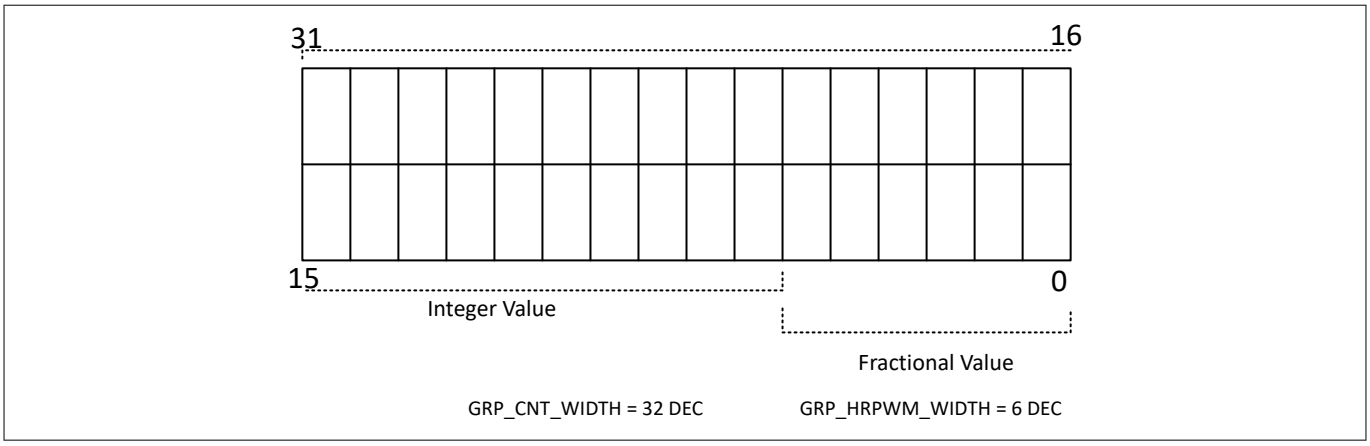


図 180　　レジスタのレイアウト例

***注**:　　整数部の最大値は 67108863 です。小数部分の最大値は 63 です。*

図 181 に、PERIOD レジスタに整数値と小数値をプログラムする方法を示します。ソフトウェアが PERIOD レジスタに 2374 DEC を書き込んで、整数値に 37 DEC、小数値に 6 DEC を設定します。

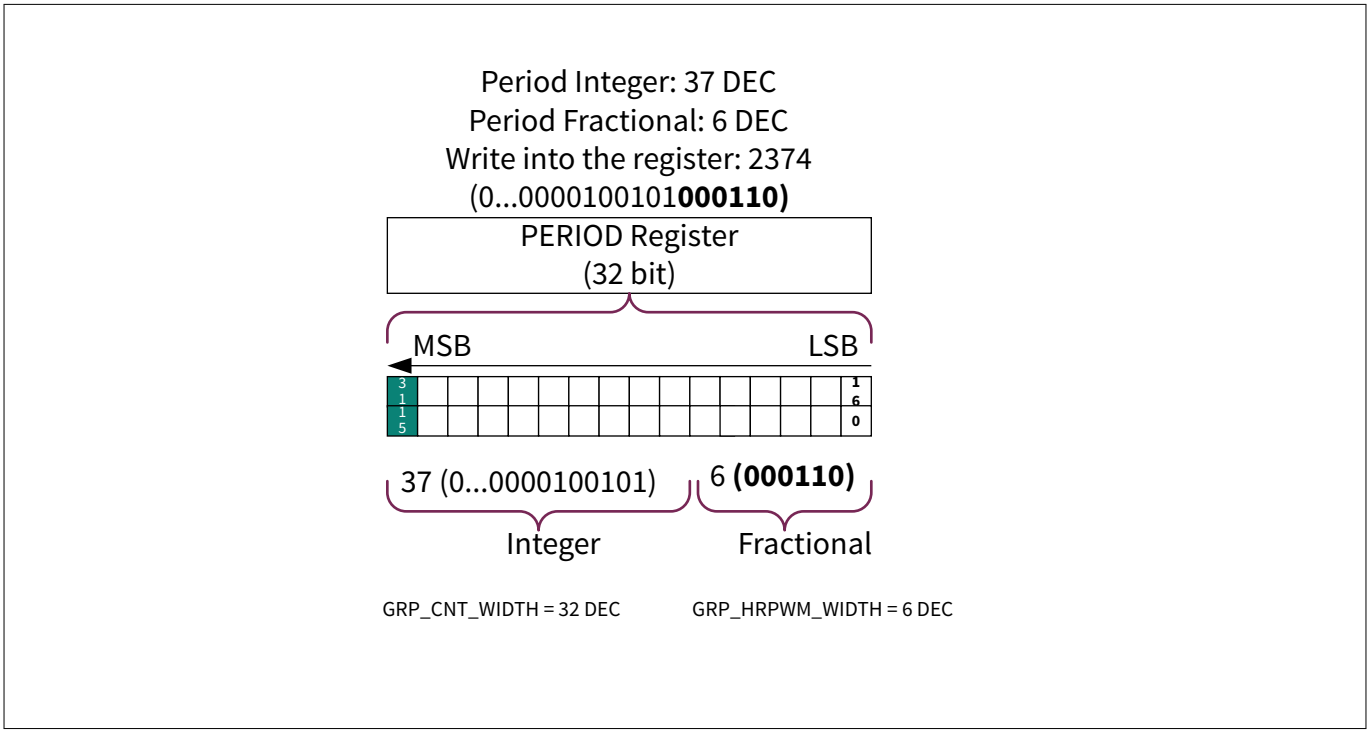


図 181　　整数値と小数値をレジスタにプログラムする例

レジスタにプログラムされた小数値は、データ出力の生成に使用されます。

- data_shift_line_out[5:0]: 遅延エレメントによって使用され、line_out の立ち上りエッジと立ち下りエッジを等価なマイクロティック単位でシフトします。TCPWM は、line_out の立ち上りエッジと立ち下りエッジの tc と cc0/cc1 の一致イベントに同期して値を生成し、対応するエッジを遅延させなければいけないマイクロティック数を制御します。
- data_shift_line_compl_out[5:0]: 遅延エレメントによって使用され、line_compl_out の立ち上りエッジと立ち下りエッジを等価なマイクロティック単位でシフトします。TCPWM は、line_compl_out の立ち上りエッジと立ち下りエッジの tc と cc0/cc1 の一致イベントに同期して値を生成し、対応するエッジを遅延させなければいけないマイクロティック数を制御します。

注: HRPWM 機能を備えた各 TCPWM カウンターには、2 つの遅延素子が存在します。1 つは line_out の立ち上りエッジと立ち下りエッジを遅延させるためのもの、もう1 つは line_compl_out の立ち上りエッジと立ち下りエッジを遅延させるためのものです。

計算コンセプト

図 182 と図 183 に、PWM モードで data_shift_line_out と data_shift_line_compl_out が計算され、line_out と line_compl_out とともに遅延素子に送られる例を示します。

PERIOD レジスタに 0 を超える小数値がある場合、PF_ACC に加算されます。PF_ACC 値がオーバーフローし、キャリーが PERIOD レジスタの整数値 (PI) に加算されると、キャリーは次の TC イベントでクリアされます。

PF_ACC もまた、CC0/CC1 レジスタに設定された小数値を加算したものです。加算値 CC0_ACC または CC1_ACC が同様にオーバーフローすると、キャリーが対応する CC0/CC1 レジスタの整数値に 1TC イベントの間隔のみ加算され、その後キャリーがクリアされます。

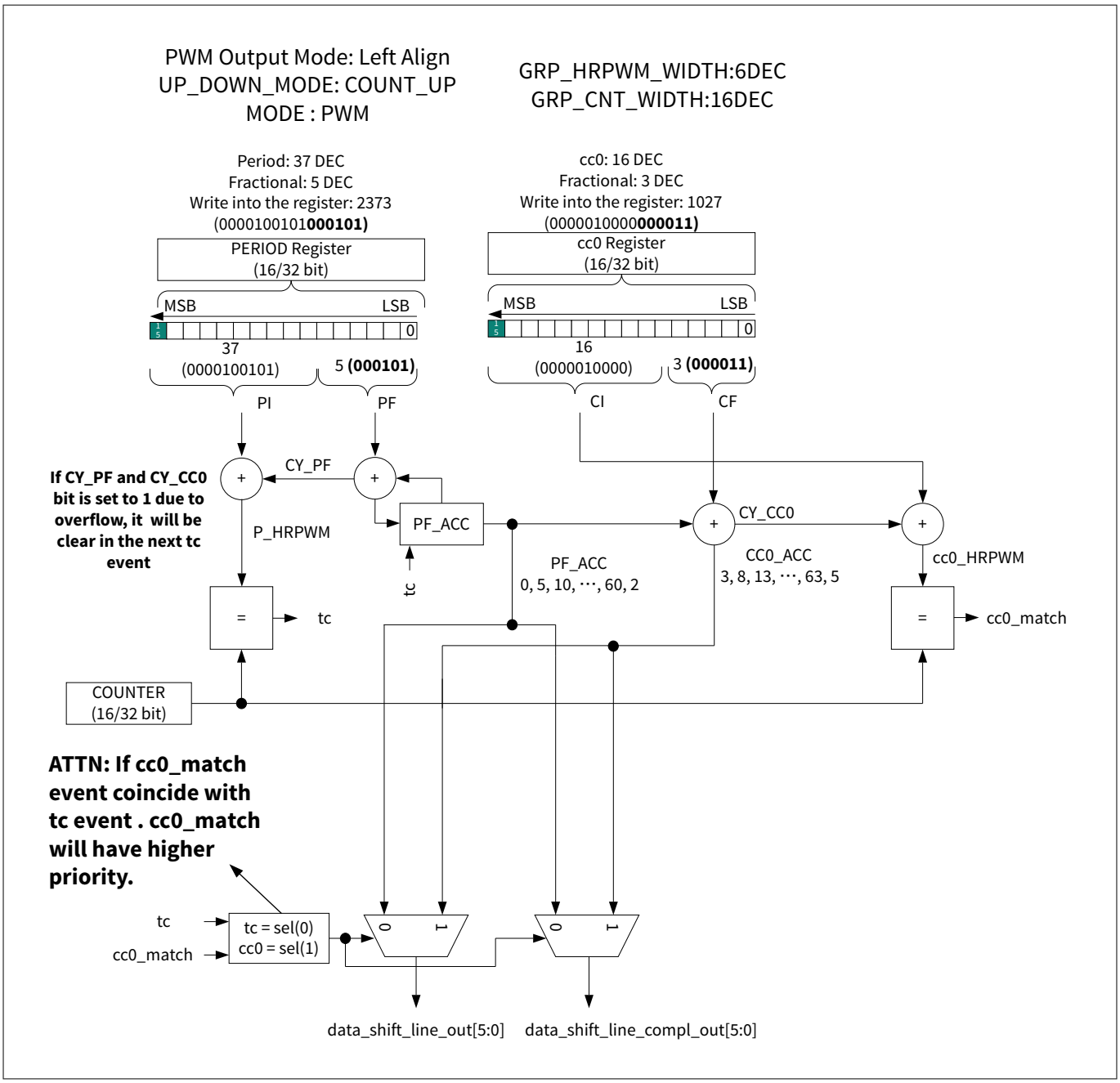


図 182 PWM モードでの data_shift_line_out と data_shift_line_compl_out の計算

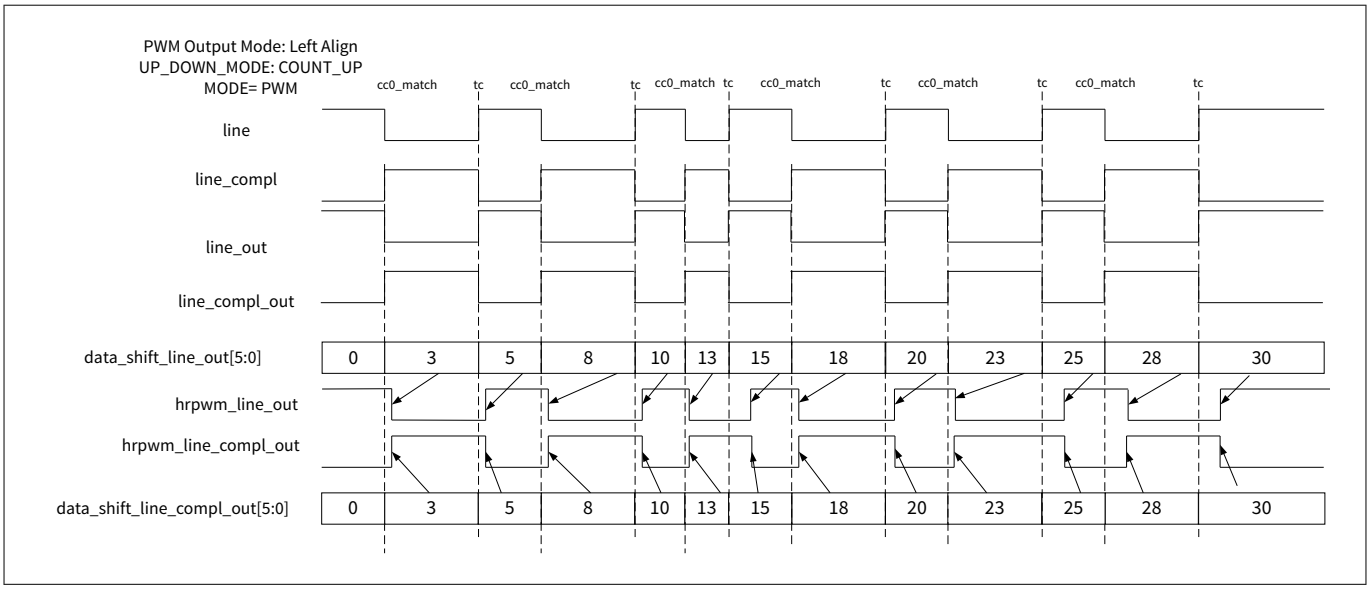


図 183　　PWM モードでの TCPWM 出力例

PWM 出力モード

HRPWM では以下の出力モードがサポートされています。

表 183　　HRPWM の PWM 出力モード

PWM 出力モード	UP_DOWN_MODE	TR_PWM_CRL	その他の設定	"line"パルス幅	周期
右揃え	COUNT_DOWN	UNDERFLOW = CLEAR CC0_MATCH = SET	対象外	CC0+1	PERIOD ＋ 1
左揃え	COUNT_UP	OVERFLOW = SET CC0_MATCH = CLEAR	対象外	CC0	PERIOD ＋ 1
中央揃え	COUNT_UPDN1	OVERFLOW = SET UNDERFLOW = CLEAR CC0_MATCH = INVERT	対象外	2xPERIOD – 2xCC0-1 (CF=非ゼロの場合) 2xPERIOD – 2xCCO (CF=ゼロの場合)	PERIOD * 2
cc0 と非対称に中央揃え	COUNT_UPDN2	OVERFLOW = SET UNDERFLOW = CLEAR CC0_MATCH = INVERT	対象外	2xPERIOD - CC0 - CC0_BUFF-1 (CF=非ゼロの場合) 2xPERIOD - CC0 - CC0_BUFF (CF=ゼロの場合)	PERIOD * 2

(続く)

表 183 (続き) HRPWM の PWM 出力モード

PWM 出力モード	UP_DOWN_MODE	TR_PWM_CRL	その他の設定	"line"パルス幅	周期
cc0 と cc1 で非対称	COUNT_DOWN COUNT_UP	CC0_MATCH = SET CC1_MATCH = CLEAR OVERFLOW = CLEAR (UP) UNDERFLOW = CLEAR (DOWN)	対象外	CC1-CC0	PERIOD ＋ 1
cc0 と cc1 で非対称に中央揃え	COUNT_UPDN1 COUNT_UPDN2	OVERFLOW = SET UNDERFLOW = CLEAR CC0_MATCH = SET CC1_MATCH = CLEAR	CC0_MATCH_UP_EN = 1, CC0_MATCH_DOWN_EN = 0, CC1_MATCH_UP_EN = 0, CC1_MATCH_DOWN_EN = 1,	2xPERIOD - CC0 - CC1 - 1 (CF=非ゼロの場合) 2xPERIOD - CC0 - CC1 (CF=ゼロの場合) (UPDN2 ではオーバーフロー時のスワップなし)	PERIOD * 2

PWM 出力モード - 中央揃え

次の図に、中央揃えの場合の HRPWM の期待値を示します。これは、遅延素子に転送される全体的な小数シフト値に加えて、デッドタイムとその小数成分の取り扱いを示します。

注: 小数周期が 0 以外に設定されている場合、各周期サイクルのデューティ サイクルは等しくなりません。

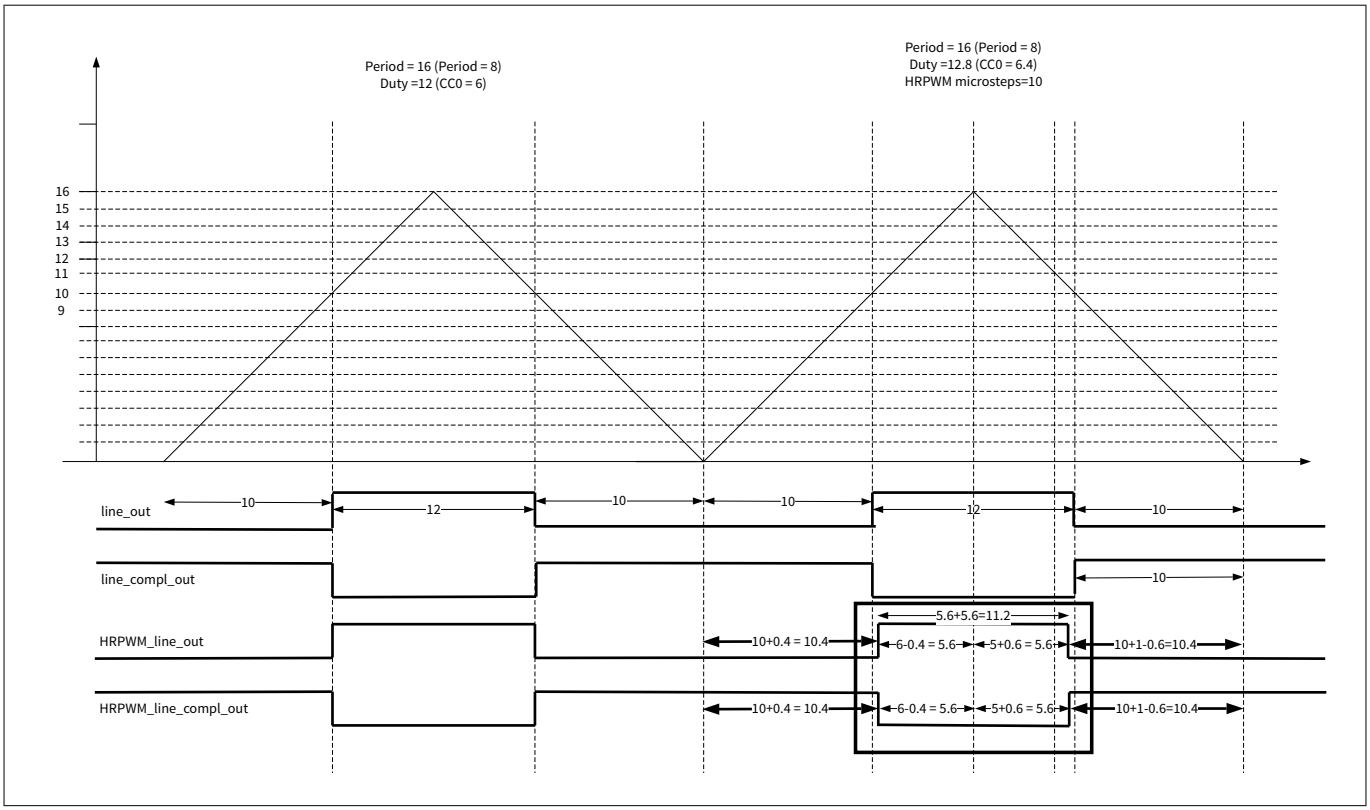


図 184　　中央揃えケース HRPWM 期待値

PWM_DT

PWM_DT モードにおいて、HRPWM 機能を有効にすると、以下のことがサポートされます。

- LSB は、PERIOD/PERIOD_BUFF、CC0/CC0_BUFF、CC1/CC1_BUFF および DT/DT_BUFF の line_out および line_compl_out 出力を遅延させるための小数またはマイクロティックに使用されます。小数制御に使用される LSB ビットの数は 6 です。GRP_CNT_WIDTH-1:GRP_HRPWM_WIDTH]で与えられる残りのビットは、TCPWM カウンターによって使用されます。
- LSB は以下のレジスタ DT/DT_BUFF のデッドタイム挿入で line_out と line_compl_out 出力を遅延させるための小数またはマイクロティックに使用されます。小数制御に使用される LSB ビットの数は 6 です。残りの 26 ビットは TCPWM カウンターで使用されます。

PWM_DT モードでの計算コンセプト

次の図に、PWM_DT モードにおいて data_shift_line_out と data_shift_line_compl_out が計算され、line_out と line_compl_out とともに遅延素子に送出される例を示します。

DT レジスタが line_out の小数値>0 を含む場合、それは PF_ACC に加算されます。同様に、line_compl_out の DT 分数値が CC0_ACC に加算されます。最終値がオーバーフローした場合、キャリーは対応する DT 整数値に 1TC イベント期間加算されます。

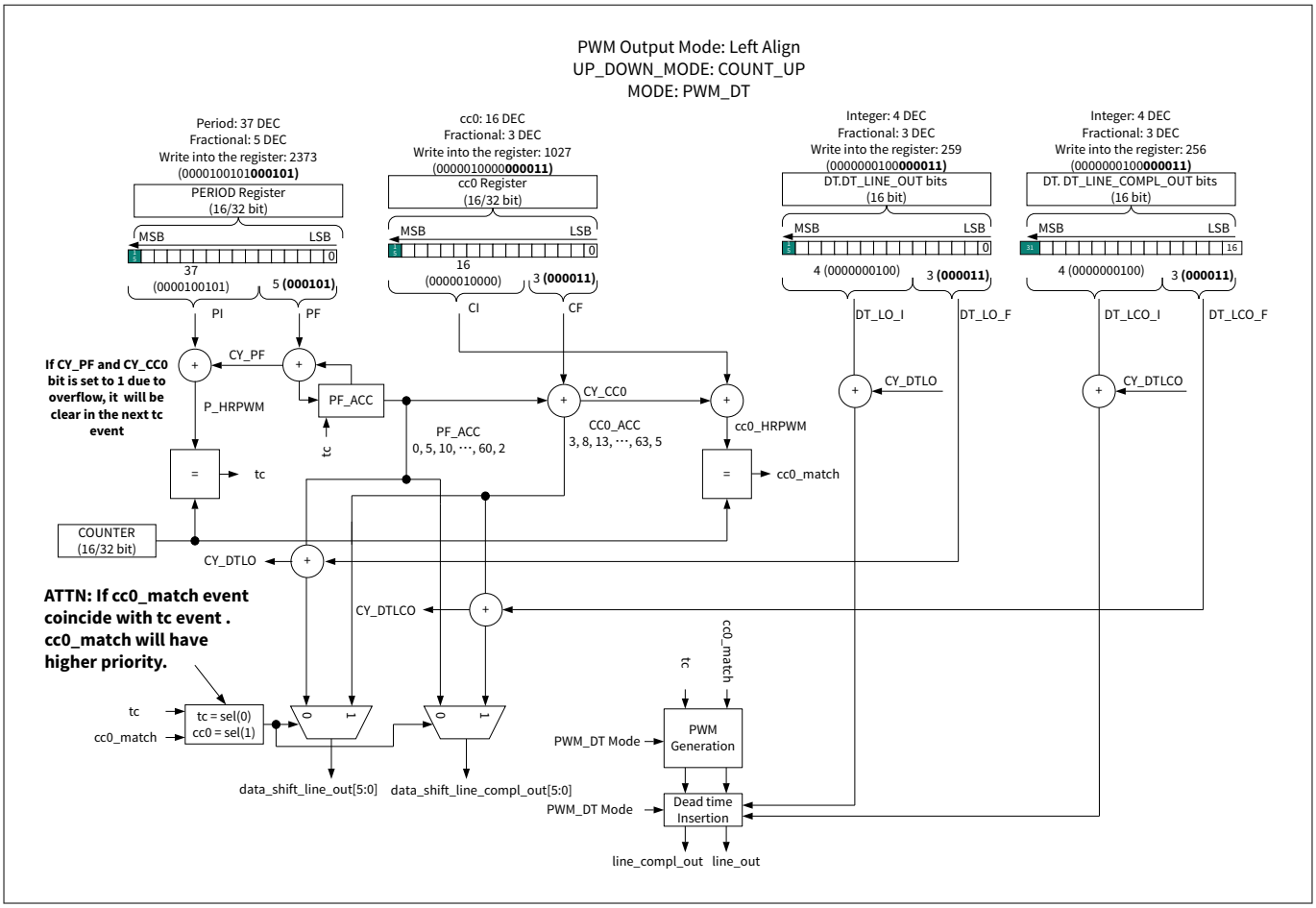


図 185　　PWM_DT モードにおける data_shift_line_out および data_shift_line_compl_out の計算

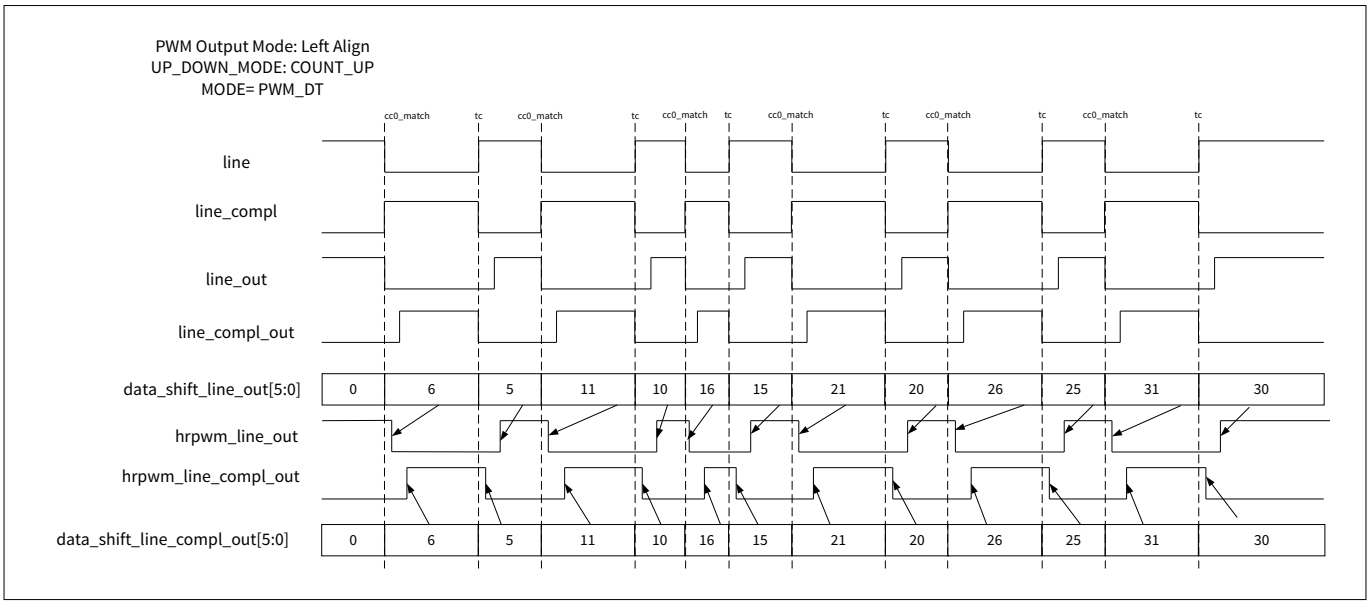


図 186　　PWM_DT モードでの TCPWM 出力例

PWM_DT - 中央揃え

図 185 と図 186 に、中央揃えの場合の HRPWM の期待値を示します。これは、遅延素子に転送される全体的な小数シフト値に加えて、デッドタイムとその小数成分の取り扱いを示します。

注: 小数周期が 0 以外に設定されている場合、各周期サイクルのデューティ サイクルは等しくなりません。

- 1 番 (青い点) のエッジは、オレンジの文字で示されるように、DEADTIME_LINE_OUT の小数部分を全体の小数に加えます
- 2 番 (青い点) のエッジは、DEADTIME_LINE_OUT の小数部分を全体の小数部分に加えてはいけません。
- 3 番 (青い点) のエッジは、DEADTIME_LINE_COMPL_OUT の小数部分をオレンジの文字で示されるように全体の小数部分に追加します。
- 4 番 (青い点) のエッジは、DEADTIME_LINE_COMPL_OUT の小数部分を全体の小数部分に加えてはいけません。

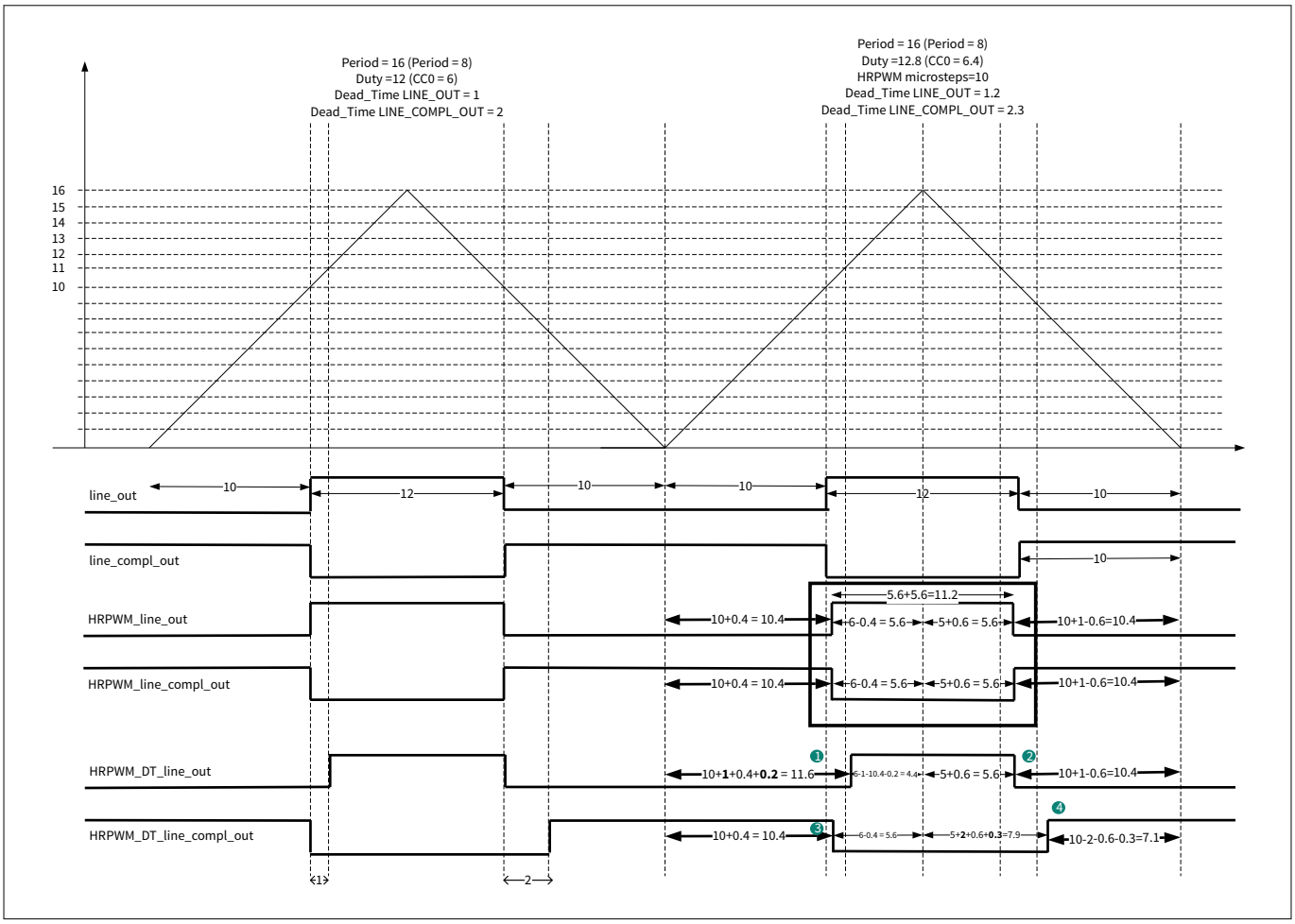


図 187 PWM_DT モード用中央揃えケース HRPWM 期待値

24.7 リカバリー

TCPWM は、以下のようなアクティブ リセット イベントで回復できます。

- パワーオン リセット (POR)
- 外部リセット (XRES)
- ウォッチドッグ タイマー リセット (MCWDT および WDT)
- ブラウンアウト検出リセット
- 過電圧 過電流検出リセット

TCPWM が想定外の状態になることはありません。

24.8 初期化

TCPWM ピンの初期状態は Hi-Z です。一部のレジスタはアクティブ リセットでリセットされ、一部の MMIO レジスタはディープスリープで保持されます。ハイバネートやその他の低消費電力モードでは、どのレジスタも保持されません。アクティブリセットは、CTRL.PWM_DISABLE_MODE が 0 以外の値に設定されない限り、ピンの状態を Hi-Z に戻します。このシナリオでは、コンフィギュレーションによって結果が異なります。

24.9 ピンの状態

TCPWM が未使用の場合、TCPWM ピンのステータスは Hi-Z となります。TCPWM を無効にするためには、CTRL レジスタの ENABLED ビットと PWM_DISABLE_MODE ビットが'0'に設定され、MOTIF_PCONF.ENABLE も'0'に設定されていることを確認します。

24.10 レジスタの概要

表 184 TCPWM レジスタの一覧

名称	説明
TCPWM0_GRPx_CNT0_CTRL	カウンター制御レジスタ
TCPWM0_GRPx_CNT0_STATUS	カウンター ステータス レジスタ
TCPWM0_GRPx_CNT0_COUNTER	カウンター カウント レジスタ
TCPWM0_GRPx_CNT0_CC0	カウンター比較/Capture0 レジスタ
TCPWM0_GRPx_CNT0_CC0_BUFF	カウンター バッファ付き比較/Capture0 レジスタ
TCPWM0_GRPx_CNT0_CC1	カウンター比較/Capture1 レジスタ
TCPWM0_GRPx_CNT0_CC1_BUFF	カウンター バッファ付き比較/Capture1 レジスタ
TCPWM0_GRPx_CNT0_PERIOD	カウンター期間レジスタ
TCPWM0_GRPx_CNT0_PERIOD_BUFF	カウンター バッファ期間レジスタ
TCPWM0_GRPx_CNT0_LINE_SEL	カウンター行選択レジスタ
TCPWM0_GRPx_CNT0_LINE_SEL_BUFF	カウンター バッファ ライン選択レジスタ
TCPWM0_GRPx_CNT0_DT	カウンター PWM デッドタイム レジスタ
TCPWM0_GRPx_CNT0_DT_BUFF	カウンター バッファ PWM デッドタイム レジスタ
TCPWM0_GRPx_CNT0_PS	カウンター プリスカラ レジスタ
TCPWM0_GRPx_CNT0_TR_CMD	カウンタートリガー コマンド レジスタ
TCPWM0_GRPx_CNT0_TR_IN_SEL0	カウンター入力トリガー選択レジスタ 0
TCPWM0_GRPx_CNT0_TR_IN_SEL1	カウンター入力トリガー選択レジスタ 1
TCPWM0_GRPx_CNT0_TR_IN_EDGE_SEL	カウンター入力トリガー エッジ選択レジスタ
TCPWM0_GRPx_CNT0_TR_PWM_CTRL	カウンタートリガー PWM 制御レジスタ
TCPWM0_GRPx_CNT0_TR_OUT_SEL	カウンター出力トリガー選択レジスタ
TCPWM0_GRPx_CNT0_INTR	割込み要求レジスタ
TCPWM0_GRPx_CNT0_INTR_SET	割込みセット要求レジスタ

(続く)

表 184 (続き) TCPWM レジスタの一覧

名称	説明
TCPWM0_GRPx_CNT0_INTR_MASK	割込みマスクレジスタ
TCPWM0_GRPx_CNT0_INTR_MASKED	割込みマスク要求レジスタ
TCPWM0_GRPx_CNT0_ONE_GF0	1 対 1 トリガー用グリッチ フィルター レジスタ
TCPWM0_GRPx_CNT0_TR_ONE_SYNC_BYPASS	1 対 1 トリガー用同期バイパス レジスタ
TCPWM0_GRP0_CNT0_HRPWM_CTRL	HRPWM 機能用カウンター コントロール レジスタ
TCPWM0_TR_ALL_GF_ALL_GF0	グループ トリガー用グリッチ フィルター モジュール
TCPWM0_TR_ALL_SYNC_BYPASS_TR_ALL_SYNC_BYPASS0	グループトリガー用トリガー同期バイパス
TCPWM0_MOTIF_GRP1_MOTIF0_PCONF	グローバル制御レジスタ
TCPWM0_MOTIF_GRP1_MOTIF0_PSUS	設定の一時停止
TCPWM0_MOTIF_GRP1_MOTIF0_PRUNS	MOTIF 実行ビットセット
TCPWM0_MOTIF_GRP1_MOTIF0_PRUN	MOTIF ランビットステータス
TCPWM0_MOTIF_GRP1_MOTIF0_HIST	ホール入力サンプルトリガー
TCPWM0_MOTIF_GRP1_MOTIF0_HMEC	ホールモード追加設定
TCPWM0_MOTIF_GRP1_MOTIF0_HALP	ホールの現状と予想されるパターン
TCPWM0_MOTIF_GRP1_MOTIF0_HALPS	ホール電流と予想されるシャドーパターン
TCPWM0_MOTIF_GRP1_MOTIF0_HOSC	ホールセンサー出力設定
TCPWM0_MOTIF_GRP1_MOTIF0_MCM	マルチチャネルモードパターン
TCPWM0_MOTIF_GRP1_MOTIF0_MCSM	マルチチャネルモードシャドウパターン LUT0
TCPWM0_MOTIF_GRP1_MOTIF0_MCMS	マルチチャネル モード制御セット
TCPWM0_MOTIF_GRP1_MOTIF0_MCMC	多チャネル モード コントロール クリア
TCPWM0_MOTIF_GRP1_MOTIF0_MCMF	マルチチャネルモードフラグの状態
TCPWM0_MOTIF_GRP1_MOTIF0_MCPF	マルチチャネル パターン フォールト
TCPWM0_MOTIF_GRP1_MOTIF0_MOSC	マルチチャネル出力設定
TCPWM0_MOTIF_GRP1_MOTIF0_QDC	直交デコーダーの設定
TCPWM0_MOTIF_GRP1_MOTIF0_QOSC	直交出力設定
TCPWM0_MOTIF_GRP1_MOTIF0_MCMEC	マルチチャネル追加設定
TCPWM0_MOTIF_GRP1_MOTIF0_PFLG	MOTIF 割込みステータス
TCPWM0_MOTIF_GRP1_MOTIF0_PFLGE	MOTIF 割込み有効化
TCPWM0_MOTIF_GRP1_MOTIF0_SPFLG	割込みセットレジスタ
TCPWM0_MOTIF_GRP1_MOTIF0_RPFLG	割込みクリアレジスタ
TCPWM0_MOTIF_GRP1_MOTIF0_MCSM1	マルチチャネルモードシャドウパターン LUT1
TCPWM0_MOTIF_GRP1_MOTIF0_MCSM2	マルチチャネルモードシャドウパターン LUT2

(続く)

表 184 (続き) TCPWM レジスタの一覧

名称	説明
TCPWM0_MOTIF_GRP1_MOTIF0_MCSM3	マルチチャネルモードシャドウパターン LUT3
TCPWM0_MOTIF_GRP1_MOTIF0_MCSM4	マルチチャネルモードシャドウパターン LUT4
TCPWM0_MOTIF_GRP1_MOTIF0_MCSM5	マルチチャネルモードシャドウパターン LUT5
TCPWM0_MOTIF_GRP1_MOTIF0_CLUT	ホールモード LUT コンフィグ
TCPWM0_MOTIF_GRP1_MOTIF0_SLUT	ホールモード LUT の状態
TCPWM0_MOTIF_GRP1_MOTIF0_PDBG	MOTIF デバッグレジスタ
TCPWM0_MOTIF_GRP1_MOTIF0_PLP0S	MOTIF ローパス 0 ステータス
TCPWM0_MOTIF_GRP1_MOTIF0_PLP1S	MOTIF ローパス 1 ステータス
TCPWM0_MOTIF_GRP1_MOTIF0_PLP2S	MOTIF ローパス 2 ステータス

注: TCPWM0_GRPx の'x'はグループ番号を示します。

各レジスタに同じ値を上書きすると、それぞれ異なる効果があることに注意してください。
ソフトウェアのアクセス属性。TCPWM レジスタには以下のアクセス制限があります。

- すべてのステータス レジスタはソフトウェア書き込み不可
- TR_CMD はソフトウェアで設定され、ハードウェアでクリアされます。
- INTR はソフトウェアでクリアされ、ハードウェアで設定されます (INTR_SET に'1'を書き込んでください)。
- INTR_SET を読み出すと、INTR の値が返されます。
- 他のレジスタは正常で、同じ値で上書きできます。

25 コントローラー エリア ネットワーク フレキシブル データ レート (CAN FD)

25.1 概要

CAN FD コントローラーは、ISO 11898-1 (CAN 仕様 2.0 パート A および B) に準拠しています。さらに、ISO 11898-4 で定義された TTCAN (Time-Triggered CAN) プロトコルに対応しています。

すべてのメッセージ処理機能は、RX ハンドラと TX ハンドラによって実装されます。RX ハンドラは、メッセージの受入フィルタリング、CAN コアからメッセージ RAM への受信メッセージの転送、メッセージのステータス情報の受信を管理します。TX ハンドラは、送信メッセージをメッセージ RAM から CAN コアに転送し、送信ステータス情報を提供します。

CAN 動作用の CAN クロック (CANFD[0].CLOCK_CAN_EN[0:1]) と内部ブロック動作用のシステムクロック (SYSCPUSS/clk_hf0) の独立した 2 つのクロックが供給されます。受入フィルタリングは、最大 192 個のフィルター エレメントの組み合わせによって実装され、それぞれがレンジ、ビット マスク、または専用 ID フィルターとして設定できます。

CAN FD コントローラーはアクティブとスリープ モードのみで使用できます。ディープスリープ モードでは機能しませんが、コンフィギュレーションとメッセージ RAM の内容は完全に保持されます。

25.1.1 特長

CAN FD コントローラーには以下の特長があります。

- フレキシブル データ レート (FD) (ISO 11898-1: 2015)
  - メッセージあたり最大 64 データ バイト
  - 最大 8 Mbps に対応
- CAN 上のタイムトリガー (TT) 通信 (ISO 11898-4: 2004)
  - TTCAN プロトコル レベル 1 およびレベル 2 を完全にハードウェア化
- AUTOSAR に対応
- 受入フィルタリング
- 各チャネルには以下が含まれます。
  - チャネルあたり最大 64 個の専用受信バッファ
  - 受入フィルターを備えた設定可能な 2 つの受信 FIFO (各最大 64 バッファ)
  - 最大 32 個の専用送信バッファ
    - 設定可能な送信 FIFO
    - 設定可能な送信キュー
    - 設定可能な送信イベント FIFO
- プログラム可能なループバックテストモード
- パワーダウン対応
- 共有メッセージ 4 KB RAM
- 受信 FIFO トップ ポインタ ロジック
  - FIFO の DMA アクセスを有効にします
- デバッグ メッセージおよび受信 FIFO 用 DMA

25.1.2 サポートされていない機能

- 非同期シリアル通信 (ASC)
- M_TTCAN のビットエラー訂正 (IR.BEC) の割込み
  - このビットは'0'に固定されます

25.2 アーキテクチャ

25.2.1 ブロック図

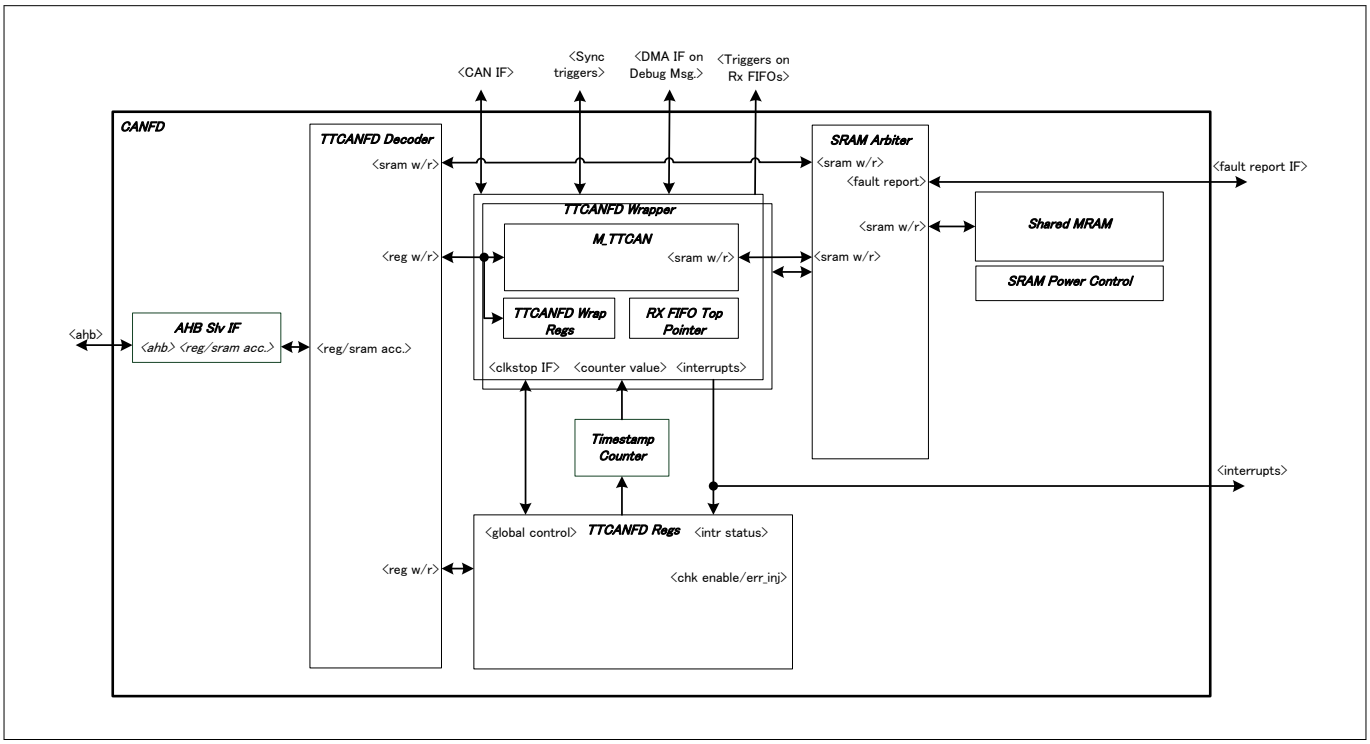


図 188 ブロック図

25.2.2 デュアルクロックソース

TTCAN チャネルには 2 つのクロック入力 (clk_can と clk_sys) があります。

システム ペリフェラルのクロック分周器から得られる `clk_can` は、CAN (または CAN FD) の動作に使用されます。20 MHz、40 MHz、または 80 MHz の使用を推奨します。

`clk_sys` は、レジスタ アクセス、SRAM アクセスなどすべての操作に使用されますが、CAN 操作には使用されません。`clk_sys` は `clk_can` と同等かそれ以上の速度で動作しなければならず、`clk_can` とは異なり EMC を最小化するためにスペクトラム拡散 PLL から派生することが許されています。

クロック コンフィギュレーションの詳細については、クロック システムセクションを参照してください。

25.2.3 割込みライン

M_TTCAN チャネルは 2 つの割込み (割込み 0 と割込み 1) ラインを提供します。M_TTCAN チャネル内の任意のソースからの割込みは、ILS と TTILS レジスタを使用することにより、割込み 0 または割込み 1 にルーティングされます。デフォルトでは、すべての割込みは interrupt0 にルーティングされます。プログラムにより、イネーブル

割込みライン 0 (ILE.EINT0) とイネーブル割込みライン 1 (ILE.EINT1) を使用することで、割込みラインは割込みソースごとに個別に有効または無効になります。

25.3 機能説明

25.3.1 動作モード

25.3.1.1 ソフトウェアの初期化

ここでは、初期化ビット (CCCR.INIT) の設定またはリセットについて説明します。CCCR.INIT ビットは以下のように設定されます。

- ソフトウェアまたはハードウェアのリセット
- MRAM で未補正ビットエラーの検出
- バスオフの選択

CCCR.INIT が設定されている間:

- CAN バスとのメッセージ送受信を停止
- CAN バス出力 canfd[0].ttcan_tx[0:1]のステータスがリセッシブ (HIGH)
- プロトコルエラーカウンターに変更なし

CCCR.INIT を設定してもコンフィギュレーション レジスタは変更されません。

CCCR.INIT をリセットすると、ソフトウェアの初期化は終了します。その後、CAN FD コントローラーは、バスのアクティビティにつながり、メッセージ転送を開始する前に、連続する 11 ビットのリセッシブ ビット (バスアイドル) が発生するのを待つことによって、CAN バス上のデータ転送に同期します。レジスタのアクセス/セット/リセットのプロパティは、コンフィギュレーション レジスタへのコンフィギュレーション チェンジ イネーブル (CCCR.CCE) アクセスに影響され、CCCR.INIT と CCCR.CCE の両方のビットが設定 (書き込み保護) されている場合に有効になります。CCCR.INIT が'1'の間、CCCR.CCE はセット/リセットされます。CCCR.CCE は CCCR.INIT がリセットされると自動的にリセットされます。

CCCR.CCE がセットされると以下のレジスタがリセットされます。

- HPMS - 優先度の高いメッセージのステータス
- RXF0S - RX FIFO 0 ステータス
- RXF1S - RX FIFO 1 ステータス
- TXFQS - TX FIFO/キュー ステータス
- TXBRP - TX バッファ要求待ち
- TXBTO - TX バッファ送信が発生
- TXBCF - TX バッファキャンセル終了
- TXEFS - TX イベント FIFO ステータス
- TTOST - TT 動作ステータス
- TTLGT - TT ローカルタイムとグローバルタイム、グローバルタイム TTLGT.GT のみリセット
- TTCTC - TT サイクルタイムとカウント
- TTCSM - TT サイクル同期マーク

上記に加えて、

- タイムアウトカウンター値 (TOCV.TOC[15:0]) は、CCCR.CCE 設定時にタイムアウト時間 (TOCC.TOP[15:0]) で設定された値にプリセットされます。
- CCCR.CCE が'1'の間、TX ハンドラと RX ハンドラのステートマシンはアイドル状態に保持されます。

以下のレジスタは、CCCR.CCE が'0'の間のみ書き込み可能です。

- TXBAR - TX バッファ追加要求
- TXBCR - TX バッファキャンセルリクエスト

テストモードイネーブル (CCCR.TEST) とバスモニターモード (CCCR.MON) は、CCCR.INIT が'1'、CCCR.CCE が'1'の間のみ CPU が設定できます。両ビットはいつでもリセットできます。

自動再送禁止 (CCCR.DAR) は、CCCR.INIT が'1'、CCCR.CCE が'1'のときのみ設定/解除できます。

メッセージ RAM 初期化

各 MRAM ワードは、CAN FD コントローラーのコンフィギュレーションの前に、'0x00000000'を書き込んでリセットする必要があります。これにより、初期化されていないワードを読み出す際の MRAM ビット エラーを防ぎ、MRAM 内の予期せぬフィルター エレメント構成を回避できます。

25.3.1.2 通常動作

ハードウェア リセット後の M_TTCAN のデフォルト動作モードは、タイム トリガーなしのイベント駆動型 CAN 通信です (TTOCF.OM = 00)。CCCR.INIT と CCCR.CCE は、TT 動作モードを変更する前に設定されます。

M_TTCAN が初期化され、CCCR.INIT が'0'にリセットされると、M_TTCAN は CAN バスに同期し、通信可能な状態になります。

受入フィルタリングを通過した後、メッセージ ID と DLC を含む受信メッセージは、専用の RX バッファ、または RX FIFO 0 または RX FIFO 1 に格納されます。

メッセージを送信するために、専用の TX バッファと TX FIFO/TX キューを初期化または更新できます。リモートフレーム受信時の自動送信は実装されていません。

25.3.1.3 CAN FD 動作

CAN FD のフレーム伝送には 2 つのバリエーションがあります。

- ビットレート切り替えなしの CAN FD フレーム
- 制御、データ、CRC の各フィールドがフレームの最初と最後よりも高いビットレートで送信される CAN FD フレーム

11 ビットの識別子と 29 ビットの識別子を持つ CAN フレームでは、それまで予約されていたビットが FDF ビットとしてデコードされます。

- FDF = リセッシブは CAN FD フレームを意味します
- FDF = ドミナントはクラシック CAN フレームを意味します

CAN FD フレームでは、FDF に続く 2 つのビット、予約ビット (RES)、ビットレートスイッチ (BRS) が、CAN FD フレーム内のビットレートを切り替えるかどうかを決定します。CAN FD のビットレート切り替えは、RES がドミナント、BRS がリセッシブです。リセッシブとして RES をコード化するのは、将来のプロトコル拡張用に予約されます。
M_TTCAN が FDF と RES をリセッシブとしてフレームを受信した場合、PSR.PXE ビットを設定することでプロトコル例外イベントを通知します。プロトコル例外処理が有効 (CCCR.PXHD = 0) の場合、次のサンプルポイントで動作状態を受信 (PSR.ACT =10) から統合 (PSR.ACT = 00) に変更します。プロトコルの例外処理が無効 (CCCR.PXHD =1) の場合、M_TTCAN はリセッシブ RES ビットをエラーとして扱い、エラーフレームで応答します。

CAN FD 動作は、CCCR.FDOE をプログラミングすることで有効になります。CCCR.FDOE が'1'の場合、CAN FD フレームの送受信が有効になります。クラシック CAN フレームの送受信は常に可能です。CAN FD フレームとクラシック CAN フレームのどちらを送信するかは、それぞれの TX バッファ エレメントの FDF ビットで設定できます。
CCCR.FDOE を'0'にすると、受信フレームは従来の CAN フレームとして解釈されるため、CAN FD フレームを受信するとエラーフレームが送信されます。CAN FD 動作が無効の場合、TX バッファ エレメントの FDF ビットが設定されていれば、CAN FD フレームは送信されません。CCCR.FDOE と CCCR.BRSE は、CCCR.INIT と CCCR.CCE がともに設定されているときのみ変更できます。

CCCR.FDOE が'0'の場合、FDF と BRS の設定は無視され、フレームはクラシック CAN フォーマットで送信されます。CCCR.FDOE = 1 かつ CCCR.BRSE = 0 のとき、TX バッファ エレメントの FDF のみが評価されます。CCCR.FDOE = 1 かつ CCCR.BRSE = 1 のとき、ビットレート切り替えによる CAN FD フレームの送信が有効になります。FDF ビットと BRS ビットが設定された TX バッファ エレメントはすべて、ビットレートを切り替えながら CAN FD フォーマットで送信されます。

CAN 動作中のモード変更は、以下の条件下でのみ推奨されます。

- CAN FD データフェーズのフォールト率は、CAN FD アービトレーションフェーズよりもかなり高いです。この場合、CAN FD のビットレート切り替えオプションを無効にしてください。
- システム起動時に、すべてのノードが CAN FD フォーマットで通信可能なクラシック CAN メッセージの送信を確認し、すべてのノードが CAN FD オペレーションに切り替わります。
- CAN パーシャルネットワークのウェイクアップメッセージは、クラシック CAN フォーマットで送信する必要があります。
- すべてのノードが CAN FD に対応していない場合、エンドオブライン プログラミングが発生します。非 CAN FD ノードは、プログラミングが完了するまでサイレント モードに保持されます。その後、すべてのノードは、クラシック CAN 通信に切り替わります。

CAN FD フォーマットでは、DLC のコーディングが標準 CAN フォーマットと異なります。DLC コード 0～8 は標準 CAN と同じコード化で、コード 9～15 は標準 CAN では 8 バイトのデータフィールドを持ちますが、表 185 に従ってコード化されます。

表 185　　CAN FD における DLC の拡張コーディング

DLC	9	10	11	12	13	14	15
データバイト数	12	16	20	24	32	48	64

CAN FD フレームでは、BRS ビットがリセッシブである場合、BRS ビットの後にフレーム内でビットタイミングが切り替わります。BRS ビットの前、CAN FD アービトレーション フェーズでは、NBTP (Nominal Bit Timing and Prescaler) レジスタで定義された公称 CAN ビット タイミングが使用されます。以下の CAN FD データ フェーズでは、データ ビット タイミングとプリスケーラ レジスタ (DBTP) で定義されたデータ フェーズのビット タイミングが使用されます。ビット タイミングは、データ フェーズのタイミングから、CRC デリミタまたはエラーが検出されたときに最初に発生する条件で切り替えられます。

CAN FD データフェーズの設定可能な最大ビットレートは、CAN クロック周波数 (clk_can) に依存します。例えば、CAN クロック周波数が 20 MHz で、設定可能な最短ビット時間が 4tq の場合、データフェーズのビットレートは 5 Mbps となります。

CAN FD とビットレート切り替え付き CAN FD の両方のデータフレームフォーマットにおいて、ビットエラーステータスインジケーター (ESI) の値は、送信開始時の送信側のエラー状態によって決定されます。送信側がエラーパッシブである場合、ESI はリセッシブに送信されます。そうでなければドミナントに送信されます。

25.3.1.4　伝搬遅延補償

CAN FD 伝搬のデータフェーズでは、1 つのノードのみが送信し、他のノードはすべて受信します。バス送信の長さは影響しません。ピン canfd[0].ttcan_tx[0:1]を介して送信する場合、M_TTCAN は、ピン canfd[0].ttcan_rx[0:1] を介してローカル CAN トランシーバーから送信データを受信します。受信したデータは送信側の遅延分遅れます。遅延がサンプルポイント (TSEG1) より前のタイムセグメントより大きい場合、ビットエラーが検出されます。伝搬遅延より短いデータ位相ビット時間を可能にするため、遅延補償が導入されます。伝搬遅延補償を使用しない場合、CAN FD フレームのデータフェーズのビットレートは伝搬遅延によって制限されます。

25.3.1.4.1 説明

M_TTCAN のプロトコル ユニットには、伝搬遅延を補償する遅延補償メカニズムが実装されます。これにより、特定の CAN トランシーバーの遅延に依存することなく、CAN FD データフェーズでより高いビットレートでの送信が可能になります。

送信ノードのデータフェーズ中にビットエラーをチェックするために、遅延送信データはセカンダリサンプルポイント (SSP) で受信データと比較されます。ビット エラーが検出された場合、送信側は次の正規のサンプル ポイントでビット エラーに対応します。アービトレーション フェーズの間、遅延補償は常に無効化されます。

伝搬遅延補償は、データビット時間が伝搬遅延より短い設定を可能にします。詳細は新しい ISO 11898-1:2015 で説明されています。DBTP.TDC ビットを設定することで有効になります。

受信ビットは SSP で送信ビットと比較されます。SSP の位置は、M_TTCAN の送信出力 canfd[0].ttcan_tx[0:1]からトランシーバを経由して受信入力 RX までの測定された遅延に、TDCR.TDCO で設定された伝搬遅延補償オフセットを加えた合計として定義されます。伝搬遅延補償オフセットは、受信ビット内の SSP の位置 (例えば、データフェーズのビット時間の半分) を調整するために使用されます。2 次サンプルポイントの位置は、mtq (clk_can 周期) の次の整数に丸められます。

PSR.TDCV は実際の伝搬遅延補償値を示します。PSR.TDCV は CCCR.INIT がセットされるとクリアされ、DBTP.TDC もセットされている間、FD フレーム送信毎に更新されます。

M_TTCAN に実装されている伝搬遅延補償については、以下の境界条件を考慮する必要があります。

- canfd[0].ttcan_tx[0:1]から canfd[0].ttcan_rx[0:1]までの測定された遅延と、設定された伝搬遅延補償オフセット TDCR.TDCO の和が、データフェーズで 6 ビット時間未満です。
- canfd[0].ttcan_tx[0:1]から canfd[0].ttcan_rx[0:1]までの測定された遅延と、設定された伝搬遅延補償オフセット TDCR.TDCO の合計が 127 mtq 以下です。この合計が 127 mtq を超える場合、伝搬遅延補償には最大値の 127 mtq が使用されます
- データ フェーズは、SSP での受信ビットのチェックを停止する CRC デリミターのサンプル ポイントで終了します。

25.3.1.4.2 伝搬遅延補償測定

DBTP.TDC に 1 を設定することで伝搬遅延補償が有効になっている場合、測定は、送信された各 CAN FD フレーム内で、ビット FDF からビット RES の立ち下りエッジで開始されます。このエッジが送信側の受信入力 canfd[0].ttcan_rx[0:1]で確認されると測定は停止します。この測定の分解能は 1 最小時間量子 (MTQ) です。

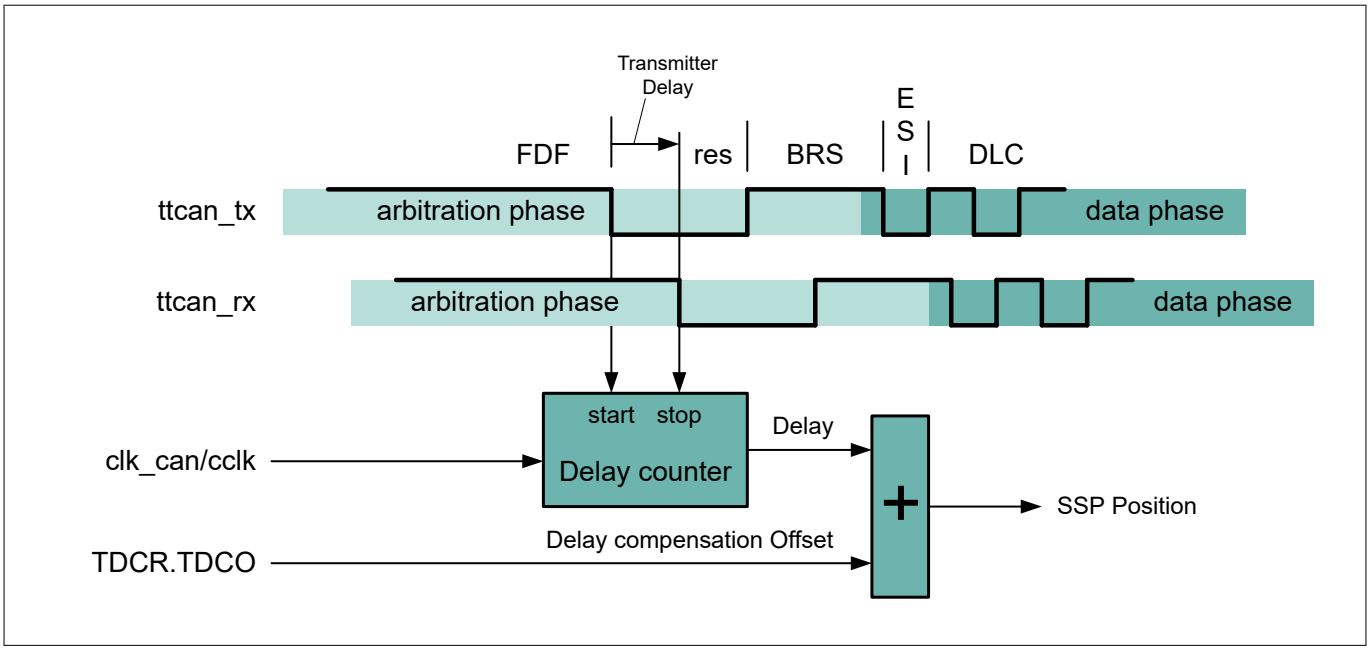


図 189　　送信側遅延測定

受信した FDF ビット内でドミナント グリッチが発生すると、受信した RES ビットの立ち下りエッジの前に伝搬遅延補償測定が終了し、結果として SSP の位置が早くなります。伝搬遅延補償フィルター ウィンドウの使用は、TDCR.TDCF を設定することで有効になります。これは、SSP ポジションの最小値を定義するものです。canfd[0].ttcan_rx[0:1]上のドミナントエッジのうち、SSP の位置が早くなるエッジは送信遅延測定では無視されます。SSP の位置が TDCR.TDCF 以上で canfd[0].ttcan_rx[0:1]が LOW の時に測定は停止します。

25.3.1.5　　制限動作モード

制限動作モードでは、ノードはデータとリモート フレームを受信し、有効なフレームを確認できますが、データ フレーム、リモート フレーム、アクティブ エラー フレーム、またはオーバーロード フレームを送信しません。エラーまたは過負荷状態の場合、ドミナントビットを送信せず、バスアイドル状態の発生を待って CAN 通信に再同期します。エラーカウンター (ECR.REC および ECR.TEC) は、エラーロギング (ECR.CEL) がアクティブな間はフリーズします。

CPU は、制限動作モードビット (CCCR.ASM) を設定することにより、CAN FD コントローラーを制限動作モードに設定できます。CCCR.ASM は CCCR.CCE と CCCR.INIT の両方が'1'に設定されているときのみ CPU によって設定できます。CCCR.ASM は CPU によっていつでもリセットできます。

CAN FD コントローラーは、TX ハンドラが MRAM からのデータ読み出しに間に合わない場合、自動的に制限動作モードに入ります。このモードを解除するためには、CPU は CCCR.ASM をリセットする必要があります。

制限動作モードは、異なる CAN ビットレートに適応するアプリケーションで使用できます。この場合、アプリケーションは異なるビットレートをテストし、有効なフレームを受信した後にモードを終了します。

***注**:　　制限動作モードは、ループバックモード (内部または外部) と組み合わせてはいけません。*

25.3.1.6　　バス モニタリング モード

M_TTCAN はバスモニターモードにおいて、CCCR.MON を'1'にプログラムするか、エラーレベル S3 (TTOST.EL = 11) が入力されると設定されます。このモードでは、M_TTCAN は有効なデータ フレームと有効なリモート フレームを受信できますが、送信の開始はできません。M_TTCAN がドミナント ビット (ACK ビット、オーバーロード フラグ、アクティブ エラー フラグ) を送信する必要がある場合、ビットは M_TTCAN がこのドミナント ビットをモニターす

るように内部的に迂回されますが、CAN バスはリセッシブ状態のままです。バス モニタリング モードでは、TXBRP レジスタはリセット状態です。

バス モニタリング モードは、ドミナント ビットの送信によって CAN バスに影響を与えることなく、CAN バスのトラフィックを分析するために使用されます。図 190 に、バス モニタリング モードにおける M_TTCAN への信号 canfd[0].ttcan_tx[0:1]と canfd[0].ttcan_rx[0:1]の接続を示します。

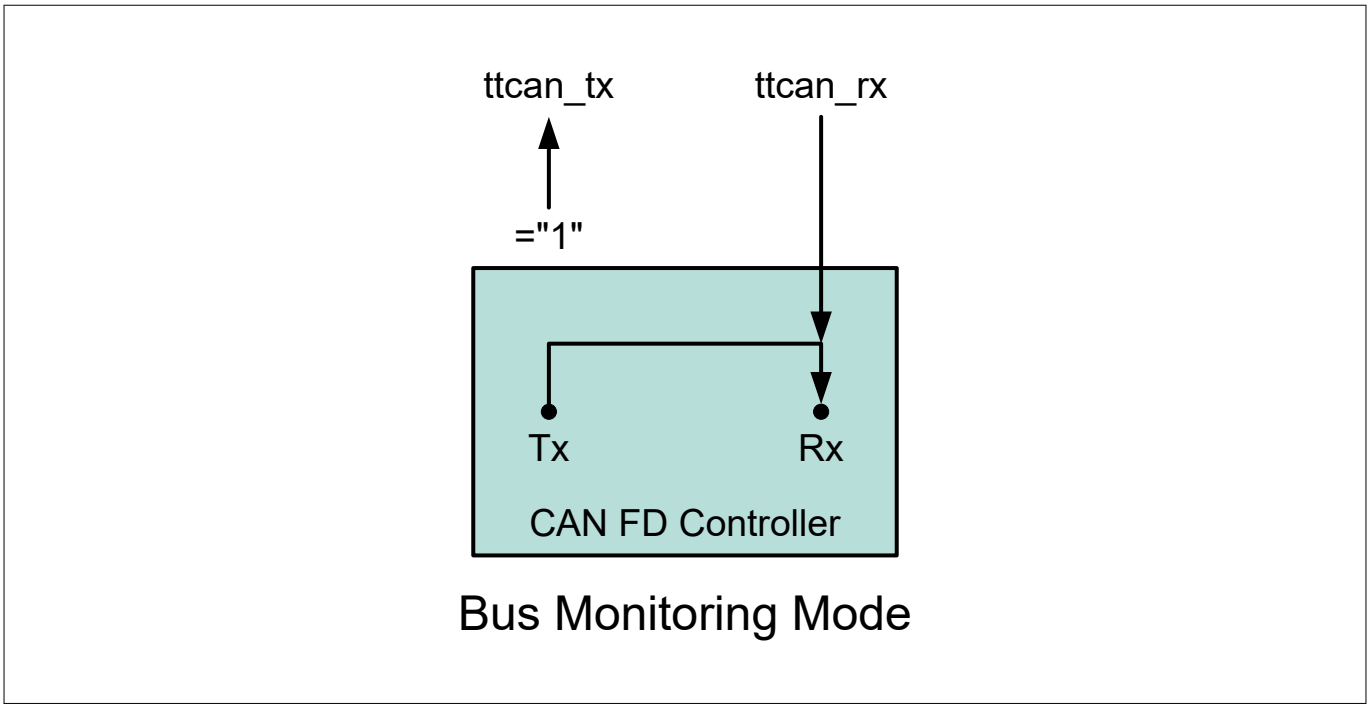


図 190　　バス モニタリング モードでのピン制御

25.3.1.7　　自動再送信の無効化

M_TTCAN は、送信中にアービトレーションを失ったフレームやエラーによって妨害されたフレームの自動再送信をサポートします。デフォルトでは、自動再送信が有効になっています。タイムトリガー通信 (ISO 11898-1:2015 の 9.2 章に記載) をサポートするために、自動再送信は CCCR.DAR を介して無効化されます。

DAR モードでは、すべての送信は CAN バス上で開始された後、自動的にキャンセルされます。TX Request Pending ビット TXBRP.TRPx は、送信が成功した後、キャンセルされた時点でまだ送信が開始されていない場合、アービトレーションが失われたために中断された場合、またはフレーム送信中にエラーが発生した場合にリセットされます。

- 送信に成功: 対応 TX バッファ送信完了ビット TXBTO.TOx が設定され、対応 TX バッファキャンセル済ビット TXBCF.CFx が設定されていません。
- キャンセルにもかかわらず送信に成功: 対応 TX バッファ送信完了ビット TXBTO.TOx と対応 TX バッファキャンセル完了ビット TXBCF.CFx が共に設定されます。
- アービトレーションが失われた、またはフレーム伝送が妨害された: 対応 TX バッファ送信ビット TXBTO.TOx がセットされておらず、対応 TX バッファキャンセル終了ビット TXBCF.CFx が設定されます。

フレーム送信に成功し、TX イベントの保存が有効になっている場合、TX イベント FIFO エレメントにイベントタイプ ET = 10 (送信にもかかわらずキャンセル) が書き込まれます。

25.3.1.8　　電源オフ (スリープモード)

M_TTCAN チャネルは、クロック停止要求 (CTL.STOP_REQ) によってパワーダウン モードに設定できます。クロック停止要求がアクティブである限り、STOP_REQ ビットは'1'です。

保留中の送信要求がすべて完了すると、M_TTCAN はバスアイドル状態が検出されるまで待機します。その後、M_TTCAN は CCCR.INIT を'1'に設定し、それ以上の CAN 転送を防止します。これで M_TTCAN は、クロック停止アクノリッジ (STATUS.STOP_ACK) を設定することで、パワーダウンの準備ができたことを確認します。チャネルから ACK を受信すると、ハードウェアは自動的にそれぞれのチャネルのクロックをオフにします。

パワーダウンモードから抜けるには、アプリケーションは CTL.STOP_REQ をリセットする必要があります。M_TTCAN は、STATUS.STOP_ACK をリセットすることでこれを確認します。その後、アプリケーションはビット CCCR.INIT をリセットすることで、CAN 通信を再開します。

CTL.STOP_REQ によりクロック停止要求がトリガーされた場合、STATUS.STOP_ACK ビットが設定されるまでクリアしてはいけません。

***注**:* *パワーダウン制御には TTCAN CCCR.CSR レジスタを使用せず、代わりに CTL.STOP_REQ を使用します。同様に、CCCR.CSA の代わりに STATUS.STOP_ACK を使用します。*

25.3.1.9 テスト モード

TEST レジスタへの書き込みアクセスを有効にするためには、テスト モード イネーブル ビット (CCCR.TEST) を'1'に設定します。これにより、テスト モードとテスト機能を設定できます。

TEST.TX をプログラミングすることで、CAN 送信ピン canfd[0].ttcan_tx[0:1]に 4 つの出力機能が利用できます。シリアル データ出力のデフォルト機能とは別に、M_TTCAN のビット タイミングをモニターするために CAN サンプル ポイント信号をドライブできます。また、定常的なドミナントまたはリセッシブ値をドライブすることも可能です。canfd[0].ttcan_rx[0:1]ピンの実際の値は TEST.RX から読み出せます。どちらの機能も、CAN バスの物理層のチェックに使用できます。

CAN クロックとホスト クロックドメイン間の同期メカニズムにより、TEST.TX に書き込んでから新しいコンフィギュレーションが出力ピン canfd[0].ttcan_tx[0:1]に表示されるまで、ホスト クロック数周期分の遅延が発生する場合があります。これは、TEST.RX を介して入力ピン canfd[0].ttcan_rx[0:1]を読み出す場合にも当てはまります。

***注**:* *テストモードは、プロダクション テストまたはセルフテストにのみ使用してください。canfd[0].ttcan_tx[0]ピンのソフトウェア制御は、すべての CAN プロトコル機能に干渉します。テストモードをアプリケーションに使用することは推奨されません。*

25.3.1.9.1 外部ループバック モード

M_TTCAN は、TEST.LBCK を'1'にプログラムすることにより、外部ループバックモードに設定できます。ループバックモードでは、M_TTCAN は送信メッセージを受信メッセージとして扱い、(受入フィルタリングを通過した場合) RX バッファまたは RX FIFO に格納します。図 191 に、外部ループバックモードにおける M_TTCAN への信号 canfd[0].ttcan_tx[0:1]と canfd[0].ttcan_rx[0:1]の接続を示します。

このモードはハードウェア セルフ テスト用に用意されます。外部刺激から独立するため、M_TTCAN はループバックモードでアクノリッジエラー (データ/リモートフレームのアクノリッジスロットでサンプリングされたリセッシブビット) を無視します。このモードでは、M_TTCAN は TX 出力から RX 入力への内部フィードバックを行います。M_TTCAN では、canfd[0].ttcan_rx[0:1]入力ピンの実際の値は無視されます。送信メッセージは canfd[0].ttcan_tx[0:1]ピンでモニターできます。

25.3.1.9.2 内部ループバック モード

TEST.LBCK ビットと CCCR.MON ビットを'1'にプログラムすることにより、内部ループバックモードに入ります。このモードは「ホットセルフテスト」に使用できます。つまり、`canfd[0].ttcan_tx[0:1]`と `canfd[0].ttcan_rx[0:1]`ピンに接続された実行中の CAN システムに影響を与えることなく、M_TTCAN をテストできます。このモードでは、`canfd[0].ttcan_rx[0:1]`ピンは M_TTCAN から切り離され、`canfd[0].ttcan_tx[0:1]`ピンはリセッシブに保持され

ます。図 191 に、canfd[0].ttcan_tx[0:1]と canfd[0].ttcan_rx[0:1]を内部ループバックモードで M_TTCAN に接続していることを示します。

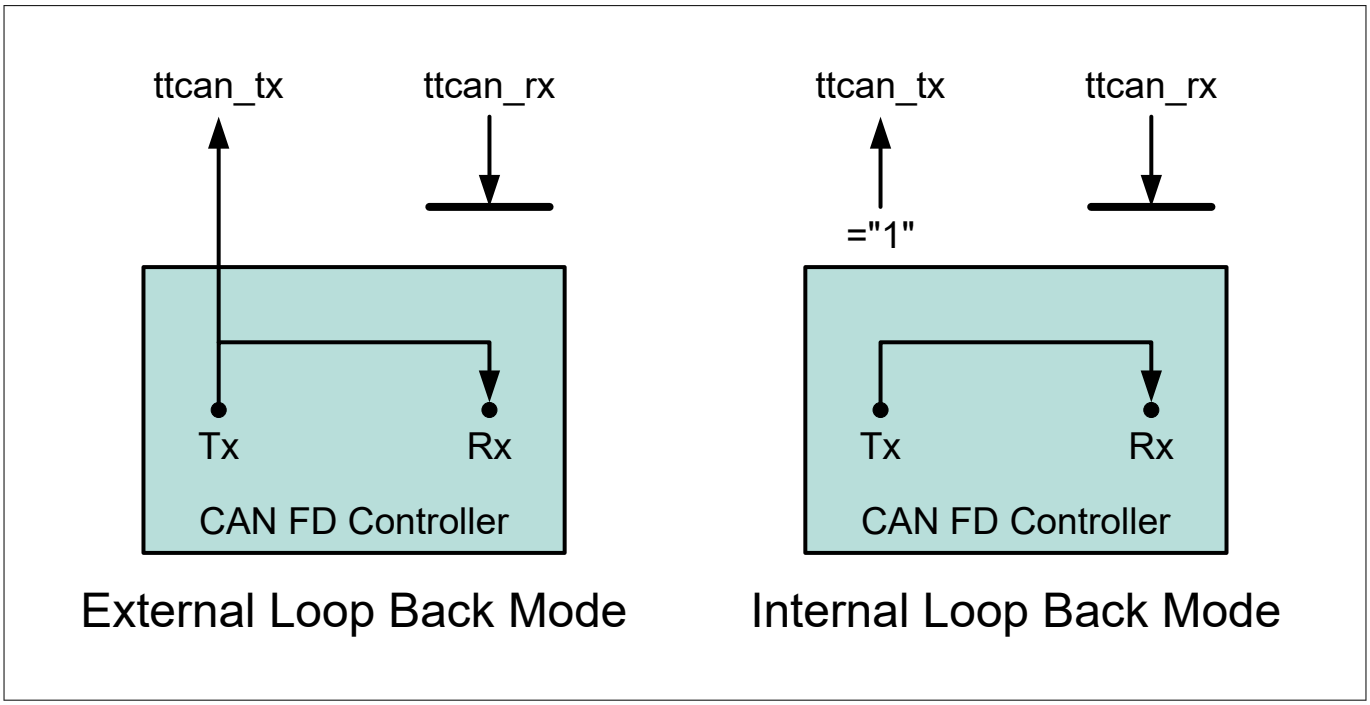


図 191　　ループバックモードでのピン制御

25.3.1.10　アプリケーション ウォッチドッグ

アプリケーション ウォッチドッグは、TTOST レジスタを読み出すことで機能します。アプリケーション ウォッチドッグが時間内に対応されない場合、TTOST.AWE ビットが設定され、すべての TTCAN 通信が停止され、M_TTCAN はバス監視モードに設定されます。

TT アプリケーション ウォッチドッグは、アプリケーション ウォッチドッグ制限 TTOCF.AWL を'0x00'に設定することで無効にできます。TTCAN アプリケーション プログラムでは、TT アプリケーション ウォッチドッグを無効にしてはいけません。

25.3.2　タイムスタンプ生成

M_TTCAN チャネルは、16 ビットのカウンターを使い、メッセージの送受信を記録します。これにより、アプリケーション ソフトはイベントの発生順序を知ることができます。

タイムスタンプ カウンターは TS_CTL レジスタで設定されます。TS_CTL.ENABLED ビットはカウンターを有効にします。イネーブルにすると、TS_CTL.PRESCALE [15:0]に従ってインクリメントを開始します。アプリケーションは TS_CNT レジスタをとおしてカウンター値を読み出せます。TS_CNT レジスタへの書き込みアクセスは TS_CNT をクリアします。

タイムスタンプカウンターが有効な場合、プリスケーラ用内部カウンターは clk_sys のサイクルごとにカウントされます。カウンター値がプリスケーラ値に達すると、タイムスタンプカウンターが 1 つインクリメントし、内部プリスケーラカウンターがクリアされます。TS_CTL.PRESCALE が変化したら、TS_CNT を書き込んでリセットします。これにより、内部プリスケーラ カウンターが TS_CTL.PRESCALE の新しい値に即座に追従するようになります。

共有タイムスタンプカウンターはラップアラウンドカウンターです。カウンターが 1 周すると、M_TTCAN チャネルの IR.TSW が立ち上がります。

フレーム受信/送信の開始時に、タイムスタンプ カウンター値がキャプチャされ、RX バッファ/RX FIFO (RXTS[15:0]) または TX イベント FIFO (TXTS[15:0]) エレメントのタイムスタンプ部に格納されます。

注: ディープスリープ モードではカウンター値 TS_CNT は保持されませんが、TS_CTL は保持されます。

25.3.3 タイムアウトカウンター

RX FIFO 0、RX FIFO 1、および TX イベント FIFO のタイムアウト状態を通知するために、M_TTCAN は 16 ビットのタイムアウトカウンターを供給します。ダウンカウンターとして動作し、TSCC によって制御される同じプリスケーラを使用します。タイムスタンプカウンターとしての TCP です。プリスケーラ TSCC.TCP は、タイムアウト カウンターを CAN ビットの倍数 (1～16) でクロックするように設定されます。タイムアウト カウンターは、レジスタ TOCC によって設定されます。実際のカウンター値は TOCV.TOC から読み出せます。

タイムアウトカウンターは、CCCR.INIT =0 の間のみ開始できます。CCCR.INIT = 1 の場合、例えば M_TTCAN が Bus_Off 状態になったときなどで停止します。

動作モードは TOCC.TOS によって選択されます。連続モードで動作している場合、CCCR.INIT がリセットされるとカウンターがスタートします。TOCV への書き込みはカウンターを TOCC.TOP で設定された値にプリセットし、ダウンカウントを継続します。

タイムアウトカウンターが FIFO の 1 つによって制御されている場合、空の FIFO はカウンターを TOCC.TOP によって設定された値にプリセットします。ダウンカウントは、最初の FIFO エレメントが格納された時点で開始されます。TOCV に書き込んでも効果はありません。

カウンターがゼロになると、割込みフラグ IR.TOO がセットされます。連続モードでは、カウンターは TOCC.TOP で直ちにリスタートします。

注: タイムアウトカウンターのクロック信号は、CAN コアのサンプルポイント信号から得られます。したがって、タイムアウトカウンターがデクリメントされる時間は、CAN コアの同期/再同期メカニズムによって異なる場合があります。CAN FD のビットレート切り替え機能が使用されている場合、アービトレーションフィールドとデータフィールドでタイムアウトカウンターのクロックが異なります。

25.3.4 受信処理

RX ハンドラは、RX FIFO の put および get インデックスと同様に、受信メッセージの受入フィルタリング、RX バッファへの転送、または 2 つの RX FIFO のうちの 1 つへの転送を制御します。

25.3.4.1 受入フィルタリング

M_TTCAN では、標準識別子用と拡張識別子用の 2 種類の受入フィルターを設定できます。これらのフィルターは RX バッファまたは RX FIFO 0,1 に割り当てられます。受入フィルタリングでは、各フィルターのリストは、エレメント#0 から最初に一致するエレメントまで実行されます。受入フィルタリングは、最初にに一致したエレメントで停止します。このメッセージでは、以下のフィルターエレメントは評価されません。

主な特長は以下のとおりです。

- 各フィルターエレメントは、次のように設定できます。
  - レンジフィルター (from - to)
  - 1 つまたは 2 つの専用 ID のフィルター
  - クラシック ビットマスク フィルター
- 各フィルターエレメントは、受容または拒絶フィルタリング用に設定可能
- 各フィルターエレメントは個別に有効/無効を設定可能
- フィルターは順次チェックされ、最初に一致したフィルター エレメントで実行は停止します。

関連する設定レジスタは以下のとおりです。

- グローバルフィルター設定 (GFC)

- 標準 ID フィルター設定 (SIDFC)
- 拡張 ID フィルター設定 (XIDFC)
- 拡張 ID AND マスク (XIDAM)

フィルター エレメント (SFEC/EFEC) の設定により、一致すると以下のいずれかのアクションがトリガーされます。

- 受信フレームを FIFO 0 または FIFO 1 に格納
- 受信したフレームを RX バッファに格納
- 受信したフレームを RX バッファに格納し、フィルター イベント端子にパルスを発生
- 受信したフレームを却下
- 高優先メッセージ割込みフラグの設定 IR.HPM
- 高優先メッセージ割込みフラグ IR.HPM をセットし、受信フレームを FIFO 0 または FIFO 1 に格納

受入フィルタリングは、完全な識別子を受信した後に開始されます。受入フィルタリングが完了した後、一致する RX バッファまたは RX FIFO が見つかれば、メッセージ ハンドラは一致する RX バッファまたは RX FIFO に受信メッセージ データを 32 ビット単位で書き込み始めます。CAN プロトコル コントローラーがエラー状態 (CRC エラーなど) を検出した場合、このメッセージは、影響を受ける RX バッファまたは RX FIFO に次のような影響を与えながら破棄されます。

RX バッファ

一致する RX バッファの新しいデータフラグはセットされませんが、RX バッファは受信データで (部分的に) 上書きされます。エラーの種類については、それぞれ PSR.LEC と PSR.DLEC を参照してください。

RX FIFO

一致する RX FIFO の Put Index は更新されませんが、関連する RX FIFO エレメントは受信データで (部分的に) 上書きされます。エラーの種類については、それぞれ PSR.LEC と PSR.DLEC を参照してください。一致する RX FIFO が上書きモードで動作する場合、RX FIFO 上書きモードで説明されている境界条件を考慮する必要があります。

注: 受け入れられたメッセージが 2 つの RX FIFO の 1 つ、または RX バッファに書き込まれると、使用されたフィルターとは無関係に、変更されていない受信識別子が格納されます。受入フィルター処理の結果は、設定されたフィルターエレメントの順序に依存します。

レンジフィルター

SF1ID/SF2ID resp. EF1ID/EF2ID で定義された範囲の Message ID を持つすべての受信フレームに一致します。

拡張フレームでレンジフィルタリングを使用する場合、2 つの可能性があります。

- EFT = 00: 受信したフレームの Message ID を XIDAM と AND してからレンジフィルターをかけます
- EFT = 11: XIDAM はレンジフィルタリングには使用しません。

特定 ID のフィルター

フィルター エレメントは、1 つまたは 2 つの特定のメッセージ ID をフィルターするように設定できます。特定の Message ID をフィルタリングするためには、フィルター エレメントにそれぞれ SF1ID = SF2ID と EF1ID = EF2ID を設定する必要があります。

クラシック ビットマスク フィルター

クラシック ビットマスク フィルタリングは、受信した Message ID の単一ビットをマスクすることによって、Message ID のグループをフィルタリングすることを意図しています。クラシック ビットマスクフィルタリングでは、SF1ID/EF1ID はメッセージ ID フィルターとして使用され、SF2ID/EF2ID はフィルターマスクとして使用されます。

フィルターマスクのゼロビットは、設定済み ID フィルターの対応するビット位置をマスクします。例えば、受信したメッセージ ID のそのビット位置の値は、受入フィルタリングにおいて関連性を持たないことになります。受信した Message ID のうち、対応するマスク ビットが 1 であるビットのみが受入フィルタリングに関係します。

すべてのマスクビットが 1 の場合、受信した Message ID と Message ID フィルターが一致したときのみ一致します。すべてのマスクビットがゼロであれば、すべての Message ID が一致します。

標準メッセージ ID フィルター

図 192 に、標準メッセージ ID (11 ビット識別子) フィルタリングのフローを示します。標準メッセージ ID フィルター エレメントは 標準メッセージ ID フィルター エレメントに記述されます。

GFC および SIDFC メッセージ ID によって制御され、受信フレームのリモート送信要求ビット (RTR) および識別子拡張ビット (IDE) は、設定されたフィルターエレメントのリストと比較されます。

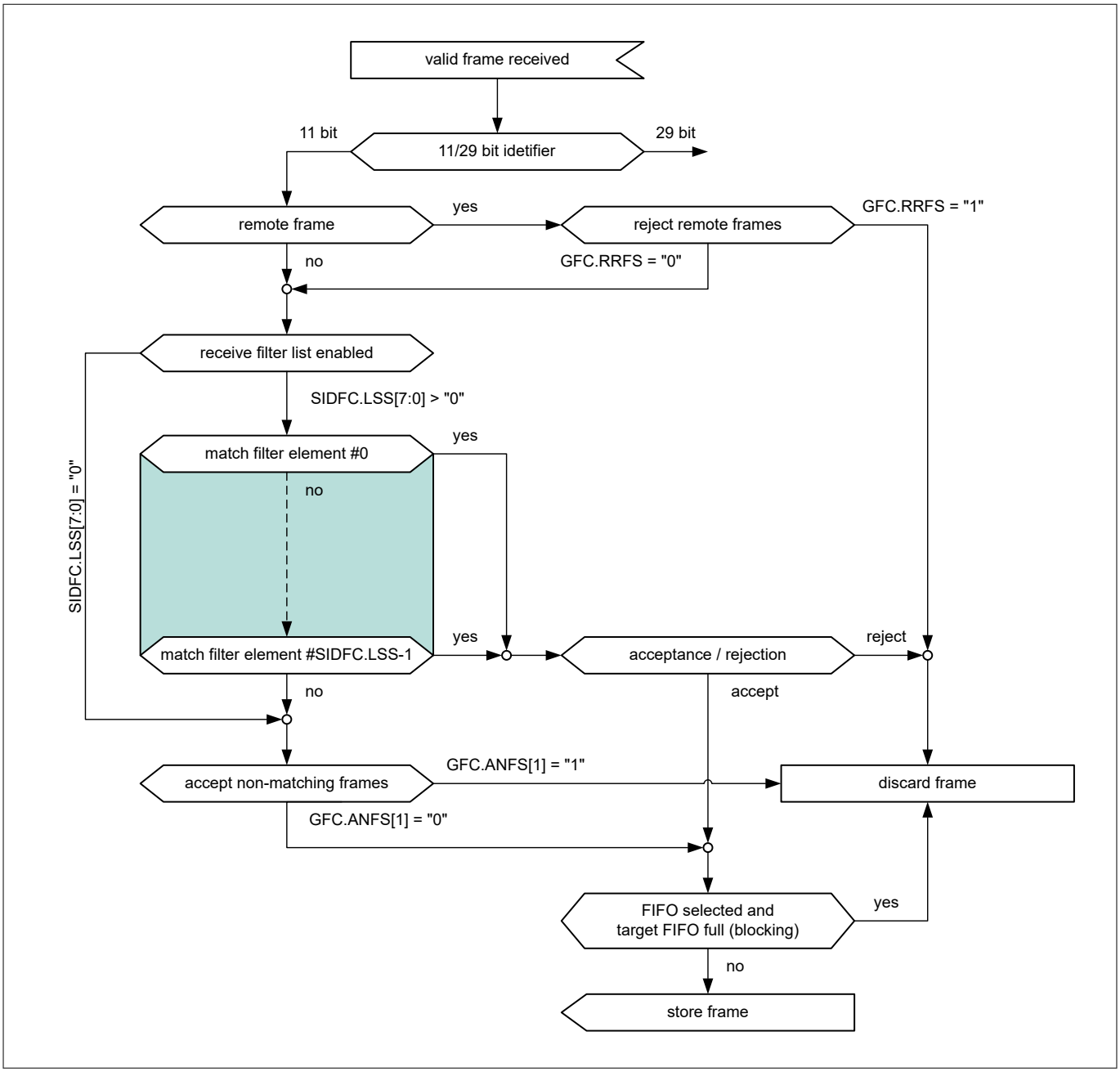


図 192　　標準メッセージ ID フィルターのフロー図

拡張メッセージ ID フィルター

図 193 に、拡張メッセージ ID (29 ビット識別子) フィルタリングのフローを示します。拡張メッセージ ID フィルター エレメントは 拡張メッセージ ID フィルター エレメントに記述されます。これは、GFC および XIDFC メッセージ ID、

受信フレームの RTR ビット、および IDE ビットによって制御され、設定されたフィルター エレメントのリストと比較されます。拡張 ID AND マスク XIDAM[28:0]は、フィルターリストが実行される前に受信識別子と AND されます。

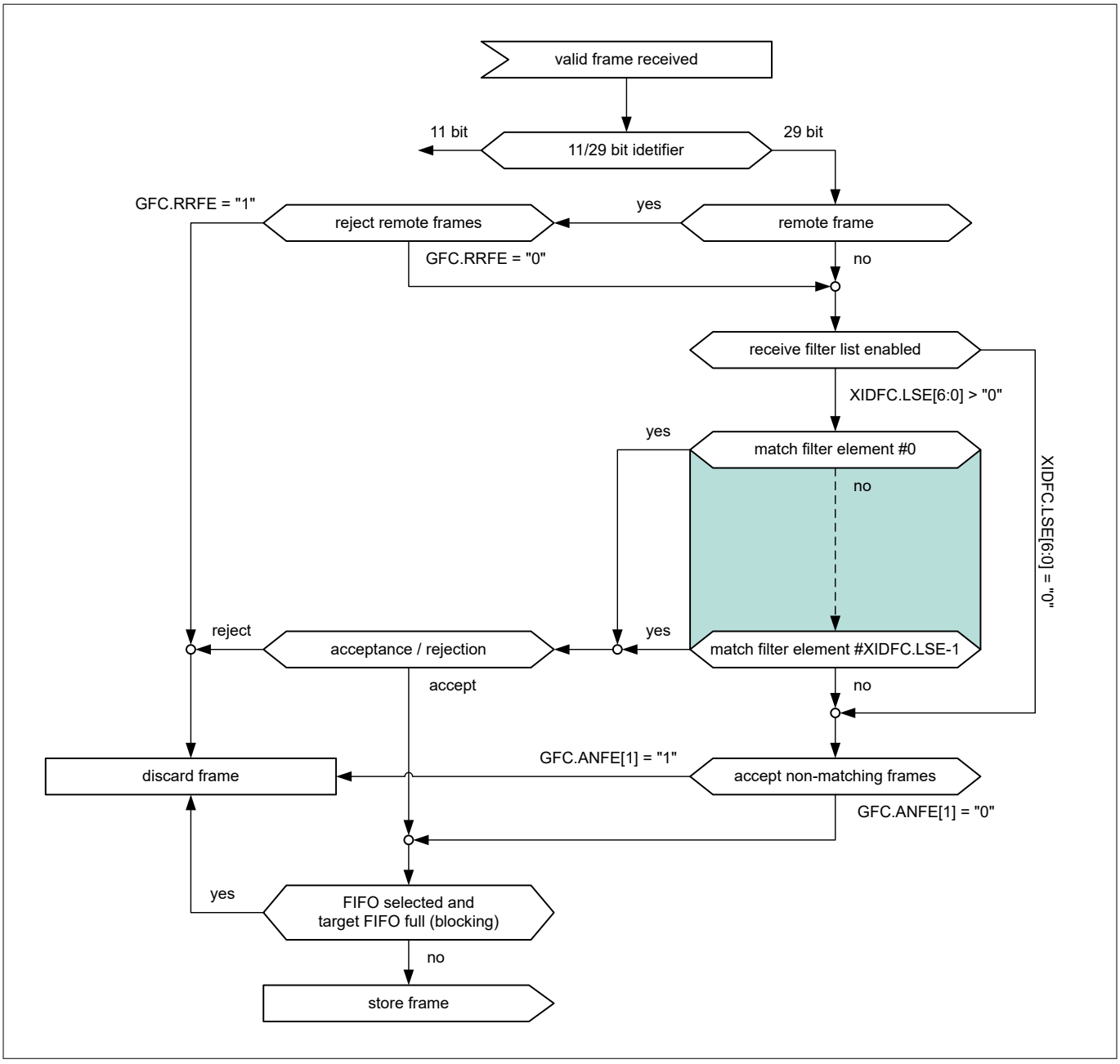


図 193　　拡張メッセージ ID フィルターのフロー図

25.3.4.2　受信 FIFO

RX FIFO 0 と RX FIFO 1 は、それぞれ最大 64 エレメントを保持できるように設定されます。2 つの RX FIFO は RXF0C と RXF1C レジスタで設定されます。

受入フィルタリングを通過した受信メッセージは、マッチング フィルター エレメントによって設定された RX FIFO に転送されます。RX FIFO 0 と RX FIFO 1 で使用可能なフィルター機構については、受入フィルタリングを参照してください。RX バッファと FIFO エレメントに、RX FIFO エレメントを示します。

RX FIFO のオーバーフローを避けるために、RX FIFO のウォーターマークを使用できます。RX FIFO のフィルレベルが RXFnC.FnWM で設定された RX FIFO ウォーターマークに達すると、割込みフラグ IR.RFnW が設定されます。RX FIFO Put Index が RX FIFO Get Index に達すると、RXFnS.FnF によって RX FIFO Full 状態が通知されます。ま

た、割込みフラグ IR.RFnF がセットされます。FIFO のウォーターマーク割込みフラグは、DMA のトリガーに使用できます。FIFO に対する DMA リクエストは、それぞれのトリガーがソフトウェアによってクリアされるまでセットされたままです。ソフトウェアは、ウォーターマークフラグをクリアすることでトリガーをクリアできます。

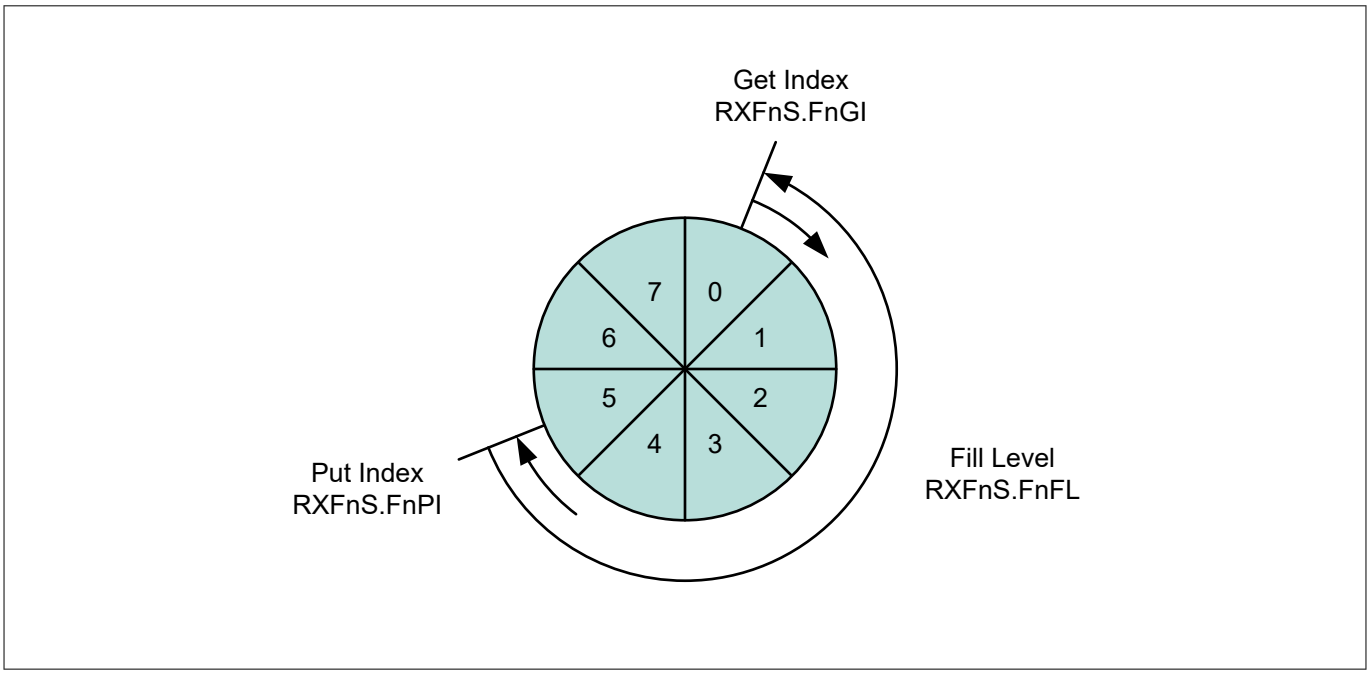


図 194　　RX FIFO の状態

RX FIFO から読み出す場合は、対応する RX FIFO 開始アドレス RXFnC.FnSA に、RX FIFO Get Index RXFnS.FnGI× FIFO エレメントサイズを加算する必要があります。RXFIFO トップ ポインタ ロジックが CAN FD コントローラーに追加され、読み出しが高速化されます。RX FIFO トップポインタの RX FIFO トップポインタを参照してください。

表 186　　RX バッファ/FIFO エレメントサイズ

RXESC.RBDS[2:0] RXESC.FnDS[2:0]	データ フィールド [バイト]	FIFO エレメントサイズ[RAM ワード]
000	8	4
001	12	5
010	16	6
011	20	7
100	24	8
101	32	10
110	48	14
111	64	18

25.3.4.2.1　　RX FIFO ブロッキング モード

RX FIFO ブロッキングモードは RXFnC.FnOM= 0 で設定されます。これは RX FIFO のデフォルト動作モードです。

RX FIFO がフル状態 (RXFnS.FnPI = RXFnS.FnGI) になると、少なくとも 1 つのメッセージが読み出され、RX FIFO の Get Index がインクリメントされるまで、対応する RX FIFO にはそれ以上メッセージが書き込まれません。RX FIFO のフル状態は RXFnS.FnF = 1 で通知されます。また、割込みフラグ IR.RFnF が設定されます。

対応する RX FIFO がフルの間にメッセージを受信した場合、このメッセージは破棄され、RXFnS.RFnL = 1 によってメッセージ喪失状態が通知されます。また、割込みフラグ IR.RFnL がセットされます。

25.3.4.2.2 RX FIFO 上書きモード

RX FIFO 上書きモードは RXFnC.FnOM = 1 で設定されます。RXFnS.FnF = 1 によって RX FIFO フル状態 (RXFnS.FnPI = RXFnS.FnGI) が通知されると、FIFO に次に受け入れられるメッセージは最も古い FIFO メッセージを上書きします。Put Index と Get Index はともに 1 ずつインクリメントされます。

RX FIFO が上書きモードで動作し、RXFIFO のフル状態が通知された場合、RX FIFO エレメントの読み出しは、少なくとも Get Index + 1 から開始する必要があります。これは、CPU が MRAM からの読み出し (GetIndex) を行っている間に、受信したメッセージが MRAM に書き込まれる (Put Index) 可能性があるためです。この場合、矛盾したデータがそれぞれの RX FIFO エレメントから読み出される可能性があります。RX FIFO からの読み出し時に GetIndex にオフセットを追加することで、この問題を回避できます。オフセットは、CPU が RXFIFO にアクセスする速度に依存します。図 195 に、RX FIFO の読み出し時の Get Index に対する 2 のオフセットを示します。この場合、エレメント 1 と 2 に格納された 2 つのメッセージは失われます。

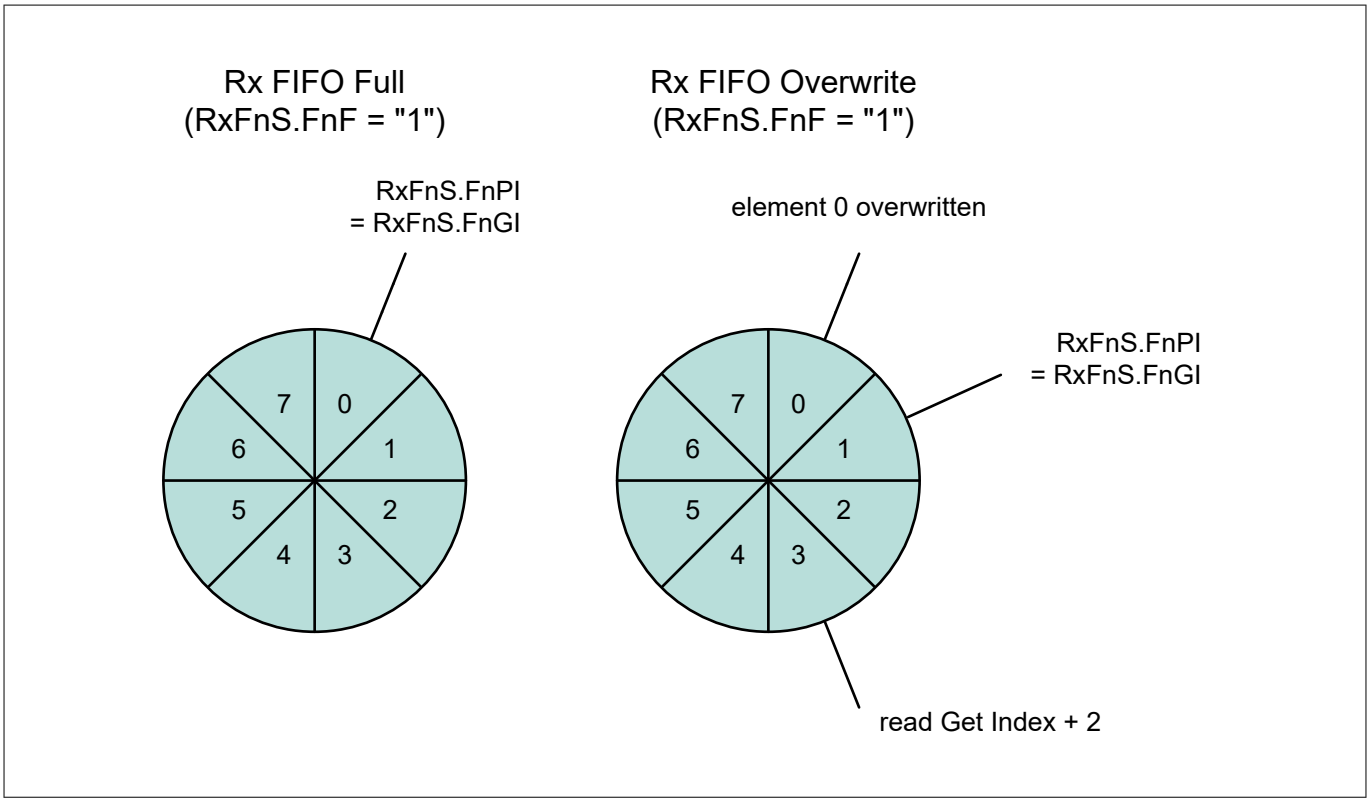


図 195 RX FIFO のオーバーフロー処理

RX FIFO から読み出した後、最後に読み出したエレメントの番号を RX FIFO Acknowledge Index RXFnA.FnA に書き込む必要があります。これにより、Get Index がそのエレメント番号までインクリメントされます。この RX FIFO エレメントに Put Index がインクリメントされなければ、RX FIFO のフル状態はリセットされます (RXFnS.FnF = 0)。

25.3.4.2.3 RX FIFO トップポインタ

M_TTCAN は 2 つの受信 FIFO に対応しています。これらの FIFO からの読み出しには、以下の手順が必要です。

- 読み出しポインタの取得
- 正しい MRAM アドレスを計算

- MRAM からデータの読み出し
- 読み出しポインタの更新

これらのステップを回避するため、RX FIFO トップ ポインタ ロジックが CAN FD コントローラーに統合されます。これは、データを読み出すための 1 つの MMIO ロケーション (RXFTOPn_DATA; n = 0,1) を提供します。このようなハードウェア ロジックを使うことには、次のような利点があります。

- より高性能なデータアクセス
- バスの通信量が減少
- CPU 負荷の軽減
- 電力削減
- FIFO への DMA アクセスを有効化

このロジックは RXFTOP_CTL.FnTPE が設定されると有効になります。このビットを設定することで、FIFO トップアドレス (FnTA) と内部メッセージワードカウンターを設定できるようになります。トップ ステータス レジスタ (RXFTOPn_STAT) の受信 FIFO はそれぞれの FIFO トップ アドレスを示し、RXFTOPn_DATA はトップ アドレスに位置するデータを示します。両レジスタの詳細については、レジスタ定義を参照してください。

RXFTOPn_DATA が読み出されると、トップポインタロジックは TTCAN チャネルの RX FIFO Acknowledge Index (RXFnA.FnA) も更新します。

***注**:* *チャネルのコンフィギュレーション中 (CCCR.CCE = 1) は、トップポインタロジックは無効となります。ロジックが無効な状態で RXFTOPn_DATA を読み出すと、無効なデータが返されます。*

***注**:* *RX FIFO サイズが'1'に設定されている場合、RXFIFO トップポインタ機能は使用しません。代わりに、受信したメッセージを MRAM から直接読み出します。*

25.3.4.3 専用 RX バッファ

M_TTCAN は最大 64 の専用 RX バッファをサポートします。専用 RX バッファ セクションの開始アドレスは RXBC.RBSA を介して設定されます。

各 RX バッファに対して、SFEC/EFEC = 111 および SFID2/EFID2[10:9] = 00 の標準または拡張メッセージ ID フィルター エレメントを設定する必要があります (標準メッセージ ID フィルター エレメントと拡張メッセージ ID フィルター エレメントを参照)。

受信したメッセージがフィルター エレメントに受け入れられた後、そのメッセージはフィルター エレメントが参照するメッセージ RAM の RX バッファに格納されます。フォーマットは RX FIFO エレメントと同じです。さらに、割込みレジスタのフラグ IR.DRX (専用 RX バッファに格納されたメッセージ) が設定されます。

表 187 RX バッファのフィルター設定例

フィルターエレメント	SFID1[10:0] EFID1[28:0]	SFID2[10:9] EFID2[10:9]	SFID2[5:0] EFID2[5:0]
0	ID message 1	00	00 0000
1	ID message 1	00	00 0001
2	ID message 1	00	00 0010

一致した受信メッセージの最後のワードがメッセージ RAM に書き込まれた後、NDAT1 と 2 つのレジスタのそれぞれの New data フラグがセットされます。New data フラグがセットされている限り、それぞれの RX バッファは受信したマッチングフレームからの更新に対してロックされます。ニュー データ フラグは、ホストがそれぞれのビット位置に'1'を書き込むことでリセットされます。

RX バッファの New data フラグがセットされている間、特定の RX バッファを参照するメッセージ ID フィルターエレメントは一致せず、受入フィルタリングが再開されます。以下のメッセージ ID フィルター エレメントは、受信したメッセージを別の RX バッファに格納するか、RX FIFO に格納するか、あるいはフィルター構成に応じてメッセージを拒否します。

- 割込みフラグ IR.DRX のリセット
- 新しいデータ レジスタの読み出し
- メッセージ RAM からメッセージの読み出し
- 処理されたメッセージの新しいデータ フラグのリセット

25.3.4.4 CAN サポートのデバッグ

デバッグ メッセージは RX バッファに格納されます。3 つの連続した RX バッファ (例えば、#61、#62、#63) は、デバッグ メッセージ A、B、C を格納するために使用されます。フォーマットは RX バッファと RX FIFO エレメントで共通です。

デバッグ メッセージをフィルタリングするためには、SFEC/EFEC = 111 の標準/拡張フィルターエレメントを設定します。これらのフィルター エレメントに一致したメッセージは SFID2/EFID2[5:0]でアドレスされた RX バッファに格納されます。

メッセージ C が保存された後、DMA 要求がアクティブになり、DMA 制御の下でメッセージ RAM から 3 つのメッセージを読み出せます。デバッグ メッセージを保持する RAM ワードは、DMA 要求が有効になっている間、M_TTCAN によって変更されることはありません。動作は、New data フラグがセットされた RX バッファの動作と同様です。

DMA 転送が完了した後、DMA からの承認が DMA リクエストをリセットします。これにより M_TTCAN は、次のデバッグ メッセージのセットを受信する準備ができました。

25.3.4.4.1 デバッグメッセージのフィルタリング

デバッグ メッセージは、3 つのデバッグ メッセージにそれぞれ 1 つの標準/拡張メッセージ ID フィルター エレメントを設定することでフィルタリングされます。デバッグ メッセージをフィルタリングするフィルター エレメントを有効にするためには、SFEC/EFEC を"111"に設定する必要があります。この場合、SFID1/SFID2 と EFID1/EFID2 フィールドには異なる意味があります (図 192 エレメントと図 193 エレメントを参照)。SFID2/EFID2[10:9]はデバッグメッセージ処理ステートマシンを制御し、SFID2/EFID2[5:0]は受信したデバッグメッセージの格納場所を制御します。

デバッグ メッセージが格納されるとき、それぞれの New データ フラグも IR.DRX も設定されません。デバッグ メッセージの受信は、RXF1S.DMS を介してモニターできます。

表 188 デバッグメッセージのフィルター設定例

フィルターエレメント	SFID1[10:0] EFID1[28:0]	SFID2[10:9] EFID2[10:9]	SFID2[5:0] EFID2[5:0]
0	ID デバッグ メッセージ A	01	11 1101
1	ID デバッグ メッセージ B	10	11 1110
2	ID デバッグ メッセージ C	11	11 1111

25.3.4.4.2 デバッグ メッセージの処理

デバッグ メッセージ処理ステート マシンは、デバッグ メッセージが 3 つの連続した RX バッファに正しい順序で格納されることを保証します。メッセージの欠落がある場合、プロセスは再開されます。DMA リクエストは、3 つのデバッグ メッセージ A、B、C がすべて不正な順序で受信された場合にのみアクティブになります。

デバッグ メッセージ処理ステート マシンの状態は、RXF1S.DMS を介して通知されます。

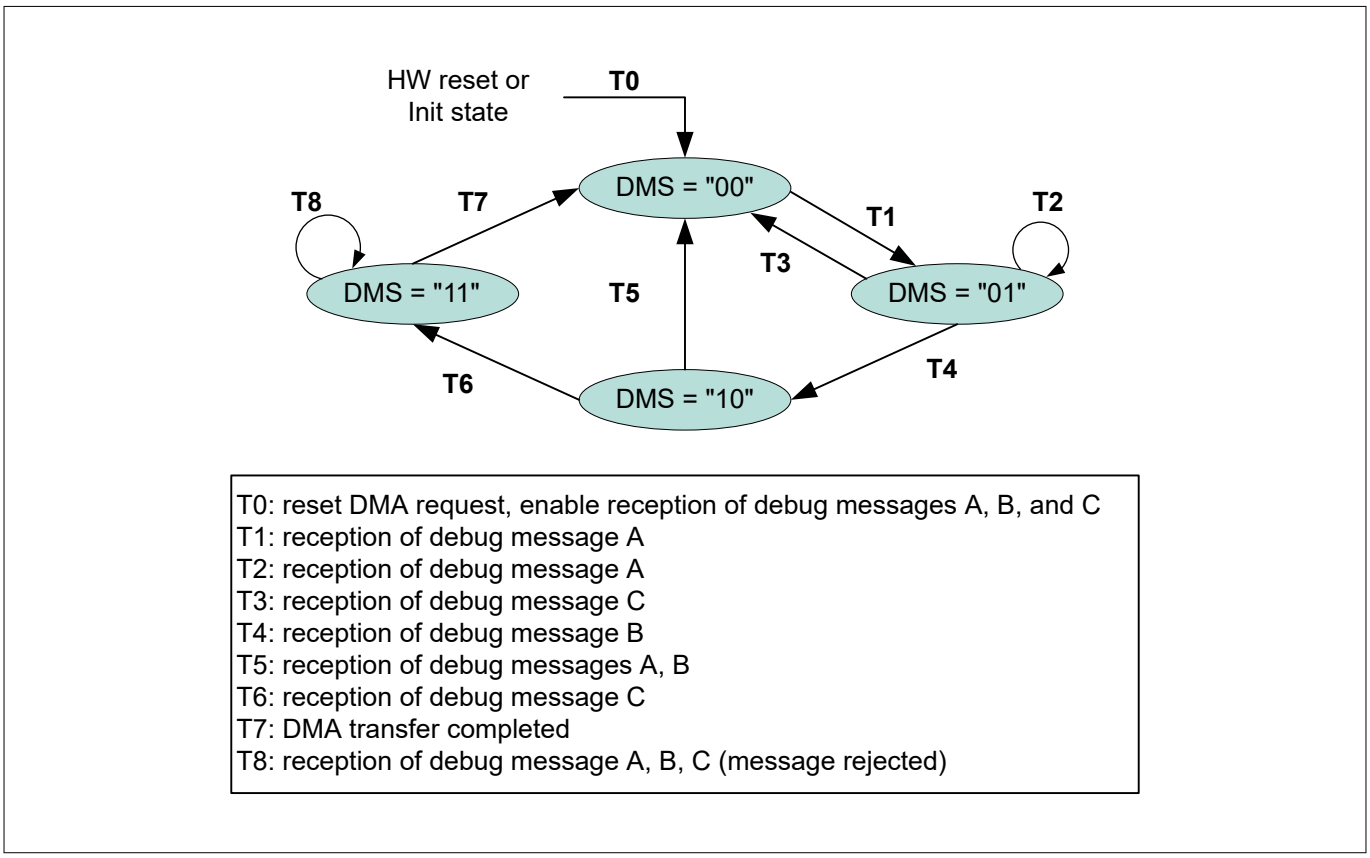


図 196　　デバッグ メッセージ処理ステートマシン

25.3.5　送信処理

TX ハンドラは、専用 TX バッファ、TX FIFO、TX キューに対する送信要求を処理します。CAN コアへの送信メッセージの転送、Put および Get Indices、TX Event FIFO を制御します。メッセージ送信用に最大 32 の TX バッファを設定できます。送信用の CAN モード (クラシック CAN または CAN FD) は、TX バッファ エレメントごとに個別に設定されます。TX バッファ エレメントは TX バッファ エレメントに記述されます。表 189 に、フレーム伝送に可能なコンフィギュレーションを示します。

表 189　　フレーム送信に可能な設定

CCR		TX バッファ エレメント		フレーム送信
BRSE	FDOE	FDF	BRS	
影響なし	0	影響なし	影響なし	クラシック CAN
0	1	0	影響なし	クラシック CAN
0	1	1	影響なし	ビットレート切り替えなしの FD
1	1	0	影響なし	クラシック CAN
1	1	1	0	ビットレート切り替えなしの FD
1	1	1	1	ビットレート切り替え付き FD

TX ハンドラは、TX バッファ要求保留レジスタ (TXBRP) が更新されたとき、または送信が開始されたときに、最も優先順位の高い保留中の TX 要求 (メッセージ ID が最も低い TX バッファ) をチェックするために TX スキャンを開始します。

25.3.5.1 送信ポーズ (一時停止)

送信ポーズ機能は、CAN メッセージ識別子が (恒久的に) 特定の値に割り当てられており、簡単に変更できない CAN システムでの使用を目的としています。これらのメッセージ識別子は、他の定義されたメッセージよりも高い CAN アービトレーション優先度を持つかもしれませんが、特定のアプリケーションでは、それらの相対的なアービトレーション優先度は逆である必要があります。このため、ある電子制御ユニット (ECU) がバースト CAN メッセージを送信すると、他の ECU の CAN メッセージの CAN アービトレーション優先順位が低くなるため、他の ECU の CAN メッセージが遅延する場合があります。

例えば、CAN ECU-1 が送信ポーズ機能を有効にしており、アプリケーションソフトウェアから 4 つのメッセージ送信を要求された場合、最初のメッセージ送信に成功すると、次の要求メッセージの送信を開始する前に、2 ノミナルビットのバスアイドル時間を待ちます。保留中のメッセージがある他の ECU がある場合、それらのメッセージはアイドル時間内に開始されるため、ECU-1 の次のメッセージとアービトレーションする必要はありません。メッセージを受信した ECU-1 は、受信したメッセージが CAN バスを解放すると、次の送信を開始できます。

送信ポーズ機能は、送信ポーズ ビット (CCCR.TXP) によって制御されます。ビットが設定されている場合、M_TTCAN コントローラーは、メッセージの送信に成功するたびに、次の送信を開始する前に、2 ノミナルビット時間一時停止します。これにより、ネットワーク内の他の CAN ノードは、たとえそのメッセージの先行識別子が低くても、メッセージを送信できます。デフォルトは送信ポーズ無効 (CCCR.TXP = 0) です。

この機能は、1 つのノードからのバースト送信を失い、アプリケーション プログラムが誤って多くの送信を要求してしまう「バブリング イディオット (Babbling Idiot)」シナリオから保護します。

25.3.5.2 専用 TX バッファ

専用 TX バッファは、CPU の完全な制御下でのメッセージ送信を目的としています。各専用 TX バッファは、特定のメッセージ ID で設定されます。複数の TX バッファが同じメッセージ ID で設定されている場合、バッファ番号の最も小さい TX バッファが送信されます。

データセクションが更新された場合、TXBAR.ARn を介して Add Request により送信が要求されます。要求されたメッセージは、内部的にはオプションの TX FIFO または TX キューからのメッセージと、外部的には CAN バス上のメッセージとアービトレーションし、メッセージ ID に従って送出されます。

25.3.5.2.1 専用 TX バッファのアドレス指定

専用 TX バッファは、表 190 に示すように、メッセージ RAM に 32 ビット ワードのエレメント サイズを割り当てます。したがって、メッセージ RAM 内の専用 TX バッファの開始アドレスは、(送信バッファ インデックス (0～31)×エレメント サイズ)+ TX バッファ開始アドレス (TXBC.TBSA[15:2]) です。

表 190 TX バッファ/FIFO/キュー エレメント サイズ

TXESC.TBDS [2:0]	データ フィールド [バイト]	エレメントサイズ [RAM ワード]
000	8	4
001	12	5
010	16	6

(続く)

表 190 (続き) TX バッファ/FIFO/キュー エレメント サイズ

TXESC.TBDS [2:0]	データ フィールド [バイト]	エレメントサイズ [RAM ワード]
011	20	7
100	24	8
101	32	10
110	48	14
111	64	18

25.3.5.3 送信 FIFO

TX FIFO 動作は TXBC.TFQM を'0'にプログラムすることで設定されます。TX FIFO に格納されたメッセージは、Get Index TXFQS.TFGI で参照されるメッセージから送信されます。各送信後、TX FIFO が空になるまで、Get Index は周期的にインクリメントされます。TXFIFO は、異なる TX バッファから同じメッセージ ID を持つメッセージを、これらのメッセージが TX FIFO に書き込まれた順序で送信することを可能にします。M_TTCAN は、TX FIFO 空きレベル TXFQS.TFFL を Get Index と Put Index の差として計算します。利用可能な (フリーの) TX FIFO エレメントの数を示します。

新しい送信メッセージは、Put Index TXFQS.TFQPI によって参照される TX バッファから始まる TX FIFO に書き込まれなければいけません。Add Request は、Put Index を次の空き TX FIFO エレメントまでインクリメントします。Put Index が Get Index に達すると、TX FIFO Full (TXFQS.TFQF = 1) がシグナルされます。この場合、次のメッセージが送信され、Get Index がインクリメントされるまで、それ以上のメッセージを TX FIFO に書き込んではいけません。

1 つのメッセージが TX FIFO に追加されると、TX FIFO の Put Index によって参照される TX バッファに関連する TXBAR ビットに'1'を書き込むことによって送信が要求されます。

複数の (n) メッセージが TX FIFO に追加されると、それらは Put Index から始まる連続した'n'個の TX バッファに書き込まれます。その後、TXBAR を介して送信を要求します。その後、Put Index は周期的に n だけインクリメントされます。要求された TX バッファの数は、TX FIFO 空きレベルによって示される TX バッファの空き数を超えてはいけません。

Get Index によって参照される TX バッファに対する送信要求がキャンセルされると、Get Index は保留中の送信要求を持つ次の TX バッファまでインクリメントされ、TX FIFO 空きレベルが再計算されます。他の TX バッファに送信キャンセルが適用されても、Get Index と FIFO Free Level は変更されません。

TX FIFO エレメントは、表 190 に示すように、メッセージ RAM にエレメント サイズの 32 ビット ワードを割り当てます。したがって、次に使用可能な (フリーの)TX FIFO バッファの開始アドレスは、TX FIFO/キュー Put Index TXFQS.TFQPI (0～31)×エレメント サイズを TX バッファ開始アドレス TXBC.TBSA に加算することによって計算されます。

25.3.5.4 TX キュー

TX キューの動作は、TXBC.TFQM を'1'に設定することにより設定されます。TX キューに格納されたメッセージは、メッセージ ID の小さい (優先順位の高い) メッセージから送信されます。複数のキュー バッファが同じメッセージ ID で設定されている場合、バッファ番号の最も小さいキュー バッファが最初に送信されます。

新しいメッセージは、Put Index TXFQS.TFQPI によって参照される TX バッファに書き込まれなければいけません。Add Request は、周期的に、次の空き TX バッファへの Put Index をインクリメントします。TX キューがフル (TXFQS.TFQF = 1) の場合、Put Index は、要求されたメッセージの 1 つが送信されるか、保留中の送信要求がキャンセルされるまで、それ以上のメッセージを書き込むことはできません。

アプリケーションは、Put Index の代わりに TXBRP レジスタを使用し、送信要求が保留されていない任意の TX バッファにメッセージを置けます。

TX キュー バッファは、表 190 に示すように、メッセージ RAM に 32 ビット ワードのエレメント サイズを割り当てます。したがって、次に使用可能な (フリーの) TX キューバッファの開始アドレスは、TX バッファ開始アドレス TXBC.TBSA に、TX FIFO/キュー Put Index TXFQS.TFQPI (0...31) × エレメント サイズを加算することによって計算されます。

25.3.5.5 混合専用 TX バッファ/TX FIFO

この場合、MRAM 内の TX バッファ セクションは、専用の TX バッファ セットと TX FIFO に細分化されます。専用 TX バッファの数は TXBC.NDTB によって設定されます。TX FIFO に割り当てられる TX バッファの数は TXBC.TFQS によって設定されます。TXBC.TFQS を'0'にプログラムすると、専用 TX バッファのみが使用されます。

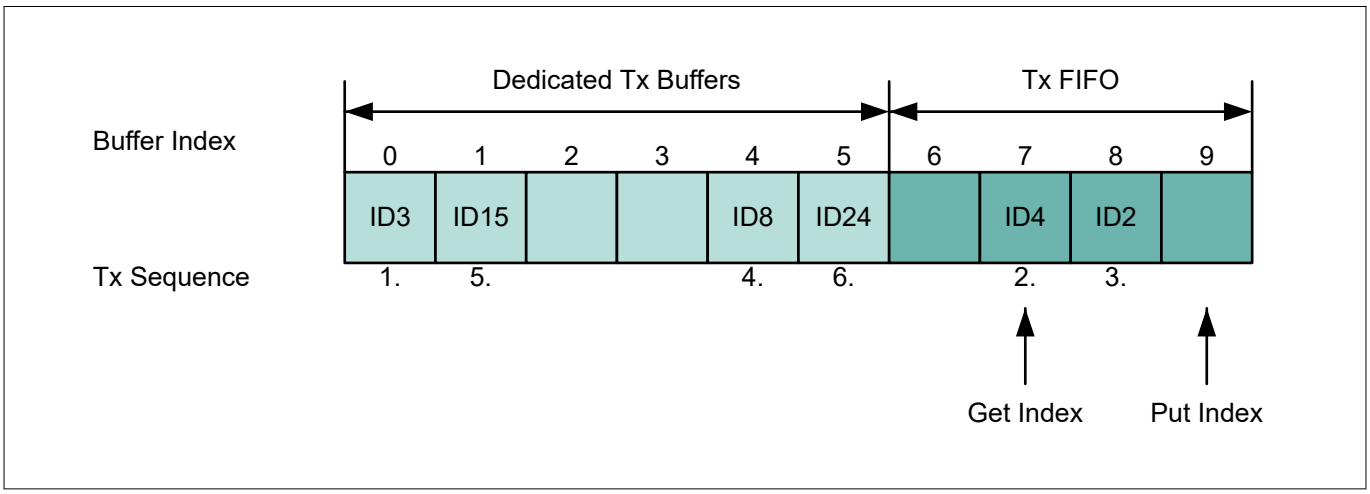


図 197 混合専用 TX バッファ/TX FIFO 設定例

TX の優先順位付け:

- 専用の TX バッファと最も古い保留中の TX FIFO バッファをスキャンする (TXFS.TFGI によって参照される)
- 専用の TX バッファと最も古い保留中の TX FIFO バッファをスキャンする (TXFS.TFGI によって参照される)

25.3.5.6 混合専用 TX バッファ/TX キュー

この場合、MRAM の TX バッファ セクションは、専用の TX バッファ セットと TX キューに細分化されます。専用 TX バッファの数は TXBC.NDTB によって設定されます。TX キューバッファの数は TXBC.TFQS によって設定されます。TXBC.TFQS が'0'にプログラムされている場合、専用 TX バッファのみが使用されます。

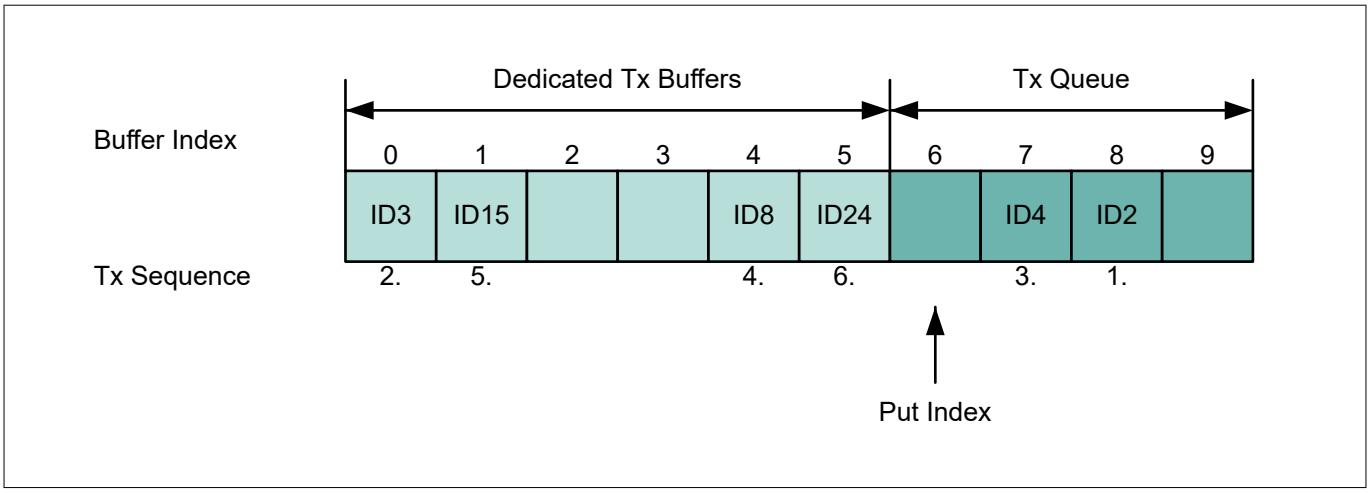


図 198　　混合専用 TX バッファ/TX キュー設定例

TX の優先順位付け:

- アクティブ化された送信要求ですべての TX バッファをスキャンします。
- メッセージ ID が最も小さい TX バッファが最も優先され、次に送信されます。

25.3.5.7　　送信キャンセル

M_TTCAN は送信キャンセルをサポートします。この機能は、特にゲートウェイ アプリケーションや AUTOSAR ベースのアプリケーションを対象としています。専用 TX バッファまたは TX キュー バッファからの要求された送信をキャンセルするためには、ホストは TXBCR レジスタの対応するビット位置 (TX バッファ数) に'1'を書き込まなければいけません。送信キャンセルは TX FIFO 動作を意図していません。

キャンセルの成功は、TXBCF レジスタの対応するビットを'1'に設定することで通知されます。

TX バッファからの送信中に送信キャンセルが要求された場合、送信中である限り、対応する TXBRP ビットはセットされたままです。送信に成功した場合、対応する TXBTO ビットと TXBCF ビットが設定されます。それ以外の場合は繰り返されず、対応する TXBCF ビットのみがセットされます。

注:　　保留中の送信がこの送信が開始される直前にキャンセルされた場合、このノードで別のメッセージが保留中であっても、送信が開始されない短いタイム ウィンドウが続きます。これによって、他のノードがメッセージを送信できるようになる可能性があります。

25.3.5.8　　送信イベントの処理

TX イベント処理をサポートするために、M_TTCAN は TX イベント FIFO を実装しています。M_TTCAN が CAN バス上でメッセージを送信した後、メッセージ ID とタイムスタンプが TX イベント FIFO エレメントに格納されます。TX イベントを TX イベント FIFO エレメントにリンクするためには、送信された TX バッファのメッセージマーカーを TX イベント FIFO エレメントにコピーします。

TX イベント FIFO は最大 32 エレメントまで設定できます。TX イベント FIFO エレメントについては、TX イベント FIFO エレメントで説明します。

TX イベント FIFO の目的は、送信ステータス情報の処理を送信メッセージの処理から切り離すことです。つまり、TX バッファは送信されるメッセージのみを保持し、送信ステータスは TX イベント FIFO に別途格納されます。これは、特に動的に管理される送信キューを運用する場合、送信に成功した直後に新しいメッセージのために TX バッファを使用できるという利点があります。TX バッファを上書きする前に、TX バッファから送信ステータス情報を保存する必要はありません。

TX イベント FIFO フル状態が IR.TEFF によって通知されると、1 つのエレメントが読み出され、TX イベント FIFO Get Index がインクリメントされるまで、TX イベント FIFO にそれ以上エレメントが書き込まれることはありません。TX イベント FIFO がフルの時に TX イベントが発生した場合、そのイベントは破棄され、割込みフラグ IR.TEFL がセットされます。

TX イベント FIFO のオーバーフローを回避するためには、TX イベント FIFO の透かしを使用します。TX イベント FIFO フルレベルが TXEFC.EFWM で設定された TX イベント FIFO 透かしに達すると、割込みフラグ IR.TEFW が設定されます。

TX イベント FIFO から読み出す場合、TX イベント FIFO の開始アドレス TXEFC.EFSA に TX イベント FIFO の Get Index TXEFS.EFGI を 2 回追加する必要があります。

25.3.6 FIFO アクノリッジ処理

RX FIFO 0、RX FIFO 1、および TX イベント FIFO の取得インデックスは、対応する FIFO Acknowledge Index によって制御されます。

RX FIFO トップポインタハードウェアロジックが使用される場合、RX FIFO Acknowledge Index を更新します。RXFTOPn_DATA が読み出された後、Acknowledge Index (RXFnA.FnA) が自動的に更新され、最終的に FIFO Get Index が FIFO Acknowledge Index に 1 を加えた値に設定され、FIFO フィル レベルが更新されます。

アプリケーションが RX FIFO トップ ポインタ ロジックを使用しない場合、Acknowledge Index の更新は以下の 2 つの使用例のいずれかを使用します。

- FIFO から (Get Index が指す) 1 つのエレメントのみが読み出される場合、Get Index 値は FIFO の Acknowledge Index に書き込まれます。
- FIFO から一連のエレメントが読み出されるとき、FIFO の Acknowledge Index は、その読み出しシーケンスの最後に 1 回のみ書き込めば十分です (値は、最後に読み出されたエレメントのインデックスです)。

CPU は M_TTCAN のメッセージ RAM に自由にアクセスできるため、2 つの RX FIFO のうちの 1 つから優先度の高いメッセージが送られることに注意してください。この場合、FIFO の Acknowledge Index は書き込まれるべきではありません。なぜなら、これによって Get Index が誤った位置に設定され、FIFO の充填レベルが変化して古い FIFO エレメントが失われるからです。

注: アプリケーションは、FIFO Acknowledge Index に有効な値が書き込まれていることを確認する必要があります。M_TTCAN は誤った値をチェックしません。

25.3.7 CAN ビットタイミングの設定

CAN ネットワーク内の各ノードは、独自のクロックジェネレーター (通常は水晶発振器) を持っています。ビット時間の時間パラメーターは、CAN ノードごとに個別に設定できます。各 CAN ノードの発振器の周期が異なっていても、共通のビットレートを生成できます。

発振器の周波数は、温度や電圧の変化、部品の劣化などによりわずかに変化します。周波数がオシレーターの許容範囲内でしか変化しない限り、CAN ノードはビットストリームに再同期することで、異なるビットレートを補正できます。

25.3.7.1 CAN ビットタイミング

CAN FD 動作は、ノミナルビット時間とデータビット時間の 2 つのビット時間が定義されます。ノミナルビット時間はアービトレーションフェーズのためのものです。データビット時間は同等かそれより短く、データフェーズの高速化に利用可能です (CAN FD 動作を参照)。

ビット時間の基本構成は、ノミナルビット時間とデータビット時間の両方で共有されます。ビット時間は、同期セグメント (Sync_Seg), 伝播時間セグメント (Prop_Seg), 位相バッファセグメント 1 (Phase_Seg1), 位相バッファセグメン

ト 2 (Phase_Seg2) の 4 つのセグメントに分割できます (図 199 を参照)。バスレベルが読み出され、対応するビットの値として解釈されるサンプルポイントは、Phase_Seg1 の直後です。

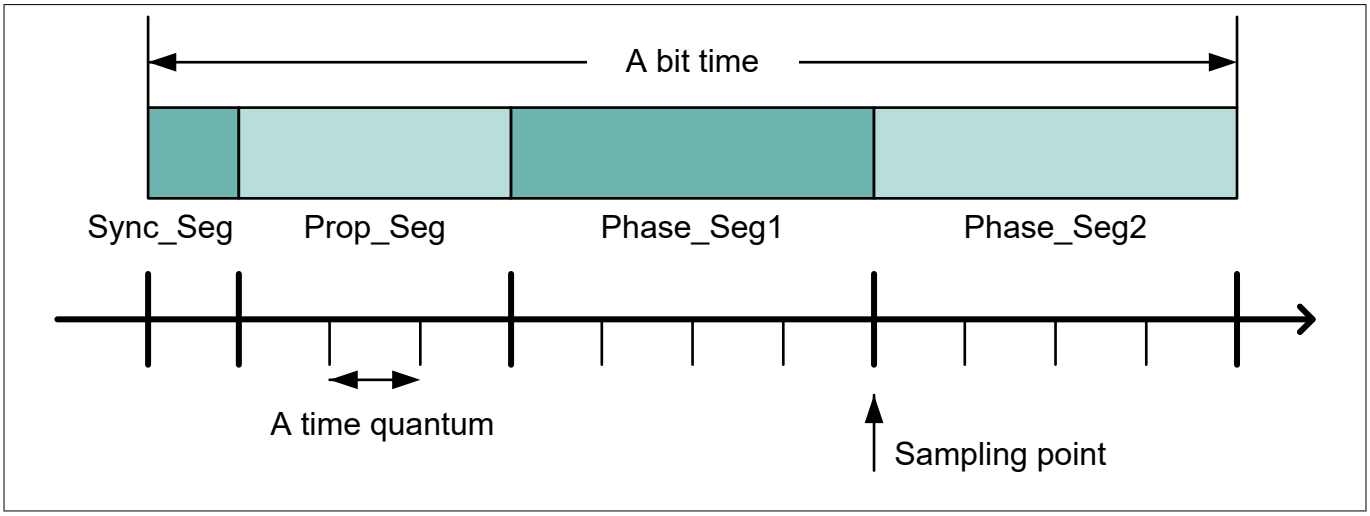


図 199　　ビット時間の構成

各セグメントは、clk_can とプリスケーラによって定義されるタイムクォンタム (Time Quanta: TQ) の倍数であるプログラム可能なタイムクォンタムで構成されています。これらのパラメーターを定義するために使用される値とプリスケーラは、ノミナルビット時間とデータビット時間で異なり、ノミナルビットタイミング&プリスケーラレジスタ (NBTP) とデータビットタイミング&プリスケーラレジスタ (DBTP) によって設定されます (表 191 を参照)。

表 191　　ビット時間パラメーター

パラメーター	説明
時間量子 tq (ノミナル) と tqd (データ)	時間量子 (タイムクォンタム) は、基本単位時間量子 (clk_can 周期) にそれぞれのプリスケーラを乗算することで導出されます。時間量子は、CAN FD コントローラーによって次のように設定されます。 ノミナル: tq = (NBTP.NBRP[8:0] +1) × clk_can 期間 データ: tqd = (DBTP.DBRP[4:0] + 1) × clk_can 期間
Sync_Seg	Sync_Seg は CAN 仕様によって 1 タイムクォンタムに固定されているため設定できません (CAN FD コントローラーに内蔵)。 ノミナル: 1 tq データ: 1 tqd
Prop_Seg	Prop_Seg は、ネットワーク内の物理的な遅延時間を保証するために使用されるビット時間の一部です。CAN FD コントローラーは Prop_Seg と Phase_Seg1 の合計を単一のパラメーターで設定します。 ノミナル: Prop_Seg + Phase_Seg1 = NBTP.NTSEG1[7:0] + 1 データ: Prop_Seg + Phase_Seg1 = DBTP.DTSEG1[4:0] + 1

(続く)

表 191　　(続き) ビット時間パラメーター

パラメーター	説明
Phase_Seg1	Phase_Seg1 はサンプリングポイント前のエッジ位相誤差を保証するために使用されます。再同期ジャンプ幅 (Resynchronization jump width) によって短縮できます。 Prop_Seg と Phase_Seg1 の合計は、CAN FD コントローラーによってノミナルとして設定されます。 ノミナル: NBTP.NTSEG1[7:0] + 1 データ: DBTP.DTSEG1[4:0] + 1
Phase_Seg2	Phase_Seg2 はサンプリングポイント後のエッジ位相誤差を保証するために使用されます。再同期ジャンプ幅 (Resynchronization jump width) によって短縮できます。 Phase_Seg2 は CAN FD コントローラーによってノミナルとして設定されます。 ノミナル: NBTP.NTSEG2[6:0] + 1 データ: DBTP.DTSEG2[3:0] + 1
SJW	再同期ジャンプ幅。Phase_Seg1 と Phase_Seg2 の長さを調整するために使用します。SJW は Phase_Seg1 または Phase_Seg2 より長くなりません。SJW は CAN FD コントローラーによってノミナルとして設定されます。 ノミナル: NBTP.NSJW[6:0] + 1 データ: DBTP.DSJW[3:0] + 1

ノミナルおよびデータビット時間は以下の式によって表せます。

ノミナルビット時間

= [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] × tq

= [1 + (NBTP.NTSEG1[7:0] + 1) + (NBTP.NTSEG2[6:0] + 1)×[(NBTP.NBRP[8:0]+1)×clk_can 期間]

データビット時間

= [1 + (DBTP.DTSEG1[4:0] + 1) + (DBTP.DTSEG2[3:0] + 1)] × [(DBTP.DBRP[4:0] +1) × clk_can 期間]

***注**:　　CAN FD コントローラーの情報処理時間 (IPT) は'0' で、サンプルポイントの後の最初の CAN クロックエッジで次のビットのデータが利用可能であることを意味します。したがって、Phase_Seg1 と IPT の最大値である Phase_Seg2 を設定する際に、IPT を考慮する必要はありません。*

25.3.7.2　CAN ビット レート

ビットレートはビット時間の逆数です。したがって、ノミナルビットレートは下記です。

1 / [{1 + (NBTP.NTSEG1[7:0] + 1) + (NBTP.NTSEG2[6:0] + 1)} × {(NBTP.NBRP[8:0] +1) × clk_can period}]

データ ビットレートは下記です。

1 / [{1 + (DBTP.DTSEG1[4:0] + 1) + (DBTP.DTSEG2[3:0] + 1)} × {(DBTP.DBRP[4:0] +1) × clk_can period}]

これらの式から、CAN FD コントローラーのビットレートは、CAN クロック (clk_can) の周期と各パラメーターの設定可能範囲に依存することがわかります。以下の表は、CAN クロック周波数を変化させた場合の設定可能なビットレートの例です。空のボックスは、指定された入力 CAN クロック周波数で希望のビットレートが設定されていないことを示します。

CAN clock frequency	8MHz		10MHz		16MHz		20MHz		32MHz		40MHz	
configuration / nominal bit rate	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1	Number of tqs per bit time	NBTP.NBRP + 1
125Kbps	64tq 32tq 16tq 8tq	1 2 4 8	80tq 40tq 20tq 10tq	1 2 4 8	128tq 64tq 32tq 16tq 8tq	1 2 4 8 16	160tq 80tq 40tq 20tq 10tq	1 2 4 8 16	256tq 128tq 64tq 32tq 16tq 8tq	1 2 4 8 16 32	320tq 160tq 80tq 40tq 20tq 10tq	1 2 4 8 16 32
250Kbps	32tq 16tq 8tq	1 2 4	40tq 20tq 10tq	1 2 4	64tq 32tq 16tq 8tq	1 2 4 8	80tq 40tq 20tq 10tq	1 2 4 8	128tq 64tq 32tq 16tq 8tq	1 2 4 8 16	160tq 80tq 40tq 20tq 10tq	1 2 4 8 16
500Kbps	16tq 8tq	1 2	20tq 10tq	1 2	32tq 16tq 8tq	1 2 4	40tq 20tq 10tq	1 2 4	64tq 32tq 16tq 8tq	1 2 4 8	80tq 40tq 20tq 10tq	1 2 4 8
1Mbps	8tq	1	10tq	1	16tq 8tq	1 2	20tq 10tq	1 2	32tq 16tq 8tq	1 2 4	40tq 20tq 10tq	1 2 4

図 200　　ノミナルビットレートの設定例

CAN clock frequency	8MHz		10MHz		16MHz		20MHz		32MHz		40MHz	
configuration / data bit rate	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1	Number of tqds per bit time	DBTP.DBRP + 1
500Kbps	16tqd 8tqd	1 2	20tqd 10tqd	1 2	32tqd 16tqd 8tqd	1 2 4	40tqd 20tqd 10tqd	1 2 4	32tqd 16tqd 8tqd	2 4 8	40tqd 20tqd 10tqd	2 4 8
1Mbps	8tqd	1	10tqd	1	16tqd 8tqd	1 2	20tqd 10tqd	1 2	32tqd 16tqd 8tqd	1 2 4	40tqd 20tqd 10tqd	1 2 4
2Mbps	-	-	-	-	8tqd	1	10tqd	1	16tqd 8tqd	1 2	20tqd 10tqd	1 2
4Mbps	-	-	-	-	-	-	-	-	8tqd	1	10tqd	1
5Mbps	-	-	-	-	-	-	-	-	-	-	8tqd	1
8Mbps	-	-	-	-	-	-	-	-	-	-	5tqd	1

図 201　　データビット レートの設定例

***注**:　ユーザーは、CAN バス上で適切な通信を確保するために、対応する CAN 規格に準拠するように CAN ビットタイミングを設定する必要があります。*

25.4　メッセージ RAM (MRAM)

MRAM は、RX/TX メッセージとフィルター設定を保持します。

***注**:*
- *初期化されていないワードを読み出す際のビットエラーを防止し、MRAM 内の予期せぬフィルター エレメント構成を回避するため、CAN FD コントローラーを設定する前に MRAM を'0'にする必要があります。*
- *未使用の MRAM は一般目的には使用されません*

25.4.1　メッセージ RAM の設定

メッセージ RAM は 32 ビット幅です。CAN FD コントローラーは、MRAM に最大 4480 ワードを割り当てるように設定できます (使用できるワード数は、実際の MRAM のサイズによって制限されることに注意)。図 202 に列挙されている各セクションを設定する必要はなく、セクションの順序に関しても制約はありません。

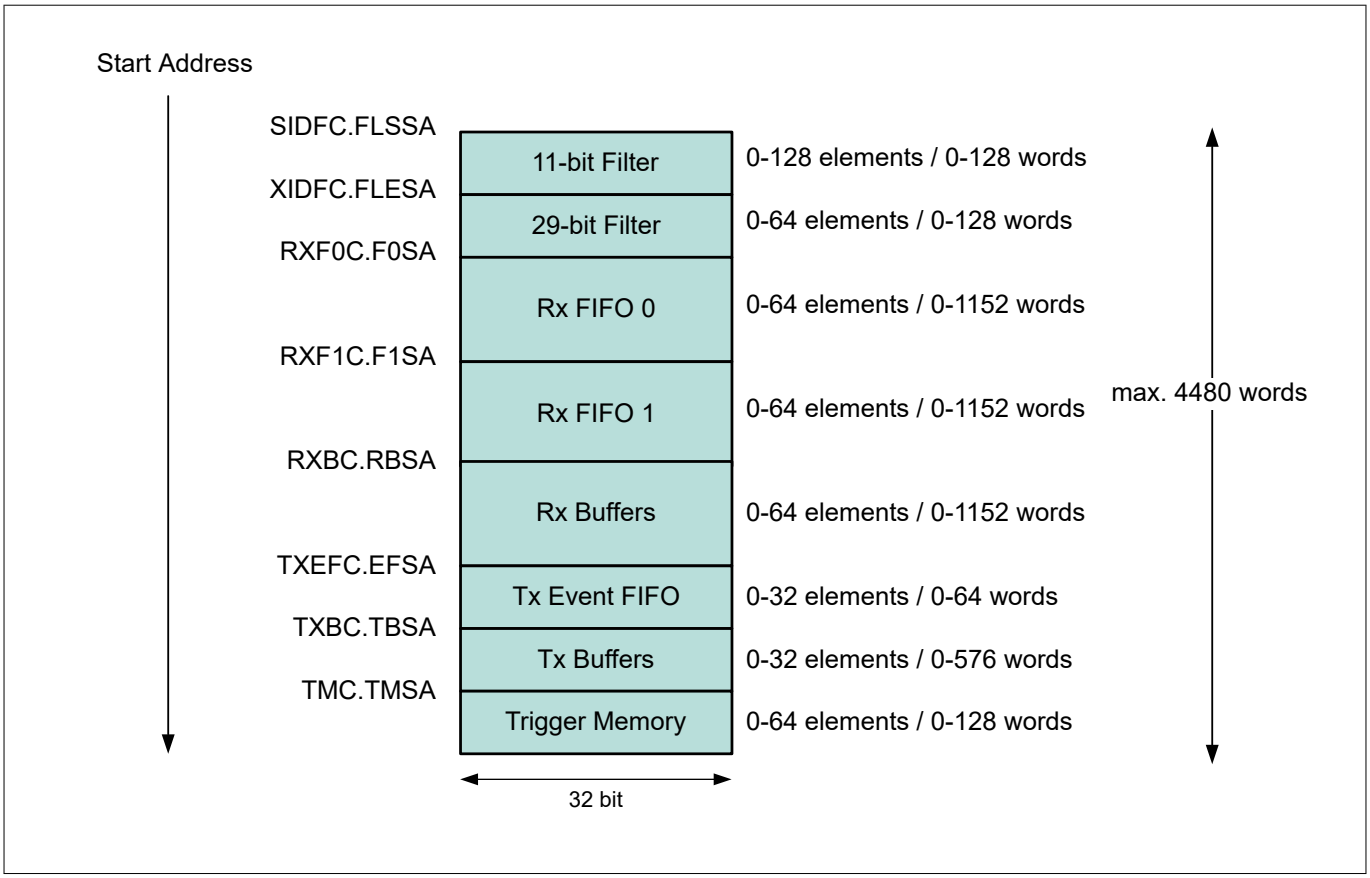


図 202　　メッセージ RAM の設定

CAN FD コントローラーは、メッセージ RAM をシングル バイトではなく、32 ビット ワードでアドレス指定します。設定可能なスタート アドレスは 32 ビットのワード アドレスで、ビット 15 からビット 2 のみが評価され、最下位 2 ビットは無視されます。

注:

- *CAN FD コントローラーは、メッセージ RAM の誤った設定をチェックしません。異なるセクションの開始アドレスと各セクションのエレメント数の設定は、データの改ざんや損失を避けるために慎重に行う必要があります。*
- *MRAM は M_TTCAN と CPU の両方からアクセスできます。動的ラウンドロビン方式でアクセスを割り当てます*

25.4.2　　RX バッファと FIFO エレメント

RX バッファと FIFO エレメントは 32 ビット ワードのブロックで、メッセージ RAM に格納された受信フレームのデータとステータスを保持します。

メッセージ RAM には、最大 64 の RX バッファと 2 つの RX FIFO が設定されます。各 RX FIFO セクションは、最大 64 の受信メッセージを格納できるように設定されます。図 203 に、RX バッファと FIFO エレメントの構造を示します。エレメント サイズは、レジスタ RXESC (RX バッファ/FIFO エレメント サイズ設定) を介して、最大 64 バイトのデータ フィールドを持つ CAN FD メッセージを格納できるように設定されます。

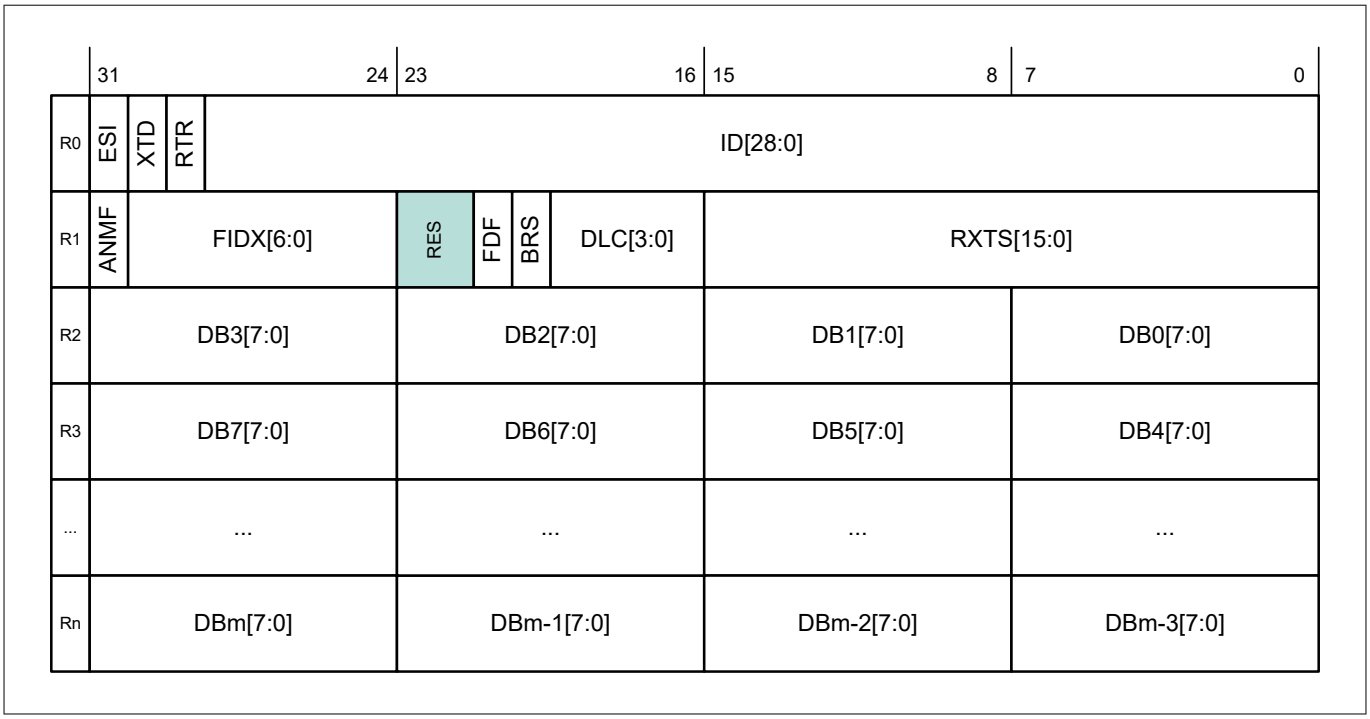


図 203 RX バッファと FIFO

表 192 R0 [bit31] ESI: エラー状態インジケータ

ビット	説明
0	送信ノードがエラーアクティブ
1	送信ノードはエラーパッシブ

受信したフレームが標準識別子か拡張識別子かを CPU に通知します。

表 193 R0 [bit30] XTD: 拡張識別子

ビット	説明
0	11 ビット標準識別子
1	29 ビット拡張識別子

受信したフレームがデータフレームかリモートフレームかを CPU に知らせます。

表 194 R0 [bit29] RTR: リモート送信要求

ビット	説明
0	受信フレームはデータフレーム
1	受信フレームはリモートフレーム

***注**: CAN FD フォーマットにはリモートフレームはありません。CAN FD フレーム (FDF = 1) では、ドミナントリモートリクエスト置換 (RRS) ビットがリモート送信リクエスト (RTR) ビットに置き換わります。*

R0 [bit28:0] ID[28:0]: 識別子
ビット XTD に応じた標準または拡張識別子。ID[28:18]には標準識別子が格納されます。

表 195 R1 [bit31] ANMF: 不一致フレームの受け入れ

ビット	説明
0	受信フレーム マッチング フィルター インデックス FIDX
1	受信フレームがどの RX フィルター エレメントにも一致しませんでした

不一致フレームの受け入れは、GFC.ANFS[1:0] (標準不一致フレームの受け入れ) および GFC.ANFE[1:0] (拡張不一致フレームの受け入れ) によって有効にできます。

表 196 R1 [bit30:24] FIDX[6:0]: フィルター インデックス

FIDX[6:0]	説明
0～127	一致する RX 受入フィルター エレメントのインデックス (ANMF = 1 の場合は無効)。範囲は 0～リストサイズ標準/拡張から 1 を引いた値 (SIDFC.LSS - 1 resp. XIDFC.LSE - 1)

R1 [bit23:22] 予約済み: 予約済みビット
書き込み時は、常に'0'を書きこんでください。読み出し値は未定義です。

表 197 R1 [bit21] FDF: 拡張データ長

ビット	説明
0	クラシック CAN フレームフォーマット
1	CAN FD フレームフォーマット

表 198 R1 [bit20] BRS: ビット レート スイッチ

ビット	説明
0	ビットレート切り替えなしで受信したフレーム
1	ビットレート切り替えで受信したフレーム

表 199 R1 [bit19:16] DLC[3:0]: データ長コード

DLC[3:0]	説明
0～8	クラシック CAN ＋ CAN FD: 受信フレームは 0～8 データバイト
9～15	クラシック CAN: 受信フレームは 8 データバイト。 CAN FD: 受信フレームのデータバイト数は 12/16/20/24/32/48/64。詳細については表 185 を参照してください。

R1 [bit15:0] RXTS[15:0]: RX タイムスタンプ
フレーム受信開始時にキャプチャされたタイムスタンプカウンター値。共有タイムスタンプ カウンター プリスケーラ TS_CTL.PRESCALE[15:0]の設定による分解能。

レジスタ名	データ ビット	データ バイト
R2 [bit31:24]	DB3[7:0] :	データバイト 3
R2 [bit23:16]	DB2[7:0] :	データバイト 2

レジスタ名	データ ビット	データ バイト
R2 [bit15:8]	DB1[7:0] :	データバイト 1
R2 [bit7:0]	DB0[7:0] :	データバイト 0
R3 [bit31:24]	DB7[7:0] :	データバイト 3
R3 [bit23:16]	DB6[7:0] :	データバイト 2
R3 [bit15:8]	DB5[7:0] :	データバイト 1
R3 [bit7:0]	DB4[7:0] :	データバイト 0
- - -	- - -	- - -
Rn [bit31:24]	DBm[7:0] :	データ バイト m
Rn [bit23:16]	DBm-1[7:0] :	データバイト m-1
Rn [bit15:8]	DBm-2[7:0] :	データバイト m-2
Rn [bit7:0]	DBm-3[7:0] :	データバイト m-3

注:
- *エレメントサイズの設定 (RX バッファ/FIFO エレメントサイズ コンフィギュレーション (RXESC) で定義) に応じて、Rn は n = 3 から 17 まで変化します。*
- *m は n の関数で、m = (n - 1) × 4 - 1 です。*
- *有効なデータ バイト数は、データ長コードによって定義されます。*

25.4.3 TX バッファ エレメント

TX バッファエレメントは、メッセージ RAM に格納された 32 ビットワードのブロックで、CAN FD コントローラーが送信するフレームのデータと制御情報を保持します。

TX バッファ部は、専用の TX バッファと TX FIFO/TX キューを保持するように設定されます。TX バッファセクションが専用 TX バッファと TX FIFO/TX キューで共有されている場合、専用 TX バッファは TX バッファセクションの先頭から始まり、TX FIFO または TX キューに割り当てられているバッファがそれに続きます。TX ハンドラは、TX バッファ コンフィギュレーション TXBC.TFQS[5:0] (送信 FIFO/キュー サイズ) と TXBC.NDTB[5:0] (専用送信バッファ数) を評価することで、専用 TX バッファと TX FIFO/TX キューを区別します。TX バッファ エレメント サイズ設定 (TXESC) レジスタにより、最大 64 バイトのデータ フィールドを持つ CAN FD メッセージを格納するためのエレメント サイズが設定されます。

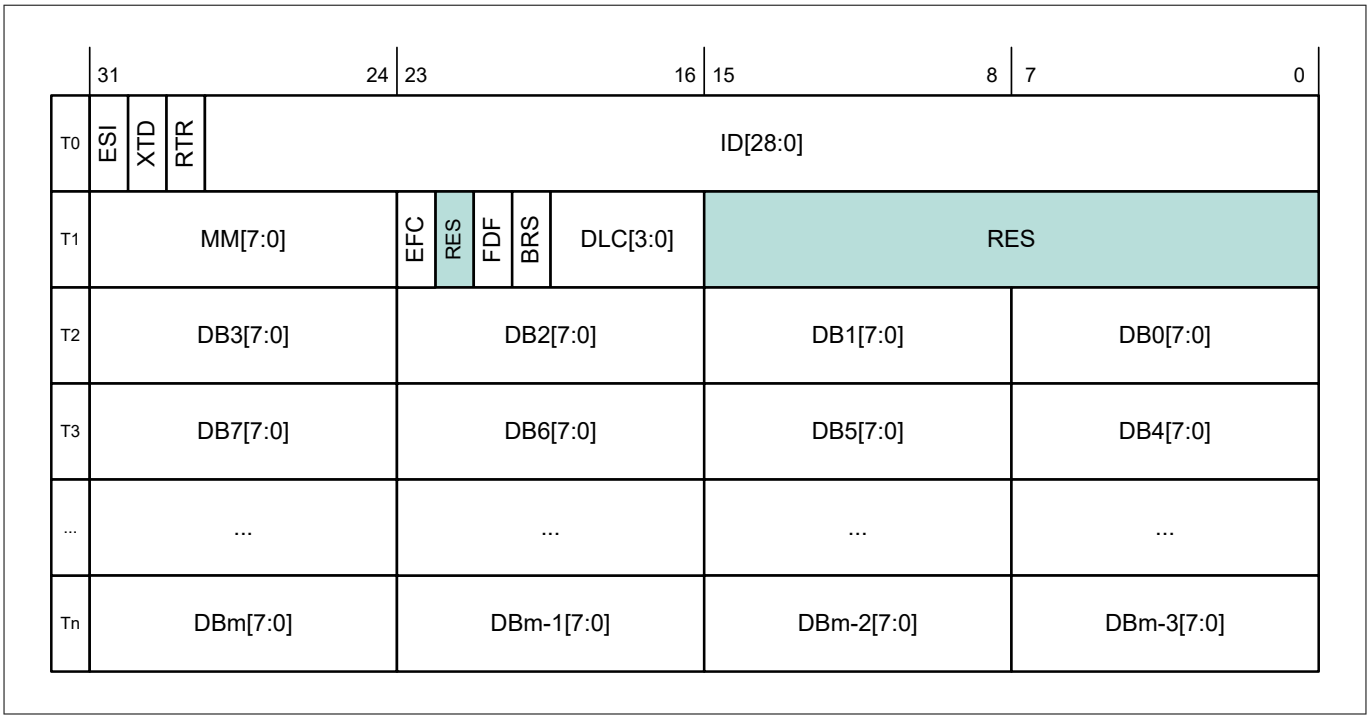


図 204　　TX バッファ エレメント

表 200　　R0 [bit31] ESI: Error State Indicator (エラーステートインジケーター)

ビット	説明
0	CAN FD フォーマットの ESI ビットはエラーパッシブフラグにのみ依存
1	CAN FD フォーマットの ESI ビットがリセッシブに送信

注: 送信バッファの ESI ビットはエラーパッシブフラグと OR され、送信 FD フレームの ESI ビットの値が決定されます。CAN FD プロトコル仕様で要求されているように、エラーアクティブノードはオプションで ESI ビットをリセッシブに送信できますが、エラーパッシブノードは常に ESI ビットをリセッシブに送信します。

表 201　　R0 [bit30] XTD: 拡張識別子

ビット	説明
0	11 ビット標準識別子
1	29 ビット拡張識別子

受信したフレームが標準識別子か拡張識別子かを CPU に通知します。

表 202　　R0 [bit29] RTR: リモート送信要求

ビット	説明
0	データフレームを送信
1	リモートフレームを送信

注: RTR = 1 の場合、FD オペレーションイネーブル (CCCR.FDOE) が CAN FD フォーマットでの送信を有効にしていても、CAN FD コントローラーは ISO 11898-1 に従ってリモートフレームを送信します。

T0 [bit28:0] ID[28:0]: 識別子

ビット XTD に応じた標準または拡張識別子。ID[28:18]には標準識別子を書き込まなければいけません。

T1 [bit31:24] MM[7:0]: メッセージマーカー

TX バッファ コンフィギュレーション中に CPU によって書き込まれます。TX メッセージ ステータスを識別するために、TX イベント FIFO エレメントにコピーされます。

表 203 T1 [bit23] EFC: イベント FIFO 制御

ビット	説明
0	TX イベントを保存しません
1	TX イベントを保存します

T1 [bit22] 予約済み: 予約済みビット

書き込み時は、常に'0'を書きこんでください。読み出し値は未定義です。

表 204 T1 [bit21] FDF: FD フォーマット

ビット	説明
0	クラシック CAN 形式で送信されたフレーム
1	CAN FD 形式で送信されたフレーム

表 205 T1 [bit20] BRS: ビットレート切り替え

ビット	説明
0	ビットレートを切り替えずに送信された CAN FD フレーム
1	ビットレートを切り替えて送信された CAN FD フレーム

注: ESI、FDF、BRS の各ビットは、CAN FD 動作が有効になっている CCCR.FDOE = 1 のときのみ評価されます。ビット BRS は、さらに CCCR.BRSE = 1 のときのみ評価されます。FDF ビットと BRS ビットの詳細については表 189 を参照してください。

表 206 T1 [bit19:16] DLC[3:0]: データ長コード

DLC[3:0]	説明
0～8	クラシック CAN ＋ CAN FD: 送信フレームは 0～8 データバイト
9～15	クラシック CAN: 送信フレームは 8 データバイト

CAN FD: 送信フレームは 12/16/20/24/32/48/64 データバイト。

T1 [bit15:0] 予約済み: 予約済みビット

書き込み時は、常に'0'を書きこんでください。読み出し値は未定義です。

T2 [bit31:24]	DB3[7:0] :	データバイト 3
T2 [bit23:16]	DB2[7:0] :	データバイト 2

T2 [bit15:8]	DB1[7:0] :	データバイト 1
T2 [bit7:0]	DB0[7:0] :	データバイト 0
T3 [bit31:24]	DB7[7:0] :	データバイト 3
T3 [bit23:16]	DB6[7:0] :	データバイト 2
T3 [bit15:8]	DB5[7:0] :	データバイト 1
T3 [bit7:0]	DB4[7:0] :	データバイト 0
- - -	- - -	- - -
Tn [bit31:24]	DBm[7:0] :	データ バイト m
Tn [bit23:16]	DBm-1[7:0] :	データバイト m-1
Tn [bit15:8]	DBm-2[7:0] :	データバイト m-2
Tn [bit7:0]	DBm-3[7:0] :	データバイト m-3

注:
- *エレメント サイズ (TXESC) の設定により、Tn は n = 3 から 17 まで変化します。*
- *m は n の関数: m = (n - 1) × 4 - 1*

25.4.4 TX イベント FIFO エレメント

各 TX イベント FIFO エレメントは、送信されたメッセージに関する情報を格納します。TX イベント FIFO の読み出しによって、CPU はメッセージが送信された順番にこの情報を得ます。TX イベント FIFO に関する状態情報はレジスタ TX イベント FIFO ステータス (TXEFS) から得られます。

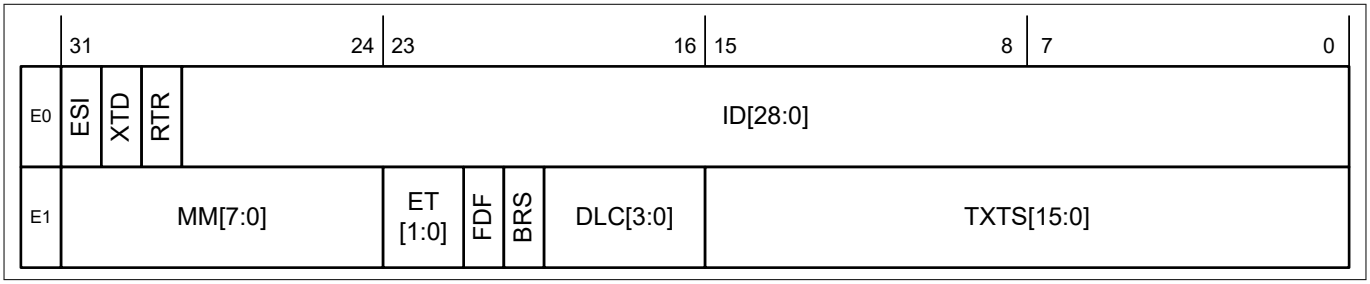


図 205 TX イベント FIFO エレメント

表 207 E0 [bit31] ESI: エラー状態インジケータ

ビット	説明
0	送信ノードがエラーアクティブ
1	送信ノードはエラーパッシブ

表 208 E0 [bit30] XTD: 拡張識別子

ビット	説明
0	11 ビット標準識別子
1	29 ビット拡張識別子

表 209 E0 [bit29] RTR: リモート送信要求

ビット	説明
0	データフレーム送信
1	リモートフレーム送信

E0 [bit28:0] ID[28:0]: 識別子

ビット XTD に応じた標準または拡張識別子。ID[28:18]には標準識別子が格納されます。

E1 [bit31:24] MM[7:0]: メッセージマーカー

TX メッセージ ステータスを識別するために、TX バッファから TX イベント FIFO エレメントにコピーされます。

表 210 E1 [bit23:22] ET[1:0]: イベントタイプ

ET[1:0]	説明
00	予約済み
01	TX イベント
10	キャンセルにもかかわらず送信 DAR モード (Disable Automatic Retransmission mode) の送信に常に設定されます。
11	予約済み

表 211 E1 [bit21] FDF: FD フォーマット

ビット	説明
0	クラシック CAN フレームフォーマット
1	CAN FD フレームフォーマット (新しい DLC コーディングと CRC)

表 212 E1 [bit20] BRS: ビットレート切り替え

ビット	説明
0	ビットレート切り替えなしで送信されたフレーム
1	ビットレートを切り替えて送信されたフレーム

表 213 E1 [bit19:16] DLC[3:0]: データ長コード

DLC[3:0]	説明
0～8	クラシック CAN + CAN FD: 0～8 データバイトのフレームを送信
9～15	クラシック CAN: 8 データバイトを送信するフレーム。 CAN FD: 12/16/20/24/32/48/64 データバイトが送信されたフレーム。 詳細については表 185 を参照してください。

E1 [bit15:0] TXTS[15:0]: TX タイムスタンプ

フレーム送信開始時にキャプチャされたタイムスタンプカウンター値。分解能は共有タイムスタンプ カウンター プリスケーラ TS_CTL.PRESCALE[15:0]の設定に依存します。

25.4.5 標準メッセージ ID フィルター エレメント

標準メッセージ ID フィルター エレメントは、単一の 32 ビット ワードで構成され、11 ビットの標準 ID を持つメッセージに対して、レンジ フィルター、デュアル フィルター、クラシック ビット マスク フィルター、または単一の専用 ID に対するフィルターとして設定できます。

最大 128 のフィルターエレメントを 11 ビットの標準 ID で設定できます。標準メッセージ ID フィルター エレメントにアクセスする場合、そのアドレスはフィルター リスト標準開始アドレス (SIDFC.FLSSA[15:2]) とフィルター エレメントのインデックス (0～127) になります。

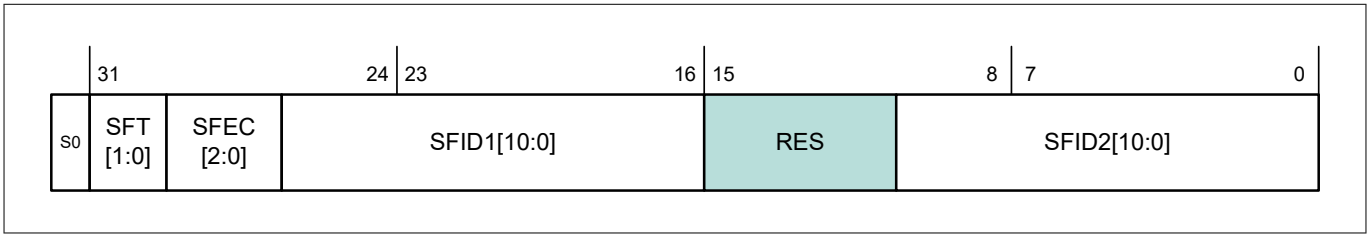


図 206 標準メッセージ ID フィルター

表 214 S0 [bit31:30] SFT[1:0]: 標準フィルタータイプ

SFT[1:0]	説明
00	SFID1[10:0]から SFID2[10:0]までのレンジフィルター (SFID2[10:0] ≥ received ID ≥ SFID1[10:0])
01	SFID1[10:0]または SFID2[10:0]のデュアル ID フィルター
10	クラシックフィルター: SFID1[10:0] = フィルター, SFID2[10:0] = マスク 対応する SFID2[10:0]ビットが 1 である SFID1[10:0]のビットのみが関連します
11	フィルターエレメント無効

注: SFT = 11 の場合、フィルターエレメントは無効になり、受入フィルタリングが継続されます (SFEC = 000 の場合と同じ動作)。

S0 [bit29:27] SFEC[2:0]: 標準フィルターエレメントの設定。

すべての有効なフィルターエレメントは、標準フレームの受入フィルタリングに使用されます。受入フィルタリングは、最初に一致する有効なフィルターエレメントか、フィルターリストの最後に達した時点で停止します。

SFEC[2:0] = 100、101、または 110 の場合、一致すると割込みフラグ IR.HPM (High Priority Message) がセットされ、有効になっていれば割込みが発生します。この場合、レジスタ HPMS (High Priority Message Status) は、プライオリティ マッチのステータスで更新されます。

表 215 SFEC[2:0]: 優先度を設定し、RX FIFO に格納します。

SFC[2:0]	説明
000	フィルターエレメントを無効化

(続く)

表 215　(続き) SFEC[2:0]: 優先度を設定し、RX FIFO に格納します。

SFC[2:0]	説明
001	フィルターが一致した場合、RX FIFO 0 に格納
100	フィルターが一致した場合、RX FIFO 1 に格納
011	フィルターが一致した場合、ID を却下
100	フィルターが一致した場合、優先順位を設定
101	優先順位を設定し、フィルターが一致した場合、RX FIFO 0 に格納
110	優先順位を設定し、フィルターが一致した場合、RX FIFO 1 に格納
111	専用 RX バッファに格納、またはデバッグメッセージとして格納、SFT[1:0]の設定は無視

S0 [bit26:16] SFID1[10:0]: 標準フィルター ID 1。

このビット フィールドには、SFEC[2:0]の設定によって異なる意味があります。

- SFEC[2:0] = 001～110

SFT[1:0]の設定に応じて SFID1[10:0]を設定

- SFEC[2:0] = 111

SFID1[10:0]は格納される標準専用 RX バッファまたはデバッグメッセージの ID を定義します。受信した識別子は一致しなければならず、マスキングメカニズムは使用されません。

S0 [bit15:11] 予約済み: 予約済みビット。

書き込み時は、常に'0'を書きこんでください。読み出し値は未定義です。

S0 [bit10:0] SFID2[10:0]: 標準フィルター ID 2。

このビット フィールドには、SFEC[2:0]の設定によって異なる意味があります。

- SFEC[2:0] = 001～11

SFT[1:0]の設定に応じて SFID2[10:0]を設定

- SFEC[2:0] = 111

専用 RX バッファまたはデバッグ メッセージ用フィルター SFID2[10:9]は受信メッセージを専用 RX バッファに格納するか、デバッグ メッセージ シーケンスのメッセージ A、B、C として扱うかを決定します。

表 216　SFID[10:9]]メッセージを専用の RX バッファに格納

SFID[10:9]	説明
00	メッセージを専用の RX バッファに格納
01	デバッグ メッセージ A
10	デバッグメッセージ B
11	デバッグ メッセージ C

SFID2[8:6]は予約ビットです。書き込み時は、常に'0'を書きこんでください。読み出し値は未定です。SFID2[5:0]は、一致するメッセージを格納する RX バッファ開始アドレス RXBC.RBSA[15:2]へのオフセットを定義します。

注:　デバッグメッセージは、CAN 機能のデバッグに使用されます。

25.4.6 拡張メッセージ ID フィルター エレメント

ビット ワードで、レンジ フィルター、デュアル フィルター、クラシックなビット マスク フィルター、または 29 ビットの拡張 ID を持つメッセージのための単一の専用 ID のフィルターとして設定できます。

最大 64 のフィルターエレメントを 29 バイト拡張 ID で設定できます。拡張メッセージ ID フィルター エレメントにアクセスする場合、そのアドレスはフィルター リスト拡張開始アドレス (XIDFC.FLESA[15:2]) + 2 × フィルター エレメントのインデックス (0～63) になります。

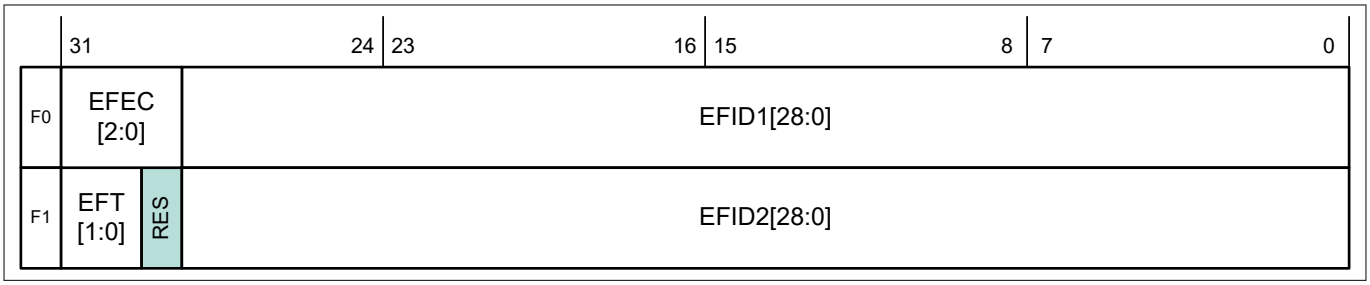


図 207 拡張メッセージ ID フィルター

F0 [bit31:29] EFEC[2:0]: 拡張フィルター エレメント設定。有効化されたフィルター エレメントはすべて、拡張フレームの受入フィルタリングに使用されます。受入フィルタリングは、最初に一致する有効なフィルターエレメントか、フィルターリストの最後に達した時点で停止します。

EFEC[2:0] = 100、101、または 110 の場合、一致すると割込みフラグ IR.HPM (High Priority Message) がセットされ、有効になっていれば割込みが発生します。この場合、高優先度メッセージ ステータス レジスタ (HPMS) は優先度一致のステータスで更新されます。

表 217 拡張フィルターエレメント

EFEC[2:0]	説明
000	フィルターエレメントを無効化
001	フィルターが一致した場合、RX FIFO 0 に格納
010	フィルターが一致した場合、RX FIFO 1 に格納
011	フィルターが一致した場合、ID を却下
100	フィルターが一致した場合、優先順位を設定
101	優先順位を設定し、フィルターが一致した場合、RX FIFO 0 に格納
110	優先順位を設定し、フィルターが一致した場合、RX FIFO 1 に格納
111	専用 RX バッファに格納、またはデバッグメッセージとして格納、EFT[1:0]の設定は無視

F0 [bit28:0] EFID1[28:0]: 拡張フィルター ID 1

このビット フィールドには、EFEC[2:0]の設定によって異なる意味があります。

- EFEC[2:0] = 001～110

EFT[1:0]の設定に従い EFID1[28:0]を設定します。

- EFEC[2:0] = 11

EFID1[28:0]は、格納される拡張専用 RX バッファまたはデバッグ メッセージの ID を定義します。受信した識別子は一致しなければならず、XIDAM のマスキング メカニズムのみが使用されます。

F1 [bit31:30] EFT[1:0]: 拡張フィルタータイプ

表 218　拡張フィルタータイプ

EFT[2:0]	説明
00	EFID1[28:0]から EFID2[28:0]までのレンジフィルター (EFID2[28:0]≧受信 ID と XIDAM≧EFID1[28:0]との AND)
01	デュアル ID フィルター EFID1[28:0]または EFID2[28:0]が XIDAM と AND された受信 ID と等しい場合に一致します
10	クラシックフィルター: EFID1[28:0] = フィルター, EFID2[28:0] = マスク。 対応する EFID2[28:0]ビットが 1 である EFID1[28:0]のこれらのビットのみが関連します。受信した ID を XIDAM と AND したものが、EFID2[28:0] でマスクされた EFID1[28:0] と等しい場合に一致します。
11	EFID1[28:0]から EFID2[28:0]までのレンジフィルター (EFID2[28:0]≧EFID1[28:0])、XIDAM マスク未適用

F1 [bit29] 予約済み: 予約済みビット
書き込み時は、常に‘0’を書きこんでください。読み出し値は未定義です。
F1 [bit28:0] EFID2[28:0]: 拡張フィルター ID 2
このビット フィールドには、EFEC[2:0]の設定によって異なる意味があります。

- EFEC[2:0] = 001～110

EFT[1:0]の設定に応じて EFID2[28:0]を設定

- EFEC[2:0] = 111

EFID2[28:0]は、専用 RX バッファまたはデバッグ メッセージ用にこのフィルターを設定するために使用します。
EFID2[28:11]は予約ビットです。書き込み時は、常に‘0’を書きこんでください。読み出し値は未定義です。
EFID2[10:9]は、受信したメッセージを専用の RX バッファに格納するか、デバッグ メッセージ シーケンスのメッセージ A、B、C として扱うかを決定します。

表 219　拡張フィルタータイプ

EFID2[10:9]	説明
00	メッセージを専用の RX バッファに格納
01	デバッグ メッセージ A
10	デバッグメッセージ B
11	デバッグ メッセージ C

EFID2[8:6]は予約ビットです。書き込み時は、常に‘0’を書きこんでください。読み出し値は未定です。
EFID2[5:0]は RX バッファ開始アドレスへのオフセットを定義します。
RXBC.RBSA[15:2]に一致するメッセージを格納します。

注: デバッグメッセージは、CAN 機能のデバッグに使用されます。

25.4.7 トリガー メモリ エレメント

最大 64 のトリガー メモリ エレメントを設定可能です。トリガー メモリ エレメントにアクセスする場合、そのアドレスはトリガー メモリ スタート アドレス (TTTMC) となります。TMSA にトリガーメモリ エレメントのインデックス (0～63) を加えたものとなります。

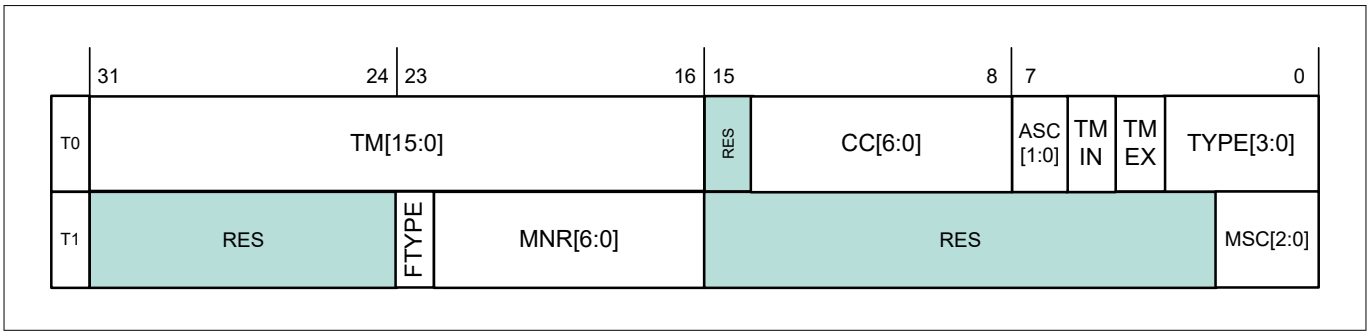


図 208 トリガー メモリ エレメント

T0 Bit 31:16 TM[15:0]: タイムマーク
トリガーがアクティブになるサイクル時間。

表 220 T0 ビット 14:8 CC[6:0]: サイクルコード

CC[6:0]	説明
000000x	すべてのサイクルで有効
000001c	サイクルカウント mod2= c で 2 サイクル目ごとに有効
00001cc	サイクルカウント mod4 =cc で 4 サイクルごとに有効
0001ccc	サイクルカウント mod8 =ccc で 8 サイクルごとに有効
001cccc	サイクルカウント mod16 =cccc で 16 サイクルごとに有効
01ccccc	サイクルカウント mod32 = ccccc で 32 サイクルごとに有効
1cccccc	サイクルカウント mod64 = cccccc で 64 サイクルごとに有効

トリガーが有効なサイクルカウント。トリガータイプ Tx_Ref_Trigger, Tx_Ref_Trigger_Gap, Watch_Trigger, Watch_Trigger_Gap, End_of_List では無視されます。

表 221 T0 ビット 7:6 ASC[1:0]: 非同期シリアル通信

ASC[1:0]	説明
00	ASC 動作なし
01	予約済み、使用不可
10	ノードは ASC 受信側
11	ノードは ASC 送信側

注: ASC 機能はサポートされていません。

表 222　　T0 ビット 5 TMIN: タイムマーク イベント インターナル

TMIN	説明
0	何もしない
1	TTIR.TTMI は、トリガー メモリ エレメントがアクティブになったときに設定されます

表 223　　T0 ビット 3:0 TYPE[3:0]: トリガータイプ

TYPE [3:0]	説明
0000	Tx_Ref_Trigger - ギャップがないときに有効
0001	Tx_Ref_Trigger_Gap - ギャップ時に有効
0010	Tx_Trigger_Single - 排他的なタイムウィンドウで単一の送信を開始します
0011	Tx_Trigger_Continous - 排他的タイムウィンドウで連続送信を開始します
0100	Tx_Trigger_Arbitration - アービトレーション タイムウィンドウで送信を開始します
0101	Tx_Trigger_Merged - マージされたアービトレーション ウィンドウを開始します
0110	Watch_Trigger - ギャップがないときに有効
0111	Watch_Trigger_Gap - ギャップがあるときに有効
1000	Rx_Trigger - 受信をチェックします
1001	Time_Base_Trigger - TMIN、TMEX、ASC のみを制御します
1010～1111	End_of_List - 不正な型、設定エラーの原因

注:
- *ASC 動作 (ASC = 10, 11) では、トリガータイプ Rx_Trigger と Time_Base_Trigger のみを使用します。*
- *ASC 操作はこのデバイスではサポートされていません。*

表 224　　T1 ビット 23 FTYPE: フィルタータイプ

FTYPE	説明
0	11 ビット標準メッセージ ID
1	29 ビット拡張メッセージ ID

T1 ビット 22:16 MNR[6:0]: メッセージ番号。

送信: トリガーは設定された TX バッファ番号に対して有効です。有効な値は 0～31 です。

受信: トリガーは、標準/拡張メッセージ ID フィルター エレメント番号に対して有効です。有効な値は 0 から 63 および 0 から 127 です。

T1 ビット 2:0 MSC[2:0]: メッセージ ステータス カウント。

排他的タイムウィンドウの定期メッセージのスケジューリング エラーをカウントします。メッセージのアービトレーションやイベント駆動型 CAN 通信 (ISO 11898-1:2015) の機能はありません。

注:
- *トリガー メモリ エレメントは、M_TTCAN が INIT 状態の時に書き込まれる必要があります。INIT 状態以外でのトリガー メモリ エレメントへの書き込みアクセスは禁止されています。*
- *TMIN と TMEX は、TYPETx_Ref_Trigger のトリガーメモリエレメントの一部として定義されている場合は例外です。この場合、実際のリファレンス トリガー オフセット (TTOST.RTO) によって変更されたタイム マークでアクティブになります。*

25.4.8 RAM ウォッチドッグ (RWD)

RAM ウォッチドッグは、MRAM の READY 出力を監視します。MRAM アクセスは、RWD.WDC で設定された値でメッセージ RAM ウォッチドッグ カウンターを開始します。MRAM の READY 出力がアクティブになり、MRAM が正常に完了したことを示すと、カウンターは RWD.WDC で再ロードされます。カウンターがゼロになるまで MRAM からの応答がない場合、カウンターは停止し、割込みフラグ IR.WDI がセットされます。RWD カウンターはホストクロック (clk_sys) でクロックされます。RWD レジスタの詳細については、レジスタリファレンスマニュアルを参照してください。

25.5 タイマートリガー CAN

25.5.1 参照メッセージ

参照メッセージは、特定の CAN 識別子によって特長付けられるデータフレームです。このメッセージは、タイムマスター (参照メッセージの送信者) を除くすべてのノードによって受信され、受け入れられます。

レベル 1 の場合、データ長は少なくとも 1 でなければいけません。レベル 0 とレベル 2 の場合、データ長は少なくとも 4 でなければいけません。参照メッセージは、8CAN データ バイトの合計まで、他のデータによって拡張されます。3 つの LSB を除く識別子のすべてのビットが、そのメッセージを参照メッセージとして特長付けます。最後の 3 ビットは、最大 8 つのタイム マスター候補の優先順位を指定します。予約ビットは論理 0 として送信され、受信側では無視されます。参照メッセージは TTRMC レジスタを使って設定します。

タイム マスターは参照メッセージを送信します。参照メッセージがエラーによって妨害された場合、それは直ちに再送されます。再送では、送信された Master_Ref_Mark が更新されます。参照メッセージは定期的に送信されますが、定期的な送信を停止することもできます (Next_is_Gap ビット)。これは、現在のタイム マスターまたは他のタイム マスター候補の 1 つによって、次の基本サイクルの開始時にイベント同期送信を開始できます。

参照メッセージを送信するノードは現在のタイムマスターです。タイム マスターは他のメッセージの送信を許可されます。現在のタイムマスターが故障した場合、その機能は最も優先順位の高いポテンシャル タイムマスターによって複製されます。タイム マスターでもポテンシャル タイムマスターでもないノードは、タイム レシーブ ノードです。

レベル 1

レベル 1 動作は、TTOCF.OM = 01 および TTOCF.GEN によって設定されます。外部クロック同期はレベル 1 では利用できません。
参照メッセージに関連する情報は、表 226 に示すように、最初のデータバイトに格納されます。Cycle_Count はオプションです。

表 225 レベル 1 参照メッセージの最初のバイト

ビット	7	6	5	4	3	2	1	0
最初のバイト	Next_is_Gap	予約済み	Cycle_Count[5:0]					

レベル 2

レベル 2 動作は、TTOCF.OM = 10 および TTOCF.GEN によって設定されます。
参照メッセージに関連する情報は、表 226 に示すように、最初の 4 つのデータ バイトに格納されます。Cycle_Count と NTU_Res の下位 4 ビットはオプションです。M_TTCAN は受信した参照メッセージから NTU_Res[3:0]を評価せず、これらのビットを常にゼロとして送信します。

表 226　　レベル 1 参照メッセージの最初の 4 バイト

ビット	7	6	5	4	3	2	1	0
最初のバイト	Next_is_Gap	予約済み	Cycle_Count[5:0]					
第 2 バイト	NTU_Res[6:4]			NTU_Res[3:0]				Disc_Bit
第 3 バイト	Master_Ref_Mark[7:0]							
第 4 バイト	Master_Ref_Mark[15:8]							

レベル 0

レベル 0 動作は TTOCF.OM = 11 で設定されます。レベル 0 では、外部イベントに同期したタイムトリガー動作は使用できません。
参照メッセージに関連する情報は、表 227 に示すように、最初の 4 つのデータ バイトに格納されます。レベル 0 では、Next_is_Gap は常に 0 です。Cycle_Count と NTU_Res の下位 4 ビットはオプションです。M_TTCAN は受信した参照メッセージから NTU_Res[3:0]を評価しません。常にこれらのビットを 0 として送信します。

表 227　　レベル 0 参照メッセージの最初の 4 バイト

ビット	7	6	5	4	3	2	1	0
最初のバイト	Next_is_Gap	予約済み	Cycle_Count[5:0]					
第 2 バイト	NTU_Res[6:4]			NTU_Res[3:0]				Disc_Bit
第 3 バイト	Master_Ref_Mark[7:0]							
第 4 バイト	Master_Ref_Mark[15:8]							

25.5.2　TTCAN コンフィギュレーション

25.5.2.1　TTCAN タイミング

ネットワーク時間単位 (NTU) は、時間の計測単位です。NTU はネットワーク全体の定数で、ネットワークシステム設計者によって定義されます。TTCAN レベル 1 では、NTU はノミナル CAN ビット時間です。TTCAN レベル 0 とレベル 2 では、NTU は物理的な秒の何分の 1 かを示します。

NTU はローカルタイムの基準となります。ローカルタイムの整数部 (16 ビット値) は、NTU ごとに 1 回インクリメントされます。サイクルタイムとグローバルタイムはどちらもローカルタイムから導かれます。ローカルタイム、サイクルタイム、グローバルタイムの端数部分 (3 ビット値) は読めません。

TTCAN レベル 0 およびレベル 2 では、NTU の長さは時間単位比 (TUR) によって定義されます。TUR は次のように与えられます。

TUR = TURNA.NAV/TURCF.DC

NTU の長さは以下によって決定されます。

NTU = CAN クロック周期 × TUR

TUR 分子コンフィギュレーション (NC) は 18 ビットの数値で、TURCF.NCL[15:0]は 0x0000-0xFFFF の範囲でプログラムされます。TURCF.NCH[17:16]は'0b01'にハードワイヤリングされます。TURCF.NCL[15:0]に数値'0xnnn'n を書き込むと、TURNA.NAV は 0x10000 + 0x0nnnn = 0x1nnnn で始まります。TUR 分母コンフィギュレーション (TURCF.DC) は 14 ビットの数値です。TURCF.DC は 0x0001～0x3FFF の範囲でプログラムされます。0x0000 は不正な値です。

レベル 1 では、NC は≧4×TURCF.DC でなければいけません。レベル 0 とレベル 2 では、NTU の内部小数部の 3 ビット分解能を可能にするため、NC は≧8×TURCF.DC でなければいけません。

ハードウェアリセットにより、TURCF.DC は 0x1000 に、TURCF.NCL は 0x10000 にプリセットされ、16 CAN クロック周期からなる NTU となります。CCCR.INIT がリセットされるか TURCF.ELT がセットされるまでは、ローカルタイムとアプリケーション ウォッチドッグは開始されません。TURCF.ELT は NTU が設定される前に設定されません。TURCF.ELT を'1'に設定すると、レジスタ TURCF への書き込みアクセスもロックされます。

起動時に TURNA.NAV は TURCF.ELT が設定されると NC (=TURCF.NCL +0x10000) から更新されます。TTCAN レベル 1 ではドリフト補正はありません。TURNA.NAV は動作中に変化することはなく、常に NC に等しいです。

TTCAN レベル 0 とレベル 2 では、TURNA.NAV には 2 つの変更の可能性があります。タイムスレーブまたはバックアップタイムマスターとして動作している場合で、TTOCF.ECC が設定されている場合、M_TTCAN が In_Schedule または In_Gap の同期状態にある限り、TURNA.NAV は監視されているグローバルタイム速度から計算された値に自動的に更新されます。同期を失うと NC に戻ります。実際のタイムマスターとして動作しているとき、TTOCF.EECS が設定されると、ホストは TURCF.NCL を更新します。ホストが TTOCN.ECS を設定すると、TURNA.NAV は次の参照メッセージで NC の新しい値から更新されます。ステータスフラグ TTOST.WECS は、TTOCN.ECS がセットされるとセットされ、TURNA.NAV が更新されるとクリアされます。TTOST.WECS が設定されている間、TURCF.NCL は書き込みロックされます。

TTCAN レベル 0 およびレベル 2 では、クロック キャリブレーションプロセスにより、TURNA.NAV は NC±2 (TTOCF.LDSDL+5) の同期偏差制限 (SDL) の範囲内で適応されます。TURNA.NAV の計算精度を高めるため、TURCF.NCL は最大の数値に設定する必要があります。

同期偏差 (SD) は、NC と TURNA.NAV の差 (SD = |NC - TURNA.NAV|) です。これは SDL によって制限され、SDL はその両対数によって設定されます。

TTOCF.LDSDL (SDL = 2 (TTOCF.LDSDL+5))

CAN ビットタイミング設定で指定されるクロック許容誤差を超えてはいけません (SD は各新しい基本サイクルで計算されます)。算出された TURNA.NAV が NC から SDL 以上ずれた場合、または参照メッセージの Disc_Bit が設定されている場合は、ドリフト補正を中断し、TTIR.GTE を設定し、TTOSC.QCS をリセットします。Disc_Bit = 1 の場合は、TTIR.GTD を設定します。表 228 に TUR 設定の例を示します。

表 228　　TUR 設定例

TUR	8	10	24	50	510	12500	32.5	100/12	529/17
NC	0x1FFF8	0x1FFFE	0x1FFF8	0x1FFEA	0x1FFFE	0x1FFE0	0x1FFE0	0x19000	0x10880
TURCF.DC	0x3FFF	0x3333	0x1555	0x0A3D	0x0101	0x0001	0xFC0	0x3000	0x0880

TTOCN.ECS は、次の参照メッセージで NC を起動するようにスケジュールします。TTOCN.SGT は TTGTP.TP を次の参照メッセージで起動するようにスケジュールします。TTOCN.ECS と TTOCN.SGT を設定するためには、M_TTCAN が実際の時間マスターである間に TTOCF.EECS が設定 (外部クロック同期が有効) されている必要があります。

M_TTCAN モジュールは、アプリケーション プログラムの機能を検証するためのアプリケーション ウォッチドッグを提供します。ホストはこのウォッチドッグに定期的にサービスを提供しなければならず、そうでない場合、すべての CAN バス アクティビティが停止します。アプリケーション ウォッチドッグ制限 TTOCF.AWL は、ウォッチドッグが提供されなければいけない NTU 数を指定します。NTU の最大値は 256 です。アプリケーション ウォッチドッグは、レジスタ TTOST を読み出すことで機能します。TTOST.AWE は、ウォッチドッグが時間内に果たされたかどうかを示します。アプリケーション ウォッチドッグに失敗した場合、割込みフラグ TTIR.AW がセットされます。ソフト

ウェア開発の場合、TTOCF.AWL を'0x00'にプログラミングすることでアプリケーション ウォッチドッグを無効にできます (アプリケーション ウォッチドッグを参照)。

25.5.2.2 メッセージのスケジューリング

TTOCF.TM は、M_TTCAN がポテンシャル タイムマスターとして動作するか、時間スレーブとして動作するかを制御します。それがポテンシャル タイムマスターである場合、参照メッセージ識別子 TTRMC.RID の 3 つの LSB がマスターの優先度を定義し、0 が最高で 7 が最低の優先度です。ネットワーク内の 2 つのノードは、同じマスター プライオリティの使用はできません。TTRMC.RID は参照メッセージを認識するために使用されます。TTRMC.RMPS はタイムスレーブには関係ありません。

初期リファレンス トリガー オフセット TTOCF.IRTO は 7 ビットの値で、参照メッセージが予期されますがバスがアイドルのままである場合に、バックアップ タイム マスターが参照メッセージの送信を開始するまでに待機する時間を (NTU 単位で) 定義します。TTOCF.IRTO の推奨値は、マスタープライオリティに、ネットワーク内のポテンシャル タイムマスター間で予想されるクロックドリフトに応じた係数を掛けたものです。バックアップ タイム マスターの順序は、現在のタイム マスターが故障した場合、そのうちの 1 つが参照メッセージを開始するとき、クロックドリフトが最大であっても、そのマスターの優先順位に対応する必要があります。

TTOCF.OM は、ノードが TTCAN レベル 0、レベル 1、レベル 2 のいずれで動作するかを決定します。1 つのネットワークでは、すべてのポテンシャル タイムマスターが同じレベルで動作します。タイムスレーブは、レベル 2 ネットワーク内のレベル 1 で動作できますが、その逆はできません。TTOCF.OM による TTCAN 動作モードの設定は、セットアップの最後のステップです。TTOCF.OM = 00 (イベント駆動型 CAN 通信) の場合、M_TTCAN は ISO 11898-1:2015 に従って動作し、タイムトリガーはありません。TTOCF.OM = 01 (レベル 1) の場合、M_TTCAN は ISO 11898-4 に従って動作しますが、基本サイクルを外部イベントに同期させることができないため、参照メッセージの Next_is_Gap ビットは無視されます。TTOCF.OM = 10 (レベル 2) の場合、M_TTCAN は、基本サイクルのイベント u0002 同期開始を含め、ISO 11898-4 に従って動作します。TTOCF.OM = 11 (レベル 0) の場合、M_TTCAN はイベント駆動型 CAN として動作しますが、レベル 2 と同様に校正されたグローバルタイムベースを維持します。

TTOCF.EECS は外部クロック同期を有効にし、現在のタイムマスターのアプリケーションプログラムが、タイムトリガー動作中に TUR 設定を更新し、クロック速度と (レベル 0,2 のみ) グローバルクロック位相を外部基準に合わせることを可能にします。

TT Matrix Limits レジスタの TTMLM.ENTT は、システム マトリックスで予想される Tx_Trigger の数を指定します。これは、Tx_Ref_Triggers を除いた、排他的単一アービトレーションおよびマージアービトレーションウィンドウの Tx_Triggers の合計です。

***注**:* *これは通常 Tx_Trigger のメモリエレメント数ではなく、システム マトリックスの基本サイクル数とトリガーのリピート係数を考慮する必要があります。TTMLM.ENTT の設定が不正確な場合、TX カウントのアンダーフロー (TTIR.TXU = 1、TTOST.EL = 01、重大度 1)、または TX カウントのオーバーフロー (TTIR.TXO = 1、TTOST.EL = 10、重大度 2) が発生します。*

***注**:* *ノードが最初に見た参照メッセージの Cycle_Count が 0 でない場合、このノードは最初のマトリックス サイクルを TX カウントで終了し、その結果 TX カウントがアンダーフロー状態になります。ノードがステートにある限り、その Tx_Trigger を同期しても送信は行われません。*

TTMLM.CCM は、システム マトリックスの最後の基本サイクルの番号を指定します。基本サイクルのカウントは'0'から始まります。8 つの基本サイクルからなるシステム マトリックスでは、TTMLM.CCM は'7'となります。TTMLM.CCM は時間スレーブによって無視され、参照メッセージの受信者は受信したサイクルカウントを実際の基本サイクルの有効なサイクルカウントとみなします。

TTMLM.TXEW は、NTU 単位で TX イネーブルウィンドウの長さを指定します。TX イネーブルウィンドウは、送信を開始するタイムウィンドウの先頭の期間です。送信メッセージの最初のビットのサンプルポイントが TX イネーブルウィンドウ内にない場合、そのタイムウィンドウ内で送信を開始することは全くできません。例えば、前のタイムウィンドウのメッセージと重複しているためです。TTMLM.TXEW は、ネットワークの同期品質と、タイムウィンドウの長さとメッセージの長さの関係に基づいて選択されます。

25.5.2.3 トリガー メモリ

トリガー メモリは MRAM の一部です。最大 64 のトリガー エレメントを保存できます。トリガー メモリ エレメントは、タイム マーク (TM)、サイクル コード (CC)、トリガー タイプ (TYPE)、フィルター タイプ (FTYPE)、メッセージ番号 (MNR)、メッセージ ステータス カウント (MSC)、タイム マーク イベント内部 (TMIN)、タイム マーク イベント外部 (TMEX)、非同期シリアル通信 ASC (詳細は、トリガー メモリ エレメントを参照) から構成されます。

タイムマークは、トリガーがアクティブになるサイクルタイムを定義します。トリガーメモリ内のトリガーエレメントは、タイムマークでソートされます。最も低いタイムマークを持つトリガーエレメントが、最初のトリガーメモリワードに書き込まれます。Tx_Ref_Trigger, Tx_Ref_Trigger_Gap, Watch_Trigger, Watch_Trigger_Gap, End_of_List タイプのトリガーでは、メッセージ番号とサイクルコードは無視されます。

サイクルタイムが実際のトリガーのタイムマークに達すると、FSE は次のトリガーに切り替わり、トリガーメモリから読み出しを開始します。送信トリガーの場合、TX ハンドラは、FSE がそのトリガーに切り替わるとすぐに、メッセージ RAM からメッセージの読み出しを開始します。RAM アクセス速度は、送信トリガーとその前のトリガーとの間の最小時間ステップを定義します。TX ハンドラは、送信トリガーのタイムマークに到達する前に、送信の準備ができる必要があります。また、RAM アクセス速度は、2 つの一致するトリガー間の非一致 (サイクルコードに関して) トリガーの数を制限し、次の一致するトリガーは、そのタイムマークに達する前に読み出されなければいけません。参照メッセージが n NTU の長さである場合、n 未満のタイムマークを持つトリガーは決してアクティブにならず、コンフィギュレーションエラーとして扱われます。

サイクルタイムの開始点は、参照メッセージのフレーム開始ビットのサンプル点です。サイクル タイムが Tx_Ref_Trigger のタイム マークに達すると、次の参照メッセージが要求されます。M_TTCAN は、次のサンプル ポイントで送信要求に応答します。新しい Sync_Mark はフレーム開始ビットで捕捉されますが、参照メッセージが正常に送信 (または受信) され、Sync_Mark が新しい Ref_Mark として捕捉されるまで、サイクルタイムはインクリメントされます。この時点でサイクルタイムが再スタートします。その結果、サイクルタイムは (初期化を除いて) n より小さい値になることはなく、n は NTU で測定される参照メッセージの長さです。

基本サイクルの長さ: Tx_Ref_Trigger タイムマーク + 1 NTU + 1 CAN ビット時間

トリガーリストは、TTCAN ネットワーク内のすべてのノードで異なります。各ノードは、自分自身の送信メッセージの Tx_Triggers、このノードが処理する受信メッセージの Rx_Triggers、および参照メッセージに関するトリガーのみを認識します。

25.5.2.3.1 トリガーの種類

Tx_Ref_Trigger (TYPE = 0000) および Tx_Ref_Trigger_Gap (TYPE = 0001) は、タイム マスターによる参照メッセージの送信を引き起こします。コンフィギュレーションエラー (TTOST.EL = 11, 重大度 3) は、タイムスレーブがトリガーメモリ内で Tx_Ref_Trigger (_Gap) に遭遇したときに検出されます。Tx_Ref_Trigger_Gap は、外部イベント同期時間トリガー動作モードでのみ使用されます。このモードでは、M_TTCAN の同期状態が In_Gap (TTOST.SYS = 10) のとき、Tx_Ref_Trigger は無視されます。

Tx_Trigger_Single (TYPE = 0010)、Tx_Continous (TYPE = 0011)、Tx_Trigger_Arbitration (TYPE = 0100)、Tx_Trigger_Merged (TYPE = 0101) は、送信の開始を引き起こします。タイムウィンドウの開始を定義します。

Tx_Trigger_Single は、メッセージ バッファの Transmission Request Pending ビットが設定されると、排他的なタイムウィンドウで 1 回の送信を開始します。送信成功後、送信要求保留ビットはリセットされます。

Tx_Trigger_Continuous は、メッセージ バッファの Transmission Request Pending ビットが設定されると、排他的なタイムウィンドウで送信を開始します。送信に成功した後、送信要求保留ビットはセットされたままで、メッセージ バッファは次に一致するタイムウィンドウで再び送信されます。

Tx_Trigger_Arbitration はアービトレーション タイムウィンドウを開始し、Tx_Trigger_Merged はマージされたアービトレーション タイムウィンドウを開始します。マージされたアービトレーション タイムウィンドウの最後の Tx_Trigger は、Tx_Trigger_Arbitration 型でなければいけません。Tx_Trigger_Merged タイプのトリガーの後に、Tx_Trigger_Merged タイプまたは Tx_Trigger_Arbitration タイプ以外の Tx_Trigger が続くと、コンフィギュレーションエラー (TTOST.EL = 11, 重大度 3) が検出されます。同じ TX メッセージ バッファに対して、複数の Tx_Trigger を

定義できる場合があります。サイクルコードによっては、基本サイクルでは無視されることもあります。Tx_Triggers (TTMLM.ENTT) の予想数を計算する際には、サイクルコードを考慮する必要があります。

Watch_Trigger (TYPE = 0110) と Watch_Trigger_Gap (TYPE = 0111) は、参照メッセージの欠落をチェックします。時間のマスターおよびスレーブも使用します。Watch_Trigger_Gap は、外部イベント同期時間トリガー動作モードでのみ使用されます。このモードでは、M_TTCAN の同期状態が In_Gap (TTOST.SYS = 10) の場合、Watch_Trigger は無視されます。

Rx_Trigger (TYPE = 1000) は、排他的なタイムウィンドウで定期的なメッセージの受信をチェックするために使用されます。In_Schedule または In_Gap の状態になるまで、Rx_Triggers はアクティブになりません。Rx_Trigger のタイムマークは、タイムウィンドウの境界に関係なく、そのメッセージ送信の終了後に置かれる必要があります。サイクルコードによっては、Rx_Triggers は基本サイクルでは無視されます。Rx_Trigger のタイムマークで、このタイムマークの前、サイクルの開始後、または前の Rx_Trigger の後に最後に受信したメッセージが、MNR によって参照される受入フィルター エレメントと一致するかどうかがチェックされます。受理されたメッセージは、Rx_Trigger とは無関係に、受入フィルタリングに従って 2 つの受信 FIFO のうちの 1 つに格納されます。Rx_Trigger によって参照される受入フィルター エレメントは、Rx_Trigger のタイム マークに到達する前にフィルタリングが終了するように、フィルタリング リストの先頭に配置してください。

Time_Base_Trigger (TYPE = 1001) は、ASC、TMIN、TMEX の設定に応じて内部/外部イベントを生成するために使用されます。

End_of_List (TYPE = 1010...1111) が不正なトリガータイプである場合、Watch_Trigger または Watch_Trigger_Gap の前に、End_of_List トリガーがトリガーメモリ内で検出されると、コンフィギュレーションエラー (TTOST.EL = 11, 重大度 3) が検出されます。

25.5.2.3.2 ノードのトリガーリストの制限事項

2 つのトリガーが同じサイクルタイム、サイクルカウントでアクティブになることはありませんが、異なる基本サイクル (異なるサイクルコード) でアクティブになるトリガーは同じタイムマークを共有します。

Rx_Trigger と Time_Base_Trigger は、Tx_Trigger_Single/Continuous/Arbitration の TX イネーブルウィンドウ内に配置できませんが、Tx_Trigger_Merged の後に配置されます。

Watch_Trigger (TTOST.SYS = 10 の場合は Watch_Trigger_Gap) の後に置かれたトリガーは決してアクティブになりません。参照メッセージが時間どおりに送信されると、時計トリガー自体はアクティブになりません。

未使用のすべてのトリガーメモリワード (Watch_Trigger の後、または TTOST.SYS = 10 の時の Watch_Trigger_Gap の後) は、トリガータイプ End_of_List に設定されなければいけません。

ポテンシャル タイムマスターの典型的なトリガーリストは、いくつかの Tx_Trigger と Rx_Trigger で始まり、Tx_Ref_Trigger と Watch_Trigger が続きます。外部イベント同期時間トリガー通信を持つネットワークでは、Tx_Ref_Trigger_Gap と Watch_Trigger_Gap がこれに続きます。タイムスレーブのトリガーリストは同じですが、Tx_Ref_Trigger と Tx_Ref_Trigger_Gap がありません。

各基本サイクルの開始時、つまり参照メッセージの受信または送信のたびに、トリガーリストは最初のトリガーメモリエレメントから処理されます。FSE は、現在のサイクルカウントと一致するサイクルコードを持つ最初のトリガーを探します。FSE は、サイクルタイムがトリガーのタイムマークに達するまで待機し、トリガーを作動させます。その後、FSE は、現在のサイクルカウントと一致するサイクルコードを持つリストの次のトリガーを探します。

Tx_Ref_Trigger と Tx_Ref_Trigger_Gap の前後については、特別な配慮が必要です。マスターシップを競うタイムマスターでは、Tx_Ref_Trigger の有効なタイムマークがデクリメントされ、参照メッセージを開始する最初のノードとなります。バックアップ タイム マスターでは、Tx_Ref_Trigger または Tx_Ref_Trigger_Gap の有効なタイム マークは、設定されたタイム マークとリファレンス トリガー オフセット TTOCF.IRTO の合計です。エラーレベル 2 に達した場合 (TTOST.EL = 10)、有効なタイムマークはそのタイムマークと 0x127 の合計となります。そうでない場合、タイムマークは順番どおりに表示されず、コンフィギュレーションエラーとしてフラグが立てられます。Tx_Ref_Trigger の後に来るトリガー エレメントは、参照メッセージが時間どおりに来るまでアクティブになりません。

以下のパラメーターの間には相互依存関係があります。

- ホストクロック周波数

- トリガー RAM アクセスの速度と待ち時間
- 受入フィルターリストの長さ
- トリガーエレメントの数
- トリガーエレメントにおけるサイクルコードのフィルタリングの複雑さ
- トリガーエレメントのタイムマーク間のオフセット

25.5.2.3.3 トリガー処理の例

以下の例に、ノードのシステム マトリックスからトリガーリストがどのように導出されるかを示します。ノード A が最初のマスターであると仮定します。表 229 にシステム マトリックスのセクションを示します。

表 229 システム マトリックス ノード A

サイクル数	タイムマーク 1	タイムマーク 2	タイムマーク 3	タイムマーク 4	タイムマーク 5	タイムマーク 6	タイムマーク 7
0	Tx7					TxRef	エラー
1	Rx3		Tx2, Tx4			TxRef	エラー
2						TxRef	エラー
3	Tx7		Rx5			TxRef	エラー
4	Tx7			Rx6		TxRef	エラー

サイクル数は 0 から始まり、0、1、3、7、15、31、63 (システム マトリックスの基本サイクル数は 1、2、4、8、16、32、64)。最大サイクル数は TTMLM.CCM によって設定されます。サイクルコード (CC) は、リピート係数 (最上位'1'の値) と、システム マトリックスの最初の基本サイクルの番号 (最上位'1'の後のビットフィールド) で設定されます。

例えば、CC が 0b0010011 (繰り返し係数: 16、最初の基本サイクル: 3) で、TTMLM.CCM の最大サイクルカウントが 0x3F の場合、サイクルカウント 3、19、35、51 でマッチングが発生します。

トリガーエレメントは、タイムマーク (TM)、サイクルコード (CC)、トリガータイプ (TYPE)、メッセージ番号 (MNR) から構成されます。送信時、MNR は TX バッファ番号 (0～31) を参照します。受信の場合、MNR は受入フィルタリング中に一致したフィルターエレメントの番号 (0～127) を参照します。フィルター タイプ FTYPE の設定に応じて、11 ビットまたは 29 ビットのメッセージ ID フィルター リストが参照されます。

さらに、トリガーエレメントは、非同期シリアル通信 (ASC)、タイムマークイベント内部 (TMIN)、タイムマークイベント外部 (TMEX) の生成のために設定できます。メッセージ ステータス カウント (MSC) は、トリガー エレメントのタイム マークがアクティブになった時、排他的なタイムウィンドウの定期メッセージのスケジューリング エラーのカウンター値 (0～7) を保持します。

表 230 トリガーリストノード A

トリガー	タイムマーク TM[15:0]	サイクルコード CC[6:0]	トリガータイプ TYPE[3:0]	メッセージ番号 MNR[6:0]
0	Mark1	0000100	Tx_Trigger_Single	7
1	Mark1	1000000	Rx_Trigger	3
2	Mark1	1000011	Tx_Trigger_Single	7
3	Mark3	1000001	Tx_Trigger_Merged	2
4	Mark3	1000011	Rx_Trigger	5

(続く)

表 230 (続き) トリガーリストノード A

トリガー	タイムマーク TM[15:0]	サイクルコード CC[6:0]	トリガータイプ TYPE[3:0]	メッセージ番号 MNR[6:0]
5	Mark4	1000001	Tx_Trigger_Arbitration	4
6	Mark5	1000100	Rx_Trigger	6
7	Mark6	該当なし	Tx_Ref_Trigger	0 (Ref)
8	Mark7	該当なし	Watch_Trigger	対象外
9	該当なし	該当なし	End_of_List	対象外

Tx_Trigger_Single、Tx_Trigger_Continous、Tx_Trigger_Merged、Tx_Trigger_Arbitration、Rx_Trigger、および Time_Base_Trigger は、指定されたサイクルコードに対してのみ有効です。他のすべてのトリガータイプでは、サイクルコードは無視されます。

FSE は、サイクルタイムより大きいタイムマークを持つトリガーに到達するか、CC が実際のサイクルカウントに一致するか、または Tx_Ref_Trigger、Tx_Ref_Trigger_Gap、Watch_Trigger、Watch_Trigger_Gap タイプのトリガーに遭遇するまで、ゼロから始まるトリガーリストをスキャンすることにより、基本サイクルを開始します。

サイクルタイムが TM に達すると、TYPE と MNR で定義された動作が開始されます。End_of_List に達した時点でコンフィギュレーションにエラーが発生します。Mark6 で、参照メッセージ (常に TxRef) が送信されます。送信後、FSE はトリガーリストの先頭に戻ります。Mark7 のウォッチ トリガーに達すると、ノードは参照メッセージを送信できなくなり、エラー処理が開始されます。

25.5.2.3.4 コンフィギュレーション エラーの検出

以下の場合、TTOST.EL = 11 (重大度 3) を介してコンフィギュレーション エラーが通知されます。

- FSE は、現在のサイクルカウントと一致する CC を持ちますが、サイクルタイムより短いタイムマークを持つトリガーをリスト内に見つけます。
- 前のアクティブトリガーは Tx_Trigger_Merged で、FSE は、現在のサイクルカウントと一致する CC を持つトリガーをリストに含みますが、それは Tx_Trigger_Merged でも Tx_Trigger_Arbitration でも Time_Base_Trigger でも Rx_Trigger でもありません。
- TTOCF.TM = 0 (タイムスレーブ) のノードの FSE が Tx_Ref_Trigger または Tx_Ref_Trigger_Gap を検出します。
- サイクルコードが一致する Tx_Trigger の TX イネーブルウィンドウ (TTMLM.TXEW で定義) 内に配置されたタイムマーク。
- Tx_Ref_Trigger と基準トリガーオフセット TTOST.RTO のタイムマークの近くにタイムマークが置かれると、サイクルタイムで測定されるそれらの順序が逆転します。

25.5.2.4 TTCAN スケジュールの初期化

CCCR.INIT がリセットされると、M_TTCAN メッセージ スケジュールへの同期が開始されます。M_TTCAN は、タイム トリガー (TTOCF.GEN = 0) または外部イベント u0002 同期タイム トリガー (TTOCF.GEN = 1) で動作します。すべてのノードは、TTOST.SYS = 00 (同期外れ) のトリガーリストの先頭で、サイクル時間ゼロで開始します。外部イベント同期時間トリガー動作モードのノードは、Tx_Ref_Trigger と Watch_Trigger を無視し、Tx_Ref_Trigger_Gap と Watch_Trigger_Gap を使用します。

25.5.2.4.1 タイムスレーブ

コンフィギュレーション後、タイムスレーブは Watch_Trigger に到達する前にメッセージを受信しなかった場合、その Watch_Trigger と Watch_Trigger_Gap を無視します。Init_Watch_Trigger に達すると、割込みフラグ TTIR.IWT がセットされ、FSE はフリーズし、サイクルタイムは無効となります。しかし、ノードは依然として CAN バス通信に参加しています (アクノリッジを与えたり、エラーフラグを送信したりします)。最初に受信した参照メッセージにより、FSE とサイクル タイムが再スタートします。

注: Init_Watch_Trigger はトリガーリストの一部ではありません。これは、0xFFFF ＝最大サイクル時間までカウントする内部カウンターとして実装されます。

Watch_Triggers に到達する前に、タイムスレーブが参照メッセージ以外のメッセージを受信した場合、致命的なエラー (TTOST.EL = 11、重大度 3) と判断し、割込みフラグ TTIR.WT を設定し、CAN バス出力をオフにし、バス監視モード (CCCR.MON を'1'に設定) に入ります。バス モニタリング モードでは、メッセージを受信しますが、ドミナント ビットを送信できないため、確認応答は行われません

注: 致命的エラー状態から抜け出すには、ホストは CCCR.INIT を'1'に設定する必要があります。CCCR.INIT のリセット後、ノードは TTCAN 通信を再開します。

同期中にエラーが発生しなかった場合、最初の参照メッセージは TTOST.SYS = 01 (同期中) に設定し、2 番目の参照メッセージは TTCAN の同期状態を (その Next_is_Gap ビットに応じて) TTOST.SYS = 11 (In_Schedule) または TTOST.SYS = 10 (In_Gap) に設定し、すべての Tx_Trigger と Rx_Trigger を有効にします。

25.5.2.4.2 ポテンシャル タイム マスター

コンフィギュレーション後、ポテンシャル タイム マスターは、Tx_Ref_Trigger (外部イベント同期時間トリガー動作の場合は Tx_Ref_Trigger_Gap) に達すると、参照メッセージの送信を開始します。Watch_Triggers に到達する前に、Watch_Trigger と Watch_Trigger_Gap を無視します。Init_Watch_Trigger に達すると、送信は中止され、割込みフラグ TTIR.IWT がセットされ、FSE はフリーズし、サイクルタイムは無効になりますが、ノードは CAN バス通信に参加します (アクノリッジまたはエラーフラグの送信)。TTIR.IWT をリセットすると、次に Init_Watch_Trigger 条件が満たされるか、別の CAN メッセージが受信されるまで、参照メッセージの送信が再び有効になります。FSE は、参照メッセージの受信によって再起動します。

ポテンシャル タイム マスターが、参照メッセージ以外のメッセージを受信した後に Watch_Triggers に達すると、致命的なエラー (TTOST.EL = 11、重大度 3) と判断し、割込みフラグ TTIR.WT をセットし、CAN バス出力をオフにし、バス モニタリング モード (CCCR.MON を'1'にセット) に入ります。バス モニタリング モードでは、メッセージは受信できますが、ドミナント ビットは送信できず、したがって確認応答もできません。

初期化中にエラーが検出されなかった場合、最初の参照メッセージは TTOST.SYS = 01 (同期中) に設定し、2 番目の参照メッセージは TTCAN の同期状態を (その Next_is_Gap ビットに応じて) TTOST.SYS = 11 (In_Schedule) または TTOST.SYS = 10 (In_Gap) に設定し、すべての Tx_Trigger と Rx_Trigger を有効にします。

ポテンシャル タイムマスターは、それが最後の参照メッセージの送信者である場合、現在の時間マスター (TTOST.MS = 11) で、そうでない場合、バックアップ時間マスター (TTOST.MS = 10) です。

すべてのタイムマスター候補がコンフィギュレーションを終了すると、ネットワーク内でタイムマスターの優先順位が最も高いノードが現在のタイムマスターとなります。

25.5.3 TTCAN ギャップ制御

ギャップ制御に関するすべての機能は、M_TTCAN が外部イベント同期時間トリガー モード (TTOCF.GEN = 1) で動作している場合にのみ適用されます。この動作モードでは、システム マトリックスの基本サイクル間にギャップを挿入することで、TTCAN メッセージスケジュールを中断できます。CAN ネットワークに接続されているすべてのノードは、外部イベントと同期したタイムトリガー動作に設定されている必要があります。

ギャップの間、すべての送信は停止され、CAN バスはアイドル状態を維持します。ギャップは、次の参照メッセージが新しい基本サイクルを開始したときに終了します。例えば、ギャップは現在のタイム マスターによって開始されます。

現在のタイム マスターには、ギャップを開始するための 2 つのオプションがあります。アプリケーション プログラムが TTOCN.NIG = 1 を書き込むと、ソフトウェア制御でギャップを開始できます。Next_is_Gap ビットは次の参照メッセージと共に'1'として送信されます。ギャップは、アプリケーション プログラムが TTOCN.GCS = 1 を書き込むと、ハードウェア制御下で開始することもできます。参照メッセージが開始され、TTOCN.GCS が設定されると、Next_is_Gap = 1 が設定されます。

その参照メッセージが完了するとすぐに、TTOST.WFE ビットはマスターとレプリカの間の時間のずれをアナウンスします。現在の基本サイクルは、最後のタイムウィンドウまで続きます。最後のタイムウィンドウの後の時間がギャップタイムです。

実際のタイムマスターおよびポテンシャル タイムマスターの場合、最後の基本サイクルが終了し、ギャップタイムが開始されると、TTOST.GSI が設定されます。タイムスレーブノードでは、TTOST.GSI ビットは'0'のままです。

ポテンシャルタイム マスターが同期状態 In_Gap (TTOST.SYS = 10) にある場合、以下の 4 つのオプションがあります。

- TTOCN.FGP = 1 書き込みによってソフトウェア制御されます。
- ハードウェア制御 (TTOCN.GCS = 1) では、内部イベントトリガー入力端子の HIGH から LOW へのエッジを検出すると、自動的に TTOCN.FGP がセットされ、スケジュールをリスタートします。
- 第 3 の選択肢は、タイムトリガーによる再スタートです。TTOCN.TMG = 1 の時、次のレジスタタイムマーク割込み (TTIR.RTMI = 1) は TTOCN.FGP をセットし、参照メッセージを開始します。
- ポテンシャル タイムマスターは、Tx_Ref_Trigger_Gap に到達した時点でギャップを終了します。

これらのオプションはいずれも、基本サイクルを参照メッセージで中断できません。

ギャップタイム開始後に TTOCN.FGP を設定すると、参照メッセージの送信を直ちに開始し、メッセージ スケジュールを同期させます。TTOCN.FGP がギャップ タイムが始まる前 (基本サイクルの進行中) に設定された場合、次の参照メッセージは基本サイクルの終わり、Tx_Ref_Trigger で開始されます。

タイムトリガー動作では、参照メッセージの Next_is_Gap = 1 ビットは無視され、TTOCN.NIG, TTOCN.FGP, TTOCN.TMG ビットが考慮されます。

25.5.4 ストップウォッチ

ストップウォッチ機能は、外部イベントによってトリガーされた M_TTCAN 内部時間値 (ローカルタイム、サイクルタイム、またはグローバルタイム) のキャプチャを可能にします。

ストップウォッチ機能を有効にするためには、まずアプリケーション プログラムが TTOCN.SWS を介してストップウォッチ ソースとしてローカルタイム、サイクルタイム、またはグローバルタイムを定義する必要があります。TTOCN.SWS が'00'でなく、TT 割込みレジスタフラグ TTIR.SWE が'0'の場合、TTOCN.SWS で選択された時間の実際の値は、ストップウォッチトリガーの次の立ち上り/立ち下りエッジ (TTOCN.SWP で設定) で TTCPT.SWV にコピーされます。これにより割込みフラグ TTIR.SWE がセットされます。アプリケーション プログラムが TTCPT.SWV を読み出した後、TTIR.SWE を'0'にリセットすることで、次のストップウォッチ イベントの有効化ができます。

25.5.5 ローカルタイム、サイクルタイム、グローバルタイム、外部クロック同期

タイムトリガー CAN には、レベル 1 とレベル 2 の 2 つのレベルがあります。レベル 1 は、サイクルタイムを使用したタイムトリガー動作のみを提供します。レベル 2 はさらに、同期品質の向上、グローバルタイム、外部クロック同期を提供します。どちらのレベルでも、すべてのタイミング機能はローカルタイムをベースにしています。

ローカルタイムは 16 ビットの周期カウンターで、各 NTU で 1 回インクリメントされます。内部的には、NTU は 3 ビットのカウンターで表され、これはローカルタイムの小数部 (2 進数 3 桁) とみなせます。通常、3 ビットの NTU カウンターは NTU ごとに 8 回インクリメントされます。NTU の長さが 8 つの CAN クロック周期より短い場合 (レベ

ル 1 で設定されるか、レベル 2 でクロック キャリブレーションの結果として)、NTU 分数の長さが適応され、NTU カウンターは NTU ごとに 4 回だけインクリメントされます。

図 209 はサイクルタイムとグローバルタイムの同期を示していて、タイムマスターを含むすべての TTCAN ノードによって同じ方法で実行されます。受信または送信されたメッセージはすべて、そのメッセージのフレーム同期イベントで取得されたローカルタイムのキャプチャを呼び出します。このフレーム同期イベントは、各フレーム開始 (SoF) ビットのサンプルポイントで発生し、ローカルタイムが Sync_Mark として保存されます。Sync_Marks と Ref_Marks は、3 ビットの小数部を含めてキャプチャされます。

有効な参照メッセージが送受信されるたびに、内部の Ref_Mark は Sync_Mark から更新されます。Ref_Mark と Sync_Mark の差は、TTCSM レジスタに格納されているサイクル シンク マーク (Cycle Sync Mark = Sync_Mark - Ref_Mark) です。Ref_Mark とローカルタイムの実際の値の差の最上位 16 ビットがサイクルタイム (サイクルタイム＝ローカルタイム-Ref_Mark) です。

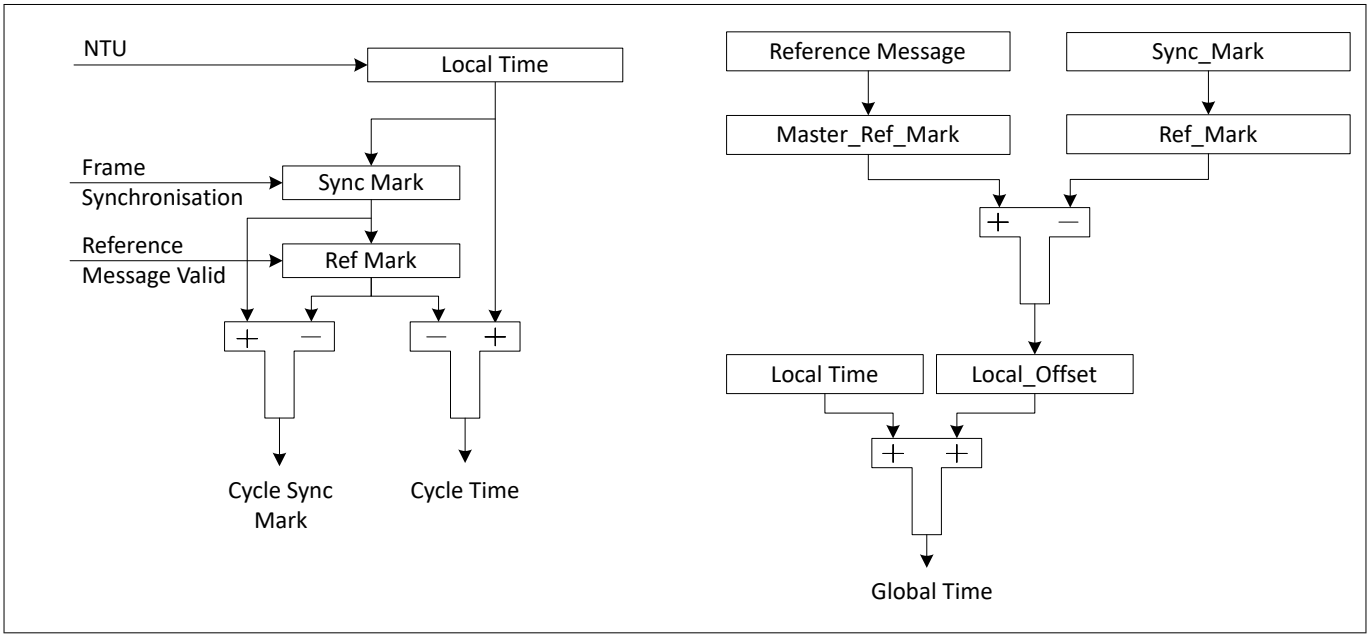


図 209　　サイクルタイムとグローバルタイム同期

TTCTC.CT から読み出すことができるサイクルタイムは、ノードのローカルタイムと Ref_Mark の差で、両方ともホストクロックドメインに同期され、16 ビットに切り捨てられたものです。

グローバルタイムは TTCAN レベル 0 とレベル 2 のみに存在し、レベル 1 では無効です。ノードのグローバルタイムに対する見かたは、(ローカル) NTU におけるグローバルタイムのローカルイメージです。コンフィギュレーション後、ポテンシャル タイムマスターは自身のローカルタイムをグローバルタイムとして使用します。これは、自身の Ref_Marks を参照メッセージの Master_Ref_Marks として送信することで行われます (バイト 3 と 4)。グローバルタイムは TTLGT.GT から読み出され、ノードのローカルタイムとローカルオフセットの合計です。両方ともホストクロックドメインに同期され、16 ビットに切り捨てられます。小数部はクロック同期にのみ使用されます。

参照メッセージを受信したノードは、そのローカルの Ref_Mark と受信した Master_Ref_Mark を比較して、グローバルタイムに対するローカル オフセットを計算します (図 209 を参照)。グローバルタイムに対するノードの見かたは、ローカルタイム＋ローカルオフセットです。他のタイム マスターの参照メッセージを受信したことのないポテンシャル タイムマスターでは、Local_Offset はゼロになります。他の参照メッセージを先に受信したノードが現在のタイム マスターになると、Local_Offset はその最後の値で凍結されます。時刻受信ノードでは、他のノードがタイム マスターになったとき、または参照メッセージの Disc_Bit で示されるグローバルタイムの不連続が発生したときに、クロックドリフトにより、Local_Offset が若干調整されることがあります。グローバルタイム不連続の例外として、これは TTLGT レジスタによってアプリケーション プログラムに提供され、連続的な単調値を持つようにローパス フィルタリングによって平滑化されます。

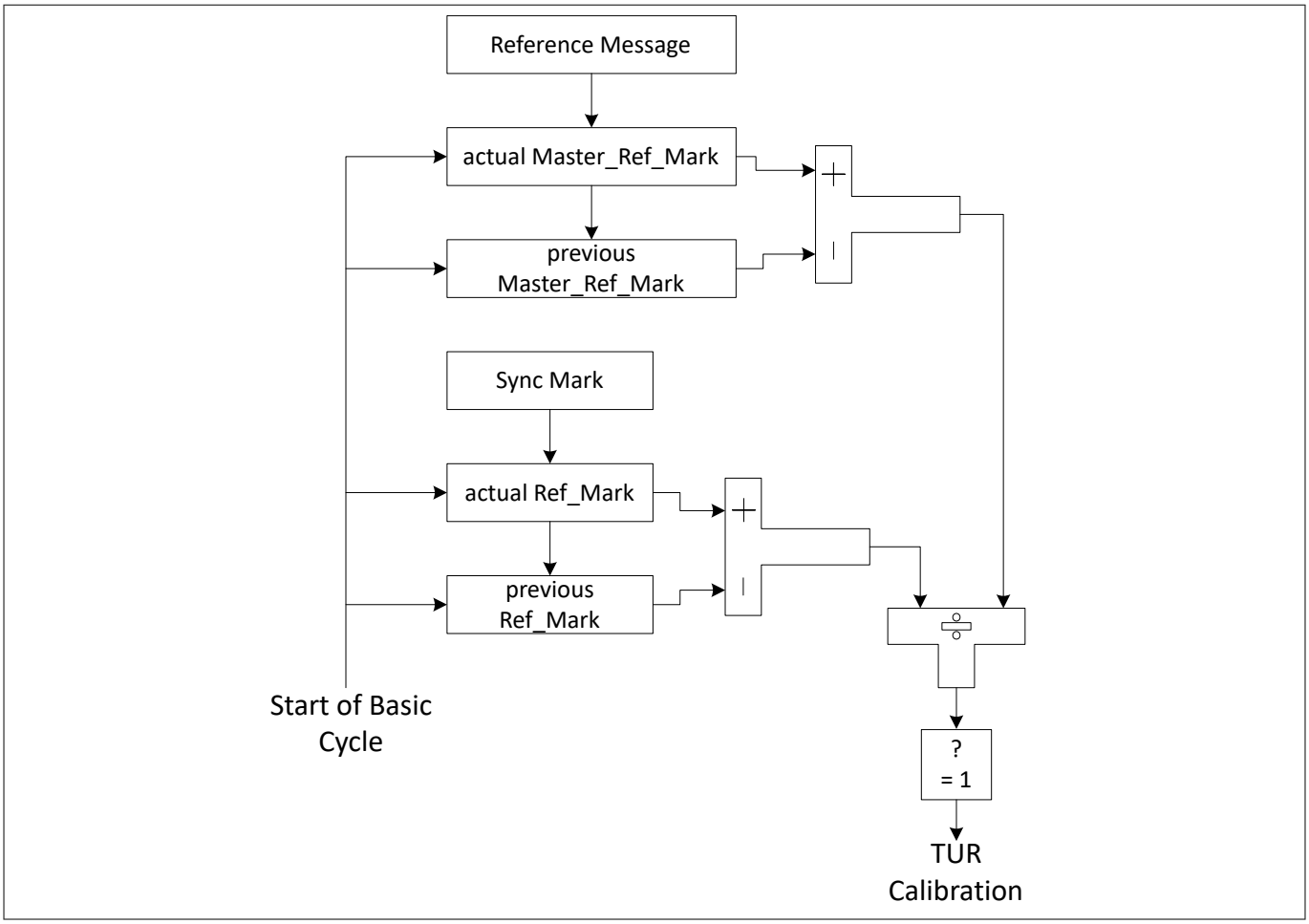


図 210　　TTCAN レベル 0 およびレベル 2 ドリフト補正

図 210 に、TTCAN レベル 0 とレベル 2 において、受信ノードが、ローカルタイムとグローバルタイムの基本サイクルの長さを比較することによって、自身のローカルクロックとタイムマスターのクロックの間のドリフトをどのように補正するかを示します。2 つの値に差があり、参照メッセージの Disc_Bit が設定されていない場合、TURNA.NAV の新しい値が計算されます。同期偏差 (SD) = |NC - TURNA.NAV| ≦SDL の場合、TURNA.NAV の新しい値が有効となります。そうでなければ、自動ドリフト補正は中断されます。

TTCAN レベル 0 およびレベル 2 では、TTOST.QCS は自動ドリフト補正がアクティブかサスペンドかを示します。TTCAN レベル 1 では、TTOST.QCS は常に'1'です。

現在のタイムマスターは、そのローカルクロック速度とグローバルタイムの位相を外部クロック ソースに同期させる場合があります。これは TTOCF.EECS ビットで有効になります。ストップウォッチ機能 (ストップウォッチ参照) を使って、ローカルクロックと外部クロックのクロックスピードの差の計測する場合があります。ローカルクロック速度は、最初に新しく計算された Numerator Configuration Low を TURCF.NCL に書き込むことで調整されます (動作中に TURCF.DC は更新できません)。新しい値は、TTOCN.ECS を'1'に書き込むことで有効になります。

グローバルタイムの位相は、まず位相オフセットを TT グローバルタイム プリセット レジスタ (TTGTP) に書き込むことで調整されます。新しい値は、TTOCN.SGT を'1'に書き込むことで有効になります。グローバルタイムの位相調整後に送信される最初の参照メッセージは、Disc_Bit が'1'に設定されます。

TTOST.QGTP は、ノードのグローバルタイムがタイムマスターのグローバルタイムと同位相であるかどうかを示します。TTOST.QGTP は、TTCAN レベル 1 および TTCAN レベル 0,2 で SDL を超過した場合 (TTOST.QCS = 0) は、恒久的に'0'です。アプリケーションに連続した単調な値を供給するため、グローバル タイムがローパスフィルターされている間、一時的に'0'になります。最後の参照メッセージに Disc_Bit = 1 が含まれている場合、または TTOST.QCS = 0 の場合、ローパスフィルタリングは行われません。

25.5.6 TTCAN エラーレベル

ISO 11898-4 では、エラーの重大度を 4 段階で規定しています。

表 231 エラーの重大度レベル

レベル	重大度	説明
S0	エラーなし	
S1	警告	エラーの通知のみで、リアクションはエラーに依存します。
S2	エラー	エラーの通知。排他的またはアービトレーション的なタイムウィンドウにおけるすべての送信は無効になります (つまり、データまたはリモートフレームは開始できません)。Potential time masu0002ters は、リファレンストリガー オフセット TTOST.RTO を最大値 127 に設定しても参照メッセージを送信します。
S3	重大なエラー	エラーの通知。すべての CAN バス動作が停止し、ドミナントビットの送信は許可されず、CCCR.MON が設定されます。S3 エラー状態は、アプリケーションがコンフィギュレーションを更新 (CCCR.CCE を設定) するまで有効なままです。

複数のエラーが同時に検出された場合は、最も重大度の高いものが優先されます。エラーが検出されると、TTIR.ELC によってアプリケーションに通知されます。エラーレベルは TTOST.EL によって監視されます。

M_TTCAN は、ISO 11898-4 の要件に従って以下のエラー状態を通知します。

表 232 エラー条件

エラー条件	説明
Config_Error (S3)	マージされたアービトレーション タイムウィンドウが適切に閉じられていない場合、または Tx_Ref_Trigger を超えるタイムマークを持つ Tx_Trigger が存在する場合、エラーレベル TTOST.EL を‘11’に設定します。
Watch_Trigger_Reached (S3)	参照メッセージが見つからないためにウォッチトリガーに達した場合、エラー レベル TTOST.EL を'11'に設定します。

(続く)

表 232 (続き) エラー条件

エラー条件	説明
Application_Watchdog (S3)	アプリケーションがアプリケーション ウォッチドッグの提供に失敗した場合、エラー レベル TTOST.EL を‘11’に設定します。アプリケーション ウォッチドッグは TTOCF.AWL を介して設定されます。TTOST レジスタを読み出すことで提供されます。ウォッチドッグが間に合わなかった場合、TTOST.AWE ビットと割込みフラグ TTIR.AW がセットされ、すべての TTCAN 通信が停止され、M_TTCAN はバス監視モード (CCCR.MON が'1') に設定されます。
CAN_Bus_Off (S3)	CAN_Bus_Off 状態になると、エラーレベル TTOST.EL が 11 になります。CAN_Bus_Off 状態は、PSR.BO = 1 および CCCR.INIT = 1 で通知されます。
Scheduling_Error_2 (S2)	1 つの Tx_Trigger の MSC が 7 に達した場合、エラー レベル TTOST.EL を'10'に設定します。また、割込みフラグ TTIR.SE2 がセットされます。TTOST.EL は、直前のマトリックス サイクルで Tx_Trigger の MSC が 7 でなかった場合、マトリックス サイクルの開始時に 00 にリセットされます。
Tx_Overflow (S2)	TX カウントが Tx_Trigger TTMLM.ENTT の予想数以上で、Tx_Trigger イベントが発生した場合、エラー レベル TTOST.EL を'10'に設定します。また、割込みフラグ TTIR.TXO がセットされます。TTOST.EL は、新しいマトリックス サイクルの開始時に TX カウントが TTMLM.ENTT 以下になると 00 にリセットされます。
Scheduling_Error_1 (S1)	1 マトリックス サイクル内であれば、エラーレベル TTOST.EL を'01'に設定します。(排他的タイムウィンドウの) すべてのトリガーメモリエレメントの最大 MSC と最小 MSC の差が 2 より大きい場合、または排他的 Rx_Trigger の MSC の 1 つが 7 に達した場合。また、割込みフラグ TTIR.SE1 がセットされます。1 マトリックス サイクル内にこれらの条件のいずれもが有効でない場合、TTOST.EL は 00 にリセットされます。
Tx_Underflow (S1)	新しいマトリックス サイクルの開始時に、TX カウントが予想される Tx_Triggers TTMLM.ENTT 数より少ない場合、エラーレベル TTOST.EL を'01'に設定します。また、割込みフラグ TTIR.TXU がセットされます。TTOST.EL は、新しいマトリックスの開始時に TX カウントが TTMLM.ENTT 以上になると 00 にリセットされます。

25.5.7 TTCAN メッセージ処理

25.5.7.1 参照メッセージ

ポテンシャル タイム マスターの場合、参照メッセージの識別子は TTRMC.RID を介して設定されます。参照メッセージの送信に専用の TX バッファは必要ありません。参照メッセージが送信されると、最初のデータ バイト (TTCAN レベル 1) と最初の 4 つのデータ バイト (TTCAN レベル 0 とレベル 2) が FSE から提供されます。

ペイロード選択参照メッセージ TTRMC.RMPS が設定されている場合、参照メッセージのペイロードの残り (レベル 1: バイト 2～8、レベル 0 とレベル 2: バイト 5～6) は TX バッファ 0 から取り出されます。この場合、メッセージバッファ 0 のデータ長 DLC コードが使用されます。

表 233 参照メッセージと共に送信されるデータ バイト数

TTRMC.RMPS	TXBRP.TRP0	レベル 0	レベル 1	レベル 2
0	0	4	1	4
0	1	4	1	4
1	0	4	1	4
1	1	4 + MB0	1 + MB0	4 + MB0

レベル 1 で参照メッセージと共に追加のペイロードを送信するためには、DLC > 1 を設定する必要があります。レベル 0 とレベル 2 では DLC>4 が必要です。さらに、メッセージ バッファ 0 の送信要求保留ビット TXBRP.TRP0 がセットされていなければいけません (表 233 を参照)。参照メッセージの開始時に TXBRP.TRP0 が設定されていない場合、参照メッセージは FSE から供給されたデータ バイトのみで送信されます。

参照メッセージの受入フィルタリングには、参照識別子 TTRMC.RID が使用されます。

25.5.7.2 メッセージ受信

メッセージの受信は、イベント駆動型 CAN 通信と同様に、2 つの RX FIFO を介して行われます。

MSC は対応するトリガー メモリ エレメントの一部で、コンフィギュレーション中にゼロに初期化されなければいけません。M_TTCAN が同期状態 In_Gap または In_Schedule にある間に更新されます。更新はメッセージの Rx_Trigger で行われます。この時点で、この基本サイクルで受信した最新のメッセージが、どの受入フィルター エレメントに一致しているかがチェックされます。一致したフィルター番号は、受入フィルターの結果として保存されます。これが、このトリガーメモリ エレメントで定義されたフィルター番号と同じであれば、MSC は 1 つデクリメントされます。受入フィルターの結果が、このフィルター エレメントに対して定義されたものと同じフィルター番号でない場合、または受入フィルターの結果がクリアされた場合、MSC は 1 つインクリメントされます。各 Rx_Trigger と各サイクルの開始時に、最後の受入フィルターの結果はクリアされます。

Rx_Trigger のタイム マークは、ターゲット メッセージの受信と受入フィルタリングが確実に完了する値に設定してください。これは、RAM のアクセス時間とフィルターリストの順番を考慮する必要があります。Rx_Triggers に使用するフィルターは、リストの先頭に配置することを推奨します。参照メッセージに Rx_Trigger を使うことは推奨されません。

25.5.7.3 メッセージ送信

タイム トリガー メッセージ伝送のために、M_TTCAN は 32 の専用 TX バッファを供給する (379 ページの「TTCAN コンフィギュレーション」を参照)。TX FIFO または TX キューは、M_TTCAN がタイムトリガー動作 (TTOCF.OM = 01 または 10) に設定されている場合は使用できません。

トリガーメモリ内の各 Tx_Trigger は、特定のメッセージを含む特定の TX バッファを指します。特定の TX バッファに対して複数の Tx_Trigger が存在する場合があります。これは、その TX バッファが基本サイクルまたはマトリックス サイクル内で複数回送信されるメッセージを含む場合です。

アプリケーションプログラムは、サイクルタイムに同期して、定期的にデータを更新する必要があります。ホスト CPU は、部分的に更新されたメッセージが送信されないようにする必要があり、ホストは次のように処理を進めます。

Tx_Trigger_Single/Tx_Trigger_Merged/Tx_Trigger_Arbitration:

- TXBTO を読み出して、前回の送信が完了したかどうかをチェック
- TX バッファのコンフィギュレーションおよび/またはペイロードの更新
- TX バッファリクエスト保留ビットを設定するために、追加リクエストを発行

Tx_Trigger_Continous:

- キャンセル要求を発行し、TX バッファ要求保留ビットをリセット
- TXBCF を読み出すことにより、キャンセルが終了したかどうかを確認
- TX バッファ設定および/またはペイロードの更新
- TX バッファリクエスト保留ビットを設定するために、追加リクエストを発行

対応する Tx_Trigger とともに格納されるメッセージ MSC は、送信の成功に関する情報を提供します。

MSC は、対応する送信有効ウィンドウ内で CAN バスがアイドル状態でなかったため送信を開始できなかった場合、またはメッセージを開始したが正常に完了できなかった場合に 1 インクリメントされます。MSC は、メッセージが正常に送信されたとき、あるいは、送信可能ウィンドウ内でメッセージを開始できますが、送信が無効になっている (エラー レベル S2 の M_TTCAN、あるいはホストがこの特定のメッセージを無効にしている) ために開始されなかったときに、1 デクリメントされます。

TX バッファは動的に管理されるかもしれません。異なる識別子を持つ複数のメッセージが同じ TX バッファ エレメントを共有するかもしれません。この場合、ホストは TXBRP をチェックすることによって、TX バッファエレメントを再設定するための送信要求が保留中でないことを確認する必要があります。

送信要求が保留されている TX バッファを更新する必要がある場合、ホストはまずキャンセル要求を発行し、更新を開始する前に TXBCF を読んでキャンセルが完了したかどうかをチェックする必要があります。

TX ハンドラは、メッセージ RAM から中間出力バッファにメッセージを転送します。トリガー エレメントは、送信ウィンドウの開始を定義する Tx_Trigger エレメントの直前にアクティブになります。転送時間中と転送時間後は、送信メッセージは更新されず、その TXBRP ビットは変更されないかもしれません。この転送時間を制御するために、Tx_Trigger の前に追加のトリガーエレメントを置けます。例えば、Time_Base_Trigger は、他のアクションを引き起こしません。Tx_Trigger とその前のトリガーとの間のタイムマークの差は、他のモジュールからの RAM アクセス負荷が高い場合でも、TX ハンドラがメッセージ RAM から 4 ワードを読み出せることを保証するのに十分な大きさでなければいけません。

25.5.7.3.1 排他的タイムウィンドウでの送信

送信は、サイクルタイムが Tx_Trigger_Single または Tx_Trigger_Continous のタイムマークに達したときにタイムトリガーで開始されます。バス上では、他のノードからのメッセージとのアービトレーションはありません。MSC は送信試行の結果に応じて更新されます。Tx_Trigger_Single によって開始された送信が成功した後、それぞれの TX Buffer Request Pending ビットはリセットされます。送信成功後、Tx_Trigger_Continous によって、それぞれの TX Buffer Request Pending ビットはセットされたままです。外乱により送信が成功しなかった場合、次に Tx_Trigger の 1 つがアクティブになると、送信は繰り返されます。

25.5.7.3.2 タイムウィンドウのアービトレーションにおける送信

送信は、サイクルタイムが Tx_Trigger_Arbitration のタイムマークに達すると、タイムトリガーで開始されます。複数のノードが同時に送信を開始することがあります。この場合、メッセージは他のノードからのメッセージとアービトレーションする必要があります。MSC は更新されません。送信に失敗した場合 (アービトレーションが失われた、または妨害された)、次に Tx_Trigger の 1 つがアクティブになると、その送信は繰り返されます。

25.5.7.3.3 マージされたアービトレーション タイムウィンドウでの送信

マージされたアービトレーション タイムウィンドウの目的は、複数のノードが限られた数のフレームを送信できるようにすることで、これらのフレームは即座にシーケンスで送信され、CAN アービトレーションによって与えられた順序で送信されます。単一ノードによるバースト送信を意図したものではありません。このタイムウィンドウ内はノードが排他的にアクセスできないため、要求された送信がすべて成功するとは限りません。

アービトレーションを失った、あるいはエラーによって妨害されたメッセージは、同じマージされたアービトレーション タイムウィンドウ内で再送信される可能性があります。。対応する送信要求待ちフラグが TX キャンセル成功によってリセットされた場合、再送は開始されません。

単一送信ウィンドウでは、TX ハンドラは、トリガーエレメントのメッセージ番号で示されるメッセージを送信します。マージされたアービトレーション タイムウィンドウでは、トリガー リストから最大 3 つのメッセージ番号を扱えます。それらの送信は、トリガーリストで定義された順序で試みられます。最初のメッセージが送信される前に、4 つ目のメッセージのタイムマークが読み出された場合 (あるいはホストがキャンセルした場合)、4 つ目のリクエストは無視されます。

マージされたアービトレーション タイムウィンドウ内の送信はタイムトリガーされません。メッセージの送信は、そのタイムマークより前に開始されるかもしれないし、バスがアイドル状態でなければタイムマークより後に開始されるかもしれません。

マージされたアービトレーション タイムウィンドウ内で特定のノードによって送信されたメッセージは、Tx_Trigger の順に開始されます。そのため、CAN の優先度が低いメッセージは、競合するバストラフィックがある場合、優先度の高い後続メッセージの送信を妨げる可能性があります。これは、トリガーリストの設定において考慮してください。連続する Tx_Trigger の間に Time_Base_Trigger を配置することで、対応する TX バッファのデータを更新するまでの時間を定義できます。

25.5.8 TTCAN 割込みとエラー処理

TT Interrupt Register TTIR は 4 つのセグメントから構成されます。各割込みは、TT Interrupt Enable レジスタ TTIE の対応するビットによって個別に有効にできます。フラグはホストがクリアするまでセットされたままです。フラグは、対応するビット位置に'1'を書き込むことでクリアされます。

最初のセグメントは、CER、AW、WT、IWT のフラグで構成されます。各フラグは、CAN 通信が停止する致命的なエラー状態を示します。このエラー状態は、通信を再開する前に M_TTCAN モジュールの再設定を必要としますが、これは IWT の例外です。

第 2 セグメントは、ELC、SE1、SE2、TXO、TXU、および GTE フラグで構成されます。各フラグは、CAN 通信が妨害されたエラー状態を示します。CAN バス上の障害など、過渡的な障害によって発生した場合は、TTCAN プロトコルの障害処理によって処理されるため、アプリケーションプログラムによる介入は必要ありません。

第 3 セグメントは、GTD、GTW、SWE、TTMI、RTMI のフラグで構成されます。最初の 2 つのフラグは、アプリケーション プログラムによる反応を必要とするグローバル タイム イベント (レベル 0 とレベル 2 のみ) によって制御されます。ストップウォッチ イベントでは、内部時間値がキャプチャされます。トリガー タイムマーク割込みは、特定のタイム ベース トリガーに達したことをアプリケーションに通知します。レジスタ タイムマーク割込みは、TTOCN.TMC によって参照される時間 (サイクル、ローカル、またはグローバル) が、タイムマーク TTTMK.TM と等しいことを知らせます。隙間を埋めるのにも使えます。

第 4 セグメントは、SOG、CSM、SMC、SBC のフラグで構成されます。これらのフラグは、アプリケーション プログラムを通信スケジュールに同期させる手段を提供します。

25.5.9 レベル 0

TTCAN レベル 0 は ISO11898-4 の一部ではありません。この動作モードにより、TTCAN レベル 2 でキャリブレーションされたグローバル タイム ベースを維持するハードウェアが、ISO 11898-1:2015 に準拠するイベント駆動 CAN でも使用できるようになります。

レベル 0 動作は TTOCF.OM = 11 で設定されます。このモードでは、M_TTCAN はイベント駆動型の CAN 通信で動作します。TTOCF.GEN の設定は無視されます。外部イベント同期動作はレベル 0 では使用できません。同期されたタイムベースは、参照メッセージの送信によって維持されます。

レベル 0 では、トリガーメモリはアクティブではないので、設定する必要はありません。サイクルタイムが TTOCF.IRTO × 0x200 に達すると、タイムマーク割込みフラグ (TTIR.TTMI) がセットされます。メッセージ バッファ 0 に対する送信要求を設定するようにホストにリマインドします。ウォッチ トリガー割込みフラグ (TTIR.WT) は、サイクルタイムが 0xFF00 に達したときにセットされます。これらの値は、安定したクロック キャリブレーションに十分なマージンを持たせるために選ばれました。それ以上の TT エラーチェックはありません。レジスタ タイム マーク割込み (TTIR.RTMI) も可能です。

参照メッセージは、レベル 2 操作用に設定されます。受信した参照メッセージは、レジスタ TTRMC で設定された識別子によって認識されます。参照メッセージの送信には、メッセージ バッファ 0 のみを使用できます。ノードは、ホストがメッセージバッファ 0 に対して送信要求を設定するたびに参照メッセージを送信します。参照トリガーオフセットはありません。

レベル 0 の動作は、以下の方法で設定します。

- TTRMC
- EVTP、AWL、GEN を除く TTOCF
- ENTT、TXEW を除く TTMLM
- TURCF

レベル 0 の動作は、以下の方法で制御されます。

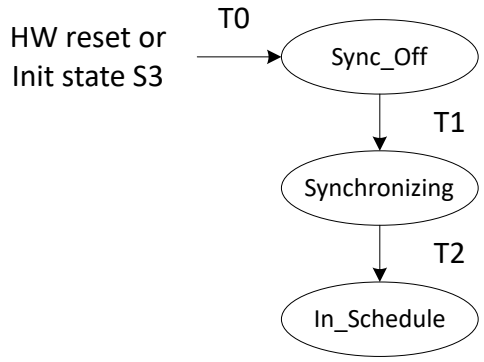
- NIG、TMG、FGP、GCS、TTMIE を除く TTOCN
- TTGTP
- TTTMK
- ビット CER、AW、IWT、SE2、SE1、TXO、TXU、SOG を除く TTIR (機能なし)
- TTIR - 以下のビットが機能変更:
  - トリガーメモリで定義されていない TTMI - サイクルタイム TTOCF.IRTO × 0x200 で起動
  - トリガーメモリで定義されていない WT - サイクルタイム 0xFF00 で起動

レベル 0 の動作は、以下の方法で通知されます。

- TTOST は AWE、WFE、GSI、GFI、RTO のビットを除く (機能なし)

25.5.9.1 同期

図 211 に、TTCAN レベル 0 動作における状態と状態遷移を示します。レベル 0 は In_Gap の状態を持ちません。



T0: transition condition always taking prevalence

T1: Init state left, cycle time is zero

T2: at least two successive reference messages observed
(last reference message did not contain a set Disc_Bit bit)

図 211　　レベル 0 スケジュール同期ステートマシン

25.5.9.2　エラー レベルの処理

レベル 0 動作中に以下のエラー条件が発生します。

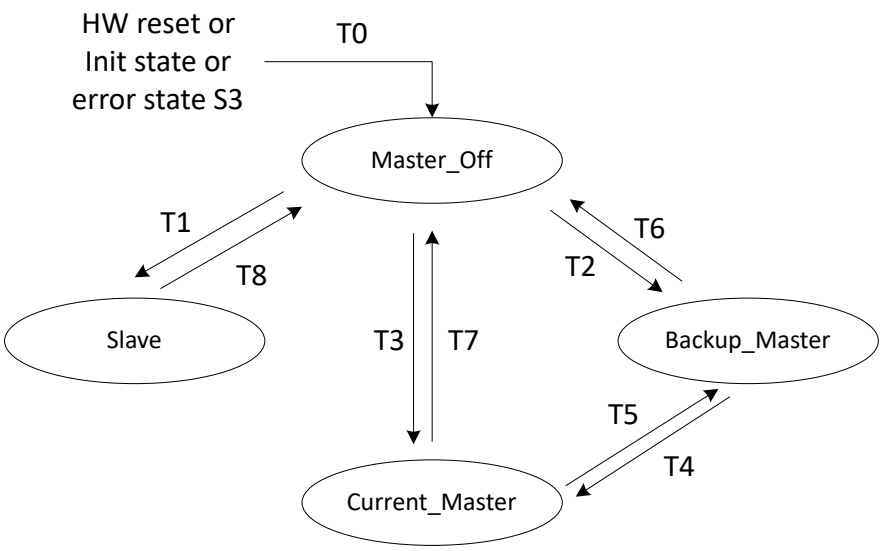
- Watch_Trigger_Reached (S3)、サイクルタイム 0xFF00 到達
- CAN_Bus_Off (S3)

S1 エラーと S2 エラーは発生しないため、エラーレベルは S0 (エラーなし) と S3 (重大なエラー) の間でしか切り替えられません。TTCAN レベル 0 では、S3 エラーの扱いが異なります。S3 エラーになると、TTOST.SYS と TTOST.MS がリセットされ、割込みフラグ TTIR.GTE と TTIR.GTD がセットされます。

S3 (TTOST.EL = 11) が入力されると、TTCAN レベル 1 およびレベル 2 とは逆に、バス監視モードにはなりません。S3 エラーレベルは、次の参照メッセージの送信 (タイムマスター) または受信 (タイムスレーブ) の後に自動的に残されます。

25.5.9.3　マスタースレーブ関係

図 212 に、TTCAN レベル 0 におけるマスター スレーブ関係を示します。S3 エラーの場合、M_TTCAN は Master_Off 状態に戻ります。



T0: transition condition always taking prevalence
T1: reference message observed when not potential master
T2: reference message observed with master priority != own master priority, error state != S3
T3: reference message observed with master priority = own master priority, error state != S3
T4: reference message observed with own master priority
T5: reference message observed with master priority higher than own master priority
T6: error state S3
T7: error state S3
T8: error state S3

図 212　　レベル 0 マスターとスレーブの関係

25.5.10　外部タイムスケジュールとの同期

この機能は、M_TTCAN のスケジュールのフェーズを外部のスケジュール (例えば、第 2 の TTCAN ネットワークのスケジュール) に同期させるために使用されます。M_TTCAN が現在のタイムマスター (TTOST.MS = 11) の場合にのみ適用されます。

外部同期は TTOCN.ESCN ビットによって制御されます。TTOCN.ESCN が設定されている場合、内部イベントトリガーピンの立ち上りエッジで、M_TTCAN は実際のサイクルタイムを TTGTP.CTP で設定されたターゲット位相値と比較します。

TTOCN.ESCN を設定する前に、ホストは例えば TTCAN ギャップ コントロール (384 ページの TTCAN ギャップ コントロール参照) を使用して、2 つのタイム スケジュールの位相を合わせる必要があります。ホストが TTOCN.ESCN を設定すると、TTOST.SPL が設定されます。

サイクルタイムとトリガー時のターゲット位相値 TTGTP.CTP の差が 9NTU より大きい場合、位相ロックビット TTOST.SPL がリセットされ、割込みフラグ TTIR.CSM がセットされます。他のノードがタイムマスターになると、TTOST.SPL もリセットされます (TTIR.CSM はセットされます)。

TTOST.SPL と TTOCN.ESCN の両方がセットされ、サイクルタイムとターゲット位相値 TTGTP.CTP の差が 9NTU 以下であれば、位相ロックビット TTOST.SPL はセットされたままとなり、測定された差は、次に送信される基準メッセージで位相を調整するための基準トリガーオフセット値として使用されます。

注: 内部ピンの立ち上りエッジ検出は、各基本サイクルの開始時に有効になります。最初の立ち上りエッジがトリガーとなり、実際のサイクルタイムと TTGTP.CTP が比較されます。次の基本サイクルが始まるまで、それ以降のエッジはすべて無視されます。

25.6 設定手順

ここでは、M_TTCAN チャネルの設定手順の例を示します。

25.6.1 一般的なプログラムのフロー

これは M_TTCAN モジュールを設定する一般的なフローです。

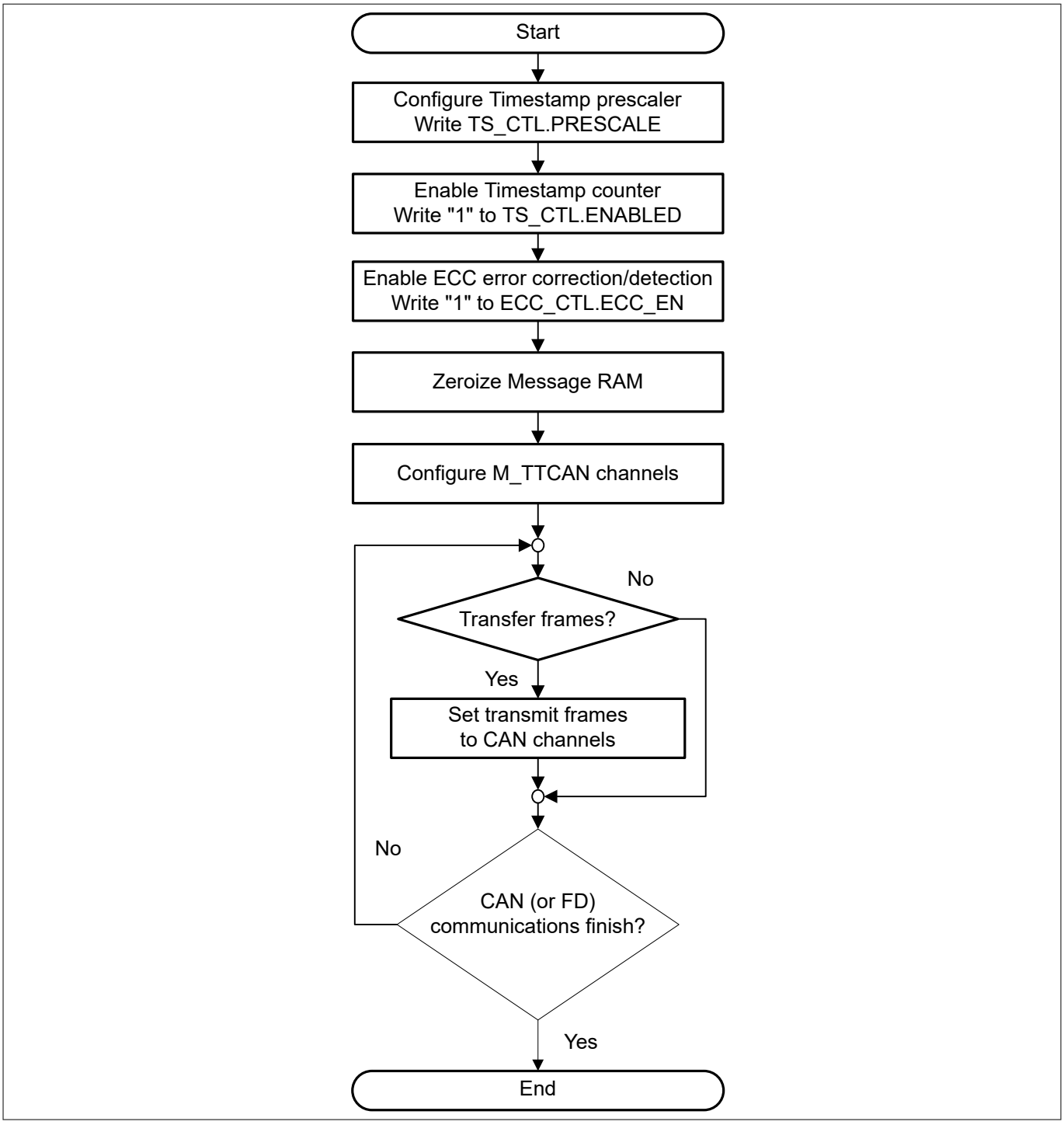


図 213　　一般的なプログラムのフロー

25.6.2　クロック停止要求

消費電力を削減するため、アプリケーションは図に示す手順で未使用の M_TTCAN チャネルへのクロック供給を停止します。

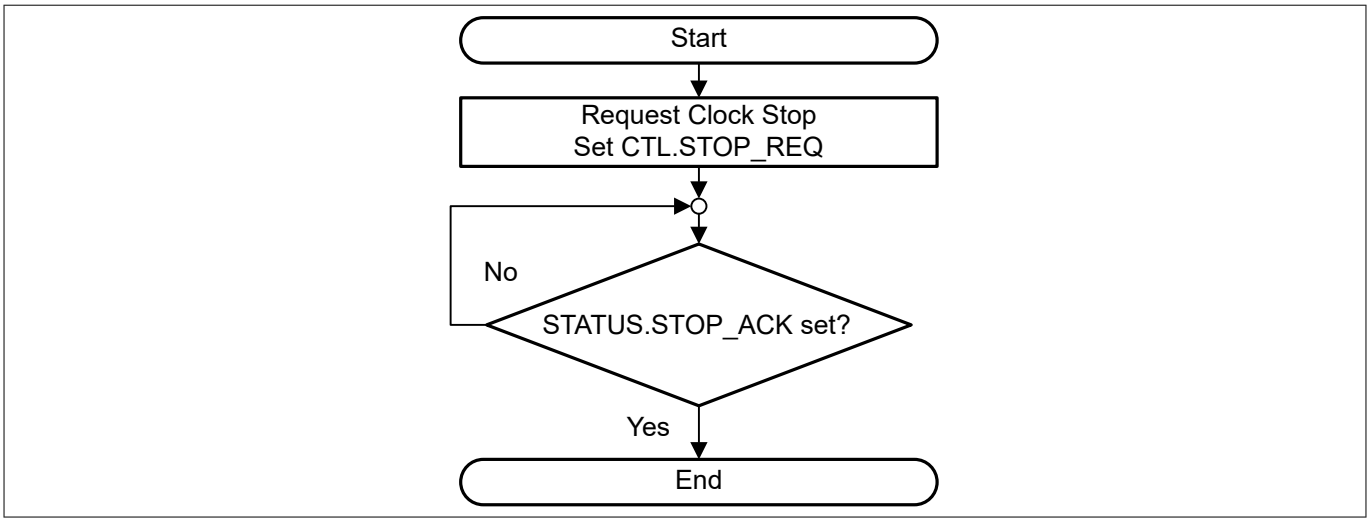


図 214　　クロック停止要求手順

クロック供給を再開するためには、CTL.STOP_REQ ビットをリセットします。

25.6.3　　MRAM オフ動作

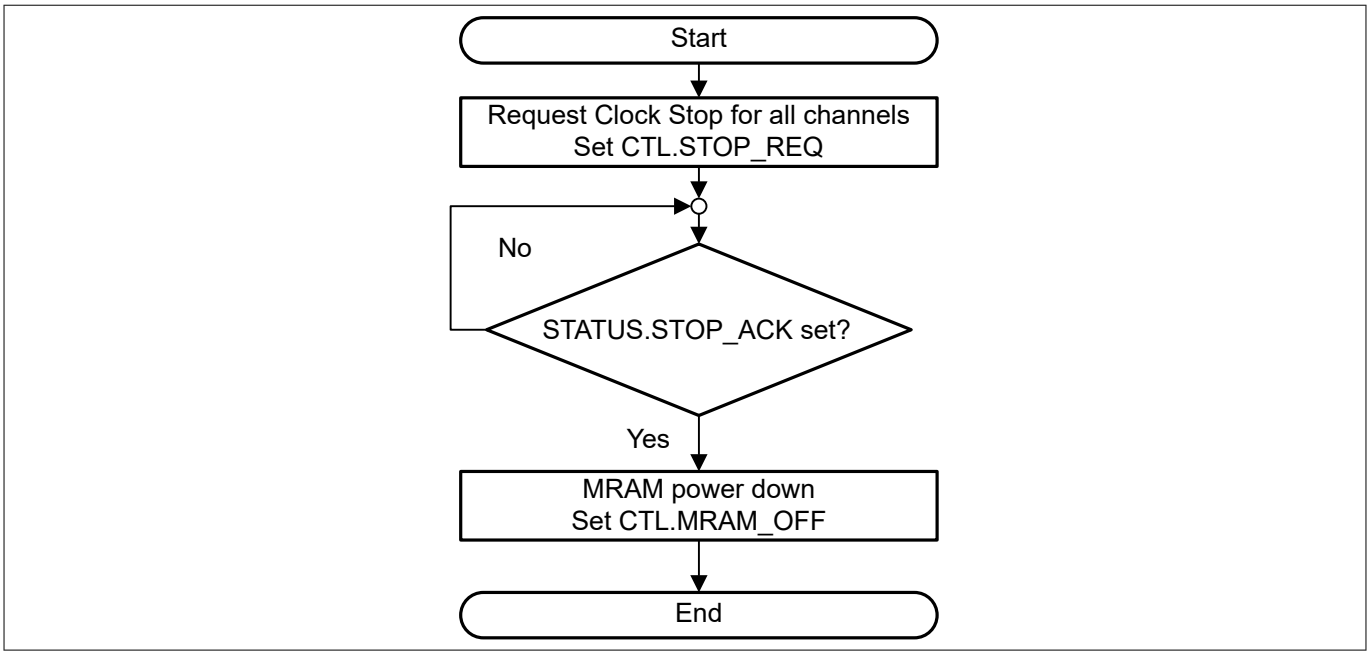


図 215　　メッセージ RAM オフ動作

25.6.4 MRAM オン動作

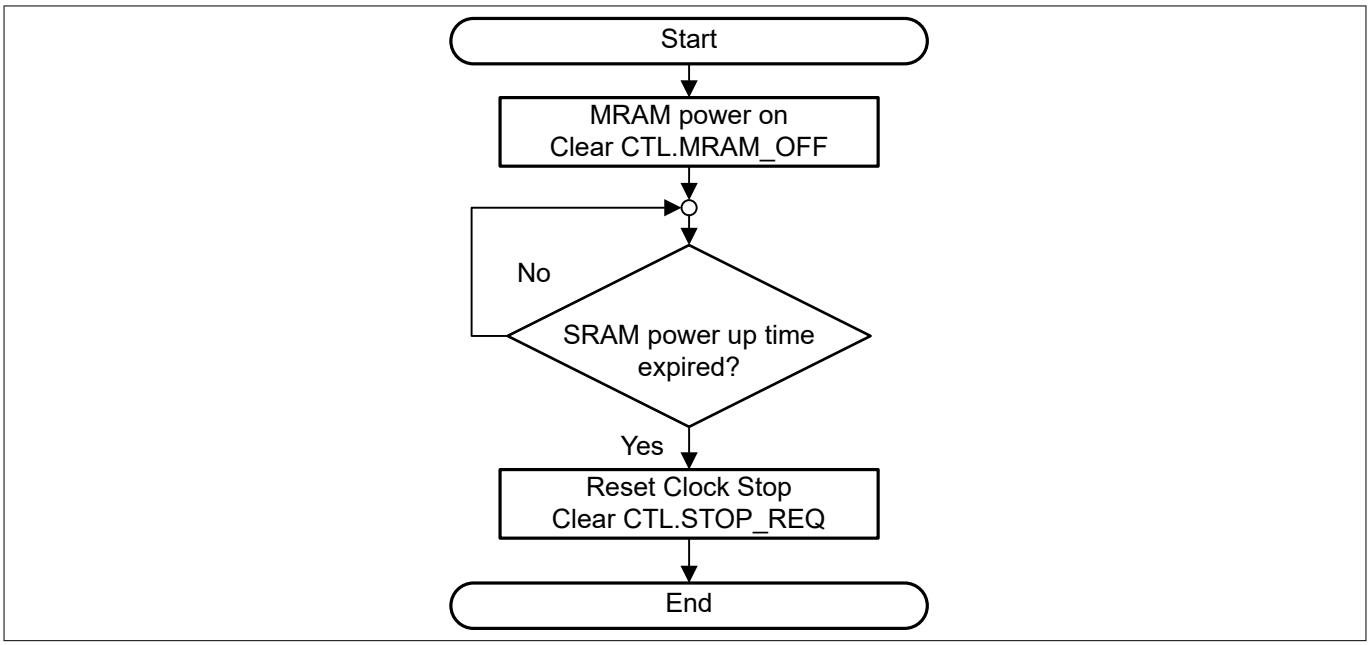


図 216 メッセージ RAM オン手順

MRAM を再びオンにした後、MRAM が使用できるようになるまで、つまり STOP_REQ がデアサートされるまで、ソフトウェアは一定のパワーアップ時間をおく必要があります。パワーアップ時間は、CPUSS.RAM_PWR_DELAY_CTL レジスタで指定されたシステム SRAM パワーアップ時間に相当します。

25.6.5 M_TTCAN チャネル固有の手順

ここでは、M_TTCAN チャネルのサンプル手順について説明します。図 217 に、一般的なプログラム フローを示します。

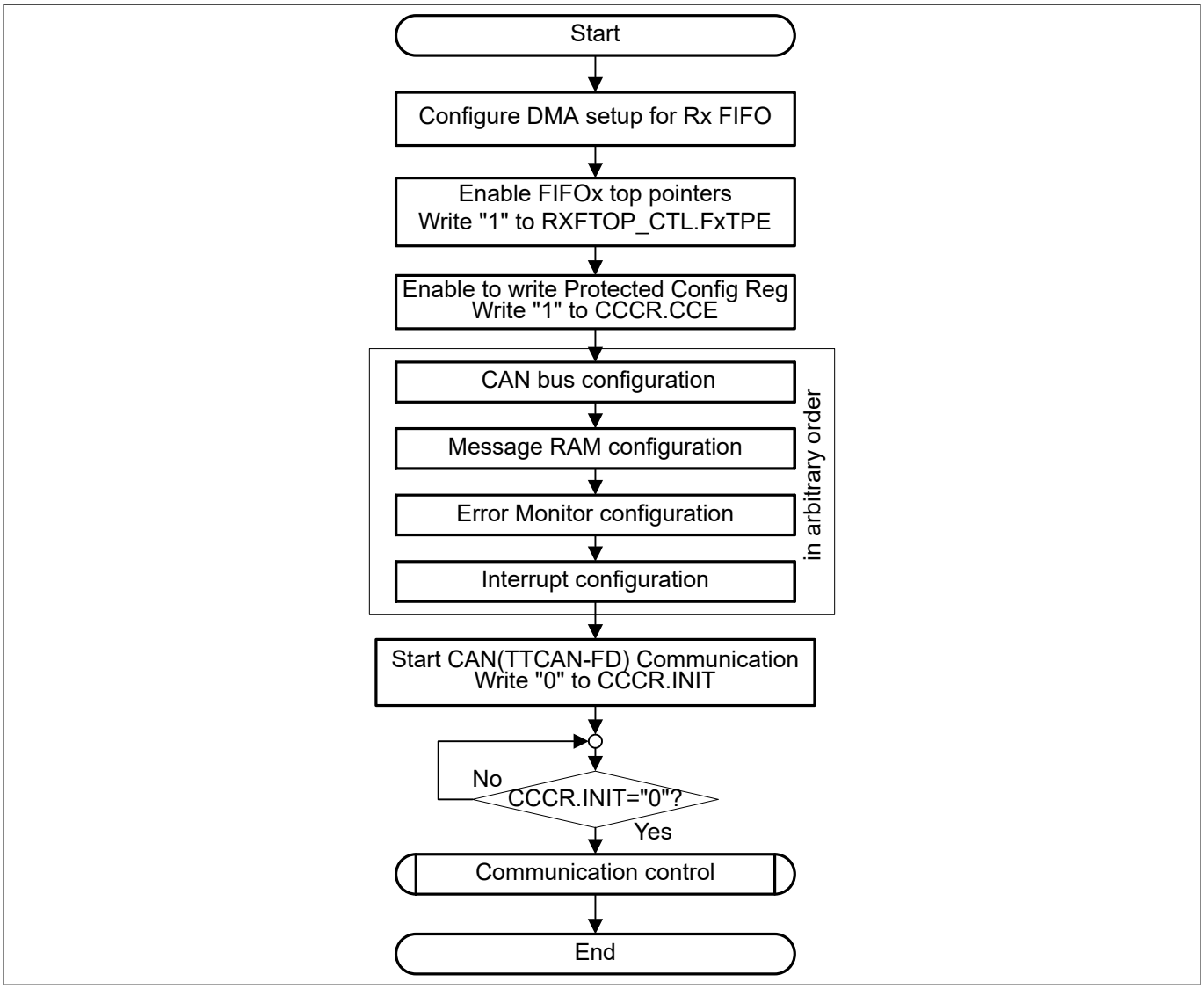


図 217　　チャネル固有の設定シーケンス

25.6.6 CAN バス設定

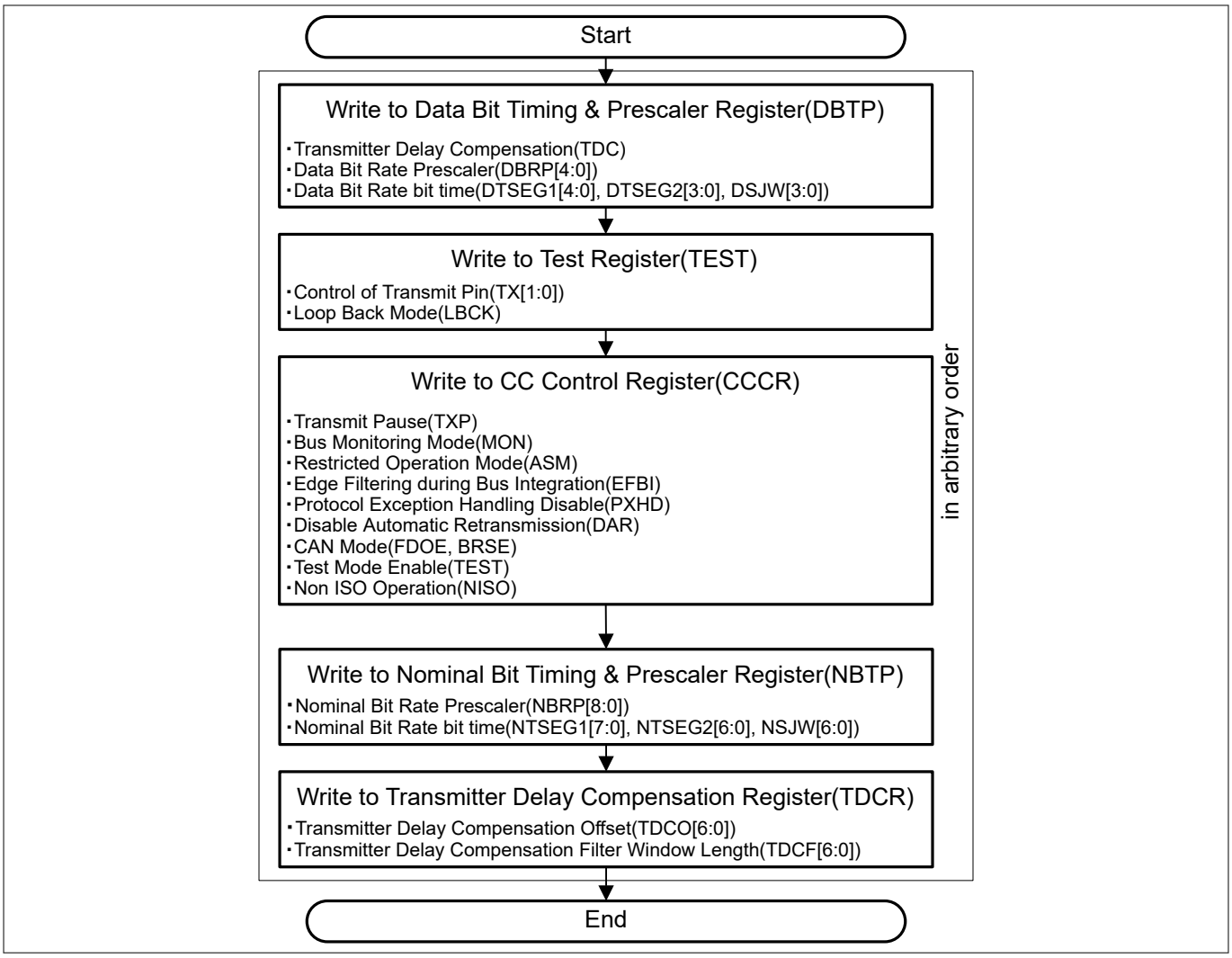


図 218 CAN バスに必要なコンフィギュレーション

25.6.7 メッセージ RAM のコンフィギュレーション

図 219 にメッセージ RAM コンフィギュレーションの概要を示します。

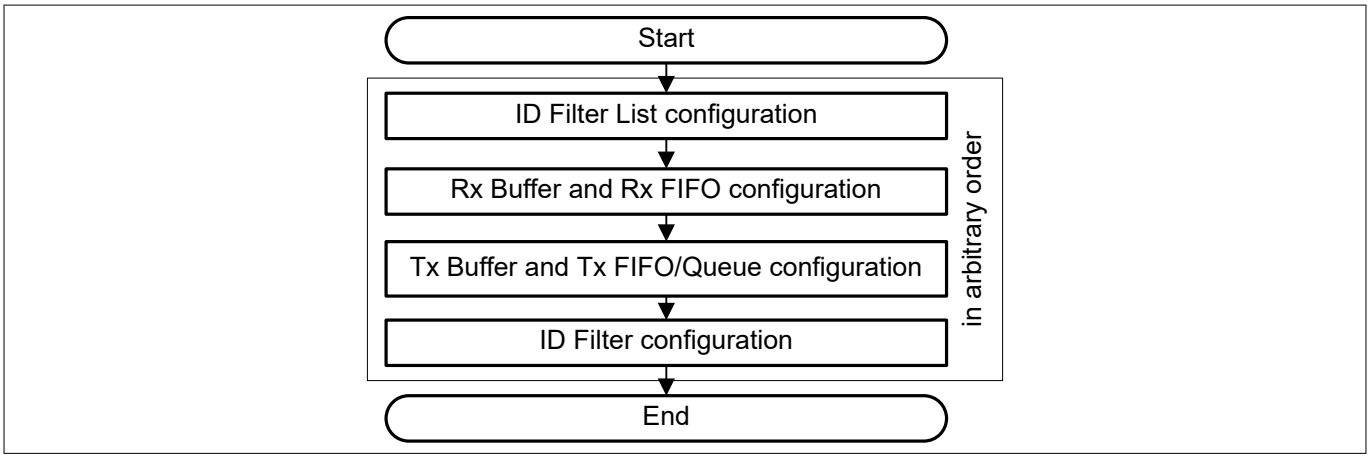


図 219 メッセージ RAM コンフィギュレーションの概要

以下の図に、概要に記載されている各コンフィギュレーションの詳細を示します。

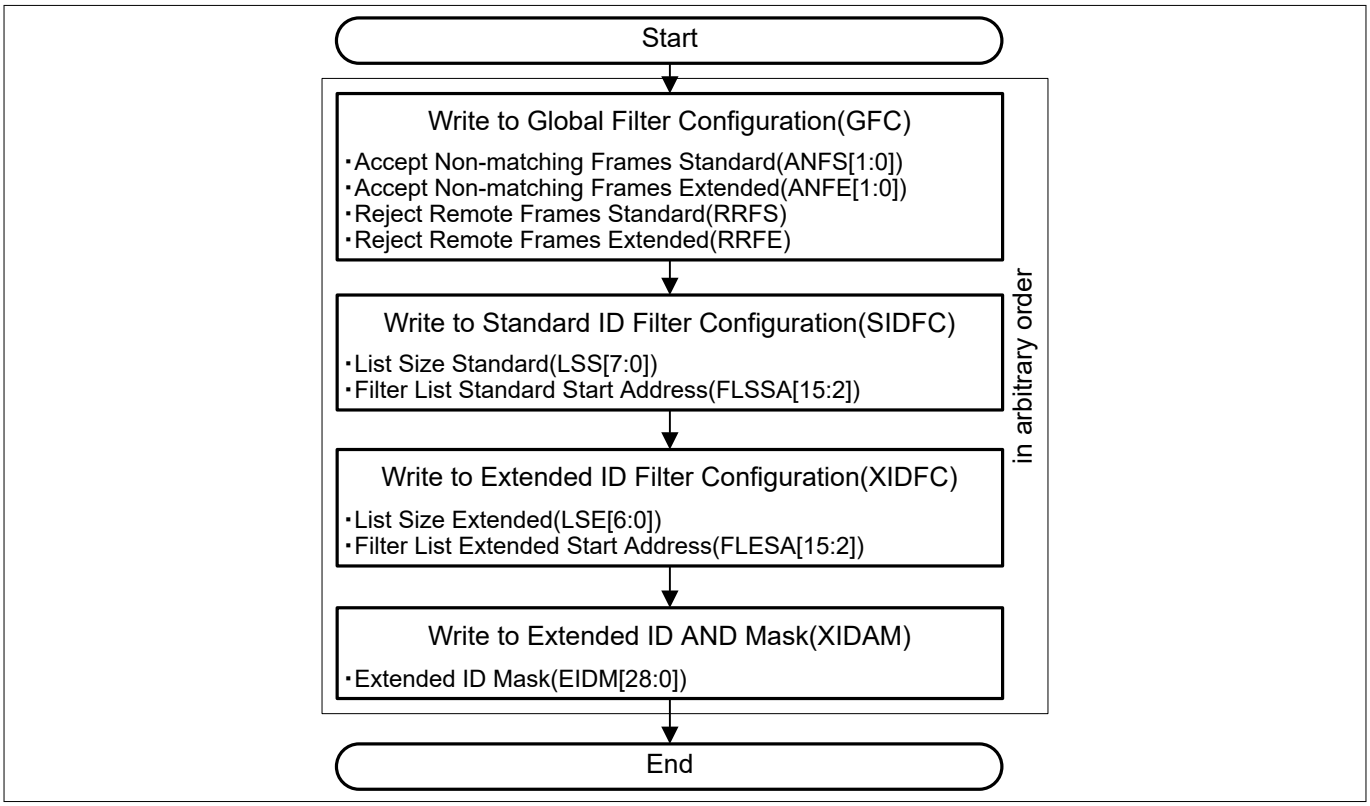


図 220　　ID フィルターリストのコンフィギュレーション

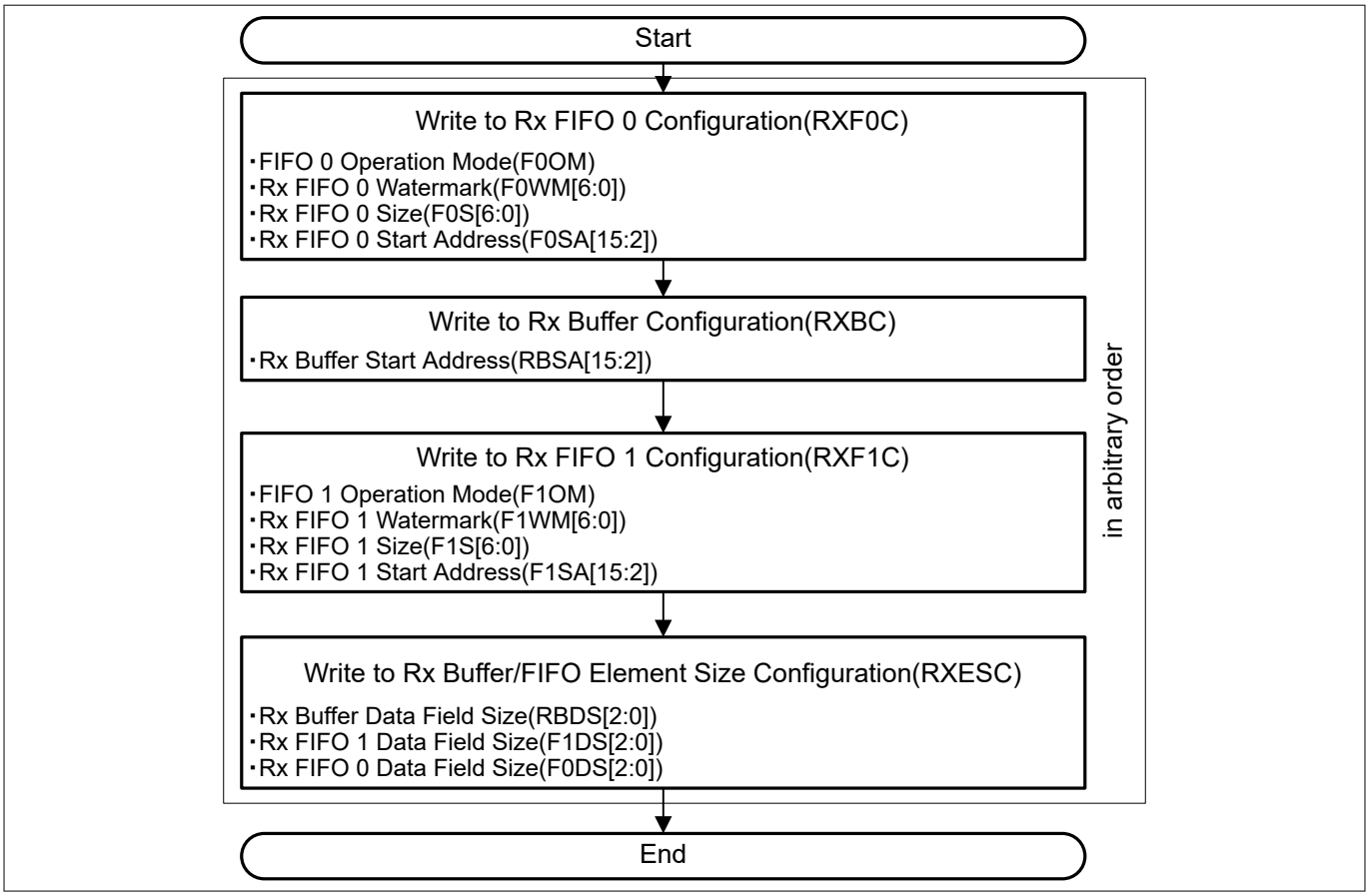


図 221　　受信 FIFO および受信バッファ コンフィギュレーション

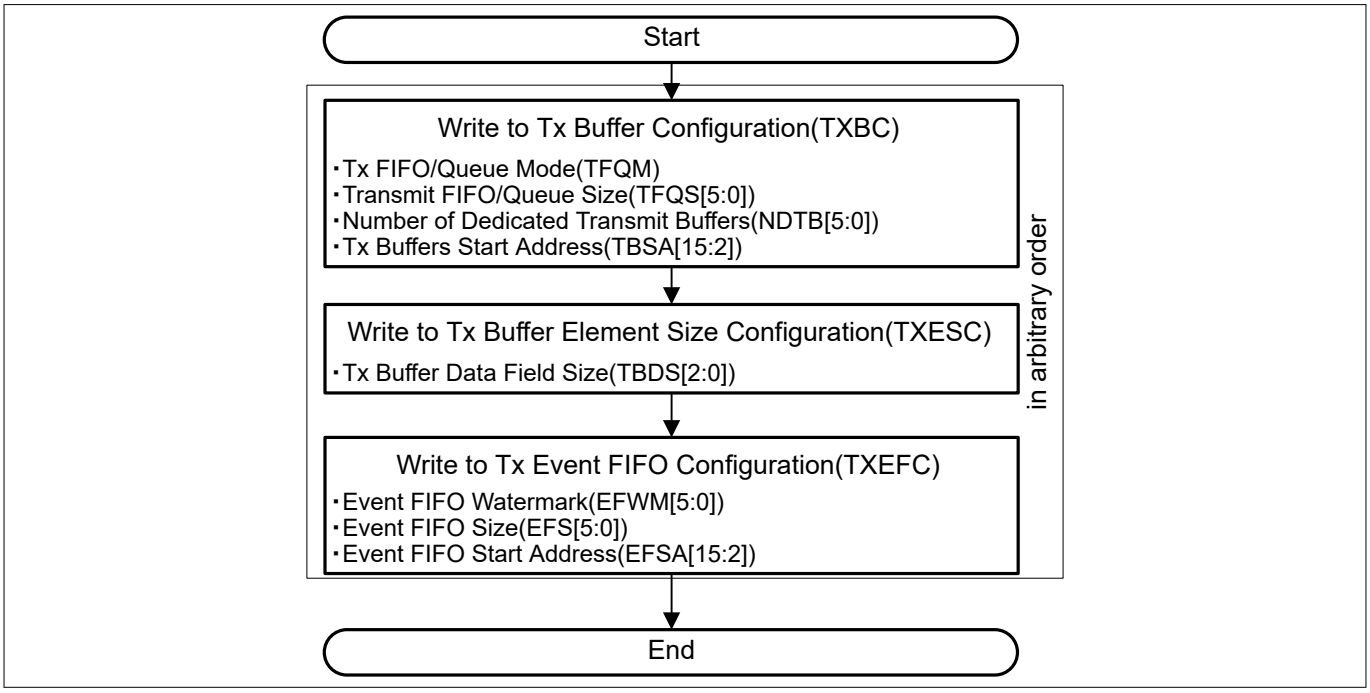


図 222　　TX バッファと TX FIFO/キューのコンフィギュレーション

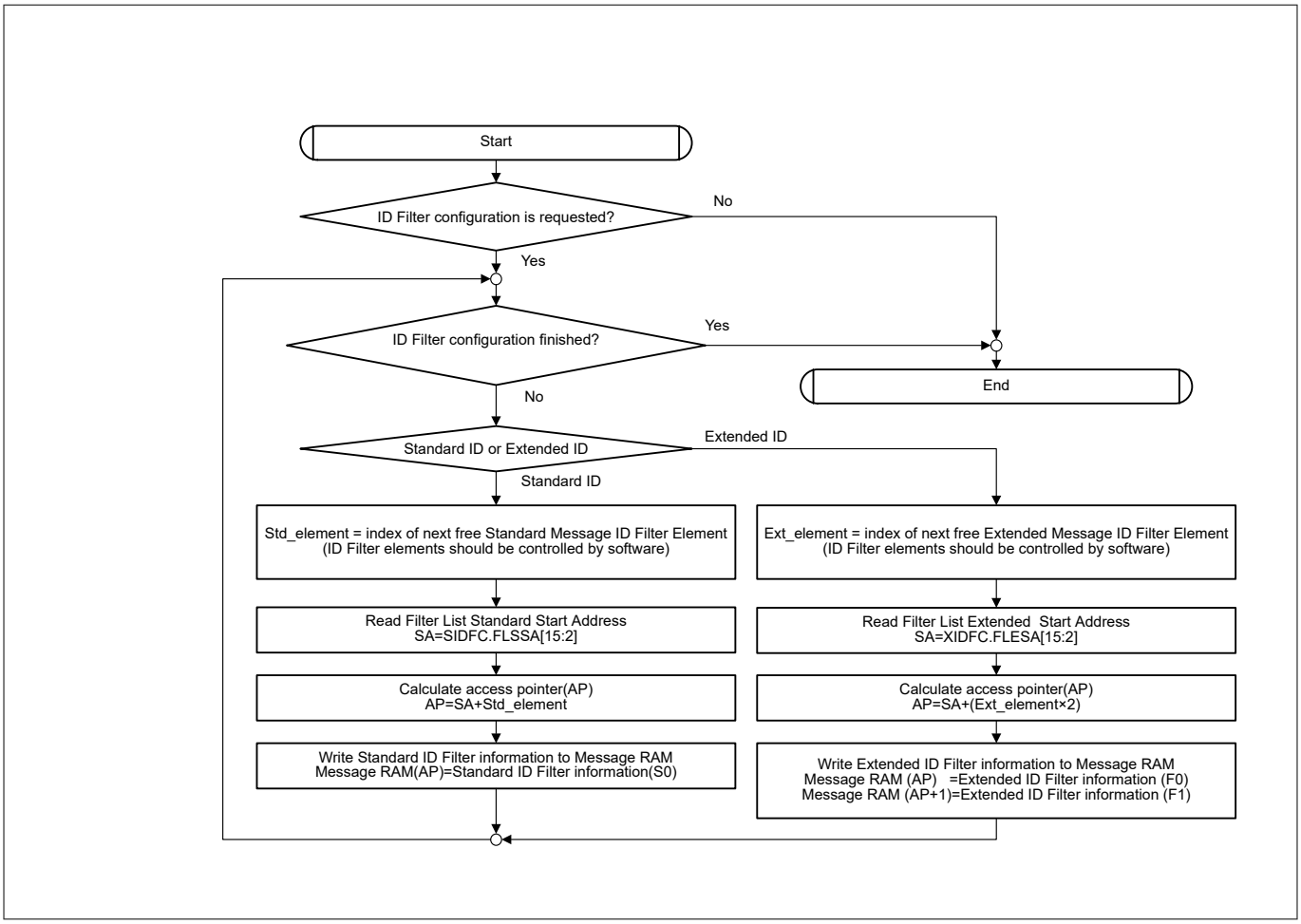


図 223 ID フィルターコンフィギュレーション

25.6.8 割込みコンフィギュレーション

図 224 に一般的な割込みコンフィギュレーションのフローを示します。

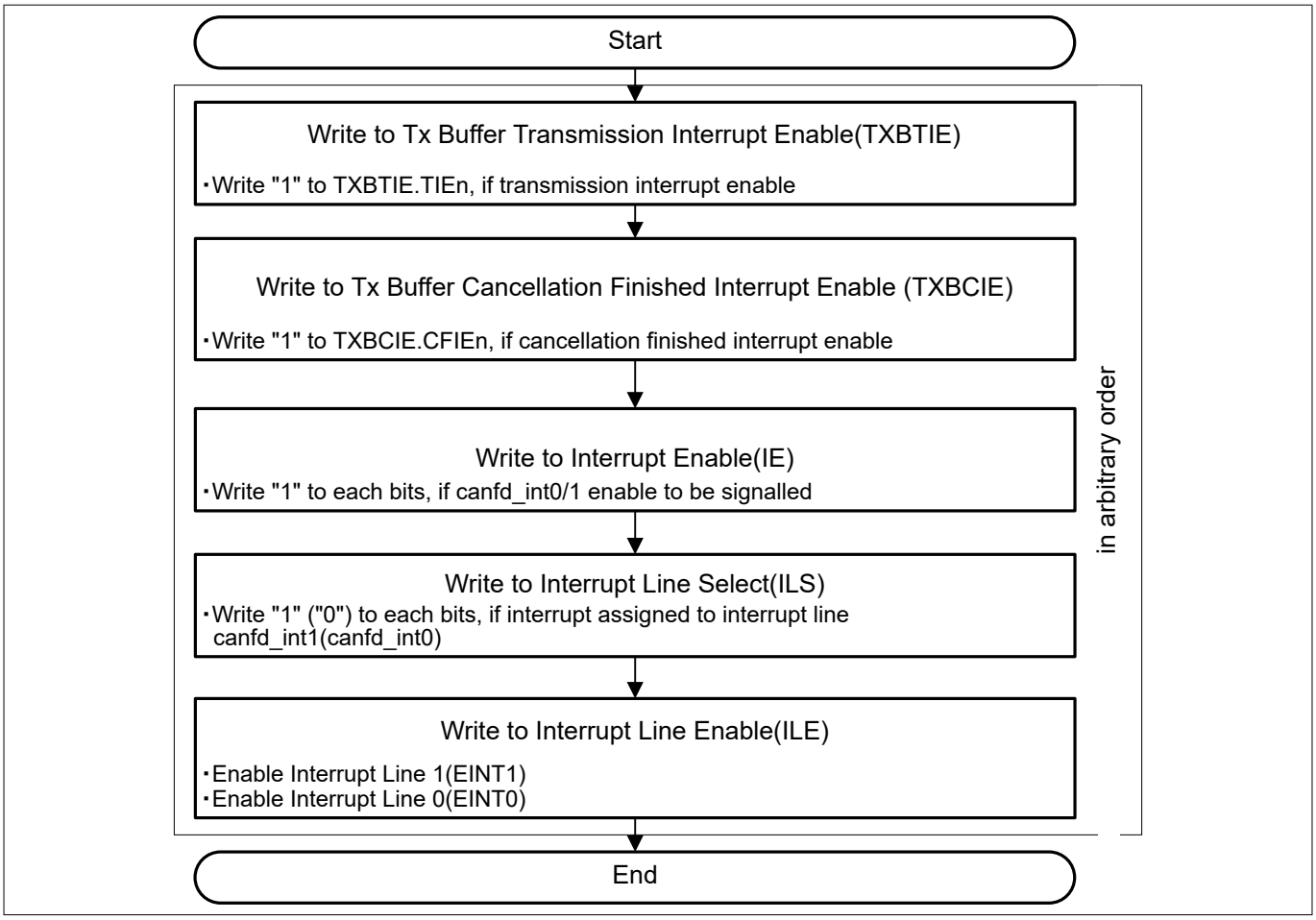


図 224　　割込みコンフィギュレーション

25.6.9　フレーム コンフィギュレーションの送信

図 225 に送信フレーム コンフィギュレーションを示します。

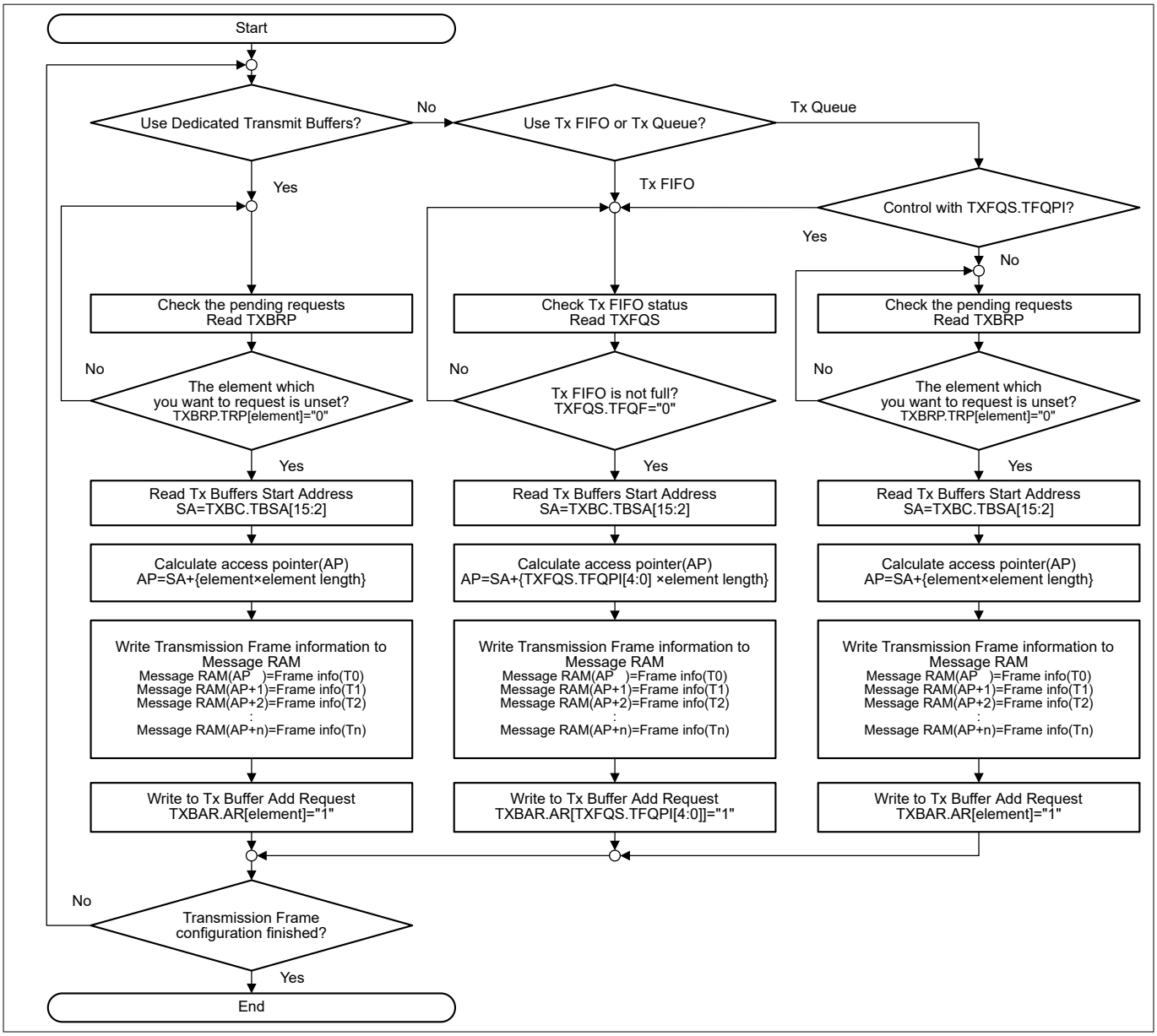


図 225　　フレーム コンフィギュレーションの送信

25.6.10　　割込み処理

図 226 に、一般的な割込み処理のフローチャートを示します。

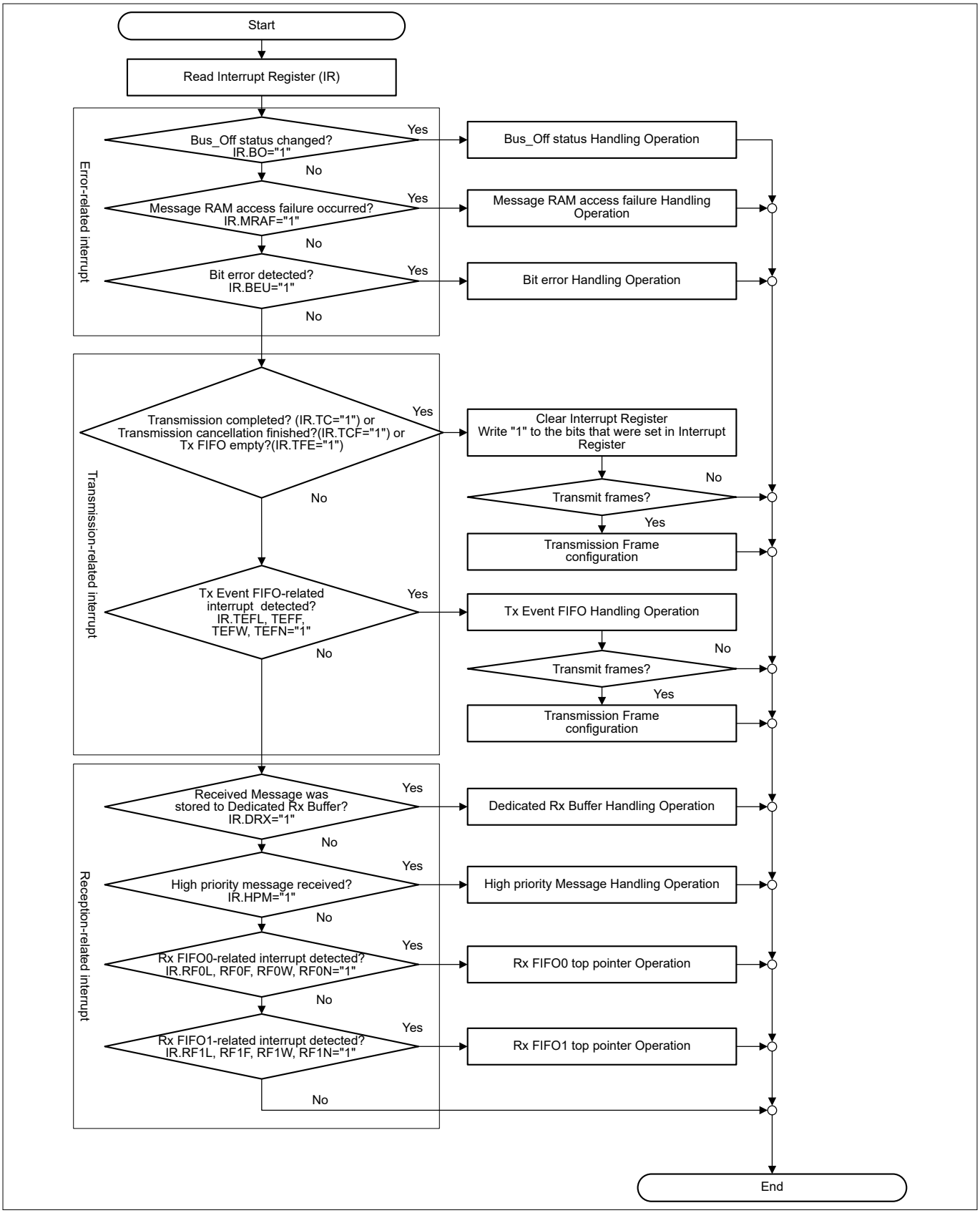

Start
Read Interrupt Register (IR)
Error-related interrupt
Bus_Off status changed? IR.BO="1"
Yes
No
Bus_Off status Handling Operation
Message RAM access failure occurred? IR.MRAF="1"
Yes
No
Message RAM access failure Handling Operation
Bit error detected? IR.BEU="1"
Yes
No
Bit error Handling Operation
Transmission-related interrupt
Transmission completed? (IR.TC="1") or Transmission cancellation finished?(IR.TCF="1") or Tx FIFO empty?(IR.TFE="1")
Yes
No
Clear Interrupt Register Write "1" to the bits that were set in Interrupt Register
Transmit frames?
No
Yes
Transmission Frame configuration
Tx Event FIFO-related interrupt detected? IR.TEFL, TEFF, TEFW, TEFN="1"
Yes
No
Tx Event FIFO Handling Operation
Transmit frames?
No
Yes
Transmission Frame configuration
Reception-related interrupt
Received Message was stored to Dedicated Rx Buffer? IR.DRX="1"
Yes
No
Dedicated Rx Buffer Handling Operation
High priority message received? IR.HPM="1"
Yes
No
High priority Message Handling Operation
Rx FIFO0-related interrupt detected? IR.RF0L, RF0F, RF0W, RF0N="1"
Yes
No
Rx FIFO0 top pointer Operation
Rx FIFO1-related interrupt detected? IR.RF1L, RF1F, RF1W, RF1N="1"
Yes
No
Rx FIFO1 top pointer Operation
End

図 226　　割込み処理

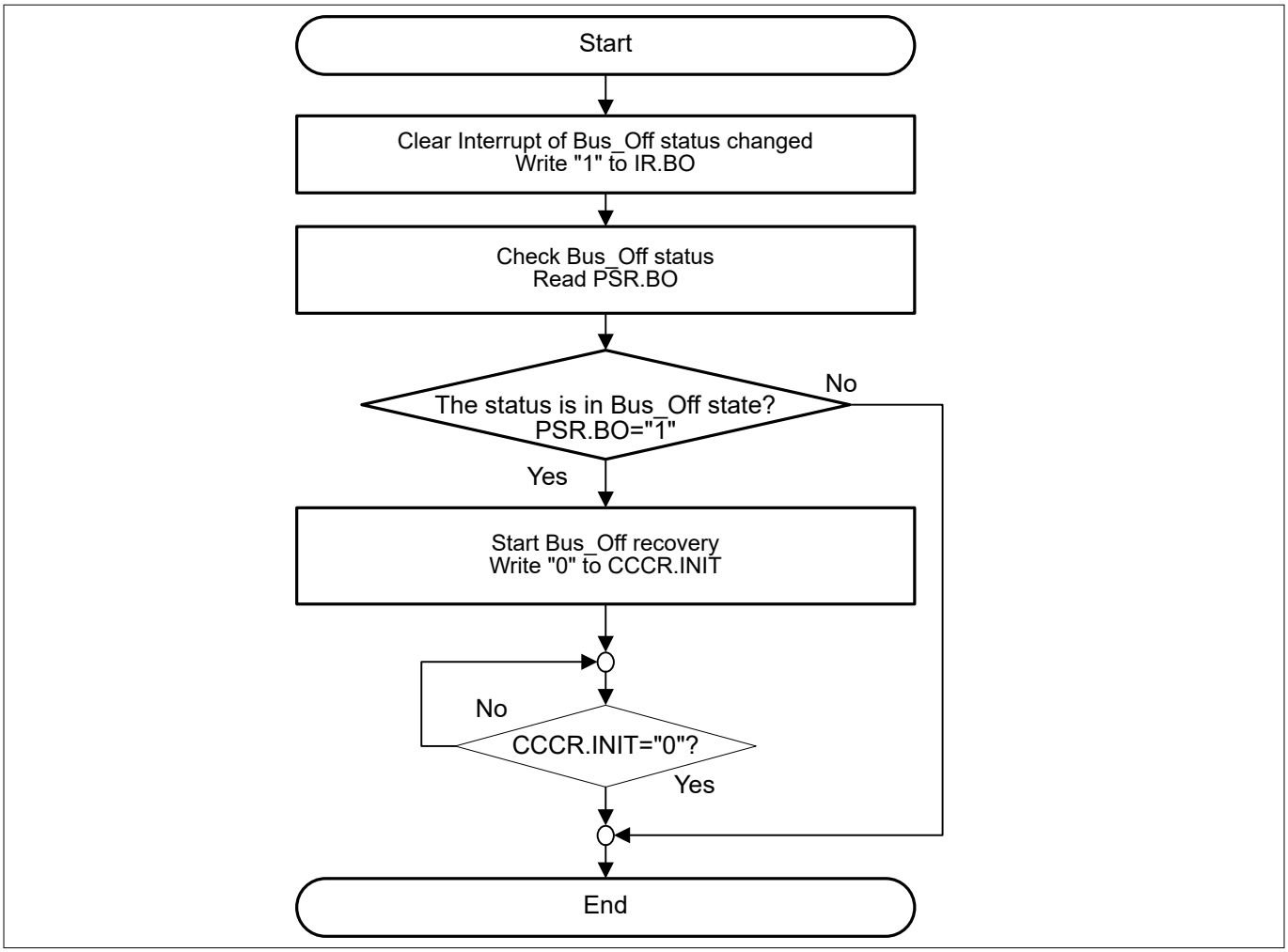


図 227　　バス OFF エラー処理

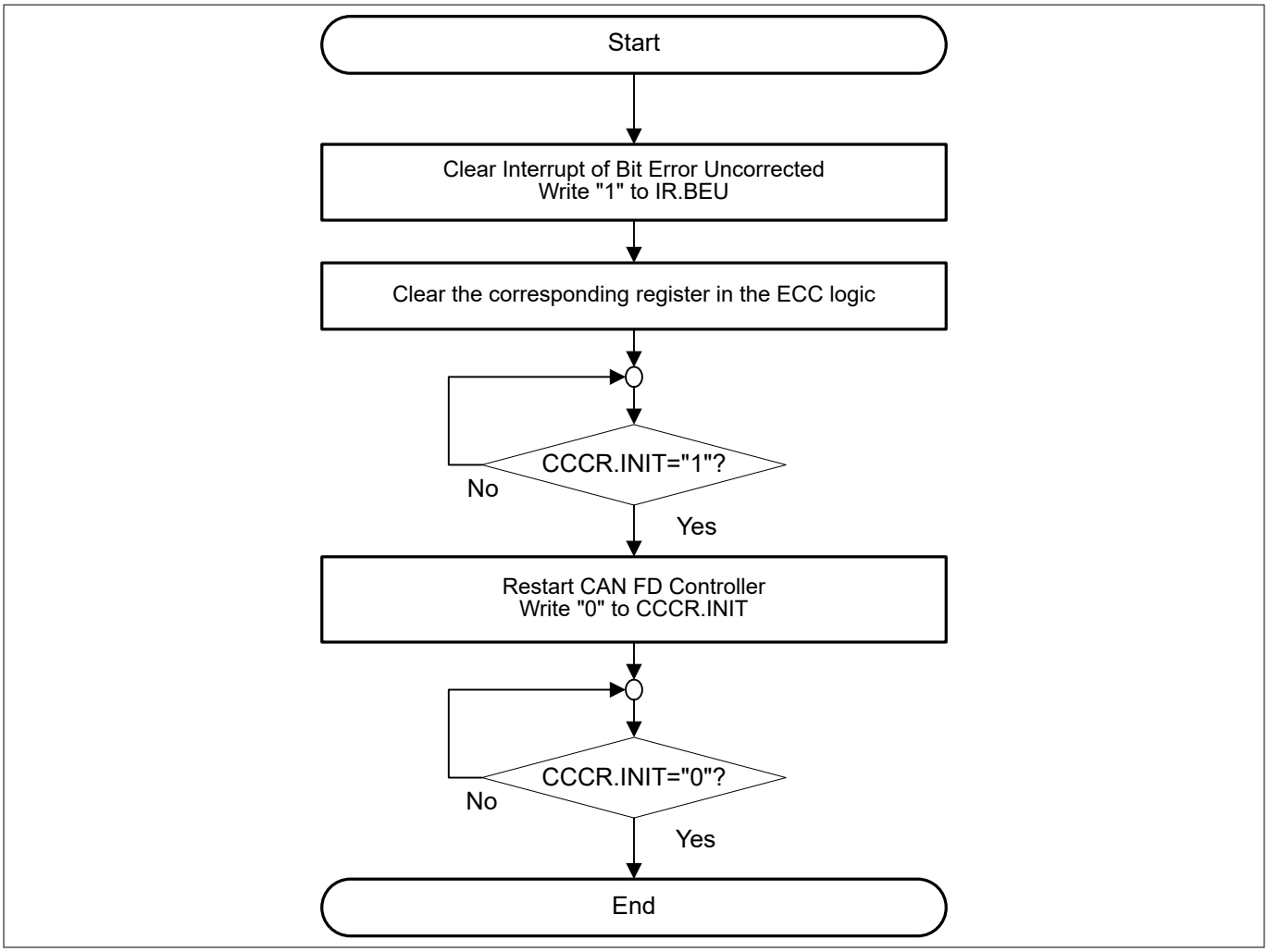


図 228　　メッセージ RAM アクセス失敗処理

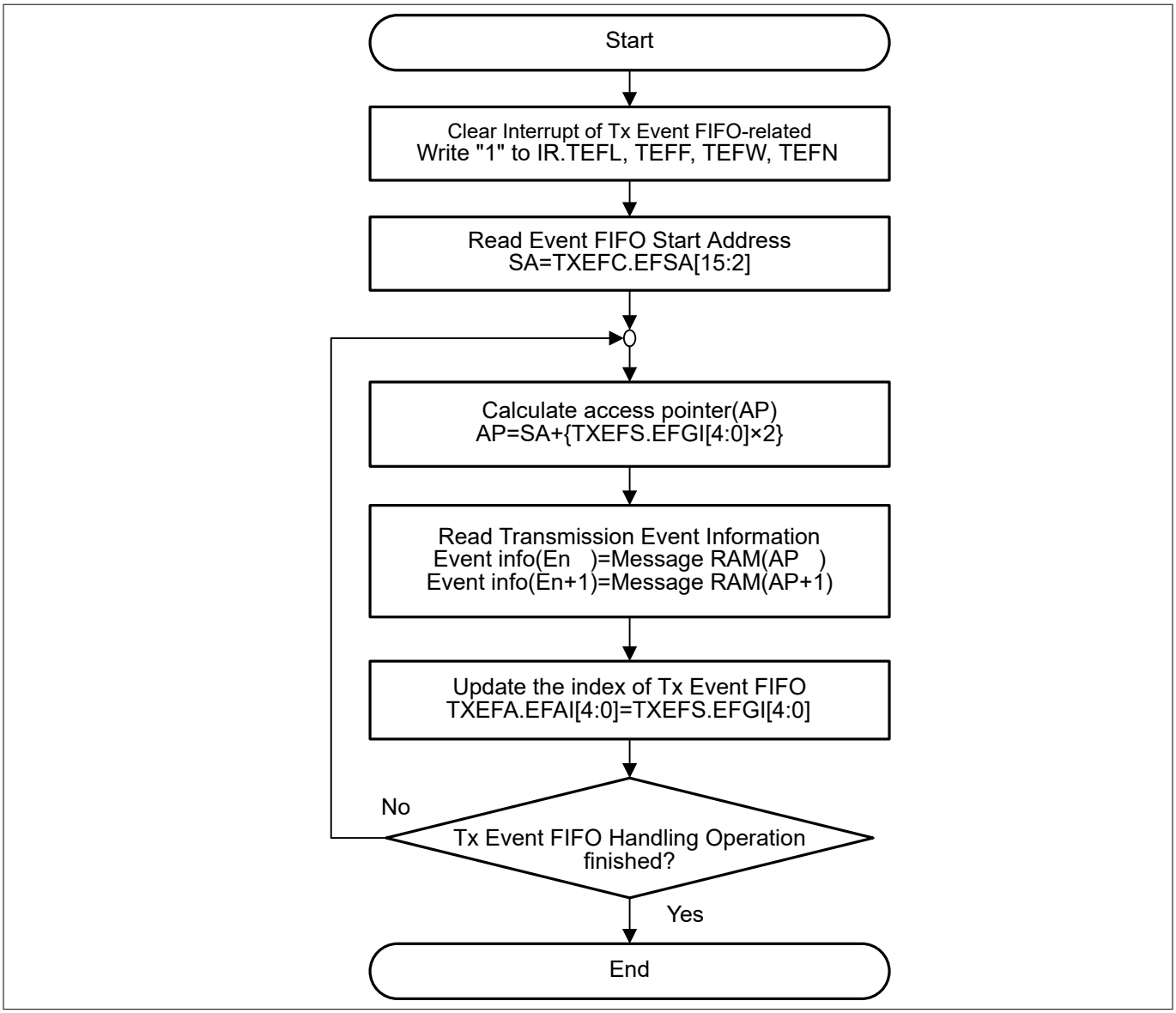


図 229　　TX イベント FIFO 処理

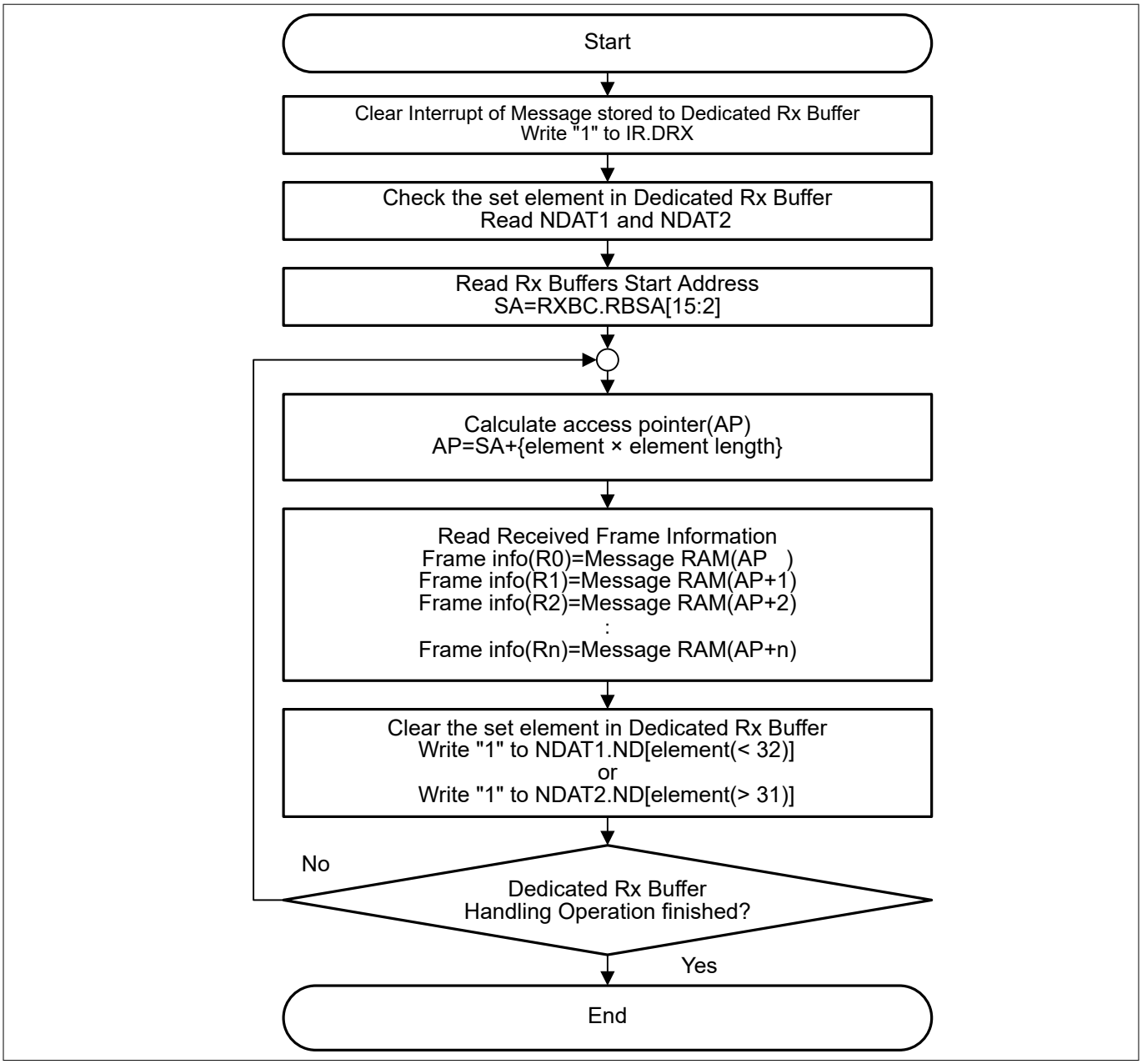


図 230　　専用 RX バッファ処理

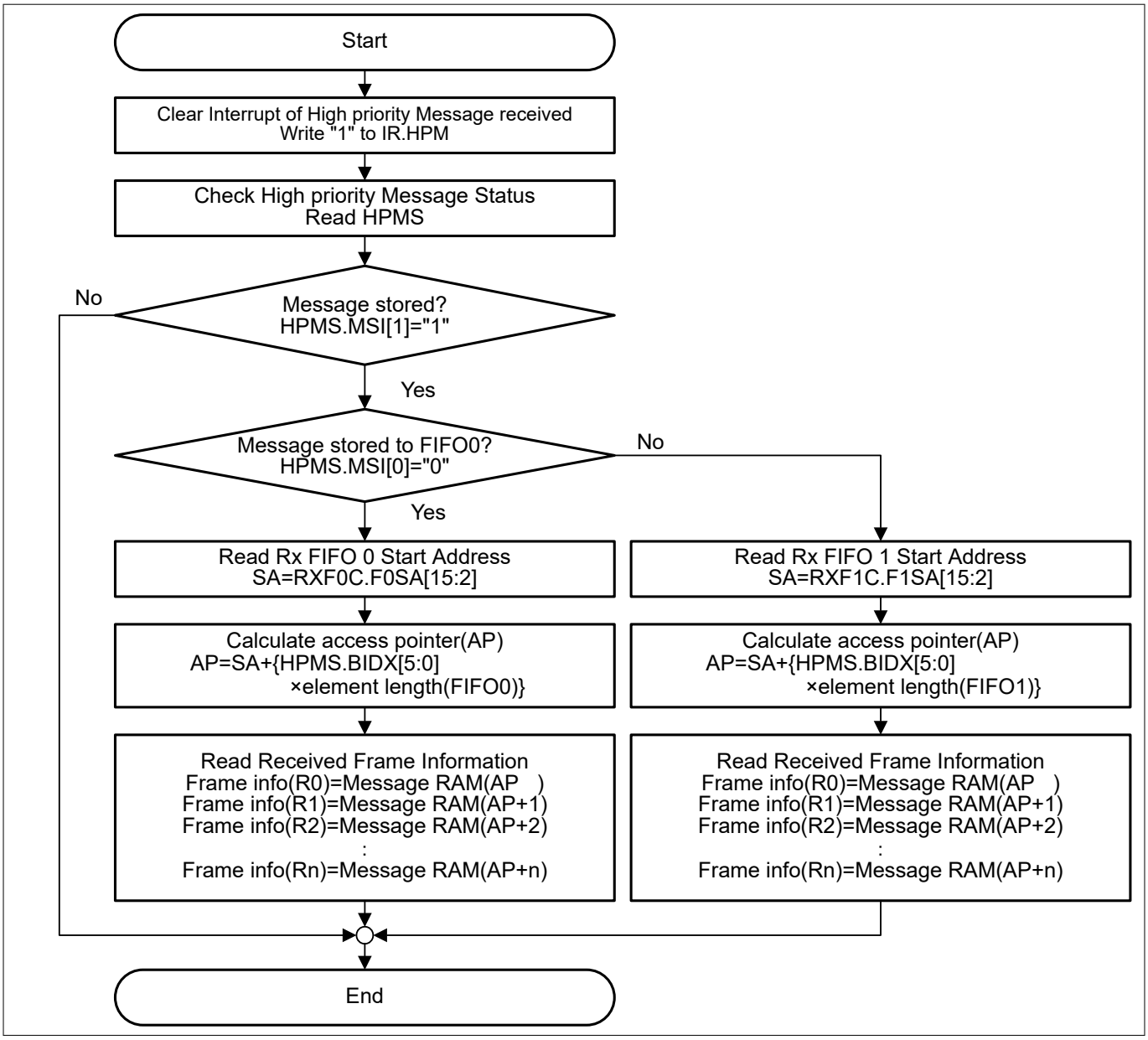


図 231　　優先度の高いメッセージ処理

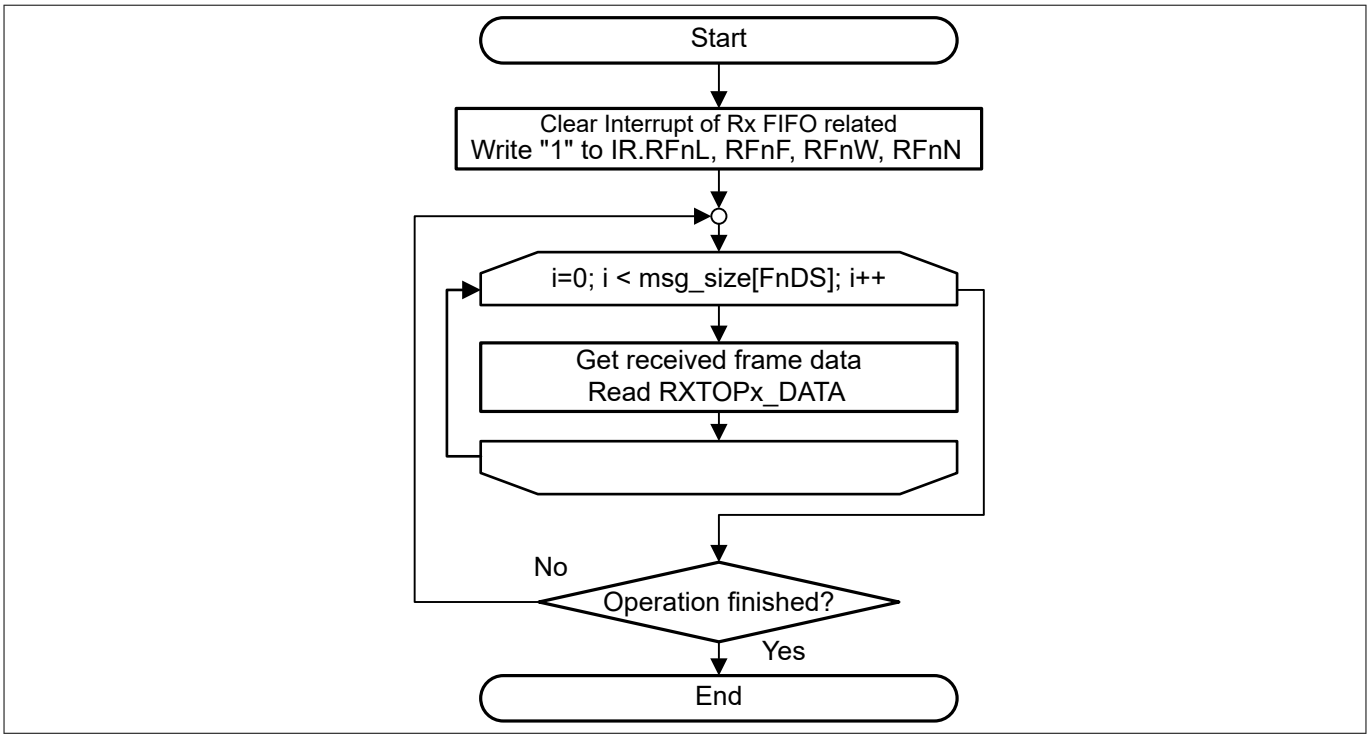


図 232　　RX FIFO トップポインタの処理

25.7　　レジスタの概要

レジスタの詳細については、PSOC™ Control C3 レジスタ RM を参照してください。

26 シリアル通信ブロック (SCB)

各 SCB は、SPI (Serial Peripheral Interface)、UART (Universal Asynchronous Receiver Transmitter)、I2C (Inter-Integrated Circuit) の 3 つのシリアル通信プロトコルに対応しています。SCB は、常にいずれか 1 つのプロトコルのみをサポートします。PSOC™ Control C3 MCU には複数の SCB が搭載されています。そのうちの 1 つは、システムのディープスリープ電力モードにおいて、I2C スレーブ モードと SPI スレーブ モードのみをサポートします。

26.1 機能

本 SCB は、以下の機能をサポートします。

- Motorola 社製、Texas Instruments 社製、National Semiconductor 社製のプロトコルと互換性のある標準的な SPI マスターと SPI スレーブ機能
- スマートカードインターフェース (ISO7816)、IrDA プロトコル、ローカル相互接続ネットワーク (LIN) サブモードを備えた標準的な UART 送受信機能
- UART 半二重通信に対応
- UART フロー制御 (CTS 入力および RTS 出力)
- プログラム可能なオーバーサンプリング付き内部クロック入力
- 標準 I2C マスターとスレーブ機能
- 各 SCB に受信 (Rx) および送信 (Tx) 用の 256 バイトの FIFO を搭載
- SPI および I2C スレーブ向けの EZ モード (CPU 介入なしの動作を実現)
- SPI および I2C スレーブ用の CMD_RESP モード。CPU の介入なしに動作可能で、システムのディープスリープ対応 SCB でのみ使用可能。
- SPI および I2C スレーブ用のシステム ディープスリープ動作モード (外部クロッキングを使用)、システム ディープスリープ対応 SCB でのみ使用可能
- I2C スレーブ アドレスの一致または SPI スレーブ セレクト時にシステム ディープスリープ ウェイクアップ。システム ディープスリープ対応は SCB でのみ使用可能。
- 25 MHz までの SPI マスター/スレーブサポート、50 MHz までの 1 つの SCB での高速 SPI サポート
- DMA 接続用トリガー出力
- FIFO と転送の状態を示す複数の割込みソース
- ローカル ループバック制御

26.2 アーキテクチャ

26.2.1 トリガー インターフェース

トリガーインターフェースは、TX FIFO ("tr_tx_req") と RX FIFO ("tr_rx_req") のステータス情報を提供します。

- **DMA トリガー信号**: "tr_tx_req"信号と"tr_rx_req"信号は通常、DW コントローラーに接続されます。
  - "tr_tx_req"は、TX FIFO が送信のためにデータ エレメントを受け入れることを示します。
  - "tr_rx_req"は、RX FIFO が (受信) データ エレメントを提供することを示します。
- **tr_i2c_scl_filtered 信号**: "tr_i2c_scl_filtered"信号は SCL 入力アナログ フィルター出力に接続されます。これによりグリッチが削除されます。
- **シリアル プロトコル インターフェース**: UART、I2C、SPI インターフェース信号は、IO サブシステム (IOSS) 内の高速 I/O マトリックス (HSIOM) に接続されます。

26.2.2 クロック モード

SCB のクロックは、周辺クロック分周器 (clk_scb) から供給される内部クロックで駆動するか、外部マスタークロックで駆動するかのどちらかです。

- UART、SPI マスター、および I2C マスター モードでは、clk_scb を使用する必要があります。
- SPI スレーブと I2C スレーブのみが外部マスターからのクロックを使用でき、ディープスリープ対応の SCB のみがこれをサポートします。

内部および外部クロックのスレーブ機能は、SCB CTRL レジスタの 2 つのレジスタ フィールドによって決定されます。

- EC_AM_MODE は、SPI スレーブ セレクトまたは I2C アドレスマッチングが内部クロック ('0') か外部クロック ('1') かを示します。
- EC_OP_MODE は、残りのプロトコル動作 (SPI スレーブ セレクトと I2C アドレスマッチング以外) が内部クロック ('0') か外部クロック ('1') かを示します。

注: FIFO モードは外部クロック動作 (SCBx_CTRL.EC_AM_MODE が'0'または'1') ではサポートされませんが、データ転送は内部クロック動作 (SCBx_CTRL.EC_OP_MODE が'0') で行う必要があります。

- EZ モードと CMD_RESP モードは外部クロック動作でサポートされます (SCBx_CTRL.EC_OP_MODE は'1')。

ディープスリープ モードに移行する前に、SCBx_CTRL.EC_ACCESS レジスタを'1'に設定する必要があります。ディープスリープモードからウェイクアップし、PLL がロックされている (CLK_SCB が期待される周波数になっている) 場合は、SCBx_CTRL.EC_ACCESS を'0'に設定します。

表 234 に、各通信モードでサポートされるクロッキング モードとバッファ モードの概要を示します。

表 234 クロックモードの互換性

	内部クロック (IC)			外部クロック (EC) (ディープスリープ SCB のみ)		
	FIFO	**EZ**	**CMD_RESP**	**FIFO**	**EZ**	**CMD_RESP**
I2C マスター	有り	無し	無し	無し	無し	無し
I2C スレーブ	有り	有り	無し	無し [1)]	有り	有り
I2C マスターまたはスレーブ	有り	無し	無し	無し	無し	無し
SPI マスター	有り	無し	無し	無し	無し	無し
SPI スレーブ	有り	有り	無し	無し [2)]	有り	有り
UART 送信側	有り	無し	無し	無し	無し	無し
UART 受信側	有り	無し	無し	無し	無し	無し

1) ディープスリープ時、I2C スレーブ FIFO モードにおいて、外部クロック駆動ロジックはスレーブアドレスマッチングを処理できます。そして割込みをトリガーして CPU をウェイクアップさせます。
2) ディープスリープ時、SPI スレーブ FIFO モードにおいて、外部クロック駆動ロジックはスレーブ セレクト検出を処理できます。そして割込みをトリガーして CPU をウェイクアップさせます。内部ロジックが引き継ぐまで、書き込みは無視され、読み出しは 0xFF を返します。これはディープスリープ対応の SCB にのみ適用されます。

表 235 I2C スレーブおよび SPI クロック コンフィギュレーションとモードのサポート

モード	SCBx_CTRL.EC_AM_MODE は'0'; SCBx_CTRL.EC_OP_MODE は'0'です。	SCBx_CTRL.EC_AM_MODE は'1'; SCBx_CTRL.EC_OP_MODE は'0'です。	SCBx_CTRL.EC_AM_MODE は'1'; SCBx_CTRL.EC_OP_MODE は'1'です。
FIFO モード	有り	有り	無し
EZ モード	有り	有り	有り
CMD_RESP モード	無し	無し	有り

26.2.3 Buffer Mode (バッファ モード)

各 SCB には、送受信動作用に 256 バイトの専用 RAM 備えています。この RAM は、3 つの異なるモード (FIFO、EZ、CMD_RESP) で設定できます。

各プロトコルのセクションに詳細を記載しています。

- マスターは FIFO モードしか使えません
- I2C および SPI スレーブは、3 つのモードすべてを使用できます。

注: CMD_RESP モードは、システムのディープスリープ対応 SCB でのみ使用可能です。

- UART は FIFO モードのみを使用

26.2.3.1 FIFO モード

このモードでは、RAM は 2 つの 128 バイト FIFO に分割され、1 つは送信 (TX) 用、もう 1 つは受信 (RX) 用です。FIFO は 8 ビット×128 エレメントまたは 16 ビット×64 エレメントに設定可能であり、これは SCBx_CTRL.MEM_WIDTH レジスタの設定によって行われます。

FIFO 動作モードは、CPU アクティブ モードとシステム スリープ モードでのみ利用可能です。ただし、I2C アドレスまたは SPI スレーブ セレクトを使用して、システム ディープスリープ対応の SCB でシステム ディープスリープからデバイスをウェイクアップできます。ステータスは RX と TX FIFO の両方について提供されます。複数の割込みソースが利用可能であり、これらは FIFO の状態 (フルまたは空など) を示します。SCB 割込みを参照してください。

26.2.3.2 EZ モード

EZ モードでは、RAM はシングル 256 バイトのバッファとして使用されます。外部マスターも内部マスターもベースアドレスを設定し、そのベース アドレスから読み出し/書き込みを開始します。

EZ モードは、SPI スレーブと I2C スレーブでのみ使用可能です。

EZ モードは、CPU アクティブおよびシステム ディープスリープ電力モードで利用可能です。

26.2.3.3 CMD RESP モード

コマンド レスポンス (CMD_RESP) モードは、ベース アドレスが外部マスターではなく CPU から提供される点を除き、EZ モードと同様です。

CMD_RESP モードは、SPI スレーブと I2C スレーブでのみ使用可能です。システム ディープスリープ対応 SCB でのみ使用可能。

CMD_RESP モード動作は、CPU アクティブおよびシステム ディープスリープ電力モードで利用可能です。

26.2.4 ブロック操作

26.2.4.1 ブロック有効化とリカバリー

SCB ブロック (CTRL.ENABLED) は、レジスタ値をそれぞれ'1'または'0'に設定することにより、有効または無効にできます。SCB ブロックを予期せぬ状態から回復させるには、CTRL.ENABLED レジスタで再有効化して、ハードウェア ブロックとレジスタのソフト リセットを実行します。CTRL.ENABLED = 0 のとき、ACTIVE 電源ドメインのすべてのレジスタはリセットされます。

26.2.4.2 リセット後の初期化フロー

SCB を初期化するためには、次の手順を実行してください。

1. SPI_CTRL, UART_CTRL (および UART_TX_CTRL と UART_RX_CTRL) または I2C_CTRL レジスタを使用してプロトコル固有の情報をプログラムしてください。これには、サブモードの選択、マスター/スレーブ機能、送信側/受信側機能 (該当する場合) が含まれます。
2. 一般的な送信側 (TX_CTRL) と受信側 (RX_CTRL) の情報をプログラムしてください。これには、送信側と受信側の機能の有効化が含まれます。
3. 送信 FIFO (TX_FIFO_CTRL) と受信 FIFO (RX_FIFO_CTRL) の情報をプログラムしてください。
4. SCB ブロックを有効にするために CTRL レジスタをプログラムしてください。特定の動作モードとオーバーサンプリング率を選択してください。

26.2.5 電力モード

SCB は、CPU アクティブおよび CPU スリープの電力モードで動作できます。SCB[0]のみがシステム ディープスリープ電力モードで動作できます。詳細については、クロック モードを参照してください。

CPU アクティブからシステム ディープスリープへの電力モードの移行

マスターモード: マスターは、低消費電力状態またはウェイクアップを要求する通信を制御します。どちらの場合も、通信、トランジション、低消費電力状態は同期されます。

スレーブモード: SCB がスレーブとして設定され、外部送信機を制御できない状態で、システム ディープスリープに移行し、外部マスターが送信を開始します。通信フレームが低消費電力状態や必要な移行期間と同期していることを確認するのは、顧客のソフトウェアや上位通信プロトコル次第です。

スレーブモードでは、いくつかの境界条件を考慮する必要があります。上位の通信プロトコルでは、通常、低消費電力状態やウェイクアップを要求するコマンドがあります。どちらの場合も、通信、移行、低消費電力状態は同期されており、制限は関係ありません。

上位プロトコルがなく、パワーステートがマスター通信と非同期である場合、双方で期待される動作をどのように実現するかは顧客のコード次第です。例えば、マスターからの読み出しアクセスの場合、スレーブはデータがまだ FIFO にあることを認識しています。マスターからの書き込みアクセスは、電源状態間の移行中に失われる可能性があります。

***注**:*

1. *外部クロックモードで CPU がアクティブからシステム ディープスリープに移行する間、移行期間中にスレーブに送信されたトランザクションが失われることが確認されています。*
2. *内部クロック モードで CPU がアクティブからシステム ディープスリープに移行する間、クロック ソースの自動切り替えは行われず、システム ディープスリープ モードでの転送は利用できません。*

26.3 シリアル ペリフェラル インターフェース (SPI)

SPI プロトコルは同期シリアル インターフェース プロトコルです。デバイスはマスター モードまたはスレーブ モードで動作します。データ転送を開始できるのはマスターのみです。SCB は、SPI のシングルマスター-マルチスレーブトポロジをサポートします。個別のスレーブ セレクト ラインにより、最大 4 つまでの複数スレーブをサポートします。

26.3.1 機能

- マスター/スレーブ機能に対応
- 3 つの種類の SPI プロトコルに対応:
  - Motorola 社製 SPI – モード 0、1、2、3
  - Texas Instruments 社製 SPI、同時および先行データ フレーム インジケーター付き - モード 1 のみ
  - ナショナルセミコンダクター (MicroWire) SPI - モード 0 のみ
- マスターは最大 3 本のスレーブ セレクト ラインに対応
  - 各スレーブ セレクトはアクティブ極性 (ハイまたはロー) を設定可能
  - スレーブ セレクトは、転送全体またはバイトごとにアクティブになるように設定可能
- マスターは、より良いタイミング マージンのための遅延サンプリングに対応
- マスターは連続 SPI クロックに対応
- データ フレーム サイズは 4 ビット～32 ビットにプログラム可能
- SELECT 出力信号タイミング可変 (SPI マスター):
  - SELECT セットアップ時間 (セレクト アクティブ～SPI クロック)、SELECT ホールド時間 (SPI クロック～セレクト インアクティブ) (Motorola 社製モードのみ)
  - データ フレーム間選択解除時間 (非アクティブの選択～アクティブの選択) (Motorola 社製モードのみ)
- パリティ サポート (奇数および偶数パリティ) (Motorola 社製モードのみ)
- 割込みまたはポーリングによる CPU インターフェース
- プログラム可能なオーバーサンプリング
- MSB または LSB ファースト
- 入力で利用可能なメディアンフィルター
- FIFO モード、EZ モード (スレーブのみ)、CMD_RESP モード (スレーブのみ) に対応
- スレーブ セレクト時にウェイクアップ割込みが発生 (SCB[0]のみ)
- ローカル ループバック制御
- 最大 50 MHz の高速 SPI 動作 (SCB[5]のみ)

26.3.2 概要

図 233 に 3 つのスレーブを備えた SPI マスターの例を示します。

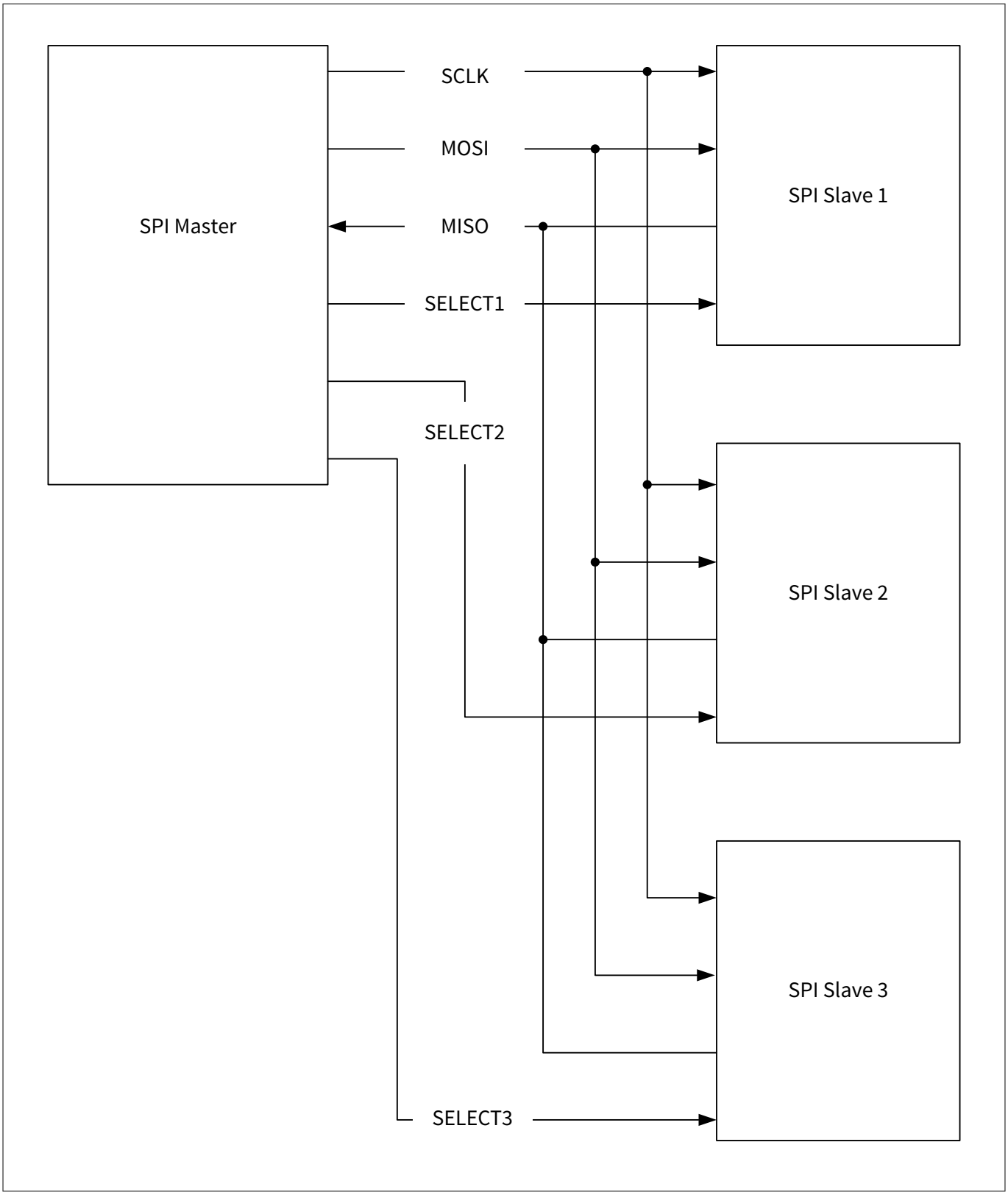


図 233　　SPI の例

標準的な SPI インターフェースは、以下の 4 つの信号で構成されています。

- SCLK: シリアル クロック (マスターからのクロック出力、スレーブへの入力)
- MOSI: マスター アウト スレーブ イン (マスターからのデータ出力、スレーブへの入力)

- MISO: マスター イン スレーブ アウト (マスターへのデータ入力、スレーブからの出力)
- SELECT: 通常アクティブロー信号 (マスターから出力、スレーブへ入力)

単純な SPI データ転送には以下のようなものがあります。

- マスターは SELECT ラインをドライブすることでスレーブを選択します。次に、MOSI ラインのデータと SCLK ラインのクロックを駆動します。スレーブは、コンフィギュレーションに応じて SCLK のいずれかのエッジを使用して、MOSI ラインのデータをキャプチャします。また、MISO ラインのデータも駆動し、これはマスターによってキャプチャされます。

データ フレーム サイズは、4 ビットから 32 ビットの範囲で任意の値に設定できます。シリアル データは、最上位ビット (MSb) を先に送信することも、最下位ビット (LSb) を先に送信することもできます。

SPI プロトコルの 3 つの種類が SCB によってサポートされます。

- Motorola 社製 SPI: オリジナルの SPI プロトコルです。
- Texas Instruments 社製 SPI: オリジナルの SPI プロトコルの変形です。このプロトコルではデータ フレームは SELECT ライン上のパルスで識別されます。
- National Semiconductor 社製 SPI: オリジナルの SPI プロトコルの半二重バリエーション

二重制御に関する注意事項:

- Motorola 社製と Texas Instruments 社製のモードは全二重通信、National Semiconductor 社製のモードは半二重通信
- SCBx_TX_FIFO_CTRL.FREEZE または SCBx_RX_FIFO_CTRL.FREEZE によって制御される全二重モードも半二重と同様に動作し、ダミーのデータワードを送信したり、受信したデータワードを無視したりします。
- MOSI は、IOSS/GPIO コンフィギュレーションを使用して Hi-Z 状態に設定できます。

26.3.3 SPI 動作モード

26.3.3.1 Motorola 社製 SPI

オリジナルの SPI プロトコルは Motorola 社によって定義されました。これは全二重プロトコルです。SELECT ラインが'0'に維持される間に複数のデータ転送が発生する可能性があります。したがって、スレーブ デバイスは個別のデータ フレームを分離するためにデータ転送の進捗を確認する必要があります。データを送信しない時に SELECT ラインは'1'に維持され、SCLK は通常 Low にプルダウンです。

Motorola 社製 SPI のクロックモード

Motorola 社製 SPI プロトコルは、データを MOSI および MISO ラインにおけるデータの駆動方法と取得方法に基づき、4 つの異なるクロックモードがあります。これらのモードはクロック極性 (SCBx_SPI_CTRL.CPOL) とクロック位相 (SCBx_SPI_CTRL.CPHA) によって決められます。

クロック極性はデータの送信中でない時の SCLK ラインの値を決定します。SCBx_SPI_CTRL.CPOL= 0 はデータの送信中でない時の SCLK が'0'であることを示します。SCBx_SPI_CTRL.CPOL= 1 はデータの送信中でない時の SCLK が'1'であることを示します。

クロック位相はデータが駆動されキャプチャされるタイミングを決めます。SCBx_SPI_CTRL.CPHA = 0 は先行 (1 クロック目) エッジでサンプル (データキャプチャ) することを意味し、SCBx_SPI_CTRL.CPHA = 1 はクロックエッジが立ち上りか立ち下りに関係なく、後行 (2 クロック目) エッジでサンプルすることを意味します。SCBx_SPI_CTRL.CPHA = 0 の場合、データは最初のクロック サイクルの前のセットアップ時間で安定する必要があります。

- モード 0: SCBx_SPI_CTRL.CPOL = 0, SCBx_SPI_CTRL.CPHA = 0; データは SCLK の立ち下りエッジで駆動されます。データは SCLK の立ち上りエッジでキャプチャされます。SCLK アイドル状態が'0'
- モード 1; SCBx_SPI_CTRL.CPOL = 0, SCBx_SPI_CTRL.CPHA = 1: データは SCLK の立ち上りエッジで駆動されます。データは SCLK の立ち下りエッジでキャプチャされます。SCLK アイドル状態が'0'

- モード 2: SCBx_SPI_CTRL.CPOL = 1, SCBx_SPI_CTRL.CPHA = 0: データは SCLK の立ち上りエッジで駆動されます。データは SCLK の立ち下りエッジでキャプチャされます。SCLK アイドル状態が'1'
- モード 3: SCBx_SPI_CTRL.CPOL = 1, SCBx_SPI_CTRL.CPHA = 1: データは SCLK の立ち下りエッジで駆動されます。データは SCLK の立ち上りエッジでキャプチャされます。SCLK アイドル状態が'1'

図 234 に、MOSI/MISO データのドライブとキャプチャを SCBx_SPI_CTRL.CPOL および SCBx_SPI_CTRL.CPHA の機能として示します。

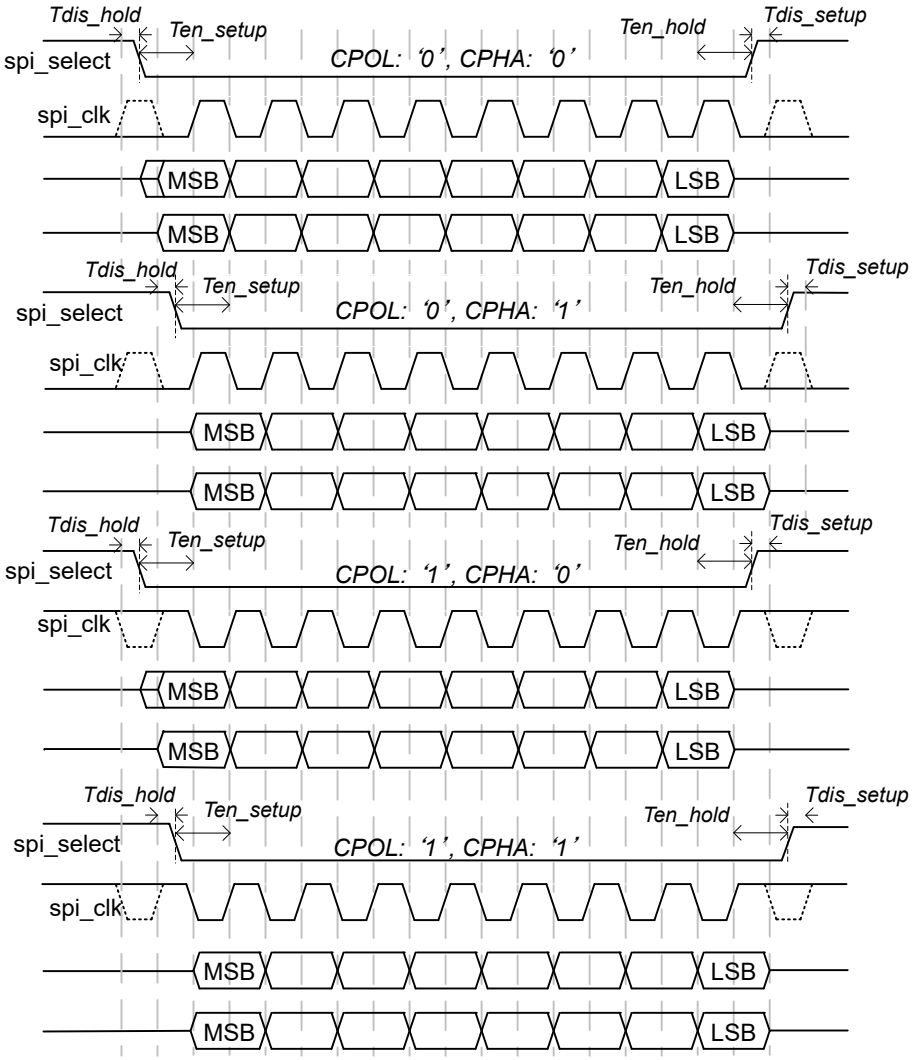


図 234　　SPI Motorola、4 つのモード

図 235 に、モード 0 (SCBx_SPI_CTRL.CPOL= 0, SCBx_SPI_CTRL.CPHA = 0) でのシングル 8 ビット データ転送と 2 つの連続した 8 ビット データ転送を示します。

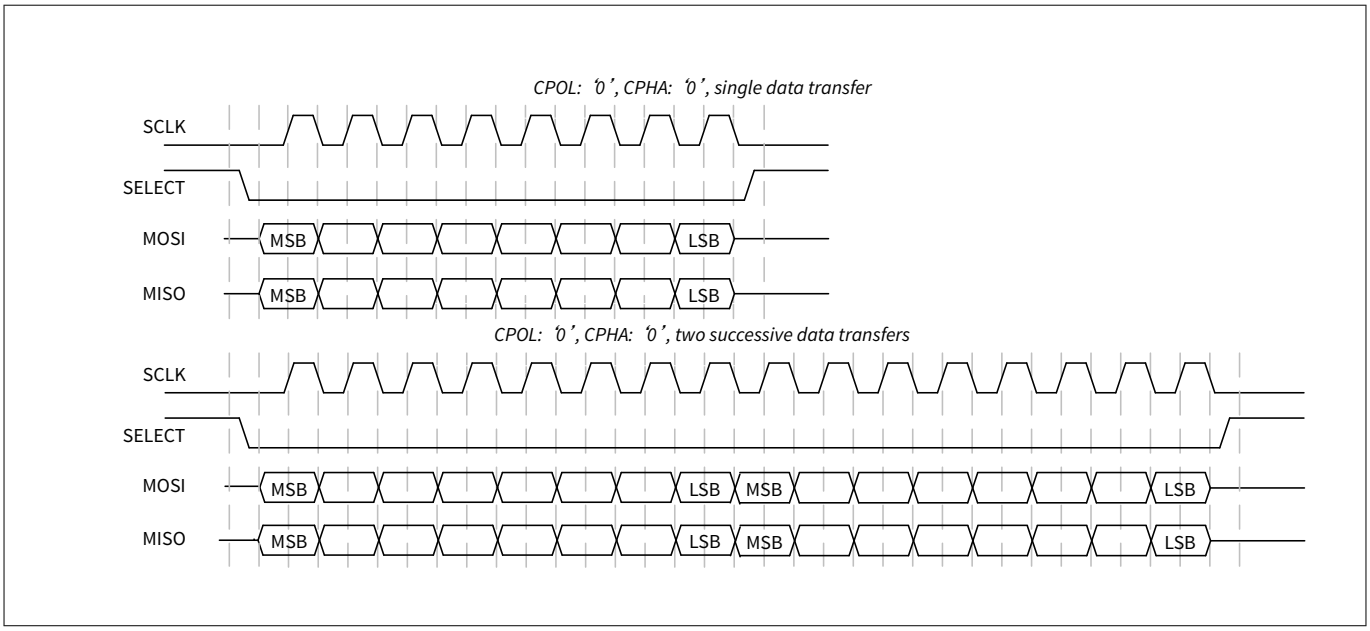


図 235　　SPI Motorola のデータ転送例

SPI Motorola モード用の SCB の設定

SPI Motorola モード用に SCB を設定するためには様々なレジスタビットを次の順で設定してください。

1. SCBx_CTRL.MODE レジスタに'01'を書き込んで SPI を選択してください。
2. SCBx_SPI_CTRL.MODE レジスタに'00'を書き込んで SPI Motorola モードを選択してください。
3. SCBx_SPI_CTRL.CPHA および SCBx_SPI_CTRL.CPOL レジスタに書き込み、Motorola 社製での動作モードを選択してください。
4. SPI の有効化と初期化 セクションに記載されているステップ 2～4 に従ってください。

これらのレジスタの詳細については、PSOC™ Control C3 レジスタリファレンス マニュアルを参照してください。

26.3.3.2　　Texas Instruments 社製 SPI

Texas Instruments 社製 SPI プロトコルは、SELECT 信号の使い方を再定義しています。このプロトコルはデータ転送の開始を示すために Motorola 社製 SPI の場合のアクティブ LOW スレーブ選択信号ではなくこの信号を使用します。したがって、スレーブ デバイスは個別のデータ フレームを分離するためにデータ転送の進捗を追跡する必要がありません。転送開始は単一ビット転送周期のアクティブ HIGH パルスで示されます。このパルスは最初のデータビットの送信の 1 サイクル前に発生するか、または最初のデータビットの送信と同時に発生します。Texas Instruments 社製 SPI プロトコルはモード 1 (SCBx_SPI_CTRL.CPOL = 0、SCBx_SPI_CTRL.CPHA = 1) のみをサポートします。データは SCLK の立ち上りエッジで駆動され、SCLK の立ち下りエッジでキャプチャされます。

図 236 に、シングル 8 ビット データ転送と 2 つの連続した 8 ビット データ転送を示します。SELECT パルスは最初のデータビットに先行します。2 回目のデータ転送の SELECT パルスが、1 回目のデータ転送の最後のデータビットとどのように一致しているかに注意してください。

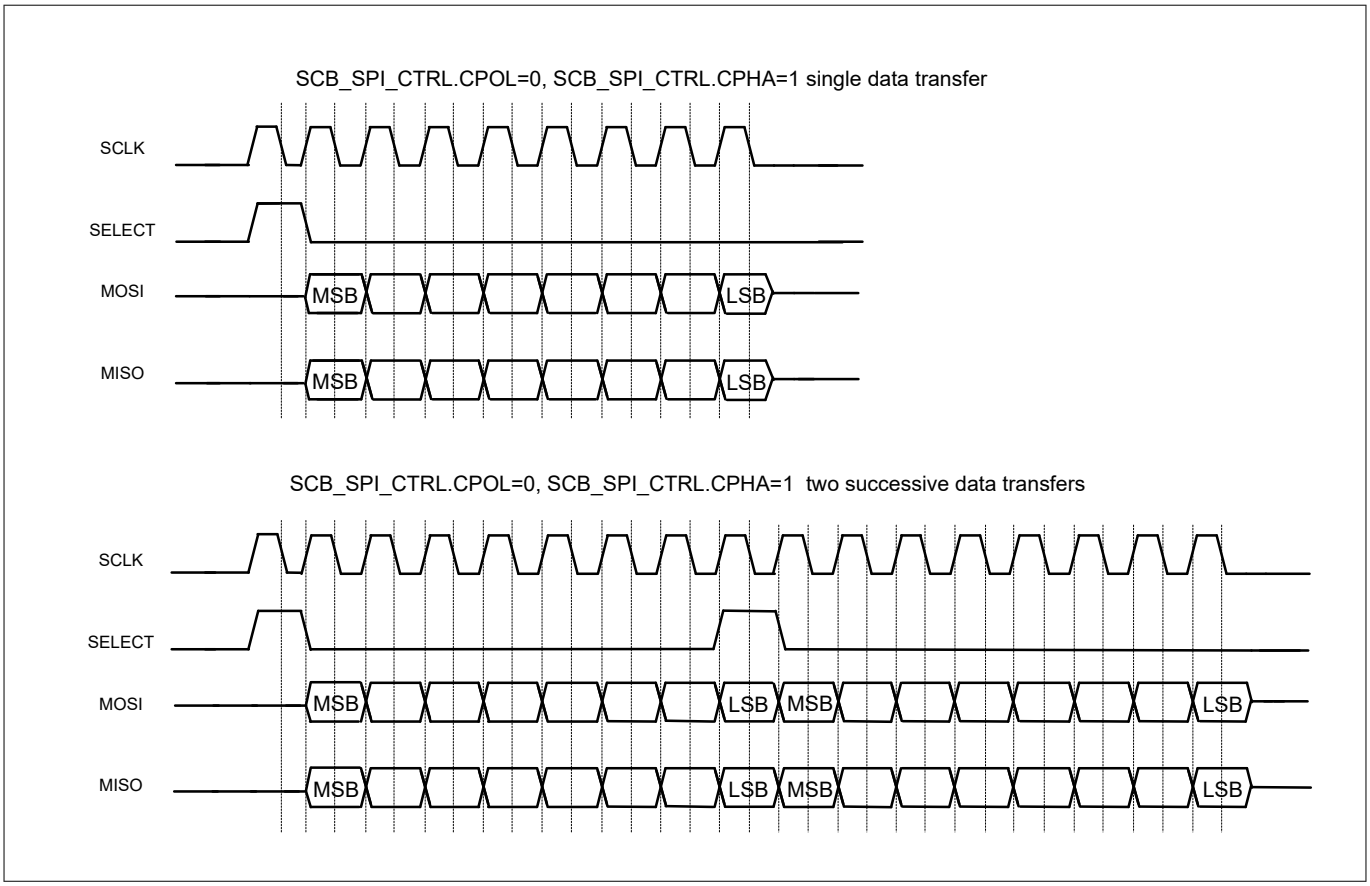


図 236　　SPI TI のデータ転送例

図 237 に、シングル 8 ビット データ転送と 2 つの連続した 8 ビット データ転送を示します。SELECT パルスはフレームの最初のデータビットと一致します。

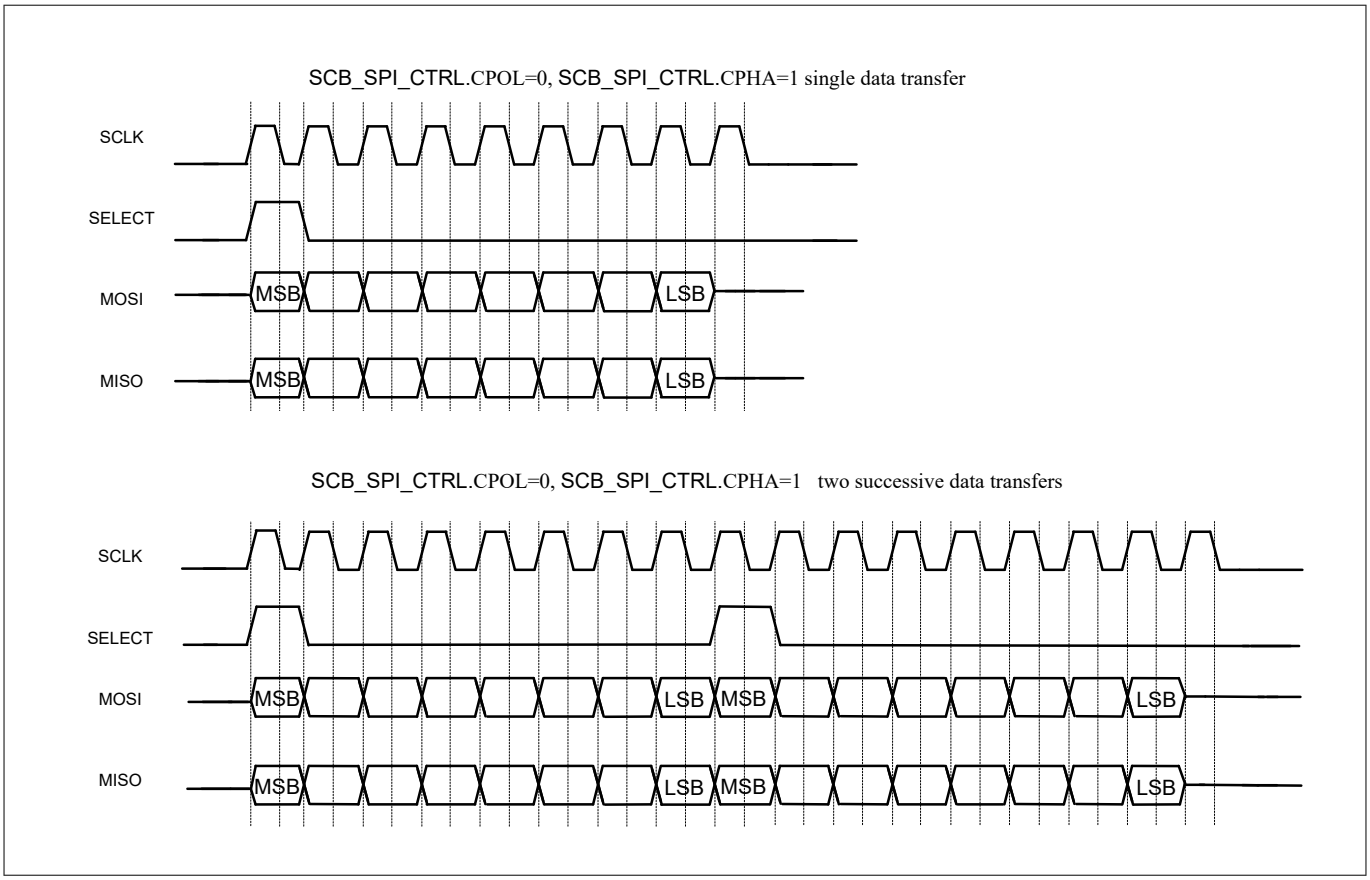


図 237 SPI TI のデータ転送例 2

SPI TI モード用の SCB の設定

SPI TI モード用に SCB を設定するためには様々なレジスタビットを次の順で設定してください。

1. SCBx_CTRL.MODE レジスタに'01'を書き込んで SPI を選択してください。
2. SCBx_SPI_CTRL.MODE レジスタに'01'を書き込んで SPI TI モードを選択してください。
3. SCBx_SPI_CTRL.SELECT_PRECEDE レジスタに書き込み、Texas Instruments 社製の動作モードを選択してください ('1': SELECT パルスが次のフレームの最初のビットの前に来るように設定、'0': その他の設定)
4. SPI の有効化と初期化 セクションに記載されているステップ 2～4 に従ってください。

これらのレジスタの詳細については、PSOC™ Control C3 レジスタリファレンス マニュアルを参照してください。

26.3.3.3 National Semiconductor 社製 SPI

National Semiconductor 社製 SPI プロトコルは半二重プロトコルです。このプロトコルでは、送信と受信は同時に行われるのではなく、交互に行われ、送信は受信の前に行われます。送信と受信のデータ サイズが異なることがあります。シングル「アイドル」(= '0') ビット転送期間が送信と受信を分離します。ただし連続したデータ転送は「アイドル」ビット転送期間で分離されません。National Semiconductor 社製 SPI プロトコルはモード 0 のみに対応しています。

図 238 に、シングルデータ転送と 2 つの連続したデータ転送を示します。いずれの場合も、送信データの転送サイズは 8 ビット、受信データの転送サイズは 4 ビットです。

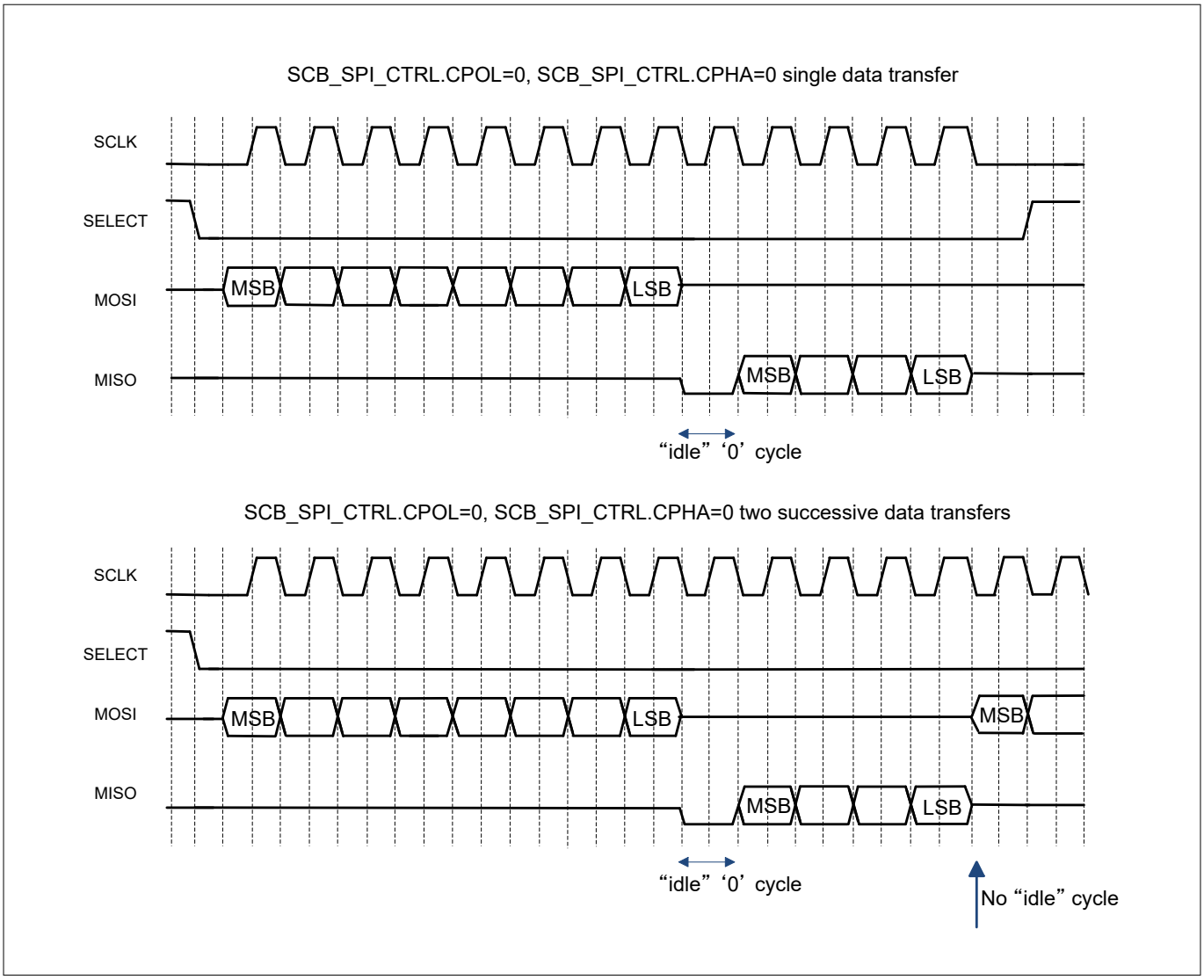


図 238　　SPI NS のデータ転送例

SPI NS モード用の SCB の設定

SPI NS モード用に SCB を設定するためには様々なレジスタビットを次の順で設定してください。

1. SCBx_CTRL.MODE レジスタに'01'を書き込んで SPI を選択してください。
2. SCBx_SPI_CTRL.MODE レジスタに'10'を書き込んで SPI NS モードを選択してください。
3. SCBx_SPI_CTRL.CPOL と SCBx_SPI_CTRL.CPHA のビットを'0'にしてください。
4. SPI の有効化と初期化 セクションに記載されているステップ 2～4 に従ってください。

これらのレジスタの詳細については、PSOC™ Control C3 レジスタリファレンス マニュアルを参照してください。

26.3.4　　SPI バッファモード

SPI は、以下の 3 つの異なるバッファ モードで動作できます。

- FIFO モード
- EZ モード
- CMD_RESP モード

バッファの使い方は、それぞれのモードで異なります。以下のサブセクションでは、それぞれのバッファモードについて詳しく説明します。

26.3.4.1 FIFO モード

FIFO モードには、送信データ用の TX FIFO と受信データ用の RX FIFO があります。各 FIFO は SRAM バッファから構成されます。FIFO は、32 ビット データ エレメントで 32 エレメント、または 16 ビット データ エレメントで 64 エレメント、または 8 ビット データ エレメントで 128 エレメントです。FIFO の幅は SCBx_CTRL.MEM_WIDTH レジスタで設定されます。

FIFO モードは、CPU のアクティブ電力モードとスリープ電力モードでのみ利用可能で、システムのディープスリープ モードでは利用できません。

送受信 FIFO により、書き込みと読み出しアクセスを許可します。送信 FIFO への書き込みアクセスには、SCBx_TX_FIFO_WR レジスタを使用します。受信 FIFO への読み出しアクセスには、SCBx_RX_FIFO_RD レジスタを使用します。

送受信 FIFO のステータス情報は、SCBx_TX_FIFO_STATUS および SCBx_RX_FIFO_STATUS ステータス レジスタを通じて入手できます。FIFO エントリーの数を示すプログラム可能な閾値を定義できます。以下の条件が満たされたときにトリガー/イベントが発生します。

- 送信 FIFO には SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL が設定されています。送信 FIFO のエントリー数が SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL 未満の場合、トリガー/イベントが発生します。
- 受信 FIFO には SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL が設定されています。受信 FIFO エントリー数が SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL より大きい場合、トリガー/イベントが発生します。

これらのトリガーは DW チャネルに接続できます。

さらに、RX FIFO と TX FIFO の両方に多数の割込みステータス ビットが用意されています。これらの情報は、SCBx_INTR_TX および SCBx_INTR_RX を参照してください。

システム ディープスリープから CPU アクティブへの移行

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, FIFO モード。

MISO は、内部クロック ロジックが引き継いで CPU が TX FIFO に書き込むまで 0xFF を送信します。MOSI 上のデータは、内部クロック ロジックが引き継ぐまで無視されます。内部クロック ロジックが引き継がれるとき、PLL/FLL ロック時間のため、内部クロックが正しい周波数になる保証はありません。これにより、RX FIFO 内のデータが破損することがあります。したがって、システムのディープスリープから CPU アクティブに移行した後、TX FIFO に新しいデータを書き込む前に RX FIFO をクリアすることを推奨します。もう一つの選択肢は、システムのディープスリープに入る前に CLK_SCB を無効にし、PLL と FLL が安定するまで有効にするのを待つことです。外部マスターは、MISO で 0xFF を読み出すとき、デバイスがまだ準備ができていないことを認識する必要があります。

26.3.4.2 EZSPI モード

EZSPI プロトコルはすべてのモード (0、1、2、または 3) で動作する Motorola 社製 SPI に基づいたものです。CPU の介入なしにマスターとスレーブ間の通信を可能にします。PSOC™ Control C3 では、ディープスリープ対応の SCB ブロックのみが EZSPI モードをサポートします。

EZSPI プロトコルは、スレーブ デバイス上にあるバッファ (1 エントリー 8 ビットの 256 エントリー配列) のインデックスとなる 8 ビットの EZ アドレスを持つ 1 つのメモリ バッファを定義します。EZ アドレスは、この 256 ロケーションをアドレス指定するために使用されます。すべての EZSPI データ転送は 8 ビットのデータ フレームで構成されます。CPU は SCBx_EZ_DATA レジスタをとおしてメモリ バッファに書き込みと読み出しを行います。これらのアクセスはワード アクセスですが、使用されるのはワードの最下位バイトのみです。

EZSPI には以下の 3 種類の転送があります。

- マスターからスレーブへの EZ アドレスの書き込み
- マスターから、アドレス指定されたスレーブのメモリ ロケーションへのデータの書き込み
- アドレス指定されたスレーブ メモリ ロケーションからマスターが読み出し

注: 複数のバイトの読み出し/書き込みを行う場合、マスターは転送中ずっと SELECT を Low に保たなければいけません。

EZ アドレス書き込み

EZ アドレスの書き込みは EZ アドレスを書き込むマスターの意図を示す MOSI ライン上のコマンド バイト (0x00) から始まります。コマンド バイトの受信中、スレーブは (MISO で) 0xfe (Ready) または 0xff (ビジー) を送信します。無効またはリセットされると、スレーブは (MISO で) 0xff (ビジー) を送信します。このバイト値は、マスターがスレーブが SPI リクエストを受け入れる準備ができているかどうかを判断するために使用できます。その後、スレーブはコマンドが承認された (0xFE) か否か (0xFF) ことを示すために MISO ライン上に返信バイトを送ります。MOSI 上の 2 番目のバイトは EZ アドレスです。

メモリ アレイ書き込み

メモリ アレイ インデックスへの書き込みはメモリ アレイに書き込むマスターの意図を示す MOSI ライン上のコマンド バイト (0x01) から始まります。コマンド バイトの受信中、スレーブは (MISO で) 0xfe (Ready) または 0xff (ビジー) を送信します。無効またはリセットされると、スレーブは (MISO で) 0xff (ビジー) を送信します。このバイト値は、マスターがスレーブが SPI リクエストを受け入れる準備ができているかどうかを判断するために使用できます。その後、スレーブはコマンドが登録 (0xFE) または非登録 (0xFF) であることを示すために MISO ライン上に返信バイトを駆動します。MOSI 上の追加書き込みのデータ バイトは通信した EZ アドレスによって示される位置のメモリ アレイに書き込まれます。バイトがメモリ アレイに書き込まれると EZ アドレスはスレーブによって自動的にインクリメントされます。EZ アドレスがメモリ エントリーの最大数 (256) を超えるとその値を維持し、0 にラップ アラウンドしません。EZ ベース アドレスは、各スレーブ セレクト時に EZ アドレス書き込みフェーズで書き込まれたアドレスにリセットされます。

メモリ アレイの読み出し

メモリ アレイ インデックスからの読み出しはメモリ アレイから読み出すマスターの意図を示す MOSI ライン上のコマンド バイト (0x02) から始まります。コマンド バイトの受信中、スレーブは (MISO で) 0xfe (Ready) または 0xff (ビジー) を送信します。無効またはリセットされると、スレーブは (MISO で) 0xff (ビジー) を送信します。このバイト値は、マスターがスレーブが SPI リクエストを受け入れる準備ができているかどうかを判断するために使用できます。その後、スレーブはコマンドが登録 (0xFE) または非登録 (0xFF) であることを示すために MISO ライン上に返信バイトを駆動します。MISO 上の追加読み出しのデータ バイトは通信した EZ アドレスによって示される位置でメモリ アレイから読み出されます。バイトがメモリ アレイから読み出されると、EZ アドレスはスレーブによって自動的にインクリメントされます。EZ アドレスがメモリ エントリーの最大数 (256) を超えるとその値を維持し、0 にラップ アラウンドしません。EZ ベース アドレスは、各スレーブ セレクト時に EZ アドレス書き込みフェーズで書き込まれたアドレスにリセットされます。

図 239 に、EZSPI プロトコルでの EZ アドレスの書き込み、メモリ アレイへの書き込み、メモリ アレイからの読み出し動作を示します。

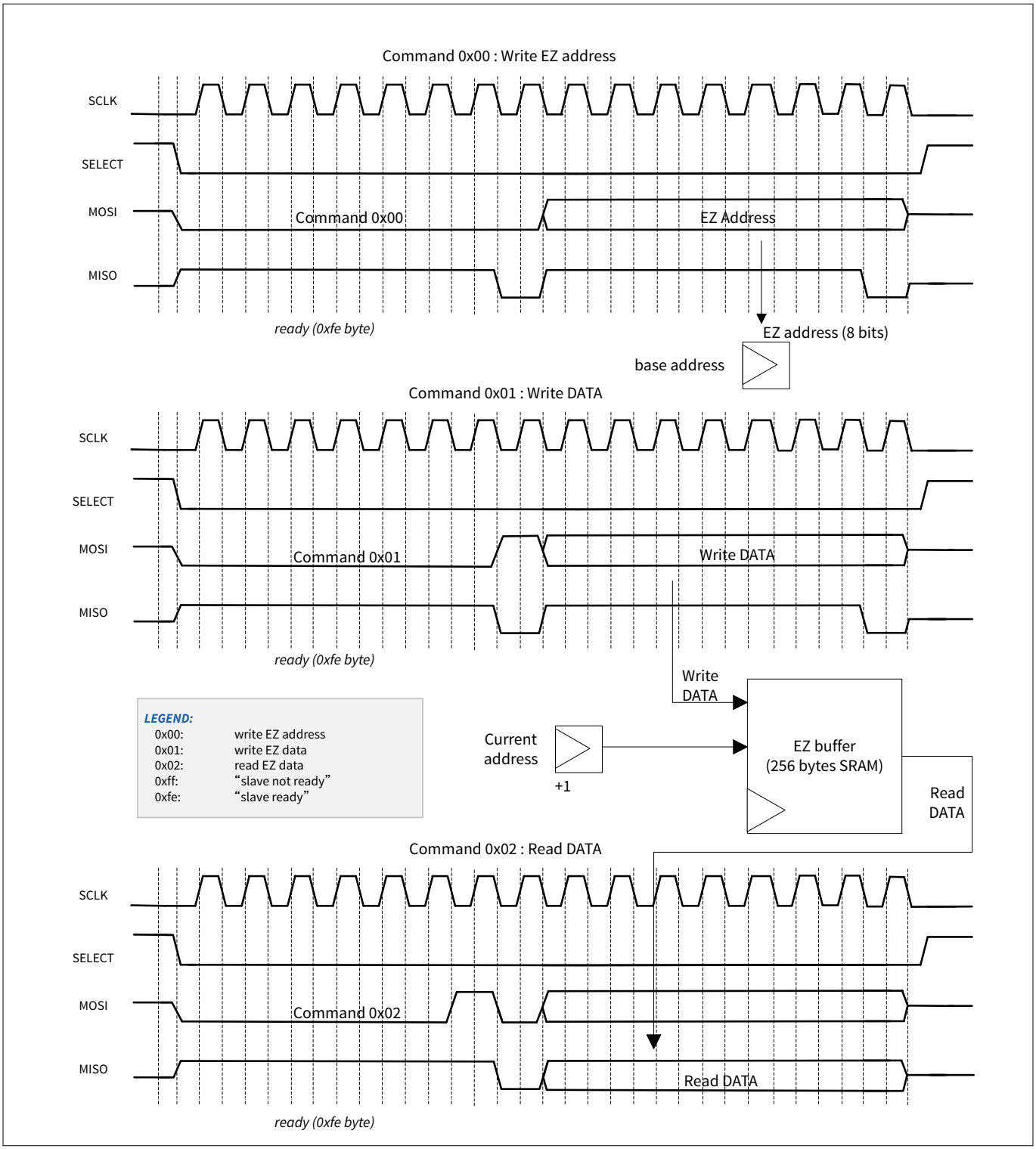


図 239　　EZSPI の例

EZSPI モード用の SCB の設定

デフォルトでは、SCB は非 EZ 動作モードに設定されています。EZSPI モード用に SCB を設定するためにはレジスタビットを次の順序で設定してください。

1. `SCBx_CTRL.EZ_MODE` レジスタに'1'を書き込んで EZ モードを選択してください。
2. Motorola 社製 SPI セクションの手順に従ってください。
3. SPI の有効化と初期化 セクションに記載されているステップ 2～4 に従ってください。

これらのレジスタの詳細については、PSOC™ Control C3 レジスタ リファレンス マニュアルを参照してください。

システム ディープスリープから CPU アクティブへの移行

- EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 0, EZ モード - 内部クロック ロジックが引き継ぐまで、MISO は 0xFF を送信します。MOSI 上のデータは、内部クロック ロジックが引き継ぐまで無視されます。このような場合、PLL/FLL のロック時間により、内部クロックが正しい周波数になる保証はありません。このため、MISO 上や EZ メモリ内のデータが破損する可能性があります。したがって、システムのディープスリープに入る前に CLK_SCB を無効にし、PLL/FLL が安定するまで有効にすることを推奨します。外部マスターは、MISO で 0xFF を読み出すとき、デバイスがまだ準備ができていないことを認識する必要があります。
- EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 1, EZ モード - システムのディープスリープから CPU アクティブモードに移行する際、PLL/FLL のロック時間により内部クロックが正しい周波数になる保証はありません。この状況では、SPI SCLK 周波数は 2 MHz に制限されます。FLL/PLL 出力が安定した後、クロックはより高速に動作します。

***注**: スレーブモードでは SCB[0] のみディープスリープに対応します。*

26.3.4.3 コマンド レスポンス モード

コマンド レスポンス モードは、SPI スレーブに対してのみ定義されます。PSOC™ Control C3 MCU では、1 つの SCB (SCB[0]) のみがコマンド レスポンス モードをサポートします。このモードには、1 つのメモリ バッファ、ベース読み出しアドレス、現在の読み出しアドレス、ベース書き込みアドレス、およびメモリ バッファのインデックス付けに使用される現在の書き込みアドレスがあります。ベースアドレスは CPU が提供します。現在のアドレスは、スレーブがメモリ バッファに順次アクセスする際のインデックスとして使用されます。メモリ バッファは 256 個の 8 ビット データ エレメントを保持します。ベースアドレスと現在のアドレスは[0, 255]の範囲です。

CPU は SCBx_EZ_DATA レジスタをとおしてメモリ バッファに書き込みと読み出しを行います。これらのアクセスはワード アクセスですが、使用されるのはワードの最下位バイトのみです。

スレーブ インターフェースは、現在のアドレスを使用してメモリ バッファにアクセスします。書き込み転送 (SPI スレーブ セレクト) の開始時に、ベース書き込みアドレスが現在の書き込みアドレスにコピーされます。データ エレメントの書き込みは、現在の書き込みアドレス位置に行われます。書き込みアクセス後、現在のアドレスは'1'インクリメントされます。読み出し転送の開始時に、ベース読み出しアドレスが現在の読み出しアドレスにコピーされます。データ エレメントの読み出しは、現在の読み出しアドレス位置までです。読み出しデータ エレメントが送信された後、現在の読み出しアドレスが'1'インクリメントされます。

現在のアドレスが最後のメモリ バッファ アドレス (255) と等しい場合、現在のアドレスはインクリメントされません。それ以降の書き込みアクセスは、最後のバッファ アドレスに以前書き込まれた値を上書きします。それ以降の読み出しアクセスは、最後のバッファ アドレスで (同じ) 読み出し値を提供し続けます。バス マスターは、コマンド レスポンス モードにおけるメモリ バッファの容量を認識しておく必要があります。

ベースアドレスは SCBx_CMD_RESP_CTRL.BASE_RD_ADDR および SCBx_CMD_RESP_CTRL.BASE_WR_ADDR が提供します。現在のアドレスは SCBx_CMD_RESP_STATUS.CURR_RD_ADDR および SCBx_CMD_RESP_STATUS.CURR_WR_ADDR が提供します。転送終了時 (SPI スレーブ セレクト解除時)、ベースアドレスと現在のアドレスの差は、何回読み出し/書き込みアクセスが行われたかを示します。このブロックは、転送の終わりを識別するための割込み要因フィールドを提供します。コマンド レスポンス モードの動作は、CPU アクティブ電力モードとシステム ディープスリープ電力モードで利用できます。

コマンド レスポンス モードには 2 つの動作段階があります。

- **書き込みフェーズ**: 書き込みフェーズは選択バイトで始まり、選択バイトの最後のビットが'0'に設定され、マスターが選択バイトに続いてスレーブのメモリ バッファに 8 ビットのデータ エレメントを書き込むことを示します。スレーブの現在の書き込みアドレスは、スレーブのベース書き込みアドレスに設定されます。受信したデ

ータ エレメントは、現在の書き込みアドレス メモリ位置に書き込まれます。各メモリ書き込みの後、現在の書き込みアドレスがインクリメントされます

- **読み出しフェーズ**: 読み出しフェーズは、読み出しを示す最後のビットが'1'に設定された選択バイトで始まります。マスターは、スレーブのメモリ バッファから 8 ビットのデータ エレメントを読み出します。スレーブの現在の読み出しアドレスは、スレーブのベース読み出しアドレスに設定されます。送信されたデータ エレメントは、現在のアドレス メモリから読み出されます。各読み出しデータ エレメントが転送されるたびに、現在の読み出しアドレスがインクリメントされます。

最初のバイトを受信している間、スレーブ (MISO) は 0x62 (ready) または異なる値 (busy) を送信します。無効またはリセットされると、スレーブは 0xFF (ビジー) を送信します。このバイト値は、マスターがスレーブが SPI リクエストを受け入れる準備ができているかどうかを判断するために使用できます。

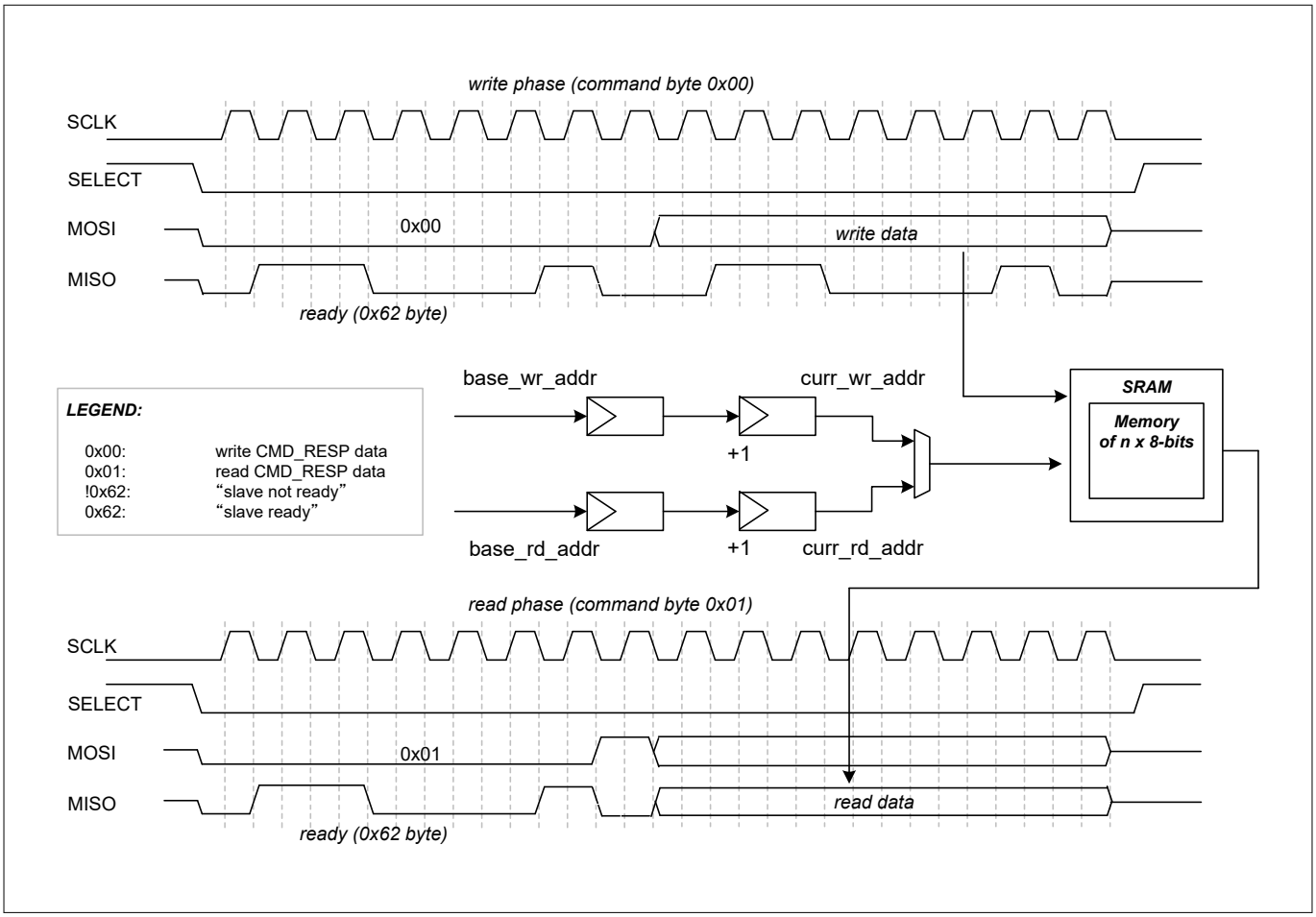


図 240　　コマンド レスポンス モードの例

注: スレーブのベースアドレスは、マスターではなくCPU が更新します。

システム ディープスリープから CPU アクティブへの移行

SCBx_CTRL.EC_AM_MODE = 1, SCBx_CTRL.EC_OP_MODE = 1, CMD_RESP モード。

システム ディープスリープから CPU アクティブモードに移行する際、PLL/FLL のロック時間のため、内部クロックが正しい周波数である保証はありません。この状況では、SPI SCLK 周波数は 2 MHz に制限されます。FLL/PLL 出力が安定した後、クロックはより高速に動作します。

CMD_RESP モード用の SCB の設定

デフォルトでは、SCB は非 CMD_RESP 動作モードに設定されています。CMD_RESP モード用に SCB を設定するためにはレジスタ ビットを次の順序で設定してください。

1. SCBx_CTRL.CMD_RESP_MODE レジスタに'1'を書き込んで CMD_RESP モードを選択してください。

2. Motorola 社製 SPI セクションの手順に従ってください。
3. SPI の有効化と初期化 セクションに記載されているステップ 2～4 に従ってください。

これらのレジスタの詳細については、PSOC™ Control C3 レジスタ リファレンス マニュアルを参照してください。

26.3.5 クロッキングとオーバーサンプリング

26.3.5.1 クロック モード

SCB SPI は、内部クロック動作モードと外部クロック動作モードの両方に対応しています。SCBx_CTRL.EC_AM_MODE および SCBx_CTRL.EC_OP_MODE レジスタは SCB クロック モードを決定します。SCBx_CTRL.EC_AM_MODE は、SPI スレーブ セレクトが内部クロック (0) か外部クロック (1) かを示します。SCBx_CTRL.EC_OP_MODE は、残りのプロトコル動作 (SPI スレーブ セレクト以外) が内部クロック (0) か外部クロック (1) かを示します。

外部クロック動作は、外部マスター (SPI SCLK) から供給されるクロックを使用します。

注: PSOC™ Control C3 MCU では、ディープスリープ対応の SCB のみが外部クロック動作モードをサポートし、SPI スレーブ モードでのみ動作します。

内部クロック動作は、プログラム可能なクロック分周器を使用します。システムクロックの詳細については、クロックシステム セクションを参照してください。

SCBx_CTRL.EC_AM_MODE および SCBx_CTRL.EC_OP_MODE は以下のように設定できます。

- EC_AM_MODE = '0', SCBx_CTRL.EC_OP_MODE = '0': アクティブ モードの機能のみが必要な場合は、この設定を使用します。
  - FIFO モード: 対応
  - EZ モード: 対応
  - コマンド レスポンス モード: 非対応。スレーブ (MISO) は、このコンフィギュレーションでコマンド レスポンス モードが試行された場合、最初のバイトの受信中に Ready (0x62) バイトとは異なる値を送信します。
- EC_AM_MODE = '1', SCBx_CTRL.EC_OP_MODE = '0': CPU アクティブ機能とシステムディープスリープ機能の両方が必要な場合は、この設定を使用してください。この構成は、スレーブ セレクトを検出するために外部クロック機能に依存し、メモリバッファにアクセスするために内部クロック機能に依存します。

外部機能から内部機能への「ハンドオーバー」は、ビジー/Ready バイト方式に依存しています。この方式は、ビジー バイトを受信したときにマスターが現在の転送を再試行することに依存しており、マスターがビジー/Ready バイトの解釈をサポートする必要があります。スレーブが選択された場合、SCBx_INTR_SPI_EC.WAKE_UP = '1'に設定されます。関連するシステム ディープスリープ機能割込みにより、システムは CPU アクティブ電力モードになります。

- FIFO モード: 対応。スレーブ (MISO) は、CPU が復帰し、TX FIFO が有効化されるまで 0xFF を送信します。CLK_SCB が有効化されるまで、MOSI ライン上のデータはすべて破棄されます (詳細は、システム ディープスリープから CPU アクティブへの遷移を参照)。
  - EZ モード: 対応。システム ディープスリープ電力モードでは、スレーブ (MISO) はコマンド バイトの受信中にビジー (0xFF) バイトを送信します。CPU アクティブ電力モードでは、スレーブ (MISO) はコマンドバイトの受信中に Ready (0xFE) バイトを送信します。
  - CMD_RESP モード: 非対応。スレーブは、最初のバイトの受信中に、Ready (0x62) バイトとは異なる値を送信 (MISO) します。
- EC_AM_MODE = '1 および SCBx_CTRL.EC_OP_MODE = '1'の場合。CPU アクティブ機能とシステム ディープスリープ機能の両方が必要な場合は、このモードを使用します。スレーブが選択された場合、SCBx_INTR_SPI_EC.WAKE_UP = '1'に設定されます。関連するシステム ディープスリープ機能割込みにより、シス

テムは CPU アクティブ電力モードになります。スレーブが選択解除された場合、SCBx_INTR_SPI_EC.EZ_STOP = '1'および/または SCBx_INTR_SPI_EC.EZ_WRITE_STOP = '1'に設定されます。

- FIFO モード: 非対応
- EZ モード: 対応
- CMD_RESP モード: 対応

表 1 SPI モードの互換性

	内部クロック (IC)			外部クロック (EC)		
	FIFO	**EZ**	**CMD_RESP**	**FIFO**	**EZ**	**CMD_RESP**
SPI マスター	有り	無し	無し	無し	無し	無し
SPI スレーブ	有り	有り	無し	無し [1)]	有り	有り

1) SPI スレーブ FIFO モードでは、外部クロック駆動のロジックが選択検出を行い、その後割込みを発生させて CPU をウェイクアップさせます。書き込みは無視され、読み出しは CPU が Ready になり FIFO にデータが入るまで 0xFF を返します。

SCBx_CTRL.EC_OP_MODE = '1'の場合、外部インターフェース ロジックは外部インターフェース クロック (SPI SCLK) でメモリ バッファにアクセスします。これにより、CPU アクティブおよびシステム ディープスリープの電力モードで、EZ および CMD_RESP モード機能を使用できます。

CPU アクティブ システム電力モードでは、メモリ バッファは外部インターフェース ロジック (SPI SCLK 上) と CPU インターフェース ロジック (システム周辺クロック上) の間のアービトレーションを必要とします。このアービトレーションは、常に外部インターフェースロジック (ホストアクセス) を最優先にします。外部インターフェース ロジックは、SPI 用に 2 つのシリアル インターフェース クロック/ビット期間を取ります。この間、内部ロジックはメモリ バッファへのアクセスを拒否されます。PSOC™ Control C3 MCU は、この「サービス拒否」に対処するために 2 つのプログラム可能なオプションを提供しています。

- SCBx_CTRL.BLOCK = 1 の場合: メモリ バッファが許可され、外部インターフェース ロジックがアクセスを完了するまで、内部ロジックのメモリ バッファへのアクセスはブロックされます。このオプションは通常の SCB レジスタ機能を提供しますが、ブロッキング時間により内部バス待機状態が追加されます。
- SCBx_CTRL.BLOCK = 0 の場合: メモリ バッファへの内部ロジック アクセスはブロックされませんが、外部インターフェースのロジック アクセスと競合すると失敗します。読み出しアクセスは値 0xFFFF:FFFF を返し、書き込みアクセスは無視されます。このオプションは、追加の内部バス待機状態を導入しませんが、メモリ バッファへのアクセスが有効にならない可能性があります。この場合、以下の障害が検出されます。
  - 読み出し失敗: 返される値が 0xFFFF:FFFF であるため、読み出し失敗は簡単に検出できます。この値は、失敗しないメモリ バッファ読み出しアクセスが 0x0000:0000-0x0000:00ff の範囲の符号なしバイト値を返すため、一意です。
  - 書き込み失敗: 書き込み失敗は、書き込まれたメモリ バッファの位置を読み返し、読み出された値が書き込まれた値と同じであることを確認することによって検出されます。

どちらのオプションでも、メモリ バッファへの競合する内部ロジック アクセスは SCBx_INTR_TX.BLOCKED フィールドを'1' (書き込みアクセス用) に、SCBx_INTR_RX.BLOCKED フィールドを'1' (読み出しアクセス用) に設定します。これらのフィールドは、ステータス フィールドとしても割込み要因フィールド (関連するマスク フィールドが有効な場合) としても使用できます。

一連の読み出しまたは書き込みアクセスが実行され、SCBx_CTRL.BLOCK が'0'の場合、すべての読み出し値の論理和を 0xFFFF:FFFF と比較し、SCBx_INTR_TX.BLOCKED と SCBx_INTR_RX.BLOCKED フィールドをチェックして、(一連の) 書き込みまたは読み出し操作で障害が発生したかどうかを判断することにより、障害が検出されます。

26.3.5.2 SPI マスターを使用してスレーブにクロックを供給する

通常の SPI マスター モード伝送では、SCLK は SCB が有効でデータが伝送されている時にのみ生成されます。SCB が有効な限り SCLK ライン上にクロックを常に生成するように変更できます。

スレーブが SPI 機能のみでなく他の機能でも SCLK を使用する時にこのオプションが使用されます。これを有効にするためには、SCBx_SPI_CTRL.SCLK_CONTINUOUS に'1'を書き込んでください。

26.3.5.3 オーバーサンプリングとビットレート

SPI マスターモード

SPI マスターは外部クロック モードをサポートしません。内部クロック モードでは、ロジックは内部クロックで動作します。内部クロックはインターフェース クロック (SCLK) よりも周波数が高く、マスターは入力信号 (MISO) をオーバーサンプリングできます。

SCBx_CTRL.OVS レジスタはオーバーサンプリングを指定します。オーバーサンプリングレートは SCBx_CTRL.OVS レジスタの値+1 として計算されます。SPI マスター モードでは有効な範囲は 4～16 です。MOSI のみ (送信のみ) の SPI マスターの場合、有効範囲は 2～16 です。ビットレートは以下のように計算されます。

ビットレート = 入力クロック/SCBx_CTRL.OVS

したがって、入力クロックが 100 MHz の場合、最大ビットレートは MISO ありで 25 Mbps、MISO なしで 50 Mbps となります。

***注**:* *50 Mbps のビットレートを使用する場合、潜在的な準安定状態を避けるため、以下のパラメーターが設定されていることを確認してください。*

- *CLK_SCB: 200 MHz*
- *OVS: 3*
- *MEDIAN: 1*

上記の値は、SCB ハードウェアが SCLK を実行できる速度を示します。しかし、マスターがその速度でスレーブからデータを正しく受信できるかどうかを示すものではありません。これを求めるには、以下の式を使って MISO のパス遅延を計算します。

$1/2 \times t_{SCLK} \geqq t_{SCLK_PCB_D} + t_{DSO} + t_{SCLK_PCB_D} + t_{DSI}$ の場合

ここで、

- t_{SCLK} は SPI クロックの時間です
- $t_{SCLK_PCB_D}$ はマスターからスレーブへの SCLK PCB 遅延時間
- t_{DSO} は内部スレーブ遅延の合計で、スレーブ端子の SCLK エッジからスレーブ端子の MISO エッジまでの時間
- $t_{SCLK_PCB_D}$ はスレーブからマスターへの MISO PCB 遅延 t_{DSI} はマスター セットアップ時間

ほとんどのスレーブのデータシートには t_{DSO} と記載されています。名称が異なる場合があるため、SCLK エッジ後の MISO 出力有効時間を確認してください。

ほとんどのマスターのデータシートには、t_{DSI} またはマスターのセットアップ時間も記載されています。特定の PCB ジオメトリに基づいて $t_{SCLK_PCB_D}$ と $t_{SCLK_PCB_D}$ を計算します。

これらの計算を行っても希望の速度が得られない場合は、SCB の MISO 遅延サンプル機能の使用を検討します。この機能は、SCB に次回の SCLK エッジで入力される MISO 信号をサンプリングするよう指示し、これにより SCLK サイクルの半分に相当する追加のタイミング マージンを実現します。

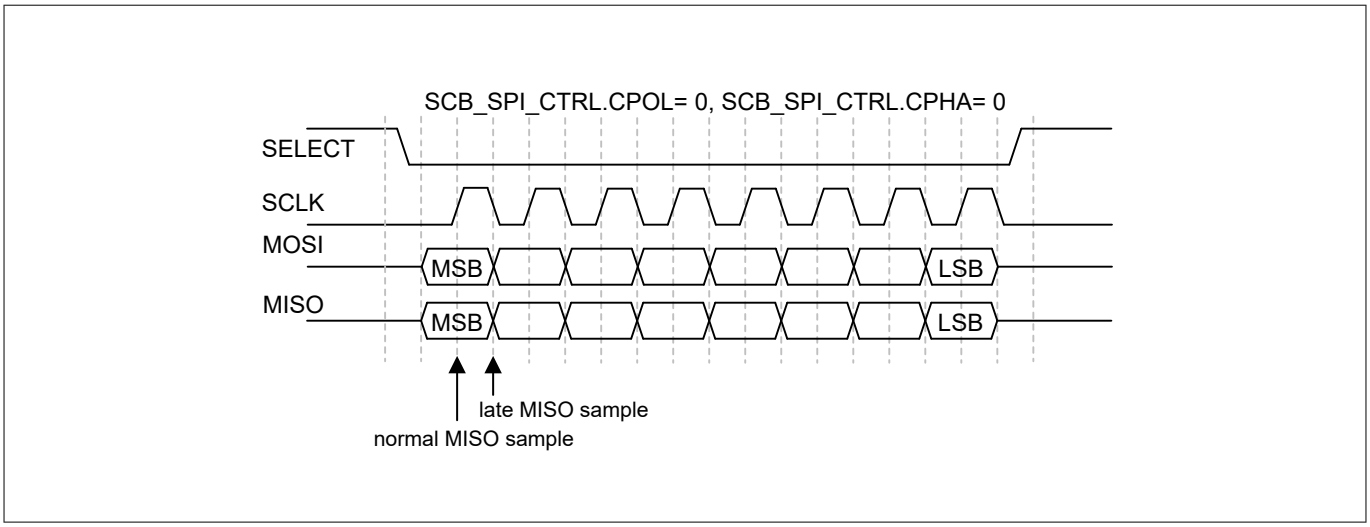


図 241　　MISO サンプリングタイミング

これは方程式を次の様に変えます。

$t_{SCLK} \geq t_{SCLK_PCB_D} + t_{DSO} + t_{SCLK_PCB_D} + t_{DSI}$

遅延サンプルはよりタイミング改善に役立つので、常に有効にしておいてください。

デバイス データシートの t_{DSI} 仕様は、遅延サンプルが有効であることを前提としています。

***注**:　　SCBx_SPI_CTRL.LATE_MISO_SAMPLE はデフォルトで'1'に設定されています。*

SPI スレーブモード

SPI スレーブモードでは、SCBx_CTRL.OVS レジスタは使用されません。データレートは先の式で決まります。遅延 MISO サンプルは、SCBx_SPI_CTRL.LATE_MISO_SAMPLE ではなく、外部マスターによって決定されます。

PSOC™ Control C3 MCU の場合、tDSO はデバイスのデータシートに記載されています。内部クロック モードでは、内部クロックの周波数に比例します。例えば、

TDSO_EXT = TDSO_INT + 3 × Tclk_scb

TDSO_EXT: SCLK 駆動エッジ (外部 SPI マスター クロック) 後に有効な出力データ

TDSO_INT: 外部 SPI マスターが LATE_SAMPLE を持っていない場合

***注**:　　50 Mbps のビットレートを使用する場合、潜在的な準安定状態を避けるため、以下のパラメーターが設定されていることを確認してください。*

- *CLK_SCB: 200 MHz*
- *OVS: 4*
- *MEDIAN: 1*

26.3.6　　SPI マスター SELECT 出力タイミング制御

SPI マスターの SELECT 出力信号の"セレクト"タイミングは可変です。これは以下に適用されます。

- セレクト セットアップ時間 (セレクト アクティブから SPI クロックまで)
- セレクト ホールド時間 (SPI クロックからセレクト インアクティブまで)
- データフレーム間の選択解除時間 (セレクトインアクティブからセレクトアクティブまで)

これらの遅延には以下のオプションが選択できます。

- SELECT セットアップ時間
  - SCBx_SPI_CTRL.CPHA = 0 の場合: 0.75 または 1.75 SPI クロック サイクル (長いセットアップ時間を必要とする SPI スレーブ用)
  - SCBx_SPI_CTRL.CPHA = 1 の場合: 0.25 または 1.25 SPI クロック サイクル (長いセットアップ時間を必要とする SPI スレーブ用)
- SELECT ホールド時間 (SCBx_SPI_CTRL.SSEL_HOLD_DEL レジスタ)
  - SCBx_SPI_CTRL.CPHA = 0 の場合: 0.25 または 1.25 SPI クロック サイクル (長いセットアップ時間を必要とする SPI スレーブ用)
  - SCBx_SPI_CTRL.CPHA = 1 の場合: 0.75 または 1.75 SPI クロック サイクル (長いセットアップ時間を必要とする SPI スレーブ用)

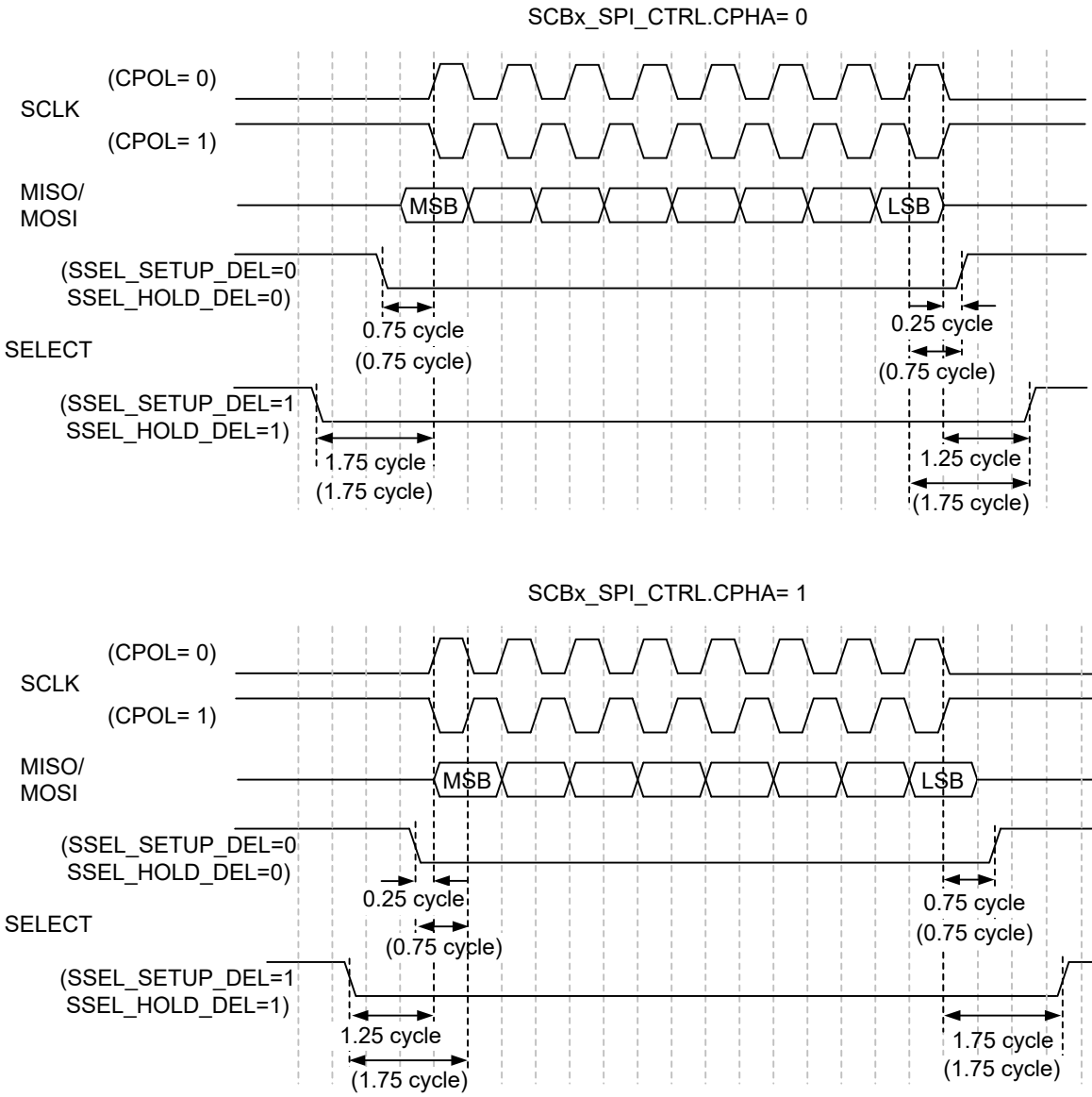


図 242　　セレクト セットアップ/ホールド遅延

- フレーム間の選択解除時間 (`SCBx_SPI_CTRL.SSEL_INTER_FRAME_DEL` レジスタ):
  - 1.5SPI クロック サイクルまたは 2.5SPI クロック サイクル (より長い選択解除時間を必要とする SPI スレーブ用)

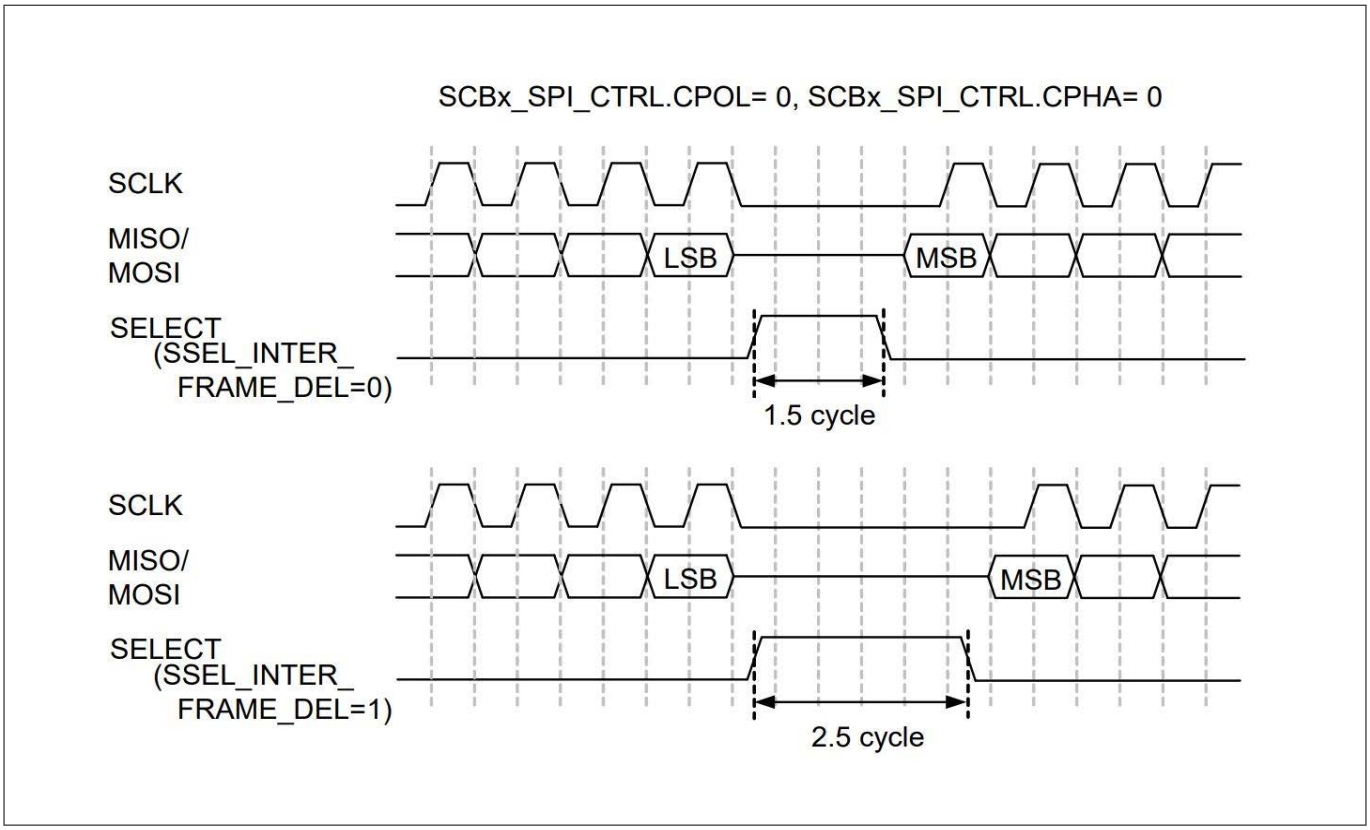


図 243　　フレーム間セレクト時間選択解除

26.3.7　SPI パリティ機能

SPI モードにおけるパリティ機能の特長は以下のとおりです。

- 内部クロック動作の SPI マスターおよび SPI スレーブに適用。
- データ フレームにパリティ ビットを追加し、1 ビットのデータ フレーム エラーを識別するために使用されます。パリティ ビットは、データ フレーム ビットに直接続きます。
- SCBx_SPI_TX_CTRL.PARITY_ENABLED および PARITY_ENABLED を個々に使用して有効または無効に設定できます。
- 送信時に、パリティビットを挿入できます。受信時に、パリティビットをチェックできます。パリティに失敗した場合、受信したデータを RX FIFO に送るか、ドロップして失うかを、DROP_ON_PARITY_ERROR レジスタを使用して選択できます。さらに、SCBx_INTR_RX の PARITY_ERROR ビットが設定されます。
- 偶数パリティと奇数パリティをサポート (SCBx_SPI_TX_CTRL.PARITY, SCBx_SPI_RX_CTRL.PARITY)

26.3.8　ループバック

SPI マスター モードでは、SCB はピンの情報に影響を与えることなく、MOSI の出力信号から MISO の入力信号への内部ループバックをサポートします。SCBx_SPI_CTRL.LOOPBACK レジスタを使用して設定します。

このループバックは National Semiconductor 社製モードではサポートされていません。

26.3.9　SPI の有効化と初期化

SPI を設定するためには、次の手順を実行します。

1. SCBx_SPI_CTRL レジスタを使用してプロトコル固有の情報をプログラムしてください。これには、プロトコルのサブモード (MODE)、マスター スレーブ機能 (MASTER_MODE)、4 つの SELECT (SSEL) のうちの 1 つ、

SELECT が転送全体にわたってアクティブなままか、データ フレーム幅ごとにアクティブなままか (SSEL_CONTINUOUS)、SELECT の極性 (SSEL_POLARITY 0～3) の選択などが含まれます。EZSPI および CMD_RESP はスレーブモードでのみ使用できます

2. SCBx_TX_CTRL および SCBx_RX_CTRL レジスタを使用して汎用送受信側情報をプログラムしてください。
   a. データフレーム幅を指定します。EZSPI では常に 8 とします
   b. 最初に送信/受信されるビットが MSb であるか LSb であるかを指定してください。EZSPI および CMD_RESP の場合、常に MS ビットを先頭に配置する必要があります
3. SCBx_TX_FIFO_CTRL/SCBx_RX_FIFO_CTRL で示したように、SCBx_TX_FIFO_CTRL および SCBx_RX_FIFO_CTRL レジスタを使用して送受信側 FIFO をプログラムしてください。FIFO モードでのみ以下を有効にしてください。
   a. トリガーレベルの設定 (TRIGGER_LEVEL)
   b. 送信側と受信側 FIFO、シフト レジスタのクリア (CLEAR)
4. ブロックの有効化 (SCBx_CTRL.ENABLED レジスタに'1'を書き込み)。ブロックが有効になった後は、制御ビットを変更しないでください。例えば、動作モードを変更したり (Motorola 社製モードから Texas Instruments 社製モードへ)、外部クロック動作から内部クロック動作へ変更する場合などです。変更はブロックを再び有効にしなければ有効になりません。ブロックを再有効化すると再初期化され、関連する状態が失われることに注意してください (例えば、FIFO の内容)。

26.3.10 高速 SPI マスター/スレーブ サポート

SCB[5]*は最大 50 MHz の高速 SPI 動作が可能です。サポートされるモードは、SPI マスター、内部クロックによる SPI スレーブ、外部クロックによる SPI スレーブです。

高速 SPI 動作の場合、clk_peri は 200～240 MHz の周波数で動作する必要があります。

26.3.11 I/O パッド接続

26.3.11.1 SPI マスター

SPI マスター モードでは、SCB はデータ送信およびデータ受信機能を提供します。図 244 と表 236 に、SPI マスター用の I/O パッドの使用法を示します。

通常、出力信号には強ドライブ モード (GPIO_PRTx_CFG.DRIVE_MODEy = 6) が使用されます。SCB が無効の場合、それぞれの out_en 信号は 0 となり、出力は High-Z となります。High-Z 状態を避けるには、以下のいずれかを実行してください。

- GPIO を使用して出力をアイドル レベルに駆動
- 内部プルアップ/プルダウン抵抗によるプルアップ/プルダウンドライブモード使用 (抵抗値固定)
- ストロング駆動モードを使用し、外部プルアップ/プルダウン抵抗 (抵抗値は任意) を適用

内部および外部のプルアップ/プルダウン抵抗は、最大データ レートにマイナスの影響を与えます。

通常の全二重モードの SPI MISO 入力では、SPI スレーブ デバイスが選択されていない場合、その MISO 出力は High-Z となります。SPI マスターに接続されているすべての SPI スレーブ デバイスが選択されていない場合、MISO ラインは High-Z となります。High-Z 状態を避けるには、MISO ラインにプルアップ抵抗が必要です。

- ハイインピーダンス (High-Z) 駆動モード: 外部プルアップ/プルダウン抵抗使用
- プルアップ/プルダウン駆動モード: 内部プルアップ/プルダウン抵抗使用
- 内部および外部のプルアップ/プルダウン抵抗は、最大データ レートに悪影響を及ぼします。

半二重モードは、通常の機能モード (spi_mosi_out_en = 1 のとき) では「強プルアップ、強プルダウン」を駆動するため、サポートされていません。

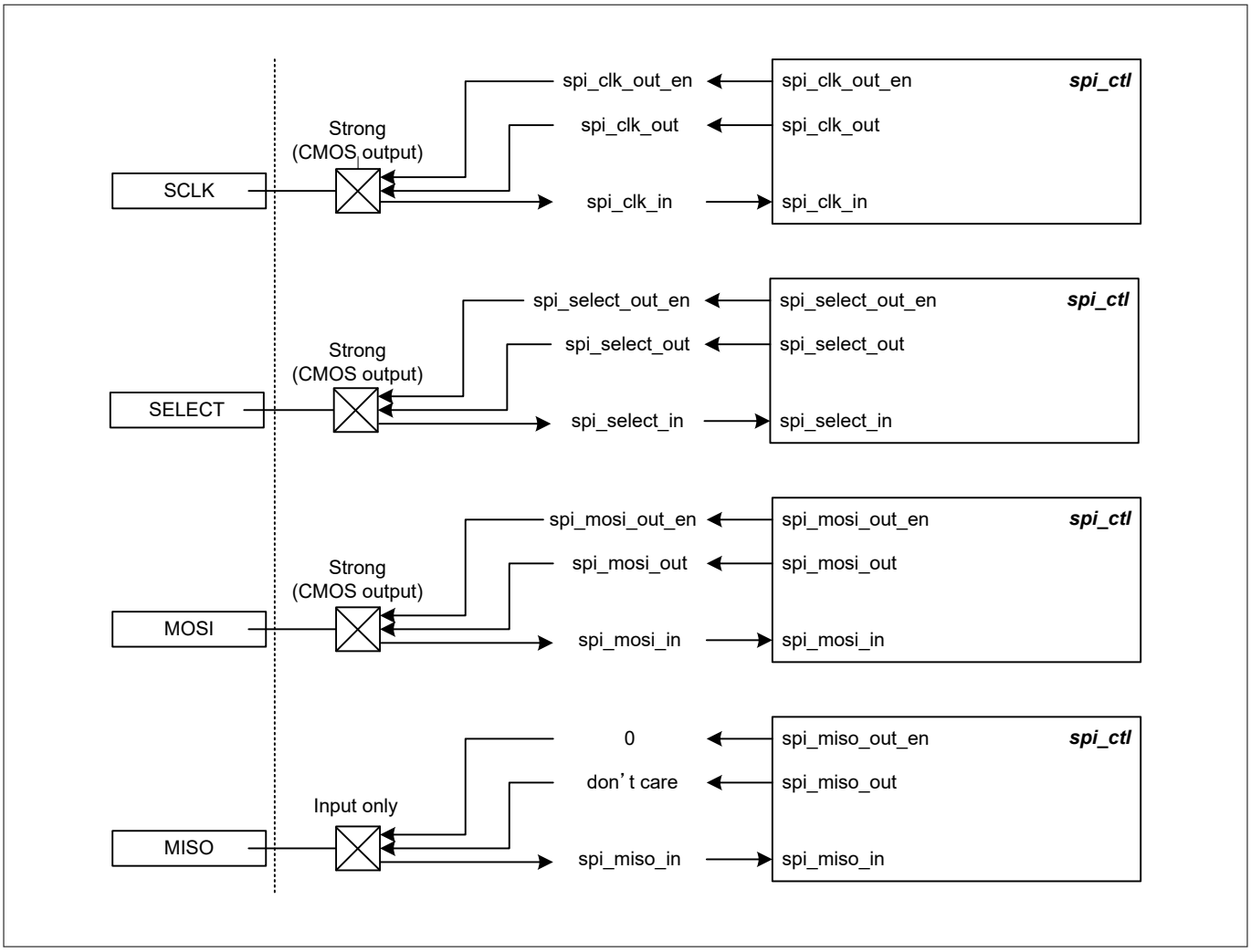


図 244　　SPI マスター I/O パッド接続

表 236　　SPI マスター I/O パッド接続の使用方法

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
SCLK	ストロング (CMOS 出力)	spi_clk_out_en spi_clk_out	クロック信号の送信
SELECT	ストロング (CMOS 出力)	spi_select_out_en spi_select_out	セレクト信号の送信
MOSI	ストロング (CMOS 出力)	spi_mosi_out_en spi_mosi_out	データ エレメントの送信
MISO	入力のみ	spi_miso_in spi_miso_out spi_miso_out_en	データ エレメントの受信

26.3.11.2　SPI スレーブ

SPI スレーブ モードでは、SCB はデータ送信とデータ受信機能を提供します。図 13 と表 3 に、SPI スレーブ用 I/O パッドの使用法を示します。

通常、出力信号には強ドライブ モード (GPIO_PRTx_CFG.DRIVE_MODEy = 6) が使用されます。SCB が無効になると、それぞれの out_en 信号は'0'になります。したがって、出力は High-Z となります。High-Z 状態を避けるには、以下のいずれかを実行してください。

- GPIO を使用して出力をアイドル レベルに駆動
- プルアップまたはプルダウン駆動モードを使用し、内部プルアップ/プルダウン抵抗 (固定抵抗値) を適用
- ストロング駆動モードを使用し、外部プルアップ/プルダウン抵抗 (抵抗値は任意) を適用

内部および外部のプルアップ/プルダウン抵抗は、最大データ レートにマイナスの影響を与えます。

SCBx_TX_CTRL.OPEN_DRAIN = 1 の時、MOSI と MISO を短絡して半二重モードで動作できます。

- MISO 出力のドライブ モードはオープンドレイン ドライブ Low のみです。
- MISO ラインに外部プルアップ抵抗を追加します。

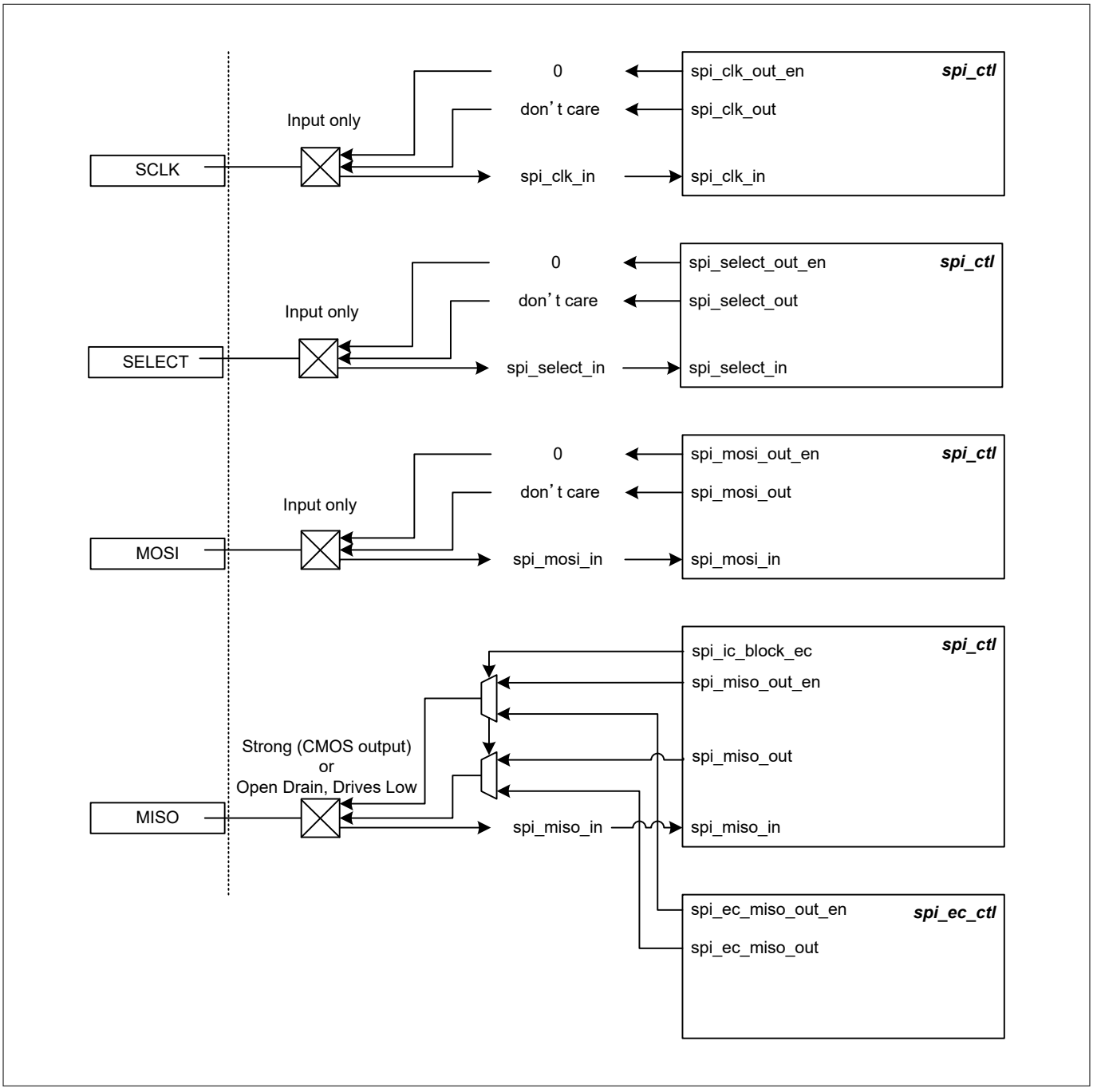


図 245 SPI スレーブ I/O パッド接続

表 237 SPI スレーブ I/O 信号の説明

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
SCLK	入力のみ	spi_clk_out_en spi_clk_out spi_clk_in	クロック信号の受信
SELECT	入力のみ	spi_select_out_en spi_select_out spi_select_in	セレクト信号の受信
MOSI	入力のみ	spi_mosi_out_en spi_mosi_out spi_mosi_in	データ エレメントの受信
MISO	ストロング (CMOS 出力)、またはオープンドレインドライブ Low	spi_miso_out_en spi_miso_out spi_miso_in	データ エレメントの送信

26.3.12 レジスタの概要

SPI インターフェースは、32 ビットのコントロール レジスタとステータス レジスタを使用して制御されます。これらのレジスタに関する詳細は、PSOC™ Control C3 レジスタ RM を参照してください。

26.4 UART

汎用非同期受信側/送信側 (UART) プロトコルは非同期のシリアルインターフェース プロトコルです。

UART 通信は通常ポイント ツー ポイントです。UART インターフェースは 2 つの信号で構成されます。

- TX: 送信側出力
- RX: 受信側入力

さらに、UART のフロー制御を実装するために、2 つのサイドバンド信号が使用されます。フロー制御は TX 機能のみに適用されることに注意してください。

- クリア トゥ センド (CTS): これは送信側への入力信号です。アクティブの場合、受信側がデータを受信する準備ができたことを送信側に知らせ、送信側がデータを送信できることを意味します。
- 送信要求 (RTS): これは受信側からの出力信号です。アクティブの場合、受信側がデータ受信の準備ができていることを示します。
- 一方のデバイスの RTS はもう一方のデバイスの CTS に接続され、その逆も同様です。あるデバイスがデータを受信する準備ができると (受信側)、そのデバイスは RTS をアクティブにし、それが今度は他のデバイス (送信側) の CTS をアクティブにして、データを送信するように合図します。

注: フロー制御は UART 標準プロトコル モードでのみ利用可能で、以下のモードでは利用できません - UART ローカル相互接続ネットワーク (LIN)、スマートカード (ISO7816)、赤外線データアソシエーション (IrDA)、および UART マルチプロセッサ モード。

26.4.1 機能

本 SCB は、以下の機能をサポートします。

- UART プロトコルに対応
  - 標準 UART (全二重、半二重)
  - マルチプロセッサモード
- スマートカード (ISO7816) リーダー
- IrDA
- ローカル インターコネクト ネットワーク (LIN) に対応
  - ブレーク検出
  - ボーレート検出
  - 衝突検出 (駆動するビット値がバスに反映されず、他のコンポーネントが同じバスを駆動していることを検出する)
- データ フレーム サイズは 4 ビット～16 ビットにプログラム可能
- プログラマブルな STOP ビット数 (1 から 7 までのビット半周期単位で設定可能)
- パリティをサポート (奇数パリティと偶数パリティ)
- プログラム可能なオーバーサンプリング
- スタートスキップ
- ハードウェア フロー制御

26.4.2 概要

図 246 に、UART の TX と RX の転送を示します。

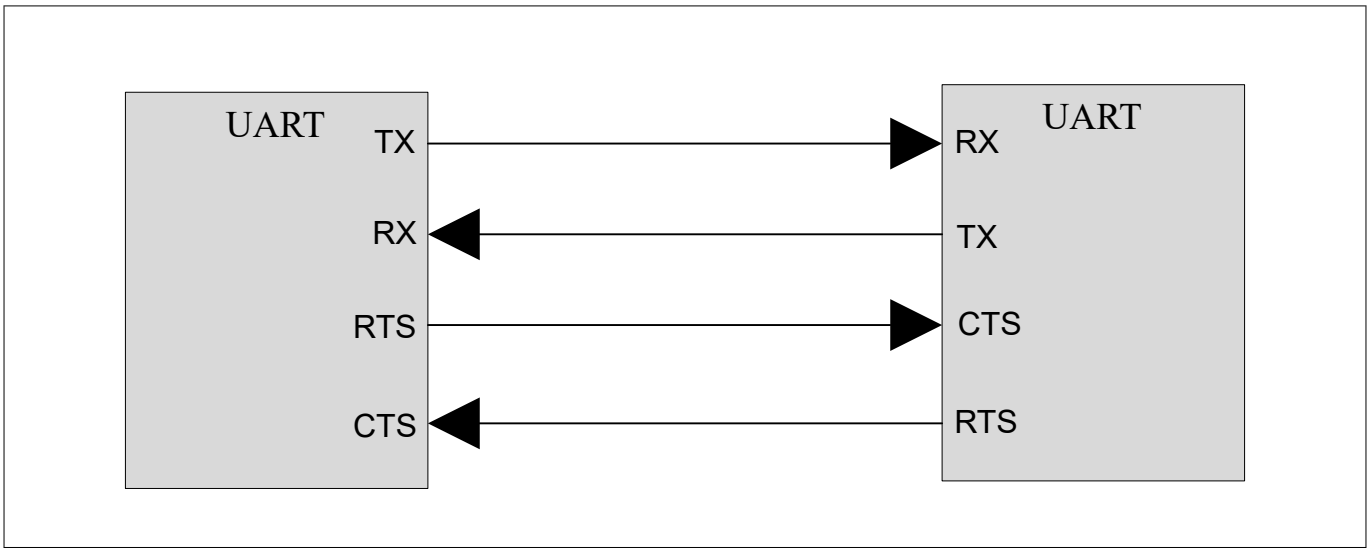


図 246 UART 例

典型的な UART 転送はスタート ビットから始まりその後に複数のデータ ビットとパリティ ビット (任意) が続き 1 つ以上のストップ ビットで終わります。スタートとストップ ビットはそれぞれデータ送信の開始と終了を示します。パリティ ビットは送信側によって送信されシングル ビット エラーを検出するために受信側によって使用されます。インターフェースはクロックを持たないため (非同期)、送信側と受信側はそれぞれのクロックを使用します。

UART はデフォルトで 8 ビットのデータ フレーム幅をサポートします。ただし、送信 (TX) と受信 (RX) のデータ フレーム幅は、4 ビットから 9 ビットの範囲で任意の値に設定できます。これには、スタート、ストップ、およびパリティ ビットは含まれません。ストップ ビット数は 1～7 の範囲で指定できます。パリティ ビットは有効にも無効にもできます。有効な場合、パリティのタイプは偶数パリティまたは奇数パリティに設定できます。パリティ ビットは標準 UART モードと SmartCard UART モードでのみ使用可能です。IrDA UART モードでは、パリティ ビットは自動的に無効化されます。

注: UART インターフェースは外部クロック動作をサポートしていません。そのため UART はアクティブとスリープのシステム電力モードでのみ動作します。また UART は FIFO バッファ モードのみをサポートします。

26.4.3 UART 動作モード

26.4.3.1 標準プロトコル

典型的な UART 転送はスタート ビットから始まりその後に複数のデータ ビットとパリティ ビット (任意) が続き 1 つ以上のストップ ビットで終わります。スタート ビットの値は常に‘0’でデータ ビットの値は転送されるデータに依存し、パリティ ビットの値はデータ ビット上の偶数または奇数パリティを保証する値にセットされ、ストップ ビットの値は‘1’です。パリティ ビットは送信側によって生成され、シングル ビット送信エラーを検出するために受信側を使用できます。データの送信中でない時、TX ラインは‘1’ (ストップ ビットと同じ値) です。

インターフェースにはクロックがないため、送信側と受信側はボーレートを合意する必要があります。送信側と受信側には、それぞれの内部クロックがあります。受信側クロックがビット転送周波数より高い周波数で動作するため、受信側は受信信号をオーバーサンプリングすることがあります。

ストップ ビットからスタート ビットへの移行は TX ライン上の‘1’から‘0’への変更で示されます。このトランジションは、受信側が送信側クロックに同期するために使用できます。各データ転送の開始時の同期化により、送信側と受信側クロック間で周波数ドリフトが存在する場合でも、エラーのない転送が可能です。必要なクロック精度はデータ転送サイズに依存します。

連続したデータ転送間のストップ周期とストップ ビット数は通常送信側と受信側間で合意され 1～3 ビットの転送周期の範囲内にあります。

図 247 に、UART プロトコルを示します。

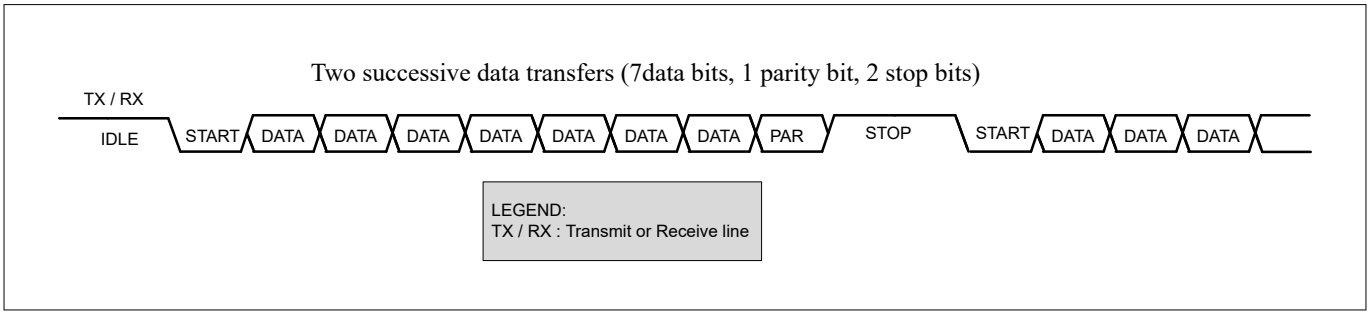


図 247 UART 標準プロトコル例

受信側は入力信号をオーバーサンプリングし、ビット転送期間の中間 (受信側のクロック上) のサンプル ポイントの値が使用されます。これを 図 248 に示します。送受信クロック間のわずかなドリフトに注意してください。

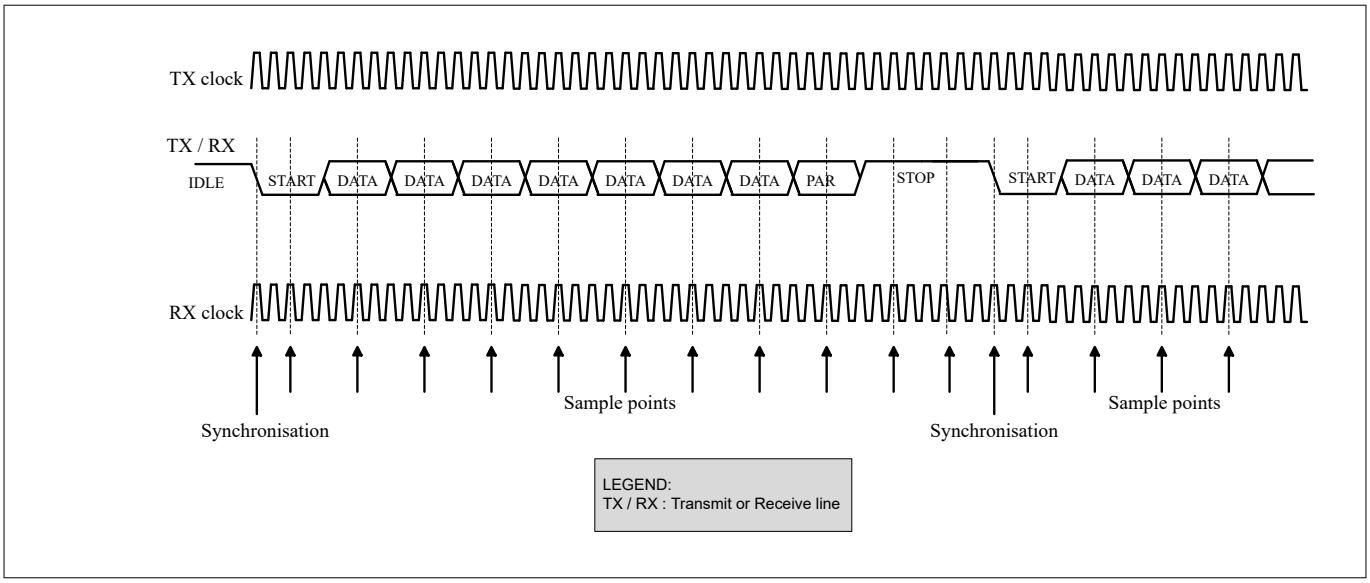


図 248　　UART 標準プロトコル オーバーサンプリング (RX クロックが TX クロックより速い)

また、ビット転送期間の中央付近の 3 サンプル (受信側のクロック) を多数決に使用することで、ノイズ感度を下げ、精度を上げることもできます。これは、SCBx_RX_CTRL レジスタで MEDIAN フィルターを有効にすることで実現されます。

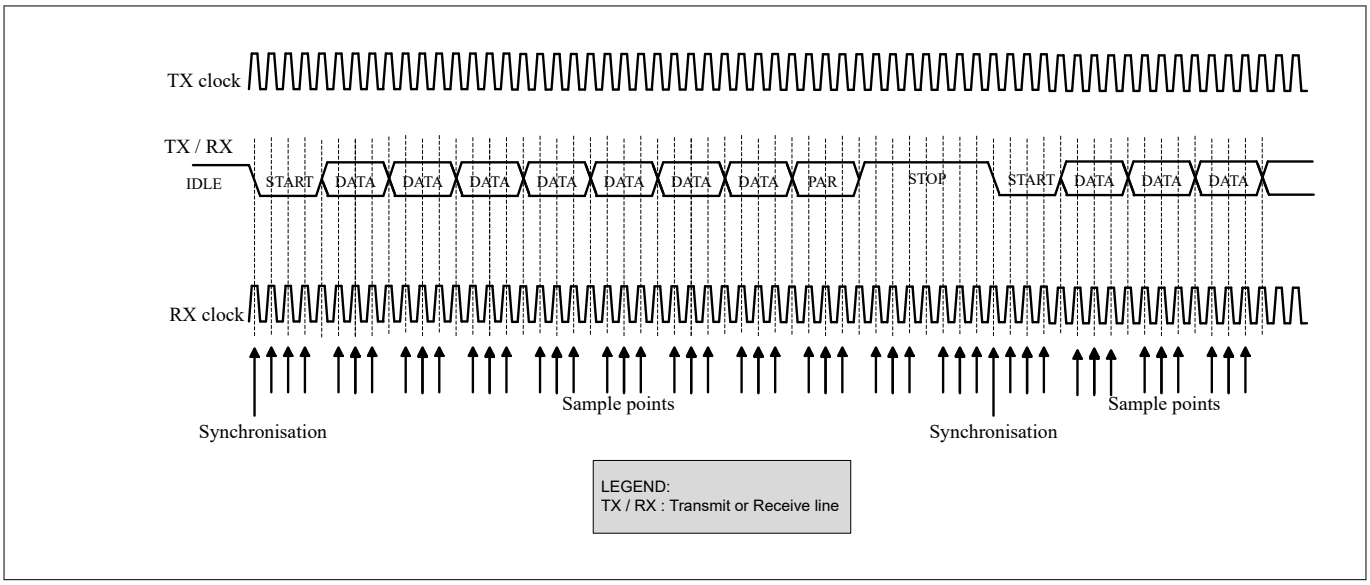


図 249　　UART 標準プロトコル オーバーサンプリング (RX クロックと TX クロックが等しい)

26.4.3.1.1　パリティ

この機能は、データ フレームにパリティ ビットを追加し、1 ビットのデータ フレーム エラーを識別するために使用されます。パリティ ビットは常にデータ フレーム ビットの直後にあります。

送信側はデータフレームビットからパリティビット (SCBx_UART_TX_CTRL.PARITY_ENABLED が'1'の場合) を計算し、データフレームビットとパリティビットが偶数 (SCBx_UART_TX_CTRL.PARITY が'0') または奇数 (SCBx_UART_TX_CTRL.PARITY が'1') のパリティを持つようにします。受信側は、データフレームビットとパリティビットが偶数 (SCBx_UART_RX_CTRL.PARITY が '0') または奇数 (SCBx_UART_RX_CTRL.PARITY が '1') のパリティを持つように、受信したデータフレームビットからパリティビット (SCBx_UART_RX_CTRL.PARITY_ENABLED が '1'の場合) をチェックします。

- パリティは TX と RX の両機能に適用され、専用の制御フィールドが利用可能
  - TX 機能: SCBx_UART_TX_CTRL.PARITY および SCBx_UART_TX_CTRL.PARITY_ENABLED
  - RX 機能: SCBx_UART_RX_CTRL.PARITY および SCBx_UART_RX_CTRL.PARITY_ENABLED
- 受信側がパリティエラーを検出すると、データフレームは RX FIFO に入れられるか (SCBx_UART_RX_CTRL.DROP_ON_PARITY_ERROR は'0')、ドロップされます (SCBx_UART_RX_CTRL.DROP_ON_PARITY_ERROR は'1')。

図 250 に、パリティ機能 (8 ビットデータフレーム) を示します。

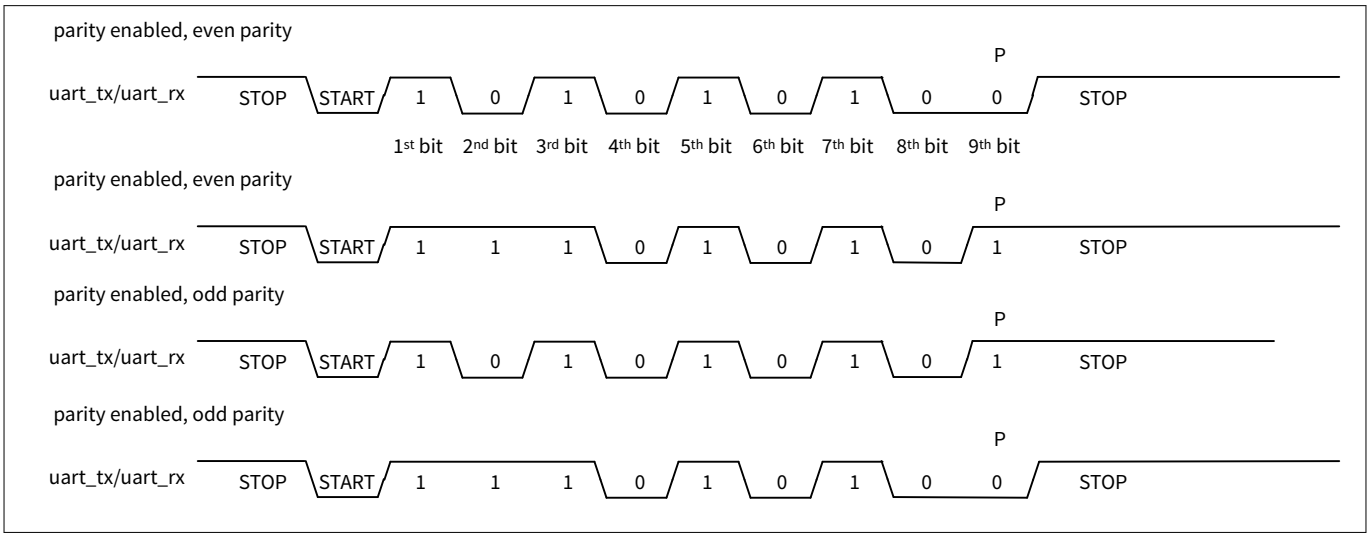


図 250　　UART パリティの例

26.4.3.1.2　スタートスキップ

スタートスキップはレシーブ機能のみに適用されます。標準 UART モードは「スタートスキップ」に対応しています。通常の受信動作は START ビット期間 (UART RX ラインの 1 から 0 への遷移) に同期し、スタートスキップ受信動作は最初の受信データフレームビット (UART RX の 0 から 1 への遷移) に同期します。

スタート スキップは、システム ディープスリープ UART RX 機能からのウェイクアップを可能にするために使用されます。そのプロセスは以下のとおりです。

1. システムディープスリープ電力モードに入る前に、UART RX 機能は無効化され、UART_RX が'1'から'0'に遷移したときに割込み原因を'1'に設定するように IOSS/GPIO がプログラムされます (START ビット)。
2. システム ディープスリープ中は、UART RX 機能は機能しません。
3. IOSS/GPIO 割込みは START ビットで起動され、システムはシステム ディープスリープから CPU アクティブ電力モードに遷移します。この遷移にはおよそ 20 µs かかります
4. CPU は、SCBx_UART_RX_CTRL.SKIP_START を'1'に設定して、UART RX 機能を有効にします。
5. UART 受信側は、次の'0'から'1'への遷移でデータフレーム受信を同期化します。UART RX 機能が時間内に有効化された場合、START ビットから 1 番目の受信データフレームビットへの遷移となります。
6. UART 受信側は通常動作で進みます。つまり、連続するデータ フレームの同期は START ビット期間で行われます。

図 251 に、そのプロセスを示します。

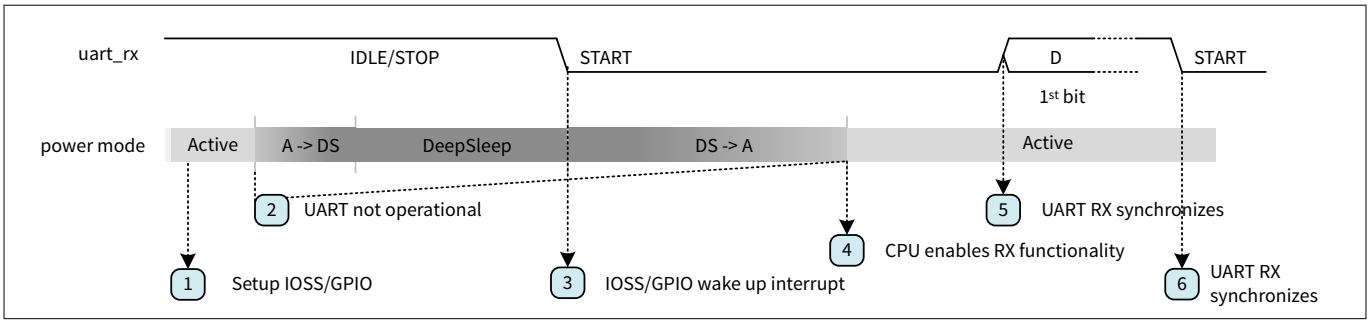


図 251　　UART スタートスキップおよびシステム ディープスリープからのウェイクアップ

上記のプロセスは、低いボーレートでのみ機能することに注意してください。システムのディープスリープから CPU のアクティブ電力モードへの移行と CPU の UART 受信機能の有効化は、UART 受信側が'0'から'1'への移行を検出する時間内にアクティブになるように、1 ビット期間未満で行う必要があります。

上記のプロセスのステップ 4 で、ファームウェアはウェイクアップ割込みルーチンを終了し、ブロックが UART RX ラインの入力立ち上りエッジを検出できるようになる前に、UART 受信機能を有効にするために時間がかかります。

上記の手順を 1 ビット未満の時間で完了できない場合は、まず"ダミー"バイトをデバイスに送信し、実際の UART データを送信する前にデバイスをウェイクアップさせます。この場合、SKIP_START ビットは 0 のままにしておけます。

この機能は、GPIO 割込みが START ビットの立ち下りエッジでトリガーされる場合にのみ動作します (他のデータビットでは動作しません)。そうでない場合、UART は破損したデータ バイトを受信します。データ破損を検出し、安全にスリープするタイミングを UART に伝えるには、より高いレベルのプロトコルに依存します。

26.4.3.1.3　半二重モード

UART 半二重接続は標準 UART サブモード (CTRL.MODE が UART、UART_CTRL.MODE が UART_STD) のみでサポートされます。プロトコルの観点からは、単一伝送線路に起因する並列送受信を除き、標準 UART サブモードのすべての機能がサポートされています。半二重パッド接続を有効にするためには、UART_RX_CTRL.HDRXEN を'1'に設定する必要があります。UART_RX_CTRL.HDRXEN='0'で全二重通信を行います。

単一伝送線路は常時駆動されないので、VDDIO への弱いプルアップ抵抗をチップ内部または外部に接続する必要があります。これにより、アイドル時間中のロジック レベルは High に保たれます。伝送パッドは、上位プロトコルと伝送速度に応じて、プッシュプルまたはオープンドレイン モードに設定できます。送信時のみ、パッドの出力ドライバが有効になります。

パッドの信号入力を恒久的に有効にする必要があります。

LIN プロトコルが有効になっている間は、受信パスも送信中に有効になります。

下図に半二重モードを示します。

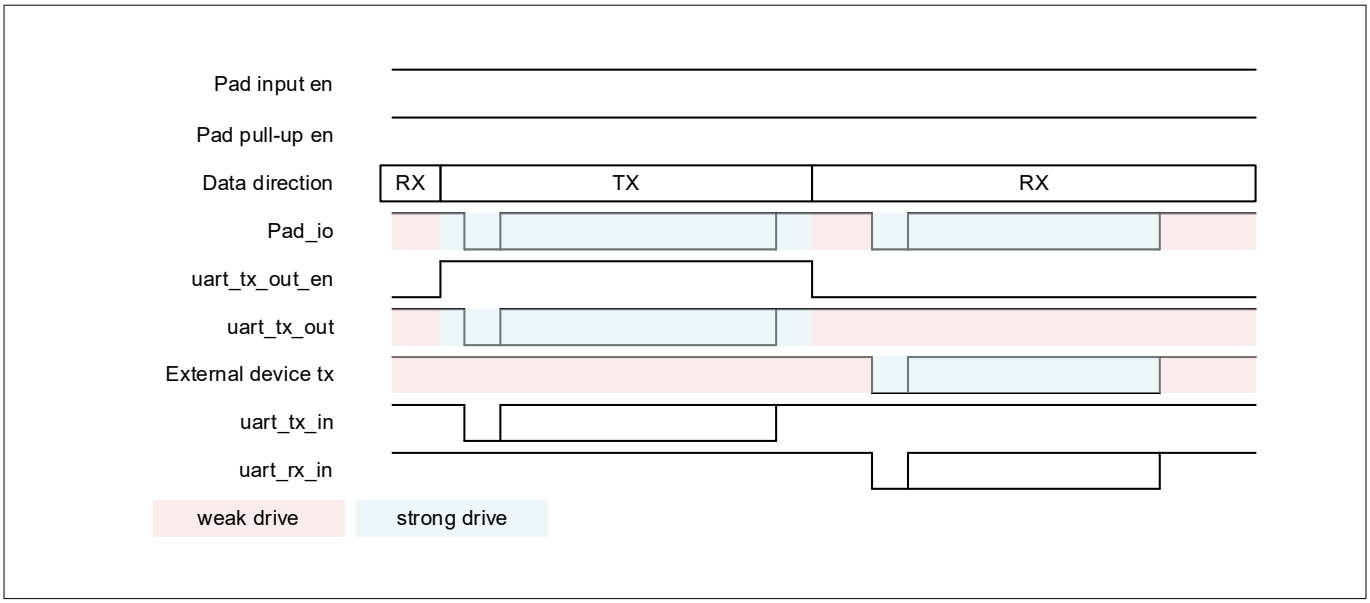


図 252　　半二重モードの UART 標準プロトコル

26.4.3.1.4　ブレーク検出

UART モード (SCBx_CTRL.MODE が UART) と標準 UART サブモード (SCBx_UART_CTRL.MODE が UART_STD) はブレーク検出をサポートします。ブレーク検出は、SCBx_UART_CTRL.BREAK_WIDTH ビット期間以上、UART_RX がロー'0'の時に検出されます。ブレーク幅は、通常のデータ転送におけるロー/'0'ビット期間の最大数より大きく、さらに 1 ビット期間を追加する必要があります。追加される 1 ビットの期間は最小要件で、できればもっと大きい方が適しています。送信側と受信側の間のクロックの不正確さのために、追加のビット期間が存在します。

例えば、8 ビットのデータフレームサイズでパリティをサポートする場合、ロー/'0'ビット期間の最大数は 10 (START ビット、8 '0'データ フレーム ビット、および 1 '0'パリティ ビット) です。したがって、ブレーク幅は 10 + 1 = 11 より大きくする必要があります (SCBx_UART_CTRL.BREAK_WIDTH は 11 に設定できます)。

- ブレーク検出は RX 機能のみに適用されます。UART 送信側は一時的に SCBx_TX_CTRL.DATA_WIDTH を増加させ、オール"ゼロデータ"フレームを送信することによってブレークを生成できます。ブレークは、送信側が受信側に特別な状態を知らせるために使用します。この状態により、受信側でリセット、シャットダウン、または初期化シーケンスが発生することがあります。
- ブレーク検出は LIN プロトコルの一部
- ブレークが検出されると、SCBx_INTR_RX.BREAK_DETECT 割込み原因を'1'に設定します。

図 253 に、通常のデータ フレームとブレーク フレーム (8 ビットのデータ フレーム、パリティ サポート、12 ビット周期のブレーク幅) を示します。

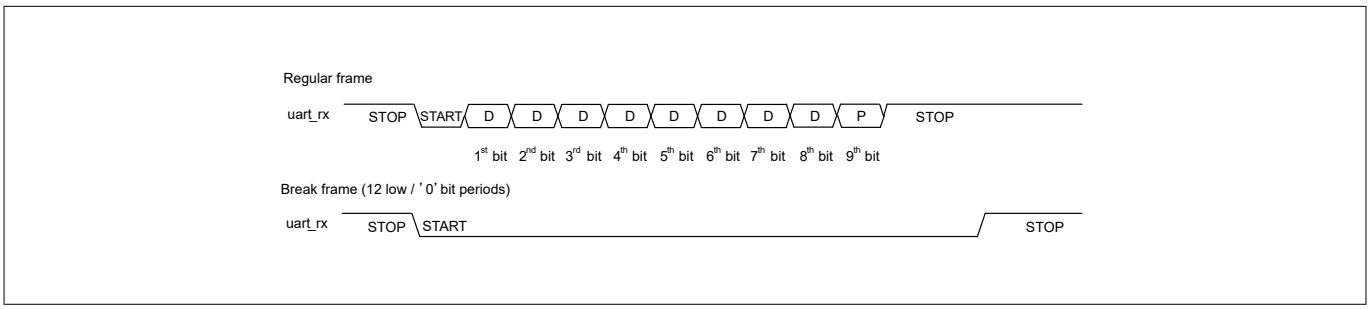


図 253　　UART - レギュラー フレームとブレーク フレーム

SCBx_UART_RX_CTRL.BREAK_LEVEL が 1 に設定されている場合、アイドルライン検出を行えます。例えば、いくつかの UART データフレームを連続して転送した後、通常のデータフレーム長 (スタート+8 データ+1 パリティ+1 ストップ) よりも長いアイドル (ハイ) レベルは、この連続転送の終了を意味します。

26.4.3.1.5 UART フロー制御

標準 UART モードはフロー制御をサポートします。モデムのフロー制御は、送信側が受信側にデータを転送するペースを制御します。フロー制御は TX 機能のみに適用されます。モデムのフロー制御は SCBx_UART_FLOW_CTRL.CTS_ENABLED レジスタフィールドで有効にできます。このフィールドが'0'の時は、送信側は TX FIFO が空でないときにデータを転送します。'1'の時は、UART CTS ラインがアクティブで、TX FIFO が空でない時に、送信側がデータを転送します。

- UART RTS (uart_rts_out): これは受信側からの出力信号です。アクティブの場合、受信側がデータ受信の準備ができていることを示します (RTS: Request to Send、送信要求)
- UART CTS (uart_cts_in): これは送信側への入力信号です。アクティブの場合、送信側がデータを転送できることを示します (CTS: Clear to Send、送信クリア)

受信側の uart_rts_out 信号は、送信側の uart_cts_in 信号に接続されています。受信側の uart_rts_out 信号は、使用された受信 FIFO エントリーの数を SCBx_UART_FLOW_CTRL.TRIGGER_LEVEL フィールドと比較することで得られます。使用済み受信 FIFO エントリー数が SCBx_UART_FLOW_CTRL.TRIGGER_LEVEL より小さい場合、uart_rts_out がアクティブになります。

通常、UART のサイドバンド信号はアクティブ LOW です。ただし、アクティブ HIGH 信号が使われることもあります。

したがって、サイドバンド信号の極性は、SCBx_UART_FLOW_CTRL.RTS_POLARITY および SCBx_UART_FLOW_CTRL.CTS_POLARITY ビットフィールドを使用して制御できます。図 254 に、フロー制御機能の概要を示します。

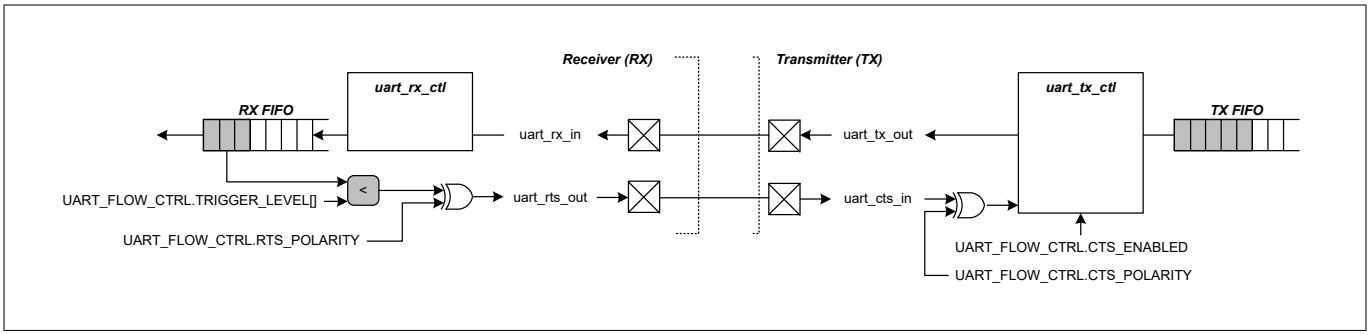


図 254 UART フロー制御接続

26.4.3.1.6 マルチプロセッサモード

図 255 に示すように、UART_MP (マルチ プロセッサ) モードは「シングル マスター マルチ スレーブ」トポロジーで定義されます。データ フィールドが 9 ビット幅であるため、このモードは UART 9 ビットプロトコルとも呼ばれています。UART_MP は標準 UART モードの一部です。

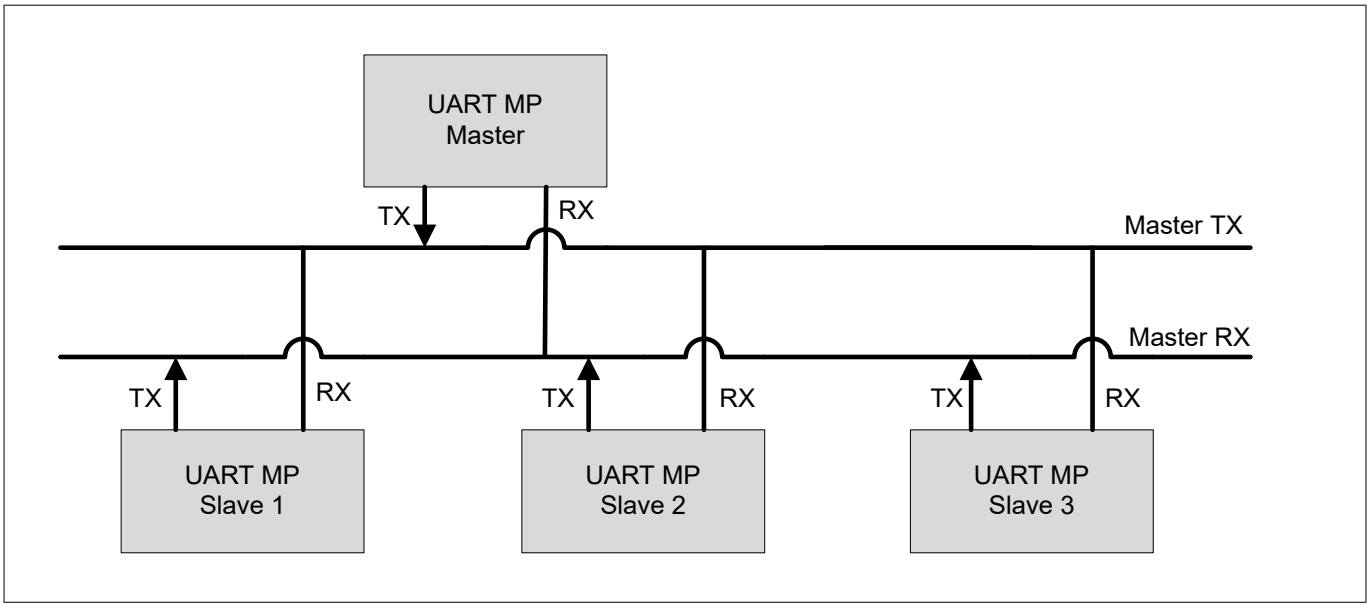


図 255　　UART MP モードのバス接続

UART_MP モードの主なプロパティは次のとおりです。

- シングル マスター マルチ スレーブ コンセプト (マルチドロップ ネットワーク)
- 各スレーブは固有のアドレスで識別
- 9 ビット データ フィールドを使用し、その 9 番目のビットをアドレス/データ フラグ ビット (MP ビット) とします。このビットは HIGH にセットされた時にアドレス バイトを示し LOW にセットされた時にデータ バイトを示します。図 256 にデータ フレームを示します。
- パリティ ビットは無効です

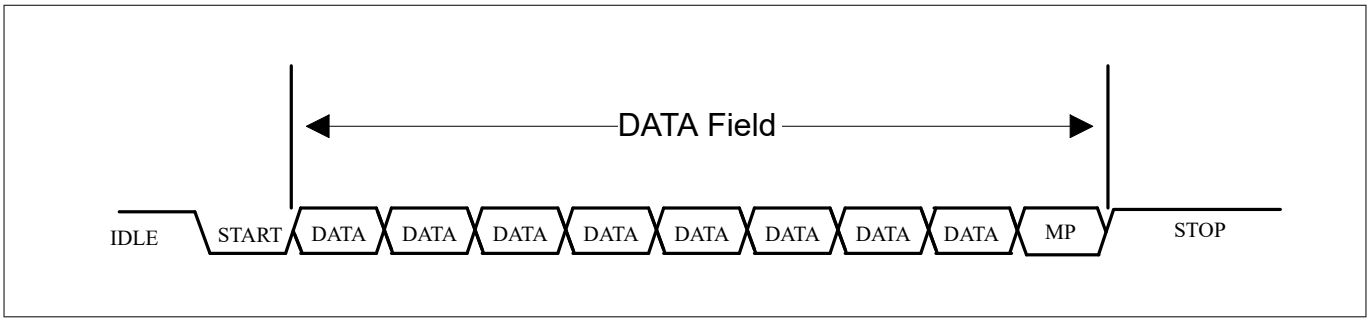


図 256　　UART MP アドレスおよびデータフレーム

SCB は UART_MP モードでマスターまたはスレーブ デバイスとして使用可能です。SCBx_TX_CTRL と SCBx_RX_CTRL レジスタは両方とも 9 ビットのデータ フレーム サイズに設定できます。SCB が UART_MP マスター デバイスとして動作する時にファームウェアはすべてのアドレスまたはデータ フレームごとに MP フラグを変更します。このブロックが UART_MP スレーブ デバイスとして機能する時に、SCBx_UART_RX_CTRL レジスタの MP_MODE フィールドを‘1’にセットする必要があります。SCBx_RX_MATCH レジスタはスレーブ アドレスとアドレス マスク用にセットする必要があります。SCBx_CTRL レジスタの ADDR_ACCEPT フィールドが‘1’にセットされた時に一致したアドレスが RX_FIFO に書き込まれます。受信したアドレスがその自身のアドレスに一致しない場合、インターフェースは比較のために次にアドレスが受信されるまで後続のデータを無視します。

標準の UART インターフェースとして SCB を設定

標準 UART インターフェースとして SCB を設定するためには様々なレジスタ ビットを次の順で設定してください。

1. SCBx_CTRL レジスタの MODE フィールド (ビット[25:24]) に「10b」を書き込むことで SCB を UART インターフェースとして設定します。

2. SCBx_UART_CTRL レジスタの MODE フィールド (ビット[25:24]) に「00」を書き込むことで標準プロトコルとして動作するように UART インターフェースを設定します。
3. UART MP モードまたは UART LIN モードを有効にするためには、それぞれ SCBx_UART_RX_CTRL レジスタの MP_MODE (ビット 10) または LIN_MODE (ビット 12) に'1'を書き込んでください。
4. UART の有効化と初期化に示したステップ 2～ステップ 4 に従ってください。

26.4.3.2 スマートカード (ISO 7816)

ISO7816 はシングル マスター シングル スレーブ トポロジで定義される非同期シリアルインターフェースです。ISO7816 はリーダー (マスター) とカード (スレーブ) 機能の両方を定義します。詳細については ISO7816 仕様を参照してください。

SCB はマスター (リーダー) とスレーブ (カード) の両方の機能に対応しています。このブロックは非同期文字送信を使用して基本物理層のサポートを提供します。UART_TX ラインは、UART_TX と UART_RX コントロール モジュール間で内部多重化されることにより、スマートカードの I/O ラインに接続されます。

SmartCard 転送は UART 転送と似ていますが、受信側から送信側への否定応答 (NACK) 信号の送信が追加されます。NACK は常に'0'です。マスターとスレーブは同じラインを駆動できる場合がありますが、その駆動を同時に行うことはできません。

SmartCard 転送では送信側がスタート ビット、データ ビット (と任意でパリティ ビット) を駆動します。これらのビットの後、バスを解放することによってストップ期間に入り、その結果ラインは'1' (ストップ ビットの値) になります。ハーフビット転送期間からストップ期間に遷移した後に受信側はライン (値が'0') 上の NACK 信号を 1～2 ビットの転送周期転送周期で駆動することがあります。この NACK が送信側によって観察され、ストップ周期を 1 ビット転送周期延長する反応をします。このプロトコルが機能するためには、ストップ期間は転送期間の少なくとも 2 ビットでなければいけません。NACK を伴うデータ転送は、NACK を伴わないデータ転送よりも 1 ビット転送期間が長いことに注意してください。実装は通常プルアップ抵抗付きのトライステートドライバを使用し、ラインはデータまたはストップ ビットの送信中でないとき、その値は'1'です。

図 257 に SmartCard プロトコルを示します。

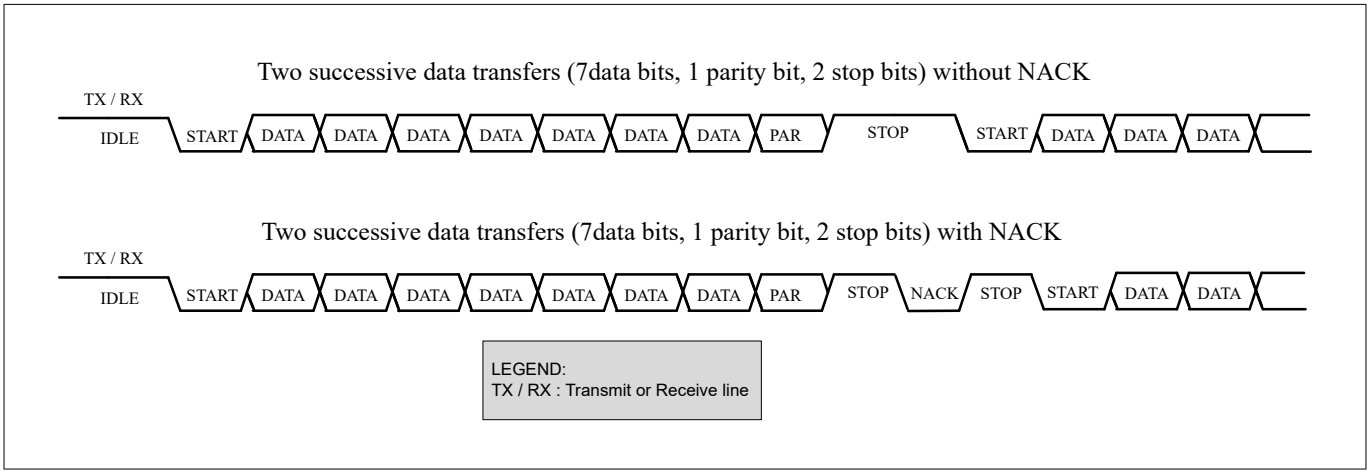


図 257 SmartCard の例

スマートカード使用時の通信ボーレートは以下のとおりです。

ボーレート = Fscb_clk/Oversample

Fscb_clk は"scb_clk"クロック周波数、Oversample はオーバーサンプル係数です。

UART SmartCard インターフェースとして SCB を設定

標準 UART インターフェースとして SCB を設定するためには様々なレジスタ ビットを次の順で設定してください。これらのレジスタの詳細については、UART レジスタを参照してください。

1. SCB_CTRL レジスタの MODE フィールド (ビット[25:24]) に‘10’を書き込んで SCB を UART インターフェースとして設定します。
2. SCB_UART_CTRL レジスタの MODE フィールド (ビット[25:24]) に‘01’を書き込んで SmartCard プロトコルとして動作するように UART インターフェースを設定します。
3. UART の有効化と初期化で述べたステップ 2～ステップ 4 を行ってください

26.4.3.3 IrDA

SCB は、UART インターフェースを使用して、最大 115.2 kbps のデータレートの IrDA プロトコルに対応しています。このブロックはデータ レートが 115.2 Kbps 未満の IrDA プロトコルの基本物理層のみをサポートします。そのため、このブロックを簡略化したシステムではその他の可能なシステム リソースを使って完全な IrDA 通信システムを搭載する方法を考慮する必要があります。

IrDA プロトコルは UART 信号に変調スキームを追加します。送信側ではビット列が変調され、受信側ではビット列が復調されます。変調スキームは Return-to-Zero-Inverted (ゼロ復帰逆転 - RZI) フォーマットを使用します。‘0’のビット値がライン上の短い‘1’パルスで信号が送られ、‘1’のビット値がラインを‘0’に維持することで信号が送られます。これらのデータ レート (115.2 Kbps 以下) には RZI 変調スキームが使用されパルスの期間はビット周期の 16 分の 3 です。サンプリング クロック周波数は、SCB_CTRL レジスタの SCB_OVS フィールドを設定することで選択したボーレートの 16 倍に設定する必要があります。さらに、PSOC™ Control C3 MCU は低消費電力の IrDA 受信モードをサポートしており、最小幅 1.41 µs のパルスを検出できます。

115.2 kbps 未満のビットレートは、clk_scb の周波数を設定することで達成できます。追加可能な速度は 2.4 Kbps、9.6 Kbps、19.2 Kbps、38.4 Kbps と 57.6 Kbps です。図 258 に、UART 転送がどのように IrDA 変調されるかを示します。

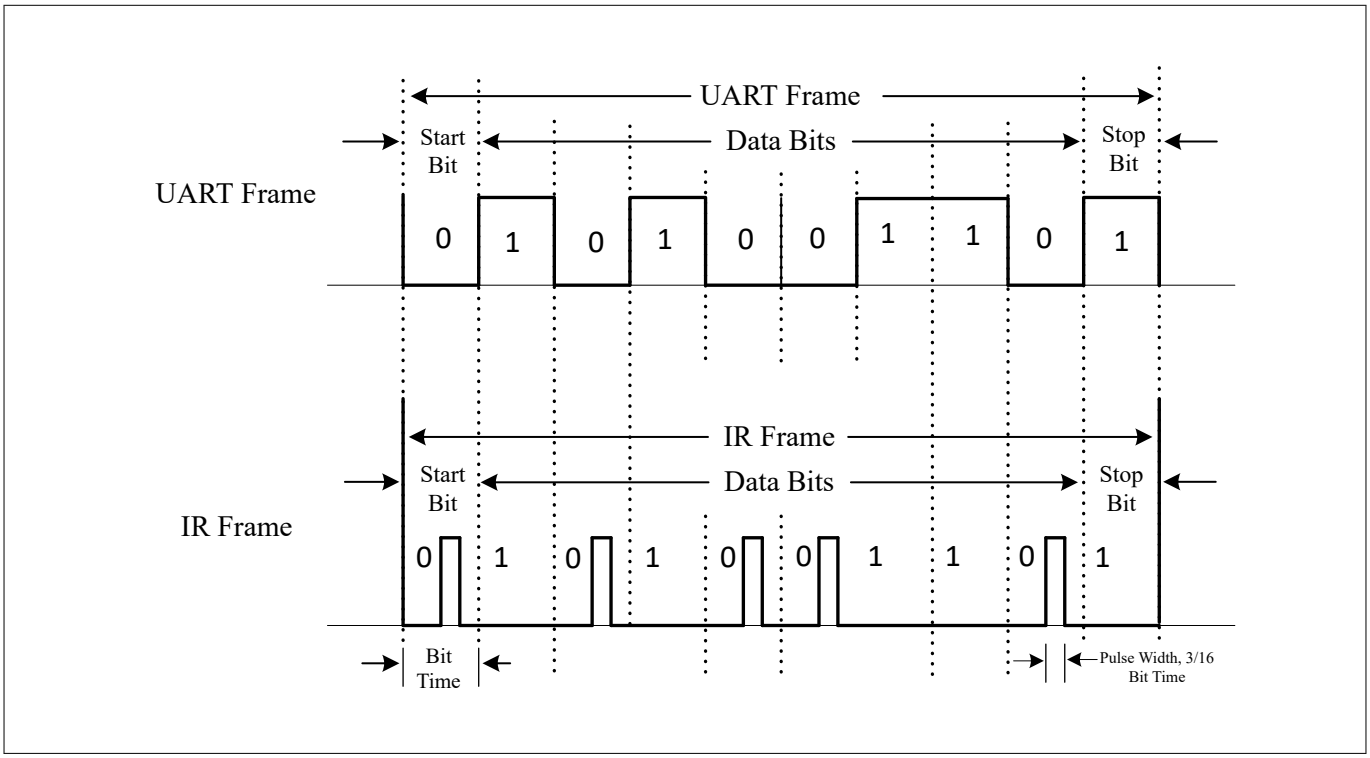


図 258 IrDA の例

UART IrDA インターフェースとして SCB を設定

SCB を UART IrDA インターフェースとして設定するためには、以下の順序で各種レジスタビットを設定してください。これらのレジスタの詳細については、UART レジスタを参照してください。

1. SCB_CTRL レジスタの MODE フィールド (ビット[25:24]) に‘10b’を書き込むことで SCB を UART インターフェースとして設定します。

2. SCB_UART_CTRL レジスタの MODE (ビット[25:24]) に‘10b’を書き込むことで、UART インターフェースを IrDA プロトコルとして動作するように設定します。
3. SCB_RX_CTRL レジスタの MEDIAN フィールド (ビット 9) に‘1’を書き込むことで入力インターフェース ライン上のメディアン フィルターを有効にします。
4. 「UART の有効化と初期化」の説明に従って SCB を設定します。

26.4.3.4 UART ローカル インターコネクト ネットワーク (LIN) モード

LIN プロトコルは SCB で標準 UART の一部としてサポートされています。LIN はシングルマスター-マルチスレーブのトポロジーで設計されています。LIN バスには 1 つのマスターノードと複数のスレーブノードがあります。SCB UART は LIN スレーブ機能のみをサポートします。LIN 仕様では、物理層 (レイヤー 1) とデータリンク層 (レイヤー 2) の両方を定義しています。図 259 に、UART_LIN と LIN トランシーバーを示します。

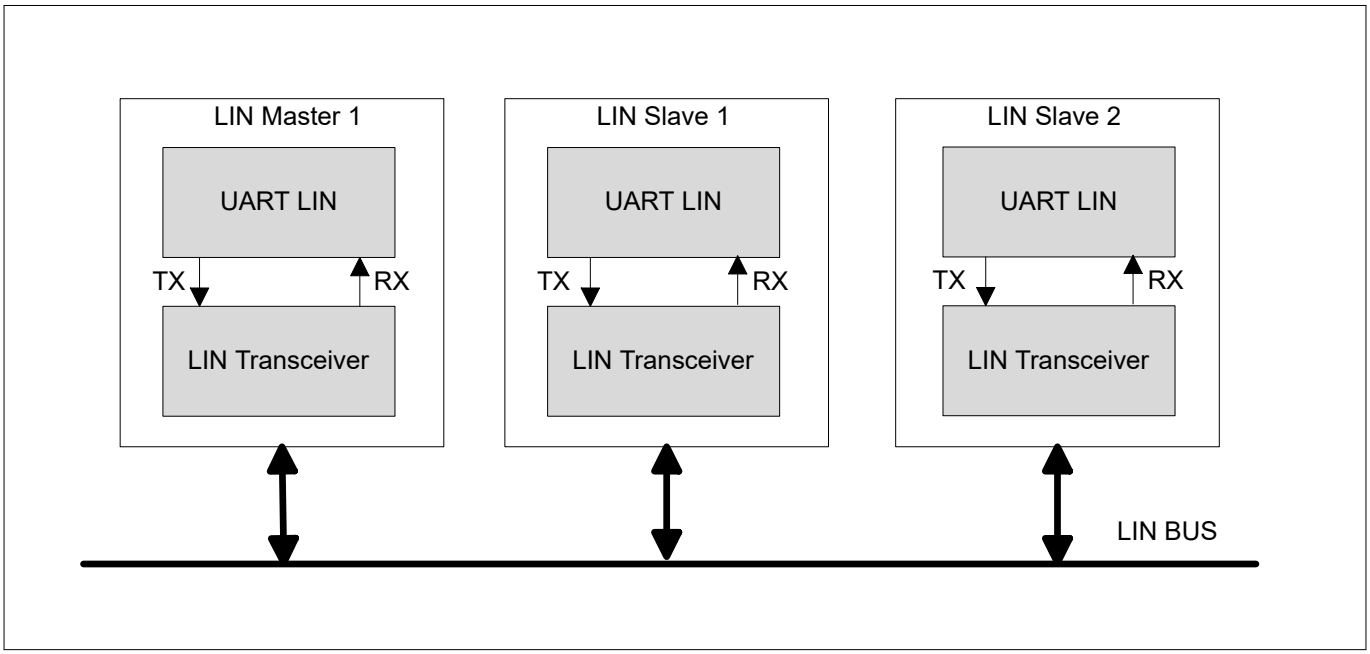


図 259 UART_LIN および LIN トランシーバー

LIN プロトコルは 2 つのタスクを定義しています。

- マスタータスク: このタスクはヘッダ パケットを送信し、LIN 転送を開始します。
- スレーブタスク: このタスクは、応答を送信または受信することを含みます。

図 260 に示すようにマスター ノードはマスター タスクとスレーブ タスクをサポートしスレーブ ノードはスレーブ タスクのみをサポートします。

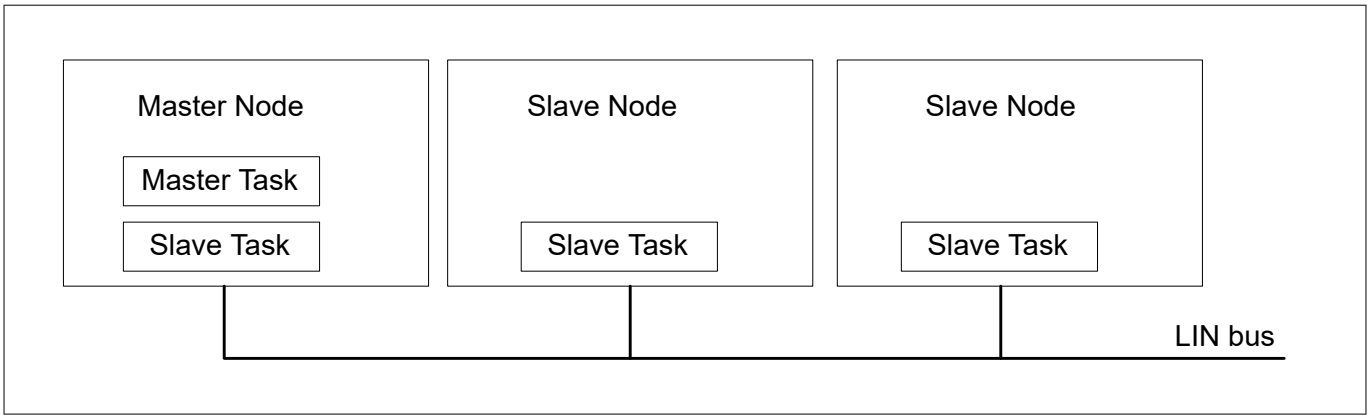


図 260　　LIN バスのノードとタスク

LIN フレームの構造

LIN は、あらかじめ決められた時間ごとにフレームを送信することを基本としています。フレームは図 261 に示すようにヘッダーと応答フィールドに分けられます。

- ヘッダー フィールドは以下のように構成されます。
  - ブレーク フィールド (少なくとも 13 ビットの期間、値は'0')
- 同期フィールド (0x55 バイトのフレーム)。同期フィールドはスレーブ タスクのクロックをマスター タスクのクロックと同期化するために使用可能
  - 識別子フィールド (特定のスレーブを指定するフレーム)
- 応答フィールドはデータとチェックサムで構成されます。

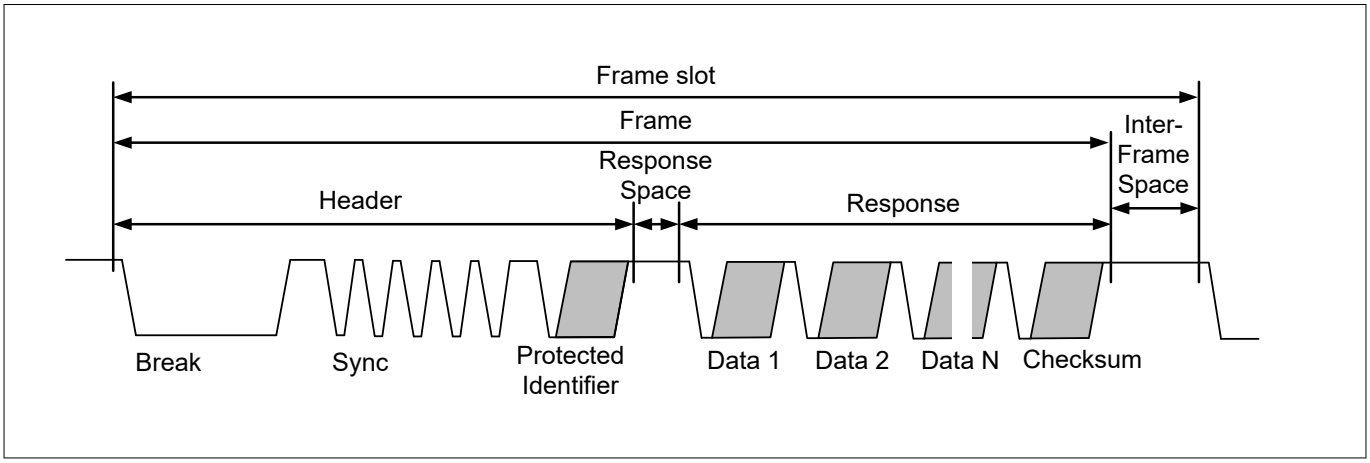


図 261　　LIN フレームの構造

LIN プロトコル通信ではデータの最下位ビット (LSb) が最初に送信され、最上位ビット (MSb) が最後に送信されます。スタート ビットが 0 としてエンコードされストップ ビットが 1 としてエンコードされます。次の節で LIN フレーム内のすべてのバイト フィールドについて説明します。

ブレーク フィールド

すべての新しいフレームはマスターによって常に生成されるブレーク フィールドから始まります。ブレーク フィールドは論理 0 の最小 13 ビット時間があり、その後にブレークデリミタが続きます。図 262 にブレーク フィールドの構造を示します。

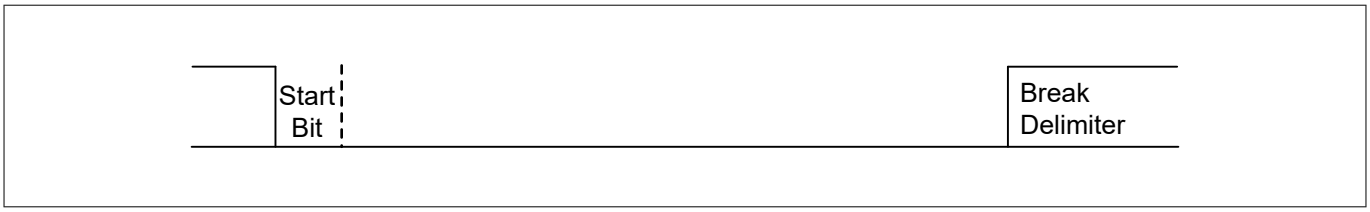


図 262　　LIN ブレークフィールド

同期フィールド

このフィールドはヘッダ フィールド内でマスターによって送信される 2 番目のフィールドです。その値は 0x55 です。スレーブタスクのクロックをマスタータスクのクロックと同期させ、ボーレートを自動検出するために、同期フィールドを使用できます。図 263 に LIN 同期フィールド構造を示します。

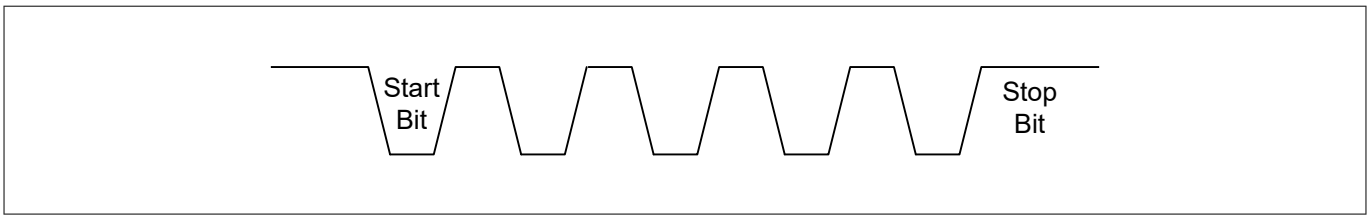


図 263　　LIN 同期フィールド

保護識別子 (PID) フィールド

保護された識別子フィールドは、フレーム識別子 (ビット 0～5) とパリティ (ビット 6 とビット 7) の 2 つのサブフィールドから構成されます。図 264 に PID フィールド構造を示します。

- フレームの識別子: フレーム識別子は以下の 3 つのカテゴリに分けられます。
  - 値 0～59 (0x3B) は信号伝送フレームに使用されます。
  - 60 (0x3C) と 61 (0x3D) は診断と設定データを伝送するために使用されます。
  - 62 (0x3E) と 63 (0x3F) は将来のプロトコル拡張のために予約されています。
- パリティ: フレーム識別子ビットがパリティを計算するために使用されます。

***注**:　　PID (保護された識別子フィールド) のパリティ計算もファームウェアによって処理され、SCB ブロックの UART は PID を一般的なデータバイトとして受け取ります。*

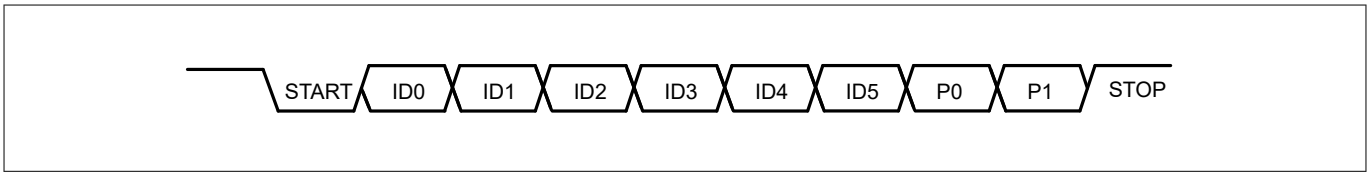


図 264　　PID フィールド

データ

LIN では各フレームは 1 バイト～8 バイトのデータを伝送できます。ここでデータ バイトの LSb が最初に送信され MSb が最後に送信されます。

チェックサム

チェックサムは LIN フレーム内の最後のバイト フィールドです。すべてのデータ バイトのみの 8 ビットのキャリー付加算またはすべてのデータ バイトと PID フィールドの 8 ビットのキャリー付加算を反転することで計算されます。LIN フレーム内のチェックサムは 2 種類あります。

- クラシック チェックサム: 全データ バイトのみについて計算 (LIN 1.x スレーブで使用)
- チェックサムの強化: すべてのデータ バイトと保護識別子で計算されるチェックサムです (LIN 2.x スレーブで使用されます)。

注: チェックサムの計算はファームウェアが行い、SCB ブロックの UART はそれを一般的なデータバイトとして受け取ります。

26.4.4 クロッキングとオーバーサンプリング

UART プロトコルは、ボーレートのオーバーサンプリング倍数として clk_scb を使用して実装されています。例えば、100 kHz の UART を実装するためには、clk_scb を 1 MHz に設定し、オーバーサンプル係数を'10'に設定します。オーバーサンプリングは SCB_CTRL.OVS レジスタ フィールドで設定します。オーバーサンプリング値は SCB_CTRL.OVS + 1 です。UART 標準サブモード (LIN を含む) と SmartCard サブモードでは、OVS フィールドの有効範囲は[7,15]です。

UART 送信 IrDA サブモードでは、このフィールドは間接的にオーバーサンプリングを指定します。オーバーサンプリングは、ビット サイクルあたりのインターフェース クロックとパルスの幅を決定します。このサブモードには有効な OVS 値-16 (SCB_CTRL レジスタの OVS フィールドの値が'0') が 1 つだけあり、パルス幅はビット周期のおよそ 3/16 です (すべてのビットレートにおいて)。

UART 受信 IrDA サブモード (1.2、2.4、9.6、19.2、38.4、57.6、115.2 kbps) では、このフィールドは間接的にオーバーサンプリングを指定します。通常の送信モードでは、このパルスはビット周期の約 3/16 です (すべてのビットレートにおいて)。低消費電力伝送モードでは、このパルスはビット周期の 3/16 より小さくなる可能性があります (115.2 kbps 未満のビットレートの場合、標準 1.62 µs、最小 1.41 µs)。SCB 入力クロック 2 サイクル以上のパルス幅は、受信側によって検出されることが保証されます。

パルス幅が 2 クロックサイクル未満で、SCB 入力の 1 クロックサイクル以上であれば、受信側で検出できます。SCB 入力クロックの 1 サイクル未満のパルス幅は、受信側によって検出されません。IrDA 受信機能を使用する場合は、SCB_RX_CTRL.MEDIAN を'1'に設定する必要があります。

SCB 入力クロックとオーバーサンプリングを合わせて IrDA ビットレートを決定します。ボーレート別の OVS 値の詳細については、レジスタリストを参照してください。

26.4.5 UART の有効化と初期化

UART を設定するためには、次の手順を実行します。

1. UART_TX_CTRL、UART_RX_CTRL、UART_FLOW_CTRL レジスタを使用して、プロトコル固有の情報をプログラムします。これはプロトコルのサブモードの選択、送信-受信機能の選択等を含んでいます
2. SCB_TX_CTRL および SCB_RX_CTRL を使用して、汎用送信側および受信側情報をプログラムします。
   a. データ フレームの幅を指定してください
   b. 最初に送信/受信されるビットが MSb であるか LSb であるかを指定してください
3. SCB_TX_FIFO_CTRL および SCB_RX_FIFO_CTRL レジスタを使用して、それぞれ送信側 FIFO および受信側 FIFO をプログラムします。
   a. トリガー レベルを設定してください
   b. 送信側と受信側 FIFO、シフト レジスタをクリアします
4. ブロックを有効にします (SCB_CTRL レジスタの ENABLE ビットに'1'を書き込んでください)。ブロックが有効になった後は、制御ビットを変更しないでください。変更はブロックを無効にしてから行います。例えば、動作モードを (SmartCard から IrDA へ) を変更する場合などです。変更はブロックを再び有効にしなければ有効になりません。

ブロックを再有効化すると再初期化され、関連する状態 (例えば、FIFO の内容) が失われることに注意してください。

26.4.6 I/O パッド接続

以下のセクションでは、UART の I/O パッド接続について説明します。

26.4.6.1 UART モード

26.4.6.1.1 標準 UART モード

図 265、図 266、図 267、表 238 に、標準 UART モードでの I/O パッドの使用方法を示します。

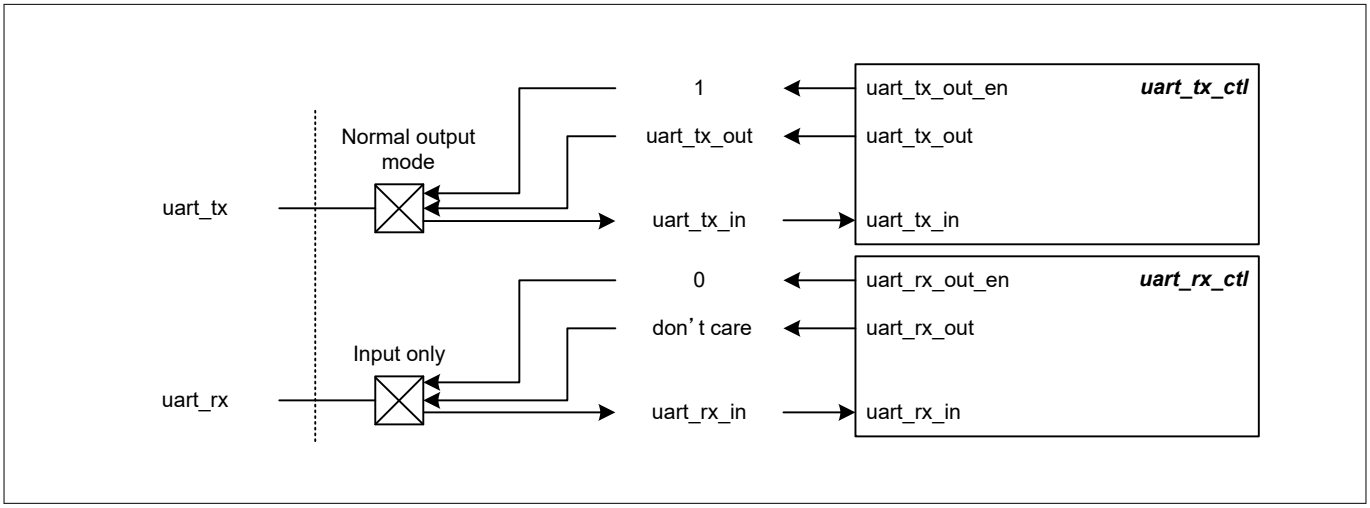


図 265 標準 UART モードの I/O パッド接続

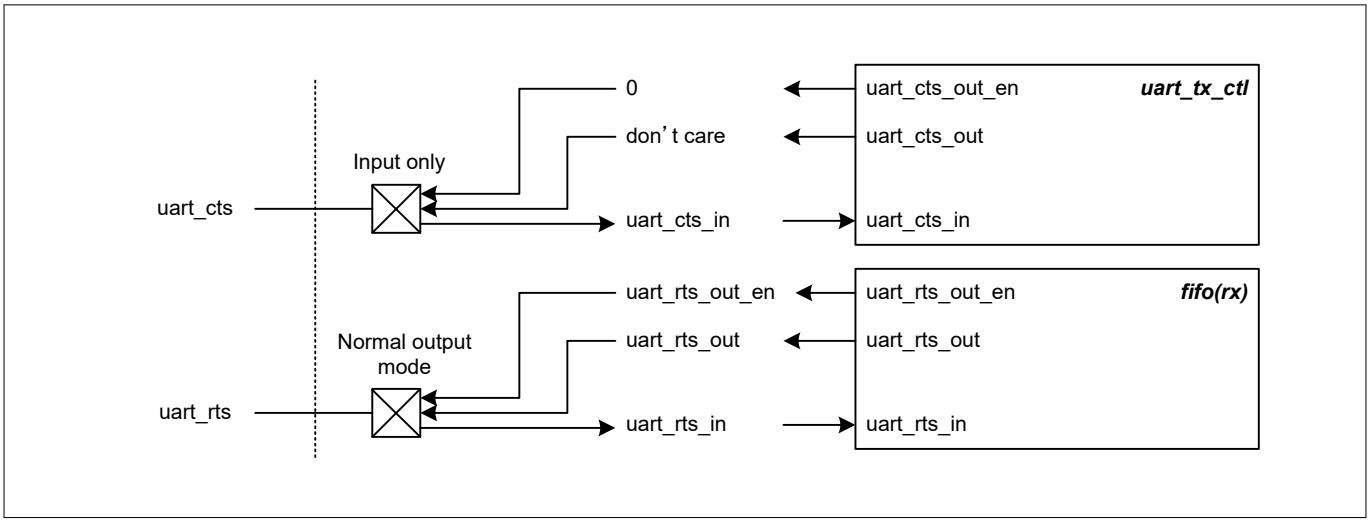


図 266 標準 UART モード、フロー制御 I/O パッド接続

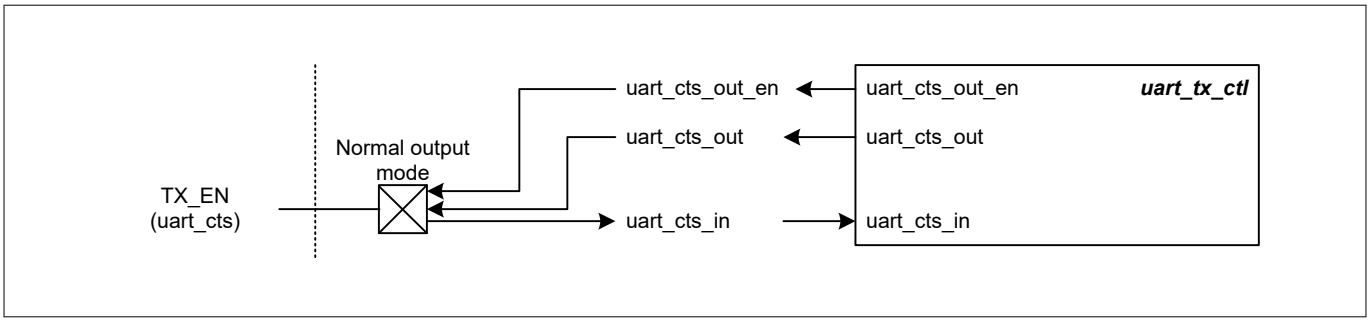


図 267　　標準 UART モード、CTS は RS485 の TX_EN として再使用

表 238　　UART I/O パッド接続の使用法

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
uart_tx	通常出力モード	uart_tx_out_en uart_tx_out	データ エレメントの送信
uart_rx	入力のみ	uart_rx_in	データ エレメントの受信
uart_cts	入力のみ	uart_cts_in	ピアパートがデータを受信する準備ができていることを示します
uart_rts	通常出力モード	uart_rts_out_en uart_rts_out	SCB がデータを受信する準備ができていることを示します
TX_EN (uart_cts)	通常出力モード	uart_cts_out_en uart_cts_out	SCB がデータを送信する準備ができていることを示します

26.4.6.1.2　半二重モード

半二重モードでは、送受信パッドは同一です。ラインを動かすアービトレーションはプロトコルレベルで行う必要があります。半二重動作には内部または外部プルアップが必要です。図 268 と表 239 に、半二重モードでの I/O パッドの使い方を示します。

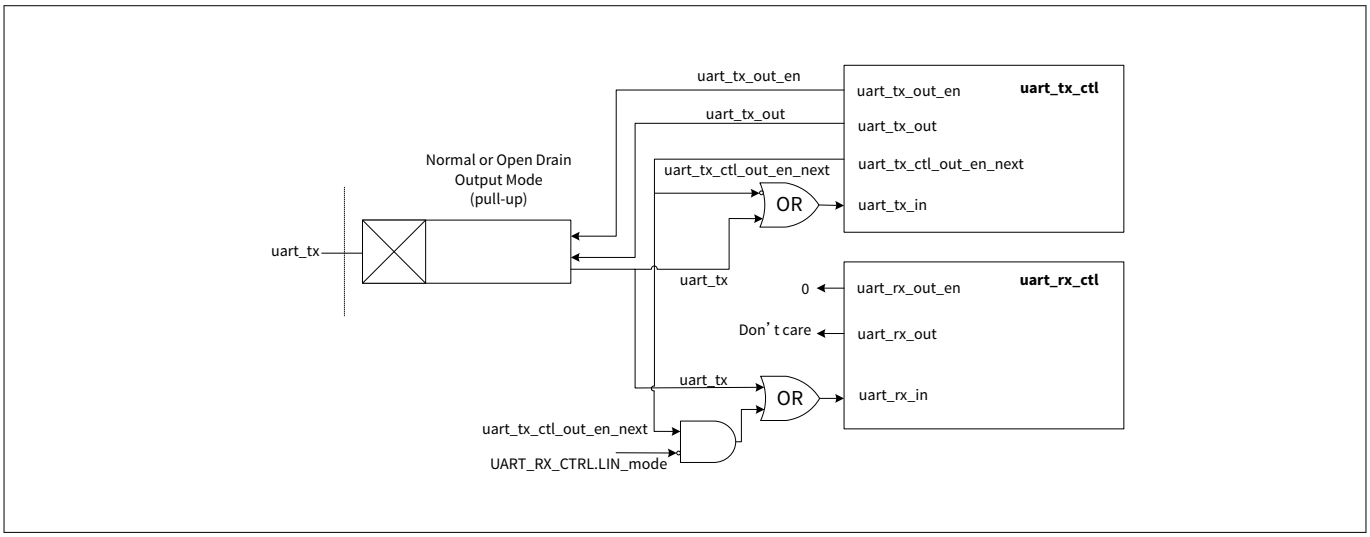


図 268　　半二重モード I/O パッド接続

表 239　　半二重モード I/O パッド接続

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
uart_tx	Push-Pull Open-drain Input Enable (入力有効化)	uart_tx_out_en uart_tx_out uart_tx_in	データ エレメントを送受信します。 uart_tx_out_en がハイの間 (送信中)、uart_rx_in 経路はブロックされ、データ受信中は uart_tx_in 経路はブロックされます。 どちらも uart_tx_out_en によって制御されます。

26.4.6.2　スマートカードモード

図 269 と表 240 に、スマートカードモードでの I/O パッドの使用方法を示します。

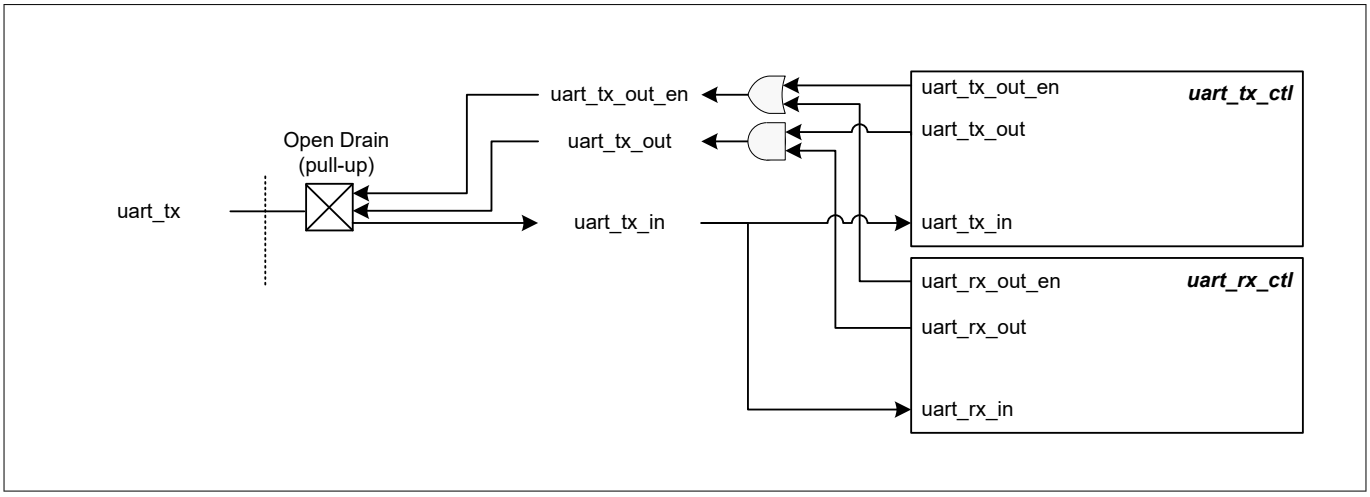


図 269　　スマートカードモードの I/O パッド接続

表 240　　スマートカードモードの I/O パッド接続

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
uart_tx	プルアップ付きオープンドレイン	uart_tx_in	データ エレメントを受け取るために使用されます。 送信されたデータ エレメントの否定応答を受け取ります。
		uart_tx_out_en uart_tx_out	データ エレメントを送信します。 受信したデータ エレメントに対して否定応答を送信します。

26.4.6.3　IrDA モード

図 270 と表 241 に、IrDA モードでの I/O パッドの使用方法を示します。

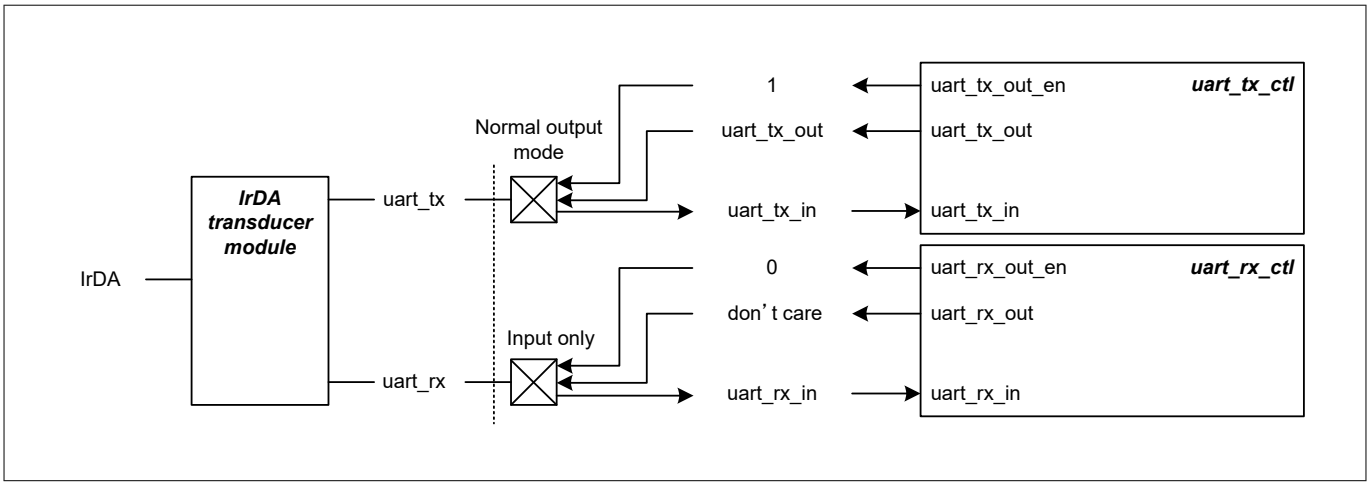


図 270　　IrDA モード I/O パッド接続

表 241　　IrDA モード I/O パッド接続

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
uart_tx	通常出力モード	uart_tx_out_en uart_tx_out	データ エレメントの送信
uart_rx	入力のみ	uart_rx_in	データ エレメントの受信

注: ***IrDA 無効の使用例シナリオ:*** *IrDA モードを使用して SCB をマスターとして設定し、TX パッドが'0'の場合、SCB が有効になり、uart_tx_out_en = 1、uart_tx_out = 0 になると、TX パッドにグリッチが観測されます。TX パッドのデフォルト値は'0'、つまり SCB が有効になる前は TX パッドは'0'です。パッドには、イネーブル信号がセットされたときにセットされる出力信号の遅延があり、それによって約 400 ps の小さなグリッチがトリガーされます。この動作は、パッドイネーブルと出力信号間の遅延によって引き起こされます。*

26.4.6.4　　LIN モード

図 271 と表 242 に、LIN モードでの I/O パッドの使い方を示します。

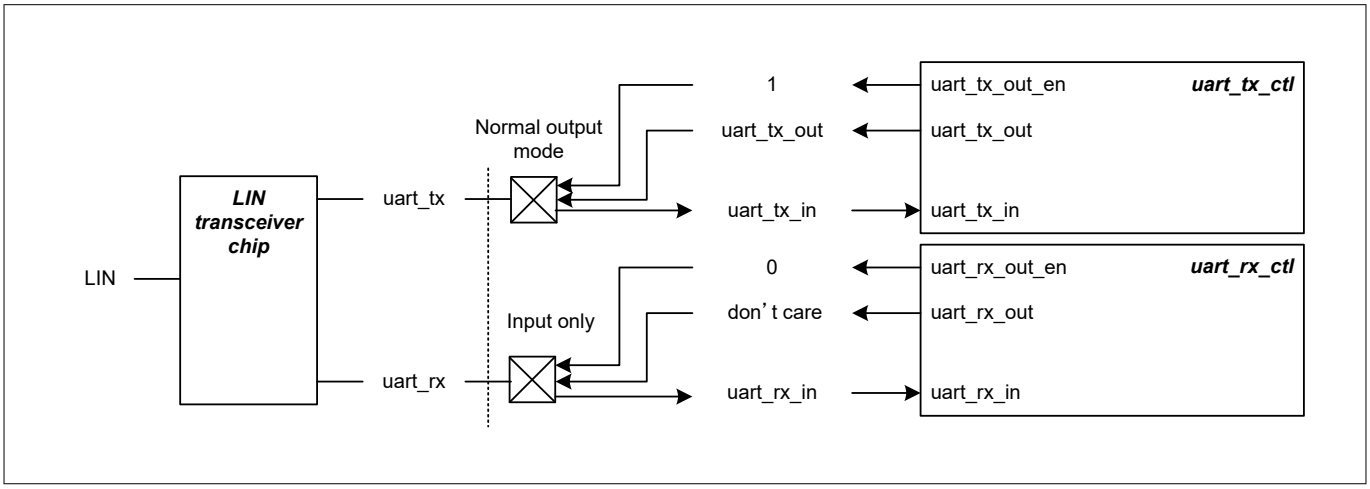


図 271　　LIN モード I/O パッド接続

表 242 LIN モード I/O パッド接続

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
uart_tx	通常出力モード	uart_tx_out_en uart_tx_out	データ エレメントの送信
uart_rx	入力のみ	uart_rx_in	データ エレメントの受信

26.4.7 レジスタの概要

UART インターフェースは 32 ビット レジスタを使用して制御されます。これらのレジスタの詳細については、PSOC™ Control C3 レジスタ リファレンス マニュアルを参照してください。

26.5 インター インテグレーテッド サーキット (I²C)

ここでは、PSOC™ Control C3 MCU の I2C 実装について説明します。I2C プロトコル仕様の詳細については、NXP のウェブサイトで入手可能な I2C バス仕様を参照してください。PSOC™ Control C3 MCU では、すべての SCB ブロックが I2C マスター モードとスレーブ モードの両方に対応しています。1 つの SCB (SCB[0]) のみがディープスリープ電力モードで利用可能で、外部クロック動作が可能です。

26.5.1 機能

このブロックは以下の機能に対応します。

- マスター、スレーブ、マスター/スレーブ機能
- 標準モード (100 kHz)、高速モード (400 kHz)、高速プラスモード (1 MHz)
- 標準 7 ビットスレーブ アドレス
- ソフトウェアによる 10 ビットアドレッシングに対応
- マルチ マスター モードとバス アービトレーション
- アナログ グリッチ フィルター
- クロック ストレッチ
- 衝突検出
- I2C クロック信号 (SCL) のプログラマブル オーバーサンプリング
- FIFO、EZ、CMD_RESP バッファモード
- ローカル ループバック制御
- 一般呼出アドレス検出
- スリープモードからのウェイクアップ

26.5.2 概要

図 272 に I2C 通信ネットワークの例を示します

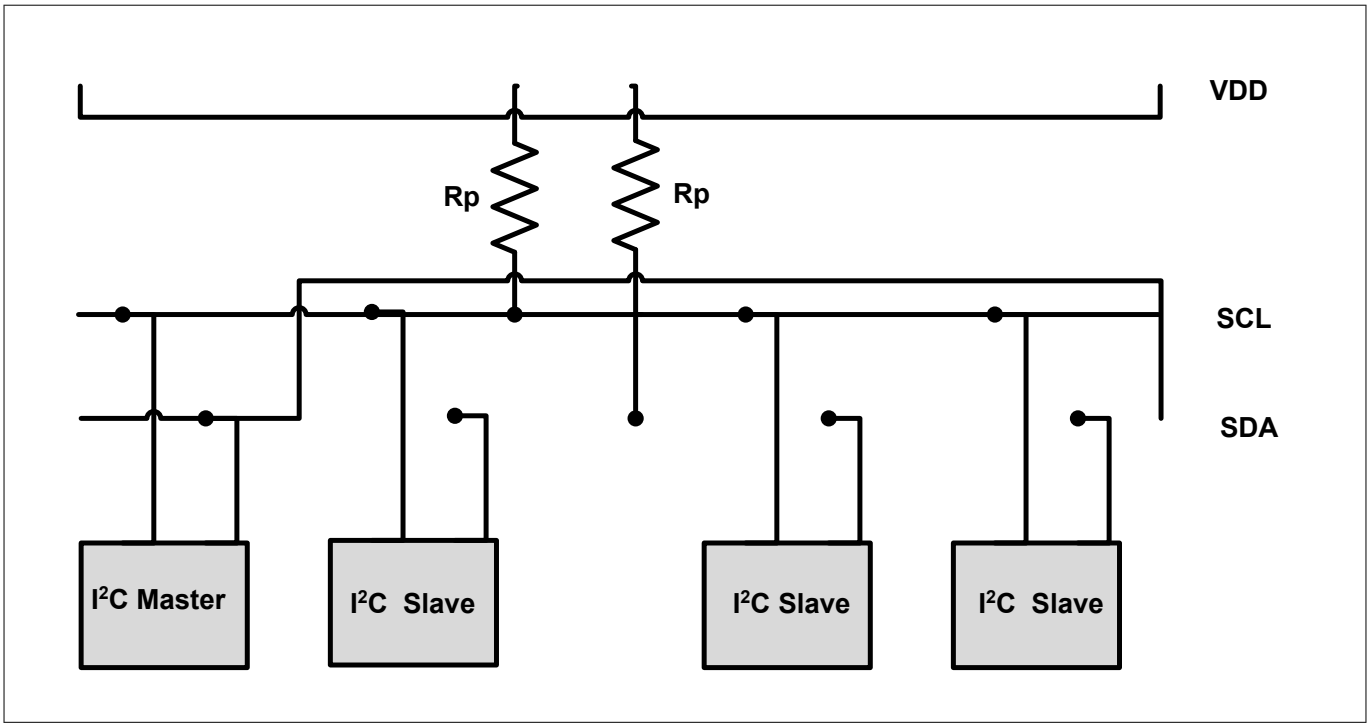


図 272　　I2C インターフェース ブロック図

標準的な I2C バスは、以下のラインを持つ 2 線式インターフェースです。

- シリアル データ (SDA)
- シリアル クロック (SCL)

I2C デバイスは、プルアップ抵抗 (Rp) 付きのオープン コレクタまたはオープンドレイン出力段を使用して、これらのラインに接続されます。デバイス間に簡単なマスター/スレーブ関係があります。マスターとスレーブは送信側か受信側のどちらでも動作できます。バスに接続する各スレーブ デバイスは、他と異なる 7 ビット アドレスをソフトウェアによりアドレス指定します。

26.5.3　　用語および定義

表に、I2C 通信ネットワークにおいて一般に使用する用語を示します。

表 243　　I2C バス用語の定義

用語	説明
マスター	転送を開始しクロック信号を生成し転送を終了するデバイス
スレーブ	マスターによってアドレス指定されるデバイス
マルチマスター	複数マスターによる同時バス制御を試みること
アービトレーション	2 つ以上のマスターが同時にバスを制御しようとする場合に、1 つのマスターだけに制御権を持たせ、メッセージが破損されないことを確保する手順
同期	2 つ以上のデバイスのクロック信号を同期化する手順

26.5.3.1 クロック ストレッチ

スレーブ デバイスはデータ処理の準備ができていない場合には SCL ラインをホールドするため‘0’に駆動することがあります。I/O 信号インターフェースの実行により、マスターまたはスレーブが SCL ラインで駆動する値に関わらず、SCL ラインは常に‘0’になります。これはクロック ストレッチといわれます。スレーブが SCL ラインを駆動するのはこの状態のみです。マスター デバイスは SCL ラインを監視し、SCL ライン上に正のクロック パルス ('1') を生成できない場合にそれを検出します。そして SCL ラインでのポジティブ エッジの生成を延期し、クロックをストレッチしているスレーブ デバイスと実質的に同期します。PSOC™ Control C3 の SCB はクロックを伸ばすことができ、実際に伸ばすでしょう。

26.5.3.2 バスアービトレーション

I2C プロトコルは、マルチ マスター、マルチ スレーブ インターフェースです。バス アービトレーションは SDA ラインを監視することで実行します。バス衝突はマスターが SDA ラインで駆動している値と一致しない値を発見した時に検出されます。例えば、マスター 1 が SDA ラインで‘1’を駆動しており、マスター 2 が SDA ラインで‘0’を駆動している時は、実際のライン値は I/O 信号インターフェースの実装により‘0’になります。マスター 1 は不整合を検出し、バスの制御を失います。マスター 2 は不整合を検出せず、バスの制御を維持します。

26.5.4 I2C 動作モード

I2C は、同期式シングルマスター、マルチマスター、マルチスレーブのシリアルインターフェースです。デバイスはマスター モード、スレーブ モード、マスター/スレーブ モードのいずれかで動作します。マスター/スレーブ モードでは、デバイスは指定されるとマスターからスレーブ モードに切り替えます。データ転送中にアクティブにできるマスターは 1 つだけの場合があります。アクティブなマスターは SCL ラインでのクロック駆動を担当します。表 244 に I2C 動作モードを示します。

表 244 I2C モード

モード	説明
スレーブ	スレーブ専用の動作 (初期設定)
マスター	マスター専用の動作
マルチマスター	バス上で複数のマスターをサポートします。

表 245 に、I2C データ転送の一部である一般的なバスイベントをいくつか示します。書き込み転送と読み出し転送のセクションでは、データ転送中の I2C バスのビットフォーマットについて説明します。

表 245 I2C バスイベント用語

バス イベント	説明
START	SCL が HIGH の間に SDA ライン上の HIGH から LOW への遷移
STOP	SCL が HIGH の間に SDA ライン上の LOW から HIGH への遷移
ACK	送信側が各バイトを送信した後、受信側は SDA ラインを LOW に引き下げ、クロックパルスの HIGH 期間中、その LOW 状態を維持します。これは、受信側がバイトを正しく受信したことを送信側に示します。

(続く)

表 245 (続き) I2C バスイベント用語

バス イベント	説明
NACK	送信側が各バイトを送信した後、受信側は SDA ラインを LOW にプルダウンせず、クロックパルスの HIGH 期間中、HIGH 状態を維持します。これは送信側に対して、受信側がそのバイトの受信に失敗したことを示します。
Repeated START	転送終了時に STOP コンディションの代わりにマスターが生成する START コンディションのこと
DATA	SCL が LOW (データ変更中) の間は SDA の状態が変化し、SCL が HIGH (データ有効) の間は変化しません。

これらのモードでは、読み出しと書き込みの 2 種の転送があります。書き込み転送では、マスターがスレーブにデータを送信します。読み出し転送では、マスターがスレーブからデータを受信します。

26.5.4.1 書き込み転送

- 典型的な書き込み転送は、マスターが I2C 上で START コンディションを生成することから始まります。次にマスターは START コンディションの後、7 ビットの I2C スレーブ アドレスと書き込みインジケーター (‘0’) を書き込みます。アドレス指定されたスレーブは 9 番目のビット時間の間にデータ ラインを LOW に引き下げることで確認応答バイトを送信します。
- スレーブ アドレスがどのスレーブ デバイスとも一致しない場合、またはアドレス指定されたデバイスが要求を確認したくない場合は、SDA ラインをプルアップしないことによって確認応答なし (NACK) を送信します。確認応答がない場合、プルアップ抵抗が実装されているため、SDA ラインの値は'1'になります。
- スレーブから確認応答が送信されない場合、マスターは STOP で書き込み転送を終了できます。
- マスターは ACK を受信した場合、バスにデータを転送できます。アドレス指定されたスレーブは書き込まれたデータの各バイトの受信を確認するために ACK を送信します。この ACK を受信すると、マスターは他のデータ バイトを送信できます。
- 転送が完了すると、マスターは STOP コンディションを生成します。

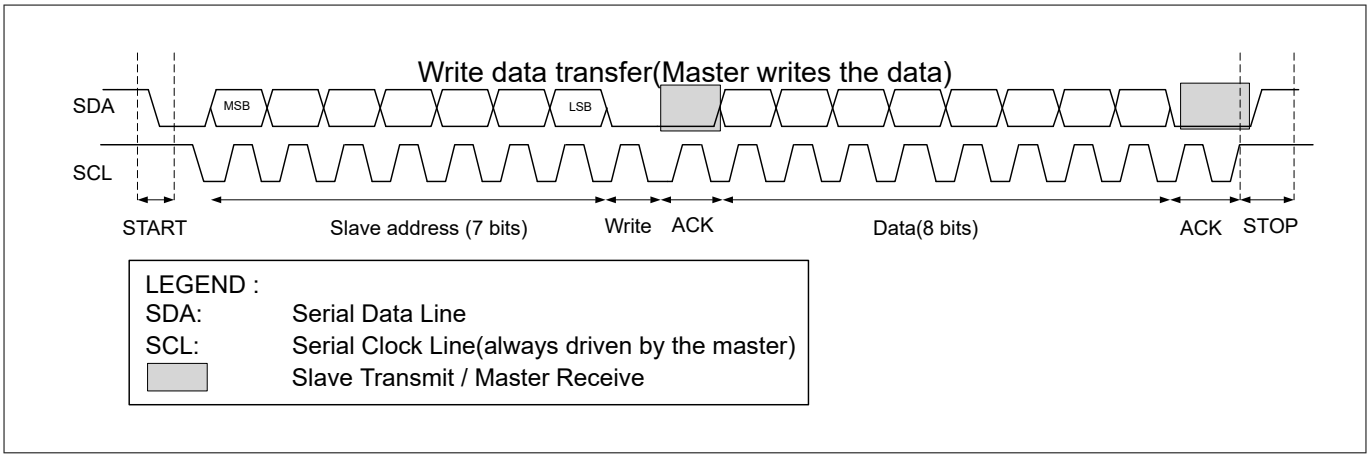


図 273 マスター書き込みデータ転送

26.5.4.2 読み出し転送

- 典型的な読み出し転送は、マスターが I2C 上で START コンディションを生成することから始まります。次にマスターは START コンディションの後、7 ビットの I2C スレーブ アドレスと読み出しインジケーター (‘1’) を書き込みます。アドレス指定されたスレーブは 9 番目のビット時間の間にデータ ラインを LOW に引き下げることで確認応答バイトを送信します。
- スレーブ アドレスが接続しているスレーブ デバイスに一致しない場合、またはアドレス指定されたデバイスが要求に確認応答しようとしない場合、送信 SDA ラインを LOW に引き下げないことにより、確認応答なし (NACK) が送信されます。確認応答がない場合、プルアップ抵搭載により SDA ラインの値は‘1’になります。
- NACK がスレーブによって送信されない場合、マスターは STOP イベントで読み出し転送を終了できます。またマスターは再転送のために、繰り返し START コンディションを生成できます。
- スレーブ アドレスを認識した場合は、ACK 信号の後にデータの転送を開始します。マスターはスレーブから送信された各データ バイトの受信を確認するために ACK 信号を送信します。この ACK を受信すると、アドレス指定されたスレーブは他のデータ バイトを送信できます。
- マスターは、スレーブのデータ バイト転送を停止するためにスレーブに NACK 信号を送信できます。これにより読み出し転送が完了します。
- 転送が完了すると、マスターは STOP コンディションを生成します。
- 個々のデータ転送 (1 つ以上のデータ エレメント) は START イベントで始まり、STOP で終わります。複合データ転送は、STOP イベントによって区切られることなく、START イベントのみが繰り返される複数の個別転送で構成されます。

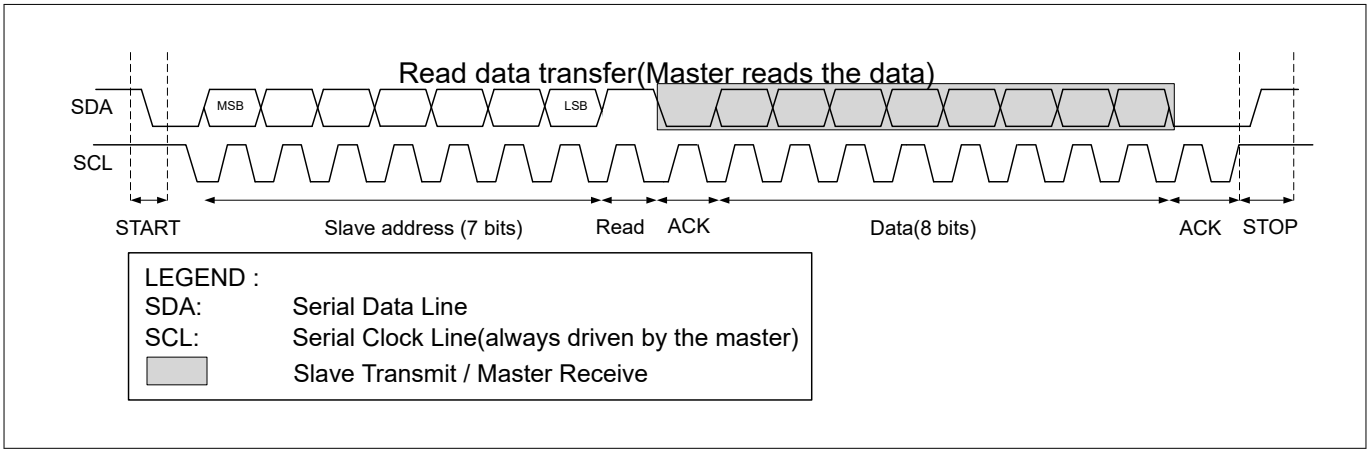


図 274 マスター読み出しデータ転送

26.5.5 I2C バッファモード

I2C は、FIFO モード、EZ モード、CMD_RESP モードの 3 つの異なるバッファ モードで動作できます。バッファの使い方は、それぞれのモードで異なります。以下のサブセクションでは、それぞれのバッファ モードについて詳しく説明します。

26.5.5.1 FIFO モード

FIFO モードには、送信データ用の TX FIFO と受信データ用の RX FIFO があります。各 FIFO は SRAM バッファから構成されます。FIFO は、16 ビット データ エレメントで 64 エレメントの深さ、または 8 ビット データ エレメントで 128 エレメントの深さです。データ エレメントの幅は SCBx_CTRL.MEM_WIDTH で設定します。I2C では、すべてのトランザクションがバイト幅であるため、FIFO を BYTE モードにすることを推奨します。

FIFO モード動作は、CPU アクティブおよび CPU スリープ電力モードでのみ有効です。システム ディープスリープ電力モードでは、システム ディープスリープ対応 SCB のみが、スレーブ アドレスの一致によってデバイスをスリ

ープからウェイクアップできます。詳細については、「FIFO バッファモードにおけるシステム ディープスリープから CPU アクティブモードへの移行」のセクションを参照してください。

送信 FIFO への書き込みアクセスには、SCBx_TX_FIFO_WR レジスタを使用します。受信 FIFO への読み出しアクセスには、SCBx_RX_FIFO_RD レジスタを使用します。

送受信 FIFO のステータス情報は、SCBx_TX_FIFO_STATUS および SCBx_RX_FIFO_STATUS ステータス レジスタを通じて入手できます。デバッグ モードの場合、このレジスタからの読み出しは SCBx_RX_FIFO_RD_SILENT レジスタからの読み出しとして動作します。

FIFO エントリーの数を示すプログラム可能な閾値を定義できます。以下の条件が満たされたときにトリガー/イベントが発生します。

- TX FIFO の場合、送信 FIFO のエントリー数が TRIGGER_LEVEL より少ないとき、トリガーが生成されます。
- RX FIFO の場合、FIFO のエントリー数が TRIGGER_LEVEL より大きいとき、トリガーが発生します。

さらに、いくつかの割込みステータス ビットも用意されており、FIFO がフルであるか、空であるかなどを示します。

FIFO バッファモードでのシステム ディープスリープから CPU アクティブモードへの移行

SCBx_CTRL.EC_AM_MODE = 1、SCBx_CTRL.EC_OP_MODE = 0、FIFO モード。以下の説明はスレーブモードにのみ適用されます。

マスター ライト

- S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = 1。クロックは、内部クロック ロジックが引き継ぐまでストレッチされ、その時点でアドレスが ACK され、マスターはデータの書き込みを開始できます。ディープスリープに入る前に、CLK_SCB を無効にする必要があります。ディープスリープからのウェイクアップ時に CLK_SCB を再有効化する必要があります。
- S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = 0。クロックは、内部クロック駆動のロジックが制御を引き継ぎ、CPU が SCBx_I2C_S_CMD.S_ACK または SCBx_I2C_S_CMD.S_NACK のいずれかを書き込むまでストレッチされます。ディープスリープに入る前に CLK_SCB を無効にする必要があります。ディープスリープからのウェイクアップ時には、CLK_SCB を必ず再有効化してください。SCBx_I2C_S_CMD.S_ACK または SCBx_I2C_S_CMD.S_NACK を設定する前にこれを行ってください。
- S_NOT_READY_ADDR_NACK = 1, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x.入力されたアドレスは、内部クロック ロジックが引き継ぐまで NACK されます。内部クロックのロジックが引き継がれる場合、PLL/FLL のロック時間により、内部クロックが正しい周波数になる保証はありません。このため、I2C バスの ACK/NACK のタイミングが正しくない可能性があります。これを避けるには、ディープスリープに入る前に CLK_SCB を無効にし、PLL/FLL が安定した後に再有効化します。

マスター リード

- S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x.受信アドレスは、内部クロック ロジックが引き継ぎ、CPU が TX FIFO にデータを書き込むまでストレッチされます。ディープスリープに入る前に CLK_SCB を無効にする必要があります。ディープスリープからのウェイクアップ時、TX FIFO にデータを書き込む前に CLK_SCB を再有効化する必要があります。
- S_NOT_READY_ADDR_NACK = 1, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x.入力されたアドレスは、内部クロック ロジックが引き継ぐまで NACK されます。このような場合、PLL/FLL のロック時間により、内部クロックが正しい周波数になる保証はありません。このため、ACK/NACK の I^2C バスのタイミングが正しくない可能性があります。これを避けるには、ディープスリープに入る前に CLK_SCB を無効にし、PLL/FLL が安定した後に再び無効にします。

注: 書き込み後に繰り返し開始する場合は、UNDERFLOW 割込みステータスがアサートされるまで待ってから、I2C_M_CMD.START ビットを設定し、新しいアドレスを TX_FIFO に書き込みます。そうでない場合、FIFO 内のアドレスはデータとして送信され、アドレスとしては送信されません。

26.5.5.2 EZI2C モード

EZI2C プロトコルは、インフィニオンの I2C プロトコル上に構築された独自の通信方式です。このプロトコルはインデックス付きメモリ転送を使用して I2C スレーブと通信するために、標準的な I2C プロトコルのメタプロトコルを使用しています。これにより、CPU が介入する必要がなくなります。

EZI2C プロトコルは、スレーブデバイス上のバッファ (1 エントリーあたり 8 ビットの 256 エントリー配列がサポートされている) をインデックスする 8 ビットアドレスを持つ単一のメモリバッファを定義しています。EZ アドレスは、この 256 ロケーションをアドレス指定するために使用されます。CPU は EZ_DATA レジスタをとおしてメモリ バッファに書き込みと読み出しを行います。これらのアクセスはワード アクセスですが、使用されるのはワードの最下位バイトのみです。

スレーブ インターフェースは、現在のアドレスを使用してメモリ バッファにアクセスします。転送開始時 (I2C START/RESTART)、ベースアドレスは現在のアドレスにコピーされます。データ エレメントの書き込みまたは読み出し操作は、現在のアドレス位置に対して行われます。アクセス後、現在のアドレスは'1'インクリメントされます。

現在アドレスが最後のメモリバッファアドレス (255) と等しい場合、現在アドレスはインクリメントされません。それ以降の書き込みアクセスは、最後のバッファ アドレスに以前書き込まれた値を上書きします。それ以降の読み出しアクセスは、最後のバッファ アドレスで (同じ) 読み出し値を提供し続けます。バス マスターは、EZ モードでのメモリ バッファ容量を認識しておく必要があります。

I2C のベースアドレスと現在のアドレスは、I2C_STATUS をとおして提供されます。転送が終了した時点で、ベースアドレスと現在のアドレスの差は、何回の読み出しまたは書き込みアクセスが実行されたかを示します。このブロックは、転送の終わりを識別するための割込み要因フィールドを提供します。EZI2C はファームウェアまたはハードウェアで実装できます。すべての SCB は、CPU アクティブおよび CPU スリープの両電力モードで、ファームウェア実装により EZI2C を実装できます。ディープスリープ SCB は、システム ディープスリープ電力モードを持つハードウェアおよびファームウェアベースの EZI2C を実装できます。

EZI2C は 3 つの動作フェーズを区別しています。

- アドレス フェーズ: マスターは 8 ビットのアドレスをスレーブに送信します。このアドレスは、スレーブのベースアドレスおよび現在のアドレスとして使用されます。
- 書き込みフェーズ: マスターは、8 ビットのデータ エレメントをスレーブのメモリ バッファに書き込んでください。スレーブの現在アドレスは、スレーブのベースアドレスに設定されます。受信したデータ エレメントは、現在のアドレス メモリ位置に書き込まれます。各メモリ書き込みの後、現在のアドレスがインクリメントされます
- 読み出しフェーズ: マスターは、スレーブのメモリ バッファから 8 ビットのデータ エレメントを読み出します。スレーブの現在アドレスは、スレーブのベースアドレスに設定されます。送信されたデータ エレメントは、現在のアドレス メモリ位置から読み出されます。メモリが読み出されるたびに、現在のアドレスがインクリメントされます。

***注**: スレーブのベースアドレスは、CPU ではなくマスターによって更新されます。*

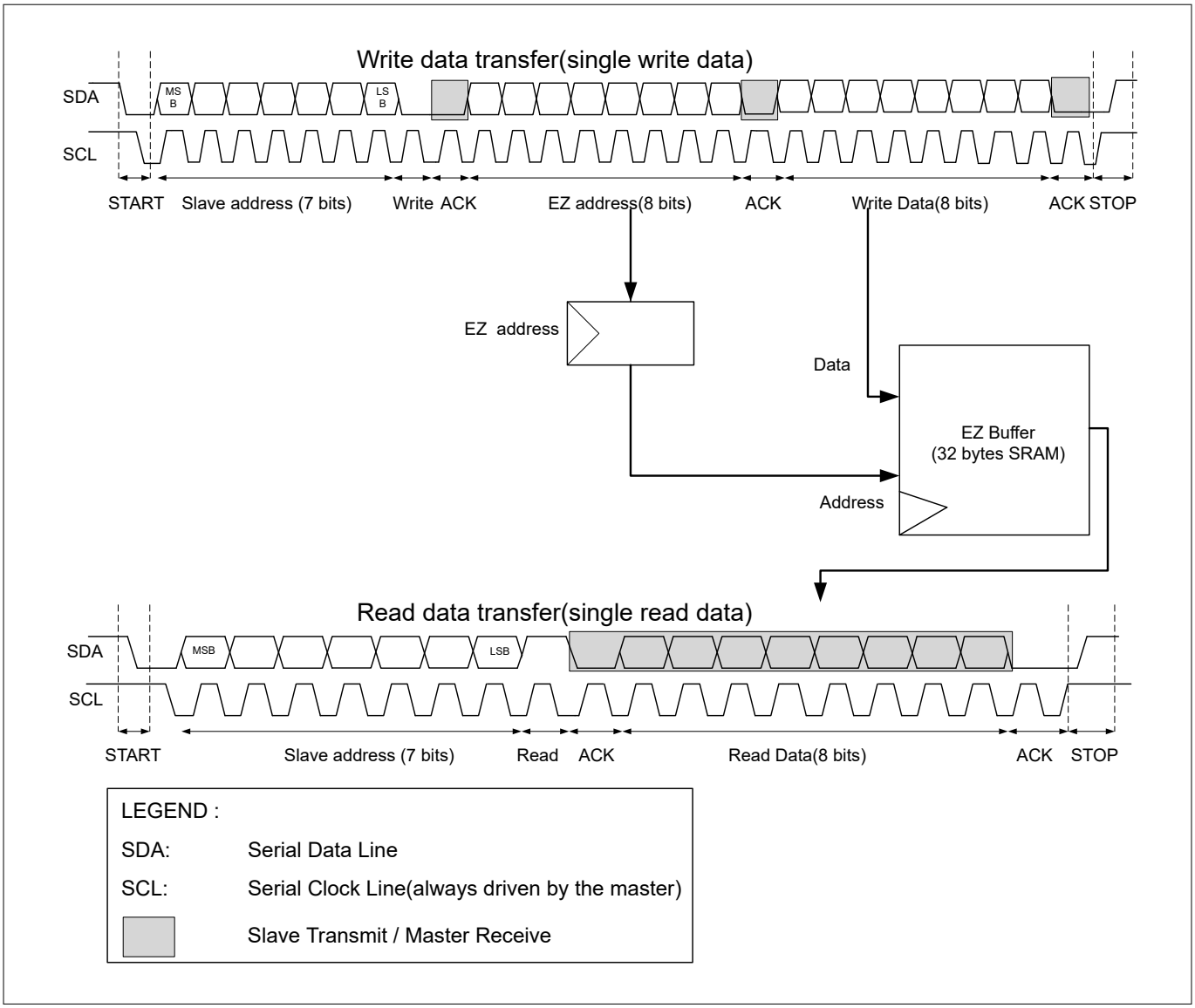


図 275 EZI2C 書き込み/読み出しデータ転送

EZ バッファモードでのシステム ディープスリープから CPU アクティブモードへの移行

SCBx_CTRL.EC_AM_MODE = 1、SCBx_CTRL.EC_OP_MODE = 0、EZ モード。

- S_NOT_READY_ADDR_NACK = 0, SCBx_I2C_CTRL.S_READY_ADDR_ACK = 1。クロックは、内部クロック ロジックが引き継ぐまでストレッチされ、その時点でアドレスが ACK され、マスターはデータの書き込みを開始できます。ディープスリープに入る前に CLK_SCB を無効にする必要があります。ディープスリープからのウェイクアップ時に CLK_SCB を再有効化する必要があります。
- S_NOT_READY_ADDR_NACK = 1, SCBx_I2C_CTRL.S_READY_ADDR_ACK = x.入力されたアドレスは、内部クロック ロジックが引き継ぐまで NACK されます。このような場合、PLL/FLL のロック時間により、内部クロックが正しい周波数になる保証はありません。これを避けるには、ディープスリープに入る前に CLK_SCB を無効にし、PLL/FLL が安定した後に再び無効にします。

26.5.5.3 コマンド レスポンス モード

このモードには、1 つのメモリ バッファ、ベース読み出しアドレス、現在の読み出しアドレス、ベース書き込みアドレス、およびメモリ バッファのインデックス付けに使用される現在の書き込みアドレスがあります。ベースアドレスは CPU が提供します。

現在のアドレスは、スレーブがメモリ バッファに順次アクセスする際のインデックスとして使用されます。メモリ バッファは 256 個の 8 ビット データ エレメントを保持します。ベースアドレスと現在のアドレスは[0～255]の範囲です。

CPU は SCBx_EZ_DATA レジスタをとおしてメモリ バッファに書き込みと読み出しを行います。これらはワードアクセスですが、ワードの最下位バイトのみが使用されます。

スレーブ インターフェースは、現在のアドレスを使用してメモリ バッファにアクセスします。書き込み転送 (I^2C START/RESTART) の開始時に、ベース書き込みアドレスが現在の書き込みアドレスにコピーされます。データ エレメントの書き込みは、現在の書き込みアドレス位置に行われます。書き込みアクセス後、現在のアドレスは'1'インクリメントされます。読み出し転送の開始時に、ベース読み出しアドレスが現在の読み出しアドレスにコピーされます。データ エレメントの読み出しは、現在の読み出しアドレス位置までです。読み出しデータ エレメントが送信された後、現在の読み出しアドレスが'1'インクリメントされます。

現在のアドレスが最後のメモリ バッファ アドレス (255) と等しい場合、現在のアドレスはインクリメントされません。それ以降の書き込みアクセスは、最後のバッファ アドレスに以前書き込まれた値を上書きします。それ以降の読み出しアクセスは、最後のバッファ アドレスで (同じ) 読み出し値を提供し続けます。バス マスターは、コマンド レスポンス モードにおけるメモリ バッファの容量を認識しておく必要があります。

ベースアドレスは SCBx_CMD_RESP_CTRL をとおして提供されます。現在のアドレスは SCBx_CMD_RESP_STATUS で確認できます。転送の終了時 (I^2C 停止)、ベースアドレスと現在のアドレスの差は、何回の読み出し/書き込みアクセスが実行されたかを示します。このブロックは、SCBx_INTR_I2C_EC および SCBx_INTR_SPI_EC レジスタ セクションにある転送の終了を識別するための割込みを提供します。

コマンド レスポンス モードの動作は、CPU アクティブおよびシステム ディープスリープ電力モードで利用可能です。コマンド レスポンス モードには 2 つの動作段階があります。

- 書き込みフェーズ - 書き込みフェーズは、START/RESTART で始まり、書き込みを示す読み出し/書き込みビットが'0'に設定されたスレーブ アドレスが続きます。スレーブの現在の書き込みアドレスは、スレーブのベース書き込みアドレスに設定されます。受信したデータ エレメントは、現在の書き込みアドレス メモリ位置に書き込まれます。各メモリ書き込みの後、現在の書き込みアドレスがインクリメントされます
- 読み出しフェーズ - 読み出しフェーズは、START/RESTART で始まり、読み出し/書き込みビットが読み出しを示す'1'に設定されたスレーブ アドレスが続きます。スレーブの現在の読み出しアドレスは、スレーブのベース読み出しアドレスに設定されます。送信されたデータ エレメントは、現在のアドレス メモリ位置から読み出されます。各読み出しデータ エレメントが転送されるたびに、現在の読み出しアドレスがインクリメントされます。

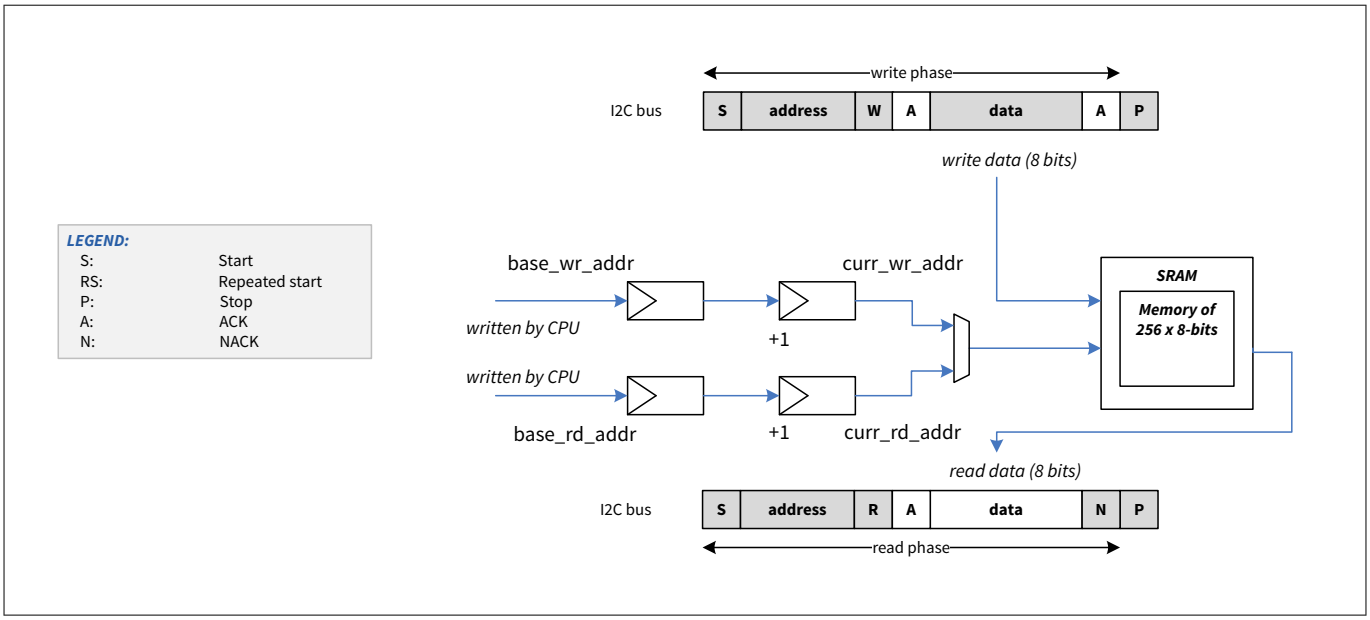


図 276　　I2C コマンド レスポンス モード

***注**:　　スレーブのベースアドレスは、マスターではなくCPU によって更新されます。*

26.5.6　　クロッキングとオーバーサンプリング

SCB I2C は、内部クロック動作モードと外部クロック動作モードの両方に対応しています。2 つのビットフィールド (SCB_CTRL レジスタの SCBx_CTRL.EC_AM_MODE および SCBx_CTRL_EC_OP_MODE) は SCB クロック モードを決定します。SCBx_CTRL.EC_AM_MODE は、I2C アドレスマッチングが内部クロック (0) か外部クロック (1) かを示します。I2C アドレス マッチングは、I2C プロトコルの最初の部分を構成します。SCBx_CTRL.EC_OP_MODE は、(I2C アドレスマッチング以外の) 残りのプロトコル動作が内部クロック (0) か外部クロック (1) かを示します。外部クロック動作モードは、I2C スレーブモードでのみサポートされます。

内部クロック動作は、プログラマブルクロック分周器を使用します。I2C の場合、マスターとスレーブの両方で整数クロック分周器を使用する必要があります。システム クロッキングの詳細については、クロッキング システムのセクションを参照してください。内部クロック モードはコマンド レスポンス モードをサポートしません。

***注**:　　PSOC™ Control C3 では、1 つの SCB のみが外部クロック動作モードをサポートします。*

SCBx_CTRL.EC_AM_MODE および SCBx_CTRL.EC_OP_MODE は以下の方法で設定できます。

- EC_AM_MODE = '0', SCBx_CTRL.EC_OP_MODE = '0': CPU アクティブ モード機能のみが必要な場合は、この設定を使用します。
  - FIFO モード: 対応
  - EZ モード: 対応
  - コマンド レスポンス モード: 非対応。スレーブはすべてのスレーブアドレスが NACK です。
- EC_AM_MODE = '1', SCBx_CTRL.EC_OP_MODE = '0': CPU アクティブ機能とシステムディープスリープ機能の両方が必要な場合は、この設定を使用してください。この設定は、I2C アドレスマッチングを外部クロック機能に依存し、メモリバッファへのアクセスを内部クロック機能に依存します。外部機能から内部機能への"ハンドオーバー"は、ACK/NACK またはクロック ストレッチ方式に依存します。前者は、現在の転送が終了することがあり、マスターのリトライに依存します。後者は、一致するアドレスを受信した後、現在の転送をストレッチします。このモードでは、マスターが NACK 生成 (およびリトライ) またはクロック ストレッチのいずれかをサポート

する必要があります。I2C アドレスが一致した場合、SCBx_INTR_I2C_EC.WAKE_UP = ‘1’に設定されます。関連するシステム ディープスリープ機能割込みにより、システムは CPU アクティブ電力モードになります。

- FIFO モード: FIFO モードを参照してください
- EZ モード: EZI2C モードを参照してください
- CMD_RESP モード: 非対応。スレーブはすべてのスレーブアドレスが NACK です。

- EC_AM_MODE = ‘1’, SCBx_CTRL.EC_OP_MODE = ‘1’。CPU アクティブとシステムディープスリープの両方の機能が必要な場合は、このモードを使用してください。スレーブが選択した場合、SCBx_INTR_I2C_EC.WAKE_UP = ‘1’に設定されます。関連するシステムのディープスリープ機能割込みにより、システムは CPU アクティブ電力モードになります。スレーブが選択解除された場合、SCBx_INTR_I2C_EC.EZ_STOP = '1'および/または SCBx_INTR_I2C_EC.EZ_WRITE_STOP = '1'に設定されます。
  - FIFO モード: 非対応
  - EZ モード: 対応
  - CMD_RESP モード: 対応

表 246　　クロック コンフィギュレーションとモード サポート

モード	SCBx_CTRL.EC_AM_MODE = '0'; SCBx_CTRL.EC_OP_MODE = '0'	SCBx_CTRL.EC_AM_MODE = ‘1’; SCBx_CTRL.EC_OP_MODE = '0'	SCBx_CTRL.EC_AM_MODE = '1'; SCBx_CTRL.EC_OP_MODE = '1'
FIFO モード	有り	有り	無し
EZ モード	有り	有り	有り
CMD_RESP モード	無し	無し	有り

外部クロック供給動作はシリアル インターフェースが供給するクロックを利用します。外部クロック モードは FIFO モードをサポートしません。SCBx_CTRL.EC_OP_MODE = '1'の場合、外部インターフェースロジックは外部インターフェースクロック (I2C SCL) でメモリバッファにアクセスします。これにより、CPU アクティブおよびシステム ディープスリープ電力モードで EZ および CMD_RESP モード機能を使用できます。

CPU アクティブ電力モードでは、メモリバッファは外部インターフェースロジック (I2C SCL 上) と CPU インターフェースロジック (システムペリフェラルクロック上) の間のアービトレーションを必要とします。このアービトレーションは、常に外部インターフェース ロジック (ホスト アクセス) を最優先にします。外部インターフェース ロジックは、I2C のシリアル インターフェース クロック/ビット期間を 1 つまたは 2 つ取ります。この間、内部ロジックはメモリ バッファへのアクセスを拒否されます。PSOC™ Control C3 は、この「サービス拒否」に対処するため、2 つのプログラム可能なオプションを提供しています。

- SCBx_CTRL.BLOCK = 1 の場合: メモリ バッファが許可され、外部インターフェース ロジックがアクセスを完了するまで、内部ロジックのメモリ バッファへのアクセスはブロックされます。100 kHz の I2C インターフェースの場合、シリアルインターフェースの 1 ビット期間の最大ブロッキング期間は 10 µs (48 MHz の SCB 入力クロックで約 208 クロックサイクル) です。このオプションは通常の SCB レジスタ機能を提供しますが、ブロッキング時間により内部バス待機状態が追加されます。
- SCBx_CTRL.BLOCK = 0 の場合: メモリ バッファへの内部ロジック アクセスはブロックされませんが、外部インターフェースのロジック アクセスと競合すると失敗します。読み出しアクセスは値 0xFFFF:FFFF を返し、書き込みアクセスは無視されます。このオプションは、追加の内部バス待機状態を導入しませんが、メモリ バッファへのアクセスが有効にならないことがあります。この場合、以下の障害が検出されます。
- 読み出し失敗: 読み出し失敗は、返される値が 0xFFFF:FFFF であるため、容易に検出できます。この値は、失敗しないメモリ バッファ読み出しアクセスが 0x0000:0000-0x0000:00FF の範囲の符号なしバイト値を返すため、一意です。
- 書き込み失敗: 書き込み失敗は、書き込まれたメモリ バッファの位置を読み返し、読み出された値が書き込まれた値と同じであることを確認することによって検出されます。

どちらのオプションでも、メモリ バッファへの競合する内部ロジック アクセスが発生すると、SCBx_INTR_TX.BLOCKED フィールドを'1' (書き込みアクセス用) に、SCBx_INTR_RX.BLOCKED フィールドを'1' (読み出しアクセス用) に設定します。これらのフィールドは、ステータス フィールドとしても割込み要因フィールド (関連するマスク フィールドが有効な場合) としても使用できます。

一連の読み出しまたは書き込みアクセスが実行され、SCBx_CTRL.BLOCKED が'0'の場合、すべての読み出し値の論理和を 0xFFFF:FFFF と比較し、SCBx_INTR_TX.BLOCKED と SCBx_INTR_RX.BLOCKED フィールドをチェックして、(一連の) 書き込みまたは読み出し操作で障害の有無を判断することにより、障害が検出されます。

26.5.6.1 グリッチフィルタリング

PSOC™ Control C3 SCB I2C には、アナログとデジタルのグリッチ フィルターがあります。i2c_scl_in および i2c_sda_in 入力信号 (AF_in) にアナログ グリッチ フィルターが適用され、最大 50 ns のグリッチをフィルタリングします。i2c_sda_out 出力信号 (AF_out) にもアナログ グリッチ フィルターが適用されます。アナログ グリッチ フィルターの有効/無効は SCBx_I2C_CFG レジスタを用いて選択されます。このレジスタの_TRIM ビット フィールドは変更せず、_SEL ビット フィールドのみを変更します。

デジタル グリッチ フィルターは、i2c_scl_in と i2c_sda_in 入力信号 (DF_in) に適用されます。デジタル グリッチ フィルターは SCBx_RX_CTRL.MEDIAN レジスタを用いて有効にできます。

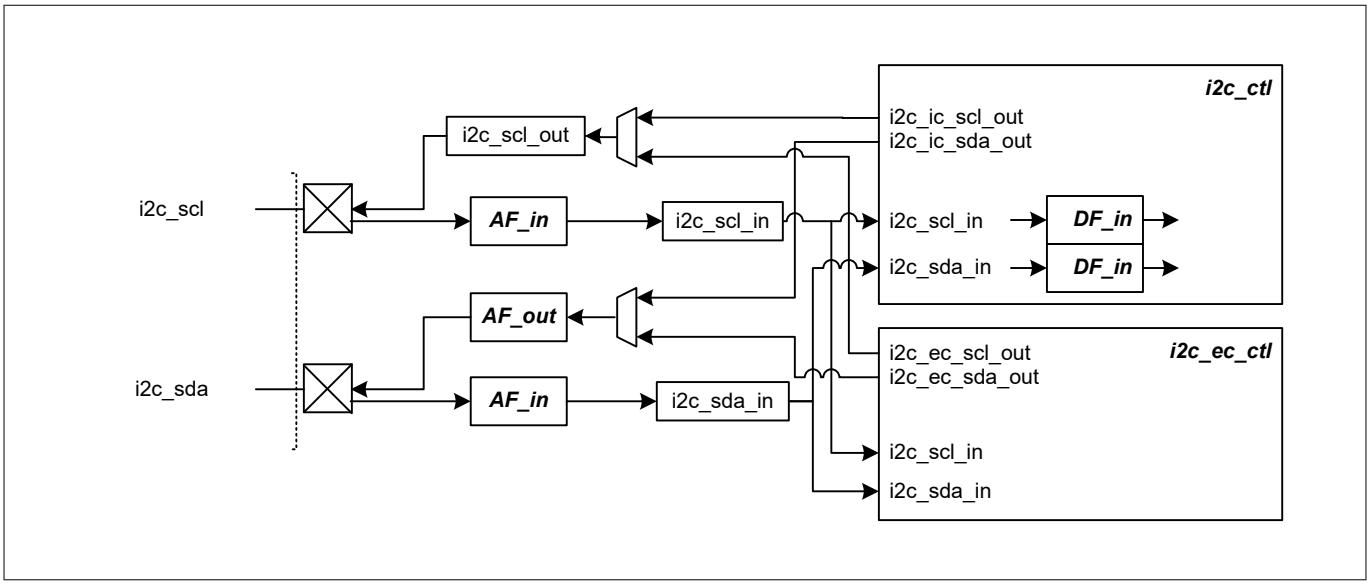


図 277 I2C グリッチフィルタリング接続

次の表は、グリッチフィルターの便利な組み合わせの一覧です。

表 247 グリッチフィルターの組み合わせ

AF_in	AF_out	DF_in	備考
0	0	1	内部クロック モードで動作し、ファストモード プラス (1 MHz スピード モード) のマスターで使用されます。
1	0	0	内部クロック動作時に使用 (SCBx_CTRL.EC_OP_MODE='0')

(続く)

表 247 (続き) グリッチフィルターの組み合わせ

AF_in	AF_out	DF_in	備考
1	1	0	外部クロックモードで動作する場合に使用 (SCBx_CTRL.EC_OP_MODE= '1')。スレーブモードのみ。

EC_OP_MODE = 1 で動作する場合、100 kHz、400 kHz、および 1000 kHz モードでは、AF_out に以下の設定が必要です。

AF_in	AF_out	DF_in	備考
1	1	0	100-kHz mode: SCBx_I2C_CFG.SDA_OUT_FILT_SEL = 3 400-kHz mode: SCBx_I2C_CFG.SDA_OUT_FILT_SEL = 3 1000-kHz mode: SCBx_I2C_CFG.SDA_OUT_FILT_SEL = 1

26.5.6.2 オーバーサンプリングとビットレート

内部クロック マスター

PSOC™ Control C3 は I2C クロックを SCB 入力クロックのオーバーサンプリング倍数として実装しています。マスターモードでは、クロック ブロックが I2C 周波数を決定します。PCB、デバイス、ブロック (アナログとデジタルのグリッチ フィルターを含む) の配線遅延はすべて、信号インターフェースのタイミングに影響します。マスターモードでは、ブロックは CLK_SCB から離れて動作し、SCL のハイ SCBx_I2C_CTRL.HIGH_PHASE_OVS およびロー SCBx_I2C_CTRL.LOW_PHASE_OVS 時間にプログラム可能なオーバーサンプリング率を使用します。

表 248 I2C マスターモードにおける I2C 周波数とオーバーサンプリング要件

AF_in	AF_out	DF_in	モード	対応周波数	SCBx_I2C_CTRL.LOW_PHASE_OVS	SCBx_I2C_CTRL.HIGH_PHASE_OVS	入力クロック周波数
0	0	1	100 kHz	[62, 100] kHz	[9, 15]	[9, 15]	[1.98-3.2] MHz
			400 kHz	[264, 400] kHz	[13, 15]	[7, 15]	[8.45-10] MHz
			1000 kHz	[447, 1000] kHz	[8, 15]	[5, 15]	[14.32-25.8] MHz
1	0	0	100 kHz	[48, 100] kHz	[7, 15]	[7, 15]	[1.55-3.2] MHz
			400 kHz	[244, 400] kHz	[12, 15]	[7, 15]	[7.82-10] MHz
			1000 kHz	未対応			

表は、I2C バスの最悪の状態を想定しています。以下の式を使用して、独自のシステムの設定を決定できます。システムの SCL ラインと SDA ラインの立ち上りと立ち下りの時間の測定などです。

tCLK_SCB (Min) = (tLOW + tF)/SCBx_I2C_CTRL.LOW_PHASE_OVS

clk_scb がこれより速いと、I2C 仕様の tLOW に違反します。tF はシステムで測定する必要があります。

tCLK_SCB (Max) = (tVD - tRF - 100 nsec)/3 (アナログフィルター有効、デジタルフィルター無効時)

tCLK_SCB (Max) = (tVD - tRF)/4 (アナログフィルター無効、デジタルフィルター有効時)

tRF は、立ち上り時間と立ち下り時間のいずれかの最大値です。clk_scb がこの周波数より遅い場合、tVD は違反となります。

内部クロック スレーブ

スレーブモードでは、I2C 周波数は入力される I2C SCL 信号によって決定されます。適切な動作を保証するために、CLK_SCB は I2C バス周波数よりかなり高くなければいけません。マスター モードとは異なり、このモードではプログラマブル オーバーサンプリング率は使用しません。表は、チップ内部遅延を含む I2C バスのワーストケース条件を想定しています。

表 249　　I2C スレーブモードでの SCB 入力クロック要件

AF_in	AF_out	DF_in	モード	CLK_SCB 周波数範囲
0	0	1	100 kHz	[1.98-12.8] MHz
			400 kHz	[8.45-17.14] MHz
			1000 kHz	[14.32-44.77] MHz
1	0	0	100 kHz	[1.55-12.8] MHz
			400 kHz	[7.82-15.38] MHz
			1000 kHz	[15.84-89.0] MHz

tCLK_SCB (Max) = (tVD - tRF - 100 nsec) / 3 (アナログフィルター有効、デジタルフィルター無効時)

tCLK_SCB (Max) = (tVD - tRF) / 4 (アナログフィルター無効、デジタルフィルター有効時)

tRF は、立ち上り時間と立ち下り時間のいずれかの最大値です。CLK_SCB がこの周波数より遅い場合、tVD は違反となります。

CLK_SCB の最小周期は、以下の式のいずれかによって決定されます。

tCLK_SCB (MIN) = (tSU_DAT (min) + tRF) /16

マスター-スレーブ

このモードでは、SCB がマスター デバイスとして動作している場合、ブロックが I2C 周波数を決定します。SCB がスレーブ デバイスとして動作している場合、ブロックは I2C 周波数を決定しません。代わりに、入力される I2C SCL 信号が決定します。

マスター モードとスレーブ モードの両方での動作を保証するために、上の表を使ってマスターとスレーブの両方で動作するクロック周波数を選択します。

26.5.7　　ループバック

マスター スレーブ モードでは、SCB は内部 SCL をサポートし、SDA ラインはペリフェラル内部で配線されます。そのため、他の I2C デバイスの影響を受けることはありません。SCBx_I2C_CTRL.LOOPBACK レジスタを使用して設定します。

26.5.8　　I2C の有効化と初期化

26.5.8.1 I2C FIFO モードの設定

I2C インターフェースを設定するためには、次の手順を実行します。

1. SCBx_I2C_CTRL レジスタを使用して、プロトコル固有の情報をプログラムしてください。これには、マスター スレーブ機能 (MASTER_MODE、SLAVE_MODE) の選択も含まれます。
2. SCBx_TX_CTRL および SCBx_RX_CTRL レジスタを使用して汎用送受信側情報をプログラムしてください。
   a. データフレームの幅を指定 (DATA_WIDTH = 7)
   b. 最初に送信/受信されるビットを MSb に指定 (MSB_FIRST = 1)
3. バイトモードを有効にするためには SCBx_CTRL.MEM_WIDTH を'00'に設定
4. SCBx_TX_FIFO_CTRL および SCBx_RX_FIFO_CTRL レジスタを使用して送受信側 FIFO をプログラムしてください。
   a. トリガーレベルの設定 (TRIGGER_LEVEL)
   b. 送信側と受信側 FIFO、シフトレジスタのクリア (CLEAR)
5. I2C ブロックを有効にするために SCBx_CTRL レジスタをプログラムし、I^2C モードを選択してください。I2C レジスタの詳細については、PSOC™ Control C3 レジスタ RM を参照してください。

26.5.8.2 EZ モードと CMD_RESP モードの設定

EZ および CMD_RESP モードのために I2C ブロックを設定するためには、次の I2C レジスタビットを設定してください。

1. EZ モードの場合は、SCBx_CTRL.EZ_MODE レジスタに'1'を書き込んで EZI2C モードを選択してください。
2. CMD_RESP モードの場合は、SCBx_CTRL.CMD_RESP に'1'を書き込んで CMD_RESP モードを選択してください。
3. S_READY_ADDR_ACK (ビット 12) と SCBx_I2C_CTRL.S_READY_DATA_ACK (ビット 13) レジスタの設定

26.5.9 I/O パッド接続

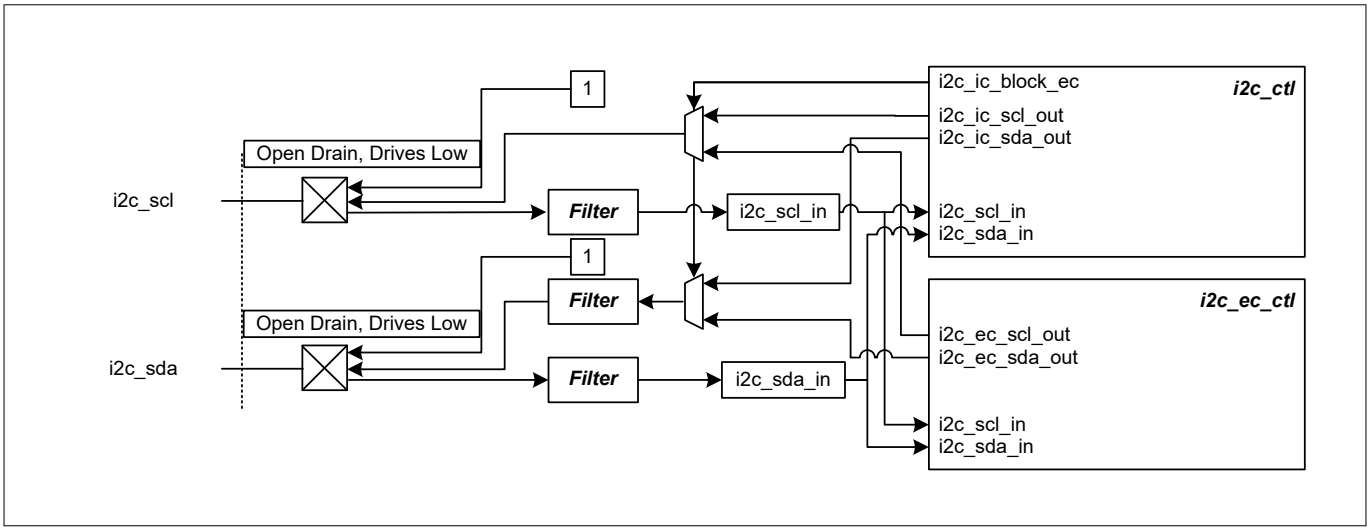


図 278 I2C I/O パッド接続

表 250 I2C I/O パッドの説明

I/O パッド	ドライブモード	オンチップ I/O 信号	使用法
i2c_scl	オープンドレイン、LOW 駆動	i2c_scl_in	クロックを受信
		i2c_scl_out	クロックの送信
		i2c_scl_out_en = 1	i2c_scl_out_en は High (有効) に固定されます。
i2c_sda	オープンドレイン、LOW 駆動	i2c_sda_in	データ受信
		i2c_sda_out	データ送信
		i2c_sda_out_en = 1	i2c_sda_out_en は High (有効) に固定されます。

26.5.10 レジスタの概要

I2C インターフェースは、コンフィギュレーション、コントロール、ステータスの各レジスタの読み出し/書き込みを行うことで制御されます。これらのレジスタの詳細については、PSOC™ Control C3 レジスタ リファレンス マニュアルを参照してください。

26.6 SCB 割込み

SCB は様々なイベントでの割込み発生に対応しています。SCB ブロックが生成する割込みは、動作モードによって異なります。

表 251 SCB 割込み

割込み	機能	CPU アクティブ/システムディープスリープ	レジスタ
interrupt_master	I2C マスターおよび SPI マスター機能	CPU アクティブ	SCBx_INTR_M, SCBx_INTR_M_SET, SCBx_INTR_M_MASK, SCBx_INTR_M_MASKED
interrupt_slave	I2C スレーブおよび SPI スレーブ機能	CPU アクティブ	SCBx_INTR_S, SCBx_INTR_S_SET, SCBx_INTR_S_MASK, SCBx_INTR_S_MASKED
interrupt_tx	UART 送信および TX FIFO 機能	CPU アクティブ	SCBx_INTR_TX, SCBx_INTR_TX_SET, SCBx_INTR_TX_MASK, SCBx_INTR_TX_MASKED
interrupt_rx	UART 受信および RX FIFO 機能	CPU アクティブ	SCBx_INTR_RX, SCBx_INTR_RX_SET, SCBx_INTR_RX_MASK, SCBx_INTR_RX_MASKED
interrupt_i2c_ec	外部クロック I2C スレーブ機能	システムディープスリープ	SCBx_INTR_I2C_EC, SCBx_INTR_I2C_EC_MASK, SCBx_INTR_I2C_EC_MASKED
interrupt_spi_ec	外部クロック SPI スレーブ機能	システムディープスリープ	SCBx_INTR_SPI_EC, SCBx_INTR_SPI_EC_MASK, SCBx_INTR_SPI_EC_MASKED

26.6.1 SPI 割込み

SPI 割込みは、マスター割込み、スレーブ割込み、TX 割込み、RX 割込み、外部クロック (EC) モード割込みに分類されます。各割込み出力は、そのセクションに分類されたすべての可能な割込みソースのグループの論理和です。例えば、TX 割込み出力は、すべての TX 割込みソースとなる信号グループの論理和です。この信号は任意の有効な TX 割込みソースが真の場合、HIGH になります。SCB はまた、割込みソースを決定するために使用できる割込み要因レジスタ (SCBx_INTR_CAUSE) を提供します。割込みレジスタは、対応するビットフィールドに'1'を書き込むことでクリアされます。ある種の割込みソースは、たとえその割込みソースがクリアされたとしても、条件が満たされる限り再びトリガーされることに注意してください。例えば、TX_FIFO_EMPTY は、割込みソースがクリ

アされても、送信 FIFO が空である限りセットされます。割込みレジスタの詳細については、PSOC™ Control C3 レジスタ リファレンス マニュアルを参照してください。SPI は以下のイベントでの割込みに対応しています。

- SPI マスター割込み (SCBx_INTR_M)
  - SPI マスター転送完了 (SPI_DONE)
- SPI スレーブ割込み (SCBx_INTR_S)
  - EZSPI 書き込み転送が発生した後、SPI スレーブ セレクト解除 (SPI_EZ_WRITE_STOP)
  - EZSPI 転送が発生した後、SPI スレーブ セレクト解除 (SPI_EZ_STOP)
  - SPI バス エラー: SPI 転送でスレーブが予期せず選択解除。ファームウェアは、このエラーに対して TX と RX の FIFO をクリアすることを決定するかもしれません。(SPI_BUS_ERROR)
- SPI TX (SCBx_INTR_TX)
  - TX FIFO には SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER) で指定された値より少ないエントリーがあります。
  - TX FIFO がフルではない (NOT_FULL)
  - TX FIFO が空 (EMPTY)

26.6.2 UART 割込み

UART 割込みは TX 割込みと RX 割込みに分類されます。各割込み出力は、そのセクションに分類されたすべての可能な割込みソースのグループの論理和です。例えば、TX 割込み出力は、すべての TX 割込みソースとなる信号グループの論理和です。この信号は任意の有効な TX 割込みソースが真の場合、HIGH になります。SCB はまた、割込みソースを決定するために使用できる割込み要因レジスタ (SCBx_INTR_CAUSE) を提供します。割込みレジスタは、対応するビットフィールドに'1'を書き込むことでクリアされます。ある種の割込みソースは、たとえその割込みソースがクリアされたとしても、条件が満たされる限り再びトリガーされることに注意してください。例えば、TX_FIFO_EMPTY は、割込みソースがクリアされても、送信 FIFO が空である限りセットされます。割込みレジスタの詳細については、PSOC™ Control C3 レジスタを参照してください。UART ブロックは以下のイベントで割込みを生成します。

- UART TX (SCBx_INTR_TX)
  - TX FIFO には SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER) で指定された値より少ないエントリーがあります。
  - TX FIFO がフルではない (NOT_FULL)
  - TX FIFO が空 (EMPTY)
  - TX FIFO オーバーフロー (OVERFLOW)
  - TX FIFO アンダーフロー (UNDERFLOW)
  - SmartCard モードで TX が NACK を受信 (UART_NACK)
  - TX 完了。これは、UART が TX FIFO 内の全データの転送を完了し、最後のストップ フィールドが送信された時 (TX FIFO と送信シフター レジスタの両方が空) に起こります。(UART_DONE)
  - アービトレーションが失われた (LIN またはスマートカードモード) (UART_ARB_LOST)
- UART RX (INTR_RX)
  - RX FIFO には SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER) で指定された値より多いエントリーがあります。
  - RX FIFO が空ではない (NOT_EMPTY)
  - RX FIFO がフル (FULL)
  - RX FIFO オーバーフロー (OVERFLOW)
  - RX FIFO アンダーフロー (UNDERFLOW)
  - 受信したデータ フレームでフレーム エラー (FRAME_ERROR)

- 受信したデータフレームでパリティエラー (PARITY_ERROR)
- LIN ボーレート検出完了 (BAUD_DETECT)
- LIN ブレーク正常検出 (BREAK_DETECT)

26.6.3 I2C 割込み

I2C 割込みは、マスター割込み、スレーブ割込み、TX 割込み、RX 割込み、外部クロック (EC) モード割込みに分類されます。各割込み出力は、そのセクションに分類されたすべての可能な割込みソースのグループの論理和です。例えば、TX 割込み出力は、すべての TX 割込みソースとなる信号グループの論理和です。この信号は任意の有効な TX 割込みソースが真の場合、HIGH になります。SCB はまた、割込みソースを決定するために使用できる割込み要因レジスタ (SCBx_INTR_CAUSE) を提供します。割込みレジスタは、対応するビットフィールドに'1'を書き込むことでクリアされます。ある種の割込みソースは、たとえその割込みソースがクリアされたとしても、条件が満たされる限り再びトリガーされることに注意してください。例えば、TX_FIFO_EMPTY は、割込みソースがクリアされても、送信 FIFO が空である限りセットされます。割込みレジスタの詳細については、PSOC™ Control C3 レジスタ リファレンス マニュアルを参照してください。I2C ブロックは、以下の条件で割込みを発生させます。

- I2C Master (SCBx_INTR_M)
  - I2C マスター ロスト アービトレーション (I2C_ARB_LOST)
  - I2C マスター受信 NACK (I2C_NACK)
  - I2C マスター受信 ACK (I2C_ACK)
  - I2C マスター送信 STOP (I2C_STOP)
  - I2C バスエラー (予期しない停止/開始状態を検出) (I2C_BUS_ERROR)
- I2C スレーブ (SCBx_INTR_S)
  - I2C スレーブ ロスト アービトレーション (I2C_ARB_LOST)
  - I2C スレーブ受信 NACK (I2C_NACK)
  - I2C スレーブ受信 ACK (I2C_ACK)
  - I2C スレーブ受信書き込み STOP (I2C_WRITE_STOP)
  - I2C スレーブ受信 STOP (I2C_STOP)
  - I2C スレーブ受信 START (I2C_START)
  - I2C スレーブアドレス一致 (I2C_ADDR_MATCH)
  - I2C スレーブ汎用コールアドレス受信 (I2C_GENERAL)
  - I2C バスエラー - 予期しない停止/開始状態を検出 (I2C_BUS_ERROR)
- I2C TX (SCBx_INTR_TX)
  - TX FIFO には SCBx_TX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER) で指定された値より少ないエントリーがあります。
  - TX FIFO がフルではない (NOT_FULL)
  - TX FIFO が空 (EMPTY)
  - TX FIFO オーバーフロー (OVERFLOW)
  - TX FIFO アンダーフロー (UNDERFLOW)
  - EZ メモリ (EZ データアクセス) がブロックされた (BLOCKED)
- I2C RX (SCBx_INTR_RX)
  - RX FIFO には SCBx_RX_FIFO_CTRL.TRIGGER_LEVEL (TRIGGER) で指定された値より多いエントリーがあります。
  - RX FIFO が空ではない (NOT_EMPTY)
  - RX FIFO がフル (FULL)
  - RX FIFO オーバーフロー (OVERFLOW)
  - RX FIFO アンダーフロー (UNDERFLOW)
- I2C 外部クロック (SCBx_INTR_I2C_EC)
  - アドレス一致時のウェイクアップ要求 (WAKE_UP)

- 各転送終了時の I2C STOP 検出 (EZ_STOP)
- 書き込み転送終了時の I2C STOP 検出 (EZ_WRITE_STOP)
- 読み出し転送終了時の I2C STOP 検出 (EZ_READ_STOP)

27 高性能プログラマブル アナログ サブシステム (HPPASS)

高性能プログラマブル アナログ サブシステム (HPPASS) は、電圧を感知してデジタル コードに変換し、ADC 変換の後処理をすることができます。処理された判定/データを保存し、自律的にデータを CPU にストリーミングできます。また、自律的な (CPU から独立した) 動作も可能です。

27.1 特長

PSOC™ Control C3 MCU HPPASS の特長は、以下のとおりです。

- 12 ビット SAR ADC:
  - シングル チャネルで動作する場合、ADC の最高速度は 12 Msps です。
  - 外部アナログ信号の同期サンプリングが可能な 16 のサンプラー (12x ダイレクト I/O 接続と 4x マルチ接続を含みます)
  - サンプラーはすべて 1 倍、3 倍、6 倍、12 倍のゲインに対応
  - 最大 16 の専用アナログ パッド、最大 16 のパラレル サンプル ステージに接続可能
  - さらに 2 つの GPIO をアナログ マルチプレクサ (AMUX) のアナログ入力として使用できます。
  - 2 つの内部電圧信号 (温度センサーと DAC 出力電圧) は、アナログ マルチプレクサ (AMUX) によって多重化されたサンプラーに接続できます。
  - AMUX は、SAR シーケンサー、ファームウェア、または自律型コントローラー (AC) により制御できます。
  - 21 チャネル: 専用アナログ入力用 16 チャネル、追加 GPIO 用 2 チャネル、内部電圧信号用 3 チャネル
  - 後処理
    - 疑似差動変換
    - 結果の符号付き/符号なし形式
    - 2 つの 16 タップ有限インパルス応答 (FIR) フィルター
    - 設定可能なサンプル数 (2、4、8、16) のハードウェア平均化 (結果蓄積、シフト機能なし)
    - 1、2、4 つの FIFO に分割可能な 32 のエントリー FIFO
    - 8 つの限界検出リソース
  - SAR シーケンサー
    - 最大 8 つのスキャン グループ
    - グループ内で最大 16 のサンプラーが同期サンプリング可能
    - 各グループは、4 つのマルチプレクス サンプラーの入力を選択できます。
    - 各グループは、ファームウェア、他のペリフェラル (TCPWM および GPIO) からのトリガー、または自律型コントローラー (AC) からのトリガーができます。
    - 各グループは、3 つのプログラム可能なサンプル時間から 1 つを選択できます。
    - グループごとにプログラム可能な 2 つの優先順位
    - どのグループも連続モードに対応
  - オフセット、直線性、ゲインのセルフ キャリブレーション
  - 割込み生成
    - スキャン グループ完了
    - スキャン グループ トリガー衝突、スキャン グループ ホールド違反
    - FIR と FIFO レベル
    - FIFO オーバーフロー/アンダーフロー
    - 限界検出
    - 結果のオーバーフロー

  - 他のオンチップ ペリフェラル (DMA、GPIO、TCPWM など) をトリガーできる 4 つの FIFO レベル トリガー
  - オンチップ温度センサー
- 5 つのコンパレーターとスロープ ジェネレーター (CSG)
  - 各 CSG スライスは 10 ビット 30 Msps DAC で構成され、コンパレーターのリファレンスを生成
  - 各コンパレーターの正入力は、2 つのアナログ入力のいずれかを選択可能
  - 各コンパレーターの負入力は、DAC 出力または 2 つのアナログ入力のいずれかを選択可能
  - 2 つの 128 のエントリー波形ルックアップテーブル (LUT)
  - DAC は直接書き込みモード、バッファ モード、ヒステリシス モード、スロープ生成モード、LUT 波形生成モードに対応
  - DAC の起動は、ファームウェア、他のペリフェラルからのトリガー、または自律型コントローラーによってトリガー可能
  - DAC の更新は、ファームウェア、他のペリフェラルからのトリガー、自律型コントローラー、コンパレーター出力によってトリガー可能
  - コンパレーターの後処理
    - 設定可能なコンパレーター出力極性
    - コンパレーターのブランキング
    - コンパレーター出力のデグリッチ
    - 設定可能なコンパレーター割込みおよび AC トリガーを生成
    - 更新トリガーとしてコンパレーター出力または反転コンパレーターを選択可能
  - DAC 出力は AMUX により ADC にルーティング可能
  - 割込み生成
    - 比較出力
    - DAC ハードウェア開始
    - DAC スロープ完了、DAC バッファ空
- 自律型コントローラー (AC) は、16 のエントリーの状態遷移表からなる有限状態機械 (FSM) シーケンサーで、タイマーやイベントに基づいて、あらかじめ定義された様々な状態を適切に遷移させます。自律型コントローラーは以下の機能を実行できます。
  - 内部ブロック (SAR ADC、CSG) の有効と無効、ファームウェアへの BLOCK_READY ステータスの提供
  - プログラムされた動作を実行するハードウェアまたはファームウェアのトリガー
  - 内部ブロックのいずれかをトリガー
  - ハードウェア遅延の実装
  - 時限ループの周期的動作 (例えばトリガー)。
  - 内部ブロックから条件を受け取り、そのイベントに対処します。
  - CPU に AC ベースの割込みを発生
  - 最大 5 つの GPIO へのデジタル出力
- ファームウェアレベルトリガー、ファームウェアパルストリガー、ハードウェアトリガー A (マルチプレクサトリガー)、ハードウェアトリガー B (1 対 1 のトリガー) に設定可能な 8 つの入力トリガー
- 8 つの出力レベル トリガー (それぞれ CSG コンパレーター出力と SAR ADC 限界検出出力を選択可能)
- 8 つの出力パルス トリガー (各パルス トリガーは ADC グループ完了、FIR 完了、AC トリガーから 1 つ選択可能)

27.2 アーキテクチャ

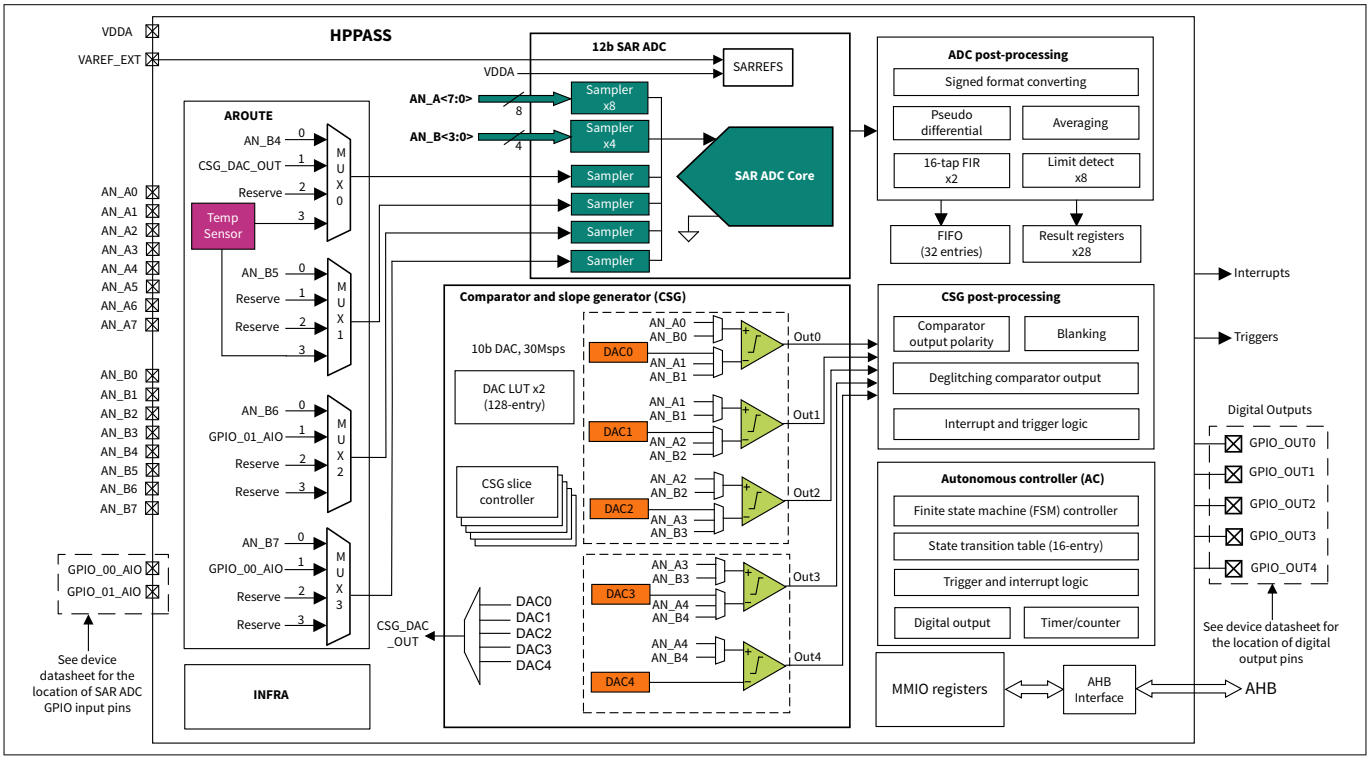


図 279 HPPASS のブロック図

図 279 に HPPASS のブロック図を示します。このブロックは、12 ビット SAR ADC、5 つの 10 ビット 30 Msps DAC と 5 つの高速コンパレーターを備えたコンパレーター＆スロープ ジェネレーター (CSG)、CPU や外部の介入を必要とせずにハードウェア制御とステータスを提供する自律型コントローラー (AC) で構成されています。HPPASS には 16 本の専用アナログ入力ピン (AN_A0 - AN_A7, AN_B0 - AN_B7) があり、SAR ADC はさらに AMUX による 2 本の GPIO 入力ピンと 5 本の GPIO によるデジタル出力に対応しています (SAR ADC の GPIO 入力ピンとデジタル出力ピンの詳細については PSOC™ Control C3 MCU データシートを参照)。

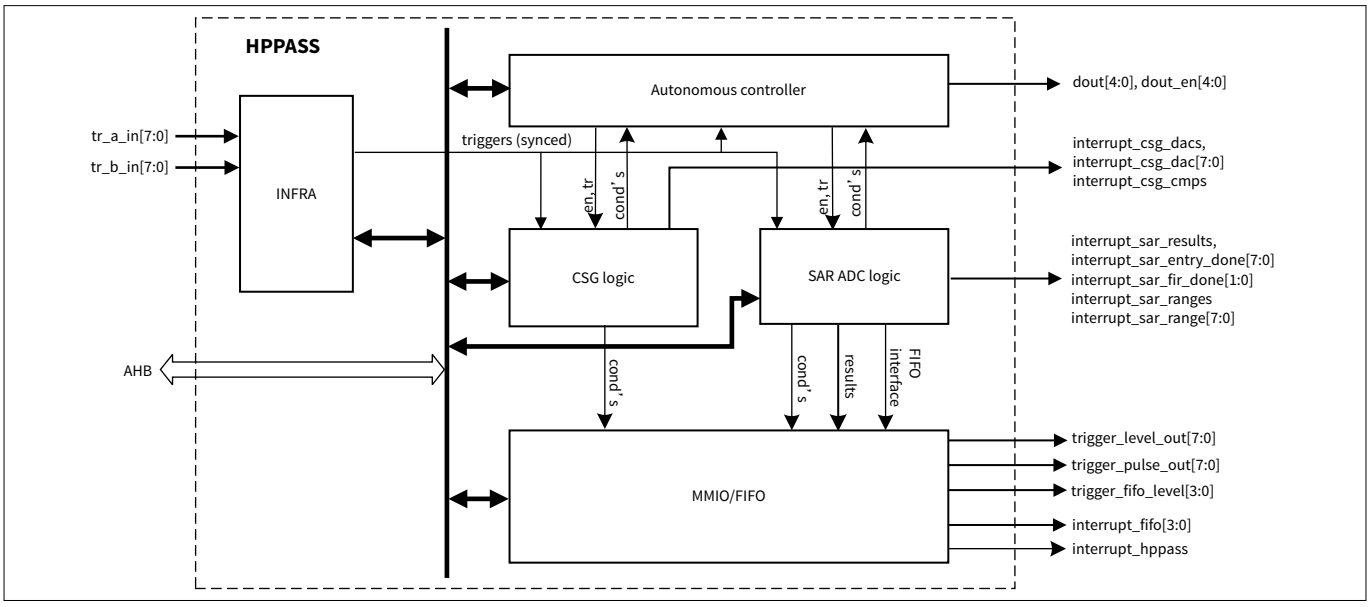


図 280 HPPASS デジタル トップレベル図

図 280 に、HPPASS デジタル システムのトップ レベルの構造と、その接続性を簡略化して示します。

トリガー

ペリフェラルからの入力トリガーはレベル (コンパレーター ブランキング用)、またはパルス (SAR ADC シーケンサーおよび CSG DAC 更新用) です。出力トリガーには、DMA 用のレベルベース FIFO トリガーや、コンパレーターステートや SAR ADC 限界検出などの一般的なパルストリガーがあります。さらに、TCPWM KILL のために、コンパレーターからペリへのレベルベースの専用低レイテンシトリガーがあります。

入力トリガーは 16 あります ("A"と"B")。デフォルトでは、これらの入力は非同期とみなされ、ダブルシンクされますが、シングルシンクのオプションもあります。詳細については、入力トリガーを参照してください。

HPPASS から TCPWM へのトリガーはレベルベースで、コンパレーター出力と SAR ADC の限界検出信号を含みます。これは完全な順列可能性を持っているので、これらの信号のどれでも、これらの tr_level_out 接続のどれにでもルーティングし、OR 接続できます。出力トリガーの詳細については トリガー セクションを参照してください。

HPPASS から DMA へのトリガーはパルス ベースで、DMA をとおして SAR ADC の結果をシステム メモリにコピーできます。

HPPASS FIFO はレベル ベースの DMA トリガーを持ち、これは FIFO 内で利用可能な設定された結果数を反映します。

割込み

割込みには個別割込みと複合割込みがあります。複合割込みを使用する場合は、どのイベントが割込みを発生させたかを知るために、関連する INTR レジスタを読み出します。DAC バッファリングとスロープ生成、およびコンパレーター用の割込みがあります。SAR ADC の割込みは SAR 結果の有無と限界検出ステータスを含みます。エラー状態は 1 つの一般的な割込みにまとめられています。詳細は、SAR ADC 割込みの章と CSG 割込みの章を参照してください。

以下のセクションでは、インフラストラクチャー (INFRA)、SAR ADC、コンパレーターとスロープジェネレーター (CSG)、自律型コントローラー (AC)、温度センサーを含む HPPASS の主要サブブロックの詳細について説明します。

27.2.1 インフラストラクチャー (INFRA)

下図は HPPASS INFRA のブロック図です。

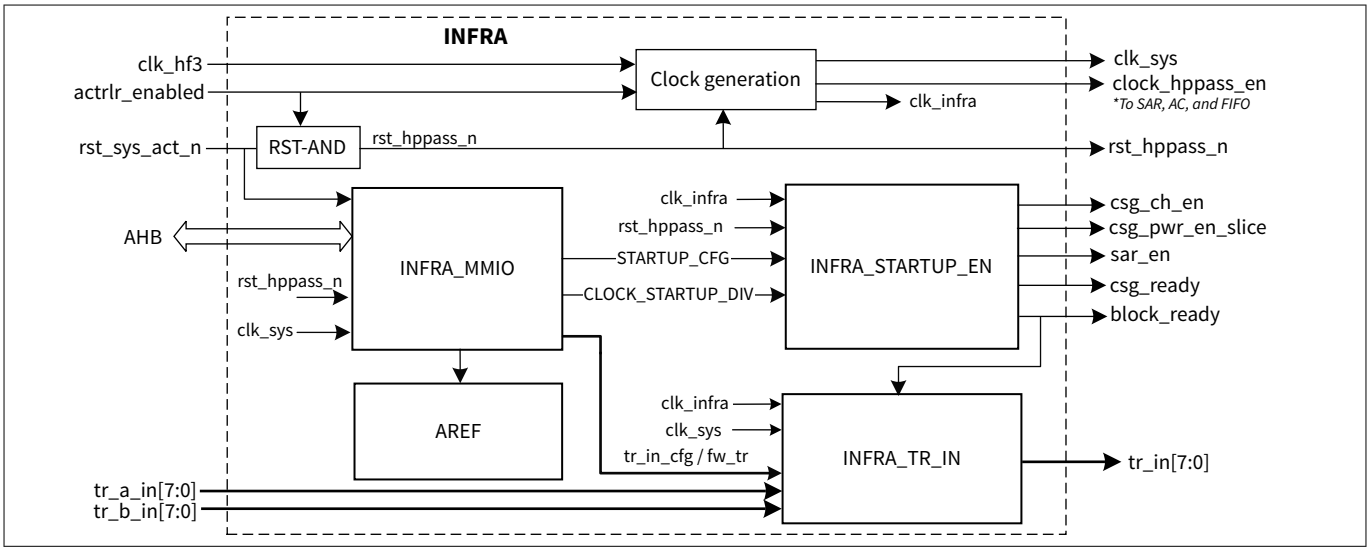


図 281 HPPASS INFRA ブロック図

インフラストラクチャーは、以下の 4 つの主要ブロックを含み、内部で使用するクロック (clk_sys と clk_hf3 から 1/2 固定である clock_hppass_en) とリセット (rst_hppass_n) を生成します。HPPASS クロックの詳細については、クロックの章を参照してください。

- INFRA_STARTUP_EN: SAR を有効にするタイミングを制御し、CSG サブブロックのスタートアップ シーケンスを提供する起動ロジック

- INFRA_TR_IN: 入力トリガーの構造。ハードウェア トリガーとファームウェア トリガーの両方に対応
- AREF: ローカル バンドギャップを含むアナログ リファレンス ブロックにより、HPPASS 内部アナログ回路の正確な電圧 電流リファレンスに対応
- INFRA_MMIO: AREF 制御、起動ロジック、入力トリガー制御用の AHB インターフェースとレジスタ

27.2.1.1 起動

INFRA は、起動時に SAR および CSG ブロックをサポートします。SAR と CSG の両ブロックは、AREF の準備が整うのを待ってから有効になり、その後は独立しています。起動シーケンスは INFRA の起動タイマーで計時されます。

AREF は HPPASS_INFRA_AREFv2_AREF_CTRL レジスタの ENABLED ビットを'1'に設定することで有効になります。

HPPASS のグローバル イネーブルは HPPASS_ACTRLR_CTRL レジスタの ENABLED ビットです。このビットが 0 のとき、すべてのアナログ ブロックはパワーダウンし、ミッション ロジックとステータスはリセット状態に保持され、ミッション クロックはゲートオフされます。利用可能な機能は AHB バスのみです。HPPASS が無効の時、SAR ADC のキャリブレーション レジスタは書き込み不可能であることに注意してください。

起動は、AC を有効にし、AC 起動命令 (複数可) を実行することによって開始されます (AC 実行前に AREF_EN が有効になっている場合)。詳細は、AC の初期化の章を参照してください。特定のアプリケーションに必要なブロックを有効にするためには、BLOCK_READY 条件の WAIT_FOR アクションとともに、少なくとも 1 つの AC 命令が必要です。AC インストラクションの詳細については、自律型コントローラー (AC) の章を参照してください。

スタートアップ ブロックは、HPPASS のスタートアップのタイミングをとるために、クロック、FSM、カウンター、その他のロジックを分割し、BLOCK_READY コンディションを生成して AC に返します。
HPPASS_ACTRLR_BLOCK_STATUS レジスタでこの状態をポーリングするか、アプリケーションを開始する前に AC 割込みを待つことができます。

パワーオン リセット (POR)

POR では、AREF は最大 40 µs の起動時間を必要とします。AREF が既に有効になっている場合は、HPPASS_INFRA_STARTUP_CFGx レジスタ設定の COUNT_VAL ビット フィールドを変更することで、起動時間を最適化できます。

AREF が Ready になった後、CSG_CH_EN と SAR_EN が有効になります。10 µs 後、CSG_PWR_EN_SLICE が有効になります。その後、clk_csg の 32 サイクル後に CSG_READY (CSG_OUT データが有効) が設定されます。
CSG_PWR_EN_SLICE から CSG_READY までのタイミング遅延は clk_csg 周波数に依存します。次の図に、スタートアップのフローを示します。

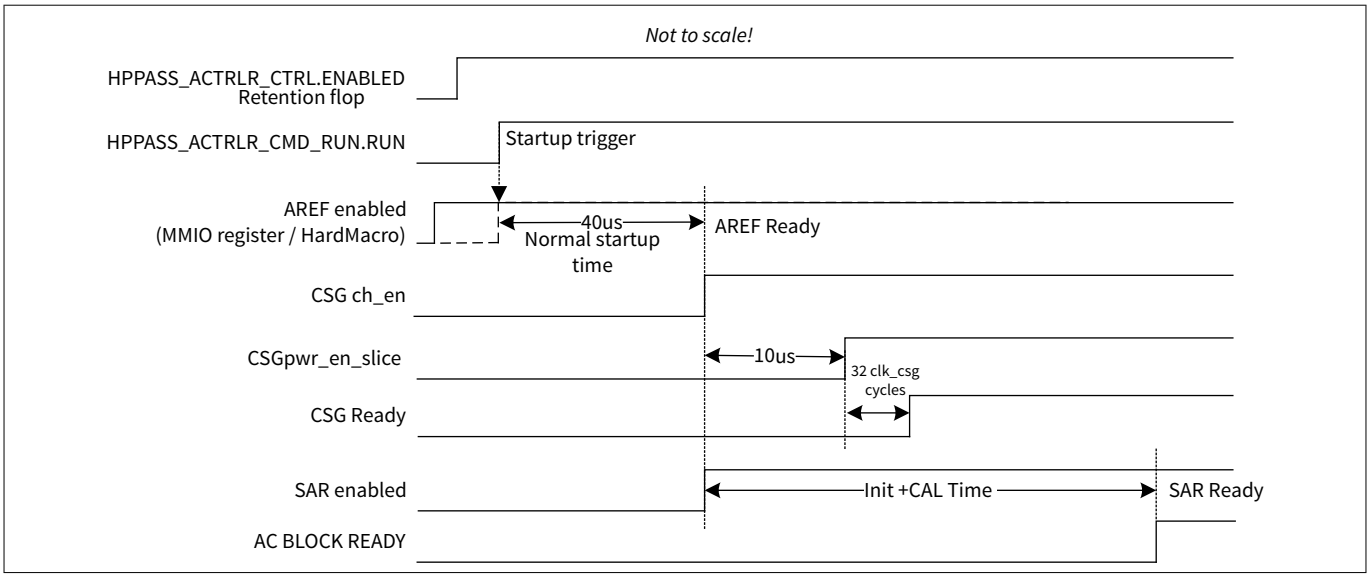


図 282　　POR でのスタートアップ

HPPASS_INFRA_CLOCK_STARTUP_DIV と HPPASS_INFRA_STARTUP_CFG0/1/2/3 レジスタを以下のようにプログラムすることにより、上図のような典型的なタイミング シーケンスを実現できます。
HPPASS_INFRA_STARTUP_CFGx (x = 0～3) は合計 4 セットあり、最大 4 つのタイミング遅延をサポートできます。PSOC™ Control C3 MCU では、以下のように 3 つのレジスタのみが使用されます。

注:　　上記のタイミング シーケンスは非常に重要で、ユーザーが変更してはいけません。

スタートアップ タイマーは、CLK_HF3 が 240 MHz のとき、120 MHz で動作する clk_infra からの分周クロックです。デフォルトでは、HPPASS_INFRA_CLOCK_STARTUP_DIV レジスタの DIV_VAL ビットフィールドは 7 で、スタートアップ タイマーの分解能は約 66.7 ns ですが、このレジスタで変更することもできます。

SAR ADC は工場キャリブレーションとセルフ キャリブレーションに対応しています。工場キャリブレーションはセルフ キャリブレーションより速いので、インフィニオンは工場キャリブレーションの使用を推奨します。詳細については、キャリブレーションの章を参照してください。SAR ADC のセルフ キャリブレーションが有効な時、POR での最長の遅延は SAR ADC のキャリブレーションのためです。長さはセルフ キャリブレーションの種類と SAR 入力クロック周波数に依存します。最大時間については PSOC™ Control C3MCU データシートを確認してください。

以下は、HPPASS 起動シーケンスの推奨プログラムです。

1. 120 MHz HPPASS INFRA クロックからのスタートアップクロック分周分解能を 200 ns に設定します。計算上 DIV_VAL= (200 ns/8.33 ns)-1 = 23 (17_H)

```
HPPASS_INFRA_CLOCK_STARTUP_DIV.DIV_VAL[7:0] = 17H
```

2. AREF_EN から SAR_EN (ビット 8) と CSG_CH_EN (ビット 9) までの 40 µs タイマーが有効になるように HPPASS_INFRA_STARTUP_CFG0 を設定します。計算上 COUNT_VAL= (40 us/200 ns)-1 =199 ($C7_H$)

```
HPPASS_INFRA_STARTUP_CFG0.COUNT_VAL[7:0] = C7H
HPPASS_INFRA_STARTUP_CFG0.SAR_EN = 1'b1
HPPASS_INFRA_STARTUP_CFG0.CSG_CH_EN = 1'b1
```

3. CSG_CH_EN から CSG_PWR_EN_SLICE (ビット 10) までの 10 µs タイマーが有効になるように HPPASS_INFRA_STARTUP_CFG1 を設定します。計算上 COUNT_VAL = (10 us/200 ns)-1 = 49 (31_H)

```
HPPASS_INFRA_STARTUP_CFG1.COUNT_VAL[7:0] = 31H
HPPASS_INFRA_STARTUP_CFG1.CSG_PWR_EN_SLICE = 1'b1
```

4. CSG_PWR_EN_SLICE から CSG_READY (ビット 11) までの 400 ns タイマーを有効にするように HPPASS_INFRA_STARTUP_CFG2 を設定します。clk_csg が 120 MHz で動作していると仮定します。clk_csg の 32 サイクルは 266.67 ナノ秒です。計算上 COUNT_VAL = (266.67 ns/200 ns)-1 = 0

```
HPPASS_INFRA_STARTUP_CFG2.COUNT_VAL[7:0] = 0H
HPPASS_INFRA_STARTUP_CFG2.CSG_READY = 1'b1
```

ディープスリープからのウェイクアップ

ディープスリープ モードからのウェイクアップ時の SAR ADC キャリブレーションの待ち時間を短縮するために、ユーザーはディープスリープに入る前に SAR ADC キャリブレーション値を読み出し、ディープスリープ対応 RAM に保存する必要があります。

下図に示すように、ウェイクアップ時に、ユーザーはこれらの値を SAR ADC キャリブレーション レジスタに書き戻すことができ、それにより SAR キャリブレーションの必要性を回避できます。SAR ADC には初期サンプリングを含むノミナルの初期化時間しかないため、起動時間は大幅に短縮されます。

***注**: インフィニオンのペリフェラル ドライバ ライブラリ (PDL) 関数 `Cy_HPPASS_DeepSleepCallback()` は、キャリブレーション値の保存または復元に必要なすべての動作を行います。詳細については、Peripheral Driver Library API Reference Manual を参照してください。*

HPPASS_INFRA_AREFv2_AREF_CTRL レジスタの ENABLED ビットは保持ビットです。ディープスリープ中もこのビットは有効ですが、AREF を無効にするために絶縁され強制的に"0"になります。ウェイクアップすると、AREF イネーブルは自動的に再び有効になり、40 秒後に AREF は Ready になります。Deep Sleep-RAM/OFF の間、そのレジスタ ビットは無効になります。ウェイクアップ時に AREF を有効にするためには、HPPASS_INFRA_AREFv2_AREF_CTRL レジスタの ENABLED ビットを再度設定する必要があります。

通常、ディープスリープ モードでは AREF は OFF です。AREF の起動時間は POR と同じ 40 **µs** です。AREF がすでに有効になっており、ディープスリープの継続時間がわかっている場合、HPPASS_INFRA_STARTUP_CFG0 レジスタ設定の COUNT_VAL ビットフィールドを変更することにより、起動時間を短縮できます。

HPPASS_ACTRLR_CTRL レジスタの ENABLED ビットは保持フロップです。ウェイクアップ時に AC は自動的に有効になり、AC の実行を指示するためには HPPASS_ACTRLR_CMD_RUN レジスタの RUN ビットのみを設定してください。残りのタイミング シーケンスは、SAR Ready タイミングを除いて POR と同じです。

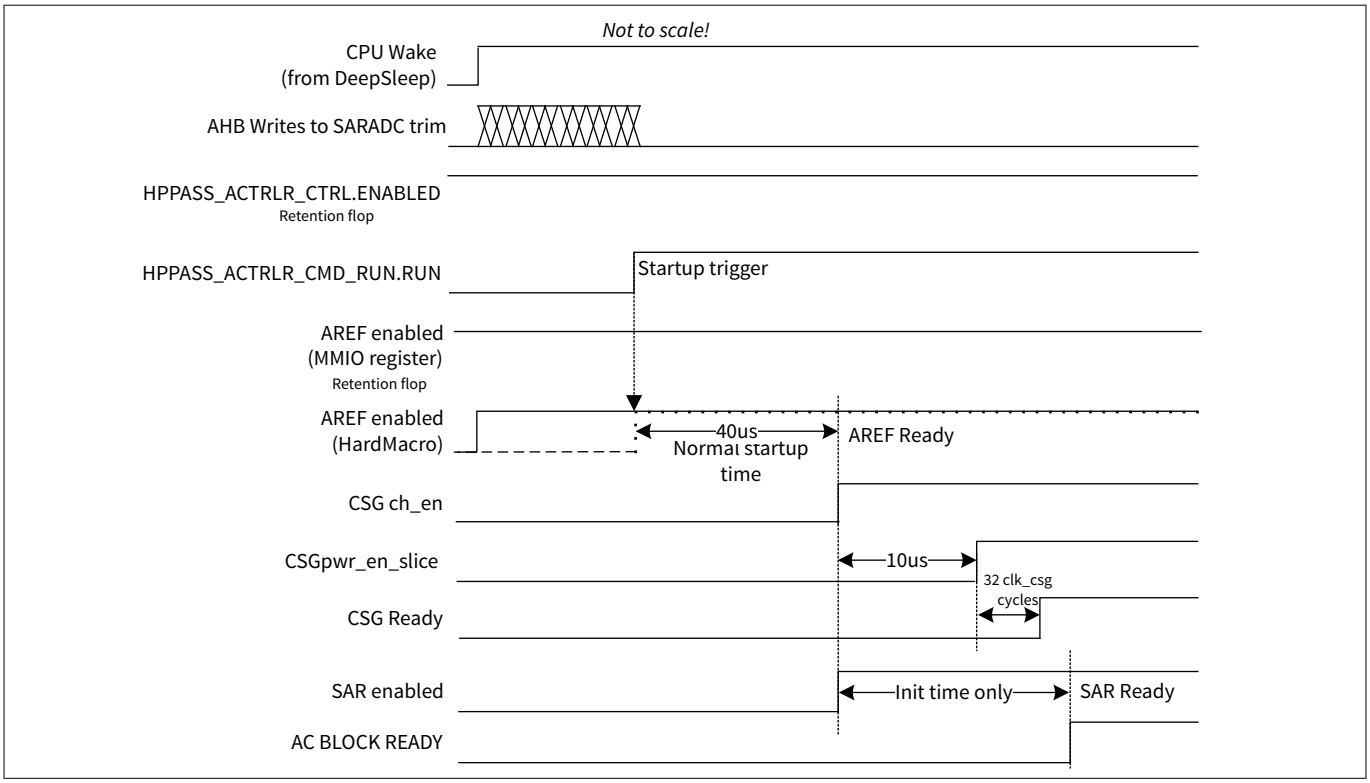


図 283　　ディープスリープ後の起動

27.2.1.2　入力トリガー

トリガー マルチプレクサから 2 組の入力トリガー (MUXED (「A」) と ONE-TO-ONE (「B」)) が接続され、レイテンシと柔軟性のトレードオフを可能にします。MUXED トリガーには複数のソースがありますが高レイテンシです。一方、ONE-TO-ONE は単一のソースから単一の宛先へのため低レイテンシです。HPPASS_INFRA_TR_IN_SEL レジスタの TRx_SEL ビットフィールドを設定し、ハードウェア MUXED ("A") または ONE-TO-ONE ("B") トリガー、ファームウェアパルスまたはレベルトリガーを選択します。

HPPASS 入力トリガーには以下の特性があります (下図参照):

- 内部には 8 つのトリガーがあり、8 つの SAR ADC グループと 5 つの CSG スライスをサポートします。
- ハードウェアトリガー入力はパルスまたはレベル
- 外部トリガーは同期していなければいけません
- ハードウェア トリガーは、TCPWM によって生成された波形からの柔軟な動作をサポートするために、正、負、または両方のエッジで PULSE として生成できます。トリガー モードを選択するためには、HPPASS_INFRA_HW_TR_MODE レジスタの HW_TRx_MODE ビット フィールドを設定します。
- どの入力トリガーでも、ハードウェアトリガーの代わりに、セルフクリアするファームウェア パルス トリガーまたはファームウェア レベル トリガーを選択し、HPPASS_INFRA_FW_TR_PULSE/HPPASS_INFRA_FW_TR_LEVEL レジスタを設定して、ファームウェア パルスまたはレベル トリガーを生成できます。

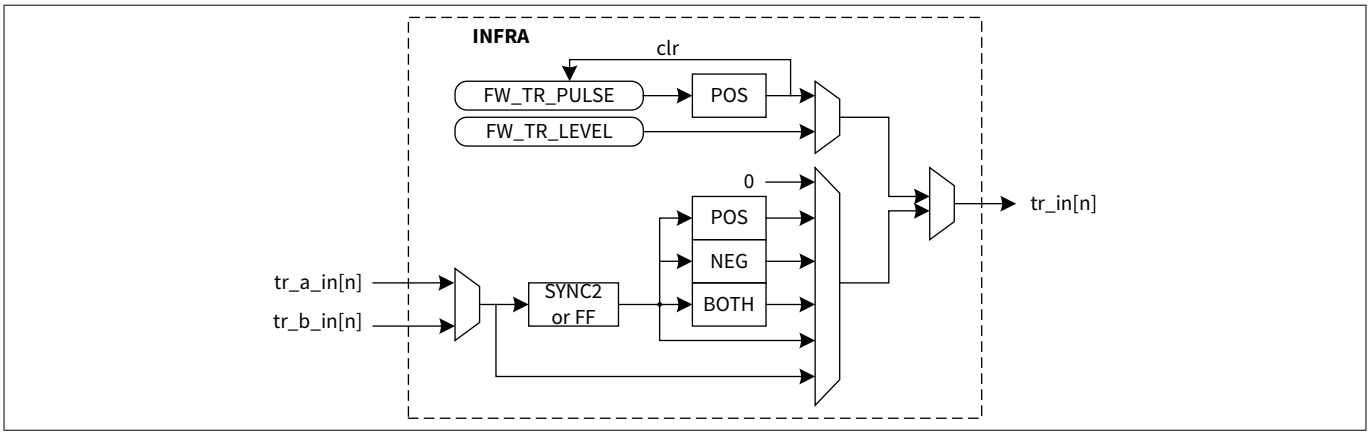


図 284　　トリガー入力処理

入力トリガーのルーティング

HPPASS の重要な特長は、複数のプロセスを同時にトリガーできることです。したがって、8 つの内部トリガーはルーティングされ、すべてのサブブロックで選択可能です。さらに、AC は各サブ ブロックに個別のトリガーを備えており、自律的なシーケンスでプログラムできます。

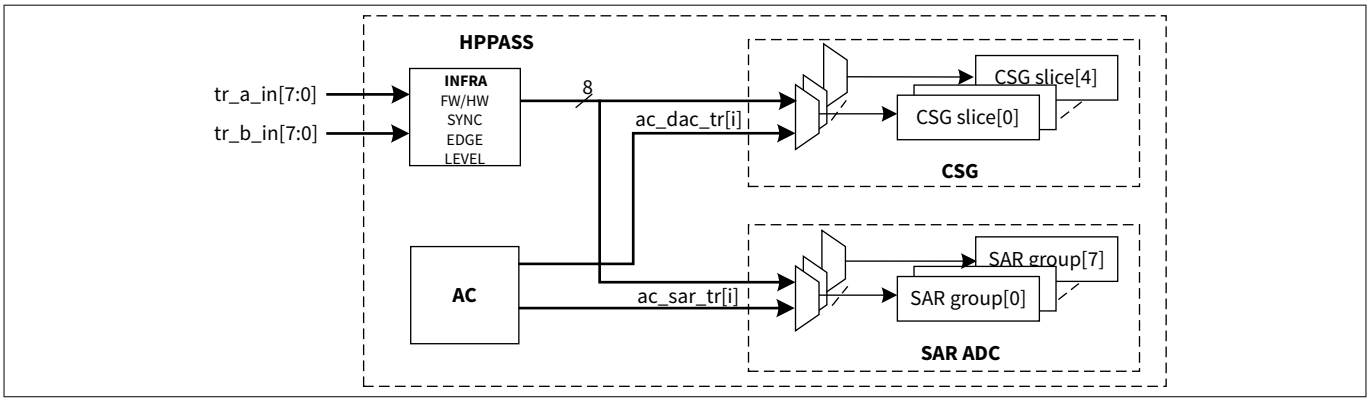


図 285　　入力トリガーのルーティング

27.2.2　SAR ADC

次の図に、SAR ADC ブロック図を示します。

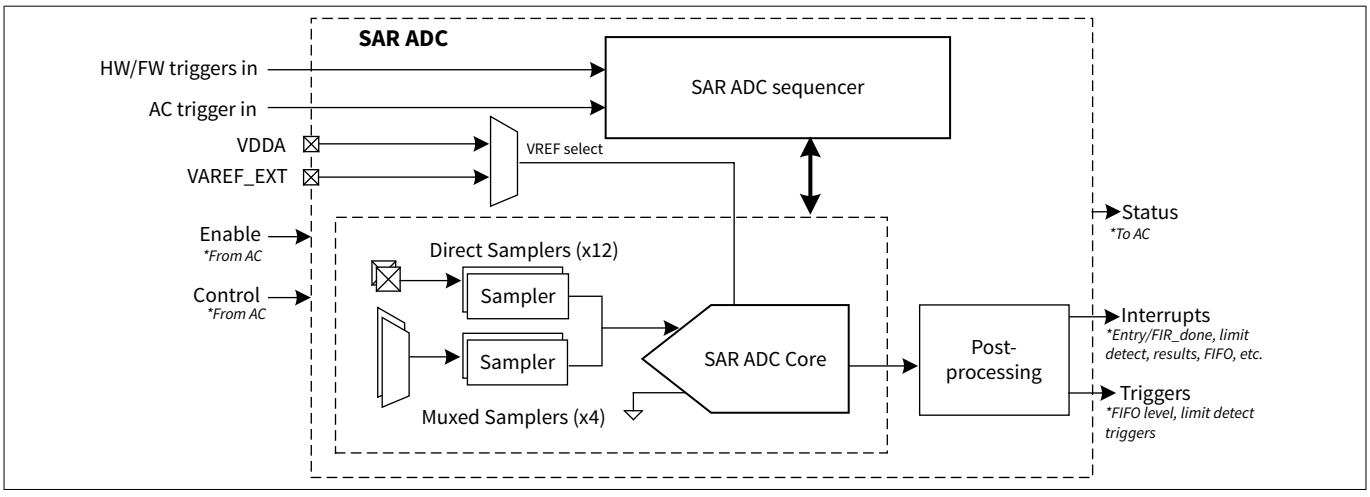


図 286　　SAR ADC ブロック図

SAR ADC は主に SAR シーケンサー、サンプラー、SAR ADC コア、後処理ブロックから構成されます。以下のセクションでは、これらのブロックの機能と操作について説明します。

注: HPPASS_SAR_CFG_CTRL レジスタの VREF_SEL フィールドを設定することにより、SAR ADC の基準電圧として VDDA、または VAREF_EXT ピンの外部基準を選択できます。SAR ADC の性能を保証するために、VAREF_EXT ピンから外部リファレンスを選択します。

注: VDDA 電源電圧が 2.7 V 未満の時、HPPASS_SAR_CFG_CTRL レジスタの LOW_SUPPLY_MODE ビットを 1 に設定すると、SAR ADC の最大サンプル レートが低下します (詳細は、データシートを参照)。

27.2.2.1 チャネル

HPPASS SAR ADC には 12 個のダイレクト サンプラーと 4 個のマルチプレクスサンプラーがあり、下図に示すように、これらはダイレクト入力とマルチプレクサ入力に対応します。

HPPASS_SAR_CFG_SAMP_EN レジスタにより、すべてのダイレクトサンプラー、および、マルチプレクスサンプラーをイネーブルにできます。アプリケーションがサンプラーを使用する前に、サンプラーが有効になっていることを確認してください。すべてのサンプラーは、HPPASS_SAR_CFG_SAMP_GAIN レジスタでゲイン値を設定することにより、1 倍、3 倍、6 倍、12 倍のゲインに対応しています。

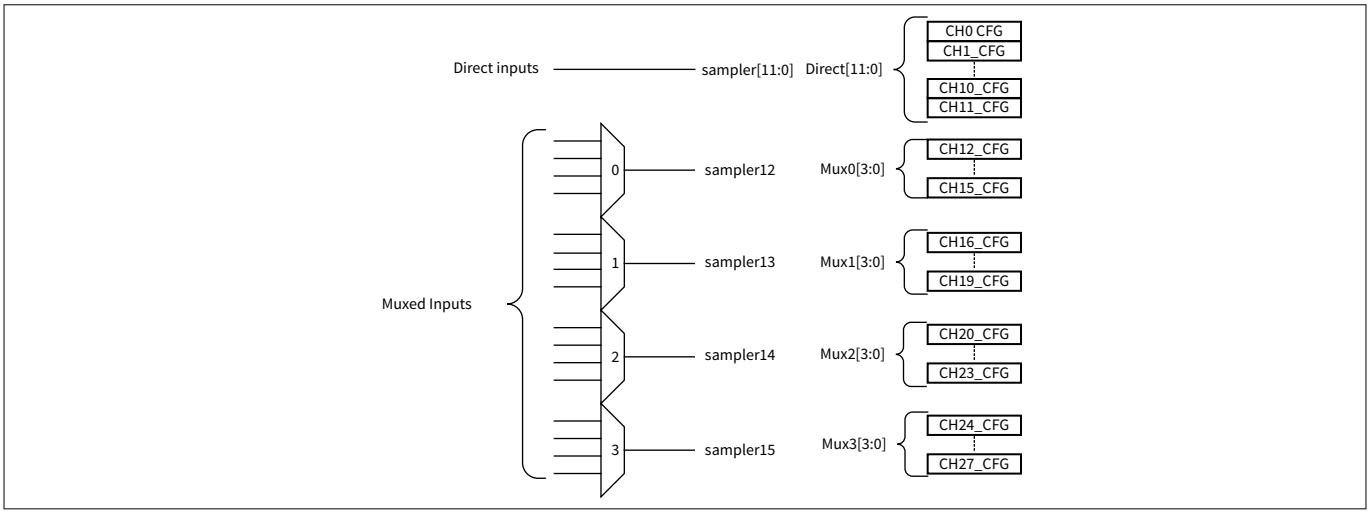


図 287 HPPASS SAR ADC チャネルマッピング

SAR ADC は 28 SAR チャネル構成です。しかし、いくつかのマルチプレクサ入力は予約されているため、PSOC™ Control C3 MCU ではチャネルが不連続になっています。実際のポートとチャネルのマッピングは 表 252 を参照してください。HPPASS の温度センサーは 1 つだけで、MUX0 または MUX1 で SAR ADC に接続できます。詳細は、温度センサーの章を参照してください。

表 252 PSOC™ Control C3 MCU SAR ADC チャネルマッピング

チャネル番号	サンプラー	チャネルタイプ	入力
0	サンプラー 0	ダイレクト	AN_A0
1	サンプラー 1	ダイレクト	AN_A1
2	サンプラー 2	ダイレクト	AN_A2
3	サンプラー 3	ダイレクト	AN_A3
4	サンプラー 4	ダイレクト	AN_A4

(続く)

表 252 (続き) PSOC™ Control C3 MCU SAR ADC チャネルマッピング

チャネル番号	サンプラー	チャネルタイプ	入力
5	サンプラー 5	ダイレクト	AN_A5
6	サンプラー 6	ダイレクト	AN_A6
7	サンプラー 7	ダイレクト	AN_A7
8	サンプラー 8	ダイレクト	AN_B0
9	サンプラー 9	ダイレクト	AN_B1
10	サンプラー 10	ダイレクト	AN_B2
11	サンプラー 11	ダイレクト	AN_B3
12	サンプラー 12 (MUX 0 用)	MUX0 選択 0	AN_B4
13		MUX0 選択 1	CSG DAC OUT
14		MUX0 選択 2	予約済み
15		MUX0 選択 3	温度センサー
16	サンプラー 13 (MUX 1 用)	MUX1 選択 0	AN_B5
17		MUX1 選択 1	予約済み
18		MUX1 選択 2	予約済み
19		MUX1 選択 3	温度センサー
20	サンプラー 14 (MUX 2 用)	MUX2 選択 0	AN_B6
21		MUX2 選択 1	GPIO_01_AIO
22		MUX2 選択 2	予約済み
23		MUX2 選択 3	予約済み
24	サンプラー 15 (MUX 3 用)	MUX3 選択 0	AN_B7
25		MUX3 選択 1	GPIO_00_AIO
26		MUX3 選択 2	予約済み
27		MUX3 選択 3	予約済み

SAR ADC チャネルは以下のチャネル後処理機能をサポートします。

- 疑似差動変換: 対の入力を同時にサンプリングし、ハードウェアで結果を変換して減算し、1 つの結果をユーザーに提示します。
- 符号付き/符号なしフォーマット: 擬似微分結果と FIR 処理に有用
- 限界検出: 利用可能な 8 つの限界検出リソースから 1 つを選択
- FIFO 書き込みオプション: 利用可能な 4 つの FIFO から 1 つを選択
- 平均化 N 個の結果の蓄積 (N は {2,4,8,16} のいずれか)。

チャネルごとではなく、リソースごとの後処理機能があります。

- 2 つの 16 タップ FIR ブロック: 28 チャネルから 1 チャネルを入力として選択

機能の詳細については 後処理の章を参照してください。

27.2.2.2 シーケンサー

下図に SAR ADC シーケンサーを示します。

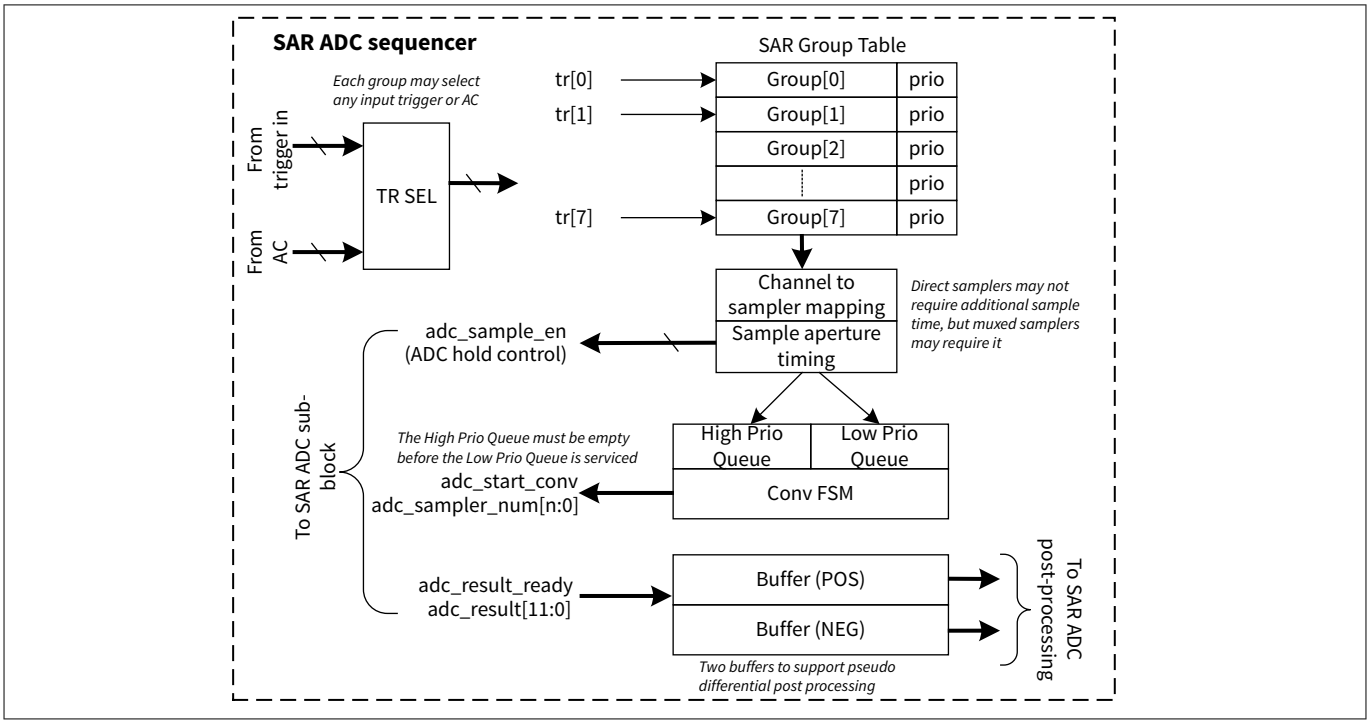


図 288 SAR ADC シーケンサー

SAR ADC シーケンサーの基本動作は SAR グループ テーブルによって制御され、ユーザー アプリケーションに応じて 8 つのグループがプログラムされます。

テーブルの各エントリーは、その行がトリガーされたときに同時にサンプリングされるチャネルのグループを定義し、各グループはサンプル時間 (デフォルトは 0) やプライオリティなどの他のプロパティを共有します。

各グループは SAR ADC の全 16 のサンプラーの制御と変換をサポートします。

SAR ADC には 3 つのサンプル時間があります。各グループは、無効化 (0 サンプル時間) または 3 つの利用可能なサンプル時間の中から 1 つを選択できます。サンプル時間は、HPPASS_SAR_CFG_SAMPLE_TIMEx (x = 0〜2) レジスタで設定可能です。サンプル時間の詳細については、以下の項目を確認してください。

SAR ADC シーケンサーの典型的なフローは以下のとおりです。

1. AC または外部ハードウェアトリガーがアサートされます
2. サンプル時間
   a. サンプル時間プログラミング = 0 または無効: そのグループ内のすべてのサンプラーは、SAMPLE 状態から HOLD 状態に即座に遷移します。
   b. サンプル時間プログラミング > 0: サンプラーはサンプル時間プログラミングでクロック数のみ SAMPLE 状態になり、その後 HOLD 状態に移行します。
3. プログラムされたサンプル時間 (または 0 サンプル時間) の後、トリガーされたグループは優先順位プログラムに基づいて変換キューに割り当てられます。キューの方式は FIFO です。しかし、2 つ以上のグループが同じサイクルでトリガーされ、同じ優先順位を持つ場合、低い番号のグループは、与えられたキューで高い番号のグループよりも優先されます。
4. SAR ADC 変換は、キューイングされたグループがあるときは実行中です。高優先度キューにあるグループのサンプラーはすべて、低優先度キューにあるグループのサンプラーより先に変換されます。

5. SAR ADC から結果が生成されると、そのチャネルの選択された後処理が実行されます。
6. 結果は、結果レジスタまたは FIFO のいずれかに保存されます。結果レジスタの場合、指定されたグループのすべての結果が完了すると、グループ完了割込み (ENTRY_DONE) が CPU に発生します。FIFO の場合、結果の数はプログラム可能です (FIFO_LEVEL)。詳細は、結果ハンドリングを参照してください。

SAR グループ テーブルは、同時にサンプリングされる入力のセットを定義します。下図は SAR ADC シーケンサーのグループ テーブル構造です。HPPASS_SAR_SEQ_ENTRYx (x = 0～7) レジスタの詳細については、レジスタ テクニカル リファレンス マニュアルを参照してください。

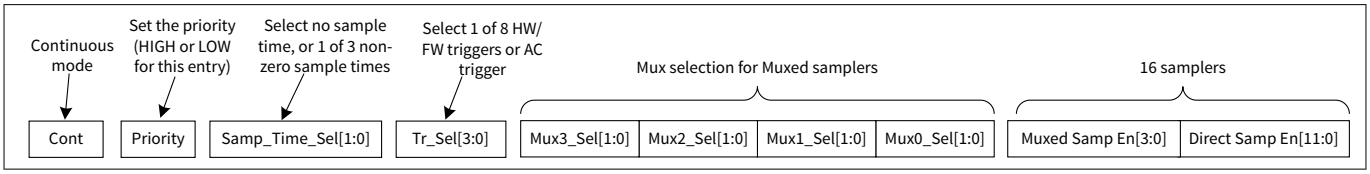


図 289　　SAR ADC グループ テーブル構造

- **Continuous**: このビットを設定すると、Continuous モードが実装されます。グループは、関連するすべてのサンプラーの変換が完了すると、自動的に再びトリガーされます。これはかなりの ADC 帯域幅を使用することがあることに注意してください。これを緩和するためには、トリガー周波数を調整するためにサンプル時間を導入することです。
- **優先順位**: このフィールドは、グループを高優先度キューに置くか低優先度キューに置くかを示します。優先順位の高いグループは、常に優先順位の低いグループより先に変換されます。優先度の高いグループは変換され、ADC の仕様を満たすことが保証されます。しかし、優先順位の低いグループは、HOLD 状態が長すぎて ADC の仕様を満たさないことがあります。したがって、HPPASS_SAR_CFG_ENTRY_HOLD_VIOLATION ステータス レジスタをチェックして、低優先度グループが期限切れ前に変換されたことを確認してください。
- **サンプル時間選択**: このフィールドでは、このグループの追加サンプル時間を選択できます。デフォルトでは、このグループがトリガーされた後、追加のサンプル時間はありません (サンプラーはトリガーイベントで即座に HOLD 状態に遷移します)。または、利用可能な 3 つのサンプル時間から 1 つを選択します。HPPASS_SAR_CFG_SAMPLE_TIMEx レジスタはサンプル時間を設定します。
- **TR の選択**: これらのビットにより、各グループは、このグループをトリガーする外部ハードウェア、ファームウェア、または AC を独立して選択できます。
- **マルチプレクサの選択**: マルチプレクスサンプラーが有効な場合、これらのフィールドはマルチプレクスサンプルインプットの AROUTE インプットの設定を示します。グループがトリガーされると、AROUTE mux は mux 選択値に変更されます。この場合、HOLD 状態に移行し変換が始まる前に、入力が安定するまでサンプリング時間が必要です。
- **サンプラー有効**: 16 サンプル (12 ダイレクト、4 ミックス)、1 に設定するとアクティブなサンプラーを起動し、このグループがトリガーされると変換されます。

***注**:　　HPPASS_SAR_CFG_SAR_STATUS.BUSY ビットは SAR のビジー状態を示し、このビットは SAR ADC グループの連続モードが有効な時は無効です。*

利用可能なサンプラー (16) の 1 つ以上、またはすべてを、所定のグループでの同期サンプリングのために有効にできます。

ダイレクトサンプラーもマルチプレクササンプラーも、指示されたビットで有効になります。しかし、マルチプレクササンプラーが有効な場合、マルチプレクサのルーティングは HPPASS_SAR_SEQ_ENTRY レジスタの MUXx_SELbit フィールドで制御されます。マルチプレクサのルーティング制御の項を参照してください。

チャネル出力の結果は、ダイレクトサンプラー (0～11) から始まり、MUX0 (12～15)、MUX1 (16～19)、MUX2 (20～23)、MUX3 (24～27) の順に、関連する RESULT レジスタに格納されます。

27.2.2.3 マルチプレクサのルーティング制御

AROUTE MUX は、SAR ADC シーケンサー、ファームウェア、または自律型コントローラー (AC) によって制御できます。各 AROUTE MUX の制御モードを選択するために HPPASS_SAR_CFG_AROUTE_CTRL_MODE レジスタの MUXx_CTRL フィールドを設定し、このレジスタの AROUTE_ENABLED ビットによって AROUTE をグローバルに有効化します。デフォルトでは、HPPASS_SAR_SEQ_ENTRYx レジスタまたは HPPASS_SAR_ENTRY_MUX_SEL_ALIAS レジスタの SAR グループ テーブル MUX0/1/2/3_SEL フィールドの値を使用して、SAR ADC シーケンサーによって制御されます。

注: ファームウェアと AC の設定は上級者向けです。これらの設定は、どの SAR ADC グループ設定よりも優先されます。したがって、あるグループで使用されるマルチプレクサの AROUTE_CTRL_MODE 設定が混在しないように、グループのプログラミングに注意してください。また、ファームウェアモードと AC モードのいずれかを使用する特定の MUX については、1 つの SAR ADC グループのみがその MUX の使用法を指定する必要があります。

AROUTE MUX 制御モードがファームウェアの場合、HPPASS_SAR_CFG_AROUTE_FW_CTRL および HPPASS_SAR_CFG_AROUTE_FW_CTRL_CLR レジスタを使用して MUX 設定をセットおよびクリアします。

AROUTE MUX 制御モードが AC の場合、AC 状態 HPPASS_ACTRLR_TTCFGx_TT_CFG3 (x = 0～15) レジスタの SAR_AROUTE_TR と SAR_AROUTE_SEL フィールドを使用して AROUTE MUX 設定を制御します。例 5 - AC を使って AROUTE を制御を参照してください。

SAR ADC シーケンサーによる AROUTE MUX 制御について説明します。

SAR シーケンサーによる AROUTE MUX の制御

各グループには、各マルチプレクスサンプラーのルーティング仕様が含まれます。そのグループ内でいずれかのマルチプレクスサンプラーが使用されている場合、対応する SAR_SEQ_ENTRY.MUX (i)_SEL ビットフィールドもプログラムする必要があります。

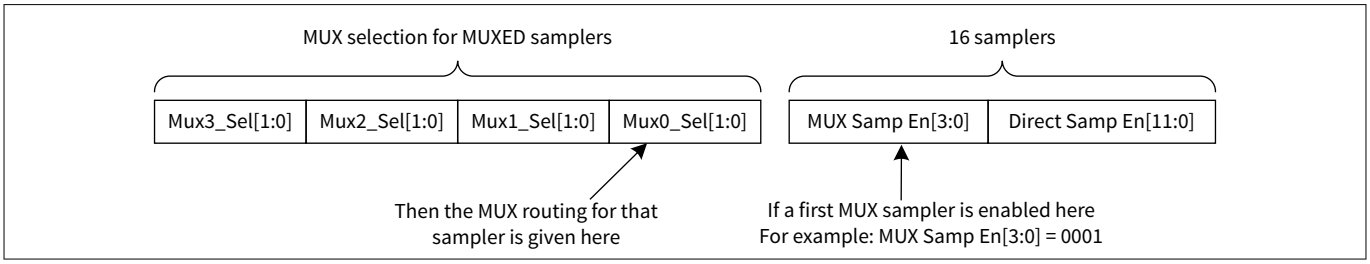


図 290 マルチプレクサのルーティング選択

指定されたグループの MUX 設定は常にトリガーイベントで設定されるため、この設定が変更された場合、そのグループには十分なサンプル時間のプログラミングが含まれます。HPPASS_SAR_SEQ_ENTRYx.SAMP_TIME_SEL ビットでグループ設定のサンプル時間を選択し、HPPASS_SAR_CFG_SAMPLE_TIME レジスタでサンプル時間値を設定します。サンプル時間が終了すると、マルチプレクスサンプラーはホールドに移行し、変換のためにキューに入れられます。

注: ダイレクトサンプラーは、任意のグループ内のマルチプレクスサンプラーでも有効にできますが、すべてのサンプラーがそのグループ内で同じサンプル時間プログラミングを共有するため、性能オーバーヘッドになることがあります。

注: 最初の MUX セレクトが変わらないのであれば、マルチプレクスサンプラーは基本的にダイレクトサンプラーと同等で、サンプル時間は最適化/短縮/削除される場合があります。

ユーザーが異なる MUX 入力を変換したいと仮定すると、マルチプレクサのルーティング制御には 2 つの使用モードがあります。

1. **ハードウェア モード**

このモードでは、異なるルーティングを選択した同じマルチプレクスサンプラーに対して、異なるグループをプログラムでき、これらのグループに対して異なるトリガーを使用できます。

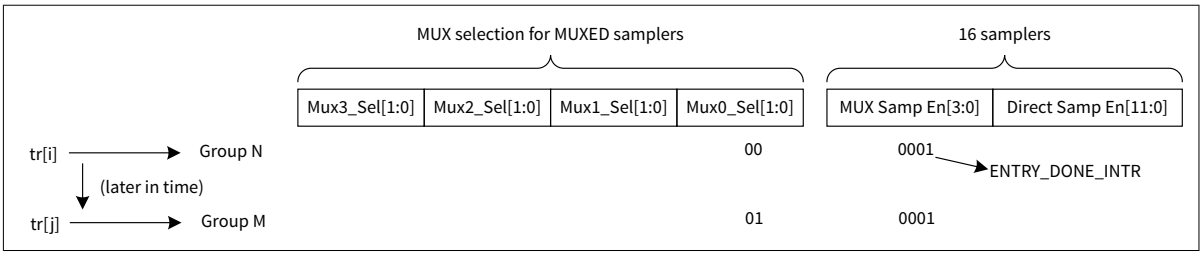


図 291　　ハードウェア モードのマルチプレクサのルーティング選択例

注:　上記の例では、同じマルチプレクスサンプラーを使っています。グループ N がサンプリングまたはコンバートを行っている間、グループ M がトリガーされた場合、結果は不定となります。したがって、グループ M のトリガーを引く前に、グループ N の変換が完了するのを待ちます。

この方法はハードウェア操作に組み込まれていますが、複数のグループを必要とし、グループ トリガーのアサートでルーティングが変更されるため、サンプル時間が常に必要となります。

2.　ファームウェアモード

このモードでは、ファームウェアは同じグループ内の MUX_SEL フィールドを更新できます。このため、グループのトリガーに関して、いくつかの同期が必要となります。例えば、前のトリガーからのグループ完了割込み (ENTRY_DONE) で、ファームウェアは、後続のトリガーに使用するために、MUX_SEL に新しい値を書き込めます。

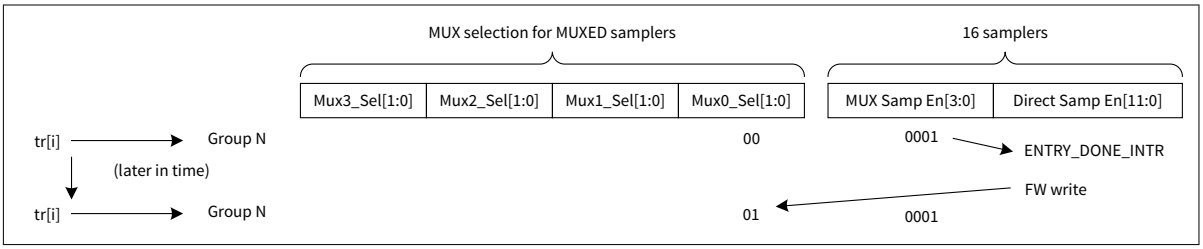


図 292　　ファームウェア モードのマルチプレクサのルーティング選択例

この方法は、異なる MUX_SEL 値に対して同じグループを使用しますが、追加のファームウェア調整を必要とします (例えば、MUX 設定を変更して次の変換をトリガーする前に、前の結果が利用可能になるまで待ちます)。

MUX セレクトエイリアスレジスタ SAR_ENTRY_MUX_SEL_ALIASx があり、グループ内の MUX_SEL フィールドを変更するだけなので、リード モディファイ ライトは必要ありません。この方法を用いると、トリガーの頻度によっては、トリガーの前にルーティングを変更できるため、追加のサンプル時間を必要としない場合があります。

27.2.2.4　トリガー、サンプリング、変換の優先順位の図解

各 SAR ADC グループには 1 つのハードウェア トリガーがあり、各 SAR ADC グループはこれら 8 つのトリガー入力のいずれかを選択できます。

以下の図に、1 つのトリガーが、異なるプライオリティを持つ 2 つのグループにルーティングされる単純なケースを示します。予想どおり、すべてのサンプラーが同時に HOLD 状態に移行し (サンプル時間が 0 と仮定)、優先順位の高いサンプラーが最初に変換され、次に優先順位の低いサンプラーが変換されます。

指定されたサンプラーで変換が完了するとサンプラーは SAMPLE 状態に戻り、結果が完了すると (後処理を考慮すると)、指定されたグループの関連する ENTRY_DONE 割込みが発生します。

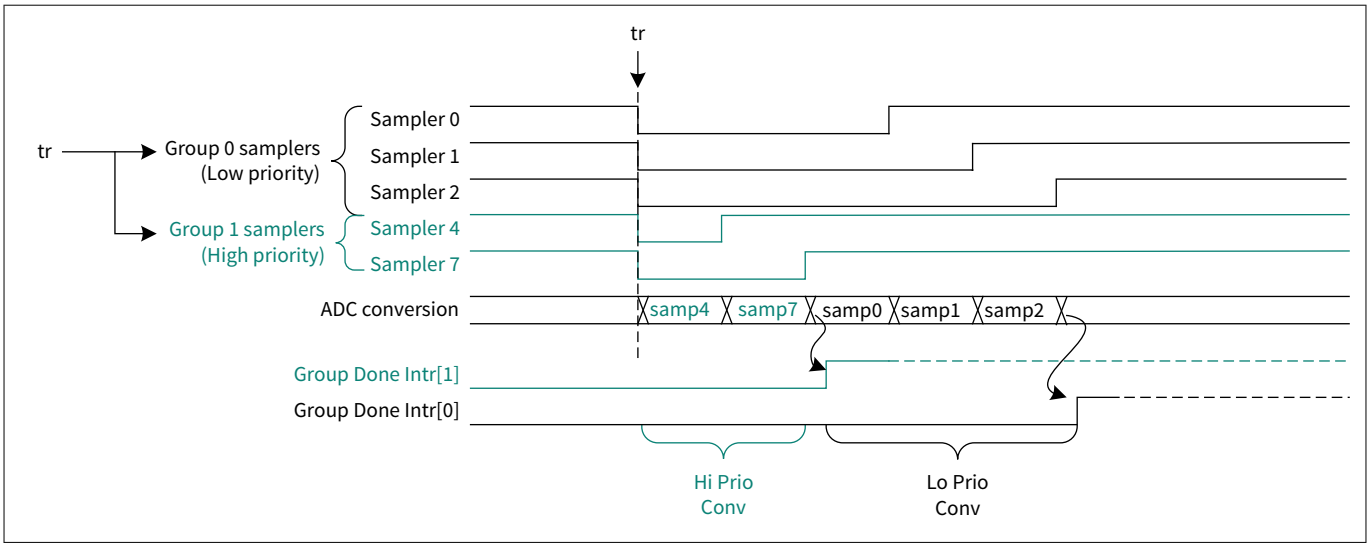


図 293　　シンプルなトリガー シナリオ

ADC サンプラーは常にサンプリング、またはホールドしているので、追加のサンプル時間プログラムは必ずしも必要ではありません。

例えば、ダイレクト チャネルの場合、サンプラーは変換していないときは常に入力をサンプリングしているので、追加のサンプル時間プログラムはほとんど必要ありません。

マルチチャネルでは、トリガーのタイミングとシーケンスに依存します。例えば、MUX セレクトがトリガーで変化するような MUX されたチャネルをユーザーが循環させる場合、サンプリングが必要となります。しかし、MUX セレクトが変更されることなく、ユーザーが MUX されたチャネルを循環している場合、追加のサンプリングは必要ない場合があります。

次の図は、同じケースを示していますが、グループ 0 が 0 より大きいサンプリング時間をプログラムしています。トリガー イベントでは、この場合、サンプリング時間はプログラムされたサイクル数のみ延長されます。

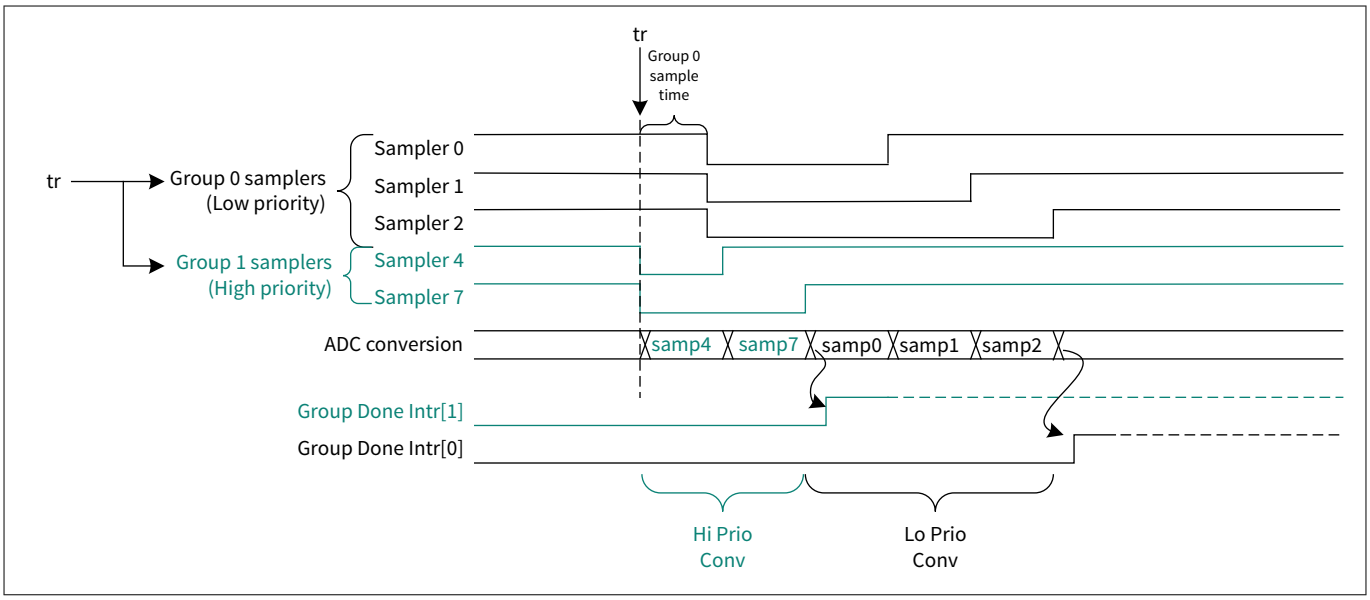


図 294　　単純なトリガーシナリオとサンプリング時間

より現実的なシナリオでは、異なるグループに対して新たなトリガーがアサートされる一方で、以前にトリガーされたグループまたはグループのチャネルが保持または変換されます。

これは、優先順位の高低の概念を示しているだけでなく、一度開始されると、他のグループが開始される前に、そのグループ内のすべてのサンプラーが完了することを例示します。

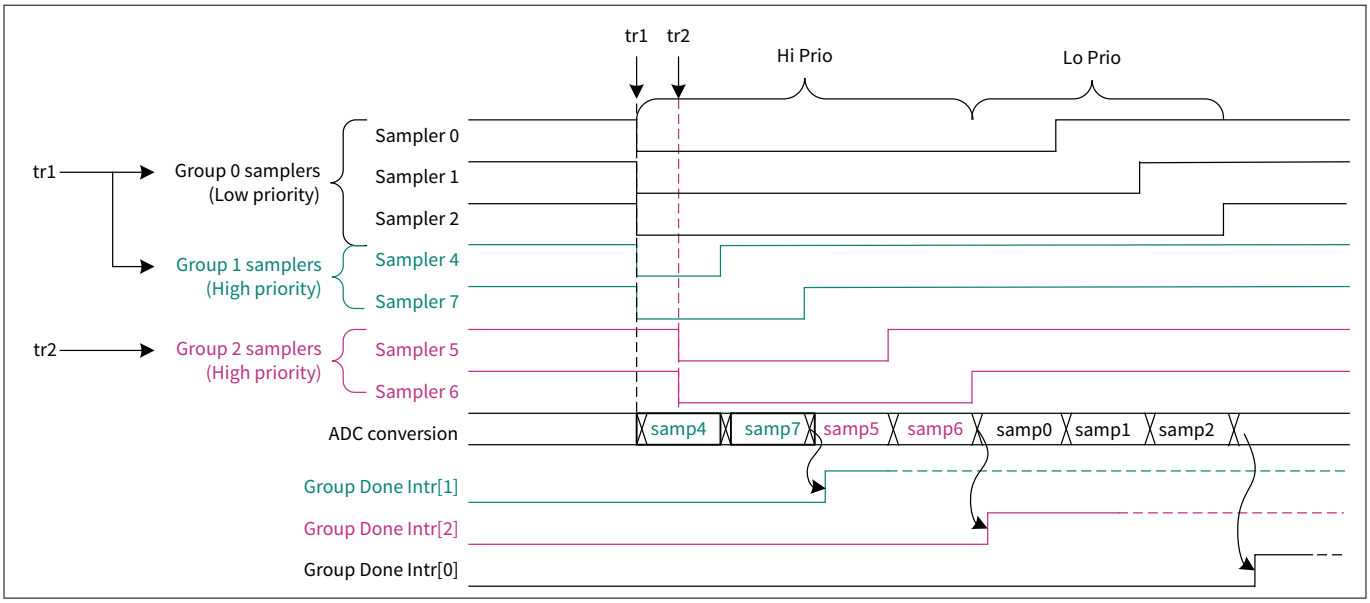


図 295　　複数のトリガーシナリオ

27.2.2.5 エラー状態

SAR ADC は以下のエラーステータスを持ちます。

- **ホールド違反ステータス**

  ユーザーは、優先度の低いチャネルがトリガーされ、サンプラーホールド状態にある時、リークによりサンプルが無効になる前に、SAR ADC がこれらのグループを変換するのに十分な帯域幅を持つように、ユーザー自身の権限でシステムトリガーをスケジュールしてください。ただし、以下のメカニズムに従って、ホールドタイム違反のフラグを立てるためにステータスが提供されます。
  - ホールドのタイミングはグループに基づいています。そのグループには複数のサンプラーが有効になっている場合がありますが、そのグループ内のいずれかのサンプラーがホールドタイムに違反した場合、そのグループに対して違反のフラグが立てられます。
  - SAR_ENTRY_HOLD_CNT レジスタのグローバル HOLD_CNT 設定は、すべてのグループに適用されます。
  - この HOLD_CNT の各ユニットの分解能は 16 CLK_HF サイクルです (1 つの ADC 変換に対応します)。
  - デフォルトの MAX ホールド時間は公称 3 µs です。グループが変換キューにあり、この制限時間以上ホールドしている場合、SAR_ENTRY_HOLD_VIOLATION レジスタの HOLD_VIOL ステータスビットが設定されます。HPPASS_MMIO_HPPASS_INTR_MASK レジスタの ENTRY_HOLD_VIOLATION ビットを設定し、このエラーを CPU に有効にします。
- **トリガーの衝突状態**

  ある SAR ADC グループに対するトリガーが、そのグループの進行中にアサートされた場合、例えば、前のトリガーからすべての結果が完了するまで、新しいトリガーは無視され、SAR_ENTRY_TR_COLLISION レジスタにトリガー衝突ステータスが設定されます。ユーザーは HPPASS_MMIO_HPPASS_INTR_MASK レジスタの ENTRY_TR_COLLISION ビットを設定することで、このエラーを CPU に有効にできます。
- **結果のオーバーフロー状態**

  ENTRY_DONE または FIR_DONE 割込みのクリアを行わなかった場合、次の ENTRY_DONE または FIR_DONE 割込みの前に、前の結果は常に上書きされ、SAR_RESULT_OVERFLOW レジスタにオーバーフローステータスビットが設定されます。ユーザーは、HPPASS_MMIO_HPPASS_INTR_MASK レジスタの RESULT_OVERFLOW ビットを設定することで、このエラーを CPU に通知できるようにすることもできます。
- **FIFO のオーバーフローとアンダーフローの状態**

  フルの FIFO への書き込みまたは空の FIFO からの読み出し時に、FIFO オーバーフローまたはアンダーフロー割込みを発生させます。ユーザーは、HPPASS_MMIO_HPPASS_INTR_MASK レジスタの FIFO_OVERFLOW ビットと FIFO_UNDERFLOW ビットを設定し、FIFO エラーを CPU に通知できるようにできます。

27.2.2.6 後処理

SAR ADC の後処理には、以下の特長があります。

- 疑似差動ハードウェア減算
- 符号付きフォーマット
- 平均化 (結果蓄積、シフト機能なし)
- FIR 処理
- 限界検出

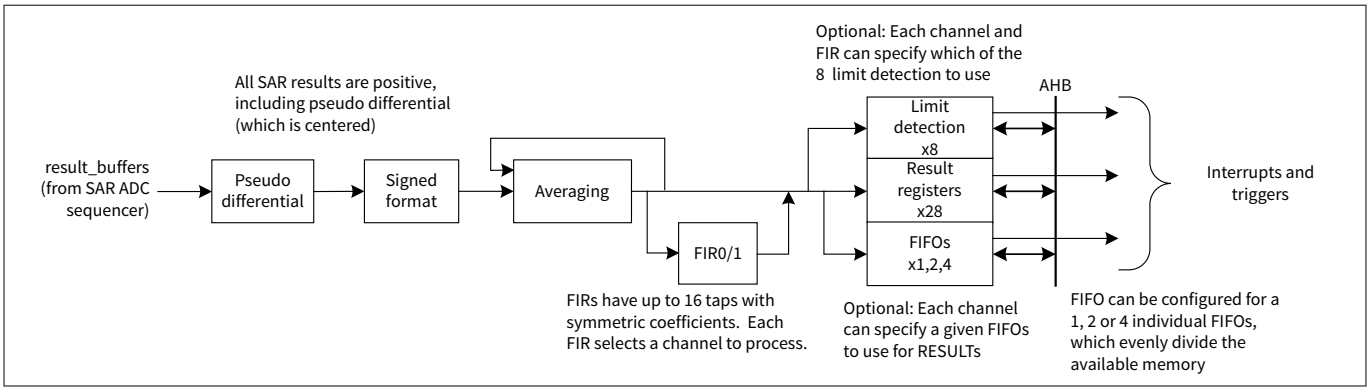


図 296　　SAR ADC 後処理データフロー

疑似差動

与えられたチャネル設定に基づき、2 つのサンプラーを差動入力として指定でき、同時にサンプリングされ、ハードウェアで変換、減算され、1 つのチャネル結果としてユーザーに表示されます。
HPPASS_SAR_CFG_CHAN_CFGx.DIFFERENTIAL_EN ビットは擬似差動動作を有効にするために使用されます (HPPASS_SAR_CFG_CHAN_CFG レジスタの詳細については、『レジスタ技術リファレンス マニュアル』を参照)。

- ダイレクトサンプラー: 偶数/奇数サンプラーペアのみサポート (偶数サンプラーが正入力、奇数サンプラーが負入力とみなされる)、そのため疑似差動設定のみ指定可能で、偶数ダイレクトチャネルで制御されます。奇数ダイレクトチャネルの疑似差動設定は無視されます
- マルチプレクスサンプラー: 偶数/奇数マルチプレクササンプラーペアのみがサポートされているため、擬似差動コンフィギュレーションは偶数 MUX0 と MUX2 に関連するチャネルにのみ指定できます。MUX1 と MUX3 に関連するチャネルの疑似差動設定は無視されます

***注**:　　あるチャネルで擬似差動モードが有効になっている場合、擬似差動動作に関連するサンプラーは、同じゲインになるように HPPASS_SAR_CFG_SAMP_GAIN レジスタで手動設定する必要があります。例えば、チャネル 0 の差動動作が設定されている場合、関連する差動サンプラー 0 と 1 は同じゲインに設定してください。*

擬似差分減算では結果の範囲が 2 倍になるため (最大は 0xFFF - 0x000、最小は 0x000 - 0xFFF)、減算結果は 13 ビットとなります。12 ビットに戻すには、まず 0x1000 を加えて 0x0000 を中心にし、次に 2 で割ってください (右シフト)。この結果は、差 0 が 0x800 で表される、中央に配置された擬似差分値です。
HPPASS_SAR_CFG_CHAN_CFGx.SIGNED を追加オプションとして使用することで、-2048 (0x800) から 2047 (0x7FF) までの符号付きフォーマットに変換できます。

***注**:　　擬似差動モードが特定のチャネルで有効になっている場合、そのチャネルの平均化モードは無効に設定されていなければいけません。*

符号なし/符号付きフォーマット

定義により、SAR ADC からのすべての出力 (擬似差動結果を含む) は符号なしです。符号付き結果への変換は、ユーザーの便宜のためです。結果は単純にシフトされ、中位が 0x000 (0x800 を引く) で表されるように変換され、符号拡張されます SAR_CFG_CHAN_CFGx.SIGNED ビットは、フォーマットを設定するために使用されます。この機能はチャネル コンフィギュレーションとして利用でき、擬似差動結果や FIR 処理の入力に最も役立ちます。

平均化

HPPASS_SAR_CFG_CHAN_CFGx.AVG_SEL ビット設定により、N を{2,4,8,16}のいずれかとして任意のチャネルに対して平均化 (N 個の結果の積算、シフト機能なし、ユーザーはソフトウェアで右シフトする必要がある) を設定できます。
任意のチャネルで平均化が選択されている場合、結果は N トリガーごとにのみ生成され、保存されます。ユーザーは、特定のグループをトリガーする選択肢があります。このグループには平均化されたチャネルが N 回 (また

は定期的に) 含まれます。あるいは、グループ内で HPPASS_SAR_SEQ_ENTRYx.CONT ビットを設定して連続トリガーを行うか、AC を使用して特定のグループを定期的に再トリガーすることも可能です。詳細は AC 例 2 - AC トリガー SAR 平均化グループを参照してください。

有限インパルス応答フィルター (FIR)

FIR のリソースは 2 つあります。各 FIR リソースは HPPASS_SAR_CFG_FIR_CFGx レジスタの CHANNEL_SEL ビットフィールドを設定することで、利用可能なチャネル結果の 1 つを選択して処理できます。FIR は 16 タップをサポートし、係数は対称です。次の図に、HPPASS_SAR_FIRx_FIR_COEFSy レジスタの COEF ビットフィールドで FIR 係数を設定する FIR フィルターの設定を示します。

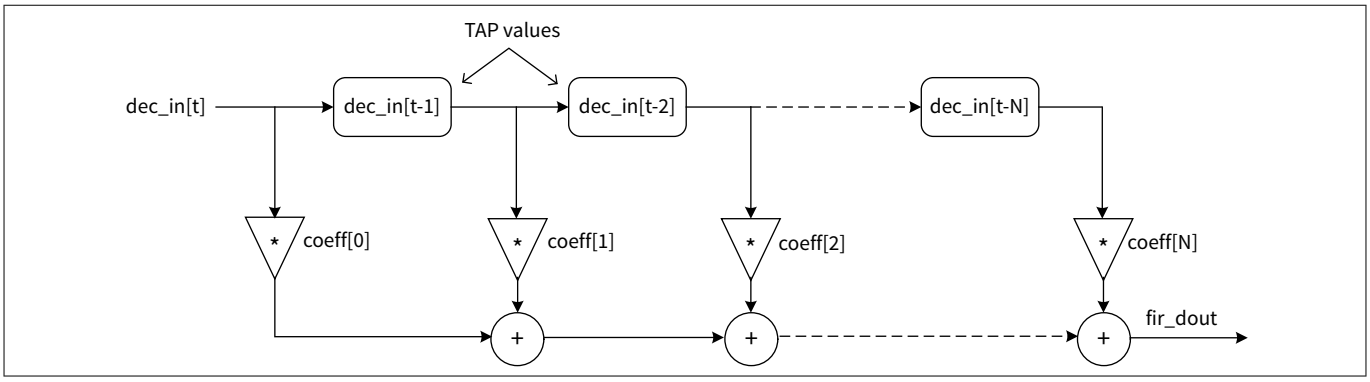


図 297　　FIR フィルターの構造

Dec_in はデシメーターから FIR への入力データを意味し、[t]は時間内のインスタンスを意味します。データが FIR に入ると、まず最初の係数値 (coeff[0]) が乗算され、次に直前のデシメータデータ[t-1]と次に格納された係数値 (coeff[1]) の積 (*) が加算され、これが繰り返されます。これは、最大 16 までプログラム可能な TAP 有効数 (N) に達するまで続きます。FIR 係数値 (16 ビット) は HPPASS_SAR_FIR0/1_FIR_COEFSx (x = 0〜15) レジスタで設定でき、保存されたタップ値 (16 ビット) は HPPASS_SAR_FIR0/1_FIR_TAPSx (x = 0〜15) レジスタで読み出せます。

係数値は-1+1/2^{15}he から 1-1/2^{15} の範囲で、分解能は 1/2^{15} です。係数は 2 の補数表記で、小数点は最上位ビットの右側に固定されます。

```
Examples: -0.25 = 0xE000 and 0.25 = 0x2000
```

HPPASS_SAR_CFG_FIR_CFGx.WAIT_TAP_INIT ビットを設定することで、フィルターが初期化 (16 サンプル) されるまで最初の結果を遅延させ、その後は継続的に結果を得られます。

限界検出

8 つの限界検出リソースがあります。任意のチャネルと FIR は、8 つの限界検出リソースのいずれかを指定できます。チャネルまたは FIR の結果が閾値以上/以下、または 2 つの閾値の内側/外側にある場合、トリガー/割込みが発生します。HPPASS_SAR_CFG_RANGE_HIGHx と HPPASS_SAR_CFG_RANGE_LOWx レジスタは、限界検出リソースの閾値を設定するために使用され、検出オプションには、ABOVE, BELOW, INSIDE, OUTSIDE があり、HPPASS_SAR_CFG_RANGE_CFGx.RANGE_COND レジスタによって設定されます。

複数のチャネルが限界検出リソースを共有する場合、結果のステータスは OR されます。リミット条件は、割込みまたはレベルベースのトリガーとして設定できます。割込みステータスは HPPASS_SAR_CFG_SAR_RANGE_INTR レジスタの RANGE[7:0]ビットで読み出せます。トリガー出力は HPPASS_MIO_TR_LEVEL_OUTx レジスタの SAR_RANGE_TR ビットで有効にできます。

割込みに対して、複数のチャネルまたは FIR がある限界検出リソースを使用して OR された場合、どのチャネルと FIR が限界検出のソースであったかを判断するために HPPASS_SAR_CFG_RANGE_STATUS ステータス レジスタを読み出せます。

27.2.2.7 結果ハンドリング

システム (CPUSS または DMA) が SAR の結果を取得する方法は 2 つあります。

- FIFO
- RESULT レジスタ

FIFO

FIFO ストレージには 32 ワードが使用可能です。特定のアプリケーションでは、この領域を均等に分割する 1、2、または 4 つの FIFO を指定でき、これは HPPASS_MMIO_FIFO_CFG レジスタの SEL[1:0]ビットで指定できます。各チャネルと FIR は、利用可能な FIFO のいずれかを選択できます。例えば、4 つの FIFO が選択され、8 つの ADC チャネルがある場合、HPPASS_SAR_CFG_CHAN_CFGx.FIFO_SEL レジスタを設定することにより、8 つの ADC チャネルは 4 つの FIFO のいずれかを選択できます。変換結果は、HPPASS_SAR_CFG_CTRL.CHAN_ID_EN ビットによって有効にされるオプションの識別タグとともに、所定の FIFO にストリーミングされます。

FIFO ステータスに特化した割込みとトリガーがサポートされており、ユーザー HPPASS_MIO_FIFO_LEVELx.LEVEL レジスタ プログラミングに基づき、レベル割込み (LEVEL、OVERFLOW、UNDERFLOW) とレベルに基づくトリガーが出力として利用可能です。

HPPASS_MMIO_FIFO_CFG.FIFO_TR_EN レジスタを設定することにより、関連 FIFO のトリガー出力を個別に有効にできます。

読み出し専用の HPPASS_MIO_FIFO_USEDx.USED レジスタがあり、FIFO 内の現在のサンプル数を反映します。また、読み出し専用レジスタ HPPASS_MIO_FIFO_RD_DATAx にアクセスすることにより、FIFO データを読み出せます。これは FIFO ごとに 1 つの AHB 読み出しアドレスです。

FIFO 状態は、HPPASS_MMIO_FIFO_CLR.CLR レジスタに「1」を書き込むことでクリアできます。または、HPPASS_ACTRLR_CTRL.ENABLED レジスタをクリアして AC を無効化するか、ディープスリープモード中にクリアすることも可能です。

RESULT レジスタ

各チャネルと 2 つの FIR には、HPPASS_SAR_CFG_CHAN_RESULTx と HPPASS_SAR_CFG_FIR_RESULTx レジスタと呼ばれる RESULT レジスタがあります。各グループの結果は、そのグループのチャネル変換がすべて完了した後に更新されます。そのグループの関連割込み (ENTRY_DONE) は、結果更新と同時にアサートされます。FIR は、FIR 結果を読み出す準備ができたとき (FIR_DONE)、FIR ごとに個別の割込みを発生します。

HPPASS_SAR_CFG_RESULT_OVERFLOW レジスタは、前の値が読み出される前に上書きされたことをユーザーに通知するために提供されます。ENTRY_DONE または FIR_DONE 割込みを受信した場合、HPPASS_SAR_CFG_SAR_RESULT_INTR レジスタの指定されたビットをクリアする必要があります。これらのビットが次の結果更新までにクリアされない場合、SAR_CFG_RESULT_OVERFLOW レジスタの RESULT_OVERFLOW ビットが設定されます。

HPPASS_SAR_CFG_RESULT_UPDATED レジスタがあり、どのチャネルの結果が更新されたかを示すビットがあります。このステータス レジスタはどの SAR ADC 結果レジスタと FIR 結果レジスタが更新されたかを決定するために、複合結果割込みと一緒に使用できます。ユーザーは、クリアしたいビットに'1'を書き込めます。別の方法として、HPPASS_SAR_CFG_SAR_RESULT_INTR レジスタを読み出して、どのグループが完了したかを判断し、それによって間接的に関連するチャネルを知ることができます。

データ転送を最適化するために、SAR 結果エイリアス レジスタ (HPPASS_SAR_CFG_CHAN_RESULT_PACKEDx) が提供され、2 つの 16 ビット結果を 1 つの 32 ビット AHB 読み出しにパックできます (詳細については、レジスタ技術リファレンス マニュアルを参照)。

27.2.2.8 SAR ADC 割込み

次の図に示すように、結果割込みは SAR ADC グループに基づいています。指定された SAR ADC グループのチャネルが完了し、HPPASS_SAR_CFG_CHAN_RESULTx レジスタから読み出す準備ができたとき、割込みが生成されます。同様に、FIR には HPPASS_SAR_CFG_FIR_RESULTx レジスタと FIR_DONE ステータスがあります。詳細は、RESULT レジスタを参照してください。

各グループごとに提供される個別割込み (interrupt_sar_entry_done[7:0]) または FIR 完了 (interrupt_sar_fir_done[1:0])、および複合結果割込み (interrupt_sar_results)。詳細については割込みの章を参照してください。複合割込みが使用される場合、割込みの原因を特定するために HPPASS_SAR_CFG_RESULT_UPDATED または HPPASS_SAR_CFG_SAR_RESULT_INTR レジスタを読まなければいけません。

限界検出割込みは、チャネルまたは FIR の結果が閾値を上回った/下回った場合、または 2 つの閾値の内側/外側にある場合に発生します。各限界検出リソースに設けられた個別割込み (interrupt_sar_range[7:0]) と複合限界検出割込み (interrupt_sar_ranges)。詳細は、限界検出を参照してください。

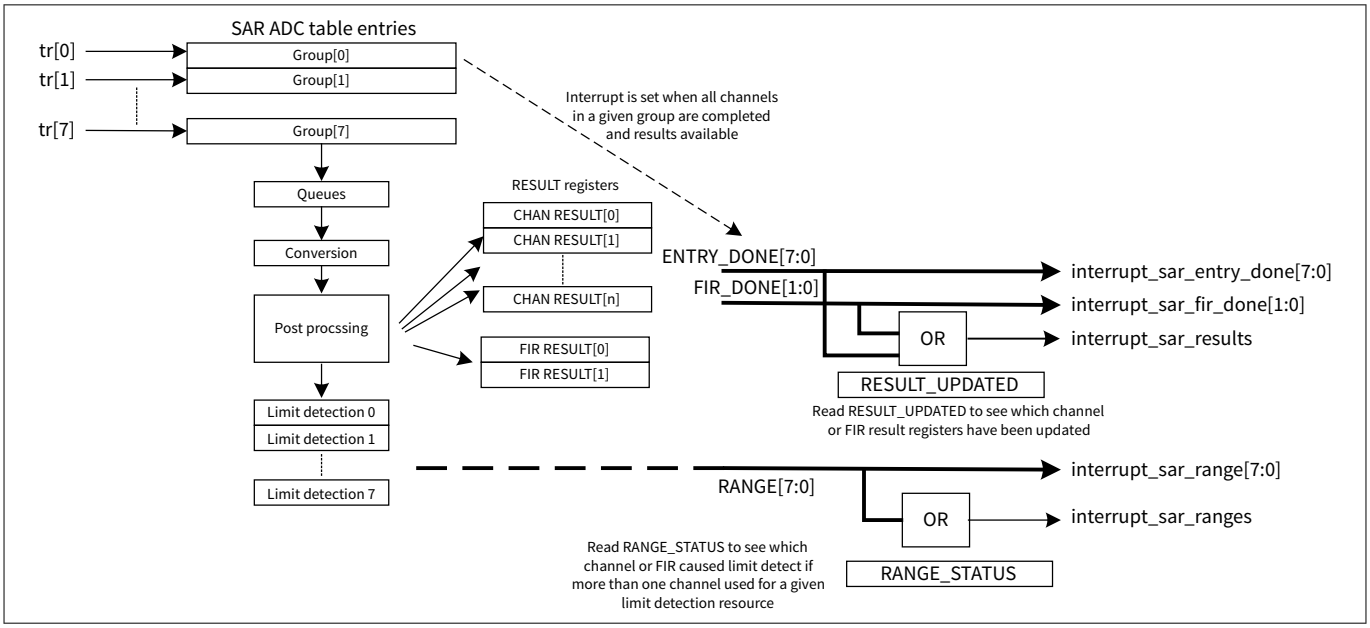


図 298　　SAR ADC 割込み

27.2.2.9　キャリブレーション

HPPASS SAR ADC コアはオフセット、直線性、ゲインのキャリブレーションに対応し、各入力サンプラーのオフセットと直線性、各ゲイン設定のゲインをキャリブレーションします。オフセットはコンパレーター、サンプラー、コモンモード電圧のミスマッチに起因し、直線性は主にプロセスのミスマッチに起因します。ゲイン誤差は、基準バッファのオフセットと基準分周器のミスマッチに起因します。

基準電圧やゲイン設定の変更など特定の事象が発生した場合、オフセットは基準電圧に比例しないため、少なくともオフセット キャリブレーションを実行する必要があります。同じ構成設定で SAR ADC を無効にして再度有効にする場合、再度キャリブレーションを実行する必要はありません。ただし、温度が急激に変化した場合は、オフセット キャリブレーションを行うことを推奨します。基準電圧と電源電圧は、動作範囲内にあり、キャリブレーション中に十分に安定していなければいけません。

SAR ADC はキャリブレーションデータを格納する 4 つのオフセット キャリブレーションレジスタ (HPPASS_SARADC_CALOFFST0～3)、16 の直線性キャリブレーション レジスタ (HPPASS_SARADC_CALLIN0～15)、粗粒度ゲイン キャリブレーション レジスタ (HPPASS_SARADC_CALGAINC)、および微細ゲイン キャリブレーション レジスタ (HPPASS_SARADC_CALGAINF) を備えています。これらのキャリブレーション レジスタの詳細については、『レジスタ テクニカル リファレンス マニュアル』を参照してください。

注: これらのキャリブレーション レジスタの読み出し/書き込みを行うには、HPPASS を ENABLED にする必要があります。HPPASS が無効になると、これらのレジスタはリセットされます。

HPPASS SAR ADC は工場キャリブレーションとセルフ キャリブレーションに対応しています。工場キャリブレーションはセルフ キャリブレーションより速いので、インフィニオンは工場キャリブレーションの使用を推奨します。

PDL は HPPASS 初期化 API の SAR ADC キャリブレーション レジスタへの工場出荷時のキャリブレーション値のコピーを実装します。アプリケーション内で初期化 API を呼び出すことを推奨します。詳細については、Peripheral Driver Library API Reference Manual を参照してください。

工場キャリブレーション

MCU が製造されるとき、SAR ADC は工場で指定された基準電圧でキャリブレーションされ、これらのキャリブレーション値を SFLASH に格納します。ユーザーは HPPASS のスタートアップを高速化するために、セルフ キャリブレーションを有効にすることなく、HPPASS の初期化でこれらの工場キャリブレーション値を SAR ADC キャリブレーション レジスタにコピーできます。オフセット キャリブレーションは温度に依存するため、SFLASH はサンプラーごとに-40、25、125 オフセット キャリブレーション値 (SFLASH_SAR_CALOFFST_0/1/2/3_N40C/125C/25C) を保存しており、ユーザーはインフィニオン ペリフェラル ドライバ ライブラリ (PDL) API を呼び出して、現在の温度と基準電圧に基づいてオフセット キャリブレーション値を自動的に調整できます。SFLASH_SAR_CAL_LIN_TABLE レジスタにはサンプラーごとに 16 の直線性キャリブレーション値が格納され、SFLASH_SAR_CALGAINC および SFLASH_SAR_CALGAINF レジスタにはゲイン キャリブレーション値が格納されます。工場出荷時のキャリブレーション基準電圧は、SFLASH_SAR_CALREFPT レジスタに格納されます。アプリケーションがこれらの工場出荷時のキャリブレーション値を使用する場合は、HPPASS の初期化でこれらの値を SFLASH から SAR ADC のキャリブレーション レジスタにコピーしてください。

セルフ キャリブレーション

SAR ADC はセルフ キャリブレーションもサポートしており、HPPASS_SAR_CFG_CTRL レジスタの STARTUP_CAL_OFFSET_EN, STARTUP_CAL_LINEARITY_EN, STARTUP_CAL_GAIN_EN ビットを設定し、それぞれ SAR ADC 起動時にオフセット、直線性、ゲイン キャリブレーションを有効にします。AC が SAR を有効にすると、このレジスタで選択されたすべてのキャリブレーション ルーチンが実行されます。ユーザーはアプリケーションを開始する前に、SAR ADC のキャリブレーションと初期化が完了する BLOCK_READY を待つ必要があります。デバイスがディープスリープ モードに入ると、SAR ADC キャリブレーション レジスタの値は失われます。ディープスリープ モードからウェイクアップする時の別の SAR ADC キャリブレーションの待ち時間を短縮するために、ユーザーは SAR ADC キャリブレーション レジスタ値を読み出し、ディープスリープ対応 RAM に保存する必要があります。ウェイクアップ時に、ユーザーはこれらの値を SAR ADC キャリブレーション レジスタに書き戻せます。スタートアップのフローの詳細については、起動の章を参照してください。

27.2.3 コンパレーターおよびスロープジェネレーター (CSG)

コンパレーターとスロープ ジェネレーターは、10 ビット 30 Msps DAC 値と選択されたアナログ入力信号を比較したり、2 つのアナログ入力を比較したりするフレキシブルなブロックです。DAC は内部基準としてのみ使用され、チップの外部で測定することはできませんが、ADC は測定できることに注意してください。このブロックの出力は、産業用アプリケーションで TCPWM の動作制御に使用できる 1 ビットのデジタル比較値です。

図 299 に示すように、CSG アナログ サブブロックの周囲には、DAC コード生成用とコンパレーター出力後処理用の 2 つのロジック パーティションが「ラップ」されています。入出力トリガーも、AHB バスを介したユーザーとの対話と同様に、CSG 機能において重要な役割を果たします。以下の図に、利用可能な動作モードとの関連を示します。以下の図は「スライス」とみなされ、PSOC™ Control C3 MCU は 5 つのスライスに対応しています。

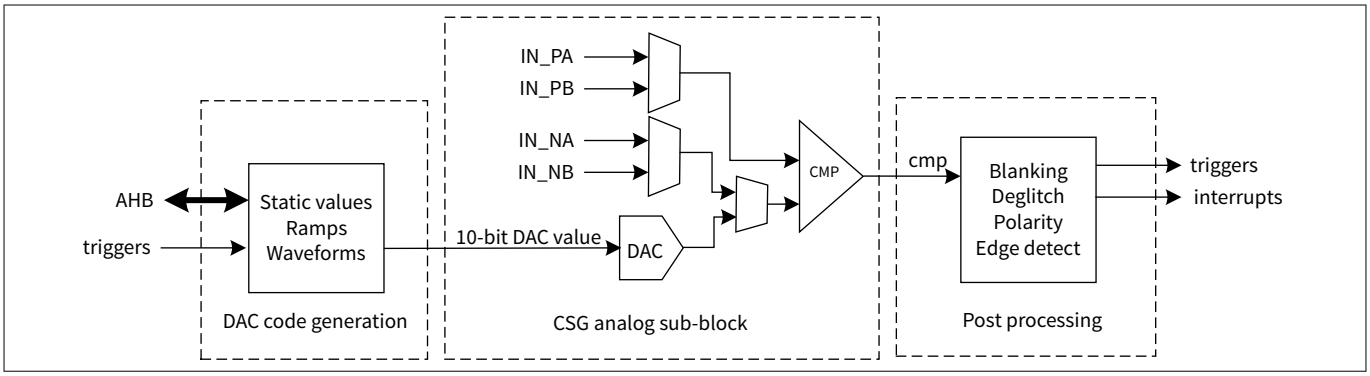


図 299 CSG ブロック図

HPPASS_CSG_SLICEx_CMP_CFG (x = 0～4) レジスタの CMP_POS_SEL フィールドと CMP_NEG_SEL フィールドを設定することにより、各 CSG スライス コンパレーターは異なる正入力と負入力を持ちます (表 253 には各スライスの全入力のコンパレーターがリストされています)。

表 253　　CSG コンパレーター正負入力

スライス	コンパレーター正入力	コンパレーター負入力
スライス 0	AN_A0/AN_B0	DAC0/AN_A1/AN_B1
スライス 1	AN_A1/AN_B1	DAC1/AN_A2/AN_B2
スライス 2	AN_A2/AN_B2	DAC2/AN_A3/AN_B3
スライス 3	AN_A3/AN_B3	DAC3/AN_A4/AN_B4
スライス 4	AN_A4/AN_B4	DAC4

27.2.3.1　10 ビット DAC

DAC コードの生成と制御には、静的および動的なさまざまなモードがあります。次のモードが含まれます。

- DAC コードをファームウェアで直接制御
- DAC コードの Buffered ファームウェア制御
- DAC コードの 2 値 (ヒステリシス) 制御
- 自動 DAC ランプ生成
- DAC 任意波形生成にルックアップテーブル (LUT) をオプションで使用可能

図 300 に、DAC コード生成の主な機能を示します。サポートされる機能は以下のとおりです。

- DAC コードを直接駆動する AHB アクセス可能な DAC_VAL (HPPASS_CSG_SLICEx_DAC_VAL レジスタ)
- 2 つの DAC バッファ、DAC_VAL_A (HPPASS_CSG_SLICEx_DAC_VAL_A レジスタ) と DAC_VAL_B (HPPASS_CSG_SLICEx_DAC_VAL_B レジスタ) は、選択されたモードに応じて定義が異なります (DAC モードのセクションを参照)。
- START、END、DAC_STEP、DAC_PERIOD パラメーターを持つプログラマブル スロープ ジェネレーター。
- ランプまたは波形のスロープを正確に変調する分数機能を備えた周期カウンター。
- START トリガー (設定されたモードを開始します) および UPDATE トリガー (次のコードに進む)。これらはハードウェア、ファームウェア、または AC (UPDATE の場合は PERIOD) の可能性があります。
- 割込み発生用 HW_START、SLOPE_DONE、BUFFER_EMPTY のステータス

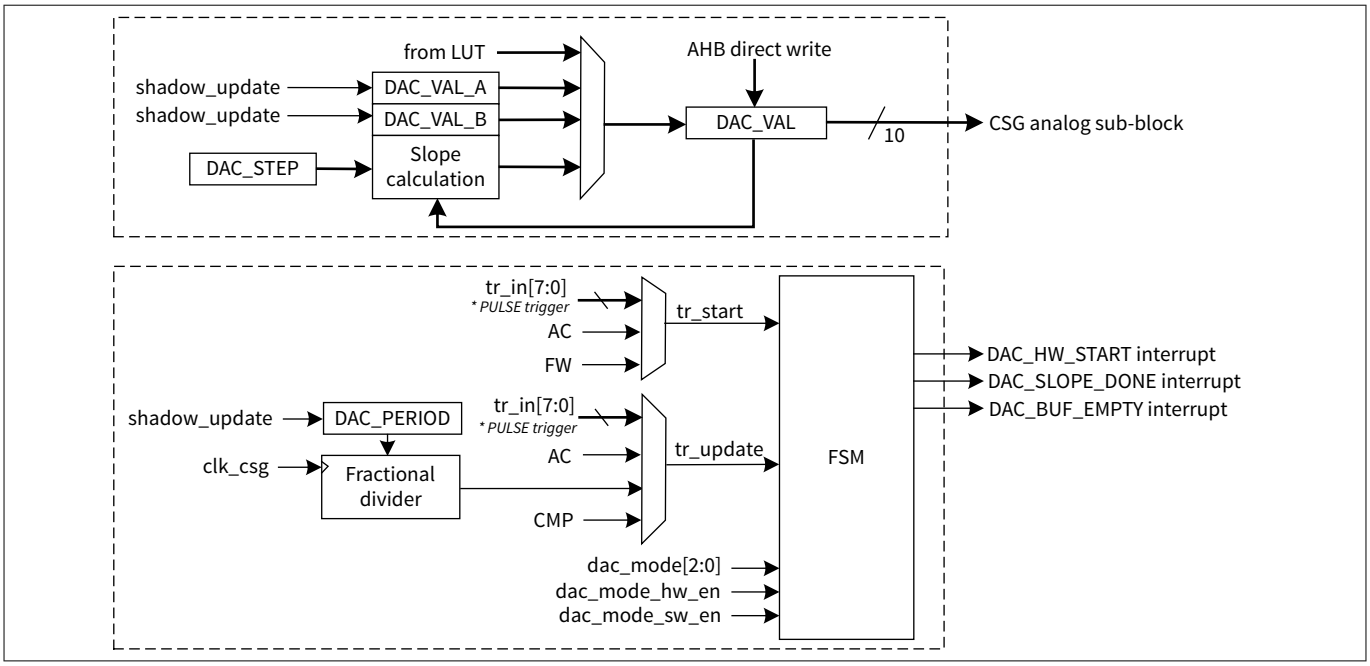


図 300 DAC コード生成

CSG DAC サポート出力の可観測性に加えて、HPPASS_CSG_CSG_CTRL レジスタの VDAC_OUT_SEL ビット フィールドを設定することにより、DAC 出力を SAR ADC に接続し、SAR ADC 変換に使用する CSG スライス DAC 出力を選択できます。CSG DAC 出力は AROUTE MUX0 によって SAR ADC に接続されます (AROUTE MUX 接続の詳細については SAR ADC のチャネルの章を参照)。CSG DAC out が SAR ADC に接続される場合、DAC の観測ブランキングを有効にします (詳細は、DAC の観測ブランキングを参照)。

27.2.3.2 DAC モード

ユーザーは、DAC データのシーケンスを制御するためのさまざまなオプションがあります。以下のモードは各スライスで個別に設定できます。

直接書き込み

ユーザーはいつでも、AHB バスを介して直接 HPPASS_CSG_SLICEx_DAC_VAL レジスタに書き込むことができ、このレジスタはアナログ DAC 入力を直接駆動します。この機能は、ブロックを有効にする前に DAC コードの初期化に使用できます。

注: このレジスタへの書き込みは、現在進行中の DAC モードを無効にします。

バッファ モード

HPPASS_CSG_SLICEx_DAC_VAL_A レジスタは HPPASS_CSG_SLICEx_DAC_VAL レジスタのシングル バッファとして動作します。このモードが有効になり、その後の UPDATE トリガーがアサートされると、DAC_VAL_A の値は DAC_VAL に転送され、HPPASS_CSG_DAC_INTR レジスタの DAC_BUF_EMPTY 割込みが発生します。DAC_VAL_A に書き込まれ、DAC_VAL に転送された新しい値のみが割込みを発生させます。以下の図に、バッファ モードの動作を説明するために考えられる使用シナリオを示します。

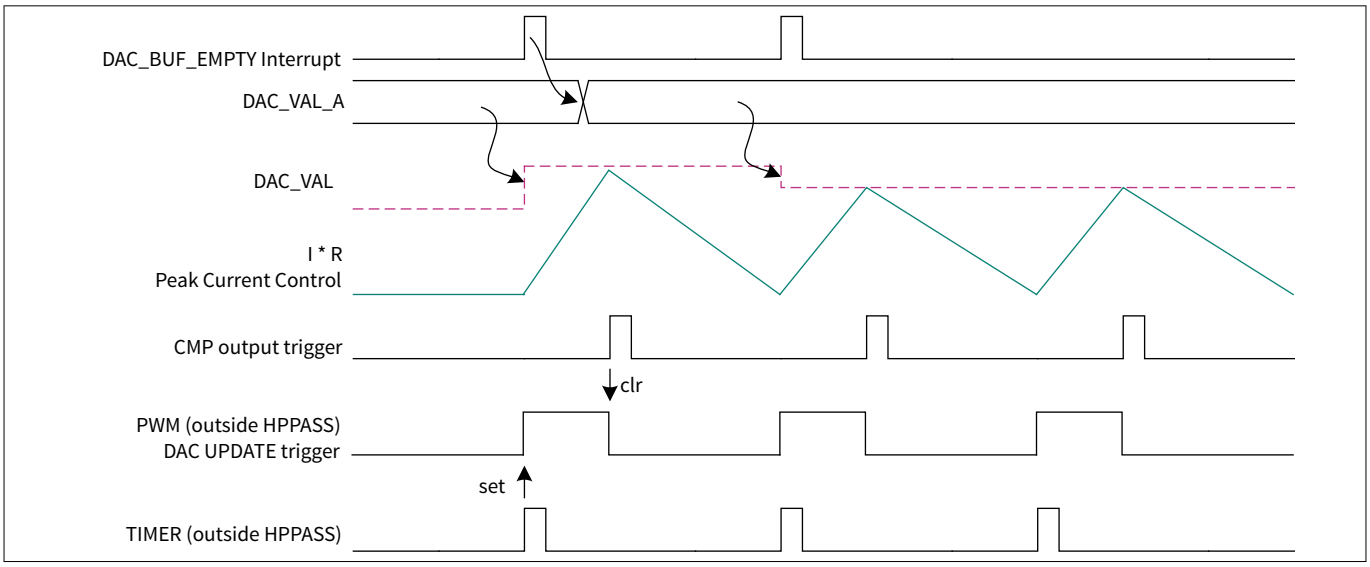


図 301 DAC バッファ モード

ヒステリシス モード

このモードでは HPPASS_CSG_SLICEx_DAC_VAL_A と HPPASS_CSG_SLICEx_DAC_VAL_B レジスタの 2 つのバッファ値があります。これにより、単一のコンパレーターを使用したハイセンス (A)、ローセンス (B) のトリップ動作が可能になります。

- 有効にすると、ハイセンス値 DAC_VAL_A が DAC_VAL に転送され、出力されます。
- UPDATE トリガーが 0 から 1 に遷移すると、DAC_VAL_B は DAC_VAL に転送されます。
- UPDATE トリガーが 1 から 0 に遷移すると、DAC_VAL_A は DAC_VAL に転送されます。

DAC_VAL_A の更新時に DAC_BUF_EMPTY 割込みが発生し、ユーザーはバッファ値を更新できます。これらは、次の使用シナリオ図の例に示すように、次の DAC_VAL_A 更新からハードウェアによって使用されます。

このモードでは、コンパレーターを (図 302 に示すように) 更新トリガーとして使用でき、ユーザーは CMP 出力を使用するか、またはそれを反転させる柔軟性があります。DAC_VAL_A と DAC_VAL_B の値を変調することにより、コンパレーター出力の CMP 周期とデューティサイクルを変調できます。

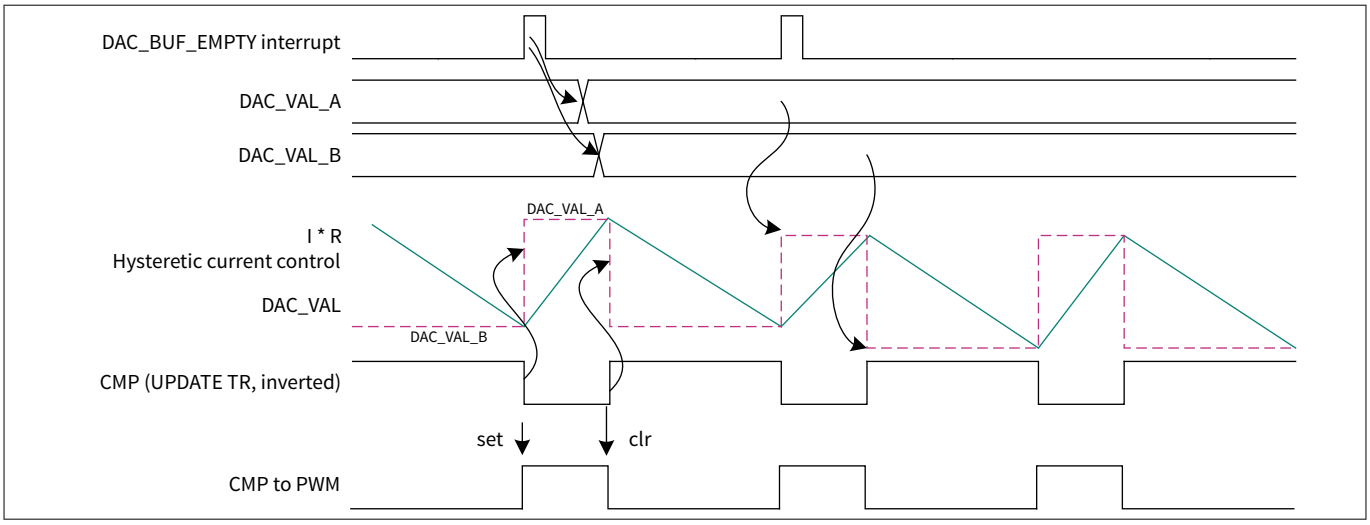


図 302　　DAC ヒステリシス モード

スロープ生成モード

スロープは以下のスロープ生成モードとして設定できます。DAC_STEP は HPPASS_CSG_SLICEx_DAC_CFG.DAC_STEP ビットによって設定され、DAC_VAL_A および DAC_VAL_B はそれぞれ HPPASS_CSG_SLICEx_DAC_VAL_A レジスタと HPPASS_CSG_SLICEx_DAC_VAL_B レジスタに設定されます。

- SLOPE_RISING: DAC_VAL_A で開始し、DAC_STEP のみインクリメントし、DAC_VAL ≧ DAC_VAL_B で停止します。
- SLOPE_FALLING: DAC_VAL_A で開始し、DAC_STEP のみデクリメントし、DAC_VAL ≦ DAC_VAL_B で停止します。
- SLOPE_TRIANGULAR: DAC_VAL_A からスタートし、DAC_VAL ≧ DAC_VAL_B まで DAC_STEP のみインクリメントし、DAC_VAL ≦ DAC_VAL_A まで DAC_STEP のみデクリメントします。

図 303 の使用例にあるように、パラメーターの更新は新しいスロープの開始時にのみ行われます。START (DAC_VAL_A)、END (DAC_VAL_B)、DAC_STEP、DAC_PERIOD の任意の組み合わせが更新用にバッファリングされます。

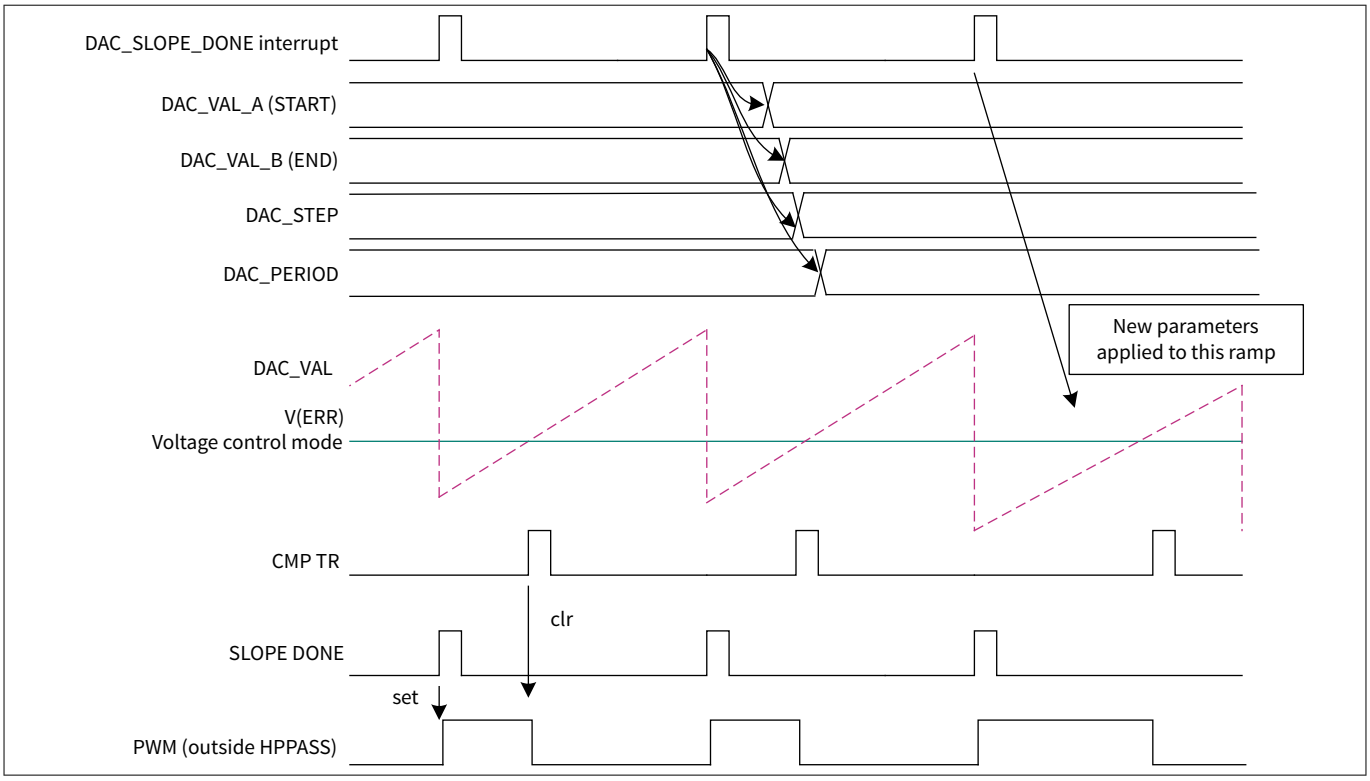


図 303　　DAC スロープ生成例

パラメーター更新を制御する方法は 3 つあります。

1. **連続**: 連続モードを有効にするためには HPPASS_CSG_SLICEx_DAC_CFG レジスタの DAC_CONT ビットを設定します; ハードウェアはスロープの開始時にバッファされたパラメーターを更新し (図 303 の例)、終了値に達すると再開します; そうでなければワンショットモードとなり、最後の出力値が保持されます。この方法では、ファームウェアのパラメーター更新書き込みのタイミングを Slope Done トリガーに合わせられます。

   ***注**:　　DAC 連続モードはスロープと LUT 生成モードでのみ有効です。*

2. **同期**: HPPASS_CSG_SLICEx_DAC_CFG レジスタの DAC_PARAM_SYNC_EN ビットを設定します。ハードウェアは、HPPASS_CSG_SLICEx_DAC_PARAM_SYNC レジスタの READY ビットが'1'に設定されたときのみ、パラメーターを更新します (まだスロープの開始時)。これは、バッファされたパラメーターがすべて書き込まれたことをハードウェアに示すビットです。このメソッドは、パラメーターの部分的な更新を防ぎます。

   ***注**:　　パラメーターがレジスタに書き込まれた後、HPPASS_CSG_SLICEx_DAC_PARAM_SYNC.READY ビットを'1'に設定し、READY ビットが'1'になるまでこのビットを読み出します。*

   ***注**:　　HPPASS_CSG_SLICEx_DAC_PARAM_SYNC.READY ビットはパラメーター更新イベントでハードウェアによりクリアされます。*

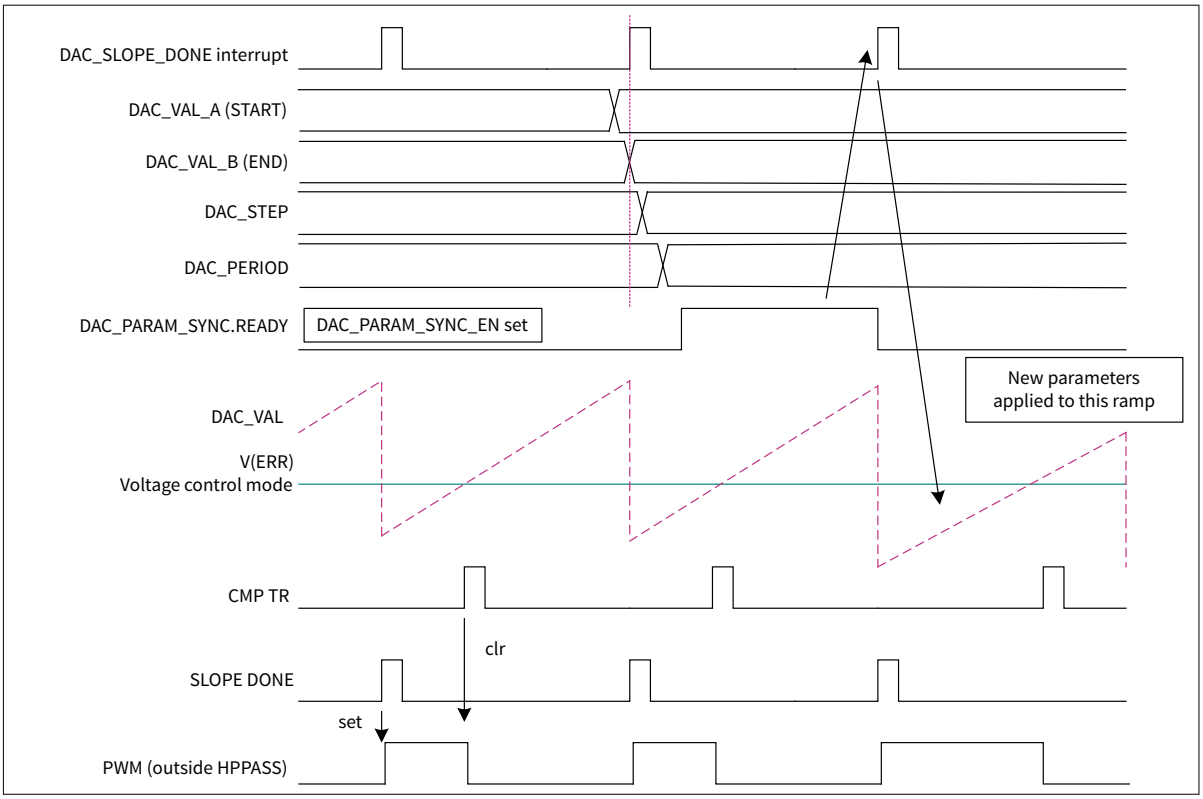


図 304　　パラメーターの同期更新

3. **カスケード**: HPPASS_CSG_SLICEx_DAC_CFG レジスタの DAC_CASCADE_EN ビットを設定して、連続する番号の CSG スライス間でパラメーター更新を同期させます。この機能は、互いに固定の位相関係で動作する 3 つの CSG スライスを使用するアプリケーションで有用です。

   ユーザーは、高次 CSG スライス (N+1 および N +2) では DAC_CASCADE_EN ビットを設定し、低次スライス (N) では HPPASS_CSG_SLICEx_DAC_CFG レジスタの DAC_PARAM_SYNC_EN ビットを設定します。

   その後、ファームウェアはこれら 3 つのスライスでバッファされたパラメーターを更新し (順番は問わない)、完了すると低次スライスに DAC_PARAM_SYNC.READY ビットが設定されます。図に示すように、以下の CSG Slope Done イベントにおいて、パラメーターはこれらの連続するスライスにおいてカスケード方式で更新されます。

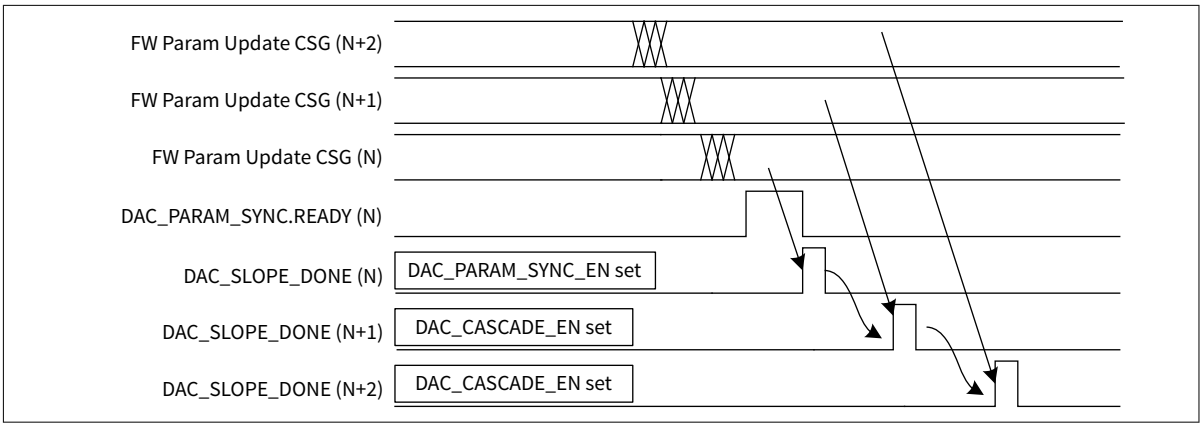


図 305　　カスケード パラメーター更新

LUT 波形生成モード

LUT モードは、パラメーターがデータ関連ではなくアドレス関連であることを除けば、スロープ生成モードと非常によく似た動作をします。生成されたアドレス シーケンスはルックアップ テーブルに送られ、ユーザーがカスタム波形を保存できます。テーブルからフェッチされた連続する各 10 ビットのデータは、DAC_VAL にロードされ、出力

のために DAC によって駆動されます。制御パラメーターの更新の詳細については、スロープ生成モードを参照してください。最大更新レートは CLK_CSG/2 に制限されます。

- LUT_FORWARD: 開始アドレスは DAC_VAL_A で、アドレスを DAC_STEP のみインクリメントし、アドレス≧DAC_VAL_B のときに停止します。
- LUT_REVERSE: 開始アドレスは DAC_VAL_A で、アドレスを DAC_STEP のみデクリメントし、アドレス ≦ DAC_VAL_B のときに停止します。
- LUT_FORWARD_REVERSE: スタート アドレスは DAC_VAL_A で、アドレスが≧DAC_VAL_B になるまで DAC_STEP のみインクリメントし、アドレスが≦DAC_VAL_A になるまで DAC_STEP のみデクリメントします。

注: 共有される CSG スライス 3 つにつき 1 つの LUT があります。CSG は PSOC™ Control C3 MCU に LUT 0 と LUT 1 を持ち、各 LUT は 128 サンプルで、DAC0/1/2 は LUT 0、DAC3/4 は LUT 1 です。

上記のモードはコンパレーターの負入力に出力電圧を生成するように DAC を制御し、各 CSG スライスの DAC 出力値は HPPASS_CSG_SLICEx_DAC_STATUS.DAC_VAL レジスタで読み出せます。

27.2.3.3 DAC モードスタート/ストップ

モードは、現在進行中の DAC 操作がない場合にのみ変更できます。

新しい DAC モードのファームウェアまたはハードウェアを開始するためには、2 つの方法があります (HPPASS_CSG_SLICEx_DAC_MODE_START レジスタの詳細については、レジスタ技術リファレンスマニュアルを参照)。

- **HW_START**: このビットが設定されている場合、DAC モード開始は次に選択される DAC_TR_START トリガーに同期されます。DAC 開始トリガーを選択するためには、HPPASS_CSG_SLICEx_DAC_CFG レジスタの DAC_TR_START_SEL フィールドを設定してください。選択可能なトリガーは、いずれかの外部ハードウェアトリガー入力または AC トリガーです。

  注: DAC モードが進行中にハードウェア開始トリガーが再度アサートされた場合、無視されます。

- **FW_START**: このビットが設定されると、すぐにモードが有効になります。

HPPASS_CSG_SLICEx_DAC_MODE_START レジスタの HW_START/FW_START ビットは、HPPASS_CSG_SLICEx_DAC_CFG.DAC_CONT ビットが連続モード用に設定されていない限り、DAC_SLOPE_DONE イベントで自動的に'0'にリセットされます。

現在のモードを停止するためには、関連するスタート ビットをクリアします。現在の DAC_VAL は、直接上書きされるか、新しいモードが開始されるまで、最後の値に固定されます。

HPPASS_CSG_SLICEx_DAC_VAL レジスタへの直接書き込みは、HW_START ビットと FW_START ビットを 0 にリセットしてモードを無効にすることに加えて、現在の DAC 出力を上書きし、進行中の DAC モードを無効にします。

HPPASS_CSG_SLICEx_DAC_MODE_START レジスタは読み出し専用の BUSY ビットを持ち、DAC モードが有効化され、現在進行中であることを示します。FW_START がモードの開始に使用される場合、FW_START ビットはこのビットと等価で、モードが完了または停止したときにローとなります。HW_START がモードの開始に使用される場合、このステータス ビットは DAC_START_TR イベントでハイになり、モードが完了または停止するとローになります。

27.2.3.4 DAC アップデート トリガー

HPPASS_CSG_SLICEx_DAC_CFG レジスタの DAC_TR_UPDATE_SEL ビットを設定し、DAC 更新トリガーを選択します。選択された DAC_TR_UPDATE がアサートされると、DAC コードは更新され、動作は選択された DAC_MODE に依存します。DAC 更新トリガーには以下のトリガーが利用可能です。

- 他のオンチップ ペリフェラルまたはファームウェア トリガーからの 8 つの HPPASS ハードウェア トリガー (詳細は、入力トリガーの章を参照)
- CSG スライス ローカル DAC_PERIOD カウンター (分周器は HPPASS_CSG_SLICEx_DAC_PERIOD レジスタでプログラムされます)

- CSG スライスのローカル コンパレーター出力または反転ローカル コンパレーター出力 (後処理の章を参照)
- AC トリガー

注: HPPASS ハードウェアトリガーまたはファームウェアトリガーのいずれかがトリガーの更新に選択されている場合、トリガーはパルス モードでなければいけません。さらに、アクティブトリガー エッジの最大周波数は、CSG CLOCK/2 以下でなければいけません。

DAC 周期カウンター

DAC_PERIOD が選択されている場合は、クロック サイクル単位で周期をプログラムします。整数 (12 ビット) フィールドと、オプションで分数 (5 ビット) フィールドがあります。少数部は 1/32 クロック分周までの分解能を可能にします。HPPASS_CSG_SLICEx_DAC_PERIOD レジスタの詳細については、レジスタリファレンス マニュアルを参照してください。

分数演算は、より高精度のスロープ生成を可能にします。しかし、この機能を使用すると、図 306 に示すようなサイクルジッターが発生します。この例では、4 で割るだけの整数があり、その下に$4\frac{1}{2}$で割り切れる整数と分数部分があります。次の図に示すように、$4\frac{1}{2}$サイクルは、4 サイクルと 5 サイクルの交互の平均として生成されます。

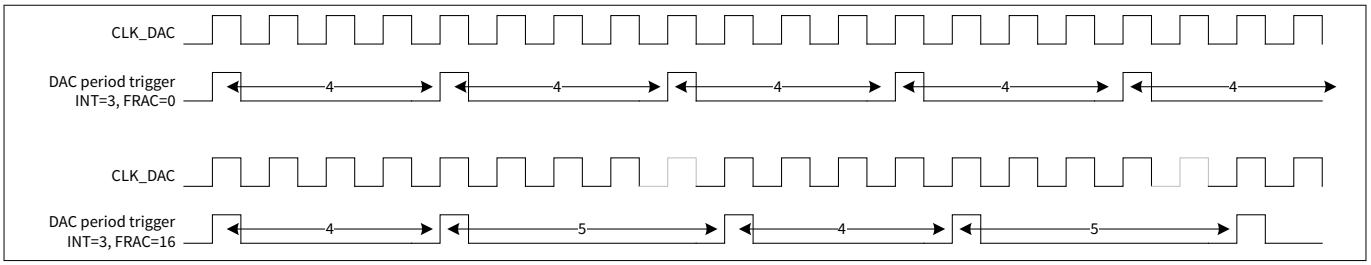


図 306 DAC 周期の例

より複雑な波形の例として、$20\frac{7}{8}$を考えてください。つまり、167 クロックの中に 8 つのトリガーがあり、その結果、20-21-21-21-21-21-21-21 というパターンになります。

27.2.3.5 後処理

次の図は、CSG コンパレーターの後処理におけるハードウェア リソースの図です。

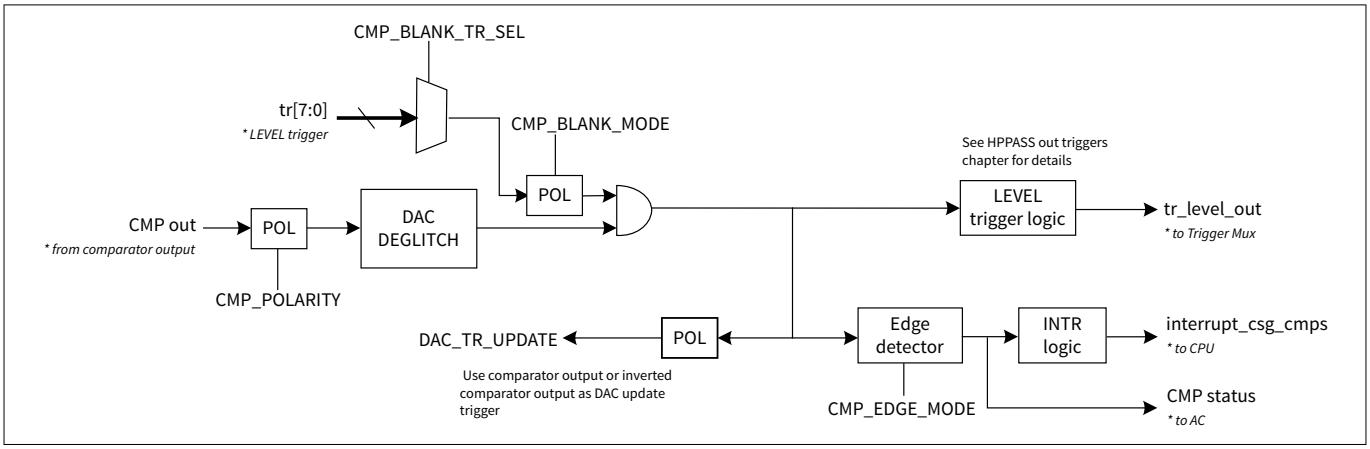


図 307 CSG コンパレーターの後処理

CSG コンパレーターの後処理の主な特長は以下のとおりです。

- コンパレーター出力の極性設定
- コンパレーターをブランキングし、入力トリガーで'0'にゲーティングします。
- DAC コードの変更により発生する可能性のあるコンパレーター出力のデグリッチ;コンパレーター出力はプログラム可能なクロックサイクル数だけラッチされます。
- 設定可能な (正のエッジ、負のエッジ、または両方のエッジ) コンパレーター割込みおよび AC ステータスの生成
- コンパレーター出力または反転コンパレーター出力を更新トリガーとして選択する機能 (例えば、ヒステリシス モードの使用など)

コンパレーター出力の極性とエッジ モードは、HPPASS_CSG_SLICEx_CMP_CFG レジスタの CMP_POLARITY と CMP_EDGE_MODE ビット フィールドで個別に設定されます。各 CSG スライスのコンパレーター出力値は HPPASS_CSG_SLICEx_CMP_STATUS レジスタの CMP_VAL ビットで読み出せます。

コンパレーターブランキング

PWM によって制御される電源スイッチが最初にオンになると、大きな電流過渡が発生し、コンパレーターが早期にトリップすることがあります。したがって、コンパレーター ブランキング機能は、入力トリガーによって提供されますが、入力トリガーはレベルトリガーでなければいけません。図 308 に、典型的なシナリオを示します。

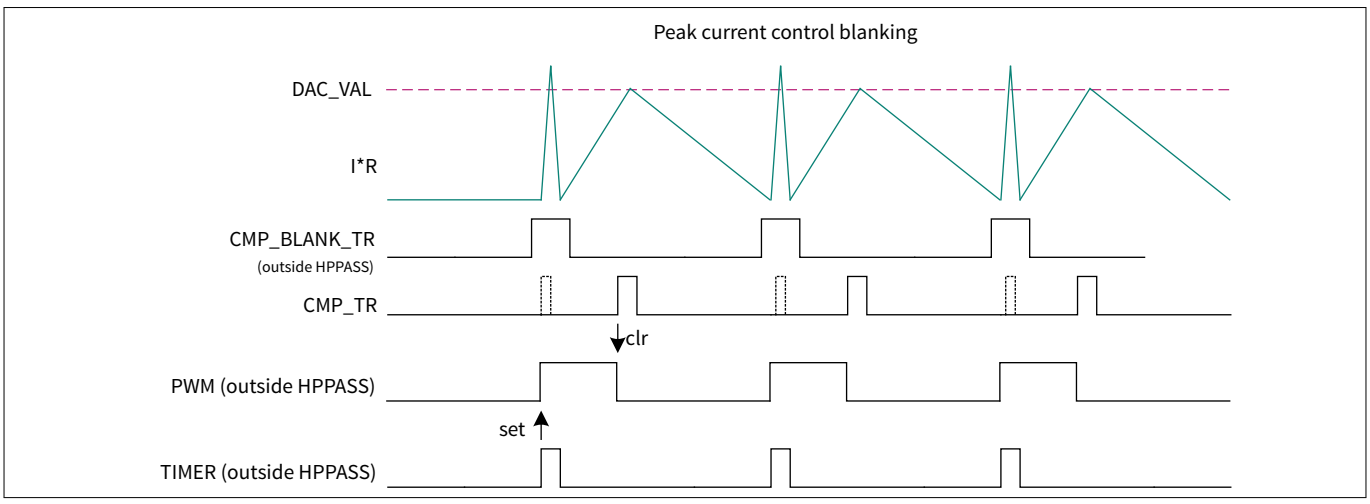


図 308 コンパレーターのブランキング例

HPPASS_CSG_SLICEx_CMP_CFG レジスタの CMP_BLANK_TR_SEL ビットフィールドを設定し、HPPASS 入力レベルトリガーから使用可能な入力の 1 つを選択します。このレジスタの CMP_BLANK_MODE ビットフィールドは、選択されたブランクトリガー信号の極性を設定するのに役立ちます (HPPASS_CSG_SLICEx_CMP_CFG レジスタの詳細については、レジスタ リファレンス マニュアルを参照)。

DAC 更新グリッチフィルター

DAC 更新時に発生する可能性のある過渡現象のため、伝播してはいけないコンパレーター出力が 1 サイクル以上存在する可能性があります。したがって、オプションの DAC 更新グリッチ フィルターは、プログラム可能なサイクル数の間、現在のコンパレーター出力値をラッチするために提供されます。サイクル数は、HPPASS_CSG_SLICEx_DAC_CFG レジスタの DAC_DEGLITCH ビット フィールドを設定することによって設定されます。図 309 に、DAC 更新時のコンパレーター出力ラッチの例を示します。

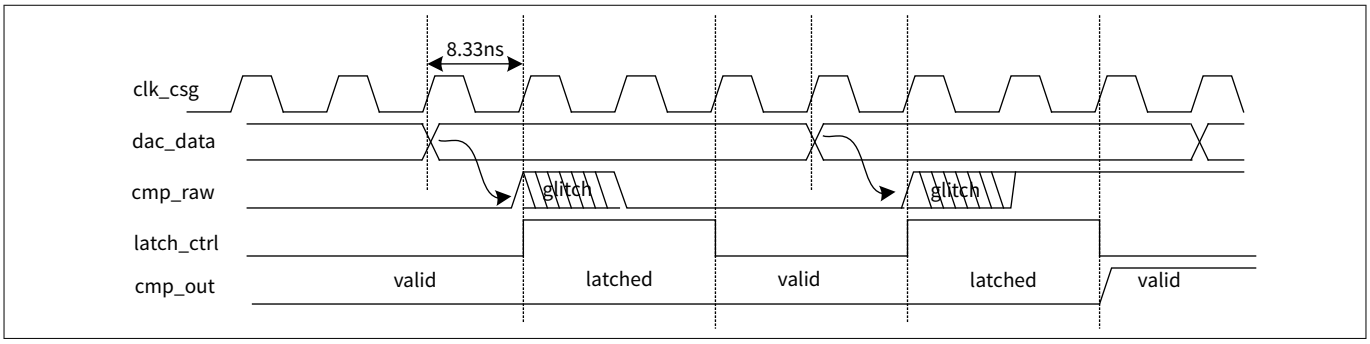


図 309　　DAC 更新時のコンパレーター ラッチ

新しいデータが DAC に提示されると、(新しい出力が伝搬される) 次のサイクルから、コンパレーターの出力は、プログラムされたサイクル数の間、ラッチされます (出力は現在の値を保持します)。CSG クロックが 120 MHz で動作している上記の例では、サイクル数は 2 つにプログラムされています。

DAC の観測ブランキング

図 310 は CSG DAC 出力の観測値から SAR ADC サンプラー入力までの信号経路の概略図です。この経路で、直列に接続されたこれらのスイッチのうち 2 つが閉じられた場合、DAC 出力に短絡過渡現象が発生し、コンパレーターの出力値が不正確になる可能性があります。

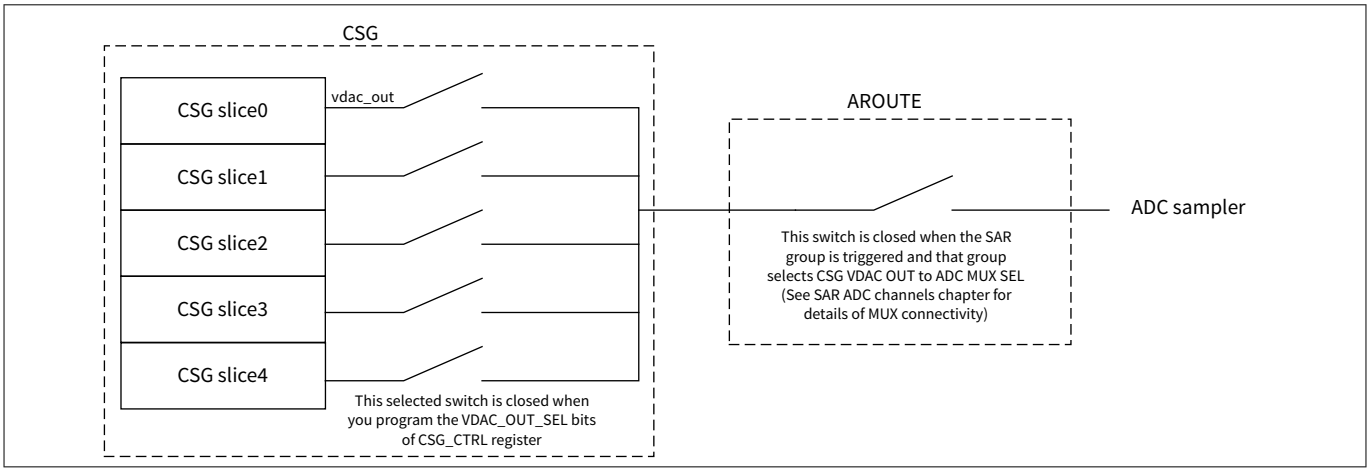


図 310　　DAC の観測ルーティング

この問題を軽減するために、選択された CSG クロック サイクル数の間、現在のコンパレーター値をラッチするために適用できるプログラム可能なコンパレーター ブランキング時間があります。このイベントはハードウェアで検出されるため、ユーザーはブランキング サイクルを HPPASS_CSG_VDAC_OUT_BLANK.BLANK_CNT レジスタで指定するだけです。デフォルトはブランキングなし。ブランキング動作は、CSG VDAC_OUT_SEL スイッチと AROUTE スイッチのいずれかが閉じているときにアクティブになります。

27.2.3.6 CSG 割込み

HPPASS CSG ブロックには、CPU 割込みコントローラーへの DAC およびコンパレーター割込みがあります。DAC 割込みは、DAC モードがハードウェアスタートトリガーで開始されたとき、DAC スロープ/LUT が完了したとき、または DAC バッファが空になったときに発生します。DAC 割込みは HPPASS_CSG_DAC_INTR_MASK レジスタを設定することで有効になり、HPPASS_CSG_DAC_INTR_MASKED レジスタを読み出すことで CPU 割込みコントローラーに転送される割込みマスク状態を知ることができます。

各 CSG DAC に対して個別の割込み (interrupt_csg_dac[4:0]) と、複合 DAC 割込み (interrupt_csg_dacs) が提供されます (詳細は、割込みの章を参照)。複合割込みが使用される場合、割込みの原因を特定するために HPPASS_CSG_DAC_INTR レジスタを読まなければいけません。

HPPASS_CSG_CMP_INTR レジスタを読めば、どの CSG コンパレーター割込みが発生したかを知ることができます。HPPASS_CSG_SLICEx_CMP_CFG レジスタの CMP_EDGE_MODE ビットを設定することにより、CSG コンパレーター割込みタイプを無効、立ち上りエッジ、立ち下りエッジ、または両方のエッジに選択でき、HPPASS_CSG_CMP_INTR_MASK レジスタの CMP 指定ビットを設定することにより有効になり、HPPASS_CSG_CMP_INTR_MASKED レジスタを読み出すことにより、CPU 割込みコントローラーに転送される割込みのマスク状態を知ることができます。

27.2.4 自律型コントローラー (AC)

27.2.4.1 概要

自律型コントローラー (AC) は、CPU や外部の介入を必要としないハードウェア制御を提供します。

AC はプログラム可能なステートマシンです。図 311 に AC システム ブロック図を示します。メモリベースの状態遷移表 (「タイマー表」)、シーケンサー、タイマーで構成されます。

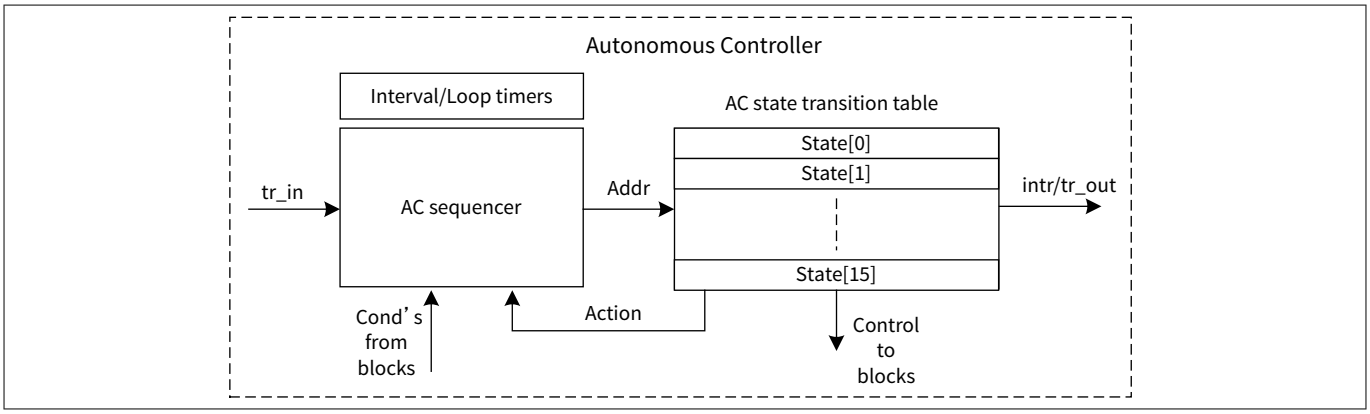


図 311 自律型コントローラーのブロック図

AC には以下の機能があります。

- ブロックの有効化と無効化、ファームウェアへの BLOCK_READY ステータスの提供
- ハードウェア/ファームウェアによってトリガーされ、プログラムされたアクションを実行する能力
- HPPASS 内部ブロック (SAR グループ、CSG スライス) のトリガー
- ハードウェア遅延の実装
- 時限ループの中で周期的なアクション (例えば、トリガー) をとります。
- HPPASS 内部ブロックから条件を受け取り、そのイベントに対して動作します (例えば、トリガーをかける)。
- CPU に AC ベースの割込みを発生させます
- AROUTE スイッチの制御 (注: 高度な使い方のみ)
- SAR ADC 限界検出状態を監視し、その状態に応じたアクションをとります。

27.2.4.2 ブロックインターフェース

AC は図 312 に示すように、HPPASS の各サブブロックへの標準インターフェースを持っています。自律機能は状態遷移表で実装され、ハードウェア リソースに制御を出力するフィールドと、それらのリソースが生成する入力イベントを符号化した RAM ワードで構成されます。

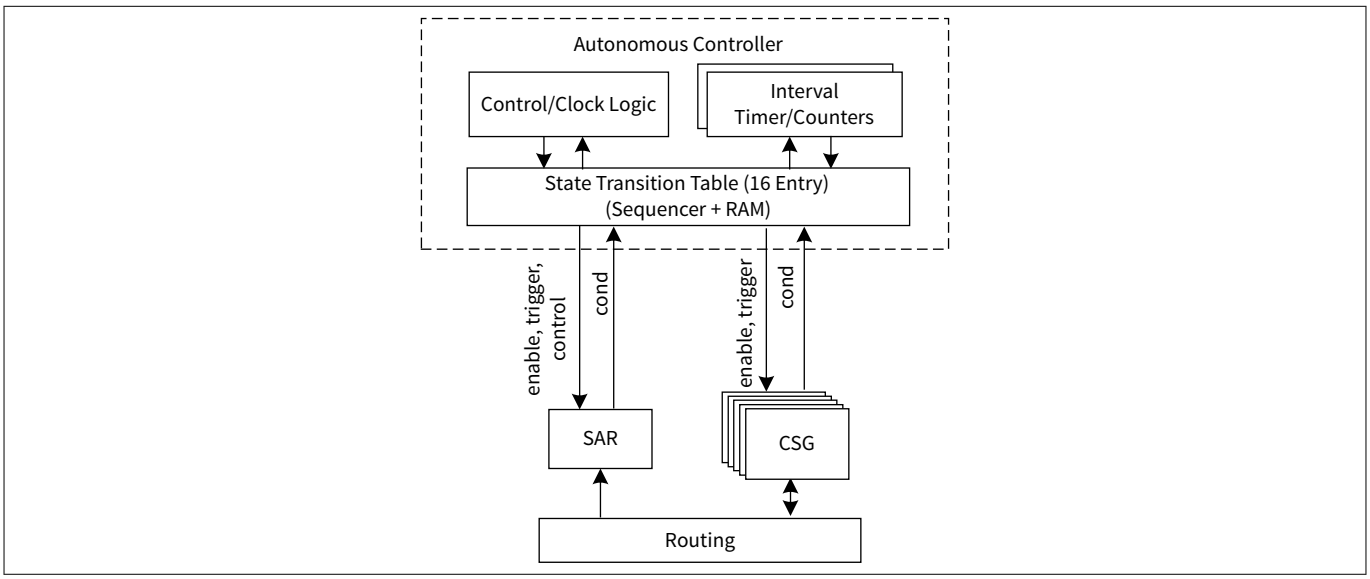


図 312 自律型コントローラーブロック インターフェース

AC ブロックのインターフェースは以下のように構成されています。

- **有効**
  各機能ブロックには 1 ビットのイネーブル制御があります。ビットが無効 ('0') の場合、ロジックはリセット状態に保持され、クロックゲートされ、ダイナミック電力を消費せず、アナログブロックへの制御はすべて無効状態になります。ビットが有効 ('1') の時、ロジックはアクティブで、クロックはゲートされていません。
  指定されたアプリケーション ブロックのセットを有効にした後、アプリケーションを開始する前に「BLOCK_READY」ステータスを待ちます。アプリケーションが完了した後に機能ブロックを無効にしても、BLOCK_READY は Low に設定されません。したがって、BLOCK_READY をポーリングするのは、機能ブロックを再有効化するときだけです。
- **トリガー**
  自律型コントローラーに関連して、トリガーは、与えられた機能ブロックのアクティビティを開始する (または、必要であれば、そのブロックを新しいアクティビティにシーケンスする) ための方法です。基本的な例は、SAR ADC トリガーです。SAR と CSG の両方にトリガー入力があります。
- **条件**
  条件入力は、コントローラーシーケンスの分岐に使用できる、機能ブロックからコントローラーへのステータスです。例: ENTRY_DONE, SAR_RANGE, DAC_SLOPE_DONE
- **その他の設定/制御**
  AC は、他のサブブロックの制御にも適応できるよう、十分に汎用性があります。例えば、AC を使用して AROUTE MUX の選択を制御するオプションの使用例があります。

27.2.4.3 状態遷移表

AC はシンプルなフォーマットでシングルスレッド プログラミング シーケンスを実装できます。

AC プログラムは、N 個の状態 (またはプログラミング命令) からなる RAM テーブル (状態遷移テーブルと呼ばれる) に格納されます。PSOC™ Control C3 MCU では N が 16 なので、16 のプログラミング命令が利用できます。

状態遷移表には、CPU ファームウェアがサブルーチンのように呼び出すことができる 1 つまたは複数のプログラムを含められます。

AC インプリメンテーションでは、各命令ステートは 4 つのレジスタ (TT_CFG0, TT_CFG1, TT_CFG2, TT_CFG3) にまたがります。詳細については Registers Technical Reference Manual を参照してください。

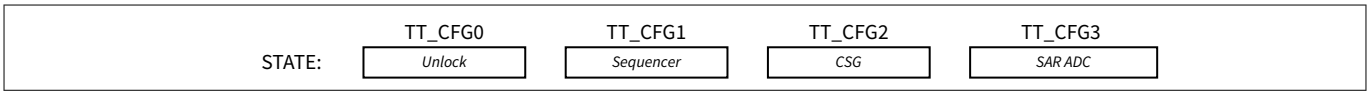


図 313　　AC 状態遷移表のステート

各レジスタ機能の簡単な説明:

- **アンロック**: アンロック機能の目的は、ブロックの有効化を特別に制御することです。あるブロックの有効化をあるステートでアンロックすると、そのステートでのみイネーブルが更新されます。これにより、通常の AC プログラム実行過程における不用意なモード移行を防げます。
- **シーケンサー**: これが AC の"頭脳"です。このレジスタでは、遅延、ループ、分岐、条件チェックなど、実際のプログラムのフローが指定されます。
- **CSG**: CSG スライスを有効にし、オプションでトリガーするために使用されます。
- **SAR ADC**: SAR ADC を有効にし、オプションで SAR ADC グループをトリガーし、AROUTE を制御するために使用されます。

AC シーケンサー プログラミングのフロー

TT_CFG1 レジスタの以下のビット フィールドは、プログラム フローを制御するために AC シーケンサーと関連しています。

表 254　　TT_CFG1 レジスタのビット フィールド

Bit_Name[Bit_Pos]	説明
BR_ADDR[3:0]	分岐した場合の絶対アドレス (0～N-1 可能)
COND[13:8]	自律型コントローラーの状態
ACTION[18:16]	自律型コントローラーのアクション
INTR_SET[19]	AC 割込みの設定設定されていれば、CPU 割込みまたはハードウェアトリガー出力を生成します。
CNT[31:20]	インターバル カウントとループ カウントのタイマー/カウンター値の設定に使用。

AC シーケンサーのフィールドについては、次のセクションで詳しく説明します。

- **条件**
  HPPASS サブ ブロックが生成、出力するブロック ステータスのデバイス固有のリスト。以下の表は、すべての AC コンディションの一覧です (詳細は、『レジスタ リファレンス マニュアル』を参照)。

  表 255　　自律型コントローラーの状態

状態	説明
FALSE	ハードコードされた FALSE 条件。無条件分岐で使用。
TRUE	ハードコードされた TRUE 条件。無条件分岐で使用。
BLOCK_READY	ブロックのスタートアップが完了したときに設定され、有効化されたブロックが動作可能な状態であることを示します
CNT_DONE	遅延とループに使用。アクションが WAIT_FOR の場合 - インターバル時間。アクションが BRANCH_IF_TRUE/FALSE の場合 - ループカウンター。

  (続く)

表 255 (続き) 自律型コントローラーの状態

状態	説明
SAR_ENTRYn_DONE (n = 0〜7)	SAR ADC グループ終了
SAR_RANGEn (n=0〜7)	SAR ADC 限界検出条件がアサートされます。
SAR_BUSY	SAR ADC busy
SAR_FIRn_DONE (n = 0/1)	SAR ADC FIR 終了
SAR_QUEUE_HI/LO_EMPTY	SAR ADC の高/低優先度キューが空です
SAR_QUEUES_EMPTY	SAR ADC 両方のキューが空です
HW_TRn_IN (n = 0〜7)	INFRA からのハードウェア/ファームウェア トリガーがアサートされました
FIFO_LEVELn (n = 0〜3)	FIFO ステータスレベル
CSGn_SLOPE_DONE (n = 0〜7)	CSG スロープ/LUT 波形が完成
CSGn_CMP_TR (n = 0〜7)	CSG コンパレータートリップが発生しました

ブロックからの条件は AC へのパルスとして生成されます。AC は常に特定のコンディションを待っているとは限らないため、これらのコンディションはラッチされ、AC がチェックしたときにのみクリアされます。これにより、以下の例のように、コンディションの AC ポーリングが可能になります。

例: グループ 0,1,2 の SAR ADC グループ完了 (SAR_ENTRY_DONE) はこのポーリング ループでチェックされます。このループでチェックされた時に条件が TRUE (セット) であれば、その条件はクリアされ、ルーチンはその SAR ADC グループを再びトリガーする状態にジャンプし、その後この POLL ループにジャンプバックします。

```
STATE: POLL
    ACTION=BRANCH_IF_TRUE    BR_ADDR=<TR_SAR_ENTRY0>    COND=SAR_ENTRY0_DONE

STATE:
    ACTION=BRANCH_IF_TRUE    BR_ADDR=<TR_SAR_ENTRY1>    COND=SAR_ENTRY1_DONE

STATE:
    ACTION=BRANCH_IF_TRUE    BR_ADDR=<TR_SAR_ENTRY2>    COND=SAR_ENTRY2_DONE

STATE:
    ACTION=BRANCH_IF_TRUE    BR_ADDR=POLL    COND=TRUE
```

- **アクション**
  各命令には、これらのアクションのいずれかが含まれていなければいけません。

表 256 自律型コントローラーのアクション

アクション	説明
STOP	この状態で停止 (デフォルトの状態) これで AC は止まります。ACTRLR.CMD_RUN への CPU ファームウェアの書き込みは、指定されたアドレスから実行を再開するために必要です。 STOP は常に、与えられた AC ルーチンの最後の命令にあります。
NEXT	次のステージへ。AC クロック 1 サイクルの間、この状態を維持します。

(続く)

表 256 (続き) 自律型コントローラーのアクション

アクション	説明
WAIT_FOR	この状態に留まり、指定された COND を待ち、条件が True の場合、次の状態に移行します。
BRANCH_IF_TRUE	指定された COND をチェックし、条件が True なら BR_ADDR の指定された状態に進み、条件が False なら次の状態に進みます
BRANCH_IF_FALSE	指定された COND をチェックし、条件が False なら BR_ADDR の指定された状態に進み、条件が True なら次の状態に進みます

注: 以下のコードフラグメントは、レジスタへの書き込み形式ではなく読みやすい形式で記述されています。それぞれの状態はオプションのラベル付きの STATE: で区切られ、コードは <FIELD_NAME> = <FIELD_VALUE> (HPPASS_ACTRLR_TTCFG レジスタから) の形式で示されます。詳細は『レジスタ技術リファレンスマニュアル』を参照してください。

例: SAR ADC グループ 0 (SAR_ENTRY0) をトリガーし、次の状態に移行します。

```
STATE:
    SAR_TR=0x01    ACTION=NEXT
```

例: SAR ADC グループ 0 (SAR_ENTRY0) をトリガーし、そのグループの変換が完了するまでこの状態で待ちます (SAR_ENTRY0_DONE)。

```
STATE:
    SAR_TR=0x01    ACTION=WAIT_FOR    COND=SAR_ENTRY0_DONE
```

例: CSG スライス 0 のコンパレーターがトリップすれば新しい状態に分岐し、そうでなければ次の状態に移動します。

```
STATE:
    ACTION=BRANCH_IF_TRUE    BR_ADDR=<STATE#_OR_LABLE>    COND=CSG0_CMP_TR
```

- **割込み**

  AC は、INTR_SET ビットを設定することにより、CPUSS 割込みまたはパルストリガーを所定の状態で発生できます。これは HPPASS_MMIO_HPPASS_INTR レジスタを設定することにより割込みとして、または HPPASS_MMIO_TR_PULSE_OUT レジスタを設定することによりトリガーとして設定できます。

```
STATE:
    INTR_SET=1
```

- **カウンター**

  HPPASS には 2 つのカウンターがあり、文脈に応じてインターバル カウンターまたはループ カウンターとして暗黙的に割り当てられています。

  条件が WAIT_FOR の場合、命令はインターバル タイマーで、AC は CNT クロック サイクルの間その状態に留まります。

  条件が BRANCH_IF_TRUE/FALSE の場合、その命令はループ カウンターです。初期 CNT がロードされ、この状態が実行されるたびにカウンターがデクリメントされます。'0'では、異なる実行フローが取られます。

例: この例は、時限遅延を実装するためのインターバル カウンターです。この状態で 64 HPPASS AC クロック サイクル待ち、次の状態に進みます。

```
STATE:
    ACTION=WAIT_FOR    COND=CNT_DONE    CNT=64
```

例: この例はループ カウンターです。ループ カウンターの場合、CNT 値は BRANCH 条件で指定されます。AC はこれをループと解釈します。BRANCH 条件がテストされるたびに、カウントはデクリメントされます。カウントダウンの間、COND は FALSE なので、コードは<TOP_OF_LOOP>アドレスに分岐します。ターミナル カウントでは、COND は TRUE なので、コードは次の状態に進みます。

```
STATE:
    ACTION=BRANCH_IF_FALSE    BR_ADDR=<TOP_OF_LOOP>    COND=CNT_DONE    CNT=16
```

27.2.4.4 初期化

HPPASS では、AC が起動のタイミングを調整するように設計されているため、初期化は AC にとって必須のユースケースです (詳細は、起動を参照)。

次の例は、SAR ADC と 3 つの CSG スライスを有効にし、BLOCK_READY ステータスを待ち、その後 CPU に割込みをアサートする標準の 2 ステート プログラムです (オプション)。

```
STATE:
    SAR_UNLOCK=1    CSG_UNLOCK=0x7    SAR_EN=1    CSG_EN=0x7    ACTION=WAIT_FOR    COND=BLOCK_READY

STATE:
    INTR_SET=1    ACTION=STOP or NEXT
```

割込みはオプションで、ファームウェアは BLOCK_READY ステータス (HPPASS_ACTRLR.BLOCK_STATUS.READY) をポーリングすることもできます。

HPPASS の起動方法:

- AC を有効 (HPPASS_ACTRLR_CTRL.ENABLED = 1)
- AC ステート アドレス (HPPASS_ACTRLR_CMD_STATE) を設定 (POR 時のデフォルトは 0)
- AC を実行 (HPPASS_ACTRLR_CMD_RUN = 1)

AC は (スタートアップ タイミング ハードウェアと連動して) ブロックのスタートアップを有効にし、シーケンスします。完了すると (BLOCK_READY がセットされる)、AC は停止状態になり、すべてのブロックはアプリケーションを開始できる状態になります。

27.2.4.5 例

以下の例は、実際に適用される場合とされない場合があります。これらの説明は、プログラム可能なシーケンスの種類を示します。また、初期化の章で説明したように、AC テーブルの最初の 1 つか 2 つの状態は常にプログラムされていると仮定しており、例には含まれていません。

***注**: AC は実際の RAM で構成され、POR で初期化されないため、プログラムで使用されるすべての状態について、4 つの TT_CFG0-3 レジスタすべてに書き込みを行う必要があります。*

SAR ADC と CSG の詳細については、SAR ADC の章とコンパレーターおよびスロープジェネレーター (CSG) の章を参照してください。

例 1 - 入力トリガーを SAR ADC または CSG (または両方) に遅延

この場合、外部トリガーがあり、1 つ以上の SAR ADC グループまたは CSG スライスをトリガーする前に、このトリガーから一定の遅延を実装することが目的です。以下のコードは、外部トリガーが HPPASS 内部 TR0 にルーティングされ (INFRA 設定)、SAR ADC グループ 4 がトリガーされると仮定しています。

固定遅延 CNT は 2048 (CNT+1) です。SAR ADC が 160 MHz で動作し、AC ロジックが固定 2 分周の場合、タイミング サイクルは 80 MHz (12.5 ns) であるため、この遅延は 25.6 µs となります。

```
STATE:TOP
    ACTION=WAIT_FOR    COND=HW_TR0_IN

STATE:
    ACTION=WAIT_FOR    COND=CNT_DONE    CNT=0x7ff

STATE:
    SAR_TR=0x10    ACTION=BRANCH_IF_TRUE    COND=TRUE    BR_ADDR=TOP
```

***注**:* *最後の命令動作はジャンプ (無条件分岐) です。*

***注**:* *HPPASS AC ロジックが 80 MHz でクロックされる場合、1 つのカウンターの最大遅延は 4096 または 51.2 µs です。ただし、HPPASS には 2 つのカウンターが存在します。詳細は AC シーケンサー プログラミングのフロー セクションの「COUNTERS」の説明を参照してください。ループ カウンターとループ内部のインターバル カウンターの両方を使用した場合、最大遅延は 2^{24} x 12.5 ns - 約 200 ms となります。*

以下は、最初に SAR ADC のトリガーを遅延させ、次に 3 つの CSG スライス DAC のトリガーに異なる遅延を実装する修正バージョンです。

```
STATE:TOP
    ACTION=WAIT_FOR    COND=HW_TR0_IN

STATE:
    ACTION=WAIT_FOR    COND=CNT_DONE    CNT=0x7ff

STATE:
    SAR_TR=0x10    ACTION=WAIT_FOR    COND=CNT_DONE    CNT=0x3ff

STATE:
    CSG_DAC_TR=0x7    BRANCH_IF_TRUE    COND=TRUE    BR_ADDR=TOP
```

***注**:* *トリガーアクションは、WAIT_FOR アクションに関係なく、常に状態の最初のサイクルにあります。*

例 2 - AC トリガー SAR 平均化グループ

あるグループ内のチャネルに対して N による平均化 (蓄積) が設定されている場合、その ADC グループに対して N トリガー毎に 1 つの結果のみが生成されます。これをサポートするために、ユーザーは以下の選択肢があります。

- N 回の外部定期トリガー (または連続トリガー)
- その ADC グループに CONT (連続トリガー) ビットを設定します。
- AC を使用して ADC グループを N 回トリガーしてください

AC オプションの利点は、平均化された変換が 1 つの外部イベントによってトリガーされる場合です。あるいは、ファームウェアによってトリガーされるなど、測定がランダムな速度で行われる場合。

次のルーチンは、HPPASS 内部トリガー (INFRA トリガープログラミングにより、外部ハードウェアまたはファームウェアトリガーでもよい) を待ち、SAR ADC グループの変換完了 (SAR_ENTRY_DONE) 条件を待つたびに、指示された SAR ADC グループを N 回 (この例では 16 回) トリガーし、次の HPPASS トリガーを待つためにジャンプバックします。

```
STATE:TOP
    ACTION=WAIT_FOR    COND=HW_TR0_IN

STATE:TR_SAR
    SAR_TR=0x01    ACTION=WAIT_FOR    COND=SAR_ENTRY0_DONE

STATE:
    ACTION=BRANCH_IF_FALSE    BR_ADDR=TR_SAR    COND=CNT_DONE    CNT=15

STATE:
    ACTION=BRANCH_IF_TRUE    BR_ADDR=TOP    COND=TRUE
```

例 3 - 定期的なトリガー発生

AC は周期的なトリガーを作成するために使用できます。この例では、トリガー周波数は 10 kHz です (HPPASS デジタル クロックが 80 MHz の場合、80 MHz/8000 = 10 kHz)。これは最大インターバル遅延よりも大きいため、内部遅延とアウターループを使用して作成されます。

ファームウェアのループと同様、正確なサイクル タイミングを得るためには、ループを注意深く構築する必要があります。コメントと、ダミーの NEXT サイクルを追加して偶数にしたことに注意してください。

```
STATE:TOP
    ACTION=WAIT_FOR    COND=CNT_DONE    CNT=3997 // 3998x2 cycles

STATE:
    ACTION=BRANCH_IF_FALSE    COND=CNT_DONE    BR_ADDR=TOP    CNT=1 // 1x2 cycles

STATE:
    ACTION=NEXT // 1 cycle

STATE:
    INTR_SET=1    ACTION=BRANCH_IF_TRUE    COND=TRUE    BR_ADDR=TOP // 1 cycle
```

例 4 - ファームウェア トリガー コマンドに AC を使用

外部ハードウェアトリガーまたはファームウェアトリガーに割り当てることができるトリガーは 8 つありますが、以下のコマンドはファームウェアトリガーを追加する方法を示します。

以下のコードには、スタートアップ ルーチンと 2 つの追加ファームウェア コマンドが含まれています。

```
STATE: // state 0
    SAR_UNLOCK=1    CSG_UNLOCK=0x7    SAR_EN=1    CSG_EN=0x7    ACTION=WAIT_FOR    COND=BLOCK_READY

STATE: // state 1, AC is STOPPED after initialization
    INTR_SET=1    ACTION=STOP

STATE: // state 2, setup firmware command for SAR ENTRY trigger
    SAR_TR=0x01    ACTION=STOP

STATE: // state 3, setup firmware command for DAC slices 0,1,2 triggers
    CSG_DAC_TR=0x07    ACTION=STOP
```

SAR ADC グループ 0 をトリガーするステート 2 を呼び出します (AC が停止している場合を想定):

- AC 状態アドレスの設定 (HPPASS_ACTRLR_CMD_STATE = 2)
- AC を実行 (HPPASS_ACTRLR_CMD_RUN = 1)

これにより、ステート 2 でコマンドが実行され、その後 STOP します。
CSG DAC スライス 0,1,2 をトリガーするルーチンはステート 3 にあります。
このようにして、定義済みのルーチンを AC テーブルに保存し、ファームウェアから呼び出せます。

例 5 - AC を使って AROUTE を制御

次の例では、ADC グループ 0 のマルチプレクスサンプラーのすべての MUX オプションを循環させる方法を示します。
各状態の開始時に、ルーティングは指示された MUX 選択に変更され、AC はその ADC グループがトリガーされ、サンプリングされ、変換されるまでその状態で待機します。その後、次の状態に進み、MUX の選択を変更するなどします。
このようにして、ルーティング選択のシーケンスが所定の ADC グループに対して設定され、AC が自律的にルーティングを変更できます。

```
STATE:TOP
    SAR_AROUTE_TR=1    SAR_AROUTE_SEL=0    ACTION=WAIT_FOR    COND=SAR_ENTRY0_DONE

STATE:
    SAR_AROUTE_TR=1    SAR_AROUTE_SEL=1    ACTION=WAIT_FOR    COND=SAR_ENTRY0_DONE

STATE:
    SAR_AROUTE_TR=1    SAR_AROUTE_SEL=2    ACTION=WAIT_FOR    COND=SAR_ENTRY0_DONE

STATE:
    SAR_AROUTE_TR=1    SAR_AROUTE_SEL=3    ACTION=WAIT_FOR    COND=SAR_ENTRY0_DONE

STATE:
    ACTION=BRANCH_IF_TRUE    COND=TRUE    BR_ADDR=TOP  // Cycle through again
```

27.2.5 温度センサー

PSOC™ Control C3 MCU にはオンチップ温度センサーがあり、内部のダイ温度を測定するために使用されます。

温度センサーは、ダイオードの形をしたバイポーラ接合トランジスター (BJT) 1 個で構成され、このトランジスターは、アナログ基準ブロック (AREF) からの基準電流を使って、1 uA と 8 uA の 2 つのバイアス電流でバイアスされます。そのベース エミッタ間電圧 (V_{BE}) はコレクタ電流を一定としコレクターベース間電圧を 0 とするとき、温度に強く依存します。このプロパティは、SAR ADC を使用してトランジスタのV_{BE}を測定することによりダイ温度を計算するために使用されます。

温度センサーの出力は、AMUX0 または AMUX1 選択 3 をとおして SAR ADC に送ることができ、これらは ADC チャネル 15 と 19 に対応します (詳細については SAR ADC のチャネルの章を参照):

$$Ratio = (V_{BE1} - V_{BE2})/V_{BE2} \quad (6)$$

V_{BE1}およびV_{BE2}は以下の方法で測定できます。

- チャネル 15 または 19 のいずれかが選択されている場合、温度センサーはデフォルトで 1 uA の電流でバイアスされ、この出力は測定値に対応します。測定値: V_{BE2}
- HPPASS_SAR_CFG_TEMPSENSE_CTRL レジスタの TEMPSENSE_CURRENT_SEL ビットを"1"に設定すると、BJT に 8 uA が追加で流れ込み、これは測定値に対応しています。測定値:V_{BE1}

ダイ温度は以下の式で計算できます。

$$Temperature = A * Ratio^3 + B * Ratio^2 + C * Ratio + D \quad (7)$$

定数 A、B、C、D は工場キャリブレーション時に SFLASH_SAR_TEMP_COEF_A、SFLASH_SAR_TEMP_COEF_B、SFLASH_SAR_TEMP_COEF_C、SFLASH_SAR_TEMP_COEF_D レジスタに格納されます。

以下のフローチャートは、上記のような典型的な一連の作業を示しています。

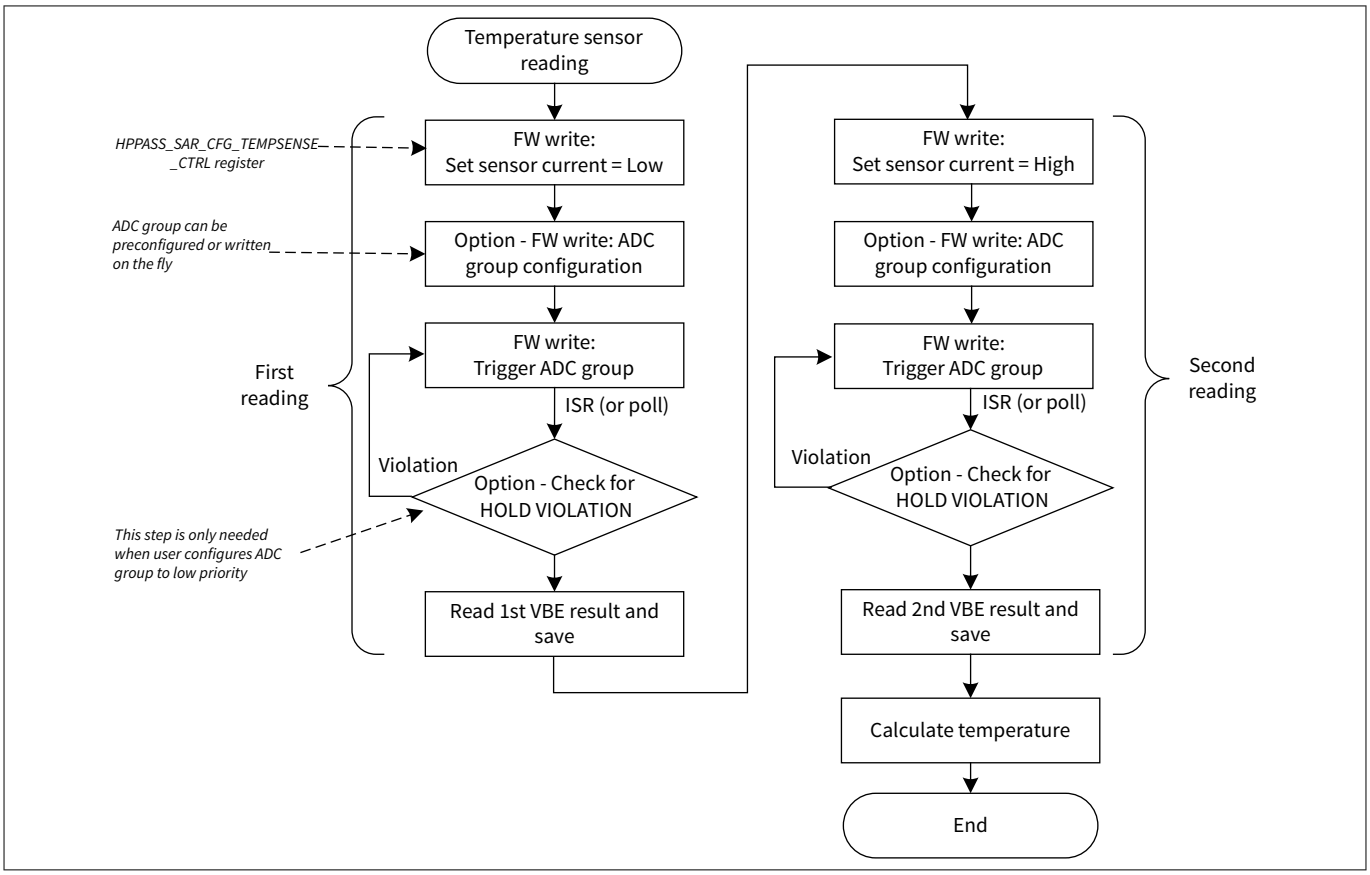


図 314　　温度測定シーケンス

最初に SAR ADC のV_{BE2}(1 µA に相当) が読み出され、2 回目にV_{BE1}(追加の 8 µA に相当) が読み出されます。その後、式を用いて温度を計算してください。

27.3 割込みおよびトリガー出力

割込み

HPPASS によって生成され、interrupt_hppass と interrupt_fifos として出力される 2 つの主要な割込みがあります。

- **interrupt_hppass** は、FIFO オーバーフロー/アンダーフロー、AC プログラム割込み、SAR ADC 結果オーバーフロー、エラー条件からなる割込み要因を組み合わせたものです。SAR ADC のエラー状態の詳細については、エラー状態の章を参照してください。割込みが設定されると、割込みをリセットするために HPPASS_MMIO_HPPASS_INTR に"1"を書き込む必要があります。割込みの要因を interrupt_hppass で観測するためには、それぞれの HPPASS_MMIO_FIFO_INTR_MASK を有効にする必要があります。
- **interrupt_fifos** は、各 FIFO からの FIFO レベルの組み合わせです。FIFO レベルイベントは HPPASS_MIO_FIFO_USEDx.USED ≧ HPPASS_MIO_FIFO_LEVELx.LEVEL+1 で発生します。interrupt_hppass とは異なり、ユーザーは interrupt_fifos をリセットする必要はありません。HPPASS_MIO_FIFO_USEDx.USED ＜ HPPASS_MIO_FIFO_LEVELx.LEVEL+1 の時にハードウェアがリセットします。interrupt_fifos で FIFO 割込みを観測するためには、それぞれの HPPASS_MMIO_FIFO_INTR_MASK を有効にする必要があります。

トリガー

HPPASS にはレベルトリガーとパルストリガーの 2 種類の出力トリガーがあります。レベルトリガーは主に TCPWM 制御を目的としたモーター制御アルゴリズムでの使用を想定しています。一方、パルストリガーは主に DMA インターフェースおよび制御を目的としたイベントです。生成され、tr_fifo_level_out[3:0]、tr_level_out[7:0]、tr_pulse_out[7:0] として出力される 3 つの主なトリガーがあります。

- **tr_fifo_level_out[3:0]**は FIFO レベルのレベルトリガーです。トリガー要因の選択は HPPASS_MMIO_FIFO_CFG レジスタの FIFO_TR_EN ビットフィールドで設定します。詳細は、SAR ADC FIFO の章を参照してください。
- **tr_level_out[7:0]**は CSG_CMP_TR[7:0]または SAR_RANGE_TR[7:0]出力のレベルトリガーです。レベルトリガー出力の原因は TR_LEVEL_OUT レジスタで設定できます。選択された CSG_CMP_TR[7:0]または SAR_RANGE_TR[7:0]出力は OR され、関連するレベルトリガーアウトにルーティングされます。

  PSOC™ Control C3 MCU には 5 つの CSG コンパレーターがあるため、CSG_CMP_TR[4:0]ビットのみが有効であることに注意してください。

  HPPASS_MMIO_TR_LEVEL_OUT レジスタは 8 セットあります。つまり、ユーザーは最大 8 つのレベルトリガーを選択できます。

  HPPASS_MMIO_TR_LEVEL_CFG.BYPASS_SYNC[7:0]レジスタの設定により、トリガー動作を同期または非同期で出力するオプションがあります。

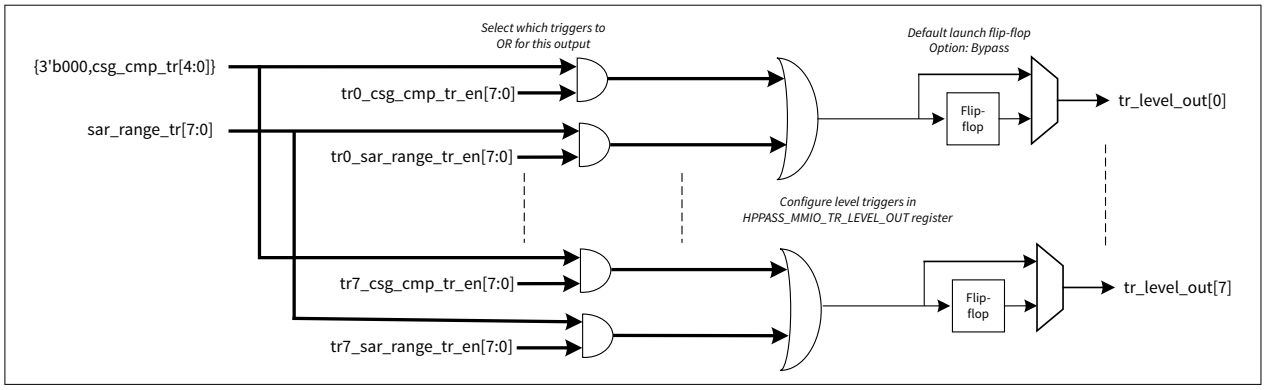


図 315　　出力レベルのトリガー ルーティング

- **tr_pulse_out[7:0]**は SAR_ENTRY_DONE[7:0]、SAR_FIR_DONE[1:0]、または AC コンディション完了のパルス トリガーです。TR_PULSE_OUT レジスタにより、パルストリガー出力の原因を選択できます。選択された出力は Mux され、関連するパルス トリガー出力にルーティングされます。

  HPPASS_MMIO_TR_PULSE_OUT レジスタは 8 セットあります。つまり、最大 8 つのパルス トリガーを選択できます。

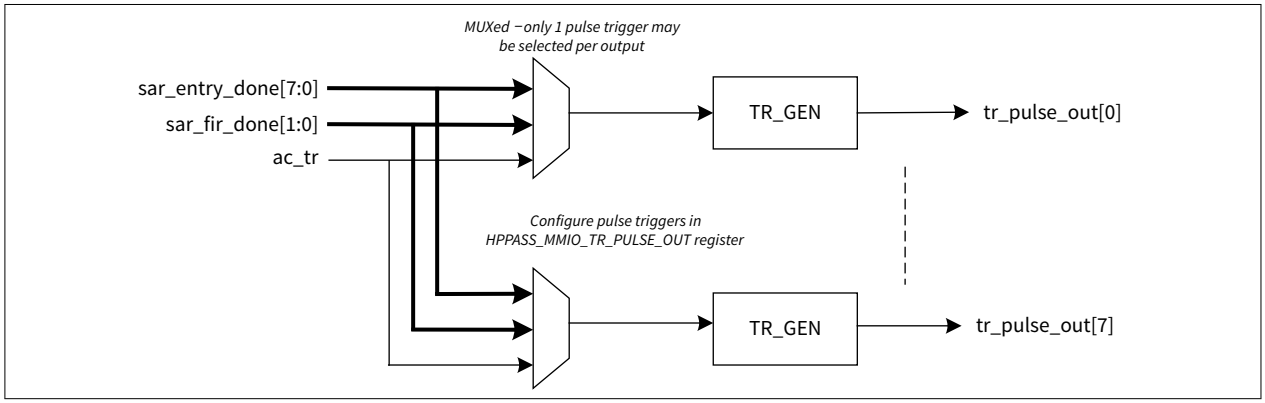


図 316　　出力パルストリガー ルーティング

27.4　クロック

HPPASS が必要とするクロックは clk_hf3 のみで、最大周波数は 240 MHz です。図 317 にクロックダイアグラムを示します。

SAR ADC コアは clk_hf3 クロックを直接使用し、SAR ADC デジタル部のクロックは AC で有効にされ clk_hf3 を 2 分周します。CSG クロックは、クロック ソースが clk_hf3 である周辺クロック分周器グループ 5 によって設定され、AC によって有効化されます。CSG クロックの周波数範囲は 10 MHz～120 MHz です。周辺クロック分周の詳細はクロック システムの章を参照してください。AC、INFRA クロックも clk_hf3 を 2 分周します。

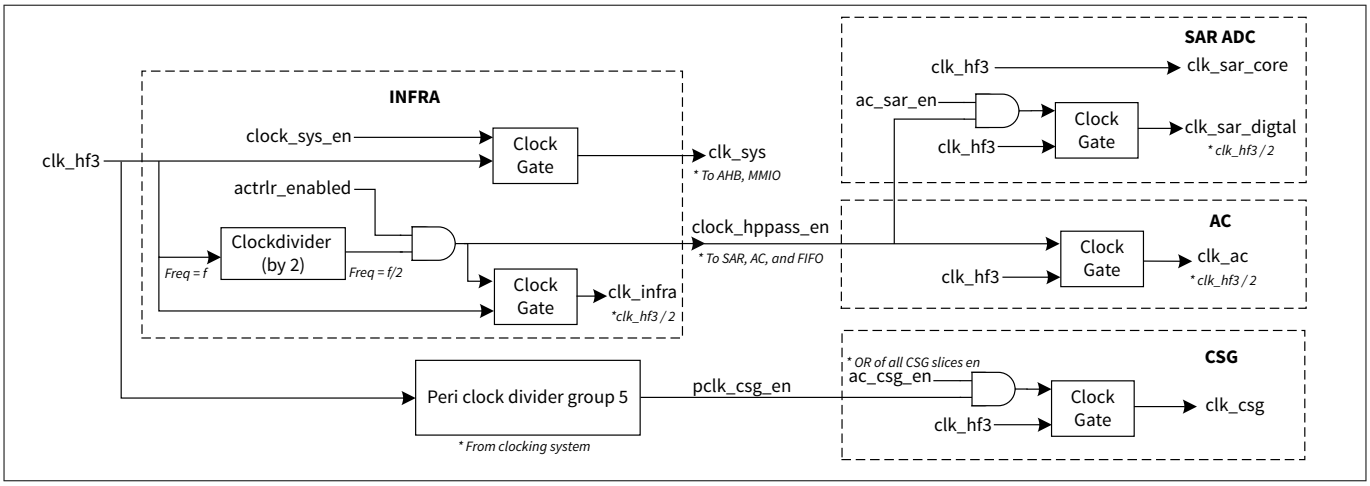


図 317　　HPPASS クロック図

27.5　レジスタの概要

表 257　　HPPASS SAR ADC レジスタの概要

レジスタ	機能
HPPASS_SARADC_CALOFFSTx	SAR ADC オフセット キャリブレーション値レジスタ、x = 0～3 の 4 つのオフセット キャリブレーション値
HPPASS_SARADC_CALLINx	SAR ADC 直線性キャリブレーション値レジスタ、x = 0 から 15 の 16 の直線性キャリブレーション値
HPPASS_SARADC_CALGAINC	SAR ADC 粗粒度ゲイン キャリブレーション値レジスタ
HPPASS_SARADC_CALGAINF	SAR ADC 細粒度ゲイン キャリブレーション値レジスタ
HPPASS_SAR_SEQ_ENTRYx	SAR ADC シーケンサー テーブル グループ レジスタ、x = 0～7 の 8 グループ
HPPASS_SAR_ENTRY_MUX_SEL_ALIASx	SAR ADC シーケンサー MUX セレクト エイリアス レジスタ、x = 0～7 の 8 本
HPPASS_SAR_CFG_CTRL	SAR ADC コントロール レジスタ
HPPASS_SAR_CFG_SAMP_EN	SAR ADC サンプラー スタティック イネーブル レジスタ
HPPASS_SAR_CFG_SAMP_GAIN	SAR ADC サンプラー ゲイン コンフィギュレーション レジスタ
HPPASS_SAR_CFG_SAMPLE_TIMEx	SAR ADC サンプル時間設定レジスタ、x = 0～2 の 3 サンプル時間
HPPASS_SAR_CFG_FIR_CFGx	FIR コンフィギュレーション レジスタ、x = 0, 1 の 2 つの FIR フィルター
HPPASS_SAR_CFG_CHAN_CFGx	SAR ADC チャネル構成レジスタ、28 チャネル、x = 0～27
HPPASS_SAR_CFG_RANGE_CFGx	SAR ADC 限界検出コンフィギュレーション レジスタ、x = 0～7 の 8 つの限界検出リソース
HPPASS_SAR_CFG_RANGE_LOWx	SAR ADC 限界検出低値レジスタ、x = 0～7 の 8 つの限界検出リソース

(続く)

表 257 (続き) HPPASS SAR ADC レジスタの概要

レジスタ	機能
HPPASS_SAR_CFG_RANGE_HIGHx	SAR ADC 限界検出高値レジスタ、x = 0〜7 の 8 つの限界検出リソース
HPPASS_SAR_CFG_CHAN_RESULTx	SAR ADC チャネル結果レジスタ、28 チャネル、x = 0〜27
HPPASS_SAR_CFG_FIR_RESULTx	FIR 結果レジスタ、x = 0, 1 の 2 つの FIR フィルター
HPPASS_SAR_CFG_RESULT_MASK	SAR ADC チャネル結果マスク レジスタ
HPPASS_SAR_CFG_RESULT_UPDATED	SAR ADC チャネル結果ステータス レジスタ
HPPASS_SAR_CFG_RANGE_STATUS	SAR ADC 限界検出ステータス レジスタ
HPPASS_SAR_CFG_SAR_STATUS	SAR ADC ステータス レジスタ
HPPASS_SAR_CFG_RESULT_OVERFLOW	SAR ADC オーバーフロー ステータス レジスタ
HPPASS_SAR_CFG_ENTRY_TR_COLLISION	SAR ADC グループ トリガー衝突ステータス レジスタ
HPPASS_SAR_CFG_ENTRY_HOLD_VIOLATION	SAR ADC グループ ホールド違反ステータス レジスタ
HPPASS_SAR_CF G_ENTRY_HOLD_CNT	SAR ADC グループ ホールド違反カウント レジスタ
HPPASS_SAR_CFG_SAR_RESULT_INTR	SAR ADC 割込みレジスタ
HPPASS_SAR_CFG_SAR_RESULT_INTR_SET	SAR ADC 割込み要求セット レジスタ
HPPASS_SAR_CFG_SAR_RESULT_INTR_MASK	SAR ADC 割込み要求マスク レジスタ
HPPASS_SAR_CFG_SAR_RESULT_INTR_MASKED	SAR ADC 割込み要求マスクド レジスタ
HPPASS_SAR_CFG_SAR_RANGE_INTR	SAR ADC 限界検出割込みレジスタ
HPPASS_SAR_CFG_SAR_RANGE_INTR_SET	SAR ADC 限界検出割込み要求セット レジスタ
HPPASS_SAR_CFG_SAR_RANGE_INTR_MASK	SAR ADC 限界検出割込み要求マスク レジスタ
HPPASS_SAR_CFG_SAR_RANGE_INTR_MASKED	SAR ADC 限界検出割込み要求マスクド レジスタ
HPPASS_SAR_CFG_CHAN_RESULT_PACKEDx	SAR ADC 結果パック レジスタ、x = 0〜13 の 14 のレジスタ
HPPASS_SAR_CFG_AROUTE_CTRL_MODE	AROUTE 制御モードレジスタ
HPPASS_SAR_CFG_AROUTE_FW_CTRL	AROUTE ファームウェア スイッチ制御セット レジスタ
HPPASS_SAR_CFG_AROUTE_FW_CTRL_CLR	AROUTE ファームウェア スイッチ制御クリアレジスタ
HPPASS_SAR_CFG_AROUTE_STATUS	AROUTE スイッチ ステータス レジスタ
HPPASS_SAR_CFG_TEMPSENSE_CTRL	温度センサー制御レジスタ
HPPASS_SAR_FIRx_FIR_COEFSy	FIR 係数レジスタ、2 つの 16 タップ FIR、x = 0、1、y = 0〜15
HPPASS_SAR_FIRx_FIR_TAPSy	FIR タップ ベール レジスタ、2 つの 16 タップ FIR、x = 0, 1、y = 0〜15

表 258 HPPASS CSG レジスタの概要

レジスタ	機能
HPPASS_CSG_SLICEx_CMP_CFG	CSG スライス比較器制御レジスタ、x = 0〜4 の 5 スライス

(続く)

表 258 (続き) HPPASS CSG レジスタの概要

レジスタ	機能
HPPASS_CSG_SLICEx_DAC_CFG	CSG スライス DAC 制御レジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_DAC_PARAM_SYNC	CSG スライス DAC パラメーター同期更新レジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_DAC_MODE_START	CSG スライス DAC モード開始レジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_DAC_VAL_A	CSG スライス DAC 値バッファ レジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_DAC_VAL_B	CSG スライス DAC 値 B バッファレジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_DAC_PERIOD	CSG スライス DAC 周期バッファ レジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_DAC_VAL	CSG スライス DAC ファームウェア書き込みレジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_DAC_STATUS	CSG スライス DAC 電流値レジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_SLICEx_CMP_STATUS	CSG スライス コンパレーター ステータス レジスタ、x = 0～4 の 5 スライス
HPPASS_CSG_LUT_CFGx_LUT_DATAy	DAC LUT 波形データ レジスタ、2 つの 128 エントリー LUT (x = 0、1、y = 0～127)
HPPASS_CSG_CSG_CTRL	CSG 制御レジスタ
HPPASS_CSG_DAC_INTR	DAC 割込みレジスタ
HPPASS_CSG_DAC_INTR_SET	DAC 割込み要求セット レジスタ
HPPASS_CSG_DAC_INTR_MASK	DAC 割込み要求マスク レジスタ
HPPASS_CSG_DAC_INTR_MASKED	DAC 割込み要求マスクド レジスタ
HPPASS_CSG_CMP_INTR	CSG コンパレーター割込みレジスタ
HPPASS_CSG_CMP_INTR_SET	CSG コンパレーター割込み要求セットレジスタ
HPPASS_CSG_CMP_INTR_MASK	CSG コンパレーター割込み要求マスク レジスタ
HPPASS_CSG_CMP_INTR_MASKED	CSG コンパレーター割込み要求マスクド レジスタ
HPPASS_CSG_VDAC_OUT_BLANK	DAC 観測可能レジスタの CSG ブランキング制御
HPPASS_CSG_CSG_SYNC	CSG 同期イネーブル レジスタ

表 259 HPPASS AC レジスタの概要

レジスタ	機能
HPPASS_ACTRLR_CTRL	AC 制御レジスタ
HPPASS_ACTRLR_CMD_RUN	AC ランコマンドレジスタ
HPPASS_ACTRLR_CMD_STATE	AC ステート アドレス レジスタ
HPPASS_ACTRLR_BLOCK_STATUS	HPPASS ブロック Ready ステータス レジスタ
HPPASS_ACTRLR_STATUS	AC ステータスレジスタ
HPPASS_ACTRLR_CFG	AC コンフィギュレーション レジスタ

(続く)

表 259 (続き) HPPASS AC レジスタの概要

レジスタ	機能
HPPASS_ACTRLR_CNTR_STATUSx	AC ループ/インターバル カウンター ステータス レジスタ、x = 0, 1 の 2 つのカウンター
HPPASS_ACTRLR_TTCFGx_TT_CFG0	AC コンフィギュレーション 0 レジスタ、x = 0〜15 の 16 ステート
HPPASS_ACTRLR_TTCFGx_TT_CFG1	AC コンフィギュレーション 1 レジスタ、x = 0〜15 の 16 ステート
HPPASS_ACTRLR_TTCFGx_TT_CFG2	AC コンフィギュレーション 2 レジスタ、x = 0〜15 の 16 ステート
HPPASS_ACTRLR_TTCFGx_TT_CFG3	AC コンフィギュレーション 3 レジスタ、x = 0〜15 の 16 ステート

表 260 HPPASS INFRA レジスタの概要

レジスタ	機能
HPPASS_INFRA_TR_IN_SEL	HPPASS トリガー入力選択レジスタ
HPPASS_INFRA_HW_TR_MODE	HPPASS ハードウェア トリガー モード レジスタ
HPPASS_INFRA_FW_TR_PULSE	HPPASS ファームウェア パルス モード トリガー レジスタ
HPPASS_INFRA_FW_TR_LEVEL	HPPASS ファームウェア レベル モード トリガー レジスタ
HPPASS_INFRA_CLOCK_STARTUP_DIV	HPPASS スタートアップクロック分周レジスタ
HPPASS_INFRA_STARTUP_CFGx	HPPASS スタートアップ設定レジスタ、x = 0〜3 で 4 つ。
HPPASS_INFRA_VDDA_STATUS	アナログ電圧ステータス レジスタ
HPPASS_INFRA_ANA_CTRL	アナログ制御レジスタ
HPPASS_INFRA_AREFv2_AREF_CTRL	AREF 制御レジスタ

表 261 HPPASS MMIO レジスタの概要

レジスタ	機能
HPPASS_MMIO_FIFO_INTR	FIFO 割込み要求レジスタ
HPPASS_MMIO_FIFO_INTR_SET	FIFO 割込み要求セット レジスタ
HPPASS_MMIO_FIFO_INTR_MASK	FIFO 割込みマスクレジスタ
HPPASS_MMIO_FIFO_INTR_MASKED	FIFO 割込み要求マスク レジスタ
HPPASS_MMIO_HPPASS_INTR	HPPASS 割込み要求レジスタ
HPPASS_MMIO_HPPASS_INTR_SET	HPPASS 割込み要求セット レジスタ
HPPASS_MMIO_HPPASS_INTR_MASK	HPPASS 割込みマスクレジスタ
HPPASS_MMIO_HPPASS_INTR_MASKED	HPPASS 割込み要求マスク レジスタ
HPPASS_MMIO_TR_LEVEL_CFG	HPPASS レベル トリガー出力設定レジスタ

(続く)

表 261 (続き) HPPASS MMIO レジスタの概要

レジスタ	機能
HPPASS_MMIO_TR_LEVEL_OUTx	HPPASS レベル トリガー出力セレクト レジスタ、x = 0～7 の 8 出力レベル トリガー
HPPASS_MMIO_TR_PULSE_OUTx	HPPASS パルス トリガー出力セレクト レジスタ、x = 0～7 の 8 出力パルス トリガー
HPPASS_MMIO_FIFO_CFG	FIFO コンフィギュレーション レジスタ
HPPASS_MMIO_FIFO_CLR	FIFO ファームウェア クリア レジスタ
HPPASS_MMIO_FIFO_LEVELx	FIFO レベルレジスタ、x = 0～3 の 4 つの FIFO
HPPASS_MMIO_FIFO_RD_DATAx	FIFO 読み出しデータ レジスタ、x = 0～3 の 4 つの FIFO
HPPASS_MMIO_FIFO_USEDx	FIFO 使用レジスタ、x = 0～3 の 4 つの FIFO
HPPASS_MMIO_FIFO_STATUSx	FIFO ステータス レジスタ、x = 0～3 の 4 つの FIFO

28 低消費電力コンパレーター (LPCOMP)

PSOC™ Control C3 MCU には 2 つの低消費電力コンパレーターがあり、すべてのシステム電力モードでアナログ信号の比較を実行できます。低消費電力コンパレーター出力は、次のことを実行します。

- CPU による検査
- CPU スリープモード時に CPU への割込み/ウェイクアップソースとしての使用
- システム ディープスリープまたはハイバネート モード時にシステム リソースへのウェイクアップ ソースとして使用されるか、非同期または同期信号 (レベルまたはパルス) として GPIO またはトリガー マルチプレクサ (トリガー マルチプレクサ (トリガー MUX) の章を参照) に供給されます。

28.1 特長

PSOC™ Control C3 MCU の低消費電力コンパレーターには、次のような特長があります。

- プログラム可能な消費電力および動作速度
- 超低消費電力モード対応
- 各コンパレーターは、片側ヒステリシス オプションあり
- コンパレーター出力の立ち上りエッジ、立ち下りエッジ、および立ち上りと立ち下りの複合エッジ検出
- ローカル基準電圧生成
- デバイスを低消費電力モードからウェイクアップ

28.2 アーキテクチャ

次の図に、低消費電力コンパレーターのブロック図を示します。

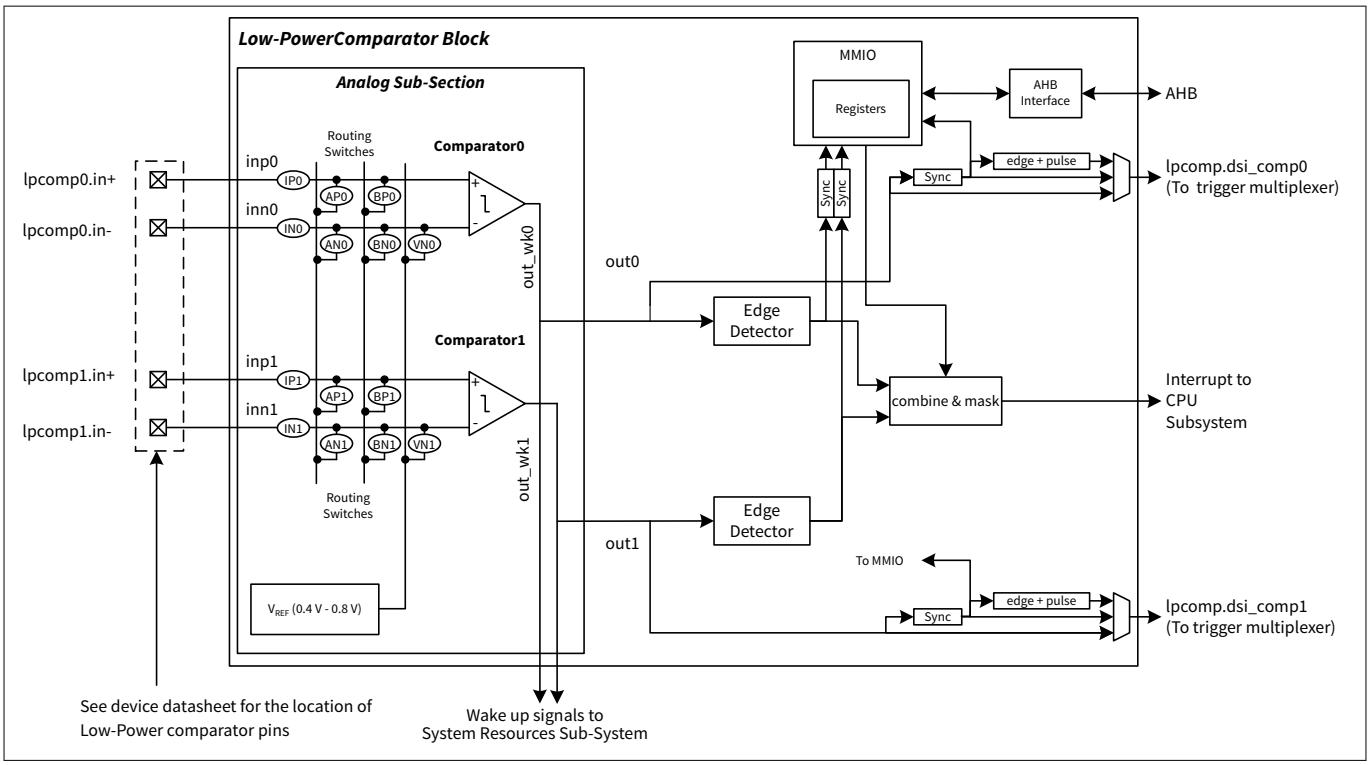


図 318 低消費電力コンパレーター ブロック図

このブロックは、より高いレベルでは、アナログ ロジックとデジタル ロジックの 2 つのサブセクションで構成されています。アナログ サブセクションは、ヒステリシスを選択できる 2 つのコンパレーター、入力への接続を行うスイッ

チ、基準電圧源で構成されています。コンパレーターからの出力は、ハイバネート モードからのデバイス ウェイクアップのためにシステム リソース サブシステム (SRSS) に直接駆動されます。

コンパレーター出力はデジタル サブセクションにもドライブされます。このサブセクションでは、同期レベル出力やパルス出力などのさまざまな出力オプションや、スリープモードやディープスリープモードからデバイスをウェイクアップさせるために、コンパレーター出力の正、負または両方のエッジで割込みをトリガーするオプションを提供します。

以下のセクションでは、入力コンフィギュレーション、電力および速度モード、出力および割込みコンフィギュレーション、ヒステリシス、および低消費電力モードからのウェイクアップを含む、PSOC™ Control C3 MCU 低消費電力コンパレーターの動作について説明します。

28.2.1 入力コンフィギュレーション

低消費電力コンパレーターは、以下の入力オプションで動作します。

- 外部ピンからの 2 つの電圧を比較します
- 外部信号をローカルに生成された基準電圧と比較します。この電圧は精密基準ではないので、0.4 V から 0.8 V の範囲で変動することに注意してください。

図 318 に示す内部配線スイッチ (IP0、AP0、BP0、IN0、AN0、BN0、VN0、IP1、AP1、BP1、IN1、AN1、BN1、VN1) は、LPCOMP_CMP0_SW、LPCOMP_CMP1_SW、LPCOMP_CMP0_SW_CLEAR、LPCOMP_CMP1_SW_CLEAR レジスタを使って制御します。これらのスイッチの制御方法については、レジスタ リファレンス マニュアルを参照してください。

コンパレーターの反転入力がローカル電圧基準に配線されている場合、LPCOMP_CONFIG レジスタの LPREF_EN ビットを設定し、電圧基準を有効にします。これらのレジスタの詳細については、レジスタ リファレンス マニュアルを参照してください。

28.2.2 出力と割込みの設定

出力は、コントロール レジスタ (LPCOMP_CMP0_CTRL および LPCOMP_CMP1_CTRL) を使用して、以下のオプションのいずれかに設定できます。

- ダイレクト コンパレーター出力
- 同期コンパレーター出力 - レベル
- 同期コンパレーター出力 - パルス

コントロール レジスタの DSI_BYPASS0 と DSI_BYPASS1 ビットは、ダイレクト コンパレーター出力か同期コンパレーター出力を選択します。

制御レジスタの DSI_LEVEL0 と DSI_LEVEL1 ビットはパルス出力かレベル出力かを選択します。パルス出力は clk_sys の 2 サイクルで、コントロール レジスタの INTTYPE0 と INTTYPE1 ビットを使って設定されたプラス エッジ、マイナス エッジ、または両方のエッジで発生させられます。表 262 にコンフィギュレーションの要約を示します。

表 262 出力コンフィギュレーション

DSI_BYPASSx	DSI_LEVELx	出力
1	x	ダイレクト コンパレーター出力
0	1	同期コンパレーター出力 - レベル
0	0	同期コンパレーター出力 - パルス

さらに、コンパレーター出力は、トリガー マルチプレクサを使用して、GPIO や、DMA や TCPWM などの他のオンチップ ペリフェラルにルーティングできます。詳細についてはトリガー マルチプレクサ (トリガー MUX) の章を参照してください。

コンパレーター 0 とコンパレーター 1 の出力のファームウェア読み出しは、LPCOMP_STATUS レジスタの OUT0 ビットと OUT1 ビットで利用可能です (表 263)。各コンパレーターの出力は、対応するエッジ検出ブロックに接続されています。このブロックは、割込みをトリガーするエッジを決定します。エッジ選択と割込みイネーブルは、それぞれコンパレーター 0 とコンパレーター 1 の LPCOMP_CMP0_CTRL と LPCOMP_CMP1_CTRL レジスタの INTTYPE0 と INTTYPE1 ビットフィールドを使用して設定されます。INTTYPE0 ビットと INTTYPE1 ビットを使用して、表 263 に示すように、割込みタイプを無効、立ち上りエッジ、立ち下りエッジ、または両方のエッジに選択できます。

エッジ イベントの間、コンパレーターは割込みをトリガーします。割込み要求は、コンパレーター 0 とコンパレーター 1 それぞれに、LPCOMP_INTR レジスタの COMP0 ビット[0]と COMP1 ビット[1]に保存されます。コンパレーター 0 とコンパレーター 1 の両方は共通の割込み信号出力 (表 263 を参照) を共有しており、これは 2 つの割込みの論理和で、CPU NVIC の低消費電力コンパレーターブロックの割込みとしてマッピングされています。詳しくは、割込みのセクションを参照してください。設計で両方のコンパレーターが使用されている場合は、割込みサービスルーチンで LPCOMP_INTR レジスタの COMP0 ビットと COMP1 ビットを読み出し、どちらが割込みをトリガーしたかを確認します。あるいは、LPCOMP_INTR_MASK レジスタの COMP0_MASK ビットと COMP1_MASK ビットを使用して、CPU へのコンパレーター 0 とコンパレーター 1 の割込みをマスクすることもできます。マスクされた割込みのみが CPU によって処理されます。割込みが処理された後、ファームウェアの LPCOMP_INTR レジスタの COMP0 ビットと COMP1 ビットに'1'を書き込むことで、割込みをクリアします。CPU への割込みがクリアされない場合、次のコンペアイベントに関係なくアクティブとなり、CPU に連続的に割込みをかけます。詳細は、割込みの章を参照してください。

LPCOMP_INTR_SET レジスタのビット[1:0]は、ファームウェア デバッグ用の割込みをアサートするために使用できます。

ディープスリープ モードでは、コンパレーターのエッジイベントによってウェイクアップ割込みコントローラー (WIC) が有効化され、CPU がウェイクアップされます。同様に、ハイバネートモードでは、LPCOMP はクロックを含むシステム リソース サブシステム (SRSS) をウェイクアップできます。このように、LPCOMP は低消費電力モードで指定された信号をモニターできます。詳細は、電源と電圧監視の章とデバイス電力モードの章を参照してください。

表 263　　出力と割込みの設定

レジスタ[Bit_Pos]	Bit_Name	説明
LPCOMP_STATUS[0]	OUT0	コンパレーター 0 の電流/瞬時出力値
LPCOMP_STATUS[16]	OUT1	コンパレーター 1 の電流/瞬時出力値
LPCOMP_CMP0_CTRL[7:6]	INTTYPE0	コンパレーター 0 が IRQ をトリガーするエッジを設定 00: 無効 01: 立ち上りエッジ 10: 立ち下りエッジ 11: 立ち上りと立ち下りエッジ
LPCOMP_CMP1_CTRL[7:6]	INTTYPE1	コンパレーター 1 が IRQ をトリガーするエッジを設定 00: 無効 01: 立ち上りエッジ 10: 立ち下りエッジ 11: 立ち上りと立ち下りエッジ
LPCOMP_INTR[0]	COMP0	コンパレーター 0 割込み: コンパレーター 0 がトリガーしたときにハードウェアがこの割込みを設定します。 '1'を書き込んで、割込みをクリアしてください。

(続く)

表 263 (続き) 出力と割込みの設定

レジスタ[Bit_Pos]	Bit_Name	説明
LPCOMP_INTR[1]	COMP1	コンパレーター 1 割込み: コンパレーター 1 がトリガーしたときにハードウェアがこの割込みを設定します。 '1'を書き込んで、割込みをクリアしてください。
LPCOMP_INTR_SET[0]	COMP0	'1'の書き込みにより、コンパレーター 0 のソフトウェア割込みをトリガーしてください。
LPCOMP_INTR_SET[1]	COMP1	'1'の書き込みにより、コンパレーター 1 のソフトウェア割込みをトリガーしてください。
LPCOMP_INTR_MASK[0]	COMP0_MASK	コンパレーター 0 割込みを有効にするためには'1'を書き込んでください
LPCOMP_INTR_MASK[1]	COMP1_MASK	コンパレーター 1 割込みを有効にするためには'1'を書き込んでください
LPCOMP_INTR_MASKED[0]	COMP0_MASKED	1 の場合、コンパレーター 0 割込みがトリガーされたことを示します
LPCOMP_INTR_MASKED[1]	COMP1_MASKED	1 の場合、コンパレーター 1 割込みがトリガーされたことを示します

28.2.3 電力モードおよび動作速度のコンフィギュレーション

低消費電力コンパレーターは次の 3 つのモードで動作可能です。

- ノーマル
- 低消費電力
- 超低消費電力

コンパレーター 0/1 のパワーまたはスピード設定は、LPCOMP_CMPx_CTRL レジスタの MODE0/1 ビットフィールドを使用して設定されます。消費電力と応答時間は、選択された電力モードによって異なります。消費電力は高速モードで最も高く、超低消費電力モードで最も低く、応答時間は高速モードで最も速く、超低消費電力モードで最も遅いです。様々な電源設定における応答時間と消費電力の仕様については、デバイスのデータシートを参照してください。

表 264 で説明されているように、これらのビットフィールドを使用してコンパレーターを有効または無効にすることもできます。

注: コンパレーターが有効の状態で電力モードが変更されると、コンパレーターの出力がグリッチすることがあります。これを避けるには、電力モードを変更する前にコンパレーターを無効にします。

表 264　　コンパレーター電力モード選択ビット

レジスタ[Bit_Pos]	Bit_Name	説明
LPCOMP_CMP0_CTRL[1:0]	MODE0	コンパレーター 0 の電力モード選択 00: オフ 01: 超低消費電力動作モード。このモードは、デバイスがディープスリープまたはハイバネート モードのときに使用します。 10: 低消費電力動作モード 11: ノーマル、フルスピード、フルパワー動作モード 各電力モードにおける電気的仕様については、データシートを参照してください。
LPCOMP_CMP1_CTRL[1:0]	MODE1	コンパレーター 1 の電力モード選択 00: オフ 01: 超低消費電力動作モード。このモードは、デバイスがディープスリープまたはハイバネート モードのときに使用します。 10: 低消費電力動作モード 11: ノーマル、フルスピード、フルパワー動作モード 各電力モードにおける電気的仕様については、データシートを参照してください。

さらに、LPCOMP_CONFIG[31]ビットを使用して、低消費電力コンパレーター システム全体をグローバルに有効または無効にできます。これらのビットフィールドの詳細については、レジスタ リファレンス マニュアルを参照してください。

28.2.4　　ヒステリシス

信号同士が近く、かつ変化が遅い信号を比較するアプリケーションで、ヒステリシスは信号にノイズがある時にコンパレーター出力が振動することを回避するのに役立ちます。このようなアプリケーションでは、コンパレーター ブロックで固定ヒステリシスの有効化ができる可能性があります。ヒステリシス電圧範囲については、デバイスのデータシートを参照してください。

ヒステリシス レベルの有効/無効は、表 265 で説明するように、コンパレーター 0 とコンパレーター 1 の LPCOMP_CMP0_CTRL と LPCOMP_CMP1_CTRL レジスタの HYST0 と HYST1 のビットフィールドを使用します。

表 265　　ヒステリシス制御ビット

レジスタ[Bit_Pos]	Bit_Name	説明
LPCOMP_CMP0_CTRL[5]	HYST0	コンパレーター 0 へのヒステリシスの有効化/無効化 1: ヒステリシスを適用 0: ヒステリシスなし ヒステリシス電圧範囲についてはデータシートを参照してください。
LPCOMP_CMP1_CTRL[5]	HYST1	コンパレーター 1 へのヒステリシスの有効化/無効化 1: ヒステリシスを適用 0: ヒステリシスなし ヒステリシス電圧範囲についてはデータシートを参照してください。

28.2.5 低消費電力モードからウェイクアップ

コンパレーターは、スリープ、ディープスリープおよびハイバネートのモードを含むデバイスの低消費電力モードで動作します。コンパレーター出力は、スリープモードおよびディープスリープモードからデバイスをウェイクアップできます。コンパレーター出力は、LPCOMP_CONFIG レジスタで有効にすると、ディープスリープ モードで割込みを発生させられます。LPCOMP_CMPx_CTRL レジスタの INTTYPEx ビットを無効に設定せず、対応するコンパレーターの LPCOMP_INTR_MASK レジスタの INTR_MASKx ビットを設定して、デバイスを低消費電力モードからウェイクアップします。

ディープスリープおよびハイバネート モードで動作する場合、コンパレーターには以下の制限があります。

- コンパレーターは超低消費電力モードで動作させてください
- コンパレーターへの入力は専用ピンから行います。内部リファレンス VREF が使用可能で、コンパレーターの反転入力に接続できます。

注: LPCOMP_CONFIG レジスタの LPREF_EN ビットを'1'に設定し、ディープスリープ モードとハイバネート モードでこの機能ブロックを使用します。このビットは、ローカル基準電圧 VREF と、低消費電力モードでの機能ブロックの動作に必要なバイアス電流を制御します。

28.2.6 コンパレーター クロック

コンパレーターは、割込みおよび出力同期用に CLK_HF[1]から供給されるクロックとして、システムメインクロック clk_sys を使用します。詳細についてはクロック システムの章を参照してください。

28.2.7 低消費電力コンパレーターの使用

コンパレーターを設定するためには、以下の手順に従ってください。

1. 内部ルーティング スイッチを設定し、入力をコンパレーターに接続してください。内部ルーティング スイッチの制御方法については、レジスタ リファレンス マニュアルの LPCOMP_CMP0_SW、LPCOMP _CMP1_SW、LPCOMP_CMP0_SW_CLE AR、LPCOMP_CMP1_SW_CLEAR レジスタを参照してください。
2. LPCOMP_CONFIG レジスタの ENABLED ビットに書き込んで LPCOMP ブロックを有効にしてください。
3. LPCOMP_CMP0_CTRL または LPCOMP_CMP1_CTRL レジスタを使用して、ヒステリシスと出力タイプ (ダイレクト、同期レベル、同期パルス) を設定してください。
4. 内部リファレンス VREF が使用され、反転入力に接続されている場合は、LPCOMP_CONFIG レジスタの LPREF_EN ビットに書き込んで有効にしてください。
5. LPCOMP_CMP0_CTRL または LPCOMP_CMP1_CTRL レジスタのモード ビットに書き込むことにより、コンパレーターの電力モードを設定してください。ディープスリープ モードやハイバネート モード中に動作が必要な場合は、必ず電力モードが超低消費電力に設定しお、LPCOMP_CONFIG レジスタの LPREF_EN ビットが設定してください。
6. 次のステップで出力をルーティングしたり割込みを有効にする前に、コンパレーターが安定するように遅延を設けてください。遅延時間は、設定された電力モードに依存します。遅延数については、デバイスデータシートを参照してください。
7. トリガー マルチプレクサを設定することにより、出力を GPIO または他のペリフェラルにルーティングしてください。
8. 割込みが必要な場合は、LPCOMP_CMP0_CTRL または LPCOMP_CMP1_CTRL レジスタに書き込んで割込みタイプを設定し、LPCOMP_INTR_MASK レジスタに書き込んで割込みを有効にしてください。

28.3 レジスタの概要

表 266 低消費電力コンパレーター レジスタの概要

レジスタ	機能
LPCOMP_CONFIG	LPCOMP グローバル コンフィギュレーション レジスタ
LPCOMP_INTR	LPCOMP 割込みレジスタ
LPCOMP_INTR_SET	LPCOMP 割込みセット レジスタ
LPCOMP_INTR_MASK	LPCOMP 割込み要求マスク レジスタ
LPCOMP_INTR_MASKED	LPCOMP マスクされた割込み出力レジスタ
LPCOMP_STATUS	出力ステータス レジスタ
LPCOMP_CMP0_CTRL	コンパレーター 0 コンフィギュレーション レジスタ
LPCOMP_CMP1_CTRL	コンパレーター 1 コンフィギュレーション レジスタ
LPCOMP_CMP0_SW	コンパレーター 0 スイッチ制御レジスタ
LPCOMP_CMP1_SW	コンパレーター 1 スイッチ制御レジスタ
LPCOMP_CMP0_SW_CLEAR	コンパレーター 0 スイッチ制御クリアレジスタ
LPCOMP_CMP1_SW_CLEAR	コンパレーター 1 スイッチ制御クリアレジスタ

改訂履歴

版数	発行日	変更内容
-	-	英語版 002-39348 Rev. **には日本語版はありません。
-	-	• 英語版 002-39348 Rev. *A には日本語版はありません。英語版の改訂内容: Added these sections: - バス インフラストラクチャー - プロセッサ間通信 (IPC) - フラッシュ - ダイレクトメモリ アクセス (DMA - DW) - トリガー マルチプレクサ (トリガー MUX) - プログラムおよびデバッグ インターフェース - 暗号化アクセラレータ - デバイス電力モード • Updated 図 1 • Updated content across the entire document
-	-	• 英語版 002-39348 Rev. *B には日本語版はありません。英語版の改訂内容: Updated 表 3 • Updated 表 4 • Updated 図 2 • Updated 図 41 • Updated 図 317 • Updated エンベデッド クロス トリガー • Updated レジスタの概要 • Updated レジスタの概要 • Updated UART 動作モード • Updated ランダム クロック • Updated 特長 • Updated インフラストラクチャー (INFRA) • Updated SAR ADC • Updated 入力コンフィギュレーション • Updated 出力と割込みの設定

改訂履歴

版数	発行日	変更内容
**	2025-12-10	これは英語版 002-39348 Rev. *C を翻訳した日本語版 Rev. **です。英語版の改訂内容: Updated the following chapters: • はじめに • CPU サブシステム (CPUSS) • フラッシュ • TrustZone • 保護ユニット • セキュア ブート • 電源と電圧監視 • デバイス電力モード • クロック システム • リセット システム • I/O サブシステム • タイマー/カウンター/パルス幅変調器 (TCPWM) • シリアル通信ブロック (SCB) • 高性能プログラマブル アナログ サブシステム (HPPASS) • 低消費電力コンパレーター (LPCOMP)

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-12-10

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-afu1678784865156

重要事項

本文書に記載された情報は、いかなる場合も、条件または特性の保証とみなされるものではありません（「品質の保証」）。

本文に記された一切の事例、手引き、もしくは一般的価値、および／または本製品の用途に関する一切の情報に関し、インフィニオンテクノロジーズ（以下、「インフィニオン」）はここに、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

さらに、本文書に記載された一切の情報は、お客様の用途におけるお客様の製品およびインフィニオン製品の一切の使用に関し、本文書に記載された義務ならびに一切の関連する法的要件、規範、および基準をお客様が遵守することを条件としています。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。