TRAVEO™ T2G ファミリ スマート I/O の使用方法

本書について

適用範囲と目的

AN220203 は、TRAVEO™ T2G ファミリ MCU で Smart I/O (スマート I/O)を使用する方法を説明します。スマート I/O は周辺機能と GPIO ポートの間にプログラム可能な論理回路を追加し、それによりボード上のグルーロジックを組み込みます。

関連製品ファミリ

TRAVEO T2G ファミリ CYT2/CYT3/CYT4/CYT6 シリーズ

目次

本書について 1
目次 2
1 はじめに 3
1.1 スマート I/O のアプリケーション 3
1.2 スマート I/O のバイパス 4
2 スマート I/O の構造 5
2.1 クロックとリセット 6
2.2 同期化 7
2.3 3 入力のルックアップテーブル (LUT3 [x]) 7
2.3.1 LUT3 [x]出力設定 8
2.3.2 LUT3 [x]入力の選択 8
2.3.3 LUT3 [x]動作 12
2.4 データユニット (DU) 13
2.4.1 入力選択 14
2.4.2 データユニットの動作 15
3 スマート I/O の設定 27
4 設定例 28
4.1 極性反転による I/O ピンから HSIOM へのルーティングの変更のユースケース 28
4.1.1 設定とサンプルコード 31
4.2 リセット検出/安定回路のユースケース 42
4.2.1 設定とサンプルコード 48
5 用語集 63
6 参考資料 64
7 その他の参考資料 65
改訂履歴 66
免責事項 67

1 はじめに

このアプリケーションノートは、TRAVEO™ T2G ファミリ CYT2/CYT3/CYT4/CYT6 シリーズ MCU のスマート I/O の使用方法を説明します。

スマート I/O は I/O Port (I/O ポート) にプログラマブルロジックを追加します。スマート I/O は AND, OR や XOR などのブール論理機能を組み込みます。また高速 I/O マトリクス (HSIOM) と I/O ポート間の信号の前処理または後処理を行います。例えば、スマート I/O は CPU の介在なしに、複数のフリップフロップを使用して入力信号にデジタルグルーロジックを有効にできます。HSIOM はユーザーが選択した周辺機能に複数の機能を共有する GPIO を多重化します。HSIOM の詳細については、Architecture Reference Manual を参照してください。

このアプリケーションノートで使用する機能と用語については、Architecture Reference Manual の Smart I/O 章を参照してください。図 1 に一般的な信号パスの例を示します。

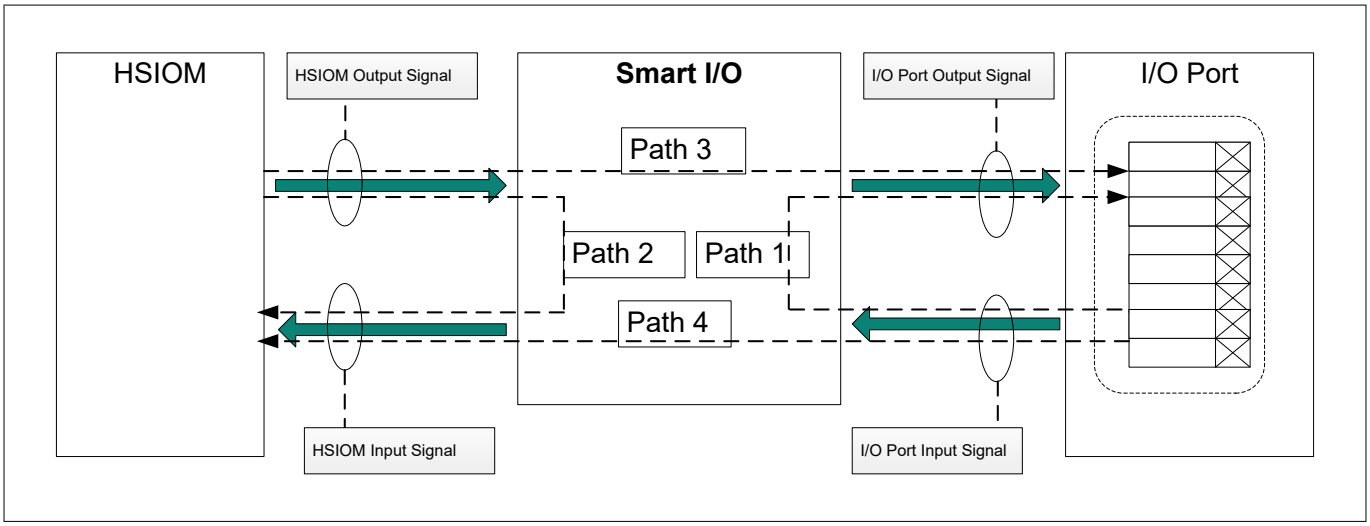


図 1 スマート I/O インタフェース

パス 1: I/O ポート信号で直接動作する独立した論理回路の実装

パス 2: HSIOM 信号で直接動作する独立した論理回路の実装

パス 3: HSIOM 出力信号を論理変換して I/O ポートにルーティング

パス 4: I/O ポート入力信号を論理変換して HSIOM にルーティング

各信号パスについて、スマート I/O 機能にはプログラム可能な出力のオプションがあります。このアプリケーションノートではスマート I/O 機能の使用例と設定例を示します。

1.1 スマート I/O のアプリケーション

スマート I/O は I/O ピンの入出力信号に簡単な論理演算またはルーティングが必要な場合に使用できます。以下は一般的なアプリケーションを示します。

- **ピンからピンへのルーティング変更:** この機能は固定された周辺機能から同一ポートにある非専用ピンへ信号を再ルーティングできます。
- **信号の極性反転:** この機能は SPI 信号のような出力信号の極性をピンから出力前に反転させます。
- **クロックまたは信号バッファ:** この機能は 2 つの GPIO バッファを通して 1 つのピンに負荷の大きい GPIO 入力信号を駆動します。
- **ピンのパターンの検出:** この機能はいくつの信号入力のパターンを検出し、結果に応じてプログラマブル信号を出力します。

これらのスマート I/O アプリケーションは低電力モード (ディープスリープ) で動作可能なため、ウェイクアップ割込みとして使用できます。

1.2 スマート I/O のバイパス

スマート I/O 機能を使用しない場合、SMARTIO_PRTx_CTL.ENABLE[1]ビットを“0” (ディセーブル) に設定することによって、スマート I/O は自動的にバイパスされます。また、SMARTIO_PRTx_CTL.BYPASS ビットを使用して、ポートグループの I/O ピンをバイパスできます。BYPASS ビットを “1” (バイパス) に設定すると、HSIOM と I/O ポートは直接接続されます。

スマート I/O を有効にする前に、バイパスを設定する必要があります。(SMARTIO_PRTx_CTL.ENABLE ビットを“1” (イネーブル) に設定)

表 1 に、バイパス設定の SMARTIO_PRTx_CTL レジスタを示します。詳細については、Registers Reference Manual を参照してください。

表 1 バイパス設定レジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_CTL	BYPASS[7:0]	スマート I/O のバイパス “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)
	ENABLED [31]	スマート I/O のイネーブル 0: ディセーブル (信号はバイパスされます: 初期値) 1: イネーブル (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)

1 この文章で使用されるレジスタ名のサブスクリプション x はポート番号です。

2 スマート I/O の構造

図 2 にスマート I/O のブロックダイヤグラムを示します。スマート I/O は HSIOM と I/O ポート間の信号パスに配置されています。

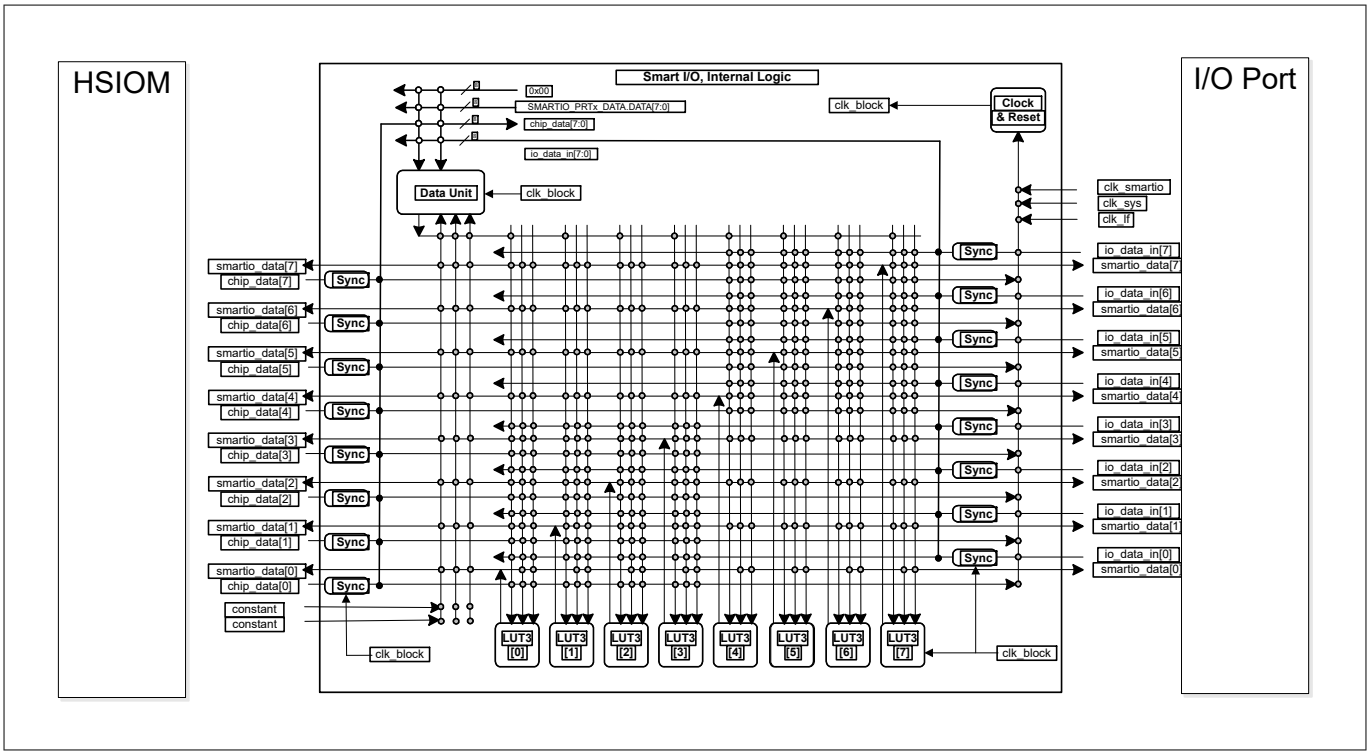


図 2 スマート I/O のブロックダイヤグラム

スマート I/O は以下のコンポーネントで構成されています。

- クロック (Clock) とリセット (Reset)
- 同期化 (Sync)
- 3 入力のルックアップテーブル (LUT3 [x]): x = 0〜7
- データユニット (Data Unit)

指定された I/O セルにスマート I/O が実装されています。スマート I/O はこれらのコンポーネントを組み合わせて HSIOM と I/O ポートにプログラマブル信号を提供します。スマート I/O が使用できる I/O ポートについては Device Datasheet の Package Pin List and Alternate Functions を参照してください。

io_data_in [7:0] は I/O ポートからの入力信号で、chip_data [7:0] は HSIOM からの入力信号です。これらの信号は同期コンポーネント (Sync) を介してスマート I/O に入力されます。smartio_data [7:0] はスマート I/O からの出力信号です。これらの信号はスマート I/O によってルーティングまたは論理変更され、HSIOM または I/O ポートに出力されます。

clk_block はスマート I/O のすべてのコンポーネントに使用されます。clk_block は I/O ポート入力信号 (io_data_in [7:0]), HSIOM 入力信号 (chip_data [7:0]), clk_smartio と clk_lf から選択できます。clk_smartio は周辺クロック分周器を用いてシステムクロック (clk_sys/CLK_HF) から生成され、クロックとリセットブロックに入力されます。clk_smartio と clk_lf の詳細については Architecture Reference Manual の Clocking system の章を参照してください。

スマート I/O ユニットごとに 8 つのルックアップテーブル (LUT3 [x]) があります。LUT3 [x]はプログラマブル信号を提供し、HSIOM と I/O ポート間の信号接続を決定します。つまり 8 つのルックアップテーブルは、入力チャネルと出力の柔軟なルーティングの組合せを提供します。

2 スマート I/O の構造

データユニットは出力信号により複雑な機能を提供します。クロックとリセットブロックは HSIOM, I/O ポート, およびスマート I/O の各ブロックの信号を同期するために使用されます。同期化は HSIOM 入力と I/O ポート入力の同期/非同期を制御します。

2.1 クロックとリセット

スマート I/O はリセット信号とクロック選択機能を持ちます。図 3 にクロックとリセット選択回路を示します。

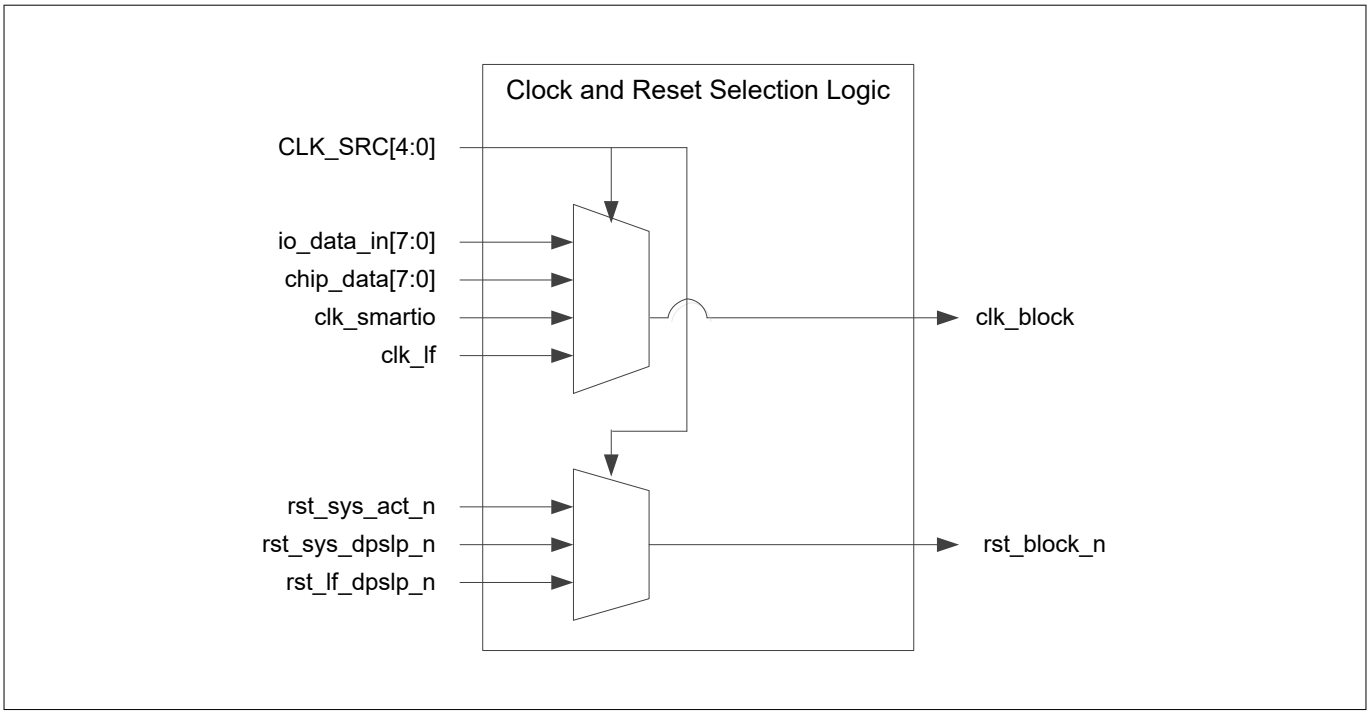


図 3 クロックとリセット設定の機能イメージ

クロックソースとして io_data_in [7:0] と chip_data [7:0] が選択される場合、クロックに関連するリセットはありません。クロックソースとして clk_smartio が選択される場合、パワーモード (アクティブまたはディープスリープ) に応じて rst_sys_act_n または rst_sys_dpslp_n が使用できます。クロック ソースとして clk_lf が選択される場合、rst_lf_dpslp_n が使用できます。クロック(clk_block) とリセット(rst_block_n) は SMARTIO_PRTx_CTL.CLOCK_SRC [12:8]レジスタで設定できます。

選択できるクロックソースを以下に示します。

- io_data_in [7:0]: I/O ポート入力信号です。
- chip_data [7:0]: HSIOM 入力信号です。
- clk_smartio : このクロックはシステムクロック clk_sys/CLK_HF から生成されます。
- clk_lf : このクロックは低周波システムクロックで、ディープスリープモードでのみ使用できます。

選択できるリセットソースを以下に示します。

- rst_sys_act_n : 周辺分周器からのクロックを使用してアクティブパワーモードでのみ、スマート I/O がアクティブになります。
- rst_sys_dpslp_n : 周辺分周器からのクロックを使用してディープスリープモードを除くすべてのパワーモードで、スマート I/O がアクティブになります。
- rst_lf_dpslp_n : ILO からのクロックを使用してすべてのパワーモードで、スマート I/O がアクティブになります。

表 2 に SMARTIO_PRTx_CTL.CLOCK_SRC [12:8]レジスタの設定を示します。詳細については、Registers Reference Manual を参照してください。

表 2 クロックとリセット設定のレジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_CTL	CLOCK_SRC [12:8]	クロック (clk_block)/リセット (rst_block_n) ソースの選択 • 0 ... 7: io_data_in[0]/1 ... io_data_in[7]/'1' • 8... 15: chip_data[0]/1 ...chip_data[7]/'1' • 16: clk_smartio/rst_sys_act_n • 17: clk_smartio/rst_sys_dpslp_n • 19: clk_lf/rst_lf_dpslp_n • 20... 30: クロックソースは定数'0'です。 • 31: 非同期モード/”1”。クロックレス動作が設定される場合、これを選択してください。

2.2 同期化

I/O ポートと HSIOM の各入力信号は同期モードまたは非同期モードのいずれかで使用できます。同期化は入力信号をスマート I/O クロック (clk_block) と同期させます。

表 3 に同期化設定レジスタと設定を示します。詳細については、Registers Reference Manual を参照してください。

表 3 同期化設定のレジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_SYNC_CTL	IO_SYNC_EN [7:0]	io_data_in [7:0]信号を clk_block と同期 0: 同期なし 1: 同期
	CHIP_SYNC_EN [15:8]	chip_data [7:0]信号を clk_block と同期 0: 同期なし 1: 同期

2.3 3 入力のルックアップテーブル (LUT3 [x])

各 LUT3 [x] には、3 つの入力と 1 つの出力があります。各 LUT3 [x] ブロックのすべての入力 (Tr0_in, Tr1_in, Tr2_in) を選択する必要があります。入力が 1 つの場合、3 つの入力ソース (Tr0_in, Tr1_in, Tr2_in) すべてに入力します。各 LUT3 [x] は 3 つの入力信号があり、レジスタに設定された構成に基づいて出力を生成します。図 4 に各 LUT3 [x] の基本ブロックダイヤグラムを示します。出力パターンはレジスタによって設定されます。

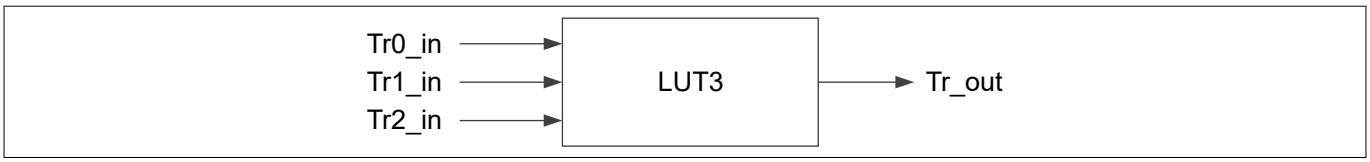


図 4　　LUT3 [x] のブロックダイヤグラム

2.3.1　　LUT3 [x]出力設定

LUT3 [x] の出力信号 (Tr_out) は 3 つの入力ソース (Tr2_in, Tr1_in, Tr0_in) に基づいて SMARTIO_PRTx_LUT_CTLy.LUT[7:0][2]を使用してプログラム可能です。表 4 に各 LUT3 [x] 設定の例を示します。

表 4　　LUT3 [x]出力設定

Tr 2_in	Tr 1_in	Tr 0_in	Tr_out	Tr_out (例 1)	Tr_out (例 2)
0	0	0	A	0	0
0	0	1	B	0	0
0	1	0	C	0	1
0	1	1	D	0	0
1	0	0	E	1	1
1	0	1	F	1	0
1	1	0	G	1	0
1	1	1	H	1	0

3 つの入力信号に対して 8 つの出力パターン (A~H) を生成します。A~H の各出力は 0 または 1 のブール値です。この出力パターン値 [H, G, F, E, D, C, B, A] は LUT [7:0] に設定されます。

例 1 の場合、出力パターンは[H, G, F, E, D, C, B, A] = [1, 1, 1, 1, 0, 0, 0, 0] です。したがって"0xF0"値が LUT [7:0] に設定されます。また例 2 の場合、出力パターンは[H, G, F, E, D, C, B, A] = [0, 0, 0, 1, 0, 1, 0, 0] です。したがって"0x14"値が LUT [7:0] に設定されます。

表 5 に LUT3 [x]出力設定の SMARTIO_PRTx_LUT_CTLy.LUT [7:0]レジスタを示します。詳細については、Registers Reference Manual を参照してください。

表 5　　LUT3 [x]出力設定のレジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_LUT_CTLy	LUT[7:0]	LUT3 [x]設定。LUT オペコード (LUT_OPC), 内部状態および LUT3 [x]入力信号 tr0_in, tr1_in, tr2_in に応じて、LUT3 [x]設定は LUT3[x]出力信号と次の順序状態の決定に使用します。

2.3.2　　LUT3 [x]入力の選択

各 LUT3 [x]の入力ソース (Tr0_in, Tr1_in, Tr2_in) は以下から選択できます。

- データユニット出力

[2]　この文章で使用されるレジスタ名のサブスクリプション y は LUT3 番号です。

- 他の LUT3 [x]出力信号 (Tr_out)
- HSIOM (chip_data [7:0]) からの入力信号
- I/O port (io_data_in [7:0]) からの入力信号

LUT3[7]~LUT3[4] は io_data/chip_data[7]~io_data/chip_data[4] で動作し、LUT3[3]~LUT3[0] は io_data/chip_data[3]~io_data/chip_data[0] で動作します。

入力ソースは SMARTIO_PRTx_LUT_SELy レジスタの LUT_TR0_SEL [3:0], LUT_TR1_SEL [11:8] および LUT_TR2_SEL [19:16]で設定できます。表 6 に SMARTIO_PRTx_LUT_SELy レジスタと入力選択設定を示します。データユニットの出力は tr0_in のみ入力できることに注意してください。詳細については、Registers Reference Manual を参照してください。

表 6 LUT3 [x]入力ソース設定のレジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_LUT_SELy	LUT_TR0_SEL, 3-0:	LUT3 [x]入力信号 tr0_in ソース選択 0: データユニット出力 1: LUT3 [1]出力 2: LUT3 [2]出力 3: LUT3 [3]出力 4: LUT3 [4]出力 5: LUT3 [5]出力 6: LUT3 [6]出力 7: LUT3 [7]出力 8:chip_data [0] (LUT3 [0], [1], [2], [3]の場合); chip_data [4] (LUT3 [4], [5], [6], [7]の場合) 9:chip_data [1] (LUT3 [0], [1], [2], [3]の場合); chip_data [5] (LUT3 [4], [5], [6], [7]の場合) 10:chip_data [2] (LUT3 [0], [1], [2], [3]の場合); chip_data [6] (LUT3 [4], [5], [6], [7]の場合) 11:chip_data [3] (LUT3 [0], [1], [2], [3]の場合); chip_data [7] (LUT3 [4], [5], [6], [7]の場合) 12:io_data_in [0] (LUT3 [0], [1], [2], [3]の場合); io_data_in [4] (LUT3 [4], [5], [6], [7]の場合) 13:io_data_in [1] (LUT3 [0], [1], [2], [3]の場合); io_data_in [5] (LUT3 [4], [5], [6], [7]の場合) 14:io_data_in [2] (LUT3 [0], [1], [2], [3]の場合); io_data_in [6] (LUT3 [4], [5], [6], [7]の場合) 15:io_data_in [3] (LUT3 [0], [1], [2], [3]の場合); io_data_in [7] (LUT3 [4], [5], [6], [7]の場合)

(続く)

表 6 (続き) LUT3 [x]入力ソース設定のレジスタ

レジスタ	ビット	設定
	LUT_TR1_SEL [11:8] / LUT_TR2_SEL [19:16]	LUT3 [x]入力信号 tr1_in / tr2_in ソース選択 0: LUT3 [0]出力 1: LUT3 [1]出力 2: LUT3 [2]出力 3: LUT3 [3]出力 4: LUT3 [4]出力 5: LUT3 [5]出力 6: LUT3 [6]出力 7: LUT3 [7]出力 8:chip_data [0] (LUT3 [0], [1], [2], [3]の場合); chip_data [4] (LUT3 [4], [5], [6], [7]の場合) 9:chip_data [0] (LUT3 [0], [1], [2], [3]の場合); chip_data [4] (LUT3 [4], [5], [6], [7]の場合) 10:chip_data [2] (LUT3 [0], [1], [2], [3]の場合); chip_data [6] (LUT3 [4], [5], [6], [7]の場合) 11:chip_data [3] (LUT3 [0], [1], [2], [3]の場合); chip_data [7] (LUT3 [4], [5], [6], [7]の場合) 12:io_data_in [0] (LUT3 [0], [1], [2], [3]の場合); io_data_in [4] (LUT3 [4], [5], [6], [7]の場合) 13:io_data_in [1] (LUT3 [0], [1], [2], [3]の場合); io_data_in [5] (LUT3 [4], [5], [6], [7]の場合) 14:io_data_in [3] (LUT3 [0], [1], [2], [3]の場合); io_data_in [7] (LUT3 [4], [5], [6], [7]の場合) 15:io_data_in [3] (LUT3 [0], [1], [2], [3]の場合); io_data_in [7] (LUT3 [4], [5], [6], [7]の場合)

各 LUT3 [x] は HSIOM と I/O ポート信号の入力接続に制限があります。柔軟なルーティングのために複数の LUT3 [x] が必要になる場合があります。

LUT3 [x]とデータユニットには組合せループが含まれません。ただし 1 つの LUT3 [x] がほかの LUT3 [x] 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。この制限を避けるために SMARTIO_PRTx_CTL.PIPELINE_EN ビットを使用します。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出力 (LUT3 [x]とデータユニット) の値が決定されます。表 7 に PIPELINE_EN 設定を示します。スマート I/O を使用しない場合、このビットを“1” (イネーブル) に設定して低消費電力を実現します。詳細については、Registers Reference Manual を参照してください。

表 7 PIPELINE_EN 設定

レジスタ	ビット	設定
SMARTIO_PRTx_CTL	PIPELINE_EN [25]	パイプラインレジスタのイネーブル 0: ディセーブル (レジスタがバイパスされます) 1: イネーブル (初期値)

2.3.3 LUT3 [x]動作

各 LUT3 [x] は 2 ビットのオペコードフィールドによって選択する次の 4 つの動作があります。4 つの動作は下記のとおりです。

組合せ

LUT3 [x] は単純な組合せです。各 LUT3 [x] 出力は LUT マッピング真理値表の結果であり、組合せ経路によってのみ遅延します。(ベーシックモード)

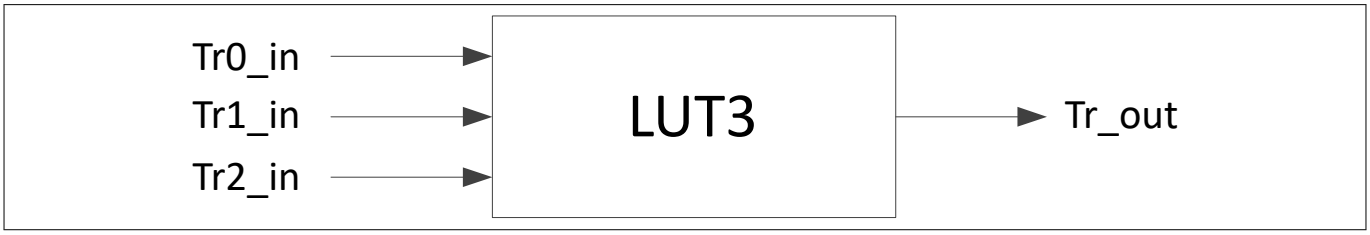


図 5 組合せ

ゲート入力 2

LUT3 [x] の入力 2 が同期します。その他の入力は LUT3 [x] に直接接続します。出力は組合せです。(入力同期)

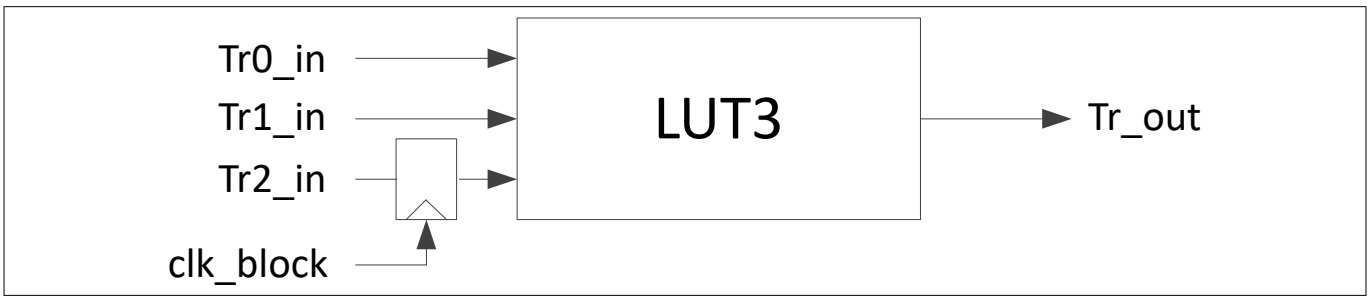


図 6 ゲート入力 2

ゲート出力

入力は LUT3 [x] に直接接続し、出力は同期します。(出力同期)

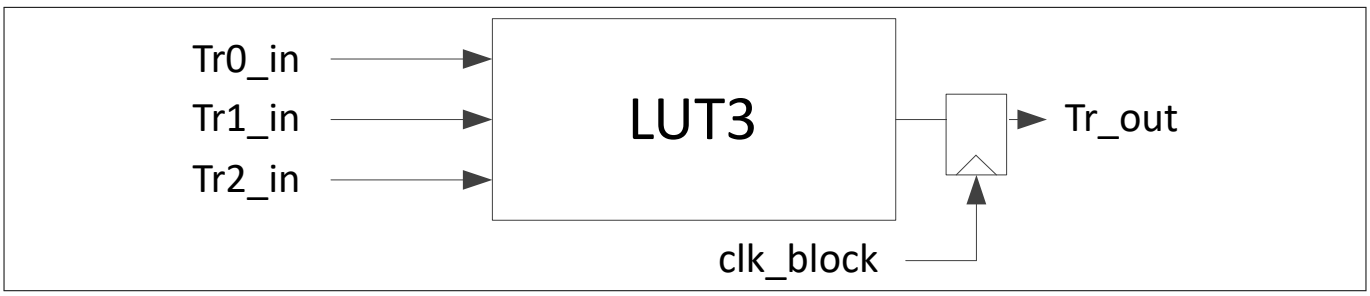


図 7 ゲート出力

フリップフロップのセット/リセット

入力信号は S/R フリップフロップの制御に使用されます。

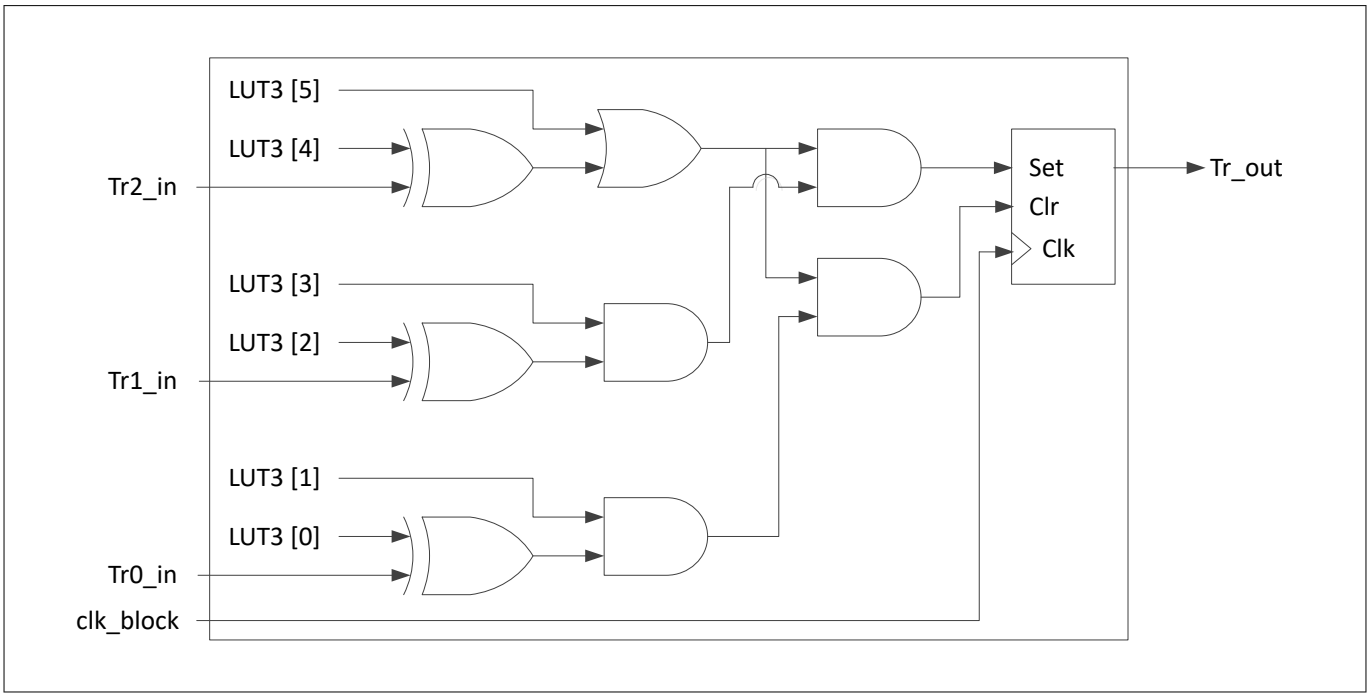


図 8　　S/R フリップフロップのイネーブル

これら 4 つの動作は表 8 に示すレジスタで設定されます。詳細は Registers Reference Manual を参照してください。

表 8　　LUT3 [x]モード設定のレジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_LUT_CTLy	LUT_OPC [9:8]	0: 組合せ 1: ゲート入力 2 2: ゲート出力 3: フリップフロップのセット/リセット

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 ビットデータ経路で構成されます。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 は DATA0 (data0_in [7:0]) と DATA1 (data1_in [7:0]) の 2 つのビットデータ入力に基づいてプログラマブル出力 (Tr_out) 信号を生成できます。フリップフロップにより内部状態は保持されます。DU 動作は最大 3 つの入力信号 (Tr0_in, Tr1_in, Tr2_in) で制御できます。図 9 にデータユニット (Data Unit) の基本ブロックダイヤグラムを示します。

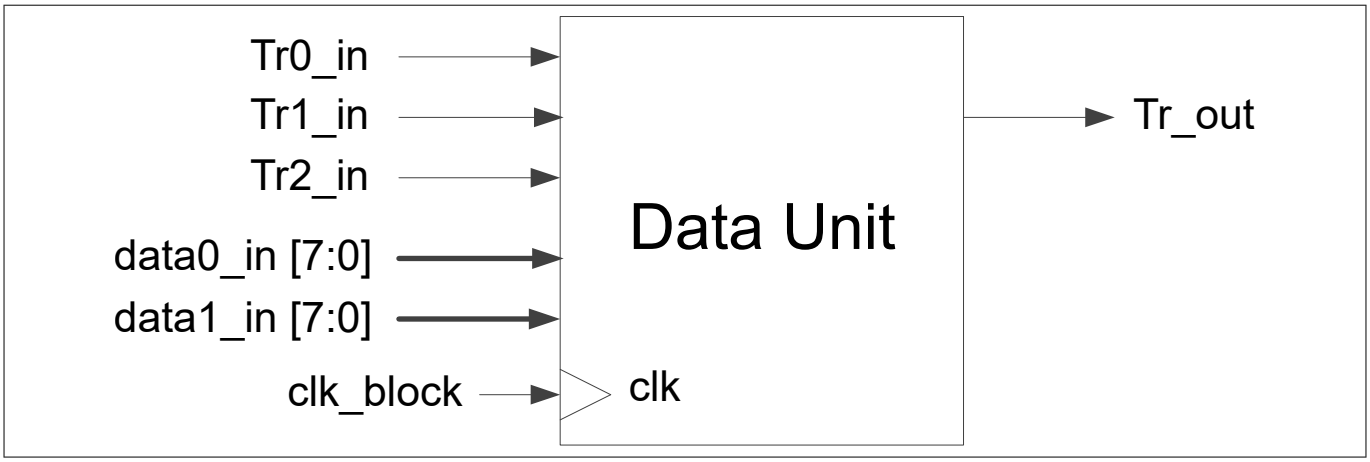


図 9　　データユニットのブロックダイヤグラム

2.4.1　　入力選択

DU には最大 3 つの制御入力信号があります。これらの信号は次からの入力として選択できます。

- 定数 “0”
- 定数 “1”
- DU 出力
- LUT3 [x]出力

必要な制御信号の数は DU オペコードによって異なります。

これらの入力は SMARTIO_PRTx_DU_SEL レジスタで設定できます。表 9 に SMARTIO_PRTx_DU_SEL レジスタと入力選択設定を示します。詳細については、Registers Reference Manual を参照してください。

表 9　　DU 入力ソース設定のレジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_DU_SEL	DU_TR0_SEL [3:0] / DU_TR1_SEL [11:8] / DU_TR2_SEL [19:16]	データユニット入力信号 “tr0_in” / “tr1_in” / “tr2_in”のソース選択 0: 定数 '0' 1: 定数 '1' 2: データユニット出力 3- 10: LUT3 [x]出力 それ以外: 未定義

DATA 0 と DATA 1 は DU ロジック初期化の入力データとして使用します。これらのデータは次から選択できます。

- 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

データユニットが扱うデータ幅は 1 ビットから 8 ビットの間で変更可能です。表 10 に DU への入力データの設定レジスタを示します。

表 10 DU データ設定のレジスタ

レジスタ	ビット	設定
SMARTIO_PRTx_DU_SEL	DU_DATA0_SEL [25:24] / DU_DATA1_SEL [29:28]	データユニット入力データ “data0_in” / “data1_in”のソース選択 0: 0x00 1: chip_data [7:0]. 2: io_data_in [7:0]. 3: SMARTIO_PRTx_DATA.DATA [7:0] MMIO レジスタフィールド
SMARTIO_PRTx_DATA	DATA[7:0]	データユニット入力データソース
SMARTIO_PRTx_DU_CTL	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。

2.4.2 データユニットの動作

DU 動作は SMARTIO_PRTx_DU_CTL.DU_OPC [11:8]により定義します。表 11 に DU オペコードの設定レジスタを示します。詳細については、Registers Reference Manual を参照してください。

表 11 DU 動作コードの設定

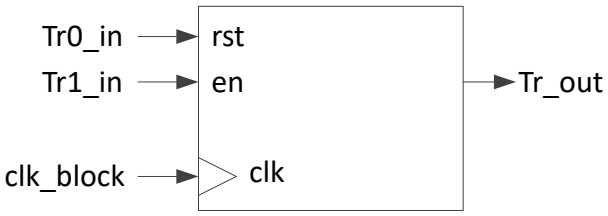
レジスタ	ビット	設定
SMARTIO_PRTx_DU_CTL	DU_OPC [11:8]	データユニットオペコードはデータユニット動作を指定します。 “1”: INCR “2”: DECR “3”: INCR_WRAP “4”: DECR_WRAP “5”: INCR_DECR “6”: INCR_DECR_WRAP “7”: ROR “8”: SHR “9”: AND_OR “10”: SHR_MAJ3 “11”: SHR_EQL それ以外: 未定義 初期値: 未定義

表 12 に各 DU の動作を示します。

表 12 の‘動作’列に動作概要と疑似コードを示します。疑似コードにて “Combinational:” は動作が以前の出力状態と無関係であることを示します。"Registered:" はデータが入力および以前の出力状態 (フリップフロップを使用して) で動作することを示します。

表 12 DU の動作

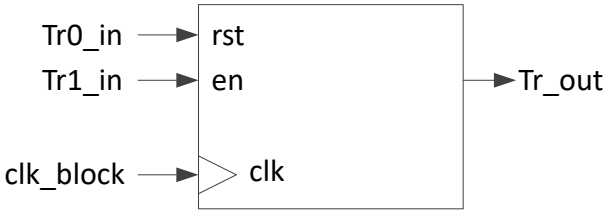
動作コード	動作

DU_OPC [11:8] = 1:
INCR
	INCR は初期値 (DATA 0) から最終値 (DATA 1) に達するまでデータを 1 ずつインクリメントします。
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_data1 = (data & mask) == DATA1`
	`& mask`
	
	Combinational:
	
	`Tr_out = data_eql_data1`
	
	
	
	Registered:
	
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? data: (data + 1) & mask;`

(続く)

表 12 (続き) DU の動作

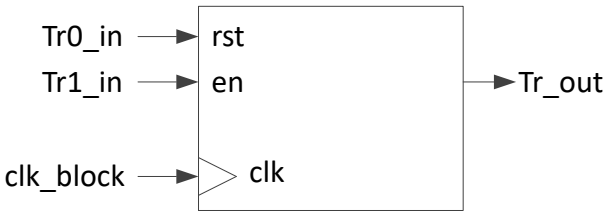
動作コード	動作

DU_OPC [11:8] = 2:
DECR
	DECR は初期値 (DATA 0) から '0'に達するまでデータをデクリメントします。
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	Combinational:
	`Tr_out = data_eql_0`
	
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? data: (data + 1) & mask;`

(続く)

表 12 (続き) DU の動作

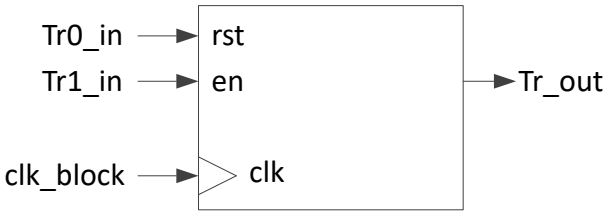
動作コード	動作

DU_OPC [11:8] = 3:
INCR_WRAP
	INCR_WRAP は INCR と同様に動作しますが、DATA 1 で停止ではなく DATA 0 に戻ります。
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	
	`Tr_out = data_eql_data1`
	
	
	
	Registered:
	
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? DATA0 & mask: (data + 1) & mask;`

(続く)

表 12 (続き) DU の動作

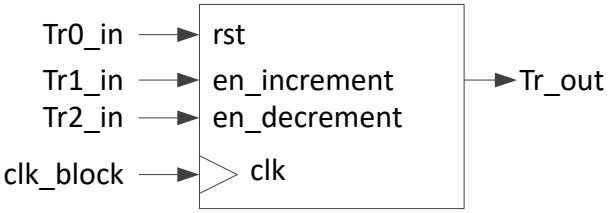
動作コード	動作

DU_OPC [11:8] = 4:
DECR_WRAP
	DECR_WRAP は DECR と同様に動作します。'0'で停止ではなく DATA0 に戻ります。
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) - 1`
	`data_eql_0 = (data & mask) == 0`
	Combinational:
	`Tr_out = data_eql_0`
	
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_0? DATA0 & mask:`
	`(data + 1) & mask;`

(続く)

表 12 (続き) DU の動作

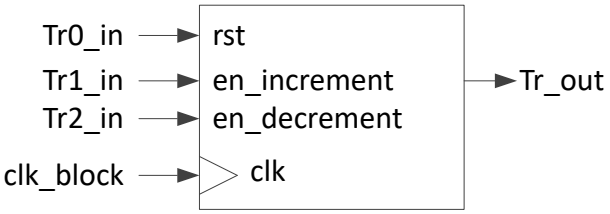
動作コード	動作

DU_OPC [11:8] = 5:
INCR_DECR
	INCR_DECR は INCR と DECR の組合せです。トリガ信号に応じてインクリメントまたはデクリメントを開始します。インクリメントは DATA 1 で、デクリメントは '0'で停止します。
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) – 1`
	`data_eql_0 = (data & mask) == 0`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	
	`Tr_out = data_eql_data1 | data_eql_0`
	
	
	
	Registered:
	
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? data: (data + 1) & mask;`
	`else if (Tr2_in)`
	`data <= data_eql_0? data: (data - 1) & mask;`

(続く)

表 12 (続き) DU の動作

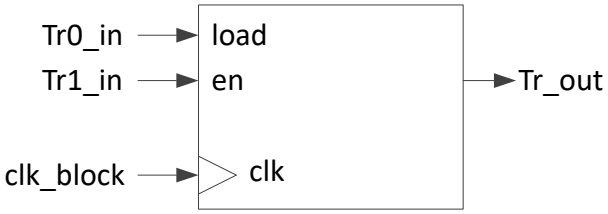
動作コード	動作

DU_OPC [11:8] = 6:
INCR_DECR_WRAP
	INCR_DECR_WRAP は INCR_DECR と同様に動作しますが、リミット (DATA 1 または '0') に達すると DATA 0 に戻ります。
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) – 1`
	`data_eql_0 = (data & mask) == 0`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	
	Combinational:
	
	`Tr_out = data_eql_data1 | data_eql_0`
	
	
	
	Registered:
	
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in)`
	`data <= data_eql_data1? DATA0 & mask: (data + 1) & mask;`
	`else if (Tr2_in)`
	`data <= data_eql_0 ? DATA0 & mask: (data - 1) & mask;`

(続く)

表 12 (続き) DU の動作

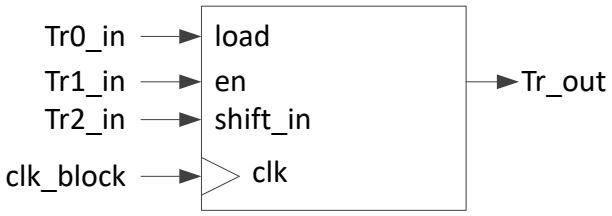
動作コード	動作

DU_OPC [11:8] = 7:
ROR
	POR がデータを右にシフトさせ LSB が送信されます。シフト用のデータは DATA 0 から取得します。
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) – 1`
	
	Combinational:
	
	`Tr_out = data [0]`
	
	
	
	Registered:
	
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= data [7:1] & mask;`
	`data [du_size] <= data [0]`
	`}`

(続く)

表 12 (続き) DU の動作

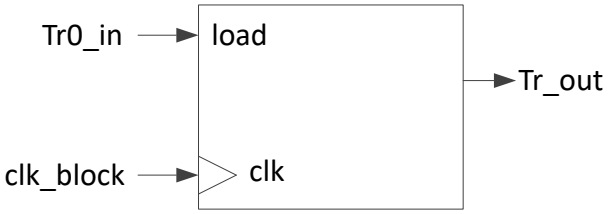
動作コード	動作

DU_OPC [11:8] = 8:
SHIR
	SHIR はシフトレジスタ動作を行います。初期データ (DATA 0) がシフトアウトされ、tr2_in のデータがシフトインされます。
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) – 1`
	
	Combinational:
	
	`Tr_out = data [0]`
	
	
	
	Registered:
	
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= data [7:1] & mask;`
	`data [du_size] <= Tr2`
	`}`

(続く)

表 12 (続き) DU の動作

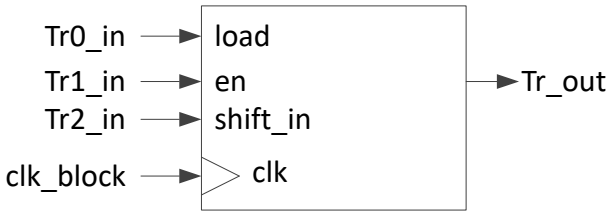
動作コード	動作

DU_OPC [11:8] = 9:
AND_OR
	data1 と data0 をマスクとともに AND し、AND された出力のすべてのビットを OR します。
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) – 1`
	Combinational:
	`Tr_out =| (data & DATA1 & mask)`
	
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`

(続く)

表 12 (続き) DU の動作

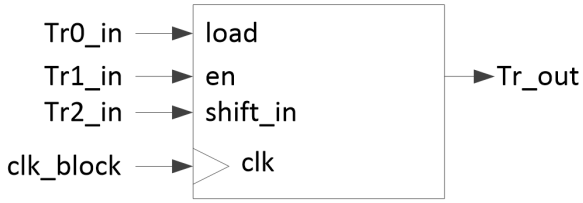
動作コード	動作

DU_OPC [11:8] = 10:
SHR_MAJ3 (Majority 3)
	SHR_MAJ3 は SHR と同様に動作します。シフトアウト値を送信ではなく data [0]の最後の 3 つのサンプル/シフトアウト値の少なくとも 2 つのサンプルが高い場合は '1'を送信します。それ以外の場合は '0'を送信します。この関数は最後の 3 つのサンプルの大部分を送信します。
	
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) – 1`
	
	Combinational:
	Tr_out = data == 0x03 | data == 0x05 | data == 0x06 | data == 0x07
	
	`Tr_out = data == 0x03 | data == 0x05 | data == 0x06 | data == 0x07`
	
	
	
	Registered:
	
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= (0, data [7:1]) & mask;`
	`data [du_size] <= Tr2_in`
	`}`

(続く)

表 12 (続き) DU の動作

動作コード	動作

DU_OPC [11:8] = 11:
SHR_EQL (Match DATA1)
	SHR_EQL は SHR と同様に動作しますが、シフトアウトではなく出力は比較結果 (DATA 0 == DATA 1) になります。
	`du_size = Size - 1`
	`mask = (1 << (DU_SIZE+1)) – 1`
	`data_eql_data1 = (data & mask) == DATA1 & mask`
	Combinational:
	`Tr_out = data_eql_data1`
	
	Registered:
	`data <= data;`
	`if (Tr0_in)`
	`data <= DATA0 & mask;`
	`else if (Tr1_in) {`
	`data <= (0, data [7:1]) & mask;`
	`data [du_size] <= Tr2_in`
	`}`

3 スマート I/O の設定

図 10 にスマート I/O の設定フローの例を示します。

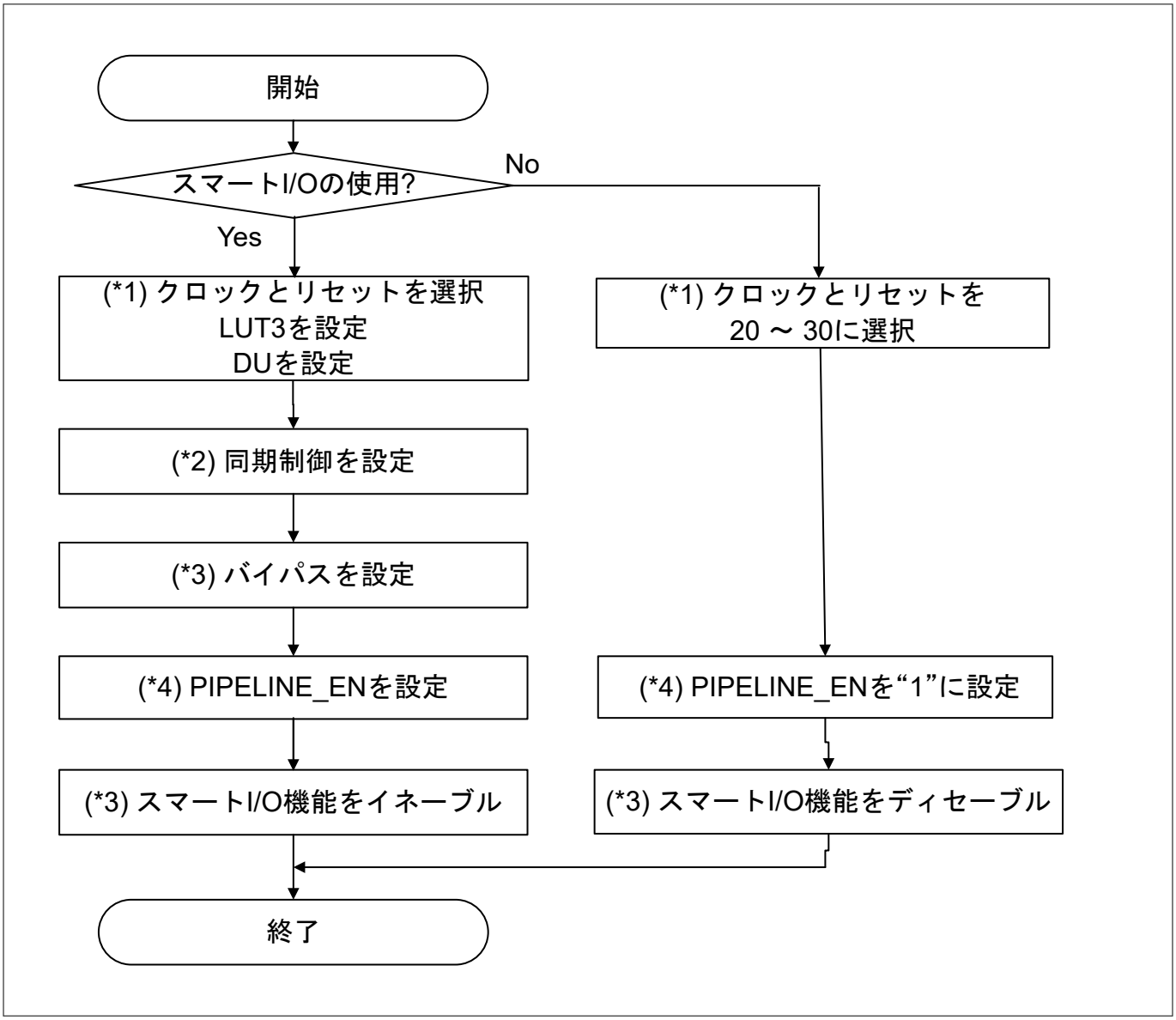


図 10 スマート I/O の設定フロー

スマート I/O を設定する場合、まずクロックとリセット, 同期化, LUT3 [x] および DU などの各コンポーネントを初期化します。スマート I/O をイネーブル (SMARTIO_PRTx_CTL.ENABLE を“1”に設定: イネーブル) する前にすべてのコンポーネントとルーティングを設定する必要があります。

スマート I/O を使用しない場合、クロックとリセットコンポーネントのクロック選択を 20 から 30 の間の値に設定し、PIPELINE_EN を“1”に設定して低消費電力を実現します。

注: *(*1) ポート, ソース, およびクロック設定については*スマート I/O の構造*を、LUT3 [x] 設定については* 3 入力のルックアップテーブル (LUT3 [x])*を、DU 設定については*データユニット (DU)*を参照してください。*

*(*2) 同期化設定については*表 3 *を参照してください。*

*(*3) バイパスとスマート I/O イネーブル設定については*表 1 *を参照してください。*

*(*4) PIPELINE_EN 設定については*表 7 *を参照してください。*

4 設定例

ここでは、サンプルドライバーライブラリ (SDL) を使用してスマート I/O を使用する方法について説明します。このアプリケーションノートのプログラムコードは SDL の一部です。SDL についてはその他の参考資料 を参照してください。

SDL には基本的に、設定部とドライバ部があります。設定部は、主に目的の操作のパラメータ値を設定します。ドライバ部は、設定部のパラメータ値に基づいて各レジスタを設定します。ご使用のシステムに合わせて設定部を設定できます。

スマート I/O は入出力信号の簡単な論理演算または内部 HSIOM ポートと I/O ポート間の内部ルーティングを必要とするアプリケーションに適しています。これらの動作に CPU は不要です。ここではユースケースに従ってスマート I/O の使用方法を説明します。

この例では、CYT2B7 シリーズを使用しています。

4.1 極性反転による I/O ピンから HSIOM へのルーティングの変更のユースケース

ここではスマート I/O を使用してルーティングと簡単な論理演算の例を説明します。

このユースケースではポート 13 のピン 7 (io_data_in [7]) からの入力を HSIOM のピン 1 (smartio_data [1]) に接続されるようにルーティングを変更します。また io_data_in [7] の極性を反転させ、反転された io_data_in [7] 信号を smartio_data [1] に出力します。スマート I/O が使用できる I/O ポートについてはデバイス データシートの Package Pin List and Alternate Functions を参照してください。

図 11 に I/O ポートから HSIOM への信号の反転接続を示します。このユースケースでは LUT3 [1] と LUT3 [7] を使用します。

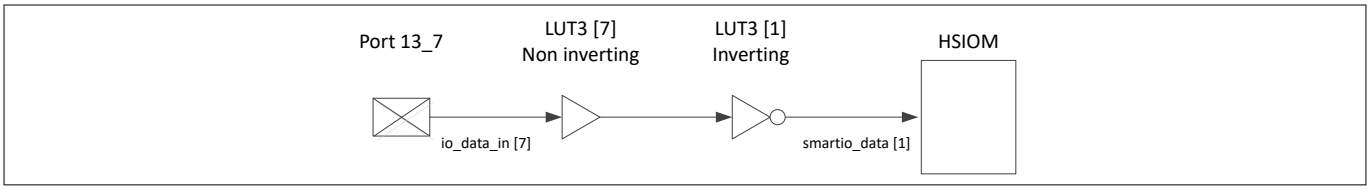


図 11 信号反転のイメージ

図 12 に、この例の信号パスを示します。

4 設定例

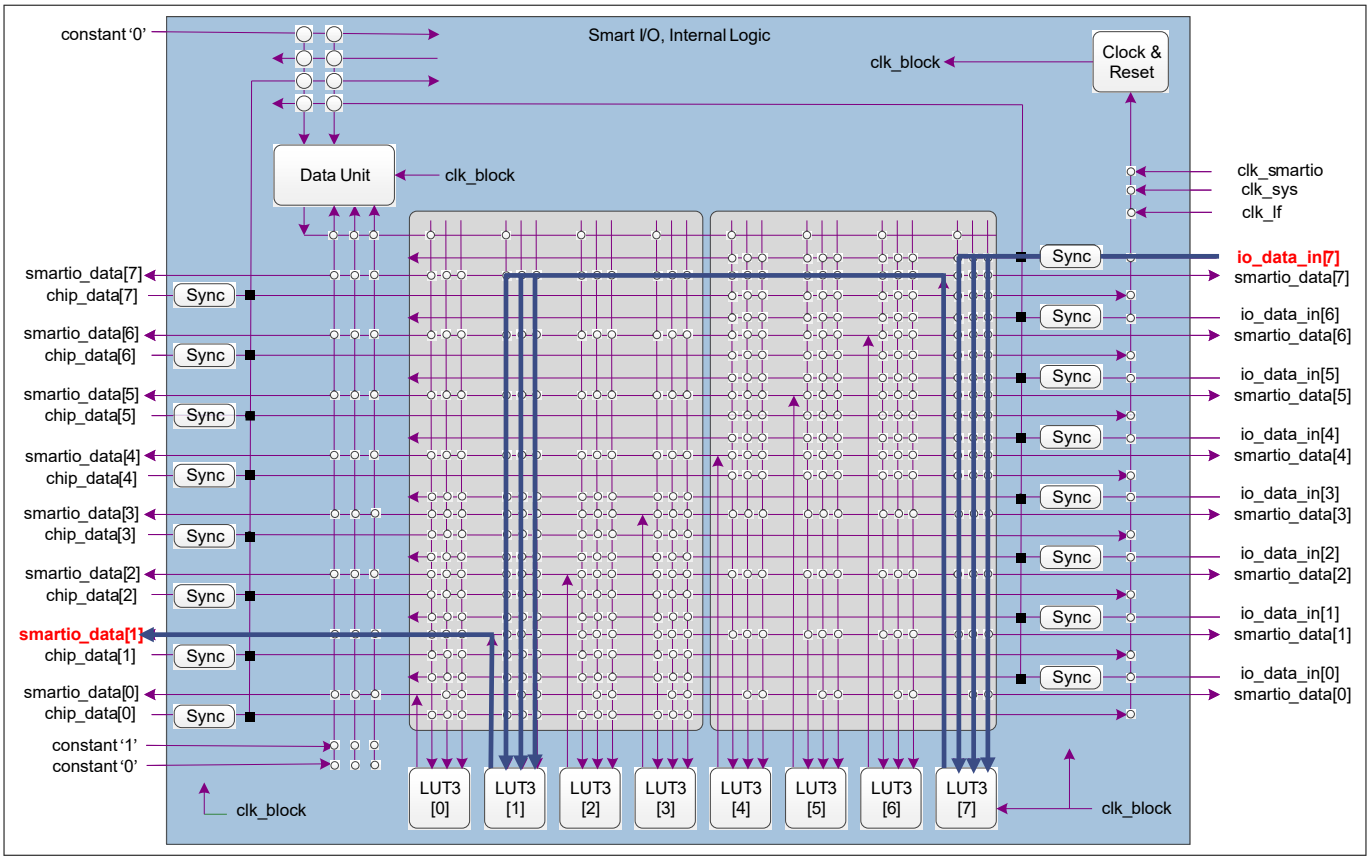


図 12　　ルーティング概要の例

LUT3 [1] と LUT3 [7] を使用することに注意してください。io_data [7] を入力として使用できる LUT3[7:4] は smartio_data [1] に直接ルーティングできません。したがって LUT3 [7] の出力は smartio_data [1] にルーティング可能な LTU3 [1] を経由する必要があります。このユースケースでは LUT3 [1] は LUT3 [7] からの入力信号を反転させて smartio_data [1] に出力します。

表 13 に LUT3 [7] の真理値表を、表 14 に LUT3 [1] の真理値表を示します。表の太字部分は無効な組合せパターンを示します。

表 13　　ルックアップテーブル LUT3 [7]

Tr2_in	Tr1_in	Tr0_in	Tr_out
0	0	0	0
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	1	1	1

4 設定例

表 14　　ルックアップテーブル LUT3 [1]

Tr2_in	Tt1_in	Tr0_in	Tr_out
0	0	0	1
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**1**
1	**0**	**0**	**1**
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	1	1	0

LUT3 [1] の 3 つの入力は同じ信号を入力し、同様に LUT3 [7] は同じ信号を入力します。したがって LTU3 の入力パターンは[Tr2_in, Tr1_in, Tr0_in] = [0, 0, 0] または [1, 1, 1] です。

LUT3 [7]は極性を変更しません。すなわち、[Tr2_in, Tr1_in, Tr0_in] = [1, 1, 1] のとき Tr_out は “1”で、それ以外のときは Tr_out = “0”です。

LUT3 [1]は極性を反転させます。すなわち、[Tr2_in, Tr1_in, Tr0_in] = [1, 1, 1] のとき Tr_out は “0”で、それ以外のときは Tr_out = “1”です。

図 13 に、スマート I/O の設定手順を示します。

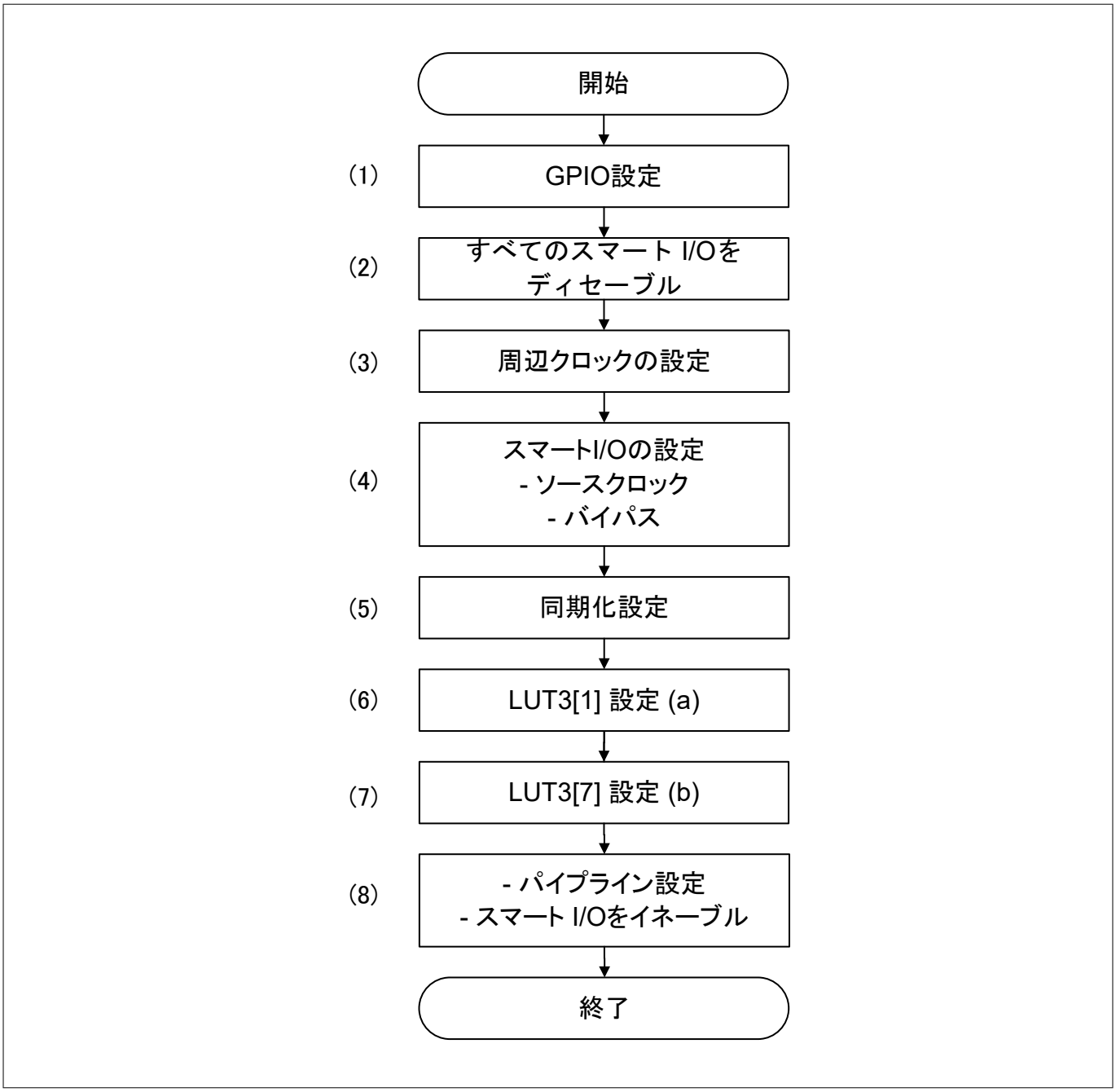


図 13　　スマート I/O の設定手順

(a) 設定パターンについては表 14 を参照してください。
(b) 設定パターンについては表 13 を参照してください。

4.1.1　　設定とサンプルコード

表 15 に SDL のスマート I/O 設定部のパラメータを示し、表 16 に関数を示します。

4 設定例

表 15　　スマート I/O の設定パラメーター覧

パラメータ	説明	値
SMART_IO_CLK_ACTIVE	スマート I/O クロックソース定義	1ul (アクティブなクロックソースを選択)
CY_SMARTIO_CLK_INV	スマート I/O クロック定義	PCLK_SMARTIO13_CLOCK
SMART_IO_PORT	スマート IO ポート定義	SMARTIO_PRT13 (スマート I/O のポート 13 に割当て)
SMARTIO_BYPASS_CH_MASK	バイパスチャネルマスク定義 io_data_in [7] to smartio_data [7]: バイパスなし io_data_in [6] to smartio_data [6]: バイパス io_data_in [5] to smartio_data [5]: バイパス io_data_in [4] to smartio_data [4]: バイパス io_data_in [3] to smartio_data [3]: バイパス io_data_in [2] to smartio_data [2]: バイパス io_data_in [1] to smartio_data [1]: バイパスなし io_data_in [0] to smartio_data [0]: バイパス	0x7Dul (表 1 参照) io_data_in [6] から io_data_in [2] および io_data_in [0] はスマート I/O では未使用
SMARTIO_IOSYNC_CH_MASK	IO sync チャネルマスク定義	0x00ul (表 3 参照)
LUT_IP_BUTTON_PORT	LUT3[7] 入力の入力ポート定義	CY_BUTTON2_PORT (GPIO ポート 13 に割当て)
LUT_IP_BUTTON_PIN	LUT3[7] 入力の入力ポートピン定義	CY_BUTTON2_PIN (GPIO ポート 7 ピンに割当て)
LUT_IP_BUTTON_PIN_MUX	入力ポートピン機能を設定	CY_BUTTON2_PIN_MUX (GPIO に割当て)
CY_SMARTIO_LUTTR_IO。	LUT3[7] 入力を選択	CY_SMARTIO_LUTTR_IO7

(続く)

表 15 (続き) スマート I/O の設定パラメーター覧

パラメータ	説明	値
LUT_INV_OUT_PORT	LUT3[7] 入力の出力ポートを定義	P13_1_PORT (GPIO ポート 13 に割当て)
LUT_INV_OUT_PIN	LUT3[7] 入力の出力ポートピンを定義	P13_1_PIN (GPIO ポート 1 ピンに割当て)
LUT_INV_OUT_PIN_MUX	出力ポートピン機能を設定	P13_1_GPIO (GPIO に割当て)
LUTx_OUT_MAP	LUT3[7] 出力パターン	0x80ul (表 13 参照)
LUTx_INV_OUT_MAP	LUT3[1] 出力パターン	0x7Ful (表 14 参照)
LUTx_LOGIC_OPCODE	LUT3 動作モードの選択	CY_SMARTIO_LUTOPC_COMB (Combinatorial に割当て)
CY_SYSCLK_DIV_16_BIT	分周器タイプを 16 ビット分周器に選択	1ul
CY_SMARTIO_ENABLE	スマート I/O とパイプラインをイネーブルに設定	1ul
CY_SMARTIO_DISABLE	スマート I/O とパイプラインをディセーブルに設定	0ul
CY_SMARTIO_DEINIT	スマート I/O をデフォルト値にリセット	0ul
CY_SMARTIO_CHANNEL_ALL	すべてのピンにスマート I/O バイパスを設定	0xfful
CY_SMARTIO_CLK_DIVACT	clk_smartio/ rst_sys_act_n へのソースクロックを選択。表 2 を参照してください。	16ul
CY_SMARTIO_CLK_GATED	ソースクロックをクロックソース定数「0」に選択。表 2 を参照してください。	20ul
CY_SMARTIO_CLK_ASYNC	ソースクロックを非同期モード/1 に選択。表 2 を参照してください。	31ul
CY_SMARTIO_LUTTR_LUT7_OUT	LUT3[1] 入力を選択	7ul (LUT3[7] 出力に割当て。表 6 参照)
CY_SMARTIO_LUTTR_IO7	LUT3[7] 入力を選択	15ul (io_data_in [7] 出力に割当て。表 6 参照)
smart_io_cfg.clkSrc	ソースクロック設定	CY_SMARTIO_CLK_DIVACT
smart_io_cfg.bypassMask	バイパス設定	SMARTIO_BYPASS_CH_MASK
smart_io_cfg.ioSyncEn	同期化設定	SMARTIO_IOSYNC_CH_MASK

(続く)

4 設定例

表 15 (続き) スマート I/O の設定パラメーター覧

パラメータ	説明	値
lutCfgLut1.opcode	LUT3[1] 動作モード設定	LUTx_LOGIC_OPCODE
lutCfgLut1.lutMap	LUT3[1] 出力パターンを設定	LUTx_INV_OUT_MAP
lutCfgLut1.tr0	LUT3[1] tr0 入力を設定	CY_SMARTIO_LUTTR_LUT7_OUT
lutCfgLut1.tr1	LUT3[1] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT7_OUT
lutCfgLut1.tr2	LUT3[1] tr2 入力を設定	CY_SMARTIO_LUTTR_LUT7_OUT
lutCfgLut7.opcode	LUT3[7] 動作モード設定	LUTx_LOGIC_OPCODE
lutCfgLut7.lutMap	LUT3[7] 出力パターンを設定	CY_SMARTIO_LUTTR_IO。
lutCfgLut7.tr0	LUT3[7] tr0 入力を設定	CY_SMARTIO_LUTTR_IO。
lutCfgLut7.tr1	LUT3[7] tr1 入力を設定	CY_SMARTIO_LUTTR_IO。
lutCfgLut7.tr2	LUT3[7] tr2 入力を設定	CY_SMARTIO_LUTTR_IO。

表 16 スマート I/O の設定関数一覧

関数	説明	備考
Init_IO_Pin()	GPIO ポートピン設定	–
Cy_SmartIO_Deinit()	スマート I/O をデフォルト値にリセット	以下のスマート I/O レジスタをリセット; SMARTIO_PRTx_CTL, SMARTIO_PRTx_SYNC_CTL, SMARTIO_PRTx_LUT_SELy, SMARTIO_PRTx_LUT_CTLy, SMARTIO_PRTx_DU_SEL, SMARTIO_PRTx_DU_CTL および SMARTIO_PRTx_DATA
Init_SmartIO()	スマート I/O 初期化とイネーブル設定 Init_SmartIO_Cfg() および Cy_SmartIO_Enable()の呼び出し	–
Init_SmartIO_Cfg()	スマート I/O の各種設定および Cy_SmartIO_Init()の呼び出し	–
Cy_SmartIO_Enable()	スマート I/O のイネーブル	PIPELINE_EN および ENABLED ビットに書込み
Cy_SmartIO_Init()	スマート I/O のレジスタ設定	ソースクロック, バイパス, 同期化, LUT3, および DU の関連レジスタに書込み

Code Listing 1 に、極性反転による I/O ピンから HSIOM へのルーティングの変更のプログラムの例を示します。GPIO とクロック設定については Architecture Reference Manual と アプリケーションノート を参照してください。

次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- base はスマート I/O レジスタのベースアドレスのポインタを示します。

4 設定例

- **base->unLUT_SEL[idx].u32Register** は、Registers Reference Manual で言及されている SMARTIO_PRTx_LUT_SEL[idx] レジスタです。他のレジスタも同様です。“**x**”はポートサフィックス番号、“**idx**”はレジスタインデックス番号を表します。
- レジスタ設定のパフォーマンス向上のため、SDL では完全な 32 ビットデータをレジスタに書き込みます。各ビットフィールドが生成され、最終的な 32 ビットデータとしてレジスタに書き込まれます。

```
un_SMARTIO_PRT_CTL_t workCTL= {.u32Register = 0ul};
workCTL.stcField.u1ENABLED     = CY_SMARTIO_DISABLE;
workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_ENABLE;
workCTL.stcField.u5CLOCK_SRC   = CY_SMARTIO_CLK_GATED;
workCTL.stcField.u8BYPASS      = CY_SMARTIO_CHANNEL_ALL;
base->unCTL.u32Register        = workCTL.u32Register;
```

レジスタ表記の結合および構造表現の詳細については、`hdr/rev_x/ip` の `cyip_smartio_v2.h` を参照してください。

4 設定例

Code Listing 1 極性反転による I/O ピンから HSIOM へのルーティングの変更例

```
/* Smart IO clock source selection */
#define SMART_IO_CLK_ACTIVE             1ul    /* Define Clock active */

/* Smart IO port selections macro */
#define SMART_IO_PORT                   SMARTIO_PRT13  /* Define Smart I/O port */

#define CY_SMARTIO_CLK_INV              PCLK_SMARTIO13_CLOCK  /* Define Smart I/O port */

/* Bypass channel mask */
#define SMARTIO_BYPASS_CH_MASK          0x7Dul    /* Define Smart I/O bypass channel */

/* IO sync channel mask */
#define SMARTIO_IOSYNC_CH_MASK          0x00ul   /* Define Smart I/O sync channel mask */

/* Lut input button pin configuration */
/* Define input port to LUT3[7] */
#define LUT_IP_BUTTON_PORT              CY_BUTTON2_PORT     /* GPIO_PRT3 */
#define LUT_IP_BUTTON_PIN               CY_BUTTON2_PIN      /* 7 */
#define LUT_IP_BUTTON_PIN_MUX           CY_BUTTON2_PIN_MUX  /* P13_7_GPIO */

#define CY_SMARTIO_LUTTR_IO             CY_SMARTIO_LUTTR_IO7  /**< I/O signal 7 (for LUT
4,5,6,7) */

/* LUT output pin configuration */
/* Define output port from LUT3[1] */
#define LUT_INV_OUT_PORT                P13_1_PORT
#define LUT_INV_OUT_PIN                 P13_1_PIN
#define LUT_INV_OUT_PIN_MUX             P13_1_GPIO

/* LUT output map */
/* Define LUT3[1] and LUT3[7] output pattern */
#define LUTx_OUT_MAP                    0x80ul
#define LUTx_INV_OUT_MAP                0x7Ful

/* LUT logic circuit type macro */
#define LUTx_LOGIC_OPCODE               CY_SMARTIO_LUTOPC_COMB  /* Define LUT logic circuit
type */

#define CY_SMARTIO_ENABLE  1ul
#define CY_SMARTIO_DISABLE 0ul
#define CY_SMARTIO_DEINIT  0ul
#define CY_SMARTIO_CHANNEL_ALL  0xfful   /**< All channels */


/* Button input configuration */
/* Configure Port for input (Port13 pin) */
cy_stc_gpio_pin_config_t button_cfg =
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_HIGHZ,
    .hsiom     = LUT_IP_BUTTON_PIN_MUX,
```

```
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};

/* Configure Port for output (Port13 pin) */
cy_stc_gpio_pin_config_t inv_out_cfg =
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
    .hsiom     = LUT_INV_OUT_PIN_MUX,
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};


int main(void)
{
    :
    Init_IO_Pin();  /* (1) Configure GPIO pin. See Code Listing 2. */

    /* Deinit before Init */
    Cy_SmartIO_Deinit(SMART_IO_PORT);   /* Disable all Smart I/O. See Code Listing 3 */

    /* SmartIO peripheral clock divider setting */
    {
        /* (3) Configure peripheral Clock */
        Cy_SysClk_PeriphAssignDivider(CY_SMARTIO_CLK_INV, CY_SYSCLK_DIV_16_BIT, 0ul);
        uint32_t sourceFreq = 80000000ul;
        uint32_t targetFreq = 12000000ul;
        uint32_t divNum = (sourceFreq / targetFreq);

        Cy_SysClk_PeriphSetDivider(CY_SYSCLK_DIV_16_BIT, 0ul, (divNum - 1ul));
        Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_16_BIT, 0ul);
    }

    /* Initialization call for the Smart IO */
    Init_SmartIO();   /* Initialize Smart I/O. See Code Listing 4 */

    for(;;);
}
```

4 設定例

Code Listing 2 Init_IO_Pin() 関数

```
void Init_IO_Pin(void)
{
    /* Configure Port13 7pin. */
    Cy_GPIO_Pin_Init(LUT_IP_BUTTON_PORT, LUT_IP_BUTTON_PIN, &button_cfg);

    /* Configure Port13 1pin. */
    Cy_GPIO_Pin_Init(LUT_INV_OUT_PORT, LUT_INV_OUT_PIN, &inv_out_cfg);
}
```

Code Listing 3 Cy_SmartIO_Deinit() 関数

```
void Cy_SmartIO_Deinit(volatile stc_SMARTIO_PRT_t* base)
{
    un_SMARTIO_PRT_CTL_t workCTL= {.u32Register = 0ul};
    /* (2) Disable all Smart I/O port */
    workCTL.stcField.u1ENABLED     = CY_SMARTIO_DISABLE;
    workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_ENABLE;
    workCTL.stcField.u5CLOCK_SRC   = CY_SMARTIO_CLK_GATED;
    workCTL.stcField.u8BYPASS      = CY_SMARTIO_CHANNEL_ALL;
    base->unCTL.u32Register        = workCTL.u32Register;

    base->unSYNC_CTL.u32Register = CY_SMARTIO_DEINIT;
    for(uint8_t idx = CY_SMARTIO_LUTMIN; idx < CY_SMARTIO_LUTMAX; idx++)
    {
        base->unLUT_SEL[idx].u32Register = CY_SMARTIO_DEINIT;
        base->unLUT_CTL[idx].u32Register = CY_SMARTIO_DEINIT;
    }
    base->unDU_SEL.u32Register = CY_SMARTIO_DEINIT;
    base->unDU_CTL.u32Register = CY_SMARTIO_DEINIT;
    base->unDATA.u32Register = CY_SMARTIO_DEINIT;
}
```

Code Listing 4 Init_SmartIO() 関数

```
void Init_SmartIO(void)
{
    cy_en_smartio_status_t retStatus = (cy_en_smartio_status_t)0xFF;

    retStatus = Init_SmartIO_Cfg();  /* Configure Smart I/O. See Code Listing 5 */
    if(retStatus == CY_SMARTIO_SUCCESS)
    {
        /* After all the configuration, enable SMART IO */
        Cy_SmartIO_Enable(SMART_IO_PORT);   /* Enable Smart I/O. See Code Listing 6 */
    }
}
```

4 設定例

Code Listing 5 Init_SmartIO_Cfg() 関数

```
cy_en_smartio_status_t Init_SmartIO_Cfg(void)
{
    cy_stc_smartio_lutcfg_t lutCfgLut1;
    cy_stc_smartio_lutcfg_t lutCfgLut7;

    cy_stc_smartio_config_t smart_io_cfg;
    cy_en_smartio_status_t retStatus = (cy_en_smartio_status_t)0xFF;

    /* initialize the Smart IO structure */
    memset(&lutCfgLut1, 0, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&lutCfgLut7, 0, sizeof(cy_stc_smartio_lutcfg_t));
    /* Clear configuration structure */
    memset(&smart_io_cfg, 0, sizeof(cy_stc_smartio_config_t));

#ifdef SMART_IO_CLK_ACTIVE
    /* Active clock source is selected */
    /* Configure Smart I/O clock source */
    smart_io_cfg.clkSrc = (cy_en_smartio_clksrc_t)CY_SMARTIO_CLK_DIVACT;
#else
    /* Asynchronous clock source is selected */
    smart_io_cfg.clkSrc = (cy_en_smartio_clksrc_t)CY_SMARTIO_CLK_ASYNC;
#endif  /* SMART_IO_CLK_ACTIVE */

    /* Bypass channel mask for input and output pin */
    /* Configure BYPASS setting */
    smart_io_cfg.bypassMask = SMARTIO_BYPASS_CH_MASK;

    /* IO channel sync mask for selected pin */
    /* Configure Synchronizer setting */
    smart_io_cfg.ioSyncEn = SMARTIO_IOSYNC_CH_MASK;

    /*    LUT3[1] setting       */
    /* Configure LUT3 [1] */
    /*****************************/
    /* Lut configuration for output, check description above */
    lutCfgLut1.opcode = LUTx_LOGIC_OPCODE;
    lutCfgLut1.lutMap = LUTx_INV_OUT_MAP;

    /* Lut configuration for input */
    lutCfgLut1.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT7_OUT;
    lutCfgLut1.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT7_OUT;
    lutCfgLut1.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT7_OUT;
    smart_io_cfg.lutCfg[LUT_INV_OUT_PIN] = &lutCfgLut1;

    /*    LUT3[7] setting       */
    /* Configure LUT3 [7] */
    /*****************************/
    /* Lut configuration for output, check description above */
    lutCfgLut7.opcode = LUTx_LOGIC_OPCODE;
    lutCfgLut7.lutMap = LUTx_OUT_MAP;
```

```
    /* Lut configuration for input (button) */
    lutCfgLut7.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_IO;
    lutCfgLut7.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_IO;
    lutCfgLut7.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_IO;
    smart_io_cfg.lutCfg[LUT_IP_BUTTON_PIN] = &lutCfgLut7;

    /* Initialization of Smart IO structure */
    /* Configure Smart I/O. See Code Listing 7. */
    retStatus = Cy_SmartIO_Init(SMART_IO_PORT, &smart_io_cfg);
    return retStatus;
}
```

Code Listing 6 Cy_SmartIO_Enable() 関数

```
void Cy_SmartIO_Enable(volatile stc_SMARTIO_PRT_t* base)
{
    un_SMARTIO_PRT_CTL_t workCTL = base->unCTL;
    workCTL.stcField.u1ENABLED     = CY_SMARTIO_ENABLE;
    workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_DISABLE;
    base->unCTL.u32Register        = workCTL.u32Register;   /* (8) Enable Smart I/O. */
}
```

4 設定例

Code Listing 7 Cy_SmartIO_Init() 関数

```
cy_en_smartio_status_t Cy_SmartIO_Init(volatile stc_SMARTIO_PRT_t* base, const
cy_stc_smartio_config_t* config)
{
    cy_en_smartio_status_t status = CY_SMARTIO_SUCCESS;

    if(NULL != config)
    {
        /* (4) Set clock source and bypass to Smart */
        un_SMARTIO_PRT_CTL_t workCTL = {.u32Register = 0ul};
        workCTL.stcField.u1ENABLED     = CY_SMARTIO_DISABLE;
        workCTL.stcField.u1HLD_OVR     = config->hldOvr;
        workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_ENABLE;
        workCTL.stcField.u5CLOCK_SRC   = config->clkSrc;
        workCTL.stcField.u8BYPASS      = config->bypassMask;
        base->unCTL.u32Register        = workCTL.u32Register;

        /* (5) Set synchronizer to Smart IO */
        un_SMARTIO_PRT_SYNC_CTL_t workSYNC_CTL = {.u32Register = 0ul};
        workSYNC_CTL.stcField.u8IO_SYNC_EN   = config->ioSyncEn;
        workSYNC_CTL.stcField.u8CHIP_SYNC_EN = config->chipSyncEn;
        base->unSYNC_CTL.u32Register         = workSYNC_CTL.u32Register;

        /* LUT configurations - skip if lutCfg is a NULL pointer */
        /* (6), (7) Set LUT3 */
        for(uint32_t i = CY_SMARTIO_LUTMIN; i < CY_SMARTIO_LUTMAX; i++)
        {
            if(NULL != config->lutCfg[i])
            {
                un_SMARTIO_PRT_LUT_SEL_t workLUT_SET = { .u32Register = 0ul };
                workLUT_SET.stcField.u4LUT_TR0_SEL = config->lutCfg[i]->tr0;
                workLUT_SET.stcField.u4LUT_TR1_SEL = config->lutCfg[i]->tr1;
                workLUT_SET.stcField.u4LUT_TR2_SEL = config->lutCfg[i]->tr2;
                base->unLUT_SEL[i].u32Register     = workLUT_SET.u32Register;

                un_SMARTIO_PRT_LUT_CTL_t workLUT_CTL = { .u32Register = 0ul };
                workLUT_CTL.stcField.u2LUT_OPC = config->lutCfg[i]->opcode;
                workLUT_CTL.stcField.u8LUT     = config->lutCfg[i]->lutMap;
                base->unLUT_CTL[i].u32Register = workLUT_CTL.u32Register;
            }
        }

        /* DU Configuration - skip if duCfg is a NULL pointer */
        /* Set DU. It is ignored in this use case. */
        if(NULL != config->duCfg)
        {
            un_SMARTIO_PRT_DU_SEL_t workDU_SEL = {.u32Register = 0ul};
            workDU_SEL.stcField.u4DU_TR0_SEL    = config->duCfg->tr0;
            workDU_SEL.stcField.u4DU_TR1_SEL    = config->duCfg->tr1;
            workDU_SEL.stcField.u4DU_TR2_SEL    = config->duCfg->tr2;
            workDU_SEL.stcField.u2DU_DATA0_SEL = config->duCfg->data0;
            workDU_SEL.stcField.u2DU_DATA1_SEL = config->duCfg->data1;
```

```
            base->unDU_SEL.u32Register         = workDU_SEL.u32Register;

            un_SMARTIO_PRT_DU_CTL_t workDU_CTL = {.u32Register = 0ul};
            workDU_CTL.stcField.u3DU_SIZE = config->duCfg->size;
            workDU_CTL.stcField.u4DU_OPC  = config->duCfg->opcode;
            base->unDU_CTL.u32Register    = workDU_CTL.u32Register;

            base->unDATA.stcField.u8DATA = config->duCfg->dataReg;
        }
    }
    else
    {
        status = CY_SMARTIO_BAD_PARAM;
    }

    return(status);
}
```

4.2 リセット検出/安定回路のユースケース

ここではスマート I/O にリセット検出/安定回路を実装する方法について説明します。図 14 にリセット検出/安定性の動作を示します。

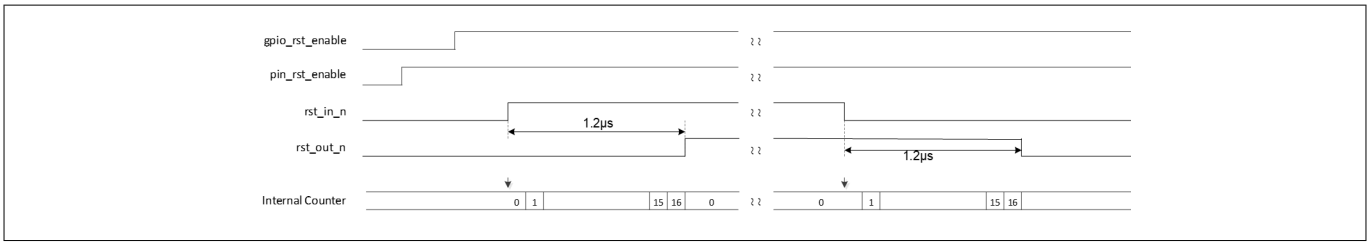


図 14 リセット検出/安定回路の動作

このユースケースでは回路は pin_rst_enable と gpio_rst_enable の 2 つのイネーブル信号を備えています。pin_rst_enable は外部回路からのイネーブル信号で、gpio_rst_enable はソフトウェアによるイネーブル制御信号です。両方の信号をイネーブルにすると回路はアクティブになります。

rst_in_n はアクティブハイの外部リセット入力で、rst_out_n はアクティブハイのリセット出力です。回路は rst_in_n をモニタリングします。rst_in_n が特定の連続サイクル数の間アクティブになると rst_out_n を出力します。CLOCK_SRC [12:8] で選択された動作クロックが 16 サイクル連続して入力されるとリセットがアクティブまたは解除します。ソースクロック 80 MHz は 6 分周して 13 MHz になります。そして 76 ns を 16 サイクルカウントするとリセットアクティブされ、そのリリース時間は約 1.2 µs です

下記に、使用される I/O ポートと HSIOM 信号を示します。

- io_data_in [6] = pin_rst_enable; (I/O ポートから)
- io_data_in [7] = rst_in_n; (I/O ポートから)
- smartio_data [5] = rst_out_n; (I/O ポートへ)
- chip_data [4] = gpio_rst_enable; (HSIOM から)

図 15 に本回路の各 LUT3 [3:0] と DU について、その接続と機能ロジックを示します。

4 設定例

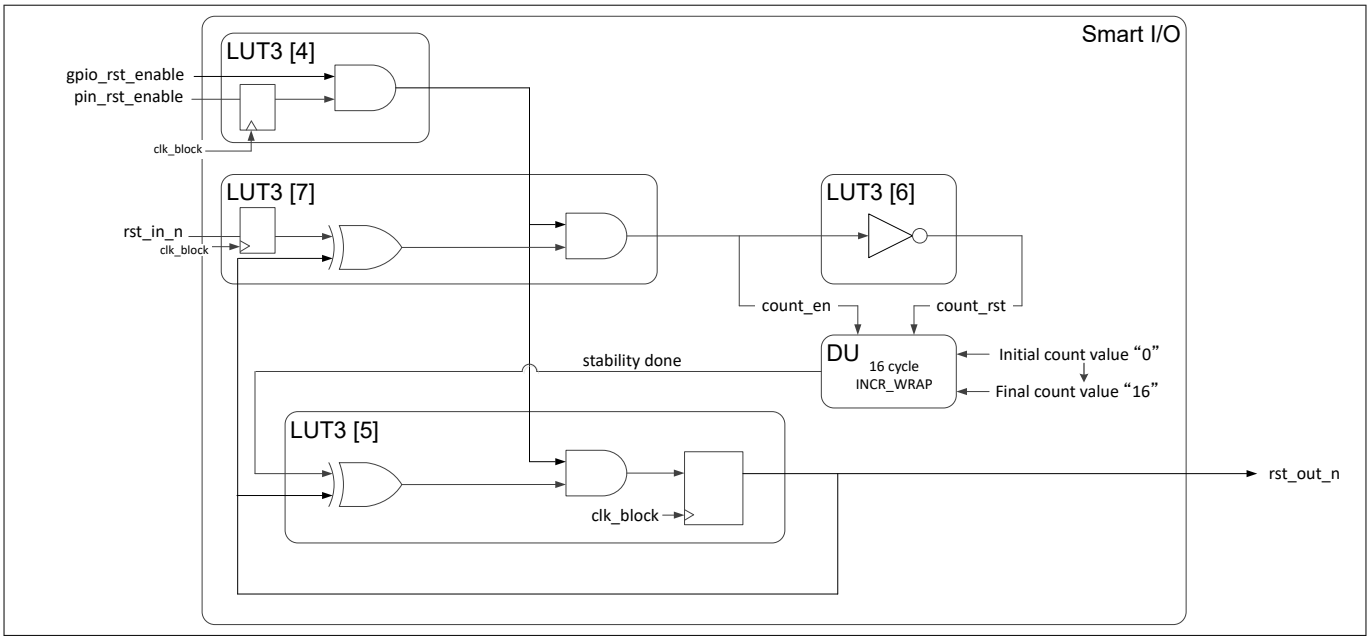


図 15　　リセット検出/安定回路の論理例

このユースケースでは 4 つの LUT3 と 1 つの DU を使用します。

LUT3 [4] は 2 つの信号 (pin_rst_enable と gpio_rst_en) からこの回路のイネーブル信号を生成するために使用されます。LUT 3 [6] と LUT 3 [7] は rst_in_n 状態を監視し DU のカウンタを起動するために使用されます。LUT 3 [5] は安定待ち完了を検出して rst_out_n を出力します。

DU はリセット安定待ち時間を生成するために使用され、LUT3 [5]の Tr_out はゲート出力モードに同期して出力します。

図 16 に、このユースケースの信号パスを示します。

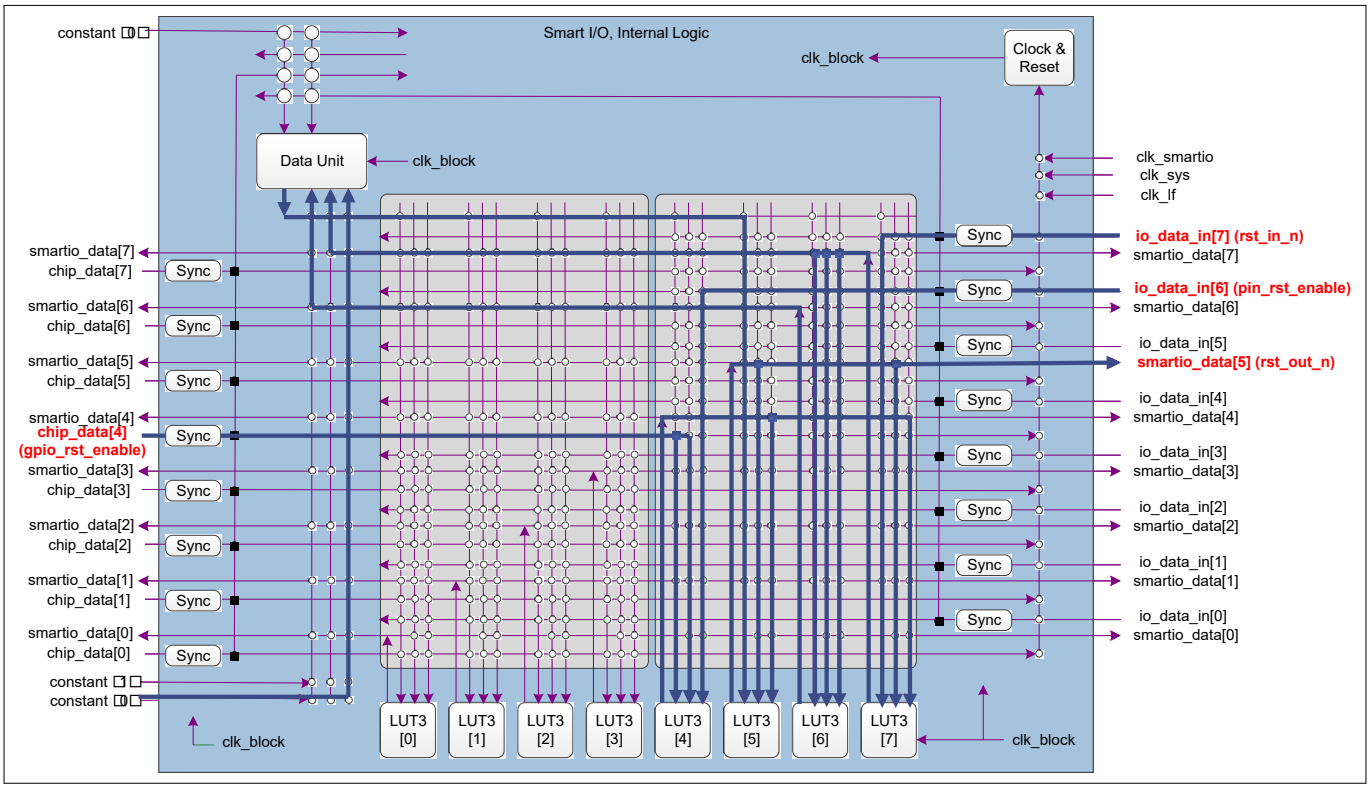


図 16　　リセット検出/安定回路の信号パス

4 設定例

このユースケースでは、入出力信号として io_data_in [7:6], chip_data [4], および smartio_data [5]を使用します。したがって、4 つの LUT3 (LUT3 [7:4]) で設定できます。rst_out_n に smartio_data [3] を使用する場合は、LUT3 [3] を経由する必要があります。すなわち、このケースでは 5 つの LUT3 [x] が必要です。

表 17, 表 18, 表 19, および表 20 に各 LUT3 の真理値表を示します。表の太字部分は無効な組合せパターンを示します。

表 17　　ルックアップテーブル LUT3 [6]

Tr2_in	Tr1_in	Tr0_in	Tr_out
0	0	0	1
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**1**
1	**0**	**0**	**1**
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	1	1	0

表 18　　ルックアップテーブル LUT3 [7]

Tr2_in	Tr1_in	Tr0_in	Tr_out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

表 19　　ルックアップテーブル LUT3 [5]

Tr2_in	Tr1_in	Tr0_in	Tr_out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

4 設定例

表 20 ルックアップテーブル LUT3 [4]

Tr2_in	Tr1_in	Tr0_in	Tr_out
0	0	0	0
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	1	1	0
1	0	0	0
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	1	1	1

LUT3 [6] は 1 入力 1 出力のインバータ回路であり、Tr0_in, Tr1_in および Tr2_in の各入力は同じ信号です。したがって、[Tr0_in, Tr1_in, Tr2_in, Tr_out] の有効な組合せパターンは[0, 0, 0, 1] または[1, 1, 1, 0] です。無効なパターンが発生するとカウンタ回路がリセットされ rst_out_n は現在の値を維持します。

LUT3 [7] には 3 つの異なる入力があります。LUT3 [4] からのイネーブル信号が有効 (=”1”) で rst_in_n 状態 (Tr2_in) が LUT3 [5] からの rst_out_n 状態 (Tr1_in) と異なる場合、“1”を出力します。

LUT3 [5] は rst_out_n 信号を生成します。LUT3 [4] からのイネーブル信号 (Tr2_in) が有効 (=”1”) で DU からの安定待ち完了信号 (Tr0_in) が検出された場合 (安定待ち時間が経過した場合) 、現在の LUT3 [5]の rst_out_n 信号 (Tr1_in、Tr_out) は反転します。

LUT3 [4] はこの回路のイネーブル信号を生成します。pin_rst_enable と gpio_rst_enabl の 2 つの入力があります。pin_rst_enable は Tr2_in に入力され、gpio_rst_enable は Tr0_in と Tr1_in に入力されます。したがって、Tr0_in と Tr1_in の異なる値の組合せは無効なパターンになります。無効なパターンが発生すると回路はディセーブル (Tr_out = "0") になります。

DU は INCR_WRAP モードで動作します。このモードでは初期値 (DATA 0) から最終値 (DATA 1) に達するまでデータを 1 ずつインクリメントします。カウント値が最終値と一致すると、DATA 0 に戻ります。rst が"1"の場合、カウンタ値は初期値に設定されます。

このモードでは、DU は 2 つの制御信号入力 (count_en および count_rst) があります。count_en は Tr1_in に入力され、count_rst は Tr0_in に入力されます。また、DU には 2 つのカウント制御レジスタ (DATA 0 および DATA 1) および 1 つの出力信号 (Tr_out) があります。DATA0 レジスタはカウント初期値、DATA1 レジスタは最終カウント値です。

表 21 に DU の設定と入出力動作を示します。

表 21 DU の動作

Tr0_in (rst)	Tr1_in (en)	動作	DATA 0 (初期値)	DATA 1 (最終値)	Tr_out
1	0	初期値 (DATA 0) から最終値 (DATA 1) に達するまでデータを 1 ずつインクリメントします。 カウント値が最終値と一致すると DATA 0 に戻ります。	0	16	0 (リセット状態)
0	1				カウント値が最終値と一致するとシングルクロックパルスを出力します。

4 設定例

Tr0_in は“rst”として動作し Tr1_in は“en”として動作します。Tr0_in は LUT3 [6]の出力に接続し Tr1_in は LUT3 [6] の入力に接続します。en に"1"を入力すると DU はカウンタを開始します。その後カウント値が最終値に達するとシングルパルスを出力します。

図 17 に、スマート I/O の設定手順を示します。

4 設定例

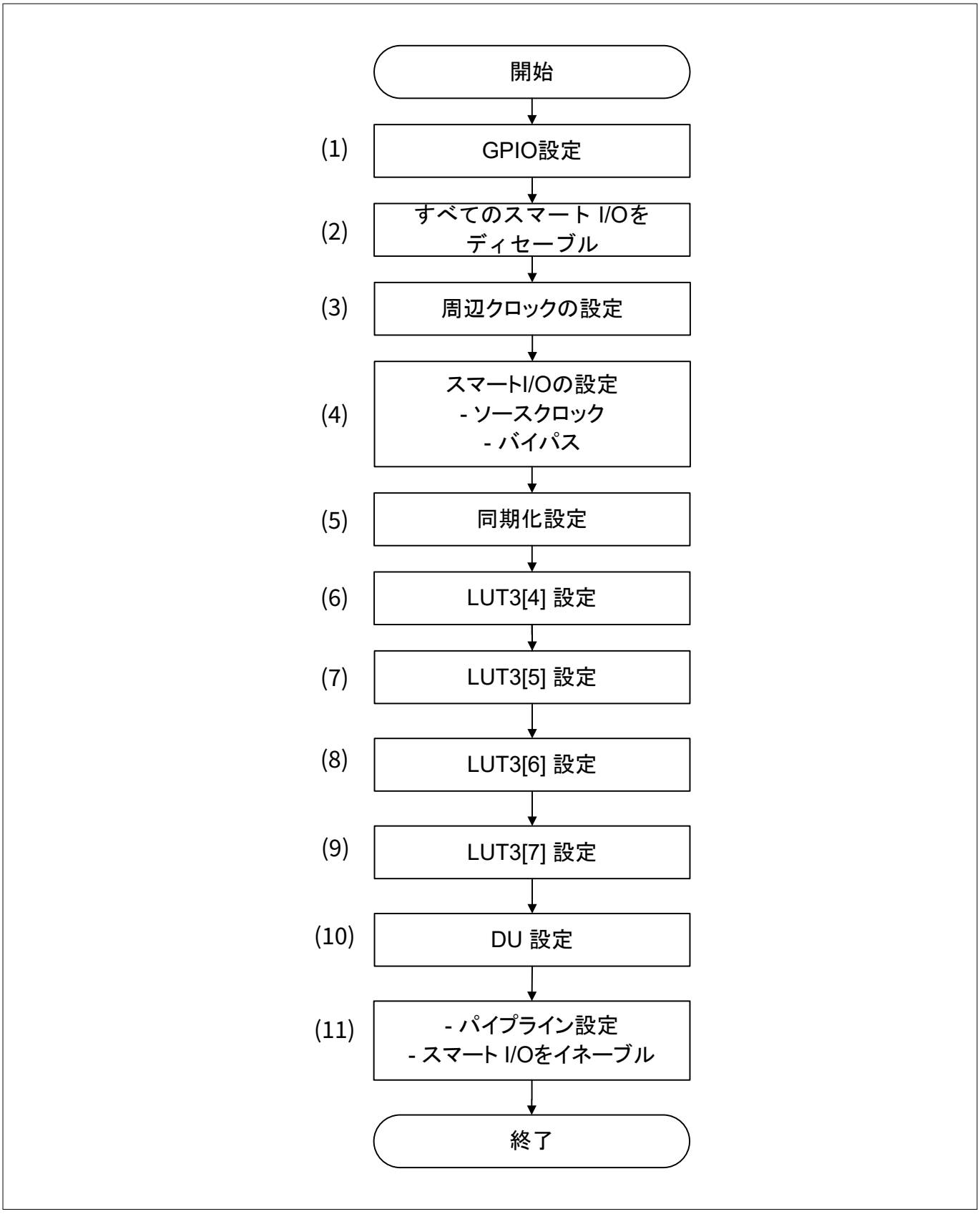


図 17　　スマート I/O の設定手順

(1) GPIO 設定についてのアプリケーションノート (AN220193) を参照してください。

(6) 設定パターンについては表 20 を参照してください。

(7) 設定パターンについては表 19 を参照してください。
(8) 設定パターンについては表 17 を参照してください。
(9) 設定パターンについては表 18 を参照してください。

4.2.1 設定とサンプルコード

表 22 に SDL のスマート I/O 設定部のパラメータを示し、表 23 に関数を示します。

表 22 スマート I/O の設定パラメーター覧

パラメーター	説明	値
SMART_IO_CLK_ACTIVE	スマート I/O クロックソース定義	1ul (アクティブなクロックソースを選択)
CY_SMARTIO_CLK_INV	スマート I/O クロック定義	PCLK_SMARTIO13_CLOCK
SMART_IO_PORT	スマート IO ポート定義	SMARTIO_PRT13 (スマート I/O のポート 13 に割当て)
SMARTIO_BYPASS_CH_MASK	バイパスチャネルマスクを定義 io_data_in [7] to smartio_data [7]: バイパスなし io_data_in [6] to smartio_data [6]: バイパスなし io_data_in [5] to smartio_data [5]: バイパスなし io_data_in [4] to smartio_data [4]: バイパスなし io_data_in [3] to smartio_data [3]: バイパス io_data_in [2] to smartio_data [2]: バイパス io_data_in [1] to smartio_data [1]: バイパス io_data_in [0] to smartio_data [0]: バイパス	0x0Ful (表 1 参照) io_data_in [3] から to io_data_in [0] はスマート I/O では未使用
SMARTIO_IOSYNC_CH_MASK	IO sync チャネルマスク定義	0x00ul (表 3 参照)
GPIO_RST_EN_PORT	LUT3[4] 入力の入力ポート定義	GPIO_PRT13 (GPIO ポート 13 に割当て)
GPIO_RST_EN_PIN	LUT3[4] 入力の入力ポートピン定義	4ul (GPIO ポート 4pin に割当て)
GPIO_RST_EN_PIN_MUX	入力ポートピン機能を設定	P13_4_GPIO (GPIO に割当て)

(続く)

表 22 (続き) スマート I/O の設定パラメーター一覧

パラメーター	説明	値
PIN_RST_EN_PORT	LUT3[6] 入力の入力ポート定義	GPIO_PRT13 (GPIO ポート 13 に割当て)
PIN_RST_EN_PIN	LUT3[6] 入力の入力ポートピン定義	6ul (GPIO ポート 6 ピンに割当て)
PIN_RST_EN_PIN_MUX	入力ポートピン機能を設定	P13_6_GPIO (GPIO に割当て)
RST_IN_PORT	LUT3[7] 入力の入力ポート定義	GPIO_PRT13 (GPIO ポート 13 に割当て)
RST_IN_PIN	LUT3[7] 入力の入力ポートピン定義	7ul (GPIO ポート 6 ピンに割当て)
RST_IN_PIN_MUX	入力ポートピン機能を設定	P13_7_GPIO (GPIO に割当て)
RST_OUT_PORT	LUT3[5] 出力の入力ポート設定	GPIO_PRT13 (GPIO ポート 13 に割当て)
RST_OUT_PIN	LUT3[5] 出力の入力ポートピン設定	5ul (GPIO ポート 6 ピンに割当て)
RST_OUT_PIN_MUX	入力ポートピン機能を設定	P13_5_GPIO (GPIO に割当て)
LUT4_OUT_MAP	LUT3[4] 出力パターン	0x80ul (表 20 参照)
LUT5_OUT_MAP	LUT3[5] 出力パターン	0x60ul (表 19 参照)
LUT6_OUT_MAP	LUT3[6] 出力パターン	0x7Ful (表 17 参照)
LUT7_OUT_MAP	LUT3[7] 出力パターン	0x28ul (表 18 参照)
LUTx_LOGIC_OPCODE_COMB	LUT3 動作モードの選択	CY_SMARTIO_LUTOPC_COMB (Combinatorial に割当て)
LUTx_LOGIC_OPCODE_GO	LUT3 動作モードの選択	CY_SMARTIO_LUTOPC_GATED_OUT
LUTx_LOGIC_OPCODE_GI2	LUT3 動作モードの選択	CY_SMARTIO_LUTOPC_GATED_TR2
CY_SYSCLK_DIV_16_BIT	分周器タイプを 16 ビット分周器に選択	1ul
CY_SMARTIO_ENABLE	スマート I/O とパイプラインをイネーブルに設定	1ul
CY_SMARTIO_DISABLE	スマート I/O とパイプラインをティセーブルに設定	0ul
CY_SMARTIO_DEINIT	スマート I/O をデフォルト値にリセット	0ul
CY_SMARTIO_CHANNEL_ALL	すべてのピンにスマート I/O バイパスを設定	0xfful
CY_SMARTIO_CLK_DIVACT	clk_smartio/rst_sys_act_n へのソースクロックを選択。表 2 を参照してください。	16ul

(続く)

表 22 (続き) スマート I/O の設定パラメーター一覧

パラメーター	説明	値
CY_SMARTIO_CLK_GATED	ソースクロックをクロックソース定数「0」に選択。表 2 を参照してください。	20ul
CY_SMARTIO_CLK_ASYNC	ソースクロックを非同期モード/1 に選択。表 2 を参照してください。	31ul
CY_SMARTIO_LUTTR_CHIP4	LUT3[4] Tr0/Tr1 入力	4ul (LUT3[4] に割当て。表 6 参照)
CY_SMARTIO_LUTTR_IO6	LUT3[4] Tr2 入力	14ul (LUT3[4] に割当て。表 6 参照)
CY_SMARTIO_LUTTR_DU_OUT	LUT3[5] Tr0 入力	0ul (LUT3[5] に割当て。表 6 参照)
CY_SMARTIO_LUTTR_LUT5_OUT	LUT3[5] Tr1 入力/LUT3[7] Tr1 入力	5ul (LUT3[5] に割当て。表 6 参照)
CY_SMARTIO_LUTTR_LUT4_OUT	LUT3[5] Tr2 入力/ LUT3[7] Tr2 入力	4ul (LUT3[5]に割当て。表 6 参照)
CY_SMARTIO_LUTTR_LUT7_OUT	LUT3[6] Tr0/Tr1/Tr2 入力	7ul (LUT3[6] に割当て。表 6 参照)
CY_SMARTIO_LUTTR_IO7	LUT3[7] Tr0 入力	15ul (LUT3[7] に割当て。表 6 参照)
CY_SMARTIO_DUTR_LUT6_OUT	DU Tr0 入力トリガソース	9ul (LUT6 出力に割当て。表 9 参照)
CY_SMARTIO_DUTR_LUT7_OUT	DU Tr1 入力トリガソース	10ul (LUT7 出力に割当て。表 9 参照)
CY_SMARTIO_DUTR_ZERO	DU Tr2 入力トリガソース	0ul (Constant 0 に割当て。表 9 参照)
CY_SMARTIO_DUDATA_ZERO	DU data0 入力 DATA ソース	0ul (Constant 0 に割当て。表 10 参照)
CY_SMARTIO_DUDATA_DATAREG	DU data1 入力 DATA ソース	3ul (SMARTIO.DATA レジスタに割当て。表 10 参照)
CY_SMARTIO_DUOPC_INCR_WRAP	DU オペコード	3ul (Increment および wrap-around (Count up and wrap) 参照)表 11 参照)
CY_SMARTIO_DUSIZE_8	DU 動作ビットサイズ	7ul (8 ビットサイズ/幅に割当て。表 10 参照)
smart_io_cfg.clkSrc	ソースクロック設定	CY_SMARTIO_CLK_DIVACT
smart_io_cfg.bypassMask	バイパス設定	SMARTIO_BYPASS_CH_MASK
smart_io_cfg.ioSyncEn	同期化設定	SMARTIO_IOSYNC_CH_MASK
lutCfgLut4.opcode	LUT3[4] 動作モード設定	LUTx_LOGIC_OPCODE_GI2
lutCfgLut4.lutMap	LUT3[4] 出力パターンを設定	LUT4_OUT_MAP
lutCfgLut4.tr0	LUT3[4] tr0 入力を設定	CY_SMARTIO_LUTTR_CHIP4
lutCfgLut4.tr1	LUT3[4] tr1 入力を設定	CY_SMARTIO_LUTTR_CHIP4

(続く)

表 22 (続き) スマート I/O の設定パラメーター一覧

パラメーター	説明	値
lutCfgLut4.tr2	LUT3[4] tr2 入力を設定	CY_SMARTIO_LUTTR_IO6
lutCfgLut5.opcode	LUT3[5] 動作モード設定	LUTx_LOGIC_OPCODE_GO
lutCfgLut5.lutMap	LUT3[5] 出力パターンを設定	LUT5_OUT_MAP
lutCfgLut5.tr0	LUT3[5] tr0 入力を設定	CY_SMARTIO_LUTTR_DU_OUT
lutCfgLut5.tr1	LUT3[5] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT5_OUT
lutCfgLut5.tr2	LUT3[5] tr2 入力を設定	CY_SMARTIO_LUTTR_LUT4_OUT
lutCfgLut6.opcode	LUT3[6] 動作モード設定	LUTx_LOGIC_OPCODE_COMB
lutCfgLut6.lutMap	LUT3[6] 出力パターンを設定	LUT6_OUT_MAP
lutCfgLut6.tr0	LUT3[6] tr0 入力を設定	CY_SMARTIO_LUTTR_LUT7_OUT
lutCfgLut6.tr1	LUT3[6] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT7_OUT
lutCfgLut6.tr2	LUT3[6] tr2 入力を設定	CY_SMARTIO_LUTTR_LUT7_OUT
lutCfgLut7.opcode	LUT3[7] 動作モード設定	LUTx_LOGIC_OPCODE_GI2
lutCfgLut7.lutMap	LUT3[7] 出力パターンを設定	LUT7_OUT_MAP
lutCfgLut7.tr0	LUT3[7] tr0 入力を設定	CY_SMARTIO_LUTTR_LUT4_OUT
lutCfgLut7.tr1	LUT3[7] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT5_OUT
lutCfgLut7.tr2	LUT3[7] tr2 入力を設定	CY_SMARTIO_LUTTR_IO7
lutCfgDu.tr0	DU 入力トリガ 0 ソース選択設定 - LUT[3]6 出力	CY_SMARTIO_DUTR_LUT6_OUT
lutCfgDu.tr1	DU 入力トリガ 1 ソース選択設定- LUT[3]7 出力	CY_SMARTIO_DUTR_LUT7_OUT
lutCfgDu.tr2	DU 入力トリガ 2 ソース選択設定- 定数 0	CY_SMARTIO_DUTR_ZERO
lutCfgDu.data0	DU 入力 DATA0 ソース選択- 固定 0	CY_SMARTIO_DUDATA_ZERO
lutCfgDu.data1	DU 入力 DATA1 ソース選択- SMARTIO_PRTx_DATA.DATA [7:0]	CY_SMARTIO_DUDATA_DATAREG
lutCfgDu.opcode	DU オペコード	CY_SMARTIO_DUOPC_INCR_WRAP
lutCfgDu.size	DU 幅サイズは 8	CY_SMARTIO_DUSIZE_8
lutCfgDu.dataReg	DU DATA レジスタ値 = 16	10ul
sourceFreq	ソース周波数	80000000ul (80MHz)
targetFreq	目標周波数	12000000ul (12MHz)

4 設定例

表 23　　スマート I/O の設定関数一覧

関数	説明	備考
Init_IO_Pin()	GPIO ポートピン設定	–
Cy_SmartIO_Deinit()	スマート I/O をデフォルト値にリセット	以下のスマート I/O レジスタをリセット; SMARTIO_PRTx_CTL, SMARTIO_PRTx_SYNC_CTL, SMARTIO_PRTx_LUT_SELy, SMARTIO_PRTx_LUT_CTLy, SMARTIO_PRTx_DU_SEL, SMARTIO_PRTx_DU_CTL および SMARTIO_PRTx_DATA
Init_SmartIO()	スマート I/O 初期化とイネーブル設定 Init_SmartIO_Cfg() および Cy_SmartIO_Enable()の呼び出し	–
Init_SmartIO_Cfg()	スマート I/O の各種設定および Cy_SmartIO_Init()の呼び出し	–
Cy_SmartIO_Enable()	スマート I/O のイネーブル	PIPELINE_EN および ENABLED ビットに書込み
Cy_SmartIO_Init()	スマート I/O のレジスタ設定	ソースクロック, バイパス, 同期化, LUT3, および DU の関連レジスタに書込み

Code Listing 8 は、リセット検出/安定回路のプログラムの例を示します。GPIO とクロック設定については Architecture Reference Manual と アプリケーションノート を参照してください。

以下の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- base はスマート I/O レジスタのベースアドレスのポインタを示します。
- **base->unLUT_SEL[idx].u32Register** は、Registers Reference Manual で言及されている SMARTIO_PRTx_LUT_SEL[idx] レジスタです。他のレジスタも同様です。“**x**”はポートサフィックス番号、“**idx**”はレジスタインデックス番号を表します。
- レジスタ設定のパフォーマンス向上のため、SDL では完全な 32 ビットデータをレジスタに書き込みます。各ビットフィールドが生成後、最終的な 32 ビットデータとしてレジスタに書き込まれます。

```
un_SMARTIO_PRT_CTL_t workCTL= {.u32Register = 0ul};
workCTL.stcField.u1ENABLED     = CY_SMARTIO_DISABLE;
workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_ENABLE;
workCTL.stcField.u5CLOCK_SRC   = CY_SMARTIO_CLK_GATED;
workCTL.stcField.u8BYPASS      = CY_SMARTIO_CHANNEL_ALL;
base->unCTL.u32Register        = workCTL.u32Register;
```

レジスタの表記の結合および構造表現の詳細については、`hdr/rev_x/ip` の `cyip_smartio_v2.h` を参照してください。

4 設定例

Code Listing 8 リセット検出/安定回路の例

```
/* Smart IO clock source selection */
/* Define Clock active */
#define SMART_IO_CLK_ACTIVE             1ul

/* Smart IO port selections macro */
/* Define Smart I/O port */
#define SMART_IO_PORT                   SMARTIO_PRT13

/* Define Smart I/O Clock */
#define CY_SMARTIO_CLK_INV              PCLK_SMARTIO13_CLOCK

/* Bypass channel mask */
/* Define Smart I/O bypass channel */
#define SMARTIO_BYPASS_CH_MASK          0x0Ful /* 00001111: 0000 means [7:4] are not bypassed
i.e. programmable SMARTIO is exposed */

/* IO sync channel mask */
/* Define Smart I/O sync channel */
#define SMARTIO_IOSYNC_CH_MASK          0x00ul

/* Lut pin configuration */
/* Define input port to LUT3[4] */
#define GPIO_RST_EN_PORT                GPIO_PRT13
#define GPIO_RST_EN_PIN                 4ul
#define GPIO_RST_EN_PIN_MUX             P13_4_GPIO /* Check signal at BB JP6.7 */

/* Define input port to LUT3[6] */
#define PIN_RST_EN_PORT                 GPIO_PRT13
#define PIN_RST_EN_PIN                  6ul
#define PIN_RST_EN_PIN_MUX              P13_6_GPIO /* Check signal at BB JP11.14 */

/* Define input port to LUT3[7] */
#define RST_IN_PORT                     GPIO_PRT13
#define RST_IN_PIN                      7ul
#define RST_IN_PIN_MUX                  P13_7_GPIO /* Check signal at BB JP11.13 */

/* Define output port to LUT3[5] */
#define RST_OUT_PORT                    GPIO_PRT13
#define RST_OUT_PIN                     5ul
#define RST_OUT_PIN_MUX                 P13_5_GPIO /* Check signal at BB JP6.6 */

/* LUT output map */
/* Define LUT3[4], LUT3[5], LUT3[6] and LUT3[7] output pattern */
#define LUT4_OUT_MAP                    0x80ul
#define LUT5_OUT_MAP                    0x28ul
#define LUT6_OUT_MAP                    0x7Ful
#define LUT7_OUT_MAP                    0x28ul

/* LUT logic circuit type macro */
/* Define LUT logic circuit type */
#define LUTx_LOGIC_OPCODE_COMB          CY_SMARTIO_LUTOPC_COMB
```

4 設定例

```
#define LUTx_LOGIC_OPCODE_GO            CY_SMARTIO_LUTOPC_GATED_OUT
#define LUTx_LOGIC_OPCODE_GI2           CY_SMARTIO_LUTOPC_GATED_TR2

#define CY_SMARTIO_ENABLE  1ul
#define CY_SMARTIO_DISABLE 0ul
#define CY_SMARTIO_DEINIT  0ul
#define CY_SMARTIO_CHANNEL_ALL  0xfful   /**< All channels */

/* Port pin configuration */
/*******************************************************************************/

/* Configure Port for output (Port13 pin) */
cy_stc_gpio_pin_config_t gpio_rst_cfg =
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_STRONG_IN_OFF, /* SmartIO from CPU */
    .hsiom     = GPIO_RST_EN_PIN_MUX,      /* P13_4_GPIO */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};

/* Configure Port for input (Port13 pin) */
cy_stc_gpio_pin_config_t pin_rst_cfg =
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_HIGHZ,         /* CPU from SmartIO */
    .hsiom     = PIN_RST_EN_PIN_MUX,       /* P13_6_GPIO */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};

/* Configure Port for input (Port13 pin) */
cy_stc_gpio_pin_config_t rst_in_cfg =
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_HIGHZ,         /* CPU from SmartIO */
    .hsiom     = RST_IN_PIN_MUX,           /* P13_7_GPIO */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};

/* Configure Port for output (Port13 pin) */
cy_stc_gpio_pin_config_t rst_out_cfg =
{
```

```
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_STRONG_IN_OFF, /* CPU from SmartIO */
    .hsiom     = RST_OUT_PIN_MUX,          /* P13_5_GPIO */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};


int main(void)
{
    :
    Init_IO_Pin();   /* (1) Configure GPIO pin. See Code Listing 9. */

    /* Deinit before Init */
    Cy_SmartIO_Deinit(SMART_IO_PORT);   /* Disable all Smart I/O. See Code Listing 10. */

    /* SmartIO peripheral clock divider setting */
    /* (3) Configure peripheral Clock */
    {
        Cy_SysClk_PeriphAssignDivider(CY_SMARTIO_CLK_INV, CY_SYSCLK_DIV_16_BIT, 0ul);
        uint32_t sourceFreq = 80000000ul;
        uint32_t targetFreq = 12000000ul;
        uint32_t divNum = (sourceFreq / targetFreq);

        Cy_SysClk_PeriphSetDivider(CY_SYSCLK_DIV_16_BIT, 0ul, (divNum - 1ul));
        Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_16_BIT, 0ul);
    }

    Init_SmartIO();    /* Initialize Smart I/O. See Code Listing 11. */

    Cy_SysEnableApplCore(CY_CORTEX_M4_APPL_ADDR);

    Cy_GPIO_Clr(GPIO_PRT13, 4ul);

    while(1)
    {

    }
}
```

4 設定例

Code Listing 9 Init_IO_Pin() 関数

```
void Init_IO_Pin(void)
{
    /* Please check ReadMe.txt for proper connection of Input and Output */
    /* Configure Port13 4pin. */
    Cy_GPIO_Pin_Init(GPIO_RST_EN_PORT, GPIO_RST_EN_PIN, &gpio_rst_cfg);

    /* Configure Port13 pin */
    Cy_GPIO_Pin_Init(PIN_RST_EN_PORT, PIN_RST_EN_PIN, &pin_rst_cfg);

    /* Configure Port13 pin */
    Cy_GPIO_Pin_Init(RST_IN_PORT, RST_IN_PIN, &rst_in_cfg);

    /* Configure Port13 pin */
    Cy_GPIO_Pin_Init(RST_OUT_PORT, RST_OUT_PIN, &rst_out_cfg);
}
```

Code Listing 10 Cy_SmartIO_Deinit() 関数

```
void Cy_SmartIO_Deinit(volatile stc_SMARTIO_PRT_t* base)
{
    un_SMARTIO_PRT_CTL_t workCTL= {.u32Register = 0ul};
    workCTL.stcField.u1ENABLED     = CY_SMARTIO_DISABLE;   /* (2) Disable all Smart I/O port */
    workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_ENABLE;
    workCTL.stcField.u5CLOCK_SRC   = CY_SMARTIO_CLK_GATED;
    workCTL.stcField.u8BYPASS      = CY_SMARTIO_CHANNEL_ALL;
    base->unCTL.u32Register        = workCTL.u32Register;

    base->unSYNC_CTL.u32Register = CY_SMARTIO_DEINIT;
    for(uint8_t idx = CY_SMARTIO_LUTMIN; idx < CY_SMARTIO_LUTMAX; idx++)
    {
        base->unLUT_SEL[idx].u32Register = CY_SMARTIO_DEINIT;
        base->unLUT_CTL[idx].u32Register = CY_SMARTIO_DEINIT;
    }
    base->unDU_SEL.u32Register = CY_SMARTIO_DEINIT;
    base->unDU_CTL.u32Register = CY_SMARTIO_DEINIT;
    base->unDATA.u32Register = CY_SMARTIO_DEINIT;
}
```

4 設定例

Code Listing 11 Init_SmartIO() 関数

```
void Init_SmartIO(void)
{
    cy_en_smartio_status_t retStatus = (cy_en_smartio_status_t)0xFF;

/* Configure Smart I/O. See Code Listing 12. */
    retStatus = Init_SmartIO_Cfg();
    if(retStatus == CY_SMARTIO_SUCCESS)
    {
        /* After all the configuration, enable SMART IO */
        /* Enable Smart I/O. See Code Listing 13. */
        Cy_SmartIO_Enable(SMART_IO_PORT);
    }
}
```

4 設定例

Code Listing 12 Init_SmartIO_Cfg() 関数

```
cy_en_smartio_status_t Init_SmartIO_Cfg(void)
{
    cy_stc_smartio_ducfg_t lutCfgDu;
    cy_stc_smartio_lutcfg_t lutCfgLut4;
    cy_stc_smartio_lutcfg_t lutCfgLut5;
    cy_stc_smartio_lutcfg_t lutCfgLut6;
    cy_stc_smartio_lutcfg_t lutCfgLut7;

    cy_stc_smartio_config_t smart_io_cfg;
    cy_en_smartio_status_t retStatus = (cy_en_smartio_status_t)0xFF;

    /* initialize the Smart IO structure */
    /* Clear configuration structure */
    memset(&lutCfgDu, 0ul, sizeof(cy_stc_smartio_ducfg_t));
    memset(&lutCfgLut4, 0ul, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&lutCfgLut5, 0ul, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&lutCfgLut6, 0ul, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&lutCfgLut7, 0ul, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&smart_io_cfg, 0ul, sizeof(cy_stc_smartio_config_t));

#ifdef SMART_IO_CLK_ACTIVE
    /* Active clock source is selected */
    /* Configure Smart I/O clock source */
    smart_io_cfg.clkSrc = (cy_en_smartio_clksrc_t)CY_SMARTIO_CLK_DIVACT;
#else
    /* Asynchronous clock source is selected */
    smart_io_cfg.clkSrc = (cy_en_smartio_clksrc_t)CY_SMARTIO_CLK_ASYNC;
#endif  /* SMART_IO_CLK_ACTIVE */

    /* Bypass channel mask for input and output pin */
    /* Configure BYPASS setting */
    smart_io_cfg.bypassMask = SMARTIO_BYPASS_CH_MASK;

    /* IO channel sync mask for selected pin */
    /* Configure Synchronizer setting */
    smart_io_cfg.ioSyncEn = SMARTIO_IOSYNC_CH_MASK;

    /************************** LUT3[4] setting ***************************/
    /* Configure LUT3 [4] */
    lutCfgLut4.opcode = LUTx_LOGIC_OPCODE_GI2;
    lutCfgLut4.lutMap = LUT4_OUT_MAP;

    /* Lut configuration for input */
    lutCfgLut4.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_CHIP4;
    lutCfgLut4.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_CHIP4;
    lutCfgLut4.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_IO6;
    smart_io_cfg.lutCfg[4] = &lutCfgLut4;

    /************************** LUT3[5] setting ***************************/
    /* Configure LUT3 [5] */
    /* Lut configuration for output, check description above */
```

```
    lutCfgLut5.opcode = LUTx_LOGIC_OPCODE_GO;
    lutCfgLut5.lutMap = LUT5_OUT_MAP;

    /* Lut configuration for input (button) */
    lutCfgLut5.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_DU_OUT;
    lutCfgLut5.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT5_OUT;
    lutCfgLut5.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT4_OUT;
    smart_io_cfg.lutCfg[5] = &lutCfgLut5;

    /************************** LUT3[6] setting **************************/
    /* Configure LUT3 [6] */
    lutCfgLut6.opcode = LUTx_LOGIC_OPCODE_COMB;
    lutCfgLut6.lutMap = LUT6_OUT_MAP;

    /* Lut configuration for input */
    lutCfgLut6.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT7_OUT;
    lutCfgLut6.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT7_OUT;
    lutCfgLut6.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT7_OUT;
    smart_io_cfg.lutCfg[6] = &lutCfgLut6;

    /************************** LUT3[7] setting **************************/
    /* Configure LUT3 [7] */
    lutCfgLut7.opcode = LUTx_LOGIC_OPCODE_GI2;
    lutCfgLut7.lutMap = LUT7_OUT_MAP;

    /* Lut configuration for input */
    lutCfgLut7.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT4_OUT;
    lutCfgLut7.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT5_OUT;
    lutCfgLut7.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_IO7;
    smart_io_cfg.lutCfg[7] = &lutCfgLut7;

    /************************** DU setting **************************/
    /* Configure DU */
    lutCfgDu.tr0 = CY_SMARTIO_DUTR_LUT6_OUT;        /**< DU input trigger 0 source selection -
LUT[3]6 output*/
    lutCfgDu.tr1 = CY_SMARTIO_DUTR_LUT7_OUT;        /**< DU input trigger 1 source selection -
LUT[3]7 output*/
    lutCfgDu.tr2 = CY_SMARTIO_DUTR_ZERO;            /**< DU input trigger 2 source selection -
Constant 0*/
    lutCfgDu.data0 = CY_SMARTIO_DUDATA_ZERO;        /**< DU input DATA0 source selection -
Fixed 0*/
    lutCfgDu.data1 = CY_SMARTIO_DUDATA_DATAREG;     /**< DU input DATA1 source selection -
SMARTIO_PRTx_DATA.DATA [7:0]*/
    lutCfgDu.opcode = CY_SMARTIO_DUOPC_INCR_WRAP;   /**< DU op-code */
    lutCfgDu.size = CY_SMARTIO_DUSIZE_8;            /**< DU width size is 8 */
    lutCfgDu.dataReg = 0x10ul;                      /**< DU DATA register value = 16 */
    smart_io_cfg.duCfg = &lutCfgDu;

    /*****************************************************************/
    /* Initialization of Smart IO structure */
    /* Configure Smart I/O. See Code Listing 14. */
    retStatus = Cy_SmartIO_Init(SMART_IO_PORT, &smart_io_cfg);
```

```
    return retStatus;
}
```

Code Listing 13 Cy_SmartIO_Enable() 関数

```
void Cy_SmartIO_Enable(volatile stc_SMARTIO_PRT_t* base)
{
    un_SMARTIO_PRT_CTL_t workCTL = base->unCTL;
    workCTL.stcField.u1ENABLED     = CY_SMARTIO_ENABLE;
    workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_DISABLE;
    base->unCTL.u32Register        = workCTL.u32Register;  /* (12) Enable Smart I/O. */
}
```

4 設定例

Code Listing 14 Cy_SmartIO_Init() 関数

```
cy_en_smartio_status_t Cy_SmartIO_Init(volatile stc_SMARTIO_PRT_t* base, const
cy_stc_smartio_config_t* config)
{
    cy_en_smartio_status_t status = CY_SMARTIO_SUCCESS;

    if(NULL != config)
    {
        /* (4) Set clock source and bypass to Smart IO */
        un_SMARTIO_PRT_CTL_t workCTL = {.u32Register = 0ul};
        workCTL.stcField.u1ENABLED     = CY_SMARTIO_DISABLE;
        workCTL.stcField.u1HLD_OVR     = config->hldOvr;
        workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_ENABLE;
        workCTL.stcField.u5CLOCK_SRC   = config->clkSrc;
        workCTL.stcField.u8BYPASS      = config->bypassMask;
        base->unCTL.u32Register        = workCTL.u32Register;

        /* (5) Set synchronizer to Smart IO */
        un_SMARTIO_PRT_SYNC_CTL_t workSYNC_CTL = {.u32Register = 0ul};
        workSYNC_CTL.stcField.u8IO_SYNC_EN   = config->ioSyncEn;
        workSYNC_CTL.stcField.u8CHIP_SYNC_EN = config->chipSyncEn;
        base->unSYNC_CTL.u32Register         = workSYNC_CTL.u32Register;

        /* LUT configurations - skip if lutCfg is a NULL pointer */
        /* (6), (7), (8), (9) Set LUT3 */
        for(uint32_t i = CY_SMARTIO_LUTMIN; i < CY_SMARTIO_LUTMAX; i++)
        {
            if(NULL != config->lutCfg[i])
            {
                un_SMARTIO_PRT_LUT_SEL_t workLUT_SET = { .u32Register = 0ul };
                workLUT_SET.stcField.u4LUT_TR0_SEL = config->lutCfg[i]->tr0;
                workLUT_SET.stcField.u4LUT_TR1_SEL = config->lutCfg[i]->tr1;
                workLUT_SET.stcField.u4LUT_TR2_SEL = config->lutCfg[i]->tr2;
                base->unLUT_SEL[i].u32Register     = workLUT_SET.u32Register;

                un_SMARTIO_PRT_LUT_CTL_t workLUT_CTL = { .u32Register = 0ul };
                workLUT_CTL.stcField.u2LUT_OPC = config->lutCfg[i]->opcode;
                workLUT_CTL.stcField.u8LUT     = config->lutCfg[i]->lutMap;
                base->unLUT_CTL[i].u32Register = workLUT_CTL.u32Register;
            }
        }

        /* DU Configuration - skip if duCfg is a NULL pointer */
        /* (10) Set DU */
        if(NULL != config->duCfg)
        {
            un_SMARTIO_PRT_DU_SEL_t workDU_SEL = {.u32Register = 0ul};
            workDU_SEL.stcField.u4DU_TR0_SEL    = config->duCfg->tr0;
            workDU_SEL.stcField.u4DU_TR1_SEL    = config->duCfg->tr1;
            workDU_SEL.stcField.u4DU_TR2_SEL    = config->duCfg->tr2;
            workDU_SEL.stcField.u2DU_DATA0_SEL = config->duCfg->data0;
            workDU_SEL.stcField.u2DU_DATA1_SEL = config->duCfg->data1;
```

4 設定例

```
            base->unDU_SEL.u32Register         = workDU_SEL.u32Register;

            un_SMARTIO_PRT_DU_CTL_t workDU_CTL = {.u32Register = 0ul};
            workDU_CTL.stcField.u3DU_SIZE = config->duCfg->size;
            workDU_CTL.stcField.u4DU_OPC  = config->duCfg->opcode;
            base->unDU_CTL.u32Register    = workDU_CTL.u32Register;

            base->unDATA.stcField.u8DATA = config->duCfg->dataReg;
        }
    }
    Else
    {
        status = CY_SMARTIO_BAD_PARAM;
    }

    return(status);
}
```

5 用語集

表 24 用語集

用語	説明
chip_data	HSIOM からの入力信号
Clk_sys/CLK_HF	周辺クロック分周器を用いてシステムクロックから生成されます。詳細については、Architecture Reference Manual の Clocking System 章を参照してください。
DeepSleep	パワーモードは低周波数周辺機能のみ有効です。詳細は Architecture Reference Manual の Device Power Modes 章の DeepSleep Mode セクションを参照してください。
DU	Data Unit (データユニット)。DU はレジスタのオペコード設定に基づいて、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 の動作を実行します。詳細は Architecture Reference Manual の I/O System 章の Smart I/O - Data Unit セクションを参照してください。
GPIO	General-purpose input/output (汎用入力/出力)
HSIOM	High Speed I/O Matrix (高速 I/O マトリクス) 。詳細は Architecture Reference Manual の I/O System 章の High-Speed I/O Matrix セクションを参照してください。
io_data_in	I/O ポートからの入力信号
I/O ポート	I/O ポートは CPU コアと周辺コンポーネント間のインタフェースを外部に提供します。詳細は Architecture reference manual の I/O System 章を参照してください。
LUT3 [x]	3-input Lookup Tables (3 入力のルックアップテーブル)。LUT3 [x]ブロックは 3 つの入力信号があり、レジスタの設定に基づいて出力を生成します。詳細は Architecture Reference Manual の I/O System 章の Smart I/O - LUT3 セクションを参照してください。
smartio_data	スマート I/O からの出力信号

6 参考資料

以下は TRAVEO™ T2G ファミリシリーズのデータシートおよびテクニカル リファレンス マニュアルです。これらのドキュメントを入手するには、テクニカルサポートに連絡してください。

- デバイス データシート
  - CYT2B6 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2BL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-33466)
  - CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4EN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-30842)
  - CYT2CL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- Body Controller Entry ファミリ
  - TRAVEO™ T2G automotive cluster entry family architecture reference manual
  - TRAVEO™ T2G automotive body controller entry registersreference manual for CYT2B7
  - TRAVEO™ T2G automotive cluster entry registers reference manual for CYT2B9
  - TRAVEO™ T2G automotive body controller entry registers reference manual for CYT2BL (Doc No. 002-29852)
- Body Controller High ファミリ
  - TRAVEO™ T2G automotive body controller high family architecture reference manual
  - TRAVEO™ T2G automotive body controller high registers reference manual for CYT3BB/4BB
  - TRAVEO™ T2G automotive body controller high registers reference manual for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers reference manual for CYT6BJ (Doc No. 002-36068)
- Cluster 2D ファミリ
  - TRAVEO™ T2G automotive cluster 2D architecture reference manual
  - TRAVEO™ T2G automotive cluster entry registers reference manual for CYT3DL
  - TRAVEO™ T2G automotive cluster entry registers reference manual for CYT4DN
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual for CYT4EN (Doc No. 002-35181)
- クラスタ Entry ファミリ
  - TRAVEO™ T2G automotive cluster entry family architecture reference manual
  - TRAVEO™ T2G automotive cluster entry registers reference manual for CYT2CL
- アプリケーションノート
  - AN220193 - TRAVEO™ T2G ファミリ GPIO の使用方法
  - AN220208 - TRAVEO™ T2G ボディエントリ ファミリのクロック設定
  - AN224434 - TRAVEO™ T2G Body High ファミリのクロック設定方法
  - AN226071 - TRAVEO™ T2G ファミリ CYT4D シリーズのクロック設定方法
  - AN229513 - Clock configuration setup in TRAVEO™ T2G family CYT2C series

7 その他の参考資料

さまざまな周辺機器にアクセスするためのサンプルソフトウェアとしてのスタートアップを含むサンプルドライバライブラリ (SDL) を提供しています。SDL は、公式の AUTOSAR 製品でカバーされないドライバの顧客へのリファレンスとしても機能します。SDL は自動車規格に適合していないため、製造目的では使用できません。このアプリケーションノートのプログラムコードは、SDL の一部です。SDL の入手については、テクニカルサポートに連絡してください。

改訂履歴

版数	発行日	変更内容
**	2019-06-04	これは英語版 002-20203 Rev. **を翻訳した日本語版 002-26949 Rev. **です。
*A	2019-07-31	これは英語版 002-20203 Rev. *A を翻訳した日本語版 002-26949 Rev. *A です。英語版の変更内容: Updated Associated Part Family as “TRAVEO™ T2G family CYT2B/CYT4B Series”. Added target part numbers “CYT4B Series” in all instances across the document.
*B	2020-03-24	これは英語版 002-20203 Rev. *B を翻訳した日本語版 002-26949 Rev. *B です。英語版の変更内容: Updated Associated Part Family as “TRAVEO™ T2G family CYT2B/CYT4B/CYT4D Series”. Added target part numbers “CYT4D Series” in all instances across the document.
*C	2020-07-10	これは英語版 002-20203 Rev. *C を翻訳した日本語版 002-26949 Rev. *C です。英語版の変更内容: Updated Associated Part Family as “TRAVEO™ T2G family CYT2/CYT3/CYT4 Series”. Changed target part numbers from “CYT2B/CYT4B/CYT4D Series” to “CYT2/CYT4 Series” in all instances across the document. Added target part numbers “CYT3 Series” in all instances across the document.
*D	2021-09-10	これは英語版 002-20203 Rev. *D を翻訳した日本語版 002-26949 Rev. *D です。英語版の変更内容: Updated 設定例: Added example of SDL Code and description in all instances. Updated to Infineon template.
*E	2024-07-17	これは英語版 002-20203 Rev. *E を翻訳した日本語版 002-26949 Rev. *E です。英語版の変更内容: Template update; no content updated.
*F	2025-10-23	これは英語版 002-20203 Rev. *F を翻訳した日本語版 002-26949 Rev. *F です。英語版の変更内容: Added CYT6 Series. Updated 参考資料.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-10-23

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-byr1680596068900

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、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。

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。