デュアル入力 7 ~ 13-Bit インクリメンタル ADC データシート DualADC V 2.2

Copyright © 2001-2011 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック			API メモリ（バイト）		ピン（外部 I/O あたり）
	デジタル	アナログ CT	アナログ SC	Flash	RAM	
CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52						
	4	0	2	444	8	1
CY8C26/25xxx	4	0	2	474	8	1

その他のコンバータについては、アプリケーション ノート「アナログ - ADC の選択」AN2239 を参照してください

このユーザ モジュールを使用する機能例として完全に設定されたプロジェクトについては、を参照してください。www.cypress.com/psocexampleprojects

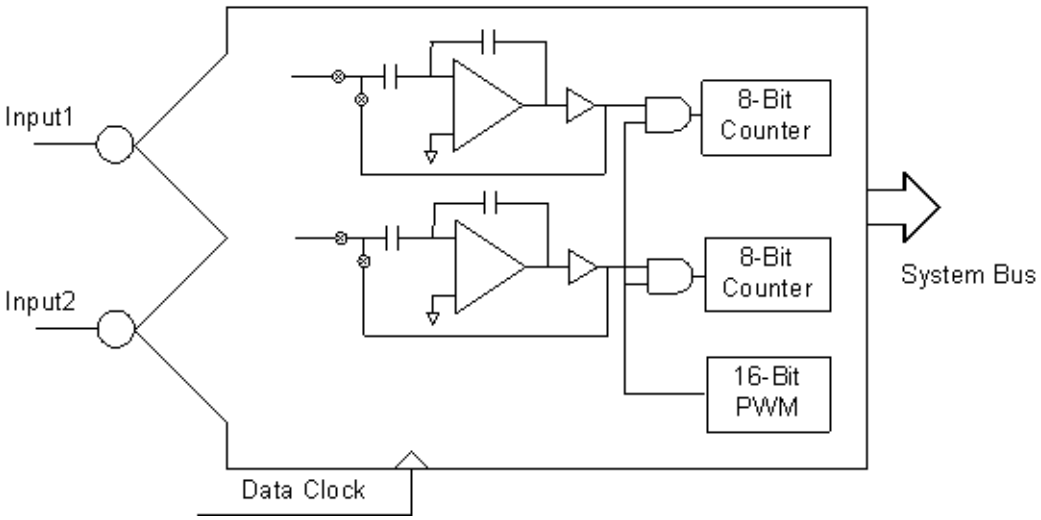
特徴と概要

- 2 つの同時入力サンプル
- 7 ~ 13-bit 分解能
- 2 の補数または符号なし整数
- 4 から 10,000 sps を超えるサンプルレート
- Vss ~ Vdd を含む複数の入力レンジ
- 積分型変換器による優れたノーマルモード ノイズの低減
- 内部または外部クロック

DualADC ユーザ モジュールは、7 ~ 13 ビットの調整可能分解能を持つデュアル入力のインクリメンタル ADC です。積分時間を最適化し、不要な高周波成分を取り除くように設定できます。レールツーレールを含む入力電圧範囲は、適切なリファレンス電圧とアナログ グラウンドを構成することで測定できます。出力は 2 の補数または符号なしの整数として構成される場合があります。DualADC は、電力測定など、2 つの信号を同時にサンプリングすることが必要な用途に最適です。その他の PSoC ADC と同様に、両方の入力信号はマルチプレクサを用いて多重化されていても問題ありません。CPU の負荷は、入力レベルによって異なります。たとえば、$V_{in} = +V_{ref}$ の場合、CPU サイクル数は 10,014 です（最大 13 ビット）。V_{in} = AGND の場合、CPU サイクル数は 5,278 です（平均 13 ビット）。また、$V_{in} = -V_{ref}$ の場合、CPU サイクル数は 542 です（最小 7-13 ビット）。

Figure 1. DualADC ブロック ダイアグラム

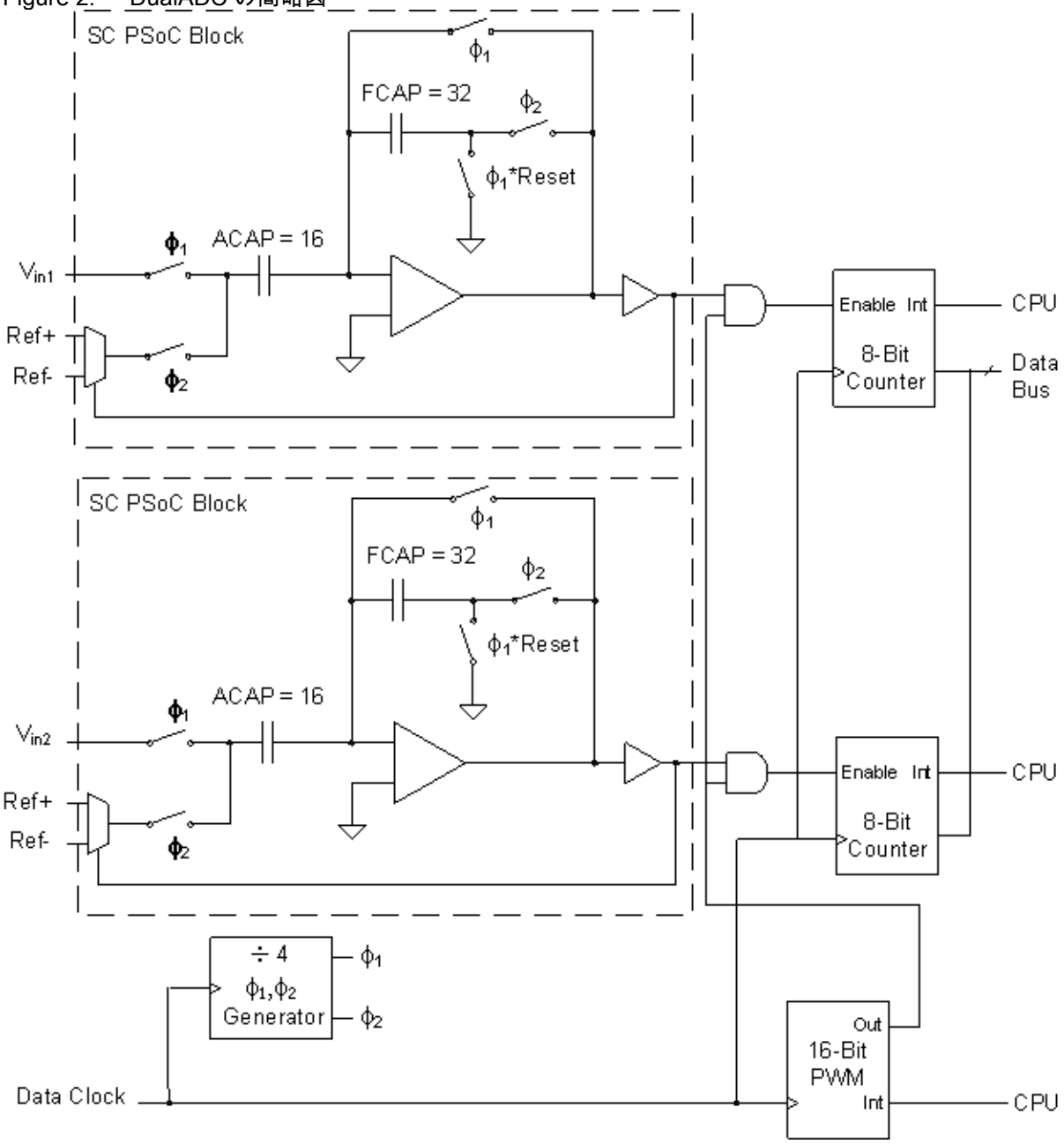
Review Placement Section Prior to Module Placement



機能説明

DualADC ユーザ モジュールは、 1 つのユーザ モジュールに 2 つのインクリメンタル ADC で構成されています。16-bit サンプリングレートタイマ（PWM）は必要なデジタルブロック数を低減するために共有されます。両方の ADC が同じタイマを使用するため、サンプリングは完全に同期されています。変換サイクルの最後に、両方の入力の結果が同時に表示されます。4 つのデジタル PSoC ブロックと、2 つのアナログスイッチド キャパシタ PSoC ブロックが必要です。下記の簡略図を参照してください。

Figure 2. DualADC の簡略図



2 つのアナログ・ブロックは、リセット可能な積分器と同一に構成されています。出力の極性によって、リファレンス電圧が入力に加算または入力から減算され、積分器に置かれるようにリファレンス制御が設定されます。リファレンス制御は、積分器の出力を AGND の電位に戻す動作もします。積分器が $2^{ビット}$回作動し、出力電圧コンパレータがこれらの回数の正の 「n」の場合、出力の残留電圧 (V_{resid}) は以下のようになります。

Equation 1

$$V_{resid} = 2^{Bits} \cdot V_{in} - (n \cdot V_{ref}) + (2^{Bits} - n) \cdot V_{ref}$$

Equation 2

$$V_{in} = \frac{n - 2^{Bits-1}}{2^{Bits-1}} V_{ref} + \frac{V_{resid}}{2^{Bits}}$$

この式は、この ADC の範囲が $\pm V_{ref}$、分解能 （ LSB ）が $V_{ref}/2^{ビット-1}$ で、計算の最後の出力の電圧が余りと定義されることを示しています。V_{resid} は常に V_{ref} 未満なので、$V_{resid}/2^{ビット}$ は LSB の半分未満となり、無視できます。その結果、式は次のようになります。

Equation 3

$$V_{in} = \frac{n - 2^{Bits-1}}{2^{Bits-1}} V_{ref}$$

例 1

V_{ref} が 1.3V で分解能が 8-bits の場合は、データの使用準備が整った時点でインクリメンタル ADC から読み取られた値を基に、入力電圧を簡単に計算できます。使用できる式は次のようになります。

Equation 4

$$V_{in} = \frac{n - 128}{128} 1.3$$

計算結果は AGND を基準とします。ADC データの値が 200 の場合、測定電圧は以下のように 0.73V と計算できます。

Equation 5

$$V_{in} = \frac{200 - 128}{128} 1.3 = 0.73V$$

計算された値は、実際的な値であり、これは、システムのノイズやチップのオフセット値に応じて大幅に異なることがあります。

[+] Feedback

特定の入力電圧を前提として予想されるコードを決定するために式を再整理すると、以下の式のようになります。

Equation 6

$$n = \frac{2^{Bits-1} \cdot V_{in}}{V_{ref}} + 2^{Bits-1}$$

例 2

V_{ref} が 1.3V で分解能が 8-bits の場合は、入力電圧を基に、予想される ADC　結果を簡単に計算できます。使用できる式は次のようになります。

Equation 7

$$n = \frac{128 \cdot V_{in}}{1.3} + 128$$

入力電圧が AGND より -1V 低い場合、次の式に基づいて、ADC からのコードは 29.53 になると予想できます。

Equation 8

$$n = \frac{128 \cdot (-1)}{1.3} + 128 = 29.53$$

計算された値は、実際的な値であり、これは、システムのノイズやチップのオフセット値に応じて大幅に異なることがあります。

積分器をインクリメンタル ADC として機能させるには、以下のデジタル リソースを利用します。

- 出力が正のサイクル数を累積する 8-bit カウンタ（1 チャネルにつき 1 つ）。
- 積分時間を計り、8-bit カウンタへのクロックをゲート制御する 16-bit PWM（両チャネルで共有）。

1 つの DataClock が、8-bit カウンタ、16-bit PWM、およびアナログ SC PSoC ブロックに接続するアナログ コラムのクロックに接続されます。アナログ コラムのクロックは、実際には DataClock から生成される φ_1 と φ_2 の 2 つのクロックです。これらの 2 つの追加クロックは、DataClock の周波数の 1/4 です。つまり、PWM とカウンタは必要な速度の 4 倍で動作するため、N+2 ビット相当のデータを累積する必要があります（N は分解能のビット数と等しい）。

Note　このモジュールを配置する場合は、すべてのブロックに対して同じクロックを設定する必要があります。これを怠ると、不適切な動作の原因となります。

カウンタは、LSB 用の 8-bit のデジタル ブロックおよび MSB 用のソフトウェア カウンタを使用して実装します。ハードウェア カウンタがオーバフローするたびに割り込みが生成され、カウンタの上位 MSB が増分されます。これにより、DualADC モジュールは 6 つではなく、4 つのデジタルブロックで実装するだけで済みます。

サンプリング速度は、DataClock を積分時間で割り、結果の計算にかかる時間（CalcTime）を足した値です。積分時間は、入力信号を DualADC がサンプリングしている時間です。

Equation 9

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

結果の計算にかかる時間、CalcTime は、CPU クロックに反比例して変化します。CalcTime には、結果の計算に必要な時間よりも大きな値を設定する必要があります。最小 CalcTime は 260 CPU サイクルと等しく、DataClock に換算して表現する必要があります。また、CalcTime は、サンプリング速度を最適化するために、最小値より増加させることもできます。

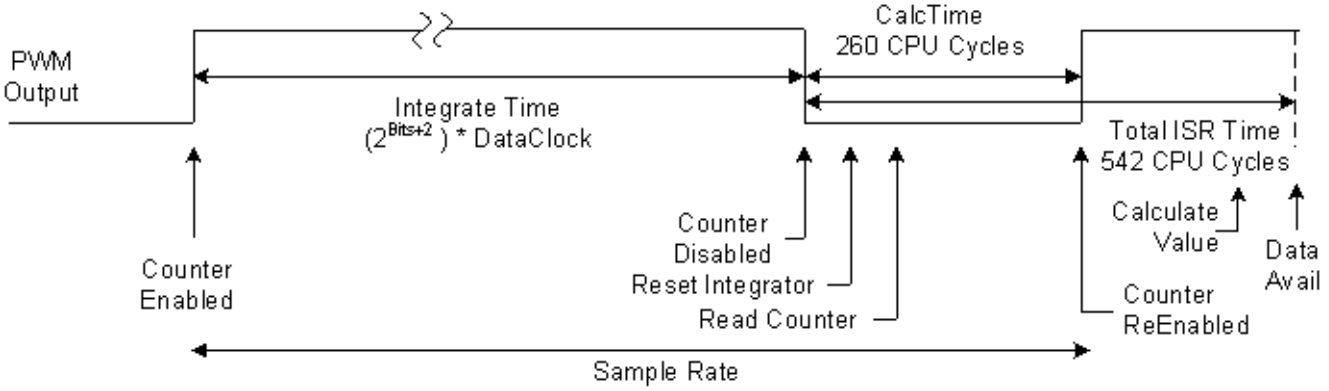
Note $2^{ビット+2}$ と CalcTime の合計は、2^{16}-1 つまり 65,535 より大きくなってはなりません。

Equation 10

$$CalcTime \geq \frac{DataClock * 260}{CPUClock}$$

16-bit PWM は、$2^{ビット+2}$ と DataClock の積である HIGH 信号を出力するようにプログラムします。たとえば、分解能を 10 ビットに設定すると、PWM 出力は 4096（2^{10+2}）DataClock の間、HIGH のままとなります。PWM の出力は、最短で、変換結果を計算して積分器をリセットするためにかかる時間の間、LOW になります。この期間は CalcTime パラメータで制御されます。CalcTime も、DataClock と組み合わせてより正確なサンプリング速度を提供する目的で調整できます。PWM の合計時間は、積分時間と CalcTime の和です。

Figure 3. PWM 出力との関連でみた DualADC タイミング



最初の読み取りが開始されると、PWM の構成が計算され、積分器がリセットされて、カウンタが FFh にリセットされます。最初の遅延は、常に計算時間の遅延以上となります。PWM は、最初の読み取りを行う前にのみ初期化されます。比較および時間レジスタをいったん設定した後は、分解能または計算時間を変更しない限り、再初期化は必要ありません。PWM カウントが積分値以下の場合、出力は HIGH になるため 8-bit カウンタがカウントダウンを実行できます。PWM の出力は、カウンタがゼロになるまで HIGH のままです。この時点で、8-bit カウンタへのクロックが無効化され、PWM 割り込みが生成されます。

8-bit ソフトウェア カウンタの初期値は、最も小さい負の数の $2^{ビット}$ /64 倍に設定されます。8-bit カウンタがオーバフローするたびに、8-bit カウンタが割り込まれ、ソフトウェア カウンタが 1 増分されます。

[+] Feedback

ADC への入力が入力範囲内の最も大きい値の場合は、DataClock が正になるたびに 8-bit カウンタが増分されます。ADC への入力が入力範囲内の最も小さい負の値以下の場合は、8-bit カウンタが減分されることはなく、したがって割り込みが生成されることもありません。理想的な条件下でアナログ グラウンドに近い入力を行うことで、カウンタは半分の時間で増分できます。入力電圧レベルによって、8-bit カウンタからの割り込みの量が 0 ~ ($2^{ビット+2}$)/256 の範囲で変化します。たとえば、分解能を 10 ビットに設定すると、PWM 比較値には 2^{10+2}（4096）が設定されます。つまり、積分時間中、最大 4096/256、つまり 16 回プロセッサへの割り込みが発生する可能性があります。

DualADC 制御は割り込みベースで、サンプル時間は結果が出るまで比較的長くかかるので、サンプルの処理中にプロセッサを待機させるのは合理的ではありません。ADC ルーチンとメインプログラム間の主な通信は、ポーリング可能なフラグです。DualADC_bfStatus の MSB がゼロ以外の値であれば、DualADC_iResultn (n=1,2) で新データが利用可能です。API は、データフラグを確認し、データを取得するために使用できます。

このデータ ハンドラは、ポーリング ベースとして設計されています。割り込みベースのデータ ハンドラが必要な場合は、ユーザは、割り込みルーチン DualADC_CNTn_ISR（アセンブリファイル *DualADCINT.asm* 内）に独自のデータ ハンドラ コードを挿入できます。コードを挿入する点は、はっきりマークされています。

CPU 使用率

DualADC では、結果を計算し、ハードウェア カウンタがオーバフローするたびにソフトウェア カウンタを増分するため CPU 時間が必要です。CPU のオーバヘッドは、CPU クロック、DataClock、および入力電圧の 3 つの変数に左右されます。入力電圧は ADC の CPU のオーバヘッドに影響することになります。–Vref に近いかそれ未満の入力電圧では、CPU のオーバヘッドはほとんど必要ありません。+Vref に近いかそれより大きい入力電圧では、より多くの CPU のオーバヘッドが必要となります。次の式は、両入力で入力信号が同じであると想定しています。特定の入力に必要な CPU サイクルを計算する手順は、次の式を参照してください。

Equation 11

$$CPUcycles = PWM16_IRQ_CPUcycles + \left(\frac{2^{Bits}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * (Counter_IRQ_CPUcycles * 2)\right)$$

Equation 12

$$CPUcycles = 542 + \left(\frac{2^{Bits}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * (37 * 2)\right)$$

10 ビットの分解能における最大 CPU サイクルを計算するには、Vin ~ Vref を設定し、次の式を参照します。

Equation 13

$$CPUcycles = 542 + \left(\frac{2^{10}}{64} * \left(\frac{Vref + Vref}{2 * Vref}\right) * 74\right) = 542 + (16 * 1 * 74) = 1726$$

DualADC の CPU 使用率を計算するには、次の式を参照します。

Equation 14

$$Percent_CPU_Utilization = \frac{Sample_Rate * CPUcycles}{CPU_freqency} * 100$$

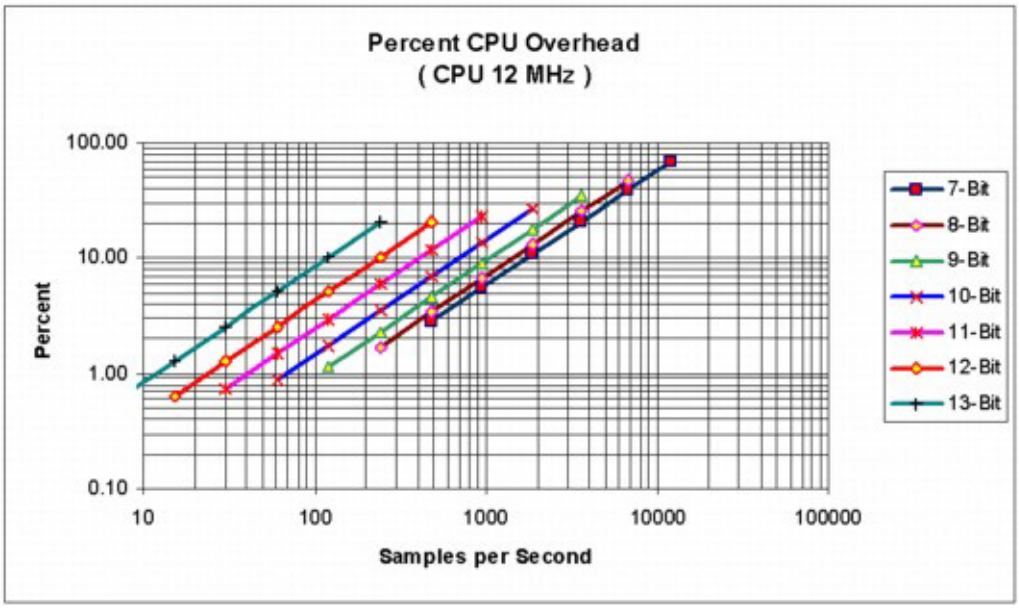
分解能を 10 ビット（上記の例参照）に、サンプリング速度を 1000 サンプル / 秒に、CPU クロックを 12 MHz に設定すると、（下の式参照）使用率は 14.4% と計算されます。

Equation 15

$$Percent_CPU_Utilization = \frac{1000 * 1726}{12MHz} * 100 = 14.4\%$$

下のグラフは、サポートされているサンプリング速度と分解能に対応する CPU 使用率を示しています。デフォルトの CPU 速度は 12 MHz に設定されています。

Figure 4. CPU 使用率とサンプリング速度



チャネル間の差

DualADC を使用した場合、同じ入力電圧を測定してもチャネル間で差がでます。この差は、スイッチドキャパシタのブロックアンプと、コラム AGND バッファの間の入力オフセット変動によるものです。このチャネル間オフセットは、各 ADC チャネルに同じ信号が送信されるように配線することにより、簡単に補正できます。チャネルのうちの 1 つをリファレンスとして使用し、その他のチャネル間の差は毎回の測定のあと引くことで達成できます。

周波数遮断

適切な積分時間を選択することによって、特定のノイズ源はある程度遮断できます。ノイズ源とその高調波を遮断するには、ノイズ信号の積分サイクルと等しい積分時間を選択します。複数の信号を遮断する場合は、両方の信号の積分サイクルと等しい積分時間を選択します。

[+] Feedback

たとえば、50 Hz および 60 Hz の信号によって引き起こされるノイズを遮断するには、50 Hz および 60 Hz の両方の信号の整数を含む時間を選択します。

Equation 16

$$IntegrateTime = 6 * \frac{1}{60} = 5 * \frac{1}{50} = 100mSec$$

IntegrateTime を 100 ms にすると、50 Hz と 60 Hz の両方およびこれらの信号の高調波が遮断されます。次に、適切な IntegrateTime（積分時間）を生成するために必要な DataClock を計算します。

Equation 17

$$DataClock = \frac{2^{Bits+2}}{IntegrateTime}$$

サンプリング速度に影響するにもかかわらず、この計算では CalcTime は使用されていないことに注意してください。IntegrateTime は、DualADC が入力電圧を実際にサンプリングする時間です。サンプリング速度は IntegrateTime および結果の計算にかかる時間に基づきます。

例

あるアプリケーションでは、100 ms の IntegrateTime と 13 ビットの A/D 分解能が必要です。IntegrateTime を 100 ms にするには、以下のようなデータ クロックが必要です。

Equation 18

$$DataClock = \frac{2^{Bits+2}}{IntegrateTime} = \frac{2^{13+2}}{100ms} = 327.7kHz$$

データ クロックに換算した CalcTime は、DataClock と CPU クロックから計算します。CPU クロックが 12 MHz の場合、最小計算時間は以下のようになります。

Equation 19

$$CalcTime \geq \frac{260 \times DataClock}{CPU_Clock}$$

この CalcTime は、最も近い整数（この例では「8」）に切り上げます。サンプリング速度を特定するには、次の手順に従います。

Equation 20

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

より長いサンプリング速度が必要な場合は、CalcTime + 2^{13+2} が 2^{16}-1（65535）以下になるまで CalcTime を増加できます。

DC および AC の電気的特徴

以下の表で別途指定されている場合を除き、T_A = 25°C、Vdd= 5.0V、パワー設定 HIGH、オペアンプ バイアス LOW で、出力は P 2[6] で 1.25 の外部 Vref を持つ P2[4] での 2.5V の外部アナログ グラウンドを基準とし、分解能は 13 ビットに設定されています。

Table 1. 5.0V DualADC の DC および AC 電気的特徴、PSoC デバイスの Arch27Name; ファミリ

パラメータ	標準	制限	単位	条件および注意
Input（入力）				
入力電圧範囲	---	Vss ~ Vdd		Ref Mux = Vdd/2 ± Vdd/2
入力静電容量 [1]	3	---	pF	
入力インピーダンス	1/(C*clk)	---	Ω	
分解能	---	7 ~ 13	ビット	
サンプリング速度	---	4 ~ 10,000	SPS	
SNR	77	---	dB	
DC 精度				
DNL	2	---	LSB	アナログコラム　クロック 2 MHz
INL	1.0	---	LSB	
オフセット誤差	9	---	mV	
ゲイン誤差				
リファレンス ゲイン誤差を含む	3.0	--	% FSR	
リファレンス ゲイン誤差を除く [2]	0.1	--	% FSR	
動作電流				
Low Power	370	---	µA	
Med Power	1200	---	µA	
High Power	4000	---	µA	
データ クロック	---	0.125 ~ 8.0	MHz	デジタル ブロックへの入力およびアナログ コラムのクロック

以下の値は初期の特性データを基に、予測される性能を入れておきます。以下の表で別途指定されている場合を除き、T_A = 25°C、Vdd= 3.3V、パワー設定 HIGH、オペアンプ バイアス LOW ですべての制限が保証され、出力は P2[6] で 1.25 の外部 Vref を持つ P2[4] での 1.64V の外部アナログ グラウンドを基準とし、分解能は 13 ビットに設定されています。

Table 2. 3.3V DualADC の DC および AC 電気的特徴、PSoC デバイスの Arch27Name; ファミリ

パラメータ	標準	制限	単位	条件および注意
Input（入力）				
入力電圧範囲	---	Vss ~ Vdd		Ref Mux = Vdd/2 ± Vdd/2
入力静電容量 [1]	3	---	pF	
入力インピーダンス	1/(C*clk)	---	Ω	
分解能	---	7 ~ 13	ビット	
サンプリング速度	---	4 ~ 10,000	SPS	
SNR	77	---	dB	
DC 精度				
DNL	2	---	LSB	アナログ　コラムのクロック 2 MHz
INL	1.0	---	LSB	
オフセット誤差	4	---	mV	
ゲイン誤差				
リファレンス ゲイン誤差を含む	3.0	--	% FSR	
リファレンス ゲイン誤差を除く [2]	0.4	--	% FSR	
動作電流				
低出力	300	---	µA	
中出力	1000	---	µA	
高出力	3800	---	µA	
データ クロック	---	0.125 ~ 8.0	MHz	デジタル ブロックへの入力およびアナログ コラムのクロック

電気的特性に関する注意

1. I/O ピンを含む。
2. リファレンス ゲイン誤差は、テスト マルチ プレクサによって経路指定され、ピンに戻される $V_{RefHigh}$ および V_{RefLow} と外部リファレンスを比較して測定します。

[+] Feedback

以下の表で別途指定されている場合を除き、T_A = -40°C to ~ +85°C、Vdd= 5.0V ± 10%、パワー設定 HIGH、オペアンプ バイアス LOW ですべての制限が保証され、出力は P 2[6] で 1.25 の外部 Vref を持つ P 2[4] での 2.5V の外部アナログ グラウンドを基準とし、分解能は 12 ビットに設定されています。

Table 3. 5.0V DualADC の DC および AC 電気的特徴、PSoC デバイスの Arch26Name; ファミリ

パラメータ	典型 [1]	制限	単位	条件および注意
Input（入力）				
入力電圧範囲 [2]	---	Vss ~ Vdd		Ref Mux = Vdd/2 ± Vdd/2
入力静電容量 [3]	0.8	---	pF	
入力インピーダンス [4,5]	1/(C*clk)	---	Ω	
分解能	---	7 ~ 13	ビット	2 の補数
サンプリング速度	---	4 ~ 10,000[6]	SPS	1 秒あたりのサンプル数（sps）
SNR[7]	68		dB	100 sps
DC 精度				
INL	0.5	1	LSB	
DNL	0.25	0.5	LSB	
オフセット誤差	12	49	mV	外部 AGND を使用
ゲイン誤差	0.5	1.5	% FSR	基準入力との比較
動作電流				
低出力	185	---	µA	
中出力	450	---	µA	
高出力	1280	---	µA	
データ クロック	---	0.032 ~ 8.0[6]	MHz	デジタル ブロックへの入力およびアナログ コラムのクロック

[+] Feedback

以下の表で別途指定されている場合を除き、T^A = -40°C ~ +85°C、Vdd= 3.0 ~ 3.6V、パワー設定 HIGH、オペアンプ バイアス LOW ですべての制限が保証され、出力は P 2[6] で 1.25 の外部 Vref を持つ P 2[4] での 1.64V の外部アナログ グラウンドを基準とし、分解能は 12 ビットに設定されています。

Table 4. 3.3V DualADC の DC および AC 電気的特徴、PSoC デバイスの Arch26Name; ファミリ

パラメータ	典型 [1]	制限	単位	条件および注意
Input（入力）				
入力電圧範囲 [2]	---	Vss +/- Vdd		Ref Mux = Vdd/2 ± Vdd/2
入力静電容量 [3]	0.8	---	pF	
入力インピーダンス [4,5]	1/(C*clk)	---	Ω	
分解能	---	7 ~ 13	ビット	2 の補数
サンプリング速度	---	4 ~ 10,000[6]	KSPS	1 秒あたりのサンプル数（sps）
SNR[7]	65		dB	100 sps
DC 精度				
INL	0.5	1	LSB	
DNL	0.25	0.5	LSB	
オフセット誤差	12	49	mV	
ゲイン誤差	0.5	2.5	% FSR	リファレンス入力との比較
動作電流				
低出力	150	---	µA	
中出力	350	---	µA	
高出力	1120	---	µA	
データ クロック	---	0.125 ~ 8[6]	MHz	デジタル ブロックへの入力およびアナログ コラムのクロック

電気的特性に関する注意

1. 典型値はパラメータ基準の +25°C で測定された値です。
2. 最大値を超えた入力電圧は最大の正の測定値を示します。最小値未満の入力電圧は最小の負の測定値を示します。
3. ユーザ モジュールのみで、入出力ピンは含みません。
4. 入力静電容量または入力インピーダンスは、アナログ ブロックへの入力が直接ピンに対するものである場合のみに該当します。
5. C = 入力静電容量、clk = データ クロック（アナログ コラムのクロック）。
6. 仕様は、別途注記のない限り、サンプリング速度 100 sps、最大データ クロック 8 MHz の場合です。サンプリング速度はデータ クロックと分解能の両方に依存します。
7. SNR = 単一のトーンのフルスケールの出力を Fsample/2 に積分した全ノイズで除算した割合です。

配置

ADC (スイッチド キャパシタ) ブロックは、どのスイッチド キャパシタ PSoC ブロックにも配置できます。各 ADC ブロックは、配置される特定のコラムのコンパレータ バスを独占して使用できる必要があります。すなわち、2 つのブロックのそれぞれが異なるコラムに配置され、コンパレータ バスに接続されている別のスイッチド キャパシタ ブロックを持つコラムは共有できません。

カウンタ ブロックは、使用可能なデジタルブロックであればどれにでも配置できますが、PWM16 は特定の場所にしか配置できません。PWB16 (LSB/MSB) 用 CY8C27xxx デバイスファミリを配置できる位置は、DBB00/DBB01、DBB01/DCB02、DBB10/DBB11、DBB11/DCB12 です。CY8C29/24/22xxx デバイスファミリでは、PWM16 は 2 つの連続するデジタルブロックであればどこでも配置できます。PWB16 (LSB/MSB) 用 CY8C26/25xxx ファミリを配置できる位置は、DBA01/DBA02 と DCA05/DCA06 です。

2 つのカウンタブロックと PWM ブロックにはそれぞれ、処理する割込みサービス ルーチンがあります。カウンタ ブロックが PWM16 ブロックよりも高い割り込み優先順位を持つことが望まれます。したがって、PWM16 ブロックが配置されるブロック番号よりも低いデジタル ブロック番号の位置にカウンタ ブロックを配置することを推奨します。

Note DualADC を初めて選択すると、「Resource allocation prevents placement (この配置は、リソースの配当により妨害されています)」という警告のメッセージが表示される場合があります。この警告は、オリジナルの配置で同じコラムに 2 つの ADC がある場合に表示されます。各 ADC ブロックを該当するコラムに移動します。

パラメータおよびリソース

ADC 入力 1, ADC 入力 2

ADC 入力は、アナログ PSoC ブロックの配置後に選択します。8 個のスイッチド キャパシタ ブロックはそれぞれ異なる入力選択ができます。各ブロックは隣接するブロックのほとんどに接続でき、一部は外部入力ピンに直接接続できます。アナログ ブロックは、そのブロックに入力信号を伝える信号経路を考慮しながら配置する必要があります。一部の配置では、パッケージ ピンから直接入力することができます。このような直接の接続では、電源レールの 40 mV 以内の入力を正確に測定できます。信号は、コラムのマルチプレクサ、または CT ブロックのテスト マルチプレクサを経由して、DualADC が電源レール付近で信号を測定することができるアナログ コラムに乗せることも可能です。2 つの ADC 入力のそれぞれに対して、1 つを選択します。

ClockPhase1, ClockPhase2

クロック位相の選択は、あるスイッチド キャパシタのアナログ PSoC ブロックの出力を別のスイッチド キャパシタのアナログ PSoC ブロックの入力と同期させるために使用します。スイッチド キャパシタのアナログ PSoC ブロックは、2 相クロック (φ_1、φ_2) を使用して信号を取得および転送します。一般的に、DualADC への入力は、通常の設定、φ_1 でサンプリングされます。ユーザ モジュールの多くは、φ_1 中に出力を自動的にゼロに設定し、φ_2 中しか有効な出力を供給しないため、問題が発生します。このようなモジュールの出力が DualADC の入力に入力されると、有効な信号の代わりに、自動的にゼロに設定された出力が取得されます。クロック位相を選択すると、切り替え設定、φ_2 中に入力信号を取得するように、位相を交換できます。2 つのスイッチド キャパシタ ブロックのそれぞれに対して、1 つを選択します。

Clock and Integrator Column Clock (クロックと積分器コラムのクロック)

データ クロックにより、サンプリング速度と信号サンプル時間枠が決まります。このクロックは、カウンタ ブロックのクロック入力、16-bit の PWM ブロック、および積分器を含むコラムのコラム クロックに送る必要があります。

[+] Feedback

Note 積分器スイッチド キャパシタ ブロックのコラム クロックには、同じクロックを手動で設定する必要があります。6 つのブロックすべてで同じクロックを使用する必要があります。同じクロックを使用しないと、DualADC ユーザ モジュールは正しく機能しません。

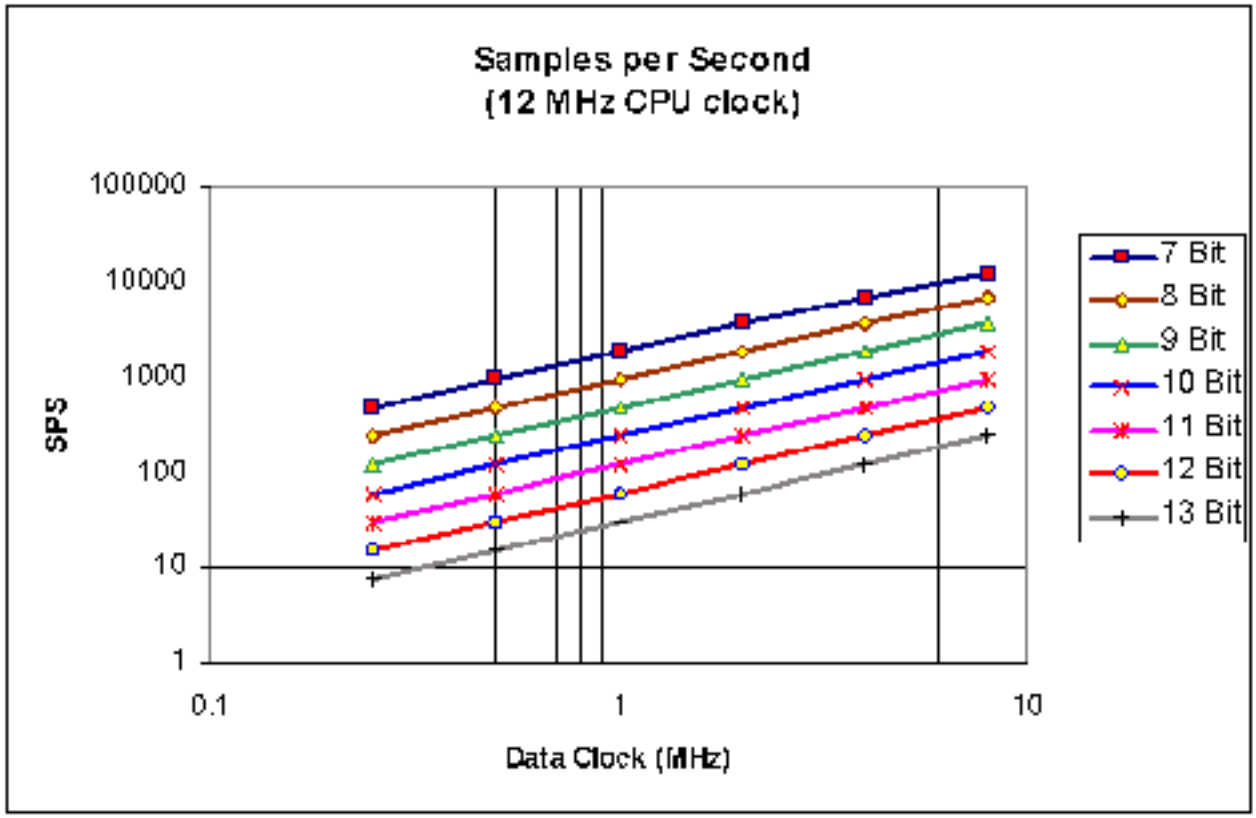
このパラメータ設定では、カウンタ ブロックおよび PWM ブロックだけにクロックが設定されます。このクロックは、125 kHz ~ 8 MHz の間のクロック レートを持つ任意のクロック ソースです。

Equation 21

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

グラフは、DualADC の分解能オプションごとに使用可能なサンプリング速度を示したものです。

Figure 5. 1 秒当たりのサンプルとデータクロック



ADCResolution（ADC 分解能）

ADCResolution を選択すると、デバイス エディタで DualADC の分解能を設定できます。分解能を設定または変更するための API ルーチンもありますが、デバイス エディタで設定を行えば、必要ありません。分解能は、API 呼び出しによってもいつでも変更できますが、DualADC の動作が停止するため、再起動する必要があります。有効な分解能設定は、7 以上 13 以下です。

CalcTime（計算時間）

CalcTime は、次の積分サイクルの前に中間の積分結果を CPU が計算するためにかかる時間です。結果の計算にかかる時間、「CalcTime」は、CPU クロックに反比例して変化します。この値は、データクロックに換算する必要があります。最小 CPU 計算時間は、260 CPU クロックです。また、CalcTime は、サンプリング速度を最適化するために増加させることもできます。

[+] Feedback

Note CalcTime + $2^{ビット+2}$ が 2^{16}-1 すなわち 65,535 を超えないよう注意する必要があります。

下記に CalcTime の設定値を決定する式を示します。

$$CalcTime \geq \frac{DataClock * 260}{CPUClock}$$

下の表は、CalcTime パラメータに選択できるレンジを示します。上記の式を使用して、特定の用途で使用できるレンジの下限を設定します。

Table 5. CalcTime レンジ

分解能	積分時間（DataClock カウント）	CalcTime レンジ（DataClock カウント）
7	512	1 ~ 65,023
8	1,024	1 ~ 64,511
9	2,048	1 ~ 63,487
10	4,096	1 ~ 61,439
11	8,192	1 ~ 57,343
12	16,384	1 ~ 49,151
13	3	1 ~ 32,767

たとえば、DataClock に 1.5 MHz が設定されており、CPU が 12 MHz で動作している場合は、CalcTime を 33 以上に設定する必要があります（下の式を参照）。

Equation 22

$$CalcTime \geq \frac{DataClock * 260}{CPUClock}$$

DataFormat（データフォーマット）

このパラメータを選択すると、結果が返されるフォーマットが決まります。「Signed（符号付き）」を選択し、選択された分解能が「N」であるとすると、結果の範囲は 2^{N-1} ~ 2^{N-1} -1 となります。「Unsigned（符号なし）」を選択すると、結果の範囲は 0 ~ 2^{N}-1 となります。データ フォーマットと分解能ごとの結果範囲については、以下の表を参照してください。

Table 6. データ フォーマットの結果レンジ

分解能設定	符号付きデータ フォーマット	符号なしデータ フォーマット
7	-64 ~ 63	0 ~ 127
8	-128 ~ 127	0 ~ 255
9	-256 ~ 255	0 ~ 511

分解能設定	符号付きデータ フォーマット	符号なしデータ フォーマット
10	-512 ~ 511	0 ~ 1023
11	-1024 ~ 1023	0 ~ 2047
12	-2048 ~ 2047	0 ~ 4095
13	-4096 ~ 4095	0 ~ 8191

RefMux グローバル リソース

アナログからデジタルの変換（ADC）でもっとも重要なグローバルリソースは RefMux です。RefMux の設定により、使用できる ADC の入力電圧範囲が決定します。次の表は、Vdd が 5V と 3.3V のときの範囲を示しています。

Table 7. RefMux 設定に対する CY8C26/25xxx の入力電圧範囲

RefMux の設定	Vdd = 5 Volts	Vdd = 3.3 Volts
(Vdd/2) ± BandGap	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
(Vdd/2) ± (Vdd/2)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
(2*BandGap) ± BandGap	$1.3 < V_{in} < 3.9$	適用外
(2*BandGap) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	適用外
P2[4] ± BandGap	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

Table 8. リファレンス マルチプレクサ設定ごとの CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 の入力電圧範囲

RefMux の設定	Vdd = 5 Volts	Vdd = 3.3 Volts
(Vdd/2) ± BandGap	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
(Vdd/2) ± (Vdd/2)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
BandGap ± BandGap	$0 < V_{in} < 2.6$	$0 < V_{in} < 2.6$
(1.6*BandGap) ± (1.6*BandGap)	$0 < V_{in} < 4.16$	適用外
(2*BandGap) ± BandGap	$1.3 < V_{in} < 3.9$	適用外
(2*BandGap) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	適用外
P2[4] ± BandGap	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

割り込み生成の制御

以下のパラメータにアクセスできるのは、PSoC Designer の **[Enable interrupt generation control (割り込み生成の制御を有効にする)]** チェックボックスが選択されている場合のみです。これは、**[プロジェクト] > [設定] > > [チップ エディタ]** で開くことができます。複数のオーバーレイが使用され、そのオーバーレイ全体の複数のユーザ モジュールにより割り込みが共用されている場合は、割り込み生成制御が重要です。

IntDispatchMode

IntDispatchMode パラメータは、同じブロック、異なるオーバーレイに存在する複数のユーザ モジュールによって共有される割り込みで、割り込みリクエストをどのように取り扱うかを指定します。「ActiveStatus」を選択すると、共有割り込みリクエストに応答する前に、ファームウェアがどちらのオーバーレイがアクティブであるかをテストします。このテストは、共有割り込みがリクエストされるたびに実行されます。これはレイテンシーを生むほか、共有割り込みリクエストに対応する非決定性のプロシージャを生成しますが、RAM は必要としません。「OffsetPreCalc」を選択すると、ファームウェアが、オーバーレイが最初にロードされるときだけ、共有割り込みリクエストのソースを計算するようになります。この計算は、割り込みレイテンシーを低減し、共有割り込みリクエストに応答する決定性のプロシージャを生成しますが、RAM のバイト消費が発生します。

アプリケーション プログラミング インタフェース (API)

アプリケーション プログラミング インタフェース (API) ルーチンは、設計者が高級言語でモジュールを操作できるようにユーザ モジュールをコンポーネントとして提供します。このセクションでは、各機能に対するインタフェースを 「include」ファイルによって提供される関連定数とともに示します。

Note すべてのユーザ モジュール API の場合と同じように、API 関数を呼び出すことで A と X レジスタの値が変更されることがあります。A と X の値が呼び出し後に必要な場合は、呼び出し元関数で A と X の値を保存してください。PSoC Designer のバージョン 1.0 以降では、効率を高めるために、この 「registers are volatile (レジスタの揮発性)」ポリシーが選択され、実施されています。C コンパイラは、自動的にこの条件で処理されています。アセンブラ言語のプログラマは、コードがこのポリシーを遵守していることも確認する必要があります。一部のユーザ モジュール API 関数では A と X は変更されないこともありますが、将来も変更されないという保証はありません。

API は、ADC の結果として生成されたデータを、初期化、構成、サンプリング開始、停止、読み込みできます。

DUALADC_Start

説明 :

このユーザ モジュールで必要なすべての初期化を実行し、スイッチド キャパシタ PSoC ブロックの出力レベルを設定します。

C プロトタイプ :

```
void  DualADC_Start (BYTE bPowerSetting)
```

アセンブリ :

```
mov   A, DualADC_HIGHPOWER
lcall  DualADC_Start
```

パラメータ：

PowerSetting: 出力レベルを指定する 1 バイト。リセットと構成の後、DualADC に割り当てられたアナログ PSoC ブロックの電力が遮断されます。C およびアセンブリで容易されたシンボル名、およびそれらに関連付けられた値は、以下の表で示されます。

記号名	値
DualADC_OFF	0
DualADC_LOWPOWER	1
DualADC_MEDPOWER	2
DualADC_HIGHPOWER	3

出力レベルは、アナログ性能に影響します。正しい出力設定はデータ クロックのサンプル レートの影響を受けやすく、アプリケーションごとに決める必要があります。開発開始時にはフル出力を選択することを推奨します。その後、テストを実行して、出力設定をどれだけ低く設定するかを決めることができます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_Stop

説明：

スイッチド キャパシタ積分器ブロックの出力レベルを 0ff に設定します。この設定は、DualADC が使用されておらず、節電したいときに行います。このルーチンは、アナログ スイッチ キャパシタ ブロックの電源を遮断し、デジタル ブロックを無効化します。最も低い消費電力を実現するには、クロックをデジタル ブロックから取り除く必要もあります。

C プロトタイプ：

```
void  DualADC_Stop()
```

アセンブリ：

```
lcall  DualADC_Stop
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

DUALADC_SetPower

説明：

スイッチド キャパシタ PSoC ブロックの出力レベルを設定します。

C プロトタイプ：

```
void  DualADC_SetPower (BYTE bPowerSetting)
```

アセンブリ：

```
mov   A, [bPowerSetting]
lcall  DualADC_SetPower
```

パラメータ：

PowerSetting: 上記の開始 API ルーチンで使用した PowerSetting パラメータと同じです。これにより、ADC を動作させながら、出力レベルを変更できます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_SetResolution

説明：

A/D 変換器の分解能を設定します。

C プロトタイプ：

```
void  DualADC_SetResolution (BYTE bResolution)
```

アセンブリ：

```
mov   A, [bResolution]
lcall  DualADC_SetResolution
```

パラメータ：

解像度：A/D 変換器の分解能は、デバイス エディタまたはユーザ ファームウェアのいずれかで設定できます。ファームウェアで設定されていない場合、デフォルトにより ADC は、デバイス エディタで設定されている分解能を使用します。分解能の値には、7 ~ 13 ビットを設定できます。

戻り値：

なし

特殊作用：

A/D 変換器を停止します。この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_GetSamples

説明：

ADC アルゴリズムを初期化して開始し、指定された数のサンプルを収集します。*M8C.inc* または *M8C.h* で定義されている M8C_EnableGInt マクロを呼び出して、グローバル割り込みを有効化します。

C プロトタイプ：

```
void  DualADC_GetSamples (BYTE bNumSamples)
```

アセンブリ：

```
mov   A, [bNumSamples]
lcall  DualADC_GetSamples
```

パラメータ：

bNumSamples: 取得するサンプル数を設定する 8-bit の値。値が 「0」の場合、ADC は継続的に実行されます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_StopAD

説明：

ADC を即座に停止します。

C プロトタイプ：

```
void  DualADC_StopAD()
```

アセンブリ：

```
lcall  DualADC_StopAD
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

DUALADC_fIsDataAvailable

説明：

データ変換が完了し、データを読み取る準備が整うと、非ゼロ値を返します。

C プロトタイプ：

```
BYTE  DualADC_fIsDataAvailable()
```

アセンブリ：

```
lcall  DualADC_fIsDataAvailable
```

パラメータ：

なし

戻り値：

データを使用できる場合は、非ゼロ値を返します。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_iGetData1

説明：

ADC 入力 1 用に最後に変換したデータを返します。データが有効なことを確認するため、データを取得する前に DUALADC_fIsDataAvailable() を呼び出してください。データは、次の変換サイクルが完了する前に取得する必要があります。取得しなかった場合、データは上書きされます。積分時間の最後にこの関数を正確に呼び出さないと、返されるデータが壊れる可能性があります。したがって、サンプリング レートより高い周波数でデータを取得することを強く推奨します。また、これを保証できない場合は、この関数を呼び出す前に割り込みをオフにすることを推奨します。

C プロトタイプ：

```
INT   DualADC_iGetData1()
```

アセンブリ：

```
lcall  DualADC_iGetData1
```

パラメータ：

なし

戻り値：

変換された整数値が返されます。アセンブラでは、X レジスタで MSB、累算器で LSB が返されます。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_iGetData2

説明：

ADC 入力 2 用に最後に変換したデータを返します。データが有効なことを確認するため、データを取得する前に DUALADC_fIsDataAvailable() を呼び出してください。データは、次の変換サイクルが完了する前に取得する必要があります。取得しなかった場合、データは上書きされます。積分時間の最後にこの関数を正確に呼び出さないと、返されるデータが壊れる可能性があります。したがって、サンプリング レートより高い周波数でデータを取得することを強く推奨します。また、これを保証できない場合は、この関数を呼び出す前に割り込みをオフにすることを推奨します。

C プロトタイプ：

```
INT   DualADC_iGetData2()
```

アセンブリ：

```
lcall  DualADC_iGetData2
```

パラメータ：

なし

戻り値：

変換された整数値が返されます。アセンブラでは、X レジスタで MSB、累算器で LSB が返されます。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_ClearFlag

説明：

データ使用可能フラグをクリアします。

C プロトタイプ：

```
void  DualADC_ClearFlag()
```

アセンブリ：

```
lcall  DualADC_ClearFlag
```

パラメータ

なし

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_iGetData1ClearFlag

説明：

ADC 入力 1 用に最後に変換したデータを返し、データ使用可能フラグをクリアします。データが有効なことを確認するため、データを取得する前に DUALADC_fIsDataAvailable() を呼び出してください。データは、次の変換サイクルが完了する前に取得する必要があります。取得しなかった場合、データは上書きされます。積分時間の最後にこの関数を正確に呼び出さないと、返されるデータが壊れる可能性があります。したがって、サンプリング レートより高い周波数でデータを取得することを強く推奨します。また、これを保証できない場合は、この関数を呼び出す前に割り込みをオフにすることを推奨します。

C プロトタイプ：

```
INT   DualADC_iGetData1ClearFlag()
```

アセンブリ：

lcall DualADC_iGetData1ClearFlag

パラメータ：

なし

戻り値：

変換された整数値が返されます。アセンブラでは、X レジスタで MSB、累算器で LSB が返されます。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

DUALADC_iGetData2ClearFlag

説明：

ADC 入力 2 用に最後に変換したデータを返し、データ使用可能フラグをクリアします。データが有効なことを確認するため、データを取得する前に DUALADC_fIsDataAvailable() を呼び出してください。データは、次の変換サイクルが完了する前に取得する必要があります。取得しなかった場合、データは上書きされます。積分時間の最後にこの関数を正確に呼び出さないと、返されるデータが壊れる可能性があります。したがって、サンプリング レートより高い周波数でデータを取得することを強く推奨します。また、これを保証できない場合は、この関数を呼び出す前に割り込みをオフにすることを推奨します。

C プロトタイプ：

```
INT   DualADC_iGetData2ClearFlag()
```

アセンブリ：

```
lcall  DualADC_iGetData2ClearFlag
```

パラメータ：

なし

戻り値：

変換された整数値が返されます。アセンブラでは、X レジスタで MSB、累算器で LSB が返されます。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

Note 関数 DUALADC_ClearFlag、DUALADC_iGetData1ClearFlag、DUALADC_iGetData2ClearFlag はすべて、同じフラグをクリアします。これらはいずれも、変換完了フラグをクリアするときに、最大の柔軟性を実現するために含まれています。ユーザは、A/D 変換が終了したら、片方または両方のチャネルの結果を無視して、データを取得せずにフラグをクリアにすることもできます。

ファームウェア ソースコードの例

次に、アセンブリ言語で書かれたサンプルプロジェクトを示します。

```
;;; Sample ASM Code for the DualADC
;;;
;;; Continuously sample using the DualADC and store the values in RAM.
;;;

include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

;; Create storage for readings
area bss(RAM)
iResult1:       BLK   2  ; ADC1 result storage
iResult2:       BLK   2  ; ADC2 result storage
export iResult1          ; Export results in case they are
export iResult2          ; used elsewhere.

area text(ROM,REL)
export _main

_main:

    M8C_EnableGInt                ; Enable interrupts
    mov   A, 10                   ; Set resolution to 10 Bits
    call  DUALADC_SetResolution

    mov   A, DUALADC_HIGHPOWER    ; Set Power and Enable A/D
    call  DUALADC_Start

    mov   A, 00h                  ; Start A/D in continuous sampling mode
    call  DUALADC_GetSamples
```

```
;A/D conversion loop
loop1:

wait:                               ; Poll until data is complete
    call  DUALADC_fIsDataAvailable
    jz    wait
    call  DUALADC_ClearFlag         ; Reset flag

    call  DUALADC_iGetData1         ; Get ADC1 Data (X=MSB A=LSB)
    mov   [iResult1+1],A            ; Store LSB
    mov   [iResult1+0],X            ; Store MSB

    call  DUALADC_iGetData2         ; Get ADC2 Data (X=MSB A=LSB)
    mov   [iResult2+1],A            ; Store LSB
    mov   [iResult2+0],X            ; Store MSB
    jmp   loop1
```

次に、C 言語で書かれたサンプルプロジェクトを示します。

```
//-------------------------------------------------------------------------
//  Sample C Code for the DualADC
//  Continuously Sample and call a user function with the data.
//  This example differs from the ASM example, in that the DataAvailable
//  flag is automatically cleared when the second value is read, instead of
//  clearing the flag prior to reading the data.
//
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

extern void User_Function(int iResult1, int iResult2);

void main(void)

{

   int iResult1, iResult2;

   M8C_EnableGInt;                        // Enable global interrupts
   DUALADC_Start(DUALADC_HIGHPOWER);      // Turn on Analog section
   DUALADC_SetResolution(10);             // Set resolution to 10 Bits
   DUALADC_GetSamples(0);                 // Start ADC to read continuously

   for(;;)

   {

      while(DUALADC_fIsDataAvailable() == 0);  // Wait for data to be ready
      iResult1 = DUALADC_iGetData1();          // Get Data from ADC Input1
      iResult2 = DUALADC_iGetData2ClearFlag(); // Get Data from ADC Input2
                                               // and clear data ready flag

      User_Function(iResult1,iResult2);        // User function to use data
   }

}
```

設定レジスタ

これらのレジスタは、初期化および API ライブラリによって構成されます。ユーザは、これらのレジスタを直接変更または読み取る必要はありません。このセクションは参照用です。

ADC は、スイッチド キャパシタ PSoC ブロックです。ADC は、アナログ変調器を作成するために構成されています。変調器を構築するため、ブロックは、入力値をデジタル パルス ストリームに変換するリファレンス フィードバックを持つ積分器として構成します。入力マルチプレクサは、デジタル化する信号を決定します。

Table 9. ブロック ADC1: レジスタ CR0

ビット	7	6	5	4	3	2	1	0
値	1	0	0	1	0	0	0	0

Table 10. ブロック ADC1: レジスタ CR1

ビット	7	6	5	4	3	2	1	0
値	ACMux、AMux			0	0	0	0	0

ブロックがタイプ「A」ブロックに配置されている場合、ACMux が使用されます。ブロックがタイプ「B」ブロックに配置されている場合、AMux が使用されます。どちらのフィールド値も、ユーザによる入力接続によって異なります。

Table 11. ブロック ADC1: レジスタ CR2

ビット	7	6	5	4	3	2	1	0
値	0	1	1	0	0	0	0	0

Table 12. ブロック ADC1: レジスタ CR3

ビット	7	6	5	4	3	2	1	0
値	1	1	1	FSW0	0	0	0	0

FSW0 は、PWM 割り込みハンドラおよび各種 API によって使用されます。値が「0」の場合、ADC は無効化された積分器となります。値が「1」の場合、ADC は有効化された積分器となります。

Table 13. ブロック ADC2: レジスタ CR0

ビット	7	6	5	4	3	2	1	0
値	1	0	0	1	0	0	0	0

Table 14. ブロック ADC2: レジスタ CR1

ビット	7	6	5	4	3	2	1	0
値	ACMux、AMux			0	0	0	0	0

ACMux は、タイプ「A」のブロックにブロックを配置する場合に使用します。AMux は、タイプ「B」のブロックにブロックを配置する場合に使用します。どちらのフィールド値も、ユーザによる入力接続によって異なります。

Table 15. ブロック ADC2: レジスタ CR2

ビット	7	6	5	4	3	2	1	0
値	0	1	1	0	0	0	0	0

Table 16. ブロック ADC2: レジスタ CR3

ビット	7	6	5	4	3	2	1	0
値	1	1	1	FSW0	0	0	0	0

FSW0 は、TMR 割り込みハンドラおよび各種 API によって使用されます。値が 「0」の場合、ADC は無効化された積分器となります。値が 「1」の場合、ADC は有効化された積分器となります。

PWM16 は、ADC の積分時間を制御するために使用されるデジタル PsoC ブロックです。比較値には 2 ビット +2 を設定し、時間には CalcTime と比較値の和を設定します。

Table 17. ブロック PWM16_MSB：レジスタ関数

ビット	7	6	5	4	3	2	1	0
値	0	0	1	比較タイプ	割り込みタイプ	0	0	1

Compare Type （ 比較タイプ) は、キャプチャ比較が 「以下」と 「未満」のどちらなのかを示すフラグです。Interrupt Type （ 割り込みタイプ) は、キャプチャ イベントと秒読み条件のどちらで割り込みをトリガするかを示すフラグです。どちらのパラメータも、デバイス エディタで設定します。

Table 18. ブロック PWM16_LSB：レジスタ関数

ビット	7	6	5	4	3	2	1	0
値	0	0	0	比較タイプ	0	0	0	1

Compare Type （ 比較値) は、比較関数が 「以下」と 「未満」のどちらに設定されているかを示すフラグです。このパラメータは、デバイス エディタで設定します。

Table 19. ブロック PWM16_MSB：レジスタ入力

ビット	7	6	5	4	3	2	1	0
値	0	0	1	1	Clock (クロック)			

クロックは 16 のクロック ソースの 1 つから入力クロックを選択します。このパラメータは、デバイス エディタで設定します。

Table 20. ブロック PWM16_LSB：レジスタ入力

ビット	7	6	5	4	3	2	1	0
値	Enable (イネーブル)				Clock (クロック)			

Enable は 16 の入力ソースからデータ入力を 1 つ選択し、Clock は 16 のソースからクロック入力を 1 つ選択します。どちらのパラメータも、デバイス エディタで設定します。

Table 21. ブロック PWM16_MSB：レジスタ出力

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	Output Enable	Output Sel	

Output Enable は、出力が有効なことを示すフラグです。Output Sel は、PWM16 の出力が送られる場所を示すフラグです。どちらのパラメータも、デバイス エディタで設定します。

Table 22. ブロック PWM16_LSB：レジスタ出力

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 23. ブロック PWM16_MSB：カウント レジスタ DR0

ビット	7	6	5	4	3	2	1	0
値	Count(MSB)							

カウントは PWM16 MSB から PWM です。この値は、PWM16 API を使用して読み取れます。

Table 24. ブロック PWM16_LSB：カウント レジスタ DR0

ビット	7	6	5	4	3	2	1	0
値	Count(LSB)							

カウントは PWM16 LSB から PWM です。この値は、PWM16 API を使用して読み取れます。

Table 25. ブロック PWM16_MSB：時間レジスタ DR1

ビット	7	6	5	4	3	2	1	0
値	Period(MSB)							

Period は、イネーブルまたは最終カウント条件によってカウンタ レジスタにロードされる時間値の MSB を保持します。この値は、デバイス エディタおよび PWM16 API で設定できます。

Table 26. ブロック PWM16_LSB：時間レジスタ DR1

ビット	7	6	5	4	3	2	1	0
値	Period(LSB)							

Period は、イネーブルまたは最終カウント条件によってカウンタ レジスタにロードされる時間値の LSB を保持します。この値は、デバイス エディタおよび PWM16 API で設定できます。

Table 27. ブロック PWM16_MSB：パルス幅レジスタ DR2

ビット	7	6	5	4	3	2	1	0
値	Pulse Width(MSB)							

PulseWidth は、比較イベントの生成に使用されるパルス幅の値の MSB を保持します。この値は、デバイス エディタおよび PWM16 API で設定できます。

Table 28. ブロック PWM16_LSB：パルス幅レジスタ DR2

ビット	7	6	5	4	3	2	1	0
値	Pulse Width(LSB)							

PulseWidth は、比較イベントの生成に使用されるパルス幅の値の LSB を保持します。この値は、デバイス エディタおよび PWM16 API で設定できます。

Table 29. ブロック PWM16_MSB：制御レジスタ CR0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Start/ Stop(0)

Start/Stop は、LSB 制御レジスタの値によって制御されます。ゼロに設定してください。

Table 30. ブロック PWM16_LSB：制御レジスタ CR0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Start/ Stop

Start/Stop は、PWM16 が設定されている場合にイネーブルであることを示します。この設定は、PWM16 API を使用して変更します。

CNT は、カウンタとして構成されるデジタル PSoC ブロックです。DR0 の値が最後までカウントダウンされると、上位桁のソフトウェア カウンタを減算するために割り込みが呼び出され、CNT が DR1 から再ロードされます。データは、DR2 を通して出力されます。

Table 31. ブロック CNT1: レジスタ関数

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

Table 32. ブロック CNT1: レジスタ入力

ビット	7	6	5	4	3	2	1	0
値	データ				Clock (クロック)			

データは、ADC ブロックが配置されたコラム コンパレータを選択します。クロックは、16 のソースから入力クロックを選択し、デバイス エディタで設定されます。

Table 33. ブロック CNT1: レジスタ出力

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 34. ブロック CNT1: レジスタ DR0

ビット	7	6	5	4	3	2	1	0
値	カウント値							

Table 35. ブロック CNT1: レジスタ DR1

ビット	7	6	5	4	3	2	1	0
値	1	1	1	1	1	1	1	1

Table 36. ブロック CNT1: レジスタ DR2

ビット	7	6	5	4	3	2	1	0
値	Data Out							

Data Out は、カウンタ値を取得するために API によって使用されるレジスタです。

Table 37. ブロック CNT1: レジスタ CR0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable (イネーブル)

Enable は、CNT を有効にします。この値は、DualADC API によって変更および制御されます。

Table 38. ブロック CNT2: レジスタ関数

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

Table 39. ブロック CNT2: レジスタ入力

ビット	7	6	5	4	3	2	1	0
値	データ				Clock (クロック)			

データは、ADC ブロックが配置されたコラム コンパレータを選択します。クロックは、16 のソースから入力クロックを選択し、デバイス エディタで設定されます。

Table 40. ブロック CNT2: レジスタ出力

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 41. ブロック CNT2: レジスタ DR0

ビット	7	6	5	4	3	2	1	0
値	カウント値							

Table 42. ブロック CNT2: レジスタ DR1

ビット	7	6	5	4	3	2	1	0
値	1	1	1	1	1	1	1	1

Table 43. ブロック CNT2: レジスタ DR2

ビット	7	6	5	4	3	2	1	0
値	Data Out							

Data Out は、カウンタ値を取得するために API によって使用されるレジスタです。

Table 44. ブロック CNT2: レジスタ CR0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable (イネーブル)

Enable は、CNT を有効にします。この値は、DualADC API によって変更および制御されます。

Table 45. レジスタ : INT_MSK1

ビット	7	6	5	4	3	2	1	0
値								

ここでは、個々の割り込みを有効にするために、TMR ブロックと CNT ブロックに対応するマスク ビットを設定します。実際のマスク値は、各ブロックの配置位置によって決まります。

改訂履歴

バージョン	作成者	説明
2.2	DHA	以下を確認するために DRC を追加。 1. デジタル リソースとアナログ リソース間でソース クロックが異なっているかどうか。 2. ADC クロックが CPU クロックより上位かどうか。

Note PSoC Designer 5.1 により、全ユーザ モジュール データシートが改訂されます。このセクションは、現在および以前のユーザ モジュール バージョン間の差を高レベルに説明するものです。

Copyright © 2001-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer ™ and Programmable System-on-Chip ™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.