AN12217

S32K1xx および S32M24x ADC ガイドライン、仕様および構成

改訂 2.0 — 2024 年 9 月 5 日 **アプリケーションノート**

ドキュメント情報

情報	内容
キーワード	S32K1xx、S32M24x、ADC
要約	このアプリケーションノートでは、ADC モジュールを最大限に活用するために、ADC の用語、ベストプラクティス、および構成例を理解するための情報を提供します。

1 はじめに

NXP S32K1xx および S32M24x 自動車用マイクロコントローラデバイスは、アナログ入力信号の収集およびデジタル化に使用される 12 ビットの逐次近似アナログ-デジタル変換器(SAR ADC)を特徴とします。

このアプリケーションノートでは、ADC モジュールを最大限に活用するための次の基本的なトピックについて説明します。

- ADC の一般的な用語、エラーソース、および仕様を理解する。
- 測定精度を向上させるためのベストプラクティス。
- S32K1xx および S32M24x ファミリの一般的なトリガ構成の例。

2 ADC の概念、エラーソースおよび仕様

このセクションでは、ADC を特徴付けるために使用される概念と用語、潜在的なエラーソース、および S32K1xx および S32M24x ファミリのデータシートに記載されている仕様パラメータについて説明します。

2.1 ADC 基本概念

分解能: アナログ入力信号を表す ADC デジタル出力のビット数。S32K1xx および S32M24x デバイスの場合、分解能は 8、10、または 12 ビットに設定できます。

リファレンス電圧: ADC は、デジタル出力を生成するために比較されるアナログ入力との逐次近似比較を作成するために使用されるリファレンス電圧を必要とします。デジタル出力は、このリファレンス電圧に対するアナログ入力の比率です。

$$VREF = VREFH - VREFL \tag{1}$$

ここで、

VREFH = 高リファレンス電圧

VREFL = 低リファレンス電圧

ADC 出力計算式: ADC の変換式は、特定のアナログ入力電圧に対応するデジタル出力を計算するために使用されます。この式は、誤差のない理想的な A/D 変換を想定しています。

$$ADCresult = \frac{(2^N)(Vin)}{VREF} \tag{2}$$

ここで、

ADC result = 変換から得られるデジタル出力値

N = ADC 分解能

VREF = リファレンス電圧

Vin = アナログ入力電圧

最下位ビット(LSB): 最下位ビット(LSB)は、ADC の最小分解能、すなわちデジタル出力に変化を引き起こす最小増分電圧に等しい電圧の単位です。

LSB は、リファレンス電圧を ADC の最大カウントで割った値です。

$$LSB = \frac{VREF}{2^N} \tag{3}$$

N = ADC 分解能。S32K1xx および S32M24x の場合、これは 8/10/12 ビットになります。

VREF = アナログリファレンス電圧。

ADC 実伝達関数: ADC は入力電圧を対応するデジタルコードに変換します。この動作を記述する曲線が*実伝達関数*であり、ADC モジュール自体に固有のすべての誤差を含みます。

ADC 理想伝達関数: *理想伝達関数*は、ADC が完全に線形であると仮定した場合、または入力電圧の所与の変化が入力の初期レベルに関係なく変換コードに同じ変化を生じると仮定した場合の ADC の挙動を表します。理想伝達関数がステップに分割される方法は、ADC が使用する量子化の方法によって異なります。次の 2 つの方法があります。

- **非補償量子化:** 最初のステップは 1 LSB で実行され、後続の各ステップは 1 LSB 間隔で実行され、最後のステップは VREFH-1 LSB で実行されます。
- **½LSB 補償量子化:** 最初のステップは½LSB で実行され、後続の各ステップは 1 LSB 間隔で実行され、最後のステップは VREFH - 1½LSB で実行されます。

以下の図は、3 ビット分解能で VREF=8 V の場合の、非補償および½LSB 補償方式の理想伝達関数グラフを示しています。

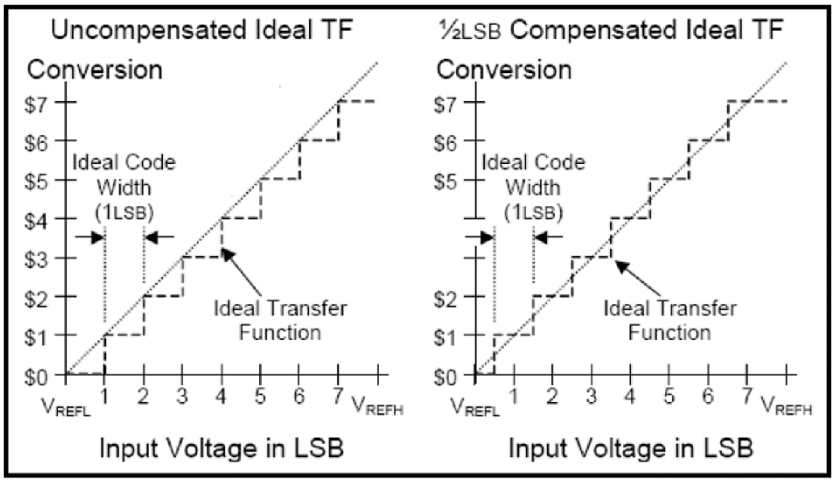

図 1.理想伝達関数

2.2 ADC 測定におけるエラーソース

このセクションでは、ADC が正確な A/D 測定を実行することを妨げるいくつかの典型的な要因を示します。

リファレンス電圧ノイズ

ADC 出力はアナログ入力電圧とリファレンス電圧に正比例します。不安定なリファレンス電圧(例えば、電源レールのノイズによって引き起こされる)は、変換されたデジタル出力に変化を引き起こします。

例:

- リファレンス電圧が 5V で入力電圧が 1V の場合、式 2 を使用すると、12 ビット分解能の ADC 結果は 819 となります。
- 絶対リファレンス電圧が 50mV 増加する(すなわち、VREF = 5.05V)と、同じ 1 V 入力電圧に対する新しい変換値は 811 になります。
- 結果として生じるリファレンス電圧ノイズ誤差は 811-819 = - 8 LSB となります。

アナログ入力信号ノイズ

アナログ入力信号のわずかではあるが高周波の変動は、ADC サンプリング時間中に大きな変換誤差を引き起こす可能性があります。ノイズは、周囲の電気機器からの電磁放射(EMI ノイズ)によって誘発されることがあります。そのため、変換精度に悪影響を及ぼします。

入力信号に存在するノイズが 1 LSB より大きい場合、信号の変動によって最下位ビットが絶えず変化するため、変換結果の信頼できるビット数が実質的に減少します。

アナログ信号ソース抵抗

アナログ信号ソースのインピーダンス、またはアナログ信号ソースと入力ピン間の直列抵抗(R_{IN})は、ピンに電流が流れるため、電圧降下を引き起こします。これは、サンプリングされる入力信号を駆動するソースへの ADC の「ルックアウト」として観察される抵抗として理解することができます。

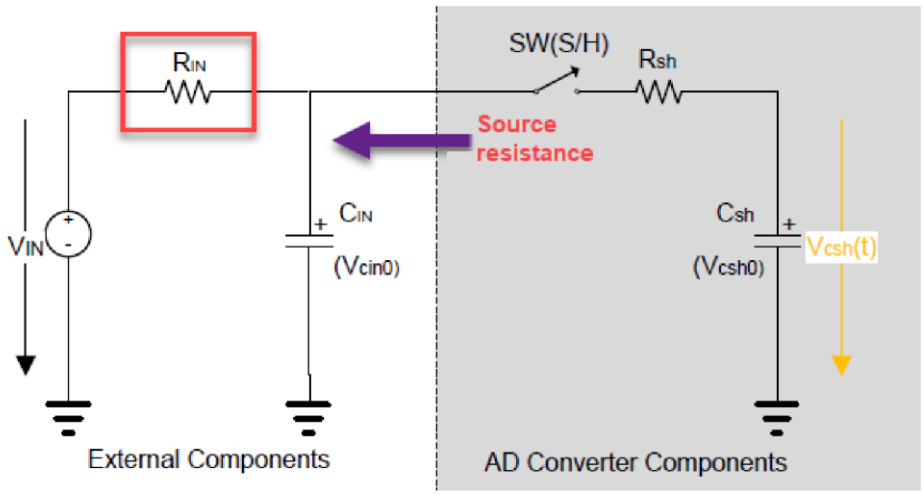

図 2.アナログ信号ソース抵抗

図 2 に示すように、入力信号のサンプリングは、内部コンデンサ(Csh)を充電し、抵抗 Rsh でスイッチを制御することによって達成されます。ソース抵抗(R_{IN})を追加すると、ホールドコンデンサを完全に充電するために必要な時間が増加します。サンプリング時間がコンデンサの充電に必要な時間よりも短い場合、ADC によって変換されたデジタル値は実際の値よりも小さくなります。

このため、アナログ入力信号ソースの抵抗値が ADC 仕様の範囲内であることを確認する必要があります。S32K および S32M デバイスのデータシートでは、このパラメータはソースインピーダンス(R_S)として表示されます。

温度の影響

システムの温度は ADC の精度に大きな影響を与える可能性があり、主にオフセット誤差ドリフトとゲイン誤差ドリフトを引き起こします。ADC のリファレンス電圧も温度変化によって変化します。これらの誤差は、マイクロコントローラのファームウェアを調整することで補正できます。例えば、内部バンドギャップ電圧を監視してリファレンス電圧が変化していないことを確認するか、またはアプリケーションの温度範囲にわたってシステムを特性評価して誤差を考慮します。

I/O ピンクロストーク

現在 ADC によってサンプリングされているアナログ入力ピン付近での I/O のスイッチングは、ピン間の容量性結合による変換にノイズを導入します。クロストークは、互いに近接して走るか、または互いに交差する PCB トラックによって引き起こされます。デジタル信号と I/O を内部的に切り替えると、高周波ノイズが発生します。

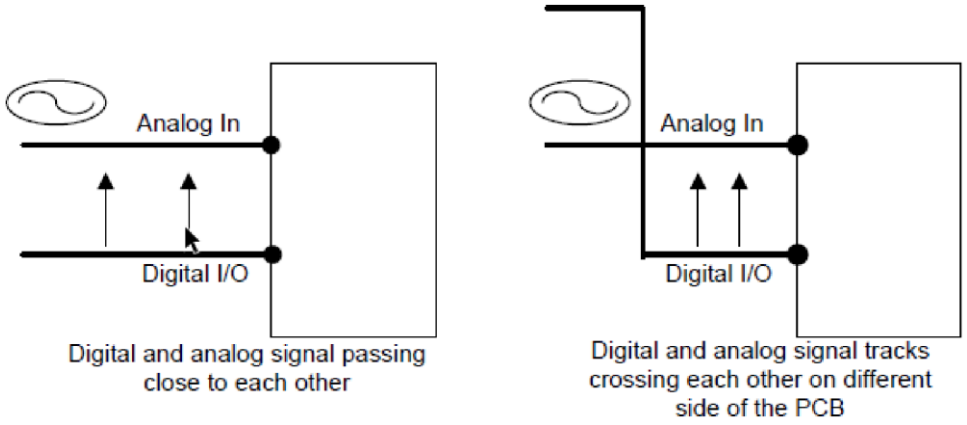

図 3.I/O ピンクロストーク

2.3 S32K1xx および S32M24x ADC の仕様

このセクションでは、S32K1xx および S32M24x デバイスのデータシートにある SAR ADC の仕様を統合するパラメータについて説明します。

ADC クロック周波数(f_{ADCK}): SAR ADC モジュールの入力変換クロックの周波数。この周波数は、特定の A/D 変換の変換時間を決定する主な要因です。内部 ADC 近似メカニズムは、このクロックを変換ステートマシンのさまざまな遷移のベースタイムとして使用します。

ADC 変換周波数(f_{CONV}): 「変換レート」または「サンプリングレート」とも呼ばれ、アナログ信号をデジタル結果に変換する速度の尺度です。変換周波数が高いほど、決定された時間ウィンドウでより多くのサンプルを取得できます。一方、変換周波数が低いほど、同じ期間に取得されるサンプルが少なくなります。

変換レートは主に次の要因に依存します。

- ADC クロック周波数
- ハードウェア平均化の有効/無効
- サンプル数
- 構成(単一または連続変換)

総変換時間の計算方法については、デバイスのリファレンスマニュアルを参照してください。

微分非直線性誤差(DNL)

微分非直線性誤差は「コード幅誤差」です。ここで、コード幅は、所定の ADC 変換値をもたらす入力電圧 V_{ADIN} の範囲です。理想的には、アナログ入力電圧が 1 LSB 変化するとデジタルコードが変化します。したがって、DNL は実際のコード幅と理想的な遷移電圧である 1 LSB との差です。

DNL は、他のコードとは独立して、各 ADC 変換コードに個別に測定されることに注意してください。

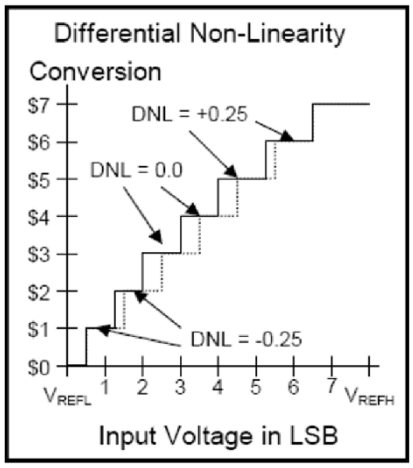

図 4.微分非直線性誤差(DNL)

DNL 誤差から導出される 2 つの重要な性能指数があります。

- **ミッシングコード:** 電圧の微小な変化によって 2 つのデジタルカウントの結果が変化し、中間コードが設定されない場合、ADC にはミッシングコードがあります。-1.0 LSB の DNL は、ADC にミッシングコードがあることを示します。
- **単調性:** ADC は、電圧の増加とともに変換結果を連続的に増加させる場合(およびその逆)、単調になります。非単調 ADC は、より高い入力電圧に対してより低い変換結果を与える可能性があり、これはまた、同じ変換が 2 つの別々の電圧範囲から生じ得ることを意味する場合があります。1.0 LSB を超える DNL は非単調性を示します。

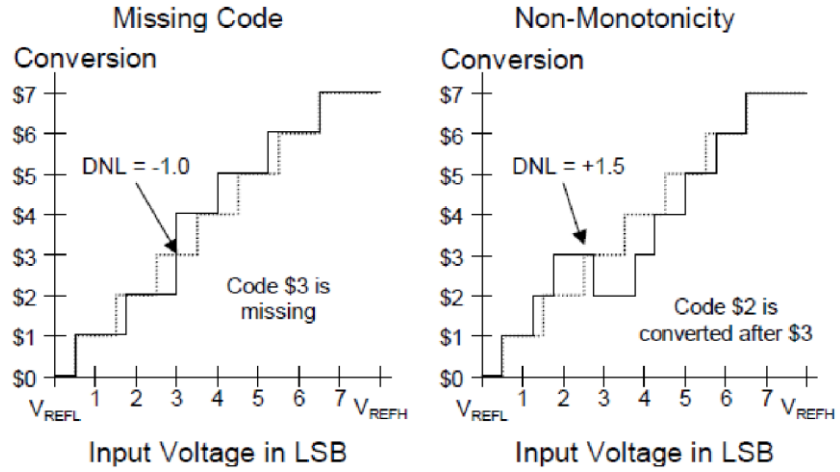

図 5.ミッシングコードと非単調性

積分非直線性誤差(INL)

DNL は理想と比較して任意の所与の ADC コードに対して与えられますが、積分非直線性誤差(INL)は、変換コード 1 から対象のコードまでのすべての DNL 誤差の累積効果です。その場合、基本的に INL は式 4 で表すことができる DNL の和になります。

$$INL(x) = \sum_{i=1}^{(x-1)} DNL(i) \quad (4)$$

以下の図は、個々の DNL の累積効果に基づく INL の形式を示しています。

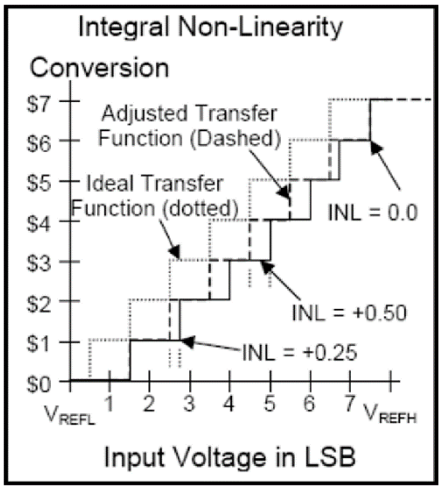

図 6.
積分非直線性誤差(INL)

総合未調整誤差(TUE)

TUE は、オフセット、ゲイン、直線性、量子化誤差の合計です。これは、ADC の実際に期待される精度を提供するため、重要なパラメータです。任意の入力電圧 V_{ADIN} に対して、TUE は理想的な期待値と比較して得られた変換値の差であり、LSB で表されます。

「未調整」という用語は、TUE が生の変換データを介して測定され、ADC 固有の誤差を除去するために何らかの方法で正規化されていないことを意味します。

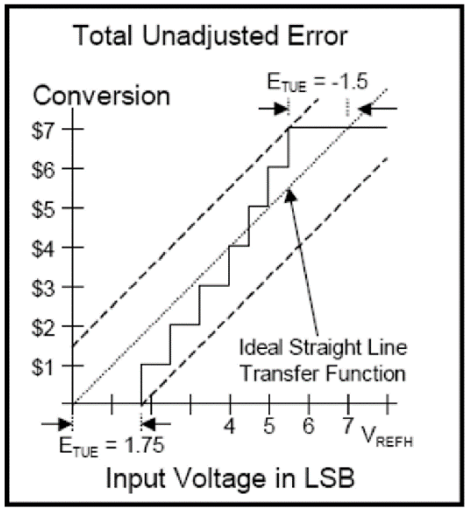

図 7.
総合未調整誤差(TUE)

DNL、INL、TUE は、スタティック/DC 入力での変換時の ADC 誤差を表します。したがって、これらの誤差は ADC スタティック/DC 性能を表します。

3 精度を向上させるためのベストプラクティス

このセクションでは、ADC 測定の精度を向上させるための一般的な推奨事項と優れたプラクティスについて説明します。

ADC 校正

S32K1xx および S32M24x ファミリの SAR ADC には、各 IC ユニットの工場出荷時の容量変動を補正するために内部サンプリングコンデンサバンクを調整する自己校正メカニズムがあります。データシートで指定された ADC 精度を得るために、各パワーオンリセット後に ADC の自己校正を起動することは必須です。

校正は 1 回だけ実行できます。その後、校正レジスタの値を不揮発性メモリに保存してリセット後に復元することで、後続の校正を回避できます。

可能な限り最適な校正を得るための推奨事項を以下に示します。

- すべてのデジタル IO を無音にし、不要なモジュールを無効にする必要があります。
- VREFH が高いほど ADC コード幅が大きくなるため、VREFH は仕様の範囲内で可能な限り安定して高くする必要があります。
- 絶縁された VREFH ピンが理想的です。
- アプリケーション内の ADC クロックが 25MHz よりも速い場合、ADC 自己校正は 25MHz 以下の ADC クロックで実行する必要があります。それ以外の場合、アプリケーション内の ADC クロックが 25MHz 以下に設定されている場合、校正を実行するときは同じ ADC 周波数を使用することをお勧めします。
- ハードウェア平均化は最大 32 サンプルに設定する必要があります。
- 校正は、POR 後に室温で 1 回行う必要があります。

内部校正メカニズムの詳細については、リンクのドキュメントを復習してください。

リファレンス電圧および電源

ADC は VREF または VDDA をアナログリファレンスとして使用するため、電源は良好なライン、負荷調整、および温度ドリフトを有する必要があります。そのため、VREF が異なる負荷で安定していることが不可欠です。回路の一部をオンにして負荷が増加した場合は、電流の増加によって電圧が低下することはありません。

電圧が広い電流範囲にわたって安定している場合、電源の負荷調整は良好です。ライン調整値が低いほど、調整が良好です。

同様に、負荷調整値が低いほど、電圧出力の調整と安定性が高くなります。高精度レギュレータで VREF のリファレンス電圧を使用することも可能です。

温度ドリフトは電圧リファレンスを考慮するためのもう 1 つの重要な要素です。特に一部のアプリケーションでは、ADC 精度が全温度範囲内で指定されます。

バンドギャップを使用したリファレンス電圧のモニタリング

VREF の変化をモニタリングするためのオプションは、内部バンドギャップ ADC チャネルを使用することです。バンドギャップチャネルは、リファレンス電圧またはアナログ電源電圧から独立して 1 V の固定電圧を提供します。

手順は次のとおりです。

1. バンドギャップ(チャネル 27)の ADC 変換をトリガします。
2. 以下の式を用いて実際の VREF を計算します。

$$VREF\left(mV\right) = \frac{\left(1000\right)\left(2^{N}\right)}{BG\_ADCresult} \tag{5}$$

ここで、

N = ビット(8/10/12 ビット)の ADC 分解能

BG_ADCresult = バンドギャップチャネルの ADC 変換結果

1. アプリケーションの電圧計算では、式 5 で得られた VREF を考慮します。

アナログソース抵抗整合

ADC の概念、誤差ソースおよび仕様で説明されているように、アナログソース抵抗は ADC の精度に重要な役割を果たします。このため、ソース抵抗をできるだけ低くすることが望ましいです。ユーザは、アナログ信号ソースの抵抗がデータシートに示された ADC 仕様の範囲内であることを常に確認する必要があります。

インピーダンス整合の一般的な方法は、アナログ信号ソースと ADC 入力ピンの間に外部演算増幅器を配置することです。ただし、外部オペアンプが追加されると、設計 BOM のコストが増加します。

ソース抵抗の高い信号を測定する場合、ADC を設定する際に次の点を考慮する必要があります。

- より低い ADC クロック周波数(f_{ADCK})
- より長いサンプル時間。S32K1xx および S32M24x デバイスのサンプリング時間は、ADC コンフィグレーションレジスタ 2(CFG2)の SMPLTS フィールドの値を大きくすることで増加できます。

外部 RC 収集回路の設計方法とコンポーネントの選択方法の詳細については、アプリケーションノート AN4373 を参照してください。このドキュメントでは 16 ビット SAR ADC と Kinetis などの他の NXP マイクロコントローラファミリについて言及していますが、S32K1xx と S32M24x の ADC モジュールは同じ基本アーキテクチャを共有しているため、この理論も適用できます。

I/O ピンクロストークの最小化

隣接する PCB トラックまたは MCU ピン間のクロストークによって発生するノイズは、中央にクリアアナロググランドトラックを配置してアナログ信号をシールドすることによって低減できます。以下の図は、このようなシールド手法を表したものです。

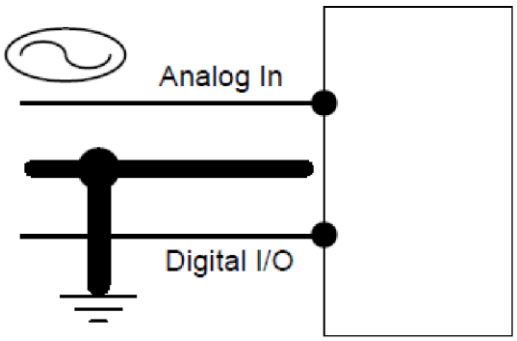

図 8.推奨される信号間のグラウンド

4 ADC トリガモードの例

S32K1xx および S32M24x ファミリの ADC は、変換を開始するためのトリガソースに関して柔軟な構成を提供します。このセクションでは、典型的なトリガ構成のために作成された例で、ADC のワークフローについて説明します。

サンプルコードは、S32K144 および S32M244 EVB ボードに基づいて作成しました。TRGMUX の例では、ポテンショメータをアナログ信号ソース、ユーザスイッチをトリガ入力として使用しています。

注: *このドキュメントでは、トリガ例の概要のみを説明します。ピン、クロック、SIM、ADC、PDB、TRGMUX モジュールの機能と構成設定の詳細については、リファレンスマニュアルを参照してください。*

4.1 ソフトウェアトリガ

サンプルコードについては、付録 A を参照してください。

ソフトウェアトリガは最も単純なトリガモードです。これは、ADCx_SC1A レジスタの ADCH フィールドへの書き込み後に、単一または連続変換を開始するだけです。S32K1xx および S32M24x の ADC は、いくつかのステータスおよびコンフィグレーション 1 レジスタ(SC1A から SC1AF まで)を提供しますが、ソフトウェアトリガモードに使用できるのは SC1A だけであることに注意してください。

この例では、外部ピン ADC0_SE12 が ADC 入力として使用されています。ADC0_SC1A[ADCH]への書き込みごとに新しい変換がトリガされます。変換が完了すると、結果は ADC0_RA レジスタに格納されます。

ワークフローの例

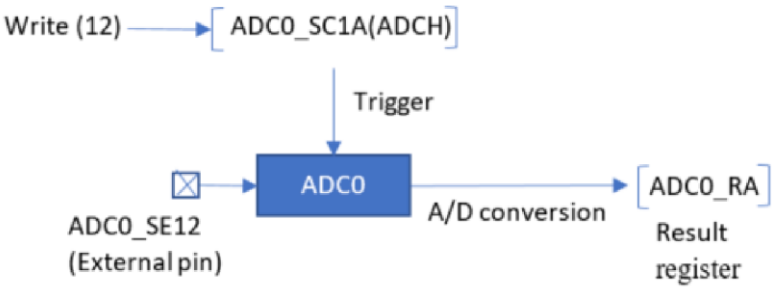

図 9.ADC ソフトウェアトリガワークフロー

4.2 PDB トリガ

サンプルコードについては、付録 B を参照してください。

PDB トリガ方式がデフォルトであり、ADC に推奨されるハードウェアトリガ方式です。この方法では、PDB タイマモジュールを使用して、同じチャネルまたは異なるチャネルから 1 つ以上の ADC 変換を定期的にトリガします。PDB をトリガとして使用する場合、PDB トリガが ADC モジュールに到達するために使用できるパスは 2 つあります。

1. ダイレクトパス: このパスは、SC1n レジスタ番号 4 以降(SC1E レジスタ～SC1AF レジスタに対応)の ADC 変換をトリガする際に使用されます。
2. PDB/TRGMUX 多重トリガパス: SC1n レジスタ 0～3(レジスタ SC1A、SC1B、SC1C、SC1D に対応)の変換をトリガする場合、トリガはトリガラッチングガスケットを通過します。ラッチングガスケットは、ADC トリガ要求をラッチする機能を提供し、ADC トリガ要求は一度に 1 つずつ処理されます。

この例では、ADC0 の外部チャネル 12(ADC0_SE12 ピン)に対する新しい変換が PDB によって毎秒トリガされます。PDB0/チャネル 0 のプリトリガ 4 は、ADC0_SC1E レジスタに基づく変換をトリガするために使用されます。そのため、「ダイレクトパス」モードを使用します。PDB タイマ自体は、最初にソフトウェアによってトリガされ、その後継続的に実行されます。

次の図は、トリガ(青色の点線)とプリトリガ(赤色の点線)のパスを示しています。

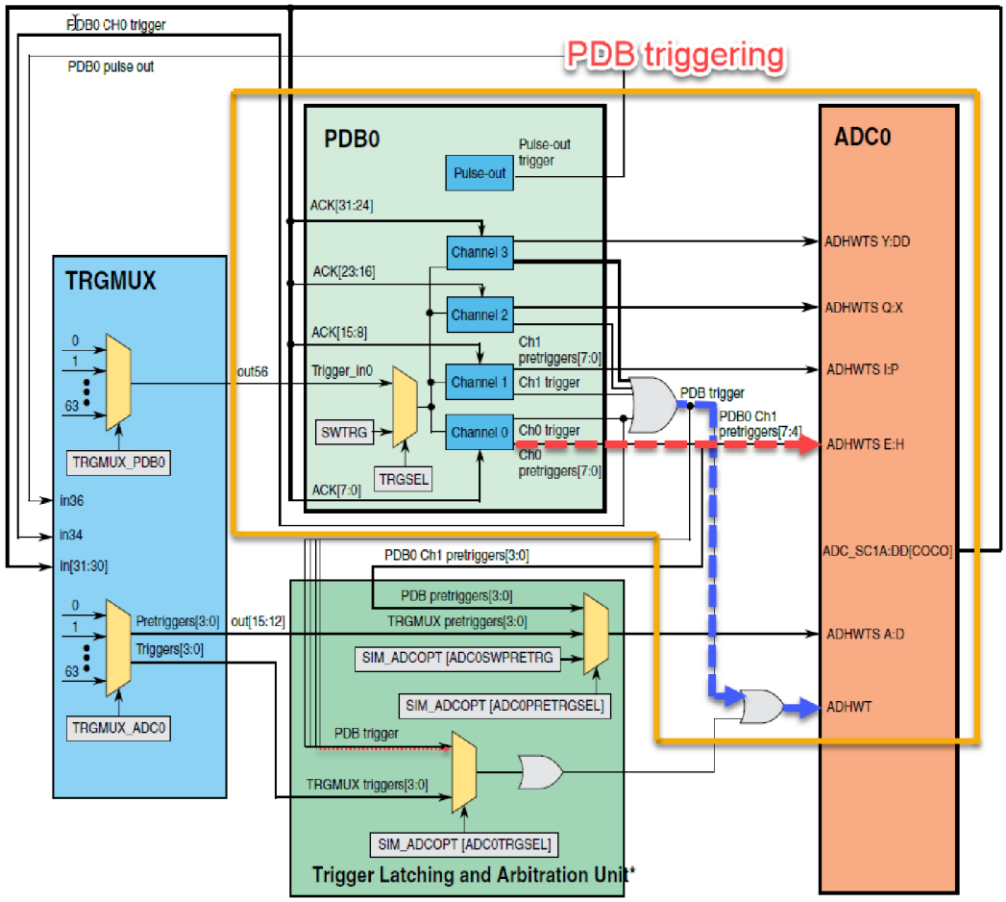

図 10.ダイレクトパスモードでの PDB トリガ

ワークフローの例

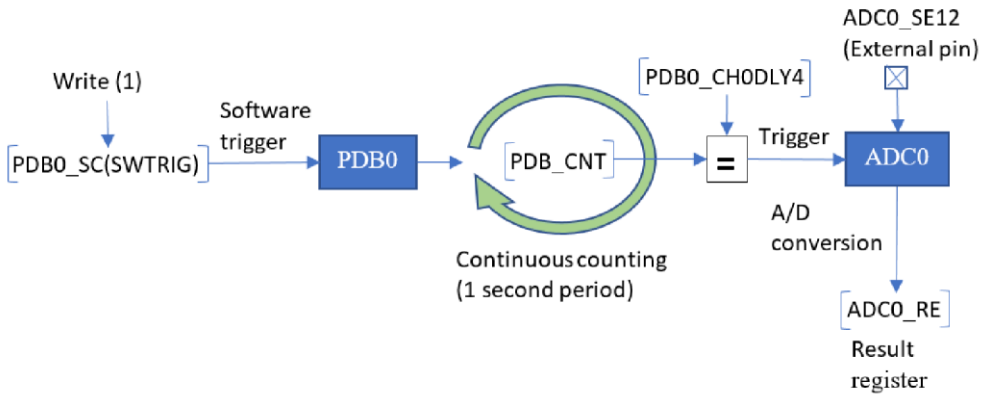

図 11.ADC と PDB トリガのワークフローの例

4.3 バックトゥバックモードでの PDB トリガ

サンプルコードについては、付録 C を参照してください。

バックトゥバックは、ADC 変換完了フラグが次の PDB チャネルのプリトリガとトリガ出力を一度に 1 つずつトリガする動作モードです。これは、複数の ADC チャネルを連続してサンプリングして変換する必要がある場合に特に便利です。前のセクションで説明した PDB トリガの 2 つのパス(ダイレクトおよび多重)は、このモードにも適用されます。

この例では、ADC0 の外部チャネル 12(ADC0_SE12 ピン)は、ダイレクトパスの PDB を使用して、毎秒 4 回連続して変換されます。PDB0/チャネル 0 のプリトリガ 4 は、対応する遅延レジスタ(PDB0_CH0DLY4)とのカウンタマッチにより有効にされます。一方、プリトリガ 5/6/7 は、対応する ADC0 COCO 変換フラグとともにバックトゥバックモードで自動的にトリガされます。使用される ADC チャネル設定は、したがって、ADC0_SC1E～ADC0_SC1H です。PDB タイマは、最初にソフトウェアによってトリガされます。

ワークフローの例

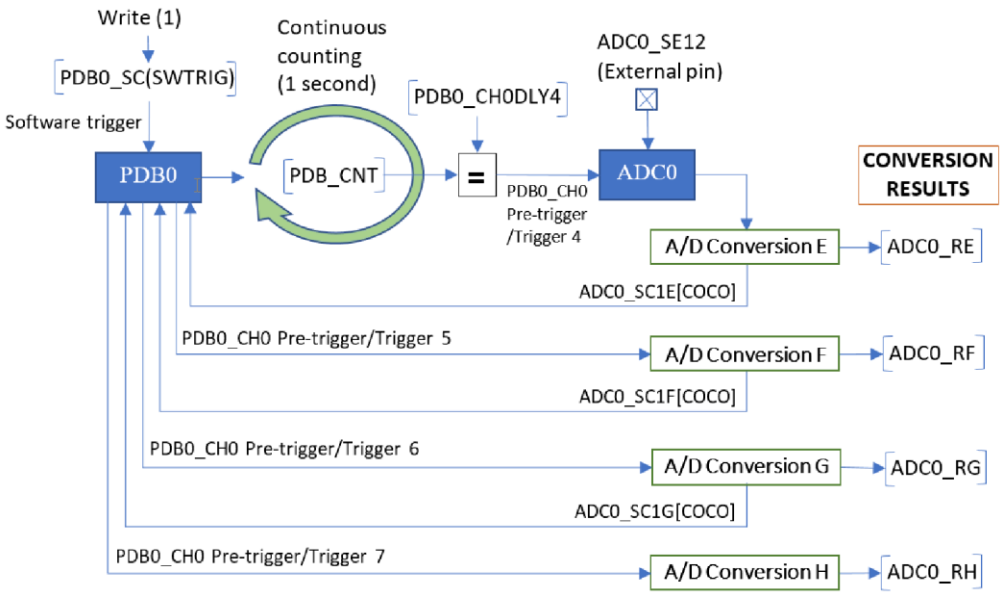

図 12.バックトゥバックモードで PDB トリガを使用する ADC

4.4 TRGMUX トリガ

サンプルコードについては、付録 D を参照してください。

TRGMUX は、周辺モジュールのトリガ入力をさまざまな内部および/または外部トリガ信号(タイマモジュール、アナログモジュールフラグ、外部ピン)に相互接続するための非常に柔軟なモジュールです。具体的には、S32K1xx および S32M24x の ADC では、TRGMUX を使用して、使用可能なトリガ信号のいずれかと変換を同期させることができます。SC1n レジスタ 0～3[レジスタ SC1A、SC1B、SC1C、SC1D]の ADC 変換をトリガするときに TRGMUX メカニズムを使用でき、この種のトリガは常にトリガラッチングガスケットを通過することに注意してください。

この例では、外部チャネル 12 の単一の ADC0 変換(ADC0_SE12)は、外部信号(この場合は信号 TRGMUX_IN0)の各立ち上がりエッジでトリガされます。このユースケースでは、SIM_ADCOPT[ADC0SWPRETRG]レジスタフィールドに書き込むことによって、ソフトウェアプリトリガを ADC0 に提供する必要があります。

次の図は、トリガ(青色の点線)とプリトリガ(赤色の点線)の信号の全体的なパスを示しています。

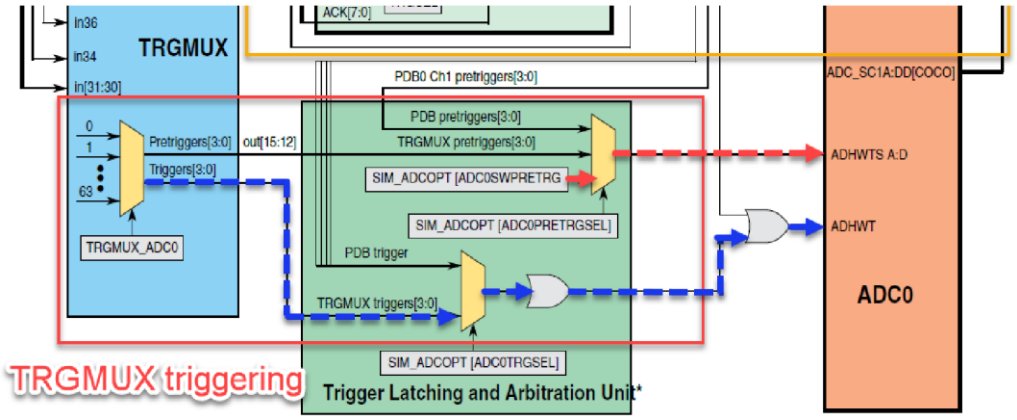

図 13.TRGMUX を介した ADC トリガ

ワークフローの例

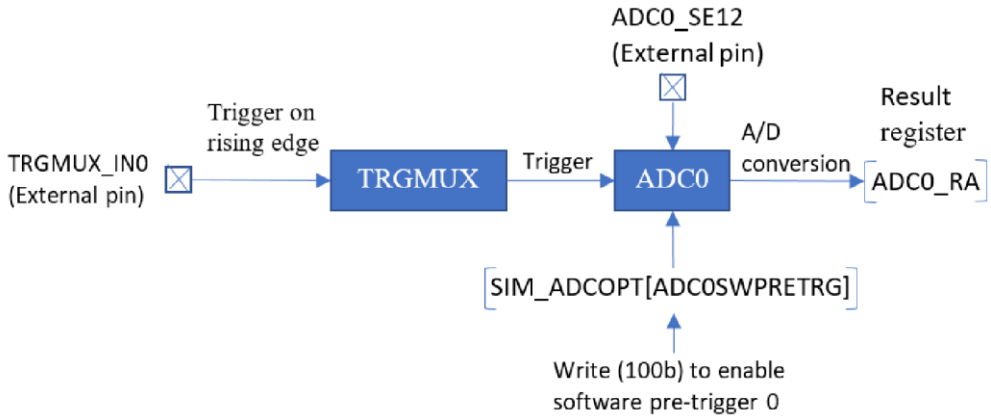

図 14.TRGMUX トリガを使用する ADC のワークフローの例

5 リファレンス

- S32K1xx データシート
- S32M2xx データシート
- S32K1xx リファレンスマニュアル
- S32M24x リファレンスマニュアル
- S32K1xx 向け AN5426 ハードウェア設計ガイドライン
- SAR ADC 測定のための AN4373 クックブック
- ADC 校正ドキュメント

6 付録

6.1 サンプルコード: S32K144 デバイスでの ADC ソフトウェアトリガ

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_RawResult;
```

```
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    IP_WDOG->CNT=0xD928C520;  /* Unlock watchdog */
    IP_WDOG->TOVAL=0x0000FFFF; /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;  /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();  /* Disable Watchdog */

    IP_SCG->FIRCDIV =  SCG_FIRCDIV_FIRCDIV2(4); /* FIRCDIV2 = 4: FIRCDIV2 divide
 by 8 */

    /***** Calibrate ADC0 *****/
    IP_PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK; /* Disable clock to
 change PCS */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3);     /* PCS = 3: Select
 FIRCDIV2 */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK;  /* Enable bus clock in
 ADC */

    IP_ADC0->SC3 = ADC_SC3_CAL_MASK /* CAL = 1: Start calibration sequence */
   | ADC_SC3_AVGE_MASK /* AVGE = 1: Enable hardware average */
   | ADC_SC3_AVGS(3);  /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
    /*************************************************
     * Initialize ADC0:
     * External channel 12, software trigger,
     * single conversion, 12-bit resolution
     *************************************************/
    IP_ADC0->SC1[0] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for
 conversions */

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* ADIV = 0: Divide
 ratio = 1 */
            /* MODE = 1: 12-bit conversion */

    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */

    IP_ADC0->SC2 = ADC_SC2_ADTRG(0); /* ADTRG = 0: SW trigger */

    IP_ADC0->SC3 = 0x00000000; /* ADCO = 0: One conversion performed */
     /* AVGE,AVGS = 0: HW average function disabled */
    for(;;)
    {
        /* Initiate new conversion by writing to ADC0_SC1A(ADCH) */
        IP_ADC0->SC1[0] = ADC_SC1_ADCH(12);   /* ADCH = 12: External channel 12
 as input */

        /* Wait for latest conversion to complete */
        while(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        ADC_RawResult = IP_ADC0->R[0]; /* Read ADC Data Result A (ADC0_RA) */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12);  /* Convert to mV
 (@VREFH = 5V) */
```

```
    }
    return 0;
}
```

6.2 サンプルコード: S32K144 デバイスで PDB トリガを使用する ADC

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    IP_WDOG->CNT=0xD928C520;  /* Unlock watchdog */
    IP_WDOG->TOVAL=0x0000FFFF; /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;  /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();    /* Disable Watchdog */

    IP_SCG->FIRCDIV = SCG_FIRCDIV_FIRCDIV2(4);   /* FIRCDIV2 = 4: FIRCDIV2
 divide by 8 */

    /*********************************************
     * Calibrate ADC0
     *********************************************/
    IP_PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK; /* Disable clock to
 change PCS */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3);      /* PCS = 3: Select
 FIRCDIV2 */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in
 ADC */

    IP_ADC0->SC3 = ADC_SC3_CAL_MASK  /* CAL = 1: Start calibration sequence */
   | ADC_SC3_AVGE_MASK /* AVGE = 1: Enable hardware average */
   | ADC_SC3_AVGS(3);  /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

    /*****************************************************
     * Initialize ADC0:
     * External channel 12, hardware trigger,
     * single conversion, 12-bit resolution
     *
     * NOTE: ADC0->SC1[4] corresponds to ADC0_SC1E register
     *****************************************************/
    IP_ADC0->SC1[4] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for
 conversions */

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* ADIV = 0: Divide
 ratio = 1 */
            /* MODE = 1: 12-bit conversion */

    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */
```

```
    IP_ADC0->SC2 = ADC_SC2_ADTRG(1); /* ADTRG = 1: HW trigger */

    IP_ADC0->SC1[4] = ADC_SC1_ADCH(12); /* ADCH = 12: External channel 12 as
 ADC0 input */

    IP_ADC0->SC3 = 0x00000000;  /* ADCO = 0: One conversion performed */
      /* AVGE,AVGS = 0: HW average function disabled */

    /*********************************************
     * Initialize PDB0:
     * 1 second period, continuous mode
     * PDB0_CH0 pre-trigger 4 output enabled
     *********************************************/

     IP_PCC->PCCn[PCC_PDB0_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in
 PDB */

     IP_PDB0->SC = PDB_SC_PRESCALER(6)/* PRESCALER = 6: clk divided by (64 x
 Mult factor) */
   | PDB_SC_TRGSEL(15)    /* TRGSEL = 15: Software trigger selected */
   | PDB_SC_MULT(3)     /* MULT = 3: Multiplication factor is 40 */
   | PDB_SC_CONT_MASK;    /* CONT = 1: Enable operation in continuous mode */

    /* PDB Period = (System Clock / (Prescaler x Mult factor)) / Modulus */
    /* PDB Period = (48 MHz / (64 x 40)) / 18750 */
    /* PDB Period = (18750 Hz) / (18750) = 1 Hz */
    IP_PDB0->MOD = 18750;

  IP_PDB0->CH[0].C1 = (PDB_C1_TOS(0x10)/* TOS = 10h: Pre-trigger 4 asserts with
 DLY match */
        | PDB_C1_EN(0x10)); /* EN = 10h: Pre-trigger 4 enabled */

    IP_PDB0->CH[0].DLY[4] = 9375; /* Delay set to half the PDB period = 9375 */

    IP_PDB0->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;  /* Enable PDB. Load
 MOD and DLY */

    IP_PDB0->SC |= PDB_SC_SWTRIG_MASK;        /* Single initial PDB trigger */

    for(;;)
    {
        /* Wait for latest conversion to complete */
        while(((IP_ADC0->SC1[4] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

        ADC_RawResult = IP_ADC0->R[4]; /* Read ADC Data Result E (ADC0_RE) */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12); /* Convert to mV
 (@VREFH = 5V) */
    }
    return 0;
}
```

6.3 サンプルコード: S32K144 デバイスで PDB およびバックトゥバックトリガを使用する ADC

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_Results[4];

void WDOG_disable (void)
{
```

```
    IP_WDOG->CNT=0xD928C520;     /* Unlock watchdog */
    IP_WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();   /* Disable Watchdog*/

    IP_SCG->FIRCDIV = SCG_FIRCDIV_FIRCDIV2(4);  /* FIRCDIV2 = 4: FIRCDIV2 divide
 by 8 */

    /*********************************************
     * Calibrate ADC0
     *********************************************/
    IP_PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK;/* Disable clock to
 change PCS */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3);     /* PCS = 3: Select
 FIRCDIV2 */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK;  /* Enable bus clock in
 ADC */

    IP_ADC0->SC3 = ADC_SC3_CAL_MASK  /* CAL = 1: Start calibration sequence */
   | ADC_SC3_AVGE_MASK /* AVGE = 1: Enable hardware average */
   | ADC_SC3_AVGS(3);  /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

    /*********************************************
     * Initialize ADC0:
     * External channel 12, hardware trigger,
     * single conversion, 12-bit resolution
     *
     * NOTE: ADC0->SC1[4] corresponds to ADC0_SC1E register
     *********************************************/
    IP_ADC0->SC1[4] = ADC_SC1_ADCH_MASK;/* ADCH = 1F: Module is disabled for
 conversions*/
      /* AIEN = 0: Interrupts are disabled */
    IP_ADC0->SC1[5] = ADC_SC1_ADCH_MASK;
    IP_ADC0->SC1[6] = ADC_SC1_ADCH_MASK;
    IP_ADC0->SC1[7] = ADC_SC1_ADCH_MASK;

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);  /* ADIV = 0: Divide
 ratio = 1 */
            /* MODE = 1: 12-bit conversion */

    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */

    IP_ADC0->SC2 = ADC_SC2_ADTRG(1); /* ADTRG = 1: HW trigger */

    IP_ADC0->SC1[4] = ADC_SC1_ADCH(12); /* SC1E[ADCH] = 12: External channel 12
 as input */
    IP_ADC0->SC1[5] = ADC_SC1_ADCH(12); /* SC1F[ADCH] = 12: External channel 12
 as input */
    IP_ADC0->SC1[6] = ADC_SC1_ADCH(12); /* SC1G[ADCH] = 12: External channel 12
 as input */
    IP_ADC0->SC1[7] = ADC_SC1_ADCH(12); /* SC1H[ADCH] = 12: External channel 12
 as input */
```

```
    IP_ADC0->SC3 = 0x00000000;  /* ADCO = 0: One conversion performed */
      /* AVGE,AVGS = 0: HW average function disabled */

    /*********************************************
     * Initialize PDB0:
     * 1 second period, continuous mode
     * PDB0_CH0 pre-trigger outputs 4/5/6/7 enabled
     * Pre-trigger 4 asserted by channel delay register match
     * Back to back mode enabled for pre-triggers 5/6/7
     *********************************************/

    IP_PCC->PCCn[PCC_PDB0_INDEX] |= PCC_PCCn_CGC_MASK;   /* Enable bus clock in
 PDB */

    IP_PDB0->SC = PDB_SC_PRESCALER(6) /* PRESCALER = 6: clk divided by (64 x
 Mult factor) */
  | PDB_SC_TRGSEL(15)  /* TRGSEL = 15: Software trigger selected */
  | PDB_SC_MULT(3)  /* MULT = 3: Multiplication factor is 40 */
  | PDB_SC_CONT_MASK;  /* CONT = 1: Enable operation in continuous mode */

    /* PDB Period = (System Clock / (Prescaler x Mult factor)) / Modulus */
    /* PDB Period = (48 MHz / (64 x 40)) / 18750 */
    /* PDB Period = (18750 Hz) / (18750) = 1 Hz */
    IP_PDB0->MOD = 18750;

    IP_PDB0->CH[0].C1 = (PDB_C1_BB(0xE0)/* BB = E0h: Back-to-back for pre-
triggers 5/6/7 */
        | PDB_C1_TOS(0x10) /* TOS = 10h: Pre-trigger 4 asserts with DLY match */
        | PDB_C1_EN(0xF0)); /* EN = F0h: Pre-triggers 4/5/6/7 enabled */

    IP_PDB0->CH[0].DLY[4] = 9375; /* Delay set to half the PDB period = 9375 */

    IP_PDB0->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;  /* Enable PDB. Load
 MOD and DLY */

    IP_PDB0->SC |= PDB_SC_SWTRIG_MASK;       /* Single initial PDB trigger */

    for(;;)
    {
        /* Wait for last conversion in the sequence to complete (ADC0_SC1H) */
        while(((IP_ADC0->SC1[7] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

        ADC_Results[0] = IP_ADC0->R[4]; /* Read ADC Data Results 4-7 (ADC0_RE to
 ADC0_H) */
        ADC_Results[1] = IP_ADC0->R[5];
        ADC_Results[2] = IP_ADC0->R[6];
        ADC_Results[3] = IP_ADC0->R[7];
    }

    return 0;
}
```

6.4 サンプルコード: S32K144 デバイスで TRGMUX トリガを使用する ADC

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;
```

```
void WDOG_disable (void)
{
    IP_WDOG->CNT=0xD928C520;     /* Unlock watchdog */
    IP_WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();   /*!Disable Watchdog*/

    IP_SCG->FIRCDIV =  SCG_FIRCDIV_FIRCDIV2(4); /* FIRCDIV2 = 4: FIRCDIV2 divide
 by 8 */
    /*********************************************
     * Configure pin PTB5 as TRGMUX_IN0
     *********************************************/
    IP_PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK;/* Enable clock gate for
 PORTB */
    IP_PORTB->PCR[5] = PORT_PCR_MUX(6);   /* Mux = 6: PTB5 as TRGMUX_IN0 */

    /* Select TRGMUX_IN0 as ADC0 Trigger Mux input source 0 */
    IP_TRGMUX->TRGMUXn[TRGMUX_ADC0_INDEX] = TRGMUX_TRGMUXn_SEL0(2U);

    /*********************************************
     * Calibrate ADC0
     *********************************************/
    IP_PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK;/* Disable clock to
 change PCS */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3); /* PCS = 3: Select FIRCDIV2
 */
    IP_PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in
 ADC */

    IP_ADC0->SC3 = ADC_SC3_CAL_MASK  /* CAL = 1: Start calibration sequence */
   | ADC_SC3_AVGE_MASK /* AVGE = 1: Enable hardware average */
   | ADC_SC3_AVGS(3);  /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

    /*********************************************
     * Initialize ADC0:
     * External channel 12, hardware trigger,
     * single conversion, 12-bit resolution
     *********************************************/
    IP_ADC0->SC1[0] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for
 conversions */

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);/* ADIV = 0: Divide
 ratio = 1 */
           /* MODE = 1: 12-bit conversion */
    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */

    IP_ADC0->SC2 = ADC_SC2_ADTRG(1); /* ADTRG = 1: HW trigger */

    IP_ADC0->SC1[0] = ADC_SC1_ADCH(12); /* ADCH = 12: External channel 12 as
 input */
```

```
    IP_ADC0->SC3 = 0x00000000;  /* ADCO = 0: One conversion performed */
      /* AVGE,AVGS = 0: HW average function disabled */

    /*********************************************
     * SIM Configurations for ADC triggering:
     * Pre-trigger source: Software pre-trigger
     * Trigger select: TRGMUX output
     *********************************************/
 IP_SIM->ADCOPT = SIM_ADCOPT_ADC0PRETRGSEL(2)/* ADC0PRETRGSEL = 10b: Software
 pretrigger */
     | SIM_ADCOPT_ADC0SWPRETRG(4) /* ADC0SWPRETRG = 100b: SW Pre-trigger 0 */
     | SIM_ADCOPT_ADC0TRGSEL(1); /* ADC0TRGSEL = 1: TRGMUX output as trigger */

    for(;;)
    {
        /* Wait for latest conversion to complete */
        while(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        ADC_RawResult = IP_ADC0->R[0];  /* Read ADC Data Result 0 */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12);  /* Convert to mV
 (@VREFH = 5V) */
 }
 return 0;
}
```

6.5 サンプルコード: S32M244 デバイスでの ADC ソフトウェアトリガ

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    IP_WDOG->CNT=0xD928C520;  /* Unlock watchdog */
    IP_WDOG->TOVAL=0x0000FFFF;  /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;  /* Disable watchdog */
}

int main(void) {

 WDOG_disable();  /* Disable Watchdog */

 IP_SCG->FIRCDIV =  SCG_FIRCDIV_FIRCDIV2(4); /* FIRCDIV2 = 4: FIRCDIV2 divide by
 8 */

 /***** Calibrate ADC1 *****/
 IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;     /* Disable clock to
 change PCS */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);     /* PCS = 3: Select
 FIRCDIV2 */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK;     /* Enable bus clock in
 ADC */

 IP_ADC1->SC3 = ADC_SC3_CAL_MASK  /* CAL = 1: Start calibration sequence */
     | ADC_SC3_AVGE_MASK /* AVGE = 1: Enable hardware average */
     | ADC_SC3_AVGS(3);  /* AVGS = 11b: 32 samples averaged */

 /* Wait for completion */
 while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
```

```
 /*****************************************************
  * Initialize ADC1:
  * External channel 11, software trigger,
  * single conversion, 12-bit resolution
  *****************************************************/
 IP_ADC1->SC1[0] = ADC_SC1_ADCH_MASK;  /* ADCH: Module disabled for conversions
 */

 IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);    /* ADIV = 0: Divide
 ratio = 1 */
              /* MODE = 1: 12-bit conversion */

 IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC clks
 */

 IP_ADC1->SC2 = ADC_SC2_ADTRG(0);  /* ADTRG = 0: SW trigger */

 IP_ADC1->SC3 = 0x00000000; /* ADCO = 0: One conversion performed */
     /* AVGE,AVGS = 0: HW average function disabled */

 for(;;)
 {
  /* Initiate new conversion by writing to ADC1_SC1A(ADCH) */
  IP_ADC1->SC1[0] = ADC_SC1_ADCH(11);  /* ADCH = 11: External channel 11 as
 input */

  /* Wait for latest conversion to complete */
  while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
  ADC_RawResult = IP_ADC1->R[0]; /* Read ADC Data Result A (ADC1_RA) */
  ADC_mVResult = (ADC_RawResult * 5000) >> 12;
         /* Convert to mV (@VREFH = 5V) */
 }
 return 0;
}
```

6.6 サンプルコード: S32M244 デバイスで PDB トリガを使用する ADC

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    IP_WDOG->CNT = 0xD928C520;  /* Unlock watchdog */
    IP_WDOG->TOVAL = 0x0000FFFF; /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;  /* Disable watchdog */
}

int main(void) {

 WDOG_disable();    /* Disable Watchdog */

 IP_SCG->FIRCDIV = SCG_FIRCDIV_FIRCDIV2(4);/* FIRCDIV2 = 4: FIRCDIV2 divide by 8
 */

 /*********************************************
  * Calibrate ADC1
  *********************************************/
```

```
  IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;
                                        /* Disable clock to change PCS */
  IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);
                                        /* PCS = 3: Select FIRCDIV2 */
  IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK;
                                        /* Enable bus clock in ADC */

  IP_ADC1->SC3 = ADC_SC3_CAL_MASK   /* CAL = 1: Start calibration sequence */
      | ADC_SC3_AVGE_MASK      /* AVGE = 1: Enable hardware average */
      | ADC_SC3_AVGS(3);       /* AVGS = 11b: 32 samples averaged */

  /* Wait for completion */
  while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        /*************************************************
   * Initialize ADC1:
   * External channel 11, hardware trigger,
   * single conversion, 12-bit resolution
   *
   * NOTE: ADC1->SC1[0] corresponds to ADC0_SC1A register
   *************************************************/
  IP_ADC1->SC1[0] = ADC_SC1_ADCH_MASK;
                                        /* ADCH: Module disabled for conversions
  */

  IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);
                                                        /* ADIV = 0: Divide
  ratio = 1 */
                /* MODE = 1: 12-bit conversion */

  IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12);
                                        /* SMPLTS = 12: sample time is 13 ADC
  clks */

  IP_ADC1->SC2 = ADC_SC2_ADTRG(1);  /* ADTRG = 1: HW trigger */

  IP_ADC1->SC1[0] = ADC_SC1_ADCH(11);
 /* ADCH = 11: External channel 11 as ADC1 input */

  IP_ADC1->SC3 = 0x00000000;  /* ADCO = 0: One conversion performed */
       /* AVGE,AVGS = 0: HW average function disabled */
     /*********************************************
      * Initialize PDB1:
      * 1 second period, continuous mode
      * PDB1_CH0 pre-trigger 1 output enabled
      *********************************************/

  IP_PCC->PCCn[PCC_PDB1_INDEX] |= PCC_PCCn_CGC_MASK;
                                                      /* Enable bus clock in PDB
  */

  IP_PDB1->SC = PDB_SC_PRESCALER(6)  /* PRESCALER = 6 */
      | PDB_SC_TRGSEL(15)   /* Software trigger selected */
      | PDB_SC_MULT(3)    /* Multiplication factor is 40 */
      | PDB_SC_CONT_MASK;  /* Enable operation in continuous mode */

  /* PDB Period = (System Clock / (Prescaler x Mult factor)) / Modulus */
  /* PDB Period = (48 MHz / (64 x 40)) / 18750 */
  /* PDB Period = (18750 Hz) / (18750) = 1 Hz */
  IP_PDB1->MOD = 18750;
```

```
 IP_PDB1->CH[0].C1 = (PDB_C1_TOS(0x01)  /* Pre-trigger 1 asserts with DLY match
 */
           | PDB_C1_EN(0x01)); /* Pre-trigger 1 enabled */

       IP_PDB1->CH[0].DLY[1] = 9375; /* Delay set to half the PDB period = 9375
 */

 IP_PDB1->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;
                                                    /* Enable PDB. Load MOD and
 DLY */

 IP_PDB1->SC |= PDB_SC_SWTRIG_MASK;  /* Single initial PDB trigger */

 for(;;)
 {
  /* Wait for latest conversion to complete */
  while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

  ADC_RawResult = IP_ADC1->R[0]; /* Read ADC Data Result E (ADC1_RA) */
  ADC_mVResult = (ADC_RawResult * 5000) >> 12;
                                                  /* Convert to mV (@VREFH = 5V)
 */
 }

    return 0;
}
```

6.7 サンプルコード: S32M244 デバイスで PDB およびバックトゥバックトリガを使用する ADC

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    IP_WDOG->CNT = 0xD928C520;  /* Unlock watchdog */
    IP_WDOG->TOVAL = 0x0000FFFF; /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;  /* Disable watchdog */
}

int main(void) {

 WDOG_disable();    /* Disable Watchdog */

 IP_SCG->FIRCDIV = SCG_FIRCDIV_FIRCDIV2(4);/* FIRCDIV2 = 4: FIRCDIV2 divide by 8
 */

 /**********************************************
  * Calibrate ADC1
  **********************************************/
 IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;
/* Disable clock to change PCS */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);
/* PCS = 3: Select FIRCDIV2 */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK;
/* Enable bus clock in ADC */

 IP_ADC1->SC3 = ADC_SC3_CAL_MASK  /* CAL = 1: Start calibration sequence */
```

```
    | ADC_SC3_AVGE_MASK  /* AVGE = 1: Enable hardware average */
    | ADC_SC3_AVGS(3);   /* AVGS = 11b: 32 samples averaged */

/* Wait for completion */
while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
     /*********************************************
 * Initialize ADC1:
 * External channel 12, hardware trigger,
 * single conversion, 12-bit resolution
 *
 * NOTE: ADC1->SC1[4] corresponds to ADC0_SC1E register
 *********************************************/
IP_ADC1->SC1[4] = ADC_SC1_ADCH_MASK;
                                /* ADCH = 1F: Module is disabled for
conversions*/
    /* AIEN = 0: Interrupts are disabled */
IP_ADC1->SC1[5] = ADC_SC1_ADCH_MASK;
IP_ADC1->SC1[6] = ADC_SC1_ADCH_MASK;
IP_ADC1->SC1[7] = ADC_SC1_ADCH_MASK;

IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);
                                                    /* ADIV = 0: Divide ratio =
1 */

            /* MODE = 1: 12-bit conversion */

IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12);
                                          /* SMPLTS = 12: sample time is 13 ADC
clks */

IP_ADC1->SC2 = ADC_SC2_ADTRG(1);
                                     /* ADTRG = 1: HW trigger */

IP_ADC1->SC1[4] = ADC_SC1_ADCH(11);
                                /* SC1E[ADCH] = 11: External channel 11 as input
*/
IP_ADC1->SC1[5] = ADC_SC1_ADCH(11);
                                /* SC1F[ADCH] = 11: External channel 11 as input
*/
IP_ADC1->SC1[6] = ADC_SC1_ADCH(11);
                                /* SC1G[ADCH] = 11: External channel 11 as input
*/
IP_ADC1->SC1[7] = ADC_SC1_ADCH(11);
                                /* SC1H[ADCH] = 11: External channel 11 as input
*/

IP_ADC1->SC3 = 0x00000000;  /* ADCO = 0: One conversion performed */
     /* AVGE,AVGS = 0: HW average function disabled */

      /*********************************************
 * Initialize PDB1:
 * 1 second period, continuous mode
 * PDB1_CH0 pre-trigger outputs 4/5/6/7 enabled
 * Pre-trigger 4 asserted by channel delay register match
 * Back to back mode enabled for pre-triggers 5/6/7
 *********************************************/

IP_PCC->PCCn[PCC_PDB1_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in PDB
*/
```

```
 IP_PDB1->SC = PDB_SC_PRESCALER(6)   /* clk divided by (64 x Mult factor) */
    | PDB_SC_TRGSEL(15)  /* Software trigger selected */
    | PDB_SC_MULT(3)  /* Multiplication factor is 40 */
    | PDB_SC_CONT_MASK; /* Enable operation in continuous mode */

 /* PDB Period = (System Clock / (Prescaler x Mult factor)) / Modulus */
 /* PDB Period = (48 MHz / (64 x 40)) / 18750 */
 /* PDB Period = (18750 Hz) / (18750) = 1 Hz */
 IP_PDB1->MOD = 18750;

 IP_PDB1->CH[0].C1 = (PDB_C1_BB(0xE0) /* Back-to-back for pre-triggers 5/6/7 */
       | PDB_C1_TOS(0x10)
                                              /* Pre-trigger 4 asserts with DLY
 match */
       | PDB_C1_EN(0xF0)); /* Pre-triggers 4/5/6/7 enabled */

 IP_PDB1->CH[0].DLY[4] = 9375; /* Delay set to half the PDB period = 9375 */

 IP_PDB1->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;
                                                /* Enable PDB. Load MOD and DLY
 */

 IP_PDB1->SC |= PDB_SC_SWTRIG_MASK; /* Single initial PDB trigger */

       for(;;)
 {
  /* Wait for last conversion in the sequence to complete (ADC1_SC1H) */
  while(((IP_ADC1->SC1[7] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

  /* Read ADC Data Results 4-7 (ADC1_RE to ADC1_RH) */
  ADC_Results[0] = IP_ADC1->R[4];
  ADC_Results[1] = IP_ADC1->R[5];
  ADC_Results[2] = IP_ADC1->R[6];
  ADC_Results[3] = IP_ADC1->R[7];
 }

 return 0;
}
```

6.8 サンプルコード: S32M244 デバイスで TRGMUX トリガを使用する ADC

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
 IP_WDOG->CNT=0xD928C520;     /* Unlock watchdog */
 IP_WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
 IP_WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void) {
 WDOG_disable();    /*!Disable Watchdog*/

 IP_SCG->FIRCDIV =  SCG_FIRCDIV_FIRCDIV2(4);  /* FIRCDIV2 divide by 8 */

 /*********************************************
```

```
   * Configure pin PTB5 as TRGMUX_IN0
   ********************************************/
  IP_PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK;
 /* Enable clock gate for PORTB */
  IP_PORTB->PCR[5] = PORT_PCR_MUX(6); /* PTB5 as TRGMUX_IN0 */

  /* Select TRGMUX_IN0 as ADC1 Trigger Mux input source 0 */
  IP_TRGMUX->TRGMUXn[TRGMUX_ADC1_INDEX] = TRGMUX_TRGMUXn_SEL0(2U);

  /*********************************************
   * Calibrate ADC1
   ********************************************/
  IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;
                                                    /* Disable clock to
  change PCS */
  IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);    /* PCS = 3: Select FIRCDIV2
  */
  IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in ADC
  */
     IP_ADC1->SC3 = ADC_SC3_CAL_MASK  /* Start calibration sequence */
      | ADC_SC3_AVGE_MASK /* Enable hardware average */
      | ADC_SC3_AVGS(3);  /* 32 samples averaged */

  /* Wait for completion */
  while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

  /*********************************************
   * Initialize ADC1:
   * External channel 11, hardware trigger,
   * single conversion, 12-bit resolution
   ********************************************/
  IP_ADC1->SC1[0] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for conversions
  */

  IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* Divide ratio = 1 */
                                  /* MODE = 1: 12-bit conversion */

  IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12); /* Sample time is 13 ADC clks */

  IP_ADC1->SC2 = ADC_SC2_ADTRG(1);  /* HW trigger */

  IP_ADC1->SC1[0] = ADC_SC1_ADCH(11);  /* External channel 11 as input */

  IP_ADC1->SC3 = 0x00000000;  /* One conversion performed */
       /* AVGE,AVGS = 0: HW average function disabled */
  /*********************************************
   * SIM Configurations for ADC triggering:
   * Pre-trigger source: Software pre-trigger
   * Trigger select: TRGMUX output
   ********************************************/
  IP_SIM->ADCOPT = SIM_ADCOPT_ADC1PRETRGSEL(2)      /* Software pretrigger */
        | SIM_ADCOPT_ADC1SWPRETRG(4)  /* SW Pre-trigger 0 */
        | SIM_ADCOPT_ADC1TRGSEL(1); /* TRGMUX output as trigger */

  for(;;)
  {
   /* Wait for latest conversion to complete */
   while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
   ADC_RawResult = IP_ADC1->R[0];     /* Read ADC Data Result A */
   ADC_mVResult = (ADC_RawResult * 5000) >> 12;
```

```
    /* Convert to mV (@VREFH = 5V) */
  }
  return 0;
}
```

7 改定履歴

表 1.改定履歴

ドキュメント ID	リリース日	説明
AN12217 v. 2	2024 年 9 月 5 日	S32M24X デバイスに関する情報が追加されました。
AN12217 v. 1	2020 年 1 月	セクション 6.2 のサンプルコードが更新されました。
AN12217 v. 0	2018 年 8 月	初版リリース

8 ドキュメント内のソースコードに関する注意

本ドキュメントに記載されているサンプルコードには、次の著作権と BSD-3-Clause ライセンスがあります。

ソースおよびバイナリ形式での Copyright 2024　NXP の再配布および使用は、変更の有無にかかわらず、次の条件が満たされている場合に許可されます。

1. ソースコードの再配布では、上記の著作権表示、この条件リスト、および次の免責事項を保持する必要があります。
2. バイナリ形式での再配布では、上記の著作権表示、この条件リスト、および次の免責事項をドキュメントおよび/またはその他の資料に複製する必要があります。
3. 著作権者の名前またはその貢献者の名前は、事前の書面による特別な許可なしに、このソフトウェアから派生した製品を推奨または宣伝するために使用することはできません。

本ソフトウェアは、著作権者および寄稿者によって「現状のまま」提供されるものであり、商品性および特定目的への適合性の黙示保証を含む(ただしこれに限定されない)、明示または黙示のいかなる保証も否認されます。いかなる場合も、著作権者または寄稿者は、いかなる直接的、間接的、偶発的、特殊的、典型的、または結果的な損害(代替の商品またはサービスの調達、使用、データ、または利益の損失、または業務の中断を含むが、これらに限定されない)に対しても、契約、厳格責任、または不法行為(過失またはその他を含む)のいずれであるかを問わず、いかなる責任の理論に基づいても、かかる損害の可能性について知らされていたとしても、本ソフトウェアの使用から生じる一切の損害に対して責任を負いません。

法務情報

定義

ドラフト— ドキュメントのドラフトステータスは、内容がまだ内部レビュー中であり、正式な承認の対象であることを示しており、その結果、修正または追加が行われる可能性があります。NXP Semiconductors は、ドキュメントのドラフト版に含まれる情報の正確性または完全性について、いかなる表明または保証も行わず、当該情報の使用の結果について責任を負わないものとします。

免責事項

保証および責任の制限— 本書に記載されている情報は、正確かつ信頼できるものと信じます。ただし、NXP Semiconductors は、当該情報の正確性または完全性について、明示または黙示を問わず、いかなる表示または保証も行わないものとし、当該情報の使用の結果について責任を負わないものとします。NXP Semiconductors は、NXP Semiconductors 以外の情報源から提供された場合、本書の内容について一切の責任を負いません。

いかなる場合も、NXP Semiconductors は、間接損害、付随的損害、懲罰的損害、特別損害、または結果的損害(逸失利益、逸失貯蓄、業務の中断、製品の取り外しまたは交換に関連する費用、または再加工費用を含むがこれらに限定されない)について、それらの損害が不法行為(怠慢を含む)、保証、契約違反、またはその他の法理論に基づくかどうかにかかわらず、一切責任を負いません。

理由のいかんを問わず、お客様が損害を被った場合でも、本ドキュメントに記載されている製品に関する NXP Semiconductors のお客様に対する責任の総額および累積額は、NXP Semiconductors の商業販売契約条件に従って制限されます。

変更を行う権利— NXP Semiconductors は、仕様および製品の説明を含むがこれに限定されない本ドキュメントに掲載された情報を、いつでも予告なしに変更する権利を留保します。本ドキュメントは、本ドキュメントの発行前に提供されたすべての情報に優先し、それに代わるものです。

アプリケーション—これらの製品についてここで説明するアプリケーションは、説明のみを目的としています。NXP Semiconductors は、かかるアプリケーションが追加のテストまたは修正なしに指定された使用に適していることを表明または保証しません。

お客様は、NXP Semiconductors 製品を使用するアプリケーションおよび製品の設計と操作について責任を負い、NXP Semiconductors は、アプリケーションまたはお客様の製品の設計に関するいかなる支援についても責任を負いません。NXP Semiconductors 製品がお客様のアプリケーションおよび計画された製品、ならびにお客様のサードパーティのお客様の計画されたアプリケーションおよび使用に適しているかどうかを判断することは、お客様の単独の責任となります。お客様は、アプリケーションおよび製品に関連するリスクを最小限に抑えるために、適切な設計および運用上の保護措置を提供する必要があります。

NXP Semiconductors は、お客様のアプリケーションもしくは製品、またはお客様のサードパーティのお客様によるアプリケーションもしくは使用における脆弱性または不履行に基づく不履行、損害、費用または問題に関して、いかなる責任も負いません。お客様は、お客様のアプリケーションおよび製品、またはお客様のサードパーティのお客様によるアプリケーションまたは使用の不履行を回避するために、NXP Semiconductors 製品を使用するお客様のアプリケーションおよび製品に必要なすべてのテストを実施する責任を負います。NXP はこの点に関していかなる責任も負いません。

商業販売契約条件— NXP Semiconductors 製品は、有効な書面による個別契約で別段の合意がある場合を除き、https://www.nxp.com/profile/terms に掲載されている商業販売の一般契約条件に従って販売されます。個別契約を締結する場合には、当該契約の条件のみを適用するものとします。NXP Semiconductors は、お客様による NXP Semiconductors 製品の購入に関して、お客様の一般契約条件を適用することに明示的に反対します。

自動車アプリケーションでの使用適性— 本 NXP 製品は、自動車アプリケーションでの使用が認定されています。本製品が、(a)安全上重要な用途に使用される、または(b)故障により死亡、人身傷害、または重大な物理的もしくは環境的損傷につながる可能性がある製品またはサービス(以下、「重要アプリケーション」といいます)の開発または組み込みにおいて、お客様によって使用される場合、お客様は、その製品に関する最終的な設計上の決定を行い、NXP によって提供される情報やサポートにかかわらず、その製品に関するすべての法律、規制、安全、およびセキュリティ関連の要件を遵守する責任を単独で負います。したがって、お客様は、重要アプリケーションおよび NXP における製品の使用に関連するすべてのリスクを負うものとし、そのサプライヤは、お客様によるかかる使用に対して責任を負わないものとします。これにより、お客様は、お客様による重要アプリケーションへの製品の組み込みに関連して NXP が被る可能性のある請求、責任、損害、および関連する費用(弁護士費用を含む)について、NXP を補償し、損害を与えないものとします。

輸出規制— 本ドキュメントおよび本ドキュメントに記載されている項目は、輸出規制の対象となる場合があります。輸出については、権限のある当局の事前の許可を必要とすることがあります。

翻訳— 英語以外のドキュメント(翻訳版)は、そのドキュメント内の法的情報を含め、参照のみを目的としています。翻訳版と英語版との間に相違がある場合には、英語版が優先するものとします。

セキュリティ— お客様は、すべての NXP 製品が、特定されていない脆弱性の影響を受ける可能性があること、または明らかな制限のある確立されたセキュリティ規格または仕様をサポートする場合があることを理解するものとします。お客様は、お客様のアプリケーションおよび製品に対するこれらの脆弱性の影響を軽減するために、その耐用期間を通じてアプリケーションおよび製品の設計および運用に責任を負います。お客様の責任は、お客様のアプリケーションで使用するために NXP 製品でサポートされているその他のオープン/独自技術にも及びます。NXP はいかなる脆弱性に対しても責任を負いません。お客様は、NXP からのセキュリティアップデートを定期的に確認し、適切に追従する必要があります。お客様は、目的とするアプリケーションの規則、規制、および標準に最も適したセキュリティ機能を備えた製品を選択し、製品に関する最終的な設計上の決定を行うものとします。また、お客様は、NXP によって提供される情報やサポートにかかわらず、製品に関するすべての法律、規制、およびセキュリティ関連の要件を遵守する責任を単独で負います。

NXP には、製品セキュリティインシデントチーム(PSIRT) (PSIRT@nxp.com)があり、NXP 製品のセキュリティ脆弱性の調査、報告、およびソリューションリリースを管理しています。

NXP B.V. — NXP B.V. は事業会社ではなく、製品の販売は行っておりません。

商標

注意: 記載されているすべてのブランド、製品名、サービス名、および商標は、それぞれの所有者に帰属します。

NXP — ワードマークおよびロゴは、NXP B.V.の商標です。

AMBA、Arm、Arm7、Arm7TDMI、Arm9、Arm11、Artisan、big.LITTLE、Cordio、CoreLink、CoreSight、Cortex、DesignStart、DynamIQ、Jazelle、Keil、Mali、Mbed、Mbed Enabled、NEON、POP、RealView、SecurCore、Socrates、Thumb、TrustZone、ULINK、ULINK2、ULINK-ME、ULINK-PLUS、ULINKpro、μVision、Versatile は、米国およびその他の国における Arm Limited (またはその子会社もしくは関連会社)の商標および/または登録商標です。関連技術は、特許、著作権、意匠および営業秘密の一部または全部によって保護される場合があります。
無断転載はご遠慮ください。

内容

ドキュメントフィードバック
リリース日: 2024 年 9 月 5 日
ドキュメント識別子: AN12217