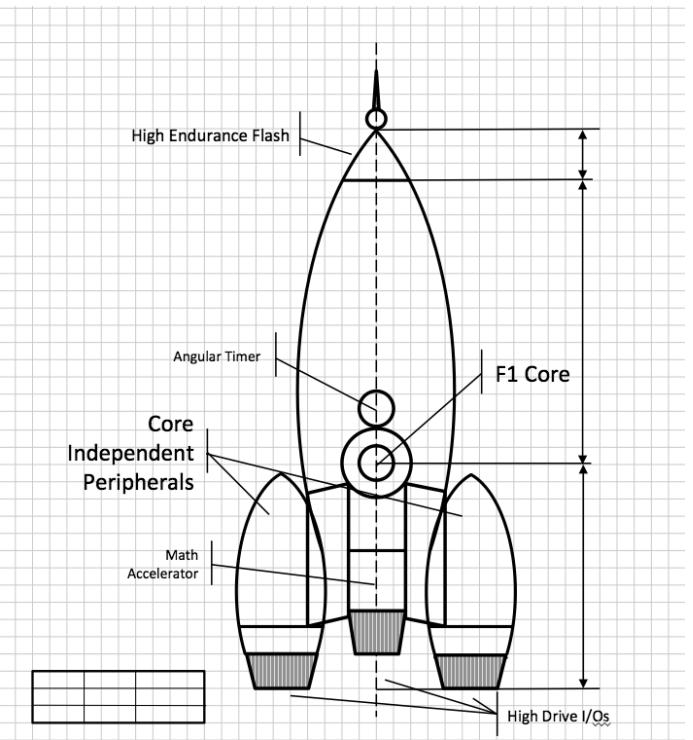
意外に簡単!

― 簡単に使えて組み込み制御の常識を塗り替える

新世代 CIP(コアから独立した周辺モジュール) ―



Lucio Di Jasio

ウェブサイト: http://www.flyingpic24.com

メール: pilot@flyingpic24.com

本書を出版するにあたり、ご協力頂きました Lulu Enterprises, Inc.に感謝を申し上げます。
http://www.lulu.com

ISBN: 978-1-63277-950-2

出版国: アメリカ合衆国

初版: 2015 年

Dieter に捧ぐ

目次

謝辞

著者の最大限の努力にもかかわらず、本書には当初の計画をはるかに上回るページ数と時間を割く結果となりました。妻 Sara からの十二分な支援と理解が得られなければ、本書の完成は不可能であった事をここに申し上げます。本書の技術的な内容を吟味し、数多くの有益な提案をしてくれた Greg Robinson 氏、Sean Steedman 氏、Cobus van Eeden 氏に深く感謝いたします。

また、著者の全ての友人、Microchip 社の同僚、数年にわたって協力させて頂いてきた多数の組み込み制御エンジニアの方々に対して、著者の仕事への多大なる影響と、組み込み制御という広範な分野で経験を積ませて頂いた事について、御礼を述べたいと思います。

最後に、読者の皆様全員、特にアイディア、タイプミス、バグについて報告してくださった皆様、提案を求められた皆様にも感謝の意を表します。この新しいプロジェクトに著者が着手したのは、「皆様の責任」でもあります。引き続き、メールでのご質問やご報告をお待ち申し上げます。

はじめに

著者が組み込みCプログラミングに関する最初の書籍を執筆し始めてから10年が過ぎました。当初は16ビット アーキテクチャに焦点を合わせていましたが、やがて32ビット アーキテクチャに焦点を合わせるようになりました。当時は、この業界の大半の方々と同様、著者は8ビット アプリケーションは緩やかながら確実に衰退してくだろうと予測していました。ところが衰退するどころか、8ビットPIC®マイクロコントローラはむしろ着実に成長を続け、出荷数は1年あたり10億個(ゼロが9個)という指標を大幅に上回ってもなお、成長を続けています。

非常に意外でしたが、おそらくは新世代のローエンド32ビット マイクロコントローラによる競争上の圧力から、最も大々的かつ刺激的な革新が起きたのは、8ビットPICアーキテクチャにおいてでした。初めに **PIC16F1** コアが発展し、続いて**コアから独立した周辺モジュール**が段階的に導入されました。

重要なのはコアではない

C言語は、過去10年で組み込み制御アプリケーションへの使用に拍車がかかり、現在では全アプリケーションの標準となっています。その間に、アセンブリ プログラミングはもっぱら少数の特殊なアプリケーションで使用され、多くはアプリケーション内でも小さな部分で使用されるようになりました。
いわゆる**エンハンスト ミッドレンジ コア**、もっと簡単に言うとPIC16**F1 コア**(全ての搭載製品で1桁目に使われているので簡単に識別可能)が開発されたのは、Microchip社の8ビット アーキテクチャ チームの当然の反応からでした。**元の** PIC®コアは、ハンドコードしたアセンブリで最も効率的に使えるように最適化設計されていましたが、コンパイラ設計者にとっては難易度が高い事で知られていました。8ビットPICのアーキテクトはすぐに、いくつかの命令に絞り込んで(合計12命令)、標準的なCコンパイラ構成の**コード密度**を高めました。また、最も賢明な対応として、メモリへの新しい**リニア アクセス** モードを提供する事で、コンパイラ設計者が懸念する最大の要因であったRAMのバンク切り換えを解消しました。同時に、コンテクスト自動保存機能付き割り込み機能の導入に加えて、さらにはハードウェア スタックを2倍にして、割り込み処理と(通常は複雑な)ネスト関数の機敏性を向上させました。
新しいコアの機能拡張は非常にスムーズに導入されたため、ほとんどの「レガシー」ユーザはコアが移植された事にほとんど気付きませんでした。実装では、

数十年分の大半がアセンブリのアプリケーションに対して、完全な下位互換性が提供されました。また、新規アプリケーションでは、C コンパイラが下層のアーキテクチャ情報を抽象化するため、ユーザ エクスペリエンスには、速度の向上、割り込み応答機能、スタックレベルの深さの増加、コードの簡潔さという影響しか残りません。

コアから独立した周辺モジュール

コアから独立した周辺モジュール(CIP)が導入された事で、最近の革新は全て 8 ビット PIC マイクロコントローラの**周辺モジュール**を中心としたものになりました。CIP の導入は PIC マイクロコントローラを使った設計に大きな変化をもたらし、また、最も重要な点として、同じアプリケーション領域を巡って競合しているローエンド 32 ビット MCU のアプローチとの差別化方法も大きく変えています。ローエンド 32 ビット MCU がこれまでになくソフトウェアを重視している一方で(重視すればするほど複雑さが増し、高いクロック速度と高い演算能力が必要になる)、CIP は次のように正反対のアプローチを取っています。直接相互接続された自律的なハードウェア周辺機能ブロックに焦点を合わせる事でクロック速度、消費電力、ソフトウェアの複雑さを低く抑え、応答時間を短縮しながら、8 ビットアプリケーションの性能を**向上**させます。

(組み込み制御の)禅へ向かう 3 つの段階

東洋的な修練で悟りを開くのと同様に、**理解**とは内面の段階的な成長を意味します。コアからの独立という**考え方**を真に理解するには、少なくとも以下の 3 つの段階が必要です。

1. **発見**段階は、この(組み込み)領域が持つ課題への新しい対処方法が、弟子に対して初めて紹介された時の事です。本書の大部分は、新しい CIP と既存コアの機能拡張の紹介に費やされています。新しい CIP ごとに、対象用途と、特定のコアの作業負荷を削減するために考えられる応用方法を説明します。
2. **精通**段階では、弟子が新しい CIP の設計の背後にある考え方をより深く理解し始め、LEGO®ブロックのように相互に接続する事で新しい**機能**を作成し、アプリケーション設計全体を根底から変える方法を把握します。本書の各章はそれぞれ特定の機能を対象としており、記載した周辺モジュールを 1 つまたは複数組み合わせて表す事ができます。

3. **禅**段階では、弟子は全体論的なアプローチを取り、アナログ機能と新たに構築されたデジタル機能を自由に組み合わせて、厳しいタイミング制約をアプリケーションからほとんど(または全て)解消するとともに、(ハードウェア)イベント**チェーン**を介して**自動**応答を実現する事で、コードのサイズと複雑さを軽減し、**完全な**(ミクストシグナル) **ソリューション**を作り上げます。

CIP のパラダイムが他とは異なる理由

個別に見た場合、CIP は必ずしも完全に新しい製品ではありません。その大半は他のアーキテクチャ、またはより大規模なシステムの一部として使われていたか、あるいは現在も使われています。異なる点は、**自律的動作**と**直接相互接続**に焦点を合わせている点です。

さらに、一部の読者は新**機能**を作成するために CIP を組み合わせる事とプログラマブルロジック(FPGA)システム設計を比べたくなるでしょう。しかし、構成要素の粒度を考慮した場合、これらのアプローチには非常に大きな違いがあります。CIP の理念は、個々のゲートではなく非常に大きな機能ユニット(周辺モジュール全体)を再度組み合わせる点にあります。明らかに柔軟性は下がりますが、この CIP アプローチには以下 3 つの大きな利点があります。

- 必要な**スキルセット**は、組み込み制御業界で最も一般的な種類のスキルです。ファームウェア開発経験を最小限しか持たないエンジニアから、コンピュータ科学科の卒業生、電気エンジニア、愛好家まで、短時間で CIP を使いこなす事ができます。これは明らかに、(ソフトの)コア開発を含む SoC 設計全体にプログラマブル ロジックを使う場合にはまったくあてはまりません。その場合に必要なスキルセットは非常に専門的なものであり、ハードウェア記述言語に熟知している事に加え、固有のコア アーキテクチャの知識、接続に関する深い知識まで含まれます。場合によってはこれらの知識は複雑な独自モデルライブラリで補完する必要があります。
- 純粋なプログラマブル ロジック アプリケーションの**消費電力**は、桁違いに高くなります。CIP を備えたマイクロコントローラは超低消費電力で動作します。
- 純粋なプログラマブル ロジック アプリケーションの**コスト**もまた、桁違いに高くなります。CIP を備えたマイクロコントローラは、ローエンドの価格帯で提供されています。

CIP の入手先

Microchip 社ではよく見られる慎重なアプローチに従い、CIP は過去約 5 年間にわたって導入された PIC16F1 マイクロコントローラ ファミリに段階的に追加されてきました。これには 2 つの大きな理由があります。リスクの軽減と新しいマイクロコントローラの競争力維持です。ソフトウェア開発の場合と同様、新しい周辺モジュールを導入して、場合によっては旧型モジュールを置き換える場合、新しいバグが入り込む恐れがあります。モジュールが完全にデバッグされ理解される前にあまりにも速く普及が進むと、確実に多数の問題が発生します。同時に、(適用)対象を明確に絞らないままで、過度に多数の機能を同じ製品に追加すると、エンドユーザに明確な利益をもたらす事なく、コストだけが上昇する結果となります。
補遺 A には、最新の周辺モジュールガイド(DS30010068)のスナップショットを掲載しています。この資料は Microchip 社のウェブサイトから入手でき(少なくとも四半期ごとに更新されます)、多数の新しいマイクロコントローラ ファミリに対する CIP の割り当て/内蔵状況がまとめられています。

ロケット サイエンス

CIP の導入は新たな課題を投げかけます。この新しいパラダイムは、組み込み制御設計者にコンフォート ゾーンからほんの少し踏み出すように求めます。ことわざにもある通り、「かなづちを持っている人には、あらゆる問題が釘に見える」のです。そのため、コンフォート ゾーンを構成する 3 つの要素がタイマ、ADC、PWM であるなら、任意の新規アプリケーションへの自然な取り組み方は、問題に対してこれらの要素を多用した上で、十分な量のソフトウェア、割り込み、そして多数の命令を使って応急的に修正する事になります。
さらに、おそらくはムーアの法則に誘導された現在主流の考え方では、多く(ソフトウェア/性能)を求める事が良い事であると見なされており、見方を変えるためにはきわめて大きい努力が必要になるでしょう。
読者の皆さん、決めるのはあなたです。どのような新しいツールが使用できるのかを確かめ、多数派に逆らってコアから独立したソリューションに目を向け、ソフトウェアと複雑さを**軽減**し、消費電力を**低く抑える**方向を目指すかどうかを。そうすれば、ロケット サイエンスのように難解に思えたソリューションが、実は難解ではないと気付くでしょう。

本書の使い方

本書は標準的な講義の時間(8 時間)を想定して書いており、読み通すのにほぼ同じぐらいの時間がかかると想定しています。微妙な変更の中には理解するのに時間がかかるものもあり、記載された題材を読者が実際に実施/テストする事が重要です。本書の第 1 章に示すハードウェアとソフトウェアの設定に従い、実際に CIP を動かしてみる事を強く推奨します。ご自身が専門知識と個人的関心を持つ特定分野に最も関連する CIP から始めると良いでしょう。各モジュールにある「ホームワーク」セクションでは、新機能がもたらす可能性と考えられる影響について、じっくり考えて頂く事を推奨します。また、理解を深めるために役立つ多数のオンライン資料を参考文献として掲載しています。

本書の対象読者

- 8 ビットマイクロコントローラの知識は既に持っているとお考えなら、改めて考えてみてください。以前の 8 ビットマイクロコントローラとは別物だと気付くでしょう。
- PIC マイクロコントローラの知識があり、好んでいた時期もあるが、ここしばらくは注目していなかったなら、大いに本書を楽しめるでしょう。
- PIC マイクロコントローラの知識があったが、分かりにくい命令セットや独特な癖に好感が持てなかったなら、もう一度見直す事をお勧めします。
- 32 ビット マイクロコントローラを使おうとしたが、ごく簡単なリアルタイム機能の実装でさえひどく複雑で、ソフトウェアに過度に依存するため嫌になった経験があるなら、本書から全く新しい視点が得られるでしょう。

ソフトウェア ツール

本書で使うのはクロス プラットフォーム(Windows、OS X、Linux)統合開発環境(IDE)の **MPLAB® X** と **MPLAB® XC8 コンパイラ**で、どちらも無償で提供されています。また、**MPLAB® Code Configurator (MCC)**も多くの個所で使います。このプラグインにより、コンパクトで安全な API (C で記述した一連の関数)を生成して、**特定**アプリケーションのコンフィグレーション向けにマイクロコントローラ周辺モジュールを初期化して制御できます。

これらのツールは**全て進行中のプロジェクト**であるため、本書の執筆中(2015 年前半)に著者が使ったバージョンとは異なる可能性がある点にご注意ください。

ハードウェア ツール

実践的な学習体験の機会を提供するため、本書で紹介する全てのデモプロジェクトは、数種類の PIC16F1 マイクロコントローラ モデルのサンプルを使ってテストできるようになっています。入手の容易な DIP パッケージで提供されるこれらのサンプルを標準的なブレッドボードに取り付けて使用できます。Microchip 社のデバッガ/プログラマ **PICkit™ 3 (PG164130)**は非常に低コストでありながら、コア アーキテクチャ(8/16/32 ビット)を問わず、1,000 種類以上の PIC マイクロコントローラのプログラミングおよびデバッグに対応しています。また、**PICDEM™ Lab 開発キット(DM163045)**は、ブレッドボードと PICkit 3 プログラマ/デバッガを含んでいます。この他に、最も対費用効果が高い、最近導入された **PICDEM™ Curiosity** ボードは、内蔵プログラマ/デバッガを備え、モジュール式拡張オプション(click ボード)に対応しています。

本書の位置付け

本書は組み込みプログラミングの手引書でも C プログラミングの入門書ではありません。**基本的**な C プログラミングの専門知識とマイクロコントローラ テクノロジに関するいくらかの事前知識を前提としています。
また、本書は個々の PIC16F1 マイクロコントローラ用データシートに取って代わるものではなく、むしろこれらのデータシートは参考文献としてしばしば参照しています。同様に本書は、記載された PIC16F1 マイクロコントローラおよびツールが提供する全機能をもれなくまとめたものではありません。
著者の説明と公式な文書との間で不一致を見つけた場合、常に後者を参照してください。併せて著者にメールでお知らせ頂ければ(pilot@flyingpic24.com)、ブログと書籍ウェブサイトに訂正内容および有益なヒントを掲載いたします。

オンライン サポート

本書で開発した全てのソースコードは書籍ウェブサイト(**http://www.flyingpic24.com**)で一般公開しています。これには、追加(ボーナス)プロジェクトと、本書に必要な(または推奨した)**オンライン コード リポジトリ**およびサードパーティ ツールへのリンクが含まれます。著者は過去数年間にわたって、ブログ『The pilot logbook』(**http://blog.flyingpic24.com**)への投稿を続けており、今後も時間の許す限り続けていく予定です。

第 1 章 セットアップのチェックリスト

本章では、ソフトウェアとハードウェアの初期セットアップを行います。これには、以下のソフトウェアのダウンロードとインストールが含まれます。

- **MPLAB® X** (v2.26 以降) - まだ MPLAB 8 をお使いの場合、MPLAB X にアップグレードしてください。アップグレードすべき理由は、ここに全てを書ききれないほど多数あります。
- **MPLAB® XC8** (v1.33 以降) - この C コンパイラは全ての 8 ビットデバイスをサポートしています。
- **MPLAB® Code Configurator** プラグイン(v. 2.10 以降) - このツールを使うと PIC16F1 モデルが備えている CIP を含む全ての周辺モジュールにアクセスするための最適化済み C 関数を簡単に生成できます。

本章の終わりまでに、「Hello World」に相当するプログラムを作成してデバイスのデジタル I/O ピンの 1 本に接続した LED を点滅させます。後続の各章では、上記 3 つの開発ツールの組み合わせにより容易な構成が実現できる事を実証するため、新しい周辺モジュールの概要を説明した後で、短いコード例を示します。

MPLAB X のインストール

MPLAB X IDE および XC8 コンパイラのインストールが完了している場合、このセクションはスキップして Code Configurator プラグインのインストールに進みます。
まだインストールしていない場合、以下のリンクを使って、Microchip 社ウェブサイトの MPLAB X サポート/ダウンロード ページを開きます。

http://www.microchip.com/mplabx

MPLAB X をインストールする時の注意事項は、以下の通りです。

1. 正しい OS バージョンを選択します。ブラウザのダウンロード ウィンドウをチェックし、Windows の場合は**.exe** ファイル、OS X の場合は**.dmg** ファイル、

Linux ボックスの場合は **tar.gz** ファイルをダウンロードしている事を確認します。

2. 以前のバージョンの MPLAB X をインストール済みの場合、初めにこれを**アンインストール**する必要があります。MPLAB X のアンインストーラを使うと、既存のプロジェクト、ツール、設定が保持され、新しいバージョンを実行すると問題なく復元されます。
3. MPLAB 8 はアンインストール不要です。MPLAB X が MPLAB 8 を上書きしたり、破損させたりする事はありません。

オンライン リソース

`http://microchip.wikidot.com` – Microchip 社の技術トレーニング チームが開発したこの大規模な **wiki** は、多数のチュートリアルと動画を収めています。一通り目を通すだけの価値は十分にあります。よく寄せられる質問(FAQ)と役に立つヒントおよびコツをチェックしてください。

MPLAB XC8 コンパイラ

MPLAB X IDE が全ての PIC®マイクロコントローラに対応するツールであるのに対し、MPLAB XC8 は 8 ビット PIC マイクロコントローラ専用のコンパイラです。
MPLAB XC8 は以下の 2 つのバージョンで提供しています。

- 無償バージョンでは最適化オプションは基本セットのみです。
- **PRO バージョン**は Omniscient(コード)最適化テクノロジを使用しており、高いコード密度と速度を実現します。

無償バージョンは期間限定(60 日間)の PRO ライセンスを含んでいます。このため、PRO バージョンだけが持つ最適化モードで何ができるのかを試し、料金に見合う価値があるかどうかを判断する事ができます。

本書の目的では高度な最適化機能は不要です。無償バージョンで十分です。実のところ、ほとんどのアプリケーションは、一切最適化せずとも問題なく実行できます。

MPLAB XC8 のインストール

MPLAB X のダウンロード ページでは、MPLAB XC8 コンパイラをオペレーティング システム別に提供しています。
ダウンロード後にインストーラを起動するとダイアログ ボックスが表示されます。

1. ライセンス使用許諾契約: 続けるには、使用条件に**同意する必要**があります。
2. XC8 コンパイラのインストール、ネットワーク ライセンス サーバ、ライセンス設定の更新という選択肢が表示されます。ここでは **[install the XC8 compiler]** を選択します。
3. 次に、XC8 コンパイラ ライセンスの種類を選択します。ネットワーク ライセンスを使用する場合を除いて、**[install the XC8 compiler on your computer]** (ローカルアクセスキー)を選択します。
4. すると**ライセンス有効化マネージャ**が表示され、ライセンス有効化キーの入力を促されます。有効化キーを持っていない場合、**フィールドが空白のままで[Next]をクリック**します。
5. 最終確認ダイアログ ボックスで**[Yes]をクリック**します(図 1.1 参照)。
6. 最後に、期間限定の PRO バージョン**評価期間**を有効にするかどうかを選択します。この機能は後で有効にする事もできます。ここでは **[run the compiler in Free mode]** を選択しておきましょう。まずはツールの使い方を覚え、コードを書いてみましょう。**ライセンス マネージャ**(XCLM ツール)を起動して、60 日間の評価期間を開始できます。

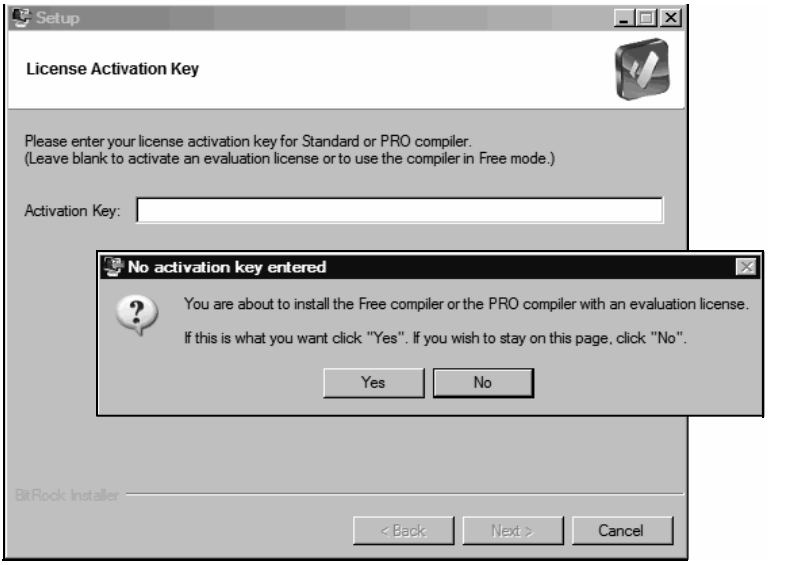


図 1.1: ライセンス有効化マネージャ ダイアログ

MPLAB Code Configurator のインストール

MPLAB Code Configurator は MPLAB X のプラグインであり、MPLAB X からインストール、有効にします。

メインメニューから**[Tools] > [Plugins]**を選択し、**[Available Plugins]**ウィンドウで**[MPLAB Code Configurator]**(図 1.2 参照)を選択して、**[Install]**をクリックします。

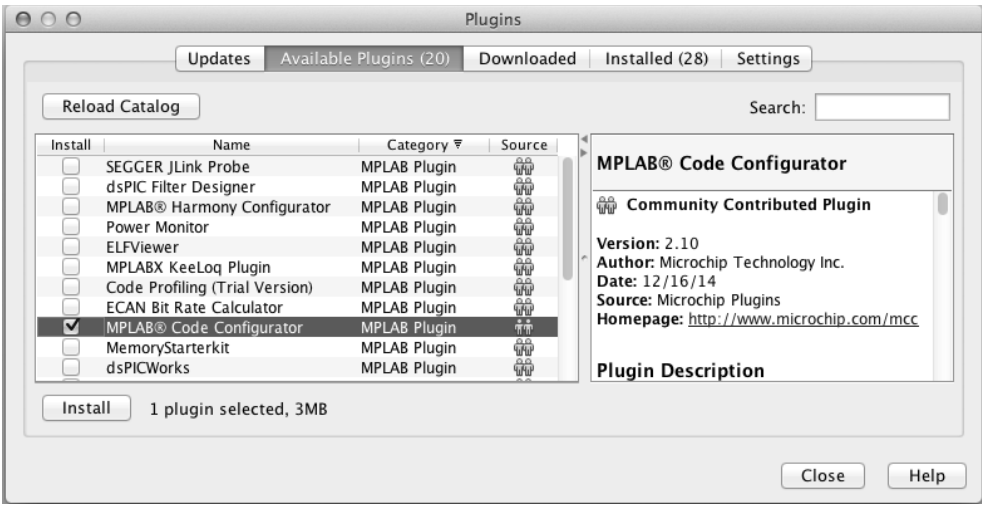


図 1.2: [Plugins]ダイアログ ボックス

インストールが完了したら MPLAB X を再起動します。これにより、[Tools] > [Embedded] メニューに[MPLAB Code Configurator]が表示されます(図 1.3)。

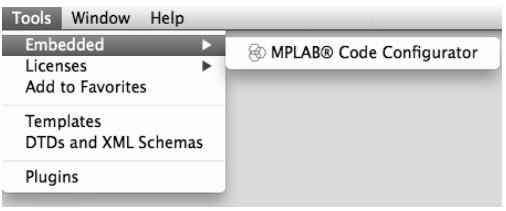


図 1.3: 有効になった MPLAB Code Configurator

プロジェクトの新規作成

既に MPLAB X と XC8 コンパイラを使ってプロジェクトを作成した事がある場合、このセクションは飛ばしてプロトタイプのセットアップに進んでください。

新規プロジェクトのチェックリスト

プロジェクトの新規作成は MPLAB X の[New Project]ウィザードを使って行います。

MPLAB X の**[Start Page]**で**[Create New Project]**を選択するか、またはメインメニューから**[File] > [New Project…]**を選択して以下の 7 ステップでプロジェクトを作成します。

1. **プロジェクト タイプの選択:** [Categories]パネルで[**Microchip Embedded**]オプションを選択します。[Projects]パネルで**[Standalone Project]**を選択して**[Next]**をクリックします。
2. **デバイスの選択:** [Family]ドロップダウン ボックスで[**Mid-range 8-bit MCU**]を選択します。[Device]ドロップダウン ボックスで、PIC マイコンの製品番号を選択します。この例では **PIC16F1619** を選択し、**[Next]**をクリックします。
3. **ヘッダの選択:** 上で選んだ PIC マイコンではヘッダは不要のため、この手順は自動的にスキップされます。その他の製品番号(特に少ピン パッケージ)では選択が必要な場合があります。
4. **ツールの選択: [Simulator]**を選択して**[Next]**をクリックします。
5. **プラグインボードの選択:** このステップは自動的にスキップされます。
6. **コンパイラの選択: [XC8]** (v1.33 以上)を選択して[Next]をクリックします。
7. **プロジェクト名とフォルダの選択:** プロジェクト名に「**1-HelloWorld**」とスペースなしで入力し、作業フォルダを指定して**[Finish]**をクリックします。これでセットアップ ウィザードが完了します。

新規[Projects]ウィンドウが表示されます(図 1.4 参照)。少数の**論理フォルダ**以外は何も表示されません。

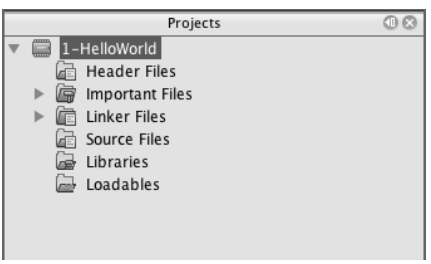


図 1.4: プロジェクトの論理フォルダ

さて、MPLAB X における「プロジェクト」とは何でしょうか。Windows Explorer(Mac 場合は Finder)を開いて作業フォルダに移動すると、MPLAB X が作成したフォルダが見つかります。このフォルダの名前は、プロジェクト名として指定した文字列の後に拡張子「**.X**」が付いたものです。このフォルダをプロジェクトと見なす事ができます。

MPLAB 8 ユーザ向けの注意

MPLAB X のプロジェクトはフォルダであるため、これをダブルクリックしても MPLAB X は起動せず、フォルダの内容が表示されるだけです。これは MPLAB 8 に慣れ親しんだユーザには使いにくいと感じる点かもしれません。ただし、ドラッグ&ドロップ操作であれば、Windows Explorer から MPLAB X のウィンドウにプロジェクト フォルダをドラッグできます。

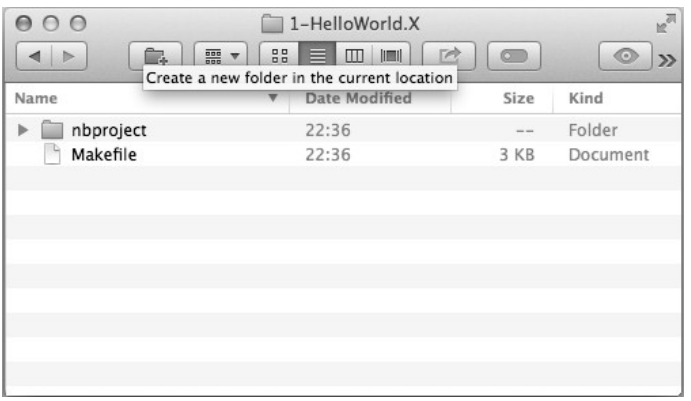


図 1.5: MPLAB X のプロジェクト フォルダ

図 1.5 からも分かるように、プロジェクト フォルダは空ではなく、少なくとも以下 2 つの要素が含まれます。

- **Makefile** は自動生成された make ファイルで、GNU **make** ツールによってプロジェクトをビルドするために使われます。
- **nbproject** フォルダは MPLAB X がプロジェクト設定を保存する場所で、コンパイルするソースファイルの名前リスト、個人設定、選択済みデバッグツール等を含みます。

NOTE:

> このフォルダ名から MPLAB X の成り立ちが分かります。既に明らかなように、MPLAB X は NetBeans IDE プロジェクトをベースとしています。

言うまでもありませんが、これらの 2 つのフォルダは MPLAB X が自動的に生成、管理しているため、内容に手を加えてはいけません。
以下で説明する基本的ガイドラインを守れば、フォルダ内容を*場所に依存しないように*できます。つまり、プロジェクト フォルダ全体を別のパスまたは別のマシンに移動/コピーしても、プロジェクトがそのまま正常に機能するという事です。

論理フォルダ

では、**論理フォルダ**(図 1.4)と実際の物理フォルダの内容(図 1.5)の間には、どのような関係があるのでしょうか。
その関係は、読者が想像するよりもずっとゆるいものです。**論理フォルダ**は単純にファイル名のリストです。これらのファイルの場所は、上で作成したプロジェクト フォルダの実際の位置とは無関係です。
最も重要な論理フォルダは、**Source Files** です。このフォルダに含まれる全てのファイルが、ファイルシステム上の場所にかかわらず、コンパイルされてアプリケーションにリンクされます。それ以外のファイルはこの処理の対象にはなりません。
対照的に、論理フォルダの**Header Files** にファイル名を記載しているのは便宜上の理由からに過ぎません。このフォルダ/リストを空にしたままでプロジェクトをコンパイルしても、問題なくコンパイルが完了します。もちろん一定のルールに従う必要はあるため、プロジェクトで最も重要なヘッダファイル名をここに記載します。こうする事で、プロジェクトの文書化と保守に役立ちます。
同様に、**Important Files** フォルダに含まれるのは、単に makefile へのリンクです。これは参照に過ぎません。ファイル名をダブルクリックすると内蔵エディタで内容を表示できますが、変更してはいけません。このフォルダにファイルを追加しても、プロジェクトのビルドプロセスには影響しません。
本書では **Linker Files**、**Library Files**、**Loadable Files** フォルダは使用しません。各フォルダについては、MPLAB X の公式文書を参照してください。

新規ファイルのチェックリスト

次に **main** プロジェクト ソースファイルを作成します。新規プロジェクトに新しいファイル(特に **main.c**)を追加する方法には、少なくとも以下の 3 つがあります。

1. **[New File]ウィザード**を起動し、テンプレートを使ってファイルを作成します。
2. [New File]ウィザードを起動し、テンプレートを使わずにファイルを作成します。
3. MPLAB Code Configurator を開き、ファイルを自動生成します(著者のような怠け者に推奨します)。

[New File]ウィザードを使うには **Ctrl+N** コマンド(Mac ユーザの場合、⌘**-N**)、MPLAB X のメインメニューから**[File] > [New File]**を選択、**[File]ツールバー**で**[New File]** アイコンをクリック(アクティブな場合)のいずれかを使います。

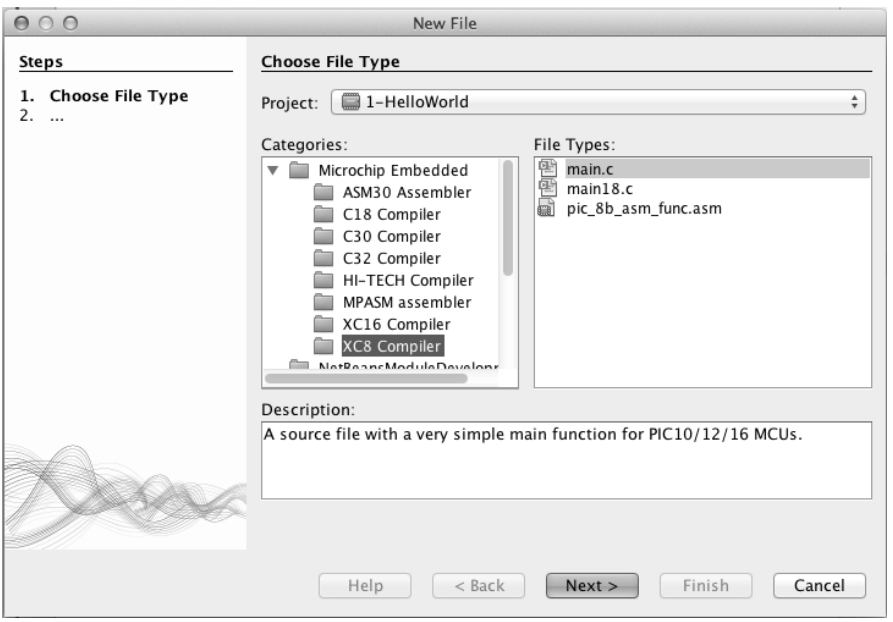


図 1.6: [New File]ウィザード

[New File]ウィザード(図 1.6 参照)は 2 つのダイアログ ボックスで構成されており、手順は以下の通りです。

1. **ファイルタイプの選択: [Categories]**ウィンドウで**[Microchip Embedded]**を選択します。

2. サブカテゴリ リストが展開されたら、**[XC8 Compiler]**を選択します。
3. 右側の**[File Types]**ウィンドウで **main.c** タイプを選択します。
4. **[Next]**をクリックします。
5. **[Name and Location]**ダイアログ ボックスが開きます。ほとんどのフィールドは、プロジェクト (フォルダ)の既定値設定から事前入力されます。指定する必要があるのは新規ファイルの名前だけです。**main.c** と入力します。
6. **[Finish]**をクリックします。

MPLAB X が xc8 用のテンプレートを使って、新しい **main.c** ファイルを作成します。テンプレートは以下に示す数行のコードで構成されています。

```
/*
 * File:   main.c
 * Author:(your name here)
 *
 * Created on (date and time here)
 */

#include "xc.h"

void main( void )
{
    return;
}
```

リスト 1.1– MPLAB XC8 の main.c テンプレート

その他に、**main.c** ファイルの作成以外でこのウィザードを使用する例は、空のファイルを作成して手作業でコードを入力する方法です。その場合、最初のダイアログボックスで**[Other]**カテゴリと**[Empty File]**タイプを選択します。

どちらのケースも、ウィザードはファイルを作成してテンプレートから必要な内容を生成するだけでなく、自動的に**新しく作成したファイルをプロジェクトに追加**します([Project]ウィンドウで **Source Files** 論理フォルダを開いて選択している場合)。
ファイルが自動的に追加されない場合、新規ファイルを保存した後に、[Project]ウィンドウで **Source Files** 論理フォルダを選択し、コンテクスト メニューから**[Add Existing Item...]**を選択します。

前述した通り、プロジェクトの **main.c** ファイルを新規作成する 3 番目の方法は、MPLAB Code Configurator で自動生成する方法です。

MCC の起動

MPLAB Code Configurator の最も有効な使い方は、プロジェクト新規作成の最初から使う方法です。MCC を有効化するには、MPLAB X のメインメニューから**[Tools] > [Embedded]**を選択します(図 1.3)。

[Embedded]メニューが表示されない場合、MCC プラグインを正しくインストールした事や、インストール後に MPLAB X を再起動して MCC プラグインを有効にした事を確認する必要があります。

MCC が有効になると、2 つのダイアログ ウィンドウ(最終的には 3 つ)で構成された MCC が表示されます(図 1.7 参照)。左側に**リソース選択**ウィンドウが表示され、**コンフィグレーション ダイアログ** ウィンドウがタブとしてエディタ ウィンドウ内に表示されます。さらに後から、右側に**[Pin Manager]**ウィンドウが表示されます。

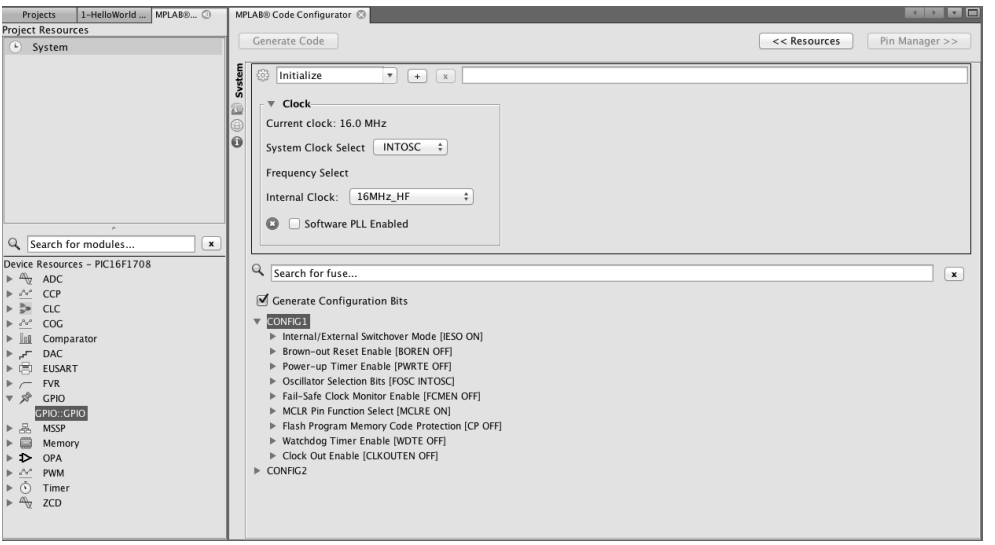


図 1.7: MPLAB® Code Configurator のシステム ダイアログ ウィンドウ

リソース選択ウィンドウは、さらに上下 2 つのウィンドウに分割されています。上のウィンドウには、プロジェクトで使うべく選択したリソースが一覧表示されます。下のウィンドウには、選択したマイクロコントローラ モデルで使える全てのリソース(かつ未使用のもの)が一覧表示されます。

システム設定

初期状態で**[Project Resources]**リスト(左上のウィンドウ)に含まれるリソースは、**System** のみです。**これをクリック**すると、中央のダイアログ ウィンドウで対応するコンフィグレーション ダイアログ ボックスが開きます。ここで、起動時のシステム既定値設定を以下の通りに指定します。

- システムクロックの選択: **INTOSC**
- 周波数の選択: **4 MHz**(ここでは PLL のチェックを外したままにする)

[Generate Configuration Bits]オプション(画面下部)にチェックを入れ、開発時の都合に合わせて以下のオプションを設定します。

- **[Config1]**ワード: **[MCLRE ON]**以外の全てのオプションを OFF にします。
- **[Config2]**ワード: [Low Voltage Programming Enable] (**[LVP ON]**)以外の全てのオプションを**既定値**に設定します。

個別の機能と上記選択理由については後述します。

この時 MCC のメインダイアログ ウィンドウで[Generate Code]ボタンが有効になり(カッコ内に「1」と表示)、少なくとも 1 つのモジュール設定(System)が変更されて、コードを(再)生成できる状態にある事が示されます。ボタンを**クリック**すると、MCC がコード生成を開始します。

このプロセスを初めて実行する場合、MCC は **main.c** ファイルが存在するかどうかをチェックし、(自動生成を選んだために)ファイルが存在しない場合、ファイルを自動的に生成するかどうかを尋ねてきます(図 1.8)。

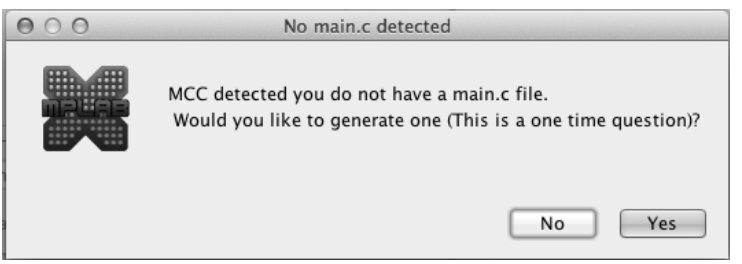


図 1.8: main.c の自動生成

MCC は以下の新しい要素をプロジェクトに追加します。

- **main.c** ファイル - 実行時に存在しなかった場合
- MCC Generated Files 論理フォルダ - 以下のファイルを収めます。
- **mcc.c** ファイルは、コンフィグレーション ビット設定(`#pragmas`)、`SYSTEM_Initialize()`関数、`OSCILLATOR_Initialize()`関数を収めています。
- 同時に、対応するヘッダファイル(**mcc.h**)が作成され、Header Files 論理フォルダに追加されます。

I/O の管理

「Hello World」メッセージを表示するためには、最低でも 1 本の I/O ピンを有効にする必要があります。[Device Resources]リスト(左下のパネル)から GPIO(汎用 I/O)リソースを選択すると、MCC で必要な設定を全て実行できます。
GPIO を**ダブルクリック**し、[Project Resources]リスト(左上のパネル)に表示されたら、**もう一度クリック**します。

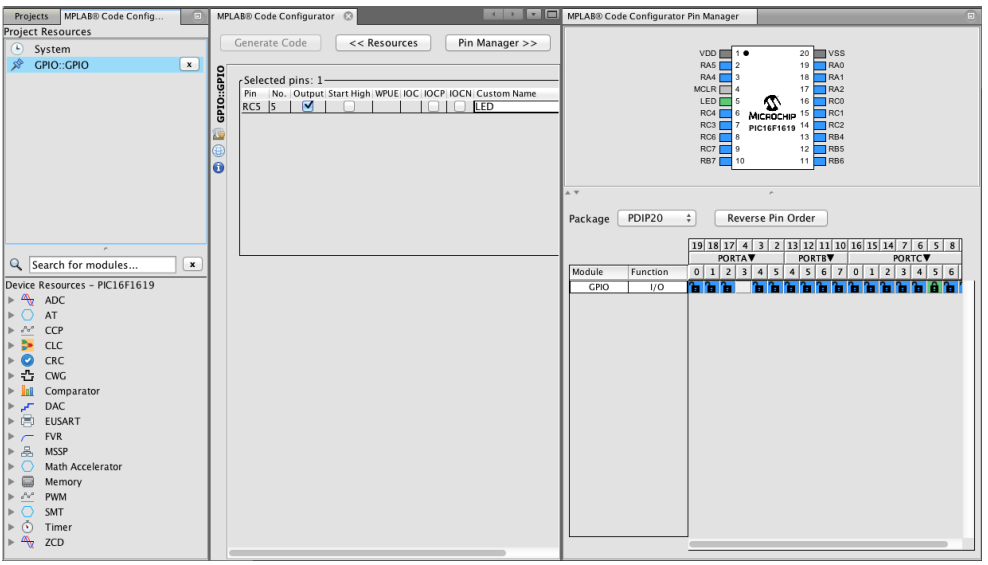


図 1.9: MCC の GPIO ダイアログ ウィンドウと Pin Manager ウィンドウ

これにより、コンフィグレーション ダイアログ ウィンドウに[Selected pins]テーブルが表示され、Pin Manager ウィンドウ(図 1.9)が表示されます。

Pin Manager ウィンドウ右下のテーブルで、RA2 ピンに対応する小さな青の鍵アイコンを**選択**すると、アイコンが緑色に変わってロックがかかります。
メインダイアログ ウィンドウで、[Selected pins]テーブルに RA2 が追加されます。**[Output]オプションにチェック**を入れ、右端の**[Custom Name]フィールドをクリック**してピン名を入力します。この例では「**LED**」としましょう。
もう一度**[Generate Code]**ボタンを**クリック**すると、MCC が新しい GPIO モジュールと設定をプロジェクトに追加します。
pin_manager.c という新しいファイル(および対応するヘッダファイル)が、MCC Generated Files 内のリストに追加されます。このファイルに既定値で含まれる `PIN_MANAGER_Initialize()` 関数は、必要な I/O 制御レジスタ全てを完全に初期化します。
さらに、**pin_manager.h** ファイルには `LED_SetHigh()`、`LED_SetLow()`、`LED_Toggle()` 等、カスタム名を付けたピンを管理する多数のマクロ(#define)が既に含まれています。

シミュレータを使った Hello World

普通のプログラミング書籍は必ず「Hello World」の例を収めています。組み込み分野では必ずしも画面/端末にテキストを表示せず、しばしば LED の点滅でアクティビティを表示します。
ここまでコードを 1 行も書いていませんが、プロジェクトには既に適切に構造化された関数一式(API)が存在し、必要な起動オプション、オシレータ、リセット設定でマイクロコントローラを初期化し、LED 出力を操作する準備が整っています。
では、MCC が生成したテンプレートの main()関数を編集して、以下のようにメインループ内に 1 行のコードを追加しましょう。

```
/*
   Generated Main File
   ...
*/
#include "mcc_generated_files/mcc.h"

/*
   Main application
 */
void main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        LED_Toggle();
    }
}
```

リスト 1.2- シミュレータを使った Hello World

コードをテストするため、**プロジェクトをビルド**し、**MPLAB X シミュレータ**を使って**実行**します。メインメニューから**[Run] > [Run Project]**を選択するか、またはキーボードショートカットの **F6** (Mac ユーザの場合、**Fn+F6**)を使います。

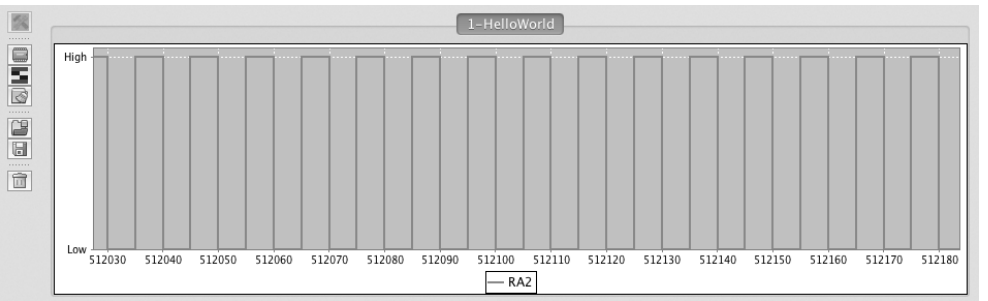


図 1.10: [Simulator Analyzer]ウィンドウ

[Simulator Analyzer]ウィンドウが開いたら、**RA2 pin** の出力を選んでグラフィック表示します(図 1.10)。
RA2 ピンで発生した実際の矩形波を識別できるようにするには、多少ズームする必要があるかもしれません。これは、プロセッサが最大速度でピンをトグルさせているためです。

ハードウェア プロトタイピング

MPLAB シミュレータは、基本的なアプリケーション ロジックを手早くチェックするために便利なツールです。このシミュレータはほとんどの基本的な周辺モジュールをサポートしており、入力信号は**同期または非同期のスティミュラス**を使って注入できます。しかし、簡単なハードウェア プロトタイプを使った方が、新しい CIP モジュールの多くをずっと楽しくテストできます。

ブレッドボード+ PICkit™ 3

最も基本的な方法は、DIP パッケージのマイクロコントローラをブレッドボードに差し込み、インサーキット シリアル プログラミング™ (ICSP™)に必要な 5 本のピンを PICkit 3 プログラマ/デバッガに接続する方法です(配線例は図 1.11 参照)。

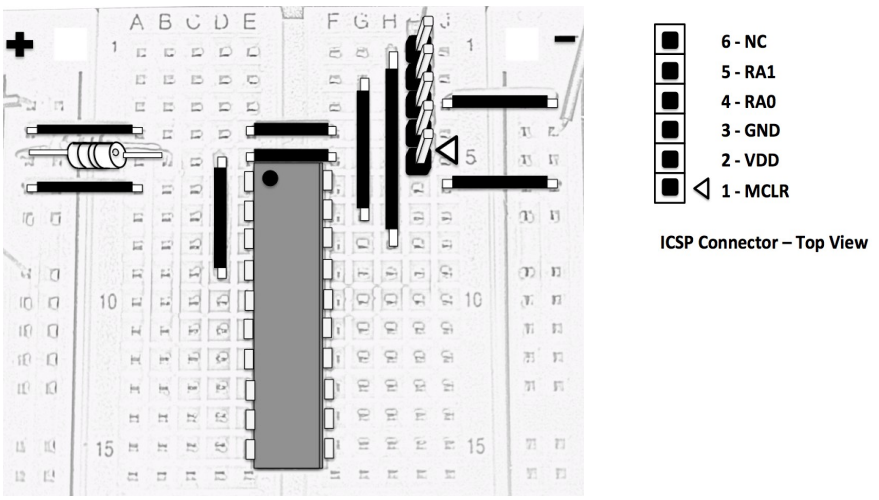


図 1.11: ブレッドボードを使ったハードウェア プロトタイピング

PICkit 3 はターゲット回路に電源を供給する事もできます。図 1.12 では**プロジェクト設定**ダイアログ ボックスで PICkit 3 を選択し、**Power** 設定ウィンドウで **Power Target** と 5 V 出力を有効にしています。

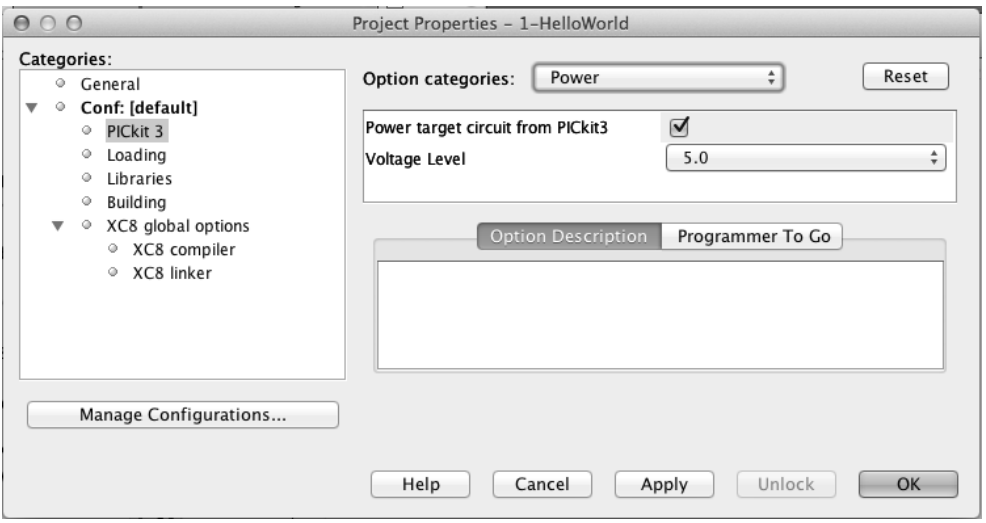


図 1.12: PICKit 3 による電源供給

PICDEM™ Lab 開発キット

PICDEM Lab 開発キット(DM163045)は PICKit 3、ブレッドボード、各種ソケットで構成されており、あらかじめ PIC マイクロコントローラに配線されています。

少ピンデモボード

少ピン(最大 20 ピン)PIC16F1 ファミリを使って迅速にプロトタイピングする方法には、**少ピンデモボード**(DM164130-9)を使う方法があります。このデモボードは PICkit 3 とともに **PICkit 3 スタータキット**(DV164130)に含まれています。この場合、基板に実装済みの LED は RC0～RC3 ピンに接続されています。これに合わせて GPIO の設定を変更する必要がありますが、それ以外は何も変更しなくても、最初に示したサンプルコードが正しく実行されます。

PICDEM™ Curiosity

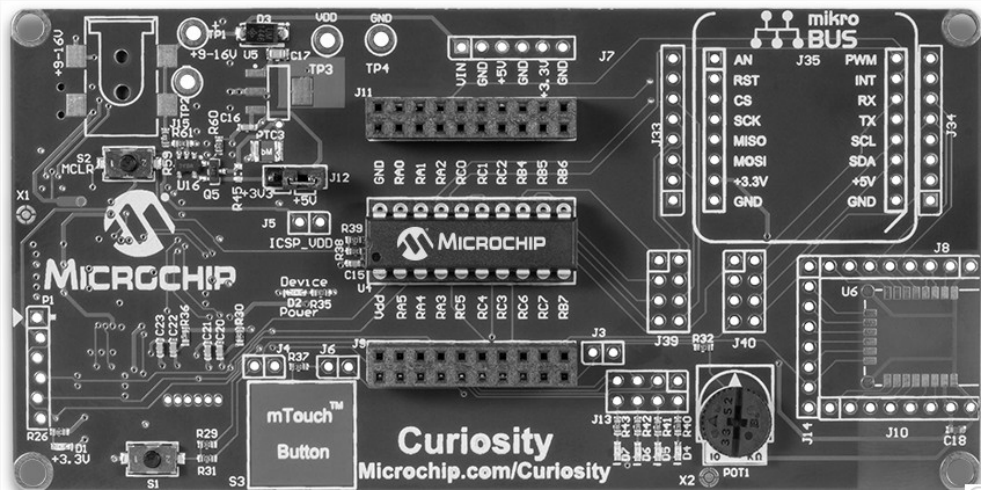


図 1.13: PICDEM Curiosity ボード

最新かつ最良のプロトタイピング方法は、**PICDEM Curiosity ボード**を使う方法です(図 1.13 参照)。このボードには各種方法で電源を供給する事ができ、MPLAB X と USB で直接接続してプログラミングとデバッグができるため、PICkit 3 等のプログラマは不要です。また、100 種類以上の **click™ボード**を接続できる **mikroBUS™**コネクタを搭載しています(右上)。mikroBUS は、センサ、アクチュエータ、ディスプレイ、各種有線/無線インターフェイスを提供する MikroElektronika D.O.O.社(**http://mikroe.com**)の製品です。

ハードウェアを使った Hello World

どのようなハードウェア プロトタイピング ソリューションを選んだかにかかわらず、PIC16F1619 の出力ピンとグランド間抵抗(500 Ω～1 kΩ)と LED を直列に**接続**します。

リスト 1.3に示すように、ビルトイン関数**__delay_ms()** を使って 0.5 秒の遅延を追加したら、**プロジェクトをリビルド**してターゲット上で**実行**します。メインメニューから**[Run] > [Run Project]**を選択するか、またはキーボード ショートカットの **F6** (Mac の場合、**Fn+F6**)を使います。

```
/*
   Generated Main File
   Device:PIC16F1619
*/
#include "mcc_generated_files/mcc.h"

/*
   Main application
 */
void main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        __delay_ms(500);    // builtin function
        LED_Toggle();
    }
}
```

リスト 1.3- ハードウェアを使った Hello World

全ての処理が正常に完了した場合、LED が約 1 Hz の周期で点滅します。

Hello World!

ホームワーク

最初のプロジェクトを生成した手順を振り返ると、非常に多くの処理が MCC によって自動的に実行された事に気付くでしょう。

- MCC が生成した全てのファイルを確認しましょう。
- MCC が選んだコンフィグレーション レジスタ設定に対してどうコメントしているか確認しましょう。
- 設定(システムクロックまたは GPIO の選択等)を変更し、MCC でソースファイルを再生成します。ソースファイルがどのように変更されたか確認しましょう。
- 生成されたソースファイルの一部を手動で変更し、MCC に再生成させます。競合が発生するとどうなるでしょうか。
- GPIO 初期化関数を追加してみましょう(ヒント: [Initialize]コンボボックス横のプラスボタンを使います)。

第 2 章 タイミング機能

Enhanced

オシレータ

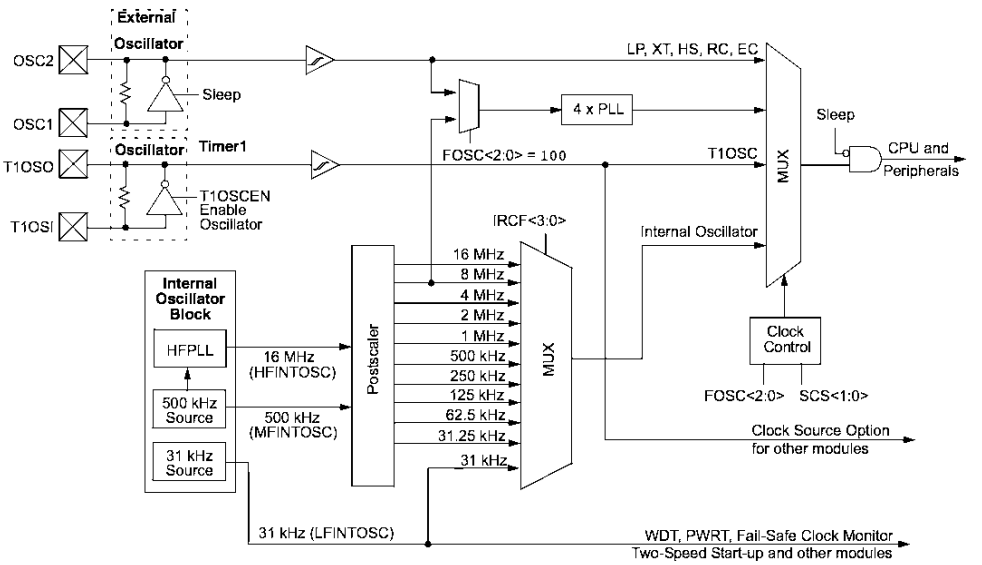


図 2.1: クロック源の概略ブロック図

概要

意外に思われる方もいるかもしれませんが、最近の PIC マイクロコントローラのほとんどが、少なくとも 5 つの異なるオシレータ回路を内蔵しています。これには以下が含まれます。

- **プライマリ外付けオシレータ**(POSC): 主に、高いタイミング精度(代表誤差 1%以下)が要求されるアプリケーションで、非同期シリアル通信のシステムクロック源として使います。3 つのモード(LP、XT、HS)から 1 つを選択する事で回路のゲインを調整し、周波数に応じて水晶振動子の性能と消費電力を最適化できます。また EC モードでは、適切なクロック信号が使える場合に直接入力が可能です。
- **セカンダリ オシレータ**(SOSC): 16 ビットタイマ(Timer1)に直結できるメイン システムクロックから独立した 2 つ目の水晶振動子に接続できます。32 kHz 動作専用の低ゲイン回路で、低消費電力リアルタイム クロック(RTC)として動作します。

- **内部オシレータ**(HFIntOSC): 主に、低コスト アプリケーションまたは高精度が不要の場合にプライマリ外付けオシレータの代わりに使います。この回路は動作周波数を 31 kHz～32 MHz の間で選択して消費電力を最適化できます。
- **低消費電力内部オシレータ**(LFIntOSC): 低消費電力/低精度のオシレータで、ウォッチドッグ回路、BOR、フェイルセーフ クロック、パワーオンタイマ回路等のモジュールが使います。
- **ADC オシレータ**(FRC): A/D コンバータ モジュール専用に割り当てられた固定周波数(公称 600 kHz)の低消費電力内部オシレータで、スリープ中でも ADC を動作可能にします(第 9 章「XLP」参照)。

一部の周辺モジュールは専用オシレータ回路とモードを備えています (USB および PSMC モジュール参照)。

動作原理

内部オシレータ(HFIntOSC)公差の公称値($T_{ambient}$、5 V)は工場で 1% (typ.)に校正済みですが、モデル依存であり、かつ周囲温度と供給電圧で変化します。オシレータ調整レジスタ(OSCTUN)を使うと、製造ラインの最後でカスタム校正を追加で実施できます。また、温度または電圧を正確に計測できる場合、動作中に温度/電圧の変化に対して**補償**する事もできます(「FVR - 固定参照電圧」と「温度インジケータ」参照)。

外付け水晶振動子が高精度で安定したクロック信号を供給するまでにはセトリングタイムが必要であるため、外付け水晶振動子を使う場合、**オシレータ起動タイマ**(OST)が自動的に有効になり、1024 カウントの遅延を発生させます。外付けオシレータの起動からコード実行までのレイテンシを最小限に抑えるために、**2 段階起動**モードを選択できます。このデバイス コンフィグレーションではデバイスが素早く起動し、内部オシレータを使って即座にアプリケーション コードの実行を開始した後で、外部 (高精度)オシレータの動作が安定したら切り換えます。

電源電圧(Vdd)の立ち上がりが非常に遅い場合、**パワーアップ タイマ**(PWRT)を使って長い遅延(64 ms)を挿入して起動シーケンスを延長する事で、Vdd が適切なレベルに達してからコード実行を開始する事ができます。内部および外部オシレータの周波数レンジを拡張するために、32 MHz で動作するモデルでは PLL 回路(x4)を使えます。PLL はソフトウェアで制御するか、または起動時から実行できます。2 段階起動、パワーアップ タイマ、PLL イネーブルは、デバイスのコンフィグレーション ビットで選択できます。

アプリケーション

オシレータの公差と消費電力の特性によって応用範囲が決まります。

- 非同期シリアル通信(UART)では 2%未満の公差が必要であるため、温度または電圧レンジを制限/補償できない場合、外部の水晶/セラミック振動子を使う必要があります。
- USB 通信(フルスピード、12 Mbps)では 0.25%未満の公差が必要なため、外付け水晶振動子または**アクティブ クロック チューニング**(第 6 章参照)を使う必要があります。
- バッテリ アプリケーションでは通常、内部/外部オシレータを最低動作周波数と最小消費電力モードで使う必要があります。また、内部オシレータを使うと起動時間を短縮できるため、アプリケーション全体の消費電力を大幅に節約できる可能性があります。

制限事項

最近導入されたいくつかのエントリレベル モデル(PIC16F150x ファミリの少ピンモデル等)は、コストと複雑さを抑えるために、全ての外付けオシレータモードには対応していません。

MCC が生成する API

MCC はオシレータ コンフィグレーション オプションの大半を **System** リソース内にグループ化しています。同じダイアログ ウィンドウで、起動時のオシレータ コンフィグレーション(FOSC)、**クロック出力**機能のイネーブル機能(**CLKOUTEN**)、**外部/内部**クロック切り換え機能(IESO)、フェイルセーフ クロック モニタ(FCMEN)、ウォッチドッグの設定に対応するデバイス **コンフィグレーション ビット**を生成できます(第 1 章「システム設定」参照)。MCC は、生成されたコンフィグレーション ビット (`#pragma`) と `Oscillator_Initialize()` 関数を **mcc.c** ファイルに格納します。初期化関数の呼び出しは、`SYSTEM_Initialize()` シーケンスに自動挿入されます。

ピン配置

以下の機能を有効にした場合、マイクロコントローラの I/O ピンの一部が専有されて、その他の設定より優先されます。

- CLKOUT (out) — クロック出力(F_{osc}/4): アプリケーションのカスケード接続またはテスト/デバッグ用

- OSC1 (in)および OSC2 (out) — 外部水晶/セラミック振動子に接続
- OSC1 (in) — 外部クロック源に接続(EC モード)
- T1OSI (in)および T1OSO (out) — 低消費電力セカンダリ オシレータ(32 kHz 水晶振動子)に接続(通常 28 ピン以上のデバイスが装備)

ホームワーク

- アプリケーションと周辺モジュールを常に最適なクロック速度で実行するため、クロック スイッチングについて学習しましょう。
- 2 段階起動について学習し、効率的なコード実行により時間/消費電力を節約できる事を確認しましょう。
- フェイルセーフ クロック モニタ機能について調べます(詳細は第 5 章「安全性機能」参照)。

オンライン リソース

『AN1303 - Software Real-Time Clock and Calendar』
『AN1798 - Crystal Selection for Low-Power Secondary Oscillator』
『AN849 - Basic PICmicro® Oscillator Design』

8/16 ビットタイマ

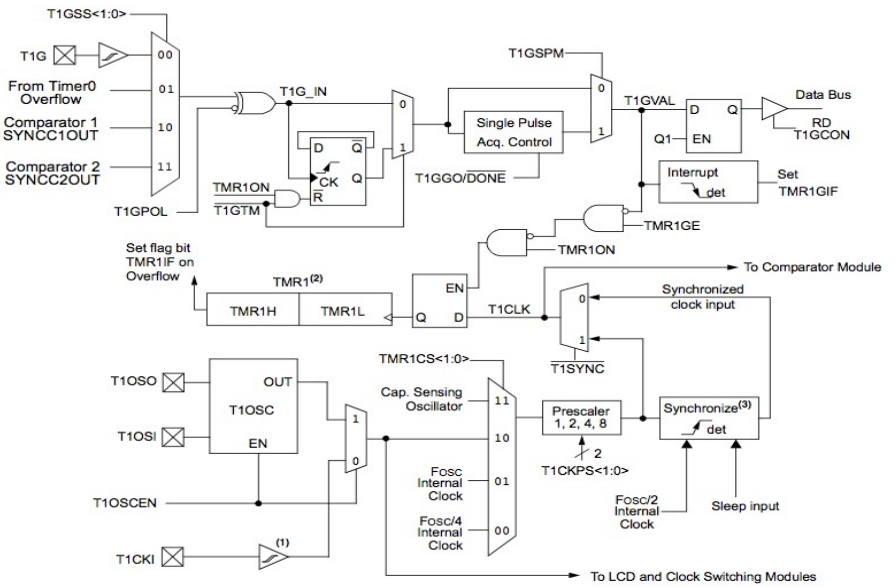


図 2.2: 16 ビット(奇数)タイマのブロック図

概要

全ての PIC マイクロコントローラで下位互換性を維持するという Microchip 社のこだわりは、タイマ部分の構造を解析すると一目瞭然です。
タイマには基本的に以下の 3 種類があります。

- 奇数番号付きタイマ(Timer1、Timer3、Timer5 等)はやや高度な 16 ビット構造でセカンダリ オシレータ(SOSC)モジュールと組み合わせた動作(第 2 章の「オシレータ」セクション参照)や、キャプチャ/コンペアとの統合(第 2 章の「CCP」セクション参照)が可能です。
- 偶数番号付きタイマ(Timer2、Timer4、Timer6 等)は単純な 8 ビット構造で、プリスケーラ、ポストスケーラ、簡単な周期レジスタを備えています。偶数番号付きタイマのクロック源は、ほぼシステム(命令)クロック(Fosc/4)に限定されます。

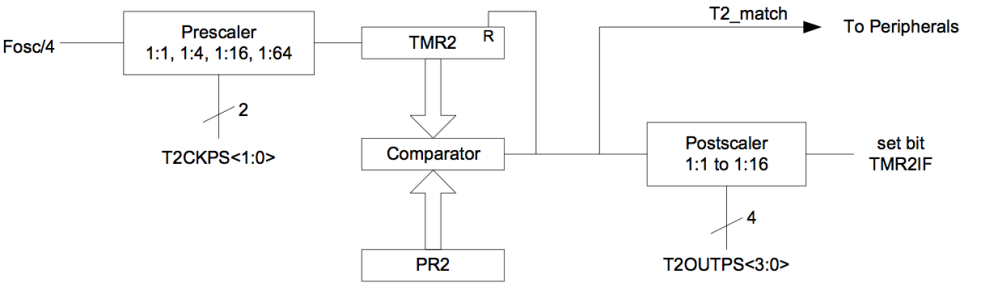


図 2.3: 8 ビット(偶数)タイマのブロック図

- Timer0 は非常に優れています。きわめてシンプルな設計の 8 ビット構造をしており、25 年の歴史を持つ PIC アプリケーションに最大限の下位互換性を提供します。周期レジスタはありませんが、T0CKI 入力ピンにパルスカウンタとしての機能を備えています。

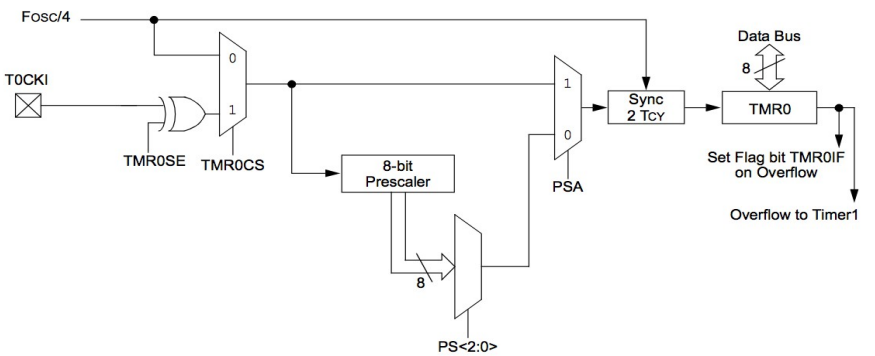


図 2.4: Timer0 のブロック図

動作原理

いかなるタイマも基本的な周期生成機能(タイマモード)と、TxCKI 入力ピンを介してパルスカウンタとして動作する機能(カウンタモード)を備えています。

Timer0 にはコンパレータと周期レジスタがなく、クロック整数倍(最大 256x)のパルス生成しかできません。手動で**オフセット値**をロードすれば、ロールオーバー(TMR0IF)までに指定の時間間隔を作る事ができます。

偶数(8 ビット)タイマは柔軟性が比較的高く、対応する周期(PRx)レジスタに必要な値を設定するだけで済みます。CCP および PWM モジュールと組み合わせて使うと、PWM 信号周期を制御できます。
奇数(16 ビット)タイマは最も柔軟性が高く、ゲート付きタイマとして動作できるだけでなく、CCP モジュールと連携してパルス幅を計測(キャプチャモード)、または任意の幅のシングルパルスを生成できます(コンペアモード)。

アプリケーション

言うまでもなく、これらのタイマの使い方は多岐にわたり、イベントのスケジューリング、安全タイムアウトの生成、入力信号の周波数、周期、デューティ サイクルの計測等があります。
また、通信モジュールによっては 8 ビットタイマ(ほとんどは Timer2)を baud レート ジェネレータとして使う場合がある事に注意します(「I²C」および「SPI」セクション参照)。

制限事項

16 ビットタイマは比較的高度な機能を備えていますが、入力 PWM 信号のデューティ サイクルの計測、または固定 ON 時間の PWM 出力の生成等、一般的な機能が複雑になり、多数の CPU サイクルを必要とする場合があります。そのような場合に、最近導入された信号計測タイマ(SMT)とハードウェア リミットタイマ(HLT)がどのように役立つかについては後述します。

MCC が生成する API

MCC はデバイス上にあるタイマモジュール別にダイアログ ウィンドウを表示し、対応する timerX.c ファイルを生成します。このファイルは、タイマごとに最適化された `TimerX_Initialize()`、`Start/Stop`、`Read/Write` 関数を収めています。
また、タイマ**割り込み**オプションが有効の場合、MCC は対応するエントリを **Interrupt Manager** のベクタテーブルに自動で作成します(さらに、**interrrupt_manager.c** ファイルを生成します)。
例外として、Timer0 の割り込み処理を有効にすると、MCC はより高機能な偶数(8 ビット)タイマをシミュレートする**周期リロード**機能を追加できます。ただし、多少の CPU オーバーヘッドが発生します。

ピン配置

初期の PIC16F18xx および PIC16F19xx ファミリでは、各タイマのゲーティングおよびクロック入力(T1G、T1CKI 等)ピンは固定でした。
低コストの PIC16F15xx ファミリでは、多数の代替ピンまたは各種周辺モジュール出力に構成可能なロジックセル(CLC)経由でタイマ入力を直接接続する機能が加わりました。
より最近のデバイス (PIC16F170x、PIC16F171x、PIC16F188xx 等) では、ペリフェラル ピンセレクト(PPS) モジュールを使って、任意のデジタル I/O にタイマ入出力を接続できます。

ホームワーク

- 8/16 ビットタイマの非同期モードについて調べてみましょう。
- Timer1 のゲートモードについて調べてみましょう。

オンライン リソース

『TB3100 - Timer1 Timer Mode Interrupt Latency』

CCP – キャプチャ/コンペア/PWM

Enhanced

概要

過去 25 年間、キャプチャ/コンペア/PWM モジュール(CCP)はほとんど常に PIC マイクロコントローラの周辺モジュールセットに含まれていました。このため、一つ一つのゲートを手動で選んで配置し、無駄にできるものが何もなかった初期の時代の特徴である「節約」が設計に表れています。

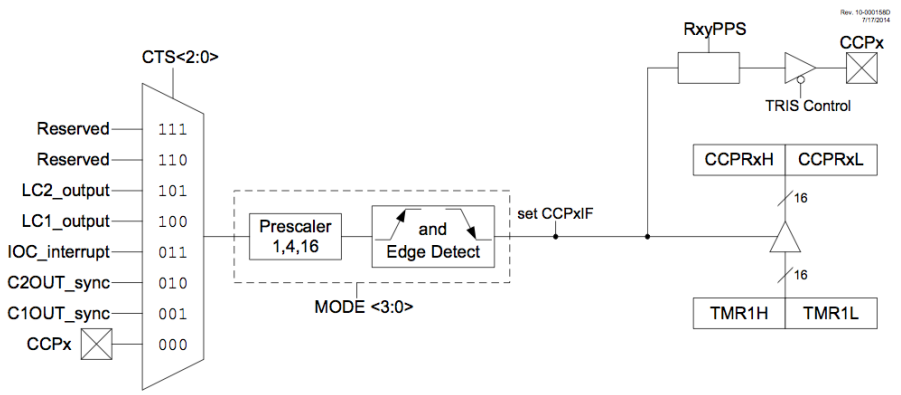


図 2.5: キャプチャモード動作のブロック図

CCP モジュールはキャプチャとコンペアという 2 つの最も基本的な動作を実際に行うために、16 ビットタイマへの接続に依存しています(図 2.5 参照)。
この結果、キャプチャおよびコンペア機能の分解能が 16 ビットになり、広いダイナミック レンジが得られます。
一方、PWM 出力を生成するために、CCP モジュールは 8 ビットタイマとその周期レジスタへの接続に依存します(図 2.6 参照)。
この結果、周期分解能は 8 ビットに限定され、フル システムクロックを有効に使用しない限り、デューティ サイクルも同様に制約を受けます。その他の全てのマイクロコントローラ機能はプロセッサの命令クロック(Fosc/4)に基づきますが、(PWM モードの)CCP はフルシステムクロック(Fosc)にアクセスできるため、分解能ビット数が 2 増えて合計で 10 になります。

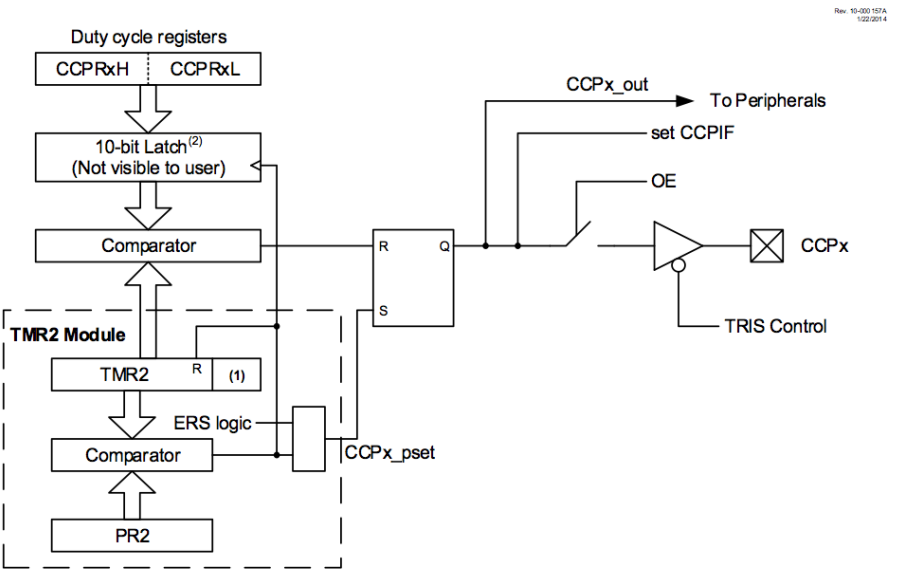


図 2.6: PWM の概略ブロック図

動作原理

前述した設計上の制約により、CCP モジュールが機能を実行するには 8 ビットまたは 16 ビットタイマが必要です。

CCP モジュールと奇数(16 ビット)タイマを組み合わせると、16 ビットの**キャプチャを**実行して入力パルスの幅を計測できます。

また、CCP モジュールと奇数(16 ビット)タイマを組み合わせると、16 ビットの**コンペア**機能を実行して必要な幅の出力パルスを生成できます。

CCP モジュールと偶数(8 ビット)タイマを組み合わせると、周期分解能が 8 ビットでデューティ サイクル分解能が 10 ビットの出力 **PWM** 信号を生成できます。

一部のモデル(PIC16F19xx 等)では PWM モードが**強化されており(ECCP)**、単純なパルス/PWM 矩形波ではなく、相補出力信号ペアを生成できます。これは、相補的な MOSFET デバイスペアを負荷に接続する、多くの電源制御アプリケーションで便利です。大きなゲート寄生容量により、転流中に両方の電源デバイスが同時に有効になる事(結果として生じる貫通電流)を防止するため、デッドバンド制御(遅延)を追加できます。

さらに最近の設計では、**相補出力を制御**する回路が CCP モジュールから分離されており、完全に独立したモジュールとして使えます。(第 2 章の「CWG」および「COG」セクション参照)。

PWM 専用モジュール

同様に、最近のマイクロコントローラ ファミリでは、PWM 機能のみを備えたモジュールが追加されています。これは、CCP モジュールの用途として PWM 機能が圧倒的に多い事が分かったためです。

アプリケーション

パルスの計測/生成(キャプチャ/コンペア)は、各種組み込みアプリケーションで使われており、その用途はセンサとの接続と簡単なアクチュエータの制御です。
PWM 出力はサーボ制御に使われており、電源およびモータ制御アプリケーションで一般的です。
また、D/A 変換ツールとして使う事もできます。実際には、10 ビットまでの分解能では、PWM 出力からアナログ出力信号への変換は単純な RC フィルタで十分です。

制限事項

公称分解能(デューティ サイクル コンパレータ回路で使用可能なビット数)がいくつであっても、PWM の**実効(DC)**分解能は使用可能なクロック源によって常に制限されます。以下にその計算式を示します。

a)

$$F_{osc} = 2^{PWMresolution} * PWM_{freq}$$

b) PWM 分解能を求める:

$$PWM_{resolution} = \log_2(F_{osc} / PWM_{freq})$$

計算式 2.1- 実効 PWM 分解能

計算式 2.1b に示した通り、クロック周波数(F_{osc})が一定の場合、PWM 周波数が高くなれば、結果的に PWM の実効分解能は低くなります。
例えば、32 MHz のシステムクロックで 10 ビットの分解能を達成できるのは、PWM 周波数が 32 kHz 未満の時だけです。

MCC が生成する API

MCC は適切なタイマを選択して CCP モジュールへ接続するプロセスを完全に自動化する事で、CCP および PWM モジュールが簡単に使えるようにしています。割り込みベクタが有効の場合、これも **Interrupt Manager** モジュールに追加されます。
既定値の `CCPx_Initialization()` 関数以外に、MCC は選択した機能に応じて各種 API を生成します。

- キャプチャ: `CCPx_IsCapturedDataReady()` は使いやすく、キャプチャ イベントをポーリングし、`CCPx_CaptureRead()` はパルス/イベント継続時間を返します。
- コンペア: `CCPx_CompareCountSet()` は必要なパルス出力幅を設定し、`CCPx_IsCompareComplete()` 関数はパルス生成シーケンスが完了したかどうかをポーリングします。
- PWM: `PWMx_LoadDutyValue()` 関数を使うと、10 ビットのデューティ サイクル レジスタを簡単に設定できます。

ピン配置

初期の PIC16F18xx および PIC16F19xx モデルでは、CCP/PWM モジュール用の入出力ピン選択が固定されていました。
PIC16F150x ファミリでは、全ての CCP-PWM ピンを CLC モジュールに再割り当てし、これを介してさらにより多くの I/O および周辺モジュールに再割り当てする事ができます。
最新ファミリ(例: PIC16F16xx、PIC16F17xx)にはペリフェラル ピンセレクト(PPS)機能が追加されており、これを使うと、使用可能な全てのデジタル I/O に CCP=PWM 信号を出力できます。

ホームワーク

- CCP または PWM モジュールを使ってサーボを制御する方法を調べましょう。
- PWM を使ってアナログ出力を生成する方法を調べ、DAC を使った場合と比較しましょう。
- 複数の PWM 出力が同じタイマを共有できる場合と、別々のタイマが必要になる場合を調べましょう。

オンライン リソース

『AN1175 - Sensorless Brushless DC Motor Control with PIC16』

『 AN1261 - Dimming Power LEDs Using a SEPIC Converter and MCP1631 - PWM Controller』

『AN1305 - PIC16FXXX を使用するセンサレス 3 相ブラシレスモータ制御』

『AN1562 - 高分解能 RGB LED の色調制御』

『AN594 - Using the CCP Module』

例

CCP または PWM モジュールを使ってサーボモータを制御する例を以下に示します。

```
/* Project:ADC to PWM Servo
 * Device:  PIC16F1509
 */

#include "mcc_generated_files/mcc.h"

#define TCLK   _XTAL_FREQ / 4
#define TPERIOD  (unsigned char)(TCLK/4 * 0.008)  // 125Hz period (8ms)
#define SERVO_MIDDLE    (unsigned)(TCLK * 0.0014) // 1.4ms
#define SERVO_MIN       (unsigned)(TCLK * 0.0004) // 0.4ms
#define SERVO_MAX       (unsigned)(TCLK * 0.0024) // 2.4ms

void main(void)
{
    // configure ADC to trigger from Timer2 and generate an interrupt
    // configure PWM1 for an 8ms period
    SYSTEM_Initialize();

    // Enable Interrupts
    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();

    while (1)
    {
    }
}
```

```
/* edited in the adc.c file
*/
void ADC_ISR( void)
{ // read potentiometer value and translate to servo angle
   uint16_t duty;

   duty = SERVO_MIN + ( ADC_GetConversion( Potentiometer));
   if ( duty > SERVO_MAX)
      duty = SERVO_MAX;
   PWM1_LoadDutyValue( duty);

   // Clear the ADC interrupt flag
   PIR1bits.ADIF = 0;
}
```

CWG – 相補波形ジェネレータ
COG – 相補出力ジェネレータ

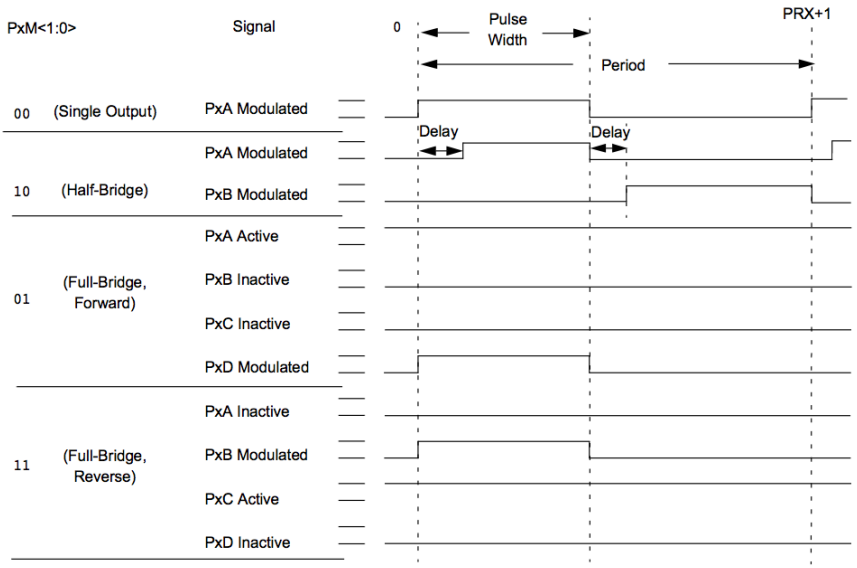


図 2.7: 相補出力

概要

多くのモータ制御アプリケーションでは、一対の MOSFET パワー トランジスタを駆動するために、PWM 出力信号を 2 つの相補波形に分割し、図 2.7 のようにデッドバンド遅延を追加する必要がありますが、これは排他的な関係ではありません。電源および照明では同じ「サービス」か恩恵を受けるアプリケーションが多くあります。これを受けて、PIC16F1 で ECCP モジュールのバックエンドにあったモジュールを独立した単体のモジュールとしたのが相補波形ジェネレータ(CWG)です。さらに、より高度な相補出力ジェネレータ(COG)となりました。

動作原理

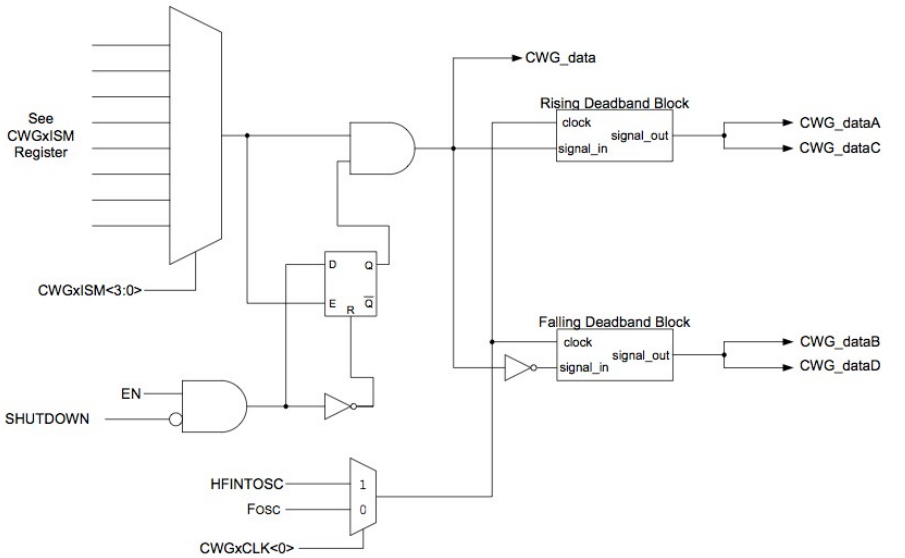


図 2.8: CWG の概略ブロック図(ハーフブリッジ)

図 2.8 に示すように、CWG モジュールは代表的な ECC バックエンド以上の事を実行しますが、最も重要なのは、大きな入力選択マルチプレクサ(ISM)を使って、任意の数の他の信号生成モジュールと接続できる点です。10/16 ビット PWM はもとより、NCO、高速コンパレータ、CLC ブロック等と接続できます。つまり CWG は設計者にとってとても便利な多目的ツールです。言い換えれば、「全体は部分の総和にまさる」という事です。
CWG モジュールで実行できる機能は以下の通りです。

- 立ち上がり/立ち下がりエッジで別々の(デッドバンド)遅延(N チャンネルと P チャンネルの MOSFET デバイスのサイズの違い(およびゲート静電容量)に合わせて調整できるため、回路効率を最適化可能)
- 自動シャットダウン制御
- 出力ステアリング制御
- ハーフブリッジおよびフルブリッジ出力モード
- プッシュプル出力モード

- イベント入力は独立しているため、スリープ中でも動作可能 (Note: HFIntOSC を COG のメインクロックとして使う場合、CPU がスリープ中でもオシレータは有効のまま)

さらに、COG モジュールは以下の機能を実行できます。

- 入力イベントに「ノイズが多い」場合、入力ブランキング カウンタによって余分の複数整流動作を防ぐ。
- 位相遅延を追加して制御システムを安定させる。
- 立ち上がり/立ち下がりイベント入力で(複数の)別々の入力を受け入れる。
- レベルまたはエッジセンス入力を受け入れる。
- 通常の同期カウンタの代わりに非同期デッドバンド遅延チェーンを提供する事で、きわめて細かい粒度(5 ns)を実現する。

アプリケーション

CWG および COG モジュールを活用できる代表的なアプリケーションを挙げます。

- 一部の電源制御アプリケーションでは、プッシュプル トポロジが使われます。
- ハーフブリッジおよびフルブリッジ モードは、方向制御オプション付きのブラシ付き DC モータを制御するために使われます。
- 立ち上がり/立ち下がりエッジイベントの入力として接続した高速コンパレータは、電源(スイッチング)アプリケーション(ヒステリシス制御、ピーク電流モード等)でよく使われます。
- ハーフブリッジ ドライバに接続した NCO 出力は、高輝度放電灯と調光可能な蛍光灯バラストの設計に使われます。

制限事項

内蔵の自動シャットダウン、ブランキング、複合出力モードを使うと、CWG/COG モジュールで電源およびモータ制御アプリケーションの設計を大幅に簡略化できます。各サイクルでの CPU サポートが不要になるため、高速割り込み応答またはその他の CPU 負荷の高い動作も不要です。アプリケーションにおける主な制限事項は、最大 PWM クロック源がデバイスの最大クロック速度(通常 32 MHz)の制約を受ける点です。それでも、電源制御

アプリケーションが 400 kHz を超えるスイッチング周波数を達成しながら、使用可能な CPU 性能のごく一部しか使っていない事はよくあります。
より高い分解能が必要な場合、PSMC モジュールの使用を検討すべきです。PSMC モジュールは、本書の執筆時点では PIC16F178x ファミリのみ内蔵しており、専用の 64 MHz オシレータを使って PWM 動作が可能です。

MCC が生成する API

MCC は CWG/COG の全オプションを、出力ピン設定、イベント制御、自動シャットダウン、ステアリング制御の 4 つのウィンドウに便宜的に分類しています。
MCC は通常の CWGx_Initialize()以外に、以下の 3 つの関数グループを含む最小限の API を生成します。

- 個々のデッドバンド カウンタを最適化する関数
- 自動シャットダウン イベントをセットまたはクリアする関数
- 新しい(全ての)設定を一度に読み込み、モード切り換えを同期する関数を使って、高消費電力 MOSFET デバイスの駆動時に危険な競合を回避します。

ピン配置

新しいデバイスファミリは全てペリフェラル ピンセレクト機能を備えているため、きわめて柔軟に最適なピン割り当てが可能です。これに対する唯一の例外は、駆動能力の高い専用パッドの特長を活かす設計とする場合です(詳細は第 3 章「HIDRV – 100mA」セクション参照)。

ホームワーク

- CWG/COG デッドバンド制御を使って、連続したイベントに自動的に短い遅延を発生させる方法を調べましょう。
- CWG/COG を使って入力信号の周波数を 2 倍にする方法を調べましょう。

オンライン リソース

『TB3118 – 相補波形ジェネレータの技術概要』
『TB3119 – 相補出力ジェネレータの技術概要』
『TB3120 - PIC®マイクロコントローラの内蔵スロープ補償器』
『AN1660 - 8 ビット PIC16 マイクロコントローラを使った単相/多相 AC 誘導モータ向け低コスト回路の設計と解析』
『AN1779 - PIC16F1613 を使ったセンサ付き単相 BLDC モータドライバ』

Core Independent

NCO - 数値制御オシレータ

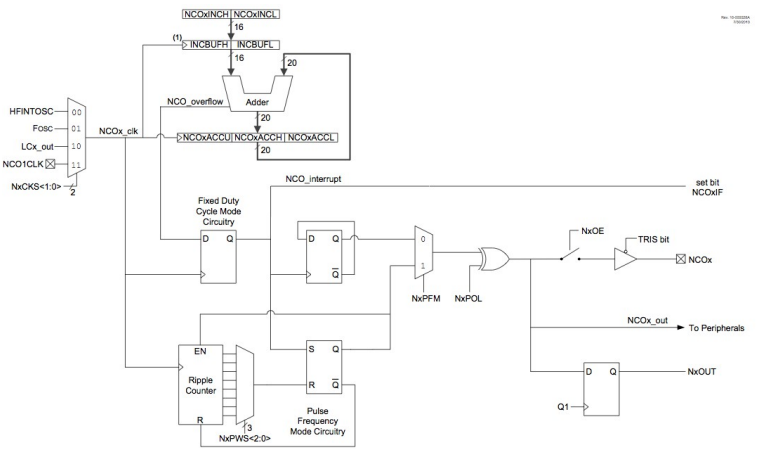


図 2.9: NCO モジュールのブロック図

概要

数値制御オシレータは、タイマ/カウンタの概念に新しい切り口をもたらします。通常のタイマはクロックパルスごとにカウントを 1 つインクリメントしますが、NCO は任意のステップ数のインクリメントを実行できます。図 2.9 では、V 字状の加算器ブロックが容易に見つかります。NCO 以外のタイマは周期一致または最大カウントに達すると、リセットしてゼロからカウントを開始します。一方 NCO は、出力パルスの生成に使われるキャリービットを生成し、剰余を使ってカウントを続行します。この違いによる影響は一見分かりにくいかもしれませんが、非常に重要な意味合いがあり、多数の電源(特に照明)アプリケーションにとって、NCO モジュールを非常に魅力的なものにしています。

動作原理

NCO は全ての周期で同じ値(ゼロ)からカウントを開始する訳ではなく、直前の周期の剰余をベースにしてカウントを続けるため、一部の周期が他より(1 クロックティック)短くなります。このため、NCO の周波数出力は一定にはならず、サイクルごとに近い 2 つの値の間を揺れ動きます。短いサイクルと長いサイクルでの分布を平均すると非常に高精度な値になり、入力クロックの正確な比に素早く収束する事を、以下の計算式から証明できます。

$$F_{out} = F_{in} * Increment / 2^{20}$$

計算式 2.- NCO の平均出力周波数

インクリメント レジスタの値が分数の分子であるため、この値と結果の出力周波数の値(Fout)の間の**関係は線形**であると推定できます。これは、NCO と他のタイマ/カウンタとの大きな違いです。

NCO の機構を詳しく調べると、特に出力周波数が非常に高い(インクリメント値が大きい)場合、レジスタ値が増加するたびに、出力周波数は比例してごくわずか増加する事が分かります(インクリメント値が大きいほど「1/インクリメント」の値は小さくなるため)。言い換えると、**NCO の周波数分解能が最高になるのは、出力周波数が最も高い時**です。これは、出力周波数が最大になる時に周波数制御が最も粗くなる従来のタイマとは反対の挙動です。

NCO について最後の注目点です。NCO が生成する出力はキャリービットだけで、出力ピンのトグルまたは固定長シングルパルスの生成に使用されるため、**出力デューティ サイクルは基本的に周波数とは関係のない固定値です**。これもまた、従来のタイマ/PWM モジュールとは対照的です。従来のタイマ/PWM モジュールで周波数の変化に関係なく 50%の出力または一定の DC を維持するには、CPU のサポートが必要です。

最後に、特別なインクリメント値を取るケースについて見てみましょう。

- インクリメント値が 1 の場合、NCO は通常のカウンタというだけでなく、非常に大きいカウンタ(20 ビット)です。
- インクリメント レジスタが 2 の累乗(2^N)に設定されている場合、入力クロックがプリスケーラ/分周器を通過した従来のタイマの結果と同じであり、その結果生成される周波数は $Fout = Fin * 2^{(N-20)}$ です。

このように特殊なケースでは NCO は従来型のタイマと同じ挙動を示すため、NCO 特有の性質が不要の場合、使う必要はないかもしれません。

他の CIP でも同様ですが、CPU 負荷を許容できるのであれば NCO を使わずに従来型のタイマモジュールとソフトウェアでも同じ事を実行できます。ただし、NCO なら CPU 負荷を取り除く事ができます。さらに、NCO はその高い汎用性が実証されており、より複雑な新しいカスタム モジュールを構築するために、CLC ブロックおよび従来型のタイマと組み合わせて使われてきました。

アプリケーション

表 21に示すように、従来型のタイマ/PWM モジュールと NCO との違いをまとめると、これら2 つのモジュールがほぼ正反対の性質を持つ事が分かります。

	NCO	PWM
周波数	線形制御	固定
デューティ サイクル	固定	線形制御
周波数出力	線形	固定
Fmax での分解能	最大	最小
出力ジッタ	1 ティック	なし

表 2.1- NCO とタイマ/PWM の比較

システムに慣性が存在するアプリケーションでは、それが機械的慣性、出力回路の静電容量、回路の共振のいずれであっても、NCO が有利です。いずれの場合でも、NCO の高分解能と線形応答が単純で低コストなソリューションを提供する事で、複雑な制御の問題を解決します。

特に、多くの電源制御アプリケーションで NCO の使用が有利である事が実証されています。HID ランプのバラストや調光機能付き蛍光灯等では、(点弧後の)共振回路の応答曲線に沿って高い(周波数)分解能で下方に移動する事で、回路電流を非常に正確に制御できます。

制限事項

最初の NCO モジュールが備えていたのは 20 ビット加算器と 16 ビットインクリメントであり、最大出力周波数が入力クロックに対して 1:16 に制限されていました。

より最近の製品(例: PIC16F171x ファミリ)では、インクリメント レジスタのサイズが 20 ビットになっており、最大比率 1:1 を達成できるため、出力周波数レンジが大幅に拡大しています。

MCC が生成する API

NCO に対する MCC のサポートは、主に正しいインクリメント値を計算し、特定のクロック入力で達成できる出力周波数レンジを見積もる部分に関係します。
生成された **nco1.c** ソースファイル内の最小限の API には、既定値の `NCOx_Initialize()` 関数しかありません。割り込みが無効の場合、ステータス ポーリング マクロの `NOCx_GetOutputStatus()` が加わります。

ピン配置

初期の PIC16F150x モデルでは NCO 外部入力(NCO1CLK)用に決まった位置(RA5)の 1 ピンしか提供しておらず、20 ピンデバイスでは NCO 出力用に最大 2 つの代替ピン(RC1 と RC6)を提供していました。
より最近の製品はペリフェラル ピンセレクト機能を備えており、任意のデジタル I/O ピンを使う事ができます。

ホームワーク

- 高周波出力と低周波出力で NCO の分解能を比較してみましょう。
- NCO と PWM の性能が逆転する個所を調べてみましょう。
- 周期ジッタが NCO ノイズシグネチャに及ぼす影響を調べてみましょう。従来型の PWM と比較してみましょう。

オンライン リソース

http://microchip.com/nco - CIP - NCO の概要
『TB3071 - 線形周波数出力の電圧制御オシレータ』
『TB3097 - PIC12F1501 の NCO モジュールを使ったデジタル SMPS(降圧型コンバータ)』
『TB3102 - PIC12F1501 の NCO モジュールを使った昇圧型コンバータ』
『AN1050 - A Technique to Increase the Frequency Resolution of PWM Modules』
『AN1470 - CLC と NCO を使ったマンチェスタ デコーダ』
『AN1476 - CLC と NCO を組み合わせた高分解能 PWM の実装』
『AN1523 - 数値制御オシレータモジュールを使った正弦波ジェネレータ』
http://www.sebulli.com/picrx – Radio Receiver using the PCI16F1713 NCO

HLT - ハードウェア リミットタイマ

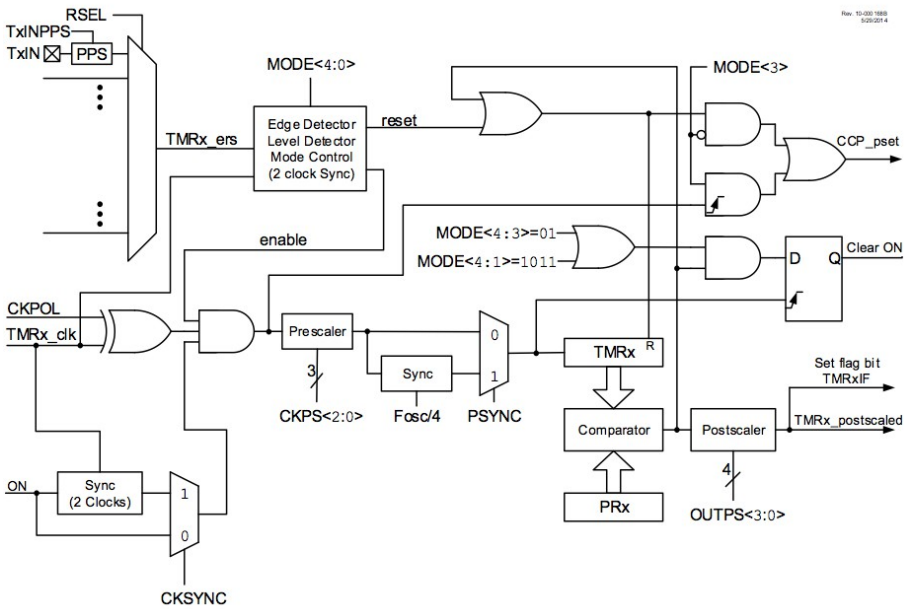


図 2.10: ハードウェア リミットを使った 8 ビット(偶数)タイマのブロック図

概要

過去 25 年間に PIC マイクロコントローラで使われた 8/16 ビット標準タイマが持つ機能には、ハードウェア外部イベントに応じてカウントを開始/停止/リセットするという基本的な機能が欠落していました。もちろん、これらの機能は全てソフトウェアでも実装できます。その場合、CPU を使って周期イベントに応答し、最終的に**手動**でタイマをクリア、有効/無効にします。しかし CIP の趣旨に従うと、このような操作で CPU に負荷をかけるべきではなく、対応する制御入力/イベントは、チップ内部のその他の周辺モジュールに簡単に直接接続できる必要があります。この要求に対する簡単な答えとして、ハードウェア リミットタイマが開発されました。ハードウェア リミットタイマは事実上大幅に機能拡張された 8 ビット(イベント)タイマと言えます。

図 2.10 と図 2.4 を比べても、2 つのタイマが非常に近い関係にあるとは明白ではないかもしれません。

ブロック図下部に注目するとプリスケーラ ブロックに気付くでしょう。続いて、タイマ/コンパレータ/周期レジスタ構造、さらにポストスケーラがあります。
ブロック図上部の新しい構造は、簡単に 2 つの部分に分ける事ができます。左側にある入力信号マルチプレクサ(ISM)を使うと、**タイマリセット**信号を多数提供できます。右上の部分は、タイマ操作を周期的にするか**ワンショット**にするか制御するかの選択肢を提供します。

動作原理

まず第一に、HLT は 8 ビットタイマです。このため、CCP-PWM モジュール動作への関与を含む周期イベントを生成する、標準のプリスケーラおよびポストスケーラの選択肢を備えています。
HLT は標準 8 ビットタイマ以外に、以下の機能を備えています。

- 選択肢が大幅に増えた参照クロック入力(各種オシレータ、CLC 出力、ゼロクロス検出イベント、外部ピンを含む)
- ワンショット動作(単一周期が完了するとリセットして即座に再開するのではなく自動停止する機能)
- 最も重要な**タイマ リセット イベント**入力(設定された周期/パルス幅に達する前にタイマカウントをリセット可能)
- リセット/開始/停止モード(選択したエッジが検出されるまで、リセット後も同じタイマリセット イベントがカウンタを保持)

アプリケーション

ハードウェア リミットタイマの使い方が良く分かる例を以下に挙げます。マイクロコントローラが周期的にある動作を開始し、短い遅延後にセンサ入力によって確認されるアプリケーションを想定しましょう。ここで、センサ入力が欠損した場合、または信号入力が遅過ぎる状況を考えてみましょう。これはフォルト、または安全性リスクをもたらしかねない例外イベントを意味します。従来のアプローチでは、適切な**制限時間**を持つタイマを設定する必要があるでしょう。タイマがこの制限時間に達すると、予防措置を取るよう MCU に要求する割り込みが必要です。しかし、(同時に複数の割り込みが有効である場合)割り込み応答時間は予測不可能であり、割り込みレイテンシがひどく長くなる(ns オーダーが必要な時に数十 ms)可能性があります。また、少数の非同期イベントが関係するだけで、たいていソフトウェアが複雑になりデバッグが手に負えなくなります。このようなアプリ

ケーションの例として、ホールセンサによる速度フィードバックを備えたモータ制御システムと、ピーク電流フィードバックでインダクタの電流サージを計測するスイッチング レギュレータ(電源)が挙げられます。
HLT なら、タイマ周期に達する**前に**タイマリセット入力を受信しなかったら出力を自動的に有効にできます。こうする事で、この問題を簡単に解決できます。HLT 出力を他の周辺モジュールの入力(PWM 自動シャットダウン等)か、安全性スイッチを直接駆動するピンに直接接続できます。自律、高速応答、CPU 負荷の軽減という CIP の理念から見た場合、これは非常に重要です。

制限事項

結局のところ HLT は 8 ビットタイマに過ぎないため、適用範囲の限界は分解能によって決まります。
16 ビット以上の粒度で長めのタイムアウト時間が求められる場合、代わりに SMT(信号計測タイマ)を検討すると良いでしょう。
外部イベントへの応答(再トリガ可能なワンショット)として非常に短いパルス(数十 ns)を生成する必要がある場合、COG の非同期デッドバンド制御を検討してください。

MCC が生成する API

MCC は、HLT を特殊な 8 ビットタイマとしてしか扱いません。このため、使用可能な周辺モジュールのリストに HLT は含まれません。(HLT を備えたデバイスで)8 ビットタイマを有効にすると、対応するダイアログ ウィンドウには非常に豊富な参照クロックオプション(クロック ISM)が含まれ、リセット要因と開始/リセットオプションの両方の選択ボックス(ISM)が追加されている事が分かります。
生成されるファイル名は **tmr2.c**(または別の偶数)となり、API は以下のような数種類の新しい関数のみを含みます。

- `TMRx_ModeSet()` - HLT モードを動的に変更できます。
- `TMRx_ExtResetSourceSet()` - タイマリセット入力源を動的に変更できます。

ピン配置

ペリフェラル ピンセレクトを備えた最近のデバイスはほぼ例外なく HLT を実装しているため、任意のデジタルピンにタイマリセット入力を割り当てる事ができます。唯一の例外として、PIC16F1612/3 モデルは HLT を実装していません。

ホームワーク

- HLT の挙動を WDT モジュールと比較してみましょう。
- HLT の挙動をウィンドウ計測モードでの SMT と比較してみましょう。
- 短いパルスが必要な時に HLT(シングルショット モード)を使った場合と CWG/COG を使った場合を比較してみましょう。

オンライン リソース

『TB3122 - PIC®マイクロコントローラの内蔵ハードウェア リミットタイマ』

SMT - 信号計測タイマ

概要

SMT(信号計測タイマ)は、PIC16F1 ファミリの CIP に導入されたタイミング モジュールの中で、おそらく最も魅力的なものです。ブロック図(図 2.11)は、一見手強い印象を与えるかもしれませんが、この後説明する各モードを概観する事を強く推奨します。読者は、多くのモードがこれまでにはなかった、将来のアプリケーションで便利だと気付くでしょう。

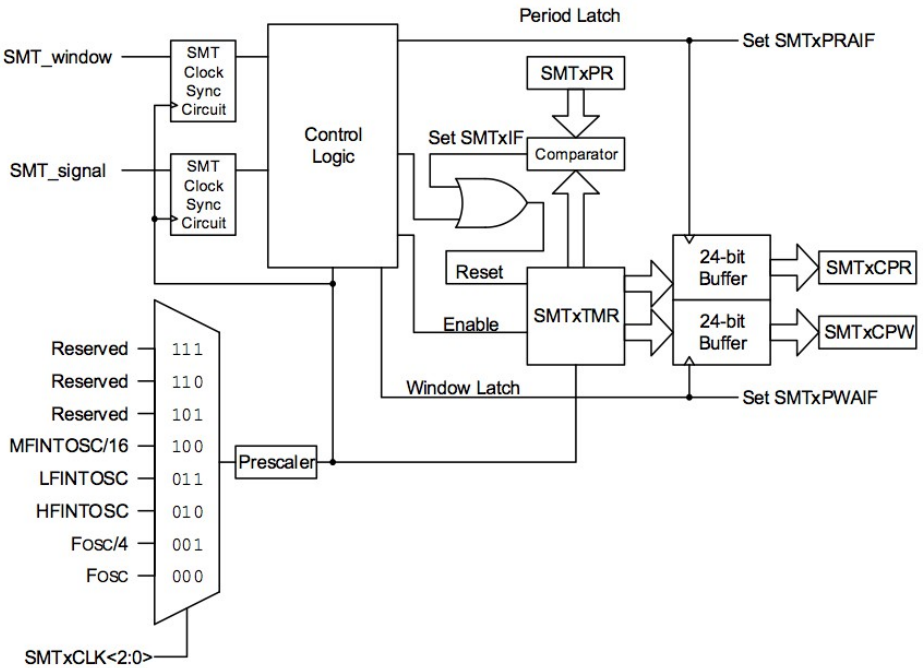


図 2.11: 信号計測タイマのブロック図

動作原理

SMT は基本的に 3 つの入力源(それぞれが ISM を持つ)と 2 つの 24 ビット出力レジスタ(**measurement**)を備えた 24 ビットタイマです。(24 ビット)周期レジスタと、**出力の一致またはタイムアウト** イベントを生成するコンパレータを備えています。3 つの入力**を計測入力(信号)、ゲーティング(ウィンドウ)入力、参照入力(クロック)**のいずれかとして使う場合の

順列から、合計 11 種類のモードを使えます。ここでは、それぞれのモードについて確認し、使い方を説明します。

周期/デューティ サイクル計測モード

このモードでは、SMT は入力信号と参照クロックのみを使って、入力信号周期とデューティ サイクル(Ton)をサイクルごとに計測します。ダブルバッファ方式により、連続 2 周期の計測値が出力レジスタに格納されます。
従来型のタイマと割り込みリソースを使っても同様の計測が可能ですが (アプリケーションノート AN1473 参照)、多くの CPU サイクルが必要である、ステートマシンの作成に RAM とフラッシュメモリを使う、計測精度または信号入力特性が大きく制限される、といった問題があります。
SMT はコアからの独立という考え方に従ってこれらの制限と CPU のタイミング制約を全て解消し、便利な方法で正確に入力信号を計測します。
このようなアプリケーションにはサイクルごとの PWM 入力信号のデコードがあります。これは車載アプリケーション、煙検出器、ファン制御アプリケーションで一般的です。

High および Low 時間計測モード

これは上記のモードと基本的に同じですが、このモードでは立ち下がりエッジでカウントがリセットされます。その結果、Ton 時間と Toff 時間が別々に計測されます。上記の周期/デューティ サイクル計測モードの留意事項とアプリケーションの全てが、このモードにもあてはまります。

ゲートおよびウィンドウ付きモード - 平均 DC 計測

このモードでは、window 信号入力によって制御される間隔に対して入力信号の ON 時間を計測します(クロックのゲート制御)。ウィンドウ周期ごとに、2 つの出力レジスタのどちらかに新たな積算 ON 時間をキャプチャします。
その結果、入力信号の Ton(ウィンドウ周期の**平均)**が計測されます。言い換えると、入力の**平均デューティ サイクル**計測値が得られます。これは、入力信号に大量のノイズ(またはジッタ)が含まれ、デューティ サイクルの瞬時値よりも平均計測値が適当なアプリケーションで便利です。

Time of Flight モード

このモードでは、**signal** と **window** という 2 つの信号入力を使ってそれぞれ参照クロックのカウントを開始および停止します。これは、2 つのイベントの間隔を正確に計測する必要のあるアプリケーションで便利です。このモードを使い慣れると、どうしてこれまで使わなかったのか不思議に思う事でしょう。 SMT を使わない場合、状態変化割り込み機能とステートマシンを使い、さらに多くの CPU サイクルを浪費していたはずです。SMT のこのモードを使えば簡単です。

タイマモード

タイマモードは最も基本的なモードの 1 つです。このモードでは SMT クロック入力のみ使いますが、大きい周期レジスタをと一致イベントを使って長いタイムベースを生成します。本質的にこの挙動は 8/16 ビットタイマと似ていますが、24 ビットのダイナミック レンジにより、より長い周期またはより高い分解能を達成できる点で異なっています。

ウィンドウ計測 - 大型の HLT

ウィンドウ計測モードでは、**window** 信号入力を使って各立ち上がりエッジのカウントを**キャプチャ**し(極性反転可能)、即座にカウントをリセットします。これにより、window 信号入力の周期をサイクルごとに計測できます。しかし、この場合も周期レジスタとランニング カウントが常時比較されているため、ウィンドウ入力信号の周期が長くなり過ぎた場合はもう 1 つの出力イベント (タイムアウト)も取得できます。別の方法として、ハードウェア リミットタイマ モジュールの挙動と比較する方法があります。window 信号入力は基本的に HLT のリセットイベントであり、周期レジスタはタイムアウト周期になります。ウィンドウ計測の大きな利点として、24 ビットのダイナミックレンジによりタイムアウトを長くするか、分解能を高くする事ができます。

ゲート付きタイマモード

このモードは 16 ビットタイマのゲート付きモードと同じです。参照クロックのゲート制御に入力信号を使いますが、このモードでは分解能が 24 ビットです。

キャプチャモード

このモードは、16 ビットタイマモジュールをキャプチャモードの CCP モジュールと組み合わせて使った場合と同じです。このモードでも分解能が高くなり、計測はダブルバッファ構成です。

非同期カウンタモード

非同期カウンタモードにはカウンタ、ゲート付きカウンタ、ウィンドウ付きカウンタの 3 種類があります。これらは参照(クロック)入力を使わず、単純に **signal** 信号入力を使ってパルスをカウントし、最終的に window 信号入力を使ってゲート制御を実施します。16 ビットタイマの場合と同様ですが、このモードではダイナミック レンジが大きいか、分解能が高くなります。

MCC が生成する API

MCC は SMT を CCP モジュールと同様に扱い、3 つ(キャプチャ、カウンタ、タイマ)にモードを分類します。各カテゴリが別々のダイアログ ウィンドウに対応しており、これを使って 11 モード全てを設定できます。

ピン配置

ペリフェラル ピンセレクトを備えた最近のデバイスはほぼ例外なく HLT を実装しているため、任意のデジタルピンにウィンドウ入力、信号入力、クロック入力を割り当てられます。唯一の例外として、PIC12F1612 (8 ピン)および PIC16F1613 (14 ピン)モデルは HLT を実装していません。

ホームワーク

- 8 または 16 ビットの分解能しか必要ではない場合、どのように SMT を使うべきか考えてみましょう。
- 従来型のタイマ構造を(場合によっては CLC も)使って SMT の Time of Flight モードと同じ機能を実装するにはどうしたら良いか考えてみましょう。

オンライン リソース

『AN1473 - パルスおよびデューティ サイクル計算の各種ソリューション』
『AN1779 - PIC16F1613 を使ったセンサ付き単相 BLDC モータドライバ』

第 3 章 入出力

I/O ポート

概要

デジタル I/O はおそらく、どの組み込みアプリケーションにおいても最も基本的な構成要素でしょう。PIC マイクロコントローラが最も堅牢で信頼性も高い I/O 構造を備えている事は業界でも知られていましたが、それは最近の PIC16F1 デバイスでも同じです。I/O の堅牢性は 8 ビット アーキテクチャが提供する最大の利点の 1 つです。コアサイズが小さいため、PIC マイクロコントローラの設計者は成熟した CMOS プロセスを使って、デバイスコストを上げる事なく大電流(最大 100 mA、本章で後述する HIDRV 参照)を処理させる事ができます。しかし、最近の世代の PIC マイクロコントローラは I/O ポートの設計が大幅に柔軟性をもつようになり、こうした特長がアプリケーションの安全性を高めています。柔軟性によりアプリケーションの設計が簡単になる場合もあります。また、外付け部品点数と総コストが削減できる場合もあります。
8 ビット PIC マイクロコントローラでは、文字(A、B...)で**区別される 8 つの入力ポート**別に I/O が分類されています。

TRIS – 3 ステート制御

全ての I/O ポートは方向制御レジスタを備えています。PIC マイクロコントローラでは、この機能は TRISx レジスタ(TRISA、TRISB...)で制御されており、ビット値の意味は I – Input、O – Output と非常に覚えやくなっています。
これらのレジスタの役割は、I/O 出力の有効化/無効化です。つまり PIC マイクロコントローラでは、ピンを常に入力として使えるという事です。

PORT – ダイレクトピン アクセス

PORTx グループのレジスタは、入力ピンの(デジタル)値に**直接**アクセスするために使います。この場合の「直接」とは、I/O ドライバ構造の外部に存在する実際の電気的な値を意味します。この違いは重要です。なぜなら、ピンに高い負荷がかかっている場合、このピ

ンを出力として High(1)レベルで駆動していても Low(0)が読み出される場合があるためです。これをうまく使うと、同様の状況を検出する、または 2 つのデバイスが同じ外部リソースを制御しようとする可能性がある場合に競合を検出できます。
出力ラッチに値を書き込むと、それにより PORTx レジスタへも書き込む事ができます。

LAT - 出力ラッチ

LATx レジスタは、(PIC18 アーキテクチャを採用した)全ての PIC16F1 デバイスが備えており、出力ラッチへのアクセスを提供します。
ここでも、ラッチに書き込まれた直近の値を読み取ると、それにより LATx レジスタを読み出す事ができます。
I/O ポートへの全ての書き込み動作に LAT レジスタを使う事を推奨します。

ANSEL – アナログ選択

アナログ機能とデジタル機能を一緒に使う事が常にうまくいくとは限りません。(ADC を使って)ピンからアナログ値を読み出すのと同時に、(対応する PORT レジスタから)デジタル値を読み出す事は可能ですが、予期しない結果を招く可能性があります。入力電圧が Vdd/2 に近づくと、COMS 入力ゲートの両側が部分的にバイアスされ、大電流が電源から吸い込まれる場合があります。正しい設計手法では、デジタル入力を切り離して接地する事により、このような状況を防止します。その時に使うのが ANSELx レジスタです。ビットをセットすると、対応するポートピンがアナログ使用専用に設定されます。

WPU – 弱プルアップ

ほとんどの I/O ピンは弱プルアップ回路を備えているため、外付け抵抗のコストとスペースを節約できます。WPUx レジスタで対応するビットをセットすると、個別(ピンごと)に有効にできます。

ODCON – オープンドレイン制御

最新の PIC16F1 ファミリでは、ODCONx レジスタを設定する事でオープンドレイン駆動を**シミュレート**する事ができます。従来はソフトウェアでシミュレートする事が多く、出力ラッチをゼロに設定して TRIS レジスタをトグルしていました。CPU に負荷をかけずに周辺モジュールで出力を直接駆動できるようにするため、CIP の導入に伴ってこの機能がハードウェアに実装されている事が重要です。ただし、これは本当のオープンドレイン回路ではないため、Vdd を超える電圧では使えません。

SLRCON - スルーレート制御

New

長いPCBトレースが意図せずアンテナになる可能性がある場合、I/O 駆動堅牢性が問題になります。最新の PIC16F1 モデルは、ノイズ エミッションを低減するスルーレート制御レジスタを備えています。

INLV – 入力レベル

一部のアプリケーションでは、ノイズ環境下での誤った入力認識を防止するために、マイクロコントローラのデジタル入力にヒステリシスが必要です。このような場合、INLVx レジスタで対応するビットがセットされていると、選択したポートで TTL からシュミットトリガ(ST)モードに入力を切り換える事ができます。

IOC – 状態変化割り込み

全ポート上の全ピンは、立ち上がり/立ち下がりエッジが検出されたら割り込みを生成するように設定できます。この挙動は各ポート(IOCxP か IOCxN)に関連付けられたレジスタペアで制御できます。
割り込みイベントを生成するピンの数に制限はありません。
他の割り込み要因と同様、IOC を使ってプロセッサをスリープから**復帰**させる事ができます。IOC モジュール割り込みが有効になっている場合(IOCIE)、実際の割り込みシーケンスが開始され、そうでない場合、プロセッサは Sleep コマンド直後の命令から実行を継続します。

HIDRV – 100 mA

New

PIC16F1 マイクロコントローラでは周知の通り、大半または全ピンが最大 20 mA の電流駆動能力(シンクおよびソース)を持っていますが、最近のモデルは最大 100 mA 5 V を駆動できます。これは最も低機能なトライアック(第 4 象限)、場合によっては小さいリレーをも十分に駆動できる電流です。これは 8 ビットテクノロジの強みの 1 つの理由です。
この機能は PIC16F161X ファミリに搭載され、COG モジュールの 2 本の出力ピン(RC4、RC5)に割り当てられています。

オンライン リソース

『TB3009 - Common 8-Bit PIC® Microcontroller I/O Pin Issues』
『TB3013 - Using the ESD Parasitic Diodes on Mixed Signal Microcontrollers』
『TB3061 - Interrupt-on-Change Operation for Mid-Range Microcontrollers』

『AN1081 - Interfacing a 4x4 Matrix Keypad with an 8-Bit GPIO Expander』

CLC - 構成可能なロジックセル

概要

CLC(構成可能なロジックセル)を PIC16F150x ファミリに導入した事は、おそらく CIP 理念の発展において決定的な瞬間でした。ハードウェア ブロックによる CPU 負荷の軽減という考えを初めて実現したのは CLC です。従来型のマイクロコントローラとプログラマブル ロジック デバイスの隙間を埋めようという試みはこれまでに何度も試みられてきましたが、両者のバランスを完全に取ったのは CLC が初めてです。Sea of Gates(SOG 型=海のようにトランジスタを敷き詰めた構造)に対して、当初 Puddle of Gates (Puddle=水たまり)という呼び方を使ったその控えめな姿勢に、成功の鍵が隠されていたと著者は考えます。

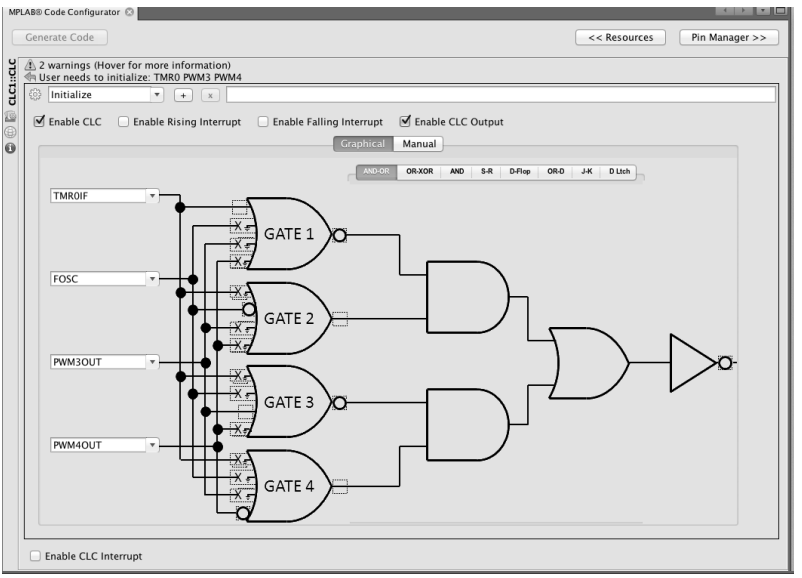


図 3.1: MPLAB Code Configurator の CLC ダイアログ ウィンドウ

動作原理

CLC は、あたかもプログラマブル ロジック アレイから取り出したかのような単一のマクロブロックです。最新世代の PIC16F1 マイクロコントローラが内蔵している CLC は最大 4 つです。完全に新しい周辺モジュールを構成するには不十分ですが、既存の周辺モジュール

を(チップ内部のグルーロジックとして)接続し、小さいイベントチェーンを有効にするには十分な数です。
どの CLC も同じ構造を持っています。入力信号マルチプレクサ (ISM)が 4 個、それぞれの入力から組み合わせ段に供給、最終的に 8 つ(AND-OR、OR-XOR、AND、S-R ラッチ、D フロップ、OR-D フロップ、J-K フロップ、D ラッチ)から選ばれたロジック機能に到達します。
入力マルチプレクサは外部からの信号を供給しますが、最も重要な点はタイマ、クロック、割り込みイベント等、CLC 周りの各種内部信号にアクセスできる事です。
次に、比較的複雑なステートマシンを構築できるように、各 CLC の出力を入力セレクタに供給します。1 つの出力を他の周辺モジュールのトリガとして使い、割り込みを生成し、出力ピンを直接駆動する事ができます。
CLC が非常に便利で柔軟なのは、以下の 3 つの重要な性質を持つためです。

- CLC はマイクロコントローラ内の他の部分と完全に非同期です。デバイスがスリープに移行し、コアおよび全ての内蔵オシレータが停止しても、CLC は通常動作を継続します。
- CLC は高速です。ロジック伝播時間は数十 ns レベルです。また、CLC は超低消費電力です。1 つのセルを有効化して増える消費電力はごくわずかで(nA レベル)、計測できないほどです。
- CLC の動作は RAM 内で設定されるため、動的に構成を変更できます。

アプリケーション

上記 3 つの性質から、以下の重要な結果を推測できます。

- CLC ロジックは非同期かつ超低消費電力であるため、インテリジェントな復帰メカニズムを実現するのに最適です。多くの低消費電力(バッテリ駆動)アプリケーションは、ほとんどの時間をスリープで過ごすか、そうでなければ頻繁(定期的)に復帰し、センサステータスを確認して外部スティミュラスに応答する必要があります。そこで CLC を使って、本当に必要な場合だけ CPU を復帰させれば、消費電力をさらに最適化できます。
- CLC は高速なため、周辺モジュールに接続して出力を直接駆動できます。必要に応じて CPU をバイパスし、しばしばソフトウェアを複雑にするアプリケーションのタイミング制約を解消し、システムの応答性と総合的な安全性を高める事ができます。

- CLC は RAM 内に設定されるため、再構成できます。アプリケーションは各種ステート/モードに移行する際に CLC を再設定すると、各種タスクで CPU をサポートできます。

制限事項

内蔵された CLC の数が少ないため(本書執筆時点で最大 4 つ)、複雑になり過ぎる事はありませんが、より高度なアプリケーションでは潜在的な制約となるでしょう。
また、ロジックセルの伝播遅延が短い事もしばしば誤解されています。入力信号はチップ内部とロジックをきわめて高速に伝播しますが、一旦出力ピンに達すると、その他全ての I/O ピンの出力ドライバと同じ制約を受けます。出力ドライバは通常、最大デバイスクロック(16 または 32 MHz)に対して設計されています。すなわち、たとえ 10 ns の伝播遅延を達成できたとしても、100 MHz の矩形波はピンに出力できません。
さらに、全ての CLC 経路で伝播遅延が正確に一致する保証はありません。これに CLC が非同期で動作するという事実を組み合わせると、グリッチの可能性がある事にすぐに気付くでしょう。本格的なデジタル回路の設計者にとって驚くには当たらないとしても、初心者はかなり手こずる可能性があります。つまり、CLC は食卓の上の鋭いナイフのようなもので、扱いには注意が必要です。

MCC によるサポート

MPLAB Code Configurator は優れた CLC サポートを備えています。図 3.4 に示したように、CLC ダイアログ ウィンドウは構成可能なセルを非常にグラフィック表示し、初期化関数の定義を簡略化します。
MCC は CLC モジュールごとに独立したソースファイル(`clc1.c`、`clc2.c...`)を生成します。

ピン配置

PIC16F150x ファミリでは、各モジュールが正方形の QFN パッケージの異なる側面に出力ピンを持つように CLC が配置されています。これは、PPS のないデバイスで、CLC をある種の PPS として使えるように意図されたものでした。
新しい PIC16F1 はほとんど PPS 機能を備えているため、任意のデジタルピンを CLC 入力または出力として選択できます。

ホームワーク

- CLC を使って、多数の内部信号(および複雑なロジックの組み合わせ)から AD 変換をトリガしてみましょう。ADC モジュールに関する文書を参照して、この実施方法を確認してみましょう。
- CLC をマイクロコントローラのコアと非同期で動作させる事によって、異常を識別し、安全性アプリケーションで CLC を使ってみましょう。
- CLC 入力ゲートが未接続の場合、入力がフローティングになっているかどうか調べてみましょう。
- アプリケーションから CLC に信号を注入する方法を調べてみましょう(ヒント: 入力ゲートの出力極性制御)。

オンライン リソース

http://microchip.com/clc – CIP – CLC の概要
『DS41631 - 構成可能なロジックセル、ヒントとコツ』
『TB3096 - Pulse Code Modulated (PCM) Infrared Remote Control』
『AN1450 - 遅延ブロック/デバウンサ』
『AN1451 - 構成可能なロジックセル(CLC)を使ったグリッチの除去』
『AN1470 - CLC と NCO を使ったマンチェスタ デコーダ』
『AN1476 - CLC と NCO を組み合わせた高分解能 PWM の実装』
『AN1606 - PIC16F1509 の構成可能なロジックセル(CLC)を使った WS2811 LED ドライバインターフェイスの作成』
『AN1660 - 8 ビット PIC16 マイクロコントローラを使った単相/多相 AC 誘導モータ向け低コスト回路の設計と解析』

PPS - ペリフェラル ピンセレクト

概要

ペリフェラル ピンセレクトは、大型の 16 ビット マイクロコントローラに先駆的に最初から搭載され、続いて 32 ビット マイクロコントローラに搭載された機能で、デバイスが内蔵する多数の周辺モジュールを簡単にピンに割り当てる事を目的としています。しかし時と共に、最も小型の PIC16F1 デバイスでも同じ問題に直面し始めました。つまり、周辺モジュールの数に対してピンの数が少ないという事です。
PPS はこの問題を解決するために、かなり柔軟にデジタル I/O 機能をピンに割り当てられるようにしています。
2013 年以降に導入されたほとんど全ての PIC16F1 デバイスはこの新機能を備えており、この傾向は将来的にも続くと予想されています。

動作原理

PPS は上記の「魔法」を実現するため、マルチプレクサを 2 セット使います。

- 多数の内部周辺モジュールから 1 つだけを制御用に選択できるように、各ピンの出力ドライバに 1 つの(機能)マルチプレクサが割り当てられています。
- 入力を取得するピンを 1 つだけ選択できるように、各周辺モジュールの入力に 1 つの(ピン)マルチプレクサが割り当てられています。

こうする事で必然的に競合が防止されます。複数の周辺モジュールが同じ出力を制御する事や、複数の入力信号が同じ周辺モジュールに到達する事は不可能です。
PPS の設定は RAM 内に保存されるため、I/O 初期化時に設定する必要があります。この設定は、フラッシュメモリおよびデータ EEPROM で使われるものと同等のロック機構で保護されます。PPS 制御レジスタを変更するには、特別なロック解除シーケンスを実行する必要があります。さらに、デバイス コンフィグレーション ビットを(フラッシュ内で)設定して、ロックを恒久的とするか、またはデバイスをリセット(POR/BOR/MCLR)するまでの**ワンショット**とするかを指定できます。

アプリケーション

PPS は PIC マイクロコントローラの内部リソース使用を最適化するのに役立ちます。

また、PCB の複雑さを緩和し、場合によっては基板上の信号トレースに関する歪みを解消する事で、ビアの使用を解消または軽減します。さらに、基板の層数を減らしてコストを削減できます。
ピン配置の最適化とは、何よりも「ノイズの多い」(高速転流)信号と敏感な(ハイ インピーダンス アナログ)入力を離して配置する事であり、寄生容量を減らしトレース長を揃える事です。
さらに、PPS を使うと複数のピンでリソースを共有できます。例えば、交互に使われる 2 つのポート間でシリアル インターフェイスを共有できます。また、同じ UART を特定のアプリケーション フェイズ中はプリンタコネクタに、それ以外の時はディスプレイまたはデバッグコンソールを駆動する別のコネクタに使えます。
最後に、PPS を使って同一出力を複数のピンに供給できます。このため、複数の CMOS 出力ドライバを同時に接続する事で、(パッケージ/デバイスの絶対定格内で)自然かつ緩やかに負荷を共有し、事実上ファンアウトが増加します。

制限事項

デバイスのピンと周辺モジュールの数が増えると、PPS の設定が複雑化します。コストを抑制するため、8 ビット PIC マイクロコントローラでは通常、PPS を 20 ピン以下に制限しています。つまり、少ピンデバイス(8/14/20 ピン)では、全ての有効な機能をどのデジタルピンに割り当てる事もできます。しかし、28 ピンおよび 40 ピンデバイスでは、ピンを 2 つのグループに分割する事ができます。無限の力等というものは存在しないのです。
I²C インターフェイス(MSSP モジュール)の出力では、SDA および SCL にピンを選ぶ時には注意が必要です。元々のバス仕様により、標準 CMOS ポート仕様とは異なる特別な I/O ドライバが決まっており、特定の 2 本のピンのみ使えます。シリアル EEPROM や小型センサ等のほとんどのアプリケーションは通常この影響を受けず、PPS の機能を活用した柔軟な選択が可能です。
PPS が使えるのはデジタル周辺モジュール/入力だけです。アナログ モジュールには別のマルチプレクサが必要であり、ノイズとコストを抑制するために選択肢は限られています。

MCC が生成する API

MPLAB Code Configurator では、[Pin Manager]ウィンドウから簡単に PPS の設定が可能です。
設定が完了すると、MCC は `pin_manager.c` ソースファイル内の `PIN_MANAGER_Initialize()` 関数に適切な PPS コマンドを追加(およびロック)します。

ピン配置

PPS の威力は、図 3.2 に示した MCC の Pin Manager の例を見ると一目瞭然です。この例では PIC16F1619 で CWG モジュールを選択しています。電源ピン以外なら制約なく入出力ピンを割り当てる事ができます。

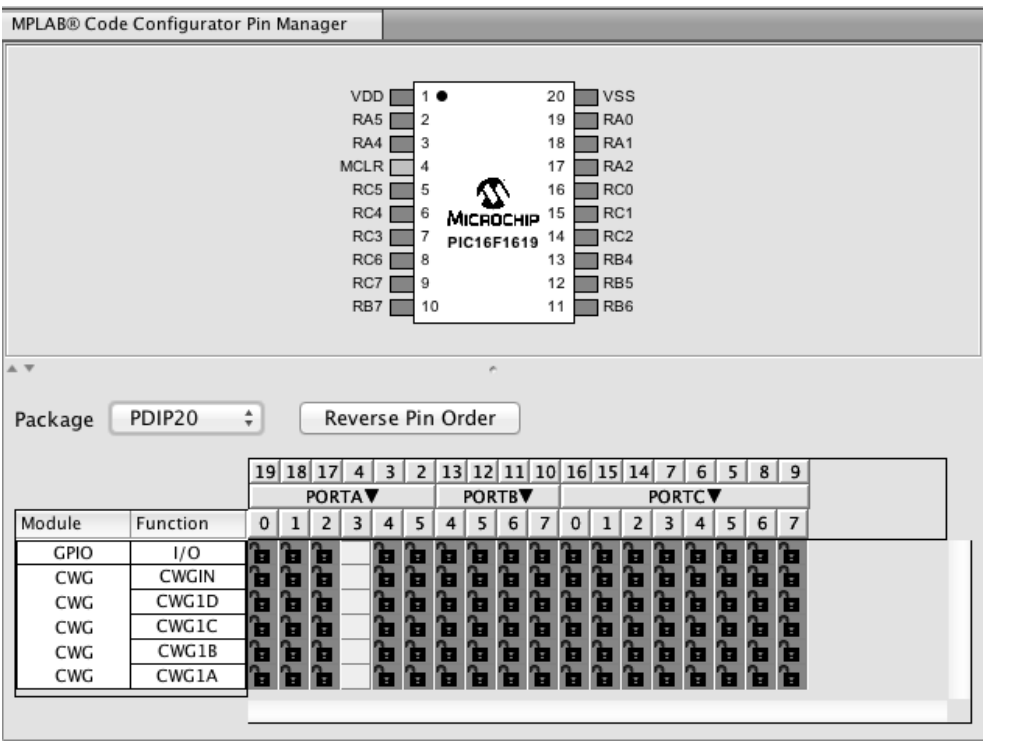


図 3.2: MCC の Pin Manager での PIC16F1619 の例

ホームワーク

- I/O ポートピンの割り当てを変更してみましょう。
- PPS がどの周辺モジュールで使えるか、また使えないかを調べてみましょう。
- PWM 出力ステアリングと PPS の違いを調べてみましょう。二重ステアリングについて調べてみましょう。
- ZCD 入力ピンの割り当てが変更できるか調べてみましょう。

- 複数のピンを 1 つの周辺モジュールへの出力として並列に接続する場合、最大ファンアウト(駆動電流)を制限する要因は何か調べてみましょう。

オンライン リソース

http://microchip.com/PPS – CIP – PPS の概要
『TB3096 - Pulse Code Modulated (PCM) Infrared Remote Control』
『TB3098 - PIC16F170X Peripheral Pin Select (PPS) Technical Brief』

DSM - データ信号モジュレータ

概要

データ信号モジュレータは基本的にデジタルミキサです。モジュレータ入力信号で 2 つのソース(搬送波)間の出力を切り換えます。モジュレータ信号が High の時、搬送波 High 入力は出力に接続されます。Low の時、搬送波 Low 入力が接続されます。搬送波信号は内部から(PWM モジュールまたはシステムクロック)、または外部ピンから取得できます。モジュレータ信号も内部的に生成できますが、各種信号源(コンパレータ出力、非同期入力源(UART TX)、同期シリアルポート(I^2C および SPI 等)または外部ピン 1 本で生成できます。モジュレータ信号を搬送波と同期させると、出力グリッチを回避できます。
この柔軟なメカニズムにより、FSK、PSK、OOK(2 つの搬送波のどちらかが接地している場合)を含む多数の変調方式に対応できます。

動作原理

図 3.3 に示すように、DSM はモジュレータ入力と 2 つの搬送波入力の AND 演算を実行しています。3 つの入力信号マルチプレクサ(ISM)がモジュール設計の大部分を占めており、これらがなければ非常に小さく単純な設計になります。

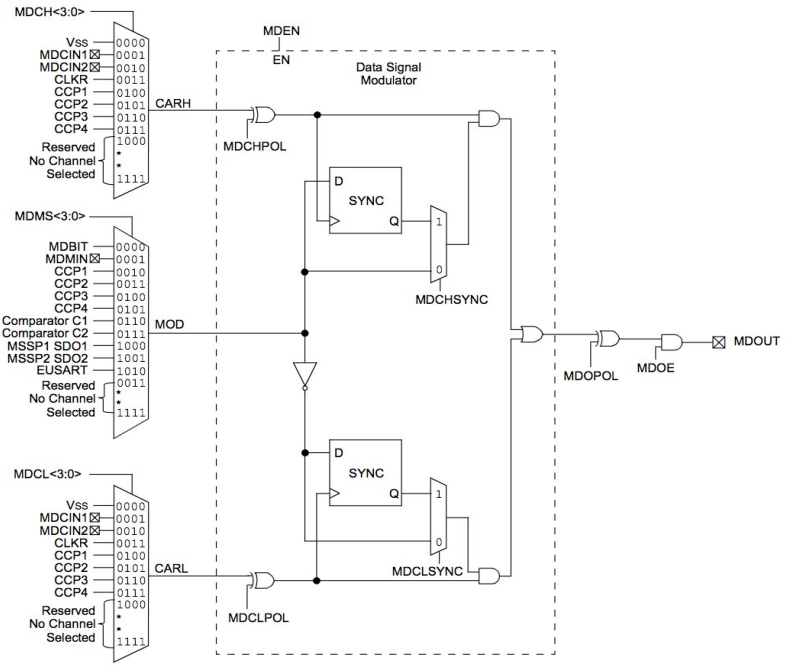


図 3.3 - DSM のブロック図

アプリケーション

DSM アプリケーションの例には、非同期シリアルポートの赤外線変調、半導体照明(高輝度 LED)機器の調光、無線信号の変調があります。

制限事項

DSM の使用は基本的に、個々のデバイスの I/O 出力ドライバの最大速度によって制限されます。32 MHz で動作する PIC16F1 の I/O ドライバは 8 MHz の出力矩形波向けに最適化されており、これが最大搬送波周波数です。これより高周波のクロック源を使う出力信号が歪みます。

MCC が生成する API

MCC で必要なコンフィグレーションを選択すると、既定値の `DSM_Initialize()` 関数と多数の短い(1 行の)関数が生成され、DSM 動作の手動制御、変調の開始/停止、出力ピンの制御を実行できます。

ピン配置

DSM は当初 PSS のないデバイスで提供されていたため、入力ピンには 3 つの選択肢しかありませんでした。最近の製品が備えている PSS では柔軟性が高められており、任意のデジタル I/O に割り当てる事ができます。

ホームワーク

- DSM が CLC AND-OR ブロックとは異なるのはどのような点か調べてみましょう。
- 複数の CLC ブロックを使えば独自の DSM を設計できるか試してみましょう。

オンライン リソース

『TB3126 - PIC16(L)F183XX のデータ信号モジュレータ(DSM)』

ZCD - ゼロクロス検出器

概要

全ての I/O ピンに見られる ESD 保護構造(しばしばダイオードで表される)を使い、大きな直列制限抵抗を挿入して I/O にかかる高電圧を検出する事は以前から当然のように行われてきました。
この方法は確かに有効であり、たとえ数千 V の電圧がかかっても損傷は一切生じません。しかし、好ましくない副作用があります。AC、特に負電圧が印加されると、抵抗を流れる電流は保護「ダイオード」によってデバイスのサブストレートを通って最終的にグランド(Vss ピン)に流れます。その影響は、ADC の読み値オフセットを加えたり参照電圧(FVR)を不正確にしたりと害のないものだけでなく、低消費電力オシレータの障害またはデバイス BOR を引き起こす事があります。
現在では小型の PIC マイクロコントローラでも非常に多数の(敏感な)アナログ回路を内蔵しており、全ての内部回路が低消費電力向けに最適化されているため、このような影響が目立ちます。
ゼロクロス検出モジュールは入力ピンを常に公称電圧レンジ内に維持し、サブストレート注入電流を完全に防止する事で、このような危険を全て排除します。

動作原理

ゼロクロス検出回路は、入力ピンを公称電圧レンジ内に維持するだけでなく、**正確**に特定の電圧で動作させるという考えに基づいています。これを実現するために(図 3.4 参照)、入力電圧を内部参照電圧(第 7 章「FVR」参照)と比較して、2 つの電流源を交互に駆動します。

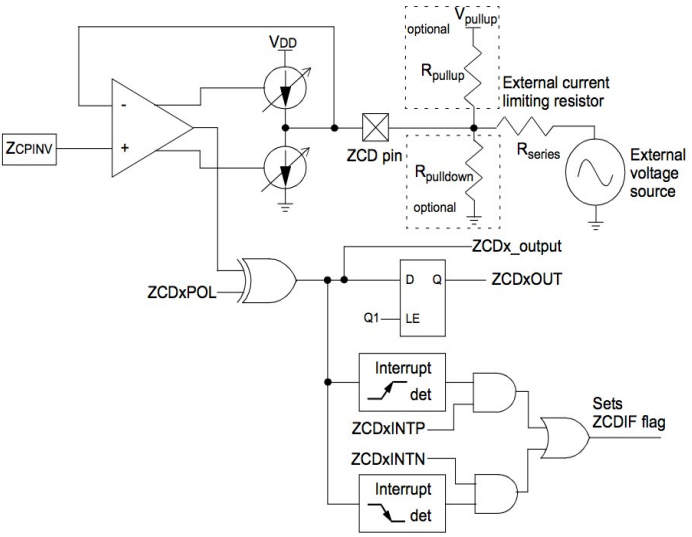


図 3.4: ゼロクロス検出器のブロック図

外部電圧源が直列抵抗の一端をプルアップすると、ローサイド電流源が有効となり電流がデバイスに流れて直列抵抗で電圧降下が発生し、その影響を無効化します。同様に、外部電圧源が直列抵抗の一端をプルダウンすると、ハイサイド電流源が有効になり、電流がデバイス外部に流れ反対の電圧降下が発生させます。直列抵抗の値が適当である場合(約 1 MΩ)、AC 電源電圧(約 240 V (typ.))を無効にするには比較的小さい電流(0.3 mA 未満)で十分です。
従って、2 つの電流源が切り換わる瞬間でゼロクロス イベントを検出できます。

アプリケーション

ゼロクロス検出は通常、AC 電源をトライアックまたは IGBT と組み合わせて使うアプリケーションで、位相カットで制御するために必要です。
しかし、ZCD は比較的安定したタイミング基準として単にライン周波数を検出するためにも使えます。

いくつかの小規模なアプリケーションでは、安全スイッチを実装するために ZCD で単に位相の存在を検出しています。これにより、配線コストを削減できます。

制限事項

0 V ではなく約 1 V の参照電圧を使っているため、コンパレータに起因するオフセットによって小さな位相誤差が生じますが、これは容易に補償できます。
本書の執筆時点では、複数の ZCD モジュールを備えた PIC マイクロコントローラはありませんが、複数の ZCD を内蔵する製品が計画されています。

MCC が生成する API

`ZCD_Initialize()` 関数以外に MCC が提供するのは、ポーリング関数 `ZCD_IsLogicLevel()`だけです。割り込みが有効になっている場合、これらは Interrupt Manager に自動的に挿入され、`zcd.c` ソースファイルに ZCD_ISR()テンプレートが追加されます。

ピン配置

ZCD は基本的にアナログ回路であるため、PPS による割り当て変更は容易ではありません。現在の全ての実装で、ZCD は決まったピンでのみ使えます。

ホームワーク

- 整流ブリッジの後で ZCD を使った場合、ゼロクロスを検出できるか調べてみましょう。
- 電力線通信信号(X10)が ZCD にどのように影響するか、またフィルタで除去するべきか調べてみましょう。
- ZCD を使って(より)高周波の AC 入力を検出する場合、どのようなトレードオフが発生するか調べてみましょう。

オンライン リソース

http://microchip.com/ZCD – CIP – ZCD の概要

『TB3013 - Using the ESD Parasitic Diodes on Mixed Signal Microcontrollers』

『TB3099 - ゼロクロス スイッチングによるリレーの長寿命化』

第 4 章 不揮発性メモリ

概要

フラッシュメモリは、現在のマイクロコントローラ アプリケーションの大多数で使われている技術です。しかし、全てのフラッシュ メモリが同じように作られている訳ではありません。PIC16F1 デバイスはその極小 8 ビットコアのおかげで、成熟した CMOS プロセスを使えます。CMOS プロセスは最大数百万回の消去/書き込みサイクルに対する耐性を持ち、40 年間データを保持できます。

PIC16F1 デバイスはデータ EEPROM、フラッシュメモリ、高書き込み耐性フラッシュという 3 種類のメモリを採用して、最善の不揮発性ストレージを提供しています。

データ EEPROM

EEPROM (Electrically Erasable Programmable Read-Only Memory)は長年使われており若いプログラマはその歴史を知らないため、もはや省略せずに正式名称を書く人はいません。

EEPROM は、データを頻繁に書き換えられる優れた**不揮発性**ストレージ技術であるため、メモリセル設計では書き込み耐性、データ保持期間、使いやすさを重視しています。

データ EEPROM はバイト単位のアクセスが可能で、ユーザには透過的に消去サイクルを実行する事で、データ書き込みシーケンスを自動化します。メモリアレイがデバイスのメインのフラッシュ プログラムメモリ アレイから切り離されているため、読み出しは瞬時に完了します。書き込みは低速(約 2 ms)ですが CPU の動作と非同期で行われます。書き込みシーケンスが完了するまでループでアイドルのまま待機してもかまいませんし、(EEPROM が関連しない)他のバックグラウンド タスクを実行する事もできます。

データ EEPROM アレイを内蔵した PIC16F1 モデルは、10 万回以上の消去/書き込みサイクルを保証しています。

XC8 の API

XC8 コンパイラは、EEPROM データにアクセスするための小さいライブラリを標準で提供しています。ライブラリには以下の関数が含まれます。

- eeprom_write()とeeprom_read()はバイト単位のアクセスです。
- eecpymem()とmemcpyee()はブロック単位のデータ転送です。

例

```
/*
 * Project:     EEPROM
 * Device:      PIC16F1829
 */
#include <xc.h>
#include <stdint.h>

// initialize EE at programming
__EEPROM_DATA( 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x8, 0x9);

void main(void) {
    uint8_t     data;
    uint8_t     buffer[8];

    data = eeprom_read( 1);     // read a byte from EE, data=0xB
    data = EEPROM_READ( 2);     // same using the macro version, data=0xC

    eeprom_write( 1, 0x55);     // write a byte to the EE
    EEPROM_WRITE( 2, 0x56);     // same using the macro version

    eecpymem( buffer, 0, sizeof( buffer));   // copy EE to ram buffer
    buffer[0] = 0x00;
    memcpyee( 0, buffer, sizeof( buffer));   // copy data back to EE
}
```

フラッシュメモリ

マイクロコントローラのフラッシュ プログラムメモリ アレイは、様々な目的を念頭に置いて設計されています。何十年間も信頼性の高い動作を維持するためにデータを長期保存し、多数の消去/書き込みサイクルを経ても書き換え可能でありながら、非常にコンパクトで低コストである事も必要です。全てではないとしてもほとんどの PIC16F1 マイクロコントローラで、フラッシュメモリはチップ上で最大の構造物です。
このため、フラッシュ プログラムメモリでは、バイト単位ではなくブロック単位で消去を実行します。フラッシュメモリへの書き込みも同様で、16 または 32 バイトのブロック(行とも呼ぶ)単位で行います。これにより大規模アレイに格納されたセルの効率が上がり、インサーキットでのデバイス プログラミングの書き込みがより高速となります。
フラッシュ プログラムメモリからの読み出しは高速で、ワード(14 ビット)単位で行います。最大デバイス速度では、4 クロックサイクルごとに 1 ワードを読み出し

ます。フラッシュメモリへの書き込みには多少注意が必要です。書き込みシーケンスを開始する前に、まず目的の行で消去シーケンスを実行し、次に一連のラッチに新しい情報をプリロードする必要があります(16 または 32 ラッチで 1 ワードを格納)。
消去および書き込みシーケンス(2 ms (typ.))中、コアは命令フェッチ用にフラッシュメモリ アレイを使えないため、コアの実行は自動的にストールします。
書き込みプロセスは 16 または 32 ワードの行単位で行うため、データ EEPROM と比べた場合の実効書き込み速度(バイト/秒)は、相当する倍数を乗じたものになります。
全ての PIC16F1 モデルはフラッシュメモリ アレイの内容に対して温度レンジ全体(-40～+ 85°C)で 1 万回以上の消去/書き込みサイクルを保証しています。

MCC が生成する API

MCC ダイアログ ウィンドウでフラッシュメモリ モジュールを選択すると、以下の 4 つの関数からなる標準セットが **memory.c** ファイルに追加されます。

- `FLASH_EraseBlock()`と `FLASH_WriteBlock()`の 2 つの関数は、行全体に対する操作です。**Write** 関数は、**write コマンド**を正しく実行するために、先に **Erase** 関数を呼び出す事に注意してください。
- `FLASH_ReadWord()`は、フラッシュまたは HEF メモリからワードを読み取ります。
- `FLASH_WriteWord()`は上記 3 つの関数全てを使って、現在の行の内容を適切なサイズ(2 番目のパラメータとして渡す)を持つバッファに読み出し、行全体を消去し、**新しい**ワードを一時的にバッファに格納し、最終的にメモリに書き込みます。

HEF - 高書き込み耐性フラッシュ

高書き込み耐性フラッシュ ブロックは新しい技術で、主に専用データ EEPROM アレイのないモデルが内蔵しています。HEF の目的は、メインのフラッシュメモリ アレイが持つ書き込み速度とコストを維持しながら、データ EEPROM と同レベルの書き込み耐性を達成する事でした。このため HEF は現在、メインのプログラムメモリ アレイのサブセットとして、通常は最後の 4 行または 8 行に実装されています。

その結果、データ EEPROM と同様の 10 万回の消去/書き込みサイクル(0～60°C 限定)を保証しながら、フラッシュメモリの速度で読み書きを実行します。最大 32 バイトのデータを 2 ms 以内に書き込めます。これは、不揮発性メモリを使ってデータ/状態を格納するアプリケーションでバッテリ電圧の低下またはブラウンアウト発生時に非常に有益な特長です。データの書き込み速度が高いほど(このケースでは 32 倍)、バックアップ中にアプリケーションに電力を供給し続けるためのパワーコンデンサ容量は小さくて済みます。
このメモリアレイはプログラムメモリ空間と共有されているため、必要に応じて容量を設定できます。フラッシュ プログラムメモリへの書き込み時と同様、HEF への書き込み中もコアはストールします。

ホームワーク

- ブラウンアウト イベント発生時に HEF を使ってアプリケーションの状態/データをバックアップするメリットの比較で、1 uF のコンデンサ(セラミック、1 x 1 x 2 mm)を使った場合と 32 uF のコンデンサ(タンタル/電解、5 x 5 x 10 mm 以上)を使った場合のサイズとコストの違いを調べてみましょう。
- アプリケーションのパラメータ更新頻度と NVM の書き込み耐性(消去/書き込みサイクル数)に対してアプリケーション寿命を比較してみましょう。
- またアプリケーション寿命を 10 年だと仮定し、アプリケーションが HEF ブロックを使ってデータ/状態を 1 日に保存できる回数はどのくらいになるか調べてみましょう。内蔵メモリの代わりに外付けシリアル フラッシュを使った場合のコスト増はどのくらいか、データアクセスの複雑さと時間はどのくらい増加するか検討してみましょう。

例

アプリケーション ノート AN1673 では、絶対メモリアドレスの代わりに HEF メモリブロックに番号(0～3)を付けて管理を簡略化する代替 API を使う方法を紹介しています。以下にその使用例を示します。

```
/* Project:     HEF
 * Device:      PIC16F1509
 */
#include "system.h"
#include “HEFlash.h”

void main(void)
{
   uint8_t r;
   typedef struct {uint16_t ID; char Name[20]; uint32_t Amount;} Record;

   // a block of data that needs saving -- fast!
   Record data = { 0x1234, "HE-FLASH", 42};

   // write data to HEF block-1  (2ms!)
   r = HEFLASH_writeBlock( 1, (void*)&data, sizeof( data));

   // empty the buffer
   memset( &data, 0, sizeof( data));

   // read back its contents
   r = HEFLASH_readBlock( (void*)&data, 1, sizeof( data));

   // read a single byte from block-1 at offset 5
   r = HEFLASH_readByte( 1, 5);

   while( 1);
}
```

オンライン リソース

http://microchip.com/hef – CIP – HEF の概要

『AN1673 - PIC16F1XXX の高書き換え耐性フラッシュ(HEF)ブロックの使い方』

『TB016 - How to Implement ICSP Using PIC16 FLASH MCUs』

『TB072 - FLASH Memory Technology: Considerations for Application Design』

『AN1019 - EEPROM 書き込み耐性の基礎』

『AN1188 - Interfacing with UNI/O® Bus-Compatible Serial EEPROMs』

『AN1449 - High-Reliability and High-Frequency EEPROM Counter』

第 5 章 安全性機能

CRC - メモリスキャナ付き巡回冗長検査

概要

巡回冗長検査とは組み込み制御で一般的に使われる誤り検出コードの事で、通信の完全性とマイクロコントローラ内部のメモリ(フラッシュ)の内容の完全性を保証するために使います。CRC アルゴリズムには多くの種類があり、**多項式**のサイズによって分類されています。多項式のサイズが大きいほど、より広く誤りを検出できます。CRC アルゴリズムは一連のシフトおよび XOR 操作に変換できるため、マイクロコントローラでソフトウェアとして実装する事は比較的容易です。しかし、多項式のサイズが大きくチェック対象のデータが多いと、CPU の負荷が非常に高くなります。

特に、**セーフティ クリティカル**なアプリケーション(EN/IEC 60335-1、EN/IEC60730 Class B 認証や UL98 認証等が必要なもの)では、初回実行前に、アプリケーションを含むフラッシュメモリ空間全体の CRC をチェックする必要があり、その後もアプリケーション寿命を通じて定期的にチェックを繰り返す必要があります。

このような場合、CPU 負荷を軽減し、アプリケーションの開発とテストを簡略化するには、コアから独立した CRC 周辺モジュールが非常に有益なツールです。

動作原理

PIC16F1 ファミリに最近導入された CRC モジュールは、16 ビット アーキテクチャ(PIC24/dsPIC)で提供されていた CRC アルゴリズムと似たアルゴリズムをハードウェアで実装したものです。本モジュールは 16 ビットまでの任意の標準多項式を適用するように設定できるため、非常に多くの通信および安全性用途に対応できます。

設定後、シリアル通信インターフェイスからチェック対象データが到着するか、または RAM バッファ内に転送用パケットデータが準備されると、CPU はこのデータを CRC モジュールに送る事ができます。

しかし、この世代の製品に追加された新機能の中で本当にユニークなのは、**メモリスキャナ** ユニットからの自動供給機能です。これにより、CPU に負荷をかける事なくバック

グラウンドで、デバイス プログラム (フラッシュ) メモリ内の任意のセグメント(または全体)の CRC を計算できます。
アプリケーションに応じて、メモリスキャナは以下のいずれかの動作モードに設定できます。

- **バーストモード**は、最大速度で CRC 計算を実行する必要がある時に使います。CRC 実行中はスキャナが絶対的な優先権を持ち、CPU を*ストール*する事で、その他のタスクによる CPU の使用を防止します。これにより CRC 計算は、ソフトウェア実装のみの場合と比べて桁違いに高速に実行されます。
- **コンカレント モード**も高 CRC スループットを目的に設計されていますが、CPU をストールさせず、CRC アクセスサイクルの合間に動作を継続させます。これにより CRC 計算は高速で実行され、CPU は減速しながらも、低優先度のタスクをバックグラウンドで実行できます(割り込みへの応答等)。
- **トリガモード**は、CRC 性能と CPU 性能のを柔軟に取るように設計されています。ユーザ設定可能なトリガ (例: タイマで定期的に生成)で、メモリスキャナを優先する CPU サイクルの割合を定義します。
- **ピークモード**は最も穏やかなモードです。CPU が最高優先権を持ち、分岐命令(ループ、割り込み等)の実行時と同様に、CPU がメモリ アクセスサイクルを使用してない時のみ、CRC メモリスキャナがメモリ アクセスサイクルを使う事を許可します。これにより、CRC の実行速度は実行アプリケーション コードに依存しますが、CRC メモリスキャナはアプリケーションに対して完全に**透過的**です。

組み込み制御アプリケーションではしばしば割り込み応答に対して厳しいレイテンシと実行が要求されるため、メモリスキャナ モジュールはもう 1 つのモード(**割り込みモード**)を備えています。このモードでは、ISR 実行中にスキャンを一時停止し、割り込みから復帰後に再開できます。

アプリケーション

概要セクションで述べたように、CRC の使用はしばしば(シリアル)通信プロトコルと関連付けられます。代表例は LIN バスまたは単線式通信を参照してください。**セーフティ クリティカル**なアプリケーションでは、業界標準によって CRC モジュールの使用が義務付けられている事がしばしばあり、ここでメモリスキャナの柔軟性が効果を発揮します。例えば、初めての全体メモリチェックを高速化するために、アプリケーション起動中に**バースト**モードを使うと良いでしょう。その後トリガまたは**ピーク**モードを使うと、定期的な再チェックの要件を満たしながら、CPU に負荷をかけません。

制限事項

CRC メモリスキャナの動作はウォッチドッグとは無関係です。このため、優先度の高い CRC モード(バーストまたはコンカレント)が選択されている場合、CPU から取られた(CRC が使った)サイクルが WDT の制限を超えないように注意する必要があります。
17 ビットを超える CRC 多項式を必要とする通信プロトコルはサポートされません。幸い、このような通信プロトコルは 8 ビットマイクロコントローラのアプリケーションではあまり使われません。

MCC が生成する API

MCC の CRC 設定ウィンドウは非常に便利です。一般的な CRC 多項式が事前に選択され、テストボタンまで含まれているため、テスト入力に対する周辺モジュールの (シミュレーション)動作と予測結果を指定した設定/多項式に対して検証できます。これらは全てアプリケーションのコンパイルやインサーキット エミュレータの起動なしで実行できます。
API には便利な関数がいくつか含まれており、CRC モジュールの読み込み、開始、停止、結果のフェッチを実行できます。

ホームワーク

- Wikipedia で「**Cyclic_redundancy_check**」を調べてみましょう。
- CRC アルゴリズムと使用多項式に対して、いくつの用途が記載されているか調べてみましょう。それらのうち、本 CRC モジュールで対応できるものはいくつあるか調べてみましょう。

オンライン リソース

『AN1817 - クラス B アプリケーションで拡張コア PIC16F1XXX のハードウェアまたはソフトウェア CRC を使う方法』
『AN1148 - Cyclic Redundancy Code (CRC)』
『AN1229 - PIC® MCU および dsPIC® DSC 向けクラス B 安全規格ソフトウェア ライブラリ』

WDT - ウォッチドッグ タイマ

概要

優れたウォッチドッグ モジュールは、1989 年の初期モデル(PIC16C54)以来ずっと、PIC の強みの 1 つです。イネーブル コンフィグレーション ビット(OTP)が設定された場合、内部オシレータはウォッチドッグ モジュール用のクロック生成専用でした。ほとんどの競合製品が目的のタイムアウト周期を実現するためにシステムクロックを使っており、コンフィグレーション ビットを RAM に格納していたため容易に上書きされてしまった当時、これは安全性に優れた機能でした。現在の PIC16F1 デバイスはこの安全性機能を維持したまま、いくつかの方法でこれを拡張して、ウォッチドッグ回路を非常に柔軟かつ便利なツールにしています。

動作原理

ウォッチドッグは基本的に独立したタイマであり、プログラムが**スムーズに実行**されている時は CPU によって(特別な命令コードを使って)定期的にリセットされています。CPU が**スタック**すると、その原因が予想外の(ソフトウェアの不具合による)**ソフト**条件であれ、宇宙線による RAM ビットの反転(実際に発生するのです!)であれ、WDT がタイムアウトしてデバイスをリセットします。

PIC16F1 マイクロコントローラ ファミリに追加された新機能の中でも、特に以下については注目する価値があります。

- タイムアウト周期を 1 ms から 256 s のレンジで設定できます。
- **スリープからの復帰** - CPU が低消費電力モード(スリープ)にある時に WDT タイムアウトが発生すると、CPU はリセットではなく復帰し、その状況を示すステータスビットがセットされます。
- **ウィンドウモード**(一部モデルのみ) - アプリケーションの安全性をさらに高め(**不正行為**を防止し)ます。有効なリセット ウィンドウを定義できます。WDT のクリアが早過ぎる、または全てのループ内で行われる場合、WDT の有効性は著しく下がります。ウィンドウ式機能はこれを防止するために役立ちます。

- **ソフトウェア制御有効化/無効化** - 独立したオシレータに電力を供給する余裕が全くないアプリケーションでは、ソフトウェアで WDT を有効化/無効化するように(フラッシュのデバイス コンフィグレーション ビットで)設定できます。
- **自動スリープ無効化** - デバイスが低消費電力モード(スリープ)に移行すると、WDT を自動的に無効にします。
- 一部のモデルでは、MFIntOSC を分周した値を代替のクロック入力として選択する事で、CPU と WDT が同じオシレータで動作しないようにできます。

アプリケーション

元々の設計目的であった安全機能以外にも、WDT はしばしば低消費電力モードからの復帰メカニズムとして使われます。

制限事項

WDT は CPU が低消費電力(スリープ)モードにある時も動作**できる**ため、超低消費電力アプリケーションで定期的な復帰を可能にするための貴重なリソースになりました。PIC16F1 ウォッチドッグ回路は、精度が低く未校正であった以前のモデルと比べると、専用オシレータの安定度と精度が大幅に高くなっています(最小で約 10%)。
PIC16F1 の WDT は温度変化の影響も大幅に低くなっており、以前は WDT が(誤った方法で)使われていたアプリケーションが使われなくなりました。そのようなアプリケーションは第 7 章「アナログ機能」の「温度インジケータ」を参照してください。
しかし、WDT 回路が特定の無線プロトコルの要件を満たすのに十分なタイミング精度を提供する事は保証できません。この場合、アイドルモードを利用し、(使える場合は)新しいパワーダウン制御を使い、適切な低消費電力の**校正済み**オシレータを選択する事でデバイスの消費電力を抑えます。

MCC が生成する API

MCC では、システム モジュールのコンフィグレーション ビット選択で WDT モジュールを設定できます。生成されるコード(プラグマ)は、`mcc.c` ソースファイルに格納されます。

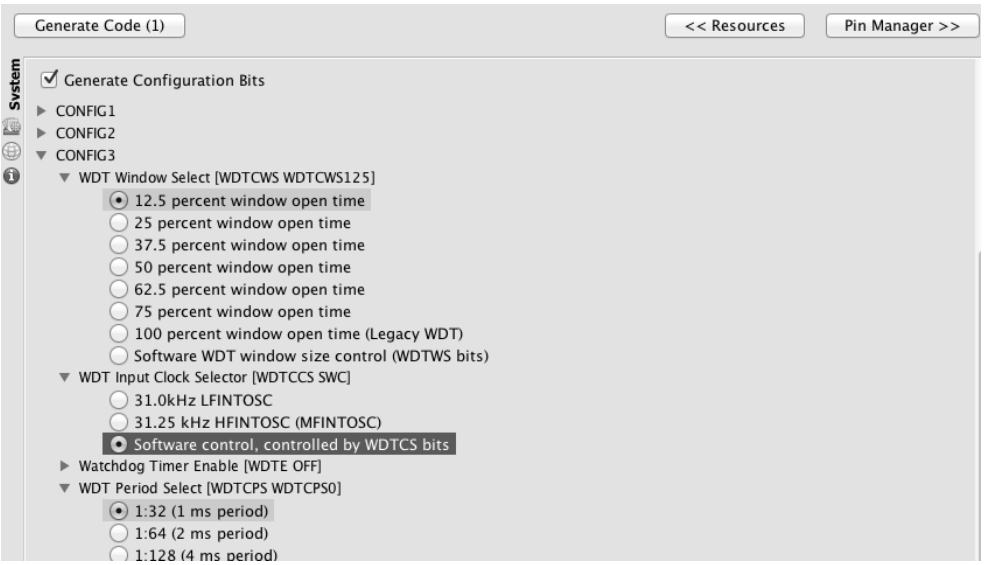


図 5.1: MCC でのシステム コンフィグレーション ビットの設定

ホームワーク

- デバイス(PIC16F161X)データシートで、ウィンドウ式ウォッチドッグ タイマ(WWDT)の章を調べましょう。
- WWDT を備えたデバイスはリアルタイムでウォッチドッグ タイマの値を観測(WDTTMR レジスタを使用)できる事、そして値が正しく増分している事を確認してみましょう。これにより、クラス B アプリケーションの認証/テストが容易になります。

オンライン リソース

『TB3123 - PIC®マイクロコントローラのウィンドウ式ウォッチドッグ タイマ』

Enhanced

リセット回路

概要

ほとんどの 8 ビット組み込みアプリケーションは過酷環境に対応できます。過酷環境とは非レギュレート電源とノイズが含まれ、さらにはデバイスが常時全ての最大定格仕様で動作すると想定される環境です。大量生産アプリケーションでは、コスト(とスペース)を節約するためなら取り除けるものは何でも取り除いてしまいます。従って、最も基本的な電源監視機能ですら贅沢品であり、めったに実装されません。PIC マイクロコントローラはいかなる状況にも対応し、通常動作に戻る事が期待されています。PIC16F1 デバイスが最も難しい状況でも動作できるリセット回路を複数備えているのはこのためです。

POR - パワーオン リセット

パワーオン リセット回路は最も重要な回路です。POR 回路は、マイクロコントローラが持つどのアナログ/デジタル機能が有効になるよりも前に動作を開始している必要があるため、いかなるコンフィグレーション、ロジック、先進アナログ リファレンスも利用できません。PIC16F1 シリーズの POR は、適切な勾配(極端に遅くないもの)の立ち上がり Vdd 電圧を検出して、固定しきい値(1.6 V (typ.))を超過すると、残りの回路の初期状態を保証するように設計されています。
電源電圧が安全なしきい値に達した後、マイクロコントローラの残りの部分が無事に立ち上がる(1.8 V)前に低下(バウンス)した場合、最小限(0.8 V (typ.)以下)のヒステリシスを前提として安全にリアームする必要があります。

BOR – ブラウンアウト リセット

ブラウンアウト検出回路は POR がデバイスを**立ち上げた後に**有効となり、マイクロコントローラの電圧を安全動作電圧の公称値(1.8 V)より下回らせる可能性のある電源電圧の変動に対処します。
BOR のしきい値はコンフィグレーション ビットを使って選択します。また、(POR が使えない)正確な内部参照電圧リソース(FVR)を使います。BOR 回路は低消費電力オシレータ(LPIntOsc)にもアクセスできるため、電源電圧が安全ゾーンまで回復すると、リセット条件が解放される前に時間遅延(64 ms (typ.))を取ってからリアームします。
BOR 回路は、消費電力を低減するためにデバイスが低消費電力(スリープ)モードに移行すると、自動的に無効になるように設定できます(上述の低消費電力タイマと参照電圧回路を参照)。

LPBOR – 低消費電力 BOR

New

一部の PIC16F1 デバイスは、超低消費電力アプリケーション用に特別に設計された低消費電力 BOR 回路を備えています。この回路を使うと誤差が大きくなる代わりに消費電力を大幅に低減できます。

MCLR - マスタクリア

上記全ての内部リセット回路が機能している場合、実際のリセット入力ピン(Microchip 用語では伝統的にマスタクリアまたは MCLR と呼ぶ)を(コンフィグレーション ビットで)無効にし、標準入力ポートピン(RA3)として使う事ができます。
MCLR はリセットとして使用されるかどうかに関係なく、インサーキット プログラミング モード(Vpp)へのゲートとしての役割を持ち続けるため、高電圧(約 9 V)が印加された場合にこれを検出し、耐える能力が必要とされます。I/O ドライバは完全なプッシュプルにはなりません。大半のデバイスでこのピンは入力専用機能になりますが、一部のデバイス(PIC16F155x ファミリ)では、I²C インターフェイスの SDA ピンとしてオープンドレイン構成でも使えます。

FSCM - フェイルセーフ クロック監視

フェイルセーフ クロック監視は、外付けオシレータの障害を検出するように設計されています。外付けオシレータはビア、PCB のレイアウトの問題、不適切なはんだ付け接点、EMI に起因する寄生容量の影響を最も受けやすいモジュールです。
FSCM はデバイスのコンフィグレーション ビットで有効化され、選択された外部クロック信号と低消費電力内部オシレータ(LFIntOSC)を比較する事でクロックを監視します。外付けオシレータがサンプリング期間(約 2 ms)中に通常発振を 1 回も生成できなかった場合、クロック障害イベントが生成されます。その結果、システムクロックが (OSCCON レジスタに従って)内部オシレータに切り換えられるため、精度は下がりますがアプリケーションは動作を継続できます。FSCM 割り込みが有効になっている場合、割り込みがトリガされ、アプリケーションは必要な予防措置を取るか、または新たに再起動する事ができます。

オンライン リソース

『TB087 - Using Voltage Supervisors with PICmicro MCU Systems』
『TB3121 - 8 ビット ミッドレンジ マイクロコントローラの伝導性および放射性エミッション』

第 6 章 通信機能

I²C – Inter Integrated Circuit バス

概要

過去には、本物のプログラマならシリアル EEPROM と通信するためだけに I²C モジュールを必要としない、という時代がありましたが、おそらくは私が年を取り過ぎただけでしょう。当時、私たちの間で広まっていたサンプルコードは、わずか 40 行(当然アセンブリ)で I²C マスタを大ざっぱに実装し、24LC16 とのデータの読み書きを実行できるものでした。ただ公平を期するために言うと、その頃は組み込みアプリケーションへの期待もずっと低いものでした。現在は、はるかに複雑なタスクをサポートする必要があるため、I²C 等のローレベルのシリアル通信プロトコルを Bit-Bang 方式で行うと、マイクロコントローラが持つ可能性が無駄になります。

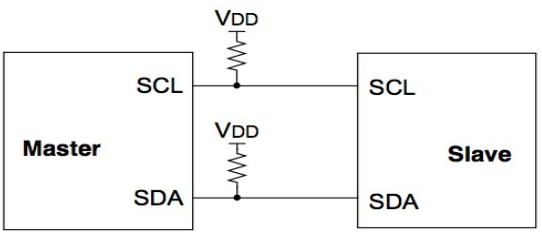


図 6.1: I2C モードの MSSP のブロック図

動作原理

I²C バスをサポートする全ての PIC16F1 マイクロコントローラは、マスタおよびスレーブ同期ポート(MSSP)モジュールで、このプロトコルをサポートします。MSSP モジュールは、Microchip 社でよく見られる倹約的な手法に従って、コンフィグレーション レジスタとバッファを節約するために、SPI インターフェイス(次のセクション参照)と共有されています。
名前の通り、マスタとスレーブの両機能が 7 および 10 ビット アドレッシング モードでサポートされており、全ての一般的な用途に対応しています。MSSP モジュールは、とりわけ

SMBus および PMBus プロトコル(I²C に基づく)の実装に役立つ追加機能をサポートしています。これには以下が含まれます。

- アドレスマスクを使う事で、デバイスはレンジ内のどのアドレスにも応答できるようになります(ベクタ処理)。
- コリジョン検出により、複数のマスタが同時にバスを制御しようとした場合にフラグを立てる事ができます。
- クロック ストレッチにより、スレーブデバイスが応答(ACK/NACK)を準備する時間を延長できます。
- ジェネラルコール検出を使うと、選択されたアドレスとマスクに関係なく、スレーブがアドレス 0x00 のコールに応答します。
- 各 start/stop イベントに割り込み生成できるため、通信シーケンスを最大限に制御できます。
- baud レート ジェネレータを使うと、メイン システムクロックを分周する事で必要なデータ転送速度を実現できます。

アプリケーション

I²C インターフェイスの使用は、通常同じ PCB 上のごく近い位置にあるデバイス同士の通信に限られます。これは通常、センサ、ディスプレイ、不揮発性メモリとの通信を指しますが、近くにある別のマイクロコントローラとの通信も含みます。シリアルポートは同期式で、高精度(水晶振動子)オシレータを使う必要がないため、アプリケーション コストを削減できます。I²C インターフェイスで双方向通信を確立するために必要なのはワイヤ/ピン 2 本だけなので、スペースが最優先で最大 1 Mbps の転送速度で十分なアプリケーションでは、しばしば SPI よりも I²C が好まれます。

制限事項

I²C インターフェイスはデバイスのスリープ中も動作可能です。スレーブモードで構成されている場合、START 条件を検出するか(対応する割り込みイベントが有効な場合)、アドレス/データバイトを受信するとデバイスを復帰させます。しかし、I²C インターフェイスが全てのデータ トランザクションを完了するには、依然として若干の介入が必要です。従って、現時点では本当にコアから独立した動作とは言えません。

SPI と I²C の機能は複数のレジスタを共有しているため、どちらかの機能に MSSP モジュールを使っているもう一方を使う事はできません。従って、アプリケーションで I²C と SPI の両インターフェイスを必要とする場合、MSSP モジュールを 2 つ備えたデバイスを選ぶ必要があります。

MCC が生成する API

MCC は I²C インターフェイスのマスタ構成とスレーブ構成を区別しますが、どちらの場合も、割り込みベースのステートマシンを介してトランザクションを処理するために、割り込みを使う事を想定しています。

マスタモードの I²C が選択された場合、MCC は複雑な**トランザクション要求**シーケンスを自動実行するステートマシンを生成します。各トランザクション要求はアドレス送信、データ送信、データ読み出し等の代表的な I²C コマンドの 1 つを記述できます。
トランザクション要求(TRB)は、一連のヘルパー関数`..Build()`を使って生成され、`I2CMasterTRBInsert()`関数でキューに追加されます。
ステートマシンはキュー内を連続的に処理して、各トランザクションの実行ステータスを報告し、最終的にキュー内の全タスクの完了を通知します。

スレーブモードの I²C が選択された場合、MCC は代表的なスレーブデバイスの挙動をエミュレートする別のステートマシンを生成します。I²C プロトコルの処理はコールバック関数(`I2C_StatusCallBack`)内に定義されており、デバイスの挙動とは明確に切り離されています。生成されたファイルにはコールバック関数の例が含まれており、RAM バッファを使ったシリアル EEPROM のエミュレーション方法が示されています。

ピン配置

I²C 仕様では、他の CMOS I/O に通常採用されているしきい値とは異なる I/O しきい値(Vol、Voh)が要求されます。このため、インターフェイスには通常特定のピンのペア (RC0 と RC1)が割り当てられます。
PPS 機能を使える場合、(バス仕様への準拠は完全ではなくなりますが)I²C インターフェイスを任意のデジタル I/O ピンに割り当てる事が**できます**。
PIC12LF1552 では初めて、小型 8 ピン パッケージでアナログ入力を最大限有効に活用するために、MCLR ピンに I²C SDA 機能が多重化されています。

ホームワーク

- System Management Bus (SMBus)について、マザーボードとスマートバッテリ システムでのアプリケーションも含めて調べてみましょう。
- Power Management Bus (PMBus)についてと、デジタル電源管理での用途について調べてみましょう。
- I²C 入力ピンの I/O(VIH と VIL)仕様を、標準 CMOS と TLL の I/O に適用される標準的なしきい値と比べてみましょう。

オンライン リソース

『AN1028 - Microchip 社製 I²C™ シリアル RTCC デバイスの推奨使用法』
『AN1248 - PIC MCU-Based KeeLoq Receiver System Interfaced Via I²C』
『AN1302 - An I²C Bootloader for the PIC16F1』
『AN1365 - マイクロチップ社製 I2C™ シリアル RTCC デバイスの推奨使用法』
『AN1488 - Bit Banging I²C on Mid-Range MCUs with the XC8 C Compiler』
『AN1630 - USB to I²C Bridge Reference Guide』
『AN690 - I²C™ Memory Autodetect』
『AN734 - Using the MSSP Module for Slave I²C Communication』

SPI – 同期ポート

概要

シリアル ペリフェラル インターフェイスはおそらく最もシンプルなシリアル インターフェイスであり、基本的にデータを交換する 2 つのシフトレジスタ(両端に 1 つずつ)のみで構成されています。I²C とは対照的に、その主な弱点はより多くのピン/ワイヤを必要とする点で、バスに接続するデバイスの数が増えると、この数はさらに増えます。

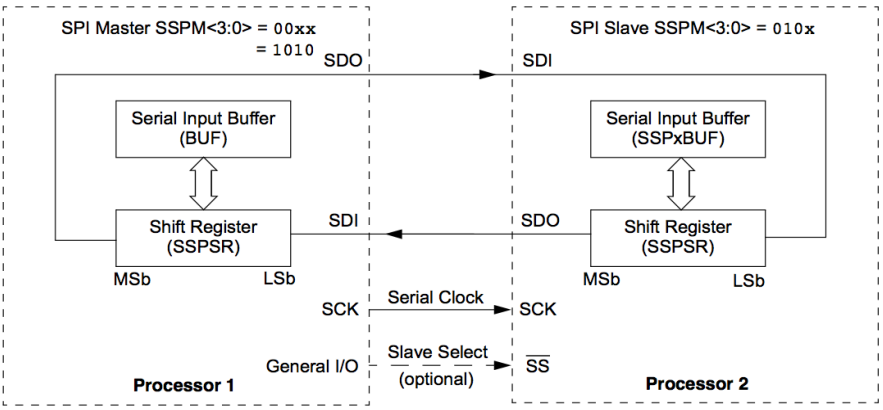


図 6.2: SPI マスタ/スレーブの接続

動作原理

SPI インターフェイスをサポートする全ての PIC16F1 マイクロコントローラは、マスタおよびスレーブ同期ポート(MSSP)周辺モジュールを介してサポートをします。MSSP モジュールは、コンフィグレーション レジスタとバッファを I^2C インターフェイス(前セクション参照)と共有しています。

マスタおよびスレーブ動作のサポート(どちらがシリアルクロック SCK を提供するかで区別)に加えて、アクティブ クロックエッジ(立ち上がり/立ち下がり)とクロック極性の順列として考えられる 4 つのモード全てをサポートしています。

その対称性により、SPI ポートでのデータ転送は送信と受信を同時に行います。

アプリケーション

SPI インターフェイスの用途は、通常同一基板上の近い位置にあるデバイス同士の通信に限られます。これにはセンサ、ディスプレイ、不揮発性メモリデバイス、無線モジュール、トランシーバとの通信を含みますが、別のマイクロコントローラとの通信も含みます。

シリアルポートは同期式で、高精度(水晶振動子)オシレータを使う必要がないため、アプリケーション コストを削減できます。

最大の利点はおそらくデータレートであり、PIC マイクロコントローラのクロック速度に制限される事はありますが、そうでなければ最大 20 Mbps です。

SPI インターフェイスに適したアプリケーションは、MMC/SD カード等のマスストレージ(フラッシュ)デバイスです。実際、このようなデバイスは全て最低限の共通インターフェイスとして SPI モードを備えています。

制限事項

SPI インターフェイスはデバイスがスリープ中でも動作できます。実際、スレーブモードで構成した場合、データ受信時にデバイスを復帰させる事ができます。しかし、SPI インターフェイスが外部デバイス(シリアル EEPROM またはセンサ)とのデータトランザクションを全て完了するには CPU の介入が必要であるため、(現時点では)本当にコアから独立した動作とは言えません。
SPI と I²C 機能は複数のレジスタを共有しているため、どちらかが MSSP モジュールが使っている場合、もう一方を使う事はできません。従って、アプリケーションで I²C と SPI の両インターフェイスを必要とする場合、MSSP モジュールを 2 つ備えたデバイスを選ぶ必要があります。

MCC が生成する API

MCC はマスタモードとスレーブモードの SPI インターフェイス構成を区別しますが、I²C の場合とは異なり、割り込みの使用には対応していません。
生成されるコード(**spi.c** ファイルに格納)には以下の 2 つの主要関数が含まれます `SPI_Exchange8bit()`と `SPI_Exchange8bitBuffer()`は、一度に 1 バイトまたは短いバッファのデータを送受信します。オーバーフローとその他のエラー条件をチェックするための**追加情報**も含まれます。

例

外部シリアル EEPROM モデル 25LC256 (32 KB)に対する SPI インターフェイスの代表的な例を以下に示します。

```
/* Project:     SPI Master
 * Device:      PIC16F1829
 */
#include "mcc_generated_files/mcc.h"

// 25LC256 Serial EEPROM commands
#define SEE_WRSR    1       // write status register
#define SEE_WRITE   2       // write command
#define SEE_READ    3       // read command
#define SEE_WDI     4       // write disable
#define SEE_STAT    5       // read status register
#define SEE_WEN     6       // write enable

main()
{
   uint8_t   status, data;
   uint16_t  address = 1234;

   SYSTEM_Initialize();

   while( 1)       //  main loop
   {
      // 1. send a Write Enable command
      CSEE_SetLow();                  // select the Serial EEPROM
      SPI2_Exchange8bit( SEE_WEN);    // send write enable command
      CSEE_SetHigh();                 // deselect, terminate command

      // 2.Check the Serial EEPROM status
      CSEE_SetLow();                  // select the Serial EEPROM
      SPI2_Exchange8bit( SEE_STAT);   // send a READ STATUS COMMAND
      status = SPI2_Exchange8bit( 0); // send/receive
      CSEE_SetHigh();                 // deselect, terminate command
      // expect status == 2

      // 3.Read a byte from EEPROM address
      CSEE_SetLow();                  // select the Serial EEPROM
      SPI2_Exchange8bit( SEE_READ);   // send a READ STATUS COMMAND
      SPI2_Exchange8bit( address>>8); // send address MSB
      SPI2_Exchange8bit( address);    // send address LSB
      data = SPI2_Exchange8bit( 0);   // get data
      //...(or more)
      CSEE_SetHigh();                 // deselect, terminate command

   } // main loop

} // main
```

ピン配置

初期の PIC16F1 デバイスでは MSSP モジュールのピンは固定されており、通常 SPI と I^2C の I/O(ポート C、ピン RC0～RC3)が一部重複していました。PPS を備えた最近のデバイスでは、SPI インターフェイスの I/O を任意のデジタルピンに割り当てする事ができます。

ホームワーク

- SPI モジュールを、非同期モードの EUSART モジュールと比較してみましょう。
- Microchip アプリケーション ライブラリを参照し、ファイルシステム ライブラリ(レガシー エディションでは MDD ファイルシステム)で、例えば MMC/SD カード上のデータにアクセスするための SPI ポートの使用例を確認してみましょう。
- SPI バス上の信号が全て一方向である事を確認します。これにより、電気的絶縁の必要なシステムの設計がどう簡略化されるかを考察してみましょう。
- Microwire インターフェイスの定義を調べ、PPS モジュールを使ってこのインターフェイスをサポートする方法を検討してみましょう。

オンライン リソース

『AN1029 - マイクロチップ社製 Microwire シリアル EEPROM デバイスの推奨使用法』
『AN1040 - マイクロチップ社製 SPI シリアル EEPROM デバイスの推奨使用法』
『AN1245 - Recommended Usage of SPI Serial SRAM Devices』
『AN767 - Interfacing Microchip's Fan Speed Controllers to a SPI』

EUSART – 非同期シリアルポート

概要

Universal Asynchronous Receiver Transmitter(UART)はかつて、組み込み制御を制していました。PC、端末、プリンタ、モデム経由で外の世界に接続する方法の 1 つで、多くの場合唯一の方法でした。今日、端末とモデムはコンピュータ史にとって考古学的遺産となりました。かつて広く普及していた 9 ピンの D タイプコネクタは、ずっと前から PC に搭載されておらず、USB ポートに取って代わられました。しかし、非同期ポートはその単純さと柔軟性のおかげで、新しいアプリケーションに採用され続けています。実際、PIC16F1 デバイスは拡張(**Enhanced**) UART を備えており、baud レートの自動検出(**LIN バス**規格で使用)と同期(**Synchronous**)通信を含む新しい「技」を実行できます。このため、新しい頭字語(EUSART)で呼ばれるようになりました。

動作原理

シリアルポートが持つ非同期という性質は、前のセクションで説明した I²C および SPI プロトコルには存在するクロック信号が存在しない事を意味します。これにはピン/ワイヤの数が減るという利点がありますが、別の面では接続される両端でビットタイミング (baud レート) が正確に一致する事が前提であるため、オシレータコストが上昇するという欠点があります。

一度この baud レートに達すると(固定のまま、または baud レート自動検出メカニズムを適用する事で)、後続の 8 ビット タイムスロットに対してレシーバとトランスミッタを同期させるために必要なのは START ビットの立ち下がりエッジだけです。STOP ビット(最終的に 2 つ)は、個々の文字列送信を終了させデータ整合性の最初の関数チェックをします。9 番目のデータビットは、データ整合性をチェックするための第 2 の手段としてプロトコルに元々含まれおり、**パリティ**を提供する事でそれぞれの文字送信に偶数個の 1 または 0 が含まれるようにしていました。

しかし、最近のアプリケーションではしばしば、9 番目のビットは別の、より適切な目的に使われています。例えば、2 点間接続であったものをバスに変換する **RS485** プロトコルでは、(マスタが送信した新規メッセージの宛先アドレスを含む)1 バイト目を後続のデータから区別するために 9 番目のビットを使います。

アプリケーション

従来型の RS232 規格通信(信号を± 12V に変換するトランシーバが必要)は、SPI と I²C では対応できない長距離のモジュール間通信に今でも時々使われています。

RS232 規格の通信は今なお、シリアル-USB アダプタを使って組み込み機器とパソコンを接続するために使われており、場合によっては 2 つのシリアル インターフェイス間のブリッジ機能を備えたデバイス(PIC16F145x ファミリのデバイス等)を使って実装されています。

RS422 規格の通信(より複雑な完全平衡型信号トランシーバが必要)を使うと、さらに長い距離(最大数百 m)に対応できますが、使われる事は次第になくなってきています。

RS485 規格のバス接続は従来産業用制御アプリケーションで多く使われた実績があり、マルチドロップの長いデバイスチェーンを構築するために使われました。LIN バス規格の通信も専用トランシーバが必要であり、接続速度とデータ整合性よりも低コストが重視される非クリティカル システムの車載アプリケーションで使われています。

照明アプリケーションでは UARTS を使って DMX512 プロトコルを実装できます。

制限事項

レシーバとトランスミッタが同期できるのは START ビットのみで、最初の STOP ビット以降は 1 ビットの半分を超えるタイミングの違いは許されないため、各デバイスのオシレータ許容誤差は 10 ビットの時(START ビット+ 8 ビットデータ+ STOP ビット)で±2.5%、11 ビットの時(START ビット+ 9 ビットデータ+ STOP ビット)で±2.27%です。2 つのデバイスのうちどちらかが**正確**だと既知のオシレータで、システムクロックの総誤差が抑制されている場合、2 倍の誤差(±5%と±4.55%)まで許容できます。
PIC16F1 デバイスの内部オシレータは工場校正されているため、室温ではこのレベルの精度を達成できますが、標準的な動作温度レンジ(-40～+125˚C)全体では精度を維持できません。さらに、電源電圧が正確にレギュレートされていない場合はさらに誤差は広がります。動作温度と電圧のどちらかが大きく変動する事が予測される場合、外部のセラミック/水晶振動子または EUSART の **baud レート自動検出**機能を使う事を推奨します。
非同期シリアル インターフェイスの最大データ転送速度は、通常距離とケーブル品質に依存します。短距離でも 115k baud を超える事はまれです。おそらく DMX512 通信が最も顕著な例外(250k baud)でしょう。

MCC が生成する API

MCC は EUSART モジュールをサポートするために、基本的な設定関数の `EUSARTx_Initialize()` に加えて、`EUSART_Read()` と `EUSART_Write()` という単純な 2 つの関数を提供しています。STDIO リダイレクト オプションが選択されている場合、これらの関数は `getch()` と `putch()` 内にラップされ、`XC8` コンパイラがこれらを使って `printf()` の全機能を実装します。
また、**割り込みサポート** オプションを選択した場合、割り込みモジュールが自動的にプロジェクトに追加され、読み書き両方の関数が完全にバッファリングされます。

例

STDIO リダイレクト オプションを選択した EUSART を使って、シリアル コンソールと通信する例を以下に示します。

```
/* Project:Console
 * Device:PIC16F1709
 */
#include "mcc_generated_files/mcc.h"

main()
{
    char c;

    SYSTEM_Initialize();

    while ( 1)
    {
        // 1. text prompt
        printf( "Is this Rocket Science?:(y/n)");

        // 2. read a character
        c = getch();
        puts("");
        if ( c == 'y')
            puts( "No, this is too easy!");
        else
            puts( "That's what I am saying!");

    } // main loop
}// main
```

ピン配置

当初、EUSART は PPS のないデバイスで提供されていたため、限られたピンしか選択できませんでした(TX-RC5、RX-RC4)。最近の製品は PPS を備えているため、任意のデジタル I/O に割り当てる事ができます。

ホームワーク

- 同期(Synchronous)モードが機能する原理を調べてみましょう(EUSART の S)。
- CRC と EUSART を組み合わせて LIN バスプロトコルのサポートを検討してみましょう。
- EUSART の入出力データストリームにマンチェスタ 符号化/復号を実装する方法を検討してみましょう(ヒント: CLC を使います)。
- EUSART の出力データストリームに 38 kHz (typ.)の IR 変調を適用する方法を検討してみましょう(ヒント: CLC または DSM モジュールを使います)。

オンライン リソース

『TB3069 - Use of Auto-Baud for Reception of LIN Serial Communications』
『AN1659 - DMX512A』
『AN1099 - LIN 2.0 Compliant Driver Using the PIC16 Microcontrollers』
『AN1310 - PIC16/PIC18 デバイス用高速シリアル ブートローダ』
『AN1465 - DALI (Digitally Addressable Lighting Interface)通信』

USB – Universal Serial Bus – アクティブ クロック チューニング

概要

Universal Serial Bus (USB)はフロッピー ディスク インターフェイス、パラレルプリンタ ポート、シリアルポート等、PC 周辺機器の旧式インターフェイスを置き換えました。
PIC マイクロコントローラは最初、簡単なキーボード コントローラとその他のヒューマン インターフェイス デバイス(HID)機能に対応するため、ロースピード USB(最大 1.2 Mbps)をサポートしていましたが、PIC16F1 世代で USB モジュール機能が拡張され、ハイスピード USB(最大 12 Mbps)にも対応しています。さらに重要な事は、アクティブ クロック チューニング テクノロジを追加した事です。これにより PIC16F145x ファミリのデバイスは内部オシレータだけを使ってハイスピード モードで接続できるようになりました。

動作原理

USB シリアル インターフェイス エンジン(SIE)は比較的複雑な周辺モジュールで、PIC とホスト間の比較的大きなデータブロック (**バルク**モードでは一度に最大 64 バイト)のトランザクション全体を実行します。SIE を CIP(コアから独立した周辺モジュール)と呼びたくもなりますが、USB プロトコルは複雑なため MCU コアによる相当な量の処理と監視が必要です。
幸い、Microchip アプリケーション用ライブラリ(MLA)は包括的な USB ライブラリを収めており、USB アプリケーションで最も一般的なクラス全てに対応した、すぐに使えるサポートモジュールとサンプルを提供しています。ライブラリには以下のクラスが含まれます。

- HID - キーボード、マウス、デジタイザ等
- CDC - 全ての重要な**シリアルポート エミュレーション**
- オーディオ - マイクとスピーカ

- プリンタ - プリンタ機器
- マスストレージ - USB メモリ、ハードディスク ドライブ

アクティブ クロック チューニング

PIC16F145x ファミリで最も興味深い機能は、外付け水晶振動子を使わずにフルスピードで USB 接続できる機能です。この結果、きわめて高速な接続を低コストで実現できます。これは、フルスピードのビットレート(12Mbps)と、この接続の規格に求められる精度(誤差 0.25%未満)を考慮すると実現不可能に思えます。
PIC16F145x の内部オシレータはそこまで高精度ではありませんが、**調整(チューニング)が可能**です。OSCTUNE と言う制御レジスタで、細かい粒度で周波数を調整できます。
アクティブ クロック チューニング テクノロジは、この機能とある USB プロトコルの特性を利用しています。USB プロトコルではホスト PC が特定の **start of frame** (SOF) **トークン**で各通信シーケンス(USB 用語では *フレーム*と呼ぶ)を開始します。フレームは「厳密」に 1 ms 間隔で送信される必要があり、ホストが提供するこのタイミングはきわめて正確です。アクティブ クロック チューニング テクノロジでは、SOF トークン検出メカニズムを使って参照クロック信号を取得し、これに内部オシレータ(調整レジスタ)を同期させます。温度と電源電圧が変化したら ACT モジュールが内部オシレータを再校正し、一定の公差と安定した USB 通信を維持します。

アプリケーション

USB を使ったアプリケーションの種類は、USB バスが置き換えた PC 周辺機器の種類を優に超えて広がっています。PIC16F1 デバイスはロースピードとフルスピード両方をサポートする事で、多くのアプリケーションに対応しています。これらは ACT のおかげで低消費電力、小型、低コストのデバイスが必要なアプリケーションです。
カスタマイズ可能な UART-USB ブリッジ(シリアルポート エミュレーション)は簡単に実装でき、HID クラスはデータとコマンドを PC との間で手軽に転送する柔軟性があるため、これら 2 つは PIC16F145x ファミリの主要用途になっています。

制限事項

MLA ライブラリが USB プロトコルの大半の機能を**標準**で提供しているとはいえ、このプロトコルを面倒に感じる方もいるでしょう。USB プロトコルの理論上の転送レートは 12 Mbps ですが、バス上の 1 つのデバイスが一度に使える帯域幅はその何分の 1 かです。さらに

プロトコルのオーバーヘッドを考慮すると、実効平均連続転送レートが 1 Mbps を超える事は現実的ではありません。マスストレージやプリンタ機器等のアプリケーションでは、RAM メモリ容量も制約要因です。

MCC が生成する API

本書の執筆時点では、MCC はまだプロジェクト設定を支援できません。これは、複雑な通信フレームワークが追加される MCC ツールの将来のエディションで変わる予定です。Microchip アプリケーション用ライブラリ(MLA)は USB アプリケーションで最も人気のあるクラス(HID、CDC、デジタイザ、マスストレージ等)に必要なサポート ファームウェアを全て収めており、全 PIC アーキテクチャ(PIC16、PIC18、PIC24、dsPIC、PIC32)と互換のソースコードを提供します。

ピン配置

他のシリアル通信インターフェイスと同様、USB アプリケーションでもピン割り当てが固定された特別な内蔵トランシーバを使う必要があります。また PIC16F145x デバイスの場合、接続速度を選択するプルアップ抵抗は内部に備えています。

ホームワーク

- Microchip アプリケーション用ライブラリで、PIC16F145x ファミリを使った(少ピン USB ボードが対象の)デモプロジェクトを調べてみましょう。
- USB アプリケーションの開発者は、アプリケーションに適合したドライバが使われるように USB_IF の使用許諾を得た一意のベンダー ID(VID)番号を取得する必要があります。アプリケーションの数量が 1 万ユニットを超えない場合、ライセンス料金を抑えるため(全てのお客様が使う)共有 VID の使用と一意の製品 ID(PID)の割り当てを Microchip 社に申請する事ができます。

オンライン リソース

https://www.microchip.com/usblicensing – USB VID/PID ライセンス

『AN1546 - USB Keypad Reference Design』

『AN956 - Migrating Applications to USB from RS-232 UART with Minimal Impact on PC Software』

『AN1630 - USB to I²C Bridge Reference Guide』

https://code.google.com/p/pic16f1454-bootloader – USB ブートローダ用のオープンソース サンプル プロジェクト(Google Code 内)

例

HID カスタムクラス(MLA USB ライブラリ)を使って単純な「センサ」を実装する例を以下に示します。

```
/* Project:USB HID Sensor
 * Device:PIC16F1459
 */

#include "system.h"
#include "USB/usb.h"
#include "USB/usb_function_hid.h"

void DecodeCMD( void)
{
   switch( RxBuffer[0])       // decode first byte
   {
     case 0x37:         // interpret as "Read POT" command
       if ( !HIDTxHandleBusy(USBInH))   // ready to send packet back
       {
          WORD_VAL w;
          w.Val = ADC_GetConversion( Potentiometer);
          TxBuffer[0] = 0x37;     // Echo back
          TxBuffer[1] = w.v[0];   // LSB
          TxBuffer[2] = w.v[1];   // MSB

          // return transmit buffer to USB SIE
          USBInH = HIDTxPacket(HID_EP, &TxBuffer[0], 64);
       }
       break;

       // add more commands here...
     default:
       break;
   } // switch
}
```

```
int main(void)
{
   SYSTEM_Initialize();

   while(1)
   {
      USBDeviceTasks();

      if ( USBDeviceState == CONFIGURED_STATE)
      {
         if ( !HIDRxHandleBusy(USBOutH))     // packet received
         {
               DecodeCMD();

            // return receive buffer to USB SIE
            USBOutH = HIDRxPacket(HID_EP, (BYTE*)&RxBuffer, 64);
         } // if received
      } // if configured
   } // main loop
} // main
```

第 7 章 アナログ機能

Enhanced

コンパレータ

概要

アナログ コンパレータは非常にシンプル(低コスト)ながら便利なモジュールです。A/D コンバータの利便性にはもちろん敵いませんが、単純なしきい値を検出するだけであれば、A/D コンバータよりもはるかに短い時間でコアに依存せずに応答できます。ここでも重視されるのは**コアからの独立性**です。これは CPU に依存する事なく出力で他の周辺モジュールの入力を駆動し、一連のイベントを高速で生成できる能力です。

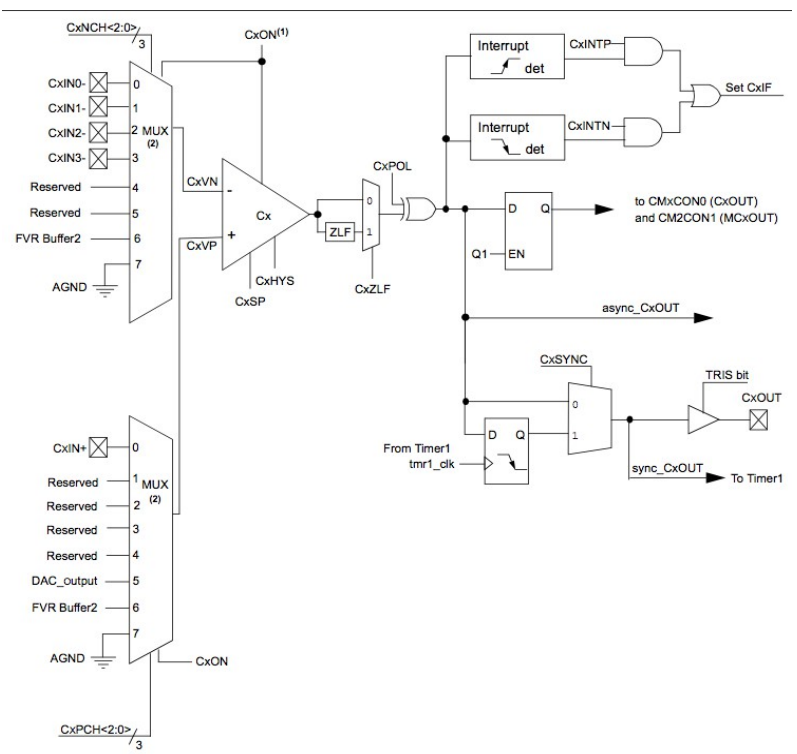


図 7.1: コンパレータのブロック図

動作原理

PIC16F1 ファミリが備える標準的なコンパレータ モジュールは、2 つの入力マルチプレクサ(各入力側に 1 つ)を備えています。FVR バッファ出力をオプションとして片側(モデルによっては両側)に配置し、必要に応じてデジタル出力をデバイスピンに印加できます。または内部で別のモジュールに接続できます。
コンパレータはしばしば DAC モジュールと組み合わせて使います。最もよく使うのは 5 ビット DAC ですが、より最近のモデルでは 8 ビット DAC を備えています。
(出力信号の立ち上がり/立ち下がりエッジの一方または両方で)割り込みイベントを生成でき、同じイベントをマイクロコントローラの復帰トリガとしても使用できます。

アプリケーション

コンパレータは多くのミクストシグナル回路の基本的構成要素で、以下を行います。

- 電源制御アプリケーションでの過電圧と低電圧のしきい値検出
- モータ制御アプリケーションでの過電流の検出
- バッテリ電圧低下の検出
- 多数のスイッチング電源回路方式でのピーク電流の検出等
- 多数の A/D 変換回路がコンパレータを使って構築されていますが、最も実用的なアプリケーションは内部 ADC モジュールを使っています。

制限事項

標準のコンパレータ モジュールは消費電力、コスト、速度のバランスを取っています。その結果、汎用マイクロコントローラが内蔵する一般的なコンパレータは、3 つのうちのいずれも優れていません。具体的には速度が 1 us 程度で消費電流は 50 uA 程度です。低消費電力モードがあっても、最小消費電流は 10 uA (typ.)程度です。言うまでもなく、これは XLP(超低消費電力)機能ではありません。スリープに移行する前にコンパレータを無効にする必要があるでしょう。

高速コンパレータ

スイッチング電源(SMPS)アプリケーション向けに設計された最近のデバイス(PIC16F17xx ファミリ等)では、コンパレータの応答速度が大幅に上がっており、50 ns に達しています(2 桁の改善)。これにより、電源制御アプリケーションは無理なく 500 kHz のスイッチング周波数を達成できます。

MCC が生成する API

MCC は通常、既定値の `CMPx_Initialize()` 関数と、コンパレータ出力をソフトウェアでテストする単純なマクロである `CMPx_GetOutputStatus()` を生成します。

ピン配置

全てのアナログ機能に関しては、コンパレータ入力のピン割り当てはデジタル機能の割り当てよりもはるかに限定されています。コンパレータの反転入力では外部ピンの選択肢がいくつかありますが、非反転入力での選択肢はピン 1 本に限定されています。コンパレータのデジタル出力は、PPS(使える場合)で外部に提供できます。

ホームワーク

- コンパレータの反応速度を調べ、A/D 変換に必要な時間と MCU 割り込み応答時間を合算して比較してみましょう。
- 通常のコンパレータと PIC16F17xx の高速コンパレータの反応時間を比較してみましょう。

オンライン リソース

『AN1427 - 携帯型 LED 照明向けの高効率ソリューション』
『AN1463 - NiMH Trickle Charger with Status Indication』
『AN1384 - Ni-MH Battery Charger Application Library』
『AN1138 - A Digital Constant Current Power LED Driver』
『AN874 - Buck Configuration High-Power LED Driver』
『AN700 - Make a Delta-Sigma Converter Using a Microcontroller's Analog Comparator Module』

DAC – D/A コンバータ

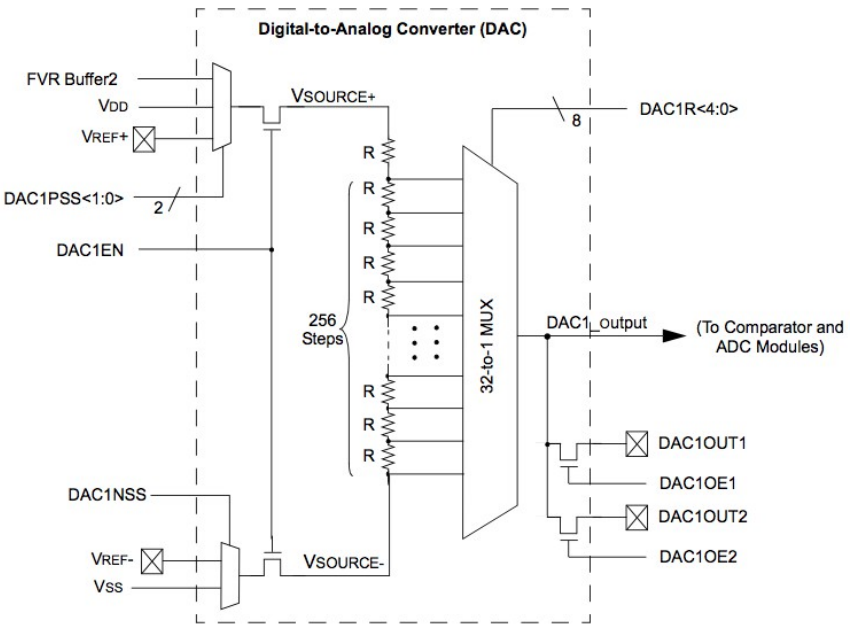


図 7.2: DAC のブロック図

概要

これまでずっと、PIC16 マイクロコントローラは非常に単純な DAC モジュールをアナログ コンパレータ ブロックの一部としてのみ搭載していました。分解能は 5 ビットに限定されており、しきい値を検出する簡単なプログラマブル回路を実装できる程度の柔軟性しか備えていませんでした。PIC16F170x/1x で DAC の分解能が 8 ビットに拡張され、PIC16F176x ファミリではさらに 10 ビットに拡張されました。

動作原理

全ての DAC モジュールはその分解能(N)にかかわらず、約 600 Ω の要素を 2^N 個持つシンプルな抵抗ラダーという基本設計を共有しています。アナログ マルチプレクサがラダー内でタップを選択します。

抵抗ラダーの上部と(場合によって)下部に、2 つの追加アナログ マルチプレクサがあります。入力の数はモデルによって異なりますが、選択した外部ピン(Vref+と Vref-)に接続する事ができます。
既定値ではラダー上部は内部でデバイスの Vdd に接続されており(下部は Vss に接続)、最も簡単な用途に対応しています。すなわち、デバイス電源電圧の分数としてアナログ値を出力します。
DAC 出力回路は通常 1 または 2 本のピンにアナログ電圧を出力できますが、専用チャンネルと任意の数のコンパレータ経由で ADC 等マイクロコントローラ内部の複数のアナログモジュールへも接続する事もできます。内部接続はピンの節約に貢献するだけでなく、ノイズを抑制するためにも非常に重要です。
DAC 出力ピンを有効にすると、デジタル I/O 機能は無効になります。つまり、ピンの入力値(PORTxbits.Rxy)を**読み出す**と、常に 0 が読み出されます。

アプリケーション

DAC ラダー上部のマルチプレクサを固定参照電圧(FVR)に接続できる場合、DAC を使って**絶対**電圧の出力を生成できます。これを使うと、バッテリ アプリケーションでバッテリ充電に関係なく正確なしきい値を得る事ができます。

電源制御アプリケーションでは、DAC ラダーの上部を外部(Vref+)ピンに接続できます。これをさらに適切なパーティション経由で AC 入力電圧(110～220V)に接続すると、DAC の出力をしきい値として使って入力正弦波を追跡し、抵抗挙動をシミュレート(つまり力率制御を提供)する事ができます。

コツ

> ラダーの上下両方を外部ピンに接続できる場合(一部のモデルでは上部のみ)、任意のアナログ入力信号に対応するプログラム可能な負荷として DAC を柔軟に使えます。実際、ラダーの上下をデバイスピンの Vref+と Vref-に接続すると、DAC をフローティング プログラマブル抵抗として使用できます。

制限事項

一般的なマイクロコントローラ設計の制約事項を考慮すると、DAC 入出力の使用には明らかな制約があります。

- Vref+が Vdd を超えてはいけません。
- Vref-が Vss を下回ってはいけません。
- 抵抗ラダー回路のインピーダンスを考慮します(常に非常に高い)。ほぼ全ての負荷を適用する前に、出力をバッファリングします。一部の PIC16F1 モデルが内蔵するオペアンプ モジュール(ユニティゲイン対応)を使えます。

MCC が生成する API

MCC が既定値の `DAC_Initialize()` 関数を含む **dac.c** ソースファイルを生成するように、`DAC` をごく簡単に設定できます。
正参照電圧として FVR が選択されている場合(ラダー上部)、MCC は現在のプロジェクトに FVR リソースを自動的に追加し、既定値の `SYSTEM_Initialize()` 関数内に初期化を含めます。
どちらかの出力ピン(`DACOUT1` または `DACOUT2`)が選択されている場合、MCC は[Pin Manager]ウィンドウで対応するピンをロックするように要求します。

例

簡単なテスト三角波形を生成する例を以下に示します。

```
/* Project:DAC Triangular Waveform
 * Device:PIC16F1619
 */
#include “mcc_generated_files.h”

main()
{
    uint8_t count=0;

    SYSTEM_Initialize();

    while(1)
    {
        for(count=0; count<=255; count++)
            DAC_SetOutput( count);
    }
}
```

ピン配置

DAC の出力は 2 本の固定ピンで多重化されています。その一方は、インサーキット シリアル プログラミング(ICSP)インターフェイスの CPD ピンと共有されています。ICSP 動作を妨げるような形でこのピンが**書き込まれ**ないようにするために対策する事が必要です。
また、DAC ラダーの上部と下部を外部ピン(Vref+と Vref-)に接続している場合、同じ位置を巡って競合が発生します。ラダーの 3 ヶ所全てにデバイス外部からアクセスする必要がある場合、唯一の方法は ICSP インターフェイスとの重複を認める事です。

ホームワーク

- 5 または 8 ビットの分解能を持つ標準 DAC モジュールと PIC12F752 の設計を比較してみましょう。電源制御アプリケーション向けの**フォーカス レンジ モード**は 9 ビット DAC と同等の分解能を提供している事を確認してみましょう。

FVR - 固定参照電圧

New

概要

マイクロコントローラに内蔵されたアナログ回路の大半は電源電圧(Vdd)を参照電圧として使えますが、独立した内蔵固定参照電圧モジュールがあると、多数のアナログ/ミクストシグナル アプリケーションにとって非常に便利です。PIC16F1 ファミリに実装された FVR モジュールは、安定性、低消費電力、低コストという相反する 3 つの目標の間で難しいバランスを取る事を目指しています。

動作原理

FVR は、比較的高いインピーダンスで 1 V を出力する基本的な参照電圧生成回路(バンドギャップ)です。続いて、これを使う回路(DAC、コンパレータ、ADC 入力、ADC 参照)からソースをデカップリングするバッファがあり、必要に応じてゲインを上げて(1x、2x、4x)、3 つ(1 V、2 V、4 V)のうちいずれかの出力電圧を生成します。

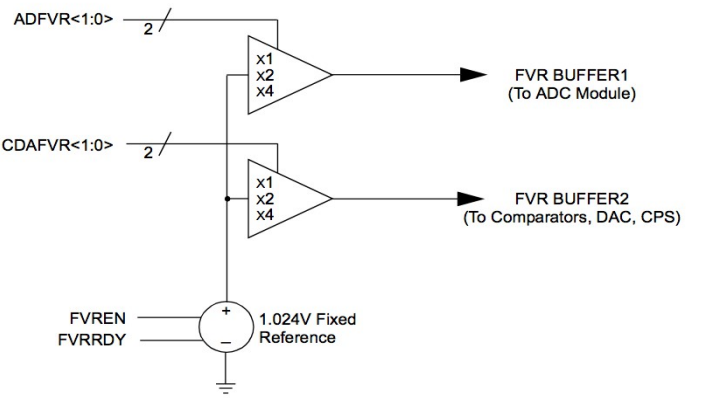


図 7.3: 固定参照電圧のブロック図

アプリケーション

固定参照電圧を備える事は全てのバッテリ アプリケーション、または電源電圧がレギュレートされておらず未知の場合に重要です。電圧レギュレータ(LDO)が使われていても、標準的な精度(+/-10%)ではアプリケーションにとって不十分の可能性がありますが、高精度なレギュレータはコストを上昇させてしまいます。

一般に、固定参照電圧は電源電圧レール上のノイズに対してより高い耐性を備えているため、より優れた(クリーンな)アナログ信号チェーンを生成します。

制限事項

概要セクションで述べた通り、最新世代の PIC16F1 デバイスに搭載された FVR は、安定性、低消費電力、低コストのバランスを取る事を意図して設計されました。その結果、FVR は厳密には XLP 仕様仕様に適合できていません。10～70 uA で動作するため、低消費電力モードに移行する前に無効にする必要があります。内部動作のために FVR を(暗黙的に)必要とする HFINTOSC、BOR、内部 LDO 等の他の回路についても検討が必要です。
さらに、目標精度を達成するには、FVR 回路(アンプ)には 30 us を超える安定化期間が必要です。
最後になりますが、温度と電源電圧が変化した場合の FVR の代表精度は 1～2%です。ADC 自体の分解能の制限(オフセットとゲイン誤差)と比べると、計測値への影響が大きいため、10 ビット ADC を使う場合はこれを考慮する必要があります。公平を期するために付け加えると、FVR の代わりに高精度の参照電圧生成器(外付けのディスクリート デバイス)を使う場合、精度と安定性は高くなりますがマイクロコントローラ自体よりも高くついてしまうでしょう。

MCC が生成する API

FVR の出力を ADC または DAC の参照電圧として使う場合、または ADC チャンネルあるいはコンパレータの入力として使う場合、MCC は自動的に FVR をプロジェクトに追加します。生成される API は、既定値の `FVR_Initialize()`と、FVR が実行中で、安定化期間が過ぎている事を確認するための `FVR_IsOutputReady()`の 2 つの関数のみです。

例

この短いシーケンス例は、無効にされていたデバイスが低消費電力モードから復帰する時に使います。

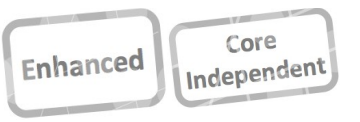
```
...
FVR_Initialize();               // enable the FVR
while( !FVR_IsOutputReady();    // ensure FVR output stable
sensor = ADC_GetConversion( InputChannel);
...
```

ピン配置

FVR の出力は**内部**接続するように設計されていますが、DAC モジュールを介して外部ピンに接続する事もできます。FVR を DAC の Vref+として使用し、DAC 出力を最大値に設定します。

ホームワーク

- Vdd = 3 V の時に 4x 出力バッファ乗算器を選択した場合、どのような結果になるか調べてみましょう。
- FVR の精度と外付け最小電圧生成器について調べてみましょう。
- FVR の精度を標準的な LDO 出力仕様と比較してみましょう。



ADC - A/D コンバータ

概要

10 ビット分解能の A/D コンバータ モジュールは、全ての PIC16F1 マイクロコントローラにほぼ標準として搭載されています。本書の執筆時点では、一部のモデル (PIC16F178x ファミリ)のみが 12 ビット分解能の ADC を備えています。
競合デバイスの中でこの ADC モジュールを際立たせている重要な 2 つの特長は、その**精度の高さ**と入力**チャンネル**の多さです。

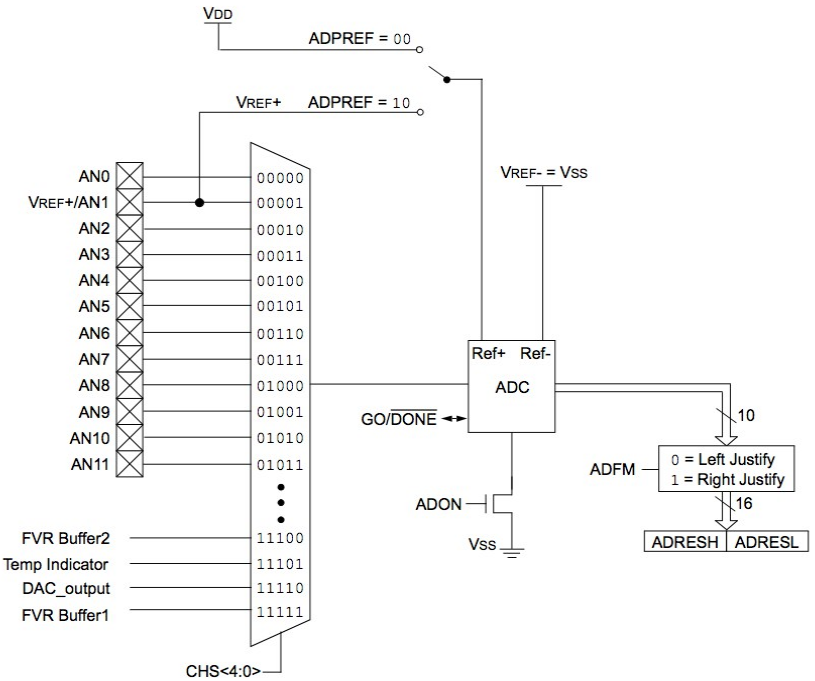


図 7.4: A/D コンバータのブロック図

精度と分解能はしばしば混同されますが、分解能が高い ADC が必ずしも高精度であるとは限りません。10 ビット ADC で±1 LSb 誤差の場合と 12 ビット ADC で±6 LSb 誤差の場合、どちらの方が精度が高いでしょうか(ヒント: ENOB を比較してみましょう)。
入力チャンネルの多さは、設計の柔軟性を高める特性の 1 つです。より多くのセンサ入力を接続するためだけでなく、配線を簡略化してアナログ信号とノイズの多いデジタルピンを分離して、アプリケーション性能を高めるためにも使えます。

動作原理

従来型の PIC16F1 ADC モジュールは逐次比較型レジスタ(SAR)であり、8 ビット マイクロコントローラの処理性能にとって十分な変換速度(約 100 ksps)を提供します。
サンプル/ホールド回路はシンプルな設計で、基本的に(MUX を介して)入力に直接接続された小型(内部)コンデンサです。(ADCON レジスタで)GO ビットがセットされるか、または**自動変換**トリガ入力を受信すると、S&H コンデンサが入力から切断され、変換プロセスが始まります。変換が完了すると、(13 ADC クロックサイクル後に)ADRES レジスタ(16 ビット幅)に結果が書き込まれ、必要に応じて割り込みを生成するフラグがセットされます。

ADC クロックはシステムクロックから派生させるか、または固定周波数(約 500 kHz)を持つ ADC 専用低消費電力内部 RC オシレータによって内部的に生成できます。後者の方法では、その他全てのクロック源が停止しているスリープ中も ADC の動作が可能であるため、精度は最も高くなります。変換中にマイクロコントローラをスリープに移行させられないアプリケーションでは、システムクロックから派生させる方法が一番です。この方法ではノイズは大きくなりますが、少なくとも ADC 内部動作のシーケンシングと**相関**するため、繰り返し性と安定性の高い結果が得られます。

自動変換トリガ

自動変換トリガ(ADCON2 レジスタの TRIGSEL フィールドで選択)は、変換と他の周辺モジュールの同期を達成するための鍵になります。この機能のおかげで複雑なイベントチェーン全体を構築できるため、複数のデジタルおよびアナログ周辺モジュールを接続してイベントを管理しながら、コアを純粋な監視役に専念させて、最小限の消費電力で高い応答性を実現できます。これこそが、コアから独立した禅の確立と言えるでしょう。

制限事項

ADC モジュールの入力インピーダンスは比較的低い(1 kΩ 未満)ため、外部信号源をこれに適合させるか(5～10 kΩ 以下)、またはバッファリングして重大なサンプリング エラーを回避する必要があります。この目的には、多くの PIC16F17xx ファミリ等が内蔵しているオペアンプ モジュールを使えます(本章で後述する「OPA」セクション参照)。
入力信号が電源電圧レンジの一部に限られる場合、計測分解能を高めるために外部 Vref+および Vref-入力が役に立つアプリケーションもあります。ただし、この方法が有効なのは 2 つの参照電圧の差が 2 V 以上である場合に限られます。このしきい値未満の場合、変換誤差の増加分によって分解能の向上が打ち消されます。

HCVD - 静電容量式センシング

HCVD は、最新の PIC16F1 ファミリが内蔵している ADC モジュールの機能です。HCVD はハードウェア静電容量式分圧器(Hardware Capacitive Voltage Divider)の頭字語で、静電容量式タッチセンシング、Metal Over Cap、圧力センシング、レベルセンシング等のアプリケーションで、静電容量式センシング向けに使われる基本テクノロジの 1 つをコアから独立して実装したもの(つまりハードウェア)を指します。

20 年以上前に Dieter Peter によって開発された基本的 CVD 手法は、並列接続された 2 つのコンデンサに適用される最も基本的な物理原則に基づくだけでなく、PIC マイクロコントローラの I/O(高電流駆動、方向制御)と ADC モジュールの S&H 回路の機能を活用しています。CVD 手法は全ての PIC16F1 マイクロコントローラですぐに使われるようになり、車載から家電製品、照明、電気メータまで、きわめて多岐にわたる分野の多数のアプリケーションで使われるようになりました。

当初の CVD アプローチには厳密なタイミング要件という制約があったため、しばしば設計者は最適化されたアセンブリ言語のルーチンを使う事を余儀なくされ、マイクロコントローラ コアにも高い負荷がかかっていました。HCVD テクノロジを採用した新しい ADC モジュールでは、CVD アクイジション全体が自動的に実行されるため、CPU がリアルタイム制約から解放されて、アルゴリズム開発ではフィルタ処理とより高レベルタスクに重点を置ける上、アプリケーション全体を C 言語で作成できます。

MCC が生成する API

MCC は、全ての ADC モジュール設定を 1 つのダイアログ ウィンドウにまとめています。このウィンドウでは以下を選択します。

- クロック源 - 正しい ADC クロック制限が守られているかどうか自動的にチェックします。
- 使用する参照電圧(FVR が選択されている場合) - 自動的に設定に追加されます。
- 入力チャンネル - それぞれのピンを設定して pin_manager 内で名前を付けます。
- 割り込み生成 - interrupts_manager モジュールを自動的にアプリケーションに追加します。

生成された関数は **adc.c** ファイル内に格納されます。これらの関数には、既定値の初期化に加えてアクイジションを開始する簡単な関数の ADC_StartConversion() (チャンネル選択と適切な(設定可能な)アクイジション タイミングを含む)、終了をチェックする ADC_IsConversionDone()、結果をフェッチする ADC_GetConversionResult()が含まれます。
または、ADC_GetConversion()を呼び出すと、1 回のブロッキング コールでシーケンス全体を取得できます。

MLA – Microchip アプリケーション用ライブラリ - タッチライブラリ

Microchip アプリケーション用ライブラリは、PIC16F1 アーキテクチャおよび XC8 コンパイラと互換性があり、包括的な**タッチセンシング フレームワーク**を提供しています。

例

以下の例は、約 10 行のコードで mTouch フレームワーク(MLA の一部)を使って 8 つの静電容量式ボタンを読み出し、デバイスの PortD に接続された 8 つの LED のうちの 1 つを有効化する方法を示しています。デバウンス、トグル、単一/複数選択パラメータは **mTouchConfig.h** ファイルで簡単に設定できます(ここでは記載していません)。

```
/*
 *  Project:         mTouch Demo
 *  Device:          PIC16F1937
 *  Board:           mTouch eval kit, 8-buttons daughter
 *  Requires:        MLA (version 12-15-2013) mTouch Framework
*/
#include "mTouch.h"

void ADC_ISR(void)
{
      mTouch_Scan();              // mTouch timer interrupt
}
```

```
void main(void)
{
    int8_t  i;

    SYSTEM_Initialize();            // init the I/Os
    mTouch_Init();                  // mTouch Initialization
    INTERRUPT_GlobalInterruptEnable();

    while(1)
    {
        if (mTouch_isDataReady())   // Button Pressed
        {
            mTouch_Service();       // Decoding
            for( i=0, LATD=0xFF; i<8; i++)
                if ( mTouch_GetButtonState( i) >=  MTOUCH_PRESSED)
                    LATD ^= (1<<i); // PORTD pins are connected to LEDs
        }
    }
} // main
```

ピン配置

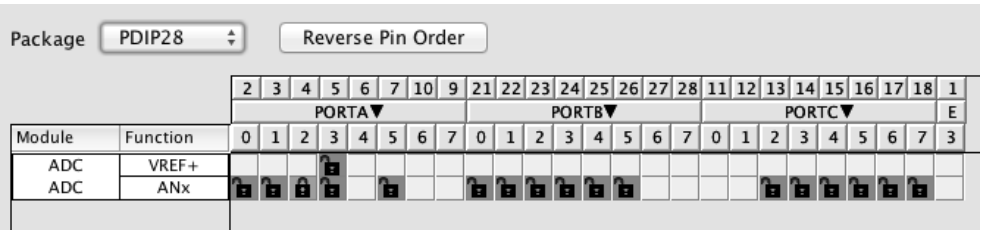


図 7.5: Pin Manager のアナログ入力選択

全ての PIC16F1 マイクロコントローラは、ADC モジュールへのアナログ入力チャンネルとして設定できるピンを多数備えています。
Vref+(と Vref-)の位置は固定されています。PIC16F1513 を使った例は、図 7.5 を参照してください。

ホームワーク

- A/D コンバータの精度と分解能の定義を比較してみましょう。分解能が高くなれば精度も上がるか考えてみましょう。そうでない場合、理由を考えてみましょう。

- マイクロコントローラがスリープ中でも ADC 機能が動作可能であり、計測完了時にコアを復帰させられるなら、どんな事ができる可能性があるか検討してみましょう。

オンライン リソース

『AN1478 - mTouch™センシング ソリューションのアクイジション手法: 静電容量式分圧器』
『AN1492 - マイクロチップ社の静電容量式近接検出設計ガイド』
『AN1334 - 堅牢なタッチセンシングの設計手法』
『AN1325 - mTouch™ Metal Over Cap テクノロジ』
『AN1334 - 堅牢なタッチセンシングの設計手法』
『AN1152 - Achieving Higher ADC Resolution Using Oversampling』
『AN1560 - Glucose Meter Reference Design』
『AN1626 - Implementing Metal Over Capacitive Touch Sensors』
『AN693 - Understanding A/D Converter Performance Specifications』
『AN699 - Anti-Aliasing, Analog Filters for Data Acquisition Systems』

温度インジケータ

New

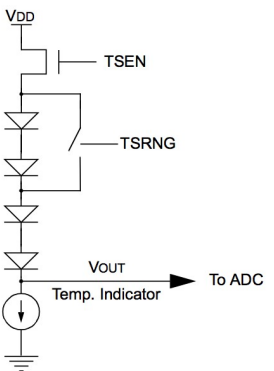


図 7.6: 温度インジケータのブロック図

概要

温度インジケータも、全ての PIC16F1 が内蔵するシンプルなアナログ モジュールです。このモジュールは温度しきい値の近似検出を備えていますが、正確な計測には校正が必要です。このため、「**センサ**」ではなく控えめに「**インジケータ**」と呼ばれているのでしょう。

動作原理

図 7.6 に示した通り、定電流源が直列に並んだ 4 つの接合部に接続されています。出力は ADC チャンネルの 1 つへ直接入力されています。
4 つの結合部のうちの 2 つは、3.6 V 未満で動作するアプリケーションに短絡できるだけでなく、結果の計測値から、Vdd からの依存性を除外する方法としても使えます。

アプリケーション

計測値はマイクロコントローラ周囲温度であり、適切に配置された場合、監視対象の組み込みアプリケーションの他の部分(モータ、パワーデバイス等)の温度の近似値と想定できます。

制限事項

Vdd からの計測値に直接依存するため、最終製品組み立ておよびテストラインで 2 点校正手順を実施しない限り、温度インジケータは比較的低精度(+/-5°C)しか達成できません。

MCC が生成する API

温度インジケータは制御レジスタを固定参照電圧と共有しているため、MCC では FVR 設定ダイアログ ウィンドウに温度インジケータの有効化オプションが含まれています (図 7.7 参照)。このため、生成される既定値の `FVR_Initialize()` 関数に選択内容が反映されます。

図 7.7: 温度インジケータ

ピン配置

温度インジケータは内部的に ADC モジュールに直接接続されています。デバイスピンにはアクセスできません。

ホームワーク

- 内蔵温度センサと外部(アナログまたはデジタル)センサとで利便性を比較してみましょう。その際、デバイス パッケージの熱抵抗とターゲットからの距離による影響も考慮してみましょう。
- 未校正の精度(通常±10°C)と外部デバイス(MCP9800 と MCP9700 ファミリ等)の予測精度を比較してみましょう。

オンライン リソース

『AN1333 - 内部温度インジケータの使用と校正』
『AN1001 - IC Temperature Sensor Accuracy Compensation』

OPA - オペアンプ

New

概要

PIC16F17xx ファミリにオペアンプが追加された事で、これらのデバイスのミクストシグナル能力が大幅に拡張されました。
このオペアンプはアナログ信号チェーンの大部分を内蔵する事でコストを削減し、多くの場合システム性能と信頼性を向上させます。

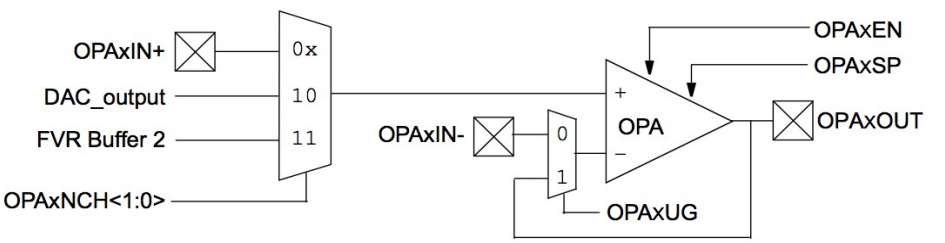


図 7.8: OPA のブロック図

動作原理

図 7.8 に示した通り、OPA モジュールはディスクリート オペアンプの 3 本のピン全てに加えて、内部接続オプションも備えています。
正(非反転)入力は、固定参照電圧(バッファ)または DAC 出力にアクセスできる小さなマルチプレクサを備えています。反転入力は内部的にはオペアンプ出力にのみ接続でき、入力信号のバッファリングとピンの節約が可能なユニティゲイン設定を作成します。

アプリケーション

内蔵オペアンプにより以下を実行する事で、ADC へのセンサ入力を調整できます。

- **アンチ エイリアシング** フィルタ処理
- 計測分解能を高めるための増幅、オフセットの加減算
- 過度に高いインピーダンスを持つ入力源のバッファリングによる計測の高速化
- 差動入力計測機能の提供

制限事項

内蔵オペアンプは Microchip 社の汎用ディスクリート オペアンプ MCP60x シリーズを基に作られており、レールツーレール入力を受け入れます。ゲイン帯域幅積は 2 MHz (typ.)を上回り、CMRR は 70 dB (typ.)です。 #TOIN 残念ながら、マイクロコントローラ構造(I/O ドライバ)に組み込みマルチプレクサを追加した結果、入力オフセットが 9 mV(最大)に達する場合があります(ただし製造時に校正)。また、オペアンプは XLP アプリケーションにとって決して理想的とは言えません。350 uA (5 V 時)の通常消費電流は、高温での最大値を考慮した場合、600 uA 以上になります。GBWP が低下する代わりに消費電力を抑制できる低ゲインモードが提供されていますが、わずかなメリットしか得られません。

MCC が生成する API

MCC はオペアンプ入力多重化オプションを豊富に提供しており、関連機能(FVR、DAC)を自動的に有効化し、ピンマネージャにおける I/O 割り当てを可能にします。**opa1.c** 出力ファイル内に生成されるのは、既定値の `OPAx_Initialize()` 関数のみです。

ピン配置

アナログ マルチプレクサのサイズと複雑さを抑制するため、オペアンプの入出力ピン位置は固定されています。オペアンプ有効時は対応する出力ピンが自動的に選択され、ピンのデジタル出力設定は無視される事に注意します。

ホームワーク

- 簡単な自動校正(オートゼロ)アルゴリズムを実装して、オペアンプ入力オフセットの検出と補償を行う方法を検討してみましょう。
- 同様に(場合によっては DAC または FVR を利用して)、低精度(低コスト)の外付け部品を使ってゲイン誤差を校正/補償する方法を検討してみましょう。

オンライン リソース

『AN1536 - Ultrasonic Range Detection』
『AN682 - Using Single Supply Operational Amplifiers in Embedded Systems』
『AN722 - オペアンプ トポロジと DC 仕様』
『AN723 – オペアンプの AC 仕様とアプリケーション』
『AN866 - Designing Operational Amplifier Oscillator Circuits For Sensor Applications』

第 8 章 算術演算サポート

AT – 位相角タイマ

概要

多くのモータおよび AC 電源制御アプリケーションでは、AC(正弦波)電源電圧またはモータの回転のどちらかの周期信号と同期して制御する必要があります。このような場合制御アルゴリズムのアプリケーションは常に位相角(周期の始まりに対する計測時の角度)を参照します。(ゼロクロス検出器、ホールセンサ、光学エンコーダ等によって)同期入力信号が使用可能になり、アプリケーション内の全てのタイミングはこの信号と計測された周期に依存します。しかし、代表的なマイクロコントローラ タイマがカウントするのは時間の単位であるクロックティックで、位相角とは間接的な関係しかありません。従って、2 つの領域(時間と角度)の間で変換が必要になります。計算式 8.1に示す簡単な線形比例によって、2 つの領域の関係が決まります。

a)

$$Phase(rad) = T * (2*PI / T_{period})$$

b)

$$T = Phase(rad) * (T_{period} / 2*PI)$$

計算式 8.1 - 時間から位相角への変換

上記で、時間はクロックティックまたは µs で計測し、角度はラジアンで表しています(度を使用する場合は PI を 180°で置き換えます)。
この計算式には好ましくない性質があり、計算を固定小数点演算(できれば整数演算)に限定するあらゆる努力をしても除算を使う必要があります。
周期が既知の場合(50 または 60 Hz の AC アプリケーション)、柔軟性が低下し、大量のメモリ空間を消費してもメモリルックアップ テーブルを使うのが一般的です。それでもサイズに制約があるため使えるのは限られた数の点のみであり、点の間では概算が必要です。

位相角タイマは、位相角の値に基づいてイベント(場合によっては割り込み)を生成する機能を提供する事で、このような懸念や制約を全て解消します。また、時間を角度に変換せずに検出した位相角の値を直接キャプチャできます。

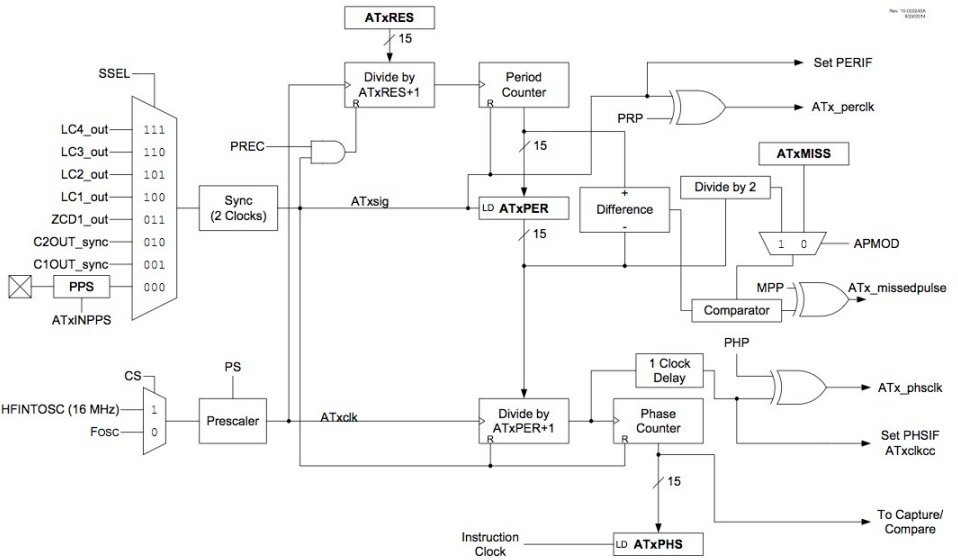


図 8.1: 位相角タイマのブロック図

動作原理

位相角タイマのブロック図(図 8.1 参照)は完全で詳細なものですが、これだけでは動作原理の説明には不十分です。以下では、より詳しく説明していきます。

位相角タイマは、**周期カウンタ**と**位相カウンタ**という 2 つのネストしたカウンタとして実装されています。周期カウンタは、指定の**分解能**で再読み込みするように設定します(例: 度を使う場合は 360 を使用)。連続した 2 つの入力間で実行された再読み込み回数を計測します(例: 360 カウントを 20 サイクル)。取得された値(例: 20)が入力周期の計測値です。次に、この値を位相カウンタのプリスケーラ値として使います。位相カウンタは正確なレートで動作します(この例では 1 ティックあたり 1 度で 20 カウントのサイクルが 360 回生成される)。

(CCP モジュールの処理と類似した方法で)メインカウンタ値と各種レジスタを比較する事で、絶対位相角値に基づく出力パルスまたは割り込みイベントを 3 つまで生成できます。同様に、1 サイクルあたり最大 3 回キャプチャを実行し、それぞれのイベントに対して絶対位相角値を取得できます。

言い換えると、位相角タイマモジュールは CPU に負荷をかける事なく、時間/角度領域の変換を即座に実行する上に、3 つのキャプチャ/コンペア モジュールに相当する機能を備えています。

アプリケーション

概要セクションで述べた通り、(場合によってはトライアックまたはその他のパワーデバイスと組み合わせて)負荷を制御するために**位相カット**手法を用いる AC 電源駆動アプリケーション(家電製品、照明、AC モータ制御等)で位相角タイマモジュールを適用するのが自然な考え方です。
AT を使うとアプリケーションの複雑さを解消でき、AC 周波数から完全に独立したソリューションを容易に設計できます。
モータ制御アプリケーションで AT を使うと、モータ回転速度とは無関係に**転流**時期を簡単に定義できます。
AT は、現時点では PIC16F161x ファミリのみが備えている比較的新しい周辺モジュールであるため、本来持つ可能性の大部分はまだ明らかにされていません。このモジュールの全く新しい意外な用途が次々と発見される事は間違いないでしょう。

制限事項

AT モジュールの分解能は設計上 10 ビットに制限されているため、約 1/3 度の分解能または 1 周期あたり 1024 ポイントです。しかし、アプリケーションの有効分解能は参照クロック(またはシステムクロック)によって制限を受ける場合があります。AT 内部で動作する 2 つのカウンタの積は、最終的に参照クロックの周波数によって決まります。従って、例えば 16 MHz のクロックでは 260 rpm (16 MHz/1024/60)までしか最高分解能(10 ビット)を使えません。この速度を下回ると、クロック プリスケーラを使用しない限り(1:8 まで可能)、モータが 1 回転する前に周期カウンタがオーバーフローします。このような場合、メインアプリケーションの制御アルゴリズムが必要とする高いクロック速度が、タイマにとって最適な分解能要件と相反します。
同様に、回転速度が高い場合、入力同期パルス間で周期カウンタが取得する値が小さくなり過ぎると、このような計測値に必然的に含まれる量子化誤差(1 ティック)がプリスケーラを使った値よりも大きくなり、計測値の実分解能を徐々に悪くします。従って、クロック速度と計測分解能を決める際は、タイマ設定を考慮する必要があります。

MCC が生成する API

MCC の位相角タイマ用ダイアログ ウィンドウは、AT の動作に必要な多数のレジスタ(3 つのキャプチャ/コンペアを含む合計 30 の 8 ビットレジスタ)の意味を理解するのに役立ちます。実際、このウィンドウを使わないと AT の設定は非常に困難です。
API は既定値の `ATx_Initialize()` 関数の他にはいくつかの簡単な関数で構成されており、ほとんどは説明を要しない 1 行のコードであるにもかかわらず、分解能と目的の比較角度の設定(`ATx_CC1_Compare_SetCount()` 等)、周期と角度の計測(キャプチャ)(`ATx_PeriodGet()`、`ATx_PhaseGet()` 等)にきわめて有用です。

ピン配置

AT モジュールはペリフェラル ピンセレクトを備えたデバイスのみで提供されているため、任意のデジタル I/O ピンに入力(同期、キャプチャ)と出力(コンペア)を自由に割り当てる事ができます。

ホームワーク

- AT タイマが備えているミッシングパルス機能について、そしてその機能のモータ制御での用途について調べてみましょう。
- ZCD モジュールを位相角タイマに接続し自動的に複数のイベントをつなぐ事で、トライアックのスイッチング シーケンスを生成してコアに負担をかける事なく変調とタイミング計測を自動実行する方法を考察してみましょう。

オンライン リソース

『AN958 - Low-Cost Electric Range Control Using a Triac』

PID - 算術演算アクセラレータ

概要

8 ビット マイクロコントローラに算術演算アクセラレーション エンジンを内蔵する事は、コアからの独立という考え方の本質に反して大幅に高性能な競合製品(DSP、16/32 ビット マイクロコントローラ等)とソフトウェア性能で張り合う事を示しているように見えるかもしれません。事実はそれとは異なり、この新しい周辺モジュールを正しいコンテクストでとらえる必

要があります。算術演算アクセラレータ モジュールは基本的に **PID エンジン**です。PID コントローラは現在モータ制御、電源、照明を含む、各種 8 ビット マイクロコントローラ アプリケーションで使われています。最近の調査によると、Libstock.com(人気のある組み込みアプリケーション共有サイト)から最もダウンロードされたサンプルコードは、ユーザが提供した PID ソフトウェア ライブラリでした。PID アルゴリズムは PIC16 の歴史と同じくらい古いアプリケーション ノートのいくつかでも取り上げています。

a)

$$\text{Output} = K_p E(t) + K_d \frac{dE(t)}{dt} + K_i \int E(t)\,dt$$

b)

$$\text{Output}[z] = \text{Output}[z-1] + K_1 E[z] + K_2 E[z-1] + K_3 E[z-2]$$

上記の式で、K_1～K_3 はそれぞれ

$$K_1 = K_p + K_d/T + K_i T; \qquad K_2 = -K_p - 2K_d/T; \qquad K_3 = K_d/T$$

T はサンプリング期間です。

計算式 8.2: PID の連続表記 a)と離散表記 b)

このようなアプリケーション全てでは、元の式(計算式 8.2参照)をラプラス変換したものから PID アルゴリズムが実装されるため、一度係数(Ki、Kp、Kd)が対応する K1、K2、K3 に変換されると、比較的容易な算術演算シーケンスとして解釈しやすくなります。また、E[z-1]と E[z-2]はループの前回の繰り返しでの入力(誤差)の値です。

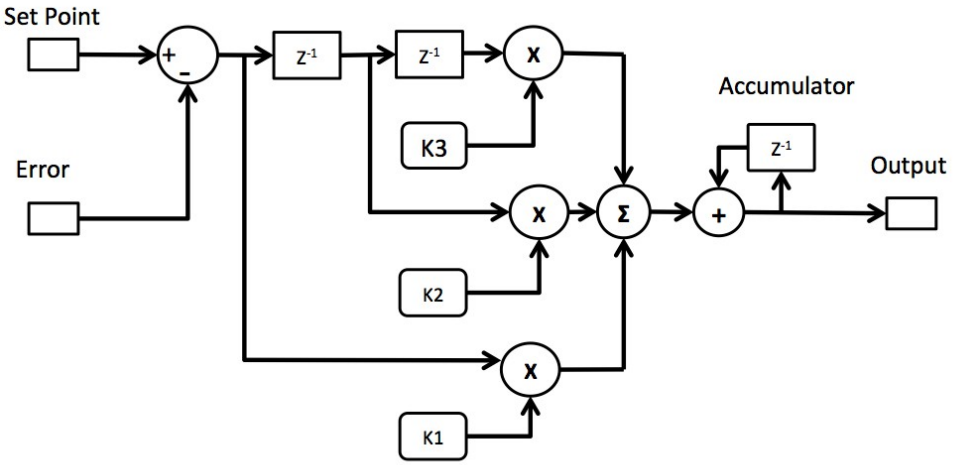


図 8.2: 離散 PID のブロック図

動作原理

図 8.2 を分析するか、または計算式 8.2から直接分かるのは、PID エンジンが(固定小数点の 16 ビット入力と 32 ビット出力)とアキュムレータ(35 ビット容量)を使った加算器と乗算器(A+B)*C の繰り返し使用で構成されている点です。

PIC16F1 で、これらの演算を PID 機能自体とは無関係に使えるようにしたのは至極当然でした。このため、入力セット A、B、C に対して算術演算アクセラレータによる実行が必要なのは 4 つの命令サイクルのみです。

PID アルゴリズム全体を実行する場合、算術演算アクセラレータはさらに効率的です。シーケンス全体の実行に必要なのはわずか 7 つの命令サイクルで、係数が変わらない限り、各ループで新しい誤差(入力)をモジュールに供給するだけです。

アプリケーション

簡単な 16 ビットの加算/減算を実行するために算術演算アクセラレータを使う事は、入力レジスタの読み込みと結果のフェッチに余分な命令サイクルを消費するため、労力に見合う結果は得られません。

16 ビットの乗算になると話は別です。16x16 ビットの符号付き乗算で 32 ビットの結果(と積算)を求める場合、数桁以上のサイクル(最適化された long 型演算を使う場合は約 300)を節約できます。PID アルゴリズム全体をソフトウェアで実装すると、1,000 近い命令サイクルになります。

制限事項

算術演算アクセラレータはマイクロコントローラ コアに不可欠な要素ではないため、XC8 C コンパイラは通常の算術演算を加速するために加算器/乗算器を使いません。ユーザは算術演算アクセラレータのレジスタを**手動** で読み込み、規定された演算シーケンスに従う必要があります。
複数の(ネストした)制御ループが必要な場合、算術演算アクセラレータを使うと大幅にサイクルを節約できます。それでもなお、係数を入れ換えるだけでなく PID エンジンが使う中間値(z、z-1、z-2)が保存され、繰り返しのたびに正しく復元される事を確実にする必要があります。

MCC が生成する API

MCC は、係数の読み込みと算術演算アクセラレータの設定という冗長な作業を既定値の `MATHACC_Initialize()` 関数内で全て処理します。標準的なアプリケーション制御ループ内で、ユーザは更新された入力のみを `MATHACC_PIDControllerModeResultGet()` 関数に渡し、積算された出力値を含むバッファを取得する必要があります。アキュムレータのサイズは 35 ビットであるため long 型の結果は返せませんが、**mathacc.h** ヘッダファイル内に 5 バイトの構造体が定義されています。そこでユーザは、出力値をスケーリングする最善の方法を選択し、アプリケーションに適した出力(アナログ/デジタル)を生成する事ができます。

ホームワーク

- 算術演算アクセラレータの速度と ACD の変換速度を比較してみましょう。上記の例で 32 MHz のクロックを想定すると、どちらがシステムのボトルネックとなる可能性が高いか計算してみましょう。
- 算術演算アクセラレータの性能を、dsPIC33F または PIC24H でソフトウェア PID を実行する場合の性能と比較してみましょう。
- (レジスタの読み込みと結果の書き出しを含む)積和演算の実行に必要なサイクル数を見積もってみましょう。
- デジタルフィルタ(FIR、IIR 等)アプリケーションで算術演算アクセラレータを使えるか検討してみましょう。

例

以下の例はシンプルなアプリケーションの main ループを示しています。この場合、PID アルゴリズムが ADC から入力を受け取り、PWM の出力を計算します。

```
/* Project:MathAcc Example
 * Device:PIC16F1619
 */
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/adc1.h"       // defines Sensor channel

void main(void)
{
    int SetPoint = 0x1234;
    MATHACCResult Output;
    SYSTEM_Initialize();    // initialize device, MATHACC, ADC and PWM3

    while (1)
    {
        int CurrentValue = ADC1_GetConversion( Sensor);

        Output = MATHACC_PIDControllerModeResultGet( SetPoint,
                                                CurrentValue);

        PWM3_LoadDutyValue( Output.byteHH + (Output.byteU<<8));
    }
}
```

オンライン リソース

『AN937 - PIC18 MCU の使用による PID コントローラの実装』

『AN964 - Software PID Control of an Inverted Pendulum Using the PIC16』

『AN1138 - A digital Constant Current Power LED Driver (using PID)』

算術演算関連のアプリケーション ノート

『TB040 - Fast Integer Square Root』

『AN1061 - Efficient Fixed-Point Trigonometry Using CORDIC Functions』

『AN821 - Advanced Encryption Standard Using the PIC16』

『AN942 - Piecewise Linear Interpolation on PIC12/14/16 Series Microcontrollers』

第 9 章 超低消費電力

XLP – 超低消費電力

8 ビット PIC マイクロコントローラの性能はこれまで以上に高く、その消費電力は**きわめて**低減されています。全てのアプリケーションが 10 年間バッテリ駆動できる必要はない一方で、エネルギ使用量をさらに抑える事から得られる多くのメリットがあります。電源回路の小型化、ノイズおよび発熱の低減、その結果としてのパッケージの小型化、さらに応用回路の小型化が望まれる特長です。

約 10 年前、全ての PIC マイクロコントローラ設計者が超低消費電力 **XLP (eXtreme Low Power)**基準と呼ばれる共通の消費電力目標値に熱心に取り組んでいました。当初の目標値には、スリープ(最低消費電力モード)中のデバイスで 100 nA という野心的な最大消費電流が含まれていました。さらに、ウォッチドッグとセカンダリ オシレータ(SOSC)の目標値はどちらも 800 nA でした。

現在、全ての PIC16F1 は当たり前のように、なおかつ大幅な余裕を持ってこれらの基準をクリアしています。

実際、デバイスのデータシートに記載された値が 1 桁近く低い(スリープ中で 20 nA、WDT と SOSC では 300 nA 未満)事はよくあります。さらに、マイクロコントローラの動的消費電力、つまりアプリケーション実行中の消費電流も同様に数桁削減されており、現在の一般的な値は 30 µA/MHz です。

これらの値を大型(16/32 ビット)アーキテクチャで達成するには、はるかに小さい(つまりリークの生じやすい)CMOS プロセスを使う必要があるため、達成できたとしても非常に困難です。実際、大幅に低い目標値(数 µA)に近づけるためだけに、RAM 内容の喪失と非常に長い復帰時間という代償を払ってでも大型のアーキテクチャで思い切った手法(ディープスリープ)を採用する事はよくあります。PIC16F1 マイクロコントローラでは、たとえ最低消費電力モード中であっても RAM メモリの内容は完全に保持され数 ms での復帰が可能です。

低消費電力モード

スリープ

スリープは当初からずっと、全ての PIC マイクロコントローラで特徴的な低消費電力モードです。当初、このモードはメイン システムクロックの停止を示すために使われており、独立したオシレータを持つウォッチドッグ タイマ以外のほぼ全てのアクティビティの停止も意味していました。
PIC16F1 世代のデバイスでは独立した内蔵オシレータの数が増えており、これに従って可能性も広がっています。スリープ中は依然としてシステムクロックが停止しており、そのためコアの命令実行も中断されていますが、多くの周辺モジュールは代替オシレータが設定されていれば動作を継続できます。このような周辺モジュールの例を以下に示します

- Timer1 - 16 ビットタイマで、セカンダリ オシレータ(SOSC)を使う場合、または非同期モードのカウンタとして使う場合
- A/D コンバータ - 専用の内部オシレータ(FRC)を使う場合
- 全ての CIP - いずれかのオシレータを直接使う場合(例: Fosc/4 の代わりの HFIntOsc)

低消費電力スリープ

低消費電力スリープ機能を、他のアーキテクチャの**ディープ**スリープと混同してはいけません。これは 5 V で動作する PIC16F1 のみに適用され、内部 LDO を使います。低消費電力機能を選択している場合、内部 LDO を低消費電力モードに切り換える事ができます(結果的に静止電流が低下)。RAM の内容は完全に保持されますが、フルパワーに戻るまで安定化時間が必要のため復帰には時間がかかります。

復帰

マイクロコントローラの低消費電力状態を終了させ、動作を再開させるイベントがいくつかあります。これには以下が含まれます。

- 外部リセット(MCLR)
- BOR リセット

- ウォッチドッグ タイマ
- 全ての外部割り込み
- システムクロックと非同期で動作する周辺モジュールによって生成される全ての割り込み

スリープから復帰させるトリガイベントがどれかにかかわらず、MCU はプログラムメモリ内の直後の命令から実行を再開します。
これは割り込みイネーブルビットがクリアされている(ディセーブルの)場合、通常であれば割り込みを生成する周辺モジュールにもあてはまります。

アイドル New

最初に PIC18 モデルに導入され、最近では PIC16F1 モデル(PIC16F183xx、PIC16F188xx ファミリ等)に導入された**アイドル**モードは、全てのクロックシステムとシステムクロックを使わずに動作するように設定された周辺モジュールの実行を維持しつつマイクロコントローラの実行を停止します。アイドルによる省電力は注目に値するものですが、スリープで達成される節減には全く及びません。

Doze New

Doze モードは最新の PIC16F1 ファミリに導入された機能で、消費電力をさらに細かく制御します。Doze モード中、コアの実行は継続しますが 1: 2 から 1: 256 の間で選択できるスケール係数でクロック源が分周されます。
その結果、選択されたスケール係数に比例して消費電力が低減し、一部のコア アクティビティと全ての周辺モジュールが動作を継続できます。

Doze 割り込みブースト New

Doze モード中に割り込みが生成されると即座にコアをフルスピードに復帰させるようにデバイスを設定できます。さらに、割り込み処理が終了した時点でコアを Doze モードに戻す事ができます。この 2 つのオプションを組み合わせて使うと、一種の割り込みブーストを生成できます。別の言葉で説明すると、低消費電力アプリケーションで要求されるクロック比(1: 8 = 4 MHz)でメインのアプリケーション ループを通常通りに実行しておき、割り込みルーチンのみがマイクロコントローラの最大性能(1: 1 = 32 MHz)とフルスピードを使えるように設定できます。

PMD - 周辺モジュール無効化

アプリケーションが**使っていない**周辺モジュールの消費電力は通常きわめて限定的ですが、最新世代の PIC16F1(PIC16F183xx、PIC16F188xx 等)が備えている **PMD** 制御レジスタを使うと、これらの周辺モジュールを全面的に**無効にする**事で消費電力を完全に除去できます。
無効にされた周辺モジュールはクロックと入力スティミュラスを受信しなくなります。マイクロコントローラ コアはこの周辺モジュールのレジスタを使えなくなり、割り当てられていた全ての I/O は解放されたのち、ピン優先度に従って次の周辺モジュール/オプションに戻されます。
周辺モジュールが再度有効になると、デバイス データシートに記載されている POR リセット条件に戻ります。

XLP の達成

アプリケーションで超低消費電力を達成するためには、単にスリープへ移行するより、ずっと多くの事項を考慮する必要があります。以下に挙げる項目は、デバイスの消費電力に大きく影響する可能性があるため、これをチェックしてスリープへ移行する前に無効にする事を推奨します。

- フローティングのままになっている I/O ピン
- I/O ピンから電流をシンクする外部回路
- 弱プルアップ オプション(有効な場合)
- LFintOSC を使っているモジュール (フェイルセーフ クロック監視)
- オシレータに直接接続したままになっている CIP(CWG または NCO と HFIntOSC)
- 内部参照電圧(FVR)を使用するアナログ モジュール(DAC、BOR、OPA、コンパレータ等) (使われている場合)

MCC が生成する API

MCC では、各モジュールに対して複数の初期化関数を簡単に生成できます。例えば、`SYSTEM_Initialize()`シーケンスで使われる既定値の初期化以外に、スリープへの移行前に使われる低消費電力構成を作成できます。

例

FVR モジュール用に新しい初期化関数を作成して「Sleep」という名前を付け、モジュール全体を無効にするように設定します。

```
// < MCC generated code >
void FVR_Sleep(void)
{
    // CDAFVR off; TSRNG Lo_range; TSEN disabled;
    //   FVREN disabled; FVRRDY disabled; ADFVR off;
    FVRCON = 0x00;
}
// < MCC generated code >
```

アプリケーションの電力消費を不必要に上昇させる可能性のある各モジュールを無効化する各関数を入手し、スリープ移行前にこれらを呼び出します。

```
// turn all things off (but the chosen wake up)
FVR_Sleep();
ADC_Sleep();
// ...

// go to lowest power mode
CLRWDT();
SLEEP();
NOP();
```

ホームワーク

- 厳密な低消費電力設計には十分に考え抜いた計画が必要です。モジュール間の相互依存性を明確に把握し、アプリケーションに最善の設定を見つけるために、デバイス データシートに記載された各周辺モジュールの「スリープ中の動作」セクションを常に確認しましょう。
- 状態変化割り込みの復帰機能を、外部(デジタル) イベントの検出に使った場合の CLC 割り込みと比較してみましょう。
- 必ずデータシートを参照して、アプリケーションの温度および電圧レンジ内での**最大**消費電力値を確認しましょう。
- Jack Ganssle による秀逸な低消費電力レポート(下記リンク参照)を読みましょう。このレポートは超低消費電力アプリケーションに関する多くの通説を払拭し、信頼性の高い(かつ現実的な)性能/バッテリ寿命のバランスを取るための優れたガイドラインを提供します。

オンライン リソース

http://www.ganssle.com/reports/ultra-low-power-design.html

『AN1416 - 低消費電力設計ガイド』

『AN1337 - MCP1640 を用いた DC 昇圧型コンバータにおけるバッテリ寿命の最適化』

『AN1088 - Selecting the Right Battery System for Cost-Sensitive Portable Applications While Maintaining Excellent Quality』

『AN1288 - Design Practices for Low-Power External Oscillators』

補遺 A - 周辺モジュールガイド

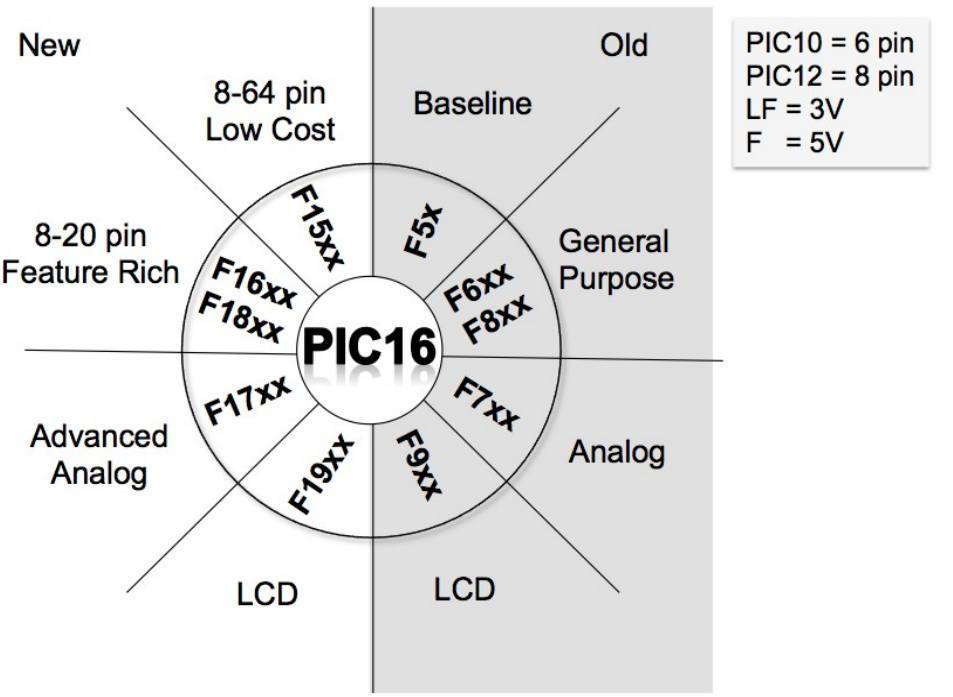
Peripheral Function Focus

Product Family	Pin Count	Program Flash Memory (KB)	Intelligent Analog: ADC (# of bits)	Comp	HSComp	DAC (# of bits)	HC I/O (mA)	OPA	PRG	SlopeComp	ZCD	Waveform Control: CCP/ECCP	10-bit PWM	16-bit PWM	PSMC (16-bit PWM)	COG	CWG	NCO	DSM
PIC10(L)F3XX	6	384–896 B	8										✓				✓	✓	
PIC16F527/570	20–28	1.5–3	8	✓				✓											
PIC1X(L)F150X	8–20	1.75–14	10	✓		5							✓				✓	✓	
PIC16(L)F151X/2X	28–64	3.5–28	10									✓							
PIC12LF1552	8	3.5	10																
PIC16LF1554/9	14–20	7–14	Dual 10										✓						
PIC16(L)F145X	14–20	14	10	✓									✓				✓		
PIC1X(L)F157X	8–20	1.75–14	10	✓		5								✓			✓		
PIC1X(HV)F752/53	8–14	1.75–3.5	10		✓	5/9	50	✓		✓		✓				✓			
PIC16(L)F182X/4X	8–20	3.5–14	10	✓								✓							✓
PIC1X(L)F1612/3	8–14	3.5	10	✓		8					✓	✓					✓		
PIC16(L)F161X	14–20	7–14	10	✓		8	100				✓	✓	✓				✓		
PIC16(L)F170X/1X	14–40	3.5–28	10		✓	5/8		✓			✓	✓	✓			✓		✓	
PIC16(L)F176X/7X	14–40	7–28	10		✓	5/10	100	✓	✓		✓	✓	✓	✓		✓			✓
PIC16(L)F178X	28–40	3.5–28	12		✓	5/8		✓				✓			✓				
PIC16(L)F183XX	8–20	3.5–14	10	✓		5						✓	✓				✓	✓	✓
PIC16(L)F188XX	28–40	7–56	10*	✓		5					✓	✓	✓				✓	✓	✓
PIC16(L)F193X/4X	28–64	7–28	10	✓								✓							
PIC18(L)FXXK20	28–40	8–64	10	✓								✓							
PIC18(L)FXXK22	20–80	8–128	10	✓		5						✓							
PIC18(L)FXXK40	28–64	16–128	10*	✓		5					✓	✓	✓				✓		✓
PIC18(L)FXXK42	28–64	16–128	10*	✓		5					✓	✓		✓			✓	✓	✓
PIC18(L)FXXJ94	64–100	32–128	12	✓								✓							
PIC18(L)FXXK50	20–40	8–32	10	✓		5						✓							
PIC18FXXK90	60–80	32–128	12	✓								✓							

Product Family	Timing and Measurements**: AngTMR	HLT (8-bit)	PSMC (16-bit)	16-bit PWM (16-bit)	NCO (20-bit)	SMT (24-bit)	RTCC	TEMP	Logic and Math: CLC	MULT	MathACC	Safety and Monitoring: CRC/SCAN	HLT	WWDT	Communications: EUSART/AUSART	I²C™/SPI	USB with ACT	LIN Capable	User Interface: mTouch® Sensing	HCVD	LCD	Low Power and System Flexibility: HEF	PPS	IDLE and PMD	DOZE	XLP
PIC10(L)F3XX					✓			✓	✓										✓							
PIC16F527/570																			✓							
PIC1X(L)F150X					✓			✓	✓						✓	✓		✓	✓			✓				✓
PIC16(L)F151X/2X								✓							2	2		✓	✓			✓				✓
PIC12LF1552								✓								✓			✓	✓		✓				
PIC16LF1554/9								✓							✓	✓		✓	✓	✓		✓				✓
PIC16(L)F145X								✓							✓	✓	✓	✓	✓							✓
PIC1X(L)F157X				✓				✓							✓			✓	✓			✓	✓			
PIC1X(HV)F752/53		✓											✓						✓							
PIC16(L)F182X/4X								✓							✓	✓		✓	✓							✓
PIC1X(L)F1612/3		✓				✓		✓				✓	✓	✓					✓			✓				✓
PIC16(L)F161X	✓	✓				✓		✓	✓		✓	✓	✓	✓	✓	✓		✓	✓			✓	✓			✓
PIC16(L)F170X/1X					✓			✓	✓						✓	✓		✓	✓			✓	✓			✓
PIC16(L)F176X/7X		✓		✓				✓	✓				✓		✓	✓		✓	✓			✓	✓			✓
PIC16(L)F178X			✓					✓							✓	✓		✓	✓							✓
PIC16(L)F183XX					✓			✓	✓						✓	2		✓	✓				✓	✓	✓	✓
PIC16(L)F188XX		✓			✓	✓		✓	✓			✓	✓	✓	✓	2		✓	✓	✓			✓	✓	✓	✓
PIC16(L)F193X/4X								✓							✓	2		✓	✓		✓					✓
PIC18(L)FXXK20										✓					✓	✓		✓	✓					✓		✓
PIC18(L)FXXK22							✓			✓					2	2		✓	✓					✓		✓
PIC18(L)FXXK40		✓								✓		✓	✓	✓	✓	✓		✓	✓	✓			✓	✓	✓	✓
PIC18(L)FXXK42		✓		✓	✓	✓	✓		✓	✓		✓	✓	✓	2	2		✓	✓	✓			✓	✓	✓	✓
PIC18(L)FXXJ94							✓			✓					4	2	✓	✓	✓		✓		✓	✓		✓
PIC18(L)FXXK50										✓					✓	✓	✓	✓	✓					✓		✓
PIC18FXXK90							✓			✓					2	✓		✓	✓		✓			✓		✓

補遺 B - PIC16F + 2/3 桁型番の解読リング

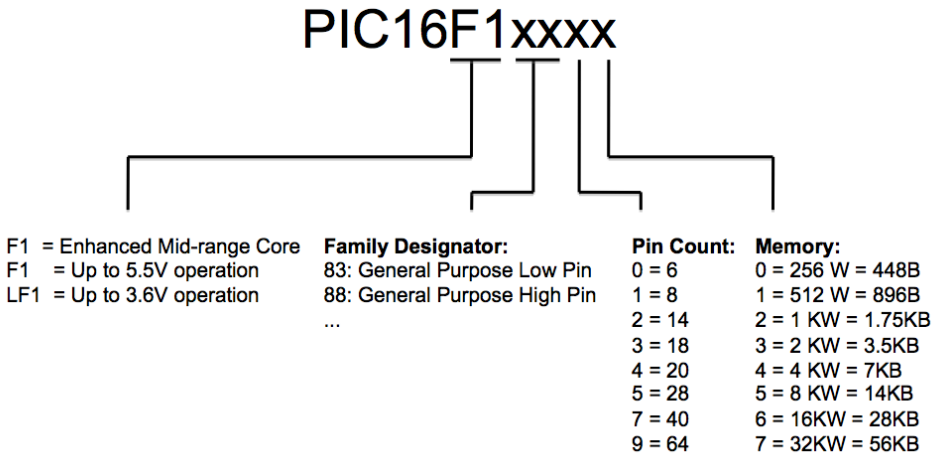
PIC16 マイクロコントローラの製品番号割り当て方式は、ごく限られた関係者以外にとっては不可解です。大半の製品番号は強力な乱数アルゴリズムを使って生成されたとか、Keeloq 暗号化アルゴリズムが使われたのだろうと考える人もいますが、このような通説を一掃するために以下の図表を作成しました。
この解読リングは、1989 年から 2014 年までの間に導入された PIC16 シリーズの PIC マイクロコントローラにあてはまります(例外あり)。



PIC16F5x で始まる製品はオリジナルの PIC ベースラインコアを備えています。
PIC16F の後に 3 桁が続く製品は、ミッドレンジのコアを備えています。
PIC16F1 の後に 3 桁が続く製品は、エンハンスト ミッドレンジのコアと CIP を備えています。

補遺 C - PIC16F1 + 4 桁型番の解読リング

2014 年後半に PIC16 ファミリのデバイス数が 1,000 を超えたため、新しい製品番号割り当て方式が採用されました。
これは最大 5 桁の英数字によって識別できます。



6 ピンと 8 ピンは現在、製品番号の 4 桁目に標準で含まれているため、PIC10 と PIC12 で始まる製品番号は廃止されています。
この新しい方式により、デバイスのピン数とメモリ容量を勘で推測される事がほとんどなくなると期待されています。2 桁の数字の**ファミリ識別**コードは存続しており、過去の世代との類似性(9: LCD、7: アナログ、5: エントリレベル、6 と 8: 汎用等)を維持するために今後も継続されると**考えられます**。

アルファベット順索引