組み込みシステムとPSoC演習

PSoC Embedded Lab and Experiment

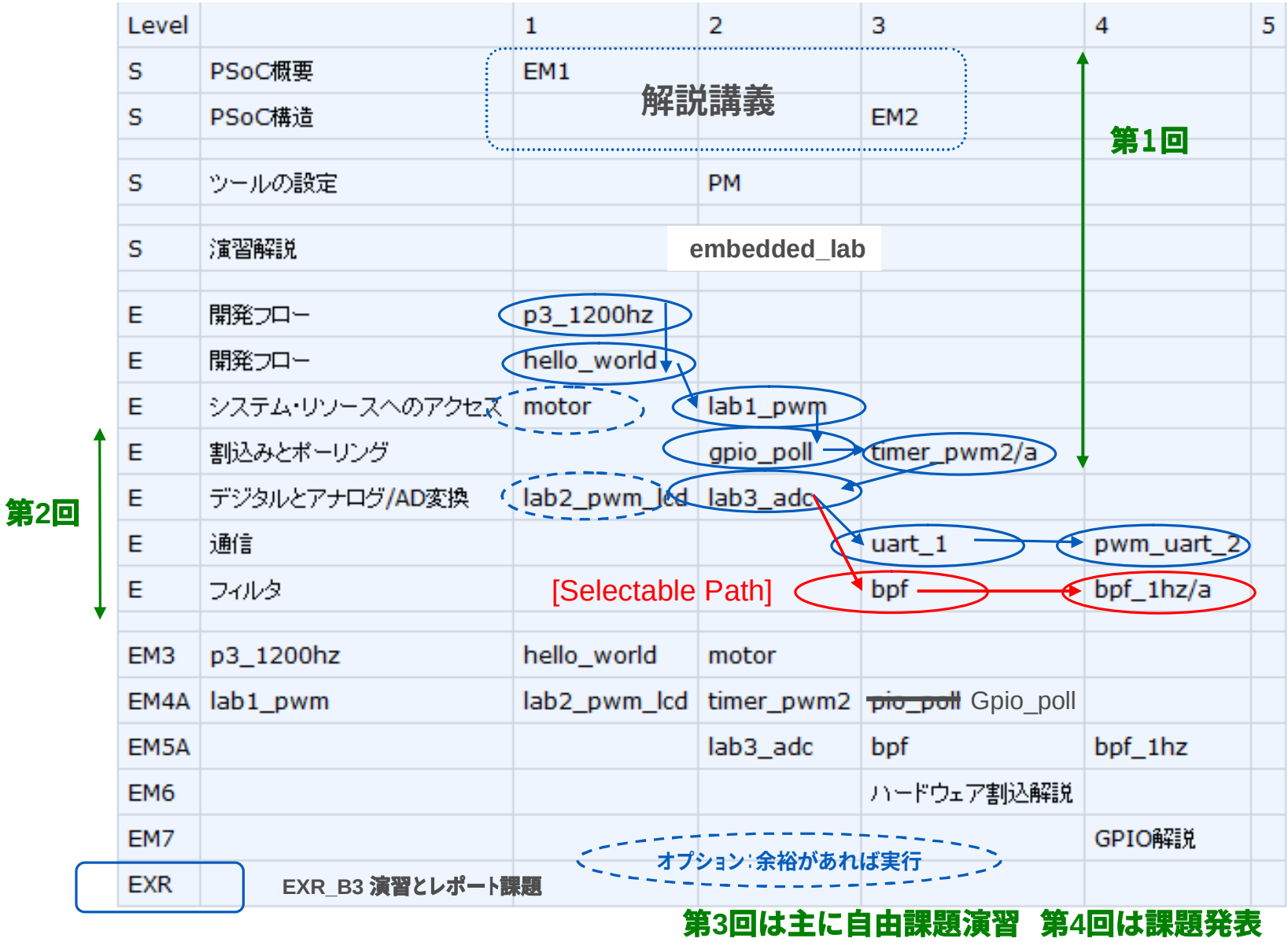
Embedded Lab Material for PSoC Experiment V1.00
May 9th. 2011
embedded_lab.PPT (118Slides)

Renji Mikami
Renji_Mikami@nifty.com

演習の全容B３

S: 講義資料　E: 演習資料

Level: 数字がおおきいものは難度が高い



演習の全容B２

S: 講義資料　E: 演習資料

Level: 数字がおおきいものは難度が高い

Level		1	2	3	4	5
S	PSoC概要	EM1				
S	PSoC構造			EM2		
S	ツールの設定		PM			
S	演習解説		embedded_lab			
E	開発フロー	p3_1200hz				
E	開発フロー	hello_world				
E	システム・リソースへのアクセス	motor	lab1_pwm			
E	割込みとポーリング		gpio_poll	timer_pwm2/a		
E	デジタルとアナログ/AD変換	lab2_pwm_lcd	lab3_adc			
E	通信			uart_1	pwm_uart_2	
E	フィルタ			bpf	bpf_1hz/a	
EM3	p3_1200hz	hello_world	motor			
EM4A	lab1_pwm	lab2_pwm_lcd	timer_pwm2	~~pio_poll~~ Gpio_poll		
EM5A			lab3_adc	bpf	bpf_1hz	
EM6				ハードウェア割込解説		
EM7					GPIO解説	
EXR						

第1回

第2回

第3回

EXR_B2 演習とレポート課題

オプション：余裕があれば実行

第4回は主に課題演習

コンピューターとシステム

(マイクロ)プロセッサ･ユニットは、メモリ、I/O、その他の周辺回路とともに、(マイクロ)プロセッサ･システムを構成する。

-このプロセッサ、システムが外部のノン･デジタルな情報を収得し、外部に接続されたさまざまな機器を制御できるように機能を拡張したものが、より大きなシステムで、その応用により◯◯システムと呼ばれる。

-ロボットやビデオカメラ、航空機や自動車もこのようなシステムである。

-このような◯◯システムの大規模なものは、たとえば自動車では、数十から百個以上のプロセッサ･システムを搭載しているものもある。

-これらの大規模で複雑なシステムでは、複数のプロセッサ･システムが特定の部分の制御を担当し、他のプロセッサ･システムや中央の集中管理プロセッサ･システムと通信を行いながら、全体としては機能的な分散処理を行うように構成されている。

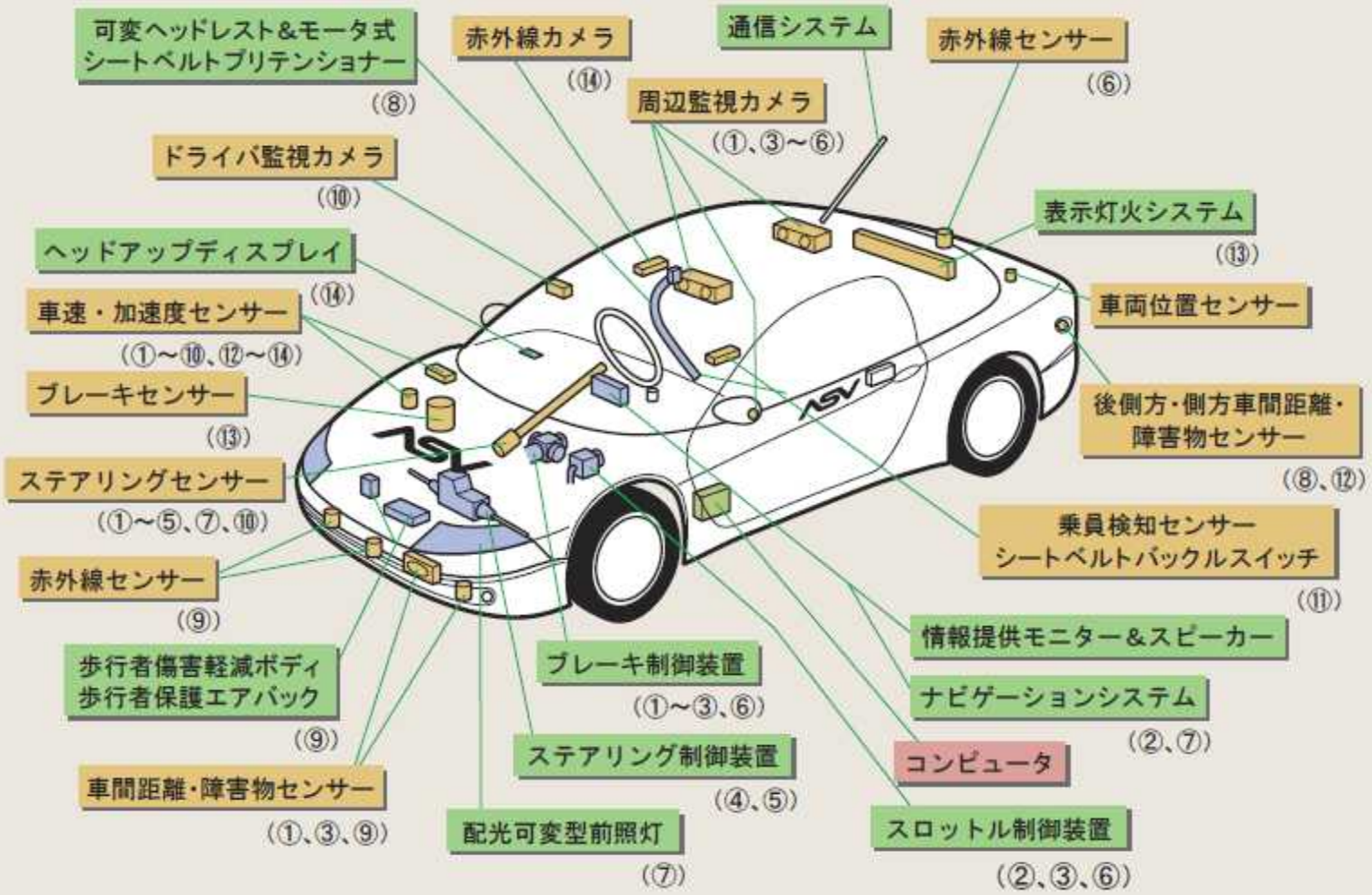
-個々の機能処理を担当するプロセッサ･システムの最もシンプルな典型は、組み込みシステムに見ることができる。

先進安全自動車（ASV）のイメージ

コンピュータ センサーの情報から危険などを認知・判断して制御装置につたえます。

センサー 走行環境や車両状態の認知に使います。

制御装置等 コンピュータからの情報をもとに情報提供や警報を行い、さらにドライバーの操作を助けます。



■第2期ASVの代表的システム

①前方障害物衝突防止支援システム

②カーブ進入危険速度防止支援システム

③ブレーキ併用式車間距離制御機能付定速走行装置（全車速域制御）

④車線逸脱防止支援システム

⑤車線維持支援装置

⑥車両死角部障害物衝突防止支援システム

⑦配光可変型前照灯

⑧被衝突予知むちうち傷害低減システム

⑨歩行者傷害軽減ボディ&歩行者保護エアバック

⑩居眠り警報装置

⑪全席シートベルト着用勧告装置

⑫後側方・側方情報提供装置

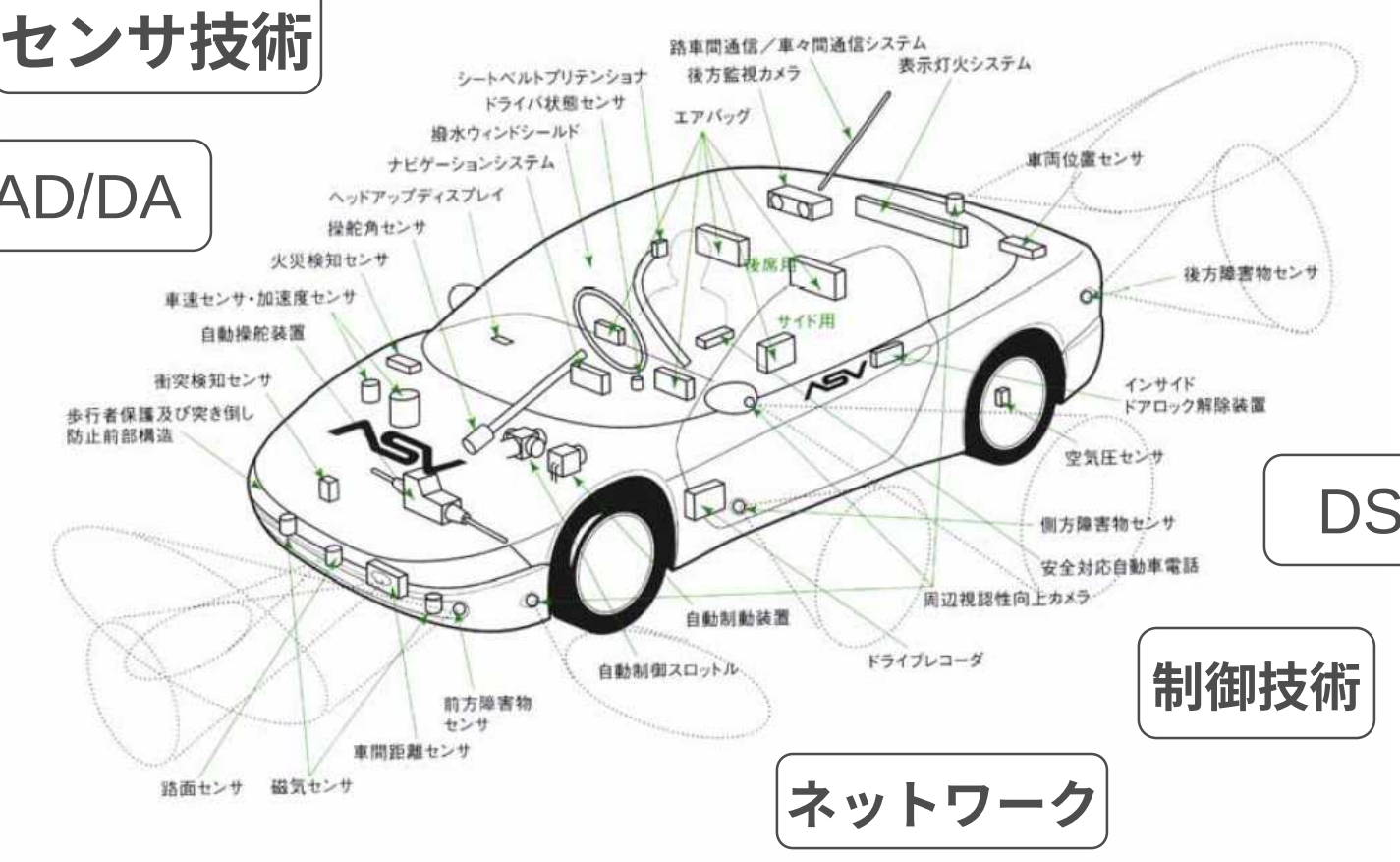
⑬緊急制動情報提供装置

⑭夜間前方歩行者情報提供装置

自動車“システム”の構成要素

センサ技術

AD/DA



DSP

制御技術

ネットワーク

引用:国土交通省先進安全自動車第三期中間報告会資料

組み込みシステム

組み込みシステムの例をあげると、炊飯器、洗濯機、などの比較的シンプルなものから、カメラやテレビなどの複雑高機能なものまで、枚挙の暇がないほどであるが、(たとえば炊飯器の例をとって考えてみると)そこには以下のような組み込みシステムの基本的な要件のいくつかを見出すことができる。

--時計機能

--外部の事象変化を察知する機能

--外部の装置を駆動する機能

--通信の機能

いずれの機能もプロセッサ単体では実現することができないが、デジタルおよびアナログの電子回路をコントロールすることによって機能を実現することができる。

-この何通りかの電子回路とプロセッサがこの電子回路をコントロールする仕組みがわかれば、組み込みのシステムを容易に理解できる。

ソフトとハードの接点: (外部)レジスタ

-この仕組みは、ソフトウェアとハードウェアの接点部分でソフトとハードが連携して動作することによって実現される。具体的には、レジスタと呼ばれる記憶回路(フリップ・フロップ)のデータをプロセッサがソフトウェアで読み書きし、同時にハードウェアもこのデータへの読み書きを行っている。

-プロセッサは外部の電子回路の制御情報をプログラムによってレジスタに書き込む。ハードウェアは、このレジスタのデータを読み込んで所定の動作を行う。

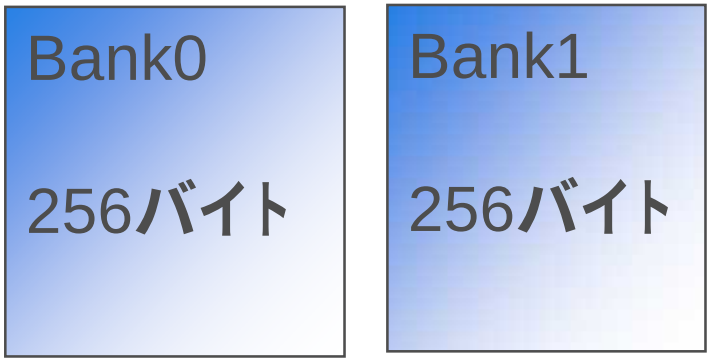
-同時に外部の電子回路は、得られた情報をレジスタにデータとして書き込み、プロセッサはこのレジスタのデータを読み込むことによって外部ハードウェアからの情報をとりこむことができる。

-プロセッサは得られた情報をもとにプログラムで処理と判断を行い、再び外部ハードウェアを制御するためにレジスタにデータを書き込む。

-レジスタはそのアドレス(番地)ごとに機能がきめられている。またレジスタ1bitの値は、0または1である。数を表す場合は複数のビットが用いられる。

ここで述べられているレジスタとは、プロセッサの外部にあるメモリであるが、プログラムを格納するメモリとは別に設置され、またプロセッサの内部にある(汎用)レジスタと区別する意味で特に(外部)レジスタと呼称している

PSoCレジスタとM8 MPU



PSoC機能設定レジスタは2バンク構成で各256バイトのアドレスを持ちます．以下の機能設定を行います

M8Cレジスタ(Fレジスタ) RAMページング
割り込みコントローラ設定
デジタルブロック設定
アナログブロック設定
GPIO設定
オシレータ･PLL設定

バンクの切り替えはFレジスタで行います．

M8 MPU のレジスタ構成

全てのインターナルレジスタは8ビット．ただし，PCは16ビット

A,X,PCレジスタはリセット後，0クリアされます

SPは次のスタック位置を指します．0XFFを指している状態でPUSH命令を実行すると0X00へロールします．

(内部) レジスタとメモリ

-コンピュータ・システムでは記憶装置が必須である。

インストラクション・フロー型(データフローに対する)コンピューターは、

命令プログラムを逐次実行するためこの命令を格納するための記憶装置が必要である。また計算結果を一時格納するための記憶装置が必要である。

-このような記憶機能は、通常メモリとよばれるハードウェアで実現される。

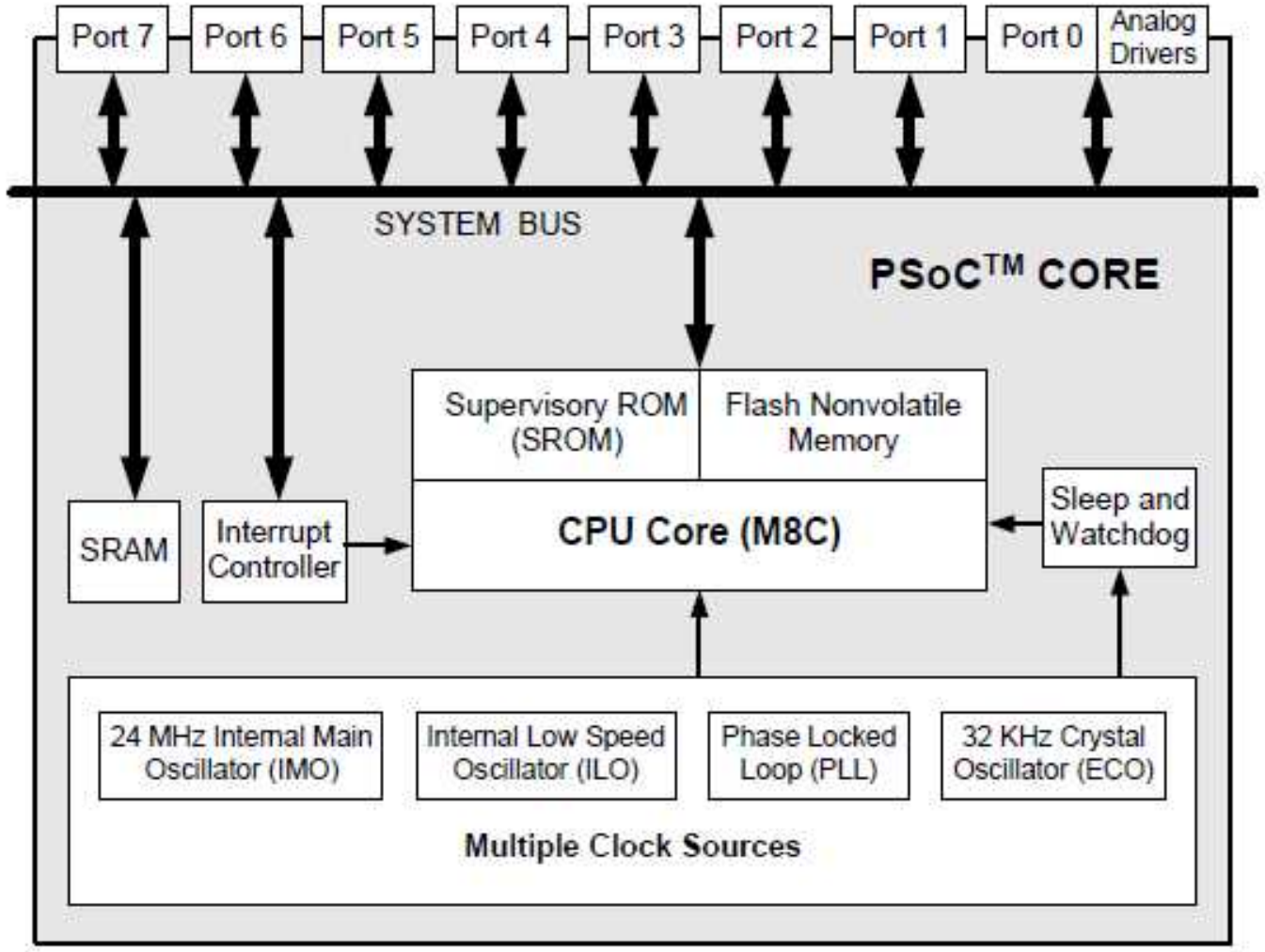
メモリには、固定的な制御用プログラムを格納するROM(Read Only Memory)と動作中にリードライトできるRAM(Random Access Memory)があり、高速で読み書きする必要があるので高価な半導体メモリが使用される。また半導体以外のメディアに大容量の記録を行うハードディスクなどはストレージと呼ばれ低速ではあるが価格が安い。

記憶装置は読み書きの速度が性能に大きく関与する。最も高速なものはプロセッサ内部でアキュムレータと最速のマシンサイクルで読み書きできる内部レジスタ(汎用レジスタ)である。続いて高速なものは,ハードウェアとのインターフェースを行う外部レジスタである。それに続くものは、プログラムを格納し、計算結果を一時格納するメモリとなるのが一般的である。

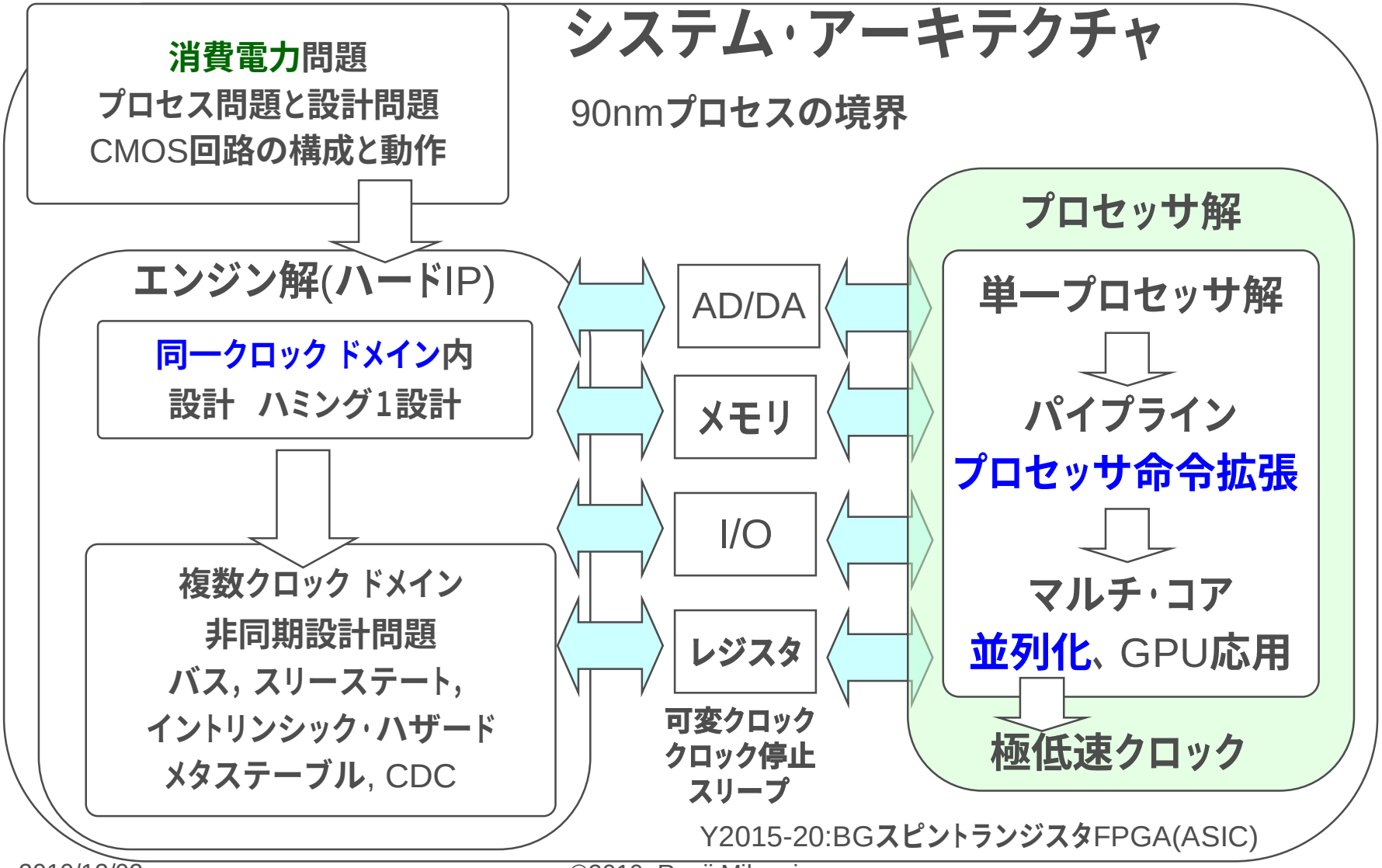
しかし割り込みが発生したときに汎用レジスタの内容を退避させるスタックなどは、メモリに置かれた場合に処理速度のネックとなる場合がある。割り込みの処理を高速化するために汎用レジスタを複数セット用意したり、スタック専用の高速メモリを用意する場合もある。またバスを経由したメモリのアクセスでは高速化に限界がある場合、キャッシュ・メモリを間に入れたり、外部のハードウェアと直接配線したレジスタをプロセッサの内部におきプロセッサの命令を拡張して外部ハードウェアを直接制御したりする場合もある。

レジスタや記憶装置とのアクセス問題は、プロセッサ・システムを構築する上での難問の一つで、そのためにさまざまなシステム・アーキテクチャが存在する。

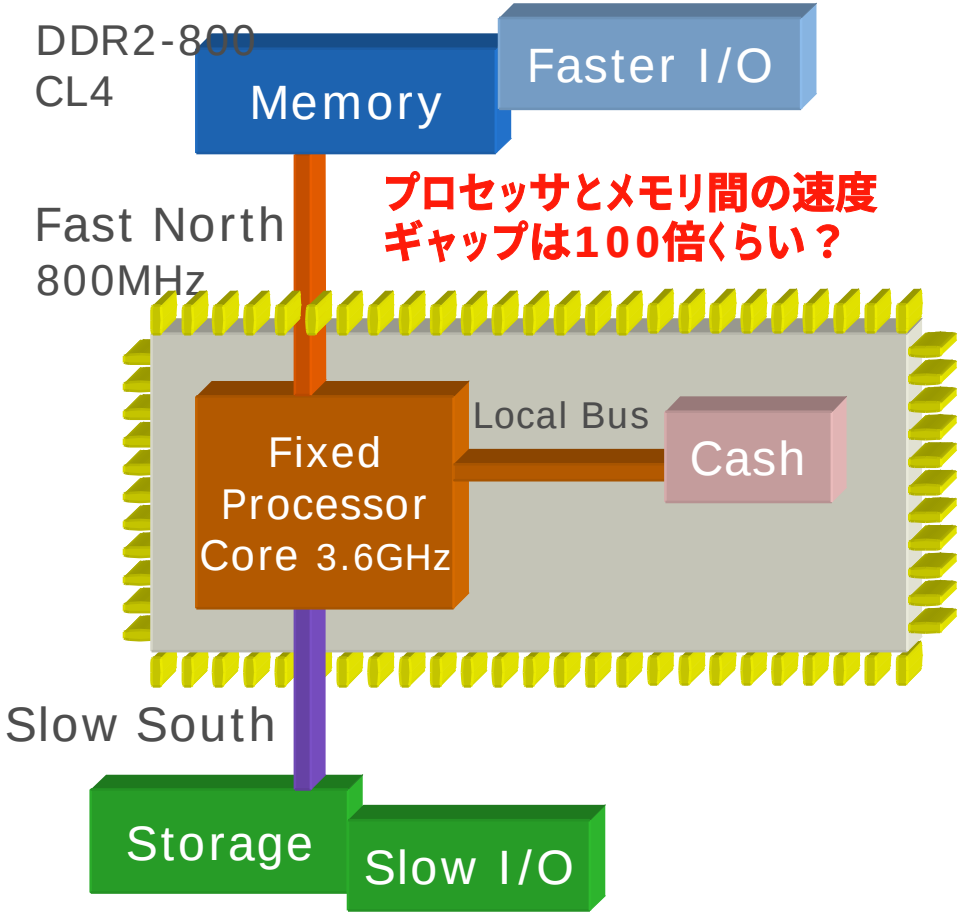
PSoCコアトップレベルブロック図



アーキテムチャレベルでの高速ローパワー化



プロセッサ・メモリ・ボトルネック

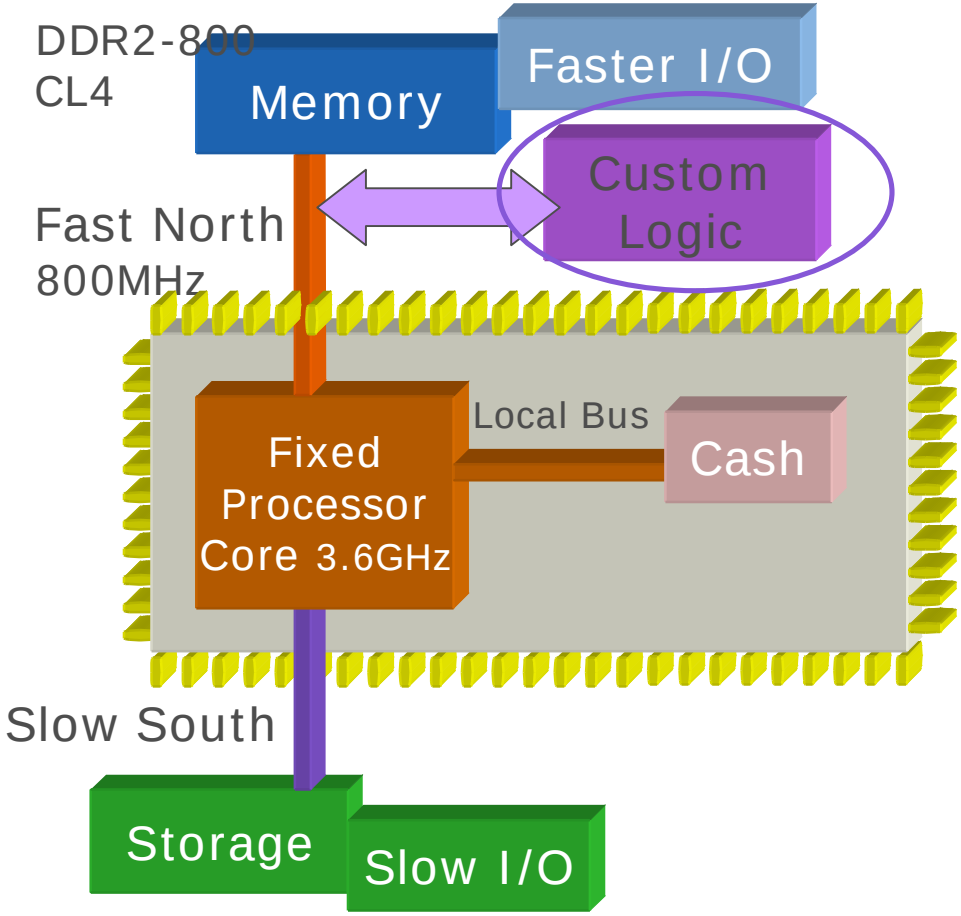


プロセッサ性能は飛躍的に高く
なりましたがコンピュータシステム
全体で見るとMPU性能ほどには
パフォーマンスは上がっていません

シングル･プロセッサの動作
クロックが3.6GHzあっても
外部高速インターフェース
バス速度は1/5に低下します

高速インターフェースバスに
接続されたメモリ･レイテンシで
さらに1/4に低下します

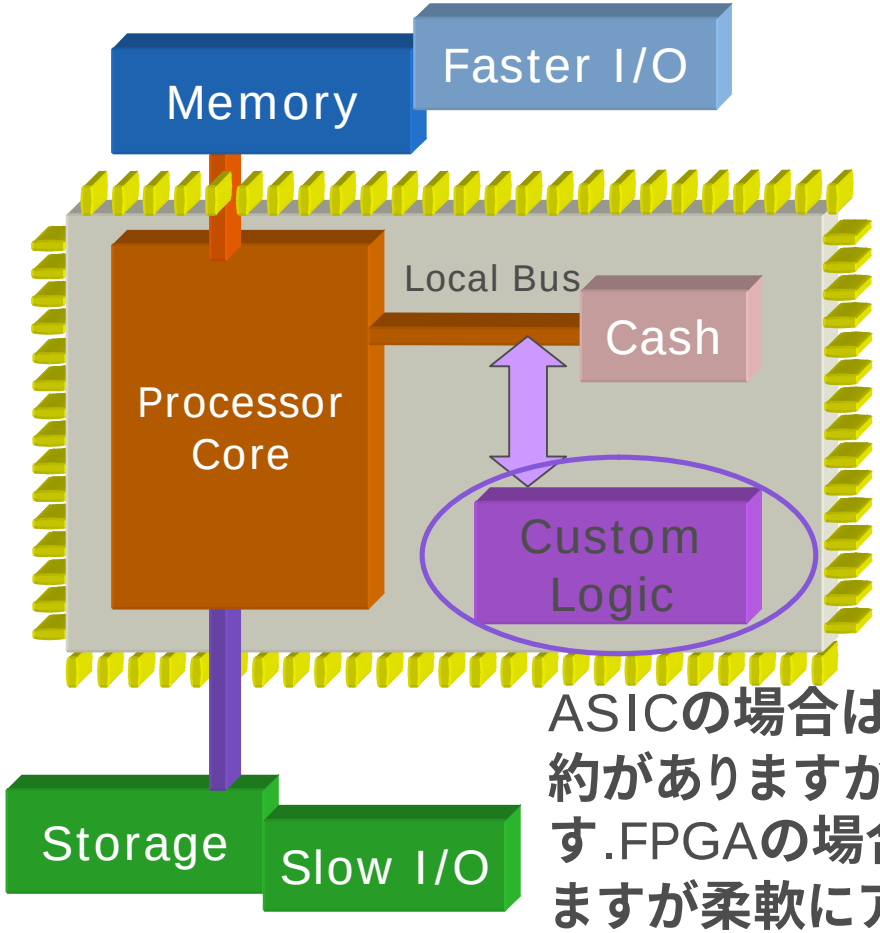
専用回路付加による一般的方法



プロセッサの仕様(ハードウェアと命令セット)は固定されていますから、性能を上げるためには外部にカスタムロジックによる専用ハードウェアを付加します。

このハードウェアが高速で動作可能であってもメモリとのRead/WriteでMPUコアとのデータのやりとりをしたのでは、膨大なオーバーヘッドが発生します。

コア内に専用回路を追加したカスタム化

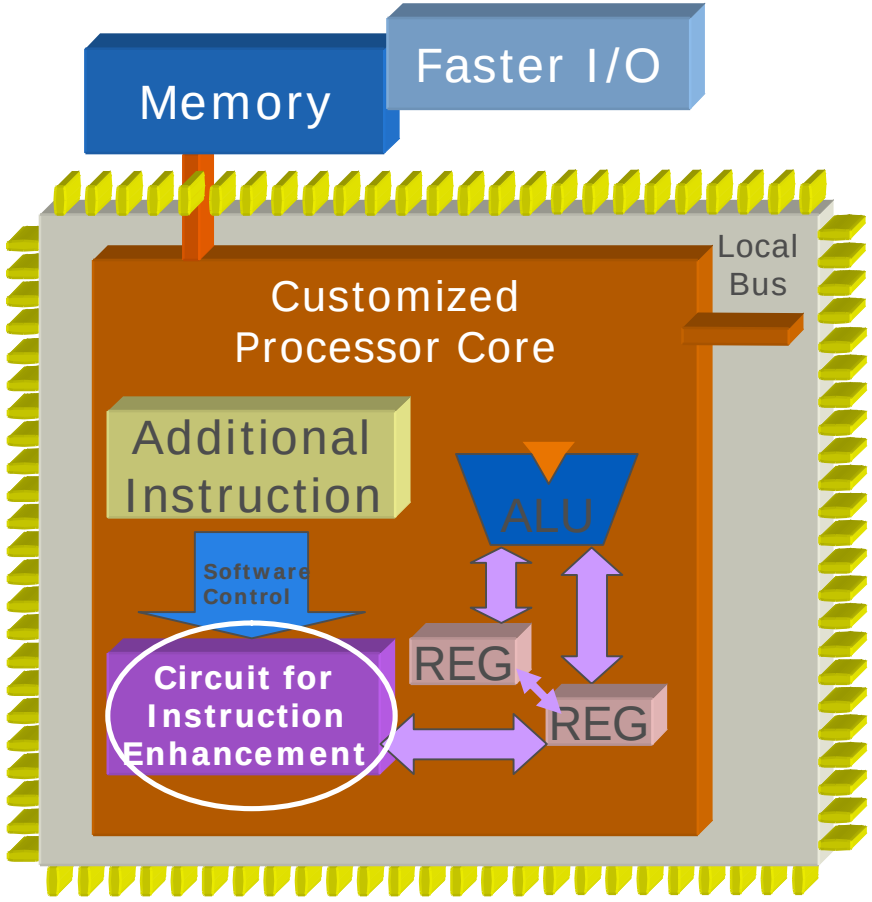


さらに専用ハードウェアをIPとしてダイの中に集積すれば高集積化,高速化,ローパワー化ができます。

このハードウェアの制御とプロセッサ間のデータのやりとりは,ダイ内部のインターフェースロジックで決定します。

ASICの場合はProcessor Coreは固定機能IPの制約がありますが性能と消費電力メリットがあります.FPGAの場合は消費電力とダイサイズが大きくなりますが柔軟にアーキテクチャ拡張が可能となります。

プロセッサの命令拡張による方法



専用ハードウェア処理をプロセッサに拡張命令として実装すれば,専用回路がMPU内部アーキテクチャに実装され,ソフトウェアで制御できます。

データのやりとりは,MPUのレジスタ間となりますのでオーバーヘッドがなくなり高速,低消費電力,ダイサイズも小型化します。

FPGAでプロトタイプ化ができASIC化で性能をさらに向上できます

乗算と除算

乗算と除算はプロセッサにとっては悩みのタネです

C言語で乗算や除算をコーディングしてコンパイルしてもこれら専用のハードウェアがプロセッサ内部にないと膨大な時間がかかります。

PSoCでは、プロセッサの外部に乗算器を実装しており、乗算はこの乗算器の入力レジスタにデータを書き込み、乗算器の出力した計算結果レジスタを読み取ることによって乗算を高速化しています。

PSoCに割り算をさせるととても遅くなります。

DSP(Digital Signal Processor) では、信号処理計算を高速で処理するためにプロセッサ内に専用ハードウェアを実装し高速の専用命令で処理できるようになっています。

A/D変換とD/A変換

外部の事象変化をプロセッサに伝えるためには、2つの要素が必要となる。

そのひとつは、事象を検知する装置で、最もポピュラーなものが、センサーである。

たとえば温度センサーは周囲温度に対しそれに対応した電圧を出力する。

たとえば温度が0℃のときの電圧出力が、1.0Vで、0.1℃温度が上がるごとに電圧が5mV上がるようというようにセンサーは設計されている。しかしこのようなアナログ電圧は、直接レジスタに書き込むことはできない。

そこで必要になるのが第2の要素で、これが、A/D変換器である。

A/D変換器は、アナログの電圧をデジタル値に変換する。

A/D変換は、ビット数とサンプリング時間でさまざまなものがあるが、たとえば8ビットのA/D変換器では、

00からFFまで値を持つことができるので、1ビット単位を0.1Vに設定した場合は、0Vから25.5Vまでを扱える。

単位を0.01Vに設定した場合は、0Vから2.55Vまでを読み取ることができる。

システムは8ビットのレジスタを用意しておき、A/D変換器は周期的にこのレジスタに8ビットのデータを書き込む。

このデータをプロセッサが読みにいけば、外の温度を知ることができるということになる。

この逆のメカニズムがD/A変換である。プロセッサはデジタル値でレジスタにNビットの値を書き込む。

(先ほどのA/D変換の例では8ビットだった)

D/A変換器は、このデジタル値をアナログの電圧に変換して出力する。この電圧で外部の装置をアナログ的に駆動できることになる。

A/D変換とD/A変換における変換器は、Converterという。よって変換器はよく略称でADC, DACと呼ばれる。

サンプリング時間

A/D変換には、一定の時間がかかる。この時間のことをサンプリング･タイムというが、これがあまりも遅いとプロセッサの処理に余力があっても処理能力の制約となる。

またA/D変換器に速度的に余裕があってもプロセッサの処理速度が遅かったり、他の処理を一緒にこなしていたりすると、この場合もまた、処理能力の制約となる。

**CDの規格

音楽CDの規格は16ビット44.1KHzサンプリングと定められている。この規格に定まった理由とそれから派生する問題点を考察せよ。

時計機能と通信機能

A/D, D/Aはアナログ信号とデジタル値を相互に変換するが、時計機能はデジタルのまま処理される。また、通信機能に関しても、

デジタルで処理されるが、信号線の本数が多くなると機器間の接続に支障をきたすので、信号線の本数を減らす方法がとられる。

例えば8ビットのレジスタのデータを送信するのに8本の線を使わずに1本の線に順次1ビットずつのレジスタのデータを

乗せて送れば、グランドとあわせて2本の線で送信ができる。このように8ビットの並列のデータを1ビットの並びに置き換えることをパラレルーシリアル変換という。

この逆は、シリアルーパラレル変換という。

調停

複数のハードウェアが同時に同じレジスタにデータを書き込むことはできない。信号の衝突(Contention)が発生するからである。

このような事態を回避するために、各ハードウェアがアクセスするタイミングを時間的に区分したり、同時にアクセスが発生した場合は、どれか一つのハードウェアだけにアクセスを許可する回路を付与する。この調停回路を、アービタという。

ポーリング

外部のハードウェアは、常にレジスタの値を書き換えている。プロセッサは、レジスタの値が変化しているかどうかを、プログラムによって定期的に監視する。

外部ハードウェアがA,B,Cの三種類あり、それぞれレジスタアドレスA,B,Cの値を書き換えているとき、プロセッサはプログラムによって、これらのレジスタの値を“定期的”に読みこみ、その値によって適した処理を行う。このように"定期的"にハードウェアのレジスタを読みにいくことをポーリングという。

ここで注意すべきことは、レジスタを読み込むタイミングは、正確に決定することはできないことだ。あるレジスタの値を読み込みその結果の分岐処理で分岐先ごとに処理時間が異なるためである。

また、レジスタの値に変化がなくとも“念のために必ず"見にいくことになるので、オーバーヘッドが発生することになる。

しかしシステムは簡素に構成できるので、外部に多くのハードウェアが接続されていなければ、実用的である。

ハードウェア割込み

プロセッサにはハードウェア割込み信号線があり、外部のハードウェアから駆動(アサート)されると、プロセッサは現在処理中のプログラムを一時中断して、“現在の汎用レジスタの内容をスタックに書き出し(PUSHする)”割込みが入った場合の処理プログラム･アドレス”(割り込みベクタ･アドレス)”にジャンプして、割込み専用のプログラムを実行する。割込み専用のプログラムの処理が終了した後、スタックから元の状態をレジスタに再読み込みし(POPし)割込み前の状態にもどし、割込み前のプログラムの処理を継続する。

このように割込みでは、ポーリングと異なり、常にレジスタの状態を見に行く必要がなく、メインの処理に集中できるので、オーバーヘッドが少なくなる。

反面、割込みのメカニズムは、ソフトウェアとハードウェア双方に実装するので設計は複雑になる。

プロセッサの割込みの信号線は1本が普通である。よって複数の外部ハードウェアがプロセッサに対して、割込み要求をした場合には、それらを調停し、どのハードウェアが割込み要求をしているかをプロセッサに知らせる

が必要である。

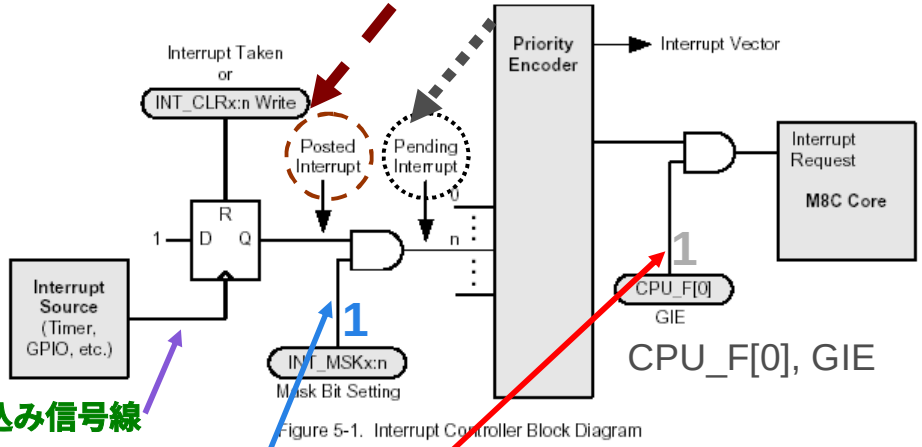
また、ソフトウェア的には、割込みしたハードウェア毎に処理が変わるので、ハードウェアごとにジャンプ先にのアドレスをあらかじめ決めておく必要がある。この割込みハードウェアごとにジャンプ先のアドレスを決めたものを、

割込みベクタ･テーブルという。

割り込みのハードウェア

PSoCの割り込みコントローラ回路

複数のユーザーモジュール・ハードウェアから割り込みがある場合はプライオリティー・エンコーダで優先度がつけられ,実行まちとなる割り込みは*ペンディング・インタラプト*となります



CPU_F[0], GIE

Figure 5-1. Interrupt Controller Block Diagram

PWM8_1を配置したDBB00からの割り込み信号線

ハードウェアとの関係

```
void main()
{
    M8C_EnableGInt;
    PWM8_1_EnableInt();
    PWM8_1_Start();

    while(1){};
}
```

INT_MSKx

M8Cの割り込みが1=“有効”
CPU_Fレジスタ[0]により
CPU割り込みはマスク可能=0

PWM8_1の割り込が1=“有効”
ポステッド割り込みはマスク可能=0

割り込みが処理されると*ポステッド・インタラプト*がリセットされます

割り込みの処理

割り込みが発生すると現在実行中の命令が終了し割り込みサービス･ルーチン(ISR)13サイクルが発生し次の処理を実行.

PCH,PCL,CPU_Fの順にスタックにPUSH

CPU_Fをクリア,GIEは0になる

PCHが0になりPCLに割り込みベクタ値をセット

割り込みベクタのアドレスにロングジャンプ

ISR(割り込みサービス･ルーチン)の実行

RETIでISRが終了

CPU_F,PCL,PCHの順にスタックからPOPして割り込み前に復帰

割り込みレイテンシは、以下の時間の和になります

現在の命令の終了までの時間

+M8Cがプログラムカウンタを割り込みアドレスに変更する時間

+割り込みテーブルのLJMP命令の実行時間（7サイクル）

（例:割り込みアクティブから5サイクルのJMP命令でISR開始とすると5Cycle+ISR13Cycle+LJMP7Cycleで25Cycleですから約1.042uSec(24MHz,25Cycle)かかります

複数割り込みと優先順位

多重割り込み

現在処理中の割り込みが継続され2つめ以降の割り込みは無視される

現在処理の割り込みが終了次第,次のプライオリティーの割り込み処理が実行される

ハードウェア割り込みのプライオリティー順

リセット　(00h)

電源モニタ　(04h)

4つのアナログカラム(08h,0Ch,10h,14h左の位置から順に右へ)

VC3　(18h)

GPIO　(1Ch)

16のデジタルブロック(左上DBB00-20hから右へ向かい右端DCB03-2Chまでいったら次の段の左端DBB10-30hへ,同様に右に進む)

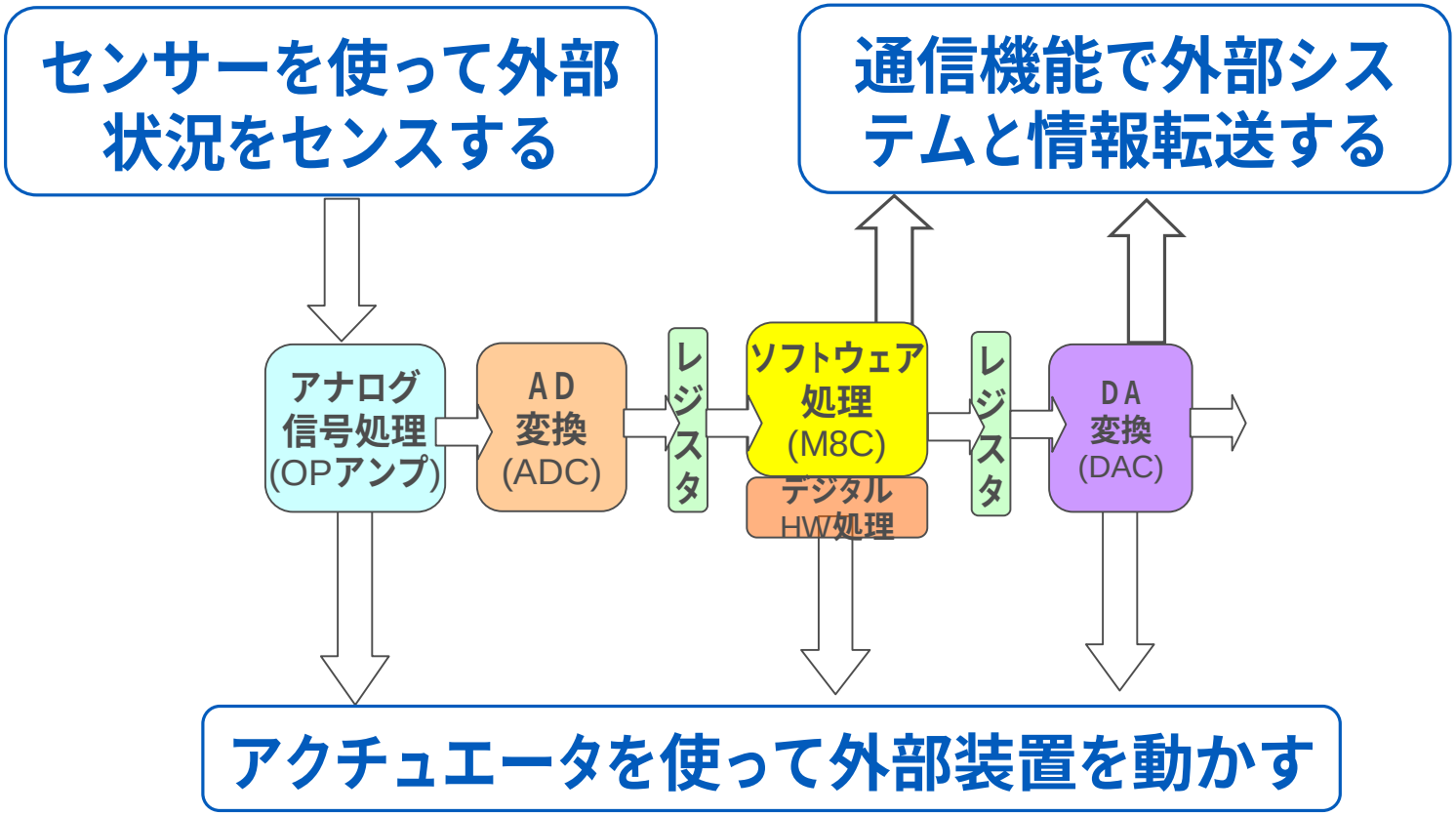
I2C　(60h)

SleepTimer (64h)

CPUへの割り込み数は
*固定6*つを含む26

PSoCを組み込みシステムとして見る

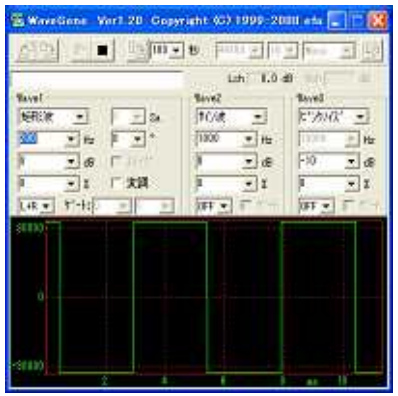
センサ、アクチュエータ、通信の統合



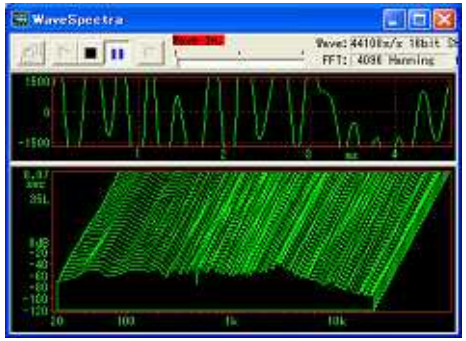
周辺ソフトウェアの事前準備

WG(WaveGen)のインストール
efu氏開発の信号発生器ソフトウェア
WS(WaveSpectra)のインストール
efu氏開発の波形観測,スプクトラム解析ソフトウェア
PSoC Designer5 ソフトウェアのインストール
PSoC Programmerソフトウェアのインストール

WG



WS



上記4つのソフトウェアがインストールされているか確認
それぞれが起動するかの確認

WGを起動しPCから音が出るかの確認
WGの音量調整の確認 (コントロールパネルのスピーカー音量調整)
スイープの試行(200Hz-4000Hz 30秒)

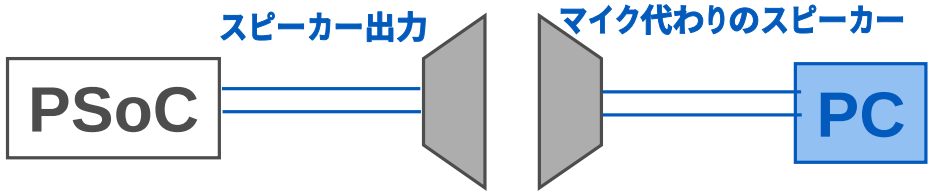
WSを起動し波形とスペクトラム表示が行われるかの確認

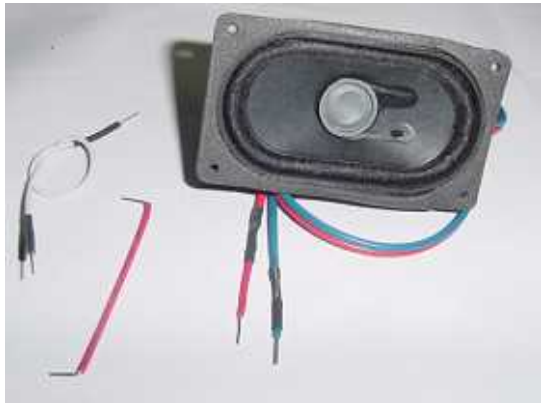
WSの入力レベル調整の確認(コントロールパネルのマイク/ライン入力調整)

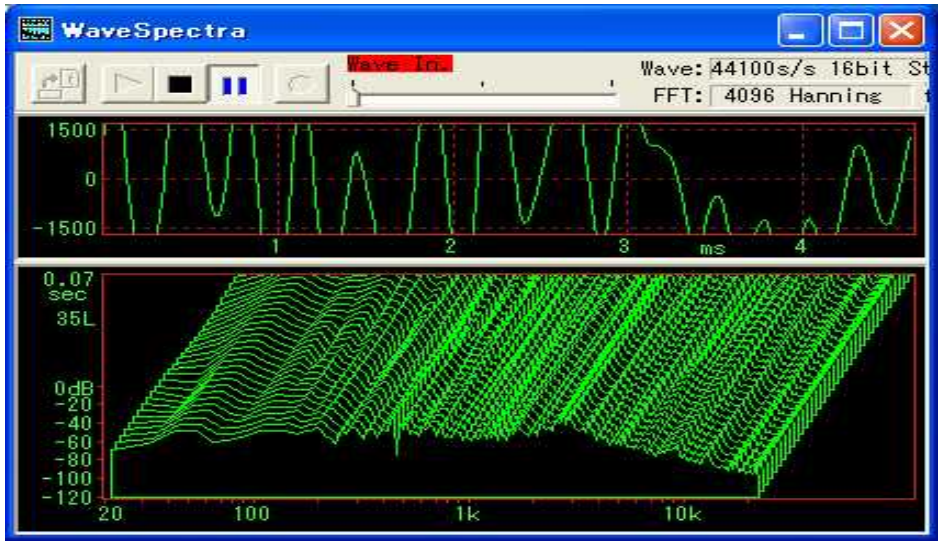
WGのスイープ信号のWSでの読み取り表示(音響カプリング)

音響カプリング

PCのライン入力にPSoCの出力を直接接続すると直流や音響帯域以外のノイズ信号がPCのサウンド回路入力に加わるので,2個のスピーカを使って音響的にカプリングしてスピーカの周波数特性をフィルターとして使用します (レベルはWindowsのコントロールパネル>サウンドとオーディオデバイスで調整できます)





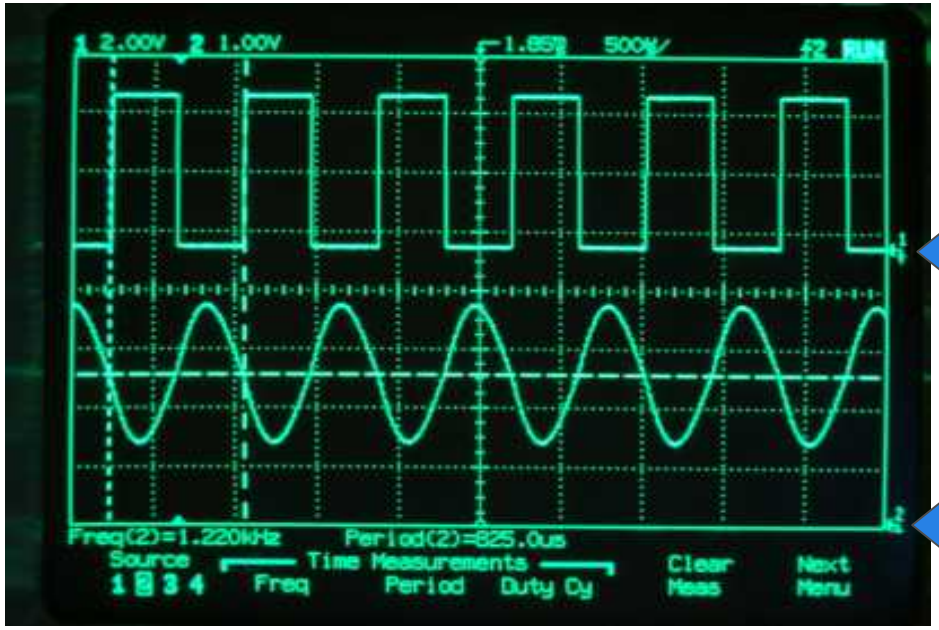


直流と交流信号について

一般のスピーカーやPCのライン/マイク入力または外部への出力はグランドレベルに対して正負に振幅する交流信号です.

PSoCのオペアンプは単電源型ですよって負の入力信号を扱うことができません.また出力は直流バイアスがかかります.

相互にインターフェースをするためには,キャパシタでDC成分を切ってやったり,直流電圧をバイアスしてやります.



PSoCの出力信号の実測波形の例

デジタル方形波出力

ここが0V

5VP-P アナログ機器に

直結すると故障を起こす

正弦波出力に2.5VのDCが

ここが0V

加わっている

実習用器材の確認

最低限必要なもの

PSoC評価基板

CY3210 EVAL1等(Miniprog書き込み器,27443(29466)デバイス，USB巻き取り式延長ケーブル付)

実習のために用意の必要なもの

ジャンパ線 1台につき最低10本程度

配線つき3Pミニプラグジャック(PCサウンドカードに接続)

振動モーター(携帯電話用),スピーカー,キャパシタ,抵抗

自由課題用

cds(光強度を抵抗変化に変換),温度センサ, 超音波センサ, 2または3軸ジャイロセンサ(ロボットやヘリコプタ), カラーセンサ, 圧力センサ, 曲げセンサ, PIRセンサ

マイクロホン, スピーカー用アンプ, Hブリッジ, パワーMOSFET, ブラシモータ,

USBシリアル変換ケーブル(PC-PSoC間通信)

はんだごて, 工具, ブレッドボード, 課題ごとのいろいろな電子部品

あれば便利なもの

ファンクションジェネレータ, オシロスコープ, スペクトラムアナライザ,好奇心

ラボで使用するファイルのルール

Labで使用する資料,ファイルのディレクトリの作成ルールは、以下のとおりとします. **絶対にまちがわないように注意**してください

各自の演習ファイルの置き先:C:¥psoc_lab¥

各ラボのファイル群はC:¥psoc_lab¥**のサブディレクトリに作成してください**

例 : **ラボ名** p3_1200hz **の場合は各自で作成するものは**

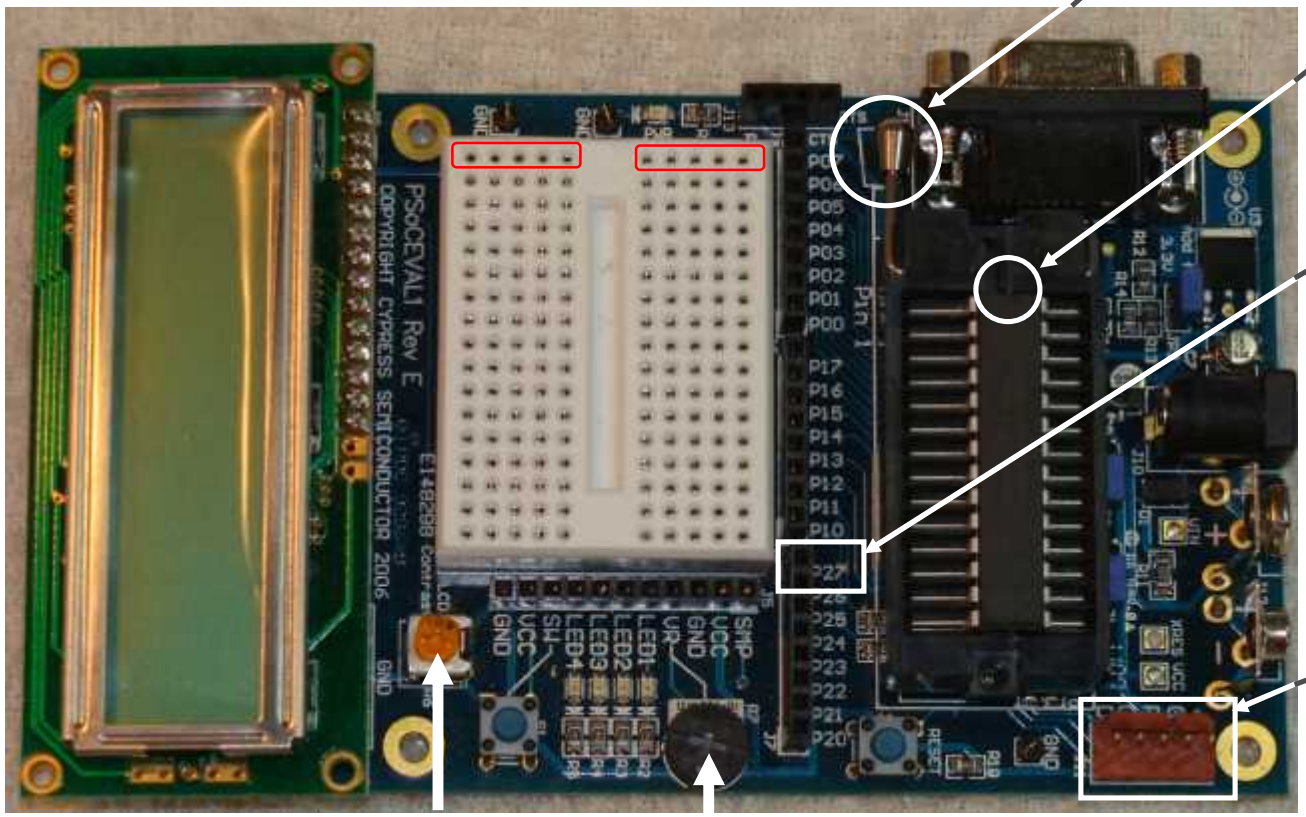
ファイル･ディレクトリ C:¥psoc_lab¥p3_1200hz **となります**

解答例となる"完成プロジェクト"は、デスクトップのpsoc_lab_master(**または**psoc_lab_master2010)**というディレクトリにあります。ない場合にはサポート**URL**からダウンロードしてデスクトップに置いてください**.

評価基板CY3210EVAL1の説明

白い穴の開いた部分がブレッドボード部で,すべて**横方向に内部接続**されています.(中央の仕切りを越えた部分で左右は絶縁).

レバーは1番ピン側にくるようになります



丸いへこみのある方を
上にして左上がデバイスの
1番ピンになります

黒いボックスソケットにデバイスの
信号が引き出してあります.
P27の場合Port2の7ビット目の
信号という意味です.
デバイスのピンの番号では
ありませんから注意してください

ここに書き込み器
MiniProgを差し込みます

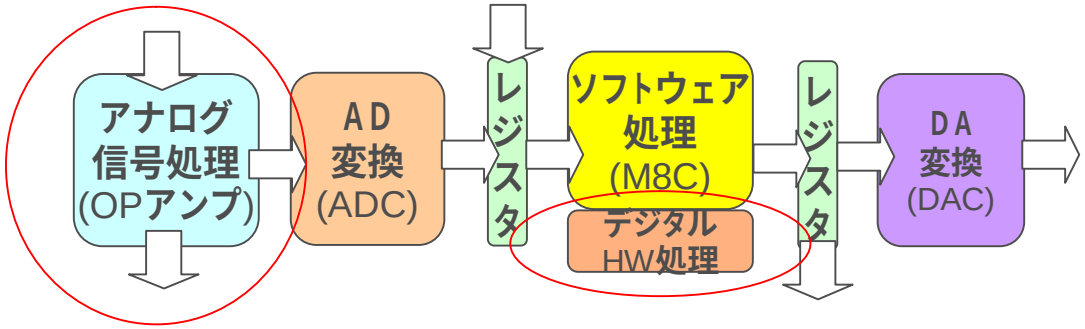
液晶の濃度を調整するVRです

可変抵抗器VRです

ラボ

p3_1200hz

MiniProg書き込み器のテストと書き込み方法の演習



ラボ p3_1200hz

1.PSoC Programmer をPCのUSB ポートに認識させます

2.PSoC Programmerを起動します

3.p3_1200hz.hex ファイルを読み込んで27443に書き込みます

4.正常に書き込めたかを確認します。

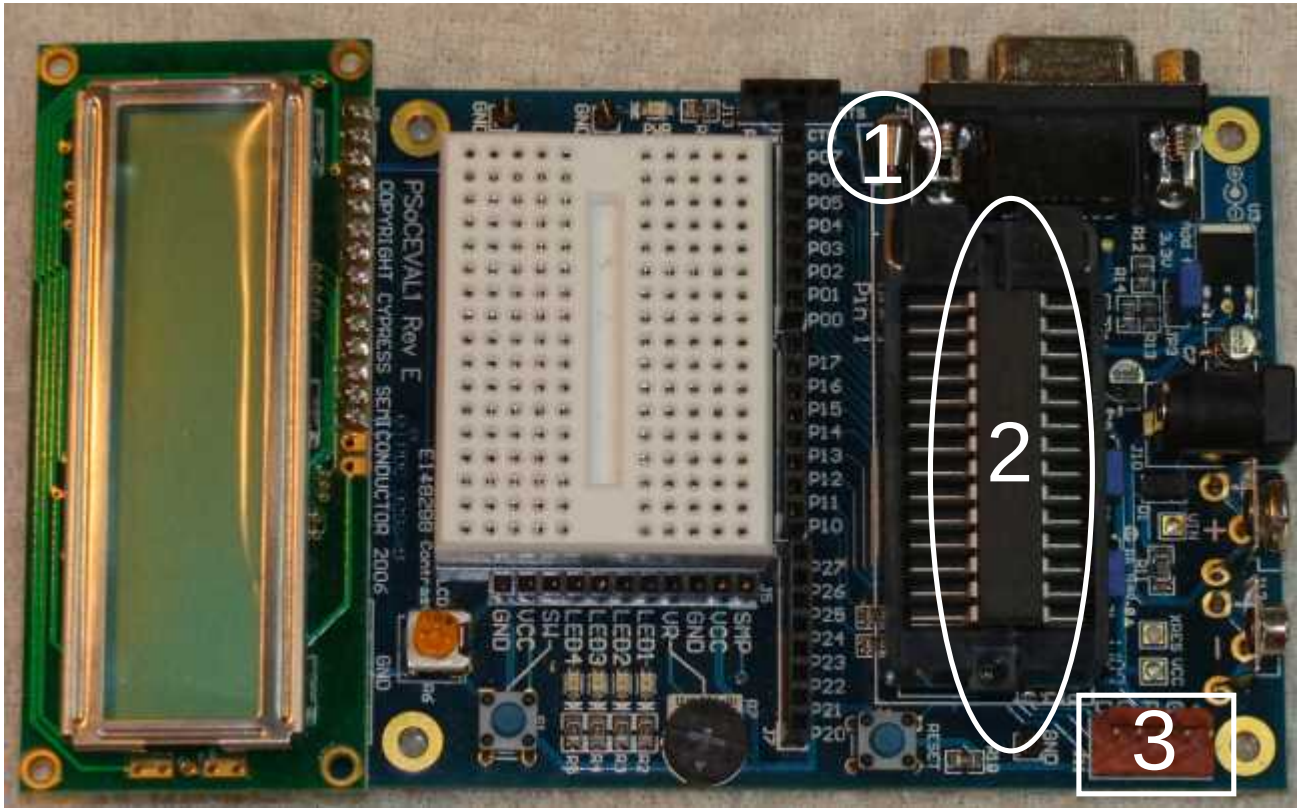
解説 : p3_1200hz.hex ファイルは,1200Hzの正弦波信号を[PORT0:3] 27443デバイスの3番ピンに出力する回路を設計したファイルです.PSoCに電源が入ると信号が出ます.(スピーカーをつなぐととても小さい音-1200Hzのサイン波が聴こえます)4番ピンにつなぐと音は大きくなります-1200Hzの矩形波が聴こえます.

MiniProg書き込み器のテストと書き込み方法の演習です.

基板の準備

レバー①を上に立てソケットを開きデバイス27443を実装②し,レバーを倒してデバイスを中央でしっかり固定してください.

続いてMIniProgを③ピン位置がずれないようにさしこんでください.

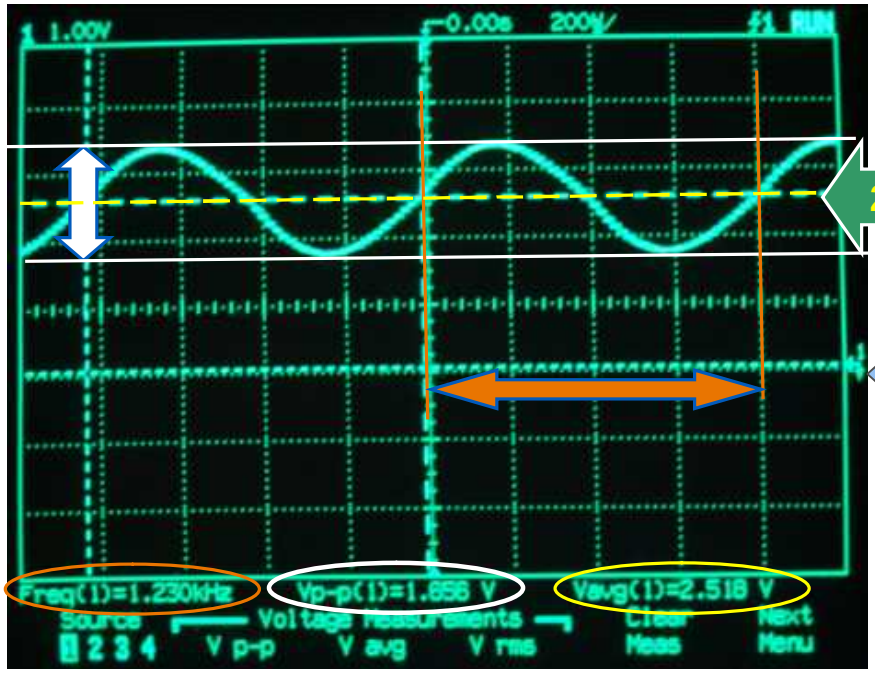


p3_1200hz正弦波発振器

27443に1,200Hz 発振器を書き込む

ファイルは.¥p3_1200hz¥p3_1200hz.hex

3番ピンにキャパシタを直列に入れてWSで波形とスペクトラムを観測する(省略可)-音響カプリングでもかまいません



2.5V

ここが0V

オシロスコープで観察するとこんな波形が出ています
アナロググランドレベルが2.5Vに浮いています

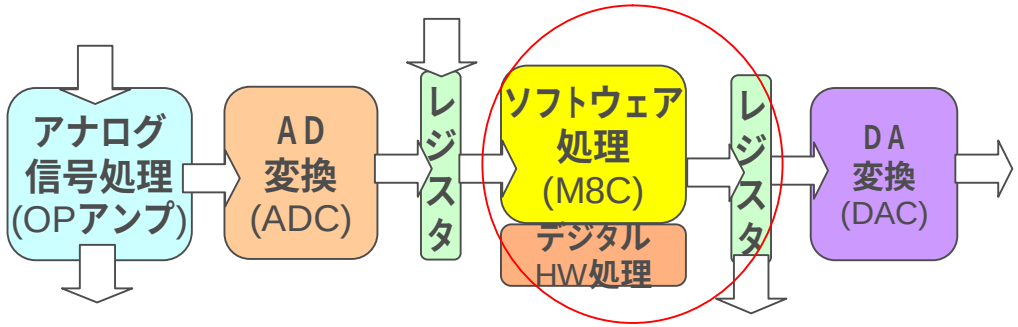
レベルがとても低いのでWSではスペクトラムレベルがとても小さくなります

4番ピンにつなぐと音は大きくなります-1200Hzの矩形波が出力されています.

ラボ

hello world

LCDに文字を表示させます. 処理フローの演習



ラボ hello_world

1.PSoC Designer を起動してProjectを作成します.

2.User Moduleから"LCD"を選択して配置します.このときモジュール名は**自動的にインスタンス番号が追加された名前**で登録されます.名前を変更することもできます.main.cにソースを記述するときはすべて登録された名前を使用してください.API関数も登録されたモジュール名が最初についたものになります.

3.LCDのパラメータを設定してPort2に割り当てます.

4.main.cにソースを記載します.

LCD_Start(); 文は自動登録デフォルトのままではLCD_1_Start() と書かなければいけません.

API関数も同様にLCD_1_PrCString(); となります. 自分の名前など好きな文字列を表示させてみてください.

5.Generate Configurationを実行します

6.PSoC Programmerで書き込みして動作を確認します.

解説 :PSoC Designer では複数のユーザーモジュールを識別するためにユニークなユーザーモジュール名を自動的に生成し識別するようになっています。

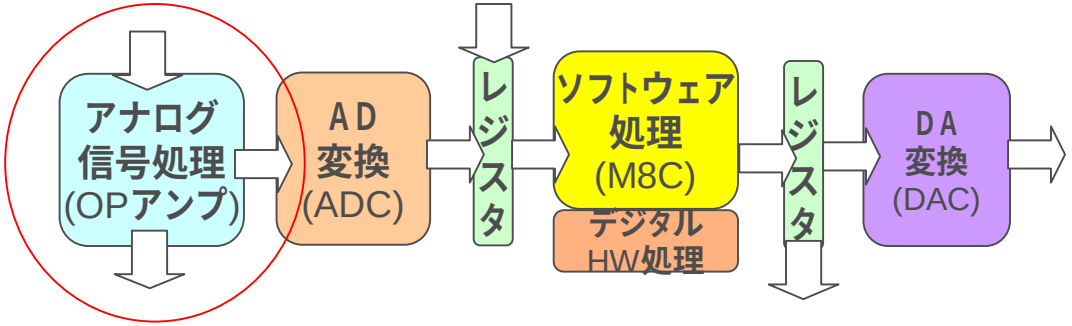
ラボ

motor

アナログ信号の増幅

非反転増幅器PGAを使用してアナログ増幅回路を作ります.

入力,出力どちらも直流です.



ラボ motor

1.PSoC Designer でPGA ユーザーモジュールを使用して単電源非反転型オペアンプによる増幅回路を設計します

2.PSoC Programmerでデバイスに書き込みします

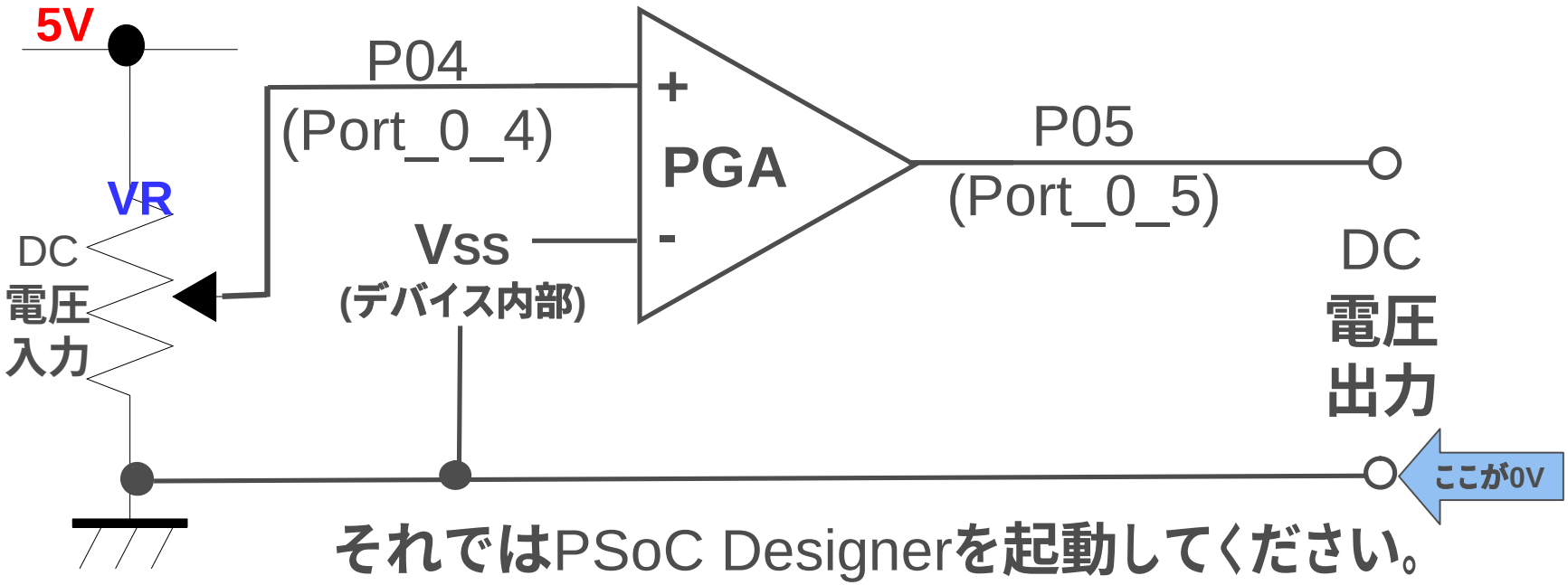
3.デバイスの出力で振動モーターを動かしてみます

解説 : この回路では外部から加えた直流の入力信号を増幅して外部の負荷を駆動します.単電源非反転型オペアンプのリファレンス入力がVSSになっていることを考慮してください

PGA : Programmable Gain Amp

プロジェクト motorの回路

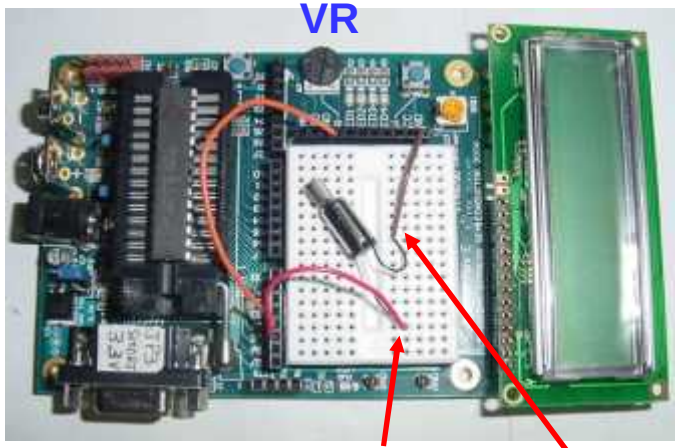
- DC出力に携帯電話用のバイブレーションモーターを接続します.
- 入力に可変のアナログDC電圧を印加し出力のアナログ電圧を制御
- PGAのゲインはプロパティウインドウで設定します.
- PGAの定格、特性はデータシートウインドウで見ることができます.
- 各ピンの設定はピンアウトウインドウで行います.
- PGAのリファレンスはVssであることに注意してください.入力には**正の信号(電圧)**を加えます.出力には**正の信号(電圧)**が出ますがこの信号は*電源電圧を超えることはありません*



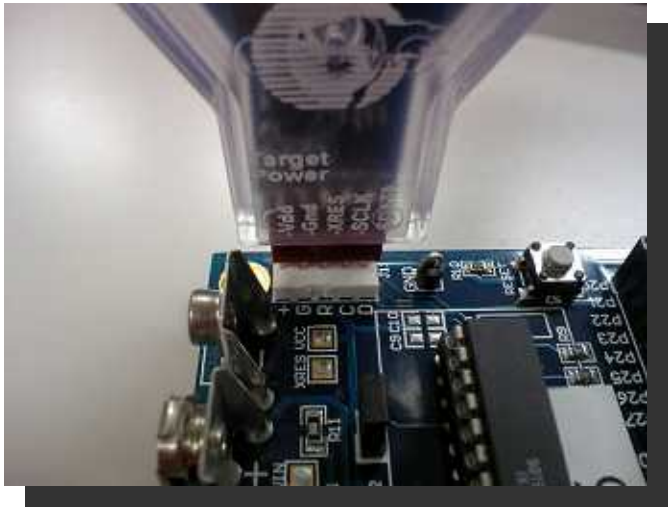
それではPSoC Designerを起動してください。

MiniProgの接続、回路配線

- P04 と VR を ジャンプワイヤーで接続,VR(可変抵抗)は*左に回しきっておきます。*この状態でPGAの入力(P04)には,VRからの可変電圧0Vが印加されます。VRを右に回していくと少しずつ電圧が上がっていきます。VRは5Vを分圧していますから右に回しきると5Vが印加されます。
- 振動モーターをP05(PGA出力)とGNDの間に接続します。2V,40mA程度でドライブしてください。長時間動作させると焼き切れる場合があります。(詳細はPGAのデータシート参照)
- MiniProg を Eval1 に接続
  Vdd と + が一致するように注意

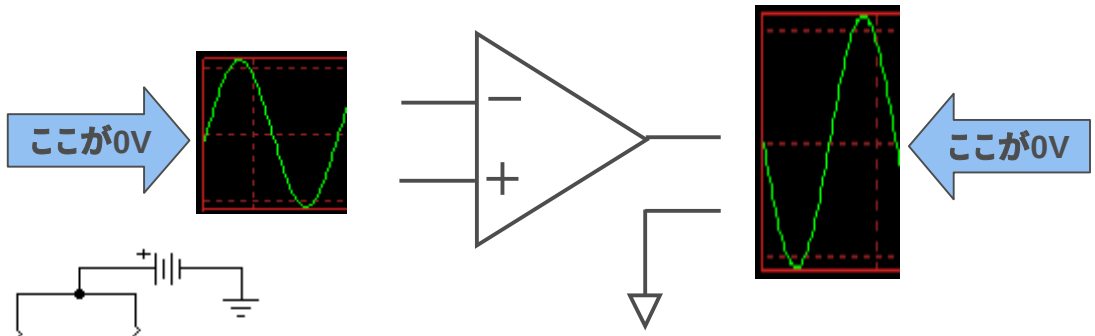


モーターのリード線を同じ穴に一緒にジャンパーで押し込むようにします

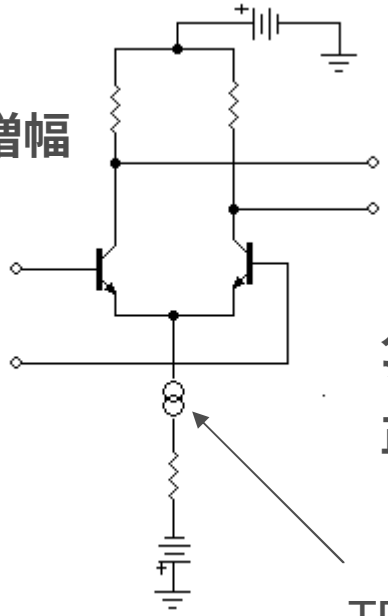


一般の正負2電源OPアンプ

入力の差を増幅して出力.出力は対グランドレベルのシングルエンド



差動増幅回路



差動出力は,逆位相出力なのであとにコンプリメンタリ出力段を組みやすい
各段は直結構成(位相特性,FB安定のため)

入力インピーダンスが高い,入力端子間に電流が流れない,イマジナリー・ショート

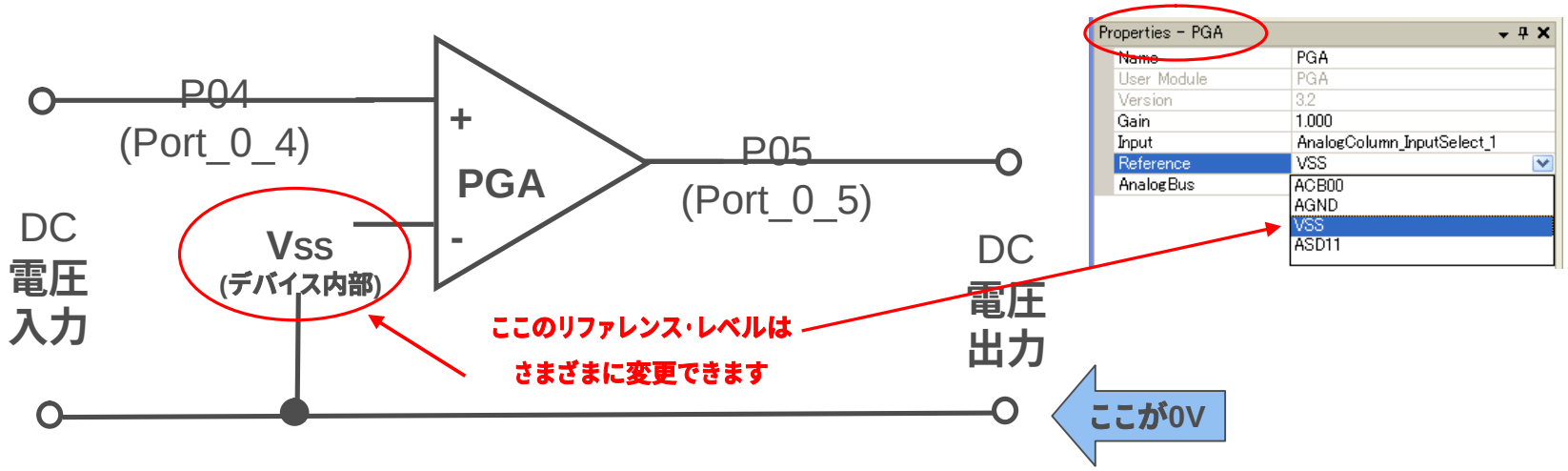
正負2電源でシングルエンド出力

TRは相互に逆位相動作,電流の和が一定

PSoCの単電源オペアンプ

PSoCのオペアンプは単電源型ですからプラスの信号入力だけを増幅します. PGAのリファレンスはVss(0V)に設定すると入力に加えた0から5Vまでの電圧に対して設定した増幅率(ゲイン)の出力を行いますがこの出力は *電源電圧を超えることはありません*

マイナス側にも振幅する一般の交流信号をそのまま入力して使用するには, 2電源型 (プラス、マイナスの電源を供給します) オペアンプを使います



PSoC のMotor プロジェクトの例

PSoCへの交流信号入力

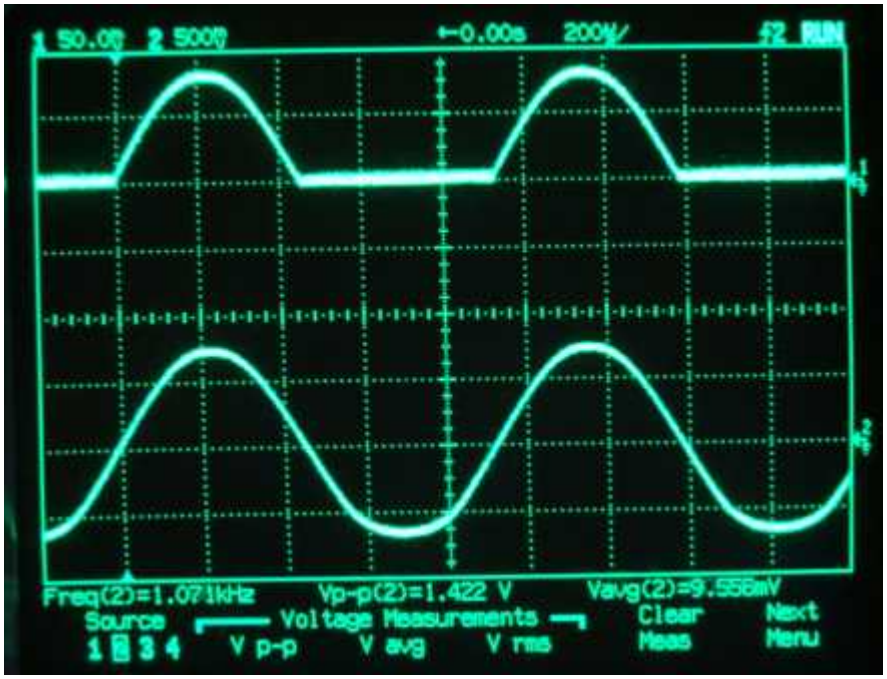
PSoCのオペアンプは単電源型ですからプラスの信号入力のみを増幅することができます. 一般の正負に振幅する交流信号を入力してもマイナス側の振幅は増幅されません. 外部から交流信号を与える場合は、PSoCのアナロググランドレベル(リファレンス・レベル 例：+2.5V)まで電圧をバイアスして上げてやる必要があります.

PGA

出力

交流

正弦波入力



ここが0V

ここが0V

単電源非反転オペアンプの入力に負の電圧を印加すると壊れることがありますので,注意してください

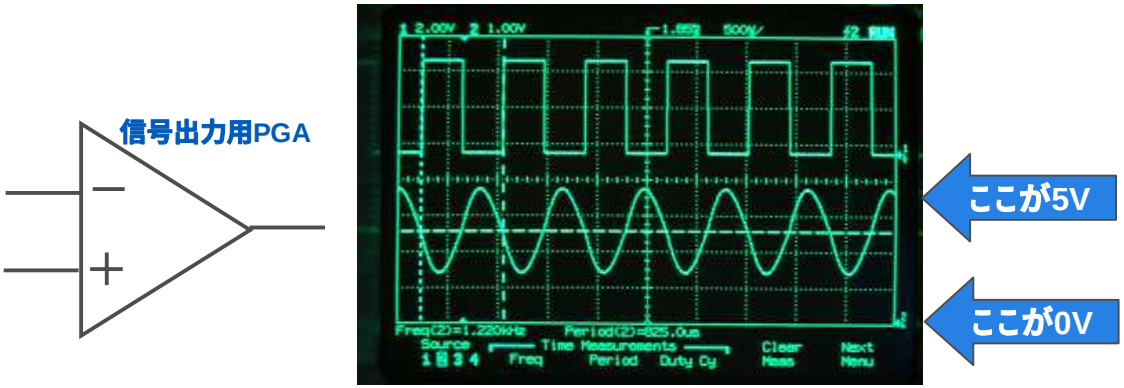
PSoCからマイナス振幅の出力を出す

信号出力回路が+2.5VのDCバイアスがかかっている場合は別のPGAで+2.5Vのレベル生成をしてやります.

Global Resources - motor

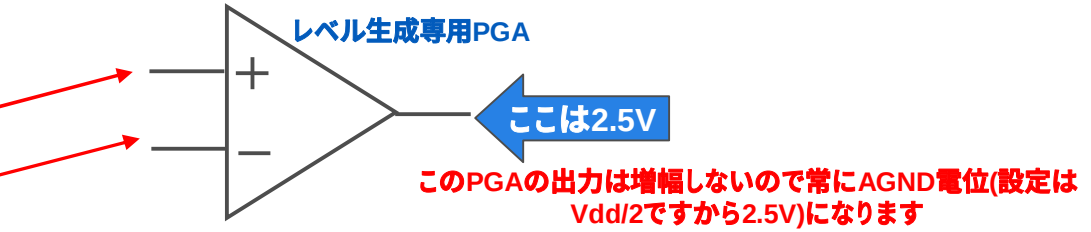
CPU_Clock	24_MHz (SysClk/1)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	16
VC2= VC1/N	16
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-(Vdd/2)
AGndBypass	(Vdd/2)+/-BandGap
Op-Amp Bias	(Vdd/2)+/-(Vdd/2)
A_Buff_Power	BandGap+/-BandGap
SwitchModePump	(1.6 BandGap)+/-(1.6 BandGap)
Trip Voltage [LVD (SM	(2 BandGap)+/-BandGap
LVDThrottleBack	(2 BandGap)+/-P2[6]
Supply Voltage	P2[4]+/-BandGap
Watchdog Enable	P2[4]+/-P2[6]

信号出力回路(0から5V振幅)とレベル生成回路(2.5V固定)間の電位を信号出力としてとり出せば±2.5V振幅の交流信号になります (確信犯的?)



Properties - PGA_3

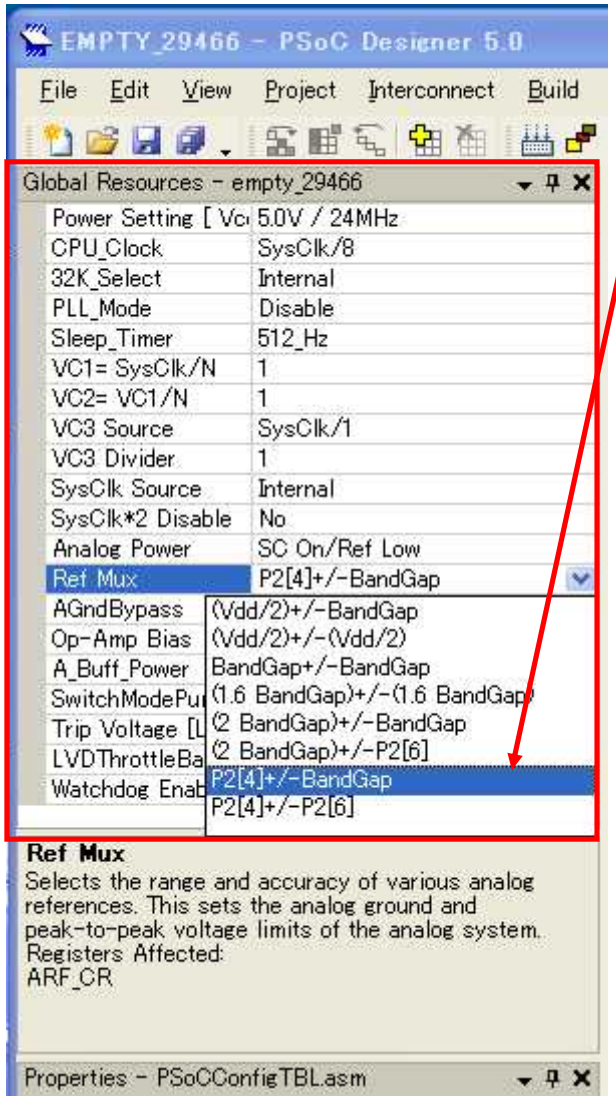
Name	PGA_3
User Module	PGA
Version	3.2
Gain	1.000
Input	AGND
Reference	AGND
AnalogBus	ACB02
	AGND
	VSS
	ASD13



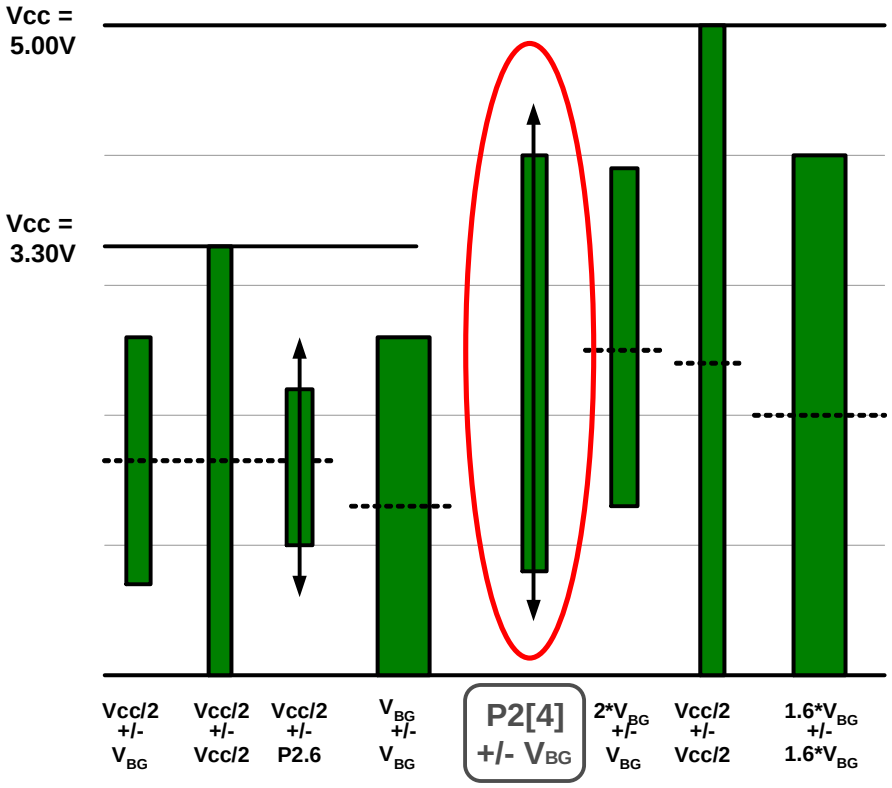
このPGAの出力は増幅しないので常にAGND電位(設定はVdd/2ですから2.5V)になります

このPGAは入力もリファレンス・レベルもAGND(アナログ・グランド)に設定しています

アナログ･リファレンスの設定



アナログのリファレンス電圧を設定する
ここではP2[4]+/- BandGap **としてみる**
BandGap**電圧は**1.3V (1.2V_{BG}+0.1V_{Gain})
レジスタは*ARF_CR: 0,63h* REF[2:0]**の**3**ビット**



アナログ･リファレンスの構造

PSoC オペアンプは単一電源型のためAGNDをV_{DD}の中間付近に設定します. AGND電位は各ブロック毎にバッファされるので各ブロック間で若干のオフセットが発生します. RefHi/LoはDACのコンパレータ･スパンを設定します.
これは 0,63h ARF_CRレジスタのREF[2:0]値で設定します

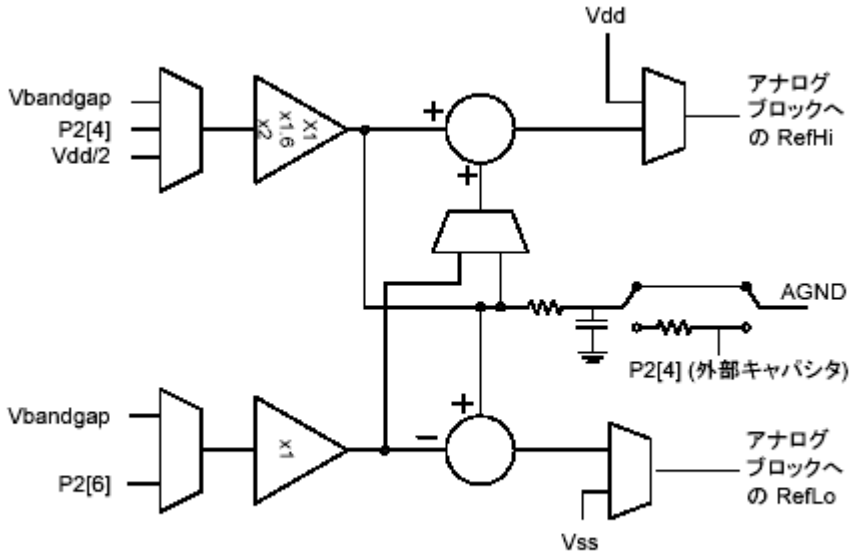


図 21-1. アナログリファレンスコントロール模型

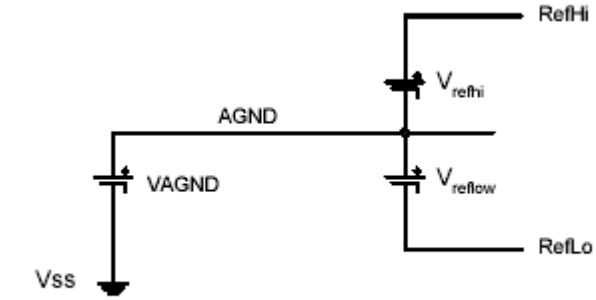


図 21-2. リファレンス構造

表 21-1. アナログリファレンスレジスタ

アドレス	名前	ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0	アクセス
0,63h	ARF_CR		HBE	REF[2:0]			PWR[2:0]			RW : 00

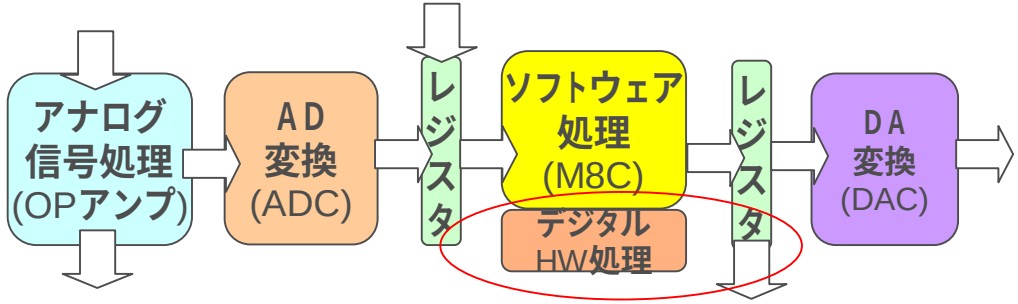
デジタル出力を使った出力の駆動,クロックリソースの演習

PWMを使用した任意の周波数（Period）と幅(Pulse Width)

を持つパルスの作り方がポイント

ラボ

lab1_pwm



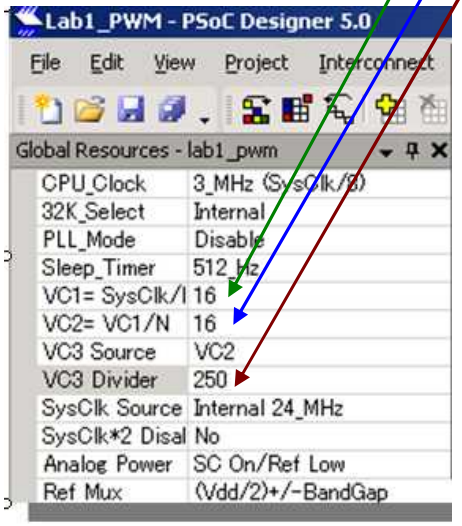
PSoCのクロック･システム

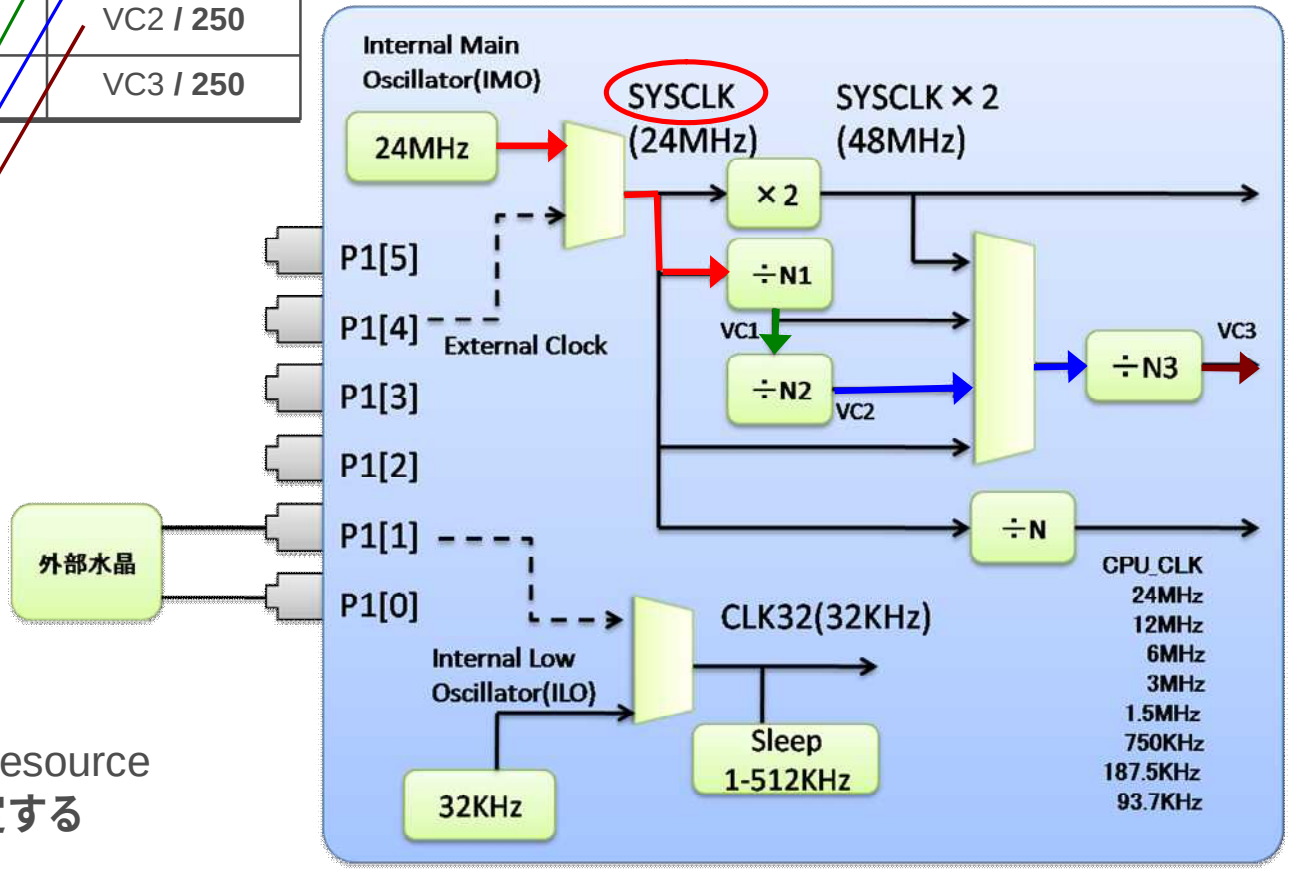
クロック	周波数	式
SysClk	24MHz	
VC1	1.5MHz	SysClk / **16**
VC2	93.75KHz	VC1 / **16**
VC3	375Hz	VC2 / **250**
PWM8出力	1.5Hz	VC3 / **250**

24MHz のSysClkを分周していき
375Hzの周波数のVC3クロックを作る

SysClk ---> VC1 ---> VC2 ---> VC3

1/16　　1/16　　1/250





分周比は、Global Resource
ウインドウで設定する

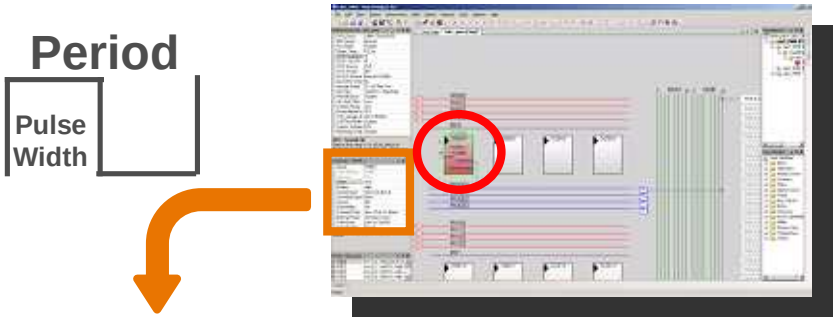
PWM8 波形決定パラメータ

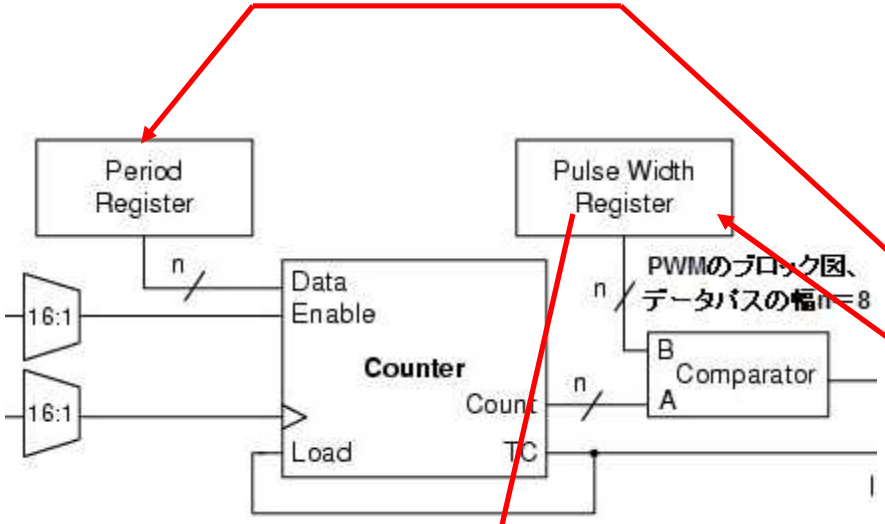
Period Register 値はPWM周波数を設定

VC3周波数が375Hz, Periodが249であればPeriod時間は、1/375 * (1+249) sec = 0.67sec ….1.5 Hz

Pulse Width Register 値はパルス幅を設定

VC3周波数が375Hz, Pulse Width が124であれば、パルス幅は、1/375 * (1+124) sec = 0.33 …. 1.5Hzに対して50%duty





Properties - PWM8_1

Name	PWM8_1
User Module	PWM8
Version	2.5
Clock	VC3
Enable	High
CompareOut	Row_0_Output_0
TerminalCount	None
Period	249
PulseWidth	124
CompareType	Less Than Or Equal
InterruptType	Terminal Count
ClockSync	Sync to SysClk
InvertEnable	Normal

Clockの選択とPeriodで周波数、
PulseWidthでDutyを設定

通販1100円のサーボモータを動かしてみる

秋月にて,GWS社のMicro STD サーボモーターを購入したが仕様不明

赤ラインが+5V, 黒がGND, 白が制御信号

この手のサーボの制御は,15ms から20msの周波数でコントロール、パルスの幅は1ms内外と見当をつけて試してみます。

PSoCのPWM16モジュールを使用して動作させてみました。結果として,制御角度は約180度, 0度に設定するパルス幅が0.7msec, 180度に設定するパルス幅が2.4msecでした。(数字はPulse Width設定値)

約2.4msec

約0.7msec

Global Resourceの設定と基本周期

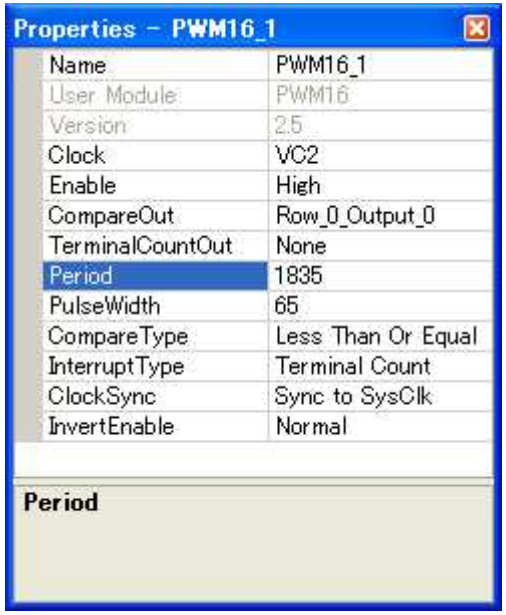
Global Resources - motor	
CPU_Clock	24_MHz (SysClk/1)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	16
VC2= VC1/N	16
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-BandGap
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (SMP)]	4.81V (5.00V)
LVDThrottleBack	Disable
Supply Voltage	5.0V
Watchdog Enable	Disable

CPU Clock 24Mhz

VC1 = 1.5MHz(24MHz x 1/16)

VC2 = 93.75KHz (VC1 x 1/16)

VC2のクロックでPWM16を駆動



PWM16でPeriod **レジスタの値を設定**

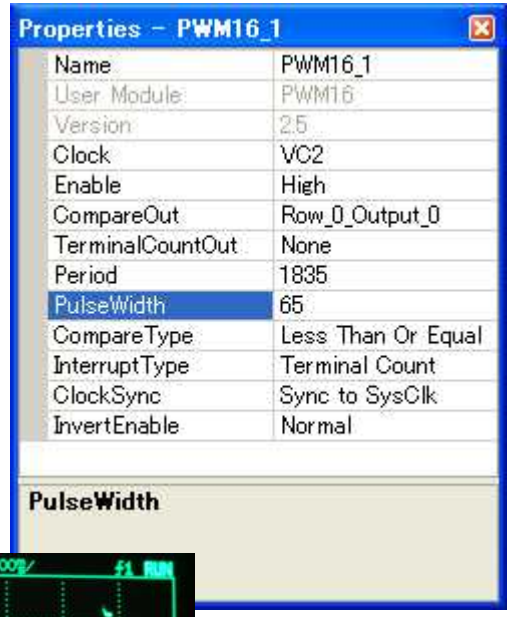
93.75KHz / 1835 = **約**51Hz (19.6msec)

これを基本周期にしてみました。

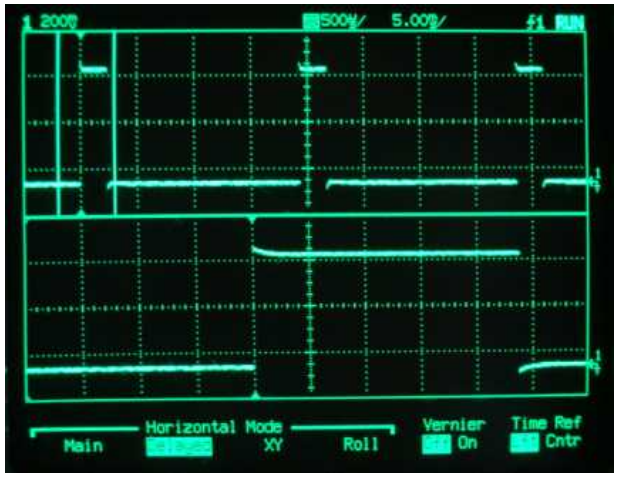
ここではPWMの設定値は簡略化していますから正確な計算法は
かならずユーザーモジュールデータシートで確認してください

パルスの幅の設定

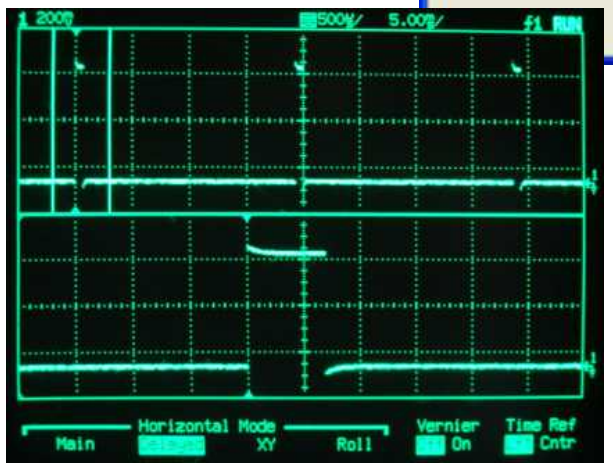
Pulse Width 65を設定
93.75KHZ = 約 0.011msecの解像度
0.011 x 65 で約0.7msecのパルスを生成
最大時は約2.4msecのためPulse Width値は225



約2.4msec (PW=225)



約0.7msec (PW=65)



パルス幅の変更で自由に回転角度を設定

プログラムソースからの制御

PWMのプロパティ・ウィンドウで初期設定した値は, main.c から直接レジスタ値を設定することで自由に変えることができます。

プログラム上から設定するには右のように
PWM16_WritePeriod()
（Period**パラメータ設定**）
PWM16_WritePulseWidth()
（Pulse**幅パラメータ設定**）

API**関数を使用します。**
PWM**ユーザーモジュールデータシートを参照**

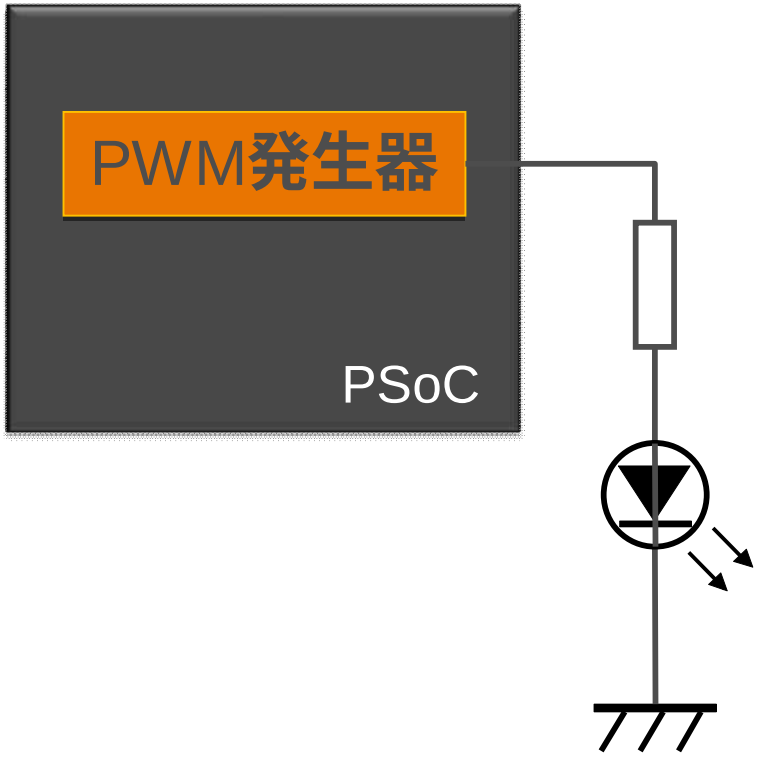
```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      This sample shows how to create a 33% duty cycle output
;      pulse. The clock selected should be 1000 times the required
; period. The comparator operation is specified to be "Less than or Equal".
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
/* include the Counter16 API header file    */
#include "PWM16.h"

/* function prototype */
void GenerateOneThirdDutyCycle(void);

/* Divide by eight function */
void GenerateOneThirdDutyCycle(void)
{
    /* set period to eight clocks */
    PWM16_WritePeriod(999);
    /* set pulse width to generate a 33% duty cycle */
    PWM16_WritePulseWidth(332);
    /* ensure interrupt is disabled */
    PWM16_DisableInt();
    /* start the PWM16! */
    PWM16_Start();  }
```

lab1_pwm

- PWM8ユーザーモジュールを用いて
  LEDを1.5Hzで点滅



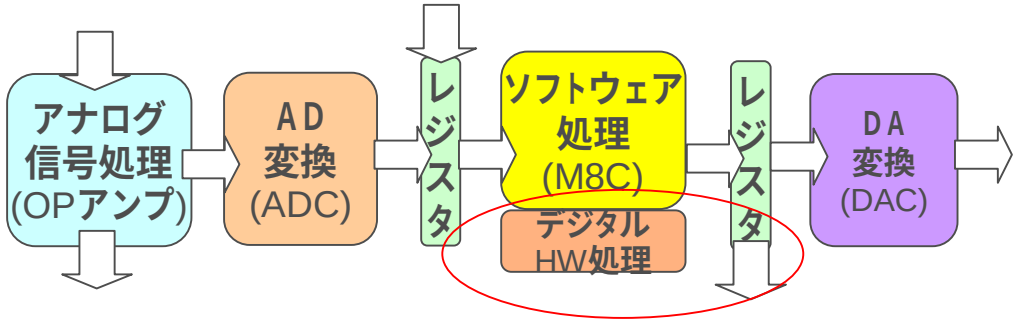
この設計では、
出力がデジタル
ドライブになって
います。

2つのPWMモジュールを使用して積分値を連続的に変化させて
アナログ的な出力変化を作ります

ラボ

lab2_pwm_lcd

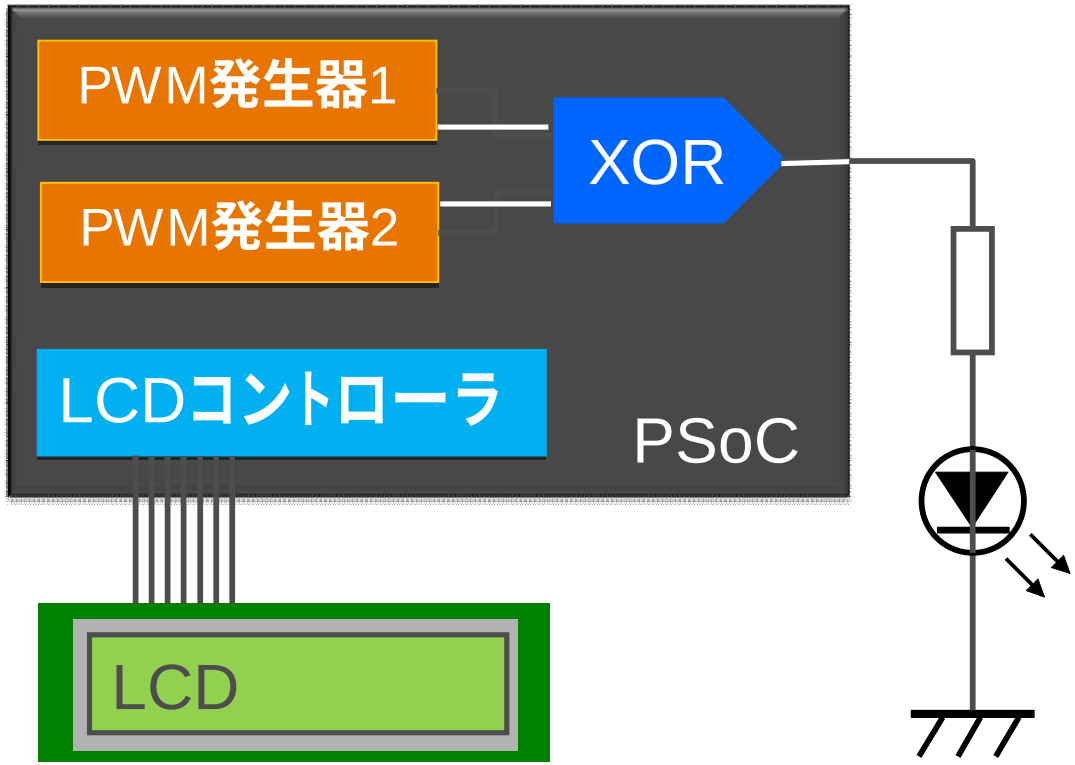
クローンプロジェクトの作り方と
モジュール相互の論理関数の作り方がポイント



lab2_pwm_lcd

- **周波数の少し違う2つの**
  PWM8
  の出力信号のXOR**をとり,**
  デジタル的に”うなり共振”
  を作り出し,LED**を蛍の光**
  のように明滅させる

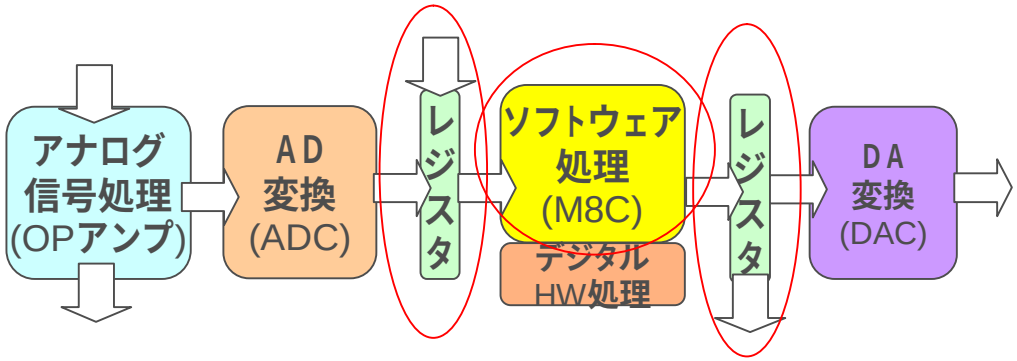
- LCD**を使い文字を表示**



ラボ

gpio_poll

レジスタを直接読み書きしてIOをリードライトする演習



PSoCに外部スイッチをつけるには 1

PSoCの入出力ピンに外部スイッチを接続すればシステム動作をコントロールすることができます。一見簡単そうですがデバイス外部信号の入出力インターフェースにはさまざまなポイントがあります.

1.入出力ピンのデータの読み込み

PSoCの入出力ピンはレジスタに割り付けられています.MCUは8ビット単位でこのレジスタを読み書きします.データを操作するピンが8ピンに満たない場合は他のピンのデータはマスクする必要があります.(読み込み時にはマスクで必要なピンのデータだけを取り出します.書き込み時にはまずバイト単位でレジスタのデータを読み込み変更するビットの値だけを変えてその上でバイト書き込みをします-PSoCの入出力ピン-GPIOの項目を参照してください)

2.プルアップとプルダウン

デバイスのピンの値を論理1(電圧H)論理0(電圧L)として扱うためには入出力のピンをプルアップまたはプルダウンしてスイッチを押していない状態の電圧を確定してやる必要があります.回路がオープンの状態(ハイ･インピーダンス)では論理値は0にも1にも確定しません

PSoCに外部スイッチをつけるには 2

3.ポーリングと割り込み

MCUから外部ピンの状態を監視する方法には、プログラムから定期的にレジスタの値を読み込む“ポーリング”と外部のハードウェアから動作中のMCUに対してハードウェア割り込みをかける“ハードウェア・インタラプト”という方法があります.(シンプルに割り込みと呼ぶ場合もありますが,ソフトウェア割り込みと区別する意味で,ここではハードウェア・インタラプトと呼んでいます-ハードウェア割り込みはMCUアーキテクチャレベルのものですがソフトウェア割り込みはオペレーティング・システムレベルのものです)通常は処理の速度の問題がない場合には簡単なポーリングにします.

4.チャタリング

外部スイッチは接点部分で電流のON/OFFを実現していますが、この接点ではON/OFF動作時の短い時間に”チャタリング”現象が発生し数ミリから数十ミリ秒の間電圧0と1の間で不安定な論理値を示します。これは回路上のノイズとして基板上の付加回路で解決したりソフトウェアのプログラムで解決します。

GPIO(入出力)関連レジスタ

入出力のピンは,レジスタにアサインされています。
レジスタにデータを書き込めば、ピンにデータが出力されます
ピンの状態を知るには、レジスタの値を読み込みします

ポートの入出力にはPRTxDRレジスタ変数を使用します。
　PRT1DR ： ポート1のデータリード・ライト・レジスタ

その他　PSoC Designerで設定可能なレジスタ
　PRTxDM2/DM1/DM0：動作モード設定
　PRTxIE　：割り込み発生許可・不許可レジスタ
　PRTxIC1/IC0：割り込みモード設定レジスタ
　PRTxGS：ポートをCPU・デジタルブロックで使うかを
　選択するレジスタ
＊xにはポート番号 (0,1,2,3…)を入れる

GPIOの使い方

ポート(8ビット単位)の読み込み

```
unsigned char prtData,pinData;
prtData = PRT1DR;                    // Port1の状態を読み込み
pinData = PRT1DR & 0b00000001; // ビットマスクでP1.0の値を得る
pinData = PRT1DR & 0b00000010; //ビットマスクでP1.1の値を得る
```

ポート(8ビット単位)への書き込み

```
PRT1DR |= 0x01;                    // Port1.0をHighにセット
(PRT1DR = PRT1DR | 0x01;       // Port1.0をHighにセット)
PRT1DR |= 0x02;                    // Port1.1をHighにセット
PRT1DR &= 0b11111011;              // Port1.2をLowにセット
```

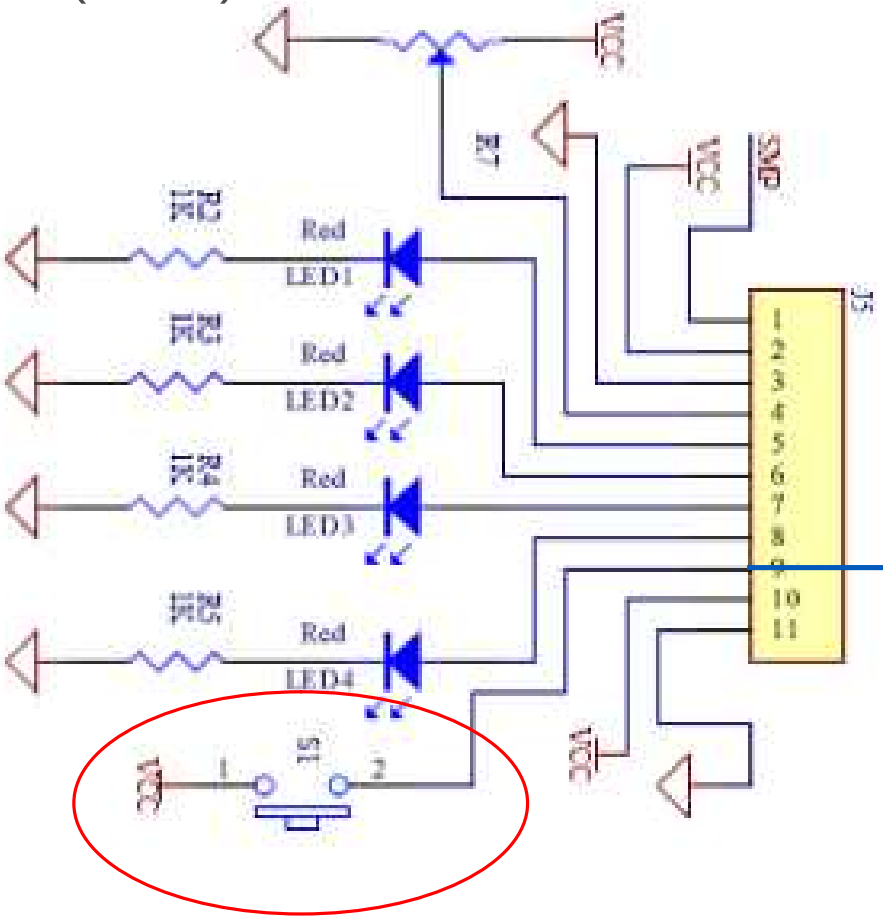
注意点

M8Cには**GPIOの1ピンに対するビット命令がありません。ポート単位8ビットの読み書きしかできませんから,ビットマスクで1バイト単位でR/Wします)**

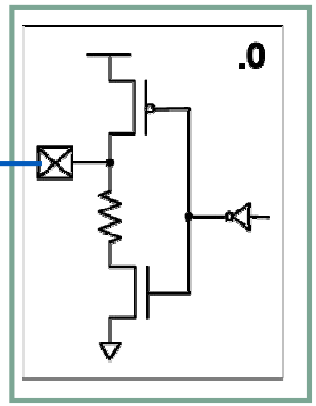
尚入力の値は実際の外部ピンの状態をそのまま読み込みます.*出力値についてはGPIOのドライブロジックに対してデータを出力しますから外部ピンの状態(値)はドライブロジックの回路に依存します.*

GPIOのプルアップとプルダウン

- S1は基板上でVCC側(5V)に接続されているので,デバイスのGPIOはプルダウンVSS(0V)側に設定しておきます。
- こうするとSWがOFFのときはピンは0V(LOW)になり,SWをONにすると5V(HIGH)になります。



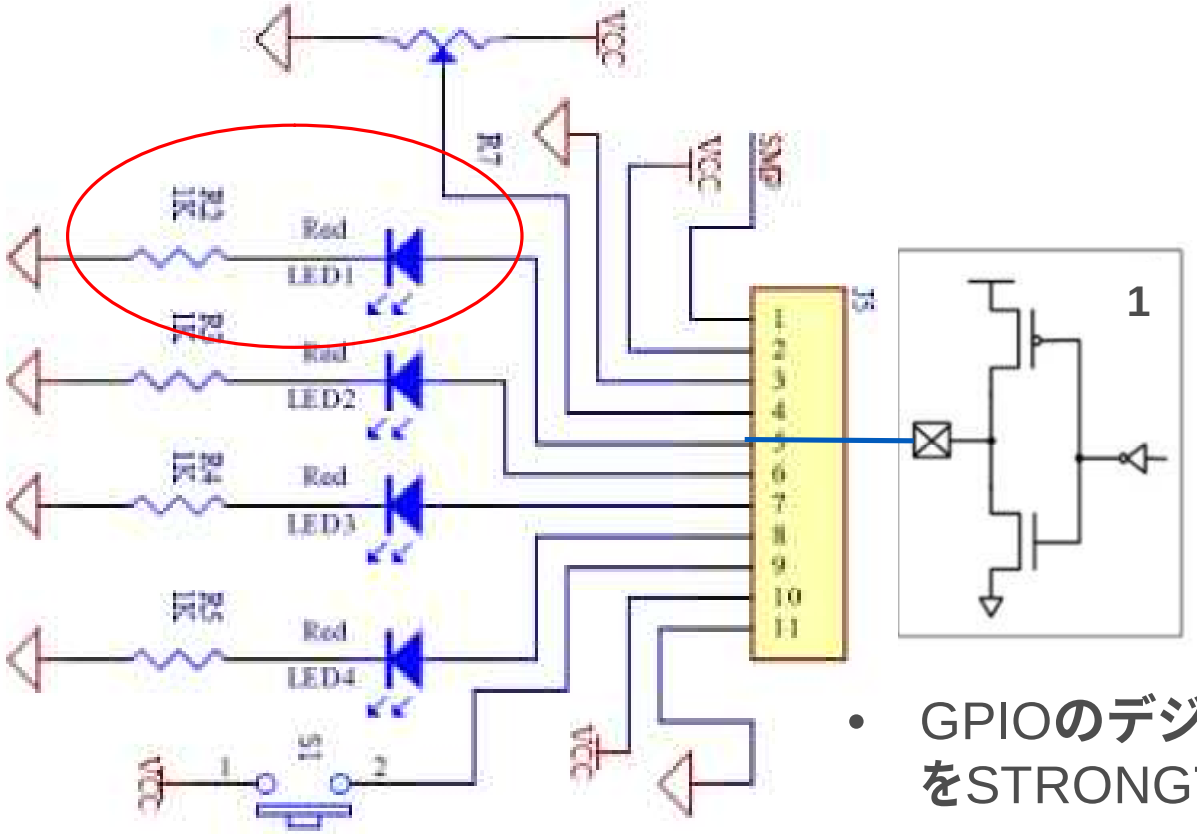
- もし基板がVSS(0V)側にピンを配線してある場合はプルアップします。この場合はON時の論理も逆になります



GPIOのプルダウン設定

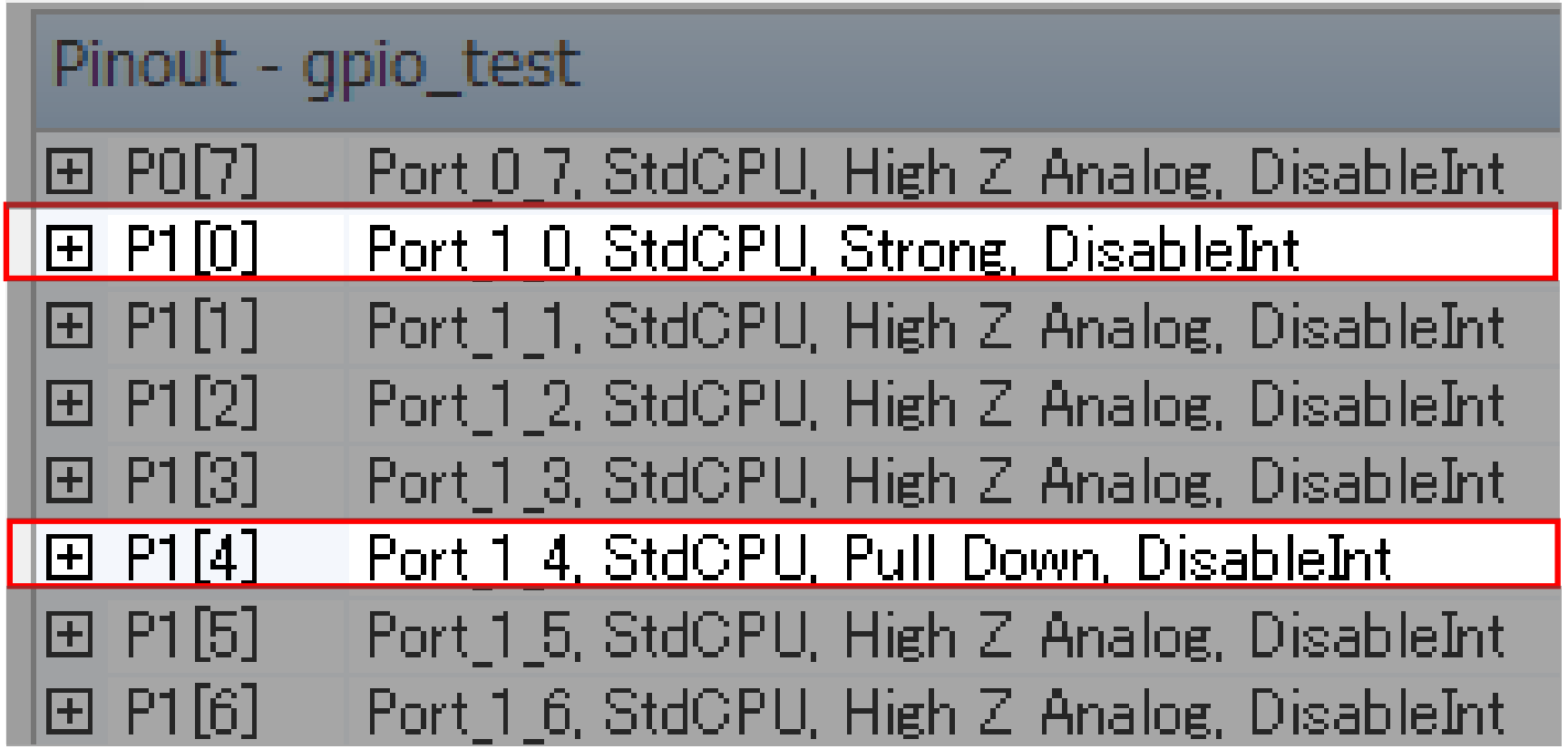
基板のLED配線とGPIOの設定

- LEDは基板上でVSS(0V)側に接続されているのでデバイスのGPIOはSTRONG(出力)モードに設定する. これで出力ピンをスイッチ･ドライブしてLED電流をON/OFFできる.



- GPIOのデジタル入出力部をSTRONG設定

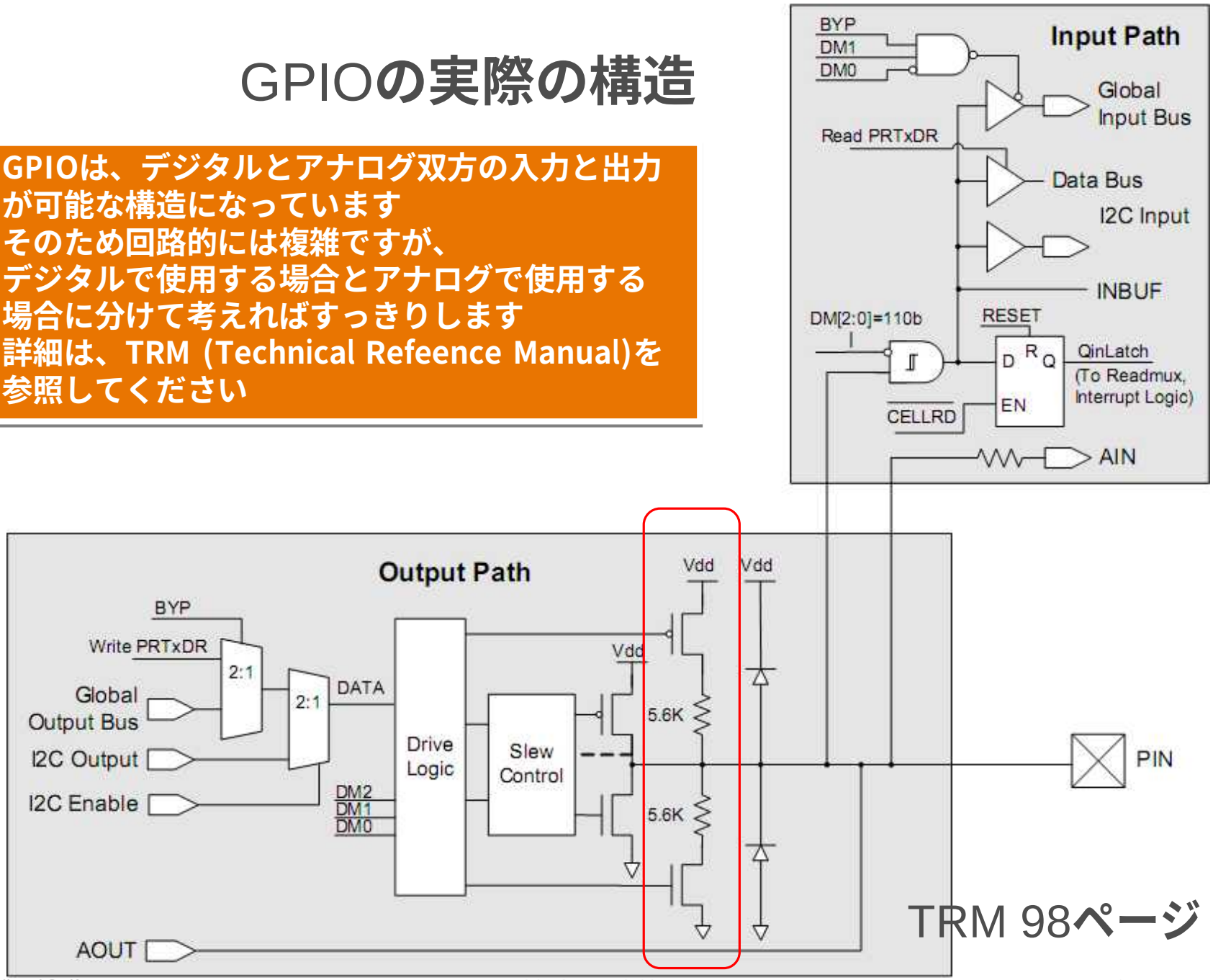
システム･リソース GPIOの設定



- P1[0] LED用　　Output
- P1[4] Switch用　　Input

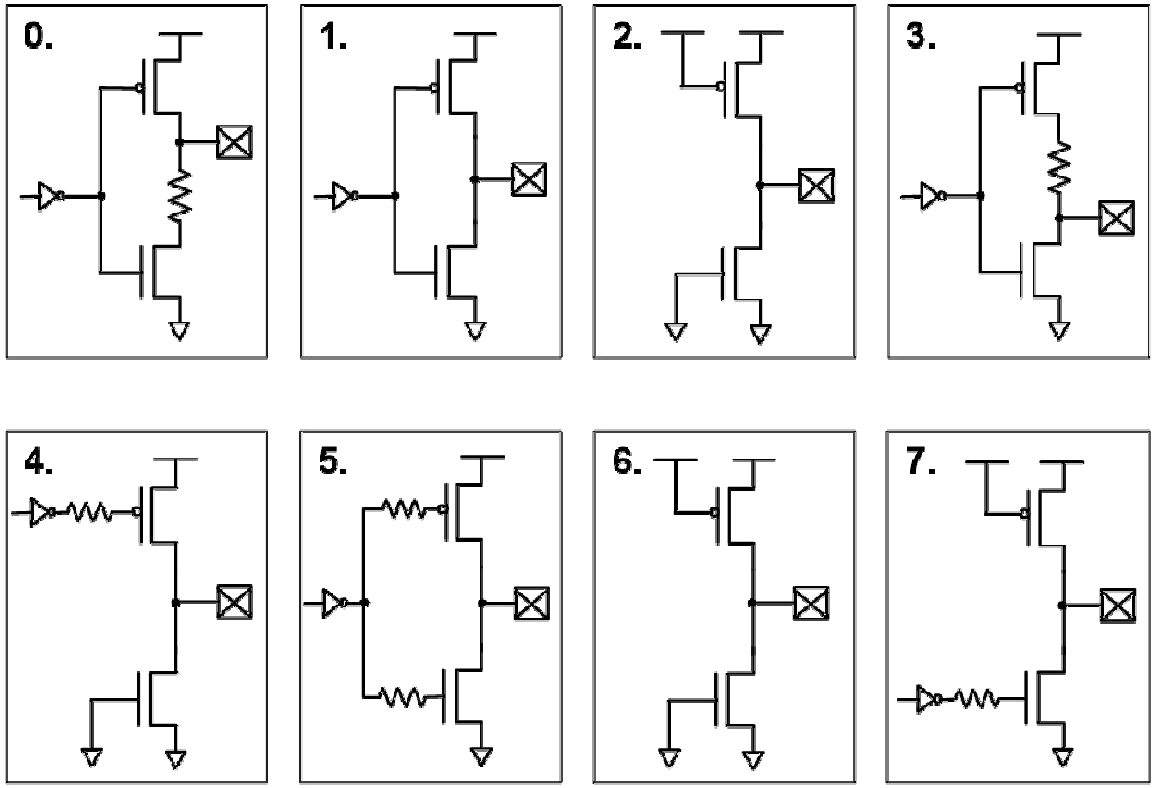
GPIOの実際の構造

GPIOは、デジタルとアナログ双方の入力と出力が可能な構造になっています
そのため回路的には複雑ですが、
デジタルで使用する場合とアナログで使用する場合に分けて考えればすっきりします
詳細は、TRM (Technical Refeence Manual)を参照してください



TRM 98ページ

GPIOのDrive Mode(TRM 98ページ)



番号	説明
0	プルダウン
1	ストロング (出力用)
2	ハイインピーダンス
3	プルアップ
4	オープンエミッタ
5	Slowストロング (出力用)
6	ハイインピーダンスアナログ
7	オープンコレクタ

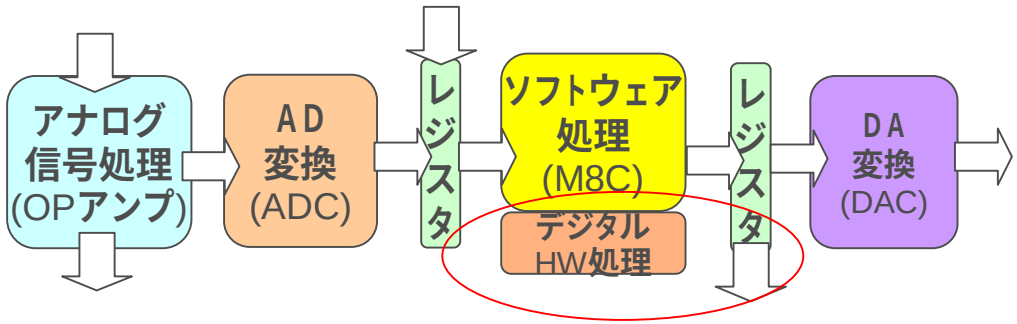
- Motor プロジェクトではアナログ回路からの出力をI/Oピンに出したので、ドライブロジック出力部はハイインピーダンスにしました

タイマー割込みを使用してPWMで音楽を演奏

ラボ (ドレミ)

timer_pwm2

タイマーからの割り込みとPWM周波数をレジスタ値の
書き換えでのコントロール法がポイント



音楽を演奏するために必要な要素

1.音符は、音程と時間の要素で成り立つ

2.音程を作るために周波数を生成する機能が必要

PWMユーザーモジュールで実現(実際的には16ビット程度必要-PWM16使用)

3.音程を変えるにはPWMのPeriod レジスタとPulse Width レジスタの値をプログラムで変えていけばよい

4.音の長さ(時間の要素)はどう実現するか

タイマー・モジュールを動作させておき一定時間ごとにMPUに割り込みをかける.この割り込みを演奏の最小制御単位時間として、”音符”を実現する

ハードウェア割り込み

1.ユーザーモジュールのハードウェアは、一定の条件が成立したときMPUの割り込み信号ラインを駆動する

2.MPUはソフトウェアの処理を実行中にハードウェア割り込みが発生した場合は、それまでのソフトウェアの処理を中断して、割り込みの処理に移る

3.MPUは、現在の（作業中の）レジスタの状態(値)をスタックにPUSHしてから割り込み処理ルーチンの実行に入る。割り込み処理ルーチン(ISR = Interrupt Service Routine)は、割り込みを行ったハードウェアごとに決められているジャンプアドレスにとんで特定の割り込み処理時のプログラムを実行する

4.割り込み処理が終了したら、MPUはスタックにPUSH(退避させておいた)データを順にPOP（取り出して）レジスタの状態を復帰させる。これによって割り込み前の処理状態に戻ることができる。

おまけ

複数のハードウェアから同時に割り込みが発生した場合は、プライオリティー・エンコーダで選択する。割り込みはプログラムからマスクすることもできる。

割り込み発生時のジャンプ先は割り込み・ベクタ・テーブルに書かれている。

いろいろな割り込み

1.ハードウェア割り込み

最強の割り込みは、RESETでこれはプログラムでマスクができない(Non Maskable Interrupt) マスク可能な割り込み（Maskable Interrupt ）はプログラム上で特定のレジスタ値を割り込み不許可に書き込むことで設定できる

2.ソフトウェア割り込み

オペレーティング・システムなどがサポートしてる。MS-DOSではINT21

ソフトウェアのプログラム処理で実行する

3.割り込みを使わないで同等の処理を行う方法 – (プログラムによる)ポーリング

ポーリングで、特定のレジスタの値を直接読みに行き、そのレジスタの値によって処理を決める。例えばハードウェアのタイマーが勝手にカウントしているとき、ソフトウェアから何度も何度もポーリングでカウンタ値を読み出しに行く。

タイマーが一定値になったときにプログラムで条件分岐して処理を行う

ポーリングの問題点:時間が計測しにくい.プログラムが煩雑になる.処理が遅くなる.

（ソフトウェアのポーリングのタイミングよりはるかに高速でハードウェアのタイマが動作していたらどうなるかを考えてみればよい）

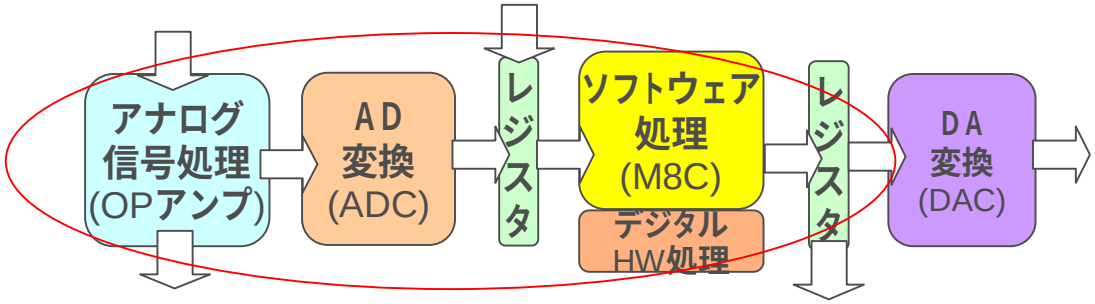
入力アナログ電圧をPGAで増幅しこれをAD変換して値をLCD表示 - デジタル電圧計を作ります.

ラボ

lab3_adc

AD変換の方法がポイントになります.

デジタル化されればパルス幅に変換できます.



ＡＤ変換によって何ができるか

これまでは、出力側のアプリケーション演習を行ってきました。出力には、アナログ出力、デジタル出力がありました。表現の形態としては、ＬＣＤディスプレイへの表示、サウンド出力、ＬＥＤの点滅などがありました。

しかしこれらは、画一的な機能の表現です。連続的な入力の変化によって出力を変化させることはありません。入力によって出力を連続的に制御したり変化させるためには、何らかの形で入力情報を連続的に取り込む必要があります。

ＡＤ変換では、アナログ的な電圧の変化をデジタル量に変換できますから、外部の変化量を電圧変化への置きかえれば、アナログ的な連続的な変化量を扱うことができます。

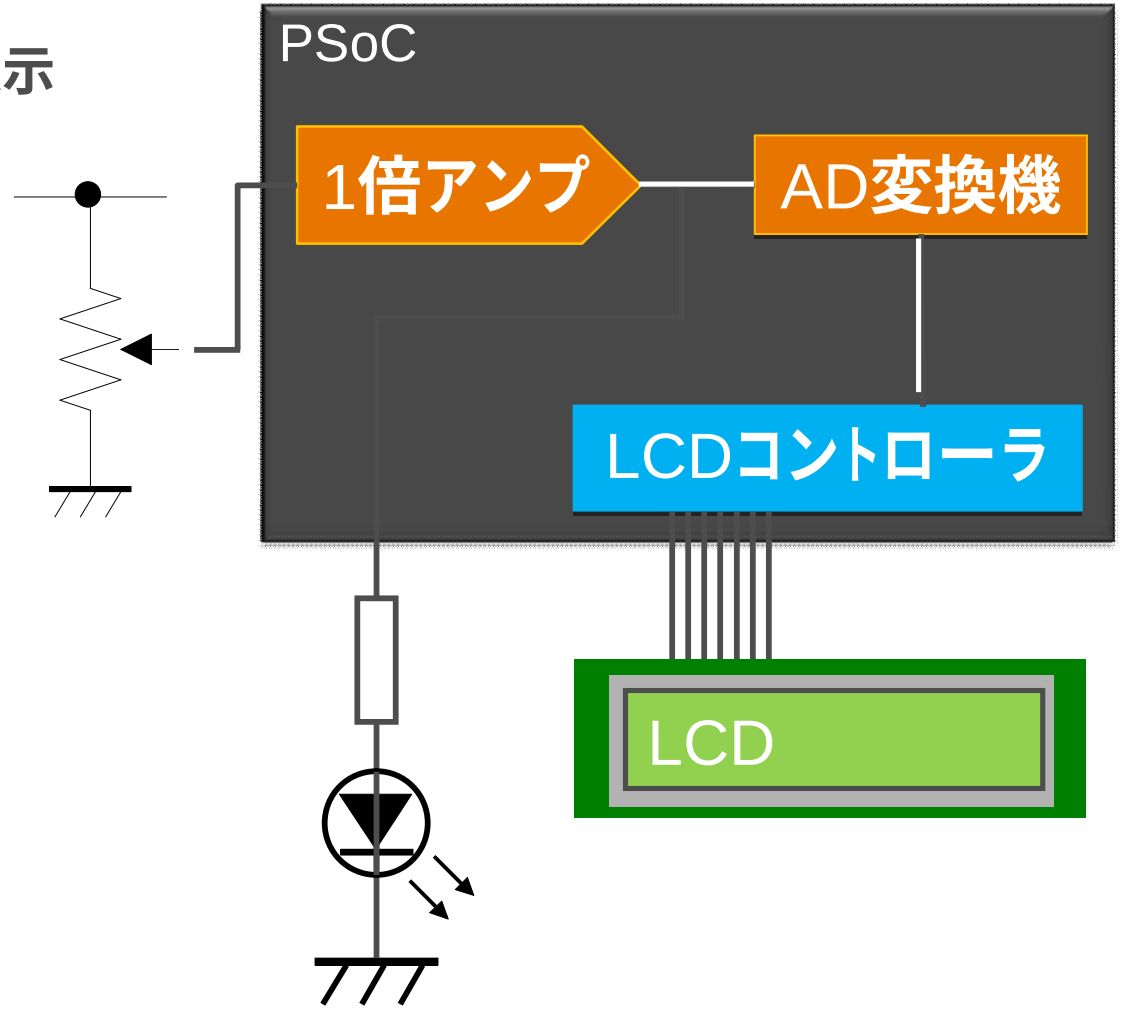
アナログ的な入力素子としては、さまざまなセンサーがありますが、この多くは電圧出力になっています。

よって、センサー出力　ー＞　増幅　－＞　ＡＤ変換　－＞　デジタル数値化　が可能になりますから、これによりＭＰＵで自由にデータの処理や外部制御や出力が可能になります

例えば、温度によって音が変化するシンセサイザ、3軸の加速度（ジャイロ）センサをコントローラにした楽器なども可能です。3入力をまとめてAD変換する場合はTRIADCユーザーモジュールを使用します.

lab3_adc

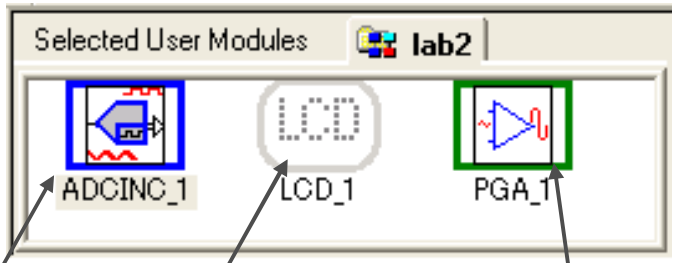
- VRの電圧をLCDで表示
- VRの電圧を
  そのままLEDへ



3つのユーザーモジュールでの設計

- ユーザーモジュールは,パラメタライズされた機能ライブラリ(IPのようなもの)
- 選択したユーザーモジュールは自動的に内部リソースのコンビネーションで実現される
- 内部リソースは,コース･グレインで作りこまれているので集積度が高く効率が高い
- 極少の配線数で機能を実現できるので,シリコンの使用効率が高い

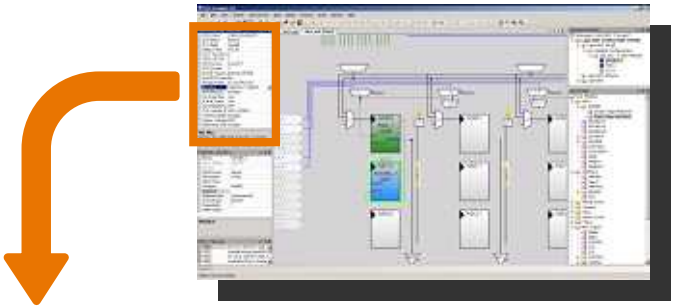
入力アナログ電圧のデジタル表示回路例



機能可変ADコンバータ　LCDドライバ　可変利得アンプ

ＡＤ変換時のRef Muxの設定

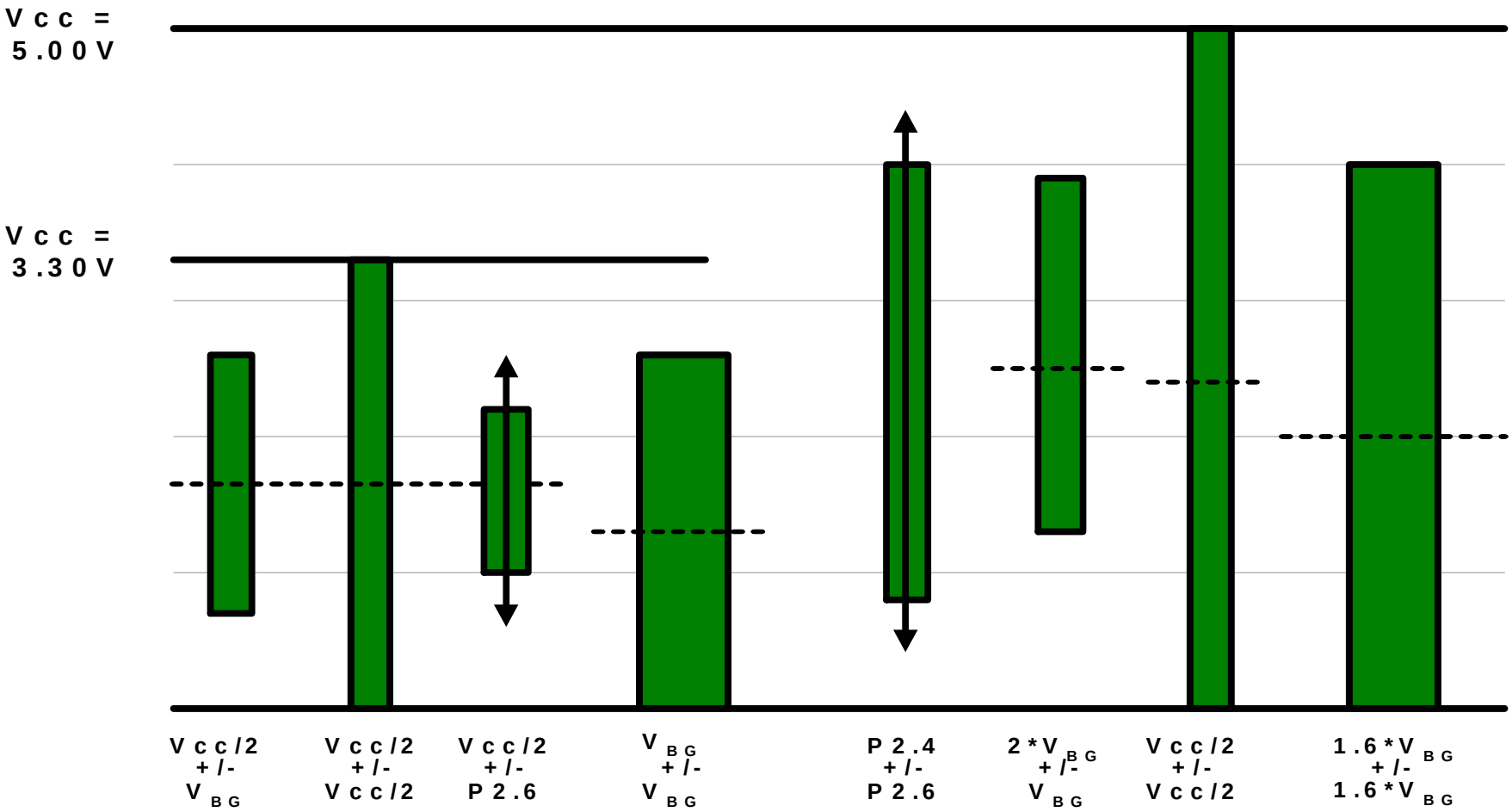
Ref Mux はアナロググランドのレベルを決定し
アナログ信号の上下の振幅範囲を決定します
これはPSoCのオペアンプが単電源のため
0Vをグランドレベルとしてマイナス側の信号を
扱えないため基準電位をかさ上げします。
このようにしてかさ上げ設定したレベルを
アナロググランド電位と呼んでいます.
lab_motor のPGAのRefの選択枝に
AGNDがありましたが,ここで設定します



Global Resources - lab3_adc

CPU_Clock	3_MHz (SysClk/8)
32K_Select	Internal
PLL_Mode	Disable
Sleep Timer	512 Hz
VC1= SysClk/I	3
VC2= VC1/N	1
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Internal 24_MHz
SysClk*2 Disal	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePu	OFF
Trip Voltage [L	4.81V (5.00V)
LVDThrottleBa	Disable
Supply Voltage	5.0V
Watchdog Enat	Disable

RefMuxの設定と範囲について



RefMuxについて

RefMuxはADの入力レンジを決定

表示	5V駆動	3.3V駆動
[Vdd/2]+/-BandGap	2.5 V±1.3 V	1.65V±1.3V
[Vdd/2]+/-[Vdd/2]	2.5 V±2.5 V	1.65V±1.65V
BandGap+/-BandGap	1.3 V±1.3 V	1.3V±1.3V
1.6 BandGap+/-1.6 BandGap	2.08V±2.08V	**使用不可**
2 BandGap+/-BandGap	2.6 V±1.3 V	**使用不可**
2 BandGap+/-P2[6]	2.6 V±P2[6]V	2.6 V±P2[6]V
P2[4] +/-BandGap	P2[4] V±1.3V	P2[4] V±1.3V
P2[4] +/-P2[6]	P2[4]V +/-P2[6]V	P2[4]V +/-P2[6]V

BandGap電圧は内部で1.2xx..Vから昇圧した1.3Vとなります.実はこの電圧もレジスタ値でトリミングできます.

Reference 電圧を外部から入力することができますが,これができるピンはPort2[4]です.

3210EVAL1ではPort2はLCDに接続されています。

AD変換ステータスをポーリングでチェック

- ADCINC_fIsDataAvailable 値は、AD変換が終了してデータが読み込み可能な状態になったときに0になるレジスタ

このプログラムでは、ポーリングで
値を読みに行って、0のとき
(データが読み込み可能のとき)
adc_dataにAD変換値を代入する

ADCINCユーザーモジュールは、
割り込みが用意されていないので
ポーリングでステータスを読む

AD変換のユーザーモジュールには
12種類あり、8ビット以下のものは,割り込みが使える

Start Page | lab3_adc [Chip] | main.c

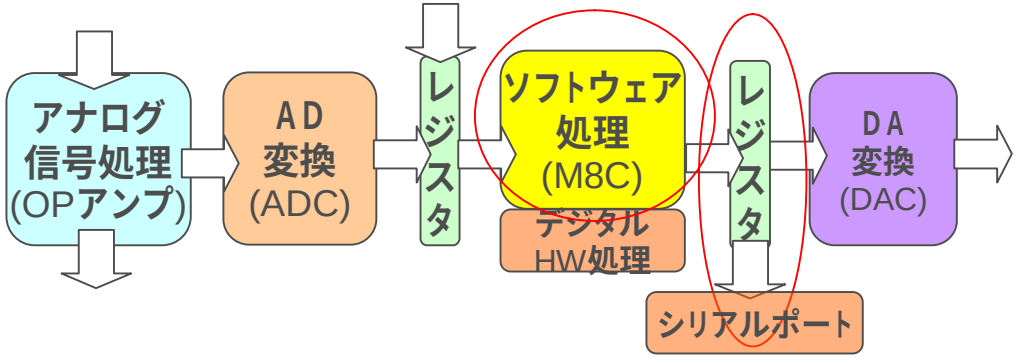
```
1  //-----------------------------------------------------------
2  // C main line
3  //-----------------------------------------------------------
4
5  #include <m8c.h>        // part specific constants and
6  #include "PSoCAPI.h"    // PSoC API definitions for all
7
8
9  void main()
10 {
11     unsigned int adc_data;
12     PGA_Start(PGA_HIGHPOWER);
13     LCD_Start();
14     LCD_InitBG(LCD_SOLID_BG);
15     M8C_EnableGInt;
16     ADCINC_Start(ADCINC_HIGHPOWER);
17     ADCINC_GetSamples(0);
18     while(1){
19         while(ADCINC_fIsDataAvailable() == 0);
20         adc_data = ADCINC_wClearFlagGetData();
21         LCD_Position(0,0);
22         LCD_PrHexInt(adc_data);
23         LCD_DrawBG(1,0,16,(adc_data/50));
24     }
25 }
26
```

大文字のI

ラボ

uart_1

シリアルポートを使っての通信演習



シリアル伝送とパラレル伝送

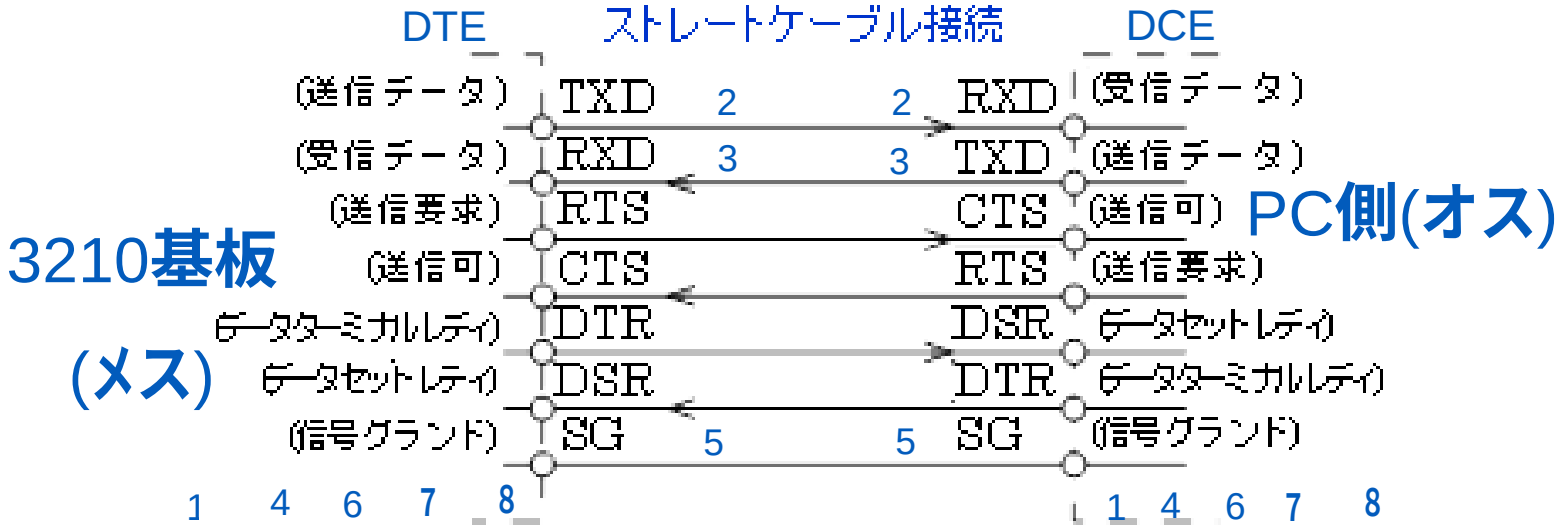
- プロセッサで処理するデータの単位は、8ビット、16ビット、32ビット、64ビットなどのまとまった単位。このデータを受け渡すハードウェアにはパラレル(並列)方式とシリアル(直列)方式がある。
- パラレル方式の信号線数は、8/16/32/64などとなっており、8ビットのハードウェア経由で16ビットのデータを送る場合は、2度にわたって転送を行う。このように信号周波数が極端に高くない場合は、並列に扱うデータ幅のビット数が多いほど転送データ量を多くできる
- 反面ビット数を増やすと伝送線数が増えるので転送データが少ない場合は、シリアル伝送を使用して線数を少なくする
- 伝送周波数が極端に高くなるとパラレルでは伝送距離と各ビットの同期をとるのが困難になり、差動信号を利用したシリアル通信方式を用いる
- ハードディスクの例
  - パラレル ATAPI/IDE 133MByte/Sec (1GBit/Sec)
  - シリアル SATA 1.5G- 6GBit/Sec
- 参考：SerDes(SERializer/DESealizer)

シリアル通信について

- シリアル通信は、比較的低速度の伝送では電圧伝送、高速の伝送では、差動信号が用いられる。(演習のEIA-574/232は電圧伝送)
- シリアル通信では、送信側は、パラレルのデータをシリアルに変換して、1本の信号線に複数ビットの信号を送りだす。このことをパラレル･シリアル変換といい、受信側は逆のシリアル･パラレル変換を行い、もとのパラレルのデータを復元する。これにはシフトレジスタ回路が用いられる。
- シリアル通信のポイント
  - 最初の1ビット目の識別はどうするか、データの最後の識別は？
  - 転送されたデータの正しさはどのように検証するか
  - 受信バッファの働き(バッファがいっぱいになったら?)
  - 転送データ内のデータ･コードと制御コード
  - プロトコルとデータフォーマット
  - ハードウェアとソフトウェアでどう実装するか
- 参照ＵＲＬ http://www.fukufukudenshi.com/LineMonitor/RS-232C.html

EIA-232-D/E(RS-232C)通信

- コネクタ DSUB9ピンEIA-574(または25ピンEIA-232)
  - DTEタイプ(オス)と DCEタイプ(メス)の信号ピン接続がある
  - DCE-DTE接続の場合は、オスメス・ストレートケーブル使用
  - DCE同士、DTE同士の接続は、クロスケーブルを使用
- 演習では、2線または4線ハンドシェイクを使用
  - 2線：TX, RX と信号グランド (ノンハンドシェイク)
  - 4線：TX, RX, RTS, CTSと信号グランド

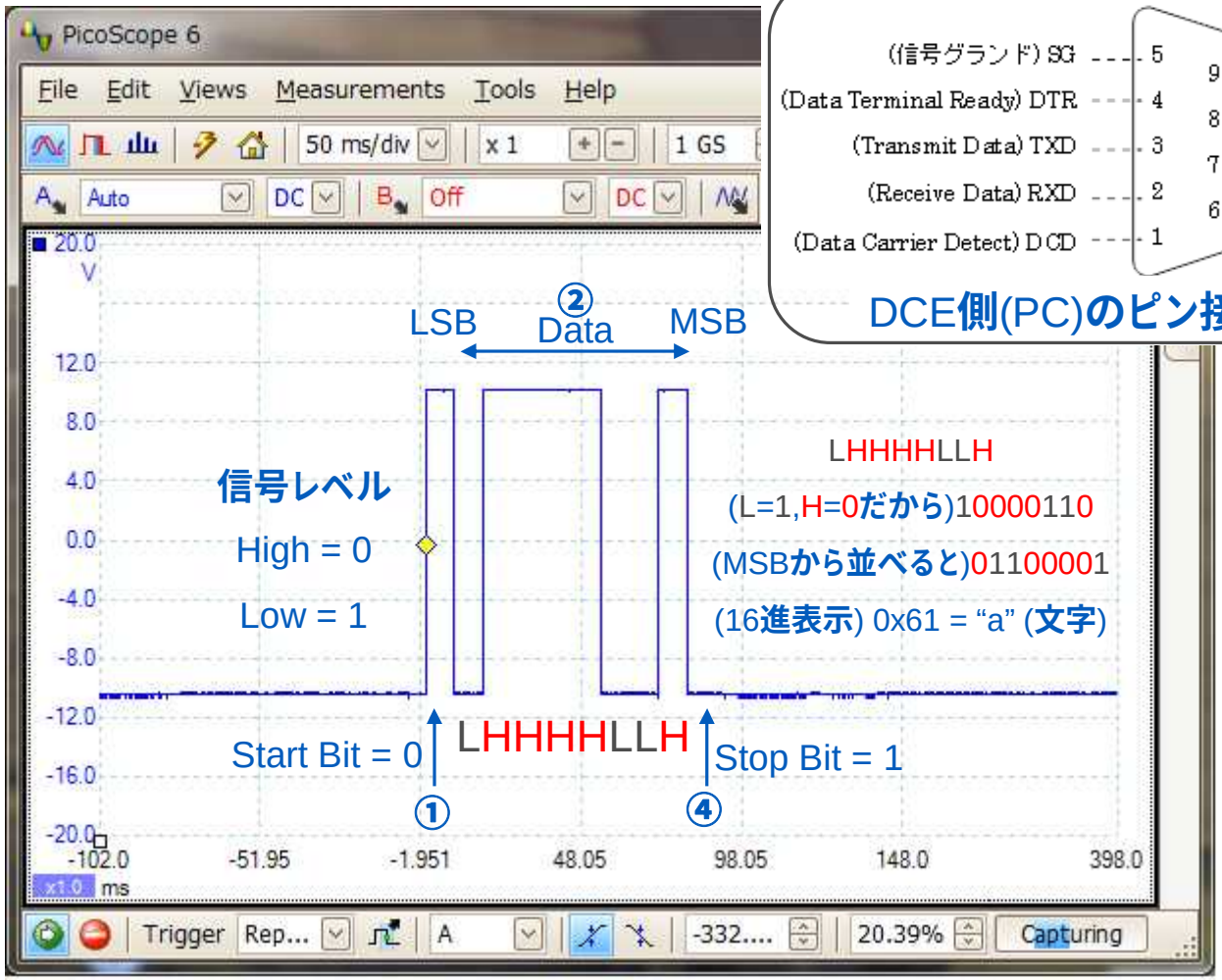


EIA-232-D/E(RS-232C)通信と規格

- コネクタ DSUB9 (EIA-574)またはDSUB25(EIA-232)ピン
  - DCEタイプ(オス)と DTEタイプ(メス)の信号ピン接続がある
  - DCE-DTE接続の場合は、オスメス・ストレートケーブル使用
  - DCE同士、DTE同士の接続は、クロスケーブルを使用
- 信号レベル 最大+12V(論理1) から-5V(論理0)
  - 実際は多種あり（+5V (論理1)　0V (論理0) TTLレベル）
  - 3210基板：PSoCTTLレベル>MAX232(互換IC)で変換
- 通信速度：規格では20KPBS (Bit per Second)
  - PCでは～115KBPS程度：PC標準は、NS16550A
- 信号フォーマット
  - ① スタートビット : 必ず1ビットで 0 (信号レベルH)
  - ② データビット : 8または7ビット,LSB側から並び,1はL, 0はH
  - ③ パリティービット: なし(0ビット)またはあり(1ビット)
  - ④ ストップビット：必ず1でビット長は, 1, 1.5, 2, 2<

DCE (PC) TX3番ピン送信信号波形(実測)

送信文字 ”a” ASCIIコード符号 “01100001” = 0x61



DCE側(PC)のピン接続例 (ピン面)

①スタートビット

②データビット
(LSBからの
並びです)

③パリティービット
(なしに設定)

④ストップビット

“a” Character Transmit to Pin3(TX) at 110BPS 8bit Non-Parity 1 Stopbit

黄色のダイヤはトリガポイント

PCのシリアル通信セットアップ

—通信ソフトウェア

- Windows　XPの場合
  - スタート>すべてのプログラム>アクセサリ>通信>ハイパーターミナル
- Windows　7/ Vistaの場合
  - 以下のアプリケーションをダウンロードしてインストール
    - TeraTerm(推薦) (TeraTerm以外のオプションはインストールしない)

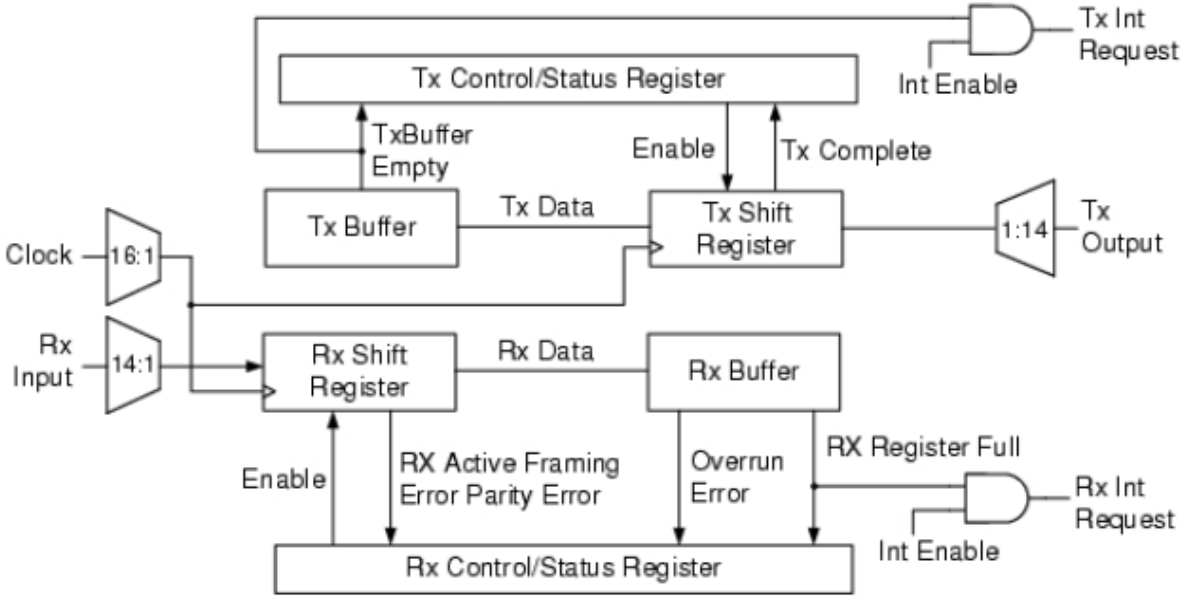
—通信のためのハードウェア

- PC装備のDSUB9ピンオスコネクタ(DCE)
- Note PCなどコネクタがない場合は、USB-シリアル・インターフェース・ケーブルを使用(Win-XP, Vista, 7の32Bit版)
  - デバイスドライバが64ビット非対応の場合があるので注意
  - 秋月ケーブルは32Bitのみ, FT232のドライバーは64Bit対応可能

PSoC UART回路ブロックの機能

- 最大6 Mビット／秒のバースト速度。• データフレーミングは、スタートビット、オプションのパリティビット、およびストップビットで構成されます。• 受信レジスタフルおよび/または送信バッファエンプティにおいて割り込みをかけるオプション。• パリティ、オーバラン、およびフレーミングエラー検出。• 高レベルの送受信API。

UARTユーザモジュールは、RS-232準拠データフォーマットの2線式二重シリアル通信をサポートする8ビット汎用非同期レシーバランスミッタです。受信および送信データフォーマットには、スタートビット、オプションのパリティ、およびストップビットが含まれます。プログラマブルクロックおよび選択可能な割り込みまたはポーリング方式の動作がサポートされます。UARTを初期化、構成、および操作するためのアプリケーションプログラミングインタフェース (API) ファームウェアルーチンが用意されています。バックグラウンドのコマンド受信および文字列出力をサポートする高レベルAPIも追加的に用意されています。



UARTレシーバー(受信)

レシーバは、デジタル通信タイプPSoCブロックの受信バッファ、受信シフト、および受信制御レジスタを使用します。

受信制御レジスタは、UARTユーザモジュールのファームウェアAPIルーチンを使用して初期化および構成します。受信の初期化は、UARTパリティの設定、オプションの受信レジスタフル状態での割り込みイネーブル、およびその後UARTのイネーブルで構成されます。

受信入力でスタートビットが検出されると、8分周ビットクロックが起動および同期し、受信ビットの中央でデータをサンプリングします。次の8ビットクロックの立ち上がりエッジで入力データがサンプリングされ、受信シフトレジスタにシフトされます。パリティがイネーブルになっている場合は、次のビットクロックでパリティビットがサンプリングされます。次のクロックでストップビットがサンプリングされる結果、受信データバイトが受信バッファレジスタに転送され、以降の1つまたは複数のイベントがトリガされます。

- 受信制御レジスタ内の受信レジスタフル ビットがセットされ、受信に対する割り込みがイネーブルになっている場合、関連する割り込みがトリガされます。
- データストリーム内の予想されるビット位置でストップビットが検出されない場合、受信制御レジスタ内のフレーミングエラー ビットがセットされます。
- 現在受信されたデータのストップビットの前にバッファレジスタが読み取られなかった場合、オーバランエラービットが受信制御レジスタ内にセットされます。
- パリティエラーが検出された場合、受信制御レジスタ内のパリティエラー ビットがセットされます。

完全に受信されたデータバイトの検出をポーリングするために、受信制御レジスタ内の受信制御レジスタフル ビットをモニタする必要があります。オーバランエラー状態を防止するために、データは、次のバイトが完全に受信される前に受信バッファレジスタから読み出さなければなりません。

UARTトランスミッタ(送信)

トランスミッタは、デジタル通信タイプPSoCブロックの送信バッファ、送信シフト、および送信制御レジスタを使用します。

送信制御レジスタは、UARTユーザモジュールのファームウェアAPIルーチンを使用して初期化および構成します。送信制御レジスタのイネーブルビットがセットされている場合は、内部8分周ビットクロックが生成されます。

送信するデータ バイトは、APIルーチンによって送信バッファレジスタに書き込まれ、送信制御レジスタ内の送信バッファエンプティステータスビットはクリアされます。このステータスビットを使用して、送信のオーバランエラーを検出および防止できます。

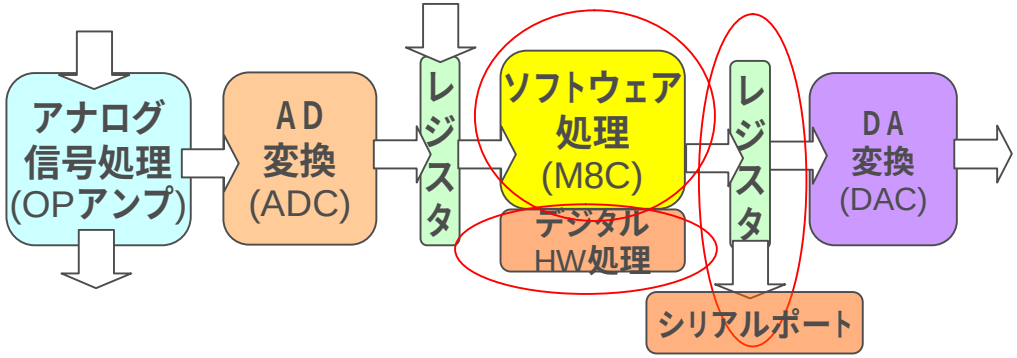
次のビット クロックの立ち上がりエッジでデータはシフトレジスタに転送され、送信制御レジスタの送信バッファエンプティ ビットがセットされます。割り込みイネーブルマスクがイネーブルになっている場合、割り込みがトリガされます。この割り込みによって、次のバイト送信のキューがイネーブルになり、現在のデータバイトの送信が完了したとき、利用可能な次の送信クロックで新しいバイトが送信されることになります。

スタートビットは、データバイトが送信バッファレジスタから送信シフトレジスタに転送されると同時に送信されます。連続するビットクロックによって、シリアルビットストリームが出力にシフトされます。ストリームは、データバイトの各ビット、最下位ビット優先、オプションのパリティビット、および最後のストップビットで構成されます。ストップビットの送信完了時に、送信制御レジスタの送信完了ステータスビットがセットされます。このビットは、読み取りが行われるまで有効です。新しいデータバイトが送信バッファレジスタに書き込まれると、そのデータバイトは送信シフトレジスタに転送され、データ送信はビットクロックの次の立ち上がりエッジで開始されます。

ラボ

pwm_uart_2

シリアルポート経由の外部コマンドでPWM音源の音階発生



uart_1 ラボと timer_pwm2 ラボの統合

- timer_pwm2では,定期的に割り込みを発生させて,ハ長調の音階を発生させました。 PWM16ユーザーモジュールのwriteperiod()関数のperiod値をプログラムで書き換える(レジスタの値を書き換える)ことにより,ハードウェアを変更して音の周波数を変更して音階を発生させました。
- このlabでは, タイマー割り込みを使用せず,シリアルポートからコマンドを読み込んで,コマンドに応じた音階の音を発生させます。シリアルポートからのコマンド入力は、接続したPCのターミナルソフトウェア(TeraTerm)を使用します。
- PCのキーボードをシンセザイザの鍵盤として使用しPSoCからPWM音源の音を発生させます。
- コマンドは、a,s,d,f,g,h,j,k とlの9文字を使用し,それぞれがド,レ,ミ,ファ,ソ,ラ,シ,ドと無音に対応します。コマンドの発行は文字コードを入力し、Enterキー押すことで実行されます。

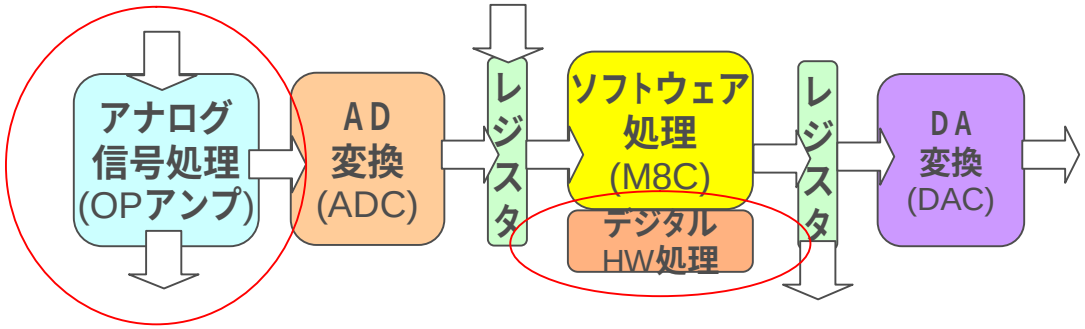
pwm_uart_2 プロジェクトの作成

- 先に作成したuart_1プロジェクトからpwm_uart_2クローン･プロジェクトを作り ,このプロジェクトにpwm16ユーザーモジュールを追加します。
- Cソースでシリアルポートから1文字を読み込みこの文字に対応した値をpwm16ユーザーモジュールのwriteperiod関数を使ってレジスタに書き込みます。
- UARTの制御手順は以下のとおりとなります
  - UART_CmdReset(); // Initialize receiver/cmd buffer
  - UART_IntCntl(UART_ENABLE_RX_INT); // Enable RX interrupts
  - UART_Start(UART_PARITY_NONE); // Enable UART
  - UART_CPutString("¥r¥nPSoC Synthesizer V1.1 ¥r¥n");
  - UART_PutString(strPtr); // Print out command
  - UART_CmdReset(); // Reset command buffer
- これら関数の詳細説明はuart_1 ラボに解説してあります.

信号処理によって応用が格段に広がってきます

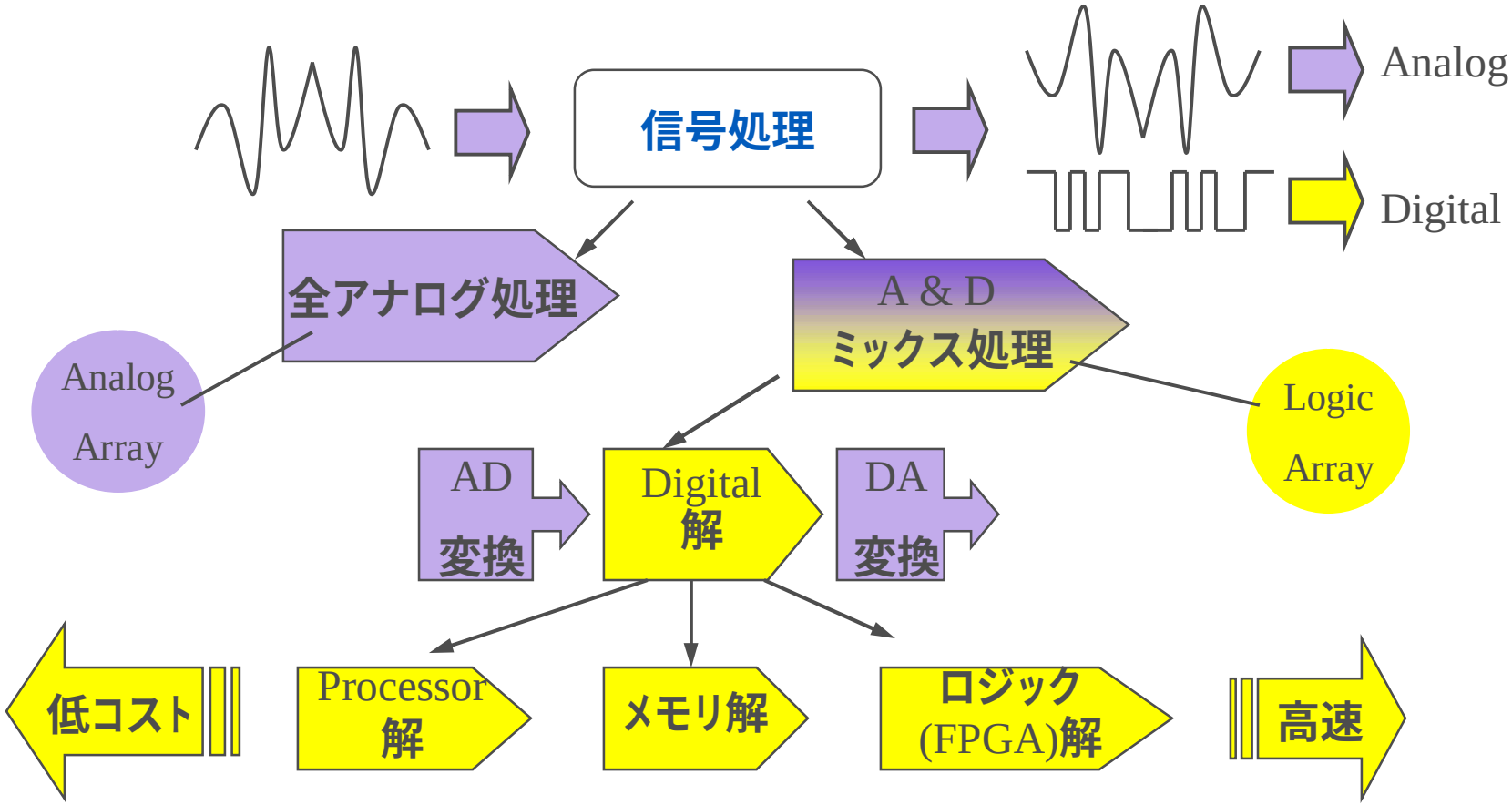
ラボ
b p f

バンドパスフィルタの設計



信号処理や制御の多様なアプローチ

アナログ,ディジタル･エンジン,プロセッサの3方式と直列,並列処理、直並列ミックス,多様なシステム構築解がある



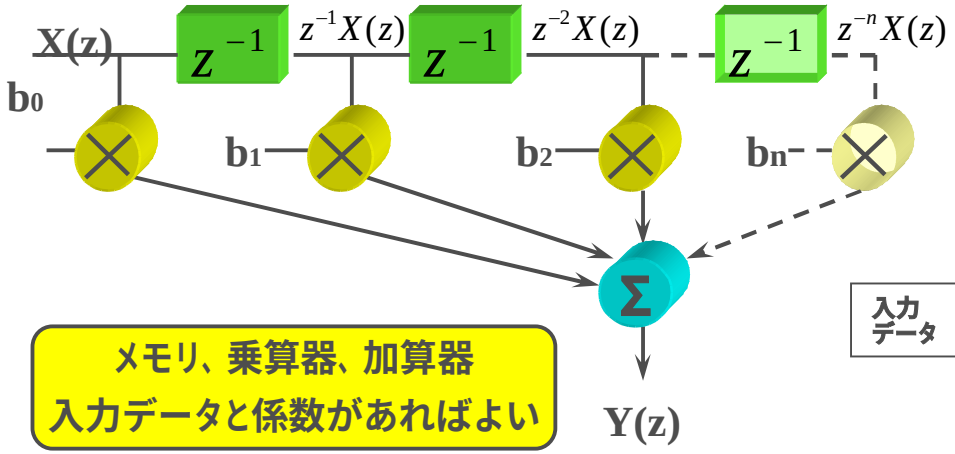
信号処理のデジタル実装解

スペック

Z変換 FIR Filter一般式の例

$$Y(z) = b_0 X(z) + b_1 z^{-1} X(z) + b_2 z^{-2} X(z) ....$$

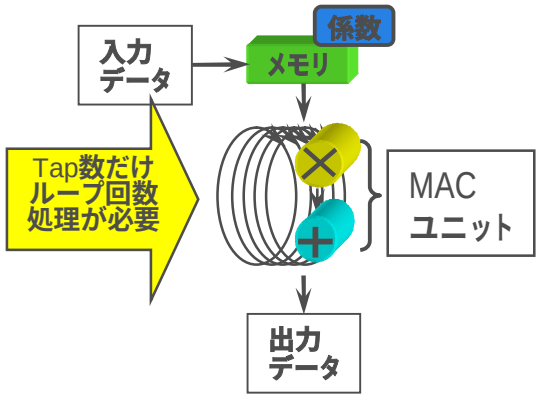
$$= (b_0 + b_1 z^{-1} + b_2 z^{-2} ....) X(z)$$



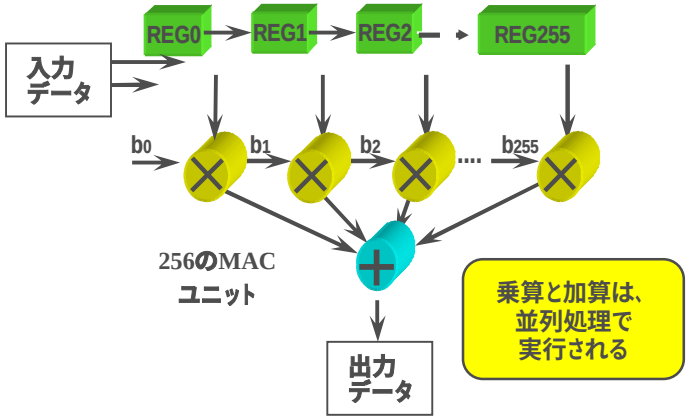
メモリ、乗算器、加算器
入力データと係数があればよい

Processor解

DS Processorの処理



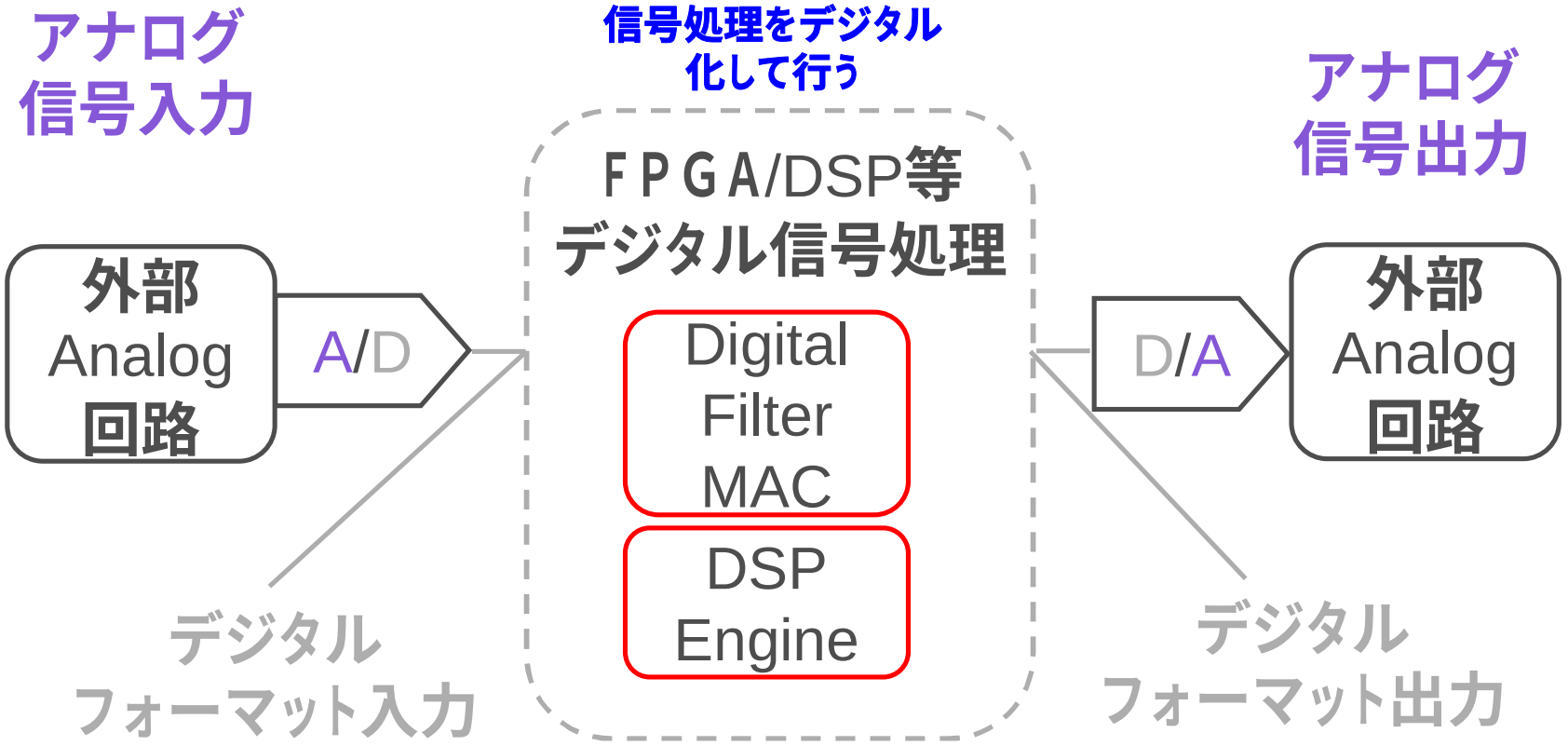
ハードロジックによる並列処理



乗算と加算は、並列処理で実行される

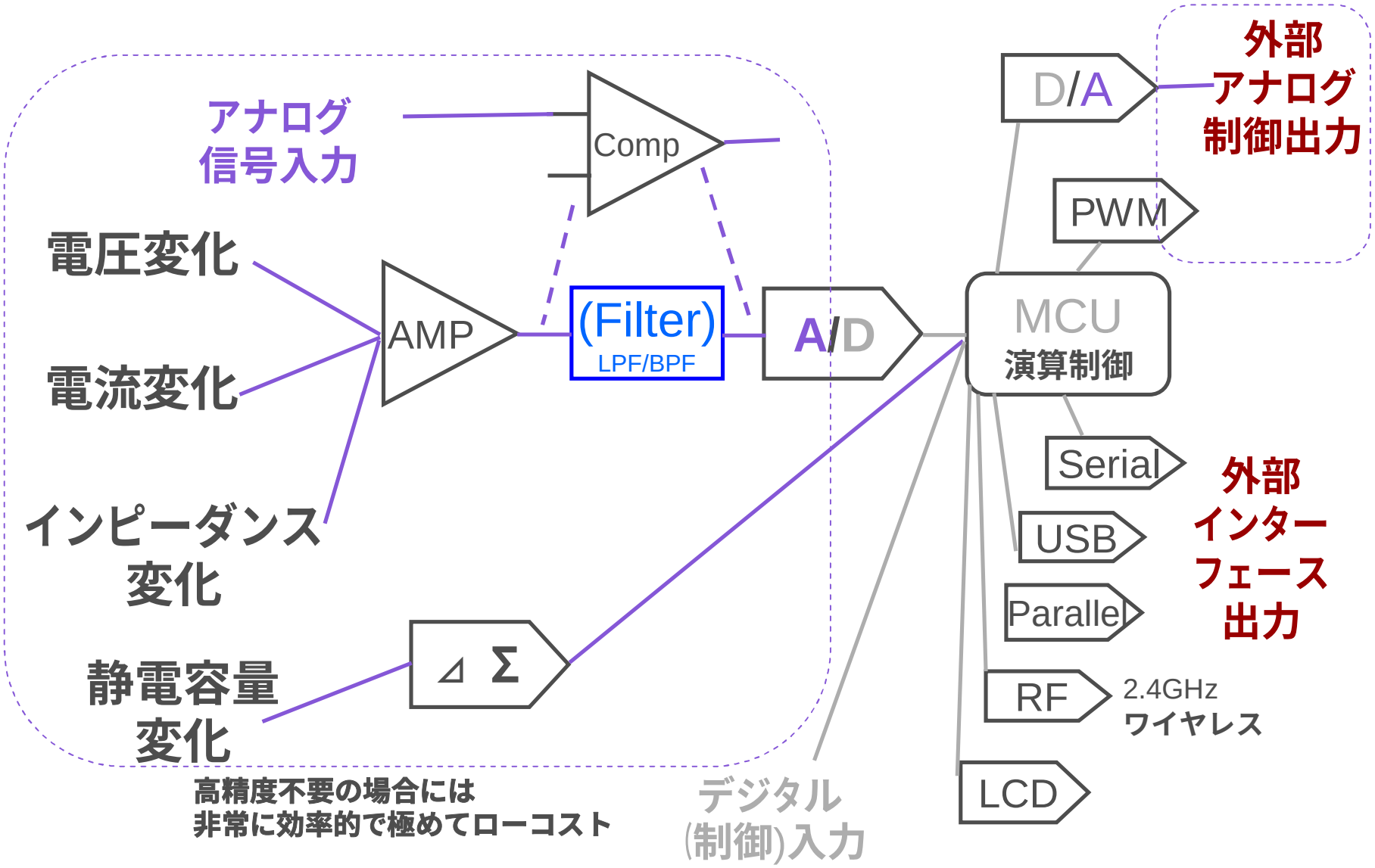
デジタル信号処理の考え方

現代のデジタル・コンバージェンスでは信号処理をA/D変換後に
すべてデジタルで行う方向に向かっている.この方法は元来膨大な
ハードウェアのバックグラウンドと高精度のD/AやA/Dが前提となるが
LSIの高集積化によって実用領域になってきたという経緯がある



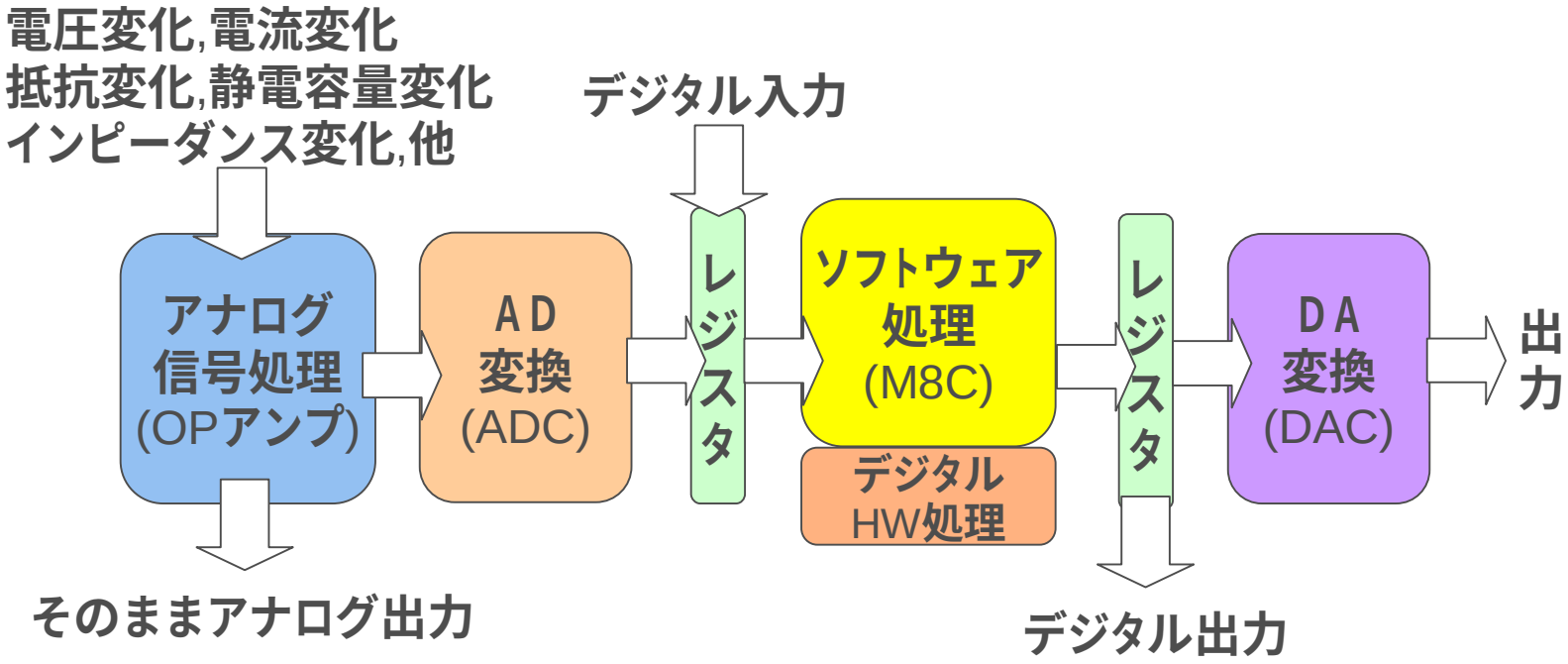
入口と出口はアナログが存在していることに注意

PSoCでは信号処理を前工程(アナログ処理)に移動



PSoCによる処理工程

外部現象変化をセンサーが電気信号に変換
センサーの出力はアナログ信号
信号処理(デジタルまたはアナログ)
AD変換してMPUで処理(レジスタがI/F)
ＤＡ変換して外部現象を発生(レジスタがI/F)



主要な各回路ブロック-ユーザーモジュール

デジタル･アナログ入出力

GPIO (General Purpose I/O)にてデジアナ双方のI/O可能

アナログ微小信号の増幅 : PGAとINSAMP

直流増幅(演算), 交流増幅 > PGA (単電源非反転オペアンプ)

高精度な計装アンプ > INSAMP (複数PGAトポロジー)

フィルタ回路 : LPFとBPF

SC(スイッチト･キャパシタ)ブロックで実装

AD変換 : ADCINC ,DA変換 : DACn

スケーラブルなADコンバータ > ADCINC, DAC

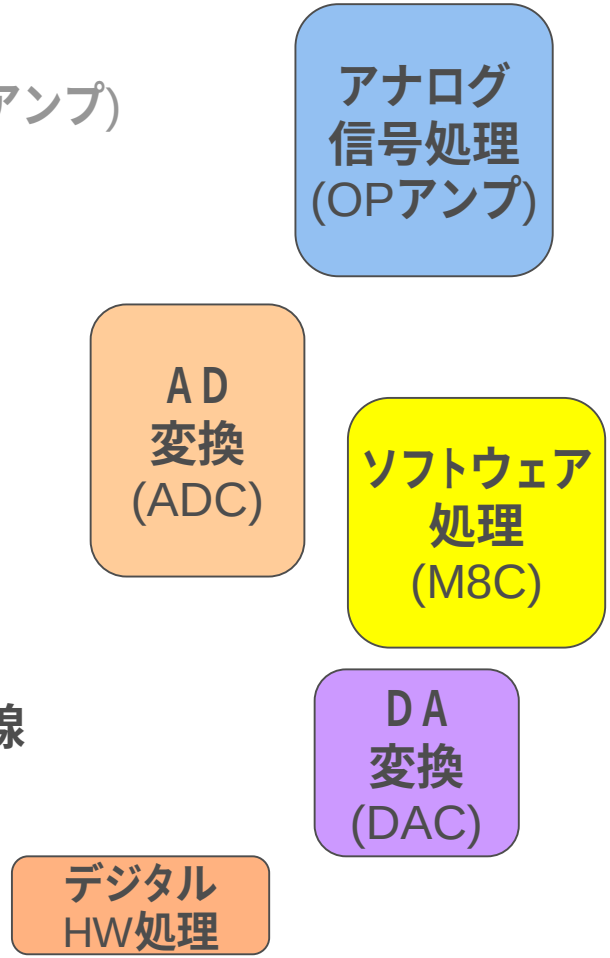
ソフトウェアによる処理 : M8C

外部制御 : **アナログ**,**デジタル**

PWMn, PDM(PRS-Psudo Random Sequence応用)

通信 : **シリアル**, **パラレル**, USB2.00, **赤外線**, 2.4G無線

割り込みコントローラ



信号波形からスペクトラム解析へ

アナログ的に変化する信号は、オシロスコープなどではレベル(Y軸)と時間（X軸）で信号波形として表示されます。

このとき信号が例えばサイン波のように繰り返して現れる（周期関数として表現できる）場合には,この繰り返し時間に帯域窓を設定して、フーリエ変換すると特定周波数のレベルを得ることができます。

しかしフーリエ変換では,窓幅(窓時間)が一定のため時間軸方向の情報は失われてしまいます.

これには窓幅を次々に変えて時間軸方向のデータを連続的に取得して計算すればよいわけです.しかし実際には適正サンプル数とレベルを得るためにはこの窓幅を低い周波数では広く,高い周波数では狭くしていく必要があります。

このような変換を行うにはウェーブレット変換を使用します.これが今日のデジタル信号処理や通信の基礎になっています.非常に面白いですから興味のある方はこれをキーワードに調べてみましょう.

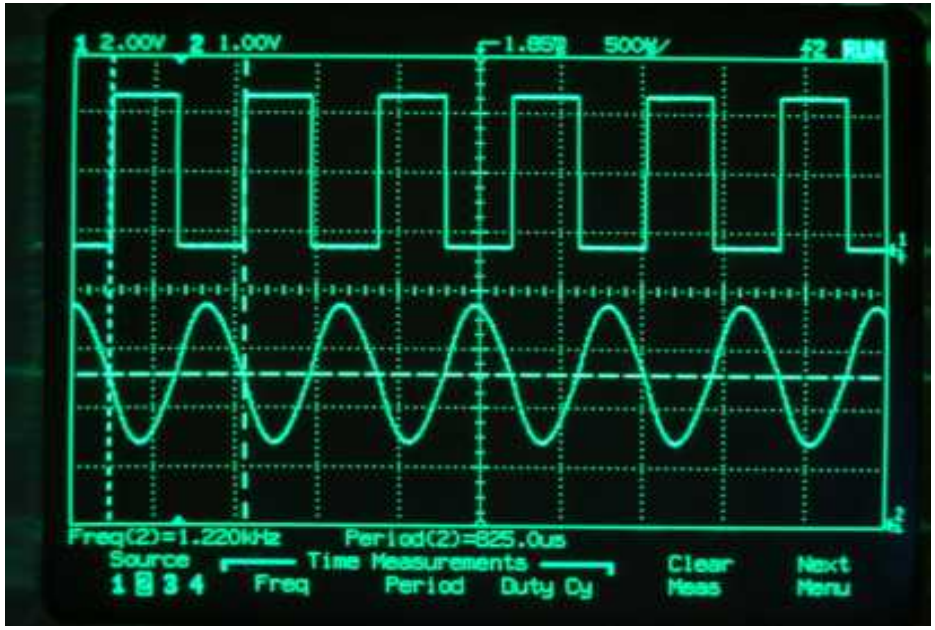
PSoCでもFFTやウェーブレット変換でデジタル信号処理の実験ができます.スペクトラム解析のための基礎となります.

方形波をBPFに通す

方形波はその基本周波数の10倍程度の高調波成分を含んでいます.これは奇数次の周波数のサイン波で合成できます(逆FFT).

よって方形波をBand Path Filterに通すとサイン波が得られます.

FFTを数式レベルで理解していても実際に基底周波数のサイン波がBPFによって現れることを実験で確かめることができます.PSoCにはSwitched Capacitor BlockでLPF,BPFユーザーモジュールを構成します.特性の設定もパラメタライズされています



PWMで作った方形波

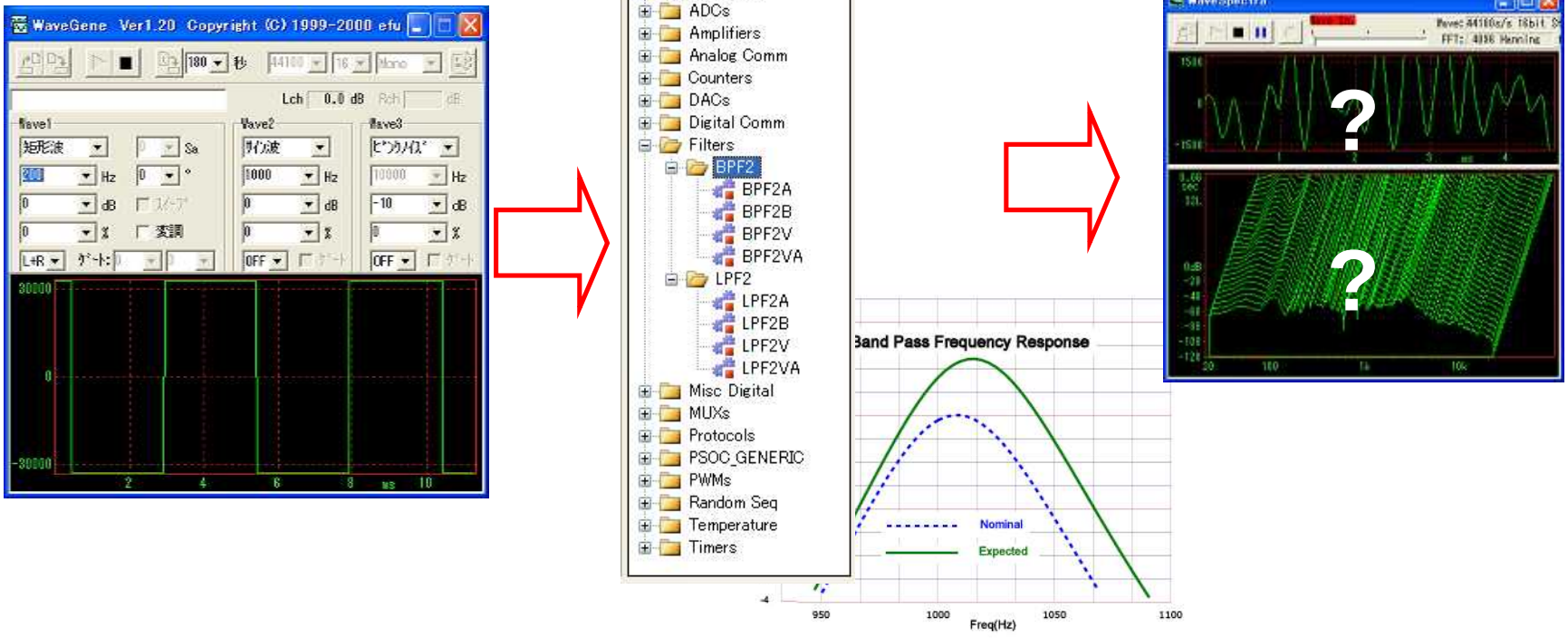
BPFフィルタリングして

現れた基底周波数の

サイン波

方形波から正弦波をとりだす

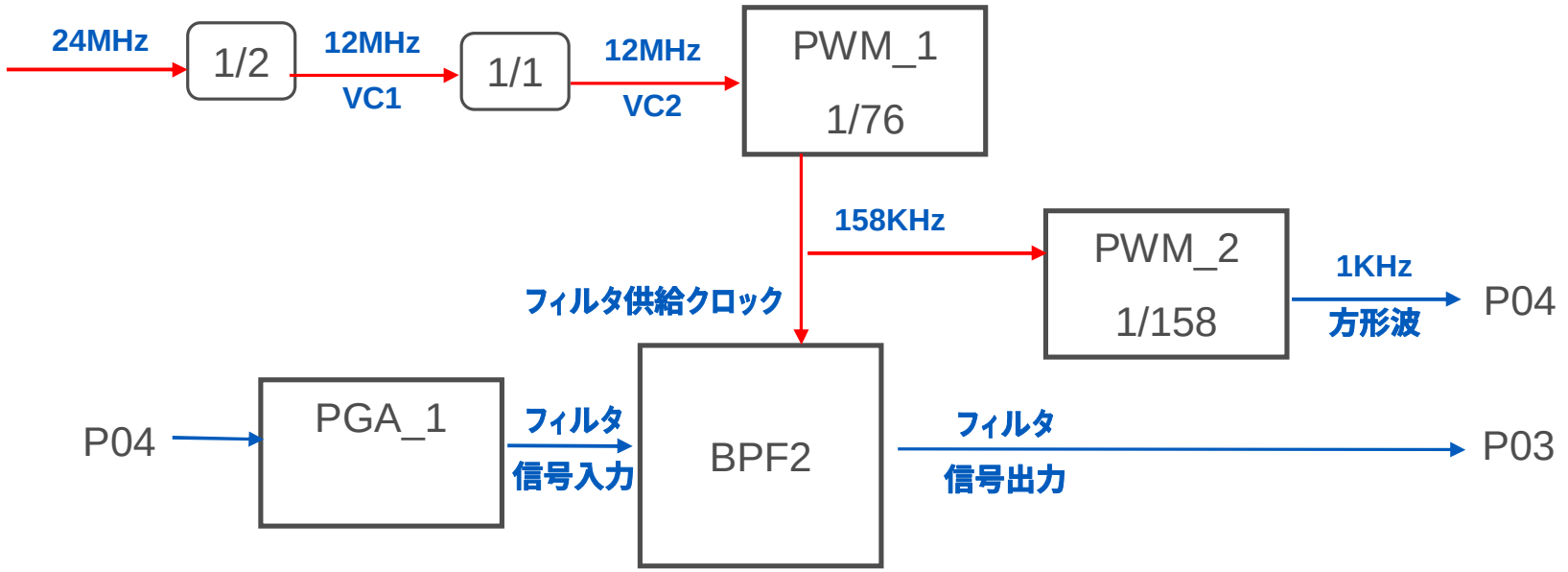
方形波信号をPSoCのバンドパス･フィルターにかける
特定スペクトラムの周波数だけをパスさせる
音を聴いてみる。口笛の音は正弦波に近い



全体の構成

使用するユーザーモジュール

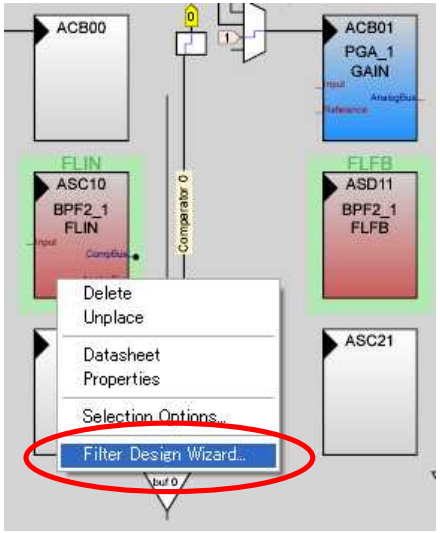
PWM8 x 2, PGA, BPF2

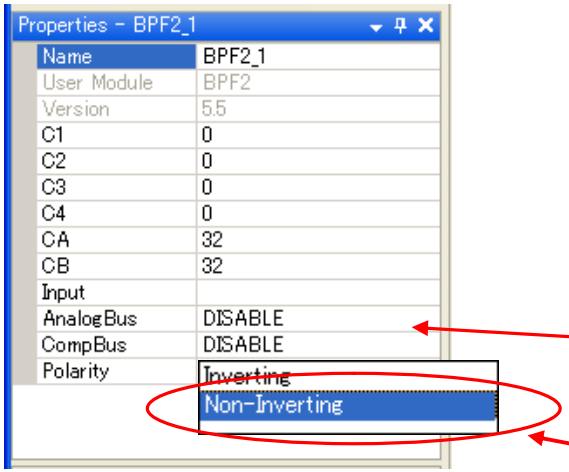


フィルタウィザードを使用してSCの設定値を計算しでBPFに設定します.続いてPWMを使用してSCフィルタ部に与えるクロックを生成します.検証用にPWM_2から1KHzの方形波を生成してPGAでゲインを調整してBPFに入力してサイン波が出るかチェックします.そのあとPWM_1の出力クロックを変更してみます.

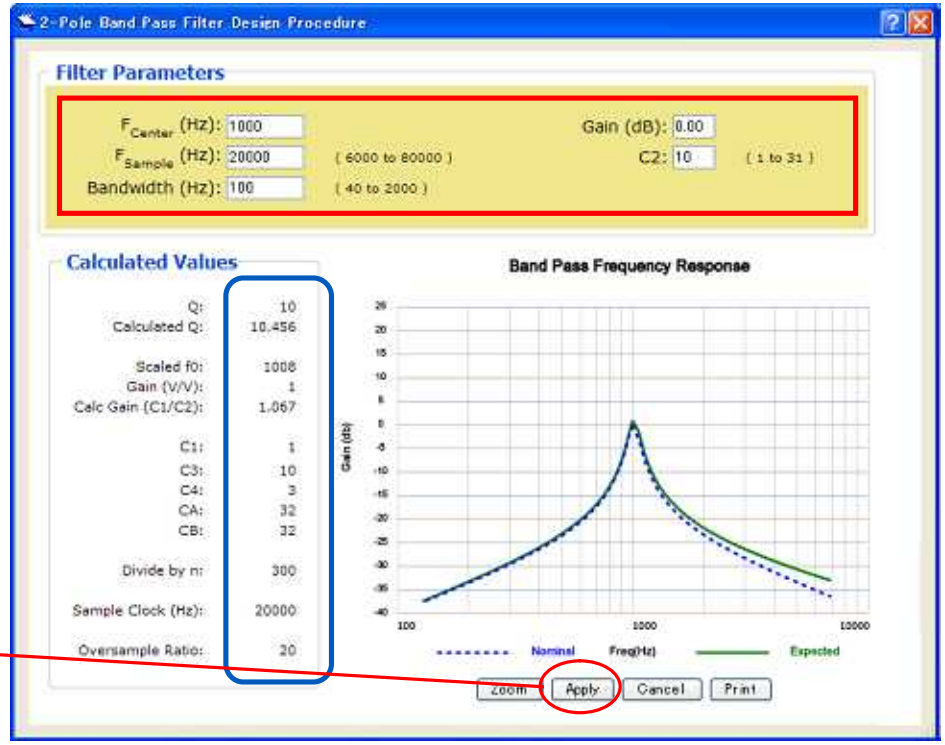
フィルター設計ウィザード

BPF2_1を右クリックしてFilter Design Wizard を起動





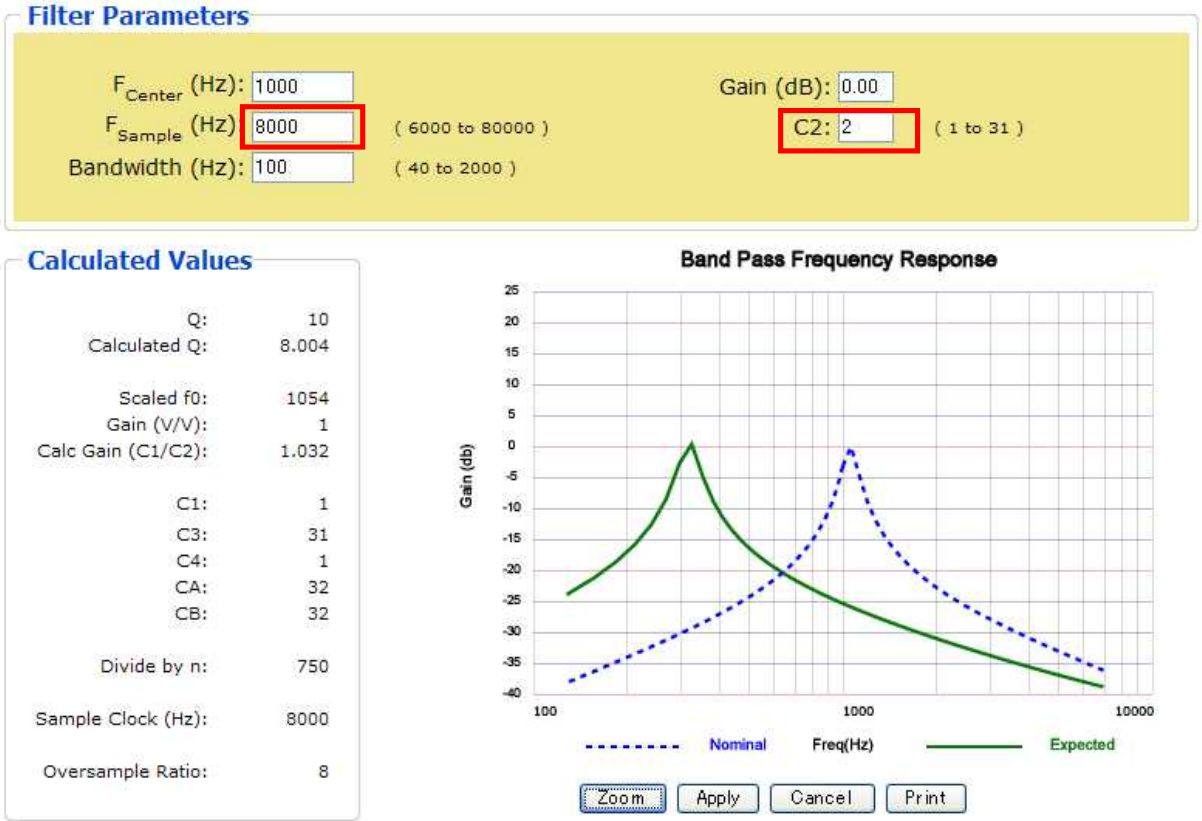
Wizard Windowの**パラメータボックスに設計値を入力すると設定値が自動計算**されます. Applyをクリックするとプロパティーウインドウに値が設定されます.複雑なSCの値計算が自動的に行われます.Zoomをクリックするとグラフが拡大されます.



Polarity はNon-Invertingに設定

設計値を変更してみる

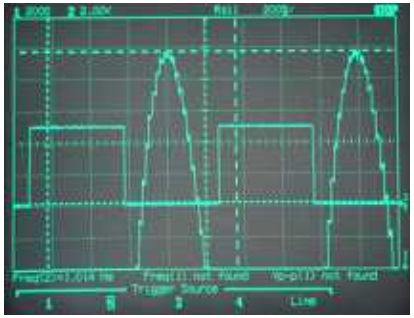
FSample，C2値を変更するとフィルタ特性グラフが変わります
SCの値は自動的に計算されます．



FSample

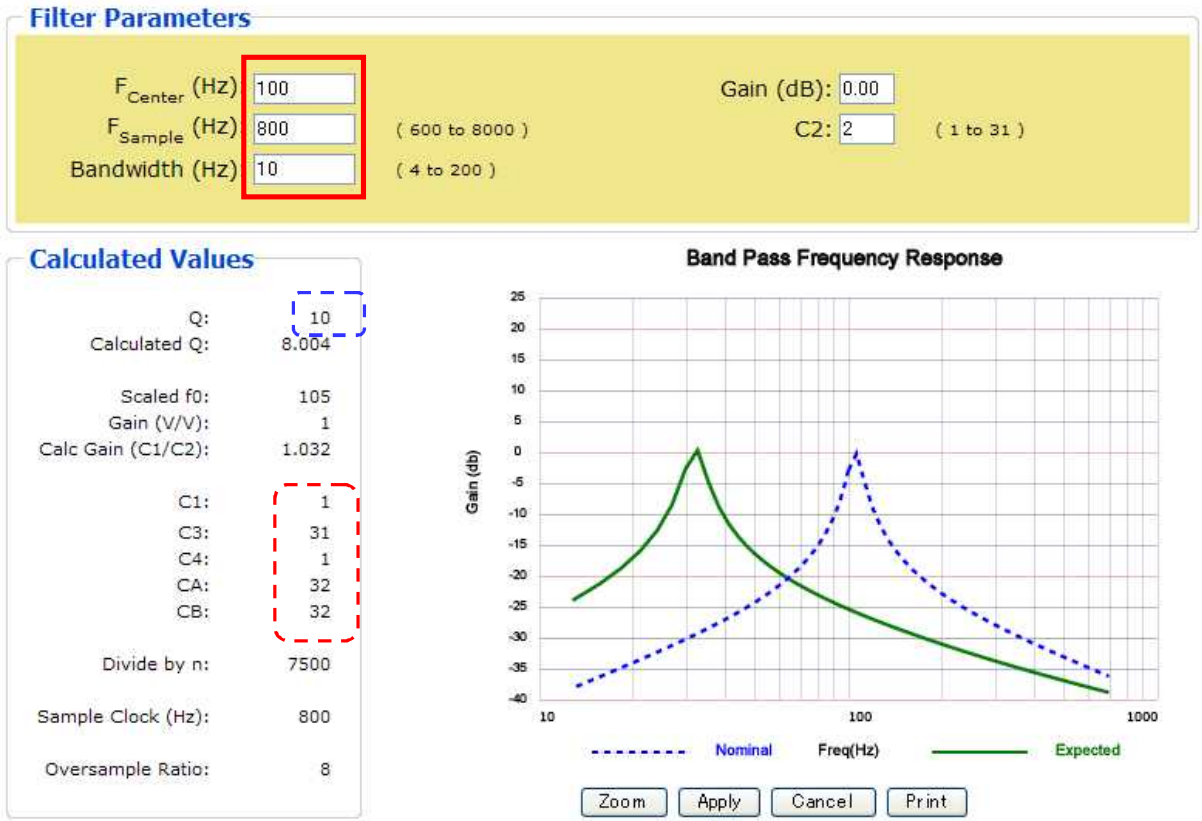
サンプリング周波数を上げるほど，フィルタリング後の波形はスムーズになります．

このオーバーサンプルレートが低いと波形には段々がつきます．



サンプリング周波数の変更の意味

F_{Center}, F_{Sample}, Bandwidth値をすべて1/10にしてみます



SCFでは
Cの定数を
そのままに
しておいて
サンプリング
周波数を変更
するだけで,
フィルタ周波数
を変更すること
ができます.
Qも変わりません

サンプルクロック,中心周波数が1/10になりましたがSCの設定値には変化はありません. このことは何を意味するでしょうか?

SC定数が変わらないということ

サンプルクロック,中心周波数が1/10になりましたがSCの設定値には変化はありません. このことは何を意味するでしょうか?

BPFブロックに与えるクロック,つまりサンプリング周波数を変更するだけでBPFのフィルター周波数を変更できるということになります.BPFのフィルター周波数はPWMで作っていましたね.PWM周波数はプログラムで自由に変更できました.

プログラムで連続的なフィルター周波数変更が可能ということになります.ここからどんな応用が考えられるでしょうか.

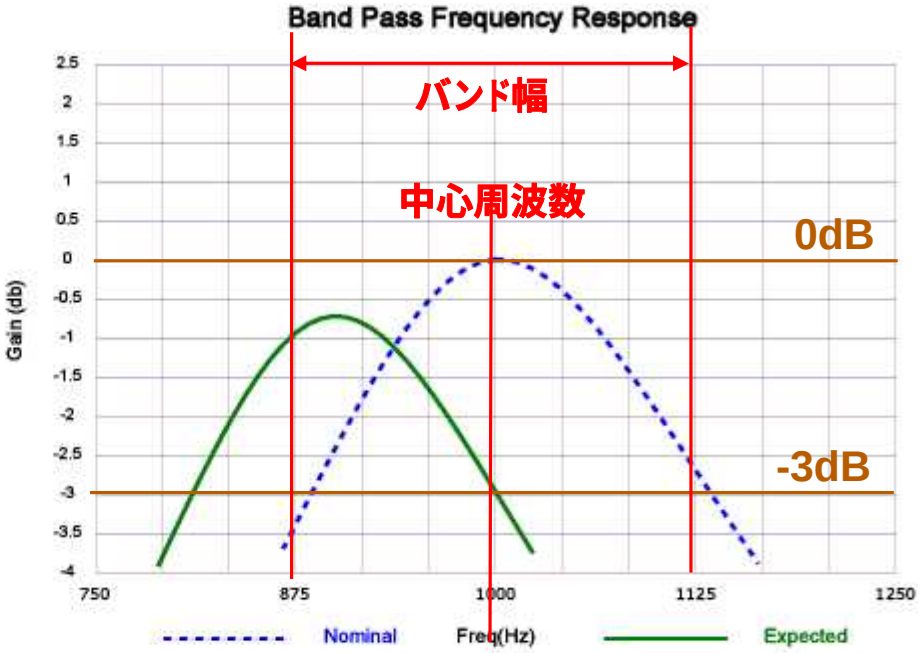
1KHz,BW250Hz,F_{Sample}=40Kで設計します

Filter Parameters

F_{Center} (Hz):	1000		Gain (dB):	0.00	
F_{Sample} (Hz):	40000	(6000 to 80000)	C2:	10	(1 to 16)
Bandwidth (Hz):	250	(40 to 2000)			

Calculated Values

Q:	4
Calculated Q:	3.93
Scaled f0:	1002
Gain (V/V):	1
Calc Gain (C1/C2):	0.889
C1:	1
C3:	2
C4:	18
CA:	32
CB:	32
Divide by n:	150
Sample Clock (Hz):	40000
Oversample Ratio:	40



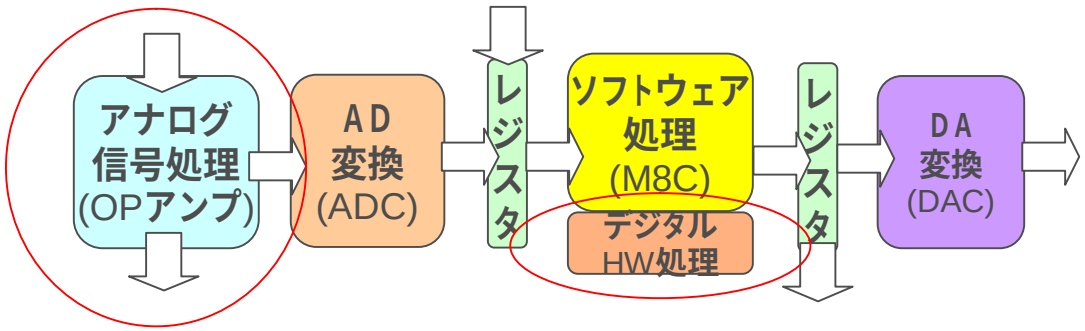
グラフ内の**青の点線**は**計算値**です.
緑の実線は**予測される性能**です.
予測性能と実測値をあとで比較してみます

ここで Q=1000Hz/250Hz=4 となります.BPFモジュールに与えるクロック周波数は,40KHz*4=160KHzとなります.このクロックはPWMで作ります.設定を確認したらAPPLYをクリックしてPWMの設計にすすみます.

任意周波数(例では1 Hz)のサイン波を作ってみます

ラボ
bpf_1hz

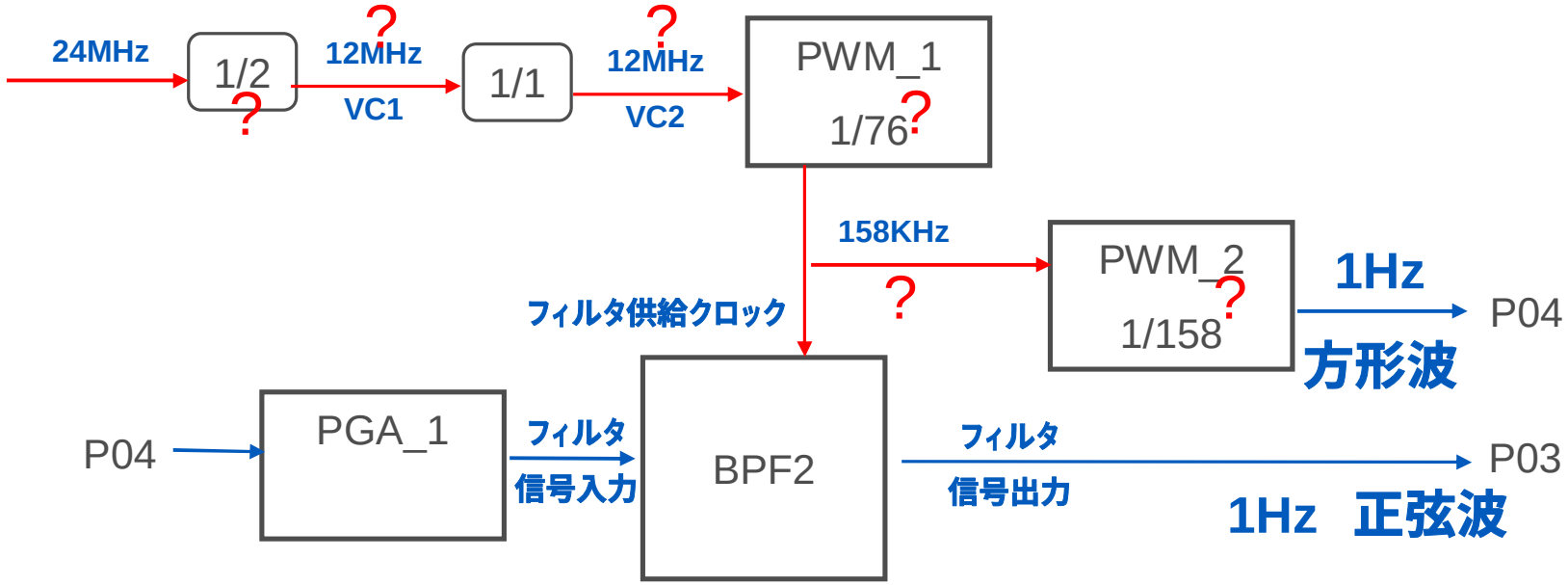
バンドパスフィルタのサンプリング
周波数変更



lab_bpf のフィルタクロックを変更

使用するユーザーモジュール

PWM8 x 2, PGA, BPF2



SCフィルタではBPFフィルタに与えるクロック周波数を変更するだけでフィルタ周波数を自由に変更することができます.

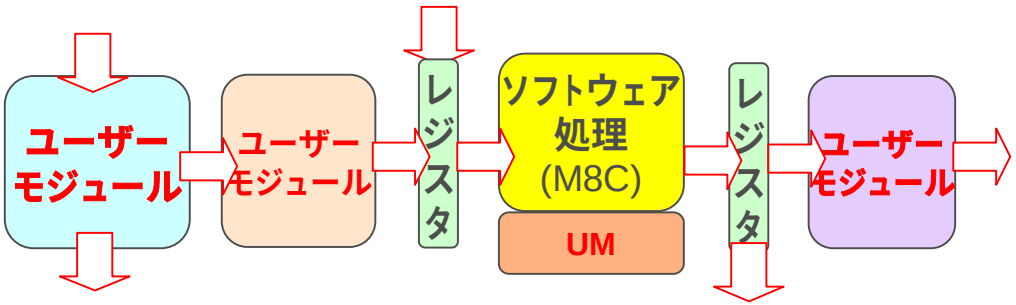
UMのパラメータ変更だけで1HzのBPFフィルタに変更してみてください.

1HzだとP04,P03をLEDにつないで目で結果をみることができます。

自由課題とします.

配置配線について

PSoCのパズル



モジュールの配置と相互の配線

PSoCの配線リソースは多くありません

そのため各ブロックや内部のバス、入出力ピンには,相互に配線可能なところと配線できないところがあります.

この関係を理解して配線になれる必要があります.

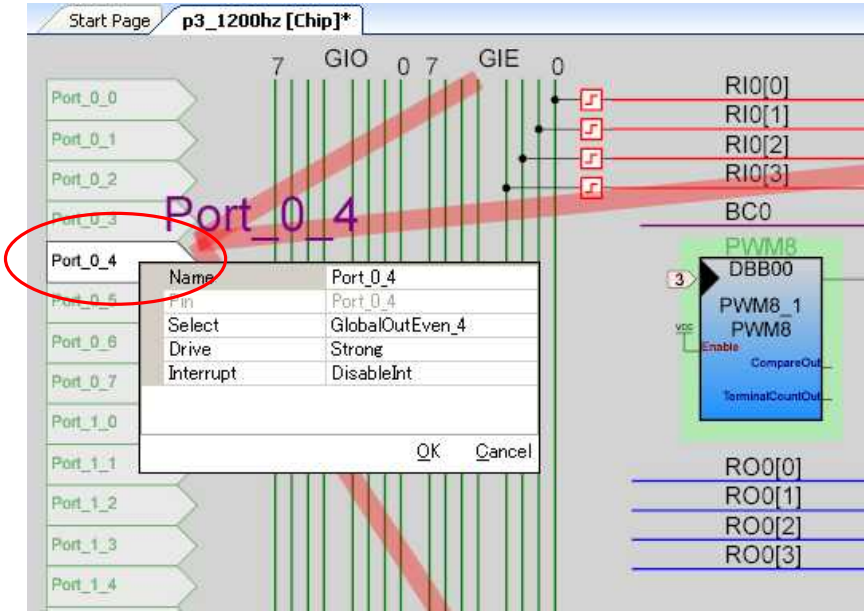
複数のブロックを使用するユーザーモジュールには複数の配置オプションを選択できるようになっており配線できない場合は配置パターンを変更します.

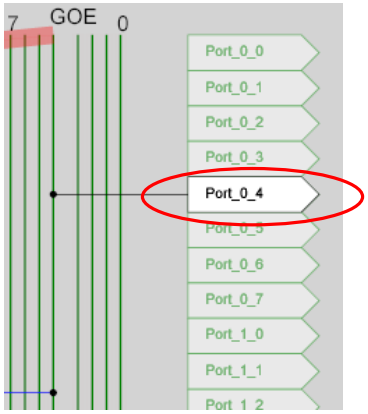
入出力ピンの例

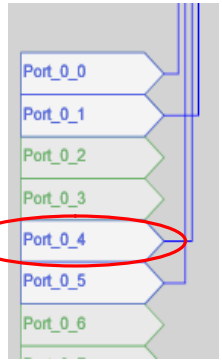
Port番号は上下左右の4ヶ所に表示されています.

上がデジタルブロック用,下がアナログブロック用,左が入力で右が出力です.

外部ピンを利用して内部信号を接続することができます

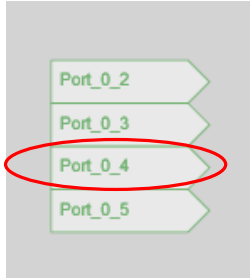






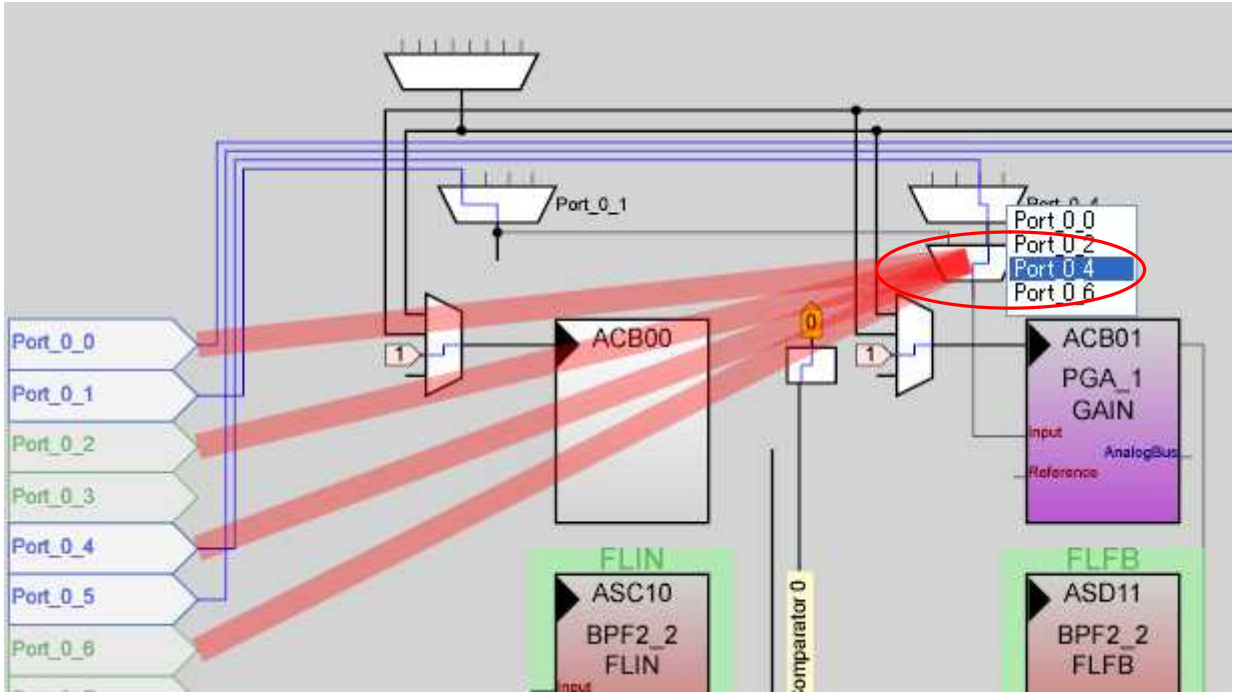
たとえばPort_0_4 を左クリックするとここから接続可能なガイド線が表示されます.

またPort_0_4は上下左右の4ヶ所に表示されており,フレキシブルに使えることがわかります。

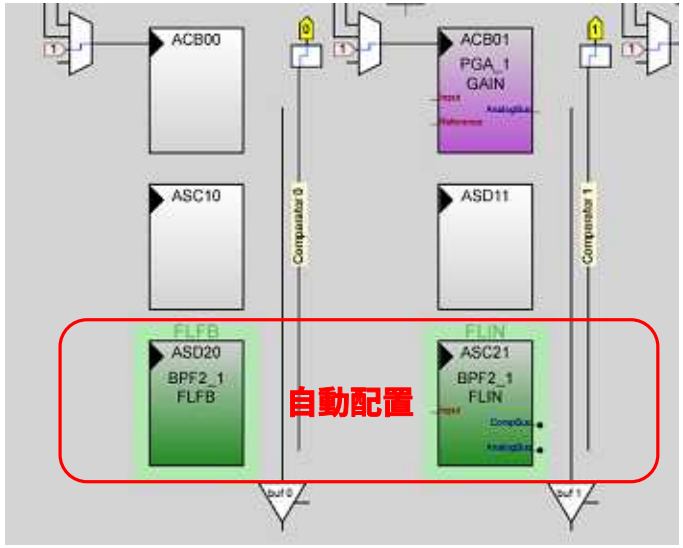


Analog Column Input

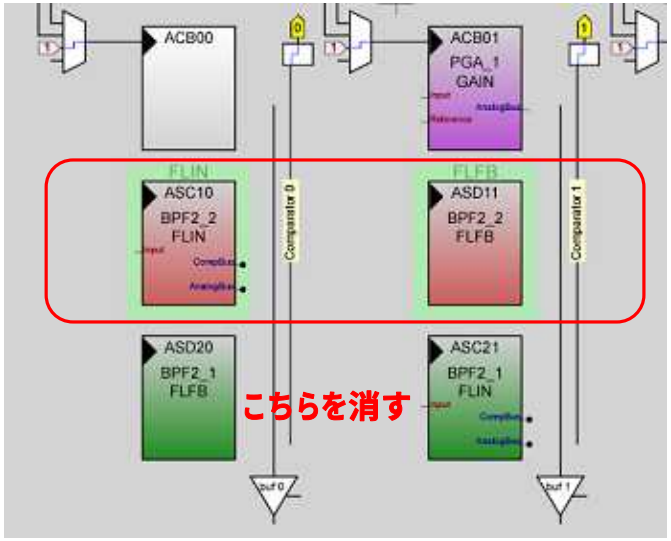
Analog Column Input Muxを右クリックすると入力信号をもってこれるPortが接続ガイドラインで表示されます. Port_0_4から信号を入力したい場合にはPGA_01の配置場所はACB01となります.



ブロックふたつの移動

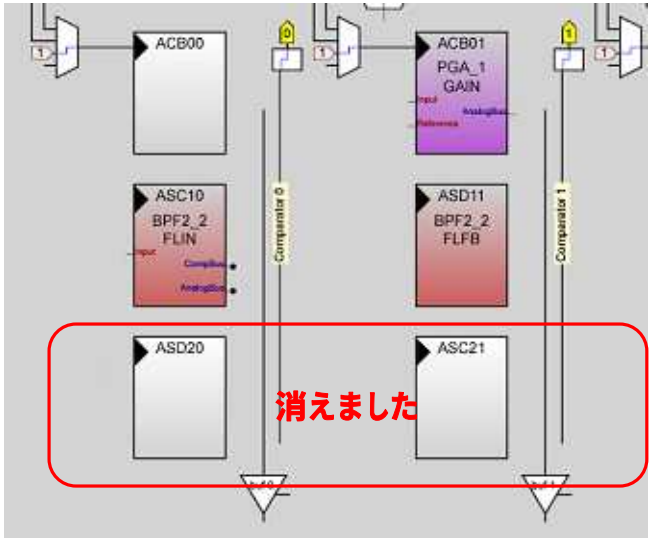


2つのブロックを使うユーザーモジュールを配置した場合中段に配置したくとも下段に配置されるときは,同じモジュールをもう一個追加配置してから下の段を削除する方法があります.



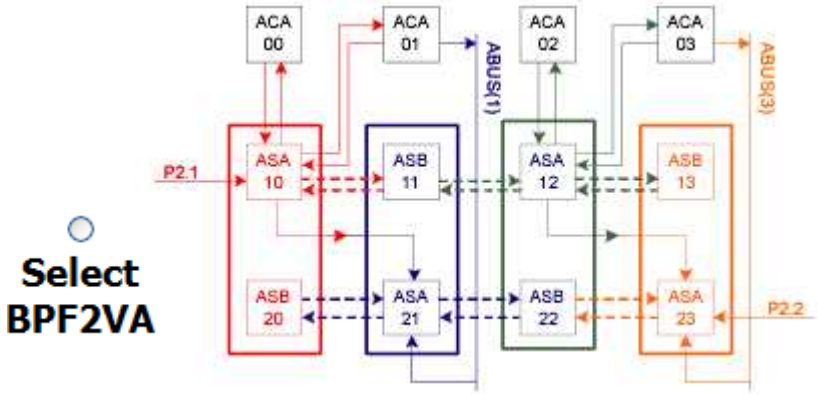
もう1つ

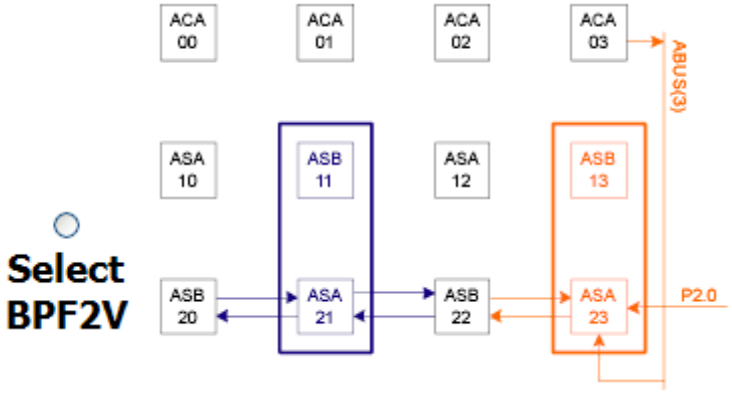
追加配置

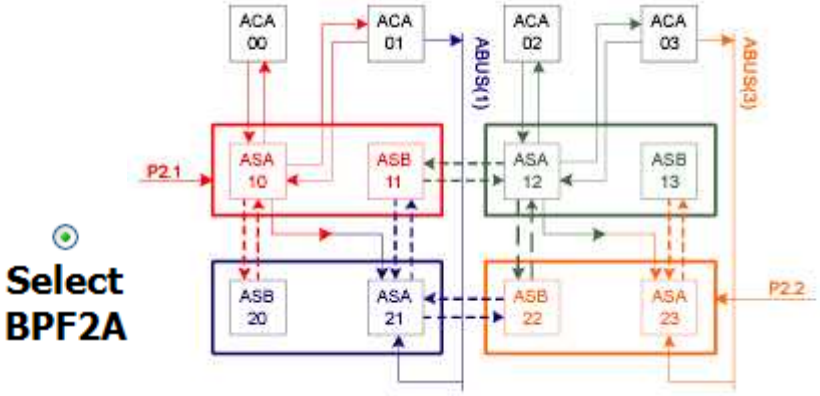


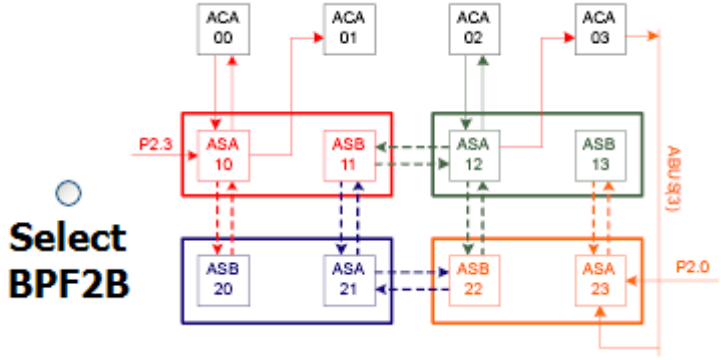
ユーザーモジュールの配置

ブロック2つを使用するBPF2の場合信号の入力先のモジュールや
配線路用にタテ2段とヨコ2段の合計4とおりに配置が選択可能









Memo

フォローアップURL

http://mikamir.web.fc2.com/?/?.htm

?**に入る文字列は、講義中に示します。**

担当講師

ミカミ設計コンサルティング

〒142-0042 **東京都品川区豊町** 2-17-8

三上廉司(みかみれんじ)

Renji_Mikami@nifty.com

http://homepage3.nifty.com/western/mikamiconsult.htm

電話 080-5422-2503(au)