GPIO解説資料

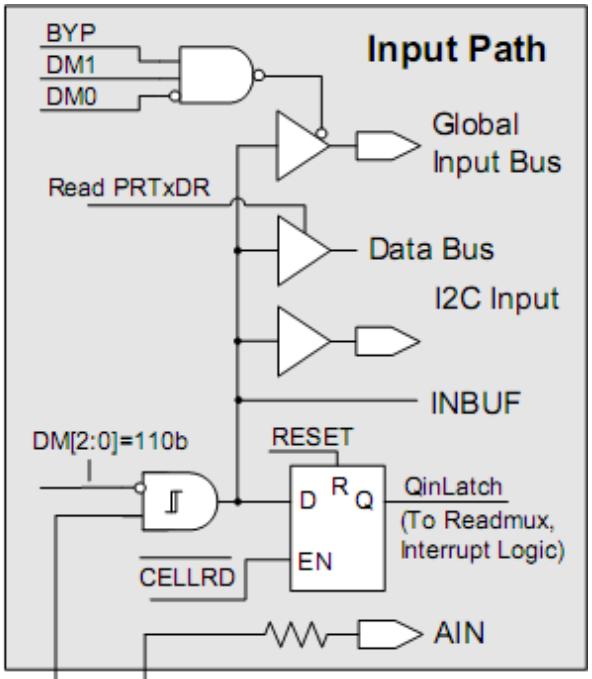
GPIO
PSoC Experiment
Course Material 7

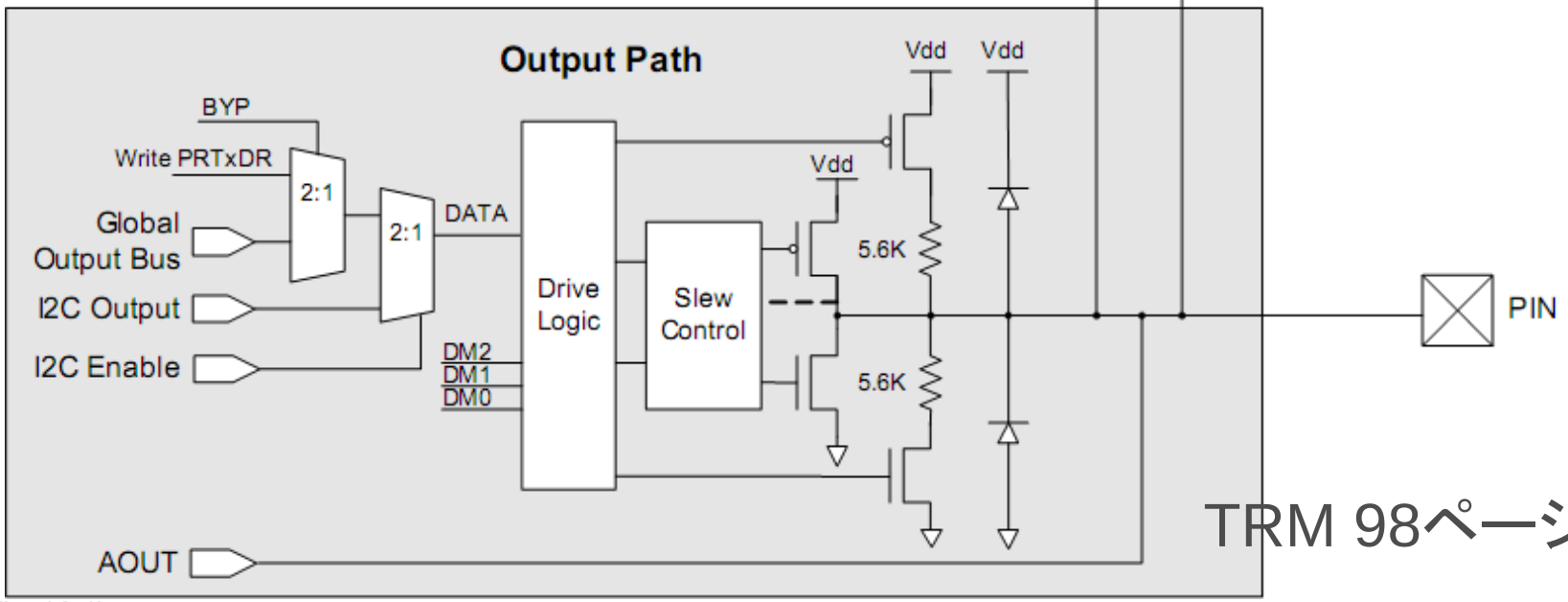
CY3210/3214評価基板ベース

GPIO Lab Material 7 V2.03
April 9th., 2019
gpio_lab_7.pptx (29 Slides)
Renji Mikami

GPIOの実際の構造

特徴
- 全てのポートにおいてデジタル入出力が可能
- アナログ入出力に関しては, ポートに依る





TRM 98ページ

GPIO関連レジスタ

ユーザーが設定する必要のあるレジスタ

PRTxDR：ポートのデータライト・リードレジスタ

PSoC Designerで設定可能なレジスタ

PRTxDM2/DM1/DM0：動作モード設定

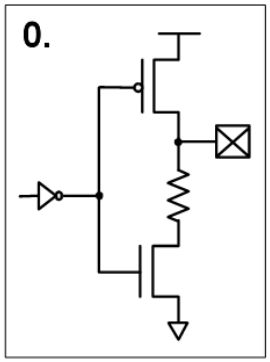
PRTxIE　：割り込み発生許可・不許可レジスタ

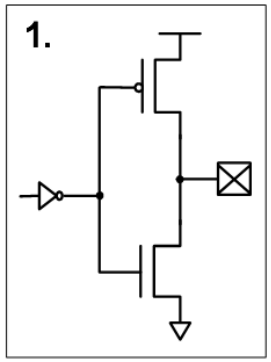
PRTxIC1/IC0：割り込みモード設定レジスタ

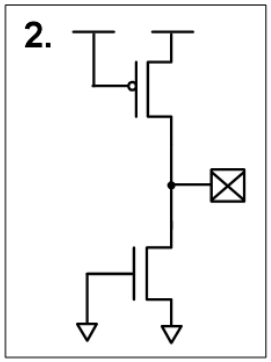
PRTxGS：ポートをCPU・デジタルブロックで使うかを選択するレジスタ

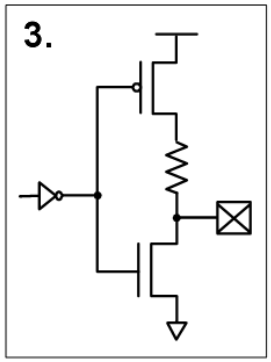
＊xにはポート番号（0,1,2,3…)を入れる

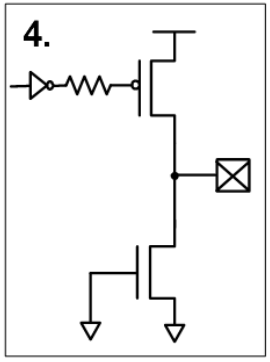
GPIOのDrive Mode(TRM 98ページ)

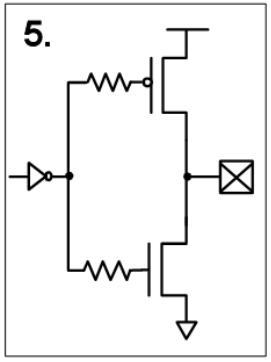


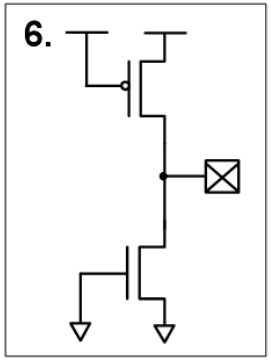


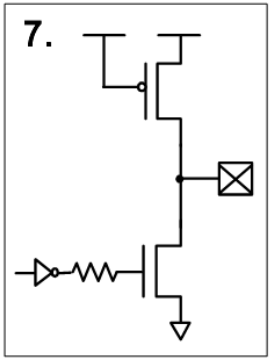












番号	説明
0	プルダウン
1	ストロング (出力用)
2	ハイインピーダンス
3	プルアップ
4	オープンエミッタ
5	Slowストロング (出力用)
6	ハイインピーダンスアナログ
7	オープンコレクタ

GPIOの使い方

読み込み

```
unsigned char prtData,pinData;
prtData = PRT1DR;                    // Port1の状態を取得
pinData = PRT1DR & 0b00000001; // P1.0の状態を取得
pinData = PRT1DR & 0b00000010; // P1.1の状態を取得
```

書き込み

```
PRT1DR |= 0x01;                      // Port1.0をHighにセット
PRT1DR |= 0x02;                      // Port1.1をHighにセット
PRT1DR &= 0x11111011;                // 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。ポート毎の読み書きしか出来ません

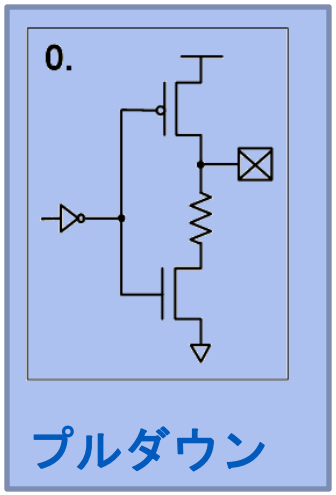
入力値は、実際の外部ピンの状態を取得します。

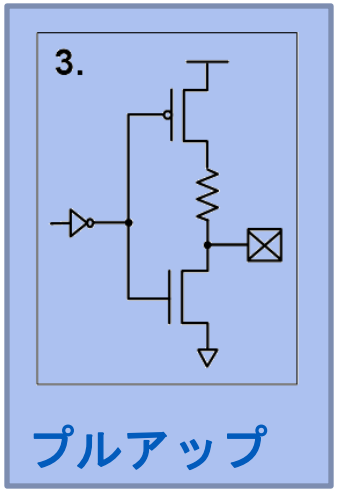
出力値は、ドライブロジックに対して出力します。

GPIOのプルアップ、プルダウン時

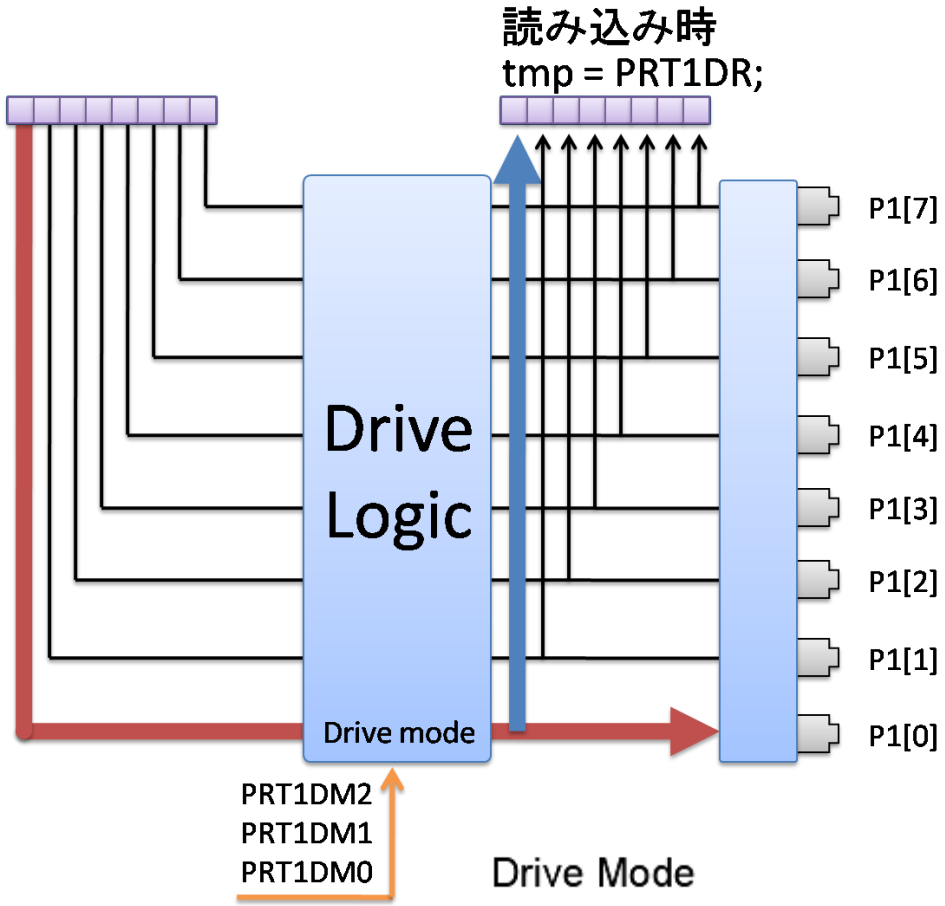
Drive Modes

DM2	DM1	DM0	Drive Mode	Diagram Number	Data = 0	Data = 1
0	0	0	Resistive Pull Down	0	Resistive	Strong
0	0	1	Strong Drive	1	Strong	Strong
0	1	0	High Impedance	2	Hi-Z	Hi-Z
0	1	1	Resistive Pull Up	3	Strong	Resistive
1	0	0	Open Drain, Drives High	4	Hi-Z	Strong (Slow)
1	0	1	Slow Strong Drive	5	Strong (Slow)	Strong (Slow)
1	1	0	High Impedance Analog	6	Hi-Z	Hi-Z
1	1	1	Open Drain, Drives Low	7	Strong (Slow)	Hi-Z





GPIO入力時の動作



DriveLogicによって、
プルアップ(ダウン)等のGPIOの設定がなされ、外部ピンの実際の状態と、
DriveLogicによって、読み込む値が決定されます。

Drive Mode	Diagram Number	Data = 0	Data = 1
Resistive Pull Down	0	Resistive	Strong
Resistive Pull Up	3	Strong	Resistive
High Impedance	2	Hi-Z	Hi-Z

GPIO入力時の動作(Hi-Z)

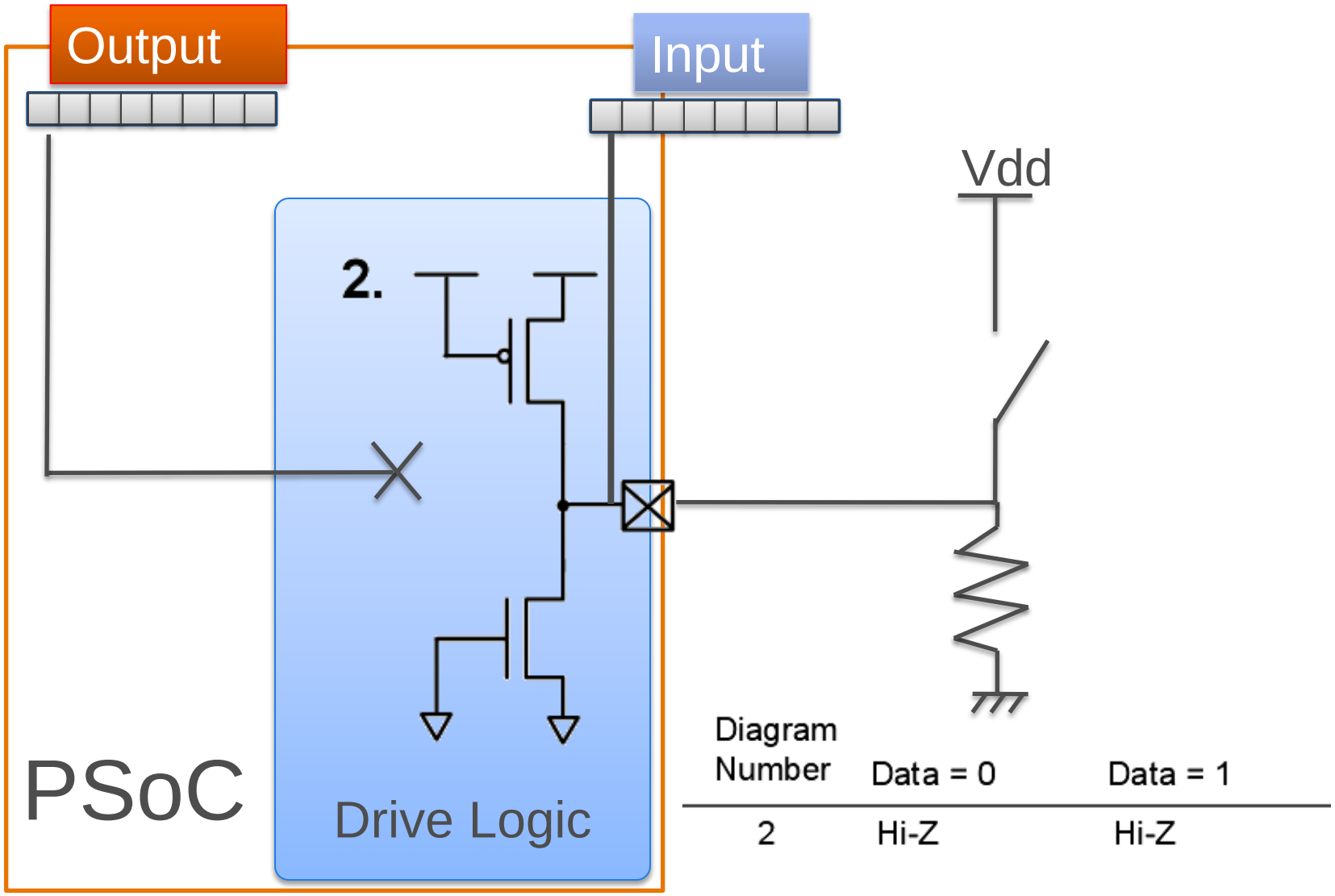


Diagram Number	Data = 0	Data = 1
2	Hi-Z	Hi-Z

GPIO入力時の動作(プルダウン)

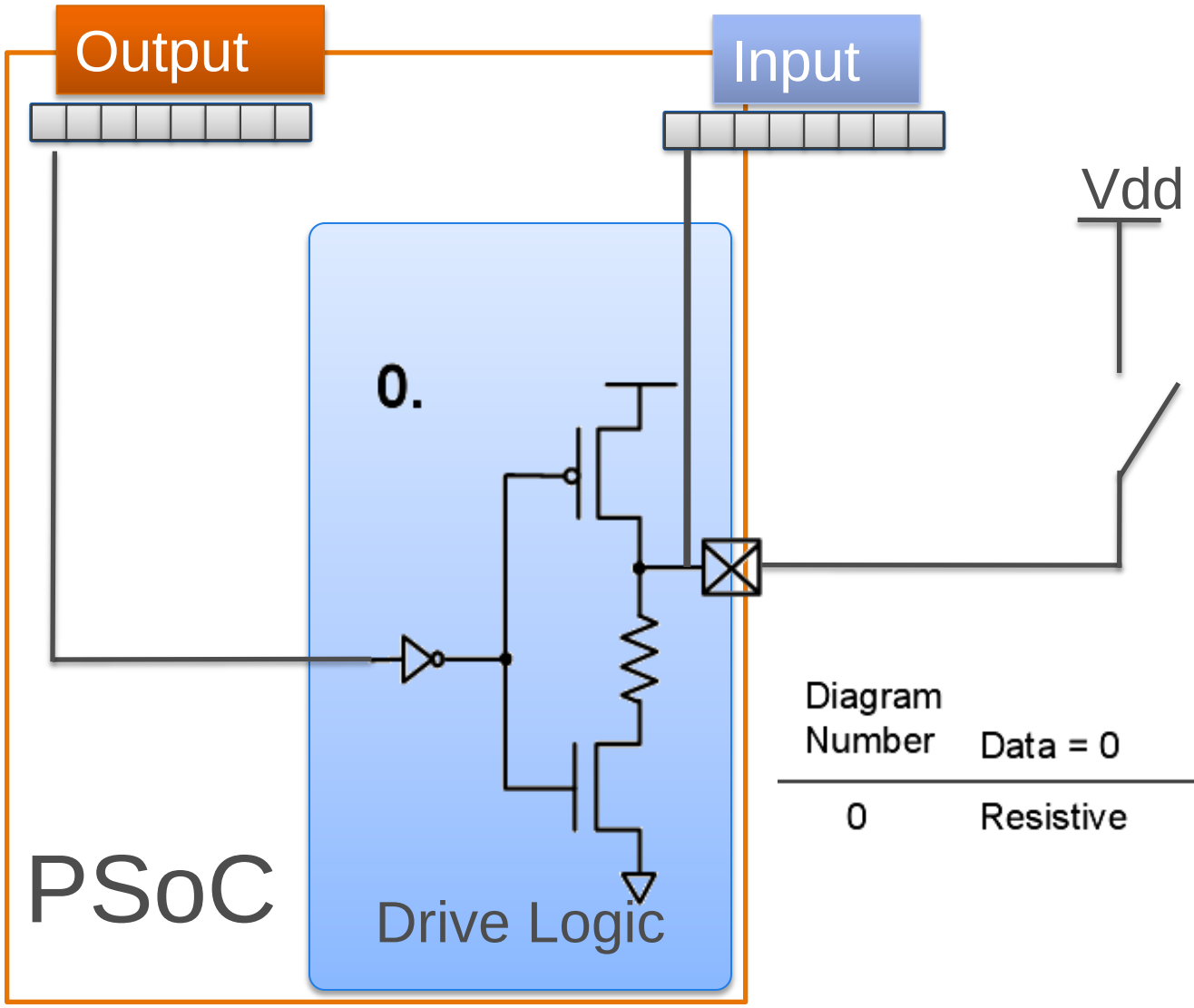


Diagram Number	Data = 0	Data = 1
0	Resistive	Strong

GPIO入力時の動作(プルダウン Low出力時)

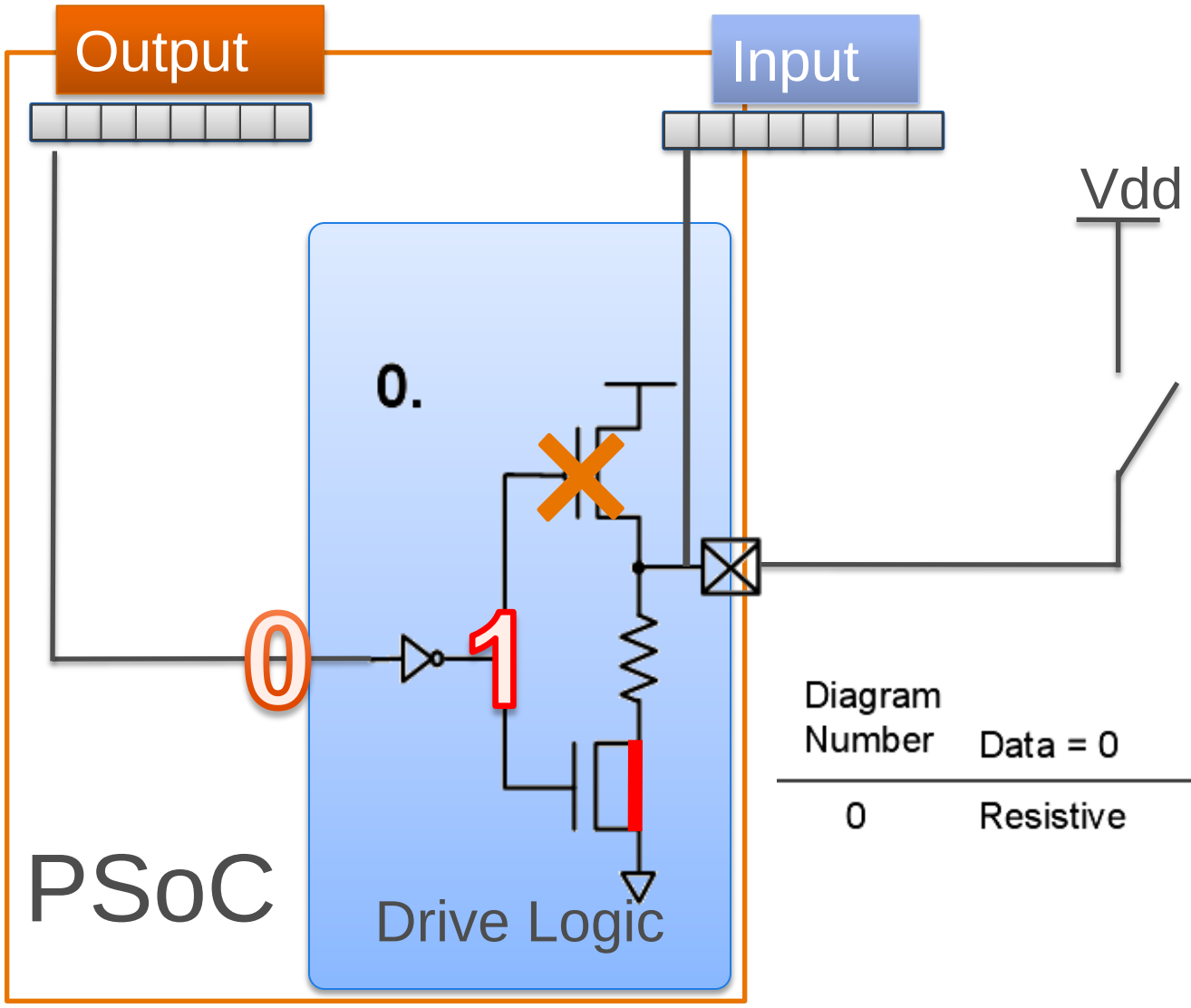


Diagram Number	Data = 0	Data = 1
0	Resistive	Strong

GPIO入力時の動作(プルダウン High出力時)

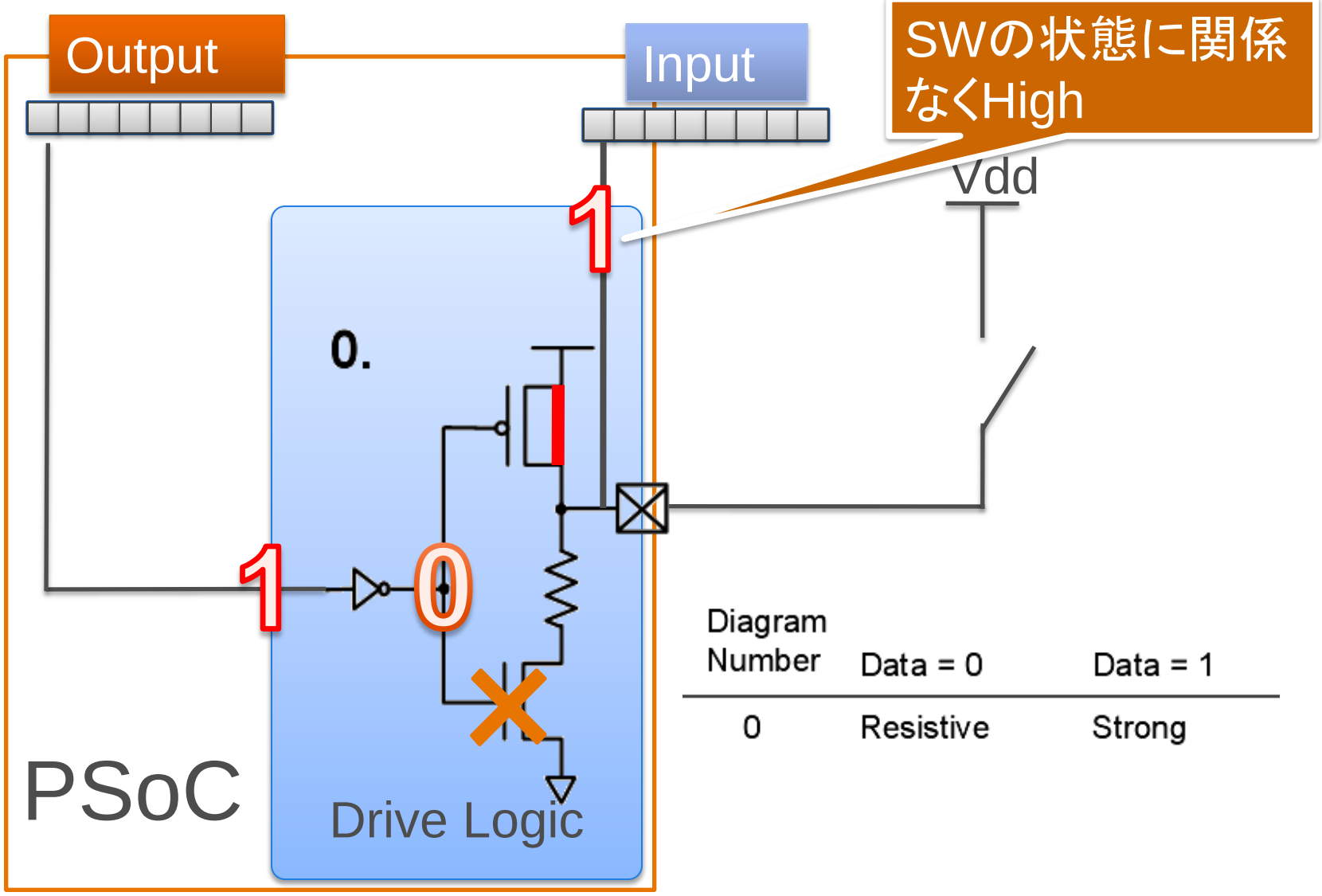


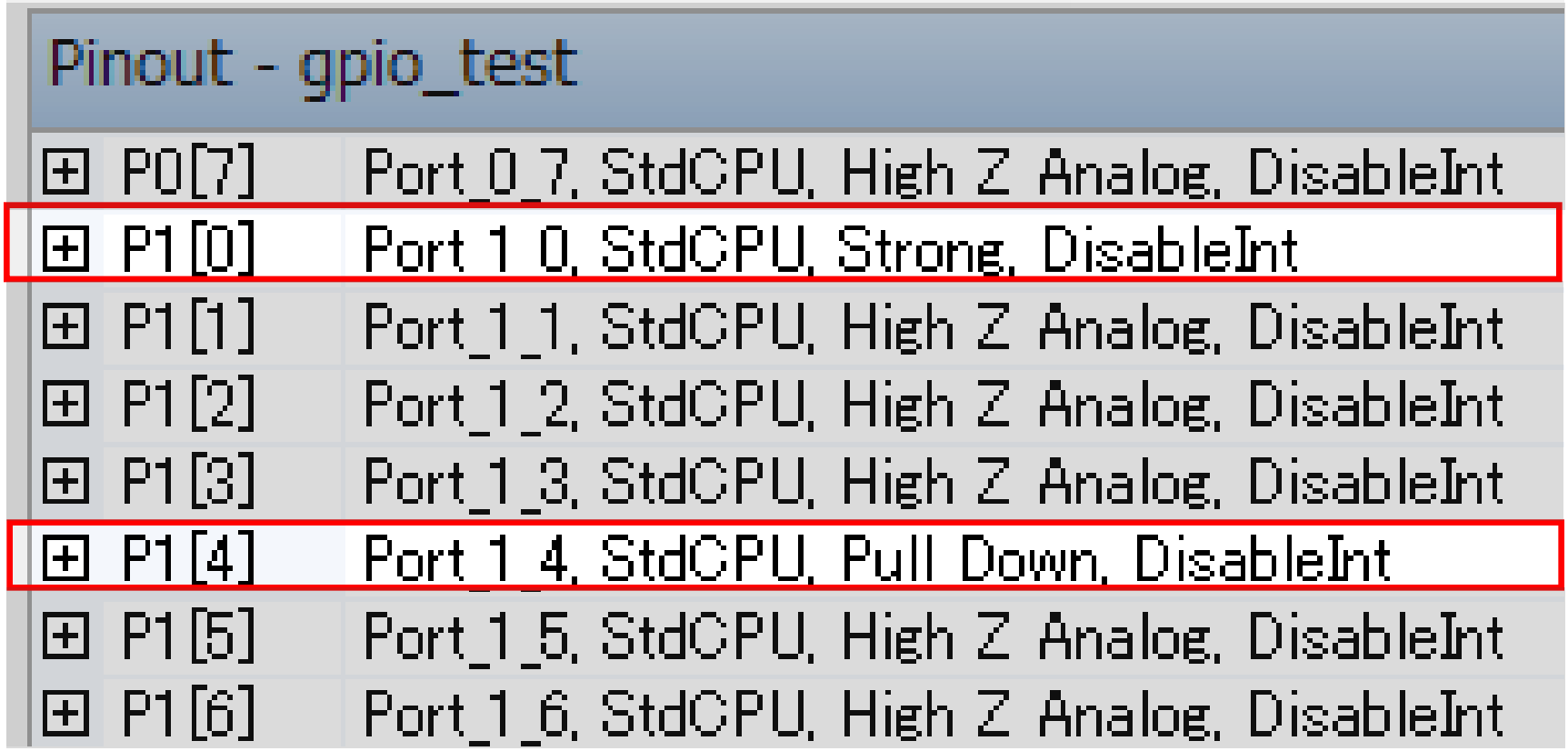
Diagram Number	Data = 0	Data = 1
0	Resistive	Strong

プロジェクト内容

新規プロジェクト作成
プロジェクト名称はgpio_test
使用デバイスはCY8C24894-24LFXI(CY3214基板)
使用デバイスはCY8C29446-24LPXI(CY3210基板)
またはCY8C27443-24LPXI(CY3210基板)

スイッチの状態をLEDに表示する
　＊．スイッチを押すとLEDが点灯
　　ポーリング方式（ループ内で順次判断）
　　割り込み方式(立ち上がり割り込み)

GPIOの設定



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

Global Resource

Global Resources - gpio_test	
Power Setting [Vcc / Sys	5.0V / 24MHz
CPU_Clock	SysClk/8
Sleep_Timer	512_Hz
VC1= SysClk/N	1
VC2= VC1/N	1
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Intern
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
Trip Voltage [LVD]	4.81V
LVDThrottleBack	Disable
Watchdog Enable	Disable

デフォルトの設定でOK

ここで Generate/Build

main.cの編集の前に

今回用いた基板では、SWはGND側に接続され、
LEDは吸い込みで点灯します。

SWが押されたかどうかは以下のコードで判別できます。

```
#define SW 0b00010000 //4番ピン
if(PRT1DR & SW){
    //押されたときの動作
} else {
    //押されていないときの動作
}
```

C言語　ビット演算子補足
| : OR
& : AND
~ : NOT

LEDは以下のコードで制御できます。

```
#define LED 0b00000001//0番ピン
PRT1DR &= (~LED);     //LED点灯
PRT1DR |= LED; //LED消灯
```

main.cの編集(ポーリング方式)

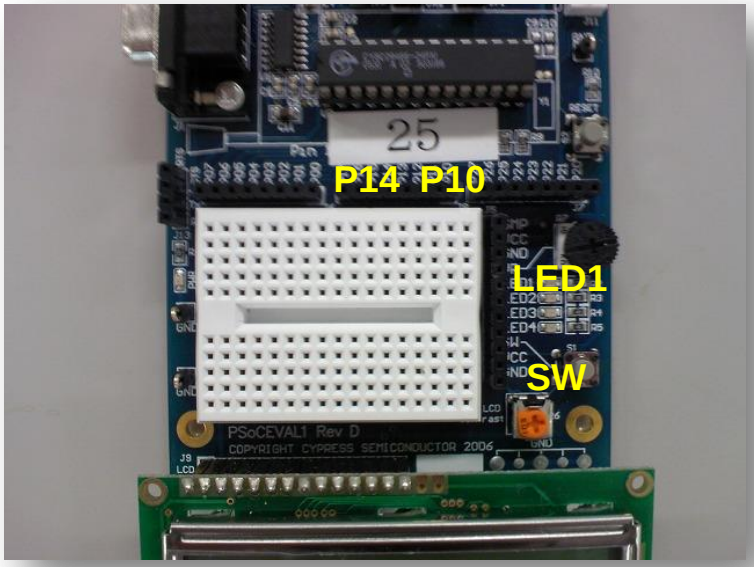
```
#define SW 0b00010000
#define LED   0b00000001

void main()
{
   PRT1DR &= (~SW);            //    SW プルダウン
   while(1){
    if( PRT1DR & SW ){         //SWが押されている時
        PRT1DR &= (~LED); //    LED点灯
        PRT1DR &= (~SW);       //    SW再プルダウン
    } else {                   //SWが押されていないとき
        PRT1DR |= LED;     //    LED消灯
        PRT1DR &= (~SW);       //    SW再プルダウン(予備)
    }
   }
}
```

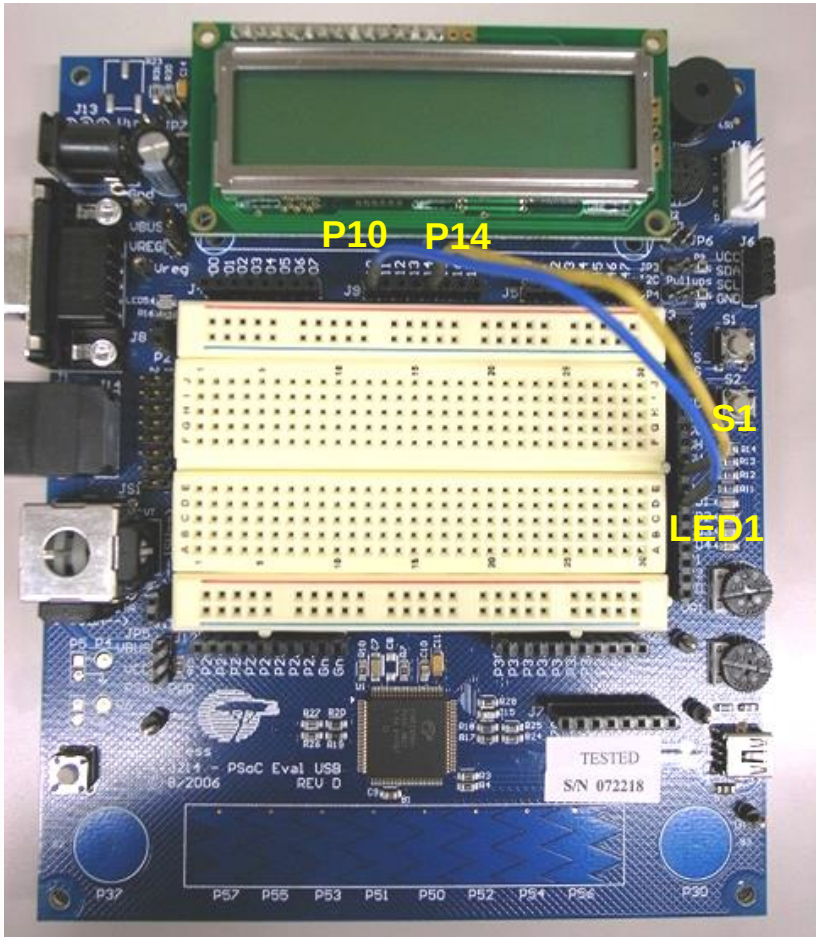
C言語　ビット演算子補足
| : OR
& : AND
~ : NOT

配線を行って下さい

P10 → LED1
P14 → S1(CY3214)
P14 → SW(CY3210)
SWを押すとLEDが
点灯します

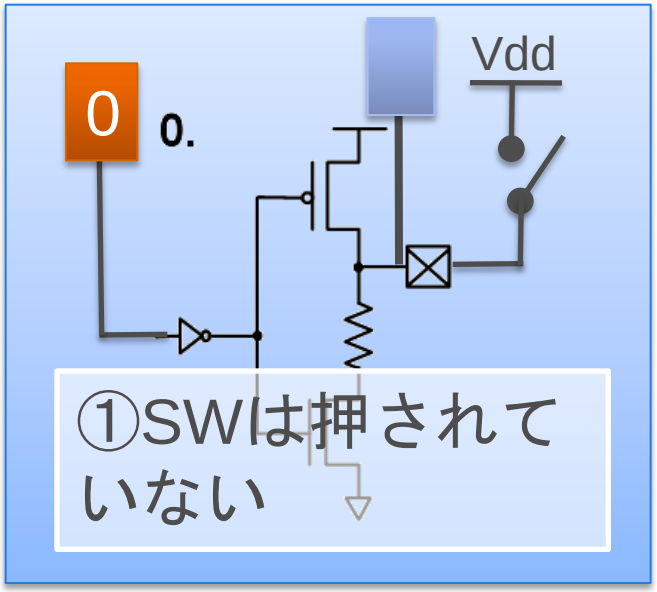


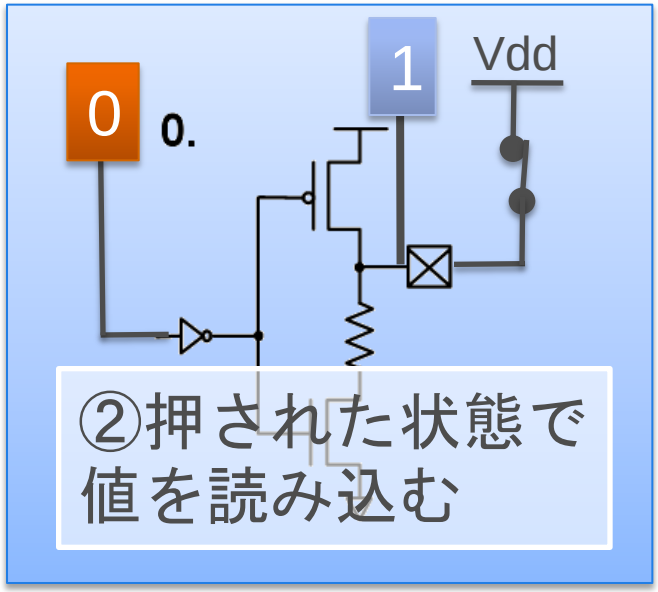
CY3210 PSoCEval1

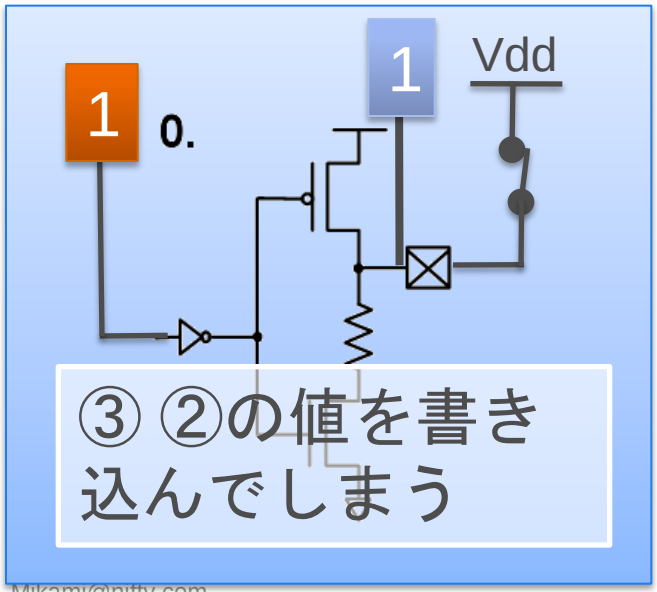


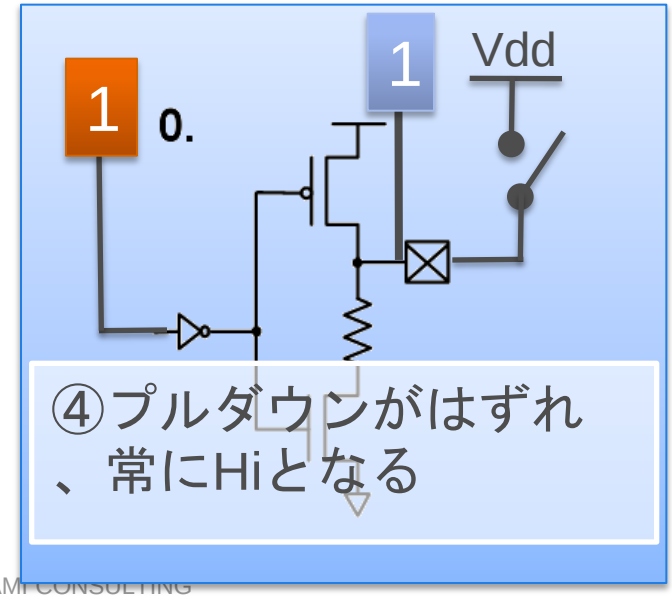
CY3214 PSoCEvalUSB

プルダウンを複数回行う理由 その1









プルダウンを複数回行う理由 その2

Before

操作	実際の状態	出力設定	読み込む
SWを押す	1	0	1
SWを押さない	0	0	0
If(PRT1DR & SW){	1	0	1
Temp = PRT1DR ;	1	0	1
Temp &= (~LED);			
PRT1DR = Temp		1	
SWを押す	1	1	1
SWを押さない	0	1	1

After

ここで
プルダウンが外れる

すると
Lowに落ちなくなる

PRT1DR &= (~LED); // LED点灯

プルダウン(アップ)が外れる時の対策

入力を行うポートと出力を行うポートをわける(Hi-Z , Hi-Z Analog 以外の入出力について)

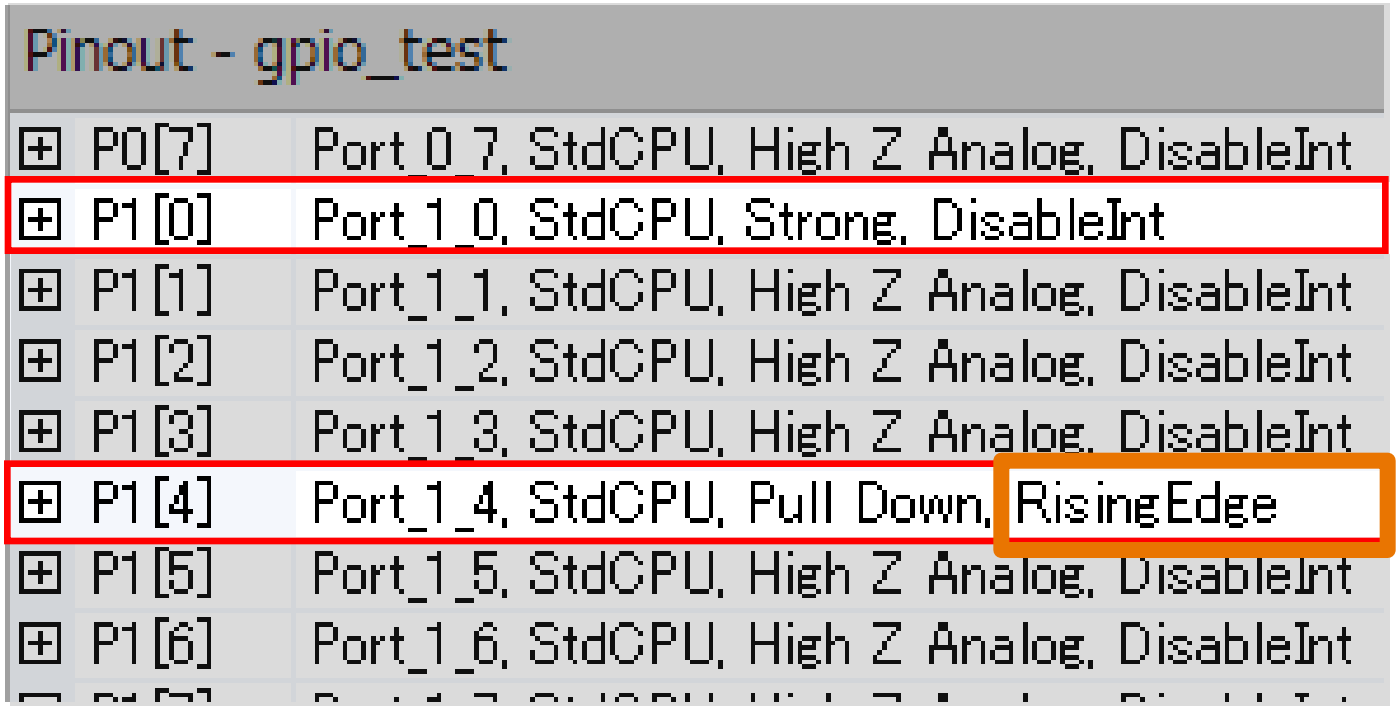
ポートに書き込む度に、ポートを再度プルアップ・ダウンする(今回の方法)

ポートを読み込む前に、毎回ポートのプルアップ・ダウンをする(ポーリング時のみ)

外部プルアップ・ダウンを行う

出力にはLEDモジュールを使う

再度, GPIOの設定



・P1[0] LED用　　Output
・P1[4] Switch用　Input　　**立ち上がり割り込み**

設定が終了しましたらGererate / Build

GPIO割り込みの記述方法

割り込み処理をアセンブリで記述する場合
１．Library Source >>PSoCGPIOINT.asm にアセンブリ記述

割り込み処理をCで行う場合
１．　PSoCGPIOINT.asmに　“ljmp　_myISR”　を記述
　　バナーの内側に書くこと，名前（myISRの部分）は任意
　　関数名の前にアンダースコアを忘れないこと
２．#pragma interrupt_handler myISR　をmain.cに記述
３．void myISR(){ } に処理をコード記述
４．GPIO割り込みを許可するコードをmain.cに記述

Application Editor−psocgpioint.asm

Library Source Files
>>PSoCGPIOINT.asm

PSoCGPIOINT.asm
ファイル全体

PSoC_GPIO_ISR以下のユーザーカスタム領域に
main.c割り込み処理を記述する 関数を定義する。
今回は　void gpio_isr(void)　を使うので、
ljmp _gpio_isr と記述する

```
52  PSoC_GPIO_ISR:
53
54
55      ;@PSoC_UserCode_BODY@ (Do not change th
56      ;---------------------------------------
57      ; Insert your custom code below this ba
58      ;---------------------------------------
59
60       ljmp _gpio_isr        記述
61
62      ;---------------------------------------
63      ; Insert your custom code above this ba
64      ;---------------------------------------
65      ;@PSoC_UserCode_END@ (Do not change thi
66
67      reti
```

ユーザー
カスタム領域

main.c の編集

```
void main()
{
  M8C_EnableGInt;

  M8C_EnableIntMask(INT_MSK0,INT_MSK0_GPIO);

  PRT1DR &= (~SW)

  while(1){};
}
```

M8Cの割り込みを許可

GPIO割り込みを許可

Application Editor−main.c

```
#include <m8c.h>
#include "PSoCAPI.h“

#define SW      0b00010000
#define LED     0b00000001

void main()
{
  M8C_EnableGInt;
  M8C_EnableIntMask(INT_MSK0,
  INT_MSK0_GPIO);

  PRT1DR &= (~SW);
}
```

```
#pragma interrupt_handler gpio_isr

void gpio_isr(void)
{
  static int flag = 0;

  if( flag ){
    PRT1DR &= (~LED);
    PRT1DR &= (~SW);
    flag = 0;
  } else {
    PRT1DR |= LED;
    PRT1DR &= (~SW);
    flag = 1;
  }
}
```

SWを押すとLEDが反転します

補足事項

C言語て定義した変数をアセンブリでアクセスするには

例 :

```
unsigned char flag=0;  //グローバル変数を定義
mov [_flag], 0x0A
```

インラインアセンブリの使い方

```
asm(“nop”);  // NOPをCコードに挿入
asm(“mov A, 0xFF”);  // Aレジスタに0XFF
```

Memo

フォローアップURL

http://mikami.a.la9.jp/meiji/MEIJI.HTM

担当講師

三上廉司(みかみれんじ)

Renji_Mikami(at_mark)nifty.com (Default - Recommended)

mikami(at_mark)meiji.ac.jp (Alternative)

http://mikami.a.la9.jp/_edu.htm