１　PSoC による圧力センサ　SCP1000 の取り込み

このドキュメントでは，圧力センサである SCP1000 を用いてどのように PSoC で取り込むかの解説を行う．SCP1000 は，絶対圧力センサであり，通信には，SPI が採用されている．SPI 通信は，PSoC でもサポートされているが，ここでは，動作の確認を最優先するため，VTI テクノロジー社の資料である,「C-CODE EXAMPLE FOR SCP1000-D01 PRESSURE SENSOR」をベースにハンドシェイクによる実装を行う。

次に，SPIM ブロックを用いたバージョン，さらに割り込みを使ったものを示していく。

２　１.サンプルドキュメントに基づいた実装例

VTI テクノロジー社の資料である,「C-CODE EXAMPLE FOR SCP1000-D01 PRESSURE SENSOR」をベースに実装を行う。このドキュメントには，C のソースコードがついているが，ATMEL Atmgega16L 用のソースコードであり，PSoC で使用するには，若干修正する必要がある。

各ブロックで使うクロック設定の考え方

データ出力ため UART を使うが，VC1,VC2,VC3 とは独立して設定するようにするため，PWM16 ブロックから UART 専用のクロックの生成を行う。 センサの SPI 通信では，最大 500kHz＊2 までのクロックを入力できるので，VC1 を 12 として 1MHz を作り，VC1 を Clock を基本周波数として使う。

今回のセンサは，3.3V 動作であるため，Global Resource の設定は次のようにした。

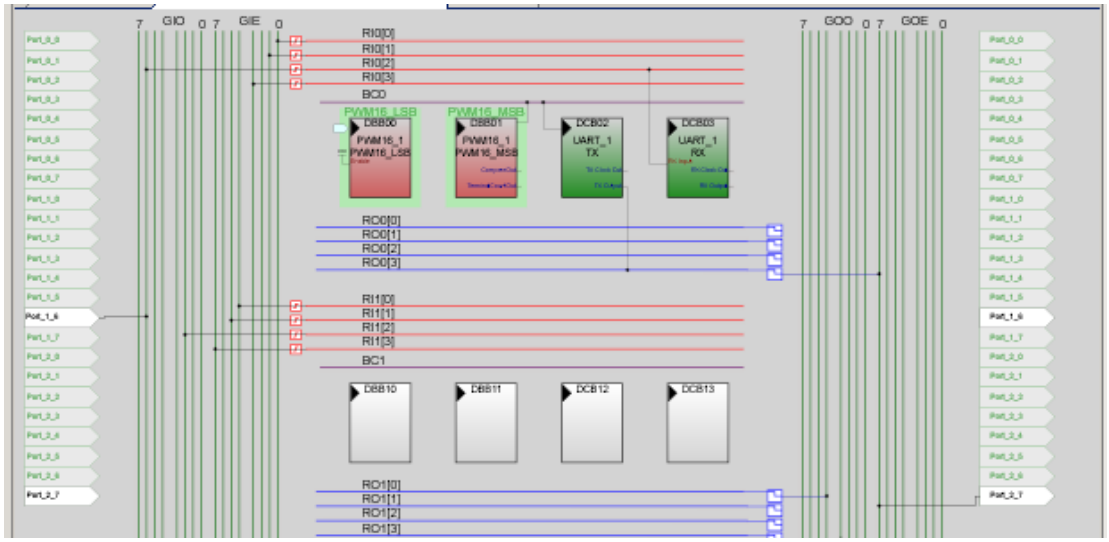
Global Resources - scp_1000_nospim

Power Setting [Vcc / SysClk	3.3V / 24MHz
CPU_Clock	SysClk/2
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	12
VC2= VC1/N	4
VC3 Source	VC1
VC3 Divider	1
SysClk Source	Internal
SysClk*2 Disable	No
Analog Power	SC On/Ref High
Ref Mux	(Vdd/2)+/-(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (SMP)]	4.81V (5.00V)
LVDThrottleBack	Disable
Watchdog Enable	Disable

Power Setting が 3.3V では，CPU_Clock は SysClk/2 が最大となる。今回は，それ以外の設定は，デフォルトのままでも構わない。

ブロックの配置は以下のようにした。

PWM16_1,UART_1 を図のように配置する。



３　使用するブロックの設定　PWM16_1

Properties - PWM16_1

Name	PWM16_1
User Module	PWM16
Version	2.5
Clock	SysClk*2
Enable	High
CompareOut	None
TerminalCountOut	None
Period	0
PulseWidth	0
CompareType	Less Than Or Equal
InterruptType	Terminal Count
ClockSync	Unsynchronized
InvertEnable	Normal

Clock は，VC1～VC3 の変更に影響が無いように，SysClk*2 を選択する。

４　使用するブロックの設定　UART_1

Properties - UART_1

Name	UART_1
User Module	UART
Version	5.2
Clock	Row_0_Broadcast
RX Input	Row_0_Input_2
TX Output	Row_0_Output_3
TX Interrupt Mode	TXRegEmpty
ClockSync	Sync to SysClk
RxCmdBuffer	Enable
RxBufferSize	16
CommandTerminator	13
Param_Delimiter	32
IgnoreCharsBelow	32
Enable_BackSpace	Disable
RX Output	None
RX Clock Out	None
TX Clock Out	None
InvertRX Input	Normal

Row_0_Broadcast から DBB01 に接続し，それを UART の Clock に接続するようにする。

ちなみにこのようにした場合，選択可能な bps は，以下のように計算できる。

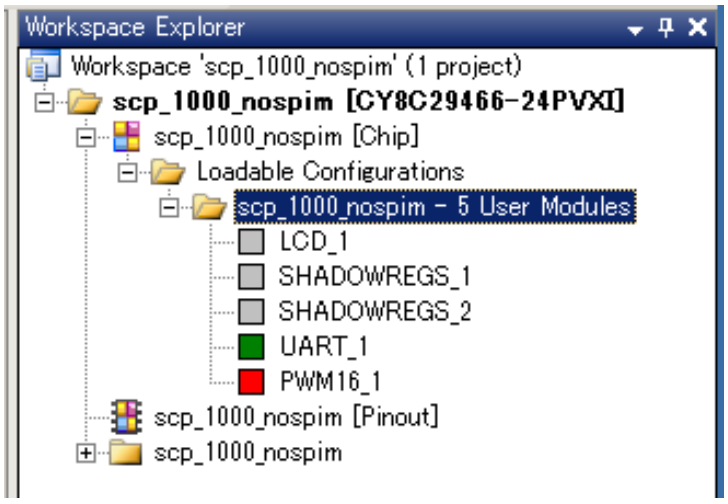
bps	計算式	PWM16 Period
9600	24*10^6*2/9600/8	625
19200	24*10^6*2/19200/8	313
38400	24*10^6*2/38400/8	156
57600	24*10^6*2/57600/8	104
115200	24*10^6*2/115200/8	52

つまり，

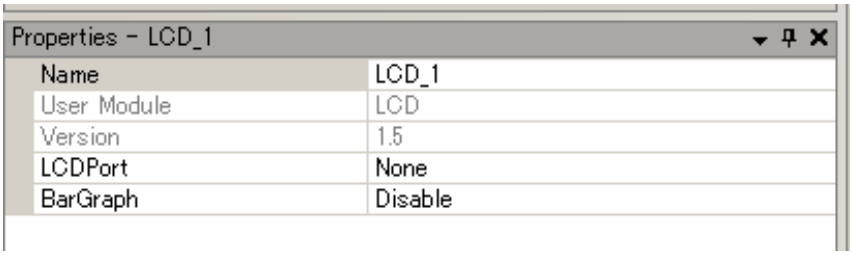
```
//   9600 bps PWM16_1_WritePeriod(625-1);PWM16_1_WritePulseWidth(312);
// 19200 bps PWM16_1_WritePeriod(313-1);PWM16_1_WritePulseWidth(156);
// 38400 bps PWM16_1_WritePeriod(156-1);PWM16_1_WritePulseWidth(78);
// 57600 bps PWM16_1_WritePeriod(104-1);PWM16_1_WritePulseWidth(52);
//115200 bps PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26);
```

５　その他のブロック

ここでは，ウェイト時間を生成するため，ビットの操作を安全に行うためさらに以下のブロックを追加した。



LCD_1 ブロックは，実際には使用しないため，LCD_Port の設定はしない。
関数の `LCD_1_Delay50uTimes` のみを使用するために配置している。



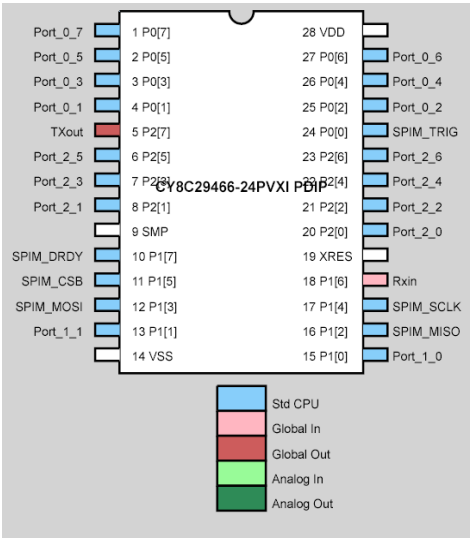
SHADOWREGES_1 は，ShadowPort　Port_0
SHADOWREGES_2 は，ShadowPort　Port_1
に対応するようにした。ちなみに，SHADOWREGES をプログラム中で使うには，
extern 宣言をする必要がある。

```
extern BYTE Port_0_Data_SHADE;
extern BYTE Port_1_Data_SHADE;
```

実際の使用例

```
Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
Port_1_Data_SHADE |=  0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='H'
Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
Port_0_Data_SHADE |=  0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='H'
DRDY=0;if(PRT1DR & 0x80) DRDY=1;
MISO=0;if(PRT1DR & 0x04) MISO=1;
```

6　PINOUT レイアウト



Port_1_0,Port_1_1 は，プログラムを書き込むときに使用するため，このプロジェクトでは，Pin アサインを避けている。センサにアクセスするためのピンと，232C からデータを出力

するためのピンは以下のとおりである。

基本的に，PSoC からの出力は，Strong，入力は，High Z としている．

PSoC から	ポート番号	設定
TXout（出力）	P27	Strong, DisableInt
SPIM_DRDY（入力）	P17	StdCPU,High Z,DisableInt
SPIM_CSB（出力）	P15	StdCPU,Strong,DisableInt
SPIM_MOSI（出力）	P13	StdCPU,Strong,DisableInt
SPIM_TRIG（出力）	P00	StdCPU,PullDown,DisableInt
Rxin（入力）	P16	High Z, DisableInt
SPIM_SCLK（出力）	P14	StdCPU,Strong,DisableInt
SPIM_MISO（入力）	P12	StdCPU,High Z,DisableInt

```
// P10,P11 for Miniprog

232 TTL Converter
Jumper J1 1-2 short 3.3V
Jumper J2 short from USB to Vcc supply
232Module---PSoC
TXD(1)------RX_in(18)P16
RXD(5)------TXout(5)P27
GND(24)
GND(7)
VCC(21)
VCC(15)
PSOC -- SCP1000
P00(24)-->1(Trig)
P17(10)<--2(DRDY)
P14(17)-->3(SCLK)
GND(14)-->4(GND)
P13(12)-->5(MOSI)
P12(16)<--6(MISO)
P15(11)-->7(CSB)
Vcc(28)-->8(Vcc)
```

以上の設定を行ない。プログラムを作成する。ここでは，サンプルプログラムに準拠し以下のように変更し作成した。

オリジナルと異なる主な変更点は以下のところである．

wait_ms 関数は，ウェイトをかける関数であり，ここでは，CPU のクロックに依存しない，LCD_1_Delay50uTimes 関数を使用して作成している．device_init 関数は，各ピンを初期化する関数である．SPIM_2_SendTxData 関数，SPIM_2_bReadRxData 関数は，PSoC の SPIM ブロックの関数とほぼ互換の関数である．

```
void wait_ms(unsigned int ms){
  int i,j;
  if(ms > 12) {
    i = (int)(ms / 12);
    for(j = 0;j < i;j++) {
      LCD_1_Delay50uTimes(240);
      ms -= 12;
    }
  }
  LCD_1_Delay50uTimes(ms * 20);
}
```

```
void device_init(void) {
 Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
 Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
 Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
 Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
}
void SPIM_2_SendTxData(unsigned char data){
  int    n = 8;
  while (n--) {
    if (data & (unsigned char)0x80) {
      Port_1_Data_SHADE |=  0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='H'
    } else {
      Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
    }
    Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
    Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
    data <<= 1;
  }
}
unsigned charSPIM_2_bReadRxData(void){
  int        n = 8;
  unsigned char  data = 0;
  while (n--) {
    Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
    data <<= 1;
    if (PRT1DR & 0x04) {// MISO
      data |= (unsigned char)0x01;
    }
    Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
  }
  return data;
}
```

また，Read_Direct_Access_SPI 関数と，Write_Direct_Access_SPI 関数は，オリジナルの関数をベースに修正した関数である．

```
unsigned int Read_Direct_Access_SPI(unsigned char address, int number_of_bytes){
  unsigned int value;
  value = 0;
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_2_SendTxData(address);
  value = (unsigned int)(SPIM_2_bReadRxData());
  if(number_of_bytes > 1) {
    value <<= 8;
      value |= (unsigned int)(SPIM_2_bReadRxData());
  }
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  return value;
}
void Write_Direct_Access_SPI(unsigned char address, unsigned char data){
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  address |= 0x02; // #2, set write bit to one (RW=1)
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_2_SendTxData(address); // get revision number
  SPIM_2_SendTxData(data); // get revision number
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
}
```

オリジナルのC言語のプログラムでは，PD ピンを利用しているが，使用しているセンサでは，PD は，グランドに接続されているため，ASIC ソフトウェアリセットによる方法で実装している．

このプログラムを MiniProg で書き込む場合，デフォルトでは，５V となるため，センサが壊れてしまう可能性がある．そこで，書き込む場合には，センサを一時抜く必要がある。

実行は，Teraterm などを起動し，文字を適当に入力し Enter をすると入力した文字が表示される。Z を入力し Enter をするとセンサのデータの読み込みは開始される。なお，こ

のプログラムをコピーペーストして実行する場合には，￥と＼に注意すること。PSoC Designer での表示は，＼になる。￥では，フォントが異なる。

```
/*************************************************************
* INITIALIZE SCP1000 AND ACTIVATE MEASUREMENT MODE
* -----------------------------------------------------------
* 1. SCP1000 power supplies are to set up and stabilized before the SCP1000 is initialized
* 2. Set CSB and PD high ('1') and wait 1ms
* 3. Pull PD low ('0') and wait 60ms
* 4. Read STATUS register (0x07) and check that STARTUP bit (LSB, bit0) is zero, if not
* SCP1000 start-up procedure is running --> check STATUS.STARTUP in loop for 6 times
* if it is not zero after 6 cycles, SCP1000 start-up has failed
* 5. Read DATARD8 register and check that LSB (bit0) is one, if not
* EEPROM checksum error is detected --> SCP1000 will not give reliable pressure data
* 6. Low noise configuration (done with indirect write)
* Write 0x2D to ADDPTR (0x02)
* Write 0x03 to DATAWR (0x01)
* Write 0x02 to OPERATION (0x03)
* 7. wait 100ms
* 8. SCP1000 is in standby mode
* 9. Set measurement mode (High resolution in this example).
*************************************************************/
void init_scp1000(void){
  unsigned int DATA;
  int i;
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  wait_ms(60); // #3, 60ms wait
  wait_ms(90); // #3, 90ms wait
  Write_Direct_Access_SPI(0x06,0x01); // ASIC software reset
  wait_ms(10); // #3, 10ms wait

  wait_ms(60); // #3, 60ms wait
  for(i = 6; i > 0 ; i--){
    DATA = Read_Direct_Access_SPI(0x07, 0x01);// #4, read STATUS register --> LSB '0'=OK
    if(!(DATA & 0x0001)) break;
    wait_ms(10);
  }
  if(i == 0) fail(0);
  DATA = Read_Direct_Access_SPI(0x1F, 1); // #5, read DATARD8 register --> LSB '1'=OK
  if(!(DATA && 0x0001)) fail(1);
  Write_Indirect_Access_SPI(0x2D, 0x03); // #6, Low noise configuration
  wait_ms(100); // #7, wait for 100ms
  Write_Direct_Access_SPI(0x03, 0x00); // Reset operation mode
  wait_ms(10); // Wait before change new mode
//Write_Direct_Access_SPI(0x03, 0x0A); // #9, set SCP1000 into 'high resolution'
  Write_Direct_Access_SPI(0x03, 0x09); // #9, set SCP1000 into 'high speed'
  // measurement mode (0x0A)
  wait_ms(100);
}
```

main.c

```
#include <m8c.h>
#include "PSoCAPI.h"
#include <stdlib.h>
#include <math.h>
extern BYTE Port_0_Data_SHADE;
extern BYTE Port_1_Data_SHADE;
// UART PWM16_1
// VC1=SysClk/N --> VC1=24MHz/24=1MHz
//  9600 bps PWM16_1_WritePeriod(625-1);PWM16_1_WritePulseWidth(312);
// 19200 bps PWM16_1_WritePeriod(313-1);PWM16_1_WritePulseWidth(156);
// 38400 bps PWM16_1_WritePeriod(156-1);PWM16_1_WritePulseWidth(78);
// 57600 bps PWM16_1_WritePeriod(104-1);PWM16_1_WritePulseWidth(52);
//115200 bps PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26);
//For SPIM_1/////////////
// CSB P1[5] 01,02,04,08,16,32,64,128
// CSB P1[5] 01,02,04,08,10,20,40, 80
//         0  1  2  3  4  5  6  7
// MISO,MOSI,SCLK, CSB,RXin,DRDY
// 0x04,0x08,0x10,0x20,0x40,0x80
// SPI have to set less than 500kHz
```

```
// 24MHz/12(VC1)/4(VC2)->500kHz
// P10,P11 for Miniprog
// Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
// Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
// Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
// Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
// Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
// Port_1_Data_SHADE |=  0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='H'
// Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
// Port_0_Data_SHADE |=  0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='H'
//  DRDY=0;if(PRT1DR & 0x80) DRDY=1;
//  MISO=0;if(PRT1DR & 0x04) MISO=1;
/*
232 TTL Converter
Jumper J1 2-3 short VCC supply
Jumper J2 short from USB to Vcc supply
232Module---PSoC
TXD(1)------ P16(18) RX_in
RXD(5)------ P27(5) TXout
GND(24)
GND(7)
VCC(21)
VCC(15)

Pin assignment
//FOR UART
TX_out,,Strong, DisableInt
RX_in,,High Z, DisableInt

//FOR SPIM_1
SPIM_SCLK,,Strong,DisableInt
SPIM_MOSI,,Strong,DisableInt
SPIM_MISO,,High Z,DisableInt

// FOR sensor handshake for SPIM_1
SPIM_CSB,StdCPU,Strong,DisableInt
SPIM_TRIG,StdCPU,Strong,DisableInt
SPIM_DRDY,StdCPU,High Z,DisableInt

*/
/* For PSoC Writer
PSOC ---MiniProg
Vdd(28)---1
GND(14)---2
XRES(19)---3
P11(13)---4
P10(15)---5
*/
/*
PSOC -- SCP1000
P00(24)-->1(Trig)
P17(10)<--2(DRDY)
P14(17)-->3(SCLK)
GND(14)-->4(GND)
P13(12)-->5(MOSI)
P12(16)<--6(MISO)
P15(11)-->7(CSB)
Vcc(28)-->8(Vcc)
*/
/* Data type
unsigned char 1 byte
unsigned short 2 byte  ***** !!!
unsigned int 2 byte
unsigned long 4 byte
*/
void wait_ms(unsigned int ms){
  int i,j;
  if(ms > 12) {
    i = (int)(ms / 12);
    for(j = 0;j < i;j++) {
      LCD_1_Delay50uTimes(240);
      ms -= 12;
```

```
    }
  }
  LCD_1_Delay50uTimes(ms * 20);
}
void device_init(void) {
 Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
 Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
 Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
 Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
}
void SPIM_2_SendTxData(unsigned char data){
  int    n = 8;
  while (n--) {
    if (data & (unsigned char)0x80) {
      Port_1_Data_SHADE |=  0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='H'
    } else {
      Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
    }
    Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
    Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
    data <<= 1;
  }
}
unsigned charSPIM_2_bReadRxData(void){
  int        n = 8;
  unsigned char  data = 0;
  while (n--) {
    Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
    data <<= 1;
    if (PRT1DR & 0x04) {// MISO
      data |= (unsigned char)0x01;
    }
    Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
  }
  return data;
}
void fail(int i){
  int j=0;
  for(j = 0;j < 100;j++) {UART_1_CPutString("Sensor Fail");UART_1_PutChar('0'+i);}
}
/*****************************************************************
* DIRECT READ, inputs: register address, number of bytes to be read
* ------------
* 1. Convert the SCP1000 register address to real address by shifting the
* bit pattern to left by 2 bits ([xxAAAAAA] --> [A A A A A A RW 0], A = address bit):
* (MSB) A A A A A A A A RW 0 (LSB)
*       |           |  | *- allways zero
*       |           |  *--- Read/Write Bit (Read: RW=0, Write: RW=1)
*       *-----------*------ Register address bits
* 2. Set Read/Write bit to '0' at the address byte:
* [A A A A A A RW 0] --> [A A A A A A 0 0]
* The RW bit is zero automatically after the register address is shifted to left by 2
* 3. Set CSB to low
* 4. Send address byte (register)
* 5. Send SCK (clock cycles) and read data bytes
* 6. Set CSB to HIGH
*****************************************************************/
unsigned int Read_Direct_Access_SPI(unsigned char address, int number_of_bytes){
  unsigned int value;
  value = 0;
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_2_SendTxData(address);
  value = (unsigned int)(SPIM_2_bReadRxData());
  if(number_of_bytes > 1) {
    value <<= 8;
      value |= (unsigned int)(SPIM_2_bReadRxData());
  }
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  return value;
}
/*****************************************************************
```

```
* DIRECT WRITE, inputs: register address, register data
* ------------
* 1. Convert the SCP1000 register address to real address by shifting the
* bit pattern to left by 2 bits ([xxAAAAAA] --> [A A A A A A RW 0], A = address bit):
* (MSB) A A A A A A A RW 0 (LSB)
*      |          |  | *- allways zero
*      |          |  *--- Read/Write Bit (Read: RW=0, Write: RW=1)
*      *-----------*------ Register address bits
* 2. Set read/write bit to '1' at the address byte:
* [A A A A A A RW 0] --> [A A A A A A 1 0]
* 3. Set CSB to low
* 4. Send register address byte
* 5. Send register content (data byte)
* 6. Set CSB to HIGH
*****************************************************************/
void Write_Direct_Access_SPI(unsigned char address, unsigned char data){
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  address |= 0x02; // #2, set write bit to one (RW=1)
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_2_SendTxData(address); // get revision number
  SPIM_2_SendTxData(data); // get revision number
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
}
/*****************************************************************
* INDIRECT WRITE, input: indirect register address, register data
* --------------
* Indirect write requires three DIRECT WRITE actions:
* 1. Use DIRECT WRITE to write register address to ADDPTR (0x02)
* 2. Use DIRECT WRITE to write data to DATAWR (0x01)
* 3. Use DIRECT WRITE to write "indirect write" command 0x02 to OPERATION (0x03)
* 4. Wait 50ms
*****************************************************************/
void Write_Indirect_Access_SPI(unsigned char address, unsigned char data){
  Write_Direct_Access_SPI(0x02, address); // #1, write reg address to ADDPTR (0x02)
  Write_Direct_Access_SPI(0x01, data); // #2, write reg data to DATAWR (0x01)
  Write_Direct_Access_SPI(0x03, 0x02); // #3, write 0x02 to OPERATION (0x03)
  wait_ms(50); // #4, wait 50ms
}
/*****************************************************************
* INDIRECT READ, input: indirect register address
* -------------
* Indirect read requires three DIRECT read/write actions:
* 1. Use DIRECT WRITE to write the register address to ADDPTR (0x02)
* 2. Use DIRECT WRITE to write the "read indirect" command 0x01 to OPERATION (0x03)
* 3. Wait 10ms
* 4. Use DIRECT READ to read 2 bytes from DATARD16 (0x20), the register content
* is in bits [7:0], bits [15:8] should be treated as zeros
*****************************************************************/
unsigned int Read_Indirect_Access_SPI(unsigned char address){
  unsigned int value;
  Write_Direct_Access_SPI(0x02, address); // #1, write reg address to ADDPTR (0x02)
  Write_Direct_Access_SPI(0x03, 0x01); // #2, write 0x01 to OPERATION (0x03)
  wait_ms(10); // #3, wait 10ms
  value = Read_Direct_Access_SPI(0x20, 0x02); // #4, read two bytes from DATARD16 (0x20)
  return value;
}
/*****************************************************************
* INITIALIZE SCP1000 AND ACTIVATE MEASUREMENT MODE
* ----------------------------------------------------------------
* 1. SCP1000 power supplies are to set up and stabilized before the SCP1000 is initialized
* 2. Set CSB and PD high ('1') and wait 1ms
* 3. Pull PD low ('0') and wait 60ms
* 4. Read STATUS register (0x07) and check that STARTUP bit (LSB, bit0) is zero, if not
* SCP1000 start-up procedure is running --> check STATUS.STARTUP in loop for 6 times
* if it is not zero after 6 cycles, SCP1000 start-up has failed
* 5. Read DATARD8 register and check that LSB (bit0) is one, if not
* EEPROM checksum error is detected --> SCP1000 will not give reliable pressure data
* 6. Low noise configuration (done with indirect write)
* Write 0x2D to ADDPTR (0x02)
* Write 0x03 to DATAWR (0x01)
* Write 0x02 to OPERATION (0x03)
* 7. wait 100ms
```

```
* 8. SCP1000 is in standby mode
* 9. Set measurement mode (High resolution in this example).
*****************************************************************/
void init_scp1000(void){
  unsigned int DATA;
  int i;
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  wait_ms(60); // #3, 60ms wait
  wait_ms(90); // #3, 90ms wait
  Write_Direct_Access_SPI(0x06,0x01); // ASIC software reset
  wait_ms(10); // #3, 10ms wait

  wait_ms(60); // #3, 60ms wait
  for(i = 6; i > 0 ; i--){
    DATA = Read_Direct_Access_SPI(0x07, 0x01);// #4, read STATUS register --> LSB '0'=OK
    if(!(DATA & 0x0001)) break;
    wait_ms(10);
  }
  if(i == 0) fail(0);
  DATA = Read_Direct_Access_SPI(0x1F, 1); // #5, read DATARD8 register --> LSB '1'=OK
  if(!(DATA && 0x0001)) fail(1);
  Write_Indirect_Access_SPI(0x2D, 0x03); // #6, Low noise configuration
  wait_ms(100); // #7, wait for 100ms
  Write_Direct_Access_SPI(0x03, 0x00); // Reset operation mode
  wait_ms(10); // Wait before change new mode
//Write_Direct_Access_SPI(0x03, 0x0A); // #9, set SCP1000 into 'high resolution'
  Write_Direct_Access_SPI(0x03, 0x09); // #9, set SCP1000 into 'high speed'
  // measurement mode (0x0A)
  wait_ms(100);
}
/*****************************************************************
* MAIN
* ----
* 1. Setup, configure MCU
* 2. Init_SCP1000, initialize SCP1000 and activate measurement mode
* 3. Read SCP1000 pressure and temperature every time when new measurement
* results are available (e.g. DRDY pin is '1')
*****************************************************************/
int main(void){
  char * strPtr; // Parameter pointer
  unsigned int temp;
  unsigned long pressure;
  Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
  device_init();

  UART_1_CmdReset();
  UART_1_IntCntl(UART_1_ENABLE_RX_INT);
  PWM16_1_WritePeriod(625-1);PWM16_1_WritePulseWidth(312);//9600 bps
//PWM16_1_WritePeriod(313-1);PWM16_1_WritePulseWidth(156);// 19200 bps
//PWM16_1_WritePeriod(156-1);PWM16_1_WritePulseWidth(78);// 38400 bps
//PWM16_1_WritePeriod(104-1);PWM16_1_WritePulseWidth(52);// 56600 bps
//PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26); // 115200 bps

   PWM16_1_DisableInt();
   PWM16_1_Start();
  UART_1_Start(UART_1_PARITY_NONE);
  M8C_EnableGInt;
  while(1) {
      if(UART_1_bCmdCheck()) {                  // Wait for command
        if(strPtr = UART_1_szGetParam()) {      // More than delimiter?
          UART_1_CPutString("Found valid command¥r¥nCommand =>");
          UART_1_PutString(strPtr);
       if(strPtr[0] == 'Z') break;
       UART_1_CPutString("<¥r¥nParamaters:¥r¥n");
          while(strPtr = UART_1_szGetParam()) { // loop on each parameter
            UART_1_CPutString("   <");
            UART_1_PutString(strPtr);         // Print each parameter
            UART_1_CPutString(">¥r¥n");
          }
        }
      UART_1_CmdReset();                        // Reset command buffer
      }
```

```
    }
    init_scp1000(); // #2, Initialize SCP1000, activate
    // 'high resolution' measurement mode
    UART_1_CPutString("init_scp1000() done");
    for(;;){
      while(!(PRT1DR & 0x80)) wait_ms(1);// DRDY
      temp = Read_Direct_Access_SPI(0x21, 0x02);// Read 2 bytes from TEMPOUT (0x21)
      temp = temp & 0x3fff; // Mask bits [13:0]
      temp = temp * 100 / 20; // Convert temperature to [□‹C] by dividing
      // the result by 20
      pressure = (unsigned long)((0x0007) & Read_Direct_Access_SPI(0x1F, 1));
      // Read 1 byte from DATARD8 (0x1F) and
      // mask bits [2:0] in order to get
      // 3MSB bits for pressure information
      pressure <<= 16; // shift 3MSB bits to left by 16
      pressure |= (unsigned long)Read_Direct_Access_SPI(0x20, 2);
      // Read 2 bytes from DATARD16 (0x20) in
      // order to get 16 LSB bits for pressure
      // information.
      pressure *= 100;
      pressure >>= 2; // Convert to [Pa] by dividing the 19 bit
      // pressure by 4 --> the 19 bit pattern
      // is shifted to right by 2
      // <- Here user can add temperature and pressure data send / display routines
      // as needed for application. Notice that temperature is in 2's complement format.
      UART_1_PutSHexInt((int)temp);
      UART_1_PutSHexInt((int)(pressure >> 16));
      UART_1_PutSHexInt((int)(pressure & 0xffff));UART_1_PutChar('¥r');
    }
    return 0;
}
```

2.SPIM ブロックを用いた実装

PSoC では，SPI 通信のための SPIM ブロックが用意されている。このブロックを使った方が，処理速度的には早くなる。

SPIM ブロックの追加とパラメータの変更

SPI の Clock の最大は，センサの制約により 500kHz となっている。

そこで，SPIM ブロックでは，2 倍のクロックを入力するので，VC2 から 1MHz 出力されるように，VC1,VC2 を設計する。

VC1 は，12，VC2 は 1 とする。

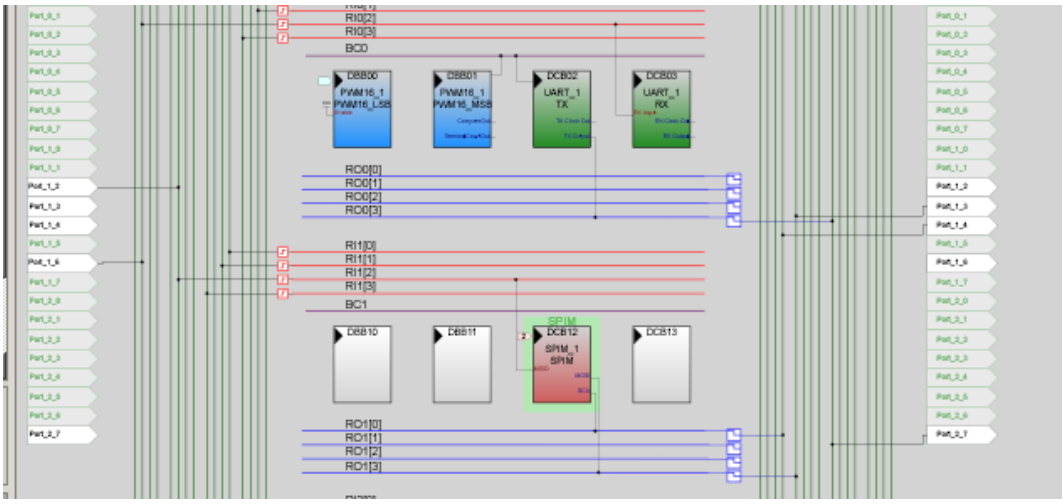
Power Setting [Vcc	3.3V / 24MHz
CPU_Clock	SysClk/2
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	12
VC2= VC1/N	1
VC3 Source	VC1
VC3 Divider	1
SysClk Source	Internal
SysClk*2 Disable	No
Analog Power	SC On/Ref High
Ref Mux	(Vdd/2)+/-(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (S	4.81V (5.00V)
LVDThrottleBack	Disable
Watchdog Enable	Disable

SPIM ブロックの設定は次の通り，

Properties - SPIM_1

Name	SPIM_1
User Module	SPIM
Version	2.6
Clock	VC2
MISO	Row_1_Input_2
MOSI	Row_1_Output_3
SClk	Row_1_Output_0
Interrupt Mode	TXRegEmpty
ClockSync	Sync to SysClk
InvertMISO	Normal

配線の追加箇所



主な変更点は，Read_Direct_Access_SPI 関数，Write_Direct_Access_SPI 関数である．

```
unsigned int Read_Direct_Access_SPI(unsigned char address, int number_of_bytes){
  unsigned int value;
  value = 0;
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_1_SendTxData(address);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  SPIM_1_SendTxData(0xff);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  value = (unsigned int)(SPIM_1_bReadRxData());
  if(number_of_bytes > 1) {
    value <<= 8;
    SPIM_1_SendTxData(0xff);
    while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
      value |= (unsigned int)(SPIM_1_bReadRxData());
  }
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  return value;
}
void Write_Direct_Access_SPI(unsigned char address, unsigned char data){
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  address |= 0x02; // #2, set write bit to one (RW=1)
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_1_SendTxData(address);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  SPIM_1_SendTxData(data);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
}
```

それに main 文内の

```
    SPIM_1_EnableInt();

    SPIM_1_Start(SPIM_1_SPIM_MODE_0 | SPIM_1_SPIM_MSB_FIRST);
```

である。

プログラミング上の注意点としては，SPIM_1_bReadRxData 関数単体では，PSoC から SCLK が発生しない。そのため，出力に対し，応答してくるようなタイプのデバイスでは，仮にダミーデータを送る必要がある。

```
#include <m8c.h>
#include "PSoCAPI.h"
#include <stdlib.h>
#include <math.h>
extern BYTE Port_0_Data_SHADE;
extern BYTE Port_1_Data_SHADE;
// UART PWM16_1
// VC1=SysClk/N --> VC1=24MHz/24=1MHz
//  9600 bps PWM16_1_WritePeriod(625-1);PWM16_1_WritePulseWidth(312);
// 19200 bps PWM16_1_WritePeriod(313-1);PWM16_1_WritePulseWidth(156);
// 38400 bps PWM16_1_WritePeriod(156-1);PWM16_1_WritePulseWidth(78);
// 57600 bps PWM16_1_WritePeriod(104-1);PWM16_1_WritePulseWidth(52);
//115200 bps PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26);
//For SPIM_1/////////////
// CSB P1[5] 01,02,04,08,16,32,64,128
// CSB P1[5] 01,02,04,08,10,20,40, 80
//          0  1  2  3  4  5  6  7
// MISO,MOSI,SCLK, CSB,RXin,DRDY
// 0x04,0x08,0x10,0x20,0x40,0x80
// SPI have to set less than 500kHz
// 24MHz/12(VC1)/4(VC2)->500kHz
// P10,P11 for Miniprog
// Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
// Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
// Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
// Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
// Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
// Port_1_Data_SHADE |=  0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='H'
// Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
```

```
// Port_0_Data_SHADE |=  0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='H'
//  DRDY=0;if(PRT1DR & 0x80) DRDY=1;
//  MISO=0;if(PRT1DR & 0x04) MISO=1;
/*
232 TTL Converter
Jumper J1 2-3 short VCC supply
Jumper J2 short from USB to Vcc supply
232Module---PSoC
TXD(1)------ P16(18)RX_in
RXD(5)------ P27(5) TXout
GND(24)
GND(7)
VCC(21)
VCC(15)

Pin assignment
//FOR UART
TX_out,,Strong, DisableInt
RX_in,,High Z, DisableInt

//FOR SPIM_1
SPIM_SCLK,,Strong,DisableInt
SPIM_MOSI,,Strong,DisableInt
SPIM_MISO,,High Z,DisableInt

// FOR sensor handshake for SPIM_1
SPIM_CSB,StdCPU,Strong,DisableInt
SPIM_TRIG,StdCPU,Strong,DisableInt
SPIM_DRDY,StdCPU,High Z,DisableInt

*/
/* For PSoC Writer
PSOC ---MiniProg
Vdd(28)---1
GND(14)---2
XRES(19)---3
P11(13)---4
P10(15)---5
*/
/*
PSOC -- SCP1000
P00(24)-->1(Trig)
P17(10)<--2(DRDY)
P14(17)-->3(SCLK)
GND(14)-->4(GND)
P13(12)-->5(MOSI)
P12(16)<--6(MISO)
P15(11)-->7(CSB)
Vcc(28)-->8(Vcc)
*/
/* Data type
unsigned char 1 byte
unsigned short 2 byte  ***** !!!
unsigned int 2 byte
unsigned long 4 byte
*/
void wait_ms(unsigned int ms){
  int i,j;
  if(ms > 12) {
    i = (int)(ms / 12);
    for(j = 0;j < i;j++) {
      LCD_1_Delay50uTimes(240);
      ms -= 12;
    }
  }
  LCD_1_Delay50uTimes(ms * 20);
}
void device_init(void){
 Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
 Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
 Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
 Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
```

```
}
void fail(int i){
  int j=0;
  for(j = 0;j < 100;j++) {UART_1_CPutString("Sensor Fail");UART_1_PutChar('0'+i);}
}
/*************************************************************
* DIRECT READ, inputs: register address, number of bytes to be read
* ------------
* 1. Convert the SCP1000 register address to real address by shifting the
* bit pattern to left by 2 bits ([xxAAAAAA] --> [A A A A A A RW 0], A = address bit):
* (MSB) A A A A A A A RW 0 (LSB)
*       |          |  | *- allways zero
*       |          |  *--- Read/Write Bit (Read: RW=0, Write: RW=1)
*       *-----------*------ Register address bits
* 2. Set Read/Write bit to '0' at the address byte:
* [A A A A A A RW 0] --> [A A A A A A 0 0]
* The RW bit is zero automatically after the register address is shifted to left by 2
* 3. Set CSB to low
* 4. Send address byte (register)
* 5. Send SCK (clock cycles) and read data bytes
* 6. Set CSB to HIGH
*************************************************************/
unsigned int Read_Direct_Access_SPI(unsigned char address, int number_of_bytes){
  unsigned int value;
  value = 0;
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_1_SendTxData(address);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  SPIM_1_SendTxData(0xff);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  value = (unsigned int)(SPIM_1_bReadRxData());
  if(number_of_bytes > 1) {
    value <<= 8;
    SPIM_1_SendTxData(0xff);
    while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
      value |= (unsigned int)(SPIM_1_bReadRxData());
  }
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  return value;
}
/*************************************************************
* DIRECT WRITE, inputs: register address, register data
* ------------
* 1. Convert the SCP1000 register address to real address by shifting the
* bit pattern to left by 2 bits ([xxAAAAAA] --> [A A A A A A RW 0], A = address bit):
* (MSB) A A A A A A A RW 0 (LSB)
*       |          |  | *- allways zero
*       |          |  *--- Read/Write Bit (Read: RW=0, Write: RW=1)
*       *-----------*------ Register address bits
* 2. Set read/write bit to '1' at the address byte:
* [A A A A A A RW 0] --> [A A A A A A 1 0]
* 3. Set CSB to low
* 4. Send register address byte
* 5. Send register content (data byte)
* 6. Set CSB to HIGH
*************************************************************/
void Write_Direct_Access_SPI(unsigned char address, unsigned char data){
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  address |= 0x02; // #2, set write bit to one (RW=1)
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_1_SendTxData(address);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  SPIM_1_SendTxData(data);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
}
/*************************************************************
* INDIRECT WRITE, input: indirect register address, register data
* --------------
* Indirect write requires three DIRECT WRITE actions:
* 1. Use DIRECT WRITE to write register address to ADDPTR (0x02)
```

```
* 2. Use DIRECT WRITE to write data to DATAWR (0x01)
* 3. Use DIRECT WRITE to write "indirect write" command 0x02 to OPERATION (0x03)
* 4. Wait 50ms
*****************************************************************/
void Write_Indirect_Access_SPI(unsigned char address, unsigned char data){
  Write_Direct_Access_SPI(0x02, address); // #1, write reg address to ADDPTR (0x02)
  Write_Direct_Access_SPI(0x01, data); // #2, write reg data to DATAWR (0x01)
  Write_Direct_Access_SPI(0x03, 0x02); // #3, write 0x02 to OPERATION (0x03)
  wait_ms(50); // #4, wait 50ms
}
/*****************************************************************
* INDIRECT READ, input: indirect register address
* -------------
* Indirect read requires three DIRECT read/write actions:
* 1. Use DIRECT WRITE to write the register address to ADDPTR (0x02)
* 2. Use DIRECT WRITE to write the "read indirect" command 0x01 to OPERATION (0x03)
* 3. Wait 10ms
* 4. Use DIRECT READ to read 2 bytes from DATARD16 (0x20), the register content
* is in bits [7:0], bits [15:8] should be treated as zeros
*****************************************************************/
unsigned int Read_Indirect_Access_SPI(unsigned char address){
  unsigned int value;
  Write_Direct_Access_SPI(0x02, address); // #1, write reg address to ADDPTR (0x02)
  Write_Direct_Access_SPI(0x03, 0x01); // #2, write 0x01 to OPERATION (0x03)
  wait_ms(10); // #3, wait 10ms
  value = Read_Direct_Access_SPI(0x20, 0x02); // #4, read two bytes from DATARD16 (0x20)
  return value;
}
/*****************************************************************
* INITIALIZE SCP1000 AND ACTIVATE MEASUREMENT MODE
* ----------------------------------------------------------------
* 1. SCP1000 power supplies are to set up and stabilized before the SCP1000 is initialized
* 2. Set CSB and PD high ('1') and wait 1ms
* 3. Pull PD low ('0') and wait 60ms
* 4. Read STATUS register (0x07) and check that STARTUP bit (LSB, bit0) is zero, if not
* SCP1000 start-up procedure is running --> check STATUS.STARTUP in loop for 6 times
* if it is not zero after 6 cycles, SCP1000 start-up has failed
* 5. Read DATARD8 register and check that LSB (bit0) is one, if not
* EEPROM checksum error is detected --> SCP1000 will not give reliable pressure data
* 6. Low noise configuration (done with indirect write)
* Write 0x2D to ADDPTR (0x02)
* Write 0x03 to DATAWR (0x01)
* Write 0x02 to OPERATION (0x03)
* 7. wait 100ms
* 8. SCP1000 is in standby mode
* 9. Set measurement mode (High resolution in this example).
*****************************************************************/
void init_scp1000(void){
  unsigned int DATA;
  int i;
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  wait_ms(60); // #3, 60ms wait
  wait_ms(90); // #3, 90ms wait
  Write_Direct_Access_SPI(0x06,0x01); // ASIC software reset
  wait_ms(10); // #3, 10ms wait

  wait_ms(60); // #3, 60ms wait
  for(i = 6; i > 0 ; i--){
    DATA = Read_Direct_Access_SPI(0x07, 0x01);// #4, read STATUS register --> LSB '0'=OK
    if(!(DATA & 0x0001)) break;
    wait_ms(10);
  }
  if(i == 0) fail(0);
  DATA = Read_Direct_Access_SPI(0x1F, 1); // #5, read DATARD8 register --> LSB '1'=OK
  if(!(DATA && 0x0001)) fail(1);
  Write_Indirect_Access_SPI(0x2D, 0x03); // #6, Low noise configuration
  wait_ms(100); // #7, wait for 100ms
  Write_Direct_Access_SPI(0x03, 0x00); // Reset operation mode
  wait_ms(10); // Wait before change new mode
//Write_Direct_Access_SPI(0x03, 0x0A); // #9, set SCP1000 into 'high resolution'
  Write_Direct_Access_SPI(0x03, 0x09); // #9, set SCP1000 into 'high speed'
  // measurement mode (0x0A)
```

```
  wait_ms(100);
}
/*****************************************************************
* MAIN
* ----
* 1. Setup, configure MCU
* 2. Init_SCP1000, initialize SCP1000 and activate measurement mode
* 3. Read SCP1000 pressure and temperature every time when new measurement
* results are available (e.g. DRDY pin is '1')
*****************************************************************/
int main(void){
  char * strPtr; // Parameter pointer
  unsigned int temp;
  unsigned long pressure;
  Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
  device_init();

  UART_1_CmdReset();
  UART_1_IntCntl(UART_1_ENABLE_RX_INT);
  PWM16_1_WritePeriod(625-1);PWM16_1_WritePulseWidth(312);//9600 bps
//PWM16_1_WritePeriod(313-1);PWM16_1_WritePulseWidth(156);// 19200 bps
//PWM16_1_WritePeriod(156-1);PWM16_1_WritePulseWidth(78);// 38400 bps
//PWM16_1_WritePeriod(104-1);PWM16_1_WritePulseWidth(52);// 56600 bps
//PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26); // 115200 bps

   PWM16_1_DisableInt();
   PWM16_1_Start();
  UART_1_Start(UART_1_PARITY_NONE);
  M8C_EnableGInt;
   SPIM_1_EnableInt();
   SPIM_1_Start(SPIM_1_SPIM_MODE_0 | SPIM_1_SPIM_MSB_FIRST);
  while(1) {
      if(UART_1_bCmdCheck()) {                  // Wait for command
        if(strPtr = UART_1_szGetParam()) {      // More than delimiter?
          UART_1_CPutString("Found valid command¥r¥nCommand =>");
          UART_1_PutString(strPtr);
       if(strPtr[0] == 'Z') break;
       UART_1_CPutString("<¥r¥nParamaters:¥r¥n");
          while(strPtr = UART_1_szGetParam()) { // loop on each parameter
            UART_1_CPutString("   <");
            UART_1_PutString(strPtr);         // Print each parameter
            UART_1_CPutString(">¥r¥n");
          }
        }
      UART_1_CmdReset();                        // Reset command buffer
      }
  }
  init_scp1000(); // #2, Initialize SCP1000, activate
  // 'high resolution' measurement mode
  UART_1_CPutString("init_scp1000() done");
  for(;;){
    while(!(PRT1DR & 0x80)) wait_ms(1);// DRDY
    temp = Read_Direct_Access_SPI(0x21, 0x02);// Read 2 bytes from TEMPOUT (0x21)
    temp = temp & 0x3fff; // Mask bits [13:0]
    temp = temp * 100 / 20; // Convert temperature to [□‹C] by dividing
    // the result by 20
    pressure = (unsigned long)((0x0007) & Read_Direct_Access_SPI(0x1F, 1));
    // Read 1 byte from DATARD8 (0x1F) and
    // mask bits [2:0] in order to get
    // 3MSB bits for pressure information
    pressure <<= 16; // shift 3MSB bits to left by 16
    pressure |= (unsigned long)Read_Direct_Access_SPI(0x20, 2);
    // Read 2 bytes from DATARD16 (0x20) in
    // order to get 16 LSB bits for pressure
    // information.
    pressure *= 100;
    pressure >>= 2; // Convert to [Pa] by dividing the 19 bit
    // pressure by 4 --> the 19 bit pattern
    // is shifted to right by 2
    // <- Here user can add temperature and pressure data send / display routines
    // as needed for application. Notice that temperature is in 2's complement format.
    UART_1_PutSHexInt((int)temp);
```

```
    UART_1_PutSHexInt((int)(pressure >> 16));
    UART_1_PutSHexInt((int)(pressure & 0xffff));UART_1_PutChar('¥r');
  }
  return 0;
}
```

7　SPIM と割り込みを利用した場合

Pinout の SPIM_DRDY を StdCPU,High Z, ChangeFromRead に変更する。

boot.tpl の中の

```
   org   1Ch                    ;GPIO Interrupt Vector
    `@INTERRUPT_7`
    reti
```

を

```
    org   1Ch                    ;GPIO Interrupt Vector
    ljmp _GPIO_ISR
    reti
```

に変更し，

さらに，割り込み用関数。

```
volatile char gate = 0;
#pragma interrupt_handler GPIO_ISR
void GPIO_ISR(void) {
  gate++;
}
```

を追加する。

割り込みによりDRDYピンを監視するため，ピンの設定を，DisableIntからChangeFromRead に変更する。全体のプログラムは以下の通りである。

```
#include <m8c.h>
#include "PSoCAPI.h"
#include <stdlib.h>
#include <math.h>
extern BYTE Port_0_Data_SHADE;
extern BYTE Port_1_Data_SHADE;
volatile char gate = 0;
// UART SysClk,VC1,PWM_1
// VC1=SysClk/N --> VC1=24MHz/12=2MHz
// 24MHz*2/(9600*8)=625
// 24MHz*2/(19200*8)=313
// 24MHz*2/(38400*8)=156
// 24MHz*2/(57600*8)=104
// 24MHz*2/(115200*8)=52
// 24MHz*2/(230400*8)=26
// 24MHz*2/(460800*8)=13
//  9600 bps PWM16_1_WritePeriod(625-1);PWM16_1_WritePulseWidth(312);
// 19200 bps PWM16_1_WritePeriod(313-1);PWM16_1_WritePulseWidth(157);
// 38400 bps PWM16_1_WritePeriod(156-1);PWM16_1_WritePulseWidth(78);
// 57600 bps PWM16_1_WritePeriod(104-1);PWM16_1_WritePulseWidth(52);
//115200 bps PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26);
//230400 bps PWM16_1_WritePeriod(26-1);PWM16_1_WritePulseWidth(13);
//460800 bps PWM16_1_WritePeriod(13-1);PWM16_1_WritePulseWidth(7);
```

```
//For SPIM_1/////////////
// CSB P1[5] 01,02,04,08,16,32,64,128
// CSB P1[5] 01,02,04,08,10,20,40, 80
//          0  1  2  3  4  5  6  7
// MISO,MOSI,SCLK, CSB,RXin,DRDY
// 0x04,0x08,0x10,0x20,0x40,0x80
// SPI have to set less than 500kHz
// 24MHz/12(VC1)/4(VC2)->500kHz
// P10,P11 for Miniprog
// Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
// Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
// Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
// Port_1_Data_SHADE |=  0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='H'
// Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
// Port_1_Data_SHADE |=  0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='H'
// Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
// Port_0_Data_SHADE |=  0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='H'
//  DRDY=0;if(PRT1DR & 0x80) DRDY=1;
//  MISO=0;if(PRT1DR & 0x04) MISO=1;
/*
232 TTL Converter
Jumper J1 2-3 short VCC supply
Jumper J2 short from USB to Vcc supply
232Module---PSoC
TXD(1)------ P16(18)RX_in
RXD(5)------ P27(5)TXout
GND(24)
GND(7)
VCC(21)
VCC(15)

Pin assignment
//FOR UART
TX_out,,Strong, DisableInt
RX_in,,High Z, DisableInt

//FOR SPIM_1
SPIM_SCLK,,Strong,DisableInt
SPIM_MOSI,,Strong,DisableInt
SPIM_MISO,,High Z,DisableInt

// FOR sensor handshake for SPIM_1
SPIM_CSB,StdCPU,Strong,DisableInt
SPIM_TRIG,StdCPU,Strong,DisableInt
SPIM_DRDY,StdCPU,High Z,ChangeFromRead

*/
/* For PSoC Writer
PSOC ---MiniProg
Vdd(28)---1
GND(14)---2
XRES(19)---3
P11(13)---4
P10(15)---5
*/
/*
PSOC -- SCP1000
P00(24)-->1(Trig)
P17(10)<--2(DRDY)
P14(17)-->3(SCLK)
GND(14)-->4(GND)
P13(12)-->5(MOSI)
P12(16)<--6(MISO)
P15(11)-->7(CSB)
Vcc(28)-->8(Vcc)
*/
/* Data type
unsigned char 1 byte
unsigned short 2 byte  ***** !!!
unsigned int 2 byte
unsigned long 4 byte
*/
```

```
void wait_ms(unsigned int ms){
  int i,j;
  if(ms > 12) {
    i = (int)(ms / 12);
    for(j = 0;j < i;j++) {
      LCD_1_Delay50uTimes(240);
      ms -= 12;
    }
  }
  LCD_1_Delay50uTimes(ms * 20);
}
void device_init(void){
 Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
 Port_1_Data_SHADE &= ~0x10;PRT1DR = Port_1_Data_SHADE;//set SCK='L'
 Port_1_Data_SHADE &= ~0x08;PRT1DR = Port_1_Data_SHADE;//set MOSI='L'
 Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
}
void fail(int i){
  int j=0;
  for(j = 0;j < 100;j++) {UART_1_CPutString("Sensor Fail");UART_1_PutChar('0'+i);}
}
unsigned int Read_Direct_Access_SPI(unsigned char address, int number_of_bytes){
  unsigned int value;
  value = 0;
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_1_SendTxData(address);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  SPIM_1_SendTxData(0xff);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  value = (unsigned int)(SPIM_1_bReadRxData());
  if(number_of_bytes > 1) {
    value <<= 8;
    SPIM_1_SendTxData(0xff);
    while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
      value |= (unsigned int)(SPIM_1_bReadRxData());
  }
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  return value;
}
void Write_Direct_Access_SPI(unsigned char address, unsigned char data){
  address = (address << 2); // #1, shift the SCP1000 reg address to left by 2
  address |= 0x02; // #2, set write bit to one (RW=1)
  Port_1_Data_SHADE &= ~0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='L'
  SPIM_1_SendTxData(address);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  SPIM_1_SendTxData(data);
  while(!(SPIM_1_bReadStatus() & SPIM_1_SPIM_SPI_COMPLETE)){};
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
}
void Write_Indirect_Access_SPI(unsigned char address, unsigned char data){
  Write_Direct_Access_SPI(0x02, address); // #1, write reg address to ADDPTR (0x02)
  Write_Direct_Access_SPI(0x01, data); // #2, write reg data to DATAWR (0x01)
  Write_Direct_Access_SPI(0x03, 0x02); // #3, write 0x02 to OPERATION (0x03)
  wait_ms(50); // #4, wait 50ms
}
unsigned int Read_Indirect_Access_SPI(unsigned char address){
  unsigned int value;
  Write_Direct_Access_SPI(0x02, address); // #1, write reg address to ADDPTR (0x02)
  Write_Direct_Access_SPI(0x03, 0x01); // #2, write 0x01 to OPERATION (0x03)
  wait_ms(10); // #3, wait 10ms
  value = Read_Direct_Access_SPI(0x20, 0x02); // #4, read two bytes from DATARD16 (0x20)
  return value;
}
void init_scp1000(void){
  unsigned int DATA;
  int i;
  Port_1_Data_SHADE |=  0x20;PRT1DR = Port_1_Data_SHADE;//set CSB='H'
  wait_ms(60); // #3, 60ms wait
  wait_ms(90); // #3, 90ms wait
  Write_Direct_Access_SPI(0x06,0x01); // ASIC software reset
  wait_ms(10); // #3, 10ms wait
```

```
  wait_ms(60); // #3, 60ms wait
  for(i = 6; i > 0 ; i--){
    DATA = Read_Direct_Access_SPI(0x07, 0x01);// #4, read STATUS register --> LSB '0'=OK
    if(!(DATA & 0x0001)) break;
    wait_ms(10);
  }
  if(i == 0) fail(0);
  DATA = Read_Direct_Access_SPI(0x1F, 1); // #5, read DATARD8 register --> LSB '1'=OK
  if(!(DATA && 0x0001)) fail(1);
  Write_Indirect_Access_SPI(0x2D, 0x03); // #6, Low noise configuration
  wait_ms(100); // #7, wait for 100ms
  Write_Direct_Access_SPI(0x03, 0x00); // Reset operation mode
  wait_ms(10); // Wait before change new mode
//Write_Direct_Access_SPI(0x03, 0x0A); // #9, set SCP1000 into 'high resolution'
  Write_Direct_Access_SPI(0x03, 0x09); // #9, set SCP1000 into 'high speed'
  // measurement mode (0x0A)
  wait_ms(100);
}
#pragma interrupt_handler GPIO_ISR
void GPIO_ISR(void) {
  gate++;
}
int main(void){
  char * strPtr; // Parameter pointer
  unsigned int temp;
  unsigned long pressure;
  Port_0_Data_SHADE &= ~0x01;PRT0DR = Port_0_Data_SHADE;//set TRIG='L'
  Port_1_Data_SHADE &= ~0x80;PRT1DR = Port_1_Data_SHADE;//set DRDY='L'
  device_init();

  UART_1_CmdReset();
  UART_1_IntCntl(UART_1_ENABLE_RX_INT);
  PWM16_1_WritePeriod(624);PWM16_1_WritePulseWidth(312);

   PWM16_1_DisableInt();
   PWM16_1_Start();
  UART_1_Start(UART_1_PARITY_NONE);
  M8C_EnableIntMask(INT_MSK0, INT_MSK0_GPIO);
  M8C_EnableGInt;
  while(1) {
      if(UART_1_bCmdCheck()) {                  // Wait for command
        if(strPtr = UART_1_szGetParam()) {      // More than delimiter?
          UART_1_CPutString("Found valid command¥r¥nCommand =>");
          UART_1_PutString(strPtr);
       if(strPtr[0] == 'Z') break;
       UART_1_CPutString("<¥r¥nParamaters:¥r¥n");
          while(strPtr = UART_1_szGetParam()) { // loop on each parameter
            UART_1_CPutString("   <");
            UART_1_PutString(strPtr);         // Print each parameter
            UART_1_CPutString(">¥r¥n");
          }
        }
      UART_1_CmdReset();                        // Reset command buffer
      }
  }
   SPIM_1_EnableInt();
   SPIM_1_Start(SPIM_1_SPIM_MODE_0 | SPIM_1_SPIM_MSB_FIRST);
  init_scp1000(); // #2, Initialize SCP1000, activate
  gate = 1;
  while(!(PRT1DR & 0x80));// DRDY
  for(;;){
    if(gate) {
//     while(!(PRT1DR & 0x80));// DRDY
      temp = Read_Direct_Access_SPI(0x21, 0x02);// Read 2 bytes from TEMPOUT (0x21)
      temp = temp & 0x3fff; // Mask bits [13:0]
      temp = temp * 100 / 20; // Convert temperature to [□‹C] by dividing
      pressure = (unsigned long)((0x0007) & Read_Direct_Access_SPI(0x1F, 1));
      pressure <<= 16; // shift 3MSB bits to left by 16
      pressure |= (unsigned long)Read_Direct_Access_SPI(0x20, 2);
      pressure *= 100;
      pressure >>= 2; // Convert to [Pa] by dividing the 19 bit
```

```
        gate = 0;
      }
      UART_1_PutSHexInt((int)temp);
      UART_1_PutSHexInt((int)(pressure >> 16));
      UART_1_PutSHexInt((int)(pressure & 0xffff));UART_1_PutChar('¥r');

   }
   return 0;
}
```

８　MATLAB による取り込み

出力データフォーマットは,

16 進数 4 桁分（温度）16 進数 8 桁分（気圧）¥r

で終了しているので以下のようなスクリプトを用いる。このスクリプトを動かすには,

rs323cj2.mexw32 が必要である。

また，COM 番号や，ボーレートは合わせる必要がある。

上記でしめした PSoC のプログラムでは，Z キーの入力をしないと，データの取り込みが始まらない。そこで，まず，Teraterm を起動し，Z を入力後，リターンをし，データの取り込みが始まったのを確認後，Teraterm を終了し，その後で，MATLAB から取り込む。

```
clear all;close all
rs232cj2('COM8',9600,8,0,1,0)
data=[];
temp=[];
pres=[];
ttime=[];
rs232cj2([])
figure(1)
tic;
for i=1:1000
    data=[];
    while(1)
       data=[data,char(rs232cj2([]))];
       ind=findstr(data,13);
       if(length(ind)>2)
          data=data(ind(1):ind(end));
          ind=findstr(data,13);
          ddata=[];
          for j=1:length(ind)-1
              ddata=[ddata;data((ind(j)+1):(ind(j+1)-1))];
          end
          break;
        end
    end
    ttime=[ttime;ones(length(ddata(:,1)),1)*toc];
    temp=[temp;hex2dec(ddata(:,1:4))/100];
    pres=[pres;hex2dec(ddata(:,5:12))/10000];
    subplot(2,1,1);plot(ttime,temp);
    subplot(2,1,2);plot(ttime,pres);
    drawnow;
 end
```