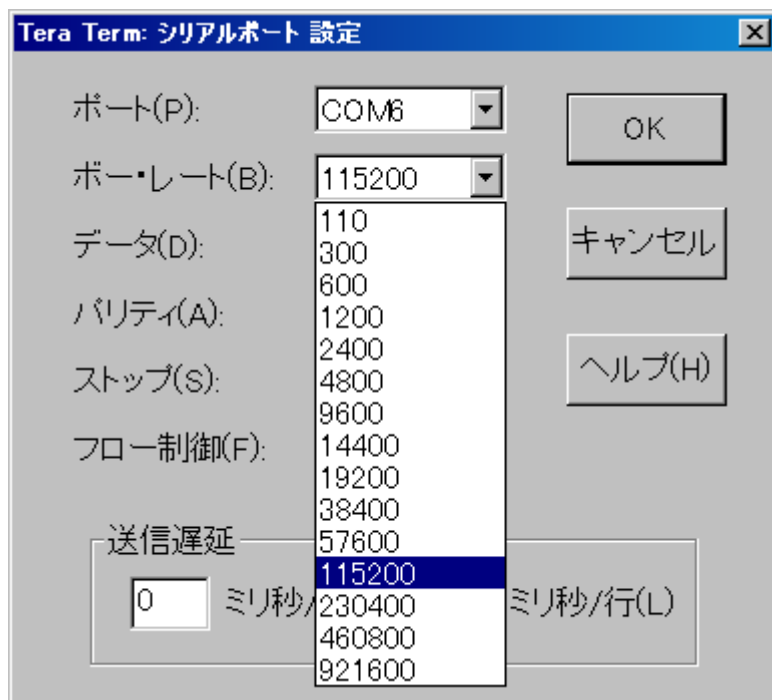


PSoC (49466 単体) で設定できるシリアル通信の最大クロックは？

PSoC の UART のマニュアルによれば、6Mbits/second まで設定できると書いてある。しかし、実際には、CPU の速度や、供給する Clock によりその制約通りとは行かない。ここでは、外部クロックは、接続しない標準的な状態、かつ、C 言語でのプログラミングで、どこまで通信レートを上げられるかを実験的に検証してみる。シリアル通信における標準的なボーレートは以下の通りである。



まずは、24MHz の $\text{sysclock} * 2$ の場合に設定できる baudrate の計算スクリプトを示す。ここでは、 $\pm 2\%$ の周波数誤差を想定し、計算した。

```
>> baudrate=[9600,19200,38400,57600,115200,230400,460800,921600];
clk=2*24*10^6;%MHz
err=0.02;
format bank
low=(clk./(8*baudrate*(1+err)));
high=(clk./(8*baudrate*(1-err)));
est=clk./8./round((low+high)/2);
[baudrate;low;high;est; round((low+high)/2);(baudrate-est)./baudrate*100]'
ans =
```

9600.00	612.75	637.76	9600.00	625.00	0
19200.00	306.37	318.88	19169.33	313.00	0.16
38400.00	153.19	159.44	38461.54	156.00	-0.16
57600.00	102.12	106.29	57692.31	104.00	-0.16
115200.00	51.06	53.15	115384.62	52.00	-0.16
230400.00	25.53	26.57	230769.23	26.00	-0.16
460800.00	12.77	13.29	461538.46	13.00	-0.16
921600.00	6.38	6.64	857142.86	7.00	6.99

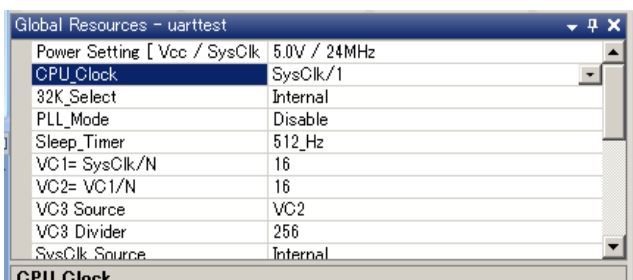
となる。つまり、この設定なら計算上最大で、460800bps まで設定できることがわかる。

実際のデザイン

UART の通信を行う際に、VC1,VC2,VC3 を使うと、AD 変換など他のブロックのデザインに影響を与えやすくなる。そこで、ここでは、UART のクロックは、PWM16 ブロックを配置しそこで設定を行う。このようにすることで、デジタルブロックは、PWM16 と UART で計 4 つのブロックを消費してしまうが、その後のデザインの自由度を増すことができる。

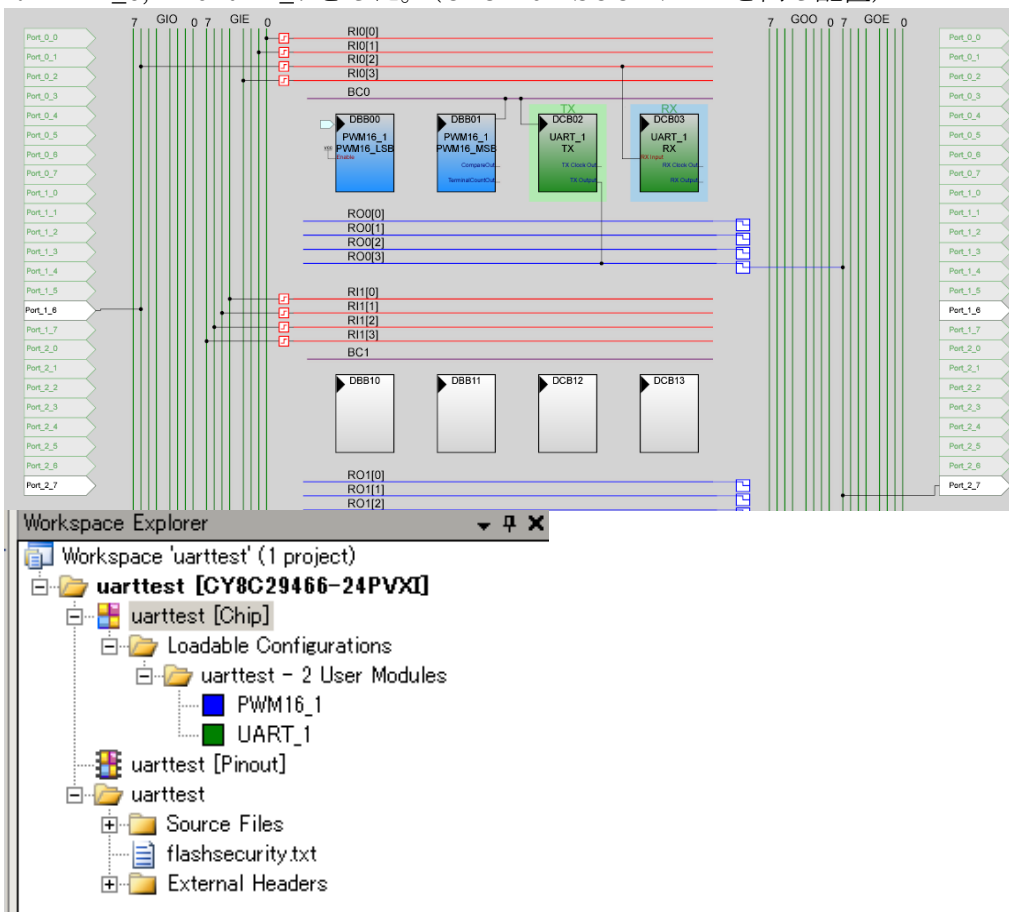
Global Resources の設定

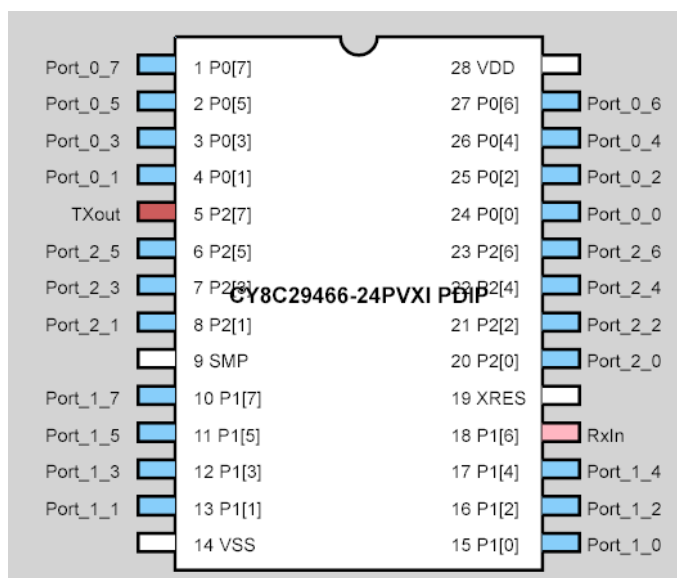
極力 CPU の処理能力を上げるため、CPU_Clock は、SysClk/1 とした。その他の変更点は無い。



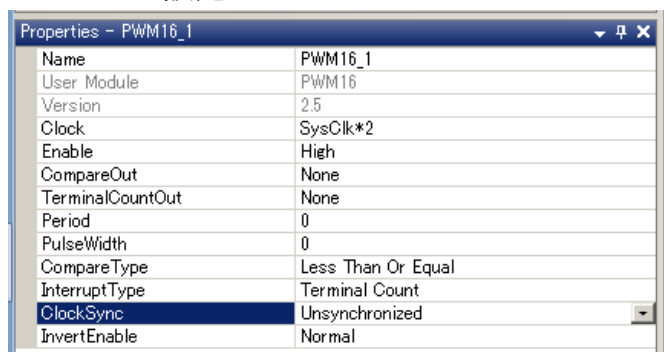
ブロックの配置

UART へのクロックは、PWM16 から供給するため UART は、横に並べて配置している。RXin P1_6, TXout P2_7 とした。(CY3210-PSOCEVAL1 と同じ配置)





PWM16 の設定



Clock は、VC1,VC2,VC3 からではなく、SysClk*2 を選択する。PWM の出力は、明示的には出していないが、BC0 の線から DBB01 を選択し、クロックを出力するようにしている。また、ClockSync は、他の設定ではワーニングが出るため、Unsyncronized としている。Period と、PulseWidth の設定は、プログラム上で、以下の設定をすればよい。

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。

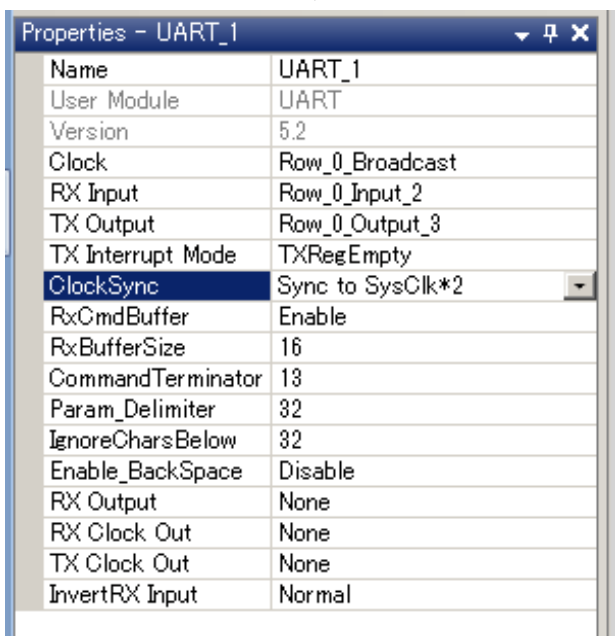
bps	計算式	PWM16 Period	PWM16 PulseWidth
9600	$24 \cdot 10^6 \cdot 2 / 9600 / 8$	625	312
19200	$24 \cdot 10^6 \cdot 2 / 19200 / 8$	313	156
38400	$24 \cdot 10^6 \cdot 2 / 38400 / 8$	156	78
57600	$24 \cdot 10^6 \cdot 2 / 57600 / 8$	104	52
115200	$24 \cdot 10^6 \cdot 2 / 115200 / 8$	52	26

プログラム上で設定するには、以下のようにする。Period は、-1 する点に注意すること。

```
//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);//57600 bps
//PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26);//115200 bps
```

UART

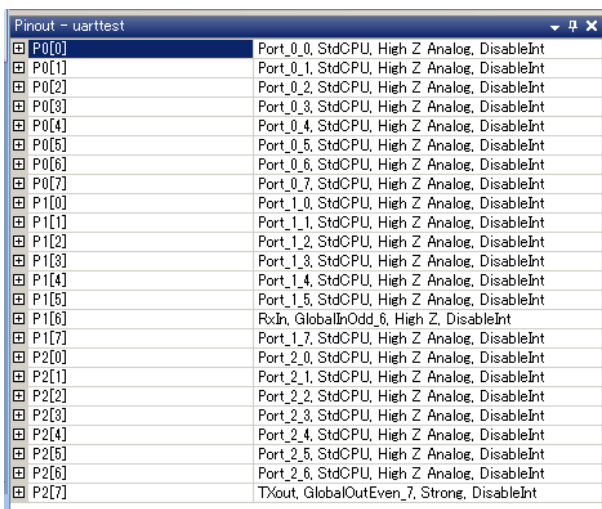
Clock は, Row_0_Broadcast を経由とし, RXinput, TXoutput は, それぞれ, P1_6, P2_7 に接続できるように設定した。ClockSync は, Sync to SysClk*2 とし, 同期ができるように設定した。残りの点は, ほぼデフォルトのままの設定である。



Name	Value
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*2
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

Pinout

使用するピンは, RxIn, TXout だけであり, 入力は, HighZ, 出力は, Strong としている。



Pin	Configuration
P0[0]	Port_0_0, StdCPU, High Z Analog, DisableInt
P0[1]	Port_0_1, StdCPU, High Z Analog, DisableInt
P0[2]	Port_0_2, StdCPU, High Z Analog, DisableInt
P0[3]	Port_0_3, StdCPU, High Z Analog, DisableInt
P0[4]	Port_0_4, StdCPU, High Z Analog, DisableInt
P0[5]	Port_0_5, StdCPU, High Z Analog, DisableInt
P0[6]	Port_0_6, StdCPU, High Z Analog, DisableInt
P0[7]	Port_0_7, StdCPU, High Z Analog, DisableInt
P1[0]	Port_1_0, StdCPU, High Z Analog, DisableInt
P1[1]	Port_1_1, StdCPU, High Z Analog, DisableInt
P1[2]	Port_1_2, StdCPU, High Z Analog, DisableInt
P1[3]	Port_1_3, StdCPU, High Z Analog, DisableInt
P1[4]	Port_1_4, StdCPU, High Z Analog, DisableInt
P1[5]	Port_1_5, StdCPU, High Z Analog, DisableInt
P1[6]	RxIn, GlobalInOdd_6, High Z, DisableInt
P1[7]	Port_1_7, StdCPU, High Z Analog, DisableInt
P2[0]	Port_2_0, StdCPU, High Z Analog, DisableInt
P2[1]	Port_2_1, StdCPU, High Z Analog, DisableInt
P2[2]	Port_2_2, StdCPU, High Z Analog, DisableInt
P2[3]	Port_2_3, StdCPU, High Z Analog, DisableInt
P2[4]	Port_2_4, StdCPU, High Z Analog, DisableInt
P2[5]	Port_2_5, StdCPU, High Z Analog, DisableInt
P2[6]	Port_2_6, StdCPU, High Z Analog, DisableInt
P2[7]	TXout, GlobalOutEven_7, Strong, DisableInt

main.c

プログラムは, 比較を容易にするため, サンプルプログラムをほぼそのまま利用してみた。

```
#include <m8c.h>
#include "PSoCAPI.h"
//P16 Rx
//P27 Tx
void main(void)
{
    char * strPtr; // Parameter pointer
    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);//57600 bps
// PWM16_1_WritePeriod(52-1);PWM16_1_WritePulseWidth(26);//115200 bps
// PWM16_1_WritePeriod(26-1);PWM16_1_WritePulseWidth(13);//230400 bps x
PWM16_1_Start();
UART_1_CmdReset(); // Initialize receiver/cmd
UART_1_IntCnt1(UART_1_ENABLE_RX_INT); // Enable RX interrupts
UART_1_Start(UART_1_PARITY_NONE); // Enable UART

M8C_EnableGInt; // Turn on interrupts
UART_1_CPutString("¥r¥nWelcome to PSoC UART test program. V1.1 ¥r¥n");
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); // Print out command
            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
    }
}
}

```

通信結果

実際に文字が送受信できるかのテストを行った。それらの結果を以下に示す。

○	9600.00	612.75	637.76	9600.00	625.00	0
○	19200.00	306.37	318.88	19169.33	313.00	0.16
○	38400.00	153.19	159.44	38461.54	156.00	-0.16
○	57600.00	102.12	106.29	57692.31	104.00	-0.16
○	115200.00	51.06	53.15	115384.62	52.00	-0.16
×	230400.00	25.53	26.57	230769.23	26.00	-0.16
×	460800.00	12.77	13.29	461538.46	13.00	-0.16

通信の同期という意味では、460800bps まで扱うことができるはずであるが、実際には、115200bps までは、文字化けせず通信できることがわかった。