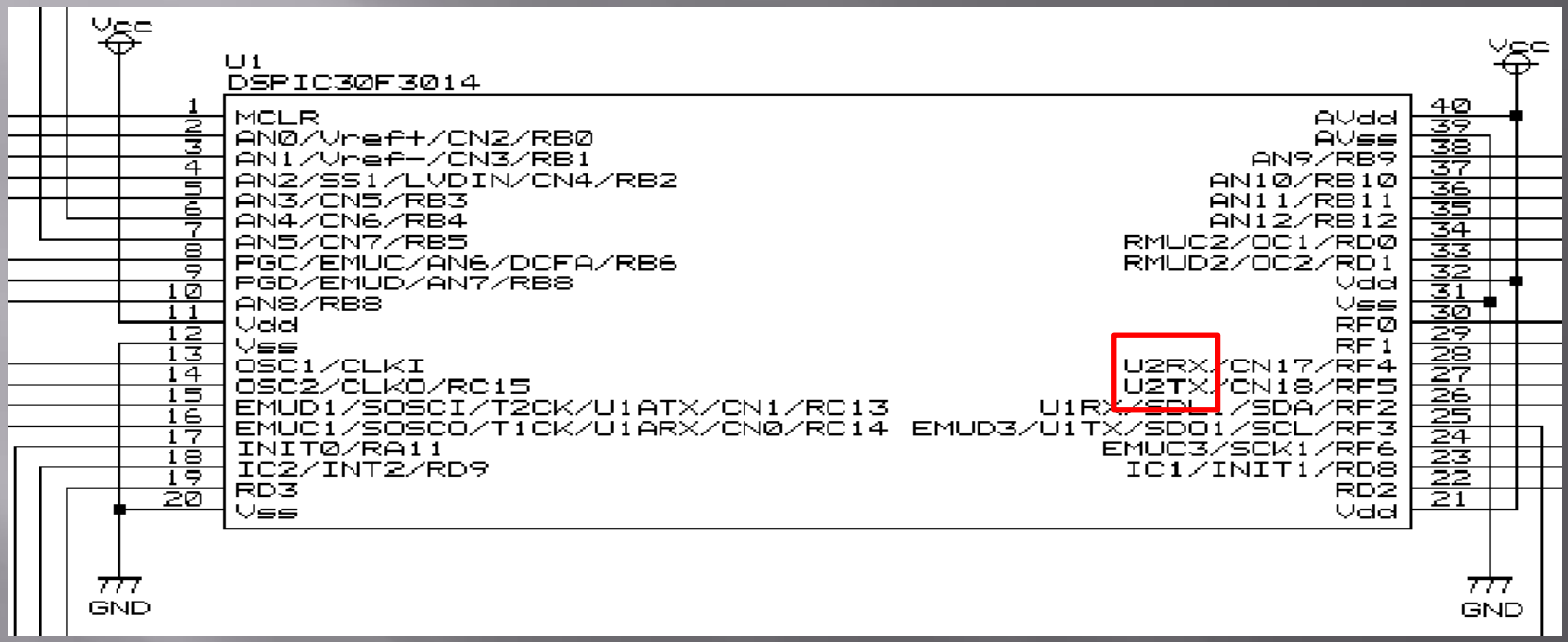
XBeeについて

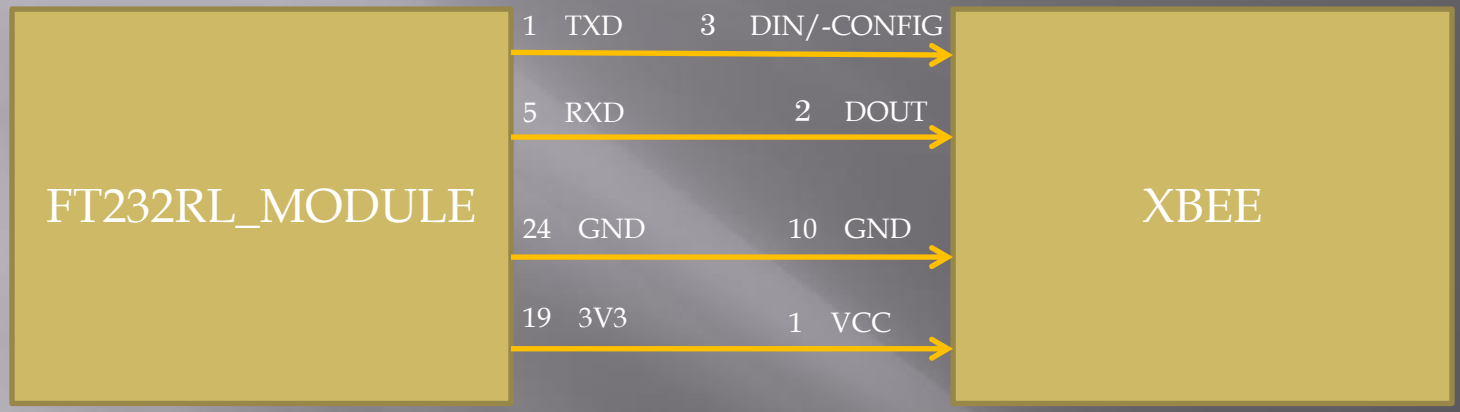
通信方式はシリアル通信なのでdsPICのUARTに直結することができます。



dsPIC側のXBeeにはUART２の端子を使用します。
RXが受信
TXが送信

XBee ← PC 用の通信回路

PCからXBeeへの接続方法はUSBで接続するだけで済むようにします。
そのために今回はFT232RL__MODULEという変換モジュールを使用します。
これはUSBの信号をXBeeに接続できるTLLレベルに変換してくれるものです。



これだけの接続でPCとXBeeの通信が
USBのみで可能になりました

XBeeの初期化設定

パソコン側のXBee
- ボーレート　115200bps
- FunctionSet　ZIGBEECOORDINATOR_AT
- PANID　500
- DH　13A200
- DL　406E8886

PIC側のXBee
- ボーレート　115200bps
- FunctionSet　ZIGBEEROUTER_AT
- PANID　500
- DH　13A200
- DL　40625D41

UARTモジュールについて

dsPICにはPCとシリアル通信するためにUARTという内部回路が組み込まれています。

UARTは単位時間当たり1bitずつ送られてくるシリアルデータをその都度シフトして適当なパラレルデータに変換する回路です。

プログラム解説

```
#ifndef MAEKAWA_H
#define MAEKAWA_H

#include "p30F3014.h"
#include "outcompare.h"
#include "timer.h"
#include "adc12.h"
#include "uart.h"
#include "string.h"
//dsPICF3014はuart.hで定義されないので
#define UART_RX_TX            0xFBE7
#define UART_ALTRX_ALTTX      0xFFE7
//LCD用に定義
#define LCD_RS LATBbits.LATB5                    //RS端子はポートBの5番ピンに繋がっ
ている
#define LCD_E LATBbits.LATB4                     //E端子はポートBの4番ピンに繋がって
いる
#define LCD_D4 LATBbits.LATB9
#define LCD_D5 LATBbits.LATB10
#define LCD_D6 LATBbits.LATB11
#define LCD_D7 LATBbits.LATB12
//プロトタイプ関数宣言
void machi_usec(int usec);
void machi_msec(const int msec);
void lcd_data(const unsigned char asci);
void lcd_clear(void);
void LcdInitFunc(void);
void lcd_puts(unsigned char *string);
void lcd_convert(const long number);
void motor(float Duty,const short MotorType);
void MotorInitFunc(void);
void SensorInitFunc(void);
unsigned int GetSensorValue(const short ADChannel);
void DeviceInitFunc(void);
void ServoInitFunc(void);
void servo(int angle);
void SendLineBreak(const short UartChannel);
void SendSignal(const short UartChannel,unsigned int *string);
void WaitSendSignal(const short UartChannel);
void WaitGetSignal(const short UartChannel);
char *GetStrData(const short UartChannel);
void UartInitFunc(void);
・・・
#endif
```

C30コンパイラは
そのままではUARTピンが定義されて
いないというバグがあります。

新しく追加したUART通信用関数郡

```
#define CLOCK  80
#define MAX_UARTWORD 256
/*PWMのピンを定義*/
#define PWM_1A_AND_2A LATDbits.LATD1
#define PWM_1B LATFbits.LATF0
#define PWM_2B LATFbits.LATF1
#define PWM_3A LATDbits.LATD2
#define PWM_4A LATFbits.LATF6
/*サーボのピンを定義*/
#define SERVO LATBbits.LATB3
/*モータの正回転、逆回転を定義*/
enum MotionType{normal,reverse};
/*モータの種類を定義*/
enum MotorType{motor1,motor2};
enum ADChannel{ch0,ch1,ch2};
enum UartChannel{uart1,uart2};
extern volatile int ServoTargetValue;
#endif                    //MAEKAWA_H
```

新しく追加した
uartの番号指定用定数

改行データ送信用関数

```
void SendLineBreak(const short UartChannel){
        SendSignal(UartChannel,(unsigned int *)"¥r");
}
```

UARTにデータを送信するための関数
これについて説明します。

UARTデータ送信用関数

```
void SendSignal(const short UartChannel,unsigned int *string){
        switch(UartChannel){
                case uart1:
                        putsUART1(string);
                break;
                case uart2:
                        putsUART2(string);
                break;
        }
        WaitSendSignal(UartChannel);
}
```

UARTに文字列を
送信する

送信完了待ちの関数
これについて説明します。

UART送信完了待ち関数

```
void WaitSendSignal(const short UartChannel){
        switch(UartChannel){
                case uart1:
                        while(BusyUART1());
                break;
                case uart2:
                        while(BusyUART2());
                break;
        }
}
```

BusyUARTx()は
送信中　1
送信完了　0
を返す関数

UARTの受信完了待ち関数

```
void WaitGetSignal(const short UartChannel){
        switch(UartChannel){
                case uart1:
                        while(!DataRdyUART1());
                break;
                case uart2:
                        while(!DataRdyUART2());
                break;
        }
}
```

DataRdyUARTx()は
受信中　0
受信完了　1
を返す関数

一見これらの送受信完了待ち関数は必要ないようにも思いますが、
標準の関数ではどちらが送信用なのか受信用なのかわかりずらく
処理中と完了時に返す0,1が逆になっていたりと使いずらいため
作成しました。

UARTの文字列受信用関数

```
char *GetStrData(const short UartChannel){
        unsigned char moji;
        int count = 0;
        static char string[MAX_UARTWORD];

        //1文字受信を待つ
        WaitGetSignal(UartChannel);

        switch(UartChannel){
                case uart1:
                        moji = ReadUART1();
                break;
                case uart2:
                        moji = ReadUART2();
                break;
        }

        while((moji != '¥r') && (count < MAX_UARTWORD)){
                string[count] = moji;
                count++;
                WaitGetSignal(UartChannel);
                switch(UartChannel){
                        case uart1:
                                moji = ReadUART1();
                        break;
                        case uart2:
                                moji = ReadUART2();
                        break;
                }

        }
        //null文字挿入
        string[count] = 0x00;

        return string;

}
```

stringのポインタを返すため
当然静的変数として宣言します。

ReadUARTx()は
1文字受信用関数
受信した文字をmojiに代入

受信文字が
改行でない且つ
作成文字列が収容文字配列の容量以下である限り

UART初期化用関数

```
void UartInitFunc(void){
        unsigned int config1 = UART_EN & UART_IDLE_CON & UART_ALTRX_ALTTX & UART_NO_PAR_8BIT &
                        UART_1STOPBIT    & UART_DIS_WAKE & UART_DIS_LOOPBACK & UART_DIS_ABAUD;

        unsigned int config2 = UART_INT_TX_BUF_EMPTY & UART_TX_PIN_NORMAL & UART_TX_ENABLE &
                        UART_INT_RX_CHAR    & UART_ADR_DETECT_DIS & UART_RX_OVERRUN_CLEAR;

        OpenUART1(config1,config2,10);
        OpenUART2(config1,config2,10);          //115kbps

        /*受信割り込み禁止,送信割り込み禁止*/
        ConfigIntUART1(UART_RX_INT_DIS & UART_TX_INT_DIS);
        ConfigIntUART2(UART_RX_INT_DIS & UART_TX_INT_DIS);

}
```

BRG = 10は
ボーレート115kbps
で動作させることになります

OpenUARTx()用のコンフィグです。
別ページで詳しく解説します。

ボーレートの設定方法を説明します。

void OpenUARTx(unsigned int config1,unsigned int config2,
unsigned int ubrg)

UxBRG=FCY/(16・Baud Rate)－1
で計算した値を指定します

FcyとはFOSCを1/2したもの

FOSCは実際の命令周波数です
つまり、クロック生成部分で出力
される周波数の1/2

PICはクロックが２回で1個の命令
が実行されるため

システムクロック80Mhzで115kbpsのボーレートを得たい場合は

UxBRG = 20000 / (16・115) – 1

UxBRG = 0.987

よって引数ubrgには10を渡せば良いことがわかる。

UARTコンフィグの設定の説明に入ります。

```
unsigned int config1 = UART_EN & UART_IDLE_CON &
        UART_RX_TX & UART_NO_PAR_8BIT & UART_1STOPBIT
    & UART_DIS_WAKE & UART_DIS_LOOPBACK & UART_DIS_ABAUD;
```

UART_EN　UARTモジュール有効
UART_DIS　UARTモジュール無効

UART_IDLE_CON　アイドル中動作継続
UART_IDLE_STOP　アイドル中動作停止

UART_ALTRX_ALTTX 代替ピン使用
UART_RX_TX　通常ピン使用

UART_NO_PAR_9BIT　9ビットデータ、パリティなし
UART_ODD_PAR_8BIT　8ビットデータ、奇数パリティ
UART_EVEN_PAR_8BIT　8ビットデータ、偶数パリティ
UART_NO_PAR_8BIT　8ビットデータ、パリティなし

ストップビット長の設定
UART_2STOPBITS　：　2ビット
UART_1STOPBIT　：　1ビット

SLEEP中にスタートビットが来た時の動作
UART_DIS_WAKE　ウェイクアップ有効

UART_EN_LOOPBACK　：　ループバックモード有効
UART_DIS_LOOPBACK　：　ループバックモード無効
※ループバックは送信したデータを自分に戻してテストする機能

ボーレートを
UART_EN_ABAUD　：　自動検出有効
UART_DIS_ABAUD　：　自動検出無効

一応パリティビットと代替ピンについて説明します。

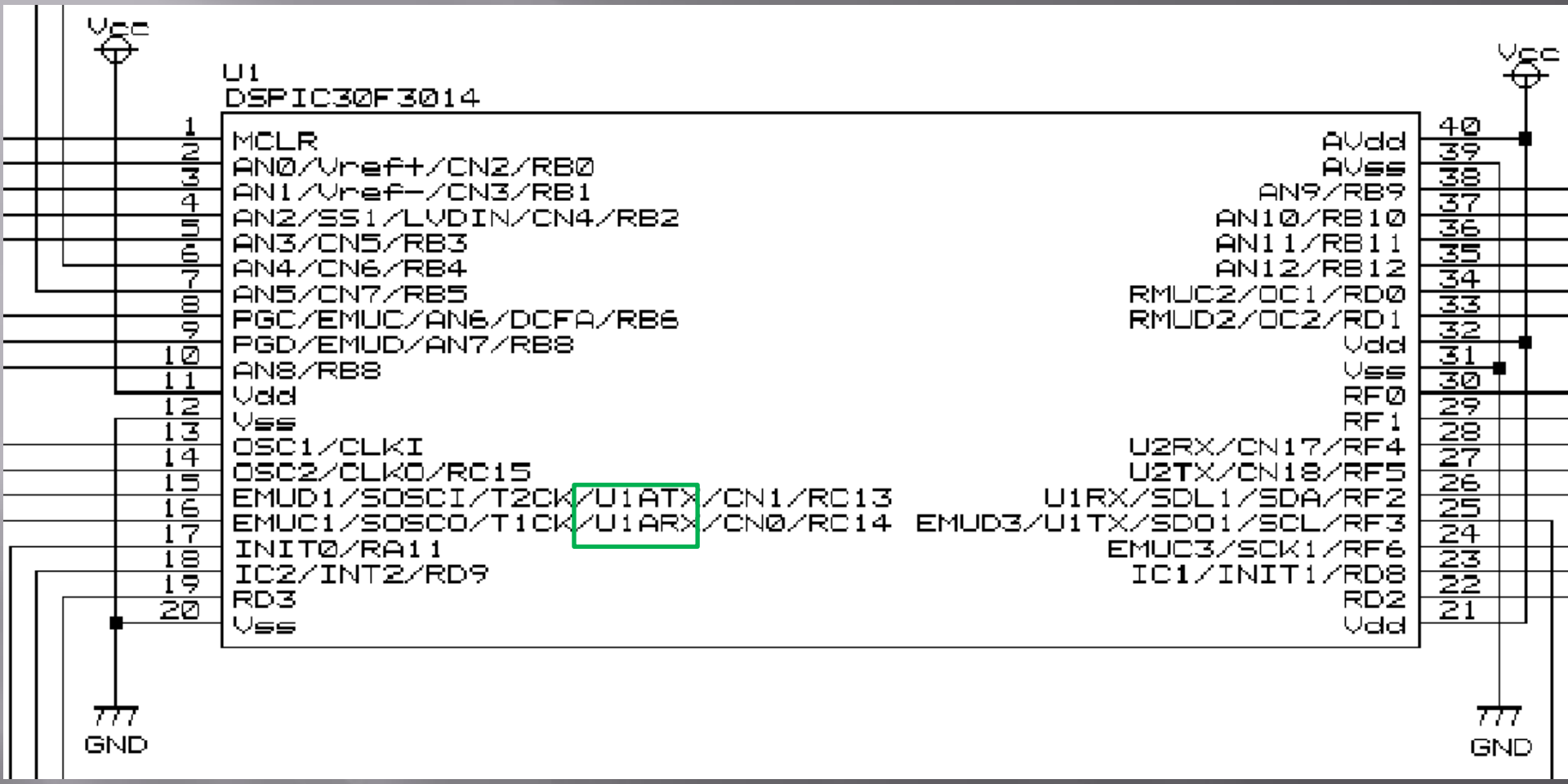
パリティビットとは、データが正常に送られたかを検査するためのビットです。
偶数パリティ、奇数パリティが選択できます。
これは、送信するデータの1の数が偶数または奇数になるようにパリティビットで調整するということです。

（例）
1000 0010　　‘A’というASCIIコードを送信する

偶数パリティ
1000 00100

奇数パリティ
1000 00101

代替ピンは



しかし、UART 1 を使用する際のみ
代替ピンは適用される。

unsigned int config2 = UART_INT_TX_BUF_EMPTY & UART_TX_PIN_NORMAL & UART_TX_ENABLE & UART_INT_RX_CHAR& UART_ADR_DETECT_DIS & UART_RX_OVERRUN_CLEAR;

UART_INT_TX_BUF_EMPTY ： 送信バッファが完全に空になったら割り込む
UART_INT_TX ： 送信バッファに空きがあれば割り込む

UART_TX_PIN_NORMAL ： 通常動作
UART_TX_PIN_LOW ： ブレーク

UART_TX_ENABLE ： 送信有効
UART_TX_DISABLE ： 送信無効

受信割り込みモード設定
文字を受信するたびに割り込む

アドレス文字を検出しない

オーバーランした場合は
ビットクリアする

ブレーク信号は中断信号です。
これを用いて通信を強制的に中断して初期状態に戻したりするような使い方が多いようです。