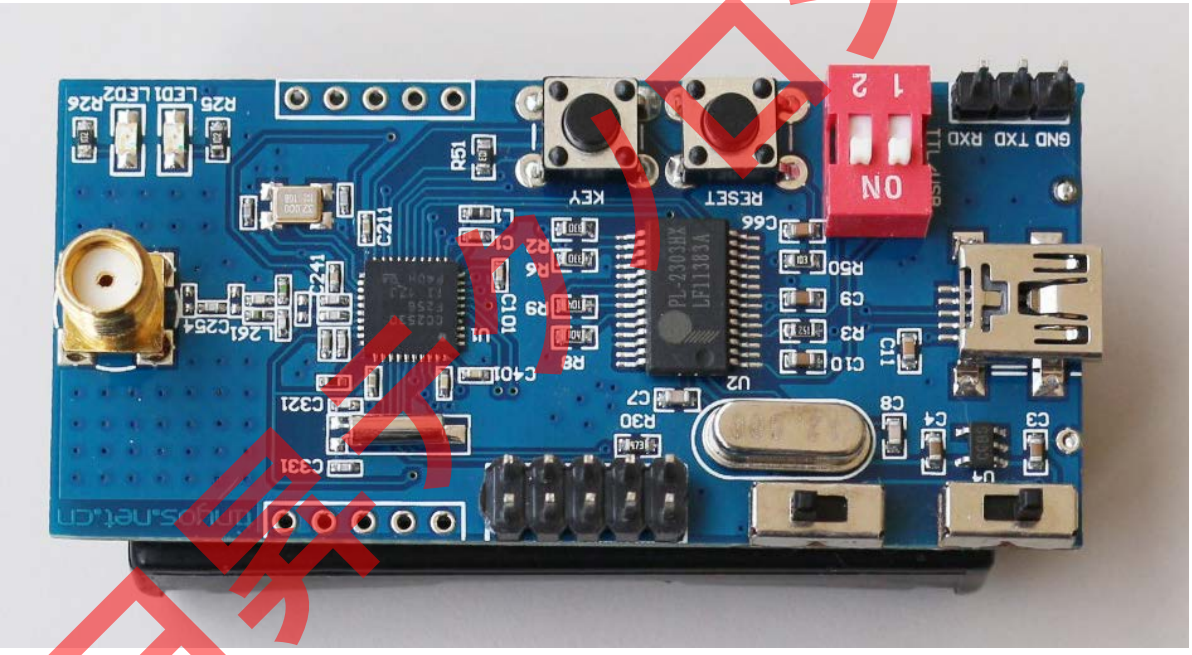
CC2530 zigbee 開発キット
実験マニュアル

株式会社日昇テクノロジー
http://www.csun.co.jp
info@csun.co.jp

作成日：2014/05/29



• 修正履歴

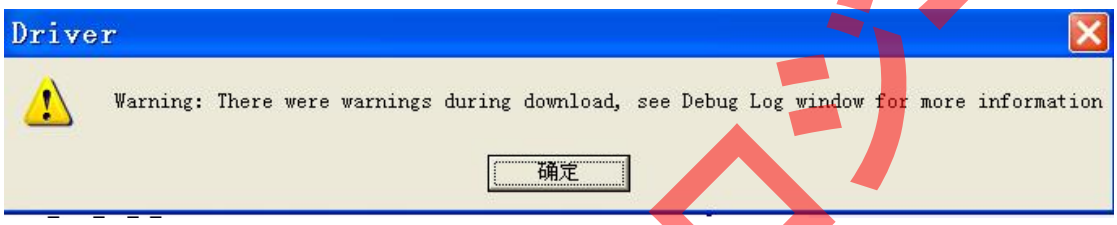
NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2014/5/29

※ この文書の情報は、文書を改善するため、事前の通知なく変更されることがあります。最新版は弊社ホームページからご参照ください。
「http://www.csun.co.jp」

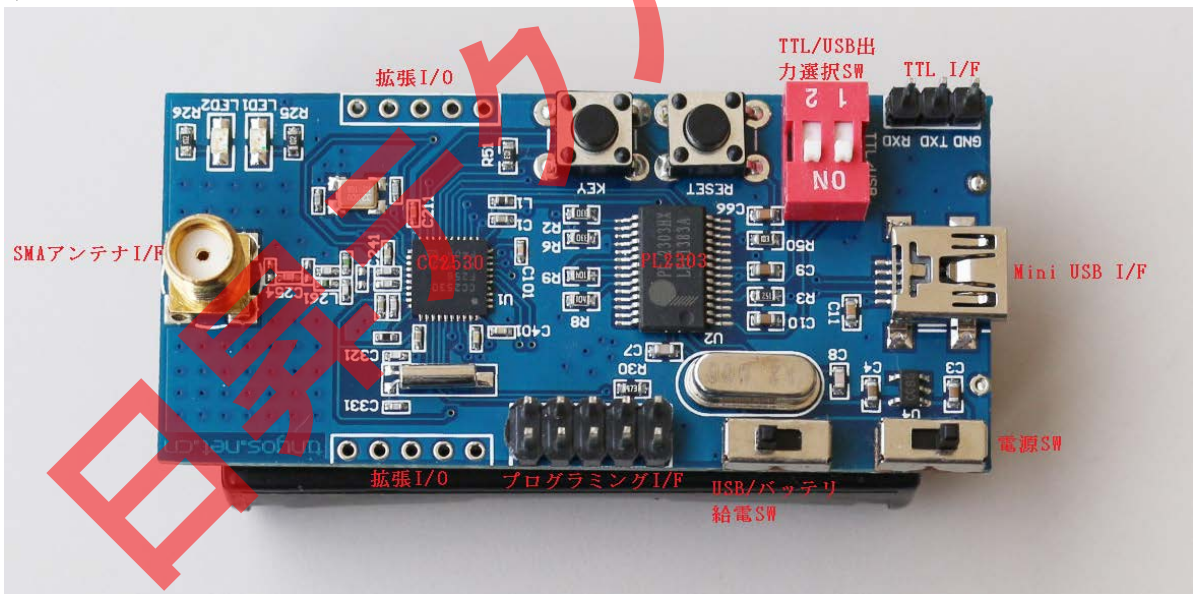
目　次

注意：
本モジュールは初めてプログラムをダウンロードする時、次の提示画面が表示されます。“OK”をクリックしてください。

ダウンロード完了後、次の提示画面が表示する、“確定”をクリックする。



本モジュール



1　IO ポートで LED 自動点滅のコントロール

1.1 実験の目的

CC2530 のデジタル IO ポートの設定と応用を把握する。

1.2 実験の原理

主に I/O ポートからハイ・ローレベル出力して LED の点滅を制御する。利用する主なレジスタは P1 、P1SEL、P1DIR です。その他の IO ポート関連のレジスタは cc2530 マニュアルをご参照下さい。

P1(0x90)_ ポート 1

ビット	名前	リセット	読み/書き	説明
7：0	P1[7：0]	0xFF	R/W	ポート 1、汎用 IO ポート、SFR ビットからアドレスを検索できる。この CPU 内部レジスタは XDATA(0x7090)から読み取りできる、なお書き込むできない。

P1SEL(0xF4) _ ポート 1 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELP1[7：0]	0x00	R/W	P1.7 から P1.0 の機能選択 0：汎用 IO 1：外部デバイス機能

P1DIR(0xFE)_ ポート 1 方向

ビット	名前	リセット	読み/書き	説明
7：0	DIRP1[7：0]	0x00	R/W	P1.7 から P1.0 の I/O 方向の選択 0：入力 1：出力

1.3　実験条件

CC2530　開発ボード*１、ダウンローダー*１、ケーブル*１

1.4　実験内容

I/O ピンで LED1 と LED2 のサイクル点滅を制御する。

ソースコード：

```
void main(void)
{
//発振器は外部 32 MHz の水晶振動子に設定する。
SLEEPCMD&= ~0X04;
CLKCONCMD = 0X10;
while(CLKCONSTA!=0X10);
SLEEPCMD = 0X04;
// P1.1 と P1.0 をデジタル I/O ピンに初期化し、出力モードに設定。
P1SEL = 0x00;
P1DIR = 0x03;
while(1)
{
P1 = 0X01; //LED1 点灯, LED2 消灯
DelayXms(10);
P1 = 0X02; //LED1 消灯, LED2 点灯
```

```
DelayXms(10);
}
}
```

1.5　実験手順

Example¥1_light にあるプロジェクトファイルをオープン。インタフェースは図 1−1 のとおり、  をクリックしてプログラムをコンパイル、  をクリックしてリンク、最後に、  をクリックし開発ボードへダウンロードする。ボードでプログラムを実行し、実験現象を観察する。

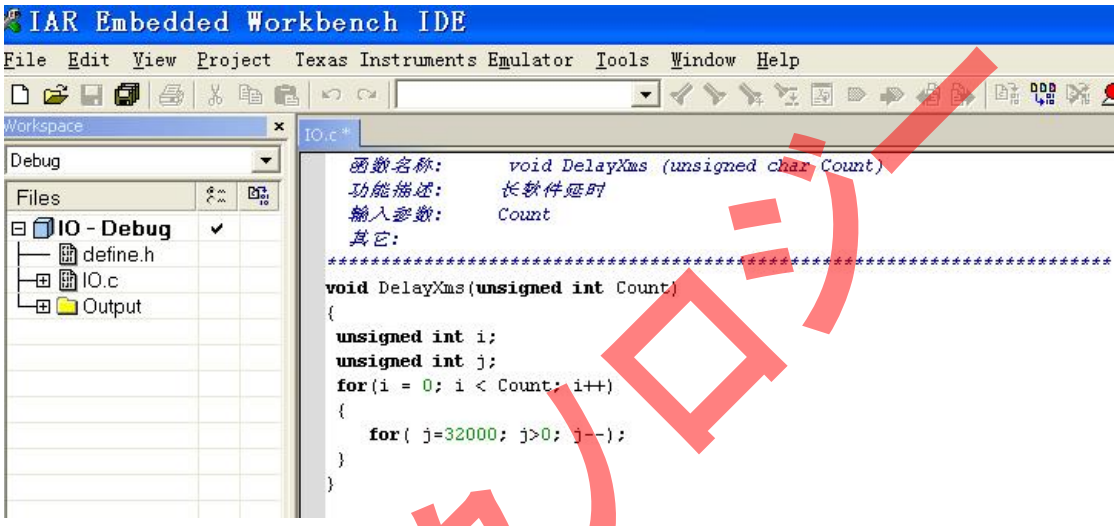


図 1−1

1.6　実験現象

二つの LED はサイクル点滅する。

2、キースキャン

2.1　実験の目的

キースキャン方法と I/O ポートの使用を把握する。

2.2　実験の原理

リクエスト方法で、キーの状態をチェックして、LED で指示する。相応なレジスタの設定でキーインタフェースを入力ポートとして設定する、そして LED 制御ポートは出力に設定する。使用するレジスタは実験１と同じ。

2.3　実験条件

CC2530　開発ボード*１、ダウンローダー*１、ケーブル*１

2.4　実験内容

最初は LED2 を点灯させ、キーを押した場合、LED1 を点灯させ、LED2 を消灯させる。キーを離す場合、LED1 を消灯させ、LED2 を点灯させる。

ソースコード：

```
void main(void)
{
SLEEPCMD&= ~0X04;
CLKCONCMD = 0X10;
while(CLKCONSTA!=0X10);
SLEEPCMD = 0X04;
P1SEL = 0x00;
P1DIR = 0x03;
P0SEL = 0X02;
P0DIR &=~0X02;
while(1)
{
P1 = 0X02;
if(!(P0&0X02))
{
DelayXms(5);
if(!(P0&0X02))
  {
P1 = 0X01;
DelayXms(10);
}
}
}
  }
```

2.5　実験手順

Example¥2_switcht にあるプロジェクトファイルをオープン。インタフェースは図 2－1 のとおり、  をクリックしてプログラムをコンパイル、  をクリックしてリンク、最後に、

 をクリックし開発ボードへダウンロードする。ボードでプログラムを実行し、実験現象を観察する。

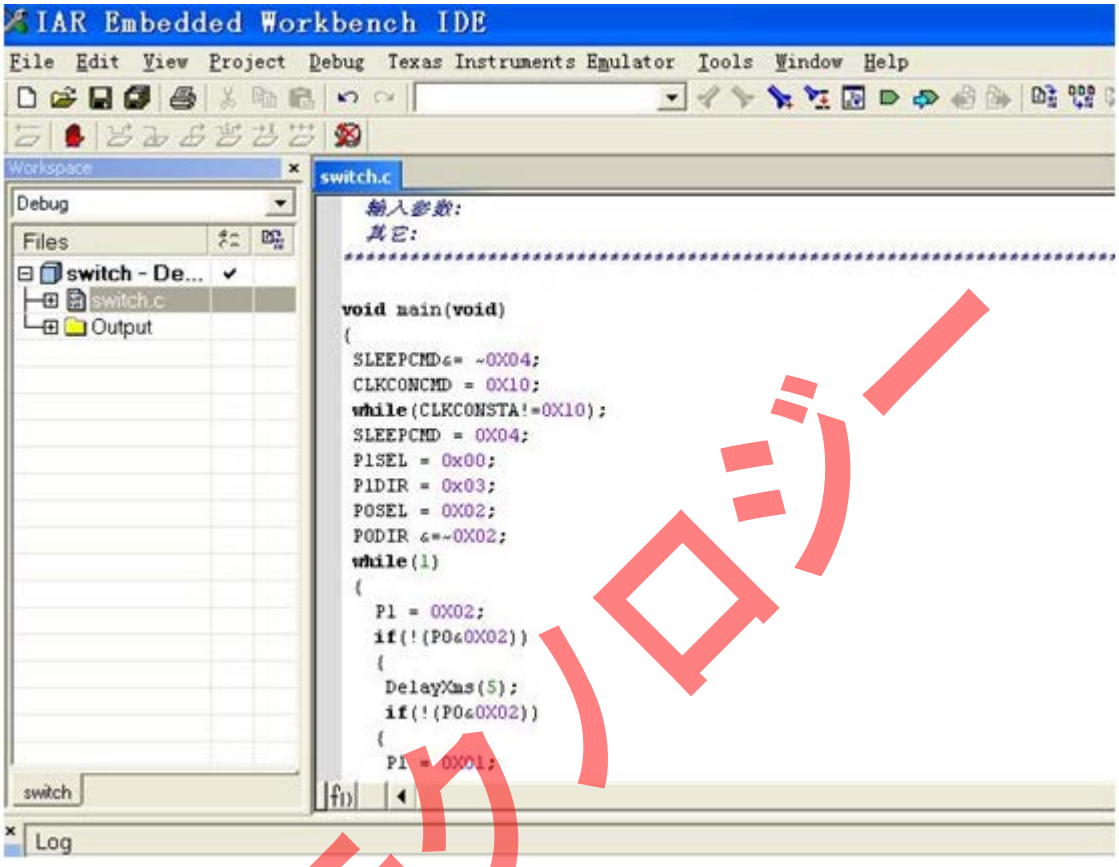


図 2－1

2.6　実験現象

プルグラムを実行後、LED2 は点灯、キーを押すと、LED2 は消し、LED1 が点灯する。キーを離すと、初期状態に戻る。

3、シリアルポート通信

3.1　実験の目的

CC2530 のシリアルポートの設定方法、基本機能を把握する。

3.2　実験の原理

CC2530 は USART0、USART1 二つのシリアルインターフェースがある。非同期通信モードまたは同期通信モードで動作できる。シリアルポートコントロール・状態レジスターU0CSR.MODE の設定によりシリアルポートの通信モードを選択する。UART モードで、インタフェースは 2 線式または RXD 、TXD を含めて、オプションの RTS と CTS の 4 線式モードを使用する。中に、RTS と CTS はハードウェア･フローを制御する。シリアルポートの設定とコントロールについての詳細内容は、CC2530 ユーザマニュアルのシリアルポート章をご参照下さい。

本実験で利用するレジスタ：

PERCFG(0Xf1)―外部デバイス制御

ビット	名前	リセット	読み/書き	説明
7	―	0	R0	使用しない
6	T1CFG	0	R/W	タイマー1 の I/O 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
5	T3CFG	0	R/W	タイマー3 の I/O 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
4	T4CFG	0	R/W	タイマー4 の I/O 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
3:2	―	00	R0	使用しない
1	U1CFG	0	R/W	USART1 の I/O 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
0	U0CFG	0	R/W	USART0 の I/O 位置 0：位置 1 へ選択する 1：位置 2 へ選択する

P1(0x90) _ ポート 1

ビット	名前	リセット	読み/書き	説明
7：0	P1[7：0]	0xFF	R/W	ポート 1、汎用 IO ポート、SFR ビットからアドレスを検索できる。この CPU 内部レジスタは XDATA(0x7090)から読み取りできる、なお書き込むできない。

P0SEL(0xF3) _ ポート 0 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELP0[7：0]	0x00	R/W	P0.7 から P0.0 への機能選択 0：汎用 IO 1：外部デバイス機能

P1SEL(0xF4) _ ポート 1 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELP1[7：0]	0x00	R/W	P1.7 から P1.0 への機能選択 0：通用 IO 1：外部デバイス機能

P1DIR(0xFE) _ ポート 1 方向

ビット	名前	リセット	読み/書き	説明
7：0	DIRP1__[7：0]	0x00	R/W	P1.7 から P1.0 への方向 0：入力 1：出力

U0CSR(0x86)―USART0 制御と状態

ビット	名前	リセット	読み/書き	説明
7	MODE	0	R/W	USART モード選択 0：SPI モード 1：UART モード
6	RE	0	R/W	URAT 受信イネーブル、注意：URAT を完全に設定される前に受信イネーブルしない。 0：受信禁止 1：受信イネーブル
5	SLAVE	0	R/W	SPI ホストモードまたはスレーブモード選択 0：SPI ホストモード 1：SPI スレーブモード
4	FE	0	R/WO	UART フレーミングエラーステータス 0：フレーミングエラーを検出しない 1：受信したバイトのストップビットレベルはエラー
3	U1CFG	0	R/W	USART1 の I/O 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
2	RX__BYTE	0	R/WO	受信バイト状態、UART モードと SPI スレーブモード,U0DBUF を読み取る時、自動にこのビットを削除する、0 と書くことより削除する、U0DBUF の中のデータを有効的に捨てる。 0：バイトを受信しない 1：受信したバイトは用意が整る。
1	TX__BYTE	0	R/WO	バイトを送信する状態、UART モードと SPI メインモード 0：バイトを送信しない 1：データバッファレジスタへ書き込んだ最後のバイトは送信された。
0	ACTIVE	0	R	USART 収/発アクティブ、SPI スレーブモードにおいて、このバイトはスレーブ選択に等しい 0：USART アイドル 1：送信または受信モードにおいて、USART 忙

U0GCR(0xC5) ―USART0 通用制御

ビット	名前	リセット	読み/書き	説明
7	CPOL	0	R/W	SPI クロック極性 0：負クロック極性 1：正クロック極性
6	CPHA	0	R/W	SPI クロックフェーズ 0：CPOL からの SCK が反転した後、CPOL へ戻る場合、データは MOSI へ出力し、CPOL からの SCK が CPOL 反転へ戻る場合、入力したデータは MOSI へサンプリングする。 1：CPOL からの SCK が CPOL 反転へ戻る場合、データは MOSI へ出力し、CPOL からの SCK が反転した後、CPOL へ戻る場合、入力したデータは MOSI へサンプリングする。
5	ORDER	0	R/W	送信用のビット手順 0：まず LSB 伝送 1：まず MSB 伝送
4：0	BAUD__E[4：0]	00000	R/W	ボーレートのインデックス値、BAUD__E は BAUD__M と一緒に UART ボーレートと SPI メイン SCK

				クロック周波数を決める

U0BAUD(0xC2) —USART0 ボーレート制御

ビット	名前	リセット	読み/書き	説明
7：0	BAUD__M[7：0]	0x00	R/W	ボーレート端数値、BAUD__E は BAUD__M と一緒に UART ボーレートと SPI メイン SCK クロック周波数を決める

U0DBUF(0xC1) —USART0 収/発データバッファ

ビット	名前	リセット	読み/書き	説明
7：0	DATA[7：0]	0x00	R/W	USART 送受信データ。データをこのレジスタへ書き込むのはデータを内部データ伝送レジスタへ書き込むこと、このレジスタを読み取るのは内部のデータ読み取りレジスタのデータを読み取ること。

3.3 実験条件

CC2530 開発ボード*１、ダウンローダー*１、ケーブル*１、シリアルケーブル*１

モジュールの USB/TTL 出力選択スイッチは ON 状態にする。初回使用の場合では PL2303 ドライバをインストール必要がある。また PC にシリアルデバッグソフトウェアをインストール必要。

3.4 実験内容

シリアルポートを通じて開発ボードと PC の通信を行う。PC からのデータを受信した後、受信データを PC に返送する。通電後、モジュールは PC のシリアルポートにデータを送り、シリアルポートからモジュールに最大 30 文字までの文字列、末尾に#を加えて送信します。モジュールは同じ文字列を返送します。ボーレートは 57600、パリティなし、1 ストップビット。

実現コード：

```
void main(void)
{
    InitIO();
    InitUart0();
    led1 = 1;
    DelayXms(10);
    len = strlen((char *)Recdata);
    SendString(Recdata,len );
    while(1)
      {
          if(RTflag == 1)              //受信
            {
              led2=0; //受信状態指示
               // led1 = 0;
                if( temp != 0)
                {
                    if((temp!='#')&&(datanumber<30))
                      {                                  //’＃‘は終了文字と定義される
                                                         //最大 30 文字まで受け取れる
                       Recdata[datanumber++] = temp;
                        }
```

```
                else
                {
                RTflag = 3; //発送状態に入る
                }
                if(datanumber == 30)RTflag = 3;
              temp = 0;
            }


          }


           if(RTflag == 3) //発送
          {


          led2 = 1;                                          //送信状態指
示
          U0CSR &= ~0x40;          //データを受信不可
          SendString(Recdata,datanumber);
          U0CSR |= 0x40;           //データの受信


          RTflag = 1;                    //受信状態に戻る
          datanumber = 0;          //ポインタは 0 にリセット
          //オフ送信指示
        }
      }
    }
```

先ず IO ポートを外部アプリケーションと設定、シリアル･ポートの設定を初期化し、シリアル･ポート割り込みを通じて PC からをデータを受信する。RTflag は受信・送信状態を標識する。受信状態の場合、シリアルポートから受信したデータは Recdata[]に保存する。送信状態の場合、配列 Recdata[]の値を送信する。

3.5　実験手順

開発ボードはダウンローダーおよびダウンロードケーブルと接続されてから、CC2530 プログラム例の中で、example¥3_uart のプロジェクトファイルをオープンし、インタフェースは図 3－1 のとおり、  をクリックしてプログラムをコンパイル、  をクリックしてリンク、最後に、  をクリックし開発ボードへダウンロードする。ボードでプログラムを実行し、実験現象を観察する。

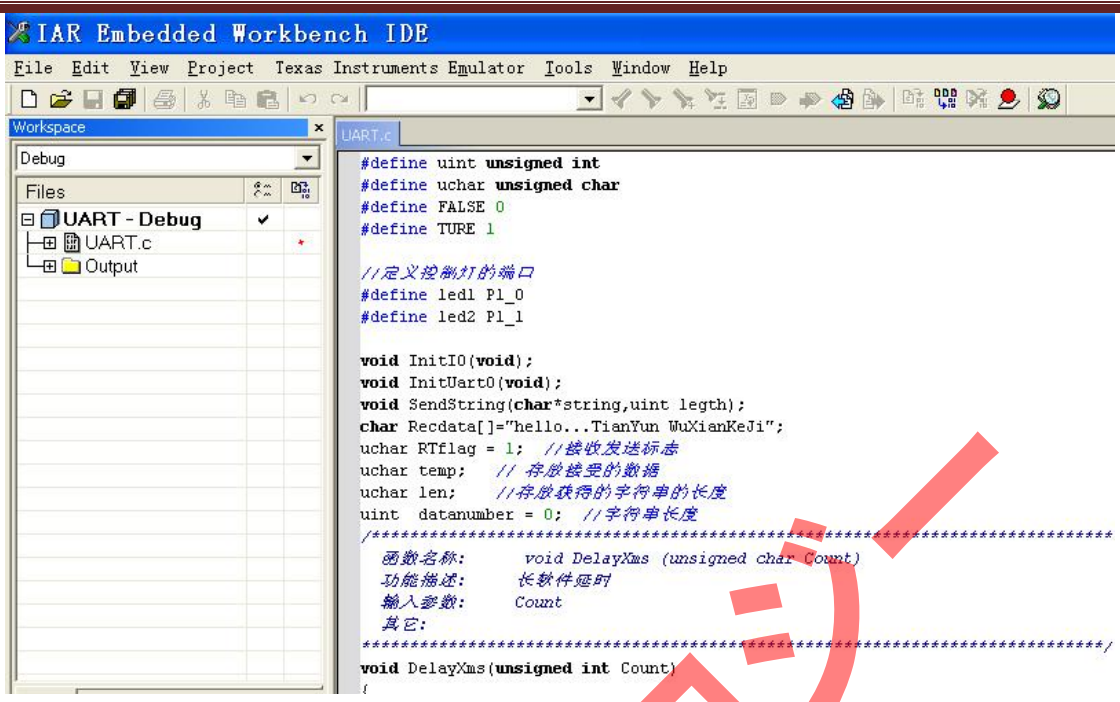


図 3－1

3.6 実験現象

プログラムをダウンロードしてから、プログラムを実行すると、LED1 は点灯、シリアルポートは図 3-2 の示すように、PC から text text text#を送信する。ボードは受信後 PC に返信する。シリアルポートは図 3-3 のように示し、LED2 は一回点滅する。

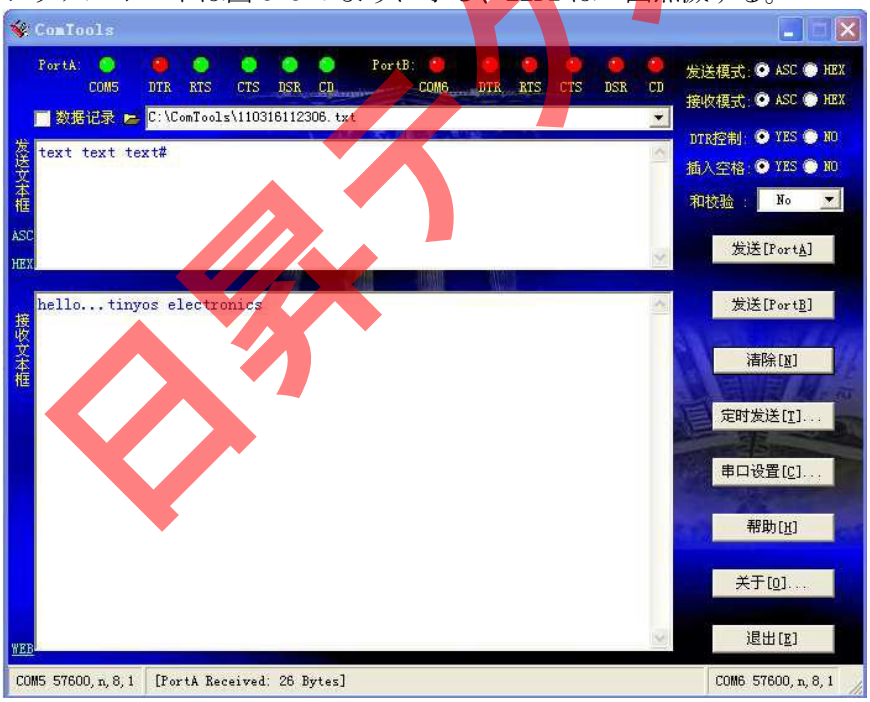


図 3－2

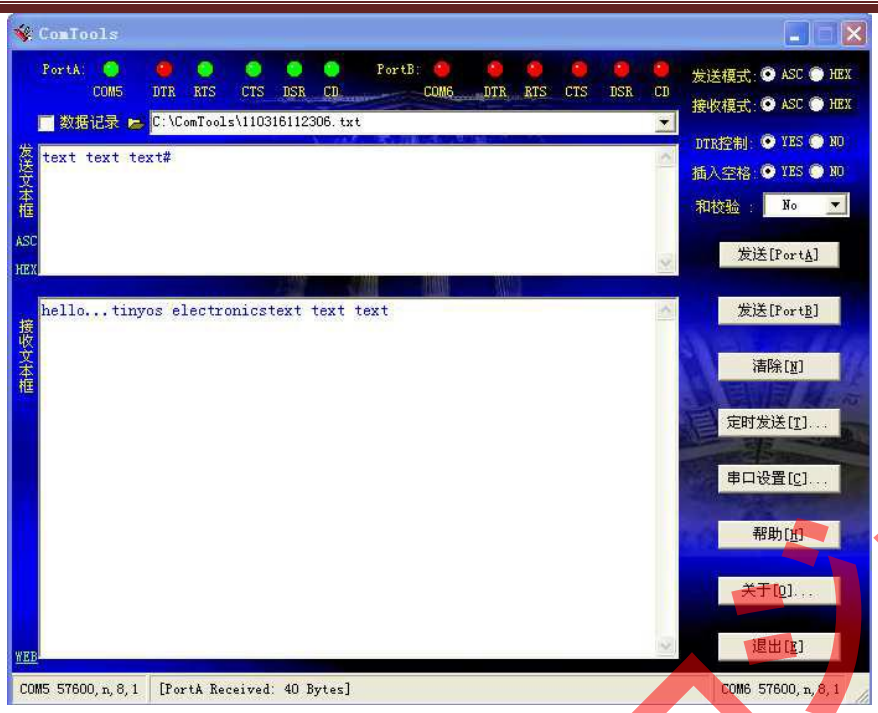


図 3－3

4　タイマー実験

4.1　実験の目的

タイマーの機能、設定方法を把握する。

4.2　実験の原理

タイマー1 は標準的なタイマー/カウンタ機能をサポートする。例えば、入力キャプチャ、出力比較と PWM 機能。独立なタイマーであり、五つ独立したキャプチャ/コンペアチャンネルを持つ。チャンネル毎に一つの IO ピンを使う。本実験はタイマー1 のタイマー機能のみを使用し、割り込み機能は使用しない。タイマーの他の機能については CC2530 ユーザーズマニュアルをご参照ください。

利用するレジスタは下記とおりであり：

P1SEL(0xF4)―ポート 1 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELPE__[7：0]	0x00	R/W	P1.7 から P1.0 への機能選択 0：汎用 I/O 1：外部デバイス機能

P1DLR(0xFE) ―ポート 1 方向

ビット	名前	リセット	読み/書き	説明
7：0	DIRP1__[7：0]	0x00	R/W	P1.7 から P1.0 への I/O 方向 0：入力 1：出力

T1CTL(0xE4) ―タイマー1 制御

ビット	名前	リセット	読み/書き	説明
7：4	―	0000	R0	保留
3：2	DIV[1:0]	00	R/W	プリスケーラ分割値 00: Tick 周波数/1 01: Tick 周波数/8 10: Tick 周波数/32 11: Tick 周波数/128
1：0	MODE[1:0]	00	R/W	タイマー1 モード選択 00：実行を中断する 01：自由に実行し、0x0000 から 0xFFFF まで繰り返しカウントする。 10： 0x0000 から T1CC0 まで繰り返しカウントする。 11:正カウント/逆カウント、0x0000 から T1CC0 まで、もう一度 T1CC0 から 0x0000 へ逆カウントし、繰り返しカウントする。

4.3　実験条件

CC2530 開発ボード*１、ダウンローダー*１、ケーブル*１

4.4　実験内容

タイマーの割り込み機能が無効で、タイマーは二回オーバーフローな場合 LED の状態が変わる。

本実験のソースコードは下記とおりであり：

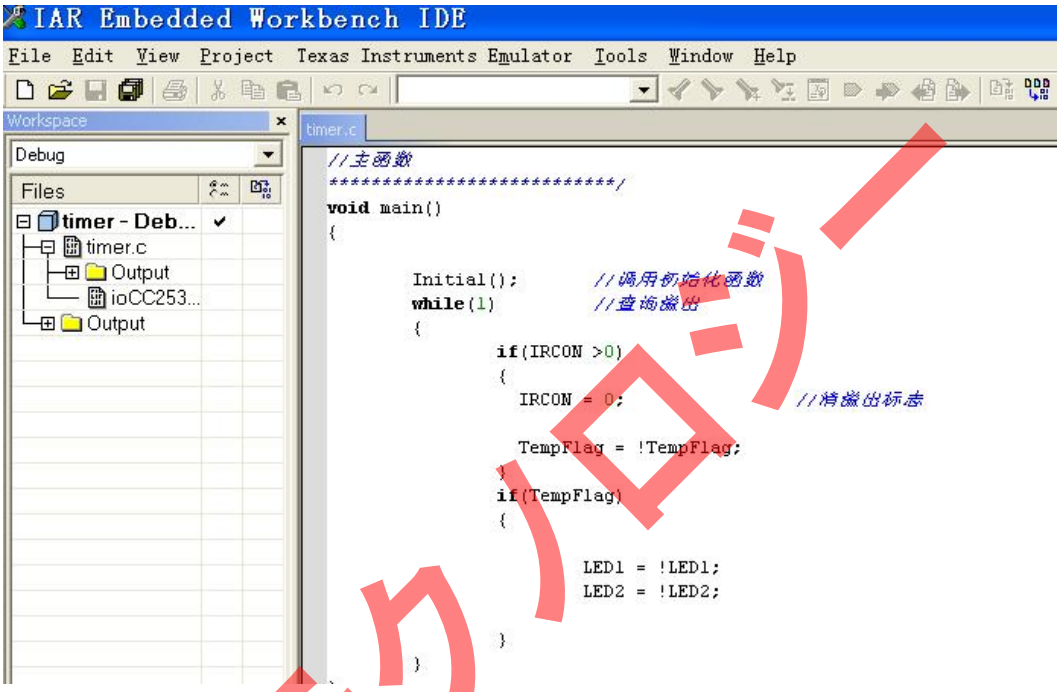
```
/*****************************
//プログラム初期化，システムクロック、IO ポートとタイマー1 を初期化，
*****************************/
void Initial(void)
```

```
{
SLEEPCMD&= ~0X04;
CLKCONCMD = 0X10;
while(CLKCONSTA!=0X10);
SLEEPCMD = 0X04;


P1SEL = 0X00;//初期化 P1
P1DIR = 0x03; //P10 P11  出力
LED1 = 0;
LED2 = 1; //LED 消し


//T1 で実験する
T1CTL = 0x0D; //中断無効   128 スケーラ分割; オートリロードモード (0x0000->0xffff)、
                //周波数は 0.25M



}
/*************************
//メイン関数
*************************/
void main()
{


    Initial(); //初期化関数呼び出し
    while(1) //オーバーフロークエリ
    {
            if(IRCON >0)
            {
            IRCON = 0;                    //オーバフローフラグをクリアする
TempFlag = !TempFlag ;
            }
        if(TempFlag)
        {
            LED1=!LED1

            LED2=!LED2
            }

      }
  }
```

ソースコードはシステムクロック、タイマー1とIOの初期化配置を行い、メインプログラムで、タイマーがカウントオーバフローな場合、割り込みフラグ中のIRCON下のT1IFはハイ・レベル、当フラグをクリアして、次のカウントサイクルに入る、オーバーフローが二回発生すると、二つLEDの状態は変わる。

4.5　実験手順

ハードウェア接続した後、CC2530 プログラム例の中で、example¥4_timer のプロジェクトファイルをオープンし、インタフェースは図 4−1 のとおり、 をクリックしてプログラムをコンパイル、をクリックしてリンク、最後に、 をクリックし開発ボードへダウンロードする。ボードでプログラムを実行し、実験現象を観察する。



4.6　実験現象

二つのLEDは循環点滅する。

5　割込み実験

5.1　実験の目的

C2530の割り込み構造、各割り込みソースに対しての割り込み処理、および割り込み優先順位を把握する。本実験はタイマーのオーバーフロー割り込みと外部割り込みをデモする。

5.2　実験の原理

CC2530のCPUは18個の割り込みソースが有り、各割り込みソースはスペシャルレジスタ中に自分の割り込みリクエストフラグがある。各割り込みは単独イネーブル・ディスエイブルできる。本実験はタイマー1のオーバーフロー割り込みとIO外部割り込みを使用する。ほかの割り込みはユーザーズマニュアルをご参照ください。

タイマー1のカウンターは最終カウント値に達つ、そして、グローバル割り込みとタイマー割り込みをイネーブル状態な場合、オーバーフロー割り込みが発生する、状態レジスタT1STATは最終カウント値イベントの割り込みフラグが含まれている。

IO外部割り込みな場合、IOポートは入力状態と設定する必要がある。割り込みは外部割り込み信号の立ち上がりエッジまたは立ち下がりエッジトリガに設定可能。本実験では立ち下がりエッジトリガを使用する。

実験に使用するレジスタは下記とおりであり：

P0(0x80) —ポート0

ビット	名前	リセット	読み/書き	説明
7：0	P0[7：0]	0xFF	R/W	ポート 0、汎用 IO ポート、SFR ビットからアドレスを検索可能。この CPU の内部レジスタは XDATA(0x7080)から読み取れる、なお書き込むできない.

P0(0x90) —ポート1

ビット	名前	リセット	読み/書き	説明
7：0	P1[7：0]	0xFF	R/W	ポート 1、汎用 IO ポート、SFR ビットからアドレスを検索可能。この CPU の内部レジスタは XDATA(0x7090)から読み取れる、なお書き込むできない.

P0SEL(0xF3) _ ポート 0 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELP0[7：0]	0x00	R/W	P0.7 から P1.0 への機能選択 0：汎用 IO 1：外部デバイス機能

P1SEL(0xF4) _ ポート 1 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELP1[7：0]	0x00	R/W	P1.7 から P1.0 への機能選択 0：汎用 IO 1：外部デバイス機能

P0DIR(0xFD) _ ポート 0 方向

ビット	名前	リセット	読み/書き	説明
7：0	DIRP0__[7：0]	0x00	R/W	P1.7 から P1.0 への方向 0：入力 1：出力

P1DIR(0xFE) _ ポート 1 方向

ビット	名前	リセット	読み/書き	説明
7：0	DIRP1__[7：0]	0x00	R/W	P1.7 から P1.0 への方向 0：入力

				1：出力

P0LNP（0x8F）_ ポート 0 入力モード

ビット	名前	リセット	読み/書き	説明
7：0	MDP0__[7：0]	0x00	R/W	P1.7 から P1.0 への入力モード 0：プルアップ/ドロップダウン（P2LNP（0xF7）_ ポート 2 入力モードをご参照ください） 1：トライステート

PICTL(0x8C) _割り込みトリガ･エッジ制御

ビット	名前	リセット	読み/書き	説明
7	PADCS	0	R/W	I/O ピンの出力モードでのドライバ能力を制御 0：最小ドライバ能力は強くなる。DVDD1/2>または＝2.6V 1:最大ドライバ能力は強くなる。DVDD1/2<2.6V
6:4	_	000	R0	使用しない
3	P2ICON	0	R/W	ポート 2、P2.4-P2.0 入力モードの割り込み配置、このビットはポート 2.4-2.0 の入力のわき込みリクエスト条件を選択する。 0：入力した立ち上がりエッジは割り込みを引き起こす. 1：入力した立ち下がりエッジは割り込みを引き起こす
2	P1ICONH	0	R/W	ポート 1、P1.7-P1.4 入力モードの割り込み配置、このビットはポート 2.4-2.0 の入力の割り込みリクエスト条件を選択する。 0：入力した立ち上がりエッジは割り込みを引き起こす. 1：入力した立ち下がりエッジは割り込みを引き起こす
1	P1ICONL	0	R/W	ポート 1、P1.3 -P1.0 入力モードの割り込み配置、このビットはポート 1.3-1.0 の入力の割り込みリクエスト条件を選択する。 0：入力した立ち上がりエッジは割り込みを引き起こす. 1：入力した立ち下がりエッジは割り込みを引き起こす。

P0IEN(0xAB) _ポート 0 割り込みイネーブル

ビット	名前	リセット	読み/書き	説明
7：0	P0_[7：0]IEN	0x00	R/W	ポート P0.7 から P0.0 への割り込みイネーブル 0：割り込み禁止 1：割り込みイネーブル

P0IFG（0x89）_ポート 0 割り込み状態フラグ

ビット	名前	リセット	読み/書き	説明
7：0	P0IF[7：0]	0x00	R/W0	ポート 0, ビット 7—ビット 0 入力割り込み状態フラグ。入力ポートの一つピンに割り込みリクエスト信号がある場合、相応するフラグビットに 1 と置く。

T1CTL(0xE4) _タイマー1 制御

ビット	名前	リセット	読み/書き	説明
7：4	—	0000 0	R0	保留
3：2	DIV[1：0]	00	R/W	プリスケーラ分割値、アクティブ･クロック･エッジを生成し、カウンター更新： 00：Tick 周波数/1 01：Tick 周波数/8 10：Tick 周波数/32 11：Tick 周波数/128

1：0	MODE[1：0]	00	R/W	タイマー1 モード選択、タイマー操作モードの選択は下記とおりであり： 00：実行中断 01：自由に実行、0x0000 から 0xFFFF まで繰り返しカウントする 10：0x0000 から T1CC0 まで繰り返しカウントする 11:正カウント/逆カウント、0x0000 から T1CC0 まで、もう一度 T1CC0 から 0x0000 へ逆カウントし、繰り返しカウントする。

IRCON(0xC0)_中断フラグ 4

ビット	名前	リセット	読み/書き	説明
7	STIF	0	R/W	スリープタイマー割り込みフラグ 0：割り込みサスペンドしない 1：割り込みサスペンドする
6	—	0	R/W	0 にしなければならない、1 と書くと、いつも割り込みソースをイネーブルする。
5	P0IF	0	R/W	ポート 0 割り込みフラグ 0：割り込みサスペンドしない 1：割り込みサスペンドする
4	T4IF	0	R/W H0	タイマー4 割り込みフラグ、タイマー4 割り込み発生する場合、1 とセットし、CPU は割り込み処理ルーチンに指定する場合、クリアする
3	T3IF	0	R/W H0	タイマー3 割り込みフラグ、タイマー3 割り込み発生する場合、1 とセットし、CPU は割り込み処理ルーチンに指定する場合、クリアする
2	T2IF	0	R/W H0	タイマー2 割り込みフラグ、タイマー2 割り込み発生する場合、1 とセットし、CPU は割り込み処理ルーチンに指定する場合、クリアする
1	T2IF	0	R/W H0	タイマー1 割り込みフラグ、タイマー1 割り込み発生する場合、1 とセットし、CPU は割り込み処理ルーチンに指定する場合、クリアする

IENE(0xB8)—割り込みイネーブル

ビット	名前	リセット	読み/書き	説明
7：6	—	0	R/0	使用しない、読み取る時、0 である。
5	P0IE	0	R/W	ポート 0 割り込みイネーブル 0：割り込み禁止 1：割り込みイネーブル
4	T4IE	0	R/W	タイマー4 割り込みイネーブル 0：割り込み禁止 1：割り込みイネーブル
3	T3IE	0	R/W	タイマー3 割り込みイネーブル 0：割り込み禁止 1：割り込みイネーブル
2	T2IE	0	R/W	タイマー2 割り込みイネーブル 0：割り込み禁止 1：割り込みイネーブル
1	T1IE	0	R/W	タイマー1 割り込みイネーブル 0：割り込み禁止 1：割り込みイネーブル
0	DMAIE	0	R/W	DMA 伝送割り込みイネーブル

5.3 実験条件

CC2530 開発ボード*1、セットダウンローダー*1、ケーブル*1

5.4 実験内容

本実験では二つの割り込みがあります：タイマー1 オーバーフロー割り込みと IO ポート外部割り込みです。タイマー1 中断はオートリロードのタイマーであり、128 頻度分割と配置しております、割り込み発生後 LED2 はステータ反転する。外部割り込みは P0.1 ポート、立ち下がりエッジトリガとする。P0.1 上のボタンが押された時に割り込み発生し、LED1 は一回点滅する。

プログラムコードは下記とおりであり：

```
//シリアルポートと外部割り込み初期化関数 InitIO(void)
void InitIO(void)
  {
SLEEPCMD&= ~0X04;
CLKCONCMD = 0X10;     //     システムクロック 32M
while(CLKCONSTA!=0X10);
SLEEPCMD = 0X04;
P1SEL = 0X00;       //LED ポートの配置
    P1DIR|= 0X03;

    P1 =0X00;
    P0SEL = 0X00;       //P0<1>外部デバイスに設定する
    P0DIR &=~0X02;
    P0INP &= ~0x02;    //プルダウンモード
    PICTL |= 0x01;     //立ち下がりエッジトリガ
    P0IEN |= 0x02;  //  イネーブル p0_1  割り込み
    P0IE = 1;          //イネーブル P0 ポート割り込み
    P0IFG = 0;
    P0IF = 0;
    }
    //タイマーを初期化し、割り込みを起動する
    void Inittimer(void)

    {
    T1CTL = 0X0D;
    T1IE = 1;//タイマー割り込みを起動する
    T1IF =0;
    }
    //メイン関数
    void main(void)
    {
    InitIO();
    Inittimer();
    EA = 1;     //割り込みを起動する
    while(1)
    {
    }
```

```
//タイマー割り込み関数、led2 のステータ反転する。
#pragma vector = T1_VECTOR
__interrupt void T1_ISR(void)
{
T1IF = 0; //割り込みフラグをクリアする
led2 =!led2;
}
//外部割り込み関数、ボタンを押した場合、led1 のステータ反転する
#pragma vector = P0INT_VECTOR
__interrupt void P0_ISR(void)
{
if(P0IFG>0)                  //ボタン割り込み
{
  DelayXms(10);
if(P0IFG>0)
{
P0IFG = 0;
led1 = !led1;
}
}
P0IF = 0;              //割り込みフラグをクリアする
}
```

5.5　実験手順

ハードウェア接続した後、CC2530 プログラム例の中で、example¥5_interrupt のプロジェクトファイルをオープンし、インタフェースは図 5－1 のとおり、  をクリックしてプログラムをコンパイル、  をクリックしてリンク、最後に、  をクリックし開発ボードへダウンロードする。ボードでプログラムを実行し、実験現象を観察する。

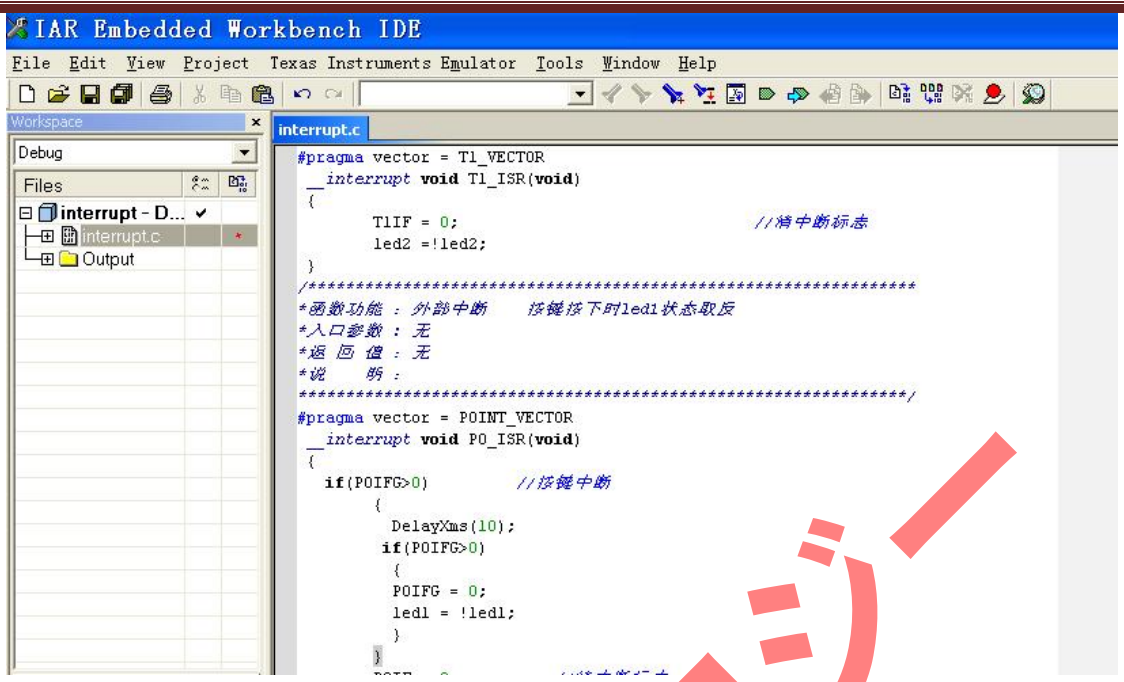


図 5－1

5.6　実験現象

プログラム実行後、LED2 は点滅状態で、ボタンを押した後、LED1 は一回点滅する。

6　1/3AVDD 実験

6.1　実験の目的

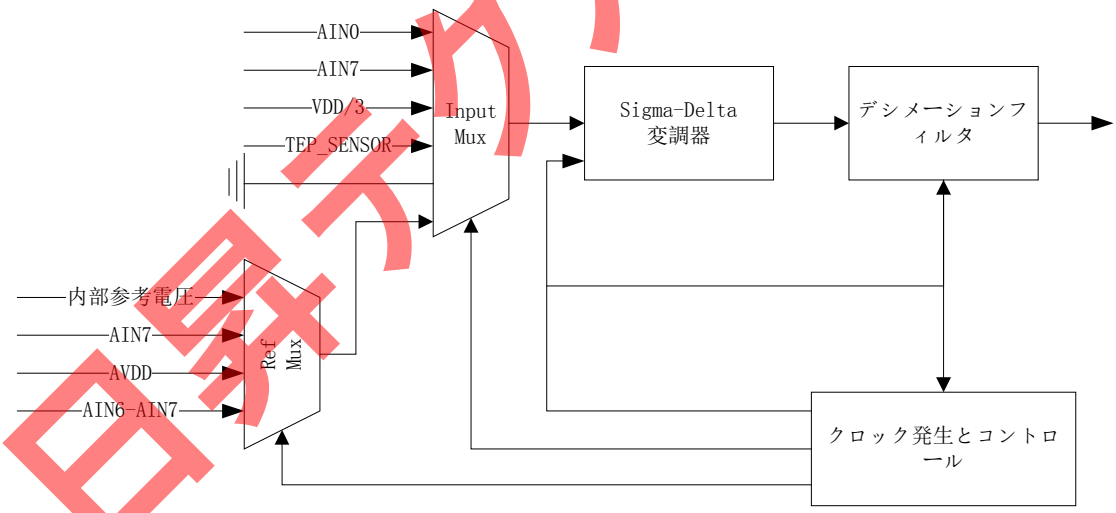
外部デバイスモジュール AD の機能、設定と使い方を把握する。

6.2　実験の原理

CC2530 の AD は 14 ビットのアナログ － デジタル変換をサポートし、最大有効ビット数は 12 ビット、8 つの独立したコンフィギュレーションチャネルアナログマルチプレクサがある。ADC の主な特徴とブロック図は下記とおりである：

主な特徴：

- 可選的デシメーションレート、解像度を設定します。（7－12 ビット）
- 8 つの独立の入力チャネル、シングルエンドや差動
- 参照電圧は内部、外部シングルエンド、外部差動或いは AVDD5
- 割込みリクエスト発生
- 転換終了の時、DMA はトリガーされる。
- 温度センサー入力
- バッテリ測定能力



ADC の詳しい説明については CC2530 のユーザーズマニュアルの ADC 節をご参照ください。

実験に使用したレジスタは下記とおりであり：

ADCL(0xBA)－ADC データのロービット

ビット	名前	リセット	読み/書き	説明
7：2	ADC[5:0]	0000　00	R	ADC 変換結果のロービット有効部分
1：0	—	00	R/0	使用しない、いつも 0 と読み取る

ADCH(0xBB)－ADC データのハイビット

ビット	名前	リセット	読み/書き	説明
7：0	ADC[13:6]	0x00	R	ADC 変換結果のハイビット有効部分

ADCCON1 (0xB4) _ADC 制御 1

ビット	名前	リセット	読み/書き	説明
7	EOC	0	R/H0	変換終了、ADCH は読み取られる時、クリアされる。前のデータは読み取られる前に、新たな

				変換は完成される場合、このビットは高と維持する、 0：変換は完成してない 1：変換は完成
6	ST	0		変換は始める。変換は完成する前に、1 と読み取られる 0：進行中の変換はない。 1：ADSSON1.STSEL は 11 で且つ変換を行っていない場合、一つ変換シーケンスを起動する
5：4	STSEL［1：0］	11	R/W1	トリガーを選択する。どのイベントで変換シーケンスを起動するかを選択する 00：P2.0 ピンの外部トリガー 01：全速、トリガーを待たない 10：タイマー1 チャンネル 0 比較イベント 11：ADCCON1.ST=1
3：2	RCTRL［1：0］	00	R/W	16 ビット乱数ジェネレータ（13 節）を制御する。01 と書くと、操作が終了する時、設置を改め、自動に 0x00 に戻る。 00：正常に実行する 01：同期 LFSR 一回 10：保留 11：停止、乱数ジェネレータをオフする
1：0	—	11	R/W	使用しない、いつも 11 にする

ADCCON1（0xB6）_ADC 制御 3

ビット	名前	リセット	読み/書き	説明
7：6	EREF[1:0]	00	R/W	単一変換のために使用する基準電圧を選択する 00：内部基準 01：AIN7 ピンの外部基準 10：AVDD5 ピン 11：AIN6—AIN7 の差動入力の外部基準
5：4	EDIV[1:0]	00	R/W	単一コンバータのためサンプリングレートを選択する、サンプリングレートは解像度と一つ変換を完成するようにかかる時間を決める。 00：64 サンプリングレート（7 ビット解像度） 01：128 サンプリングレート（9 ビット解像度） 10:256 サンプリングレート(10 ビット解像度) 11:512 サンプリングレート(12 ビット解像度)
3：0	ECH[3:0]	0000	R/W	単一チャンネル選択する。ADCCON3 を書くことによりトリガーする単一変換のチャンネル番号を選択し、単一の変換を完成する時、これらのビットは自動にクリアする。 0000：AIN0 0001：AIN1 0010：AIN2 0011：AIN3 0100：AIN4 0101：AIN5 0110：AIN6 1000：AIN0- AIN1 1001：AIN2- AIN3 1010：AIN4- AIN5 1011：AIN6- AIN7 1100：GND 1101：保留 1110：温度センサー

				1111：VDD3

PERCFG(0xF1)—外部デバイス制御

ビット	名前	リセット	読み/書き	説明
7	—	0	R/0	使用しない
6	T1CFG	0	R/W	タイマー1 の I/0 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
5	T3CFG	0	R/W	タイマー3 の I/0 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
4	T4CFG	0	R/W	タイマー4 の I/0 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
3：2	—	00	R/ 0	使用しない
1	U1CFG	0	R/W	USART1 の I/0 位置 0：位置 1 へ選択する 1：位置 2 へ選択する
0	U0CFG	0	R/W	USART0 の I/0 位置 0：位置 1 へ選択する 1：位置 2 へ選択する

P1(0x90) _ ポート 1

ビット	名前	リセット	読み/書き	説明
7：0	P1[7：0]	0xFF	R/W	ポート 1、汎用 IO ポート、SFR ビットからアドレスを検索できる。この CPU の内部レジスタは XDATA(0x7090)から読み取りできる、なお書き込むできない。

P0SEL(0xF3) _ ポート 0 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELP0_ [7：0]	0x00	R/W	P0.7 から P1.0 への機能選択 0：汎用 IO 1：外部デバイス機能

P1SEL(0xF4) _ ポート 1 機能選択

ビット	名前	リセット	読み/書き	説明
7：0	SELP1_ [7：0]	0x00	R/W	P1.7 から P1.0 への機能選択 0：汎用 IO 1：外部デバイス機能

P1DIR(0xFE) _ ポート 1 方向

ビット	名前	リセット	読み/書き	説明
7：0	DIRP1_ [7：0]	0x00	R/W	P1.7 から P1.0 への方向 0：入力 1：出力

U0CSR(0x86)—USART0 制御と状態

ビット	名前	リセット	読み/書き	説明
7	MODE	0	R/W	USART モード選択 0：SPI モード 1：UART モード
6	RE	0	R/W	URAT レシーバーイネーブル、注意：URAT は完全に配置される前にイネーブルしない。 0：レシーバー禁止 1：レシーバーイネーブル
5	SLAVE	0	R/W	SPI ホストモードまたはスレーブモード選択 0：SPI ホストモード 1：SPI スレーブモード
4	FE	0	R/WO	UART フレームエラーステータス 0：フレームエラーを検出してない

				1：受信したバイトのストップビットレベルはエラー
3	ERR	0	R/WO	UART パリティチェックエラー状態 0：パリティチェックエラーを検出してない 1：受信したバイトのパリティチェックはエラー
2	RX_BYTE	0	R/WO	受信バイト状態、UART モードと SPI スレーブモード,U0DBUF を読み取る時、自動にこのビットをクリアする、0 と書くことよりクリアする、U0DBUF の中のデータを有効的に捨てる。 0：バイトを受信してない 1：受信したバイトは用意済み。
1	TX_BYTE	0	R/WO	バイトを送信する状態、UART モードと SPI ホストモード 0：バイトを送信してない 1：データバッファレジスタへ書き込んだ最後のバイトは送信された。
0	ACTIVE	0	R	USART 収/発アクティブ。SPI スレーブモードの場合、このビットはスレーブ選択に等しい 0：USART アイドル 1：送信または受信モードにおいて、USART ビジー

U0BAUD(0xC2) —USART0 ボーレート制御

ビット	名前	リセット	読み/書き	説明
7：0	BAUD_M[7：0]	0x00	R/W	ボーレート端数値、BAUD__E は BAUD__M と一緒に UART ボーレートと SPI メイン SCK クロック周波数を決める

U0DBUF(0xC1) —USART0 収/発データバッファ

ビット	名前	リセット	読み/書き	説明
7：0	DATA [7：0]	0x00	R/W	USART は送受信データバッファ。データをこのレジスタへ書き込むのはデータを内部データ伝送レジスタへ書き込むこと、このレジスタを読み取るのは内部のデータ読み取りレジスタのデータを読み取ること。

U0GCR(0xC5) —USART0 制御

ビット	名前	リセット	読み/書き	説明
7	CPOL	0	R/W	SPI クロック極性 0：負クロック極性 1：正クロック極性
6	CPHA	0	R/W	SPI クロックフェーズ 0：CPOL からの SCK が反転した後、CPOL へ戻る場合、データは MOSI へ出力し、CPOL からの SCK が CPOL 反転へ戻る場合、入力したデータは MOSI へサンプリングする。 1：CPOL からの SCK が CPOL 反転へ戻る場合、データは MOSI へ出力し、CPOL からの SCK が反転した後、CPOL へ戻る場合、入力したデータは MOSI へサンプリングする。
5	ORDER	0	R/W	送信用のビット順序 0：LSB から伝送 1：MSB から伝送
4：0	BAUD__E[4：0]	00000	R/W	ボーレートのインデックス値、BAUD__E は BAUD__M と一緒に UART ボーレートと SPI メイン SCK クロック周波数を決める

6.3 実験条件

CC2530 開発ボード*１、ダウンローダー*１、ケーブル*１、シリアルライン*１

6.4 実験内容

本実験は CC2530 の AD 参照電圧の 1/3 を変換対象とする。参照電圧は 3.3V。シリアルポートから変換後の電圧値を PC へ送信し、表示させる。プログラム中ではシリアルポートと ADC を初期化し、AD が変換後の結果を公式 num = adc*1.25/8192 で電圧に変換し、シリアルポートから輸出します。プログラムコードは下記とおりであり：

//IO ポードとシリアルポート 1 を初期化し、ボーレートは 57600、8 つデータビット、パリティなし、1 ストップビット

```
void initUARTtest(void)
{
  SLEEPCMD&= ~0X04;
  CLKCONCMD = 0X10;
  while(CLKCONSTA!=0X10);
  SLEEPCMD = 0X04;



  PERCFG = 0x00;              //位置 1 P0 ポート
  P0SEL = 0x3c;               //P0 をシリアルポートとして使用

  U0CSR |= 0x80;              //UART モード
  U0GCR |= 10;               //baud_e = 11;
  U0BAUD |= 216;              //ボーレートは 57600 と設定
  UTX0IF = 1;

  U0CSR |= 0X40;              //受信を許可する
  IEN0 |= 0x84;              //割り込みを起動し、割り込みを受信する
  }

  //  ADC を初期化。電圧 AVDD を参照して変換対象は AVDD の 1/3

  void InitialAD(void)
   {
      ADCH &= 0X00;       //EOC フラグをクリア
      ADCCON3=0x3f;       //シングル変換,参照電圧は 1.25V で,
                          //1/3 AVDD に対して A/D 変換を行い
                          //14 ビット解像度
      ADCCON1 = 0X30;             //A/D を停止
```

```
    ADCCON1 |= 0X40; //A/D を起動

    }

  //  シリアルポート送信関数。 data は送信データで、len はデータサイズ。
  void UartTX_Send_String(char *Data,int len)
{
    int j;
    for(j=0;j<len;j++)
    {

      U0DBUF = *Data++;
      while(UTX0IF == 0);
      UTX0IF = 0;
    }
   }


  //  メイン関数
  void main(void)

  {
     uint adc;
     //P1 out
   P1DIR = 0x03;          //P1 は LED をコントロールする
   led1 = 1;
   led2 = 1;        //LED をオフする

   char temp[2];
     float num;

  initUARTtest(); //シリアルポートを初期化する
  InitialAD();//ADC を初期化する
  while(1)
  {
    if(ADCCON1>=0x80)
    {
      led1 = 1;          //変換終了指示
      temp[1] = ADCL;
      temp[0] = ADCH;
                    InitialAD();
      ADCCON1 |= 0x40;        //次の変換をスタートする

      adc = (uint)temp[1];
                  adc |= ( (uint)temp[0] )<<8;
```

```
                if(adc&0x8000)adc = 0; //D15 ビットは1の場合,今度のサンプリング
                した電圧値はマイナスで、処理しない。
                adc >>= 2;
                num = adc*1.25/8192; //一つ符号ビットがあり、2^13 を取る;
                //参照電圧は 3.3V と設定する。14 ビット精度
                adcdata[1] = (char)(num)%10+48; //48 は 0 が ASCII のなかの値である。
                //adcdata[2] = '.';
                adcdata[3] = (char)(num*10)%10+48;
        UartTX_Send_String(adcdata,6); //シリアルポート送信
                                                    //スペースを含め
        Delay(30000);
                        led1 = 0;                   //データ処理を完成
                        Delay(30000);
        }
    }
}
```

6.5　実験手順

ハードウェア接続した後、CC2530 プログラム例の中で、example¥6_AD_VDD のプロジェクトファイルをオープンし、インタフェースは図 6－1 のとおり、 をクリックしてプログラムをコンパイル、 をクリックしてリンク、最後に、 をクリックし開発ボードへダウンロードする。ボードでプログラムを実行し、実験現象を観察する。

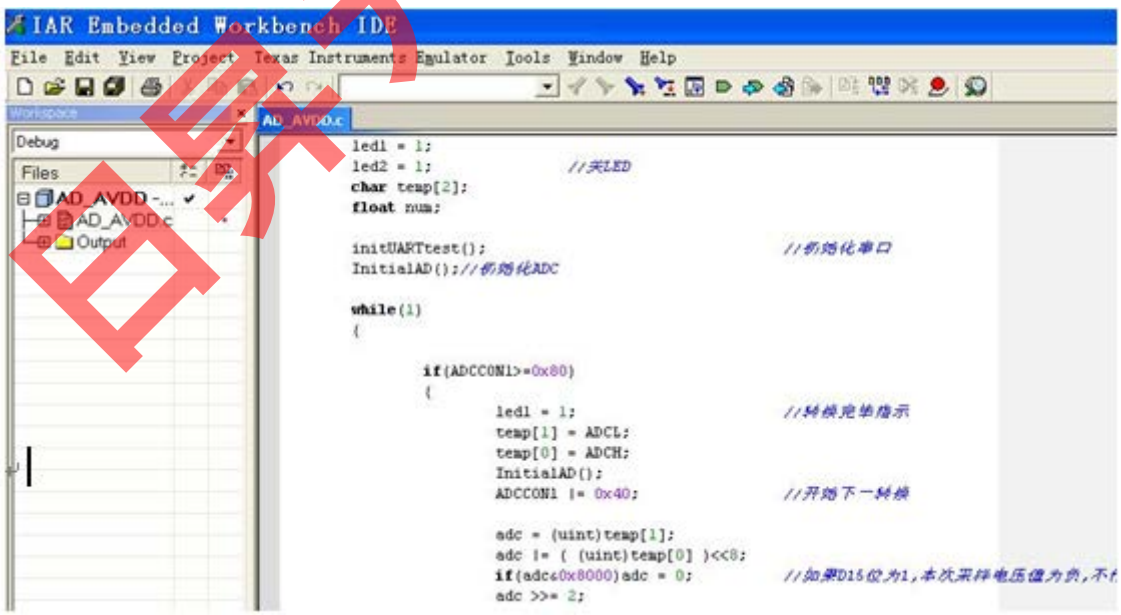


図 6－1

6.6　実験現象

ダウンロードしたをプログラム実行し、シリアルポートで変換電圧値を確認できる、図 6－2 をご覧下さい。発送時にボード上の LED1 は点滅状態で、LED2 は点灯する。

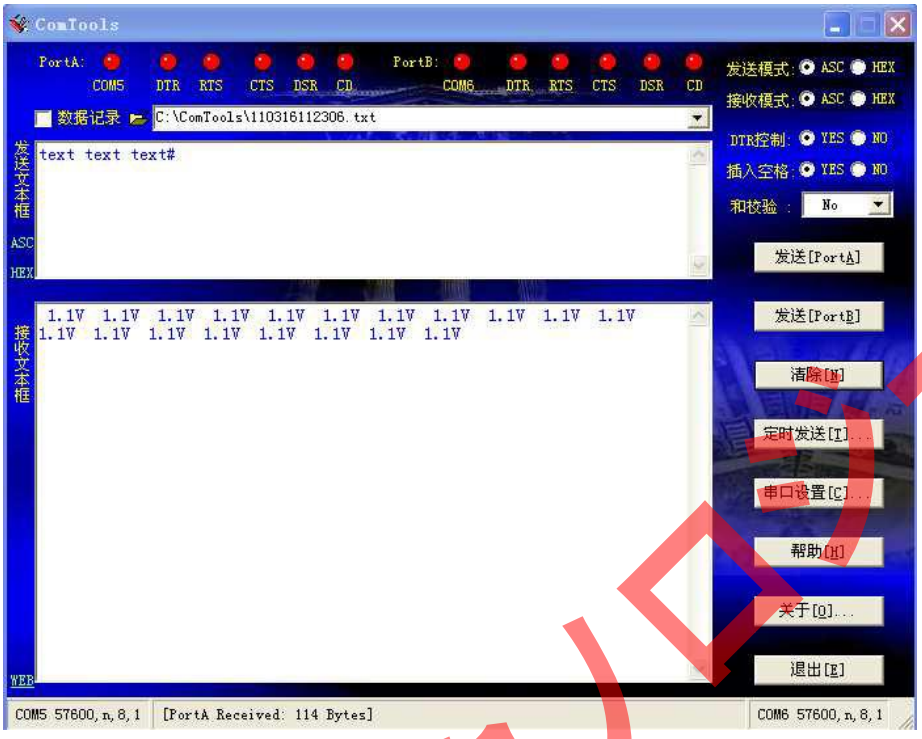


図 6－2

7　ポイントツーポイント通信

7.1　実験の目的

RF の構成を理解し、CC2530 RF モジュールの使用を把握する。

7.2　実験の原理

CC2530 の RF のカーネルはアナログ無線モジュールを制御し、同時に MCU と無線のあいだにインターフェイスを提供する。インターフェイスを通じでコマンドを出し、状態を読み取り、イベントの自動ソートなどができる。FSM のサブモジュールは RF トランシーバの状態、FIFO と大部分の動的に制御されるアナログ信号を制御する。変調器はオリジナルデータを I/Q 信号に変換し、トランスミッタ・DAC へ送信する。復調器は受信した信号からワイヤレスデータを検索し、復調器のボラティリティ情報を自動的にゲインコントロールする。周波数合成器は RF 信号にキャリアを生じて、ストロボプロセッサは CPU からの全てのコマンドを処理する。タイマー2 はワイヤレスイベントをタイミングし、出力パケットのタイムスタンプを取得する機能がある。

ワイヤレス RAM はデータを送信する FIFO（TXFIFO）とデータを受信する FIFO(RXFIFO)が有り、各 FIFO の長さは 128 バイト。TXFIFO と RXFIFO は SFR レジスタ経由でアクセスする、RFD レジスタを通じ TX FIFO にデータを書き込み、そして RXFIFO のデータを読み取る。

本実験では TI が提供する CC2530 シンプル・プロトコルスタックを利用し、簡単なワイヤレス通信機能を実現する。

7.3　実験条件

CC2530 実験モジュール*２、ダウンローダー*２、シリアルケーブル*２、USB アダプタケーブル*１（給電用）

7.4　実験内容

本実験では TI が提供する CC2530 シンプルなプロトコルスタックを使用し、二つの対等なレベルのモジュールの間の簡単なデータ発送機能を実現する。

7.4.1 シンプル･プロトコルスタックのディレクトリ

シンプル･プロトコルスタックのディレクトリは図 7－1 で示すように：コア関数は basicrf と hal の二つのグループ。下記図のように application：ユーザーが書いたアプリケーションを保存する。Basicrf：RF モジュールの設定に関連するライブラリで、Hal：回路ボードのハードウェア･コンフィギュレーション･ライブラリです。Cul と hal はプロトコルがカプセル化済みなライブラリです。ユーザーはニーズによって利用できる。Tools：使用ツール、output：プロジェクトが生成するファイル（自動生成、ユーザー呼出とコンフィギュレーションする必要はない）。

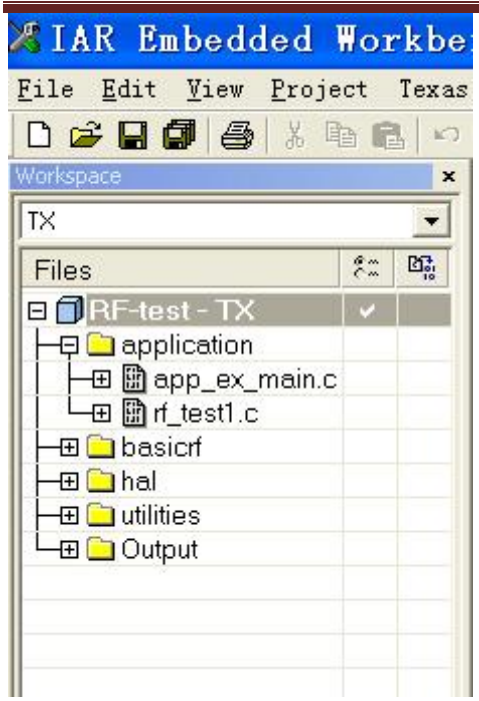


図 7－1

7.4.2 CC2530　シンプル・プロトコルスタックの RF 初期化

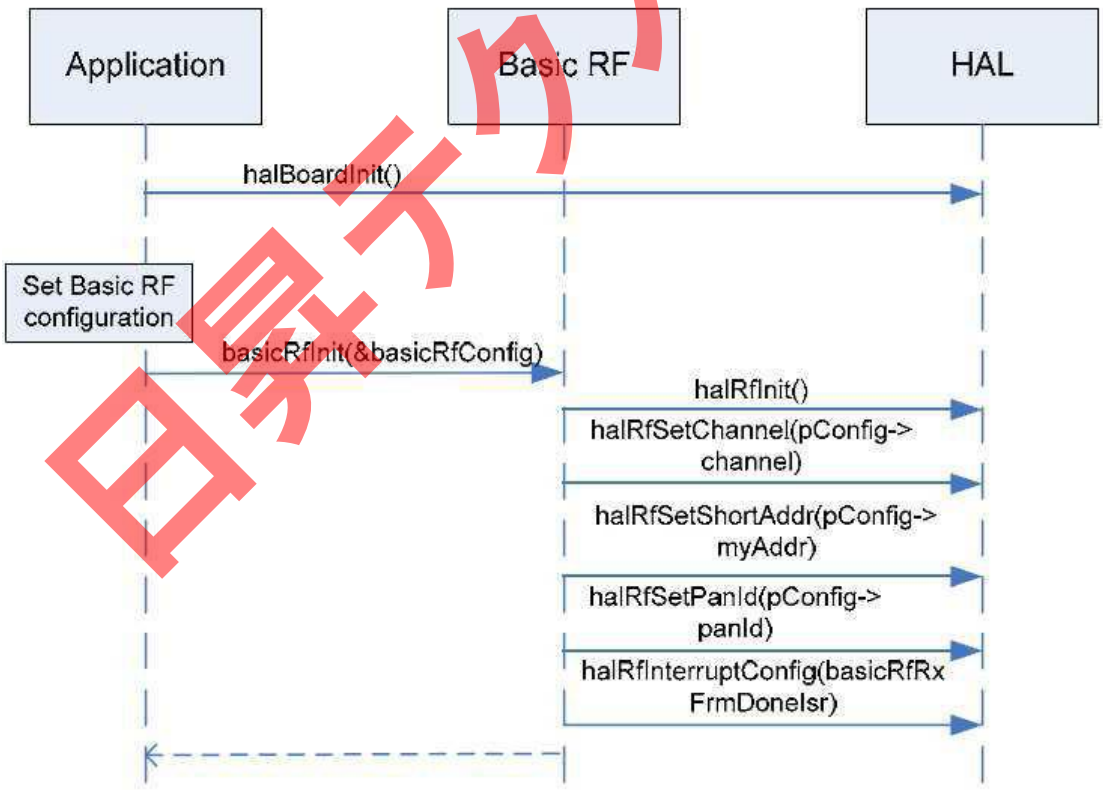


図 7－2　初期化

図 7－2 は RF の初期化プロセス。RF のコンフィギュレーション完了後初期化を行う。関数 halBoardinit()を呼び出し、CC2530 周辺モジュールを初期化し IO ポートを設定する。関数 basicRfinit()を呼び出し、RF を初期化する。関数 basicRfinit()を呼び出す前に、basicRfCfg_t

型の配列をコンフィギュレーションする、ソースアドレス、ターゲットアドレス、チャンネル、ショートアドレスと PANID などのパラメーターを設定する。関数 basicRfInit()で halRFInit()を呼び出して、無線レジスタのコンフィギュレーションとレシーバー割込みを起動する。

7.4.3 データ送信プロセス

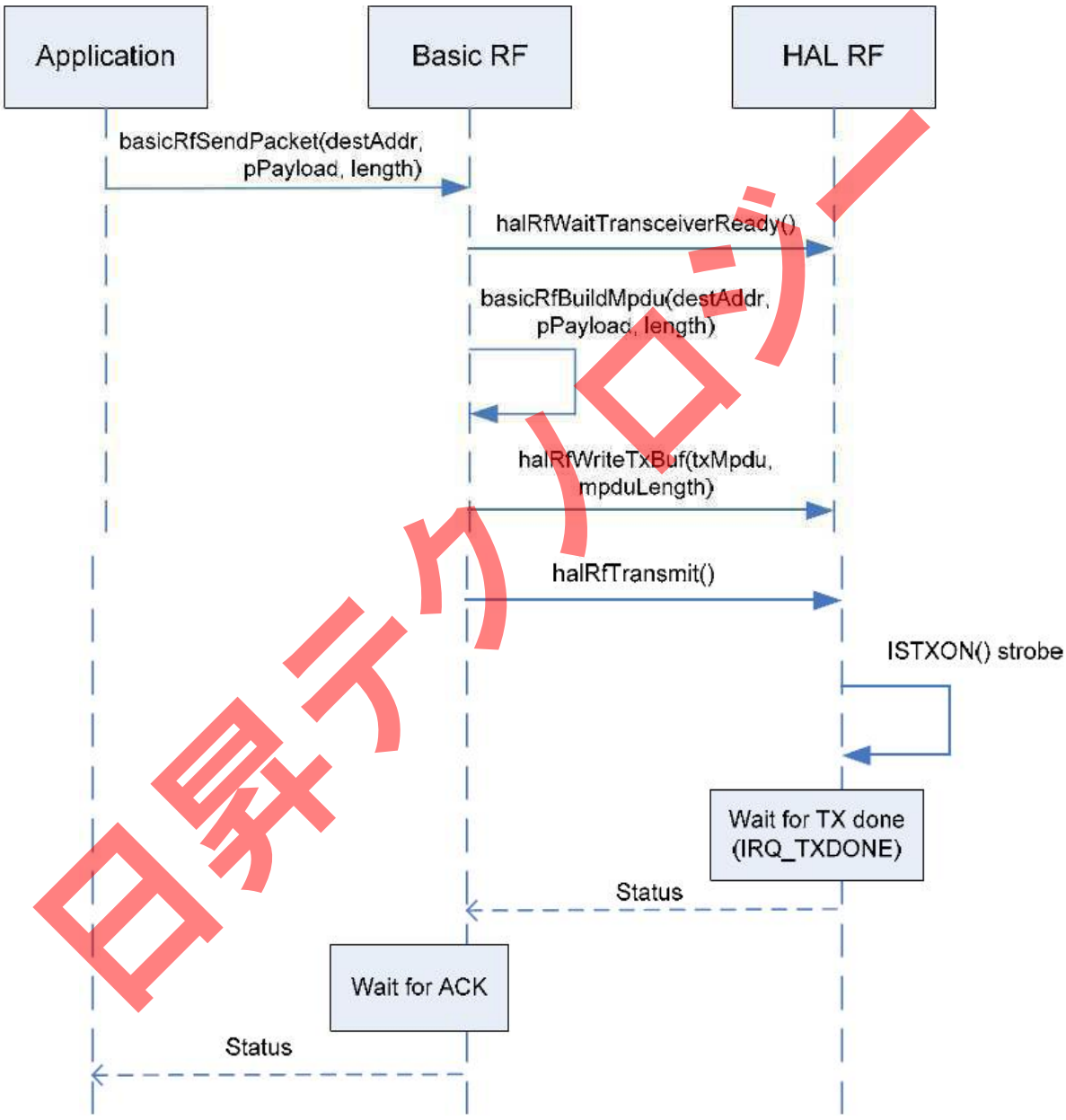


図 7−3　データの送信

データ送信プロセスは図 7−3 の示すように:アプリケーションに関数で basicRfSendPacket()を呼び出し、データを発送する。関数にターゲットアドレス、送信内容とデータの長さを含む。上記図のように、送信関数を送信後、halRfWaitTransceiverReady()は発射アイドル状態を待ち、関数 basicRfBuildMpdu()を呼び出し、MPDU を生成する。関数 halRfWriteTxBuf()を呼び出し、MPDU

を CC2530 TX FIFO レジスタに書き込む。最後に、関数 halRfTransmit()を呼び出し、データを送信する。送信完了後、ACK 確認待ち、所定返信時間内 ACK 確認を受信すると、SUCCESS 状態を戻す。

7.4.4 データの受信プロセス

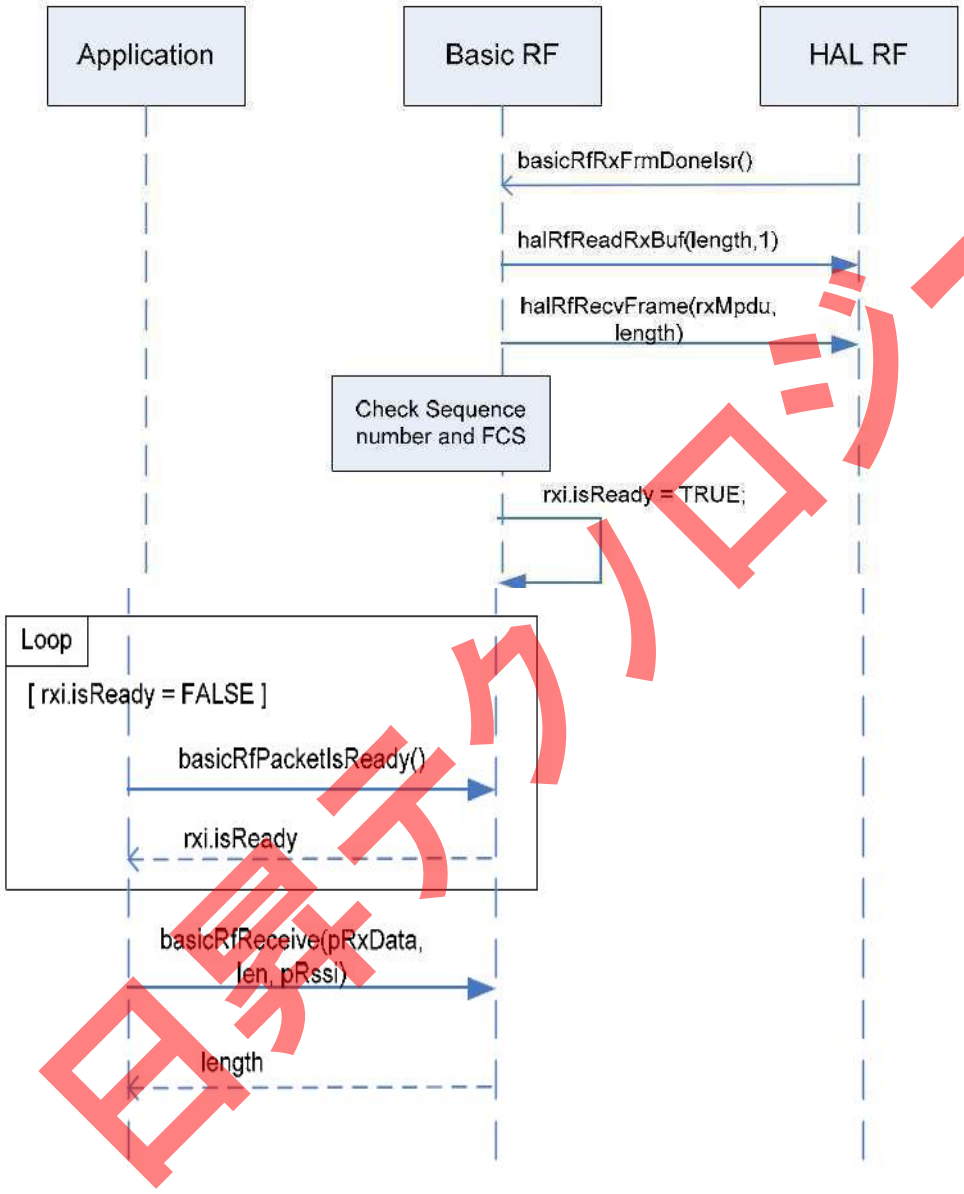


図 7－4　データの受信

データの送信関数と同じ、各ステップに相応な関数を呼び出してデータの受信プロセスを実現する。上記の RF の初期化、データの送信、データの受信プロセスも把握する必要です。RF の初期化の時、halBoardinit()と basicRfinit()二つ関数を呼び出し、データを送信する時、関数 basicRfSendPacket()を呼び出して、データを受信する時、関数 basicRfReceive()を呼び出す。

7.4.5 フローチャート

本実験の基本プロセスは図 7−5 のとおりである。

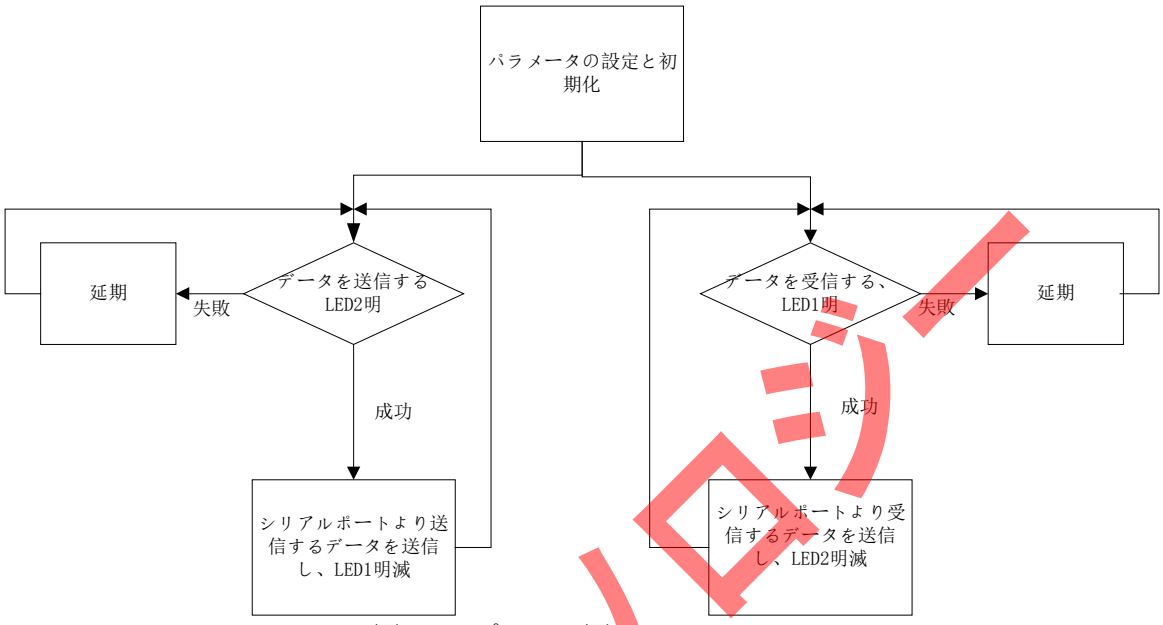


図 7−5 プロセス図

7.4.6 ソースコード

1、RF パラメータを設定して初期化する

```
void rf_test_main(void){

#ifdef RX      //受信モジュール
{

        basicRfConfig.myAddr = ADDRESS_0;
        remoteAddr = ADDRESS_1;
        basicRfConfig.channel = RF_CHANNEL;
        basicRfConfig.panId = PAN_ID;
        basicRfConfig.ackRequest = TRUE;//;FALSE
         #ifdef SECURITY_CCM
        basicRfConfig.securityKey = key;
        #endif
        initRfTest();
        halLedSet(2);
        halLedSet(1);
        receiveMode();

 }
```

```
    #else       //送信モジュール
     {
            basicRfConfig.myAddr = ADDRESS_1;

        remoteAddr = ADDRESS_0;
        basicRfConfig.channel = RF_CHANNEL;
        basicRfConfig.panId = PAN_ID;
        basicRfConfig.ackRequest =TRUE ;//;FALSE
        #ifdef SECURITY_CCM
        basicRfConfig.securityKey = key;
        #endif
        initRfTest();
        halLedSet(2);
        halLedSet(1);
          c ontionuousMode();
   }
   #endif
}
```

2、送信データ関数

本関数では主に RF トランスミッタ関数 basicRfSendPacket()を呼び出し、配列 sendBuffer[]の値を送信し、送信が成功でしたらシリアルポートより本配列の値を PC へ送信する。

```
  void contionuousMode(void)
   {
  uint8 res;// BOOL
uint8 sendBuffer[5] ={1,2,3,4,5} ;//"Hello";BYTE   BYTE
if(halRfInit()==FAILED) {
  HAL_ASSERT(FALSE);
  }
  while(1)
     {

      halLedClear(1);
      if(basicRfInit(&basicRfConfig)==FAILED)
      {
      HAL_ASSERT(FALSE);
      }
      // Keep Receiver off when not needed to save power
      basicRfReceiveOff();
      halLedSet(2);
      res = basicRfSendPacket(remoteAddr,sendBuffer, sizeof(sendBuffer));
```

```
    halIntOff();
    halMcuSetLowPowerMode(HAL_MCU_LPM_3); // Will turn on global
            // interrupt enable
    halIntOn();
    basicRfReceiveOn();
  halMcuWaitMs(200);
    U0DBUF = res;
    while (!UTX0IF);
    UTX0IF = 0;
    //halWait(200);
    if(res == TRUE)
     {
     res =0;
     int j,m;
     m=sizeof(sendBuffer);
       halLedSet(1);
       for(j=0;j<m;j++)
     {
       U0DBUF =sendBuffer[j];
       while (!UTX0IF);
       UTX0IF = 0;
       //0X01;
        }
       j=0;
   halMcuWaitMs(200);
      }
       //halLedClear(2);
        halMcuWaitMs(200);
    }
  }
}
```

3、受信関数

本関数は主に basicRfReceive()を呼び出して無線データを受信する。データを受信後、シリアルポートより PC へ送信する。

```
void receiveMode(void)
  {
    uint8 receiveBuffer[]={0};
    uint8 length;
    uint8 ress;
    //BYTE sender;
    if(halRfInit()==FAILED) {
```

```
    HAL_ASSERT(FALSE);
  }

while(1)
 {
  halLedClear(2);
    if(basicRfInit(&basicRfConfig)==FAILED)
     {
    HAL_ASSERT(FALSE);
     }
  basicRfReceiveOn();
    halLedSet(1);
  while(!basicRfPacketIsReady());
    ress = basicRfReceive(receiveBuffer, length,NULL);
  //if(res>0)

  if(ress > 0)
  {
    int j;
    halLedSet(2);
    for(j=0;j<5;j++)
     {
    U0DBUF =receiveBuffer[j] ;//0X02;
    while (!UTX0IF);
    UTX0IF = 0;
     ress = 0;
  }
   j=0;
  halMcuWaitMs(200);
  }
  Else
   {
  //halLedClear(1);
  halMcuWaitMs(200);
}

}
//while (!UTX0IF);
//UTX0IF = 0;
```

```
  //U0DBUF = &receiveBuffer;
  }
```

7.5 実験手順

一、ハードウェアを接続後、プロジェクトを開く。example¥7_2530_P2P¥IAR/RF-test.eww。インターフェイスは下記図の通り：

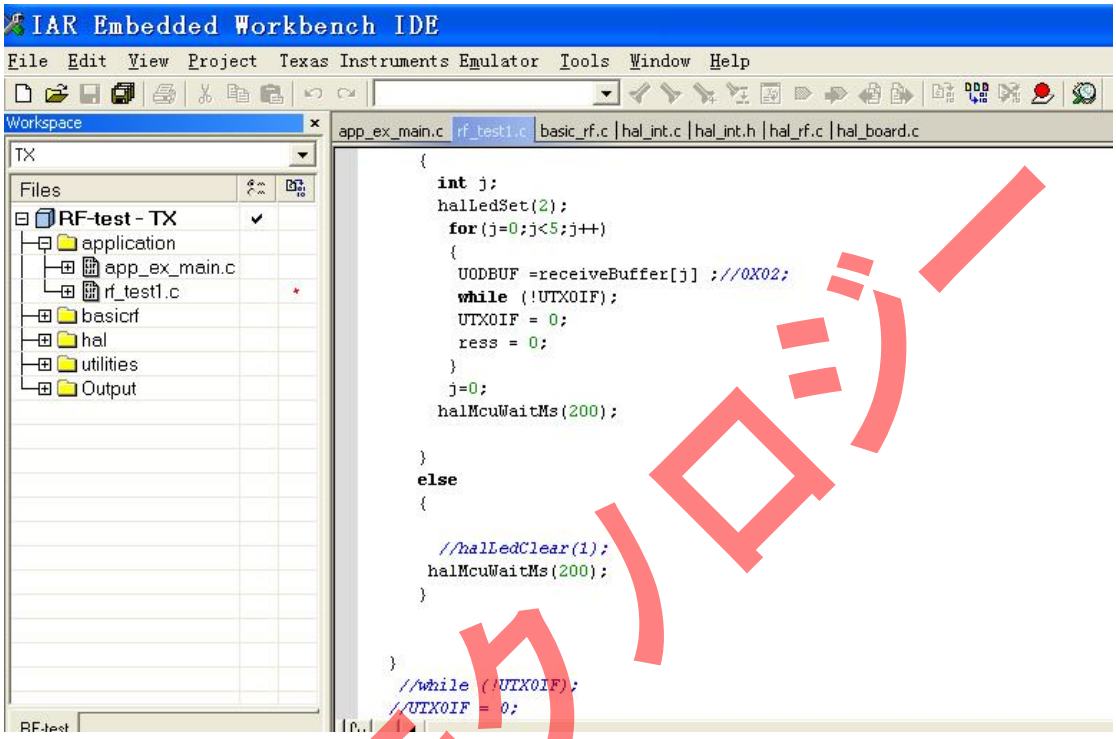


図 7－6

二、二つモジュールを RX/TX とする、workspace で RX/TX を選択して設定できる。図 7－7 をご参照ください。

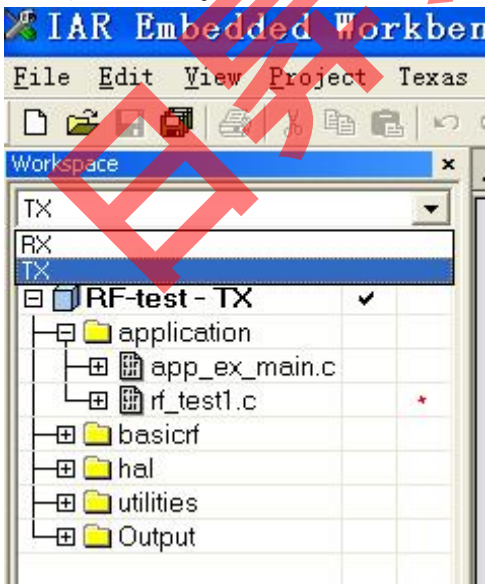


図 7－7

三、  をクリックしてプログラムをコンパイル、  をクリックしてリンク、最後に、  をクリックし開発ボードへダウンロードする。

四、プログラムをダウンロード後、IAR インターフェイスを終了し、物理アドレスのプログラミングする。スタートー/すべてのプログラムで Texas Instruments を見つけて、SmartRF Flash Programmer を開く、図 7－8 をご参照ください。

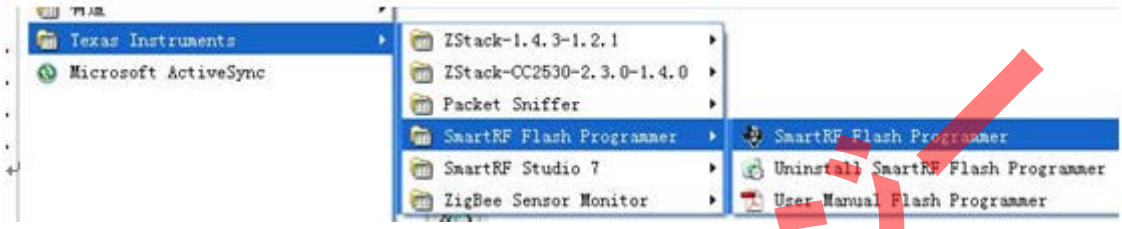


図 7－8

スタートインターフェイスは図 7－9 になる：

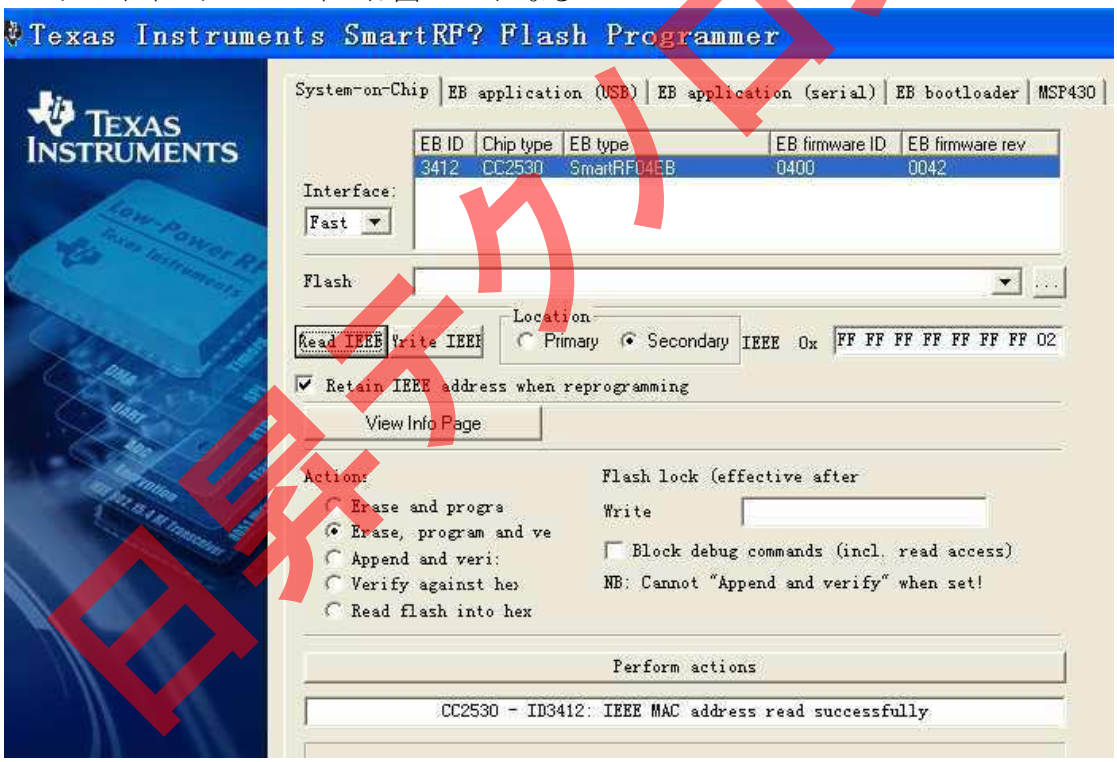


図 7－9

System-on-chip タブで Chip type に CC2530 を表示する。Read IEEE をクリックしてボートの IEEE アドレスを読み取り（IEEEox の後ろの 64 ビットアドレス)、そしてこのアドレスを指定のアドレスに変更する。注意： Location は Seconday を選択。Write IEEE をクリックし、修正後の IEEE アドレスをチップに書き込む。完了後、図 7－10 の示すように、Perform actions では CC2530 - ID3412: IEEE MAC address written successfully to chip を表示する。

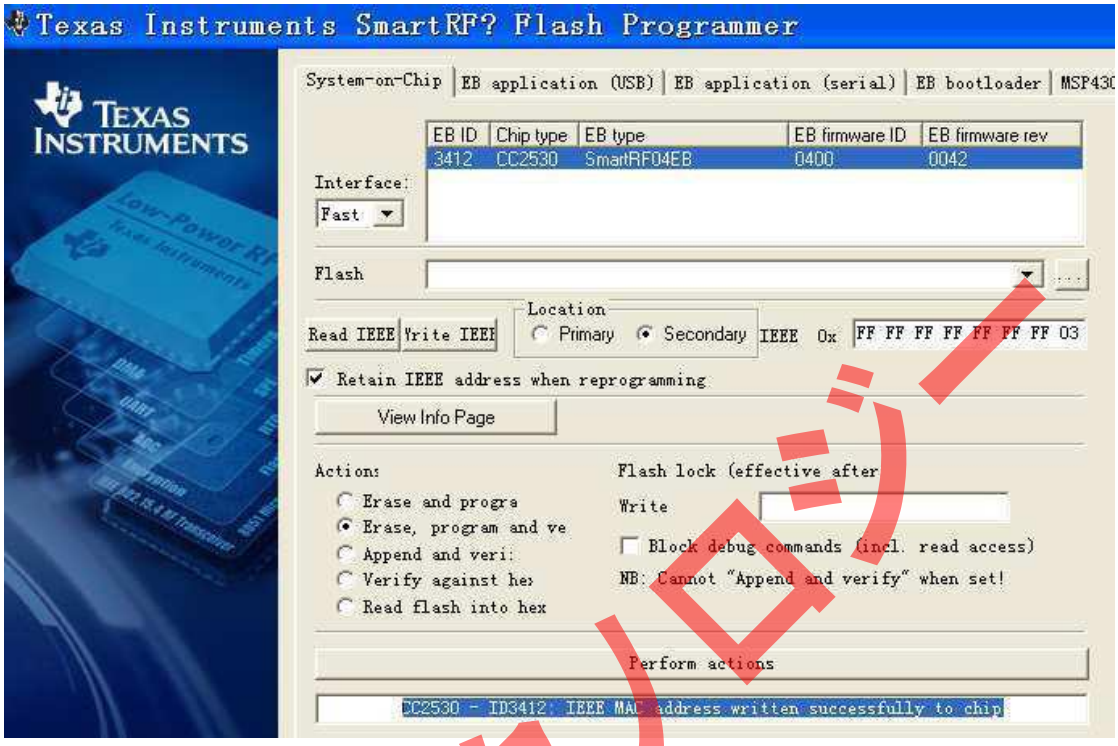


図 7－10

二つの試験ボートはプログラムをダウンロードして、上記の方法で物理アドレスを設定する。物理アドレスをプログラミング完了後、二つのボートをリセットして、試験現象を観察する。

7.6　実験現象

先ず送信モジュールを通電する。LED2 は点灯し、LED1 点滅状態になる。受信モジュールに通電する。送信モジュールの LED1 は点滅しない、シリアルポートはデータを送信しない、受信モジュールの LED1 は点灯し、LED2 は点滅する。シリアルポートのデータの送信状態は図 7－11 をご参照ください。

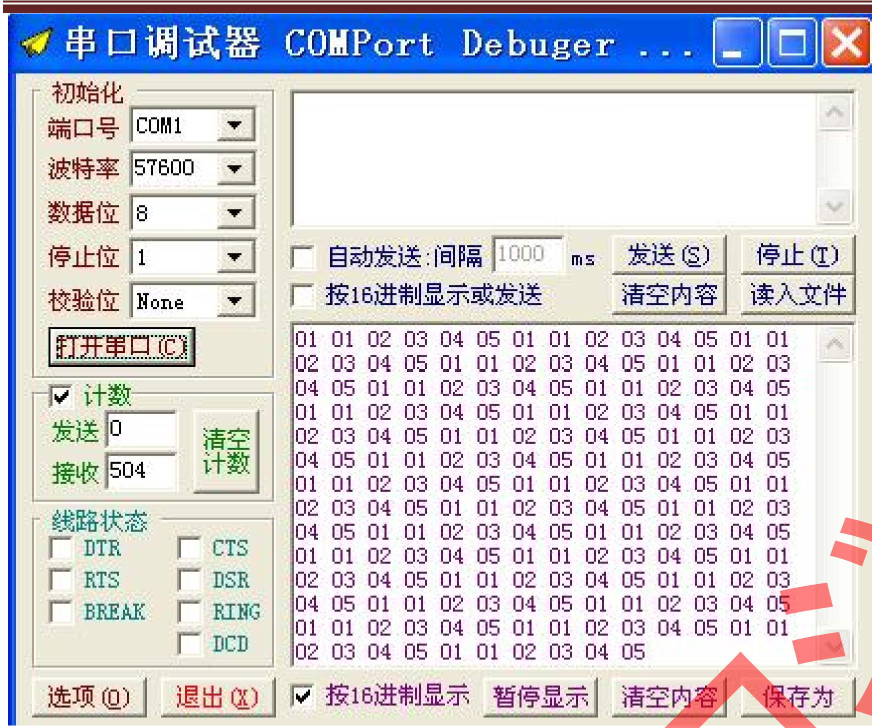


図 7－11

8　ZigBee2007 プロトコルスタックでシリアル透過伝送の実現

8.1　実験の目的

zstack-2007/pro プロトコルスタックの構造、シリアル透過伝送アプリケーションを把握し、zstack-2007/pro プロトコルスタックを基にニーズに応じて修正して、相応なアプリケーションを実現する。

8.2　実験の原理

プロトコルスタックの SerialApp サンプルを基づき CC2530 モジュール上無線透過伝送を実現する。

8.3　実験条件

CC2530 モジュール	×２（1＃、2＃）
エミュレータ	×２（ダウンロード・デバッガ、給電)、
シリアルケーブル	×２、

8.4　実験内容

シリアルポートで CC2530 モジュール 1＃にデータを送信する、本モジュールはワイヤレスでモジュール 2＃にデータを送信する、そしてモジュール 2＃はシリアルポートでデータを送信する。無線通信の透過伝送を実現する。

8.4.1　プロトコルスタック内容とディレクトリ紹介

プロトコルスタックを開き、プロジェクトファイルの左側 Workspace に全体プロトコルスタックのアーキテクチャが表示する。図 8－1 をご参照ください。

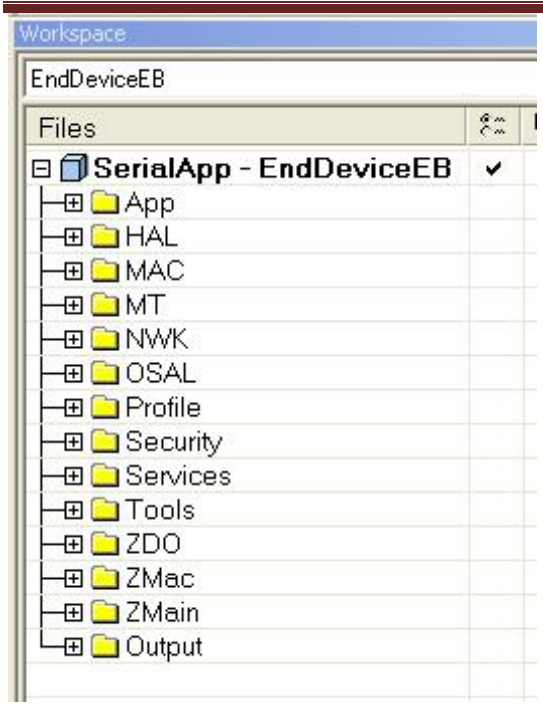


図 8－1

APP：アプリケーション層のディレクトリ、ユーザーがプロジェクトを作成するエリアです。アプリケーション層の内容とプロジェクトの重要内容を含める。プロトコルスタックでは OS のタスク機能により実現する場合が多い。

HAL：ハードウェア層のディレクトリ、ハードウェアに関するコンフィグレーションとドライバ及び操作関数を含める。

MAC：MAC 層のディレクトリ、MAC 層のパラメータコンフィギュレーションファイル及び LIB ライブラリ関数インタフェースファイルを含める。

MT：シリアルポートより各層を制御し、各層と直接通信する。

NWK：ネットワーク層のディレクトリ、ネットワーク層のパラメータ･ファイルとネットワーク層ライブラリの関数インタフェースのファイル、APS 層ライブラリの関数インターフェースを含める。

OSAL：プロトコルスタックのオペレーティングシステム

Profile：AF 層のディレクトリ、AF 層の関数処理ファイルを含める。

Security:セキュリティ層のディレクトリ､セキュリティ層処理関数､例えば暗号化関数など。

Services：アドレス処理関数ディレクトリ、アドレスモードの定義とアドレス処理関数を含める。

Tools：プロジェクトコンフィギュレーション･ディレクトリ、スペース分割と ZStack 関連の設定情報を含める。

ZDO：ZDO ディレクトリ

ZMac：MAC 層のディレクトリ、MAC 層パラメーター配置と MAC 層 LIB のライブラリ関数コールバックハンドラを含める。

ZMain：メイン関数ディレクトリ、エントリ関数とハードウェア・コンフィギュレーションファイルを含める。

Output：出力ファイルディレクトリ、EW8051 IDE 自動的に生成する。

8.4.2　重要な関数の紹介

プロトコルスタックのアプリケーションにおいて、ユーザーが修正必要なのは APP 層の内容と各自のハードウェアのニーズによって相応な HAL 層の内容のみです。APP 層では三つのファイルがあります。一つはシステムに関するファイル、ファイル名は OSAL を含め。一つはユーザーのアプリケーション関数ファイル。一つはヘッダファイル、この三つのファイルはユーザーのニーズで編集する。

プロトコルスタックのメイン関数は ZMain.c にある。メイン関数で各層とシステムの初期化し、最後に関数 osal_start_system()に入り、各タスクをサイクルにクエリー実行する。この関数においてポインタ関数配列(tasksArr[idx])(idx, events)を呼び出し、異なる idx を基づき相応な関数を実行する。本実験に処理するイベント関数は SerialApp_ProcessEvent()です。

メイン関数において初期化関数 osal_init_system()を呼び出し、操作システムを初期化し、この関数において osalInitTasks()関数は呼び出して、タスクを初期化し、この関数にアプリケーションタスクの初期化関数 SerialApp_Init(taskID)を加える。

操作システムとアプリケーション関数の間に OSAL_SerialApp.c ファイルの二つ関数で接続される。下記はアプリケーション層の関数を紹介する。

1、 SerialApp_Init(uint8 task_id)，この関数はシステムタスクが初期化する場合、呼び出され、アプリケーション層の実行タスクとハードウェアパラメータ配置を初期化する。task_id はシステムが配分するタスク ID です。

2、 SerialApp_ProcessEvent(uint8 task_id, UINT16 events)、この関数はアプリケーションタスクイベントを処理する機能で、異なるタスク ID より相応な事件を処理する。

3 、 SerialApp_HandleKeys(uint8 shift, uint8 keys)，ボタン処理関数、異なるボタンより相応な処理をし、この関数は SerialApp_ProcessEvent （） で呼び出される。

4、 SerialApp_ProcessMSGCmd(afIncomingMSGPacket_t *pkt)，この関数は他のデバイスから受信したデータを処理し、データ処理コールバック関数であり、クラスタ ID 値に基づいて相応なアクションを実行する。

またシリアルトランシーバなどの関数について、詳しいことはプログラムをご参照ください。

8.4.3　フローチャート

本実験のフローチャートは下記図 8－2 の通りであり：

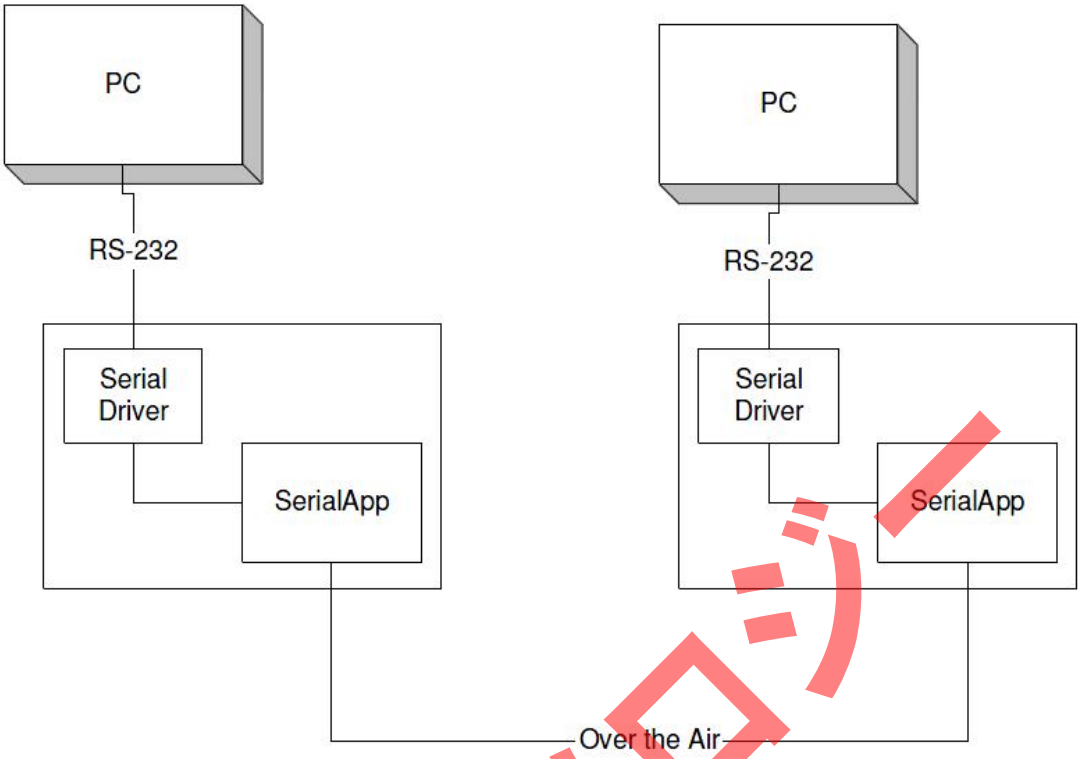


図 8－2

8.5　実験手順

1、Example¥8_2530_transparent-transmission¥フォルダにあるプロジェクトファイルを開く。

2、workspace 下で二つのモジュールをそれぞれ CoordinatorEB と RouterEB に設定する、図 8－3 をご参照ください。

図 8－3

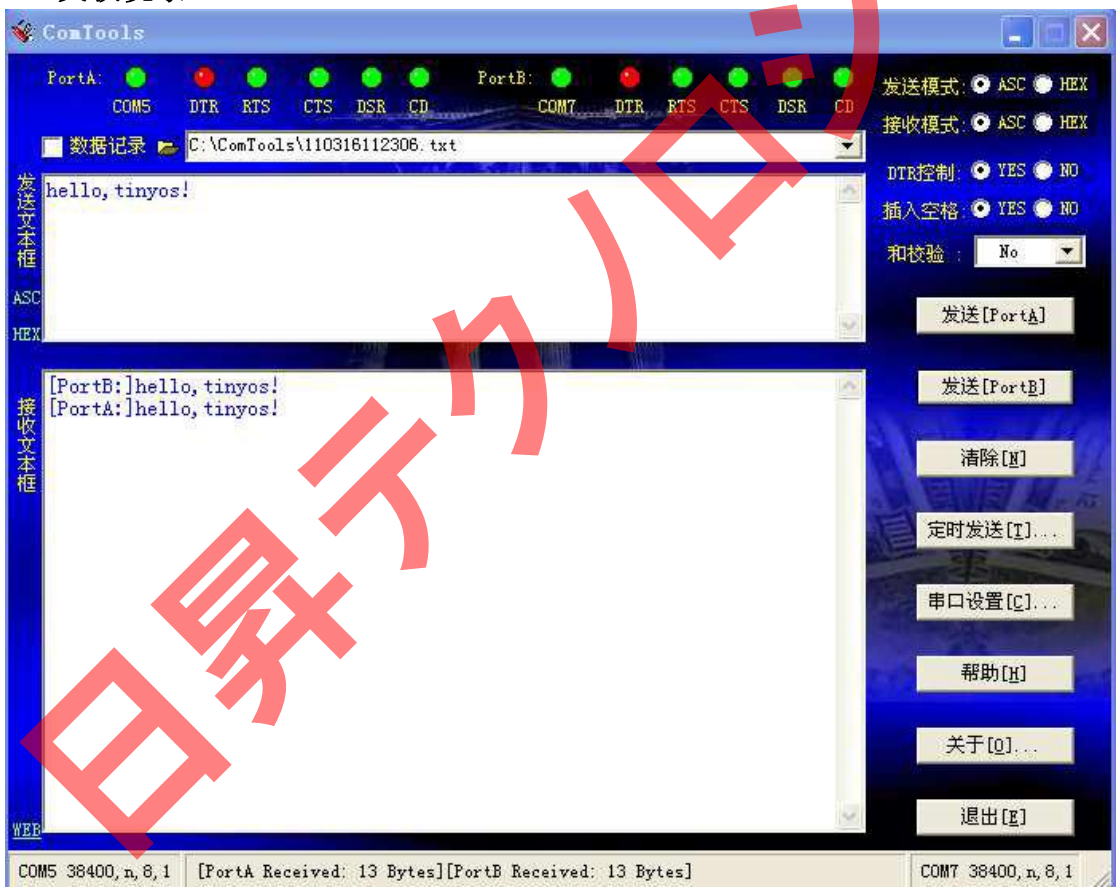
3、  をクリックしてプログラムをコンパイル、  をクリックしてリンク、最後に、  をクリックし開発ボードへダウンロードする。

4、二本の USB ケーブルでコーディネーター·デバイス、ターミナルデバイスをそれぞれ PC と接続する。二つシリアルデバッグツールを開き、相応なシリアルポートを選択し、ボーレートを設定し、38400，パリティビットなし、8 ビットデータ、1 ストップビット。

5、モジュールの左上の二つのスイッチを設定し、USB パワーサプライ設定は“ON,USB”と設定する。赤いディップスイッチは ON 状態に動かし、コーディネーターは先に通電する場合、LED2 が点灯し、ネットワークが成功、他のノードの加入待ちと示す。ルーターノードが先に通電する場合、指示灯が点灯しない、コーディネーターをオープン後、ルーターの LED2 も共に点灯し、ネットワークに加入が成功すると示す。

6、Comtools で二つポートからデータを送信し、対応するポートは送信したデータを受信する。

8.6　実験現象



以上。