

1 LFBacon の概要

本機は LFBacon と称し、LF 帯においてビーコン電波を発生するためのユニットです。一般にビーコンは長時間運用するので、PC によらない専用回路にして省電力化と操作の簡易化を図りました。

本機の機能は次のとおりです。

- CW キーヤ内蔵の 136.5kHz 信号発生
- コールサイン自動送しの 136.8kHz ビーコン発生
- WSPR2 モードの 137.490kHz ビーコン発生
GPS による時刻制御、4 分に 1 回の偶数分または 00 分および 30 分の選択、2 種のメッセージ実装
- WSPR15 モードの 137.612kHz ビーコン発生
GPS による時刻制御で 00 分および 30 分の送出、2 種のメッセージ
- DFCW モードの 137.776kHz ビーコン発生
RZ 符号による符号長の明確化

ハードウェア構成では、信号発生部に中華 DDS ユニット HC-SR08 を用い集積度を上げるとともにクロックを 12.288MHz 水晶発振ユニットに換装し周波数精度を上げました。クロックは PIC16F1823 の駆動にも共用します。

その制御部には PIC16F1823 の機能を有効利用しました。GPS 受信機からの正確な UTC 時刻で時刻管理をしており、WSPRX ビーコンの時刻精度は高いです。GPS 出力の NMEA 信号と 1PPS 信号は RS-232C レベルに変換し、PC など外部機器での利用に備えます。

GPS 受信機が必要とする 3.3V への降圧回路を含み、LFBacon ユニットは DC 5V 電源で動作します。

2 回路と実装

回路は図 1 のとおり簡単です。

HC-SR08 ユニットは、125MHz 水晶発振器を 12.288MHz 正方形発振器に換装した以外は変更を加えずカット周波数が高すぎる LPF も一応そのまま使用します。HC-SR08 に搭載されている VR は出力 LF の波形率が 50% となるように微調整します。

PTT 端子に接続されている ZD は、PIC16F1823 の RA4 端子電圧が電源電圧 5V を超えないことを考慮して選択します。

HC-SR08 ユニットの W_CLK にグリッチ雑音が混入すると DDS の周波数設定を誤りますので、キャパシタ 470pF を添えます。

LFBCN v1.50 (c) 2015 JA5FP

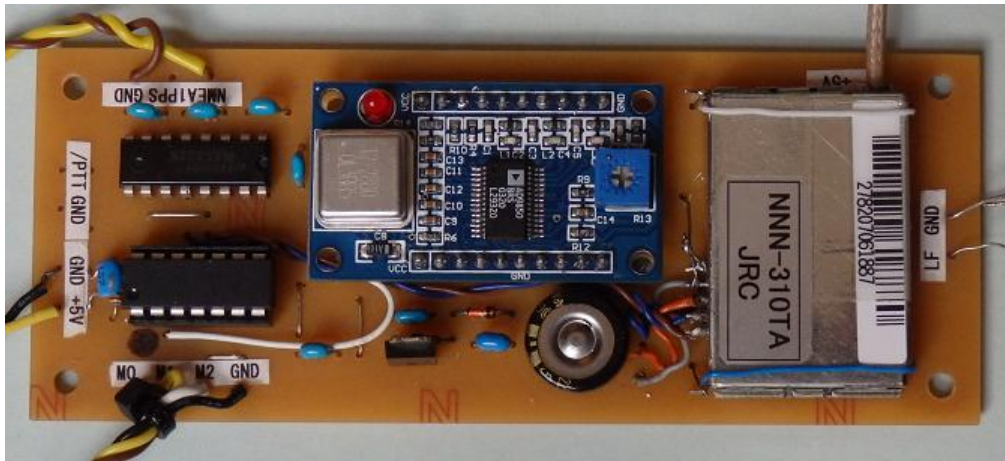
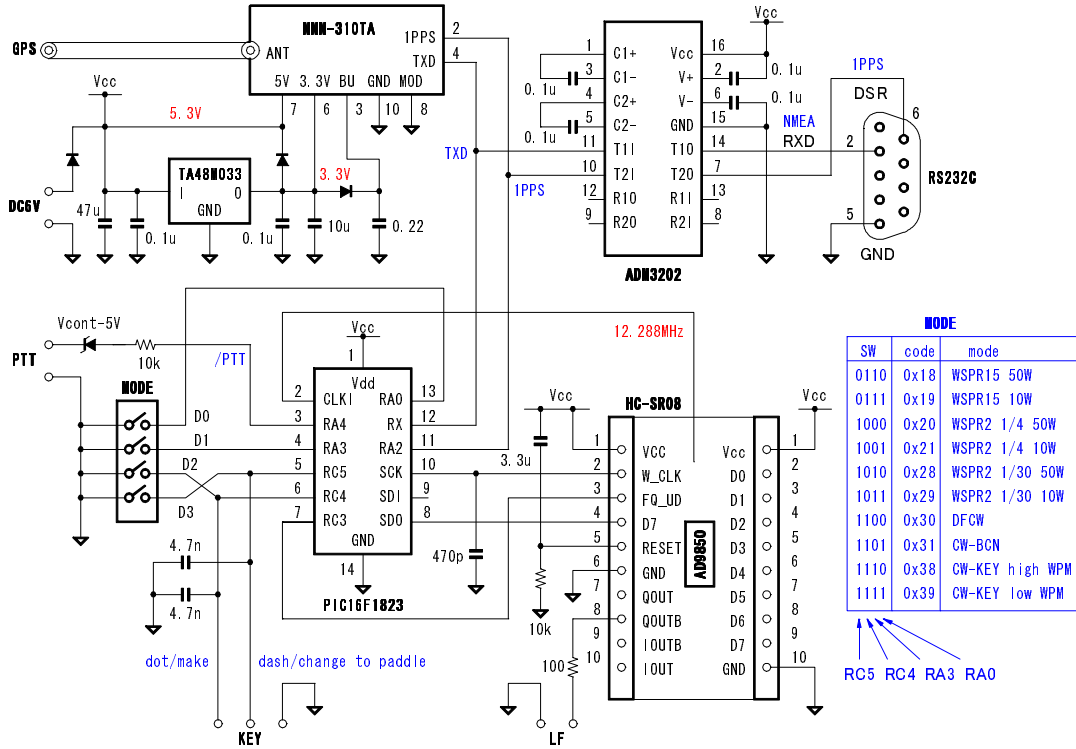


図 1: LFBeacon の回路と実装状態 (中央に水晶発振器を換装した HC-SR08 右は GPS 受信機)

3 出力

LFBeacon 出力電波を LF 帯受信機で実際に受信し、PC で復調した画像を図 2 に示します。いずれのモードも信号純度良く変調されて、完全にデコードできていることが分かります。

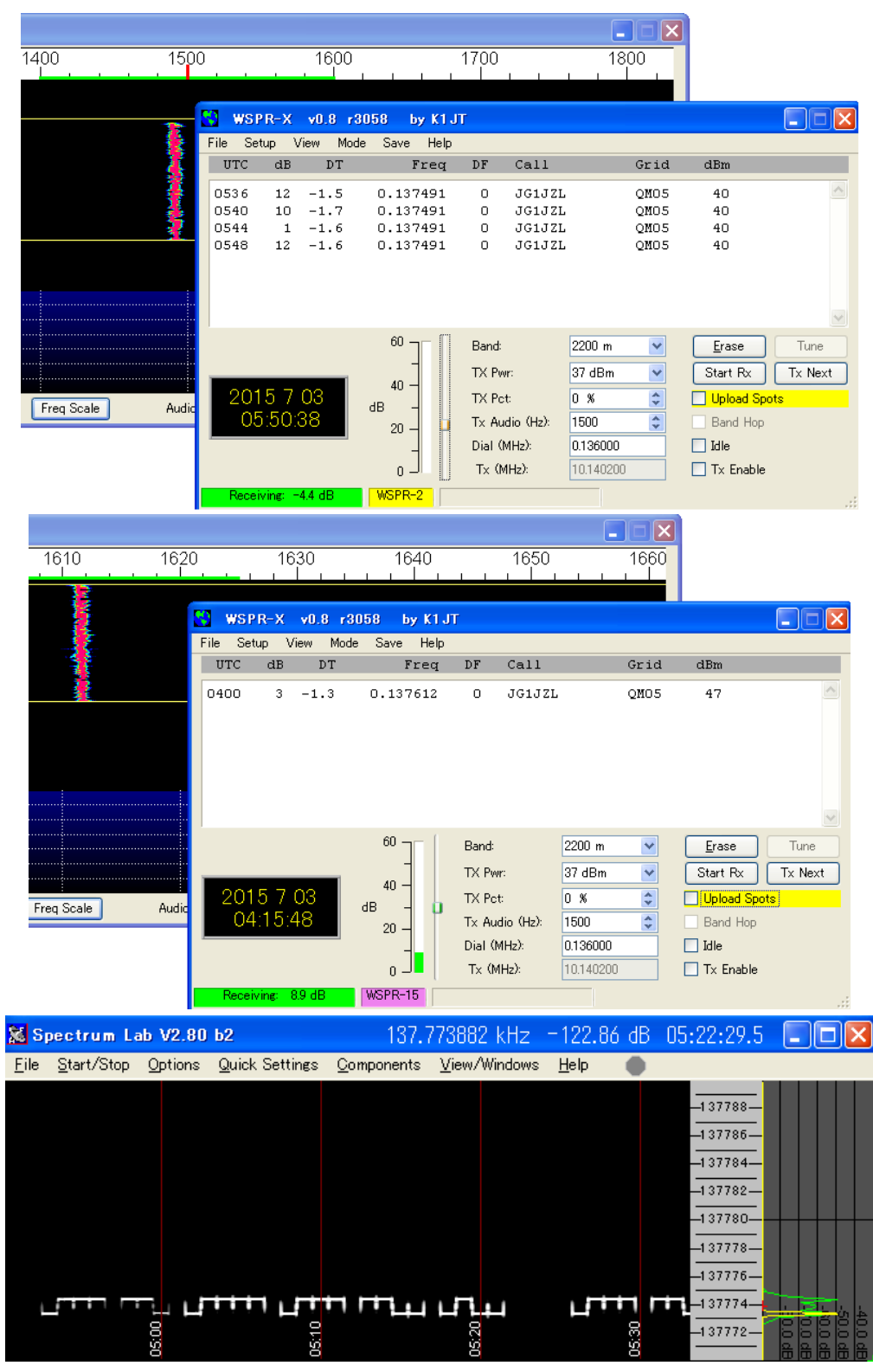


図 2: 上から WSPR2 モード WSPR15 モード DFCW モード

4 ソフトウェアの関数と機能

本機の機能は、PIC用のXC8Cコンパイラ言語で記述された関数の組み合わせで実現しています。まずその機能一覧は図3のとおりです。

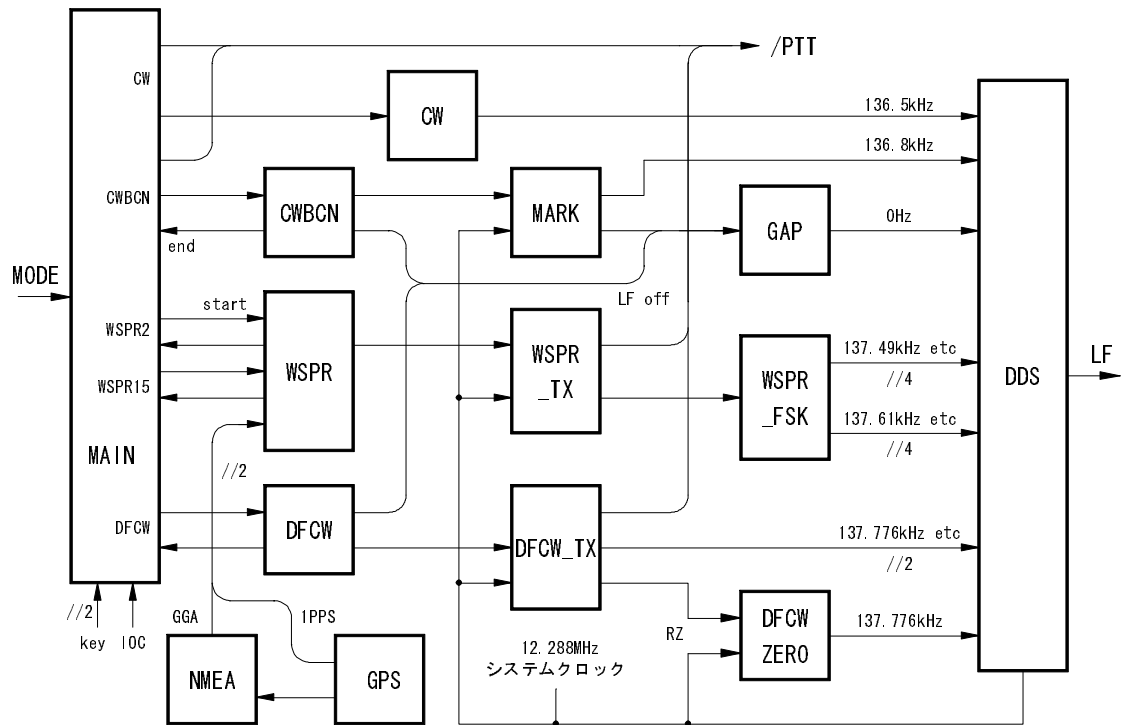


図 3: LFBacon を動かすための関数群とその依存関係

動作のあらすじは、関数 main() でモードの選択をします。モードは「内蔵キーヤ付き CW」「CW ビーコン」「WSPR2 1/30 MSG1」「WSPR2 1/30 MSG2」「WSPR2 1/4 MSG1」「WSPR2 1/4 MSG2」「WSPR15」「DFCW」の 8 個が用意されています。関数 main() が各モードごとの具体的な設定をする wspr() などの個別関数を呼出して、時刻待ちをします。さらに、関数 wspr_tx() などがエンコードをし、最後に関数 wspr_fsk() などが FSK や ASK 変調を DDS にかけて、成就します。

以下、プログラム作成上の工夫を中心に要点のみを解説しますので、図 3 と稿末のソースコードを参照してご覧ください。

4.1 GPS からの USART 通信

PIC16F1823 は 14 ピンですが、ミッドレンジ CPU とされる多様な機能のモジュールを持っています。

GPS 受信機が 4,800bps で吐出す NMEA データの取込みは、PIC16F1823 の EUSART モジュールを受信用に用いて実現できます。TXSTA、RCSTA、SPBRGL、BAUDCON などの関係 SFR をソースコードのように指定すれば特に問題はありません。NMEA メッセージは例えば JRC 製 GPS 受信機では、毎秒毎に発する "\$GPGGA,002233,3540.0790,N,... " というふうな内容の文字列です。(全文は、RS-232C を介してハイパーターミナルなどの通信ソフトで確認できます。)

取得すべき 1 文字は、ソースコードの関数 nmea() のとおり RCREG に収納されています。

EUSART モジュールは送信機能も含んでいますが、本機では受信だけで済みます。そこで送信用の端子 TX が I/O(RA0) として使えれば好都合です。Microchip 社のマニュアルや解説書では答えが見つからないのですが、実物で確認したところ EUSART 使用中であっても TXEN をオフにすれば TRISA の支配下になり I/O 端子とできることが分かりました。そこで、RA0 はモード設定の端子に使用します。

4.2 12.288MHz クロックとする理由

HC-SR08 の中核は Analog Devices の DDS シンセサイザ AD9850 です。その詳細な説明は次のデータシートを参照してください。

<http://www.analog.com/media/en/technical-documentation/data-sheets/AD9850.pdf>

要約しますと、(1) 最高 250MHz のクロック周波数 (2) 2^{32} の分解能 (3) シリアルまたはパラレル設定 (4) D/A コンバータ内蔵 (5) コンパレータ内蔵という優れた機能が集積されています。

折角ながら HC-SR08 に装着されている 125MHz 基準クロックは、LF 帯用途には必要以上に高すぎます。また、125MHz の PLL に伴う信号純度の低下を避けるために、クロックは 10MHz 台の水晶発振器にするのが得策です。そうすれば、DDS の基準クロックと PIC の外部入力クロックを共用することもできます。HC-SR08 の 125MHz 発振ユニットは表面実装ハンダ付けがされているので、ケース全体を十分に熱し、基板保護を優先して取外します。

具体的に適当な周波数を見極めるには、次のように思考します。

AD9850 における発振周波数を f 、周波数設定数値を n 、基準クロック周波数を f_c とすると、次式の関係があります。

$$f = \frac{n}{2^{32}} f_c$$

これに対応して WSPR15 のプロトコルである偏移周波数 $f_1 - f_0 = 12000/2^{16}$ を満たすためには、4 値 FSK の周波数を f_0, f_1, \dots とし、関連する周波数設定数値を n_0, n_1, \dots とすると、次式の関係が必要です。

$$f_1 - f_0 = \frac{(n_1 - n_0)}{2^{32}} f_c = \frac{12000}{2^{16}}$$

最小分解である $(n_1 - n_0) = 1$ の場合を計算しますと、 $f_c = 786.432\text{MHz}$ です。もちろんこの基準クロックは物理的に高すぎますので、その整数分の 1 を基準クロックとして代わりに最小分解数の整数倍でシフトする構成とします。PIC と水晶発振器の性能を総合的に見て、10MHz 台で 786.432MHz の整数分の 1 の周波数の水晶発振器が入手できればそれを使います。

本機では、12.288MHz を選択しました。その理由は、WSPR15 の場合は $(n_1 - n_0) = 64$ 、WSPR2 の場合は $(n_1 - n_0) = 512$ という 2 進法で区切りの良い数値となるからです。 $(n_2 - n_1)$ など同じ数値を使って周波数設定をします。具体的には、ソースコードの `wspr.fsk()` を見てください。

AD9850 の分解能の良さからすると、例えば 10.000Hz や 12.000MHz などの 786.432MHz と整数関係のない周波数でも十分な精度で周波数設定ができますが、計算に手間がかかります。

4.3 DDS の制御に SSP 通信

PIC16F1283 の端子数が制約されているので、DDS の周波数設定と制御はシリアル通信で行います。PIC16F1283 の SPI モードは (1) D7 端子へのデータ入力 (2) W_CLK 端子へのクロック (3) FQ_UD 端子へのアップデート入力の 3 端子で済みます。PIC16F1283 側では、クロックタイミングなどの手順が自動的に行われますので、ソフトウェアでは細部を手抜きできます。ただし、出力端子は SCK と SDO に特定されています。

AD9850 は電源投入後のイニシャル状態ではパラレル制御となっていますが、HC-SR08 の内部結線で W2=0、W1=1、W0=1 となっていますから、ソフトウェアで FQ_UD に 1 パルスを送れば以後シリアルデータを受付けます。パルス送出はソースコードの RA3=1 で行います。

AD9850 に 136.5kHz の周波数を設定する場合の数値計算は次のとおりです。

- (1) $136,500 \times 2^{32} / 12,288,000 = 47,710,208$ を得る。
- (2) 47,710,208 の 2 進数=0x02d80000 を計算する。
- (3) 逆順の 5 バイト=0x00,0x00,0x1b,0x40,0x00 を D7 に送る。

4.4 EEPROM へのデータ収納

PIC16F1823 の EEPROM は 256 バイトのメモリー空間があります。読出しはバイト単位でアクセスできます。

WSPR 送信に際して予め用意するメッセージコードは、<http://www.physics.princeton.edu/pulsar/K1JT/> からダウンロードした WSPRcode.exe で生成します。

1 メッセージ分のデータは 2 ビット 4 値構成で 162 個あります。これを EEPROM に収納するには、41 バイトが必要です。したがって PIC16F1823 の場合は、5 メッセージ程度を限度として記憶できます。本バージョンのソースコードでは 2 メッセージが書込み済みです。

図 4 が記憶域とデータの対応図です。1 バイト当たり 4 個の 4FSK データが得られます。結局、40.5 バイトで 162 個のデータを収納している訳です。

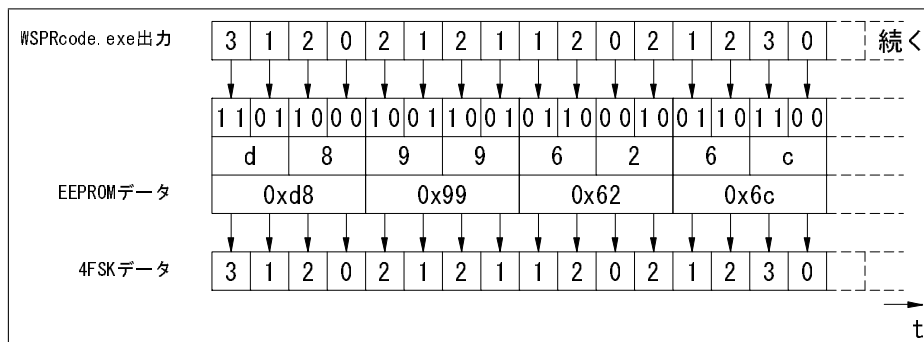


図 4: 2 ビットデータをバイト単位で EEPROM に保存、利用は 2 ビット単位

4.5 正確なタイミングの考慮

WSPRX のプロトコルでは符号長は、WSPR15 の場合は $2^{16} / 12000$ 秒、WSPR2 の場合は $2^{13} / 12000$ 秒に規定されています。この値に誤差があると、メッセージの送信最後では誤差は 162 倍に達しますので、正常にデコードできなくなるおそれが生じます。PIC16F1823 の外部クロックは水晶発振器の 12.288MHz が加えられて十分に安定ですが、ソースコード上で繰返しループを使う際の命令コード実行時間を最小限に抑えなければなりません。

本ソースコードでは、符号長を作成するためのタイマーをフリーランとし、命令コードとは無関係な時刻を取得しています。また、命令コードの実行を TMR1 の待機時間中に行うように考慮し、最短時間で周波数設定を行います。

AD9850 の周波数応答は 12 サイクル必要ですが、相対的な符号長には影響しないので差支えありません。

4.6 DFCW 信号の RZ 化

一般の DFCW 信号は NRZ 符号ですが、本機では RZ 符号として受信者が符号長を明瞭に区別できるように工夫しました。

NRZ 符号ではマーク (高周波数) またはスペース (低周波数) のデータが連続する場合に、何個連続しているのかが曖昧です。RZ 符号は符号の最初と最後にゼロ (中周波数) を必ず挿入しますので、図 2 で見るように、トランジション時のエコーで符号の連続を正確に計数できます。

4.7 入力端子の多重化

数的な限界がある PIC16F1823 の入力端子ですが、RA0、RC4 および RC5 をソフトウェアで二つの用途に兼用させます。なお、RA0 の関係は次項で述べます。

RC4=1 で RC5=1 のスイッチ状態として、一旦 CW-KEY モードのシーケンス (case 1111 または 1110) に入ってしまうと、RC4 と RC5 は自由に入力端子として使えるようになりますので、これらをエレキーのパドルに割当てます。したがって、このモードでは一般のパドル操作が行えることとなります。

加えて、エレキーとストレートキーの切替えを、パドル操作で行えるようにプログラムしました。エレキーでドット送出手を連続 10 個送出すると、RC4 はストレートキーの接点となり押続けるとキャリアは連続オンとなります。エレキーに戻るには、RC5 を一寸オンするだけで良いのです。

4.8 即時リセット

ビーコンモードでは 1 回の送信ルーチンが例えば WSPR15 では 15 分間に及びます。その間に別のモードで送信したい要求が生じる場合があるでしょう。そのために、本機ではモードスイッチの切替えによる RA0 の変化を検出し、強制リセットをかけます。ソースコードの関数 `chng()` が働いて、即時にモード変更ができます。

5 ソースコード

```
//"ddsbcn.c" ver.1r53 (c) 2015.07.10 JA5FP
//This program may be compiled with MPLAB XC8 C Compiler on MPLAB X IDE v2.20.
//Target hardware is the LFBeacon which produces continuous wave or various
//type of beacons.
//The functions are:
// (1)Selectable CW(136.5kHz), CW-beacon(136.8kHz), DFCW60(137.776kHz),
// WSPR2(137.490kHz) or WSPR15(137.612kHz).
// (2)WSPR2:Send pre-written messages (callsign, gridlocater, power).
// Controlled by correct GPS time sequence.
// Transmitt rate of once per 30 or 4 minute.
// (3)WSPR15:Begin transmitt at 00 and 30 minute.
//Now under building, changeable without notice.

#include <xc.h>
#include <pic16f1823.h>
#pragma config FCMEN=OFF, IESO=OFF, CLKOUTEN=OFF, BOREN=ON, CPD=OFF, CP=OFF
#pragma config MCLRE=OFF, PWRTE=ON, WDTE=OFF, FOSC=ECH
#pragma config LVP=OFF, BORV=LO, STVREN=OFF, PLEN=OFF, WRT=OFF

__EEPROM_DATA(0xda,0xaa,0x68,0x56,0x2e,0xbb,0xdc,0x00); // "JG1JZL QM05 40"
__EEPROM_DATA(0xac,0x93,0xaa,0x06,0x5a,0x59,0x09,0x6c);
__EEPROM_DATA(0x2b,0xc6,0x66,0x6b,0x86,0xf0,0x3e,0xe4);
__EEPROM_DATA(0x8e,0x0a,0x69,0xa5,0x6d,0xad,0x38,0x35);
__EEPROM_DATA(0x80,0x93,0x27,0x02,0x21,0x46,0x50,0x1c);
__EEPROM_DATA(0x0f,0xff,0xff,0xff,0xff,0xff,0xff,0xff);
__EEPROM_DATA(0xda,0x8a,0x68,0x56,0x2c,0x99,0xde,0x02); // "JG1JZL QM05 47"
__EEPROM_DATA(0xae,0x93,0x8a,0x26,0x5a,0x5b,0x09,0x4e);
__EEPROM_DATA(0x29,0xe6,0x44,0x69,0x84,0xd2,0x1e,0xc4);
__EEPROM_DATA(0x8c,0x08,0x4b,0x87,0x6f,0x8f,0x38,0x17);
__EEPROM_DATA(0x80,0xb1,0x27,0x00,0x21,0x64,0x70,0x3c);
__EEPROM_DATA(0x0f,0xff,0xff,0xff,0xff,0xff,0xff,0xff);

void device1823(void){ // hardware setting
    APFCON=0b10000100; //
    ANSELA=0x00; // digital portA
    ANSELC=0x00; // digital portC
    TRISA=0b00101111; // RA4=/PTT RA0=MODE
    // RA3=MODE RA2=1PPS RX
    TRISC=0b00110010; // SCK SDI SDO
    // RC3=FQ_UD RC4,RC5=MODE
    OPTION_REG=0b00000010; // WPU
    WPUA=0b00001001; // WPU portA
    WPUC=0b00110010; // WPU portC
    SSP1CON1=0b00100000; // SPI master mode
    SSP1STAT=0x00;
    SSP1CON3=0x00;
    T1CON=0x00;
    T1GCON=0x00;
    INTCON=0b10001001; // RA0 interrupt
    IOCAP=0b00000001; // positive change
    IOCAN=0b00000001; // negative change
    PIE1=0x00;
    PIE2=0x00;
    EECON1=0x00; // read only
    CCP1CON=0x00; // ECCP1 off
    TXSTA=0x00; // 8N
```



```

RCSTA=0b10010000; // asynchronous
SPBRGH=0x00; // n=Fosc/64*4800-1
SPBRGL=0x27; // n=39=0x27
BAUDCON=0x00; // low BRG
RA4=1; // /PTT off
RC2=0; // DDS serial mode
RC0=1; RC0=0; // W_CLK pulse
RC3=1; RC3=0; // FQ_UD pulse
} // device1823()

void gap(unsigned int term){ // LF off
    unsigned int iii; // counter
    SSP1BUF=0x00; while(!BF); // W0-W7
    SSP1BUF=0x00; while(!BF); // W8-W15
    SSP1BUF=0x00; while(!BF); // W16-W23
    SSP1BUF=0x00; while(!BF); // W24-W31
    SSP1BUF=0x00; while(!BF); // W32-W39
    RC3=1; RC3=0; // FQ_UD pulse
    switch(term){
        case 'n': break;
        default :
            TMR1H=0; TMR1L=0; // initiate TMR1
            TMR1IF=0; TMR1ON=1; // hold
            for(iii=0; iii<term; iii++){ // period=0.17s*term
                while(!TMR1IF); TMR1IF=0;
            } // for(term)
            TMR1ON=0;
        } // switch(term)
    } // gap(term)

void dfcw_zero(void){ // fz=137.776kHz
    unsigned char ccc; // counter
    SSP1BUF=0x54; while(!BF); // W0-W7
    SSP1BUF=0x73; while(!BF); // W8-W15
    SSP1BUF=0x7b; while(!BF); // W16-W23
    SSP1BUF=0x40; while(!BF); // W24-W31
    SSP1BUF=0x00; while(!BF); // W32-W39
    RC3=1; RC3=0; // FQ_UD pulse
    TMR1H=0; TMR1L=0; // initiate TMR1
    TMR1IF=0; TMR1ON=1;
    for(ccc=0; ccc<10; ccc++){ // hold 1.7 sec
        while(!TMR1IF); TMR1IF=0;
    } // for(10)
    TMR1ON=0;
} // dfcw_zero()

void dfcw_tx(unsigned char fsk2){ // 2FSK
    unsigned int iii; // counter
    dfcw_zero(); // fz
    switch(fsk2){
        case 'l': // fl=137.7755kHz
            SSP1BUF=0x3e; while(!BF); // W0-W7
            SSP1BUF=0xb3; while(!BF); // W8-W15
            SSP1BUF=0x7b; while(!BF); // W16-W23
            SSP1BUF=0x40; while(!BF); // W24-W31
            SSP1BUF=0x00; while(!BF); // W32-W39
            break;
        case 'h': // fh=137.7765kHz
            SSP1BUF=0x9b; while(!BF); // W0-W7

```

```

        SSP1BUF=0x73; while(!BF);           // W8-W15
        SSP1BUF=0x7b; while(!BF);           // W16-W23
        SSP1BUF=0x40; while(!BF);           // W24-W31
        SSP1BUF=0x00; while(!BF);           // W32-W39
        break;
    }// switch(fsk2)
    TMR1H=0; TMR1L=0;                       // initiate TMR1
    TMR1IF=0; TMR1ON=1;
    RC3=1; RC3=0;                           // FQ_UD pulse
    for(iii=0; iii<331; iii++){              // hold 56.5 sec
        while(!TMR1IF); TMR1IF=0;
    }// for(331)
    TMR1ON=0;
    dfcw_zero();                             // fz
    }// dfcw_tx(fsk2)
void dfcw(void){                             // encode 2FSK
    RA4=0;                                   // /PTT on
    dfcw_tx('l'); dfcw_tx('h'); dfcw_tx('h'); // 'J'
    dfcw_tx('h'); gap(351);
    dfcw_tx('h'); dfcw_tx('h'); dfcw_tx('l'); // 'G'
    gap(351);
    dfcw_tx('l'); dfcw_tx('h'); dfcw_tx('h'); // '1'
    dfcw_tx('h'); dfcw_tx('h'); gap(351);
    dfcw_tx('l'); dfcw_tx('h'); dfcw_tx('h'); // 'J'
    dfcw_tx('h'); gap(351);
    dfcw_tx('h'); dfcw_tx('h'); dfcw_tx('l'); // 'Z'
    dfcw_tx('l'); gap(351);
    dfcw_tx('l'); dfcw_tx('h'); dfcw_tx('l'); // 'L'
    dfcw_tx('l'); gap(1500);
    RA4=1;                                   // /PTT off
    }// dfcw()

void cwbcn_mark(unsigned char duty2){        // fm=136.8kHz
    unsigned char ccc;                       // counter
    SSP1BUF=0x59; while(!BF);                // W0-W7
    SSP1BUF=0x99; while(!BF);                // W8-W15
    SSP1BUF=0x9b; while(!BF);                // W16-W23
    SSP1BUF=0x40; while(!BF);                // W24-W31
    SSP1BUF=0x00; while(!BF);                // W32-W39
    RC3=1; RC3=0;                           // FQ_UD pulse
    TMR1H=0; TMR1L=0;                       // initiate TMR1
    TMR1IF=0; TMR1ON=1;
    for(ccc=0; ccc<duty2; ccc++){             // hold 0.34*duty
        while(!TMR1IF); TMR1IF=0;
    }// for(duty2)
    TMR1ON=0;
    gap(1);                                   // tail
    }// cwbcn_mark(duty2)
void cwbcn(void){                             // encode 2ASK
    RA4=0;                                   // /PTT on
    T1CKPS1=1; T1CKPS0=0;                   // TMR1 prescaler=1/4
    cwbcn_mark(1); cwbcn_mark(3); cwbcn_mark(3); // 'J'
    cwbcn_mark(3); gap(2);
    cwbcn_mark(3); cwbcn_mark(3); cwbcn_mark(1); // 'G'
    gap(2);
    cwbcn_mark(1); cwbcn_mark(3); cwbcn_mark(3); // '1'
    cwbcn_mark(3); cwbcn_mark(3); gap(2);
    cwbcn_mark(1); cwbcn_mark(3); cwbcn_mark(3); // 'J'

```

```

    cwbcn_mark(3); gap(2);
    cwbcn_mark(3); cwbcn_mark(3); cwbcn_mark(1); // 'Z'
    cwbcn_mark(1); gap(2);
    cwbcn_mark(1); cwbcn_mark(3); cwbcn_mark(1); // 'L'
    cwbcn_mark(1); gap(100);
    RA4=1; // /PTT off
} // cwbcn()

unsigned char nmea(void){
    if(FERR) RCREG; // frame error
    if(OERR){ CREN=0; CREN=1; // overload error
    } // if(error)
    while(!RCIF); // fill RCREG
    return(RCREG); // clear RCREG
} // nmea()

void wspr_fsk(unsigned char wsprx2, unsigned char shift4){
    if(wsprx2==0x01){ // WSPR2
        switch(shift4){ // 4FSK
            case 0x00: // f0=137.48854kHz
                SSP1BUF=0x75; while(!BF); // W0-W7
                SSP1BUF=0xa2; while(!BF); // W8-W15
                SSP1BUF=0xbb; while(!BF); // W16-W23
                SSP1BUF=0x40; while(!BF); // W24-W31
                SSP1BUF=0x00; while(!BF); // W32-W39
                break;
            case 0x40: // f1=137.49000kHz
                SSP1BUF=0x75; while(!BF); // W0-W7
                SSP1BUF=0xe2; while(!BF); // W8-W15
                SSP1BUF=0xbb; while(!BF); // W16-W23
                SSP1BUF=0x40; while(!BF); // W24-W31
                SSP1BUF=0x00; while(!BF); // W32-W39
                break;
            case 0x80: // f2=137.49146kHz
                SSP1BUF=0x75; while(!BF); // W0-W7
                SSP1BUF=0x92; while(!BF); // W8-W15
                SSP1BUF=0xbb; while(!BF); // W16-W23
                SSP1BUF=0x40; while(!BF); // W24-W31
                SSP1BUF=0x00; while(!BF); // W32-W39
                break;
            case 0xc0: // f3=137.49293kHz
                SSP1BUF=0x75; while(!BF); // W0-W7
                SSP1BUF=0xd2; while(!BF); // W8-W15
                SSP1BUF=0xbb; while(!BF); // W16-W23
                SSP1BUF=0x40; while(!BF); // W24-W31
                SSP1BUF=0x00; while(!BF); // W32-W39
                break;
        } // switch(shift4)
    } // if(wsprx2)
    if(wsprx2==0x08){ // WSPR15
        switch(shift4){ // 4FSK
            case 0x00: // f0=137.61200kHz
                SSP1BUF=0x02; while(!BF); // W0-W7
                SSP1BUF=0x77; while(!BF); // W8-W15
                SSP1BUF=0xbb; while(!BF); // W16-W23
                SSP1BUF=0x40; while(!BF); // W24-W31
                SSP1BUF=0x00; while(!BF); // W32-W39
                break;

```

```

        case 0x40:                // f1=137.61218kHz
            SSP1BUF=0x01; while(!BF); // W0-W7
            SSP1BUF=0x77; while(!BF); // W8-W15
            SSP1BUF=0xbb; while(!BF); // W16-W23
            SSP1BUF=0x40; while(!BF); // W24-W31
            SSP1BUF=0x00; while(!BF); // W32-W39
            break;
        case 0x80:                // f2=137.61238kHz
            SSP1BUF=0x03; while(!BF); // W0-W7
            SSP1BUF=0x77; while(!BF); // W8-W15
            SSP1BUF=0xbb; while(!BF); // W16-W23
            SSP1BUF=0x40; while(!BF); // W24-W31
            SSP1BUF=0x00; while(!BF); // W32-W39
            break;
        case 0xc0:                // f3=137.61256kHz
            SSP1BUF=0x00; while(!BF); // W0-W7
            SSP1BUF=0xf7; while(!BF); // W8-W15
            SSP1BUF=0xbb; while(!BF); // W16-W23
            SSP1BUF=0x40; while(!BF); // W24-W31
            SSP1BUF=0x00; while(!BF); // W32-W39
            break;
    } // switch(shift4)
} // if(wspr15)
} // wspr_fsk(wsprx2, shift4)
void wspr_tx(unsigned char wsprx2, unsigned char adr41){
    unsigned char bits8; // data file
    unsigned char sel4, sel40, ccc; // counter
    asm("        BANKSEL(_PORTA)");
    asm("rise0   BTFSS   _PORTA,2"); // find 00 sec
    asm("        goto   rise0");
    asm("fall0   BTFSC   _PORTA,2");
    asm("        goto   fall0");
    RA4=0; // /PTT on
    sel40=39; //
    EEADRL=adr41; // 0x00 or 0x30
    RD=1; bits8=EEDATL;
    wspr_fsk(wsprx2, bits8&0xc0); // 1st and 2nd bits
    TMR1IF=0; TMR1H=0x00; TMR1L=0x00; // n=65536
    asm("rise1   BTFSS   _PORTA,2"); // find 01 sec
    asm("        goto   rise1");
    asm("fall1   BTFSC   _PORTA,2");
    asm("        goto   fall1");
    TMR1ON=1;
    do{
        sel4=3;
        do{
            RC3=1; RC3=0; // start DDS
            for(ccc=1; ccc<(wsprx2*4); ccc++){ // term
                while(!TMR1IF); TMR1IF=0;
            } // for(wsprx2*4)
            bits8*=4; // next 2 bits
            if(sel4==0){ // next byte
                adr41++; EEADRL=adr41; // address
                RD=1; bits8=EEDATL; // data
            } // if(sel4)
            wspr_fsk(wsprx2, bits8&0xc0); // 3rd to 8th bits
            while(!TMR1IF); TMR1IF=0; // 0.683 or 5.461 sec
        } while(sel4--); // 109 or 874 sec
    }
}

```

```

        }while(sel40--);
RC3=1; RC3=0; // newer freq
for(ccc=1; ccc<(wsprx2*4); ccc++){
    while(!TMR1IF); TMR1IF=0;
    }// for(wsprx2*4)
bits8*=4; // next 2 bits
wspr_fsk(wsprx2, bits8&0xc0);
while(!TMR1IF); TMR1IF=0; // 0.683 or 5.461 sec
RC3=1; RC3=0; // newest freq
for(ccc=0; ccc<(wsprx2*4); ccc++){
    while(!TMR1IF); TMR1IF=0;
    }// for(wsprx2*4) // 0.683 or 5.461 sec
gap(1);
TMR1ON=0;
RA4=1; // /PTT off
} // wspr_tx(wsprx2,adr41)
void wspr(unsigned char wsprx2, unsigned char rate2, unsigned char msg5){
    unsigned char ccc; // character
    while(1){
        while(nmea()!='G') continue; // find 'G'
        if(nmea()=='G'&&nmea()=='A') break; // find "GA"
        }// while(1)
    for(ccc=0; ccc<3; ccc++) nmea(); // over look ".hh"
    ccc=nmea(); // get m*
    switch(rate2){
        case 'r': {
            if(ccc=='0' || ccc=='3'){ // seek m*=0 or 3
                if(nmea()=='0') wspr_tx(wsprx2, msg5); // seek *m=0
                }// if()
            break;
        }// case 'r'
        case 'm': {
            if(ccc&=0x01){ // seek m*=***1
                ccc=nmea(); // seek mm=12,16,32,
                if(ccc=='2' || ccc=='6') // 36,52,56
                    wspr_tx(wsprx2, msg5); break;
                }// if(odd)
            if(ccc==0x00){ // seek m*=***0
                ccc=nmea(); // seek mm=00,04,20,24,
                if(ccc=='0' || ccc=='4' || ccc=='8') // 28,40,44,48
                    wspr_tx(wsprx2, msg5);
                }// if(even)
            }// case 'm'
        }// switch(rate2)
    }// wspr(wsprx2,rate2,msg5)

void cw_mark(unsigned char term3){ // 136.5kHz
    unsigned char ccc; // counter
    SSP1BUF=0x00; while(!BF); // W0-W7
    SSP1BUF=0x00; while(!BF); // W8-W15
    SSP1BUF=0x1b; while(!BF); // W16-W23
    SSP1BUF=0x40; while(!BF); // W24-W31
    SSP1BUF=0x00; while(!BF); // W32-W39
    RC3=1; RC3=0;
    if(term3==1 || term3==3){
        TMR1H=0; TMR1L=0; // initiate TMR1
        TMR1IF=0; TMR1ON=1;
        for(ccc=0; ccc<term3; ccc++){ // hold 0.34*duty

```

```

        while(!TMR1IF); TMR1IF=0;
    }// for(duty2)
    }// if()
    }// cw_mark(term3)
void cw(unsigned char wpm2){
    unsigned char key2='p'; // key=paddle
    unsigned char dots; // counter
    RA4=0; gap(2); // /PTT wait contact
    T1CKPS1=1; T1CKPS0=0; // high WPM
    if(wpm2=='1') T1CKPS0=1; // low WPM
    while(1){
        switch(key2){
            case 'p': if(RC4) dots=0;
                    if(!RC4){
                        cw_mark(1); gap(1); // "dot"
                        if(dots++>11) key2='s'; // change to straight
                    }// if(!RC4)
                    if(!RC5){
                        cw_mark(3); gap(1); // "dash"
                    }// if(!RC5)
                    break;
            case 's': if(!RC5) key2='p'; // change to paddle
                    if(!RC4){
                        cw_mark('f'); // "mark"
                        while(!RC4); gap('n'); // "space"
                    }// if(!RC4)
                    break;
        }// switch(key2)
    }// while(1) // infinite
}// cw(wpm)

void main(void){
    unsigned char mode; // variable
    device1823(); // interface
    mode=(PORTA&0b00001001); // switch state
    mode+=(PORTC&0b00110000); //
    T1CKPS1=1; T1CKPS0=1; // TMR1 prescaler 1/8
    while(1){
        switch(mode){ // select sequence
            case 0x39: cw('l'); break; // CW-KEY low WPM
            case 0x38: cw('h'); break; // CW-KEY high WPM
            case 0x31: cwbcn(); break; // CW-BCN
            case 0x30: dfcw(); break; // DFCW60
            case 0x29: wspr(0x01, 'r', 0x00); break; // WSPR2 1/30, 10W
            case 0x28: wspr(0x01, 'r', 0x30); break; // WSPR2 1/30, 50W
            case 0x21: wspr(0x01, 'm', 0x00); break; // WSPR2 1/4, 10W
            case 0x20: wspr(0x01, 'm', 0x30); break; // WSPR2 1/4, 50W
            case 0x19: wspr(0x08, 'r', 0x00); break; // WSPR15, 10W
            case 0x18: wspr(0x08, 'r', 0x30); break; // WSPR15, 50W
            default: ; // inop
        }// switch(mode)
    }// while(1)
}// main()

void interrupt chng(void){ // reset when
    if(IOCIF==1) IOCAF0=0; // mode changed
    gap(1);
    PCL=0x00; // restart
}

```

}// chng()

6 参考文献

- WSPR 2.0 ユーザーガイド:physics.princeton.edu/pulsar/K1JT/WSPR_2.0_User_Japanese.pdf
- WSPR 3.0 User's Guide:physics.princeton.edu/pulsar/K1JT/WSPR_3.0_User.pdf
- WSPR-X User's Guide:physics.princeton.edu/pulsar/K1JT/WSPR-X_Users_Guide.pdf
- Play the C 初級 C 言語講座 [上巻][下巻]:林 晴比古 著 日本ソフトバンク
- PIC16F 活用ガイドブック:後閑 哲也 著 技術評論社
- MPLAB® XC8 Compiler User's Guide:Microchip Technology Inc.
- MPLAB® X IDE User's Guide:Microchip Technology Inc.
- PIC12(L)F1822/PIC16(L)F1823 Data Sheet:Microchip Technology Inc.