LF 帯多モードビーコン信号発生器

.....PIC16F1823 AD9833 GYSFDMAXB .....

1 LF ビーコンの概要

LF(136kHz) 帯において CW・WSPR・DFCW のビーコン電波または手動 CW 電波を発生するための励振ユニットです。その機能と特徴は次のとおりです。

- CW キーヤ内蔵の 136.5kHz 信号発生
- コールサイン自動送出の CW 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
- DC12V 電源を使用
  商用電源のない地域または車載運用を考慮

2 回路と実装

ハードウェア構成は、信号発生部に DDS ユニット GY9833 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。

それの制御部には 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、WSPR ビーコンの時刻精度は高いです。これには GPS 出力の NMEA 信号を使用します。

GY9833 の出力はサイン波形なので、D 級アンプを励振するには矩形波への変換回路を挿入す必要があります。それは LM393 などのコンパレータで実現できます。

ビーコン信号発生部の回路は図 1 のとおりです。図中の GPS ユニットに組まれているダイオードマトリックスは、電源の短絡防止回路です。

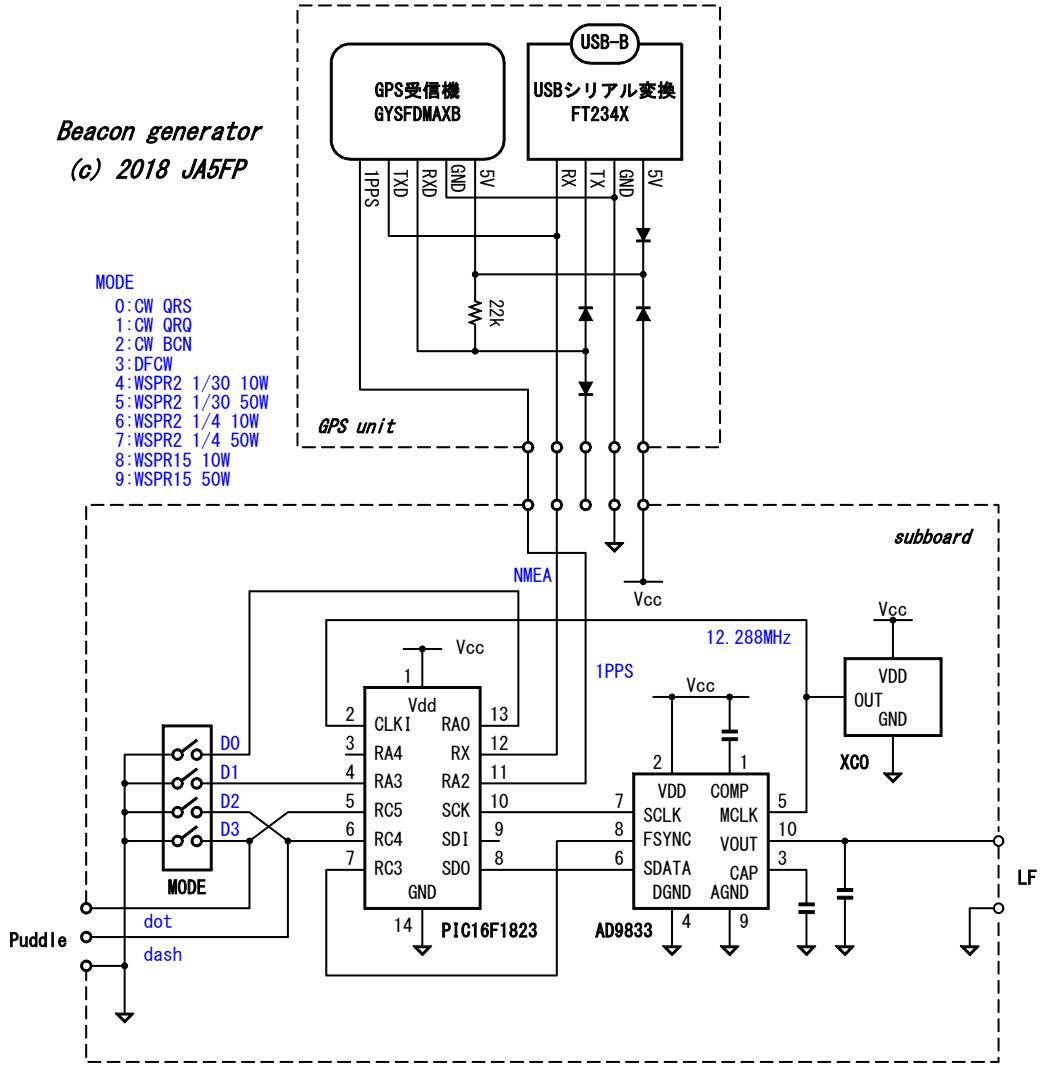

図 1: GPS による時刻管理と信号発生器の回路

3　出力

本機の出力電波を 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。

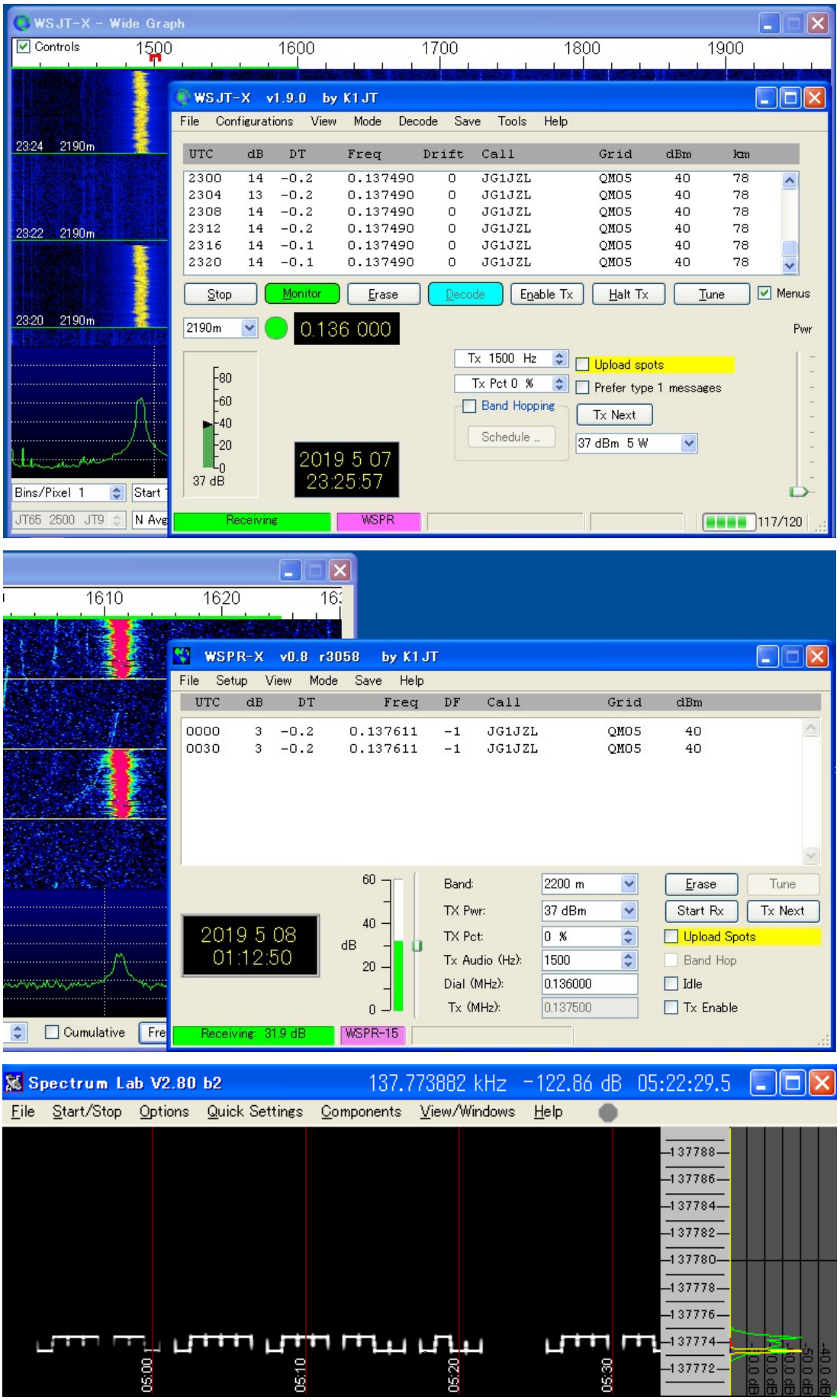

図 2: 上から WSPR2 モード WSPR15 モード DFCW モード

4　ソフトウェアの関数と機能

本機の機能は、PIC 用の XC8 C 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。

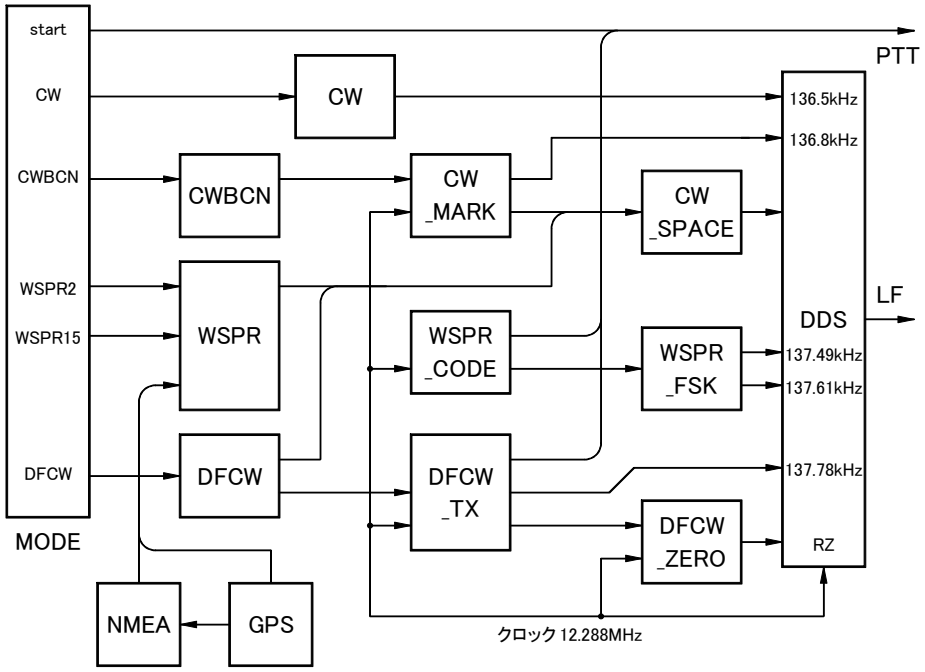

図 3: PIC を動かすための関数群とその依存関係

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_code() 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 と別項のソースコードを参照してご覧ください。

4.0.1　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 受信機に太陽誘電製のモジュール GYSFDMAXB を使用します。この受信機は 3.3V 電源電圧ですが、PIC16F1823 を電源電圧 5V で使用しても入力レベルの許容範囲ですので、直接接続できます。GYSFDMAXB はデフォルトの設定のままとし、UART 通信速度は 9,600bps になります。受け側は PIC16F1823 の EUSART モジュールの受信部分を使います。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なく動作す

るでしょう。

GPS 受信機が出力する NMEA メッセージは、図 4 のデータが毎秒毎に連続して現れます。この製作では、8 行目の GPRMC センテンスを使います。

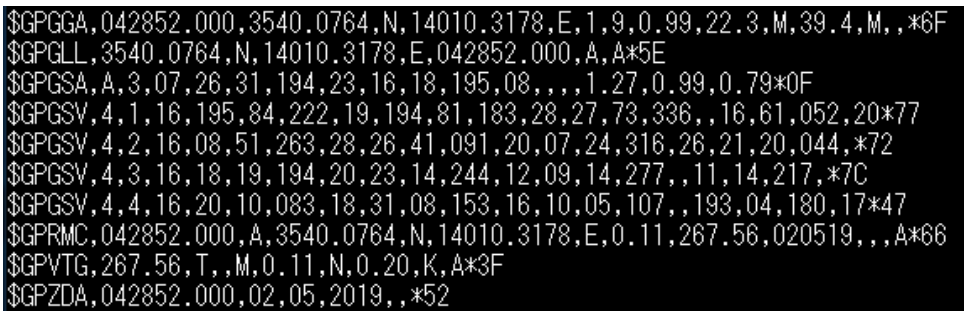

図 4: NMEA メッセージの例

EUSART で転送された NMEA データは、PIC16F1823 の RCREG に収納されます。その連続データから WSPR 送信開始時刻である特定の時分を知るには、GPRMC に続く 5 文字目 (図 4 で言えば、`$GPRMC,0428` の最後の文字である 8) を取得しなければなりません。そのシーケンスは図 5 の手順を経ます。

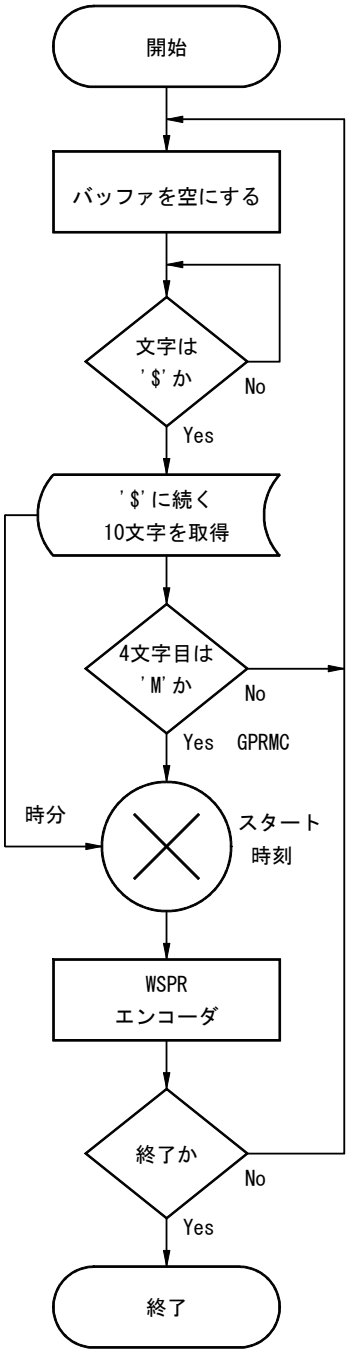

図 5: NMEA メッセージから特定の文字 (時分) で WSPR をスタートする手順

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足ります。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 端子化できることが分かりました。そこで、RA0 はモード設定の入力端子に使用します。

4.0.2 12.288MHz クロックとする理由

GY9833 の中核は Analog Devices の DDS シンセサイザ AD9833 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/AD9833.pdf

要約しますと、(1) 最高 40MHz のクロック周波数 (2)2^{28} の分解能 (3) シリアル設定 (4)D/A コンバ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 GY9833 に装着されている 25MHz 基準クロックは、LF 帯用途には必要以上に高すぎます。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。GY9833 の 25MHz 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。

AD9833 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。

$$f = \frac{n}{2^{28}} f_c$$

これに対応して WSPR2 のプロトコルである偏移周波数 $f_1 - f_0 = 12000/2^{13}$ を満たすためには、4 値 FSK の周波数を f_0、f_1、... とし、関連する周波数設定数値を n_0、n_1、... とすると、次式の関係が必要です。

$$f_1 - f_0 = \frac{(n_1 - n_0)}{2^{28}} f_c = \frac{12000}{2^{13}}$$

最小分解である $(n_1 - n_0) = 1$ の場合を計算しますと、$f_c = 786.432MHz$ 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、$2.4576MHz$ 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。$(n_2 - n_1)$ なども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。

AD9833 の分解能の良さからすると、例えば $10.000Hz$ や $12.000MHz$ などの $786.432MHz$ と整数関係に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。

4.0.3 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) 端子へのデータ入力 (2) 端子へのクロック (3)FSYNC 端子へのアップデート入力の 3 端子で済みます。PIC16F1823 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。

AD9833 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)$136,500 \times 2^{28}/12,288,000 = 2,981,888$ を得る。

(2)2,981,888 の 16 進数=0x02d80000 を計算する。

(3) その 28 ビットを上位 14 ビットと下位 14 ビットに分割する。

(4) 上位に 2 ビット (周波数レジスタへのアドレスである 0b01) を付加して
MSB および LSB とする。

(5)LSB を最初に D7 に送り、次に MSB を送る。

4.1 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 6 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。

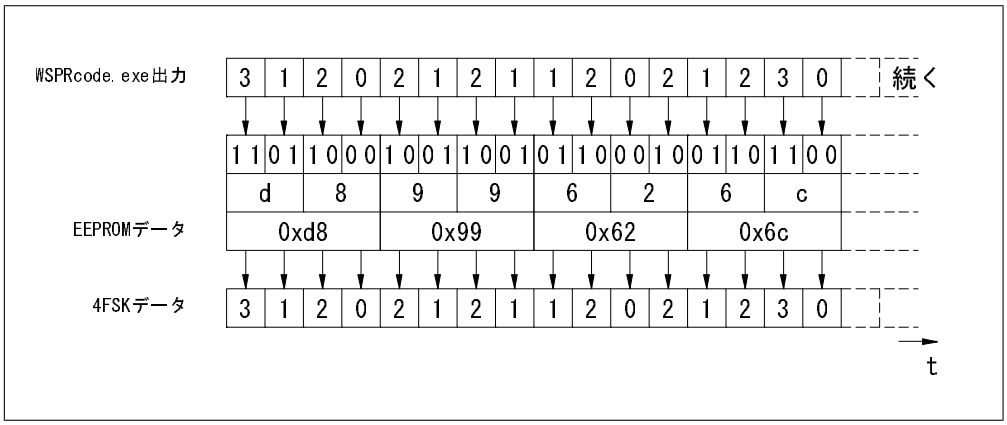

図 6: 2 ビットデータをバイト単位で EEPROM に保存、利用は 2 ビット単位

4.1.1 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 $12.288MHz$ が加えられで十分に安定ですが、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。

AD9833 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ないです。

4.1.2 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。

4.1.3　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になります。

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。

4.1.4　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。

4.2 ソースコード

```
// "ddsbcn.c" ver2.3.5  (c) 2019.05.07 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.
// Target hardware is PIC16F1823 and DDS unit GY9833(AD9833) replaced XTAL 12.288 MHz.
// 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 accurate GPS time sequence.
//      Transmitt rate of once per 30 or 4 minute.
//   (3)WSPR15:Begin transmitt at 00 and 30 minute.
//      Same massege as WSPR2.

#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=OFF, WDTE=OFF, FOSC=ECH, LVP=OFF, BORV=LO
#pragma config STVREN=OFF, PLLEN=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,0x4a,0x74,0x2e,0xbb,0xde,0x22);   // "JG1JZL QM05 50"
__EEPROM_DATA(0xae,0xb1,0xaa,0x06,0x78,0x79,0x29,0x6c);
__EEPROM_DATA(0x29,0xc6,0x68,0x6b,0x84,0xd0,0x1c,0xe6);
__EEPROM_DATA(0x8e,0x2a,0x4b,0x87,0x4d,0x8d,0x1a,0x15);
__EEPROM_DATA(0x80,0xb1,0x25,0x02,0x21,0x66,0x72,0x3e);
__EEPROM_DATA(0x0f,0xff,0xff,0xff,0xff,0xff,0xff,0xff);

unsigned char freq03, freq02, freq01, freq00;             // f0
unsigned char freq13, freq12, freq11, freq10;             // f1
unsigned char freq23, freq22, freq21, freq20;             // f2
unsigned char freq33, freq32, freq31, freq30;             // f3

void pic1823(void){                                       // device setting
    APFCON=0b10000100;                                    // RX=RA1
    ANSELA=0b00000000;                                    // digital portA
    ANSELC=0b00000000;                                    // digital portC
    TRISA=0b00101111;                                     // RA5=CLK,RA3=D1,
                                                          // RA2=1PPS,RA1=NMEA,RA0=D0
    TRISC=0b00110010;                                     // RC5=D3,RC4=D2,RC3=FSYNC,
                                                          // RC2=SDATA,RC1=DDI,RC0=SCLK
    OPTION_REG=0b00000000;                                // WPU enable
    WPUA=0b00001001;                                      // WPU portA
    WPUC=0b00110010;                                      // WPU portC
    SSP1CON1=0b00110000;                                  // SPI master mode,
    SSP1CON3=0b00000000;
    SSP1STAT=0b01000000;                                  // falling edge
    T1CON=0b00110000;                                     // TMR1 prescaler 1:8
    T1GCON=0b00000000;
    IOCAP=0b00000001;                                     // positive change
    IOCAN=0b00000001;                                     // negative change
```

```
    INTCON=0b10001000;                                  // RA0 change interrupt
    PIE1=0b00000000;
    PIE2=0b00000000;
    TXSTA=0b00000000;                                   // 8N
    RCSTA=0b10010000;                                   // asynchronous
    SPBRGH=0x00;                                        // Fin/16/9600-1
    SPBRGL=0x4f;                                        // 80-1
    BAUDCON=0b00001000;                                 // BRG16
    EECON1=0b00000000;
    EECON2=0b00000000;
    CCP1CON=0b00000000;
    }// pic1823()

void timer(unsigned int term){                          // hold up
    unsigned int multi;                                 // local counter
    TMR1H=0; TMR1L=0;                                   // initial TMR1
    TMR1IF=0; TMR1ON=1;                                 // start TMR1
    for(multi=0; multi<term; multi++){                  // 0.17s*term
        while(!TMR1IF); TMR1IF=0;
        }// for(multi)
    TMR1ON=0;                                           // stop TMR1
    }// timer(term)

void spi(unsigned char value){                          // set register
    SSP1BUF=value; while(!BF);                          // send a byte (8bits)
    }// spi(value)

void cw_space(unsigned int unit){                       // sleep AD9833
    RC3=0; spi(0x20); spi(0x40); RC3=1;                 // clock off
    timer(unit);                                        // gap of character
    }// cw_space(unit)

void cw_mark(unsigned char unit){                       // awake AD9833
    RC3=0; spi(0x20); spi(0x00); RC3=1;                 // clock on
    if(unit=='f') return;
    timer(unit);                                        // length of mark
    cw_space(1);                                        // gap of code
    }// cw_mark(unit)

void cw(unsigned char wpm){
    unsigned long freq;                                 // 0<0x10000000
    unsigned char key='p';                              // paddle
    unsigned char dots;                                 // counter
    T1CKPS0=0;                                          // prescaler (1:4)
    if(wpm=='s') T1CKPS0=1;                             // prescaler (1:8)
    freq=0x2d0000;                                      // preset 135.0kHz
    freq+=0x8000;                                       // 1500*8192/375
    freq00=freq&0x00000ff;                              // bit[7:1]
    freq01=freq>>8;  freq01&=0x0003f; freq01+=0x40;     // bit[13:8] with address
    freq02=freq>>14; freq02&=0x00ff;                    // bit[21:14]
    freq03=freq>>22; freq03+=0x40;                      // bit[27:22] with address
    RC3=0; spi(freq01); spi(freq00);                    // write f0
    spi(freq03); spi(freq02); RC3=1;                    // LSB first
    while(1){
        switch(key){
            case 'p': if(RC4) dots=0;
                      if(!RC4){
                          cw_mark(1);                   // "dot"
```

```
                        if(dots++>11) key='n';          // straight-key
                        }// if(!RC4)
                    if(!RC5){
                        cw_mark(3);                     // "dash"
                        }// if(RC5)
                    break;
            default:  if(!RC5) key='p';                 // paddle-key
                    if(!RC4){
                        cw_mark('f');                   // long "mark"
                        while(!RC4);                    // long "space"
                        }// if(!RC4)
             }// switch(key)
        }// while(1)                                    // infinitive
    }// cw(wpm)

void cwbcn(void){
    unsigned long freq;                                 // 0<0x10000000
    T1CKPS0=1;                                          // 1:8
    freq=0x2d0000;                                      // preset 135.0kHz
    freq+=0x9999;                                       // 1800*8192/375
    freq00=freq&0x00000ff;                              // bit[7:1]
    freq01=freq>>8;  freq01&=0x0003f; freq01+=0x40;     // bit[13:8] with address
    freq02=freq>>14; freq02&=0x00ff;                    // bit[21:14]
    freq03=freq>>22; freq03+=0x40;                      // bit[27:22] with address
    RC3=0; spi(freq01); spi(freq00);                    // write f0
    spi(freq03); spi(freq02); RC3=1;                    // LSB first
    cw_mark(1); cw_mark(3); cw_mark(3); cw_mark(3);     // 'J'
    timer(2);
    cw_mark(3); cw_mark(3); cw_mark(1);                 // 'G'
    timer(2);
    cw_mark(1); cw_mark(3); cw_mark(3); cw_mark(3);     // '1'
    cw_mark(3); timer(2);
    cw_mark(1); cw_mark(3); cw_mark(3); cw_mark(3);     // 'J'
    timer(2);
    cw_mark(3); cw_mark(3); cw_mark(1); cw_mark(1);     // 'Z'
    timer(2);
    cw_mark(1); cw_mark(3); cw_mark(1); cw_mark(1);     // 'L'
    timer(30);                                          // interval
    }// cwbcn()

void dfcw_tx(unsigned char fsk){
    RC3=0;
    spi(freq11); spi(freq10);                           // RZ
    spi(freq13); spi(freq12);                           // MSB last
    spi(0x20); spi(0x00); RC3=1;                        // clock on
    timer(30);                                          // length of f0
    RC3=0; spi(0x20); spi(0x40);                        // clock off
    switch(fsk){
        case 'l':                                       // write -df
            spi(freq01); spi(freq00);                   // LSB first
            spi(freq03); spi(freq02);
            break;
        case 'h':                                       // write +df
            spi(freq21); spi(freq20);                   // LSB first
            spi(freq23); spi(freq22);                   //
            break;
        case 'z':
            spi(freq11); spi(freq10);                   // RZ
```

```
            spi(freq13); spi(freq12);                       // MSB last
            break;
        }// switch(fsk)
    spi(0x20); spi(0x00); RC3=1;                            // clock on
    if(fsk=='l' || fsk=='h') timer(300);                    // length of fsk
    else timer(10);
    RC3=0; spi(0x20); spi(0x40); RC3=1;                     // clock off
    }// dfcw_tx(fsk)

void dfcw(void){
    unsigned long freq;                                     // 0<0x10000000
    T1CKPS0=1;                                              // 1:8
    freq=0x2d0000;                                          // preset 135.0kHz
    freq+=0xece2;                                           // 1776*8192/375
    freq10=freq&0x00000ff;                                  // bit[7:1]
    freq11=freq>>8;  freq11&=0x0003f; freq11+=0x40;         // bit[13:8] with address
    freq12=freq>>14; freq12&=0x00ff;                        // bit[21:14]
    freq13=freq>>22; freq13+=0x40;                          // bit[27:22] with address
    freq-=0x05;                                             // -df
    freq00=freq&0x00000ff;                                  // bit[7:1]
    freq01=freq>>8;  freq01&=0x0003f; freq01+=0x40;         // bit[13:8] with address
    freq02=freq>>14; freq02&=0x00ff;                        // bit[21:14]
    freq03=freq>>22; freq03+=0x40;                          // bit[27:22] with address
    freq+=0x0a;                                             // +df
    freq20=freq&0x00000ff;                                  // bit[7:1]
    freq21=freq>>8;  freq21&=0x0003f; freq21+=0x40;         // bit[13:8] with address
    freq22=freq>>14; freq22&=0x00ff;                        // bit[21:14]
    freq23=freq>>22; freq23+=0x40;                          // bit[27:22] with address
    dfcw_tx('l'); dfcw_tx('h'); dfcw_tx('h');               // 'J'
    dfcw_tx('h');
    dfcw_tx('z'); cw_space(330);
    dfcw_tx('h'); dfcw_tx('h'); dfcw_tx('l');               // 'Z'
    dfcw_tx('l');
    dfcw_tx('z'); cw_space(330);
    dfcw_tx('l'); dfcw_tx('h'); dfcw_tx('l');               // 'L'
    dfcw_tx('l');
    dfcw_tx('z'); cw_space(1500);
    }// dfcw()

void wspr_fsk(unsigned char fsk4){
    RC3=0; spi(0x20); spi(0x40);                            // clock off
    switch(fsk4){
        case 0x00:                                          // write f0
            spi(freq01); spi(freq00);                       // LSB first
            spi(freq03); spi(freq02);                       //
            break;
        case 0x40:                                          // write f1
            spi(freq11); spi(freq10);                       // LSB first
            spi(freq13); spi(freq12);                       //
            break;
        case 0x80:                                          // write f2
            spi(freq21); spi(freq20);                       // LSB first
            spi(freq23); spi(freq22);                       //
            break;
        case 0xc0:                                          // write f3
            spi(freq31); spi(freq30);                       // LSB first
            spi(freq33); spi(freq32);                       //
            break;
```

```
        }// switch(fsk4)
    spi(0x20); spi(0x00); RC3=1;                            // clock on
    }// wspr_fsk(fsk4)

void wspr_code(unsigned char wsprx, unsigned char msg){
    unsigned char data;                                     // data
    unsigned char quart, addrROM;                           // address
    asm("      BANKSEL(_PORTA)");                           // 1PPS
    asm("fall0 BTFSC _PORTA,2");                            // TTL low
    asm("      goto fall0");
    asm("rise0 BTFSS _PORTA,2");                            // TTL high
    asm("      goto rise0");
    for(addrROM=msg; addrROM<(msg+40); addrROM++){
        EEADRL=addrROM; RD=1; data=EEDATL;
        for(quart=0; quart<4; quart++){
            wspr_fsk(0xc0&(data<<(2*quart)));
            timer(wsprx*4);
            }// for(quart)
        }// for(addrROM)
    EEADRL=addrROM++; RD=1; data=EEDATL;
    for(quart=0; quart<2; quart++){
        wspr_fsk(0xc0&(data<<(2*quart)));
        timer(wsprx*4);
        }// for(quart)
    RC3=0; spi(0x20); spi(0x40); RC3=1;                     // clock off
    }// wspr_code(wsprx, msg)

void wspr(unsigned char wsprx, unsigned char rate,
          unsigned char msg){
    unsigned char nmea[10], col;                            // NMEA
    unsigned long freq;                                     // <0x10000000
    T1CKPS0=1;                                              // 1:8
    freq=0x2d0000;                                          // offset 135.0kHz
    if(wsprx==0x01) freq+=0xd47a;                           // 2490*8192/375
    else freq+=0xdee4;                                      // 2612*8192/375
    freq10=freq&0x00000ff;                                  // bit[7:1]
    freq11=freq>>8;  freq11&=0x0003f; freq11+=0x40;         // bit[13:8] with address
    freq12=freq>>14; freq12&=0x00ff;                        // bit[21:14]
    freq13=freq>>22; freq13+=0x40;                          // bit[27:22] with address
    if(wsprx==0x01) freq-=0x20; else freq-=0x04;            // -df
    freq00=freq&0x00000ff;                                  // bit[7:1]
    freq01=freq>>8;  freq01&=0x0003f; freq01+=0x40;         // bit[13:8] with address
    freq02=freq>>14; freq02&=0x00ff;                        // bit[21:14]
    freq03=freq>>22; freq03+=0x40;                          // bit[27:22] with address
    if(wsprx==0x01) freq+=0x40; else freq+=0x08;            // +df
    freq20=freq&0x00000ff;                                  // bit[7:1]
    freq21=freq>>8;  freq21&=0x0003f; freq21+=0x40;         // bit[13:8] with address
    freq22=freq>>14; freq22&=0x00ff;                        // bit[21:14]
    freq23=freq>>22; freq23+=0x40;                          // bit[27:22] with address
    if(wsprx==0x01) freq+=0x20; else freq+=0x04;            // +2df
    freq30=freq&0x00000ff;                                  // bit[7:1]
    freq31=freq>>8;  freq31&=0x0003f; freq31+=0x40;         // bit[13:8] with address
    freq32=freq>>14; freq32&=0x00ff;                        // bit[21:14]
    freq33=freq>>22; freq33+=0x40;                          // bit[27:22] with address
    do{
        CREN=0;                                             // clear OERR
        nmea[0]=RCREG; nmea[0]=RCREG;                       // clear buffers
        CREN=1;                                             // start EUSART
```

```
        do{
            while(!RCIF);                                   // wait for fill
            }while('$'!=RCREG);                             // find '$'
        for(col=0; col<10; col++){                          // store 10 chars
            while(!RCIF);
            nmea[col]=RCREG;
            }
        }while(nmea[3]!='M');                               // get gprMc
    switch(rate){
        case 'r':                                           // MODE4,5,8,9
            if(nmea[8]%3==0){                               // seek m*=0 or 3
                wspr_code(wsprx, msg);                      // WSPR15
                }// if()
            break;
        case 'm':                                           // MODE6,7
            if(nmea[8]%2==1){                               // seek m*=1*
                if(nmea[9]=='2' || nmea[9]=='6')            // 12,16,32,36,52,56
                    wspr_code(wsprx, msg);                  // WSPR2
                break;
                }// if(odd)
            else{                                           // seek m*=0*
                if(nmea[9]%4==0)                            // 00,04,08,20,24,28,40,44,48
                    wspr_code(wsprx, msg);                  // WSPR2
                break;
                }// if(even)
        }// switch(rate)
    }// wspr(wsprx,rate,msg)

void main(void){
    unsigned char mode;                                     // mode selector
    pic1823();                                              // device setting
    RC3=0; spi(0x21); spi(0x40); RC3=1;                     // reset AD9833
    do{                                                     // infinitive
        mode=(PORTA&0x09); mode+=(PORTC&0x30);              // hex-dial encoder
        switch(mode){                                       // select mode
            case 0x39: cw('s'); break;                      // CW 136.5kHz QRS
            case 0x38: cw('q'); break;                      // CW 136.5kHz QRQ
            case 0x31: cwbcn(); break;                      // CWBCN 136.8kHz
            case 0x30: dfcw();  break;                      // DFCW 137.776kHz
            case 0x29: wspr(0x01,'r',0x00); break;          // WSPR2 137.49kHz 1/30 10W
            case 0x28: wspr(0x01,'r',0x30); break;          // WSPR2 137.49kHz 1/30 50W
            case 0x21: wspr(0x01,'m',0x00); break;          // WSPR2 137.49kHz 1/4 10W
            case 0x20: wspr(0x01,'m',0x30); break;          // WSPR2 137.49kHz 1/4 50W
            case 0x19: wspr(0x08,'r',0x00); break;          // WSPR15 137.612kHz 10W
            case 0x18: wspr(0x08,'r',0x30); break;          // WSPR15 137.612kHz 50W
            }// switch(mode)
        }while(1);
    }// main()

void __interrupt() chng(void){                              // interrupt routein
    if(IOCIF==1){ IOCAF0=0; PCL=0x00; }
    }// chng()
```

4.3 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.

5 電力増幅部

本機の電力増幅部を含む全回路図は図 7 です。

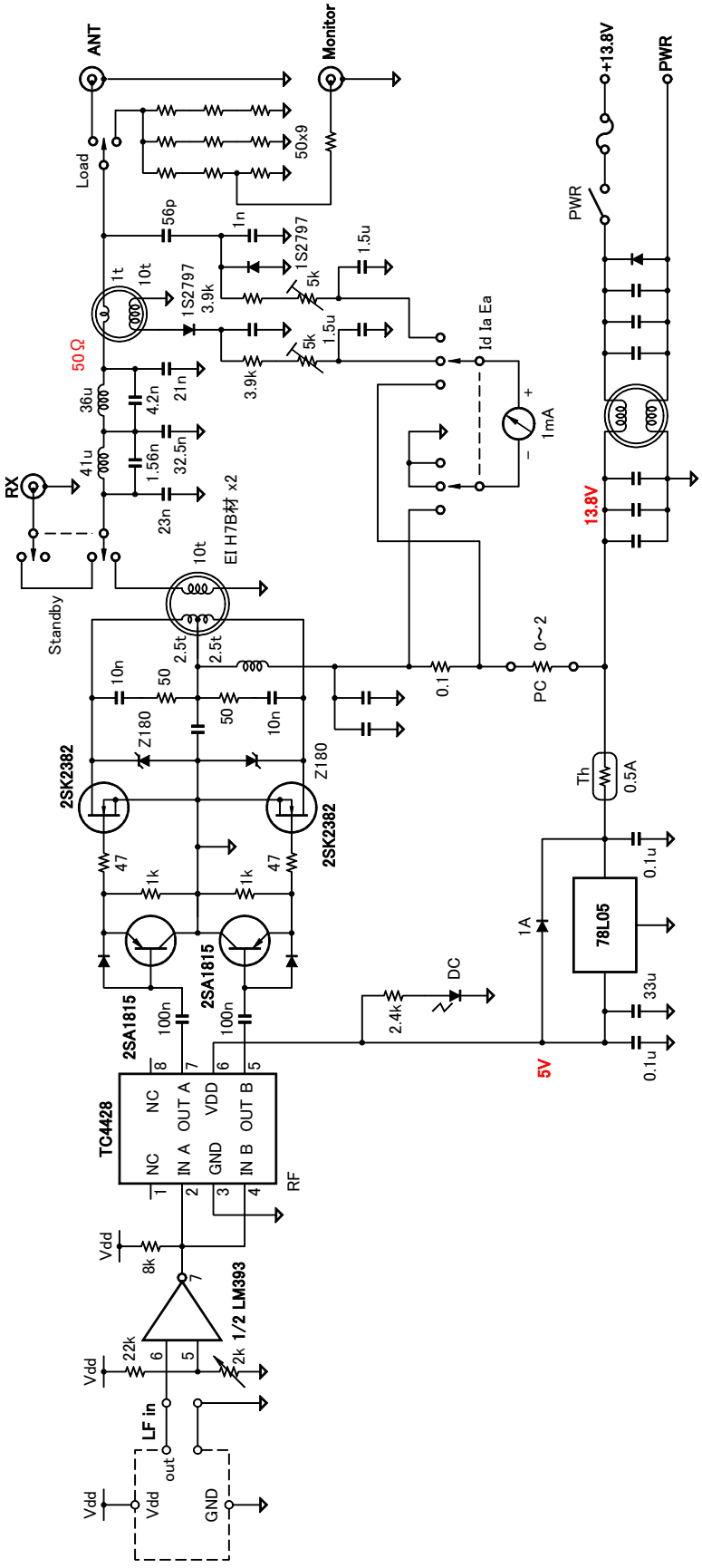

図 7: LF122 電力増幅部の回路