STD-302 Interface Method

By Tomihiko Uchikawa

◆ Introduction

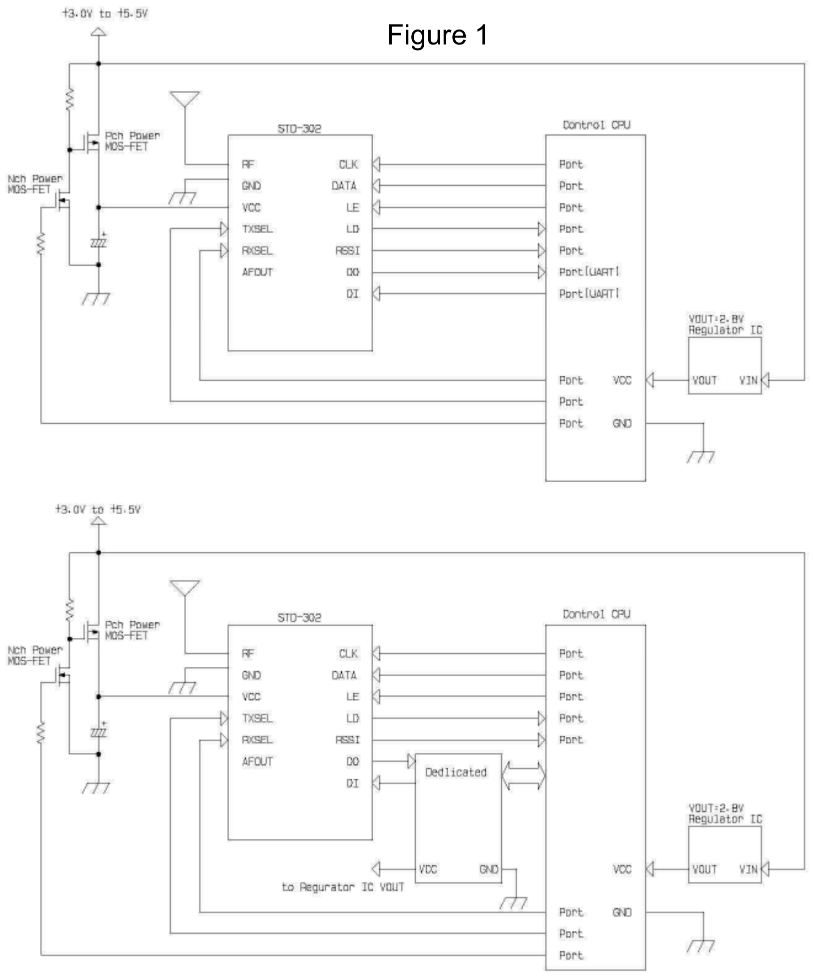
STD-302 is a module that provides optimized radio function offering flexible control for various industrial applications. It is especially suitable for feedback systems using half duplex communication.
This reference guide explains the interface method for engineers developing applications using STD-302.

◆ Typical interface

A typical interface of STD-302 is shown in Figure 1. The upper example controls the STD-302 using CPU ports and UART ports, while the lower example uses a dedicated IC with data format or error detection or correction functions. Both systems perform the following processes.

1. The transmission data (digital signal) is input to the DI terminal.
2. The received data (digital signal) is output from the DO terminal.
3. PLL IC control when the RF channel is changed.
4. The lock detect (LD) signal is processed when the RF channel is set.
5. Uses RSSI output.
6. Transmission mode and reception mode control.

Figure 1



◆ Power supply interface

The VCC voltage of the STD-302 is +3.0 to 5.5 V, however the internal circuit operates at regulated 2.8 V.
Power MOS FET with low ON resistance is used for the power supply interface as shown in Figure 1 to enable ON/OFF control (high active) from the control CPU.
STD-302 is equipped with an internal PLL frequency synthesizer (Fujitsu MB15E03SL). The absolute maximum rating for the input terminals of the PLL IC is VCC + 0.5 V, meaning that the absolute maximum rating for the input terminals of the STD-302 is +3.3 V.
It is recommended that a voltage regulator with 2.8V output is used for power supply of the control CPU. If such a regulator is not used, use STD-302 within the VCC voltage range of +3.0 to +3.3V.
**If a regulator is not used for power supply of the control CPU, it is recommended that a level conversion circuit is used for the interface to avoid overshoot voltage.*

◆ Input/Output data

Transmission data (digital signal) is input to the DI terminal, which accepts signals up to 4.8 kHz (9800 bps). Received data (digital signal) is output from the DO terminal. However, the internal digitization circuit (slicer circuit) of STD-302 cannot handle consecutive high or low levels of 10 ms or longer due to its time constant. Therefore it is necessary to devise a method such as processing the data input to the DI transmission terminal so that it does not contain consecutive highs or lows of 10 ms. Data encoding (Manchester coding, CMI coding etc.) is a suitable method, however care should be taken that the frequency bandwidth is not wider than that specified for the module.
If a signal in UART format from the CPU is used, you can transfer data at a maximum of 9600 bps without concern for this issue. For more information about using a UART interface with radio modules, please contact Circuit Design, Inc.

◆ RSSI signal

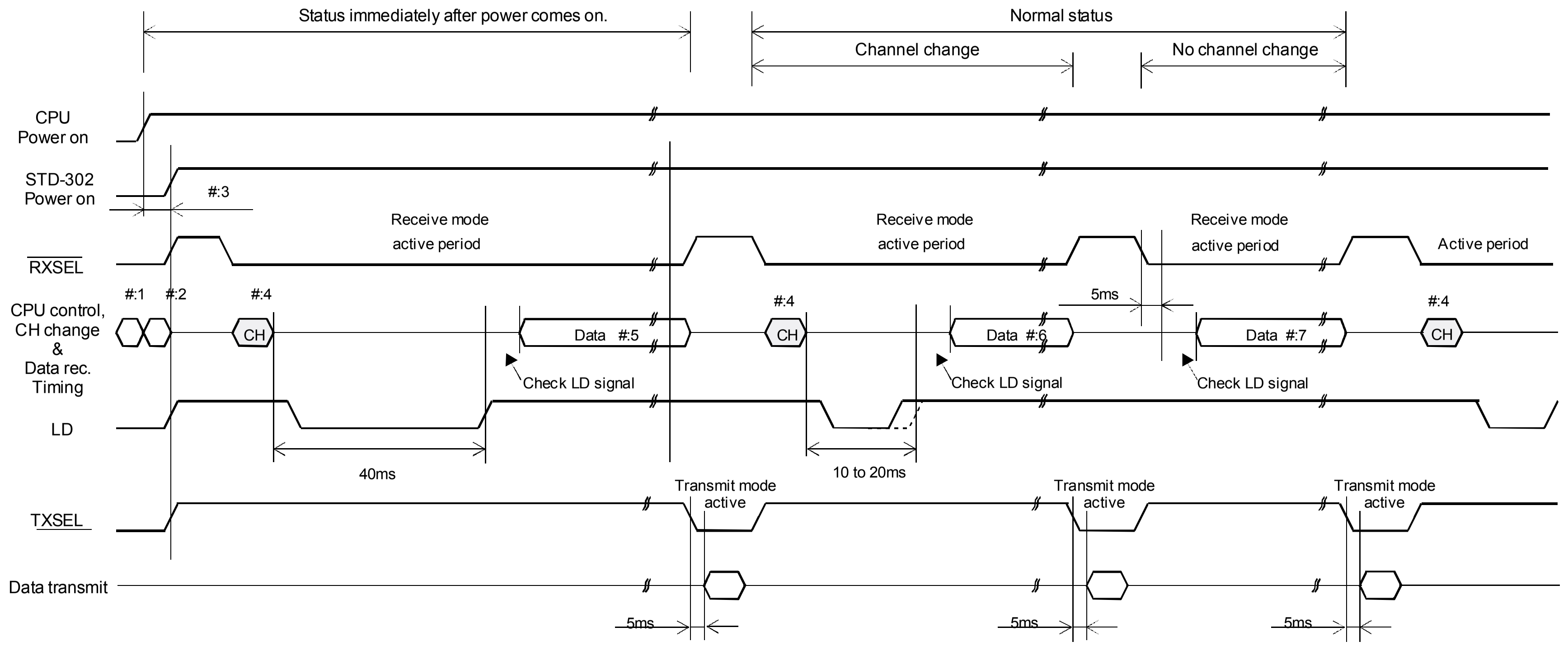
The RSSI terminal outputs the voltage corresponding to the received signal level. This voltage level can be used for functions such as carrier sense[*1], channel auto select[*2], and electric field strength monitoring[*3].
Normally, the CPU processes the RSSI voltage using an A/D converter.
[*1] Carrier sense: A function that judges whether a RF channel is currently open for transmission or not.
[*2] Channel auto select: A function that automatically selects a frequency (RF channel) that is not currently used.
[*3] Electric field strength monitoring: A function that measures the strength of electric field.

◆ Control timing

Control timing in a typical application is shown in Figure 2.
Initial setting of the CPU port to be used is performed when power is supplied by the control CPU and reset is completed. MOS-FET for supply voltage control of the STD-302, RXSEL and TXSEL are set to inactive to avoid unwanted emissions. The power supply of the STD-302 is then turned on. When the STD-302 is turned on, the PLL internal resistor is unstable. Therefore data transmission and reception is possible 40 ms after the frequency setting data is sent to the PLL. For channel setting except when the power is turned on, STD-302 can handle the data 20 ms after.
**Setting channels must be carried out in the receive mode. If setting is performed in transmission mode, unwanted emission occurs.*
**Sending PLL setting data while the STD-302 is turned off may cause latch-up. Ensure that interface communication is performed with the power of the STD-302 turned on.*

If STD-302 is switched to the receive mode when operating in the same channel (a new PLL setting is not necessary), it can receive data after 5 ms elapses. For data transmission, if the RF channel to be used for transmission is set while still in receiving mode, data can be sent at 5 ms after the STD-302 is switched from reception to transmission.
Check that the LD signal is “high” 20 ms after the channel is changed. In some cases the LD signal becomes unstable before the lock is correctly detected, so it is necessary to note if processing of the signal is interrupted. It is recommended to observe the actual waveform before writing the process program.

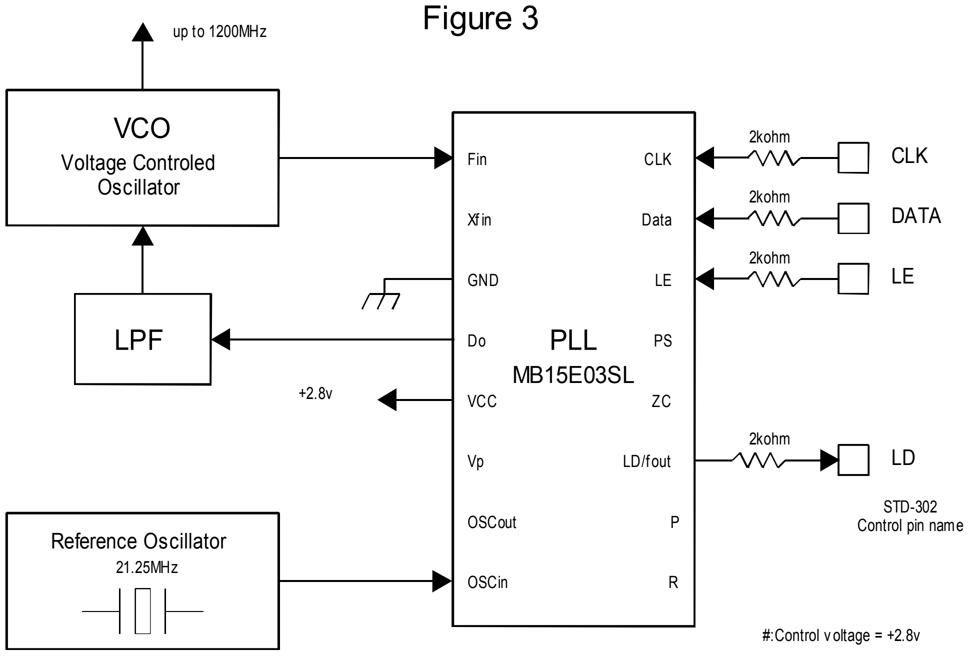
Figure 2: Timing Diagram for STD-302



#:1 Reset control CPU
#:2 Initialize the port connected to the module.
#:3 Supply power to the module after initializing CPU.
#:4 RFchannel change must be done in receiving.
#:5 40ms later, the receiver can receive the data after changing the channel..
#:6 10 to 20ms later, the receiver can receive the data after changing the channel.
#:7 5ms later, the data can be received if the RF channel is not changed.

◆ PLL IC control

Figure 3



STD-302 is equipped with an internal PLL frequency synthesizer as shown in Figure 3. The operation of the PLL circuit enables the VCO to oscillate at a stable frequency. Transmission and reception frequency is set externally by the controlling IC. STD-302 has control terminals (CLK, LE, DATA) for the PLL IC and the setting data is sent to the internal register serially via the data line. Also STD-302 has a Lock Detect (LD) terminal that shows the lock status of the frequency. These signal lines are connected directly to the PLL IC through a 2 kΩ resistor.

STD-302 comes equipped with a Fujitsu MB15E03SL PLL IC. Please refer to the manual of the PLL IC. The following is a supplementary description related to operation with STD-302. In this description, the same names and terminology as in the PLL IC manual are used, so please read the manual beforehand.

◆ How to calculate the setting value to PLL register

The PLL IC manual shows that the PLL frequency setting value is obtained with the following equation.

$f_{vco} = [(M \times N)+A] \times f_{osc} / R$ -- Equation 1

f_{vco} : Output frequency of external VCO
M: Preset divide ratio of the prescaler (64 or 128)
N: Preset divide ratio of binary 11-bit programmable counter (3 to 2,047)
A: Preset divide ratio of binary 7-bit swallow counter ($0 \leq A \leq 127$ $A<N$))
f_{osc}: Output frequency of the reference frequency oscillator
R: Preset divide ratio of binary 14-bit programmable reference counter (3 to 16,383)

With STD-302, there is an offset frequency (f_{offset}) 21.7MHz for the transmission RF channel frequency f_{ch}. Therefore the expected value of the frequency generated at VCO (f_{expect}) is as below.

$f_{vco} = f_{expect} = f_{ch} - f_{offset}$ ---- Equation 2

The PLL internal circuit compares the phase to the oscillation frequency $f_{vco.}$ This phase comparison frequency (f_{comp}) must be decided. f_{comp} is made by dividing the frequency input to the PLL from the reference frequency oscillator by reference counter R. STD-302 uses 21.25 MHz for the reference clock $f_{osc.}$ f_{comp} is one of 6.25 kHz, 12.5 kHz or 25 kHz.

The above equation 1 results in the following with n = M x N + A, where "n" is the number for division.

$f_{vco}=n*f_{comp}$ ---- Equation 3 $n = f_{vco}/f_{comp}$ ---- Equation 4 note: $f_{comp} = f_{osc}/R$

Also, this PLL IC operates with the following R, N, A and M relational expressions.

$R=f_{osc}/f_{comp}$ ---- Equation 5 $N = \mathrm{INT}(n / M)$ ---- Equation 6 $A = n - (M \times N)$ ---- Equation 7
INT: integer portion of a division.

As an example, the setting value of RF channel frequency f_{ch} 869.725 MHz can be calculated as below.
The constant values depend on the electronic circuits of STD-302.

Conditions:	Channel center frequency:	f_{ch} = 869.725 MHz
	Constant: Offset frequency:	f_{offset}=21.7 MHz
	Constant: Reference frequency:	f_{osc}=21.25 MHz
	Set 25 kHz for Phase comparison frequency and 64 for Prescaler value M	

The frequency of VCO will be
$f_{vco} = f_{expect} = f_{ch} - f_{offset} = 869.725 - 21.7 = 848.025\text{MHz}$
Dividing value "n" is derived from Equation 4
$n = f_{vco} / f_{comp} = 848.025\text{MHz}/25\text{kHz} = 33921$
Value "R" of the reference counter is derived from Equation 5.
$R = f_{osc}/f_{comp} = 21.25\text{MHz}/25\text{kHz} = 850$
Value "N" of the programmable counter is derived from Equation 6.
$N = \mathrm{INT}(n/M) = \mathrm{INT}(33921/64) = 530$
Value "A" of the swallow counter is derived from Equation 7.
$A = n - (M \times N) = 33921 - 64 \times 530 = 1$

The frequency of STD-302 is locked at a center frequency f_{ch} by inputting the PLL setting values N, A and R obtained with the above equations as serial data. The above calculations are the same for the other frequencies. Excel sheets that contain automatic calculations for the above equations can be found on our web site (www.circuitdesign.jp/eng/).

The result of the calculations is arranged as a table in the CPU ROM. The table is read by the channel change routine each time the channel is changed, and the data is sent to the PLL.

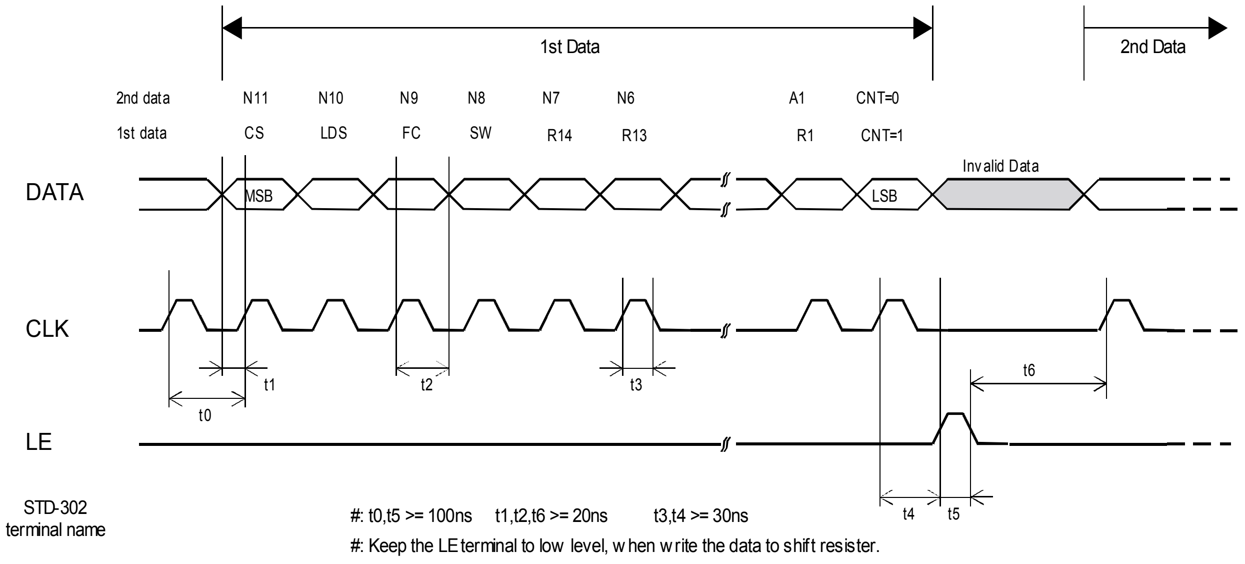
**Hint: You will notice when using the automatic calculation in the Excel sheet, that actually the values R and M can be fixed and only the values A and B are variable.*

◆ Method of serial data input to the PLL

After the RF channel table plan is decided, the data needs to be allocated to the ROM table and read from there or calculated with the software.
Together with this setting data, operation bits that decide operation of the PLL must be sent to the PLL.
The operation bits for setting the PLL are as follows. These values are placed at the head of the reference counter value and are sent to the PLL.

1. CS: Charge pump current select bit
   CS = 0 +/-1.5 mA select VCO is optimized to +/-1.5 mA
2. LDS: LD/fout output setting bit
   LDS = 0 LD select Hardware is set to LD output
3. FC: Phase control bit for the phase comparator
   FC = 1 Hardware operates at this phase

Figure 4



The PLL IC, which operates as shown in the block diagram in the manual, shifts the data to the 19-bit shift register and then transfers it to the respective latch (counter, register) by judging the CNT control bit value input at the end.

1. CLK [Clock]: Data is shifted into the shift register on the rising edge of this clock.
2. LE [Load Enable]: Data in the 19-bit shift register is transferred to respective latches on the rising edge of the clock. The data is transferred to a latch according to the control bit CNT value.
3. Data [Serial Data]: You can perform either reference counter setup or programmable counter setup first.

◆ Reference: PLL set up program

Please refer to the following PLL setting program written in C language. It is also useful when using an assembler. In the program, the channel change is performed with only swallow counter change.
This program is for a CPU operating with a 10 MHz clock. For actual use, it is necessary to observe the waveform with an oscilloscope and adjust the program to meet the conditions of the serial communication timing.

****

```
/***********************************************************************
            PLL Setting Program for MB15E03SL(Selective Part)
            Tested on M16C/30620FCMGP at 9.8304MHz
            Written by T.Uchikawa           Date: 2002.07.05
            Copyright Circuit Design, INC. in Japan
***********************************************************************/
typedef unsigned char byte;
typedef unsigned int word;
typedef unsigned char pragmabit;

#define bithigh 1
#define bitlow 0

//**********************************************************************
//                      Structure and Union Declare
//**********************************************************************
#pragma BIT sPLL_Ref                              // for Reference Counter
struct sPLL_Ref{
            pragmabit R01:      1;
            pragmabit R02:      1;
            pragmabit R03:      1;
            pragmabit R04:      1;
            pragmabit R05:      1;
            pragmabit R06:      1;
            pragmabit R07:      1;
            pragmabit R08:      1;
            pragmabit R09:      1;
            pragmabit R10:      1;
            pragmabit R11:      1;
            pragmabit R12:      1;
            pragmabit R13:      1;
            pragmabit R14:      1;
};

union uPLL_Ref{
            struct sPLL_Ref     PLLRef;
            word                wPLLRef;
};

#pragma BIT sPLL_Any                              // for Any Data
struct sPLL_Any{
            pragmabit SW:       1;
            pragmabit FC:       1;
            pragmabit LDS:      1;
            pragmabit CS:       1;
};
union uPLL_Any{
        struct sPLL_Any     PLLAny;
        byte                bPLLAny;
};

#pragma BIT sPLL_Swa                    // for Swallow Counter
struct sPLL_Swa{
        pragmabit A01:      1;
        pragmabit A02:      1;
        pragmabit A03:      1;
        pragmabit A04:      1;
        pragmabit A05:      1;
        pragmabit A06:      1;
        pragmabit A07:      1;
};
union uPLL_Swa{
        struct sPLL_Swa     PLLSwa;
        byte                bPLLSwa;
};

#pragma BIT sPLL_Pro                    // for Programmable Counter
struct sPLL_Pro{
        pragmabit N01:      1;
        pragmabit N02:      1;
        pragmabit N03:      1;
        pragmabit N04:      1;
        pragmabit N05:      1;
        pragmabit N06:      1;
        pragmabit N07:      1;
        pragmabit N08:      1;
        pragmabit N09:      1;
        pragmabit N10:      1;
        pragmabit N11:      1;
};
union uPLL_Pro{
        struct sPLL_Pro     PLLPro;
        word                wPLLPro;
};

//**********************************************************************
//                  Function Declare
//**********************************************************************
void far chSelect(void);                // Channel Select Routine
void far setPLL(word, byte, word);      // PLL Set Routine


//**********************************************************************
//                  Main Function
//**********************************************************************

void far main()
{
        while(1)
        {
                chSelect();                     // call channel select routine
        }
}
```

```
//--------------------------------------------------------------------
// Send the setting value to PLL,
// only when the channel was changed by switch1.
void far chSelect(void)
{
        byte valSwa;
        word valRef;
        word valPro;
        static byte valSW1;                     // Channel Switch 1 on Port2


        if(valSW1 != ~p2)                       // p2 = port2 data
        {
                valSW1 = ~p2;
                valRef = 850;                   // Reference Counter Value
                valSwa = valSW1 & 0x0f;         // Swallow Counter Value
                valPro = 530;                   // Programmable Counter Value
                setPLL(valRef, valSwa, valPro);
        }
}


//--------------------------------------------------------------------
void far setPLL(word valRef, byte valSwa, word valPro)
{
        int i, j;
        union uPLL_Ref Ref;
        union uPLL_Any Any;
        union uPLL_Swa Swa;
        union uPLL_Pro Pro;

// Basic setting
        Any.PLLAny.CS = 0;                              // ±1.5mA
        Any.PLLAny.LDS = 0;                                     // Lock Detect Signal
        Any.PLLAny.FC = 1;                              // carve (1) select
        Any.PLLAny.SW = 1;                              // Prescaler M=64
        Ref.wPLLRef = valRef;

        Swa.bPLLSwa = valSwa;
        Pro.wPLLPro = valPro;


// Actual Hardwere Port Control
        Module_CLK = bitlow;
        Module_LE = bitlow;
        for(j=1; j < 5; j++)                            // Send Any Data, CS,LDS,FC,SW
        {
                Module_CLK = bitlow;
                if(Any.PLLAny.CS == 1)
                        Module_DATA = bithigh;
                else
                        Module_DATA = bitlow;
                Any.bPLLAny = Any.bPLLAny << 1;
                Module_CLK = bithigh;
        }
        for(j=1; j < 15; j++)                           // Send Reference Counter Data
        {
                Module_CLK = bitlow;
                if(Ref.PLLRef.R14 == 1)
                        Module_DATA = bithigh;
                else
                        Module_DATA = bitlow;
                Ref.wPLLRef = Ref.wPLLRef << 1;
                Module_CLK = bithigh;
        }
        Module_CLK = bitlow;
        Module_DATA = bithigh;                          // set CNT bit = 1
        Module_CLK = bithigh;
        Module_CLK = bitlow;

        Module_LE = bithigh;                                    // Data Latch
        Module_LE = bitlow;

        for(j=1; j < 12; j++)                           // Send Programmable Counter Data
        {
                Module_CLK = bitlow;
                if(Pro.PLLPro.N11 == 1)
                        Module_DATA = bithigh;
                else
                        Module_DATA = bitlow;
                Pro.wPLLPro = Pro.wPLLPro << 1;
                Module_CLK = bithigh;
        }
        for(j=1; j < 8; j++)                            // Send Swallow Counter Data
        {
                Module_CLK = bitlow;
                if(Swa.PLLSwa.A07 == 1)
                        Module_DATA = bithigh;
                else
                        Module_DATA = bitlow;
                Swa.bPLLSwa = Swa.bPLLSwa << 1;
                Module_CLK = bithigh;
        }
        Module_CLK = bitlow;
        Module_DATA = bitlow;                           // set CNT bit = 0
        Module_CLK = bithigh;
        Module_CLK = bitlow;

        Module_LE = bithigh;                                    // Data Latch
        Module_LE = bitlow;
}

*****************************************************************************************
```