



Bob Nelson

5/8/2003

```
//*****
// RF Receive Uart Code Example for MicroChip 16Fxxx micros      *
// This is example code only and the user will have to modify to *
// comply with his/her design tools and design environment.     *
// There are no guarantees or warranties implied with this example.*
// Please use at your own risk.                                  *
//*****
//
// Here are two links that may be use full while designing code:
//
// http://www.rfm.com/products/apnotes/simpleclock.pdf
// http://www.rfm.com/products/tr\_des24.pdf
//
// This ISR Detects the RFM Start Symbol (000111111110)
// Resyns the RTC(Timer)to one bit time at the falling edge of the
// last zero of the start symbol.
// Then shifts in symbols keeping track of lsb & mbs symbols
// When a complete byte is received (two symbols) a flag is set so
// "main" can decode the byte.
// This receiver is designed to use the RFM 4 bit to 6 bit symbols,
// 8 bits(byte) = 12 bits
//
//
//          Hex          Nibble          Bit
//          0x15         NIBBLE = 0         010101
//          0x31         NIBBLE = 1         110001
//          0x32         NIBBLE = 2         110010
//          0x23         NIBBLE = 3         100011
//          0x34         NIBBLE = 4         110100
//          0x25         NIBBLE = 5         100101
//          0x26         NIBBLE = 6         100110
//          0x16         NIBBLE = 7         010110
//          0x1A         NIBBLE = 8         011010
//          0x29         NIBBLE = 9         101001
//          0x2A         NIBBLE = 10        101010
//          0x0B         NIBBLE = 11        001011
//          0x2C         NIBBLE = 12        101100
//          0x0D         NIBBLE = 13        001101
//          0x0E         NIBBLE = 14        001110
//          0x1C         NIBBLE = 15        011100
//
// Using the above Symbol table the hex number 0x31 would be equal to:
// 100011110001b or 0x23 = msb nibble, 0x31 = lsb nibble
// (both being 6 bits long)
//
// Using a 8Mhz xal for the micro
// Timer interrupt can be set for "1" bit time, i.e.
// 52us for 19.2Kbs RF data rate.
// Using a 4Mhz xal for the micro
```



```

    goto    wait_for_hi        // wait for falling edge of the first
                                // zero(invetred data)

gone_hi:                          // it's hi, reload RTCC, sets clk in center of
                                // data

    movlw   1                  // reload at set timer2 to center the clock
                                // **this value will need to be tweaked**
    movwf   0x11               // RTCC address
    bsf     symbol_done        // start symbol flag
    BCF     t0if                // clear timer zero interrupt flag, 0xb,2

clr_cnt:                          // reset start symbol counter, go here do to a
                                // low or start symbol detect
    movlw   0x07               // reload start symbol counter
    movwf   low_counter        // save it
    goto    exit1a             // done with start symbol stuff, skip data
                                // stuff

rx_bit:                          // receive data, one bit at a time
    btfsc   rx_data            // test data input for Hi(data is inverted now)

    goto    dat_low            // here if low
dat_hi:                          // here if hi
    bsf     carry              // here if hi
    rrf     rx_symbol,1        // put new bit in
    goto    exit2              // was hi, now done
dat_low:                          // was low
    bcf     carry              // clear carry
    rrf     rx_symbol,1        // put new bit in

exit2:                          // bit done, now test for symbol done
    decfsz  bit_counter        // symbol done?
    goto    exit1a             // NOP
    movlw   0x07               // reload bit counter
    movwf   bit_counter        // save it

                                // keep track of nibbles
    btfsc   lsb_flag           // if 0, lsb nibble
    goto    msb_nibble         // must be msb nibble

lsb_nibble:                      // now did you guess, LSB nibble
    bcf     carry              // clear carry
    rrf     rx_symbol,1        // get rid of start bit
    bcf     carry              // clear carry
    rrf     rx_symbol,1        // get rid of start bit
    bsf     lsb_flag           //
    goto    sym1

msb_nibble:                      // da, must be MSB nibble, boy your smart
    bcf     carry              // clear carry
    rrf     rx_symbol,1        // get rid of start bit
    movlw   0x3f               // and off msb
    andwf   rx_symbol,1        // turn off lsb
    bcf     lsb_flag           // clr the flag

sym1:

```

```

    btfsc    rx_sym_flag        // if 0, symbol 1
    goto     sym2              // if 1, symbol 2
    movlw    rx_symbol         // I bet we are saving symbol1
    movwf    rx_symbol1       // save it
    bcf      rx_symbol         // ensure we get real data
    bsf      rx_sym_flag       // symbol 2 next time

    goto     exit1a

sym2:
    movlw    rx_symbol         // I bet we are saving symbol1
    movwf    rx_symbol2       // save it
    bcf      rx_symbol         // get ready for next symbol
    bsf      rx_sym_ready      // tell the rest of the code we have a symbol
                                // ready
    bcf      rx_sym_flag       // symbol 1 next time
                                // restore processor flags and return from
                                // interrupt

exit1a:
    incf     time,f           // bump 52us(if 19.2kbs)timer
    BCF      t0if             // clear timer zero interrupt flag, 0xb,2
    bcf      cprb             // clear the int flag

//*****Restore the Proc. state, flags and reg's *****
    movf     save_PCLATH,w
    movwf    0x0a
    SWAPF   save_status,W
    MOVWF   status
    SWAPF   save_w,F
    SWAPF   save_w,W
    #endasm
}

```