Communications Tips and Tricks

Introduction

The stand-alone SPI and I^2C modules include many advanced features that are not present on the traditional MSSP module, which allows for core independent operation using the integrated hardware capabilities of each peripheral. Many devices also now come equipped with an advanced UART module that boasts several new features, and has integrated hardware that supports protocols such as DMX, LIN, and DALI. This document describes some of the features and modes of operation that can be found on these commonly used serial communication peripherals.

Table of Contents

1. SPI Transfer Counter

The SPI module features a transfer counter that allows users the ability to configure how many data transfers will occur in a given transaction. The transfer counter is comprised of the SPIxTCNT registers and is partially controlled using the SPIxTWIDTH register. The transfer counter can operate in two different modes, depending on the configuration of the Bit-Length Mode Select (BMODE) bit. Regardless of the BMODE setting, the TCZIF interrupt flag will be set when the transfer counter decrements to zero. Rather than writing a software loop that controls the number of SPI transfers that occur within a given data transaction, the transfer counter can be configured to do the same thing in hardware, which eliminates the need for software intervention, freeing up the CPU.

Although the transfer counter can control the amount of data and the bit-length of data being transferred by the SPI module, there are still other requirements that must be met for subsequent transmission and reception to occur without disruption. Data will only be transmitted if the Transmit FIFO (TXFIFO) has data written to it and new data will not be received if the Receive FIFO (RXFIFO) is not empty. The user must ensure that any received data is read from the buffer so that it can be cleared to receive new data. The behavior of the transfer counter and SPI module is dependent on the status of the BMODE Configuration bit.

1.1 Total Bit Count Mode (BMODE = 0)

In this mode of operation, the SPIxTCNTH/L register pair and the SPIxTWIDTH register are concatenated to determine the total number of bits to be transferred. The total number of bits that will be transmitted equals (SPIxTCNT * 8) + SPIxTWIDTH. When using the SPI module in this mode of operation, the user can control the total number of bits being transmitted by writing the appropriate values to the SPIxTCNT and SPIxTWIDTH registers. The following example shows how the SPI module can be configured to send 45 bits of data, demonstrating Total Bit Count mode. Using these configuration settings in this example, the SPI module will transmit five 8-bit bytes of data along with one final packet that is five bits wide as specified by the SPIxTWIDTH register setting.

Example 1-1. Total Bit Count Mode Configuration

```
uint8_t SPI_Exchange8bit(uint8_t data) {
    SPIxCON1 = 0x00;
    SPIxCON2 = 0x03;            // Full-Duplex Mode (TXR = RXR = 1)
    SPIxBAUD = 0x00;
    SPIxCLK = 0x00;
    SPIxINTEbits.TCZIE = 1;     // Transfer Counter is Zero Interrupt Enabled
    SPIxTCNTL = 0x05;           // SPI1TCNT
    SPIxTCNTH = 0x00;
    SPIxTWIDTH = 0x05;          // SPI1TWIDTH
    SPIxCON0 = 0x82;            // EN = 1, LSBF = 0, MST = 1, BMODE = 0

    SS1_SetHigh();
    while (SPIxINTFbits.TCZIF == 0x0) {
        SPIxTXB = data;
        while (PIR2bits.SPIxRXIF == SPI_RX_IN_PROGRESS) {
        }
        data_RX = SPIxRXB;
    }
    SS1_SetLow();
    SPIxINTFbits.TCZIF = 0;
}
```

1.2 Variable Transfer Size Mode (BMODE = 1)

In Variable Transfer Size mode, the SPIxTWIDTH register is used to determine the width of each data transfer and the SPIxTCNTH/L register pair specifies the number of transfers of that bit-length that will occur. If SPIxTWIDTH is equal to zero, each data transfer will be a full byte of data, otherwise the length of the data transfer is whatever value was written to the SPIxTWIDTH register. The following example shows how the SPI module can be configured to transfer data that is five bits wide (SPIxTWIDTH = 0x05) a total of 12 times demonstrating Variable Transfer Size mode.

Example 1-2. Variable Transfer Size Configuration

```
uint8_t SPI_Exchange8bit(uint8_t data) {

    SPIxCON1 = 0x00;
    SPIxCON2 = 0x03;            // Full-Duplex Mode (TXR = RXR = 1)
    SPIxBAUD = 0x00;
    SPIxCLK = 0x00;
    SPIxINTEbits.TCZIE = 1;     // Transfer Counter is Zero Interrupt Enabled
    SPIxTCNTL = 0x0C;           // SPI1TCNT
    SPIxTCNTH = 0x00;
    SPIxTWIDTH = 0x05;          // SPI1TWIDTH
    SPIxCON0 = 0x83;            // EN = 1, LSBF = 0, MST = 1, BMODE = 0

    SS1_SetHigh();
    while (SPIxINTFbits.TCZIF == 0x0) {
        SPIxTXB = data;
        while (PIR2bits.SPIxRXIF == SPI_RX_IN_PROGRESS) {
        }
        data_RX = SPIxRXB;
    }
    SS1_SetLow();
    SPIxINTFbits.TCZIF = 0;
}
```

2. SPI Transfer-Only Mode

The SPI module found on newer PIC18 devices feature the ability to operate in Transmit-Only mode. This mode may be useful in applications where the SPI master only needs to transmit data and the data being received either does not matter or does not exist. When the SPI operates in Full-Duplex (Legacy) mode, received data is stored in the receive buffer and the data must be read from the RXFIFO before the SPI will continue data transfers. In Transmit-Only mode, data that is received from the slave device is not stored in the Receive FIFO and the SPI master will continue transmitting data via the transmit buffer (TXFIFO) without the need to clear the receive FIFO with every data transfer.

The TXR and RXR bits of the SPIxCON2 register determine the Transfer/Receive mode that the SPI module operates in. To operate in Transmit-Only mode, the TXR bit must be set and the RXR bit needs to be cleared. The following example shows how the SPI module can be used in Transmit-Only mode to send eight bytes of data. Please note that in this code snippet it is not necessary to poll the SPI Receive Buffer Interrupt flag because the SPI module is configured to only transmit data. In full-duplex operation, the user would need to intentionally clear the receive buffer with every transmission to keep the transaction going.

Example 2-1. SPI Transmit-Only Configuration

```
uint8_t SPI_Exchange8bit(uint8_t data) {

    SPIxCON1 = 0x00;
    SPIxCON2 = 0x02;            // Transmit Only Mode (TXR = 1; RXR = 0)
    SPIxBAUD = 0x00;
    SPIxCLK = 0x00;
    SPIxINTEbits.TCZIE = 1;     // Transfer Counter is Zero Interrupt Enabled
    SPIxTCNTL = 0x08;           // SPIxTCNT
    SPIxTCNTH = 0x00;
    SPIxTWIDTH = 0x00;          // SPIxTWIDTH
    SPIxCON0 = 0x82;            // EN = 1, LSBF = 0, MST = 1, BMODE = 0

    SS1_SetHigh();
    while (SPIxINTFbits.TCZIF == 0x0) {
        SPIxTXB = data;
    }
    SS1_SetLow();
    SPIxINTFbits.TCZIF = 0;
}
```

3. SPI Receive-Only Mode

The SPI module can be configured to operate in Receive-Only mode. This mode is useful in applications where the SPI master only needs to receive data and the contents of the data being transmitted to the slave device are not important. When configured in Receive-Only mode, the SPI will only begin receiving data when a non-zero value is written to the transfer counter. The SPI will then continue to receive bytes of data from the slave device until the transfer counter has decremented to zero or the SPI module has been disabled. Receive-Only mode performs the write to the transmit buffer in hardware, removing this responsibility from the user.

The TXR and RXR bits of the SPIxCON2 register determine the Transfer/Receive mode that the SPI module operates in. To operate in Receive-Only mode, the RXR bit must be set and the TXR bit needs to be cleared. The following example shows how to configure the SPI module for Receive-Only mode. In this example, the transfer counter has been set up to receive eight bytes of data from a slave device.

Example 3-1. SPI Receive-Only Configuration

```
uint8_t SPI_Exchange8bit(uint8_t data) {

    SPIxCON1 = 0x00;
    SPIxCON2 = 0x01;            // Receive Only Mode (TXR = 0; RXR = 1)
    SPIxBAUD = 0x00;
    SPIxCLK = 0x00;
    SPIxTCNT = 0x08;            // SPIxTCNT
    SPIxCON0 = 0x82;            // EN = 1, LSBF = 0, MST = 1, BMODE = 0

    while (SPIxTCNT > 0) {
        data_RX = SPIxRXB;
    }
}
```

4. Enabling Direct Hardware Control of the SPI Slave Select Output

The SPI protocol allows a master device to interface one or more slave devices, all on the same bus. When more than one slave device is connected to a SPI master, each slave should have an independent Slave Select (or Chip Select) connection from the master device that is used to determine which slave is being communicated with at a given time. The master selects the slave that it will communicate with by enabling the Slave Select line for that slave on the bus. Traditionally, a GPIO would need to be dedicated to serve as the Slave Select, which required that the user set/clear the pin as needed in software, whenever a SPI communication was to occur.

The stand-alone SPI module features the ability to control the SPI Slave Select output in hardware, removing the need to enable/disable the Slave Select line in software. To enable hardware control of the SPI Slave Select output, the desired pin must be configured using PPS as the “SPIxSS”. In addition to using PPS to choose a Slave Select output, the corresponding TRISx setting must also be configured as an output. Once these configuration steps have been completed, every time that the SPI master initiates communication, the Slave Select line will be enabled and disabled, as needed in hardware. The Slave Select Input/Output Polarity Control (SSP) bit of the SPIxCON1 register can be used to select whether the Slave Select is active-high or active-low. Example 4-1 below contains a code snippet demonstrating how the PPS and TRIS settings can be configured to enable hardware control of the Slave Select output.

Example 4-1. Pin Configuration Settings for SPI1 with Hardware Slave Select Control

```
void PIN_MANAGER_Initialize(void) {

    TRISE = 0x07;
    TRISA = 0xFF;
    TRISB = 0xFF;
    TRISC = 0xAF;
    TRISD = 0xEF;        //RD4 = Output (Slave Select Output)
    .
    .
    .
    SPI1SCKPPS = 0x16;    //RC6->SPI1:SCK1;
    RC4PPS = 0x1F;        //RC4->SPI1:SDO1;
    RC6PPS = 0x1E;        //RC6->SPI1:SCK1;
    RD4PPS = 0x33;        //RD4->SPI1:SS1;(Refer to PPS table for this value!)
    SPI1SDIPPS = 0x15;    //RC5->SPI1:SDI1;
}
```

5. Configuring I²C Bus Time-out Feature

This I²C module allows the user the option to implement a configurable bus time-out in both Master and Slave modes of operation. The bus time-out will occur whenever the Serial Clock (SCL) line has been held low, beyond the amount of time specified in the user configuration. The I²C protocol specification does not require a specific bus time-out period and typically, the slave device can hold (clock-stretch) the clock line low indefinitely. The SMBus protocol requires a bus time-out of 35 ms, and the PMBus™ protocol requires a bus time-out period of 25 to 35 ms. The purpose of implementing a bus time-out is to prevent stalled devices from indefinitely holding the bus during communication. Although not required by the normal I²C specification, a defined bus time-out may be advantageous as a fail-safe, in events where a stall in communication occurs.

In the event where a device on the bus stalls and the time-out expires, the module will be Reset and forced into an Idle state. The I²C bus time-out can be used in both Master and Slave modes. Refer to the device data sheet for more information about how bus time-outs and the respective interrupts are handled in each mode of operation. To configure the bus time-out, the I²C Bus Time-Out register (I2CxBTO) is used to select the timer resource that serves as the time-out time base. The Bus Time-Out Interrupt Flag (BTOIF) has a corresponding enable bit (BTOIE) that the user can set if they wish to implement an Interrupt Service Routine when a time-out occurs.

The following code snippet shows a simple initialization routine that enables the I²C as a master device and uses Timer4 as the bus time-out source. The bus time-out period has been configured to be 35 ms in this code snippet, to comply with SMBus and PMBus™ protocol specifications. With this implementation, Timer4 will begin counting when SCL is inactive, and if SCL is held in an Inactive state for more than 35 seconds, the module will force itself to Reset.

Example 5-1. Setting up the I²C Bus Time-out using Timer4

```
// Configure Timer4 (I2C Bus Time-Out Source) – 35ms
T4CLKCON = 0x06;        // Timer4 Clock Source = MFINTOSC 31.25 KHz
T4HLT = 0x00;
T4RST = 0x00;
T4PR = 0xDA;            // Timer4 PR = 218
T4TMR = 0x00;
PIR7bits.TMR4IF = 0;    // Clear Timer4 Interrupt Flag
T4CON = 0x84;           // T4CKPS 1:1, T4OUTPS 1:5, TMR4ON on

// Configure I2C Master with Bus Time-Out
I2CxCON1 = 0x80;
I2CxCON2 = 0x00;
I2CxCLK = 0x00;         // I2C CLK = FOSC / 4
I2CxCON0 = 0x87;        // I2C EN = 1, Mode = Master Mode (SMBus 2.0)
I2CxPIR = 0x00;         // Clear I2C Interrupt Flags
I2CxPIE = 0x08;         // ADRIE = 1 (Must be enabled in SMBus Mode)
I2CxERR = 0x00;         // Clear I2C Error Register
I2CxBTO = 0x02;         // Bus Time-Out Selection = TMR4 Postscaler Out
I2CxERRbits.BTOIE = 1;  // Enable I2C Bus Time-Out InterruptsSetting up the I2C
```

6. Using the I²C Data Byte Count

The I2CxCNT register is used to specify the length in bytes of a complete message in an I²C data transaction. The value of the I2CxCNT register will be decremented in hardware every time a byte of data has been transmitted or received by the module, until it has decremented to zero. Once the I2CxCNT register decrements to zero, the Byte Count Interrupt Flag (CNTIF) bit will be set, signifying the end of that message and data transfer will halt until the I2CxCNT register has been written to again. This feature is unique to the stand-alone I²C module and can be used to send variable length messages with minimal software intervention.

The example below demonstrates how the I²C module can be used to transmit 'N' bytes of data using the Data Byte Counter (I2CxCNT) register. In this code snippet, the value used for the variable 'len' would be written to the I2CxCNT register and will decrement every time data is written to the slave device. The same principal can be applied to read 'N' bytes of data as the I2CxCNT register should be used for both receiving and transmitting data on the I²C bus.

Example 6-1. Writing 'N' Bytes of Data using the I2CxCNT Register

```
void WriteNBytesRegister(uint8_t address, uint8_t reg, void* data, uint8_t len)
{
   uint8_t *dataPointer = data;
   I2CxADB1 = (uint8_t)(address << 1); // Load address with write = 0
   I2CxTXB = reg;                      // Load beginning slave register address
   I2CxCNT = len;                      // Load with size of array to write
   I2CxCON0bits.S = 1;                 // Set Start to get things going
   while(I2CxCNT)                      // While count is true
     {
        while(!I2CxSTAT1bits.TXBE);    // Wait until buffer is empty before
load
        I2CxTXB = *dataPointer++;      // Load next byte to transmit,
     }
                                       // Hardware decrements I2CxCNT
    wait4Stop();                       // Wait for hardware to issue Stop
}
```

7. Auto-Loading the I²C Byte Count Register

The stand-alone I²C module features the ability to configure the I2CxCNT register to be automatically loaded in hardware by enabling the Auto Load I²C Count (ACNT) register enable bit. When auto-loading of the I²C Count register has been enabled, the first received or transmitted data byte following the address will be loaded into the I2CxCNT register in hardware. The value of the Acknowledge Data (ACKDT) bit will be used for the ACK response, which prevents a false NACK bit from being generated before the I2CxCNT register has been updated. Auto-loading the I2CxCNT register is a valid configuration for all I²C modes of operation and can be very useful in applications where the master needs information from the slave about the size of an incoming data packet or vice versa.

Example 7-1. Auto-Loading I²C Byte Count

```
void WriteNBytes_AutoLoad(uint8_t address, uint8_t reg, void* data, uint8_t
len){
      uint8_t *dataPointer = data; // Data Packet for transmission
      // When ACNT = 1, First Transmitted/Received Byte
      // Following Address is Loaded into I2CxCNT
      I2CxCON2bits.ACNT = 1; // Auto-Load Count Enabled
      I2CxTXB = len; // Transmits length, automatically Loads len into I2CxCNT
      I2CxCON0bits.S = 1; // Set start bit to transmit 0x03 (I2CxCNT)
      while (!I2CxSTAT1bits.TXBE); //Waiting for the buffer to be empty

      I2CxCON0bits.S = 1; // Set Start bit to transmit data packet;
      while (I2CxCNT) // While Count is true
      {
            while (!I2CxSTAT1bits.TXBE); // Wait until buffer is empty;
            I2CxTXB = *dataPointer++; // Load next byte to transmit;
      }
      wait4Stop(); // Wait for hardware to issue Stop
}
```

8. External Pull-Up Resistor Selection

Serial Data (SDL) and Serial Clock (SCL) are the two signal connections used for communication in the I²C protocol. Both SDL and SCL are open-drain lines, meaning that each pin requires a pull-up resistor to bring the signal high when it is not being driven low by the I²C module hardware. The I²C specification proposes two methods to determine the correct pull-up resistor size, both of which can be used to calculate and appropriate range for pull-up resistors in specific applications.

The first method is to calculate the maximum pull-up resistor size as a function of the total bus capacitance (C_{BUS}) and maximum rise time (T_{RISE}). Bus capacitance is total capacitance of the bus wires/traces, connection points, pins, all of which must be considered when calculating the total bus capacitance. Rise time is the period in which the signal transitions from V_{ILMAX} to V_{IHMIN} and can typically be found in the device data sheet. To achieve the most accurate pull-up values using this method, the total bus capacitance would need to be measured. Rather than attempting to measure the total bus capacitance, the maximum allowable bus capacitance as defined by the I²C specification can be used instead.

Example 8-1 demonstrates how the maximum pull-up resistor size can be calculated using the I²C module on the microcontroller in Fast mode (400 kHz). The value used for the maximum rise time can be found in the I²C bus data requirements in the Electrical Specification section of the device data sheet. The maximum allowable bus capacitance must meet the maximum rise time in this application (300 ns) and is about 200 pF, as defined by the I²C specification.

Example 8-1. Calculating the Maximum I²C Pull-up Resistor Size (PIC18F57Q43)

$$R_{P\ (MAX)} = \frac{T_{RISE}}{0.8473\ \times\ C_{BUS}} = \frac{300\ ns}{0.8473\ \times 200\,pF} = 1.77\ k\Omega$$

The second method calculates the minimum pull-up resistor size as a function of V_{DD}. V_{DD} limits the minimum resistor value that can be used as a pull-up due to the specified minimum sink current (I_{OL}) of 3 mA for I²C in Standard mode (100 kHz) and Fast mode (400 kHz), or 20 mA for Fast mode plus (1 MHz). Example 8-2 demonstrates how the minimum pull-up resistor size can be calculated when using the I²C module on the PIC18F57Q43 microcontroller in Fast mode (400 kHz). In this example, it can be assumed the PIC18F57Q43 is operating at a V_{DD} of 3.3V and has a maximum output low voltage (V_{OL} (max)) of 0.7V. The maximum output low voltage can typically be found in the Electrical Specifications section of the device data sheet.

Example 8-2. Calculating the Minimum I²C Pull-up Resistor Size (PIC18F57Q43)

$$R_{P\ (MIN)} = \frac{VDD\ -\ V_{OL\ (MAX)}}{I_{OL}} = \frac{3.30V\ -\ 0.6V}{3\ mA} = 900\,\Omega$$

9. Configuring Internal Pull-ups on Default I²C pins

Newer device families feature dedicated I²C pads that provide the ability to reconfigure certain GPIO characteristics to better meet the I/O requirements of I²C communication. The weak pull-ups on standard GPIO pins are typically not strong enough to be used on the I²C bus and instead external pull-up resistors of the appropriate values are typically incorporated into the design. The I²C specific pads allow users to program the weak pull-ups on those pins to allow either twice or ten times the standard weak pull-up current for a GPIO. In many applications, these stronger internal pull-ups can sufficiently drive the I²C bus and can be used in place of external pull-up resistors.

Table 9-1 compares the standard GPIO weak pull-ups to the higher current configurations that are available on the dedicated I²C pads. For the calculations, the typical weak pull-up current value of 140 µA was used from the PIC18F57Q43 electrical specifications. It can also be assumed that for the purposes of these calculations, the operating voltage V_{DD} is equal to 3.0V. These calculations are meant to serve as an approximation and can be different for each application, depending upon many other factors, such as operating voltage and the specified weak pull-up current.

Table 9-1. Calculating Internal Pull-ups based on the I²C Pull-up Selection Bits - PU[1:0]

PU[1:0]	I²C Pull-up Selection Bits	Internal Pull-up Value (R_{PU})
11	Reserved	Reserved
10	10x Current of Standard Weak Pull-up	$RPU = \frac{V_{DD}}{I_{PUR}} = \frac{3.0V}{10 \times 140\,\mu A} = 2.1\,k\Omega$
01	2x Current of Standard Weak Pull-up	$RPU = \frac{V_{DD}}{I_{PUR}} = \frac{3.0V}{2 \times 140\,\mu A} = 10.7\,k\Omega$
00	Standard GPIO Weak Pull-up	$RPU = \frac{V_{DD}}{I_{PUR}} = \frac{3.0V}{140\,\mu A} = 21.4\,k\Omega$

The internal pull-ups on I²C dedicated pads can be controlled by writing to the PU[1:0] bits of the I²C pad Rxy Control (RxyI2C) register. In order to use the PU[1:0] bits for internal pull-up selection, weak pull-ups must be enabled by setting the corresponding WPUx bits. Once weak pull-ups have been enabled on the I²C dedicated pins being used for SDL and SCL, they will default to the standard GPIO weak pull-up strength. If the user wishes to use the internal pull-ups as opposed to external pull-ups on the bus, the PU[1:0] bits would need to be set accordingly to adjust the strength of the internal pull-ups to either twice or ten times the current.

10. UART Module with Protocol Support – Setting up a DMX Master (Controller)

The UART module with built-in protocol support can be configured for DMX communication, which is an industry standard for stage lighting and other theatrical effects. The module manages the timing requirements and generates the correct sequence of individual events required as a DMX master, effectively reducing the amount of firmware needed for communication. Example 10-1 demonstrates how a device that has this UART module can be configured as a DMX Controller. Note that the PPS settings might not be the same for different devices, so minor changes to the code snippet may be required. The key configuration settings needed to configure the UART module as a DMX controller are:

- MODE[3:0] = `1010` - selects the DMX mode
- TXEN = `1` - enables the transmitter
- RXEN = `0` - disables the receiver
- TXPOL = `0` - selects regular polarity
- STP[1:0] = `10` - for two Stop bits
- UxP1 = one less than the number of bytes to transmit (excluding the start code)
- UxBRGH:L = value to achieve 250K baud rate
- RxyPPS = TX pin output code
- ON = `1`

Example 10-1. Setting up a DMX Master (Controller) using the UART with Protocol Support

```
void UART_Initialize(){
 UxCON0bits.MODE = 0b1010;      //Select DMX mode
 UxCON0bits.TXEN = 1;           //Enable transmitter
 UxCON0bits.RXEN = 0;           //Disable receiver
 UxCON2bits.TXPOL = 0;          //Standard polarity, TX pin will idle high
 UxCON2bits.STP = 0b10;         //2 stop bits

 //DMX data packet settings
 UxP1 = MAXCHANNELS-1;          //Total number of data bytes - 1
 UxP2 = 0x00;                   //Not used in DMX controller
 UxP3 = 0x00;                   //Not used in DMX controller

 //Baud rate generator settings
 UxCON0bits.UxBRGS = 1;         //High speed baud generation
 UxBRG = 0x3F;                  //Value for UxBRG for Fosc = 64MHz

 //PPS settings for TX functionality (May vary device to device)
 ANSELxbits.ANSELxy = 0;        //Make Rxy a digital I/O
 TRISxbits.TRISxy = 0;          //Make Rxy an output
 RxyPPS = 0b010011;             //Assign TX functionality to Rxy
 UxON = 0x01;                   //Turn on UART module
}
```

11. UART Module with Protocol Support – Setting up a DMX Receiver

The UART module with protocol support can be configured as a DMX receiver and includes the built-in hardware support to automatically detect and respond to a valid Break condition. The UART with protocol module features a configurable address range, in which the module will ignore all the data bytes outside of the specified range in a DMX packet. In addition to the configurable address range, the module can also be setup to wait for and verify the reception of two Stop bits at the end of a DMX packet by setting the STP[1:0] bits of the UxCON2 register accordingly.

The start address of the configurable address range can be specified by writing to the UART Parameter 2 (UxP2) register and the end address can be set using the UART Parameter 3 (UxP3) register. The UART module only needs to be configured once as a DMX receiver and does not need to be updated for every DMX packet, unless the address range has changed. Example 11-1 demonstrates how a device that has this UART module can be configured as a DMX Receiver. The key configuration settings needed to configure the UART module as a DMX Receiver are:

- MODE[3:0] = `1010` - selects the DMX mode
- TXEN = `0` - disables the transmitter
- RXEN = `1` - enables the receiver
- RXPOL = `0` - selects regular polarity
- STP[1:0] = `10` for verifying the reception of two Stop bits
- UxP2 = address of first byte to receive
- UxP3 = address of last byte to receive
- UxBRGH:L = value to achieve 250K baud rate
- UxRXPPS = code for desired input pin
- Selected input pin should be made a digital input by clearing the corresponding ANSEL bit
- ON = `1`

Example 11-1. Setting up a DMX Master (Controller) using the UART with Protocol Support

```
void UART_Initialize(){
 UxCON0bits.MODE = 0b1010;       //Select DMX mode
 UxCON0bits.TXEN = 0;            //Disable transmitter
 UxCON0bits.RXEN = 1;            //Enable receiver
 UxCON2bits.RXPOL = 0;           //Standard polarity, RX pin will idle high
 UxCON2bits.STP = 0b10;          //Recevie and verify 2 stop bits

 //DMX data packet settings
 UxP1 = 0x00;                    //Not used in DMX receiver
 UxP2 = 0x10;                    //Address of first byte of interest
 UxP3 = 0x20;                    //Address of last byte of interest

 // Baud rate generator settings
 UxCON0bits.UxBRGS = 1;          //High speed baud generation
 UxBRG = 0x3F;                   //Value for U1BRG for Fosc = 64MHz

 //PPS settings for RX functionality (May vary device to device)
 ANSELxbits.ANSELxy = 0;         //Make Rxy a digital I/O
 TRISxbits.TRISxy = 0;           //Make Rxy an input
 UxRXPPS = 0b010111;             //Assign RX functionality to RC7
 UxON = 0x01;                    //Turn on UART module
}
```

12. DALI Communication using UART Module with Protocol Support

Digital Addressable Lighting Interface (DALI) is a widely used digital lighting control protocol that is used for intelligent lighting control for building automation. The UART module with protocol has built-in support for a DALI control device or controlgear. DALI communication uses Manchester encoding, which is performed using the UART hardware.

Tip: This section will only cover configuring the UART module for DALI communication. For more details about the protocol and how it can be implemented, see the UART with DALI Protocol Technical Brief.

A DALI control device is an application controller than is used to transmit commands to the other devices (light fixtures) connected on the same bus. The UART module has an available mode of operation that can be used for DALI communication that is configured using the UART Mode Select (MODE[3:0]) bits of the UART Control Register 0 (UCON0). Example 12-1 shows a code snippet of how the UART module can be configured as a DALI control device. A DALI control gear is a device that receives commands in order to control its output from a control device. In most applications, the control gear will be some type of light fixture that is connected on the same bus as the control device. Example 12-2 shows how the UART module can be configured as a DALI control gear. For both code examples, note that minor changes may be required, depending on the device being used.

Example 12-1. DALI Control Device Configuration

```
void UART_Initialize(){
 UxCON0bits.MODE = 0b1000;          //Select DALI Control Device Mode
 UxCON0bits.TXEN = 1;               //Enable transmitter
 UxCON0bits.RXEN = 1;               //Enable receiver
 UxCON2bits.TXPOL = 0;              //Appropriate polarity for interface circuit
 UxCON2bits.STP = 0b10;             //Receive and verify 2 stop bits

 //data packet settings
 UxP1 = 0x01;                       //Forward Frame Hold Time (Half-bit Periods)
 UxP2 = 0x10;                       // Forward/backward frame threshold
delimiter

 // Baud rate generator settings
 UxBRG = 0xCF;                      //Value for 1200 baud rate @ 1MHz

 //PPS settings for TX functionality (May vary device to device)
 ANSELxbits.ANSELxy = 0;            //Make Rxy a digital I/O
 TRISxbits.TRISxy = 0;              //Make Rxy an output
 RxyPPS = 0b010011;                 //Assign TX functionality to Rxy
 UxON = 0x01;                       //Turn on UART module
}
```

Example 12-2. DALI Control Gear Configuration

```
void UART_Initialize(){
 UxCON0bits.MODE = 0b1001;          //Select DALI Control Gear Mode
 UxCON0bits.TXEN = 1;               //Enable transmitter
 UxCON0bits.RXEN = 1;               //Enable receiver
 UxCON2bits.TXPOL = 0;              //Appropriate polarity for interface circuit
 UxCON2bits.TXPOL = 0;              //Same as RXPOL
 UxCON2bits.STP = 0b10;             //Receive and verify 2 stop bits

 //data packet settings
 UxP1 = 0x01;                       //Back Frame Hold Time (Half-bit Periods)
 UxP2 = 0x10;                       //Forward/backward frame threshold
                                    //delimiter

 //Baud rate generator settings
 UxBRG = 0xCF;                      //Value for 1200 baud rate @ 1MHz

 //PPS settings for RX functionality (May vary device to device)
 ANSELxbits.ANSELxy = 0;            //Make Rxy a digital I/O
 TRISxbits.TRISxy = 0;              //Make Rxy an input
 UxRXPPS = 0b010111;                //Assign RX functionality to RC7

 //PPS settings for TX functionality (May vary device to device)
 ANSELxbits.ANSELxy = 0;            //Make Rxy a digital I/O
 TRISxbits.TRISxy = 0;              //Make Rxy an output
 RxyPPS = 0b010011;                 //Assign TX functionality to Rxy
 UxON = 0x01;                       //Turn on UART module
}
```

13. Configuring the UART Module for LIN

Local Interconnect Network (LIN) is a serial network protocol that is used primarily in automotive applications and consists of a LIN cluster that has one master and up to 16 slave modes on the bus. The UART module with protocol support has hardware built-in that handles the serial communication required by the LIN protocol and can be implemented using either the master process or slave process. With the combined hardware support and a LIN software library, users can get started with basic master and slave reception/transmission. In the LIN protocol, the master controls the bus activity, while the slave(s) send or receive data based on the scheduled tasks.

Tip: This section will only cover configuring the UART module for LIN bus communication. For more details about the protocol, scheduling events and to see any available software libraries, see the device data sheet or visit http://www.microchip.com.

LIN Master mode has the capabilities of generating slave processes. Any data that is transmitted in Master/Slave mode is done as a slave process. Example 13-1 shows a basic initialization routine that can be used to configure the UART module with protocol support in LIN Master/Slave mode. Example 13-2 shows an initialization routine that can be used to configure the UART module with protocol support in LIN Slave mode.

Example 13-1. LIN Master/Slave Mode Configuration

```
void UART_Initialize(){
 UxCON0bits.MODE = 0b1100;          //Select LIN Master / Slave mode
 UxCON0bits.TXEN = 1;               //Enable transmitter
 UxCON0bits.RXEN = 1;               //Enable receiver
 UxCON2bits.TXPOL = 0;              //Standard polarity, High Idle State
 UxCON2bits.STP = 0b10;             //Desired Stop Bits Selection
 UxCON2bits.C0EN = 0;               //Desired Checksum Mode, Legacy LIN checksum

 // Baud rate generator settings
 UxBRG = 0x3F;                      //Value to achieve desired baud rate

 //PPS settings for TX functionality (May vary device to device)
 ANSELxbits.ANSELxy = 0;            //Make Rxy a digital I/O
 TRISxbits.TRISxy = 0;              //Make Rxy an output
 RxyPPS = 0b010011;                 //Assign TX functionality to Rxy
 UxON = 0x01;                       //Turn on UART module
}
```

Example 13-2. LIN Slave-only Mode Configuration

```
void UART_Initialize(){
 UxCON0bits.MODE = 0b1011;          //Select LIN Slave mode
 UxCON0bits.TXEN = 1;               //Enable transmitter
 UxCON0bits.RXEN = 1;               //Enable receiver
 UxCON2bits.TXPOL = 0;              //Standard polarity, High Idle State
 UxCON2bits.STP = 0b10;             //Desired Stop Bits Selection
 UxCON2bits.C0EN = 0;               //Desired Checksum Mode, Legacy LIN checksum

 //Data packet settings
 UxP2 = 0x10;                       // Number of Data Bytes to Transmit
 UxP3 = 0x10;                       // Number of Data Bytes to Recieve


 // Baud rate generator settings
 UxBRG = 0x3F;                      //Value to achieve desired baud rate

 //PPS settings for TX functionality (May vary device to device)
 ANSELxbits.ANSELxy = 0;            //Make Rxy a digital I/O
 TRISxbits.TRISxy = 0;              //Make Rxy an output
 RxyPPS = 0b010011;                 //Assign TX functionality to Rxy
 UxON = 0x01;                       //Turn on UART module
}
```

14. Using the UART Module to Transmit Data

The UART module with protocol support allows users to perform serial data transfers and can easily be configured to transmit data. To configure the UART for data transmission, the Transmit Enable (TXEN) bit of UxCON0 must be set and the MODE[3:0] bits of UxCON0 must be set as '0000' through '0011', depending on the application. The Baud Rate Generator (BRG) must also be configured and can be accessed using the UxBRG register and the BRGS bits of UxCON0. Data is transmitted via the TX pin of the microcontroller, which is PPS remappable using the RxyPPS register. The UART module is enabled by setting the ON bit of UxCON1.

Once the UART module has been initialized as a transmitter, data can be transmitted by writing to the UART Transmit Buffer (UxTXB) register. The UART Transmit Interrupt Flag (UxTXIF) bit is set when UART transmission has been enabled and the UxTXB register is ready to accept data. Example 14-1 demonstrates the initialization of the UART module for data transmission.

Example 14-1. UART Transmit Mode Configuration

```
void UART_Initialize(){
    U1CON0 = 0xA0;              // BRGS normal; 8-bit mode; RXEN disabled;
                                // TXEN enabled
    U1CON2 = 0x00;              // TXPOL not inverted; FLO off;
    U1BRGL = 0x19;              // BRGL 25 = 9600 BR @ 1 MHz FOSC
    U1BRGH = 0x00;
    U1FIFO = 0x00;              // STPMD in middle of first Stop bit;
    U1UIR = 0x00;               // Auto-baud not enabled
    U1ERRIR = 0x00;
    U1ERRIE = 0x00;
    RC6PPS = 0x13;              // UART PPS Settings, RC6->UART1:TX1;
    U1RXPPS = 0x17;             // UART PPS Settings, RC7->UART1:RX1;
    U1CON1 = 0x80;              // ON enabled;
}
```

15. Receiving Data using the UART Module

To enable the receiver part of the UART module with protocol support, the Receive Enable (RXEN) bit of the UxCON0 register must be set. The MODE[3:0] bits should be programmed with any of the configurations between '0000' and '0011' depending on the application. The Baud Rate Generator (BRG) must also be configured using the UxBRG register and the BRGS bits of UxCON0, to match the baud rate of the other device. Data is received via the RX pin of the microcontroller, which can be remapped using PPS by writing the code for the desired input pin to the RxyPPS register. The UART module must then be enabled by setting the ON bit of UxCON1.

After configuring the UART module as a receiver, when a received byte of data is transferred from the Receive Shift register to the Receive Buffer (UxRXB) register, the UART Receiver Interrupt Flag (UxRIF) bit becomes set. If the UART Receiver Interrupt Enable (UxRXIE) bit has been set, an interrupt will be generated any time the UxRXB register contains received data. The data received and contained within the UART receive buffer must be read from the UxRXB register and the UxRXIF bit should be cleared before more data can be received. The UART Error Interrupt Flag register can be used to monitor if any errors were detected during the data transfer. Example 15-1 demonstrates the initialization of the UART module as a receiver.

Example 15-1. UART Receive Mode Configuration

```
void UART_Initialize(){
    U1CON0 = 0x90;              // BRGS normal; 8-bit mode; RXEN enabled;
                                // TXEN disabled
    U1CON2 = 0x00;              // TXPOL not inverted; FLO off;
    U1BRGL = 0x19;              // BRGL 25 = 9600 BR @ 1 MHz FOSC
    U1BRGH = 0x00;
    U1FIFO = 0x00;              // STPMD in middle of first Stop bit;
    U1UIR = 0x00;               // Auto-baud not enabled
    U1ERRIR = 0x00;
    U1ERRIE = 0x00;
    RC6PPS = 0x13;              // UART PPS Settings, RC6->UART1:TX1;
    U1RXPPS = 0x17;             // UART PPS Settings, RC7->UART1:RX1;
    U1CON1 = 0x80;              // ON enabled;
}
```

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

Trademarks

ISBN: 978-1-5224-5888-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820