AN1076

Using a PIC® Microcontroller for DMX512 Communication

Author: Parthiv Pandya
Microchip Technology Inc.

INTRODUCTION

DMX512 is a communication protocol used in most professional theater lighting components such as dimmers, scanners, moving lights, strobes, etc. This application note presents a solution to transmit and receive the DMX512 communication protocol that can be implemented using any PIC® microcontroller offering a Universal Asynchronous Receiver Transmitter (UART) module. In particular, the PIC18F24J10, a general purpose device, was used in the code examples provided with this application note. It provides 1024 bytes of data memory, which allows the demonstration code to store the data for the entire 512 channel buffer (although this is not required for the typical application). Only an external RS-485 compatible transceiver is required to complete the application schematic.

The DMX solution is provided in two parts:

1. DMX512 Transmitter:

This part will explain how to generate and transmit the DMX512 packets. This is divided into two subsections:

(a) how to generate and transmit the DMX512 packets

and

(b) a demo program that shows how to send commands to a DMX512 light dimming receiver.

2. DMX512 Receiver:

This part will explain how to receive the DMX512 packets. Once more, it is divided into two subsections:

(a) how to receive the data

and

(b) a demo program that sends the received data to the PWM module to control the brightness of a LED.

BACKGROUND

In the past, variable auto-transformers were used to control theatre stage lights. That required long wires around the stage to supply electricity to the lamps and a whole team would be required to manually control thetransformers. Later, electric motors were connected to the auto-transformers, which made the controlling less cumbersome. Eventually, analog controls took the place of auto-transformers, becoming quite popular, particularly the 0-10V analog consoles. Still, this system had three major drawbacks:

1. It was prone to noise.
2. Dimming could be nonlinear depending on different kinds of lamps.
3. A separate control wire was required for each lamp.

As computer technology became more cost effective, new digital consoles came to the market and with them the need for a new standard that would allow equipment from different manufacturers to interoperate.

The United States Institute of Theatre Technology, USITT, first developed the DMX512 protocol in 1986 as a standard digital interface between dimmers and consoles, later expanded and improved in 1990. The current version, known as DMX512-A, has also been adopted as an American National Standards Institute (ANSI) standard (E1.11). The development of DMX512-A is currently managed by the Entertainment Services & Technology Association (ESTA). You can obtain (purchase) a copy of the protocol specifications from the www.esta.org web site or the www.ansi.org web site.

ANATOMY OF THE DMX512 PROTOCOL

DMX512 (an acronym for Digital MultipleX), is extremely simple, low cost and relatively robust. Due to these advantages DMX512 has gained a great popularity. As the name suggests, it can support up to 512 separate control channels/devices. It is a unidirectional asynchronous serial transmission protocol which does not provide for any form of handshake between receiver and transmitter, nor does it offer any form of error checking, or correction mechanism. Hence, it is not suitable for any safety critical application. Data is transmitted at 250k baud rate using a physical interface compatible with the RS-485 transmission standard over two wires and ground.

A DMX512 system has only one transmitter and multiple receivers. A DMX512 transmitter connects a DMX512 receiver via XLR 5-pin or XLR 3-pin connectors. A female connector is connected to a transmitter and a male connector on a receiver. The specification states that 2 pairs of shielded cables should be used.

However, the use of a second cable is optional.Table 1 shows the physical pinout when a XLR 5-pin connector is used.

TABLE 1: XLR 5-PIN CONNECTOR

XLR Pin Number	DMX 512 Application	Function
1	Common	Common Reference
2	DMX Data 1-	Primary Data link
3	DMX Data 1+	
4	DMX Data 2-	Secondary (Optional) Data link (Unimplemented for 3 pin XLR connector)
5	DMX Data 2+	

Note: XLR connectors are commonly used in professional audio, video and lighting applications. The connector has a rugged shell and a locking mechanism.

Each DMX512 transmitter sends 512 8-bit dimming values, between 0 and 255, where 0 represents the lights off and 255 represents the maximum intensity.

Each receiver connected to the DMX512 line can choose one of the 512 channels (address selection) to control its output lamp (load).

The DMX512 protocol requires the transmitter to continuously repeat (at least once a second) the transmission of a frame as shown in the timing diagram in Figure 1 and Table 2.

FIGURE 1: DMX512 TIMING DIAGRAM

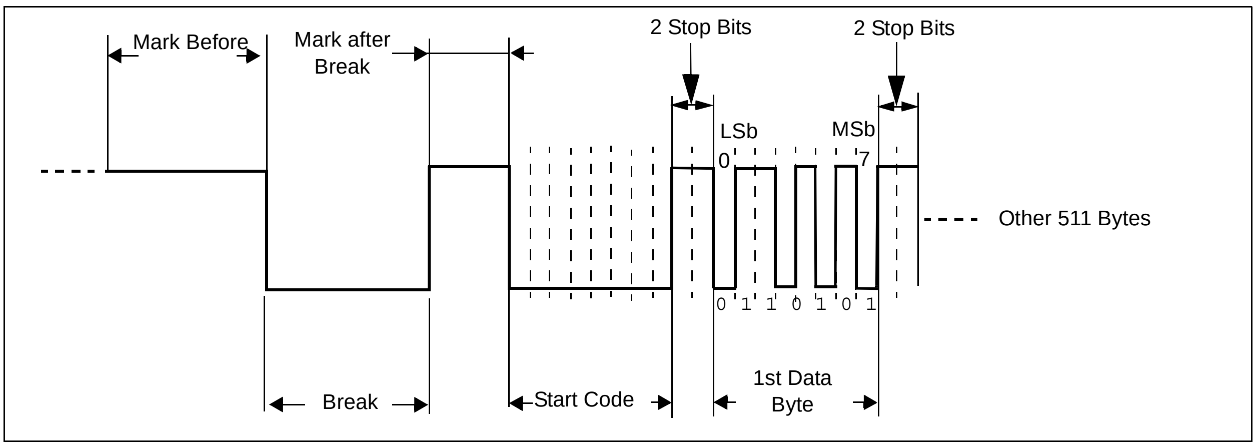


TABLE 2: DMX512 TIMING VALUES

Description	Minimum	Maximum	Typical	Unit
Break	92	—	176	µSec
Mark after Break	12	<1,000,000	—	µSec
Bit Time	3.92	4.02	4	µSec
DMX512 Packet	1204	1,000,000	—	µSec

DMX512 TRANSMITTER

To generate the DMX512 packets, the software solution employs a simple state machine comprised of four states:

1. `SENDMBB` – DMX data line is Idle
2. `SENDDATA` – Bytes 0 to 511 of the DMX frame
3. `SENDMAB` – DMX data line is Idle
4. `SENDBREAK` – DMX data line is driven low

FIGURE 2: TRANSMITTER STATE MACHINE

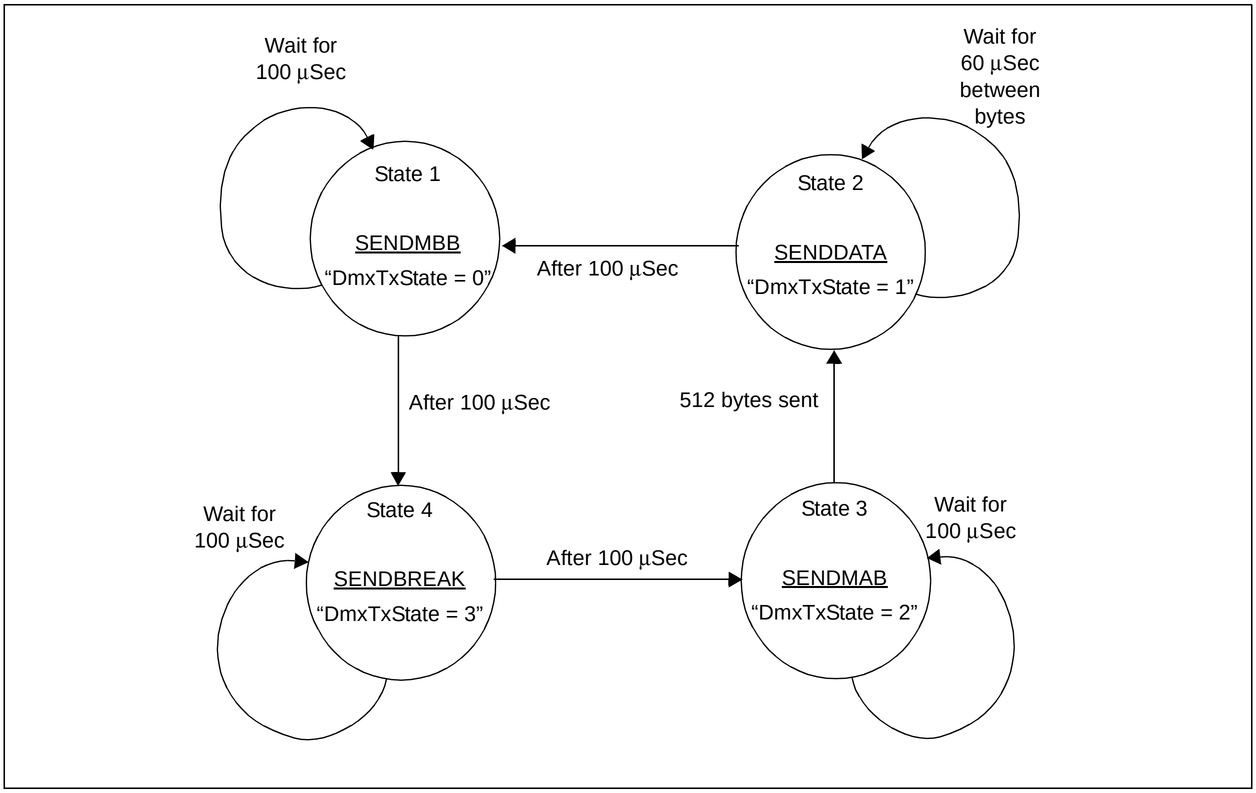


Figure 2 shows the state machine. In this application, to simplify the code and still remain within the timing constrains, the `SENDBREAK, SENDMAB` and `SENDMBB` intervals were all set to 100 µSec. These timings can be easily changed if required. The Timer0 module is used to control the 100 µSec timing and the spacing between the transmitted bytes.

EXAMPLE 1: DMX512 TRANSMITTER STATE MACHINE CODE

```
;Jump Table
DMXTransmit:
    rlncf       DmxTxState,W
    andlw       0x0E
    addwf       PCL
    bra         SENDMBB
    bra         SENDDATA
    bra         SENDMAB
    bra         SENDBREAK

SENDMBB
    .
    .
    return

SENDDATA
    .
    .
    return
SENDMAB
    .
    .
    return

SENDBREAK
    .
    .
    return
```

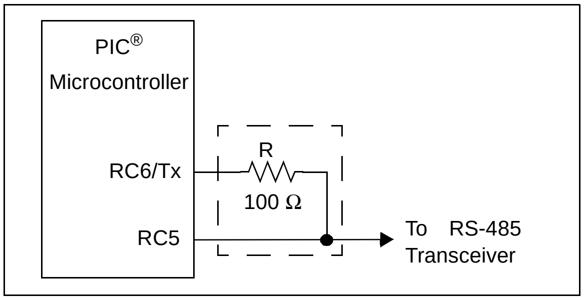
Example 1 shows the outline of the DMXTransmit subroutine implementing the state machine.

The DMXTransmit subroutine is designed for use in a cooperative multitasking application. To avoid any timing issues, the state machine should be called frequently enough (approximately every 40 µs or less) from the main program loop. The DmxTxState variable is used to represent the current state and as an offset in a jump table to access the corresponding code segment in the state machine subroutine.

GENERATING THE BREAK SIGNAL

The Break signal allows receivers to synchronize with the DMX transmitter identifying the beginning of a new packet of data.The EUSART module available on most PIC18 microcontrollers has the ability to automatically generate a 12-bit long Break signal, corresponding to 48 µs at 250k baud. Unfortunately, this is too short for use in a DMX512 application as the protocol requires a minimum length of 92 µSec. Figure 3 shows the alternative hardware method chosen in this application note to generate the longer Break signal. A 100Ω resistor is connected in series with the microcontroller's EUSART transmit pin and the other end of the resistor to an I/O pin. In the specific example, pin RC5 was used. With this solution, the Break time can be varied in software, from 92 µSec to 176 µSec to meet the DMX protocol Break time specification, when sending a Break signal, pin RC5 is driven low. Later Pin RC5 is tri-stated to allow the transmission from the EUSART to resume.

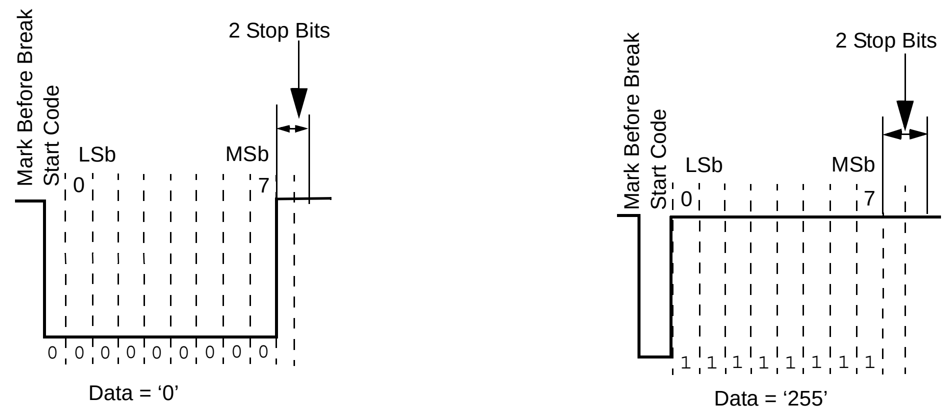
FIGURE 3: GENERATING A LONG BREAK SIGNAL



SENDING THE DIMMING DATA

The dimming data is 8-bits wide, where '0' represents a light off and '255' represents full intensity. Figure 4 shows the digital representation of the dimming data. To generate the two Stop bits required by the DMX512 protocol, the PIC18 EUSART is configured for 9-bit mode and the 9th bit is set permanently to '1'.

FIGURE 4: DIGITAL REPRESENTATION OF DIMMING DATA



The dimming data is stored in a 512 bytes buffer (TxBuffer), allocated in the PIC18F24J10 RAM memory. The data is written to or read from the buffer using the indirect addressing registers available on PIC18 microcontroller architecture for linear memory access. A counter keeps track of the number of bytes transmitted from the buffer.

Note:	Although the demonstration code stores and transmits the dimming data for all 512 channels it can be easily modified to store and transmit only a subset of channels, while leaving all remaining channels off (0). This could reduce considerably the MCU RAM requirements for a reduced functionality transmitter.

TRANSMITTER APPLICATION DEMO: DIMMING A LAMP

In the previous section we saw that it is very easy to generate a DMX512 packet using a PIC18F device. In this demonstration application, we will use a potentiometer connected to the DMX512 transmitter to control remotely a lamp attached to a standard DMX512 receiver.

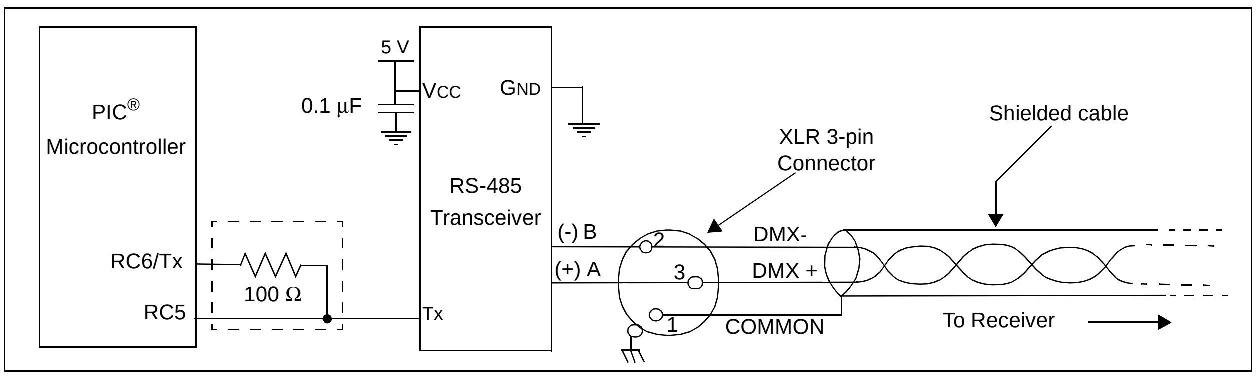
The PIC18F24J10 has a 10-bit Analog-to-Digital Converter module with 13 inputs. The potentiometer can be connected on pin RA0 of the MCU corresponding to the analog input channel 0.

Since the potentiometer won't change very rapidly, sampling it every 10 mSec is sufficient.To generate an automatic and periodic activation of the Analog-to-Digital Converter, a convenient feature of the PIC18F24J10 microcontroller can be used. The ADC module can, in fact, start periodically a new conversion triggered by the Capture Compare and PWM module (CCP). The 16-bit Timer1 module is used in conjunction with the CCP module configured in 16-bit Compare mode. When the compare trigger occurs (Timer1 = CCPR1), the ADC conversion starts on the preselected input channel and Timer1 is reset.

When the ADC conversion is complete a new result is loaded into the ADRESH register and the ADIF flag is set.

When the ADIF bit is detected in the main loop, the transmitter will retrieve from ADRESH the Most Significant 8 bits encoding the potentiometer position and will transfer them to the transmission buffer at the position corresponding to the desired channel. The same channel will be selected at the dimming receiver for demonstration.

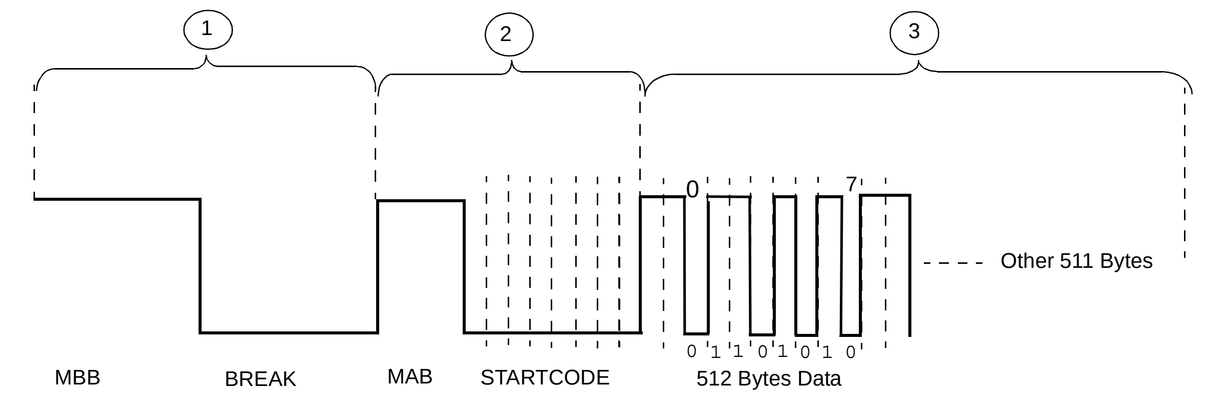
FIGURE 5: DMX512 TRANSMITTER CIRCUIT SCHEMATIC



Note:	Please see **Appendix A: "DMX512 Transmitter Demo"** for a complete code listing of the transmitter demo.

A SIMPLE DMX512 RECEIVER

FIGURE 6: RECEIVING A DMX512 PACKET



The problem of receiving a DMX512 packet can be decomposed in three parts.

1. The first part is the synchronization of the receiver with the beginning of a new data packet identified by a prolonged Break condition of the line. This condition can be conveniently identified by a Framing error flag reported by the UART. In fact, when the line is taken to the Break level, at the beginning of a new DMX512 packet, the UART initially interprets the condition as the beginning of a new data byte. But when, after the duration of the Start bit and 8 more data bits, instead of the two Stop bits (mark) the line remains in the Break condition, a frame error is reported.

   Since there is no way to predict at which point of a transmission sequence the receiver will be activated, during this phase the UART is polled continuously in a loop to discard any data received until a first framing error is detected.
2. Once the Break condition is identified, the receiver needs to wait for the line to return to the Idle state (mark) and a first byte of data to arrive. During this phase the UART is polled continuously as frame errors continue to be detected. Eventually the first byte received correctly is interpreted as the Start code. In this simple application only frames with a Start code of 0 are received, frames beginning with a different Start code (DMX512 extensions) are ignored.
3. The last part consists, once more, of a loop where the receiver captures up to 512 bytes of data and stores them sequentially in the receiver buffer. A 12-bit pointer, available in the PIC18 architecture, is used to provide linear memory access to the RAM memory space.

RECEIVER APPLICATION DEMO

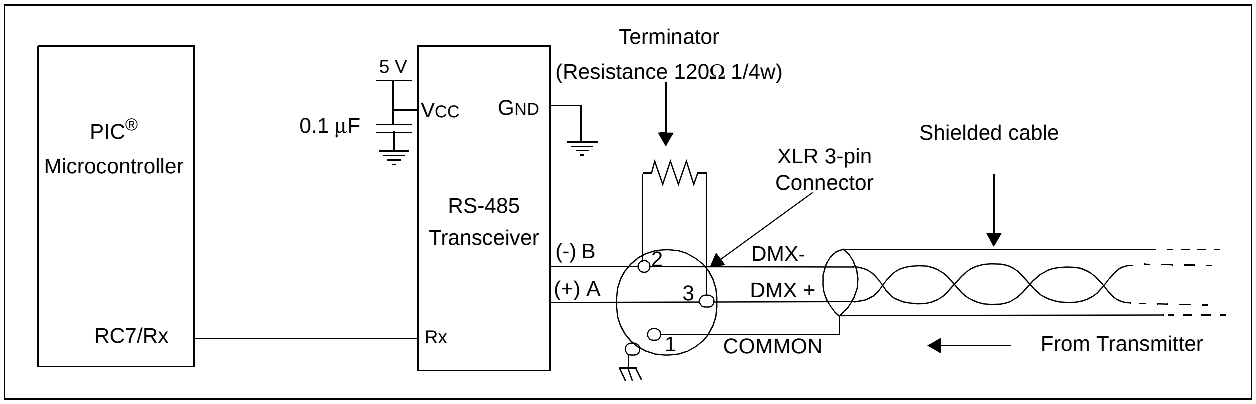
In the previous section we saw how to get the DMX512 data for 512 channels and to store them into a receiver buffer. In this section we will use the received data to control the PWM module of a PIC microcontroller. Connecting a LED to the PWM output pin we will observe the LED brightness change in response to DMX512 dimming commands.

The PIC18F24J10 Capture Compare and PWM (CCP) module offers 10-bit resolution. When used in PWM mode, it uses Timer2 as its time base and the PR2 register determines the PWM period. Since the DMX512 protocol provides only 8-bit of resolution for each channel, setting the PR2 register to ‘0xFF’ allows us to use just the 8 Most Significant bits to control the duty cycle while still providing a PWM output frequency of approximately 16 kHz. This value greatly exceeds the minimum requirement, of approximately 100 Hz, usually considered sufficient to eliminate any visible flicker of the LED.

Since the Most Significant 8 bits of the PWM duty cycle are controlled by the CCPR2L register, it is sufficient to periodically update it copying the contents of the location corresponding to the desired DMX512 address (defined by the constant CHANNEL) from inside the receive buffer.

In the demonstration code, the CCPR2L register is updated every time a complete DMX512 frame has been received.

FIGURE 7: DMX512 RECEIVER CIRCUIT SCHEMATIC



Note:	Please see **Appendix B: "DMX512 Receiver Demo"** for a complete code listing of the receiver demo.

In the schematic, the EUSART receiver pin is connected to the RS-485 transceiver's receiver output pin. A 120Ω, ¼ W resistor should be connected between DMX- and DMX+ data link as a line terminator. Figure 7 shows the line terminator between pin 2 (DMX- data link) and pin 3 (DMX+ data link) of an XLR-3 connector. Proper Termination greatly reduces signal transmission problems.

TESTING SETUP

To test the DMX512 transmitter and receiver, a separate pair of PICDEM™ 2 PLUS demo boards was used. The PICDEM 2 PLUS can be used to demonstrate the capabilities of 18, 28 and 40-pin PIC16 and PIC18 devices. The board has a small prototyping area where the transmitter and receiver transceiver circuits can be built.

In order to take advantage of the (4) LEDs available on the board for the receiver demo, the output of the PIC18F24J10 CCP2 module can been redirected to PORTB output pin RB3 by modifying the microcontroller nonvolatile Configuration register CONFIG3H, 'CCP2 MUX' bit.

INTERRUPT

The provided transmitter and receiver demonstration code uses the polling method to transmit and receive the DMX512 packets. The CPU is waiting for a timer to expire to generate the mark and the Break signals or for the EUSART to transmit or receive the data. To reduce the CPU polling time, the provided code can be written using interrupts.

CONCLUSION

This application note presents a very simple software solution to generate, transmit and receive the DMX512 signals using a low-cost MCU.

REFERENCES

1. PIC18F24J10 Data sheet (DS39682)

   The data sheet provides all the necessary information regarding the EUSART module, CCP module, ADC module and electrical characteristics of the PIC microcontroller.

2. PICDEM™ 2 PLUS User's Guide (DS51275)

   This application note has been tested using a pair of PICDEM 2 PLUS demo boards.

3. American National Standard E1.11 – 2004.

   The official DMX512 protocol specifications are available on www.esta.org.

APPENDIX A: DMX512 TRANSMITTER DEMO

```
; File:  DMX512TrmtDemo.asm
; DMX512 Transmitter demo
;
; This source code uses the PIC18F24J10 to transmit a DMX-512 packet via
; the EUSART peripheral.  An external 16MHz clock input is used.
; The DMX transmitter code is written as a polled state machine with
; 4 states.  The state machine is called periodically from the main
; software loop and a jump table determines the present state.
; Timer0 is used to control the state machine timing, including length
; of the Break signal and the spacing between transmitted bytes.
; The CCP module is configured to start an ADC conversion every 10msec.
; A potentiometer voltage is sampled with the ADC and the result is
; written to the first data slot in the DMX frame to control a remote
; device.

      list p=18f24j10               ; define target processor
      #include <p18f24j10.inc>      ; include processor specific definitions

; Configuration bits setup
      CONFIG     CCP2MX = ALTERNATE ; assign CCP2 output to pin RB3
      CONFIG     WDTEN = OFF        ; To use ICD2 as a debugger disable Watch Dog Timer
      CONFIG     STVERN = ON        ; Reset on stack overflow/underflow enabled
      CONFIG     XINST = OFF        ; Instruction set extension and Indexed Addressing
                                    ; mode disabled (Legacy mode)
      CONFIG     CP0 = OFF          ; Program memory is not code-protected
      CONFIG     FOSC = ECPLL       ; EC oscillator, PLL enabled and under software
                                    ; control, CLKO function on OSC2
      CONFIG     FOSC2 = ON         ; Clock selected by FOSC as system clock is enabled
                                    ; when OSCCON<1:0> = 00
      CONFIG     FCMEN = OFF        ; Fail-Safe Clock Monitor disabled
      CONFIG     IESO = OFF         ; Two-Speed Start-up disabled
      CONFIG     WDTPS = 32768      ; 1:32768

; Timing constants (assuming 16MHz clock input and assigned prescaler
; values to produce 1us tick)
#define T100US.256-.100             ; preload value for TMR0 to roll over in 100us
#define T60US.256-.60               ; 60us value

; Variables memory allocation
      CBLOCK     0x008
      DmxTxState                    ; State Variable
      CountH                        ; 16-bit counter
      CountL
      TxBuffer: .512                ; allocate 512 bytes for the transmit buffer
      ENDC
```

```
;******************************************************************************
      ORG        0x0000

Main
      rcall      InitTX              ; initialize the I/O ports and TMR0
      rcall      SetupSerial         ; initialize serial comm
      rcall      SetupADC            ; initialize the ADC for the demo

;******************************************************************************
;Main Application Loop

MainLoop
      rcall      DMXTransmit         ; Execute the state machine
      rcall      CheckADC            ; Check to see if ADC conversion is complete.
      goto       MainLoop


;**************************************************************************************************
;DMX Transmit state machine

DMXTransmit
      ; The DMX transmit code is driven by the TMR0 roll-over
      ; events.  Just return if a roll-over has not occured.
      btfss      INTCON,TMR0IF       ; wait until TIMER0 roll-over
      return
      bcf        INTCON,TMR0IF       ; clear the flag

      clrf       PCLATH              ; (assumes the jump table is located in
                                     ; the first page of program memory)
      rlncf      DmxTxState,W        ; state x2 to account for PC byte
                                     ; addressing
      andlw      0x0E                ; reduce offset to valid range (0-14)
      addwf      PCL                 ; computed jump

; Jump Table
      bra        SENDMBB             ; 0 IDLE period after each complete frame
      bra        SENDDATA            ; 1 send one byte of data
      bra        SENDMAB             ; 2 IDLE period between BREAK and START slot
      bra        SENDBREAK           ; 3 BREAK synchronization signal
      reset                          ; not used
      reset                          ; not used
      reset                          ; not used
      reset                          ; not used

; DmxTxState = 3. Generates a Break Signal (100uSec)
SENDBREAK
      bsf        TRISC,5             ; tri-state pin RC5 to end break signal
      movlw      T100US              ; preload TIMER0 for a roll over in 100us
      movwf      TMR0L

      decf       DmxTxState,F        ; proceed to State2 SENDMAB
      return

; DmxTxState = 2. Mark After Break (line IDLE for 100uSec) send a start code
SENDMAB
      clrf       CountL              ; init 16-bit counter
      clrf       CountH
      lfsr       1,TxBuffer          ; init pointer to transmit buffer

      clrf       TXREG               ; send NULL START CODE
      movlw      T60US               ; pre-load TMR0 for a short delay (> (12bit x 4us) >48us)
      movwf      TMR0L
      decf       DmxTxState,F        ; proceed to state1 SENDDATA
      return
```

```
; DmxTxState = 1. wait for UART to complete transmission of current byte and an additional short
; amount of time
SENDDATA

     btfsc      CountH,1           ; check if 512 slot sent already
     bra        TXDone

     btfss      PIR1,TXIF          ; make sure TX buffer is available
     return

     movff      POSTINC1,TXREG     ; send a new byte of data (use IND1 pointer to read data from
                                   ; TX buffer)
                                   ; automatically advance pointer 1
     incf       CountL,F           ; increment 16-bit counter
     btfsc      STATUS,C
     incf       CountH,F

         movlw      T60US              ; pre-load TMR0 for a short delay (> (12bit x 4us) >48us)
         movwf      TMR0L
         return

TXDone
         movlw      T100US             ; pre-load TMR0 for a 100us delay before the frame repeats
         movwf      TMR0L
         decf       DmxTxState,F       ; proceed to next state SENDMBB
         return

;DmxTxState = 0. sends Mark Before repeating the frame transmission
SENDMBB
         movlw      T100US             ; pre-load the timer for 100us BREAK
         movwf      TMR0L
         bcf        INTCON,TMR0IF      ; clear the flag

         bcf        TRISC,5            ; make pin RC5 an output
         bcf        LATC,5             ; pull pin RC5 low to force a break condition

         movlw      .3                 ; proceed to State3 SENDBREAK
         movwf      DmxTxState
         return


;******************************************************************************
;CheckADC verify a new conversion result is available and copy the value to 6 channels/location in
; the TX buffer

CheckADC
         btfss      PIR1,ADIF          ;check the flag for ADC conversion completed
         return

         bcf        PIR1,ADIF          ; clear the ADC flag
         bcf        PIR2,CCP2IF        ; clear the Compare flag

         lfsr       0,TxBuffer         ; use indirect pointer IND0 to copy the conversion result
         movff      ADRESH,POSTINC0    ; to the first slot in the transmit buffer (->1)
         movff      ADRESH,POSTINC0    ; slot 2
         movff      ADRESH,POSTINC0    ; slot 3
         movff      ADRESH,POSTINC0    ; slot 4
         lfsr       0,TxBuffer + .508
         movff      ADRESH,POSTINC0    ; slot 509
         movff      ADRESH,POSTINC0    ; slot 510
         movff      ADRESH,POSTINC0    ; slot 511
         movff      ADRESH,POSTINC0    ; slot 512
         ; Note:  This code places the transmit data in the first 4 data slots
         ; and the last 4 data slots of the DMX data frame.  This was done to
         ; make sure that the code worked properly with a 4-channel dimmer
```

```
        ; unit that was used during code development.  Add code above as
        ; required to fill other slots with transmit data.

        return

;******************************************************************************
;Setup Serial port

SetupSerial

        bsf         TRISC,7             ; allow the UART RX to control pin RC7
        bsf         TRISC,6             ; allow the UART TX to control pin RC6

        movlw       0x65                ; enable TX, 9-bit mode, high speed mode, 9th bit =1
                                        ; (2 stop)
        movwf       TXSTA

     movlw       0x80                ; enable serial port, disable receiver
     movwf       RCSTA

     bsf         BAUDCON,BRG16       ; select EUART 16-bit Asynchrnounou mode operation

     movlw       .15                 ; init baud rate generator for 250k baud (assume Fosc=16MHz)
     movwf       SPBRG

     return

;******************************************************************************
;ADC setup
SetupADC
     bsf         TRISA,0             ; make RA0 an input pin
     movlw       0x01                ; enable ADC and select input channel 0
     movwf       ADCON0

     movlw       0x0E                ; make only channel 0 an analog input pin
     movwf       ADCON1

     movlw       0x35                ; ADC result left aligned and clock = Fosc/16
     movwf       ADCON2

;Set the CCP2 module in Compare mode with a 10mSec interval, CCPR2 = 10.000us
     movlw       0x27
     movwf       CCPR2H

     movlw       0x10
     movwf       CCPR2L

;A/D Conversion started by the Special Event Trigger of the CCP2 module
     movlw       0x0B
     movwf       CCP2CON

;init Timer1 as the time base for CCP2
     clrf        TMR1H
     clrf        TMR1L
     movlw       0x21                ; enable 16-bit Timer1, prescale 1:4 (1us tick@16MHz),
                                     ; internal clock
     movwf       T1CON

     return

;******************************************************************************
;InitTX          init Timer0, clear TXbuffer, init state machine
InitTX
     clrf        CountL              ; init 16-bit counter
     clrf        CountH
```



```
; clear Transmit buffer
     lfsr       1,TxBuffer         ; use IND1 pointer to address the RAM buffer
CBloop
     clrf       POSTINC1           ; clear the location pointed to by IND1 then increment pointer
     incf       CountL,F           ; increment 16-bit counter
     btfss      STATUS,C
     bra        CBloop
     incf       CountH,F

     btfss      CountH,1           ; check if counter >= 512
     bra        CBloop

; init Timer0
         movlw      0xC1           ; enable Timer0, as an 8-bit timer, use prescaler 1:4
                                   ;(1us tick@16MHz)

         movwf      T0CON

         movlw      T100US             ; preload timer for 100us interval to roll over
         movwf      TMR0L
         bcf        INTCON,TMR0IF      ; clear roll over flag

; init state machine
         movlw      .03                ; Start with BREAK state
         movwf      DmxTxState

         bcf        TRISC,5            ; make pin RC5 an output
         bcf        LATC,5             ; pull RC5 output low to force a break condition

         return


         END
```

APPENDIX B: DMX512 RECEIVER DEMO

```
; File:  DMX512RecDemo.asm
; DMX512 Receiver
; This file uses a PIC18F24J10 device to receive DMX-512 data and store it
; into a 512 byte receive buffer.
; For demonstration purposes, a selected data slot is written to the
; CCP module.  The CCP module is configured in PWM mode and the received
; data adjusts the duty cycle.  If a resistor and LED is connected to the
; PWM output, the received DMX data can be visually observed.

      list p=18f24j10               ;define target processor
      #include <p18f24j10.inc>      ;include processor specific definitions

; Configuration bits setup
      CONFIG     CCP2MX = ALTERNATE ; assign CCP2 output to pin RB3
      CONFIG     WDTEN = OFF        ; To use ICD2 as a debugger disable Watch Dog Timer
      CONFIG     STVERN = ON        ; Reset on stack overflow/underflow enabled
      CONFIG     XINST = OFF        ; Instruction set extension and Indexed Addressing
                                    ; mode disabled (Legacy mode)
      CONFIG     CP0 = OFF          ; Program memory is not code-protected
      CONFIG     FOSC = ECPLL       ; EC oscillator, PLL enabled and under software
                                    ; control, CLKO function on OSC2
      CONFIG     FOSC2 = ON         ; Clock selected by FOSC as system clock is enabled
                                    ; when OSCCON<1:0> = 00
      CONFIG     FCMEN = OFF        ; Fail-Safe Clock Monitor disabled
      CONFIG     IESO = OFF         ; Two-Speed Start-up disabled
      CONFIG     WDTPS = 32768      ; 1:32768


; Constants
#define CHANNEL .510                ;select the receiver slot/channel

; Variables
      CBLOCK     0x8
          CountH                    ;16-bit counter
          CountL
          RxBuffer: .512            ;512 bytes buffer allocation

      ENDC

;******************************************************************************
      ORG        0x0

Main
      call       SetupSerial        ;Setup Serial port and buffers

MainLoop

; first loop, synchronizing with the transmitter
WaitBreak
      btfsc      PIR1,RCIF          ; if a byte is received correctly
      movf       RCREG,W            ; discard it
      btfss      RCSTA,FERR         ; else
      bra        WaitBreak          ; continue waiting until a frame error is detected
      movf       RCREG,W            ; read the Receive buffer to clear the error condition

; second loop, waiting for the START code
WaitForStart
      btfss      PIR1,RCIF          ; wait until a byte is correctly received
      bra        WaitForStart
      btfsc      RCSTA,FERR
      bra        WaitForStart
      movf       RCREG,W
```

```
; check for the START code value, if it is not 0, ignore the rest of the frame
      andlw      0xff
      bnz        MainLoop           ; ignore the rest of the frame if not zero

; init receive counter and buffer pointer
      clrf       CountL
      clrf       CountH
      lfsr       0,RxBuffer

; third loop, receiving 512 bytes of data
WaitForData
      btfsc      RCSTA,FERR         ; if a new framing error is detected (error or short frame)
      bra        RXend              ; the rest of the frame is ignored and a new synchronization is
                                    ; attempted

      btfss      PIR1,RCIF          ; wait until a byte is correctly received
      bra        WaitForData        ;
      movf       RCREG,W            ;

MoveData
      movwf      POSTINC0           ; move the received data to the buffer
                                    ; (auto-incrementing pointer)
      incf       CountL,F           ; increment 16-bit counter
      btfss      STATUS,C
      bra        WaitForData
      incf       CountH,F

      btfss      CountH,1            ; check if 512 bytes of data received
      bra        WaitForData


;******************************************************************************
; when a complete frame is received
; use the selected CHANNEL data to control the CCP2 module duty cycle

RXend
      lfsr       0,RxBuffer         ; use indirect pointer 0 to address the receiver buffer
GetData
      movlw      LOW(CHANNEL)       ; add the offset for the select channel
      addwf      FSR0L,F
      movlw      HIGH(CHANNEL)
      addwfc     FSR0H,F

      movff      INDF0,CCPR2L       ; retrieve the data and assign MSB to control PWM2

      bra        MainLoop           ; return to main loop


;******************************************************************************
; Setup Serial port and buffers
SetupSerial

;Clear the receive buffer
      lfsr       0,RxBuffer
CBloop
      clrf       POSTINC0           ; clear INDF register then increment pointer
      incf       CountL,F
      btfss      STATUS,C
      bra        CBloop
      incf       CountH,F

      btfss      CountH,1
      bra        CBloop
```

```
; Setup EUSART
      bsf         TRISC,7             ; allow the EUSART RX to control pin RC7
      bsf         TRISC,6             ; allow the EUSART TX to control pin RC6

      movlw       0x04                ; Disable transmission
      movwf       TXSTA               ; enable transmission and CLEAR high baud rate

      movlw       0x90
      movwf       RCSTA               ; enable serial port and reception

      bsf         BAUDCON,BRG16       ; Enable UART for 16-bit Asyn operation
      clrf        SPBRGH

      movlw       .15                 ; Baud rate is 250KHz for 16MHz Osc. freq.
      movwf       SPBRG

;Setup PWM module
      movlw       0x0c                ; configure CCP2 for PWM mode
      movwf       CCP2CON

;Timer2 control
      movlw       0x04                ; enable Timer2, select a prescale of 1:1
      movwf       T2CON

;PWM period
      movlw       0xFF                ; 256 x .25us = 64us period
      movwf       PR2

;init I/O
      movlw       b'11110111'         ; make pin RB3 (CCP2) output
      movwf       TRISB


      return

      END
```

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

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
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Habour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

12/08/06