AN1076

Using a PIC® Microcontroller for DMX512 Communication

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 the transformers. 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.

Author: Parthiv Pandya
Microchip Technology Inc.
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**

<table>
<thead>
<tr>
<th>XLR Pin Number</th>
<th>DMX 512 Application</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Common</td>
<td>Common Reference</td>
</tr>
<tr>
<td>2</td>
<td>DMX Data 1-</td>
<td>Primary Data link</td>
</tr>
<tr>
<td>3</td>
<td>DMX Data 1+</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DMX Data 2-</td>
<td>Secondary (Optional) Data link (Unimplemented for 3 pin XLR connector)</td>
</tr>
<tr>
<td>5</td>
<td>DMX Data 2+</td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Description</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Typical</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break</td>
<td>92</td>
<td>—</td>
<td>176</td>
<td>μSec</td>
</tr>
<tr>
<td>Mark after Break</td>
<td>12</td>
<td>&lt;1,000,000</td>
<td>—</td>
<td>μSec</td>
</tr>
<tr>
<td>Bit Time</td>
<td>3.92</td>
<td>4.02</td>
<td>4</td>
<td>μSec</td>
</tr>
<tr>
<td>DMX512 Packet</td>
<td>1204</td>
<td>1,000,000</td>
<td>—</td>
<td>μSec</td>
</tr>
</tbody>
</table>
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

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.
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 pre-selected 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.
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 in a loop to discard any data received until a first framing error is 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.
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.

**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.

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

The software supplied herewith by Microchip Technology Incorporated (the “Company”) is intended and supplied to you, the Company’s customer, for use solely and exclusively with products manufactured by the Company. The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

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 WDTE = ON ; To use ICD2 as a debugger disable Watch Dog Timer
CONFIG STVER = OFF ; 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
Dmx TxState ; 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 Loop
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 occurred.
btfs 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
andlw 0x0E ; reduce offset to valid range (0-14)
addwf PCL ; computed jump
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
clr T2CL ; init 16-bit counter
clr T2CH ; init pointer to transmit buffer
lsr 1,TxBuffer ; 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

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

  btfs  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
  btssc 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

  btfs  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 Asynchronous 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 at 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 WDTE = OFF ; To use ICD2 as a debugger disable Watch Dog Timer
CONFIG STVREN = ON ; Instruction set extension and Indexed Addressing
CONFIG XINST = OFF ; mode disabled (Legacy mode)
CONFIG CP0 = OFF ; Program memory is not code-protected
CONFIG FOSC = ECPLL ; EC oscillator, PLL enabled and under software
CONFIG FOSC2 = ON ; Clock selected by FOSC as system clock is enabled
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
btfsb PIR1,RCIF ; if a byte is received correctly
movf RCREG,W ; discard it
btfsb RCSTA,FERR ; else
bra WaitBreak
movf RCREG,W ; read the Receive buffer to clear the error condition

; second loop, waiting for the START code
WaitForStart
btfsb PIR1,RCIF ; wait until a byte is correctly received
bra WaitForStart
btfsb 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
  bnez MainLoop ; ignore the rest of the frame if not zero

; init receive counter and buffer pointer
  clr CountL
  clr 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)
  addwf 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
  clr 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   ; Baud rate is 250KHz for 16MHz Osc. freq.
  movlw .15
  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
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.

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 your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademark

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELLOO, KEELLOO logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries. AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MxLAB, PS logo, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2007, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona, Gresham, Oregon and Mountain View, California. The Company’s quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELLOO® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001:2000 certified.