Freescale Semiconductor
Application Note

Document Number: AN3315
Rev. 0, 09/2006

DMX512 Protocol Implementation Using MC9S08GT60 8-Bit MCU

by: Oscar Luna
Daniel Torres
RTAC Americas

1 Overview

This application note details the implementation of the DMX512 using the MC9S08GT60 microcontroller.

Illumination entertainment markets have adopted DMX512 protocol due to its simplicity and characteristics and the necessity to unify a single illumination standard. It has been adapted in the high-brightness LED (HBLED) lighting industry for light sources in microscopes and even fixtures in ships and submarines. This market attracts new players as low-end MCUs enable new applications. In many cases, these new applications will be controlled by a central control unit requiring a connectivity interface. In the lighting market today, connectivity is dominated by two protocols: DMX512 and DALI.

The newly-enabled applications using HBLEDs and low-end MCUs include automotive signage, mobile appliances, and illumination. Other novel uses for LEDs range from water purification modules to automated

Contents

systems for cultivating vegetables through the combined use of RGB LEDs.

2 DMX512 Protocol Overview

The DMX512 protocol is simple because it is an asynchronous 8-bit serial protocol and works in an unidirectional line generated by a master device (or console). The protocol can handle up to 512 devices in a DMX network and communicates at 250 kbps baud rate. Each bit in the frame is generated every 4us.

Every slave in the network is addressed using one specific byte in the frame. Each byte in the frame describes 256 possible dimming levels on each slave. The media is driven using ANSI/TIA/EIA-485-A-1998 balanced data-transmission techniques. Physical connection between devices is via 5-pin XRL connectors or by hard-wiring the terminals.

2.1 Physical Layer

The EIA-485 are the protocol’s electrical specifications. The EIA-485-A standard does not point any advice related to isolation issues but DMX512 standard does. Any suitable isolation method such as optical isolation, transformer isolation, or other means may be used to avoid the undesirable propagation of voltages exceeding limits based on the EIA-485-A standard.

The data link is compounded by a single, active differential-line driver, a terminated transmission line, and one or more differential-line receivers. A DMX512 connection point has five assigned end-points: data 1-, data1+, data2-, data2+ and a common reference. In some cases, data2- and data 2+ are not used or they simply carry the same signals of the first differential line (data1- and data1+). In both cases, there should be a low impedance connection between data link and common pin or contact of the DMX512 port.

The data link must have a terminator to eliminate ringing issues or signal reflection. The terminator must be a 120 Ohms +5%/-10% impedance placed between data+ and data-.

Female connectors are used on controllers and other transmitting devices. Male connectors are used on receiving devices. The required connector is a 5-pin XLR; pins are assigned as: common reference (1), data1- (2), data1+ (3), data2- (4), data2+ (5).

2.2 Data Protocol

DMX512 data is transmitted sequentially in asynchronous serial format, starting with slot 0 and ending with slot 512. Prior to the first data slot being transmitted, there is a reset sequence consisting of a BREAK followed by a MARK AFTER BREAK and a START code. Valid data values under a NULL START code are between 0 and 255.

(Figure 1 shows detailed information about the DMX512 frame’s compounding elements.)

The data transmission format is compounded as follows:

- Bit 1 start bit, low or space
- Bit 2-9 data bits (least significant bit to most significant bit)
- Bit 10,11 stop bits, high or MARK

The data frame is compounded by these time slots:

- BREAK

  Indicates the start of a new packet, typical value is 176 µsec.
- MARK AFTER BREAK

  Separates the break and start code time slots. Values can be between 8 µsec and 1 second.
- START CODE

  The first slot (slot 0) following a MARK AFTER BREAK. Identifies the function of subsequent data bytes in the packet. For dimming commands the star code value is 0x00; therefore, it is also known as a NULL START CODE. Alternate start codes may be used (refer to the USSIT-DMX512-A protocol specification, annex D).
- DATA SLOTS

  Subsequent data bytes where the dimming levels for each receiving device are placed. The number of data slots is 512; DMX512 data packets with fewer data packets should not be transmitted. Time between any two data packets may vary from 0 up to 1 sec. This time is known as MARK TIME BETWEEN DATA SLOTS.
- MARK BEFORE BREAK

  Time between the second stop bit of the last data slot of a given data packet and the falling edge of the beginning of the BREAK for the next data packet. This time may vary from 0 up to 1 sec. Every data packet transmitted over the data link starts with a BREAK, MARK AFTER BREAK, and a START CODE sequence defined as a RESET sequence.
- BREAK-TO-BREAK.

  The period between the falling edge at the start code of any one BREAK and the falling edge at the start of the next BREAK. May vary from 1196 µsec up to 1.25 sec.

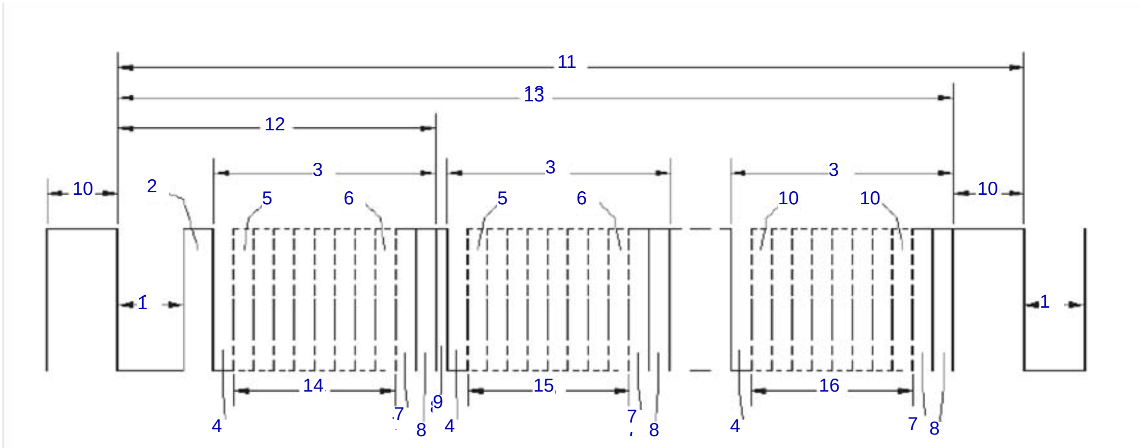

Figure 1. Typical DMX512 Frame

Each listed point describes the numbers shown in Figure 1

1. SPACE for BREAK
2. MARK AFTER BREAK
3. Slot Time
4. START Time
5. LEAST SIGNIFICANT data bit
6. MOST SIGNIFICANT data bit
7. STOP bit
8. STOP bit
9. MARK TIME BETWEEN SLOTS
10. MARK BEFORE BREAK
11. BREAK to BREAK time
12. RESET sequence
13. DMX512 packet
14. START CODE (Slot 0, data)
15. SLOT 1, data
16. SLOT nnn, data (Max 512)

3 Software Flowchart

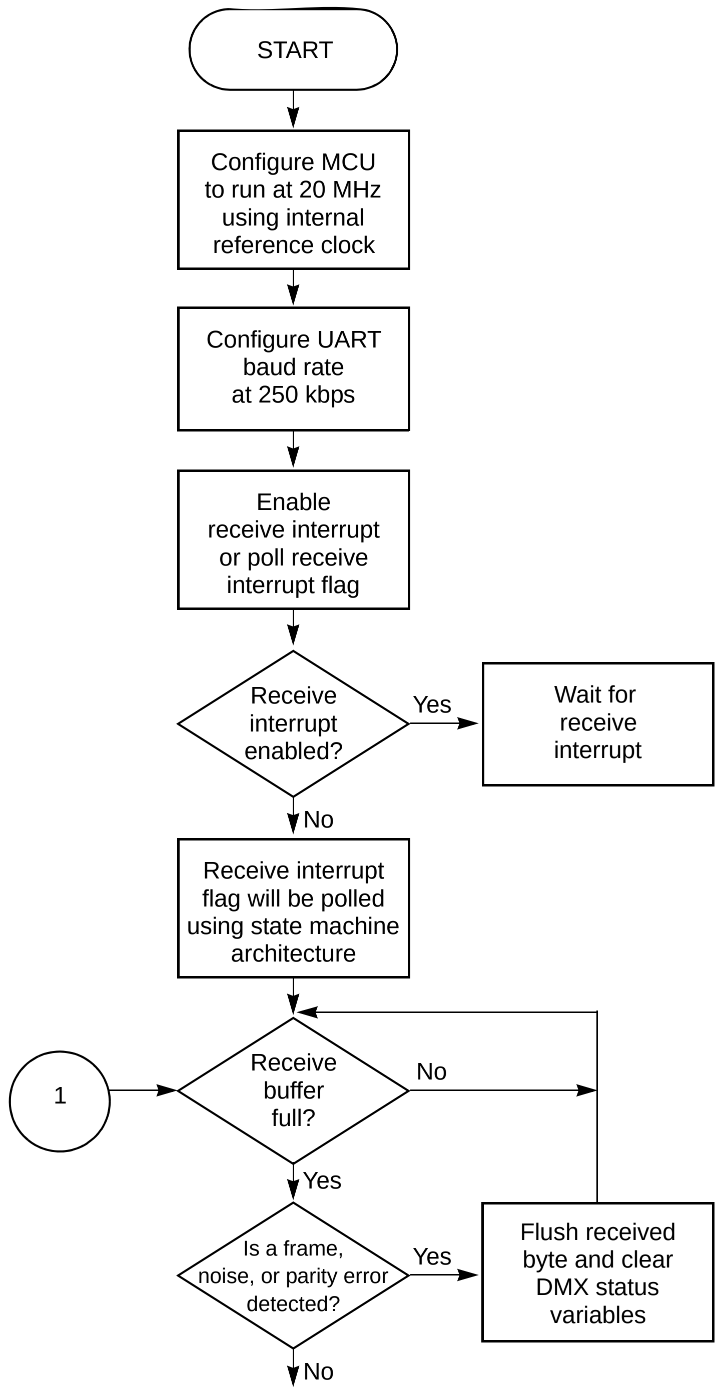

Figure 2. Flowchart

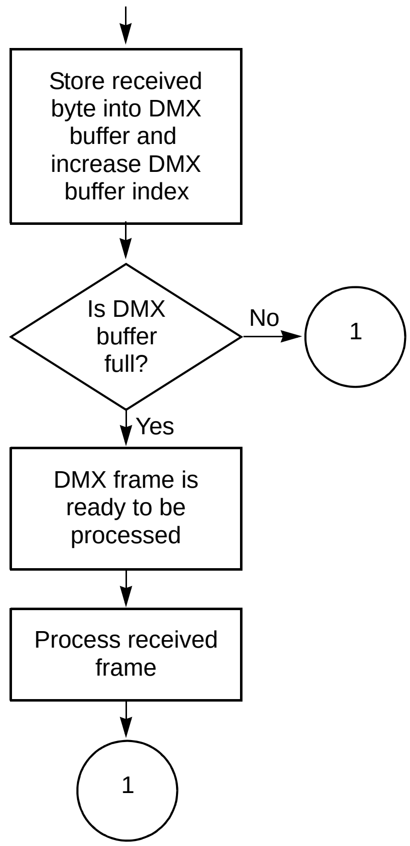

Figure 2. Flowchart (continued)

3.1 Software Implementation

Because the protocol consists of an untyped byte stream produced by standard UARTs, the implementation can be easily adapted to work with any microcontroller supporting baud rates up to 250 kbps.

The application software architecture developed in this application note is based on a State Machine model. The shown software was developed to demonstrate DMX512 protocol implementation. User has to develop the application layer which will control the lighting devices attached to the network.

3.2 Code Description

Required steps to implement the DMX512 driver using a generic UART:

1. MCUInit_Internal

   Configures microcontroller to work with internal reference, enables FLL to operate at 20MHz, and waits until FLL frequency becomes stable.

```
/* SOPT: COPE=0,COPT=1,STOPE=0,BKGDPE=1 */
  SOPT = 0x53;
  /* SPMSC1: LVDF=0,LVDACK=0,LVDIE=0,LVDRE=1,LVDSE=1,LVDE=1 */
  SPMSC1 = 0x1C;
  /* SPMSC2: LVWF=0,LVWACK=0,LVDV=0,LVWV=0,PPDF=0,PPDACK=0,PDC=0,PPDC=0 */
  SPMSC2 = 0x00;
```

```
  /* ICGC1: RANGE=1,REFS=0,CLKS1=0,CLKS0=1,OSCSTEN=1 */
  ICGC1 = 0x4C;
  /* ICGC2: LOLRE=0,MFD2=1,MFD1=1,MFD0=1,LOCRE=0,RFD2=0,RFD1=0,RFD0=0 */
  ICGC2 = 0x70;
  ICGTRM = *(unsigned char*far)0xFFBE; /* Initialize ICGTRM register from a non volatile
memory */
  while(!ICGS1_LOCK) {                  /* Wait FLL be stable */
```

2. vfnDMX512Init

   Configures SCI module to work with a baud rate of 250 kbps, enables RX interrupt, clears all RX flags, and initializes DMX control variables.

```
 /** Variable used to clear the SCI flags */
UINT8 uInt8ClearFlags = 0;

DMX512BDH = 0x00;
DMX512BDL = BAUD_RATE_250K; /* Configure Baud Rate at 250Kbps*/
DMX512C1 = DMX512_RESET_VALUE;

 /* Disable Rx interrupt. Data arrived will be polled */
DMX512C2 = CONFIG_DMX512_RX_POLL;
                                                                                     /*
Clear any Flag that could be set */
uInt8ClearFlags = DMX512S1;
uInt8ClearFlags = DMX512S2;
DMX512C3 = DMX512_RESET_VALUE;
uInt8ClearFlags = DMX512Data;  /* Reading the DMX512_RX Data Register */

/*Set the program flow control flags*/
DMX512Comm.RxOffset    = 0;
DMX512Comm.RxFlags     = 0;
DMX512Comm.StartRxFlag = 0 ;
DMX512Comm.RxStatus = DMX512_READY ;
```

3. vfnDMX512Driver:

   There are two methods to detect a valid DMX512 frame. The first method uses the SCI receiver interrupt. The second method polls the SCI receiver interrupt flag. (See sections 3.1 and 3.2 for code.)

3.3 Advantages and Disadvantages of Using Interrupt or Non-Interrupt Methods

Developer must choose which method better suits the application. Both alternatives are presented in the source code.

- Interrupt-Method Advantage:

  Byte reception is assured because MCU services the UART receiver interrupt routine every time a byte is received. Therefore, data reception is independent of program flow. This task is always serviced.
- Interrupt-Method Disadvantage:

Because the DMX512 console sends every time the DMX frame completes (512 data bytes + 1 command byte), the MCU will constantly service the SCI receiver interrupt, consuming most of the MCU processing time. The application tasks must be completed during the remaining time. Time to complete these tasks could be longer than using polling method. Developer must be aware the interrupt-service routine will have higher priority than program flow.

- Non-Interrupt Method Advantage:

  Program flow time could be prioritized in order to attend the incoming DMX frame and continue with normal program flow.
- Non-Interrupt Method Disadvantage:

  DMX512 frames could be lost if the DMX512 receiving task is not called on time during the normal program flow.

3.4 Interrupt and Non-Interrupt Based Code:

The frame-detection method is the same in both solutions. The main difference between methods resides in the state-machine model. In the non-interrupt method, the state machine is always polling the receiver interrupt flag (see code for more details). If non-interrupt method is not used then developer should change the following line code in the DMX initialization routine (vfnDMX512Init):

```
DMX512C2 = CONFIG_DMX512_RX_POLL;
```

If developer decides to use the interrupt-base method, the following line in the DMX initialization routine must be declared instead of the above code:

```
DMX512C2 = CONFIG_DMX512_RX_ISR_EN;
```

3.5 DMX512 Frame-Detection Code

The following code is used to detect the DMX512 data frames. Two local variables are declared, u8Status is used to capture and clear the SCI status register flags. The u8Dummy is a temporal variable where the data byte retrieved from the SCI Rx data register is stored. After declaring these variables, the next step is to check if the receiver buffer is full. That does not mean the received data is valid. To verify data, the framing, noise, and parity flags must be checked. If one flag is set, data is flushed and the DMX512 driver status flags are cleared. The driver will wait for the next DMX512 frame to sync.

When a valid data is received, it is stored in the DMX512 buffer until the 1 command + 512 data bytes have been received. If buffer is full, the DMX512 driver tells the main program a complete DMX512 frame has been received. It will set the RxStatus flag to DMX512_RX_PROC_MSG state and initialize the DMX512 buffer index.

The RxStatus and the StartRx flags must be cleared in the main program flow once the DMX512 data buffer has been read, then the buffer can be used again. See AN3315SW software files for details.

```
/** Variable used to capture the SCI status register */
 UINT8 u8Status = 0;
 /** Variable used to store the Data register content */
 UINT8 u8Dummy  = 0;

   u8Status = DMX512S1;
```

```
    u8Dummy  = DMX512Data;

    if((u8Status & 0x20) !=0)      /* Rx data buffer is full ?*/
    {

      if ((u8Status & 0x07) != 0)  /* Check for Errors (Framing, Noise, Parity) */
        {
          u8Dummy = DMX512Data;
          DMX512Comm.RxOffset = 0;
          DMX512Comm.StartRxFlag = 0;
          DMX512Comm.RxStatus = DMX512_RX_DATA;
          return;
        }

          DMX512Comm.RxBuffer1[DMX512Comm.RxOffset++] = u8Dummy ;

              if( DMX512Comm.RxOffset >= DMX_RX_BUFFER_SIZE )
              {
                DMX512Comm.RxOffset = 0;
                DMX512Comm.StartRxFlag = 1;
                DMX512Comm.RxStatus = DMX512_RX_PROC_MSG;
              }
   }
```

4 DMX512 Implementation Constrains

To implement the DMX512 protocol in an 8-bit microcontroller, it is important to keep in mind the following considerations:

- MCU UART must support a baud rate of 250 kbps.
- MCU UART must support a data frame compounded by 1 start bit, 8 data bits, 1 stop bit in a NRZ format.
- 513 bytes in RAM are needed to store the complete DMX512 frame. MCU must have at least 1Kb of RAM.

5 DMX512 Hardware Implementation

Figure 3 shows the recommended hardware that should be implemented using the referenced code in this document. Listed below are considerations to keep in mind when the proposed HW becomes implemented:

- Jumper JP9 must be set in case the device becomes the last in the DMX bus.
- Pins 4 & 5 in the male connector are not used.
- The developer is allowed to Enable/Disable the RE signal in the transceiver by software or hardware. The DMX512 driver in this document does not control the RE signal.

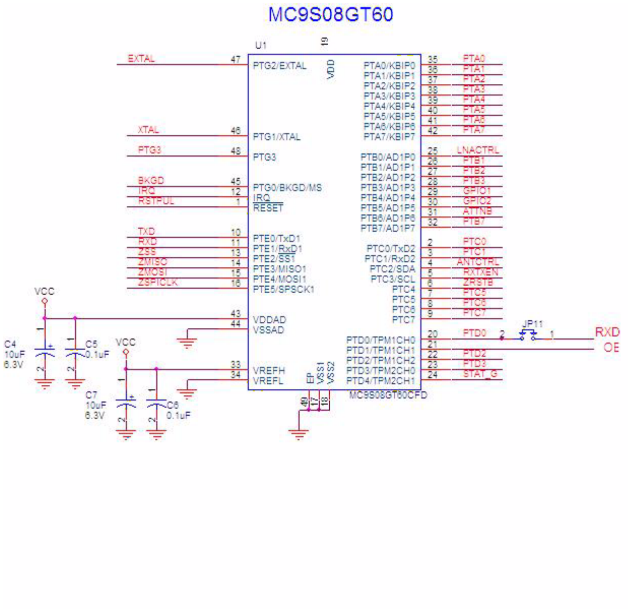

Figure 3. Proposed Hardware Implementation

6 Considerations

This example code was developed using the CodeWarrior IDE version 5.0 for HC(S)08 and tested on the MC9S08GT60 device. Interrupt vectors must be modified to fit the specific MCU vector table located in the vectors and interrupts section of the data sheet for each MCU. The complete software driver was tested using the proposed hardware schematic shown in this document.

7 Conclusion

A DMX512 driver has been presented in this document. The described software and hardware fully comply with the ANSI/TIA/EIA-485-A-1998 standard. Today, the DMX512 standard has been adopted as the main lighting protocol in applications where multiple receiving points are needed. DMX512 is used not only in the entertainment market but also in home, industrial, and appliance lighting markets.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3315
Rev. 0
09/2006

freescale™
semiconductor