Implementing an Integrated DMX512 Receiver Using the EZ-Color HB-LED Controllers

AN47023

Author: Mukund Krishna

Associated Project: Yes
Associated Part Family: CY8CLED16, CY8CLED08, CY8CLED04
GET FREE SAMPLES HERE
Software Version: PSoC Designer™ 4.4
Associated Application Notes: None

Application Note Abstract

This application note introduces the DMX512 communication standard. It describes the implementation of a DMX512 receiver using Cypress' EZ-Color™ family of devices. The same device can also control the LED based light fixtures. This application note includes example code and an example project for PSoC Designer™.

Introduction

DMX512 is derived from **D**igital **M**ultiple**X** with **512** pieces of information. It is maintained by the Entertainment Services and Technology Association (ESTA) and is officially known as E1.11-2004 USITT DMX512-A. DMX512 is developed as a non-command based protocol to enable the interoperability of systems made by multiple manufacturers.

DMX512 describes the digital data transmission between controllers and lighting equipment and other accessories. It is a serial, digital, packet based protocol that specifies the 'daisy-chain' method of connecting multiple slaves to a DMX512 host, with a maximum of 512 slaves on one bus. The transmission speed is 250 Kbps and the electrical specifications are governed by the RS485 standard (differential signaling) with 5-pin XLR as the interface connectors.

This document provides an overview of the communication protocol defined by DMX512. It also includes the detailed procedure to implement a DMX512 receiver and LED controller using Cypress' EZ-Color family of devices (CY8CLEDxx) and the CY3261A RGB evaluation kit.

Figure 1. CY3261A EZ-Color RGB Evaluation Kit

DMX512: The Standard

The standard covers electrical characteristics, data format, data protocol, and connector type. The most important aspects pertaining to the implementation of the receiver are the data protocol and format.

The data is transmitted in discrete packets, with 513 slots of data in each packet. Figure 2 shows the format of a packet in the form of a timing diagram. The first slot or byte contains the start code and the remaining 512 bytes contain dimming data for the slaves connected. The start code is the first byte of data in the packet and informs the receiver the following details of the data in the packet (it is NULL for dimming data). The standard specifies a maximum of 512 slots in each packet.

- The idle state of the bus is 'high', called 'mark'.
- The packet starts with a period of low, called 'break', followed by a high (mark).
- The actual byte then starts with LSB, followed by two stop bits.
- The slot ends with a mark (high) before the next slot starts.

The bit rate is 250 Kbps and the refresh rate for the packet (with 512 slots) is typically 44 Hz. From the receiver's perspective, the time allotted for a break before a packet is between 88 and 176 uS.

The standard states the electrical specification to be followed according to the EIA-485 standard (differential signaling) and the use of 5-pin XLR connectors as interfaces between the physical layer devices and the cables.

Cypress' DMX512 Receiver solution supports the latest version of the standard released in 2004.

DMX512: The System

The typical DMX512 system comprises a DMX host that controls up to 512 receivers. The DMX512 standard specifies unidirectional flow of data to the controlled element (the light fixture), to control its intensity. However, the latest version of the standard does allow optional implementation of Enhanced Function topologies using alternate start codes (non-NULL), which allow bi-directional data.

The host has a DMX512 transmitter that assembles the packet and transmits it over the bus, as shown in the packet structure in Figure 2. The host also includes an EIA-485 (RS485) PHY that interfaces to the actual wires.

Figure 3 represents a typical DMX512 system.

- The multiple receivers are connected to the DMX host in a daisy-chain manner and every packet goes through every receiver in entirety.
- At each receiver, the differential signal is received by a PHY and given to the receive side controller.
- Each receiver is programmed with a specific slot address so it knows which slot it has to extract from each packet.

The receiver can also extract multiple slots from every packet and thereby control more than one light fixture or more than one attribute of a light fixture.

Figure 2. DMX512 Packet Breakdown
(Source E1.11-USITT DMX512-A Standard)

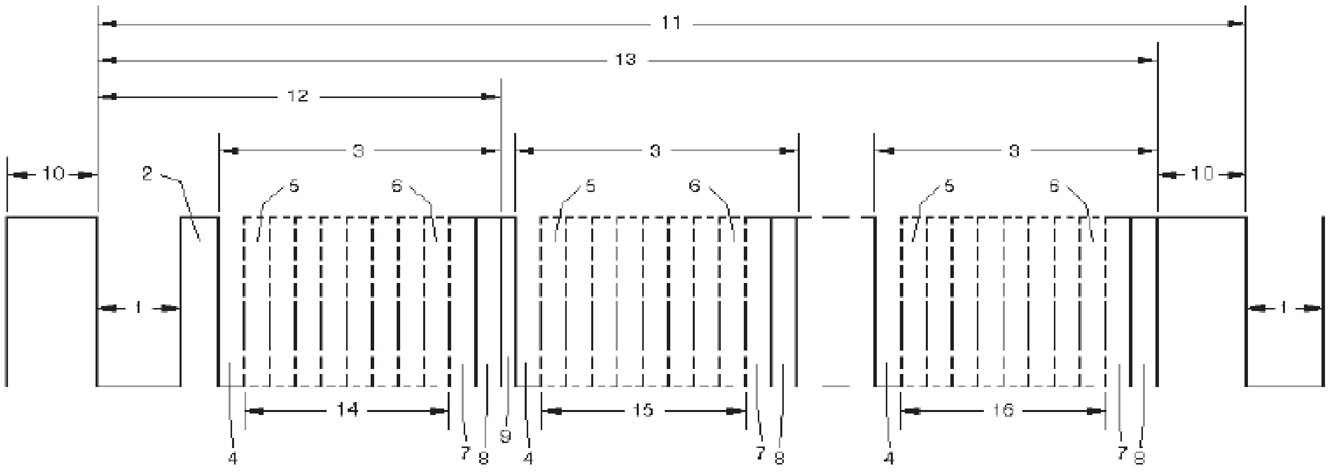
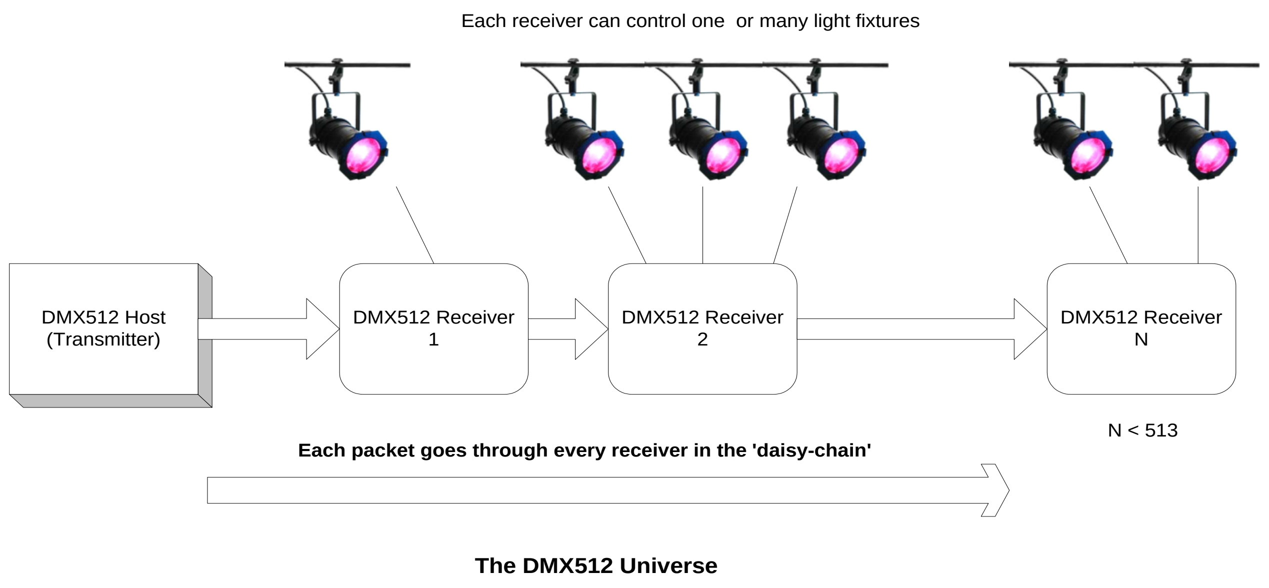


Figure Key
1 - "SPACE" for BREAK
2 - "MARK" After BREAK (MAB)
3 - Slot Time
4 - START Time
5 - LEAST SIGNIFICANT Data Bit
6 - MOST SIGNIFICANT Data Bit
7 - STOP Bit
8 - STOP Bit
9 - "MARK" Time Between Slots
10 - "MARK" Before BREAK (MBB)
11 - BREAK to BREAK Time
12 - RESET Sequence (BREAK, MAB, START Code)
13 - DMX512 Packet
14 - START CCODE (Slot 0 Data)
15 - SLOT 1 DATA
16 - SLOT nnn DATA (Maximum 512)

Figure 3. Representation of a Typical DMX512 System


Each receiver can control one or many light fixtures
DMX512 Host (Transmitter)
DMX512 Receiver 1
DMX512 Receiver 2
DMX512 Receiver N
N < 513
Each packet goes through every receiver in the 'daisy-chain'
The DMX512 Universe

EZ-Color Solution for DMX512: An Overview

The EZ-Color solution for a DMX512 system is the implementation of the receiver that controls the light fixtures. The EZ-Color device provides the following functionalities:

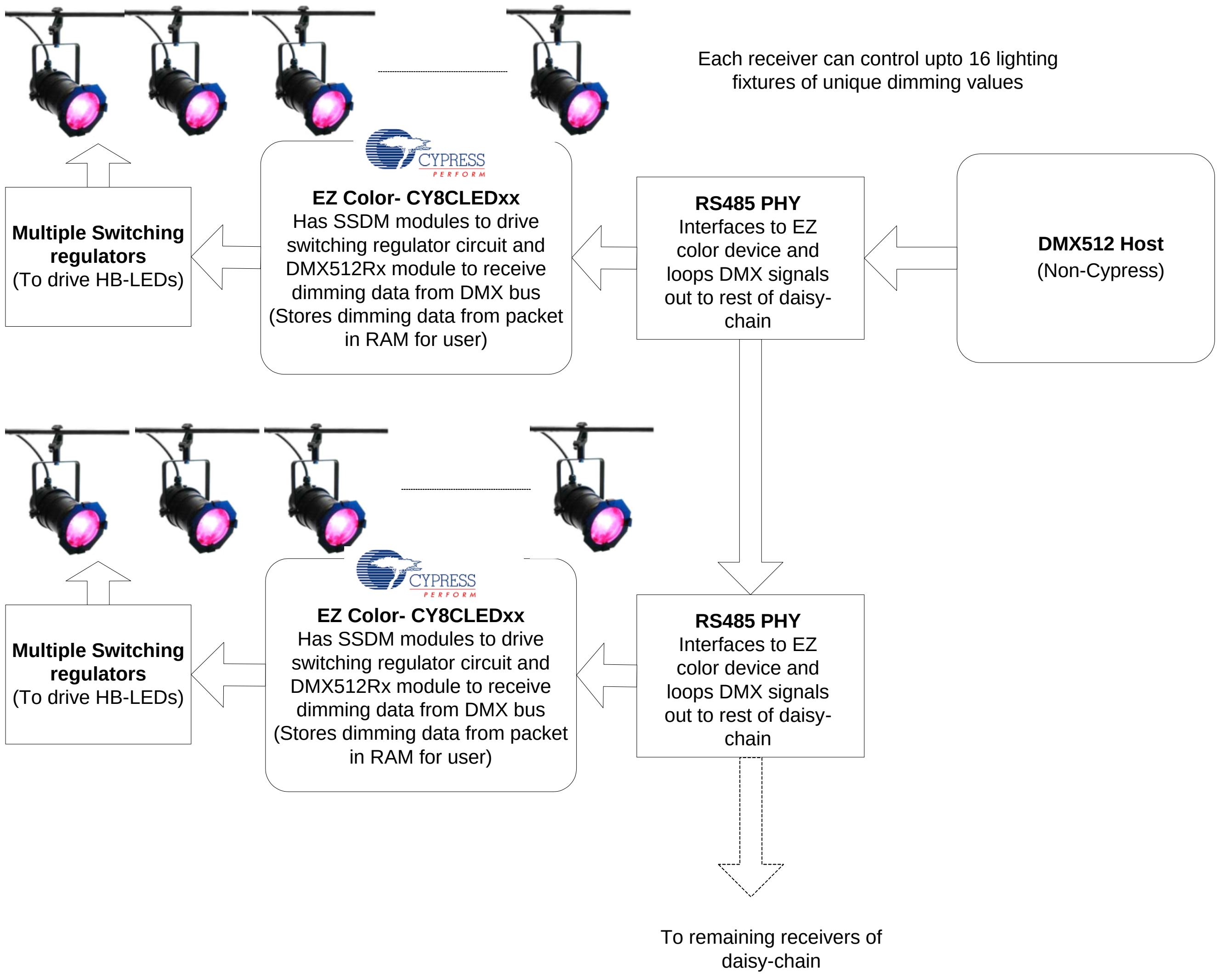
- Dynamically sets its address during run time.
- Extracts the required data (possibly multiple slots or bytes) from every received packet and stores it in a buffer.
- Dims the HB-LED based light fixtures via a switching regulator (either IC based or EZ-Color based) according to the dimming data received.
- If data is in the form of a color coordinate, processes the data using a color mixing algorithm (on the M8C) and drives 3 or 4 channels.

Therefore, the EZ-Color device (CY8CLEDxx) forms the receiver and controller in a DMX512 based lighting communication system.

Figure 4 is a representation of the system level solution using the EZ-Color controllers as DMX512 receivers.

- The DMX512 host sends signals based on the RS485 signaling standard to its universe.
- The signals are first received by the RS485 Physical Layer device that has two functions:
  - Convert the RS485 voltage levels to TTL voltage levels compatible with EZ-Color.
  - Retransmit the incoming signals to enable daisy-chaining more receivers.
- The EZ-Color device receives the packet (through the RS485 PHY) via a GPIO port pin and according to the address programmed.
- The address of the EZ-Color based receiver can be dynamically changed using a simple button interface instead of 'dip' switches. This is shown in the firmware project accompanying this application note.
- The receiver then extracts the particular sequence of slots from each packet. These are stored in a buffer memory.
- The dimming values stored in the buffer are passed onto the EZ-Color SSDM modules to vary the signal densities of their outputs. Alternatively, if the data is in the form of a color coordinate, it is passed onto a color mixing function in firmware.
- The SSDM modules are based on Cypress' PrISM™ (Precise Intensity Signal Modulation) technology that is similar to the PWMs with added benefits such as reduction in EMI and low frequency flicker.
- The outputs of the SSDM modules are routed out of the device through GPIO port pins and given to buck regulators that drive the high brightness LEDs.
- Each SSDM output is considered a channel. Therefore, a string of LEDs can be connected to one channel and they are dimmed identically.
- An EZ-Color receiver can control multiple channels of LEDs simultaneously because it can extract multiple slots of data from every packet.

Figure 4. EZ-Color System Solution


Each receiver can control upto 16 lighting fixtures of unique dimming values
Multiple Switching regulators (To drive HB-LEDs)
EZ Color- CY8CLEDxx Has SSDM modules to drive switching regulator circuit and DMX512Rx module to receive dimming data from DMX bus (Stores dimming data from packet in RAM for user)
RS485 PHY Interfaces to EZ color device and loops DMX signals out to rest of daisy-chain
DMX512 Host (Non-Cypress)
Multiple Switching regulators (To drive HB-LEDs)
EZ Color- CY8CLEDxx Has SSDM modules to drive switching regulator circuit and DMX512Rx module to receive dimming data from DMX bus (Stores dimming data from packet in RAM for user)
RS485 PHY Interfaces to EZ color device and loops DMX signals out to rest of daisy-chain
To remaining receivers of daisy-chain

Accompanying Hardware and Software

This implementation can also be carried out on the CY3261A EZ-Color evaluation kit available from the Cypress website at www.cypress.com.

This implementation requires the PSoC Designer software tool. The latest versions of this software tool and service pack are available for free download at www.cypress.com.

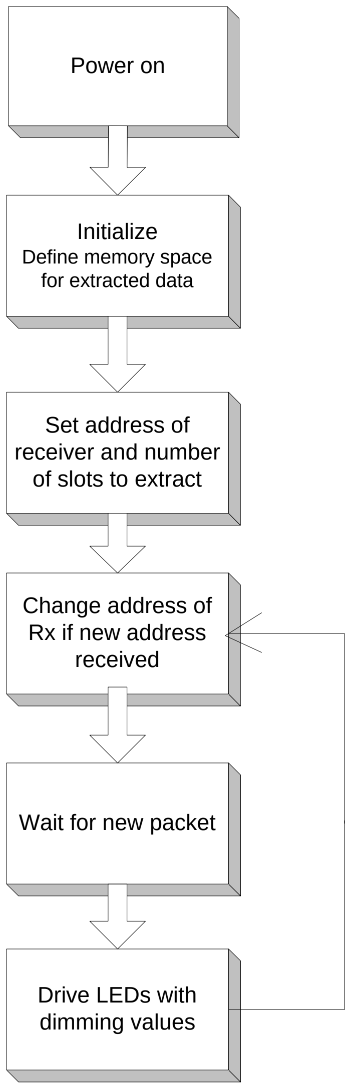
Firmware: High Level Overview

The overview of the firmware at a higher level of abstraction is shown in Figure 5. To first understand the firmware at this level, it is assumed that the DMX512 Rx user module is correctly initialized. The procedure to configure it is described in a later section of this application note.

The chart in Figure 5 indicates the simplicity of the firmware required to use the EZ-Color device as a DMX512 slave.

- Initialization refers to defining the memory space required by the user module to store the extracted slots of data and informing it of the location in the memory.
- The memory size also defines the number of slots the module extracts from every packet.
- The address of the receiver (between 1 and 512) is then defined.
- In a continuously running loop:
  - The module changes its address dynamically if requested by the user.
  - The module waits till a packet is received.
  - When it is received, the slots that are stored in the previously defined memory space are used to drive the different channels of LEDs to dim them appropriately.

Figure 5. Flow of Firmware


Power on
Initialize
Define memory space for extracted data
Set address of receiver and number of slots to extract
Change address of Rx if new address received
Wait for new packet
Drive LEDs with dimming values

Implementation with PSoC Designer

PSoC Designer is a development tool from Cypress (available free for download from www.cypress.com) that enables users to configure, customize, and efficiently manage the programmable resources in the EZ-Color device family. Along with PSoC Programmer, the development environment allows users to write related firmware, compile, and build projects that are then programmed into the EZ-Color devices.

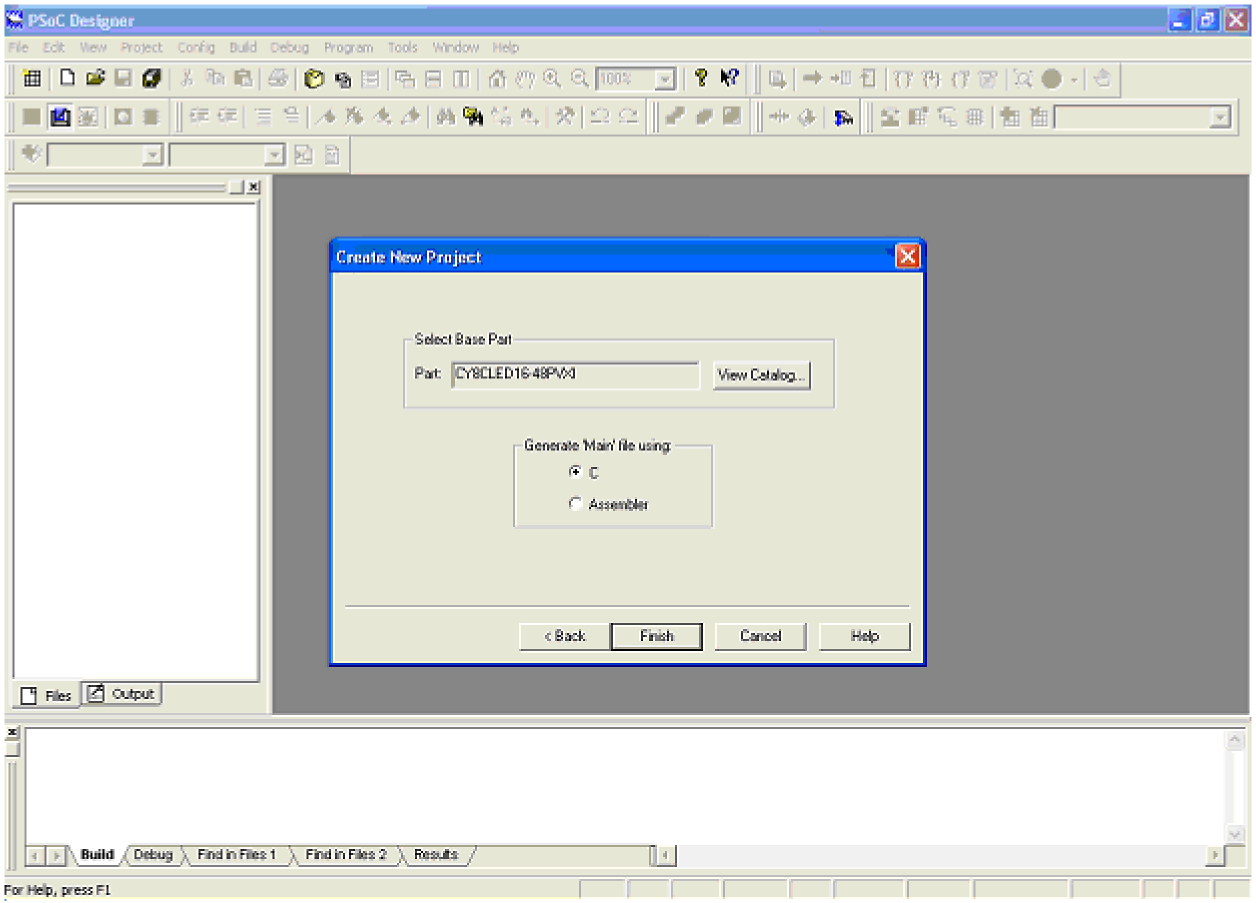
The following are the steps to implement a simple DMX512 receiver to dim three LEDs. A completed, functional example project is attached as a zip file with this application note. The project design is done such that the receiver's implementation is tested using the CY3261A RGB board. If a different development board or a custom-designed board with an EZ-Color device is used, the pin connections must be set up appropriately.

1. Open PSoC Designer and start a new project.
2. In the option to choose the device (after entering the project name), select CY8CLED16 or CY8CLED08. The CY8CLED04 can also be used if the number of LED channels to be driven is two or less.
3. Select 'C' in the 'Generate 'Main' file using:' option and press Finish.
4. The user module selection view is displayed in the device editor. From the categories on the left, select the Digital Comm tab.
5. Clicking on the DMX512Rx icon shows the user module data sheet in the window on the right. The data sheet covers functional descriptions, electrical specifications, parameters, APIs, and registers used by the module.
6. Change to interconnect view.

   Note To get familiar with using PSoC Designer, it is recommended to take the online course at http://www.cypress.com/training/
7. As shown in Figure 8, from the Digital Comm tab, select the DMX512Rx and click on the yellow '+' button to select the module. Set the CPU clock to Sys_Clk/1
8. Place the user module. It occupies one basic block (DBBxx) and one communication block (DCBxx). The communication block is the receiver part of the module while the basic block is responsible for detecting the start of a packet.
9. Set VC1 and VC2 to 12 and 16 respectively in the global resources tab. This is to accommodate the frequency requirements for the two blocks of the user module. The communication block must have an input frequency of 2 MHz and the basic block requires anywhere between 91 and 166 KHz.
10. Set VC3 divider to 12 in the Global Resources tab.
11. Configure the user module as shown in Figure 7.

    Note that the address of the receiver is set to '1'. This can be reset to the desired address or dynamically changed in firmware.
12. Right click the user module in the selected user modules window and rename it. In this project, it is named DMX512_Rx.

Figure 6. Starting the Project in PSoC Designer



Part: CY8CLED16-48PVXI

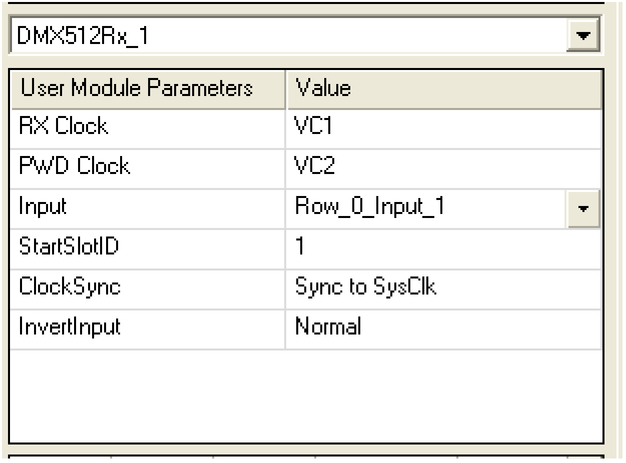
Generate 'Main' file using: [✓ C] Assembler

13. Using the same procedure, instantiate three 8-bit SSDM modules from the LED Dimming category and place them. They will each occupy a digital block. It is not important what kind of block (basic or communication) they occupy.

14. Rename them to SSDM_RED, SSDM_GREEN, and SSDM_BLUE to denote the three channels.

15. Configure the three SSDM modules as shown in Figure 9. These modules function based on Cypress' PrISM technology that reduces EMI and low frequency flicker.

Figure 7. Configuration of DMX512Rx User Module

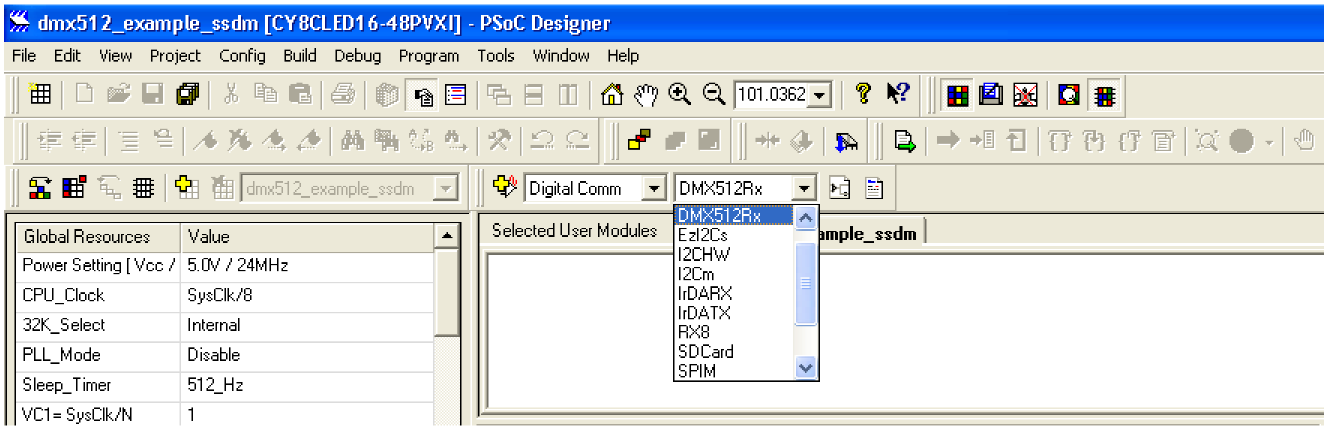


DMX512Rx_1

User Module Parameters	Value
RX Clock	VC1
PWD Clock	VC2
Input	Row_0_Input_1
StartSlotID	1
ClockSync	Sync to SysClk
InvertInput	Normal

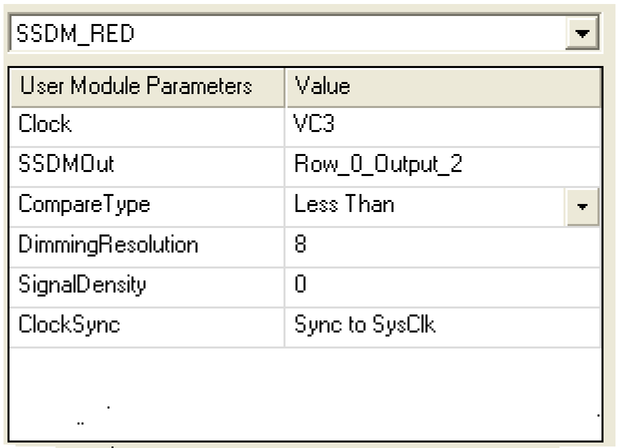
[+] Feedback

Figure 8. Choose the DMX512Rx User Module



Global Resources	Value
Power Setting [Vcc /	5.0V / 24MHz
CPU_Clock	SysClk/8
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	1

Figure 9. SSDM User Module Configuration



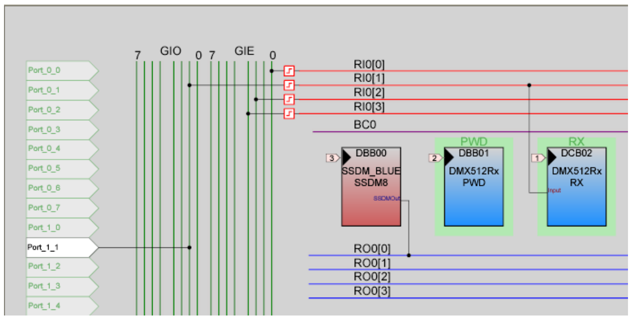
SSDM_RED

User Module Parameters	Value
Clock	VC3
SSDMOut	Row_0_Output_2
CompareType	Less Than
DimmingResolution	8
SignalDensity	0
ClockSync	Sync to SysClk

16. Direct the 'SSDMOut' of SSDM_RED, SSDM_GREEN, and SSDM_BLUE to Row0Output2, Row1Output3, and Row0Output0 respectively. All other parameter configurations of the three modules remain identical.

17. Connect port pin P1[1] to the input of the DMX512Rx module's input, Row0Input1 (Figure 10).

Figure 10. Connection of Input to DMX512 Receiver



18. Similarly, connect the SSDM_RED module's output to port pin P1[2], SSDM_GREEN module's output to pin P1[3], and SSDM_BLUE module's output to pin P1[4].

19. This project is also implemented to enable daisy-chaining more receivers using the interface board shown in Figure 13. Therefore, pin 1[0] should be set to a High-Z driver mode, so that connecting the RX and TX of the RS485 transceiver chip directly loops the input back to the output.

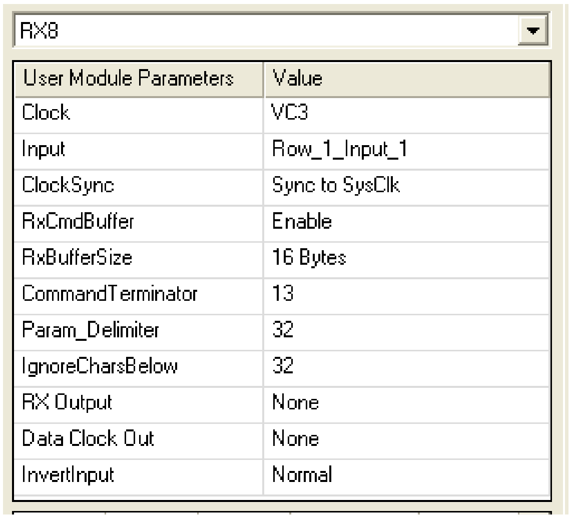
    If this is implemented using a different board, and if the DMX transmit signals are not coming from within the chip and the incoming DMX signals need to be looped back, ensure that the pin connected to the TX pin is at High-Z drive mode.

20. To implement the functionality of dynamic addressing of the receiver for the CY3261A kit, a communication interface is set up between the USB chip and the CY8CLED16 chip on the board.

21. Using the procedure detailed earlier, instantiate a RX8 user module from the Digital Comm tab. Place it in a digital communication block such as DCB12.

22. Configure the RX8 module as shown in Figure 11.

Figure 11. Configuration of RX8 User Module



RX8

User Module Parameters	Value
Clock	VC3
Input	Row_1_Input_1
ClockSync	Sync to SysClk
RxCmdBuffer	Enable
RxBufferSize	16 Bytes
CommandTerminator	13
Param_Delimiter	32
IgnoreCharsBelow	32
RX Output	None
Data Clock Out	None
InvertInput	Normal

23. Route pin P1[5] to Row_1_Input_1 to connected it to the input of the RX8 module.

    Note If the dynamically changeable address functionality is not desired, ignore steps 20 to 23.

24. The basic configuration required to make the DMX512 receiver work and control the three LED channels is now complete. Select the 'Generate Application' option from the Config menu.

25. Change to the application editor view using the View menu and open *main.c* from the source files tab.

    The following is an example of the code that can be written in *main.c* to implement a receiver controlling a 3-channel light fixture.

```
#define DMX_RAM_BUF_SIZE 3
BYTE DMX_RAM_BUF[DMX_RAM_BUF_SIZE];
```

    The first line determines the number of slots to be extracted and the second line defines the buffer in memory that holds the slots.

```
void main()
{
  M8C_EnableGInt    (a)

DMX512Rx_SetRamBuffer(DMX_RAM_BUF_SIZE,
&DMX_RAM_BUF[0]);     (b)
  DMX512Rx_Start();         (c)
  SSDM_BLUE_Start();
  SSDM_RED_Start();
    SSDM_GREEN_Start();

  RX8_EnableInt();             (d)

  RX8_Start(RX8_PARITY_NONE);

  DMX512Rx_EnableInt();     (e)
  DMX512Rx_SetStartSlotID(1);   (f)
```

    Inside the main function:

    - The global interrupts are enabled.
    - The DMX user module is informed of the memory buffer's location.
    - The DMX512Rx and the three SSDM modules are started.
    - The UART receiver's interrupts are enabled and the module is started.
    - The DMX512 user module's interrupts are enabled.
    - If necessary, the starting address of the receiver can be redefined.

26. Now the initialization is complete and the remaining code periodically

    - Checks if a new address is set and changes the receiver's address accordingly.
    - Waits for the packet to arrive.
    - Writes the dimming values stored in the memory buffer to the SSDM modules as their signal densities.

```
while(1)
  {
   if (bAddrecd)
   {
   bAddrecd = 0;
   wDMXaddress = bRxBuf[0] + bRxBuf[1];

   DMX512Rx_SetStartSlotID(wDMXaddress)
;
   }


    while(DMX512Rx_bGetSlotActivity());
// Wait for required slot received
PWM_RED_WritePulseWidth(DMX_RAM_BUF[0])
;  //Update the PWM pulse widths
PWM_GREEN_WritePulseWidth(DMX_RAM_BUF[1
]);

PWM_BLUE_WritePulseWidth(DMX_RAM_BUF[2]
);
}
```

    Note On the CY3261A RGB board, the two buttons are connected to the enCoRe™ USB chip. Therefore, button presses must be communicated from the USB chip to the CY8CLED16 chip. In this project, a UART communication interface is set up to implement this.

27. To implement the RX8 receiver, the following code should be inserted in the interrupt service routine of the module. The file is *RX8int.asm*.

```
push A
   push X
   lcall RX8_bReadRxData
Append:
   mov X, [_bByteCount]
   mov [X+_bRxBuf], A
   inc [_bByteCount]
   cmp [_bByteCount], 2
   jnz Skip
   mov [_bByteCount], 0
   mov [_bAddrecd], 1
Skip:  pop X
   pop A
```

    This assembly code must be inserted in the file only between the banners that specify the insertion of user code. Placement of this code anywhere else in the file can cause unpredictable behavior. This interrupt service routine is executed every time the UART receiver receives a set of bytes from the USB chip.

    The code basically carries out the functionality of storing the received information (address) into a specific 2-byte buffer and sets a flag to indicate a new address.

28. The project is now ready to be built and programmed into the device. Select the ‘Build’ option from the Build menu. This compiles all the files of the project and generates a hex file if the compiler does not find any errors.

    Note To build this project, a compiler must be installed for PSoC Designer. A free version and Pro version of the compiler is available at www.cypress.com/downloads.

When the project is successfully built, it can be programmed into the EZ-Color device using the ‘MiniProg’ via USB. The MiniProg is connected to the ISSP header (of the EZ-Color device) on the CY3261A RGB board, if this board is used for testing this design.

Selecting the ‘Program Part’ option in the Program menu carries out the programming task. This calls the PSoC Programmer software. If the MiniProg is not detected, select the correct port to which it is connected in the PSoC Programmer software and click on Connect.

When the MiniProg is detected and there is a ’connected’ message at the bottom right, clicking on Program initiates the programming process.

The procedure to set up the button functionality, LCD display, and a UART transmitter on the USB chip is described in Appendix B of this application note. The completed PSoC Designer project for the CY7C64215 chip also accompanies this application note.

Hardware Interface

Any off the shelf DMX512 controllers can be used to test this project. This section highlights the important details of the test hardware being developed and used for a DMX512 Receiver using EZ-Color.

- The controller transmits signals using the RS485 signaling protocol. Therefore, an interface circuit is required between the controller and the EZ-Color receiver comprising an RS485 PHY. It translates the RS485 differential signals to 5V TTL signals as depicted in Figure 3. The data out on the TTL side of the PHY is connected to the DMX512 receiver’s input via the port pin P1[1] (or any other input pin).

- Often, the controller transmits at the 5V level because it is within the voltage range of the RS485 protocol. In this case, only for testing purposes, the positive differential line can be connected directly to port pin P1[1] (or any other input pin). The interface circuit is also required for ESD and over-voltage protection and must be present for a complete implementation.

- On the CY3261A, P1[1] is accessed via pin 4 of the ISSP header of the EZ-Color device.

- The ground of the DMX controller and the EZ-Color device must also be common. On the CY3261A RGB board, it is pin 2 on the ISSP header of the EZ-Color device.

- Power supply must be given to the LED driver circuits and the EZ-Color device. On the CY3261A RGB board, this is done by supplying 12V DC via the power connector on board.

- Pins P1[5] and P1[7] of both chips on board the kit are already connected to each other to enable either I^2C™ or UART communication interfaces.

The system is now fully setup and ready. Changing the dimming values of the particular slots using the DMX controller causes the respective LEDs to change brightness. On the CY3261A board, this can be seen in changes in brightness of the red, green, and blue LEDs.

As implemented, pressing the ‘Change Color’ button increases the receiver’s address from 1 all the way to 512 before looping back to 1. Pressing the ‘On/Off’ button decreases the address and loops back to 512 from 1.

Summary

This application note provides an overview of the DMX512 communication protocol and details the method of implementing a DMX512 Receiver that also controls the LED based light fixtures. To get more details of the features of EZ-Color and how to use them, read the following application notes: AN16035—“Firmware RGB Color Mixing Firmware for EZ-Color”, AN33640—“Color Mixing Accuracy with EZ-Color High Brightness LED Controllers”, AN15733—“Power Management – RGB Color Mixing Hardware for EZ-Color Controllers”, AN14406—“Temperature Compensation for High Brightness LEDs Using EZ-Color and PSoC Express”, AN44533—“ColorLock Optical Feedback for EZ-Color”. New application notes are constantly added to the database on http://www.cypress.com/.

The projects associated with this application note contain code with detailed comments. To see this project in action with hardware, the CY3261A RGB kit may be purchased.

Figure 12 and Figure 13 shows the CY3261A RGB kit and an off-the-shelf DMX host controller used to implement the DMX512 system. The DMX controller has a 3-pin XLR output which is connected to the RGB kit’s inputs via the ISSP header.

Figure 14 shows a schematic of possible interface circuit that can be used between the controller and the CY3261 kit with XLR connectors on it to facilitate easy connections.

Figure 12. DMX512 System Implemented Using a Host Controller and the CY3261A Evaluation Kits

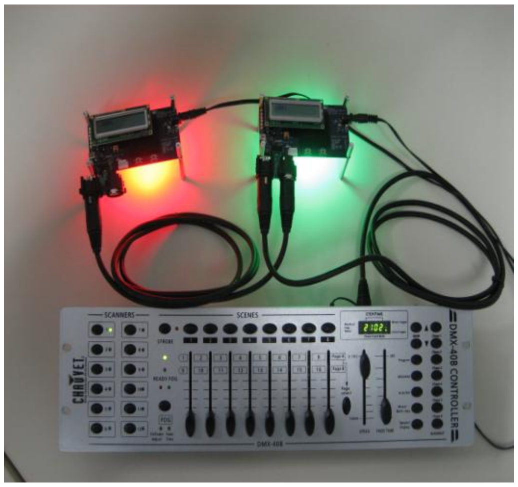


Figure 13. CY3261A RGB Evaluation Kit with RS485 Interface Board

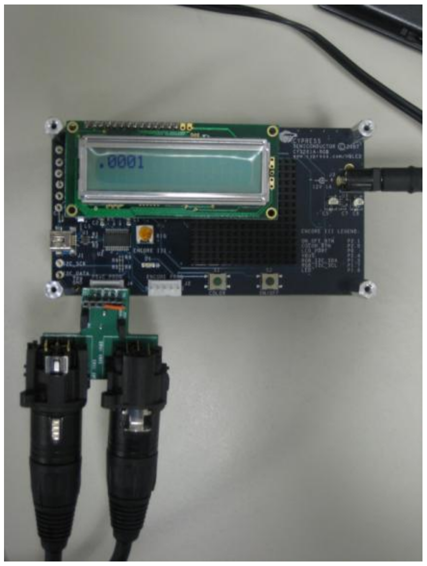
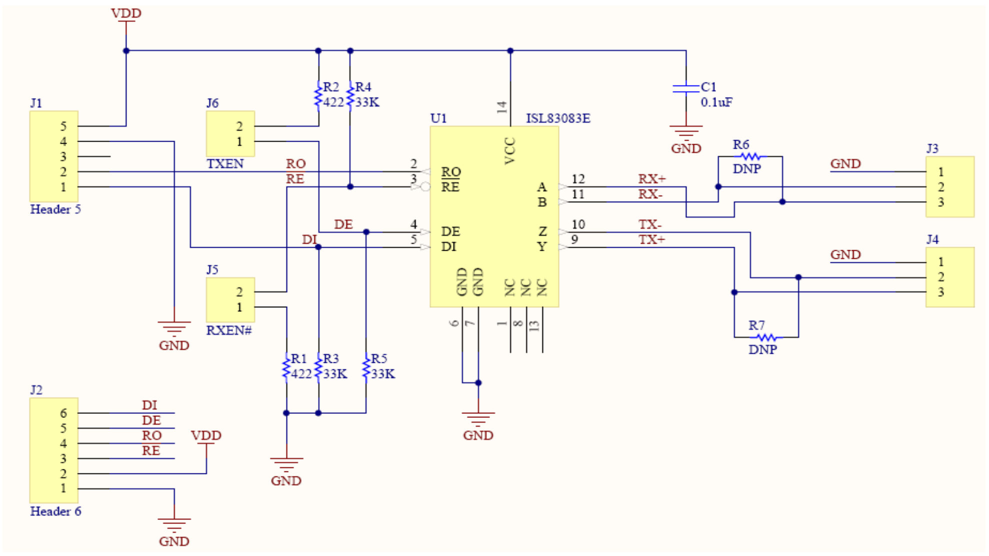


Figure 14. Schematic of Interface Circuit between DMX Controller and EZ-Color Device



Appendix A: Code for CY8CLED16

File: *main.c*

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

#define DMX_RAM_BUF_SIZE 3  //defines how many slots needed to stored
BYTE DMX_RAM_BUF[DMX_RAM_BUF_SIZE];  //RAM buffer that holds slot data starting from
                                        //the slot address
BYTE bRxBuf[2] = {1,0};                 //buffer for I2C slave to receive address info
BYTE i, bByteCount = 0, bAddrecd = 0;
WORD wDMXaddress; //Variable that holds the receiver's current address
void main()
{
  M8C_EnableGInt;              // Enable global interrupts
  DMX512Rx_SetRamBuffer(DMX_RAM_BUF_SIZE, &DMX_RAM_BUF[0]);  //Set buffer

  DMX512Rx_Start();            // Turn on DMX512Rx User Module

    SSDM_BLUE_Start();                      //Turn on the SSDMs to drive LEDs
  SSDM_RED_Start();
  SSDM_GREEN_Start();

  RX8_EnableInt();                          //Turn on the RX8 receiver's interrupts
  RX8_Start(RX8_PARITY_NONE);       //Turn on the RX8 receiver with no parity check

  DMX512Rx_EnableInt();        // Enable DMX512Rx interrupts

  while(1)
  {
   if (bAddrecd)    //if a new address has been received, this flag will have been
   {         //set by the RX8's interrupt service routine
   bAddrecd = 0;           //Reset the flag
   wDMXaddress = bRxBuf[0] + bRxBuf[1]; //combine the 8-bit buffer values to form 16-
bit address
   DMX512Rx_SetStartSlotID(wDMXaddress);      //Configure the address of the DMX512 receiver
   }
    while(DMX512Rx_bGetSlotActivity());  // Wait for required slot received
    SSDM_RED_WriteSignalDensity(DMX_RAM_BUF[0]);     //Update the SSDM pulse widths
    SSDM_GREEN_WriteSignalDensity(DMX_RAM_BUF[1]);
    SSDM_BLUE_WriteSignalDensity(DMX_RAM_BUF[2]);
      }
}
```

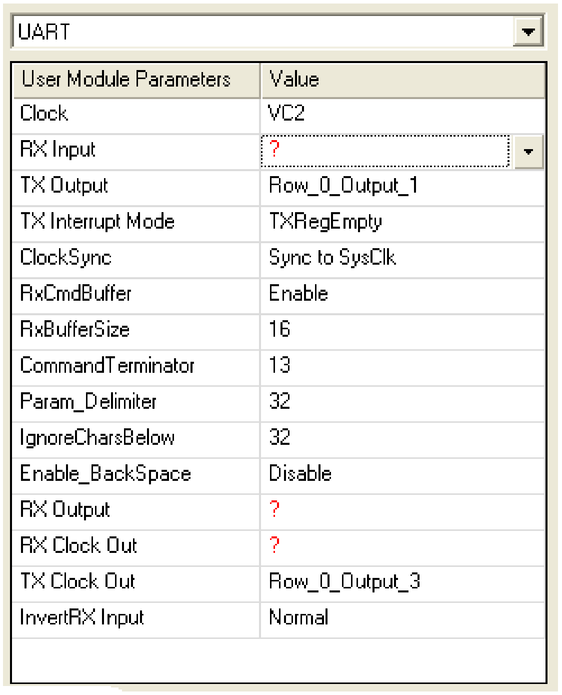
File: *RX8int.asm*

```
_RX8_ISR:

   ;@PSoC_UserCode_BODY@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.

   push A           ;save A and X on stack
   push X

   lcall RX8_bReadRxData  ;call the receiver's read API function
                          ;This will store the received data in the accumulator A
Append:
   mov X, [_bByteCount]   ;X is index for the buffer

   mov [X+_bRxBuf], A            ;The received data is transferred from A to the buffer.

   inc [_bByteCount]             ;the counter to track the num of bytes received is
incremented
   cmp [_bByteCount], 2   ;If 2 bytes have been received, that's one address
   jnz Skip               ;if not, 1 more byte is remaining. Exit.
   mov [_bByteCount], 0   ;If 2 bytes have been received, reset the byte count and..
   mov [_bAddrecd], 1            ;Set the flag to indicate to the main program
Skip:

   pop X                  ;Restore original values of A and X
   pop A
   ;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

Appendix B: Implementing Button Functionality and UART Tx with CY7C64215

1. Start a new PSoC Designer project and select the device CY7C64215.
2. In the interconnect view, select the UART user module from the Digital Comm tab; place it.
3. The module consists of a TX and an RX; only the TX is required for this project. Configure the user module as shown in Figure 15.
4. In the global resources table, set the VC1 divider to ‘12’ and the VC2 divider to ‘2’.
5. Route the UART’s TX block’s output to pin P1[5] via Row_0_Output_1.
6. To facilitate the digital readout of the current address, the LCD module is used. Select it from the Misc Digital tab.
7. Place it and configure the port to Port 0 and disable the bar graph.
8. Generate the project as explained previously and move to the application editor view.
9. The code for the button functionality and the TX8 UART transmitter is not included in this application note. However, the PSoC Designer project for the CY7C64215 is available with this application note and contains all the necessary code in a ready to build form.

Figure 15. Configuration of UART User Module


UART

User Module Parameters	Value
Clock	VC2
RX Input	?
TX Output	Row_0_Output_1
TX Interrupt Mode	TXRegEmpty
ClockSync	Sync to SysClk
RxCmdBuffer	Enable
RxBufferSize	16
CommandTerminator	13
Param_Delimiter	32
IgnoreCharsBelow	32
Enable_BackSpace	Disable
RX Output	?
RX Clock Out	?
TX Clock Out	Row_0_Output_3
InvertRX Input	Normal

About the Author

Name:	Mukund Krishna
Title:	Applications Engineer
Background:	Mukund has a graduate degree in Electrical Engineering from the University of Southern California. His past experiences at Cypress include video broadcast products. He currently works in the EZ-Color lighting solutions group.
Contact:	Mukund.Krishna@cypress.com
	Phone: (408)-432-7058

Document History

Document Title: Implementing an Integrated DMX512 Receiver Using the EZ-Color HB-LED Controllers

Document Number: 001-47023

Revision	ECN	Submission Date	Orig. of Change	Description of Change
**	2521105	06/27/08	UKK	New application note

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/