Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 001-56199

Spec Title: RDM COMPATIBLE DMX512 TRANSMITTER IMPLEMENTATION IN POWERPSOC(R) - AN56199

Sunset Owner: Madhan Kumar Kuppaswamy. (MKKU)

Replaced by: NONE

RDM Compatible DMX512 Transmitter Implementation in PowerPSoC®

AN56199

Author: Vincent Cai
Associated Project: Yes
Associated Part Family: CY8CLED0xD/G0x
Software Version: PSoC® Designer™ 5.1
Associated Application Notes: AN51891

Abstract

AN56199 provides an overview of PowerPSoC®, and the DMX512 and Remote Device Management (RDM) protocols. It also demonstrates the implementation of an RDM compatible DMX512 transmitter in PowerPSoC with the help of an code example.

Introduction

PowerPSoC

PowerPSoC is the first device to combine the power of an embedded controller with integrated high power peripheral functionality. This includes the following features:

- Four internal 32V, 1A rated low side n-channel MOSFETs
- Four 32V, 6 MHz rated current sense amplifiers
- Four 2 MHz hysteretic controllers that can be configured as buck, boost, or buck-boost
- A 5V regulator that operates between 7 and 32V input

Along with the flexibility of PSoC®, PowerPSoC offers additional functionality including Cypress's CapSense®. It also offers digital (PWMs, timers, counters), analog (ADCs, PGAs), and communication (DMX512, DALI, SPI, RS-232) options.

The ability to reconfigure power with this level of integration simplifies hardware design, lowers testing time, and decreases bill-of-material costs.

DMX512

DMX512, often abbreviated to DMX (Digital MultipleX), is a communications protocol used to control stage lighting. This was developed by the Engineering Commission of USITT in 1986, with subsequent revisions in 1990 leading to USITT DMX512/1990. ESTA took control of the standard in 1998 and revised it. The new standard, known as "Entertainment Technology - USITT DMX512-A - Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories", was approved by ANSI in November 2004. The current standard is also known as "E1.11, USITT DMX512–A" or just "DMX512-A", and is maintained by ESTA.

RDM

Remote Device Management (RDM) is a protocol enhancement to USITT DMX512 that enables bi-directional communication between a lighting or system controller and the attached RDM compliant devices over a standard DMX communication link. This protocol enables configuration, status monitoring, and management of these devices. This is done in a manner that does not disturb the normal operation of standard DMX devices that do not recognize the RDM protocol. The standard was developed by the ESTA Technical Standards (www.esta.org) and is officially known as "ANSI/ESTA E1.20, Entertainment Technology - Remote Device Management over USITT DMX512".

Technical Details

Figure 1. DMX512 Timing Diagram

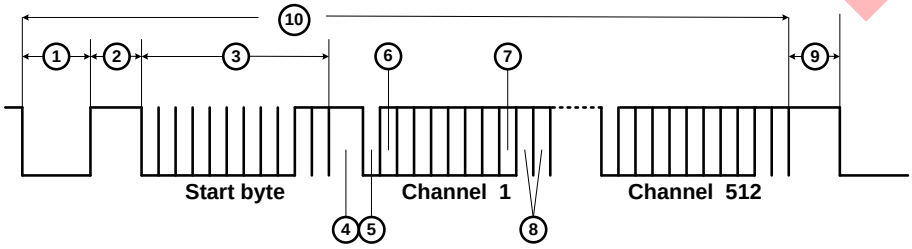


Table 1. DMX512 (RDM Compatible) Timing Values

Reference	Description	Duration (DMX512 TX)	Duration (RDM TX)
1	Space for break (reset)	Min 92 µs	176 µs – 352 µs
2	Mark after break (MAB)	12 µs – 1s	12 µs – 88µs
3	Slot time	44 µs	44 µs
4	Mark time between slots	0 µs – 1s	0 µs – 2ms[1]
5	Start bit	4 µs	4 µs
6	Least significant data bit	4 µs	4 µs
7	Most significant data bit	4 µs	4 µs
8	Stop bit	4 µs	4 µs
9	Mark before break (MBB)	0 µs – 1s	0 µs – 1s
10	Total Packet Time	1204 µs – 1s	440µS[2] +(n[3] x 44µS[4]) + ((n[3] -1) x 76µS[5])

Notes

1. This is the maximum inter-slot time for any individual slot. However, the total packet time requirement must be observed.
2. The 440 µS in the total packet time formula is the sum of the maximum break time and maximum MAB.
3. n is the number of data slots in the packet.
4. Slot time per ANSI E1.11.
5. The average inter-slot time must not exceed 76 µS.

Figure 1 and Table 1 illustrate that the DMX512 packet is composed of two parts:

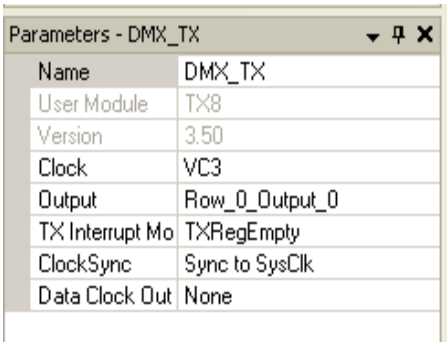
- Break header
- UART data packet (1 start bit, 8 data bits, 2 stop bits)

The UART data packet is a major portion of the DMX512 packet, which can be implemented by the UART TX device. The break header acts as a packet start indicator. It can also be generated by the UART TX device by setting a proper baud rate. The following sections show the detailed implementation.

Hardware Implementation

To implement DMX512 transmitter, only one TX8 user module with configurable clock input is required. The hardware resources to implement DMX512 transmitter are shown in the following figure.

Figure 2. TX8 User Module Configuration



VC3 is exclusively occupied because its frequency must be modified dynamically. It should be able to generate precisely 250 Kbps for the UART data packet and provide a proper frequency to generate break header.

Firmware Implementation

Break Header Generation

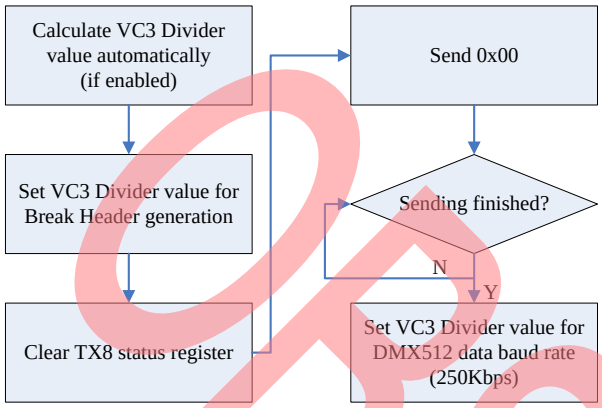
Algorithm: When the TX8 User Module sends data 0x00, it generates a negative pulse with a width of 9 bits (1 start bit and 8 data bits). With proper baud rate setting, this pulse can act as the DMX512 break header.

Implementation: Use VC3 clock source for TX8 clock input, to generate the break header. The TX8 baud rate should be set based on the following factors:

- DMX512 timing requires a minimum of 92 µs break header. (RDM protocol requires a minimum of 176 µs break header.)
- System clock frequency is set in the Device Editor ,as SysClk is the source for VC3.The tolerance (± 2.5%) of system clock (source of VC3) is considered. In the absence of a high precision input source from the 32.768 kHz crystal oscillator, the accuracy of the internal 24/48 MHz clocks is ±2.5% over temperature variation and two voltage ranges (3.3V ± 0.3V and 5.0V ± 0.25%).
- The clock rate must be set to eight times the desired bit transmit rate, as described in the TX8 user module data sheet.
- The break header has 9-bit data as described earlier.

The firmware associated with this application note, provides a section of code to calculate the VC3 divider value to generate proper baud rate based on these factors. Refer to the function CyDmxTx_SendBreakHeader() in the source file *CyApp_DmxTx.c*.

Figure 3. Break Header Generation Flowchart



MAB Generation

The requirement for MAB is 12 µs to 1s in DMX512 timing. In RDM timing, the requirement is 12 µs to 88 µs.

The code to send the first byte data after break header generates the delay to satisfy minimum MAB requirement for the DMX512 case. However, it is not easy to match the maximum MAB requirement (88 µs) in RDM timing. A 6 MHz CPU clock with ImageCraft compiler does not satisfy this requirement (see Table 2). To be compatible with RDM protocol, the global interrupt should be disabled during generating the break header until the first byte data is sent completely. Refer to the function CyDmxTx_SendPacketWithHeader() in source file *CyApp_DmxTx.c*.

Data Transmission

After MAB, TX8 starts to transmit DMX512 data. The firmware provides the following working modes:

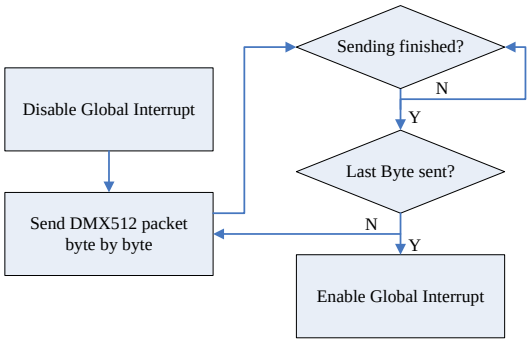
- Polling Mode
- Interrupt Mode
  - Transmission Complete Interrupt Mode
  - Transmitting Register Empty Interrupt Mode

Polling Mode

Polling Mode means that the firmware judges data transmitting status by polling the transmission completed bit of TX8. In this mode, the mark time between slots is predictable and stable; it only depends on the CPU clock. However, polling mode requires disabling global interrupt during data transmission, which may decrease CPU utilization.

Note To be compatible with RDM protocol, the average inter-slot time must not exceed 76 µS (as described in page 10 in the ANSI/ESTA E1.20). The CPU clock should be high enough to meet this requirement (see Table 2).

Figure 4. Polling Mode Flowchart



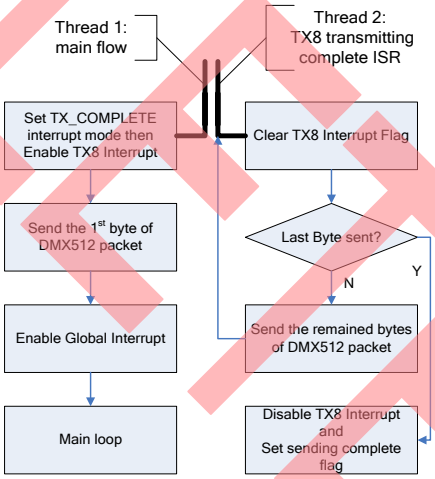
Interrupt Mode

Transmission Complete Interrupt Mode

Transmission Complete Interrupt Mode means that the firmware judges data transmitting status using the TX8 data transmission complete interrupt. This mode maximizes CPU utilization. However, the mark time between slots is unpredictable and unstable; it may be affected by other ISRs.

Note To be compatible with RDM protocol, the average inter-slot time must not exceed 76 µS (as described in page 10 in ANSI/ESTA E1.20). Table 2 shows that in case of the ImageCraft compiler with 6 MHz CPU clock, this requirement cannot be met. In other cases, special care should be taken to make sure that other ISRs do not occupy too much CPU utilization

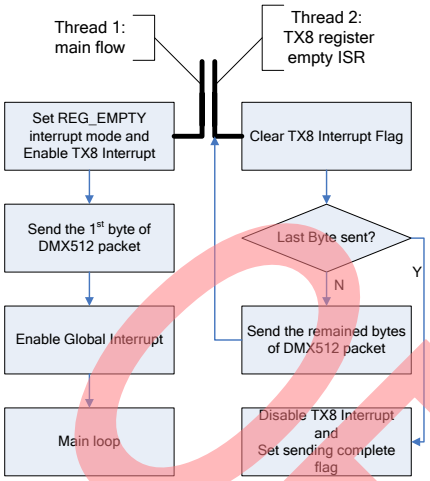
Figure 5. Transmission Complete Mode Flowchart



Transmitting Register Empty Interrupt Mode

Transmitting Register Empty Interrupt Mode means that the firmware judges data transmitting status by TX8 transmitting register empty interrupt. This mode maximizes CPU utilization and TX8 data throughput. However, the mark time between slots is unpredictable and unstable; it may be affected by other ISRs.

Figure 6. Transmitting Register Empty Mode Flowchart



Notes

1. These modes can be easily configured in the header file *CyApp_DmxTx.h* by the following macros:
   - ❑ CYDMXTX_INTERRUPT_MODE
   - ❑ CYDMXTX_VERSION

For example , the attached transmitter code example uses Transmission complete interrupt mode . The required definitions in the *CyApp_DmxTx.h* are as below *:*

```
#define CYDMXTX_POLLING_MODE (1 << 0)
#define CYDMXTX_REG_EMPTY_INT_MODE(1<< 1)
#define CYDMXTX_TX_COMPLETE_INT_MODE   (1 << 2)
#define                    CYDMXTX_INTERRUPT_MODE
(CYDMXTX_TX_COMPLETE_INT_MODE)
#define                              CYDMXTX_VERSION
(CYDMXTX_INTERRUPT_MODE)
```

2. Boot.tpl should be edited according to the mode selected. When using the polling mode, the TX8 ISR call should be eliminated. In the interrupt mode, a call to the ISR function '`CyDmxTx_ISRHandler`' should be placed in the appropriate digital block ISR call.

In the attached Transmitter project, the TX8 module occupies DCB03. So, the boot.tpl is edited as below:

```
// PSoC Block DCB03 Interrupt Vector
// `@INTERRUPT_11`
org   2Ch
ljmp _CyDmxTx_ISRHandler
reti
```

To meet the timing requirements of the DMX512 protocol and RDM protocol, the working modes should be carefully selected based on the values in Table 2.

DMX512 timing requires two stop bits for every byte data, but the TX8 user module for PSoC only supports one stop bit feature. Therefore, the second stop bit should be inserted manually by enabling some extra mark time between slots. This means TX8 should not work consecutively (the start bit of current byte data follows right after the stop bit of last byte data). Table 2 shows that, in most cases, TX8 works consecutively. It also shows that this mode cannot work for RDM timing in all cases.

Timing Analysis

The following table lists the timing raw data for the DMX512 transmitter. The data in the shaded cells show the conflicts with the requirements of the DMX512 transmitter (RDM compatible) timing. This table is a useful guide to select the CPU clock and working modes. This values were taken with PSoC Designer 5.1, SP1.

Table 2. Timing Raw Data

CPU Clock (MHz)	Working Mode	Break Header (µs)	MAB (µs)	Inter-slot (0-1) Mark Time (minus 2 stop bits 8 µs)	Inter-slot (1-last) Mark Time (minus 2 stop bits 8 µs)
24	Polling	190	54	22	22
12	Polling	190	76	38	38
6	Polling	190	120	68	68
24	Interrupt(TX_COMPLETE)	190	54	22	22
12	Interrupt(TX_COMPLETE)	190	78	40	40
6	Interrupt(TX_COMPLETE)	190	126	78	78
24	Interrupt(REG_EMPTY)	190	56	0	0
12	Interrupt(REG_EMPTY)	188	78	0	8
6	Interrupt(REG_EMPTY)	188	126	36	66

Code Examples

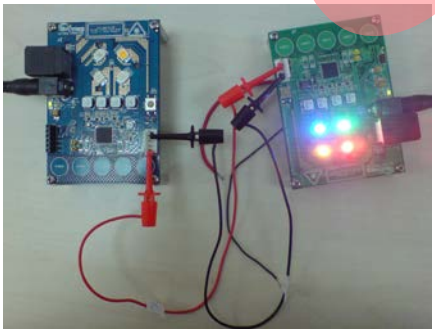
The code examples accompanying this application note demonstrates the DMX512 transmitter function.

Hardware Equipment

- 1 CY3268 demo kit for DMX512 Transmitter and CapSense buttons input.
- 1 CY3268 demo kit for DMX512 Receiver and dimming 4-channel LEDs based on the dimming value received.

Hardware Interconnection

- Pin 5 on connector J2 of the CY3268-DMX Transmitter is connected to Pin 4 on connector J2 of the CY3268-DMX Receiver. Pin 5 on J2 for transmitter is P1_0, which is the DMX_TX output. Pin 4 on J2 or receiver is P1_1, which is the DMX_RX input.
- Pin 2 on connector J2 of the CY3268-DMX Transmitter is connected to Pin 2 on connector J2 of the CY3268-DMX Receiver. Pin 2 on J2 is GND.
- Refer the schematics of CY3268 for further details on connector J2.



Functional Description

DMX Transmitter:

- This is responsible for the transmission of dimming data in DMX512 (RDM compatible) format, based on CapSense buttons input.
- Four CapSense buttons are used to select one of the four LEDs .
- The fifth CapSense button is used to control the dimming value of the selected LED.

DMX Receiver:

- The DMX receiver is built on the DMX512Rx module.
- This is responsible for receiving the dimming data from the DMX transmitter and driving the selected LED with the received dimming value.
- For a better understanding of the DMX512Rx module, refer the application note Implementing an Integrated DMX512 Receiver using the PowerPSoC Family - AN51891
- For a basic understanding of the Power components used for LED driving in PowerPSoC, refer PowerPSoC Firmware Design Guidelines - AN51012

The following resources are also available in the project files:

- *DMX_TX.zip*
  Note The source files of FW Module CyApp_DmxTx are available in this project
  - CyApp_DmxTx.c
  - CyApp_DmxTx.h
- DMX_RX.zip
- Demo Video: DMX_TX.3GP

Note The code examples are only to show the generation and timing of the DMX/RDM packets. Therefore, we use a single-end signal for this demo. For a full DMX communication application, a DMX transceiver is needed to convert the single-end signal to differential RS485 signals.

Summary

This application note introduces the implementation of an RDM compatible DMX512 transmitter in PowerPSoC. It illustrates the ability of PowerPSoC to meet the critical timing requirements for the DMX512 transmitter. PowerPSoC is also able to implement the RDM Responder to effectively match intelligent lighting control system requirements for the future.

References

- Implementing a DALI Receiver System Using PowerPSoC - AN52525

About the Author

Name:	Vincent Cai
Title:	Application Engineer Sr.
Contact:	wcai@cypress.com

Document History

Document Title: RDM Compatible DMX512 Transmitter Implementation in PowerPSoC® - AN56199

Document Number: 001-56199

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2865264	WCAI	01/25/2010	New application note.
*A	3205820	MKKU	03/25/2011	Updated the title with 'RDM'. Added description and code snippet in 'Polling Mode' section. Added details in 'Code Examples' section. Projects updated to PD 5.1 SP1. Added 'Timing Raw data' with PD 5.1, SP1. Removed references to Hi-Tech compiler.
*B	3503177	MKKU	01/20/2012	Obsolete document.

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