CSE 5400 Final Project Report

Objective:

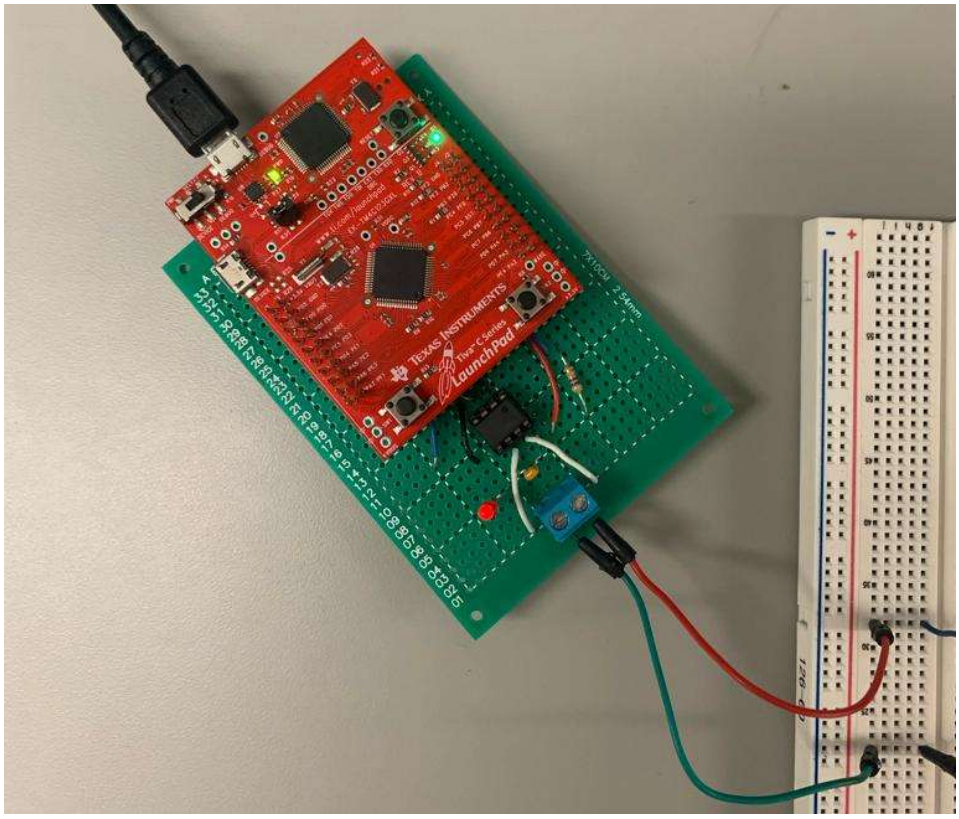
This project aims to create a multifunctional device for managing theatrical lighting through the DMX512-A protocol. The PC transmitter will receive commands from a computer via a virtual COM port and continually transmit a serial stream to control up to 512 devices on an RS-485 communication bus. On the other hand, the PC receiver will relay data received from the devices on the communication bus back to the PC through a virtual COM port. The design includes support for transmitting delayed actions on the DMX512-A bus to facilitate unattended operations. Devices connected to the bus will extract information from the asynchronous data stream to control one or more devices, and they will send acknowledgments to the controller when requested.

DMX 512 Working and Protocol:

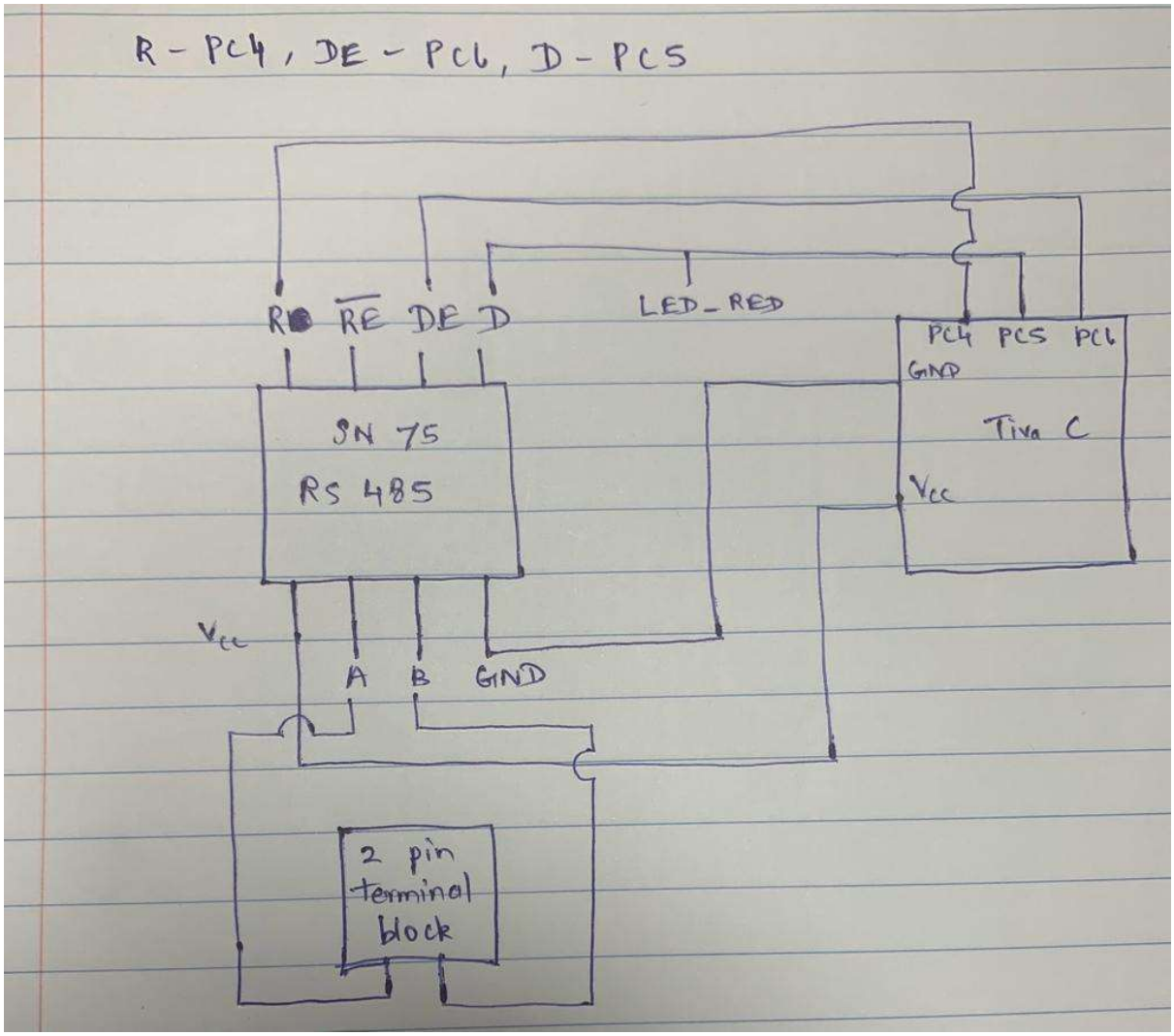
A DMX512 controller communicates using asynchronous serial data at a rate of 250 kbit/s with a fixed format of one start bit, eight data bits (ordered from least significant to most significant), two stop bits, and no parity. Each transmission frame includes a break condition, a Mark-After-Break, and Slot 0 or Data 0 carrying a one-byte Start Code, followed by up to 512 slots containing channel data, each consisting of one byte. The initiation of a packet is indicated by a break, followed by a "mark" (logical one) known as the "Mark After Break" (MAB), serving as a frame reference for data bytes within the packet.

The initial slot, designated as Slot 0, contains a one-byte Start Code specifying the type of data in the packet. A standard value for DMX512-compatible devices is a start code of 0x00. Following the start code, up to 512 slots contain control settings for slave devices, with each slot's position determining the controlled device and function, and its data value specifying the control set point.

Setup:



Block Diagram:



RS-485 is commonly used as the physical layer for DMX512 communication. RS-485 is a standard for serial communication that specifies the electrical characteristics of a balanced differential voltage digital interface. It allows for reliable communication over relatively long distances and is well-suited for connecting multiple devices in a network, making it suitable for DMX512 applications. In a DMX512 setup, RS-485 is often employed for communication between the DMX512 controller (such as a lighting console or a computer) and the lighting fixtures or other DMX512-compatible devices. The RS-485 bus enables the controller to send digital commands to multiple devices and receive data from them in a daisy-chain configuration.

Pin Configuration:

PC5 is the UART4 TX transmitter pin.

PC4 is the UART4 RX receiver pin.

PC6 is the DE_PIN that sets us to be in controller mode. It is off in device mode.

Code logic and explanations:

In the controller mode, the commands we type are “controller”, “on”, “clear”, “set”, “get”, “max”. To be in controller mode, we must first type controller, that sets DE_PIN to high and type on that turns on the TIMER 1A, one shot timer. The transmit buffer used here is global2 [], an array of size 512. The clear command sets the transmit buffer to 0. get command returns the value being sent to transmit buffer. max command overwrites the default MAX value set to 512.

When “on” is typed, it turns on the TIMER 1A as we call timer1_BRK() that uses UART line control register and UART Send Break. When it is set to '1', A Low level is continually output on the UnTx signal. Then, we enable the timer TIMER1A for 92us as we give the load value as 3680. The C code that implements this are

```
UART4_LCRH_R  |= UART_LCRH_BRK;
TIMER1_CTL_R  |= TIMER_CTL_TAEN;
```

This activates the timer1_isr() function. Here, we do the MAB (Mark after break) logic by setting UART line to HIGH for 12us. We use waitmicrosecond() function here. Next we wait until the UART4 TX buffer is full. We then send the START Code by UART4_DR_R = 0x00;

Then, we transfer the data from the transmit buffer to UART4_DR_R the UART 4 data register. We once again wait for the transmit buffer is full. Finally, we clear the interrupt at the end of the timer isr.

In the device mode, we set the DE_PIN to 0. In the UART4 ISR we first check if break error is set and set a flag. Then we check if break error flag is set and start code flag is not set. If it satisfies the condition, we set the start code flag to 1. Before, that we check if null start code is received. IF both the flags are set we transfer the data from UART data register to the receiver buffer. At the end, we clear the UART interrupt flag. The UART 4 baud rate 250kbps.

We also added ramp and pulse function while we are in the controller mode. Ramp logic uses its own timer TIMER2A and pulse uses TIMER0A. Both the timers are configured as periodic timer.

In the ramp logic we increment by the formula (stop_ramp - start_ramp) / (time_ramp). We start the ramp by storing start_ramp in transmit buffer. The TIMER2A load value is 40000 multiplied by time_ramp. This is for the ramp operation to continue for time_ramp number of clock cycles. We then enable the timer.

In the TIMER2A ISR, the logic is, we keep incrementing the increment value with the transmit buffer until transmit buffer value reaches the stop_ramp value. If the condition is false, we once again reset to start_ramp value. Finally, we clear the interrupt flag.

In the pulse logic, tmp_pulse is temporary counter variable that increments until it reaches the time_pulse value. We initially store the first_pulse value in the transmit buffer. If the condition fails, we set transmit buffer to last_pulse. We also turn off the timer manually here. Finally, we clear the interrupt flag.

Conclusion:

Thus, a multifunctional lighting controller device was successfully designed and implemented for managing theatrical lighting through the DMX512-A protocol.