Application Note: Milestone 1

Alex Marino & Cameron Bendzynski
Rowan University

October 21, 2018

1 Design Overview

The Stranger Things program is designed to create a string of different-colored lights, similar to the Christmas lights used to communicate with Will Byers in the popular Netflix series Stranger Things. For the program to work as designed, the development boards would each need to be connected to an RGB LED, and to each other. The head node of the chain would receive a large packet containing color information, and would have to set its LED to the specified combination of colors, and then send the remaining color information as a packet to the next board. This was achieved by communicating between boards over UART, and using pulse width modulation to control the red, green, and blue pins of the LED.

1.1 Design Features

- Receive UART Rx input (9600 baud rate)
- Parse packet length byte and identify invalid packet lengths
- Send packet length byte, less the three RGB values used, via UART Tx
- Parse Red, Green and Blue bytes
- Modify PWM output for each color based on input RGB values
- Output PWM value to each pin of RGB LED
- Send out remaining bytes in packet through UART Tx

1.2 Featured Applications

- Chained UART Customization
- RGB LED Customization
- LED Brightness Control
- Decoration
- Mood Lighting

1.3 Design Resources

The entire project, including the initial assignment, are stored on the team GitHub repository for ease of access. All files can be found here.

1.4 Block Diagram

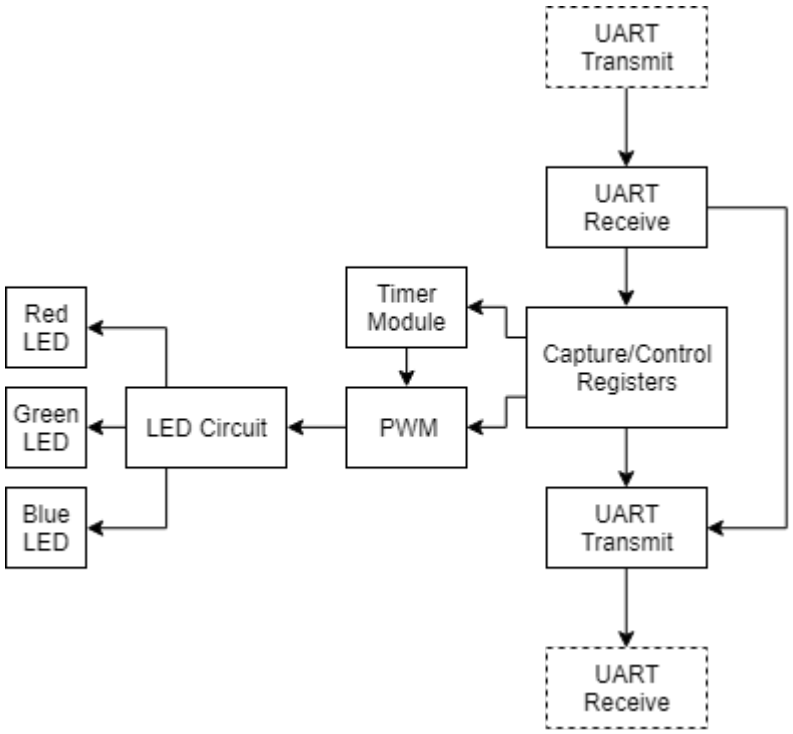

Figure 1: System Block Diagram

1.5 Breadboard Image

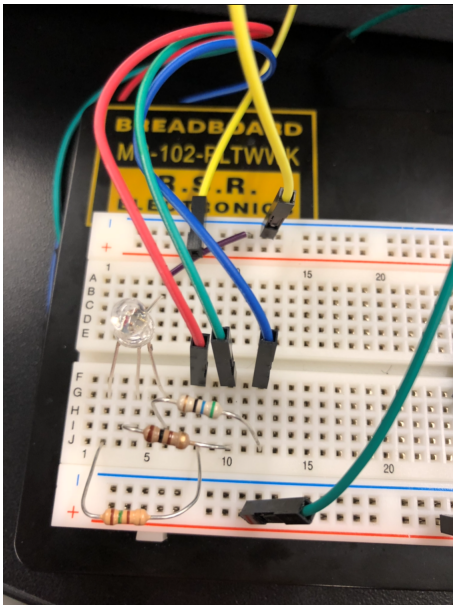

Figure 2: LED driving circuitry on breadboard

2 Key System Specifications

PARAMETER	SPECIFICATIONS	DETAILS
Packet Size Limit	0 - 255	Limited to a byte
CCR Limit	0 - 65,535	Limited to 16 bits
Board Input Voltage	5V	
LED Common Anode Voltage	3.3V	
UART Rx/Tx Buffer Size	0 - 255	Limited to a byte

Table 1: Key System Specifications

3 System Description

The system designed to replicate the Will Byers lights consists of two parts: the development board and the LED circuitry. As described in the Highlighted Devices section, the MSP430F5529 was chosen for the development board, and a common anode RGB LED was chosen to display the output. The development boards are connected to one another via their UART pins, which is described in the following sections. Each board also has its own LED circuitry, which it is connected to via its PWM output pins,

as shown in Figure 5. The system would receive a packet containing bytes of color information, take three bytes to address the red, green, and blue pins of its LED, and then send the updated packet to the next board in the chain.

3.1 Detailed Block Diagram

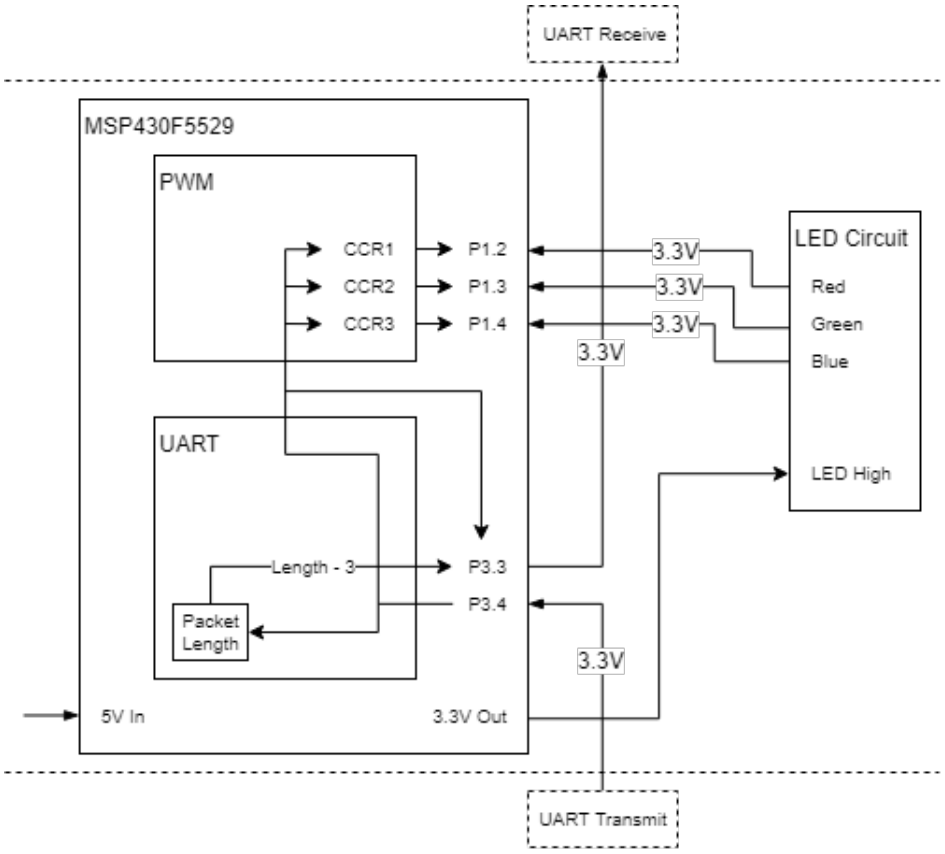

Figure 3: Detailed System Block Diagram

3.2 Highlighted Devices

Two key devices were required to implement this system:

- MSP430F5529: Receives and sends UART transmissions, controls LED values with PWM output
- Breadboard with LED circuitry: houses LED, resistors, and connections to MSP430F5529

3.3 Device 1: MSP430F5529

The board chosen for this project was the MSP430F5529LP. This board was responsible for carrying the software and selecting the RGB color by providing the correct PWM signal. The MSP430G2553 was also considered, making these the primary two to choose from. Ultimately the F5529 was chosen due to its number of capture compare registers. Because CCR0-4 were all used to implement the software PWM, only a single timer needed to be used. Software PWM was chosen for its flexibility, as any GPIO pins can be set as the output of the PWM. The G2553, with less capture compare registers per timer, would have required multiple timers to implement this PWM. Furthermore, the F5529 is capable of easily communicating via UART both over USB and via direct pin out connections without modifying the jumpers, making both testing and setup easy. Pins P1.2, P1.3, and P1.4 were connected to the red, green, and blue pins of the LED in order to control their color and brightness. The board was also used to communicate with other boards carrying the same software which were chained together via their UART Rx and Tx pins. This allowed a head node to send a packet down the chain such that each device could read in the packet, address its LED, and send the remaining bytes of the packet to the next node.

3.4 Device 2: LED Driving Circuitry

The driving circuitry of the LED, which is shown in Figure 5, is responsible for visualizing the PWM output of the F5529. For this configuration the LED used was common anode, meaning that one pin supplies power to all three colors, and the three color pins are meant to go to ground. This required the logic of the PWM to be reversed such that the output pin for each color would go low to enable the respective color. Resistor values were chosen such that the voltage to the red pin was 1.8V, the voltage to the green pin was 2.3V, and voltage to the blue pin was 2.7V. The common anode was driven by the 3.3V output pin of the F5529, but could also be driven by an external power supply.

4 System Design Theory

4.1 UART Communication

The UART communication routine in the software is what allows the device to interpret the incoming packets into something which can be used to address the individual LEDs. To enable UART over pins P3.3 (Tx) and P3.4 (Rx), UCA0 was chosen. As seen in the Test Setup section, UCA1 can be used for UART communication over USB. To set the baud rate at 9600 with the 1 MHz clock, the UCA0 baud rate register 0 was set to a value of 6, and the baud rate register 1 was set to a value of 0. Rx interrupt also needed to be enabled such that, whenever a new packet was received, the system would begin to process it.
When an interrupt was detected on the UART Rx P3.4, the system would enter the UART interrupt service routine. Here, the UART interrupt vector was checked using a

case statement. For the purposes of addressing the pins of the LED, the only interrupt the system was concerned with was a receive interrupt flag (RXIFG), which would set the UART interrupt vector UCA0IV to 2. Within case two, a second case statement was used to implement a state machine. The state machine was only considered if the Tx buffer was ready, meaning there was no pending data still on the pin. When the software is launched, the state counter is set to 1, so that the first Rx interrupt enters state 1 of the state machine. In state 1, a variable i is used to store the first byte of the packet. Each byte is received one at a time by the UCA0RXBUF, which is the 8-bit receive buffer for UART. Per the communication protocol, the first byte is the length of the packet, including the length byte itself and the stop character. The system checks that the packet is valid by ensuring that the length is greater than or equal to 5 and that the first byte is not 0x0D. The minimum valid packet length is five because the packet must include the length byte, a red, green, and blue byte, and a stop byte. The stop character 0x0D represents a carriage return in ASCII. If the first byte received were the stop character, it would mean that the original packet sent was not long enough to service the entire chain. As long as these criteria are met, the variable i is decremented, and the current byte is subtracted by three and sent to the next board in the chain. This is because the system will use the next three bytes to address its LED, and therefore the final packet sent to the next board will be three bytes shorter. This updated length packet is stored in UCA0TXBUF so that it may be sent via UART. The state variable is then updated to two.
In state two, the packet received is stored in the second capture compare register of timer 0 (TA0CCR1). This allows the red LED to be controlled by the PWM associated with this register. The variable i is then decremented and the state variable is increased to three. States three and four are identical, except for that they send the received byte to TA0CCR2 for green, and TA0CCR3 for blue, respectively. Finally, state five is reached. In state five, all additional bytes in the receive buffer are sent directly to the transmit buffer, and the internal length counter i is decremented. This operation continues until the variable i is equal to one. Once the internal length count is one, only the stop character remains. In this case, the software sends 0x0D over the transmit and then sets the state back to one, making the device ready to receive additional packets. An example packet can be seen in the Getting Started section as Table 2.

4.2 Pulse Width Modulation

The Pulse Width Modulation (PWM) system provides the microcontroller with the ability to operate the RGB LED at different levels of brightness. The PWM is achieved through the use of a single timer (Timer A0) operated at 1MHz through the use of SMCLK. Each color in the RGB LED is controlled individually, utilizing four Capture/Control registers (CCR) as a whole. CCR0 is used to turn all three LEDs on after 270 clock cycles, while the individual Capture/Control registers labeled CCR1 for red, CCR2 for green, and CCR3 for blue are able to be operated at any value between 0 and 255. This limit is due to the fact that the packet received by the system contains only a byte for each LED, and that the UART receive buffer is only eight bits.

Rather than polling, the PWM utilizes interrupts and a timer to control how long each LED is on for. The timer is set in Up Mode to provide a baseline for all three LEDs for the sake of time-keeping. This decision was made because three separate timers for three separate LEDs could lead to a desynchronization of the PWM. The values of CCR1 - CCR3 decide how long the corresponding LED will be on for, shutting that LED off once the timer reaches the value set in the register. The LED will remain off until the timer reaches the value of CCR0 at 270, where all three LEDs turn on and the timer resets back to zero. Additional logic is used in the system to ensure the LED is completely off when the Capture/Control Register corresponding to the LED is set to 0. Due to the limitations of the board, without this additional logic, the LED would appear very dim as opposed to the desired off state. The value of the CCR0 is never altered, but the other Capture/Control Registers are manipulated through the packets received through UART.
The control for the individual LEDs is done on Pins 1.2, 1.3, and 1.4, for red, green, and blue respectively. Each pin is connected to the breadboard and its corresponding LED. Since the RGB LED is common anode, the positive lead of the LED is connected to the 3.3V source on the MSP430F5529, and each pin grounds that connection across the LED, creating a potential difference that powers the LED. Due to the use of a common anode LED, the logic regarding the powering of the LEDs had to be inverted. This meant that setting the pins to high would turn off the LED, whereas setting them to low would turn it on.

4.3 Design Requirement 1: UART Communication

To properly enable UART communication, several considerations must be met.The first is which Universal Serial Communication Interface (USCI) to use. On the MSP-430F5529, USCI A0 can be connected to any GPIO pins. Choosing P3.3 and P3.4, as was the case for this implementation, connects the send and receive buffers to these pins. USCI A1 allows for UART over USB. This implementation was used when data needed to be received directly from the laptop, either during testing or for receiving the initial packet at the start of the chain. The next consideration is the baud rate. On the MSP430F5529, the baud rate is controlled by the baud rate registers. The values of these registers should be referenced from the device data sheet. The USCI also needs to be reset, and then enabled, before it can begin sending and receiving. This is controlled by modifying the UCSWRST bit of the USCI control register. Finally, a consideration for how UART communication is going to be used is recommended. For this design, only a UART receive would generate an interrupt, and therefore interrupt only needed to be enabled for receive.

4.4 Design Requirement 2: PWM/CCRs

The value of CCR0 was originally set to 256 since setting the other Capture/Compare Registers to the same value as CCR0 produced some unwanted effects, such as the LED appearing to be off. In this case, the highest possible value for CCR1 - CCR3 is 255, so setting CCR0 to 256 prevents this situation from ever happening. However,

through testing with the RGB LED, it was observed that the LEDs would still appear dim at values close to CCR0, not just exactly the same. To avoid this issue, the value of CCR0 was raised to 270 from 256, and tested on all three LEDs for effectiveness. While each LED individually worked perfectly, all three together had some unforeseen consequences in which the LEDs were brighter at half brightness than they were at full brightness. This problem was not solved.

4.5 Design Requirement 3: LED Circuit

The resistance values of the circuit were originally 400 across the board, but later changed to 150 for red, 100 for green, and 56 for blue. This was done to provide the proper current of 10mA through the individual LEDs, as well as the proper voltage for each color.

5 Getting Started / How To Use The Device

This software is designed to be used in conjunction with other microprocessors running the same or similar software. For simplicity, a chain of two MSP430F5529 devices running the same software, plus a third MSP430F5529 running the USB communication software, will be considered, but the software is designed in such a way that there is a practically limitless amount of boards that could be linked together. To begin, the LED circuitry found in Figure 5 must be created using a breadboard or other such device. For this example, three LED circuits would be created. The RGB pins of the LED should be connected to pins P1.2, P1.3, and P1.4 of each F5529. The LED common anode was connected to the 3.3V output of each F5529. The USB communication board, which is described further in the Test Setup section, should be connection to the laptop via micro-USB. In order to send packets to the chain, software such as RealTerm or some other serial communication software may be used. Then, pin P3.3 (UART Tx) of the first board should be connected to pin P3.4 (UART Rx) of the second board. On each board that is not the beginning or end of the chain, P3.3 will be connected to P3.4 of the next board, and P3.4 will be connected to P3.3 of the previous board. The boards also need to have common ground and so the ground pins should be connected across each board in the chain. Each board in the chain can either be powered via the Micro-USB or a power supply.
Once the boards are connected to each other, the laptop, and their respective LED circuitry, packets can be sent to control the LEDs. Using a piece of software such as RealTerm, a packet would look like the following:

Length	R	G	B	R	G	B	R	G	B	STOP
0x0B	0xFF	0xFF	0xFF	0x00	0xFF	0x00	0x7F	0x7F	0x7F	0x0D

Table 2: Example Packet

The first byte is the length byte, and it tells the software how long the entire packet

is, including itself and the stop character. Then, each byte thereafter is interpreted as a value for red, green, and blue, in that order. In this example, the first LED would be white with 100% brightness, the second would be completely green, and the third would be white at 50% brightness. The stop character 0x0D, which is a carriage return in ASCII, signifies the end of the packet.The minimum packet length is five (0x05) which would include a single R, G, and B byte, and the stop character. To add more boards to the chain, three more bytes, one each for red, green, and blue, need to be added to the packet, and the length byte would need to be updated to reflect the new size. A full description of how the system handles packets can be found in the System Design Theory section.

5.1 Communicating with the Device

Communication with the device can either be done via UART over USB or via UART using the Rx and Tx pins as described in System Design Theory. For UART over USB, USCI A1 should be utilized in conjuction with a serial communication software such as RealTerm. For UART over direct pin connections, USCI A0 should be used, and no additional software is needed, assuming the packets arrive from a board further up the chain.

6 Test Setup

In order to verify functionality of the system, additional software needed to be utilized. First, the original, unedited milestone software under test was flashed to an MSP430F5529. Then, the software RealTerm was installed on the test computer. This program allows communication via USB with the MSP430, and allows test packets to be sent to the board. However, the unaltered implementation is unable to communicate over UART via usb. This is because UART over USB uses UCA1, but the final implementation of the code uses UCA0 configured to communicate via pins P3.3 and P3.4. So, in the flashed version of the code, UCA0 was used. However, in the version of the code meant to communicate over USB, UCA1 was used for receiving and UCA0 was used for sending. This way, the board connected to the computer could receive packets via USB and send them using UART to the board under test.
Once the software under test was flashed to one F5529 and the modified software was flashed to another F5529, testing could begin. The modified F5529 was connected via USB to the laptop with the RealTerm software. The pins P1.2, P1.3, and P1.4 were connected to the red, green, and blue pins of the LED, respectively. Each LED pin had its own resistor between it and the board, and the common anode pin of the LED was connected to 3.3V power. The values of the resistors can be seen in Figure 5. Next, the board under test was connected to its LED in the same way. The UART receive pin of the board under test, P3.4, was connected to the send pin of the modified board, P3.3. The grounds of the two boards were also connected to each other. Next, the RealTerm software was opened and a connection was established. The software uses a baud rate of 9600, and Windows Device Manager was used to

determine the correct COM port. RealTerm was used to send packets to the modified board as well as the board under test.

Several types of packets were sent to test functionality. If a packet length of 5 was sent, it was followed by three values ranging from 0x00 to 0xFF, each of which would correspond to the red, green, and blue of the first LED. If a packet length of 8 or greater was set, bytes 2-4 would control the first LED and bytes 5-7 would control the second LED. This verified that UART communication was successful from laptop to board and also from board to board. This also demonstrated that the PWMs were correctly addressing the pins of the LED.

Further testing was done by connecting several configured boards together. For this test, only the board under test software, which was the final main.c, was needed. In total, 11 boards were connected via their UART pins, and the first board in the chain received a large packet from a laptop at the beginning of the chain. This demonstrated that the boards could successfully send the data from the beginning of the chain, all the way to the end, with a different combination of colors being sent to each LED.

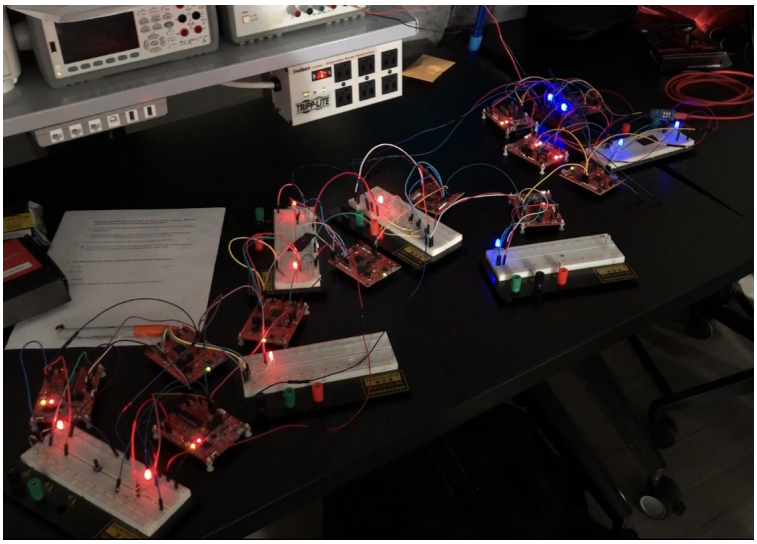

Figure 4: Many MSP430s Connected

7 Design Files

7.1 Schematics

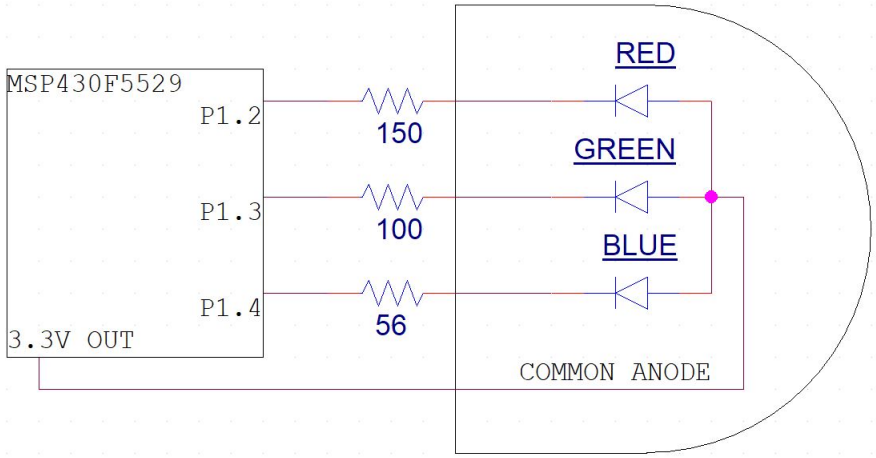

Figure 5: LED driving circuitry schematic.

7.2 Bill of Materials

- MSP430F5529
- Micro USB Cable
- Breadboard
- Common Anode RGB LED
- Resistors
  - 160 Ohm
  - 100 Ohm
  - 56 Ohm
- Jumpers and other miscellaneous wires