Project 2: UART

CECS 447

Liam Gogley 012742858
Mark Munoz 013069795

Table of Contents

Table of Figures

I. Introduction:

The purpose of this project was to explore the construction and interfacing of the UART protocol with a desktop computer. This project uses two microcontroller units (MCUs) to send data over a UART protocol, or universal asynchronous receiver/transmitter. This project uses two of the total 8 onboard hardware UARTs for the communication, and a micro USB interface to a desktop computer for character transmitting and receiving.

II. Operation:

The project operates from the onboard UART modules, UART0 and UART1. A potentiometer is connected on a 3.3v source, and the adjusted voltage is an input to Port E0 of MCU 2. The analog voltage value is converted to a 12 bit resolution digital value that is then sent via UART1 to MCU 1. The first MCU uses the digital value to control the duty cycle of the hardware PWM for increasing or decreasing the brightness of the onboard LED. MCU 1 also receives commands from a host computer via UART0. This operation allows for MCU1 to relay the command to MCU2, which will then start blinking the LED once per second.

In order to receive or transmit data with the UART protocol, a baud rate must be agreed upon from the two devices. The baud rate determines the total amount of bits per second to be sent or received by the UART, and it is imperative to the protocol's operation. The data connection of the terminal relies on a communication port or teletype (tty) device determined by the OS kernel. Once connected, the user has the option to display the ADC value, and input a character to be decoded by MCU1.

When sending the character 'R' to MCU 1, it will relay that command to MCU2, which will cause a red LED will blink every 1 second. MCU2 then sends confirmation data back to MCU1, which will output a string to the serial terminal through UART0. The first 'R' character sent will cause the red LED to flash and display 'Red LED is now blinking.' Upon sending any subsequent 'R' values, the terminal will report that the 'Red LED is already blinking.' In order to stop the red LED flash, onboard switch 1 of MCU2 must be pressed, and will transmit the messages 'Red LED is now off' and 'Red LED is already off' upon the first and any subsequent button press using the same confirmation data approach as above.

III. Hardware:

An adjustable analog voltage is produced from a 3.3v voltage source and a 10k potentiometer. This value is converted to digital value on MCU2, and is transmitted over UART 1 to the receiver on MCU 1. The baud rate chosen for our implementation was 57600, with a bit time of 17.631 µs. To ensure proper operation and data transmission, a common ground was necessary for an equivalent zero voltage reference.

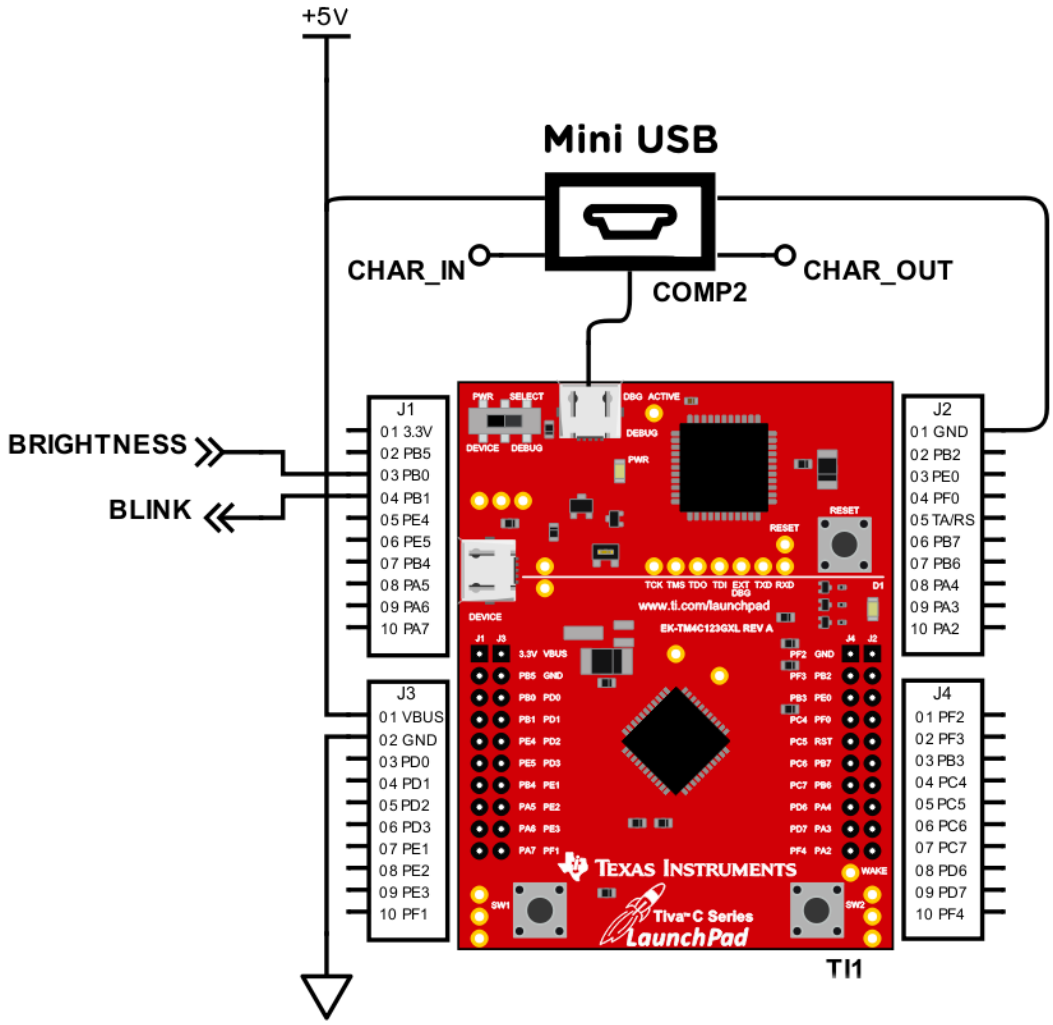

Illustration 1: MCU 1 Electrical Connection

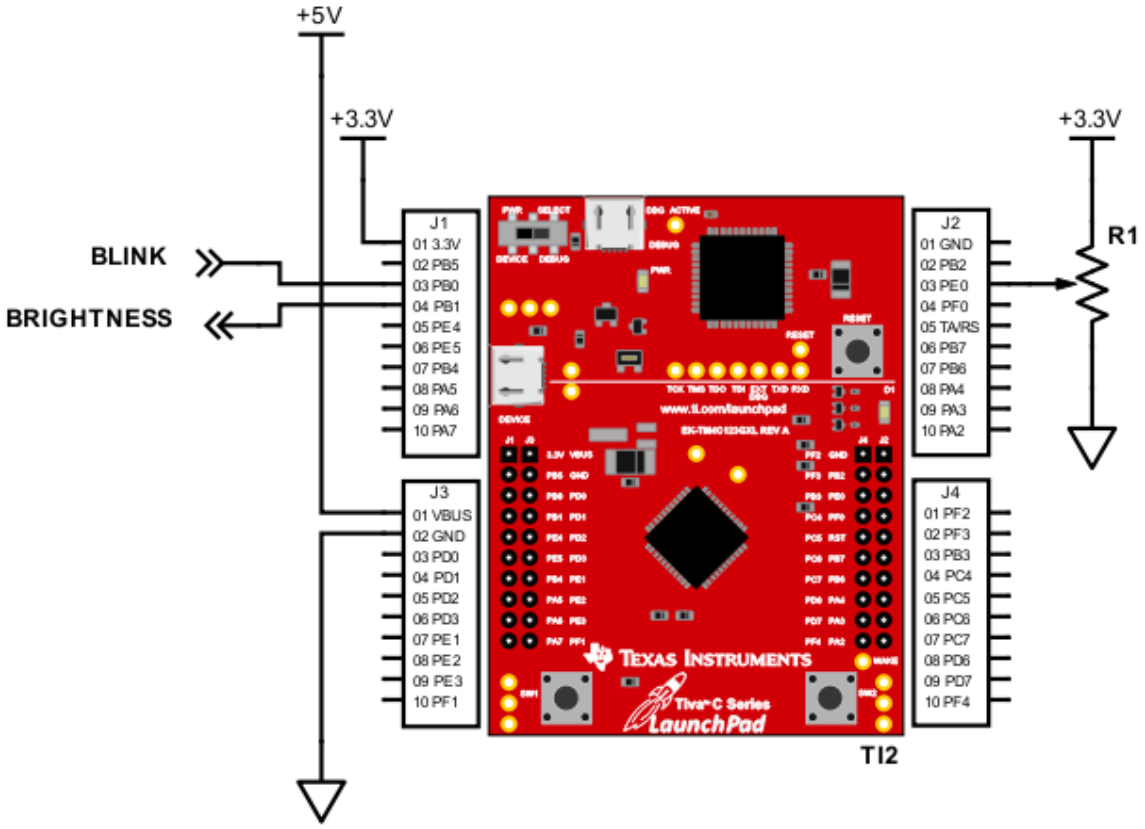

Illustration 2: MCU 2 Electrical Connection

IV. Software:

To generate the baud rate necessary for operation, the values for the integer baud rate divisor (IBRD) and fraction baud rate divisor (FBRD) were assigned using the following formulas:

IBRD = int(80,000,000 / (16 * 57600)) = int(86.80555555)
FBRD = int(0.805555 * 64 + 0.5) = 52

It was very important that the UARTs between two communicating devices were initialized using the same baud rate, as this ensures proper communication between the two devices.

One feature we felt was important to include was to only output ADC value on the serial terminal when the potentiometer was being turned. If the potentiometer was stationary, there would not be a continuous stream of the same ADC value being sent to the terminal. To accomplish this, we used two variables and would

alternate which variable got the most recent potentiometer value. If the most recent potentiometer value of the POT differed by more than a certain tolerance(we used 0.001%) since the last time the POT was sampled, then the value of the POT was updated on the screen.

To handle the toggling of MCU 2's red LED, we implemented a 10 Hz timer interrupt and a toggle flag. The toggle flag was asserted when MCU2 received the character 'r' from MCU1's UART. Once the toggle flag was asserted, the state of the red LED would toggle after 10 timer interrupts occured, resulting in an LED that blinks every 1s. Additionally, an external interrupt switch was implemented on MCU2 that would set the toggle flag low, ensure the red LED was off, and send a confirmation message back to MCU1 that the LED was no longer blinking.

To control the brightness of MCU1's blue LED, a decimal value was read from UART1, which was connected to MCU2. If the decimal value was determined to be valid(i.e. not a NULL character), the value was then converted into an equivalent duty cycle using the following equation:

$$Duty = (potVal / 4095) * period$$

where potVal is the decimal value received from the UART and period is the period of the hardware PWM(intialized to be 40,000 for our implementation).

The sending and receiving commands were was implemented by sending a predetermined character over the UART. If a recognized character was received from a microcontroller, a certain block of code would be executed and if any unrecognized characters were received over the UART, no action would be taken. This method of communication was used for sending 'blink LED' commands, 'command received' messages, and 'LED now off' messages.

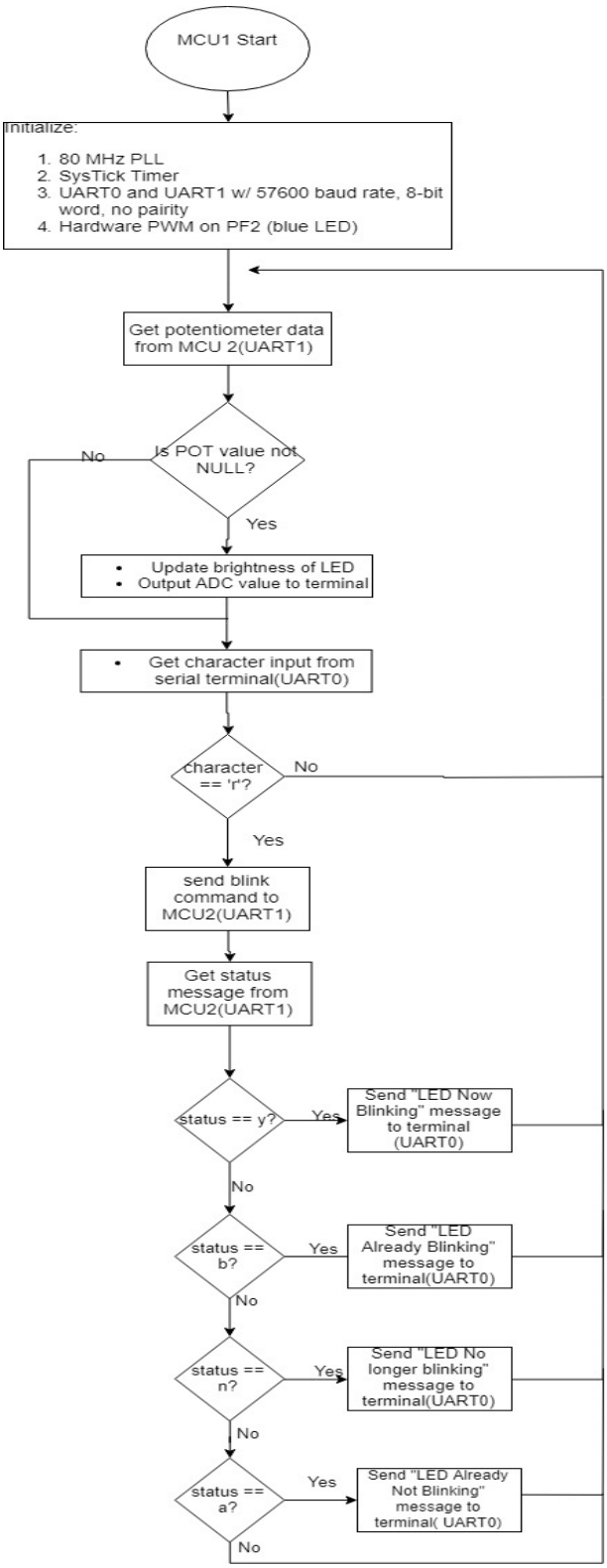

Illustration 3: MCU 1 Software Flow

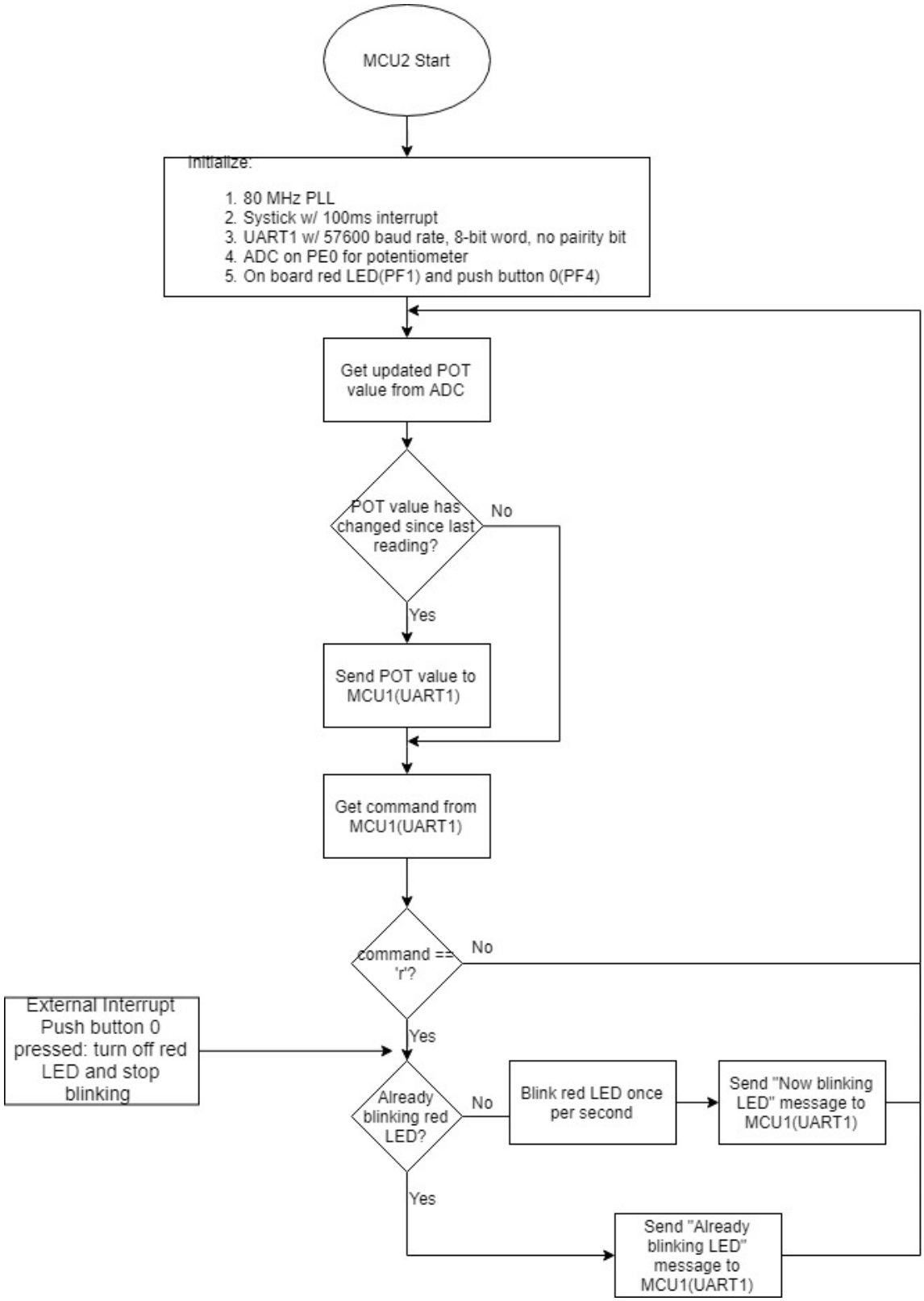

Illustration 4: MCU 2 Software Flow

V. Conclusion:

In this project, we explore the usage and implementation of the UART protocol to send and receive serial data between two MCUs. Using two microcontrollers we can utilize two of the eight onboard UART modules to both send and receive characters from a terminal. This allows for both the control of the brightness of the blue LED present on MCU 2, and for MCU 1 to blink the red LED when the character ‘R’ is sent. The purpose of this project was to evaluate the implementation of the UART protocol, and the importance of communication protocols in designs for serial data transmission.

One challenge we faced was getting MCU1 to differentiate what data was being received from MCU2. Since we were using one UART to send two different types of data, potentiometer output data and command received data, it was important that command received messages were not being interpreted as potentiometer outputs and vice versa. To do this, we implemented a SysTick delay, where all potentiometer data sending and receiving was done before the SysTick delay, and all command data sending and receiving was done after the SysTick delay. While this method is not ideal, as the SysTick delays slows the overall execution of the code down, it was one workaround we found. Another way of solving this issue was to implement dedicated UARTs for potentiometer data and command data, however we elected not to do this as the project instructions were to implement a single UART between the two microcontrollers.