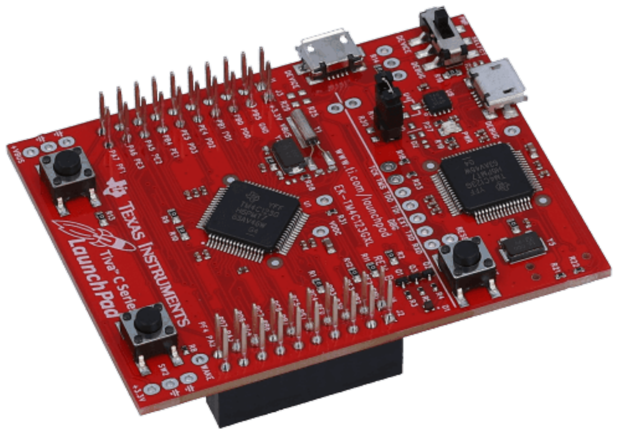
CSE-3442: Embedded Systems 1
Term Project: Plan and Design Report



Subhaan Elburz
ID: 1002135522
sme5522@mavs.uta.edu
Department of Computer Science & Engineering

November 26, 2025

1 Introduction

1.1 Project Overview

The term project for this semester in Embedded 1 is actually designing an embedded system that can communicate (send and receive) data using an IR333A LED and a TSOP134 IR Receiver.

My understanding of the project is basically that the IR333A is an LED that emits infrared light, and the goal is to use the IR LED to transmit data to the TSOP134 IR Receiver. And this sounds pretty straightforward, but the TSOP134 will only pick up light at 38 kHz, so we basically have to turn the LED on/off at that frequency while also transmitting the data.

So, the project should work so that I can send data from my laptop to the common terminal interface, which will then send it to UART0 at 8N1 (115200 baud). After that, the microcontroller will store the data in a variable. Then, it will send the data to the IR333A, which will transmit it in light at 38 kHz to the TSOP134, which will then send it to another system (or back to itself in testing).

However, sending the data to the IR333A is not that simple either. At first, I thought I could just send it at the required 300 baud 8E1 with another UART, but I could not understand how you would send that data at the required frequency for the receiver to work.

Initially, I thought the best solution would be using some GPIO interrupts on the pin where we transmit the data from the UART, so that when it goes high, we could enable the 38 kHz PWM signal, but with some help from Abhishek, I devised the following plan instead.

1.2 Proposed Plan

1. First, I will send data like the character ‘C’ from my laptop using the common terminal interface. This will be done using UART0, and I will have UART0 configured to 8N1 at 115200 baud.

2. Then, I will call getsUart0() to “get” the data from the UART0 RX FIFO and store it into some variable on the microcontroller itself.

3. Now, the solution that I created for transmitting the data using IR light waves involves using UART, PWM, and some logic gates together.

4. So first, I will generate the required 38 kHz square wave using PWM on the GPIO pin PB6.

5. Then, I will send out the same data/character out of the UART7 TX FIFO at 300 baud and 8E1 to the GPIO pin PE1.

6. Additionally, I will have to use a NOT gate to invert the UART7 TX signal because the TSOP134 receiver is active low and is default high (so like if LED is on, the receiver output would be low, and for like the start bit (0) we need to invert it so that the LED turns on, which would make the receiver output low correctly).

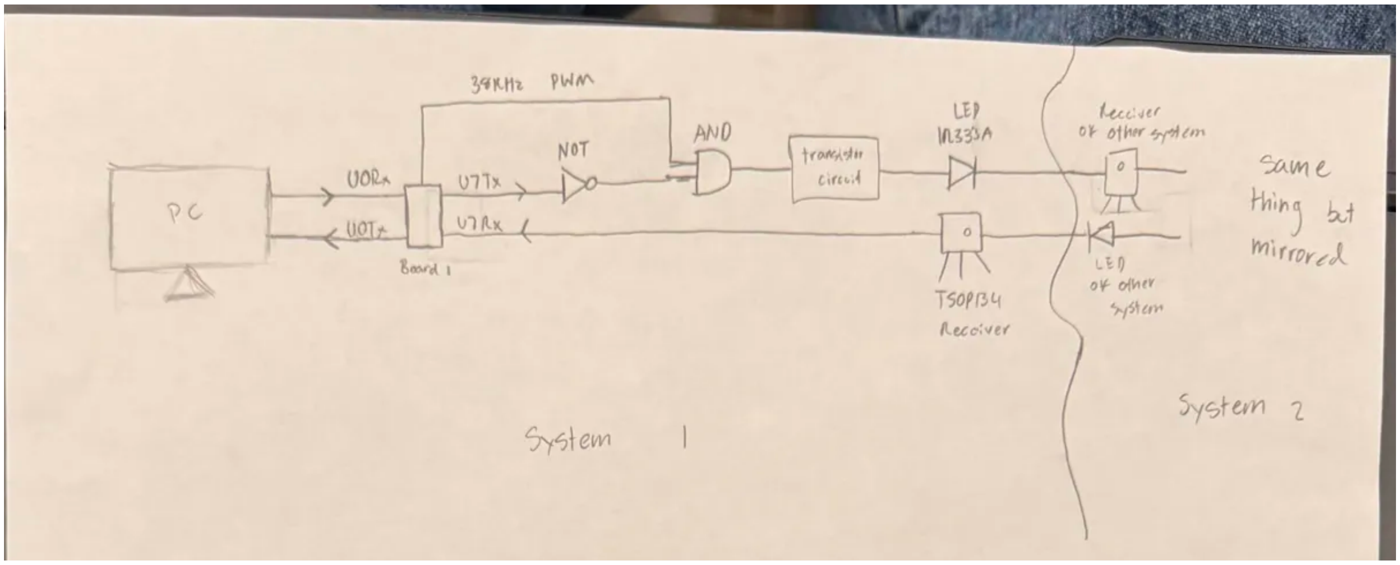
7. Afterwards, I will take the PWM signal and the inverted UART7 signal and put them into an AND gate, which will basically make an output signal that will allow for the data to be sent at the required frequency.

8. Then, this output signal will be sent into the NPN transistor circuit from lab 3 (with resistor modifications), which will control the IR333A LED to transmit the data to the TSOP134 receiver on the other board.

9. Lastly, the receiver will be connected to the GPIO pin PE0 (The UART7 RX pin), and I will implement UART interrupts so that when it receives any data, the interrupt will trigger, and send the received data back to UART0 TX, so I can see it on my laptop.

2 Block Diagram

2.1 Diagram



Just to be clear, the communication is still wireless between the IR333A and TSOP134. I just connected it to show how it should work with the other system; I just forgot to erase the wire connection between the two systems.

2.2 Pins to be Used

VBUS (5 V)
GND
U0Rx: PA0
U0Tx: PA1
U7Rx: PE0
U7Tx: PE1
M0PWM0: PB6

2.3 Timing Constraints

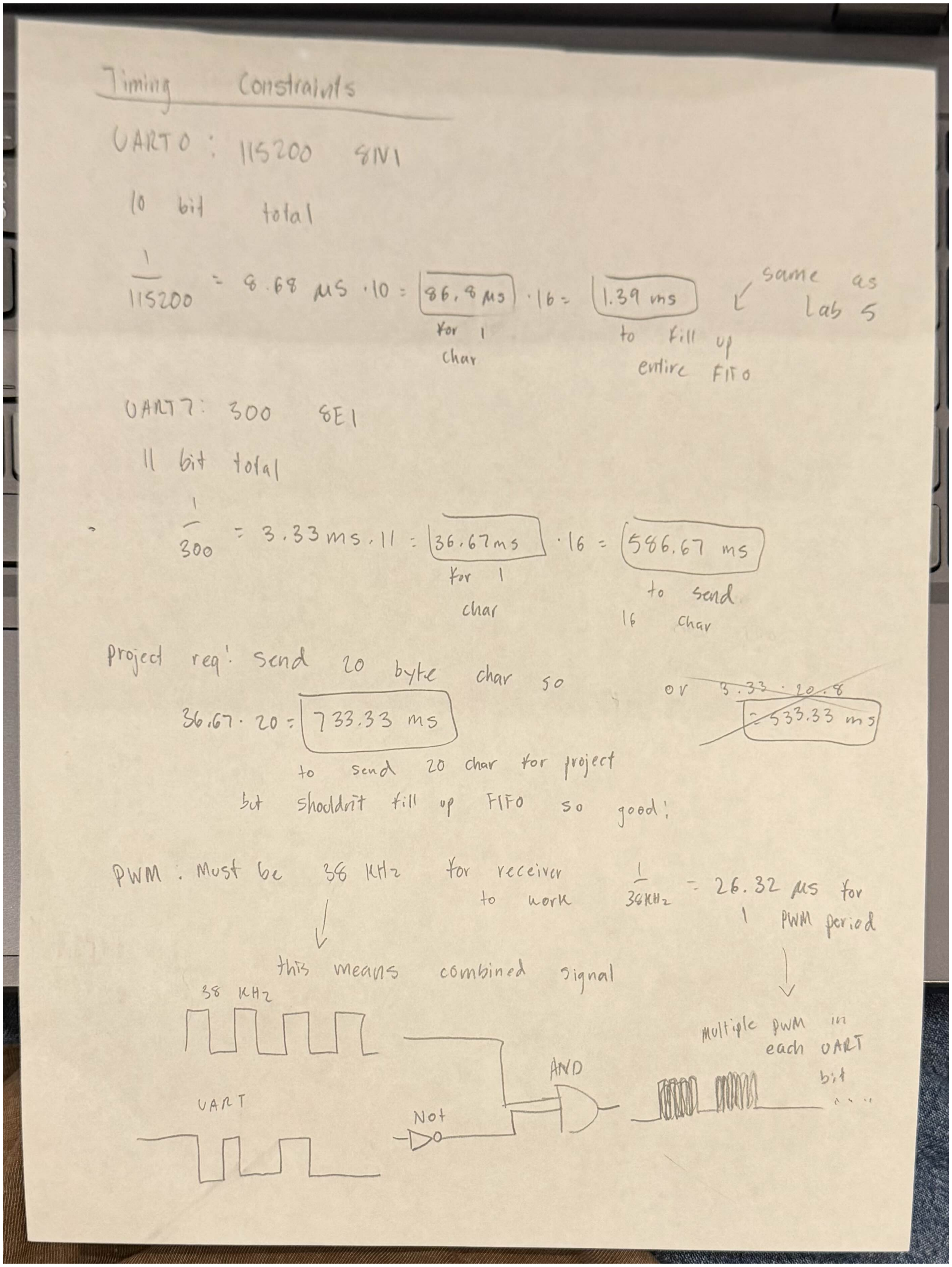
The main timing constraints for the projects are the UART0 115200 baud rate at 8N1, the UART7 baud rate of 300 at 8E1, and the 38 kHz PWM signal.

So, UART0 sends 1 bit in 8.68 µs, which means it sends 1 character at 86.8 µs (8N1 has 10 bits), which also means that the FIFO would fill up in 1.39 ms (if data is spammed continuously at least).

UART7 is way slower since it is at 300 baud, meaning it sends 1 character in around 36.7 ms (8E1 has 11 bits, so that also increases time). And also, the required 20-byte message would take around 0.73 s, which seems to work fine, as that is still pretty fast.

The 38 kHz PWM signal has a corresponding period of 26.3 µs, which means that there are like a couple of hundred PWM cycles in each transmitted bit of UART7, which means that it should accurately send the data when the two signals are ANDed together.

Overall, I believe my planned approach should have all of these timing constraints in check so that it works.



Timing Constraints

UART0: 115200 8N1

10 bit total

$\frac{1}{115200} = 8.68\ \mu s \cdot 10 = \boxed{86.8\ \mu s} \cdot 16 = \boxed{1.39\ ms}$ ← same as Lab 5

for 1 char

to fill up entire FIFO

UART7: 300 8E1

11 bit total

$\frac{1}{300} = 3.33\ ms \cdot 11 = \boxed{36.67\ ms} \cdot 16 = \boxed{586.67\ ms}$

for 1 char

to send 16 char

project req: send 20 byte char so

$36.67 \cdot 20 = \boxed{733.33\ ms}$ or ~~$3.33 \cdot 20 \cdot 8 = \boxed{533.33\ ms}$~~

to send 20 char for project

but shouldn't fill up FIFO so good!

PWM: Must be 38 kHz for receiver to work $\frac{1}{38\,kHz} = 26.32\ \mu s$ for 1 PWM period

↓ this means combined signal

↓ multiple PWM in each UART bit

3 Software Approach

3.1 Threads of Execution

For my plan, there really are only two threads of execution. The first is going to be the main line of execution with all of the code, and the other will be the UART7 interrupt handler for when it receives any data.

In the main thread, it should be pretty much like the other labs, where I initialize everything, UART0, UART7, and the PWM signal. Then, it will enter a while loop for the common terminal interface and wait for user input. The only command should be send, and basically, the command should transmit data using UART7 if it is typed in correctly.

The second thread will be the handler for when UART7 receives any data. I was not sure if I would be able to use GPIO interrupts on the same pin as the UART RX, and Abhishek told me to use the UART ones. So, the plan is to set it up so that there will be an interrupt when any data is received on UART7. Then, in the handler, it should send that data back to UART0, and I believe that by using an interrupt here, it will become full duplex.

So, for the UART7 interrupt, the priority should just be the highest since there are no other interrupts. Some configuration values for the interrupt are that UART7 is Interrupt 63, which means I would have to use the NVIC_EN1 register, and also set the priority using the NVIC_PRI15 register.

3.2 Requirements to be Met

With this approach, I believe that all 9 of the basic requirements will be met because of how I am planning to set up the UARTs, the timing, and the use of the UART7 RX interrupt.

Additionally, from the goal requirements, I believe I could do requirements 1, 3, and 7.

So, requirement 1 is ensuring that it can be full duplex, and I believe that by using a UART7 RX interrupt, it should be able to do both at the same time.

Requirement 3 is changing the message limit to 128 bytes, and that doesn't really seem to be too complicated, as I would just have to change the size of the variables.

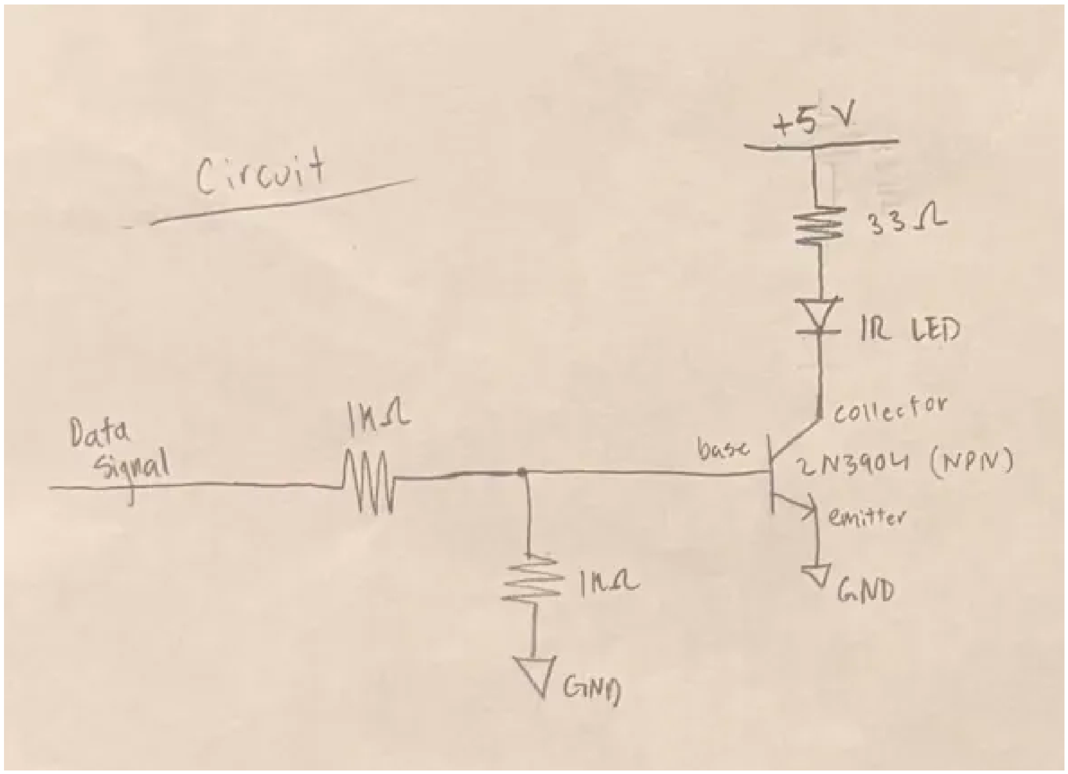
Requirement 7 also seems pretty straightforward, as I just have to turn the on-board blue LED on/off inside of the interrupt handler.

4 HW Checkout

4.1 Hardware Testing

So the main way I plan on testing the hardware is building the circuit on the breadboard and debugging with the oscilloscope before soldering anything.

I did actually build the following circuit and tested it with just a GPIO pin going high/low, and it did turn the LED on correctly with the resistor values:



So, the current plan is to use an AND gate of the PWM signal and the negated signal of the UART data, and that would serve as the data signal for the circuit above.

Then, I would have the other system's receiver close by, and the communication should work theoretically.

Additionally, I can also check the software by just connecting a wire between PE1 and PE0 (connecting UART7 TX to UART7 RX) to make sure the UART communication is working without the circuit. When I do this, it should just send the data back into the terminal, so I know I am transmitting the data correctly without the circuit.

Then, I could also check that the LED is turning on/off by looking at it through my phone camera, or what works even better, is replacing the IR LED with a regular blue LED, so that I know it is blinking properly while I am sending data.