

Electronics 1 Report

Stanley Vass – 2306978 – ACDB5002

Contents

Introduction.....	2
Background Research.....	2
Traffic Light Operation.....	2
Traffic Light Implementation.....	2
Endpoint Controller Implementation Discussion.....	2
Discrete Logic.....	2
Integrated Logic.....	3
Communication Protocols	3
RS485.....	3
RS232	3
I2C.....	4
SPI.....	4
Pulse coding	4
Produced Circuits.....	5
CMOS Logic Implementation.....	5
MCU Implementation.....	5
Pulse Coding scheme.....	5
Software Proofing Simulation.....	6
Explanation	6
Flow Charts.....	7
Testing	11
Conclusion	13
References.....	14

Introduction

This document details the design and testing of a traffic light junction controller (Master) and multiple Traffic Light endpoints (Slaves).

The master-slave control scheme was selected due to it making software programming and synchronization of the individual traffic lights significantly less cumbersome.

In implementing this overall topography multiple communication methods are discussed and compared, as well as multiple methods of decoding the communication data stream, eventually settling upon a novel pulse coding scheme.

Then follows information regarding the testing of the aforementioned system, and the problems encountered in attempting to simulate it.

Background Research

Traffic Light Operation

Traffic Lights in the United Kingdom cycle through four valid states, Red (Stop), Red and Yellow (Ready to go), Green (Go), Yellow (Stop if possible). [1]

Traffic Light Implementation

Traffic lights could be implemented a number of different ways some of which include:

Description	Pros	Cons
Central Control and Current Drive	- Extremely simple circuit - Easy to replace - Simple circuit maintenance - Practical EMC immunity	- Extreme cost in wiring - Might have to add more conduit under road to accommodate wire
Central Control, Remote Current Drive (Parallel)	- Relatively simple circuit - Cheaper maintenance from modularity	- High wiring cost - Extremely susceptible to EMC issues
Central Control, Remote Current Drive (Serial)	- Better EMC immunity - Lowest possible wiring cost - Cheaper maintenance from modularity	- More complex circuit - Not as EMC resilient as Central Control & Current Drive

Central Control, Remote Current Drive (Serial) was selected as it offers the best combination of benefits, it is the better compromise between EMC resilience, Maintenance, Wiring Cost, and Circuit complexity.

Endpoint Controller Implementation Discussion

There are broadly 2 main methods to implementing the logic required of the Endpoint Slave, that being either through discrete logic (ie SN7400 series, or CD4000 series) or large integrated ICs like a microcontroller or ASIC.

Discrete Logic

This method is generally more expensive and less power efficient, compared to a similarly capable monolithic IC.

However, this method does carry some significant advantages.

For instance, many monolithic ICs have a relatively short product lifespan, and are never replaced by a pin-compatible replacement. Whereas, many discrete logic ICs or pin-compatible variants have been in production for over 50 years. This fact means that if the circuit is ever in need of repair in the future, the likelihood of a replacement part being available is high, allowing for a cheaper and quicker repair of the device. [2]

The more modular nature of a discrete design also bestows benefits in terms of future adaptation of designs, as the individual parts can be reconfigured to perform different functions allowing for quicker and cheaper modifications in future. [2]

This type of implementation also carries the advantage of being less error prone i.e. a programming mistake in an MCU or other similar such device, could be left completely unseen and undetected, due to the inability to test every single possible input and processor state. Whereas with a discrete logic circuit, the number of possible states is generally significantly lower, allowing for a designer to practically test every possible state, and eliminate all possible erroneous conditions.

Integrated Logic

This method carries the advantage of being cheaper and quicker to implement, however, as described above it is more error prone and may not be replaceable in many years' time, requiring development time then to design a replacement board. As opposed to simply replacing the faulty part.

Communication Protocols

There are a wide variety of communication protocols and PHY layers that could be suitable for the purpose of relaying data between the controllers on this network, such as:

RS485

RS485 (official EIA-485) is a differential PHY layer, designed to be operated in a bus configuration. [3]

RS485 is an attractive option for such a system as the fact that it operates differentially gives it significantly improved common-mode noise immunity, something which due to the EMC environment is greatly desirable. However, the bus topography presented by RS485 does pose some degree of a downside as not only would it allow one fault on the bus to bring down every single light at once. It also requires either more wiring in the form of chip select lines for each Slave controller, or significantly more complexity in the receiver to decode an address from a packet on the bus.

If CS lines are not used some form of collision avoidance would be required as well, massively ballooning complexity to a ridiculous degree.

RS232

Is an asynchronous serial communication protocol, which does use only 2 signal lines, and is not a bus network, although it does come with the downside of not being a differential signal. [4]

While it being asynchronous is an advantage the fact that no clock signal is encoded on the transmission lines, makes any potential receiving hardware significantly more complex, as it has to generate its own synchronized clock.

I2C

I2C or Inter-Integrated-Circuit is a serial communication protocol, which uses a Serial Data Line (SDL) and a Serial Clock Line (SCL) to communicate between 2 ICs. [5]

However, as the name implies it is designed for communicating between ICs, not across streets, and as such it has not been designed with much noise immunity and I have personally had issues communicating with I2C over a distance around 30cm, let alone 10m across a street.

The short maximum range is a result of the protocols reliance on a pullup resistor which is pulled low by whichever device is initiating the transfer, this means that over meters the pullup resistor and drain transistors, need to move significant current to overcome the line capacitance, within the timing specification. [6]

The pullup resistor would likely have to be adjustable for each line as well, as different lengths of wire will exhibit different line capacitances.

While the DS28E17 [6] from Analog Devices seems like it could indeed solve the problems associated with I2C long distance communications, the lifespan of this product is unclear, and within the lifetime of a traffic light system deployment, the chip could become obsolete with no pin-compatible replacement. [2] Thereby making repair of the system impossible, necessitating a partial redesign and incurring significant cost.

SPI

Serial Peripheral Interface is a serial protocol which uses 3 wires, Master Out Slave In (MOSI), Master In Slave Out (MISO), and Serial Clock (SCLK), a chip select line is typical however is regularly omitted. [7]

In order to effectively utilise this serial protocol 3 twisted pairs would be required, in order to differentially encode and transmit the data.

This would increase the cost of installing the system to a large degree when compared with a protocol that only requires 1 twisted pair, due to the cost of the extra wire and the additional labour required to install it.

While this scheme would not require an especially complex endpoint interface circuit, the additional cost and labour imposed by the additional signal lines makes it not a particularly favourable option.

Pulse coding

A custom pulse coding scheme would allow the system to operate with minimal wiring and 2 twisted pairs per endpoint.

It also can be optimized to work in a star network configuration, reducing receiver complexity, only requiring a slight bit more complexity on the transmit side.

Because of windowing and pulse width restrictions, a pulse coding scheme can offer good noise immunity even when run single-ended.

It does come with the disadvantage however, in that it is somewhat complex to decode on the receiving end, though still less so than RS232.

Produced Circuits

Both of these circuits are incomplete, due to Proteus failing to simulate them properly, when adding more circuit elements.

CMOS Logic Implementation

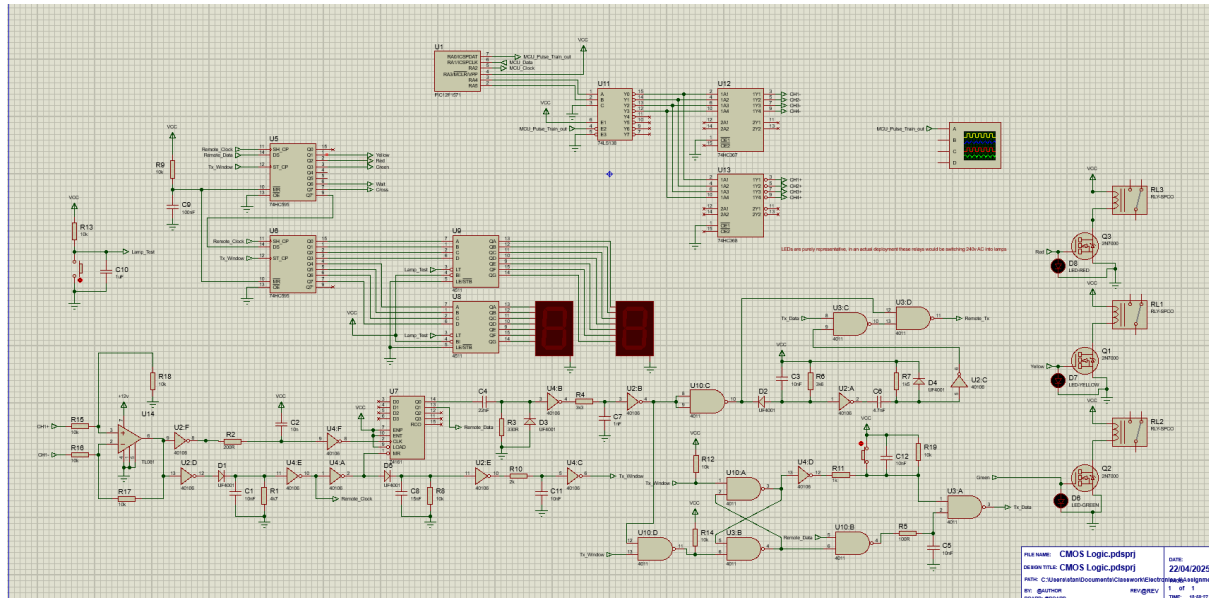


Figure 1

MCU Implementation

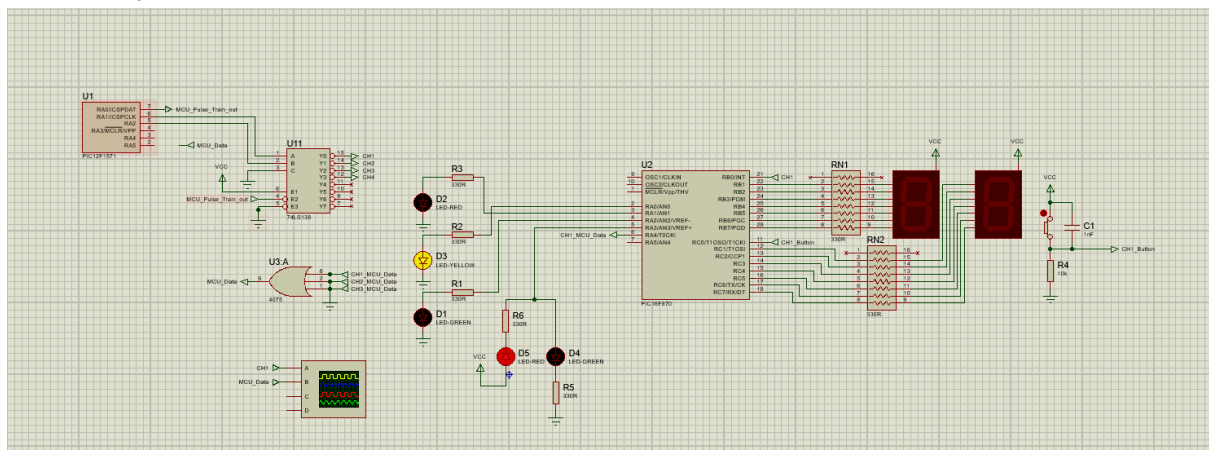


Figure 2

Pulse Coding scheme

The pulse coding scheme employed in this circuit operates by producing clock pulses with a period of approximately 110uS. These clock pulses are used to clock data into a serial to parallel shift register.

The period in between these pulses is called the bit period and if another pulse is produced within this period and within 30uS of the rising edge of the starting clock pulse the receiving circuit will register the bit period to have been equal to a one.

The slave controller will then delay and regenerate the clock pulses fed into, retransmitting the data it is currently receiving with the MSB of the retransmitted data replaced by the current state of the pedestrian crossing button. This allows the Master controller to both remain apprised of the pedestrian button's state, but also to compare the data received with that transmitted, allowing for it to detect and compensate for any transmission errors.

Software Proofing Simulation

The "Simulated CMOS" Proteus project is a project where the functionality of the pulse decoding circuit in Figure 1 has been replicated by a PIC16F870.

Explanation

The reason for doing this is that Proteus began experiencing glitches and errors with just 2 pulse decoding circuits, for instance the Tx_Data net pictured in Figure 1, would sit in an analogue state between Vcc and Ground, when adding unrelated circuit elements. Not to mention the circuit should not be capable of generating an analogue voltage on the node in question.

Not to mention the incredible degree of slowdown Proteus experienced, only being able to process a few ms in multiple minutes.

Flow Charts

The following are a series of flowcharts documenting the behaviour of the software.

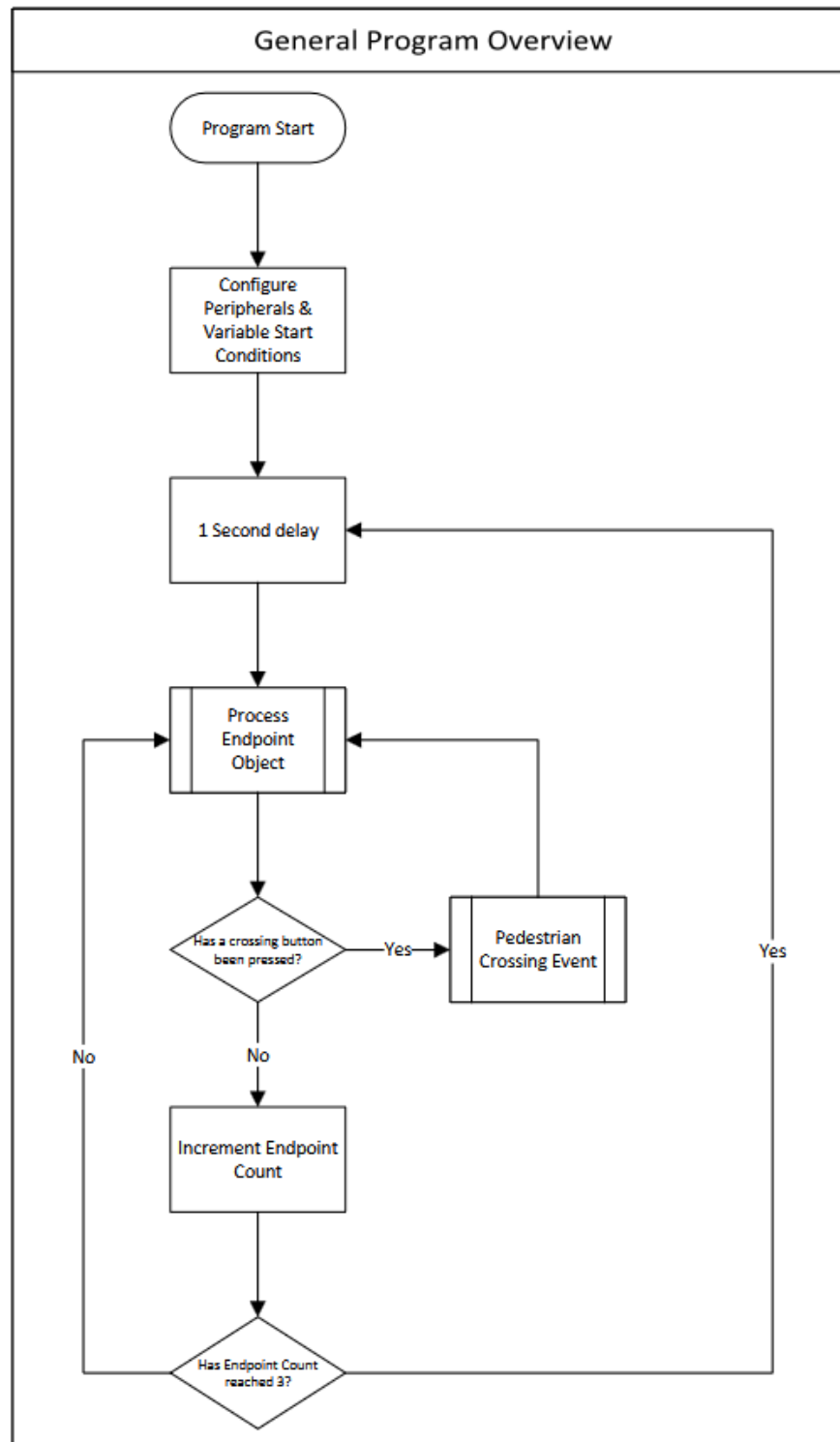


Figure 3

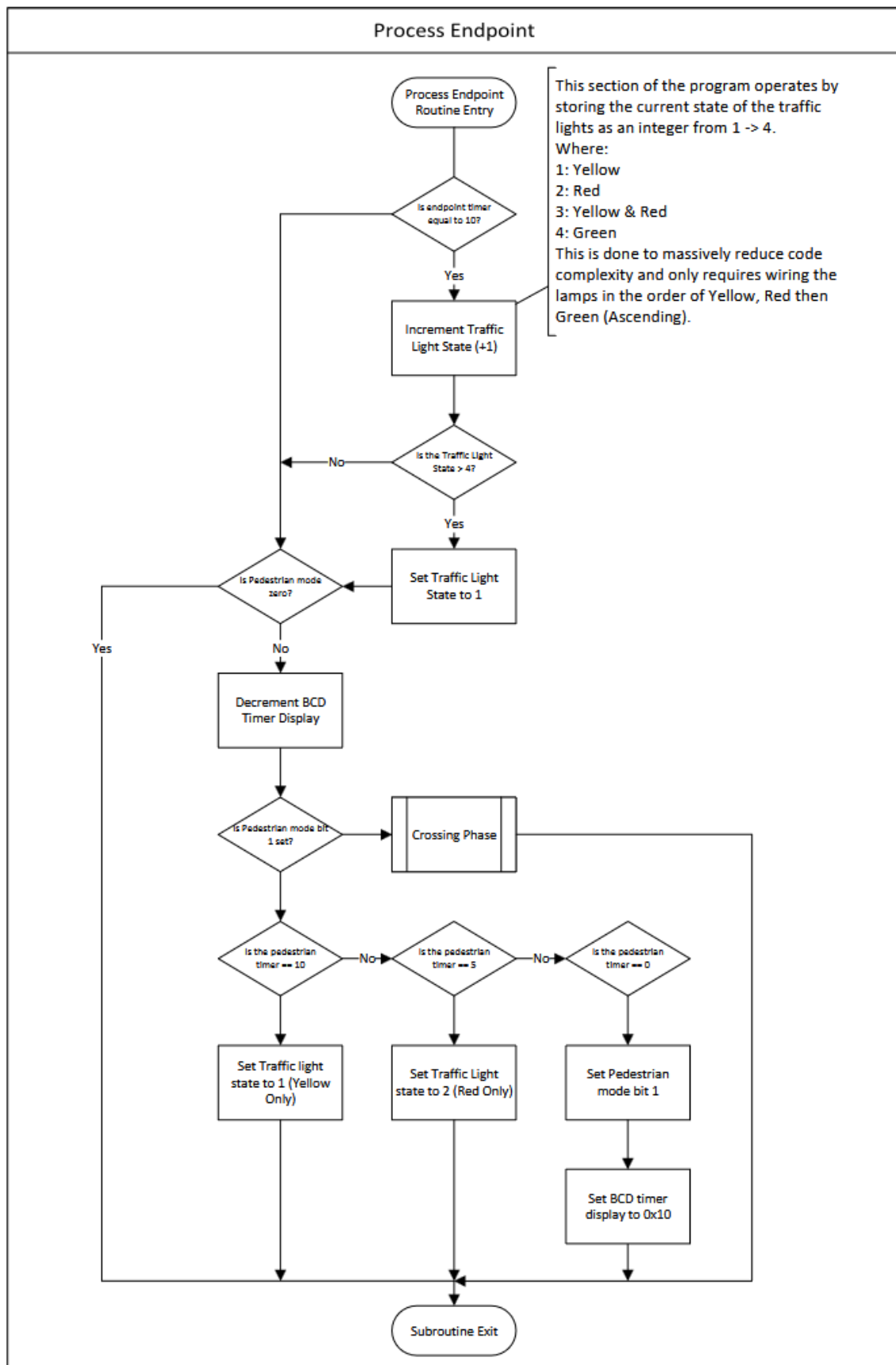


Figure 4

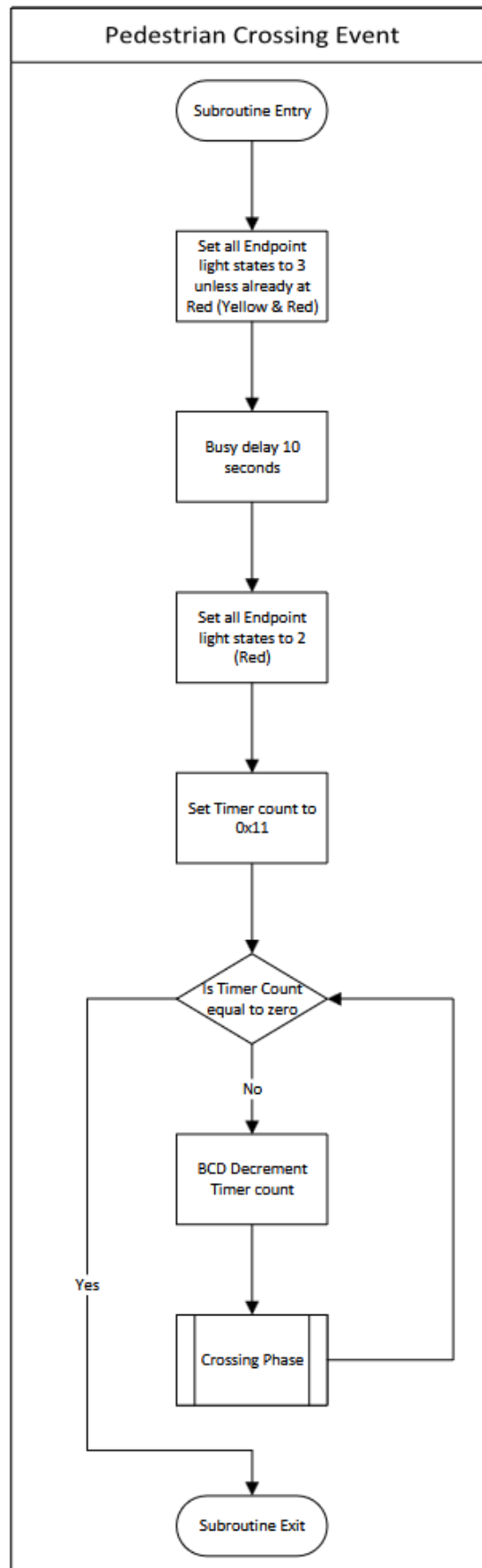


Figure 5

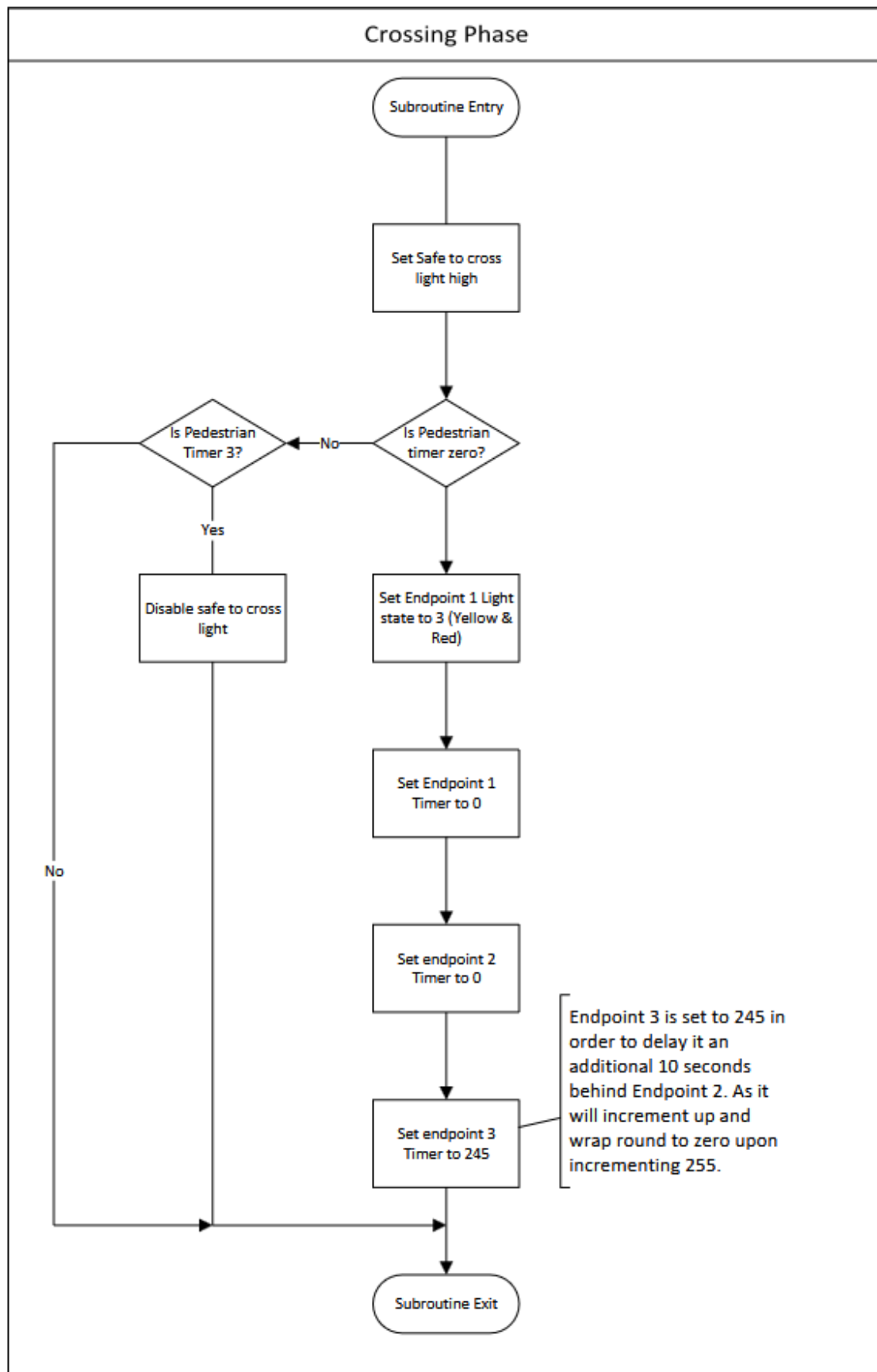


Figure 6

Testing

Testing was never completed due to running into issues with Proteus disagreeing with reality, but nonetheless both designs and simulations were developed to the point that the endpoints were capable of receiving and displaying data

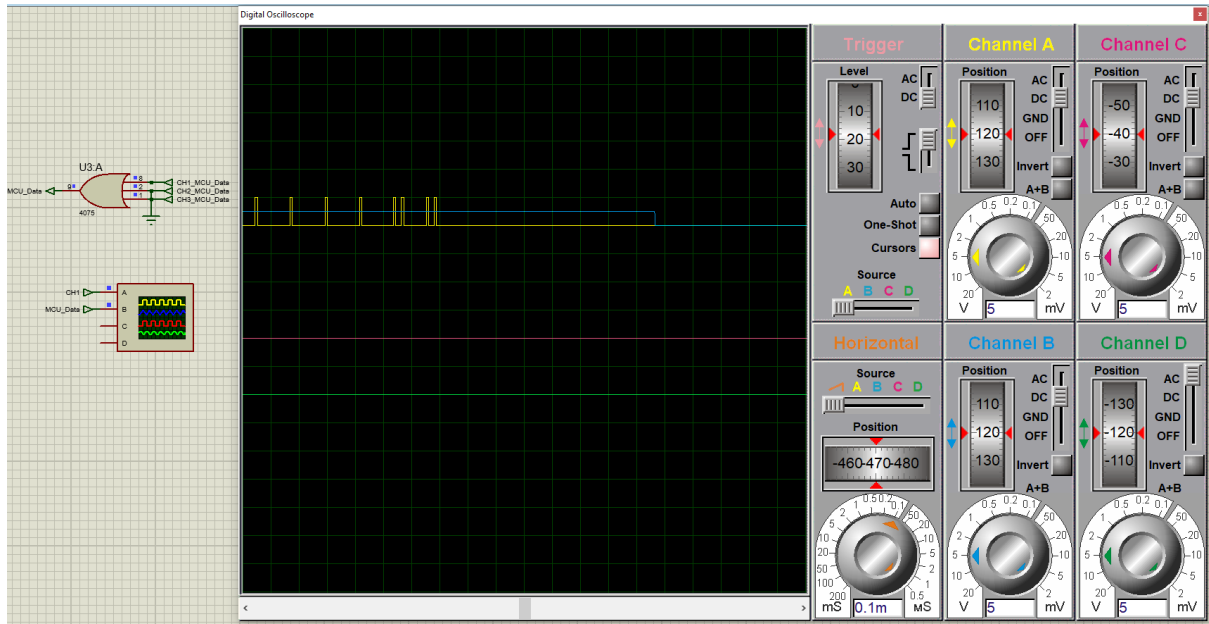


Figure 7

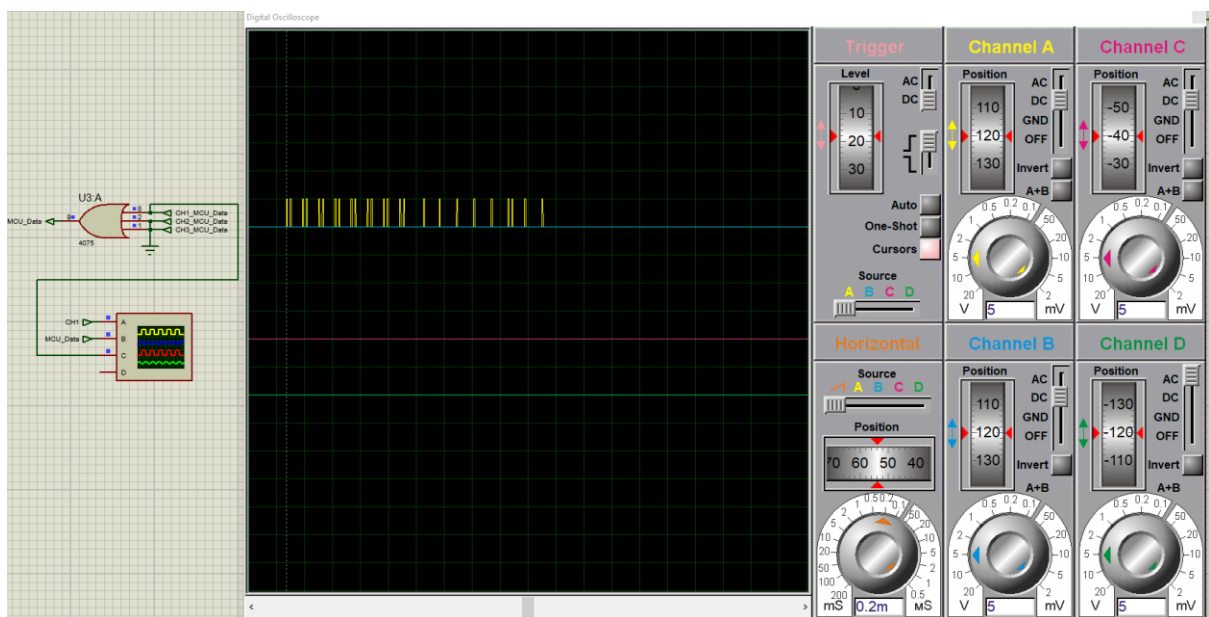


Figure 8

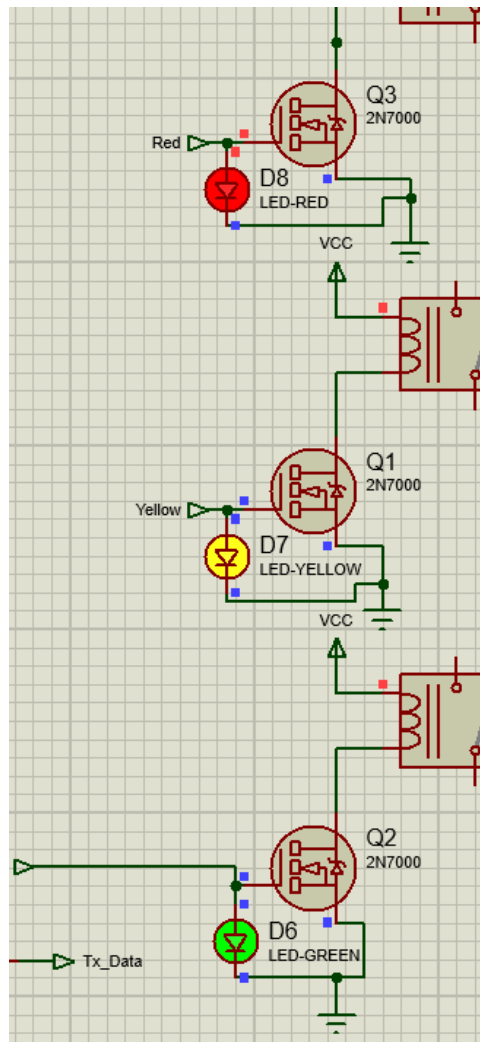


Figure 9

As can be seen from the above Proteus was not simulating either circuit correctly, these were not the only issues encountered just the ones which I could not figure out how to fix.

I instead decided to test the circuits to the extent which was possible in their current states.

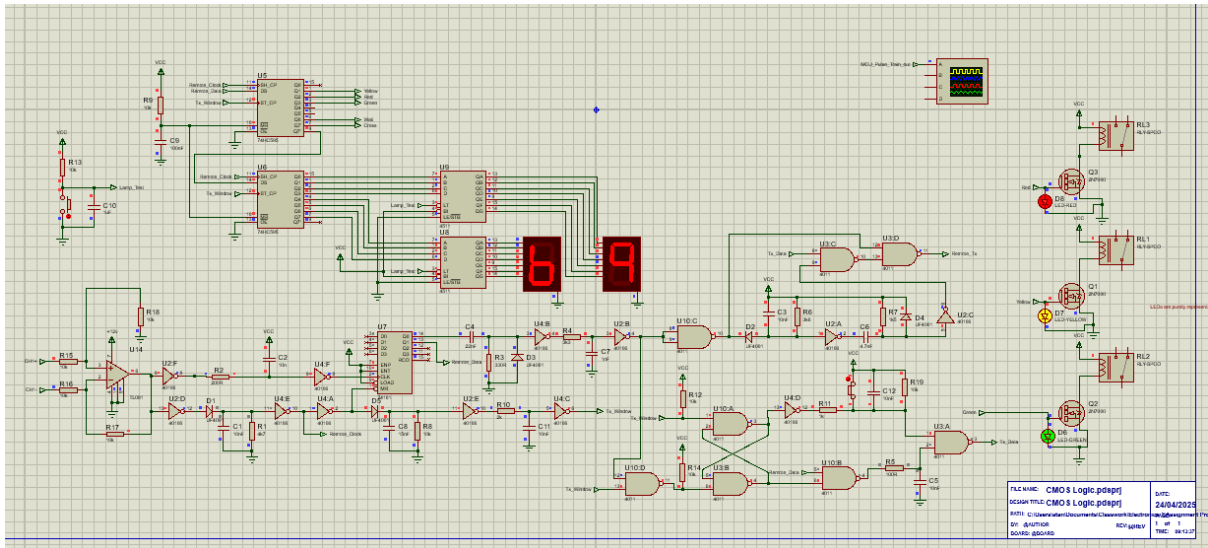


Figure 10

The CMOS logic implementation of the endpoint worked successfully interpreting the 16bit words sent to it and correctly loading them into the output register as seen in Figure 10. It also correctly retransmitted the button state back on the MSB of the retransmitted word back to the MCU.

Conclusion

In conclusion Proteus as a software suite does not provide a stable or accurate simulation of real-world electronic circuits.

References

- [1] UK Department for Transport, “Traffic Signs Manual,” 2019. [Online]. Available: <https://assets.publishing.service.gov.uk/media/5df0e29fed915d15f42c4820/dft-traffic-signs-manual-chapter-6.pdf>. [Accessed 24 04 2025].
- [2] D. Feucht, “Planet Analog,” 2 July 2013. [Online]. Available: <https://www.planetanalog.com/asics-vs-semi-discrete-design/>. [Accessed 24 April 2025].
- [3] Telecommunications Industry Association, “TIA/EIA-485,” March 1998. [Online]. Available: <https://www.mikrocontroller.net/attachment/428561/eia485.pdf>. [Accessed 24 April 2025].
- [4] ANSI, “ANSI EIA/TIA-232-E Interface Standard,” July 1991. [Online]. Available: http://www.bitsavers.org/pdf/datapro/communications_standards/2741_EIA_RS-232-E.pdf. [Accessed 24 April 2025].
- [5] NXP, “I2C-Bus Specification and User-Manual,” 1 October 2021. [Online]. Available: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>. [Accessed 24 April 2025].
- [6] Analog Devices, “Extending I2C Communication Distance with the DS28E17,” [Online]. Available: <https://www.analog.com/en/resources/design-notes/extending-i2c-communication-distance-with-the-ds28e17.html>. [Accessed 24 April 2025].
- [7] P. Dhake, “Introduction to SPI Interfac,” 9 September 2018. [Online]. Available: <https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html>. [Accessed 24 April 2025].