California State University, Northridge

Department of Electrical & Computer Engineering

ECE 526L - Digital Design With Verilog

Spring 2025

Final Project

Inter-Integrated Circuit Hardware

May 7, 2025

Cristian Robles

Professor Orod Haghighiara

Table of Contents

1. Academic Integrity Statement
2. Introduction
3. Theory
4. Methodology
5. Results
6. Conclusion
7. Acknowledgements

1. Academic Integrity Statement

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, nor have I allowed or will I allow anyone to copy my work.

Name (printed) Cristian Robles

Name (signed) Cristian Robles Date 2/6/25

2. Introduction

The objective of this final project was to design a custom motor control system using the inter-integrated circuit (I2C) communication protocol, implemented entirely in Verilog on the Zybo Z7-10 FPGA board. In the original robot system, a Raspberry Pi communicated with a DFRobot Romeo motor controller to drive motors. This project replaces the Romeo board with a hardware-only Verilog solution, enhancing customization and eliminating the need for embedded software on the motor controller. The Raspberry Pi functions as the I²C master, sending two-byte commands over the bus. The Zybo acts as the I²C slave, receiving these bytes, decoding them through a finite state machine (FSM), and producing a PWM output to drive a motor. This project demonstrates practical application of digital design, protocol decoding, and real-time control in an embedded hardware system.

3. Theory

A. I2C Circuit

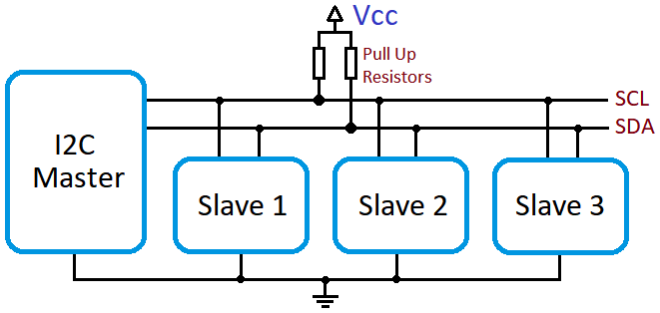

Figure 1 - Typical I2C Setup

The I2C configuration shown in Figure 1 illustrates a standard master slave setup. The master device communicates with multiple slave devices over a shared two wire interface, Serial Clock (SCL) and Serial Data (SDA). Each line is pulled high through pull-up resistors to ensure proper voltage levels during idle periods.

In this architecture only one slave is active at any given time which is determined by

address based selection started by the master. The bus acts sort of like a multiplexer selecting which slave listens or responds based on the address. This approach enables multiple devices to work on the same bus while minimizing the number of physical connections required.

B. I2C Data Packet

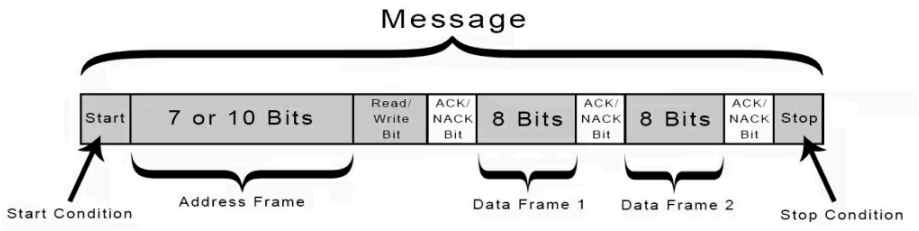

Figure 2 - I2C Data Packet Broken Down Into Parts

Figure 2 illustrates the structure of a standard I2C data packet. Each communication sequence begins with a start condition, where the master pulls SDA low while SCL remains high. This is followed by an address frame, consisting of 7 or 10 bits that identify the target slave, along with a single read/write bit that indicates the direction of data flow.

Once the slave is addressed, the master transmits one or more data frames, each composed of 8 data bits followed by an acknowledgment (ACK) or no acknowledgment (NACK) bit from the receiver. The presence of the ACK confirms that the receiving device has successfully captured the data. Multiple data frames can be chained together in a single transmission.

The transaction ends with a stop condition, where SDA transitions from low to high while SCL remains high. This structure ensures synchronized communication with acknowledgment at every stage.

C. Pulse Width Modulation

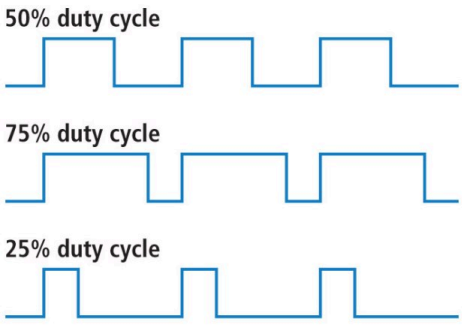

Figure 3 - Varying Pulse Width Modulated Signals

PWM controls power by adjusting how long a signal stays high within each cycle. As shown in Figure 3, a higher duty cycle means more power is delivered. For example, a 75% duty cycle keeps the signal high longer than a 25% duty cycle. In this project, the I2C slave converts received data into a PWM signal that sets motor speed based on duty cycle.

4. Methodology

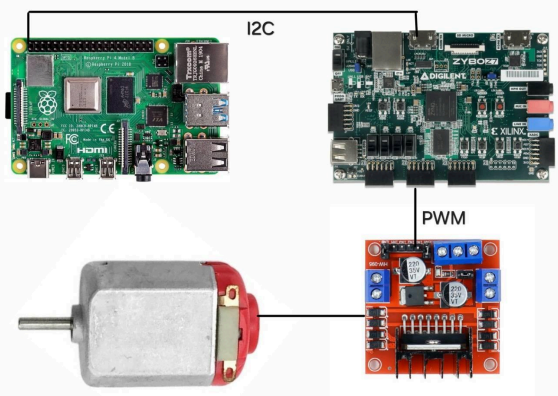

Figure 4 - Project Setup

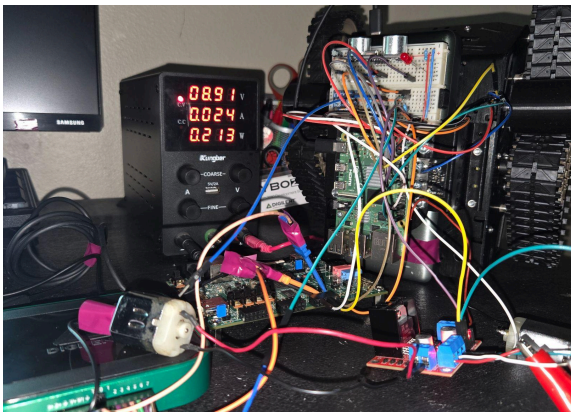

Figure 5 - Real Project Setup

The project was implemented using Verilog on the Zybo Z7-10 board. A Raspberry Pi running Ubuntu 22.04 acted as the I2C master and sent two bytes of data to the FPGA over the I2C bus. The FPGA, acting as an I2C slave, received the bytes using a custom finite state machine and passed the result to a PWM generator. The complete setup can be seen in Figure 4 and Figure 5.

The Verilog design was split into three main modules, I2CSlave.v which received and decoded I2C data using an 8-state FSM. PWMGen.v which converted the received value into a PWM signal, and top.v which connected both modules and exposed the necessary inputs and outputs.

Simulation was performed in Vivado to verify functional correctness. A custom .xdc constraints file was written to assign the correct FPGA pins to the SCL, SDA, and PWM signals. After successful simulation, the design was synthesized, implemented, and a bitstream was generated in Vivado. This bitstream was then loaded onto the Zybo board.

The design was tested on hardware using Digilent Analog Discovery to probe the I2C lines and verify the PWM output. The PWM signal was then connected to a motor controller to confirm that the motor responded correctly to the values sent by the Raspberry Pi.

5. Results

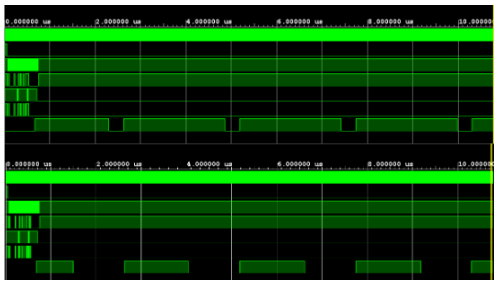

Figure 6 - Two Simulated I2C Transactions

Figure 6 shows two successful I2C transactions captured in simulation. Each transaction consists of a start condition, followed by address and data frames. The waveform confirms that the FPGA correctly receives the transmitted bytes and responds by generating a distinct PWM output for each case. The difference in PWM width between the two transactions verifies that the data was decoded properly and applied to the output. These results demonstrate correct end to end functionality from I2C input to PWM output.

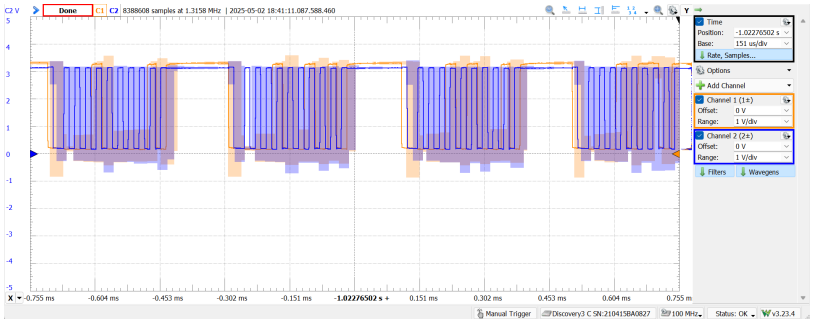

Figure 7 - Oscilloscope Capture of I2C Hardware

Figure 7 shows the oscilloscope capture of the i2cdetect command issued from the Raspberry Pi. The waveform confirms that the Pi is actively scanning for connected I2C devices, sending out address frames while monitoring for acknowledgment. This verifies that the hardware setup responds correctly at the electrical level and that the slave device is properly recognized on the bus.

Figure 8 - Oscilloscope Capture of PWM Wave

Figure 8 shows the oscilloscope capture of the PWM output generated by the FPGA. The waveform confirms the expected periodic signal, with a consistent high pulse followed by a low interval. The duty cycle corresponds to the value sent from the Raspberry Pi, validating that the I2C data was correctly decoded and translated into motor control. This confirms end-to-end functionality from command input to physical PWM output.

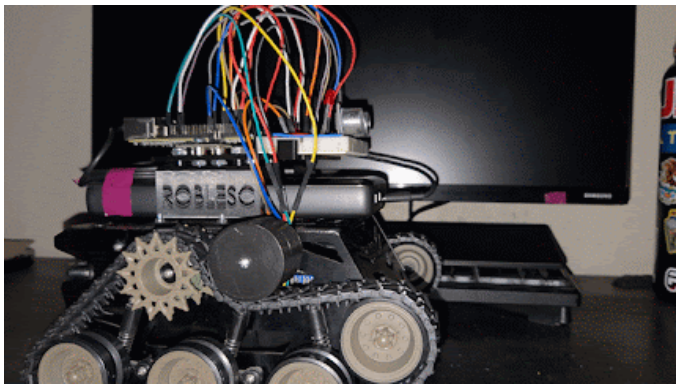

Figure 9 - Robot Running

Figure 9 shows the robot in operation using the custom I2C hardware setup. The FPGA receives commands from the Raspberry Pi and drives the motors through a PWM signal generated in Verilog. The movement confirms that the signal was correctly interpreted by the motor controller and that the full system from software command to physical actuation is functioning as intended.

6. Conclusion

This project successfully demonstrated a complete hardware implementation of an I2C-based motor control system using Verilog on the Zybo Z7-10 FPGA. The system received data from a Raspberry Pi I2C master, decoded it with a custom designed finite state machine, and converted the result into a PWM signal to control a motor. Simulation in Vivado verified functional correctness, while oscilloscope captures confirmed accurate I2C communication and PWM output on hardware. The robot's successful response to command changes validated the design. This project highlighted practical skills in protocol decoding, digital design, hardware integration, and embedded control.

7. Acknowledgements

I would like to thank Professor Saba Janamian for inspiring me to work on this robot and explore F Prime. Although hardware implementation was not required for this project, I chose to include it so I could continue developing and learning through hands-on work. I also want to thank everyone who has supported me throughout my robotics journey, as well as Professor Orod Haghighiara for being a great instructor and helping me strengthen my understanding of Verilog.