E2.07 – "404 Bot Not Found"
HDL FPGA Development

Product Specification

Luis Leon Pineda, Project Manager

Farrah Balbuena

Matthew Carter

Theodore Freij

Alexandra Nuez

Miguel Octavo Castro

Aaron Ong

Marie-Albert Sweeney

Sponsored by:

Mark W. Welker
Texas State University

601 University Dr.

San Marcos, 78666

Date: October 1, 2025

TEXAS STATE®
INGRAM SCHOOL OF ENGINEERING

Revision History Table		Template Date 6/15/2022
Version	Summary of Changes	Date
0.1	Revised Subsystem Roles for Luis, Aaron, Alex, Theodore, and Farrah in Executive Summary	10/15/2025
0.2	Assigned and completed ready for review	10/17/2025
0.3	Updated revisions for the following subsystems: ARM integration (C Coding) and Peripherals – SPI – ARM (C Coding)	10/20/2025
0.4	Revised wording and block diagram for testbenches for IPS and top-level diagram	11/11/2025
1.0	Document revision for D1 submission and updated section 8	12/1/2025
1.1	Name revision, updated I2C added features, and added Appendix B	1/28/2026
1.2	Executive Summary	2/4/2026
1.3	3.2 diagram changed, made sure owners were assigned to CH.4, added Appendix C and D.	2/16/2026
1.4	Updated Appendices and Testing	4/23/2026

Contents

Contents 2

1 Executive Summary 5

2 Top Level Block Diagram 6

3 Functional Description 7

3.1 IP Integration Subsystem 7

3.2 Testbenches for IP Blocks (HDL) Subsystem 10

3.3 ARM Integration (C Coding) Subsystem 12

3.4 ARM Validation (C Coding) Subsystem 14

3.5 PCB Design Peripheral Subsystem 17

3.6 Peripherals – SPI – ARM (C Coding) Subsystem 19

3.7 Peripherals – I2C – ARM (C Coding) Subsystem 20

3.8 Peripherals – PWM – ARM (C Coding) Subsystem 21

3.9 Safety, Security and Administrative Functions 23

4 Performance 24

4.1 Boundary Conditions and Constraints 26

5 Project Alignment Matrix 27

6 References 29

7 Approvals....29
7.1 Contingencies....30
8 Appendix A: Test Procedures....31
8.1 IP Integration Subsystem Unit Tests....32
8.1.1 XSA Generation....32
8.1.2 Performance Testing for Fabric Clock....33
8.2 Testbenches for IP Blocks (HDL) Subsystem Unit Tests....34
8.2.1 Testbench IP Validation GPIO (Simulation)....34
8.2.2 Testbench IP Validation SPI (Simulation)....35
8.2.3 Testbench IP Validation I2C (Simulation)....36
8.2.4 Testbench IP Validation PWM (Simulation)....37
8.3 ARM Integration (C Coding) Subsystem Unit Test....38
8.3.1 ARM Integration....38
8.4 ARM Validation (C Coding) Subsystem Unit Tests....39
8.4.1 Validation IP Blocks....39
8.5 PCB Design Peripheral Subsystem Unit Tests....40
8.5.1 I2C Integration on PCB with OLED Screens....40
8.5.2 SPI Integration on PCB with the LSM9DS1 Inertial Module....41
8.5.3 RGB LED Control with PWM on PCB....42
8.5.4 H-Bridge Motor Driver on PCB with the SN754410NE....43
8.6 Peripherals – SPI – ARM (C Coding) Subsystem Unit Tests....44
8.6.1 SPI ARM Validation and Peripheral Integration....44
8.7 Peripherals – I2C – ARM (C Coding) Subsystem Unit Tests....45
8.7.1 OLED I2C Screen Integration....45
8.8 Peripherals – PWM – ARM (C Coding) Subsystem Unit Tests....46
8.8.1 PWM Cora Z7 Test....46
8.9 Overall System Integration Unit Tests....47
8.9.1 Inertial Module to OLED Display on PCB Test on PCB....47

8.9.2 Full Peripheral Integration on PCB with Switching Modes 48
9 HDL FPGA Operation Procedure 49
9.1 IP Integration Subsystem 49
9.2 Testbenches for IP Blocks Subsystem 49
9.3 ARM Integration (C Coding) Subsystem 49
9.4 ARM Validation (C Coding) Subsystem 49
9.5 PCB Design Peripherals Subsystem 49
9.6 Peripherals – SPI – ARM (C Coding) Subsystem 49
9.7 Peripherals – I2C – ARM (C Coding) Subsystem 49
9.8 Peripherals – PWM – ARM (C Coding) Subsystem 49
10 Appendix B: Autonomous Bot Executive Summary 50
11 Appendix C: Autonomous Bot Top-Level Diagram 51
12 Appendix D: Functional Description 52
12.1 Movement Control Subsystem 52
12.2 Chassis Design Subsystem 53

1 Executive Summary

The Executive Summary was written by Farrah Balbuena.

Our product is a low-cost FPGA development board that uses Hardware Design Language (HDL) to implement and develop IP blocks for the Cora Z7, using HDL to improve the educational environment. By creating this product, future curricula for EE 4321, the "Digital Systems using HDL" course, will allow students to not only develop their skills in software programming but also apply their learning to real-world applications using the Cora Z7.

The need for this product stems from the proven benefits of experiential learning in engineering education. According to research from the National Training Laboratories in Bethel, Maine, students retain up to 75% more information when engaged in hands-on, "practice-by-doing" activities compared to passive learning methods.[1] By incorporating the CORA Z7 into the curriculum, educators can enhance student comprehension and retention of HDL principles. This approach not only improves academic outcomes but also prepares students for careers in embedded systems, hardware design, and digital logic—fields that demand both theoretical knowledge and practical proficiency.

The team will research the capabilities of the Cora Z7 and implement their findings using Xilinx applications. Development will include creating block diagrams in Vivado [1] to integrate IP protocols and peripherals such as SPI, I2C, PWM, and GPIO. Testbenches will be designed to simulate and validate functionality. All project work will be conducted at Texas State University. By the end of Design Phase 1 (D1), the team will have a functioning SPI interface. By the end of Design Phase 2 (D2), the team will have functioning IP blocks and test benches for the FPGA, a fully documented ARM code demonstrating functional peripherals, a completed PCB, and a fully functional educational board. As a second goal, the team aims to begin transforming the board into an HDL-programmed autonomous bot.

Team Member	Subsystem
Aaron Ong	Peripherals – PWM – ARM (C Coding)
Luis Leon Pineda	Master XSA Iterations
Farrah Balbuena	Testbenches for IPS (HDL)
Marie-Albert Sweeney	PCB Design Peripherals
Theodore Freij	ARM Validation for GPIOs/PWM (C Coding) & H-Bridge
Matthew Carter	Peripherals – SPI – ARM (C Coding)
Miguel Octavo Castro	Peripherals – I2C – ARM (C Coding)
Alex Nuez	ARM Integration (C Coding)

Table 1: Preliminary Subsystem Responsibilities

[1] https://www.arlo.co/blog/overview-of-the-learning-pyramid-for-training-providers accessed 9/17/2025

2 Top Level Block Diagram

The top-level block diagram was drawn by Alexandra Nuez

The top-level diagram of our system is shown in Figure 1. This diagram illustrates the complete integration of the hardware and its peripherals implemented on the Cora Z7. The procedure begins with the HDL modules and test benches, defining and verifying hardware functionality before being mapped into memory and connected to the ARM processor through AXI interconnect. The ARM core is responsible for handling communication with peripherals such as SPI, I2C, GPIO, and PWM which is implemented through programmable logic and interfaced with external PCB components. Inputs consist of sensor data from the LSM9DS1 and button signals which are processed by the ARM processor. This generates the output controlling the OLED display, LEDs, and DC motors using the H-Bridge. The design is divided among eight team members who are responsible for specific subsystems listed in Figure 1. With each responsibility, the whole team ensures reliable coordination for hardware development, testing, and integration across both the FPGA and PCB components for a complete and functional system.

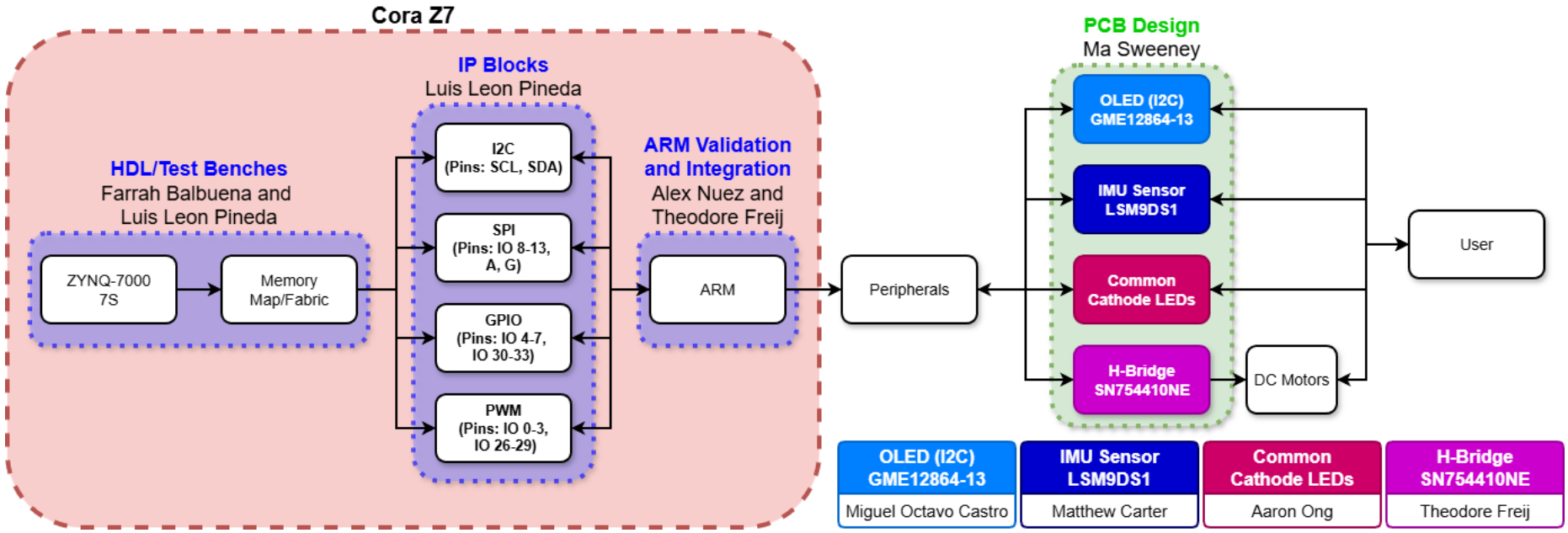


Figure 1: Top-Level Block Diagram

3 Functional Description

3.1 IP Integration Subsystem

The IP Integration Subsystem is owned by Luis Leon Pineda

As shown in Figure 2, the block design of IPs was utilized to generate the XSA for the Digilent Cora Z7 board.

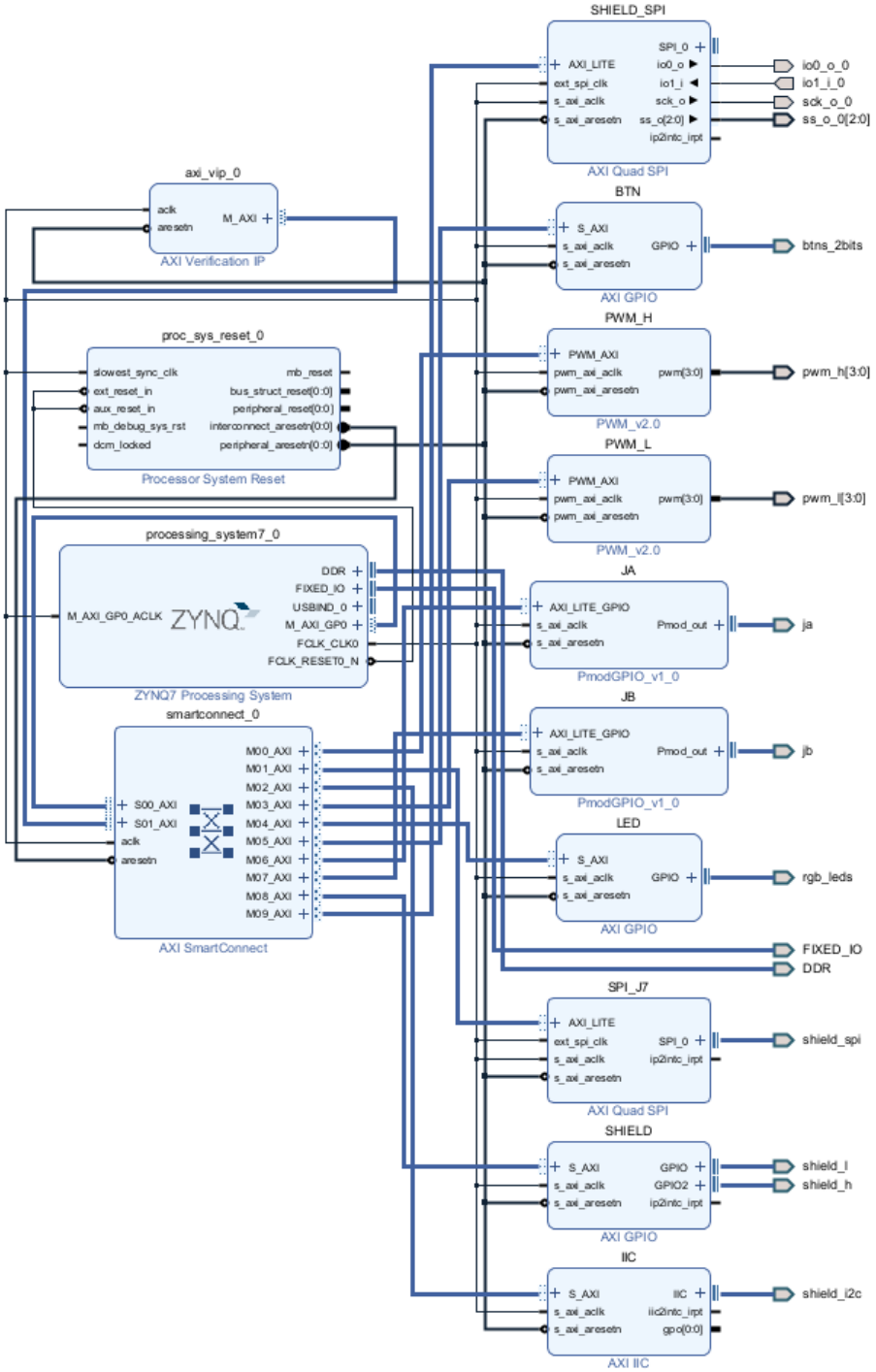


Figure 2: Block Level Diagram of the XSA

The following tables describe the power input and outputs provided by the Cora Z7; these are standard pin locations and outputs designated by the board manufacturer.

Input	Source	Min	Max	Description
5V, 1A DC	USB	4.75V DC	5.25 V DC	Power supplied to the Digilent Cora Z7 via USB OTG Port.
5V, 4A DC	Barrel Jack	4.75V DC	5.25V DC	Power supplied needed for demanding projects through the board Barrel Jack J15

Table 2: Power supply for the CoraZ7

Output	Destination	Min	Max	Description
3.3VDC 100mA	GPIO PMOD JA JB	0VDC	3.3V DC	PMOD JA JB
3.3 VDC	Shield Pins	0VDC	3.3V DC	GPIO Shield Pins IO0-41

Table 3: Expected Output voltage on IO pins

The master XSA subsystem is responsible for connecting the IPs required for interfacing with peripherals by utilizing Digilent and Xilinx AMD IP blocks. The connection paths are demonstrated in Figure 2. In this procedure, the Zynq7 Processing System IP is introduced to the system to match the physical components on the Cora Z7 board.

In the configuration of the Zynq7, the clock speeds of the Zynq7, ARM processor, DDR [2], and Fabric-enabled clocks are defined; other clocks are present in the configuration but are left to factory settings defined by the IP block.

Clock Location	Frequency (MHZ)	Min	Max	Description
CPU (ARM)	650.00	50.00	667.00	ARM Processor speed
DDR	525.00	200.00	534.00	DDR Memory Speed
FCLK_CLK0	50.00	.100	250.00	Fabric Clock for IP Blocks
SPI Protocol	6.25	6.25	10.00	The Fabric Clock provides a base speed SPI module that uses a divider to match speeds to peripherals.
I2C	.100	.001	1.0	Driven By IIC IP Block

Table 4: System Clock signals

The processor generates the following inputs and outputs, as described in the following table.

Name	Input/Output	Origin	Destination	Description
DDR	Output	Zynq7	Cora Z7 DDR	Connects the fabric to the DDR memory
FIXED_IO	Output	Zynq7	Fixed Ios	Zynq required IO(Input/Output)
M_AXI_GP0	Output	Zynq7	Axi_SmartConnect	Master AXI bus from Zynq 7
FCLK_CLK0	Output	Zynq7	All IP Blocks	Fabric-generated clock (50MHz)
FCKL_RESET0_N	Output	Zynq7	Processor System Reset	Fabric reset signal for the processor and all the peripherals
M_AXI_GP0_ACLK	Input	Zynq7	Zynq7	Internal clock signal check

Table 5: IOs from the Zynq Processing System

A Processor System Reset IP is added to control the reset signals of all interconnects and other IP blocks. This block uses the FCKL_CLK0 and FCKL_RESET0_N as inputs and produces two output signals: interconnect and peripheral asynchronous resets.

The following block is the AXI SmartConnect. This is required to create an internal communication protocol from the ARM to the fabric IP blocks. The inputs for this IP block are the S00_AXI (slave input from the Zynq7 M_AXI_GP0), FCLK_CLK0, and aresetn. This produces 10 independent MXX_AXI masters to communicate with each independent IP block.

The following IP blocks all receive the following signals s_axi_aclk (FCLK_CLK0), s_axi_areset (peripheral_aresetn). Their connections and protocol are explained in the following table.

IP Name	Protocol	Outputs	Base Address	Description
PMOD JA	GPIO [3]	JA	0x4000_0000	PMOD JA GPIO
PMOD JB	GPIO [3]	JB	0x4010_0000	PMOD JB GPIO
LED	GPIO [4]	LD0, LD1	0x4100_0000	Cora Z7 board-mounted RGB LED
BTN	GPIO [4]	BTN0, BTN1	0x4120_0000	Cora Z7 board-mounted buttons
SHIELD	GPIO [4]	IO4-IO7, IO26-IO41	0x4200_0000	Cora Z7 J4 and J5 Pin Headers
IIC	I2C [6]	Shield I2C J3	0x4300_0000	Shield I2C J3 pins SCL, SDA.
SPI	SPI [5]	J7	0x4400_0000	Cora Z7 J7 SPI header
SHIELD_SPI	SPI [5]	IO8-IO13	0x4401_0000	Second SPI bus
PWM_L	PWM [3]	IO0-IO3	0x4600_0000	PWM header pins.
PWM_H	PWM[3]	IO23-IO2	0x4601_0000	PWM header pins

Table 6: Memory Mapping IP Addresses

3.2 Testbenches for IP Blocks (HDL) Subsystem

The Testbenches for IP Blocks (HDL) Subsystem is owned by Farrah Balbuena.

As shown in Figure 3, the code flowchart illustrates the procedural steps for iterating the testbenches in Vivado. The testbenches will implement PWM, SPI, I2C, and GPIOs (RGB LEDs, Buttons, PMOD A, and PMOD B) and display both the inputs and outputs of the Cora Z7. Vivado will emulate the real-world behaviours of the Cora Z7 through virtual simulation.

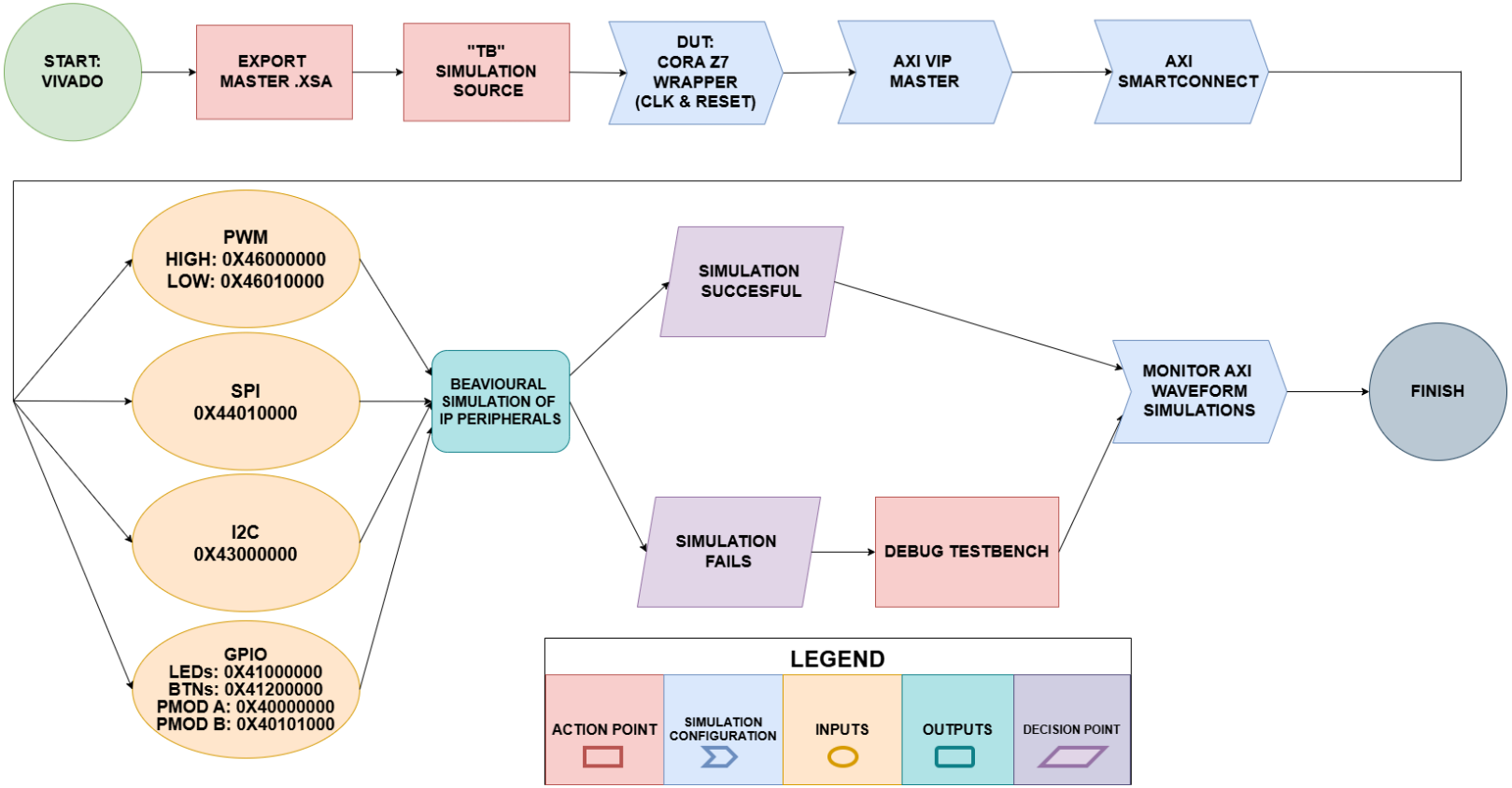


Figure 3: Code Flowchart of Testbenches for IPS (HDL).

The Hardware Design Language (HDL) testbenches for the Integrated Peripheral (IP) Blocks are a critical part of the simulation and validation process for the Cora Z7 hardware design. After the HDL modules are integrated into a block design and the bitstream is successfully generated, Vivado exports the design as an XSA file. This wrapper file encapsulates the hardware configuration and serves as the top-level module for simulation. In the testbench design module, the XSA wrapper is instantiated and connected to all simulated interfaces (either inputs or outputs), including:

- **Pulse Width Modulation (PWM)** (for motor speed/control).
- **Serial Peripheral Interface (SPI)** (for external device communication).
- **Inter-Integrated Circuit (I2C)** (for external device communication).
- All relevant **General-Purpose Input/Output (GPIO)** signals (Buttons, LEDs, and the configurable PMOD A/B pins).

The testbench includes clock generation, reset logic, and tailored input sequences to replicate real-world conditions. Alongside, the testbench simulation source will be accessed through the AXI Verification IP (AXI VIP), which acts as a Master interface. AXI VIP generates realistic AXI transactions, performs protocol compliance checks, and injects configurable stimuli such as burst transfers and error conditions. This ensures that all AXI-based communication paths are validated linearly to hardware deployment.

The AXI VIP communicates directly with the AXI SmartConnect, which serves as the interconnect fabric between the Zynq Processing System and multiple AXI peripherals. SmartConnect dynamically routes transactions, manages address decoding, and handles clock domain crossing between the PS and PL. It also optimizes throughout using pipelining and arbitration, ensuring low-latency communication for peripherals such as SPI, I2C, PWM, and GPIOs.

Simulation is run within Vivado to analyse the system's behaviour, verify timing, and detect logic errors. This process ensures that the HDL logic performs as expected before deployment to physical hardware. If simulation results indicate issues, the design debugged, iterated, and re-tested. The behavioural simulations are going to thoroughly verify the operation of each peripheral. The testbench workflow provides a structured and repeatable method for validating the Cora Z7 wrapper, aligning with the overall design flow from block diagram creation to simulation completion. The structured testbenches for IPS will be the final validation for the Hardware Design Language (HDL) portion of the project and allow students to verify their own testbenches within the EE 4321, the "Digital Systems using HDL" course.

As mentioned previously, the configuration of the Zynq7, the clock speeds of the Fabric-enabled clocks are defined; other clocks are present in the configuration but are left to factory settings defined by the IP block.

Clock Location	Frequency (MHZ)	Min	Max	Description
FCLK_CLK0	50.00	.100	250.00	Fabric Processor for IP Blocks

Table 7: Simulation Clock Signals to be used

3.3 ARM Integration (C Coding) Subsystem

The ARM Integration (C Coding) Subsystem is owned by Alexandra Nuez

The ARM Integration subsystem is the main control system that connects the ARM Cortex-A9 processor on the Zynq-7000 SoC and the external peripherals such as SPI, I2C, PWM, and GPIO interfaces. This subsystem allows the software and the hardware to efficiently work together, providing reliable communication across all functional IP blocks of the Cora Z7.

This subsystem is responsible for utilizing the validation of all peripherals to integrate all software and hardware peripherals used in the project to ensure they operate synchronously and efficiently through memory-mapped AXI parameters and interrupt-driven event handling. Its functions include:

- **System initialization:** Configures the processing system, establishing necessary clock and interrupt settings and enabling communication with programmable logic. Memory-mapped I/O modules for the SPI, I2C, PWM, and GPIO peripherals are defined in the Vivado hardware design and registered by the software to allow consistent access to the device.
- **GPIO Interface:** Provides digital input and output between the ARM core and external hardware. Button states are read through the GPIO input channel, while LEDs and other status indicators are controlled through the output channel. The processor either prompts or responds to interrupts from the GPIO to execute the corresponding control logic.
- **PWM Controller:** Generates variable-duty pulse signals for analog-like controls such as LED dimming or motor control. The ARM writes period and width values to PWM registers to enable flexible duty-cycle adjustment and smooth transitions.
- **SPI Communication:** Enables high-speed data exchange with external sensors. The ARM configures SPI mode, clock polarity, and data length to ensure connected components are compatible. Data being exchanged occur under software control or interrupt-driven sequences.
- **I2C Interface:** Provides serial, synchronous communication with integrated peripherals such as magnetometers, accelerometers, gyroscopes, displays, and motor configurations using an H-Bridge. The ARM manages the address mapping of I2C, timing parameters, and read/write operations using standard protocol sequences.
- **PuTTY Feedback:** Supports debugging and user interactions through a serial terminal. System status, sensor values, and debug messages will be transmitted to the PuTTY interface using "xil_printf()" calls.

As seen in Figure 4, the block diagram demonstrates how the Processing System (PS) communicates with the programmable hardware blocks in the Processing Logic (PL) through AXI interconnect. The ARM processor is responsible for high-level control and software tasks, while PL handles peripherals such as GPIO, PWM, SPI, and I2C that connects to external devices like LEDs, buttons, sensors, and external devices. Together, this subsystem enables efficient communication between software running on the ARM core and hardware logic implemented in the FPGA fabric. Figure 5 visualizes how the system cycles through main peripheral function modes which includes color selection, brightness adjustment, text display, motor control, and SPI sensor output using the on-board buttons.

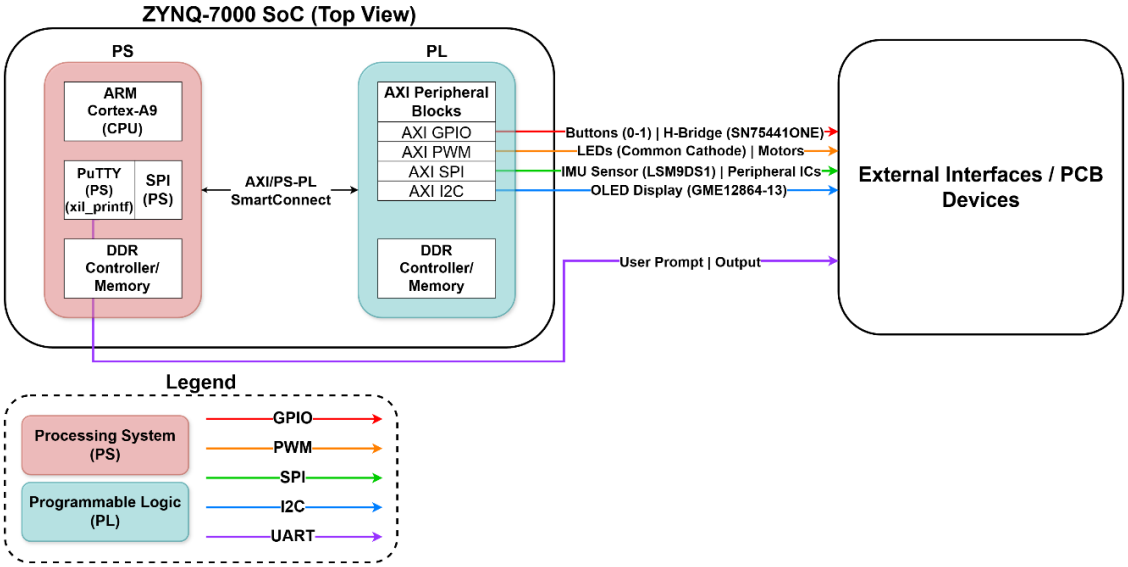


Figure 4: Top-level block diagram for ARM integration, showing how FPGA communicates with peripherals.

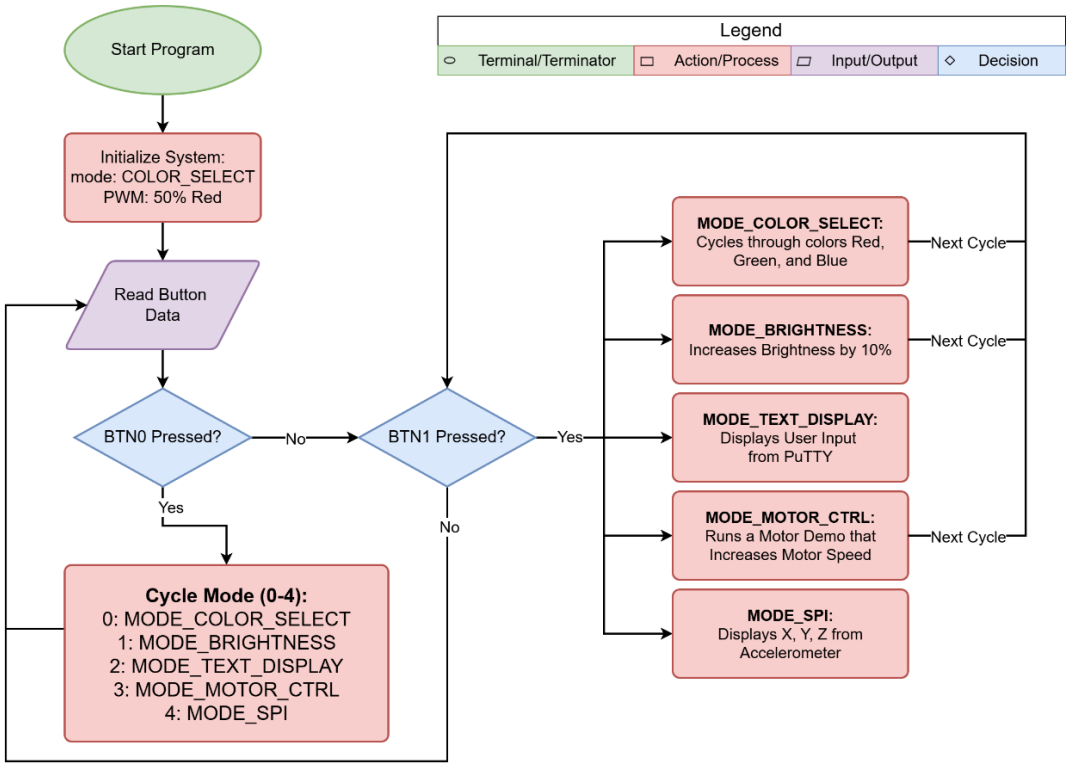


Figure 5: High-level flowchart of user-control design using on-board buttons to cycle peripheral functions and their options.

3.4 ARM Validation (C Coding) Subsystem

The ARM Validation (C Coding) Subsystem is owned by Theodore Freij

The ARM validation subsystem demonstrates and verifies the ability of the Zynq-7000 ARM processor to communicate with the GPIO peripherals and the external H-Bridge motor driver. This subsystem properly validates proper operations of the GPIO cores connected to the LEDs, buttons, and PMOD connectors alongside the I2C interface used to control the Adafruit Motor Shield v2.3 [7]. The goal is to confirm full end-to-end communication between the ARM processor, programmable logic, and external equipment connected.

Input	Source	Min	Max	Description
3.3 VDC	Cora Z7 Onboard Supply	3.1 V	3.5 V	Logic-level voltage powering the PS/PL GPIO, LEDs, and buttons.
SDA, SCL	ARM AXI IIC @ 0x43000000	Logic Low 0 V	Logic High 3.3 V	I2C lines carrying control commands to the PCA9685 PWM controller on the H-Bridge shield.
BTN [1:0]	Onboard Pushbuttons	Logic Low 0 V	Logic High 3.3 V	User inputs for triggering LED toggles and motor control validation sequences.
External 5-18V DC	Variable Adapter Input	5 V	18 V	Motor supply voltage for powering the H-Bridge drivers on the Adafruit Shield.

Table 8: Inputs supplied to the Cora z7

Output	Destination Hardware	Min	Max	Description
LED (GPIO Writes)	On-board RGB LEDs (AXI GPIO @ 0x41000000)	0 V	3.3 V	Visual confirmation that GPIO base address and register writes function correctly through ARM.
SDA / SCL (I^2C Bus)	OLED Display via AXI I^2C (0x43000000)	0 V	3.3 V	I^2C traffic used to validate correct addressing, ACK responses, and command/data transfers to the OLED module.
SPI Signals (MOSI / MISO / SCK / SS)	J7 SPI Header via AXI Quad-SPI (0x44000000)	0 V	3.3 V	SPI signaling confirming proper control-register configuration, FIFO operation, and full-duplex transfers.
PWM Outputs	External RGB LED via PWM_L / PWM_H	0 %	100 % Duty Cycle	PWM period and duty-cycle offsets verified by observing brightness/color changes on external LED.
UART TX/RX	UART0 Console (0xE0000000)	0 V	3.3 V	Serial logging interface used to print register values, test results, and debug messages during validation.

Table 9: Outputs generated by the Cora Z7

Subsystem Description

The ARM validation subsystem confirms that the Cora Z7's Processing System can correctly access and control all assigned AXI peripherals. This includes GPIO, I2C, SPI, and PWM modules mapped into the ARM's memory space. Validation ensures that each peripheral responds at its documented base address, that register offsets behave as expected, and that hardware outputs reflect software-controlled changes.

Confirmations on the ARM

- Verified full GPIO read/write functionality using on-board LEDs and buttons through AXI GPIO.
- Confirmed I2C communication at 0x43000000 by sending command/data frames to the OLED display and receiving proper ACK responses.
- Validated SPI operation through the J7 SPI header, confirming proper control-register setup and full-duplex transfers over MOSI/MISO.
- Confirmed PWM generation on PWM_L and PWM_H, validating enable bits, period registers, and duty-cycle offsets by observing brightness and color shifts on an external RGB LED.

GPIO Validation

- LEDs Base Address: 0x41000000
- Buttons Base Address: 0x41200000
- LEDs provided visual confirmation of correct register writes, and button reads validated ARM to AXI to GPIO input routing.

I2C Validation

- Base Address: 0x43000000
- Validated proper I2C timing, START/STOP control, ACK responses, and data transfers to the OLED display.
- Confirmed successful rendering of characters sent from UART input, proving the I2C controller and slave addressing operate correctly under the ARM.

SPI Validation

- J7 SPI Base Address: 0x44000000
- Verified SPI control-register configuration (CPOL/CPHA, enable, master mode) and successful MOSI/MISO transfers.
- Confirmed correct behavior and FIFO operation by reading back known responses from SPI test devices.

PWM Validation

- PWM_L Base Address: 0x46000000
- PWM_H Base Address: 0x46010000
- Validated PWM enable, period, and duty-cycle registers.
- Observed real brightness and color changes on an external RGB LED, confirming correct PWM timing and register mapping.

PMOD GPIO Validation

- PMOD JA/JB pins mapped to unused GPIO addresses for discretionary I/O expansion.
- Toggled and monitored with a logic analyzer to confirm pin routing, output-enable behavior, and expected transitions.

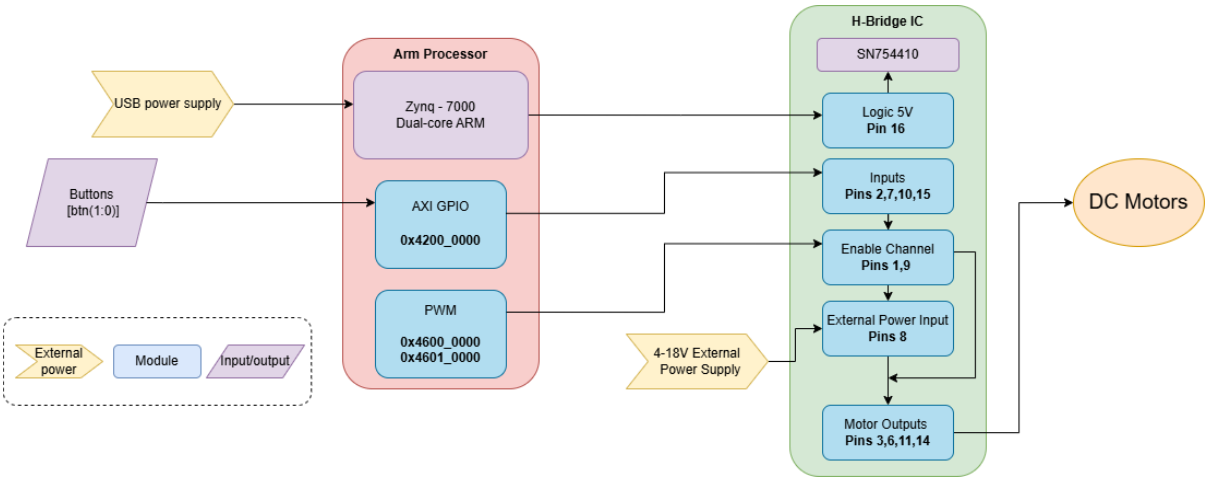


Figure 6: Block Diagram for the implementation of the H-Bridge and GPIO validation.

3.5 PCB Design Peripheral Subsystem

The PCB Design Peripheral Subsystem is owned by Marie-Albert Sweeney.

The printed circuit board (PCB) subsystem is a custom circuit board that connects to the Arduino shield headers on the Cora Z7. Figure 7 provides a block diagram of the connections from the Cora Z7 board, through the PCB, to the peripheral components. The board contains the LSM9DS1 inertial module, two OLED screens, two RGB LEDs, and the SN754410NE H-Bridge motor driver.

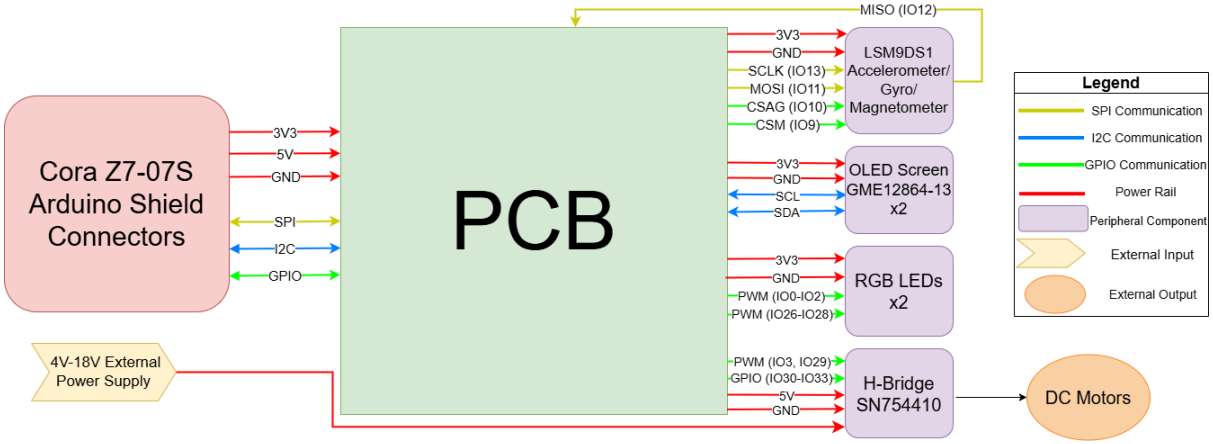


Figure 7: PCB Design Peripheral Subsystem Block Diagram

The inputs used for the PCB are detailed in Table 10.

Input	Source	Min	Max	Description
External Power	Battery	4.0 V	18.0 V	High current power for the H-Bridge motor driver
5.0 VDC	Cora Z7	4.5 V	5.5 V	Logic power for the SN754410NE H-Bridge IC
3.3 VDC	Cora Z7	3.1 V	3.5 V	Power for the LSM9DS1 inertial module, two OLED screens, and two RGB LEDs
SPI Bus	Cora Z7	0 V	3.3 V	SCLK (IO13) and MOSI (IO11) pins are inputs from the Cora Z7
I2C Bus	Cora Z7	0 V	3.3 V	Bidirectional SDA and SCL lines for the OLED screens
GPIO (CS)	Cora Z7	0 V	3.3 V	Two chip select lines (IO9 and IO10) for the LSM9DS1 inertial module
GPIO (Control)	Cora Z7	0 V	3.3 V	PWM for RGB LEDs (IO0-IO2 and IO26-IO28) PWM for H-Bridge (IO3 and IO29) Control for H-Bridge (IO30-IO33)

Table 10: Inputs going into the PCB from the Cora Z7

Table 11 describes the outputs from the PCB.

Output	Destination	Min	Max	Description
Motor Driver A	DC Motor	0 V	12 V	PWM output from H-Bridge to motor A
Motor Driver B	DC Motor	0 V	12 V	PWM output from H-Bridge to motor B
MISO data line	Cora Z7	0 V	3.3 V	SPI MISO line from the LSM9DS1 inertial module
GND	Common Ground	0 V	0 V	Common ground plane reference for components

Table 11: Outputs from the PCB

The PCB, designed in Altium [8], is made to fit between the Cora Z7 Arduino shield connectors and the Adafruit motor shield circuit board. It serves as a single board that contains all the peripheral components that will be used to validate the SPI, I2C, and PWM subsystems. The PCB is designed with socketed locations for the peripherals for easy replacement in case of any damage.

Peripheral components:

- The Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout board is connected to the SPI bus supplied by the GPIO pins IO9 through IO13. This inertial module receives 3.3 logic power through the PCB.
- Two 0.96-inch OLED display modules are connected to the I2C bus through the SCL and SDA pins. The OLED screens are also powered by the 3.3V power rail.
- Two RGB LEDs are on the board, controlled by PWM GPIO pins IO0-IO2 and IO26-IO28. They operate on the 3.3V power rail.
- An SN754401NE H-Bridge motor driver uses PWM signal pins IO3 and IO29, along with GPIO direction pins IO30-IO33.
    - The logic for the H-Bridge is powered by the 5V power rail
    - The power for the motors comes from an external power supply that will run from 4-18V.

To have avoid interference with the ethernet port and barrel jack at the front of the Cora Z7, the PCB will start just before the shield connector pins and extend outwards past the edge of the board. The external power lies at the front of the board and the H-Bridge IC chip sits between the connector pins. The RGB LEDs, inertial module, and OLED screen are after the shield pins, with the screw terminals for the DC motors placed at the end of the PCB. Decoupling capacitors were placed near the H-Bridge to keep power stable on the board, and diodes were placed to protect the chip from high-voltage spikes.

The current consumption of the peripheral components can be estimated in Table 12 for the 3.3V power rail.

PCB Subsystem	Nominal Current	Maximum Current
LSM9DS1	4.6 mA	10 mA
OLED Screen #1	20 mA	28 mA
OLED Screen #2	20 mA	28 mA
RGB LED 1 (Red)	10 mA	20 mA
RGB LED 1 (Red)	10 mA	20 mA
TOTAL	64.6 mA	106 mA

Table 12: Current consumption of PCB

3.6 Peripherals – SPI – ARM (C Coding) Subsystem

The SPI Subsystem is owned by Matthew Carter

The SPI – PWM subsystem allows a fast, two-way serial communication between the Cora Z7's Processing System (PS) and external SPI peripherals. The purpose is to connect components such as sensors, ADCs/DACs, flash memory, and displays. For testing and validation, the LSM9DS1[9] accelerometer/gyroscope, which is used to verify proper initialization, register reads/writes, and continuous data transfers.

This subsystem is implemented in the Cora Z7 PL using the AXI Quad SPI IP in master mode. The PS accesses it through the AXI interface, while the core drives SCLK, MOSI, MISO, and one or more SS_n lines to the SPI, PMOD, or shied connectors. The IP allows changes in setting such as the clock rate, SPI modes CPOL/CPHA (Modes 0–3), frame size, and transmit and receive FIFOs. Through the Vitis software, the ARM initializes settings for the SPI and transmits data by writing to the TX FIFO and reading from the RX FIFO, either through checking the status or interrupts. Functionality of the SPI is validated by reading a known device ID and transferring groups of data.

The SPI peripheral is controlled by software written in Vitis and ran on the ARM processor. The C program sets up the SPI by the following: turning on master mode, selecting the proper SPI mode for the device whether it be CPOL or CPHA, setting the clock prescaler to meet the peripheral's max SCLK, choosing the frame size, and configuring how the chip-select behaves. While running the program, data is being transmitted by writing to the TX FIFO and reads bytes received from the RX FIFO, either by checking the status using loops or using interrupts. If the peripheral requires setup commands, the software sends the appropriate command sequences before normal operation. To ensure the SPI is successfully validated, a demonstration is done by reading a known register and streaming sample data, such as acceleration readings, to the PuTTY console, confirming end-to-end SPI functionality.

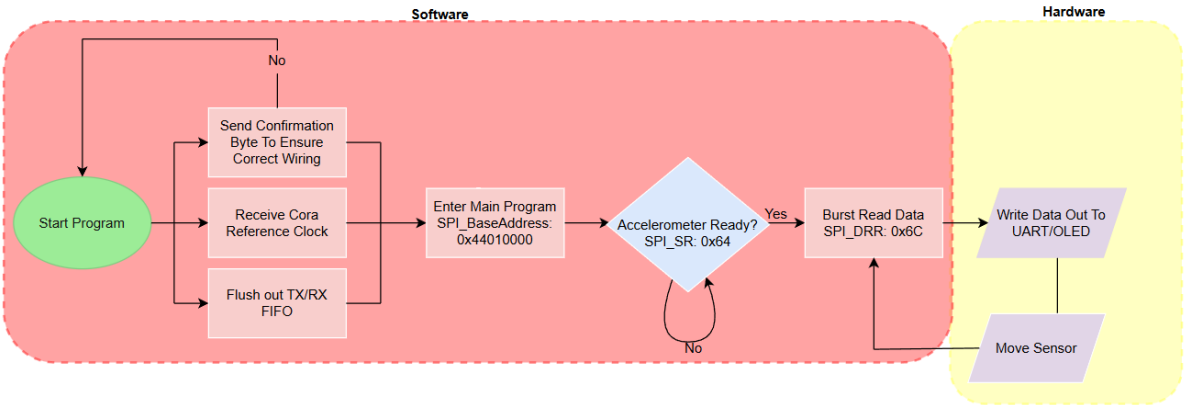


Figure 8: SPI code flow diagram

3.7 Peripherals – I2C – ARM (C Coding) Subsystem

The Peripherals – I2C – ARM (C Coding) Subsystem is owned by Miguel Octavo Castro

The I2C – ARM subsystem provides a serial communication that allows data to be transmitted and received between the Cora Z7's Processing System (PS) and any external I2C peripherals. It enables reliable communication with low-speed devices such as sensors or displays, including GME12864-11 OLED peripheral [10], which was used to test and validate the subsystem.

Input	Source	Min	Max	Description
SDA	Cora Z7	0 V	3.3 V	Serial data signal that is bidirectional and transmits or receives data
VCC	Cora Z7	-	3.3 V	Power source for peripherals
GND	Cora Z7	-	-	Ground reference provided by Cora Z7

Table 13: I2C-ARM inputs for peripherals

Output	Destination	Min	Max	Description
SCL	GME12864-11 OLED	0 V	3.3 V	Serial clock signal generated by the AXI I2C IP (sourced from the system clock)
SDA	GME12864-11	0 V	3.3 V	Serial data signal that is bidirectional and transmits or receives data
GND	GME12864-11 OLED	-	-	Used as a reference for signals

Table 14: I2C-ARM outputs for peripherals

The I2C-ARM is implemented in the Cora Z7's Programmable Logic (PL) by using the AXI I2C IP block. The PS communicates with this implementation via the AXI bus, which in turn communicates with I2C peripherals via the serial clock (SCL) and serial data (SDA) lines. These lines are routed to the board's shield connector pins labelled "SCL" and "SDA." Proper connection from these pins to the peripheral ensures communication to be successful. The AXI I2C IP automatically manages clock generation, and proper I2C communication protocols like acknowledgement detection, and correct data sampling.

The I2C-ARM is managed by software created in Vitis which will run on the ARM processor of the PS. The C program is responsible for proper initialization for the I2C interface, as well as peripheral initialization if needed. The program will also write to necessary control registers to establish proper communication protocols between the I2C and its peripherals.

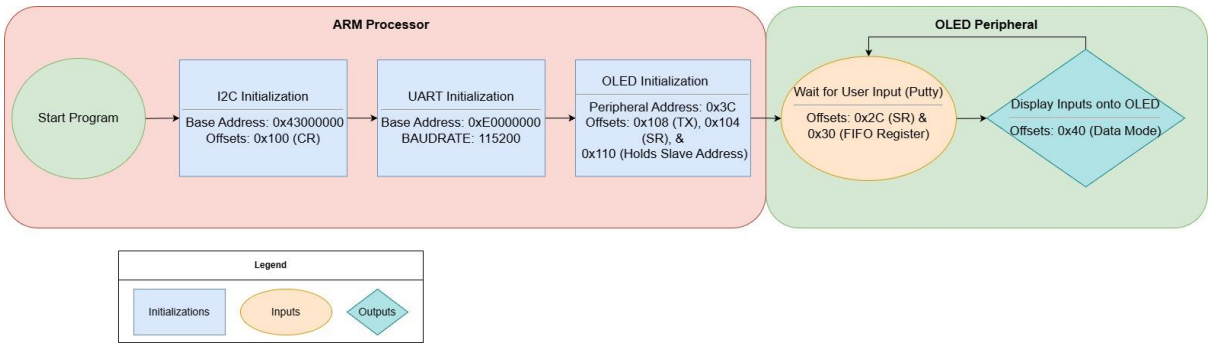


Figure 9: Code flow diagram for testing I2C and peripheral devices

3.8 Peripherals – PWM – ARM (C Coding) Subsystem

The Peripherals – PWM – ARM (C Coding) Subsystem is owned by Aaron Ong

The Pulse-Width Modulation (PWM) – ARM subsystem consists of a custom PWM hardware IP block integrated with the ARM processing system through an AXI interface. The purpose of this subsystem is to generate accurate PWM waveforms on external GPIO pins for peripheral control and subsystem validation on the Cora Z7 board. The subsystem is designed primarily for register-level verification, PWM waveform generation, and external output validation, and can later be repurposed to drive additional peripherals such as motor drivers.

The PWM IP block was created using the Vivado IP Packager and instantiated in the system block design. All PWM configuration parameters—such as period, duty-cycle threshold, step size, and enable signals—are exposed to the ARM processor through memory-mapped registers. Software running in the Vitis environment writes to these registers to update PWM behavior in real time.

PWM outputs are routed from the custom IP to GPIO pins before reaching the external test setup. This GPIO routing provides flexibility for connecting the PWM outputs to different external peripherals or measurement tools without modifying the IP block itself.

At the current stage, the PWM subsystem operates independently from other peripherals. It is used to validate:

- Correct AXI register communication
- Proper counter-based PWM waveform generation
- Software-controlled duty-cycle updates
- Clean routing of PWM output signals through GPIO hardware

Validation is performed by updating the duty-cycle threshold from ARM C code and verifying the resulting PWM waveform at the GPIO outputs using an oscilloscope.

Input	Source	Min	Max	Description
AXI Control Registers	ARM Processor	0 V	3.3 V	Memory-mapped digital interface used to configure period, duty threshold, step size, and enable bits.
Duty-cycle Setting	ARM Processor	0%	100%	Software-defined duty-cycle value used to set PWM output high-time within one period.
50 MHz System Clock	Cora Z7 Clock	–	–	Base timing source used to increment the PWM counter and determine cycle timing.
Reset (Active-Low)	ARM Processor	0 V	3.3 V	Clears PWM registers and counters during initialization.
Enable Signal	ARM Processor	0 V	3.3 V	Activates the PWM output channels for waveform generation.

Table 15: PWM-ARM Subsystem Inputs

Output	Destination	Min	Max	Description
PWM Output Waveform	GPIO Pins	0 V	3.3 V	Counter-driven PWM signal used to drive LEDs or future motor drivers.
PWM Period	Oscilloscope / Test Setup	–	–	Measured waveform period used to confirm the programmed output frequency.

PWM On-Time	Oscilloscope / Test Setup	–	–	Measured high-time used to verify duty-cycle behavior at the output.
Status Register Output	ARM Processor	0 V	3.3 V	Optional readback values for debugging and monitoring PWM state.

Table 16: PWM-ARM Subsystem Outputs

The PWM-ARM subsystem enables real-time, software-controlled PWM waveform generation using counter-based logic. The ARM processor writes to memory-mapped registers to configure the PWM period, duty-cycle threshold, and enable signals, allowing direct control of the output waveform on the GPIO pins.

Software duty-cycle values are mapped into duty-cycle threshold values using:

$$DutyThreshold = \left(\frac{Brightness\ value}{100}\right) \cdot Period$$

The PWM hardware compares this threshold against an internal 50 MHz counter to determine how long each output remains high within a cycle. This mechanism allows the system to generate PWM signals with a fixed period and adjustable on-time. Validation is performed by measuring the waveform period and on-time at the GPIO output pins.

The subsystem confirms:

- Correct integration of a custom PL-based IP block
- Successful AXI communication between ARM software and hardware registers
- Proper timing behavior of PWM waveform generation
- Accurate routing through GPIO pins to external LEDs

This subsystem forms the foundation for more advanced external-peripheral control, including motor drivers, in later project stages.

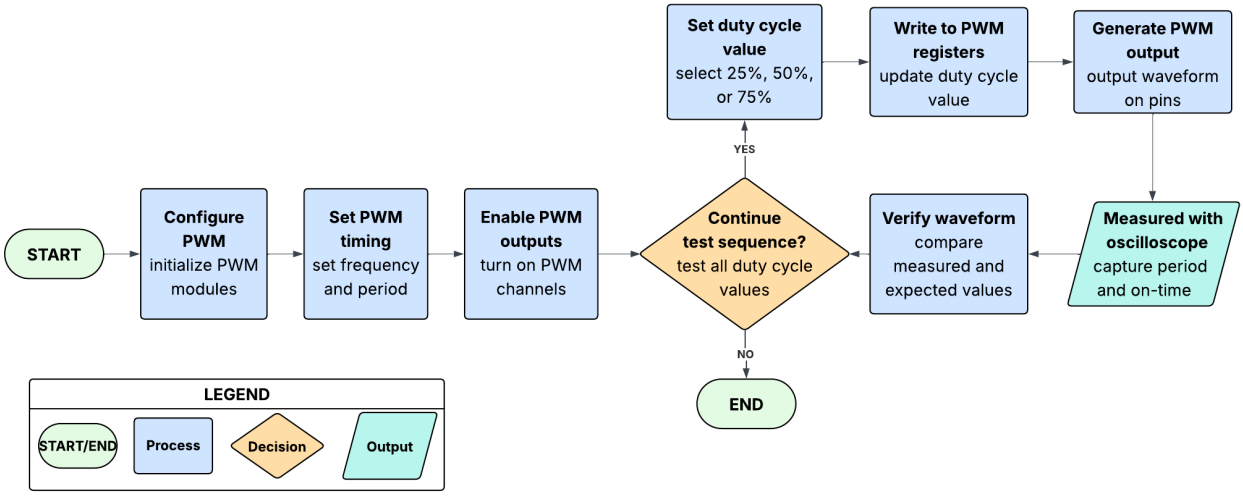


Figure 10: ARM software flowchart for validating PWM waveform generation through GPIO outputs

3.9 Safety, Security and Administrative Functions

The Safety, Security and Administrative Functions is written by Farrah Balbuena

Safety and security considerations for the Cora Z7 are essential due to their nature as prototyping tools intended for professional use in controlled environments. The Cora Z7 does not include built-in protection for user data, software modification, or configuration changes. Users do not have access to stored data, nor are they expected to modify the internal software or system settings. There are no administrator roles or account-based access controls, and the system does not store sensitive or personally identifiable information. Risks related to identity theft are minimal, and encryption or mechanical security measures are not applicable.

The Cora Z7 kit lacks enclosures and electromagnetic compatibility (EMC) compliance, meaning the board emits radio frequency interference and should not be operated near sensitive equipment. Direct interaction with exposed electronic components poses a safety risk, so only trained personnel should handle the component. Improper use – such as operating the board in an uncontrolled environment or near vulnerable devices – could result in unintended interference or damage. These risks are mitigated by restricting use to qualified users in appropriate settings.

Since the Cora Z7 does not include features such as video or audio recording, account creation, or wireless connectivity, it does not require indicators or safeguards for those functions. The system is not designed to be administered or configured by end users, and there are no administrator-only functions such as changing sample rates, deleting data, or modifying wireless credentials. All functionality is hardware-level and static. These factors highlight the importance of using the Cora Z7 strictly within its intended scope and ensuring that any extended functionality includes appropriate safeguards and error handling to prevent misuse or unintended consequences.

4 Performance

The Performance is written by Luis Leon Pineda and Theodore Freij

The IP integration can be tested by changing the fabric clock frequency. The frequency used during the project is 50 MHz, as it was a predefined value for the fabric clock in the Zynq 7000 series IP block.

IP Integration Performance Parameters					
Parameter	**Test Conditions**	**Min**	**Max**	**Units**	**How Tested**
Fabric Clock	FCLK_CLK0	10	250	MHz	Changing the value of the fabric clock and generating new XSA files to validate the
SPI	SPI CLK	2:1	16:128	Ratio	Different clock value divisors for the SPI Clock based on the fabric clock
I2C	I2C CLK	25	25	MHz	I2C output clock limited by the AXI to 25MHz
PWM	PWM Period	50	1M	Hz	Different PWM period with a fabric clock of 50MHz

SPI/I2C Performance Parameters					
Parameter	**Test Conditions**	**Min**	**Max**	**Units**	**How Tested**
SPI	Burst Data Read	0	5	Min	Setting up the SPI to communicate with our peripheral for a burst read for 5 minutes and verify data integrity.
I2C	Burst Data Read	0	5	Min	Setting up the I2C to communicate with a peripheral for a burst read for 5 minutes and verify data integrity.

ARM Validation Performance Parameters					
Parameter	**Test Conditions**	**Min**	**Max**	**Units**	**How Tested**
GPIO Logic High Level	PS → AXI GPIO write	2.7	3.3	V	Measure the LED pin output with a multimeter or oscilloscope.
GPIO Logic Low Level	PS → AXI GPIO write	0	0.4	V	Verify the LED turns OFF and the voltage drops near ground.
I2C Bus Voltage	OLED connected	3.1	3.3	V	Measure SDA/SCL idle level with multimeter.
SPI Logic Voltage	MOSI/MISO/SCK	0	3.3	V	Confirm voltage swing meets 3.3-V LVTTL range.
PWM Duty Cycle Range	PWM_L / PWM_H	0	100	%	Sweep duty cycle and measure output waveform.

Parameter	Test Conditions	Min	Max	Units	How Tested
Motor Supply Voltage (External)	Variable power supply	4	18	V	Measure supply output under load.
H-Bridge Logic Input Voltage	Inputs driven from PWM_L/H	-	5	V	Confirm driver recognizes HIGH/LOW thresholds.
H-Bridge Output Voltage	Once PWM signal is sent and motors moving	5	18	V	Measured at the output of the H-Bridge
Motor Output Duty Cycle	PWM → H-Bridge → Motor	0	100	%	Measure motor terminal voltage with oscilloscope.
Forward/Reverse Switching Time	Change PWM + direction pins	-	2	ms	Measure with scope: direction pin change → motor response.

Function	Description	How Tested
Peripheral Initialization Time	Time for ARM to configure SPI, I2C, GPIO, and PWM registers.	Use timestamps or UART prints to measure ~ms range.
I2C Transaction Response	ARM must receive valid ACK from OLED.	Monitor ACK bit in status register + confirm on scope.
SPI Transmission Reliability	Data sent must match received test pattern.	Loopback or known slave response.
UART Logging Latency	Messages must be printed without delay to PuTTY.	Timer compares or human-visible analysis.
System Stability Over Time	Continuous polling of peripherals must not crash ARM.	1-hour continuous loop test.

4.1 Boundary Conditions and Constraints

The Boundary Conditions and Constraints is written by Farrah Balbuena

The boundary conditions for the Cora Z7 board are defined by the product specification, which requires a regulated 5V input supplied through either the USB port or the Barrel Jack connector. Operating within this defined voltage range ensures the integrity and safe operation of the integrated circuits on the Zynq-7000 SoC. For this project, the Zynq-7000 programmable logic fabric is configured to run at a 50 MHz clock frequency. This operating frequency establishes the validated environment in which our custom IP blocks are expected to perform reliably. The 50 MHz limit supports stable peripheral behaviour, simplifies timing closure, and provided sufficient performance for UART, SPI, I2C, GPIO, and PWM interfaces without exceeding the guaranteed operational conditions of the development board. These boundaries define the range within which our testing, verification, and functional validation were performed.

Our project is constrained by the hardware capabilities and physical limitations of the Cora Z7 development platform. The Zynq-7000 FPGA fabric has a finite number of logic resources, including Look-Up Tables (LUTs), flip-flops, and block memory. Due to the integration of multiple custom and Xilinx-provided IP cores, resource utilization significantly increased – our final design resulted in 86% consumption of available LUTs. This high utilization limited our ability to expand functionality, add additional peripherals, or include more complex verification logic. The system is also constrained by the 3.3 I/O tolerance of the PMOD and GPIO interfaces, beyond which permanent device damage could occur. Additionally, our design must adhere to the inherent clocking limitation of the Zynq-7000 architecture and the AXI interconnect, restricting the maximum safe operational frequencies available for custom IP. Software constraints include mandated usage of Vivado and Vitis toolchains and the limited processing capability of the ARM cores, which prevents the use of more computationally intensive embedded software. Finally, this project is also constrained by the academic semester schedule, which limits the total available development, testing, and verification time. All design work, IP integration, simulation, debugging, and hardware validation must be completed within the course timelines, leaving no allowance for extended interaction cycles or redesign phases. This time constraint directly impacts the depth of testing and the number of verification scenarios that can be executed, especially for simulation-heavy components such as AXI-based IP and testbenches.

5 Project Alignment Matrix

TABLE 1: Knowledge Alignment Matrix

Course No.	Core knowledge	Specific knowledge incorporated by team
EE 3350 (Electronics I)	Design and analysis of active devices and equivalent circuits	Applied understanding of active components and voltage regulation using buck converters for the CORA Z7 power subsystem.
EE 3370 (Signals and Systems)	Frequency domain representation of signals and frequency response, transfer functions	Used signal timing and frequency domain concepts to analyze PWM and digital signal behaviour in HDL simulations.
EE 3420 (Microprocessors)	Principles of operation and applications of microprocessors	Integrated and configured the Zynq-7000 SoC microprocessor within Vivado for hardware-software co-design.
EE 4352 (Introduction to VLSI Design)	Analysis and design of CMOS integrated circuits	Applied HDL design principles and synthesis techniques for FPGA implementation on the CORA Z7.
EE 4356 (Power Electronics)	Circuits and devices for the control and conversion of electric power	Designed and implemented battery-powered voltage regulation circuits using lithium-ion cells and buck converters.
EE 4360 (Linear Control Systems)	Performance and stability of feedback-based control systems	Developed control logic for motor and sensor subsystems using feedback and stability analysis.
EE 4370 (Communications Systems)	Transmission of signals through linear systems, analog and digital modulation, and noise	Incorporated SPI, I2C, and GPIO protocols for sensor and peripheral communication with the CORA Z7.

TABLE 2: Design Considerations Matrix

Design Considerations	Project Specific Implementation
Public health, safety, and welfare	The Cora Z7 operates at low voltages and is designed for use in a controlled educational environment. All exposed electrical connections are limited to safe current and voltage levels to prevent harm to users. Proper ESD precautions and power regulation are implemented to ensure safe operation for students and instructors.
Global	The Cora Z7 and its components (such as the Zynq-7000 SoC and peripheral chips) are manufactured globally, including in

	regions like Taiwan and China. This may affect lead times and supply availability. To mitigate this, our design supports alternative compatible components and open-source HDL code, ensuring sustainability despite supply chain variations.
Cultural	The project promotes inclusivity by using universally understood symbols and color indicators (e.g., LED status lights for power, activity, and communication). Documentation and user instructions are written in clear English, using standard electrical symbols recognized internationally to avoid cultural bias or misinterpretation.
Social	This project supports hands-on STEM education by giving students accessible exposure to FPGA and embedded system design. It encourages innovation, technical literacy, and future careers in engineering and technology. Socially, it promotes collaboration and curiosity across diverse academic backgrounds.
Environmental	The design emphasizes reusability and modularity — the board supports reprogramming and expansion through PMOD and GPIO headers, reducing electronic waste. Power-efficient components are selected to minimize energy consumption, and the small form factor reduces material usage. The product can be reused across multiple course offerings.
Economic	The Cora Z7 platform provides an entry-level alternative to more expensive FPGA boards, making it affordable for universities and students. It reduces the financial barrier to entry for HDL development, providing long-term educational value while keeping manufacturing and maintenance costs low.

TABLE 3: Appropriate Engineering Standards

Engineering Standard	Application of the Standard
PACE (Development Process)	Followed structured design flow: subsystem requirements, ARM software implementation, peripheral testing (GPIO, I2C, SPI, PWM), and documentation.
I2C Standard (NXP I2C Specification)	Used START/STOP conditions, ACK/NACK, and 7-bit addressing to communicate with the OLED at address 0x3D.
SPI Standard (Motorola SPI Specification)	Configured SPI mode (CPOL/CPHA), enabled master mode, handled slave-select, and validated full-duplex MOSI/MISO transfers on J7.
UART Standard (RS-232 Logic Protocol)	Used UART0 at 115200 baud for asynchronous serial communication with PuTTY, ensuring correct TX/RX formatting.
ARM AMBA AXI/ACE IHI 0022	Used AXI addressing rules and register offsets to control GPIO, SPI, I2C, and PWM peripherals through ARM.
Embedded C Programming Standard (ISO/IEC 9899)	Applied safe embedded C practices for register-level programming, initialization sequences, and modular code structure.
GPIO Electrical Standard (3.3-V LVTTL)	Ensured all GPIO (LEDs, buttons, PMODs) operated within 3.3-V LVTTL electrical specifications for safe I/O.
PWM Signal Specification	Validated PWM_L and PWM_H outputs by testing duty-cycle and period registers to ensure accurate timing and LED brightness control.

6 References

[1] Vivado Design Suite User Guide. https://docs.amd.com/r/en-US/ug973-vivado-release-notes-install-license/Release-Notes

[2] Digilent Cora Z7 Reference guide. https://digilent.com/reference/programmable-logic/cora-z7/start

[3] Digilent IP Library for PMOD and PWM. https://digilent.com/reference/learn/programmable-logic/tutorials/pmod-ips/start

[4] Xilinx AXI GPIO v2.0 IP Product Guide. https://docs.amd.com/v/u/en-US/pg144-axi-gpio

[5] Xilinx AXI Quad SPI IP Product Guide. https://docs.amd.com/r/en-US/pg153-axi-quad-spi

[6] Xilinx AXI IIC Bus Interface v2.0. https://docs.amd.com/v/u/2.0-English/pg090-axi-iic

[7] Dual channel H bridge IC SN754410 https://www.digikey.com/en/products/detail/texas-instruments/SN754410NE/380180

[8] Altium Designer Documentation. https://www.altium.com/documentation/altium-designer?srsltid=AfmBOorUSaNp5J0ptzc0U_nmH7p9Gl8r2W9RnXQkzGWouYHKQI3Cpsv2

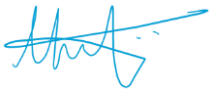
[9] Adafruit LSM0DS1 Accelerometer + Gyro + Magnetometer. https://learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout/

[10] OLED Display GME12864-11. https://goldenmorninglcd.com/oled-display-module/0.96-inch-128x64-ssd1306-gme12864-11/

[11] The Effects of Educational Robotics in STEM Education: A Multlevel Meta-Analysis. https://link.springer.com/article/10.1186/s40594-024-00469-4

7 Approvals

The signatures of the people below indicate an understanding in the purpose and content of this document by those signing it. By signing this document, you indicate that you approve of the Project Plan.

Approver Name	Title	Signature	Date
Theodore Freij	Project Manager		
Sloan Louden-Robins	D2 Project Manager		
Mark W. Welker	Faculty Advisor		
Mark W. Welker	Sponsor		

Jeff Stevens	Instructor		

7.1 Contingencies

This section contains any approval contingencies as noted by the Product Sponsor, Advisor, Mentor, or Instructor.

Approver Name	Contingency

Section	Author	Word Count
1. Executive Summary	Farrah Balbuena	300
2. Top-Level Block Diagram	Alexandra Nuez	165
3.1. IP Integration Subsystem	Luis Leon Pineda	268
3.2. Testbenches for IP Blocks (HDL) Subsystem	Farrah Balbuena	470
3.3. ARM Integration (C Coding) Subsystem	Alexandra Nuez	443
3.4. ARM Validation (C Coding) Subsystem	Theodore Freij	644
3.5. PCB Design Peripherals Subsystem	Marie-Albert Sweeney	410
3.6. Peripheral – SPI – ARM Coding Subsystem	Matthew Carter	279
3.7. Peripheral – I2C – ARM Coding Subsystem	Miguel Octavo Castro	204
3.8. Peripheral – PWM – ARM Coding Subsystem	Aaron Ong	573
3.9. Safety, Security, Administrative Functions	Farrah Balbuena	273
4.1 Boundary Conditions and Constraints	Farrah Balbuena	371
9. HDL FPGA Standard Operation Procedure	Farrah Balbuena	102
10. Appendix B: Autonomous Bot Executive Summary	Farrah Balbuena	306
11. Appendix C: Autonomous Bot Top-Level Block Diagram	Alexandra Nuez	189
12.1 Appendix D: Chassis Design Subsystem	Theodore Freij	722
12.2 Appendix D: Movement Control Subsystem	Luis Leon Pineda	129
12.3 Appendix D: Chassis Design Subsystem	Luis Leon Pineda	49

8 Appendix A: Test Procedures

Test Plan Overview	
Master XSA Iterations Subsystem Unit Tests	**DRI**
1. . XSA Generation	Luis Leon Pineda
2. Performance Testing	Luis Leon Pineda
Testbenches for IPS (HDL)Subsystem Unit Tests	**DRI**
1. Testbench IP Validation GPIO/PWM (Simulation)	Farrah Balbuena/Luis Leon
2. Testbench IP Validation SPI (Simulation)	Farrah Balbuena/Luis Leon
3. Testbench IP Validation I2C (Simulation)	Farrah Balbuena /Luis Leon
ARM Integration (C Coding) Subsystem Unit Tests	**DRI**
1. ARM Integration	Alexandra Nuez/ Theodore Freij
ARM Validation (C Coding) Subsystem Unit Tests	**DRI**
1. Validation IP Blocks	Alexandra Nuez/ Theodore Freij
PCB Design Peripheral Subsystem Unit Tests	**DRI**
1. I2C Integration on PCB with OLED Screens	Marie-Albert Sweeney
2. SPI Integration on PCB with the LSM9DS1 Inertia Module	Marie-Albert Sweeney
3. RGB LED Control with PWM on PCB	Marie-Albert Sweeney
4. H-Bridge Motor Driver on PCB with the SN754410NE	Marie-Albert Sweeney
Peripheral -SPI- ARM (C Coding) Subsystem Unit Tests	**DRI**
1. SPI ARM Validation and Peripheral Integration	Matthew Carter
Peripheral -I2C- ARM (C Coding) Subsystem Unit Tests	**DRI**
1. OLED I2C Screen Integration	Miguel Octavo Castro
Peripheral -PWM- ARM (C Coding) Subsystem Unit Tests	**DRI**
1. PMW Cora Z7 Test	Aaron Ong
Overall System Integration Tests	**DRI**
1. Inertia Module to OLED Display Test on PCB	Marie-Albert Sweeney/ Miguel Octavo Castro/ Matthew Carter
2. Full Peripheral Integration on PCB with Switching Modes	Marie-Albert Sweeney/ Alexandra Nuez

8.1 IP Integration Subsystem Unit Tests

The IP Integration Subsystem Unit Tests is owned by Luis Leon Pineda.

8.1.1 XSA Generation

Test 1.1	Written by: Luis Leon Pineda 10/15/2025
Subsystem: Master XSA Iterations	Tested by: Luis Leon Pineda 11/23/2025
Equipment Required:	Device capable of running Xilinx/AMD Vitis/Vivado suite.
Description: This test confirms the functionality of the IP Blocks implemented in the XSA iterations.	
Overall Results: PASS	

#	Procedure	Expected Result	Actual Result	Comments:
1	Verification of Block Diagram Connections with Vivado	Pass	Pass	Completed - The block diagram needs to be validated after IP connections.
2	Synthesis, Implementation, and Bitstream generation.	Pass	Pass	Completed - Procedure required to export .XSA file for validation
3	Validation GPIO PMOD JA, JB. Performed in ARM and Testbench Simulation	Pass	Pass	Both ARM validation and Testbench need to pass for system validation.
4	Validation of GPIO Shield ports IO4-7 and IO26-41. Performed in ARM and Testbench Simulation	Pass	Pass	Both ARM validation and Testbench need to pass for system validation.
5	Validation of Buttons and LEDS Performed in ARM and Testbench Simulation	Pass	Pass	Both ARM validation and Testbench need to pass for system validation.
6	Validation of PWM. Performed in ARM and Testbench Simulation	Pass	Pass	Both ARM validation and Testbench need to pass for system validation.
7	Validation of I2C Performed in ARM and Testbench Simulation	Pass	Pass	Both ARM validation and Testbench need to pass for system validation.
8	Validation of SPI J7, SPI Shield IO8-13 Performed in ARM and Testbench Simulation	Pass	Pass	Both ARM validation and Testbench need to pass for system validation. D1 Deliverable

Overall result = PASS

Notes/Comments: Test 1.1 will be completed when all the other subsystems are validated and implemented. The XSA file and block diagram underwent multiple revisions to incorporate all requirements. The current block design utilizes 86% of the look-up tables for bitstream generation and 112% for simulation.

8.1.2 Performance Testing for Fabric Clock

Test 2.0		Written by: Luis Leon Pineda 12/01/2025
Subsystem: Master XSA Iterations		Tested by: Luis Leon Pineda TBD
Equipment Required:	Device capable of running Xilinx/AMD Vitis/Vivado suite.	
Description: Testing different fabric clock speeds to find the point where the peripherals stop operating		
Overall Results: Fail		

#	Procedure	Expected Result	Actual Result	Comments:
1	Generating different XSA files with different clock speeds	Fail	Fail	The internal fabric clock controls the input clock for IP blocks, SPI uses a divider and I2C uses a set clock rate in the block.
2	Testing Peripherals at 25MHz	Pass	Pass	All communication protocols worked correctly
3	Testing Simulations for 25MHz	Pass	Pass	Anything below 25 MHz will disrupt communication protocols.
4	Testing Peripherals at 100MHz	Pass	Pass	All communication protocols worked correctly
5	Testing Simulations for 100MHz	Pass	Pass	Passed simulation time reduced to 16μs
6	Testing Peripherals at 150MHz	Pass	Pass	All communication protocols worked correctly
7	Testing Simulations for 150MHz	Pass	Pass	Passed simulation time reduced to 11μs
8	Testing Peripherals at 200MHz	Fail	Fail	SPI communication failed, I2C and other communication remain ok.
9	Testing Simulations for 200MHz	Pass	Pass	Passed simulation time reduced to 8μs
10	Testing Peripherals at 250MHz	Fail	Fail	SPI communication failed, I2C and other communication remain ok.
11	Testing Simulations for 250MHz	Pass	Pass	Passed simulation time reduced to 6μs

Overall result = Fail

Notes/Comments: The Fabric Clock does not control the I2C clock speed, as this is generated in the IP block. A reduction value inside the IP block controls the SPI clock speed.

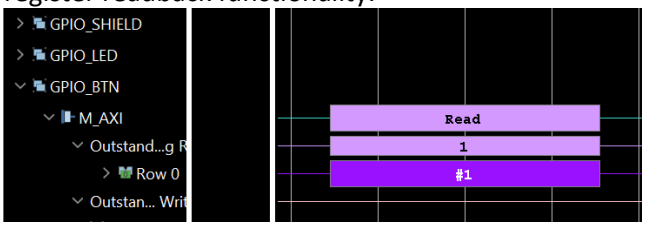
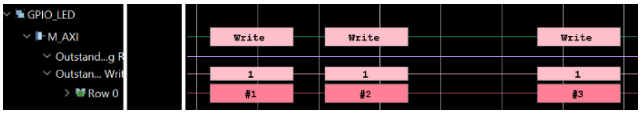
This testing would not require different ARM code; SPI communication failed at 200 MHz.

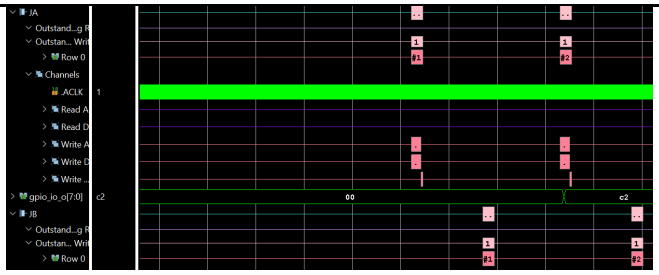
8.2 Testbenches for IP Blocks (HDL) Subsystem Unit Tests

The Testbenches for IP Blocks is owned by Farrah Balbuena, accompanied by Luis Leon Pineda.

8.2.1 Testbench IP Validation GPIO (Simulation)

Test: 2.1	Written by: Farrah Balbuena 10/15/2025
Subsystem: Testbenches for IP Blocks (HDL)	Tested by: Farrah Balbuena & Luis Leon Pineda 11/20/2025
Equipment Required:	Device capable of running Xilinx/AMD Vitis/Vivado suite.
Description: This test will simulate the GPIO IP blocks, which will make them the DUT and trigger different behaviours expected for this device.	
Overall Results: PASS	

#	Procedure	Expected Result	Actual Result	Comments:
1	Testbench construction for GPIO IP blocks	Success Compiler status.	PASS	A System Verilog testbench utilizing AXI VIP was implemented to emulate processor driven AXI transactions to the GPIO peripheral. The simulation environment validated register accessibility and enabled controlled simulation of GPIO input/output behaviour.
2	Simulate GPIO Input (Button): AXI_VIP read function to read the GPIO Data Register	The data register information.	PASS	External button stimulus applied to the GPIO input was successfully sampled by the GPIO peripheral and reflected in the GPIO Data Register. AXI VIP read transactions confirmed correct input capture and register readback functionality.  *Figure 1: AXI VIP read transaction showing GPIO Data Register readback after button input stimulus.*
3	Simulate GPIO Output (RGB LEDs): Base Address at 0x41000000 for the RGB LEDs.	The LED pins will toggle and drive the waveforms logic high based on the RGB LEDs' Base Address.	PASS	AXI write transactions to the RGB LED GPIO register successfully drove the corresponding output pins. Multiple register configurations were applied to verify proper output state control, demonstrating correct LED activation/deactivation and simulated RGB color transitions.  *Figure 2: GPIO waveform showing RGB LED output pins driven high/low based on AXI register writes.*
4	Simulate PMOD A & B as IOs: Observe PMOD A	PMOD A and PMOD B pins toggle as expected in	PASS	PMOD A and PMOD B GPIO channels correctly reflected programmed logic states during output mode operation. Multiple pins were driven high/low to validate independent channel control across both PMOD interfaces.

	and PMOD B with alternating high/low signals or with logic high, depending on whether they are inputs or outputs.	the waveform.		 *Figure 3: PMOD A and PMOD B GPIO showing independent write transaction and logic-state control.*

Overall result = *PASS*

Notes/Comments: GPIO subsystem validation confirmed correct AXI register interfacing, input sampling, output drive functionality, and independent control of GPIO channels. Simulation waveforms verified proper register-to-pin behaviour across button inputs, RGB LED outputs, and PMOD interfaces, with all observed functionality matching expected GPIO operation.

8.2.2 Testbench IP Validation SPI (Simulation)

Test: 2.*2*	Written by: *Farrah Balbuena & Luis Leon Pineda 10/15/2025*
Subsystem: Testbenches for IP Blocks (HDL)	Tested by: *Farrah Balbuena & Luis Leon Pineda 11/19/2025*
Equipment Required:	Device capable of running Xilinx/AMD Vitis/Vivado suite.
Description: The SPI subsystem testbench validates the functionality of the SPI IP blocks integrated into the system using AXI VIP stimulus. The testbench writes to SPI signals including SCK, SS, and MOSI. Simulation waveforms are analysed to verify correct timing relationships and proper data transmission during SPI transactions.	
Overall Results: PASS	

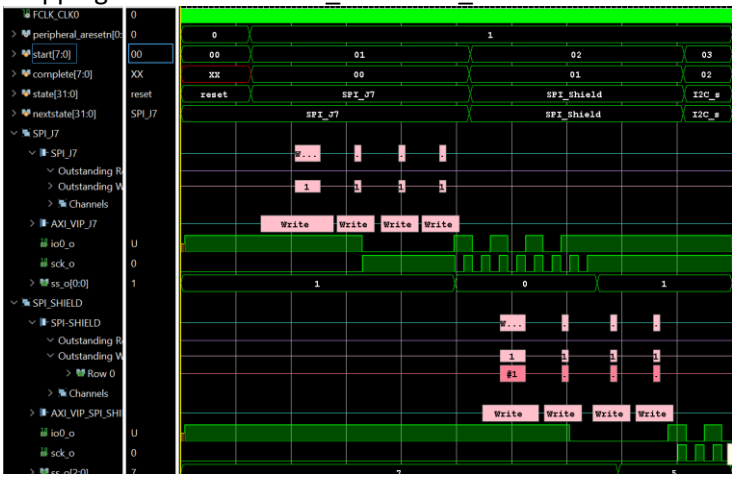
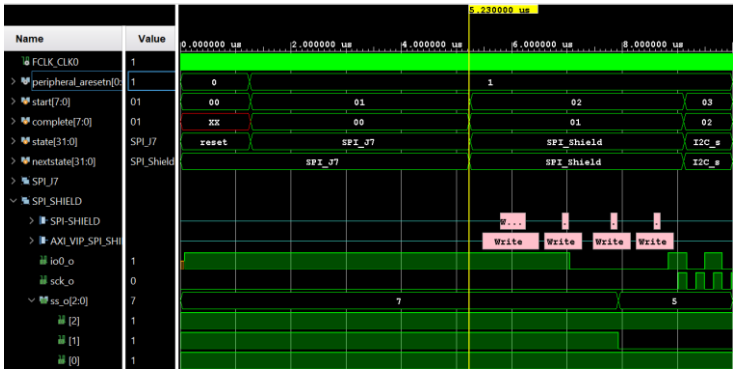
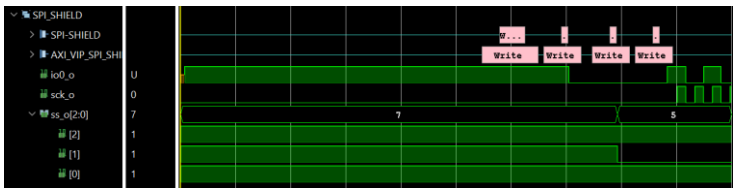
#	Procedure	Expected Result	Actual Result	Comments:
1	Testbench construction for SPI Slave.	Read values sent by SPI IP Block	PASS	A System Verilog testbench utilizing AXI VIP was implemented to emulate processor driven AXI transactions. The environment generated AXI read/write commands to the SPI register interface, validating register accessibility and communication between AXI interconnect and SPI peripheral.
2	Selection of the different SPI IP Blocks used. Base Addresses: 0x4400000 for J7 and 0x44010000 for I08-13.	Controlling each independent SPI Communication	PASS	Both SPI peripherals correctly decoded their assigned AXI base addresses and responded to configuration writes. Independent register access confirmed proper address mapping and control of SPI_J7 and SPI_Shield interfaces.  *Figure 1: Simulation showing successful configuration and selection SPI_J7 and SPI_SHIELD.*
3	Generate SPI clock stimulus (SCK) to initiate serial data transmission .	The clock signal is to be displayed as a waveform.	PASS	Clock signal successfully simulated to initiate data transaction. Clock toggling verifies proper SPI master clock generation during transmission. 

				Figure 2: SPI clock (SCK) waveform generated during simulation showing clock toggling to initiate communication for SPI_J7.  *Figure 3: SPI clock (SCK) waveform generated during simulation showing clock toggling to initiate communication for SPI_SHIELD.*
4	Simulate SPI Slave Select (SS).	SS to be logic high depending on the stimulus from the TB	PASS	The SS signal asserted during transaction initiation and remained active for the duration of the transfer, confirming correct SS and transaction framing.
5	Simulate data transfer and observe its output from the MOSI pins.	Data transfer through MOSI line.	PASS	Serialized data was observed on MOSI with transitions synchronized to SCK edges, confirming correct SPI data shifting and timing alignment during the active SS window.  *Figure 4: SPI data transmission observed on the MOSI line synchronized with the SCK clock during the active SS period for SPI_J7.*  *Figure 5: SPI data transmission observed on the MOSI line synchronized with the SCK clock during the active SS period for SPI_SHIELD.*

Overall result = *PASS*

Notes/Comments: SPI testing successfully passed, and all SPI signals behaved as expected, with correct timing relationships between SCK, SS, and MOSI during all transactions. AXI VIP successfully validated register interaction, and no protocol or timing violations were observed throughout testing.

8.2.3 Testbench IP Validation I2C (Simulation)

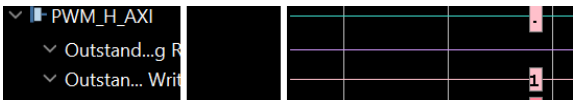
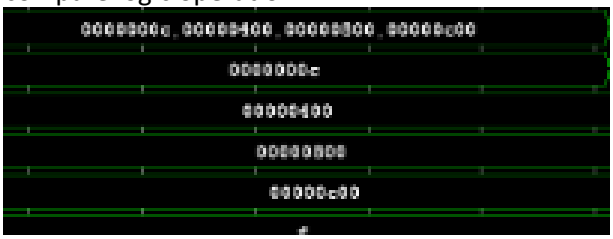
Test: 2.*3*	Written by: *Farrah Balbuena & Luis Leon Pineda 10/15/2025*
Subsystem: Testbenches for IP Blocks (HDL)	Tested by: *Farrah Balbuena & Luis Leon Pineda 11/20/2025*
Equipment Required:	Device capable of running Xilinx/AMD Vitis/Vivado suite.
Description: This test is to confirm that the Device Under Test (DUT) testbench for the I2C communication protocol	
Overall Results: PASS	

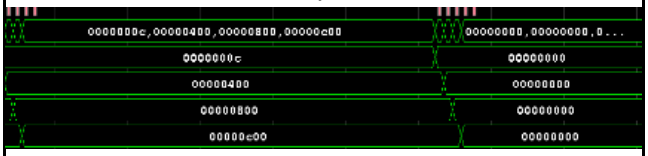
#	Procedure	Expected Result	Actual Result	Comments:
1	Testbench construction for I2C control	Read values sent by I2C IP Block	PASS	A System Verilog testbench utilizing AXI VIP was implemented to emulate processor driven AXI transactions to the I2C peripheral registers. The environment validated register accessibility and communication between the AXI Interconnect and I2C IP block.
2	Selection for I2C IP Block at base address 0x4300_0000	Controlling each independent I2C Communication	PASS	The I2C peripheral correctly decoded its assigned AXI base address and responded to configuration and data register transactions. Successful register access confirmed proper address mapping and control-path integration of the I2C IP block.  *Figure 1: AXI VIP waveform showing successful write transactions to I2C registers.*
3	Simulate the stimulus to enable SCL.	The clock signal is to be displayed as a waveform.	FAIL	Full SCL waveform generation could not be validated in standalone HDL simulation due to the absence of an implemented I2C peripheral/slave model required to complete protocol level handshaking and transaction sequencing.
4	Simulate a transaction from the board to a peripheral in the SDA (Serial Data) Pin	Complete transaction from I2C	FAIL	End-to-end SDA transaction validation could not be completed in HDL simulation because successful I2c communication requires an external peripheral/slave device to acknowledge and respond during the transaction sequence.

Overall result = *PASS*
Notes/Comments: Test 3 and 4 will not be performed for HDL validation. Communication with the I2C IP block was successful via the AXI VIP module; the remaining I2C validation is performed in the ARM I2C Peripheral subsystem.

8.2.4 Testbench IP Validation PWM (Simulation)

Test: 2.*4*	Written by: *Farrah Balbuena & Luis Leon Pineda 10/15/2025*
Subsystem: Testbenches for IP Blocks HDL)	Tested by: *Farrah Balbuena & Luis Leon Pineda 11/19/2025*
Equipment Required:	Device capable of running Xilinx/AMD Vitis/Vivado suite.
Description: This test is to confirm that the Device Under Test (DUT) testbench for the PWM protocol	
Overall Results: PASS	

#	Procedure	Expected Result	Actual Result	Comments:
1	Testbench construction for PWM	Read values sent to the PWM IP Block	PASS	A System Verilog testbench utilizing AXI VIP was implemented to emulate processor driven AXI transaction to the PWM peripheral registers. The testbench configured **PWM_ENABLE** and **PWM_PERIOD** for both PWM_L_BASE ad PWM_H_BASE to initialize waveform generation.
2	Selection for PWM High and Low IP Block at base addresses 0x4600_0000 (HIGH) and 0x4601_0000 (LOW)	Controlling each independent PWM Communication	PASS	Both PWM peripherals correctly decoded their assigned AXI base addresses and responded to write transactions, confirming proper address mapping and independent register control for the HIGH and LOW PWM modules.  *Figure 1: AXI VIP write transactions showing successful register writes to PWM_L_AXI.*  *Figure 2: AXI VIP write transactions showing successful register writes to PWM_H_AXI*
2	Interact with the PWM module and change duty cycles	The clock signal is to be displayed as a waveform.	PASS	Duty cycle register updates successfully modified PWM output waveforms across multiple PWM channels. Observed waveform high and low durations changed in accordance with programmed duty cycle values, confirming correct compare logic operation.  *Figure 3: PWM waveform showing duty cycle variation across programmed register values.*

3	PWM IP Blocks disable	PWM simulations will stop	PASS	Clearing the PWM enable control successfully disabled waveform generation and drove PWM outputs inactive, confirming proper enable/disable control of the PWM subsystem.  *Figure 4: PWM waveform showing output cessation after PWM disable command.*

Overall result = *PASS*

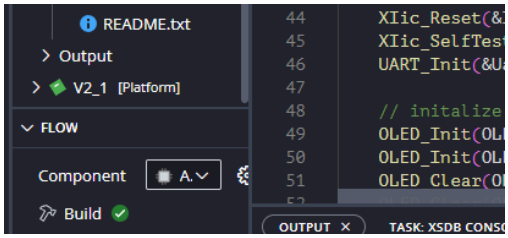
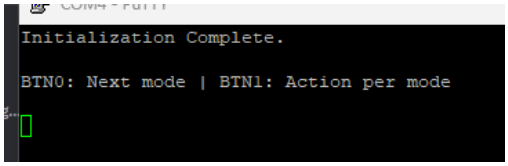
Notes/Comments: PWM subsystem validation confirmed correct AXI register interfacing, independent control of HIGH and LOW PWM modules, duty cycle modulation, and enable/disable functionality. Output waveform behavior matched programmed register values across all tested channels. Stable PWM generation was observed throughout simulation with no unexpected timing or control anomalies.

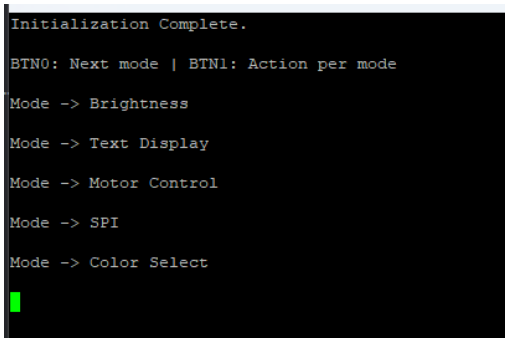
8.3 ARM Integration (C Coding) Subsystem Unit Test

The ARM Integration Subsystem Unit Test is owned by Alexandra Nuez.

8.3.1 ARM Integration

Test: 3.1		Written by: Alexandra Nuez - Updated 11/18/2025
Subsystem: ARM Integration (C Coding)		Tested by: Alexandra Nuez and Theodore Freij
Equipment Required:	Cora Z7, USB Cable, Program Execution and Serial Communication via PuTTY, Computer with Vitis IDE, PCB with peripheral components (IMU Sensor, OLED Display, H-Bridge IC, Common Cathode LEDs)	
Description: This test validates that through C programming in Vitis, the ARM integration, consisting of all internal and external peripherals, correctly initializes the processing system and configures connected peripherals through AXI registers. This test confirms that the ARM processor can successfully execute initializations, configurations, and communications of SPI, I2C, GPIO, and PWM modules altogether through register access. UART communication through PuTTY will be utilized to output system status messages to the user. Physical demonstration onto the PCB will be utilized to confirm all peripherals are integrated appropriately.		
Overall Results: PASS		

#	Procedure	Expected Result	Actual Result	Comments:
1	Open Vitis workspace and import hardware platform from Vivado and write the ARM integration C code.	Application builds and compiles with no errors.	PASS 	ARM validation is needed to ensure that external and internal peripherals work properly before integrating all into the program.
2	Run a system initialization routine to configure necessary clocks and interrupt controller	No runtime errors, system status printed to PuTTY.	PASS	Confirms PS boot procedure and interrupt readiness.
3	Read and store base addresses of SPI, I2C, GPIO, and PWM modules and validate access through testing.	Valid addresses returned with no errors.	PASS	Pending validation from ARM peripherals.
4	Confirm system initialization and peripheral readiness message through PuTTY.	Message outputs to PuTTY within the startup period.	PASS 	Validates UART configuration and processor output signalling.

5	Perform read/write test transactions for each peripheral (GPIO button test, PWM duty-cycle update through RGBs/motors, SPI loopback/device read from accelerometer, and I2C detection/read on OLED display).	Successful register writes and valid feedback responses with minimal to no errors.	PASS 	Demonstrates full integrated peripheral functions under ARM control onto breadboard/PCB.

Overall result = *PASS*

Notes/Comments: Test 3.1 will be verified through a main program ran in Vitis. The program is a user-control design that cycles through main peripheral functions and options to change configurations/data through on-board buttons.

8.4 ARM Validation (C Coding) Subsystem Unit Tests

The ARM Validation Subsystem Unit Tests is owned by Theodore Freij.

8.4.1 Validation IP Blocks

Test: 4.1		Written by: Theodore Freij *10/16/2025*
Subsystem: ARM Validation		Tested by: Theodore Freij and Alexandra Nuez
Equipment Required:	Cora Z7, USB Cable (Program Execution and Serial Communication via PuTTY, Computer with Vitis IDE, Oscilloscope (Signal Monitoring)	
Description: This test validates the ARM's ability to control all mapped peripherals on the Cora Z7. Using bare-metal C code, the processor initializes each AXI device and confirms proper communication with the LEDs, buttons, PMOD GPIO, I2C OLED display, SPI interface on J7, and both PWM modules. The goal is to ensure that every peripheral responds correctly to register reads and writes, and that the hardware outputs—such as LED states, SPI activity, I2C data transfers, and PWM duty-cycle changes accurately reflect the instructions issued by the ARM through software.		
Overall Results: PASS / FAIL / Could not be completed		

#	Procedure	Expected Result	Actual Result	Comments
1	Write code in Vitis to test the GPIO module through buttons, LEDs and shield pin connections.	BTN0 and BTN1 register digital logic transitions on every press. RGB LEDs respond to software commands with independent red, green, and blue channel control. Shield GPIO pins support tri-state operation and can be configured as either digital inputs or 3.3V outputs.	BTN0 tested with 20 presses and BTN1 tested with 20 presses, with 20/20 successful detections for each button. RGB LEDs were tested across all six individual color channels (LED0 RGB and LED1 RGB), with each channel activating correctly on command. Combined color patterns were also verified visually. Shield GPIO pins configured as outputs measured 3.29V to 3.31V for logic HIGH and 0.00V for logic LOW using a multimeter. Shield GPIO pins were then reconfigured as inputs and correctly detected externally applied	-

			HIGH and LOW logic states over 50 test cycles with an IR sensor connected. No missed responses were observed.	
2	Initialize I2C and communicate with the OLED display to verify correct ACK and data transfer.	OLED responds correctly, ACKs received, data renders properly.	OLED communication successful; characters displayed correctly.	-
3	Validate GPIO communication by toggling LEDs and reading button inputs.	LED outputs change and buttons serve as inputs.	PASS	-
4	Confirm system initialization message through PuTTY.	Message outputs to PuTTY within startup period.	PASS	-
5	Initialize SPI peripheral and verify configuration through status register read.	SPI peripheral initializes successfully with no errors.	PASS	-
6	Transmit a known byte sequence to a loopback or connected SPI slave device.	Data successfully transmitted and received; loopback matches expected pattern.	PASS	-
7	Output SPI initialization and transaction success via UART to PuTTY terminal.	Message confirming successful SPI communication printed in PuTTY.	PASS	-

Overall result = PASS
Notes/Comments: [Placeholder for Test Notes/Comments]

8.5 PCB Design Peripheral Subsystem Unit Tests

The PCB Design Peripheral Subsystem Unit Tests is owned by Marie-Albert Sweeney.

8.5.1 I2C Integration on PCB with OLED Screens

Test: 5.1	Written by: Marie-Albert Sweeney *10/16/2025*
Subsystem: PCB Design Peripheral	Tested by: Marie-Albert Sweeney 2/4/2026
Equipment Required:	Cora Z7, custom PCB, two 0.96-inch OLED screens, USB power source, device capable of running Vitis and PuTTY or equivalent terminal
Description: This test confirms the I2C bus is routed correctly on the PCB, and the OLED screens work properly.	
Overall Results: PASS	

#	Procedure	Expected Result	Actual Result	Comments:

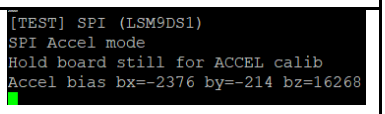
1	Connect the PCB to the Cora Z7 and plug the OLED screens into the PCB. Give the Cora Z7 power via the USB port.	The PCB and the OLED screens fit into their respective spaces. The Cora Z7 turns on.	PASS	The screens were able to fit into the header pins allocated for it and the Cora board turns on, indicated by the power light
2	Run the OLED code in Vitis onto the Cora Z7	The UART sends message to PuTTY asking for user input.	PASS	The proper message was sent to PuTTY.
3	Check OLED screen #1 for OLED display message	OLED screen 1 displays connection message	PASS	
4	Check OLED screen #2 for display message	The OLED screen 2 displays the connection message	PASS	

Overall result = PASS

Notes/Comments: Since both screens turned on and were able to be written to, that indicates they are wired correctly on the PCB. Communication signal testing can be found in test 8.7.1.

8.5.2 SPI Integration on PCB with the LSM9DS1 Inertial Module

Test: 5.2		Written by: Marie-Albert Sweeney *10/16/2025*
Subsystem: PCB Design Peripheral		Tested by: Marie-Albert Sweeney 2/4/2026
Equipment Required:	Cora Z7, custom PCB, Adafruit LSM9DS1 breakout board, USB power source, device capable of running Vitis and PuTTY or equivalent terminal	
Description: This test confirms the SPI bus and GPIO select pins are routed correctly on the PCB to the inertial module.		
Overall Results: PASS		

#	Procedure	Expected Result	Actual Result	Comments:
1	Connect the PCB to the Cora Z7 and plug the inertial module into the PCB. Give the Cora Z7 power via the USB port.	The PCB and the inertial module fit into their respective spaces. The Cora Z7 turns on.	PASS	The inertial module fit into the space for it with the header pins on the PCB
2	Run the inertial module code in Vitis onto the Cora Z7.	The SPI Test with acceleration data is output onto PuTTY	PASS	

Overall result = PASS
Notes/Comments: The output of data to PuTTY from the LSM9DS1 confirms the correct wiring of the SPI communication pins on the PCB.

8.5.3 RGB LED Control with PWM on PCB

Test: 5.3		Written by: Marie-Albert Sweeney *11/21/2025*
Subsystem: PCB Design Peripheral		Tested by: Marie-Albert Sweeney
Equipment Required:	Cora Z7, custom PCB, RGB LEDs, resistors, USB power source, device capable of running Vitis	
Description: This test confirms the PWM control pins (IO0-IO3 and IO26-IO28) are routed correctly to the LEDs and they function properly		
Overall Results: PASS		

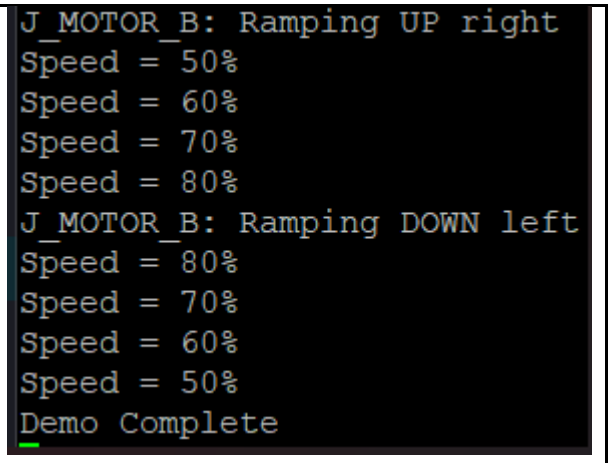
#	Procedure	Expected Result	Actual Result	Comments:
1	Connect the PCB to the Cora Z7 and plug the board into the power source	The Cora Z7 turns on.	PASS	
2	Run the LED PWM code in Vitis onto the Cora Z7	LED 1 and 2 turn on to a single color	PASS	Both LEDS turn on to red
3	Press BTN1 to change brightness of the LEDs with the PWM duty cycle	LEDs change in brightness as the button gets pressed	PASS	The brightness of the red LEDs changes

Overall result = PASS
Notes/Comments: The PWM signals for the RGB LEDs can handle the constantly changing of the colors or brightness when they change, indicating correct wiring and traces.

8.5.4 H-Bridge Motor Driver on PCB with the SN754410NE

Test: 5.4	Written by: Marie-Albert Sweeney *11/21/2025*
Subsystem: PCB Integration	Tested by: Marie-Albert Sweeney 2/4/2026
Equipment Required:	Cora Z7, custom PCB, 12V external power for the motor shield, test motor, multimeter
Description: This test verifies the SN754410NE chip is wired correctly and capable of driving the high-current loads on the Cora Z7	
Overall Results: PASS	

#	Procedure	Expected Result	Actual Result	Comments:
1	Connect the battery to J_MOTOR_PWR and check the voltage at pin 8 on the H-Bridge	Multimeter reads external voltage level at pin 8	PASS	After setting the external voltage to 12V, the voltage at pin 8 also reads 12V
2	Connect motors to J_MOTOR_A and J_MOTOR_B and load the motor driver code onto the Cora Z7	Motors don't automatically spin	PASS	After connecting the motors, they stayed still and didn't receive instruction to start automatically
3	Make Motor A spin forward	Motor A spins forward and backwards	PASS	Pressing BTN0 starts the motor demo where motor 1 spins and comes to a stop. J_MOTOR_A: Ramping UP Speed = 50% Speed = 60% Speed = 70% Speed = 80% J_MOTOR_A: Ramping DOWN Speed = 80% Speed = 70% Speed = 60% Speed = 50%
4	Make Motor B spin forward	Motor B spins forward and backwards	PASS	After motor 1 runs, motor 2 spins, speeding up then slowing down

Overall result = PASS

Notes/Comments: The working functionality of the motors confirms the correct wiring and trace widths on the PCB to allow the signal to travel to the motors.

8.6 Peripherals – SPI – ARM (C Coding) Subsystem Unit Tests

The Peripherals – SPI – ARM Subsystem Unit Tests is owned by Matthew Carter.

8.6.1 SPI ARM Validation and Peripheral Integration

Test: 6.1		Written by: Matthew Carter *10/16/2025*
Subsystem: SPI – ARM (C Coding)		Tested by: Matthew Carter
Equipment Required: Cora Z7 Board	Cora Z7, device capable of running Vivado/Vitis IDE, Accelerometer (SPI), cables to connect peripheral to board, and USB cable to connect Cora Z7 and capable device.	
Description: This test validates the functionality of the **AXI Quad SPI** IP by communicating correctly with a target SPI peripheral. A C program in Vitis, running on the ARM processor, configures SPI mode, clock prescaler, and chip-select, then performs register reads/writes to confirm protocol compliance. The program verifies operation by reading a known device-ID and streaming live data/transaction results to the OLED peripheral		
Overall Results: Pass		

#	Procedure	Testing / Verification	Expected Result	Actual Result	Comments:
1	Connect the Cora Z7 board to a capable device with the USB cable.	Ensure LEDs are on to ensure getting power to the board	Power supplied to the board and SPI Peripheral.	PASS	
2	Boot the Vitis app to initialize AXI Quad SPI: enable master, select correct CPOL/CPHA	Print initialization message to PuTTy confirming SPI setup	SPI core enabled with desired mode/clock; transfers can be issued.	PASS	
3	Perform a basic read using the required command/address sequence.	Read WHO_AM_I Register and compare returned value to datasheet	Returned ID matches the datasheet value for the device.	PASS	Confirms SPI Connections and correct wiring
4	Write essential config registers, then read back to verify.	Write known values to confirmation registers	Read-backs match written values; device enters active/measurement state	PASS	Confirms READ/WRITE Functionality
5	Start a burst read and print results to PuTTY/console or OLED display.	Continuously read Output register while physically moving the peripheral	Streaming numeric data appears at the expected rate; values change when the device is moved	PASS	Confirms Real time SPI outputs
6	Switch between SPI Chip select and read respective WHO_AM_I registers for Accelerometer and magnetometer		Each device responds with their correct ID only when chip select is active	PASS	Confirms proper chip select routing and multi-device SPI operation

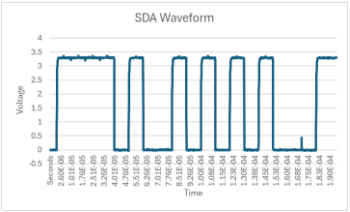
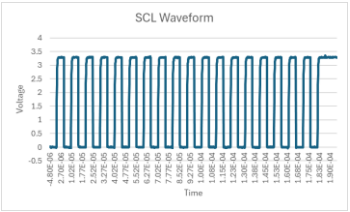
Overall result = PASS
Notes/Comments:

8.7 Peripherals – I2C – ARM (C Coding) Subsystem Unit Tests

The Peripherals – I2C – ARM Subsystem Unit Tests is owned by Miguel Octavo Castro.

8.7.1 OLED I2C Screen Integration

Test: 7.1	Written by: Miguel Octavo Castro *10/16/2025*
Subsystem: I2C-ARM (C coding)	Tested by: Miguel Octavo Castro 11/01/2025
Equipment Required:	Cora Z7, device capable of running Vivado/Vitis IDE, OLED peripheral (I2C), cables to connect peripheral to board, and USB cable to connect Cora Z7 and capable device.
Description: This test validates the functionality of the AXI I2C IP by communicating to the OLED properly. This is controlled and done through C code done on Vitis, which runs on the ARM processor. The program ensures I2C communication protocols are followed and gets user inputs via Putty, which then echo those inputs onto the OLED device. This test was done a minimum of 10 times to confirm total functionality before handing the code to the ARM integration subsystem.	
Overall Results: PASS	

#	Procedure	Expected Result	Actual Result	Comments:
1	Connect the Cora Z7 to a capable device using the USB cable and connect the OLED peripheral using cables.	Power is supplied to the board, and OLED data lines are connected.	PASS	Proper connections between devices and peripherals must be done correctly to ensure proper communication and power is given.
2	Ensure that proper I2C communication protocols are being done.	START and STOP conditions are done, and ACK/NACK bits are being read after every byte.	PASS	Probing both SDA and SCL lines confirms proper I2C communication protocols.  
3	Initialize both the I2C and the OLED device correctly through ARM software on Vitis.	The I2C interface and its peripheral are ready to communicate to each other.	PASS	Must be done to properly communicate to each other. For the OLED, proper initialization should be done according to its data sheet recommendations.
4	Get user input from Putty.	Putty will prompt user to “Type into	PASS	Proper settings must be applied to the Putty terminal

		PuTTY, characters will echo onto OLED: “Then user will type desired characters.		to receive correct character inputs.
5	Echo inputs onto OLED.	All character user inputs are displayed on the OLED.	PASS	Validates the functionality of I2C communication.
6	OLED works like a functional additional terminal/screen.	Inputs like backspace, enter, and all uppercase and lowercase letters, as well as all special characters are displayed	PASS	Shows proper functionality of OLED user interface.

Overall result = PASS

8.8 Peripherals – PWM – ARM (C Coding) Subsystem Unit Tests

The Peripherals – PWM – ARM Subsystem Unit Tests is owned by Aaron Ong.

8.8.1 PWM Cora Z7 Test

Test: 8.1		Written by: Aaron Ong *10/16/2025*
Subsystem: PWM-ARM (C coding)		Tested by: Aaron Ong 11/06/2025
Equipment Required:	Cora Z7 board, USB cable, device capable of running Vivado/Vitis IDE, breadboard, jump wires, oscilloscope with probes	
Description: This test validates the correct operation of the custom PWM IP integrated with the ARM processor. The goal is to verify that AXI register writes correctly configure the PWM period and duty-cycle threshold, and that the resulting output waveform appears correctly on the GPIO pins. Oscilloscope measurements are used to confirm the programmed PWM frequency and duty-cycle timing. This verifies register control, proper counter operation, and correct routing of PWM outputs through the GPIO pins.		
Overall Results: PASS		

#	Procedure	Expected Result	Actual Result	Comments
1	Connect the Cora Z7 board to the PC using the USB cable and power it on.	Board powers on and is ready for programming.	PASS	Ensured hardware was powered and ready for testing.
2	Program the FPGA with the Vivado-generated bitstream and launch the Vitis application.	PWM IP initializes successfully and software can access the mapped registers.	PASS	Confirmed AXI interface and IP mapping were correct.
3	Connect the selected PWM GPIO output pin and ground to the oscilloscope using the breadboard/test setup.	Oscilloscope is able to display the PWM waveform from the selected output pin.	PASS	Confirmed correct physical routing from GPIO pin to measurement setup.
4	Write the PWM period register for a 1.00 kHz frequency output using a 50 MHz system clock.	Measured frequency remained at 1.00 kHz across multiple trials, confirming stable PWM timing.	PASS	Five oscilloscope measurements confirmed that the PWM output frequency remained centered at 1 kHz with a measured variation of ±0.03 kHz across repeated trials.
5	Enable the PWM output channel through ARM C software.	PWM waveform appears on the selected GPIO output pin.	PASS	Verified enable register function and output activation.
6	Set the duty cycle to 25% in software and measure the waveform on-time.	On-time is measured at 25% of the full period.	PASS	25% test: Measured on-time = 0.25 ms ± 0.004 ms, corresponding to 25.0% ± 0.4%

7	Set the duty cycle to 50% in software and measure the waveform on-time.	On-time is measured at 50% of the full period.	PASS	50% test: Measured on-time = 0.50 ms ± 0.004 ms, corresponding to 50.0% ± 0.4%
8	Set the duty cycle to 75% in software and measure the waveform on-time.	On-time is measured at 75% of the full period.	PASS	75% test: Measured on-time = 0.75 ms ± 0.004 ms, corresponding to 75.0% ± 0.4%
9	Compare measured period and on-time values against expected results.	Period remains constant at 1 ms while on-time changes according to duty cycle.	PASS	Frequency and on-time results indicate stable PWM timing. The output remained centered at 1.00 kHz ± 0.03 kHz, corresponding to an output period of approximately 1.00 ms ± 0.03 ms.

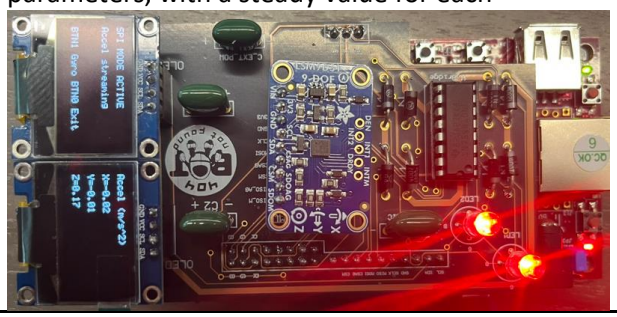
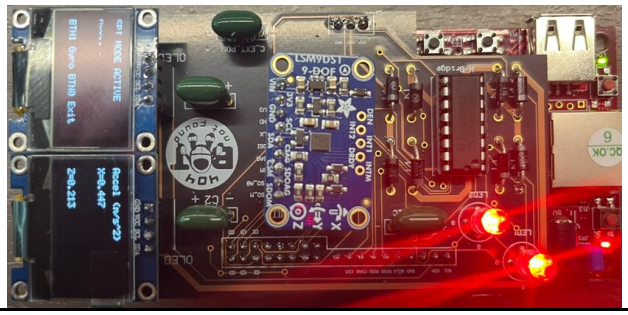
Overall result = PASS

8.9 Overall System Integration Unit Tests

The Overall System Integration Unit Tests is owned by Marie-Albert Sweeney.

8.9.1 Inertial Module to OLED Display on PCB Test on PCB

Test: 9.1	Written by: Marie-Albert Sweeney *10/16/2025*
Subsystem: PCB Integration	Tested by: Marie-Albert Sweeney, Matthew Carter, and Miguel Octavo Castro 2/4/2026
Equipment Required:	Cora Z7, PCB with the LSM9DS1 inertial module and OLED screens on the board, device capable of running Vitis, power source
Description: This test verifies the SPI and I2C components can both run on the PCB and speak to each other.	
Overall Results: PASS	

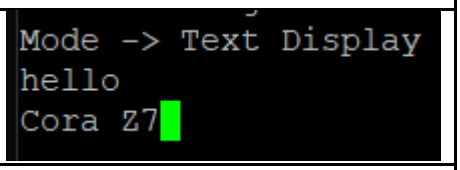
#	Procedure	Expected Result	Actual Result	Comments:
1	Plug the Cora Z7 into the power source and run the inertial module/display integrated code.	The board turns on, and the code builds/runs.	PASS	
2	While keeping the board flat, the accelerometer x-axis value is output to an OLED screen.	The display screen shows a value at or close to 0.	PASS	The OLED screen displays each of the x-, y-, and z-parameters, with a steady value for each 
3	Tilt the board in different directions to get changing x-, y-, and z-axis values.	The OLED screen displays the changes in the axes as the board moves.	PASS	The values on the screen updates as the board moves 

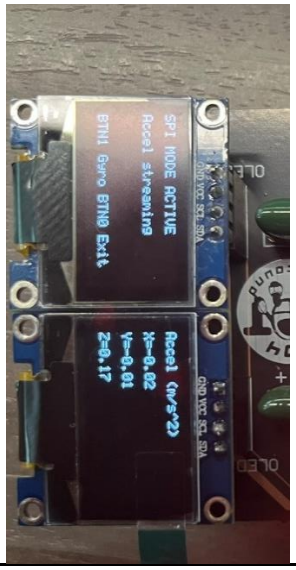
Overall result = PASS

Notes/Comments: Since the accelerometer data was displayed on the OLED screen, we were able to confirm that the SPI IMU and OLED screens were successfully incorporated to work together.

8.9.2 Full Peripheral Integration on PCB with Switching Modes

Test: 9.2		Written by: Marie-Albert Sweeney *11/21/2025*
Subsystem: PCB Integration		Tested by: Marie-Albert Sweeney and Alexandra Nuez 2/4/2026
Equipment Required:	Cora Z7, fully assembled PCB with all peripherals, USB connection to PC, serial terminal connection to PuTTY 12V external power	
Description: This test verifies the operation of the complete system with the fully assembled PCB and integration code. BTN0 cycles through 5 modes of operation, and BTN1 is used for the specific action of the set mode.		
Overall Results: PASS		

#	Procedure	Expected Result	Actual Result	Comments:
1	Plug the Cora Z7 into the PC, turn on the board, and run the integration code	OLED 1 displays MODE: Color Select, OLED 2 displays the configuration information for mode	PASS	
2	Press BTN1 to cycle through the set colors	The color name on OLED 1 changes and the colors displayed on the LEDs match	PASS	The screen updates the color as they change on the LEDs
3	Press BTN0 to switch to Brightness mode	OLED 1 displays the brightness mode for the board	PASS	
4	Press BTN1 to adjust brightness.	The brightness % increases/decreases on OLED 2 as the brightness of the LED changes	PASS	The brightness of the LEDs changes in 10% increments
5	Press BTN0 to switch to Text Display mode. Open PuTTY	OLED 1 shows Text Display mode and OLED 1 and PuTTY prompt to enter text	PASS	
6	Type characters into PuTTY.	Text appears under prompt onto OLED screen #1	PASS	Text input works and are able to backspace and enter a new line
7	Press BTN0 to switch to Motor Control mode	OLED 1 shows Motor Control mode and says BTN1 = Start Demo	PASS	

8	Press BTN1 to run the motor demo.	The motor runs at varying PWM duty cycles	PASS	Each motor goes through a cycle of ramping up then slowing down J_MOTOR_A: Ramping UP Speed = 50% Speed = 60% Speed = 70% Speed = 80% J_MOTOR_A: Ramping DOWN Speed = 80% Speed = 70% Speed = 60% Speed = 50% J_MOTOR_B: Ramping UP right Speed = 50% Speed = 60% Speed = 70% Speed = 80% J_MOTOR_B: Ramping DOWN left Speed = 80% Speed = 70% Speed = 60% Speed = 50% Demo Complete
9	Press BTN0 to switch to SPI Mode.	OLED 1 shows SPI Mode	PASS	
10	Press BTN1 to cycle through the SPI sensor modes	The screen on OLED 2 cycles from the accelerometer to the gyroscope to the temperature sensor to the magnetometer	PASS	SPI mode can cycle through each mode and display the changing data stream 
11	Press BTN0 to exit SPI Mode.	System goes back to Color Select mode	PASS	The code loop restarts how it is supposed to

Overall result = PASS

Notes/Comments: The full integration code was tested on the PCB to ensure that while each individual subsystem works together, the communication protocols were still able to function correctly. Further testing for the code section of this can be found in test 8.3.1.

9 HDL FPGA Standard Operation Procedure

The HDL FPGA Standard Operation Procedure was written by Farrah Balbuena.

The Standard Operation Procedure[1] is essential to ensuring consistent, reliable, and safe operation of the system by providing a clear, standardized set of instructions for users to follow. It minimizes variability in performance, reduces the likelihood of user error, and supports efficient troubleshooting and maintenance. By adhering to the Standard Operation Procedure, users can achieve repeatable results while maintaining compliance with project and safety requirements. To use this document effectively, users should review the procedure in its entirety before operation and follow each step in the prescribed sequence. This structured approach ensures proper system use and maximizes both performance and longevity of the product.

[1] SOP-OP. PROC 2.07

10 Appendix B: Autonomous Bot Executive Summary

The Executive Summary was written by Farrah Balbuena.

Our product is a low-cost FPGA development board that uses Hardware Design Language (HDL) to implement and develop IP blocks for the Cora Z7, improving the educational environment for digital systems instructions. This platform supports future curricula for EE 4321, the "Digital Systems using HDL", enabling students to develop software and hardware skills while applying concepts to real-world embedded systems using industry-standard tools.

The need for this product stems from the proven benefits of experiential and robotics-based learning in engineering education. Research published in the *International Journal of STEM education* reports that educational robotics produces moderate to large positive effects on learning performance ($g \approx 0.665$), indicating that students in robot-assisted environments outperform most peers in traditional classrooms [1]. By pairing the development board with an autonomous educational bot, students gain a learning-by-teaching experience in which they configure, test, and "teach" the system through peripheral programming and control logic. This approach strengthens conceptual understanding, engagement, and retention while preparing student for careers in embedded systems, hardware design, and digital logic.

The team will research the capabilities of the Cora Z7 and implement their findings using Xilinx applications. Development will include creating block diagrams in Vivado [11] to integrate IP protocols and peripherals such as SPI, I2C, PWM, and GPIO. Testbenches will be designed to simulate and validate functionality. All project work will be conducted at Texas State University. By the end of Design Phase 1 (D1), the team will have a functioning SPI interface. By the end of Design Phase 2 (D2), the team will have functioning IP blocks and test benches for the FPGA, a fully documented ARM code demonstrating functional peripherals, a completed PCB, and a fully functional educational board. As a second goal, the team will integrate an autonomous HDL- programmed bot that can demonstrate and extend these concepts through interactive, physical operation.

Team Member	Subsystem
Aaron Ong	Movement Control
Luis Leon Pineda	IP Iteratations, Testbenches for IPS & Chassis Design
Farrah Balbuena	Testbenches for IPS (HDL)
Marie-Albert Sweeney	PCB Design Peripherals
Theodore Freij	ARM Validation & Movement Control
Matthew Carter	Movement Control
Miguel Octavo Castro	I2C OLED Screen & Visuals (C Coding)
Alex Nuez	ARM Integration (C Coding)

Table 1: Preliminary Subsystem Responsibilities

[1] https://link.springer.com/article/10.1186/s40594-024-00469-4 accessed 2/4/2026

11 Appendix C: Autonomous Bot Top-Level Diagram

The top-level block diagram was drawn by Alexandra Nuez.

The top-level diagram of our system is shown in Figure 1. This diagram illustrates the complete integration of the hardware and its peripherals implemented on the Cora Z7. The procedure begins with the HDL modules and test benches, defining and verifying hardware functionality before being mapped into memory and connected to the ARM processor through AXI interconnect. The ARM core is responsible for handling communication with peripherals such as SPI, I2C, GPIO, and PWM which is implemented through programmable logic and interfaced with external PCB components. Inputs consist of sensor data from the ToF (VL53L0X) and IR sensors (TCRT5000) along with button signals which are all processed by the ARM processor. This generates outputs that control the OLED display and DC motors through the H-Bridge. Additionally, the system enables interaction between the FPGA, PCB, and user, where sensor data influences movement control decisions. The project is divided among eight team members who are responsible for specific subsystems listed in Figure 11. With each responsibility, the whole team ensures reliable coordination for hardware development, testing, and integration across both the FPGA and PCB components for a complete and functional system.

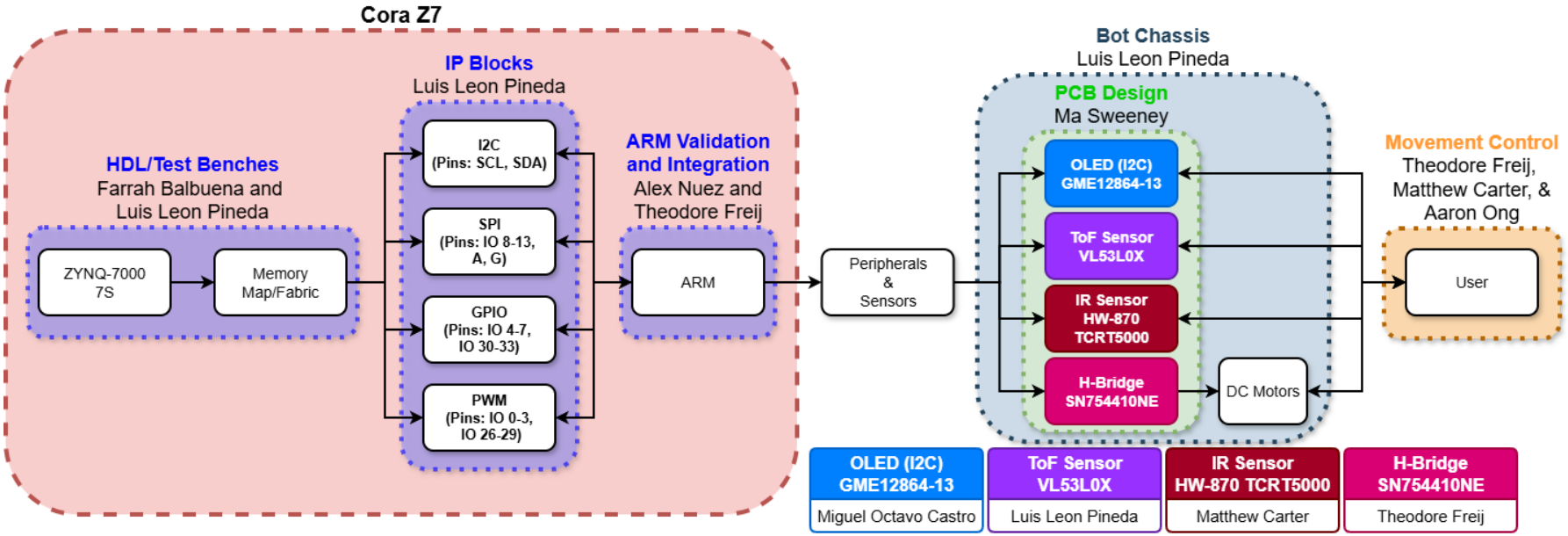


Figure 11: Top-level Block Diagram

12 Appendix D: Functional Description

12.1 Movement Control Subsystem

The Movement Control Subsystems are owned by Theodore Freij, Matthew Carter, and Aaron Ong.

The Bot's movement control subsystem provides an autonomous way of moving within a bounded arena using sensors and motors tied together. This is achieved through 4 PWM controlled DC motors driven by 2 H-Bridges connected to omnidirectional wheels which enables the bot to manoeuvre in any direction/orientation. The motion commands were simplified into functions that is called to trigger motor sequence initiating movement (forward, backward, left, right).

To fulfil autonomy, movement decisions are determined based on the sensor feedback with commands being executed to adapt and satisfy all conditions.

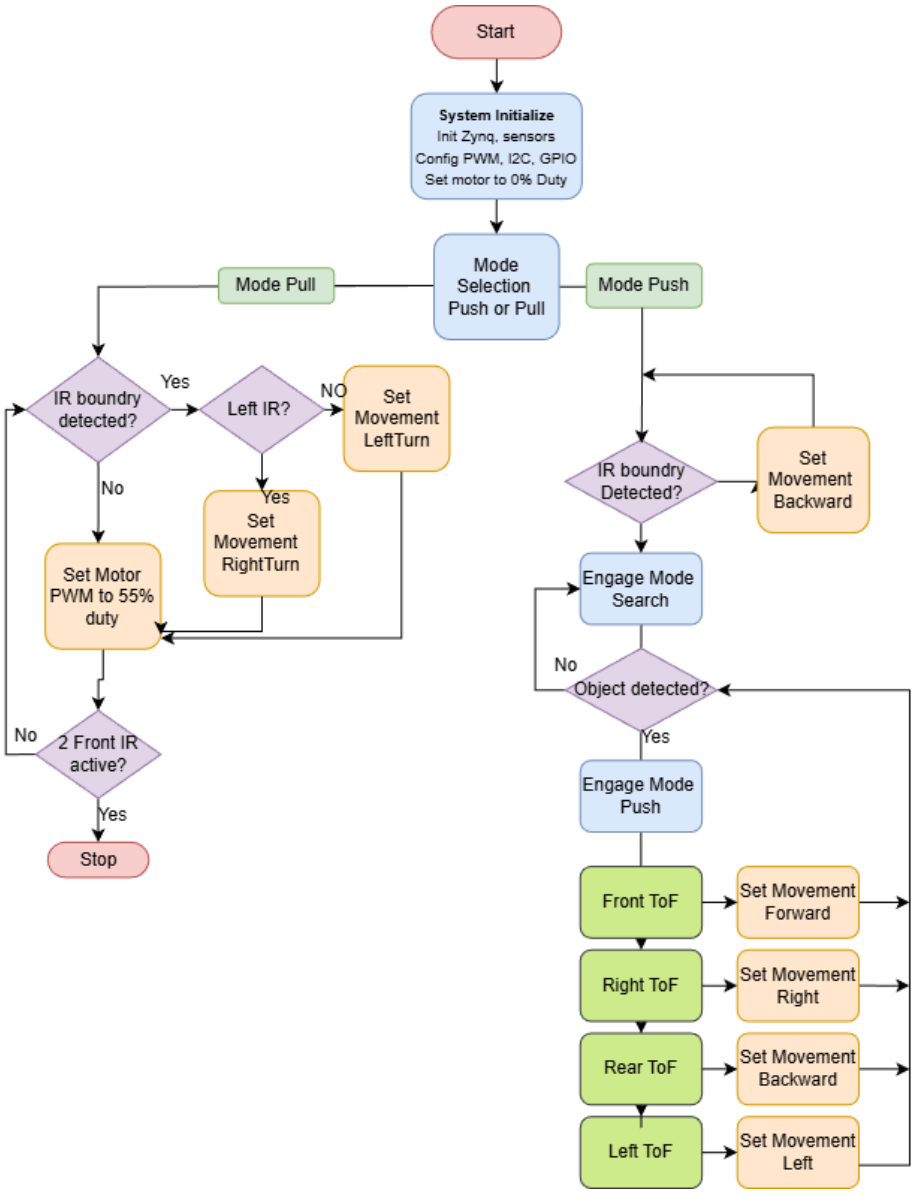


Figure 12: Code Flowchart of movement modes

12.1.1 Sensing and Environmental Awareness

The Bot uses sensors distributed around the chassis to map out its surroundings and perform movement decision. This includes:

- Time-of-Flight (ToF) Sensor:
  Four ToF sensors are places on each side of the chassis to measure the distance between the bot and an object. When operating these sensors are detecting objects only within a predetermined threshold to ensure objects outside the boundary are not detected.

- Infrared Sensors:
  The IR sensors are places on each corner of the chassis which enables the bot to detect the boundaries and trigger a reaction accordingly.

Multiple samples are recorded from sensors to ensure detection and reduce the chances of unstable or random movement.

12.1.2 Control Logic and decision Frame

The system implements a hierarchical control strategy that governs how sensor inputs are translated into motion commands.

- Priority-Based Control:
  Boundary detection has the highest priority. When an IR sensor detects a boundary condition, all current motion commands are overridden, and corrective maneuvers are executed to maintain safe positioning within the boundary

- Obstacle Evaluation:
  In the absence of boundary conditions, ToF sensor data is evaluated against predefined distance thresholds to classify obstacle presence and direction (front, left, right, or rear).

This layered decision-making approach ensures safe operation while enabling effective interaction with the environment.

12.1.3 Boundary Detection and Response

Boundary detection is performed using IR sensors positioned to monitor the edges of the robot's operating area. When a boundary condition is detected, the system immediately initiates an evasive response.

Boundary response behaviors include:

- Reversing away from the detected edge
- Turning away from boundary direction
- Repositioning the Bot to the center of the lane when mode pull is initiated

These actions are executed using reduced motor duty cycles to maintain control and prevent overshooting. Boundary handling is enforced as a safety function and overrides all other behaviors.

12.1.4 Obstacle Detection and Interaction

Obstacle detection is achieved using ToF sensors, which provide directional distance measurements to nearby objects.

When an object is detected within ToF range:

- The system determines the relative direction of the object and samples it
- A forward motion command is issued to engage and displace the object

Different motion profiles are used when bot is running:

- Search behavior: moderate speed scanning for objects
- Engagement behavior: increased duty cycle for pushing force

This section enables active interaction with external objects while maintaining overall system stability.

12.1.5 Operating Modes

The Bot is configured with 2 modes of operation can be used depending on the mode needed. Each mode uses the same sensors but applies them with slightly different constraints.

The two modes:

- Sumo Bot (push mode):
  In this mode the bot is actively searching for objects within the specified range of the ToF sensors and engages them. Using the inputs from the ToF sensors the system detects an object and samples it to the corresponding sensor, when enough samples have been picked up (also a predetermined value) the bot turns to face the object and pushes objects off. The IR boundary detection is always active ready to override any movement taking place to avoid leaving the arena.
- Tug Bot (Pull Mode):
  In this mode the bot is attached to a weighted block and must pull it to the finish line while maintaining positions in the lane. The IR sensors are used to detect the lines and provide corrective measures preventing the bot from crossing the lines. Motor output is stable to ensure traction as reaching the finish line is prioritized over aggressive movement.

12.1.6 System initialization and Safety

Upon startup, the system initializes all subsystems, including motor drivers, sensors, and communication interfaces. Motors are set to a safe (disabled or zero-duty) state during initialization to prevent unintended motion.

Safety considerations include:

- Default motor shutdown on invalid or undefined states
- Boundary override protection at all times
- Sensor validation to prevent random movement

These measures ensure reliable and safe operation throughout all modes.

12.2 Chassis Design Subsystem

The Chassis Design Subsystem is owned by Luis Leon Pineda.

The Chassis Design Subsystem provides the structural foundation and mechanical support for the autonomous bot platform. In Figure 13 we can observe the chassis design assembly, this process was done using SolidWorks.

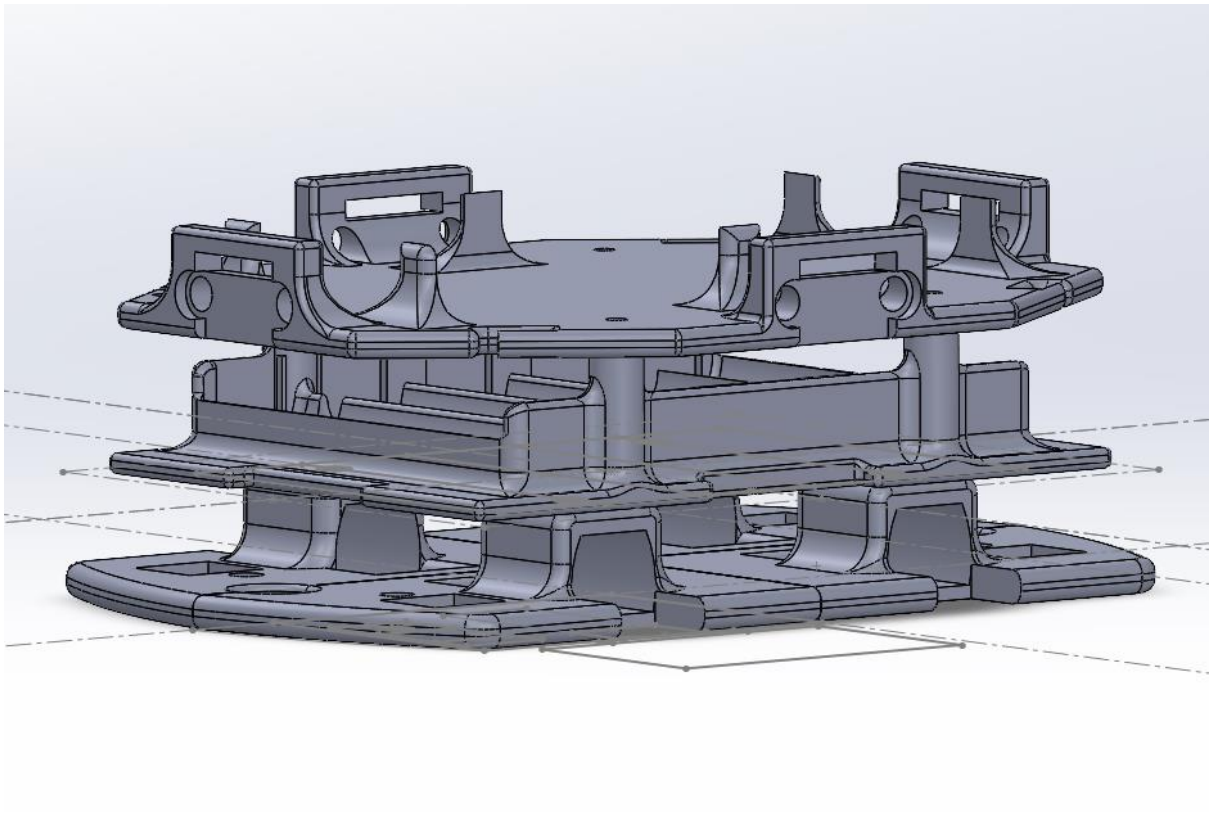


Figure 13: The Assembly of the 3D model of the Bot Chassis

The chassis design comprises three layers that house different arrays of components. On the Zchassis's lower layer, we have the FingerTech Robotics motors and the IR sensors TCRT5000 for boundary detection. The middle layer of the chassis houses our Battery array, consisting of five 18650 9900 mAh 3.7V batteries. The top layer houses the four Time of Flight (VL53L0X) sensors to aid object detection, along with the Digilent Cora Z7 and the custom PCB.

The primary function of the chassis is to protect the Cora Z7 board while allowing the bot to detect, push, and avoid obstacles.

12.3 IP Integration Autonomous Bot

The IP Integration Subsystem is owned by Luis Leon Pineda.

The same procedure used for the first stage of the project is utilized for this subsection. By integrating the IP blocks from AMD/Vivado and Digilent, a bitstream file is generated for the new array of peripherals. The block design produced by this new integration is shown in Figure 14.

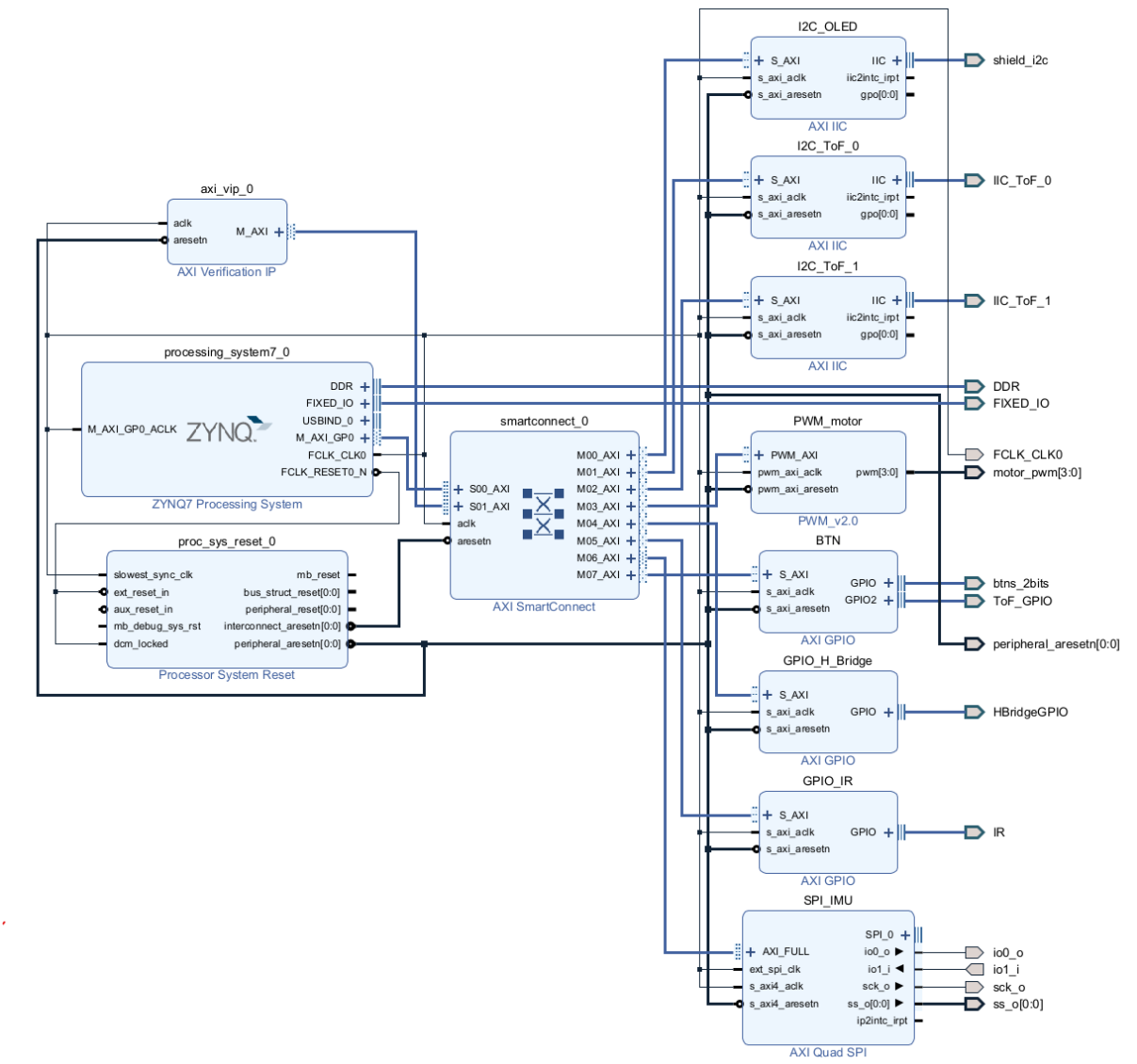


Figure 14: IP integration block design.