Creating and deploying an open standard SPI on an FPGA, involving C based test case development for emulation

MELZG628T: Dissertation

by

SELVA KUMAR S

2022HT80170

Dissertation work carried out at

Qualcomm Technologies Pvt. Ltd, Bengaluru

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI (RAJASTHAN)

April 2024

Creating and deploying an open standard SPI on an FPGA, involving C based test case development for emulation

MELZG628T: Dissertation

by

SELVA KUMAR S

ID No. 2022HT80170

Dissertation work carried out at

Qualcomm Technologies Pvt. Ltd, Bengaluru

Submitted in partial fulfillment of MTech. Microelectronics degree programme

Under the Supervision of
Siva Selvamani and Staff Engineer,
Qualcomm Technologies Pvt. Ltd, Bengaluru

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)
April 2024

CERTIFICATE

This is to certify that the Dissertation entitled "Creating and deploying an open standard SPI on an FPGA, involving C based test case development for emulation" and submitted by Selva Kumar S having ID-No. 2022HT80170 for the partial fulfillment of the requirements of MTech. Microelectronics degree of BITS embodies the bonafide work done by him under my supervision.

Signature of the Supervisor

Place: Bengaluru
Date: 30.04.2024

Siva Selvamani
Staff Engineer,
Qualcomm Technologies Pvt. Ltd,
Bengaluru

Birla Institute of Technology & Science, Pilani

Work-Integrated Learning Programmes Division

Second Semester 2023-2024

MEL ZG628T: Dissertation

ABSTRACT

BITS ID No. : **2022HT80170**

NAME OF THE STUDENT : **SELVA KUMAR S**

EMAIL ADDRESS : **2022ht80170@wilp.bits-pilani.ac.in**

STUDENT'S EMPLOYING ORGANIZATION & LOCATION : **Qualcomm Technologies Pvt. Ltd, Bengaluru**

SUPERVISOR'S NAME : **SIVA SELVAMANI**

SUPERVISOR'S EMPLOYING ORGANIZATION & LOCATION : **Qualcomm Technologies Pvt. Ltd, Bengaluru**

SUPERVISOR'S EMAIL ADDRESS: **sselvama@qti.qualcomm.com**

DISSERTATION TITLE : **Creating and deploying an open standard SPI on an FPGA, involving C based test case development for emulation**

ABSTRACT:

IP qualification at the initial phase is the crucial part of SoC development. In the fast phased industry, Quality assurance comes with the penalty of time to market. As part of quality assurance, the newly designed or customized IP will be tested well and then integrated into SoC. This project deals with one of the methodologies that is used in the industry to qualify the IP.

This project is based on verifying the functionality of a custom IP in an KR26 based FPGA platform. This SOM is almost equivalent to XC7A100T general purpose FPGA device in terms of LUTs, Flipflops and BRAM resources. Based on this data, resource planning is made in this dissertation. Further technical details about the SOM are discussed in the Hardware Requirements section.

First phase of the project is to bring up the eco system which is required to verify the functionality of any custom IP (i.e. Custom SPI in this project). Eco system is nothing but the Processor subsystem, Memory Subsystem, Clocking and reset and other basic peripherals like Timer, UART and GPIO.

Second phase of the project will be integrating the Design Under Test (DUT) with the platform, writing the C based testcase for the integrated Eco system. The development of testcases includes the understanding of different things like system specifications, system use cases, System interfaces and members. Once the requirement of DUT is understood in the second phase, will proceed with the C based Drivers and testcase development.

The core ideology of the project is to take a custom SPI IP and integrate it with the small SoC like ecosystem. Emulate this ecosystem in the FPGA and writing the C based testcases to verify the functionality of the IP. SPI is a communication protocol, which is used to interface with the External non-volatile memory like serial flash. To demonstrate this use case, we are using Winbond w25q64 SPI flash, that is interfaced with the Custom IP in fast read mode.

Key Words: IP Emulation, Embedded Standalone Application, SPI, Validation
Project Areas: FPGA based IP Emulation

Signature of the Student

Name: SELVA KUMAR S

Date: 30.04.2024

Place: Bengaluru

Signature of the Supervisor

Name: SIVA SELVAMANI

Date: 30.04.2024

Place: Bengaluru

Contents

Table of Figures

1. Functional Block Diagram

The functional block diagram of FPGA based emulation environment that supports the IP validation is given below.

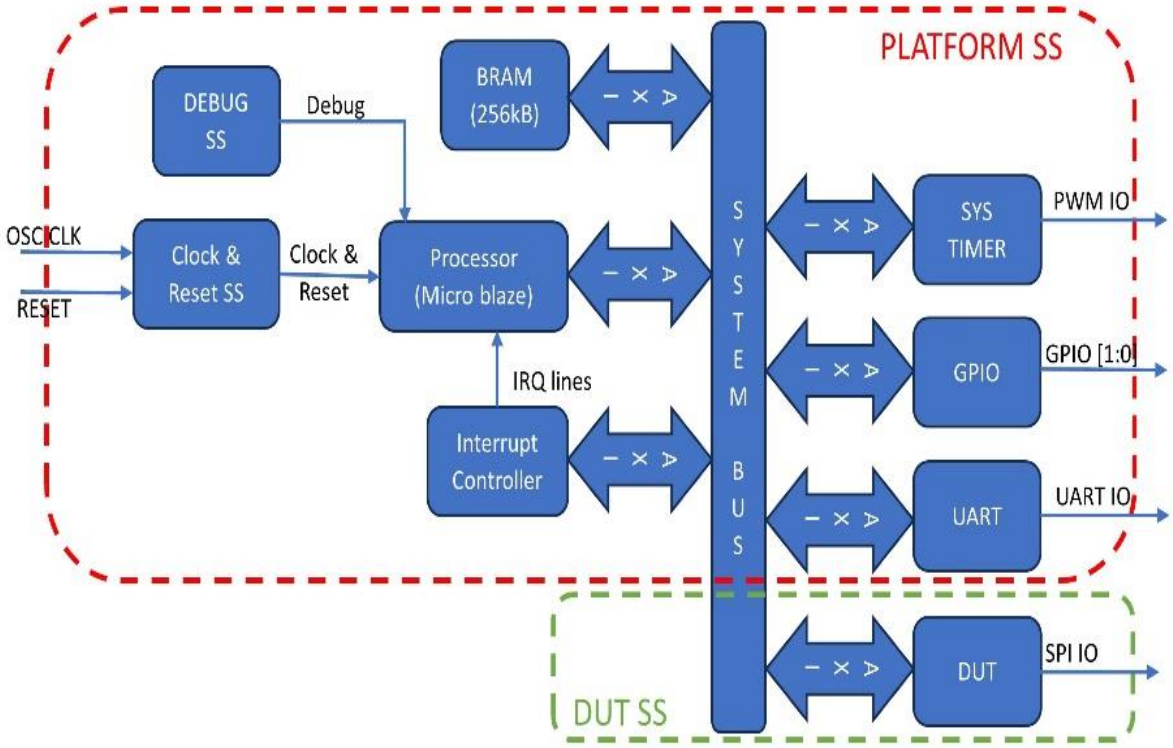


Figure 1 Block diagram of the SoC

Entire SoC is divided into two major subsystem which are,

1. Platform Subsystem

   This subsystem consists of all major components of SoC like Processor (Micro blaze), Memory (BRAM), Interrupt Controller, Clock PLL and Reset System, Debug subsystem, System tick timer, GPIO, UART, System Bus (AXI Interconnect). All these components are connected to each other via AXI Interconnect through a 32-bit AXI bus.

2. DUT Subsystem

   This subsystem contains the targeted Custom IP to be qualified in Emulation Platform. In our case, it's a AXI to QSPI IP. This is customized to interface only with Standard SPI with only one slave interface and FIFO mode.

Xilinx IP used in Platform and DUT Subsystem

Xilinx IP Used	Description	Address map	Size
clk_wiz:6.0\	PLL to generate different clock domain for the SoC	NA	
microblaze:11.0\	Processor system to execute the test code	NA	
mdm:3.2\	Processor debug interface for code JTAG	NA	
proc_sys_reset:5.0\	Processor and SoC reset system	NA	
axi_gpio:2.0\	GPIO module to toggle the Application alive LED	0x40000000	1kB
axi_bram_ctrl:4.1\	AXI BRAM controller to connect RAM to Sys Bus	NA	
blk_mem_gen:8.4\	BRAM block to have array of system memory	0xC0000000	512kB
xlconcat:2.1\	Multiplex to concatenate the Interrupt from different IP	NA	
axi_intc:4.1\	AXI Interrupt Controller to aggregate the IRQs	0x41200000	1kB
axi_uartlite:2.0\	AXI UART IP to display the debug prints in Code	0x40600000	1kB
system_ila:1.1\	Integrated Logic analyzer to probe the RTL signals	NA	
axi_timer:2.0\	AXI Timer to generate the general delay functions	0x41C00000	1kB
lmb_v10:3.0\	Processor Cache memory generator	0x00000000	16kB
axi_quad_spi:3.2\	Targeted Custom SPI block which is to be tested	0x44A00000	4kB

Table 1 List of IPs used, and its main macros assigned

IP wise resource utilization in the FPGA is given in Figure 2. This help us to understand how each IP in the design is implemented and how much resource it consumed.

Hierarchy

Name	CLB LUTs (117120)	CLB Registers (234240)	CARRY8 (14640)	F7 Muxes (58560)	CLB (14640)	LUT as Logic (117120)	LUT as Memory (57600)	Block RAM Tile (144)	Bonded IOB (189)	HPIOB_M (58)	HPIOB_S (58)	HDIOB_M (35)	HDIOB_S (35)	GLOBAL CLOCK BUFFERs (352)
Microblaze_MPU_wrapper	5716	5954	65	117	1192	5046	670	142	13	3	1	5	4	4
Microblaze_MPU_i (Microblaze_MPU)	5716	5954	65	117	1192	5046	670	142	0	0	0	0	0	4
axi_bram_ctrl_0 (Microblaze_MPU_a	202	239	0	2	69	202	0	0	0	0	0	0	0	0
axi_bram_ctrl_0_bram (Microblaze_	89	5	0	0	81	89	0	128	0	0	0	0	0	0
axi_bram_ctrl_0_bram1 (Microblaze_	4	0	0	0	3	4	0	8	0	0	0	0	0	0
axi_bram_ctrl_1 (Microblaze_MPU_a	225	241	0	2	64	225	0	0	0	0	0	0	0	0
axi_gpio_0 (Microblaze_MPU_axi_gp	63	103	0	0	25	63	0	0	0	0	0	0	0	0
axi_intc_0 (Microblaze_MPU_axi_intc	81	90	0	0	20	81	0	0	0	0	0	0	0	0
axi_quad_spi_0 (Microblaze_MPU_a	386	624	0	0	99	366	20	0	0	0	0	0	0	0
axi_smc (Microblaze_MPU_axi_smc_	2413	2725	0	4	508	1994	419	0	0	0	0	0	0	0
axi_timer_0 (Microblaze_MPU_axi_ti	291	244	40	0	77	291	0	0	0	0	0	0	0	0
axi_uartlite_0 (Microblaze_MPU_axi_	98	108	0	1	26	80	18	0	0	0	0	0	0	0
clk_wiz_0 (Microblaze_MPU_clk_wiz_	0	0	0	0	0	0	0	0	0	0	0	0	0	1
mdm_0 (Microblaze_MPU_mdm_0_	88	110	0	0	25	81	7	0	0	0	0	0	0	3
microblaze_0 (Microblaze_MPU_mic	1538	1299	25	108	319	1333	205	6	0	0	0	0	0	0
microblaze_0_axi_periph (Microblaze	233	132	0	0	80	233	0	0	0	0	0	0	0	0
rst_clk_wiz_0_100M (Microblaze_MP	14	34	0	0	6	13	1	0	0	0	0	0	0	0
xlconcat_0 (Microblaze_MPU_xlconca	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2 Utilization Report of Design Implemented in KR26 SOM board

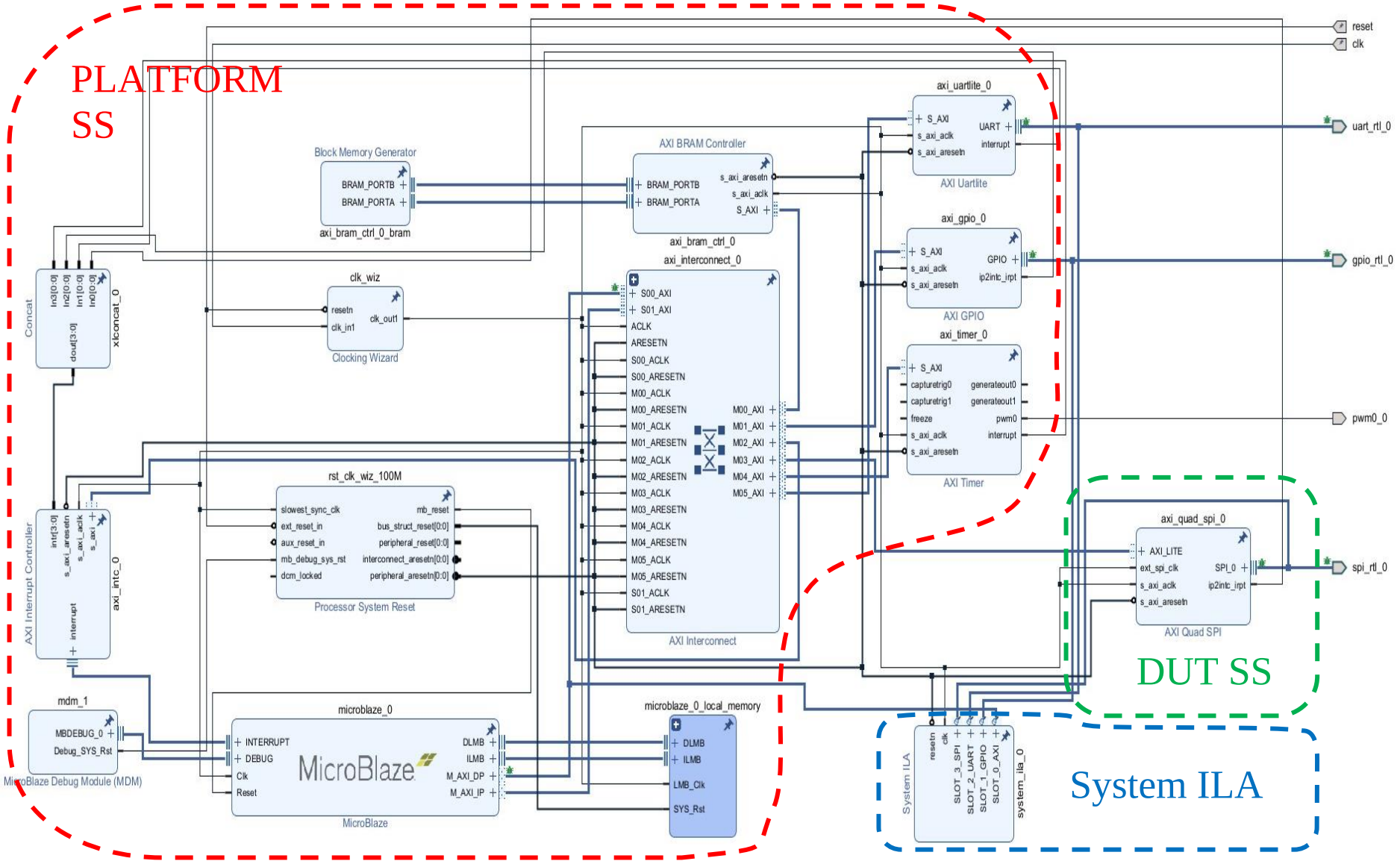


Figure 3 Xilinx IP Integrator Block Design

2. Hardware and Software Requirements

The below table contains the Hardware and Software Requirements for developing an FPGA based emulation environment for Custom SPI IP.

Sl. No	Hardware Required	Software Required
1.	KR26 based SOM	Xilinx Vivado 2023.1
2.	FPGA Carrier Card	Xilinx Vitis 2023.1
3.	W25Q64 SPI Daughter card	SPI Protocol Analyzer
4.	USB to UART FTDI adaptor	
5.	Logic Analyzer	

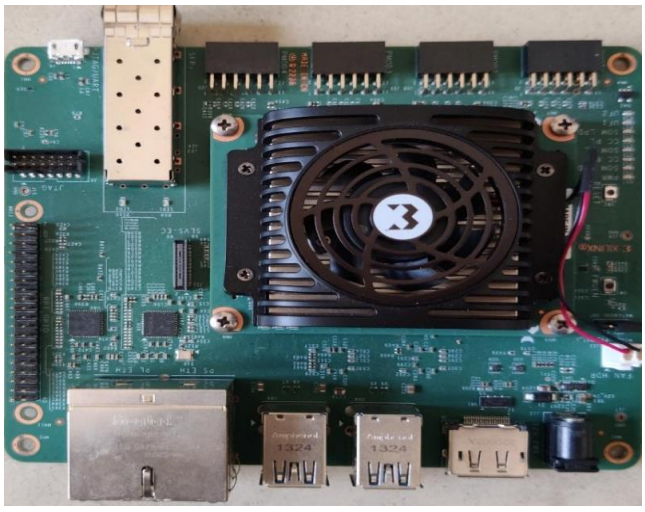
Table 2 Hardware and Software Requirements

2.1. Hardware Requirements

2.1.1. KR26 FPGA SOM

KR26 is a FPGA SoM which can be used with custom carrier card. This FPGA have the following specification, 189 IO's, 117120 LUT's, 234240 FF's, 144 BRAM, 64 URAM, 1248 DSP slices. This FPGA part is suitable for small Microcontroller implementations.

2.1.2. FPGA Carrier Card

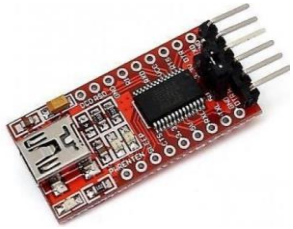


Xilinx custom carrier card is used to expose the IOs of this FPGA. This carrier card has all the peripheral interfaces and its respective slave devices. It has 4 PMOD connectors to connect FTDI adaptor and SPI Flash daughter card. This has two user LEDs which is used as Notification LED. Reset pin is tied to high by default. It has 1 SFP cage, 4 USB connectors, 4 RJ45 connectors, 1 Display port that can be used for expansions.

2.1.3. W25Q64 SPI Daughter card

W25Q64 is a breakout board which has 64Mbits flash with Standard SPI interface. It supports up to 104MHz. It works on 2.7V to 3.5V range. Based on this we need to allocate the IO bank from the FPGA.

2.1.4. USB to UART FTDI adaptor



FTDI USB port is used for COM port connectivity via which debug prints in the code will be sent out. This FTDI will work in 3.3V Range and it supports up to 230Kbps UART baud rate.

2.2. Software Requirements

2.2.1. Xilinx Vivado 2023.1

Vivado tool is used to generate the RTL from the Block design of the SoC and handle all the SoC integration part. This tool will also handle the flow of synthesis, place and route, and bit file generation which is need for the target KR26 Family FPGA. Block design for this dissertation is shown in Hardware Requirements section of the document. This Block design is generated with the help of Xilinx IP integrator flow. This tool has Hardware manager tool which will get the debug probe data via Integrated Logic Analyzer from the real hardware.

2.2.2. Xilinx Vitis 2023.1

Vitis tool is used to generate the Embedded Application for the SoC design implemented on FPGA and handles the Application compilation and debug part. This tool will also handle the flow of compilation, debug and binary file generation which is needed for the target KR26 Family FPGA. The linker script required to handle the memory map of the SoC will be generated automatically from hardware specification file (.xsa file). This tool will also provide serial terminal to monitor the debug prints in the application.

3. Design Considerations

- UART COM port is working in 115.2KBps.
- Target DUT is Operating at 5MHz.
- Timer is configured to generate interrupt on every 30 microseconds.
- System BUS clock frequency is 40MHz
- BRAM size is 256 kB.
- Non-volatile memory controller (flash controller) is not part of platform design.
- Entire Application code is stored and executed from BRAM.
- FPGA is having a separate config flash to store the BIT file.

4. Hardware Setup

The numbered parts from the picture are explained as sub sections below,

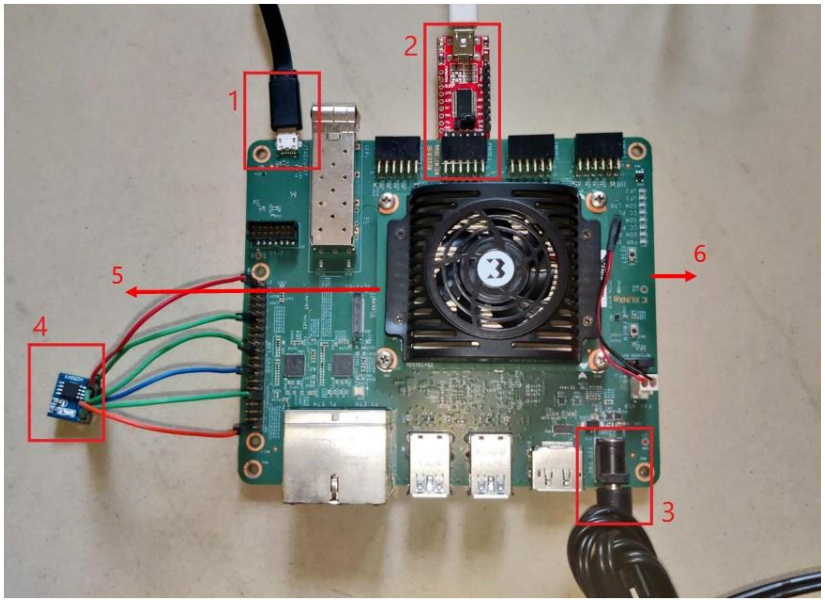


Figure 4 Hardware setup used for this project

4.1. JTAG Connector

This is a USB micro-B type connector to load bit files into the FPGA. This provides a bridge between FPGA and the Software running at the PC like Vivado and Vitis. Using this, we can dump the live debug signals via hardware integrated logic analyzer. Since this ILA hardware is inbuilt with the FPGA we do have some resource limitations. That's the reason for additional Protocol analyzer connected into this system.

4.2. Debug UART

This is a user defined normal UART channel via which we'll dump the software print message. To access this hardware, we need to enable and configure UART IP in the software also. This procedure is explained in the below section of the report. This just a normal FTDI level translator available in the market to communicate with the PC via COM port. This converts the CMOS level that is 0-3.3V output from the UART IP IOs USB serial IO level.

4.3. Power Supply Jack

This hardware setup requires a 12V 3A barrel role power supply to power the KR26 SOM, carrier card and other peripherals connected to the board. This carrier board can provide power supply to additional daughter card like SPI Flash, FTDI adapter (UART).

4.4. SPI Serial Flash

This is the actual slave device which is going to communicate with the DUT from External world. It requires a 3.3V supply and SPI lines to operate properly.

4.5. KR26 SOM

This is the KR26 FPGA System On Module where our design is loaded and executed. This consists of all modules like Oscillator, DDR3 memory and QSPI flash which are the facilitators for FPGA to operate properly. This will have 2 board to board IO connectors which is used to connect the carrier card to interface with the connected peripherals.

4.6. Carrier Card

This card is kind of a base board which holds the IO peripherals like camera interface, 10G Ethernet SFP+ cage, RJ45 Connector for 2.5G EMAC etc. These peripherals were connected to respective IO lines coming out from SOM B2B connecter.

5. Technologies used

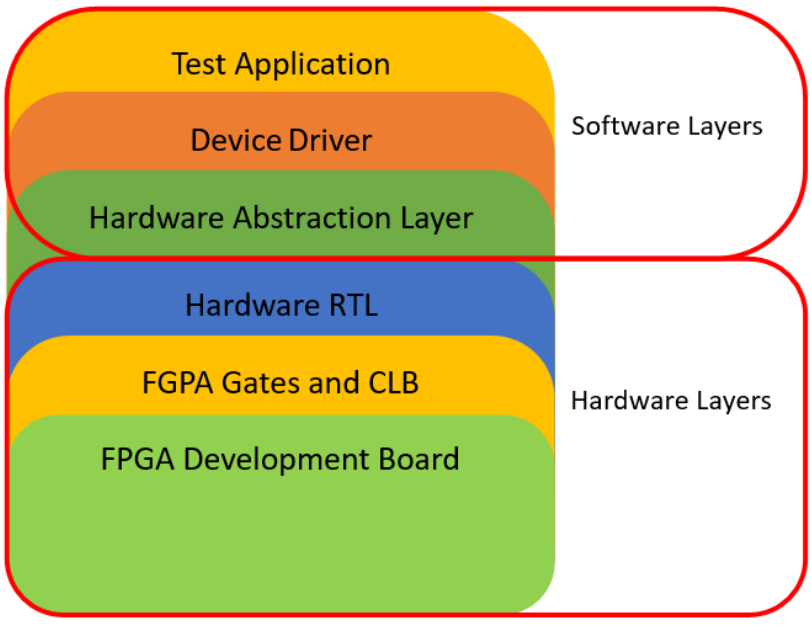


Figure 5 Layers of the implemented Design

In this project, a complete Embedded system is built over an FPGA to verify the DUT. The entire system is separated into different layers and mentioned in the Figure 5. And each layer is explained as follows,

5.1. Test Application

This layer is the topmost software layer from where all the other layers were accessed based on the code flow. In our case all our test cases are non-OS standalone application, from here we'll call all the API's sequentially to enable as per the functionality. Here the APIs are defined based on the feature so our test API calls will happen here.

5.2. Device Driver

This layer is kind of a middleware, where all hardware specific configuration APIs like baud rate setting, FIFO writes, Interrupt callbacks etc., will be present. From Application layer device drivers are invoked to perform the functions like initializing SPI, transmitting data from SPI, Read the received data from SPI slave. In this project, device driver for DUT and SPI Flash were written. Additionally, the device driver for platform IPs like UART and GPIO were written.

5.3. Hardware Abstraction layer

This layer consists of all register level read/write APIs which is specific to each hardware IP. This layer is very close to hardware, and this will have all the physical address, offsets, and bit field details of individual IP. This layer is mandatory for all embedded software to interact with the hardware. In this project this layer of software will be provided by the IP vendor that is Xilinx which is invoked in the Device driver layer.

5.4. Hardware RTL Logic

This layer consists of synthesized behavioral model of the IP which will have the information of how the Logic gates must be connected to achieve the required functionality. This will be generated basically using Xilinx IP Integrator and synthesized with FPGA tools like Xilinx Vivado and flashed into FPGAs to emulate the DUT. In this project, entire block design (DUT + Platform) is generated and synthesized using a Xilinx tool and flashed in the FPGA hardware.

5.5. FPGA Gates and CLB

FGPAs are the physical Hardware IC which have millions of Logic gates which can be configured in a different way to achieve any type of Digital logic design. In this project, KR26 or XC7A100T both are general purpose FPGAs used for digital logic implementation. This can be programmed via JTAG interface from Vivado Lab Design tool.

5.6. FGPA Development Board

In this project, the complete setup is built over a KR26 based System on Module (SOM) which have so many peripherals like DDR, QSPI, CSI, DP, HDMI, SFP+, USB, Ethernet, IOs, PMOD connectors. Among this, we utilized IOs and PMOD connectors to build our test environment. Here, we connected the FTDI converter in PMOD connectors and SPI Flash is connected to Raspberry PI connector via which DUT IO lines are taken out. Additionally, we have one user LED to see the application Status. Signal details are listed below,

Sl No	Signals from IP	Board Connector
1	spi_rtl_0_ss_io	RHEADER_GPIO_2
2	spi_rtl_0_io0_io	RHEADER_GPIO_5
3	spi_rtl_0_io1_io	RHEADER_GPIO_4
4	spi_rtl_0_sck_io	RHEADER_GPIO_3
5	rtl_0_txd	PMOD3_GPIO_1
6	rtl_0_rxd	PMOD3_GPIO_0
7	gpio_rtl_0_tri_io	USER_LED

Table 3 Signal to Board IO mapping

6. Testcase Planning and Use case mapping

The target use case of the DUT is to communicate with any high speed PHY chips that is controlled via SPI interface. It can be any interface like EtherCAT where Ethernet Master will be connected to slave in daisy chain fashion and all slave controllers will have SPI slave interface to control the EtherCAT packet transfer. In this project, we are not bothered about the EtherCAT IP we are concentrating only on SPI master which is going to control the EtherCAT in the end application. This SPI should follow the open standard spec which mean it has to be compatible with any SPI slave in the market. Thats why we have chosen the Winbond SPI flash. So, now our target is DUT must communicate with a SPI serial Flash. This will indirectly prove that the DUT is compatible with any SPI Slave.

To test this, we are planning to develop two types of testcase which is listed below,

1. Internal Loopback test
2. SPI Flash test

This splitting of test scenarios helps to make the debug and bring up easier. In case of any failure, finding the root cause will be easier. If the Loopback (LB) is enabled, the TX data in the TX FIFO is routed to RX FIFO with this we can check the data after the CDC logic. To bring up the IP initially this loopback is needed. Once the IP is stable then the Transmitter and Receiver IO logic can be validated. This is done via interfacing the SPI Flash to the DUT via the IP IOs.

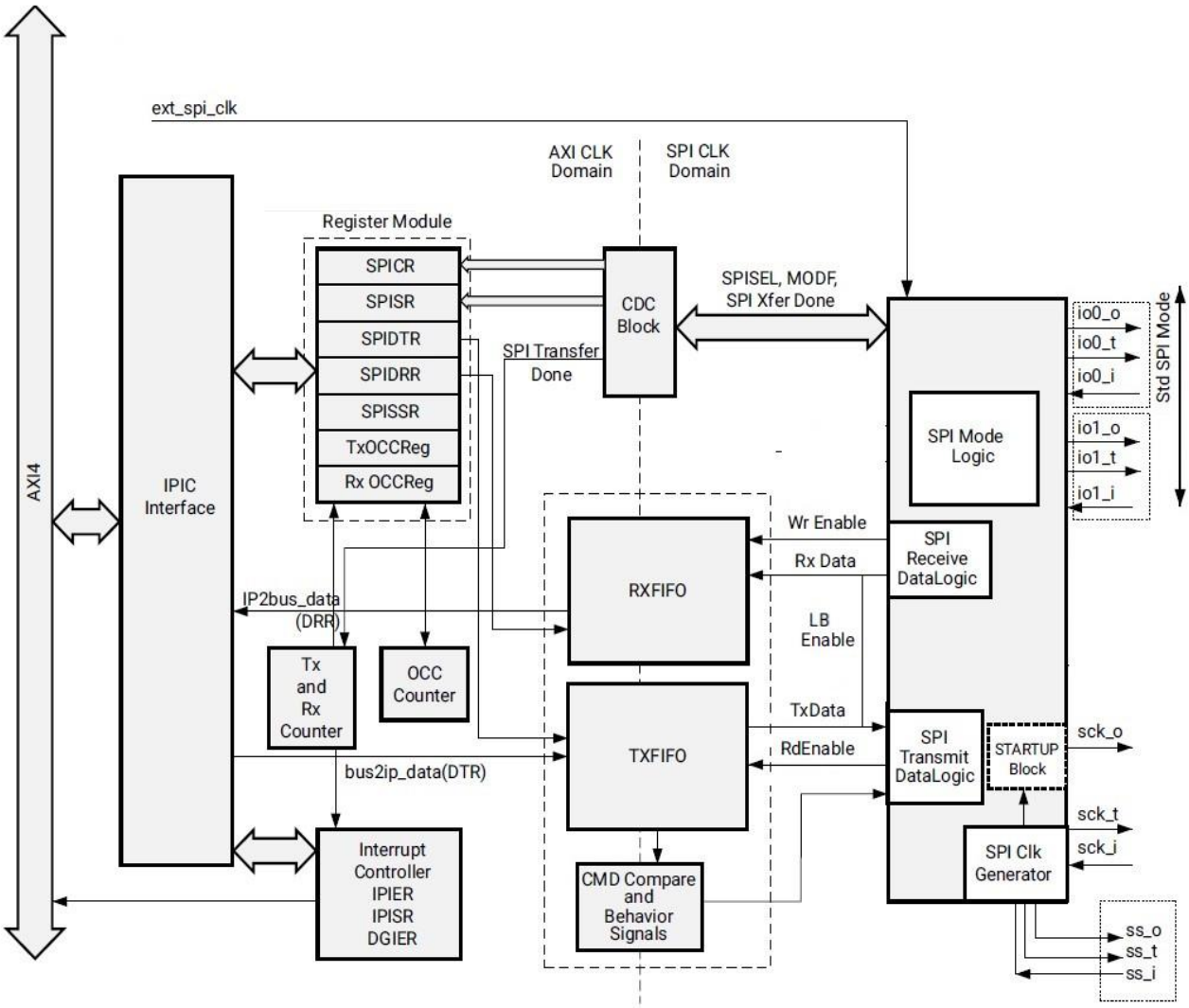


Figure 6 Block Diagram of DUT (SPI IP)

7. Testcase Design and Implementation

7.1. Testcase files and API

The device driver of Platform IPs like GPIO, Interrupt Controller and UART will be taken from Xilinx repository. The Application layer is written in the platform initialization files to configure the software debug interfaces. The list of files included in the testcases is given in Figure 7. The nomenclature followed in the project is given bellow,

1. <IP>_g.c, <IP>_i.h, <IP>_l.h are HAL files
2. <IP>.c, <IP>.h are Device driver or Middleware.
3. <IP>_intr.c are the ISR related to the IP

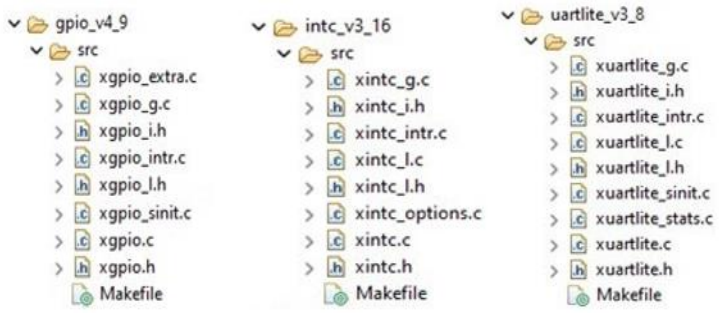


Figure 7 HAL and Driver for Platform IPs

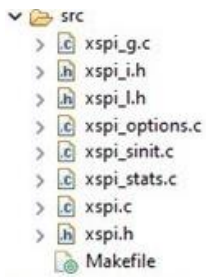


Figure 9 HAL and Driver for DUT

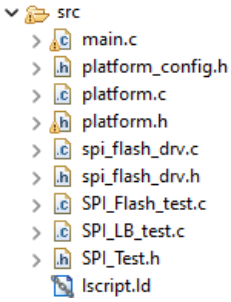


Figure 8 Testcases for DUT SPI Flash driver

Same Nomenclature is followed for DUT IP also. In testcase directory the file description is given below

1. Platform.c, platform.h, platform_config.h are platform IP's application layer files
2. Spi_flash_drv.c, Spi_flash_drv.h are the device driver for SPI slave device.
3. Spi_LB_test.c and spi_Flash_test.c are the respective test applications called from main.c
4. Lscript.ld is the custom linker script for this software project.

7.2. Testcase Implementation Flow Chart

The flow diagram of two testcases planned in the chapter 6 is given below. This will give a clear idea about the functionalities done in each layer of the software. It also shows debug prints in each stage of the code

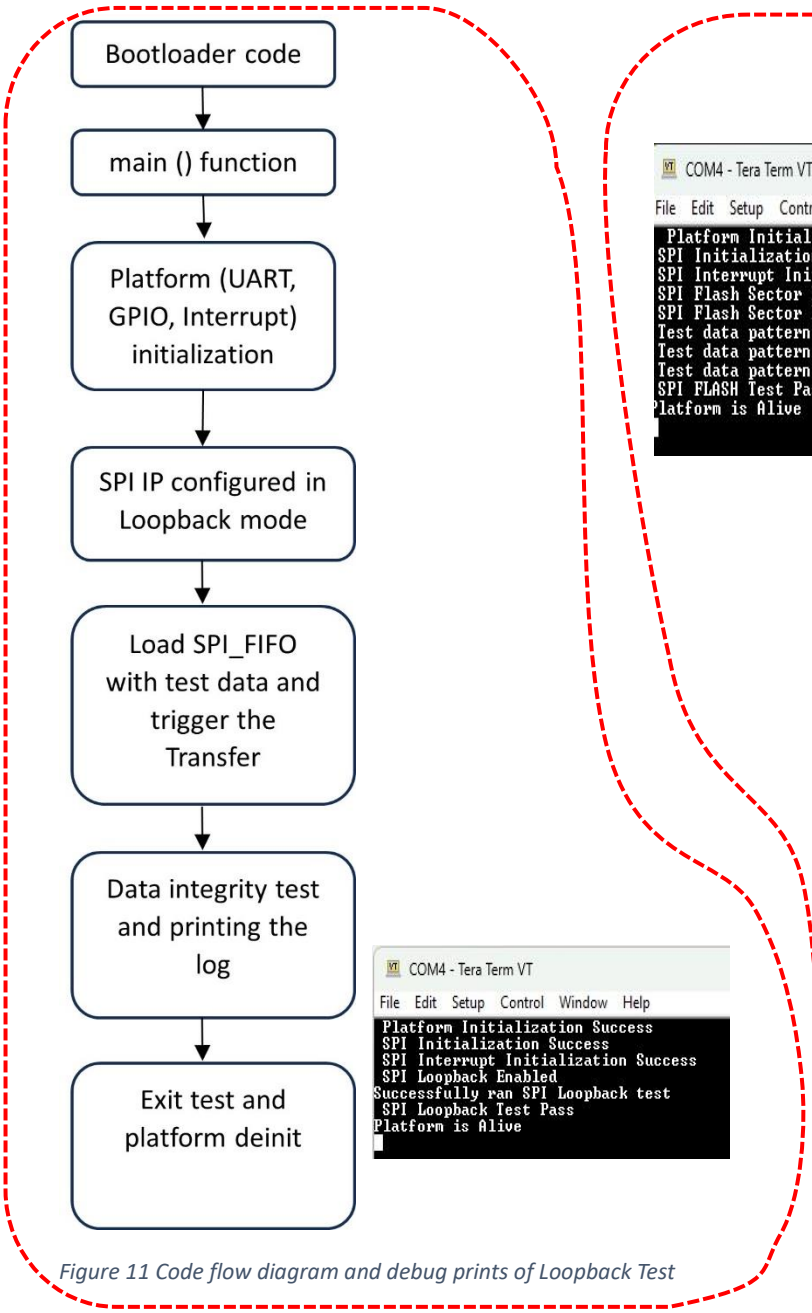


Figure 11 Code flow diagram and debug prints of Loopback Test

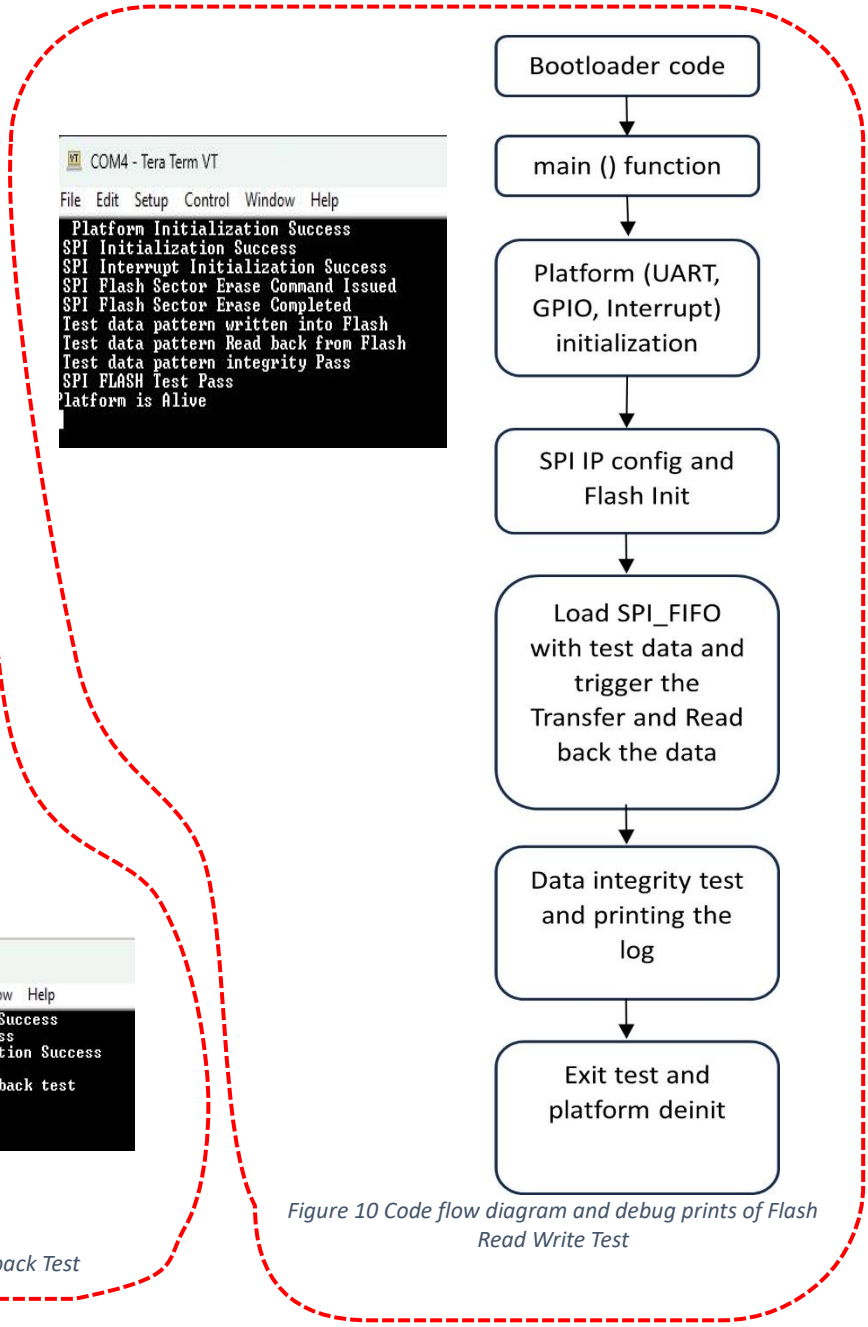
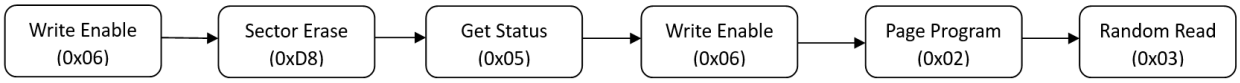


Figure 10 Code flow diagram and debug prints of Flash Read Write Test

Command sequence for a successful write and read back to the SPI serial flash as per the datasheet is given as follow, this sequence is followed in flash test code and SPI Flash device driver.



8. Execution Results

8.1. Write Enable command Waveform

The Xilinx ILA snapshot of write enable(0x06) command given out from the SPI IP is shown below,

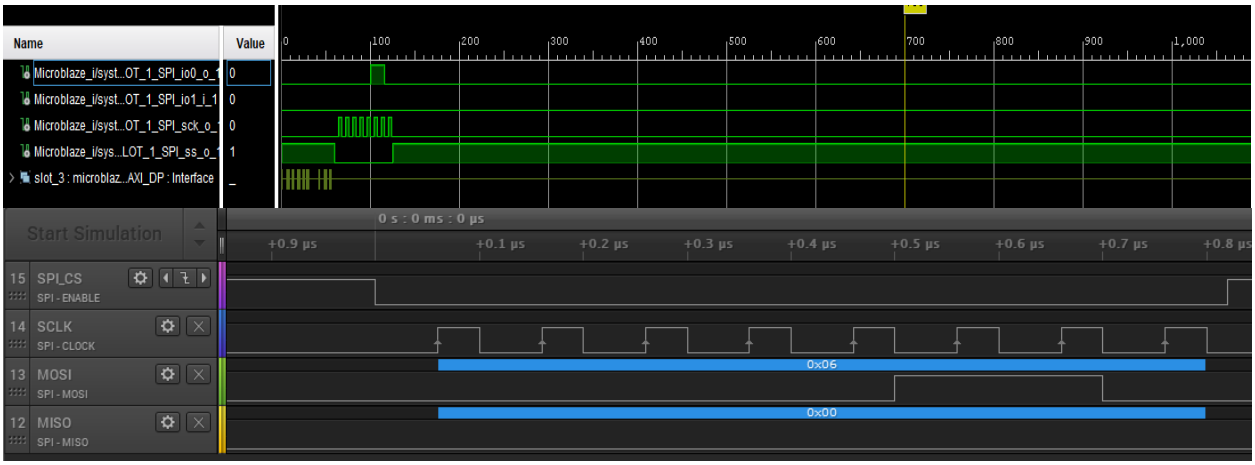


Figure 12 Write Enable command to Slave

8.2. Sector Erase Command Waveform

Due to hardware resource constraints, unable to capture this command in the Xilinx ILA instead, captured the live bus with logic analyzer and added the snapshot of it below,

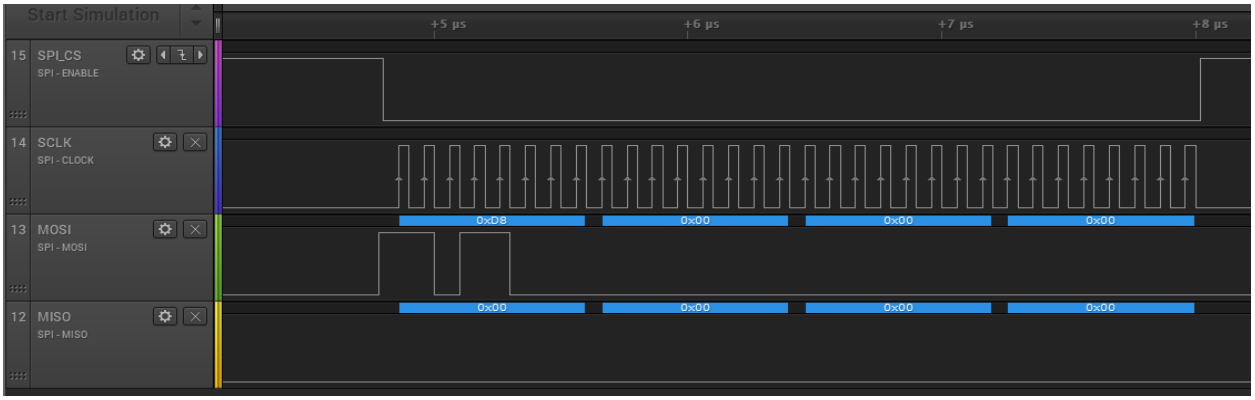


Figure 13 Sector Erase Command to the slave

8.3. Get slave status Command Waveform

In this waveform master is poling for slave status register and it is responding with BUSY bit set to 1 since the sector erase command is in progress.

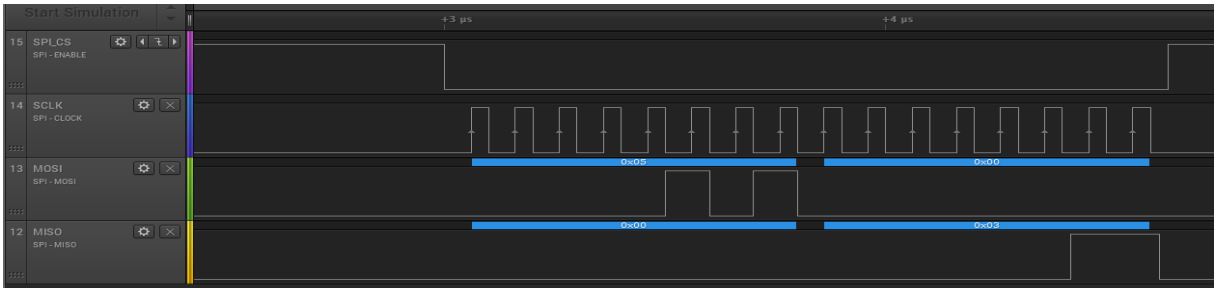


Figure 14 Get Status Command to the slave

8.4. Get slave status Command Waveform

In this waveform master is poling for slave status register and it is responding with BUSY bit set to 0 since the sector erase command is completed.

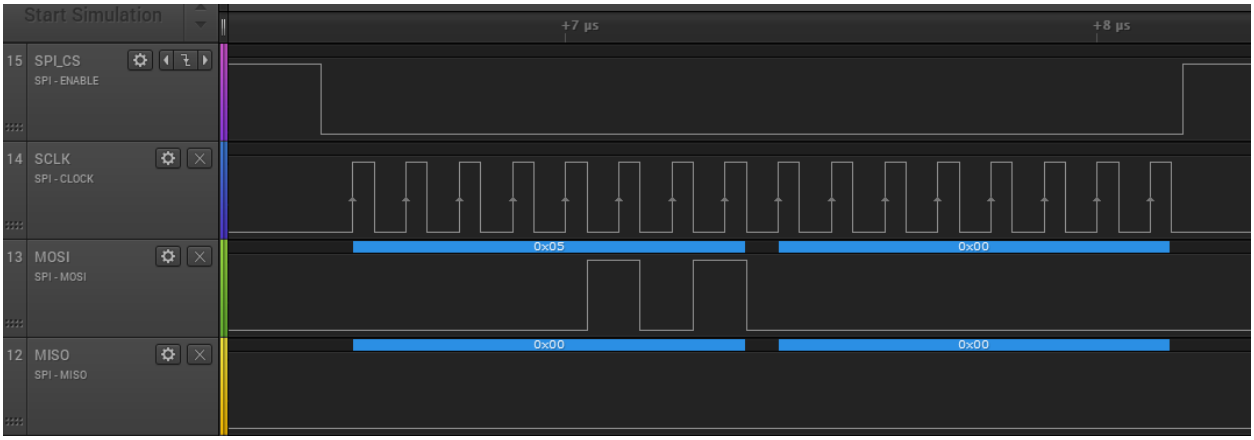


Figure 15 Get status command with ready slave response

8.5. Page Program Command Waveform

In this waveform we are writing 10 bytes of data to the slave from master at address of 0x00. The data is 0x20-0x29. This is the test pattern we are writing to the slave device.

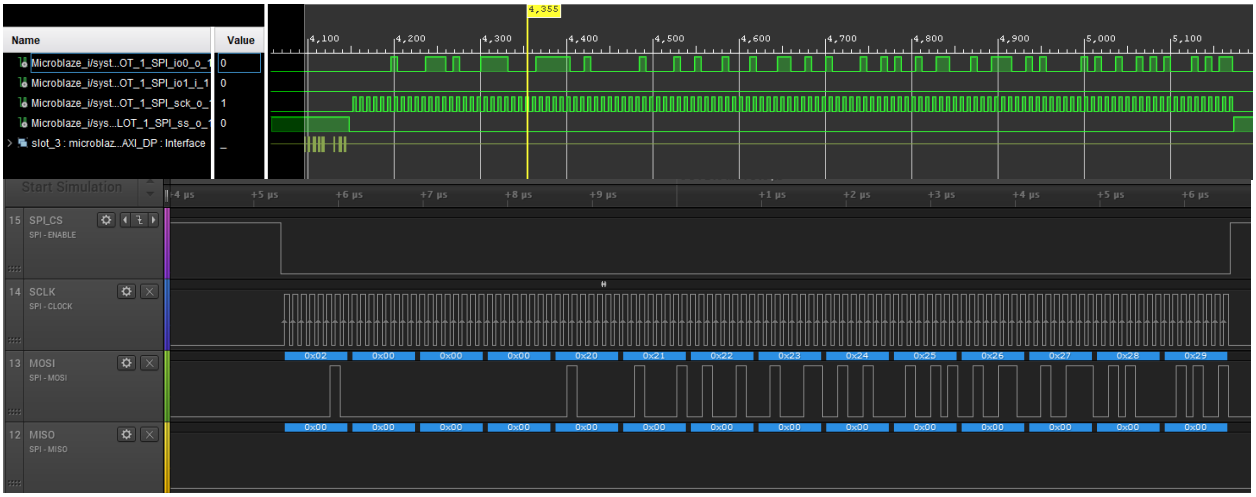


Figure 16 Page Program to the slave with data pattern

8.6. Random Read Command Waveform

The written data is read back from the flash via MISO line by issuing the command of 0x03.

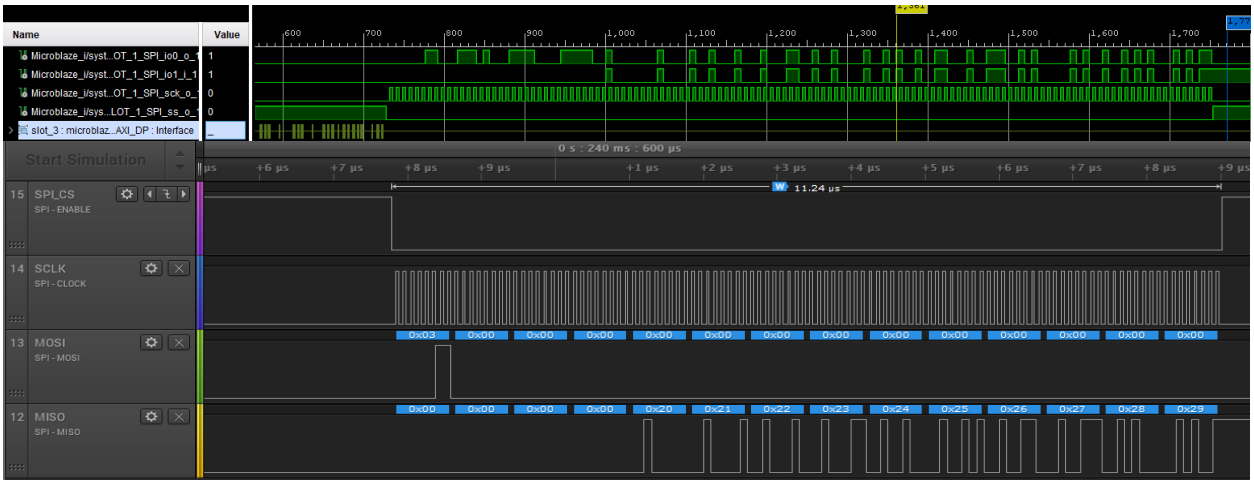


Figure 17 Read back of the pattern from the slave

Time [s]	Protocol	Decoded Protocol Result
6E-08	SPI	MOSI 0x06; MISO 0x00
2.49E-05	SPI	MOSI 0xD8; MISO: 0x00
2.57E-05	SPI	MOSI: 0x00; MISO: 0x00
2.65E-05	SPI	MOSI: 0x00; MISO: 0x00
2.73E-05	SPI	MOSI: 0x00; MISO: 0x00
5.31E-05	SPI	MOSI: 0x05; MISO: 0x00
5.39E-05	SPI	MOSI: 0x00; MISO: 0x03
0.240207	SPI	MOSI: 0x05; MISO: 0x00
0.240207	SPI	MOSI: 0x00; MISO: 0x00
0.240222	SPI	MOSI: 0x06; MISO: 0x00
0.240535	SPI	MOSI: 0x02; MISO: 0x00
0.240536	SPI	MOSI: 0x00; MISO: 0x00
0.240537	SPI	MOSI: 0x00; MISO: 0x00
0.240538	SPI	MOSI: 0x00; MISO: 0x00
0.240539	SPI	MOSI: 0x20; MISO: 0x00
0.240539	SPI	MOSI: 0x21; MISO: 0x00
0.24054	SPI	MOSI: 0x22; MISO: 0x00
0.240541	SPI	MOSI: 0x23; MISO: 0x00
0.240542	SPI	MOSI: 0x24; MISO: 0x00
0.240543	SPI	MOSI: 0x25; MISO: 0x00
0.240543	SPI	MOSI: 0x26; MISO: 0x00
0.240544	SPI	MOSI: 0x27; MISO: 0x00
0.240545	SPI	MOSI: 0x28; MISO: 0x00
0.240546	SPI	MOSI: 0x29; MISO: 0x00
0.240598	SPI	MOSI: 0x03; MISO: 0x00
0.240599	SPI	MOSI: 0x00; MISO: 0x00
0.240599	SPI	MOSI: 0x00; MISO: 0x00
0.2406	SPI	MOSI: 0x00; MISO: 0x00
0.240601	SPI	MOSI: 0x00; MISO: 0x20
0.240602	SPI	MOSI: 0x00; MISO: 0x21
0.240603	SPI	MOSI: 0x00; MISO: 0x22
0.240603	SPI	MOSI: 0x00; MISO: 0x23
0.240604	SPI	MOSI: 0x00; MISO: 0x24
0.240605	SPI	MOSI: 0x00; MISO: 0x25
0.240606	SPI	MOSI: 0x00; MISO: 0x26
0.240607	SPI	MOSI: 0x00; MISO: 0x27
0.240607	SPI	MOSI: 0x00; MISO: 0x28
0.240608	SPI	MOSI: 0x00; MISO: 0x29

Table 4 Sequence of commands sent and received in bus probed via protocol analyzer

Table 4 shows the actual commands and data transmitted and received between master and slave along with the timestamps which helps to sequence the command and data based on the recommendations from the flash datasheet.

9. Conclusion

This project successfully implemented a SPI IP in FPGA platform and developed standalone non-OS C based testcase to test the interface with the SPI Flash. This Qualifies the IP is suitable for the targeted application of controlling an Industrial grade EtherCAT PHY chip via this SPI master IP. In Real time application this SPI master is going to control a High speed EtherCAT bus via SPI slave interface provided to it. As part of this project, we tested the code developed and presented the result in different sections. For demo and results presentation, a different development boards (based on XC7A100T) are used due to resource constraints. The small-scale demo setup for the same design is attached below.

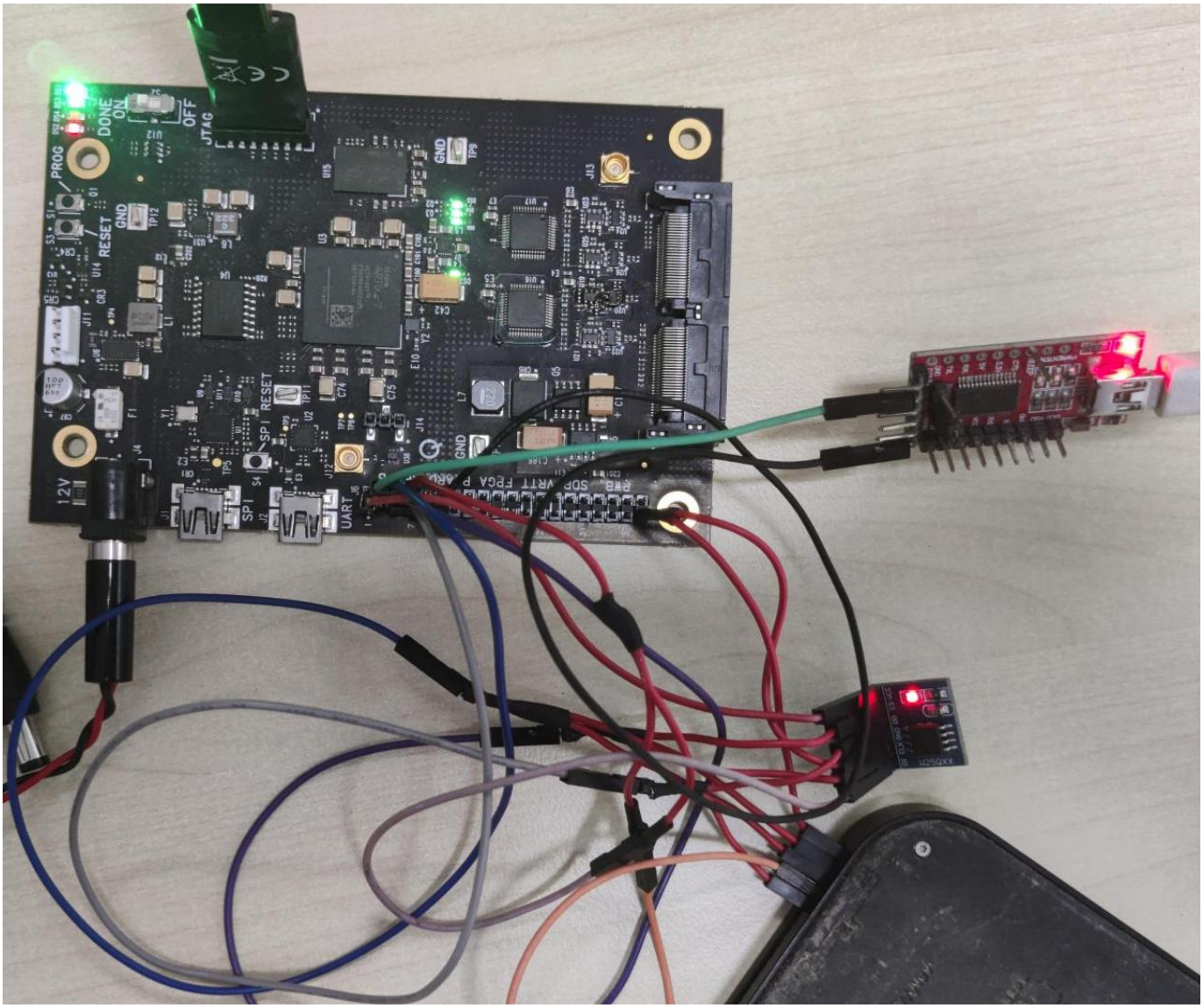


Figure 18 Demo setup used for personal studies and Experiments

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	Block RAM Tile (135)	Bonded IOB (170)	ILOGIC (170)	BUFGCTRL (32)	PLLE2_ADV (6)	BSCANE2 (4)
Microblaze_wrapper	4220	4995	115	1903	3919	301	78	11	2	6	1	2
dbg_hub (dbg_hub)	450	741	0	241	426	24	0	0	0	1	0	1
inst (xsdbm_v3_0_0_xs	450	741	0	241	426	24	0	0	0	1	0	1
Microblaze_i (Microblaze)	3770	4254	115	1668	3493	277	78	0	2	5	1	1
axi_bram_ctrl_0 (Microbl	220	238	2	116	220	0	0	0	0	0	0	0
axi_bram_ctrl_0_bram (	54	1	0	42	54	0	64	0	0	0	0	0
axi_gpio_0 (Microblaze_	67	93	0	31	67	0	0	0	0	0	0	0
axi_intc_0 (Microblaze_a	82	90	0	29	82	0	0	0	0	0	0	0
axi_interconnect_0 (Micr	242	132	0	111	242	0	0	0	0	0	0	0
axi_quad_spi_0 (Microbl	359	543	0	166	347	12	0	0	2	0	0	0
axi_timer_0 (Microblaze_	293	244	0	114	293	0	0	0	0	0	0	0
axi_uartlite_0 (Microblaz	88	107	1	40	78	10	0	0	0	0	0	0
clk_wiz (Microblaze_clk_	0	0	0	0	0	0	0	0	0	3	1	0
mdm_1 (Microblaze_md	93	110	0	44	86	7	0	0	0	2	0	1
microblaze_0 (Microblaz	1160	1021	108	472	1042	118	0	0	0	0	0	0
microblaze_0_local_me	16	14	0	12	14	2	8	0	0	0	0	0
rst_clk_wiz_100M (Micro	18	38	0	14	17	1	0	0	0	0	0	0
system_ila_0 (Microblaz	1081	1623	4	555	954	127	6	0	0	0	0	0
xlconcat_0 (Microblaze_)	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19 Utilization Report of Design Implemented in XC7A100T based demo board

10. Abbreviations:

FPGA	Field Programable Gate Array
IP	Intellectual Property
GPIO	General Purpose Input and Output Pin
SoC	System On Chip
DUT	Design Under test
BRAM	Block RAM (Random Access Memory)
DP	Display port
ILA	Integrated Logic Analyzer
ISR	Interrupt Service Routine
CDC	Clock Domain Crossing
FIFO	First In First Out
CSI	Camera Serial Interface
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
SFP+	Small Form Factor+
MISO	Master In Slave Out
MOSI	Master Out Slave In
OS	Operating System

11. Literature Survey

For a successful research, literature survey is the starting and a base point. The foundation for technical aspects of this project is came from the following journals and reference materials,

[1] Xilinx, AXI to Quad SPI v3.2 LogiCORE IP Guide PG153 April 26, 2022.

[2] Xilinx.com, MicroBlaze Processor Reference Guide. UG984 (v2021.2) October 27, 2021

[3] Quick Start Guide: MicroBlaze Soft Processor Presets.

[4] Xilinx 7 Series FPGAs Data Sheet: DS180 (v2.6.1) September 8, 2020

[5] Volnei A. Pedroni, Programming FPGAs: Getting Started with Verilog by Simon Monk, 2020.

[6] Xilinx.com, AXI GPIO_2.0 LogiCORE IP Product Guide PG144 April 26, 2022

[7] Xilinx.com, AXI UARTlite_2.0 LogiCORE IP Product Guide PG142 April 5, 2017

[8] Xilinx, AXI Interrupt Controller_4.1 LogiCORE IP Product Guide PG99 July 15, 2021

[9] Orhan Gazi, A.Çağrı Arlı ,State Machines using VHDL: FPGA Implementation of Serial Communication and Display Protocols, Springer 2021.

[10] Embedded System Design, Peter Marwedel, Springer 2003.

[11] Kamal, Raj, Embedded Systems: Architecture, Programming & Design, Tata McGraw Hill, 2nd Ed., 2008

[12] Wolf, Wayne, Computers as Components – Principles of Embedded Computing System Design, Second Edition, Morgan-Kaufmann, 2010.

Checklist of items for the Final Dissertation Report

This checklist is to be attached as the last page of the report.

This checklist is to be duly completed, verified, and signed by the student.

1.	**Is the final report neatly formatted with all the elements required for a technical Report?**	Yes
2.	Is the Cover page in proper format as given in Annexure A?	Yes
3.	Is the Title page (Inner cover page) in proper format?	Yes
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	Yes Yes
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	Yes Yes
6.	Is the title of your report appropriate? **The title should be adequately descriptive, precise and must reflect scope of the actual work done.** Uncommon abbreviations / Acronyms should not be used in the title	Yes
7.	Have you included the List of abbreviations / Acronyms?	Yes
8.	Does the Report contain a summary of the literature survey?	Yes
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	Yes Yes Yes Yes Yes Yes
10.	Is the conclusion of the Report based on discussion of the work?	Yes
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	Yes Yes Yes
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a Power Point Presentation, or a user manual. Source code of software need not be included in the report.	Yes

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Signature of the Student

Place: Bengaluru **Name : SELVA KUMAR S**

Date: 30.04.2024 **ID No. : 2022HT80170**