Dissertation Title:

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

**Course No.: MELZG628T**

**Course Title: Dissertation**

**Dissertation Done by: SELVA KUMAR S**

**Student Name: SELVA KUMAR S**

**BITS ID: 2022HT80170**

**Degree Program: M.Tech. Microelectronics**

**Research Area: IP Emulation and Validation**

**Dissertation carried out at: Qualcomm Technologies Pvt. Ltd, Bengaluru**

A logo of a science and technology company

Description automatically generated

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**VIDYA VIHAR, PILANI, RAJASTHAN - 333031.**

**(Jan 2024)**

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 the resource planning is made in this dissertation. Further technical details about the SOM are discussed in the Hardware Requirement 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 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 this understanding is done, we’ll start thinking of 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.

A close-up of a signature

Description automatically generated

A signature on a white background

Description automatically generated

Signature of the Student Signature of the Supervisor

Name: Selva Kumar S Name: Siva Selvamani

Date: 22/3/2024 Date:22/3/2024

Place: Bengaluru Place: Bengaluru

Contents

[1. Functional Block Diagram 5](#_Toc164860220)

[Xilinx IP used in Platform and DUT Subsystem: 5](#_Toc164860221)

[2. Hardware and Software Requirements: 7](#_Toc164860222)

[Hardware Requirements: 7](#_Toc164860223)

[KR26 FPGA SOM: 7](#_Toc164860224)

[FPGA Carrier Card 7](#_Toc164860225)

[W25Q64 SPI Daughter card: 8](#_Toc164860226)

[USB to UART FTDI adaptor: 8](#_Toc164860227)

[Software Requirements: 8](#_Toc164860228)

[Xilinx Vivado 2023.1: 8](#_Toc164860229)

[Xilinx Vitis 2023.1: 8](#_Toc164860230)

[3. Design Considerations: 9](#_Toc164860231)

[4. Hardware Setup: 9](#_Toc164860232)

[4.1. JTAG Connector: 9](#_Toc164860241)

[4.3. Power Supply Jack: 10](#_Toc164860242)

[4.4. SPI Serial Flash: 10](#_Toc164860243)

[4.5. KR26 SOM: 10](#_Toc164860244)

[4.6. Carrier Card: 10](#_Toc164860245)

[5. Technologies used: 10](#_Toc164860246)

[5.1. Test Application: 10](#_Toc164860248)

[5.2. Device Driver: 11](#_Toc164860249)

[5.3. Hardware Abstraction layer: 11](#_Toc164860250)

[5.4. Hardware RTL Logic: 11](#_Toc164860251)

[5.5. FPGA Gates and CLB: 11](#_Toc164860252)

[5.6. FGPA Development Board: 11](#_Toc164860253)

[6. Testcase Application Design: 12](#_Toc164860254)

[7. Testcase Implementation and Use case mapping: 12](#_Toc164860255)

[8. Execution Results: 13](#_Toc164860256)

[9. Conclusion: 15](#_Toc164860257)

# Functional Block Diagram

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

A diagram of a system

Description automatically generated

Figure 1‑1Block 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.

1. 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 |

A diagram of a computer

Description automatically generated

System ILA

DUT SS

PLATFORM SS

Figure 1‑2Xilinx IP Integrator Block Design

# 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 1Hardware and Software Requirements

## Hardware Requirements:

### KR26 FPGA SOM:

A close-up of a computer chip

Description automatically generated

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.

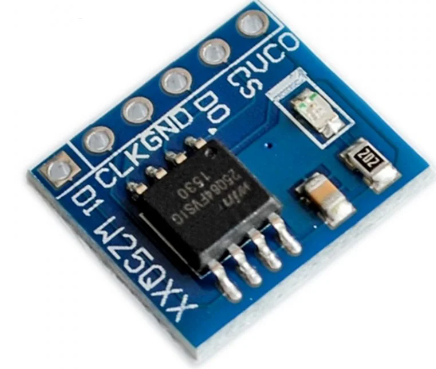
### FPGA Carrier Card

A green electronic board with a fan

Description automatically generated

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 FDTI adaptor and SPI Flash daughter card. This also have two user LEDs which we used as Notification LED. Reset pin is default tied to high. This also have 1 SFP cage, 4 USB connectors, 4 RJ45 connectors, 1 Display port. This we can use it for expansions

### 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.

### 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.

## Software Requirements:

### 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 the section 2 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 date via Integrated Logic Analyzer from the real hardware.

### Xilinx Vitis 2023.1:



Vitis tool is used to generate the Embedded Application for the SoC design implemented on FPGA and handle the Application compilation and debug part. This tool will also handle the flow of compilation, debug and binary file generation which is need 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.

# 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.
* Nonvolatile 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.

# Hardware Setup:

A circuit board with wires and a fan

Description automatically generated

Figure 4‑1 Hardware setup used for this project



## JTAG Connector:

This is a USB micro-B type connector to load bit files into the FPGA. This provides a bridge between our FPGA and the Software running at the PC like Vivado and Vitis. Using this we can dumb 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.

* 1. 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.

## 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 additional daughter card like SPI Flash, FTDI adapter (UART).

## 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.

## KR26 SOM:

This the KR26 FPGA System On Module where our design is loaded and executed. This consist 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.

## Carrier Card:

This card is kind of a base board which holds the IO peripheral 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.

# Technologies used:

A diagram of software components

Description automatically generated

Figure 5‑1Layers of the implemented Design

In this project a complete Embedded system is build over an FPGA to verify the DUT. The entire system is separated into different layers and mentioned in the Figure 4. And each layer is explained as follow,



## 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.

## Device Driver:

This layer is kind of a middleware, where all hardware specific configuration APIs will be there like baud rate setting, FIFO writes, Interrupt callbacks etc. Usually 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 also device driver for DUT and SPI Flash were written. Additionally, the device driver for platform IPs like UART and GPIO were written.

## 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

## 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 a HDL 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.

## FPGA Gates and CLB:

FGPAs are the physical Hardware IC which have millions of Logic gates which can be configured in different way to achieve any type of Digital logic design. In our 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.

## FGPA Development Board:

In this project the complete setup is build 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 GPIO connector via which out 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 2 Signal to Board IO mapping

# 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. That 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.

# Testcase Design and Implementation:

A screenshot of a computer

Description automatically generated

Figure 7‑1 HAL and Driver for Platform IPs

A screenshot of a computer program

Description automatically generated A screenshot of a computer

Description automatically generated

Figure 7‑3 Testcases for DUT SPI Flash driver

Figure 7‑2 HAL and Driver for DUT

The st

A diagram of a software system

Description automatically generated

Figure 6‑1 Code flow diagram

# Execution Results:

A green and black background

Description automatically generatedA black and blue line

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

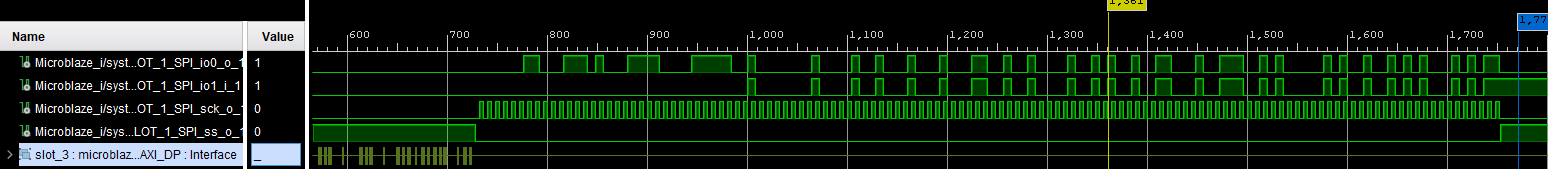
A screenshot of a computer

Description automatically generated

A green and white lines on a black background

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
| 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 |

# Conclusion: