SEPTEMBER 13, 2023

Processing System & Programmable Logic Communication

AREEBA MEHBOOB
BCUBE PVT LIMITED

Table of Contents

List of Figures

Chapter 1

Introduction

1.0 Processing system & Programmable logic

The processing system is capable of dynamic tasks and complicated logic controls. It is used for general purpose sequential tasks, operating system, GUIs, and user applications.

Each processor is constituted of one or more individual processing units called “cores”. Each core processes instructions from a single computing task at a certain speed, defined as “clock speed” and measured in gigahertz (GHz). Since increasing clock speed beyond a certain point became technically too difficult, modern processors now have several cores (dual-core, quad-core, etc.). They work together to process instructions and complete multiple tasks at the same time.

A processor is made of four basic elements: the arithmetic logic unit (ALU), the floating-point unit (FPU), registers, and the cache memories. The ALU and FPU carry basic and advanced arithmetic and logic operations on numbers and then results are sent to the registers, which also store instructions. Caches are small and fast memory that store copies of data for frequent use, and act similarly to a random-access memory (RAM).

Programmable Logic is for static parallel tasks and peripheral controls. The main difference between a standard processor and a programmable logic device is that their internal structure is not fixed. Taking standard CPUs as an example, one can run a program which will be executed on the internal infrastructure of logic components. Programming logic devices means describing and defining this infrastructure. Instead of fixed structure, PLDs consist of arrays of general logic blocks, which can be configured individually. Using a Hardware Description Language and a dedicated compiler, the abstract logic functions described by the developer are translated into series of logic blocks configured and interconnected accordingly.

1.1 ZYNQ SOC

Figure 1.1 ZYNQ SOC

The Zynq SoC family integrates the software programmability of an ARM-based processor with the hardware programmability of an FPGA, enabling key analytics and hardware acceleration while integrating CPU, DSP, ASSP, and mixed signal functionality on a single device. It combines programmable logic with dual, single, or quad-core Arm Cortex-A series processors. It has **AXI BUS for PS and PL communication.**

PS-PL Interface

The PS-PL interface includes:

• AMBA AXI interfaces for primary data communication

• Two 32-bit AXI master interfaces

• Two 32-bit AXI slave interfaces

• Four 64-bit/32-bit configurable, buffered AXI slave interfaces with direct access to DDR memory and OCM, referred to as high-performance AXI ports

• One 64-bit AXI slave interface (ACP port) for coherent access to CPU memory

Processing System features:

ARM Cortex-A9 Based Application Processor Unit (APU)

• 2.5 DMIPS/MHz per CPU

• CPU frequency: Up to 1 GHz

• Coherent multiprocessor support

• ARMv7-A architecture

• TrustZone security

• Thumb-2 instruction set

• Jazelle RCT execution Environment Architecture

• NEON media-processing engine

• Single and double precision Vector Floating Point Unit (VFPU)

• CoreSight and Program Trace Macrocell (PTM)

• Timer and Interrupts

• Three watchdog timers

• One global timer

• Two triple-timer counters

Caches

• 32 KB Level 1 4-way set-associative instruction and data caches (independent for each CPU)

• 512 KB 8-way set-associative Level 2 cache (shared between the CPUs)

• Byte-parity support

On-Chip Memory

• On-chip boot ROM

• 256 KB on-chip RAM (OCM)

• Byte-parity support

External Memory Interfaces

• Multiprotocol dynamic memory controller

• 16-bit or 32-bit interfaces to DDR3, DDR3L, DDR2, or LPDDR2 memories

• ECC support in 16-bit mode

• 1GB of address space using single rank of 8-, 16-, or 32-bit-wide memories

• Static memory interfaces

• 8-bit SRAM data bus with up to 64 MB support

• Parallel NOR flash support

• ONFI1.0 NAND flash support (1-bit ECC)

• 1-bit SPI, 2-bit SPI, 4-bit SPI (quad-SPI), or two quad-SPI (8-bit) serial NOR flash

8-Channel DMA Controller

• Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and scatter-gather transaction support

I/O Peripherals and Interfaces

• Two 10/100/1000 tri-speed Ethernet MAC peripherals with IEEE Std 802.3 and IEEE Std 1588 revision 2.0 support

• Scatter-gather DMA capability

• Recognition of 1588 rev. 2 PTP frames

• GMII, RGMII, and SGMII interfaces

• Two USB 2.0 OTG peripherals, each supporting up to 12 Endpoints

• USB 2.0 compliant device IP core

• Supports on-the-go, high-speed, full-speed, and low-speed modes

• Intel EHCI compliant USB host

• 8-bit ULPI external PHY interface

• Two full CAN 2.0B compliant CAN bus interfaces

• CAN 2.0-A and CAN 2.0-B and ISO 118981-1 standard compliant

• External PHY interface

• Two SD/SDIO 2.0/MMC3.31 compliant controllers

• Two full-duplex SPI ports with three peripheral chip selects

• Two high-speed UARTs (up to 1 Mb/s)

• Two master and slave I2C interfaces

• GPIO with four 32-bit banks, of which up to 54 bits can be used with the PS I/O (one bank of 32b and one bank of 22b) and up to 64 bits (up to two banks of 32b) connected to the Programmable Logic
• Up to 54 flexible multiplexed I/O (MIO) for peripheral pin assignments

Interconnect

• High-bandwidth connectivity within PS and between PS and PL

• ARM AMBA AXI based

• QoS support on critical masters for latency and bandwidth control

Programmable Logic Features:

Configurable Logic Blocks (CLB)

• Look-up tables (LUT)

• Flip-flops

• Cascade able adders

36 Kb Block RAM

• True Dual-Port

• Up to 72 bits wide

• Configurable as dual 18 Kb block RAM

DSP Blocks

• 18 x 25 signed multiply

• 48-bit adder/accumulator

• 25-bit pre-adder

Programmable I/O Blocks

• Supports LVCMOS, LVDS, and SSTL

• 1.2V to 3.3V I/O

• Programmable I/O delay and SerDes

JTAG Boundary-Scan

• IEEE Std 1149.1 Compatible Test Interface

PCI Express Block

• Supports Root complex and End Point configurations

• Supports up to Gen2 speeds

• Supports up to 8 lanes

Serial Transceivers

• Up to 16 receivers and transmitters

• Supports up to 12.5 Gb/s data rates

Two 12-Bit Analog-to-Digital Converters

• On-chip voltage and temperature sensing

• Up to 17 external differential input channels

• One million samples per second maximum conversion rate

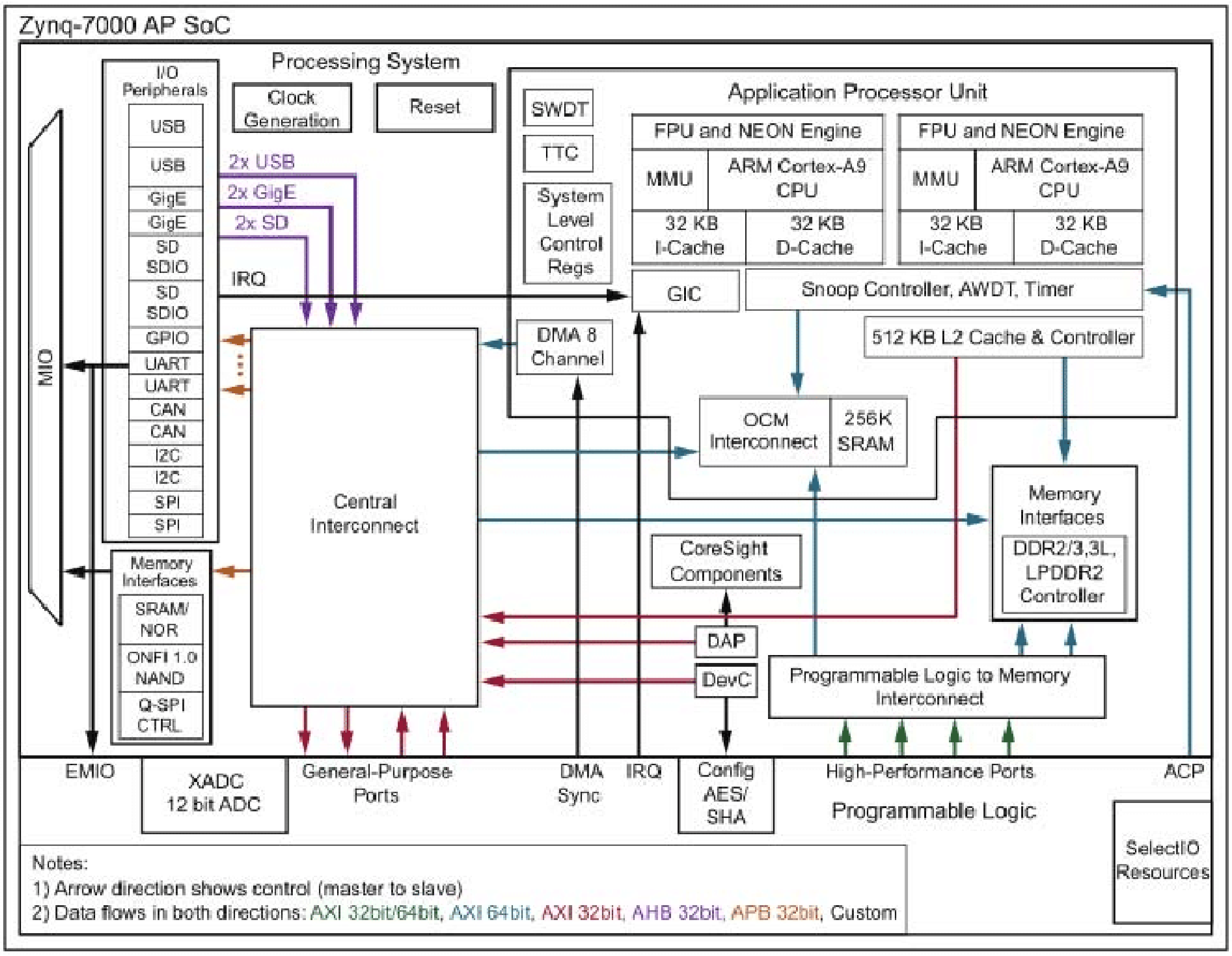


Figure 1.2 ZYNQ SOC Architecture

1.2 Advanced Extensible Interface (AXI)

The “advanced extensible interface” (AXI) bus is a high-performance, point-to-point, master-slave parallel bus used to connect on-chip peripheral circuits (or IP blocks) to processor cores. The complete AXI standard includes many high-performance features, like variable address and data bus widths, burst operations, advanced caching functions, and out-of-order transactions.

Many systems don’t require these high-performance features, so the simpler, “lighter weight” AXI4-Lite protocol was introduced in 2010 to provide system designers with a smaller and less complex bus interface (AXI4-Lite does not include variable bus widths or cache support and allows only one 32-bit data transfer per read/write transaction). The AXI4-Lite interface targets control and monitoring of IP blocks, and it is used in the ZYNQ device to connect the ARM and FPGA.

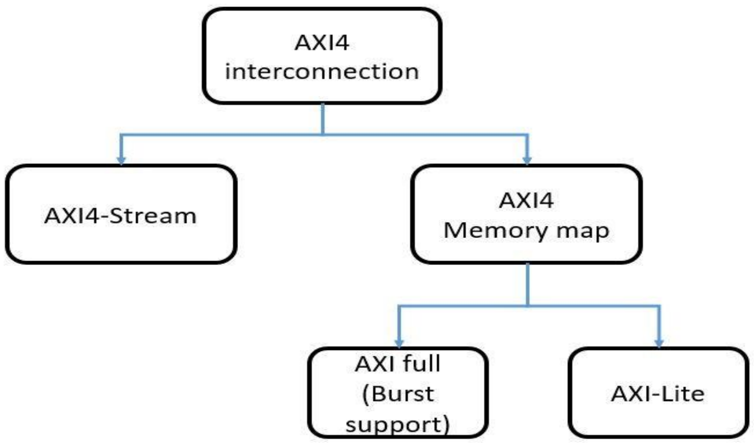


Figure 1.3 AXI Interfaces

AXI-Stream Protocol

The AXI-Stream protocol is used for applications that typically focus on a data-centric and data-flow paradigm where the concept of an address is not present or not required. Each AXI-Stream acts as a single unidirectional channel for a handshake data flow. At this lower level of operation (compared to the memory mapped AXI protocol types), the mechanism to move data between IP is defined and efficient, but there is no unifying address context between IP. The AXI-Stream channel is modeled after the write data channel of the AXI. Unlike AXI, AXI-Stream interfaces can burst an unlimited amount of data.

Memory Mapped Protocols

In memory mapped AXI (AXI3, AXI4, and AXI4-Lite), all transactions involve the concept of a target address within a system memory space and data to be transferred. Memory mapped systems often provide a more homogeneous way to view the system, because the IPs operate around a defined memory map.

AXI full:

It supports Burst mode transactions.

AXI Lite:

It supports beat mode transactions (Single transaction).

Both AXI full and AXI-Lite interfaces consist of five different channels:

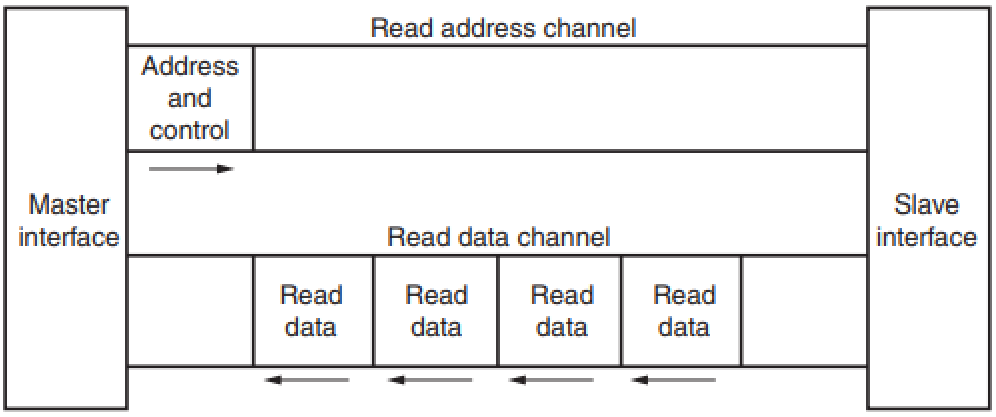
• Read Address Channel

• Write Address Channel

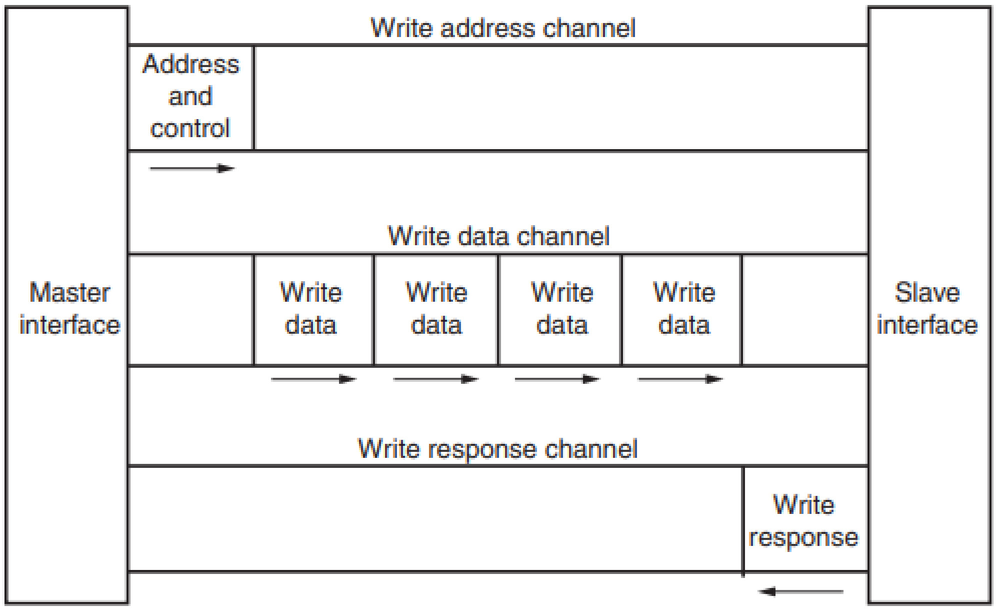
• Read Data Channel

• Write Data Channel

• Write Response Channel



(a)



(b)

Figure 1.4 AXI memory map channels

In its most basic configuration, the AXI protocol connects and facilitates communication between one master and one slave device. As expected, the master initiates and drives data requests, while the slave responds accordingly. This communication, or transactions as we will now refer to, occurs over multiple channels, each one dedicated to a specific purpose.

AXI Handshake Protocol

The sender must always assert a VALID signal before the receiver and keep it HIGH until the handshake is completed. By using handshakes, the speed and regularity of any data transfer can be controlled.

There are five channels, each one transmitting a data payload in one direction. Each channel implements a handshake mechanism, wherein the sender drives a VALID signal when it has prepared the payload for delivery and the receiver drives a READY signal in response when it is ready to receive the data. Data transfer is also known as a *beat*.

The five AXI4 channels are as follows:

- Write Address channel (AW): Provides address where data should be written (AWADDR)
- Can also specify burst size (AWSIZE), beats per burst (AWLEN+1), burst type (AWBURST), etc.
- AWVALID (Master to Slave) and AWREADY (Slave to Master)
- Write Data channel (W): The actual data sent (WDATA)
- Can also specify data and beat ID.
- Sender will always assert a finished transfer when done (WLAST)
- WVALID (Master to Slave) and WREADY (Slave to Master)
- Write Response channel (B): Status of write (BRESP)
- BVALID (Slave to Master) and BREADY (Master to Slave)
- Read Address channel (AR): Provides address where data should be read from (ARADDR)
- Can also specify burst size (ARSIZE), beats per burst (ARLEN+1), burst type (ARBURST), etc.
- ARVALID (Master to Slave) and ARREADY (Slave to Master)
- Read Data channel (R): The actual data sent back.
- Can also send back status (RRESP), data ID, etc.
- Sender will always assert a finished transfer when done (RLAST)
- RVALID (Slave to Master) and RREADY (Master to Slave)

Here is an example of a typical read/write AXI transaction.

- To write, the master first provides the address (0x0) to write to, as well as the data specifications (4 beats of 4 bytes each, data type of INCR). Both the master and slave then exchange a handshake for verification.
- The master then prepares and writes the actual data payload to send over the channel (0x10, 0x11 0x12, and 0x13), again using a handshake to verify the transfer. The master will signal the end of the payload to the slave using WLAST.
- The slave responds with a status of the write and whether it was successful or a failure (all OKAY in this case) and finishes the entire transaction with another handshake.

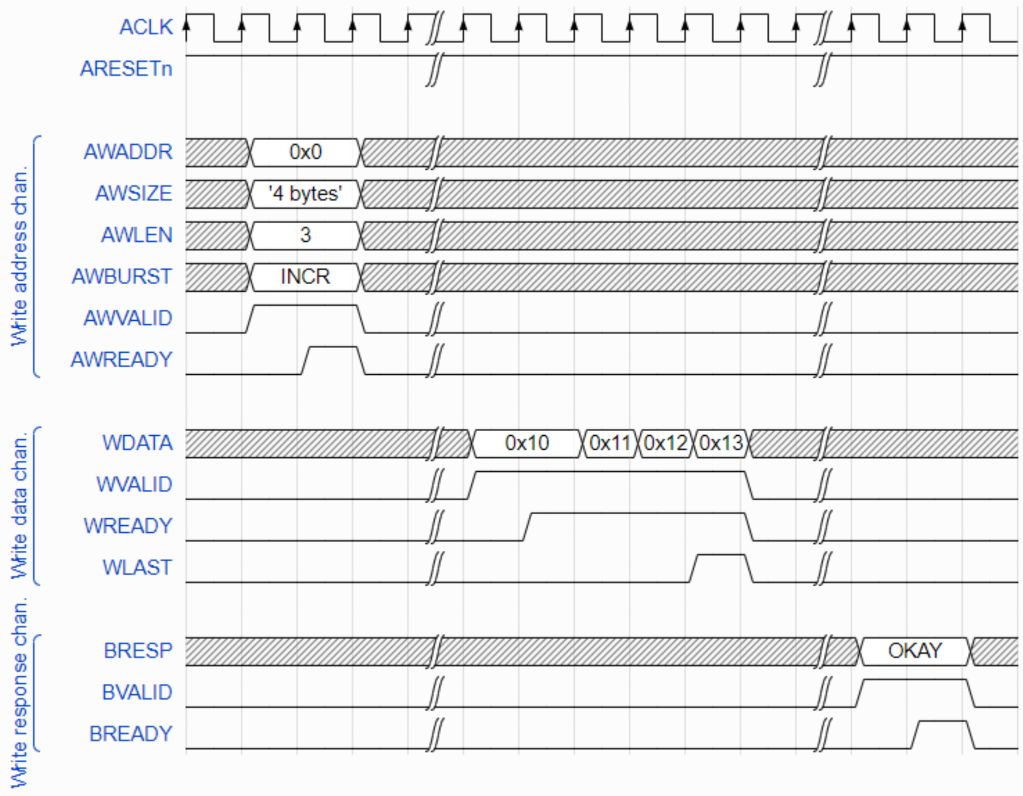


Figure 1.5 AXI Write transactions.

- To read, the master first provides the first address to read from (0x0), as well as the data specifications (4 beats of 4 bytes each, data type of INCR). The usual handshake occurs.
- The slave then provides the actual data payload, as well as the status of each beat (all beats are OKAY). The slave will signal the end of the payload to the master using RLAST. As we can see, what was written to the specified addresses was the same as what was read back.

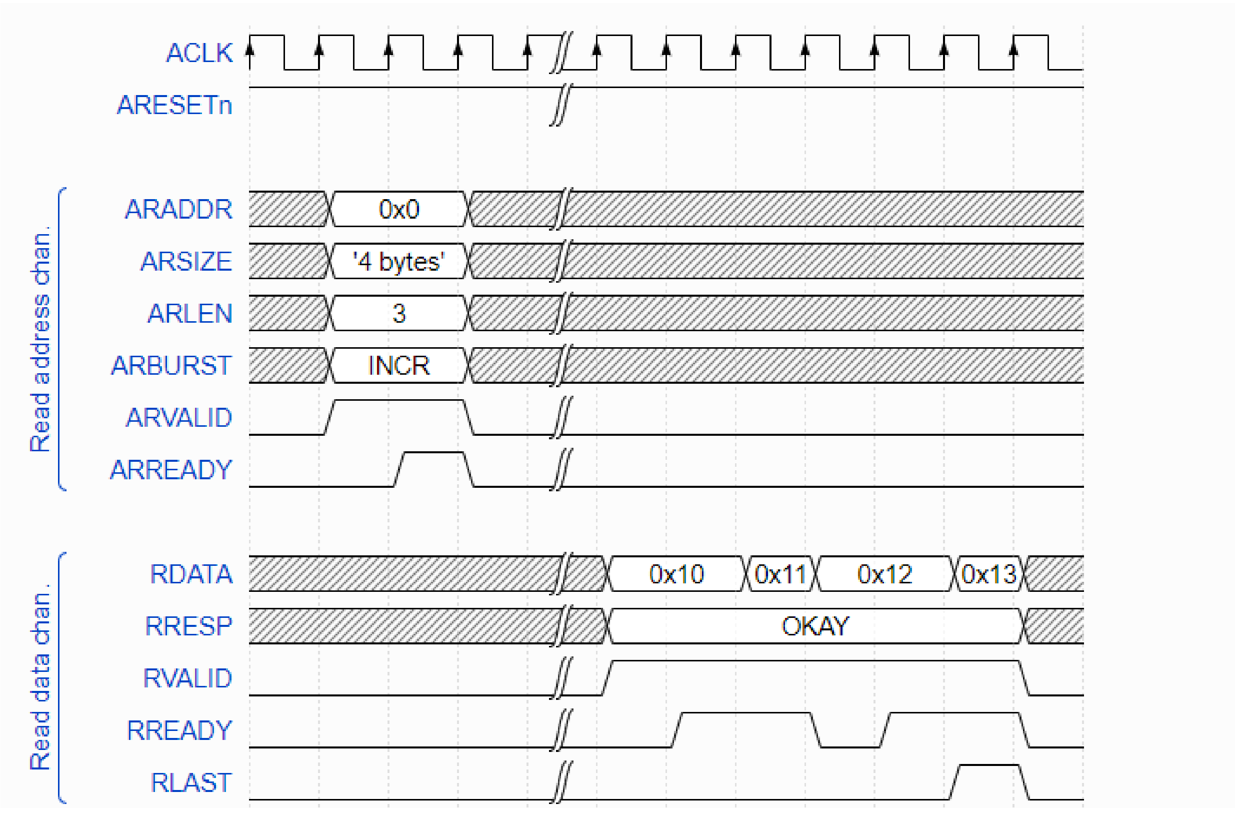


Figure 1.6 AXI read transactions.

1.3 PYNQ development board

In this project we are using PYNQ development board for processing system and programmable logic communication.

PYNQ boards, such as the PYNQ-Z1 or PYNQ-Z2, are designed for use with Xilinx Zynq SoCs and provide an accessible platform for FPGA development. These boards are equipped with various peripherals and interfaces, making them suitable for a wide range of embedded system development and FPGA prototyping tasks. Developers can use PYNQ boards in conjunction with the PYNQ framework to leverage the power of the Zynq SoC while using Python and Jupiter notebooks for programming and development.

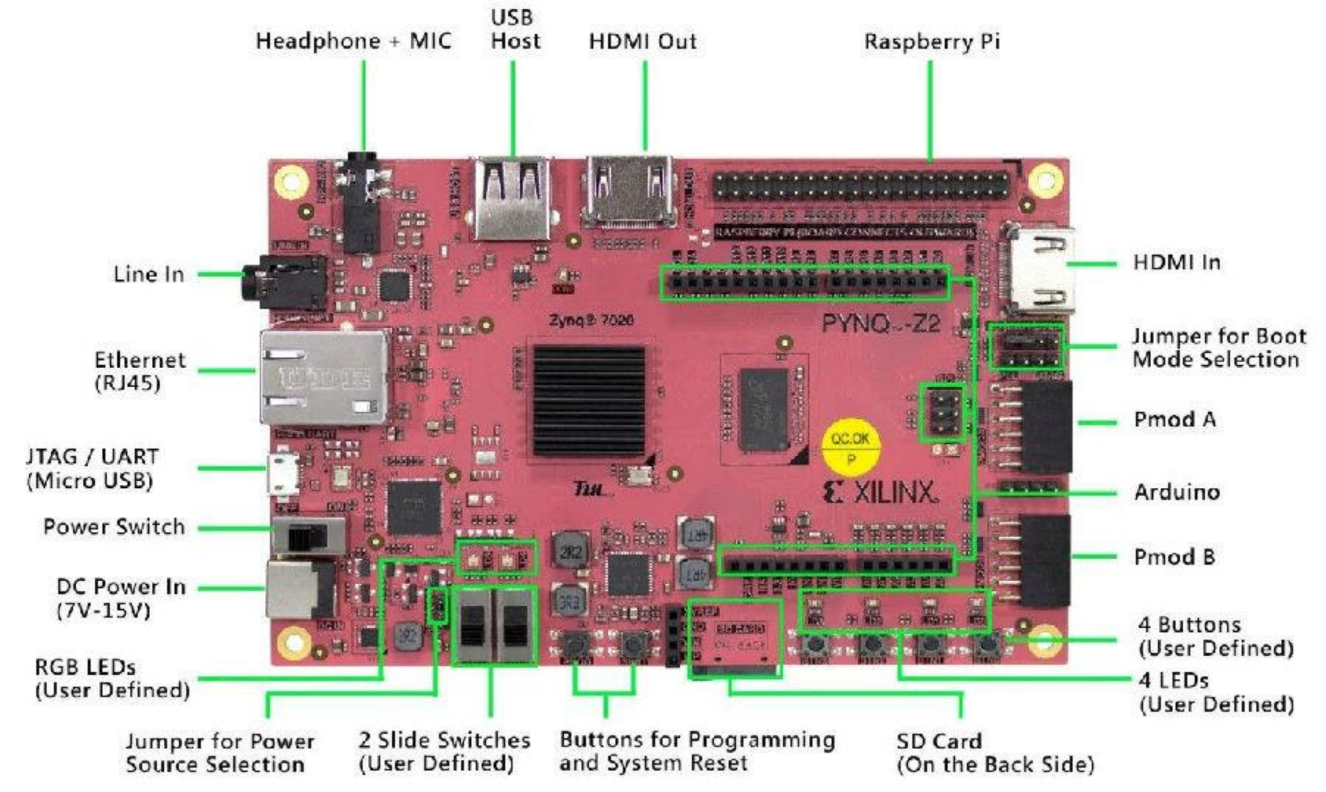


Figure 1.7 PYNQ Z2

1.3.1 Key features of PYNQ:

SOC

- Zynq-7000 SoC XC7Z020-1CLG400C

I/O interfaces

- USB-JTAG Programming circuitry
- USB OTG 2.0
- USB-UART bridge
- One 10/100/1G Ethernet
- HDMI Input
- HDMI Output
- I2S interface with 24bit DAC with 3.5mm TRRS jack
- Line-in with 3.5mm jack

Memory

- 512 Mbyte DDR3 with 16-bit bus @ 1050 Mbps

- 128 Mbit Quad-SPI Flash
- Micro SD card connector

Switches and LEDs

- 2 Slide switches
- 2 RGB LEDs
- 4 LEDs
- 4 Pushbuttons

Clocks

- One 125 MHz for PL
- One 50 MHz for PS

Expansion ports

- 2 Pmod ports
    - 16 Total FPGA I/O (8 shared pins with Raspberry Pi connector)
- 1 Arduino Shield connector
    - 24 Total FPGA I/O
    - 6 Single-ended 0-3.3V Analog inputs to XADC
- Raspberry Pi connector
    - 28 Total FPGA I/O (8 shared pins with Pmod A port)

Chapter 2

Problem Definition

2.1 Problem 1

PS & PL communication through user interface.

Provided Solution:

For PS & PL communication through user interface we are using Block Rams. PS is writing data on a user provided address of Block RAM which is read on VIO. VIO has five channels connected to Block memory generator IP: Address, Read, Write, Enable, and Strobe.

VIO can read or write from a specific address using these channels. So, the data which has been written by PS on a specific address can be read by VIO by providing that address and vice versa.

2.2 Problem 2

PS sending data to PL which is further transmitted over SPI.

Provided Solution:

In this part again PS and PL are using Block Rams to read and write data from over AXI interface. The data which is written by PS is read by SPI Logic controller which will provide it to SPI transmitter which is transmitting it to receiver and at receiver end we can see the data which is written by PS. All these signals are viewed using ILA.

For now, PS is writing on a fixed address and PL is reading from that address but for future we can modify it so that PS can write on any address and PL can read from various addresses.

2.3 BLOCK RAMS

Block RAM (BRAM) is a type of random-access memory embedded throughout an FPGA for data storage. The BRAM is a dual-port RAM module instantiated into the FPGA fabric to provide on-chip storage for a relatively large set of data. The two types of BRAM memories are available in a device and the available amount of these memories is device specific. The dual-port nature of these memories allows for parallel, same-clock-cycle access to different locations.

BRAMs can implement either RAM or a ROM, covering on-chip, local, and private memory types. In a RAM configuration, the data can be read and written at any time during the runtime of the circuit. In contrast, in a ROM configuration, data can only be read during the runtime of the circuit. The data of the ROM is written as part of the FPGA configuration and cannot be modified in any way.

Block RAMs come in a finite size; 4/8/16/32 kb (kilobits) are common. They have a customizable width and depth.

Single Port BRAM Configuration

The Single Port Block RAM configuration is useful when there is just one interface that needs to retrieve data. This is also the simplest configuration and is useful for some applications. One example would be storing Read-Only Data that is written to a fixed value when the FPGA is programmed. That's one thing about Block RAM, is that they can all be.

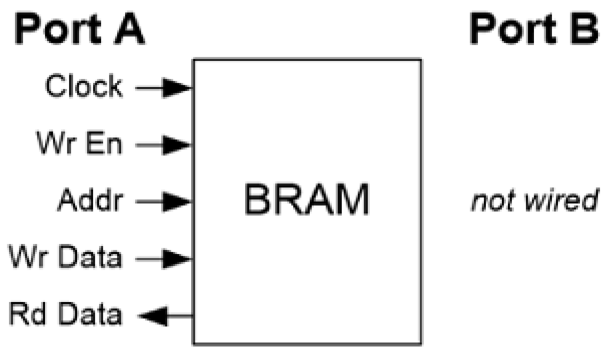


Figure 2.1 Single Port BRAM

Dual Port BRAM Configuration

The Dual Port Block RAM (or DPRAM) configuration behaves the same way as the single port configuration, except you have another port available for reading and writing data. Both Port A and Port B behave the same. Port A can perform a read on Address 0 on the same clock cycle that Port B is writing to address 200. Therefore, a DPRAM can perform a write on one address while reading from a completely different address. I personally find that I have more use cases for DPRAMs than I do for Single-Port RAMs.

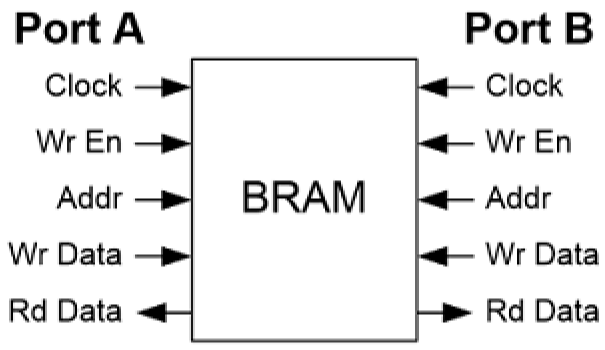


Figure 2.2 Dual port BRAM

Uses:

- A data or program store for a Micro Blaze or similar processor implemented in the FPGA fabric.
- Storing large look-up tables (e.g., converting Celsius to Fahrenheit)
- Storing read-only data such as calibration parameters
- Storing data read off external device such as ADC.
- Creating a FIFO to store temporary data such as raw video.
- Crossing clock domains using a FIFO
- In general, storing large amounts of data.

Chapter 3

Problem 1 Methodology

3.1 Block Diagram

PS & PL are communicating via Block Rams over AXI interface. PS is writing data on a user provided address of Block RAM which is read by PL on VIO. VIO has five channels connected to Block memory generator IP: Address, Read, Write, Enable and WE. If PS has written data on any address, we can view it on VIO read channel by providing that address on VIO address channel. We can also write data on any address by using VIO address channel and VIO read channel will read same data from that address, we can also read that data from PS read channel.

Here is the Block Diagram of problem 1 methodology:

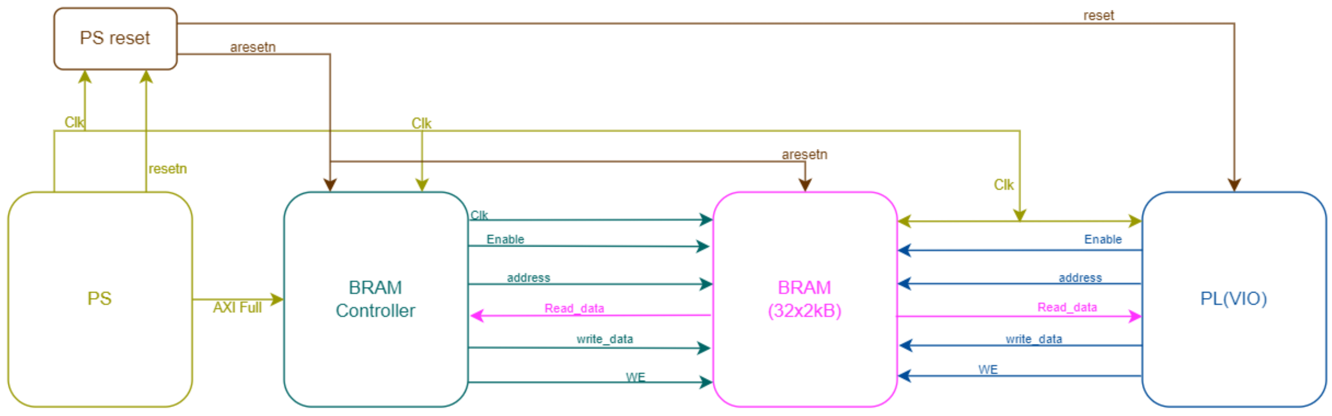


Figure 3.1 Problem 1 Block Diagram

Chapter 4

Problem 1 Implementation & Testing

4.1 Vivado Design Flow

Vivado is a popular integrated development environment (IDE) provided by Xilinx for designing and programming Xilinx FPGAs (Field-Programmable Gate Arrays) and other programmable logic devices. The design flow in Vivado typically follows a series of stages to take your design from initial concept to programming the FPGA. Here is a high-level overview of the design flow in Vivado:

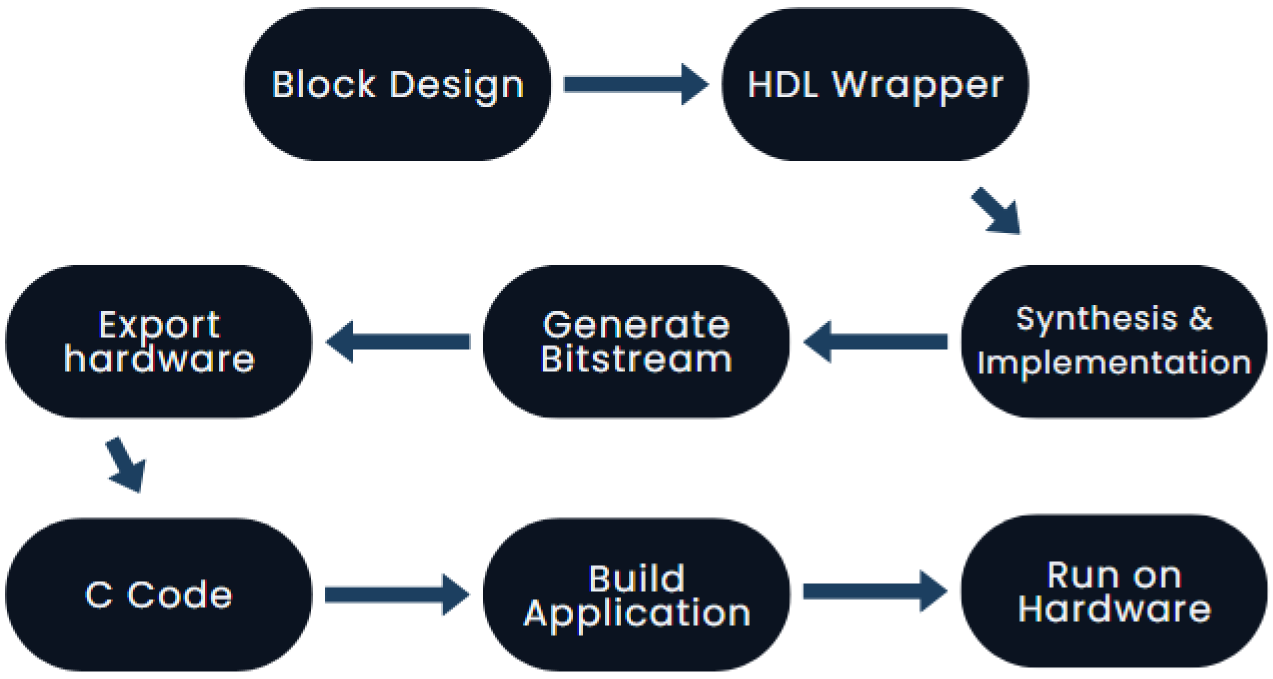


Figure 4.1 Problem 1 Design Flow

4.2 Vivado Block design

Figure 4.1 shows the vivado block diagram of Problem 1. It shows the functionality of different IPs used in this project and their relationship with each other.

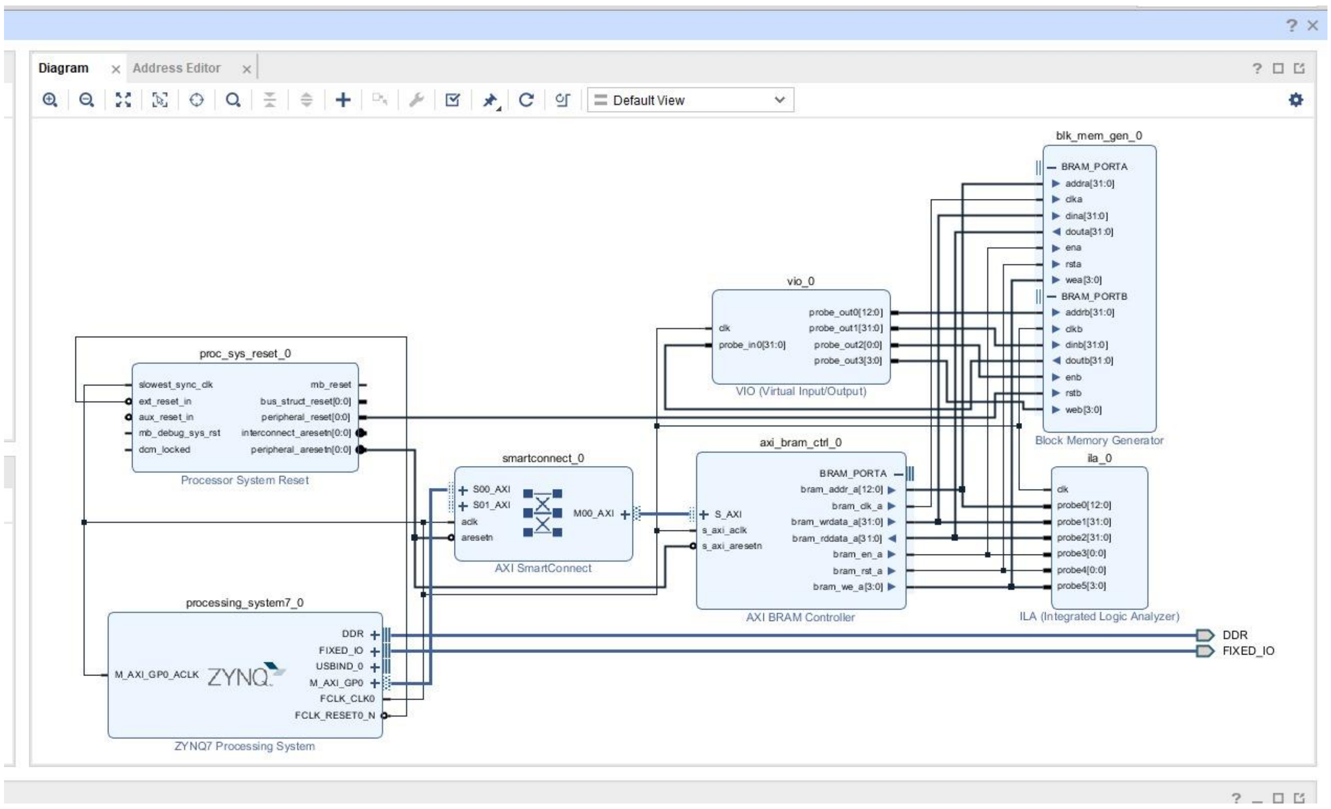


Figure 4.2 Problem 1 Block design

4.3 Program PS

To program PS, create a new application project in Xilinx select exported hardware from vivado and write C code in empty hello world file. Build your application and run it on hardware.

In this problem our Processing system is continuously asking to choose between write or read if we want to read data user need to provide address from which we need to read data and PS will read data from that address but if we want to write data user need to provide address at which we need to write data and data which should be written at that address.

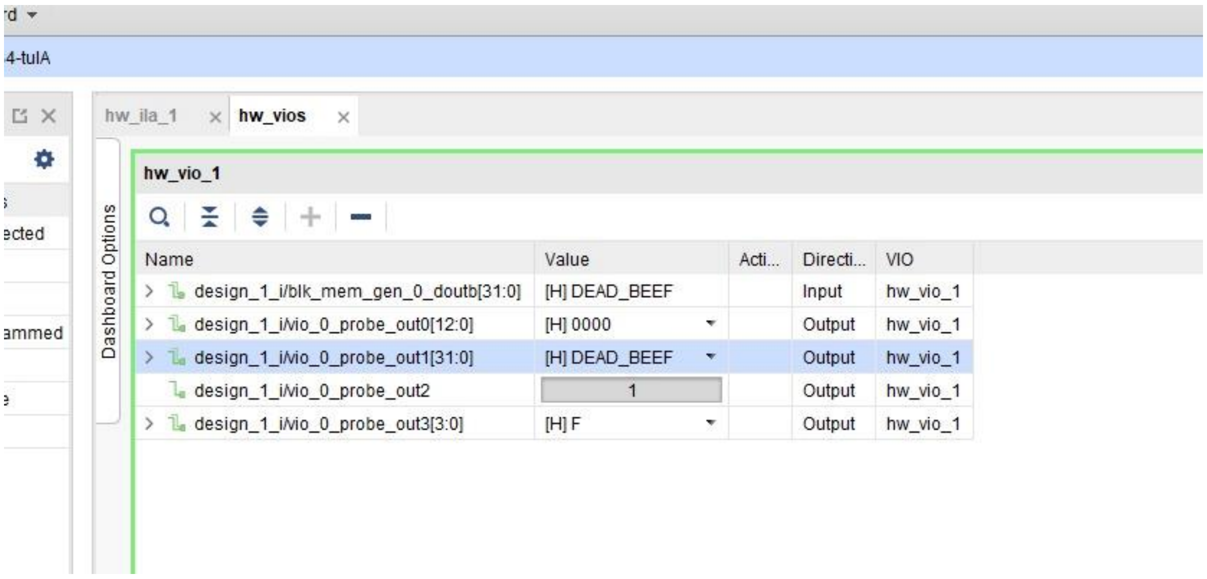
For code refer to appendix A.1

Chapter 5

Problem 1 Results and Discussion

5.1 PL is reading or writing Data.

Figure 5.1 shows the reading or writing of data from a certain memory address in BRAM by programmable logic (VIO).

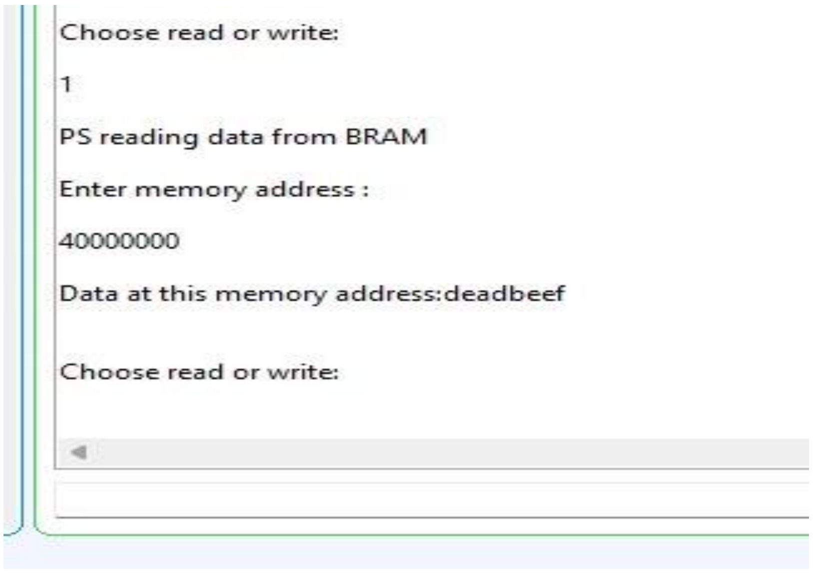


Name	Value	Acti...	Directi...	VIO
design_1_i/blk_mem_gen_0_doutb[31:0]	[H] DEAD_BEEF		Input	hw_vio_1
design_1_i/vio_0_probe_out0[12:0]	[H] 0000		Output	hw_vio_1
design_1_i/vio_0_probe_out1[31:0]	[H] DEAD_BEEF		Output	hw_vio_1
design_1_i/vio_0_probe_out2	1		Output	hw_vio_1
design_1_i/vio_0_probe_out3[3:0]	[H] F		Output	hw_vio_1

Figure 5.1 VIO

5.2 PS is reading or writing Data.

Figure 5.2 shows PS reading data from that memory address which is written by PL in BRAM.



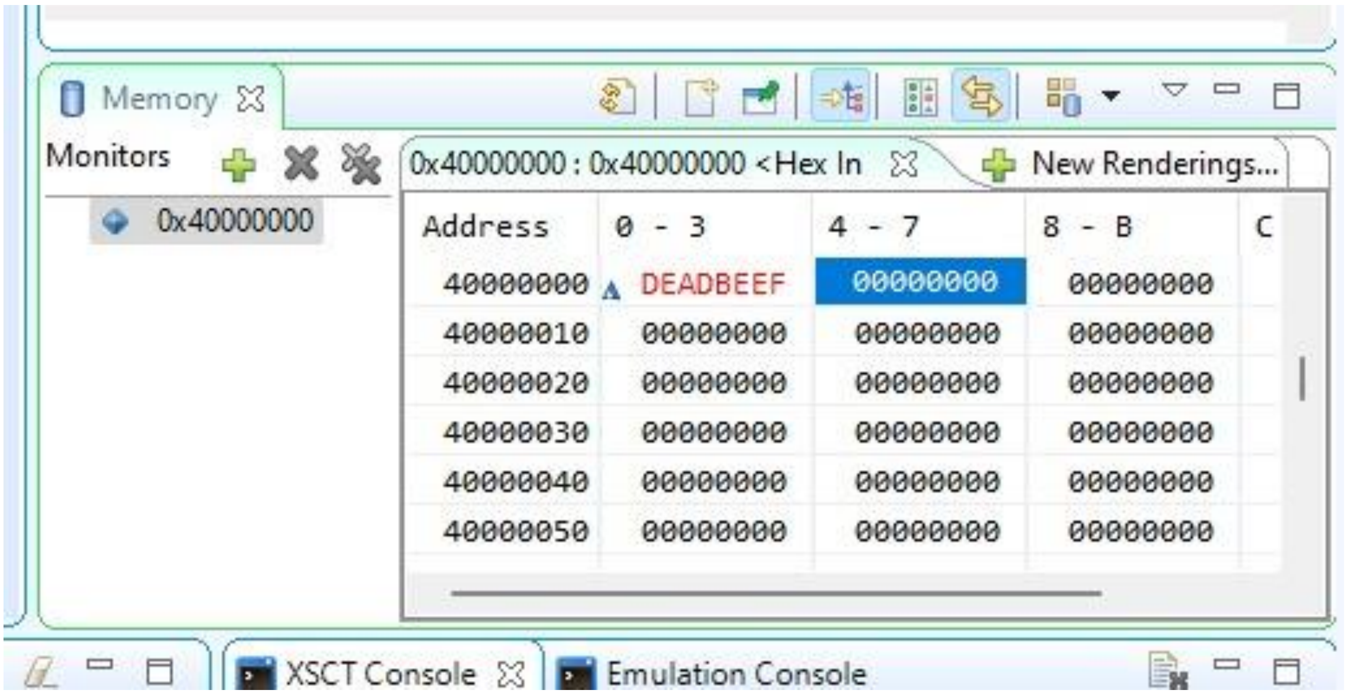
```
Choose read or write:
1
PS reading data from BRAM
Enter memory address :
40000000
Data at this memory address:deadbeef

Choose read or write:
```

Figure 5.2 PS is reading data

5.2.1 Memory

Figure 5.2.1 shows that memory address which is written by PL in BRAM.

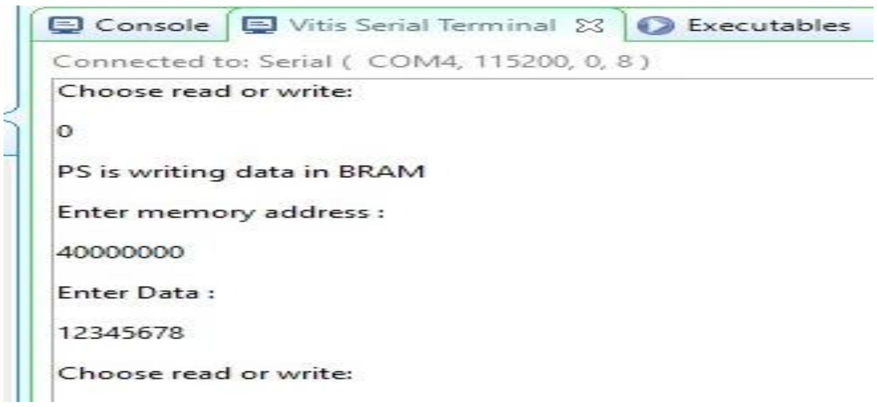


```
Address    0 - 3      4 - 7      8 - B      C
40000000   DEADBEEF   00000000   00000000
40000010   00000000   00000000   00000000
40000020   00000000   00000000   00000000
40000030   00000000   00000000   00000000
40000040   00000000   00000000   00000000
40000050   00000000   00000000   00000000
```

Figure 5.3 Memory map

5.2.2 PS is writing data.

Figure 5.2.2 shows PS is writing data on a memory address.

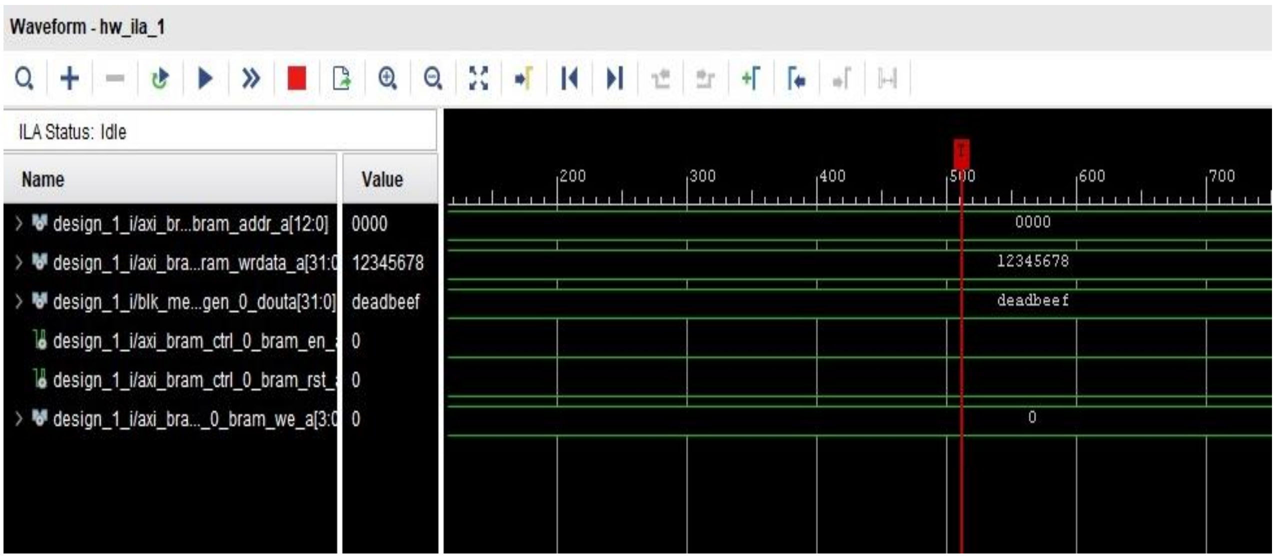


```
Connected to: Serial ( COM4, 115200, 0, 8 )
Choose read or write:
0
PS is writing data in BRAM
Enter memory address :
40000000
Enter Data :
12345678
Choose read or write:
```

Figure 5.4 PS writing data

5.3 PS signals on ILA.

Figure 5.3 shows PS is writing data on a memory address and also reading previous data from that address on ILA.



Name	Value
design_1_i/axi_br...bram_addr_a[12:0]	0000
design_1_i/axi_bra...ram_wrdata_a[31:0	12345678
design_1_i/blk_me...gen_0_douta[31:0]	deadbeef
design_1_i/axi_bram_ctrl_0_bram_en_	0
design_1_i/axi_bram_ctrl_0_bram_rst_	0
design_1_i/axi_bra..._0_bram_we_a[3:0	0

Figure 5.5 ILA

Chapter 6

Problem 1 Resource Utilization Report

```
Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
-----------------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2019.2 (win64) Build 2708876 Wed Nov  6 21:40:23 MST 2019
| Date         : Thu Sep  7 15:53:38 2023
| Host         : Areeba running 64-bit major release  (build 9200)
| Command      : report_utilization -file design_1_wrapper_utilization_synth.rpt -pb design_1_wrapper_utilization_synth.pb
| Design       : design_1_wrapper
| Device       : 7z020clg400-1
| Design State : Synthesized
-----------------------------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Slice Logic Distribution
3. Memory
4. DSP
5. IO and GT Specific
6. Clocking
7. Specific Feature
8. Primitives
9. Black Boxes
10. Instantiated Netlists
1. Slice Logic
--------------
```

Site Type	Used	Fixed	Available	Util%

Slice LUTs*	0	0	53200	0.00
LUT as Logic	0	0	53200	0.00
LUT as Memory	0	0	17400	0.00
Slice Registers	0	0	106400	0.00
Register as Flip Flop	0	0	106400	0.00
Register as Latch	0	0	106400	0.00
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	-	-	-
0	-	-	Set
0	-	-	Reset
0	-	Set	-
0	-	Reset	-
0	Yes	-	-
0	Yes	-	Set
0	Yes	-	Reset
0	Yes	Set	-
0	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Available	Util%
Slice	0	0	13300	0.00
SLICEL	0	0		
SLICEM	0	0		
LUT as Logic	0	0	53200	0.00
LUT as Memory	0	0	17400	0.00
LUT as Distributed RAM	0	0		
LUT as Shift Register	0	0		
Slice Registers	0	0	106400	0.00
Register driven from	0			

within the Slice				
Register driven from outside the Slice	0			
Unique Control Sets	0	0	13300	0.00

3. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	140	0.00
RAMB36/FIFO*	0	0	140	0.00
RAMB18	0	0	280	0.00

4. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	220	0.00

5. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	0	0	125	0.00
Bonded IPADs	0	0	2	0.00
Bonded IOPADs	0	0	130	0.00
PHY_CONTROL	0	0	4	0.00
PHASER_REF	0	0	4	0.00
OUT_FIFO	0	0	16	0.00

IN_FIFO	0	0	16	0.00
IDELAYCTRL	0	0	4	0.00
IBUFDS	0	0	121	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	200	0.00
ILOGIC	0	0	125	0.00
OLOGIC	0	0	125	0.00

6. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	0	0	32	0.00
BUFIO	0	0	16	0.00
MMCME2_ADV	0	0	4	0.00
PLLE2_ADV	0	0	4	0.00
BUFMRCE	0	0	8	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	16	0.00

7. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

8. Primitives

Ref Name	Used	Functional Category

9. Black Boxes

Ref Name	Used
design_1_vio_0_0	1
design_1_smartconnect_0_0	1
design_1_processing_system7_0_0	1
design_1_proc_sys_reset_0_0	1
design_1_ila_0_0	1
design_1_blk_mem_gen_0_0	1
design_1_axi_bram_ctrl_0_0	1

10. Instantiated Netlists

Ref Name	Used

Chapter 7

Problem 2 Methodology

7.1 Block Diagram

In this part again PS and PL are communicating via Block Rams over AXI interface the data which is written by PS is read by SPI Logic controller which will provide it to SPI transmitter and receiver and at receiver end we can see the data which is written by PS. All these signals are viewed using ILA.

Here is the Block Diagram of problem 1 methodology:

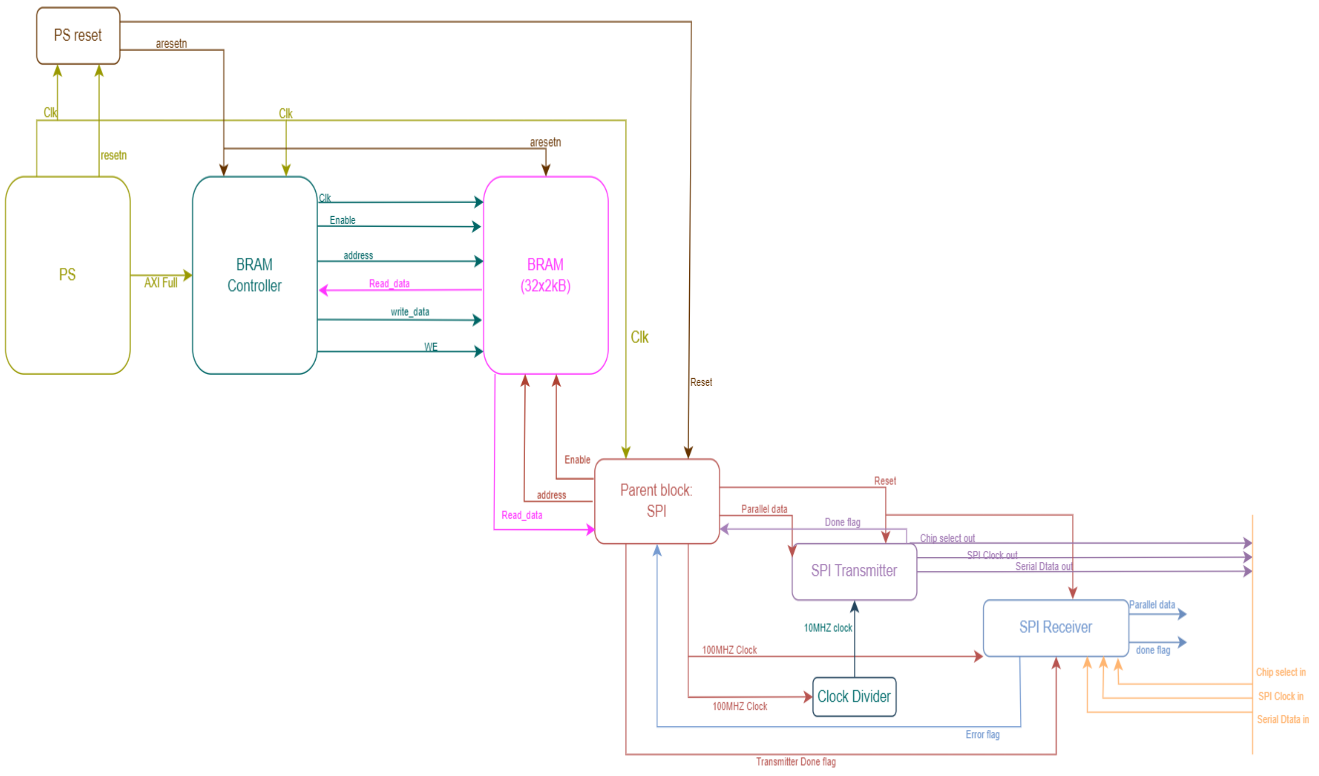


Figure 7.1 Problem 2 Block Diagram

Chapter 8

Problem 2 Implementation & Testing

8.1 SPI

SPI (Serial Peripheral Interface) is a synchronous serial communication protocol that enables high-speed, full-duplex communication between a master device (usually a microcontroller) and multiple slave devices (such as peripherals, sensors, or other microcontrollers). It offers a straightforward and cost-effective way to connect and control various hardware components. Unlike UART, which employs asynchronous communication over RX and TX lines, SPI employs synchronous communication with a clock signal. This ensures controlled data transfer and consistent data rates between the transmitting and receiving devices.

8.1.1 Key features of SPI

1. Master-Slave Relationship: SPI operates in a master-slave configuration. The master device (typically a microcontroller) controls the communication process, while the slave devices respond to the master's commands.

2. Single Master, Multiple Slaves: An SPI bus supports a single master but can connect to multiple slave devices. Each slave is selected using a Chip Select (CS) or Slave Select (SS) line.

3. Four Signal Lines: The SPI bus employs four signal lines: - Master Out/Slave In (MOSI): Data is transmitted from the master to the slave. - Master In/Slave Out (MISO): Data is transmitted from the slave to the master. - Serial Clock (SCLK): This clock signal synchronizes data transmission between devices. - Chip Select (CS)/Slave Select (SS): Used by the master to select the target slave device for communication.

4. Shift Registers and Data Latches: Both master and slave devices include shift registers and data latches. The master device initiates data transmission by shifting data into the MOSI line, while the slave responds by shifting data into the MISO line. Data is then latched for further processing.

5. Loop Connection: The shift registers of both master and slave devices are interconnected, forming a loop. This loop ensures that data can be continuously transmitted between the devices.

6. Data Sizes: The typical size of the shift registers is 8 bits, but larger sizes, such as 16 bits, are also common. Overall, SPI provides a versatile and efficient means of connecting various hardware components, including memory modules, sensors, displays, and more. Its synchronous nature, precise clocking, and support for multiple slaves make it suitable for applications requiring high-speed data exchange and precise control over communication.

The following image shows the minimal system requirements for both the devices.

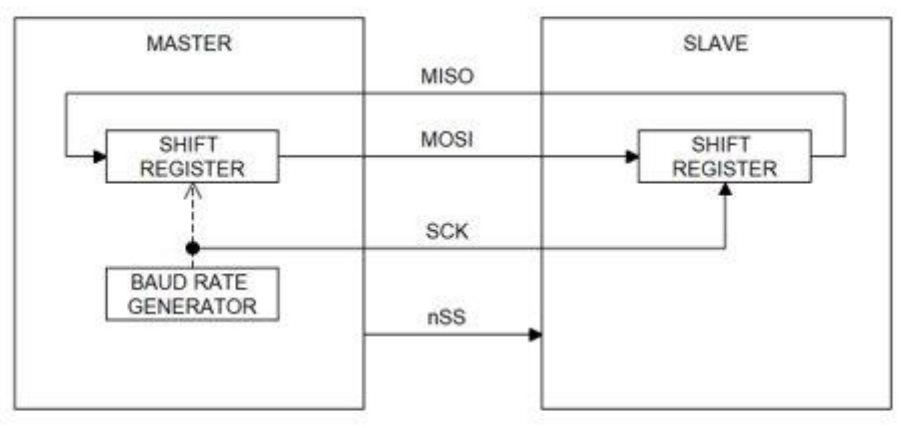


Figure 8.1 SPI signals

From the image, the Master device consists of a Shift Register, a data latch, and a clock generator. The slave consists of similar hardware: a shift registers and a data latch. Both the shift registers are connected to form a loop. Usually, the size of the register is 8 – bits but higher size registers of 16 – bits are also common.

During the positive edge of the clock signal, both the devices (master and slave) read input bit into LSB of the register. During the negative cycle of the clock signal, both the master and slave place a bit on its corresponding output from the MSB of the shift register.

Hence, for each clock cycle, a bit of data is transferred in each direction i.e., from master to slave and slave to master. So, for a byte of data to be transmitted from each device, it will take 8 clock cycles.

8.1.2 SPI modes of operation

In SPI, the main can select the clock polarity and clock phase. The CPOL bit sets the polarity of the clock signal during the idle state. The idle state is defined as the period when CS is high and transitioning to low at the start of the transmission and when CS is low and transitioning to high at the end of the transmission. The CPHA bit selects the clock phase. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data. The main must select the clock polarity and clock phase, as per the requirement of the sub node. Depending on the CPOL and CPHA bit selection, four SPI modes are available.

Mode 0:

In this mode, clock polarity is 0, which indicates that the idle state of the clock signal is low. The clock phase in this mode is 0, which indicates that the data is sampled on the rising edge and the data is shifted on the falling edge of the clock signal.

Mode 1

In this mode, clock polarity is 0, which indicates that the idle state of the clock signal is low. The clock phase in this mode is 1, which indicates that the data is sampled on the falling edge (shown by the orange dotted line) and the data is shifted on the rising edge (shown by the dotted blue line) of the clock signal.

Mode 2:

In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 0, which indicates that the data is sampled on the rising edge and the data is shifted on the falling edge of the clock signal.

Mode 3:

In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 1, which indicates that the data is sampled on the falling edge and the data is shifted on the rising edge of the clock signal.

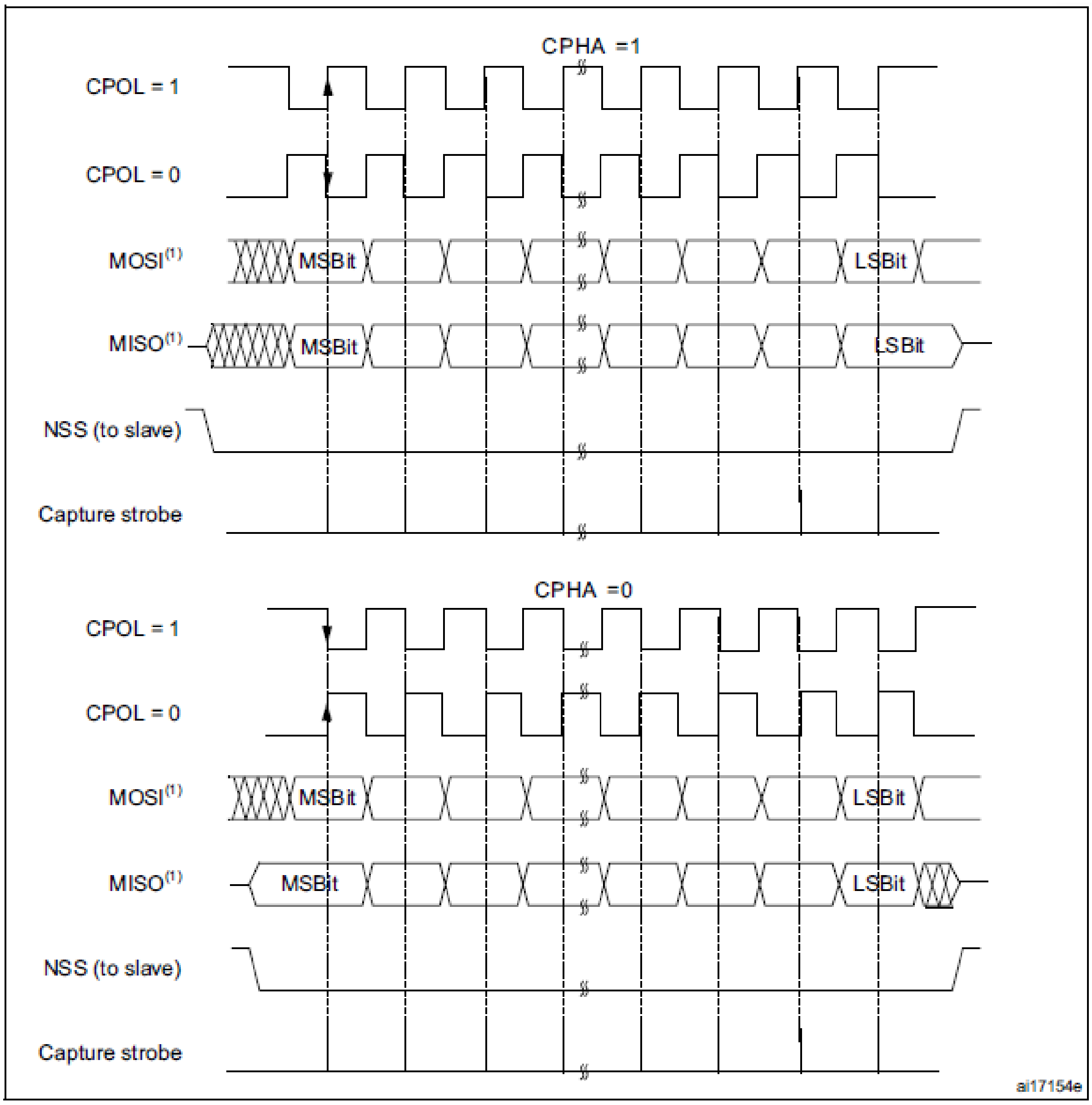


Figure 8.2 SPI modes

8.1.3 Advantages of SPI

1. The main advantage of the SPI is to transfer the data without any interruption.

2. It is simple hardware.

3. It provides full-duplex communication.

4. There is no need for a unique address of the slave in this protocol.

5. This protocol does not require precise oscillation of slave devices because it uses the master's clock.

6. In this, software implementation is very simple.

7. It provides high transfer speeds.

8. Signals are unidirectional.

9. It has separate lines of MISO and MOSI, so the data can be sent and received at the same time.

8.1.4 Disadvantages of SPI

1. Usually, it supports only one master.

2. It does not check the error like the UART.

3. It uses more pins than the other protocol.

4. It can be used only from a short distance.

5. It does not give any acknowledgment that the data is received or not.

8.1.5 Applications of SPI

Applications of SPI

1. Memory: SD Card, MMC, EEPROM, and Flash.

2. Sensors: Temperature and Pressure.

3. Control Devices: ADC, DAC, digital POTS, and Audio Codec.

4. Others: Camera Lens Mount, Touchscreen, LCD, RTC, video game controller, etc.

8.2 SPI module Design

8.2.1 Parent block control logic

This module contains instances of SPI transmitter and receiver, with additional logic for reading parallel input from BRAM at address 0. The limited clock signal, chip select, and serial data generated by the transmitter is transmitted to the receiver.

- All the ports and registers are defined in this section.
- Taking reset, clk_in (slow clock) from processing system.
- It has a set of outputs including pi (parallel input to transmitter), s_d_rx (serial data line), cs_rx (chip select signal), po (parallel output from receiver), shift_done_r (receiver shift done flag), next_state_r (receiver's next state), and state_r (receiver's current state).
- There are also some debugging signals to check for error, Tx, and Rx states.
- In this module First 4 bits of parallel out data from receiver are assigned to 4 LEDs on PYNQ to check data out.

For above explained code refer to appendix B.1

8.2.2 Clock divider

Clock divider can divide slow clock to make faster clock. The clock divider's function is essential for controlling timing-critical operations, enabling synchronization, and maintaining efficient data processing in various digital applications.

- This section of code defines a Verilog module named clk_divider.
- It takes two ports: clk_in (input clock) and clk_out (output clock).
- The module also has a parameter clk_count to specify how half cycle of a fast clock can contain how many cycles of slow clock.
- A 4-bit wide register named counter is declared. It is used as a counter to track the number of clock cycles.

For code reference see appendix B.2.1

- In this section the counter value is incremented by 1 on every positive edge of the clk_in signal.
- It checks if the value of counter has reached the specified clk_count.
- If true, it toggles the clk_out signal and resets the counter back to 0.
- This division of frequency effectively generates the desired clock frequency at clk_out.

For code reference see appendix B.2.2

8.2.3.1 SPI transmitter

This module can be used for scenarios where parallel data needs to be efficiently transmitted over a serial communication channel. The Clock Divider guarantees accurate timing synchronization, and the state machine arranges the entire process, ensuring that the conversion from parallel to serial is executed correctly. Once the transmission is finished, the shift_done flag provides a clear indication of the completion, making it an essential component for reliable serial data communication.

In this section of code all the ports and registers are defined.

For above explained code reference see appendix B.3.1

In this section we are making instance of the clock divider module so that we can provide divided clock to our transmitter.

For above explained code reference see appendix B.3.2

- In this code section, a clock signal clk is generated based on the falling edge of the input clock clk_in.
- When the latch_t signal is high (1), the clock signal clk is set to low (0), effectively halting its operation.
- When latch_t is low, the clock signal clk takes on the value of clkout, which is the output clock of the clock divider.
- Transmitter is generating clock for receiver which it will provide along with data to receiver.

For above explained code reference see appendix B.3.3

8.2.3.2 State machine of SPI transmitter

In the Control path of state machine our state will be idle (state 0) when we reset our transmitter otherwise our state will be next_state.

For above explained code reference see appendix B.3.4

This section of code is the next state control path of our state machine which is thoroughly explained in the figure below:

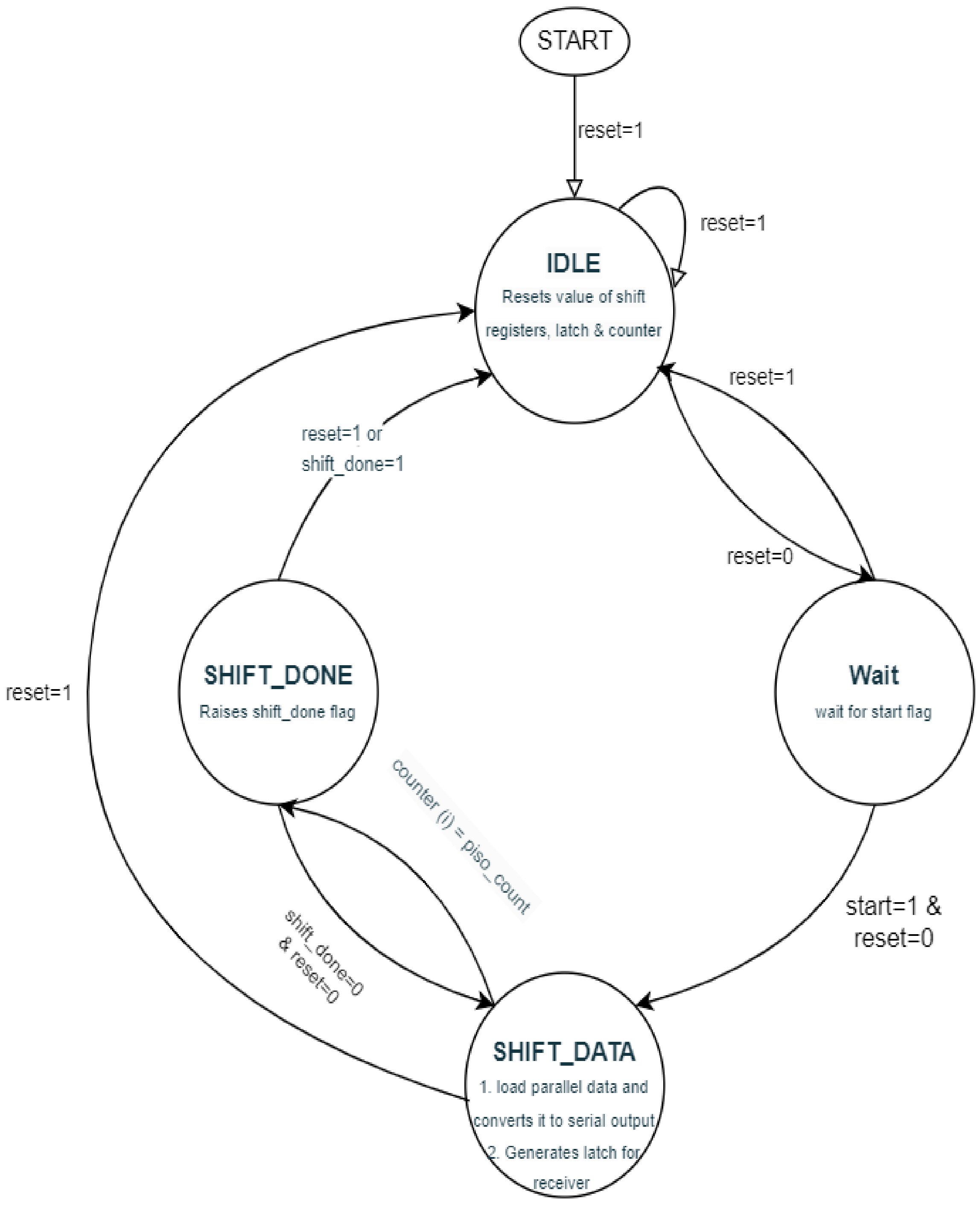


Figure 8.3 Transmitter state machine

In this state machine we have 4 states:

1. **IDLE (state 0)**
2. **WAIT (state 1)**
3. **SHIFT_DATA (state 2)**
4. **SHIFT_DONE (state 3)**

IDLE State:

When the Reset is 1, the module is in the IDLE state. The shift_done flag is cleared (set to 0). The parallel input pi is loaded into the parallel-to-serial shift register m. The counter i is initialized to 0. The shift_done flag is cleared (set to 0). The latch signal latch_t is set to 1.

In this state if Reset=1 it will remain in IDLE state. Otherwise, if Reset=0 it will go to state 1 which is Wait.

Wait State:

In this state the start flag is raised.

if Start=1 it will go to SHIFT_DATA state. Otherwise, if Reset=1 it will go to state 0 which is Idle.

SHIFT_DATA State:

When the Reset is 0 and start flag is raised, the module is in the SHIFT_DATA state. The parallel input pi is continuously loaded into the shift register ‘m’. The latch signal latch_t is set to 0, allowing data to be shifted out. If the counter ‘i’ is less than the specified piso_count, the shift register ‘m’ is shifted left, and its MSB is assigned to the serial out ‘so’ output. The counter ‘i’ is incremented.

In this state if our counter ‘i’ has reached the value of parameter ‘piso_count’ then it will go to state SHIFT_DONE otherwise, it will remain in state SHIFT_DATA.

SHIFT_DONE State:

In this state, the module waits until the counter 'i' reaches piso_count. Once the counter reaches the desired count, the shift_done flag is set to 1 indicating that the shift operation is complete. The counter 'i' is reset to 0, and the latch signal latch_t is set to 1 to prepare for the next transmission.

In this state if our done flag has raised (shift_done=1) or Reset=1 then it will go to state IDLE but if done flag has not raised (shift_done=0) then it will go back to state SHIFT_DATA.

8.2.4.1 SPI Receiver

Overall, the SIPO module operates as a serial-to-parallel shift register with control logic driven by a state machine. It converts serial input data into parallel data and raises a shift_done_r flag when the conversion is complete. The control path manages state transitions, while the data path handles data shifting and output storage.

In the above section all the ports and registers are defined for SPI Receiver module and positive edge detector to detect positive edge of SPI clock.

For above explained code reference see appendix B.4.1

8.2.4.2 SPI Receiver State Machine

Control path for state machine that our state will be idle_r (state 0) when we reset our transmitter otherwise our state will be next_state_r (state 1).

For above explained code reference see appendix B.4.2

This section of code is the next state control path of our state machine which is thoroughly explain in the figure below:

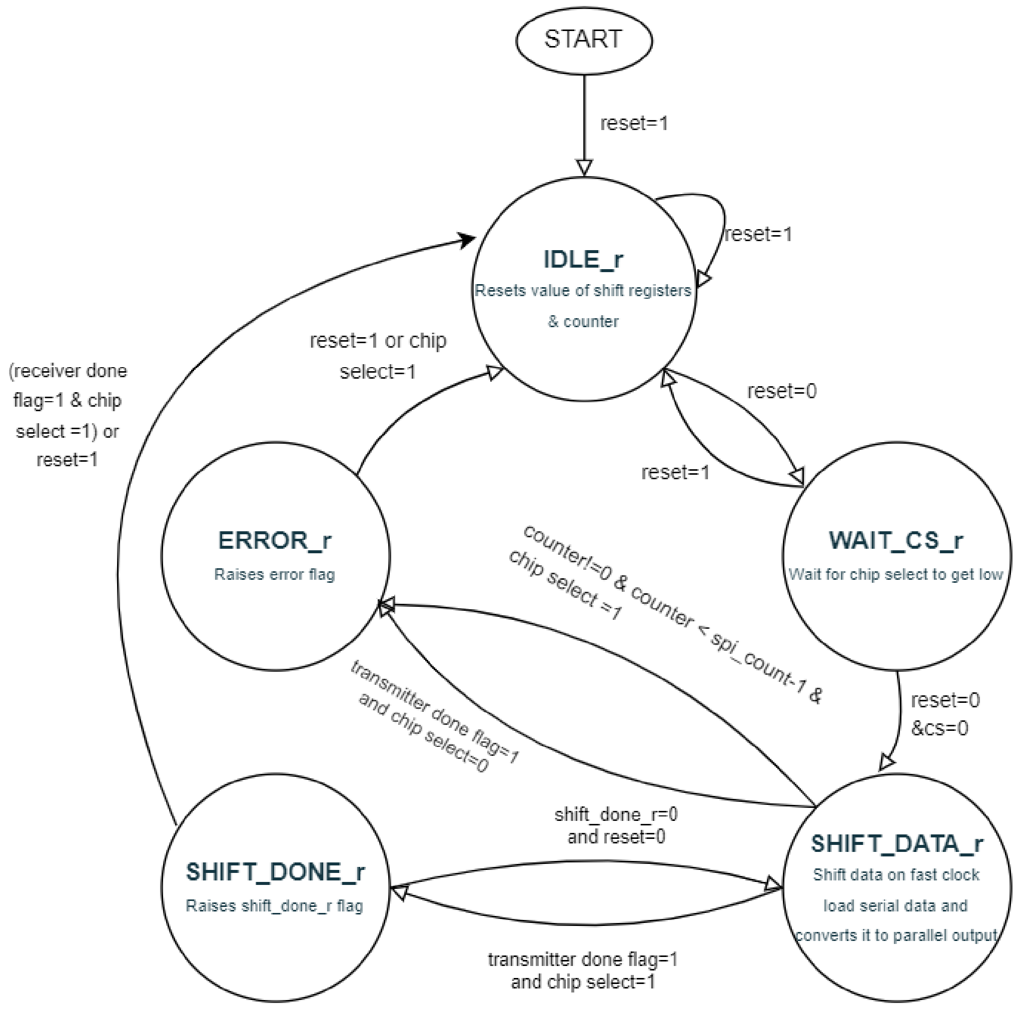

START
reset=1
IDLE_r
Resets value of shift registers & counter
reset=1
reset=1 or chip select=1
(receiver done flag=1 & chip select =1) or reset=1
reset=0
reset=1
ERROR_r
Raises error flag
WAIT_CS_r
Wait for chip select to get low
counter!=0 & counter < spi_count-1 & chip select =1
transmitter done flag=1 and chip select=0
reset=0 &cs=0
shift_done_r=0 and reset=0
SHIFT_DONE_r
Raises shift_done_r flag
SHIFT_DATA_r
Shift data on fast clock load serial data and converts it to parallel output
transmitter done flag=1 and chip select=1

Figure 8.4 Receiver state machine

In this state machine we have 5 states:

1.IDLE_r (state 0)

2. WAIT_CS_r (state 1)

2.SHIFT_DATA_r (state 2)

3.SHIFT_DONE_r (state 3)

5. ERROR_r(state 4)

1.IDLE_r State:

If mode is 1 (reset condition), the module is in the IDLE_r state. The shift_done_r flag is cleared (set to 0). Parallel output (po) sets to zero and counter (j) sets to zero.

In this state if Reset=1 it will remain in IDLE_r state. Otherwise, if Reset=0 it will go to state 1 which is WAIT_CS_r.

2. WAIT_CS_r State

In this state our receiver is waiting for chip select from transmitter.

If chip select=0 it will go to state shift_data_r state. Otherwise, if chip select=1 it will remain in state WAIT_CS_r.

3.SHIFT_DATA_r State:

In this state a positive edge detector is implemented according to which it will shift data only at the positive edge of the SPI clock. If the counter ‘j’ is less than or equal to the specified sipo_count serial input pi is continuously loaded into the LSB of shift register p_o and shift register is shifted left. The counter ‘j’ is incremented. If the counter is greater than sipo_count, the shift_done_r flag is set to 1, indicating that all data has been received and stored in the form of parallel data.

In this state if our transmitter has raised done flag and chip select is high then it will go to state SHIFT_DONE_r.

But if our transmitter has raised done flag and chip select is still low which is condition of overflow then it will go to state ERROR_r

In case of under flow that our counter has not been completed yet, but our chip select gets high it will again go to state ERROR_r.

4.SHIFT_DONE_r State:

In this state, the module waits until the counter ‘j’ is greater than sipo_count. Once the counter reaches the desired count, the shift_done_r flag is set to 1 to indicate that the shift operation is complete. The counter ‘j’ is reset to 0 to receive the next coming data.

SHIFT_DONE_r: In this state if our done flag has raised (shift_done_r=1) or reset=1 then it will go to state IDLE_r but if done flag has not raised (shift_done=0) and reset=0 then it will go back to state SHIFT_DATA_r.

5.ERROR_R State:

In this state after raising ERROR_r flag it will go back to IDLE_r state

For above explained code reference see appendix B.4.3

In the above given code parallel data from p_o register will be assigned to output ‘po’ only when a done flag is raised, and this done flag is raised after all the serial data has been successfully stored in p_o register.

For above explained code reference see appendix B.4.4

8.3 Vivado Design Flow

The Vivado Design Suite offers multiple ways to accomplish the tasks involved in Xilinx device design, implementation, and verification. We can use the traditional register transfer level (RTL)- to-bitstream FPGA design flow. We can also use system-level integration flows that focus on intellectual property (IP)-centric design and C-based design.

Figure 8.1 shows the vivado design flow of my problem 2.

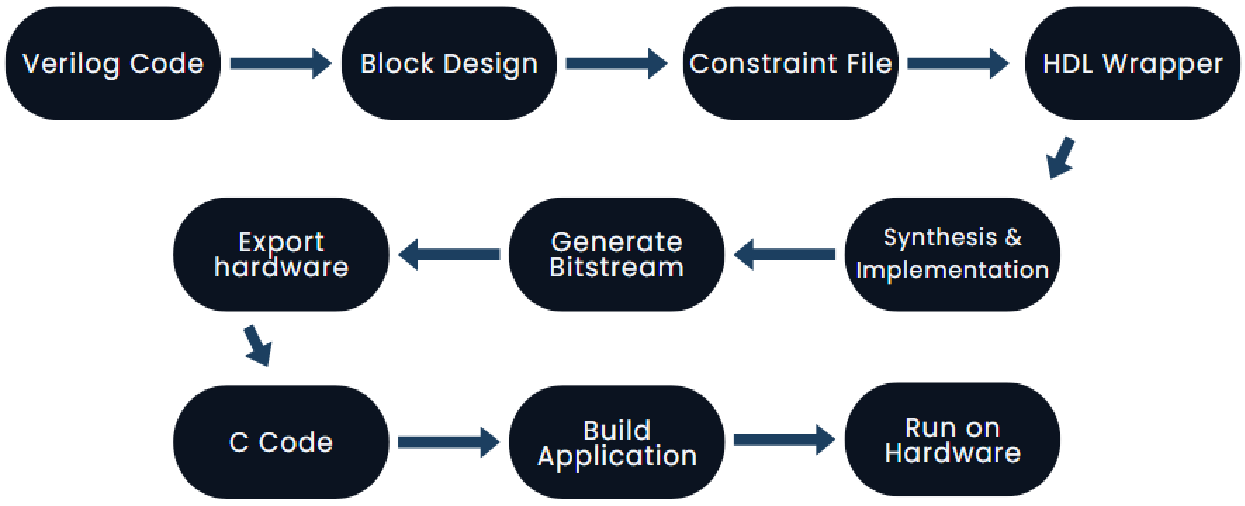


Figure 8.5 problem 2 design flow

8.2 Vivado Block design

Figure 8.2 shows the vivado block diagram of Problem 2. It shows the functionality of different IPs used in this project and their relationship with each other.

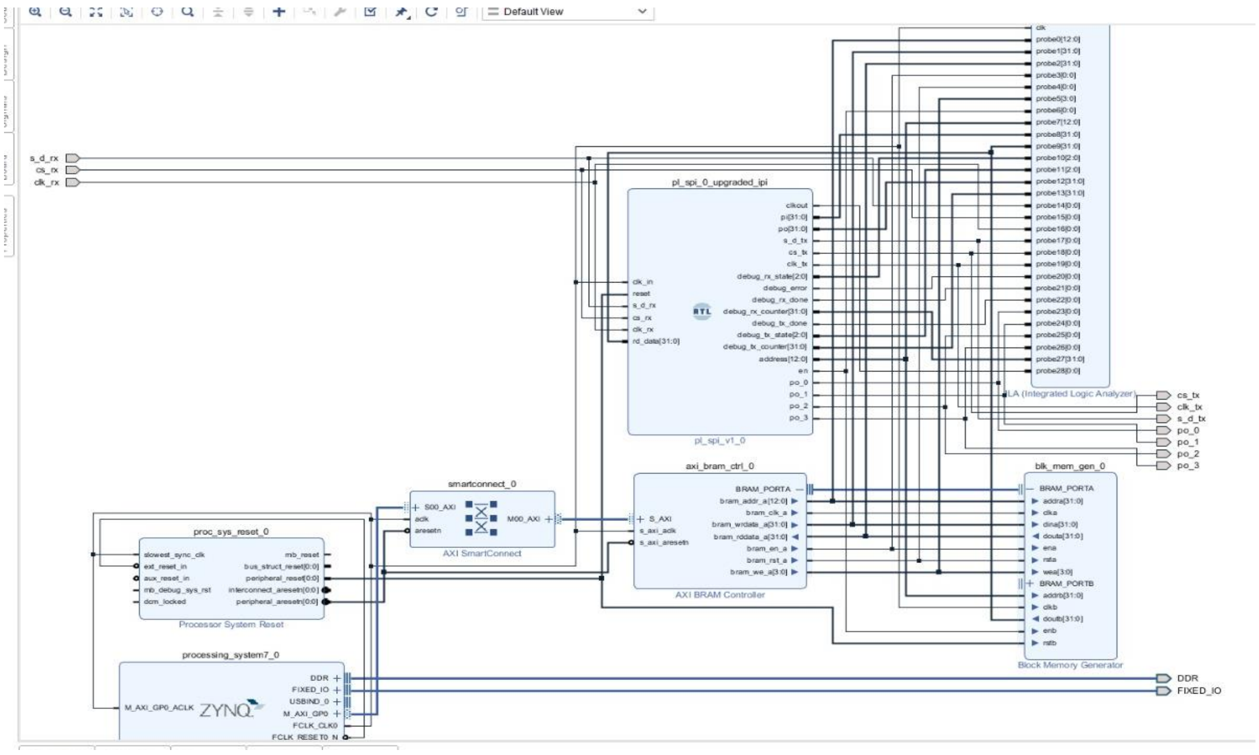


Figure 8.6 Problem 2 Block design

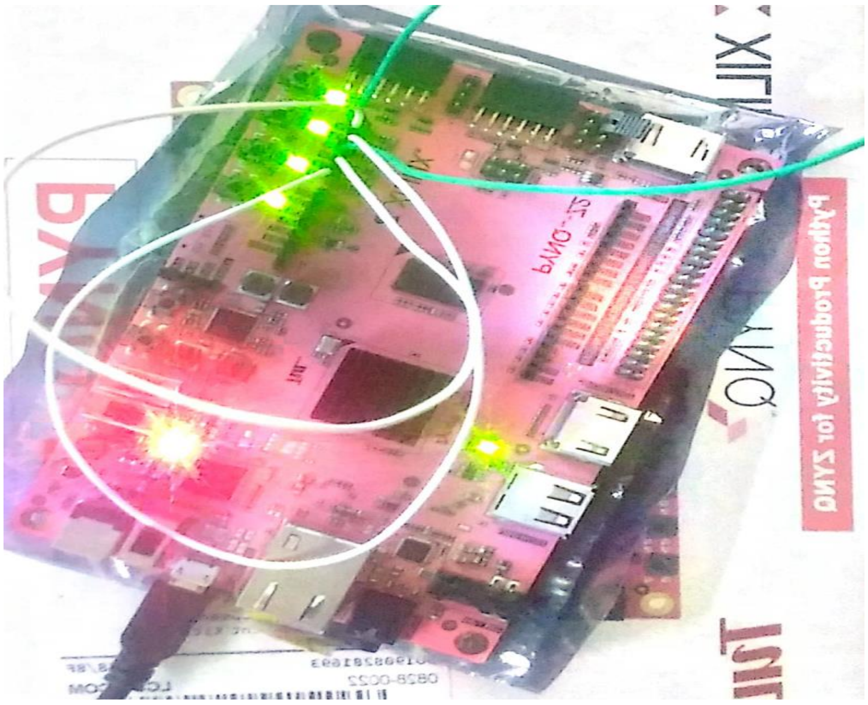
8.3 Hardware setup

Our project setup includes a PYNQ device, a USB cable to connect hardware to PC, and 3 male-to-male jumper wires.

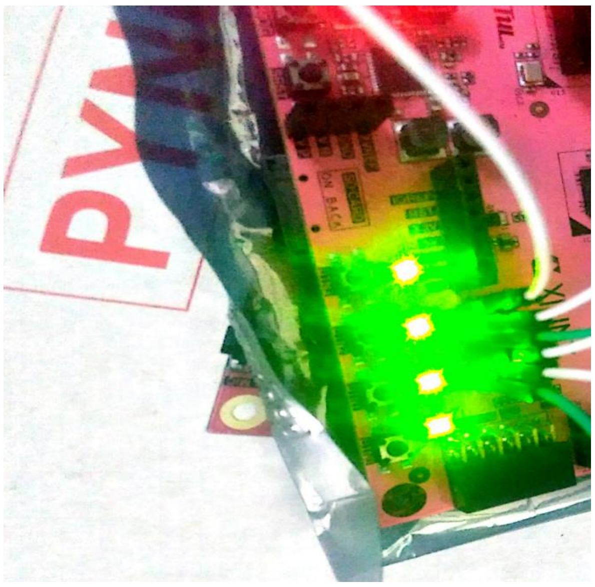
1 For providing clock to SPI receiver from SPI transmitter.

3 For providing Chip Select from SPI receiver to SPI transmitter.

4 For transmitting serial data from transmitter to receiver.



(a)



(b)

Figure 8.7 hardware setup

8.4 Program PS

To program PS, create a new application project in Xilinx select exported hardware from vivado and write C code in empty hello world file. Build your application and run it on hardware.

In this problem our Processing system is continuously asking to choose between write or read if we want to read data user need to press 0 and PS will read data from BRAM base address but if we want to write data user need to press 1 and write data at BRAM base address.

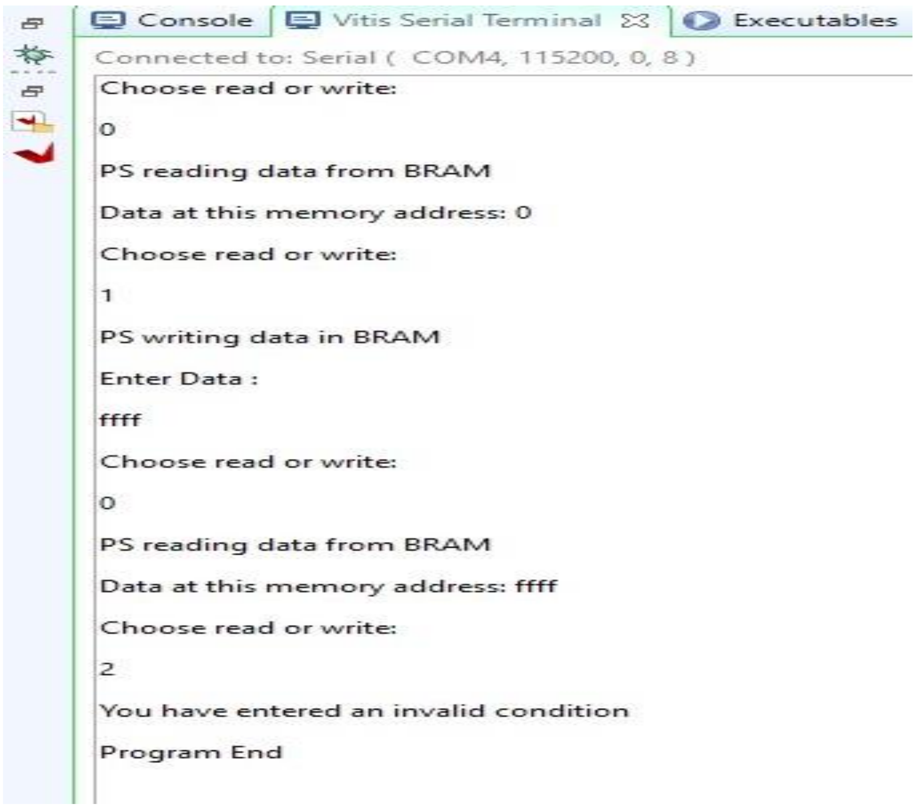
For code refer to appendix B.5

Chapter 9

Problem 2 Results and Discussion

9.1 PS is reading or writing Data.

Figure 9.1 shows the reading or writing of data from a certain memory address in BRAM by processing system.

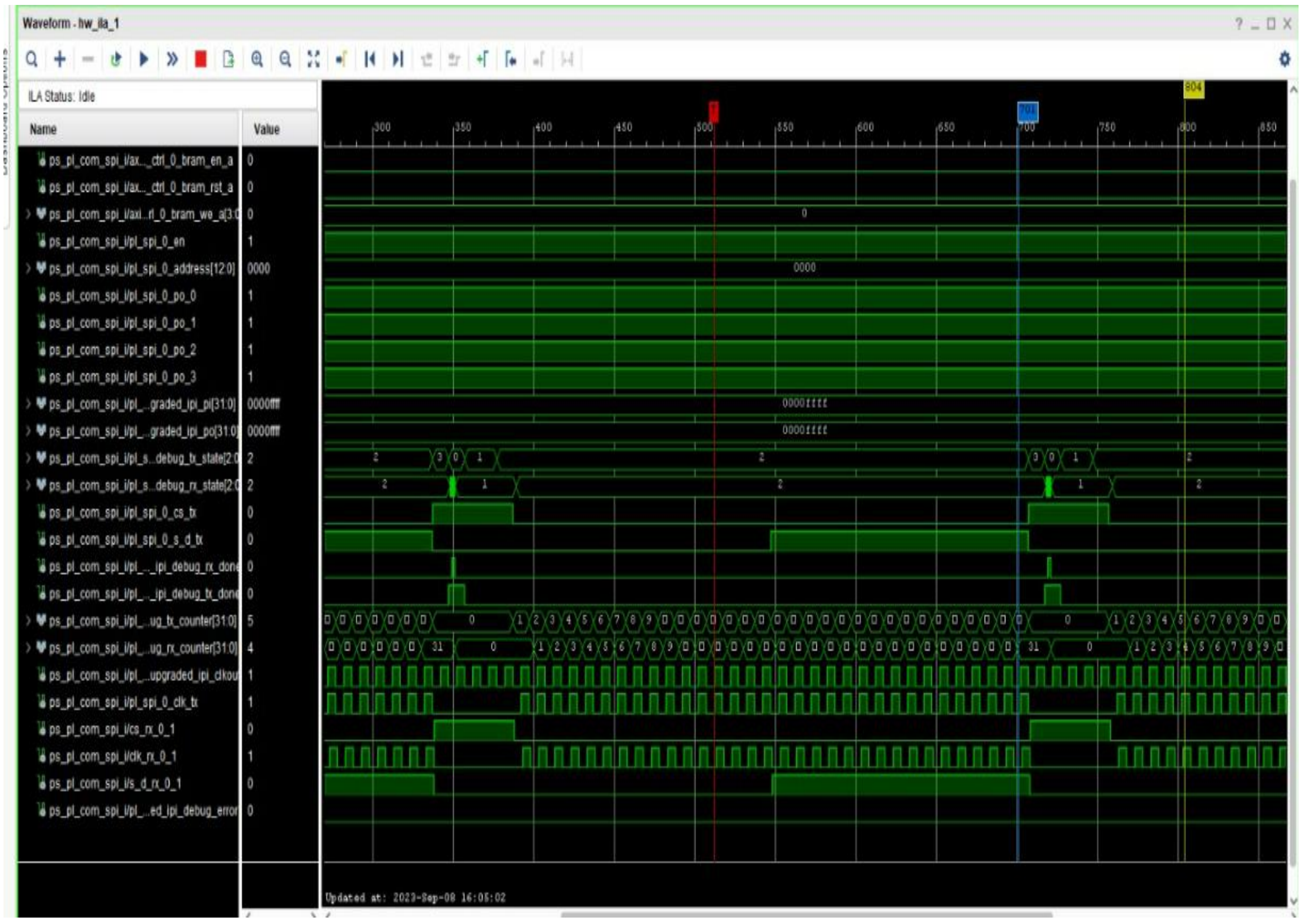


```
Connected to: Serial ( COM4, 115200, 0, 8 )
Choose read or write:
0
PS reading data from BRAM
Data at this memory address: 0
Choose read or write:
1
PS writing data in BRAM
Enter Data :
ffff
Choose read or write:
0
PS reading data from BRAM
Data at this memory address: ffff
Choose read or write:
2
You have entered an invalid condition
Program End
```

Figure 9.1 PS reading & writing

9.2.1 PL is reading Data.

Figure 9.2.1 shows PL reading data from that memory address which is written by PS in BRAM and Performing SPI transmission on that data.


Waveform - hw_ila_1

ILA Status: Idle

Name	Value
ps_pl_com_spi_i/ax..._ctrl_0_bram_en_a	0
ps_pl_com_spi_i/ax..._ctrl_0_bram_rst_a	0
ps_pl_com_spi_i/axi...rl_0_bram_we_a[3:0	0
ps_pl_com_spi_i/pl_spi_0_en	1
ps_pl_com_spi_i/pl_spi_0_address[12:0]	0000
ps_pl_com_spi_i/pl_spi_0_po_0	1
ps_pl_com_spi_i/pl_spi_0_po_1	1
ps_pl_com_spi_i/pl_spi_0_po_2	1
ps_pl_com_spi_i/pl_spi_0_po_3	1
ps_pl_com_spi_i/pl_...graded_ipi_pi[31:0]	0000ffff
ps_pl_com_spi_i/pl_...graded_ipi_po[31:0]	0000ffff
ps_pl_com_spi_i/pl_s...debug_tx_state[2:0	2
ps_pl_com_spi_i/pl_s...debug_rx_state[2:0	2
ps_pl_com_spi_i/pl_spi_0_cs_tx	0
ps_pl_com_spi_i/pl_spi_0_s_d_tx	0
ps_pl_com_spi_i/pl_..._ipi_debug_rx_done	0
ps_pl_com_spi_i/pl_..._ipi_debug_tx_done	0
ps_pl_com_spi_i/pl_...ug_tx_counter[31:0]	5
ps_pl_com_spi_i/pl_...ug_rx_counter[31:0]	4
ps_pl_com_spi_i/pl_...upgraded_ipi_clkout	1
ps_pl_com_spi_i/pl_spi_0_clk_tx	1
ps_pl_com_spi_i/cs_rx_0_1	0
ps_pl_com_spi_i/clk_rx_0_1	1
ps_pl_com_spi_i/s_d_rx_0_1	0
ps_pl_com_spi_i/pl_...ed_ipi_debug_error	0

Updated at: 2023-Sep-08 16:05:02

Figure 9.2 PL reading & writing.

9.2.2 PYNQ LEDs

Figure 9.2.2 shows 4 bits of that data which is written by PS in BRAM on PYNQ LEDs.

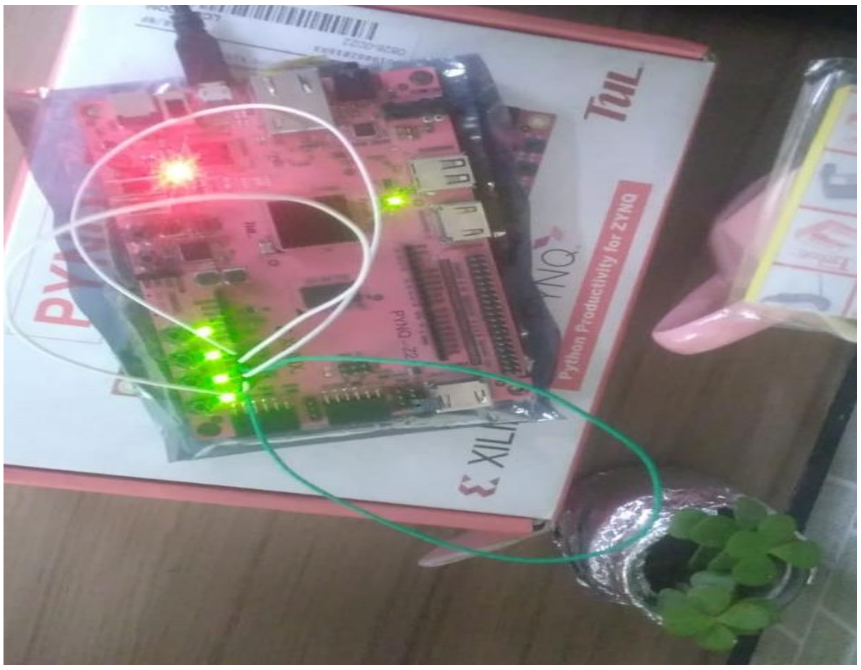


Figure 9.3 output on LEDs

Chapter 10

Problem 2 Resource Utilization Report

```
Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------
| Tool Version: Vivado v.2019.2 (win64) Build 2708876 Wed Nov 6 21:40:23 MST 2019
| Date        : Fri Sep  8 14:58:19 2023
| Host        : Areeba running 64-bit major release  (build 9200)
| Command      : report utilization -file
ps_pl_com_spi_pl_spi_0_upgraded_ipi_0_utilization_synth.rpt -pb
ps_pl_com_spi_pl_spi_0_upgraded_ipi_0_utilization_synth.pb
| Design       : ps_pl_com_spi_pl_spi_0_upgraded_ipi_0
| Device       : 7z020clg400-1
| Design State : Synthesized
------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------
```

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	70	0	53200	0.13
LUT as Logic	70	0	53200	0.13
LUT as Memory	0	0	17400	0.00

Slice Registers	152	0	106400	0.14
Register as Flip Flop	152	0	106400	0.14
Register as Latch	0	0	106400	0.00
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	-	-	-
0	-	-	Set
0	-	-	Reset
0	-	Set	-
0	-	Reset	-
0	Yes	-	-
0	Yes	-	Set
0	Yes	-	Reset
1	Yes	Set	-
151	Yes	Reset	-

2. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	140	0.00
RAMB36/FIFO*	0	0	140	0.00
RAMB18	0	0	280	0.00

3. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	220	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	0	0	125	0.00
Bonded IPADs	0	0	2	0.00

Bonded IOPADs	0	0	130	0.00
PHY_CONTROL	0	0	4	0.00
PHASER_REF	0	0	4	0.00
OUT_FIFO	0	0	16	0.00
IN_FIFO	0	0	16	0.00
IDELAYCTRL	0	0	4	0.00
IBUFDS	0	0	121	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	200	0.00
ILOGIC	0	0	125	0.00
OLOGIC	0	0	125	0.00

5. Clocking

Site Type	Use d	Fixe d	Availabl e	Util %
BUFGCTRL	0	0	32	0.00
BUFIO	0	0	16	0.00
MMCME2_AD V	0	0	4	0.00
PLLE2_ADV	0	0	4	0.00
BUFMRCE	0	0	8	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	16	0.00

6. Specific Feature

Site Type	Use d	Fixe d	Availabl e	Util %
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE 2	0	0	1	0.00
ICAPE2	0	0	2	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	151	Flop & Latch
LUT4	43	LUT
LUT6	12	LUT
LUT5	8	LUT
LUT2	8	LUT
CARRY4	8	
LUT3	7	LUT
LUT1	1	LUT
FDSE	1	Flop & Latch

8. Black Boxes

Ref Name	Used

9. Instantiated Netlists

Ref Name	Used

Conclusion

In this report we have discussed Processing system and Programmable logic. Their communication using BRAM. We have learned about BRAM.

We have discussed ZYNQ SOC, its features and PYNQ board and its functionality. We learned SPI communication.

We have also discussed AXI protocols. We dealt with two problems one is PS and PL communication using VIO and second is PS and PL communication and further process that data through SPI which has been written by PS.

Chapter 8

Appendix

Appendix A:

Appendix B:

Chapter 9

References

https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

https://www.elprocus.com/piso-shift-register/

https://techmasterplus.com/programs/verilog/verilog-shiftregister.php

https://www.allaboutcircuits.com/textbook/digital/chpt-12/serial-in-parallel-out-shift-register/

https://www.edaboard.com/threads/clock-divider-in-verilog.303639/

https://www.realdigital.org/doc/9c21eab4a0f85c50486858a87380d1f6

https://www.design-reuse.com/articles/52504/serial-peripheral-interface-spi.html

https://www.electronicshub.org/basics-serial-peripheral-interface-spi/

https://fastbitlab.com/spi-cpol-cpha-discussion/

https://www.javatpoint.com/spi-protocol

https://nandland.com/lesson-15-what-is-a-block-ram-bram/

https://docs.xilinx.com/v/u/en-US/pg058-blk-mem-gen

https://www.xilinx.com/support/documents/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf

https://xilinx.eetrend.com/files-eetrend-xilinx/download/201510/9366-20789-picozed70107020userguidev14.pdf

https://www.mouser.com/datasheet/2/903/ds190-Zynq-7000-Overview-1595492.pdf

https://www.xilinx.com/support/university/xup-boards/XUPPYNQ-Z2.html#tools

http://www.pynq.io/board.html

https://docs.xilinx.com/r/en-US/am011-versal-acap-trm/PL-Memory-Building-Blocks