M. Tech (Signal Processing and Digital Design) MAHESH SAI CHAGANTI 2021

Functional Coverage and Assertion Based Verification of Communication Protocol Using System Verilog

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTER OF TECHNOLOGY
IN
SIGNAL PROCESSING AND DIGITAL DESIGN

Submitted by:

Mahesh Sai Chaganti

2K19/SPD/09

Under the supervision of

Mr. Rajesh Birok (Associate Professor)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

2021

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, **Mahesh Sai Chaganti, Roll No. 2K19/SPD/09** student of M.Tech. (Signal Processing and Digital Design), hereby declare that the project Dissertation titled "**FUNCTIONAL COVERAGE AND ASSERTION BASED VERIFICATION OF COMMUNICATION PROTOCOL USING SYSTEM VERILOG**" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associate ship, Fellowship or other similar title or recognition.

[illegible]

Place: Delhi **Mahesh Sai Chaganti**

Date: 17-08-2021

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the M. Tech.[SPDD], Major Project-II [SPD602] Report titled "**FUNCTIONAL COVERAGE AND ASSERTION BASED VEIFICATION OF COMMUNICATION PROTOCOL USING SYSTEM VERILOG**" which is submitted by **SH. MAHESH SAI CHAGANTI, [ROLL NO: 2K19/SPD/09]** of Electronics and Communication Engineering Department, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the dissertation work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

[illegible]

Place: Delhi

Date: 22-08-2021

(RAJESH BIROK)

Supervisor

&

Associate Professor

Department of ECE, DTU

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

A successful project can never be prepared by the efforts of the person to whom the project is assigned, but it also demands the help and guardianship of people who helped in completion of the project. With profound sense of gratitude, I thank MR. Rajesh Birok(Associate Professor) my Project Supervisor, for his encouragement, support, patience and his guidance in this project work. I take immense delight in extending my acknowledgement to my family and friends who have helped me throughout this project work.

Date: 17-08-2021

Mahesh Sai Chaganti
M.Tech. (SPDD)
Roll No: 2K19/SPD/09

CONTENTS

List of Figures

List of Tables

ABSTRACT

ABSTRACT

There are different types of protocols formulated for specific needs in the Very Large Scale Integrated circuit design. Some of those include On-chip Communication protocols, Interface protocols, Memory management protocols, I/O protocols. A protocol helps devices connect to each other and acts a mediator which carries data, addresses and other information required for the communication. There are different types of protocols which define a set of rules for the communication to happen. Each protocol is uniquely defined for the needs. Generally a protocol needs to be functionally verified in the design stage itself. Suppose at a later stage a bug is encountered it takes huge amount of money, time and work to be put in to fix that. Hence it is recommended to fix any errors or bugs in the early stages. These days Verification engineers are at high demand for this reason. Generally for a design with more complexity, the verification environment is developed using System Verilog. As the title of project suggests, the Functional Verification and Assertion based verification is performed for a communication protocol using System Verilog. The project aims on creating a testbench environment using the most common Hardware Verification Language(HVL).

The design involves an interfacing block which can be employed in between an APB Master and an SPI Slave. So it acts fairly as both APB slave and SPI Master. Advanced peripheral bus(APB) is a low power peripheral protocol under Advanced Microcontroller Bus Architecture(AMBA) bus. AMBA is an enrolled brand name of ARM Ltd. Serial Peripheral Interface(SPI) is a full duplex serial communication protocol developed by Motorola. APB is in a way considered to be a parallel bus. It is capable of upto 32 bit wide data transfer between the master and the slave.

In the project, the focus is on the

- Design of the APB slave
- Deisgn of SPI Master
- Creating a verification environment to verify if the read and write operations are done successfully without any mismatches.

Since AMBA is a very vast protocol and has many components like AHB, AXI. The APB bridge here acts as a communication bridge between the AXI and the APB bus. Here the APB bus is designed to communicate with the peripherals.

AMBA is generally considered as the basic interconnect specification. Since it has the bus architecture, it enables the assembly and organization of the functional blocks in an SoC design. AMBA helps in even multiprocessor designs where many components and peripherals are involved. These days, every other ASIC or SoC is in a way incorporating AMBA bus architecture. On a variety of processors, which are a part of cellphones and gadgets use AMBA protocol. So, APB interface is picked up as an input.

One of the most used serial communication protocols is Serial Peripheral Interface. It is used for communication with the external peripherals serially. It acts as one of the high speed and highly reliable seral data stream supporting protocol. So the design this work emphasises, is about connecting these two protocols. The input comes from an SoC device which employs a high end communication and the output obtained will be a serial data stream driving a peripheral device like a Flash memory or a display driver.

CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

The communication between electronic devices has been a challenging task over the years. So protocols have been devised by different companies for different levels. Each protocol has a set of rules by which the data transfer can be done between the devices. Some protocols can handle large amount of data which have more hardware complexities, some are small enough to handle minor data transfers. Some of the protocols include System-On-Chip communication protocols, Embedded system Interface Protocols, Memory Management Protocols, I/O protocols.

1.1 On-Chip Communication Protocols:

Such a protocol is used to define the inter connect specification inside an SoC. It defines the management of connections between the functional blocks inside the chip. SoC is an Integrated circuit chip which requires a busing system to connect to various components it is comprised of. Usually SoC consists of one or more microprocessors, memory, peripherals, and other special logic but not limited to these. So a shared bus is required for those components to connect to each other. Many SoC designs are implemented by the bus architectures defined by the leading manufacturers like IBM, ARM[1]. Those are listed as below:

1. CoreConnect by IBM
2. AMBA designed by ARM
3. SiliconBackPlane from Sonics
4. Avalon made by Altera
5. Wishbone by OpenCores
6. STBus by STMicroElectronics
7. Open Core Protocol(OCP) made by Accellera

Some of the architectures are compared by researchers [3][4].However AMBA specification is the most popular and most used among the SoC bus specifications for high performance devices including modern day Mobile devices.

1.2 Embedded System Protocol:

A set of rules that define a way to connect two devices to each other is called a protocol. Embedded system is group of components like microprocessor, memory, and external peripheral devices which has a specific function.

The two types of protocols used in the embedded designs are:

1. Protocol Communication between two systems
2. Protocol Communication inside the system

1.2.1 Protocol Communication between two systems:

This type of protocol is used for the data transfer between a CPU and some micro controller kit. It is often referred as Inter system Protocol.

This type of communication includes the following protocols:

1.2.1.1 Universal Asynchronous Receiver Transmitter Protocol:

The protocol serves as one of the serial protocols for the data transfer. Therese mainly two signals each for transmission(Tx) and reception(Rx). It sends and receives the data stream serially without any clock which makes it asynchronous. It is only half duplex communication which means only one data line is active at a time. UART needs a start bit and a stop as a validation signal to avoid data errors. It’s data format generally used is 8 bit data stream. Usually microcontroller kits have UART signals on board.

1.2.1.2 Universal Synchronous Asynchronous receiver Transmitter Protocol :

This protocol serves both as an Asynchronous and synchronous mode. It facilitates the use of clock for the data transfer. In asynchronous mode the data is sent indefinitely without a particular data transfer rate. Unlike that Synchronous mode helps to send the data with respect to clock. Even this mode has two signals each for transmission and reception. In addition to the data clock pulses are also sent in this protocol when used in synchronous mode. It supports full duplex serial communication which allows the data transfer simultaneously on both the lines..

1.2.1.3 Universal Serial Bus Protocol:

It is again a 2-wire serial communication protocol. The data signals are known as D+ and D-. This is mainly used for the system to communicate with its peripherals. So here the host can transmit or receive data to or from the peripherals without any request on the host system. But it needs a driver software according to the specification of the system and the peripheral. Almost every peripheral like mouse and keyboard are connected to the present day PCs using the USB protocol. There are different speed with which a USB can operate till 400mbps.

1.2.2 Protocol Communication inside the system Protocol:

Such a protocol is used for the data transfer between the two devices which are accommodated on the same circuit board. The circuit might become complex because the protocol is also employed inside the main circuit. But such a design is cost effective and is secure to access the data since no data is going out.

Such protocols include the following:

1.2.2.1 Inter Integrated Circuit Protocol:

The Inter Integrated circuit is often referred as I2C protocol. It has two wires which stand for SDA(serial data) and SCLK(serial clock). Since it is a 2-wire serial communication protocol only, one each for data and clock. It has a data word format which has a start and stop bits. It can support 8 bit data transfer but in a half-duplex mode. Since it has only one data line it can only send data to or receive data from a slave at a time. Each slave has a unique address. The start and stop bits are the deciding factors the serial data transfer. The Master first sends the target slave address on the address after the start bit and when the slave address matches it responds with an acknowledgement signal. After it is received the master starts sending the data serially. Even after the data transfer slave sends an ack signal which indicates the master to issue a stop bit. I2C is used in embedded systems where there is a need for a microcontroller to communicate with peripheral devices serially[17].

1.2.2.2 Serial Peripheral Interface Protocol:

The SPI is the one of the most commonly used serial protocols developed by Motorola . There are 4 signals under SPI called clk, mosi, miso, cs making it a 4-wire protocol [6]. It is full duplex communication protocol which is described in detail in the coming chapters. The SPI can operate in 3-pin mode which is even without chip select pin accomodating only one slave. The data rates and clock frequencies can be configured according to the needs.

1.2.2.3 Controller Area Network Protocol:

It can be abbreviated as CAN. This is also a serial communication protocol developed by Robert Bosch. It consists of two signals CAN high and CAN low(H+ and H-). This protocol often finds use in the vehicular applications where we need airbag, antilock braking system.

1.3 Interface Protocols:

The protocols designed to act as interface between devices may be between multi device box or a single device box. These are commonly used as computer interface bus protocols where data between devices, Hard Disk Drives, RAM can be achieved. Some of the interface protocols defined are:

1. SATA(Serial Advanced Technology Attachment)
2. PCIe(Peripheral Component Interconnect Express)

PCIe is the most common Interface used in motherboard architecture through which HDD, SSD, Ethernet, WiFi, Graphic Cards, Host Adapters connections happen.

1.4 Memory Management Protocols:

A bus protocol which helps in communication from an IC such as microprocessor to an external memory on another circuit. The external main memory interface protocols are:

1. DDR
2. DDR2
3. DDR3
4. DDR4 etc.

DDR refers to Double Data Rate which means it is capable of two data reads and two data writes per clock cycles. Among these DDR4 has more speeds compared to DDR3 and so on. So each new version replaces the older memory interface protocols. Some of the memory interface protocols which are used for low power devices are:

1. LPDDR
2. Wide I/O
3. Memory MCP

CHAPTER 2

LITERATURE REVIEW

CHAPTER-2

Literature Review

2.1. Literature Survey on communication protocol designs

Communication protocol designs have evolved in a fast pace in the recent years. When it comes to On-Chip communications protocols many companies have left their footsteps in this field like AMBA developed by ARM [1] is by far the most popular communication protocol. After ARM becoming a giant in this field, companies like IBM, Altera, STMicro, Accellera have developed SoC communication through their interconnect structures which have been briefly discussed in [3]. Jovanovic et al. in [4] discussed about the three interconnect specifications. A. Shrivastav et al. took up the comparative study of various AMBA releases over the years in [5]. Each year ARM released newer and advanced versions of high performance system buses which added to the present day performance of the SoC. Under the AMBA specification , a low bandwidth interface is present which is used to drive the peripherals. The APB version used in this work is presented in [10].

Leens et.al. discussed the overview and a brief comparison study of most used serial communication protocols which are I2C and SPI in [6]. These are used as a connections from the microcontrollers to the external peripherals. The I2C and SPI need 2 wires and 4 wires respectively. Basic SPI was first developed by Motorola, which is then carried forward by Freescale semiconductor [7] which is now NXP Semiconductors. The thesis work is related to the guide published by the Texas Instruments [8]. Many companies came up with different specifications formulated for their own controllers. So SPI has multiple specifications unlike its 2-wire competitor I2C which is given by firstly developed by Philips but then developed later by NXP [9]. These are the most common peripheral interfaces used for the serial communication with microcontrollers.

2.2. Literature Survey on designs involving physical implementation

Anand N et al. discusses the design approaches which explore the flexibility of the transmission modes of SPI and capability of driving multiple slaves with chip select lines. They verified the model using an FPGA where the results showed a very high clock frequency in [16]. Oudjida et al. discussed physical implementation aspects of the two serial communication protocols as a comparative study in [17]. A. K. Oudjida in [18] proposed general purpose Master/Slave Ips which are mapped on to FPGA for verification. The model proposed runs on OPB bus(On-chip Peripheral Bus) which is an interconnect specification developed by IBM. M. B. Aykenar et al. proposed a less logic resourced SPI Master design which helps in making cost effective FPGAs in [19]. Ge Zhiwei et al. proposed an image processing technique based on APB bus with CMOS sensor in [35]. It shows that can enhance the image quality using the auto white balance method.

N. b. Mohd Noor et al. proposed a new concept of utilizing the SPI design and achieving a different data transfer mode in contrast to continuous data stream in [20]. The design can incorporate burst, continuous, interleave data streams. S. Saha in [21] formulated a self-testing capability in SPI interface which eliminates the need of two devices to verify the communication. This method which mainly reduces the testing cost is run over an FPGA Recently, Internet of Things has seen a boom in the electronics industry. L. Huihui in [34] designed a SPI master controller driver which is based on VxWorks, a Real time operating system developed by WindRiver. VxWorks is an RTOS developed for IoT applications. R. R. Pahlevi et al. in [22] designed a faster SPI slave for scalable IoT platform which needed change in control register. P. S. Mutha in [23] proposed a design where the reconfiguration of FPGA can be done through UART lines and an SPI Flash. Since FPGAs are reusable they are reconfigured using SPI flash which in turn reduces time.

2.3. Literature survey on IP Core designs involving two protocols

Since the communication on a chip happens with a high performance interconnect specification, there needs to be a low bandwidth peripheral driving protocol. APB happens to be the low bandwidth bus that is used to communicate with the peripherals. APB being a parallel sometimes cannot be integrated with a serial data slave device. So, to eliminate this problem serial communication protocols should be accompanied. One such protocol is Serial Peripheral Interface which is serial full duplex 4-wire communication protocol.

One more popular 2-wire protocol used for this purpose is I2C. Implementing a bridge between I2C and APB bus is proposed by J. Chhikara et al. in [11]. D. Trivedi et al. proposed a communication between two widely accepted protocols which are I2C and SPI. In [33] they proposed a design where SPI as 4-wire protocol is the sender and is received by a receiver which runs on I2C. Using the high speed of SPI and less hardware of I2C the design is proposed. L. Bacciarelli et al. proposed an architecture to interface the serial protocols for the APB bus in [12]. M. Hafeez et al. in [13] showed up an IP Core design which takes input as APB signals and giving out SPI signals. They designed a SPI Master controller which is controlled by the different registers to configure. Cadence developed an IP core for production into the market which acts as an interface between the APB bus and the SPI peripherals. It [14] involves a 32 bit APB parallel bus as an input which ultimately produces a serial full duplex output. J. Yang et al. proposed a configurable SPI interface for the AMBA APB bus in [15]. They gave a model which is convenient and flexible to use. The design works with a programmable transmission mode, data transfer rate, clock frequency, direction of bit transfer. Using a FPGA toolkit they verified the working with slave devices for communication. The high performance system bus ASB needs a bridge to communicate with the peripheral bus APB which is discussed in [10].

2.4. Literature Survey on Verification of protocol designs

If the verification is about reusability the testbench methodology called Universal Verification Methodology is used. Dipti Girdhar Shankar et al. developed a UVM module for verification of APB protocol in [31]. Even R.K Vaishnavi et al. developed a design of APB module and verified it through the reusable methodology in [32]. Jaideep Varier E.V. et al. discussed about the Verification of the IIC protocol using UVM in [24]. Here, the interface used is an APB interface. Z. Zhou et al. in [25] developed a verification environment for SPI with an APB driver where coverage, assertions were used with checker and scoreboard for reusability. Constrained random vectors are driven into the DUT for higher functional coverage.

In [26] Y P. Jain et al. formulated a design which acts as APB Slave and a verification environment using random values driven onto the design. P. Dwivedi et al. in [27] proposed an assertion and coverage based verification model for the APB bus. It covers all the internal transactions in the APB cycle using the constrained random checks and assertions. Y. Guo et al. in [28] formulated constrained random coverage along with self-check verification module with SPI interface using UVM. P. Gurha et al. in [29] modulated an SVA based approach to verify the AHB to APB bridge. B. Vineeth et al.

in [30] proposed a UVM based verification plan where coverage based verification is performed. At the same time scoreboards are included to check automatically if the functions are met correctly.

2.5. Dissertation Contribution

Following are the detailed contributions of this dissertation:

2.5.1 Brief about APB and SPI:

We first review the design specs developed by the major tech companies. Each company have its own interconnect specifications for On-chip communication. But, needless to say AMBA is the most used SoC protocols in the market today. Every mobile phone accommodates an SoC which uses high performance AMBA bus. Out of all the high performance system buses APB is the one bus which is used for the communication with the external peripheral devices.

It is one of the important aspects for a microcontroller to communicate with peripherals such as keypad, timer etc. When it comes to streaming data serially two most used protocols are I2C and SPI. Since SPI is the faster protocol and a full duplex communication supported one, it is chosen by most. Even though it has 4 wires it is used extensively till date. So these protocols are briefly discussed in the early chapters of this work.

2.5.2 Design specification involving APB Slave/SPI Master IP Core:

Here, the major design specification is proposed. In the hierarchy of AMBA buses, a system bus like AHB or ASB is mandatory for an SoC. When the data stream is to be transported inside a major bus like AXI, AHB, ASB is needed but when it comes to low bandwidth peripheral operations those higher end protocols are not needed. That’s where the use of Advanced Peripheral Bus comes in. Even to convert an AHB transaction into an APB transaction, a bridge is needed. As an output from the bridge we obtain APB bus which is distributed on to every peripheral device.

The APB bus is however a complex bus for a simple serial data driven peripheral. So that’s where an SPI bus finds a need. In devices like sensors, ADC, Flash memory, video game controllers SPI is used. So to communicate with these we need an SPI Master to stream the data outwards. The design proposed in this work acts both as an

APB Slave which takes input as APB bus and an SPI Master which drives SPI Slaves. So in a way this IP Core acts as a parallel to serial communication protocol where at the output end, we achieve a high speed, serial, full duplex communication protocol.

2.5.3 Functional Coverage and Assertion based Verification of the design:

This design is verified functionally using coverage metrics. Assertions are used to check if the condition is met, if not the assertion fails. The assertions are written using System Verilog Assertions(SVA). The coverage done in this thesis involves code coverage also which basically gives an idea about how much of our code is used and not wasted. This type of verification is growing popular because all the test cases are checked in the simulation stage itself which drastically reduces costs.

CHAPTER 3

AMBA

CHAPTER 3
AMBA

3.1 Background:

Advanced RISC Machine(ARM) Limited first introduced AMBA(Advanced Microcontroller Bus Architecture) in the year 1996. In the beginning, APB and ASB protocols were introduced. In 1999, as its second release the Advance High performance bus was added. In the coming years, they were succeeded by the third generation AMBA.

AMBA 3 contained Advanced extensible Interface which focused on higher interconnect performance and also the Advanced Trace Bus. AMBA 4 that was released recently in 2011 had AXI4 along with the coherency extensions. In the following years AMBA released Coherent Hub Interface with a redesigned layers and features that reduced overcrowding.

The SOC building not only depends on the components it contains, even the connections it possesses. AMBA helps interconnection of the components inside a System on chip.

The aim of the AMBA protocol is to:

1. It enables development of a processor unit with either single or multiple processors
2. It allows reuse of intellectual property cores and other macrocells
3. It encourages system design module-by-module where peripheral and system libraries can be reused.
4. Helps in reducing silicon while enabling high performance and less power consuming designs

3.2 AMBA protocol specifications:

The AMBA specification provides a SoC standard that enables designing high performance microcontrollers. Even though it was developed by ARM Limited, it enjoys wide participation throughout the industry which makes it easy for the designers to develop a processor or microcontroller.

The AMBA release facilitates a SoC standard that enables us in designing a high performance micro-controller. The three buses that are widely used in the bus protocol specification are:

3.2.1 Advanced High-performance Bus:

It is mainly designed for high performance and which involves high frequencies. It supports the effective connection of processors, memory blocks that are present on-chip and off-chip with all the low-power peripherals. Its main advantages include: high performance, pipelined process, multiple bus transactions, burst transfers.

3.2.2 Advanced System Bus:

It is the one stop alternate solution if high performance system bus modules are needed, but all the AHB features are not required. It also supports all other functionalities such as efficient connection of processors, on and off-chip memory and the peripherals. Its main advantages include: high performance, pipelined process, multiple bus transactions

3.2.3 Advanced Peripheral Bus:

The APB protocol is basically designed for the peripheral that consume low power. Over the years APB is improved so that it is comfortably used with devices consuming low power and those which involve reduced interface complexity. It is generally adaptable to any release of the system bus. Its main advantages include: less power consumption, quite simple interface, latched address, appropriate for a large number of peripherals.

3.3 A general purpose AMBA built microcontroller:

A microcontroller built on AMBA generally comprises of a high level system bus which is able to transact with the on-chip residing memory, external memory devices and direct memory access controllers. This bus provides a high-bandwidth interface between the components concerned within the most of the transfers. Conjointly situated on the AHB may be a bridge that connects to APB bus where all the peripheral devices are connected (see Fig. 3.1).

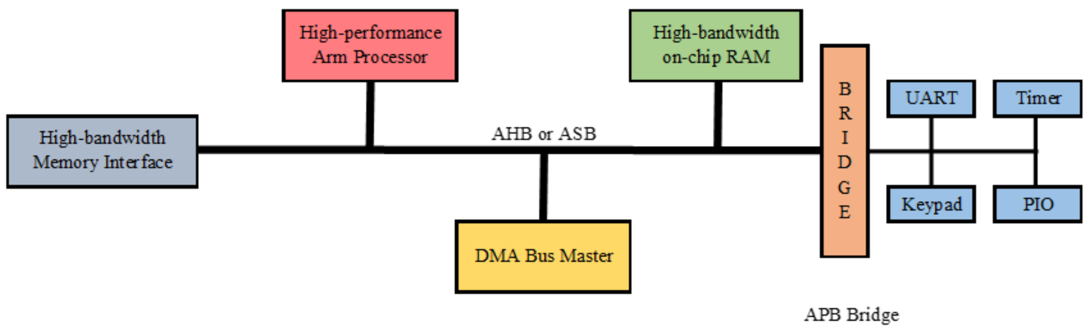

High-performance Arm Processor
High-bandwidth on-chip RAM
High-bandwidth Memory Interface
AHB or ASB
BRIDGE
UART
Timer
Keypad
PIO
DMA Bus Master
APB Bridge

Fig. 3.1 AMBA Functional Block Diagram

APB helps in providing the peripheral transactions as the minor bus whereas the high performance system bus(AMBA AHB and AMBA APB) acts as the main system. The peripherals under this generally:

- May have interfaces with memory map
- Has low-bandwidth interface
- They are under manageable level when coded accordingly

Generally the external memory interface is specific to the desired application. In this case we use SPI data bus as the peripheral interface. It supports a test access mode which allows the internal AMBA AHB, ASB and APB modules to be tested.

3.4 AMBA Release Versions:

The ARM till now released 5 versions of AMBA bus. In each version, it has added some new protocols or released a new edition of the same protocol. Highlights of each release is listed below:

AMBA v5 release involves the buses:

- AXI v5.0, AXI5Lite and ACE v5.0
- AHB5, AHBLite
- CHI interface

AMBA 4 release involves following main buses:

- ACE, ACELite, AXI4, AXI4Lite
- APB4 v2

AMBA 3.0 release involves following buses:

- AXI4
- AHBLite v1
- APB3 v1
- ATB v1.0

AMBA 2.0 release involves 3 buses:

- AHB2
- ASB2
- APB2

AMBA 1 release involves 2 buses:

- ASB
- APB

Advanced System Bus(ASB) is the one of the first generation buses developed under AMBA. ASB sits on top of the peripheral bus for the high performance system operations such as pipelining, burst transfers, and supporting multiple bus master modes. Bus masters could be a high performance processor, high bandwidth memory interface or even a DMA or DSP processor. An APB bridge or internal memory are the most common slaves. Any peripheral connected to the system might also serve as an ASB slave

ASB Master initialises the read and write operations providing the address and control information. The ASB arbiter takes care that only one bus master initiates the data transfers at a time. ASB decoder decodes the transfer addresses and selects the slave appropriately. ASB slave responds about the read and write to the master if it’s a success or failure.

Another High performance bus that is most used is Advanced High Performance Bus(AHB). The features required for high bandwidth and high performance operations include Wider data bus configurations(64 or 128 bits), Burst transfers, Split transactions among others. All other components are similar to the system bus. These higher performance buses need a bridge to implement the connection to peripheral bus like APB and should be done efficiently to integrate it to other existing designs.

CHAPTER 4
ADVANCED PERIPHERAL BUS

CHAPTER 4
ADVANCE PERIPHERAL BUS

4.1 Introduction:

APB as a part of AMBA has been far more power efficient and has been effective in terms of interfacing. The APB is a component of the AMBA buses and is far more effective for reduced power consumption and less interface complexness[2]. It is generally not regarded as a primary bus. But when accompanied by a high performance bus they both bind together in a very good way. It is known for providing a low power bus protocol to a performance oriented system bus. This protocol is mainly used when there is no use of high performance and the peripherals just need a low bandwidth operation. The APB specification specifies, signal change is all dependent on the rising clock edge.

The above changes made possible the below listed advantages:

- It is easy to attain a large frequency for process
- Output is regardless of the clock’s mark space ratio
- STA is simplified by the usage of one clock edge
- No special consideration are necessary for automatic insertion of the control
- Several ASIC libraries have an improved choice of rising edge registers

APB bus mainly contains a bridge that is responsible to convert the high performance transactions into the peripheral understandable language. The bridge enables latching of all types of signals, and also provides a deciphering mechanism that generates the slave select signals. Sometimes the main system bus might change for the SoC. In [10] he system bus involved was ASB. The authors managed to design a bridge which could convert the transactions into peripheral bus transaction. Generally for high performance systems, AHB is the higher level bus that would be involved. The same AHB-APB bridge is discusses in [36]. Similarly bridge between I2C and APB is designed because both need to share resources but the protocols are different. The AHB/ASB to APB bridge is shown in the Fig. 4.1. The signals on the right indicate the APB interface signals used to drive an APB Slave.

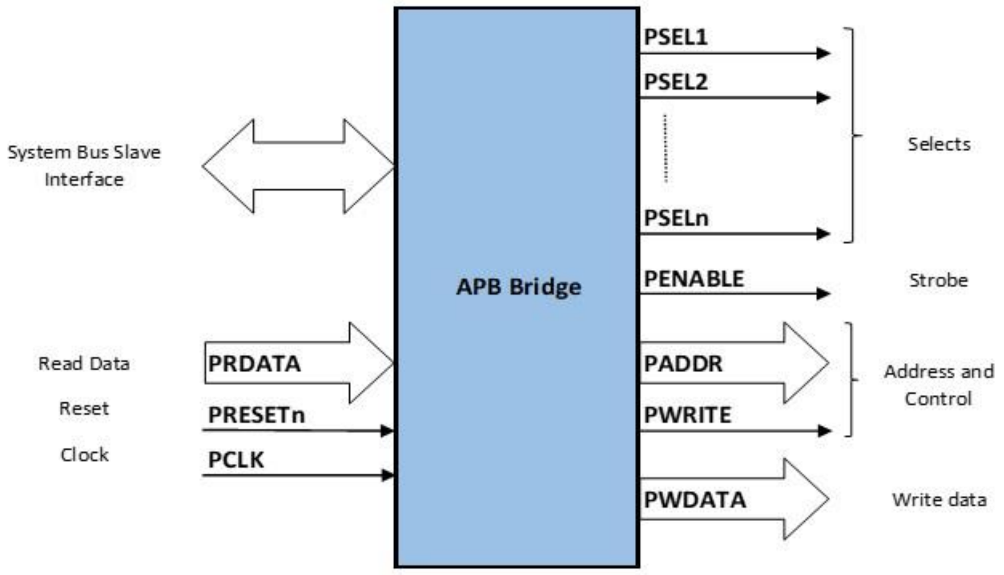


Fig. 4.1 APB Bridge between ASB and APB slave

All the other modules are typically slaves. A typical APB Slave is as shown in the Fig.4.2

The APB peripherals have the below listed features:

- The address and control can be valid throughout the access
- zero-power interface throughout non-peripheral bus activity (peripheral bus is static once not in use)
- Timing can be analyzed by means of decode with the unclocked interface.
- Write information valid for the total access (allowing glitch-free clear latch implementations).

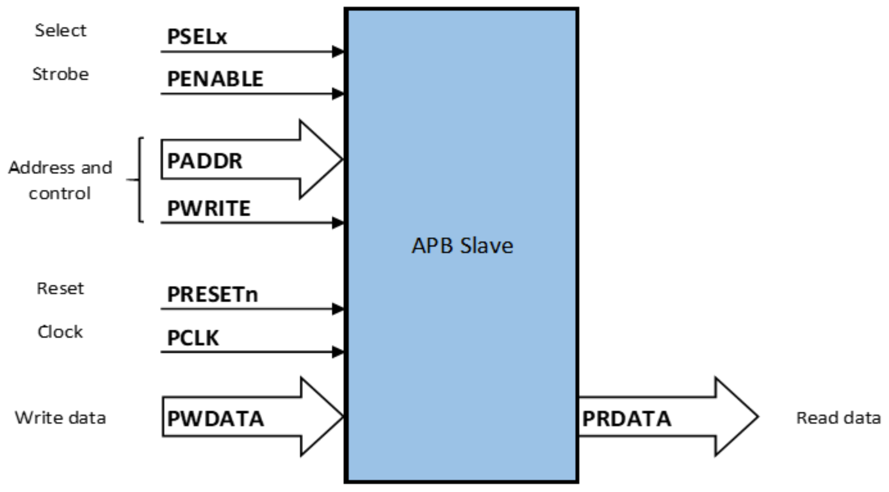


Fig. 4.2 APB slave

4.2 APB signal list:

All APB signals have the letter P as prefix. The below table shows the list of APB signals

AMBA APB signals

APB Bus Signal	Description
PCLK(Bus Clock)	The rising edge of this signal times all signals on the bus
PRESETn(APB reset)	The active low signal that is generally connected to system bus
PADDR[0:31]	Address bus which comes from the APB Interface
PSELx(APB Select)	Each bus slave has this and it indicates that particular slave is selected and needs a transfer
PENABLE(APB strobe)	This indicates 2nd cycle on the transfer.
PWRITE(APB transfer direction)	Low is read and high is write
PRDATA(APB read data bus)	Driven by the selected slave for a read operation which is 32 bit wide
PWDATA(APB write data bus)	Driven by APB bus and is 32 bit wide when pwrite is high

Table 4.1 APB Signal List

4.3 APB specifications: The table above represents the signals of an APB bus. Since APB is a parallel bus the address and data are 32 bit wide.

State Diagram:

APB state machine jumps to three states:

IDLE: This is the default state.

SETUP: When a transfer is needed, the bus moves into the SETUP state **PSELx**, is asserted. The bus remains for only one cycle then jumps to enable after a rising clock edge.

ENABLE: Here enable signal, **PENABLE** is asserted. All signals stay stable during the jump. This state only lasts for one cycle if no other transfers then returns to IDLE if there then jumps to SETUP state. Every signal can be subjected to a change during the jump to SETUP. The jump from the ACCESS state depends on the state of the PREADY signal.

If PREADY is held low by the peripheral the state remains unchanged. When that is held High the state jumps to SETUP if transfer is needed. If not, state jumps to IDLE state.

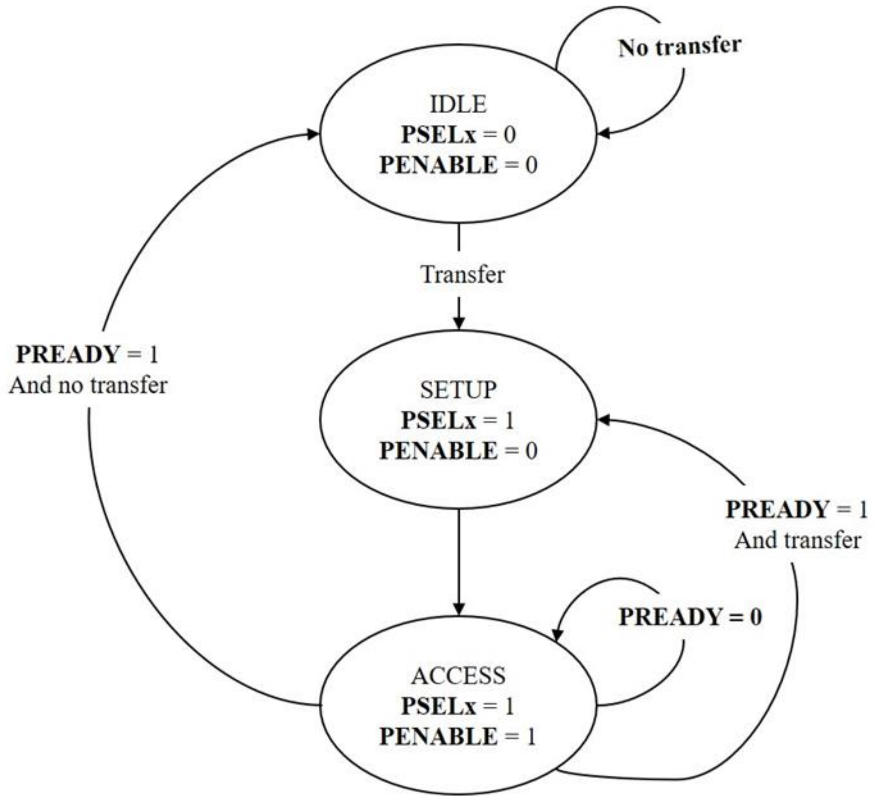


Fig. 4.3 APB State Diagram

There are 3 states in the state diagram of a APB transaction:

1. Setup
2. Write Enable
3. Read Enable

APB signals	**Setup**	**Read Enable**	**Write enable**
Psel	1	1	1
Penable	0	1	1
Pwrite	-	0	1

Table 4.2 Different States of APB

Whenever there is a positive edge of clock, the above values are checked for every cycle. According to these values, the state changes and the read, write operations are performed

CHAPTER 5

SPI PROTOCOL

CHAPTER 5
SPI PROTOCOL

5.1 Introduction:

Serial Peripheral Interface is abbreviated as SPI. It works as a high speed synchronous serial communication protocol[16] which lets you program a convenient bit transfer rate to be transmitted and received. Since it supports full duplex mode it can be simultaneously at a time[6]. The SPI is generally used with a micro controller device and a peripheral. Since it acts a high speed serial port it assumes use in applications like memory devices(FLASH), LCD display drivers, sensors and some converters(ADCs,DACs).

5.1.1 Features of SPI include:

1. Programmable master or slave mode
2. Chip select signal
3. Registers used to transmit data
4. Buffer register for reception
5. Registers to select Data Format
6. Capability to be driven by Interrupts

5.1.2 Basic configurable features of SPI:

1. The clock frequency of SPI
2. Number of pins(3-pin or 4-pin)
3. The data word length can also be modified with the shift direction.
4. 4 clock transmission modes
5. Any type of delay(b/w the transfers, setup, hold)
6. The setup and hold times of chip select signal
7. The chip select can also be held

The protocol supports change of number of pins. If it’s a bus master every pins is driven by itself. Additionally chip select pin is added to the list if it operates in a 4 pin mode. It means it can operate multiple slaves, but SPI supports only a single master. The Fig. 5.1 shows a functional block diagram of SPI protocol[8]. The four external pins or signals are:

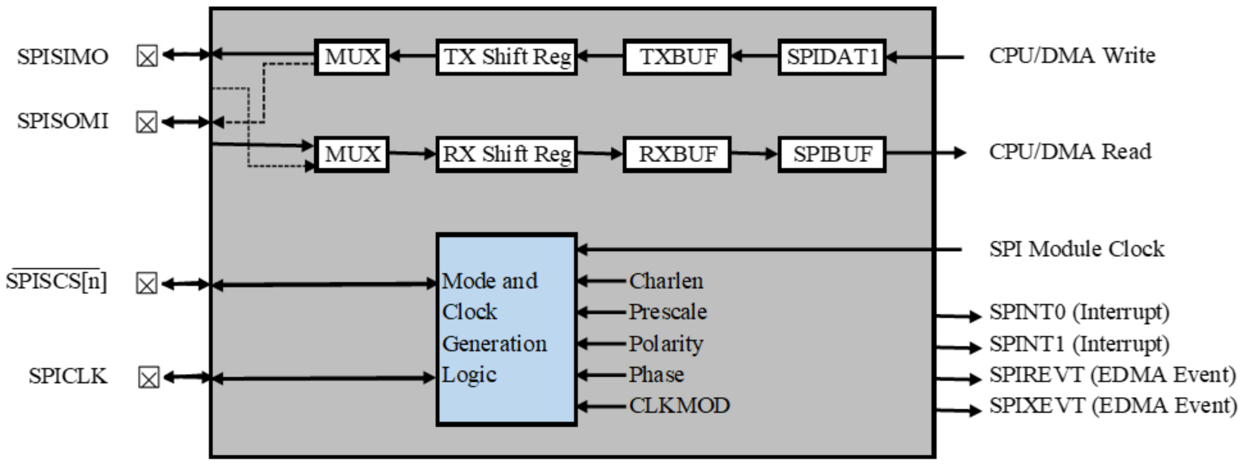


Fig. 5.1 Functional Block diagram of SPI protocol

5.2 Signal List:

MOSI: This pin is used to transmit data out of the SPI Master and receive data into SPI Slave.

MISO: This pin is used to transmit data out of the SPI Slave and receive data into the SPI Master.

SS: This pin is used to select the peripheral device(slave) when it is configured as Master and used to receive the signal when configured as Slave. This enables to drive multiple slaves by the same single SPI bus

SCK: This pin is used to generate the clock when it is configured as SPI Master according to which the data transfer happens. When the module is configured as Slave pin is used to receive the clock.

SPI supports reset mode when the RESET bit is set in the Global control register. SPI supports power conservation using a POWERDOWN mode. When set, the module enters into power saving mode which means all operations are inactive.

Clock: The clock is obtained from the module clock provided. The maximum achievable frequency is half of the module clock frequency. The clock frequency for the SPI is determined by the PRESCALE bit in the format registers(FORMATn).The clock is obtained from the module clock through the below formula:

Frequency of clock = (Module clock) /(PRESCALE + 1)

When the prescale is cleared, the clock frequency is set to the default value of half of module clock.

5.3 Pin modes:

5.3.1 3-pin mode:

This mode uses the clock and both the data pins for data transfer between the master and slave devices. The below Fig. 5.2 shows the mode with only 3 pins.

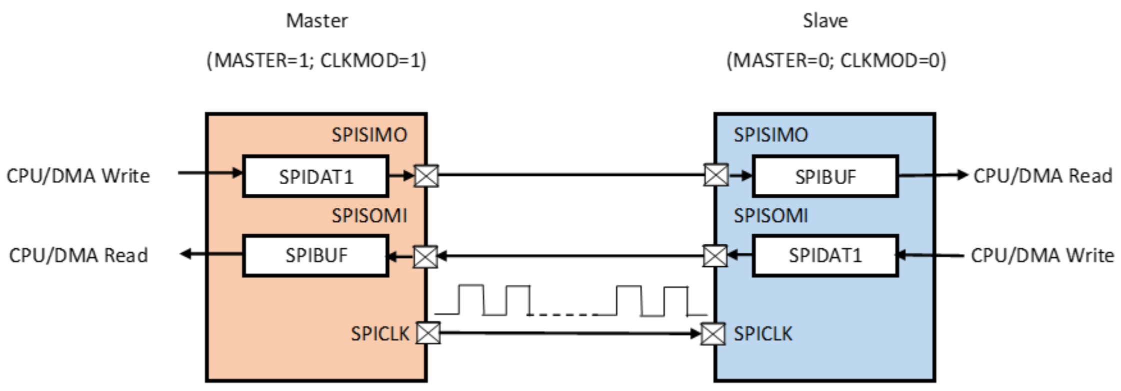


Fig. 5.2 3-pin mode

To run the SPI in 3-pin mode, the CLK, SOMI, and SIMO pins should be made as functional pins by setting the bits high in pin control register 0 (PC0).

To make SPI as master, the bits in global control register should be set high. The master is the device decides whether to generate clock and to initiate data transfer. The necessary bits in the Global Control register should be set high. In master generally the master in slave out pin is in high impedance state and output buffer of the other data pin is enabled.

The master mode SPI device takes in the data through the two data transmission registers which initiates the data transfer. A series of clock pulses help the data to be transferred simultaneously in and out of the master and slave devices. Either of the two data transmission registers can used on both master and slave.

5.3.2 4-pin mode:

This mode has an additional chip select pin along with the data and clock pins[14]. Fig. 5.3 shows the 4-pin mode of the SPI.

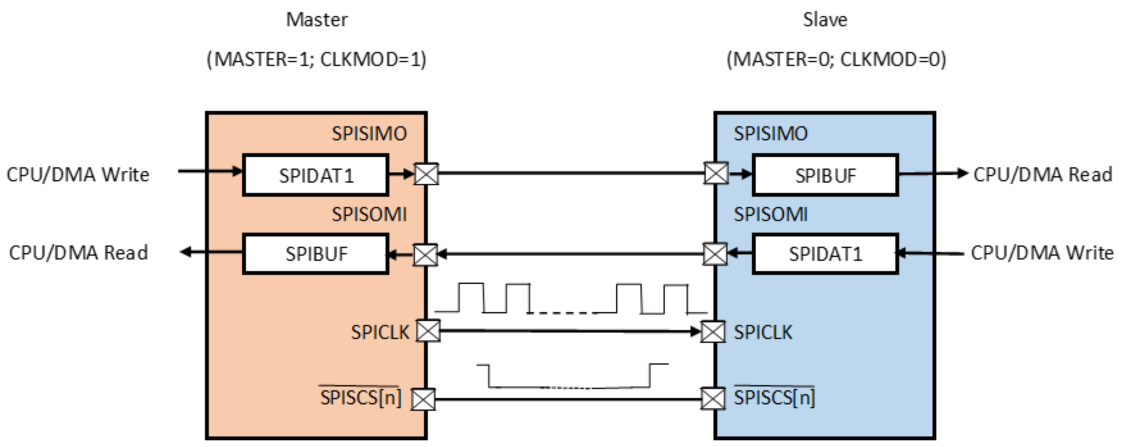


Fig. 5.3 4-pin mode

In addition to MOSI, MISO, CLK, the chip select pin(SPISCS[n]) is also configured as functional pin using the pin control register bits.

When SPI is configured as master the chip select pin works as output pin and is activated when a slave is selected. Only one slave chip select pin can be incorporated in the SPI. However, multiple general I/O pins are needed to support multiple slave chip selects.

The advantage of using the chip select as a functional pin is set the timing parameters using the delay register's SPIDELAY pin. To meet the slave timing requirements the such a delay can be used to automatically add the delays. Another advantage is to have the error detection capability.

5.4 Data formats:

The protocol offers an option to configure with format of the data used[15]. The data formats are configured by varying the bits in the data format registers(FORMATn). The following characteristics change in each data format:

- Character length: It is represented by CHARLEN field in the data format register. It can vary from 2 to 16 bits.
- Shift direction: The data transferred can be even configured for the direction. SHIFTDIR represents if the LSB or MSB is to be shifted first.
- Clock polarity: It can be configured by the CPOL bit.
- Clock phase: It is configured by the CPHA bit.

On each transaction, the data format is chosen. The data format can change for every transaction or can be configured once for every transaction that follows until it is changed.

Shift direction:

The protocol automatically right aligns the data even if its transmitted or received data.

Character Length:

The character length of the data can have values from 2 bits (2h) to 16 bits (10h). It should be programmed in the master mode. It is independently configured for each of the four data formats.

CPOL and CPHA:

The protocol gives a flexibility to choose between the options of four different clock modes. These 4 combinations are defined upon the clock phase and the clock polarity. These modes help to connect easily to the other serial peripheral devices.

CPOL	CPHA	Action
0	0	Output data is obtained at posedge of clk. Input data is latched on the negedge of clk
0	1	Output data is obtained one half clock prior to the first posedge of clk. Input data is latched on the negedge.
1	0	Output data is obtained on the negedge of the clk. Input data is latched on the posedge of the clk.
1	1	Output data is obtained one half clock cycle prior to the first negedge of clk. Input data is latched on the negedge

Table 5.1 Clock transmission modes

1. Clock Phase=0 which means clock without delay:
    - Data is outputted on the positive edge of clock
    - Input data is latched on the negative edge of the SPICLK
    - The clock starts with the data coming at the data register

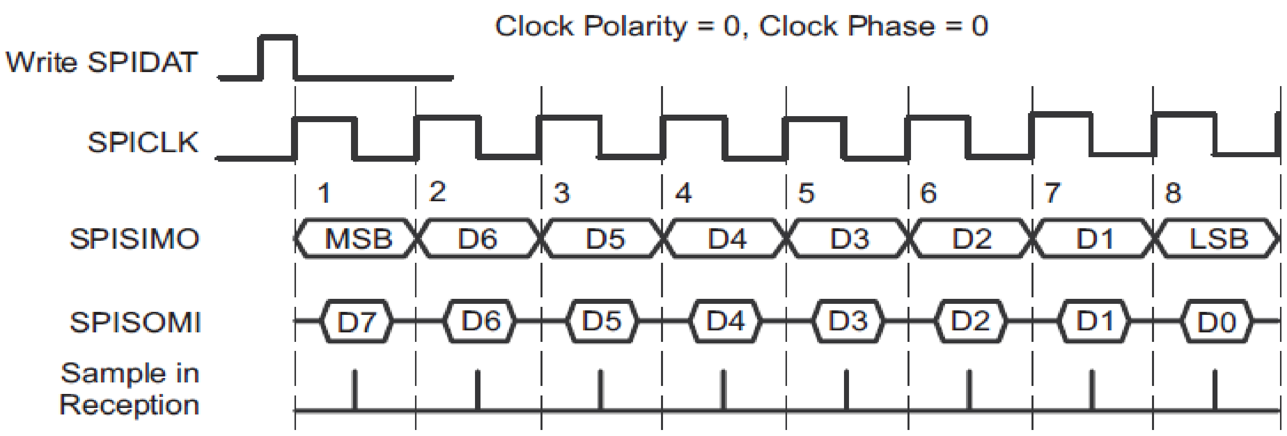


Fig. 5.4 Timing diagram for CPOL=0, CPHA=0

2. Clock Phase=1 which means clock with delay:

 - Data is outputted a half cycle before a the first positive edge of the clock and subsequently falls on every negative edge of the clock
 - Input data is latched on the positive edge of the SPICLK

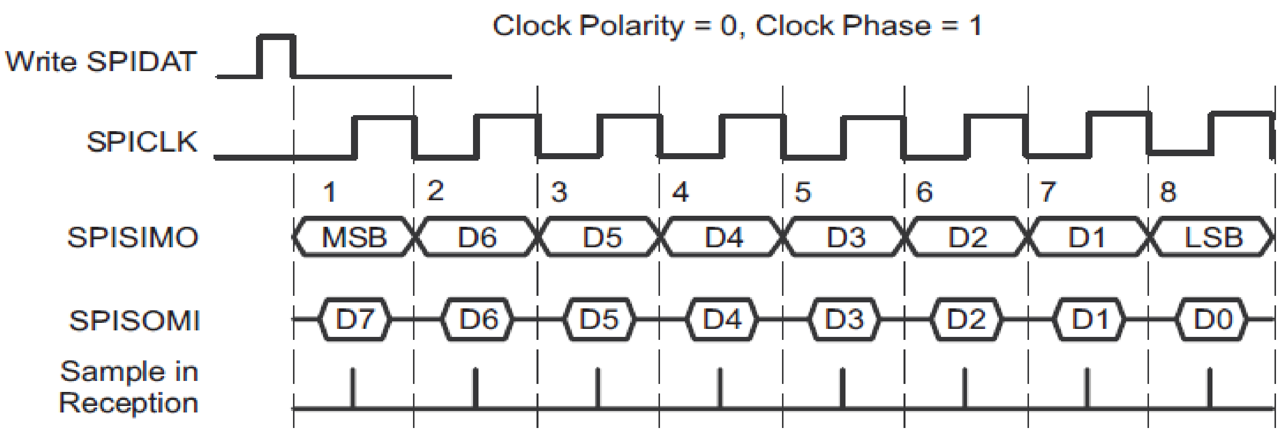


Fig. 5.5 Timing diagram for CPOL=0, CPHA=1

3. Clock Phase=0 which means clock without delay:

 - Data is outputted on every negative edge of clock
 - Input data is latched on every positive edge of the SPICLK
 - The clock starts with the data coming at the data register

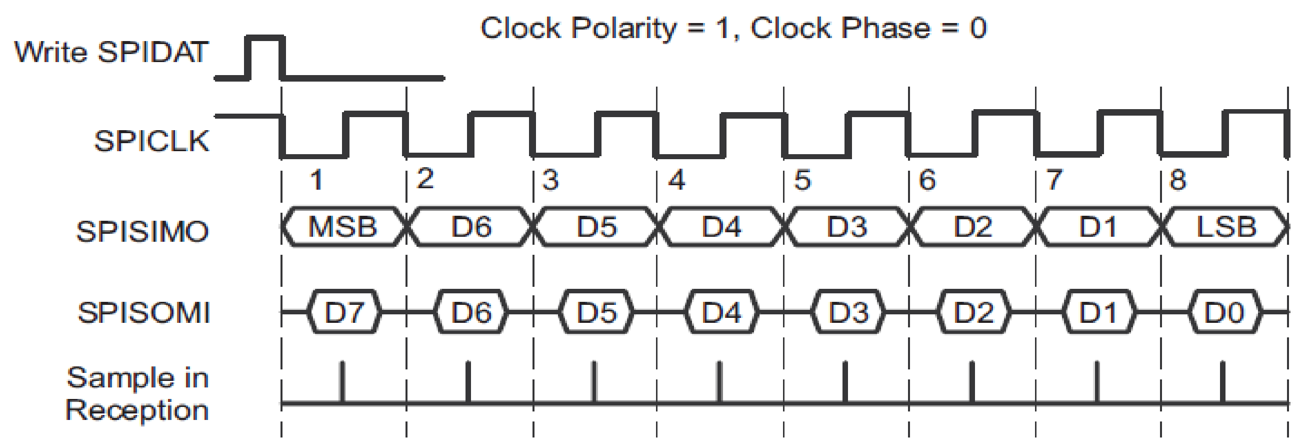


Fig. 5.6 Timing diagram for CPOL=1, CPHA=0

4. Clock Phase=1 which means clock with delay:
   - Data is outputted a half cycle before the first negative edge of the clock and subsequently falls on every positive edge of the clock
   - Input data is latched on the negative edge of the SPICLK
   - The clock starts with the data coming at the data register

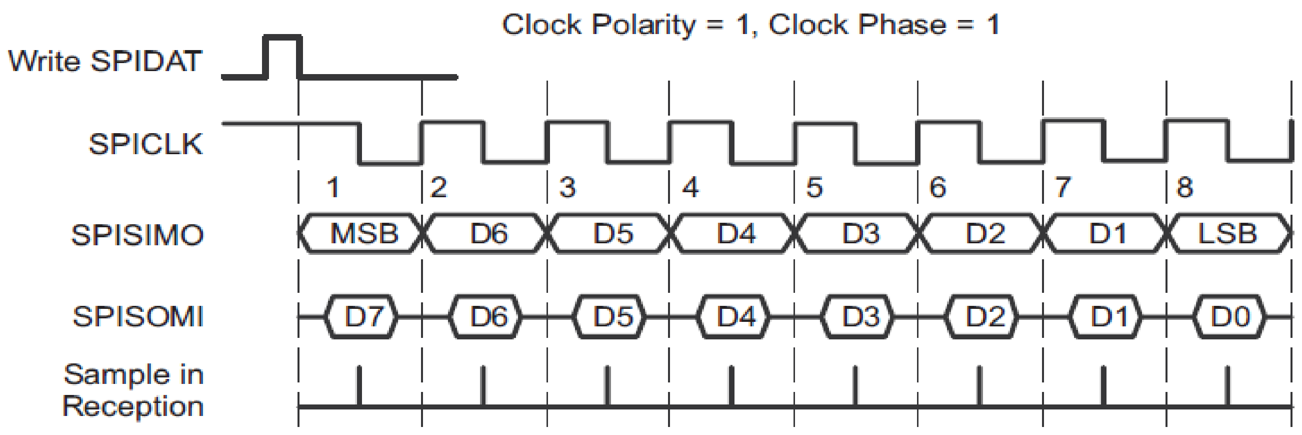


Fig. 5.7 Timing diagram for CPOL=1, CPHA=1

5.5 Master mode:

If the Master bit in Global control register is set high the SPI operates in Master mode ready to drive the signals to slave devices. In addition to that for the SPI module to function as Master, it needs to be set with the following configuration bits in 3-pin and 4-pin modes respectively

Name of the register	Bit	3-pin	4-pin
GCR0	RESET	1	1
GCR1	ENABLE	1	1
GCR1	LOOPBACK	0	0
GCR1	CLKMOD	1	1
GCR1	MASTER	1	1
PC0	SOMIFUN	1	1
PC0	SIMOFUN	1	1
PC0	CLKFUN	1	1
PC0	ENAFUN	0	0
PC0	SCS0FUN	0	1

Table 5.2 Master mode configurable pins

Timing Options in SPI Master mode:

The SPI operating in master mode gives multiple options to supports several options to alter the timing for generating the chip select pin (CS[n]). This allows to add additional timing delays automatically according to the requirement to synchronize the multiple slave devices.

5.5.1 Chip Select setup time:

The master can be made to provide a slower slave, a slave select setup time before the first edge of clock. This setup time can only be enabled while the master is in 4-pin mode. The setup time can be configured using C2TDELAY in the delay register of the SPI registers. The C2TDELAY values range from 2 to 257 SPI module clock cycles. The C2TDELAY begins after the chip select pin is asserted.

The C2T delay period is specified by:

Max C2TDELAY period = C2TDELAY + 2 clock cycles(2 to 257 clocks)

The prior value of the chip select hold bit in the transmit data register must be cleared for this delay to be enabled.

5.5.2 Chip Select Hold time:

The master device can be made to provide a slower slave, a chip select setup time after the last edge of clock. This setup time can only be enabled while the master is in 4-pin mode. The setup time can be configured using T2CDELAY in the delay register of the SPI registers. The T2CDELAY values range from 1 to 256 SPI module clock cycles. The T2CDELAY begins after the end of shift period.

The T2CDELAY delay period is specified by:

Max T2CDELAY period = T2CDELAY + 1 clock cycles(1 to 256 clocks)

The present value of the chip select hold bit in the transmit data register must be cleared for this delay to be enabled

5.5.3 Delay between transmission:

The Master device can automatically put a delay of 2 to 65 clock cycles in between the transmissions. This delay is set by the WDELAY and can be enabled by the delay counter bit. This delay period begins when the T2CDELAY or T2EDELAY terminates or when the master deasserts the chip select when both are disabled.

The WDELAY delay period is specified by:

Max WDELAY period = WDELAY + 2 clock cycles(2 to 65 clocks)

5.5.4 Chip Select Hold option:

There can be slaves which require the chip select to be active for continuous data transmission. There might be slave devices which need the chip selects to be deactivated during successive word transfers. The CSHOLD in data transmit register determines if the chip select is to be held or not. If this hold option is enabled, then both the above delays will not applied before the second transfer and after the first transfer. However the WDELAY period can be applied if it is enabled.

When this bit is disabled while the data transfer is happening, the values of chip select number field in the data transmit register is written to slave chip select pins. All the chip select default pattern values(CSDEF[n:0]) are put on to the slave chip select pins

5.6 SPI Registers:

The SPI module is often considered to be a configurable serial interface. It can be usually configured using the registers. Three types of registers are used mainly: Control, Data and pin registers. The offset is relative to the base address of the module.

Offset Address	Acronym	Register Description
0h	GCR0	Global Control Register 0
4h	GCR1	Global Control Register 1
8h	INT0	Int Register
Ch	INTLVL	Int Level Register
10h	FLG	Flag Register
14h	PC0	Pin Control Register 0
38h	DAT0	Data Tx Register 0
3Ch	DAT1	Data Tx Register 1
40h	RXBUF	Rx Buffer Register
44h	RXEMU	Rx Emulation Register
48h	DELAY	Delay Register
4Ch	DEFCS	Default Chip Select Register
50h	FORMAT0	Data Format Register 0
54h	FORMAT1	Data Format Register 1
58h	FORMAT2	Data Format Register 2
5Ch	FORMAT3	Data Format Register 3
60h	INTVEC0	Int Vector Register 0
64h	INTVEC1	Int Vector Register 1

Table 5.3 Offset addresses for registers

5.6.1 Global Control Register 0:

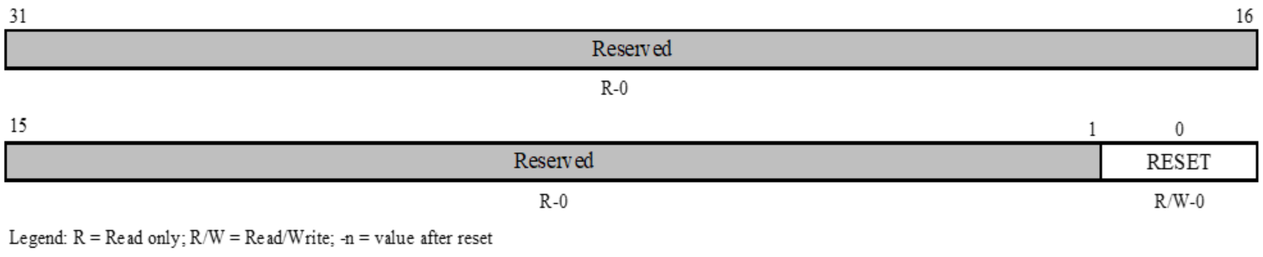

31
16
Reserved
R-0
15
1
0
Reserved
RESET
R-0
R/W-0
Legend: R = Read only; R/W = Read/Write; -n = value after reset

Fig. 5.8 GCR0 Register

Bit	Field	Description
0	RESET	If 0 it is in Reset mode If 1 it's not in reset mode

Table 5.4 Description of GCR0 Register

5.6.2 Global Control Register 1:

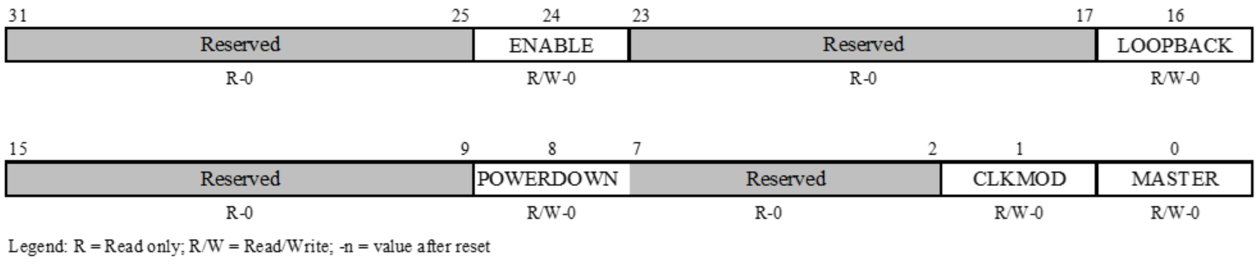


Fig. 5.9 GCR1 Register

Bit	Field	Description
24	ENABLE	This bit when set high enables the SPI transfers When cleared to 0, some of the SPI registers go to their default states 0= Disabled for transfers. 1 = SPI is activated.
16	LOOPBACK	The internal loopback testing can be done through this bit. Both the data bits are internally connected and the transmit data is stored in the receive field. Externally the data pins are in high impedance state and clock gives an inactive value. SPI

		should be in master mode to select loopback 0 = Disabled loopback mode 1 = Loopback mode enabled
8	POWERDOWN	When set high, the SPI enters an inactive state. 0 = Active state. 1 = Inactive mode
1-0	CLKMOD, MASTER	These bits when configured as 3h, represent Master mode. All other modes are retained for compatibility. All the 4 pins act functional when SPI operates as Master.

Table 5.5 Description of GCR1 Register

5.6.3 Interrupt Register:

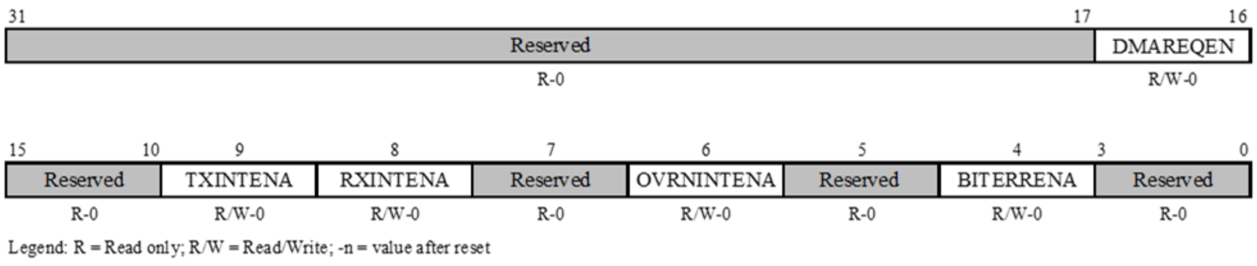


Fig. 5.10 INT0 Register

Bit	Field	Description
16	DMAREQEN	Direct Memory Access Request Enable This bit should be set high only after setting the Enable bit. 0 = Direct Memory Access is not used. 1 = Requests are generated.
9	TXINTENA	Tx int enable. 0 = No int even if flag is high 1 = Int will be generated if the flag is set high
8	RXINTENA	Rx int enable. 0 = No int even if flag is high

		1 = Int will be generated if the flag is set high
6	OVRNINTENA	Overrun int enable 0 = Not enabled. 1 = It will be enabled.
4	BITERRENA	when bit error occurs this interrupt is enabled. 0 = No Int enabled on bit error. 1 = Enables an int on a bit error

Table 5.6 Description of INT0 Register

5.6.4 Interrupt Level Register:

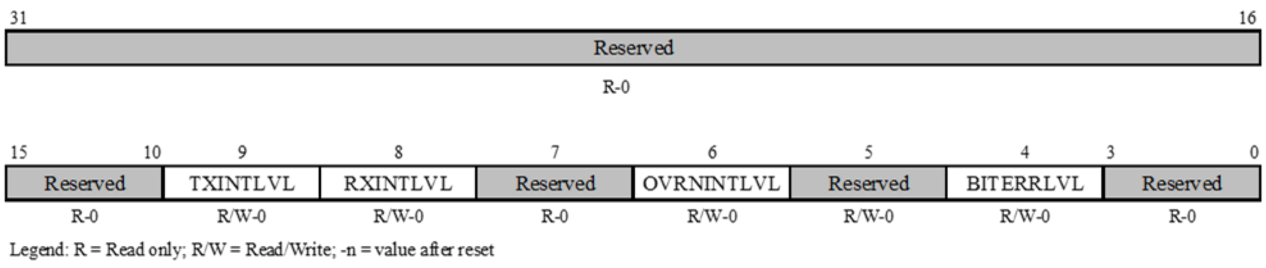


Fig. 5.11 INTLVL Register

Bit	Field	Description
9	TXINTLVL	Tx int level. 0 = mapped to INT0. 1 = mapped to INT1.
8	RXINTLVL	Rx int level. 0 = mapped to INT0. 1 = mapped to INT1.
6	OVRNINTLVL	Rx overrun int level. 0 = mapped to INT0. 1 =mapped to INT1.

4	BITERRLVL	Bit error int level. 0 = mapped to INT0. 1 = mapped to INT1.

Table 5.7 Description of INTLVL Register

5.6.5 Flag Register:

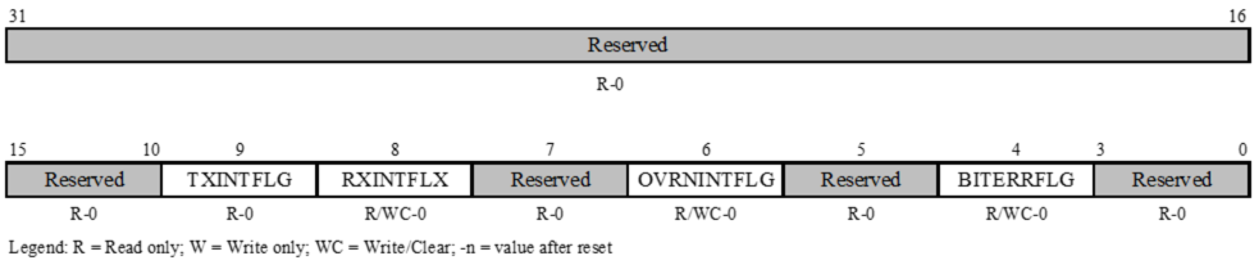


Fig. 5.12 FLG Register

Bit	Field	Description
9	TXINTFLG	Transmitter empty interrupt flag. 0 = It indicates transmitter is full. 1 = It indicates transmit buffer is empty.
8	RXINTFLG	Receiver full interrupt flag. 0 = Rx buffer is empty. 1 = Some new rx data is ready to be read. Rx buffer is full.
6	OVRNINTFLG	Receiver overrun flag. 0 =Receiver overrun did not occur. 1 = Receiver overrun has occurred
4	BITERRFLG	When internal Tx data and the Txed data don't match this flag is set high. If 1 bit error 0 = No bit error 1 = A bit error

Table 5.8 Description of FLG Register

5.6.6 Pin Control Register 0 (Function):

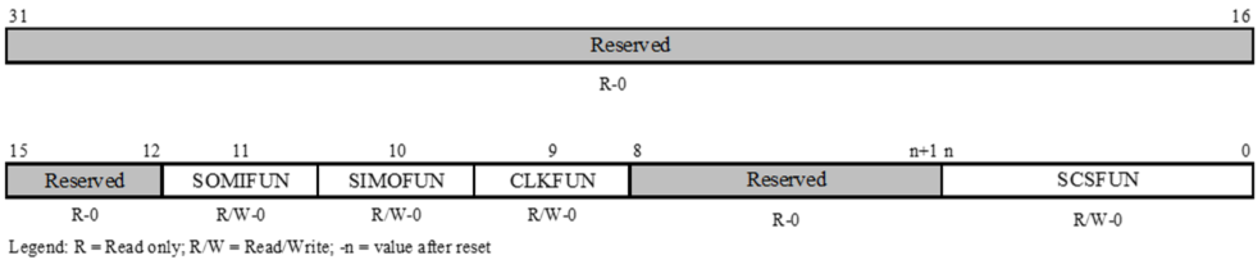


Fig. 5.13 PC0 Register

Bit	Field	Description
11	SOMIFUN	Master In Slave Out pin. It determines if the pin is functional pin. If 1 functional pin
10	SIMOFUN	Master Out Slave In pin. It determines if the pin is functional pin. If 1 functional pin.
9	CLKFUN	Clock function pin. It determines if the pin is functional pin If 1 functional pin.
n-0	SCSFUN	Chip select function pin. It determines if the pin is functional pin. According to i in the chip select position the n bit is configured here. If 1 corresponding functional pin

Table 5.9 Description of PC0 Register

5.6.7 Data Transmit Register 0:

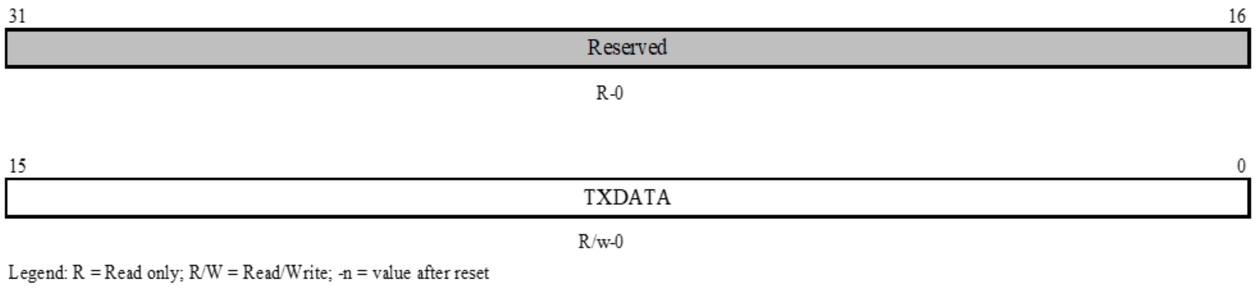

31
16
Reserved
R-0
15
0
TXDATA
R/w-0
Legend: R = Read only; R/W = Read/Write; -n = value after reset

Fig. 5.14 DAT0 Register

Bit	Field	Description
15-0	TXDATA	It represents SPI transmit data. The values vary from 0 to FFFFh.

Table 5.10 Description of DAT0 Register

5.6.8 Data Transmit Register 1:

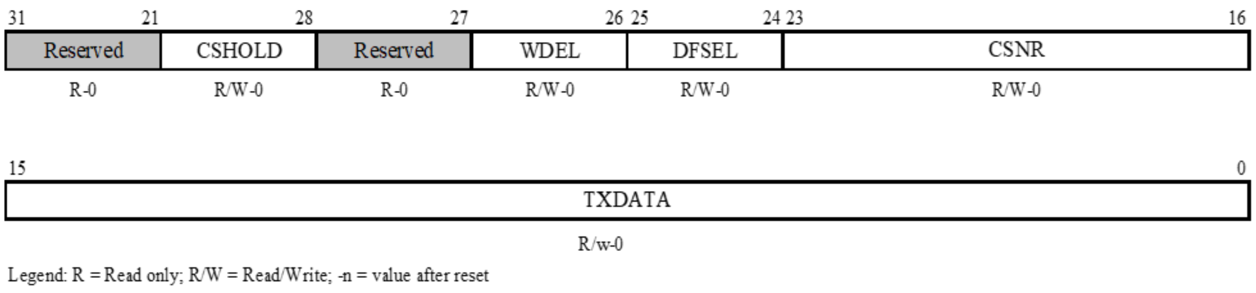


Fig. 5.15 DAT1 Register

Bit	Field	Description
28	CSHOLD	Chip select hold bit. 0 = The Slave chip select pins are restored to default chip select bits at transfer end 1 = The Slave CS pins are continued as chip select number value pins at the end of data transfer
26	WDEL	A bit to enable the delay counter 0 = No delay will be inserted. 1 = After a transfer selected delay will be inserted
25-24	DFSEL	Data word format select: 4 data word formats can be selected. Legal values are 0 to 3h
23-16	CSNR	This field indicates which chip select pin is to be activated during the transfer of data. Legal values are 0 to FFh
15-0	TXDATA	Transmit data field. One of the 16 bit data registers in SPI. Legal values are 0 to FFFFh

Table 5.11 Description of DAT1 Register

5.6.9 Receive Buffer Register:

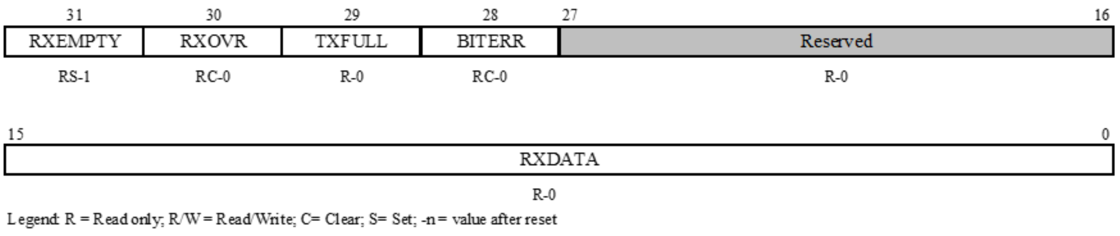

31
RXEMPTY
RS-1
30
RXOVR
RC-0
29
TXFULL
R-0
28
BITERR
RC-0
27
16
Reserved
R-0
15
0
RXDATA
R-0
Legend: R = Read only; R/W = Read/Write; C= Clear; S= Set; -n = value after reset

Fig. 5.16 RXBUF Register

Bit	Field	Description
31	RXEMPTY	Rx buffer empty
30	RXOVR	Rx Buffer Overrun
29	TXFULL	Tx buffer full
28	BITERR	Bit error
15-0	RXDATA	It indicates SPI received data. Legal values are 0 to FFFFh(value = 0-FFFFh).

Table 5.12 Description of RXBUF

5.6.10 Receive Emulation Register:

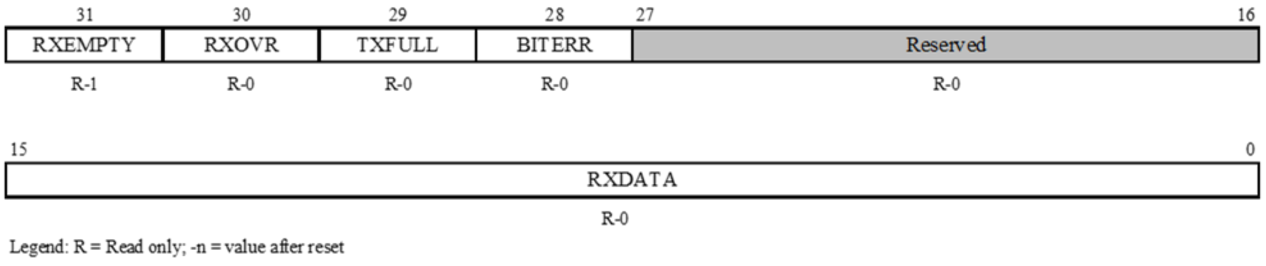


Fig. 5.17 RXEMU Register

Bit	Field	Description
31	RXEMPTY	Rx buffer empty.
30	RXOVR	Rx buffer overrun
29	TXFULL	Tx buffer full flag
28	BITERR	Bit error

15-0	RXDATA	It indicates SPI received data. Legal values are 0 to FFFFh(value = 0-FFFFh). But since this is emulation register, the values are read only

Table 5.13 Description of RXEMU Register

5.6.11 Delay Register:

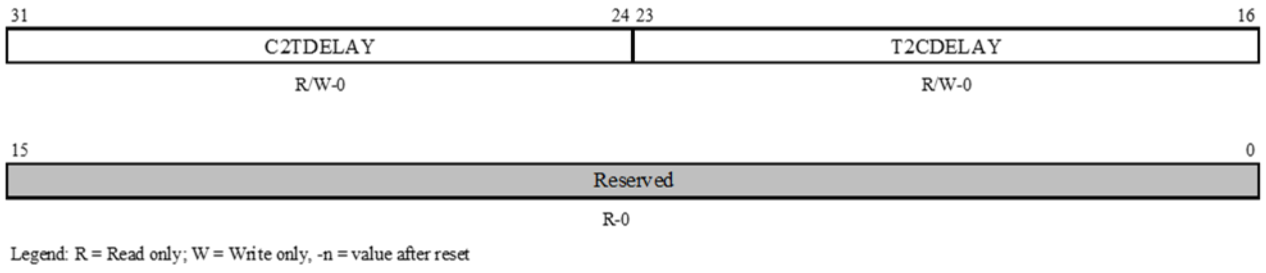

31
24 23
16
C2TDELAY
T2CDELAY
R/W-0
R/W-0
15
0
Reserved
R-0
Legend: R = Read only; W = Write only, -n = value after reset

Fig. 5.18 DELAY Register

Bit	Field	Description
31-24	C2TDELAY	CS setup delay
23-16	T2CDELAY	CS hold delay.

Table 5.14 Description of DELAY Register

5.6.12 Default Chip Select Register:

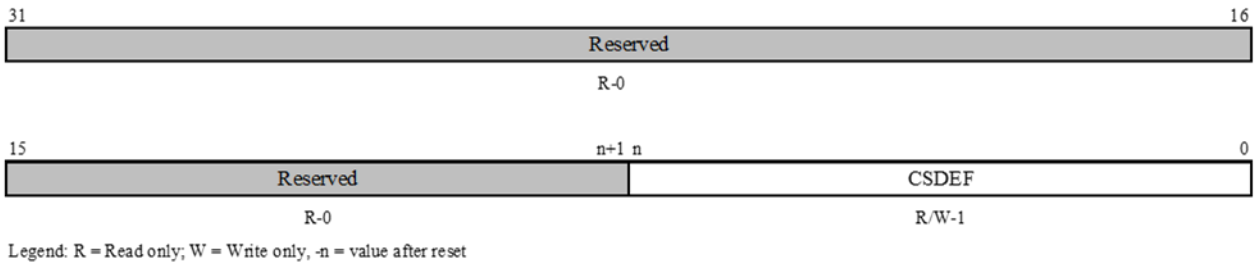

31
16
Reserved
R-0
15
n+1 n
0
Reserved
CSDEF
R-0
R/W-1
Legend: R = Read only; W = Write only, -n = value after reset

Fig. 5.19 DEFCS Register

Bit	Field	Description

n-0	CSDEF	Chip Select Default pattern. 0 = The corresponding CS pin is cleared. 1 = The corresponding CS pin is set to high.

Table 5.15 Description of DEFCS Register

5.6.13 Data Format Register n:

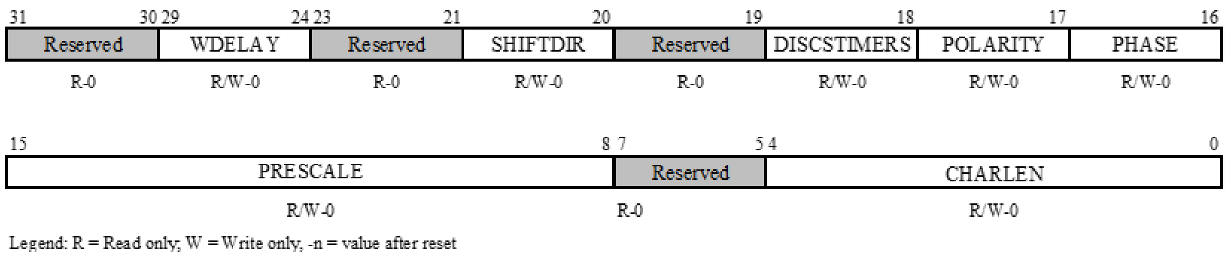


Fig. 5.20 FORMATn Register

Bit	Field	Description
29-24	WDELAY	Delay b/w the transmissions.
20	SHIFTDIR	Shift direction. 0 = MSB first. 1 = LSB first.
18	DISCSTIMERS	If 0 both delays are disabled If 1 both are inserted
17	POLARITY	It represents high or low
16	PHASE	If 0 clock is not delayed If 1 clock is delayed according to data stream
15-8	PRESCALE	Bits to configure the clock
4-0	CHARLEN	Character Length(legal values are 2h to 10h)

Table 5.15 Description of FORMATn Register

5.6.14 Interrupt Vector Register 0:

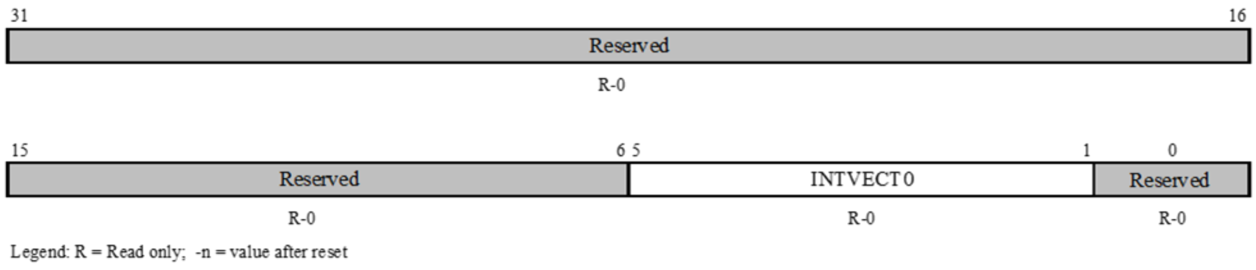


Fig. 5.21 INTVEC0 Register

Bit	Field	Description
5-1	INTVECT0	Int vector for int line INT0. The values vary from 0h to 1Fh 0 = No pending interrupt

Table 5.16 Description of INTVEC0 Register

5.6.15 Interrupt Vector Register 1:

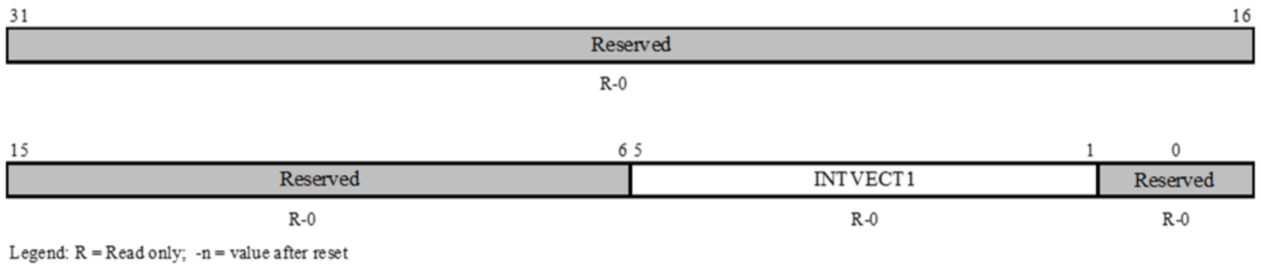


Fig. 5.22 INTVEC1 Register

Bit	Field	Description
5-1	INTVECT1	Int vector for int line INT1. It can range from 0 to 1Fh 0 = No pending interrupt

Table 5.17 Description of INTVEC1 Register

Each register is supplied with a specific function where user gets a chance to configure. These configuration resisters help to program SPI in a way the user wants

CHAPTER 6

DESIGN SPECIFICATION

CHAPTER 6
DESIGN SPECIFICATION

6.1 Introduction:

In the period of integrated circuits, engineers had to take a seat down and physically draw transistors and their connections on paper to style them specified it will be fancied on semiconducting material. Larger and complicated circuits demanded a lot of engineers, time and different resources and shortly enough there was a requirement to own a more robust approach of planning integrated circuits.

VHDL was presently developed to reinforce the design method by permitting engineers to explain functionality of the required hardware and let automation tools convert that behavior into actual hardware components like combinative gates and serial logic. Verilog was developed to reinforce the design method and create the HDL additional sturdy and versatile.

Generally conventional HDL’s are used to define any hardware design which may again contain both combinational and sequential logic. In general, to verify a hardware design we need to have a language that provides tough checking procedures and is referred to as HVL. Hardware Verification Language generally comprises of additional object oriented programming features which help achieve the verification plan.

Back in those days, Verilog was the language that has the capability to check a hardware design. As style quality will increase, thus will the urge of development of bigger tools to style and verify it arises. System Verilog is much superior to Verilog due to its ability to perform unnatural random stimuli, use OOP options in test bench construction, useful coverage, assertions among several others. Therefore today System Verilog is the most widely used for variety of designs.

The basic code which is used as the main building block for any hardware product is referred as RTL design. The RTL design is written generally using Hardware description language such as Verilog/System Verilog/VHDL. RTL(register transfer level) is an idea that defines the digital portion of a design. The language constructs used in a rtl design are mainly suitable to be fed into a logic synthesis tool which in turn creates a gate level netlist that is used for the implementations involved in the flow.

The Design consists of an IP core which should be capable of taking APB bus as input and driving it to output through the SPI Master signals. Since APB is considered as input from an external device which might be part of a higher performance system bus(AHB, AXI, ASB), the module is considered to be a peripheral(slave).

The IP Core should be able to drive SPI slaves which only consist of four signals and are driven serially through clock(synchronous communication). Being a 4 wire protocol it takes very less hardware. Since SPI supports full duplex communication it is important to transmit and receive at the same time. In this case it should act as an interface between the SPI and APB. So, both SPI Master functions and APB slave functions are combinedly embedded into one module. The Fig. 6.1 shows the block diagram of the top module involved in the design specification.

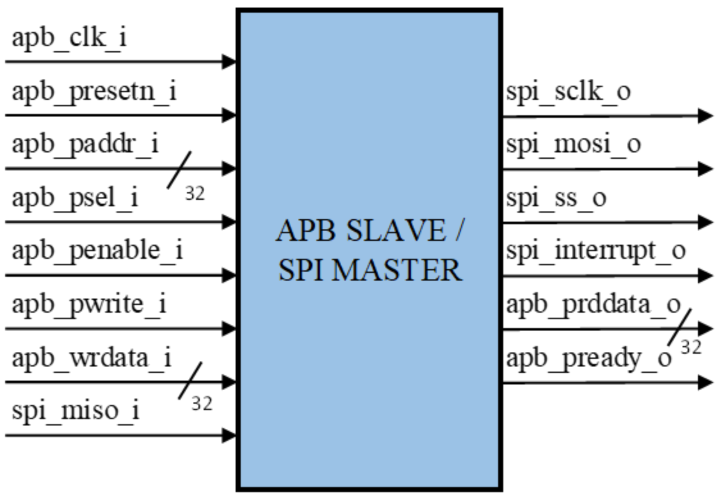


Fig. 6.1 Main block diagram of the design

6.2 APB Slave:

The APB bus is the primary input to this IP core. The bus is generally driven onto an APB slave. The APB bus supports data widths of upto 32 bits wide which consists of a separate data line for read and write operations. The top module has following inputs: psel, penable, pwdata, pwrite, presetn, paddr which are the conventional APB bus inputs along with the miso line of the spi master. The outputs include mosi, SlaveSelect(SS), clk, interrupt and the APB rddata line.

The APB slave provides the following advantages: The address, data, and control are valid throughout the access. When the peripheral bus is not in use it can be static(i.e. consumes zero power). The APB bus generally has three states namely IDLE, SETUP, ENABLE. The SETUP state lasts only for one clock cycle after which if transfer is initiated the state jumps to ENABLE [2]. Depending on the value of pwrite, the read and write operations are performed. The ENABLE also lasts for one clock and when the transfer is required the state again moves to SETUP state. The address and data which are provided through the APB bus are latched on to the corresponding registers respectively.

6.3 SPI Master:

The Serial peripheral Interface(SPI) protocol considered here is a 4-wire protocol which means 3-pin mode along with the chip select or SS(slave-select) pin. The Master Out Slave In (MOSI), Master In Slave Out(MISO), SS(slave Select), Clock(CLK) are the wires used to drive SPI slaves. Depending on the number of slaves the slave select lines increase, it means it requires separate line for each slave device.

It has configurable clock which can be set to certain frequency. Even the clock phase(delay or no delay) and clock polarity(high or low) can be programmed[4]. The character length that is to be written as data can also be configured using the data format register. The SPI Master [3] can be represented as shown in F ig2. The CPU write line is derived from the APB bus and similarly CPU Read data is read to APB prddata line. The SPI Module clock is obtained from the apb pclk which is in turn changed to programmable SPI clock based on the clock generation logic.

6.4 APB to SPI Interface:

Since the APB bus is the input to the whole module, all the APB signals are to be registered which means the data, address and control pins are written into local registers. According to the standard state diagram of APB, the registers of reads and writes are enabled. The basic idea is to latch down the inputs(address and data) whenever both the enable and setup signal is set to high. The address and data is written into the registers only at every positive edge of the clock. Similarly read data is outputted on to pread line from the contents of the rd_data_i register which is obtained internally. The pready signal is asserted according to status of the pwrite and psel signals.

6.5 Registers:

Every operation which happens internally in this work involves registers. Each register in the SPI module helps in the data transfer in some way. Any register that's under SPI is written with data using a data register which is declared in the register module. Firstly after the data and address pass through the apb interface, it is given to the reg_wr_data. Similarly the same happens with the address register. Generally the address register carries the offset value which determines which SPI register the data should be written into whenever the write enable is high. For instance if offset address is 0X50, the data goes into the Data format register(SPIFMT0).

Firstly to write the 32 bit APB data into SPI module we need to configure other registers such as Data format register and Global control register. Before every transaction, the data should be such that, it should help us configure the control and format registers. Following this, the address registers can be pointed to offset addresses 0X38 and 0X3C to write the data into the data transmit registers. However only the first 8 bits of the address registers are compared with the offset addresses. This data which copied into data registers will then go through the Fifo module which functions as a temporary storage element.

6.6 FIFO:

Fifo is a memory structure which can be used in chip designs[14]. It consists of a queue or an array of storage elements where data is written to or read from. Fifo stands for First In First Out. Whenever the data rates and clocks are different we need an intermediate module to synchronise everything. In this case the data rate from an APB bus might be different from the one that is needed to be on the output side. In our case SPI data is to be transferred as output which happens on a full duplex serial communication mode. Since each module has different clock we need to maintain a constant clock rate to synchronise the bits to prevent data loss. So asynchronous fifo is used for this purpose to eliminate the need of same clocks.

A general Fifo is used to perform read and write operations. The read operation is performed by the rd_en and rd_clk. Similarly the write operations is performed by wr_clk and wr_en. The wr_ptr is used to write into the fifo. The rd_ptr is used to read from the fifo. The rd_ptr and wr_ptr are to be checked every time to know whether the Fifo is full or empty. Generally rd_ptr is driven by the rd_clk and wr_ptr is driven by the wr_clk.

6.6.1 Clock domain crossing:

Every time we compare the pointer value we need to have a clock synchroniser to synchronise both the rd_clk and wr_clk, because rd_ptr runs on rd_clk and wr_ptr runs on wr_clk. A typical clock synchroniser works with two flops driven by the same clock as the output clock. In this case if we need output synchronised to rd_clk it should be fed to two flipflops through which wr_ptr goes. So, here clock domain crossing is achieved.

6.6.2 Binary to Gray and Gray to Binary code conversion:

Another thing to concern about is the data loss that occurs here. Since every operation happens in binary format there is a chance of getting a wrong value at the output, in turn getting a false full value(wr_ptr as full). So it would be better to convert a binary value into a gray value using a binary to gray code converter. In turn, after the data is received at the receiver side the data can be reverted back to the gray to binary converter. Here, only graycode is chosen because at a time only one bit changes making it more immune to false full values.

6.6.3 Comparator:

Since the rd_ptr and wr_ptr values are to be compared, there needs to be a comparator. Generally when both the pointers are equal it should mean the Fifo is empty. But whenever the write operation is performed exactly equal to the depth of fifo the pointer returns back to the start position thereby making the pointers equal. To avoid this wrap around condition both the pointers are extended with one bit.

So now if the MSB and LSB bits are different we can say Fifo is full, because of the wrap around the wr_ptr might have the MSB as 1 but the rd_ptr still has MSB cleared to 0. The empty condition is true whenever both the pointers are exactly equal. Based on these comparator values Fifo is read or written. When the fifo is full the data should not be written and when the fifo is empty, the memory is not read. The generalised Asynchronous fifo block is represented by Fig. 6.2

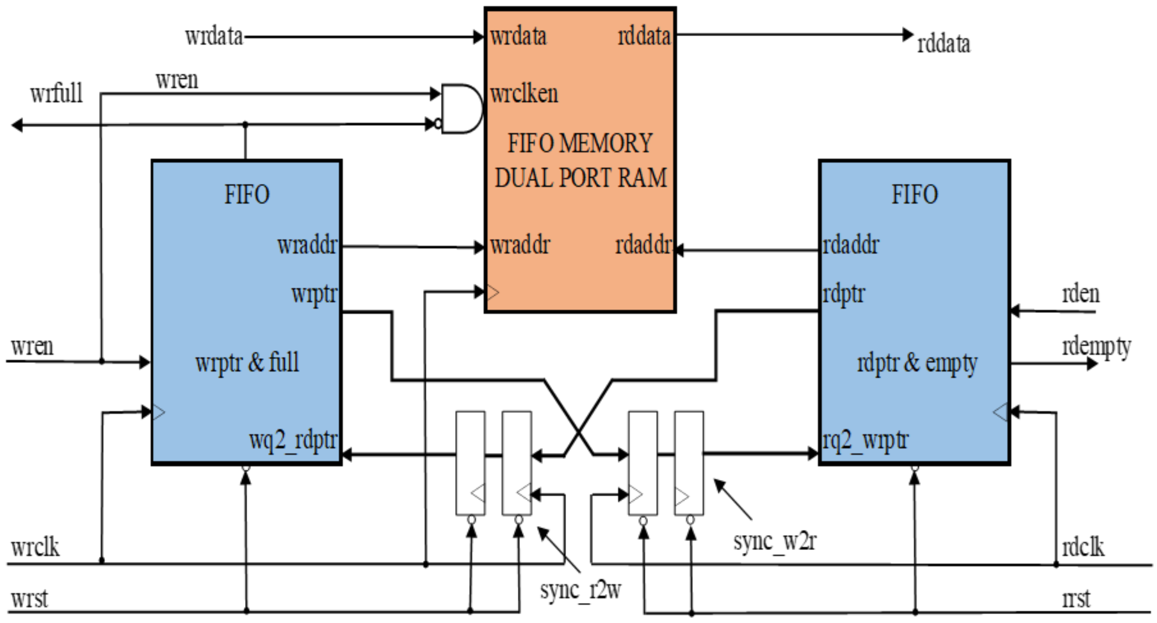


Fig. 6.2 Asynchronous FIFO block diagram

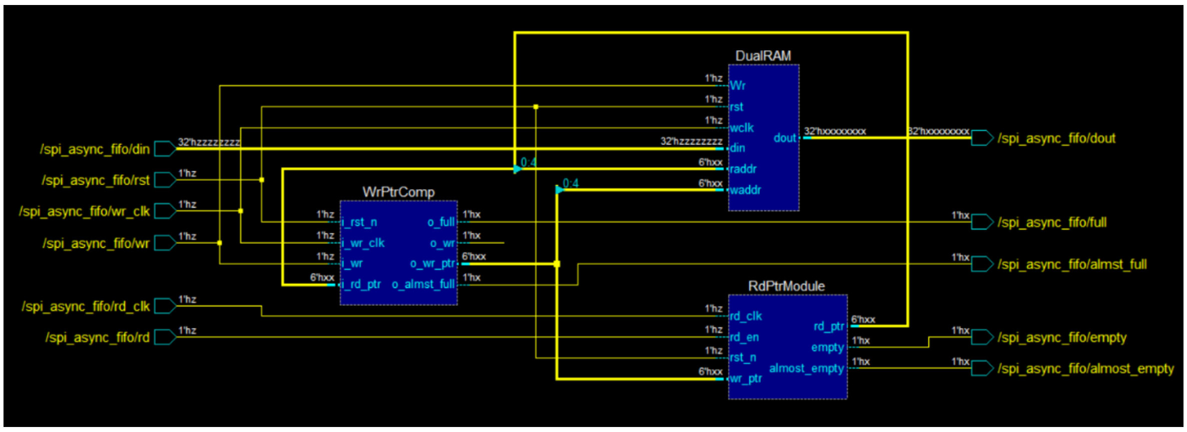


Fig. 6.3 Asynchronous FIFO Schematic

6.7 Data pipeline:

Whenever the data is fed in to the apb_wrdata line, the data is written into the transmit data registers. It may be the txdata0(38h) or txdata1(3Ch) register depending on the offset address you give. The input apb_paddr will determine the offset address you feed into the SPI module. Depending on the bits that configure, data is fed into the data registers. Post the passage through the data registers, the data is managed by an Asynchronous Fifo as in Fig. 6.3. To generate if the fifo is empty or full, we have rd_ptr_empty and wr_ptr_full modules.

A dual port RAM is maintained to deal with the read and write operations. This is used a basic memory element which has some address depth and data width. Here data width is considered to be 32 bit wide. The address size is considered to be 4 bit wide which means 16 addresses are accommodated in the RAM. It generally consists of following input and output ports as in Fig. 6.4.

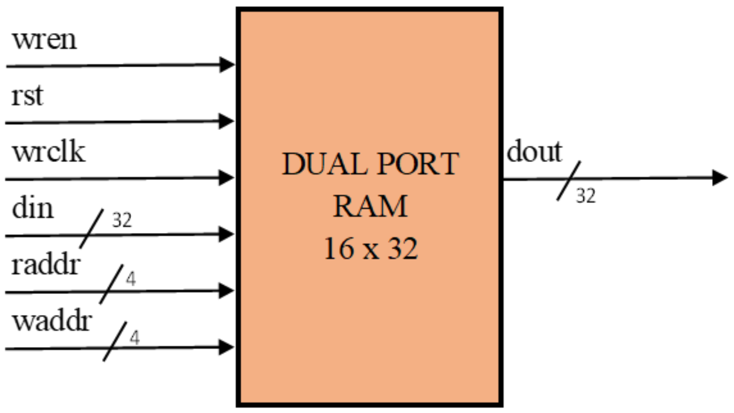


Fig. 6.4 Dual port RAM module

To eliminate loss of data and glitches binary to gray converter and gray to binary converters are used. Since it is an asynchronous fifo the rdptr and wrptr values are to be compared for us to know whether the fifo is full or empty. To transmit and receive correct addresses, the converters are used. The data from Tx data register is fed into the tx_fifo which is in turn shifted and outputted on the MOSI line. The received data through MISO line is shifted first and then given to rx_fifo which is then stored in the rx_buf register.

6.8 Data Transfer:

The Data transfer is initiated when the register bits are configured accordingly. Let’s suppose we need 32 bit data to be transferred. The CHARLEN which describes the word length is to be configured with 11. The design checks for the word size and is assigned to ccount variable. As per Fig. the data you feed to a data transmit register goes through tx_fifo and shifted to Txbuf and then to Txshiftreg via which each bit is outputted on to MOSI line. The rest of the bits are filled with 0 when the bit rates are less than 32 and the data is fed to Txbuf register according to the SHIFTDIR bit. If the bit is cleared to zero, the data is reversed and fed.

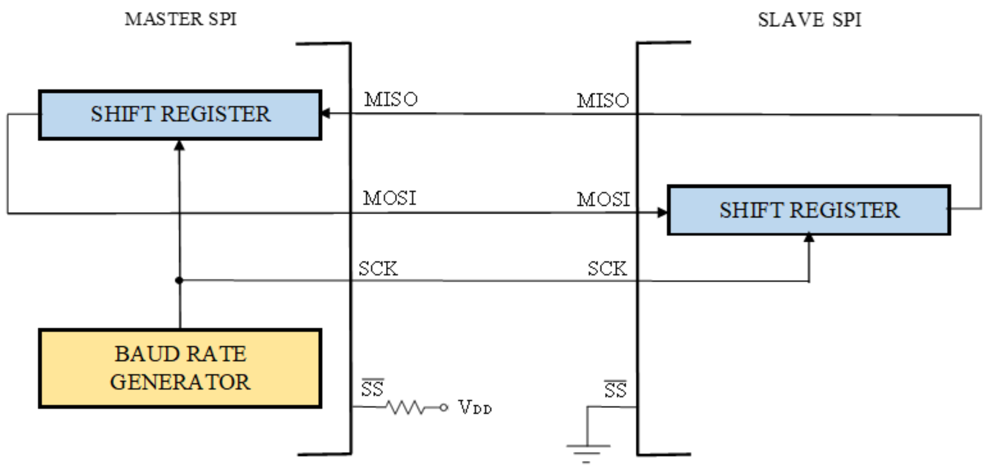


Fig. 6.5 Data transfer between SPI master and slave

When a slave is connected to SPI Master, the data is transmitted mainly via shift registers. These two shift registers(SPIshiftreg and slaveshiftreg) form an inter-device circular buffer[7] as in Fig. 6.5. It means the output of slave shift register is connected to input of the master shift register and the output of the master shift register is connected to input of the slave shift register. The data transfer happens in SISO mode. So, whenever load bit is 1, the first bit from Txbuf is sent on the MOSI line and the SPI shift register is updated with one zero bit. The clock count variable is incremented for every bit of transfer. When the slave is connected and loadnew bit is 1 the Rxshiftreg is updated with the MISO bit and the previous contents of the register. The clock count variable is incremented with 1 for every bit. Conversely when the clock enable is 1 each bit from the shift register is transferred serially on to the MOSI line, the clock variable is decremented by 1. Similarly, the Rxshiftreg is updated with one MISO bit and the previous contents.

6.9 Finite State Machine:

The states considered are IDLE, ASSERT_SS, SHIFT_REG, DISABLE CLK, DEASSERT SS, LOAD_NEW, READ_FIFO. Initially, whenever rst is cleared to zero the state remains at IDLE state. For every positive edge of clock pulse the state jumps to next state. There are 5 states which are generally used. The default state is IDLE. For every posedge of clk, when reset is not cleared the state jumps to next state. Initially slave select is set high when in IDLE state. When SPI enable is set high and when fifo is not empty the state goes to READ_FIFO. Expect in the DEASSERT_SS state the slave select is cleared to zero in every state. The states jump according to the state diagram shown in Fig. 6.6.

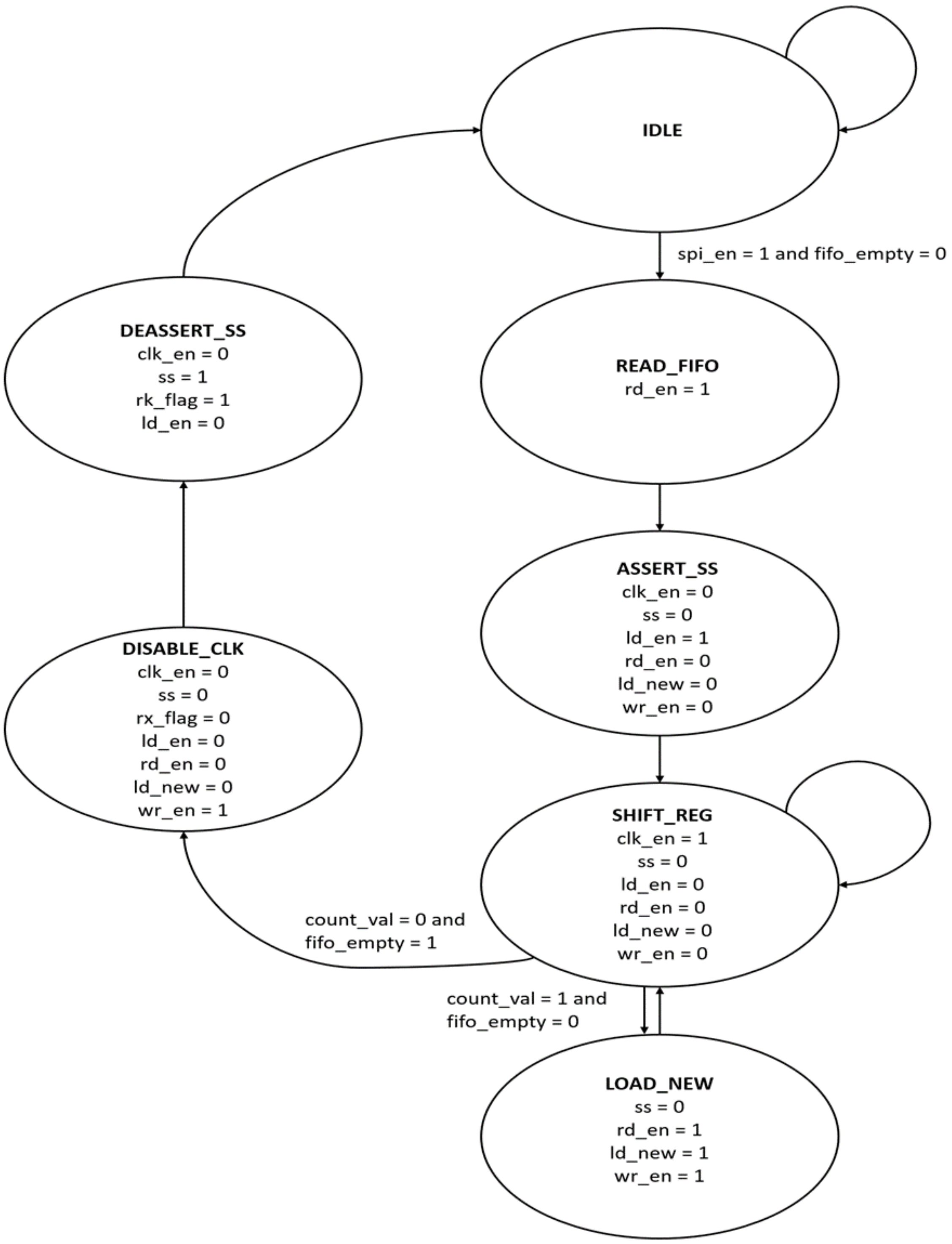


Fig. 6.6 Design state diagram

CHAPTER 7

VERIFICATION

CHAPTER 7
VERIFICATION

7.1 Introduction

It is the technique of making certain that a given hardware style works as designed. IC design can be a difficult and time taking method. Any defect that is guessed at an early stage can save a lot of money. If something undesirable happens in the chip design but detected at the last stage, it can lead to a very big loss in terms of time and money. It might also lead to repeating all the steps

The HVL's like Vera, e, and alternative similar languages are in use for a few time. System Verilog will be a good alternative of Verilog, and it would be feasible to verify a Verilog design in System Verilog. Also System Verilog has object oriented programming constructs which help at later level of hierarchy to verify all the functionalities. System Verilog may be a language similar to Verilog and has its own constructs, syntax and options, however UVM[24][28][32] may be a framework of System Verilog categories from where totally useful testbenches are written.

Testbench is the module that contains all the stimuli needed for verification of a particular design[31]. In olden days the testbench was written using Verilog only, since design would be of few lines only. But these days since complexity is increasing day by day the design needs to have multiple random stimuli to verify each and every case

So basically what we do is, DUT is instantiated in the testbench top module. Generally in system Verilog there is a practice of declaring all the signals as logic in the interface. This interface has a mix of all the inputs and outputs. The testbench job is to hold the stimuli and mount them on to the design. The verification plan ensures that the stimuli is covering every functionality around the design and in turn verifies if there is a match between the expected and actual values of the design.

Verification is the process of making sure the design will work expected. Chip designing becomes a time consuming process when not verified at the right time [9]. So the design has to be verified in the coding stage itself to detect any functional defects at an early stage. Generally testbench is required to verify a hardware design. Testbench is an environment consisting of different set of stimuli which are driven on to the design to check if the output matches the expected results. For smaller designs even Verilog testbenches are sufficient. For a complex hardware design like this testbenches are written in System Verilog. Generally the outputs are matched externally by users. In System Verilog verification can be automated using Functional Coverage and System Verilog Assertions[27] with just very minimal code.

7.2 Testbench design:

The testbench design includes driving the inputs with a task[24]. The task is driven multiple times with different values whenever it is called. A covergroup is defined which consists of coverpoint for every APB input. Each coverpoint is created with implicit bins. To cover every value the design is run for a specific time. Generally if the inputs are run manually by assigning values the bins might not be hit because the range of address of data values are large. Just for the verification purpose the address and data values are randomised and checked for the coverage. Since the testbench is given a specific finish time, the coverage is achieved fully. Using just a command, code coverage can be enabled which shows the coverage with respect to toggles, branches, statements etc.

System Verilog Assertions are written to verify the features of the design. Concurrent Assertions are used to check the state diagram if it works or not. Each state when followed by particular state as given in the state diagram, it indicates the design is working fine. The exact same thing is checked with SVAs included in the testbench.

7.2.1 Functional Coverage

Functional Coverage is a measure of testing features which are included in the hardware design[30]. This poses limitations when all the specifications are not included in the design. The coverage is used just to see if all the written design is covered functionally. Here the idea is to sample interesting variables which may be constrained to check if they reach a certain set of values. Coverpoints are defined for each variable which are given as input stimuli. An Entity called Bins are used to separate values in the given range of possible values for a coverpoint variable. Under each coverpoint there can be any number of bins. There are two types of bins

- Implicit bins: For an n bit variable, 2^n bins are created automatically
- Explicit: Using the keyword these are created. These can be along with some transitions or ignore bins

All the coverpoints can be encapsulated into a covergroup. A covergroup can be included into a module, package, program, interface or a class. Evaluation of a coverpoint expression happens when the covergroup is sampled. An instance of the covergroup has to be created to initiate the process. We can use sample() method to sample the coverpoints within that group. The other method can be to specify an event at which the covergroup has to be sampled. Usually positive edge of clock is used an event to sample at.

Example of coverpoint:

covergroup apbspiCG @(posedge apb_pclk_i);
apb_wr: coverpoint apb_pwrite_i {}
endgroup

7.2.2 Assertions

The behaviour of a design can be validated using an Assertion[29]. The behaviour of a system can be defined using properties. The assertions are defined such that the property is true at all times. The assertion gets passed if it is true, if not it gets failed. A sequence of multiple logical events form the functionality of a design. A number of sequences can be combined to form a property or complex sequences. Using $assert we can define an immediate assertion which is included in a procedural block of code and runs during the simulation. Concurrent assertions are the one which should be included using a clocking event. Using assert keyword we can check if the property defined is true or not.

Example of Assertion to check FSM:
property a1;
@(posedgeclk) (state==IDLE)|=>(state==READ_FIFO)||(state==IDLE);
endproperty

This is a concurrent assertion and is only sampled at the clocking event. The operator “|=>” is called Non overlapped Implication operator which checks if when the antecedent is high and then the consequent is high in the next cycle.

The verification environment consists of the testbench cases where the input is checked for all the possible values and the output is checked with the assertions. The design covers APB interface as input checked in[26] and the SPI Master interface checked in [25] using System Verilog. The coverage report that is generated includes the toggles, branches, statements, expressions used in the APB Slave - SPI Master interface design.

CHAPTER 8

SIMULATION RESULTS

CHAPTER 8

SIMULATION RESULTS

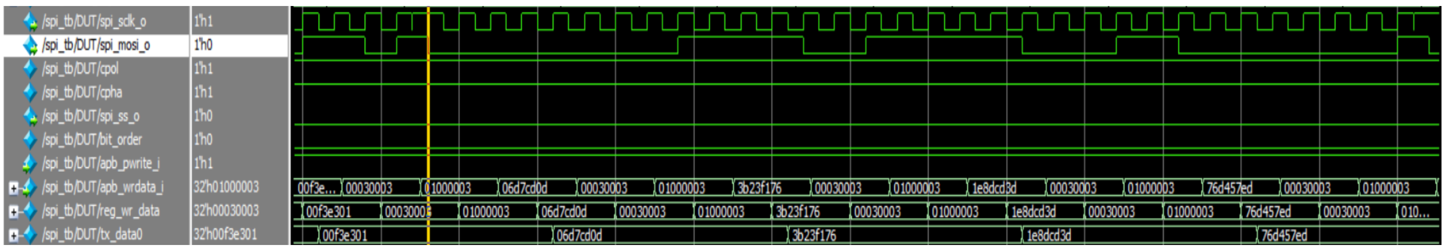
The simulation is performed with Questa Sim 10.7. The design and the testbench is written in system Verilog. Since the testbench includes covergroups and System Verilog Assertions, the simulation should include those also. The simulation is run through commands in the transcript tab. For example the command to include coverage and to compile the testbench is:

vlog +cover tb_apb_spi.sv

The command to run the simulation with coverage and to create a schematic design stored in some database is as follows:

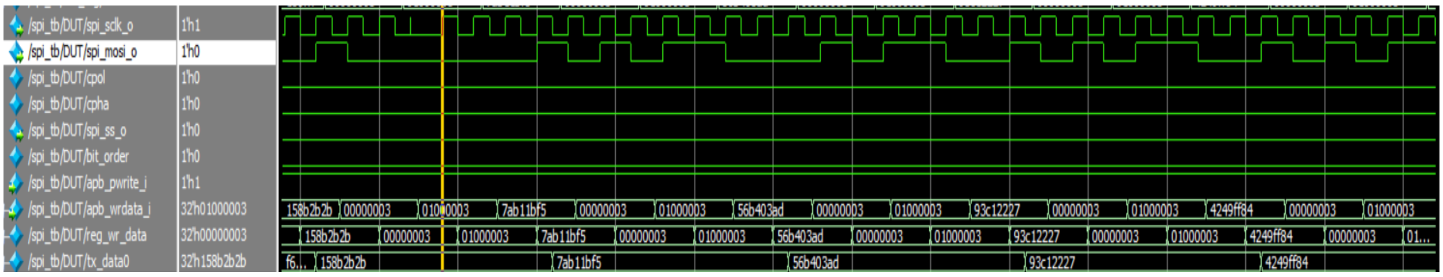
vsim -cover -debugDB work.main_tb

The Fig. 8.1 represents the data stream on MOSI line while SPI is operated in clock mode-4. The Fig. 8.2 represents the data stream on MOSI line while SPI is operated in clock mode-1. The Fig. 8.3 represents the coverage statistics of the covergroup that is included in the testbench. It shows that every coverpoint is hit and shows a 100% coverage of the inputs. Fig. 8.4 shows the System Verilog Assertions that are included for the main state machine. The state machine is tested with the concurrent assertions which are best for checking the state that follows the previous state. It shows every state is asserted at the right time to perform the data transfer. Fig. 8.5 represents the overall coverage report including the code coverage, assertions and the functional coverage. The coverage obtained is 82.13% which degraded mainly because of some code coverage elements. Those include some expressions and toggles of some minor variables. Else, the coverage obtained is a fairly good amount and the design can be implemented for future applications.



/spi_tb/DUT/spi_sclk_o 1'h1
/spi_tb/DUT/spi_mosi_o 1'h0
/spi_tb/DUT/cpol 1'h1
/spi_tb/DUT/cpha 1'h1
/spi_tb/DUT/spi_ss_o 1'h0
/spi_tb/DUT/bit_order 1'h0
/spi_tb/DUT/apb_pwrite_i 1'h1
/spi_tb/DUT/apb_wrdata_i 32'h01000003
/spi_tb/DUT/reg_wr_data 32'h00030003
/spi_tb/DUT/tx_data0 32'h00f3e301

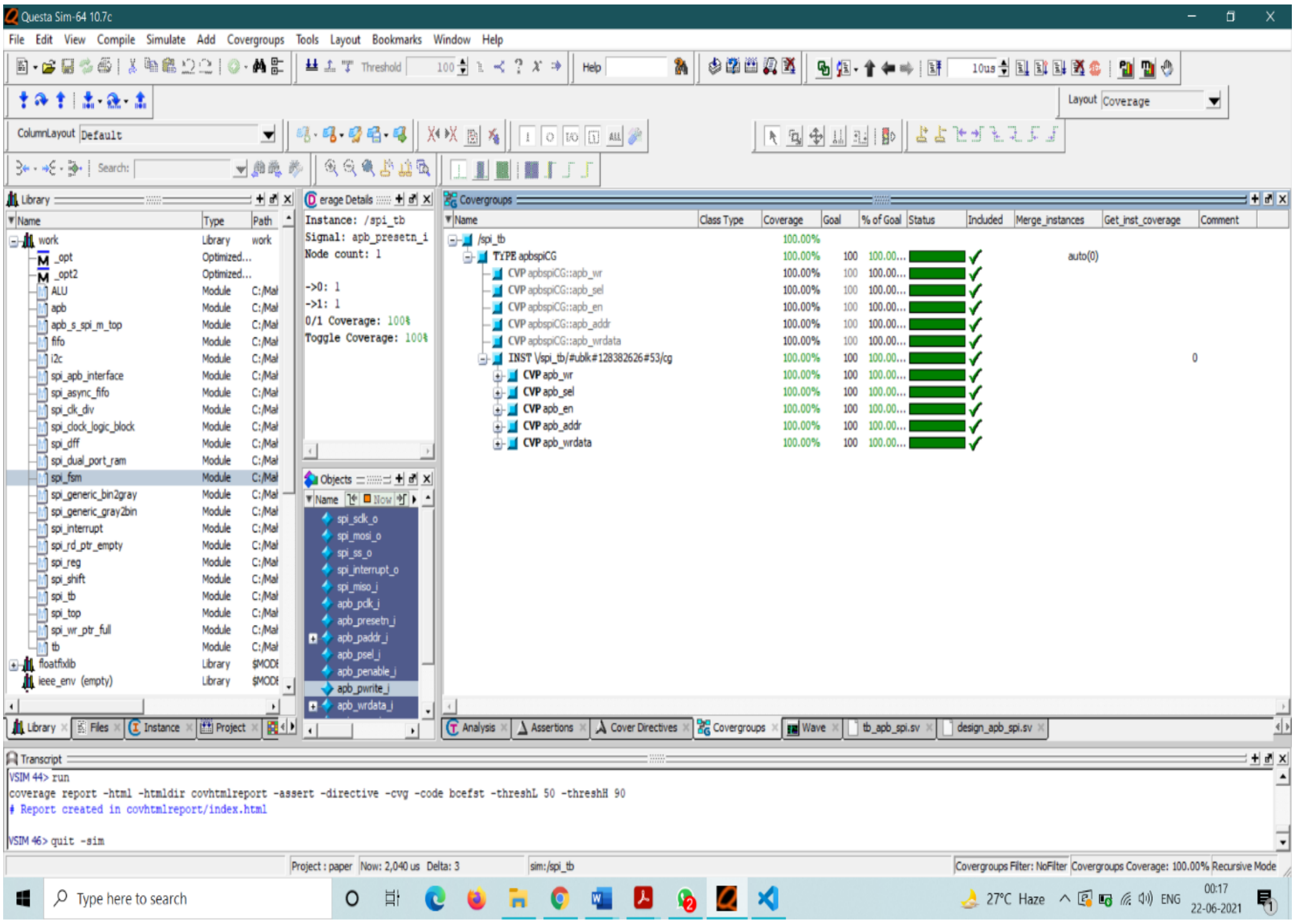
Fig. 8.1 Data transfer on MOSI line with clock mode-4



Signal	Value
/spi_tb/DUT/spi_sclk_o	1'h1
/spi_tb/DUT/spi_mosi_o	1'h0
/spi_tb/DUT/cpol	1'h0
/spi_tb/DUT/cpha	1'h0
/spi_tb/DUT/spi_ss_o	1'h0
/spi_tb/DUT/bit_order	1'h0
/spi_tb/DUT/apb_pwrite_i	1'h1
/spi_tb/DUT/apb_wrdata_i	32'h01000003
/spi_tb/DUT/reg_wr_data	32'h00000003
/spi_tb/DUT/tx_data0	32'h158b2b2b

158b2b2b 00000003 01000003 7ab11bf5 00000003 01000003 56b403ad 00000003 01000003 93c12227 00000003 01000003 4249ff84 00000003 01000003
158b2b2b 00000003 01000003 7ab11bf5 00000003 01000003 56b403ad 00000003 01000003 93c12227 00000003 01000003 4249ff84 00000003 01...
f6... 158b2b2b 7ab11bf5 56b403ad 93c12227 4249ff84

Fig. 8.2 Data transfer on MOSI line with clock mode-1



Questa Sim-64 10.7c

Layout: Coverage
ColumnLayout: Default

Library

Name	Type	Path
work	Library	work
_opt	Optimized...	
_opt2	Optimized...	
ALU	Module	C:/Mal
apb	Module	C:/Mal
apb_s_spi_m_top	Module	C:/Mal
fifo	Module	C:/Mal
i2c	Module	C:/Mal
spi_apb_interface	Module	C:/Mal
spi_async_fifo	Module	C:/Mal
spi_clk_div	Module	C:/Mal
spi_clock_logic_block	Module	C:/Mal
spi_dff	Module	C:/Mal
spi_dual_port_ram	Module	C:/Mal
spi_fsm	Module	C:/Mal
spi_generic_bin2gray	Module	C:/Mal
spi_generic_gray2bin	Module	C:/Mal
spi_interrupt	Module	C:/Mal
spi_rd_ptr_empty	Module	C:/Mal
spi_reg	Module	C:/Mal
spi_shift	Module	C:/Mal
spi_tb	Module	C:/Mal
spi_top	Module	C:/Mal
spi_wr_ptr_full	Module	C:/Mal
tb	Module	C:/Mal
floatfixlib	Library	$MODE
ieee_env (empty)	Library	$MODE

Coverage Details

```
Instance: /spi_tb
Signal: apb_presetn_i
Node count: 1

->0: 1
->1: 1
0/1 Coverage: 100%
Toggle Coverage: 100%
```

Objects

spi_sclk_o, spi_mosi_o, spi_ss_o, spi_interrupt_o, spi_miso_i, apb_pclk_i, apb_presetn_i, apb_paddr_i, apb_psel_i, apb_penable_i, apb_pwrite_i, apb_wrdata_i

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/spi_tb		100.00%							
TYPE apbspiCG		100.00%	100	100.00...	🟩	✓	auto(0)		
CVP apbspiCG::apb_wr		100.00%	100	100.00...	🟩	✓			
CVP apbspiCG::apb_sel		100.00%	100	100.00...	🟩	✓			
CVP apbspiCG::apb_en		100.00%	100	100.00...	🟩	✓			
CVP apbspiCG::apb_addr		100.00%	100	100.00...	🟩	✓			
CVP apbspiCG::apb_wrdata		100.00%	100	100.00...	🟩	✓			
INST \/spi_tb/#ublk#128382626#53/cg		100.00%	100	100.00...	🟩	✓		0	
CVP apb_wr		100.00%	100	100.00...	🟩	✓			
CVP apb_sel		100.00%	100	100.00...	🟩	✓			
CVP apb_en		100.00%	100	100.00...	🟩	✓			
CVP apb_addr		100.00%	100	100.00...	🟩	✓			
CVP apb_wrdata		100.00%	100	100.00...	🟩	✓			

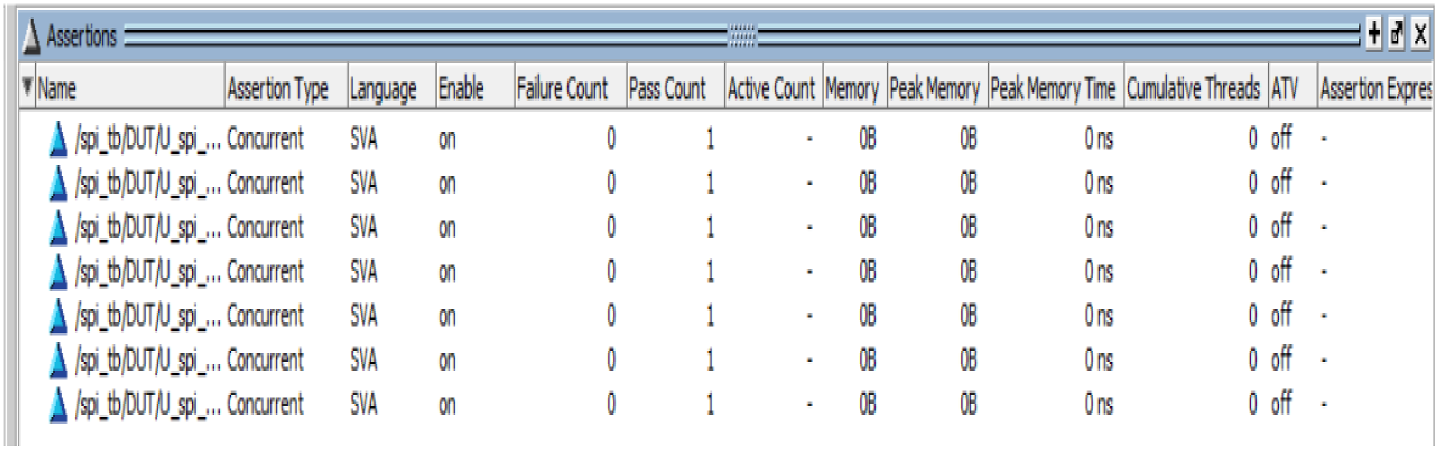
Transcript

```
VSIM 44> run
coverage report -html -htmldir covhtmlreport -assert -directive -cvg -code bcefst -threshL 50 -threshH 90
# Report created in covhtmlreport/index.html

VSIM 46> quit -sim
```

Project : paper | Now: 2,040 us Delta: 3 | sim:/spi_tb | Covergroups Filter: NoFilter | Covergroups Coverage: 100.00% | Recursive Mode

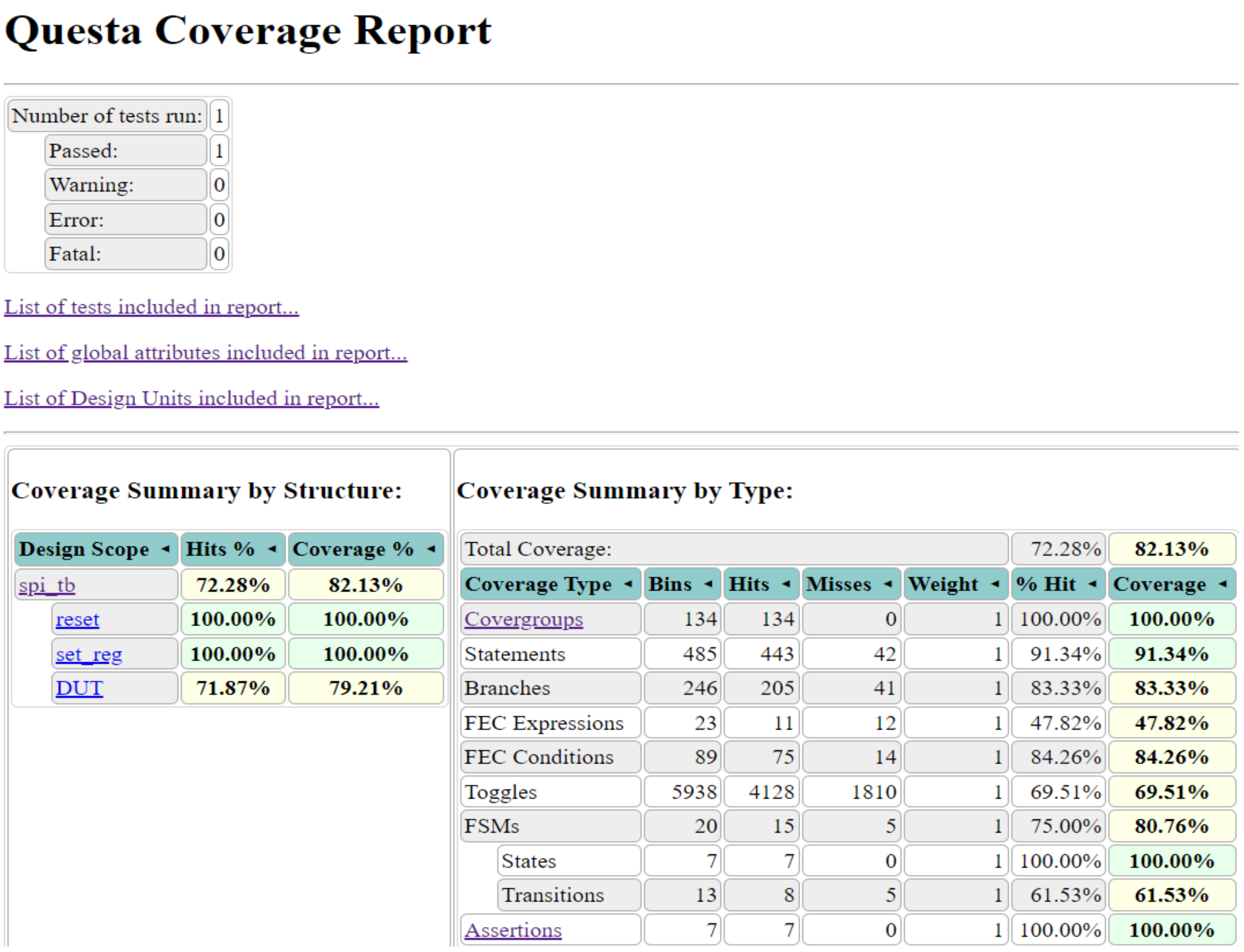
Fig. 8.3 Covergroup statistics



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expres
/spi_tb/DUT/U_spi_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
/spi_tb/DUT/U_spi_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
/spi_tb/DUT/U_spi_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
/spi_tb/DUT/U_spi_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
/spi_tb/DUT/U_spi_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
/spi_tb/DUT/U_spi_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
/spi_tb/DUT/U_spi_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-

Fig. 8.4 Assertions status



Questa Coverage Report

Number of tests run:	1
Passed:	1
Warning:	0
Error:	0
Fatal:	0

List of tests included in report...

List of global attributes included in report...

List of Design Units included in report...

Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
spi_tb	72.28%	82.13%
reset	100.00%	100.00%
set_reg	100.00%	100.00%
DUT	71.87%	79.21%

Coverage Summary by Type:

Total Coverage:					72.28%	82.13%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	134	134	0	1	100.00%	100.00%
Statements	485	443	42	1	91.34%	91.34%
Branches	246	205	41	1	83.33%	83.33%
FEC Expressions	23	11	12	1	47.82%	47.82%
FEC Conditions	89	75	14	1	84.26%	84.26%
Toggles	5938	4128	1810	1	69.51%	69.51%
FSMs	20	15	5	1	75.00%	80.76%
States	7	7	0	1	100.00%	100.00%
Transitions	13	8	5	1	61.53%	61.53%
Assertions	7	7	0	1	100.00%	100.00%

Fig. 8.5 Overall Coverage report

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

CHAPTER 9
CONCLUSIONS

9.1 Conclusion

In this thesis work, SoC to serial peripheral device communication protocol is designed. It is verified using functional and code coverage based verification. The features are tested using the System Verilog Assertions(SVA). This work presents a design which acts both as a APB Slave and an SPI Master. Being a peripheral bus, APB serves lower bandwidth operations which involve lesser number of peripherals and low speed devices. APB being hierarchically the lowest in the AMBA buses, it has the capability to be driven by a high performance system bus and share the bus to a low performace device. But sometimes the peripherals might need serial data stream which is common for devices like ADC, Flash memory. In that situation this design might be helpful. Since the design is verified functionally at an early stage, working on a bug is easy and cost effective.

The design involves a configurable interface of SPI driven by the APB bus. Now-a-days serial protocols are used in most of the external peripheral devices. Out of those SPI can be considered one of the most powerful protocols which provides a high speed, full duplex, serial communication. Each step discussed inside the Serial Peripheral Interface can be programmed according to the needs. The design mainly involves the keystone architecture defined by Texas Instruments in [8]. There are different types of registers for functions such as data format, data transmit, global control, interrupt, delay, pin control. Using these each function can be programmed according to the needs which makes the design flexible and convenient to use.

Through configuration registers one can use the SPI module as Master or a slave, one can decide on the four clock transmission modes, one can change the direction of shift for the transmit/receive data, one can alter the character length of the bit stream, one can program the SPI clock frequency, one can put the SPI under powerdown mode, one can even configure the SPI to function in 3-pin mode, one can work on the delays which affect the slave data transfer.

The design involves a very efficient Asynchronous fifo which is employed both for transmission and reception. The configurable width and height of the fifo makes it compatible for any design and throughput requirements. Since the IP accommodates APB bus, it makes it easy to integrate any SoC which is compliant with APB bus and require SPI peripherals to be communicated with.

Here, the data from the parallel bus passes through the registers and according to the state diagram the state jumps to each operation and thereby transmitting the data stream from the txbuf to the output MOSI line. Since our design includes just an SPI Master, MISO line is received with a data stream that is driven by the testbench itself. However employing a SPI Slave would serve the purpose of full duplex communication. Since there is a flexibility of configuring the SPI device as a slave also, we can reiterate the code and create a slave connected to this design thereby creating a closed loop of connections. In this design there is an option even to self-test the SPI module by just a configuring a bit. The loopback option developed by [8] helps to create a feature where the MISO and MOSI pins are internally connected to each other creating a loop.

9.2 Future work:

An APB bus is integrated with the SPI peripheral devices in this thesis work. Generally AMBA bus is the most commonly used On-chip communication protocol, but other SoC protocols like On-chip Peripheral Bus, Avalon bus, STbus, Open Core protocol need integration with SPI peripheral devices. The design has an improvement scope of implementation on a FPGA like in [18-21]for a real time use check. It can be deployed on a complex hardware where it needs a communication with the external serial peripherals. The SPI as discussed in literature review is used in the FPGA reconfiguration with a Flash memory[23]. So it can be used for a variety of IPs which include the FPGA implementation. The APB interface is used in a CMOS image sensor for better enhancement of images[35]. Similarly an SPI driver is implemented based on an embedded real time OS VxWorks which is used in IoT applications[34]. So the SPI and APB interfaces will find use in several embedded and SoC applications respectively.

References

References

[1] ARM, "AMBA Specification (Rev 2.0)" [online] Available: http://www.arm.com.

[2] ARM, "AMBA APB protocol Specification (v 2.0)" [online] Available: http://www.arm.com.

[3] Patil, Rohita & Sangamkar, Pratima. (2015). A Review of System-On-Chip Bus Protocols. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering. 04. 271-281. 10.15662/ijareeie.2015.0401042. Arthur Mazer "Electric Power Planning for Regulated and Deregulated Markets" IEEE press Wiley inter science, e-Book,, ISBN: 978-0-470-11882-5, Overview

[4] Jovanovic, Milica & Stojcev, Mile. (2006). A Survey of Three System-on-Chip Buses: AMBA, CoreConnect and Wishbone.

[5] A. Shrivastav, G. S. Tomar and A. K. Singh, "Performance Comparison of AMBA Bus-Based System-On-Chip Communication Protocol," 2011 International Conference on Communication Systems and Network Technologies, 2011, pp. 449-454, doi: 10.1109/CSNT.2011.98.

[6] Leens, F. (2009). An introduction to I2C and SPI protocols. IEEE Instrumentation & Measurement Magazine, 12(1), 8–13. doi:10.1109/mim.2009.4762946

[7] Freescale Semiconductor, (2008, October 14). Freescale SPI Block Guide V04.01 Jul. 14

[8] Texas Instruments, "KeyStone Architecture Serial Peripheral Interface (SPI), User Guide," Mar. 2012.

[9] NXP Semiconductors, "I2C-bus specification and user manual, UM10204 datasheet, Rev. 6," Apr. 2014.

[10] B. N. Manu and P. Prabhavathi, "Design and implementation of AMBA ASB APB bridge," 2013 International Conference on Fuzzy Theory and Its Applications (iFUZZY), 2013, pp. 234-238, doi: 10.1109/iFuzzy.2013.6825442.

[11] J. Chhikara, R. Sinha and S. Kaila, "Implementing Communication Bridge between I2C and APB," 2015 IEEE International Conference on Computational Intelligence & Communication Technology, 2015, pp. 235-238, doi: 10.1109/CICT.2015.19.

[12] L. Bacciarelli, G. Lucia, S. Saponara, L. Fanucci and M. Forliti, "Design, testing and prototyping of a software programmable I2C/SPI IP on AMBA bus," 2006 Ph.D. Research in Microelectronics and Electronics, 2006, pp. 373-376, doi: 10.1109/RME.2006.1689973.

[13] M. Hafeez and A. Saparon, "IP Core of Serial Peripheral Interface (SPI) with AMBA APB Interface," 2019 IEEE 9th Symposium on Computer Applications & Industrial Electronics (ISCAIE), 2019, pp. 55-59, doi: 10.1109/ISCAIE.2019.8743871.

[14] Cadence, “32-bit APB Serial Peripheral Interface (SPI) IP”, Cadence design system, Inc. 2014 [online] Available: https://ip.cadence.com/uploads/435/cdn-dsd-sys-design-ip-for-apb-spi-pdf

[15] J. Yang, Y. Xiao, D. Li, Z. Li, Z. Chen and P. Wan, "A Configurable SPI Interface Based on APB Bus," 2020 IEEE 14th International Conference on Anti-counterfeiting, Security, and Identification (ASID), 2020, pp. 73-76, doi: 10.1109/ASID50160.2020.9271704.

[16] Anand N, G. Joseph, S. S. Oommen and R. Dhanabal, "Design and implementation of a high speed Serial Peripheral Interface," 2014 International Conference on Advances in Electrical Engineering (ICAEE), 2014, pp. 1-3, doi: 10.1109/ICAEE.2014.6838431.

[17] Oudjida, A. K., Berrandjia, M. L., Tiar, R., Liacha, A., & Tahraoui, K. (2009). “FPGA implementation of I2C & SPI protocols: A comparative study”, 2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009), doi:10.1109/icecs.2009.5410881

[18] A. K. Oudjida, M. L. Berrandjia, A. Liacha, R. Tiar, K. Tahraoui and Y. N. Alhoumays, "Design and test of general-purpose SPI Master/Slave IPs on OPB bus," 2010 7th International Multi- Conference on Systems, Signals and Devices, 2010, pp. 1-6, doi: 10.1109/SSD.2010.5585592.

[19] M. B. Aykenar, G. Soysal and M. Efe, "Design and Implementation of a Lightweight SPI Master IP for Low Cost FPGAs," 2020 28th Signal Processing and Communications Applications Conference (SIU), 2020, pp. 1-4, doi: 10.1109/SIU49456.2020.9302434.

[20] N. b. Mohd Noor and A. Saparon, "FPGA implementation of high speed serial peripheral interface for motion controller," 2012 IEEE Symposium on Industrial Electronics and Applications, 2012, pp. 78-83, doi: 10.1109/ISIEA.2012.6496676.

[21] S. Saha, M. A. Rahman and A. Thakur, "Design and implementation of SPI bus protocol with Built-in-self-test capability over FPGA," 2014 International Conference on Electrical Engineering and Information & Communication Technology, 2014, pp. 1-6, doi: 10.1109/ICEEICT.2014.6919076.

[22] R. R. Pahlevi, A. G. Putrada S. and M. Abdurohman, "Fast UART and SPI Protocol for Scalable IoT Platform," 2018 6th International Conference on Information and

Communication Technology (ICoICT), 2018, pp. 239-244, doi: 10.1109/ICoICT.2018.8528745.

[23] P. S. Mutha and Y. M. Vaidya, "FPGA reconfiguration using UART and SPI flash," 2017 International Conference on Trends in Electronics and Informatics (ICEI), 2017, pp. 59-63, doi: 10.1109/ICOEI.2017.8300765.

[24] J. V. E.V., P. V. and K. Balamurugan, "Design of Generic Verification Procedure for IIC Protocol in UVM," 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), 2019, pp. 1146-1150, doi: 10.1109/ICECA.2019.8821815.

[25] Z. Zhou, Z. Xie, X. Wang and T. Wang, "Development of verification envioronment for SPI master interface using SystemVerilog," 2012 IEEE 11th International Conference on Signal Processing, 2012, pp. 2188-2192, doi: 10.1109/ICoSP.2012.6492015.

[26] Y P. Jain and S. Rao, "Design and Verification of Advanced Microcontroller Bus Architecture-Advanced Peripheral Bus (AMBA-APB) Protocol," 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 462-467, doi: 10.1109/ICICV50876.2021.9388549.

[27] P. Dwivedi, N. Mishra and A. Singh-Rajput, "Assertion & Functional Coverage Driven Verification of AMBA Advance Peripheral Bus Protocol Using System Verilog," 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2021, pp. 1-6, doi: 10.1109/ICAECT49130.2021.9392518.

[28] Y. Guo et al., "A SPI Interface Module Verification Method Based on UVM," 2020 IEEE International Conference on Information Technology,Big Data and Artificial Intelligence (ICIBA), 2020, pp. 1219-1223, doi: 10.1109/ICIBA50161.2020.9277156.

[29] P. Gurha and R. R. Khandelwal, "SystemVerilog Assertion Based Verification of AMBA-AHB," 2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE), 2016, pp. 641-645, doi: 10.1109/ICMETE.2016.67.

[30] B. Vineeth and B. B. Tripura Sundari, "UVM Based Testbench Architecture for Coverage Driven Functional Verification of SPI Protocol," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2018, pp. 307-310, doi: 10.1109/ICACCI.2018.8554919.

[31] Dipti Girdhar Shankar and Neeraj Kr. Shukla "Design and Verification of AMBA APB Protocol" International Journal of Computer Applications vol. 95 no. 21 June 2014.

[32] R.K Vaishnavi S Bindu and Sheik Chandbasha "Design and Verification of APB Protocol by using System Verilog and Universal Verification Methodology" International Research Journal of Engineering and Technology (IRJET) vol. 06 no. 06 pp. 23950072 2019.

[33] D. Trivedi, A. Khade, K. Jain and R. Jadhav, "SPI to I2C Protocol Conversion Using Verilog," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1-4, doi: 10.1109/ICCUBEA.2018.8697415.

[34] L. Huihui, "Design and Realization of SPI Driver Based on VxWorks," 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2020, pp. 200-205, doi: 10.1109/ITNEC48623.2020.9084665.

[35] Ge Zhiwei, Yao Suying and Xu Jiangtao, "Design of on-chip image processing based on APB bus with CMOS image sensor," 2009 IEEE 8th International Conference on ASIC, 2009, pp. 963-966, doi: 10.1109/ASICON.2009.5351535.

[36] A. Suresh and S. Nandi, "Physical Bridge Design for High Performance to Peripheral Bus: An Open Source Approach," 2020 Third International Conference on Advances in Electronics, Computers and Communications (ICAECC), 2020, pp. 1-5, doi: 10.1109/ICAECC50550.2020.9339522.