Project Report

On

DMA Controller using SPI

Submitted in partial fulfillment for the award of

Post Graduate Diploma in VLSI Design (PG-DVLSI)

From

C-DAC, ACTS (Pune)

Guided by:

Mr. Ankur Vijay

Presented by:

Mr. Gollu Pavankumar	PRN: 200240133009
Mr. Jayesh Thakre	PRN: 200240133012
Mr. Mohite Karan Jitendra	PRN: 200240133016
Mr. Puneet Kumar	PRN: 200240133020
Ms. Takale Vishakha Balasaheb	PRN: 200240133027

Centre for Development of Advanced Computing (C-DAC), Pune.

ACKNOWLEDGEMENT

This project "DMA Controller using SPI" was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS). We are very thankful to Mr. Ankur Vijay for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

Our heartfelt thanks goes to Mrs. Seema Sanjeev (Course Coordinator, PG-DVLSI C- DAC ACTS,Pune) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

From:

Mr. Pawan Gollu PRN: 200240133009

Mr. Jayesh Thakre PRN: 200240133012

Mr. Karan Mohite PRN: 200240133016

Mr. Puneet Kumar PRN: 200240133020

Ms. Vishakha Takale PRN: 200240133027

TABLEOFCONTENTS

ABSTRACT

In this project, the design and verification of Direct Memory Access Controller is proposed using Verilog and system verilog . Direct memory access is a feature of modern computers that allows certain hardware within the computer to access system memory for reading or writing independently of the CPU. CPU that have DMA channels can transfer data to and from devices with much smaller CPU load than computers without a DMA channel.
SPI bus is use for interfacing the memory and input output devices.

Key features-

- Address are 8 bits

  Data line are 8 bits

  Data count are 4 bits

  Maximum operating frequency for DMA 3 MHz

  Maximum operating frequency for SPI Communication 1.5 MHz

Chapter 1: Introduction to the DMA Controller

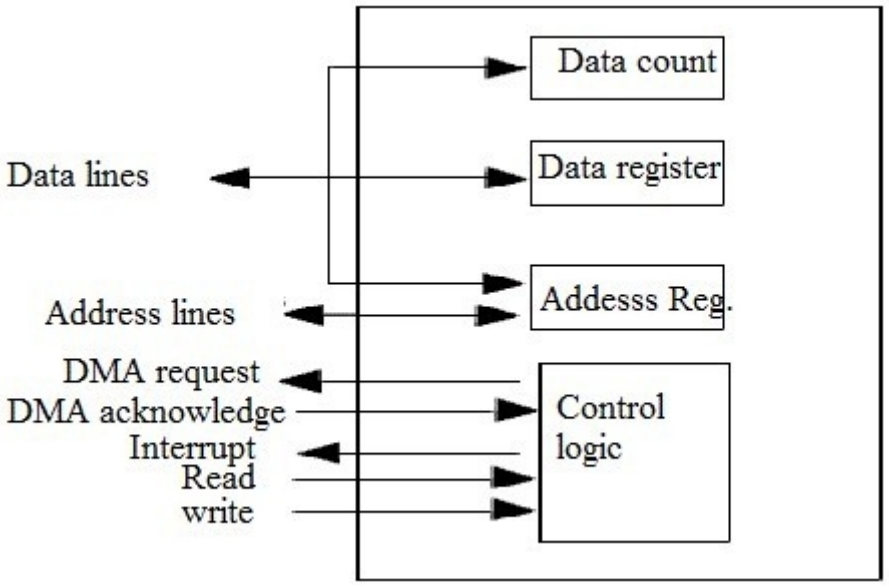
1.1. What is a DMAController

DMA stands for direct memory access. The hardware device used for direct memory access is called the DMA controller. DMA controller is a control unit, part of I/O device's interface circuit, which can transfer blocks of data between I/O devices and main memory without involving the

DMA controller provides an interface between the bus and the input-output devices. Although it transfers data without intervention of processor, it is controlled by the processor.

The processor initiates the DMA controller by sending the data, i.e. from I/O devices to the memory or from main memory to I/O devices. More than one external device can be connected to the DMA controller.

DMA controller contains an address unit, for generating addresses and selecting I/O device for transfer. It also contains the control unit and data count for keeping counts of the number of blocks transferred and indicating the direction of transfer of data. When the transfer is completed, DMA informs the processor by raising an interrupt. The typical block diagram of the DMA controller is shown in the figure below.



Typical Block Diagram of DMA Controller

Working of DMA:

DMA controller has to share the bus with the processor to make the data transfer. The device that holds the bus at a given time is called bus master. When a transfer from I/O device to the memory or vice verse has to be made, the processor stops the execution of the current program, increments the program counter, moves data over stack then sends a DMA select signal to DMA controller over the address bus.

If the DMA controller is free, it requests the control of bus from the processor by raising the bus request signal. Processor grants the bus to the controller by raising the bus grant signal, now DMA controller is the bus master. The processor initiates the DMA controller by

sending the memory addresses, number of blocks of data to be transferred and direction of data transfer. After assigning the data transfer task to the DMA controller, instead of waiting

ideally till completion of data transfer, the processor resumes the execution of the program after retrieving instructions from the stack.

DMA controller now has the full control of buses and can interact directly with memory and I/O devices independent of CPU. It makes the data transfer according to the control instructions received by the processor. After completion of data transfer, it disables the bus request signal and CPU disables the bus grant signal thereby moving control of buses to the CPU.
When an I/O device wants to initiate the transfer then it sends a DMA request signal to the DMA controller, for which the controller acknowledges if it is free. Then the controller requests the processor for the bus, raising the bus request signal. After receiving the bus grant signal it transfers the data from the device. For n channeled DMA controller n number of external devices can be connected.

The DMA transfers the data in three modes which include the following.
a) Burst Mode: In this mode DMA handover the buses to CPU only after completion of whole data transfer. Meanwhile, if the CPU requires the bus it has to stay ideal and wait for data transfer.
b) Cycle Stealing Mode: In this mode, DMA gives control of buses to CPU after transfer of every byte. It continuously issues a request for bus control, makes the transfer of one byte and returns the bus. By this CPU doesn't have to wait for a long time if it needs a bus for higher priority task.

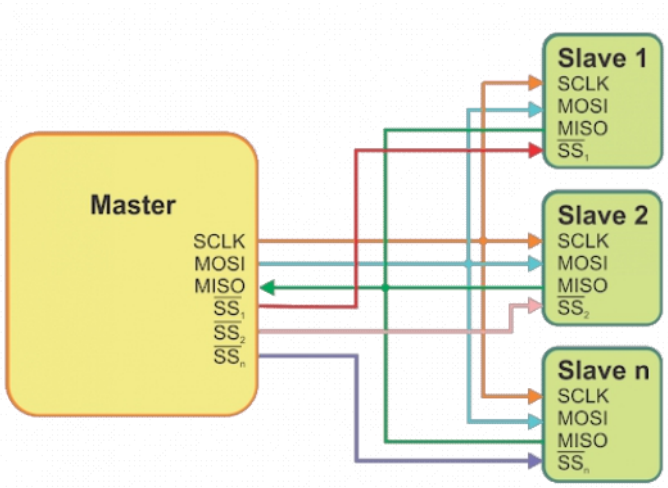
Chapter 2: Introduction to the SPI

2.1. What is SPI

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS), sometimes called chip select (CS), lines.

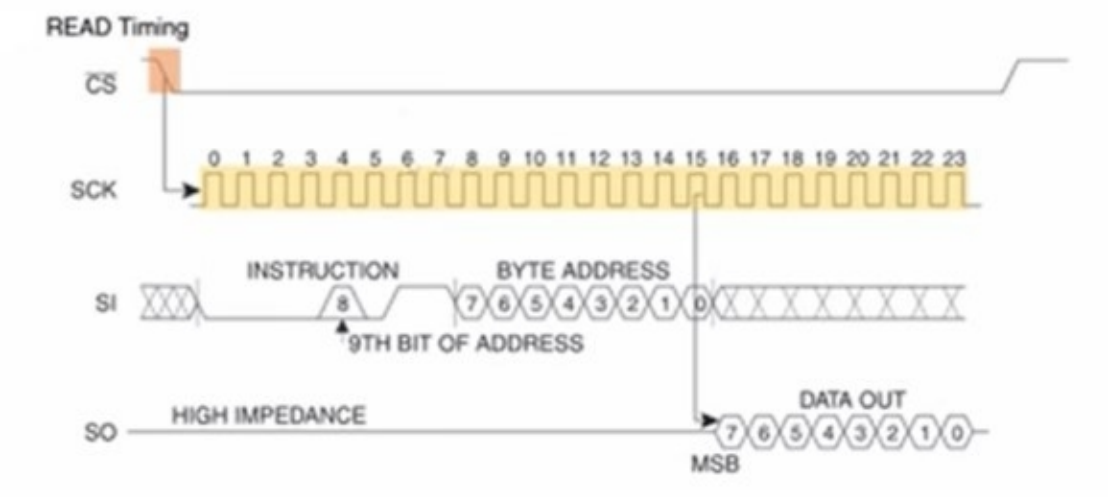
Sometimes SPI is called a four-wire serial bus. The SPI may be accurately described as a synchronous serial interface. SPI is one master and multi slave communication.

- SCLK: Serial Clock (output from master)
- MOSI: Master Out Slave In (data output from master)
- MISO: Master In Slave Out (data output from slave)
- SS: Slave Select (often active low, output from master)

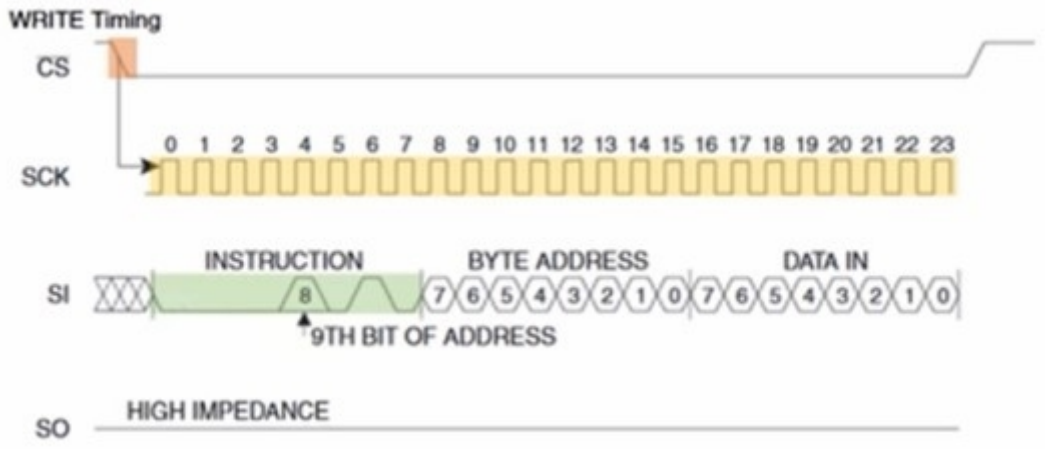


SPI mode	Clock polarity (CPOL)	Clock phase (CPHA)	Clock edge (IDEL CLK)
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

2.2 SPI MASTER to SLAVE communication
MASTER DMA --- SLAVE Memory
SPI master and slave consist of 8bit shift register
Master send read instruction and address to slave memory through MOSI and expecting data out from memory through MISO



Master send write instruction, address and data to slave memory through MOSI

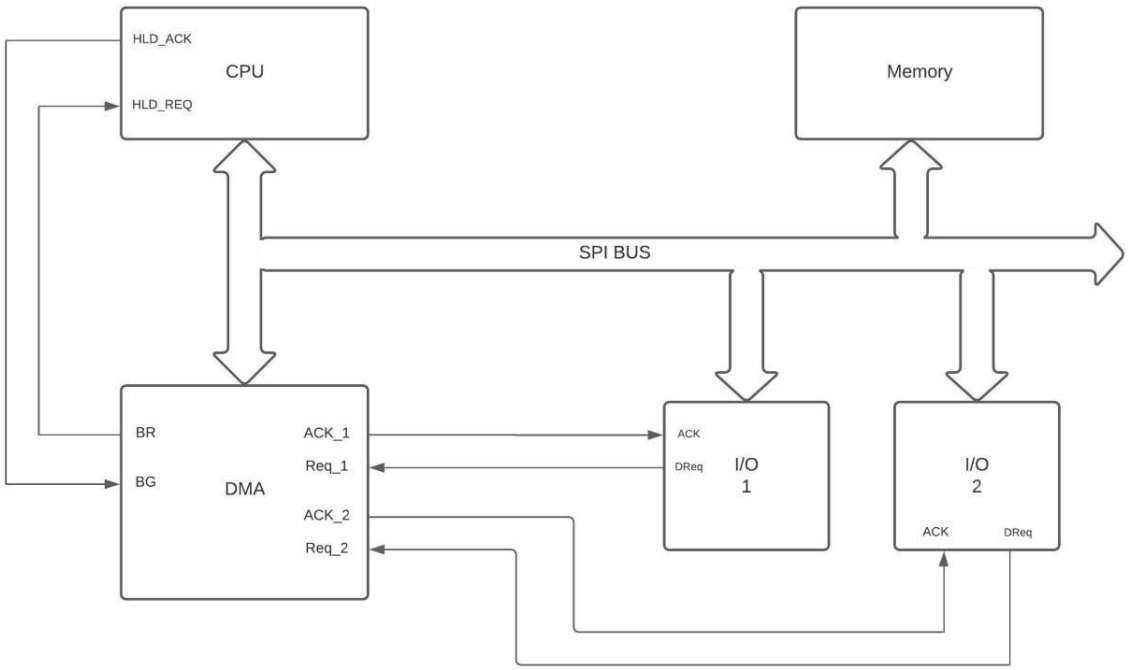


Chapter 3: Literature Survey

The idea or technique of making this project to reduce load on CPU by using DMA controller, The term DMA stands for direct memory access. The hardware device used for direct memory access is called the DMA controller. In this project we have used SPI Bus for establish communication I/O to memory.

Reference	Inference
DMA CONTROLLER 8257 Author - R.Kavitha	DMA Controller DMA Operation Mode DMA option for data transfer
DESIGN AND IMPLEMENTATION OF SPI PROTOCOL Author - Kamal Prakash Pandey, ApoorvUpadhyay	SPI protocol SPI Modes SPI Master Slave Communication

Chapter 4: Design



4.2 Module Description

4.2.1 DMA Master Design

Signals	Type	Size	Description
MISO	input	1	Master In Slave Out (SPI)
MOSI	output	1	Master out Slave in (SPI)
clk	input	1	Clock
rst	input	1	Reset
Dack	output	4	DMA acknowledge for I/O device
BGNT/HLDA	input	1	Bus grant/Hold request ack
Data	input	8	Data line
BREQ/HRQ	output	1	Bus request/Hold request
Dreq	input	4	I/O device request to DMA
DS	input	2	Data select
SS	output	5	Slave select
Dma_busy	Output	1	Busy signal
Dma_done	Output	1	Operation done signal

Siganls - HLDA, HRQ, data, Dreq, Dack, DS

Control logic and mode set registers

Setting the values of control registers using data select(ds) and data lines

-Address

-word count register

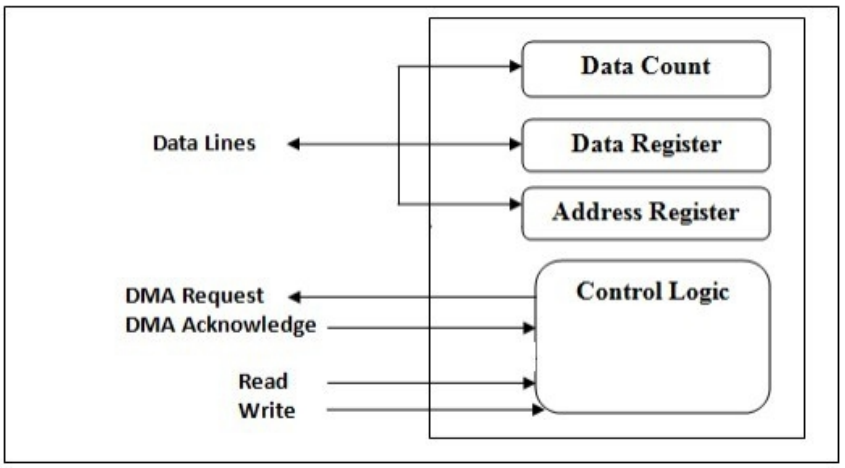
-Channel select register

-Mode set registers

Controlling and handling the bus request and bus grant

Handling the DMA request from IO device

Handling the DMA acknowledge for IO device



HRQ(Hold request) - Bus request

HLDA(Hold Acknowledge) - Bus grant

- when any IO device wants to send data to memory, then device has to send DMA request (DRQ) to DMA controller.
- The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert the HLDA.
- Now this above two process will cause two cycle to complete.
- CPU will set the value for IOW, IOR, MR, MW signal
- Now before giving bus grant to DMA, CPU will send value like data count, Address register, Mode set register and channel set register through 8 bit data line and DS(data select) will use for selecting the registers present inside DMA.

Address register - 8bit

Data count register - 4 bit

Data Select	register
00	address
01	data count
10	Channel select
11	Mode set register

Mode set register

x	x	x	x	IOR	IOW	MR	MW

IOR	IOW	MR	MW	Result or Operation
1	0	0	1	Read from IO device and write into memory
0	1	1	0	Read from memory and write into IO device

For writing information in above register - 10 clock cycle s required
- Then CPU provide bus grant(HLDA) to DMA controller. - 1 clock cycle
- After that DMA send acknowledge to respective IO device and then read write Operation can begin. - 1 clock cycle

If there are multiple DMA request from different IO channel then DMA will only acknowledge that IO channel that is granted by priority resolver.

Channel select register

x	x	x	en	Ch3	Ch2	Ch1	Ch0

Ch0 - enable channel 0
Ch1 - enable channel 1
Ch2 - enable channel 2
Ch3 - enable channel 3
En - rotating priority enable

1. Read write unit

Signals - mode_set {IOW,IOR,MW,MR}, SS(slave select),MISO and MOSI
Shift_in and shift_out register
Address and word count register.
Address = starting memory address
Read write operations

A. {IOW,IOR,MW,MR} = 1001 => memory read IO write

Memoryslavewillselectbymaster

- DMA send read instruction and address to memory using shift_out register of SPI master then get return data using shift_in register and store it into the fifo data buffer.
- Above process will repeat till the word count and address will increment every time whenever the one read operation is done.

IOdevicewillselectbymaster

- Master will send write instruction and data store inside the FiFo data buffer to IO device.

The above write operation will repeat until data buffer get empty or till count value.

B. {IOW,IOR,MW,MR} = 0110 => IO read memory write

IOdeviceslavewillselectbymaster

- DMA send read instruction to IO device using shift_out register then get return data using shift_in register and store it into the fifo data buffer.

Above operation will repeat till data count.

Memoryslavewillbeselectedbymaster.
-DMA sends write instruction , address and data that is store inside the fifo buffer to memory using shift_out register
-Above step will repeat till data count and address register will increment every time whenever one write operation is done.

4.2.2 Memory Slave Design

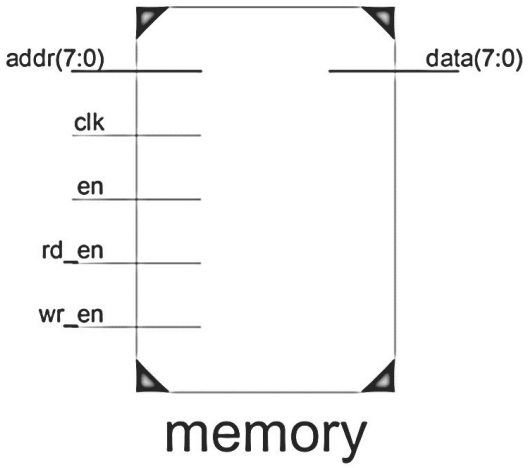
Signals	Type	Size	Description
clk	input	1	Clock
rst	input	1	Reset
MOSI	input	1	Master out slave in
MISO	output	1	Master in slave out
cs	input	1	Chip select(active low)

- Memory spi slave will consist of shift_in and shift_out register with shift done flag. These shift register can transmit or receiver 8bit value at time.
- Internal register - instruction(read/write), address and data
- Whenever cs is active then slave will start receiving command from DMA master via MOSI line using Shift_in register in instruction and address register.
- Depending on the instruction provide by master read and write operation will perform.
- if instruction receive is write then slave will again receive data from shift_in register and store it into the data register and write operation into memory will be perform.
- if instruction receive is read then read operation from memory will done and output data from memory will send via MISO line using shift_out registers.

Memory

Signals	Type	Size	Description
Wr_en	Input	1	Write enable
Rd_en	Input	1	Read enable
Clk	Input	1	Clock
Addr	Input	8	Address
En	Input	1	Chip enable
Data	Inout	8	Bidirectional Data

In the memory we are using 256 x 8 RAM memory module with one read/write enable & Chip enable. In this, we have used bidirectional i/o port.
That is depth x width = 8-bit x 256 locations;



wr_en (write enable)-

WRITE ENABLE signal when set high, indicates that you are writing to the RAM chip, and is normally low during reading.

rd_en (read enable)–

READ ENABLE signal when set high, indicates that you are reading from RAM chip, and is normally low during writing.

Addr (address)-

it is 8 bit size address bus, it indicates working location address of RAM (8-bit)

en (enable) –

en (Chip Enable) is a memory select signal and the memory is selected when it is set at high level. This is used to assign a specific address by adding an address-decoded signal.

Data (bidirectional data)–

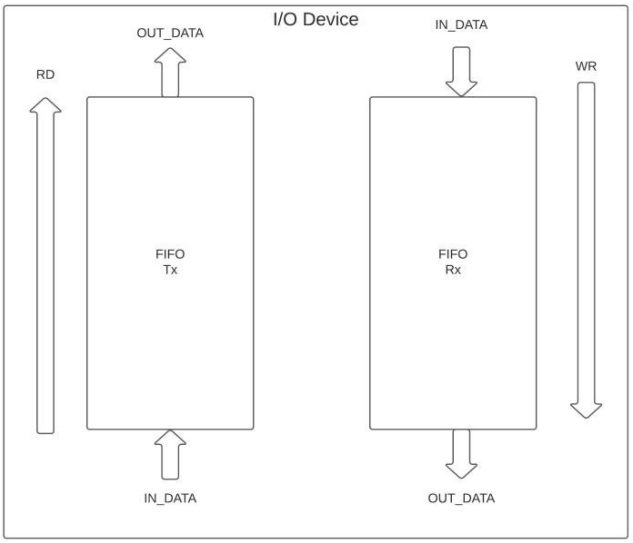
It is bidirectional 8-bit i/o bus which used for data exchange from RAM to other components.

en	wr_en	rd_en	Action
0	X	X	Since chip enable is low no operation gets performed.
1	1	0	Write
1	0	1	Read
X	1	1	No action

If en = 1 then chip enable else no any operation gets perform,
For write, rd_en = 0 & wr_en = 1 & en = 1;
Then Data write into memoey in specified location from i/o port,
For read, rd_en = 1 & wr_en = 0 & en = 1;
Then Data gets read from memory using 8-bit address & given to i/o port

4.2.3 IO Device Slave Design

Signals	Type	Size	Description
clk	input	1	Clock
MOSI	Input	1	Master out slave in
MISO	Output	1	Master in slave out
Cs	Input	1	Chip select
Wreq	Input	1	Write request
Indata_tx	input	8	Input data of transmitter
Outdata_rx	output	8	Output data of transmitter
Dreq	output	1	DMA request
Rd_en_rx	input	1	Read enable of receiver
Dack	input	1	DMA acknowledge



- IO device slave will consist of shift_in, shift_out, transmitter and receiver fifo register with shift done flag.
These shift register can transmit or receiver 8bit value at time.
- Internal register - instruction(read/write), address and data
- Whenever cs is active then slave will start receiving command from DMA master via MOSI line using Shift_in register in instruction and address register.
- Depending on the instruction provide by master read and write operation will perform.
- if instruction receive is write then slave will again receive data from shift_in register and store it into the data register and write operation into IO device will be perform.
- if instruction receive is read then read operation from memory will done and output data from IO device will send via MISO line using shift_out registers.

Input can be provide by enabling the write enable and indata of of transmitter fifo.
Output can observe by enabling the read enable and outdata of of receiver fifo.

4.2.4 Priority Resolver Design

In Priority resolver some signals given below which are used to connect and communicate with the DMA and SPI.

No	Signals	Type	Size	Description
1	reqA	input	1	Request A
2	reqB	input	1	Request B
3	reqC	input	1	Request C
4	reqD	Input	1	Request D
5	clk	input	1	clock
6	rst	Input	1	Reset
7	Rot_en	input	1	Rotating priority enable
8	gntA	output	1	Grant A
9	gntB	Output	1	Grant B
10	gntC	Output	1	Grant C
11	gntD	Output	1	Grant D
12	Busy	Output	1	Busy flag
13	en	Input	1	Priority enable

A Priority resolver means giving priority to a one bit if two or more bits meet the criteria.
We are using two type of priority resolver:-

1. Fixed priority resolver
2. Rotating priority resolver

In the Fixed priority resolver all the inputs are assign a fixed priority reqA higher priority and reqD lower priority. And after completing the function/cycles it should be same as above.

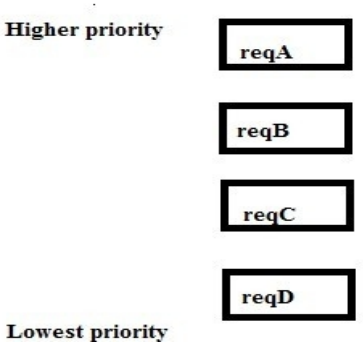


Fig1: Fixed priority

In the Rotating priority, priority assign channel not fix its rotate. Suppose reqA is connected and its function is completed then it should go lowest position, Now reqB will become higher priority and reqA become lowest priority, and this process go until we not get reqA as highest priority.

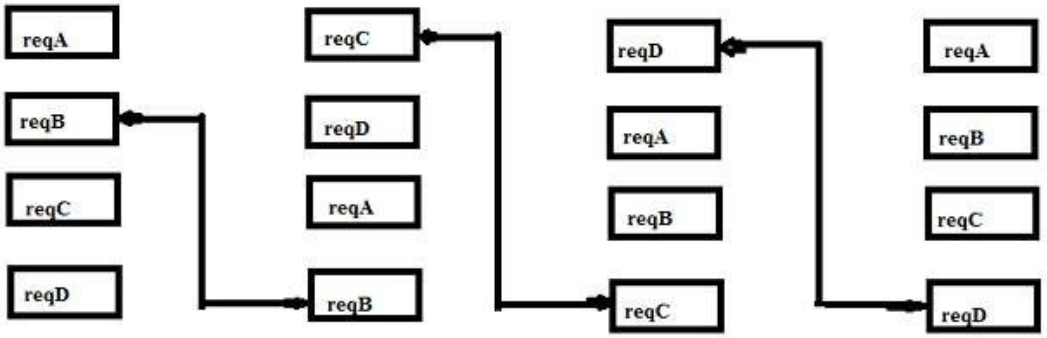


Fig 2: Rotating resolver

Rot_en stand for rotating enable, when Rot_en will high then rotating priority will work and when Rot_en is low then fixed priority will work.
We are using asynchronous negative edge. In this at time of active low reset will work.
Busy signal is used to check our priority resolver working or free. Means if priority resolver is working then busy signal should be high and that time if any grant received by DMA that will be invalid. And if priority resolver is free then busy signal will low and ready to accept the grant.

3.2.5 8bit FIFO Design

8 bit data buffer is also called as FIFO (First In First Out), which describe how data is managed relative to time on priority. In this case, the First data that arrives will also be the first data to leave from a group of data. A FIFO Buffer is a read/write memory array that automatically keep track of the order in which data enter into the module and read the data out in the same order. Read and write addresses are initially both at the first memory location and the FIFO queue is Empty. When the difference between the read address and write address of the FIFO buffer is equal to the size of the memory (8 bit) of the memory array then the FIFO queue is FULL.

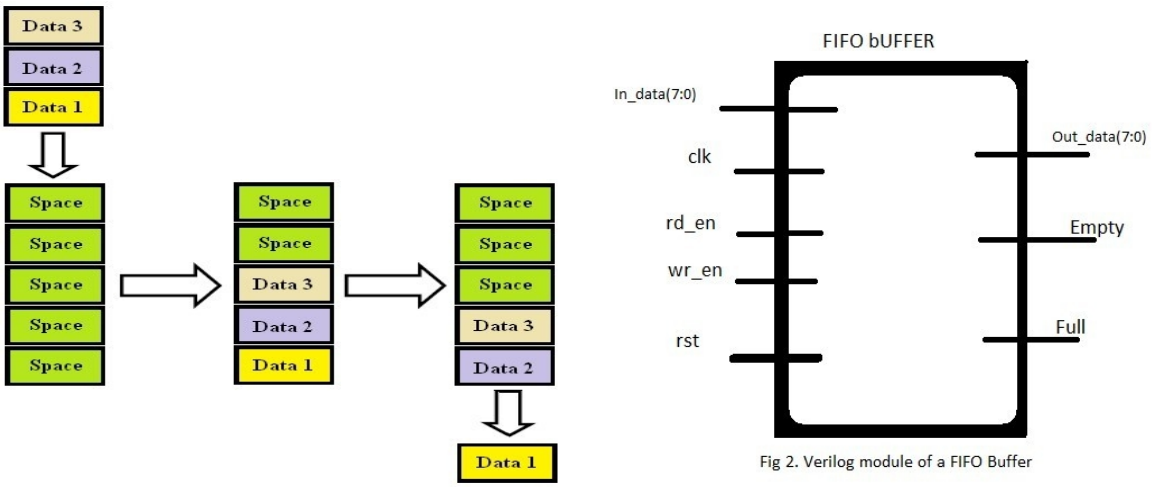


Fig 2. Verilog module of a FIFO Buffer

Operation of FIFO with the three data value.

FIFO can be classified as Synchronous and Asynchronous depending on whether same clock (Synchronous) or different clocks(Asynchronous) control the read and write operation.
A Synchronous FIFO refers to a FIFO design where data value are written sequentially into a memory array using a clock signal and the data value are read out sequentially from the memory array using the same clock signal. As shown in fig 1. The flow of the operation of FIFO.

4.2.6 Shift_out register

Signals	Type	Size	Description
rst	input	1	Reset
clk	input	1	clock
Shift_en	input	1	Shift enable
Indata	Input	8	Input data
dout	Output	1	Serial data out
done	Output	1	Done flag

-Shift_out register will use to send data which consist of 8 bit registers
When sample enable is high then input value will be assign to 8 bit internal register.
-when shift enable is and sample is low then 8 time shifting will be done on the register and data on MSB of register will be assign to dout.

D7	D6	D5	D4	D3	D2	D1	D0

Left shifting by 1 on above register will be done at every clock cycle.
Value at D7 will assign to dout.
Done flag will high if shifting is done.

3.2.7 Shift_in register

Signals	Type	Size	Description
rst	input	1	Reset
clk	input	1	clock
Shift_en	input	1	Shift enable
din	Input	1	Serial data input
dataout	Output	8	Output data
done	Output	1	Done flag

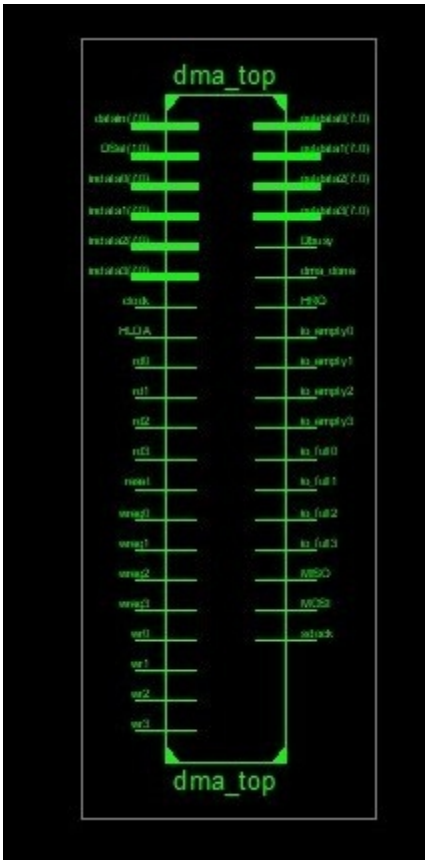
Shift_in register consist of 8 bit register.
When shift enable is high then value from din will shift from LSB on every clock pulse 8 time.

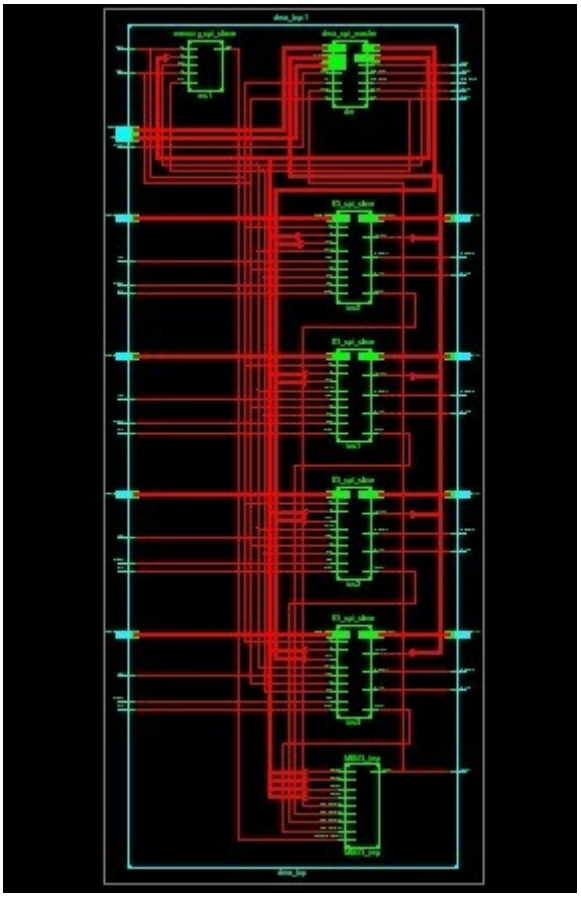
D7	D6	D5	D4	D3	D2	D1	D0

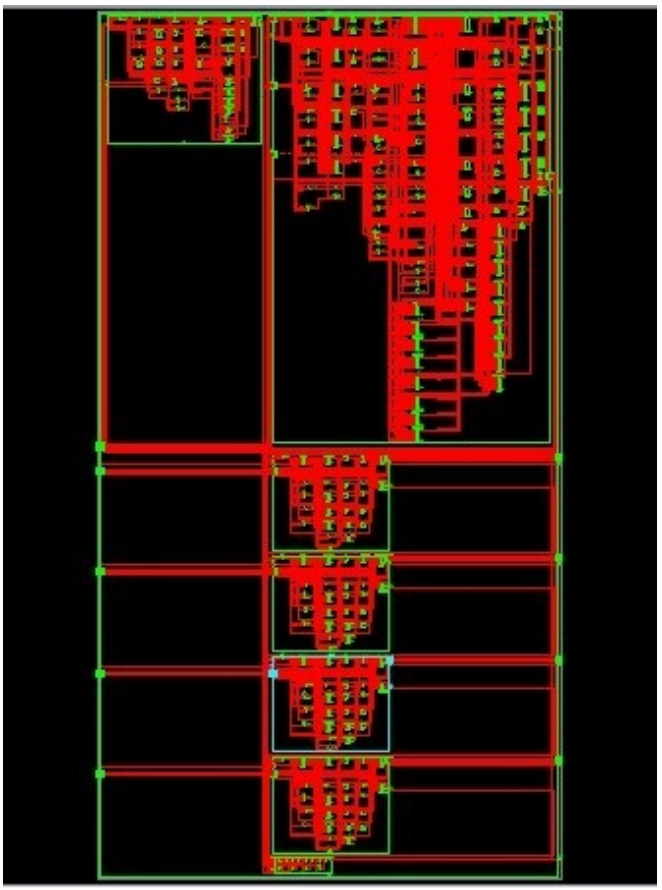
Din will assign to D0 ,
Then Left shifting by 1 on above register will be done at every clock cycle.
Done flag will high if shifting is complete

Chapter 5: RTL :

5.1 RTL Schematic







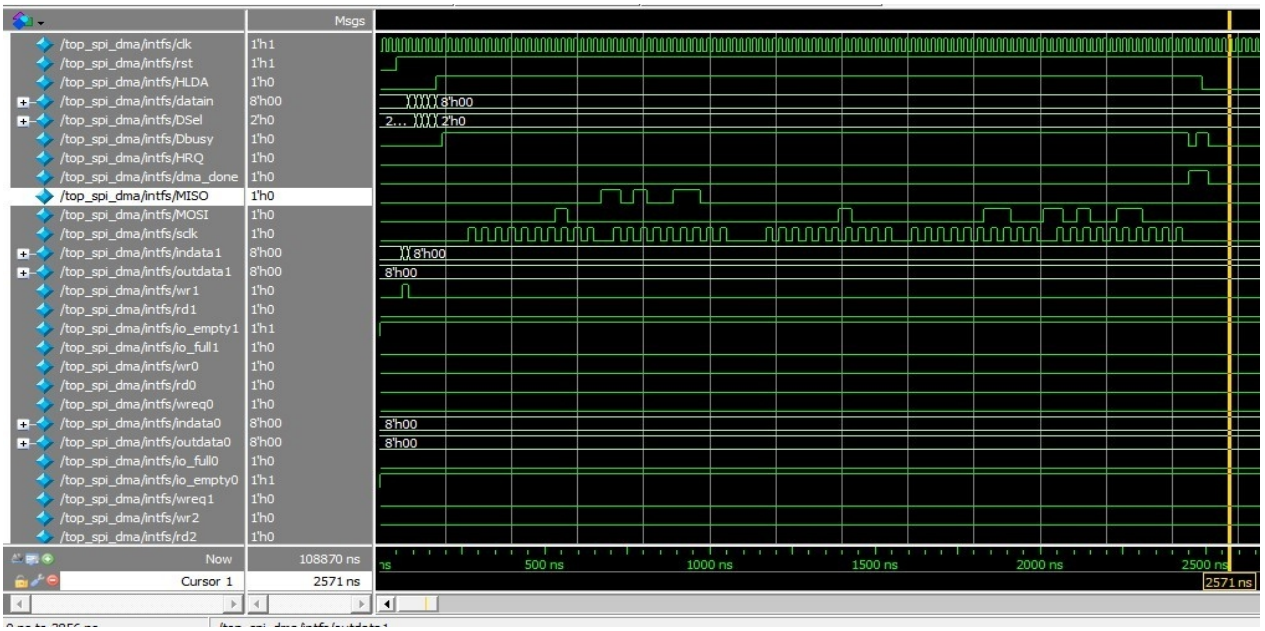
5.2 RTL Technology Schematic

Chapter 6: VERIFICATION

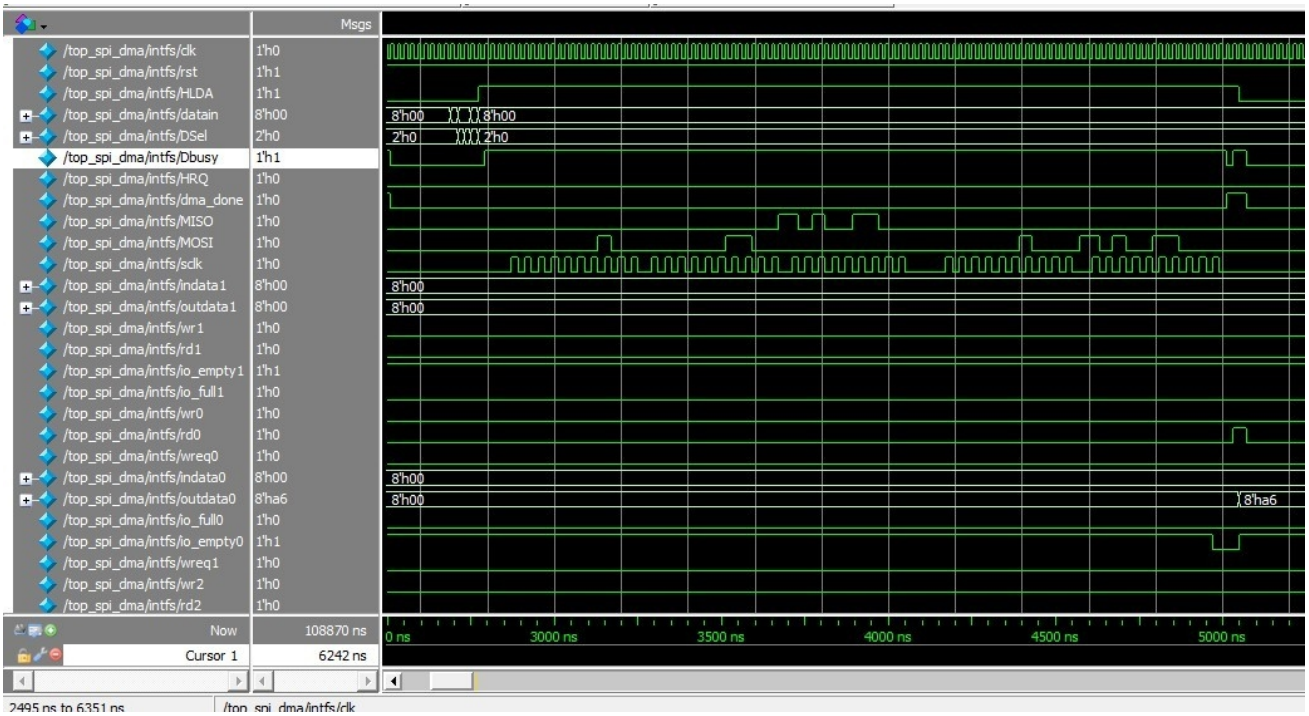
6.1 UVM Test bench environment

Test list
1. Single data Read from IO device and write into memory
2. Single data Read from memory and write into IO device
3. Burst data Read from IO device and write into memory
4. Burst data Read from memory and write into IO device
5. Provide multiple IO device request to DMA
6. Provide multiple IO device request to DMA with rotating priority enable

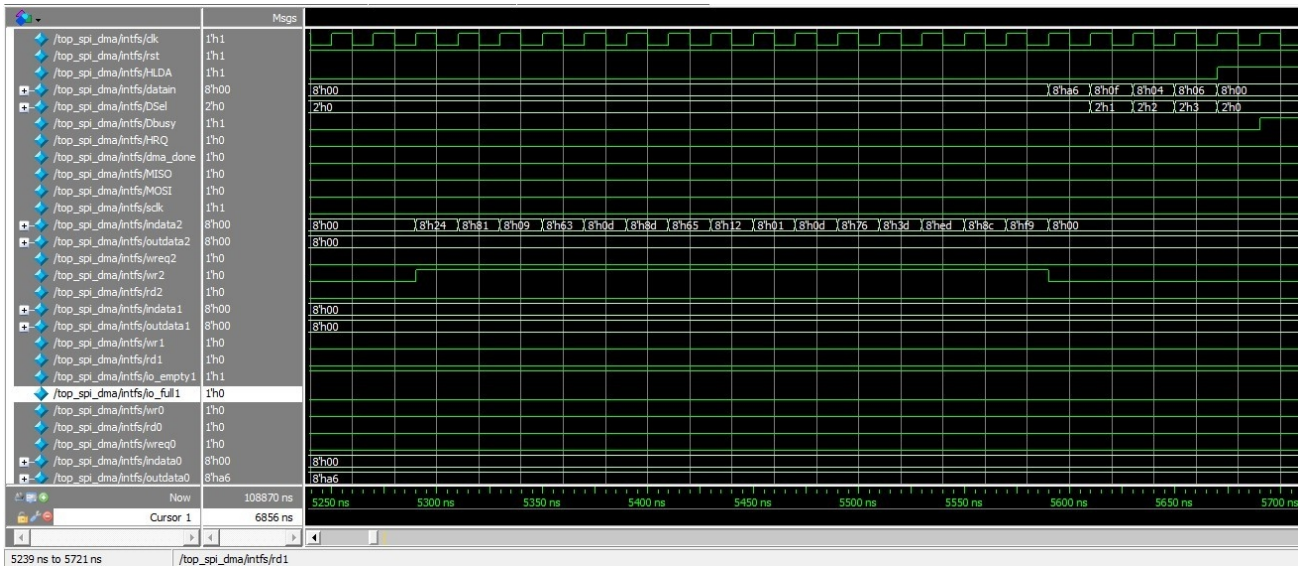
SIMULATION in QuestaSim
1. Single data Read from IO device and write into memory

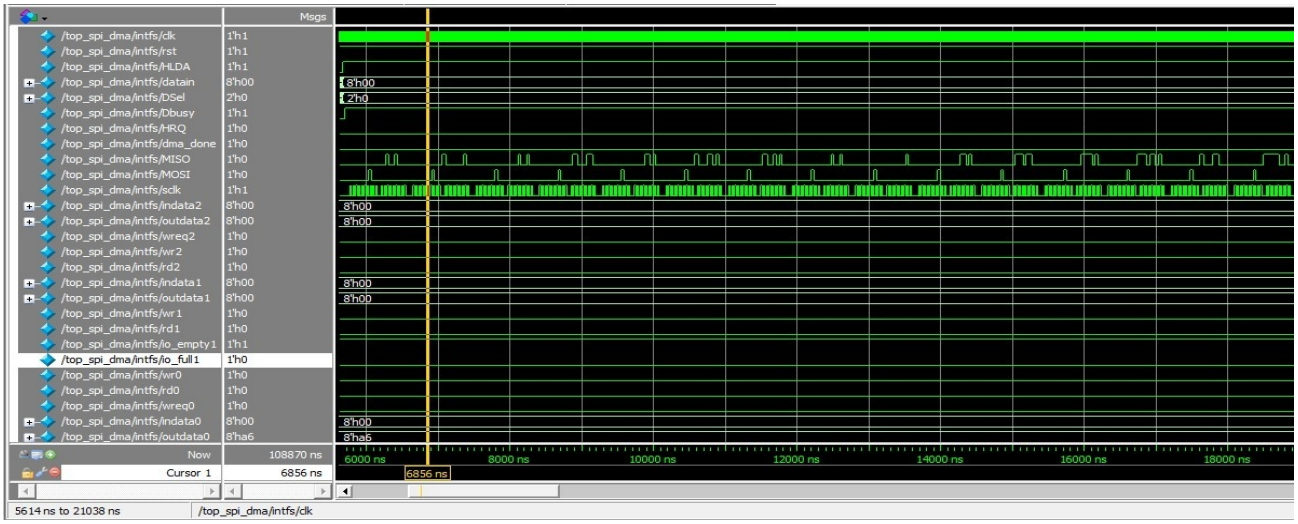


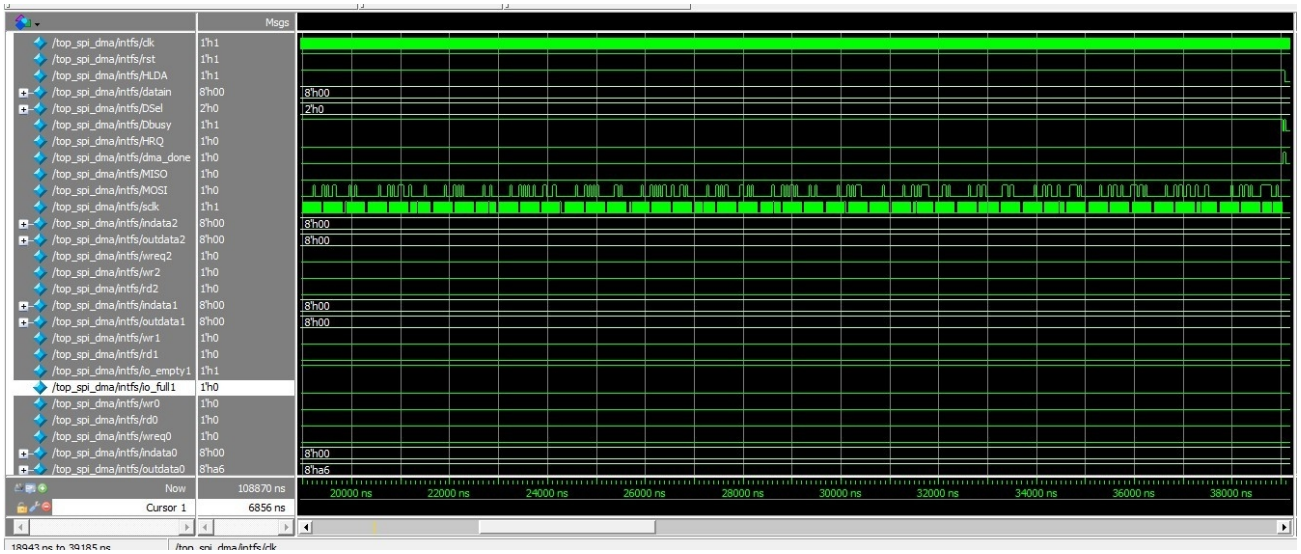
2.Single data Read from memory and write into IO device



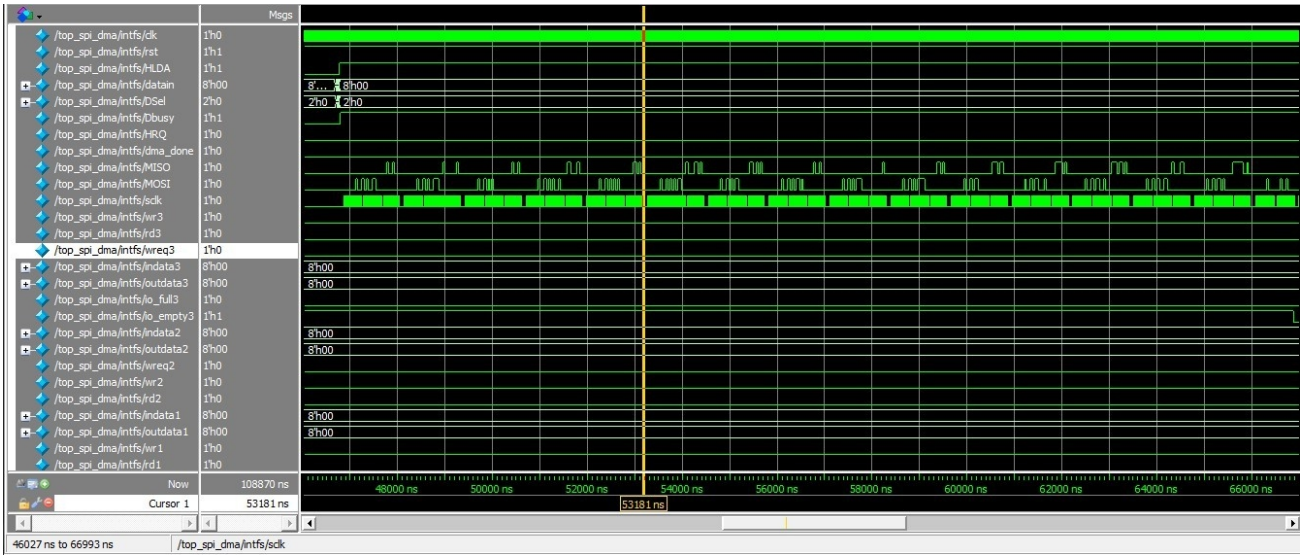
2. Burst data Read from IO device and write into memory

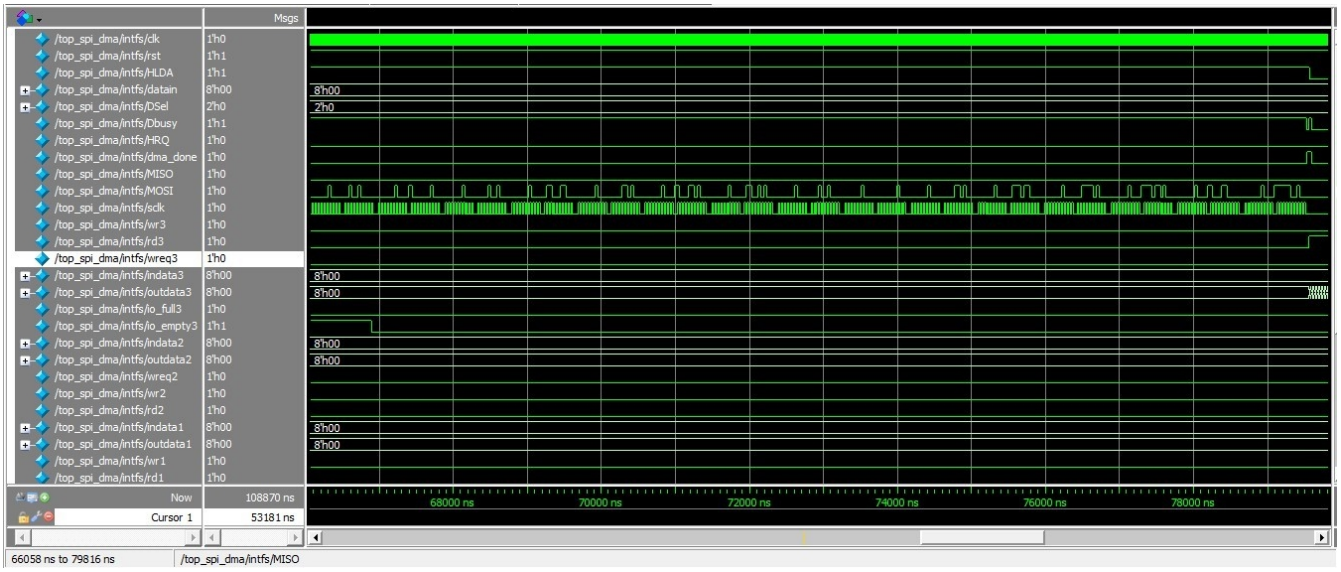


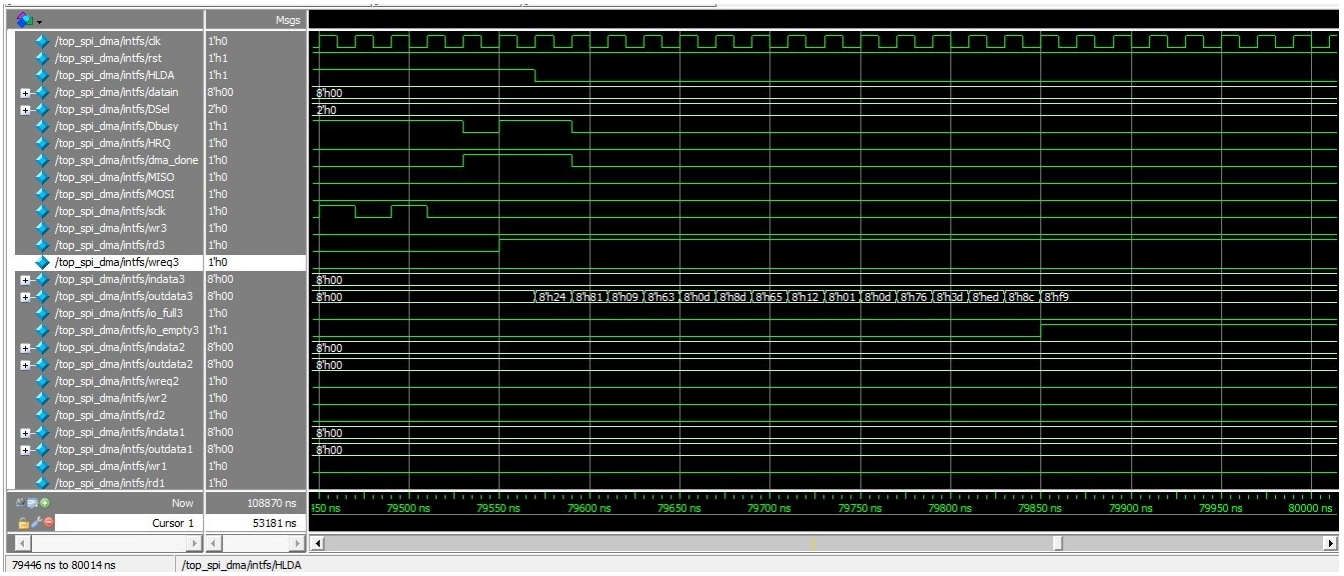




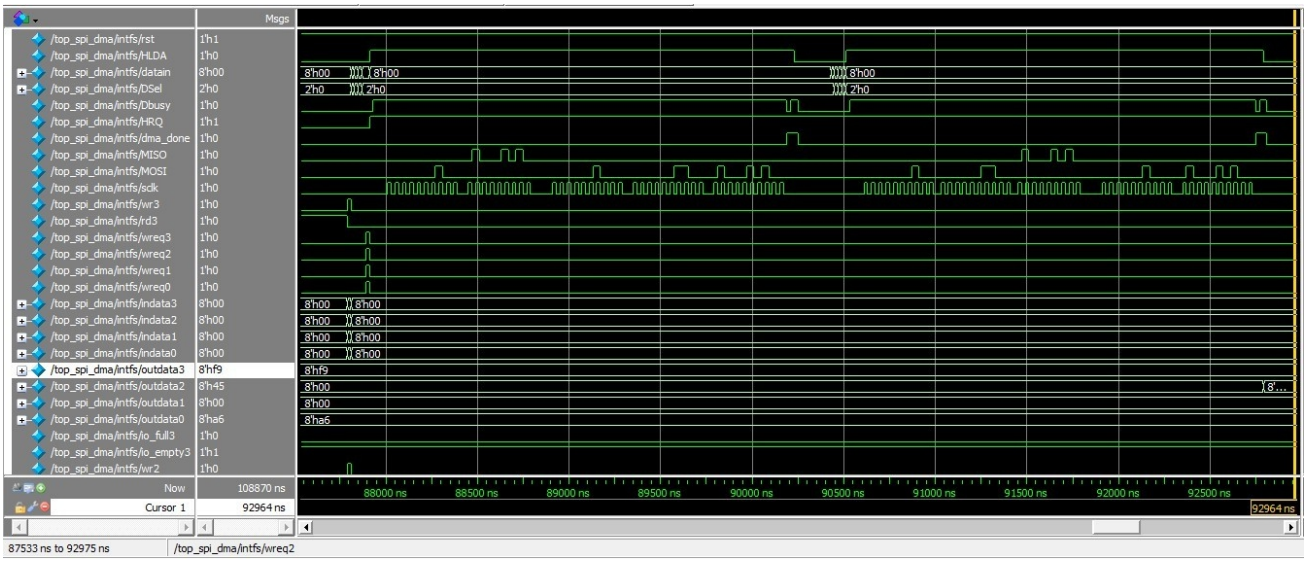
4.Burst data Read from memory and write into IO device



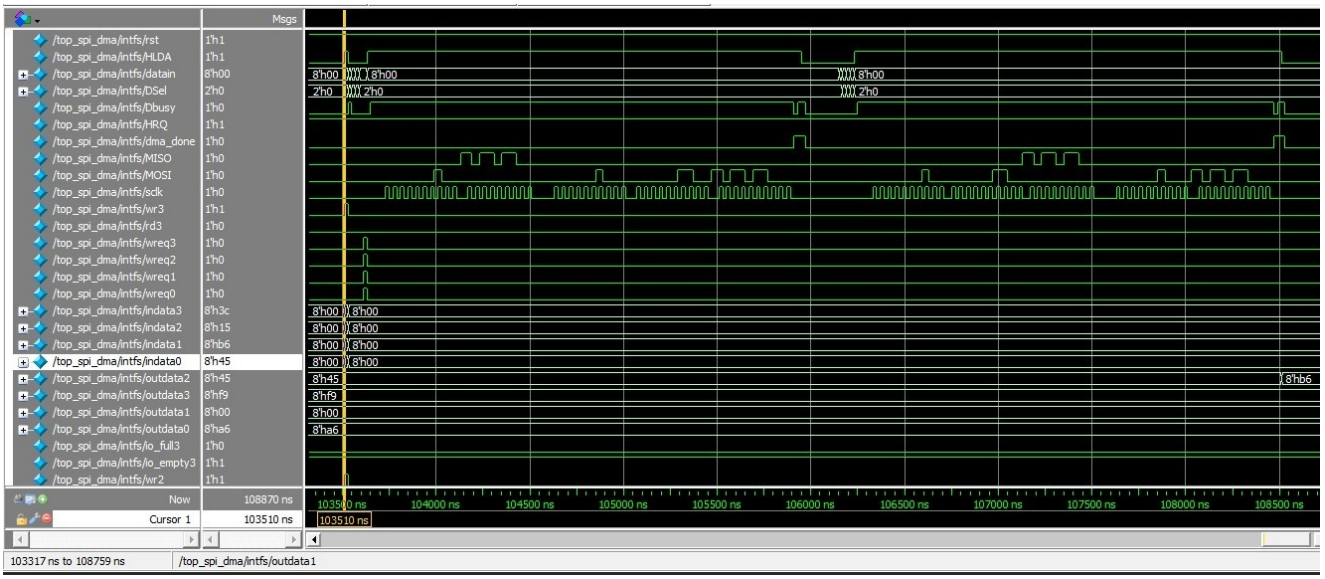

Msgs
/top_spi_dma/intfs/clk
/top_spi_dma/intfs/rst
/top_spi_dma/intfs/HLDA
/top_spi_dma/intfs/datain
/top_spi_dma/intfs/DSel
/top_spi_dma/intfs/Dbusy
/top_spi_dma/intfs/HRQ
/top_spi_dma/intfs/dma_done
/top_spi_dma/intfs/MISO
/top_spi_dma/intfs/MOSI
/top_spi_dma/intfs/sclk
/top_spi_dma/intfs/wr3
/top_spi_dma/intfs/rd3
/top_spi_dma/intfs/wreq3
/top_spi_dma/intfs/indata3
/top_spi_dma/intfs/outdata3
/top_spi_dma/intfs/io_full3
/top_spi_dma/intfs/io_empty3
/top_spi_dma/intfs/indata2
/top_spi_dma/intfs/outdata2
/top_spi_dma/intfs/wreq2
/top_spi_dma/intfs/wr2
/top_spi_dma/intfs/rd2
/top_spi_dma/intfs/indata1
/top_spi_dma/intfs/outdata1
/top_spi_dma/intfs/wr1
/top_spi_dma/intfs/rd1
Now
108870 ns
Cursor 1
53181 ns
68000 ns
70000 ns
72000 ns
74000 ns
76000 ns
78000 ns
66058 ns to 79816 ns
/top_spi_dma/intfs/MISO


Msgs
/top_spi_dma/intfs/clk
/top_spi_dma/intfs/rst
/top_spi_dma/intfs/HLDA
/top_spi_dma/intfs/datain
/top_spi_dma/intfs/DSel
/top_spi_dma/intfs/Dbusy
/top_spi_dma/intfs/HRQ
/top_spi_dma/intfs/dma_done
/top_spi_dma/intfs/MISO
/top_spi_dma/intfs/MOSI
/top_spi_dma/intfs/sclk
/top_spi_dma/intfs/wr3
/top_spi_dma/intfs/rd3
/top_spi_dma/intfs/wreq3
/top_spi_dma/intfs/indata3
/top_spi_dma/intfs/outdata3
/top_spi_dma/intfs/io_full3
/top_spi_dma/intfs/io_empty3
/top_spi_dma/intfs/indata2
/top_spi_dma/intfs/outdata2
/top_spi_dma/intfs/wreq2
/top_spi_dma/intfs/wr2
/top_spi_dma/intfs/rd2
/top_spi_dma/intfs/indata1
/top_spi_dma/intfs/outdata1
/top_spi_dma/intfs/wr1
/top_spi_dma/intfs/rd1
Now
108870 ns
Cursor 1
53181 ns
79446 ns to 80014 ns
/top_spi_dma/intfs/HLDA

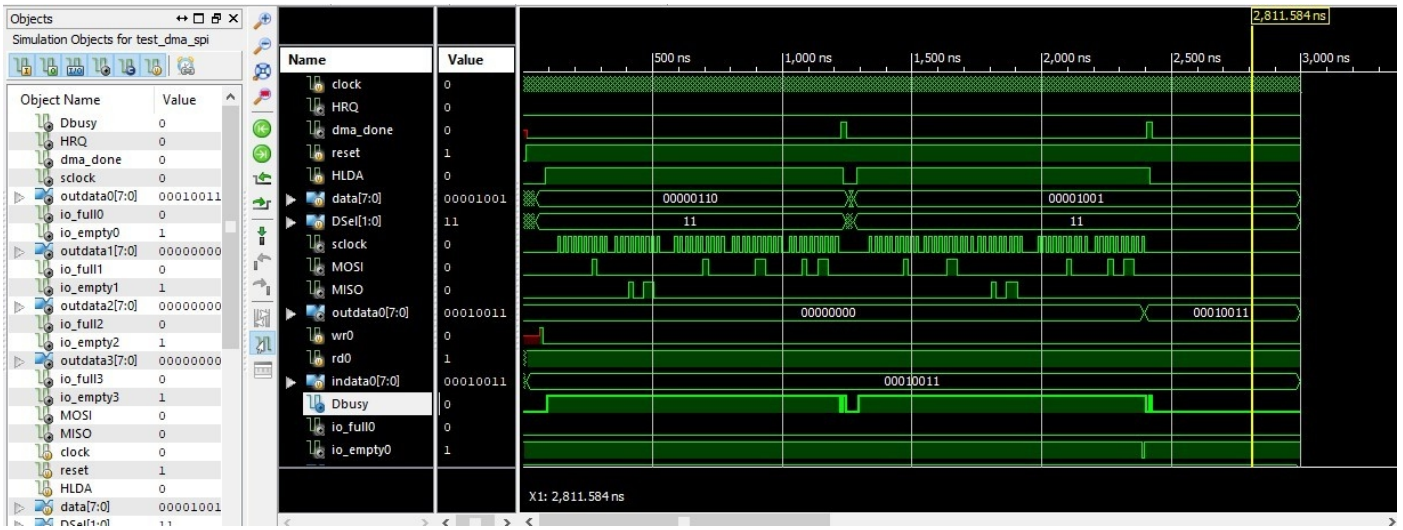
5.Provide multiple IO device request to DMA



6.Provide multiple IO device request to DMA with rotating priority enable



SIMULATION in Isim



6.2 Verification Results

Single read write operation

```
# UVM_INFO scoreboard.sv(28) @ 50: uvm_test_top.m_env.m_scrbrd [my_scoreboard] -scoreboard reset done-
# UVM_INFO sequence.sv(64) @ 50: uvm_test_top.m_env.m_agn.m_sqr@@seq.s1 [readIo_writeMem_1] single Read from
IO and write into Memory
# UVM_INFO scoreboard.sv(47) @ 90: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata1=a6 io_ptr=0
# UVM_INFO driver.sv(157) @ 2490: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO driver.sv(157) @ 2530: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO sequence.sv(108) @ 2670: uvm_test_top.m_env.m_agn.m_sqr@@seq.s2 [readMem_writeIo_1] single Read
from Memory and write into IO device
# UVM_INFO driver.sv(157) @ 5050: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO scoreboard.sv(76) @ 5070: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=a6
```

Burst Read write operation

```
# UVM_INFO sequence.sv(158) @ 5270: uvm_test_top.m_env.m_agn.m_sqr@@seq.s3 [readIo_writeMem_burst] Burst Read from IO and write into Memory
# UVM_INFO scoreboard.sv(55) @ 5310: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=24 io_ptr=0
# UVM_INFO scoreboard.sv(55) @ 5330: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=81 io_ptr=1
# UVM_INFO scoreboard.sv(55) @ 5350: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=9 io_ptr=2
# UVM_INFO scoreboard.sv(55) @ 5370: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=63 io_ptr=3
# UVM_INFO scoreboard.sv(55) @ 5390: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=d io_ptr=4
# UVM_INFO scoreboard.sv(55) @ 5410: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=8d io_ptr=5
# UVM_INFO scoreboard.sv(55) @ 5430: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=65 io_ptr=6
# UVM_INFO scoreboard.sv(55) @ 5450: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=12 io_ptr=7
# UVM_INFO scoreboard.sv(55) @ 5470: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=1 io_ptr=8
# UVM_INFO scoreboard.sv(55) @ 5490: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=d io_ptr=9
# UVM_INFO scoreboard.sv(55) @ 5510: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=76 io_ptr=a
# UVM_INFO scoreboard.sv(55) @ 5530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=3d io_ptr=b
# UVM_INFO scoreboard.sv(55) @ 5550: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=ed io_ptr=c
# UVM_INFO scoreboard.sv(55) @ 5570: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=8c io_ptr=d
# UVM_INFO scoreboard.sv(55) @ 5590: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=f9 io_ptr=e
# UVM_INFO driver.sv(157) @ 39070: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO driver.sv(157) @ 39110: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO sequence.sv(199) @ 46670: uvm_test_top.m_env.m_agn.m_sqr@@seq.s4 [readMem_writeIo_burst] Burst Read from memory and write into IO
# UVM_INFO driver.sv(157) @ 79570: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO scoreboard.sv(100) @ 79590: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=24 outdata3=24 op_ptr=0
# UVM_INFO driver.sv(157) @ 79610: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO scoreboard.sv(100) @ 79610: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=81 outdata3=81 op_ptr=1
# UVM_INFO scoreboard.sv(100) @ 79630: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=9 outdata3=9 op_ptr=2
# UVM_INFO scoreboard.sv(100) @ 79650: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=63 outdata3=63 op_ptr=3
# UVM_INFO scoreboard.sv(100) @ 79670: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=d outdata3=d op_ptr=4
# UVM_INFO scoreboard.sv(100) @ 79690: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=8d outdata3=8d op_ptr=5
# UVM_INFO scoreboard.sv(100) @ 79710: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=65 outdata3=65 op_ptr=6
# UVM_INFO scoreboard.sv(100) @ 79730: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=12 outdata3=12 op_ptr=7
# UVM_INFO scoreboard.sv(100) @ 79750: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=1 outdata3=1 op_ptr=8
# UVM_INFO scoreboard.sv(100) @ 79770: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=d outdata3=d op_ptr=9
# UVM_INFO scoreboard.sv(100) @ 79790: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=76 outdata3=76 op_ptr=a
# UVM_INFO scoreboard.sv(100) @ 79810: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=3d outdata3=3d op_ptr=b
# UVM_INFO scoreboard.sv(100) @ 79830: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=ed outdata3=ed op_ptr=c
# UVM_INFO scoreboard.sv(100) @ 79850: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=8c outdata3=8c op_ptr=d
# UVM_INFO scoreboard.sv(100) @ 79870: uvm_test_top.m_env.m_scrbrd [my_scoreboard] data matching: indata=f9 outdata3=f9 op_ptr=e
```

Multiple IO device request

```
# UVM_INFO sequence.sv(267) @ 93010: uvm_test_top.m_env.m_agn.m_sqr@@seq.s5 [multiple_IO_req]  multiple IO device request to DMA
# UVM_INFO driver.sv(150) @ 93030: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus granted....
# UVM_INFO scoreboard.sv(39) @ 93050: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata0=45 ip_ptr=0
# UVM_INFO scoreboard.sv(124) @ 93050: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata0=45
# UVM_INFO scoreboard.sv(129) @ 93050: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata1=b6
# UVM_INFO scoreboard.sv(134) @ 93050: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=15
# UVM_INFO scoreboard.sv(139) @ 93050: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata3=3c
# UVM_INFO scoreboard.sv(117) @ 93150: uvm_test_top.m_env.m_scrbrd [my_scoreboard] --Multiple IO device request arrived--
# UVM_INFO driver.sv(157) @ 95470: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO driver.sv(157) @ 95510: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO sequence.sv(378) @ 95650: uvm_test_top.m_env.m_agn.m_sqr@@seq.s7 [readMem_writeIo] single Read from Memory and write into IO device
# UVM_INFO scoreboard.sv(144) @ 98030: uvm_test_top.m_env.m_scrbrd [my_scoreboard] ==========device select==========
# UVM_INFO scoreboard.sv(159) @ 98030: uvm_test_top.m_env.m_scrbrd [my_scoreboard] IO device0 is selected : outdata=45
# UVM_INFO driver.sv(157) @ 98030: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
```

Multiple IO device request to DMA with rotating priority enable

```
# UVM_INFO sequence.sv(337) @ 103490: uvm_test_top.m_env.m_agn.m_sqr@@seq.s6 [multiple_IO_req_rot]  multiple IO device request to DMA with rotating priority enable
# UVM_INFO driver.sv(150) @ 103510: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus granted....
# UVM_INFO scoreboard.sv(39) @ 103530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata0=45 ip_ptr=0
# UVM_INFO scoreboard.sv(124) @ 103530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata0=45
# UVM_INFO scoreboard.sv(129) @ 103530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata1=b6
# UVM_INFO scoreboard.sv(134) @ 103530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata2=15
# UVM_INFO scoreboard.sv(139) @ 103530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] input data: indata3=3c
# UVM_INFO scoreboard.sv(117) @ 103630: uvm_test_top.m_env.m_scrbrd [my_scoreboard] --Multiple IO device request arrived--
# UVM_INFO driver.sv(157) @ 105950: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO driver.sv(157) @ 105990: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
# UVM_INFO sequence.sv(378) @ 106130: uvm_test_top.m_env.m_agn.m_sqr@@seq.s7 [readMem_writeIo] single Read from Memory and write into IO device.
# UVM_INFO scoreboard.sv(144) @ 108530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] ==========device select==========
# UVM_INFO scoreboard.sv(155) @ 108530: uvm_test_top.m_env.m_scrbrd [my_scoreboard] IO device1 is selected : outdata=b6
# UVM_INFO driver.sv(157) @ 108550: uvm_test_top.m_env.m_agn.c_drv [cpu_drv] ....bus grant disable....
```

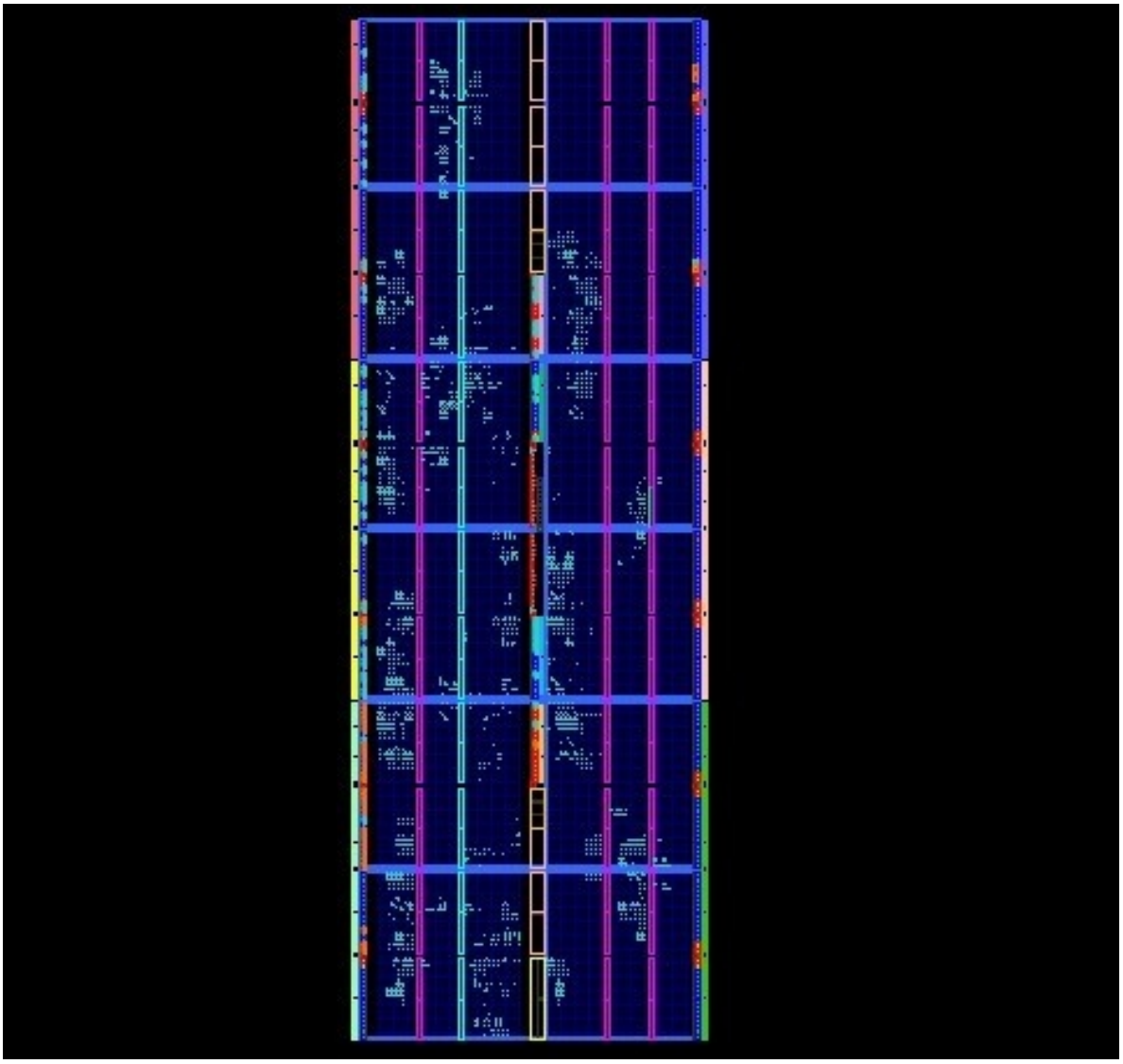
Chapter 7: Implementation:

Design Summary

dma_top Project Status (02/12/2021 - 00:42:51)			
Project File:	dma_spi.xise	**Parser Errors:**	No Errors
Module Name:	dma_top	**Implementation State:**	Programming File Generated
Target Device:	xc4vlx25-10ff668	**•Errors:**	No Errors
Product Version:	ISE 14.7	**•Warnings:**	27 Warnings (0 new)
Design Goal:	Balanced	**•Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	All Constraints Met
Environment:	System Settings	**•Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	1,662	21,504	7%	
Number of 4 input LUTs	2,176	21,504	10%	
Number of occupied Slices	1,737	10,752	16%	
Number of Slices containing only related logic	1,737	1,737	100%	
Number of Slices containing unrelated logic	0	1,737	0%	
Total Number of 4 input LUTs	2,176	21,504	10%	
Number of bonded IOBs	103	448	22%	
Number of BUFG/BUFGCTRLs	2	32	6%	
Number used as BUFGs	2			
Number of FIFO16/RAMB16s	1	72	1%	
Number used as RAMB16s	1			
Average Fanout of Non-Clock Nets	3.61			

Device: Layout



Timing Summary

```
Timing Summary:
---------------
Speed Grade: -10

   Minimum period: 4.764ns (Maximum Frequency: 209.901MHz)
   Minimum input arrival time before clock: 4.854ns
   Maximum output required time after clock: 6.744ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Power Report

Device	
Family	Virtex4
Part	xc4vlx25
Package	ff668
Temp Grade	Commercial
Process	Typical
Speed Grade	-10

Environment	
Ambient Temp (C)	50.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250

Characterization	
PRODUCTION	v1.0,02-02-08

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.071	2	---	---
Logic	0.000	2176	21504	10
Signals	0.000	3393	---	---
BRAMs	0.000	1	72	1
DCMs	0.000	0	8	0
IOs	0.000	103	448	23
Leakage	0.234			
Total	0.305			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	8.7	82.3	52.7

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.176	0.059	0.117
Vccaux	2.500	0.036	0.000	0.036
Vcco25	2.500	0.001	0.000	0.001

	Total	Dynamic	Quiescent
Supply Power (W)	0.305	0.071	0.234

CHAPTER 8: Conclusion

The Design of DMA using Serial Peripheral Interface (SPI) with Single DMA Master and Multiple slave(Memory and IO device) configuration has been done successfully showing that it operates in different Modes of operation. DMA controller can efficiently transfer data between main memory and IO device without involvement of CPU. Using SPI as interface, the design become much more flexible, has improved speed and cost. It consumes less power.

This work is focused on implementation of DMA Controller using SPI
protocol for both master DMA and slave mode (memory/IO device). The entire design is designed with Verilog HDL language and synthesized with Xilinx . And verification of design is done using System Verilog in QuestaSim using UVM. This design can be used in a system, which requires low power and high-speed data transfer.

CHAPTER 9: Future Scope

Direct Memory access (DMA) design works with processor and reduced the load of it. DMA is a logical block to access the data of peripherals and easily to understand individually.DMA allows accessing the data easily without the involvements of the processor and widely use in computer system. It makes accessing the device memory and system memory easily and allows the processor to work simultaneously on its own job while on-going operations of memory usage are carried-out by externally connected devices. Doing this DMA made system performance to boost by allowing processor to perform more other task. Many hardware entities like desk drive controllers, sound, graphics and network cards are using DMA in many systems. It plays an important role in computers to access system memory directly. Similarly it plays an important role in embedded systems. DMA becomes an important unit of System on Chip (SoC) architecture. It offers significantly fast data transfer rate between memory and external devices connected to system. The DMA performance enhances while using with the bus architecture.

Future Scope of this project is that multiple DMA can be connect to channel port of one DMA using SPI interface so that more number of devices can be connected to it and hence single DMA can perform transfer operation to more number of peripherals devices.

Chapter 10: References

Aljumah, A., Ahmed, M.A., 2015. Design of High Speed Data Transfer Direct Memory Access Controller for System on Chip Based Embedded Products. Journal of Applied Sciences

DESIGN AND IMPLEMENTATION OF A DIRECT MEMORY ACCESS CONTROLLER FOR EMBEDDED APPLICATIONS Mohammed Altaf Ahmed1* , Abdullah Aljumah1 , M. Gulam Ahmad1

- SPI Block Guide v04.01 NXP Semiconductors

- Design and Implementation of a High Speed Serial Peripheral Interface
Anand N, George Joseph, Suwin Sam Oommen, and R Dhanabal
School of Electronics Engineering, VIT University, Vellore, India.

- www.TESTBENCH.in

- www.edaboard.in

DMA CONTROLLER 8257 Prepared by R.Kavitha