AMBA AXI3 BUS PROTOCOL

TEAM MEMBERS:

1. Haranadh Chintapalli
2. Soma Sai Charitha Yenuga
3. Durga Swetha Chintakunta
4. Apoorva Kattamanchi

PRESENTATION OVERVIEW

INTRODUCTION

DESIGN

VERIFICATION

EMULATION

OBJECTIVE

- Design and verification of AMBA AXI3 protocol
- To design read/write address channels,read/write data channels and write response channels.
- Verification environment in System Verilog
- Driver
- Monitor
- Assertions
- Emulation: Standalone mode.

INTRODUCTION

- Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect specification
- Targeted at high performance,high frequency system designs
- Separate address/control and data phases.
- Burst based transactions with only start address issued.
- Flexibility in implementation of interconnect architectures.
- Backward compatible with AHB and APB interfaces.

DESIGN

Slave Memory is designed to be 4K addresses.

- 000 to 1FF – RAM1-Invalid Range
- 200 to 5FF – ROM- Read Only
- 600 to FFF – RAM2- Valid Writes and Reads.

Burst implementation modes are:

1. Fixed, Incremental, Wrapping

The AMBA AXI 3 has the following 5 channels

•**Write Address Channel** – The 32 bit Address is given by the master to the slave along with the transaction ID and other control signals

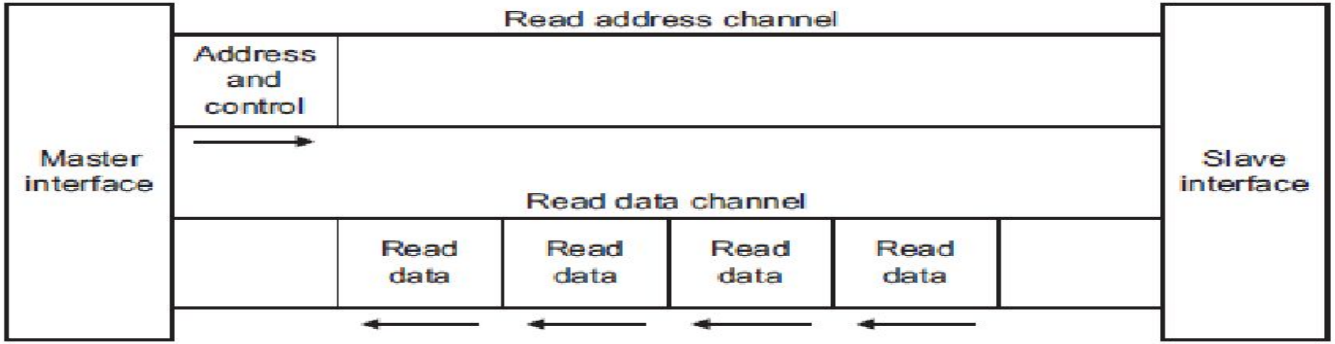
•**Write Data Channel** – With the same transaction ID, data(maximum of 32 bit data per transaction) to be written is transmitted by master to the slave along with the burst vector value

•**Write Response Channel** – As per the received data the slave gives a response

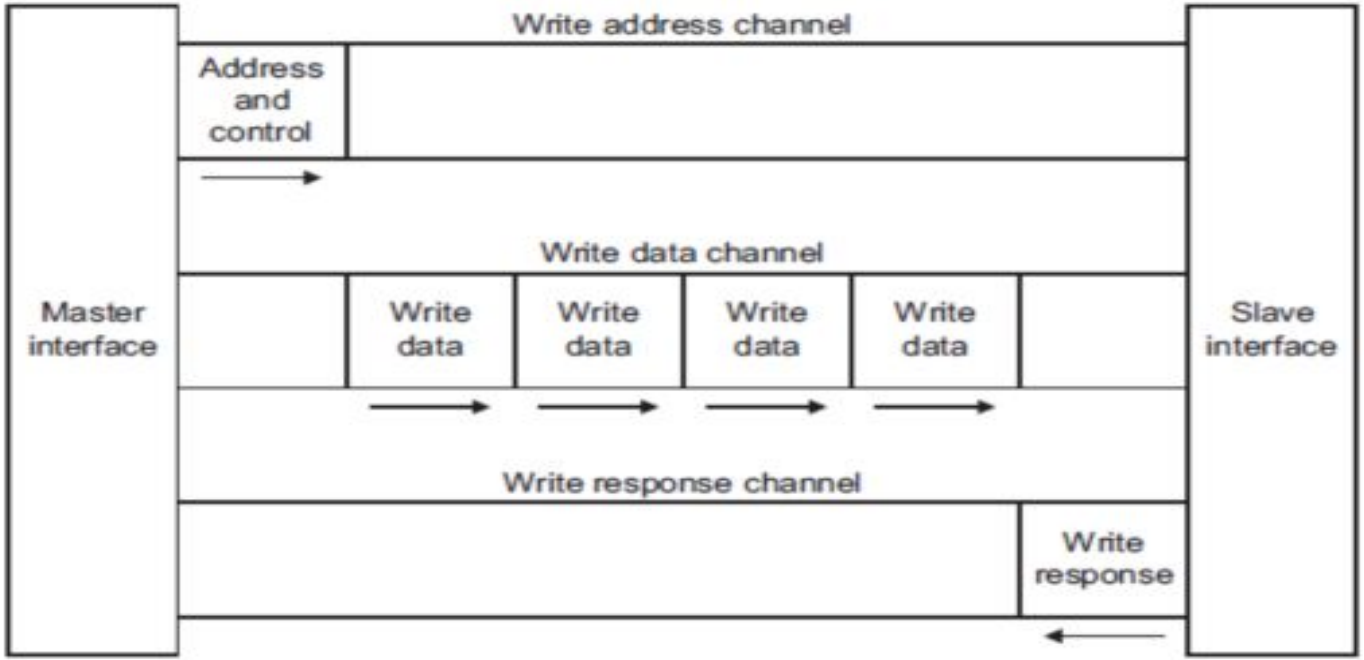
•**Read Address Channel** – Master sends a 32 bit address to the slave requesting to read data from that address

•**Read Data Channel** – Slave transmits the data to the master, corresponding to the address received and also as per the burst vector value

READ TRANSACTION



WRITE TRANSACTION



SYSTEM VERILOG CONSTRUCTS USED IN DESIGN

Interface

Modports

Always_ff and always_comb blocks

Enumerated signals

Unique Case

AXI PROTOCOL SIGNALS

- Address,Data Read,Write, Response Signals from all the channels

Clock and reset (active low)

- VALID and READY signals for each channel
- AWADDR: Write Address, WDATA: Write Data, ARADDR: Read Address, RDATA: Read Data
- AWID, WID, WLAST ,RLAST, BREADY,BVALID,BID, BRESP

Table 7-1 RRESP[1:0] and BRESP[1:0] encoding

RRESP[1:0] BRESP[1:0]	Response	Meaning
b00	OKAY	Normal access okay indicates if a normal access has been successful. Can also indicate an exclusive access failure.
b01	EXOKAY	Exclusive access okay indicates that either the read or write portion of an exclusive access has been successful.
b10	SLVERR	Slave error is used when the access has reached the slave successfully, but the slave wishes to return an error condition to the originating master.
b11	DECERR	Decode error is generated typically by an interconnect component to indicate that there is no slave at the transaction address.

Table 4-3 Burst type encoding

ARBURST[1:0] AWBURST[1:0]	Burst type	Description	Access
b00	FIXED	Fixed-address burst	FIFO-type
b01	INCR	Incrementing-address burst	Normal sequential memory
b10	WRAP	Incrementing-address burst that wraps to a lower address at the wrap boundary	Cache line
b11	Reserved	-	-

Table 4-1 Burst length encoding

ARLEN[3:0] AWLEN[3:0]	Number of data transfers
b0000	1
b0001	2
b0010	3
. . .	
b1101	14
b1110	15
b1111	16

Table 4-2 Burst size encoding

ARSIZE[2:0] AWSIZE[2:0]	Bytes in transfer
b000	1
b001	2
b010	4
b011	8
b100	16
b101	32
b110	64
b111	128

Modports in the design

```
modport master(
        //input          clk,
        //input resetn,

// ADDRESS WRITE CHANNEL
        input   AWREADY,
        output  AWVALID,
        output  AWBURST,
        output  AWSIZE,
        output  AWLEN,
        output  AWADDR,
        output  AWID,

// DATA WRITE CHANNEL
        input   WREADY,
        output  WVALID,
        output  WLAST,
        output  WSTRB,
        output  WDATA,
        output  WID,

// WRITE RESPONSE CHANNEL
        input   BID,
        input   BRESP,
        input   BVALID,
        output  BREADY,

// READ ADDRESS CHANNEL
        input   ARREADY,
        output  ARID,
        output  ARADDR,
        output  ARLEN,
        output  ARSIZE,
        output  ARBURST,
        output  ARVALID,

// READ DATA CHANNEL
        input   RID,
        input   RDATA,
        input   RRESP,
        input   RLAST,
        input   RVALID,
        output  RREADY
);
```

```
modport slave(
        //input          clk,
        //input resetn,

// ADDRESS WRITE CHANNEL
        output  AWREADY,
        input   AWVALID,
        input   AWBURST,
        input   AWSIZE,
        input   AWLEN,
        input   AWADDR,
        input   AWID,

// DATA WRITE CHANNEL
        output  WREADY,
        input   WVALID,
        input   WLAST,
        input   WSTRB,
        input   WDATA,
        input   WID,

// WRITE RESPONSE CHANNEL
        output  BID,
        output  BRESP,
        output  BVALID,
        input   BREADY,

// READ ADDRESS CHANNEL
        output  ARREADY,
        input   ARID,
        input   ARADDR,
        input   ARLEN,
        input   ARSIZE,
        input   ARBURST,
        input   ARVALID,

// READ DATA CHANNEL
        output  RID,
        output  RDATA,
        output  RRESP,
        output  RLAST,
        output  RVALID,
        input   RREADY
);
```

AXI WRITE TRANSFERS DEPENDING ON BURST

FIXED:

```
unique case(AMBAS.AWBURST)
  2'b00:begin
            masteraddress = AWADDR_r;

            unique case (AMBAS.WSTRB)
            4'b0001:begin
                        slave_memory[masteraddress] = AMBAS.WDATA[7:0];
                    end

            4'b0010:begin
                        slave_memory[masteraddress] =  AMBAS.WDATA[15:8];
                    end

            4'b0100:begin
                        slave_memory[masteraddress] =  AMBAS.WDATA[23:16];
                    end

            4'b1000:begin
                        slave_memory[masteraddress] =  AMBAS.WDATA[31:24];
                    end

            4'b0011:begin
                        slave_memory[masteraddress] =  AMBAS.WDATA[7:0];
                        slave_memory[masteraddress+1] =  AMBAS.WDATA[15:8];
                    end
```

INCREMENTAL:

```
unique case (AMBAS.WSTRB)
4'b0001:begin
            slave_memory[masteraddress] =  AMBAS.WDATA [7:0];
            masteraddress_reg = masteraddress + 1;
        end

4'b0010:begin
            slave_memory[masteraddress] =  AMBAS.WDATA [15:8];
            masteraddress_reg = masteraddress + 1;
        end

4'b0100:begin
            slave_memory[masteraddress] =  AMBAS.WDATA [23:16];
            masteraddress_reg = masteraddress + 1;
        end

4'b1000:begin
            slave_memory[masteraddress] =  AMBAS.WDATA [31:24];
            masteraddress_reg = masteraddress + 1;
        end

4'b0011:begin
            slave_memory[masteraddress] =  AMBAS.WDATA [7:0];
            slave_memory[masteraddress+1] =  AMBAS.WDATA [15:8];
            masteraddress_reg = masteraddress + 2;
        end
```

WRAPPING

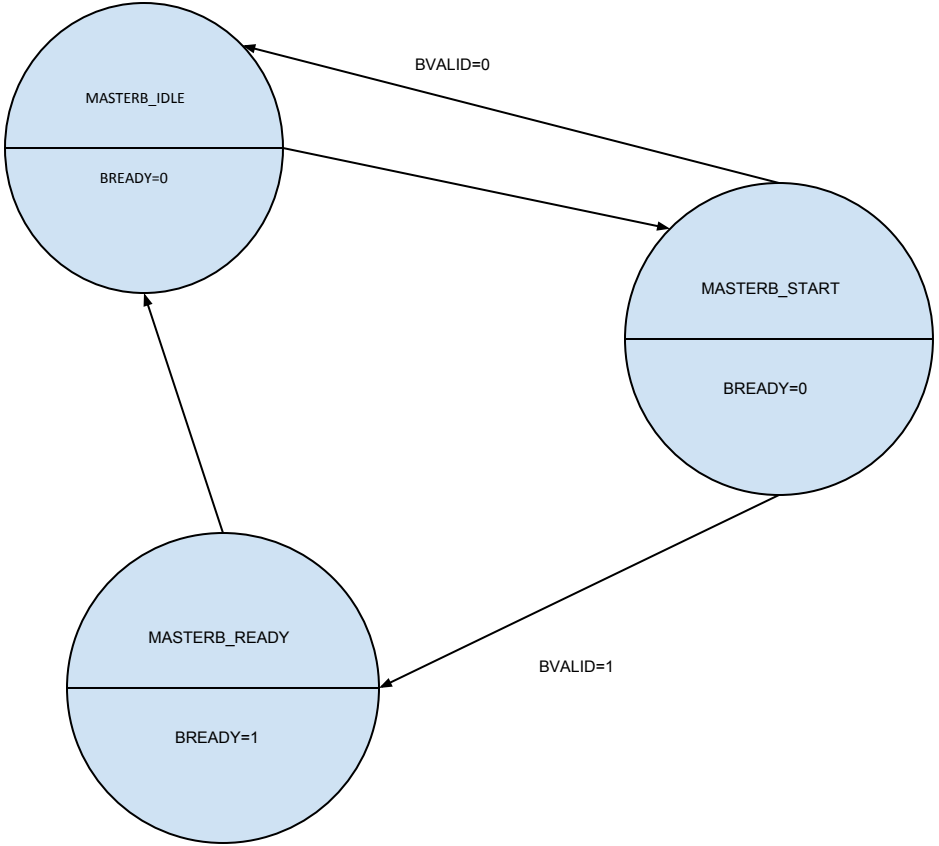
```
unique case(AMBAS.AWLEN)
4'b0001:begin
            unique case(AMBAS.AWSIZE)
            3'b000: begin
                        wrap_boundary = 2 * 1;
                    end
            3'b001: begin
                        wrap_boundary = 2 * 2;
                    end
            3'b010: begin
                        wrap_boundary = 2 * 4;
                    end
             default: begin end
            endcase
        end

4'b0011:begin
            unique case(AMBAS.AWSIZE)
            3'b000: begin
                        wrap_boundary = 4 * 1;
                    end
            3'b001: begin
                        wrap_boundary = 4 * 2;
                    end
            3'b010: begin
                        wrap_boundary = 4 * 4;
```

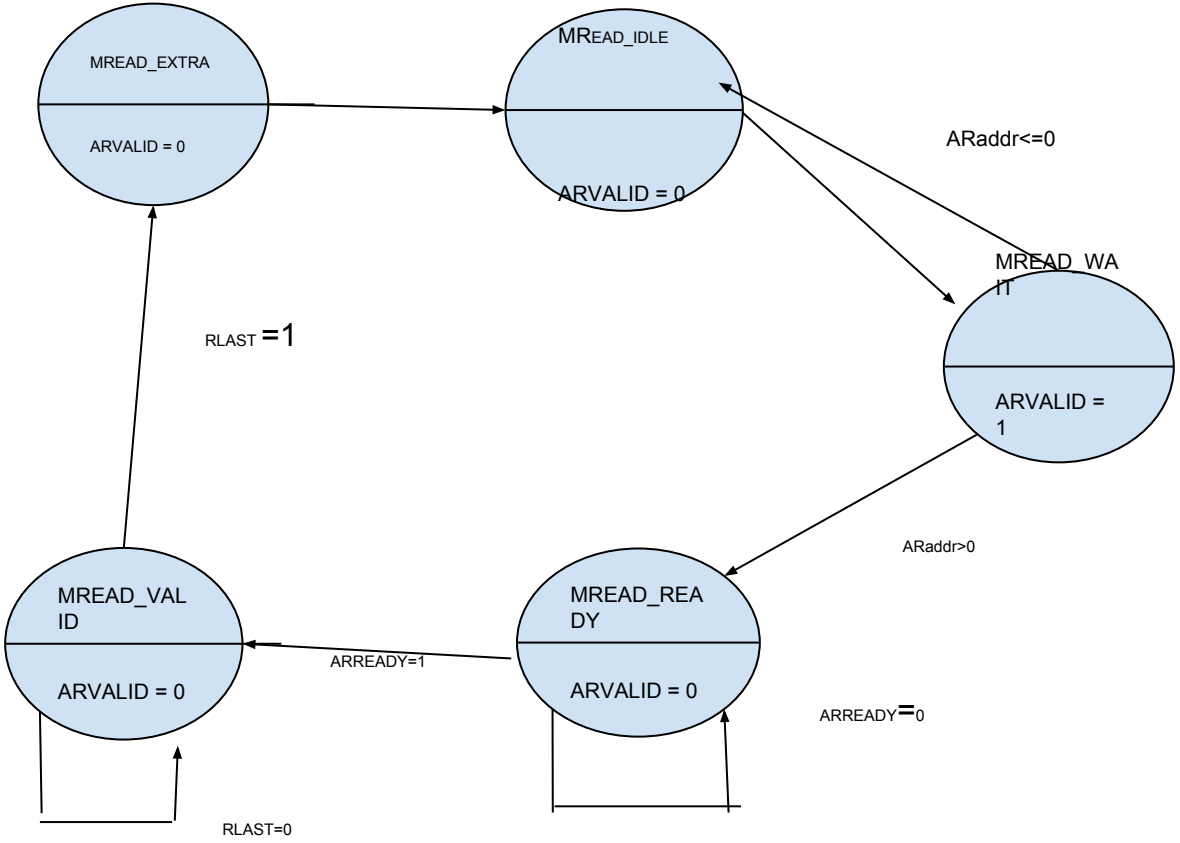
The master generates the start address. The slave computes the wrap boundary depending on the length of the burst and size of each data transfer. Then the address wraps around this wrap boundary to a lower address.

MASTER AND SLAVE FSM’

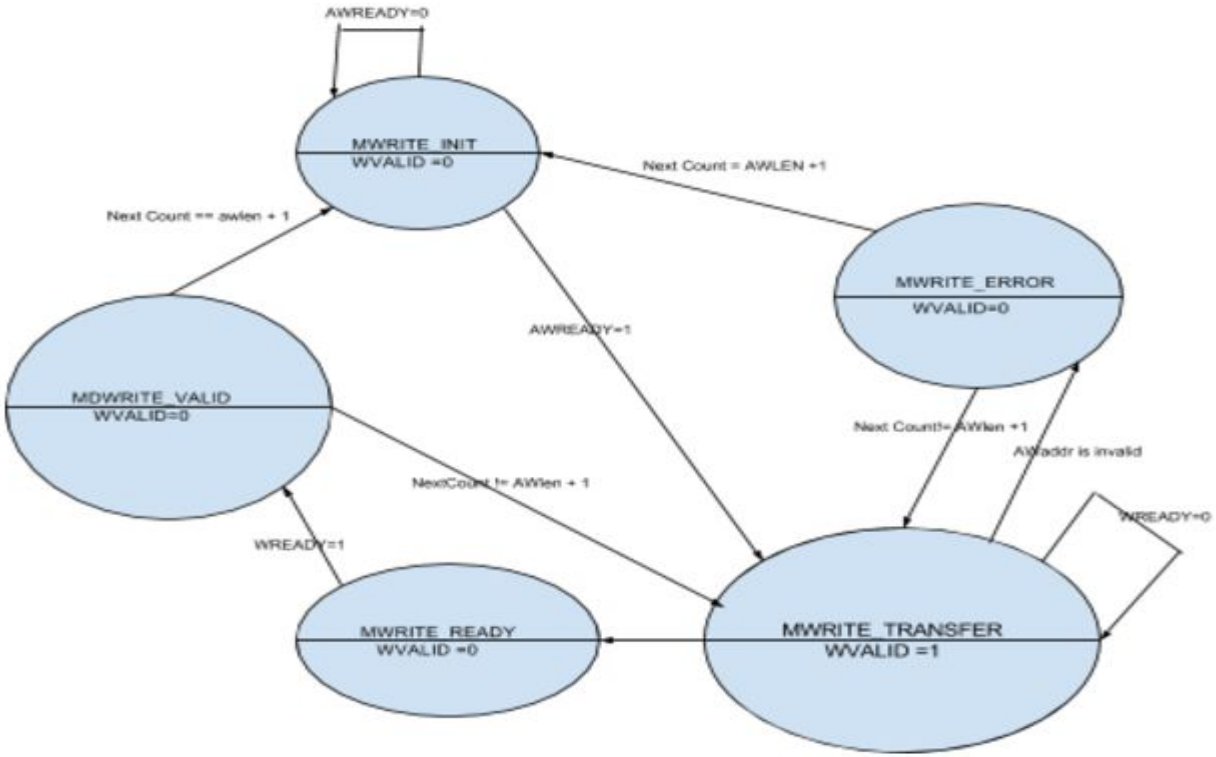
WRITE RESPONSE MASTER



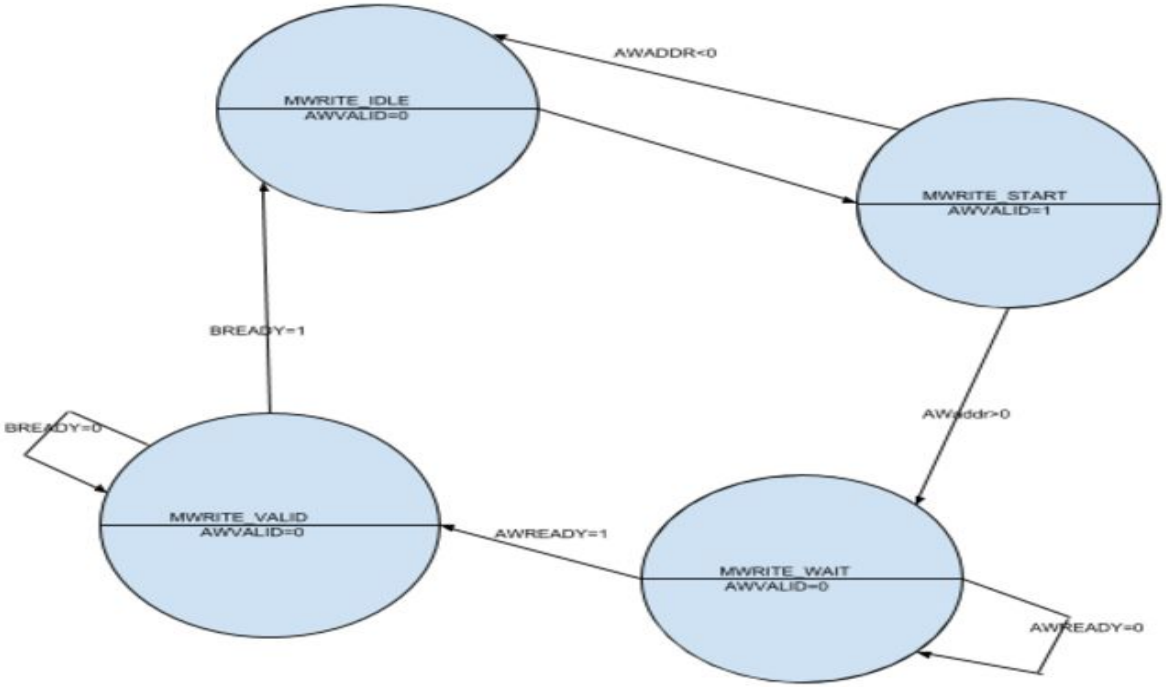
READ ADDRESS MASTER



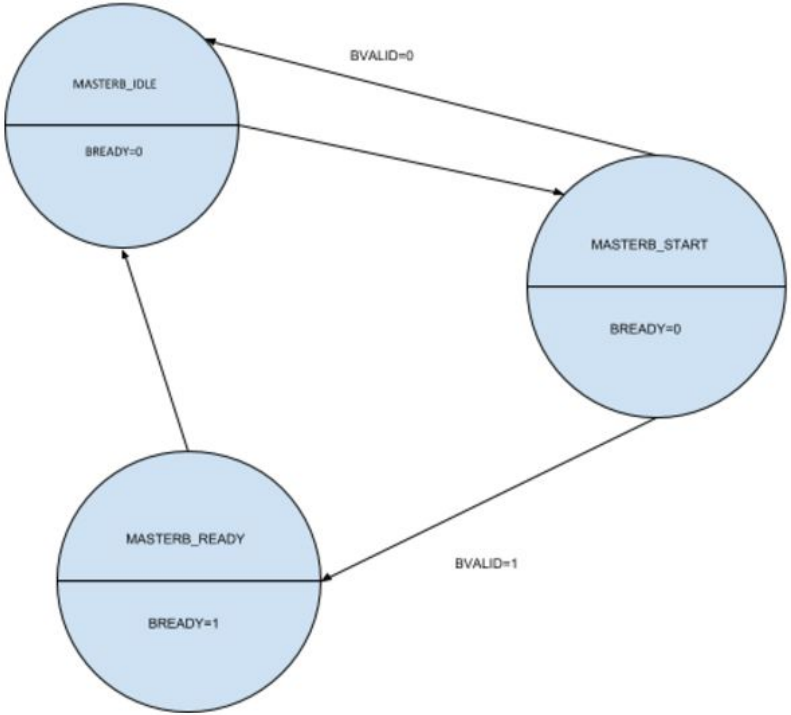
WRITE DATA MASTER

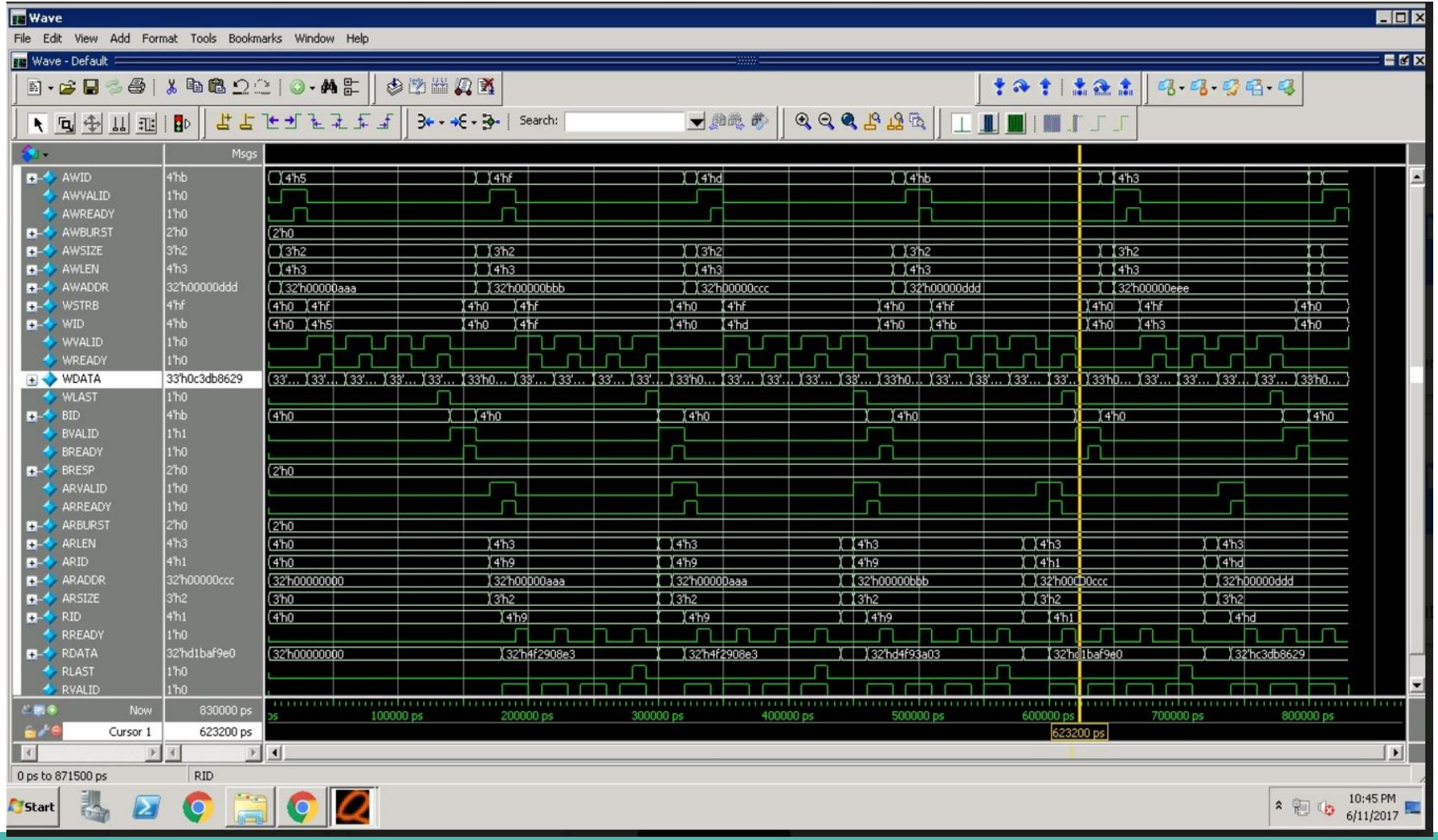


WRITE ADDRESS MASTER



READ ADDRESS MASTER




Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
AWID 4'hb
AWVALID 1'h0
AWREADY 1'h0
AWBURST 2'h0
AWSIZE 3'h2
AWLEN 4'h3
AWADDR 32'h00000ddd
WSTRB 4'hf
WID 4'hb
WVALID 1'h0
WREADY 1'h0
WDATA 33'h0c3db8629
WLAST 1'h0
BID 4'hb
BVALID 1'h1
BREADY 1'h0
BRESP 2'h0
ARVALID 1'h0
ARREADY 1'h0
ARBURST 2'h0
ARLEN 4'h3
ARID 4'h1
ARADDR 32'h00000ccc
ARSIZE 3'h2
RID 4'h1
RREADY 1'h0
RDATA 32'hd1baf9e0
RLAST 1'h0
RVALID 1'h0
Now 830000 ps
Cursor 1 623200 ps
100000 ps
200000 ps
300000 ps
400000 ps
500000 ps
600000 ps
700000 ps
800000 ps
0 ps to 871500 ps
RID
Start
10:45 PM
6/11/2017

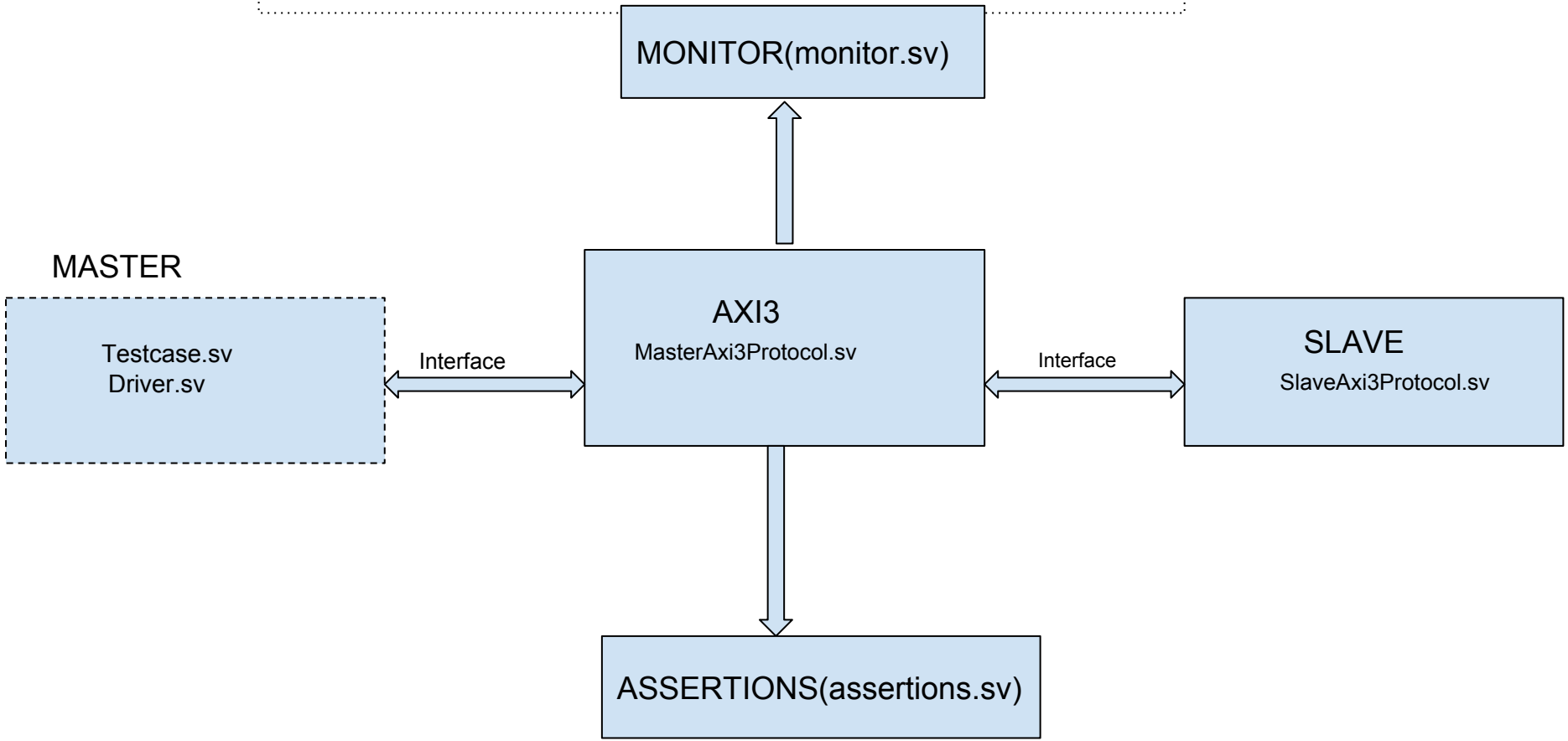
VERIFICATION ENVIRONMENT

ASSERTIONS

MONITOR

DRIVER

VERIFICATION ENVIRONMENT



VERIFICATION BLOCK MODULES

Driver : This module generates addresses , data and all the testcases are defined.

Monitor: This module monitors the outputs.

Assertions: This module is implemented to verify the design.

SYSTEM VERILOG CONSTRUCTS USED

ASSERTIONS

RANDOMIZATION

CLASSES

ASSERTIONS

Concurrent assertions have been used in design to verify the design

```
// ********************Write Data Channel Assertions*******
// WID and AWID should match
property AXI_AWID_WID;
@(posedge clock)
    intf.WVALID |=> (intf.WID == intf.AWID);
endproperty
AXI_WVALID_WID_c: assert property (AXI_AWID_WID);
```

WRITE DATA CHANNEL ASSERTIONS

- Check when WVALID is asserted,WID and AWID matches.
- Check when WVALID is asserted and WREADY is not asserted WDATA, WSTRB,WLAST,WID should remain stable
- Check when WVALID is asserted and WREADY is not asserted WDATA, WSTRB,WLAST,WID should not have any X's or Z's
- Check when WVALID is asserted WREADY becomes high after 1 clock cycle

WRITE ADDRESS CHANNEL ASSERTIONS

- Check size of data transfer must not exceed the width of data interface
- AWBURST cannot be 2'b11
- Check for write address master control signals,if they are stable after asserting AWVALID.
- Check when AWVALID is asserted, AWVALID goes high and then goes low

WRITE RESPONSE CHANNEL ASSERTIONS

- Check if BID and AWID matches
- Check if after BVALID is asserted BREADY becomes high after 1 clock cycle
- Check if slave should initiate response, after WLAST is asserted

READ ADDRESS CHANNEL ASSERTIONS

- Check when WVALID is asserted and WREADY is not asserted WDATA, WSTRB,WLAST,WID should remain stable.
- After ARVALID is asserted ARREADY becomes high after 1 clock cycle

READ DATA CHANNEL ASSERTIONS

- Check if RID and ARID is same
- Check if CONTROL SIGNALS ARE STABLE
- Check after RVALID is asserted RREADY becomes 1 after 1 clock cycle

SCREENSHOT OF ALL ASSERTIONS PASSED

Questa Sim-64 10.4c

File Edit View Compile Simulate Add Structure Tools Layout Bookmarks Window Help

sim - Default

Instance	Design unit	Design unit type	Top Category	Visibility	Total coverage	Assertions hit	Assertions missed	Assertion %	Assertion graph
top	top(fast)	Module	DU Instance	+acc=p	100.0%	20	0	100.0%	
intf	axi(fast__1)	Interface	DU Instance	+acc=<none>					
test	testcase(fa...	Program	DU Instance	+acc=<none>					
Top_inst	Top_HDL(f...	Module	DU Instance	+acc=<none>	100.0%	20	0	100.0%	
testcase_sv_unit	testcase_s...	VlPackage	Package	+acc=<none>					
Top_HVL_sv_unit	Top_HVL_s...	VlPackage	Package	+acc=<none>					
std	std	VlPackage	Package	+acc=<full>					
#vsim_capacity#		Capacity	Statistics	+acc=<none>					

Transcript | Dataflow | Analysis | Assertions | Objects | Processes | Details | Project | sim

Now: 700 ns Delta: 6 | sim:/top - Limited Visibility Region | Keep 0 | Recursive Mode

Start

3:08 AM
6/12/2017

DRIVER

- Driver uses ‘rand’, ‘constraint’ for generating the different range of addresses.
- Driver also generates Tasks for the Burst transactions.

Driver - Stimulus

```
class DRVI;                                  //Creating a class for generating random signals which are passed to the driver for performing different type of transactions.
rand    bit [3:0]   rand_awid;               //Random write address id generating
rand    bit [31:0]  rand_awaddr_valid;       //Random Valid Write Address generating with Constraint S1
constraint S1{                               //
    rand_awaddr_valid > 32'h5ff;             //
    rand_awaddr_valid <=32'hfff;             //
        }
rand    bit [31:0]  rand_awaddr_readonly;    //Random Read-only address with Constraint S2
constraint S2{
    rand_awaddr_readonly > 32'h1ff;
    rand_awaddr_readonly <= 32'h5ff;
        }
rand    bit [31:0]  rand_awaddr_invalid;     //Random InValid write address with Constraint S3
constraint S3{
    rand_awaddr_invalid <= 32'h1ff;
        }
rand    bit [31:0]  rand_wdata;              //Random write data generating. 32bit  used
rand    bit [31:0]  rand_araddr_valid;       // Random Valid read address with Constraint S4
constraint S4{
    rand_araddr_valid > 32'h1ff;
    rand_araddr_valid <= 32'hfff;
        }
rand    bit [31:0]  rand_araddr_invalid;     // Random InValid read address with Constraint S5
constraint S5{
    rand_araddr_invalid <= 32'h1ff;
        }
rand    bit [3:0]   rand_arid;               //Random read address id generating

endclass
```

Inside Driver- Tasks

Reset all the signals.

```
task reset_enable;                    //Reset task
    top.reset          =   1'b0;
    intf.AWREADY       =   '0;
    intf.AWVALID       =   '0;
    intf.AWBURST       =   '0;
    intf.AWSIZE     =   '0;
    intf.AWLEN      =   '0;
    intf.AWADDR     =   '0;
    intf.AWID       =   '0;
    intf.WREADY     =   '0;
    intf.WVALID     =   '0;
    intf.WLAST      =   '0;
    intf.WSTRB      =   '0;
    intf.WDATA      =   '0;
    intf.WID        =   '0;
    intf.BID        =   '0;
    intf.BRESP      =   '0;
    intf.BVALID     =   '0;
    intf.BREADY     =   '0;
    intf.ARREADY       =   '0;
    intf.ARID       =   '0;
    intf.ARADDR     =   '0;
    intf.ARLEN      =   '0;
    intf.ARSIZE     =   '0;
    intf.ARBURST       =   '0;
    intf.ARVALID       =   '0;
    intf.RID        =   '0;
    intf.RDATA      =   '0;
    intf.RRESP      =   '0;
    intf.RLAST      =   '0;
    intf.RVALID     =   '0;
    intf.RREADY     =   '0;
    top.AWaddr  =   '0;
    top.AWid    =   '0;
    top.AWsize  =   '0;
    top.AWlen   =   '0;
    top.WStrb   =   '0;
    top.AWburst =   '0;
    top.WData   =   '0;
    top.ARid    =   '0;
    top.ARaddr  =   '0;
    top.ARlen   =   '0;
    top.ARsize  =   '0;
    top.ARburst =   '0;
    #10;
    top.reset       =   1'b1;
endtask
```

ALTERNATE WRITE AND READ

```
////////////////////////////////// 1. Task for 'Alternate write and read transactions' ////////////////////////////////////////////////////////////////////////////////////////////////////////////

task burst_write_read(input bit [3:0] rand_awid, input bit [31:0] rand_awaddr_valid, input bit [31:0] rand_awaddr_invalid, input bit [31:0] rand_awaddr_readonly, input bit [31:0] rand_wdata,  input bit [31:0] rand_

for(b=2'b10;b<2'b11;b++) begin
                stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                top.AWid    =   rand_awid;
                top.AWaddr  =   32'hddd; //Valid write address;
                top.AWburst =   top.AWburst + b;
                top.AWsize  =   3'b010;
                top.AWlen   =   4'b0011;
                top.WStrb   =   4'b1111;
                for(i='0;i<=top.AWlen;i=i+4'b1)
                    begin
                    wait(intf.WVALID)
                    stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                    top.WData  = rand_wdata;
                    wait(!intf.WVALID);
                    end
                    wait(intf.BREADY)
                repeat(2) @(posedge top.clock);

                stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                top.AWid    =   rand_awid;
                top.ARid    =   rand_arid;
                top.AWaddr  =   32'hccc;    //rand_awaddr_valid;
                top.ARaddr  =   32'hddd;    //rand_araddr_valid;
                top.AWburst =   top.AWburst + b;
                top.ARburst =   top.ARburst + b;
                top.AWsize  =   3'b010;
                top.ARsize  =   3'b010;
                top.AWlen   =   4'b0011;
                top.ARlen   =   4'b0011;
                top.WStrb   =   4'b1111;
                for(i='0;i<=top.AWlen;i=i+4'b1) begin
                wait(intf.WVALID)
                stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                top.WData  = rand_wdata;
                wait(!intf.WVALID);
                end
                wait(intf.BREADY)
                repeat(2) @(posedge top.clock);
```

VALID READ OPERATION

```
////////////////////////////////2. Valid Read Operation ////////////////////////////////////////////////////////////

task burst_read(input bit [3:0] rand_awid, input bit [31:0] rand_awaddr_valid, input bit [31:0] rand_awaddr_invalid, input bit [31:0] rand_awaddr_readonly, input bit [31:0] rand_wdata,  input bit [31:0] rand_arad


for(x='0;x<2'b11;x++) begin
    for(y='0;y<=3'b010;y++) begin
        if(x!=2'b10) begin
        for(z='0;z<=4'b1111;z++) begin
        stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
            top.ARid    =   4'b1;//id
            top.ARaddr  =   32'ha11;// Valid Write address
            top.ARburst =   top.ARburst + x;
            top.ARsize  =   top.ARsize + y;
            top.ARlen   =   top.ARlen + z;
            wait (intf.RLAST)
            repeat(3) @(posedge top.clock);
        end
        end
        else
        begin
        for(p=4'b1;p<=4'b0100;p++)
            begin
            l=((2**(p))-1);
            stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
            top.ARid    =   4'b1;//id
            top.ARaddr  =   32'ha11;// Valid Read address
            top.ARburst =   top.ARburst + x;
            top.ARsize  =   top.ARsize + y;
            top.ARlen   =   top.ARlen + l;
            wait (intf.RLAST)
            repeat(3) @(posedge top.clock);
            end
        end
    end
end

endtask
```

Write to Read Only Location

```
//////////////////////////////// 3.Writing to a read only location ////////////////////////////////////////////////

task burst_write_readonly(input bit [3:0] rand_awid, input bit [31:0] rand_awaddr_valid, input bit [31:0] rand_awaddr_invalid, input bit [31:0] rand_awaddr_readonly, input bit [31:0] rand_wdata,  input bit [31:0] r

for(n=0;n<10;n++) begin
    for(b=2'b00;b<2'b11;b++) begin
            stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                top.AWid        =   rand_awid;
                top.AWaddr  =   rand_awaddr_readonly;
                top.AWburst =   top.AWburst + b;
                top.AWsize  =   3'b010;
                top.AWlen   =   4'b0011;
                top.WStrb   =   4'b1111;
                for(i='0;i<=top.AWlen;i=i+4'b1)
                begin

                stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                top.WData  = rand_wdata;

                end
                wait(intf.BREADY)
                repeat(2) @(posedge top.clock);
    end
end
endtask
```

Write to invalid address

```
////////////////////////////////////4. Writing to invalid address   ////////////////////////////////////////////////////

task burst_write_invalid(input bit [3:0] rand_awid, input bit [31:0] rand_awaddr_valid, input bit [31:0] rand_awaddr_invalid, input bit [31:0] rand_awaddr_readonly, input bit [31:0] rand_wdata,  input bit [31:0] ra

for(n=0;n<10;n++) begin
    for(b=2'b00;b<2'b11;b++) begin
            stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                top.AWid        =   rand_awid;
                top.AWaddr  =   rand_awaddr_invalid;
                top.AWburst =   top.AWburst + b;
                top.AWsize  =   3'b010;
                top.AWlen   =   4'b0011;
                top.WStrb   =   4'b1111;
                for(i='0;i<=top.AWlen;i=i+4'b1)
                begin

                stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
                top.WData  = rand_wdata;

                end
                wait(intf.BREADY)
                repeat(2) @(posedge top.clock);
    end
end
endtask
```

Reading from Invalid address

```
//////////////////////////////// 5.Reading from invalid address  ////////////////////////////////////////////////////////////////////////

task burst_read_invalid(input bit [3:0] rand_awid, input bit [31:0] rand_awaddr_valid, input bit [31:0] rand_awaddr_invalid, input bit [31:0] rand_awaddr_readonly, input bit [31:0] rand_wdata,  input bit [31:0] ran

for(n=0;n<10;n++) begin
for(x='0;x<2'b11;x++) begin  //for each burst type
        stimulus(rand_awid,rand_awaddr_valid,rand_awaddr_invalid,rand_awaddr_readonly,rand_wdata,rand_araddr_valid,rand_araddr_invalid,rand_arid);
        top.ARid    =   rand_arid;
        top.ARaddr  =   rand_araddr_invalid;
        top.ARburst =   top.ARburst + x;
        top.ARsize  =   3'b010;
        top.ARlen   =   4'b0011;
        wait (intf.RLAST)
        repeat(3) @(posedge top.clock);
    end
end

endtask
```

MONITOR

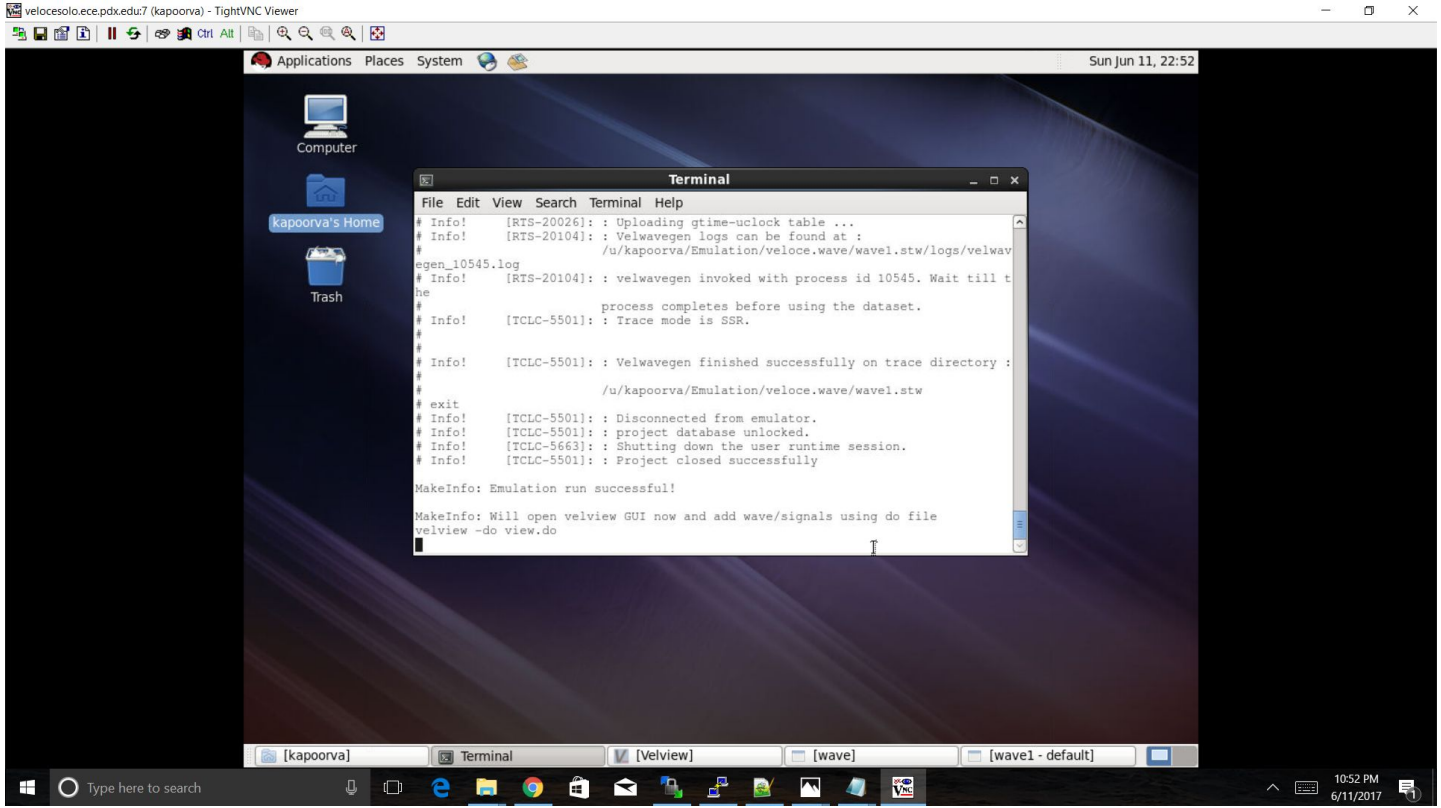
- Displays the output Write, Read Data of DUT

EMULATION

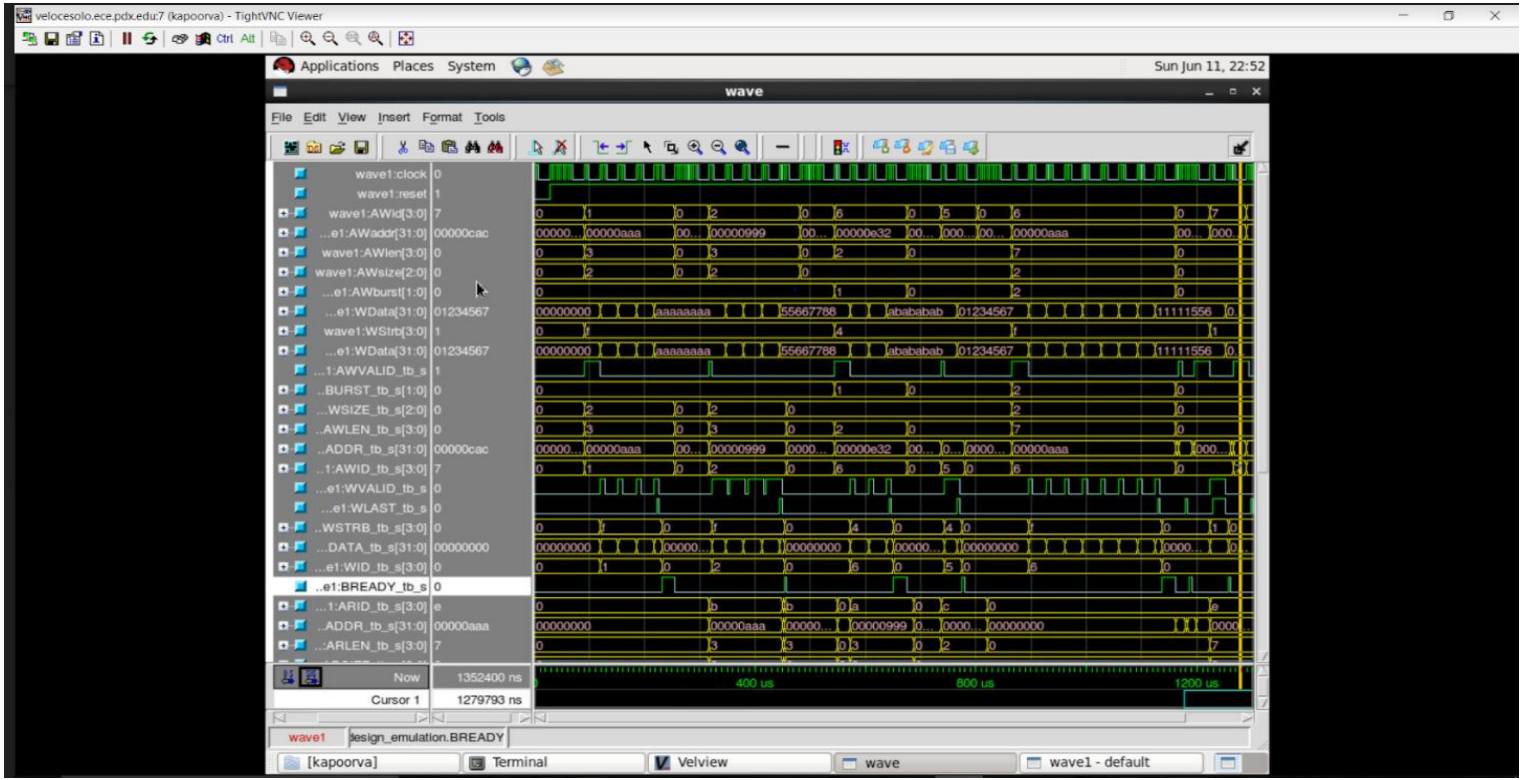
STANDALONE MODE:

- We have implemented our design in standalone mode.
- Design is synthesized without tasks.
- Implemented test patterns, loaded into emulator via run.do file.

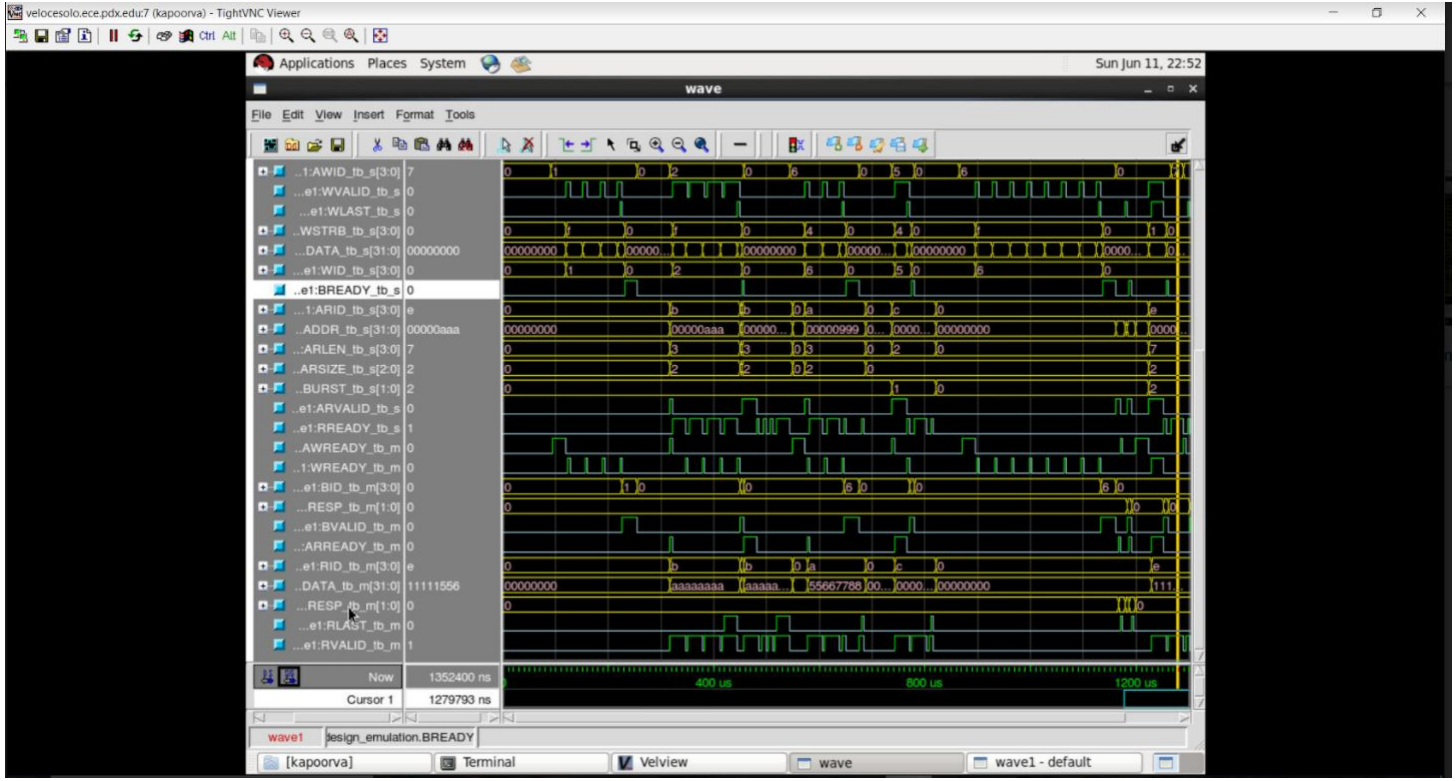
EMULATION


velocesolo.ece.pdx.edu:7 (kapoorva) - TightVNC Viewer
Applications Places System
Sun Jun 11, 22:52
Computer
kapoorva's Home
Trash
Terminal
File Edit View Search Terminal Help
Info! [RTS-20026]: : Uploading gtime-uclock table ...
Info! [RTS-20104]: : Velwavegen logs can be found at :
/u/kapoorva/Emulation/veloce.wave/wave1.stw/logs/velwavegen_10545.log
Info! [RTS-20104]: : velwavegen invoked with process id 10545. Wait till the
process completes before using the dataset.
Info! [TCLC-5501]: : Trace mode is SSR.
#
#
Info! [TCLC-5501]: : Velwavegen finished successfully on trace directory :
#
/u/kapoorva/Emulation/veloce.wave/wave1.stw
exit
Info! [TCLC-5501]: : Disconnected from emulator.
Info! [TCLC-5501]: : project database unlocked.
Info! [TCLC-5663]: : Shutting down the user runtime session.
Info! [TCLC-5501]: : Project closed successfully
MakeInfo: Emulation run successful!
MakeInfo: Will open velview GUI now and add wave/signals using do file
velview -do view.do
[kapoorva]
Terminal
[Velview]
[wave]
[wave1 - default]
Type here to search
10:52 PM
6/11/2017

EMULATION RESULT



EMULATION RESULT



CHALLENGES FACED

- 2D arrays:Not displayed in timing setup
- `include for interface:Gave duplicate definition warning
- Multi driven port
- Dead logic:setting up signals in top modules

REFERENCES

1.ARM, AMBA Specifications (Rev2.0). [Online]. Available at http://www.arm.com.

2. http://www.ijeijournal.com/papers/v1i3/D0131926.pdf

3. ECE571 LECTURE SLIDES

THANK YOU!!!!!