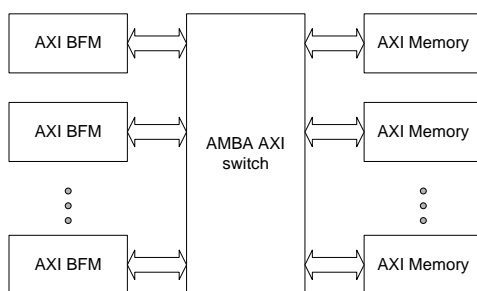


# AMBA AXI Task-based BFM and Simple Memory

2013 – 2016

Ando Ki  
(adki@future-ds.com)

## AMBA AXI bus design & verification



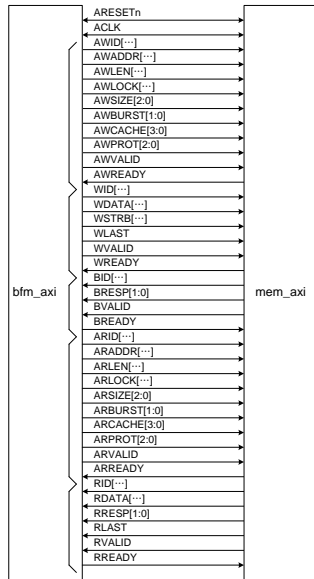
❑ It is assumed that we are going to design AMBA AXI bus.

- ◆ AMBA AXI is DUT/DUV (design under test or design under verification)
- ◆ Test-bench includes 'BFM' and 'MEMORY'.
- ◆ BFM can generate test-pattern and test-vector.

❑ To test AMBA AXI, we need test-bench.

❑ Thus, the test-bench for AMBA AXI is prepared, which includes BFM and Memory.

## Preparing BFM and MEMORY



- ❑ Note that only one master, i.e., bfm\_axi
- ❑ Note that only one slave, i.e., mem\_axi

Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 3 )

## AXI BFM: module

```

`timescale 1ns/1ns

module bfm_axi #(parameter MST_ID =0 // Master ID
    , WIDTH_CID=4
    , WIDTH_ID=4 // ID width in bits
    , WIDTH_AD=32 // address width
    , WIDTH_DA=32 // data width
    , WIDTH_DS=(WIDTH_DA/8) // data strobe width
    , WIDTH_DSB=clogb2(WIDTH_DS) // data strobe width
    , EN =1
    , ADDR_LENGTH=12)
(
    input wire ARESETn
    , input wire ACLK
    , output wire [WIDTH_CID-1:0] MID
    //-----
    , output reg [WIDTH_ID-1:0] AWID
    , output reg [WIDTH_AD-1:0] AWADDR
    `ifdef AMBA_AXI4
    , output reg [7:0] AWLEN
    , output reg AWLOCK
    `else
    , output reg [3:0] AWLEN
    , output reg [1:0] AWLOCK
    `endif
    , output reg [2:0] AWSIZE
    , output reg [1:0] AWBURST
    `ifdef AMBA_AXI_CACHE
    , output reg [3:0] AWCACHE
    `endif
    `ifdef AMBA_AXI_PROT
    , output reg [2:0] AWPROT
    `endif
)

```

- ❑ ADDR\_LENGTH determines range of address to test
- ❑ WIDTH\_AD/DA specifies width of address and data bus
- ❑ MST\_ID specifies which master and WIDTH\_CID specifies which transaction from this master
- ❑ EN controls active or not
- ❑ Macro 'AMBA\_AXI4', 'AMBA\_AXI\_CACHE', 'AMBA\_AXI\_PROT' determine corresponding ports are used or not.

Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 4 )

# AXI BFM: module

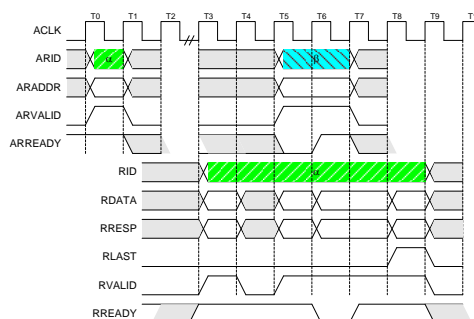
<pre>, output reg      AWVALID , input  wire     AWREADY `ifdef AMBA_AXI4 , output reg [3:0] AWQOS , output reg [3:0] AWREGION `endif  //----- , output reg [WIDTH_ID-1:0] WID , output reg [WIDTH_DA-1:0] WDATA , output reg [WIDTH_DS-1:0] WSTRB , output reg      WLAST , output reg      WVALID , input  wire     WREADY  //----- , input  wire [WIDTH_ID-1:0] BID , input  wire [1:0]      BRESP , input  wire     BVALID , output reg      BREADY  //----- , output reg [WIDTH_ID-1:0] ARID , output reg [WIDTH_AD-1:0] ARADDR `ifdef AMBA_AXI4 , output reg [7:0]      ARLEN , output reg      ARLOCK `else , output reg [3:0]      ARLEN , output reg [1:0]      ARLOCK `endif , output reg [2:0]      ARSIZE , output reg [1:0]      ARBURST `ifdef AMBA_AXI_CACHE , output reg [3:0]      ARCACHE `endif</pre>	<pre>`ifdef AMBA_AXI_PROT , output reg [2:0]      ARPROT `endif , output reg      ARVALID , input  wire     ARREADY `ifdef AMBA_AXI4 , output reg [3:0]      ARQOS , output reg [3:0]      ARREGION `endif  //----- , input  wire [WIDTH_ID-1:0] RID , input  wire [WIDTH_DA-1:0] RDATA , input  wire [1:0]      RRESP , input  wire     RLAST , input  wire     RVALID , output reg      RREADY  //----- , input  wire     CSYSREQ , output reg      CSYSACK , output reg      CACTIVE );  //----- assign MID = MST_ID; //-----</pre>
--	--

# AXI BFM: module

<pre>assign MID = MST_ID;  //----- reg [15:0] bnum ; initial bnum = 0; reg [15:0] blen ; initial blen = 0; reg      delay; initial delay = 0; reg [31:0] saddr, depth; reg      DONE = 1'b0; integer  nm, ns;  //----- initial begin     ....     wait (ARESETn==1'b0);     wait (ARESETn==1'b1);     repeat (5) @ (posedge ACLK);     //----- if (EN) begin     // single-burst with different size     if (1) begin         test_raw( 32'h1 //input [31:0] id;                 , 32'h0 //input [31:0] saddr; // start address                 , 32'h10 //input [31:0] depth; // size in byte                 , 32'h4 //input [31:0] bsize; // burst size in byte                 , 32'h1 //input [31:0] bleng; // burst length         );     end     if (1) begin         test_raw( 32'h1 //input [31:0] id;                 , 32'h10 //input [31:0] saddr; // start address                 , 32'h10 //input [31:0] depth; // size in byte                 , 32'h2 //input [31:0] bsize; // burst size in byte                 , 32'h1 //input [31:0] bleng; // burst length         );     end end</pre>	<pre>if (1) begin     test_raw( 32'h1 //input [31:0] id;             , 32'h20 //input [31:0] saddr; // start address             , 32'h10 //input [31:0] depth; // size in byte             , 32'h1 //input [31:0] bsize; // burst size in byte             , 32'h1 //input [31:0] bleng; // burst length     ); end  //----- // single-burst with different size if (1) begin     test_raw_all( 32'h1 //input [31:0] id;                 , 32'h30 //input [31:0] saddr; // start address                 , 32'h10 //input [31:0] depth; // size in byte                 , 32'h4 //input [31:0] bsize; // burst size in byte                 , 32'h1 //input [31:0] bleng; // burst length     ); end ... end  //----- repeat (10) @ (posedge ACLK); DONE = 1'b1; //\$finish(2); end  //----- `include "bfm_axi_test.v" `include "bfm_axi_tasks.v" //----- endmodule</pre>
---	--

## AXI BFM: read task (1/3)

```
task read_task;
    input [31:0] id;
    input [WIDTH_AD-1:0] addr;
    input [31:0] size; // 1 ~ 128 byte in a beat
    input [31:0] leng; // 1 ~ 16 beats in a burst
    input [31:0] type; // burst type
begin
    fork
        read_address_channel(id,addr,size,leng,type);
        read_data_channel(id,addr,size,leng,type);
    join
end
endtask
```



Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 7 )

## AXI BFM: read task (2/3)

```
task read_address_channel;
    input [31:0] id;
    input [WIDTH_AD-1:0] addr;
    input [31:0] size; // 1 ~ 128 byte in a beat
    input [31:0] leng; // 1 ~ 16 beats in a burst
    input [31:0] type; // burst type
begin
    @ (posedge ACLK);
    ARID <= #1 id;
    ARADDR <= #1 addr;
    ARLEN <= #1 leng-1;
    ARLOCK <= #1 'b0;
    ARSIZE <= #1 get_size(size);
    ARBURST <= #1 type[1:0];
    `ifdef AMBA_AXI_PROT
    ARPROT <= #1 'h0; // data, secure, normal
    `endif
    ARVALID <= #1 'b1;
    @ (posedge ACLK);
    while (ARREADY==1'b0) @ (posedge ACLK);
    ARVALID <= #1 'b0;
    @ (negedge ACLK);
end
endtask
```

```
task read_data_channel;
    input [31:0] id;
    input [WIDTH_AD-1:0] addr;
    input [31:0] size; // 1 ~ 128 byte in a beat
    input [31:0] leng; // 1 ~ 16 beats in a burst
    input [31:0] type; // burst type
    reg [WIDTH_AD-1:0] naddr;
    reg [WIDTH_DS-1:0] strb;
    reg [WIDTH_DA-1:0] maskT;
    reg [WIDTH_DA-1:0] dataR;
    integer idx, idy, idz;
begin
    idz = 0;
    naddr = addr;
    @ (posedge ACLK);
    RREADY <= #1 'b1;
    for (idx=0; idx<leng; idx=idx+1) begin
        @ (posedge ACLK);
        while (RVALID==1'b0) @ (posedge ACLK);
        strb = get_strb(naddr, size);
        dataR = RDATA;
        for (idy=0; idy<WIDTH_DS; idy=idy+1) begin
            if (strb[idy]) begin
                dataRB[idz] = dataR&8'hFF; // justified
                idz = idz + 1;
            end
            dataR = dataR>>8;
        end
    end
end
```

Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 8 )

## AXI BFM: read task (3/3)

```

if (id!=RID) begin
    $display($time, "%m Error id/RID mis-match for read-data-
channel", id, RID);
end
if (idx==leng-1) begin
    if (RLAST==1'b0) begin
        $display($time, "%m Error RLAST expected for read-
data-channel");
    end
    end else begin
        @ (negedge ACLK);
        naddr = get_next_addr( naddr // current address
            , size // num of bytes in a beat
            , type); // type of burst
    end
end
RREADY <= #1'b0;
@ (negedge ACLK);
end
endtask

```

Copyright © 2013-2017 by Ando Ki

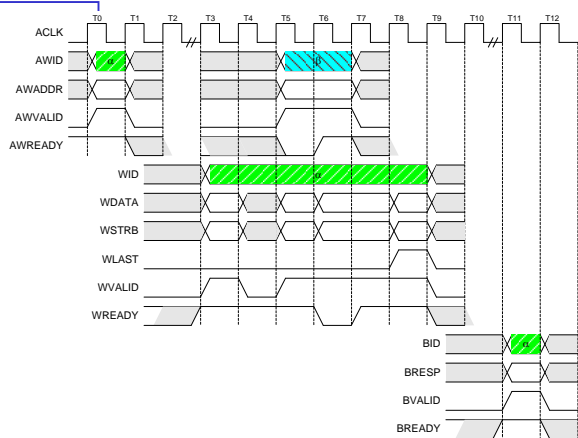
AXI BFM Task ( 9 )

## AXI BFM: write task (1/3)

```

task write_task;
    input [31:0] id;
    input [WIDTH_AD-1:0] addr;
    input [31:0] size; // 1 ~ 128 byte in a beat
    input [31:0] leng; // 1 ~ 16 beats in a burst
    input [31:0] type; // burst type
begin
    fork
        write_address_channel(id,addr,size,leng,type);
        write_data_channel(id,addr,size,leng,type);
        write_resp_channel(id);
    join
end
endtask

```



Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 10 )

## AXI BFM: write task (2/3)

```
task write_address_channel;
  input [31:0] id;
  input [WIDTH_AD-1:0] addr;
  input [31:0] size; // 1 ~ 128 byte in a beat
  input [31:0] leng; // 1 ~ 16 beats in a burst
  input [31:0] type; // burst type
begin
  @ (posedge ACLK);
  AWID <= #1 id;
  AWADDR <= #1 addr;
  AWLEN <= #1 leng-1;
  AWLOCK <= #1 'b0;
  AWSIZE <= #1 get_size(size);
  AWBURST <= #1 type[1:0];
  `ifdef AMBA_AXI_PROT
  AWPROT <= #1 'h0; // data, secure, normal
  `endif
  AWVALID <= #1 'b1;
  @ (posedge ACLK);
  while (AWREADY==1'b0) @ (posedge ACLK);
  AWVALID <= #1 'b0;
  @ (negedge ACLK);
end
endtask
```

```
task write_data_channel;
  input [31:0] id;
  input [WIDTH_AD-1:0] addr;
  input [31:0] size; // 1 ~ 128 byte in a beat
  input [31:0] leng; // 1 ~ 16 beats in a burst
  input [31:0] type; // burst type
  reg [WIDTH_AD-1:0] naddr;
  integer idx;
begin
  naddr <= addr;
  @ (posedge ACLK);
  WID <= #1 id;
  WVALID <= #1 'b1;
  for (idx=0; idx<leng; idx=idx+1) begin
    WDATA <= #1 get_data(addr, naddr, size);
   WSTRB <= #1 get_strb(naddr, size);
    WLAST <= #1 (idx==(leng-1));
    naddr <= get_next_addr(naddr, size, type);
    @ (posedge ACLK);
    while (WREADY==1'b0) @ (posedge ACLK);
  end
  WLAST <= #1 'b0;
  WVALID <= #1 'b0;
  @ (negedge ACLK);
end
endtask
```

Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 11 )

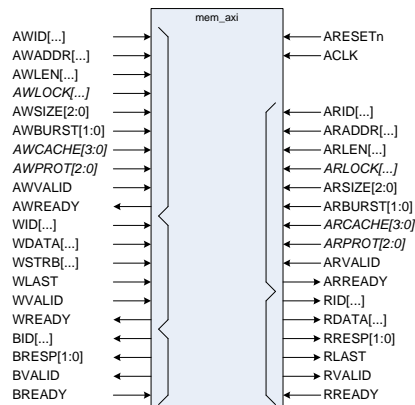
## AXI BFM: write task (3/3)

```
task write_resp_channel;
  input [31:0] id;
begin
  BREADY <= #1 'b1;
  @ (posedge ACLK);
  while (BVALID==1'b0) @ (posedge ACLK);
  if (id!=BID) begin
    $display($time, "%m Error id mis-match for write-resp-
channel 0x%x/0x%x", id, BID);
  end else begin
    case (BRESP)
    2'b00: begin
      `ifdef DEBUG
      $display($time, "%m OK response for write-resp-channel:
OKAY");
      `endif
    end
    2'b01: $display($time, "%m OK response for write-resp-
channel: EXOKAY");
    2'b10: $display($time, "%m Error response for write-resp-
channel: SLVERR");
    2'b11: $display($time, "%m Error response for write-resp-
channel: DECERR");
    endcase
  end
  BREADY <= #1 'b0;
  @ (negedge ACLK);
end
endtask
```

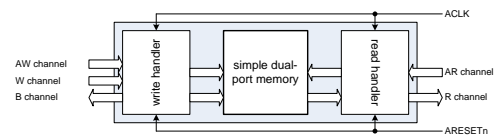
Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 12 )

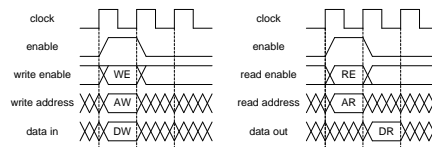
## AXI Memory



### Internal structure



### Internal memory timing



Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 13 )

## AXI Memory

```
*timescale 1ns/1ns

#include "mem_axi_dpram_sync.v"

module mem_axi #(parameter AXI_WIDTH_CID= 4 // Channel ID width in bits
, AXI_WIDTH_ID = 4 // ID width in bits
, AXI_WIDTH_AD =32 // address width
, AXI_WIDTH_DA =32 // data width
, AXI_WIDTH_DS =(AXI_WIDTH_DA/8) // data strobe width
, AXI_WIDTH_DSB=clogb2(AXI_WIDTH_DS) // data strobe width
, AXI_WIDTH_SID=(AXI_WIDTH_CID+AXI_WIDTH_ID)
, ADDR_LENGTH =12 // effective addre bits
)
(
    input wire          ARESETn
    , input wire          ACLK
    , input wire [AXI_WIDTH_SID-1:0] AWID
    , input wire [AXI_WIDTH_AD-1:0] AWADDR
    ... ..
);
```

Note how to make slave ID

Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 14 )

## Simulation with ModelSim (1/4)

```
# Makefile
SHELL = /bin/sh
MAKEFILE = Makefile

#-----
VLIB = $(shell which vlib)
VLOG = $(shell which vlog)
VSIM = $(shell which vsim)
WORK = work

#-----
TOP = top

#-----
all: vlib compile simulate

vlib:
    if [ -d $(WORK) ]; then /bin/rm -rf $(WORK); fi
    $(VLIB) $(WORK)

compile:
    $(VLOG) -lint -work $(WORK) -f modelsim.args

simulate: compile
    $(VSIM) -novopt -c -do "run -all; quit" $(WORK).$(TOP)
```

Modelsim commands

Specify where to store compile results

Compilation

Simulation

```
@ECHO OFF
REM RunMe.bat
SET MODELSIMWORK=work
SET MODELSIMVLIB=vlib
SET MODELSIMVSIM=vsim
SET MODELSIMVCOM=vcom
SET MODELSIMVLOG=vlog

SET DESIGNTOP=top

IF EXIST %MODELSIMWORK% RMDIR /S/Q %MODELSIMWORK%

%MODELSIMVLIB% %MODELSIMWORK%
%MODELSIMVLOG% -work %MODELSIMWORK% -lint^
-f modelsim.args
%MODELSIMVSIM% -novopt -c -do "run -all; quit"^
%MODELSIMWORK%.%DESIGNTOP%
```

## Simulation with ModelSim (2/4)

```
//-----
+incdir+../design/verilog
./sim_define.v
../design/verilog/bfm_axi.v
../design/verilog/mem_axi.v
//-----
// Below are test-bench
//-----
+incdir+../bench/verilog
../bench/verilog/top.v
//-----

`ifndef _SIM_DEFINE_V_
`define _SIM_DEFINE_V_
//-----
`define SIM // define this for simulation case if you are
not sure
`define VCD // define this for VCD waveform dump
`define DEBUG
`define RIGOR
`define LOW_POWER
//-----
`endif
```

modelsim.args

sim\_define.v

top

bfm\_axi

mem\_axi



## Simulation with ModelSim (3/4)

[illegible]

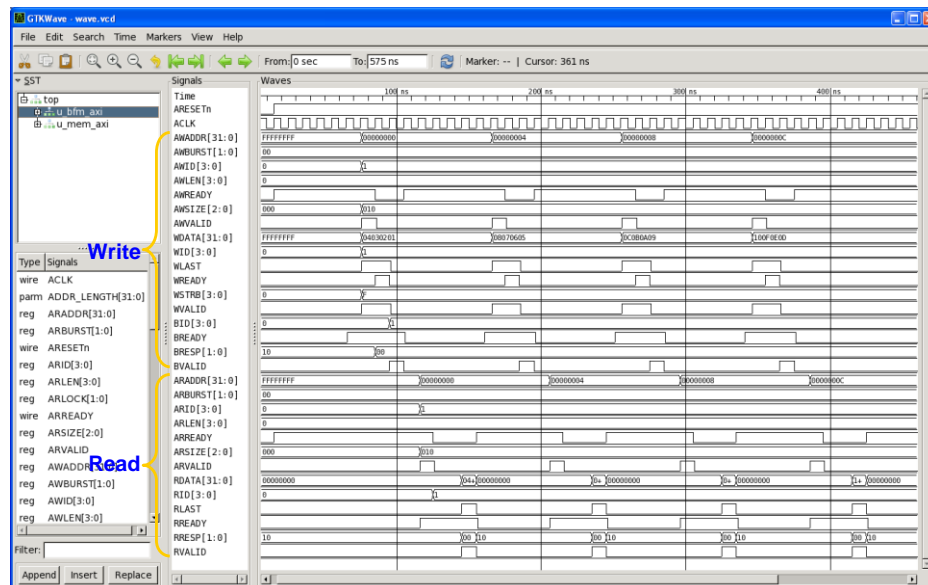
## Compilation

### Simulation according to the scenario

Copyright © 2013-2017 by Ando Ki

**AXI BFM Task ( 17 )**

## Simulation with ModelSim (4/4)



Copyright © 2013-2017 by Ando Ki

AXI BFM Task ( 18 )

## Example: AXI BFM task-based case

🚧 This example shows how to use BFM with tasks

- ◆ Step 1: go to your project directory
  - ❏ [user@host] cd \$(PROJECT)/codes/bfm\_axi\_task
- ◆ Step 2: see the codes
  - ❏ [Some tasks in "\\$\(PROJECT\)/codes/bfm\\_axi\\_task/bench/verilog/bfm\\_axi\\_tasks.v" should be filled](#)
  - ❏ [user@host] cd \$(PROJECT)/codes/bfm\_axi\_task/desing/verilog
  - ❏ [user@host] cd \$(PROJECT)/codes/bfm\_axi\_task/bench/verilog
- ◆ Step 3: compile and run
  - ❏ [user@host] cd \$(PROJECT)/codes/bfm\_axi\_task/sim/modelsim
  - ❏ [user@host] make
- ◆ Step 4: waveform view
  - ❏ [user@host] gtkwave wave.vcd &

```
[user@host] cd $(PROJECT)/codes/bfm_axi_task/sim/modelsim
[user@host] make
[user@host] gtkwave wave.vcd &
```