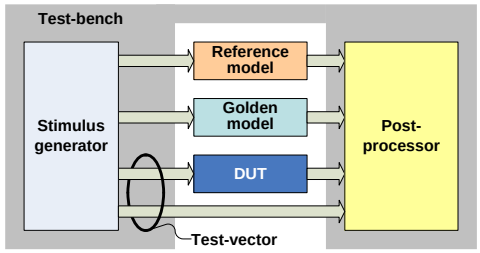
AMBA AHB BFM Design

2013 – 2017

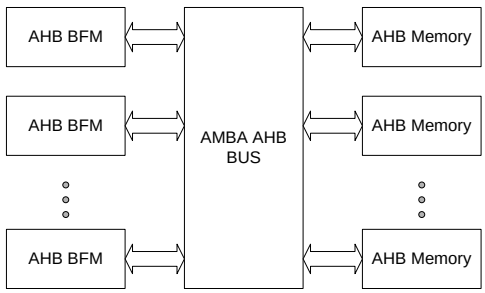
Ando Ki
(adki@future-ds.com)

General form of hardware test/verification



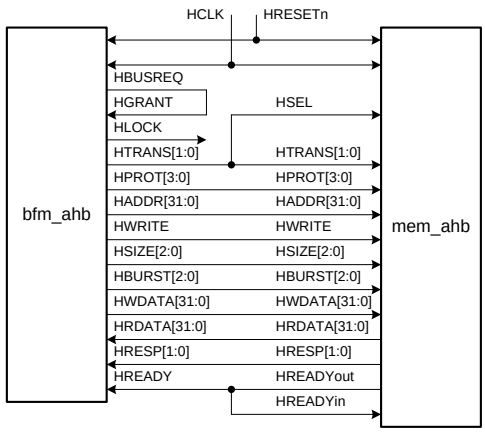
- A test-bench is a layer of code that is created to apply input patterns (stimulus) to the DUT (design under test) and to determine whether the DUT produces the outputs expected.
- A test-vector is a set of values for all the expected input ports (stimuli) and expected values for the output ports of a module under test.
- A test-bench that is created to apply inputs, sample the outputs of the DUT, and compare the outputs with the expected (golden) results is called a self-checking test-bench.

AMBA AHB bus design & verification



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

Preparing BFM and MEMORY



- Note that only one master, i.e., bfm_ahb
- Note that only one slave, i.e., mem_ahb
- Note that how 'HSEL' is connected
- Note that how 'HREADYin' and 'HREADYout' are connected.

AHB BFM: module

```
`timescale 1ns/1ns

module bfm_ahb #(parameter START_ADDR=0
                 , DEPTH_IN_BYTES=32'h100
                 , END_ADDR=START_ADDR+DEPTH_IN_BYTES-1)
(
     input  wire       HRESETn
   , input  wire       HCLK
   , output reg        HBUSREQ
   , input  wire       HGRANT
   , output reg  [31:0] HADDR
   , output reg  [3:0]  HPROT
   , output reg        HLOCK
   , output reg  [1:0]  HTRANS
   , output reg        HWRITE
   , output reg  [2:0]  HSIZE
   , output reg  [2:0]  HBURST
   , output reg  [31:0] HWDATA
   , input  wire [31:0] HRDATA
   , input  wire [1:0]  HRESP
   , input  wire       HREADY
   , input  wire       IRQ
);
```

specify test range

AHB BFM: scenario

```
initial begin
    HBUSREQ = 0;
    HADDR   = 0;
    HPROT   = 0;
    HLOCK   = 0;
    HTRANS  = 0;
    HWRITE  = 0;
    HSIZE   = 0;
    HBURST  = 0;
    HWDATA  = 0;
    while  (HRESETn===1'bx) @ (posedge HCLK);
    while  (HRESETn===1'b1) @ (posedge HCLK);
    while  (HRESETn===1'b0) @ (posedge HCLK);
    `ifdef SINGLE_TEST
    repeat (3) @ (posedge HCLK);
    memory_test(START_ADDR, END_ADDR, 1);
    memory_test(START_ADDR, END_ADDR, 2);
    memory_test(START_ADDR, END_ADDR, 4);
    `endif
    `ifdef BURST_TEST
    repeat (5) @ (posedge HCLK);
    ... ...
    memory_test_burst(START_ADDR, END_ADDR, 4);
    ... ...
    memory_test_burst(START_ADDR, END_ADDR, 16);
    memory_test_burst(START_ADDR, END_ADDR, 32);
    repeat (5) @ (posedge HCLK);
    `endif
    $finish(2);
  end
```

single test

burst test

AHB BFM: scenario

```
//-------------------------------------------------------
 // Test scenario comes here.
 task memory_test;
     input [31:0] start;  // start address
     input [31:0] finish; // end address
     input [2:0]  size;   // data size: 1, 2, 4
     //------------------
     integer i, error;
     reg [31:0] data, gen, got;
     reg [31:0] reposit[START_ADDR:END_ADDR];
     begin
        $display("%m: read-after-write test with %d-byte access", size);
        error = 0;
        gen = $random(7);
        for (i=start; i<(finish-size+1); i=i+size) begin
           gen = $random&~32'b0;
           data = align(i, gen, size);
           ahb_write(i, size, data);
           ahb_read(i, size, got);
           got = align(i, got, size);
           if (got!==data) begin
             $display("[%10d] %m A:%x D:%x, but %x expected", $time, i, got, data);
             error = error+1;
           end
        end
        if (error==0)
             $display("[%10d] %m OK: from %x to %x", $time, start, finish);
```

single read-after-write

AHB BFM: scenario

```
//-------------------------------------------------------------
        $display("%m read-all-after-write-all with %d-byte access", size);
        error = 0;
        gen = $random(1);
        for (i=start; i<(finish-size+1); i=i+size) begin
           gen = {$random} & ~32'b0;
           data = align(i, gen, size);
           reposit[i] = data;
           ahb_write(i, size, data);
        end
        for (i=start; i<(finish-size+1); i=i+size) begin
           data = reposit[i];
           ahb_read(i, size, got);
           got = align(i, got, size);
           if (got!==data) begin
             $display("[%10d] %m A:%x D:%x, but %x expected", $time, i, got, data);
             error = error+1;
           end
        end
        if (error==0)
             $display("[%10d] %m OK: from %x to %x", $time, start, finish);
     end
 endtask
```

single read-all after write-all

AHB BFM: scenario

```
task memory_test_burst;
      input [31:0] start; // start address
      input [31:0] finish;   // end address
      input [7:0]  leng;  // burst length
      integer i, j, k, r, error;
      reg [31:0] data, gen, got;
      reg [31:0] reposit[0:1023];
      integer seed;
      begin
        $display("%m: read-all-after-write-all burst test with %d-beat access", leng);
        error = 0;
        seed  = 111;
        gen = $random(seed);
        k = 0;
        if (finish>(start+leng*4)) begin
          for (i=start; i<(finish-(leng*4)+1); i=i+leng*4) begin
            for (j=0; j<leng; j=j+1) begin
              data_burst[j] = $random;
              reposit[j+k*leng] = data_burst[j];
            end
            @ (posedge HCLK);
            ahb_write_burst(i, leng);
            k = k+1;
          end
```

write part of read-all after write-all

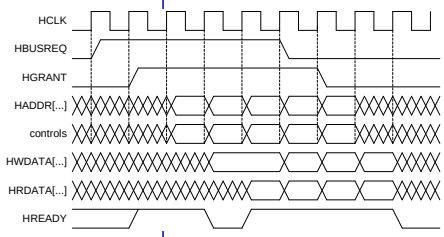
AHB BFM: scenario

```
        gen = $random(seed);
        k = 0;
        for (i=start; i<(finish-(leng*4)+1); i=i+leng*4) begin
          @ (posedge HCLK);
          ahb_read_burst(i, leng);
          for (j=0; j<leng; j=j+1) begin
            if (data_burst[j] != reposit[j+k*leng]) begin
              error = error+1;
              $display("%m A=%hh D=%hh, but %hh expected",
                    i+j*leng, data_burst[j], reposit[j+k*leng]);
            end
          end
          k = k+1;
          r = $random&8'h0F;
          repeat (r) @ (posedge HCLK);
        end
        if (error==0)
          $display("%m %d-length burst read-after-write OK: from %hh to %hh",
                leng, start, finish);
      end else begin
        $display("%m %d-length burst read-after-write from %hh to %hh ???",
                leng, start, finish);
      end
    end
  endtask
```

read and check part of read-all after write-all

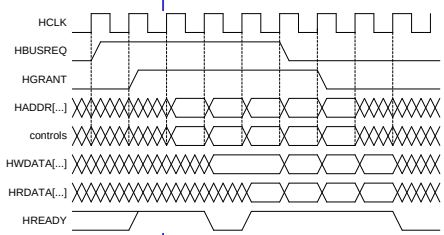
AHB BFM: ahb_read task

```
task ahb_read;
input  [31:0] address;
input  [ 2:0] size;
output [31:0] data;
begin
   @ (posedge HCLK);
   HBUSREQ <= #1 1'b1;
   @ (posedge HCLK);
   while ((HGRANT!==1'b1)||(HREADY!==1'b1)) @ (posedge HCLK);
   HBUSREQ <= #1 1'b0;
   HADDR   <= #1 address;
   HPROT   <= #1 4'b0001; //`HPROT_DATA
   HTRANS  <= #1 2'b10;  //`HTRANS_NONSEQ;
   HBURST  <= #1 3'b000; //`HBURST_SINGLE;
   HWRITE  <= #1 1'b0;   //`HWRITE_READ;
   case (size)
   1:  HSIZE <= #1 3'b000; //`HSIZE_BYTE;
   2:  HSIZE <= #1 3'b001; //`HSIZE_HWORD;
   4:  HSIZE <= #1 3'b010; //`HSIZE_WORD;
   default: $display($time,, "ERROR: unsupported transfer size: %d-byte", size);
   endcase
   @ (posedge HCLK);
   while (HREADY!==1'b1) @ (posedge HCLK);
   HTRANS <= #1 2'b0;
   @ (posedge HCLK);
   while (HREADY===0) @ (posedge HCLK);
   data = HRDATA; // must be blocking
   if (HRESP!=2'b00)   $display($time,, "ERROR: non OK response for read");
   @ (posedge HCLK);
end
endtask
```



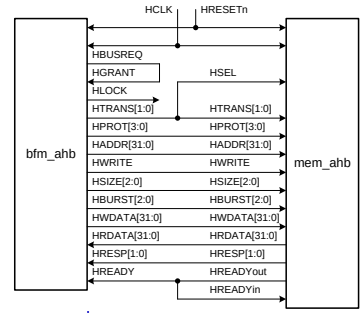
AHB BFM: ahb_write task

```
task ahb_write;
input  [31:0] address;
input  [ 2:0] size;
input  [31:0] data;
begin
   @ (posedge HCLK);
   HBUSREQ <= #1 1;
   @ (posedge HCLK);
   while ((HGRANT!==1'b1)||(HREADY!==1'b1)) @ (posedge HCLK);
   HBUSREQ <= #1 1'b0;
   HADDR   <= #1 address;
   HPROT   <= #1 4'b0001; //`HPROT_DATA
   HTRANS  <= #1 2'b10;  //`HTRANS_NONSEQ;
   HBURST  <= #1 3'b000; //`HBURST_SINGLE;
   HWRITE  <= #1 1'b1;   //`HWRITE_WRITE;
   case (size)
   1:  HSIZE <= #1 3'b000; //`HSIZE_BYTE;
   2:  HSIZE <= #1 3'b001; //`HSIZE_HWORD;
   4:  HSIZE <= #1 3'b010; //`HSIZE_WORD;
   default: $display($time,, "ERROR: unsupported transfer size: %d-byte", size);
   endcase
   @ (posedge HCLK);
   while (HREADY!==1) @ (posedge HCLK);
   HWDATA <= #1 data;
   HTRANS <= #1 2'b0;
   @ (posedge HCLK);
   while (HREADY===0) @ (posedge HCLK);
   if (HRESP!=2'b00)  $display($time,, "ERROR: non OK response write");
   @ (posedge HCLK);
end
endtask
```



AHB BFM: test-bench

```
module top ;
  //-------------------------------------------------
  localparam SIZE_IN_BYTES=`SIZE_IN_BYTES // memory size
           , DELAY        =`MEM_DELAY; // access delay if any for AMBA3/4
  //-------------------------------------------------
  reg        HRESETn= 1'b0;
  reg        HCLK   = 1'b0;
  wire       HBUSREQ;
  wire       HGRANT = HBUSREQ; // no arbiter
  wire [31:0] HADDR  ;
  wire [3:0]  HPROT  ;
  wire       HLOCK  ;
  wire [1:0]  HTRANS ;
  wire       HWRITE ;
  wire [2:0]  HSIZE  ;
  wire [2:0]  HBURST ;
  wire [31:0] HWDATA ;
  wire [31:0] HRDATA ;
  wire [1:0]  HRESP  ;
  wire       HREADY ;
  wire       HSEL   = HTRANS[1]; // no address decoder
  //-------------------------------------------------
```



AHB BFM: test-bench

```
bfm_ahb #(.START_ADDR(0),.DEPTH_IN_BYTES(32'h100))
  u_bfm_ahb (
      .HRESETn (HRESETn)
    , .HCLK    (HCLK   )
    , .HBUSREQ (HBUSREQ)
    , .HGRANT  (HGRANT )
    , .HADDR   (HADDR  )
    , .HPROT   (HPROT  )
    , .HLOCK   (HLOCK  )
    , .HTRANS  (HTRANS )
    , .HWRITE  (HWRITE )
    , .HSIZE   (HSIZE  )
    , .HBURST  (HBURST )
    , .HWDATA  (HWDATA )
    , .HRDATA  (HRDATA )
    , .HRESP   (HRESP  )
    , .HREADY  (HREADY )
    , .IRQ     (1'b0   )
  );
```

```
mem_ahb #(.SIZE_IN_BYTES(SIZE_IN_BYTES),.DELAY(DELAY))
  u_mem_ahb (
      .HRESETn   (HRESETn)
    , .HCLK      (HCLK   )
    , .HSEL      (HSEL   )
    , .HADDR     (HADDR  )
    , .HTRANS    (HTRANS )
    , .HWRITE    (HWRITE )
    , .HSIZE     (HSIZE  )
    , .HBURST    (HBURST )
    , .HWDATA    (HWDATA )
    , .HRDATA    (HRDATA )
    , .HRESP     (HRESP  )
    , .HREADYin  (HREADY )
    , .HREADYout (HREADY )
  );
```

AHB BFM: test-bench

```
//-------------------------------------------------------
  localparam CLK_FREQ=`CLK_FREQ;
  localparam CLK_PERIOD_HALF=1000000000/(CLK_FREQ*2);
  //-------------------------------------------------------
  always #CLK_PERIOD_HALF HCLK <= ~HCLK;
  //-------------------------------------------------------
  real stamp_x, stamp_y, delta;
  initial begin
     HRESETn <= 1'b0;
     repeat (5) @ (posedge HCLK);
     `ifdef RIGOR
      @ (posedge HCLK);
      @ (posedge HCLK); stamp_x = $time;
      @ (posedge HCLK); stamp_y = $time; delta = stamp_y - stamp_x;
      @ (negedge HCLK); $display("%m HCLK %f nsec %f Mhz", delta, 1000.0/delta);
     `endif
     repeat (5) @ (posedge HCLK);
     HRESETn <= 1'b1;
  end
  //-------------------------------------------------------
  `ifdef VCD
  initial begin
     $dumpfile("wave.vcd");
     $dumpvars(0);
  end
  `endif
endmodule
```


AHB MEMORY

```
module mem_ahb #(parameter SIZE_IN_BYTES=1024
                         , DELAY=0
                         , INIT=0)
(
      input  wire         HRESETn
    , input  wire         HCLK
    , input  wire         HSEL
    , input  wire  [31:0]  HADDR
    , input  wire  [ 1:0]  HTRANS
    , input  wire         HWRITE
    , input  wire  [ 2:0]  HSIZE
    , input  wire  [ 2:0]  HBURST
    , input  wire  [31:0]  HWDATA
    , output reg   [31:0]  HRDATA
    , output wire  [ 1:0]  HRESP
    , input  wire         HREADYin
    , output reg          HREADYout
);
  //-------------------------------------------------------------------
  assign HRESP = 2'b00;
  //-------------------------------------------------------------------
  localparam ADD_WIDTH = clogb2(SIZE_IN_BYTES);
  localparam NUM_WORDS = SIZE_IN_BYTES/4;
  //-------------------------------------------------------------------
  reg  [31:0]         mem[0:NUM_WORDS-1];
  reg  [ADD_WIDTH-1:0] T_ADDR, T_ADDRw;
  reg  [31:0]         T_DATA;
  reg  [ 3:0]         T_BE, T_BE_D;
  reg                T_WR, T_WR_D;
  wire               T_ENABLED = HSEL && HREADYin && HTRANS[1];
```

AHB MEMORY

```
always @ (posedge HCLK or negedge HRESETn) begin
   if (HRESETn==0) begin
      HRDATA    <= ~32'h0;
      ... ...
   end else begin
      if (T_ENABLED) begin
         T_ADDR <= HADDR[ADD_WIDTH-1:0];
         T_BE   <= byte_enable(HADDR[1:0], HSIZE);
         T_WR   <= HWRITE;
         HRDATA <= mem[HADDR[ADD_WIDTH-1:2]];
      end else begin
         T_BE   <= 4'h0;
         T_WR   <= 1'b0;
      end
      if (T_WR) begin
         T_DATA[ 7: 0] <= (T_BE[0]) ? HWDATA[ 7: 0]  : HRDATA[ 7: 0];
         T_DATA[15: 8] <= (T_BE[1]) ? HWDATA[15: 8] : HRDATA[15:8];
         T_DATA[23:16] <= (T_BE[2]) ? HWDATA[23:16] : HRDATA[23:16];
         T_DATA[31:24] <= (T_BE[3]) ? HWDATA[31:24] : HRDATA[31:24];
         T_BE_D <= T_BE;
         T_WR_D <= T_WR;
      end else begin
         T_BE_D <= 4'h0;
         T_WR_D <= 1'h0;
      end
      if (T_WR_D) begin
         mem[T_ADDRw[ADD_WIDTH-1:2]] <= T_DATA;
      end
      T_ADDRw <= T_ADDR;
   end
end
```


AHB MEMORY

```
//------------------------------------------------------------------------
reg [5:0] count;
reg       state;
localparam IDLE = 0,
           WAIT = 1;
always @ (posedge HCLK or negedge HRESETn) begin
   if (HRESETn==0) begin
      HREADYout <= 1'b1;
      count     <= 'h0;
      state     <= IDLE;
   end else begin
      case (state)
         IDLE: begin
            if (T_ENABLED&&(DELAY!=0)) begin
               HREADYout <= 1'b0;
               count     <= 'h1;
               state     <= WAIT;
            end
         end
         WAIT: begin
            if ((DELAY==count)||(count=='h0)) begin
               HREADYout <= 1'b1;
               count     <= 'h0;
               state     <= IDLE;
            end else begin
               count <= count + 1;
            end
         end
      endcase
    end
end
```

AHB MEMORY

```
  //------------------------------------------------------------------------
   function [3:0] byte_enable;
      input [1:0] add;   // address offset
      input [2:0] size;  // transfer size
      reg   [3:0] be;
      begin
         case ({size,add})
`ifdef ENDIAN_BIG
            5'b010_00: be = 4'b1111; // word
            5'b001_00: be = 4'b1100; // halfword
            5'b001_10: be = 4'b0011; // halfword
            5'b000_00: be = 4'b1000; // byte
            5'b000_01: be = 4'b0100; // byte
            5'b000_10: be = 4'b0010; // byte
            5'b000_11: be = 4'b0001; // byte
`else // little-endian -- default
            5'b010_00: be = 4'b1111; // word
            5'b001_00: be = 4'b0011; // halfword
            5'b001_10: be = 4'b1100; // halfword
            5'b000_00: be = 4'b0001; // byte
            5'b000_01: be = 4'b0010; // byte
            5'b000_10: be = 4'b0100; // byte
            5'b000_11: be = 4'b1000; // byte
`endif
            default: be = 4'b0;
         endcase
         byte_enable = be;
      end
   endfunction
```


AHB MEMORY

```
    //-----------------------------------------------------
    function integer clogb2;
    input [31:0] value;
    reg   [31:0] tmp, rt;
    begin
         tmp = value - 1;
         for (rt=0; tmp>0; rt=rt+1) tmp=tmp>>1;
         clogb2 = rt;
    end
    endfunction
   //------------------------------------------------------------------------
   // synthesis translate_off
   integer xxy;
   initial begin
        if (INIT) begin
          for (xxy=0; xxy<NUM_WORDS; xxy=xxy+1) begin
              mem[xxy] = xxy;
          end
        end
   end
   // synthesis translate_on
   //------------------------------------------------------------------------
endmodule
```

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%
```

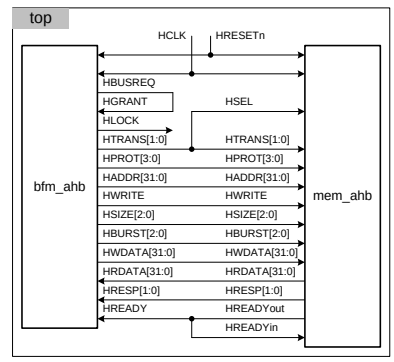

Simulation with ModelSim (2/4)

modelsim.args

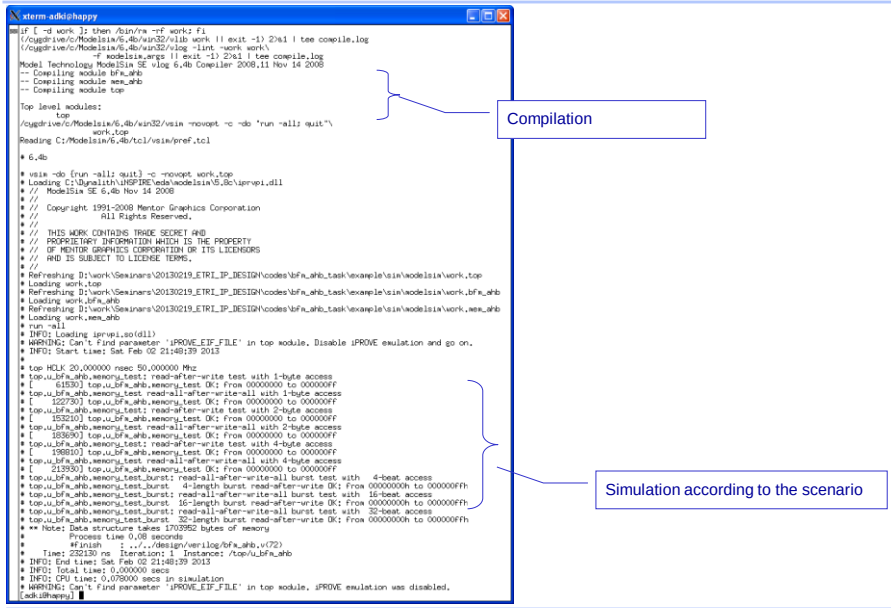
```
//------------------------------------------------------------
+incdir+../../design/verilog
        ./sim_define.v
        ../../design/verilog/bfm_ahb.v
        ../../design/verilog/mem_ahb.v
//------------------------------------------------------------
// Below are test-bench
//------------------------------------------------------------
+incdir+../../bench/verilog
        ../../bench/verilog/top.v
//------------------------------------------------------------
```

sim_define.v

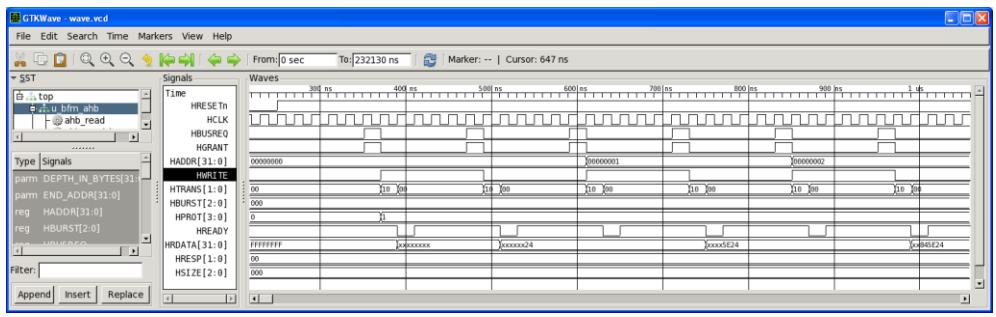
```
`ifndef _SIM_DEFINE_V_
`define _SIM_DEFNE_V_
//------------------------------------------------------------
`define SIM      // define this for simulation case if you are
not sure
`define VCD      // define this for VCD waveform dump
`undef  DEBUG
`define RIGOR
`define LOW_POWER
//------------------------------------------------------------
`define CLK_FREQ   50000000
`define MEM_DELAY  1
//------------------------------------------------------------
`define SINGLE_TEST
`define BURST_TEST
//------------------------------------------------------------
`endif
```



Simulation with ModelSim (3/4)



Simulation with ModelSim (4/4)



Example: AHB 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_ahb_task
  - Step 2: see the codes
    - [user@host] cd $(PROJECT)/codes/bfm_ahb_task/desing/verilog
  - Step 3: compile and run
    - [user@host] cd $(PROJECT)/codes/bfm_ahb_task/sim/modelsim
    - [user@host] make
  - Step 4: waveform view
    - [user@host] gtkwave wave.vcd &

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


Issues and quiz

- Wrapping cases
- Non-OK response cases
- Early-termination cases

References

- AMBA Specification, Rev 2.0, ARM Limited.
- AHB-Lite Overview, ARM Limited, 2001.
- Multi-layer AHB Overview, ARM Limited, 2001.