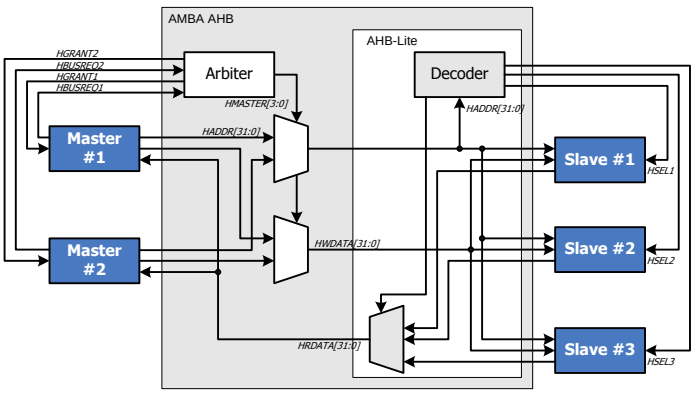
AMBA AHB Design

2013 – 2017

Ando Ki
(adki@future-ds.com)

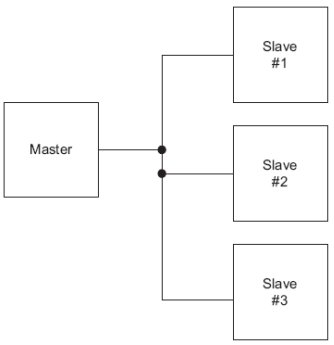
AMBA AHB and AHB-Lite

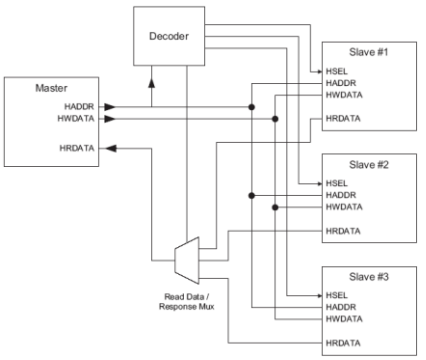
- AHB-Lite
  - Single-master and scale-down version of AHB
- AMBA AHB
  - Multi-master and full-scale versin of AHB



AMBA AHB-Lite

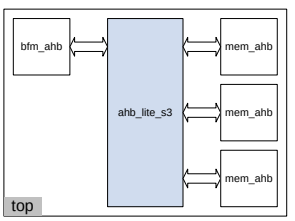
- AHB-Lite is a subset of the full AHB specification, where only a single AHB master is used.
  - A single master → No master-to-slave multiplexor
  - No request/grant protocol to the arbiter → No arbiter
  - No split/retry responses from slaves





AHB-Lite for three-slave case

```
`timescale 1ns/1ns
`include "ahb_decoder_s3.v"
`include "ahb_s2m_s3.v"
`include "ahb_default_slave.v"

module ahb_lite_s3
      #(parameter P_HSEL0_START = 16'h0000, P_HSEL0_SIZE  = 16'h0100,
                  P_HSEL1_START = 16'h1000, P_HSEL1_SIZE  = 16'h0100,
                  P_HSEL2_START = 16'h2000, P_HSEL2_SIZE  = 16'h0100)
(
      input  wire        HRESETn
    , input  wire        HCLK
    , input  wire  [31:0] M_HADDR
    , input  wire  [1:0]  M_HTRANS
    , input  wire        M_HWRITE
    , input  wire  [2:0]  M_HSIZE
    , input  wire  [2:0]  M_HBURST
    , input  wire  [3:0]  M_HPROT
    , input  wire  [31:0] M_HWDATA
    , output wire  [31:0] M_HRDATA
    , output wire  [1:0]  M_HRESP
    , output wire        M_HREADY
    , output wire        HWRITE
    , output wire  [31:0] HADDR
    , output wire  [1:0]  HTRANS
    , output wire  [2:0]  HSIZE
    , output wire  [2:0]  HBURST
    , output wire  [3:0]  HPROT
    , output wire  [31:0] HWDATA
    , output wire        HREADY
```




```
    , output wire        HSEL0
    , input  wire  [1:0]  HRESP0
    , input  wire  [31:0] HRDATA0
    , input  wire        HREADY0
    , output wire        HSEL1
    , input  wire  [1:0]  HRESP1
    , input  wire  [31:0] HRDATA1
    , input  wire        HREADY1
    , output wire        HSEL2
    , input  wire  [1:0]  HRESP2
    , input  wire  [31:0] HRDATA2
    , input  wire        HREADY2
    , input  wire        REMAP
);
```

In order to support bus-remap feature

AHB-Lite for three-slave case

```
/*********************************************************/
  wire HSELd; // default slave
  wire [31:0] HRDATAd;
  wire [1:0]  HRESPd;
  wire        HREADYd;
  /*********************************************************/
  assign HADDR  = M_HADDR;
  assign HTRANS = M_HTRANS;
  assign HSIZE  = M_HSIZE;
  assign HBURST = M_HBURST;
  assign HWRITE = M_HWRITE;
  assign HPROT  = M_HPROT;
  assign HWDATA = M_HWDATA;
  assign HREADY = M_HREADY;
  /*********************************************************/
  defparam
  Uahb_decoder.P_NUM        = 3, // how many slaves
  Uahb_decoder.P_ADDR_START0 = P_HSEL0_START,
  Uahb_decoder.P_ADDR_SIZE0  = P_HSEL0_SIZE ,
  Uahb_decoder.P_ADDR_START1 = P_HSEL1_START,
  Uahb_decoder.P_ADDR_SIZE1  = P_HSEL1_SIZE ,
  Uahb_decoder.P_ADDR_START2 = P_HSEL2_START,
  Uahb_decoder.P_ADDR_SIZE2  = P_HSEL2_SIZE ;
  ahb_decoder_s3 Uahb_decoder (
            .HADDR(M_HADDR),
            .HSELd(HSELd), // default
            .HSEL0(HSEL0),
            .HSEL1(HSEL1),
            .HSEL2(HSEL2),
            .REMAP(REMAP));
```

decoder selects one of three slaves or default slave.

AHB-Lite for three-slave case

```
/*********************************************************/
  ahb_s2m_s3 Uahb_s2m (
            .HRESETn(HRESETn),
            .HCLK   (HCLK),
            .HSEL0  (HSEL0),
            .HSEL1  (HSEL1),
            .HSEL2  (HSEL2),
            .HSELd  (HSELd),
            .HRDATA(M_HRDATA),
            .HRESP (M_HRESP),
            .HREADY(M_HREADY),
            .HRDATA0(HRDATA0),
            .HRESP0 (HRESP0 ),
            .HREADY0(HREADY0),
            .HRDATA1(HRDATA1),
            .HRESP1 (HRESP1 ),
            .HREADY1(HREADY1),
            .HRDATA2(HRDATA2),
            .HRESP2 (HRESP2 ),
            .HREADY2(HREADY2),
            .HRDATAd(HRDATAd),
            .HRESPd (HRESPd ),
            .HREADYd(HREADYd));
```

Slave to master multiplexer

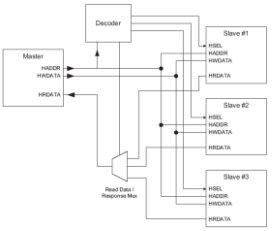
- Default slave takes care of the case no slaves are selected.
  - For NON-SEQ or SEQ transfers, responds with ERROR.
  - For IDLE or BUSY transfers, responds with OKAY.

```
/*********************************************************/
  ahb_default_slave
            Uahb_default_slave (
            .HRESETn(HRESETn),
            .HCLK   (HCLK),
            .HSEL   (HSELd),
            .HADDR  (HADDR),
            .HTRANS (HTRANS),
            .HWRITE (HWRITE),
            .HSIZE  (HSIZE),
            .HBURST (HBURST),
            .HWDATA (HWDATA),
            .HRDATA(HRDATAd),
            .HRESP (HRESPd),
            .HREADYin(HREADY),
            .HREADYout(HREADYd));

  /*********************************************************/
endmodule
```

AHB-Lite for three-slave case: decoder

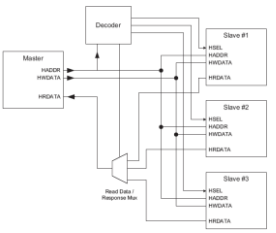
```
`timescale 1ns/1ns

module ahb_decoder_s3 #(parameter P_NUM         = 3, // how many slaves
                        P_ADDR_START0 = 'h0000, P_ADDR_SIZE0  = 'h0010,
                        P_ADDR_START1 = 'h0010, P_ADDR_SIZE1  = 'h0010,
                        P_ADDR_START2 = 'h0020, P_ADDR_SIZE2  = 'h0010)
(
      input   wire [31:0] HADDR
    , output  wire        HSELd // default slave
    , output  wire        HSEL0
    , output  wire        HSEL1
    , output  wire        HSEL2
    , input   wire        REMAP
);
  /*********************************************************/
  localparam P_ADDR_END0 = P_ADDR_START0 + P_ADDR_SIZE0 - 1;
  localparam P_ADDR_END1 = P_ADDR_START1 + P_ADDR_SIZE1 - 1;
  localparam P_ADDR_END2 = P_ADDR_START2 + P_ADDR_SIZE2 - 1;
  /*********************************************************/
  reg ihseld, ihsel0, ihsel1, ihsel2;
  assign HSELd = ihseld;
  assign HSEL0 = (REMAP) ? ihsel1 : ihsel0;
  assign HSEL1 = (REMAP) ? ihsel0 : ihsel1;
  assign HSEL2 = ihsel2;
  /*********************************************************/
```

Swap HSEL0 and HSEL1, when REMAP is 1



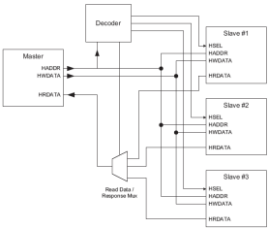
AHB-Lite for three-slave case: decoder

```
/*********************************************************/
  wire [15:0] thaddr = HADDR[31:16];
  always @ (thaddr) begin // must be blocking assignment
    if ((P_NUM>0)&&(thaddr>=P_ADDR_START0)&&(thaddr<=P_ADDR_END0)) ihsel0 <= 1'b1;
    else                                                            ihsel0 <= 1'b0;
    if ((P_NUM>1)&&(thaddr>=P_ADDR_START1)&&(thaddr<=P_ADDR_END1)) ihsel1 <= 1'b1;
    else                                                            ihsel1 <= 1'b0;
    if ((P_NUM>2)&&(thaddr>=P_ADDR_START2)&&(thaddr<=P_ADDR_END2)) ihsel2 <= 1'b1;
    else                                                            ihsel2 <= 1'b0;

    if (((P_NUM>0)&&(thaddr>=P_ADDR_START0)&&(thaddr<=P_ADDR_END0))||
       ((P_NUM>1)&&(thaddr>=P_ADDR_START1)&&(thaddr<=P_ADDR_END1))||
       ((P_NUM>2)&&(thaddr>=P_ADDR_START2)&&(thaddr<=P_ADDR_END2))) ihseld <= 1'b0;
    else                                                            ihseld <= 1'b1;

  end // always
endmodule
```



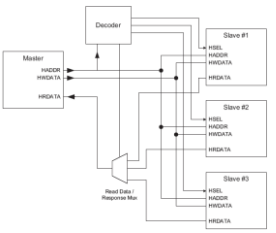
AHB-Lite for three-slave case: multiplexer

```
`timescale 1ns/1ns

module ahb_s2m_s3 (
      input  wire       HRESETn
    , input  wire       HCLK
    , input  wire       HSEL0
    , input  wire       HSEL1
    , input  wire       HSEL2
    , input  wire       HSELd
    , output reg  [31:0] HRDATA
    , output reg  [1:0]  HRESP
    , output reg         HREADY
    , input  wire [31:0] HRDATA0
    , input  wire [1:0]  HRESP0
    , input  wire        HREADY0
    , input  wire [31:0] HRDATA1
    , input  wire [1:0]  HRESP1
    , input  wire        HREADY1
    , input  wire [31:0] HRDATA2
    , input  wire [1:0]  HRESP2
    , input  wire        HREADY2
    , input  wire [31:0] HRDATAd
    , input  wire [1:0]  HRESPd
    , input  wire        HREADYd
);
```



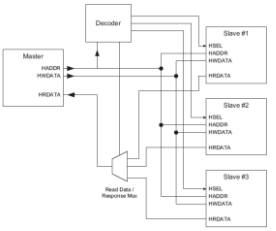
AHB-Lite for three-slave case: multiplexer

```
/*********************************************************/
  localparam D_HSEL0 = 4'b0001;
  localparam D_HSEL1 = 4'b0010;
  localparam D_HSEL2 = 4'b0100;
  localparam D_HSELd = 4'b1000;
  wire [3:0] _hsel = {HSELd,HSEL2,HSEL1,HSEL0};
  reg  [3:0] _hsel_reg;
  always @ (negedge HRESETn or posedge HCLK) begin
    if (~HRESETn)   _hsel_reg <= 'h0;
    else if(HREADY) _hsel_reg <= _hsel; // default HREADY must be 1'b1
  end
  always @ (_hsel_reg or HREADY0 or HREADY1 or HREADY2 or HREADYd)
begin
    case(_hsel_reg) // synopsys full_case parallel_case
      D_HSEL0: HREADY = HREADY0; // default
      D_HSEL1: HREADY = HREADY1;
      D_HSEL2: HREADY = HREADY2;
      D_HSELd: HREADY = HREADYd;
      default: HREADY = 1'b1;
    endcase
  end
  always @ (_hsel_reg or HRDATA0 or HRDATA1 or HRDATA2 or HRDATAd)
begin
    case(_hsel_reg) // synopsys full_case parallel_case
      D_HSEL0: HRDATA = HRDATA0;
      D_HSEL1: HRDATA = HRDATA1;
      D_HSEL2: HRDATA = HRDATA2;
      D_HSELd: HRDATA = HRDATAd;
      default: HRDATA = 32'b0;
    endcase
  end
```



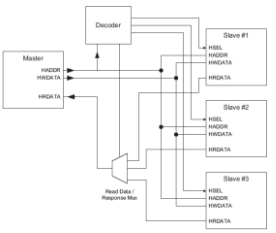
AHB-Lite for three-slave case: multiplexer

```
always @ (_hsel_reg or HRESP0 or HRESP1 or HRESP2 or HRESPd) begin
   case(_hsel_reg) // synopsys full_case parallel_case
     D_HSEL0: HRESP = HRESP0;
     D_HSEL1: HRESP = HRESP1;
     D_HSEL2: HRESP = HRESP2;
     D_HSELd: HRESP = HRESPd;
     default: HRESP = 2'b01; //`HRESP_ERROR;
   endcase
 end

endmodule
```



AHB-Lite for three-slave case: default slave

```
`timescale 1ns/1ns

module ahb_default_slave (
      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  wire  [31:0] HRDATA,
      output  reg   [ 1:0] HRESP,
      input  wire        HREADYin,
      output  reg         HREADYout
);
  /*****************************************************/
  assign HRDATA = 32'h0;
```

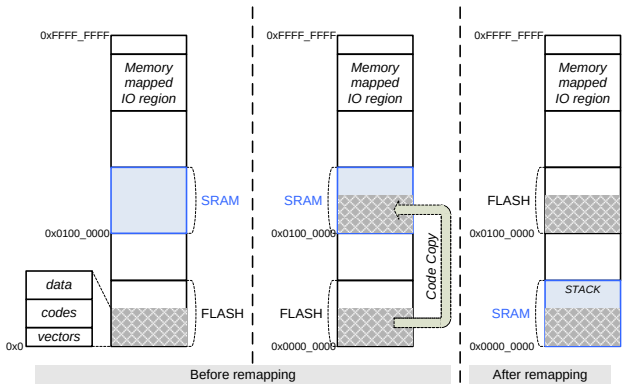


AHB-Lite for three-slave case: default slave

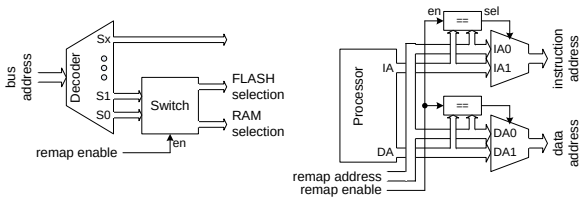
```
/*********************************************************/
    reg [1:0] state;
    localparam STH_IDLE   = 2'h0,
               STH_WRITE  = 2'h1,
               STH_READ0  = 2'h2;
  /*********************************************************/
  always @ (posedge HCLK or negedge HRESETn) begin
    if (HRESETn==0) begin
      HRESP     <= 2'b00; //`HRESP_OKAY;
      HREADYout <= 1'b1;
      state     <= STH_IDLE;
    end else begin // if (HRESETn==0) begin
      case (state)
      STH_IDLE: begin
        if (HSEL && HREADYin) begin
          case (HTRANS)
          2'b00, 2'b01: begin
              HREADYout <= 1'b1;
              HRESP     <= 2'b00; //`HRESP_OKAY;
              state     <= STH_IDLE;
          end // HTRANS_IDLE or HTRANS_BUSY
          2'b10, 2'b11: begin
              HREADYout <= 1'b0;
              HRESP     <= 2'b01; //`HRESP_ERROR;
              if (HWRITE) begin
                state <= STH_WRITE;
              end else begin
                state <= STH_READ0;
              end
          end // HTRANS_NONSEQ or HTRANS_SEQ
          endcase // HTRANS
          endcase // HTRANS
        end else begin// if (HSEL && HREADYin)
          HREADYout <= 1'b1;
          HRESP     <= 2'b00; //`HRESP_OKAY;
        end
        end // STH_IDLE
      STH_WRITE: begin
          HREADYout <= 1'b1;
          HRESP     <= 2'b01; //`HRESP_ERROR;
          state     <= STH_IDLE;
        end // STH_WRITE
      STH_READ0: begin
          HREADYout <= 1'b1;
          HRESP     <= 2'b01; //`HRESP_ERROR;
          state     <= STH_IDLE;
        end // STH_READ0
      endcase // state
    end // if (HRESETn==0)
  end // always

endmodule
```


What is address remap

- In order to achieve the following
  - swap two memory regions
  - alias a memory region into two different address ranges
  - move an address region
  - remove an address region.

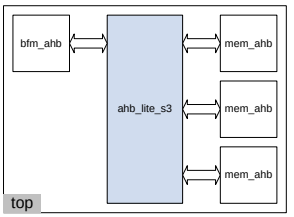


What is address remap



(a) bus remapping (b) processor remapping

AHB-Lite testing



AHB-Lite testing: test-bench

```
`timescale 1ns/1ns

`ifndef CLK_FREQ
`define CLK_FREQ       50000000
`endif
`ifndef MEM_DELAY `define MEM_DELAY 0 `endif
`ifndef SIZE_IN_BYTES `define SIZE_IN_BYTES 1024 `endif

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      M_HBUSREQ;
  wire      M_HGRANT = M_HBUSREQ; // no arbiter
  wire [31:0] M_HADDR     ;
  wire [3:0]  M_HPROT     ;
  wire      M_HLOCK     ;
  wire [1:0]  M_HTRANS    ;
  wire      M_HWRITE    ;
  wire [2:0]  M_HSIZE     ;
  wire [2:0]  M_HBURST    ;
  wire [31:0] M_HWDATA    ;
  wire      M_HREADY    ;
  wire [31:0] M_HRDATA    ;
  wire [1:0]  M_HRESP     ;
```

```
  wire [31:0] S_HADDR     ;
  wire [3:0]  S_HPROT     ;
  wire [1:0]  S_HTRANS    ;
  wire      S_HWRITE    ;
  wire [2:0]  S_HSIZE     ;
  wire [2:0]  S_HBURST    ;
  wire [31:0] S_HWDATA    ;
  wire      S_HREADY    ;
  wire [31:0] S_HRDATA    [0:2];
  wire [1:0]  S_HRESP     [0:2];
  wire      S_HREADYout [0:2];
  wire      S_HSEL      [0:2];
```


AHB-Lite testing: test-bench

```
  bfm_ahb #(.START_ADDR(0),.DEPTH_IN_BYTES(32'h100))
  u_bfm_ahb (
      .HRESETn (HRESETn  )
    , .HCLK    (HCLK     )
    , .HBUSREQ (M_HBUSREQ)
    , .HGRANT  (M_HGRANT )
    , .HADDR   (M_HADDR  )
    , .HPROT   (M_HPROT  )
    , .HLOCK   (M_HLOCK  )
    , .HTRANS  (M_HTRANS )
    , .HWRITE  (M_HWRITE )
    , .HSIZE   (M_HSIZE  )
    , .HBURST  (M_HBURST )
    , .HWDATA  (M_HWDATA )
    , .HRDATA  (M_HRDATA )
    , .HRESP   (M_HRESP  )
    , .HREADY  (M_HREADY )
    , .IRQ     (1'b0   )
  );
  //-----------------------------------------------
```

AHB-Lite testing: test-bench

```
//-----------------------------------------------------
ahb_lite_s3 #(.P_HSEL0_START(16'h0000),.P_HSEL0_SIZE(16'h0100)
              ,.P_HSEL1_START(16'h1000),.P_HSEL1_SIZE(16'h0100)
              ,.P_HSEL2_START(16'h2000),.P_HSEL2_SIZE(16'h0100))
u_ahb_lite (
    .HRESETn (HRESETn )
  , .HCLK    (HCLK    )
  , .M_HADDR (M_HADDR )
  , .M_HTRANS(M_HTRANS)
  , .M_HWRITE(M_HWRITE)
  , .M_HSIZE (M_HSIZE )
  , .M_HBURST(M_HBURST)
  , .M_HPROT (M_HPROT )
  , .M_HWDATA(M_HWDATA)
  , .M_HRDATA(M_HRDATA)
  , .M_HRESP (M_HRESP )
  , .M_HREADY(M_HREADY)
  , .HWRITE  (S_HWRITE  )
  , .HADDR   (S_HADDR   )
  , .HTRANS  (S_HTRANS  )
  , .HSIZE   (S_HSIZE   )
  , .HBURST  (S_HBURST  )
  , .HPROT   (S_HPROT   )
  , .HWDATA  (S_HWDATA  )
  , .HREADY  (S_HREADY  )
  , .HSEL0   (S_HSEL     [0])
  , .HRESP0  (S_HRESP    [0])
  , .HRDATA0 (S_HRDATA   [0])
  , .HREADY0 (S_HREADYout [0])
  , .HSEL1   (S_HSEL     [1])
  , .HRESP1  (S_HRESP    [1])
  , .HRDATA1 (S_HRDATA   [1])
  , .HREADY1 (S_HREADYout [1])
  , .HSEL2   (S_HSEL     [2])
  , .HRESP2  (S_HRESP    [2])
  , .HRDATA2 (S_HRDATA   [2])
  , .HREADY2 (S_HREADYout [2])
  , .REMAP   (1'b0)
);
//-----------------------------------------------------
```


AHB-Lite testing: test-bench

```
generate
genvar GM;
for (GM=0; GM<3; GM=GM+1) begin :BM_BLK
  mem_ahb #(.SIZE_IN_BYTES(SIZE_IN_BYTES),.DELAY(DELAY))
  u_mem_ahb (
      .HRESETn   (HRESETn)
    , .HCLK      (HCLK    )
    , .HADDR     (S_HADDR  )
    , .HTRANS    (S_HTRANS )
    , .HWRITE    (S_HWRITE )
    , .HSIZE     (S_HSIZE  )
    , .HBURST    (S_HBURST )
    , .HWDATA    (S_HWDATA )
    , .HREADYin  (S_HREADY )
    , .HSEL      (S_HSEL     [GM])
    , .HRDATA    (S_HRDATA   [GM])
    , .HRESP     (S_HRESP    [GM])
    , .HREADYout (S_HREADYout[GM])
  );
end
endgenerate
//-----------------------------------------------------
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
```

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%
```

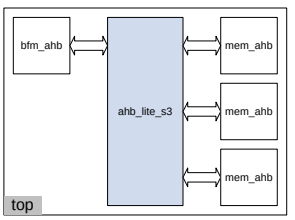

Simulation with ModelSim (2/4)

modelsim.args

```
//-----------------------------------------------------------------
+incdir+../../design/verilog
        ./sim_define.v
        ../../design/verilog/ahb_lite_s3.v
//-----------------------------------------------------------------
+incdir+../../../../bfm_ahb_task/example/design/verilog/
        ../../../../bfm_ahb_task/example/design/verilog/bfm_ahb.v
        ../../../../bfm_ahb_task/example/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
`define DEBUG
`define RIGOR
//-----------------------------------------------------------------
`define CLK_FREQ   50000000
`define MEM_DELAY  0
//-----------------------------------------------------------------
`define  AMBA3
`define  AMBA4
//-----------------------------------------------------------------
`ifdef AMBA4
`define  AMBA3
`endif
//-----------------------------------------------------------------
`endif
```



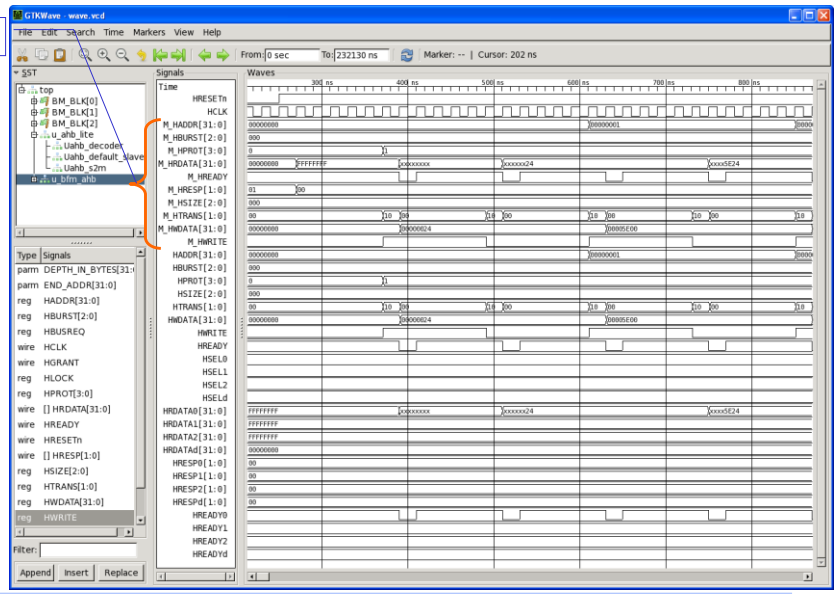
Simulation with ModelSim (3/4)

Compilation

Simulation according to the scenario

Simulation with ModelSim (4/4)

Master ports



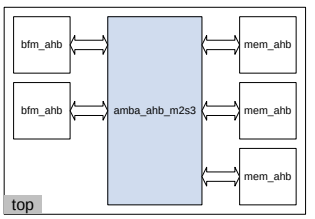
Example: AHB-Lite case

- This example shows how to use BFM with tasks
  - Step 1: go to your project directory
    - [user@host] cd $(PROJECT)/codes/ahb_lite
  - Step 2: see the codes
    - [user@host] cd $(PROJECT)/codes/ahb_lite/desing/verilog
  - Step 3: compile and run
    - [user@host] cd $(PROJECT)/codes/ahb_lite/sim/modelsim
    - [user@host] make
  - Step 4: waveform view
    - [user@host] gtkwave wave.vcd &

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

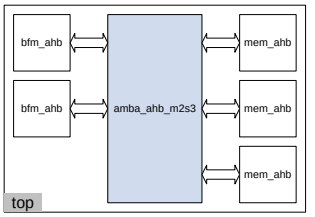

AMBA AHB two-master and three-slave case

```
`timescale 1ns/1ns
`include "ahb_arbiter_m2.v"
`include "ahb_m2s_m2.v"
module amba_ahb_m2s3
#(parameter P_HSEL0_START=16'h0000, P_HSEL0_SIZE=16'h0010,
          P_HSEL1_START=16'h1000, P_HSEL1_SIZE=16'h0010,
          P_HSEL2_START=16'h2000, P_HSEL2_SIZE=16'h0010)
(
  input  wire       HRESETn    ,
  input  wire       HCLK       ,
  input  wire       M_HBUSREQ_0,
  output wire       M_HGRANT_0 ,
  input  wire [31:0] M_HADDR_0  ,
  input  wire [ 1:0] M_HTRANS_0 ,
  input  wire [ 2:0] M_HSIZE_0  ,
  input  wire [ 2:0] M_HBURST_0 ,
  input  wire [ 3:0] M_HPROT_0  ,
  input  wire       M_HWRITE_0 ,
  input  wire [31:0] M_HWDATA_0 ,
  input  wire       M_HBUSREQ_1,
  output wire       M_HGRANT_1 ,
  input  wire [31:0] M_HADDR_1  ,
  input  wire [ 1:0] M_HTRANS_1 ,
  input  wire [ 2:0] M_HSIZE_1  ,
  input  wire [ 2:0] M_HBURST_1 ,
  input  wire [ 3:0] M_HPROT_1  ,
  input  wire       M_HWRITE_1 ,
  input  wire [31:0] M_HWDATA_1 ,
  output wire [31:0] M_HRDATA   ,
  output wire       M_HREADY   ,
  output wire [ 1:0] M_HRESP    ,
```



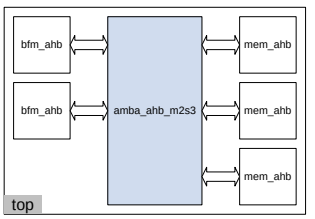
AMBA AHB two-master and three-slave case

```
  output wire [31:0] HADDR      ,
  output wire [31:0] HWDATA     ,
  output wire [ 1:0] HTRANS     ,
  output wire [ 2:0] HSIZE      ,
  output wire [ 2:0] HBURST     ,
  output wire        HWRITE     ,
  output wire [3:0]  HPROT      ,
  output wire        HREADY     ,
  output wire        HSEL_0     ,
  input  wire [31:0] HRDATA_0   ,
  input  wire [ 1:0] HRESP_0    ,
  input  wire        HREADY_0   ,
  output wire        HSEL_1     ,
  input  wire [31:0] HRDATA_1   ,
  input  wire [ 1:0] HRESP_1    ,
  input  wire        HREADY_1   ,
  output wire        HSEL_2     ,
  input  wire [31:0] HRDATA_2   ,
  input  wire [ 1:0] HRESP_2    ,
  input  wire        HREADY_2   ,

  input  wire        REMAP
);
```



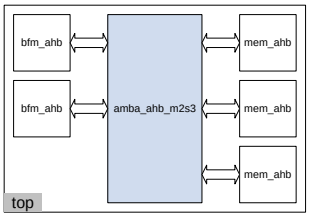
AMBA AHB two-master and three-slave case

```
//-----------------------------------------------
  wire [3:0] M_HMASTER;
  ahb_arbiter_m2 Uahb_arbiter (
      .HRESETn   (HRESETn    ),
      .HCLK      (HCLK       ),
      .HREADY    (M_HREADY   ),
      .HBUSREQ_0 (M_HBUSREQ_0),
      .HBUSREQ_1 (M_HBUSREQ_1),
      .HGRANT_0  (M_HGRANT_0 ),
      .HGRANT_1  (M_HGRANT_1 ),
      .HMASTER   (M_HMASTER  )
  );
  //-----------------------------------------------
  wire  [31:0] M_HADDR;
  wire  [1:0]  M_HTRANS;
  wire  [2:0]  M_HSIZE;
  wire  [2:0]  M_HBURST;
  wire  [3:0]  M_HPROT;
  wire         M_HWRITE;
  wire  [31:0] M_HWDATA;
  ahb_m2s_m2 Uahb_m2s (
      .HRESETn  (HRESETn  ),
      .HCLK     (HCLK     ),
      .HREADY   (M_HREADY   ),
      .HMASTER  (M_HMASTER  ),
      .HADDR    (M_HADDR    ),
      ... ...
  );
```



AMBA AHB two-master and three-slave case

```
//-----------------------------------------------
 ahb_lite_s3 #(.P_HSEL0_START(P_HSEL0_START), .P_HSEL0_SIZE(P_HSEL0_SIZE),
            .P_HSEL1_START(P_HSEL1_START), .P_HSEL1_SIZE(P_HSEL1_SIZE),
            .P_HSEL2_START(P_HSEL2_START), .P_HSEL2_SIZE(P_HSEL2_SIZE))
 Uahb_lite
 (
     .HRESETn (HRESETn),
     .HCLK    (HCLK   ),
     .M_HADDR   (M_HADDR  ),
     .M_HTRANS  (M_HTRANS ),
     .M_HWRITE  (M_HWRITE ),
     .M_HSIZE   (M_HSIZE  ),
     .M_HBURST  (M_HBURST ),
     ... ...
     .HREADY1 (HREADY_1),
     .HSEL2   (HSEL_2  ),
     .HRDATA2 (HRDATA_2),
     .HRESP2  (HRESP_2 ),
     .HREADY2 (HREADY_2),

     .REMAP   (REMAP)
 );

endmodule
```



AMBA AHB two-master: arbiter

```
`timescale 1ns/1ns

module ahb_arbiter_m2 #(parameter P_NUM = 2)
(
     input  wire     HRESETn
   , input  wire     HCLK
   , input  wire     HREADY
   , input  wire     HBUSREQ_0
   , input  wire     HBUSREQ_1
   , output wire      HGRANT_0
   , output wire      HGRANT_1
   , output reg  [3:0] HMASTER
);
```

Pure priority-based arbitration

```
//-----------------------------------------------
   wire [0:1] hbusreq = {HBUSREQ_0,HBUSREQ_1};
   reg  [0:1] hgrant;
   assign {HGRANT_0, HGRANT_1} = hgrant;
   //-----------------------------------------------
   always @ (posedge HCLK or negedge HRESETn) begin
     if (HRESETn==1'b0) hgrant <= 4'h0;
     else begin
        if (HREADY==1'b1) begin
          casex ({hbusreq,hgrant})
            // priority
            4'b1x_00: hgrant <= 2'b10;
            4'b01_00: hgrant <= 2'b01;
            // stay
            4'b1x_10: hgrant <= 2'b10;
            4'bx1_01: hgrant <= 2'b01;
            // last
            4'b00_xx: hgrant <= 2'b00;
            // last and handover
            4'b01_10: hgrant <= 2'b01;
            4'b00_10: hgrant <= 2'b00;
            4'b10_01: hgrant <= 2'b10;
            4'b00_01: hgrant <= 2'b00;
            default : hgrant <= 2'b00;
          endcase
        end
     end
   end
```

AMBA AHB two-master: arbiter

```
   always @ (posedge HCLK or negedge HRESETn)
   begin
      if (HRESETn==1'b0) begin
         HMASTER <= 4'hF;
      end else begin
         if (HREADY==1'b1) begin
            casex (hgrant)
               2'b1x:   HMASTER <= #1 4'h0;
               2'b01:   HMASTER <= #1 4'h1;
               default: HMASTER <= #1 4'hF;
            endcase
         end
      end
   end
// synopsys translate_off
`ifdef RIGOR
  wire [1:0] _hgrant = {HGRANT_0, HGRANT_1};
  always @ (posedge HCLK) begin
    if ((_hgrant!=2'b01)&&
                (_hgrant!=2'b10)&&
                (_hgrant!=2'b00))
       $display($time,, "%m ERROR: more than one has been granted! 0x%x", _hgrant);
  end
`endif
// synopsys translate_on
endmodule
```

Check any arbitration error

AMBA AHB two-master: master to slave mux

```
`timescale 1ns/1ns

module ahb_m2s_m2 #(parameter NUM_MASTER = 3)
(
     input  wire       HRESETn
   , input  wire       HCLK
   , input  wire       HREADY
   , input  wire [3:0]  HMASTER
   , output  reg  [31:0] HADDR
   , output  reg  [3:0]  HPROT
   , 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] HADDR_0
   , input  wire [3:0]  HPROT_0
   , input  wire [1:0]  HTRANS_0
   , input  wire       HWRITE_0
   , input  wire [2:0]  HSIZE_0
   , input  wire [2:0]  HBURST_0
   , input  wire [31:0] HWDATA_0
   , input  wire [31:0] HADDR_1
   , input  wire [3:0]  HPROT_1
   , input  wire [1:0]  HTRANS_1
   , input  wire       HWRITE_1
   , input  wire [2:0]  HSIZE_1
   , input  wire [2:0]  HBURST_1
   , input  wire [31:0] HWDATA_1
);
```

AMBA AHB two-master: master to slave mux

```
reg [3:0] hmaster_delay;
always @ (posedge HCLK or negedge HRESETn)
begin
   if (HRESETn==1'b0) begin
       hmaster_delay <= 4'b0;
   end else begin
       if (HREADY) begin
         hmaster_delay <= HMASTER;
       end
   end
end
//---------------------------
always @ (HMASTER or HADDR_0 or HADDR_1)
   case (HMASTER)
   4'h0: HADDR = HADDR_0;
   4'h1: HADDR = HADDR_1;
   default: HADDR = ~32'b0;
   endcase
always @ (HMASTER or HPROT_0 or HPROT_1)
   case (HMASTER)
   4'h0: HPROT = HPROT_0;
   4'h1: HPROT = HPROT_1;
   default: HPROT = 32'b0;
   endcase
always @ (HMASTER or HTRANS_0 or HTRANS_1)
   case (HMASTER)
   4'h0: HTRANS = HTRANS_0;
   4'h1: HTRANS = HTRANS_1;
   default: HTRANS = 32'b0;
   endcase
```

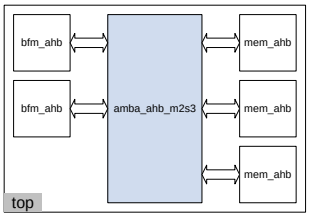

AMBA AHB two-master: master to slave mux

```
always @ (HMASTER or HWRITE_0 or HWRITE_1)
   case (HMASTER)
   4'h0: HWRITE = HWRITE_0;
   4'h1: HWRITE = HWRITE_1;
   default: HWRITE = 32'b0;
   endcase
always @ (HMASTER or HSIZE_0 or HSIZE_1)
   case (HMASTER)
   4'h0: HSIZE = HSIZE_0;
   4'h1: HSIZE = HSIZE_1;
   default: HSIZE = 32'b0;
   endcase
always @ (HMASTER or HBURST_0 or HBURST_1)
   case (HMASTER)
   4'h0: HBURST = HBURST_0;
   4'h1: HBURST = HBURST_1;
   default: HBURST = 32'b0;
   endcase
always @ (hmaster_delay or HWDATA_0 or HWDATA_1)
   case (hmaster_delay)
   4'h0: HWDATA = HWDATA_0;
   4'h1: HWDATA = HWDATA_1;
   default: HWDATA = 32'b0;
   endcase
endmodule
`endif
```

AMBA AHB two-master and three-slave case: test-bench

```
... ...
module top ;
... ...
 generate
  genvar GM;
  for (GM=0; GM<2; GM=GM+1) begin :GM_BLK
    bfm_ahb #(.START_ADDR((16'h1000*GM)<<16),
              .DEPTH_IN_BYTES(32'h100))
    u_bfm_ahb (
        .HRESETn (HRESETn  )
      , .HCLK    (HCLK     )
      , .HBUSREQ (M_HBUSREQ[GM])
      , .HGRANT  (M_HGRANT [GM])
      , .HADDR   (M_HADDR  [GM])
      , .HPROT   (M_HPROT  [GM])
      , .HLOCK   (M_HLOCK  [GM])
      , .HTRANS  (M_HTRANS [GM])
      , .HWRITE  (M_HWRITE [GM])
      , .HSIZE   (M_HSIZE  [GM])
      , .HBURST  (M_HBURST [GM])
      , .HWDATA  (M_HWDATA [GM])
      , .HRDATA  (M_HRDATA     )
      , .HRESP   (M_HRESP      )
      , .HREADY  (M_HREADY     )
      , .IRQ     (1'b0         )
    );
  end
  endgenerate
  ... ...
```

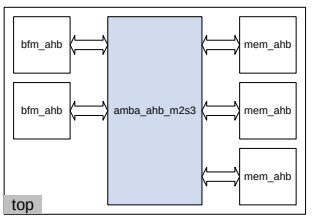
Two BFM with different starting address



AMBA AHB two-master and three-slave case: test-bench

```
amba_ahb_m2s3 #(.P_HSEL0_START(16'h0000),.P_HSEL0_SIZE(16'h0100)
                ,.P_HSEL1_START(16'h1000),.P_HSEL1_SIZE(16'h0100)
                ,.P_HSEL2_START(16'h2000),.P_HSEL2_SIZE(16'h0100))
 u_amba_ahb ( ... ... );

generate
  genvar GS;
  for (GS=0; GS<3; GS=GS+1) begin :GS_BLK
    mem_ahb #(.SIZE_IN_BYTES(SIZE_IN_BYTES),
              .DELAY(DELAY))
    u_mem_ahb (
        .HRESETn   (HRESETn)
      , .HCLK      (HCLK   )
      , .HADDR     (S_HADDR  )
      , .HTRANS    (S_HTRANS )
      , .HWRITE    (S_HWRITE )
      , .HSIZE     (S_HSIZE  )
      , .HBURST    (S_HBURST )
      , .HWDATA    (S_HWDATA )
      , .HREADYin  (S_HREADY )
      , .HSEL      (S_HSEL     [GS])
      , .HRDATA    (S_HRDATA   [GS])
      , .HRESP     (S_HRESP    [GS])
      , .HREADYout (S_HREADYout[GS])
    );
  end
  endgenerate
```

Three memories



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%
```

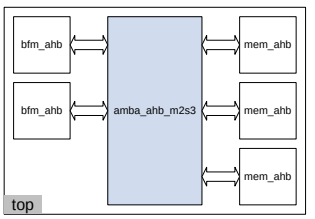

Simulation with ModelSim (2/4)

modelsim.args

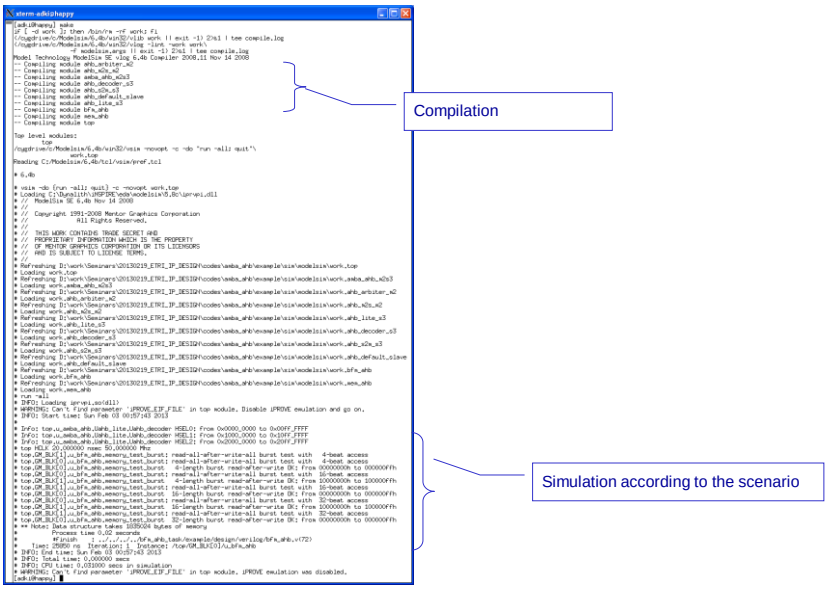
```
//-----------------------------------------------------------------
+incdir+../../design/verilog
        ./sim_define.v
        ../../design/verilog/amba_ahb_m2s3.v
//-----------------------------------------------------------------
+incdir+../../../../ahb_lite/example/design/verilog/
        ../../../../ahb_lite/example/design/verilog/ahb_lite_s3.v
//-----------------------------------------------------------------
+incdir+../../../../bfm_ahb_task/example/design/verilog/
        ../../../../bfm_ahb_task/example/design/verilog/bfm_ahb.v
        ../../../../bfm_ahb_task/example/design/verilog/mem_ahb.v
//-----------------------------------------------------------------
// Below are test-bench
//-----------------------------------------------------------------
+incdir+../../bench/verilog
        ../../bench/verilog/top.v
//-----------------------------------------------------------------
```

sim_define.v

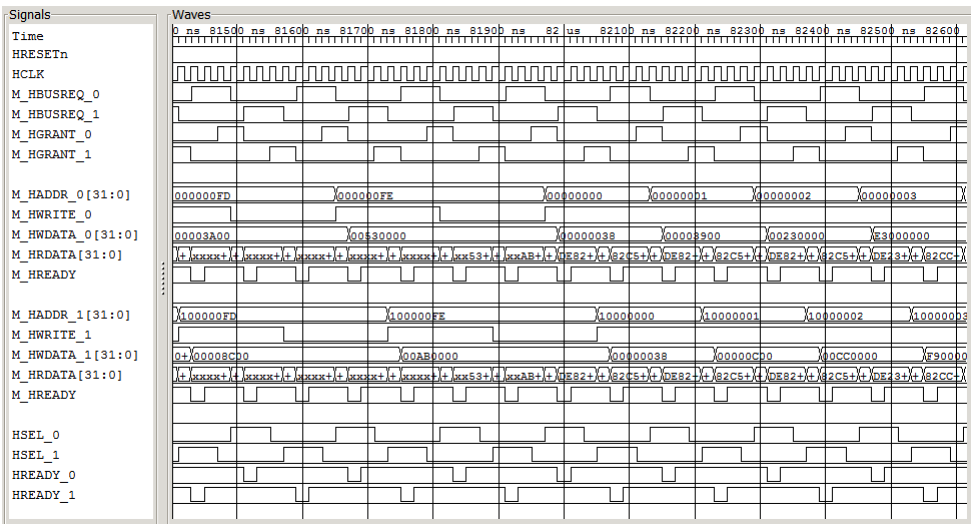
```
//-----------------------------------------------------------------
`define SIM      // define this for simulation case if you are not sure
`define VCD      // define this for VCD waveform dump
`undef  DEBUG
`define RIGOR
`define LOW_POWER
//-----------------------------------------------------------------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



Simulation with ModelSim (3/4)



Simulation with ModelSim (4/4)



Example: AMBA AHB case

- This example shows how to use BFM with tasks
  - Step 1: go to your project directory
    - [user@host] cd $(PROJECT)/codes/amba_ahb
  - Step 2: see the codes
    - [user@host] cd $(PROJECT)/codes/amba_ahb/desing/verilog
  - Step 3: compile and run
    - [user@host] cd $(PROJECT)/codes/amba_ahb/sim/modelsim
    - [user@host] make
  - Step 4: waveform view
    - [user@host] gtkwave wave.vcd &

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


References

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