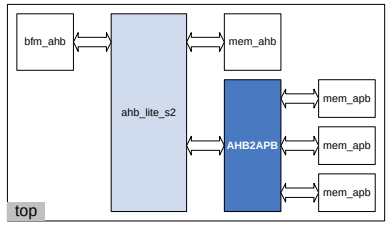
Design and Verification of AHB2APB

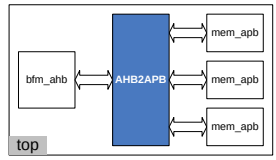
2013 – 2017

Ando Ki
(adki@future-ds.com)

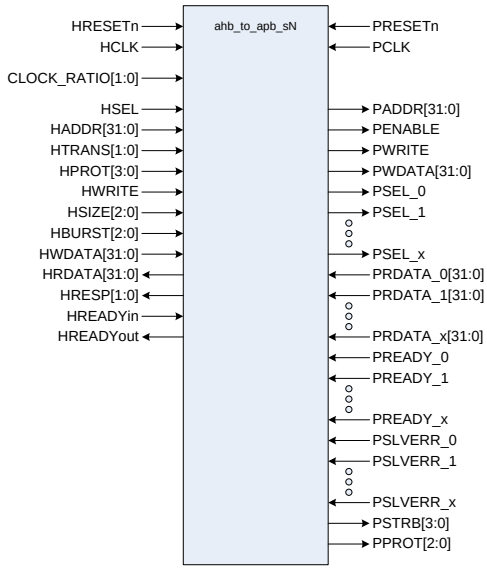
AMBA AHB-to-APB design & verification



- Test-bench includes 'BFM', 'AMBA AHB', and 'MEMORY'.
- BFM generates test-pattern and test-vector.

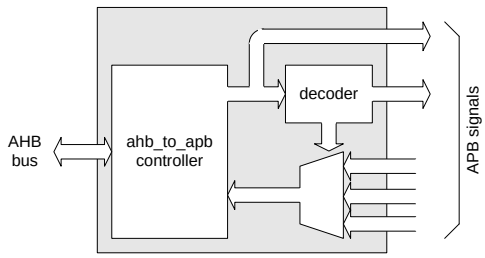


AHB-to-APB bus bridge



- HCLK and PCLK can be asynchronous

AHB-to-APB internal



AHB2APB: module

```
`timescale 1ns/1ns
`include "ahb_to_apb_controller.v"

module ahb_to_apb_s3
    #(parameter P_PSEL0_START = 16'hC000, P_PSEL0_SIZE  = 16'h0010,
                P_PSEL1_START = 16'hC010, P_PSEL1_SIZE  = 16'h0010,
                P_PSEL2_START = 16'hC020, P_PSEL2_SIZE  = 16'h0010)
(
      input  wire        HRESETn
    , input  wire        HCLK
    , input  wire        HSEL
    , input  wire [31:0] HADDR
    , input  wire [ 1:0] HTRANS
    , input  wire [ 3:0] HPROT
    , input  wire        HWRITE
    , input  wire [ 2:0] HSIZE
    , input  wire [ 2:0] HBURST
    , input  wire [31:0] HWDATA
    , output wire [31:0] HRDATA
    , output wire [ 1:0] HRESP
    , input  wire        HREADYin
    , output wire        HREADYout
    , input  wire        PCLK
    , input  wire        PRESETn
    , output wire        PENABLE
    , output wire [31:0] PADDR
    , output wire        PWRITE
    , output wire [31:0] PWDATA
    , output wire        PSEL0
    , input  wire [31:0] PRDATA0
    `ifdef AMBA_APB3
    , input  wire        PREADY0
    , input  wire        PSLVERR0
    `endif
    , output wire        PSEL1
    , input  wire [31:0] PRDATA1
    `ifdef AMBA_APB3
    , input  wire        PREADY1
    , input  wire        PSLVERR1
    `endif
    , output wire        PSEL2
    , input  wire [31:0] PRDATA2
    `ifdef AMBA_APB3
    , input  wire        PREADY2
    , input  wire        PSLVERR2
    `endif
    `ifdef AMBA_APB4
    , output wire [ 2:0] PPROT
    , output wire [ 3:0] PSTRB
    `endif
    , input  wire [ 1:0] CLOCK_RATIO // 0=1:1, 3=async
);
```


AHB2APB: module

```
//------------------------------------------------------
  wire        PSEL   ;
  reg  [31:0] PRDATA ;
  `ifdef AMBA_APB3
  reg         PREADY ;
  reg         PSLVERR;
  `endif
  //------------------------------------------------------
  wire [2:0] _psel = {PSEL2,PSEL1,PSEL0};
  //------------------------------------------------------
   ahb_to_apb_controller Uahb_to_apb_controller (
        .HRESETn   (HRESETn)
      , .HCLK      (HCLK)
      , .HSEL      (HSEL)
      , .HADDR     (HADDR)
      , .HTRANS    (HTRANS)
      , .HPROT     (HPROT)
      , .HWRITE    (HWRITE)
      , .HSIZE     (HSIZE)
      , .HBURST    (HBURST)
      , .HWDATA    (HWDATA)
      , .HRDATA    (HRDATA)
      , .HRESP     (HRESP)
      , .HREADYin  (HREADYin)
      , .HREADYout (HREADYout)
      , .PCLK      (PCLK)
      , .PRESETn   (PRESETn)
      , .PSEL      (PSEL)
      , .PENABLE   (PENABLE)
      , .PADDR     (PADDR)
      , .PWRITE    (PWRITE)
      , .PRDATA    (PRDATA)
      , .PWDATA    (PWDATA)

      `ifdef AMBA_APB3
      , .PREADY    (PREADY )
      , .PSLVERR   (PSLVERR)
      `endif
      `ifdef AMBA_APB4
      , .PPROT     (PPROT)
      , .PSTRB     (PSTRB)
      `endif

      , .CLOCK_RATIO(CLOCK_RATIO)
   );
```

AHB2APB: module

```
//-------------------------------------------------------
 apb_decoder_s3 #(3, P_PSEL0_START, P_PSEL0_SIZE,
                  P_PSEL1_START, P_PSEL1_SIZE,
                  P_PSEL2_START, P_PSEL2_SIZE)
           Uapb_decoder (
                .PSELin(PSEL),
                .PADDR( PADDR), .PSELout1(PSEL0),
                        .PSELout2(PSEL1),
                        .PSELout3(PSEL2)
 );
 //-------------------------------------------------------
 always @ (_psel or PRDATA0 or PRDATA1 or PRDATA2) begin
  case(_psel)
    3'b001: PRDATA = PRDATA0;
    3'b010: PRDATA = PRDATA1;
    3'b100: PRDATA = PRDATA2;
    default: PRDATA = 32'b0;
  endcase
 end
 //-------------------------------------------------------
 `ifdef AMBA_APB3
 always @ (_psel or
         PREADY0 or PREADY1 or PREADY2 ) begin
  case(_psel)
    3'b001: PREADY = PREADY0;
    3'b010: PREADY = PREADY1;
    3'b100: PREADY = PREADY2;
    default:  PREADY = 1'b1  ;
  endcase
 end
```

```
always @ (_psel or PSLVERR0 or PSLVERR1 or PSLVERR2 ) begin
   case(_psel)
     3'b001: PSLVERR = PSLVERR0;
     3'b010: PSLVERR = PSLVERR1;
     3'b100: PSLVERR = PSLVERR2;
     default: PSLVERR = 1'b0    ;
   endcase
 end
 `endif
```


AHB2APB controller module

```
`timescale 1ns/1ns

module ahb_to_apb_controller (
      input  wire        HRESETn
    , input  wire        HCLK
    , input  wire        HSEL
    , input  wire [31:0] HADDR
    , input  wire [ 1:0] HTRANS
    , input  wire [ 3:0] HPROT
    , input  wire        HWRITE
    , input  wire [ 2:0] HSIZE
    , input  wire [ 2:0] HBURST
    , input  wire [31:0] HWDATA
    , output reg  [31:0] HRDATA
    , output reg  [ 1:0] HRESP
    , input  wire        HREADYin
    , output reg         HREADYout
    , input  wire        PCLK
    , input  wire        PRESETn
    , output reg         PSEL
    , output reg         PENABLE
    , output wire [31:0] PADDR
    , output wire        PWRITE
    , input  wire [31:0] PRDATA
    , output wire [31:0] PWDATA
    `ifdef AMBA_APB3
    , input  wire        PREADY
    , input  wire        PSLVERR
    `endif
```

```
    `ifdef AMBA_APB4
    , output wire [ 2:0] PPROT
    , output wire [ 3:0] PSTRB
    `endif
    , input  wire [ 1:0] CLOCK_RATIO // 0=1:1, 3=async
);
```

AHB2APB controller module

```
//------------------------------------------------------
 `ifndef AMBA_APB3
 wire    PREADY  = 1'b1;
 wire    PSLVERR = 1'b0;
 `endif
 `ifndef AMBA_APB4
 wire [ 2:0] PPROT;
 wire [ 3:0] PSTRB;
 `endif
 //------------------------------------------------------
 reg  [31:0] tADDR ;
 reg         tWRITE;
 reg  [31:0] tWDATA;
 reg  [31:0] tRDATA;
 reg         tREQ  ;
 reg         tACK  ;
 reg         tERROR;
 reg  [ 2:0] tPROT ;
 reg  [ 3:0] tSTRB ;
 //------------------------------------------------------
 assign PADDR   = tADDR ;
 assign PWRITE  = tWRITE;
 assign PWDATA  = tWDATA;
 assign PPROT   = tPROT ;
 assign PSTRB   = tSTRB ;
```

```
//------------------------------------------------------
   reg        tACKsync, tACKsync0, tACKsync1;
   always @ (posedge HCLK or negedge HRESETn) begin
      if (HRESETn==0) begin
         tACKsync0 <= 1'b0;
         tACKsync1 <= 1'b0;
      end else begin
         tACKsync0 <= tACK;
         tACKsync1 <= tACKsync0;
      end
   end
   always @ ( * ) begin
      case (CLOCK_RATIO)
      2'b00: tACKsync = tACK;
      2'b01: tACKsync = tACKsync1;
      2'b10: tACKsync = tACKsync1;
      2'b11: tACKsync = tACKsync1;
      endcase
   end
```


AHB2APB controller module

```
//------------------------------------------------------
 reg [2:0] state;
 localparam STH_IDLE   =  'h0,
            STH_WRITE0 =  'h1,
            STH_WRITE1 =  'h2,
            STH_READ0  =  'h3,
            STH_WAIT   =  'h4;
 //------------------------------------------------------
 // AHB bus wrapper
 always @ (posedge HCLK or negedge HRESETn) begin
    if (HRESETn==0) begin
       HRDATA    <= 32'b0;
       HRESP     <=  2'b00; //`HRESP_OKAY;
       HREADYout <=  1'b1;
       tADDR     <= 32'h0;
       tWDATA    <= 32'h0;
       tWRITE    <=  1'b0;
       tPROT     <=  3'h0;
       tSTRB     <=  4'hF;
       tREQ      <=  1'b0;
       state     <= STH_IDLE;
    end else begin // if (HRESETn==0) begin
       case (state)
       STH_IDLE: begin
          if (HSEL && HREADYin) begin
             case (HTRANS)
             //`HTRANS_IDLE, `HTRANS_BUSY: begin
             2'b00, 2'b01: begin
                   HREADYout <= 1'b1;
                   HRESP     <= 2'b00; //`HRESP_OKAY;
                   state     <= STH_IDLE;
             end // HTRANS_IDLE or HTRANS_BUSY
```

AHB2APB controller module

```
            2'b10, 2'b11: begin
                    HREADYout <= 1'b0;
                    HRESP     <= 2'b00; //`HRESP_OKAY;
                    tADDR     <= HADDR[31:0];
                    tWRITE    <= HWRITE;
                    tPROT     <= {~HPROT[0],1'b1,HPROT[1]};
                    tSTRB     <= get_strb(HADDR[1:0],HSIZE);
                    if (HWRITE) begin
                       state <= STH_WRITE0;
                    end else begin
                       tREQ  <= 1'b1;
                       state <= STH_READ0;
                    end
                end // HTRANS_NONSEQ or HTRANS_SEQ
                endcase // HTRANS
            end else begin// if (HSEL && HREADYin)
                HREADYout <= 1'b1;
                HRESP     <= 2'b00; //`HRESP_OKAY;
            end
            end // STH_IDLE
        STH_WRITE0: begin
                tWDATA    <= HWDATA;
                tREQ      <= 1'b1;
                state     <= STH_WRITE1;
            end // STH_WRITE0
```

```
        STH_WRITE1: begin
                if (tACKsync ) begin
                   tREQ      <= 1'b0;
                   HRESP     <= {1'b0,tERROR};
                   tADDR     <= 32'b0;
                   tWDATA    <= 32'b0;
                   tWRITE    <= 1'b0;
                   if (CLOCK_RATIO==2'b00) begin
                      HREADYout <= 1'b1;
                      state     <= STH_IDLE;
                   end else begin
                      state     <= STH_WAIT;
                   end
                end
            end // STH_WRITE1
        STH_READ0: begin
                if (tACKsync ) begin
                   tREQ      <= 1'b0;
                   HRDATA    <= tRDATA;
                   HRESP     <= {1'b0,tERROR};
                   if (CLOCK_RATIO==2'b00) begin
                      HREADYout <= 1'b1;
                      state     <= STH_IDLE;
                   end else begin
                      state     <= STH_WAIT;
                   end
                end
            end // STH_READ0
```


AHB2APB controller module

```
        STH_WAIT: begin
                if (tACKsync==1'b0) begin
                   HREADYout <= 1'b1;
                   state     <= STH_IDLE;
                end
                end // STH_WAIT
            endcase // state
        end // if (HRESETn==0)
    end // always
    //-------------------------------------------------------
    reg         tREQsync, tREQsync0, tREQsync1;
    always @ (posedge PCLK or negedge PRESETn) begin
        if (PRESETn==0) begin
            tREQsync0 <= 1'b0;
            tREQsync1 <= 1'b0;
        end else begin
            tREQsync0 <= tREQ;
            tREQsync1 <= tREQsync0;
        end
    end
    always @ ( * ) begin
        case (CLOCK_RATIO)
        2'b00: tREQsync = tREQ;
        2'b01: tREQsync = tREQsync1;
        2'b10: tREQsync = tREQsync1;
        2'b11: tREQsync = tREQsync1;
        endcase
    end
```

```
    //-------------------------------------------------------
      reg [1:0] pstate;
      localparam STP_IDLE  = 2'h0,
                 STP_SETUP = 2'h1,
                 STP_GO    = 2'h2,
                 STP_WAIT  = 2'h3;
      //-------------------------------------------------------
      always @ (posedge PCLK or negedge PRESETn) begin
          if (PRESETn==0) begin
              PSEL    <=  1'b0;
              PENABLE <=  1'b0;
              tACK    <=  1'b0;
              tRDATA  <= 32'b0;
              tERROR  <=  1'b0;
              pstate  <= STP_IDLE;
          end else begin
              case (pstate)
              STP_IDLE: begin
                  if (tREQsync ) begin
                      PSEL    <= 1'b1;
                      pstate  <= STP_SETUP;
                  end
                  end // STP_IDLE
              STP_SETUP: begin
                  PENABLE <= 1'b1;
                  pstate  <= STP_GO;
                  end // STP_SETUP
```

AHB2APB controller module

```
        STP_GO: begin
            if (PREADY) begin
               PENABLE <= 1'b0;
               PSEL    <= 1'b0;
               tACK    <= 1'b1;
               tRDATA  <= PRDATA;
               tERROR  <= PSLVERR;
               pstate  <= STP_WAIT;
            end
            end // STP_GO
          STP_WAIT: begin
            if (CLOCK_RATIO==2'b0) begin
               tACK    <= 1'b0;
               pstate <= STP_IDLE;
            end else begin
               if (tREQsync==1'b0) begin
                  tACK   <= 1'b0;
                  pstate <= STP_IDLE;
               end
            end
            end // STP_WAIT
         endcase
      end // if (PRESETn==0)
   end // always @ (posedge PCLK or negedge PRESETn) begin
```

```
  //-------------------------------------------------------
   function [3:0] get_strb;
      input [1:0] add;   // address offset
      input [2:0] size;  // transfer size
      reg   [3:0] be;
      begin
         case ({size,add})
            `ifdef ENDIAN_BIG
            .....
            `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: begin
                  be = 4'b0;
                   end
         endcase
         get_strb = be;
      end
   endfunction
   //-------------------------------------------------------
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
+incdir+../../../../bfm_ahb_task/example/design/verilog
+incdir+../../../../bfm_apb_task/example/design/verilog
        ./sim_define.v
        ../../design/verilog/bfm_ahb.v
        ../../design/verilog/ahb_to_apb_s3.v
        ../../../../bfm_apb_task/example/design/verilog/mem_apb.v
//----------------------------------------------------------------
// Below are test-bench
//----------------------------------------------------------------
+incdir+../../bench/verilog
        ../../bench/verilog/top.v
//----------------------------------------------------------------
```

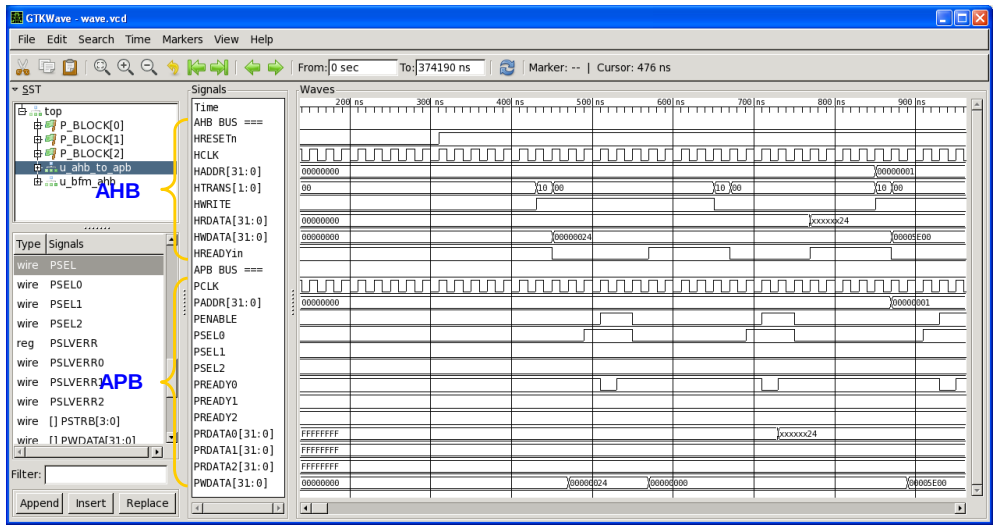
sim_define.v

```
`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 HCLK_FREQ   50000000
`define PCLK_FREQ   50000000
`define MEM_DELAY  1
//----------------------------------------------------------------
`define AMBA_APB3
`define AMBA_APB4
`define AMBA3
`define AMBA4
//----------------------------------------------------------------
`define SINGLE_TEST
`define BURST_TEST

//----------------------------------------------------------------
`endif
```


Simulation with ModelSim (3/4)

Simulation with ModelSim (4/4)



Example: AHB to APB case

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

```
[user@host] cd $(PROJECT)/codes/ahb_to_apb/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.