

# AMBA APB BFM

2013 – 2014 – 2015 – 2016

Ando Ki  
(adki@future-ds.com)

## Verilog task and function

### Task

- ◆ Declared within a module
- ◆ Referenced only by a behavioral within the module
  - Can be referenced only from within a cyclic (always) or single-pass behavior (initial).
- ◆ Parameters passed to task as inputs and inouts and from task a outputs or inputs
- ◆ Local variables can be declared
- ◆ Recursion not supported although nesting permitted
- ◆ A task can contain time-controlling statements.
- ◆ A task can enable other tasks and functions.
- ◆ A task can have zero or more arguments of any type.
- ◆ A task shall not return a value, since it creates a hierarchical organization of the procedural statements within a Verilog behavior.

### Function

- ◆ Implement combinational behavior
- ◆ No timing control
  - '#' or '@' or 'wait' is not permitted
- ◆ May call other functions with no recursion
- ◆ Reference in an expression, e.g., RHS
- ◆ No 'output' or 'inout' allowed
- ◆ No non-blocking assignment
- ◆ The purpose of a *function* is to respond to an input value by returning a single value.
  - A function can be used as an operand in an expression.
  - The value of that operand is the value returned by the function.
- ◆ A function shall execute in one simulation time unit.
- ◆ A function cannot enable a task.
- ◆ A function shall have at least one **input** type argument and shall not have an **output** or **inout** type argument.
- ◆ A function shall return a single value, which can be scalar (single-bit) or vector (multi-bit).

## Verilog: system & user-defined tasks/function

❏ System tasks and system functions are commands executed by a Verilog simulator, not hardware modeling constructs.

- ◆ While VPI routines are functions to be used in C program.

❏ System tasks and system functions begin with \$.

❏ System task -- a sub-routine procedure.

❏ System function -- a function returning a return value.

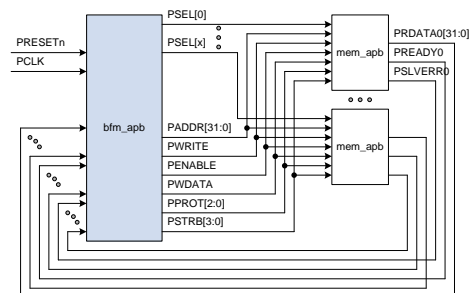
❏ Built-in system tasks and system functions

- ◆ These are part of Verilog language and built into Verilog simulator.
- ◆ Standard system tasks
  - ❏ \$display, \$stop, \$finish
- ◆ Standard system functions
  - ❏ \$time, \$random

❏ User-defined tasks and functions

- ◆ Its name must start with \$.

## APB BFM example: task-based case



## APB BFM: module

```
// bfm_apb_s3.v
module bfm_apb_s3 #(parameter P_DWIDTH=32
    , P_STRB =(P_DWIDTH/8)
    , P_NUM=3
    , P_ADDR_START0 = 16'h0000, P_ADDR_SIZE0 = 16'h0010
    , P_ADDR_START1 = 16'h1000, P_ADDR_SIZE1 = 16'h0010
    , P_ADDR_START2 = 16'h2000, P_ADDR_SIZE2 = 16'h0010)
(
    input wire          PRESETn
    , input wire        PCLK
    , output reg [P_NUM-1:0] PSEL
    , output reg [31:0]   PADDR
    , output reg         PENABLE
    , output reg         PWRITE
    , output reg [P_DWIDTH-1:0] PWDATA
    , input wire [P_DWIDTH-1:0] PRDATA0
    , input wire [P_DWIDTH-1:0] PRDATA1
    , input wire [P_DWIDTH-1:0] PRDATA2
    `ifdef AMBA3
    , input wire [P_NUM-1:0] PREADY
    , input wire [P_NUM-1:0] PSLVERR
    `endif
    `ifdef AMBA4
    , output reg [2:0]   PPROT
    , output reg [P_STRB-1:0] PSTRB
    `endif
);
    ....
endmodule
```

Up to 3 APB slaves

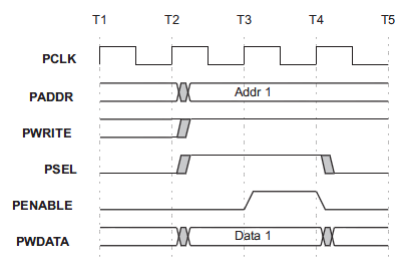
Address decoding parameters

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 5 )

## APB BFM: write task

```
// bfm_apb_tasks.v
//-----
task apb_write;
input [31:0] addr;
input [31:0] data;
input [2:0] size;
begin
    @ (posedge PCLK);
    PADDR <= #1 addr;
    PWRITE <= #1 1'b1;
    PSEL <= #1 decoder(addr);
    PWDATA <= #1 data;
    PSTRB <= #1 get_pstrob(addr,size);
    @ (posedge PCLK);
    PENABLE <= #1 1'b1;
    @ (posedge PCLK);
    while (get_pready(addr)==1'b0) @ (posedge PCLK);
    `ifdef LOW_POWER
    PADDR <= #1 32'h0;
    PWRITE <= #1 1'b0;
    PWDATA <= #1 ~32'h0;
    `endif
    PSEL <= #1 {P_NUM{1'b0}};
    PENABLE <= #1 1'b0;
    if (get_pslverr(addr)==1'b1) $display($time, "%m PSLVERR");
end
endtask
```

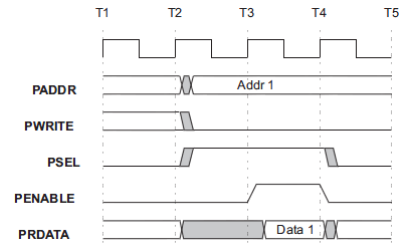


Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 6 )

## APB BFM: read task

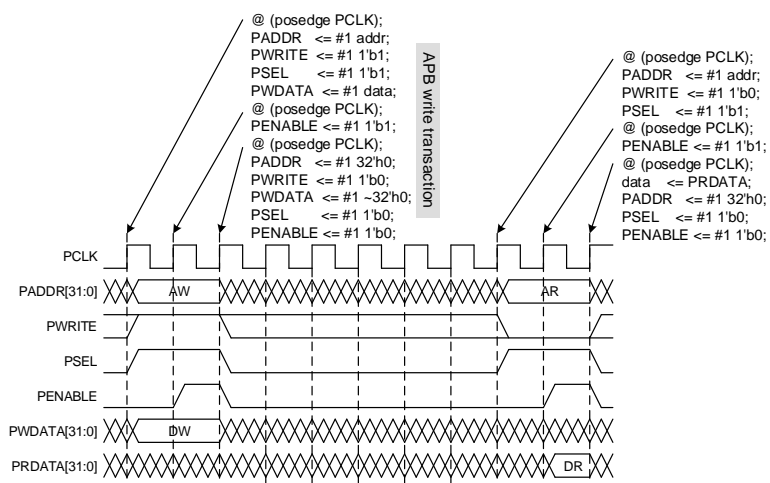
```
// bfm_apb_tasks.v
//-----
task apb_read;
input [31:0] addr;
output [31:0] data;
input [2:0] size;
begin
    @ (posedge PCLK);
    PADDR <= #1 addr;
    PWRITE <= #1 1'b0;
    PSEL <= #1 decoder(addr);
    PSTRB <= #1 {P_NUM{1'b1}};
    @ (posedge PCLK);
    PENABLE <= #1 1'b1;
    @ (posedge PCLK);
    while (get_pready(addr)==1'b0) @ (posedge PCLK);
    `ifdef LOW_POWER
    PADDR <= #1 32'h0;
    `endif
    PSEL <= #1 {P_NUM{1'b0}};
    PENABLE <= #1 1'b0;
    if (get_pslverr(addr)==1'b1) $display($time, "%m PSLVERR");
    data = get_prdata(addr); // it should be blocking
end
endtask
```



Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM (7)

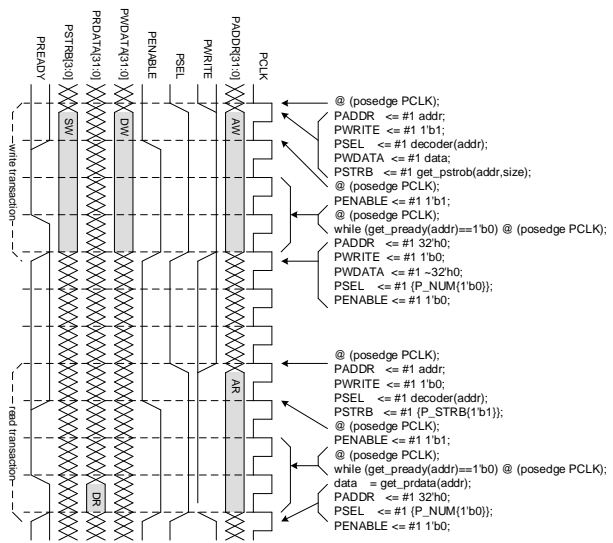
## APB BFM: read and tasks



Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM (8)

# APB BFM: read and tasks



# APB BFM: testing scenario

```
// bfm_apb_s3.v
...
//-----
initial begin
  ...
  memory_test(P_ADDR_START0, P_ADDR_START0+32'h10-0, 4);
  memory_test(P_ADDR_START1, P_ADDR_START1+32'h10-1, 4);
  memory_test(P_ADDR_START2, P_ADDR_START2+32'h10-2, 4);
  ...
  $finish(2);
end
```

testing scenario

Finish simulation

## APB BFM: testing scenario

```
// bfm_apb_s3.v
... ..
task memory_test;
input [31:0] start ; // starting address
input [31:0] finish; // ending address, inclusive
input [ 2:0] size ; // num of byte to access
reg  [P_DWIDTH-1:0] dataW, dataR;
integer a, b, err;
begin
    err = 0;
    //-----
    // read-after-write test
    for (a=start; a<=(finish-size); a=a+size) begin
        dataW = $random;
        apb_write(a, dataW, size);
        apb_read (a, dataR, size);
        if (dataR!=dataW) begin
            err = err + 1;
            $display($time, "%m RAW error at A:0x%08x D:0x%x, but 0x%x expected",
                a, dataR, dataW);
        end
    end
    if (err==0) $display($time, "%m RAW 0x%x-%x test OK", start, finish);
    ... ..
end
```

Read after write cases

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 11 )

## APB BFM: testing scenario

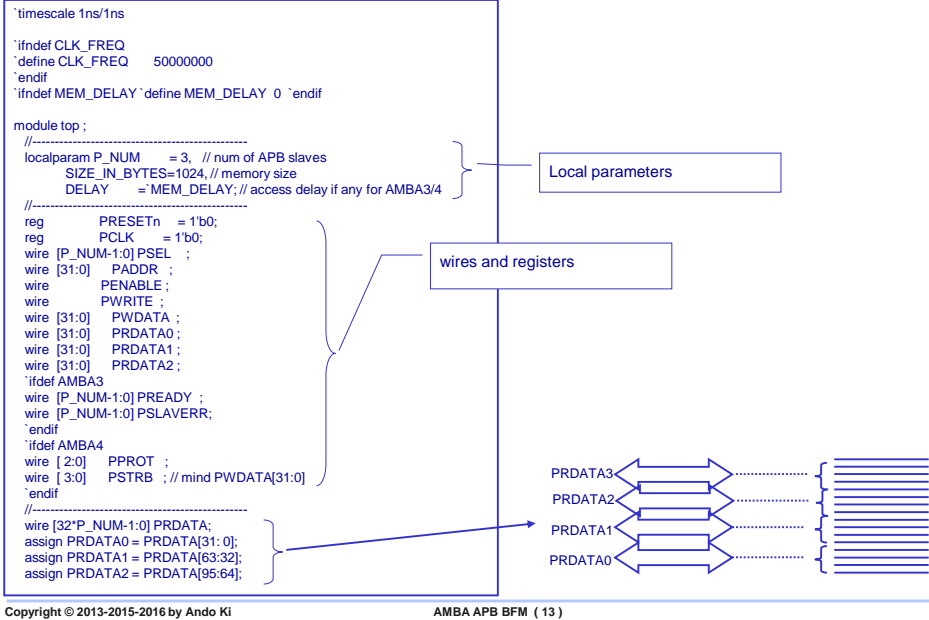
```
// bfm_apb_s3.v
... ..
task memory_test;
... ..
    err = 0;
    //-----
    // read_all-after-write_all test
    for (a=start; a<=(finish-size); a=a+size) begin
        b = a - start;
        reposit[b] = $random;
        apb_write(a, reposit[b], size);
    end
    for (a=start; a<=(finish-size); a=a+size) begin
        b = a - start;
        apb_read(a, dataR, size);
        if (dataR!=reposit[b]) begin
            err = err + 1;
            $display($time, "%m RAAWA error at A:0x%08x D:0x%x, but 0x%x expected",
                a, dataR, reposit[b]);
        end
    end
    if (err==0) $display($time, "%m RAAWA 0x%x-%x test OK", start, finish);
end
endtask
```

Read-all after write-all cases

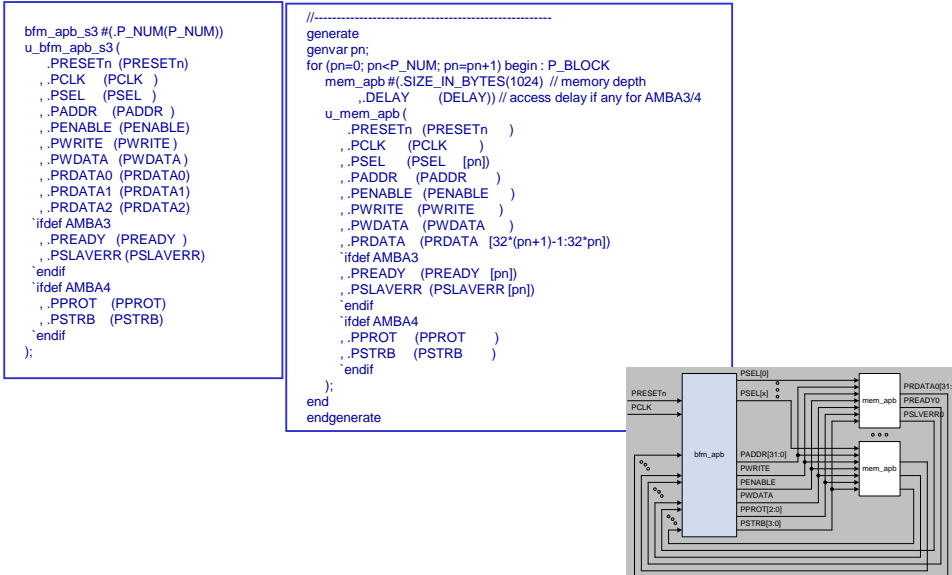
Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 12 )

## APB BFM: testbench (1/3)



## APB BFM: testbench (2/3)



## APB BFM: testbench (3/3)

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

How to describe a clock

How to check clock frequency  
at the beginning of simulation

How to prepare VCD file to see waveform

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 15 )

## APB Memory (1/3)

```
`timescale 1ns/1ns

`ifndef AMBA4
`ifndef AMBA3
ERROR AMBA3 should be defined when AMBA4 is defined
`endif
`endif

module mem_apb
    #(parameter SIZE_IN_BYTES=1024 // memory depth
      , DELAY=0 // access delay if any for AMBA3/4
    )
    (
        input wire PRESETn
        , input wire PCLK
        , input wire PSEL
        , input wire [31:0] PADDR
        , input wire PENABLE
        , input wire PWRITE
        , input wire [31:0] PWDATA
        , output reg [31:0] PRDATA
        `ifdef AMBA3
            , output reg PREADY
            , output reg PSLAVERR
        `endif
        `ifdef AMBA4
            , input wire [2:0] PPROT
            , input wire [3:0] PSTRB
        `endif
    );

    //-----
    `ifndef AMBA3
        reg PREADY = 1'b1;
        reg PSLAVERR = 1'b0;
    `endif
    `ifdef AMBA4
        wire [2:0] PPROT = 3'h0;
        wire [3:0] PSTRB = 4'hF;
    `endif
    //-----
    reg [7:0] mem0[0:SIZE_IN_BYTES/4];
    reg [7:0] mem1[0:SIZE_IN_BYTES/4];
    reg [7:0] mem2[0:SIZE_IN_BYTES/4];
    reg [7:0] mem3[0:SIZE_IN_BYTES/4];
    //-----
    localparam A_WIDTH = clogb2(SIZE_IN_BYTES);
    wire [A_WIDTH-1:2] addr = PADDR[A_WIDTH-1:2];
    //-----
    reg [31:0] dcnt = 'h1; // delay counter
```

For coding convenience;  
AMBA3/4 signals are valid even  
AMBA3/4 macro is not defined.

Byte-wise memory blocks

How to calculate log2(num)

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 16 )



## APB Memory (2/3)

```

localparam ST_IDLE = 'h0,
          ST_DELAY = 'h1,
          ST_DONE = 'h2;

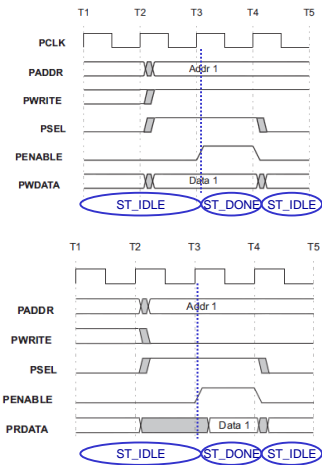
reg [1:0] state = ST_IDLE;
always @ (posedge PCLK or negedge PRESETn) begin
  if (PRESETn==1'b0) begin
    PRDATA <= ~32'h0;
    PREADY <= 1'b1;
    PSLAERR <= 1'b0;
    dcnt <= 'h1;
    state <= ST_IDLE;
  end else begin
    case (state)
      ST_IDLE: begin
        if (PSEL==1'b1) begin
          if (DELAY==0) begin
            PREADY <= 1'b1;
            state <= ST_DONE;
            if (PWRITE==1'b1) begin
              if (PSTRB[0]==1'b1) mem0[addr] <= PWDATA[7:0];
              if (PSTRB[1]==1'b1) mem1[addr] <= PWDATA[15:8];
              if (PSTRB[2]==1'b1) mem2[addr] <= PWDATA[23:16];
              if (PSTRB[3]==1'b1) mem3[addr] <= PWDATA[31:24];
            end else begin
              PRDATA[7:0] <= mem0[addr];
              PRDATA[15:8] <= mem1[addr];
              PRDATA[23:16] <= mem2[addr];
              PRDATA[31:24] <= mem3[addr];
            end
          end else begin
            PREADY <= 1'b0;
            dcnt <= 'h1;
            state <= ST_DELAY;
          end
        end // ST_IDLE
      end
    endcase
  end
end

```

State of finite state machine (FSM)

Write case;  
note PSTRB[...]

Read case



Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 17 )

## APB Memory (3/3)

```

ST_DELAY: begin
  dcnt <= dcnt + 1;
  if (dcnt>=DELAY) begin
    PREADY <= 1'b1;
    state <= ST_DONE;
    if (PWRITE==1'b1) begin
      if (PSTRB[0]==1'b1) mem0[addr] <= PWDATA[7:0];
      if (PSTRB[1]==1'b1) mem1[addr] <= PWDATA[15:8];
      if (PSTRB[2]==1'b1) mem2[addr] <= PWDATA[23:16];
      if (PSTRB[3]==1'b1) mem3[addr] <= PWDATA[31:24];
    end else begin
      PRDATA[7:0] <= mem0[addr];
      PRDATA[15:8] <= mem1[addr];
      PRDATA[23:16] <= mem2[addr];
      PRDATA[31:24] <= mem3[addr];
    end
  end
end // ST_DELAY
ST_DONE: begin
  PREADY <= 1'b1;
  state <= ST_IDLE;
end // ST_DONE
endcase
end // if
end // always
//-----
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
//-----
endmodule

```

Write case;  
note PSTRB[...]

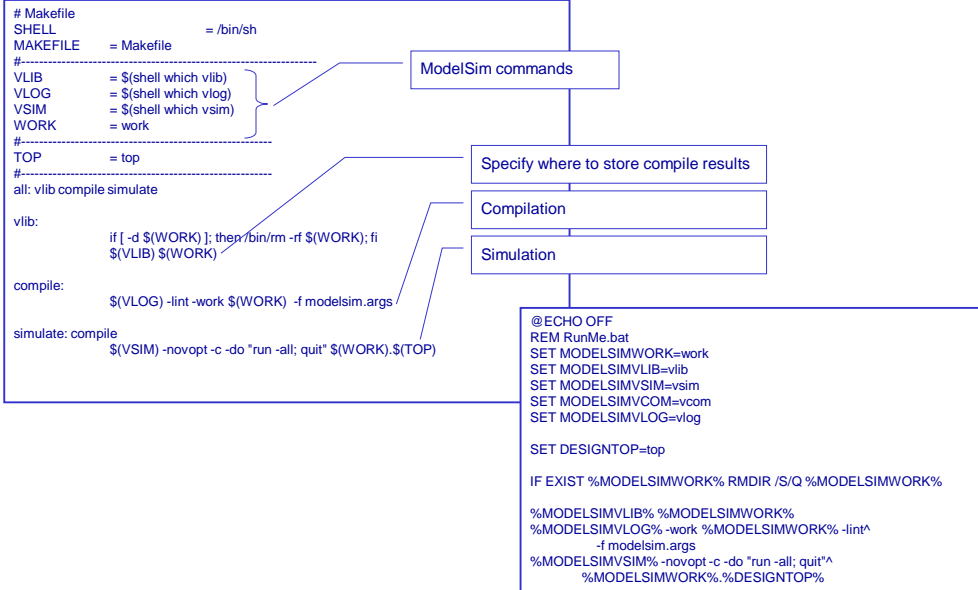
Read case

log2(num)

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 18 )

## Simulation with ModelSim (1/4)



Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 19 )

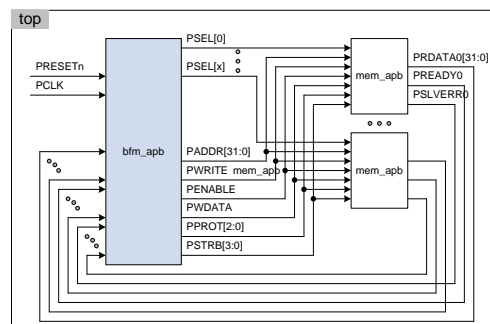
## Simulation with ModelSim (2/4)

```
//-----
+incdir+./../design/verilog
./sim_define.v
./../design/verilog/bfm_apb_s3.v
./../design/verilog/mem_apb.v
//-----
// Below are test-bench
//-----
+incdir+./../bench/verilog
./../bench/verilog/top.v
//-----

`ifdef _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 CLK_FREQ 50000000
`define MEM_DELAY 0
//-----
`define AMBA3
`define AMBA4
//-----
`ifdef AMBA4
`define AMBA3
`endif
//-----
`endif
```

modelsim.args

sim\_define.v



Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 20 )

## Simulation with ModelSim (3/4)

```
xterm addshappy
[ack@happy] $ make
if [ -d work ] ; then /bin/rm -rf work; fi
/cygdrive/c/Modelsim/6.4b/win32/vlib work || exit -1 2>&1 | tee compile.log
/cygdrive/c/Modelsim/6.4b/win32/vlog -linc -work work\
+ modelsim.angs || exit -1 2>&1 | tee compile.log
Model Technology ModelSim SE vlog 6.4b Compiler 2008.11 Nov 14 2008
-- Compiling module bfm_apb_s3
-- Compiling module mem_apb
-- Compiling module top

Top level modules:
top
/cygdrive/c/Modelsim/6.4b/win32/vsim -novopt -c -do 'run -all; quit'\
work.top
Reading C:/Modelsim/6.4b/tcl/vsim/pref.tcl

# 6.4b
# vsim -do [run -all; quit] -c -novopt work.top
# Loading C:\Modelsim\6.4b\win32\vsim\vsimui.dll
# // ModelSim SE 6.4b Nov 14 2008
# //
# // Copyright 1991-2008 Mentor Graphics Corporation
# // All Rights Reserved.
# //
# // THIS WORK CONTAINS TRADE SECRET AND
# // PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# // OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# // AND IS SUBJECT TO LICENSE TERMS.
# //
# // Refreshing D:\work\Seminars\20130219_ETRI_IP_DESIGN\codes\bfm_apb_task\example\sim\modelsim\work\top
# Loading work.top
# Refreshing D:\work\Seminars\20130219_ETRI_IP_DESIGN\codes\bfm_apb_task\example\sim\modelsim\work\bfm_apb_s3
# Loading work.bfm_apb_s3
# Refreshing D:\work\Seminars\20130219_ETRI_IP_DESIGN\codes\bfm_apb_task\example\sim\modelsim\work\mem_apb
# Loading work.mem_apb
# run -all
# INFO: Loading iprvpi.so(dll)
# WARNING: Can't find parameter 'iPROVE_EIF_FILE' in top module. Disable iPROVE emulation and go on.
# INFO: Start time: Sat Feb 02 14:06:15 2013

# top PCLK 20.000000 nsec 50.000000 Mhz
# 790 top.u_bfm_apb_s3.memory_test RAM 0x00000000-00000010 test OK
# 1270 top.u_bfm_apb_s3.memory_test RAM 0x00000000-00000010 test OK
# 1630 top.u_bfm_apb_s3.memory_test RAM 0x00001000-0000100F test OK
# 1990 top.u_bfm_apb_s3.memory_test RAM 0x00001000-0000100F test OK
# 2350 top.u_bfm_apb_s3.memory_test RAM 0x00002000-0000200E test OK
# 2710 top.u_bfm_apb_s3.memory_test RAM 0x00002000-0000200E test OK
# ** Note: Data structure takes 1835024 bytes of memory
# Process time 0.00 seconds
# #initsh : ./../design/verilog/bfm_apb_s3.v(72)
# Time: 2810 ns - Iteration: 1 Instance: /top.u_bfm_apb_s3
# INFO: End time: Sat Feb 02 14:06:15 2013
# INFO: Total time: 0.000000 secs
# INFO: CPU time: 0.016000 secs in simulation
# WARNING: Can't find parameter 'iPROVE_EIF_FILE' in top module. iPROVE emulation was disabled.
[ack@happy]
```

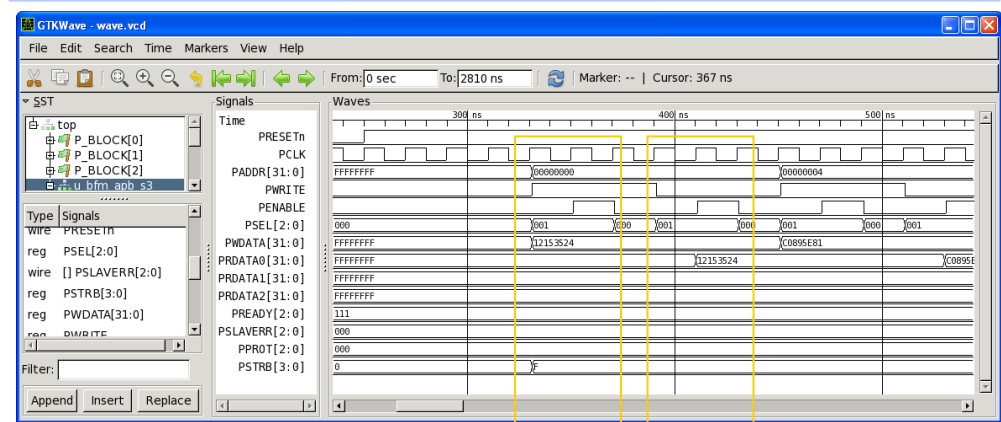
Compilation

Simulation according to the scenario

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 21 )

## Simulation with ModelSim (4/4)



Write case

Read case

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 22 )

## Example: APB BFM task-based case

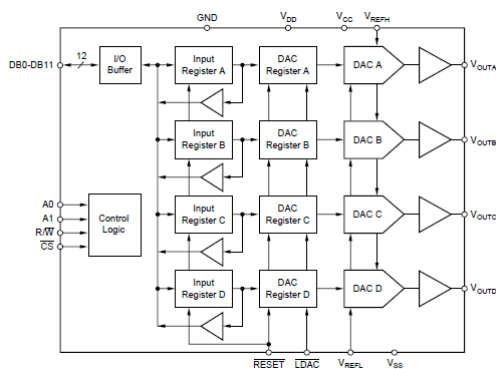
❏ This example shows how to use BFM with tasks

- ◆ Step 1: go to your project directory
  - ❏ [user@host] cd \$(PROJECT)/codes/bfm\_apb\_task/example
- ◆ Step 2: see the codes
  - ❏ [user@host] cd \$(PROJECT)/codes/bfm\_apb\_task/example/desing/verilog
- ◆ Step 3: compile and run
  - ❏ [user@host] cd \$(PROJECT)/codes/bfm\_apb\_task/example/sim/modelsim
  - ❏ [user@host] make
- ◆ Step 4: waveform view
  - ❏ [user@host] gtkwave wave.vcd &

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

## How to deal with long latency device with AMBA 2.0 APB

Burr-Brown (TI) DAC7724 12-bit quad voltage output Digital-to-Analog converter

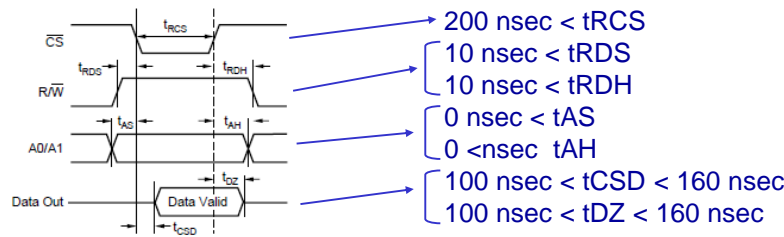


# BB DAC7724

A1	A0	R/W	CS	RESET	LDAC	SELECTED INPUT REGISTER	STATE OF SELECTED INPUT REGISTER	STATE OF ALL DAC REGISTERS
L <sup>(1)</sup>	L	L	L	H <sup>(2)</sup>	L	A	Transparent	Transparent
L	H	L	L	H	L	B	Transparent	Transparent
H	L	L	L	H	L	C	Transparent	Transparent
H	H	L	L	H	L	D	Transparent	Transparent
L	L	L	L	H	H	A	Transparent	Latched
L	H	L	L	H	H	B	Transparent	Latched
H	L	L	L	H	H	C	Transparent	Latched
H	H	L	L	H	H	D	Transparent	Latched
L	L	H	L	H	H	A	Readback	Latched
L	H	H	L	H	H	B	Readback	Latched
H	L	H	L	H	H	C	Readback	Latched
H	H	H	L	H	H	D	Readback	Latched
X <sup>(3)</sup>	X	X	H	H	L	NONE	(All Latched)	Transparent
X	X	X	H	H	H	NONE	(All Latched)	Latched
X	X	X	X	L	X	ALL	Reset <sup>(4)</sup>	Reset <sup>(4)</sup>

NOTES: (1) L = Logic LOW. (2) H= Logic HIGH. (3) X = Don't Care. (4) DAC7724 resets to 800<sub>H</sub>, DAC7725 resets to 000<sub>H</sub>. When RESET rises, all registers that are in their latched state retain the reset value.

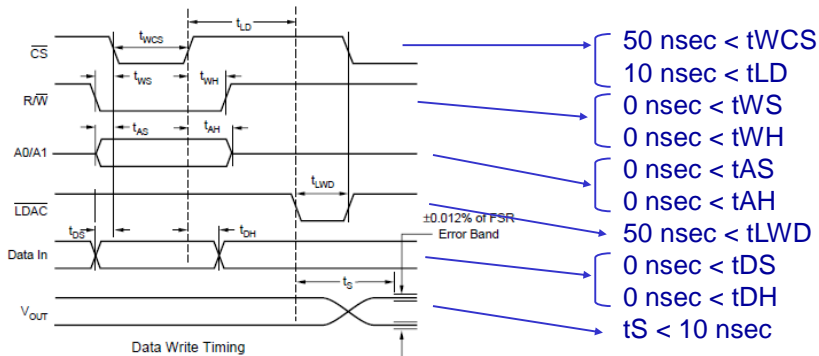
# BB DAC7724



Data Read Timing

SYMBOL	DESCRIPTION	MIN	TYP	MAX	UNITS
$t_{RCS}$	CS LOW for Read	200			ns
$t_{RDS}$	R/W HIGH to CS LOW	10			ns
$t_{RDH}$	R/W HIGH after CS HIGH	10			ns
$t_{DZ}$	CS HIGH to Data Bus in High Impedance		100		ns
$t_{CSD}$	CS LOW to Data Bus Valid		100	160	ns
$t_{WCS}$	CS LOW for Write	50			ns
$t_{WS}$	R/W LOW to CS LOW	0			ns
$t_{WH}$	R/W LOW after CS HIGH	0			ns
$t_{AS}$	Address Valid to CS LOW	0			ns
$t_{AH}$	Address Valid after CS HIGH	0			ns
$t_{LD}$	LDAC Delay from CS HIGH	10			ns
$t_{DS}$	Data Valid to CS LOW	0			ns
$t_{DH}$	Data Valid after CS HIGH	0			ns
$t_{LWD}$	LDAC LOW	50			ns
$t_{RESET}$	RESET LOW Time	50			ns
$t_S$	Settling Time			10	µs

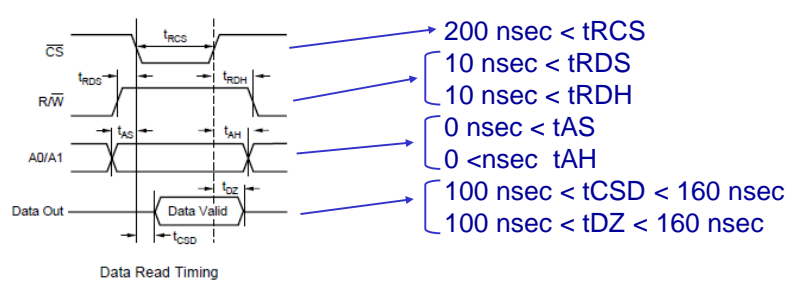
BB DAC7724



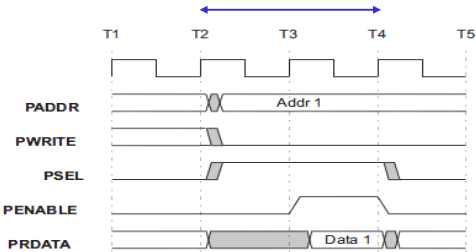
SYMBOL	DESCRIPTION	MIN	TYP	MAX	UNITS
t <sub>WCS</sub>	CS LOW for Read	200			ns
t <sub>LD</sub>	R/W HIGH to CS LOW	10			ns
t <sub>WS</sub>	R/W HIGH after CS HIGH	10			ns
t <sub>WH</sub>	CS HIGH to Data Bus in High Impedance		100		ns
t <sub>AS</sub>	CS LOW to Data Bus Valid		100	160	ns
t <sub>AH</sub>	CS LOW for Write	50			ns
t <sub>LWD</sub>	R/W LOW to CS LOW	0			ns
t <sub>DS</sub>	R/W LOW after CS HIGH	0			ns
t <sub>DH</sub>	Address Valid to CS LOW	0			ns
t <sub>LD</sub>	Address Valid after CS HIGH	0			ns
t <sub>WCS</sub>	LDAC Delay from CS HIGH	10			ns
t <sub>DS</sub>	Data Valid to CS LOW	0			ns
t <sub>DH</sub>	Data Valid after CS HIGH	0			ns
t <sub>LWD</sub>	LDAC LOW	50			ns
t <sub>RESET</sub>	RESET LOW Time	50			ns
t <sub>S</sub>	Settling Time			10	µs

Copyright © 2013-2015-2016 by Ando Ki AMBA APB BFM ( 27 )

BB DAC7724

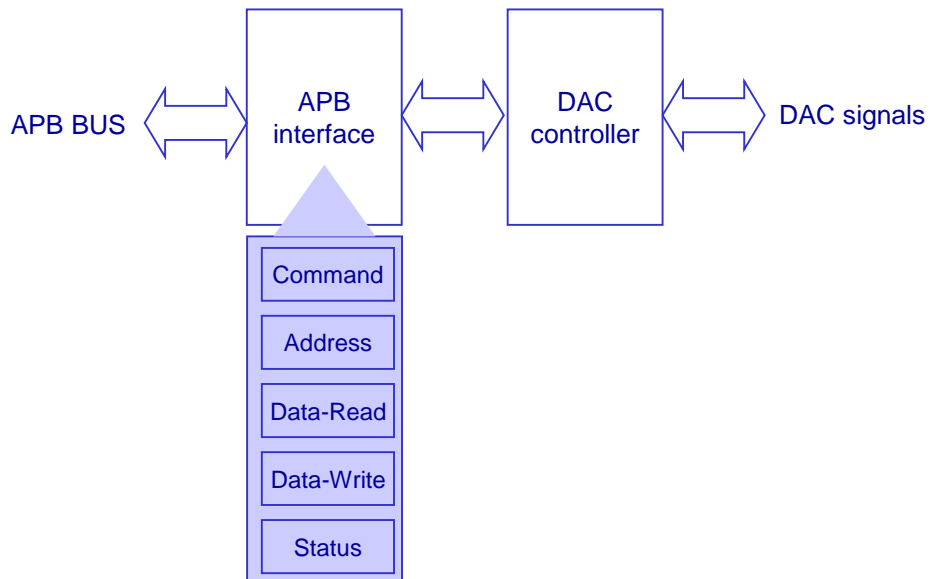


Assume that PCLK is 50Mhz, i.e., 20nsec period.  
40nsec



Copyright © 2013-2015-2016 by Ando Ki AMBA APB BFM ( 28 )

## BB DAC7724



Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 29 )

## References

- AMBA Specification, Rev 2.0, ARM Limited. (AMBA 2.0 APB)
- AMBA™ 3 APB Protocol v1.0, IHI 0024B, ARM, 2004. (AMBA 3.0 APB 1.0)
- AMBA® APB Protocol Version: 2.0, IHI 0024C (ID041610), ARM, 2010. (AMBA 4.0 APB 2.0)

Copyright © 2013-2015-2016 by Ando Ki

AMBA APB BFM ( 30 )