BFM-Based Verification Methods

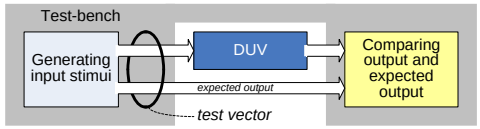
2013 – 2017 - 2018

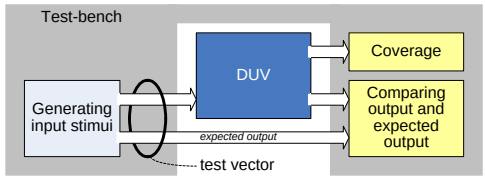
Ando Ki, Ph.D.
(adki@future-ds.com)

Agenda

- What is BFM
- Usage of BFM
- Task-based BFM
- File-driven BFM
- Native code-driven BFM

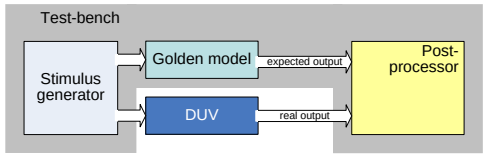
Test-bench

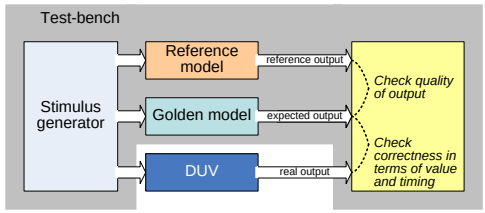




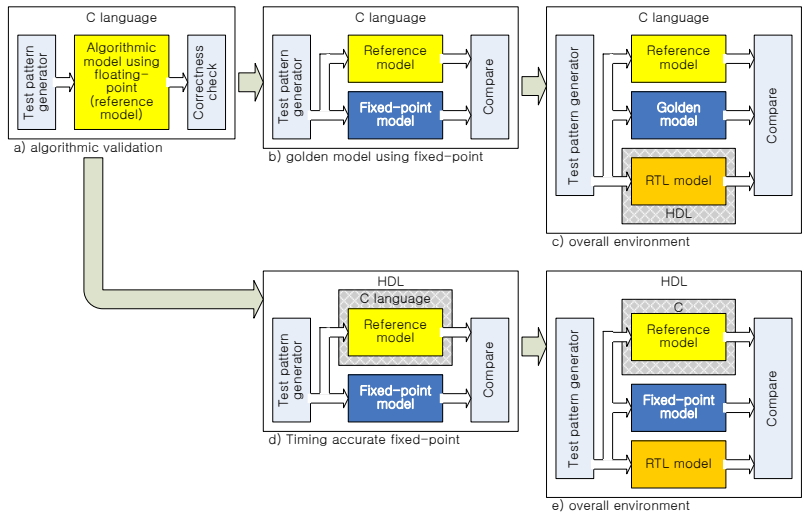
- Test-bench is an environment that verifies the functional and timing correctness of the design under verification (DUV) according to functional and timing specifications.
- Test-bench should apply test cases, i.e., test vectors as many as possible in order to cover everything that could happen to that design.
- Coverage assessment is required to measure how much has been verified.
  - code coverage
    - measures which part of implemented design has been exercised by the test vectors
  - functional coverage
    - measure how much functionality of the design has been exercised by the verification environment

Self-checking test-bench



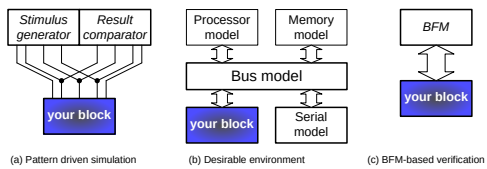


Complex function design



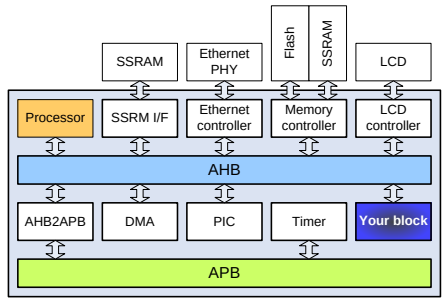
How about bus-based system

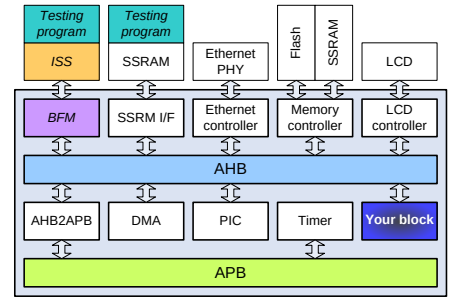
- What does testing scenario mean for bus-based system?
  - A combination of reads and writes
- What does expected result mean for bus-based system?



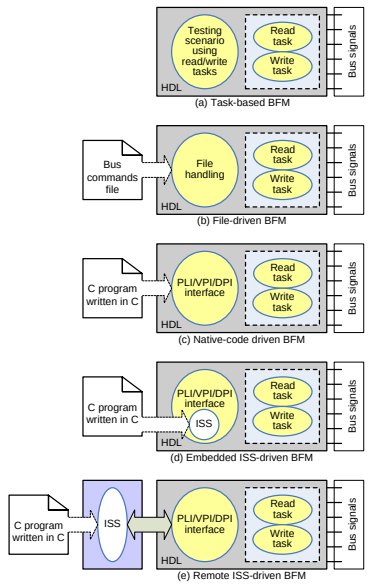
What is BFM?

- BFM: Bus Functional Model, Bus Functional Module
- BIM: Bus Interface Module
- BFM is a functional model generates bus transaction.
- BFM describes the behavior of the part (e.g. CPU) at the interface-level (bus transaction level) without modeling the internal operation of the part.
  - A sort of transactor



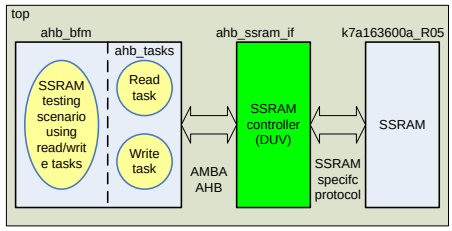


Usages of BFM



- Task-based BFM
- File-driven BFM
- Native-code driven BFM
- Embedded ISS-driven BFM
- Remote ISS-driven BFM

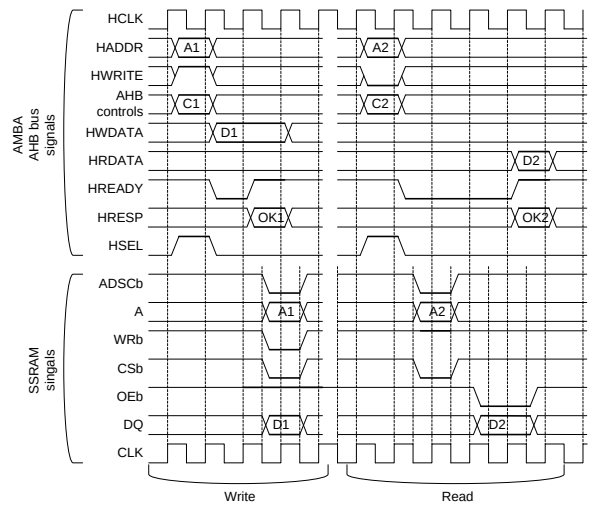
Task-based BFM example



- 'task' is a language construct of Verilog, which is a kind of sub-routine can contain time-controlling statements.
  - Note that 'function' is a similar with 'task', but it should execute in zeri simulation time.
- A relatively complex testing scenario can be build by combining tasks.
- BFM code should be re-written when testing scenario changes.

DUV: Design Under Verification
SSRAM: Synchronous Static Random Access Memory

AMBA AHB and SSRAM Timing Diagram



top.v

```
module top ;
      reg        HRESETn;
      reg        HCLK;
      ... ...
      wire       SRAM_CLK;
      ... ...
      wire [31:0] SRAM_D;
      wire [31:0] SRAM_D_O;
      wire [31:0] SRAM_D_I;
      wire        SRAM_D_T;
 //-------------------------------------------------
 // _T controls tri-state buffer; drive when low and
 // tristate when high
 assign SRAM_D    = SRAM_D_T  ? 32'bz   : SRAM_D_O;
 assign SRAM_D_I  = SRAM_D_T  ? SRAM_D  : 32'bz;
 assign SRAM_DP   = SRAM_DP_T ? 32'bz   : SRAM_DP_O;
 assign SRAM_DP_I = SRAM_DP_T ? SRAM_DP : 32'bz;
  /*************************************************************/
  ahb_bfm #(0, 32'hFFF) Uahb_bfm (
     .HRESETn(HRESETn),
     .HCLK   (HCLK),
     ... ...
  );

  ... ...
```

```
  ... ...
 ahb_ssram_if Uahb_ssram_if (
     .HRESETn   (HRESETn ),
     .HCLK      (HCLK    ),
     ... ...
 );

 k7a163600a Usram (
     .CLK   (SRAM_CLK  ),
     .A     (SRAM_A[18:0]),
     ... ...
     .DQ    (SRAM_D    ),
     .DQP   (SRAM_DP   )
 );

  /*************************************************************/
  always #5 HCLK <= ~HCLK;
  initial begin
     HCLK    = 0;
     HRESETn = 0;
     repeat (5) @ (posedge HCLK);
     HRESETn = 1;
  end
endmodule
```


bfm.v (1/2)

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

Testing 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);
     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);
     repeat (5) @ (posedge HCLK);
     $finish(2);
  end
```

bfm.v (2/2)

```
/*************************************************************/
  // 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
         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);
            ...
            ...
  `include "ahb_tasks.v"

endmodule
```

Testing scenario in details

ahb_tasks.v: AHB read tasks

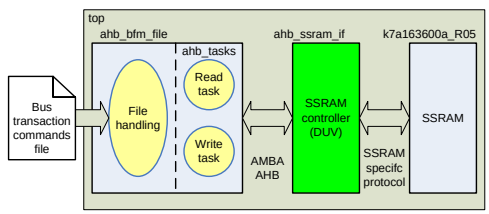
```
task ahb_read;
    input  [31:0] address;
    input  [2:0]  size;
    output [31:0] data;
    begin
       @ (posedge HCLK);
       HBUSREQ <= #1 1'b1;
       @ (posedge HCLK);
       while ((HGRANT!==1'b1)||(HREADY!==1'b1)) @ (posedge
HCLK);
       HBUSREQ <= #1 1'b0;
       HADDR   <= #1 address;
       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);
       HADDR  <= #1 32'b0;
       HPROT  <= #1 4'b0000; //`HPROT_OPCODE
       HTRANS <= #1 2'b0;
       HBURST <= #1 3'b0;
       HWRITE <= #1 1'b0;
       HSIZE  <= #1 3'b0;
       @ (posedge HCLK);
       while (HREADY===0) @ (posedge HCLK);
       data = HRDATA; // must be blocking
       if (HRESP!=2'b00) //if (HRESP!=`HRESP_OKAY)
          $display($time,, "ERROR: non OK response for read");
       @ (posedge HCLK);
    end
endtask
```

ahb_tasks.v: AHB write tasks

```
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);
        HADDR  <= #1 32'b0;
        HPROT  <= #1 4'b0000; //`HPROT_OPCODE
        HTRANS <= #1 2'b0;
        HBURST <= #1 3'b0;
        HWRITE <= #1 1'b0;
        HSIZE  <= #1 3'b0;
        HWDATA <= #1 data;
        @ (posedge HCLK);
        while (HREADY===0) @ (posedge HCLK);
        if (HRESP!=2'b00) //if (HRESP!=`HRESP_OKAY)
            $display($time,, "ERROR: non OK response write");
        HWDATA <= #1 0;
        @ (posedge HCLK);
    end
endtask
```


File-driven BFM example

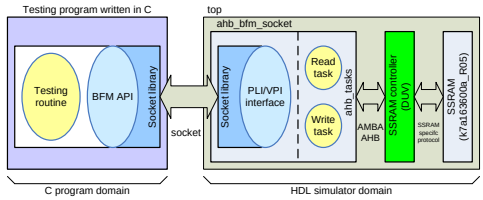
- Use file to build testing scenario
- This scheme does not need modify BFM code even testing scenario changes.




```
w 4 00 11111111
……
r 4 00 11111111
……
W 2 08 00009ABC
……
R 2 08 00009ABC
……
w 1 0C 00000087
……
r 1 0C 00000087
```

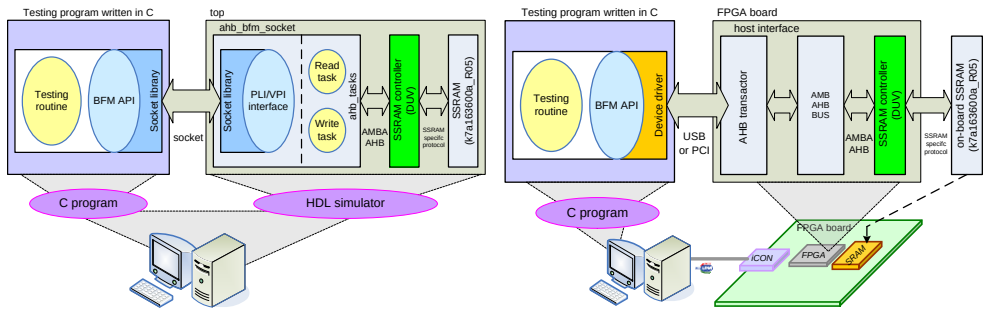
Native Code-driven BFM example

- Use program to test
- This scheme needs more advanced techniques including VPI/PLI/DPI, IPC and so on.
- This scheme does not need modify BFM code even testing scenario changes.
- This scheme make it possible to develop active/dynamic-verification scenario.
  - conditional check
  - conditional branch



PLI: Programming Language Interface
VPI: Verilog Programming Interface
DPI: Direct Programming Interface

Native Code-driven Transactor with FPGA



Reference

- 기안도, 시스템 집적 반도체 설계검증 환경과 기법, 제4장 BFM을 이용한 설계 자산의 기능검증, 홍릉과학출판사, 2008.