A Verilog BFM Methodology

Rodney Pesavento
rodney.pesavento@intel.com
Michael Privett
michael.privett@intel.com

Outline

- Model Overview
- Test Bench Overview
- Clocking Suggestions
- Adding Timing to a Model
- A State Machine BFM
- A Task BFM
- Conclusion

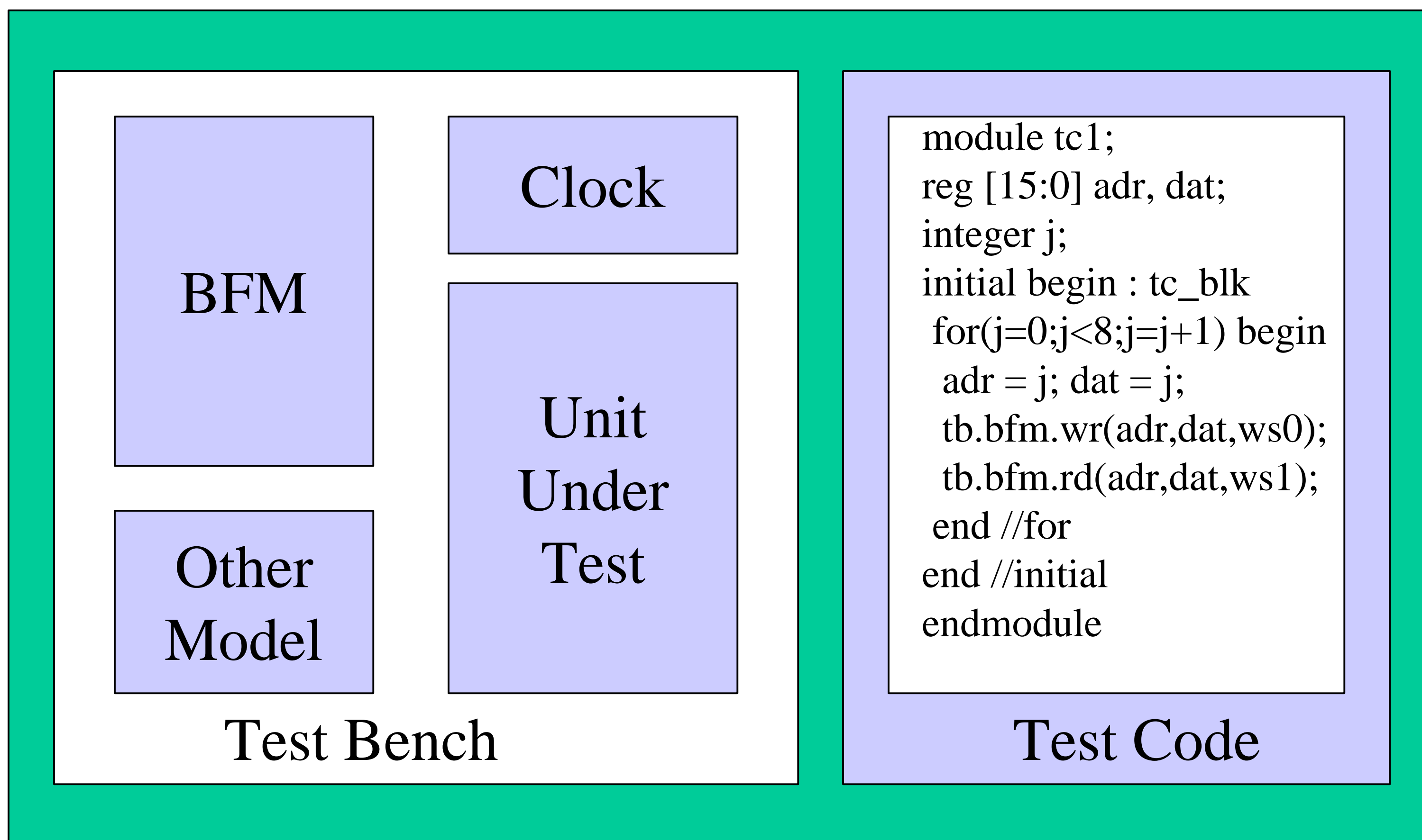
Types of Models

- Two Types of Models
- Hard Model
  - A full RTL Representation
  - A Gate Level Implementation
- Bus Functional Model (BFM)
  - Models only the Bus Interface
  - Models Bus Cycles

Test Bench Overview

- Verilog can have multiple top levels
  - One top level is for the Test Bench
    - The BFM
    - The Clocking unit
    - The Device Under Test
  - Another one is for the Test Code
    - Calls to the BFM
    - Signal timing control
    - test flow control

Test Environment Diagram



Speed Up A Single Clock

- Use a forever loop in the always block
- Use Blocking assignment
- Improves clock speed by 25%

```
// Normal Clock Block
always begin : clk_blk
  clk <= 0;
 #10 clk <= 1;
 #10;
end
```

```
// Improved Clock Block
always begin : clk_blk
  clk = 0;
  forever #10 clk = ~clk;
end
```

Speed Up Multiple Clocks

- Create a parameterized clock module
- Instance it for each clock
- Use keyword macromodule

```
macromodule clk_mod (clk);
  output clk;
  parameter  hper   = 5;
reg clk;
always begin
  clk = 0;
  forever #hper clk = ~clk;
end
endmodule
```

```
module test_bench;
  wire clk0, clk1, clk2;
  ebfm eb1 (…,…);
  xcntr u1 (…,…);
  clk_mod #(7) c0 (clk0);
  clk_mod #(20) c1 (clk1);
  clk_mod #(100) c2 (clk2);
endmodule
```

Adding Timing to a Model

- In-line Delays: sig_a <= #10 sig_b.
  - Distributed throughout model
  - Works for wires or regs
  - Slightly faster than Specify…
- Specify Block Delays
  - Concisely located in one place
  - Easy to enable or disable
  - Require ports to connect to wires

Specify Block Example

```
specify
  specparam data_rise 1:2:3;
  specparam data_fall 2:3:4;
  specparam data_float 3:4:5;

  (clk => data) = (data_rise, data_fall,data_float);

  specparam data_setup = 2:3:4;
  specparam data_hold = 0:1:2;

  $setup(data, posedge clk, data_setup);
  $hold(posedge clk, data, data_hold);
endspecify
```

•BFM Overview

- A BFM Generates Bus Cycles
  - Either with a State Machine
  - Or a Task
  - But, a Task is always used to call the cycle
- A Bus Cycle can be any type of cycle
  - Memory or IO, Read or Write
  - A Hold Cycle
  - An Interrupt Cycle

State Machine BFM

- Tasks are used to initiate the SM
- The SM creates the Bus Cycle
- The SM and Tasks use handshaking
  - Tasks assert a request for a bus cycle
  - The SM asserts a busy flag when active
  - This allows a task to return before the SM completes the bus cycle
- The BFM SM executes idle cycles

Partial SM BFM Example

```
// sm bfm body
always @(posedge clk or posedge rst)
begin
  if (rst) // initialize SM signals
  else case (state)
      `IDLE :  // assert idle state values
        busy <= 0;
        if (rd_req) state = `RD_CYC; ...
      `RD_CYC : // assert read values
        busy <= 1; ...
      `RD_RDY : // capture data on rdy
        if (rdy) begin
          check_data; ...
  endcase
end // always
```

An initial block would set up local signal values

IDLE: initiates cycles based on the req signals

RD_CYC: drives the bus to a read cycle

RD_RDY: Captures data on rdy. check_data errors on ws timeout; compares actual data to expected.

Partial SM BFM Example

The mrd task initiates a memory read cycle

It waits if the SM is busy, before asserting a request

Task inputs are assigned to local signals for SM use

The task does not terminate until the SM asserts busy

Test code references tasks via the Verilog hierarchy

```
// read task
task mrd;
begin
  if (busy) @(negedge busy);
  rd_req = 1;
  ldat = tdat;
  ladr = tadr;
  lws = tws;
  if (!busy) @(posedge busy);
end // task
```

```
module test_code;
tb.bfm.rst;
tb.bfm.mwr(adr,dat,ws1);
tb.bfm.mrd(adr,dat,ws3);
```

Task BFM

- Is only made up of tasks, no SM
- Each task creates its own Bus cycle
- The task is active for the entire cycle
- The BFM does not execute when idle
- Good for compute intensive block
- Hard to use with another BFM
  - Because the task does not return quickly

Partial Task BFM Example

Module has only one statement. The control is entirely in tasks.

A task BFM for an internal bus might contain block selects.

Generate wait states

Drive local data with task data

All Signals need to be set to Idle value at the end of a task

```
module ibfm(clk,rst,adr,dat,…);
assign dat = wr ? ldat : 8'hz;

task mwr; // tadr, tdat, tws
while (busy) @(posedge clk);
  busy = 1;
  sel0=tadr[15];  sel1=~tadr[15];
  adr = tadr[14:0];
  for (ii=1:ii<=tws;ii=ii+1)
    @(posedge clk);
  wr = 1;
  ldat = tdat;
  @(posedge clk);
  // assign signals to idle values
endtask
```

Conclusion

- Task BFMs are good for Unit Tests
  - Faster Simulation Speeds
- SM BFMs are good for Chip Tests
  - Greater Functionality
- Use Specify blocks for Timing
- Use Blocking assignments
  - For Temporary Variables
  - For Clocks in forever loops