

CSE/CEN 598

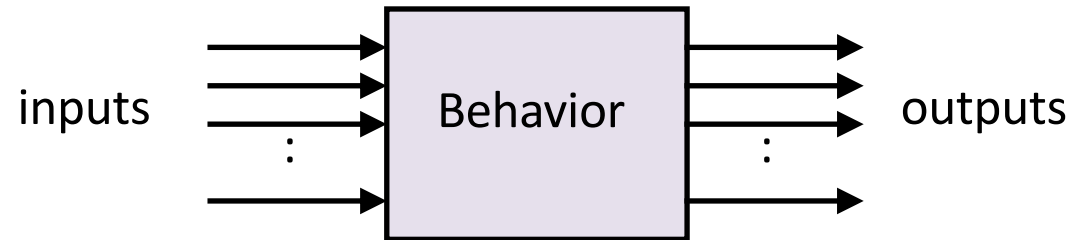
Hardware Security & Trust

Trusted Digital System Design:
Verilog Fundamentals II

Prof. Michel A. Kinsy & Mishel Paul

Computer System Description

- A system is a set of related components that works as a whole to achieve a goal.
- A system contains:
 - Inputs
 - Behavior
 - Outputs



- Behavior is a function that translates inputs to outputs

Verilog Fundamentals


- Data types
- Structural Verilog
- Functional Verilog
 - Gate level
 - Register transfer level
 - High-level behavioral

Verilog Test Bench Basics

- Test Bench – A wrapper module to apply test inputs to a “Device Under Test”
 - Also written in Verilog
- Three main components
 - Device Under Test Instantiation
 - Test Inputs
 - Output Checking

Example Test Bench


- Declare test bench module
 - No inputs/output



```
6 module tb_mm_uart ();
7
8 localparam DATA_WIDTH      = 8;
9 localparam ADDR_WIDTH      = 8;
10 localparam UART_TXDATA_ADDR = 8'd0;
11 localparam UART_RXDATA_ADDR = 8'd4;
12
13 reg clock;
14 reg clock_baud;
15 reg reset;
16
17 // UART Rx/Tx
18 reg uart_rx;
19 wire uart_tx;
20
21 // Memory Mapped Port
22 reg readEnable;
23 reg writeEnable;
24 reg [DATA_WIDTH/8-1:0] writeByteEnable;
25 reg [ADDR_WIDTH-1:0] address;
26 reg [DATA_WIDTH-1:0] writeData;
27 wire [DATA_WIDTH-1:0] readData;
28 wire ready;
29
```

Example Test Bench

- Declare test bench module
 - No inputs/output
- Declare parameters for module/DUT



```
6 module tb_mm_uart ();
7
8 localparam DATA_WIDTH      = 8;
9 localparam ADDR_WIDTH      = 8;
10 localparam UART_TXDATA_ADDR = 8'd0;
11 localparam UART_RXDATA_ADDR = 8'd4;
12
13 reg clock;
14 reg clock_baud;
15 reg reset;
16
17 // UART Rx/Tx
18 reg uart_rx;
19 wire uart_tx;
20
21 // Memory Mapped Port
22 reg readEnable;
23 reg writeEnable;
24 reg [DATA_WIDTH/8-1:0] writeByteEnable;
25 reg [ADDR_WIDTH-1:0] address;
26 reg [DATA_WIDTH-1:0] writeData;
27 wire [DATA_WIDTH-1:0] readData;
28 wire ready;
29
```

Example Test Bench

- Declare test bench module
 - No inputs/output
- Declare parameters for module/DUT
- Declare test inputs/outputs
 - "reg" means input for a test bench
 - "wire" means output for a test bench

```
6 module tb_mm_uart ();
7
8 localparam DATA_WIDTH      = 8;
9 localparam ADDR_WIDTH      = 8;
10 localparam UART_TXDATA_ADDR = 8'd0;
11 localparam UART_RXDATA_ADDR = 8'd4;
12
13 reg clock;
14 reg clock_baud;
15 reg reset;
16
17 // UART Rx/Tx
18 reg uart_rx;
19 wire uart_tx;
20
21 // Memory Mapped Port
22 reg readEnable;
23 reg writeEnable;
24 reg [DATA_WIDTH/8-1:0] writeByteEnable;
25 reg [ADDR_WIDTH-1:0] address;
26 reg [DATA_WIDTH-1:0] writeData;
27 wire [DATA_WIDTH-1:0] readData;
28 wire ready;
29
```

Example Test Bench

- Declare test bench module
 - No inputs/output
- Declare parameters for module/DUT
- Declare test inputs/outputs
 - "reg" means input for a test bench
 - "wire" means output for a test bench
- Some reg/wires are more than 1 bit
 - readData is DATA_BITS large

```
6 module tb_mm_uart ();
7
8 localparam DATA_WIDTH      = 8;
9 localparam ADDR_WIDTH      = 8;
10 localparam UART_TXDATA_ADDR = 8'd0;
11 localparam UART_RXDATA_ADDR = 8'd4;
12
13 reg clock;
14 reg clock_baud;
15 reg reset;
16
17 // UART Rx/Tx
18 reg uart_rx;
19 wire uart_tx;
20
21 // Memory Mapped Port
22 reg readEnable;
23 reg writeEnable;
24 reg [DATA_WIDTH/8-1:0] writeByteEnable;
25 reg [ADDR_WIDTH-1:0] address;
26 reg [DATA_WIDTH-1:0] writeData;
27 wire [DATA_WIDTH-1:0] readData;
28 wire ready;
29
```


Example Test Bench

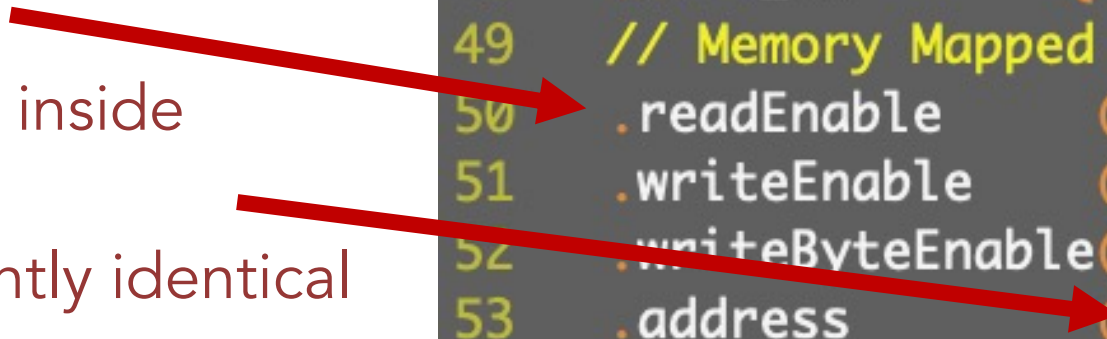
- Instantiate Device Under Test

```
44 mm_uart DUT (  
45     .clock      (clock),  
46     .reset      (reset),  
47     .uart_rx    (uart_rx),  
48     .uart_tx    (uart_tx),  
49     // Memory Mapped Port  
50     .readEnable (readEnable),  
51     .writeEnable (writeEnable),  
52     .writeByteEnable (writeByteEnable),  
53     .address    (address),  
54     .writeData  (writeData),  
55     .readData   (readData),  
56     .ready      (ready)  
57 );
```

Example Test Bench

- Instantiate Device Under Test
- Connect DUT inputs/outputs to test bench signals
 - DUT I/O after "."
 - Test bench signals inside parenthesis
 - Names are frequently identical

```
44 mm_uart DUT (  
45     .clock          (clock),  
46     .reset          (reset),  
47     .uart_rx        (uart_rx),  
48     .uart_tx        (uart_tx),  
49     // Memory Mapped Port  
50     .readEnable     (readEnable),  
51     .writeEnable    (writeEnable),  
52     .writeByteEnable(writeByteEnable),  
53     .address        (address),  
54     .writeData      (writeData),  
55     .readData       (readData),  
56     .ready          (ready)  
57 );
```



Example Test Bench


- Test Stimulus
 - Apply inputs at given times of simulation

```
138 //-----  
139 //  
140 // Simulation  
141 //  
142 //-----  
143  
144 //100MHz CLK  
145 always #5 clock = ~clock;  
146 always #50 clock_baud = ~clock_baud;  
147  
148 initial begin  
149     initialize;  
150     inactive(100);  
151     test_init;  
152     inactive(100);  
153     test_tx;  
154     inactive(100);  
155     test_rx;  
156     inactive(100);  
157     test_exit;  
158 end
```

Example Test Bench

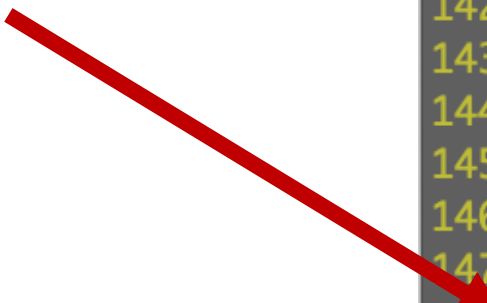
- Test Stimulus
 - Apply inputs at given times of simulation
- Clock Setup
 - Toggle "clock" signal every 5 simulation timesteps
 - "Always" block – repeats in a loop

```
138 //-----  
139 //  
140 // Simulation  
141 //  
142 //-----  
143  
144 //100MHz CLK  
145 always #5 clock = ~clock;  
146 always #50 clock_baud = ~clock_baud;  
147  
148 initial begin  
149     initialize;  
150     inactive(100);  
151     test_init;  
152     inactive(100);  
153     test_tx;  
154     inactive(100);  
155     test_rx;  
156     inactive(100);  
157     test_exit;  
158 end
```



Example Test Bench

- DUT test inputs
 - “Initial” Block – Apply test inputs once

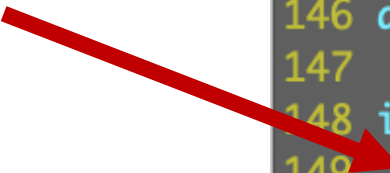


```
138 //-----  
139 //  
140 // Simulation  
141 //  
142 //-----  
143  
144 //100MHz CLK  
145 always #5 clock = ~clock;  
146 always #50 clock_baud = ~clock_baud;  
147  
148 initial begin  
149     initialize;  
150     inactive(100);  
151     test_init;  
152     inactive(100);  
153     test_tx;  
154     inactive(100);  
155     test_rx;  
156     inactive(100);  
157     test_exit;  
158 end
```

Example Test Bench

- DUT test inputs
 - “Initial” Block – Apply test inputs once
- Use “Tasks” to apply specific test inputs
 - Initialize, inactive, test_tx, test_rx, and test_exit are tasks
 - A task groups a set of inputs for re-use

```
138 //-----
139 //
140 // Simulation
141 //
142 //-----
143
144 //100MHz CLK
145 always #5 clock = ~clock;
146 always #50 clock_baud = ~clock_baud;
147
148 initial begin
149     initialize;
150     inactive(100);
151     test_init;
152     inactive(100);
153     test_tx;
154     inactive(100);
155     test_rx;
156     inactive(100);
157     test_exit;
158 end
```



Example Test Bench

- Initialize task
 - \$display writes to console

```
67 task initialize;  
68 begin  
69     $display("INITIALIZING...");  
70     clock = 1'b1;  
71     clock_baud = 1'b1;  
72     uart_rx = 1'b1;  
73     reset = 1'b1;  
74     readEnable      = 1'b0;  
75     writeEnable     = 1'b0;  
76     writeByteEnable = 4'h0;  
77     address         = 32'h0;  
78     writeData        = 32'h0;  
79  
80     repeat (3) @ (posedge clock);  
81     reset = 1'b0;  
82 end  
83 endtask
```


Example Test Bench

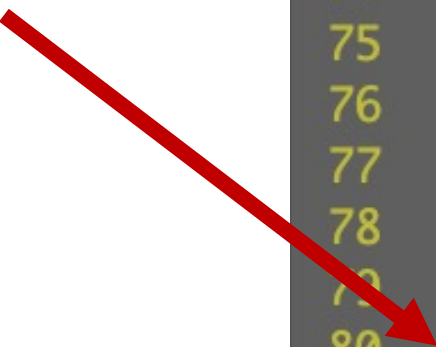
- Initialize task
 - \$display writes to console
 - Set inputs to specific values
 - All set at the same simulation time

```
67 task initialize;
68 begin
69     $display("INITIALIZING...");
70     clock = 1'b1;
71     clock_baud = 1'b1;
72     uart_rx = 1'b1;
73     reset = 1'b1;
74     readEnable      = 1'b0;
75     writeEnable     = 1'b0;
76     writeByteEnable = 4'h0;
77     address         = 32'h0;
78     writeData       = 32'h0;
79
80     repeat (3) @ (posedge clock);
81     reset = 1'b0;
82 end
83 endtask
```


Example Test Bench

- Initialize task
 - \$display writes to console
 - Set inputs to specific values
 - All set at the same simulation time
- Repeat statement waits for 3 clock cycles
 - Reset signal stays high for 3 clock cycles


```
67 task initialize;  
68 begin  
69   $display("INITIALIZING...");  
70   clock = 1'b1;  
71   clock_baud = 1'b1;  
72   uart_rx = 1'b1;  
73   reset = 1'b1;  
74   readEnable      = 1'b0;  
75   writeEnable     = 1'b0;  
76   writeByteEnable = 4'h0;  
77   address         = 32'h0;  
78   writeData       = 32'h0;  
79  
80   repeat (3) @ (posedge clock);  
81   reset = 1'b0;  
82 end  
83 endtask
```



Example Test Bench

- Initialize task
 - \$display writes to console
 - Set inputs to specific values
 - All set at the same simulation time
- Repeat statement waits for 3 clock cycles
 - Reset signal stays high for 3 clock cycles
- Apply new inputs after waiting
 - Reset signal lowered

```
67 task initialize;
68 begin
69     $display("INITIALIZING...");
70     clock = 1'b1;
71     clock_baud = 1'b1;
72     uart_rx = 1'b1;
73     reset = 1'b1;
74     readEnable      = 1'b0;
75     writeEnable     = 1'b0;
76     writeByteEnable = 4'h0;
77     address         = 32'h0;
78     writeData       = 32'h0;
79
80     repeat (3) @ (posedge clock);
81     reset = 1'b0;
82 end
83 endtask
```



Additional Resources

- Asic-world.com
 - Verilog Tutorial – “Art of Writing Test Benches”
 - Additional info on Verilog syntax
 - Getting started examples
- yosyshq.net/yosys/
 - Documentation for open-source synthesis tool
 - Used in Project 3

Obfuscated Netlists

- Verilog Obfuscated with Yosys
 - Read in Verilog
 - Parse to internal representation
 - Write out to plain Verilog again
- Obfuscation
 - Parsed Verilog is already hard to read/understand
 - Renamed wires, regs, and modules
- Top-level ports still the same

```
648 assign _19_ = _48_(8'h00, { _29_[7:0],
649 assign _22_ = _23_ ? _19_ : 8'h00;
650 assign _24_ = _25_ ? 8'h00 : _22_;
651 assign _26_ = _27_ ? 8'h00 : _24_;
652 assign _28_ = !TX89tb;
653 UXmg CJTDIIx (
654     .ET5uS(pRm6vGPowaWGO),
655     .Jmv(m6Powv),
656     .Kogc(1'h0),
657     .X8qqE(GooRagLjgP4dw),
658     .ffkGpLq4E(MKHJv93PQ0dt0r1lG),
659     .jT1j(TX89tb),
660     .k7ZJ(UMIaxeJUmgYR),
661     .kfno(ewTNYHIdEpVj0),
662     .sMr1R(NXKSJycPvnP0a),
663     .vgGEfgslfn(Zl6sD57Nd43NzyG19f)
664 );
665
```

Take away points

- Structural Verilog enables us to describe a hardware schematic textually
- Verilog can model hardware at three levels of abstraction
 - Gate level, register transfer level, and behavioral
- Understanding the Verilog execution semantics is critical for understanding blocking + non-blocking assignments
- Designers must have the hardware they are trying to create in mind when they write their Verilog