CSE/CEN 598
Hardware Security & Trust

Trusted Digital System Design: Verilog Fundamentals II

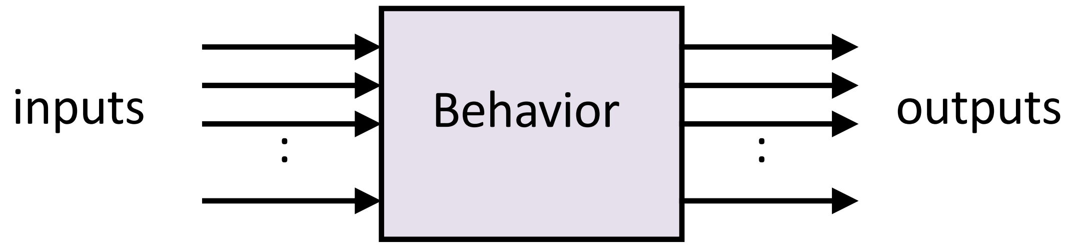
Prof. Michel A. Kinsy & Mishel Paul

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Ira A. Fulton Schools of Engineering
Arizona State University

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

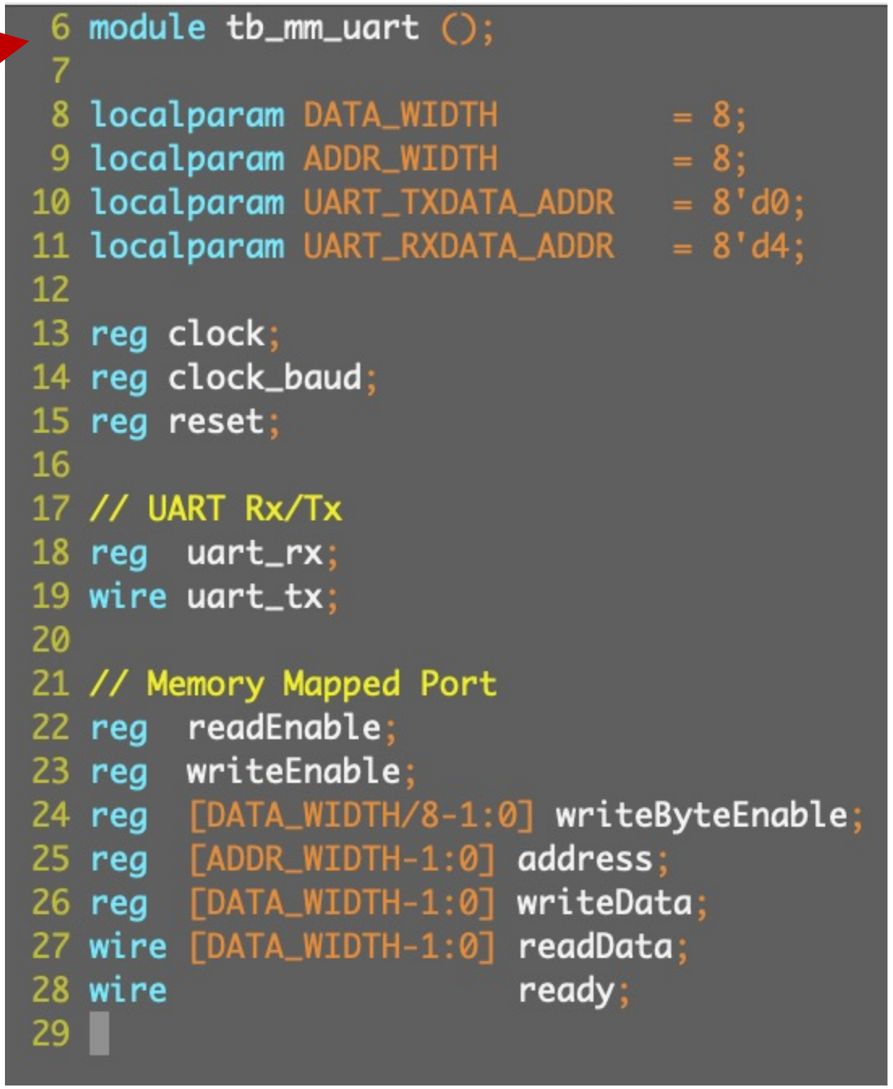
- Data types
- Structural Verilog
- Functional Verilog
  - Gate level
  - Register transfer level
  - High-level behavioral

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

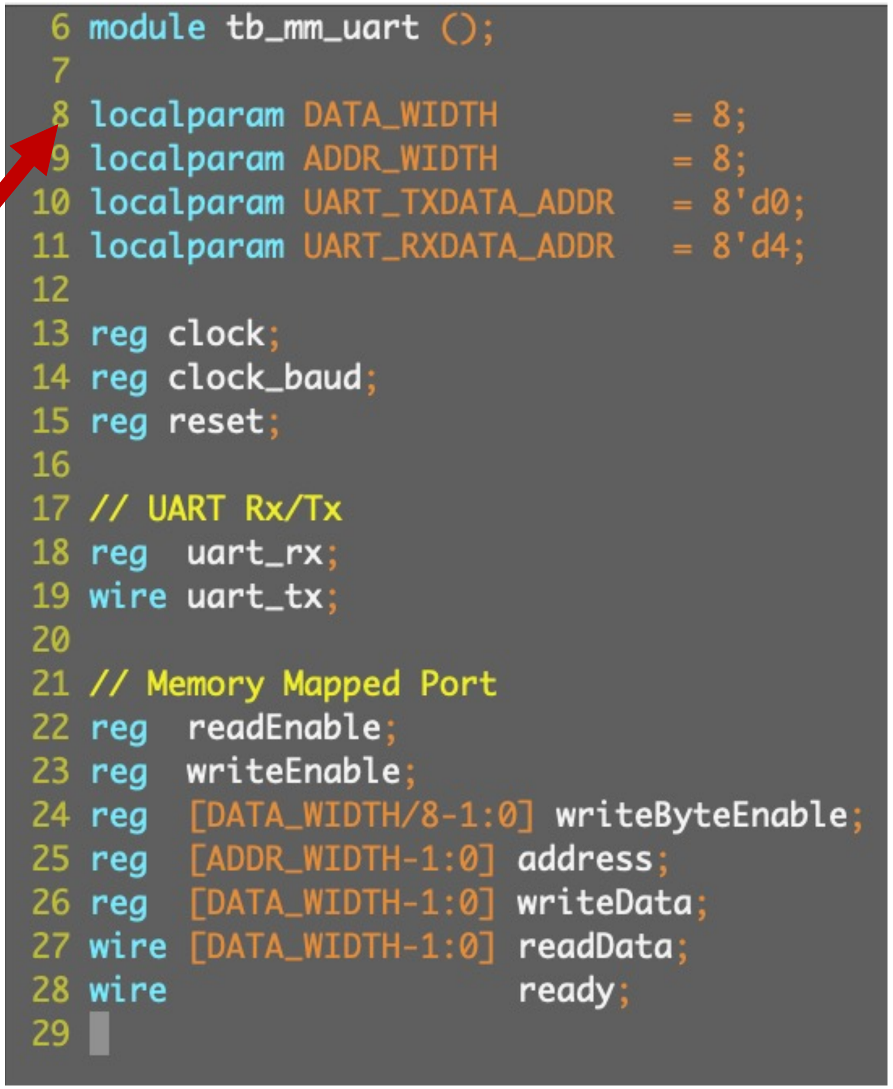
- Declare test bench module
  - No inputs/output



```
module tb_mm_uart ();

localparam DATA_WIDTH          = 8;
localparam ADDR_WIDTH          = 8;
localparam UART_TXDATA_ADDR    = 8'd0;
localparam UART_RXDATA_ADDR    = 8'd4;

reg clock;
reg clock_baud;
reg reset;

// UART Rx/Tx
reg  uart_rx;
wire uart_tx;

// Memory Mapped Port
reg  readEnable;
reg  writeEnable;
reg  [DATA_WIDTH/8-1:0] writeByteEnable;
reg  [ADDR_WIDTH-1:0] address;
reg  [DATA_WIDTH-1:0] writeData;
wire [DATA_WIDTH-1:0] readData;
wire                  ready;

```

Example Test Bench

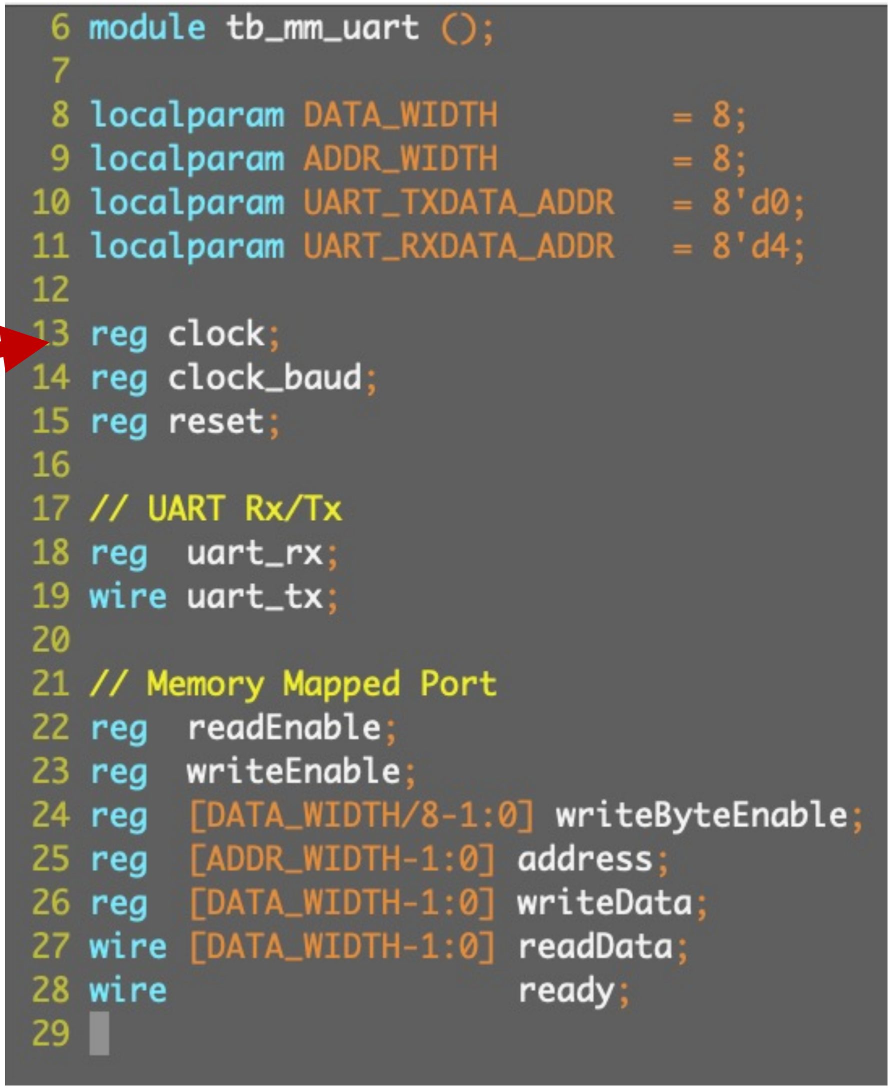
- Declare test bench module
  - No inputs/output
- Declare parameters for module/DUT



```verilog
module tb_mm_uart ();

localparam DATA_WIDTH         = 8;
localparam ADDR_WIDTH         = 8;
localparam UART_TXDATA_ADDR   = 8'd0;
localparam UART_RXDATA_ADDR   = 8'd4;

reg clock;
reg clock_baud;
reg reset;

// UART Rx/Tx
reg  uart_rx;
wire uart_tx;

// Memory Mapped Port
reg  readEnable;
reg  writeEnable;
reg  [DATA_WIDTH/8-1:0] writeByteEnable;
reg  [ADDR_WIDTH-1:0] address;
reg  [DATA_WIDTH-1:0] writeData;
wire [DATA_WIDTH-1:0] readData;
wire                  ready;

```

Example Test Bench

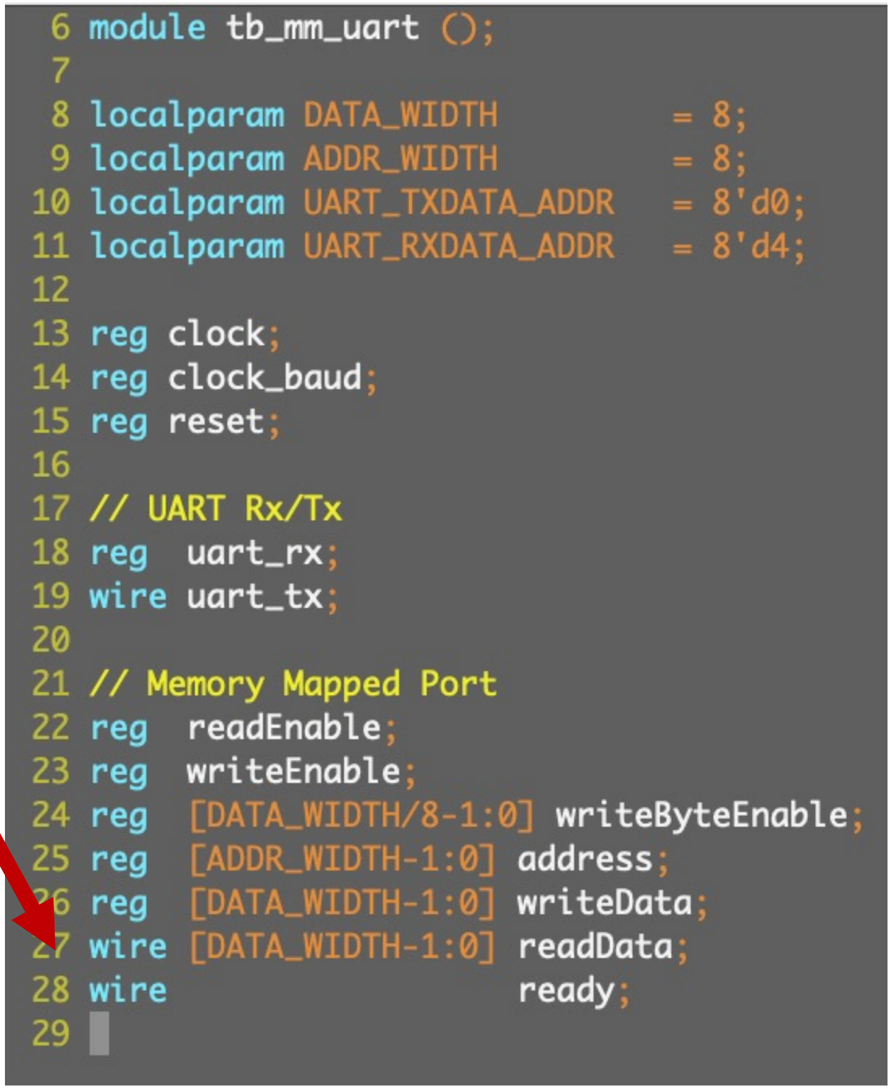
- Declare test bench module
  - No inputs/output
- Declare parameters for module/DUT
- Declare test inputs/outputs
  - "reg" means input for a test bench
  - "wire" means output for a test bench



```
module tb_mm_uart ();

localparam DATA_WIDTH          = 8;
localparam ADDR_WIDTH          = 8;
localparam UART_TXDATA_ADDR    = 8'd0;
localparam UART_RXDATA_ADDR    = 8'd4;

reg clock;
reg clock_baud;
reg reset;

// UART Rx/Tx
reg  uart_rx;
wire uart_tx;

// Memory Mapped Port
reg  readEnable;
reg  writeEnable;
reg  [DATA_WIDTH/8-1:0] writeByteEnable;
reg  [ADDR_WIDTH-1:0] address;
reg  [DATA_WIDTH-1:0] writeData;
wire [DATA_WIDTH-1:0] readData;
wire                  ready;

```

Example Test Bench

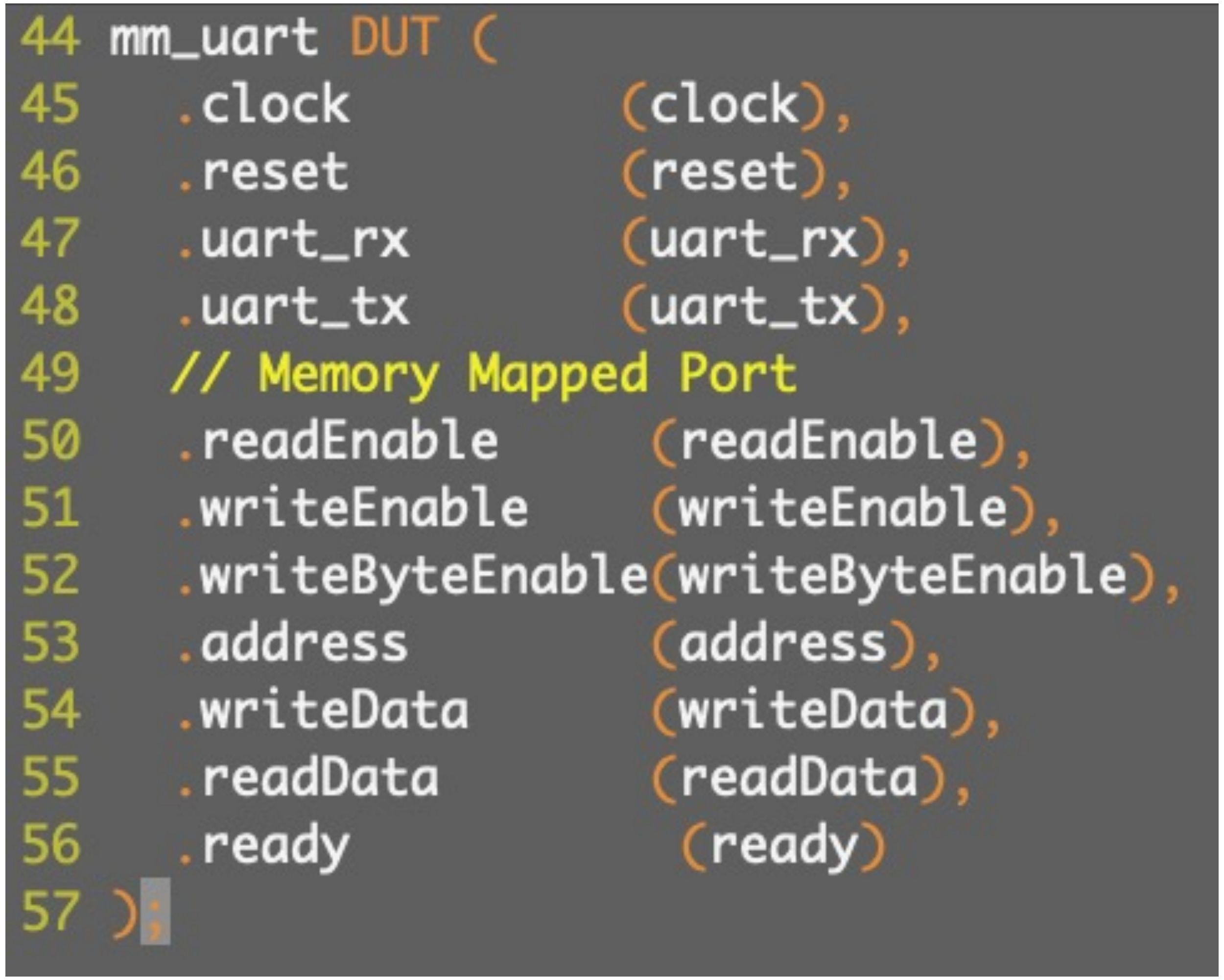
- Declare test bench module
  - No inputs/output
- Declare parameters for module/DUT
- Declare test inputs/outputs
  - "reg" means input for a test bench
  - "wire" means output for a test bench
- Some reg/wires are more than 1 bit
  - readData is DATA_BITS large



```
module tb_mm_uart ();

localparam DATA_WIDTH          = 8;
localparam ADDR_WIDTH          = 8;
localparam UART_TXDATA_ADDR    = 8'd0;
localparam UART_RXDATA_ADDR    = 8'd4;

reg clock;
reg clock_baud;
reg reset;

// UART Rx/Tx
reg  uart_rx;
wire uart_tx;

// Memory Mapped Port
reg  readEnable;
reg  writeEnable;
reg  [DATA_WIDTH/8-1:0] writeByteEnable;
reg  [ADDR_WIDTH-1:0] address;
reg  [DATA_WIDTH-1:0] writeData;
wire [DATA_WIDTH-1:0] readData;
wire                  ready;

```

Example Test Bench

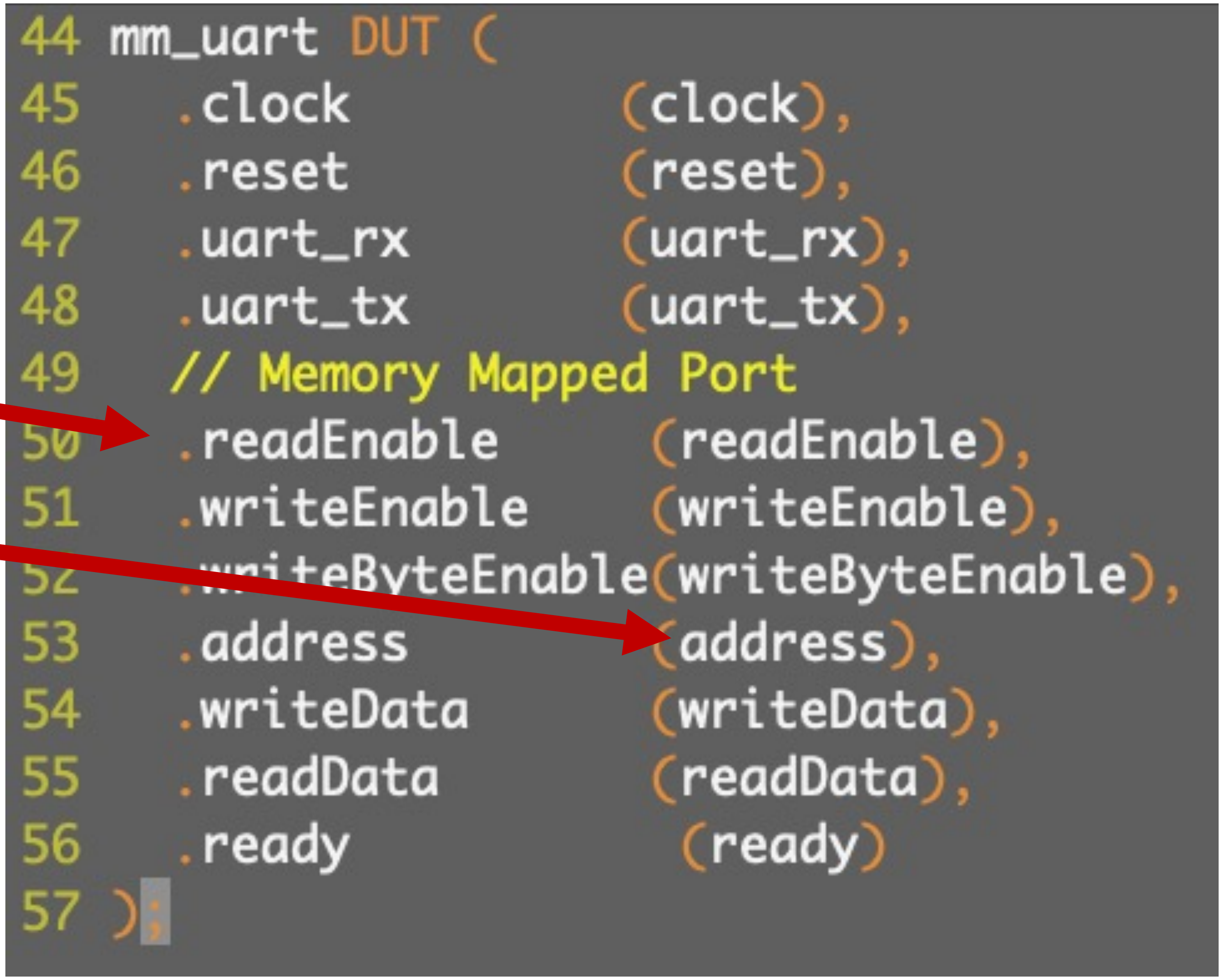
- Instantiate Device Under Test



```
mm_uart DUT (
    .clock          (clock),
    .reset          (reset),
    .uart_rx        (uart_rx),
    .uart_tx        (uart_tx),
    // Memory Mapped Port
    .readEnable      (readEnable),
    .writeEnable     (writeEnable),
    .writeByteEnable(writeByteEnable),
    .address         (address),
    .writeData       (writeData),
    .readData        (readData),
    .ready            (ready)
);
```

Example Test Bench

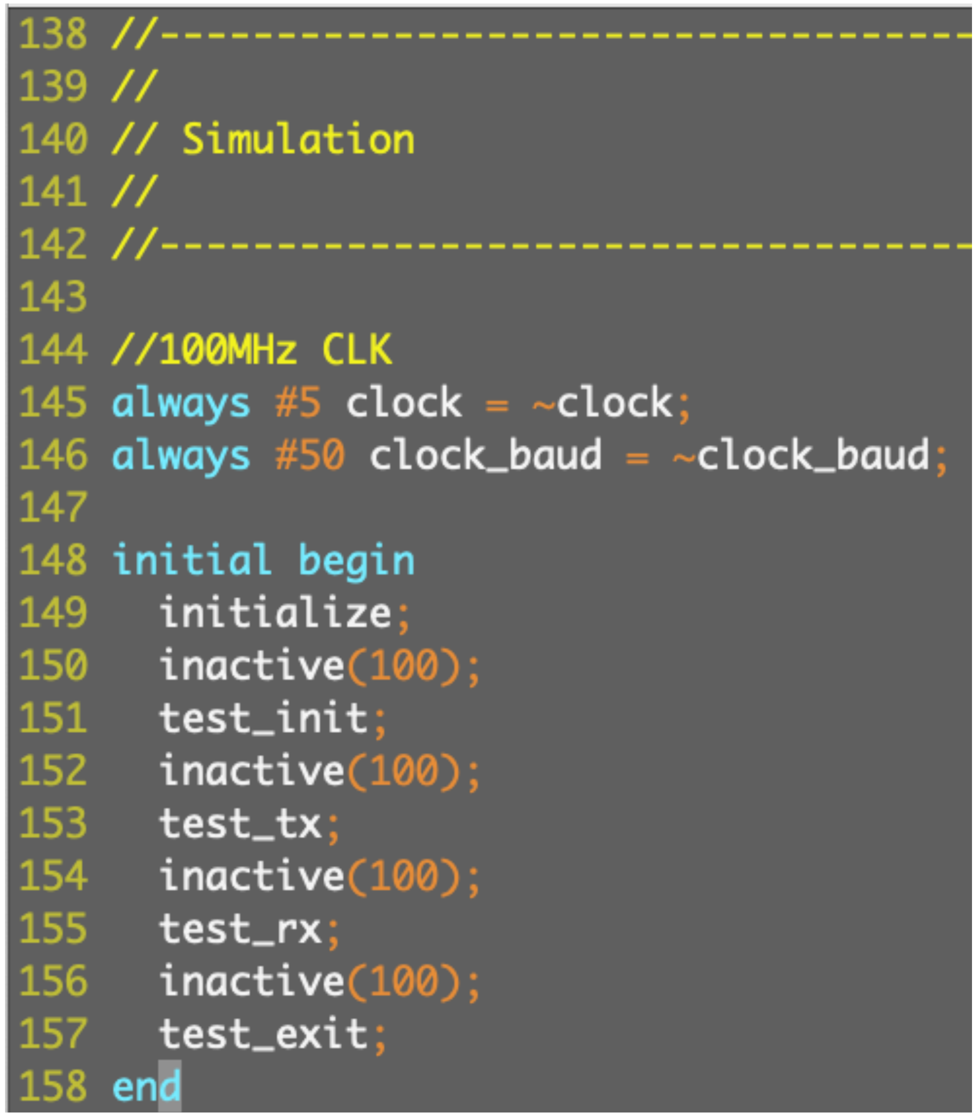
- Instantiate Device Under Test
- Connect DUT inputs/outputs to test bench signals
  - DUT I/O after "."
  - Test bench signals inside parenthesis
  - Names are frequently identical



```
mm_uart DUT (
    .clock          (clock),
    .reset          (reset),
    .uart_rx        (uart_rx),
    .uart_tx        (uart_tx),
    // Memory Mapped Port
    .readEnable     (readEnable),
    .writeEnable    (writeEnable),
    .writeByteEnable(writeByteEnable),
    .address        (address),
    .writeData      (writeData),
    .readData       (readData),
    .ready           (ready)
);
```

Example Test Bench

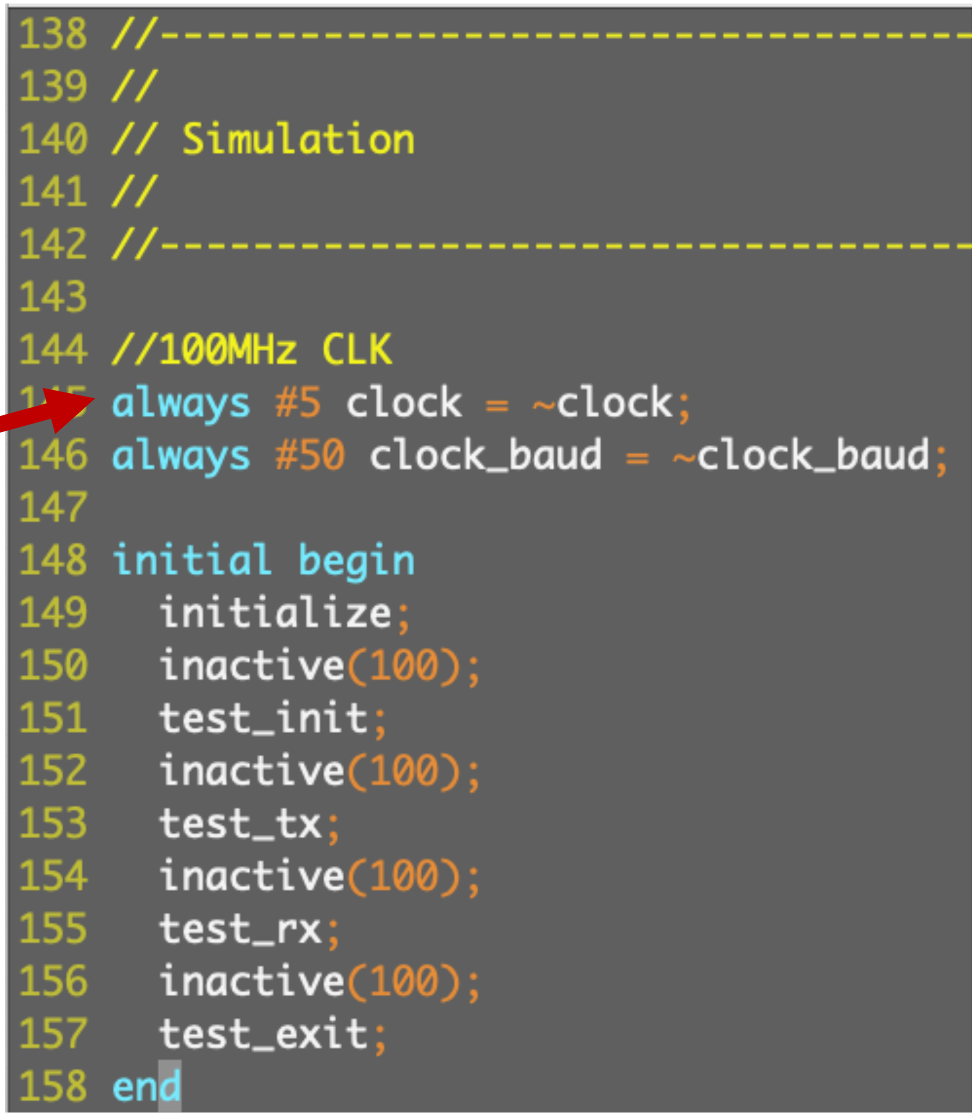
- Test Stimulus
  - Apply inputs at given times of simulation



```
//------------------------------------
//
// Simulation
//
//------------------------------------

//100MHz CLK
always #5 clock = ~clock;
always #50 clock_baud = ~clock_baud;

initial begin
  initialize;
  inactive(100);
  test_init;
  inactive(100);
  test_tx;
  inactive(100);
  test_rx;
  inactive(100);
  test_exit;
end
```

Example Test Bench

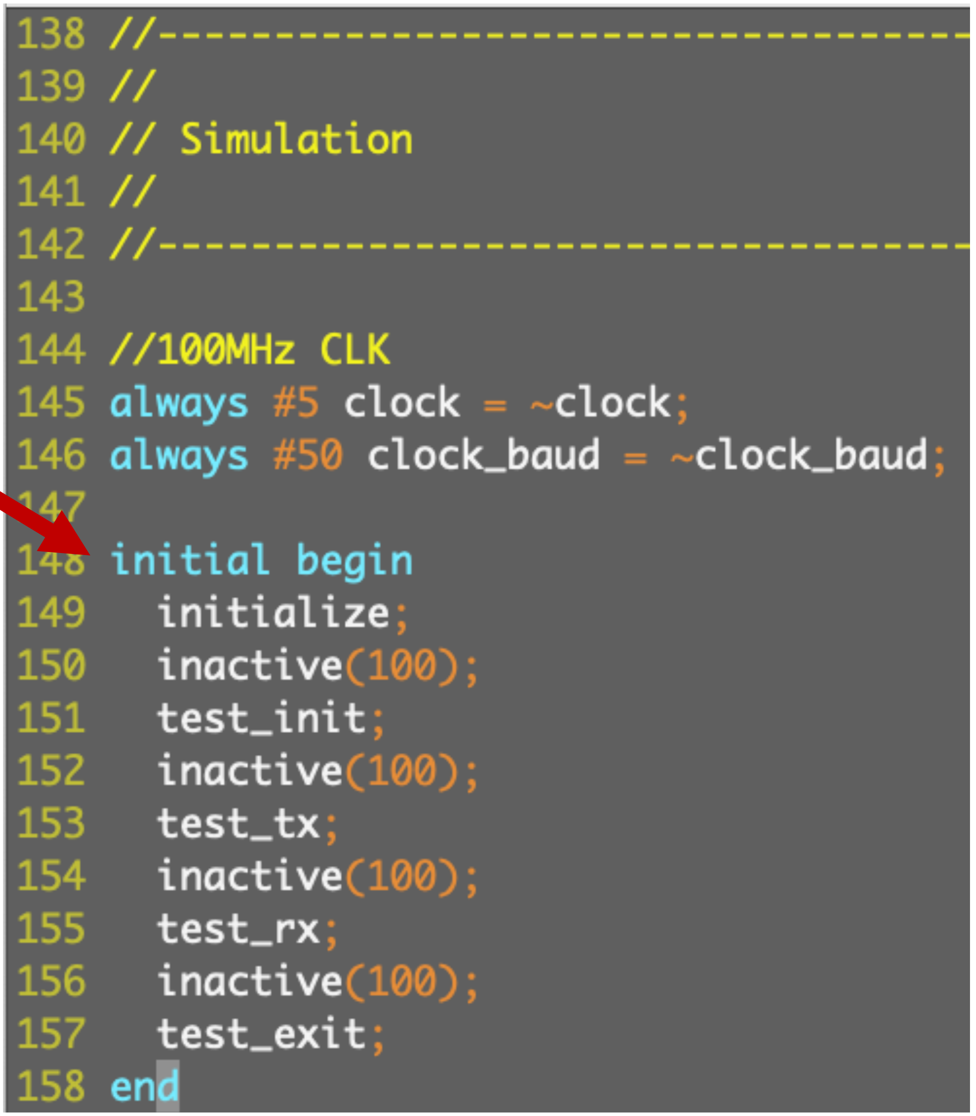
- Test Stimulus
  - Apply inputs at given times of simulation
- Clock Setup
  - Toggle "clock" signal every 5 simulation timesteps
  - "Always" block – repeats in a loop



```
//-------------------------------------
//
// Simulation
//
//-------------------------------------

//100MHz CLK
always #5 clock = ~clock;
always #50 clock_baud = ~clock_baud;

initial begin
  initialize;
  inactive(100);
  test_init;
  inactive(100);
  test_tx;
  inactive(100);
  test_rx;
  inactive(100);
  test_exit;
end
```

Example Test Bench

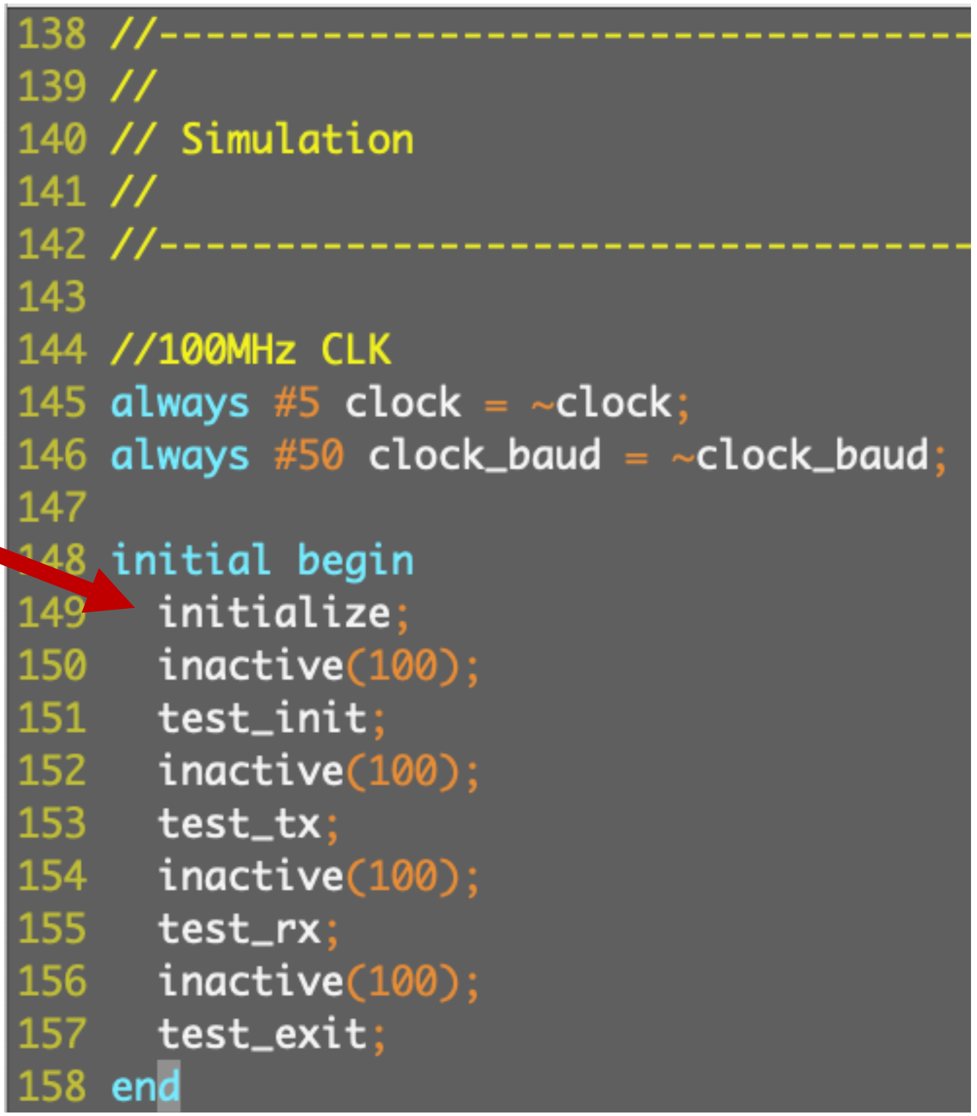
- DUT test inputs
  - "Initial" Block – Apply test inputs once



```
//------------------------------------
//
// Simulation
//
//------------------------------------

//100MHz CLK
always #5 clock = ~clock;
always #50 clock_baud = ~clock_baud;

initial begin
  initialize;
  inactive(100);
  test_init;
  inactive(100);
  test_tx;
  inactive(100);
  test_rx;
  inactive(100);
  test_exit;
end
```

Example Test Bench

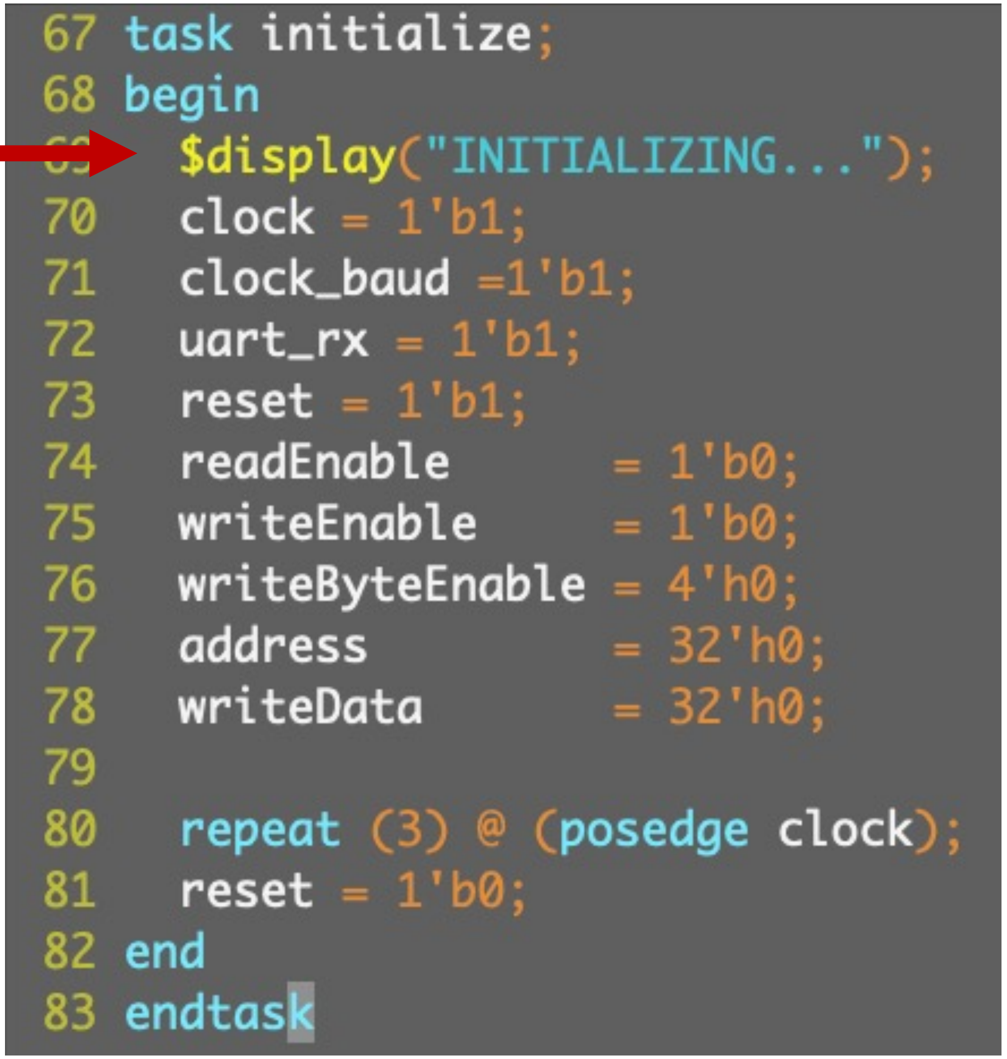
- DUT test inputs
  - "Initial" Block – Apply test inputs once
- Use "Tasks" to apply specific test inputs
  - Initialize, inactive, test_tx, test_rx, and test_exit are tasks
  - A task groups a set of inputs for re-use



```
//-----------------------------------
//
// Simulation
//
//-----------------------------------

//100MHz CLK
always #5 clock = ~clock;
always #50 clock_baud = ~clock_baud;

initial begin
  initialize;
  inactive(100);
  test_init;
  inactive(100);
  test_tx;
  inactive(100);
  test_rx;
  inactive(100);
  test_exit;
end
```

Example Test Bench

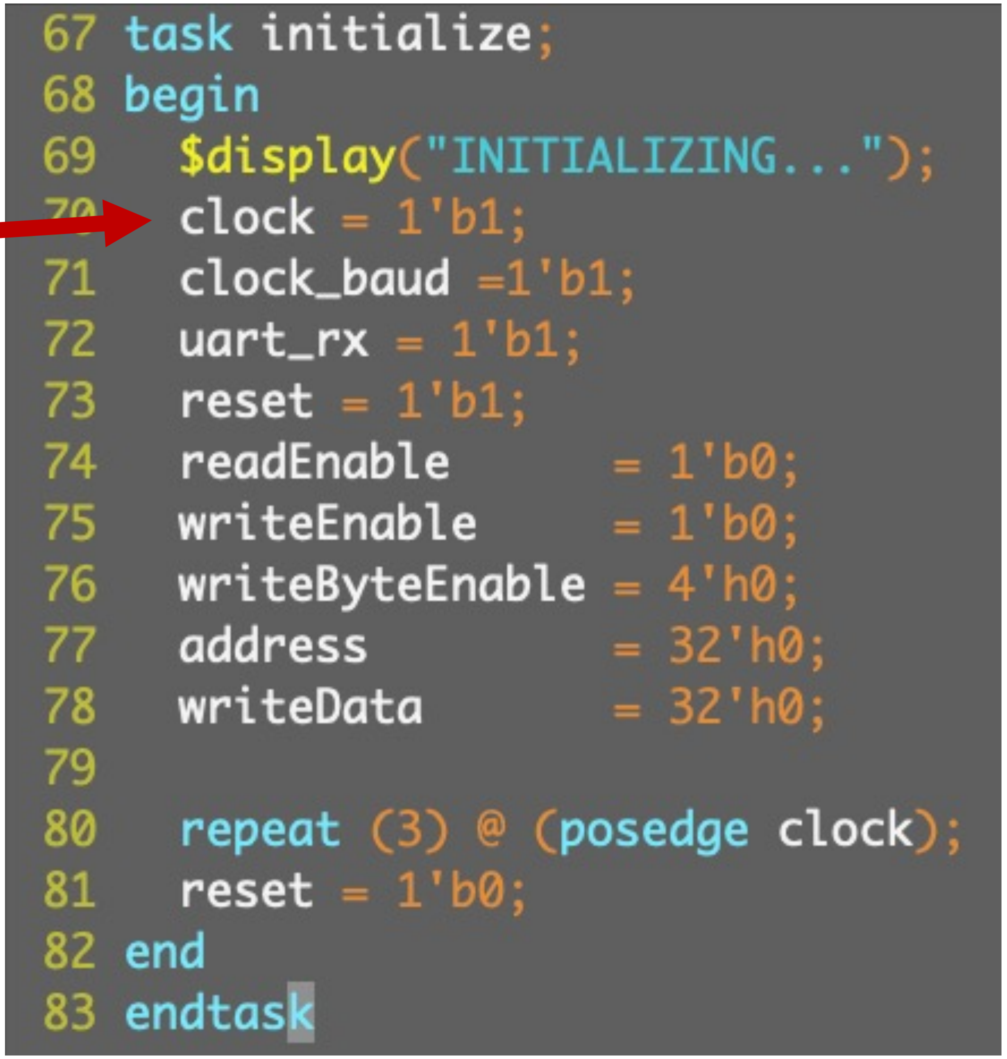
- Initialize task
  - $display writes to console



```
task initialize;
begin
   $display("INITIALIZING...");
   clock = 1'b1;
   clock_baud =1'b1;
   uart_rx = 1'b1;
   reset = 1'b1;
   readEnable      = 1'b0;
   writeEnable     = 1'b0;
   writeByteEnable = 4'h0;
   address         = 32'h0;
   writeData       = 32'h0;

   repeat (3) @ (posedge clock);
   reset = 1'b0;
end
endtask
```

Example Test Bench

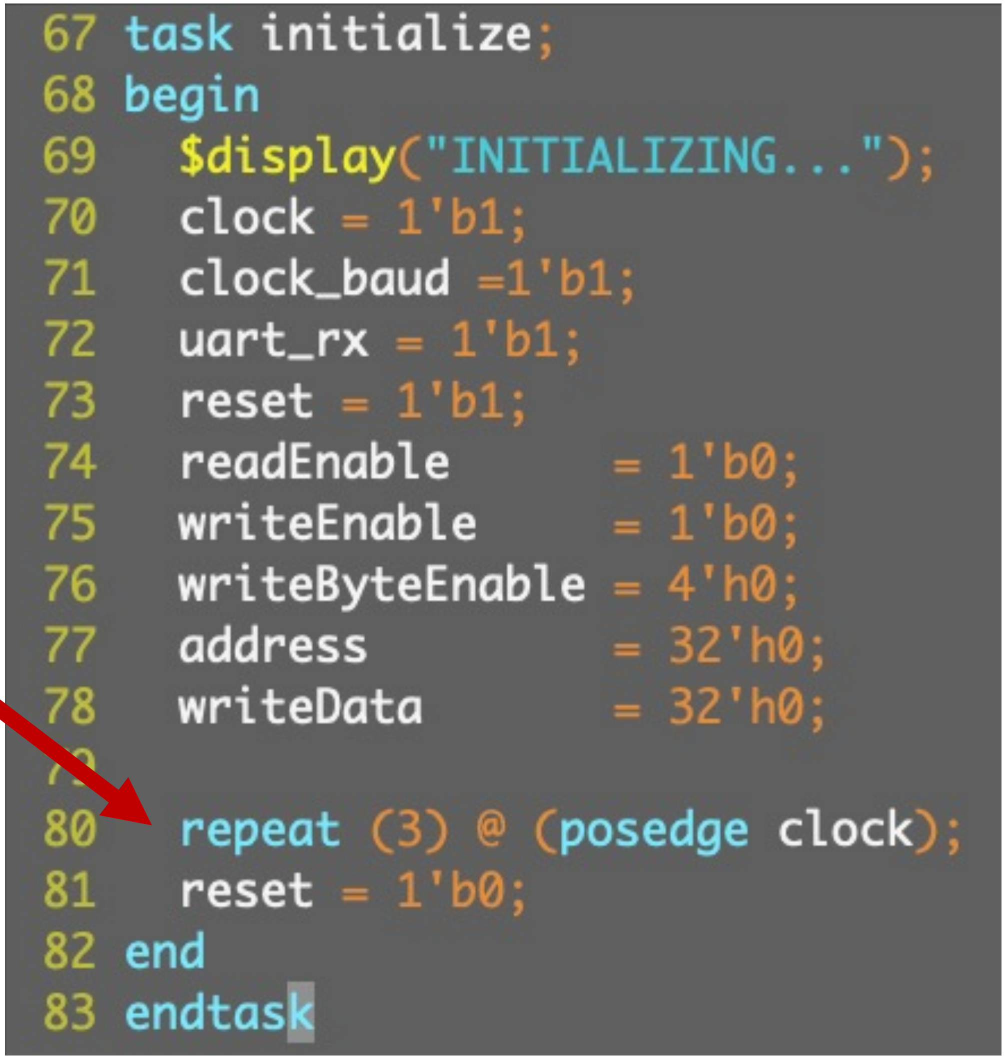
- Initialize task
  - $display writes to console
  - Set inputs to specific values
    - All set at the same simulation time



```
task initialize;
begin
  $display("INITIALIZING...");
  clock = 1'b1;
  clock_baud =1'b1;
  uart_rx = 1'b1;
  reset = 1'b1;
  readEnable        = 1'b0;
  writeEnable       = 1'b0;
  writeByteEnable = 4'h0;
  address           = 32'h0;
  writeData         = 32'h0;

  repeat (3) @ (posedge clock);
  reset = 1'b0;
end
endtask
```

Example Test Bench

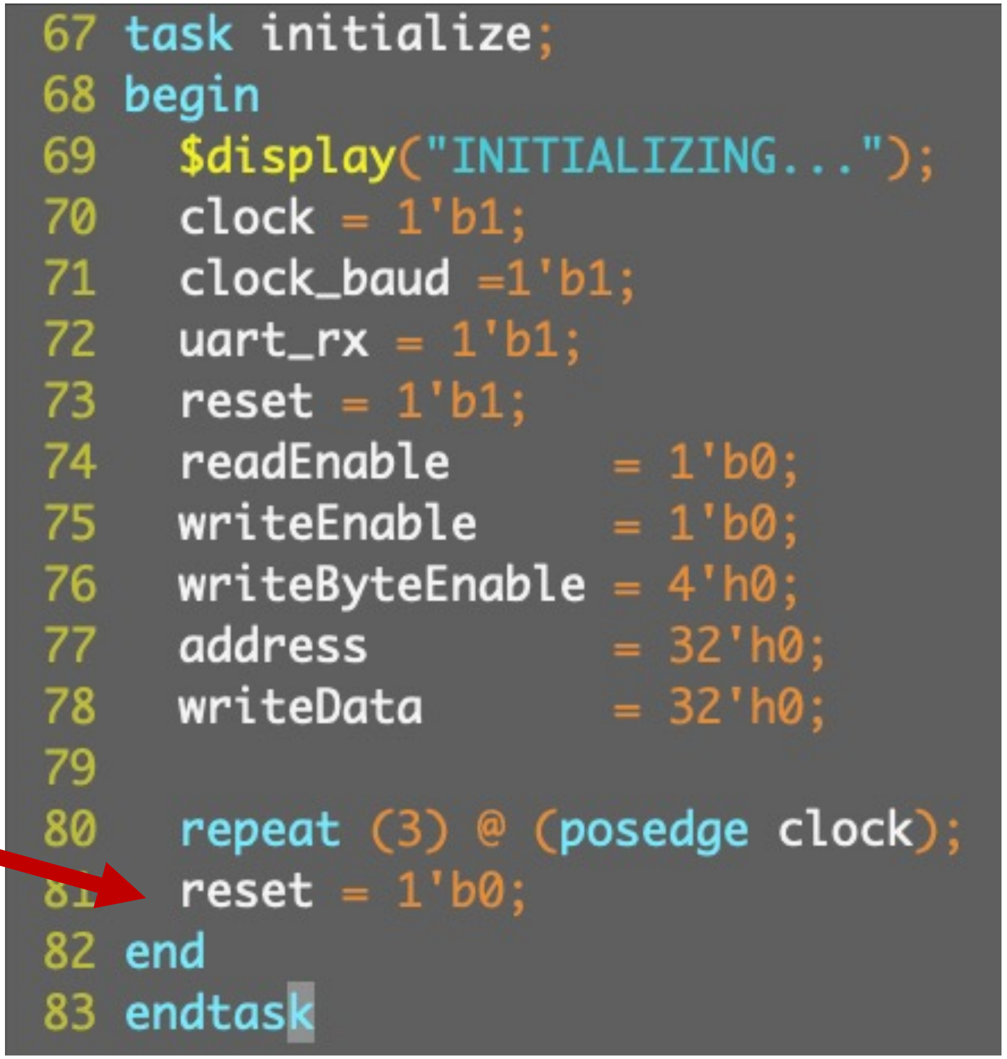
- Initialize task
  - $display writes to console
  - Set inputs to specific values
    - All set at the same simulation time
- Repeat statement waits for 3 clock cycles
  - Reset signal stays high for 3 clock cycles



```
task initialize;
begin
  $display("INITIALIZING...");
  clock = 1'b1;
  clock_baud =1'b1;
  uart_rx = 1'b1;
  reset = 1'b1;
  readEnable      = 1'b0;
  writeEnable     = 1'b0;
  writeByteEnable = 4'h0;
  address         = 32'h0;
  writeData       = 32'h0;

  repeat (3) @ (posedge clock);
  reset = 1'b0;
end
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



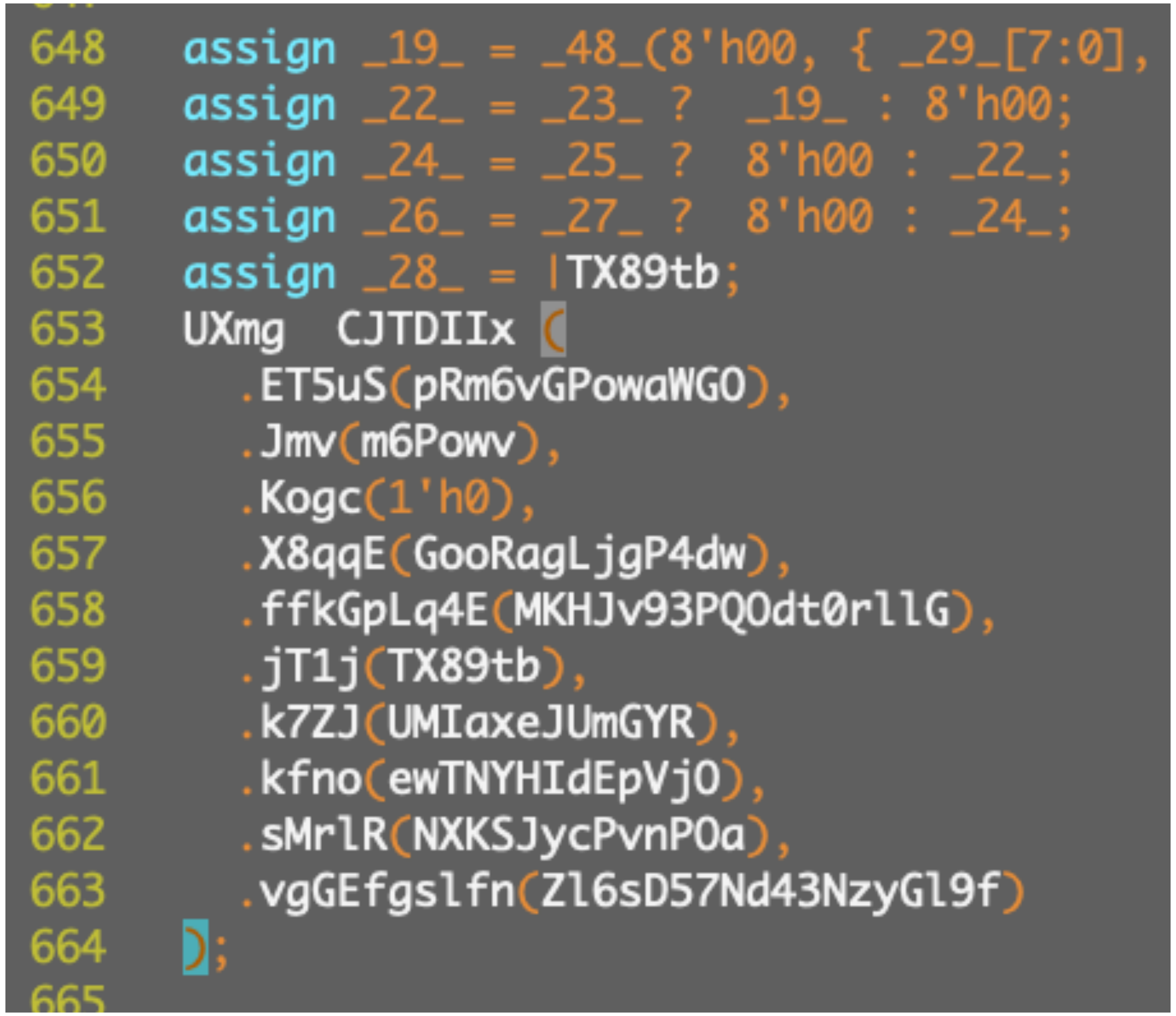
```
task initialize;
begin
  $display("INITIALIZING...");
  clock = 1'b1;
  clock_baud =1'b1;
  uart_rx = 1'b1;
  reset = 1'b1;
  readEnable      = 1'b0;
  writeEnable     = 1'b0;
  writeByteEnable = 4'h0;
  address         = 32'h0;
  writeData       = 32'h0;

  repeat (3) @ (posedge clock);
  reset = 1'b0;
end
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



```
assign _19_ = _48_(8'h00, { _29_[7:0],
assign _22_ = _23_ ?  _19_ : 8'h00;
assign _24_ = _25_ ?  8'h00 : _22_;
assign _26_ = _27_ ?  8'h00 : _24_;
assign _28_ = |TX89tb;
UXmg  CJTDIIx (
    .ET5uS(pRm6vGPowaWGO),
    .Jmv(m6Powv),
    .Kogc(1'h0),
    .X8qqE(GooRagLjgP4dw),
    .ffkGpLq4E(MKHJv93PQOdt0rllG),
    .jT1j(TX89tb),
    .k7ZJ(UMIaxeJUmGYR),
    .kfno(ewTNYHIdEpVjO),
    .sMrlR(NXKSJycPvnPOa),
    .vgGEfgslfn(Zl6sD57Nd43NzyGl9f)
);
```

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