Technische Universiteit
Eindhoven
University of Technology

Computation 2
Verilog
Syntax

2018-2019

Where innovation starts

Modules

Creating a module

-
  ```
  module module_name #(
  parameter integer
  parameter1_name =
  parameter1_default_value
  )
  (
  input variable1,
  input [3:0] variable2,
  output wire variable3,
  output reg [3:0] variable4
  );

  parameter2_name =
  parameter2_default_value

  endmodule
  ```

- **parameter1 is a variable for the module which can be set externally.**
- **variable1 is an 1 bit input**
- **variable2 is a 4 bits input**
- **variable3 is an one bit output wire**
- **variable4 is a 4 bits output register**

- **parameter2 is a variable which can only be used within the module**

Instantiate a module

```
module #(
.parameter_external1(value1),
.parameter_external2(value2)
) inst_name
(
.input_external(variable1),
.output_external(variable2)
);
```

- module is the name of the module you want to use
- inst_name is the local unique name
- parameter_external is the parameter name of the module that will be included
- input_external is the external input port
- output_external is the external output port
- variable1 is a **wire or a reg** that will be input to the module
- variable 2 is a **wire** that will be the output of the module

Using a module in a for loop

- genvar i;
- Generate
- for(i = 0; i<**N**; i = i+1)
- begin : *blockname*
- *logic*
- end
- endgenerate

- Makes **N** modules
- blockname is the name of the begin end statements of the for loop
- logic is the place where you will tell how to connect a single instance to the wires.

Technische Universiteit
Eindhoven
University of Technology

Constants

Binary

• 1’b0	• Is an one bit constant with value zero
• 3’b101	• Is a three bit constant with value five
• 3’b10	• Is a three bit constant with value two

decimal

• 1’d0	• Is an one bit constant with value zero
• 3’d5	• Is a three bit constant with value five
• 3’d2	• Is a three bit constant with value two

Hexidecimal

- 4’hA
- 2’hE
- 8’HFF

- Is a four bit constant with value ten
- Is a two bit constant with value two
- Is an eight bit constant with value 255

Technische Universiteit
Eindhoven
University of Technology

Different sizes wire and registers

Separate wires / registers-1

- reg [4:0] memory1
- reg [6:0] memory2
- wire [8:0] bus

- always @(posedge clk)
- begin
  memory1 <= memory2[4:0];
  memory2 <= bus[7:1];
  end

- memory1 gets the value of the lowest five bits of memory2

- memory2 gets the value of the middle seven bits of bus

Separate wires / registers-2

- reg [4:0] memory1
- reg [6:0] memory2
- wire [2:0] smallbus

- always @(posedge clk)
- begin
  memory2[4:0] <= memory1;
  memory1[3:1] <= smallbus;
  end

- The lowest five bits of memory2 gets the value of memory1

- The middle three bits of memory1 gets the value of smallbus

Concatenate wires / registers

- wire [4:0] bus
- reg [6:0] memory1
- reg [11:0] memory2

- always @(posedge clk)
- Begin
  memory1 <= {2'b0,bus};
  memory2 <= {bus,memory1};

  end

- The upper two bits of memory1 gets the value of zero, the lower five bits of memory1 gets the value of bus

- The upper five bits of memory2 gets the value of bus and the lower seven bits of memory2 gets the value of memory1

Testbenches

General overview

- Testbenches in this course don’t have any inputs or outputs.
- Using an always block to generate a clock.
- Initial block to run you own code in this course.
- Wait statements to push values with time delay to your module.
- Declaring registers with a initial value allowed

- Order of statements in a testbench:
  - AXI4-lite bus reset
  - Module reset
  - Input to test your module

Generating a clock

- reg clk = 0;

- always begin
  #10 clk = ~clk;
- end

- This will generate a clock with a period of 20 ns.
- Every 10 ns the clock will be inversed

Initial block

- reg [31:0] temp1 = 1;
- reg [31:0] temp2 = 2;
- wire [31:0] output;
- adder #() adder_inst (.a(temp1),.b(temp2),.c(output));

- Initial begin
  #5
  temp1 <= 32‘b0;
  #20
  temp2 <= 32’b1;
  #20
  temp1 <= 32’b10;
  end

- Initial blocks runs only once
- Value assignment at moment of creating are allowed
- Using clock of previous slide
- Make the statements out of sync with the posedge clock using #5
- Waiting between statements to see different outputs at output. #20 is waiting till the next clock cycle since period of clock is 20

Using AXIWrite

- axi_read*(addr*, *data);*

- **Example:**
- axi_write*(32*'d4,32'd20*);*

- Uses the AXI bus to read a slave register
- *Addr* is the address of the desired slave register.
- *Data* is the value that has to be transferred to the slave register
- *slv_reg0 has address 4'd0*
- *slv_reg1 has address 4'd4*
- *slv_reg2 has address 4'd8*
- *slv_reg3 has address 4'd12*

- The example will write the value 20 to slv_reg1.

Using AXIRead

- axi_read*(addr*, *output_reg);*
- **Example:**
- reg [31:0] output_data;
- axi_read*(32*'d4, output_data*);*

- Uses the AXI bus to read a slave register
- *Addr* is the address of the desired slave register.
- *slv_reg0 has address 4'd0*
- *slv_reg1 has address 4'd4*
- *slv_reg2 has address 4'd8*
- *slv_reg3 has address 4'd12*
- The example will read the data from slv_reg1 and store the output in output_data

Wave-window

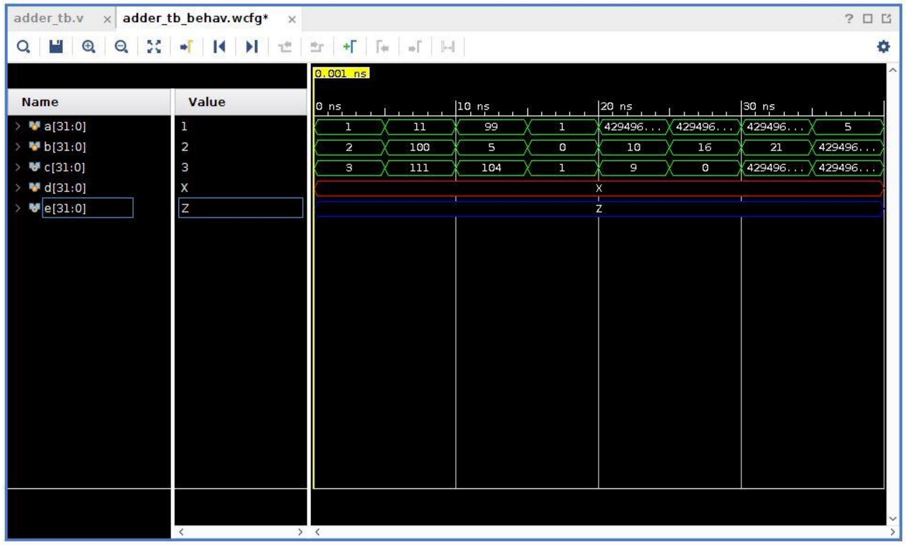
Technische Universiteit **Eindhoven** University of Technology

Where innovation starts

Technische Universiteit **Eindhoven** University of Technology

General info wave-window

- In the wave window you can see the values of the wires / busses as a function of time
- Use the wave window to debug your modules
- Check at which moment a signal rises during a clock cycle. Most of the time a signal is only processed at the rising edge of the clock. A change in the input for a short time (<time_clk_cycle) is not processed in a module most of the time.
- A value X in the wave window stands for value unknown
- A value Z in the wave window stands for high impedance (not connected)



Technische Universiteit
Eindhoven
University of Technology

Basic understanding

Combinatorial vs sequential logic 1

- Combinatiorial:
  - Runs instantly
  - assign statements
  - No clock
  - Wires
- Sequential:
  - Clocked
  - Registers
  - Always and initial blocks

Technische Universiteit Eindhoven University of Technology

Combinatorial vs sequential logic 2

- Error:

- concurrent assignment to a non-net count is not permitted

- Explanation:

- You are trying to write a value to a register with combinational logic. For example “assign a = b” where a is a register and b is a wire. You need to use an always block in which you set the value of a to b.
- You will also get this if you use a module in an always block. You can solve this by placing the module outside the always block. And connecting a wire to the outputs. You can than use a clocked always block to write the value of the wire to a register.

Non Synthesizable code

- initial blocks
- reg a = 1’b1;
- always @(posedge a, posedge b, posedge c, …)
- Edge triggering on a non clock signal or multiple signals.

- Not all code that works in the simulation will actually synthesize
- These examples work in simulation but are impossible to implement on a fpga.
- You can use the reset wire to give your registers a initial value.

Synthesizable code

```
always @(posedge clk) begin
    if(reset == 1'b1) begin
        output <= 1'b0;
    end else begin
        output <= next_output;
    end
end

always @(*) begin
    …
end
```

- Use non blocking assignment (<=) only in clocked always blocks and blocking (=) in other always blocks.
- Latch the output of your logic. So for every output create two registers a ‘output’ and ‘next_output’. At the positive edge of the clock set ‘output’ to equal ‘next_output’ or its initial value if the reset is low or high. Use always @(*) to set the value of ‘next_output’.
- The * has the meaning of every change of every signal.