Digital Design with the Verilog HDL
Chapter 5 Behavioral Model - part 1

Binh Tran-Thanh

Department of Computer Engineering
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology

February 28, 2022

Behavioral Verilog

- Use procedural blocks: `initial`, `always`
- These blocks contain series of statements
  - Abstract – works *somewhat* like software
  - Be careful to still remember it's hardware!
- Parallel operation **across** blocks
  - All blocks in a module operate simultaneously
- Sequential **or** parallel operation **within** blocks
  - Depends on the way the block is written
  - Discuss this in a later lecture
- LHS of assignments are **variables**(reg)

Types of Blocks

initial

- Behavioral block operates ONCE
- Starts at time 0 (beginning of operation)
- Useful for test benches
- Can sometimes provide initialization of memories/FFs
  - Often better to use ”reset” signal
- Inappropriate for combinational logic
- Usually cannot be synthesized

always

- Behavioral block operates CONTINUOUSLY
- Can use a trigger list to control operation; `@(a, b, c)`

initial vs. always

initial

```
reg[7:0] v1, v2, v3, v4;
initial begin
     v1 = 1;
  #2 v2 = v1 + 1;
     v3 = v2 + 1;
  #2 v4 = v3 + 1;
     v1 = v4 + 1;
  #2 v2 = v1 + 1;
     v3 = v2 + 1;
end
```

always

```
reg[7:0] v1, v2, v3, v4;
always begin
     v1 = 1;
  #2 v2 = v1 + 1;
     v3 = v2 + 1;
  #2 v4 = v3 + 1;
     v1 = v4 + 1;
  #2 v2 = v1 + 1;
     v3 = v2 + 1;
end
```

What values does each block produce?

initial Blocks

```
`timescale 1ns /1ns
module t_full_adder;
    reg [3:0] stim;
    wire s, c;

    // instantiate UUT
    full_adder(sum, carry, stim[2], stim[1], stim[0]);

    // monitor statement is special - only needs to be made once,
    initial $monitor("%t: s=%b c=%b stim=%b", $time, s, c, stim[2:0]);

    // tell our simulation when to stop
    initial #50 $stop;

    initial begin  // stimulus generation
        for (stim = 4'd0; stim < 4'd8; stim = stim + 1) begin
                #5;
        end
    end
endmodule
```

all initial blocks start at time 0

single-statement block

*multi-statement block enclosed by **begin** and **end***

always Blocks

- Operates continuously or on a trigger list
- Can be used with **initial** blocks
- Cannot ”nest” initial or always blocks
- Useful example of continuous always block:

```
reg clock;
initial clock = 1’b0;
always #10 clock = ~clock;
```

- Clock generator goes in the **testbench**

always blocks with trigger lists

- Conditionally "execute" inside of **always** block
  - Always block continuously operating
  - If trigger list present, continuously checking triggers
  - Any change on trigger (sensitivity) list, triggers block

```
always @(a, b, c) begin
   ...
end
```

- Sounds like software! It isn't!
  - This is how the *simulator* treats it
  - The hardware has the same resulting operation, but ...
    - See examples in later slides to see what is **actually** created

Trigger Lists

- Uses "event control operator" @
- When net or variable in trigger list changes, **always** block is triggered

```
always @(a, b, c) begin
  a1 = a & b;
  a2 = b & c;
  a3 = a & c;
  carry = a1 | a2 | a3;
end

always @(in1, in0, sel) begin
  if(sel== 1'b0)
    out = in0;
  else
    out = in1;
end
```

```
always @(state, in ) begin
  if(in == 1'b0) begin
    if(state != 2'b11)
      nextstate = state + 1;
    else
      nextstate = 2'b00;
  end
  else
    nextstate = state;
end
```

Event or

- Original way to specify trigger list `always @ (X1 or X2 or X3)`
- In Verilog 2001 can use ,instead of or `always @ (X1, X2, X3)`
- Verilog 2001 also has * for combinational only `always @(*)`
  - Shortcut that includes all nets/variables used on RHS in statements in the block
  - Also includes variable used in if statements; if (x)
- You may be asked to specify input s to trigger list without *

Your turn: comparison module

```
module compare_4bit_behave(
            output reg A_lt_B, A_gt_B, A_eq_B,
            input [3:0] A, B);

   always @(A, B) begin
      .....................................................
      .....................................................
      .....................................................
      .....................................................
      .....................................................
      .....................................................
      .....................................................
      .....................................................
   end

endmodule
```

Edge Triggering

- A negedge is on the transitions
  - 1 $\rightarrow$ x, z, 0
  - x, z $\rightarrow$ 0
- A posedge is on the transitions
  - 0 $\rightarrow$ x, z, 1
  - x, z $\rightarrow$ 1
- Used for clocked (synchronous) logic

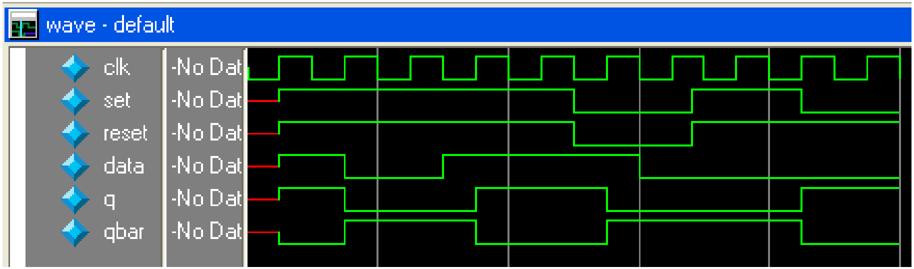
```
always @( posedge clk)
  register <= register_input ;
```

Example: DFF

Remember LHS in `always` block is always declared as reg.

```
module dff(output reg q, input d, input clk);
  always @(posedge clk) begin
    q <= d;
  end
endmodule

module dff_reset(output reg q, input d, clk, reset);
  always @(posedge clk, negedge reset) begin
    if(!reset) q <= d;
    else q <= 0;
  end
endmodule
```

DFF with Set Control



```
module dff(output q, qbar, input reset, set, data, clk);
  reg ...;
  always @(posedge clk) begin
    ...
  end
endmodule
```

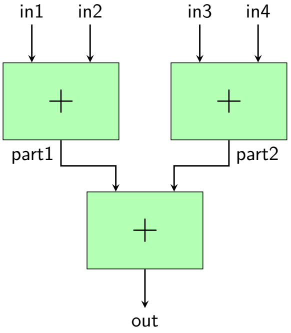
Procedural Assignments

- Used within a behavioral block (initial, always)
- Types
  - `= // blocking assignment`
  - `<= // non-blocking assignment`
- Assignments to variables:
  - reg
  - integer
  - real
  - realtime
  - time

Blocking Assignments

- "Evaluated" **sequentially**
- Works a lot like software (danger!)
- Used for **combinational** logic

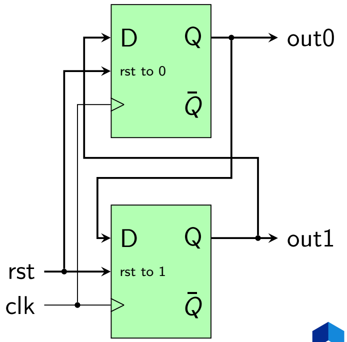
```
module addtree(output reg[9:0] out,
  input [7:0] in1, in2, in3, in4);
  reg[8:0] part1, part2;

  always @(in1, in2, in3, in4)
  begin
    part1 = in1 + in2;
    part2 = in3 + in4;
    out = part1 + part2;
  end
endmodule
```



Non-Blocking Assignments

- "Updated" **simultaneously** if no delays given
- Used for **sequential** logic

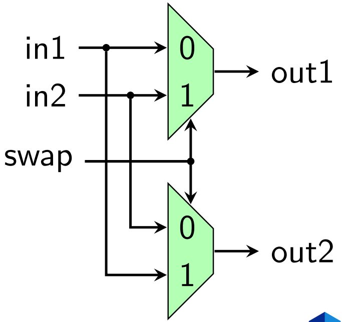
```
module swap(output reg out0, out1,
            input rst, clk);
   always @(posedge clk) begin
      if(rst) begin
         out0 <= 1'b0;
         out1 <= 1'b1;
      end
      else begin
         out0 <= out1;
         out1 <= out0;
      end
   end
endmodule
```



Swapping

- In blocking, need a "temp" variable

```
module swap(output reg out0, out1,
            input in0, in1, swap);
   reg temp;

   always @(*) begin
      out0 = in0;
      out1 = in1;
      if(swap) begin
         temp = out0;
         out0 = out1;
         out1 = temp;
      end
   end
endmodule
```



Blocking & Non-Blocking Example

```
reg[7:0] A, B, C, D;

always @(posedge clk) begin
   A = B + C;
   B = A + D;
   C = A + B;
   D = B + D;
end
```

```
reg[7:0] A, B, C, D;

always @(posedge clk) begin
   A <= B + C;
   B <= A + D;
   C <= A + B;
   D <= B + D;
end
```

- Assume initially that A = 1, B = 2, C = 3, and D = 4
- Note: shouldn't use blocking with sequential!

Correcting The Example

```
reg[7:0] A, B, C, D;
reg[7:0] newA, newB, newC, newD;

always @(posedge clk) begin
  A <= newA; B <= newB;
  C <= newC; D <= newD;
end

always @(*) begin
  newA = B + C;
  newB = newA + D;
  newC = newA + newB;
  newD = newB + D;
end
```

```
reg[7:0] A, B, C, D;

always @(posedge clk)
    begin
  A <= B + C;
  B <= B + C + D;
  C <= B + C + B + C +D;
  D <= B + C + D + D;
end
```

Why Not '=' In Sequential?

Yes, it can "work", but...

- <= models pipeline stages better
- = can cause problems if multiple always blocks
- Order of statements is important with =

Use the style guidelines given!

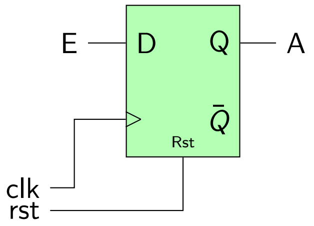
- <= for sequential block
- = for combinational block
- Don't mix in same block!

Shift Register: Blocking

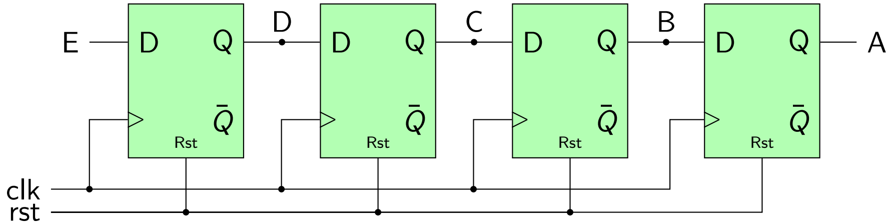
```
module shiftreg(A, E, clk, rst);
  output A;
  input E, clk, rst;
  reg A, B, C, D;
  always @ (posedge clk or posedge rst) begin
    if(rst) begin A = 0; B = 0; C = 0; D = 0; end
    else begin A = B; B = C; C = D; D = E;
    end// option 1

    //else begin D = E; C = D; B = C; A = B;
    //end// option 2
  end
endmodule
```

- What do we get with each option?

Shift Register: Blocking



The order is matters!



Combinational vs. Sequential

Combinational

- Not edge-triggered
- All ”input s” (RHS nets/variables) are triggers
- Does not depend on clock

Sequential

- Edge-triggered by clock signal
- Only clock (and possibly reset) appear in trigger list
- Can include combinational logic that feeds a FF or register