ECE 351
Verilog and FPGA Design

Week 6_1: **Questions about HW #2?**
SystemVerilog for looping (wrap-up)
Review for midterm exam
Modeling combinational logic by example

Roy Kravitz
Electrical and Computer Engineering Department
Maseeh College of Engineering and Computer Science

Questions about Homework #2?

Homework #2: ..\..\assignments\hw2\ece351sp21_hw2_release\docs\ece351sp21_hw2.pdf

SystemVerilog `for` loops (wrap-up)

Source material drawn from:

- Roy's ECE 351 and ECE 571 lecture material
- *RTL Modeling with SystemVerilog* by Stuart Sutherland
- *Logic Design and Verification Using SystemVerilog* by Donald Thomas

Review: **`for`** loops

- Syntax:

  for (*initial_assignment*; *condition; step_assignment*)
  procedural_statement

- Repeats the execution of the procedural statement a certain number of times
  - *initial_assignment* is the starting value of the loop index
  - *condition* specifies when loop execution must stop; statement(s) in the loop are executed as long as the condition is true
  - *step_assignment* specifies the assignment to modify (typically to increment or decrement the step count
- Ex:

```
integer k;
for (k = 0; k < MAX_RANGE; k = k + 1) begin
  if (hold_data[k] == 0)
    // do something
end
```


Portland State UNIVERSITY

Review: **`for`** loops (cont'd)

SystemVerilog permits declaration of the loop variable in the **`for`** loop

```
module chip (...); // SystemVerilog style loops
  ...
  always_ff @(posedge clock) begin
    for (bit [4:0] i = 0; i <= 15; i++)
         ...
  end

  always_ff @(posedge clock) begin
    for (int i = 1; i <= 1024; i += 1)
         ...
  end
endmodule
```

Scope of *i* is local to the for loops

When declared in this way, the loop variable is an automatic variable and

- Cannot be referenced hierarchically
- Has no existence (or value) outside the loop

Review: Synthesizing **for** loops

- Synthesis compilers "unroll" the loop
  - Statement or `begin..end` group is replicated the number of times that the loop iterates
  - For synthesis the number of loop iterations must be a fixed number of times (called a static loop)
- Static loop (also called a data-independent loop) -> number of iterations can be determined w/o having to know the value of any nets or variables
- Data-dependent loops cannot be synthesized because synthesis compiler cannot determine the number of times to replicate the logic inside the for loop
- Code synthesizable for loops w/ 0 delay -> result is combinational logic

More Looping Statements

Source material drawn from:

- Roy's ECE 351 and ECE 571 lecture material
- *RTL Modeling with SystemVerilog* by Stuart Sutherland
- *Logic Design and Verification Using SystemVerilog* by Donald Thomas

repeat loops

- Syntax:

  repeat (*loop_count*)
  procedural_statement

- Executes the procedural statement *loop_count* times
  - Procedural statement could be a begin…end block of statements
- Ex:

```
repeat (count)
  sum = sum + 10;

repeat (shift_by) begin
  wdog_reg = wdog_reg << 1;
end

accum = repeat(load_count) @posedge(clk_rtc)  //event ctrl
  accum + 1;
```

Synthesizing `repeat` loops

- Synthesizable if the number of times the loop will iterate is fixed and not dependent on value of something that can change
- A static zero-delay repeat loop will synthesize to combinational logic
  - If output of combinational logic is registered in flip-flops the total propagation delay of the combinational logic must be less than a clock cyle

Synthesizing repeat loops (cont'd)

```
module exponential
#(parameter E = 3,    // power exponent
  parameter N = 4,    // input bus size
  parameter M = N*2   // output bus size
)
(input  logic         clk,
 input  logic [N-1:0] d,
 output logic [M-1:0] q
);

  always_ff @(posedge clk) begin: power_loop
    logic [M-1:0] q_temp; // temp variable for inside the loop
    if (E == 0)
      q <= 1;  // do to power of 0 is a decimal 1
    else begin
      q_temp = d;
      repeat (E-1) begin
        q_temp = q_temp * d;
      end
      q <= q_temp;
    end
  end: power_loop
endmodule: exponential
```


Source: Sutherland Ex. 6.9

while loops

- Syntax:

  while (*condition*)
    procedural_statement

- Executes the procedural statement or `begin..end` block of statements *until the specified condition becomes false*
  - If the condition is false to begin with the procedural statement is never executed
  - If the condition is an `x` or a `z` it is treated as false
- Ex:

```
while (shift_by > 0) begin
  acc = acc << 1;
  shift_by = shift_by - 1;
end
```

do…while loops

As with C, test is at end of loop so loop always executes at least once

```
always_comb begin
  do begin
    done = 0;
    OutOfBound = 0;
    out = mem[addr];
    if (addr < 128 || addr > 255) begin
      OutOfBound = 1;
      out = mem[128];
    end
    else if (addr == 128) done = 1;
    addr -= 1;
  end
  while (addr >= 128 && addr <= 255);
end
```

forever Loops

- Syntax:
  forever
    procedural_statement
- Continuously execute the procedural statement
  - Could be a begin…end block of statements
- The only way out of the loop is with a disable statement
- Some form of timing controls must be used otherwise the forever loop will loop forever in zero delay
- Ex:

```
initial begin
  clk1hz = 0;

  #5 forever
    #10 clk1hz = ~clk1hz;
end
```

Review for midterm exam

Review for midterm exam

- Exam will be open book, open notes, open internet
  - You will need internet and D2L access to complete the exam
  - Unless the problem specifically says so we are not going to deduct points for syntax errors (missing ; etc.) but we do expect your code to be indented and formatted so that we can read it
  - I do not recommend trying to simulate your answers – it will take too long, and you have a strict deadline…however using a context-sensitive text editor like Notepad++ for the programming questions is allowed and encouraged.
- The exam will be:
  - ~60% - T/F, multiple choice, short answer
  - ~40% - SystemVerilog programming
- All students in the class who have not made special arrangements with me are expected to take the exam on Thursday from 2:00 PM – 4:30 PM (extra 30 minutes, but still in time to make a 4:40 PM class)

Review for midterm exam (cont'd)

- Fair game for exam:
  - SystemVerilog language rules (literals, vectors, part and part selects, net and var types, etc.)
  - Modules, ports, hierarchy, port by name, port by position, instantiating modules, parameters, etc.)
  - User-defined types (`typedef`, `enum`, `struct`, `union`, etc.)
  - RTL expression operators (bitwise and reduction, logical, arithmetic, shifts, etc.)
  - Continuous assigns and procedural blocks (always and its variants, initial, blocking and non-blocking)
  - Decision statements (case variants, if..else)
  - `for` loops and loop unrolling
- You can expect to write SystemVerilog code

Test process (cont'd)

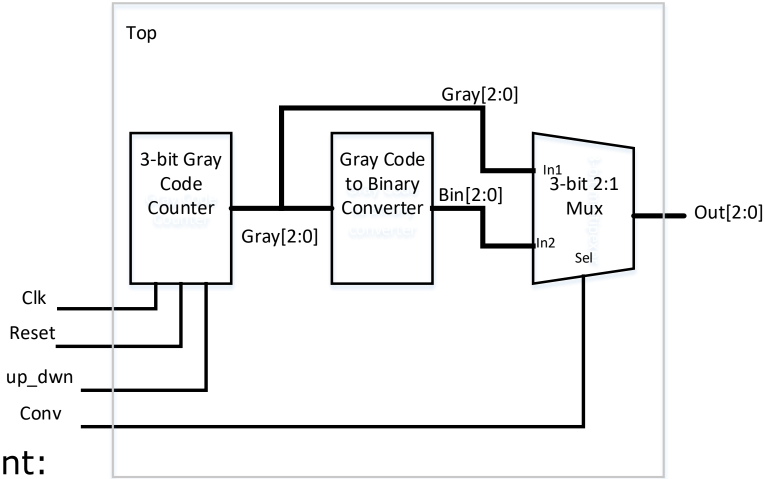
- You will provide your answers to a problem in the associated .txt or.sv file (ex: prob1.txt) using a text editor
- When you have completed the exam, please create a single .zip or .rar file with all of your answers and upload it to your D2L midterm exam dropbox
  - IMPORTANT: You will be able to submit more than once until the dropbox closes…**no submissions after the dropbox closes!**
  - We will keep/grade your latest submission so submit the full package
  - Make sure that you submit what you want us to grade – there are no do-overs
- The dropbox will close **10 minutes after the exam "ends"** so allow time to bundle your answers and upload them to D2L before then (7:40 PM for this exam)

Modeling combinational Logic (by example)

Source material drawn from:

- Roy’s ECE 351 and ECE 571 lecture material
- *RTL Modeling with SystemVerilog* by Stuart Sutherland
- *Logic Design and Verification Using SystemVerilog* by Donald Thomas

Gray Code counter/mux design example



Problem statement:

Implement a gray-code counter w/ a selectable binary and gray code output

When Conv == 0 you want Out[2:0] to be the unconverted Gray Code from the counter. When Conv == 1 you want the Out[2:0] to be the Binary code from the converter. The 3-bit Up/Down Gray Code counter will be implemented as a FSM.

Design Flow

- Step 1: Create a Gray Code -> Binary converter module
- Step 2: Create an up/down Gray code counter as an FSM
- Step 3: Create a top-level module for the converter
  - Instantiate Gray code up/down counter
  - Instantiate Gray Code -> Binary converter module
  - Write the code for the 3-bit 2:1 mux
- Step 4: Write a testbench for the design
- Step 5: Simulate the design

Example: ..\examples\gray_code_cntr_mux\hdl

Gray Code -> Binary

Gray Code			Binary Code		
Gray[2]	Gray[1]	Gray[0]	Bin[2]	Bin[1]	Bin[0]
0	0	0	0	0	0
0	0	1	0	0	1
1	0	1	0	1	0
1	0	0	0	1	1
1	1	0	1	0	0
1	1	1	1	0	1
0	1	1	1	1	0
0	1	0	1	1	1

Next Time

- Zoom midterm exam scheduled for Thu, 06-May from 2:00 PM – 4:30 PM (+10 min. to upload your solution)
- Topics:
  - Modeling sequential logic
- You should:
  - Read Sutherland Ch 7 and Ch 8
- Homework, projects and quizzes
  - Homework #2 has been released. Due to D2L by 10:00 PM on Mon, 10-May