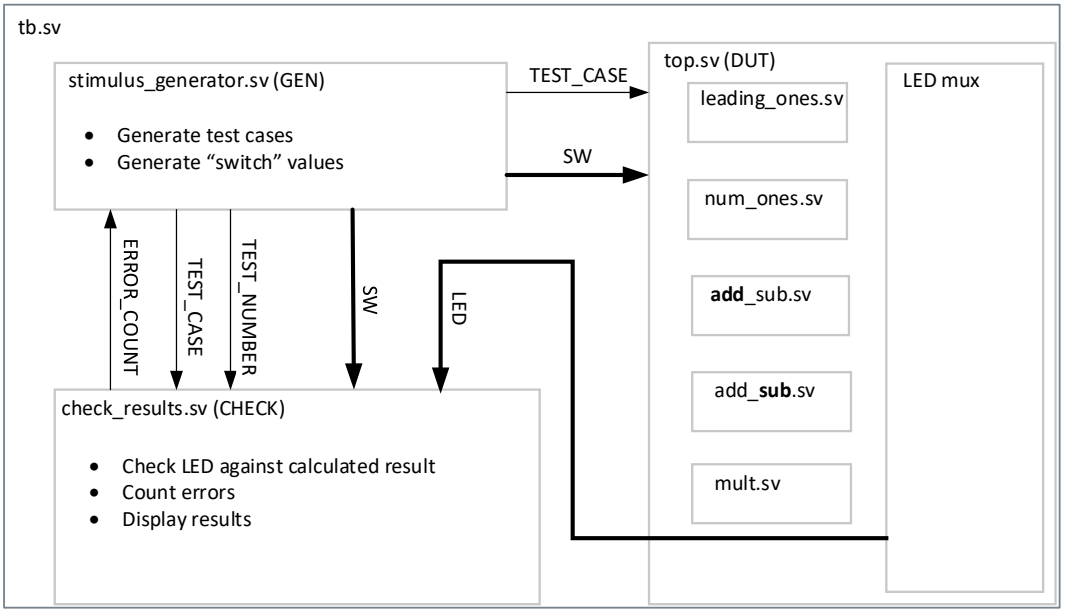
ECE 351
Verilog and FPGA Design

Week 5_2: **Review Roy's solution to Exercise #2**
RTL programming statements (wrap-up)
Review Roy's Homework #1 solution

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

Roy's solution to Exercise #2

Exercise 2 – Combinational Logic
(RTL programming statements)



RTL Files

- definitions_pkg.sv: common enums, typedefs, etc.
- tb.sv: testbench
  - stimulus_generator.sv: generates test cases
  - check_results.sv: determines whether test passes
  - top.sv: top-level model of DUT
    - leading_ones.sv: finds position of most-significant one
    - num_ones.sv: counts number of ones
    - add_sub.sv (x2): signed adder/subtractor
    - mult.sv: signed multiplier

RTL Programming statements (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

SystemVerilog `for` loops

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

SystemVerilog looping statements

- Allow execution of a programming statement or begin..end group to be executed multiple times
- Looping statements:
  - for
  - repeat
  - while
  - do..while
  - foreach
  - forever

restrictions for synthesis

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
```

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

for loops (cont'd)

SystemVerilog: Multiple assignments are supported

```
for (int i=1, j=0; i*j < 128; i++, j+=3)
  ...
for (int i=1, byte j=0; i*j < 128; i++, j+=3)
  ...
```

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

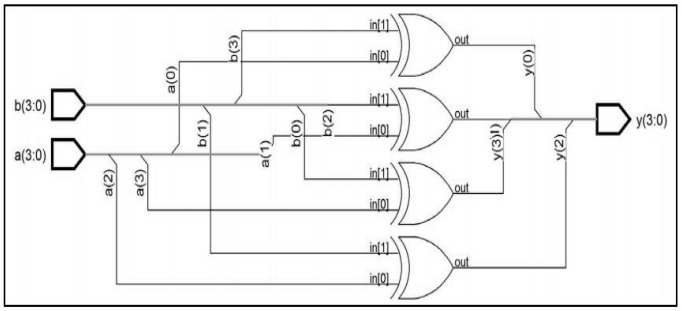
Static Loop

```
module bus_xor
#(parameter N = 4)              // bus size
(input  logic [N-1:0] a, b,     // scalable input size
 output logic [N-1:0] y         // scalable output size
);

  always_comb begin
    for (int i=0; i<N; i++) begin
      y[i] = a[i] ^ b[(N-1)-i]; // XOR a and reverse order of b
    end
  end

endmodule: bus_xor
```

```
always_comb begin
  y[0] = a[0] ^ b[3-0];
  y[1] = a[1] ^ b[3-1];
  y[2] = a[2] ^ b[3-2];
  y[3] = a[3] ^ b[3-3];
end
```



Technology independent schematic (no target ASIC or FPGA selected)

Source: Sutherland Fig. 6.7

Data-dependent Loop

```
always_comb begin
  // find lowest bit that is set in a 32-bit vector
  low_bit = '0;
  end_count = 32;
  for (int i=0; i<end_count; i++) begin
    if (data[i]) begin
      low_bit = i;
      end_count = i;  // cause loop to terminate early
    end
  end
end
```

Cannot be synthesized
`end_count` cannot be statically determined

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)

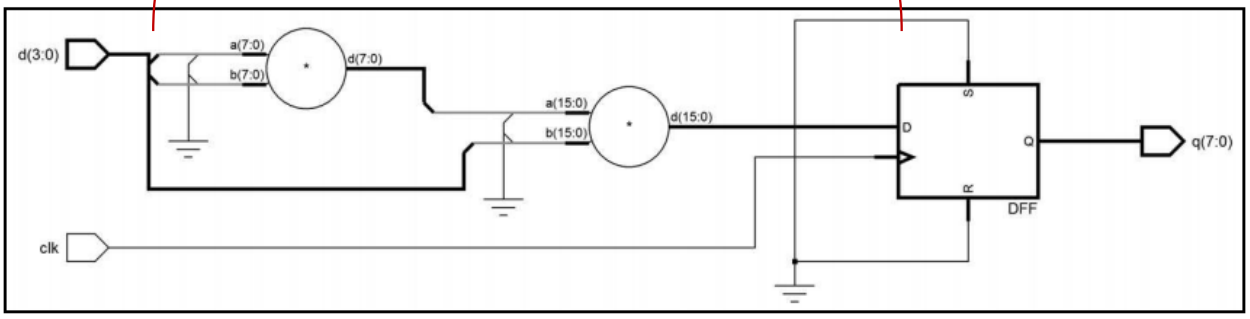
```
module exponential
#(parameter E = 3,     // power exponent
  parameter N = 4,     // input bus size
  parameter M = N*2    // output bus size
)
(input  logic          clk,
 input  logic [N-1:0] d,
 output logic [M-1:0] q
);

  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

Synthesizing `repeat` loops (cont'd)

< one clock cycle



Technology independent schematic (no target ASIC or FPGA selected)

Source: Sutherland Fig. 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
```

Loop control statements

SystemVerilog introduces C language **break**, **continue**, **return**

- Not necessary to have named **begin…end** blocks
- No **goto**
- Are synthesizable

```
// find first bit set within a range of bits
always_comb begin
  first_bit = 0;
  for (int i=0; i<=63; i=i+1) begin
    if (i < start_range) continue;
    if (i > end_range)   break;  // exit loop
    if ( data[i] ) begin
      first_bit = i;
      break;  // exit loop
    end
  end  // end of the loop
  ...  // process data based on first bit set
end
```

Roy's solution for Homework #1

Next Time

- Topics:
  - Modeling combinational logic
  - Functions and tasks
  - Review for midterm exam
- You should:
  - Read Sutherland Ch 6
- Homework, projects and quizzes
  - Homework #2 will be released tonight. Due date adjusted to 10-May by 10:00 PM.
  - Zoom midterm exam scheduled for Thu, 06-May from 2:00 PM – 4:00 PM but I'd like to extend the exam for 30 minutes
    - Poll on exam length (2:00 – 4:30 vs. 1:30 – 4:00 vs. no extension)