ECE 551

Digital Design And Synthesis

Fall'21

Random Testing
For Loops & Synthesis
Generate Statements
Use of X in Synthesis
Synthesis Pitfalls
Coding for Synthesis

1

Motivation for CRV (Constrained Random Verification)

- Directed functional testing requires you to think about all possible scenarios. You are a human *(therefore flawed)* you will miss some corner cases. Plus…does ...ring and implementing all possible corner c... ...incredibly boring and tedious?
- We could ge... ...m values for our test vectors. But there a... ...many possible cases, many of which are inva... *(or at least not of interest).*
- What we need is some way to constrain the random values to values of interest.
- System Verilog added amazing level of support for constrained random testing.

Directed tests find bugs you expect to be in the design, whereas random testing finds bugs you never anticipated.

2

2

Declaring a **rand** signal

- Unlike **logic** and **reg** which are 4-value logic (0,1,x,z), type **bit** has only 0,1. The type **bit** is typically used for random vectors.
- The keyword **rand** (or **randc**) modifies a signal declaration to enable randomization. The built in method (***randomize()***) can be applied to signals declared with this modifier.

rand bit [7:0] rand_byte;

- **randc** (cylical) ensures every possible random value of the signal is used before it repeats a number. **rand** does not have this constraint.
- Many forms of constraints can be placed on these random vectors. This occurs within the context of a **class**.

3

3

Constraining a **rand/randc** signal

```
class myPacket;

 rand bit [1:0] mode;
 randc bit [2:0] key;                  // cyclic (all allowed values used before repeating)

 constraint c_mode1 { mode < 3; }      // mode can be 00, 01, 10
 constraint c_key1 { key > 2; key <7; } // key in [3,6] set

endclass


module demo();

 myPacket pkt;          // declare packet of class myPacket

 initial begin
  pkt = new();          // class constructor

  assert (pkt.randomize());        // call the built in randomize method

  $display("mode = %d, key = %d",pkt.mode,pkt.key);
 end
endmodule
```

4

4

More Constraint Examples

```
rand bit [31:0] len, src, dst;
rand bit congestion_test;

constraint c_stim {
  len < 3000;
  len > 20;
  if (congestion_test) {
    dst inside { [32,128] };
    src = 32'h22;
  }
  else
    src inside {0, [2:10], [100:107] };
}
```

Examples take from:
SystemVerilog for Verification
Spear/Tumbush

Constraints of vectors can be related to the values of other vectors. Here **dst** and **src** have constraints that are dependent on the value of random bit **congestion_test**.

```
rand bit [7:0] low, med, high;

constraint relate {
  low < med;
  med < high;
}
```

Constraints can enforce relations between random vectors. Here we see that low must be less than med, which in turn must be less than high.

5

5

More Constraint Examples

```
class Packet;
  rand bit [31:0] pkt_len1, pkt_len2;
  constraint total_len {
    pckt_len1 + pckt_len2 = 256;
  }
endclass
```

Examples take from:
SystemVerilog for Verification
Spear/Tumbush

```
class BusOp;
  typedef enum {UART, A2D, STACK, MEM} addr_spc_t;
  rand addr_spc_t addr_spc;

  rand bit [31:0] addr;

  constraint c_addr {
    if (addr_spc == UART)
      addr inside {[32'hFFFFFF00:32'hFFFFFF3F]};
    else if (addr_spc == A2D)
      addr inside {[32'hFFFFFF80:32'hFFFFFFBF]};
    else if (addr_spc == STACK)
      addr inside {[32'h00000000:32'h0FFFFFFF]};
    else
      addr inside {[32'h10000000:32'h7FFFFFFF]};
  }

endclass
```

Here we see a **rand** variable of an enumerated type.

We also see a multiple if-else if-else construct used to create a relationship between two rand vectors.

6

6

System Verilog Support for Random Testing

- Can also create/control **dist**ributions.

```
class stim_t
  rand bit [1:0] func;
  rand bit src1sel,src2sel,cin;
  rand bit [15:0] instr,RF,mem;

  contraint RF_lim {
    RF dist {[16'h0000:16'h003F]:=99,[16'h0040:16'hFFFF]:=1};
  }
endclass

initial begin
  stim_t stim = new();              // class constructor

  integer x;

  for (x=0; x<10; x++) begin
    stim.randomize();               // built in method (generate set of stimulus)
    func = stim.func;               // assign stimulus vector
    …
  end
end
```

99% of all RF values will fall in the range 0x0000 to 0x003F and 1% will fall in remaining range.

7

7

Full Solution

```
module datapath_tb();

////////////////////////////
// Define stimulus vectors //
////////////////////////////
bit [1:0] func;
bit src1sel,src2sel,cin;
bit [15:0] instr,RF,mem;

wire [15:0] dst;

//////////////////////
// Instantiate DUT //
//////////////////////
datapath iDUT(.func(func), .src2sel(src2sel), .src1sel(src1sel),
              .instr(instr), .RF(RF), .mem(mem), .cin(cin), .dst(dst));

class stim_t;
  rand bit [1:0] func;
  rand bit src1sel,src2sel,cin;
  rand bit [15:0] instr,RF,mem;

  constraint RF_lim {
      RF dist { [16'h0001:16'h0003]:=99 };
  }
endclass
```

```
initial begin
  stim_t stim = new();

  integer x;

  for (x=0; x<10; x++) begin
    stim.randomize();
    func = stim.func;
    src1sel = stim.src1sel;
    src2sel = stim.src2sel;
    cin = stim.cin;
    instr = stim.instr;
    RF = stim.RF;
    mem = stim.mem;
    #10;
  end

  $stop();
end

endmodule
```

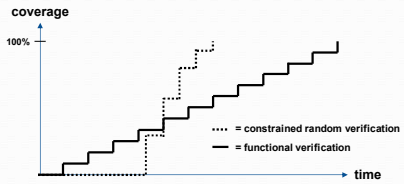
8

8

How Do You Self Check with Random Stimulus

大開銷

- There is a large overhead to using random stimulus. You need a parallel model of the DUT
- Build parallel high level model of the DUT in the testbench (scoreboard class)
- Perhaps a high level model of the architecture was built in systemC
- Can interface verilog simulator with C/C++ models through PLI or DPI
- Goes beyond the scope of this class



There is a large upfront cost to CRV. Some of this is developing the more complex framework, and a big chunk is developing some form of parallel model. In the end, however it results in quicker time to 100% coverage.

9

They Don't Give it Away

- We really only scratched the surface of what can be done with constraints
- Think about writing the solver that comes up with a solution to many interlinked constraints (i.e. writing the built in **randomize()** method?
  - It is wicked complicated.
  - They don't give it away with the free version of ModelSim
  - We do have it on the linux version (vsim)
- I do not expect you to use CRV on the project.
- I just wanted you to know about it so you don't look like a total dumb ass when entered industry
- There are some simpler random functions available in free version of ModelSim.
  - $**urandom();** // returns 32-bit unsigned integer
  - $**urandom_range**(100,2000); // returns 32-bit in range of 100 to 2000

10

For Loops & Synthesis

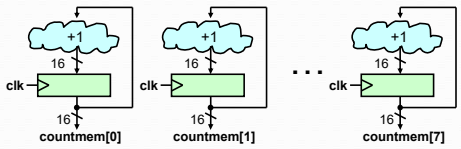
- Can a For Loop be synthesized?
  - Yes, if it is fixed length
  - The loop is *"unrolled"*

```
reg [15:0] countmem [0:7];
integer x;
always @(posedge clk) begin
      for (x = 0; x < 8; x = x + 1) begin
            countmem[x] <= countmem[x] +1;
      end
end
```

How do you think this code would synthesize?

11

For Loops & Synthesis



- These loops are *unrolled* when synthesized
  - That's why they must be fixed in length!
  - Loop index is type **integer** but it is not actually synthesized
  - Example creates eight 16-bit incrementers.
- What if loop upper limit was a parameter?

12

Unnecessary Calculations

- Expressions that are fixed in a **for loop** are replicated due to "loop unrolling."
- Solution: Move fixed (unchanging) expressions outside of all loops.

```
for (x = 0; x < 8; x = x + 1) begin
    for (y = 0; y < 8; y = y + 1) begin
        index = x*8 + y;
        value = (a + b)*c;
        mem[index] = value;
    end
end
```

This is just basic common sense, and applies to any language (in a programming language you would be wasting time, not hardware).

Yet this is a common mistake

- Which expression(s) should be moved?

13

13

More on Loops & Synthesis

- A loop is static (data-independent) if the number of iterations is fixed at compile-time
- Loop Types
  - Static without internal timing control
    - Combinational logic
  - Static with internal timing control (i.e. @(posedge clk))
    - ✓Sequential logic
  - Non-static without internal timing control
    - ✓Not synthesizable
  - Non-static with internal timing control (i.e. @(posedge clk))
    - ✓Sometimes synthesizable, Sequential logic

14

14

Static Loops w/o Internal Timing

- Combinational logic results from "loop unrolling"
- Example

```
always@(a) begin
    andval[0] = 1;
    for (i = 0; i < 4; i = i + 1)
        andval[i + 1] = andval[i] & a[i];
end
```

- What would this look like?
- For registered outputs:
  - Change sensitivity list 'a' with '**posedge** clk'

15

15

Static Loops with Internal Timing

- If a static loop contains an internal edge-sensitive event control expression, then activity distributed over multiple cycles of the clock

```
always begin
    for (i = 0; i < 4; i = i + 1)
        @(posedge clk) sum <= sum + i;
end
```

- What does this loop do?
- Does it synthesize?...Yes, but…

16

16

Non-Static Loops w/o Internal Timing

- Number of iterations is variable
  - Not known at compile time
- Can be simulated, but not synthesized!
- Essentially an iterative combinational circuit of data dependent size!

```
always@(a, n) begin
    andval[0] = 1;
    for (i = 0; i < n; i = i +1)
        andval[i + 1] = andval[i] & a[i];
end
```

- What if n is a parameter?

17

17

Non-Static Loops with Internal Timing

- Number of iterations determined by
  - Variable modified within the loop
  - Variable that can't be determined at compile time
- Due to internal timing control—
  - Distributed over multiple cycles
  - Number of cycles determined by variable above
- Variable must still be bounded

```
always begin
    continue = 1'b1;
    while (continue) begin
        @(posedge clk) sum = sum + in;
        if (sum > 8'd42) continue = 1'b0;
    end
end
```

Can this be synthesized?

What does it synthesize to?

Who really cares!
This is a stupid way to do it!
Use a SM.

18

18

Any loop with internal timing can be done as a SM

```
module sum_till (clk,rst_n,sum,in);
input clk,rst_n;
input [5:0] in;
output [5:0] sum;
always @(posedge clk, negedge rst_n)
  if (!rst_n)
    sum <= 6'h00;
  else if (en_sum)
    sum <= sum + in
assign en_sum = (sum<6'd43) ? 1'b1 : 1'b0;
endmodule
```

Previous example didn't really even require a SM.

This code leaves no question how it would synthesize.

RULE OF THUMB:

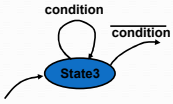
If it takes you more than 15 seconds to conceptualize how a piece of code will synthesize, then it will probably confuse Synopsys too.

19

19

FSM Replacement for Loops

- Not all loop structures supported by vendors
- Can always implement a loop with internal timing using an FSM
  - Can make a "while" loop easily
  - Often use counters along with the FSM



- All synthesizers support FSMs!
- Synopsys supports for-loops with a static number of iterations

20

20

End of part I of Lecture08

21

21

Generated Instantiation

- Generate statements ➔ control over the instantiation/creation of:
  - Modules
  - gate primitives
  - continuous assignments
  - **initial** blocks & **always** blocks
- Generate instantiations resolved during "elaboration" (compile time)
  - ✓When module instantiations are linked to module definitions
  - ✓***Before*** the design is simulated or synthesized – this is NOT dynamically created hardware

22

22

Generate-Loop

- A generate-loop permits making one or more instantiations (pre-synthesis) using a for-loop.

```
module gray2bin1 (bin, gray);
  parameter SIZE = 8;    // this module is parameterizable
  output [SIZE-1:0] bin; input [SIZE-1:0] gray;
  genvar i;
  generate
    for (i=0; i<SIZE; i=i+1) begin: bit
      assign bin[i] = ^gray[SIZE-1:i];  \\ Data Flow replication
    end
  endgenerate
endmodule
```

Does not exist during simulation of design

Typically name the generate as reference

23

23

Generate Loop

- Is really just a code replication method. So it can be used with any style of coding. Gets expanded prior to simulation.

```
module replication_struct(i0, i1,  out);
parameter N=32;
input [N-1:0] i1,i0;
output [N-1:0] out;
genvar j;
generate
for (j=0; j<N; j=j+1)
  begin : xor_loop
    xor g1 (out[j],in0[j],in1[j]);
  end
endgenerate
endmodule
```

Hierarchical reference to these instantiated gates will be:

…xor_loop[0].g1
…xor_loop[1].g1
.
.
.
…xor_loop[31].g1

Structural replication

24

24

Generate-Conditional

- A generate-conditional allows conditional (pre-synthesis) instantiation using **if-else-if** constructs

```
module multiplier(a ,b ,product);
  parameter a_width = 8, b_width = 8;
  localparam product_width = a_width+b_width;
  input [a_width-1:0] a; input [b_width-1:0] b;
  output [product_width-1:0] product;
  generate
    if ((a_width < 8) || (b_width < 8))
      CLA_multiplier #(a_width,b_width) u1(a, b, product);
    else
      WALLACE_multiplier #(a_width,b_width) u1(a, b, product);
  endgenerate
endmodule
```

These are parameters, not variables!

25

25

Generate-Case

- A generate-case allows conditional (pre-synthesis) instantiation using case constructs
- See Standard 12.1.3 for more details

```
module adder (output co, sum, input a, b, ci);
  parameter WIDTH = 8;
  generate
    case (WIDTH)
      1: adder_1bit x1(co, sum, a, b, ci); // 1-bit adder implementation
      2: adder_2bit x1(co, sum, a, b, ci); // 2-bit adder implementation
      default: adder_cla #(WIDTH) x1(co, sum, a, b, ci);
    endcase
  endgenerate
endmodule
```

Of course case selector has to be deterministic at elaborate time, can not be a variable. Usually a parameter.

26

26

Generate-Conditional (FAST_SIM)

- Sometimes we need a way to accelerate the function of a block so it is practical to simulate.

```
module IR_intf(input clk, rst_n, line_present, output [11:0] IR_R0, …);

  parameter FAST_SIM = 0;
    .
    .
    .
  generate if (FAST_SIM) begin
      assign next_round = &timer[13:0];
      assign settled = &timer[10:0];
  end else begin
      assign next_round = &timer;
      assign settled = &timer[12:0];
  endgenerate

endmodule
```

27

27

Synthesis of **x** and **z**

- Only allowable uses of **x** is as "don't care", since **x** cannot actually exist in hardware
  - in **casex** or **case inside()**
  - in defaults of conditionals such as **:**
    - The **else** clause of an **if** statement
    - The **default** selection of a **case** statement
- Only allowable use of **z**:
  - Constructs implying a 3-state output

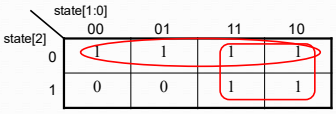
28

28

Don't Cares

- x, ?, or z within case item in **casex** or **case inside()**
  - Does not actually output "don't cares"!
  - Values for which input comparison to be ignored
  - Simplifies the case selection logic for the synthesis tool

```
case (state) inside()
  3'b0??: out = 1'b1;
  3'b10?: out = 1'b0;
  3'b11?: out = 1'b1;
endcase
```



out = $\overline{state[2]}$ + state[1]

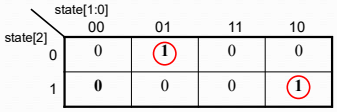
29

29

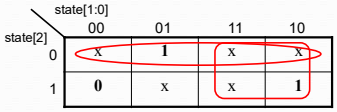
Use of Don't Care in Outputs

- Can really reduce area

```
case (state)
  3'b001: out = 1'b1;
  3'b100: out = 1'b0;
  3'b110: out = 1'b1;
  default: out = 1'b0;
endcase
```




```
case (state)
  3'b001: out = 1'b1;
  3'b100: out = 1'b0;
  3'b110: out = 1'b1;
  default: out = 1'bx;
endcase
```



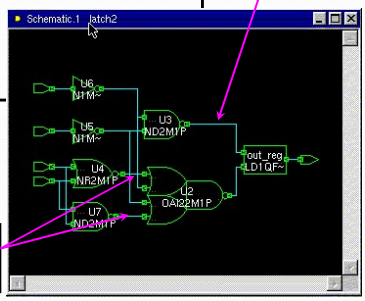
30

30

Synthesis Example [1]

```
module Hmmm(input a, b, c, d, output reg out);
always_comb begin
  if (a) out = c | d;
  else if (b) out = c & d;
end
endmodule
```

a | b enables latch



How will this synthesize?

Area = 44.02

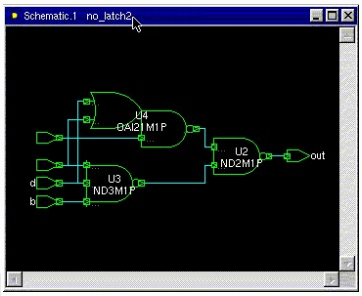
Either **c | d** or **c&d** are passed through an inverting mux depending on state of **a** / **b**

31

31

Synthesis Example [2]

```
module Better(input a, b, c, d, output reg out);
always @(a, b, c, d) begin
  if (a) out = c | d;
  else if (b) out = c & d;
  else out = 1'b0;
end
endmodule
```



Perhaps what you meant was that if not **a** or **b** then **out** should be zero??

Area = 16.08

Does synthesize better…no latch!

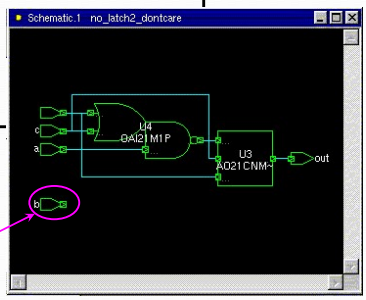
32

32

Synthesis Example [3]

```
module BetterYet(input a, b, c, d, output reg out);
always @(a, b, c, d) begin
  if (a) out = c | d;
  else if (b) out = c & d;
  else out = 1'bx;
end
endmodule
```

Area = 12.99



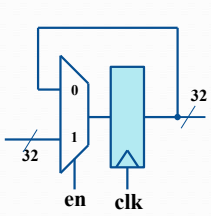
But perhaps what you meant was if neiter **a** nor **b** then you really don't care what **out** is.

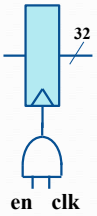
Hey!, Why is **b** not used?

33

33

Gated Clocks





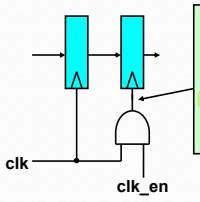
Recirculating seems rather wasteful we need a 2:1 mux for each flop. Plus all the cap and gates inside the flops on the clock network will be toggling.

Wouldn't it be better to just gate the clock? Pretty much same functionality but should save area and power? Yes, but you have to do it right.

34

Gated Clocks

- Use only if necessary (e.g., for low-power)
  - Has become quite necessary with much demand for battery operated devices.



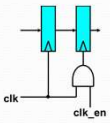
This clock arrives later than the system clock it was derived from. We just created a min-delay problem. (race condition) (shoot through)

$$Min\_delay\_slack = clk2q - T_{Hold} - Skew\_Between\_clks$$

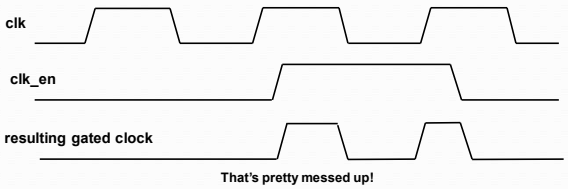
35

35

Gated Clocks (not so simple)



Something this simple does not work. Recall the enable function is probably coming from a SM. So it changes slightly after the rising edge of clock.

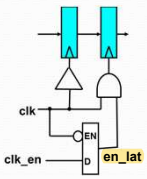


That's pretty messed up!

36

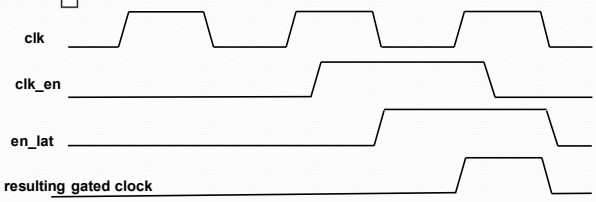
36

Gated Clocks (do it this way)



Have to use an enable low latch to latch the clock enable signal. This ensures the enable is setup and ready for the next high phase of clock

Have to balance from clock root to all leaf nodes of clock



37

Gated Clocks

Gated clock domains can't be treated lightly:

1.) Skew between domains

2.) Loading of each domain. How much capacitance is on it? What is its rise/fall times

3,) RC in route. Is it routed in APR like any old signal, or does it have priority?

Clocks are not signals…don't treat them as if they were.

1.) Use clock tree synthesis (CTS) within the APR tool to balance clock network (usually the job of a trained APR expert)

2.) Paranoid control freaks (like me) like to generate the gated domains in custom logic (like clock reset unit). Then let CTS do a balanced distribution in the APR tool.

38

38

End of part II of Lecture08

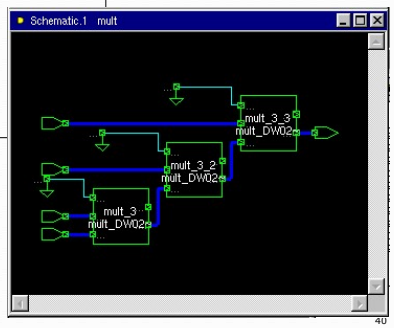
39

39

Chain Multiplier

```
module mult(output reg [31:0] out,
            input [31:0] a, b, c, d);

  always@(*) begin
    out = ((a * b) * c) * d;
  end

endmodule
```

Area: 47381
Delay: 8.37

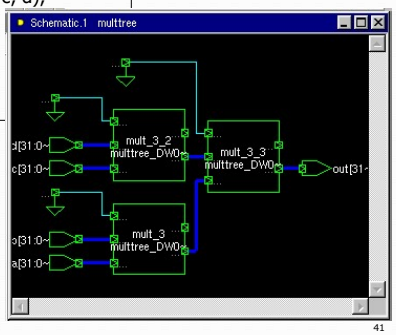


40

40

Tree Multiplier

```
module multtree(output reg [31:0] out,
                input [31:0] a, b, c, d);
  always@(*) begin
    out = (a * b) * (c * d);
  end
endmodule
```



Area: 47590
Delay: 5.75 vs 8.37

41

41

Multi-Cycle Shared Multiplier

```
module multshare(output reg [31:0] out,
                 input [31:0] in, input clk, rst);
  reg [1:0] cycle;
  wire [31:0] multval;

  always @(posedge clk)
    if (rst) cycle <= 0;
    else cycle <= cycle + 1;

  always @(posedge clk)
    out <= (|cycle) ? out*in : in;

endmodule
```

This design uses a 2-bit cycle counter, and forms the desired across 4 clock cycles.

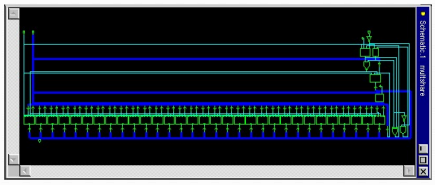
42

42

Multi-Cycle Shared Multiplier (results)

Area: 15990 vs 47500
Delay: 4*3.14

4 clocks, minimum period 3.14



43

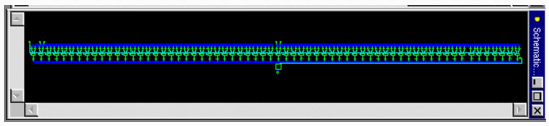
43

Shared Conditional Multiplier

```
module multcond1(output reg [31:0] out,
                 input [31:0] a, b, c, d, input sel);

always @(*) begin
  if (sel) out = a * b;
  else out = c * d;
end

endmodule
```

Mutually exclusive use of the multiply

Area: 15565
Delay: 3.14



44

44

Selected Conditional Multiplier [1]

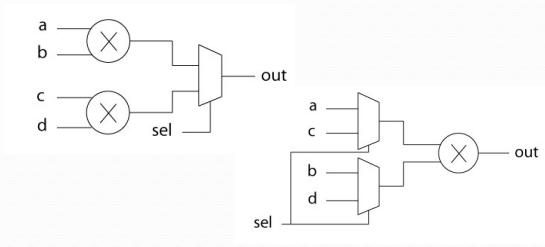
```
module multcond2(output reg [31:0] out,
                 input [31:0] a, b, c, d, input sel);

wire [31:0] m1, m2;
assign m1 = a * b;
assign m2 = c * d;


always @(*) begin
  if (sel) out = m1;
  else out = m2;
end

endmodule
```

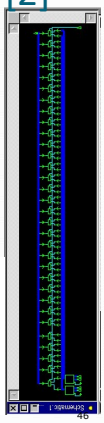
45

45

Selected Conditional Multiplier [2]

- Area: 30764 vs. 15565
- Delay: 3.02 vs. 3.14
- Why is the area larger?





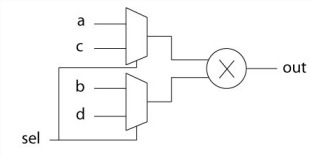
46

46

Conditional Multipler – One More Time

- If you know ahead of time that you want two muxes and one multiplier, describe that directly!
- Don't rely on the synthesis tool to improve inefficient HDL; describe what you want first.
- Caveat: You have to know what you want.

```
module multcond2(output reg [31:0] out,
        input [31:0] a, b, c, d, input sel);
        wire [31:0] op1, op2;
        assign op1 = sel ? a : c;
        assign op2 = sel ? b : d;

        always @(*) begin
                out = op1 * op2;
endmodule
```



47

47

Late-Arriving Signals

- After synthesis, we will identify the critical path(s) that are limiting the overall circuit speed.
- It is often that one signal to a datapath block is late arriving.
- This signal causes the critical path…how to mitigate?:
  - Circuit reorganization
    - ✓Rewrite the code to restructure the circuit in a way that minimizes the delay with respect to the late arriving signal
  - Logic duplication
    - ✓This is the classic speed-area trade-off. By duplicating logic, we can move signal dependencies ahead in the logic chain.

48

48

Logic Reorganization Example [1]

Example 4-5 Original Verilog With Operator in Conditional Expression

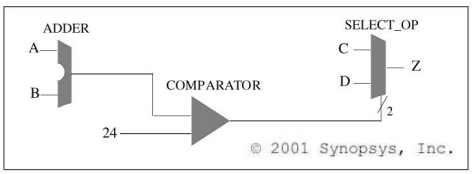
```
module cond_oper(A, B, C, D, Z);
parameter N = 8;
input [N-1:0] A, B, C, D; //A is late arriving
output [N-1:0] Z;

reg [N-1:0] Z;

always @(A or B or C or D)
begin
     if (A + B < 24)
          Z <= C;
     else
          Z <= D;
end

endmodule
```


49

49

Logic Reorganization Example [2]

Figure 4-3 Structure Implied by Original HDL With Late Arriving A Signal



What can we do if **A** is the late-arriving signal?

50

50

Logic Reorganization Example [3]

Example 4-7 Improved Verilog With Operator in Conditional Expression

```
module cond_oper_improved (A, B, C, D, Z);
parameter N = 8;
input [N-1:0] A, B, C, D; // A is late arriving
output [N-1:0] Z;

reg [N-1:0] Z;

always @(A or B or C or D)
begin
     if (A < 24 - B)
          Z <= C;
     else
          Z <= D;
     end
     endmodule
```

That's right! We have to do the math, and re-arrange the equation so the comparison does not involve and arithmetic operation on the late arriving signal.

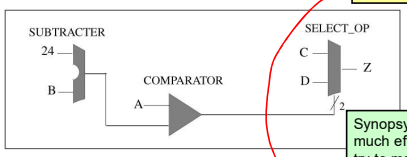
51

51

Logic Reorganization Example [4]

Figure 4-4 Structure Implied by Improved HDL With A as Input to Compar ator

Why the area improvement?



Synopsys didn't spend so much effort upsizing gates to try to make transisitons faster. This new design is **faster**, **lower area**, and **lower power**

Table 4-2 Timing and Area Results for Conditional O

	Data Arrival Time	Area
Original Design	4.33	411.1
Improved Design	3.89	271.0

52

52

Logic Duplication Example [1]

Example 4-1 Original Verilog Before Logic Duplication

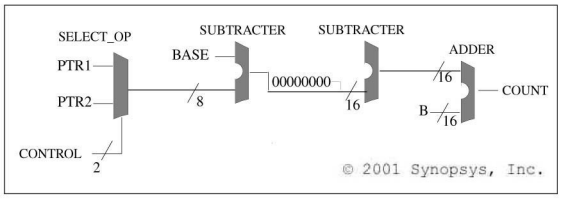
```
module BEFORE (ADDRESS, PTR1, PTR2, B, CONTROL, COUNT);
input [7:0] PTR1,PTR2;
input [15:0] ADDRESS, B;
input CONTROL;          // CONTROL is late arriving
output [15:0] COUNT;

parameter [7:0] BASE = 8'b10000000;
wire [7:0] PTR, OFFSET;
wire [15:0] ADDR;

assign PTR = (CONTROL == 1'b1) ? PTR1 : PTR2;
assign OFFSET = BASE - PTR; //Could be any function f(BASE,PTR)
assign ADDR = ADDRESS - {8'h00, OFFSET};
assign COUNT = ADDR + B;

endmodule
```

53

53

Logic Duplication Example [2]

Figure 4-1 Structure Implied by Original HDL Before Logic Duplication



What if *control* is the late arriving signal?

54

54

Logic Duplication Example [3]

Example 4-3 Improved Verilog With Data Path Duplicated

```
module PRECOMPUTED (ADDRESS, PTR1, PTR2, B, CONTROL, COUNT);
input [7:0] PTR1, PTR2;
input [15:0] ADDRESS, B;
input CONTROL;
output [15:0] COUNT;

parameter [7:0] BASE = 8'b10000000;
wire [7:0] OFFSET1,OFFSET2;
wire [15:0] ADDR1,ADDR2,COUNT1,COUNT2;

assign OFFSET1 = BASE - PTR1;  // Could be f(BASE,PTR)
assign OFFSET2 = BASE - PTR2;  // Could be f(BASE,PTR)
assign ADDR1 = ADDRESS - {8'h00 , OFFSET1};
assign ADDR2 = ADDRESS - {8'h00 , OFFSET2};
assign COUNT1 = ADDR1 + B;
assign COUNT2 = ADDR2 + B;
assign COUNT = (CONTROL == 1'b1) ? COUNT1 : COUNT2;

endmodule
```

55

55

Logic Duplication Example [4]

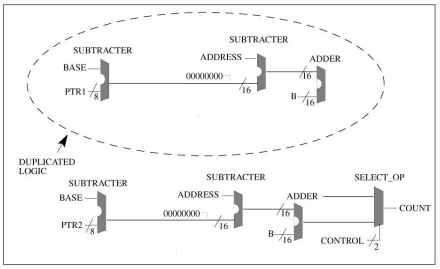


Table 4-1 Timing and Area Results for Data-Path Duplication

	Data Arrival Time	Area
Original Design	5.23	1057
Improved Design	2.33	1622

56

56

Conclusions

- The designer is responsible for some optimizations that cannot be achieved by the synthesis tool.
- It takes a lot of knowledge to be an expert designer
  - Hardware Design
  - HDL
  - Synthesis Tool
- One of the largest roles of the designer is to understand tradeoffs and make appropriate decisions

57