if/else, unique, priority

- To create hierarchy and create combinatorial logic gates, we use the `always_comb` block.
- Inside the `always_comb` block, we describe the behavior of combinational logic in a sequential, algorithmic way with `if`, `else`, `while` and `case` statements.
- These statements, inherited from procedural languages, provide a powerful means of expression. We therefore refer to the `always_comb` block as a procedural block.
- Keep in mind that you are not writing a program, but instead you are describing how the *logic* behaves.
- Digital logic does not "execute" statement by statement but instead it evaluates in a parallel "all-at-once", and "all-the-time" fashion.

if/else, unique, priority

- If there are multiple statements within always_comb, they must be bracketed with `begin`, `end` statements.

```
always_comb begin
  prod_reg_ld_high  = 1'b0;  //statement 1
  prod_reg_shift_rt = 1'b0;  //statement 2
end //always_comb
```

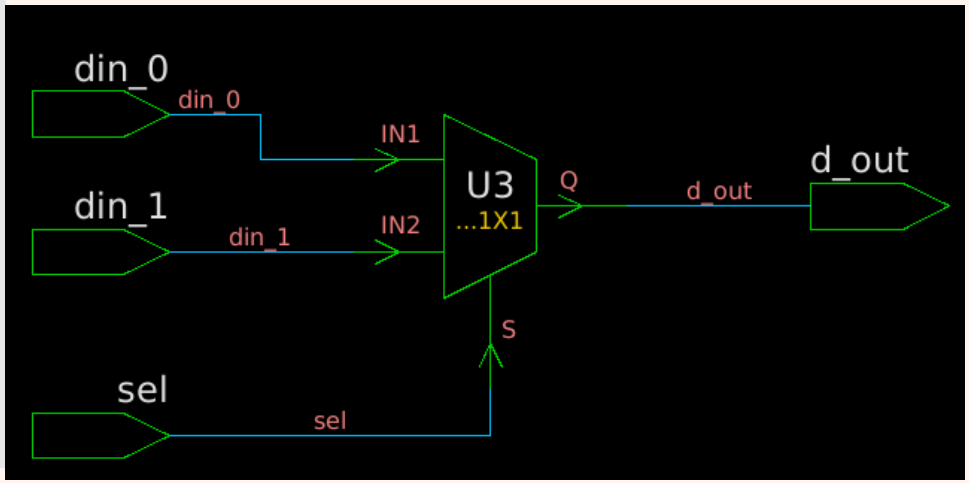
- Single statements within always_comb do not need `begin`, `end`.

```
  always_comb
    if (sel) d_out = din_1;
    else     d_out = din_0;
```

if/else, unique, priority

- The `if` statement creates combinatorial logic that outputs a value depending upon the value of a conditional expression.
- If the conditional expression evaluates true (non-zero), the following signal assignment is made. Othewise, that assignment is skipped and the next statement is evaluated.
- A conditional expression evaluates to false when its `X`, `Z`, or `zero`.
- Note the use of blocking assignments.

```
module mux2_1 (
  input          sel,
  input          din_0,
  input          din_1,
  output logic   d_out);
  always_comb
    if (sel) d_out = din_1;
    else     d_out = din_0;
endmodule
```



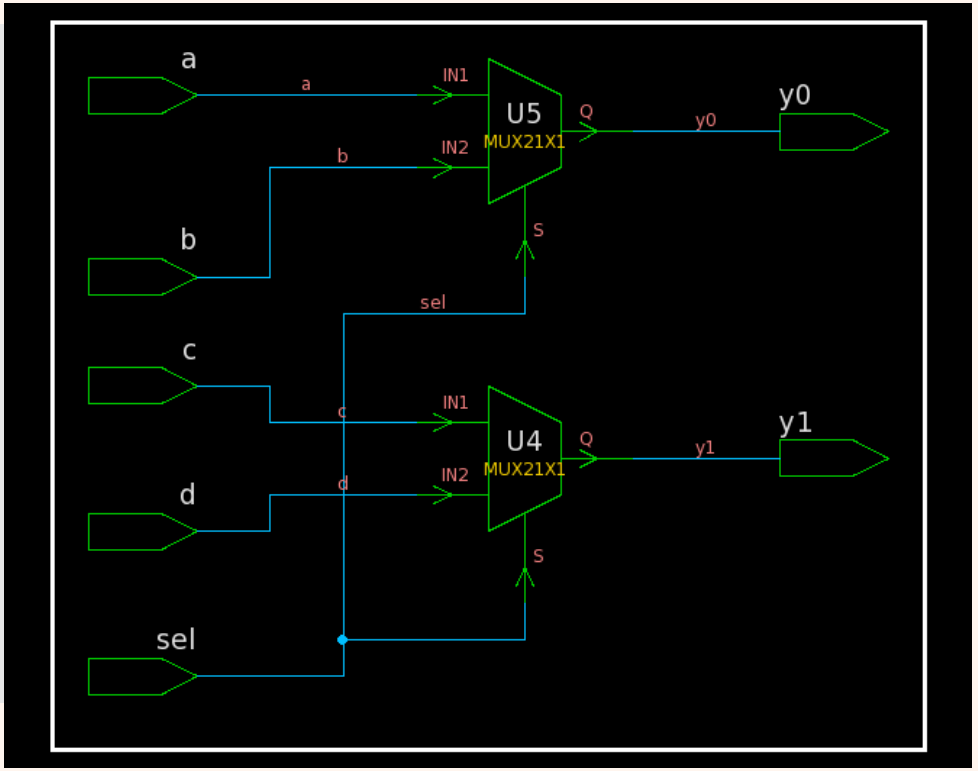
Synthesis Output

Note: ridiculously small module, example only

if/else, unique, priority

- If there are multiple assignment statements they must be grouped within a begin, end set.

```
module dual_mux2_1 (
  input          sel,
  input          a,b,c,d,
  output logic   y0,
  output logic   y1);
  always_comb
    if (sel) begin y0 = b;
                   y1 = d;
             end
    else     begin y0 = a;
                   y1 = c;
             end
endmodule
```



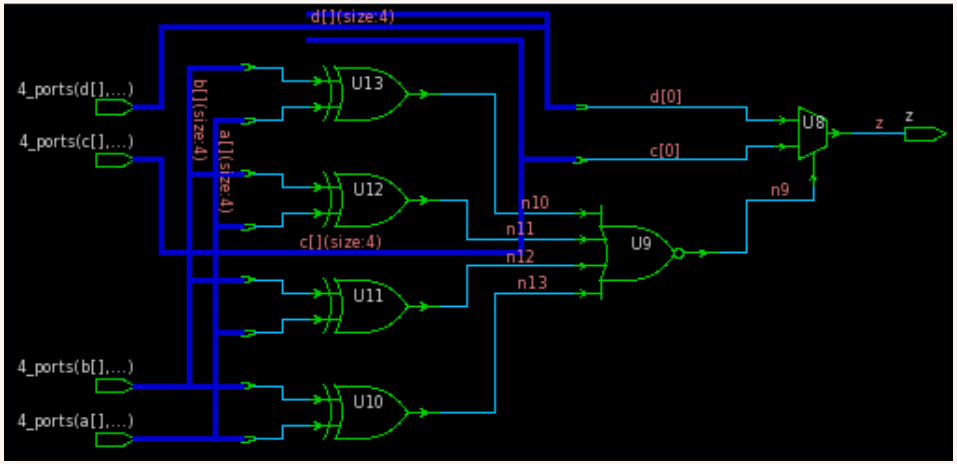
Synthesis Output

if/else, unique, priority

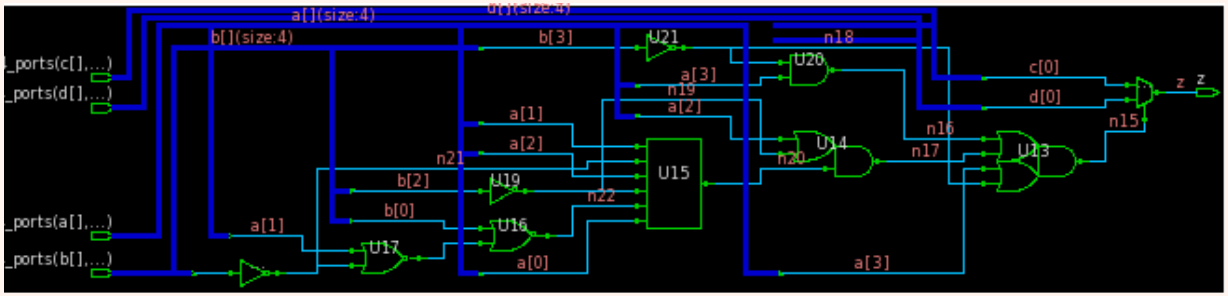
- The `if` statement tests a conditional expression to determine which output assignment to make.
- Realtional operators in the conditional expression can add substantial additonal logic.
- Commonly used relational operators used are:
  - equals ($==$)
  - not-equals ($!=$)
  - greater-than ($>$)
  - less-than ($<$)
  - greater-than-or-equal-to ($>=$)
  - less-than-or-equal-to ($<=$)
- How would you realize these operators in logic gates? Which would be cheapest?

if/else, unique, priority

- Four-bit, 2:1 Mux with 4-bit comparison operators.



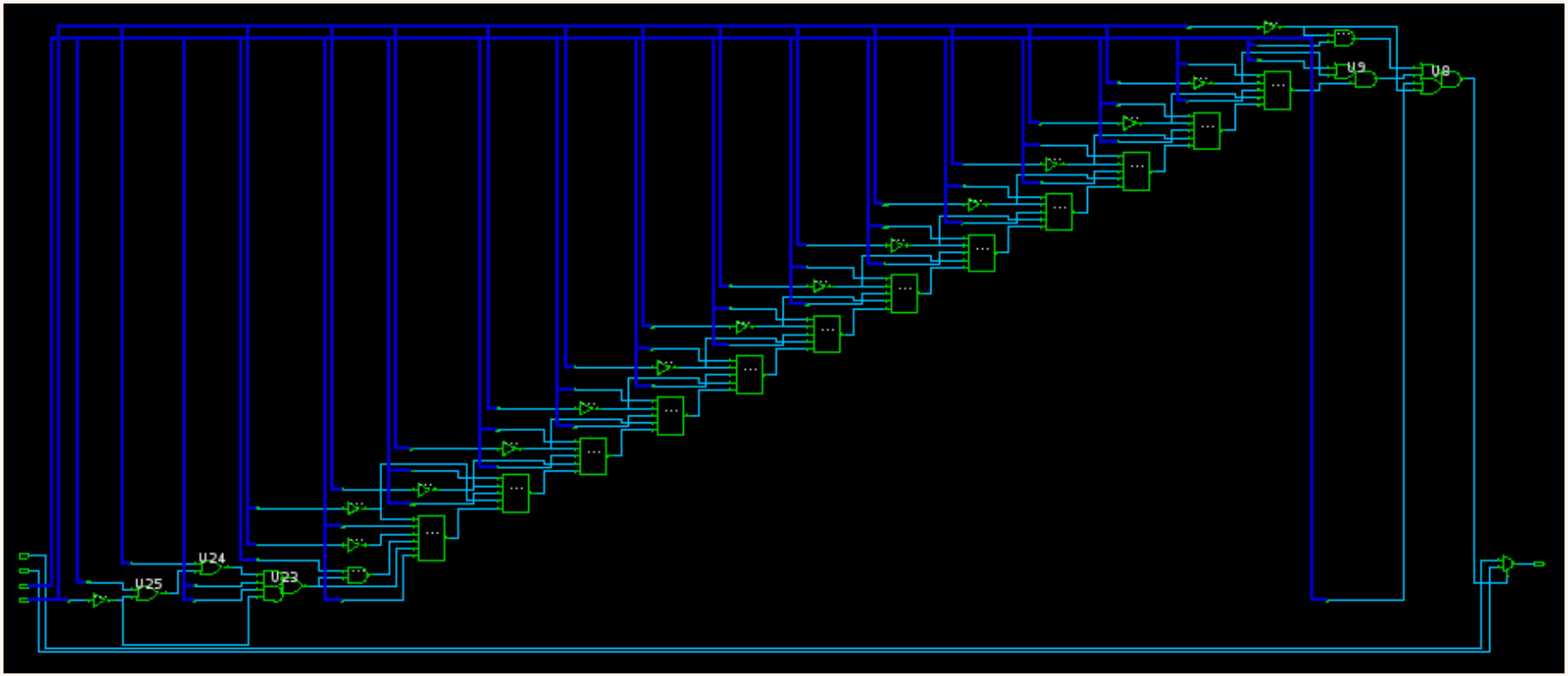

```
if (a==b) z = c; //equals uses XOR primarily
else      z = d;
```




```
if (a<=b) z = c; //lteq, gteq, takes far more gates
else      z = d;
```

if/else, unique, priority

- Wide comparisons can be produce lots of deep and thus slow logic.
- Here, the comparison logic is far bigger that the desired 2x1 mux at bottom right.



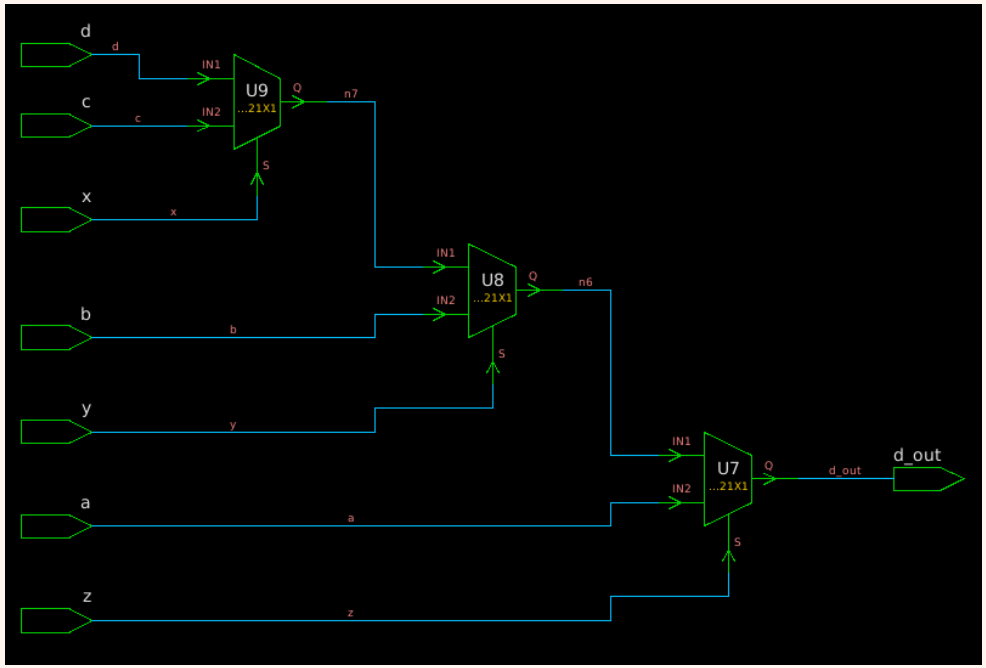

```
if (a<=b) z = c;
else      z = d;
```

if/else, unique, priority

- Deeply nested `if` statements can produce slow logic.
- Some paths are far slower than others.

```
module serial_mux (
  input  a,b,c,d
  input  x,y,z
  output logic d_out
  );

  always_comb
    if       (z) d_out = a;
    else if  (y) d_out = b;
    else if  (x) d_out = c;
    else         d_out = d;
endmodule
```



if/else, unique, priority

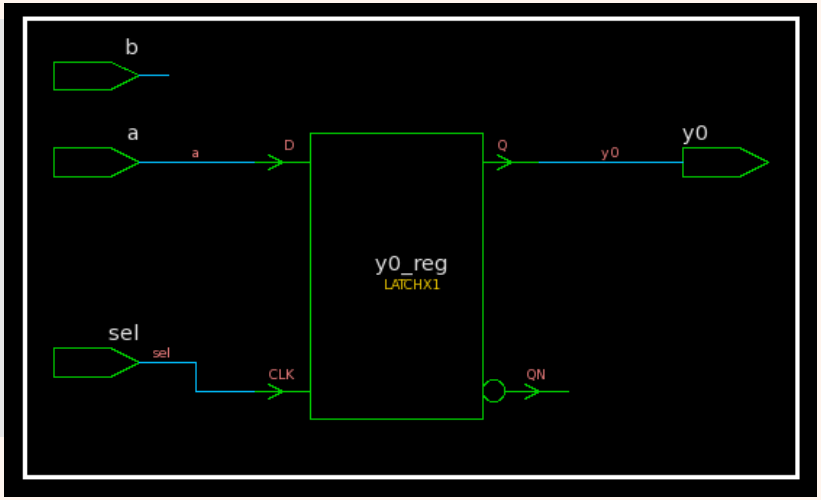
- The schematics seen here are produced by Synopsys Design Compiler aka "`dc_shell`" or "`design_vision-xg`" using a standard cell logic library.
- FPGA combinational logic uses SRAM looktables.
- As such, an *apples-to-apples* comparison is not possible.
- However, standard cell synthesis helps us to see the complexity of inferred gate structures.
- FPGA SRAM memories can be used to create wide logic structures that have uniform delay paths. Why?

if/else, unique, priority

- If an `if` statement has no `else`, the synthesizer has to make an assumption as to what the newest value of the output is to be.
- The assumption, wrong or right, is to hold the last value. Note the synthesis warning. Hence the saying...*"if without else gives a latch"*

```
Warning: no_else_mux.sv:6:
Netlist for always_comb block contains a latch.
```

```
module no_else_mux (
  input         sel,
  input         a,b,
  output logic  y0   );
  always_comb
    if (sel) y0 = a;
//  else     y0 = b; //commented
endmodule
```

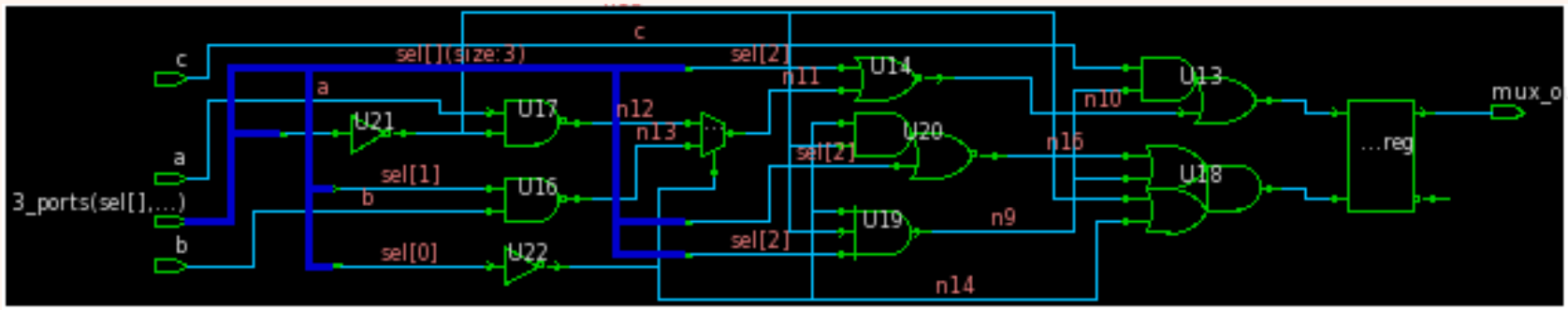


Synthesis Output

if/else, unique, priority

- The decision modifiers `unique` and `priority` reduce ambiguities and can trap potential design errors early.
- What is the intention here? Priority or not? Note inferred latch.

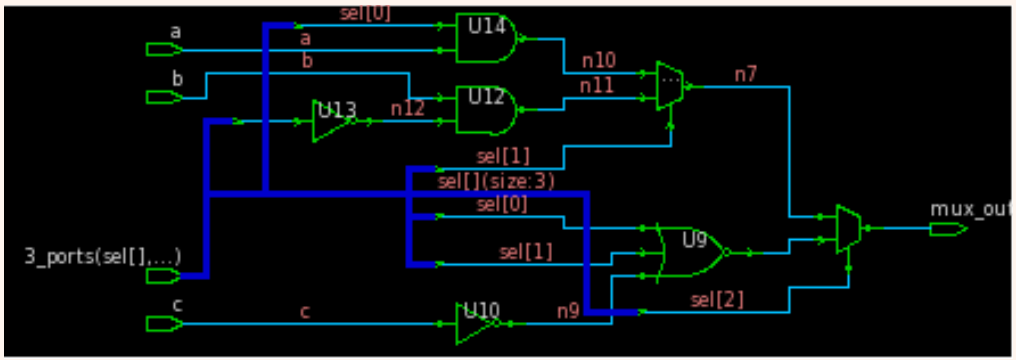
```
module nounique(
input [2:0]    sel,  //selection input
input          a,b,c,
output logic   mux_out);
always_comb
  if        (sel == 3'b001) mux_out = a;
  else if   (sel == 3'b010) mux_out = b;
  else if   (sel == 3'b100) mux_out = c;
endmodule
```



if/else, unique, priority

- The simulator evaluates `if...else...if` decisions in order and synthesis creates logic enforcing priority so that RTL and gate simulation behave identically.
- What if you didn't care about order or priority? The `unique` modifier lets you say so. The result is no priority logic, and no latch.

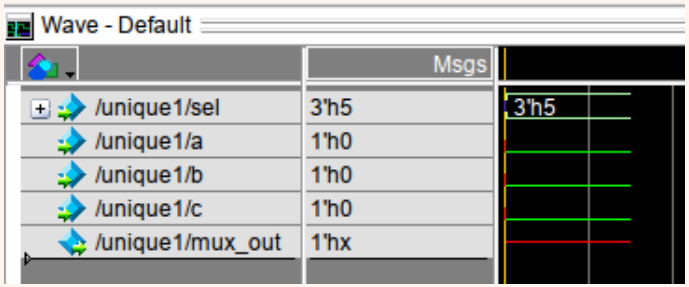
```
module unique1(
input [2:0]    sel,  //selection input
input          a,b,c,
output logic   mux_out);
always_comb
  unique if (sel == 3'b001) mux_out = a;
  else if   (sel == 3'b010) mux_out = b;
  else if   (sel == 3'b100) mux_out = c;
endmodule
```



if/else, unique, priority

- Another action of the `unique` modifier is to direct the simulator to issue a run-time warning if none of the `if` branchs are executed.
- In our present example, a run-time warning would be issued if `sel` was any other value than 1, 2, or 4.

```
VSIM 73> force a 0
VSIM 74> force b 0
VSIM 75> force c 0
VSIM 76> force sel 16#5
VSIM 77> run 50
# ** Warning: (vsim-8315) uniquel.sv(6): No condition is true in the unique/priority if/case statement.
#    Time: 0 ns  Iteration: 0  Instance: /uniquel
```

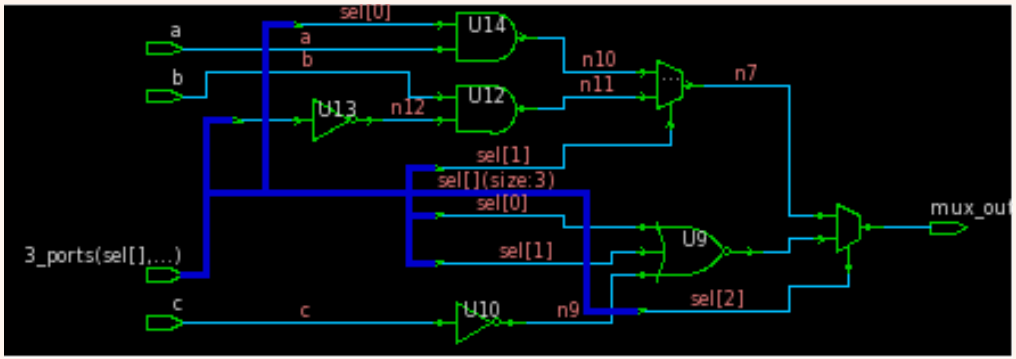


- Note that the simulator output is "`x`" for `sel=3'b101`. What would the gate output be? Can real gates ever have an output of "`x`"?

if/else, unique, priority

- Since unique gives us a way to check in simulation that at least one branch is taken, a "concluding else" is not needed to prevent latches.

```
module unique1(
input [2:0]    sel,  //selection input
input          a,b,c,
output logic   mux_out);
always_comb
  unique if (sel == 3'b001) mux_out = a;
  else if   (sel == 3'b010) mux_out = b;
  else if   (sel == 3'b100) mux_out = c;
endmodule
```



if/else, unique, priority

- Here, potential for overlapping conditions exist as any number of conditions could be true at once.

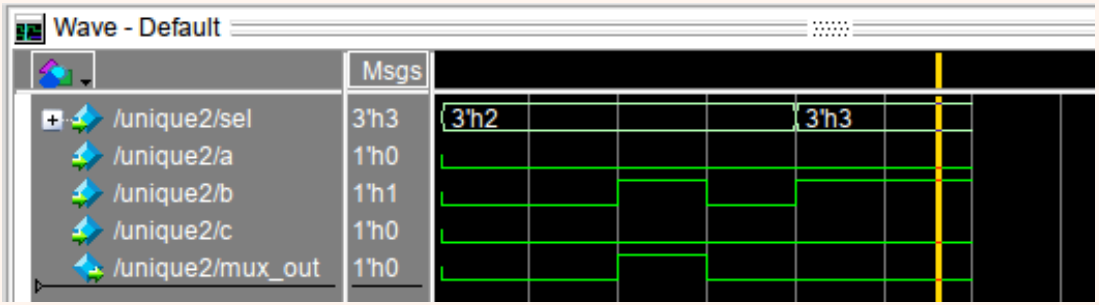
```
module unique2(
input [2:0]    sel,  //selection input
input          a,b,c,
output logic   mux_out);
always_comb
  unique if (sel[0]) mux_out = a;
  else if   (sel[1]) mux_out = b;
  else if   (sel[2]) mux_out = c;
endmodule
```

- If `unique` removes priority encoding, the conditions must be mutually exclusive. Therefore in this situation, `unique` acts as an assertion, telling the simulator to issue a run-time warning if more than one condition ever true.

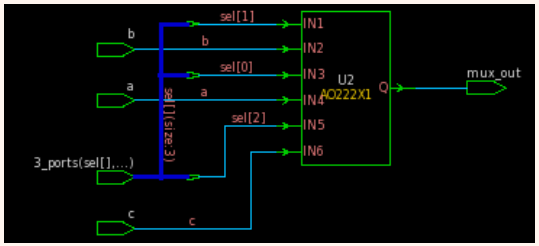
if/else, unique, priority

- Forcing `sel[2:0]` to 3'b011 makes the conditions non-mututally exclusive and the simulator issues a warning.

```
force sel 16#3
run 100
#  Warning: (vsim-8360) unique2.sv(6):
#  The if/case statement is not unique.
#  Time: 200 ns  Iteration: 0  Instance: /unique2
```



- This totally appropriate as if priority was enforced, the "a" input should have been chosen. The synthesis results show why.



if/else, unique, priority

- `unique if` final points...
- At any point in time, `unique` causes the simulator to issue a run-time warning if:
  - *more than one condition is true*
  - *no condition is true and there is no else branch*
- `unique`, informs the synthesis tool that:
  - *every legal condition has been listed*
  - *the conditions are mutually exclusive*
  - *no priority logic is required*
- `unique` is placed just before the `if`.
- In a series of `if...else` decisions, `unique` is only specified for the first `if`. All subsequent if statements within the sequence are affected.
- Presently, Quartus does not support modifiers for `if`, only `case`.

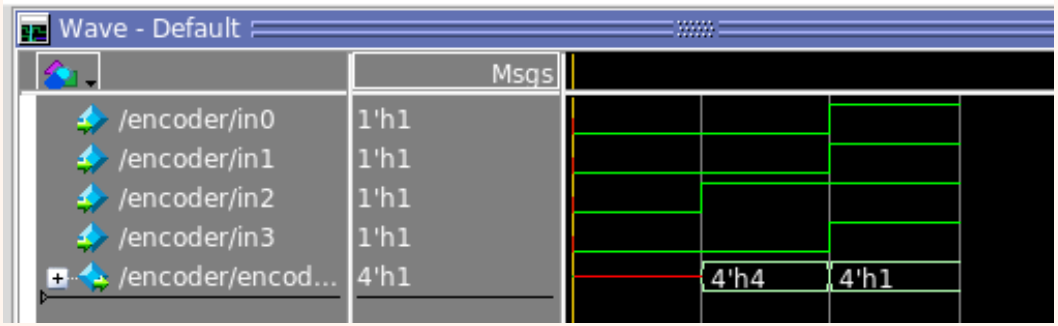
if/else, unique, priority

- The decision modifier `priority` points to the designer's intention that the order of decisions is important.
- Each conditional statement is evaluated in the order listed.
- An example would be a priority encoder.

```
module encoder(
input                 in0, in1, in2, in3,
output logic [3:0]    encoded_output;
always_comb
  priority if (in0) encoded_output = 4'b0001;
  else if      (in1) encoded_output = 4'b0010;
  else if      (in2) encoded_output = 4'b0100;
  else if      (in3) encoded_output = 4'b1000;
endmodule
```

if/else, unique, priority

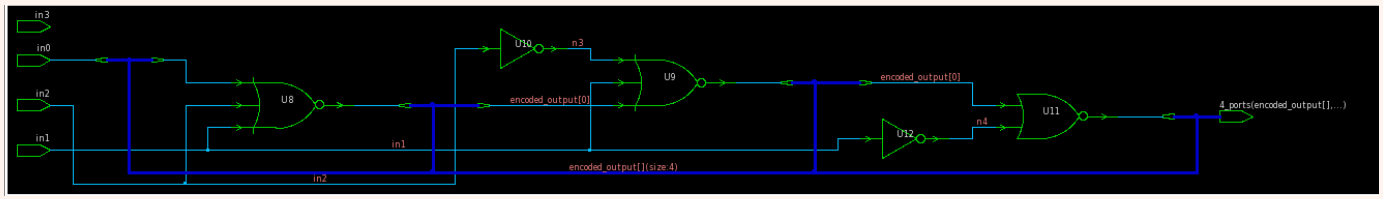
- The priority encoder in operation:



- Note run-time warning in the first 100ns.

```
VSIM 6> run 100
# ** Warning: (vsim-8315) encoder.sv(5): No condition is true in the unique/priority if/case statement.
#    Time: 0 ns  Iteration: 0  Instance: /encoder
```

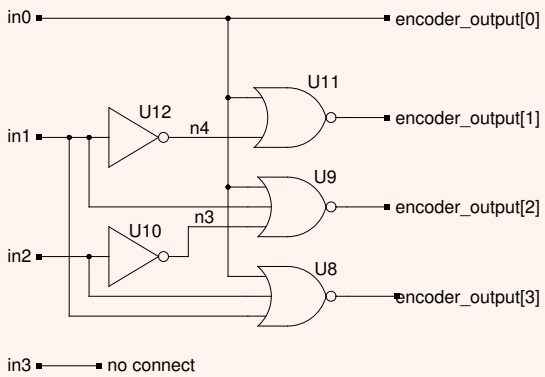
- Synthesis results are rather unexpected



if/else, unique, priority

- Redraw schematic from the synthesis netlist. Is this right?

```
////////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : L-2016.03-SP2
////////////////////////////////////////////////////////////////
module encoder ( in0, in1, in2, in3, encoded_output );
  output [3:0] encoded_output;
  input in0, in1, in2, in3;
  wire   n3, n4;
  assign encoded_output[0] = in0;
  NOR3X0 U8  (.IN1(encoded_output[0]), .IN2(in2), .IN3(in1), .QN(encoded_output[3]) );
  NOR3X0 U9  (.IN1(n3), .IN2(in1), .IN3(encoded_output[0]), .QN(encoded_output[2])  );
  INVX0 U10  (.IN(in2), .QN(n3)                                                     );
  NOR2X0 U11 (.IN1(encoded_output[0]), .IN2(n4), .QN(encoded_output[1])             );
  INVX0 U12  (.IN(in1), .QN(n4)                                                     );
endmodule
```



if/else, unique, priority

- `priority if` final points...
- At any point in time, `priority` causes the simulator to issue a run-time warming if:
  - *no condition is true and there is no else branch*
- `priority`, informs the synthesis tool that:
  - *every legal condition has been listed*
  - *priority logic is required*
- `priority` is placed just before the `if`.
- In a series of `if...else` decisions, `priority` is only specified for the first `if`. All subsequent if statements within the sequence are affected.
- Presently, Quartus does not support modifiers for `if`, only `case`.