CME 341: Synthesizable Logic Best Practices

Brian Berscheid

Department of Electrical and Computer Engineering
University of Saskatchewan

Today's agenda

1. Key guidelines for synthesizable logic

Key guidelines for synthesizable logic

Overview

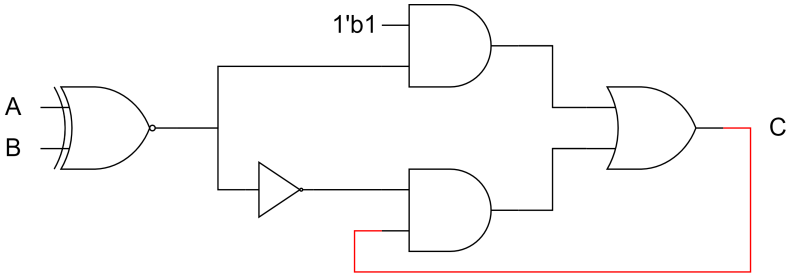
- Students typically spend a large amount of time tracking down bugs in CME 341
- Previous instructors have noticed a set of issues that many students run into
- This topic will hopefully save you time by helping to avoid these common errors
  - ▷ Of course, it is not feasible to cover all potential errors - these are just a few common ones!
- Each error will be presented through one or more examples

Error 1

```
always @ (*)
  if (a == b)
    c = 1;
  else
    c = c;
```

Error 1 - hardware built... latch!

```
always @ (*)
  if (A == B)
    C = 1'b1;
  else
    C = C;
```



Error 1, alternate forms

```
always @ (*)
  if (A==B)
    C = 1'b1;
```

```
always @ (*)
  case(D[1:0])
    2'b10:         C = 1'b1;
    2'b00, 2'b11: C = 1'b0;
  endcase
```

Compiler adds missing case, builds feedback.

Rule 1:
Avoid feedback in combinational circuits!

Error 2

```
always @ (*)
  if (A == 1'b1)
    C = 8'd27;
  else
    C = 8'd0; // (or omit)

always @ (*)
  if (B == 1'b1)
    C = 8'd64;
  else
    C = 8'd0; // (or omit)
```

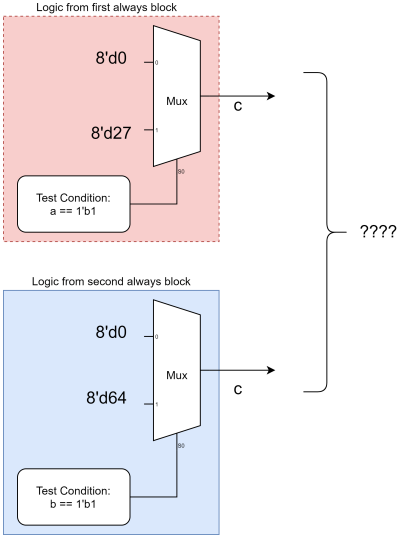
Error 2

```
always @ (*)
  if (A == 1'b1)
    C = 8'd27;
  else
    C = 8'd0; // (or omit)

always @ (*)
  if (B == 1'b1)
    C = 8'd64;
  else
    C = 8'd0; // (or omit)
```

- Remember that Verilog is a parallel language!
- Each procedure is converted into hardware independently during synthesis
- In this case, we have asked the compiler to build two copies of the if-else template
  - ▷ Both are trying to drive the value of signal C
  - ▷ How should they be connected together?
  - ▷ Which should "win"?
- Quartus will issue an error ("multiple constant drivers")

Error 2: requested hardware



Rule 2:
Each variable must be completely defined in one procedure (or assign statement)!

Error 3 - two examples

```
always @ (*)
  begin
    if(x==4'b0001) y = 4'hD;
    if(x==4'b0010) y = 4'hE;
    if(x==4'b1000) y = 4'hA;
    else           y = 4'h0;
end
// should have used else if for
// 2nd and 3rd if statements
```

```
always @ (*)
  begin
    c = 8'd7;  // assign value to c

    // some  logic
    d = c & b; //
    e = f | g; //
    // perhaps more unrelated logic ...
    // finally,
    if (x == 1'b1)
      c = 8'd4;
    else
      c = 8'd12;
  end
```

Error 3 explanation

- Problem: assigning values to a variable more than once in a procedure is a no-no!
- Compiler will use the last assignment within the procedure and ignore the others
  - ▷ A warning message may show up, but no error will be generated
- In complex procedures it can be easy to lose track of when a signal is written to twice
- Strong recommendation for CME 341: avoid begin-end wrappers and use a single procedure per output!

Rule 3:
Avoid begin-end wrappers!
One output per procedure!

Summary of guidelines

- Rule 1: Avoid feedback in combinational circuits!
- Rule 2: Each variable must be completely defined in one procedure (or assign statement)!
- Rule 3: Avoid begin-end wrappers! One output per procedure!
- Following these guidelines will save you many hours of debugging and frustration in this class.
- *These rules are for synthesizable code only (designs that will be compiled in Quartus). When writing testbenches, it is ok to not follow these rules.*

Thank you!
Have a great day!