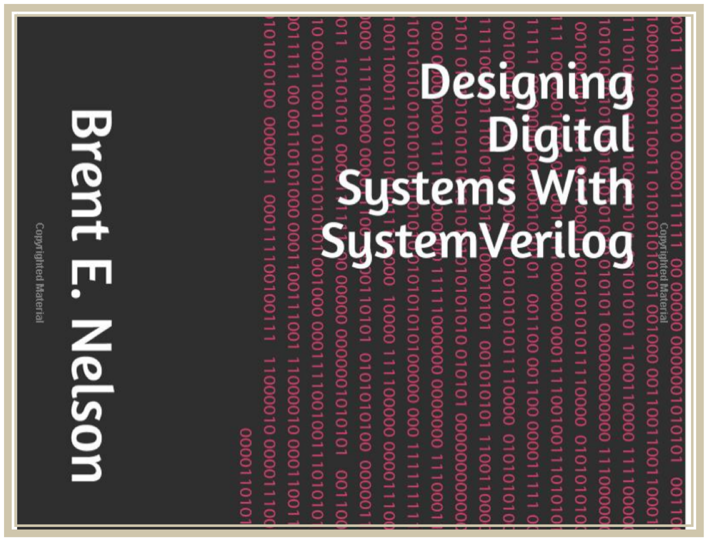
Chapter 18
Combinational Behavioral SystemVerilog

ECEn 220

Fundamentals of Digital Systems

BRIGHAM YOUNG UNIVERSITY



Designing Digital Systems With SystemVerilog

Brent E. Nelson

Computer Engineering
Electrical Engineering

Behavioral Design

- Used for creating both combinational and sequential circuits
  - Combinational: output depends only on inputs
  - Sequential: output depends on inputs and state
- Behavioral SystemVerilog syntax for creating logic is critical – you must learn the syntax style
  - It is possible to accidently create sequential circuits when you want combinational circuits
  - It is possible to create code that is "non-synthesizable"
  - It is easy to make mistakes!

always_comb Blocks

- Combinational logic can be created using always_comb blocks
  - This declares your intent to the synthesizer that you only want combinational logic
  - Throws an error if you unintentionally create sequential circuits
- Can contain case statements, if-then-else statements, etc. (Not available for structural and Dataflow)
- Same operators used in Dataflow are available in always_comb blocks
- Uses the blocking assignment statement
- Considered part of behavioral SystemVerilog

Blocking vs. Non-Blocking

Blocking

- Only used in always_comb blocks
- <variable> **=** <statement>
- Statements are executed sequentially, similar to C-code.
- Used for combinational logic

Non-blocking

- Only used in always_ff blocks
- <variable> **<=** <statement>
- Statements are executed in parallel.
- Used for sequential logic

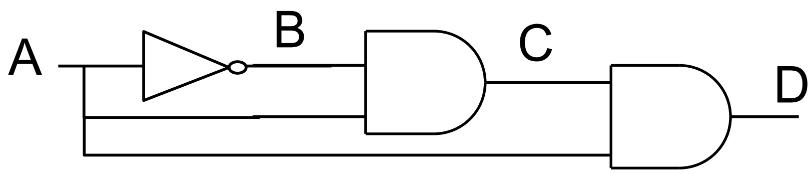
Do not mix blocking and non-blocking assignments in one always block

Blocking vs. Non-Blocking

~~Structural Design – Built-in gates~~
Dataflow – operators, cond. ..
Behavioral Design –
    always_comb
    always_ff

Blocking

```
always_comb
Begin
   B = ~A;
   C = B & A;
   D = C & A;
end
```



```
Module XYZ(….);

assign A = W&X;
assign A = (W > 0) ? 1'b1 : 1'b0;

always_comb
Begin
   B = ~A;
   C = B & A;
   D = C & A;
End

endmodule
```

When a signal changes:
B gets A's value, then
C gets B's new value, and then
D gets C's new value.

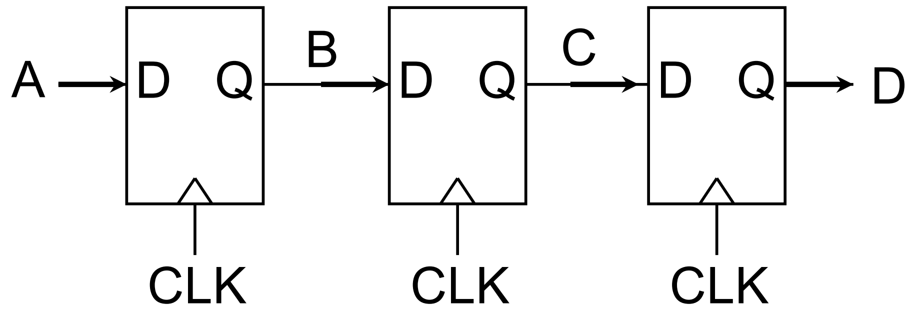
This always block puts A's value into B, C, and D.

Assignments must be in the right order.

Blocking vs. Non-Blocking

Non-Blocking

```
always_ff @(posedge clk)
begin
   B <= A;
   C <= B;
   D <= C;
end
```



On every clock edge:

B gets A’s value,
C gets B’s old value, and
D gets C’s old value.

Order of assignments DOES NOT MATTER…

Individual statement semantics are sequential. Should be coded in order of logic flow.

```
always_comb begin
   b = a;
   c = b;                  // c = a
end
```

Not intended.

```
always_comb begin
   c = b;                  // c = previous value of a
   b = a;                  // changing b does not re-trigger the always_comb in this
                           time step
end
```

'if' Statement

Syntax

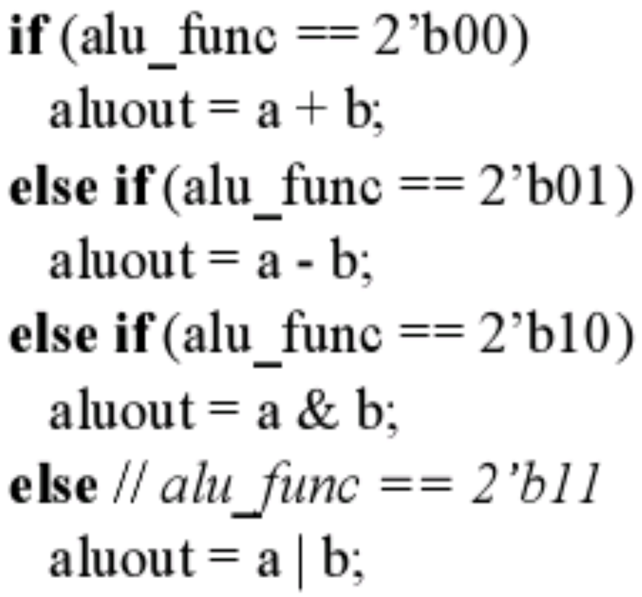
```
 if (expression)
 begin
     ...statements...
 end

 else if (expression)
 begin
   ...statements...
 end
    ...more else if blocks

  else
  begin
    ...statements...
  end
```

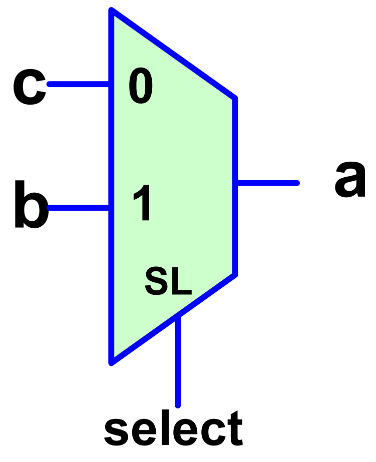
Note: "Begin" and "end" keywords (used if more than one statement in clause)

Example:



```
if (alu_func == 2'b00)
   aluout = a + b;
else if (alu_func == 2'b01)
   aluout = a - b;
else if (alu_func == 2'b10)
   aluout = a & b;
else // alu_func == 2'b11
   aluout = a | b;
```

If Statement Examples



```
always_comb begin
   if(select) a=b;
   else       a=c;
end
```

```
// Nested IF
   statements
always_comb begin
   if(large == 1'b1)
      if (e>f)
         a=e;
       else
         a=f;
   else
      if (e<f)
         a=e;
       else
         a=f;
end
```

```
// Decoder
always_comb begin
   if(s==2'b00)
       a=4'b0001;
   else if(s==2'b01)
       a=4'b0010;
   else if(s==2'b10)
       a=4'b0100;
   else
       a=4'b1000;
end
```

```
// Multiple
   assignments
always_comb begin
   if(select) begin
      a=b;
      z=x;
   end
   else  begin
      a=c;
      z=y;
   end
end
```

Covering All Cases

- For *combinational logic*, you must cover *all* cases when using an if statement
  - Otherwise a "latch" will be inferred

```
//  Bad Example #1
always_comb begin
  if(s==2'b00)
     a=4'b0001;
  else if(s==2'b01)
     a=4'b0010;
  else if(s==2'b10)
     a=4'b0100;
end
```

What happens when s==2'b11?

The synthesis tool needs to "remember" what a used to be. Infers a latch for a – not combinational)

```
//  Bad Example #2
always_comb begin
  if(a<b)
     w=2'b01;
  else if(a>b)
     x=2'b10;
  else
     y=2'b11;
end
```

What is wrong with this?

w, x, and y are not all assigned values under all conditions. Infers latches for w, x, and y.

Covering All Cases

```
//  Bad Example #1
always_comb begin
  if(s==2'b00)
     a=4'b0001;
  else if(s==2'b01)
     a=4'b0010;
  else if(s==2'b10)
     a=4'b0100;
  else
     a=4'b0000;
end
```

Always include an “else” statement. Or a=‘X; to indicate something is wrong such as s is ‘X’ in simulation.

```
//  Bad Example #2
always_comb begin
  w=2'b00;
  x=2'b00;
  y=2'b00;
  if(a<b)
     w=2'b01;
  else if(a>b)
     x=2'b10;
  else
     y=2'b11;
end
```

Assign default values to all signals.

case statements

Syntax

```
case (expression)
  case_choice1:
  begin
    ...statements...
  end

  case_choice2:
  begin
    ...statements...
  end

  ...more case choices blocks...

  default:
  begin
    ...statements...
  end
endcase
```

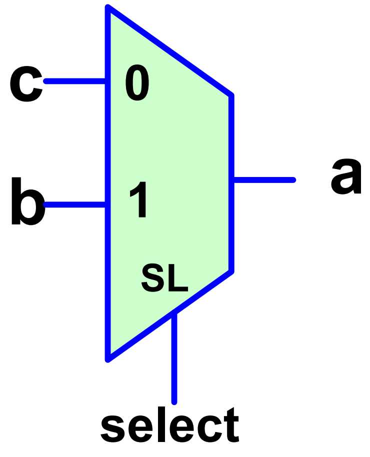
Note: “Begin” and “end” keywords
(used if more than one statement in clause)

Note: endcase keyword

Example:

```
case (alu_ctr)
  2'b00:  aluout = a + b;
  2'b01:  aluout = a - b;
  2'b10:  aluout = a & b;
  default: aluout = 1'bx; // Treated as don't cares for
endcase                   // minimum logic generation.
```

case statement examples



```
case(select)
   1'b1: a=b;
   1'b0: a=c;
endcase
```

```
// Decoder
always_comb begin
  case(select)
   2'b00: a=4'b0001;
   2'b01: a=4'b0010;
   2'b10: a=4'b0100;
   default: a=4'b1000;
    endcase
end
```

```
// Multiple assignments
always_comb begin
 case(select)
  2'b00: begin
          a=4'b0001;
          b=2'b01;
         end
  2'b01: begin
          a=4'b0101;
          b=2'b11;
         end
  2'b10: begin
          a=4'b1111;
          b=2'b00;
         end
  default: begin
          a=4'b0000;
          b=2'b10;
         end
  endcase
end
```

Covering all cases

- Like the 'if' statement, you must cover *all* cases when using a case statement for combinational logic
  - Otherwise a "latch" will be inferred

```
// Bad #1
always_comb begin
   case(s)
    2'b00: a=4'b0001;
    2'b01: a=4'b0010;
    2'b10: a=4'b0100;
   endcase
end
```

What happens when s==2'b11 or 'XX'?

The synthesis tool needs to "remember" what a used to be (infers a latch – not combinational)

```
// Bad #2
always_comb begin
 case(select)
   2'b00:
      a=4'b0001;
   2'b01:
          b=2'b11;
   2'b10:
           c=4'b1111;
   default:
           d=4'b0000;
   endcase
end
```

a, b, and c are not all assigned values under all conditions. Infers latches for a, b, and c.

Covering all cases

```
// Bad #1
always_comb begin
  case(s)
   2'b00: a=4'b0001;
   2'b01: a=4'b0010;
   2'b10: a=4'b0100;
    dafault:
           a=4'b0000;
  endcase
end
```

Always include a 'default' statement. Or default: a='X; to indicate something is wrong such as when s is 'X' in simulation.

```
// Bad #2
always_comb begin
 a=4'b0000;
 b=4'b0000;
 c=4'b0000;
 d=4'b0000;
 case(select)
   2'b00:
      a=4'b0001;
   2'b01:
          b=2'b11;
   2'b10:
           c=4'b1111;
   default:
           d=4'b0000;
   endcase
end
```

Assign default values to all signals.