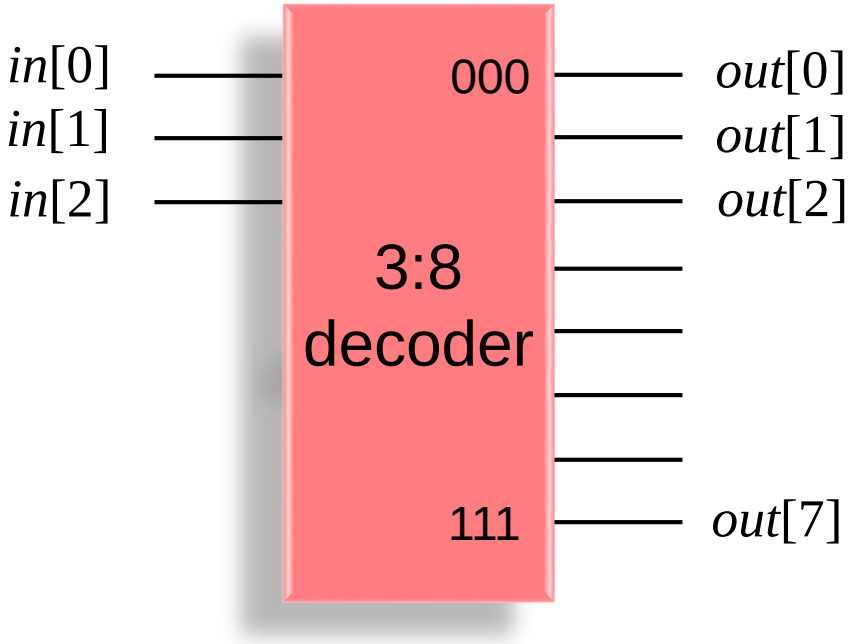
VERILOG 6:

DECODER DESIGN EXAMPLES

Decoder

- A decoder with *i* inputs and fully-populated outputs has 2^i outputs
- It is generally better to work with both the input and output as buses rather than individual signals
- Output is "one-hot"
  - One and only one output is high at a time
- Common uses:
  - Selection of a word within a memory
  - Selection of one module connected to a bus when many modules are connected (e.g., tri-state drivers)



3:8 Decoder
Example 1: Individual wires

- Example 1: A very manual approach à la EEC 180A methods
- Recall that a block with 8 outputs is really 8 distinct designs

```
// in[2:0] may be a wire, reg, or input
wire [7:0] out1;

// Individual wires
assign out1[0] = ~in[2] & ~in[1] & ~in[0];
assign out1[1] = ~in[2] & ~in[1] &  in[0];
assign out1[2] = ~in[2] &  in[1] & ~in[0];
assign out1[3] = ~in[2] &  in[1] &  in[0];
assign out1[4] =  in[2] & ~in[1] & ~in[0];
assign out1[5] =  in[2] & ~in[1] &  in[0];
assign out1[6] =  in[2] &  in[1] & ~in[0];
assign out1[7] =  in[2] &  in[1] &  in[0];
```


ece:/home/bbaas/180b/test.encoder/

3:8 Decoder
Example 2a: case Statement

- Example 2a: Straightforward case statement
- In this and all following examples, *out* is an 8-bit bus

```
// in[2:0] may be a wire, reg, or input
reg [7:0] out2;

// Example 1: case statement
always @(in) begin
  case(in)
    3'b000: begin  out2=8'b00000001;  end
    3'b001: begin  out2=8'b00000010;  end
    3'b010: begin  out2=8'b00000100;  end
    3'b011: begin  out2=8'b00001000;  end
    3'b100: begin  out2=8'b00010000;  end
    3'b101: begin  out2=8'b00100000;  end
    3'b110: begin  out2=8'b01000000;  end
    3'b111: begin  out2=8'b10000000;  end
  endcase
end
```

3:8 Decoder
Example 2b: case Statement

- Example 2b: Straightforward case statement with three independent inputs rather than one 3-bit bus

```
// inputs a, b, c may be a wire, reg, or input
reg [7:0] out2;

// Example 1: case statement
always @(a or b or c) begin
  case({a,b,c})
    3'b000: begin  out2=8'b00000001;  end
    3'b001: begin  out2=8'b00000010;  end
    3'b010: begin  out2=8'b00000100;  end
    3'b011: begin  out2=8'b00001000;  end
    3'b100: begin  out2=8'b00010000;  end
    3'b101: begin  out2=8'b00100000;  end
    3'b110: begin  out2=8'b01000000;  end
    3'b111: begin  out2=8'b10000000;  end
  endcase
end
```

Decoder Test Environment

- These are independent blocks inside the test module

```
//----- Main test loop
// This block is executed once at the
// beginning of the simulation.
initial begin
   $write("Simulation beginning\n");
   #100;
   in = 3'b000;  #100;
   in = 3'b001;  #100;
   in = 3'b010;  #100;
   in = 3'b011;  #100;
   in = 3'b100;  #100;
   in = 3'b101;  #100;
   in = 3'b110;  #100;
   in = 3'b111;  #100;

   $stop;          // ends simulation
end
```

```
reg     [2:0] in;
reg     [7:0] out1;

// example decoder hardware
always @(in) begin
  case(in)
    3'b000: begin  out1=8'b00000001;  end
    ...
  endcase
end
```

```
//----- Print statements
always @(in) begin
  #10;   // gives a tiny delay after "in" changes
  $write("in = %b, out1 = %b\n", in, out1);
end
```

```
Simulation beginning
in = 000, out1 = 00000001
in = 001, out1 = 00000010
in = 010, out1 = 00000100
in = 011, out1 = 00001000
in = 100, out1 = 00010000
in = 101, out1 = 00100000
in = 110, out1 = 01000000
in = 111, out1 = 10000000
```

3:8 Decoder
Example 3: Shift Left Operator

- Example 3: Treat the 8 outputs as a single reg bus

```
// in[2:0] may be a wire, reg, or input
reg [7:0] out3;

// Example 3: reg with <<
always @(in) begin
  out3 = 8'b0000_0001 << in;
end
```

3:8 Decoder
Example 4: Shift Left Operator

- Example 4: Treat the 8 outputs as a single wire bus

```
// in[2:0] may be a wire, reg, or input
wire [7:0] out4;

// Example 4: wire with <<
assign out4 = 8'b0000_0001 << in;
```

3:8 Decoder, Partially-Defined Output Cases; Example 5

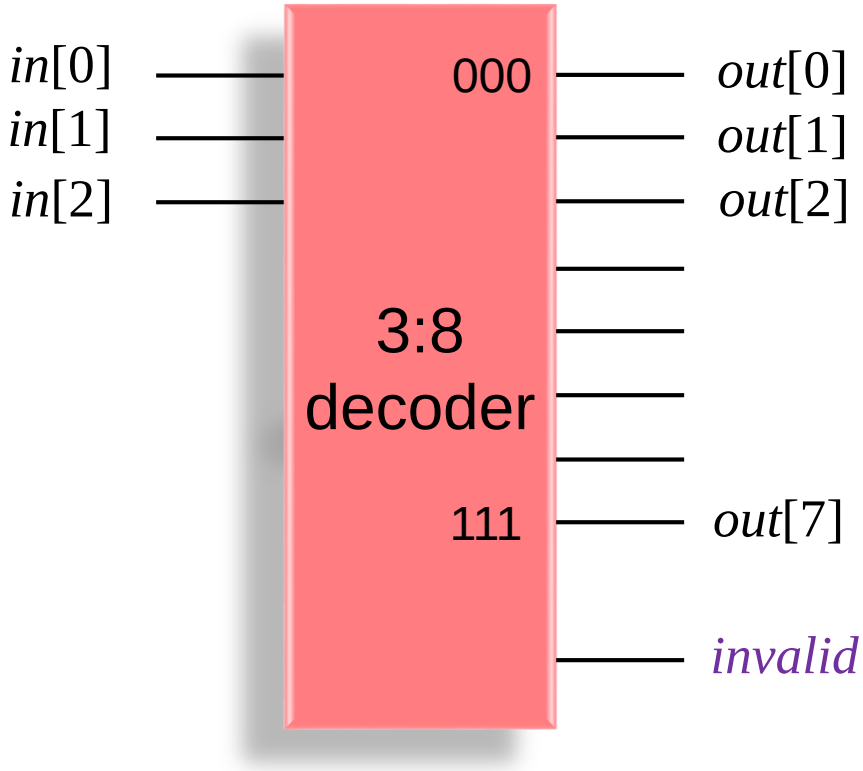
- Example 5: Only 4 of the input combinations are defined: 000, 001, 100, 110
- Choose invalid inputs to have "x" output (trivial change to set outputs to zero instead)
- Could also implement by:
  1) setting default at beginning of always block
  2) setting default then using "if" statements
  3) many other options

```
reg [7:0] out5;

// Example 5: partially-defined outputs
always @(in) begin
  case(in)
    3'b000: begin  out5=8'b00000001;  end
    3'b001: begin  out5=8'b00000010;  end
    //
    //
    3'b100: begin  out5=8'b00010000;  end
    //
    3'b110: begin  out5=8'b01000000;  end
    //
    default:begin  out5=8'bxxxxxxxx;  end
  endcase
end
```

Decoder With Additional *invalid* Output Signal

- The next example is for a 3:8 decoder that has the same valid input combinations: 000, 001, 100, 110
- But in this example when the input does not have a valid value:
  - The eight main outputs must be all zeros
  - A new *invalid* output signal is set high



3:8 Decoder, *invalid* Output Example 6

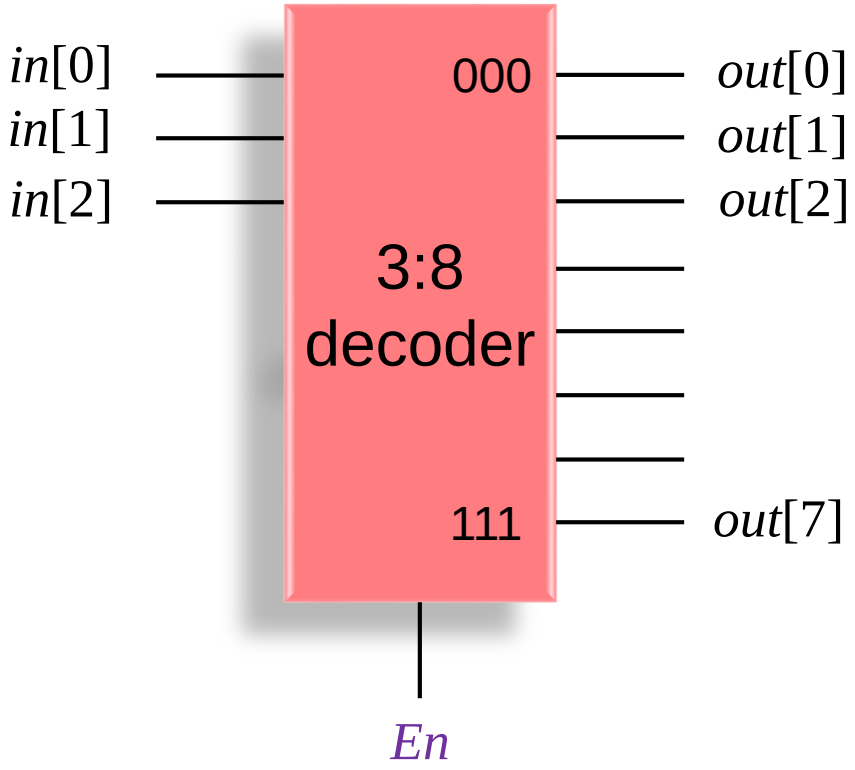
- Example 6: Same as previous example except:
  - Undefined cases have zero output
  - *invalid* signal set when input equals a value for which the outputs are undefined

```
reg [7:0] out6;

// Example 6: invalid output signal
always @(in) begin
  invalid = 1'b0;
  case(in)
    3'b000: begin  out6=8'b00000001;  end
    3'b001: begin  out6=8'b00000010;  end
    //
    //
    3'b100: begin  out6=8'b00010000;  end
    //
    3'b110: begin  out6=8'b01000000;  end
    //
    default: begin
       out6    = 8'b0000_0000;
       invalid = 1'b1;
    end
  endcase
end
```

Decoder With Enabled Output

- Decoders frequently have an *Enable* input which controls the overall state of the outputs
  - When input *En* is high, it functions as a normal decoder
  - When input *En* is low, all outputs are low



3:8 Decoder, Enabled Output Example 7

- Example 7: Enabled output:
  - When input *En* is high, it functions as a normal decoder
  - When input *En* is low, all outputs are low
- Many implementations are possible

```
reg [7:0] out7;

// Example 7: Enabled output version 1
always @(in) begin
  // enabled, function like a normal encoder
  if (En == 1'b1) begin
    case(in)
      3'b000: begin  out7=8'b00000001;  end
      3'b001: begin  out7=8'b00000010;  end
      3'b010: begin  out7=8'b00000100;  end
      3'b011: begin  out7=8'b00001000;  end
      3'b100: begin  out7=8'b00010000;  end
      3'b101: begin  out7=8'b00100000;  end
      3'b110: begin  out7=8'b01000000;  end
      3'b111: begin  out7=8'b10000000;  end
    endcase
  end

  // not enabled, set all outputs to zero
  else begin
    out7=8'b00000000;
  end
end
```

3:8 Decoder, Enabled Output Example 8

- Example 8: Enabled output:
  - When input *En* is high, it functions as a normal decoder
  - When input *En* is low, all outputs are low
- Many implementations are possible

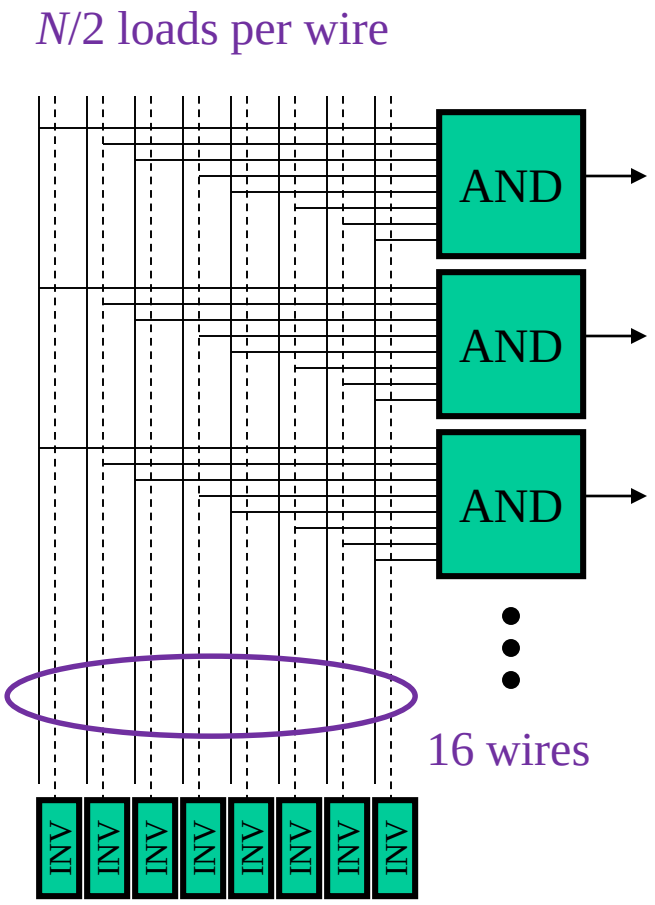
```
reg [7:0] out8;

// Example 8: Enabled output version 2
always @(in) begin
  // default. Initialize output as if not enabled.
  out8=8'b00000000;

  // if enabled, function like a normal encoder
  if (En == 1'b1) begin
    case(in)
      3'b000: begin  out8=8'b00000001;  end
      3'b001: begin  out8=8'b00000010;  end
      3'b010: begin  out8=8'b00000100;  end
      3'b011: begin  out8=8'b00001000;  end
      3'b100: begin  out8=8'b00010000;  end
      3'b101: begin  out8=8'b00100000;  end
      3'b110: begin  out8=8'b01000000;  end
      3'b111: begin  out8=8'b10000000;  end
    endcase
  end
end
```

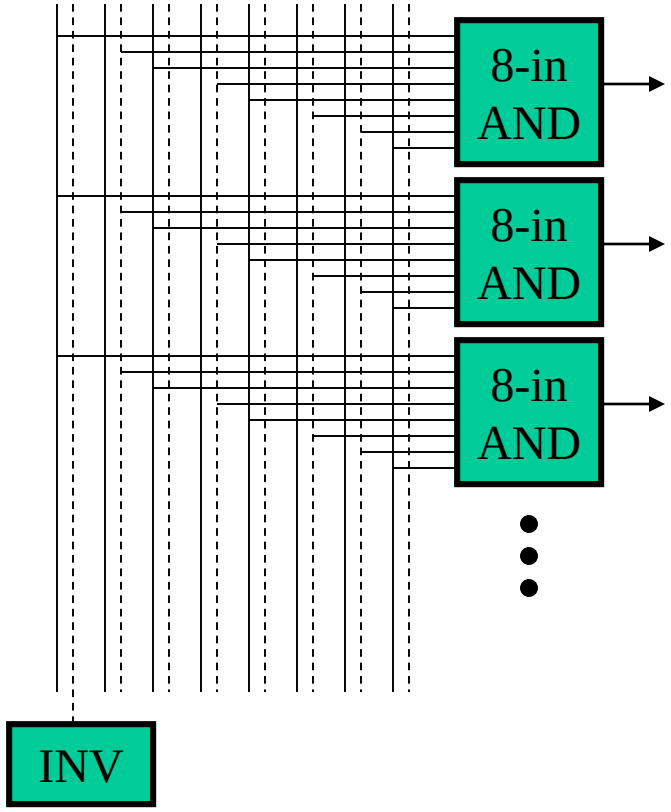
Decoder "Manually" Designed For Example: Memory Array Design

- Example: 256-output (8 input bits)
  - Straightforward approach
    - Route 8 input wires plus inverted versions (16 wires) along array
    - Each word uses an 8-input AND gate
    - Each long wire has $N/2$=128 gate loads



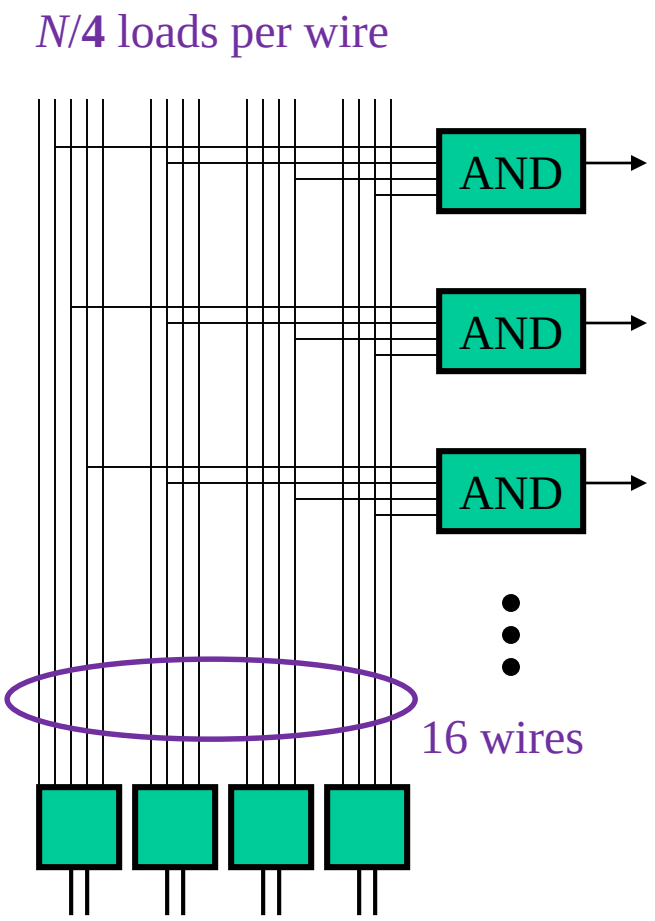
Decoder Design

- Critical path
  - INV + 8-input AND
  - Neglect assertion level because we'll have to add a number of inverters (buffers) anyway
  - Building an 8-input AND:
    - AND-AND (inefficient)
    - NAND-NOR (better)
  - Total critical path
    - INV + 4-input NAND + 2-input NOR, or INV + 3-input NAND + 3-input NOR



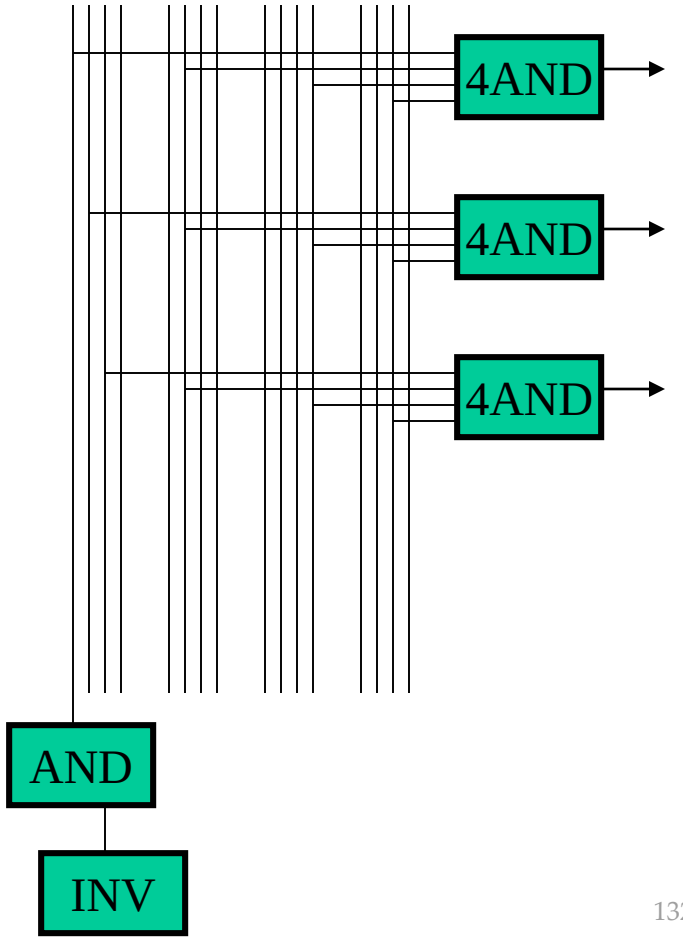
Decoder Design Using Predecoding

- Example: 256-output (8 input bits) with predecode
  - Ex: take groups of 2 input bits
    - Four possibilities; activate one of four wires (use 2 INVs, 4 ANDs)
  - Wires along array: 2^2 * (8/2) = 4 groups of 4 = 16 (same as non-predecoded)
  - Each output uses a 4-input AND gate (much faster)
  - Each long wire has *N*/4=64 gate loads (half of other approach!)
  - Predecoding works best with large decoders
    - May have less toggling, be faster



Decoder Design Using Predecoding

- Critical path
  - Predecoder + 4-input AND
  - Neglect assertion level because we'll have to add a number of inverters (buffers) anyway
  - INV + 2-input AND + 4-input AND
    - May be able to use NANDs instead of ANDs



Decoder Design Using Predecoding

- Example: predecode groups of 3 address bits
  - Predecoding groups: 3 + 3 + 2 for the same 8:256 decoder
  - Each 3-input predecode group has 2^3 = 8 output wires
  - Each 3-input predecoded wire has *N*/8 loads
  - Total of 8 + 8 + 4 = 20 predecoded wires
- Example: predecode groups of 4 address bits
  - Predecoding groups: 4 + 4 for the same 8:256 decoder
  - Each predecode group has 2^4 = 16 output wires
  - Each predecoded wire has *N*/16 loads
  - Total of 16 + 16 = 32 predecoded wires