`for` Statement

- Provides an indexed looping expression
- Must be used inside `always` or `initial` statement
- The `for` loop is structured just as in C.
- For loops must be used carefully when constructing combo logic.
- Non-synthesizible loops are easy to create.
- For loops are seldom used in synthesizible code.

for Statement

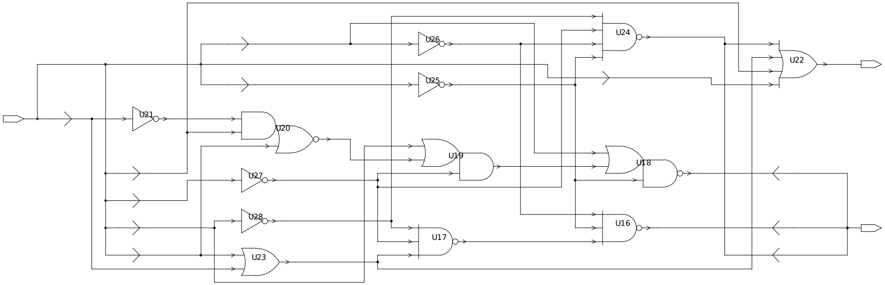
- The priority encoder looks at input bus a and outputs the encoding of the highest bit that is set on bus y.
- Input value 0x00 is illegal. The valid bit detects this case.
- Could the valid bit be used for a timing (hardware) indication that the output is ready?

```
module priority_encoder (
  input         [7:0]   a,
  output  reg           valid, //indicates output valid
  output  reg   [2:0]   y);

  integer i;
  always_comb begin
    valid = 0;  //default output
    y = 3'bx;   //default output
    for (i=0; i<8; i++) begin
      if (a[i]) begin
        valid = 1; y = i;
      end //if
    end   //for
  end     //always_comb
endmodule
```

for Statement

Output from synthesis for priority_encoder.sv



for Statement

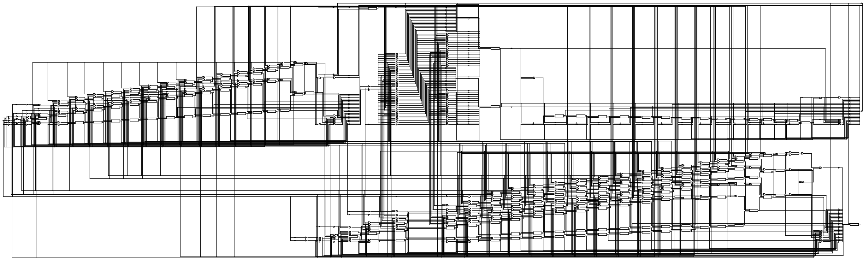
- Here's a neat bit of code using a for loop. What's it do?

```
module whatisit(
  input        [15:0]  a,   //data in a
  input        [15:0]  b,   //data in b
  output reg [31:0]  z     //data output
  );

  integer i;
  always_comb begin
    z=0;
    for(i=0;i<=15;i++) begin
      if (a[i]) begin
        z = z + (b<<i);
      end
    end
  end
endmodule
```

for Statement

Synthesis output from whatisit.sv
1700 gates, 6.5nS worst case delay b[1] to z[31]



`for` Statement

- Note that in both cases, the sequential behavioral description using `for` DOES NOT produce a sequential implementation.
- The operation of the gates is produced via concurrent evaluation.