Lecture 3: HDL II

ECE 228
Mostafizur Rahman
rahmanmo@umkc.edu

Critical Thinking Exercise

Dear Neighbor,

You and I share a passion about dogs. I am especially fond of German Shepherd. When I see you playing with your dog, I feel jealous!

I wanted to tell you that a section of Your fence is broken. Your dog often sneaks in and create a mess. The other day he came in my deck, it was a bit scary. I know you will be addressing the fence situation soon. Thanks!

Ross

Anything Unusual?

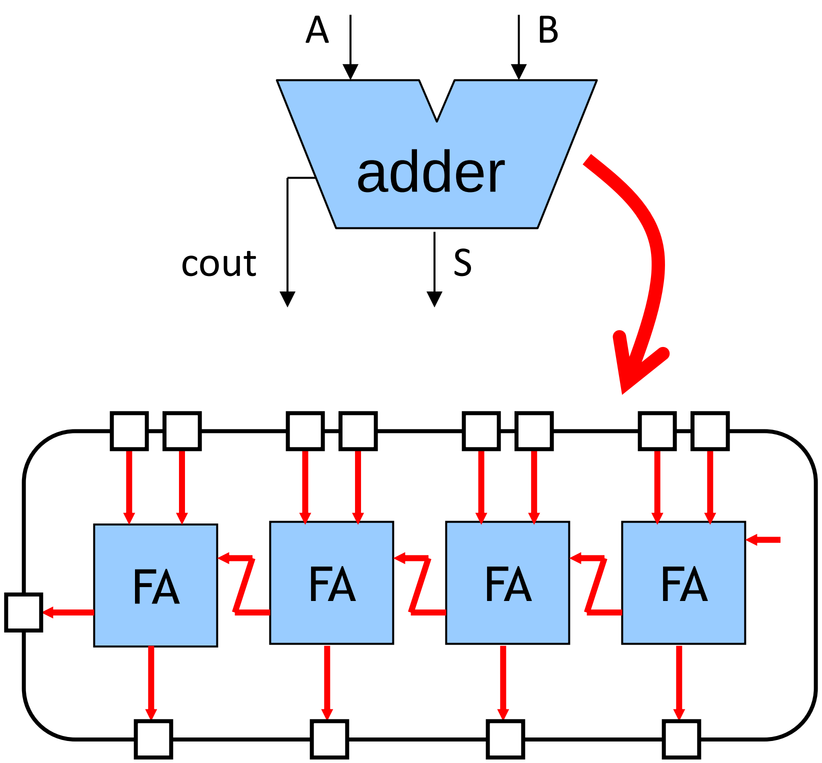
Traditional computer architecture has fundamental limitations which stem from the memory and computing separation. An architecture that merges the memory and computing gap has the potential to revolutionize next generation electronics. Here we present an approach to computing with multi-valued logic. Inherently, the computing as we perceive it, is multi-valued. For traditional computing, data conversion is needed to go from decimal to binary and hence, it is efficient. Our approach for multi-valued computing solves these problems and achieves huge efficiencies.

Anything unusual?

```
module build_xor (a, b, c);
 input a, b;
 output c;

// wire declaration to meet internal wiring needs
     wire c, a_not, b_not;
     not a_inv (a_not, a);
     and a1 (x, a_not, b);
     and a2 (y, b_not, a);
     or out (c, x, y);
     not b_inv (b_not, b);

endmodule
```

Hierarchical Modeling with Verilog

- **A module can contain other modules through module instantiation creating a module hierarchy**
  – Modules are connected together with nets
  – Ports are attached to nets either by position or by name




```
module adder( input  [3:0] A, B,
              output       cout,
              output [3:0] S );

  wire c0, c1, c2;
  FA fa0( .a(A[0]), .b(B[0]),
          .cin(0), .cout(c0),
          .sum(S[0] );

  FA fa1( .a(A[1]), .b(B[1]),
          ...

endmodule
```

Activity

- Design a 2-bit subtractor unit. Inputs are A and B, and output is S, 2-bits each. As a sub-module you can call the Full_Adder module which takes in 3 inputs (X, Y and Z) and produces 1 output (P).

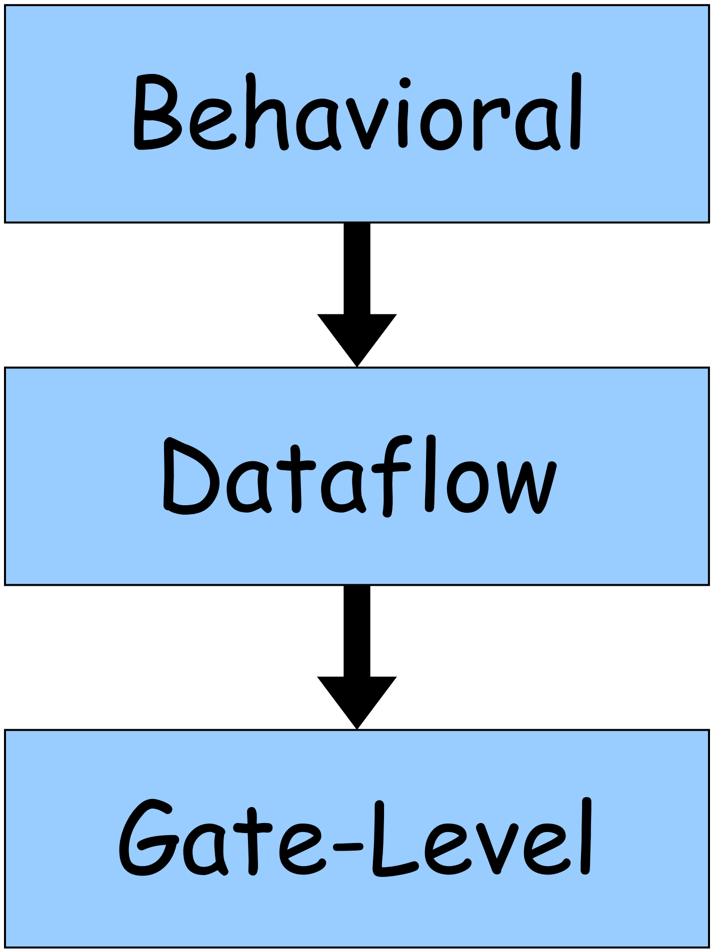
  Write structural HDL

Using parameters

- Constants: Declared using the keyword **parameter**
- The use of parameters make code easy to read and modify
  - **parameter** byte_size = 8 ; // integer
- Example:

```
……
parameter  bussize = 8;
reg  [bussize-1 : 0]  databus1;
reg  [bussize-1 : 0]  databus2;
……
```

3 Common Abstraction Levels

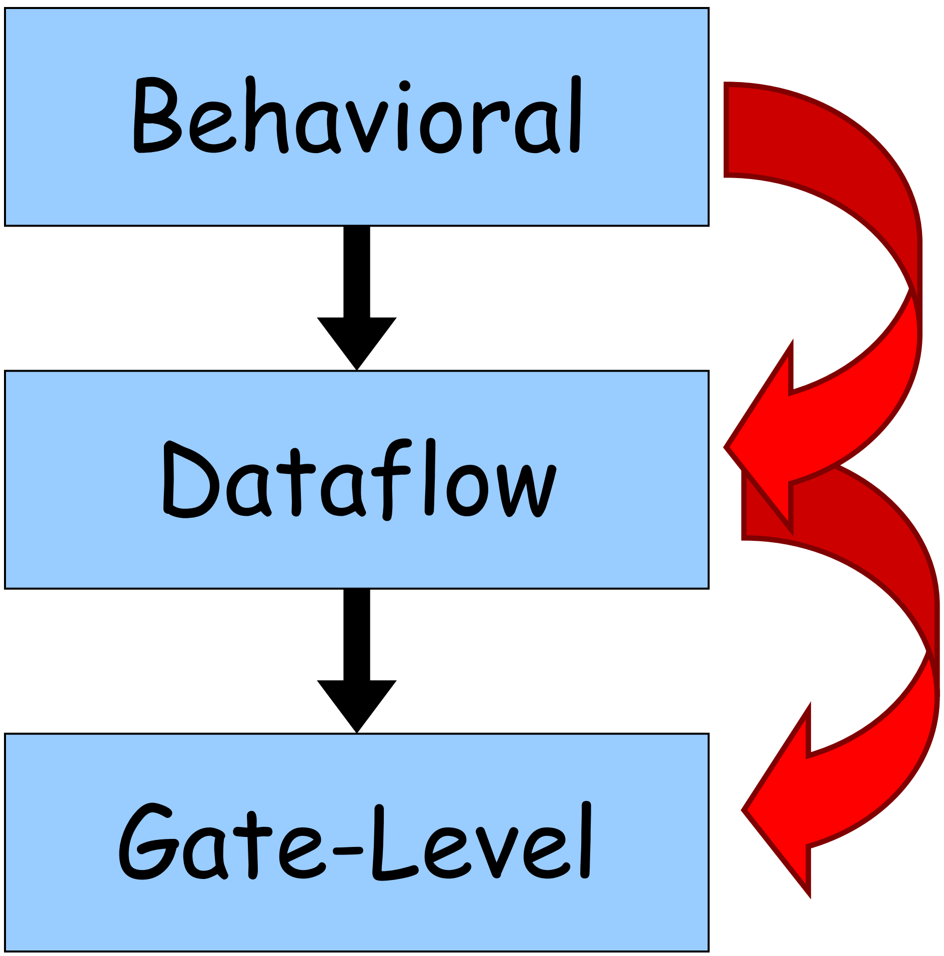


Module's high-level algorithm is implemented with little concern for the actual hardware

Module is implemented by specifying how data flows between registers

Module is implemented in terms of concrete logic gates (AND, OR, NOT) and their interconnections

3 Common Abstraction Levels



Designers can create lower-level models from the higher-level models either manually or automatically

The process of automatically generating a gate-level model from either a dataflow or a behavioral model is called

Logic Synthesis

Continuous Assignment

- Identified by the keyword “**assign**”

  assign   a= b & c;
- Form a static binding between
  - The ‘net’ being assigned on the LHS,
  - The expression on the RHS
- The assignment is continuously active, with respect to the simulation
- Used to model combinational logic
- For an “assign” statement:
  - The expression on RHS may contain both “register” or “net” type variables
  - The LHS must be of “net” type, typically a “wire”

Dataflow Modeling

- Uses continuous assignment statement
  - Format: assign [delay] net = expression;
  - Example: assign sum = a ^ b;
- **Delay**: Time duration between assignment from RHS to LHS
- All continuous assignment statements execute concurrently
- Delay can be introduced
  - Example: assign #2 sum = a ^ b;
  - "#2" indicates 2 time-units
  - No delay specified : 0 (default)
- Associate time-unit with physical time
  - `timescale time-unit/time-precision
  - Example: `timescale 1ns/100 ps

Net Declaration Assignments

- Only one net declaration assignment per net

```
wire out;                      // regular continuous assignment
assign out = in1 & in2;
```

An implicit continuous assignment combines the net declaration with an assign statement

```
wire  out = in1 & in2;    // net declaration assignment
```

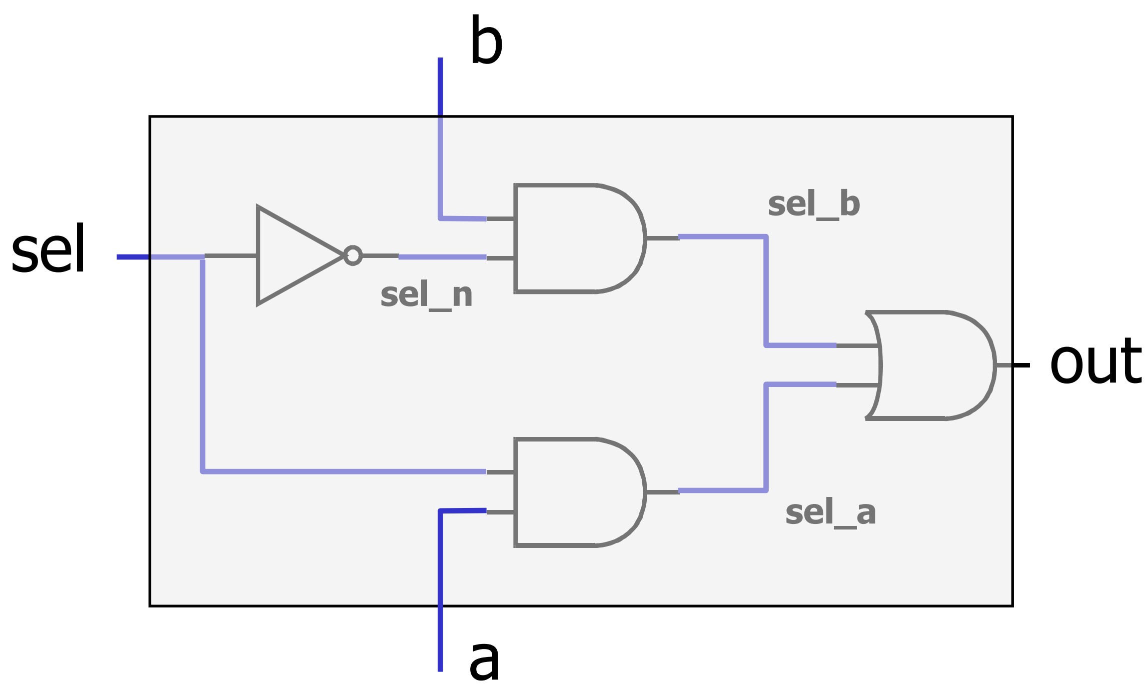
- Implicit Net Declarations

```
wire in1, in2;
assign out = in1 & in2;
```

```
// net declaration assignment delay
wire #10 out = in1 & in2;

// regular assignment delay
wire out;
assign #10 out = in1 & in2;
```

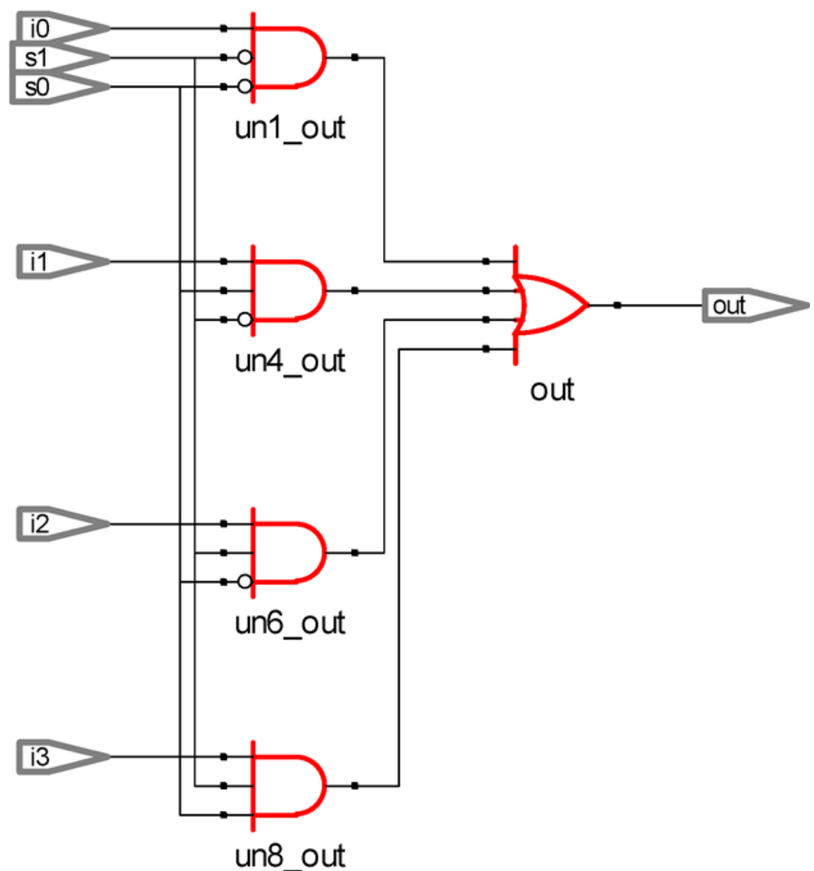
Dataflow

- **Dataflow**: Specify output signals in terms of input signals
- Example:
  assign out = (sel & a) | (~sel & b);



An Example - A 4-to-1 MUX

```
// using basic and, or , not logic operators
assign out =    (~s1 & ~s0 & i0) |
                (~s1 & s0 & i1)  |
                (s1 & ~s0 & i2)  |
                (s1 & s0 & i3) ;
```



Activity

- You are required to develop a chip that will have 3 inputs (i.e., A, B and C) and 1 output (S). The relationship between the output and inputs is-

If (A or B or C == 1) S = A + B + C

Else S = 0

Write the structural level VHDL module definition.

Dataflow : Key Points

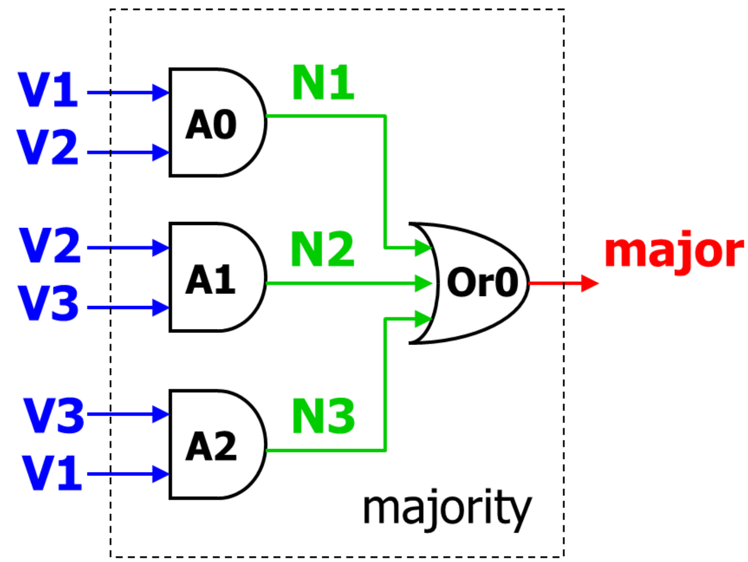
- Dataflow modeling enables the designer to focus on where the state is in the design and how the data flows between these state elements without becoming bogged down in gate-level details
  - Continuous assignments are used to connect combinational logic to nets and ports
  - A wide variety of operators are available including:

```
Arithmetic:      + - * / % **
Logical:         ! && ||
Relational:      > < >= <=
Equality:        == != === !===
Bitwise:         ~ & | ^ ^~
Reduction:       & ~& | ~| ^ ^~
Shift:           >> << >>> <<<
Concatenation:   { }
Conditional:     ?:
```

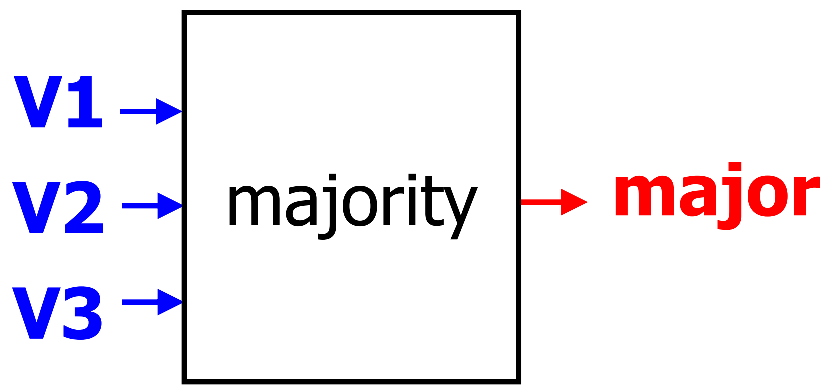
```
assign signal[3:0]
          = { a, b, 2'b00 }
```

Dataflow Example: Majority Detector

- Use continuous assignment statements to assign Boolean expressions to signals.
- If an input value changes, the value of the assignment is immediately updated. This is combinational *hardware*, not *software*.

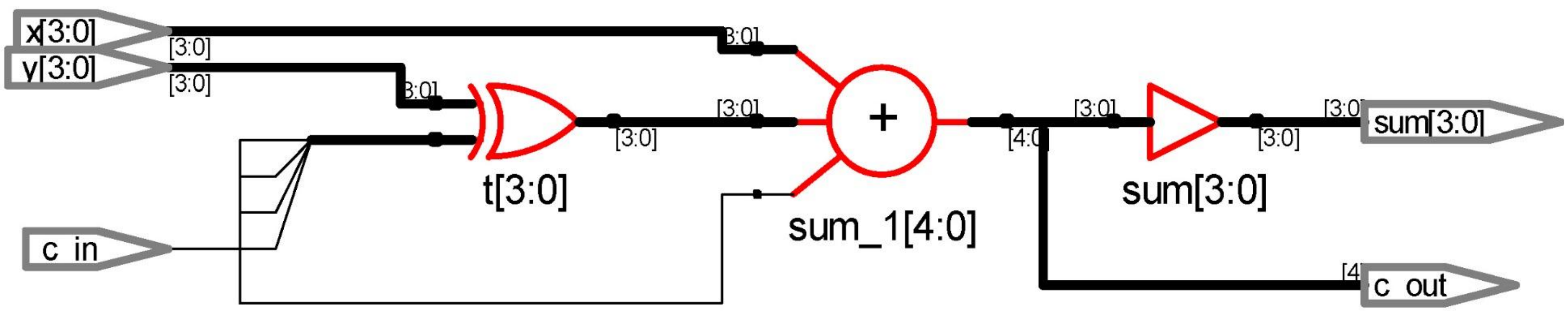



```
module majority (major, V1, V2, V3) ;

output major ;
input V1, V2, V3 ;

assign major = (V1 & V2)
             | (V2 & V3)
             | (V1 & V3);
endmodule
```



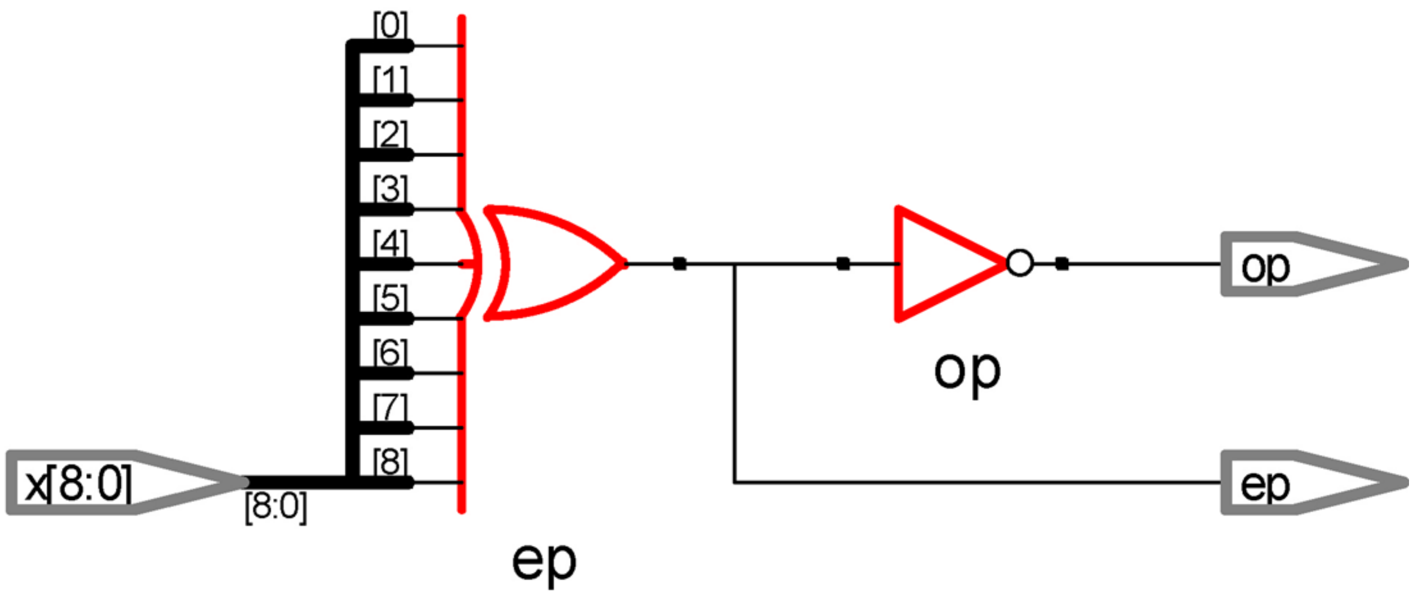
A 4-Bit Two's Complement Adder

```
// specify the function of a two's complement adder
   assign t = y ^ {4{c_in}};
   assign {c_out, sum} = x + t + c_in;
```



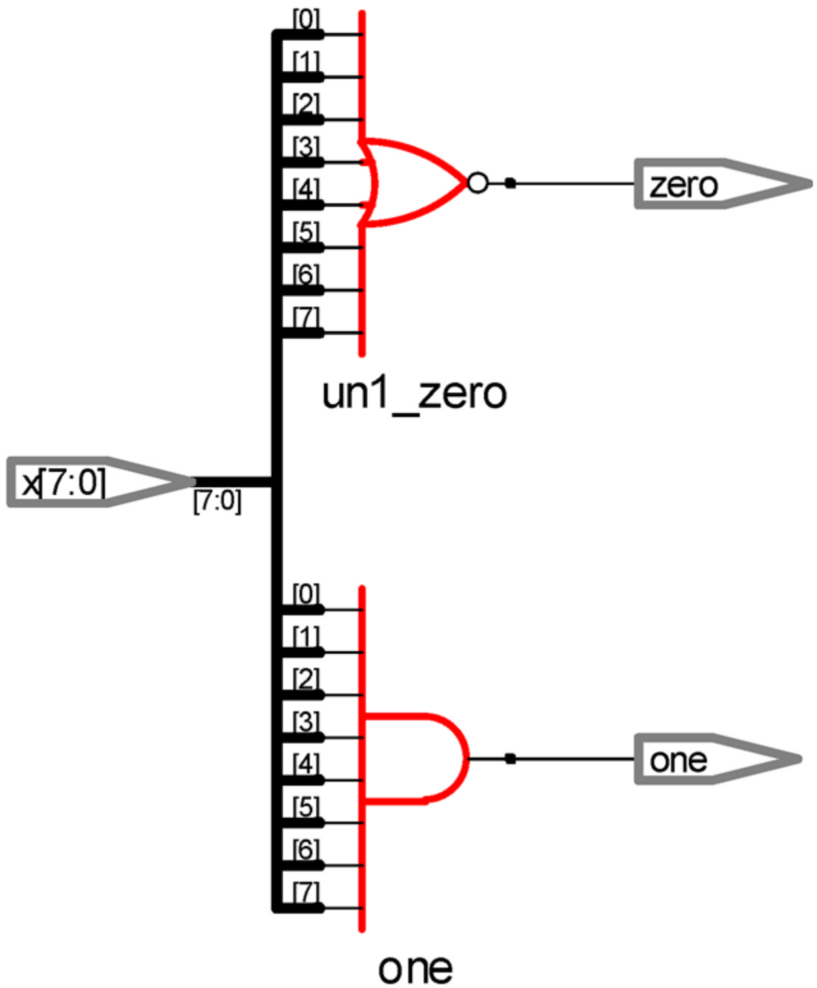
A 9-Bit Parity Generator

```
// dataflow modeling using reduction operator
assign ep = ^x;    // even parity generator
assign op = ~ep;  // odd parity generator
```

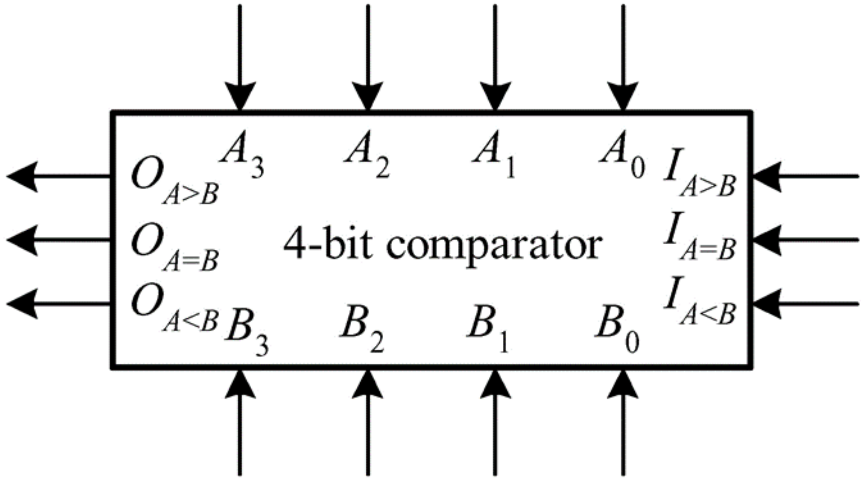


An All-Bit-Zero/One Detector

```
// dataflow modeling
   assign zero = ~(|x);  // all-bit zero
   assign one = &x;      // all-bit one
```



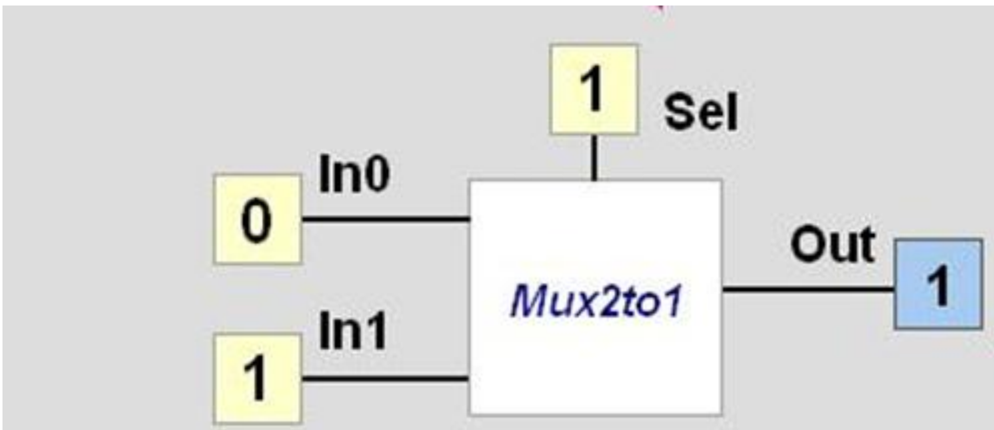
A 4-B Magnitude Comparator




```
// dataflow modeling using relation operators
   assign Oaeqb = (A == B) && (Iaeqb == 1);                  //  =
   assign Oagtb  = (A > B) || ((A == B)&& (Iagtb == 1));  // >
   assign Oaltb   = (A < B) || ((A == B)&& (Ialtb == 1));   // <
```

2-to-1 MUX – Continuous Assignment

- A conditional assignment has three signals
  - The first signal is the control signal
  - If the control signal is true, the second signal is assigned to the left hand side (LHS) signal ; otherwise, the third signal is assigned to LHS signal.

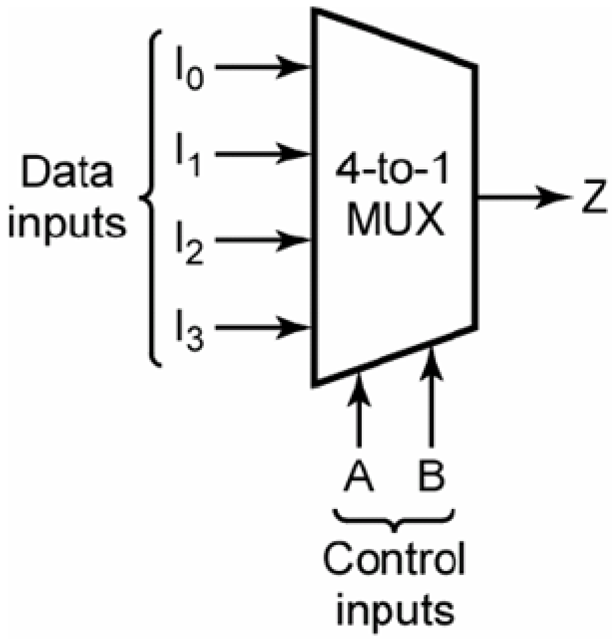



```
module Mux2to1(Out, In0, In1, Sel);
output Out;
input In0, In1, Sel;

assign Out = Sel ? In1 : In0;

endmodule
```

An Example - A 4-to-1 MUX

```
// using conditional operator (?:)
assign Z = A ? ( B ? I3 : I2) : (B ? I1 : I0) ;
```

$$assign\ Z = A\ ?\ (B\ ?\ I_3 : I_2) : (B\ ?\ I_1 : I_0)\ ;$$



$$Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$