

EECS 270 W25

Discussion 6

February 13th, 2025
By: Mick Gordinier



AGENDA

1. Announcements
2. Building Large Multiplexers Practice
3. More Combinational Block Building
4. Procedural Statements Introduction
5. Project 4 Discussion

Announcements

Discussion Exam Review Next Week!

- Exam 1: Monday February 24th
- Professor Review Session
 - Monday February 24th (During Lecture)
- **Discussion Additional Review Session**
 - **Thursday February 20th (During Discussion)**

Verilog Linter for Windows/Linux

Walkthrough Steps for Installation

```
module testing(  
    input [2:0] A,  
    output [1:0] B  
);  
    extra digits given for sized binary constant. Icarus Verilog(iverilog)  
    Numeric constant truncated to 2 bits. Icarus Verilog(iverilog)  
    View Problem (Alt+F8) No quick fixes available  
    assign B = 2'b100;  
endmodule
```

```
module testing(  
    input [2:0] A,  
    Superfluous comma in port declaration list. Icarus Verilog(iverilog)  
    View Problem (Alt+F8) No quick fixes available  
    output [1:0] B,  
);
```

```
module TopLevel(  
    input [2:0] SW,  
    output [6:0] HEX  
);  
    Instantiation of module testing requires an instance name. Icarus Verilog(iverilog)  
    module testing(  
        View Problem (Alt+F8) No quick fixes available  
        testing (SW, HEX[1:0]);  
endmodule
```

```
module TopLevel(  
    input [2:0] SW,  
    output [6:0] HEX  
);  
    Wrong number of ports. Expecting 2, got 1. Icarus Verilog(iverilog)  
    View Problem (Alt+F8) No quick fixes available  
    testing a(HEX[1:0]);  
endmodule
```

```
module testing(  
    input [2:0] A,  
    output B  
);  
    implicit definition of wire 'b'. Icarus Verilog(iverilog)  
    View Problem (Alt+F8) No quick fixes available  
    assign b = A[2] & A[1] & A[0];  
endmodule
```

Boolean Algebra Practice (Good Midterm Review)

Practice Problems and Solution Walkthrough

- | | |
|--|---|
| 1) $a + 0 =$ _____ | 14) $y + y\bar{y} =$ _____ |
| 2) $\bar{a} \cdot 0 =$ _____ | 15) $xy + x\bar{y} =$ _____ |
| 3) $a + \bar{a} =$ _____ | 16) $\bar{x} + y\bar{x} =$ _____ |
| 4) $a + a =$ _____ | 17) $(w + \bar{x} + y + \bar{z})y =$ _____ |
| 5) $a + ab =$ _____ | 18) $(x + \bar{y})(x + y) =$ _____ |
| 6) $a + \bar{a}b =$ _____ | 19) $w + [w + (wx)] =$ _____ |
| 7) $a(\bar{a} + b) =$ _____ | 20) $x[x + (xy)] =$ _____ |
| 8) $ab + \bar{a}b =$ _____ | 21) $\overline{(\bar{x} + \bar{x})} =$ _____ |
| 9) $(\bar{a} + \bar{b})(\bar{a} + b) =$ _____ | 22) $(x + \bar{x}) =$ _____ |
| 10) $a(a + b + c + \dots) =$ _____ | 23) $w + (w\bar{x}yz) =$ _____ |
| For (11), (12), (13), $f(a, b, c) = a + b + c$ | 24) $\bar{w} \cdot \overline{(wxyz)} =$ _____ |
| 11) $f(a, b, ab) =$ _____ | 25) $xz + \bar{x}y + zy =$ _____ |
| 12) $f(a, b, \bar{a} \cdot \bar{b}) =$ _____ | 26) $(x + z)(\bar{x} + y)(z + y) =$ _____ |
| 13) $f[a, b, \overline{(ab)}] =$ _____ | 27) $\bar{x} + \bar{y} + xy\bar{z} =$ _____ |

Discussion 5 Walkthrough of Buggy Code:

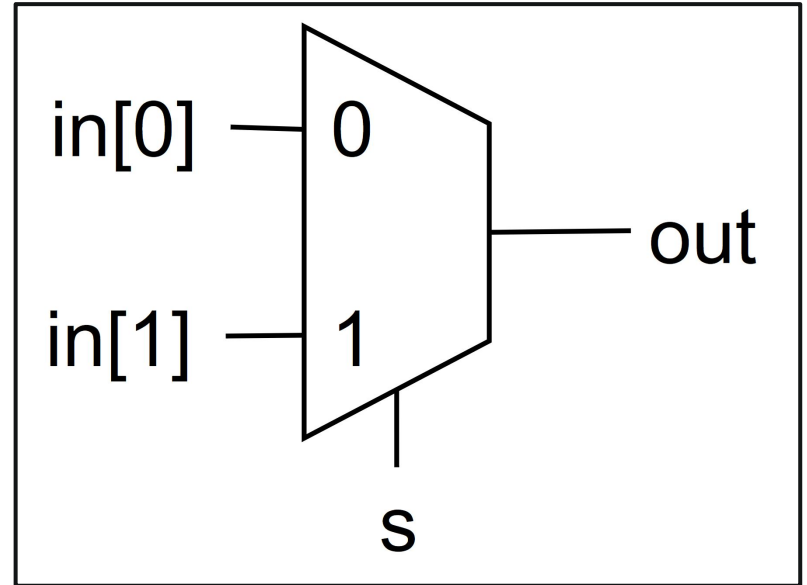
[Link to Doc](#)

Building Large Multiplexers

Build a 2-1 MUX

```
// If sel = 0 --> out = in[0]
// If sel = 1 --> out = in[1]
module mux21(
    input sel,
    input [1:0] in,
    output out
);

endmodule
```



Build a 2-1 MUX (Textual)

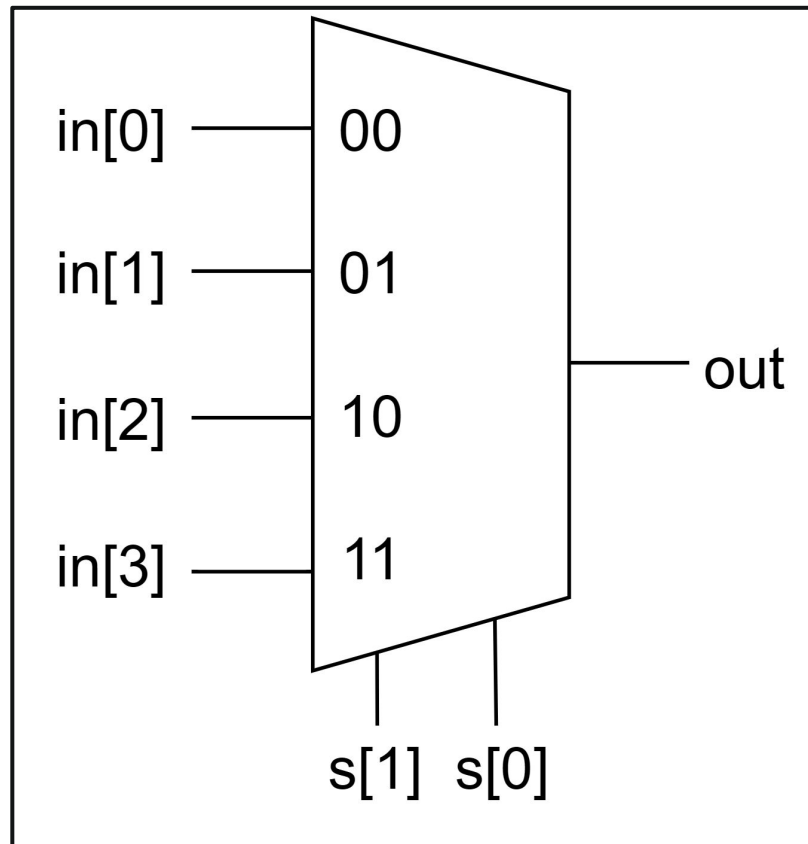
```
module TopLevel21(  
  input [1:0] SW,  
  input [3:3] KEY,  
  output [0:0] LEDR  
);  
  
  // Inverting the keys to become active low  
  mux21 m(.sel(~KEY[3]), .in(SW[1:0]), .out(LED[0]));  
endmodule  
  
// If sel = 0 --> out = in[0]  
// If sel = 1 --> out = in[1]  
module mux21(  
  input sel,  
  input [1:0] in,  
  output out  
);  
  // Finish here  
endmodule
```

Build a 4-1 MUX (Using 2-1 MUXes)

```
// If sel = 00 --> out = in[0]
// If sel = 01 --> out = in[1]
// If sel = 10 --> out = in[2]
// If sel = 11 --> out = in[3]
```

```
module mux41(
  input [1:0] sel,
  input [3:0] in,
  output out
);
```

```
endmodule
```



Build a 4-1 MUX (Textual)

```
module TopLevel41(  
    input [3:0] SW,  
    input [3:2] KEY,  
    output [0:0] LEDR  
);  
  
    // Inverting the keys to become active low  
    mux41 m(.sel(~KEY[3:2]), .in(SW[3:0]), .out(LED[0]));  
endmodule  
  
// If sel = 00 --> out = in[0]  
// If sel = 01 --> out = in[1]  
// If sel = 10 --> out = in[2]  
// If sel = 11 --> out = in[3]  
module mux41(  
    input [1:0] sel,  
    input [3:0] in,  
    output out  
);  
    // Finish here  
endmodule
```

Build a 4-1 MUX (Using 2-1 MUX) (SOLUTION)

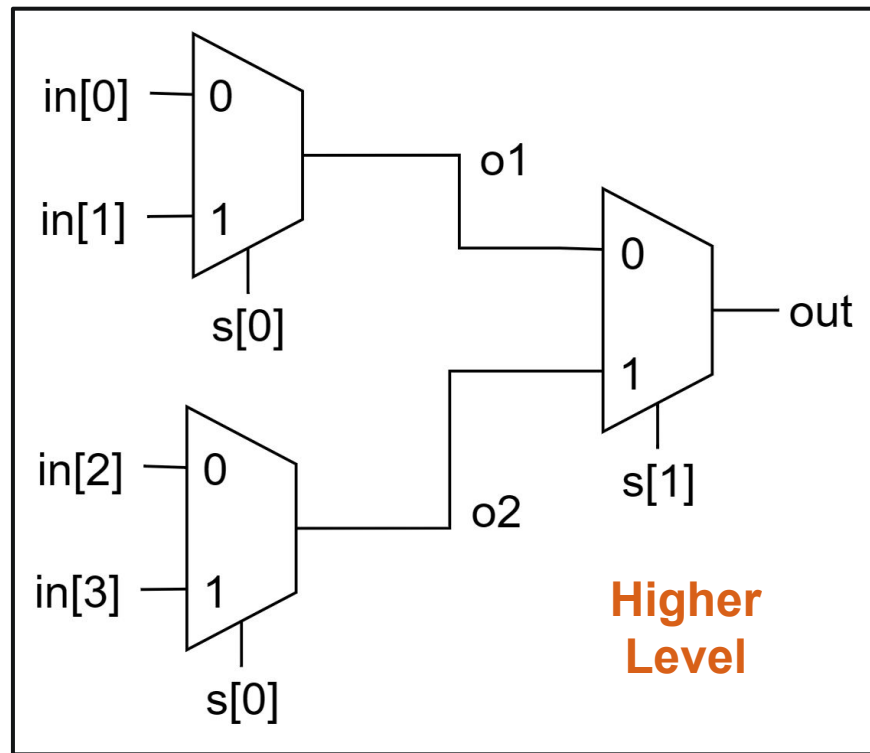
```
// If sel = 00 --> out = in[0]
// If sel = 01 --> out = in[1]
// If sel = 10 --> out = in[2]
// If sel = 11 --> out = in[3]
module mux41(
    input [1:0] sel,
    input [3:0] in,
    output out
);

    wire o1, o2;

    // Lower-level muxes
    mux21 m1(.sel(sel[0]), .in(in[1:0]), .out(o1));
    mux21 m2(.sel(sel[0]), .in(in[3:2]), .out(o2));

    // Higher-level mux
    // Concatenating o1 and o2 into a 2-bit wire
    mux21 m3(.sel(sel[1]), .in({o2, o1}), .out(out));

endmodule
```



Lower Level

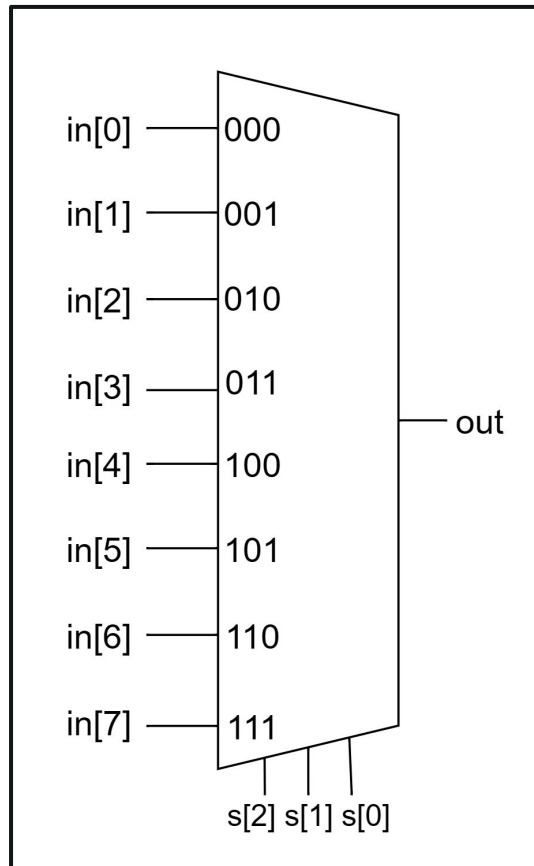
Higher
Level

Build a 8-1 MUX (Using 2-1 MUX)

```
// If sel = 000 --> out = in[0]
// If sel = 001 --> out = in[1]
// If sel = 010 --> out = in[2]
// If sel = 011 --> out = in[3]
// If sel = 100 --> out = in[4]
// If sel = 101 --> out = in[5]
// If sel = 110 --> out = in[6]
// If sel = 111 --> out = in[7]
```

```
module mux81(
  input [2:0] sel,
  input [7:0] in,
  output out
);
```

```
endmodule
```



Build a 8-1 MUX (Textual)

```
module TopLevel81(  
    input [7:0] SW,  
    input [3:1] KEY,  
    output [0:0] LEDR  
);  
  
    // Inverting the keys to become active low  
    mux81 m(.sel(~KEY[3:1]), .in(SW[7:0]), .out(LEDR[0]));  
endmodule  
  
// If sel = 000 --> out = in[0]  
// If sel = 001 --> out = in[1]  
// If sel = 010 --> out = in[2]  
// If sel = 011 --> out = in[3]  
// If sel = 100 --> out = in[4]  
// If sel = 101 --> out = in[5]  
// If sel = 110 --> out = in[6]  
// If sel = 111 --> out = in[7]  
module mux81(  
    input [2:0] sel,  
    input [7:0] in,  
    output out  
);  
    // Finish here  
endmodule
```

Build a 8-1 MUX (Using 2-1 MUX) (SOLUTION)

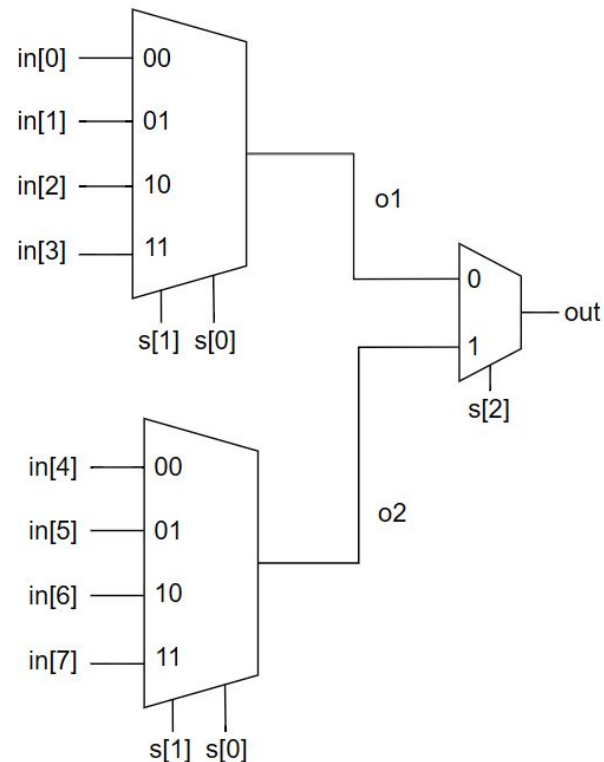
```
// If sel = 000 --> out = in[0]
// If sel = 001 --> out = in[1]
// ...
// If sel = 110 --> out = in[6]
// If sel = 111 --> out = in[7]
module mux81(
    input [2:0] sel,
    input [7:0] in,
    output out
);

    wire o1, o2;

    // Lower-level 4-1 muxes
    mux41 m1(.sel(sel[1:0]), .in(in[3:0]), .out(o1));
    mux41 m2(.sel(sel[1:0]), .in(in[7:4]), .out(o2));

    // Higher-level 2-1 mux
    // Concatenating o1 and o2 into a 2-bit wire
    mux21 m3(.sel(sel[2]), .in({o2, o1}), .out(out));

endmodule
```



Building More Complex Combinational Gates/Operators

What is this Operation Performing?

i_3	i_2	i_1	i_0	o_1	o_0	A
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

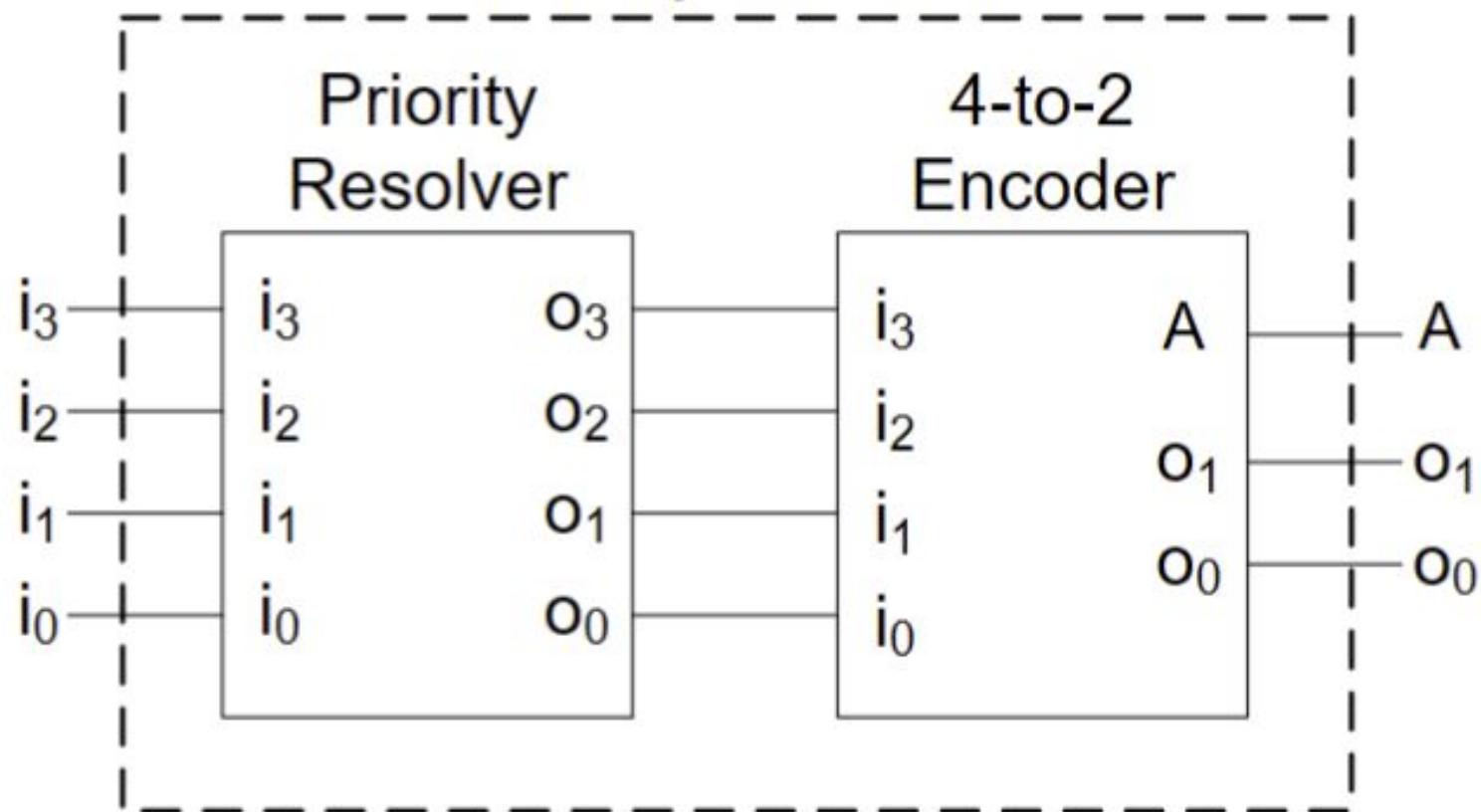
Hint: This was discussed in lecture

What does 'x' mean?

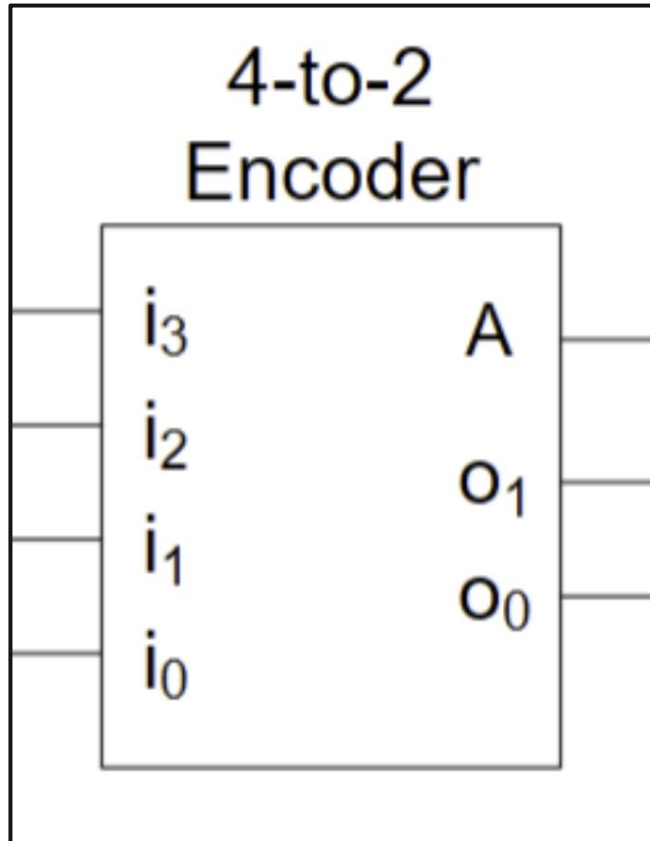
Does this cover all possible inputs?

How can we split the logic into easier parts?

Priority Encoder

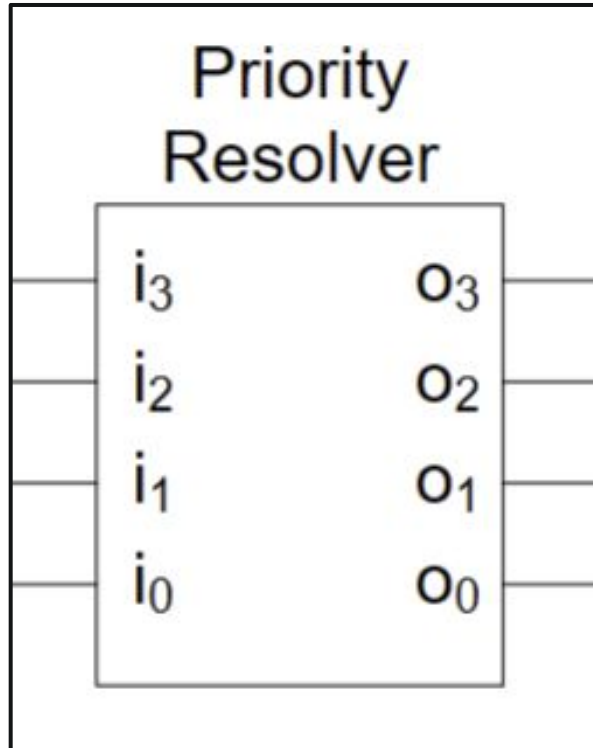


4-2 Encoder (**Expecting One-Hot Input**)



i_3	i_2	i_1	i_0	o_1	o_0	A
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1
Other Inputs				d	d	d

Priority Resolver (Preparing Input for Encoder)



Priority Resolver

i_3	i_2	i_1	i_0	o_3	o_2	o_1	o_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	x	0	0	1	0
0	1	x	x	0	1	0	0
1	x	x	x	1	0	0	0

Now It's Your Turn! (Code and Test in LabsLand)

```
module PriorityEncoder(  
    input [3:0] SW,  
    output [1:0] LEDR,  
    output [0:0] LEDG  
);  
  
    PriorityResolver pr(.in(??), .out(??));  
    Encoder42 e(.in(??), .A(??), .out(??));  
  
endmodule
```

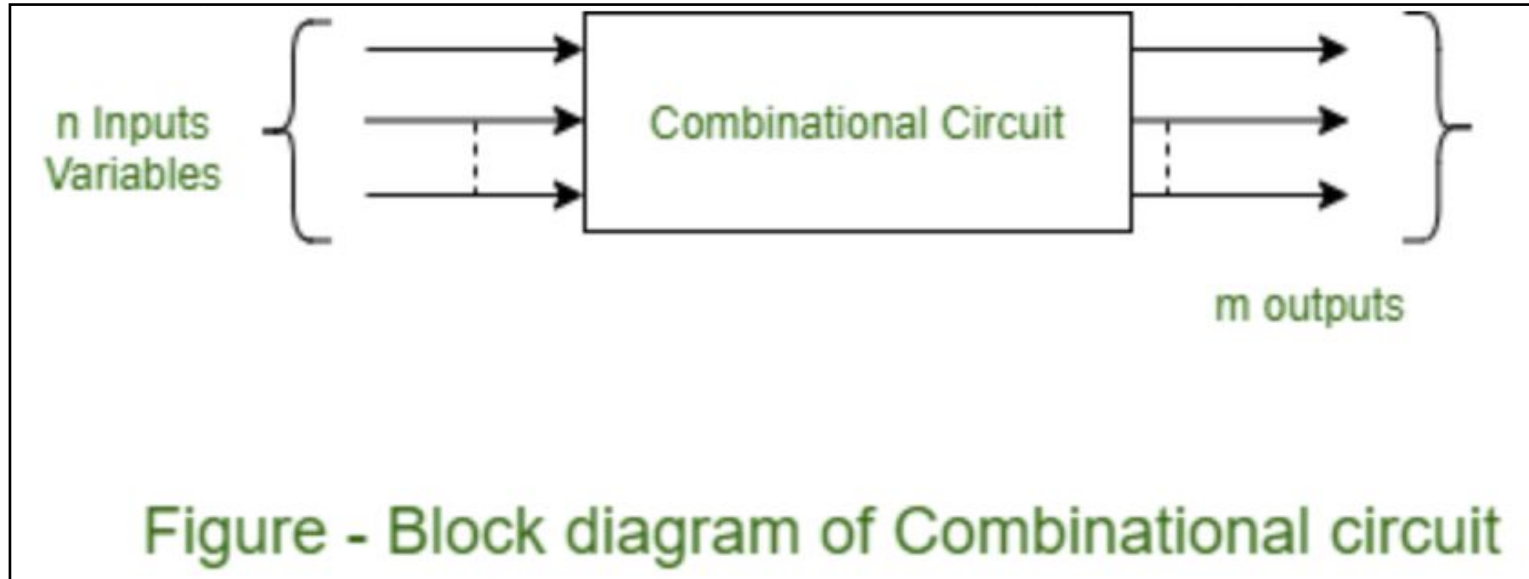
```
module PriorityResolver(  
    input [3:0] in,  
    output [3:0] out  
);  
  
endmodule
```

```
module Encoder42(  
    input [3:0] in,  
    output A,  
    output [1:0] out  
);  
  
endmodule
```

Project 4 “Calculator” Overview

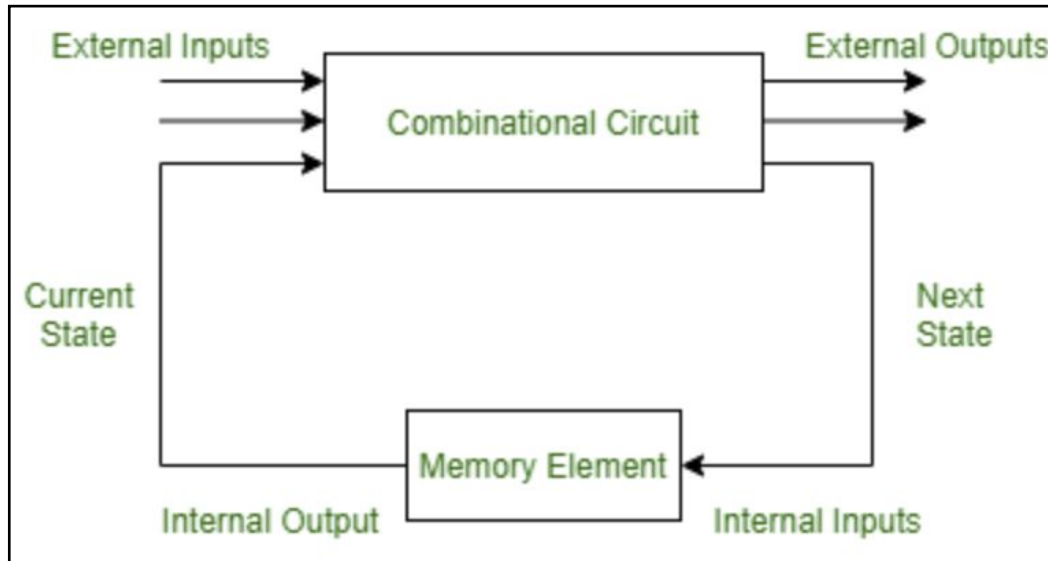
Combinational Circuits Overview (Projects 1 - 4)

- Outputs depend ONLY on present combination of current inputs
- Stateless / No Memory
- Use of wires primarily

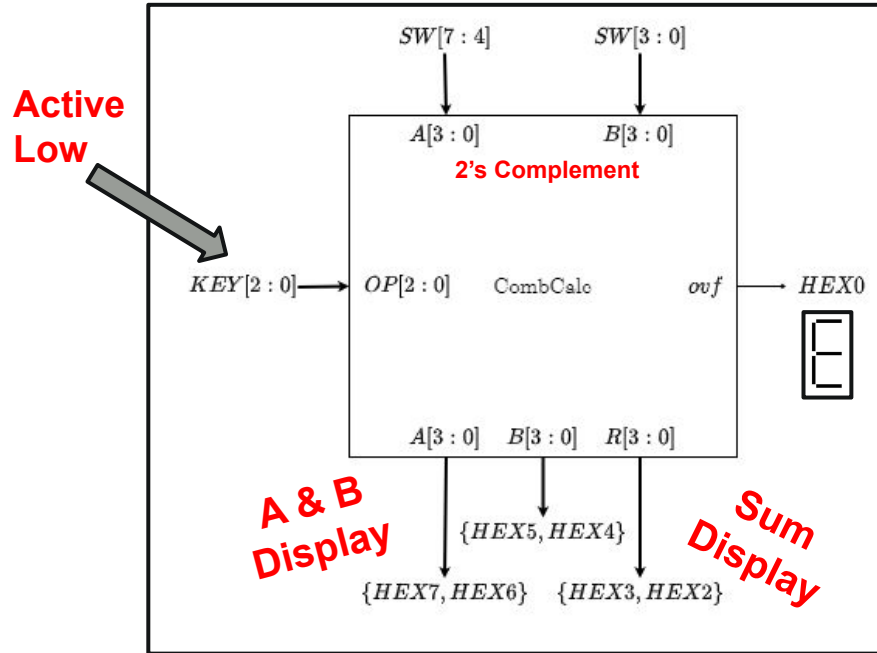


Sequential Circuits Overview (Projects 5 - 7)

- Combinational Circuit + **Memory Element**
- Outputs depend on present combination of inputs and current state
- Use of memory to store previous state/output

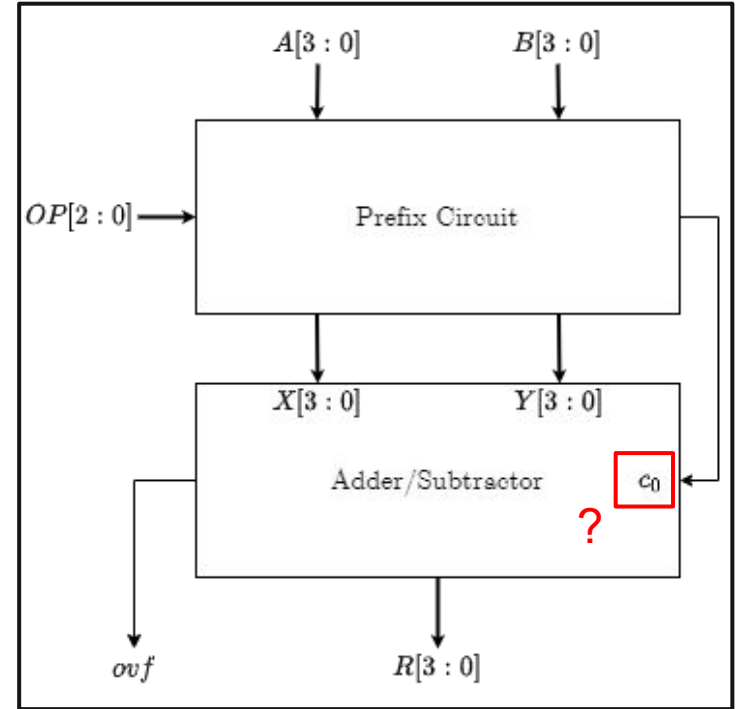
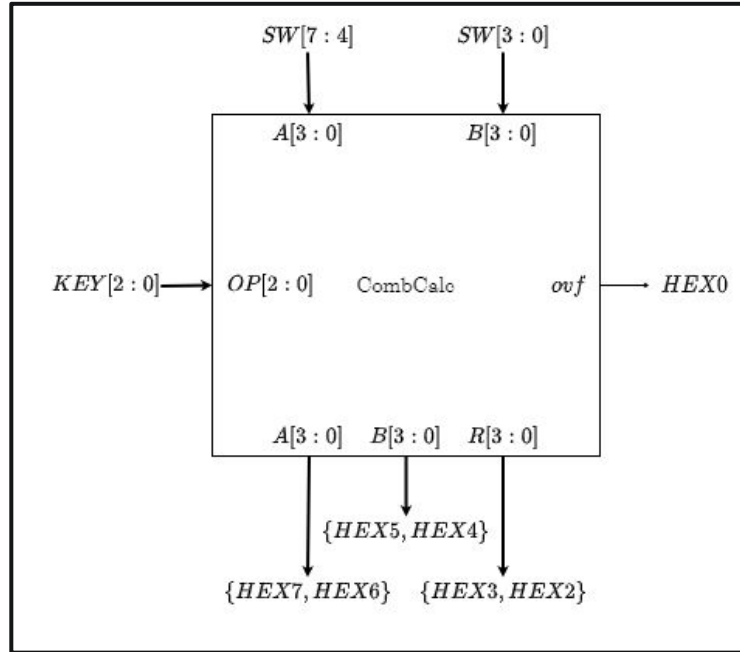


Combinational Calculator Overview



OP[2]	OP[1]	OP[0]	R	Operation
0	0	0	$A + B$	A plus B
0	0	1	$A - B$	A minus B
0	1	–	$ B $	$\text{abs}(B)$
1	0	0	$B + A$	B plus A
1	0	1	$B - A$	B minus A
1	1	–	$ A $	$\text{abs}(A)$

Splitting Up The Logic



***Only 1 Adder/Subtractor Instantiation that must be able to do all operations!**

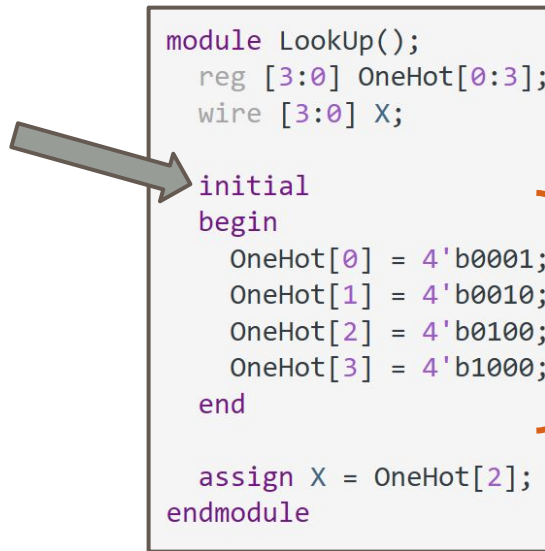
USE BEHAVIORAL VERILOG FOR Project 4!!!

- **You are allowed to use**
 - Continuous 'assign' statements
 - Ternary operators / Concatenation
 - Procedural Blocks (Though not preferred until the later projects)
- **Behavioral Operations NOT allowed for this project**
 - Addition (+) (**Exception** to genvar 'i+1' in the sample code)
 - subtraction/integer negation (-)
 - Greater than (>), less than (<), or equality (==)
 - If-Else statements
 - Case statements

Procedural Statements and Blocks

- 2 procedural statements: **always** and **initial**
 - Each procedural statement represents a separate activity flow
 - All other behavioral modeling statements go inside the procedural blocks

Procedural Statement



```
module LookUp();  
    reg [3:0] OneHot[0:3];  
    wire [3:0] X;  
  
    initial  
    begin  
        OneHot[0] = 4'b0001;  
        OneHot[1] = 4'b0010;  
        OneHot[2] = 4'b0100;  
        OneHot[3] = 4'b1000;  
    end  
  
    assign X = OneHot[2];  
endmodule
```

Procedural Block

THANK YOU

**Discussion
Review Session
Next Week!!**