Lecture 9, Sep 27, 2022

Additional Verilog Statements

- `always` block
  - Statements in an always block execute sequentially
  - In an always block, we don't use `assign`
  - `=` (as opposed to `assign`) is a blocking assignment; must be used inside `always` blocks and enforces sequential execution order
- Conditionals such as if/else/else if must exist in an `always` block
  - *if* without *else* generates a latch

```
module mux(input logic x1, x2, s,
           output logic f);
    always_comb // comb for combinational logic
    begin // Enclose multiple statements, akin to {}
        if (s == 0)
            f = x1; // assign is not used
        else
            f = x2;
    end
endmodule
```

- `case` statements
  - Can be used to do pattern matching
  - Instead of deriving a logic expression, we can let Verilog do it for us
  - Also needs to be inside an `always` block
  - `default` catches unspecified cases; without this the compiler will generate latches (more on this later)

```
module seg7(input logic [3:0] sw,
            output logic [6:0] h);
    always_comb
    begin
        case (sw)
            0: HEX0 = 7'b1000000;
            1: HEX0 = 7'b1111001;
            // ...
            9: HEX0 = 7'b0000100;
            // Catch cases that have not been specified, since sw can go up to 15
            default: HEX0 = 7'b1111111;
        endcase
    end
endmodule
```

Karnaugh Maps (K-Maps)

- A method of optimizing logic expressions
- The point of logic simplification is to reduce the cost (area) of a circuit; for our purposes, our metric for cost is the number of gates and inputs
  - cost = # of gates + # of inputs
- Optimization using boolean algebra is awkward and error prone
  - When optimizing using boolean algebra, we need to combine terms, but seeing that those combinations are possible is challenging
- Karnaugh Maps are a type of truth table in which minterms that cam be combined are adjacent
- Example: 2-variable K-Map

x_2 \ x_1	0	1
0	m_0	m_2
1	m_1	m_3

- Looking at the first column, $f = m_0 + m_1 = \bar{x}_1\bar{x}_2 + \bar{x}_1 x_2 = \bar{x}_1$
  - The second row: $f_2 = m_1 + m_2 = \bar{x}_1 x_2 + x_1 x_2 = x_2$
- Example: $f(x_1, x_2) = \sum m(0, 1, 3)$
  - $f = \bar{x}_1\bar{x}_2 + \bar{x}_1 x_2 + x_1\bar{x}_2$
    * As it is the circuit has a cost of 17 (3 AND, 1 3-input OR, 2 NOT + 2 inputs per AND, 3 inputs per OR, 1 input per NOT)
  - K-Map:

x_2 \ x_1	0	1
0	1	0
1	1	1

- This lets us simplify our circuit to $f = \bar{x}_1 + x_2$ which only has a cost of 5
- To simplify using a K-Map, we group adjacent minterms in the map
  - The second row shows that regardless of x_1, as long as x_2 is 1, the expression is 1, so that row simplifies to x_2
  - The first column shows that regardless of x_2, as long as x_1 is 0, the expression is 1, so the row simplifies to $\bar{x}_1$
- We can only group terms in group sizes of powers of 2 (for a 2×2 K-Map, we can group 2 terms or 4 terms)