SOL's Verilog Cheatsheat for Not So Beginners

- For grammar, go to https://marceluda.github.io/rp_dummy/EEOF2018/Verilog_Cheat_Sheet.pdf
- Different views of verilog:
  - Behavioral
    - Describe how data flows/changes
    - Probably not be synthesizable
    - Cannot help beginners understand the real composition of circuits
    - Can be optimized by synthesizer
    - Use it only when you are sure what you are constructing (or building simulation test bench)
  - Structural
    - Describe how circuits are formed (registers and combinational logics)
    - Good for beginners
    - Complex
  - Maybe you should try codegen or HLS
- Wire connection:
  - Behavioral:

```
always @ (*) begin
    a <= b;
end
```

  - Structural:

```
assign a = b;
```

- Multiplexer:
  - Behavioral:

```
always @ (*) begin
    if (s) begin
        y <= exp1;
    end
    else begin
        y <= exp0;
    end
end
```

  - Structural:

```
// Use submodule
Mux mux_asbc(y, s, exp0, exp1);

// Use ?:
assign y = s ? exp1 : exp0;

// For vec input
assign y = exp[s];
```

- Multilevel Multiplexer:
  - Behavioral:

```
always @ (*) begin
    case (s):
        2'd0: y = exp0;
        2'd1: y = exp1;
        2'd2: y = exp2;
        2'd3: y = exp3;
    endcase
end
```

  - Structural:

```
Mux mux_01(y01, s[0], exp0, exp1);
Mux mux_23(y23, s[0], exp2, exp3);
Mux mux_02(y  , s[1], y01 , y23);
```

- Prioritized Encoder:
  - Behavioral:

```
integer i;

always @ (*) begin
    for (i = 0; i < n; i = i + 1) begin
        if (x[i])
            y <= i;
    end
end
```

  - Structural:

```
genvar i;
assign __y[0] = 0;  // __y is temporary
assign y = __y[n];
generate
    for (i = 1; i <= n; i = i + 1) begin
        __y[i] = x[i] ? i : __y[i-1];
    end
endgenerate
```

- Moore State Machine
  - Behavioral:

```
always @ (posedge clk) begin
    state <= some_behavioral_logic(state, in);
    out <= some_behavioral_logic(state, in);
end
```

  - Structural:

```
assign next_state = some_combinational_logic(state, in);
assign out = state;
always @ (posedge clk) begin
    state <= next_state;
end
```

  - From here you can learn that sequential logic is only registers connected by combinational circuit.

- Mealy State Machine
  - Behavioral:

```
assign out = some_behavioral_logic(state, in);
always @ (posedge clk) begin
    state <= some_behavioral_logic(state, in);
end
```

  - Structural:

```
assign next_state = some_combinational_logic(state, in);
assign out = next_state;
always @ (posedge clk) begin
    state <= next_state;
end
```

- Async-negedge sync-posedge FF
  - Behavioral:

```
always @ (posedge clk or posedge start) begin
    if (start)
        ready <= 1'b0;
    else if (valid)
        ready <= 1'b1;
end
```

  - Structural:

```
reg ready_ff;
always @ (posedge clk) begin
    ready_ff <= next_ready_ff;
end

// When start, async reset ready
assign ready = start ? 1'b0 : ready_ff;
// Not started at this cycle
// ready: already ready (idle)
// valid: not ready now, but recieved valid now, so next cycle can be
ready
assign next_ready_ff = !start & (ready | valid);
```