3. Introduction to Design With Verilog

3.4 Exercises

Dr. Paul D. Franzon
Exercise: Three Timing Examples (from Timing Notes)

- What do these look like in Verilog?

```verilog
always@(A or B or C)
begin
  if (A>B) then E = A; else E = B;
  if (C>E) then F = C; else F = E;
end
always@(posedge clock)
  if (D>F) then G <= D; else G <=F;
```

Why not move E, F assignments down to here?
... *Three timing examples*

- Produce a Verilog code fragment for ...
  - Use continuous assignment

```
assign E = (A>B) ? A : B;
assign F = (C>D) ? C : D;
always@(posedge clock)
  if (E>F) then G <= E; else G <= F;
```
… Three Timing Examples

- And for this…

Note: Outputs of all flip-flops have to be named
Sample Problem

- **Accumulator:**
  - Design an 8-bit adder accumulator with the following properties:
  - While ‘accumulate’ is high, adds the input, ‘in1’ to the current accumulated total and add the result to the contents of register with output ‘accum_out’.
    - use absolute (not 2’s complement) numbers
  - When ‘clear’ is high (‘accumulate’ will be low) clear the contents of the register with output ‘accum_out’
  - The ‘overflow’ flag is high is the adder overflows

*Hint:*

8-bit adder produces a 9-bit result:

\[ \{\text{carry\_out}, \text{sum}\} = A+B; \]
Sketch Design

1. Determine and name registers.
2. Determine combinational logic

\[
\text{Clear} \\
\text{accumulate}
\]
module accum (clock, accumulate, clear, in1, accum_out, overflow);

input clock, accumulate, clear;
input [7:0] in1;
output [7:0] accum_out;
output overflow;

reg [7:0] accum_out;
wire [7:0] accum_in;
wire overflow;

always @(posedge clock)
begin
if (clear) accum_out <= 8'b0; else if (accumulate) accum_out <= accum_in;
end

assign {overflow, accum_in} = accum_out + in1;
endmodule /* counter */
Summary

- What are our two “mantras” used here?

- What is built for all assignments after always@(posedge clock)?

- What is built after always@(A or B)

- What is built with assign C =
Summary

• In Synthesis with Synopsys
  • What is important after the “read” statement?

• Which timing library do we use for the first compile?

• What does “compile” do?

• What is important to check before finishing the synthesis run?