Introduction

Sequential logic is constructed using flip-flops for state storage, combinational next state logic, and combinational output logic. These constructs can be described in Verilog using “clocked” always blocks, combinational always blocks, and continuous assign (i.e. assign) statements.

In the project last semester, we built the FSM for our Successive Approximation circuit using a clocked always block for state flip-flops and a combinational always block for the next-state and output logic. Separate synchronous Set/Reset flip-flops were used to store the estimate under the control of the FSM.

Other approaches are also possible. For example, the next-state logic of the FSM can be combined with the flip-flops in the clocked always block. This approach is often used in counters, as described last semester. A separate combinational always block or assign statements are still needed for the output logic, because any outputs in the clocked always block are registered – they pass through flip-flops before reaching their output. This is usually not desirable.

However, in some cases this extra storage can be useful. For example, the Successive Approximation circuit needs to store its estimate value in a separate set of flip-flops. Last semester we implemented these flip-flops in separate always blocks. They can instead be implemented directly in the clocked always block using nonblocking assignments. Figure 1 shows a state diagram with nonblocking assignments to set and clear the bits of the estimate value, as appropriate.

![State Diagram](image)

**Figure 1 – State Diagram for Modified Successive Approximation Circuit**

In this laboratory we will explore this coding style and compare it to the approach used in the design project last semester. Figure 2 shows a “skeleton” of code that implements the Successive Approximation circuit. You will complete this skeleton, simulate it using
Verilog, synthesize it, and compare the size of your circuit with the circuit you generated last semester in your project.

```verilog
module SAR_REG(clk, reset, start, rdy, E)
    input  clk, reset, start;
    output rdy;
    output E[3:0];

    reg    E[3:0];
    reg    rdy;

    reg    cs[2:0];

    parameter INIT=3’d0, TST3=3’d3, ...;

    always @(posedge clk)
    begin
        if (reset) .... ; // add code here to handle reset
        case (cs)
        INIT: if (start)
            begin
                E  <= 4’b1000;
                cs <= INIT
            end
        else cs <= TST2;
        TST3: // add more code here for remaining states
        endcase
    end
endmodule
```

Figure 2 – Code Skeleton for Successive Approximation Circuit

In the Lab
1. Copy the code skeleton above and add code to implement the full state sequencing.
2. Note that the rdy output must still be implemented with combinational logic. Use an assign statement to implement this output.
3. Simulate your circuit with Verilog and verify that it simulates properly.
4. Synthesize your circuit using the Synopsys Design Compiler and the script "/home/cad/compile_design.scr" as during last semester.
5. Use design_analyzer to view a schematic of your design and plot it on the laser printer.
6. Use the db2mag script to place and route your synthesized circuit. Use Magic to measure its cell area, and use flea to generate a plot of the resulting layout. How does it compare to the circuit that you designed last semester?

Report
For your lab report, hand in the following:
1. A short technical memorandum which describes (a) what was done, (b) what you learned, and (c) what difficulties you encountered. Discuss any errors you found in your design, describing how you found out what was wrong and how you fixed the problems.
2. A listing of your Verilog source file and simulation timing diagram showing its proper operation.
3. A schematic plot of your synthesized circuit.
4. A layout plot of your design after place & route. Note the area of the circuit and how it compares to the circuit you designed for the project.