M4 Lab Prep - 31/10/21

How many programmable logic elements are available on the Cyclone V device?

85K Programmable Logic Elements

What clock signals are available for user logic?

Four 50MHz clock sources from clock generator

Which FPGA pins are connected to the push-buttons, KEY0-KEY3, to the switches, SW0-SW9, and to the red LEDs, LEDR0-LEDR9?

Table 3-6 Pin Assignments for Slide Switches

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
SW[0]	PIN_AB12	Slide Switch[0]	3.3V
SW[1]	PIN_AC12	Slide Switch[1]	3.3V
SW[2]	PIN_AF9	Slide Switch[2]	3.3V
SW[3]	PIN_AF10	Slide Switch[3]	3.3V
SW[4]	PIN_AD11	Slide Switch[4]	3.3V
SW[5]	PIN_AD12	Slide Switch[5]	3.3V
SW[6]	PIN_AE11	Slide Switch[6]	3.3V
SW[7]	PIN_AC9	Slide Switch[7]	3.3V
SW[8]	PIN_AD10	Slide Switch[8]	3.3V
SW[9]	PIN_AE12	Slide Switch[9]	3.3V

Table 3-7 Pin Assignments for Push-buttons

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
KEY[0]	PIN_AA14	Push-button[0]	3.3V
KEY[1]	PIN_AA15	Push-button[1]	3.3V
KEY[2]	PIN_W15	Push-button[2]	3.3V
KEY[3]	PIN_Y16	Push-button[3]	3.3V

Table 3-8 Pin Assignments for LEDs

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
LEDR[0]	PIN_V16	LED [0]	3.3V
LEDR[1]	PIN_W16	LED [1]	3.3V
LEDR[2]	PIN_V17	LED [2]	3.3V
LEDR[3]	PIN_V18	LED [3]	3.3V
LEDR[4]	PIN_W17	LED [4]	3.3V
LEDR[5]	PIN_W19	LED [5]	3.3V
LEDR[6]	PIN_Y19	LED [6]	3.3V
LEDR[7]	PIN_W20	LED [7]	3.3V
LEDR[8]	PIN_W21	LED [8]	3.3V
LEDR[9]	PIN_Y21	LED [9]	3.3V

Each push-button switch provides a high logic level when it is not pressed, and provides a low logic level when depressed

Are the LEDs turned on by a logic 1 or by a logic 0? (In other words, are they active high or active low?) Are the push-buttons active high, or active low?

Each LED is driven directly by a pin on the Cyclone V SoC FPGA; driving its associated pin to a high logic level turns the LED on, and driving the pin low turns it off.

Before submissions

```
always_ff @(posedge clock, negedge n_rst)
    begin: SEQ                              //Sequential label for modelsim
        if (!n_rst)                         //Reset
            present_state <= idle;
        else                                //Update state on clockedge
            present_state <= next_state;
    end

always_comb
    begin: COM                              //Combinational label for modelsim

        //Set default values of output
        add = '0
        shift = '0
        ready = '0
        reset = '0

        unique case (present_state)         //Unique case label gives compiler
error if a state is missing
            idle : begin
                output1 = '1;               //Assert unconditional output
                if (start)                  //Conditional to progress to next
state
                    next_state = adding;
                else                        //Else stay in ready state
                    next_state = idle;
            end
            adding : begin
                count <= count-1;           //update count
                if (QO)                     //Conditional to progress to next
state
                    begin
                        ADD = '1;           //Assert conditional output
                        next_state = shifting;
                    end
```

```
                else
                    next_state = shifting;
            end
            shifting : begin
                shift = '1;                 //Assert unconditional output
                if (count > 0)              //Conditional to progress to next
state
                    next_state = adding;
                else                        //Else stay in ready state
                    next_state = stopped;
            end
            stopped : begin
                ready = '1;                 //Assert unconditional output
                if (start)                  //Conditional to progress to next
state
                    next_state = idle;
                else                        //Else stay in ready state
                    next_state = stopped;
            end
        endcase
    end
```

After Compilation

```
always_ff @(posedge clock, negedge n_rst)
    begin: SEQ                              //Sequential label for modelsim
        if (!n_rst)                         //Reset
            present_state <= idle;
        else                                //Update state on clockedge
            present_state <= next_state;
    end

always_comb
    begin: COM                              //Combinational label for modelsim

        //Set default values of output
        add = '0;
        shift = '0;
        ready = '0;
        reset = '0;

        unique case (present_state)         //Unique case label gives compiler
error if a state is missing
            idle : begin
                reset = '1;                //Assert unconditional output
                if (start)                  //Conditional to progress to next
state
                    next_state = adding;
                else                        //Else stay in ready state
                    next_state = idle;
            end
            adding : begin
                count <= count-1;           //update count
                if (Q0)                     //Conditional to progress to next
state
                    add = '1;               //Assert conditional output
                next_state = shifting;
            end
            shifting : begin
                shift = '1;                 //Assert unconditional output
                if (count > 0)              //Conditional to progress to next
state
                    next_state = adding;
                else                        //Else stay in ready state
                    next_state = stopped;
```

```
            end
            stopped : begin
                ready = '1;                 //Assert unconditional output
                if (start)                  //Conditional to progress to next
state
                    next_state = idle;
                else                        //Else stay in ready state
                    next_state = stopped;
            end
        endcase
    end
endmodule
```

After Verilator

```
always_ff @(posedge clock, negedge n_rst)
    begin: SEQ                              //Sequential label for modelsim
        if (!n_rst)                         //Reset
            present_state <= idle;
        else                                //Update state on clockedge
            present_state <= next_state;
        if (count_flag)                         //Reset
            count <= count-1;
    end


always_comb
    begin: COM                              //Combinational label for modelsim

        //Set default values of output
        add = '0;
        shift = '0;
        ready = '0;
        reset = '0;
        count_flag = '0;

        unique case (present_state)         //Unique case label gives compiler
error if a state is missing
            idle : begin
                reset = '1;                 //Assert unconditional output
                if (start)                  //Conditional to progress to next
state
                    next_state = adding;
                else                        //Else stay in ready state
                    next_state = idle;
            end
            adding : begin
                count_flag = '1;            //update count
                if (Q0)                     //Conditional to progress to next
state
                    add = '1;               //Assert conditional output
                next_state = shifting;
            end
            shifting : begin
                shift = '1;                 //Assert unconditional output
```

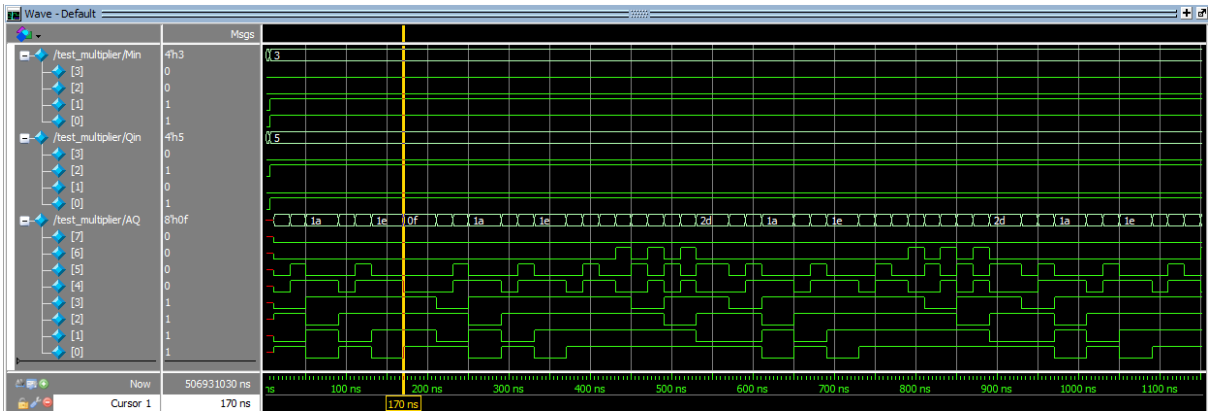
```
                if (count > 0)              //Conditional to progress to next
state
                    next_state = adding;
                else                        //Else stay in ready state
                    next_state = stopped;
            end
            stopped : begin
                ready = '1;                 //Assert unconditional output
                if (start)                  //Conditional to progress to next
state
                    next_state = idle;
                else                        //Else stay in ready state
                    next_state = stopped;
            end
        endcase
    end
```

```
To: Joseph Butterworth (jdb1g20)

Logfile for user jdb1g20 attempt 11; sequencer.sv
Starting ModelSim
QuestaSim-64 vlog 10.6c Compiler 2017.07 Jul 25 2017
Start time: 21:32:50 on Oct 31,2021
vlog -work jdb1g20work -sv sequencer.sv
-- Compiling module sequencer

Top level modules:
       sequencer
End time: 21:32:50 on Oct 31,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
================================================================
Starting Verilator
================================================================
Starting Quartus
```

Modelsim output shows correct multiplier result:



Why should you take notice about warnings about latches and combinational loops?

This is likely to cause a stuck state in your design.

If you have seen such warnings, how have you modified your code to eliminate these warnings?

Remove any blocking assignments in combinational logic and make sure no feeding back occurs.

The testbench has no inputs or outputs. What would Quartus synthesise if you made the testbench the top level of your design?

If the testbench was the top level entity, the board would still synthesise the verilog code. However, there would be no connection to the inputs and outputs, so there would be no way to interact with the multiplier or easily extract the value.