Modeling Registers and Counters

Introduction

When several flip-flops are grouped together, with a common clock, to hold related information the resulting circuit is called a register. Just like flip-flops, registers may also have other control signals. You will understand the behavior of a register with additional control signals. Counters are widely used sequential circuits. In this lab you will model several ways of modeling registers and counters. Please refer to the Vivado tutorial on how to use the Vivado tool for creating projects and verifying digital circuits.

Objectives

After completing this lab, you will be able to:

- Model various types of registers
- Model various types of counters

Registers Part 1

In a computer system, related information is often stored at the same time. A **register** stores bits of information in such a way that systems can write to or read out all the bits simultaneously. Examples of registers include data, address, control, and status. Simple registers will have separate data input and output pins but clocked with the same clock source. A simple register model is shown below.

```
module Register (input [3:0] D, input Clk, output reg [3:0] Q);
   always @(posedge Clk)
        Q <= D;
endmodule
```

Notice that this is similar to a simple D Flip-flop with multiple data ports.

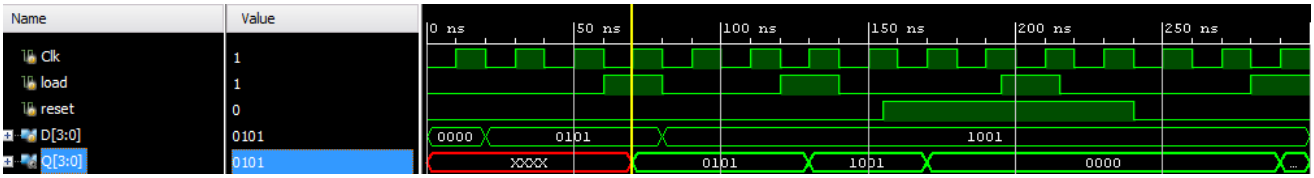
The simple register will work where the information needs to be registered every clock cycle. However, there are situations where the register content should be updated only when certain condition occurs. For example, a status register in a computer system gets updated only when certain instructions are executed. In such case, a register clocking should be controlled using a control signal. Such registers will have a clock enable pin. A model of such register is given below.

```
module Register_with_synch_load_behavior(input [3:0] D, input Clk, input
load, output reg [3:0] Q);
   always @(posedge Clk)
      if (load)
         Q <= D;
endmodule
```

Another desired behavior of registers is to reset the content when certain condition occurs. A simple model of a register with synchronous reset and load (reset has a higher priority over load) is shown below

```
module Register_with_synch_reset_load_behavior(input [3:0] D, input Clk,
input reset, input load, output reg [3:0] Q);
   always @(posedge Clk)
      if (reset)
      begin
         Q <= 4'b0;
      end else if (load)
      begin
        Q <= D;
      end
endmodule
```

1-1. Model a 4-bit register with synchronous reset and load using the model provided above. Develop a testbench and simulate the design. Assign Clk, D input, reset, load, and output Q. Verify the design in hardware.



1-1-1. Open Vivado and create a blank project called lab6_1_1**.**

1-1-2. Create and add the Verilog module that will model the 4-bit register with synchronous reset and load. Use the code provided in the above example.

1-1-3. Develop a testbench and simulate the design for **300ns**. Analyze the output.

1-1-4. Add the appropriate board related master XDC file to the project and edit it to include the related pins, assigning Clk to **SW15**, D input to **SW3-SW0**, reset to **SW4**, load to **SW5**, and Q to **LED3-LED0**.

1-1-5. Add the following line of code in the XDC file to allow SW15 be used as a clock

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets { clk }];
```

1-1-6. Synthesize the design.

1-1-7. Implement the design.

Look at the Project Summary and Utilization table, and record what FPGA resources are used.

1-1-8. Generate the bitstream, download it into the Basys3 or the Nexys4 DDR board, and verify the functionality.

1-1-9. **Demonstrate simulation results, resources, working circuit and Verilog code to the TA.**

In some situations, it is necessary to set the register to a pre-defined value. For such a case, another control signal, called set, is used. Typically, in such registers, reset will have a higher priority over set, and set will have a higher priority over load control signal.

1-2. Model a 4-bit register with synchronous reset, set, and load signals. Assign Clk, D input, reset, set, load, and output Q. Verify the design in hardware.

1-2-1. Open Vivado and create a blank project lab6_1_2**.**

1-2-2. Create and add the Verilog module that will model the 4-bit register with synchronous reset, set, and load control signals.

1-2-3. Add the appropriate board related master XDC file to the project and edit it to include the related pins. Note: You may have to add the CLOCK_DEDITCATED_ROUTE property for the Clk pin depending on which switch you select.

1-2-4. Synthesize the design.

1-2-5. Implement the design. Look at the Project Summary and record the resources used. Understand the result.

1-2-6. Generate the bitstream, download it into the Basys3 or the Nexys4 DDR board, and verify the functionality.

1-2-7. **Demonstrate resources used, working circuit and Verilog code to the TA.**

The above registers are categorized as parallel registers. There are another kind of registers called shift registers. A shift register is a register in which binary data can be stored and then shifted left or right when the control signal is asserted. Shift registers can further be sub-categorized into parallel load serial out, serial load parallel out, or serial load serial out shift registers. They may or may not have reset signals.

In Xilinx FPGAs, a LUT can be used as a serial shift register with one bit input and one bit output using one LUT as an SRL32 (32 bit shift register) providing efficient design (instead of cascading up to 32 flip-flops in multiple LUTs) provided the code is written properly. It may or may not have enable signal. When the enable signal is asserted, the internal content gets shifted by one bit position and a new bit value is shifted in. Here is a model for a simple one- bit serial shift in and shift out register without enable signal. It is modeled to shift for 32 clock cycles before the shifted bit is brought out. This model can be used to implement a delay line.

```
module simple_one_bit_serial_shift_register_behavior(input Clk, input
ShiftIn, output ShiftOut);
    reg [31:0] shift_reg;

    always @(posedge Clk)
          shift_reg <= {shift_reg[30:0], ShiftIn};
    assign ShiftOut = shift_reg[31];

endmodule
```

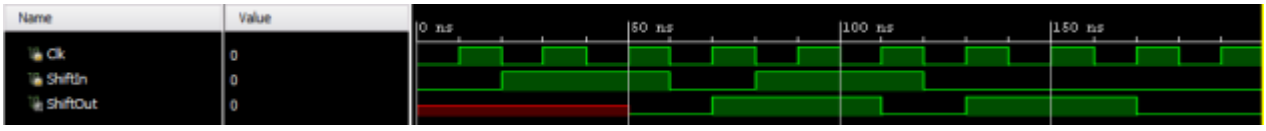
The above model can be modified if we want to implement a delay line less than 32 clocks. Here is the model for the delay line of 3 clocks.

```
module delay_line3_behavior(input Clk, input ShiftIn, output ShiftOut);
    reg [2:0] shift_reg;

    always @(posedge Clk)
          shift_reg <= {shift_reg[1:0], ShiftIn};
    assign ShiftOut = shift_reg[2];

endmodule
```

1-3. Model a 1-bit delay line shift register with 3 clocks delay using the above code. Develop a testbench and simulate the design using the stimuli provided below. Assign Clk, ShiftIn, and output ShiftOut. Verify the design in hardware.



1-3-1. Open Vivado and create a blank project lab6_1_3**.**

1-3-2. Create and add the Verilog module that will model the 1-bit delay line shift register using the provided code.

1-3-3. Develop a testbench and simulate the design for **200ns**.

1-3-4. Add the appropriate board related master XDC file to the project and edit it to include the related pins. Note: You may have to add the CLOCK_DEDITCATED_ROUTE property for the Clk pin depending on which switch you select.

1-3-5. Synthesize the design.

1-3-6. Implement the design. Look at the Project Summary and note the resources used. Understand the result.

1-3-7. Generate the bitstream, download it into the Basys3 or the Nexys4 DDR board, and verify the functionality. (note: the slide switch for the clk signal may "bounce" when being turned on causing multiple clock signals each time it is turned on. For this lab we will not try to fix this).

1-3-8. **Demonstrate simulation results, resources, working circuit and code to the TA.**

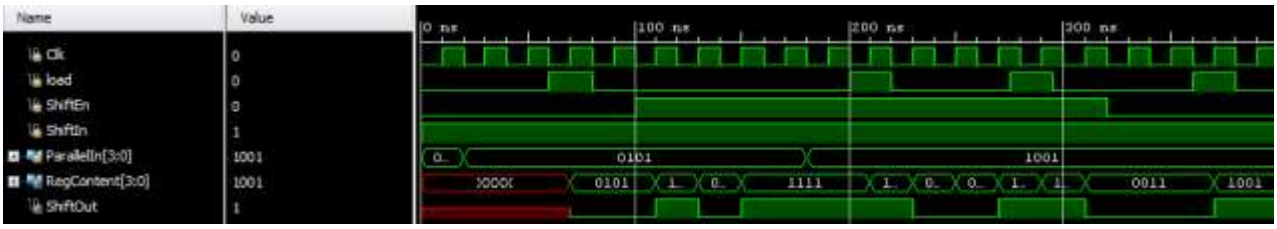
The following code models a four-bit parallel in shift left register with load and shift enable signal..

```
module Parallel_in_serial_out_load_enable_behavior(input Clk, input ShiftIn,
input [3:0] ParallelIn, input load, input ShiftEn, output ShiftOut, output
[3:0] RegContent);
    reg [3:0] shift_reg;

always @(posedge Clk)
       if(load)
          shift_reg <= ParallelIn;
       else if (ShiftEn)
          shift_reg <= {shift_reg[2:0], ShiftIn};
    assign ShiftOut = shift_reg[3];
    assign RegContent = shift_reg;

endmodule
```

1-4. Model a 4-bit parallel In left shift register using the above code. Develop a testbench and simulate the design using the stimuli provided below. Assign Clk, ParallelIn, load, ShiftEn, ShiftIn, RegContent, and ShiftOut. Verify the design in hardware.



1-4-1. Open Vivado and create a blank project lab6_1_4**.**

1-4-2. Create and add the Verilog module that will model the 4-bit parallel in left shift register using the provided code.

1-4-3. Develop a testbench and simulate the design for **400ns**.

1-4-4. Add the appropriate board related master XDC file to the project and edit it to include the related pins. Note: You may have to add the CLOCK_DEDITCATED_ROUTE property for the Clk pin depending on which switch you select.

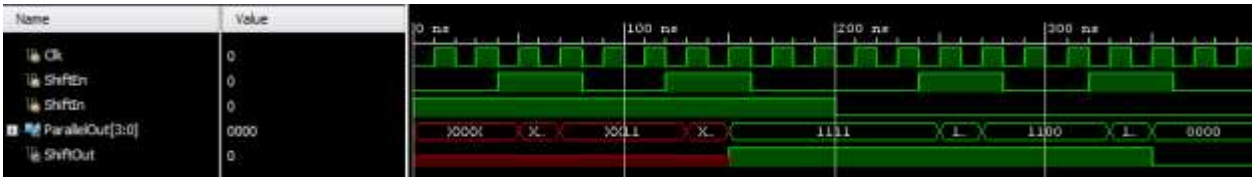
1-4-5. Synthesize the design.

1-4-6. Implement the design. Look at the Project Summary and note the resources used. Understand the result.

1-4-7. Generate the bitstream, download it into the Basys3 or the Nexys4 DDR board, and verify the functionality.

1-4-8. **Demonstrate simulation results, resources, working circuit and code to the TA.**

1-5. Write a model for a 4-bit serial in parallel out shift register. Develop a testbench and simulate the design. Assign Clk, ShiftEn, ShiftIn, ParallelOut, and ShiftOut. Verify the design in hardware.



1-5-1. Open Vivado and create a blank project lab6_1_5**.**

1-5-2. Create and add the Verilog module that will model the 4-bit serial in parallel shift register.

1-5-3. Develop a testbench and simulate the design for **400ns.**

1-5-4. Add the appropriate board related master XDC file to the project and edit it to include the related pins. Note: You may have to add the CLOCK_DEDITCATED_ROUTE property for the Clk pin depending on which switch you select.

1-5-5. Synthesize the design.

1-5-6. Implement the design.Look at the Project Summary and note the resources used. Understand the result.

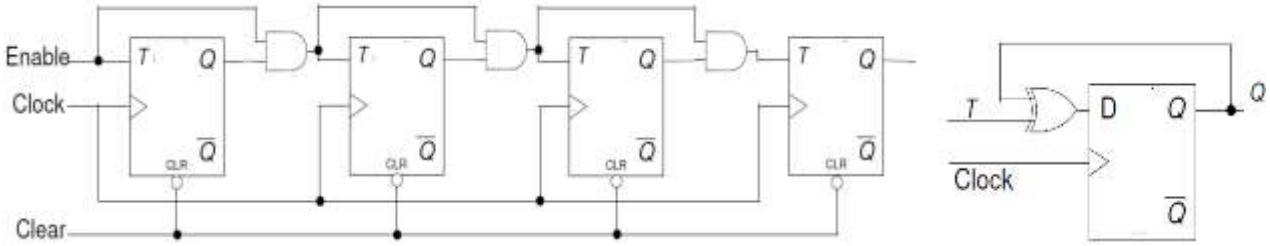
1-5-7. Generate the bitstream, download it into the Basys3 or the Nexys4 DDR board, and verify the functionality

1-5-8. **Demonstrate simulation results, working circuit and Verilog code to the TA.**

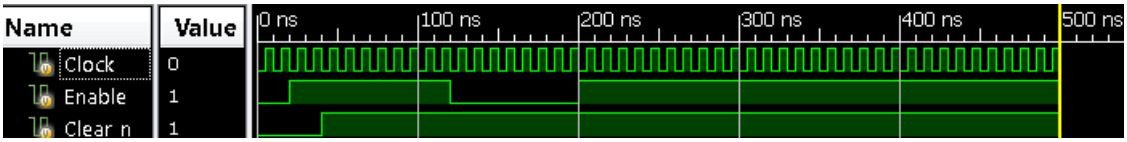
Counters Part 2

Counters can be asynchronous or synchronous. Asynchronous counters count the number of events solely using an event signal. Synchronous counters, on the other hand, use a common clock signal so that when several flip-flops must change state, the state changes occur simultaneously.

A binary counter is a simple counter which counts values up when an enable signal is asserted and will reset when the reset control signal is asserted. Of course, a clear signal will have a higher priority over the enable signal. The following diagrams show such a counter and how to construct a T flip-flop using a D flip-flop (note: you will require a D flip-flop with an active low CLR signal, also note that in the counter the clear signal is an asynchronous active low signal whereas the Enable is synchronous active high logic signal).



2-1. Design a 8-bit counter using T flip-flops, extending the above structure to 8-bits. Your design needs to be hierarchical, The design should be hierarchical, defining D flip-flop in behavioral modeling, creating T flip-flop from the D flip-flop, implementing additional functionality using dataflow modeling. Assign Clock input, Clear_n, Enable, and Q. Implement the design and verify the functionality in hardware.



2-1-1. Open Vivado and create a blank project lab6_2_1**.**

2-1-2. Create and add the Verilog module to provide the desired functionality.

2-1-3. Develop a testbench and validate the design.

2-1-4. Add the appropriate board related master XDC file to the project and edit it to include the related pins. Note: You may have to add the CLOCK_DEDITCATED_ROUTE property for the Clk pin depending on which switch you select.

2-1-5. Synthesize the design and view the schematic under the Synthesized Design process group. Indicate what kind and how many resources are used.

2-1-6. Implement the design.

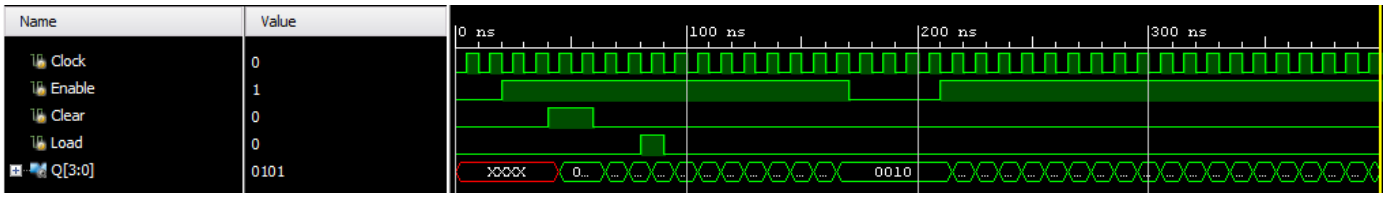
2-1-7. Generate the bitstream, download it into the Basys3 or the Nexys4 DDR board, and verify the functionality.

2-1-8. Demonstrate simulation results, working circuit and Verilog code to the TA.

Other types of binary counters include (i) up, (ii) down, and (iii) up-down. Each one of them may have count enable and reset as control signals. There may be situations where you may want to start the count up/down from some non-zero value and stop when some other desired value is reached. Here is an example of a 4-bit counter which starts with a value 10 and counts down to 0. When the count value 0 is reached, it will re-initialize the count to 10. At any time, if the enable signal is negated, the counter pauses counting until the signal is asserted back. It assumes that load signal is asserted to load the pre- defined value before counting has begun.

```
reg [3:0] count;
wire cnt_done;
assign cnt_done = ~| count;
assign Q = count;
always @(posedge Clock)
       if (Clear)
      count <= 0;
  else if (Enable)
     if (Load | cnt_done)
        count <= 4'b1010;    // decimal 10
          else
        count <= count - 1;
```

2-3. Model a 4-bit down-counter with synchronous load, enable, and clear as given in the code above. Develop a testbench (similar to the waveform shown below) and verify the design works. Assign Clock input, Clear, Enable, Load, and Q. Implement the design and verify the functionality in hardware.



2-3-1. Open Vivado and create a blank project lab6_2_3**.**

2-3-2. Create and add the Verilog module to provide the desired functionality.

2-3-3. Develop a testbench and validate the design.

2-3-4. Add the appropriate board related master XDC file to the project and edit it to include the related pins. Note: You may have to add the CLOCK_DEDITCATED_ROUTE property for the Clk pin depending on which switch you select.

2-3-5. Synthesize the design and view the schematic under the Synthesized Design process group.

2-3-6. Implement the design.

2-3-7. Generate the bitstream, download it into the Basys3 or the Nexys4 DDR board, and verify the functionality.

2-3-8. **Demonstrate simulation results, working circuit and Verilog code to the TA.**

Conclusion

In this lab, you learned how various kinds of registers and counters work. You modeled and verified the functionality of these components. These components are widely used in a processor system design.