Modeling Latches and Flip-flops

Introduction

Sequential circuits are digital circuits in which the output depends not only on the present input (like combinatorial circuits), but also on the past sequence of inputs. In effect, these circuits must be able to remember something about the past history of the inputs. Thus the timing concept is introduced and the clock signal provides the timing essence to the sequential circuits. Latches and flip-flops are commonly used memory devices in sequential circuits. *Please refer to the PlanAhead tutorial on how to use the PlanAhead tool for creating projects and verifying digital circuits*.

Objectives

After completing this lab, you will be able to:
- Model various types of latches
- Model flip-flops with control signals

Latches   Part 1

Storage elements can be classified into latches and flip-flops. A latch is a device with exactly two stable states: a high-output and a low-output. A latch has a feedback path, so information can be retained by the device. Therefore latches are volatile memory devices, and can store one bit of data for as long as the device is powered. As the name suggests, latches are used to "latch onto" information and hold the data in place.

An **SR latch** (Set/Reset) is an asynchronous device: it works independently of control signals and relies only on the state of the S and R inputs. The symbol, the circuit using NOR gates, and the truth table are shown below.

| S	R	Q	$\overline{Q}$
0	0	Latch	
0	1	0	1
1	0	1	0
1	1	Metastable	
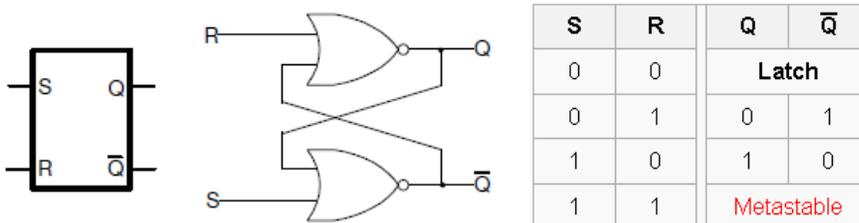
Though Xilinx FPGA can implement such a latch using one LUT (Look-Up Table) circuit, the following VHDL code shows how such circuit can be modeled using structural and dataflow modeling.

```
architecture SR_latch of SR_latch_structural is begin
     Q <= Qbar nor R;
     Qbar <= Q nor S;
end SR_latch;




architecture SR_latch of SR_latch_dataflow_arch is begin
     Q_i <= Q after 2ns;
     Qbar_i <= Qbar after 2ns;
     Q <= not(R or Qbar) after 2ns;
     Qbar <= not(S or Q) after 2ns;
end SR_latch;
```
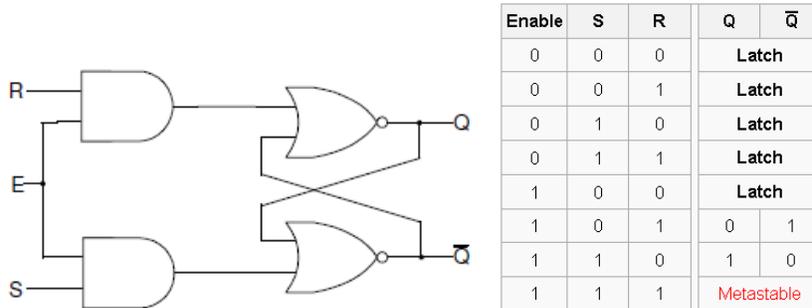
1-1. Design a SR latch (you can base it on the example shown above). Synthesize the design and view the RTL schematic of the synthesized design. Develop a testbench to test (see waveform below) and validate the design. Simulate the design. Assign S input to SW0 and R input to SW1. Assign Q to LED0 and Qbar to LED1. Implement the design and verify the functionality in hardware.
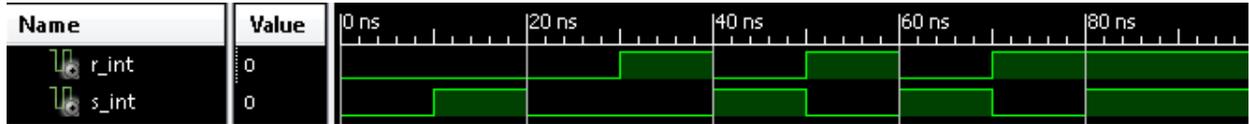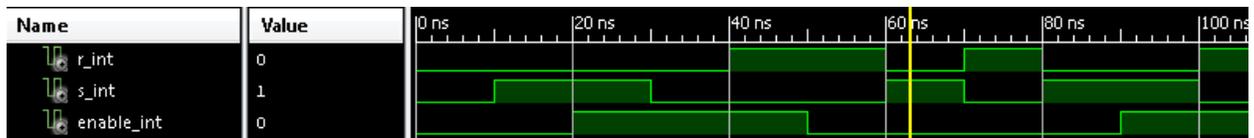

1-1-1. Open PlanAhead and create a blank project called lab5_1_1.

1-1-2. Create and add the VHDL module with the SR_latch_dataflow code.

1-1-3. Synthesize the design and view the schematic under the Synthesized Design process group. Verify that it uses 3 LUTs and 4 IOs (2 IBUF, and 2 OBUF).

1-1-4. Implement the design and view the project summary. It should show 1 LUTs, 1 slice, and 4 IOs. The reason it uses only one LUT because LUT in Spartan-6 has two outputs O5 and O6. It uses those two outputs to implement Q and Qbar.

1-1-5. Develop a testbench (see waveform above) to test and validate the design.

1-1-6. Create and add the UCF file, assigning *S* input to **SW0**, *R* input to **SW1**, *Q* to **LED0**, and *Qbar* to **LED1**.

1-1-7. Re-implement the design.

1-1-8. Generate the bitstream, download it into the Nexys3 board, and verify the functionality.

In some situations it may be desirable to dictate when the latch can and cannot latch. The **gated SR latch** is a simple extension of the SR latch which provides an *Enable* line which must be driven high before data can be latched. Even though a control line is now required, the SR latch is not synchronous, because the inputs can change the output even in the middle of an enable pulse. When the *Enable* input is low, then the outputs from the AND gates must also be low, thus the *Q* and bar *Q* outputs remain latched to the previous data. Only when the *Enable* input is high can the state of the latch change, as shown in the truth table. When the enable line is asserted, a gated SR latch is identical in operation to an SR latch. The *Enable* line is sometimes a clock signal, but is usually a read or writes strobe. The symbol, circuit, and the truth table of the gates SR latch are shown below.

| Enable	S	R	Q	$\overline{Q}$
0	0	0	Latch	
0	0	1	Latch	
0	1	0	Latch	
0	1	1	Latch	
1	0	0	Latch	
1	0	1	0	1
1	1	0	1	0
1	1	1	Metastable	

1-2.   Design a gated SR latch (shown in the figure above) using dataflow modeling. Synthesize the design and view the schematic of the synthesized design. Develop a testbench to test (generate input as shown below) and validate the design. Simulate the design. Assign S input to SW0, R input to SW1, and Enable input to SW2. Assign Q to LED0 and Qbar to LED1. Implement the design and verify the functionality in the hardware.

1-2-1.   Open PlanAhead and create a blank project called lab5_1_2**.**

1-2-2.   Create and add the VHDL module that will model the gated SR latch using dataflow modeling. Assign 2 units delay to each assignment statement used in the model.

1-2-3.   Synthesize the design and view the schematic under the Synthesized Design process group. Verify that it uses 3 LUTs and 5 IOs.

1-2-4.   Implement the design and view the map report. It should show 1LUTs, 1 slice, and 5 IOs.

1-2-5.   Develop a testbench to test and validate the design. It should generate the input stimuli as shown in the figure above.

1-2-6.   Create and add the UCF file, assigning *S* input to **SW0**, *R* input to **SW1**, *Enable* to **SW2**, *Q* to **LED0**, and *Qbar* to **LED1**.

1-2-7.   Re-implement the design.

1-2-8. Generate the bitstream, download it into the Nexys3 board, and verify the functionality.

The **D latch** (D for "data") or **transparent latch** is a simple extension of the gated SR latch that removes the possibility of invalid input states (metastability). Since the gated SR latch allows us to latch the output without using the *S* or *R* inputs, we can remove one of the inputs by driving both the *Set* and *Reset* inputs with a complementary driver, i.e. we remove one input and automatically make it the inverse of the remaining input. The D latch outputs the *D* input whenever the *Enable* line is high, otherwise the output is whatever the *D* input was when the *Enable* input was last high. This is why it is also known as a transparent latch - when *Enable* is asserted, the latch is said to be "transparent" - it signals propagate directly through it as if it isn't there.

| Enable	D	Q	$\overline{Q}$
0	0	Latch	
0	1	Latch	
1	0	0	1
1	1	1	0
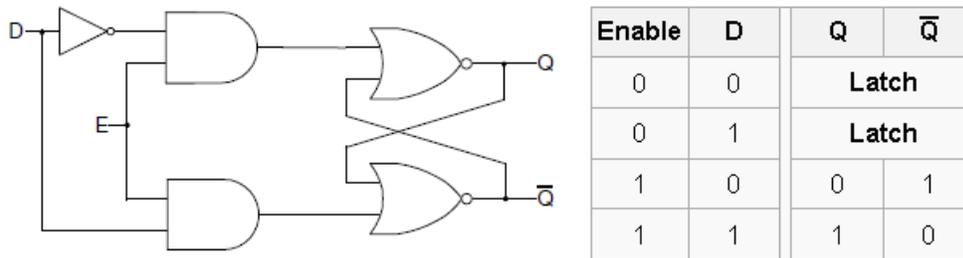
D-latches can be modeled in behavioral modeling as shown below.

```
architecture behavior of D_latch is begin
process (D, Enable) begin
      if (Enable = '1') then
            Q <= D;
            Qbar <= not(D);
      end if;
end process;
end behavior;
```

Note that since we do not say what to do when Enable is low, the circuit "remembers" the previous state. While Enable is high and since the `always` block is also sensitive to D, Q and Qbar will be updated at any time D changes, giving it a "transparent" behavior. Also note that the non-blocking assignment operator (<=) is used instead of blocking (=) operator which had been used in dataflow modeling. The distinction between the blocking and non-blocking assignment is covered in Lab 7 (Testbenches for Sequential Circuits).

1-3. **Design a D latch (shown in the figure above) using dataflow modeling. Synthesize the design and view the schematic of the synthesized design. Develop a testbench to test (generate input as shown below) and validate the design. Simulate the design. Assign D input to SW0, and Enable input to SW1. Assign Q to LED0 and Qbar to LED1. Implement the design and verify the functionality in hardware.**
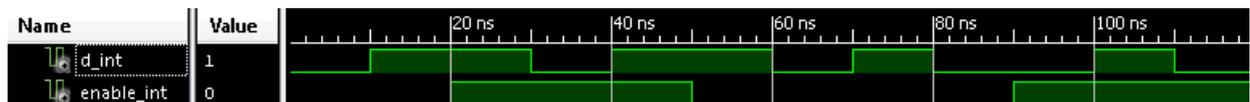
1-3-1. Open PlanAhead and create a blank project called lab5_1_3.

1-3-2. Create and add the VHDL module that will model the D latch using dataflow modeling. Assign 2 units delay to each assignment statement used in the model.

1-3-3. Synthesize the design and view the schematic under the Synthesized Design process group. Verify that it uses 2 LUTs and 4 IOs.

1-3-4. Implement the design and view the map report. It should show 2 LUTs, 1 slice, and 4 IOs.

1-3-5. Develop a testbench to test and validate the design. It should generate the input stimuli as shown in the figure above.

1-3-6. Create and add the UCF file, assigning *D* input to **SW0**, *Enable* input to **SW1**, *Q* to **LED0**, and *Qbar* to **LED1**.

1-3-7. Re-implement the design.

1-3-8. Generate the bitstream, download it into the Nexys3 board, and verify the functionality.

Flip-flops   Part 2

Flip-flops are clocked circuits whose output may change on an active edge of the clock signal based on its input. Unlike latches, which are transparent and in which output can change when the gated signal is asserted upon the input change, flip-flops normally would not change the output upon input change even when the clock signal is asserted. Flip-flops are widely used in synchronous circuits.

The D flip-flop is a widely used type of flip-flop. It is also known as a *data* or *delay* flip-flop. The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell or a delay line. The active edge in a flip-flop could be rising or falling. The following figure shows rising (also called positive) edge triggered D flip-flop and falling (negative edge) triggered D flip-flop.

The positive edge triggered D flip-flop can be modeled using behavioral modeling as shown below.

```
architecture behavior of D_ff is begin
process (clk) begin
      if rising_edge(clk) then
            Q <= D;
      end if;
end process;
end behavior;
```

Note that the `process` block is sensitive to the rising edge on Clk signal. When a change (event) on the sensitive signal occurs, the statements in the *if* block will be executed.

2-1.   Model a D flip-flop using behavioral modeling.   Develop a testbench to validate the model (see diagram below). Simulate the design.
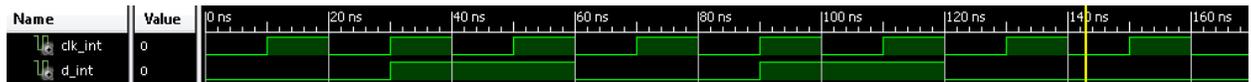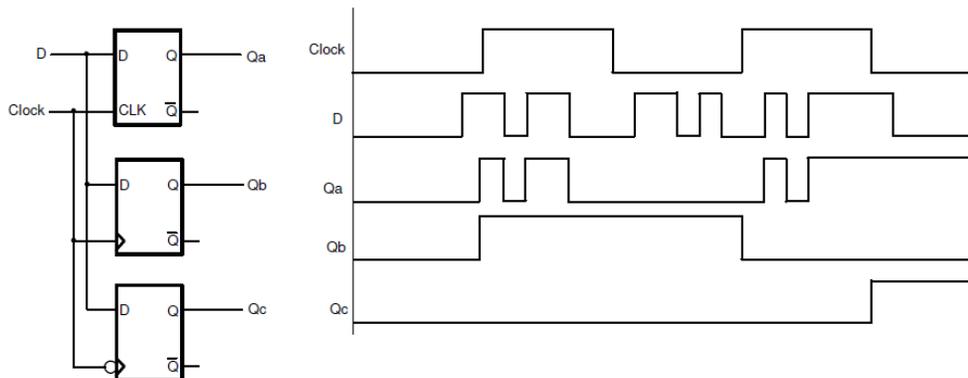

2-1-1.   Open PlanAhead and create a blank project called lab5_2_1**.**

2-1-2.   Create and add the VHDL module that will model simple D flip-flop.

2-1-3.   Develop a testbench to validate the design behavior. It should generate the input stimuli as shown in the <u>above</u> timing diagram.

The following circuit and timing diagrams illustrate the differences between D-latch, rising edge triggered D flip-flop and falling edge triggered D flip-flops.

2-2.   Model the circuit, as shown above, using behavioral modeling.   You will use three *process* blocks. Develop a testbench generating input as shown above. Simulate and validate the design.

2-2-1.   Open PlanAhead and create a blank project called lab5_2_2**.**

2-2-2.   Create and add the VHDL module that will model the given circuit.

2-2-3.   Develop a testbench to test and analyze the design behavior. It should generate the input stimuli as shown in the timing diagram.

Often it is necessary to have the synchronous element to start with a defined output. It is also desired and required in some circuits to force the synchronous element to a known output ignoring input at the D input. The D flip-flop discussed above can be modified to have such functionality. Such D flip-flop is known as D flip-flop with synchronous set and reset capabilities if the desired output is obtained on the active edge of the clock, otherwise it is viewed to have asynchronous preset and clear. The models of each kind are shown below.

XILINX®

```
architecture behavior of D_ff_with_synch_reset is begin
process (clk) begin
      if rising_edge(clk) begin
            if (reset = '1') then
                  Q <= '0';
            else
                  Q <= D;
            end if;
      end if;
end process;
end behavior;

architecture behavior of D_ff_with_asynch_reset is begin
process (clk, clear) begin
      if (clear = '1') then
            Q <= '0';
      elsif rising_edge(clk) then
            Q <= D;
      end if;
end process;
end behavior;
```

**2-3.   Model the D flip-flop with synchronous reset using behavioral modeling.
Develop a testbench to test (generate input as shown) and validate the
design. Simulate the design. Assign D input to SW0, reset to SW1, Clk to
SW2, and output Q to LED0. Verify the design in hardware.**

2-3-1.   Open PlanAhead and create a blank project called lab5_2_3**.**

2-3-2.   Create and add the VHDL module that will model the D flip-flop with synchronous reset.

2-3-3.   Develop a testbench to test and analyze the design behavior. It should generate the input stimuli
as shown in the timing diagram.

2-3-4.   Create and add the UCF file, assigning *D* input to **SW0**, *reset* input to **SW1**, *Clk* to **SW2**, and *Q* to
LED0.

2-3-5.   Synthesize and implement the design.

Look at the Project Summary and note that 1 BUFG and 4 IOs are used. The BUFG is used
because the clock signal is used in the design.

If you open the map report file, you will notice that there are no slice registers used. Instead one
IOB register is used. The reason for this is because the design does not have any other logic and
the model is written properly, so the tools will pack the flip-flop into the output IO.

2-3-6.   Generate the bitstream, download it into the Nexys3 board, and verify the functionality.

In FPGAs, LUT and FF located in different configurable logic blocks (CLB) are connected using routing resources. During implementation, the tools will use these resources depending on the way the circuits are modeled, the type and amount of resources required, and the speed at which the circuit is going to be driven. Often resources used for exchanging information are placed close to each other; however, there can be a situation when it may not be possible. When related flip-flops, between which the information gets exchanged are placed away from each other, the clocks arriving at the source and destination flip-flops may not be at the same time creating what is called clock-skew. The clock-skew can alter the behavior of the circuit. In some other cases, certain flip-flops may not need to update their output at every asserted clock edges. In order to control the behavior, flip-flops in FPGA have an additional control signal called Clock Enable (CE). In ASIC technology, gated clocks are used to control the behavior. A symbol of the flip-flop with CE is shown below.
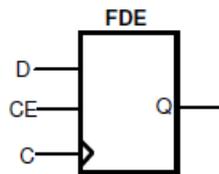

```
architecture behavior of D_ff_with_ce is begin
process (clk) begin
      if rising_edge(clk) then
            if (ce = '1') then
                  Q <= D;
            end if;
      end if;
end process;
end behavior;

architecture behavior of D_ff_with_ce_and_synch_reset is begin
process (clk) begin
      if rising_edge(clk) then
            if (reset = '1') then
                  Q <= '0';
            elsif (ce = '1') then
                  Q <= D;
            end if;
      end if;
end process;
end behavior;
```

2-4.   Model the D flip-flop with synchronous reset and clock enable using behavioral modeling.   Develop a testbench to test (generate input as shown) and validate the design. Simulate the design. Assign D input to SW0, reset to SW1, Clk to SW2, ce to SW3, and output Q to LED0. Verify the design in hardware.
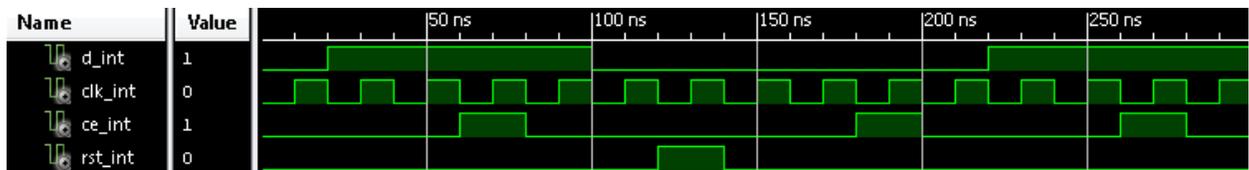
2-4-1. Open PlanAhead and create a blank project called lab5_2_4**.**

�devilXILINX.

2-4-2. Create and add the VHDL module that will model the D flip-flop with synchronous reset and clock enable.

2-4-3. Develop a testbench to test and analyze the design behavior. It should generate the input stimuli as shown in the above timing diagram.

2-4-4. Create and add the UCF file, assigning *D* to **SW0**, *reset* to **SW1**, *Clk* to **SW2**, *ce* to **SW3**, and *Q* to **LED0**.

2-4-5. Synthesize and implement the design.

Look at the Project Summary and note that 1 BUFG and 5 IOs are used. The BUFG is used because the clock signal is used in the design.

If you open the map report file, you will notice that there are no slice registers used. Instead one IOB register is used. The reason for this is because the design does not have any other logic and the model is written properly, so the tools will pack the flip-flop into the output IOB.

2-4-6. Generate the bitstream, download it into the Nexys3 board, and verify the functionality.

In digital circuits, another kind of flip-flop, called T or Toggle, is used to implement clock divider circuits. It can be used to divide the input by 2. If more than one T flip-flop is cascaded then the clock division can be 2 power of the number of flip-flops used. The T flip-flop has a T input (data), a clock input, and optionally reset and enable control signals.

```
architecture behavior of T_ff is begin
process (clk) begin
      if falling_edge(clk) then
            Q <= not(Q);
      end if;
end process;
end behavior;
```

The T flip-flop can also have a control signal called CE (clock enable) which will allow clock division to take place only when it is asserted. The following code models the functionality of the T flip-flop that is sensitive to a falling edge of clock and has active-low reset and active-high T control signals.

```
architecture behavior of T_ff_enable is begin
process (clk) begin
      if falling_edge(clk) then
            if (enable = '1') and (T = '1') then
                  Q <= not(Q);
            else
                  Q <= Q;
            end if;
      end if;
end process;
end behavior;
```

2-5.   Model a T flip-flop with synchronous negative-logic reset and clock enable using the above code.   Assign T input to SW0, reset_n to SW1, Clk to SW2, and output Q to LED0. Verify the design in hardware.

2-5-1.   Open PlanAhead and create a blank project lab5_2_5**.**

2-5-2.   Create and add the VHDL module that will model the D flip-flop with synchronous reset and clock enable.

2-5-3.   Create and add the UCF file, assigning *T* input to **SW0**, *enable* input to **SW1**, *Clk* to **SW2**, and *Q* to **LED0**.

2-5-4.   Synthesize and implement the design.

2-5-5.   Generate the bitstream, download it into the Nexys3 board, and verify the functionality.

Conclusion

In this lab, you learned the functionality of various kinds of latches and flip-flops. You modeled and verified the functionality of these components. Xilinx also provides some basic latches and flip-flops library components which a designer can instantiate and use instead of writing a model. Writing a model provides portability across vendors and technologies whereas instantiating library components enable a quick use of a component without re-inventing the wheel.

Here is the table that lists some of the the functionality and the corresponding library component supported by the Xilinx Synthesis Tools (XST). Please refer to Spartan6_hdl.pdf file for more components and instantiation examples.

FDCE	D Flip-Flop with Asynchronous Clear and Clock Enable
FDPE	D Flip-Flop with Asynchronous Preset and Clock Enable
FDRE	D Flip-Flop with Synchronous Reset and Clock Enable
LDCE	Transparent Latch with Asynchronous Clear and gate Enable
LDPE	Transparent Latch with Asynchronous Preset and gate Enable

XILINX®