CRN: 22840

<u>CET3136 – Logic Devices Programming</u>

Spring 2025

Experiment 7

Latches and Flip-flops

Performed By:

Anthony Paul Sevarino

Submitted to:

Prof. Ashley Evans

Department of Electrical & Computer Engineering Technology (ECET)
School of Engineering, Technology, and Advanced Manufacturing (ETAM)
Valencia College

Date Submitted

03/06/2025

Introduction

The objective of this lab is to create a four-bit ripple carry adder, using the switches and LEDs on the MAX10 development board. While the experiment statement explains that any coding style may be used to perform this task, it specifies that the program should be written as efficiently as possible. Thus, a combination of dataflow and structural coding styles will be used. The use of Karnaugh mapping will also assist in the design of the individual full-adder segment to be reused for each component of the four-bit full adder. Note that the fundamental function of a four-bit ripple carry adder is to add 2 four-bit binary values together, with an optional carry value added when desired. The result should display as a four bit binary value, with a fifth bit reserved for the carry value (overflow).

The objective of this experiment is to create a series of latches and flip-flops of different parameters in VHDL for integration onto the MAX10DA Intel Development board. Both behavioral and structural coding styles will be used in the 3 parts which make up this lab. The first part of the experiment will deal with creating an inverse clock D latch using only the behavioral coding style. The second part takes the previous latch constructed in part A and creates a parent/child latch, now implementing the structural coding style to combine the latch components. Finally, part C of the experiment takes the part A, D latch, and requires both a positive-edge triggered and negative-edge triggered flip-flop to create a 3 component circuit, using the structural coding style.

LIST OF EQUIPMENT/PARTS/COMPONENTS/SOFTWARE

- DE10-Lite MAX-10 Dev Board
- USB 2.0 Type B Cable
- Windows Desktop
- Quartus FPGA Design Software

PROCEDURE / DISCUSSION

Part A – The D Latch

First, creating a logic schematic for a D latch will aid in designing the VHDL program efficiently and correctly. Below is such a schematic, provided by the Laboratory Exercise Manual[1].

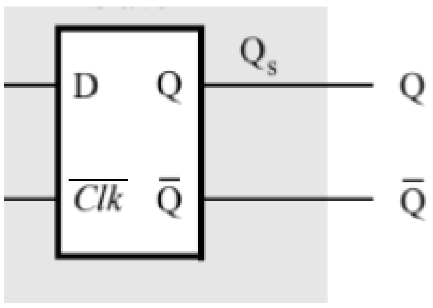
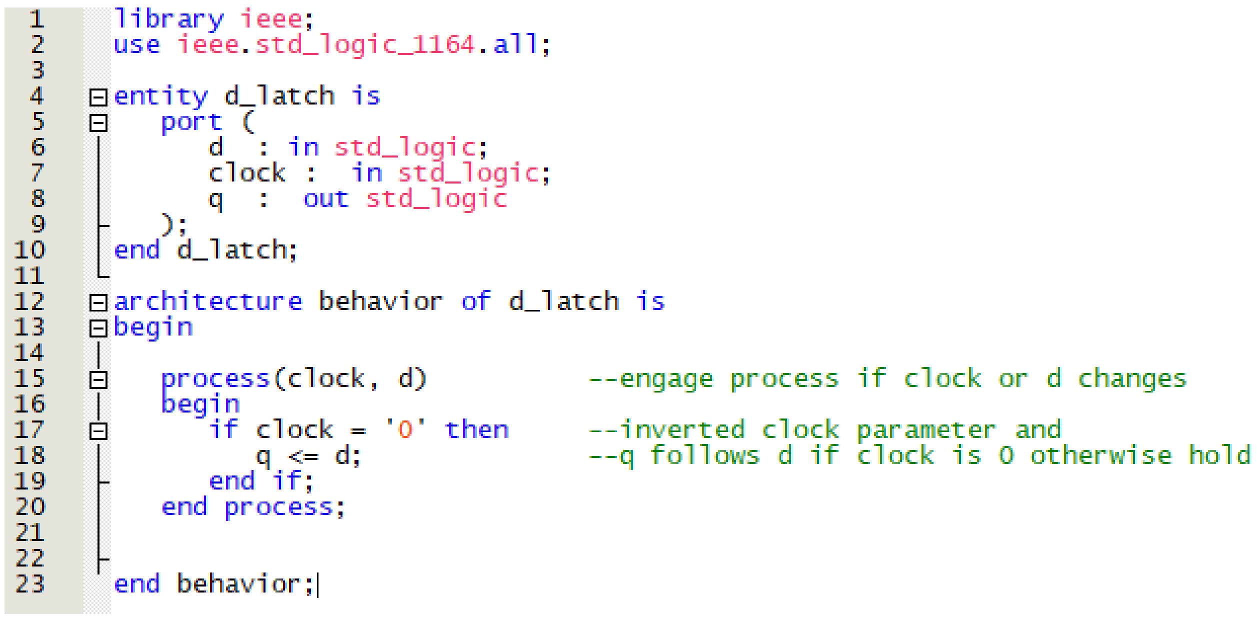


Figure 1 - Logic Schematic for the D Latch

In a typical D-Latch, there is a data input *D*, alongside a clock signal acting as its two inputs, with a single output *q*. Contrary to a Flip-flop, a Latch operates on level-sensitive operation, which leads to a "transparent" behavior of the latch when the clock value is logic high. This means that the output *q* matches the value from the data input *D* while the clock is enabled. When the clock is logic low, the output *q* holds the last "seen" value of *D*, until the clock is reenabled at which point *q* once again follows *d*.

Code Explanation Part 1



```
library ieee;
use ieee.std_logic_1164.all;

entity d_latch is
    port (
        d  : in std_logic;
        clock :  in std_logic;
        q  :  out std_logic
    );
end d_latch;

architecture behavior of d_latch is
begin

    process(clock, d)           --engage process if clock or d changes
    begin
        if clock = '0' then     --inverted clock parameter and
            q <= d;             --q follows d if clock is 0 otherwise hold
        end if;
    end process;


end behavior;
```

Figure 2 - Completed Code for D Latch

```
library ieee;
use ieee.std_logic_1164.all;

entity d_latch is
	port (
		d	: in std_logic;
		clock	:	in std_logic;
		q	:	out std_logic
	);
end d_latch;

architecture behavior of d_latch is
begin

	process(clock, d)				--engage process if clock or d changes
	begin
		if clock = '0' then		--inverted clock parameter and
			q <= d;				--q follows d if clock is 0 otherwise hold
		end if;
```

```
    end process;

end behavior;
```

The entity portion of this code takes two inputs *d* and *clock* as standard logic, and one output *q* as standard logic. (See previous explanation for naming conventions.)

The architecture begins with a sensitivity list process which accepts *clock*, then *d* as parameters. This means that any time either of the two inputs changes, the process will run. Within this process, a conditional if statement is used to check if *clock* input is logic low (recall from the schematic that the clock signal is inverted for this Latch.) If so, then *q* becomes the value *d*.

Below is the block diagram and pin assignments for this part of the experiment:

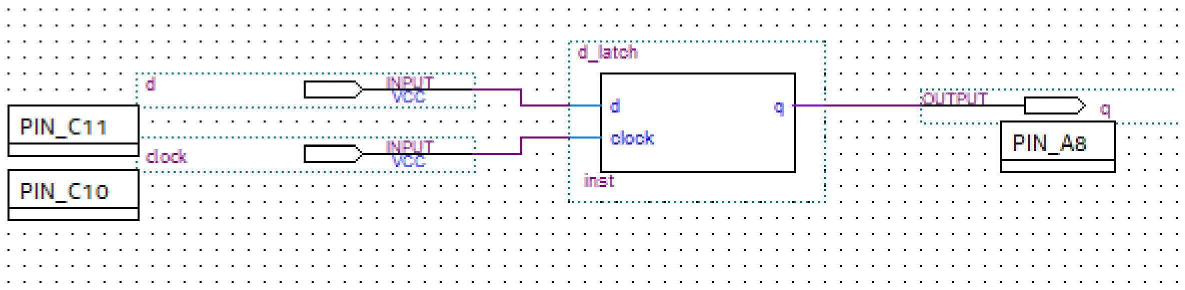
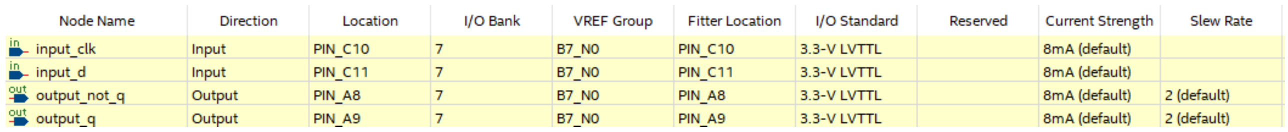


Figure 3 - Part B Block Diagram



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
input_clk	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)	
input_d	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)	
output_not_q	Output	PIN_A8	7	B7_N0	PIN_A8	3.3-V LVTTL		8mA (default)	2 (default)
output_q	Output	PIN_A9	7	B7_N0	PIN_A9	3.3-V LVTTL		8mA (default)	2 (default)

Figure 4 - Pin Assignments for Part B

Part B – The Parent/Child D Latch

Luckily, the D Latch was completed in the **Part A** of the study, and this part of the lab calls for a parent-child D Latch, where two latches are used in series. An example provided by the Lab Experiments Manual[1] is provided below:

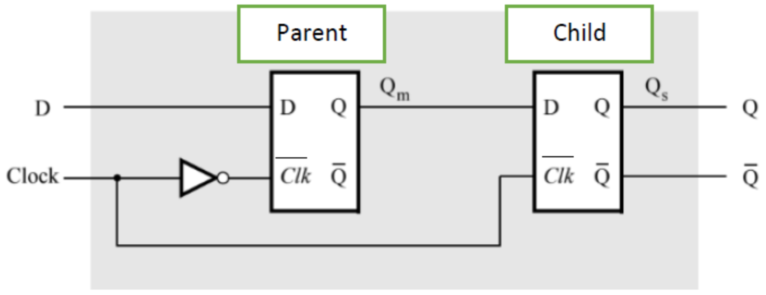
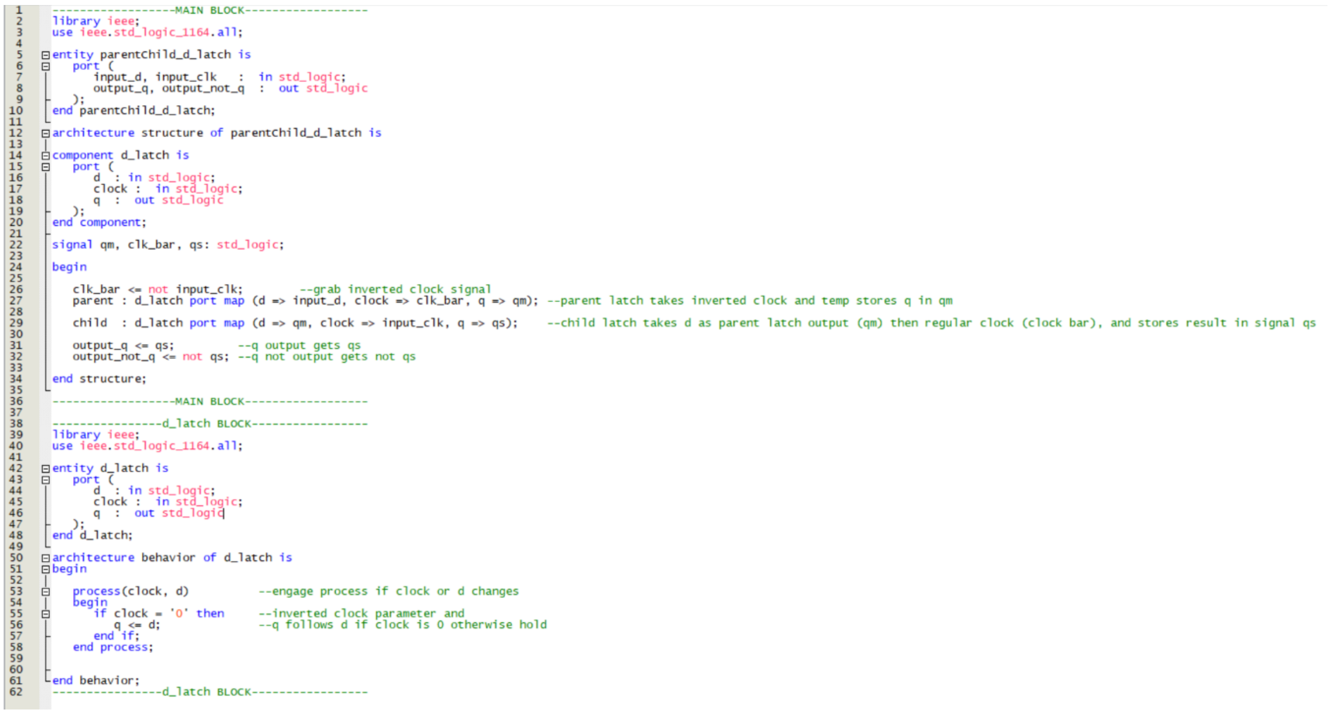


Figure 5 – Parent/Child D Latch Schematic

Also note that in this circuit, the inverted clock input on the D Latch is itself inverted from the clock signal in the parent latch, therefore it accepts a logic HIGH clock as its parameter for transparency.

Following the logic flow, Q_m will follow D if the clock signal is high. Then, the child latch will accept Q_m as its input *D*, which the Child's Q_s output will receive once the clock is low, or alternates from the parent's latch parameter.

Code Explanation Part 2



```
------------------MAIN BLOCK------------------
library ieee;
use ieee.std_logic_1164.all;

entity parentChild_d_latch is
   port (
      input_d, input_clk    :  in std_logic;
      output_q, output_not_q  :  out std_logic
   );
end parentChild_d_latch;

architecture structure of parentChild_d_latch is

component d_latch is
   port (
      d   : in std_logic;
      clock :  in std_logic;
      q  :  out std_logic
   );
end component;

signal qm, clk_bar, qs: std_logic;

begin

   clk_bar <= not input_clk;         --grab inverted clock signal
   parent : d_latch port map (d => input_d, clock => clk_bar, q => qm); --parent latch takes inverted clock and temp stores q in qm

   child  : d_latch port map (d => qm, clock => input_clk, q => qs);    --child latch takes d as parent latch output (qm) then regular clock (clock bar), and stores result in signal qs

   output_q <= qs;          --q output gets qs
   output_not_q <= not qs; --q not output gets not qs

end structure;

------------------MAIN BLOCK------------------

----------------d_latch BLOCK----------------
library ieee;
use ieee.std_logic_1164.all;

entity d_latch is
   port (
      d   : in std_logic;
      clock :  in std_logic;
      q  :  out std_logic
   );
end d_latch;

architecture behavior of d_latch is
begin

   process(clock, d)          --engage process if clock or d changes
   begin
      if clock = '0' then     --inverted clock parameter and
         q <= d;              --q follows d if clock is 0 otherwise hold
      end if;
   end process;


end behavior;
----------------d_latch BLOCK----------------
```

Figure 6 - Completed Code for D Latch

```
------------------MAIN BLOCK------------------
library ieee;
use ieee.std_logic_1164.all;

entity parentChild_d_latch is
		port (
				input_d, input_clk		:		in std_logic;
				output_q, output_not_q	:		out std_logic
		);
end parentChild_d_latch;

architecture structure of parentChild_d_latch is

component d_latch is
		port (
				d		: in std_logic;
				clock	:		in std_logic;
				q		:		out std_logic
		);
end component;

signal qm, clk_bar, qs:	std_logic;

begin

	clk_bar <= not input_clk;			--grab inverted clock signal
	parent : d_latch port map (d => input_d, clock => clk_bar, q => qm);  --parent latch takes inverted clock 
and temp stores q in qm

	child	:	d_latch port map (d => qm, clock => input_clk, q => qs);		--child latch takes d 
as parent latch output (qm) then regular clock (clock bar), and stores result in signal qs

	output_q <= qs;			--q output gets qs
	output_not_q <= not qs;	--q not output gets not qs

end structure;

------------------MAIN BLOCK------------------

----------------d_latch BLOCK-----------------
library ieee;
use ieee.std_logic_1164.all;

entity d_latch is
	port (
		d	: in std_logic;
		clock	:	in std_logic;
		q	:	out std_logic
	);
end d_latch;

architecture behavior of d_latch is
begin

	process(clock, d)					--engage process if clock or d changes
	begin
		if clock = '0' then		--inverted clock parameter and
			q <= d;					--q follows d if clock is 0 otherwise hold
		end if;
	end process;


end behavior;
----------------d_latch BLOCK-----------------
```

The entity portion of this code takes two inputs *input_d* and *input_clk* as standard logic, and two outputs *output_q* and *output_not_q* as standard logic. (See previous explanation for naming conventions.)

The architecture begins with instantiating the previously constructed D Latch as a component, then defining three signal statements to be used in the main body of the architecture, those being *qm*, *qs*, and *clk_bar* to create the inverted clock signal.

Then, the architecture logic begins by assigning a value to *clk_bar* (by using the not of the clock input.) Then the parent and child components are constructed using port maps, and finally outputs are assigned according to the signals used to store the output values in each of the components.

Below is the block diagram for this part of the experiment:

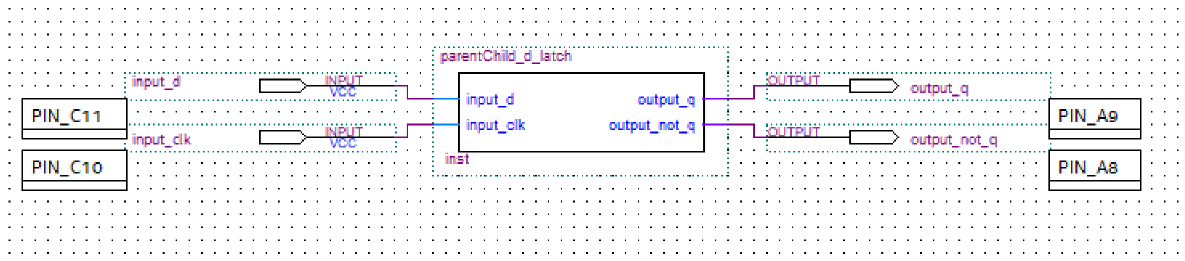


Figure 7 - Part B Block Diagram

Part C – The 3D Flip-Flop

In this section of the experiment, two new but similar components must be constructed, those being D Flip-flops.

This Flip-flop takes two inputs, *D* as data and *CLK* as clock, and has one primary output as *Q*. In a positive-edge triggered flip-flop, *Q* is updated to be *D* when the clock moves from LOW to HIGH logic. Therefore, even if the clock is logic HIGH, *Q* will not be updated to *D* until another ‘rising edge’ appears in the clock. Conversely, in a negative-edge triggered flip flop, the same behavior occurs only when the clock arrives at a ‘falling-edge’, or when the clock goes from logic HIGH to logic LOW. Below is the provided schematic for this circuit:

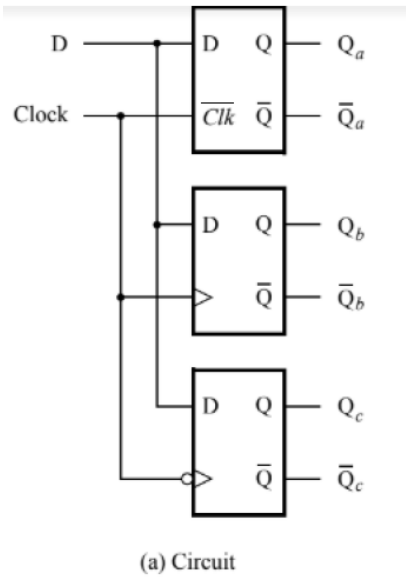
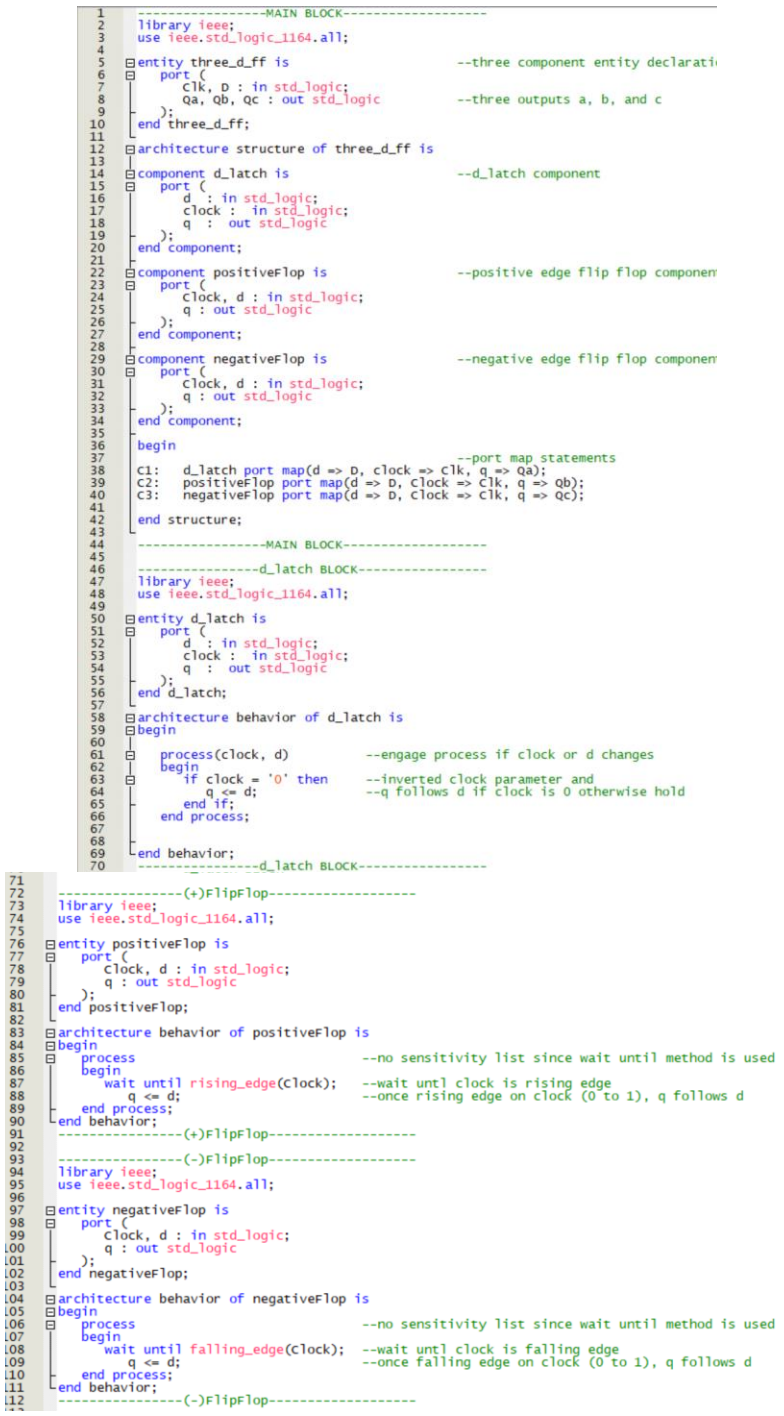


Figure 8 – Parent/Child D Latch Schematic

Note that the only difference between the positive and negative edge triggered flip-flops is the inverter symbol placed on the clock input of the negative-edge triggered flip-flop.

Code Explanation Part 3



```
-----------------MAIN BLOCK-------------------
library ieee;
use ieee.std_logic_1164.all;

entity three_d_ff is                            --three component entity declarati
    port (
        Clk, D : in std_logic;
        Qa, Qb, Qc : out std_logic              --three outputs a, b, and c
    );
end three_d_ff;

architecture structure of three_d_ff is

component d_latch is                            --d_latch component
    port (
        d   : in std_logic;
        clock :  in std_logic;
        q   :  out std_logic
    );
end component;

component positiveFlop is                       --positive edge flip flop componen
    port (
        Clock, d : in std_logic;
        q : out std_logic
    );
end component;

component negativeFlop is                       --negative edge flip flop componen
    port (
        Clock, d : in std_logic;
        q : out std_logic
    );
end component;

begin
                                                --port map statements
C1:   d_latch port map(d => D, clock => Clk, q => Qa);
C2:   positiveFlop port map(d => D, Clock => Clk, q => Qb);
C3:   negativeFlop port map(d => D, Clock => Clk, q => Qc);

end structure;

-----------------MAIN BLOCK-------------------

----------------d_latch BLOCK-----------------
library ieee;
use ieee.std_logic_1164.all;

entity d_latch is
    port (
        d   : in std_logic;
        clock :  in std_logic;
        q   :  out std_logic
    );
end d_latch;

architecture behavior of d_latch is
begin

    process(clock, d)           --engage process if clock or d changes
    begin
        if clock = '0' then     --inverted clock parameter and
            q <= d;             --q follows d if clock is 0 otherwise hold
        end if;
    end process;


end behavior;
----------------d_latch BLOCK-----------------

----------------(+)FlipFlop-------------------
library ieee;
use ieee.std_logic_1164.all;

entity positiveFlop is
    port (
        Clock, d : in std_logic;
        q : out std_logic
    );
end positiveFlop;

architecture behavior of positiveFlop is
begin
    process                               --no sensitivity list since wait until method is used
    begin
        wait until rising_edge(Clock);    --wait untl clock is rising edge
            q <= d;                       --once rising edge on clock (0 to 1), q follows d
    end process;
end behavior;
----------------(+)FlipFlop-------------------

----------------(-)FlipFlop-------------------
library ieee;
use ieee.std_logic_1164.all;

entity negativeFlop is
    port (
        Clock, d : in std_logic;
        q : out std_logic
    );
end negativeFlop;

architecture behavior of negativeFlop is
begin
    process                               --no sensitivity list since wait until method is used
    begin
        wait until falling_edge(Clock);   --wait untl clock is falling edge
            q <= d;                       --once falling edge on clock (0 to 1), q follows d
    end process;
end behavior;
----------------(-)FlipFlop-------------------
```

Figure 8 - Completed Code for Section C of the Experiment

-----------------MAIN BLOCK-------------------
library ieee;
use ieee.std_logic_1164.all;

```
entity three_d_ff is                                                                     --three component entity
declaration
            port (
                        Clk, D : in std_logic;
                        Qa, Qb, Qc : out std_logic                                 --three outputs a, b, and c
            );
end three_d_ff;

architecture structure of three_d_ff is

component d_latch is                                                                     --d_latch component
            port (
                        d           : in std_logic;
                        clock     :           in std_logic;
                        q           :           out std_logic
            );
end component;

component positiveFlop is                                                    --positive edge flip flop component
            port (
                        Clock, d : in std_logic;
                        q : out std_logic
            );
end component;

component negativeFlop is                                                    --negative edge flip flop component
            port (
                        Clock, d : in std_logic;
                        q : out std_logic
            );
end component;

begin

            --port map statements
C1:         d_latch port map(d => D, clock => Clk, q => Qa);
C2:         positiveFlop port map(d => D, Clock => Clk, q => Qb);
C3:         negativeFlop port map(d => D, Clock => Clk, q => Qc);

end structure;

-----------------MAIN BLOCK-------------------

----------------d_latch BLOCK-----------------
library ieee;
use ieee.std_logic_1164.all;

entity d_latch is
            port (
                        d           : in std_logic;
                        clock     :           in std_logic;
                        q           :           out std_logic
            );
end d_latch;

architecture behavior of d_latch is
```

```
begin

            process(clock, d)                                           --engage process if clock or d changes
            begin
                        if clock = '0' then              --inverted clock parameter and
                                    q <= d;                                             --q follows d if clock is 0 otherwise hold
                        end if;
            end process;


end behavior;
----------------d_latch BLOCK-----------------

----------------(+)FlipFlop-------------------
library ieee;
use ieee.std_logic_1164.all;

entity positiveFlop is
            port (
                        Clock, d : in std_logic;
                        q : out std_logic
            );
end positiveFlop;

architecture behavior of positiveFlop is
begin
            process                                                                                             --no sensitivity list
since wait until method is used
            begin
                        wait until rising_edge(Clock);        --wait untl clock is rising edge
                                    q <= d;                                                                         --once rising edge
on clock (0 to 1), q follows d
            end process;
end behavior;
----------------(+)FlipFlop-------------------

----------------(-)FlipFlop-------------------
library ieee;
use ieee.std_logic_1164.all;

entity negativeFlop is
            port (
                        Clock, d : in std_logic;
                        q : out std_logic
            );
end negativeFlop;

architecture behavior of negativeFlop is
begin
            process                                                                                             --no sensitivity list
since wait until method is used
            begin
                        wait until falling_edge(Clock);       --wait untl clock is falling edge
                                    q <= d;                                                                         --once falling edge
on clock (0 to 1), q follows d
            end process;
end behavior;
```

----------------(-)FlipFlop------------------

The entity portion of this code takes two inputs *Clk* and *D* as standard logic, and three outputs *Qa*, *Qb, and Qc* as standard logic. (See previous explanation for naming conventions.)

The architecture begins with instantiating the previously constructed D Latch as a component, as well as the two new flip-flops. Next, the components are instantiated using a port map statement for each. Since none of the inputs of the three components are dependent on outputs from other components, no local signals are required for circuit functionality.

Analyzing the new flip-flop blocks, in this case the positive-edge triggered flip-flop, the entity defines two inputs, *Clock* and *d*, as well as a single output *q*. In its architecture, a process without a sensitivity list begins. Since there is no sensitivity list, a *wait until* statement can be used, which prevent continuing through the program until a condition is met, which in this case is for clock to arrive at a rising edge. Since this method is used rather than a single if statement, the output *q* will only change at certain intervals of the clock, rather than every time the data input *d* is switched. Note that this behavior is the same for the negative-edge triggered flip-flop but with a falling edge.

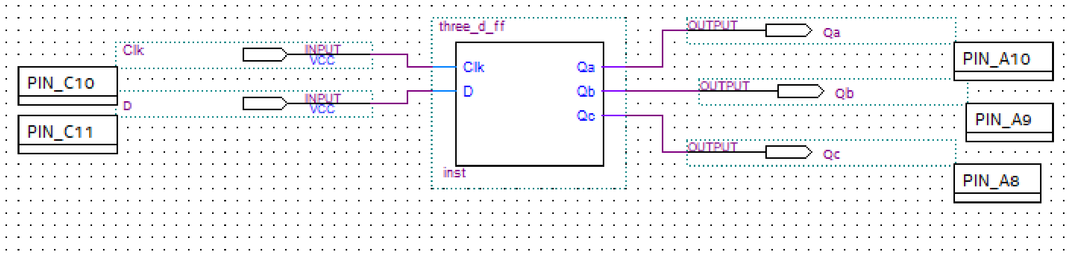
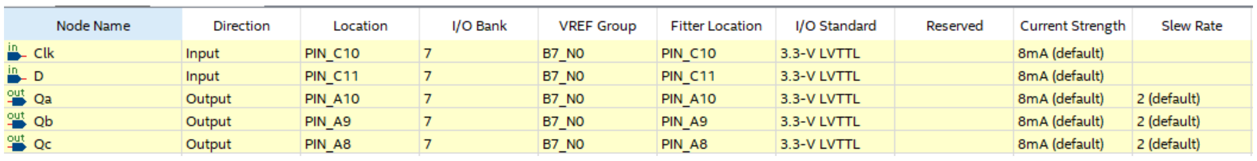


Figure 9 - Part C Block Diagram



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
Clk	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)	
D	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)	
Qa	Output	PIN_A10	7	B7_N0	PIN_A10	3.3-V LVTTL		8mA (default)	2 (default)
Qb	Output	PIN_A9	7	B7_N0	PIN_A9	3.3-V LVTTL		8mA (default)	2 (default)
Qc	Output	PIN_A8	7	B7_N0	PIN_A8	3.3-V LVTTL		8mA (default)	2 (default)

Figure 10 - Pin Assignments for Part C

VALIDATION OF DATA

Functional Simulation Analysis

Below are the functional simulation analysis results for each part of the Lab, labelled accordingly. Note that these simulations were conducted in Modelsim-Altera.

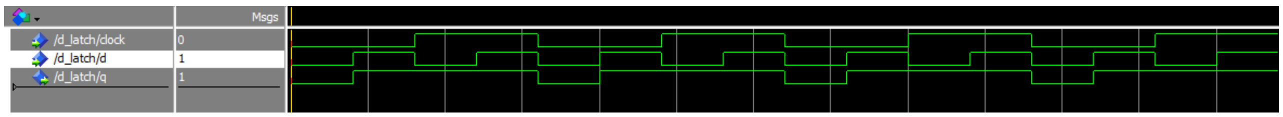


Figure 11 – Part A Functional Simulation

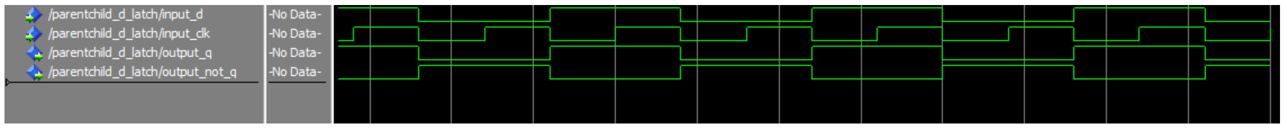

/parentchild_d_latch/input_d
/parentchild_d_latch/input_clk
/parentchild_d_latch/output_q
/parentchild_d_latch/output_not_q
-No Data-
-No Data-
-No Data-
-No Data-

Figure 12- Part B Functional Simulation

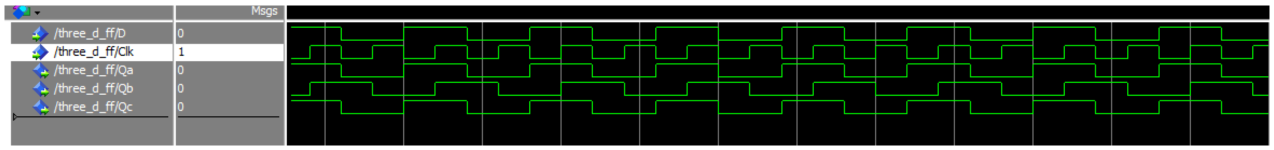


Figure 13- Part C Functional Simulation

CONCLUSION

The experiment was conducted successfully and efficiently demonstrated the creation, function, and use of both a D-Latch and positive and negative-edge triggered flip-flops. Understanding the fundamental behavior of both a latch and a flip-flop aides greatly in the developing the functionality of the program, where a latch operates on level-triggered operation and a flip-flop on edge-triggered operation. Utilizing the structural coding style in parts B and C of the experiment allowed for the production of more modular and reactive code, rather than creating a large main block which would likely be prone to errors, and difficult to follow both for the code producer and code readers. The behavioral coding style is superior in each of the specific components over the dataflow coding style as much of their operations are condition based, which can be difficult to work with when dealing strictly with Boolean operations. If this experiment were to be conducted again, it would prove beneficial to construct systems wherein these components are dependent on each other's outputs, making a reactive and interesting circuit.

REFERENCES

[1] Professor Ashley Evans, *Logic Devices Programming Lab Manual*, 1st ed. Orlando, FL: Valencia College, 2025.