# LAB #4

# Display the output of sequential circuit using VHDL programming techniques

## Objective

- To learn to design sequential circuits in VHDL
- To learn sequential circuit constructs in VHDL
- To observe the behaviour of D-Latch and Flip-Flop

## Pre-Lab

### Familiarize yourself with Sequential Circuit

### Introduction:

A sequential circuit is a circuit with *memory*, which forms the internal state of the circuit. Unlike a combinational circuit, in which the output is a function of input only, the output of a sequential circuit is a function of the input and the internal state. The synchronous design methodology is the most commonly used practice in designing a sequential circuit. In this methodology, all storage elements are controlled (i.e., synchronized) by a global ***clock*** signal and the data is sampled and stored at the ***rising*** or ***falling edge*** of the clock signal. It allows designers to separate the storage components from the circuit and greatly simplifies the development process. This methodology is the most important principle in developing a large, complex digital system and is the foundation of most synthesis, verification, and testing algorithms.

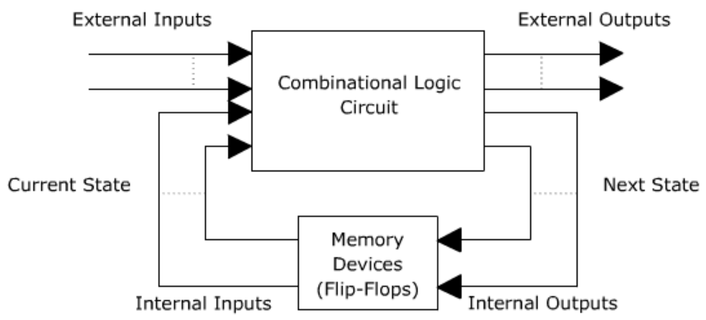


*Figure 1 Sequential Circuit*

### Types of Sequential Circuit:

There are two types of sequential circuit, **synchronous** and **asynchronous**.

### Synchronous Circuit:

**Synchronous** types use pulsed or level inputs and a clock input to drive the circuit (with restrictions on pulse width and circuit propagation).

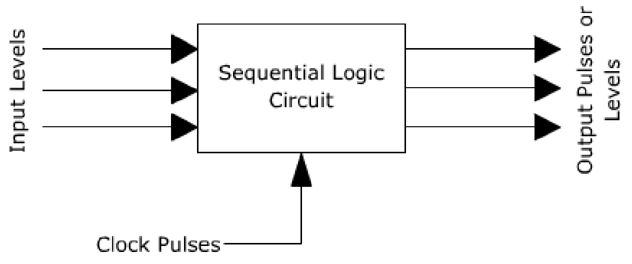


*Figure 2 Synchronous circuit*

## Asynchronous Circuit:

**Asynchronous** sequential circuits do not use a clock signal as synchronous circuits do. Instead, the circuit is driven by the pulses of the inputs. You will not need to know any more about asynchronous circuits for this course.

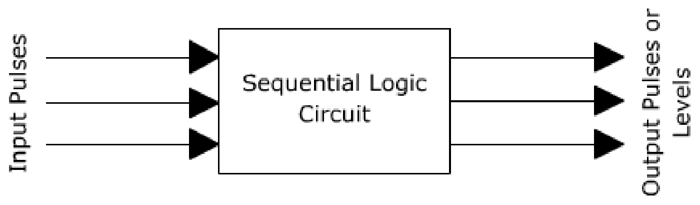


*Figure 3 Asynchronous Circuit*

## Sequential circuit coding in VHDL

In VHDL, concurrent code is intended only for the design of combinational circuits, while sequential code can be used indistinctly to design both sequential and combinational circuits. The statements intended only for completely concurrent code, referred to as concurrent statements, are WHEN, SELECT, and GENERATE (Lab 1 to Lab 3), while those for sequential code, referred to as sequential statements, are IF, WAIT, LOOP, and CASE.
In VHDL, there are three kinds of sequential code:

i. PROCESS
ii. FUNCTION (subprograms)
iii. PROCEDURE (subprograms)

## PROCESS:

PROCESS is intended for the architecture body (main code, for example), while subprograms, being intended mainly for libraries (Lab 5), which deals with system-level code. ***Remember*** that a PROCESS or a subprogram call is a concurrent statement.

**PROCESS** is a sequential section of VHDL code, located in the statements part of an architecture. Inside it, only sequential statements (IF, WAIT, LOOP, CASE) are allowed. A simplified syntax is shown below

```
[label:] PROCESS [(sensitivity_list)] [IS]
   [declarative_part]
BEGIN
  sequential_statements_part
END PROCESS [label];
```

- The **label**, whose purpose is to improve readability in long codes, is optional.
- The **sensitivity list** is mandatory (but is forbidden when WAIT is used), and causes the process to be run every time a signal in the list changes (or the condition associated with WAIT is fulfilled).
- The **declarative part** of PROCESS can contain the following:
  - subprogram declaration
  - subprogram body
  - type declaration

- subtype declaration
- constant declaration
- variable declaration
- file declaration
- alias declaration
- attribute declaration
- attribute specification
- use clause
- group template declaration
- group declaration.
- ***Signal declaration is not allowed*, while variable is by far the most common declaration**

- The **statements part** of PROCESS, only sequential statements are allowed (besides operators as they can go in any kind of code).

A crucial point when dealing with sequential code is to fully understand the difference between SIGNAL and VARIABLE.

**Main properties of SIGNAL:**

- A signal can only be declared outside sequential code (though it can be used there).
- A signal is not updated immediately (when a value is assigned to a signal inside sequential code, the new value will only be ready after the conclusion of that run).
- A signal assignment, when made at the transition of another signal, will cause the inference of registers (given that the signal affects the design entity).
- Only a single assignment is allowed to a signal in the whole code (even though the compiler might accept multiple assignments to the same signal in PROCESS or subprograms, only the last one will be effective, so again it is just one assignment).

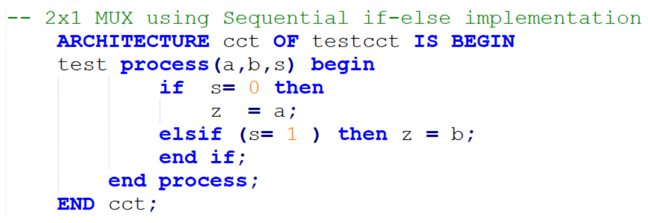
**Main properties of VARIABLE:**

- A variable can only be declared and used inside a PROCESS or subprogram (if it is a shared variable, then the declaration is made elsewhere, but it still should only be modified inside a sequential unit).
- A variable is updated immediately (hence the new value can be used/tested in the next line of code).
- A variable assignment, when made at the transition of another signal, will cause the inference of registers (assuming that the variable's value affects a signal, which in turn affects the design entity).
- Multiple assignments are fine.

## Sequential statements

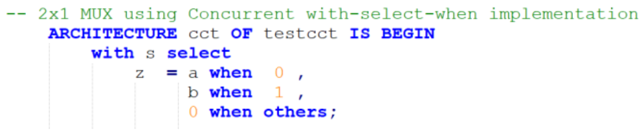
### a) If-Then-Else statements

The if-then-else statements have the same meaning in VHDL as they do in the C-language. By comparison with the combinational statements, these statements are the sequential equivalents of the ***with-select-when*** and ***when-else statements***.



```
-- 2x1 MUX using Sequential if-else implementation
    ARCHITECTURE cct OF testcct IS BEGIN
    test process(a,b,s) begin
            if  s= 0 then
                z  = a;
            elsif (s= 1 ) then z = b;
            end if;
        end process;
    END cct;
```

*Figure 4 2x1 MUX using if-else statement*



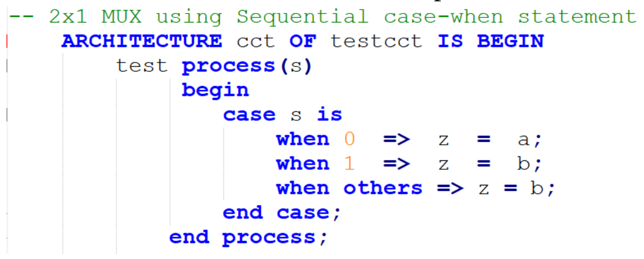
```
-- 2x1 MUX using Concurrent with-select-when implementation
    ARCHITECTURE cct OF testcct IS BEGIN
        with s select
            z  = a when  0 ,
                b when  1 ,
                0 when others;
```

*Figure 5 4x1 MUX concurrent implementation*

### b) Case-When Statements

A **case-when** statement is the sequential equivalent of a **with-select-when** statement. Figure 6 shows the case-when equivalent of the simple, single-bit multiplexor-type circuit of Figures 4 and 5.

Note that this code must account for the other possible values of the signal s.



```
-- 2x1 MUX using Sequential case-when statement
    ARCHITECTURE cct OF testcct IS BEGIN
        test process(s)
            begin
                case s is
                    when 0  =>  z  =  a;
                    when 1  =>  z  =  b;
                    when others => z = b;
                end case;
            end process;
```

*Figure 6 2x1 MUX using case-when statement*

## Pre-Lab Tasks

### Latches and Flip-Flops

### D Latch:

Latch is an electronic device that can be used to store one bit of information. The D latch is used to capture, or 'latch' the logic level which is present on the Data line when the clock input is high. If the data on the D line changes state while the clock pulse is high, then the output, Q, follows the input, D. When the CLK input falls to logic 0, the last state of the D input is trapped and held in the latch.

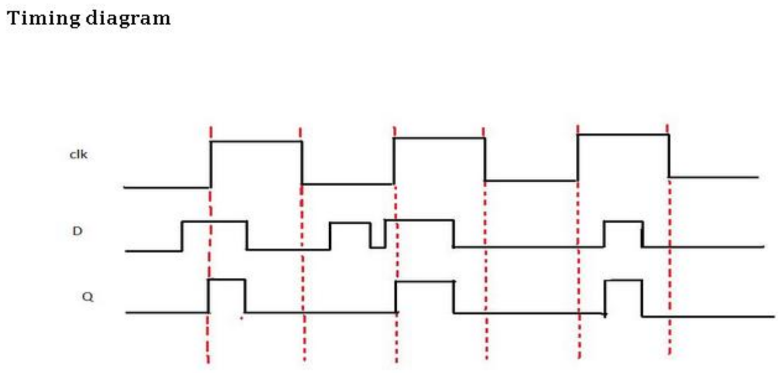
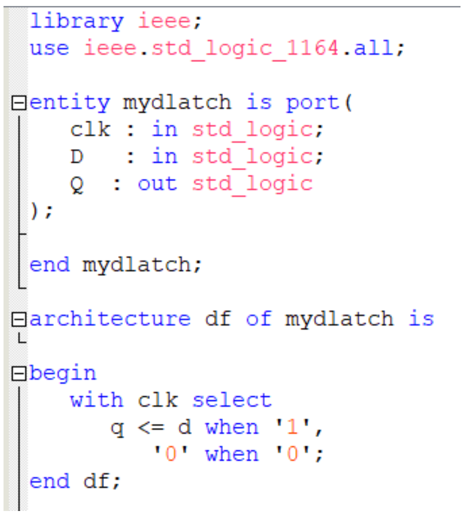


*Figure 7 Timing of D Latch*

From Figure 7, we can see that the input D is transferred to output Q when the clock remains at logic ‘1’. This is the true behaviour of a level triggered circuit.

## Pre-Lab Task1: Create and observe the waveform of D-latch

For pre-lab task1, you are required to make a waveform of the code given in Figure 8 and compare it with waveform of Figure 7.



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

entity mydlatch is port(
    clk : in std_logic;
    D   : in std_logic;
    Q  : out std_logic
);

end mydlatch;

architecture df of mydlatch is

begin
    with clk select
        q <= d when '1',
           '0' when '0';
end df;
```

*Figure 8 Code for D Latch*

### D Flip-Flop:

The working of D flip flop is similar to the D latch except that the output of D Flip Flop takes the state of the D input at the moment of a positive edge at the clock pin (or negative edge if the clock input is active low) and delays it by one clock cycle.

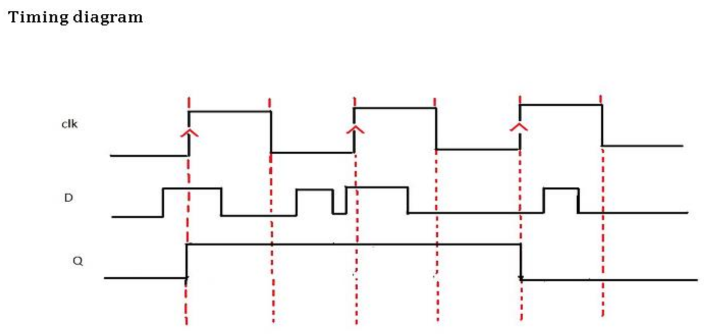
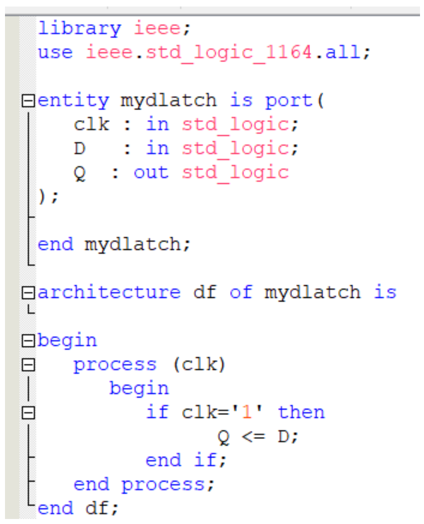


*Figure 9 Timing of D Flip-Flop*

## Pre-Lab Task2: Create and observe the waveform of D Flip-Flop

For pre-lab task2, you are required to make a waveform of the code given in Figure 10 and compare it with waveform of Figure 9.



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

entity mydlatch is port(
    clk : in std_logic;
    D   : in std_logic;
    Q  : out std_logic
);

end mydlatch;

architecture df of mydlatch is

begin
    process (clk)
        begin
            if clk='1' then
                  Q <= D;
            end if;
    end process;
end df;
```

*Figure 10 Code for D Flip-Flop*

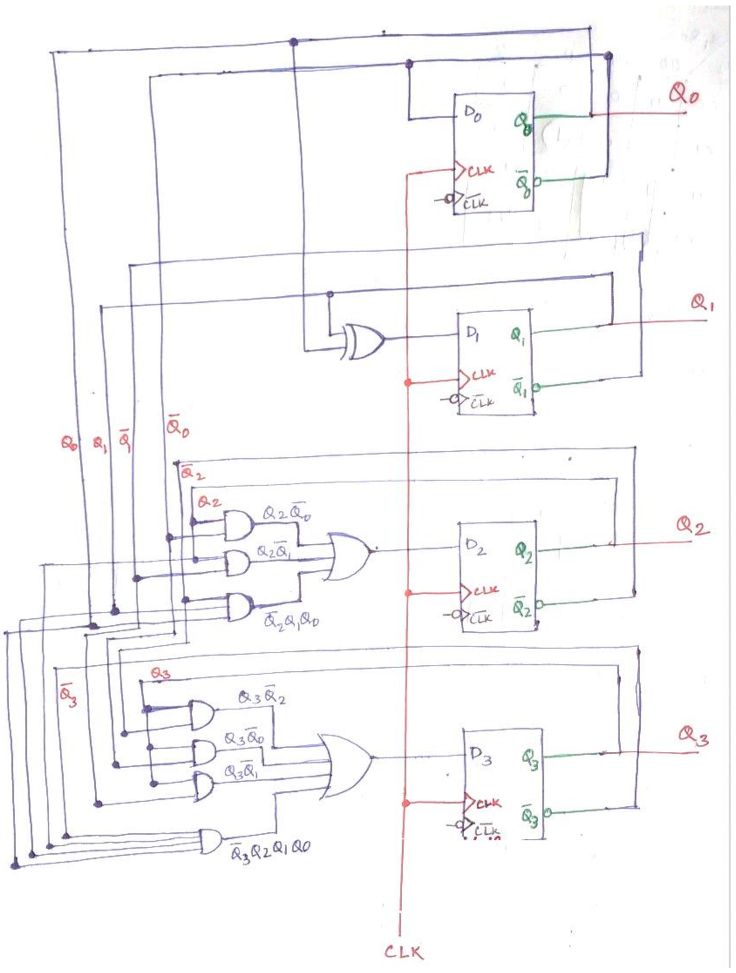
## In-Lab Tasks

## Lab Task 1: Binary counter using D Flip-Flop

For first lab task, you are to design a free-running binary counter that circulates through a binary sequence repeatedly. For example, a 4-bit binary counter counts from "0000", "0001", “0010”… “1111” and wraps around. When the code starts you will display “0000” on LEDs. After every clock pulse, the LEDs display the next number according to the truth table given below

| Present State (Q3 Q2 Q1 Q0) | Next State (Q3+ Q2+ Q1+ Q0+) | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|
| 0000 | 0001 | 0 | 0 | 0 | 1 |
| 0001 | 0010 | 0 | 0 | 1 | 0 |
| 0010 | 0011 | 0 | 0 | 1 | 1 |
| 0011 | 0100 | 0 | 1 | 0 | 0 |
| 0100 | 0101 | 0 | 1 | 0 | 1 |
| 0101 | 0110 | 0 | 1 | 1 | 0 |
| 0110 | 0111 | 0 | 1 | 1 | 1 |
| 0111 | 1000 | 1 | 0 | 0 | 0 |
| 1000 | 1001 | 1 | 0 | 0 | 1 |
| 1001 | 1010 | 1 | 0 | 1 | 0 |
| 1010 | 1011 | 1 | 0 | 1 | 1 |
| 1011 | 1100 | 1 | 1 | 0 | 0 |
| 1100 | 1101 | 1 | 1 | 0 | 1 |
| 1101 | 1110 | 1 | 1 | 1 | 0 |
| 1110 | 1111 | 1 | 1 | 1 | 1 |
| 1111 | 0000 | 0 | 0 | 0 | 0 |

The schematic of the binary counter is given below



Implement this schematic using packages, show the waveforms and test it on FPGA.
To test the functionality of sequential circuits we will assign a push button for clock signal, after testing is done, we will assign the original or slow version of clock available on FPGA.
***For Pin Assignment, consult the Altera handout.***
The state diagram of 4-bit binary counter is

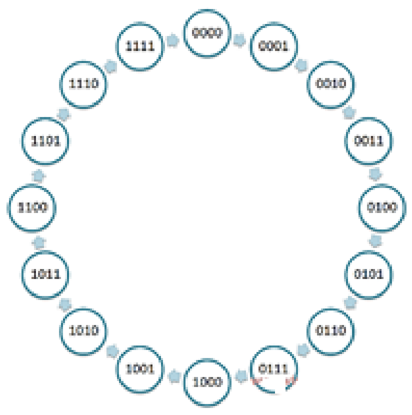


*Figure 11 State diagram of 4-bit counter*

## Lab Task 2: Up/Down Binary counter

A 3-bit binary up/down counter is more versatile. It can count, up or down depending upon selection. If selection is ZERO then the output should go from 000,001 to 111 on rising edge of the clock and if selection is ONE, then the output should go from 111,110 to 000.
Your second lab task is to modify the design from task 1 and add the functionality of up/down counting. Remember that you might need to do the same working for the down counter as it is done for you previously.

1) Complete the state transition table

| **Input** | **Current State** | | | **Next State** | | | **Flip Flop inputs** | | |
|---|---|---|---|---|---|---|---|---|---|
| updown | $Q_2$ | $Q_1$ | $Q_0$ | $Q_2$ | $Q_1$ | $Q_0$ | $D_2$ | $D_1$ | $D_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | | | | | | |
| 0 | 1 | 0 | 0 | | | | | | |
| 0 | 1 | 0 | 1 | | | | | | |
| 0 | 1 | 1 | 0 | | | | | | |
| 0 | 1 | 1 | 1 | | | | | | |
| 1 | 0 | 0 | 0 | | | | | | |
| 1 | 0 | 0 | 1 | | | | | | |
| 1 | 0 | 1 | 0 | | | | | | |
| 1 | 0 | 1 | 1 | | | | | | |
| 1 | 1 | 0 | 0 | | | | | | |
| 1 | 1 | 0 | 1 | | | | | | |
| 1 | 1 | 1 | 0 | | | | | | |
| 1 | 1 | 1 | 1 | | | | | | |

2) For each $D_2$, $D_1$, $D_0$ find out the minimize state equation.
3) Finally, implement using dataflow modeling in VHDL

**Note:** A simpler and easier approach for handling such problems is to use the idea of state machines. State machine design is fairly an easy approach once it comes to VHDL design. We will revisit these problems in Lab 5.

## Rubric for Lab Assessment

| **The student performance for the assigned task during the lab session was:** | | | |
|---|---|---|---|
| Excellent | The student completed assigned tasks without any help from the instructor and showed the results appropriately. | 4 | |
| Good | The student completed assigned tasks with minimal help from the instructor and showed the results appropriately. | 3 | |
| Average | The student could not complete all assigned tasks and showed partial results. | 2 | |
| Worst | The student did not complete assigned tasks. | 1 | |

**Instructor Signature: _______________________** **Date:________________**