Week-2

After the week-1 content, I hope you are able to figure out how to do port mappings and what an entity is, in VHDL(If not I insist you to try learning what exactly each line does, after all it was just some 20 lines of code). Now, let's see what other type of Digital circuits exist and how to model them.

1 Combinational and Sequential Circuits

Digital circuits can be broadly divided into two categories: *combinational* and *sequential*. Understanding the difference is fundamental.

1.1 Combinational Circuits

A combinational circuit produces outputs that depend *only* on the current inputs, without any memory of past inputs. In other words:

$$\text{Output} = f(\text{Inputs}),$$

where f is a Boolean function (e.g., sum-of-products, logical expressions). There are no storage elements, feedback loops, or clocked registers.

Examples of combinational gates:

- **AND gate:** $Y = A \cdot B$.
- **OR gate:** $Y = A + B$.
- **NOT gate (inverter):** $Y = \overline{A}$.
- **XOR gate:** $Y = A \oplus B$.
- **Multiplexer (2:1 MUX):** $Y = A$ if $\text{Sel} = 0$, else $Y = B$.

Since combinational gates have no memory, they always compute the same output for the same inputs. Any circuit you build solely by interconnecting such gates (e.g., adders, multiplexers, decoders, encoders) is a combinational circuit.

1.2 Sequential Circuits

Sequential circuits, in contrast, have *memory*: their outputs depend on both current inputs *and* the circuit's past history. This is achieved by including storage elements (flip-flops, latches, registers) that hold state information. A sequential circuit typically operates under a clock signal:

$$\begin{cases} \text{Next State} = g(\text{Current State}, \text{Inputs}) \\ \text{Output} = h(\text{Current State}, \text{Inputs}) \end{cases}$$

where g and h are Boolean functions.

Example of a storage element: D Flip-Flop (DFF). A D flip-flop captures the value of the D input at a specific clock edge (rising or falling), and then drives that value onto its output Q until the next relevant clock edge.

1.3 DFF

On rising edge of CLK: $Q \leftarrow D$.

Otherwise: Q holds its previous value.

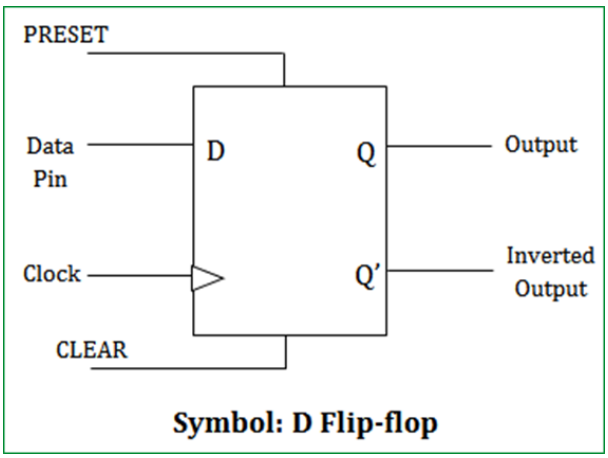

Figure 1: Positive clock edge triggered DFF

How a DFF works (step-by-step):

1. At time t_0, before a rising clock edge, Q holds the old state Q_{old}.
2. When the clock *rises* at t_1, the DFF samples the input D. The sampled value D_{sample} is transferred to Q.
3. After t_1 (until the next rising edge), Q remains stable at D_{sample}, regardless of changes at D.
4. On the next rising edge at t_2, the process repeats: Q takes the current D at that moment.

Usage in memory blocks: Because D flip-flops can store a single bit reliably from one clock cycle to the next, they are the fundamental building blocks of registers and memory arrays. For instance, an 8-bit register memory can be built by placing eight DFFs in parallel, each storing one bit. In larger memory blocks (block RAM, register files), arrays of DFFs or specialized memory cells are used, but the core idea remains: each DFF holds one bit of state.

1.4 Combining Both: Combinational vs. Sequential

- A **pure combinational circuit** has no clock or storage: examples include adders, comparators, and simple logic gates.
- A **pure sequential circuit** uses only flip-flops or latches (e.g., an array of registers), but typically you include some combinational logic with the flip-flops to create counters, finite state machines (FSMs), shift registers, etc.
- In practice, most digital designs intermix combinational logic (to compute next-state or outputs) with sequential elements (to hold state).

2 Modeling Styles in Hardware Description Languages

In hardware description languages (HDLs) such as VHDL and Verilog, there are three main modeling styles used to describe digital circuits: *structural*, *behavioral*, and *dataflow*. Each style serves a different purpose and helps designers express a design at various levels of abstraction.

2.1 Structural Modeling

Structural modeling describes a circuit by explicitly connecting lower-level components (building blocks) together, much like drawing a schematic. In this style:

- You instantiate basic components (e.g., gates, flip-flops, modules) and wire them up.
- You think in terms of "these building blocks are connected in this way."
- It corresponds closely to how the circuit would be physically laid out on silicon or an FPGA.

Key idea: *"I have a set of known components. I connect their inputs and outputs to build a bigger circuit."*

Example (conceptual). Suppose you want to build a 2-input AND gate using two smaller 2-input NAND gates and one inverter (NOT gate). In a structural description, you would write:

- Instantiate NAND1, NAND2, and INV (inverter).
- Connect the inputs A and B to the inputs of both NAND1 and NAND2 (in different configurations).
- Feed the output of NAND2 into the inverter.
- The final output is the inverted output of NAND2.

This explicitly shows how the gates are wired, without relying on algebraic expressions.

2.2 Behavioral Modeling

Behavioral modeling describes *what* the circuit should do (its behavior), rather than *how* it is implemented. It uses high-level constructs, like `if-then-else` statements, `case` statements, and sequential processes. In this style:

- You specify the desired functionality in a programming-like way.
- You do not need to worry about which gates implement the logic.
- The synthesizer/compiler infers the necessary gates or registers to realize the behavior.

Key idea: *"Describe the algorithm or function; let the tool figure out the hardware mapping."*

Example (conceptual). To implement a 4-bit adder with carry:

```
-- Pseudocode in HDL (behavioral)
process(A, B, Cin)
begin
    Sum   <= A + B + Cin;  -- Summation expression
    Cout  <= (A + B + Cin) > 15;  -- Carry-out bit
end process;
```

Here, we write a simple mathematical expression. The synthesizer translates it into adder trees and carry logic.

2.3 Dataflow Modeling

Dataflow modeling sits between structural and behavioral. It describes how data "flows" through a network of operators using *concurrent* signal assignments and Boolean expressions. In this style:

- You define expressions that relate inputs to outputs.
- You use constructs like `assign` (Verilog) or `<=` with `with/select` or `when/else` (VHDL).
- The focus is on the transfer of data rather than on explicit gate instantiation.

Key idea: *"Define equations or logical expressions that show how outputs depend on inputs."*

Example (conceptual). Implementing a 2-to-1 multiplexer using dataflow:

$$\mathrm{Y} = (\overline{\mathrm{Sel}} \wedge A) \vee (\mathrm{Sel} \wedge B).$$

In Verilog, this might look like:

```
assign Y = (Sel ? B : A);
```

In VHDL, a similar concurrent assignment is:

```
Y <= A when (Sel = '0') else B;
```

Resources

- **Logic Gates**
  https://youtu.be/0lwhoQ5aQe8?si=ZjTHH4Z1p1VEbuN1

- **Adder Circuit (Add Two Numbers)**
  https://youtu.be/5XbRIVWFRIw?si=$Z_U XxYDGXquyC_d 1$

- **Multi-Bit Adder**
  https://youtu.be/b70ZQwci5sY?si=iChexnZBdju96OfQ

- **Subtractor**
  https://youtu.be/lqN8xLTtdaA?si=YEmxhC0YZpQSIZQa

- **Multi-Bit Subtractor**
  https://youtu.be/J7gPUP0aRug?si=JoyQIKSAJXrkXWka

- **Multiplexer**
  https://youtu.be/piZFYtQSy58?si=2wp8mRJeZzFABxbK

- All these are few examples for combinational circuits, try modelling them structurally in VHDL and view the RTL view as shown in week-1 material to confirm the structure of the component)

- **Sequential Circuits (Videos 91–103)**
  https://youtube.com/playlist?list=PLwjK$_i yK4LLBC_s o3odA64E2MLgIRKaflsi = IxYHxfS3ub61SkII$

  (Refer to videos numbered 91 through 103 in this playlist for various sequential circuit examples.)