Verilog Module

1. module module_name: Starts the module definition with a chosen unique name.
2. (port1, port2, ...): This is the **port list**. It defines the interface of the module – the signals that go into or out of it.
   - **Port names** must be unique within the module.
   - Ports can be single bits or multi-bit vectors (e.g., [7:0]).
3. **Port Declarations:** Inside the module body, you declare the direction of each port:
   - input: For signals coming into the module.
   - output: For signals going out of the module.
   - inout: For bi-directional signals (can be input or output, often used for data buses).
4. **Internal Declarations:**
   - **Wires (wire):** Used to connect continuous assignments or outputs of gates/modules. Wires do not store values; they simply pass them.
   - **Registers (reg):** Used to store values in procedural blocks (always, initial). reg implies a storage element (like a flip-flop or latch) if assigned within an edge-triggered always block, or a wire if assigned within an always @(*) block (combinational).
5. **Module Body:** Contains the actual description of the hardware, which can be:
   - **Behavioral:** Using always or initial blocks for higher-level descriptions.
   - **Dataflow:** Using assign statements for combinational logic.
   - **Structural:** Instantiating other modules or primitive gates

Example of a sample module

```
module module_name (

    port1,

    port2,

    ...

    portN

);



    // Port declarations (input, output, inout)

    // Wire and Reg declarations

    // Behavioral (always, initial) or Structural (gate instantiations) code

endmodule
```

Verilog Data Types

Verilog has several data types to represent different kinds of signals and variables found in hardware. They fall into two main categories: **Nets** and **Registers**.

1. Net Data Types

Nets represent physical connections between hardware elements, like wires on a circuit board. They are continuously driven by something (an assign statement, a gate output, or a module output) and cannot store a value themselves. If nothing drives a net, its value is z (high impedance).

- **wire**: The most common net type. Represents a physical wire. It can be driven by a continuous assignment (assign), a gate output, or the output of a module instance.

2. Register Data Types (Procedural Variables)

Registers (or procedural variables) are used to store values and are assigned within initial or always procedural blocks. They hold their value until a new value is assigned.

- **reg**: The most common register type. It can represent:
    - **Hardware Registers/Flip-Flops/Latches:** When assigned within an edge-triggered always block (e.g., always @(posedge clk)).
    - **Combinational Logic Output:** When assigned within a level-sensitive always @(*) block. In this case, even though it's reg, synthesis tools will infer combinational logic, and it behaves like a wire from a hardware perspective, but it *must* be declared as reg because it's assigned procedurally.