2. Components of a Testbench

2.1 Entity Declaration

The testbench entity is declared **without any ports**. It's a self-contained module because it doesn't connect to any higher-level design; it *is* the top-level entity for the simulation.

```
entity project_tb is
    -- Empty because testbench doesn't have any port
end project_tb
```

2.2 Architecture Declaration

2.2.1 UUT Component Declaration

Inside the architecture, we first declare the entity we want to test (the UUT) as a component. The component declaration must match the entity declaration of the UUT.

For example, if we have a UUT with these entity declaration:

```
entity earth_destroyer is
    Port (
        clk, rst    : IN STD_LOGIC;
        input       : IN STD_LOGIC VECTOR(7 downto 0);
        mode        : IN STD_LOGIC_VECTOR(3 downto 0);
        output      : OUT STD_LOGIC_VECTOR(7 downto 0)
    );
end earth_destroyer;
```

The UUT component declaration for that entity will be:

```
component earth_destroyer is
    Port (
        clk, rst    : IN STD_LOGIC;
        input       : IN STD_LOGIC VECTOR(7 downto 0);
        mode        : IN STD_LOGIC_VECTOR(3 downto 0);
        output      : OUT STD_LOGIC_VECTOR(7 downto 0)
```

```
    );
end component;
```

As you can see, component declaration is almost the exact same as entity declaration. Just make sure you change `entity` to `component` and use `end component` instead of `end <entity name>`, and you're good to go.

2.2.2 Signals Declaration

We **must** declare internal signals within the testbench architecture. You should **at least** declare all the signals that corresponds to the entity input/output ports. These signals will be **connected to the ports of the UUT** to drive its inputs and monitor its outputs.

For example, if we have `earth_destroyer` entity as in Part 2.2.1, we should declare these signals:

```
signal clk_tb    : STD_LOGIC := '0';
signal rst_tb    : STD_LOGIC;
signal input_tb  : STD_LOGIC VECTOR(7 downto 0);
signal mode_tb   : STD_LOGIC_VECTOR(3 downto 0);
signal output_tb : STD_LOGIC_VECTOR(7 downto 0);
```

The name of the signal doesn't really matter, as long as its **data type** match the port it corresponds to.

2.3 Port Map

In VHDL, a `port map` is part of the **component instantiation** within an architecture that maps the input and output ports of an entity to local signals. By using `port map`, we can connect a testbench to the entity being tested, allowing inputs to be driven and outputs to be observed through the testbench.

The general syntax of `port map` in a testbench is:

```
UUT : entity_name port map (entity_port_name => local_signal_name);
```

For example, if we want to instantiate the component as in Part 2.2, we can write it like this:

```
UUT : earth_destroyer port map (
    clk    => clk_tb,
    rst    => rst_tb,
    input  => input_tb,
    mode   => mode_tb,
    output => output_tb
);
```

Revision #2
Created 2025-09-15 16:17:28 UTC by RE
Updated 2025-09-15 16:19:30 UTC by RE