Generic Clause

A generic statement is a general mechanism used to pass information to an entity. Generics aid readability, maintenance and configuration by allowing a component to be customized by creating a parameter to be passed on to the architecture.

Generic information is static and cannot change during simulation.

Generics can have default values that can be overwritten. i.e., they can be initalized. (be careful!)

Format:

```
GENERIC (generic_name:type[:= default_value]);
```

If default_value is missing, it must be present when the component is instantiated.

Examples:

```
ENTITY half_adder IS
  GENERIC(
        tpd_result : delay := 4ns;
        tpd_carry  : delay := 3ns);
  PORT( x  IN  :  std_logic;
        y  IN  :  std_logic;
        z  OUT :  std_logic);
END half_adder;

ARCHITECTURE dataflow OF half_adder
  BEGIN
    I result <= x XOR y AFTER tpd_result;
      carry  <= x AND y AFTER tpd_carry;
  END dataflow;
```

Example: A varaible width register

```
ENTITY register_n IS
  GENERIC(width : INTEGER := 8);
  PORT(d     : IN   STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
       q     : OUT  STD_LOGIC_VECTOR(WDITH-1 DOWTNO 0);
       clk   : IN   STD_LOGIC;
       rst_n : IN   STD_LOGIC);
END ENTITY register_n;

ARCHITECTURE silly of register_n IS
  BEGIN
    PROCESS(d, clk, rst_n)
      BEGIN
        IF (rst_n = ‘0’) THEN
          q <= (OTHERS => ‘0’);
        ELSIF (clk’EVENT AND clk = ‘1’) THEN
          q <= d;
        END IF;
    END PROCESS;
END silly;
```

Example: Instantiating the paramatized register

```
pipeline_reg0: register_n
  GENERIC MAP(width => 32)
  PORT MAP(
          clk       => clk,
          rst_n     => reset_n,
          d         => reg_in,
          q         => reg_out
          );
```

Note that is no semicolon after the GENERIC MAP line.