For-Generate Loop

The Concurrent 'for-generate' Loop

We now switch from sequential loops to a **concurrent** one.

The `for-generate` statement is **not** a loop that executes over time inside a process. Instead, it's a command that tells the synthesizer to create multiple copies of hardware structures.

It's good for repetitive structures like registers and chains of components. A key rule is that `for-generate` is used **outside** of a `process`, in the main architectural body.

Main Differentiators

- This loop is **Concurrent**, not **Sequential**. All the hardware instances created by the loop exist simultaneously and operate in parallel. The synthesizer "unrolls" the loop, creating all the copies before the design is even simulated or synthesized.
- This is most often used to create multiple instances of a `component`, but **can also be used for concurrent signal assignments or even entire `process` blocks**.

Syntax

The basic structure of a `for-generate` statement is:

```
generate_label: for <identifier> in <range> generate
    -- Concurrent statements to be replicated go here...
    -- (e.g., component instantiations)
end generate generate_label;
```

- Label, identifier, and range, all follow the same rules as the other loops (`for` and `while`).

Example: 4-Bit Adder

We start with a **1-bit full adder** and use `for-generate` to create and connect four of them in a chain to make a **4-bit adder**.

1. Replication Target First, we need the definition of the 1-bit full adder that we want to copy.

```
component full_adder is
    port (
        A, B, Cin  : in  std_logic;
        S, Cout    : out std_logic
    );
end component;
```

2. 4-Bit Adder Architecture Next, we use `for-generate` to create four instances of the `full_adder` and wire them together.

```
entity four_bit_adder is
    port (
        A, B    : in  std_logic_vector(3 downto 0);
        Cin     : in  std_logic;
        S       : out std_logic_vector(3 downto 0);
        Cout    : out std_logic
    );
end entity;

architecture structural of four_bit_adder is
    -- Internal signal to wire the carry chain between the adders.
    -- It needs 5 bits to include the first Cin and final Cout.
    signal C : std_logic_vector(4 downto 0);
begin

    -- The first carry wire is connected to the adder's carry-in pin
    C(0) <= Cin;

    -- Generate the chain of 4 full adders
    ADDER_CHAIN: for i in 0 to 3 generate
        -- Create one instance of the full_adder in each "iteration"
        FA_INSTANCE: full_adder
            port map (
                A    => A(i),       -- Connect to the i-th bit of input A
                B    => B(i),       -- Connect to the i-th bit of input B
                Cin  => C(i),       -- The carry-in for this bit
                S    => S(i),       -- The sum output for this bit
                Cout => C(i+1)      -- The carry-out for this bit
            );
    end generate ADDER_CHAIN;
```

```
    -- The final carry-out of the chain is the adder's carry-out pin
    Cout <= C(4);

end architecture structural;
```

Physically (RTL wise), this is the exact same as copy-pasting the 4 adders manually. So, this approach is much cleaner and more organized.

Revision #1
Created 2025-10-03 17:25:09 UTC by JD
Updated 2025-10-03 17:35:18 UTC by JD