VHDL Modularity

We will build a 4-bit Ripple Carry Adder using 4 Full Adders in Structural Style Programming. Each Full Adder's carry-out serves as the carry-in for the next Full Adder, creating a ripple effect in addition.

Step 1 - Full Adder Entity

Inside full adder entity, we will declare the input and output ports. The input ports are A, B, and Cin, and the output ports are Sum and Cout.

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

architecture structural of full_adder is
begin
    Sum <= A xor B xor Cin;
    Cout <= (A and B) or (B and Cin) or (A and Cin);
end architecture structural;
```

Step 2 - Ripple Carry Adder Architecture

Next step is to create the architecture for the 4-bit Ripple Carry Adder. We will instantiate 4 Full Adders and connect them in a way that the carry-out of one Full Adder is connected to the carry-in of the next Full Adder.

```
entity ripple_carry_adder is
    port(
        A, B : in std_logic_vector(3 downto 0);
        Sum : out std_logic_vector(3 downto 0);
        Cout : out std_logic
    );
```

```
end entity ripple_carry_adder;

architecture structural of ripple_carry_adder is
    component full_adder
        port(
            A, B, Cin : in std_logic;
            Sum, Cout : out std_logic
        );
    end component full_adder;

    signal C : std_logic_vector(3 downto 0);
begin
    FA0 : full_adder port map(A(0), B(0), '0', Sum(0), C(0));
    FA1 : full_adder port map(A(1), B(1), C(0), Sum(1), C(1));
    FA2 : full_adder port map(A(2), B(2), C(1), Sum(2), C(2));
    FA3 : full_adder port map(A(3), B(3), C(2), Sum(3), Cout);
end architecture structural;
```

Based on the above code, we can see that the carry-out of each Full Adder is connected to the carry-in of the next Full Adder. This creates a ripple effect in addition. Therefore the port map explanation is as follows:

FA0 - Full Adder 0

- A(0) and B(0) are the input bits for the first Full Adder.
- '0' is the carry-in for the first Full Adder.
- Sum(0) is the output sum of the first Full Adder.
- C(0) is the carry-out of the first Full Adder.

FA1 - Full Adder 1

- A(1) and B(1) are the input bits for the second Full Adder.
- C(0) is the carry-in for the second Full Adder.
- Sum(1) is the output sum of the second Full Adder.
- C(1) is the carry-out of the second Full Adder.

FA2 - Full Adder 2

- A(2) and B(2) are the input bits for the third Full Adder.
- C(1) is the carry-in for the third Full Adder.
- Sum(2) is the output sum of the third Full Adder.
- C(2) is the carry-out of the third Full Adder.

FA3 - Full Adder 3

- A(3) and B(3) are the input bits for the fourth Full Adder.
- C(2) is the carry-in for the fourth Full Adder.
- Sum(3) is the output sum of the fourth Full Adder.
- Cout is the carry-out of the fourth Full Adder.

With this architecture, we have successfully implemented a 4-bit Ripple Carry Adder using 4 Full Adders in Structural Style Programming.

Step 3 - Testbench

To test the functionality of the 4-bit Ripple Carry Adder, we will create a testbench that provides input values to the adder and checks the output values.

```
entity tb_ripple_carry_adder is
end entity tb_ripple_carry_adder;

architecture testbench of tb_ripple_carry_adder is
    signal A, B : std_logic_vector(3 downto 0);
    signal Sum : std_logic_vector(3 downto 0);
    signal Cout : std_logic;
    signal clk : std_logic := '0';

    component ripple_carry_adder
        port(
            A, B : in std_logic_vector(3 downto 0);
            Sum : out std_logic_vector(3 downto 0);
            Cout : out std_logic
        );
    end component ripple_carry_adder;

begin
    dut : ripple_carry_adder port map(A, B, Sum, Cout);

    process
    begin
        A <= "0000"; B <= "0000"; wait for 10 ns;
        A <= "0001"; B <= "0001"; wait for 10 ns;
        A <= "0010"; B <= "0010"; wait for 10 ns;
        A <= "0011"; B <= "0011"; wait for 10 ns;
```

```
        A <= "0100"; B <= "0100"; wait for 10 ns;
        A <= "0101"; B <= "0101"; wait for 10 ns;
        A <= "0110"; B <= "0110"; wait for 10 ns;
        A <= "0111"; B <= "0111"; wait for 10 ns;
        A <= "1000"; B <= "1000"; wait for 10 ns;
        A <= "1001"; B <= "1001"; wait for 10 ns;
        A <= "1010"; B <= "1010"; wait for 10 ns;
        A <= "1011"; B <= "1011"; wait for 10 ns;
        A <= "1100"; B <= "1100"; wait for 10 ns;
        A <= "1101"; B <= "1101"; wait for 10 ns;
        A <= "1110"; B <= "1110"; wait for 10 ns;
        A <= "1111"; B <= "1111"; wait for 10 ns;
        wait;
    end process;


end architecture testbench;
```

In the testbench, we provide different input values to the 4-bit Ripple Carry Adder and observe the output values. The testbench will simulate the addition process and verify the correctness of the adder. The output values can be checked against the expected results to ensure the adder is functioning correctly

---

Revision #1
Created 23 September 2024 16:03:36 by GI
Updated 23 September 2024 16:03:43 by GI