VHDL (II)

Two-Bit Adder

- Design a circuit that takes two 2-bit inputs as unsigned binary sequences, add them, and generate a 3-bit outputs.
- We can design the circuit using gates like what we did for the full adder, but we can also leverage the numeric_std library to simplify the process.
  - numeric_std library provides numeric types (signed/unsigned) and arithmetic functions.
- We need a vhd file describe the circuit for 2-bit adder, twoBitAdder.vhd

```
-- Ying Li
-- A circuit that takes two 2-bit inputs and generate a 3-bit output

-- import useful packages
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- provides numeric types (unsigned/signed) and arithmetic functions

-- filename must be the same as entity name
-- entity defines the inputs and outputs of the circuit
entity twoBitAdder is
  -- A and B are an array of inputs
  -- F is an array of outputs
  port (A, B: in unsigned (1 downto 0); -- array/vector of unsigned type
       F: out unsigned (2 downto 0));
end twoBitAdder;

-- architecture define the circuit
architecture behavior of twoBitAdder is
begin
  -- extend both 2-bit inputs to 3-bit inputs and then do the addition
  F <= ('0' & A) + ('0' & B);
end behavior;
```
To test this circuit, we also need a test file, `twoBitAdderTest.vhd`.

```vhdl
-- Ying Li
-- A testbench for the 2bitAdder

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- A testbench has no ports
entity twoBitAdderTest is
end entity;

architecture behavior of twoBitAdderTest is
  -- Declaration of the component that will be instantiated
  component twoBitAdder
    port (
      A, B: in unsigned (1 downto 0); -- array/vector of unsigned
      F: out unsigned (2 downto 0)
    );
  end component;

  -- the twoBitAdder signals
  signal I0, I1: unsigned (1 downto 0);
  signal O: unsigned (2 downto 0);
  constant num_iterations: integer := 4;

  begin
    -- component instantiation
    -- connect the inputs and outputs of the entity to the local signals
    twoBitAdder1 : twoBitAdder port map (A=>I0, B=>I1, F=>O);

    I0(1) <= '0', '1' after 8 ns;
    I0(0) <= '0', '1' after 4 ns, '0' after 8 ns, '1' after 12 ns;

    process begin
      for i in 1 to num_iterations loop
        I1(1) <= '0';
        wait for 2 ns;
        I1(1) <= '1';
        wait for 2 ns;
      end loop;
    end process;

    process begin
      for i in 1 to num_iterations loop
        I1(0) <= '0';
        wait for 1 ns;
        I1(0) <= '1';
        wait for 1 ns;
        I1(0) <= '0';
        wait for 1 ns;
        I1(0) <= '1';
        wait for 1 ns;
      end loop;
    end process;

  end behavior;
```
- Then, compile and run the code

```
mbp-190250:2bitAdder yingli$ ghdl -a twoBitAdder.vhd
mbp-190250:2bitAdder yingli$ ghdl -a twoBitAdderTest.vhd
mbp-190250:2bitAdder yingli$ ghdl -e twoBitAdderTest
mbp-190250:2bitAdder yingli$ ghdl -r twoBitAdderTest --vcd=twoBitAdderTest.vcd
mbp-190250:2bitAdder yingli$
```

- We can get a gtkwave file like this to demonstrate the correctness of the circuit.
Test the twoBitAdder using Counter

- Now we know how to test a circuit using process and signal assignment to simulate the input signals.
- We can also use a counter to generate the signals for inputs of the tested circuit.
- Here is a revised counter.vhd file.

```vhdl
-- Quartus II VHDL Template
-- Binary Counter
-- modified by Ying Li

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
  port(
    clk: in std_logic;
    reset: in std_logic;
    enable: in std_logic;
    q: out std_logic_vector (1 downto 0) -- array/vector of std_logic
  );
end entity;

architecture behavior of counter is
  signal cnt: unsigned (1 downto 0);

begin
  process (clk)
  begin
    if reset = '1' then
      -- Reset the counter to 0
      cnt <= "00";
    elsif rising_edge(clk) then
      if enable = '1' then
        -- Increment the counter if counting is enabled
        cnt <= cnt + 1;
      end if;
    end if;
  end process;

  q <= std_logic_vector(cnt);
end behavior;
```
- The twoBitAdder.vhd is the same as the previous sample code.

```vhdl
-- Ying Li
-- A circuit that takes two 2-bit inputs and generate a 3-bit output

-- import useful packages
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- provides numeric types (unsigned/signed) and arithmetic functions

-- filename must be the same as entity name
-- entity defines the inputs and outputs of the circuit
entity twoBitAdder is
  -- A and B are an array of inputs
  -- F is an array of outputs
  port ( 
      A, B: in unsigned (1 downto 0); -- array/vector of unsigned type
      F: out unsigned (2 downto 0) 
  );
end twoBitAdder;

-- architecture define the circuit
architecture behavior of twoBitAdder is
begin
  -- extend both 2-bit inputs to 3-bit inputs and then do the addition
  F <= ('0' & A) + ('0' & B);
end behavior;
```

- In the twoBitAdderTest.vhd, we want to use counter. So, there are two components in this test file: one for twoBitAdder, and the other for counter.vhd.

- We want each input of the twoBitAdder get signals from a counter, so we instantiate two counter instances (counter1 and counter2).

- We then map the ports of the two counters to the local signals, and assign the two counter outputs to the two inputs of the twoBitAdder.

  • Please note that if the local signals have the same names as the component ports, we can simplify the port map as counter1: counter port map (clk, reset, enable, unsigned(q)=>I0); No => needed here.

  • We use a type casting here. q is a std_logic_vector, and I0 is unsigned array, although they have the same size. So, we type cast q to an unsigned array here to make the assignment happen.

- Finally, we set the reset, enable, and clk signals appropriately, so that the counters can work.
A testbench for the 2bitAdder

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- A testbench has no ports
entity twoBitAdderTest is
end entity;

architecture behavior of twoBitAdderTest is
  -- Declaration of the component that will be instantiated
  component twoBitAdder
    port (  
      A, B: in unsigned (1 downto 0); -- array/vector of unsigned  
      F: out unsigned (2 downto 0)
    );
  end component;

  -- counter is used to generate signals for the two inputs of the twoBitAdder
  component counter
    port (  
      clk: in std_logic;  
      reset: in std_logic;  
      enable: in std_logic;  
      q: out std_logic_vector (1 downto 0)
    );
  end component;

  -- the twoBitAdder signals
  signal I0, I1: unsigned (1 downto 0);  
  signal O: unsigned (2 downto 0);

  -- the counter signals
  signal clk: std_logic;  
  signal enable: std_logic;  
  signal reset: std_logic;  
  signal q: std_logic_vector (1 downto 0);

  constant num_cycles : integer := 6;

begin
  -- component instantiation
  -- connect the inputs and outputs of the entity to the local signals
  twoBitAdder1 : twoBitAdder port map (A=>I0, B=>I1, F=>O);
  counter1: counter port map (clk, reset, enable, unsigned(q)=>I0);
  counter2: counter port map (clk, reset, enable, unsigned(q)=>I1);

  -- start off with a short reset
  reset <= '1', '0' after 1 ns;
  enable <= '1';

  -- create a clock
  process begin
    for i in 1 to num_cycles loop
      clk <= '0';
      wait for 1 ns;
      clk <= '1';
      wait for 1 ns;
      end loop;
  end process;
end behavior;
```
- After compiling the code, we can get a gtkwave like this.