EE214: Experiment 2 (Wednesday Batch)

Wadhwani Electronics Lab, IIT Bombay

3rd March 2021

Instructions:

1. So far we have dealt with structural and data-flow kind of design styles. However it may become tedious to optimize boolean expressions manually for complex designs. Behavioral description comes handy when exact design schematic is not known or too large to implement structurally. However one should not take this availability for granted and write C/C++ style codes without considering the synthesizability and timing of the design.
2. You should not use any other library & package (except ieee.std_logic_1164.all)
3. A "Latch" or Register or combinational loop should not be inferred from your VHDL code by Quartus since we are making a combinational circuit.
4. We encourage students to use generics appropriately and avoid hard-coding as much as possible
5. Please upload well –commented– code

Problem Statement

1. Design and describe 4-bit multiplier in VHDL using Behavioral modeling. You can use the Skeleton code attached or you can code from scratch. (5)

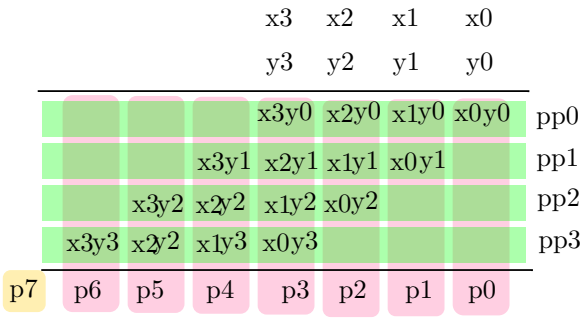

Figure 1: Multiplier

2. Generate tracefile with the following format. (2)
(A3 A2 A1 A0 B3 B2 B1 B0) (M7 M6 M5 M4 M3 M2 M1 M0) (1 1 1 1 1 1 1 1)
3. Verify working of your design by performing RTL and Gate-level simulation. Show the results to your TA.(3)
4. Verify working of your design using scanchain. Show the results to your TA. (5)

Hints:

- 1D X 1D array can be declared as
type pp_type is array (range <>) of std_logic_vector(range <>);
- 1D X 1D array can be initialized as
variable pp : pp_type := (others => (others =>' 0'));
- Values can be assigned to 1D X 1D array variable as
pp(2)(3) := '1';

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

entity mul is
    generic(
        N : integer:=4; -- operand width
        NN : integer:=8 -- result width
        );
    port (
        A: in std_logic_vector(N-1 downto 0);
        B: in std_logic_vector(N-1 downto 0);
        M: out std_logic_vector((NN)-1 downto 0)
    ) ;
end mul;

architecture beh of mul is
    -- unbounded 1D X 1D array declaration
    type pp_type is array (natural range<>) of std_logic_vector(NN-1 downto 0);

    -- adder function adds two 8-bit number. [Usage: var := adder(X,Y) where var is a variable
    --                                                  and X,Y are two 8-bit inputs to be added]
    function adder(A: in std_logic_vector; B: in std_logic_vector)
        return std_logic_vector is
            -- variable declaration
            variable sum : std_logic_vector(NN downto 0):= (others=>'0');
            variable carry : std_logic_vector(NN downto 0):= (others=>'0');
        begin
            -- describing behaviour of adder
            for i in 0 to NN-1 loop
                sum(i) := A(i) xor B(i) xor carry(i);
                carry(i+1) := (A(i) and B(i)) or (carry(i) and (A(i) xor B(i)));
            end loop;
            sum(NN):=carry(NN);
            return sum;
    end adder;
begin


multiplier : process(A, B)
-- declaration of 1D X 1D array to store partial products
-- ////////////// TODO

-- declaration of summation of partial product will give multiplication result which is stored in
-- variable, result.
variable result : std_logic_vector(NN-1 downto 0):= (others=>'0');

-- temporary array is required to avoid combinaitonal loop while adding partial product

begin
    -- Calculation of partial product
    -- ////////////// TODO

    -- summation of partial product
    --////////////// TODO

    M <= result; -- assignment of final result

end process ; -- multiplier
end beh ; -- beh
```