UTS VLSI forum

PLD Basics and Xilinx ISE tool flow

By

K. Vasu

Senior Application engineer

vasu@unistring.com, vasu.vlsi@gmail.com

Unistring Tech Solutions Pvt. Ltd

D10, 5th Floor, Eureka court, Beside Image hospitals, Ameerpet, Hyderabad, ph:040-23732798, 09440318188

www.unistring.com

String Technologies

#309, Vederi complex, opp Dilshuk Nagar bus Depot, Dilsukh nagar, Hyderabad, ph: 040 – 24151900

www.stringtechnologies.net

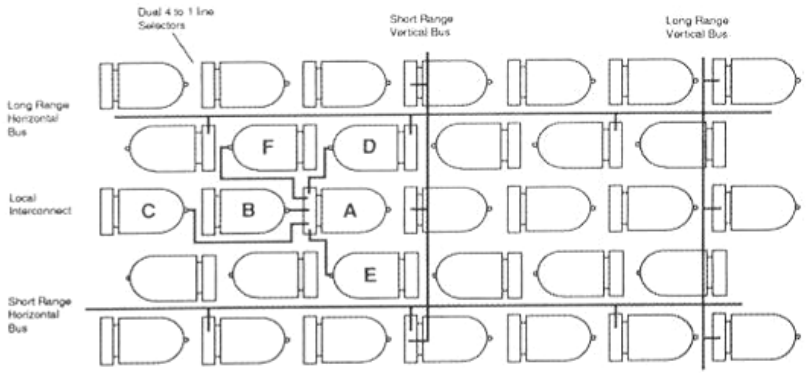
What is programmable logic device (PLD) ?

- A device which comes with some generic (or) configurable hardware whose functionality can be changed as per the application
  - Different than microprocessors and DSPs
  - Useful when hardware solution is required but ASIC is not feasible
  - PALs, CPLDs and FPGAs are popular

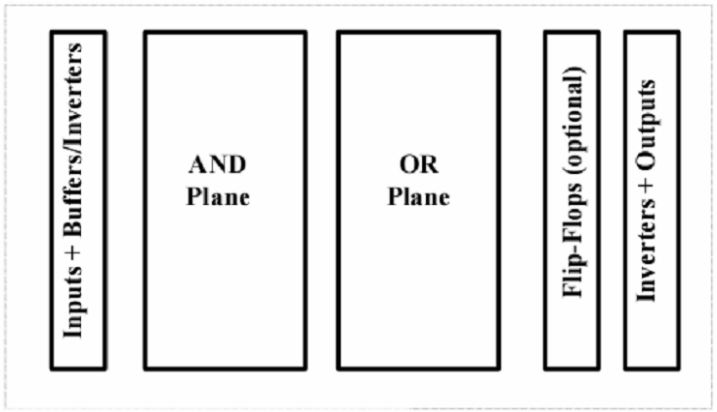
Programmable Logic Devices

- Sea of gates architecture (obsolete today)
- SPLDs (simple PLDs).
- CPLDs (complex PLDs).
- FPGAs (field programmable gate arrays).

Sea of gates architecture

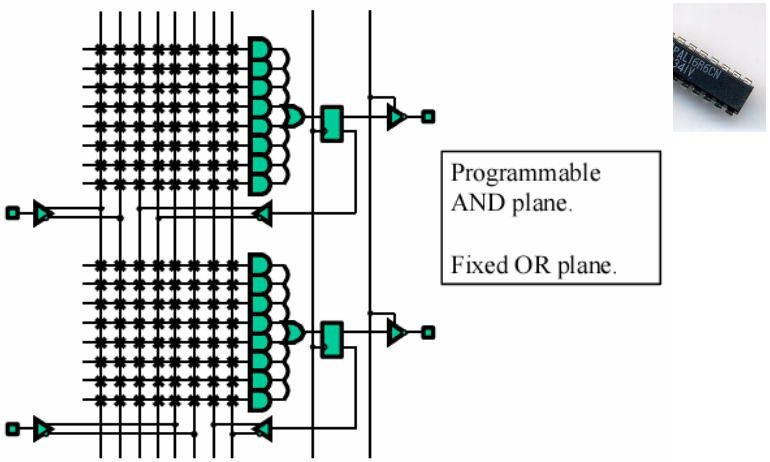


SPLD basic Layout

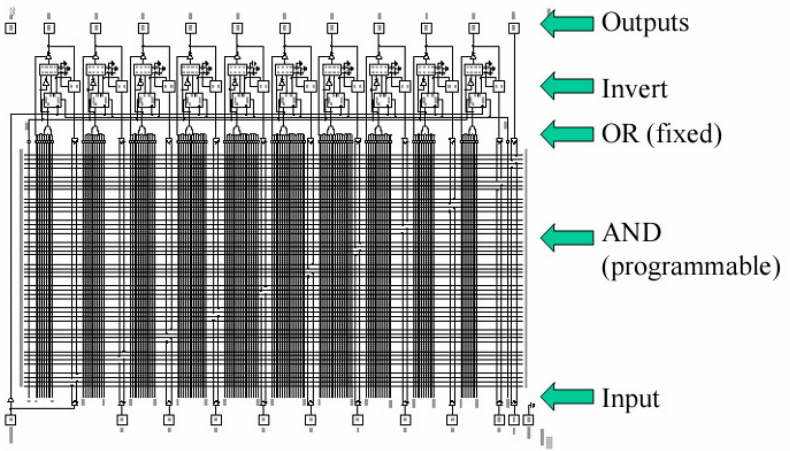


- PAL, PLA and PROM

PAL Architecture

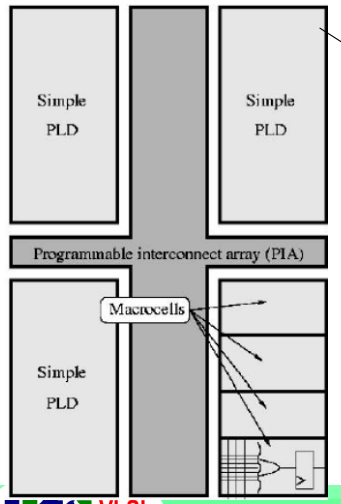


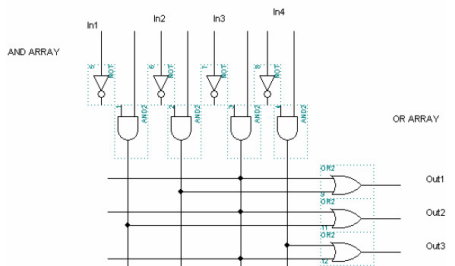
22V10 PAL Logic Diagram



Complex programmable logic device (CPLD)

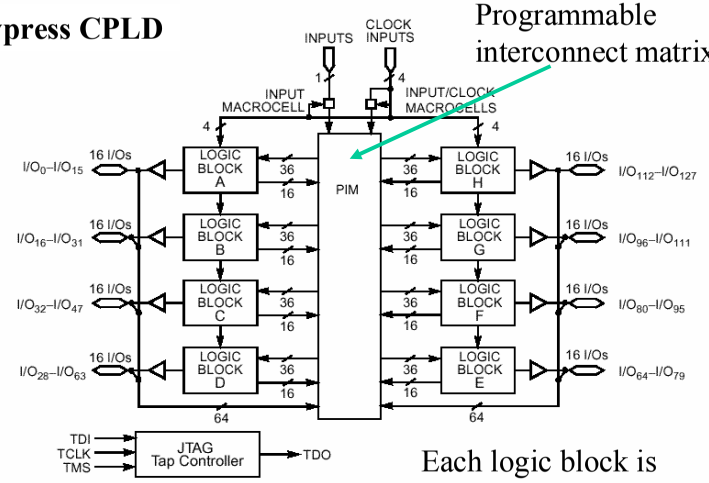
Typically a CPLD consists of 50 SPLDs (An SPLD is a programmable logic array with AND and OR planes)





Programmable Logic Array

Cypress CPLD



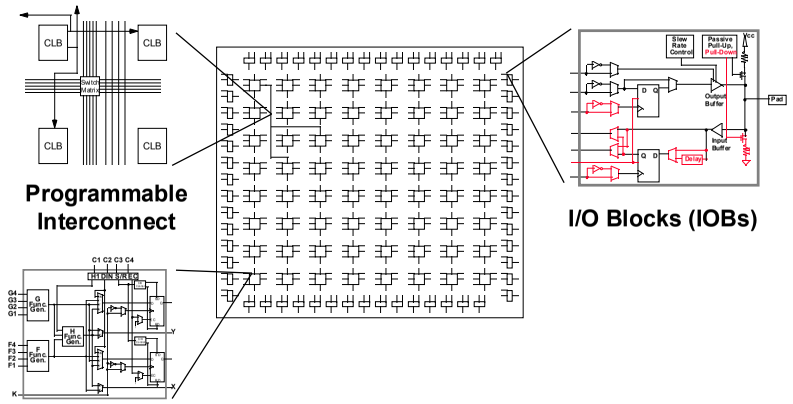
Programmable interconnect matrix.

Each logic block is similar to a 22V10.

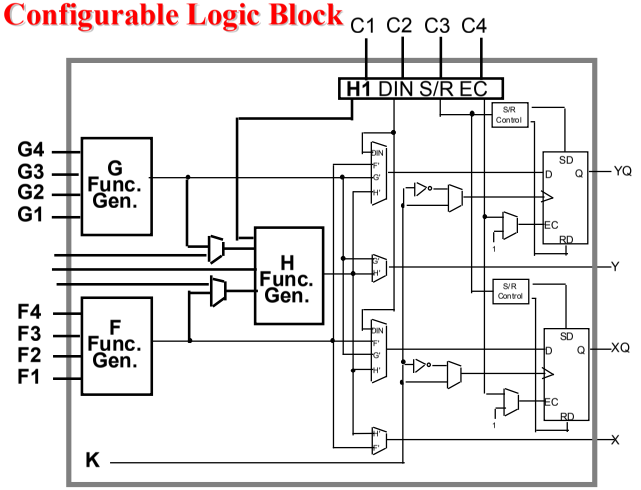
Figure 1. Ultra37128 Block Diagram

FPGA

Programmable devices containing repeated sections of small logic blocks



Xilinx XC4000 Configurable Logic Block



Conceptual difference between PLDs

PLDs - Classification by Granularity

Granularity - What is the logic block size which is being configured

In general three different granularity classes can be found

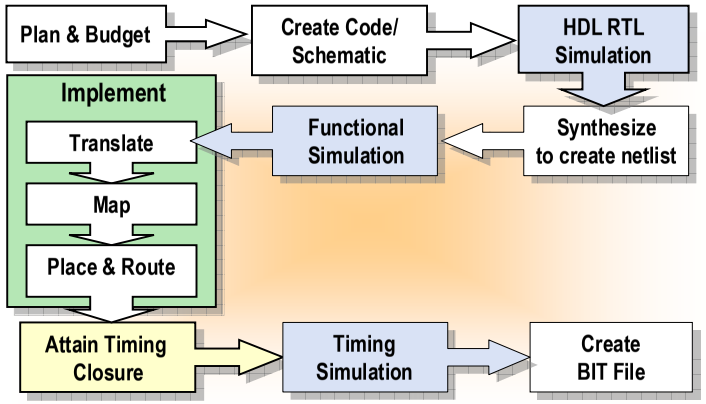
- Small granularity (sea of gates architecture)
- Medium granularity (FPGA)
- Large granularity (SPLD and CPLD)

If the granularity is less then the effort required to synthesize and complete the wiring between the blocks is high.

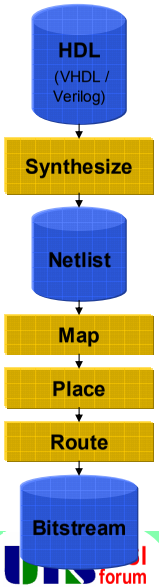
If the granularity is high then while implementing a circuit the utilization of available programmable logic is less

Xilinx tool flow for simulation and synthesis

Xilinx Design Flow

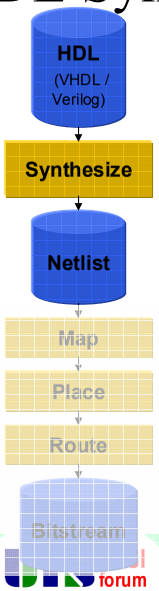


FPGA toolflow

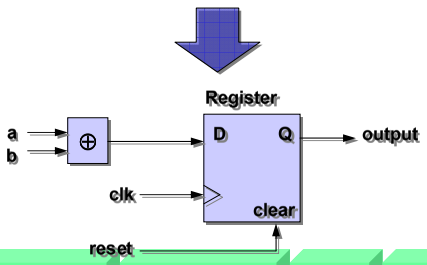


- Hardware design is traditionally done by modeling the system in a hardware description language
- An FPGA “compiler” (synthesis tool) generates a netlist,
- which is then mapped to the FPGA technology,
- the inferred components are placed on the chip,
- and the connecting signals are routed through the interconnection network.

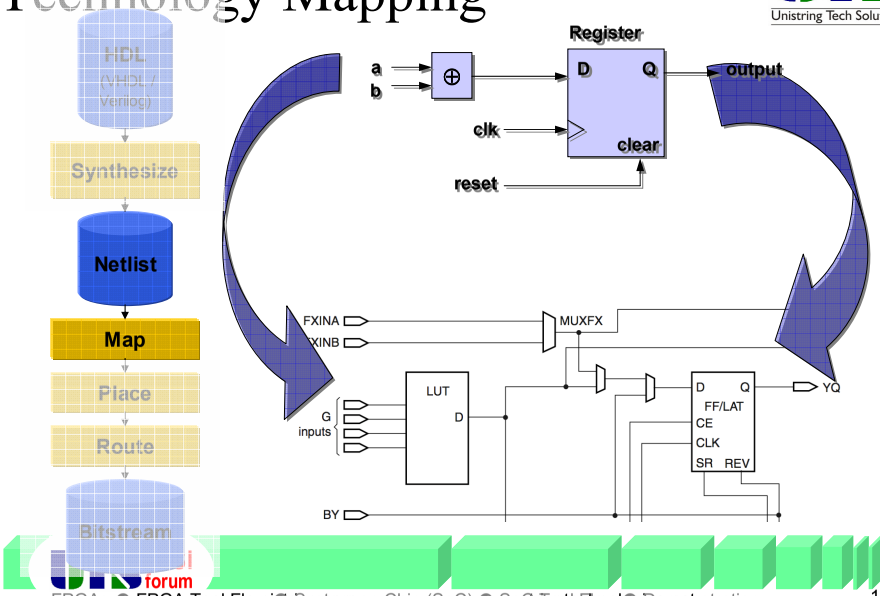
HDL Synthesis



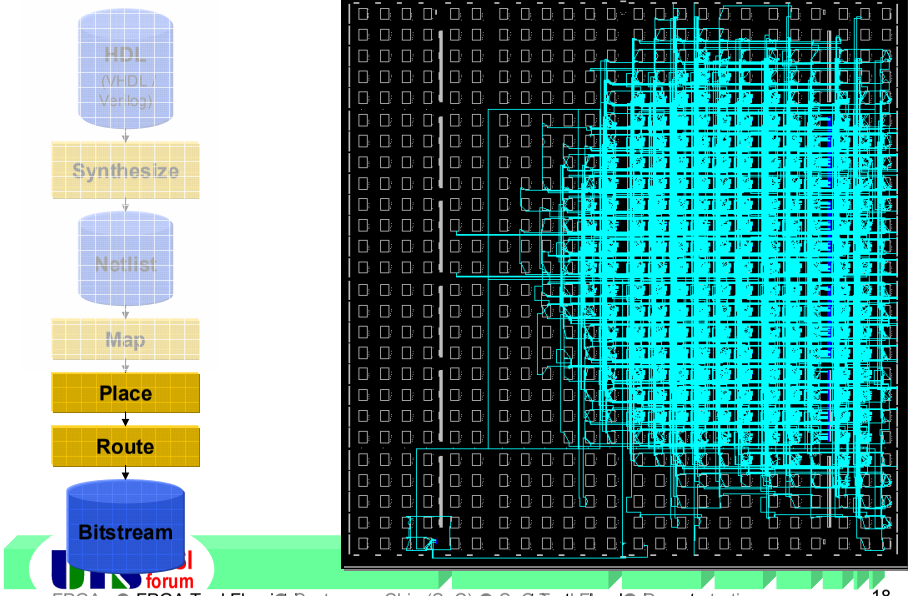

```
process(clk, reset)
begin
        if reset = ‚1‘ then
                output <= ‚0‘;
        elsif rising_edge(clk) then
                output <= a XOR b;
        end if;
end process;
```



Technology Mapping

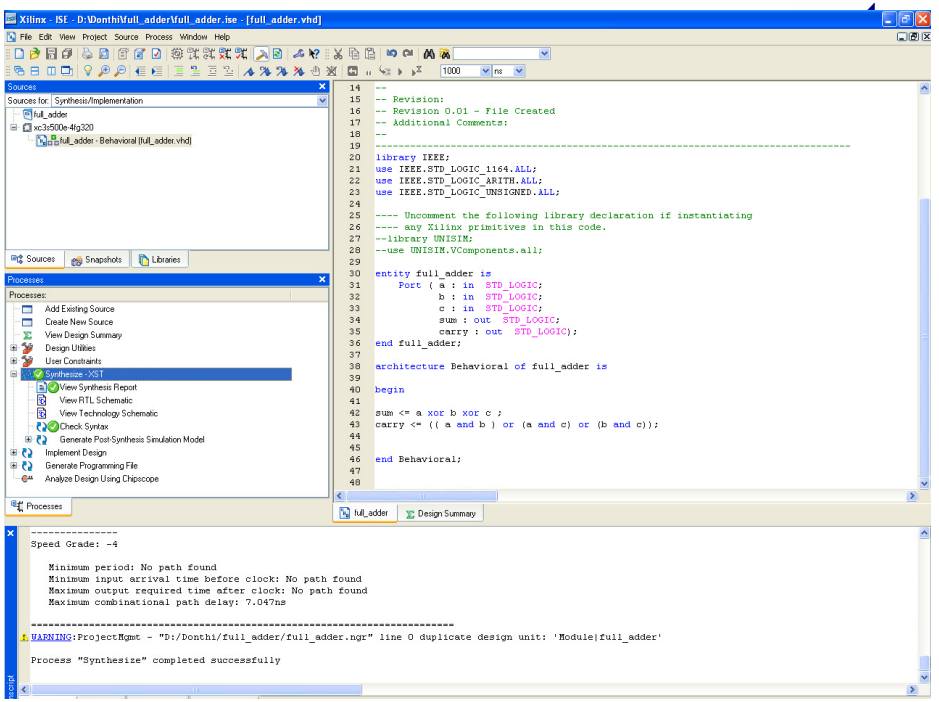


Place & Route



Download

- Once a design is implemented, you must create a file that the FPGA can understand
  - This file is called a bitstream: a BIT file (.bit extension)
- The BIT file can be downloaded directly into the FPGA, or the BIT file can be converted into a PROM file, which stores the programming information



ECAD lab syllabus
&
UTS FPGA board solutions

ECAD lab syllabus

- Exp1: Various Digital Gates
- Exp2: DFF (7474)
- Exp3: Decade counter (7490)
- Exp4: 4 bit counter (7493)
- Exp5: shift register (7495)
- Exp6: Universal shift register (74194 / 74195)
- Exp7: 3 to 8 decoder (74138)
- Exp8: 4 bit comparator (7485)
- Exp9: 8 by 1 Multiplexer (74150)
- Exp10: 16 by 1 Multiplexer (74151)
- Exp11: RAM 16 by 1 - read and write operations (74189)
- Exp12: Stack and queue implementation using RAM

UTS
FPGA boards, softwares & IP cores

Experiment – 1
Various digital gates

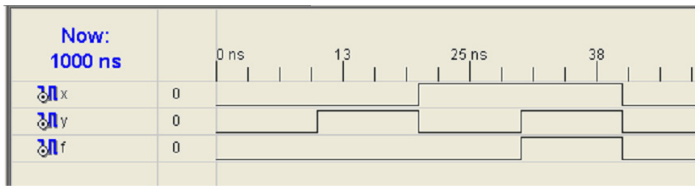
- VHDL coding, FPGA synthesis and on board verification of various digital gates.
- A simple 2 input AND gate will be explained here…
- Similarly the other gates can be tried out.

```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity AND_ent is
        port(  x: in std_logic;
          y: in std_logic;
          F: out std_logic);
end AND_ent;
-----------------------------------------------
architecture behav2 of AND_ent is
begin
        F <= x and y;
 end behav2;
 -----------------------------------------------
```

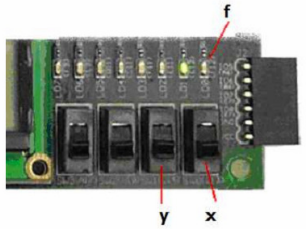


```
-----------------------------------------------
architecture behav1 of AND_ent is
begin
    process(x, y)
    begin
       -- compare to truth table
       if ((x='1') and (y='1')) then
            F <= '1';
       else
            F <= '0';
 end if;
   end process;

end behav1;
```

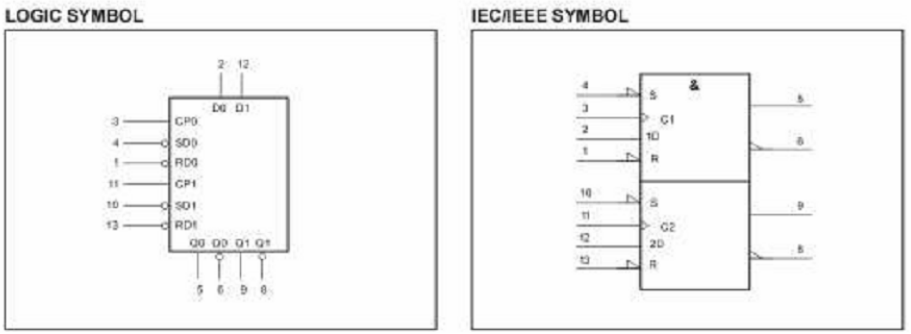



The output on kit can be observed with two pushbuttons and one LED as shown in the below figure.



Experiment - 2

- VHDL coding, simulation and FPGA synthesis of Dual D type Flip Flop (7474).



VHDL code for exp-2

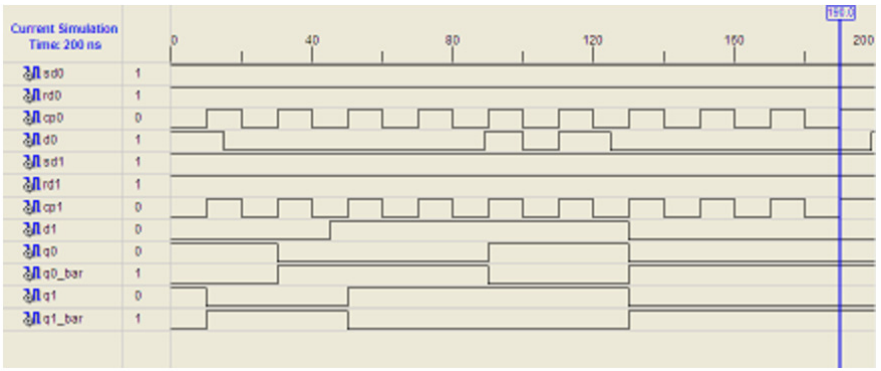
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity exp2_dual_dff is
   Port ( SD0 : in  STD_LOGIC;
          RD0 : in  STD_LOGIC;
          CP0 : in  STD_LOGIC;
          D0 : in  STD_LOGIC;
          SD1 : in  STD_LOGIC;
          RD1 : in  STD_LOGIC;
          CP1 : in  STD_LOGIC;
          D1 : in  STD_LOGIC;
          Q0 : out  STD_LOGIC;
          Q0_bar : out  STD_LOGIC;
          Q1 : out  STD_LOGIC;
          Q1_bar : out  STD_LOGIC);
end exp2_dual_dff;
```



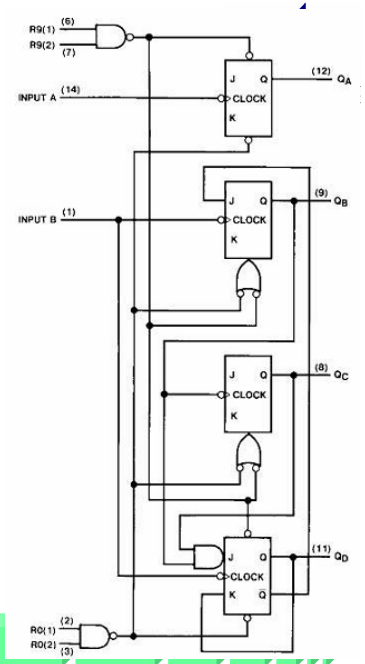
```
architecture Behavioral of exp2_dual_dff is
begin
 FF1: process(SD0,RD0,CP0)
   variable q_temp:std_logic;
 begin
   if SD0='1' then
     q_temp := '1';
   elsif RD0='1' then
     q_temp := '0';
   elsif CP0'event and CP0='1' then
     q_temp := D0;
   end if;
   Q0 <= q_temp;
   Q0_bar <= not q_temp;
 end process;

 FF2: process(SD1,RD1,CP1)
   variable q_temp:std_logic;
 begin
   if SD1='1' then
     q_temp := '1';
   elsif RD1='1' then
     q_temp := '0';
   elsif CP1'event and CP1='1' then
     q_temp := D0;
   end if;
   Q1 <= q_temp;
   Q1_bar <= not q_temp;
 end process;
end Behavioral;
```

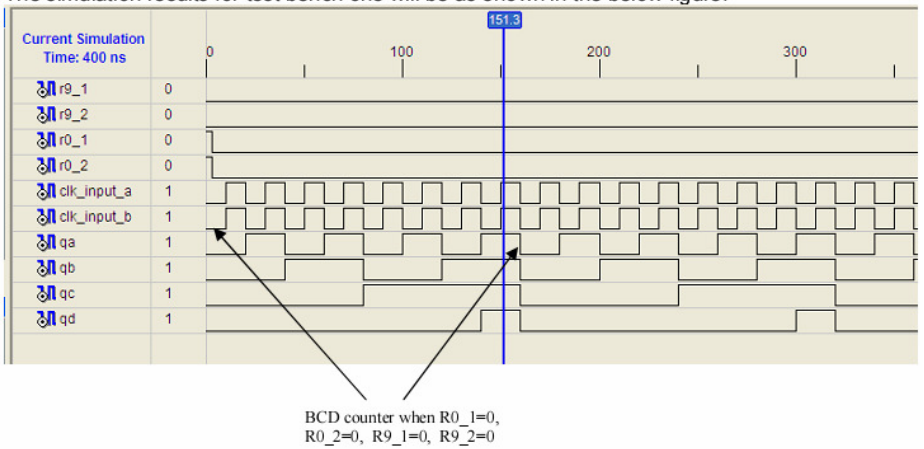



Experiment - 3

- VHDL coding, simulation and FPGA synthesis of the 7490 decade and binary counter



The simulation results for test bench one will be as shown in the below figure.



BCD counter when R0_1=0, R0_2=0, R9_1=0, R9_2=0

Let us continue remaining experiments in LAB

Thank you
(please fill the feedback form)