LAB Assignment 1

Adder Circuits

Dr. Ahmed Elhossini

Introduction

In this lab assignment you are going to implement several adder circuits using VHDL. These circuits should be developed using Xilinx ISE, simulated and compared after synthesis. The purpose of this lab is familiarizing you with VHDL/FPGA design flow and various types of adder circuits discussed in lecture.

Pre-requisites

1. Basic Knowledge of VHDL.
2. Tutorials 1 and 2 (Optional)

Objectives:

1. Model various adder circuits using VHDL.
    1. Carry Ripple Adder.
    2. Conditional Sum Adder.
    3. Carry-Lookahead Adder.
2. Simulate each circuit.
3. Synthesis each circuit using the FPGA flow.
4. Perform timing simulation for each adder circuit
5. Compare the area and critical path delay of each component.
6. Integrate the adder circuit with the Zynq SoC as descripted in tutorials 1 and 2 (optional)

Equipment and Tools

1. Xilinx ISE 14.5 tool chain or later versions.
2. Xilinx EDK tool chain or later versions.
3. The ZedBoard (optional parts only).

Lab Time:

3 Weeks from November 2nd until November 23th 2016.

Lab Steps

Part 1: Carry Ripple Adder

Step 1: Start the ISE:

1. On the sunray login to Ubuntu using your university ID.
2. Open a terminal window.

3. On the terminal enter the following command:
   >> source /afs/tu-berlin.de/units/Fak_IV/aes/scripts/mathenv

This will setup all the required environment variables required to start the ISE.

4. Start the ISE using the command:
   >> ise&

5. Create a new project for Spartan3E XC3S500E-FG320-5 as shown in Figure 1.
6. Add a new VHDL Module to the project for the full adder. The code is shown in Figure 2.

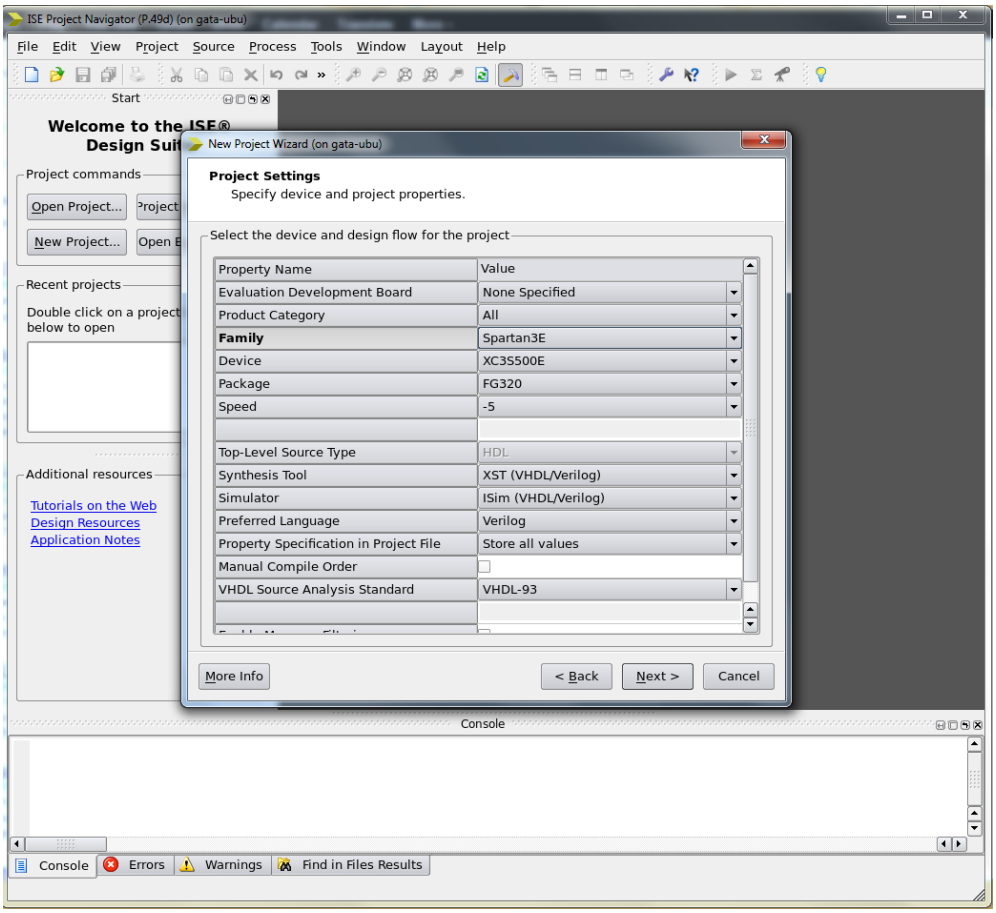

Figure 1: Create a new project using ISE

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

entity FullAdder is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : in  STD_LOGIC;
          s : out  STD_LOGIC;
          co : out  STD_LOGIC);
end FullAdder;

architecture Behavioral of FullAdder is

begin
          s <= a xor b xor c;
          co <= (a and b) or ((a xor b) and c);
end Behavioral;
```

Figure 2 VHDL code for Full Adder circut.

7. Use this module to build the ripple adder. Create another VHDL file and (Adder8bit for example) and use instances of the FullAdder module to build. Sample code for 2-bits ripple adder is shown in *Figure 4*. This code shows you how to use modular design approach to build large circuits. Expand this circuit to build 8-bit ripple adder as shown in *Figure 3* .

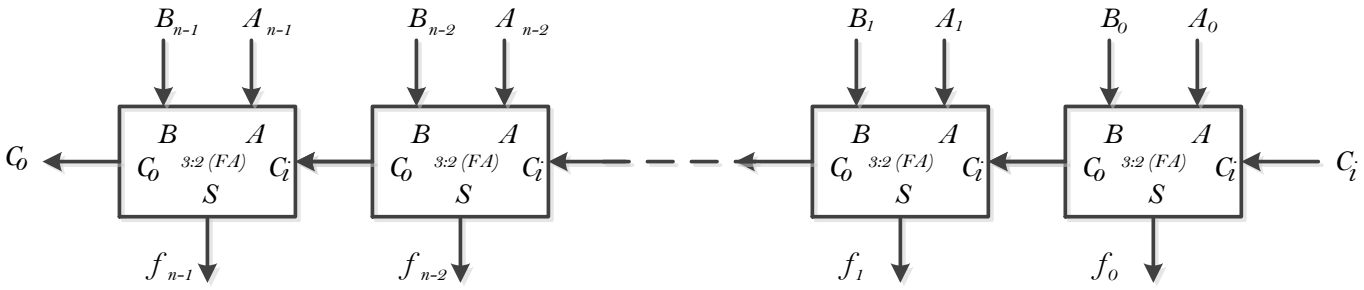

Figure 3 Carry Ripple Adder

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

entity Adder2Bits is
   Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
          b : in  STD_LOGIC_VECTOR (1 downto 0);
          c : in  STD_LOGIC;
          s : out  STD_LOGIC_VECTOR (1 downto 0);
          co : out  STD_LOGIC);
end Adder2Bits;

architecture Behavioral of Adder2Bits is

component FullAdder is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : in  STD_LOGIC;
          s : out  STD_LOGIC;
          co : out  STD_LOGIC);
end component;

signal ci : std_logic;

begin

u1 : FullAdder port map (a=>a(0),b=>b(0),c=>c,s=>s(0),co=>ci);
u2 : FullAdder port map (a=>a(1),b=>b(1),c=>ci,s=>s(1),co=>co);

end Behavioral;
```

Figure 4 VHDL for Ripple Adder 2-bits

Step 2: Simulate the Ripple Adder:

1. Create a VHDL test-bench for your adder. **Right click in you project window-> Select New-Source -> Select VHDL test-bench.**
2. Modify the test bench code to look like *Figure 5*.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Adder2Bits_tb IS
END Adder2Bits_tb;

ARCHITECTURE behavior OF Adder2Bits_tb IS
    COMPONENT Adder2Bits
   PORT(
```

```
      a : IN  std_logic_vector(1 downto 0);
      b : IN  std_logic_vector(1 downto 0);
      c : IN  std_logic;
      s : OUT  std_logic_vector(1 downto 0);
      co : OUT  std_logic
     );
  END COMPONENT;

  signal a : std_logic_vector(1 downto 0) := (others => '0');
  signal b : std_logic_vector(1 downto 0) := (others => '0');
  signal c : std_logic := '0';

  signal s : std_logic_vector(1 downto 0);
  signal co : std_logic;

BEGIN
  uut: Adder2Bits PORT MAP (
       a => a,       b => b,       c => c,
       s => s,       co => co
     );
  stim_proc: process
  begin
    wait for 100 ns;
            c <= '0';
            a <="01";
            b <="10";
    wait for 10us;
            c <= '1';
            a <="11";
            b <="11";
    wait;
  end process;

END;
```

Figure 5 VHDL code for the Testbench

3. The test-bench code allows you to control the simulation process. In the project window select "**Simulation"** then select the test-bench module. Of course you need to modify the test-bench code to match you design.

4. In the process window click **"Behavioral Simulation"** which will start the simulation and run all the steps in the test-bench simulation process as shown in *Figure 6*.

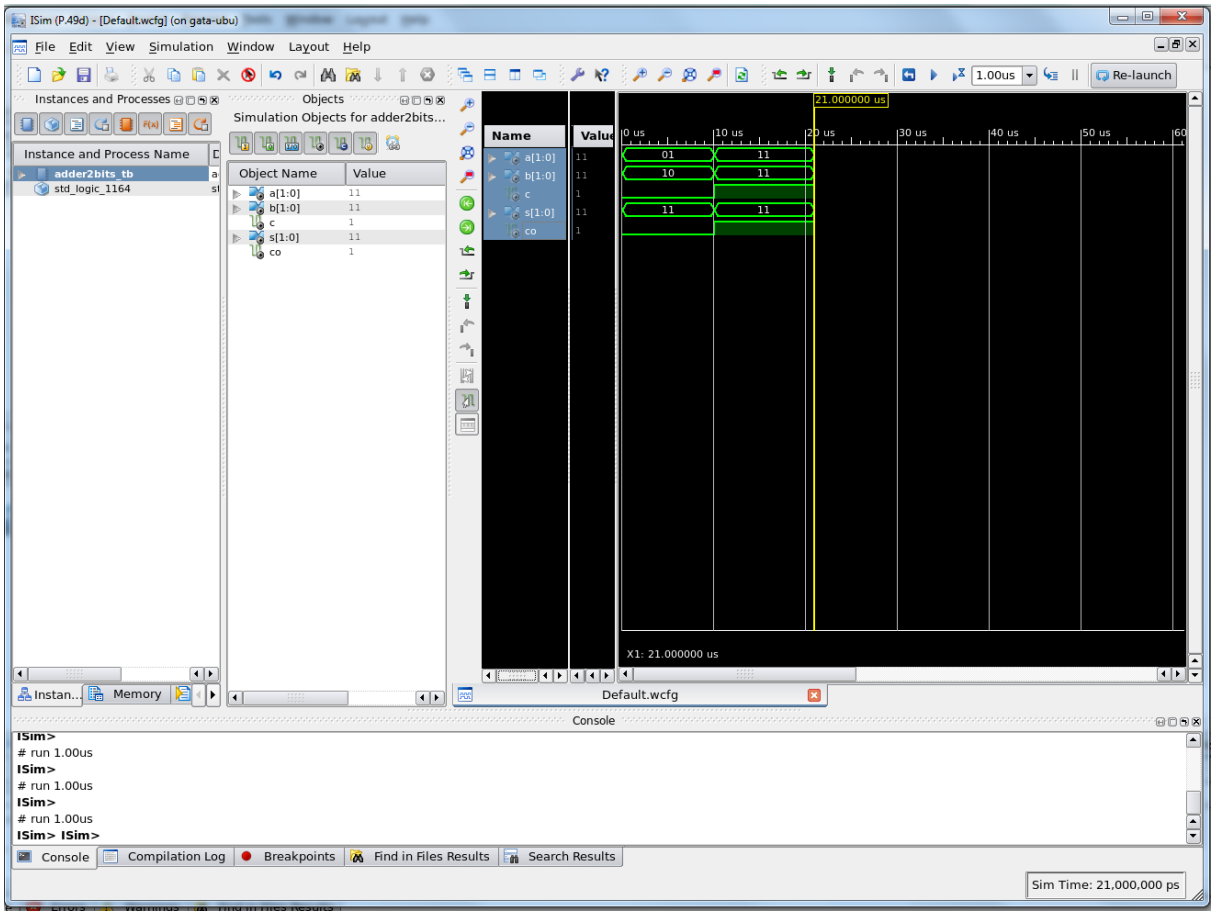

Figure 6 Simulation Results

Step 3: Synthesis The Ripple Adder:

1. Switch back to **"Implementation"** view and click **"Synthesis"** in the process view and then click **"Implement Design"**.
2. When the synthesis complete, check the "**Design Summary"** and record the number of LUT required to build the 8-bit ripple adder.
3. In the **"Design Summary"** look at the static timing option and find the critical path delay.
4. Fill out Table 1.

Step 3: Post-Route simulation

Repeat item 4 of step 1, after selecting **"Post-Route"** in the simulation window. **"Post-Route"** simulation takes into account all timing parameters of each component into account during the simulation. This gives you the close to real behavior of the circuit when deployed to the FPGA.

Table 1 Results

Number of LUTs	
Number of IOs	
Critical Path Delay	

Part 2: Conditional Sum Adder

Repeat the same steps of Part1 for the conditional sum adder discussed in the lectures. Build an 8 bit version of the adder and simulate it. After simulation implement the design and fill out Table 2.

A block diagram of the circuit is shown *Figure 7*. Use the same modular approach in Part 1 to build this architecture.

Table 2 Results for Conditional Sum Adder

Number of LUTs	
Number of IOs	
Critical Path Delay	

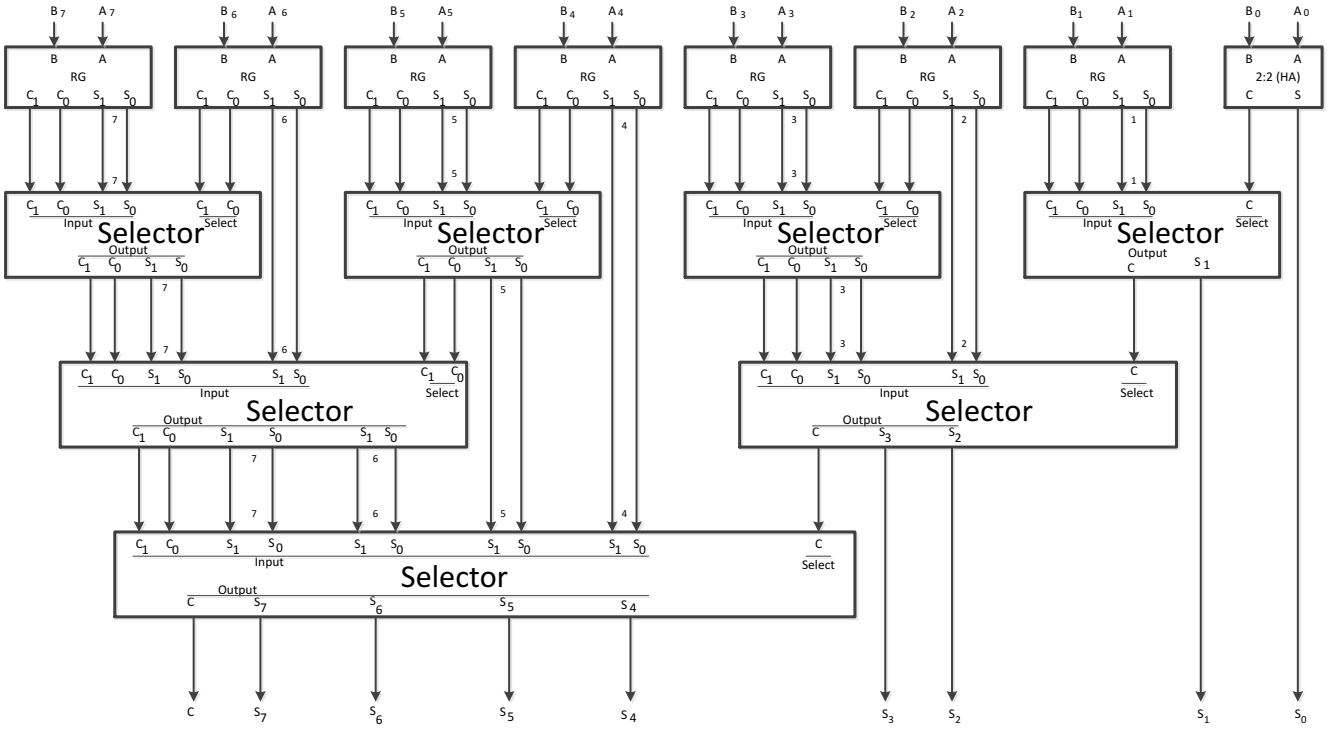

Figure 7 Conditional Sum Adder

Part 3: Carry-Lookahead Adder

Repeat the same steps of Part1 for the carry-lookahead adder discussed in the lectures. Build an 8 bit version of the adder and simulate it. After simulation implement the design and fill out *Table 3*.

Use the same modular approach in Part 1 to build this architecture. For the carry look-adder with larger number of input, the design will be large. Try to use logic reuse.

Table 3 Results for the Carry-Lookahead Adder

Number of LUTs	
Number of IOs	
Critical Path Delay	

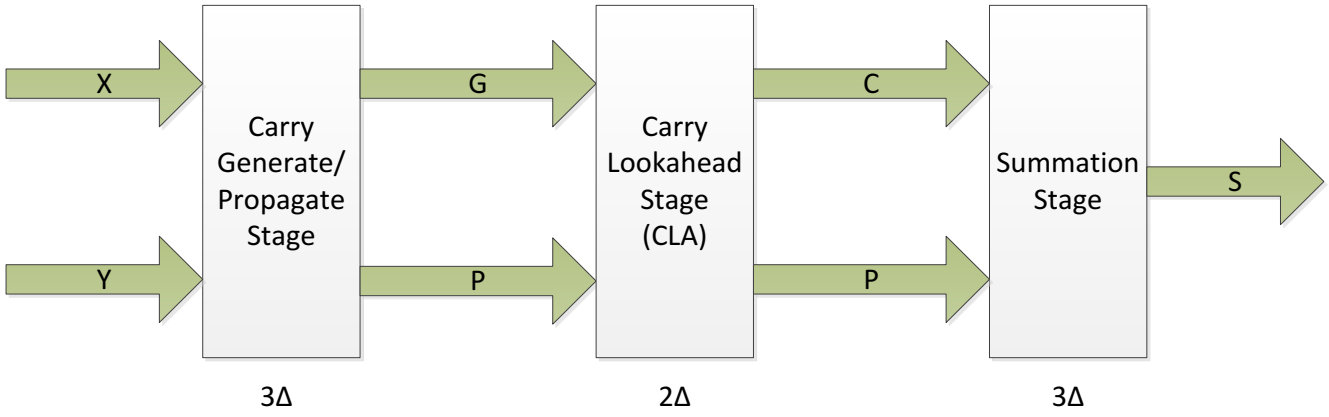

Figure 8 Carry-Lookahead Adder

Important Note:

1. **For each part, use the simulation to verify the function of the adder. Make sure that you gain a correct result. You should demonstrate that all parts are working. Post-route simulation is important to verify timing information of the circuit.**
2. **Report the area and delay numbers for each port along with a snap shot of all simulations and your code and submit this to ISIS2.**
3. **Simulations required:**
    a. **Behavioral simulation for each circuit**

   b. **Timing Simulation (Post place and route). Timing simulation will explain to you the critical path delay.**

4. **For each part try to integrate the circuit with Zynq SoC in the same approach shown in Tutorial 1 and 2. This will allow you to run the circuit on the FPGA.**