# Report of Lab 2

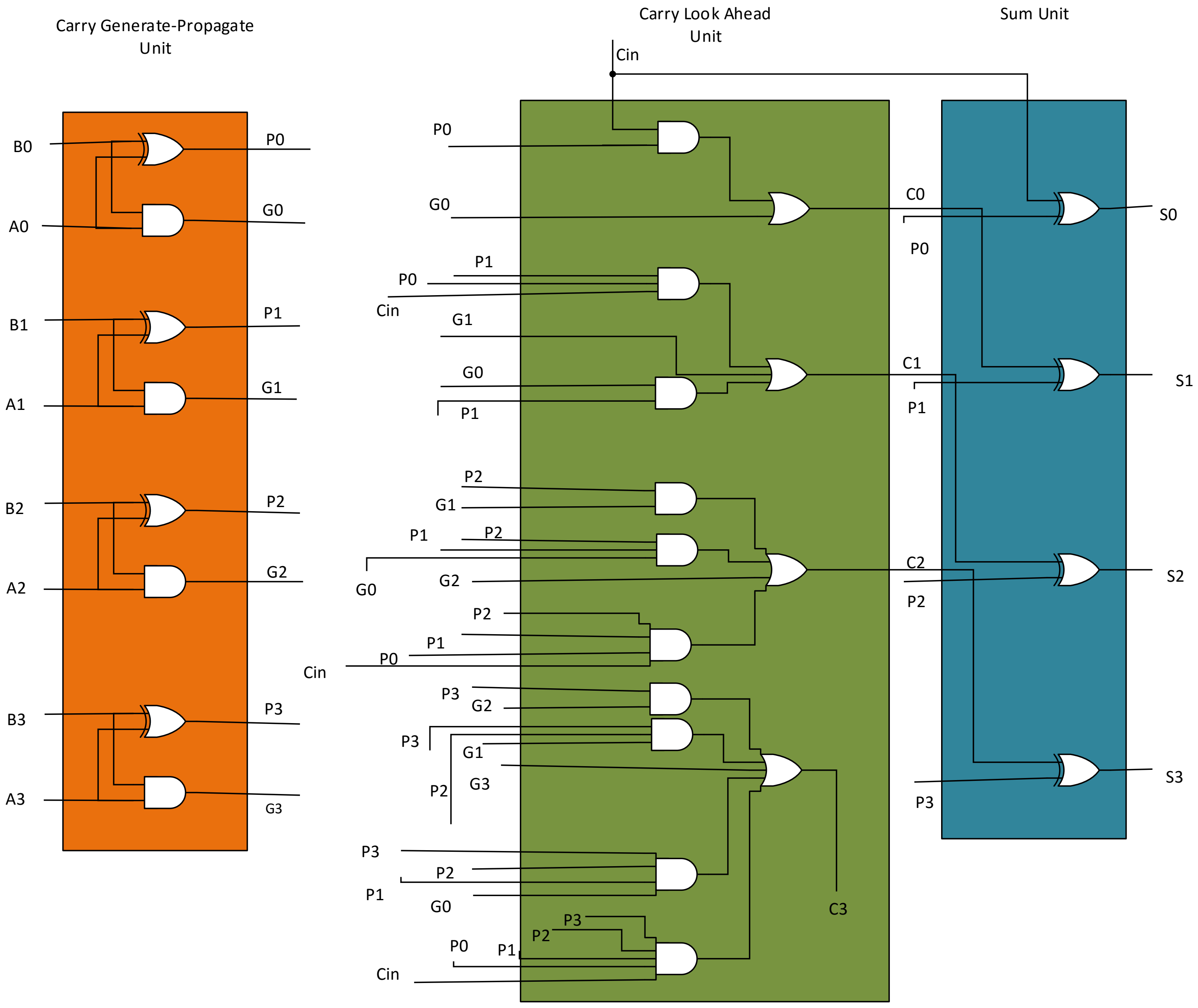
## Team

| |
|---|
| *Kritharakis Emmanouil* |
| *Fotakis Tzanis* |

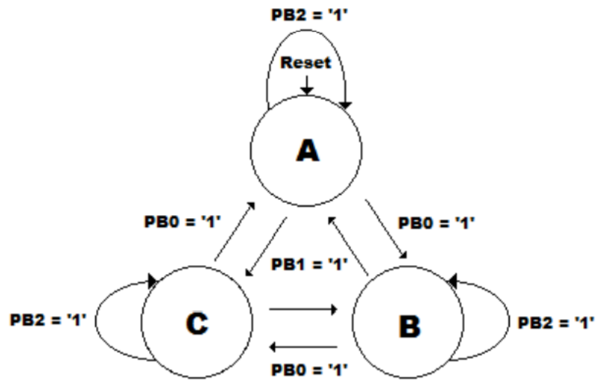
## Preparation

To begin with, the lab was divided into 2 parts. In the first part, all we have to do as a preparation was the diagram of a carry look ahead adder, which was divided into 3 pieces in order to be as specific as it should. In the second part, we should have done the transition state table, which shows how the current state is going to change based on its inputs. Both the diagram of cla adder and transition state table are displayed below in picture 1 and 2 respectively.

### CLA Adder Diagram



## FSM Diagram



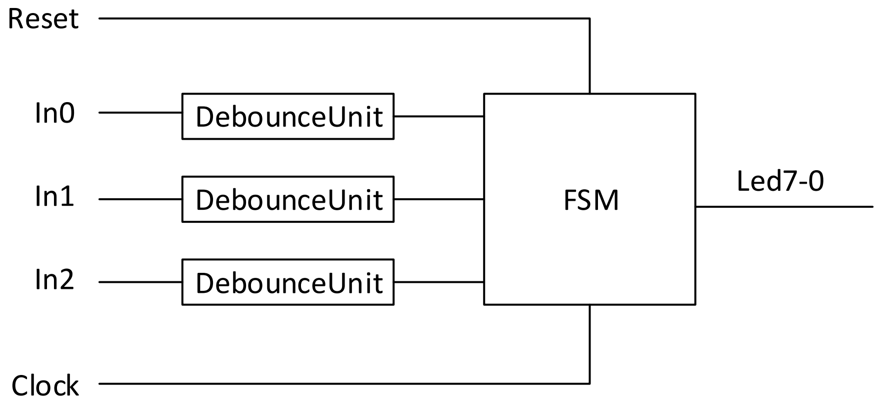
## State Assignment Table

| State | Binary Representation |
|---|---|
| A | 00 |
| B | 01 |
| C | 10 |

### FSM Transition Table

| Current state | Input 1 | | Input 2 | | Input 3 | | Reset | |
|---|---|---|---|---|---|---|---|---|
| | In0=0 | In0=1 | In1=0 | In1=1 | In2=0 | In2=1 | Rst=0 | Rst=1 |
| 00 | 00 | 01 | 00 | 10 | 00 | 00 | 00 | 00 |
| 01 | 01 | 10 | 01 | 00 | 01 | 01 | 01 | 00 |
| 10 | 10 | 00 | 10 | 01 | 10 | 10 | 10 | 00 |
| 11 | 00 | 00 | 00 | 00 | 00 | 00 | 11 | 00 |

### FSM Connection Diagram



## Overview

Having done lab1 we realized the “structural thought” of building a ripple carry adder. The current purpose of the first circuit was to help us come in touch with a carry look ahead adder of 4 bits, which is different from the former as it has less delay. To elaborate it, every single output bit either as a sum bit or as a carry bit didn’t wait till the former bit take its final value (0 or 1), as it relied on its own inputs. So, this first circuit has some inputs (A0, A1, B0, 1, Cin) and some outputs (Sum 0 to 3, Carry 0 to 3 and Carry out). To begin with, in our design we decompose the problem into 3 pieces. First, was the propagate and generate unit, second the carry look ahead unit and last the sum unit. Those 3 component were inserted at the top module file, where through the right port maps of them we successfully design the cla adder.

The purpose of the second circuit was to understand better the pulse mode design through the implementation of a finite state machine (Moore). The second circuit of fsm has some inputs (in0, in1, in2, reset, clock) and some outputs (leds 0 to 7). This time, our design included one fsm component and three so called debouncebutton components, which were parts of a top module file. The fsm has 3 process, one for the changes of current states and next states, one for the outputs and the final one for the reset and clock. The process function was vital for our design as each and every change of signals in the sensitivity list took place, a specific part of code was executed as it should be in a Moore fsm. The reason for the debouncebutton components was to prevent the logical mistake to allow after any input the fsm to change state with the rising edge of clock as long as we press the input's button. So through the debounce button we achieve to allow only one change of state in one press of button, despite the rising edge of clock. Those 4 components were inserted at the top module file, where through the right port maps of them we successfully design the Moore fsm.

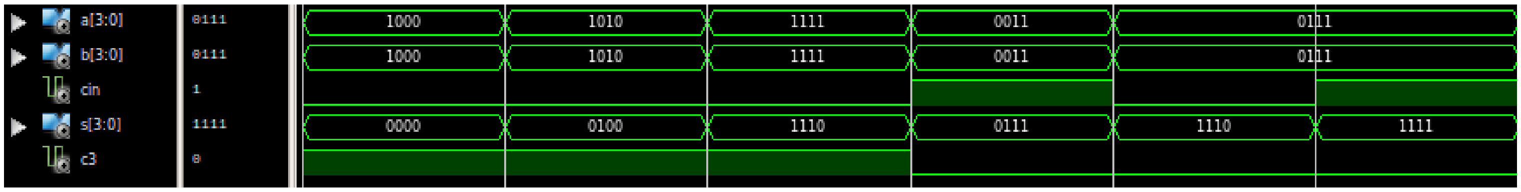
## Waveforms –Simulation

Having done the vhd files for both circuits all we have to do was to create test benches to check if the circuits work as we want. In order to do that, we should take into consideration some of the possible inputs and see if the outputs are the correct ones. The test benches help us as they produce simulations of how the circuit is going to work with specific inputs.

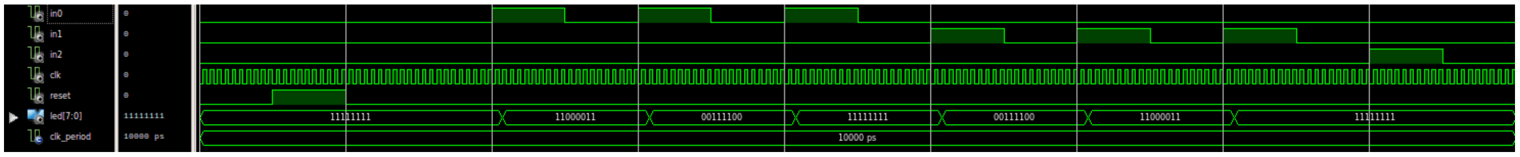
As the possible alternatives of inputs of the first circuit was 512, we decided to simulate only 3-4, which are the most discreet ones. We follow the same idea and in the second circuit, where we progressively see how the fsm was going to react through the different inputs we inserted.

Here are the simulations of the first and second circuit:

### CLA Adder



### FSM



## Conclusion

This second lab exercise help us a lot to understand better the vhdl language and the fpga. First of all, we realize the importance of debounce button in the real time execution and how we should inserted it in a vhdl file. Furthermore, we produce a UCF file (user constrain file), from which we realize the proper way to match the clock of our design with the clock of fpga through a mistake (we insert Loc="C8" instead of "B8" and as a result the speed of clock downgrade to 1 hertz (we saw the problem through the delay of button, considered the possible mistakes and found the solution).

## Code

### CLA Adder

#### *Carry Look Ahead Unit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Carry_Look_Ahead_Unit is
    Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
           G : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           C : out  STD_LOGIC_VECTOR (3 downto 0));
end Carry_Look_Ahead_Unit;
architecture Behavioral of Carry_Look_Ahead_Unit is
begin
 C(0)<=G(0) or (P(0) and Cin);
 C(1)<=G(1) or (P(1) and G(0)) or (P(1) and P(0) and Cin);
 C(2)<=G(2) or (P(2) and G(1)) or (P(2) and P(1) and G(0)) or (P(2) and P(1) and P(0) and Cin);
 C(3)<=G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or (P(3) and P(2) and P(1) and G(0)) or (P(3) and P(2) and P(1) and P(0) and Cin);
end Behavioral;
```

#### *Carry Generate Propagate Unit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Carry_Generate_Propagate_Unit is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           P : out  STD_LOGIC_VECTOR (3 downto 0);
           G : out  STD_LOGIC_VECTOR (3 downto 0));
end Carry_Generate_Propagate_Unit;
architecture Behavioral of Carry_Generate_Propagate_Unit is
begin
      P(0) <= A(0) xor B(0);
                  G(0) <= A(0) and B(0);
                              P(1) <= A(1) xor B(1);
                  G(1) <= A(1) and B(1);
                              P(2) <= A(2) xor B(2);
                  G(2) <= A(2) and B(2);
                              P(3) <= A(3) xor B(3);
                  G(3) <= A(3) and B(3);
end Behavioral;
```

#### *Sum Unit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Sum_Unit is
    Port ( C : in  STD_LOGIC_VECTOR (2 downto 0);
           P : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0));
end Sum_Unit;
architecture Behavioral of Sum_Unit is
begin
 S(0)<=P(0) xor Cin ;
 S(1)<=P(1) xor C(0);
 S(2)<=P(2) xor C(1);
 S(3)<=P(3) xor C(2);
end Behavioral;
```

### *Four Bit CLA Adder*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Fulladder_4_bits is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0);
           C3 : out  STD_LOGIC);
end Fulladder_4_bits;
architecture Behavioral of Fulladder_4_bits is
 component Carry_Generate_Propagate_Unit
   port(
                a,b : in std_logic_vector(3 downto 0);
                                p,g : out std_logic_vector(3 downto 0)
               );
 end component;
 component Carry_Look_Ahead_Unit
   port(
                                p,g : in std_logic_vector(3 downto 0);
                                cin : in std_logic ;
                                c : out std_logic_vector(3 downto 0)
               );
 end component;
 component Sum_Unit
   port(
                c : in std_logic_vector(2 downto 0);
                                p : in std_logic_vector(3 downto 0);
                                cin :in std_logic;
                                s : out std_logic_vector(3 downto 0)
               );
 end component;
 signal prop : std_logic_vector(3 downto 0);
 signal gener : std_logic_vector(3 downto 0);
 signal car : std_logic_vector(2 downto 0);
begin
 CGP: Carry_Generate_Propagate_Unit port map(a=>A,b=>B,p=>prop,g=>gener);
 CLA: Carry_Look_Ahead_Unit port map(p=>prop,g=>gener,cin=>Cin,c(0)=>car(0),c(1)=>car(1),c(2)=>car(2),c(3)=>C3);
 Sumy: Sum_Unit port map(c=>car,p=>prop,cin=>Cin,s=>S);
end Behavioral;
```

## FSM

### *FSM Unit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity The_fsm is
    Port ( CLOCK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           IN0 : in  STD_LOGIC;
           IN1 : in  STD_LOGIC;
           IN2 : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 0));
end The_fsm;
architecture Behavioral of The_fsm is
type state is (a,b,c);
signal current_state,next_state:state;
        component debouncebutton
```

```
                port (
                                                clk                     : in std_logic;                         -- connect it to the Clock of the board
                                                rst                     : in std_logic;                         -- connect it to the Reset Button of the board

                                                input         : in std_logic;                     -- connect it to the Push Button of the board
                                                output   : out std_logic                          -- connect it to your circuit
                                    );
               end component;
begin
  process(current_state,IN0,IN1,IN2)
  begin
   next_state<=a; --for sake of warning 737
   case current_state is
               when a =>
                        if IN0='1' then
                                                          next_state<=b;
                                                        elsif IN1='1' then
                                                          next_state<=c;
                                                        elsif IN2='1' then
                                                          next_state<=a;
                                                        else
                                                          next_state<=a;
                                                        end if;
               when b =>
                        if IN0='1' then
                                                           next_state<=c;
                                                        elsif IN1='1' then
                                                           next_state<=a;
                                                        elsif IN2='1' then
                                                           next_state<=b;
                                                        else
                                                          next_state<=b;
                                                        end if;
               when c =>
                        if IN0='1' then
                                                          next_state<=a;
                                                        elsif IN1='1' then
                                                          next_state<=b;
                                                        elsif IN2='1' then
                                                          next_state<=c;
                                                        else
                                                          next_state<=c;
                                                        end if;

               when others=> next_state<=a;
              end case;
  end process;
  process(current_state)
  begin
   case current_state is

                when a => LED<=("11111111");
                when b => LED<=("11000011");
                when c => LED<=("00111100");

              end case;
  end process;
process(CLOCK,RST)
  begin
    if (RST='1') then current_state<=a;
```

```
                elsif rising_edge(CLOCK) then current_state<=next_state;
                end if;
  end process;
end Behavioral;
```

## Top Level FSM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity the_top_fsm is
    Port ( in0 : in  STD_LOGIC;
           in1 : in  STD_LOGIC;
           in2 : in  STD_LOGIC;
           led : out  STD_LOGIC_VECTOR (7 downto 0);
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC);
end the_top_fsm;
architecture Behavioral of the_top_fsm is
                component debouncebutton
                  port (
                                                        clk                     : in std_logic;                    -- connect it to the Clock of the board
                                                        rst                     : in std_logic;                    -- connect it to the Reset Button of the board

                                                        input         : in std_logic;                    -- connect it to the Push Button of the board
                                                        output   : out std_logic                         -- connect it to your circuit
                                        );
                end component;
                component The_fsm
                  port (
                                                                                CLOCK : in  STD_LOGIC;
                                                                                RST : in  STD_LOGIC;
                                                                                IN0 : in  STD_LOGIC;
                                                                                IN1 : in  STD_LOGIC;
                                                                                IN2 : in  STD_LOGIC;
                                                                                LED : out  STD_LOGIC_VECTOR (7 downto 0)
                                 );
                end component;
   signal s0,s1,s2: std_logic;
begin
  --instantiation of debounce buttons
  debounce_button_1:debouncebutton port map(clk=>clk,rst=>reset,input=>in0,output=>s0);
  debounce_button_2:debouncebutton port map(clk=>clk,rst=>reset,input=>in1,output=>s1);
  debounce_button_3:debouncebutton port map(clk=>clk,rst=>reset,input=>in2,output=>s2);
   --instantiation of fsm
  fsm:The_fsm port map(CLOCK=>clk,RST=>reset,IN0=>s0,IN1=>s1,IN2=>s2,LED=>led);
end Behavioral;
```