Experiment 3: BCD Number Subtractor

Debasish Panda 21D070021

August 30, 2022

1 Overview of the experiment

The purpose of the experiment is to design a BCD Number Subtractor using fundamental logic gates and 4-bit Ripple Adders as well as describe them in VHDL using Structural Modelling. Following a successful RTL simulation, we have to demonstrate the same on Xen10 FPGA Board.
In order to perform the experiment, first the BCD Subtractor is designed using AND, OR and INVERTER gates along with Ripple Adders (which can also be used as Subtractors by changing the Select input). A rough sketch of the digital circuit is drawn first. The appropriate code for Structural Modelling of these components is written using VHDL on Quartus Prime software. Next, we have to demonstrate the BCD Subtractor practically by using the Xen10 FPGA board. In order to run the VHDL code on the board, we also need to determine the input and output pins correctly as per the specifications of the circuit board.
In this report first we discuss the setup required for the experiment and the approach used by me to solve the assignment. Attached alongwith are the pen-and-paper diagram and picture showing the simulation output of the design.

2 Experimental setup

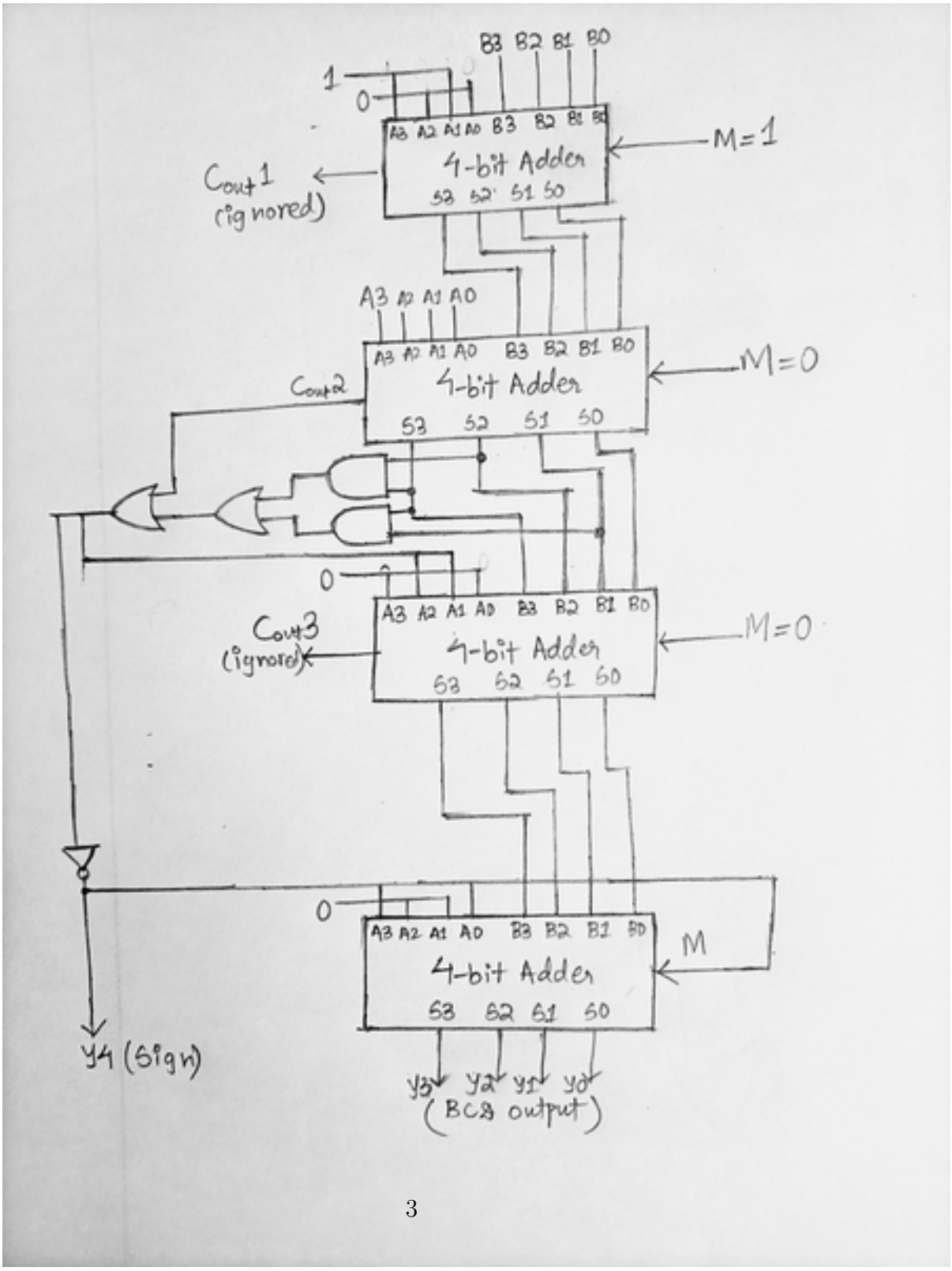
First we need to design the Fibonacci using logic gates and Ripple Adders. The required design specifications for the BCD Subtractor are-

(a) A 4-bit binary Subtractor is used to obtain the 10's complement of the subtrahend.
(b) A 4-bit binary Adder is also used to add A and the 10's complement of B.
(c) An appropriate logic circuit is used to determine sum which is greater than 9.
(d) A 4-bit binary Adder is utilised to add $(0110)_2$ to the sum if sum generated is greater than 9 or carry is 1.
(e) A 4-bit binary Adder-Subtractor is used to decide the inputs properly to get the output in desired BCD format. The rules to do so are as follows:

(i) If carry is generated, the obtained result is positive. Discard the carry to get the result.

(ii) If carry is not generated, the obtained result is negative, find the 10's complement to get the final result.

The pen-and-paper diagram of the BCD Number Subtractor is:


B3 B2 B1 B0
1
0
A3 A2 A1 A0 B3 B2 B1 B0
4-bit Adder
S3 S2 S1 S0
M=1
Cout1
(ignored)
A3 A2 A1 A0
A3 A2 A1 A0 B3 B2 B1 B0
4-bit Adder
S3 S2 S1 S0
M=0
Cout2
0
A3 A2 A1 A0 B3 B2 B1 B0
4-bit Adder
S3 S2 S1 S0
M=0
Cout3
(ignored)
0
A3 A2 A1 A0 B3 B2 B1 B0
4-bit Adder
S3 S2 S1 S0
M
y4 (Sign)
y3 y2 y1 y0
(BCD output)

Now that we have the required circuit, we need to assign various component instances, architecture and signals as per VHDL syntax. The required code for the same is discussed in the next section.

2.1 Design Documentation and VHDL Code

The VHDL code for designing the entity-architecture pairs associated with the digital circuit is as follows:

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

library work;
use work.Gates.all;

entity HA is
 port (A, B: in std_logic; S, C: out std_logic);
end entity HA;

architecture Simple of HA is
signal T: std_logic;

for all: XOR_2
use entity work.XOR_2(Equations);

begin
 XOR1: XOR_2 port map (A => A, B => B, Y => S);
 NAND1: NAND_2 port map (A => A, B => B, Y => T);
 NAND2: NAND_2 port map (A => T, B => T, Y => C);
end Simple;

--------------------------------------------------------------------------------

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

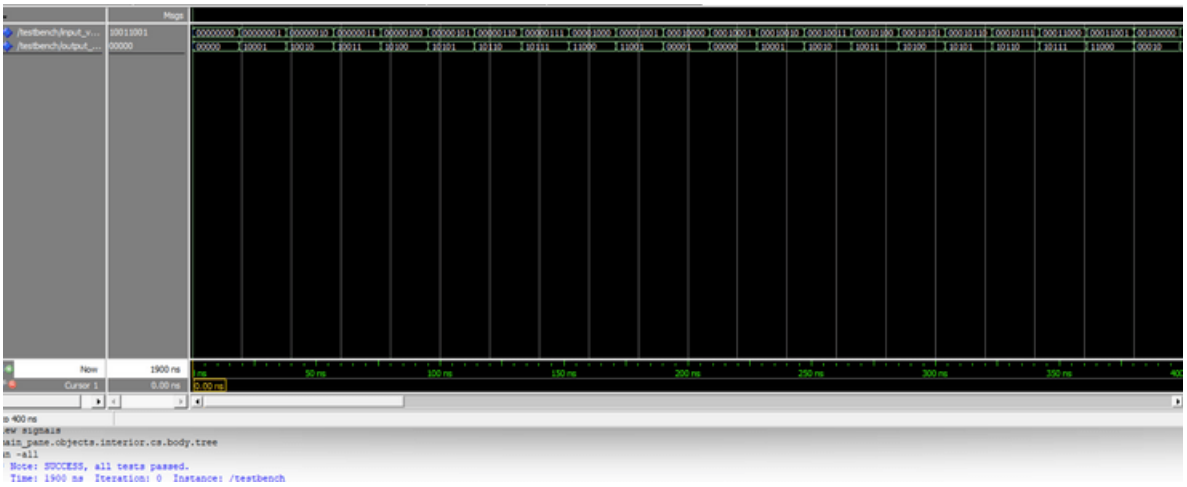
```
library Work;
use work.Gates.all;

entity FA is
 port (A, B, C1: in std_logic; Sum, C2: out std_logic);
end entity FA;

architecture Easy of FA is
signal T1, T2, T3: std_logic;

   component HA is
port (A, B: in std_logic; S, C: out std_logic);
end component HA;

   for all: HA
   use entity work.HA(Simple);

for all: OR_2
use entity work.OR_2(Equations);

begin
 HA1: HA port map (A => A, B => B, S => T1, C => T2);
 HA2: HA port map (A => T1, B => C1, S => Sum, C => T3);
 OR1: OR_2 port map (A => T3, B => T2, Y => C2);
end Easy;

--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

library Work;
use work.Gates.all;

entity Ripple_Adder is
 port(A3, A2, A1, A0, B3, B2, B1, B0, M: in std_logic; S3, S2, S1, S0, Cout: out
 std_logic);
end entity Ripple_Adder;
```

```
architecture Complex of Ripple_Adder is
signal P2, P1, P0, Q3, Q2, Q1, Q0: std_logic;

   component FA is
port (A, B, C1: in std_logic; Sum, C2: out std_logic);
end component FA;

   for all: FA
use entity work.FA(Easy);

for all: XOR_2
use entity work.XOR_2(Equations);

begin
 FA1: FA port map (A => A0, B => Q0, C1 => M, Sum => S0, C2 => P0);
 FA2: FA port map (A => A1, B => Q1, C1 => P0, Sum => S1, C2 => P1);
 FA3: FA port map (A => A2, B => Q2, C1 => P1, Sum => S2, C2 => P2);
 FA4: FA port map (A => A3, B => Q3, C1 => P2, Sum => S3, C2 => Cout);
 XOR1: XOR_2 port map(A => M, B => B0, Y => Q0);
 XOR2: XOR_2 port map(A => M, B => B1, Y => Q1);
 XOR3: XOR_2 port map(A => M, B => B2, Y => Q2);
 XOR4: XOR_2 port map(A => M, B => B3, Y => Q3);
end Complex;

--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

library Work;
use work.Gates.all;

entity BCD_Subtractor is
 port(A3, A2, A1, A0, B3, B2, B1, B0: in std_logic; Y4, Y3, Y2, Y1, Y0: out std_lo
end entity BCD_Subtractor;

architecture Devil of BCD_Subtractor is
```

```
signal S3, S2, S1, S0, P3, P2, P1, P0, U3, U2, U1, U0, R1, R2, Q, S, T, Cout1, Cou

 component Ripple_Adder is
 port (A3, A2, A1, A0, B3, B2, B1, B0, M: in std_logic; S3, S2, S1, S0, Cout: out
 end component Ripple_Adder;

 for all: Ripple_Adder
 use entity work.Ripple_Adder;

 for all: OR_2
 use entity work.OR_2(Equations);

 for all: AND_2
 use entity work.AND_2(Equations);

 for all: INVERTER
 use entity work.INVERTER(Equations);

 for all: XOR_2
 use entity work.XOR_2(Equations);

begin
 RA1: Ripple_Adder port map (A3 =>'1', A2 => '0', A1 => '1', A0 => '0', B3 => B3,
 RA2: Ripple_Adder port map (A3 => A3, A2 => A2, A1 => A1, A0 => A0, B3 => S3, B2
 RA3: Ripple_Adder port map (A3 => '0', A2 => Q, A1 => Q, A0 => '0', B3 => P3, B2
 RA4: Ripple_Adder port map (A3 => T, A2 => '0', A1 => T, A0 => '0', B3 => U3, B2
 AND1: AND_2 port map (A => P3, B => P2, Y => R1);
 AND2: AND_2 port map (A => P1, B => P3, Y => R2);
 OR1: OR_2 port map (A => R1, B => R2, Y => S);
 OR2: OR_2 port map (A => S, B => Cout2, Y => Q);
 INV1: INVERTER port map (A => Q, Y => T);
 Y4 <= T;
end Devil;
```

3 Observations

The experiment was conducted successfully, and the expected output was obtained through the RTL Simulation. The expected output of the BCD Subtractor is shown in the form of the truth table below, alongwith the final RTL Simulation result of the design.



RTL Simulation of BCD Number Subtractor

Following this, we have to do the pin planning correctly and convert our VHDL file to a .svf file, which is executable by the Xen10 board.

4 References

[1] EE214 Github page

[2] Overleaf LaTeX tutorial for beginners

[3] Introduction to Xen10 Board