ECGR 4146 Introduction to VHDL
Lab 5 – Ripple Carry Adder
Jonathon Nguyen
ID: 801093003

Objective of the lab:

The objective of the lab is to make a ripple carry adder that can add two 8-bit number. The adder will output a sum and a carry. The ripple carry adder will be contructed using structural modeling.

VHDL code for the Half Adder:

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

entity halfadder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sum : out STD_LOGIC;
           Cout : out STD_LOGIC);
end halfadder;

architecture Behavioral of halfadder is
begin

    sum <= a xor b;
    Cout <= a and b;

end Behavioral;
```

The entity of the half adder consist of a two input, A and B. It also consist two output which are the sum and the Cout. A and B are one bit each and will add together and the sum will be on the sum port. If there is a carry from the summation then the Cout will have a 1 on the port.

The architecture is using a dataflow model. The circuit was realized on the lab assignment, and I used that to make the architecture. The sum output is just a xor of A and B. The carry is just AND of A and B.

VHDL code for the Full Adder:

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

entity fulladder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           Cin : in STD_LOGIC;
           sum : out STD_LOGIC;
           Cout : out STD_LOGIC);
end fulladder;

architecture Behavioral of fulladder is
    component halfadder is
        Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sum : out STD_LOGIC;
           Cout : out STD_LOGIC);
    end component;

    signal s1, c1, c2: std_logic;
begin

   halfadd1: halfadder
   port map (a => a, b => b, sum => s1, Cout => c1);

   halfadd2: halfadder
   port map (a => s1, b => Cin, sum => sum, Cout => c2);

   Cout <= c1 or c2;

end Behavioral;
```

The entity is a little more complex than the half adder, there is one more input port. The input is now A, B and Cin. Cin allows the adder to add a carry bit with A and B. However the output port is the same as the Half adder with sum port and Cout.

A architecture of the full adder first starte with having the half adder as a component. Three signals are defined to connect the half adder together.
In main body of the architecture, two half adder are instantiated. One of the half adder is connected to the input of the full adder ports. The output of the first half adder is connected to the signals. The signals are connected to the input of the next half adder and the output are connected to the output of the full adder.

To calculate the Cout bit, the signal c1 and c2 are OR.

VHDL code for the 8-bit Ripple Carry Adder:

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

entity ripplecarryadder is

    generic (WIDTH: positive := 7);
    Port ( a : in STD_LOGIC_VECTOR (WIDTH downto 0);
           b : in STD_LOGIC_VECTOR (WIDTH downto 0);
           sum : out STD_LOGIC_VECTOR (WIDTH downto 0);
           Cout : out STD_LOGIC);

end ripplecarryadder;

architecture Behavioral of ripplecarryadder is

    component halfadder is
        Port ( a : in STD_LOGIC;
               b : in STD_LOGIC;
               sum : out STD_LOGIC;
               Cout : out STD_LOGIC);
    end component;

    component fulladder is
        Port ( a : in STD_LOGIC;
               b : in STD_LOGIC;
               Cin : in STD_LOGIC;
               sum : out STD_LOGIC;
               Cout : out STD_LOGIC);
    end component;

    signal c: std_logic_vector(WIDTH+1 downto 1) := (others => '0');

begin

    halfadd1: halfadder
    port map (a => a(0), b => b(0), sum => sum(0), Cout => c(1));

    fulladders: for i in 1 to WIDTH generate
        uut: FullAdder
        port map (a => a(i), b => b(i), cin => c(i), sum => sum(i), cout => c(i+1));
    end generate;

    cout <= c(7);

end Behavioral;
```

The ripple carry has a entity that start with a generic that define WIDTH to 7. I used this generic to make input A and B a std_logic_vector(WIDTH downto 0) and the output sum std_logic_vector(WIDTH downto 0). The output Cout is one bit.
The architecture of the ripple carry adder loads both the half adder and the full adder. The 8-bit ripple carry adder need 7 full adder and a half adder. The one half adder was instantiated the normal way. However, the full adder is generated using the generate function and a loop. The loop is defined from 1 to WIDTH.

Testbench code for the Half Adder:

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

entity halfadder_tb is
--  Port ( );
end halfadder_tb;

architecture Behavioral of halfadder_tb is

    component halfadder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sum : out STD_LOGIC;
           Cout : out STD_LOGIC);
    end component;

    signal a, b, sum, Cout: std_logic;

begin

    UUT: halfadder port map (a => a, b => b, sum => sum, Cout => Cout);

    process
        variable temp : std_logic_vector(1 downto 0);
    begin

    for i in 0 to 3 loop
            temp := std_logic_vector(to_unsigned(i, 2));
            a <= temp(0);
            b <= temp(1);
            wait for 20 ns;
    end loop;
    end process;


end Behavioral;
```

In the architecture, the half adder was defined as a component with all the ports it needs. Three different signals was made to see the input and output of the half adder. A for loop was used to loop through all 4 input states of the half adder.

Testbench code for the Full Adder:

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

entity fulladder_tb is
--  Port ( );
end fulladder_tb;

architecture Behavioral of fulladder_tb is

    component fulladder is
        Port ( a : in STD_LOGIC;
               b : in STD_LOGIC;
               Cin : in STD_LOGIC;
               sum : out STD_LOGIC;
               Cout : out STD_LOGIC);
    end component;

    signal a, b, Cin, sum, cout : STD_Logic;

begin

    UUT: fulladder port map (a => a, b => b, Cin => Cin, sum => sum, Cout => cout);

    process
        variable temp : std_logic_vector(2 downto 0);
    begin

    for i in 0 to 7 loop
            temp := std_logic_vector(to_unsigned(i, 3));
            a <= temp(0);
            b <= temp(1);
            Cin <= temp(2);
            wait for 100 ns;
    end loop;
    end process;

end Behavioral;
```

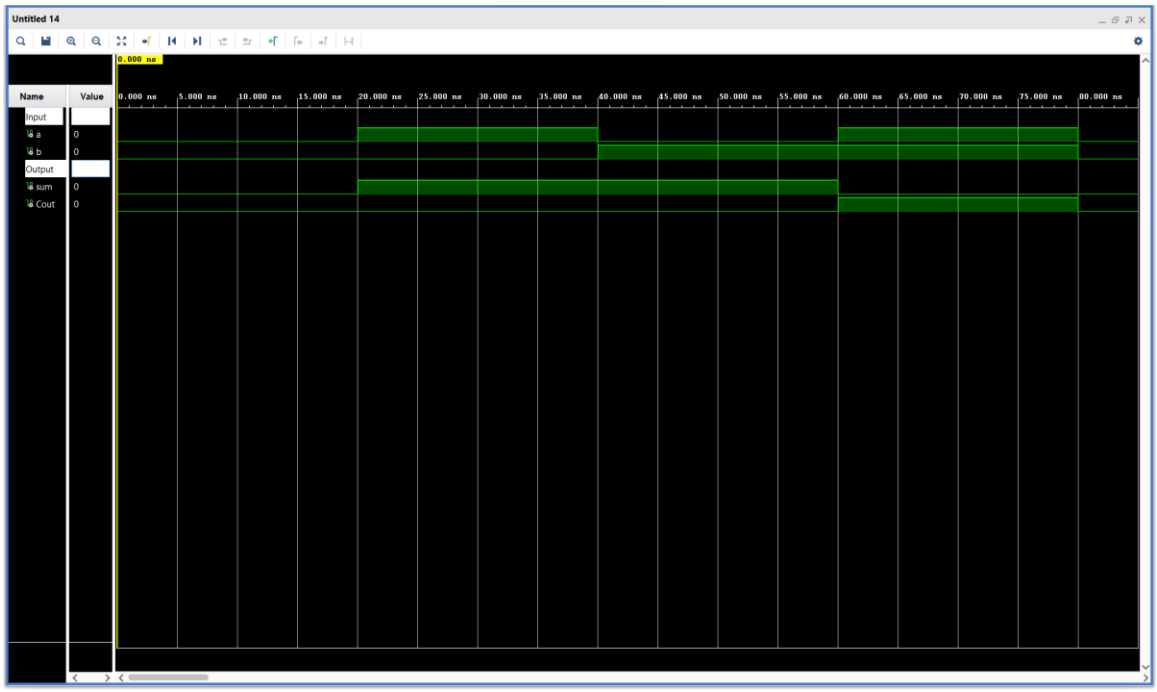
Same logic as the half adder testbench. In the architecture, the full adder was defined as a component with all the ports it needs. A for loop was used to loop through all 8 input states of the half adder.

Testbench code for the Half Adder:

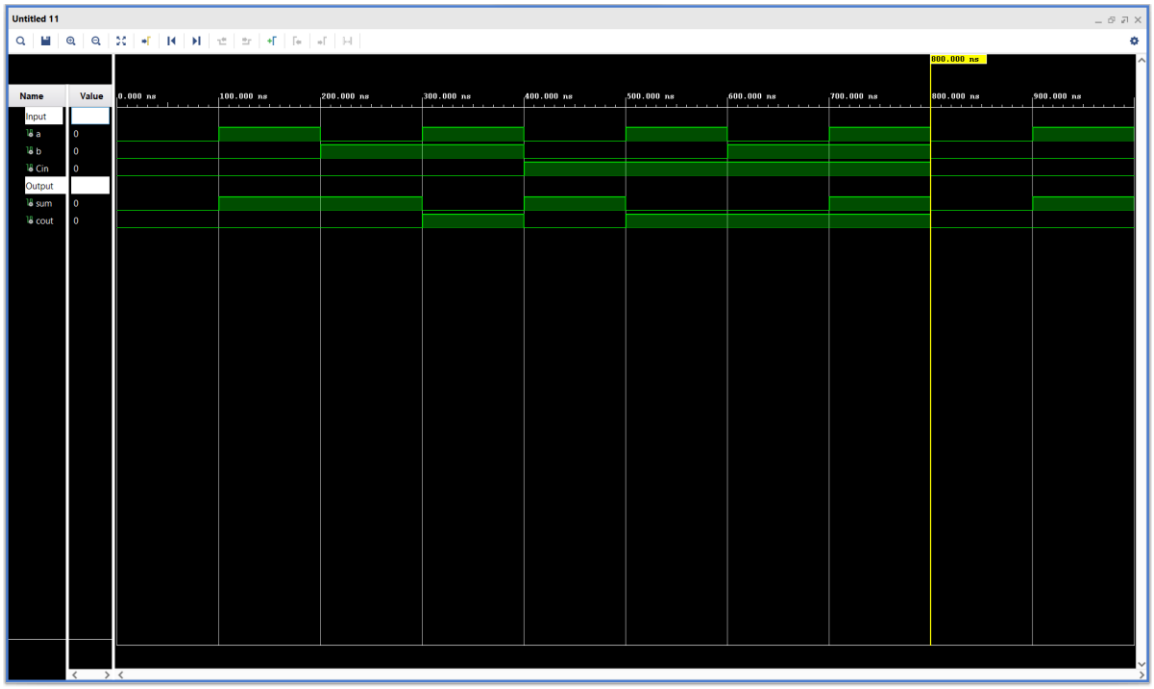
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ripplecarry_tb is
-- Port ( );
end ripplecarry_tb;

architecture Behavioral of ripplecarry_tb is
    component ripplecarryadder is
    generic ( WIDTH : positive := 7);
    Port (  a : in STD_LOGIC_VECTOR (WIDTH downto 0);
            b : in STD_LOGIC_VECTOR (WIDTH downto 0);
            sum : out STD_LOGIC_VECTOR (WIDTH downto 0);
            Cout : out STD_LOGIC);
    end component;
signal A, B, sum : STD_Logic_vector(7 downto 0);
signal cout : STD_Logic;
begin

    UUT : ripplecarryadder port map (a => A, b => B, sum => sum, Cout => cout);
    process
    begin

    A <= "00000000";
    B <= "00000000";
    wait for 100 ns;

    A <= "00000000";
    B <= "00000001";
    wait for 100 ns;

    A <= "11100000";
    B <= "11111111";
    wait for 100 ns;

    A <= "00010101";
    B <= "00100101";
    wait for 100 ns;

    A <= "10100111";
    B <= "00000011";
    wait for 100 ns;

    A <= "01101111";
    B <= "11100101";
    wait for 100 ns;

    A <= "00000100";
    B <= "00000010";
    wait for 100 ns;

    A <= "11100111";
    B <= "01110111";
    wait for 100 ns;

    end process;
end Behavioral;
```

Ripple adder was defined as a component. I could not use a for loop this testbench due to the amount of states that the adder could be in. Instead I defined eight different test with 100 nanosecond inbetween each test.

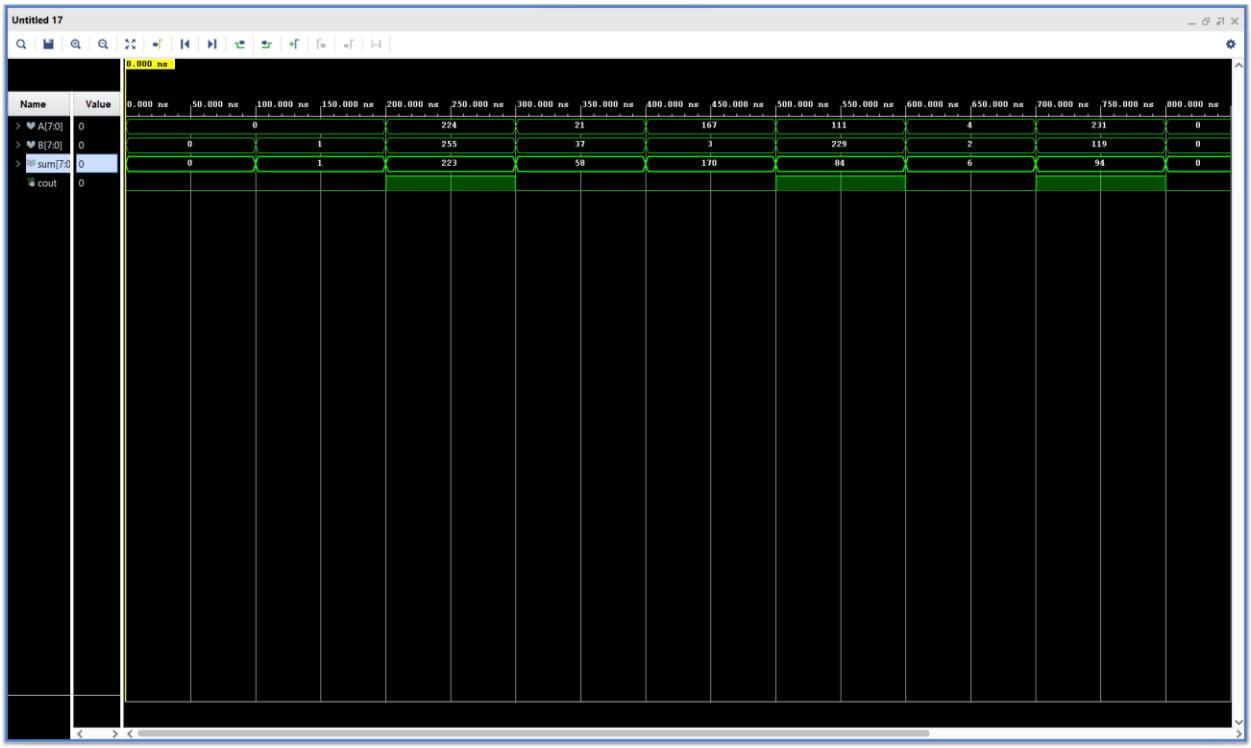
Simulation Waveform for the Half Adder:



Simulation Waveform for the Full Adder:



Simulation Waveform for the 8-bit Ripple Carry Adder:



Conclusion

A half adder was coded. From the half adder, the full adder was made with two half adder. Then the ripple carry adder was made from the half and full adder. When running the all the testbench made for three adders, they all preform as expected. They all add properly and produced a carry bit if needed.