Experiment 0: 4-bit Ripple Carry Adder

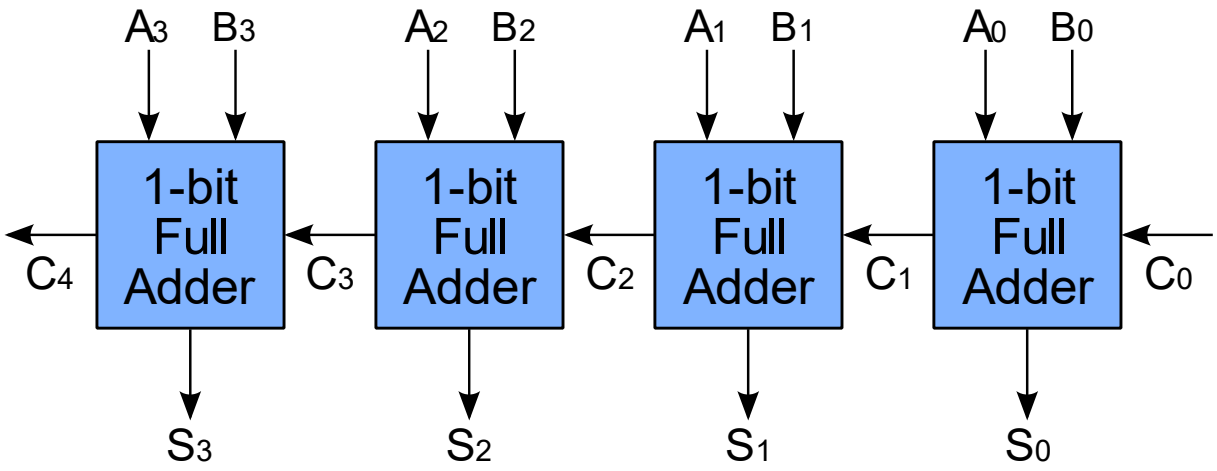
Rathour Param Jitendrakumar Roll Number: 190070049
EE-214, WEL, IIT Bombay
January 29, 2021

Overview of the experiment:

The purpose of this experiment is to add two 4-bit numbers using Ripple Carry Adder using given Full Adder as a component in VHDL and simulate the adder using the generic testbench using Quartus.
I designed the circuit and generated the trace file for all 256 combinations using Python as well as Julia

This report contains my approach to the experiment, design of circuit with relevant code followed by DUT input-output format, RTL netlist view, verified by output waveforms using RTL and Gate Level Simulation

Approach to the experiment:



4-bit adder Ripple Carry Adder is implemented by using a Full Adder for each bit and connecting the carry-out of a given stage to the next stage's carry-in, as shown in the above figure.

Design document and VHDL code if relevant:

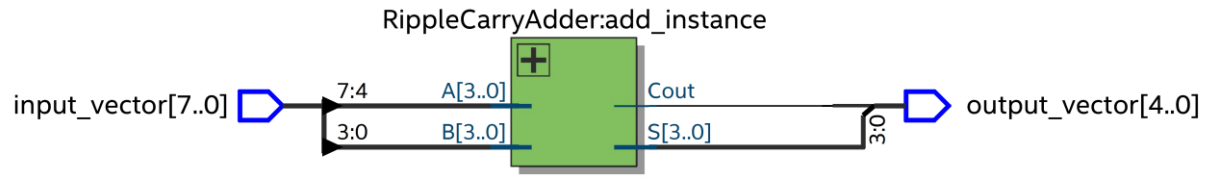
This whole design is part of Device Under Test with 8 input bits & 5 output bits which confirm our design. See DUT Input/Output Format: I constructed an entity named RippleCarryAdder following the above approach (Code given on next page). This contains 4 cascaded full adders (1-bit)
Input carry to first full adder taken as '0' and subsequent C_{in}'s as previous stage's C_{out}
The i^{th} full adder (FAi) gives i^{th} output bit from right (S_{i-1}) and 4^{th} full adder (FA4) gives resultant carry.
Each Full adder is made of 2 half adders and a OR gate.
Each Half adder is made of a XOR gate and a AND gate.

```
architecture Struct of RippleCarryAdder is
  signal C: std_logic_vector(2 downto 0);
begin

  FA1: FULL_ADDER port map (A => A(0), B => B(0), Cin => '0',  S => S(0), Cout => C(0));

  FA2: FULL_ADDER port map (A => A(1), B => B(1), Cin => C(0), S => S(1), Cout => C(1));

  FA3: FULL_ADDER port map (A => A(2), B => B(2), Cin => C(1), S => S(2), Cout => C(2));

  FA4: FULL_ADDER port map (A => A(3), B => B(3), Cin => C(2), S => S(3), Cout => Cout);

end Struct;
```

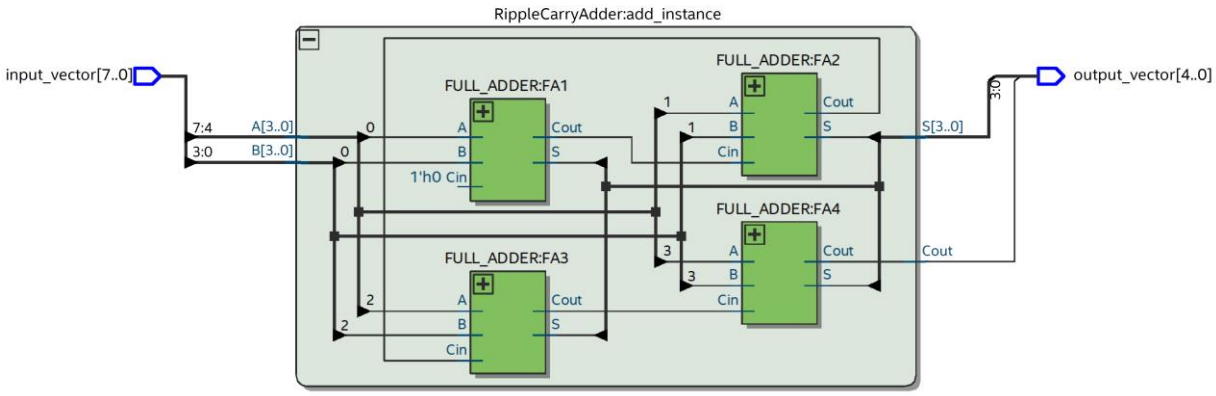
Python code used for generating trace file

```
n = 4
Max = 2**n

numberOfOutputBits = n+1

def generateBinaryStrings(n, Max):
    return [bin(i)[2:].zfill(n) for i in range(Max)]
BinaryStrings = generateBinaryStrings(n, Max)

def addBinaryIntegers(a,b,n, Max):
    c = a + b
    carry = 0
    if c > Max - 1:
        c = c - Max
        carry = 1
    return c, carry
addBinaryIntegers(1,15,n, Max)

with open("Tracefile.txt", "w") as file:
    for i in range(Max):
        for j in range(Max):
            I1 = BinaryStrings[i]
            I2 = BinaryStrings[j]
            output, carry = addBinaryIntegers(i,j,n, Max)
            mask = '1'*numberOfOutputBits
            o = bin(output)[2:].zfill(n)
            str = "{}{} {}{} {}\n".format(I1, I2, carry, o, mask)
            file.write(str)
```

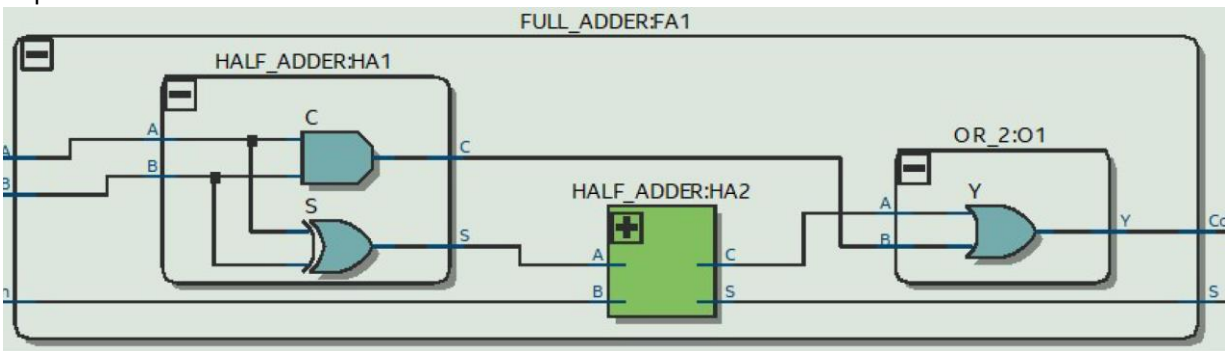
RTL View:



Implementation of Ripple Carry Adder



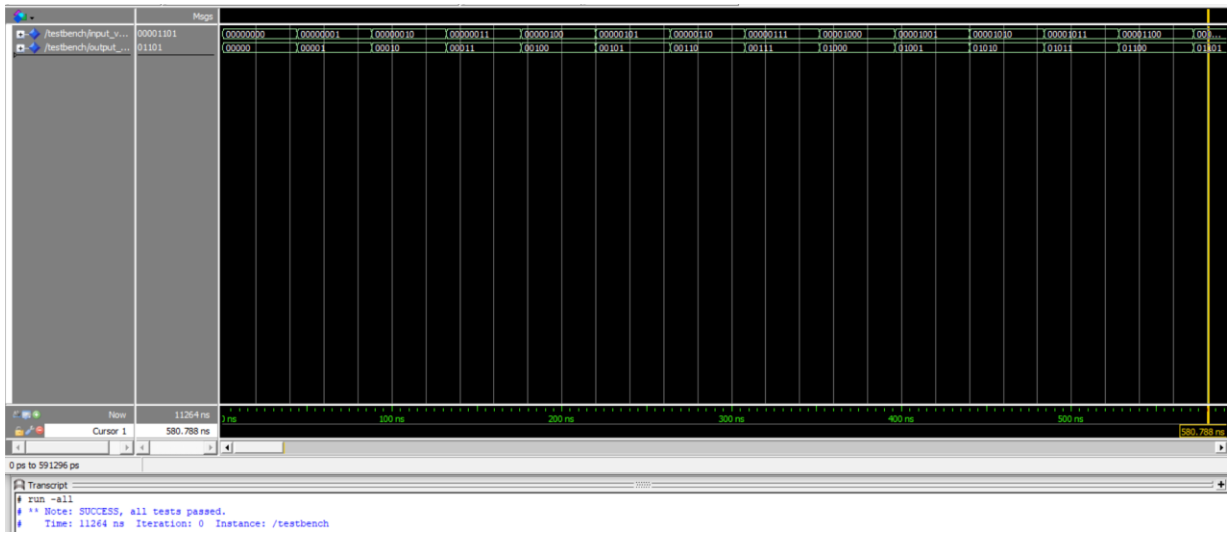
Implementation of Full Adder



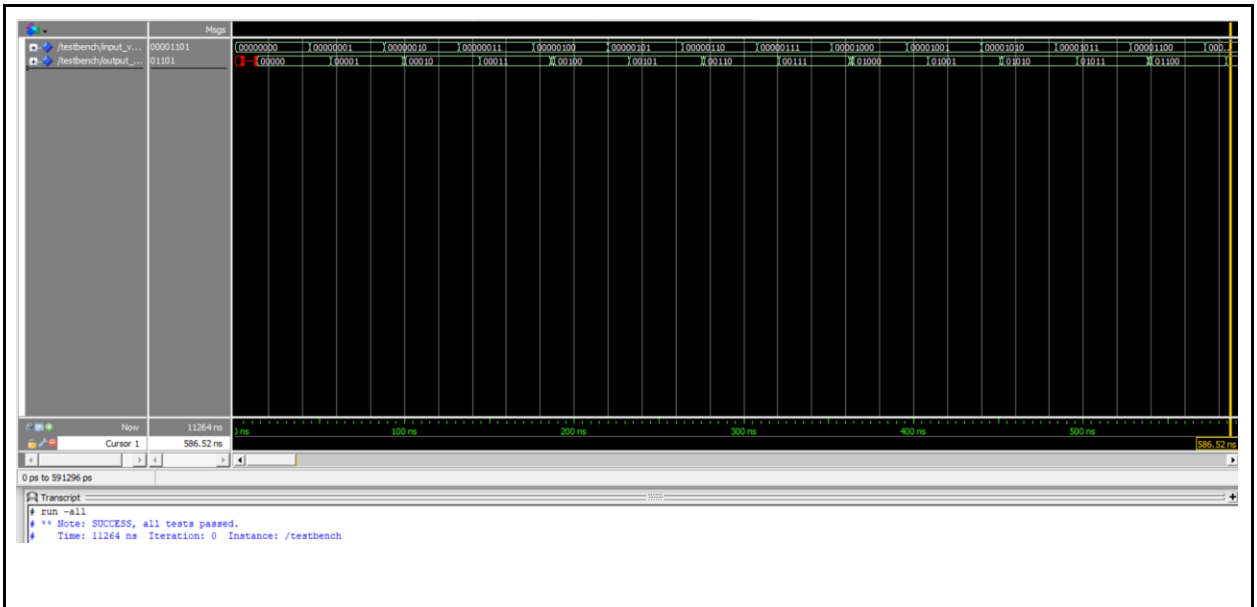
DUT Input/Output Format:

Input Format: $A_3A_2A_1A_0B_3B_2B_1B_0$	(2 4-bit numbers A (A_3 is MSB A_0 is LSB) and B (B_3 is MSB B_0 is LSB)
Output Format: $C_{out}S_3S_2S_1S_0$	(Carry (1-bit) and Sum (4-bit S_3 is MSB S_0 is LSB)
00000000 00000 11111 01111111 10110 11111 10010110 01111 11111 10100110 10000 11111 11111111 11110 11111	

RTL Simulation:



Gate-level Simulation:



Krypton board*:

Map the logic circuit to the Krypton board and attach the images of the pin assignment and output observed on the board (switches/LEDs).

Observations*:

You must summarize your observations, either in words, using figures and/or tables.

References:

Tertulien Ndjountche *Digital Electronics 2 Sequential and Arithmetic Logic Circuits* Wiley
Ripple Carry Adder Graphic
https://upload.wikimedia.org/wikipedia/commons/5/5d/4-bit_ripple_carry_adder.svg

* To be submitted after the tutorial on ”Using Krypton.