

Experiment 4: Combinational Circuit 4

Madan Y N Roll Number 200070040

EE-214, WEL, IIT Bombay

September 21, 2021

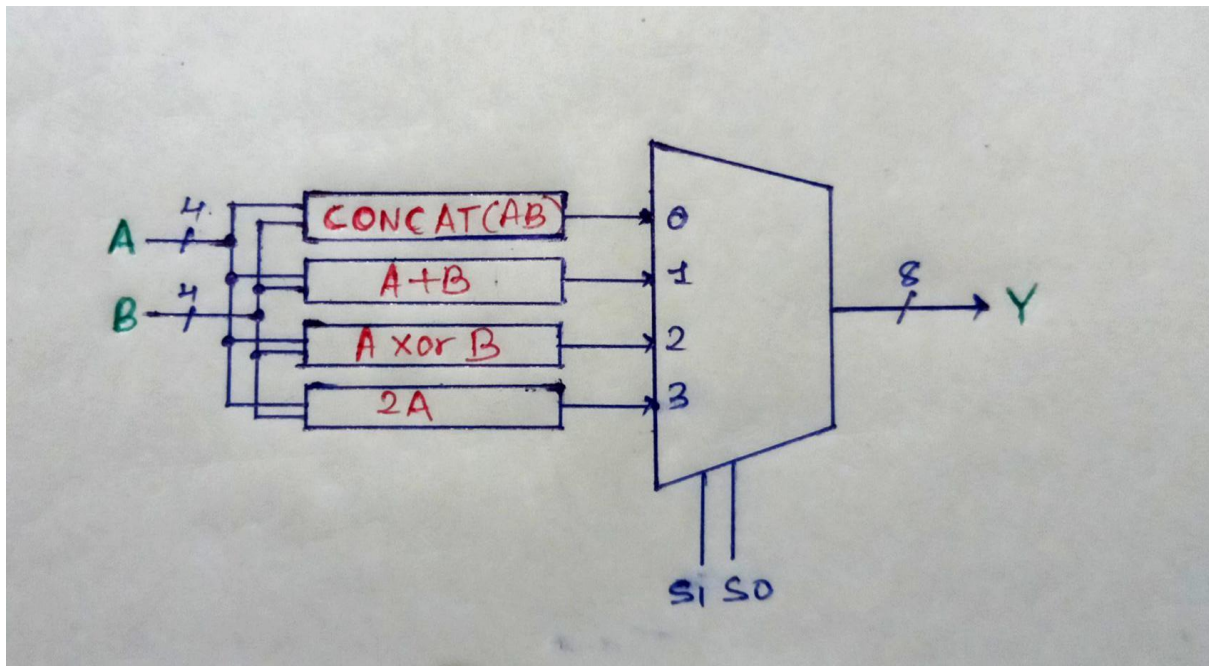
Overview of the experiment:

The purpose of the experiment is to build an ALU(Arithmetic Logic Unit) using behavioral and dataflow modelling in VHDL and to test the correctness of ALU built using scanchain.

In this report I present in a brief how the experiment was conducted and would also present the results of the experiment.

Approach to the experiment:

The ALU performs 4 functions on two 4-bit inputs $A3A2A1A0$ and $B3B2B1B0$ based on two select lines $S1S0$. The block diagram of the ALU is as below



Of the 4 functions to be performed, CONCAT can be done using "&" operator and $A \text{ xor } B$ can be done with bitwise xor operation. For the remaining two, we make use of the function "add".

Sum of a 4 bit number with another 4 bit number is a 5 bit number. The i^{th} bit of sum $S(i)$ is given by

$$S(i) = (A(i) \text{ xor } B(i)) \text{ xor } \text{Carry}(i)$$

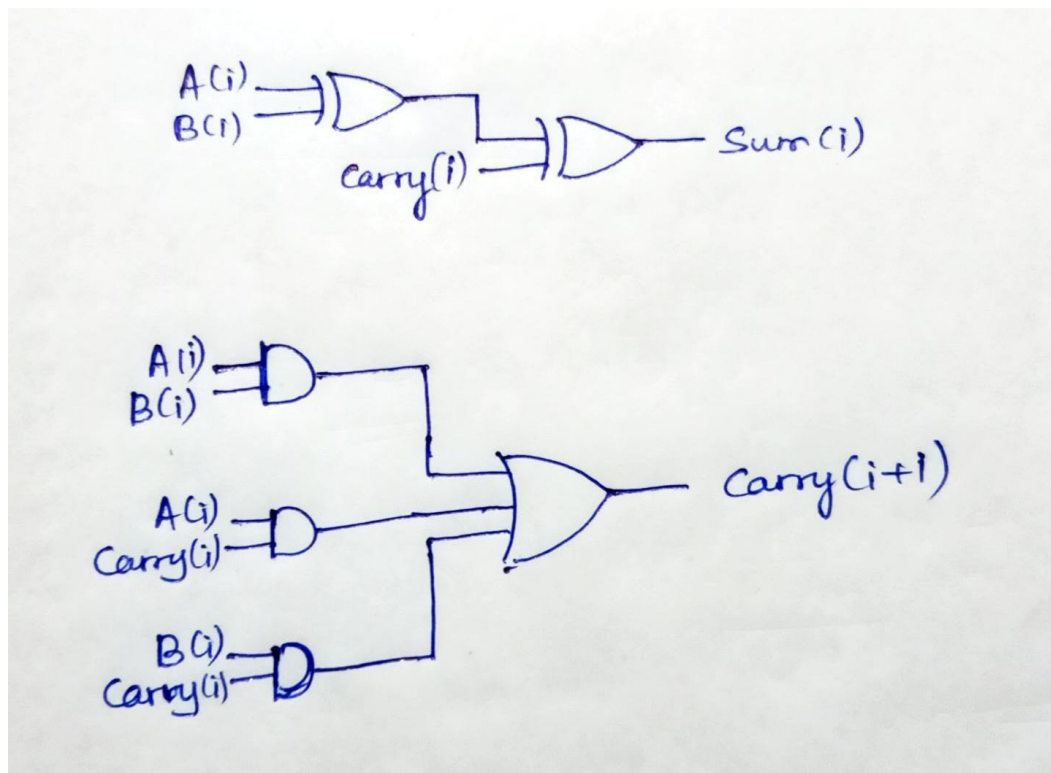
and $\text{Carry}(i)$ is given by

$$\text{Carry}(i + 1) = (A(i) \text{ and } B(i)) \text{ or } (A(i) \text{ and } \text{Carry}(i)) \text{ or } (B(i) \text{ and } \text{Carry}(i))$$

with $\text{Carry}(0) = 0$

Then using a for loop with suitable range for i sum is calculated.

Circuit implementation of above equations is as below



Design document and VHDL code

For the experiment an entity was created for the ALU with suitable inputs and outputs. The function "add" was created inside the architecture "a1" of the entity. Later this function add was used in the process "alu" for implementing ALU.

The architecture of this is as follows

architecture a1 of alu_beh is

```
function add(A: in std_logic_vector(operand_width-1 downto 0); B: in
std_logic_vector(operand_width-1 downto 0))
  return std_logic_vector is
    variable sum:std_logic_vector(4 downto 0) := (others => '0');
    variable carry:std_logic_vector(4 downto 0) := (others => '0');
    -- Declare "sum" and "carry" variable
    -- you can use aggregate to initialize the variables as shown below
    -- variable variable_name : std_logic_vector(3 downto 0) := (others
=> '0');
    begin
      addn: for i in 0 to 3 loop
        sum(i) := (A(i) xor B(i)) xor carry(i);
        carry(i+1) := (A(i) and B(i)) or (B(i)
and carry(i)) or (A(i) and carry(i));
      end loop;
      sum(4) := carry(4);

      -- write logic for addition
      -- Hint: Use for loop
      return sum;
    end add;
```

The process “alu” is as follows

```
begin
alu : process( A, B, sel )
begin

  if (sel(0) = '0' and sel(1) = '0') then
    op <= A&B;
  elsif (sel(1) = '0' and sel(0) = '1') then
    op <= "000"&add(A,B);
  elsif (sel(1) = '1' and sel(0) = '0') then
    op <= "0000"&(A xor B);
  else
    op <= "000"&add(A,A);
```

```

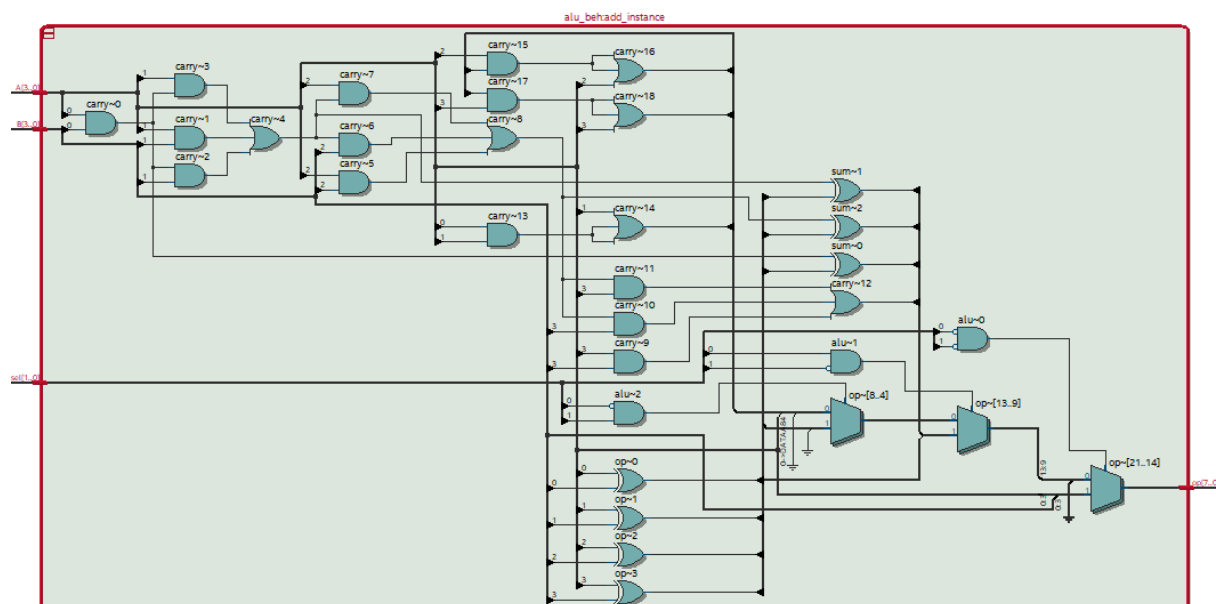
        end if;
    end process ;
end a1 ;

```

In order to test the correctness of the design on hardware we made use of scanchain. This also allowed us to check if our design could actually be implemented on hardware. We generated a svf file which was loaded onto the krypton board, which was later used to run the scanchain test.

RTL View:

RTL view of the ALU



DUT Input/Output Format:

The input vector to the multiplier is of length 10 i.e two 4 bit numbers $A3A2A1A0$ and $B3B2B1B0$ and 2 bit select line $S1S0$. $S1$ is the msb of the input and $B0$ is the lsb.

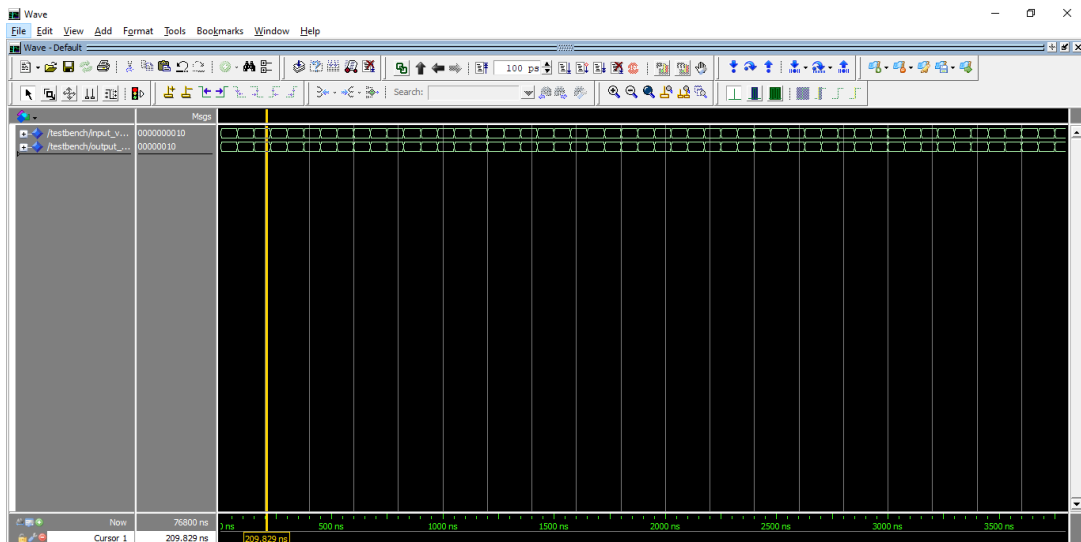
Output of the multiplier is a 8 bit number $Y7Y6Y5Y4Y3Y2Y1Y0$ as it is evident $Y7$ is the msb and $Y0$ is the lsb.

Some test cases from the tracefile are

Input	Output
0100011000	00001001
0100011001	00001010
0100011010	00001011
0100011011	00001100
0100011100	00001101
0100011101	00001110

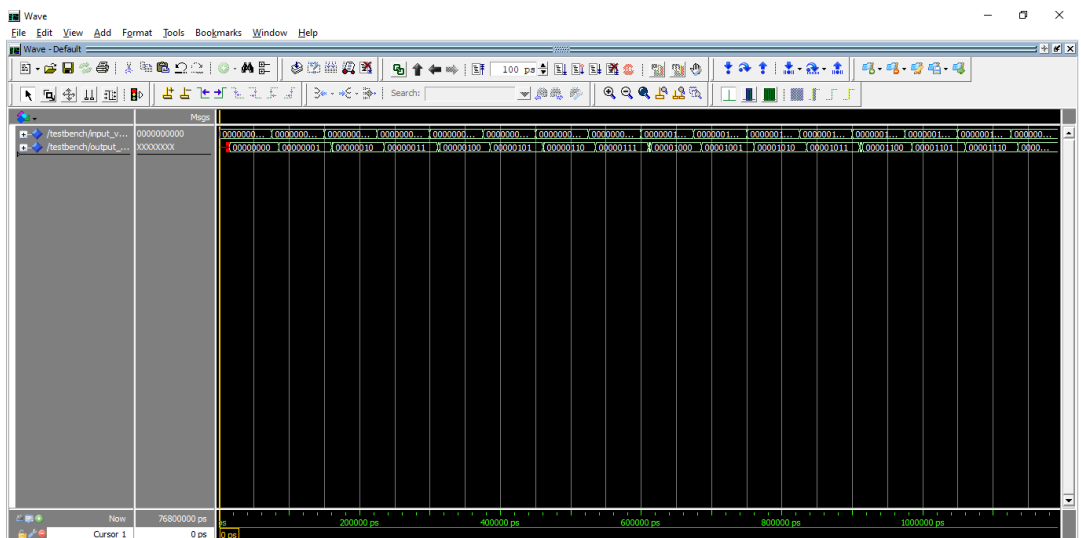
RTL Simulation:

RTL simulation of ALU



Gate-level Simulation:

Gate level simulation of Multiplier



Krypton Board

The pin planning of the krypton is as below

The screenshot shows the Pin Planner tool interface. The top view diagram displays the Krypton board layout with pins numbered 1 to 130. The pin legend on the right lists various pin types: User I/O, User assigned I/O, Fitter assigned I/O, Unbonded pad, Reserved pin, DEV_OE, DEV_CLR, and DIFF_n output. The table below lists the pin assignments for each node.

Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair	Strict Preservation
input_vector[9]	Input			PIN_97	3.3-V L...efault		16mA (default)		
input_vector[8]	Input			PIN_124	3.3-V L...efault		16mA (default)		
input_vector[7]	Input			PIN_66	3.3-V L...efault		16mA (default)		
input_vector[6]	Input			PIN_59	3.3-V L...efault		16mA (default)		
input_vector[5]	Input			PIN_52	3.3-V L...efault		16mA (default)		
input_vector[4]	Input			PIN_131	3.3-V L...efault		16mA (default)		
input_vector[3]	Input			PIN_88	3.3-V L...efault		16mA (default)		
input_vector[2]	Input			PIN_58	3.3-V L...efault		16mA (default)		
input_vector[1]	Input			PIN_125	3.3-V L...efault		16mA (default)		
input_vector[0]	Input			PIN_48	3.3-V L...efault		16mA (default)		
output_vector[7]	Output			PIN_61	3.3-V L...efault		16mA (default)		
output_vector[6]	Output			PIN_63	3.3-V L...efault		16mA (default)		
output_vector[5]	Output			PIN_57	3.3-V L...efault		16mA (default)		
output_vector[4]	Output			PIN_129	3.3-V L...efault		16mA (default)		
output_vector[3]	Output			PIN_62	3.3-V L...efault		16mA (default)		
output_vector[2]	Output			PIN_55	3.3-V L...efault		16mA (default)		
output_vector[1]	Output			PIN_130	3.3-V L...efault		16mA (default)		
output_vector[0]	Output			PIN_53	3.3-V L...efault		16mA (default)		

Observation:

Upon testing using scanchain we got an output file *out.txt* which contained the result of the experiment. In it we get to see if the design made by us is correct and if the design could be implemented on hardware. All the test cases passed in scanchain which assures the correctness of the design. Another observation is that Behaviour modelling is simpler than Structural modelling which was being used in the previous labs. Behaviour modelling gives us more flexibility and freedom to create projects in vhd.