Experiment 1: ALU

A submission Report

Hardik Panchal

Roll Number 200070054

EE-214, WEL, IIT Bombay

September 27, 2021

## Overview of the experiment:

|  |
| --- |
| In the experiment we have to describe an ALU using behavioral modelling. The ALU has to perform 4 different operations based on the input S1 and S0.  First function was to concatenate two 4-bit inputs and make one 8-bit output.  Second function was to add two 4-bit inputs and make one 8-bit output.  Third function was to do a bitwise XOR operation of two 4-bit inputs and report 8-bit output by concatenating extra zeros in front of output.  Last function was to output the double of 4-bit input as 8-bit output.  I will be presenting here the design and code for this and also screenshots of successful RTL and Gate Level simulations. |

## Approach to the experiment:

|  |
| --- |
| I firstly described the addition function in behavioral style and used that for accomplishing first and last function.  I used the in built xor operator to do bitwise xor of two inputs to implement third function.  I used the in built & operator to concatenate two inputs to implement fourth function and also, I used this operator to make my other 4-bit output into 8-bit output via concatenating extra zeros in front of them. |

## 

## Design document and VHDL code if relevant:

|  |
| --- |
| **Design:**    **Code of an architecture:**  architecture a1 of alu\_beh is  function add(A: in std\_logic\_vector((operand\_width\*2)-1 downto 0); B: in std\_logic\_vector((operand\_width\*2)-1 downto 0))  return std\_logic\_vector is    variable sum : std\_logic\_vector(7 downto 0) := (others => '0');  variable carry : std\_logic\_vector(8 downto 0) := (others => '0');      begin    carry(0) := '0';  for i in 0 to 7 loop  sum(i) := A(i) xor B(i) xor carry(i);  carry(i+1) := (A(i) and B(i)) or (carry(i) and (A(i) xor B(i)));  end loop;  return sum;  end add;    begin  alu : process( A, B, sel )  variable answer1 : std\_logic\_vector(7 downto 0);    variable answer2 : std\_logic\_vector(7 downto 0);  variable Anew : std\_logic\_vector(7 downto 0);  variable Bnew : std\_logic\_vector(7 downto 0);    variable answer3 : std\_logic\_vector(7 downto 0);  variable answer3\_0 : std\_logic\_vector(3 downto 0);    variable answer4 : std\_logic\_vector(7 downto 0);  variable answer4\_0 : std\_logic\_vector(7 downto 0);    begin    if (sel = "00") then    answer1 := A&B;  op <= answer1;    elsif (sel = "01") then    Anew := "0000" & A;  Bnew := "0000" & B;    answer2 := add(Anew,Bnew);  op <= answer2;    elsif (sel = "10") then    answer3\_0 := A xor B;  answer3 := "0000" & answer3\_0;  op <= answer3;    else    answer4\_0 := "0000" & A;  answer4 := add(answer4\_0, answer4\_0);    op <= answer4;      end if;  end process ; -- alu  end a1 ; -- a1 |

## RTL View:

|  |
| --- |
|  |

## DUT Input/Output Format:

|  |
| --- |
| **Input**: S1 S0 A3 A2 A1 A0 B3 B2 B1 B0 **LSB** = B0, **MSB** = S1  **Output**: Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0 **LSB** = Y0, **MSB** = Y7  **Some Test Cases from TRACEFILE.txt**  **Format**: S1 S0 A3 A2 A1 A0 B3 B2 B1 B0 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0  0101010110 00001011 11111111  0101010111 00001100 11111111  0101011000 00001101 11111111  0101011001 00001110 11111111  0101011010 00001111 11111111  0101011011 00010000 11111111  0101011100 00010001 11111111  0101011101 00010010 11111111  0101011110 00010011 11111111  0101011111 00010100 11111111  0101100000 00000110 11111111  0101100001 00000111 11111111  0101100010 00001000 11111111  0101100011 00001001 11111111  0101100100 00001010 11111111  0101100101 00001011 11111111  0101100110 00001100 11111111  0101100111 00001101 11111111 |

## RTL Simulation:

|  |
| --- |
|  |

## Gate-level Simulation:

|  |
| --- |
|  |

## 

## Krypton board:

|  |
| --- |
| We have used scanchain for this experiment. So out.txt has output which I got using scanchain.      **Some outputs from out.txt**  0111011011 00011000 Success  0111011100 00011001 Success  0111011101 00011010 Success  0111011110 00011011 Success  0111011111 00011100 Success  0111100000 00001110 Success  0111100001 00001111 Success  0111100010 00010000 Success  0111100011 00010001 Success  0111100100 00010010 Success  0111100101 00010011 Success  0111100110 00010100 Success  0111100111 00010101 Success  0111101000 00010110 Success  0111101001 00010111 Success  0111101010 00011000 Success  0111101011 00011001 Success  0111101100 00011010 Success  0111101101 00011011 Success  0111101110 00011100 Success |

## Observations:

|  |
| --- |
| The main observation and learning from this experiment were to how to implement the functions using behavioral style.  We can see that if S1=’0’ and S2=’0’ then our design will concatenate two 4-bit inputs A and B and report it as 8-bit output.  if S1=’0’ and S2=’1’ then our design will add two 4-bit inputs A and B and report it as 8-bit output.  if S1=’1’ and S2=’0’ then our design will do bitwise xor of two 4-bit inputs A and B and report it as 8-bit output.  if S1=’1’ and S2=’1’ then our design will add two 4-bit inputs A and A and report it as 8-bit output 2A. |

## References:

|  |
| --- |
| My main reference was our course webpage it contained many useful things such as a sample code and many other specifications. This I used as my reference. |