A PROJECT REPORT

ON

“HARDWARE DESIGN OF 3 BIT LOGICAL UNIT”

A Comprehensive Study of Digital Circuits

By :- Kushal Jain

INDEX

I. INTRODUCTION

A logical unit is a fundamental component in digital systems, performing basic logic operations such as AND, OR, XOR, and NOT. It plays a crucial role in arithmetic and logical operations in computing systems. Logical units are used in microprocessors, embedded systems, and digital circuits to process binary data efficiently. In this project, we design a 3-bit Logical Unit using Verilog in Xilinx Vivado, verify the design in Simulink, implement the circuit using Tinker CAD, generate Opcodes, and perform testing using Arduino IDE with ATPG. The implementation ensures that the logical unit can handle multiple operations with high accuracy and efficiency, making it suitable for real-time applications.

The logical unit performs the following operations based on the 3-bit Opcode:
000: CLEAR
001: MOV out, A
002: MOV out, B
011: AND out, A, B
100: OR out, A, B
101: XOR out, A, B
110: NOT out,
111: NOT out, B

These operations allow the logical unit to execute essential Boolean functions necessary for digital computation. The design ensures optimized performance, minimal hardware complexity, and ease of implementation in various digital systems.

II. DESIGN AND METHODOLOGY

- **Xilinx Vivado: Verilog Implementation & Simulation**

  The 3-bit Logical Unit is implemented using Verilog, supporting fundamental logic operations. The functionality is verified through simulation in Xilinx Vivado using a testbench.

- **Verilog module :**

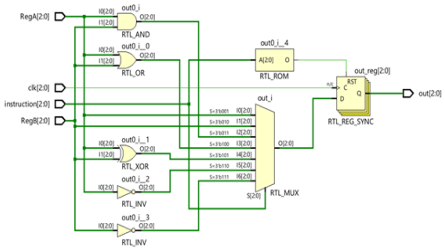
```
module arithmatic(
                    input [2:0] RegA ,
                    [2:0] RegB ,
                     clk ,
                     [2:0] instruction ,
                     output reg [2:0] out);

    always @(posedge clk) begin
       casez (instruction)
          3'b000 : out = 0 ;
          3'b001 : out = RegA;
           3'b010 : out = RegB;
           3'b011 : out = RegA & RegB;
           3'b100 : out = RegA | RegB;
           3'b101 : out = RegA ^ RegB;
           3'b110 : out = ~RegA;
           3'b111 : out = ~RegB;
       endcase
    end
endmodule
```

- **Verilog test bench:**

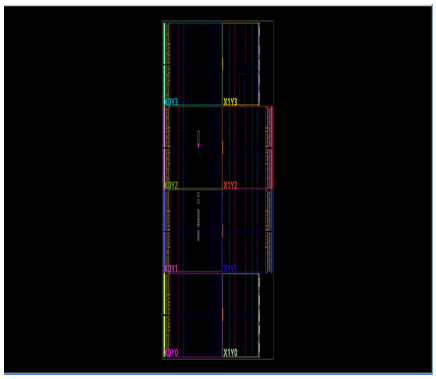
```
module arithmatic_tb();
    reg [2:0] RegA , RegB;
    reg [2:0] instruction;

reg clk = 0 ;

wire [2:0] out;

    arithmatic inst1 (.RegA(RegA) , .RegB(RegB) , .clk(clk) , .instruction(instruction) ,
.out(out));

   always begin
      #5 clk = ~clk;
```

```
    end

    initial begin
        $monitor("Time = %b , RegA = %b , RegB = %b , Instruction = %b , output = %b" ,
$time , RegA , RegB , instruction , out);
        RegA = 2 ; RegB = 5;
        #10 instruction = 0;
        #10 instruction = 1;
        #10 instruction = 2;
        #10 instruction = 3;
        #10 instruction = 4;
        #10 instruction = 5;
        #10 instruction = 6;
        #10 instruction = 7;
        #10 $finish();
    End
Endmodule
```

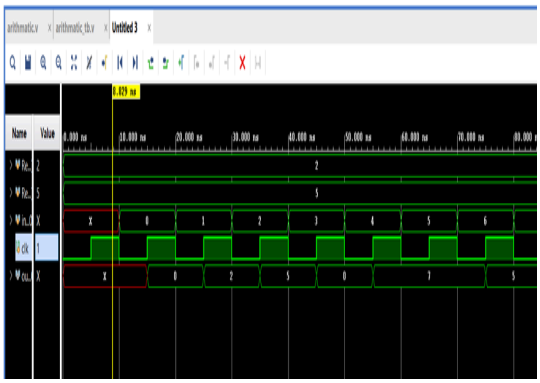
RTL design :



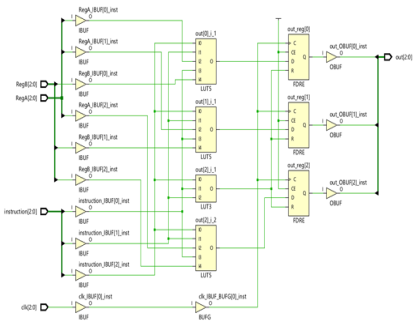
Synthesis :



Simulation Waveform :



implementation Schematic :



III. TPG and testing :

Arduino is a versatile microcontroller platform that can be programmed to generate multiple test patterns for various applications. In this project, Arduino is used to create different test signals that can be applied to a system for validation and performance analysis. By utilizing its digital and analog output pins, along with pulse-width modulation (PWM) and serial communication features, Arduino can generate waveforms such as square waves, sine waves (using a DAC or PWM filtering), and custom logic sequences. The flexibility of Arduino's programming environment allows for easy modification of test patterns based on project requirements. Additionally, using libraries such as Tone() for frequency generation and Bit-banging techniques for custom sequences, Arduino can effectively simulate different input conditions for testing purposes. This capability is particularly useful for debugging circuits, verifying sensor responses, and evaluating the behavior of digital and analog systems in real-time.

- **Testing code for arduino :**

Arduino code :

```
int byte1bit0 = 2;
int byte1bit1 = 3;
int byte1bit2 = 4;
int byte2bit0 = 5;
int byte2bit1 = 6;
int byte2bit2 = 7;
int instructionBit0 = 8;
int instructionBit1 = 9;
int instructionBit2 = 10;

void setup()
{
  pinMode(byte1bit0, OUTPUT);
  pinMode(byte1bit1, OUTPUT);
  pinMode(byte1bit2, OUTPUT);
  pinMode(byte2bit0, OUTPUT);
  pinMode(byte2bit1, OUTPUT);
  pinMode(byte2bit2, OUTPUT);
  pinMode(instructionBit0,
OUTPUT);
  pinMode(instructionBit1,
OUTPUT);
  pinMode(instructionBit2,
OUTPUT);
}
void loop()
```

```
{
   digitalWrite(byte1bit0,HIGH);
   digitalWrite(byte1bit1,LOW);
   digitalWrite(byte1bit2,HIGH);

   digitalWrite(byte2bit0,LOW);
   digitalWrite(byte2bit1,HIGH);
   digitalWrite(byte2bit2,LOW);

   digitalWrite(instructionBit0,LOW);
   digitalWrite(instructionBit1,LOW);
   digitalWrite(instructionBit2,LOW);
   delay(1000*1.25);


digitalWrite(instructionBit0,HIGH);
   digitalWrite(instructionBit1,LOW);
   digitalWrite(instructionBit2,LOW);
   delay(1000*1.25);

   digitalWrite(instructionBit0,LOW);

digitalWrite(instructionBit1,HIGH);
   digitalWrite(instructionBit2,LOW);
   delay(1000*1.25);



digitalWrite(instructionBit0,HIGH);

digitalWrite(instructionBit1,HIGH);
   digitalWrite(instructionBit2,LOW);
   delay(1000*1.25);

   digitalWrite(instructionBit0,LOW);
   digitalWrite(instructionBit1,LOW);

digitalWrite(instructionBit2,HIGH);
   delay(1000*1.25);


digitalWrite(instructionBit0,HIGH);
digitalWrite(instructionBit1,LOW);
digitalWrite(instructionBit2,HIGH);
delay(1000*1.25);

digitalWrite(instructionBit0,LOW);
digitalWrite(instructionBit1,HIGH);
digitalWrite(instructionBit2,HIGH);
delay(1000*1.25);

digitalWrite(instructionBit0,HIGH);
digitalWrite(instructionBit1,HIGH);
digitalWrite(instructionBit2,HIGH);
delay(1000*1.25);

digitalWrite(byte1bit0,LOW);
digitalWrite(byte1bit1,HIGH);
digitalWrite(byte1bit2,LOW);

digitalWrite(byte2bit0,HIGH);
digitalWrite(byte2bit1,LOW);
digitalWrite(byte2bit2,HIGH);

digitalWrite(instructionBit0,LOW);
digitalWrite(instructionBit1,LOW);
digitalWrite(instructionBit2,LOW);
delay(1000*1);
```

```
digitalWrite(instructionBit0,HIGH);
digitalWrite(instructionBit1,LOW);
digitalWrite(instructionBit2,LOW);
delay(1000*1);

digitalWrite(instructionBit0,LOW);
digitalWrite(instructionBit1,HIGH);
digitalWrite(instructionBit2,LOW);
delay(1000*1);

digitalWrite(instructionBit0,HIGH);
digitalWrite(instructionBit1,HIGH);
digitalWrite(instructionBit2,LOW);
delay(1000*1);

digitalWrite(instructionBit0,LOW);
digitalWrite(instructionBit1,LOW);
digitalWrite(instructionBit2,HIGH);
delay(1000*1);

digitalWrite(instructionBit0,HIGH);
digitalWrite(instructionBit1,LOW);
digitalWrite(instructionBit2,HIGH);
delay(1000*1);

digitalWrite(instructionBit0,LOW);
digitalWrite(instructionBit1,HIGH);
digitalWrite(instructionBit2,HIGH);
delay(1000*1);

digitalWrite(instructionBit0,HIGH);
digitalWrite(instructionBit1,HIGH);
digitalWrite(instructionBit2,HIGH);
delay(1000*1);
}
```

IV. SUMMARY AND FUTURE PLANS:

This project presents an innovative approach to implementing fundamental logical operations on two 3-bit data inputs using an 8-to-1 multiplexer (MUX). The core concept revolves around using the MUX as an arithmetic logic unit (ALU) that performs operations such as AND, OR, XOR, NAND, NOR, XNOR, and basic arithmetic manipulations based on a predefined operation code (OpCode) fed through the MUX's select lines. An Arduino acts as an intelligent input pattern generator, dynamically feeding 3-bit inputs and the corresponding OpCode to dictate which logical operation the MUX should execute. The design is meticulously crafted using Xilinx FPGA tools, where the RTL schematic is generated to visualize and optimize the hardware architecture, ensuring high efficiency and minimal resource utilization. This setup not only showcases the versatility of multiplexers but also demonstrates the potential of low-power, hardware-optimized logic processing.

To push the boundaries of this project, a three-stage pipelining mechanism will be incorporated to improve the efficiency and speed of operation execution. The pipelining approach will break down the processing into distinct stages—input buffering, operation execution, and output stabilization—ensuring seamless data flow and reduced latency. Additionally, to eliminate human intervention in testing, an automatic test pattern generator (TPG) will be developed. This will autonomously generate a diverse range of input test cases, systematically verifying the accuracy and robustness of the MUX-based logic processor. By eliminating the dependency on Arduino for manual input feeding, the system will evolve into a fully automated testing and validation environment, paving the way for scalability in more complex logical and arithmetic processing applications.