# Project Phase 1

## Arithmetic Logic Unit

MADE BY

| NAMES | ID |
| --- | --- |
| Youssef Hassan Abdelmaksoud | 211002056 |


## Description

We created an Arithmetic Logic Unit (ALU) which is a digital circuit that performs arithmetic and logical operations on two inputs. Our ALU has four basic operations: AND, OR, addition, and subtraction. We also separated the adder and subtractor operations into separate files for modularity and ease of use.

# Adder.sv File

```
1  module adder(
2    input [N-1:0] a,
3    input [N-1:0] b,
4    input cin,
5    output wire [N-1:0] sum,
6    output wire  cout
7  );
8    parameter N=32;
9    wire [N:0] temp;
10   assign sum=a+b+cin;
11   assign temp={1'b0,a}+{1'b0,b};
12   assign cout=temp[N];
13
14
15 endmodule
```

# Subtractor.sv File

```
1  module subtractor(
2    input [N-1:0] a,
3    input [N-1:0] b,
4    input bin,
5    output reg [N-1:0] diff,
6    output reg bout
7  );
8    parameter N=32;
9    wire [N-1:0] neg_a;
10   wire [N-1:0] neg_b;
11   assign neg_a = ~a + 1;
12   assign neg_b = ~b + 1;
13
14   reg [N:0] temp;
15
16   always @* begin
17       temp = {1'b0, a} + {1'b1, neg_b} + bin;
18       diff = temp[N-1:0];
19       bout = temp[N];
20   end
21
22 endmodule
```

# Design.sv File(ALU module)

```
//Bonus Part
`include "adder.sv"
`include "subtractor.sv"


// ALU module
module ALU(
  input [N-1:0] Op1,
  input [N-1:0] Op2,
  input [3:0] OpCode,
  input cin,
  output reg [N-1:0] Result,
  output reg zFlag,
  output reg oFlag,
  output reg cFlag
);
  parameter N=32;
  wire [N-1:0] adder_out;
  wire [N-1:0] subtractor_out;
  wire cFlag_adder;
  wire cFlag_subtractor;


    // Perform addition operation
  adder adder_inst(Op1, Op2, cin, adder_out, cFlag_adder);

    // Perform subtraction operation
    subtractor subtractor_inst(Op1, Op2, cin, subtractor_out, cFlag_subtractor);

always @ (*) begin
  case (OpCode)
    // And operation
    4'b0000:
    begin
      Result = Op1 & Op2;
      cFlag = 0;
    end
    // OR operation

    4'b1111 :
    begin
      Result = Op1 | Op2;
      cFlag = 0;
    end

    // Adder operation
    4'b1001:
    begin
      Result = adder_out;
      cFlag=cFlag_adder;
    end

    // Subtraction operation
    4'b0110:
    begin
      Result = subtractor_out;
      cFlag=cFlag_subtractor;
    end

    default: Result =32'b0;
  endcase


  // Set the zero flag if the result is zero
  zFlag = (Result == 32'b0);

  // Set the overflow flag if the result overflows
  if (OpCode == 4'b1001 || OpCode == 4'b0110)
    begin
    oFlag = ((Op1[31] == Op2[31]) && (Result[31] != Op1[31]));
    end
  else
    begin
    oFlag = 0;
    end
end

endmodule
```

# Testbench.sv File

```
1  module test();
2    reg [31:0] op1=32'b00110011001100110011001100110011;
3    reg [31:0] op2=32'b11001100110011001100110011001100 ;
4    reg [3:0] opcode=4'b1111;
5    reg cin=1'b1;
6    wire cflag;
7
8
9    wire [31:0] result;
10   wire  zflag;
11   wire oflag;
12
13   ALU my_alu(op1,op2,opcode,cin,result,zflag,oflag,cflag);
14
15   initial begin
16     $monitor("The Result is : %b",result);
17     #10
18     $monitor("The carry flag is: %b",cflag);
19     #10
20     $monitor("The Zero Flag is %b",zflag);
21     #10
22     $monitor("The Overflow Flag is %b",oflag);
23   end
24 endmodule
25
```

# TEST CASES

## Test case 1

### INPUTS

```
reg [31:0] op1=32'b00110011001100110011001100110011;
reg [31:0] op2=32'b11001100110011001100110011001100;
reg [3:0] opcode=4'b1111;
reg cin=1'b1;
```

### OUTPUTS

```
The Result is : 11111111111111111111111111111111
The carry flag is: 0
The Zero Flag is 0
The Overflow Flag is 0
```

---

## Test case 2

### INPUTS

```
reg [31:0] op1=32'b00110011001100110011001100110011;
reg [31:0] op2=32'b11001100110011001100110011001100;
reg [3:0] opcode=4'b0000;
reg cin=1'b0;
```

### OUTPUTS

```
The Result is : 00000000000000000000000000000000
The carry flag is: 0
The Zero Flag is 1
The Overflow Flag is 0
```

## Test case 3

### INPUTS

```
reg [31:0] op1=32'b00000000000000001111111111111111;
reg [31:0] op2=32'b00000000000000000000000000000001;
reg [3:0] opcode=4'b1001;
reg cin=1'b1;
```

### OUTPUTS

```
The Result is : 00000000000000010000000000000001
The carry flag is: 0
The Zero Flag is 0
The Overflow Flag is 0
Done
```

## Test case 4

### INPUTS

```
reg [31:0] op1=32'b00000000000000000000000000001000;
reg [31:0] op2=32'b00000000000000000000000000000001;
reg [3:0] opcode=4'b0110;
reg cin=1'b0;
```

### OUTPUTS

```
The Result is : 00000000000000000000000000000111
The carry flag is: 0
The Zero Flag is 0
The Overflow Flag is 0
Done
```

## Test case 5

### INPUTS

```
reg [31:0] op1=32'b11110000111100001111000011110001;
reg [31:0] op2=32'b00001111000011110000111100001111;
reg [3:0] opcode=4'b1001;
reg cin=1'b0;
```

### OUTPUTS

```
The Result is : 00000000000000000000000000000000
The carry flag is: 1
The Zero Flag is 1
The Overflow Flag is 0
Done
```

## Test case 6

### INPUTS

```
reg [31:0] op1=32'b00110011001100110011001100110011;
reg [31:0] op2=32'b01001100110011001100110011001100;
reg [3:0] opcode=4'b0110;
reg cin=1'b1;
```

### OUTPUTS

```
The Result is : 11100110011001100110011001100110
The carry flag is: 1
The Zero Flag is 0
The Overflow Flag is 1
Done
```

## EDA Playground Project Link

https://www.edaplayground.com/x/wzEA