PROYEK 2

PROGRAMMABLE ALU 8 BIT

PERANCANGAN KOMPONEN TERPROGRAM

Disusun oleh:

Nama	:	Yohanes Stefanus
NRP	:	5022211089
Github repository	:	https://github.com/yohanesstef/ALU_8-Bit_PKT.git

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

TEKNIK ELEKTRO

2023/2024

Spesifikasi:

Membuat sebuah ALU yang terdapat 4 operasi, yaitu NAND, OR, NOR, dan Full adder. Dengan input 8 bit, dan carry.

Tabel Kebenaran NAND, OR, dan NOR, dengan input 2 Bit sebagai contoh:

input				NAND		OR		XOR	
a1	b1	a0	b0	f1	f0	f1	f0	f1	f0
0	0	0	0	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
0	0	0	1	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
0	0	1	0	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
0	0	1	1	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
0	1	0	0	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
0	1	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
0	1	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
0	1	1	1	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
1	0	0	0	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
1	0	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
1	0	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
1	0	1	1	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
1	1	0	0	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE
1	1	0	1	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
1	1	1	0	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
1	1	1	1	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE

Tabel Kebenaran Full adder, dengan input 2 Bit sebagai contoh:

input					Full adder		
a1	b1	a0	b0	carry_in	carry_out	f1	f0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	1	0
0	1	0	0	1	0	1	1
0	1	0	1	0	0	1	1
0	1	0	1	1	1	0	0
0	1	1	0	0	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	0	1	0	0
0	1	1	1	1	1	0	1

1	0	0	0	0	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	0	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	0	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	0	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	0	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

Program ALU 8 Bit:

Berikut adalah program verilog ALU 8-Bit:

```
`timescale 1ns / 1ps
module ALU_8_Bit(
        input [7:0] a,b,
        input [1:0] alu_cont,
        input carry_in,
        output reg [7:0] alu_out,
        output reg carry_out
    );
    reg [8:0] over;

    always@(*) begin
        case(alu_cont)
            2'b00: alu_out = ~(a & b);
            2'b01: alu_out = a | b;
            2'b10: alu_out = a ^ b;
            2'b11: alu_out = a + b + carry_in;
            default: alu_out = ~(a & b);
        endcase
        over = {1'b0,a} + {1'b0,b};
        carry_out = over[8];
    end
endmodule
```

Pembahasan Program:

- Inisiasi Input dan Output

```
module ALU_8_Bit(
        input [7:0] a,b,
        input [1:0] alu_cont,
        input carry_in,
        output reg [7:0] alu_out,
        output reg carry_out
    );
```

Input a, b merupakan bilangan 8 bit, alu_cont 2 bit, carry_in 1 bit, output alu_out 8 bit, dan 1 bit untuk carry_out.

- Inisiasi register untuk menyimpan nilai overflow

```
    reg [8:0] over;
```

register ini untuk menyimpan nilai perhitungan dan carry_out dari ALU.

- Switch case
  - Operasi logika

```
  always@(*) begin
        case(alu_cont)
            2'b00: alu_out = ~(a & b);
            2'b01: alu_out = a | b;
            2'b10: alu_out = a ^ b;
```

Program di atas merupakan operasi logika NAND, OR, dan XOR;

  - Full adder dan mengambil nilai carry out

```
            2'b11: alu_out = a + b + carry_in;
            default: alu_out = ~(a & b);
        endcase
        over = {1'b0,a} + {1'b0,b};
        carry_out = over[8];
```

Niali carry_out akan diambil dari MSB register over.

Program Test Bench:

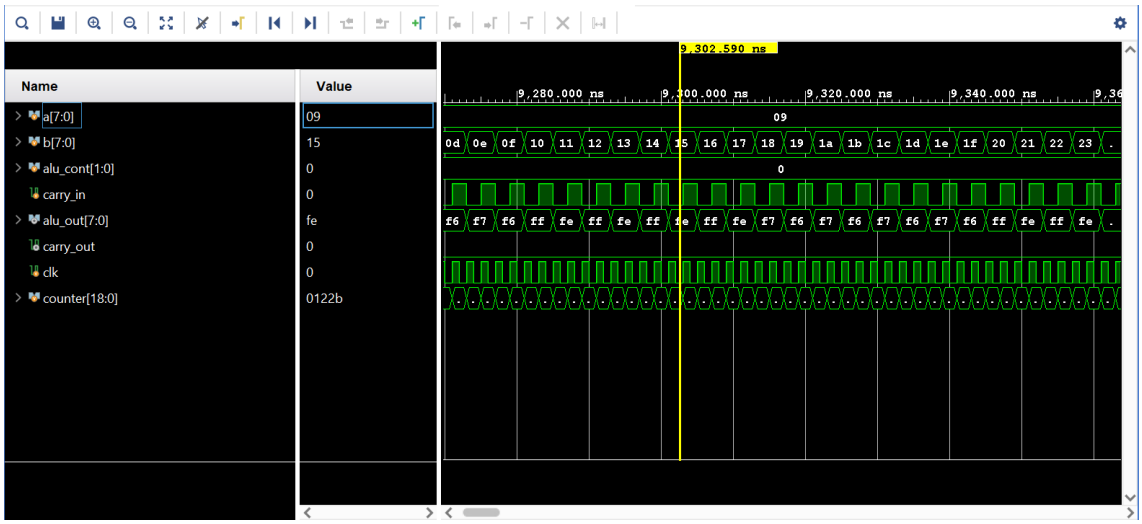
Berikut adalah program test bench nya:

```
`timescale 1ns / 1ps
module ALU_8_Bit_tb;
    // Inputs
    reg [7:0] a;
    reg [7:0] b;
    reg [1:0] alu_cont;
    reg carry_in;
```

```
    // Outputs
    wire [7:0] alu_out;
    wire carry_out;

    //clock untuk test bench
    reg clk;
    reg [18:0] counter; //[0] carry_in, [8:1]b, [16:9]a, [18:17] alu_cont

    // Instantiate the Unit Under Test (UUT)
    ALU_8_Bit uut (
        .a(a),
        .b(b),
        .alu_cont(alu_cont),
        .carry_in(carry_in),
        .alu_out(alu_out),
        .carry_out(carry_out)
    );

    initial begin
        // Initialize Inputs
        a = 8'd0;
        b = 8'd0;
        alu_cont = 2'b00;
        carry_in = 1'b0;
        clk = 0;
        counter = 19'd0;
    end

    always #1 clk = ~clk;

    always @(posedge clk)begin
        counter <= counter + 1;
        carry_in <= counter[0];
        b <= counter[8:1];
        a <= counter[16:9];
        alu_cont <= counter[18:17];
    end
endmodule
```

Hasil Test Bench:

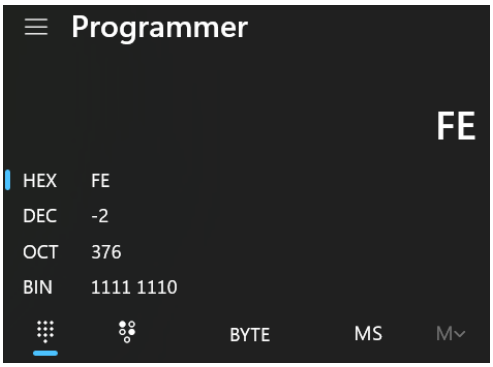
Sinyal Output:



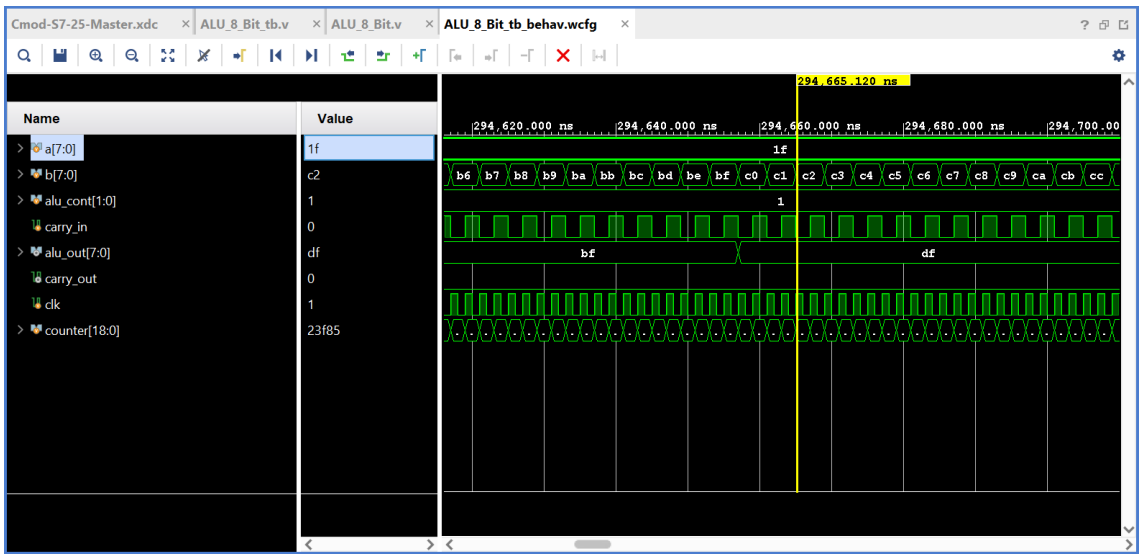
Tabel kebenaran:

Operasi:	alu_cont = 0 (Operasi NAND)					
Jenis:	Input ALU			Alu selector	Output ALU	
Bilangan:	A	B	carry_in	alu_cont	carry_out	alu_out
Size:	8 Bit	8 Bit	1 Bit	2 Bit	1 Bit	8 Bit
Hexadecimal:	9	15	0	0	0	FE
Biner:	0000 1001	0001 0101	0	00	0	1111 1110

Hasil Calculator:



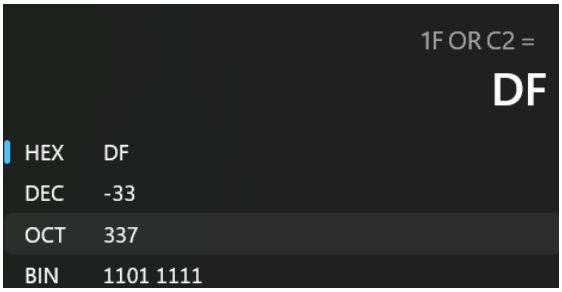
Output sinyal:



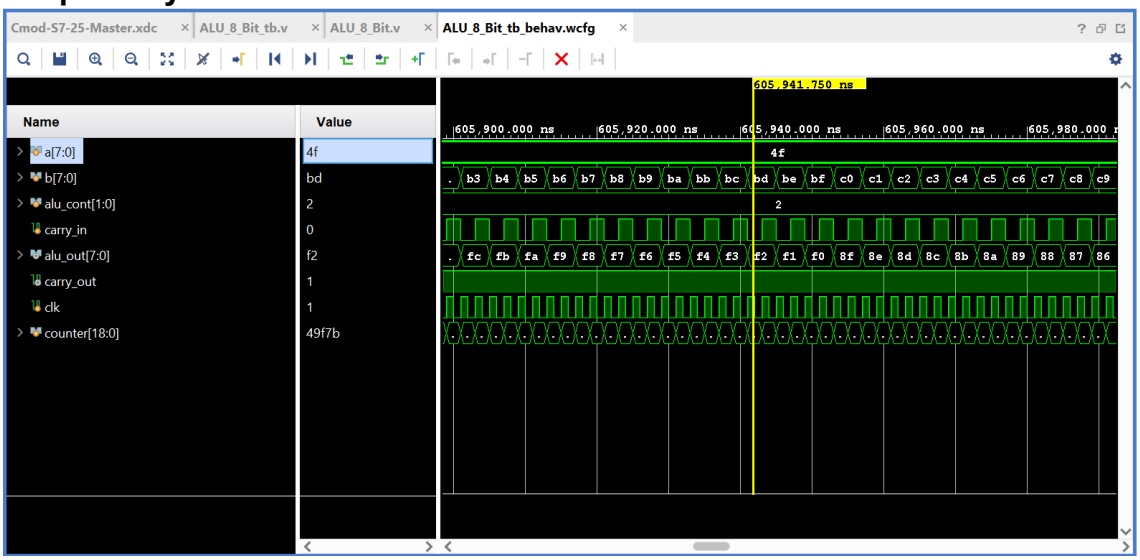
Tabel kebenaran:

Operasi:	**alu_cont = 1 (Operasi OR)**					
Jenis:	Input ALU			Alu selector	Output ALU	
Bilangan:	A	B	carry_in	alu_cont	carry_out	alu_out
Size:	8 Bit	8 Bit	1 Bit	2 Bit	1 Bit	8 Bit
Hexadecimal:	1F	C2	0	1	0	DF
Biner:	0001 1111	1100 0010	0	01	0	1101 1111

Hasil calculator:



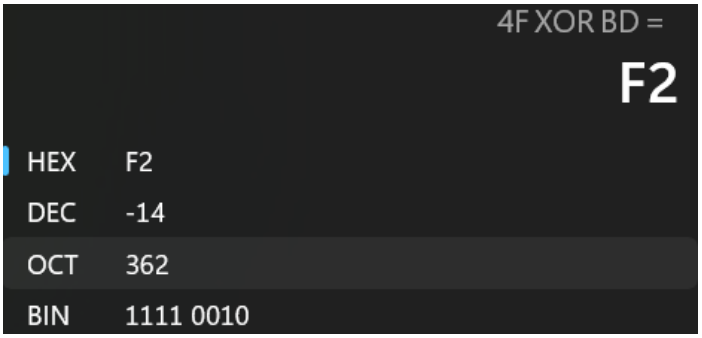
Output sinyal:



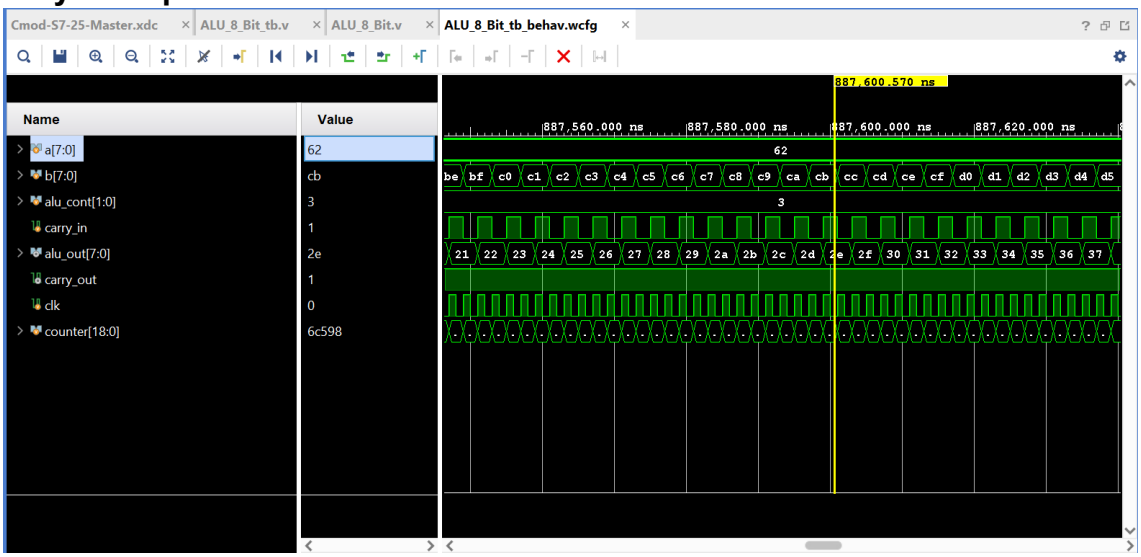
Tabel kebenaran:

Operasi:	**alu_cont = 2 (Operasi XOR)**					
Jenis:	Input ALU			Alu selector	Output ALU	
Bilangan:	A	B	carry_in	alu_cont	carry_out	alu_out
Size:	8 Bit	8 Bit	1 Bit	2 Bit	1 Bit	8 Bit
Hexadecimal:	4F	BD	0	2	1	F2
Biner:	0100 1111	1011 1101	0	10	1	1111 1111

Hasil Calculator:



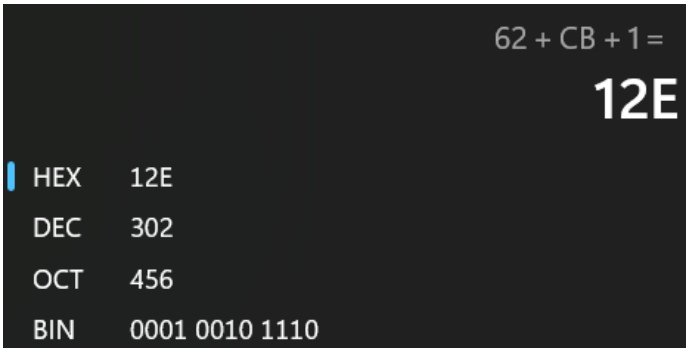
Sinyal Output:



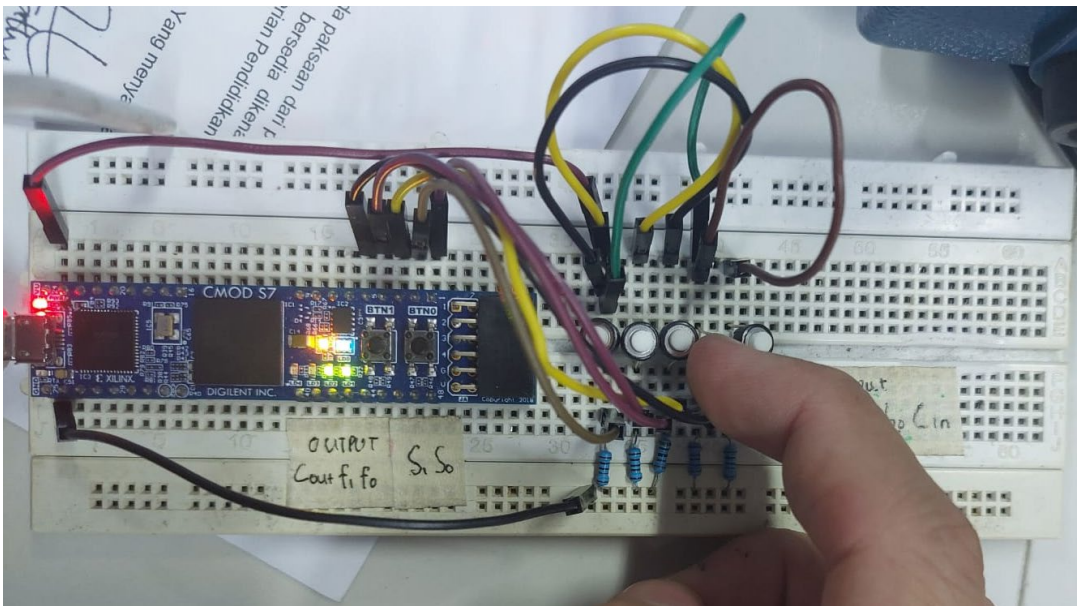
Tabel kebenaran:

Operasi:	**alu_cont = 3 (Operasi Full Adder)**					
Jenis:	Input ALU			Alu selector	Output ALU	
Bilangan:	A	B	carry_in	alu_cont	carry_out	alu_out
Size:	8 Bit	8 Bit	1 Bit	2 Bit	1 Bit	8 Bit
Hexadecimal:	62	CB	1	3	1	2E
Biner:	0110 0010	1100 1011	1	11	1	0010 1110

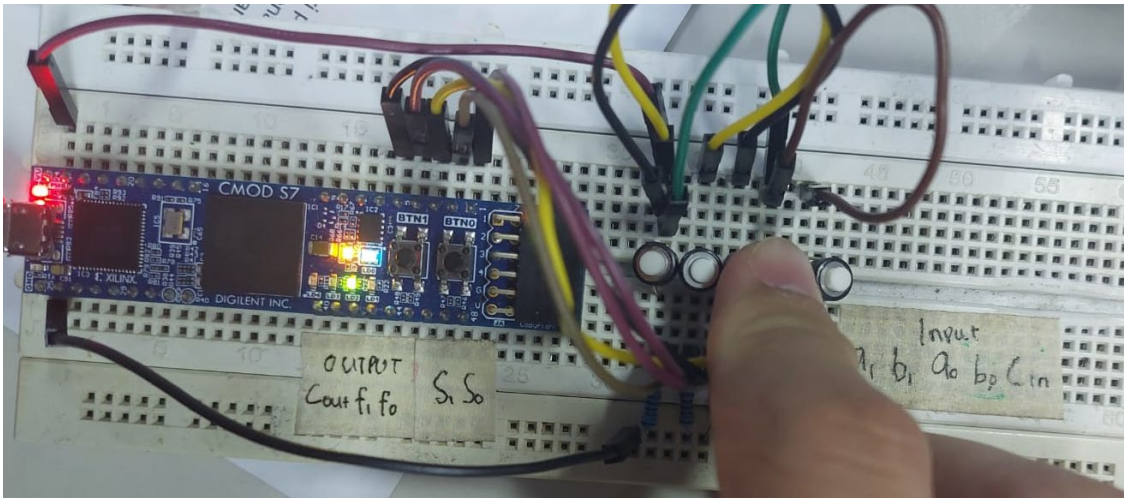
Calculator:



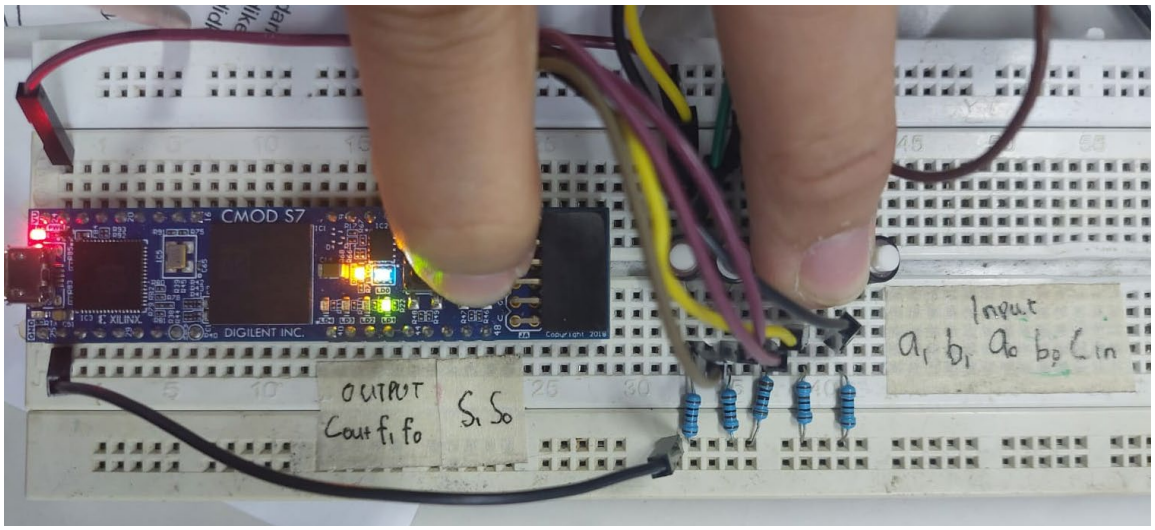
Implementasi FPGA dengan ALU 2 Bit:



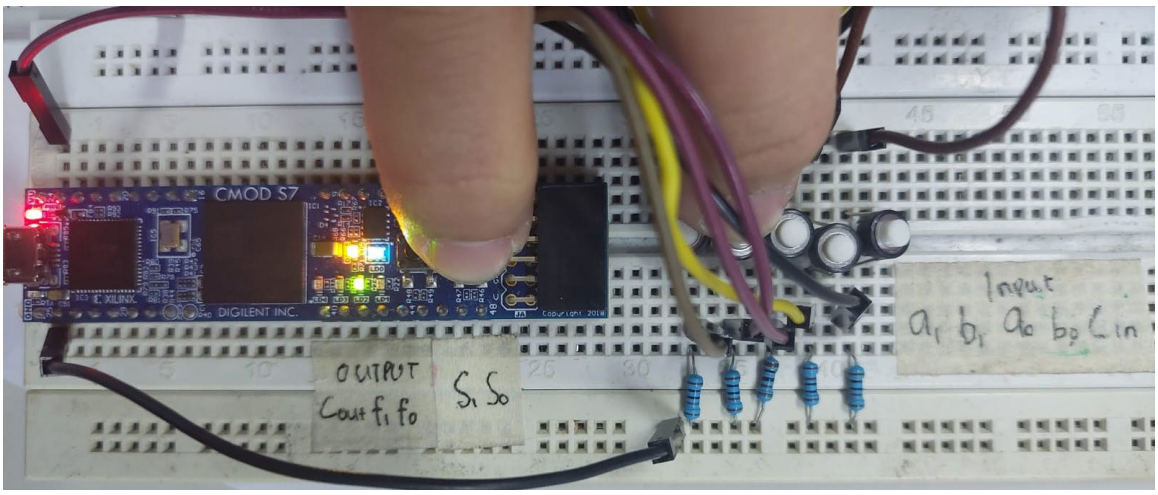
NAND									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
0	0	0	1	1	0	0	0	1	0



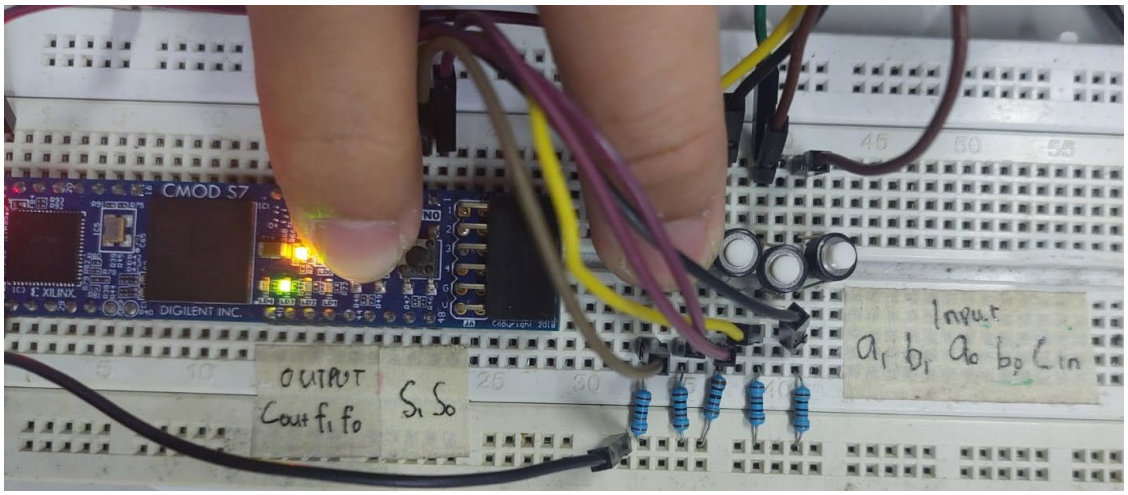
NAND									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
0	0	0	1	0	0	0	1	1	0



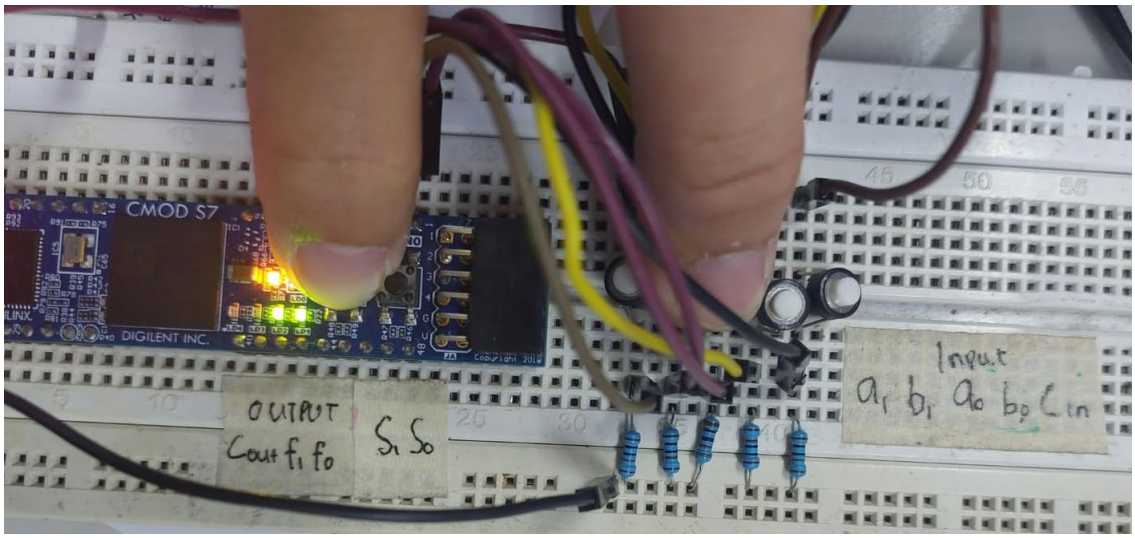
OR									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
0	1	0	0	1	0	0	1	1	0



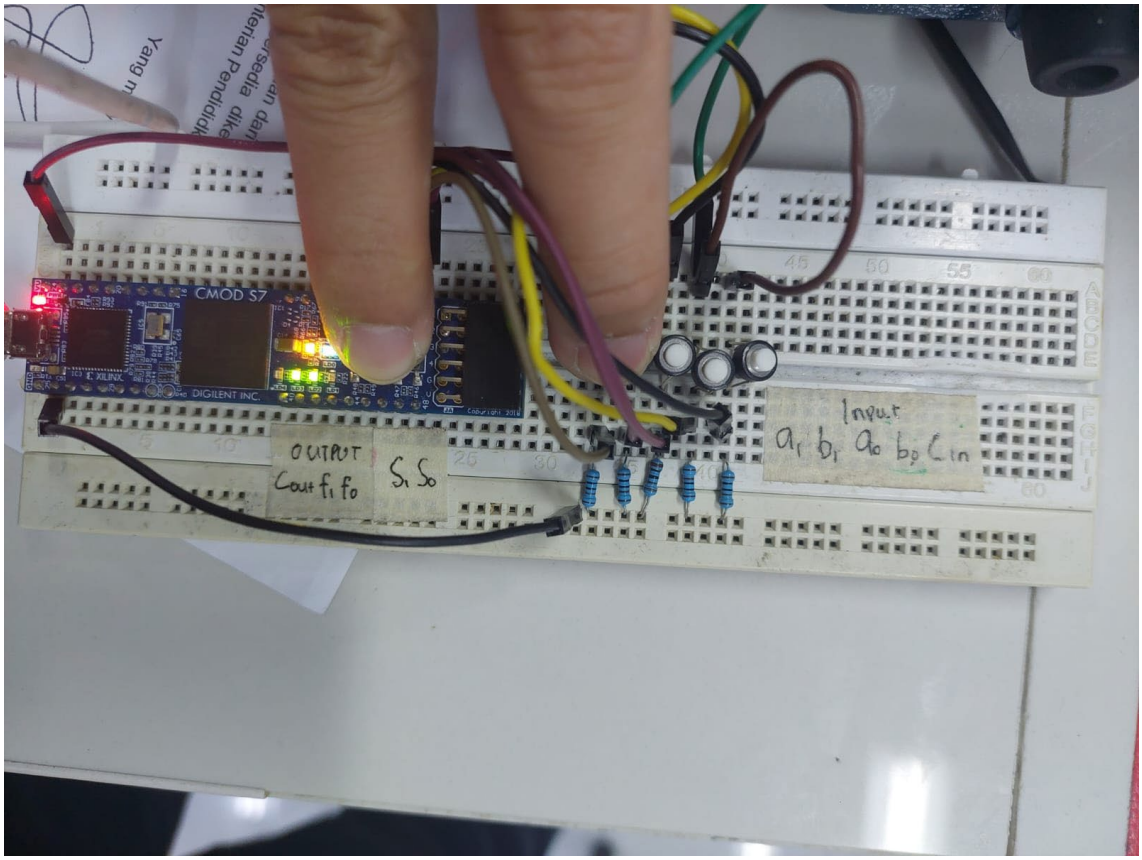
OR									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
0	1	0	1	0	0	1	0	0	0



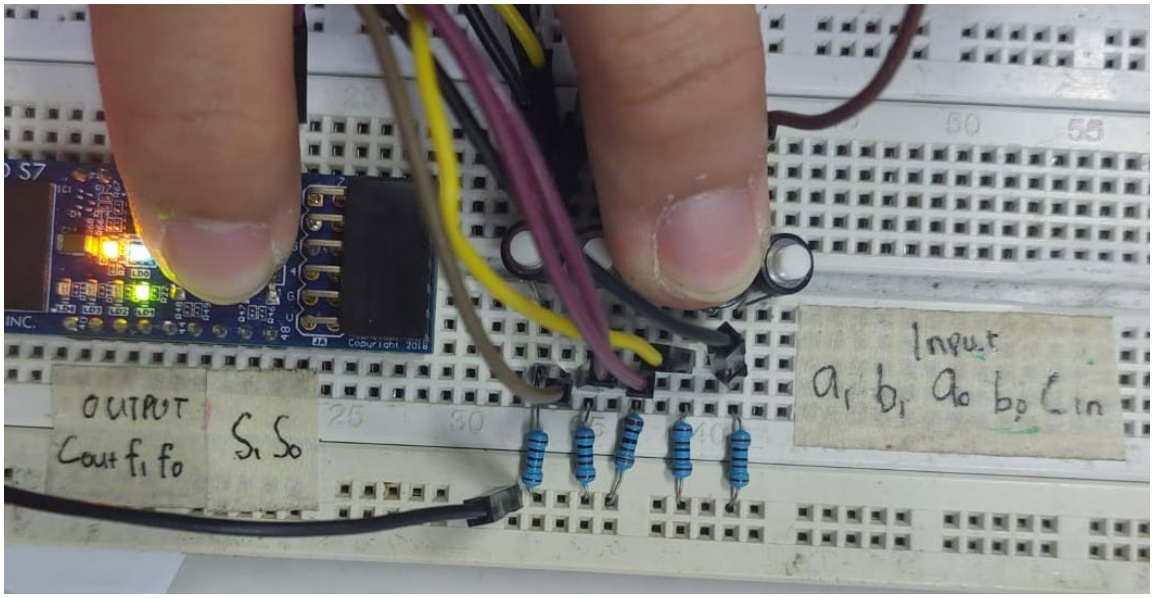
XOR									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
1	0	1	0	0	1	1	0	0	0



XOR									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
1	0	0	1	1	0	1	1	0	0



Full Adder									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
1	1	1	1	0	1	1	0	0	0



Full Adder									
S1	S0	Cout	F1	F0	A1	B1	A0	B0	Cin
1	1	0	0	1	0	0	1	1	0

Program ALU 2 Bit:

```
`timescale 1ns / 1ps
module ALU_2_Bit(
        input [1:0] a,b,
        input [1:0] alu_cont,
        input carry_in,
        output reg [1:0] alu_out,
        output reg carry_out
    );
    reg [2:0] over;

    always@(*) begin
        case(alu_cont)
            2'b00: alu_out = ~(a & b);
            2'b01: alu_out = a | b;
            2'b10: alu_out = a ^ b;
            2'b11: alu_out = a + b + carry_in;
            default: alu_out = ~(a & b);
        endcase
        over = {1'b0,a} + {1'b0,b};
        carry_out = over[2];
    end
endmodule
```

Repository:

https://github.com/yohanesstef/ALU_8-Bit_PKT.git