Objectives:

- Designing and Implementing 8-Bit ALU in Behavioral and Structural methods as shown in *figure (1-1)*.
- Performing the time and power analysis.
- Improving the design metrics to achieve optimized design.

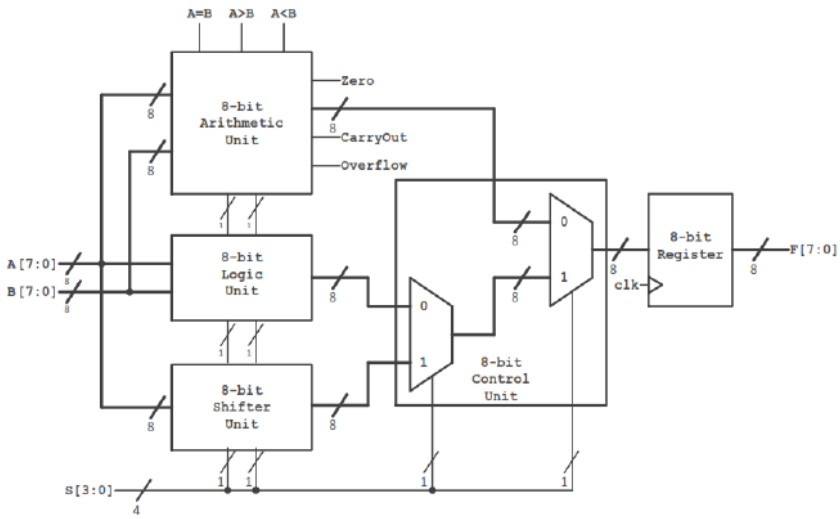


Figure (1-**Error! No text of specified style in document.**-1): ALU 8 bit

S0	S1	S2	S3	F	Description
0	0	0	0	A+B	Add
0	0	0	1	A-B	Subtract
0	0	1	1	B'+1	2's Complement
1	0	0	0	A AND B	AND
1	0	0	1	A XOR B	XOR
1	0	1	0	A OR B	OR
1	0	1	1	B'	1's Complement
1	1	0	0	A → →	RIGHT ROTATE
1	1	0	1	← ← A	LEFT ROTATE
1	1	1	0	A →	RIGHT SHIFT
1	1	1	1	← A	LEFT SHIFT

***Error! No text of specified style in document.**2*

Behavioral Method:

- We start by declaring inputs and outputs and their sizes for the ALU in *figure (1-1).*
- We select what function (Arithmetic or Logic or Shift) will be performed according to the selection input as shown in figure (1-2).
- We expressed the *overflow, zero, A>B, A<B and A=B* as shown in figure (1-3).
- The change in output only occurs when the clock reaches its positive edge.
- Notice the truth table in figure (1-2), You will figure out that:
  - Arithmetic operations will occur when the S_0 and S_1 become 0.
  - Logic operations will occur when the S_0 becomes 1 and S_1 becomes 0.
  - Shift operations will occur when the S_0 and S_1 becomes 1.
- The selections S_3 and S_4 will determine the internal operations in each unit.

```
module ALU_8(input [7:0] A,input [7:0] B,input clk, input cin,
input [3:0] s, output reg A_EQUAL_B,output reg A_GREATER_B,output reg A_SMALLER_B ,
output reg CARRY,output reg ZERO , output reg OVERFLOW ,
output reg[7:0] F);

always @(posedge clk)
begin
//ARITHMETIC
if(s[3:2] == 2'b00)
begin
A_EQUAL_B <= (A==B ? 1'b1:1'b0);
A_GREATER_B <= (A>B ? 1'b1:1'b0);
A_SMALLER_B <= (A<B ? 1'b1:1'b0);
//A+B
        if(s[1:0] == 2'b00)
        begin
        {CARRY,F} <= A+B+cin;
        if(F==8'b0)ZERO <= 1'b1;
        else ZERO <= 1'b0;
        if(~F[7]&A[7]&B[7])OVERFLOW <= 1'b1;///COMMENT
        else OVERFLOW <= 1'b0;
        end
//A-B
        else if(s[1:0] == 2'b01)
        begin
        {CARRY,F} <= A-B;
        ZERO <= (F==8'b0 ? 1'b1:1'b0);
        if(~F[7]&A[7]&B[7])OVERFLOW <= 1'b1;///COMMENT
        else OVERFLOW <= 1'b0;
        end
//2's complement
        else if(s[1:0] == 2'b11)
        begin
        {CARRY,F} <= ~B+8'b00000001;
        ZERO <= (F==8'b0 ? 1'b1:1'b0);
        if(~F[7]&A[7]&B[7])OVERFLOW <= 1'b1;///COMMENT
        else OVERFLOW <= 1'b0;
        end
```

figure1-3: Arithmetic unit

```
end

//LOGIC UNIT
else if(s[3:2] == 2'b10)
begin
        if(s[1:0] == 2'b00)
        begin
        F <= A&B;
        end
        if(s[1:0] == 2'b01)
        begin
        F <= A^B;
        end
        if(s[1:0] == 2'b10)
        begin
        F <= A|B;
        end
        if(s[1:0] == 2'b11)
        begin
        F <= ~B;
        end

end
```

Figure1-4: Logic unit

```
//shift UNIT
else if(s[3:2] == 2'b11)
begin
//right rotate
        if(s[1:0] == 2'b00)
        begin
        F <= {A[0],A[7:1]};
        end
        //left rotate
        if(s[1:0] == 2'b01)
        begin
        F <= {A[6:0],A[7]};
        end
        //right shift
        if(s[1:0] == 2'b10)
        begin
        F <=  A>>1;
        end
        //left shift
        if(s[1:0] == 2'b11)
        begin
        F <=  A<<1;
        end
end

end
endmodule
```

Figure1-4: Shift unit

Test bench:

- Starting by instantiating an object from ALU module.
- Declaring the clock to change the inputs according to the clock signal.
- We build a task called CHECK to make sure that the expected outputs are the same as the actual outputs and display them as shown in figure (1-6).
- Some Inputs Values and their expected outputs can be found in figure (1-7).
- The simulation outputs will be found in figures (1-8) and (1-9).

```
module ALU_8_tb();
reg cin , clk;
reg [7:0] A;
reg [7:0] B;
reg [3:0] S;
wire [7:0] F;
wire  A_EQUAL_B, A_GREATER_B, A_SMALLER_B , CARRY, ZERO ,  OVERFLOW ;

ALU_8 obj(.A(A), .B(B),.cin(cin), .s(S),.F(F),.OVERFLOW(OVERFLOW),.ZERO(ZERO),.CARRY(CARRY),
.A_SMALLER_B(A_SMALLER_B),.A_GREATER_B(A_GREATER_B),.A_EQUAL_B(A_EQUAL_B),.clk(clk));

initial begin
    clk = 1;
    forever #1 clk = ~clk; // Invert every 5 time units (adjust as needed)
  end
```

*Error! No text of specified style in document.**5***

```
task check (input[7:0] out);
begin
if (out == F)
 $display("Test case is successful for F = %b and S = %b,OVERFLOW = %b , ZERO = %b , CARRY = %b , A_SMALLER_B = %b , A_GREATER_B = %b , A_EQUAL_B = %b"
, F, S,OVERFLOW,ZERO,CARRY,A_SMALLER_B,A_GREATER_B,A_EQUAL_B);
else
$display("Test case is failed for A = %b, B = %b, S = %b, F = %b , OUT = %B", A, B, S, F,out);
end
endtask
endmodule
```

Figure1-6: CHECK task used to make sure that the expected output is the same as the actual output.

```
initial
begin
A = 8'b11111111; B = 8'b11111111; cin = 1'b0; S = 4'b0000; //ADD carry = 1
#2
check(8'b11111110);
A = 8'b00000010; B = 8'b00000011;cin = 1'b0  ; S = 4'b0001; //SUB  2-3
#2
check('b11111111);
A = 8'b00000010; B = 8'b00000011; S = 4'b0011; //2's complement
#2
check(8'b11111101);
A = 8'b00000010;B=8'b00000011; S = 4'b1000; //AND
#2
check(8'b00000010);
A = 8'b00000010;B=8'b00000011; S = 4'b1001; //XOR
#2
check(8'b00000001);
A = 8'b00000010;B=8'b00000011; S = 4'b1010; //OR
#2
check(8'b00000011);
A = 8'b00000010;B=8'b00000011; S = 4'b1011; //1's complement
#2
check(8'b11111100);
A = 8'b00000010;B=8'b00000011; S = 4'b1100; //right rotate
#2
check(8'b00000001);
A= 8'b00000010;B=8'b00000011; S = 4'b1101; //left rotate
#2
check(8'b00000100);

A = 8'b00000010;B=8'b00000011; S = 4'b1110; //right shift
#2
check(8'b00000001);

A = 8'b00000010;B=8'b00000011; S = 4'b1111; //left shift
#2
check(8'b00000100);
   end
```

Figure1-7

Simulation Results:

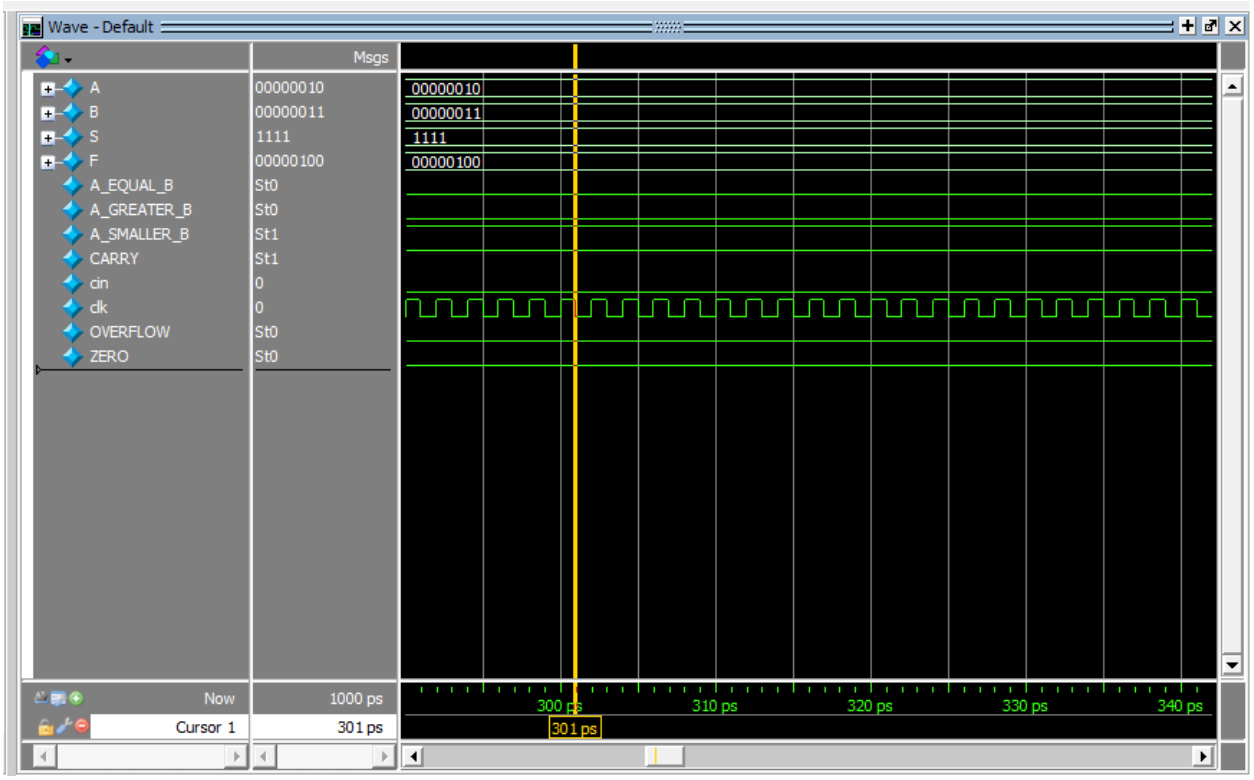


Figure1-8: wave

```
# Test case is successful for F = 11111110 and S = 0000,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 0 , A_GREATER_B = 0 , A_EQUAL_B = 1
# Test case is successful for F = 11111111 and S = 0001,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 11111101 and S = 0011,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000010 and S = 1000,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000001 and S = 1001,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000011 and S = 1010,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 11111100 and S = 1011,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000001 and S = 1100,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000100 and S = 1101,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000001 and S = 1110,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000100 and S = 1111,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
```

Figure 1-10: test bench results

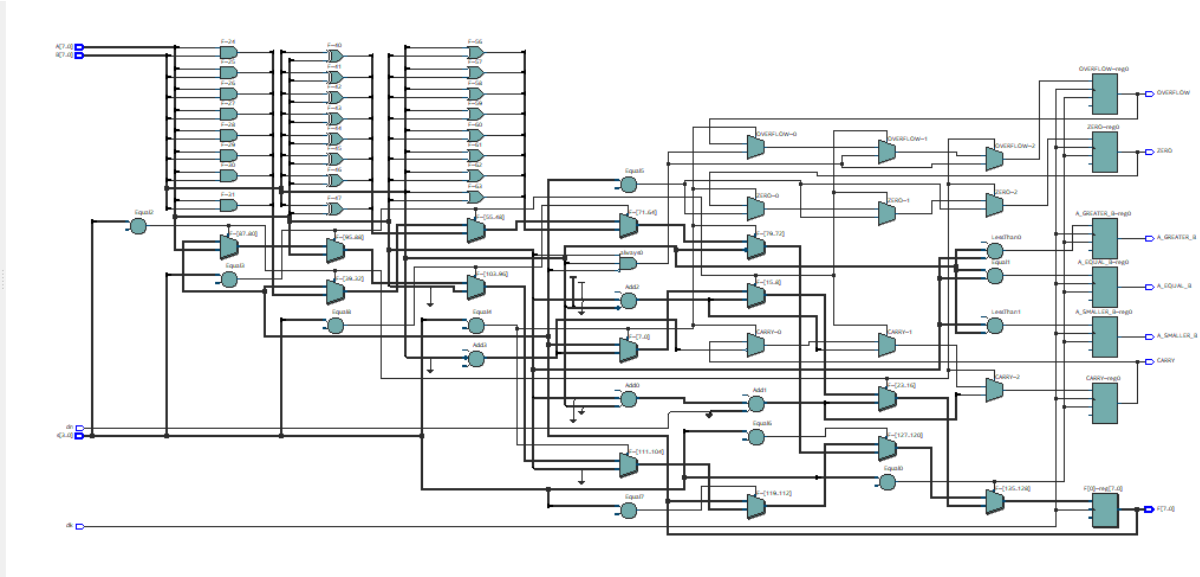


Figure1-10

Time Analysis:

- The time analysis tool in quartos before modification appears in figure (1-11).
- We need to set the frequency of the clock to be under 513.87 MHz as shown in figure (1-12).
- The suitable period time is 1.946 ns.
- The clock, FMAX, Hold and setup time after modifying the can be found on figures (1-13,14,15).
- Our focus will be the worst case here [slow 1100mv 85c model].

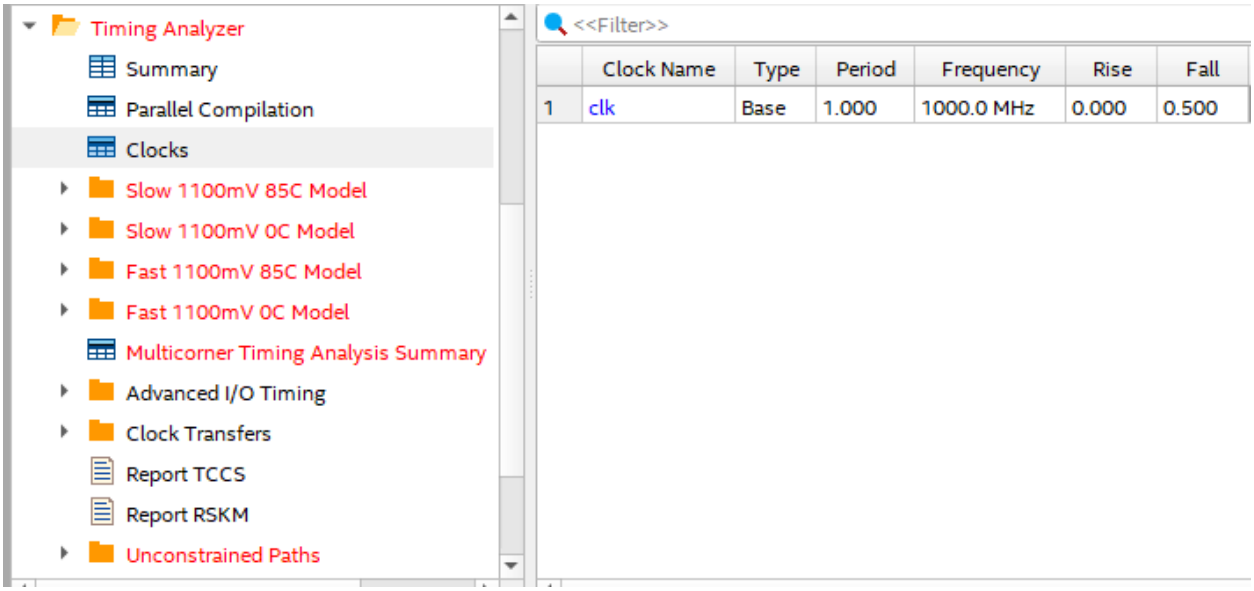


Figure 1-11

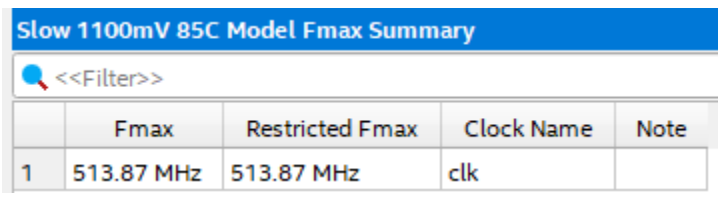


Figure 1-12 FMAX Before sitting the clock period.

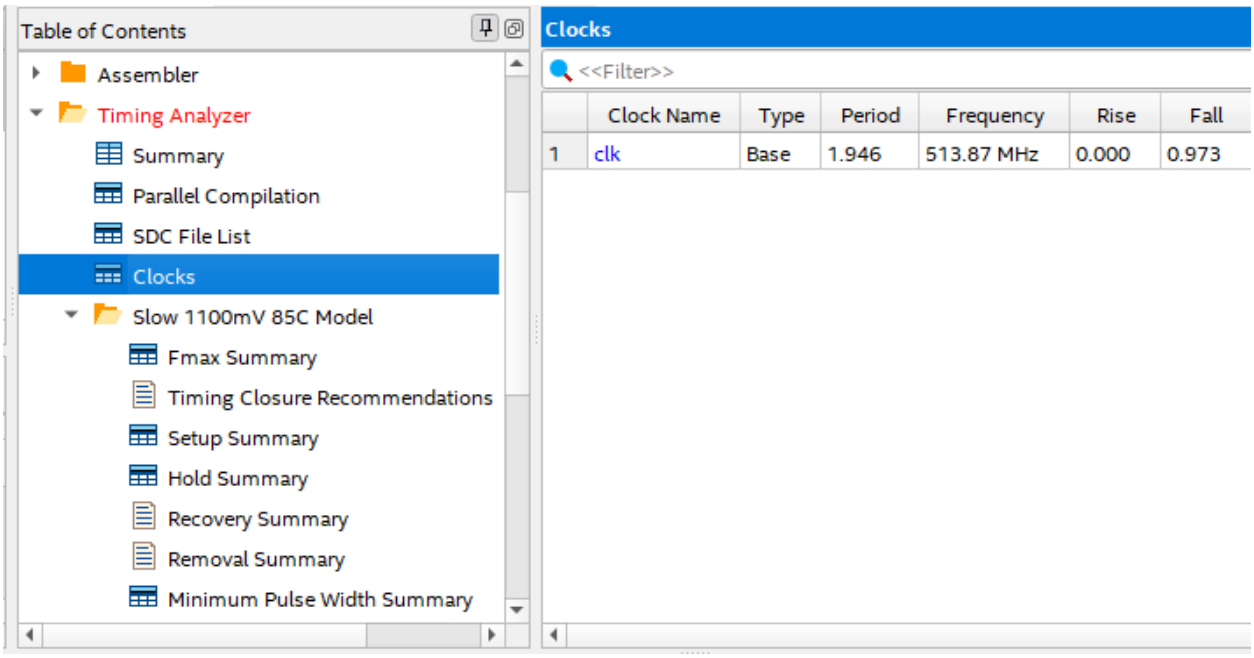


Figure 1-13

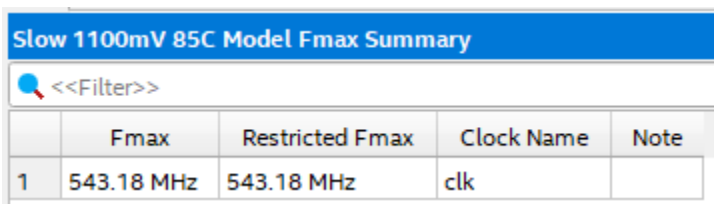


Figure1-14

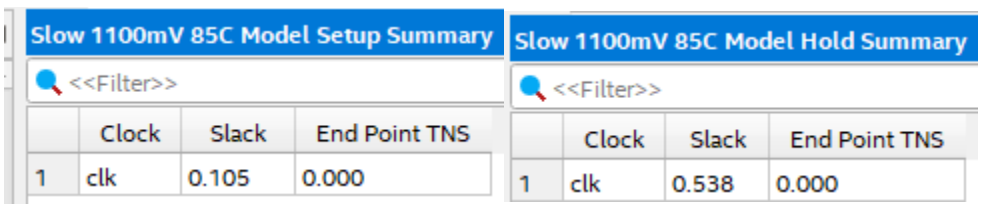


Figure1-15

Power Analysis:

- We can decrease the power dissipation by adding enable wires to the design in order to control which unit should be enabled.

Power Analyzer Summary

<<Filter>>

Power Analyzer Status	Successful - Mon Dec 18 23:37:43 2023
Quartus Prime Version	22.1std.0 Build 915 10/25/2022 SC Lite Edition
Revision Name	ALU_8
Top-level Entity Name	ALU_8
Family	Cyclone V
Device	5CGXFC7C7F23C8
Power Models	Final
Total Thermal Power Dissipation	375.17 mW
Core Dynamic Thermal Power Dissipation	3.92 mW
Core Static Thermal Power Dissipation	349.58 mW
I/O Thermal Power Dissipation	21.66 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Figure1-16

Structural Method:

- We described each unit in structural way and separated each unit in file.
- The top module is shown in figure (2-1).
- The top module contains the instantiation of other modules (ARITHMETIC, LOGIC, SHIFT).
- We added an enable pin to control the Alu operations to optimize the power consumption.
- If we return to the selection table in figure (1.2), we will find that the units depend on s_0 and s_1 as we mentioned before. So, the (always block) contain a case statement the initialize the required enable according to the selection.

```
module ALU_8(input [7:0] A,input [7:0] B,input clk, input cin,
input [3:0] s, output  A_EQUAL_B,output  A_GREATER_B,output  A_SMALLER_B ,
output  CARRY,output  ZERO , output  OVERFLOW ,output [7:0]F);

reg en_AR,en_LO,en_SH;
wire [7:0]F_AR,F_LO,F_SH;
wire [7:0] F_CON;

ARITH obj1(.en(en_AR),.A(A),.B(B),.cin(cin),.s(s[1:0]),.OVERFLOW(OVERFLOW),.F_AR(F_AR),.CARRY(CARRY),.A_EQUAL_B(A_EQUAL_B)
,.A_GREATER_B(A_GREATER_B),.A_SMALLER_B(A_SMALLER_B),.zero(ZERO));

LOGIC obj2(.en(en_LO),.A(A),.B(B),.s(s[1:0]),.F_LO(F_LO));

SHIFT obj3(.en(en_SH),.s(s[1:0]),.A(A),.F_SH(F_SH));

CONTROL obj4(.F_LO(F_LO),.F_SH(F_SH),.F_AR(F_AR),.s(s[3:2]),.F_CON(F_CON));

//REGISTER obj5(.w(F_CON),.clk(clk),.F_R(F));

assign F = F_CON;
always @(posedge clk)
begin
case(s[3:2])
2'b00:begin en_AR = 1'b1;en_LO = 1'b0;en_SH=1'b0; end
2'b10:begin en_AR = 1'b0;en_LO = 1'b1;en_SH=1'b0; end
2'b11:begin en_AR = 1'b0;en_LO = 1'b0;en_SH=1'b1; end
endcase
//$display("w = %b , F_R = %b", obj5.w,obj5.F_R);
end
endmodule
//enable for arith &logic&shift
```

Figure 2-1: the top module

- The logic module contains the logic operations and, It is controlled by selection S_3 and S_4.

```
module LOGIC(input  en,input [7:0]A,input [7:0]B,input [1:0]s,
output reg [7:0]F_LO
);
always @(*)
begin
if(en == 1'b1)begin
        if(s[1:0] == 2'b00)
        begin
        F_LO <= A&B;
        end
        if(s[1:0] == 2'b01)
        begin
        F_LO <= A^B;
        end
        if(s[1:0] == 2'b10)
        begin
        F_LO <= A|B;
        end
        if(s[1:0] == 2'b11)
        begin
        F_LO <= ~B;
        end
end
end
endmodule
```

Figure2-2: Logic unit module.

- SHIFT unit that contains shift and rotate operations

```
module SHIFT(
input  en,input [1:0] s,input[7:0]A,
output reg [7:0]F_SH
);

always @(*)
begin
if(en == 1'b1)begin
if(s[1:0] == 2'b00)
        begin
        F_SH <= {A[0],A[7:1]};
        end
        //left rotate
        if(s[1:0] == 2'b01)
        begin
        F_SH <= {A[6:0],A[7]};
        end
        //right shift
        if(s[1:0] == 2'b10)
        begin
        F_SH <=  A>>1;
        end
        //left shift
        if(s[1:0] == 2'b11)
        begin
        F_SH <=  A<<1;
        end
end
end
endmodule
```

Figure 2-3: SHIFT UNIT

- Multiplixer module has 2 inputs and 1 output.

```
module MUX_2_1(
input  [7:0] w1,input [7:0] w2,input s,
output [7:0] F_M
);
 assign F_M = ((s) ? w2:w1);

endmodule
```

Figure 2-4: MUX unit

- Arithmetic Unit shown in figures (2-5.1 and 2-5.2) contains three modules:
  - 8-Bit Full Adder.

```
module XOR_gate(input A, input B, output Y);
    assign Y = A ^ B;
endmodule
module AND_gate(input A, input B, output Y);
    assign Y = A & B;
endmodule
module OR_gate(input A, input B, output Y);
    assign Y = A | B;
endmodule
module NOT_gate(input A, output Y);
    assign Y = ~A;
endmodule
module FullAdder(input A, input B, input Cin, output Sum, output Cout);
    // XOR gates for sum and intermediate carry
    XOR_gate xor1(A, B, S0);
    XOR_gate xor2(S0, Cin, Sum);

    // AND gates for generating carries
    AND_gate and1(A, B, C1);
    AND_gate and2(S0, Cin, C2);
    AND_gate and3(A, Cin, C3);

    // OR gates for final carry-out
    OR_gate or1(C1, C2, C4);
    OR_gate or2(C3, C4, Cout);
endmodule
module EightBitFullAdder(input [7:0] A, input [7:0] B, input Cin, output [7:0] Sum, output Cout,output OVERFLOW);
    wire [7:0] C;
    FullAdder f0(A[0], B[0], Cin, Sum[0], C[0]);
    FullAdder f1(A[1], B[1], C[0], Sum[1], C[1]);
    FullAdder f2(A[2], B[2], C[1], Sum[2], C[2]);
    FullAdder f3(A[3], B[3], C[2], Sum[3], C[3]);
    FullAdder f4(A[4], B[4], C[3], Sum[4], C[4]);
    FullAdder f5(A[5], B[5], C[4], Sum[5], C[5]);
    FullAdder f6(A[6], B[6], C[5], Sum[6], C[6]);
    FullAdder f7(A[7], B[7], C[6], Sum[7], Cout);
//assign OVERFLOW = C[6]^Cout;i will do it behav
endmodule
```

- 8-Bit Full Subtractor.

```
module FullSubtractor(input A, input B, input Bin, output Dif, output Bout);
    wire D1, D2, D3, B1, B2;

    // XOR gates for difference and intermediate borrow
    XOR_gate xor1(A, B, D1);
    XOR_gate xor2(D1, Bin, Dif);

    // AND gates for generating borrows
    AND_gate and1(~A, B, B1);
    AND_gate and2(~D1, Bin, B2);
    AND_gate and3(A, Bin, D2);

    // OR gates for final borrow-out
    OR_gate or1(B1, B2, Bout);
    OR_gate or2(D2, Bout, D3);
endmodule

module EightBitFullSubtractor(input [7:0] A, input [7:0] B, input Bin, output [7:0] Dif, output Bout,output OVERFLOW);
    wire [7:0] Bouts;

    // Instantiate eight full subtractors
    FullSubtractor fs0(A[0], B[0], Bin, Dif[0], Bouts[0]);
    FullSubtractor fs1(A[1], B[1], Bouts[0], Dif[1], Bouts[1]);
    FullSubtractor fs2(A[2], B[2], Bouts[1], Dif[2], Bouts[2]);
    FullSubtractor fs3(A[3], B[3], Bouts[2], Dif[3], Bouts[3]);
    FullSubtractor fs4(A[4], B[4], Bouts[3], Dif[4], Bouts[4]);
    FullSubtractor fs5(A[5], B[5], Bouts[4], Dif[5], Bouts[5]);
    FullSubtractor fs6(A[6], B[6], Bouts[5], Dif[6], Bouts[6]);
    FullSubtractor fs7(A[7], B[7], Bouts[6], Dif[7], Bout);
//xor (OVERFLOW,Bouts[6],Bout);i will do it behav
endmodule
```

- 8-Bit two's Complement.

```
module TwosComplement(input [7:0] A, output [7:0] TwosComp ,output T_C,output OVERFLOW);
    wire [7:0] onesComplement;
    wire [7:0] C;

    // ones' complement using XOR gates
XOR_gate xor0(A[0], 1'b1, onesComplement[0]);
XOR_gate xor1(A[1], 1'b1, onesComplement[1]);
XOR_gate xor2(A[2], 1'b1, onesComplement[2]);
XOR_gate xor3(A[3], 1'b1, onesComplement[3]);
XOR_gate xor4(A[4], 1'b1, onesComplement[4]);
XOR_gate xor5(A[5], 1'b1, onesComplement[5]);
XOR_gate xor6(A[6], 1'b1, onesComplement[6]);
XOR_gate xor7(A[7], 1'b1, onesComplement[7]);

FullAdder fa0(onesComplement[0], 1'b1, 1'b0, TwosComp[0], C[0]);
FullAdder fa1(onesComplement[1], 1'b0, C[0], TwosComp[1], C[1]);
FullAdder fa2(onesComplement[2], 1'b0, C[1], TwosComp[2], C[2]);
FullAdder fa3(onesComplement[3], 1'b0, C[2], TwosComp[3], C[3]);
FullAdder fa4(onesComplement[4], 1'b0, C[3], TwosComp[4], C[4]);
FullAdder fa5(onesComplement[5], 1'b0, C[4], TwosComp[5], C[5]);
FullAdder fa6(onesComplement[6], 1'b0, C[5], TwosComp[6], C[6]);
FullAdder fa7(onesComplement[7], 1'b0, C[6], TwosComp[7], T_C);
//assign OVERFLOW = C[6]^T_C;i will do it behav
endmodule
```

```
module ARITH(input  en,input [7:0] A, input [7:0] B, input cin ,
 input [1:0]s,output reg OVERFLOW, output reg [7:0] F_AR ,output reg CARRY ,
output reg A_EQUAL_B,output reg A_GREATER_B,output reg A_SMALLER_B,
output reg zero
);
wire [7:0] Sum;
wire Cout,oa;
wire [7:0] Subtr;
wire Bout,os;
wire [7:0] TwosComp;
wire T_C,ot;

// Instantiate the 8-bit full adder
EightBitFullAdder fa(A, B, cin, Sum,Cout,oa);
EightBitFullSubtractor sub(A,B,cin,Subtr,Bout,os);
TwosComplement tw(B,TwosComp,T_C,ot);
```

Figure2-5.1: ARITHMATIC TOP MODULE

```
always @(*)

begin
if(en == 1'b1)begin
A_EQUAL_B <= (A==B ? 1'b1:1'b0);
A_GREATER_B <= (A>B ? 1'b1:1'b0);
A_SMALLER_B <= (A<B ? 1'b1:1'b0);
if(en)begin

end
case(s)
2'b00: begin
F_AR <= Sum;
CARRY <= Cout;
OVERFLOW <= oa;

end
2'b01:begin
 F_AR <= Subtr;
CARRY <= Bout;
OVERFLOW <= os;
end
2'b11: begin
F_AR <= TwosComp;
CARRY <= T_C;
OVERFLOW <= ot;
end
endcase
if(~F_AR[7]&A[7]&B[7])OVERFLOW <= 1'b1;///COMMENT
        else OVERFLOW <= 1'b0;
zero <= (F_AR==8'b0 ? 1'b1:1'b0);
end
end
endmodule
```

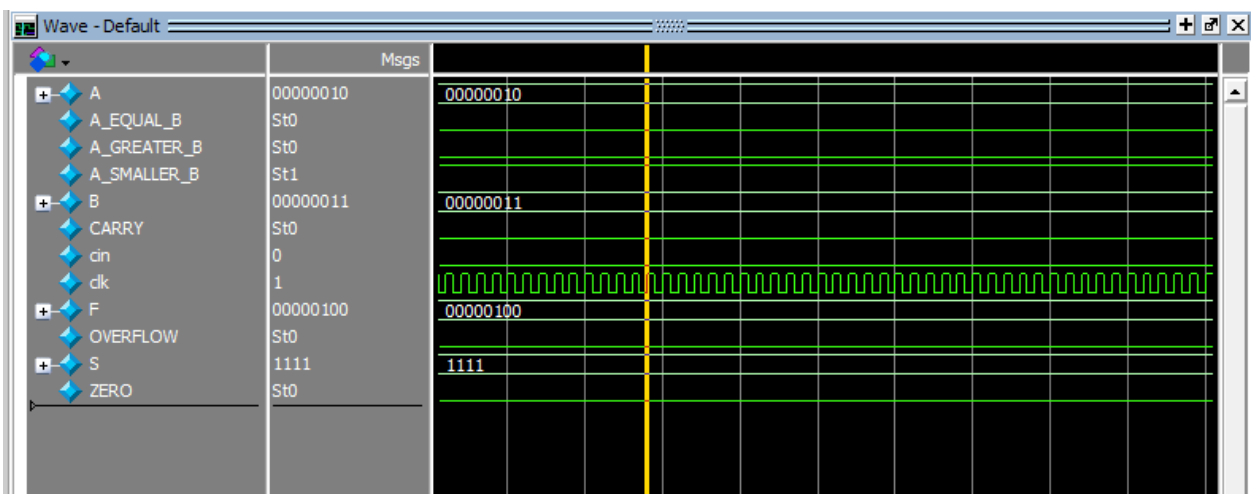
Figure2-5.2 Arithmetic flags

- Finally, we have the control unit as shown in figure (2-6) which uses the 2-1 MUX module we had built to assign the output F to the be the result of one of the three units' output (Arithmetic, Logic and shift) according to the selection.

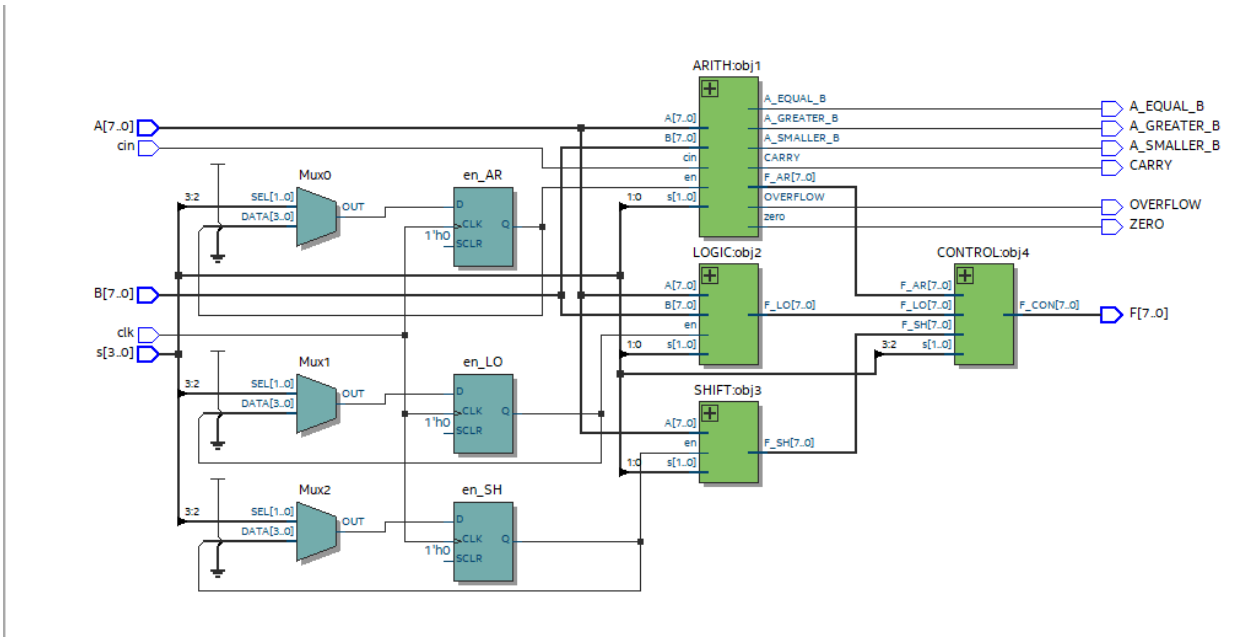
```
module CONTROL(output [7:0]F_CON, input wire[1:0]s,
input wire [7:0] F_AR,input wire [7:0] F_LO,input wire [7:0] F_SH
);
wire [7:0] FM_LO_SH;
MUX_2_1 LO_SH(.w1(F_LO),.w2(F_SH),.s(s[0]),.F_M(FM_LO_SH));
MUX_2_1 AR_LH(.w1(F_AR),.w2(FM_LO_SH),.s(s[1]),.F_M(F_CON));
endmodule
```

Figure 2-6: Control Unit

Simulation results:



RTL VIEW:



Test bench results:

- **We applied the same test bench of the behavioral method.**

```
# Test case is successful for F = 11111110 and S = 0000,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 0 , A_GREATER_B = 0 , A_EQUAL_B = 1
# Test case is successful for F = 11111111 and S = 0001,OVERFLOW = 0 , ZERO = 0 , CARRY = 1 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 11111101 and S = 0011,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000010 and S = 1000,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000001 and S = 1001,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000011 and S = 1010,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 11111100 and S = 1011,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000001 and S = 1100,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000100 and S = 1101,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000001 and S = 1110,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
# Test case is successful for F = 00000100 and S = 1111,OVERFLOW = 0 , ZERO = 0 , CARRY = 0 , A_SMALLER_B = 1 , A_GREATER_B = 0 , A_EQUAL_B = 0
```

Time analysis:

- The period before modification appears in figure (2-7).
- We need to set the frequency of the clock [clk] to be under restricted frequency. 513.87 MHz as shown in figure (2-11).
- The suitable period time is 1.946 ns.
- Our focus will be the worst case here [slow 1100mv 85c model].

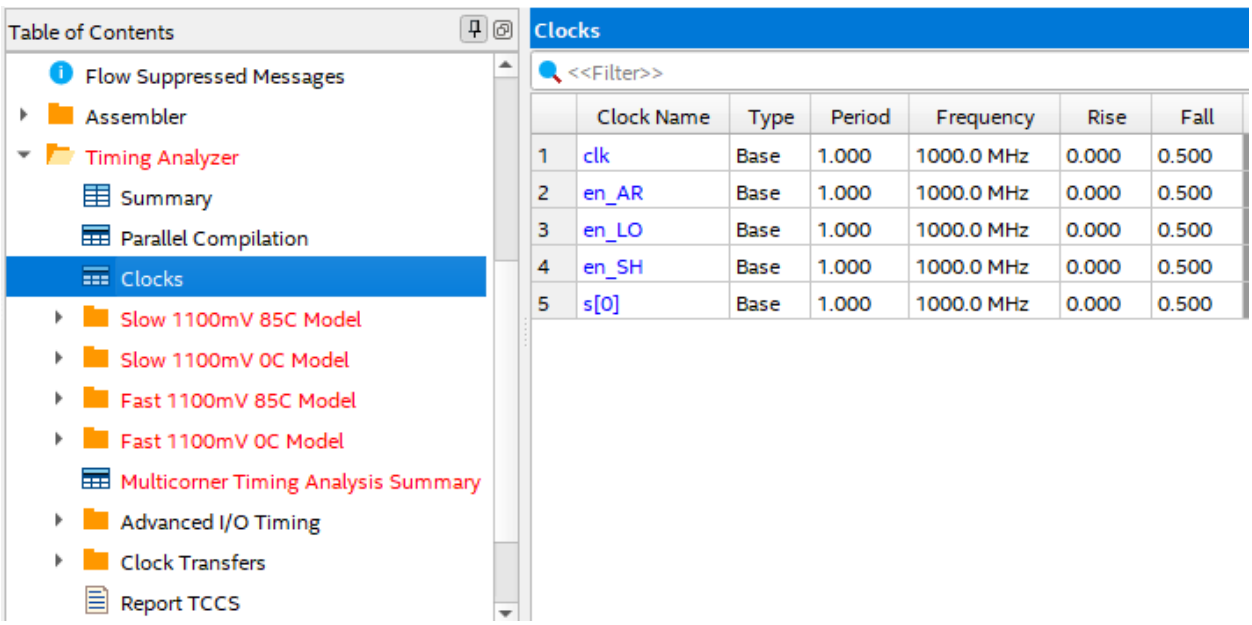


*Figure **Error! No text of specified style in document.**-2-7*

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	49.08 MHz	49.08 MHz	s[0]	
2	210.93 MHz	210.93 MHz	en_AR	

Figure 2-8: max frequency before modification

Slow 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	en_AR	-13.014	-121.068
2	s[0]	-9.687	-75.625
3	en_SH	-7.210	-56.387
4	en_LO	-7.148	-56.505

Figure 2-9: setup time before modification

Slow 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	en_AR	1.877	0.000
2	s[0]	2.051	0.000
3	en_SH	3.235	0.000
4	en_LO	3.315	0.000

Figure 2-10: hold time before modification.

Clocks

<<Filter>>

	Clock Name	Type	Period	Frequency	Rise	Fall
1	clk	Base	1.946	513.87 MHz	0.000	0.973
2	en_AR	Base	10.000	100.0 MHz	0.000	5.000
3	en_LO	Base	4.740	210.97 MHz	0.000	2.370
4	en_SH	Base	1.000	1000.0 MHz	0.000	0.500
5	s[0]	Base	20.408	49.0 MHz	0.000	10.200

Figure 2-11: clocks after modification

Power analysis:

- **WE can notice that the power dissipation decreased because we used the enables inputs.**

Power Analyzer Summary

<<Filter>>

Power Analyzer Status	Successful - Tue Dec 19 04:11:14 2023
Quartus Prime Version	22.1std.0 Build 915 10/25/2022 SC Lite Edition
Revision Name	ALU_ST
Top-level Entity Name	ALU_ST
Family	Cyclone V
Device	5CGXFC7C7F23C8
Power Models	Final
Total Thermal Power Dissipation	362.73 mW
Core Dynamic Thermal Power Dissipation	5.08 mW
Core Static Thermal Power Dissipation	349.51 mW
I/O Thermal Power Dissipation	8.15 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Figure 2-12