Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

OBJECTIVES

- The purpose of this lab is to learn how to connect simple input (switches) and output devices (LEDs and 7-segment) to an FPGA chip and implement a circuit that uses these devices.
- Examine arithmetic circuits that add, subtract, and multiply numbers.

PREPARATION FOR LAB 1

- Finish Pre Lab 1 at home.
- Students have to simulate all the exercises in Pre Lab 1 at home. All results (codes, waveform, RTL viewer, …) have to be captured and submitted to instructors prior to the lab session. *If not, students will not participate in the lab and be considered absent this session.*

REFERENCE

1. Intel FPGA training

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

EXPERIMENT 1

Objective: Known how to program a system to add the value of an input A to itself repeatedly.

Requirement: The circuit in Figure 1, which is often called an *accumulator*, is used to add the value of an input *A* to itself repeatedly. The circuit includes a carry out from the adder, as well as an *overflow* output signal. If the input *A* is considered as a 2's-complement number, then *overflow* should be set to 1 in the case where the output *sum* produced does not represent a correct 2's-complement result.

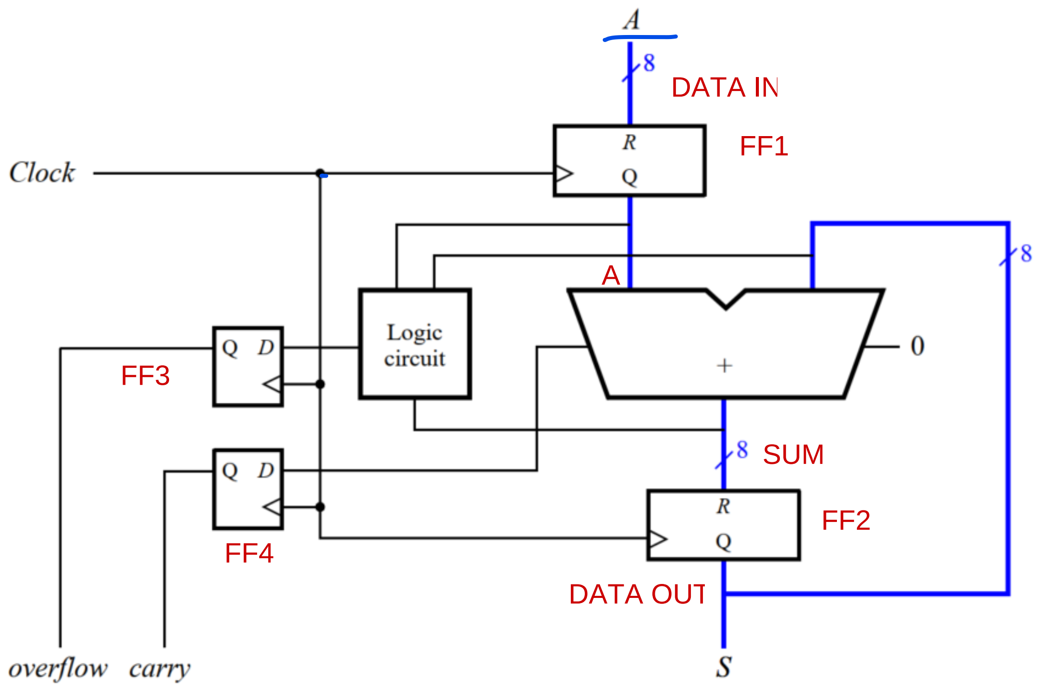


Figure 1: An eight-bit accumulator circuit.

Instruction:

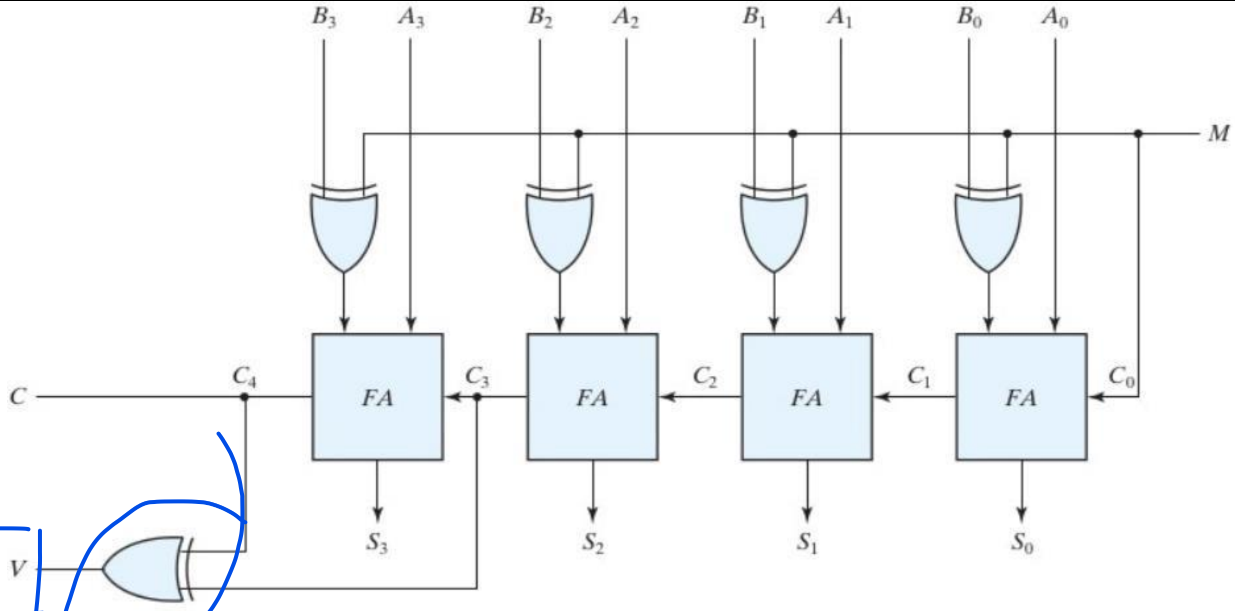
- Write System Verilog code to describe 4-bit adder circuit.
- There are two ways to detect overflow for 2's complement:

  - Compare inputs and outputs: The inputs and outputs always have the same sign. If not, the overflow must be set.

  - Compare carry bits: The circuit for checking overflow of 4-bit adder is shown below.

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS



In the figure, V is the overflow flag.

- Write System Verilog code to describe the circuit given in Figure 1.
  The code includes 4 module
  The module register to implement a 8bit register.

```
module register (
input clk,
input rst_n,
input [7:0] data_in,
output [7:0] data_out
);

reg [7:0] reg_data;
always @(posedge clk, negedge rst_n)begin
if (~rst_n)
reg_data <= 8'b0;
else
reg_data <= data_in;
end

assign data_out = reg_data ;
    endmodule
```

The module adder8bits to implement 8 bits full adder (the module for 1bit full adder already inside)

```
module adder1bit (
input A,
input B,
input Cin,
output Sum,
output Cout);
assign Sum = A^B^Cin;
assign Cout= (A&B | (A^B)&(Cin));
endmodule

module adder8bits (
input [7:0]A,
input [7:0]B,
input Cin,
output [7:0]Sum,
output Cout,
output V
);
wire [7:0] a;
assign a[0] = Cin;
assign V= a[6]^a[7];
assign  Cout = a[7];
genvar i;
generate
for ( i=0; i<7;i=i+1)begin: k1
adder1bit add(
.A(A[i]),
.B(B[i]),
.Cin(a[i]),
.Sum(Sum[i]),
.Cout(a[i+1])
);
end
endgenerate
    endmodule
```

The module d_ff to implement d flip flop

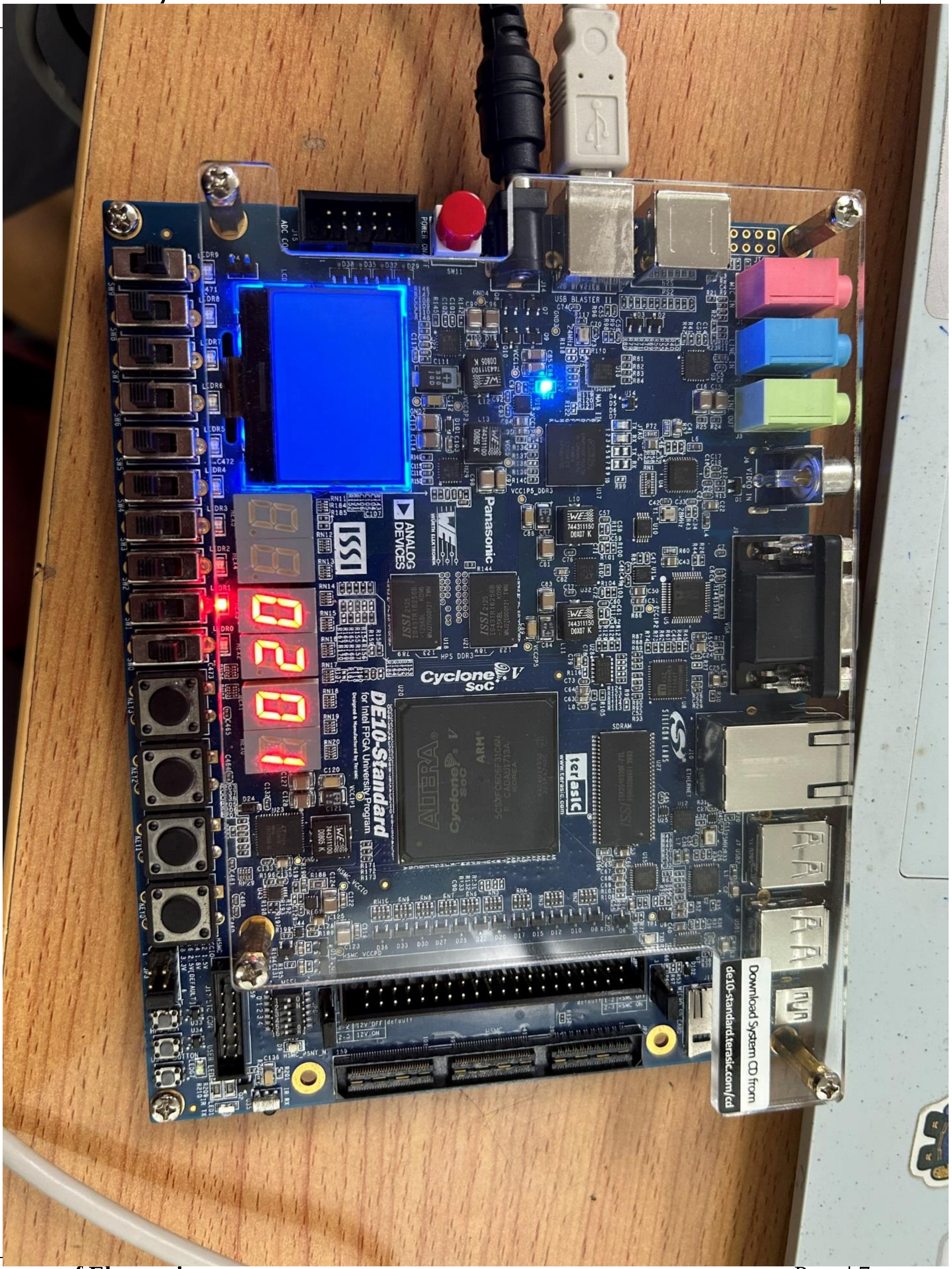
```
module d_ff(
input clk,
input rst_n,
input D,
output Q
);
always @(posedge clk, negedge rst_n)begin
if (~rst_n)
Q <= 1'b0;
else
Q <=D;
end
```

```
endmodule
```

The module lab1ex1lan2 to connect all modules above

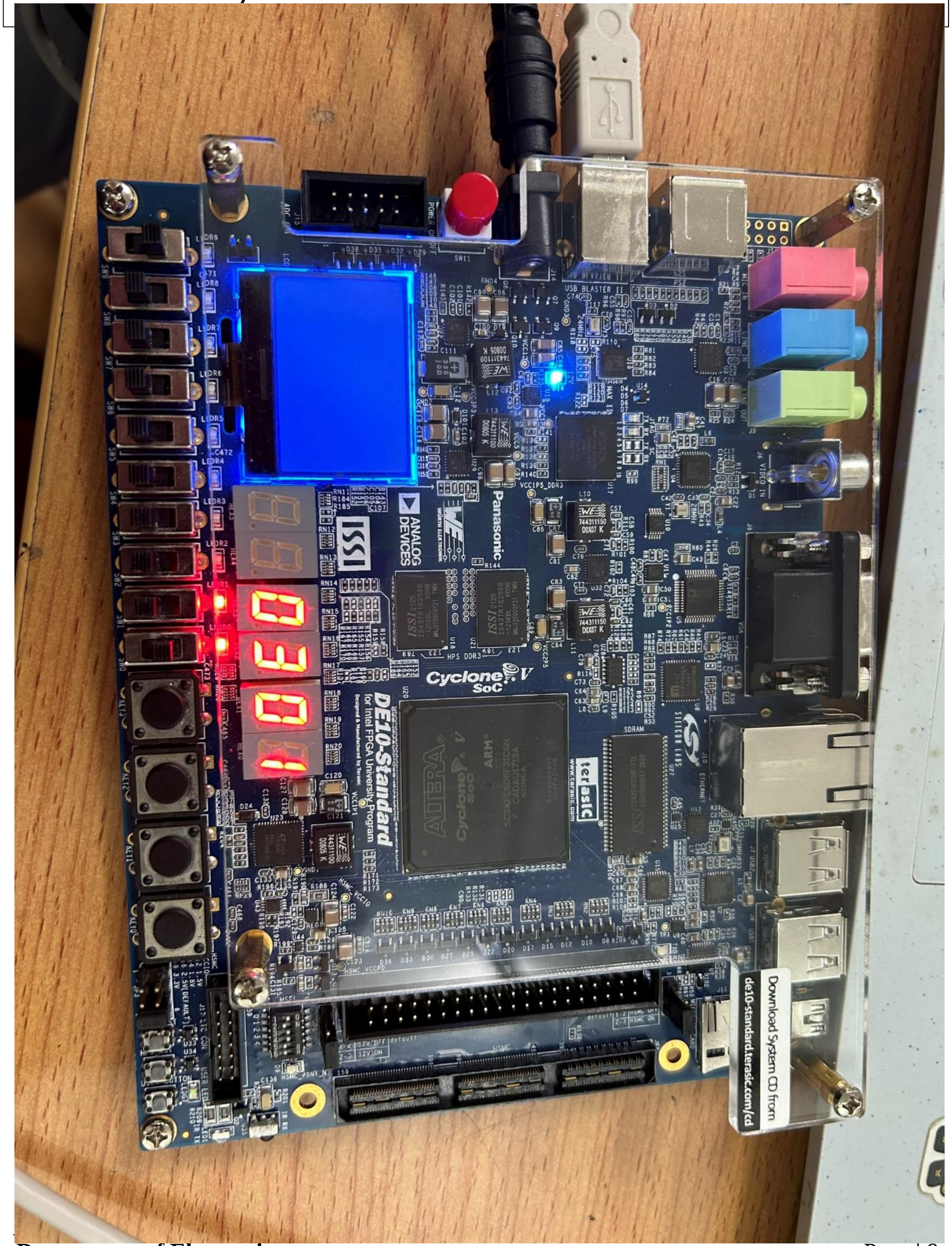
```
module lab1ex1lan2 (
input clk_system,
input [7:0] A,
input rst_n_system,
output [7:0] Sum,
output ca_o,
output ov_o
);
wire [7:0] reg1toadder;
wire [7:0] addertoreg2;
wire [7:0] reg2toadder;
wire ca,ov;
assign Sum = reg2toadder;

register r1 (
 .clk (clk_system),
  . rst_n (rst_n_system),
  .data_in(A),
  .data_out(reg1toadder),
);
register r2 (
.clk (clk_system),
  . rst_n (rst_n_system),
  .data_in(addertoreg2),
  .data_out(reg2toadder)
  );

 adder8bits add5 (
 .A(reg1toadder),
 .B(reg2toadder),
 .Cin(1'b0),
 .Sum(addertoreg2),
 .Cout(ca),
 .V(ov)
 );

 d_ff ff1 (
 .clk(clk_system),
 .rst_n(rst_n_system),
 .D(ov),
 .Q(ov_o)
);
 d_ff ff2 (
 .clk(clk_system),
 .rst_n(rst_n_system),
 .D(ca),
 .Q(ca_o)
 );
 endmodule
```

- Connect input A to switches SW7−0, use KEY0 as an active-low asynchronous reset, and use KEY1 as a manual clock input. The sum from the adder should be displayed on the red lights LEDR7−0, the registered carry signal should be displayed on LEDR8, and the registered overflow signal should be displayed on LEDR9. Show the registered values of A and S as hexadecimal numbers on the 7-segment displays HEX3−2 and HEX1 − 0.



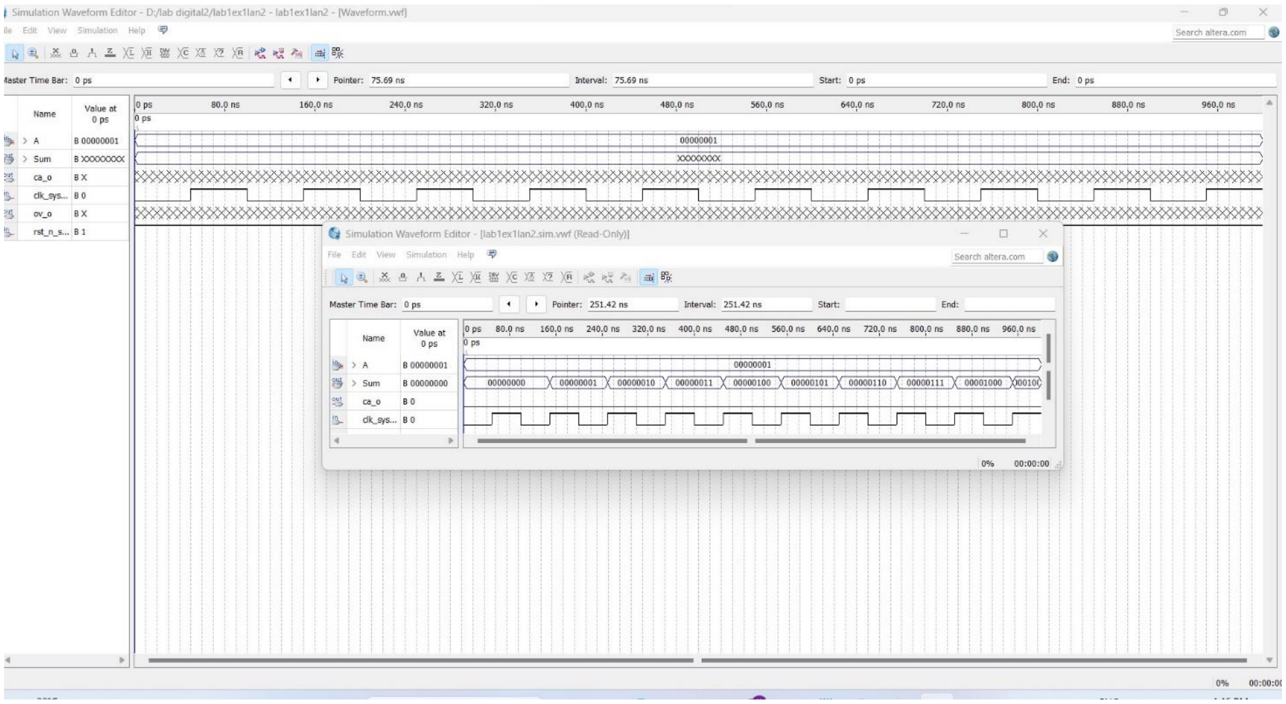
Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

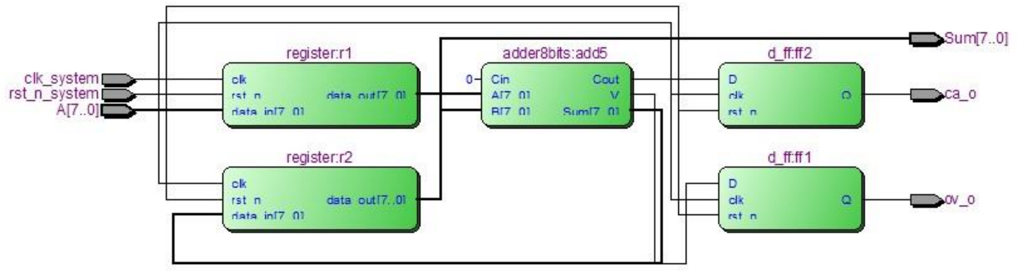


Check: Your report has to show two results:

- The waveform to prove the circuit works correctly. You must show all the cases of overflow flag and carry flag.



- The result of RTL viewer.


clk_system
rst_n_system
A[7..0]
register:r1
clk
rst_n
data_in[7..0]
data_out[7..0]
register:r2
clk
rst_n
data_in[7..0]
data_out[7..0]
adder8bits:add5
0
Cin
A[7..0]
B[7..0]
Cout
V
Sum[7..0]
d_ff:ff2
D
clk
rst_n
Q
d_ff:ff1
D
clk
rst_n
Q
Sum[7..0]
ca_o
ov_o

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

EXPERIMENT 2

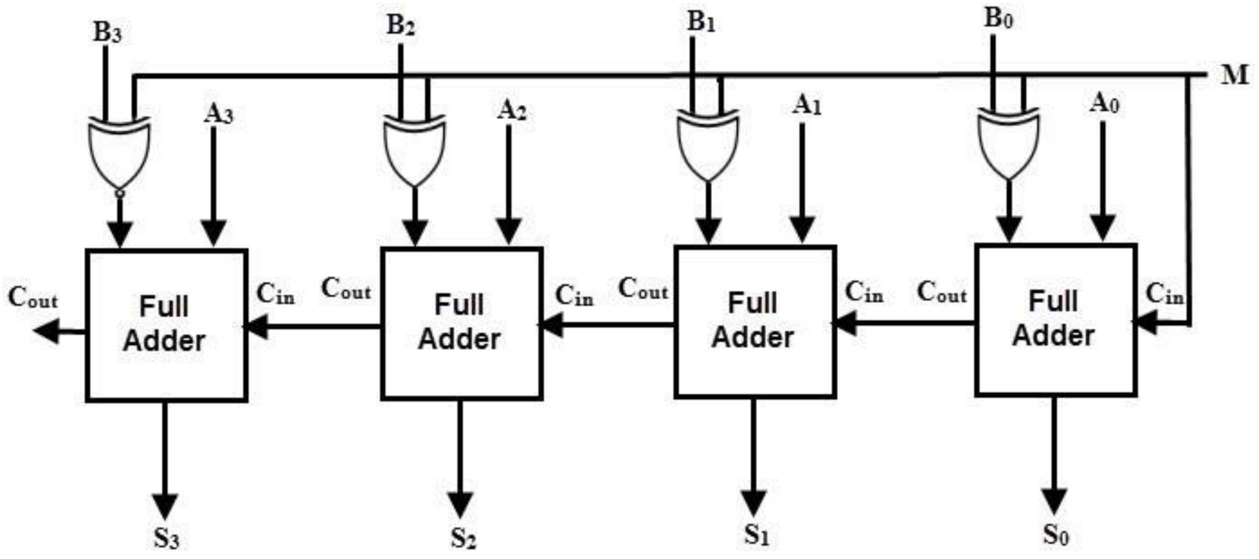
Objective: Known how to program a system to add or subtract the value of an input A to itself repeatedly.

Requirement: Extend the circuit from Experiment 1 to be able to both add and subtract numbers. To do so, add an *add_sub* input to your circuit. When *add_sub* is 1, your circuit should subtract *A* from register *S*, and when *add_sub* is 0 your circuit should add *A* to register *S* as in experiment 1.

Instruction:

- There are two ways to implement that exercise:

  - If the 4-bit adder circuit is constructed from four instances of the FA, using the example 4-bit circuit:



  - If the 4-bit adder entity is written by using a '+' sign in System Verilog, you need to modify the circuit using IF_ ELSE statement.

```
module register8bits (
input clk,
input rst_n,
input [7:0] data_in,
output [7:0] data_out
);

reg [7:0] reg_data;
```

```
always @(posedge clk, negedge rst_n)begin
if (~rst_n)
reg_data <= 8'b0;
else
reg_data <= data_in;
end
assign data_out = reg_data ;
    endmodule


module adder1bit (
input A,
input B,
input Cin,
output Sum,
output Cout);
assign Sum = A^B^Cin;
assign Cout= (A&B | (A^B)&(Cin));
endmodule


module adder8bits (
input [7:0]A,
input [7:0]B,
input Cin,
output [7:0]Sum,
output Cout,
output V,
input M
);
wire [8:0] a;
assign a[0]=M;
//wire MtoCin;
//assign Cin = MtoCin;
//assign a[0] = Cin;
assign V= a[7]^a[8];
assign  Cout = a[8];
genvar i;
generate
for ( i=0; i<8;i=i+1)begin: k1
adder1bit add(
.A(A[i]^Cin),
.B(B[i]),
.Cin(a[i]),
.Sum(Sum[i]),
.Cout(a[i+1])
);
end
endgenerate
    endmodule
module d_ff(
input clk,
input rst_n,
input D,
output Q
);

always @(posedge clk, negedge rst_n)begin
if (~rst_n)
Q <= 1'b0;
else
Q <=D;
end
endmodule
module register8bits (
input clk,
input rst_n,
input [7:0] data_in,
output [7:0] data_out
);

reg [7:0] reg_data;
always @(posedge clk, negedge rst_n)begin
if (~rst_n)
reg_data <= 8'b0;
else
reg_data <= data_in;
end

assign data_out = reg_data ;
endmodule
```

- Connect input A to switches SW7−0, input *add_sub* to switch 9. Use KEY0 as an active-low asynchronous reset, and use KEY1 as a manual clock input. The sum from the adder should be displayed on the red lights LEDR7−0, the registered carry signal should be displayed on LEDR8, and the registered overflow signal should be displayed on LEDR9.

Show the registered values of A and S as hexadecimal numbers on the 7-segment displays HEX3−2 and HEX1 − 0.

Module to display 7 segment LED

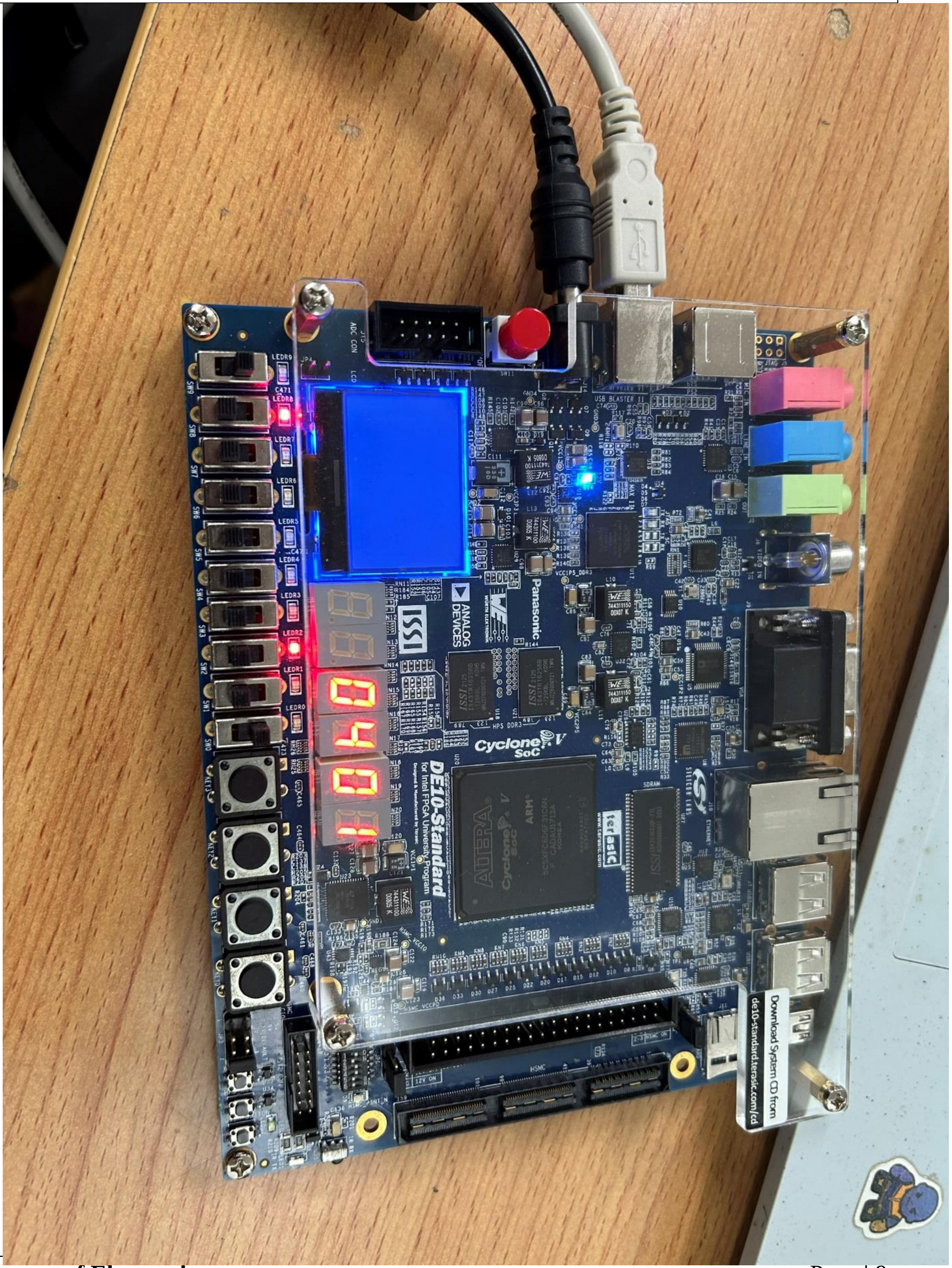
```
module hex_display1 (
  input  [3:0] hex_in,
  output reg [6:0] seg_out
);

  always_comb begin
    case (hex_in)
      0: seg_out = 7'b1000000;
      1: seg_out = 7'b1111001;
      2: seg_out = 7'b0100100;
      3: seg_out = 7'b0110000;
      4: seg_out = 7'b0011001;
      5: seg_out = 7'b0010010;
      6: seg_out = 7'b0000010;
      7: seg_out = 7'b1111000;
      8: seg_out = 7'b0000000;
      9: seg_out = 7'b0011000;
      //10a: seg_out = 7'b0001000;
      //1'hb: seg_out = 7'b0000011;
      //1'hc: seg_out = 7'b0100111;
      //1'hd: seg_out = 7'b0100001;
      //1'he: seg_out = 7'b0000110;
      //1'hf: seg_out = 7'b0001110;
      default: seg_out = 7'b1111111;
    endcase
  end
endmodule
```

Modul wrapper:

```
module lab1ex1lan3 (
input [1:0] KEY,
input [9:0] SW,
output [9:0] LEDR,
output [6:0] HEX1,
output [6:0] HEX0,
output [6:0] HEX2,
output [6:0] HEX3
);
Bai2 khoideptrai(
.clk_system(KEY[1]),
.A(SW[7:0]),
.rst_n_system(KEY[0]),
.Sum(LEDR[7:0]),
.ca_o(LEDR[8]),
.ov_o(LEDR[9]),
.M(SW[9])
);

hex_display1 firstdigit(
.hex_in(SW[3:0]),
.seg_out(HEX0)
  );


hex_display1 seconddigit(
.hex_in(SW[7:4]),
   .seg_out(HEX1)
  );

  hex_display1 firstdigitsum(
.hex_in(LEDR[3:0]),
.seg_out(HEX2)
  );

  hex_display1 seconddigitsum(
.hex_in(LEDR[7:4]),
.seg_out(HEX3)
  );
//assign HEX[1:0]=LEDR[7:0];
//assign HEX[3:2]=SW[7:0];
endmodule
```

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

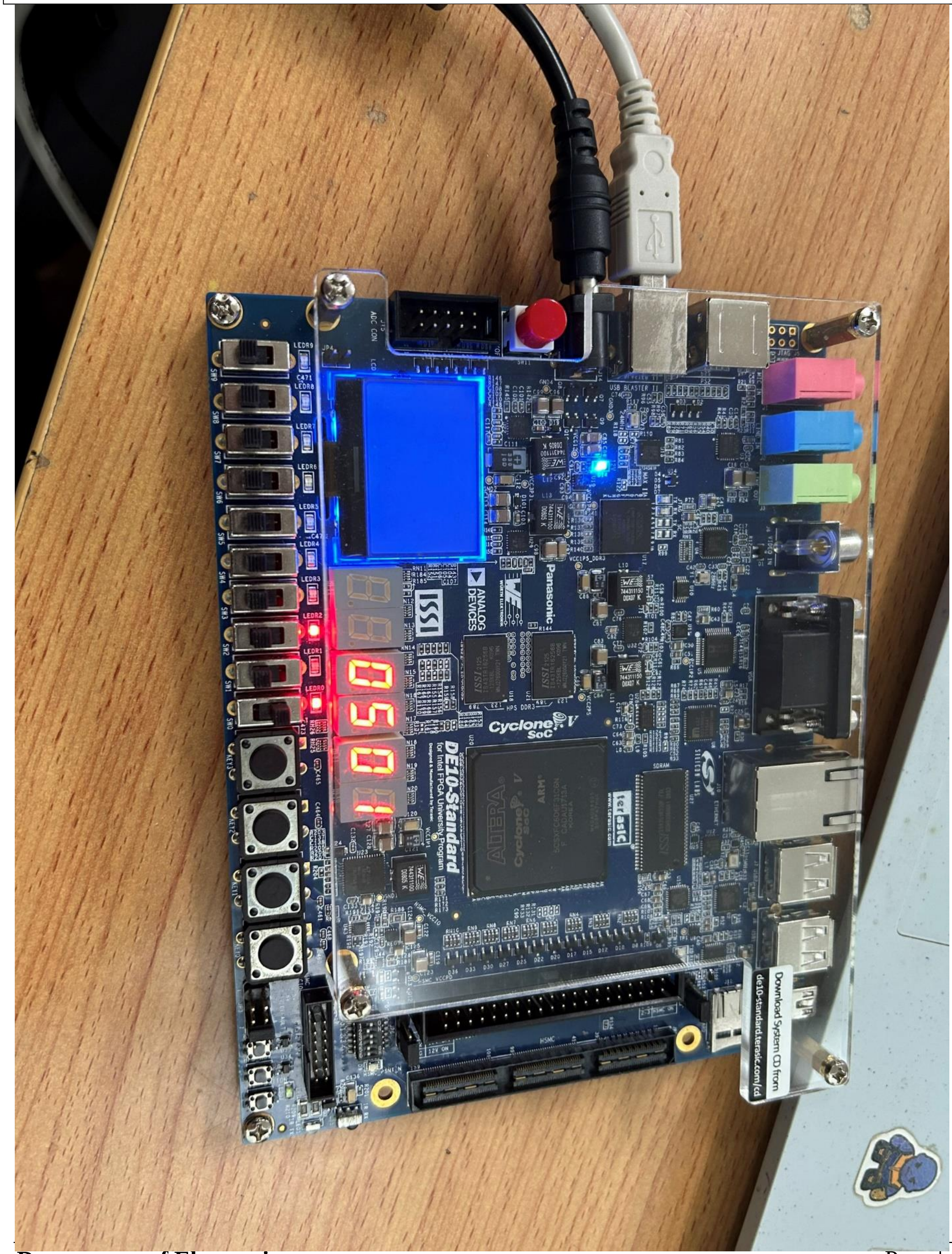


Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

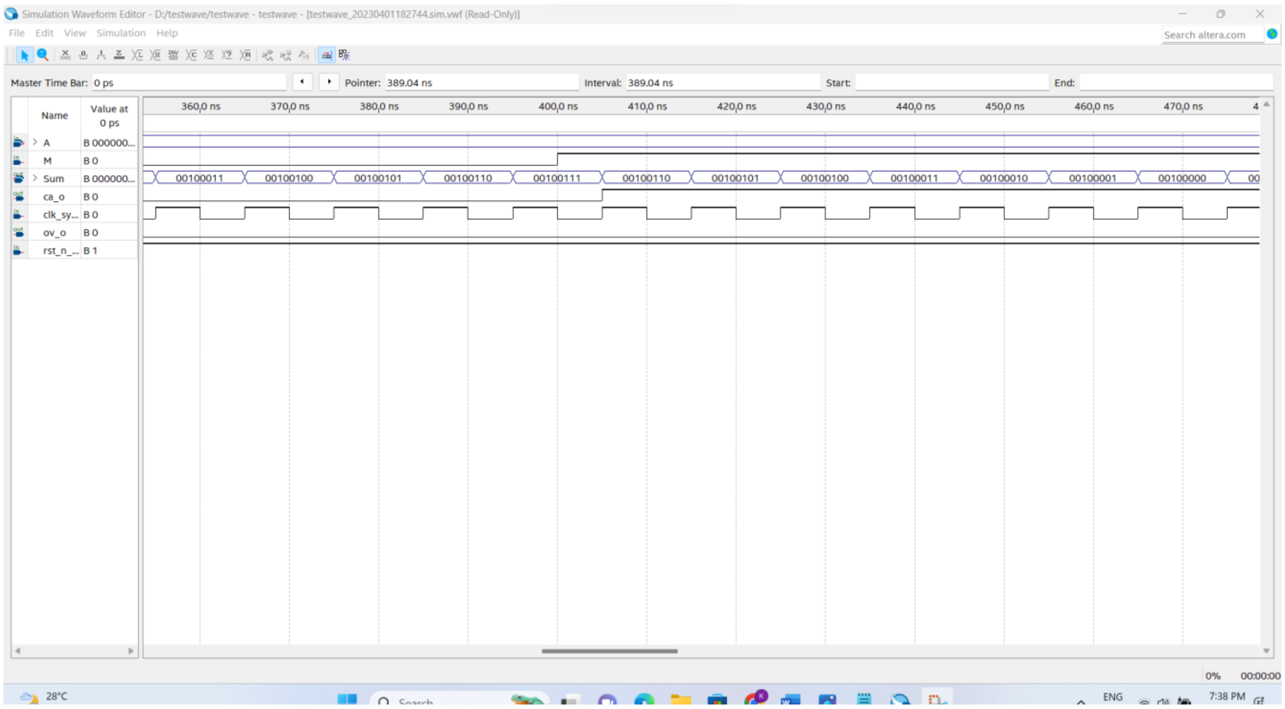


Laboratory 1:

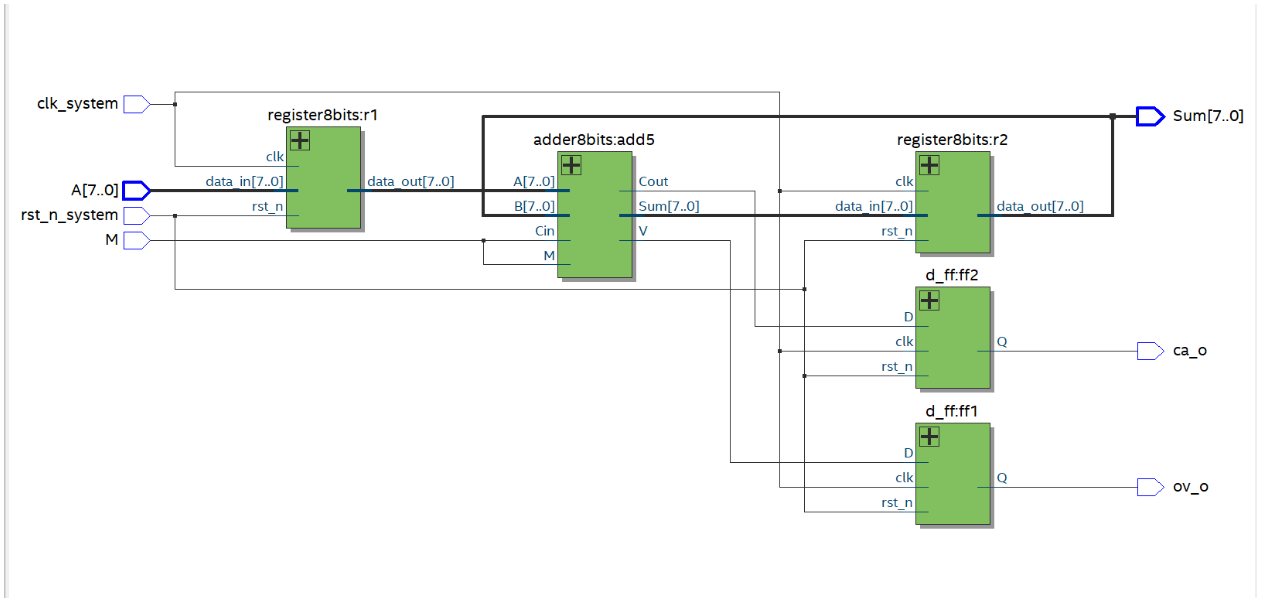
ADDERS, SUBTRACTORS AND MULTIPLIERS

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly. You must show all the cases of overflow flag and carry flag.



- The result of RTL viewer.



Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

EXPERIMENT 3

Objective: Known how to program 8x8 multiplier circuit with registered inputs and outputs.

Requirement: The circuit in Figure 2 shows an 8 x 8 multiplier circuit with registered inputs and outputs. Write System Verilog code to implement this system.

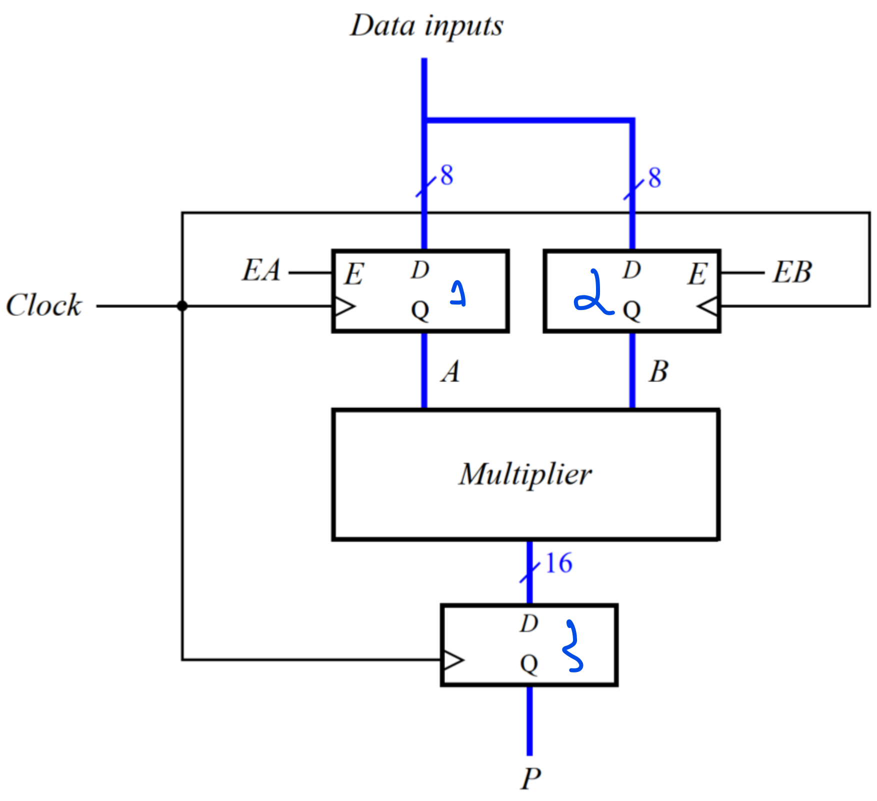


Figure 2: A registered multiplier circuit.

Instruction:

- In exercise 2, pre lab 3, you have write the multiplier. Write System Verilog module that instantiates the instances of this multiplier and three registers to describe the circuit given in Figure 2.

  This code includes 3

  The module register8bits for implement register

module register8bits (

input clk,

```
input rst_n,
input [7:0] data_in,
output [7:0] data_out
);


reg [7:0] reg_data;
always @(posedge clk, negedge rst_n)begin
if (~rst_n)
reg_data <= 8'b0;
else
reg_data <= data_in;
end
assign data_out = reg_data ;
    endmodule
```

The module lab1ex3 to implement a 8x8 mutiplier as well as connecting to registers

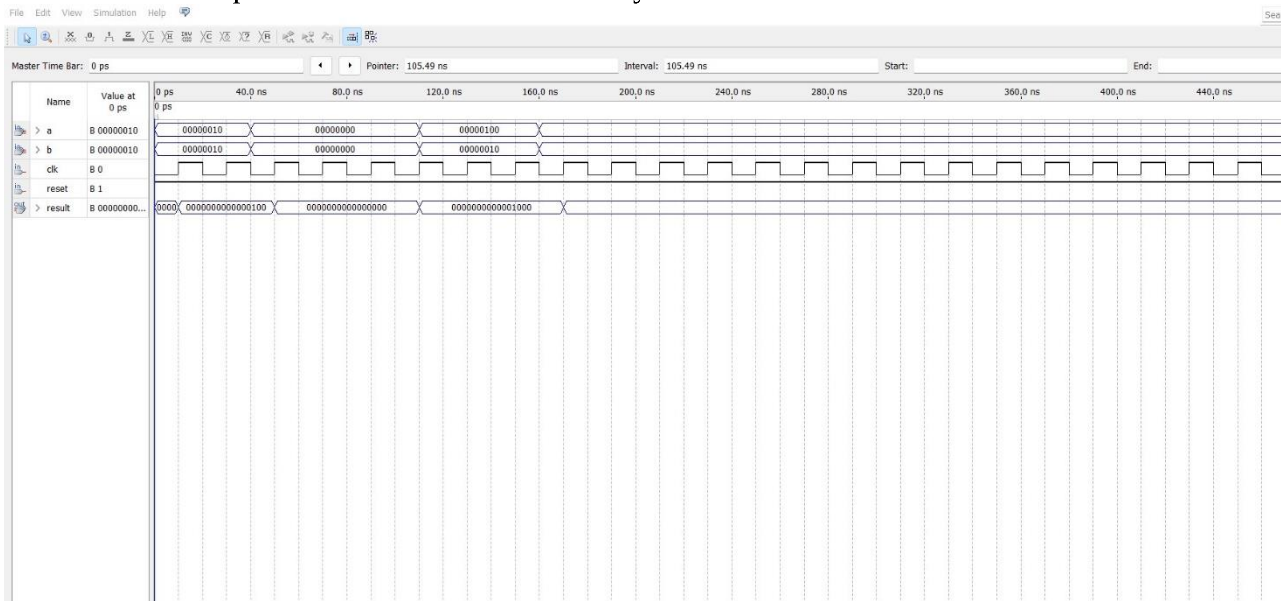
```
module lab1ex3 (
  input logic clk,
  input logic reset,
  input logic signed [7:0] a,
  input logic signed [7:0] b,
  output logic signed [15:0] result
);

  reg [8:0] a_reg;
  reg [8:0] b_reg;

  register8bits r1(
    .clk(clk),
    .rst_n(reset),
    .data_in(a),
    .data_out(a_reg)
  );

  register8bits r2(
    .clk(clk),
    .rst_n(reset),
    .data_in(b),
    .data_out(b_reg)
  );
  reg [15:0] summand [0:7];

  always @* begin
  for (int i = 0; i < 8; i++) begin
  if(b_reg[i]==1)begin
  summand [i]= a_reg << i;
  end else begin summand [i] ='0;
  end
  end
  end

assign result = summand [1] + summand [2] + summand [3] + summand [4] + summand [5]
   + summand [6] + summand [7] + summand [0];
   endmodule
```

- Use switches SW7−0 to provide the data inputs to the circuit. Use SW9 as the enable signal EA for register A, and use SW8 as the enable for register B. When SW9 = 1 display the contents of register A on the red lights LEDR, and display the contents of register B on these lights when SW8 = 1. Use KEY0 as a synchronous reset input, and use KEY1 as a manual clock signal. Show the product P = A×B as a hexadecimal number on the 7-segment displays HEX3-0.

- Using waveform, test the functionality of your design by inputting various data values and observing the generated products.

Check: Your report has to show two results:

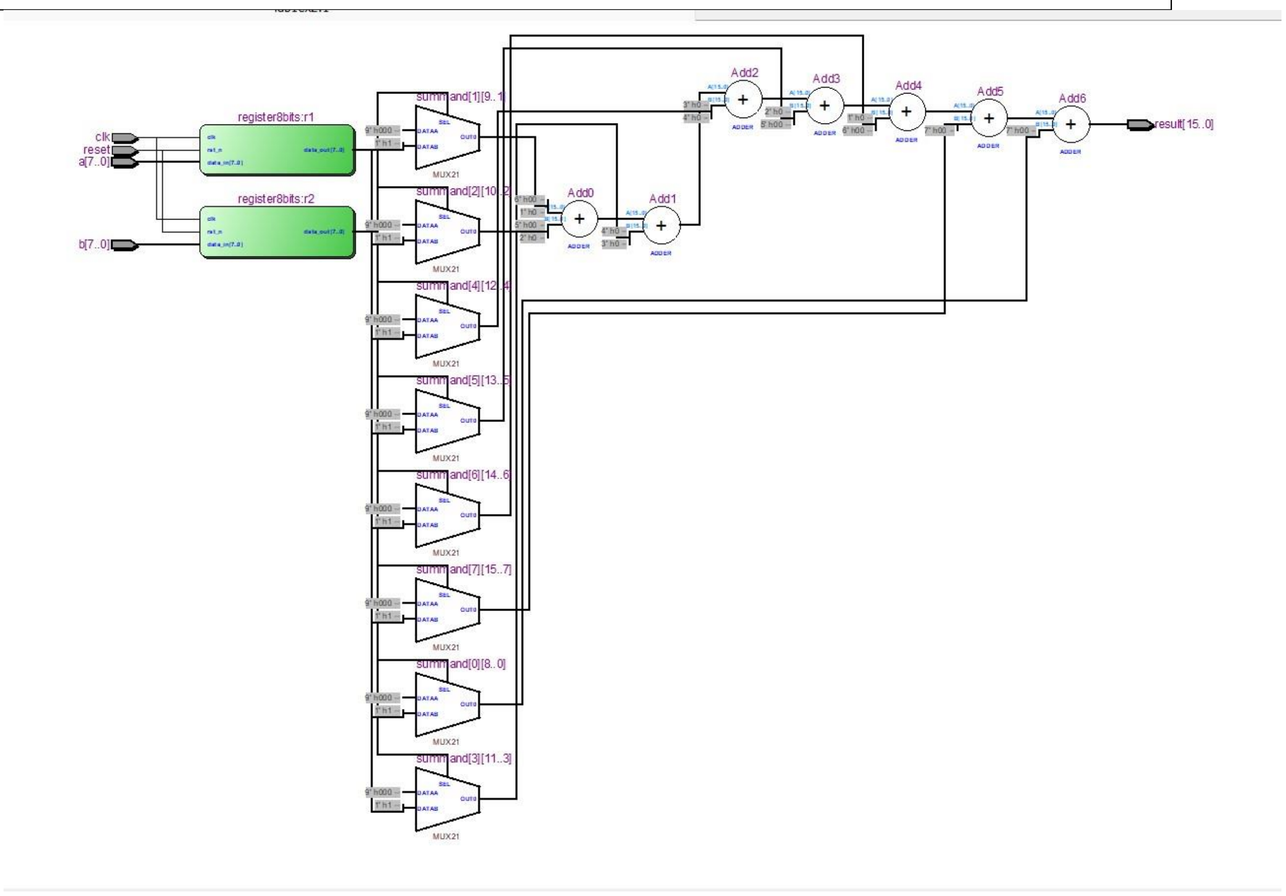
- The waveform to prove the circuit works correctly.



Master Time Bar: 0 ps Pointer: 105.49 ns Interval: 105.49 ns Start: End:

Name	Value at 0 ps
a	B 00000010
b	B 00000010
clk	B 0
reset	B 1
result	B 00000000...

- The result of RTL viewer.


register8bits:r1
register8bits:r2
clk
reset
a[7..0]
b[7..0]
summand[1][9..1]
summand[2][10..2]
summand[4][12..4]
summand[5][13..5]
summand[6][14..6]
summand[7][15..7]
summand[0][8..0]
summand[3][11..3]
MUX21
Add0
Add1
Add2
Add3
Add4
Add5
Add6
ADDER
result[15..0]

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

EXPERIMENT 4

Objective: Known how to program ALU circuit with registered inputs and outputs.

Requirement: The circuit in Figure 3 shows an ALU circuit with registered inputs and outputs. The ALU can implement add, subtract and multiply depending on *Sel* input which is shown in Table 1. Write System Verilog code to implement this system.

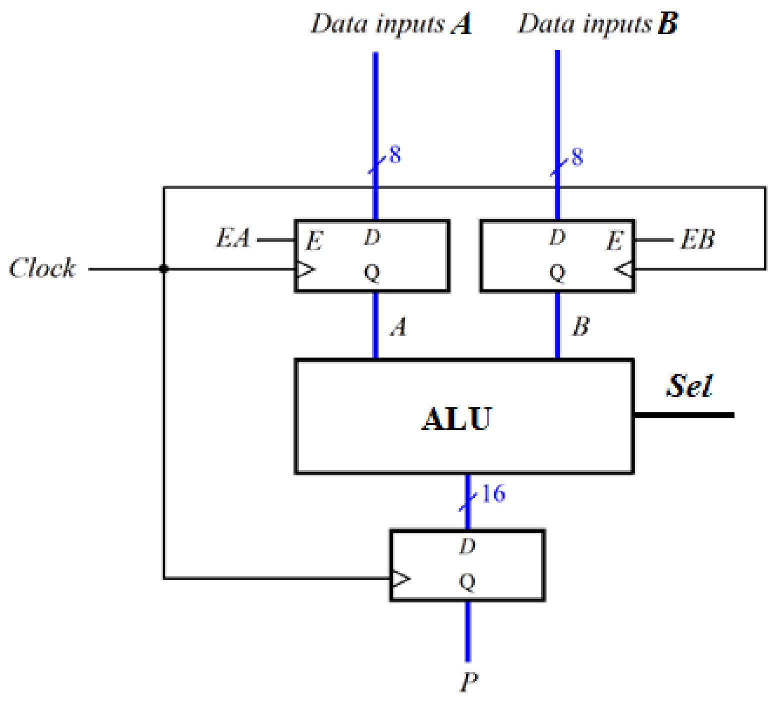


Figure 3: A registered ALU circuit.

Sel Input	ALU operator
000	Add
001	Subtract
010	Multiply

Table 1: Operation of ALU circuit.

Instruction: This circuit has a lot of inputs (22 inputs) then we can not implement this circuit on DE10 standard. Only test the functionality of your design by inputting various data values and observing the generated outputs.

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

This code include 3 main module:

- Module to implement D flip flop

```
module D_ff (
input [7:0] D,
input clk,
input enable,
output [7:0] Q
);
always@(posedge clk)begin
if(enable) begin
Q <=D;
end
end
endmodule
```

- Module ALU to implement ALU

```
module ALU (
input[7:0] A,
input[7:0] B,
input[3:0] Sel,
output [15:0] P
);

always @(*)begin
case (Sel)
0:P = A+B;
1:P = A-B;
2:P = A*B;
default: P=A+B;
endcase
end
endmodule
```

- Module lab1ex3 to connect

```
module lab1ex3l (
input clk,
input [3:0] Sel,
input [7:0] A,
input [7:0] B,
output [15:0] P,
input EA,
input EB
);
wire[15:0] AfftoALU;
wire[15:0] BfftoALU;
wire[15:0] ALUtoPff;
```

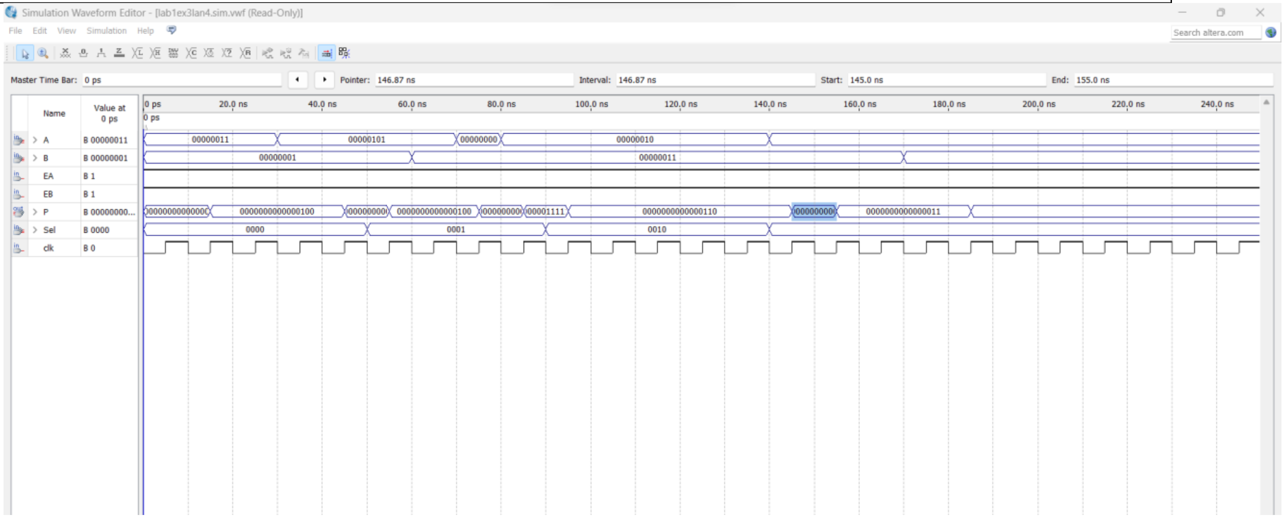
```
D_ff ffA(
.D(A),
.clk(clk),
.enable(EA),
.Q(AfftoALU)
);
D_ff ffB(
.D(B),
.clk(clk),
.enable(EB),
.Q(BfftoALU)
);
ALU (
.A(AfftoALU),
.B(BfftoALU),
.Sel(Sel),
.P(ALUtoPff)
);

D_ff ffP(
.D(ALUtoPff),
.clk(clk),
.Q(P)
);
endmodule
```

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.

Laboratory 1:

ADDERS, SUBTRACTORS AND MULTIPLIERS

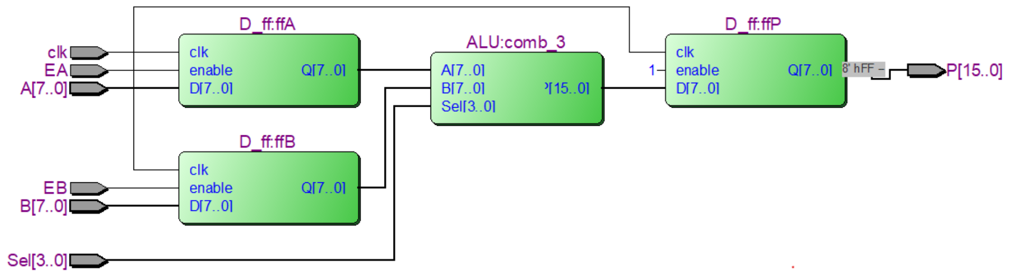


Simulation Waveform Editor - [lab1ex3lan4.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com

Master Time Bar: 0 ps | Pointer: 146.87 ns | Interval: 146.87 ns | Start: 145.0 ns | End: 155.0 ns

Name	Value at 0 ps
A	B 00000011
B	B 00000001
EA	B 1
EB	B 1
P	B 00000000...
Sel	B 0000
clk	B 0

➢ The result of RTL viewer.



clk
EA
A[7..0]
EB
B[7..0]
Sel[3..0]
D_ff:ffA
clk
enable
D[7..0]
Q[7..0]
D_ff:ffB
clk
enable
D[7..0]
Q[7..0]
ALU:comb_3
A[7..0]
B[7..0]
Sel[3..0]
P[15..0]
D_ff:ffP
1
clk
enable
D[7..0]
Q[7..0]
8' hFF
P[15..0]