MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

MIDDLE EAST TECHNICAL UNIVERSITY NORTHERN CYPRUS CAMPUS

ELECTRICAL AND ELECTRONICS ENGINEERING PROGRAM

EEE 446 Computer Architecture

Lab Module 2

8-bit INTEGER ARITHMETIC PROCESSOR DATAPATH DESIGN

Name, Surname	Doğukan Fikri Arat
Student ID	2079648
Name, Surname	Saad Yousaf
Student ID	2246536

Contents

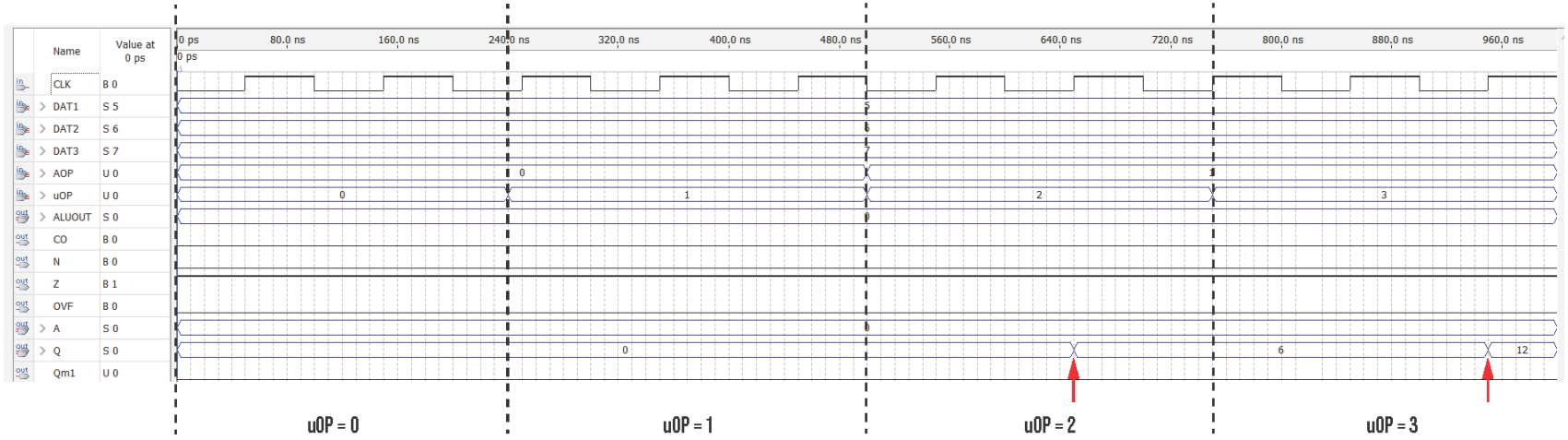
Objective

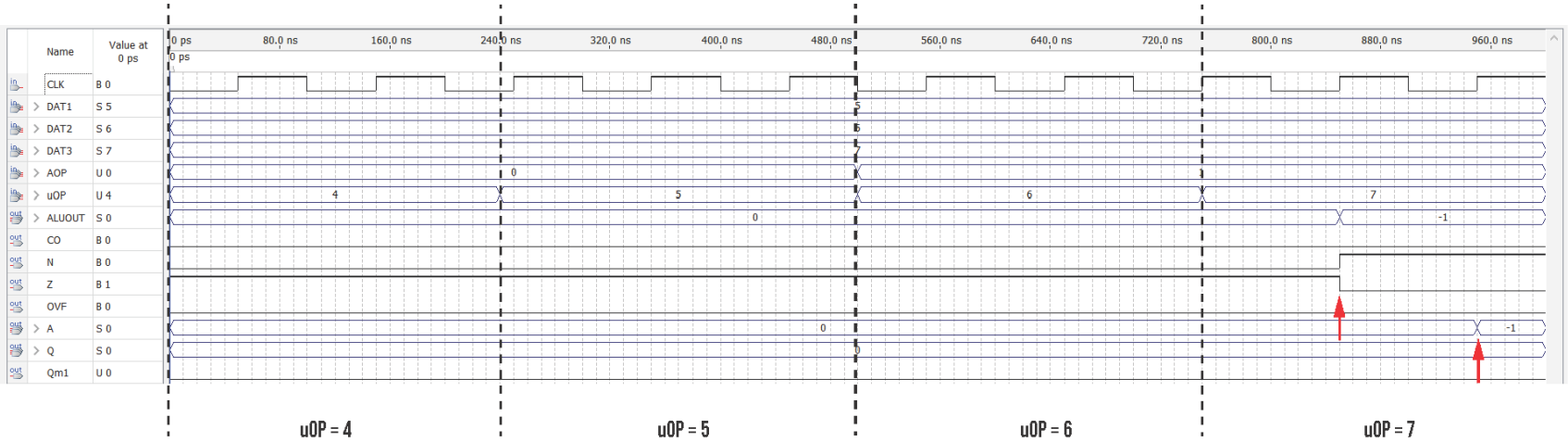
In this lab module, we were responsible for designing an 8-bit arithmetic processor (AP) and a control unit to implement the microinstructions associated with this arithmetic processor. The 8-bit arithmetic processor can do all the required operations specified by the lab manual. The control unit executes the micro-operations according to the uOP code assigned to each instruction. This will then be used later to build our multicycle machine after the ISA is designed to carry out the operations in the benchmark specifications. The booth's multiplier that was developed in the first lab module has been integrated into this AP datapath by doing some minor modifications to the AP datapath. All the operations and functionality of the datapath has been shown and verified in the report below.

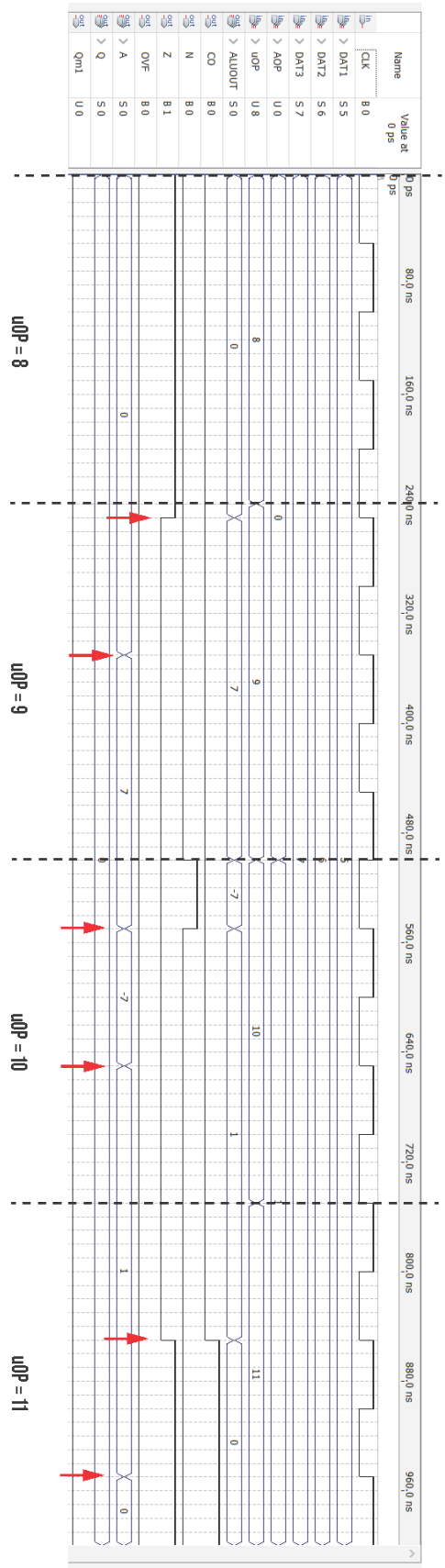
AP Datapath Simulations

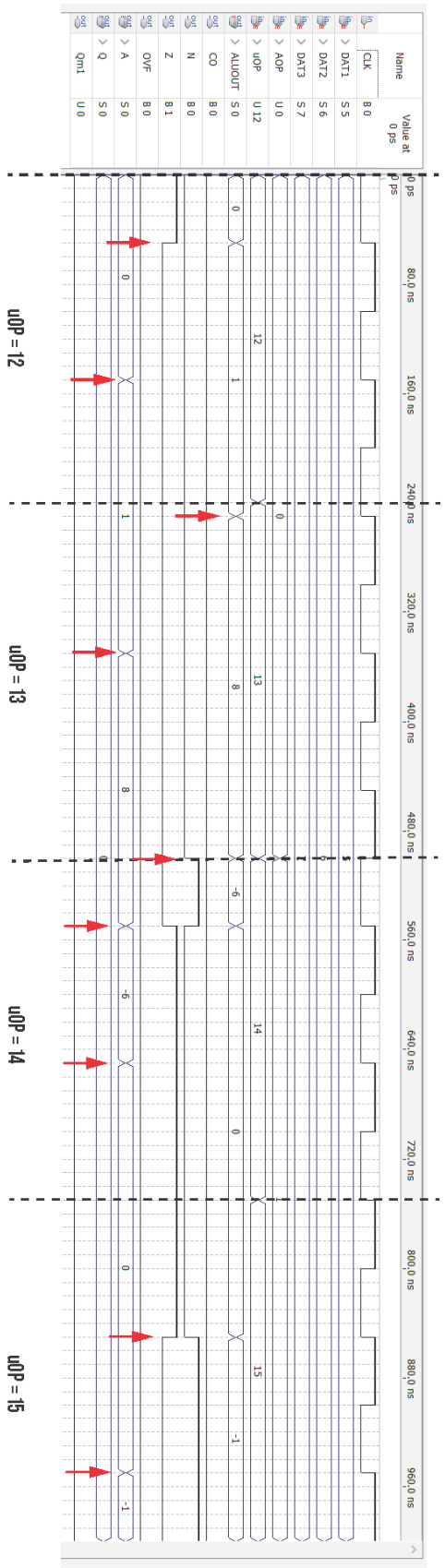
In this part of the report, simulation screenshots are shown. In the simulations, all the micro-operations are specified and changes are emphasized with red arrow on the simulations. Beside of the micro-operation in the experiment handouts, multiplier control sequence was also added to the control unit and the simulation.

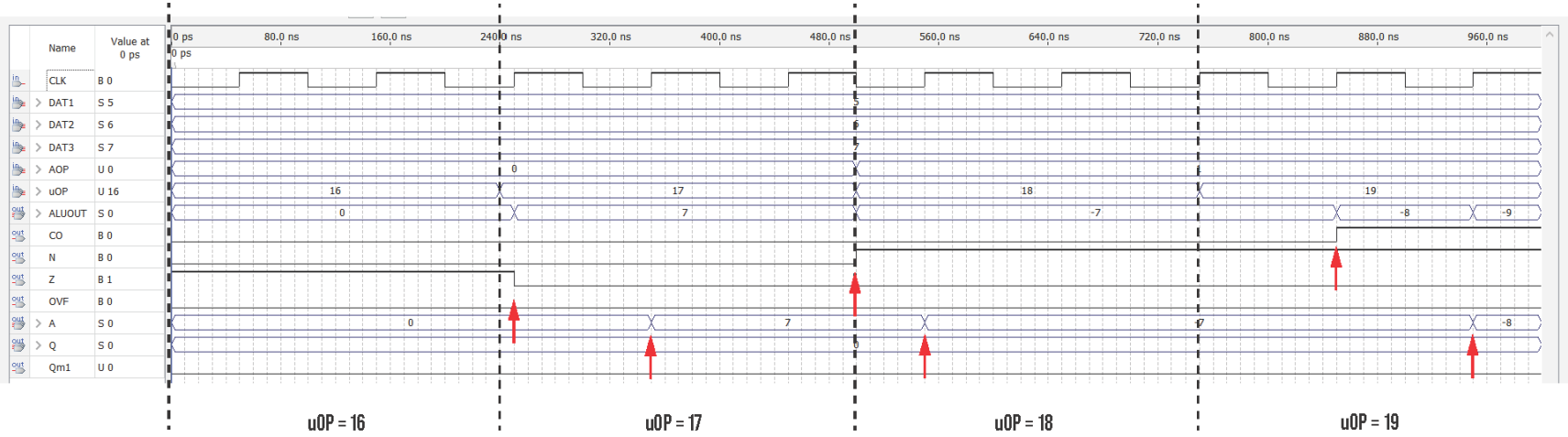
Red arrow indicates the change

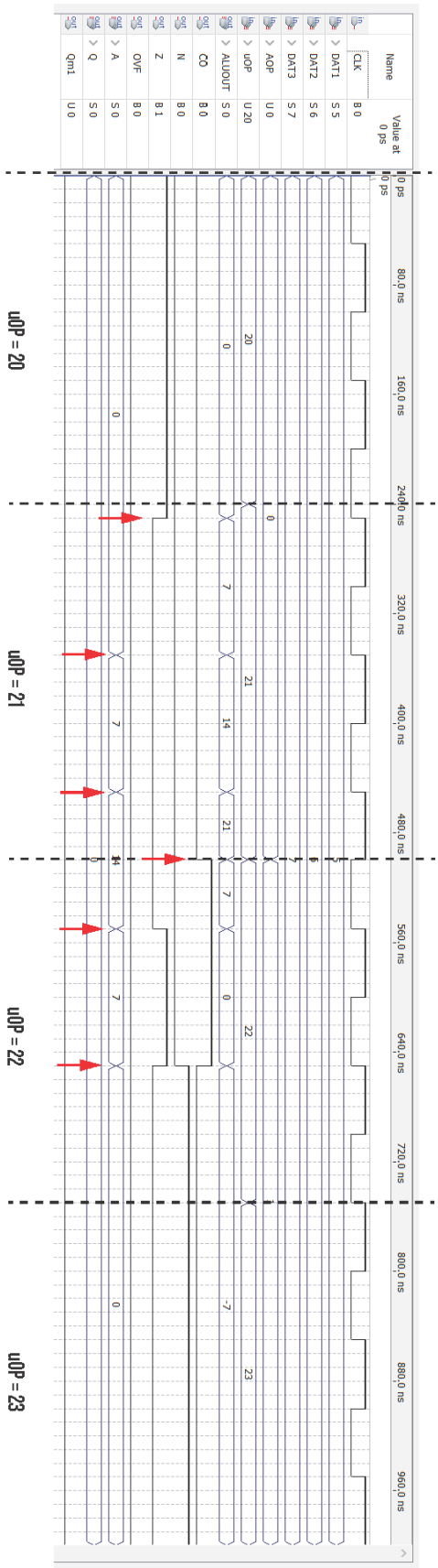


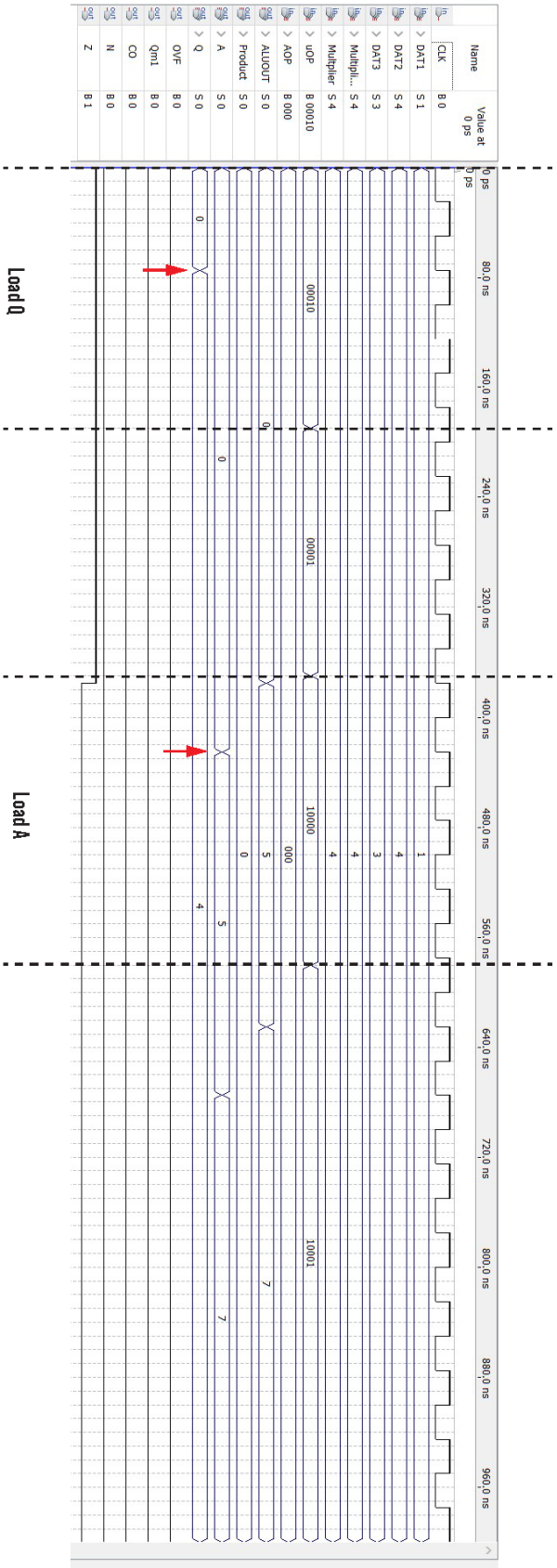

Name
Value at 0 ps
CLK B 0
DAT1 S 5
DAT2 S 6
DAT3 S 7
AOP U 0
uOP U 4
ALUOUT S 0
CO B 0
N B 0
Z B 1
OVF B 0
A S 0
Q S 0
Qm1 U 0
0 ps
80,0 ns
160,0 ns
240,0 ns
320,0 ns
400,0 ns
480,0 ns
560,0 ns
640,0 ns
720,0 ns
800,0 ns
880,0 ns
960,0 ns
uOP = 4
uOP = 5
uOP = 6
uOP = 7


Name
Value at 0 ps
CLK
DAT1
DAT2
DAT3
AOP
uOP
ALUOUT
CO
N
Z
OVF
A
Q
Qm1
uOP = 8
uOP = 9
uOP = 10
uOP = 11

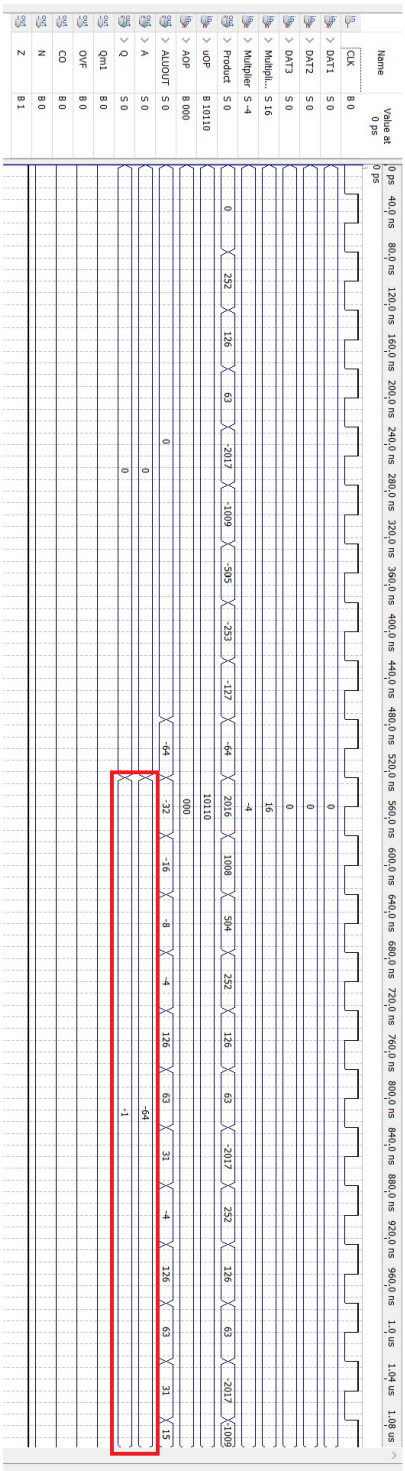

Name
Value at 0 ps
CLK
DAT1
DAT2
DAT3
AOP
uOP
ALUOUT
CO
N
Z
OVF
A
Q
Qm1
uOP = 12
uOP = 13
uOP = 14
uOP = 15


Name
Value at 0 ps
CLK B 0
DAT1 S 5
DAT2 S 6
DAT3 S 7
AOP U 0
uOP U 16
ALUOUT S 0
CO B 0
N B 0
Z B 1
OVF B 0
A S 0
Q S 0
Qm1 U 0
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
uOP = 16
uOP = 17
uOP = 18
uOP = 19

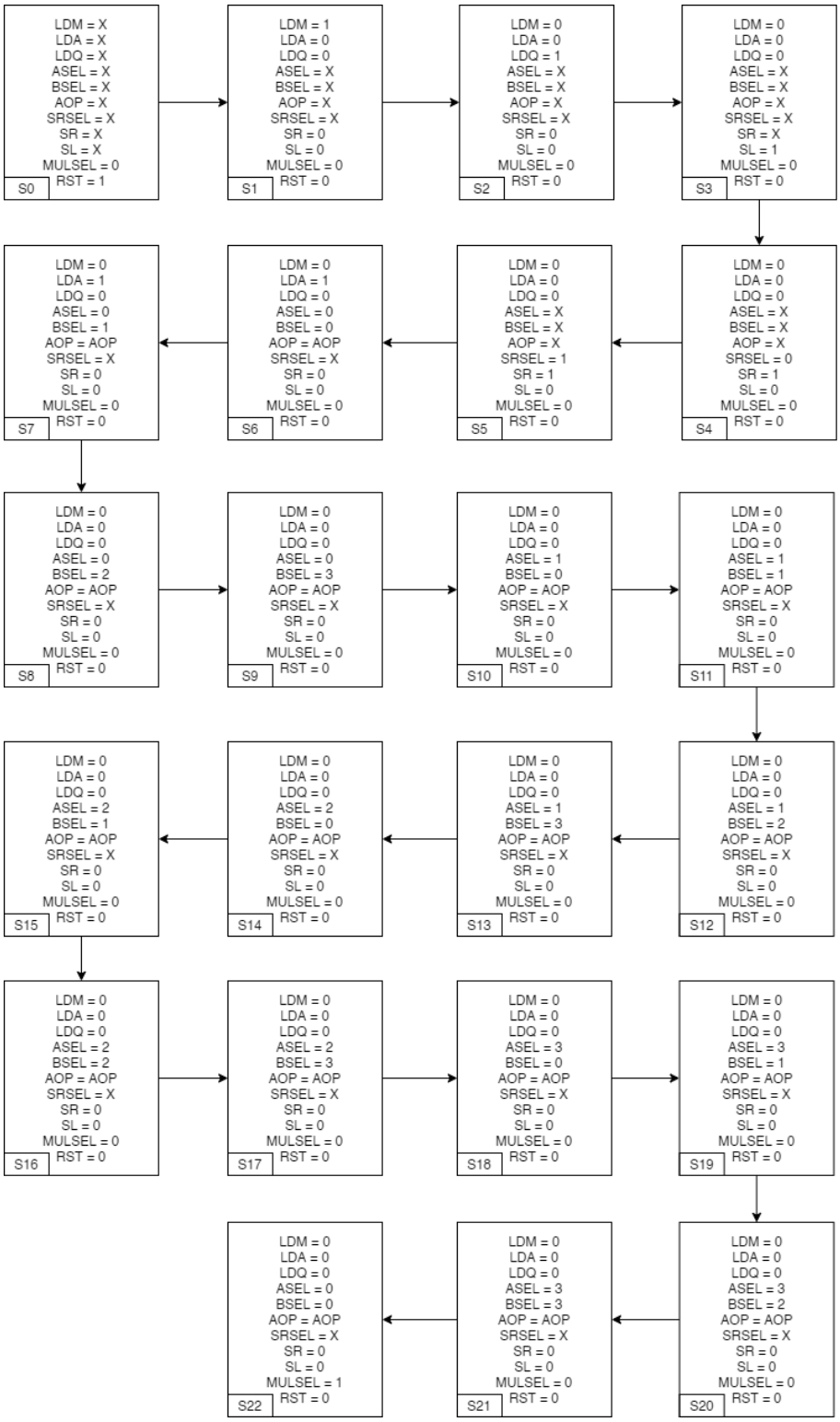

uOP = 20
uOP = 21
uOP = 22
uOP = 23


Load Q
Load A

Multiplication Simulation



State Diagram

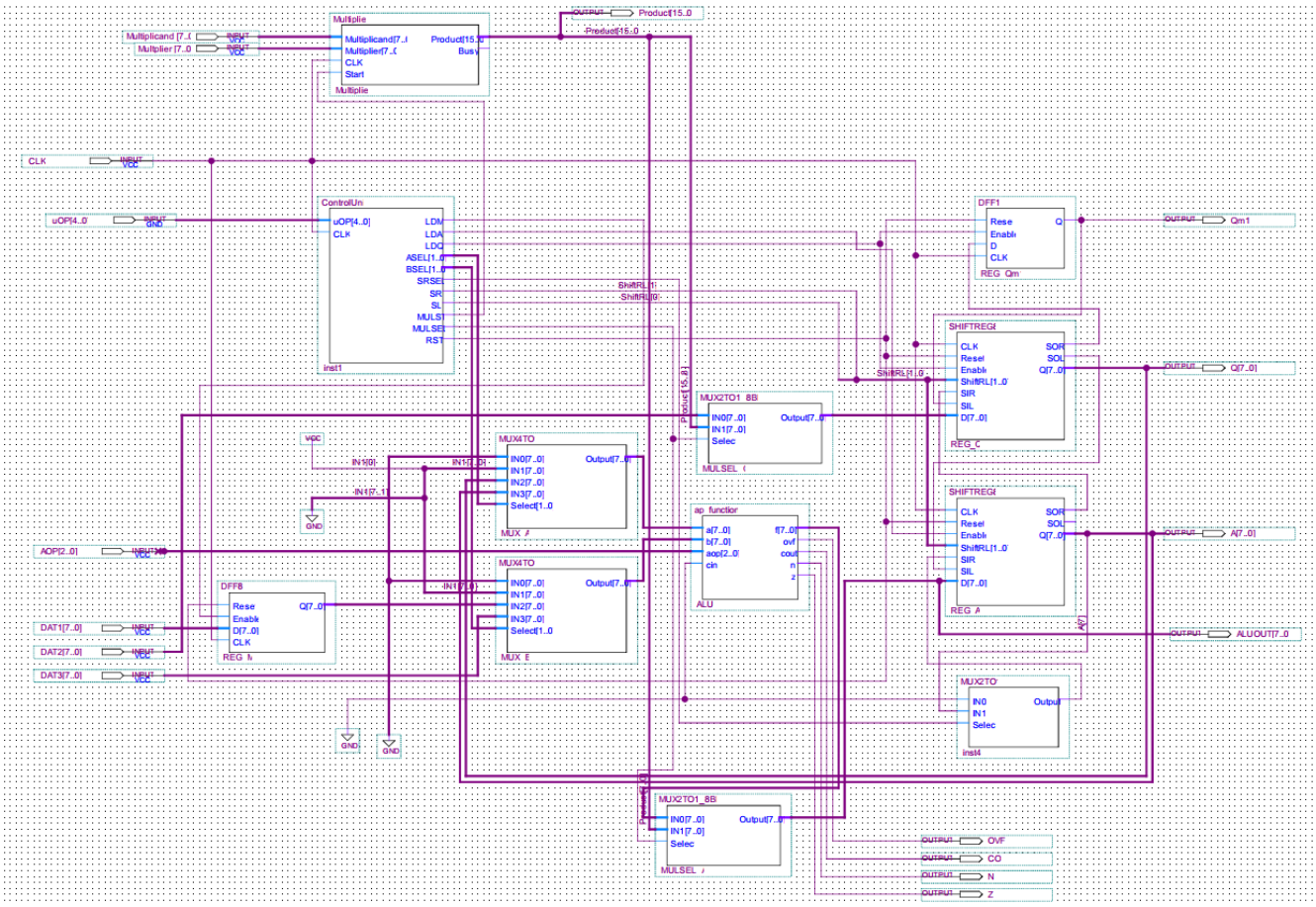


Timing Analyzer Summary

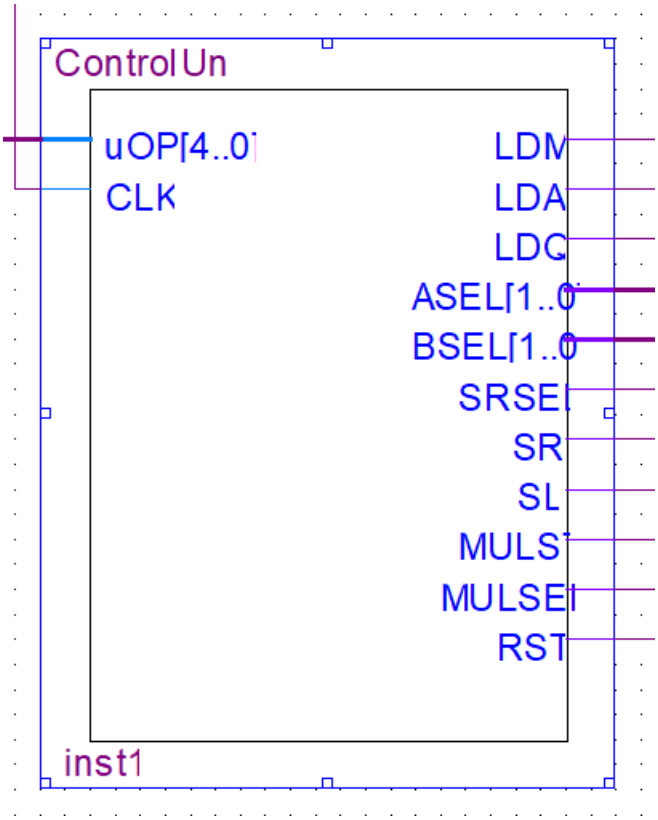
Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	170.36 MHz	170.36 MHz	CLK	
2	405.52 MHz	405.52 MHz	AOP[0]	

Schematic Design Files

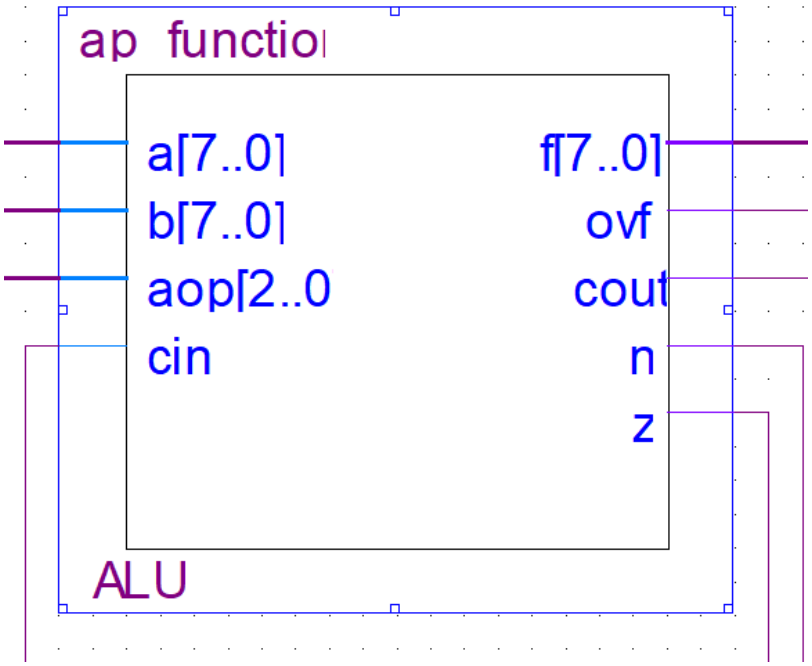
- Datapath



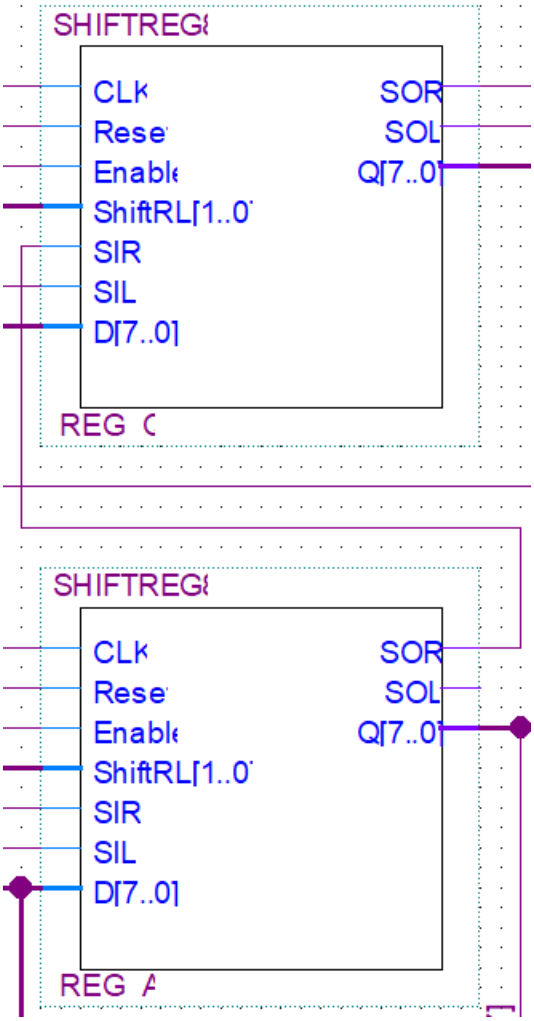
- Control unit



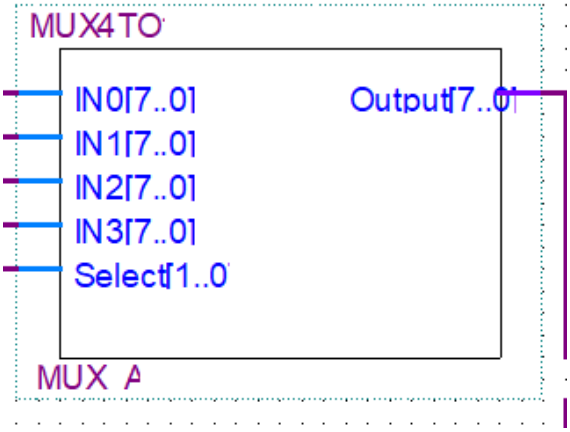
- Arithmetic Processor



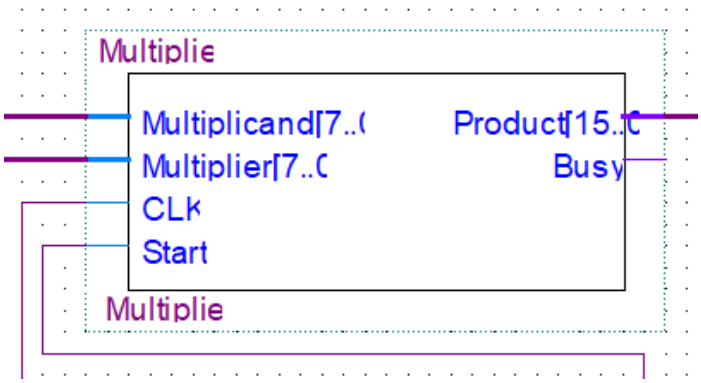
- Register A and Q



- 4 to 1 MUX



- Booth Multiplier



Verilog Source Codes

- Control Unit

```
module ControlUnit(LDM, LDA, LDQ, ASEL, BSEL, SRSEL, SR, SL, MULST, MULSEL,
RST, uOP, CLK);

output reg LDM, LDA, LDQ, SRSEL, SR, SL, MULST, MULSEL, RST;
output reg [1:0] ASEL, BSEL;

input CLK;
input [4:0] uOP;

reg [3:0] Count;

initial
begin
	RST <= 1'b1;
	Count <= 4'b0;
	MULSEL <= 1'b0;
end

always @(posedge CLK)
begin
	case(uOP)
		5'b00000:
			RST <= 1'b1;

		5'b00001:
		begin
			LDM <= 1'b1;
			LDA <= 1'b0;
			LDQ <= 1'b0;
			SR <= 1'b0;
			SL <= 1'b0;
```

```
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b00010:
begin
        LDM <= 1'b0;
        LDA <= 1'b0;
        LDQ <= 1'b1;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b00011:
begin
        LDM <= 1'b0;
        LDA <= 1'b0;
        LDQ <= 1'b0;
        SL <= 1'b1;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b00100:
begin
        LDM <= 1'b0;
        LDA <= 1'b0;
        LDQ <= 1'b0;
        SR <= 1'b1;
        SL <= 1'b0;
        SRSEL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b00101:
begin
        LDM <= 1'b0;
        LDA <= 1'b0;
        LDQ <= 1'b0;
        SR <= 1'b1;
        SL <= 1'b0;
        SRSEL <= 1'b1;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b00110:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        SR <= 1'b0;
        SL <= 1'b0;
```

```
        ASEL <= 2'b0;
        BSEL <= 2'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b00111:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b0;
        BSEL <= 2'b01;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b01000:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b0;
        BSEL <= 2'b10;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b01001:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b0;
        BSEL <= 2'b11;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b01010:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b01;
        BSEL <= 2'b0;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
```

```
        MULSEL <= 1'b0;
end
5'b01011:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b01;
        BSEL <= 2'b01;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b01100:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b01;
        BSEL <= 2'b10;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b01101:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b01;
        BSEL <= 2'b11;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b01110:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b10;
        BSEL <= 2'b0;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b01111:
```

```
begin
	LDM <= 1'b0;
	LDA <= 1'b1;
	LDQ <= 1'b0;
	ASEL <= 2'b10;
	BSEL <= 2'b01;
	SR <= 1'b0;
	SL <= 1'b0;
	RST <= 1'b0;
	MULSEL <= 1'b0;
end
5'b10000:
begin
	LDM <= 1'b0;
	LDA <= 1'b1;
	LDQ <= 1'b0;
	ASEL <= 2'b10;
	BSEL <= 2'b10;
	SR <= 1'b0;
	SL <= 1'b0;
	RST <= 1'b0;
	MULSEL <= 1'b0;
end
5'b10001:
begin
	LDM <= 1'b0;
	LDA <= 1'b1;
	LDQ <= 1'b0;
	ASEL <= 2'b10;
	BSEL <= 2'b11;
	SR <= 1'b0;
	SL <= 1'b0;
	RST <= 1'b0;
	MULSEL <= 1'b0;
end
5'b10010:
begin
	LDM <= 1'b0;
	LDA <= 1'b1;
	LDQ <= 1'b0;
	ASEL <= 2'b11;
	BSEL <= 2'b0;
	SR <= 1'b0;
	SL <= 1'b0;
	RST <= 1'b0;
	MULSEL <= 1'b0;
end
5'b10011:
begin
	LDM <= 1'b0;
	LDA <= 1'b1;
```

```
        LDQ <= 1'b0;
        ASEL <= 2'b11;
        BSEL <= 2'b01;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b10100:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b11;
        BSEL <= 2'b10;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b10101:
begin
        LDM <= 1'b0;
        LDA <= 1'b1;
        LDQ <= 1'b0;
        ASEL <= 2'b11;
        BSEL <= 2'b11;
        SR <= 1'b0;
        SL <= 1'b0;
        RST <= 1'b0;
        MULSEL <= 1'b0;
end
5'b10110:
begin

        if (Count == 4'b0) begin
                MULST <= 1'b1;
        end else if (Count == 4'b0001) begin
                MULST <= 1'b0;
        end else if (Count == 4'b1001) begin
                LDA <= 1'b1;
                LDQ <= 1'b1;
                MULSEL <= 1'b1;
                RST <= 1'b0;
        end else begin
                RST <= 1'b0;
                LDA <= 1'b0;
                LDQ <= 1'b0;
        end

        Count <= Count + 1;
```

```
            end

            default:
                  RST <= 1'b1;
      endcase
end
endmodule
```

- Multiplexer 2 to 1 (1 Bits)

```
module MUX2TO1_8BIT (IN0, IN1, Select, Output);

input [7:0] IN0, IN1;
input Select ;
output reg [7:0] Output;

always @ *
      case (Select)
            1'b0   : Output <= IN0;
            1'b1   : Output <= IN1;
      endcase

endmodule
```

- Shift Register (8 Bits)

```
module SHIFTREG8(CLK, Reset, Enable, ShiftRL, SIR, SIL, SOR, SOL, D, Q);

input CLK, Enable, Reset;

input [1:0] ShiftRL;

input SIR, SIL;
output SOR, SOL;

wire SOR, SOL;

input [7:0] D;
output reg [7:0] Q;

always @(posedge CLK)
begin
      if (Reset)
            Q <= 8'b0;
      else
            if (Enable)
                  Q <= D;
            else if (ShiftRL == 2'b10) // Shift Right
                  {Q} <= {SIR, Q[7:1]};
            else if (ShiftRL == 2'b01) // Shift Left
```

```
                    {Q} <= {Q[6:0], SIL};
end

assign SOR = Q[0];
assign SOL = Q[7];

endmodule
```

- Arithmetic Processor

```
module ap_function (a, b, aop, f, ovf, cin, cout, n, z);
  input [7:0] a, b;
  input cin;
  input  [2:0] aop;
  output [7:0] f;
  output ovf, cout, n, z;
  reg z;
  reg [7:0] temp, carry, neg_a, neg_b;
  always @*
  begin
       neg_a = ~a+1;
       neg_b = ~b+1;


    case (aop)
      3'b000   :
                    begin

                            temp [0] <= (a[0] ^ b[0] ^ cin);
                            carry[0] <= ((a[0] & b[0]) | ((a[0] ^ b[0]) & cin)
);

                            temp [1] <= (a[1] ^ b[1] ^ carry[0]);
                            carry[1] <= ((a[1] & b[1]) | ((a[1] ^ b[1]) &
carry[0]));

                            temp [2] <= (a[2] ^ b[2] ^ carry[1]);
                            carry[2] <= ((a[2] & b[2]) | ((a[2] ^ b[2]) &
carry[1]));

                            temp [3] <= (a[3] ^ b[3] ^ carry[2]);
                            carry[3] <= ((a[3] & b[3]) | ((a[3] ^ b[3]) &
carry[2]));

                            temp [4] <= (a[4] ^ b[4] ^ carry[3]);
                            carry[4] <= ((a[4] & b[4]) | ((a[4] ^ b[4]) &
carry[3]));

                            temp [5] <= (a[5] ^ b[5] ^ carry[4]);
```

```
                        carry[5] <= ((a[5] & b[5]) | ((a[5] ^ b[5]) &
carry[4]));

                        temp [6] <= (a[6] ^ b[6] ^ carry[5]);
                        carry[6] <= ((a[6] & b[6]) | ((a[6] ^ b[6]) &
carry[5]));

                        temp [7] <= (a[7] ^ b[7] ^ carry[6]);
                        carry[7] <= ((a[7] & b[7]) | ((a[7] ^ b[7]) &
carry[6]));

                end


            3'b001   :
                  begin

                       temp [0] <= (a[0] ^ neg_b[0] ^ cin);
                        carry[0] <= ((a[0] & neg_b[0]) | ((a[0] ^
neg_b[0]) & cin) );

                        temp [1] <= (a[1] ^ neg_b[1] ^ carry[0]);
                        carry[1] <= ((a[1] & neg_b[1]) | ((a[1] ^
neg_b[1]) & carry[0]));

                        temp [2] <= (a[2] ^ neg_b[2] ^ carry[1]);
                        carry[2] <= ((a[2] & neg_b[2]) | ((a[2] ^
neg_b[2]) & carry[1]));

                        temp [3] <= (a[3] ^ neg_b[3] ^ carry[2]);
                        carry[3] <= ((a[3] & neg_b[3]) | ((a[3] ^
neg_b[3]) & carry[2]));

                        temp [4] <= (a[4] ^ neg_b[4] ^ carry[3]);
                        carry[4] <= ((a[4] & neg_b[4]) | ((a[4] ^
neg_b[4]) & carry[3]));

                        temp [5] <= (a[5] ^ neg_b[5] ^ carry[4]);
                        carry[5] <= ((a[5] & neg_b[5]) | ((a[5] ^
neg_b[5]) & carry[4]));

                        temp [6] <= (a[6] ^ neg_b[6] ^ carry[5]);
                        carry[6] <= ((a[6] & neg_b[6]) | ((a[6] ^
neg_b[6]) & carry[5]));

                        temp [7] <= (a[7] ^ neg_b[7] ^ carry[6]);
                        carry[7] <= ((a[7] & neg_b[7]) | ((a[7] ^
neg_b[7]) & carry[6]));

                end
```

```
            3'b010   :
                  begin

                       temp [0] <= (neg_a[0] ^ b[0] ^ cin);
                          carry[0] <= ((neg_a[0] & b[0]) | ((neg_a[0] ^
b[0]) & cin) );

                          temp [1] <= (neg_a[1] ^ b[1] ^ carry[0]);
                          carry[1] <= ((neg_a[1] & b[1]) | ((neg_a[1] ^
b[1]) & carry[0]));

                          temp [2] <= (neg_a[2] ^ b[2] ^ carry[1]);
                          carry[2] <= ((neg_a[2] & b[2]) | ((neg_a[2] ^
b[2]) & carry[1]));

                          temp [3] <= (neg_a[3] ^ b[3] ^ carry[2]);
                          carry[3] <= ((neg_a[3] & b[3]) | ((neg_a[3] ^
b[3]) & carry[2]));

                          temp [4] <= (neg_a[4] ^ b[4] ^ carry[3]);
                          carry[4] <= ((neg_a[4] & b[4]) | ((neg_a[4] ^
b[4]) & carry[3]));

                          temp [5] <= (neg_a[5] ^ b[5] ^ carry[4]);
                          carry[5] <= ((neg_a[5] & b[5]) | ((neg_a[5] ^
b[5]) & carry[4]));

                          temp [6] <= (neg_a[6] ^ b[6] ^ carry[5]);
                          carry[6] <= ((neg_a[6] & b[6]) | ((neg_a[6] ^
b[6]) & carry[5]));

                          temp [7] <= (neg_a[7] ^ b[7] ^ carry[6]);
                          carry[7] <= ((neg_a[7] & b[7]) | ((neg_a[7] ^
b[7]) & carry[6]));

                  end


            3'b011   :
                  begin

                          temp <= a | b;
                          carry [7] <= 1'b0;
                  end

            3'b100   :
                  begin

                          temp <= a & b;
                          carry [7] <= 1'b0;
```

```
                end

            3'b101   :
                begin

                        temp <= (~a) & b;
                        carry [7] <= 1'b0;
                end

            3'b110   :
                begin

                        temp <= a ^ b;
                        carry [7] <= 1'b0;
                end

            3'b111   :
                begin

                        temp <= a ~^ b;
                        carry [7] <= 1'b0;
                end


    endcase

        if(temp == 8'b00000000)
            z <= 1'b1;

        else
            z <= 1'b0;

  end

  assign ovf = ( carry[7] ^ carry[6] );
  assign n = temp[7];
  assign cout = carry[7];
  assign f = temp;

endmodule
```