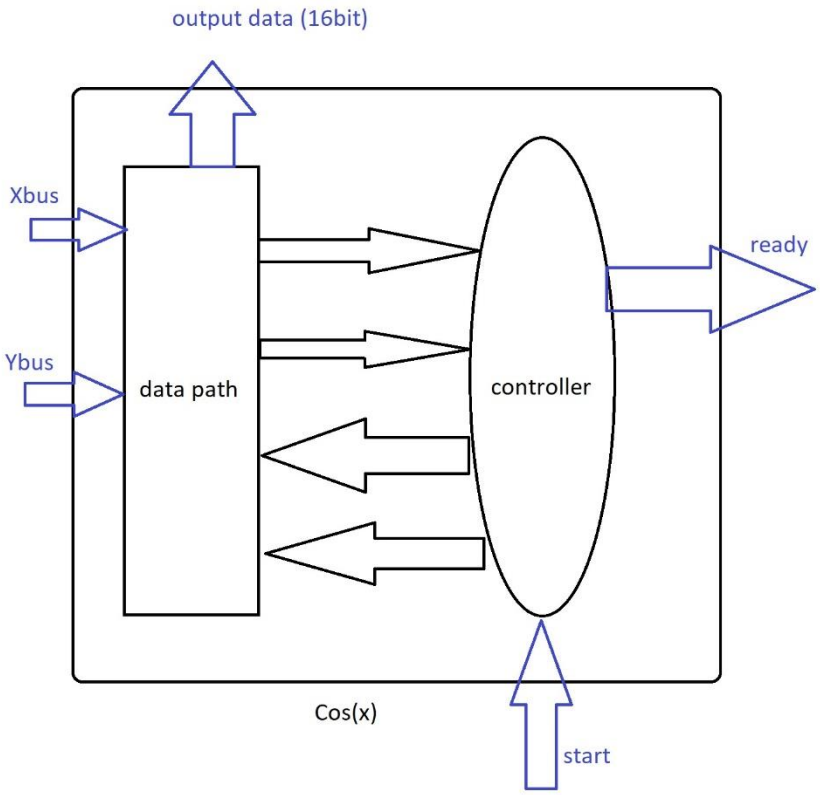
Parisa yahyapour fatideh 810101551

Computer assignment 6 – ECE 367

Design phase:

Question 1)

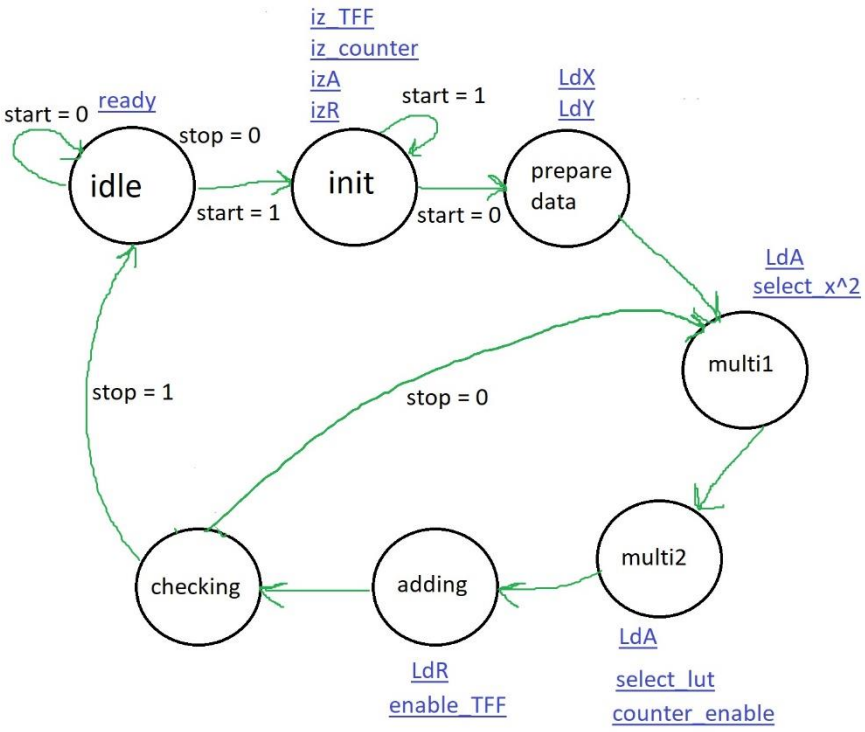
Based on that the circuit will be as below:



Based on the problem description, input to data path of the circuit are x and y. between the controller and data path there will be some signals that their details will be described in further questions. The controller will decides when the total amount is prepared and also when to start calculating.

Question 2)

As we have the components and data path, now, it's time to decide when components will work. The state machine of controller is as below:



We have 7 states, the first state is idle which all controller should have it. In state "init" all components that need initialization will be initialized. The third state is "prepare data" which means the circuit will collect data from bus and save h=them in appropriate registers. Then the process of calculating the amount of $cos(x)$. First the multiplication of previous a with x^2 will be done. Then, second multiplication is related to the data that should come from LUT. The last part of calculation is adding the new amount to previous result. With a TFF, the process of adder component will be chosen, when the output of TFF is 0 then adding will be done and when the output of TFF is 1 subtraction will be done. The checking state is based on y input. As it's been described, y is precision of calculation. We will compare the new amount of a which comes from a register with the y that comes from Y bus. If a < y then we will stop the process and go back to idle state.

Question 3)

Based on the Taylor series of $cos(x)$f, first we implement a for loop for the total functionality.

a = 1

r = 1

sign[2] = [1,-1]

For (i= 0; i < n; i++)

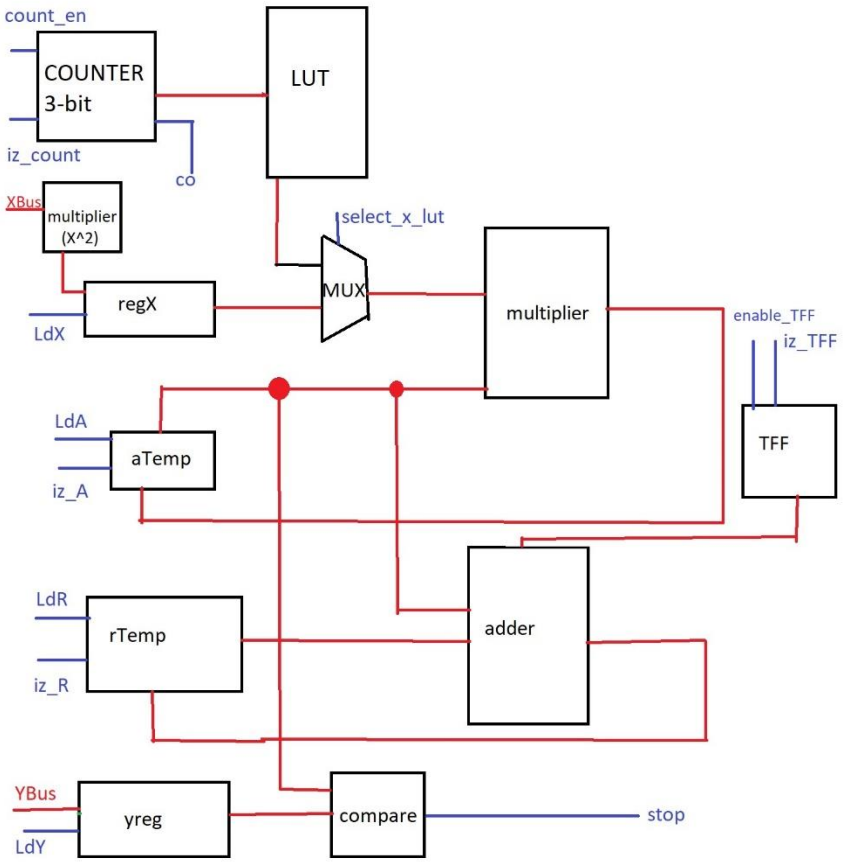
{a = a * x^2

a = a * (1/(k)(k+1)

a = a*sign[i%2]

r = r + a}

Here, based on the for loop, it will be easier to implement the total circuit. Based on that, the components that we need are: look up table, adder, multiplier, registers, MUX and counter.



There are some choices. For instance, for implementing x^2 there are 2 ways: 1- same as above, instead of saving the amount of x, we can save the amount of x^2 directly from bus (this way saves a clock cycle) 2- doing the first multiplication twice (this way saves hardware usage).

Those registers that are related to bus, don't need initialize signal as they are collecting data from bus. But those registers that are saving an amount after some calculation, must have initialization signals. Also, it's clear from the for loop. The variable r which is related to result and the variable a which is related to the amount that is a part of multiplication have initialized value of 1. So, their registers must have this signal.

In above schematic view, blue ones are signals, red ones are data path or the way that components should be connected.

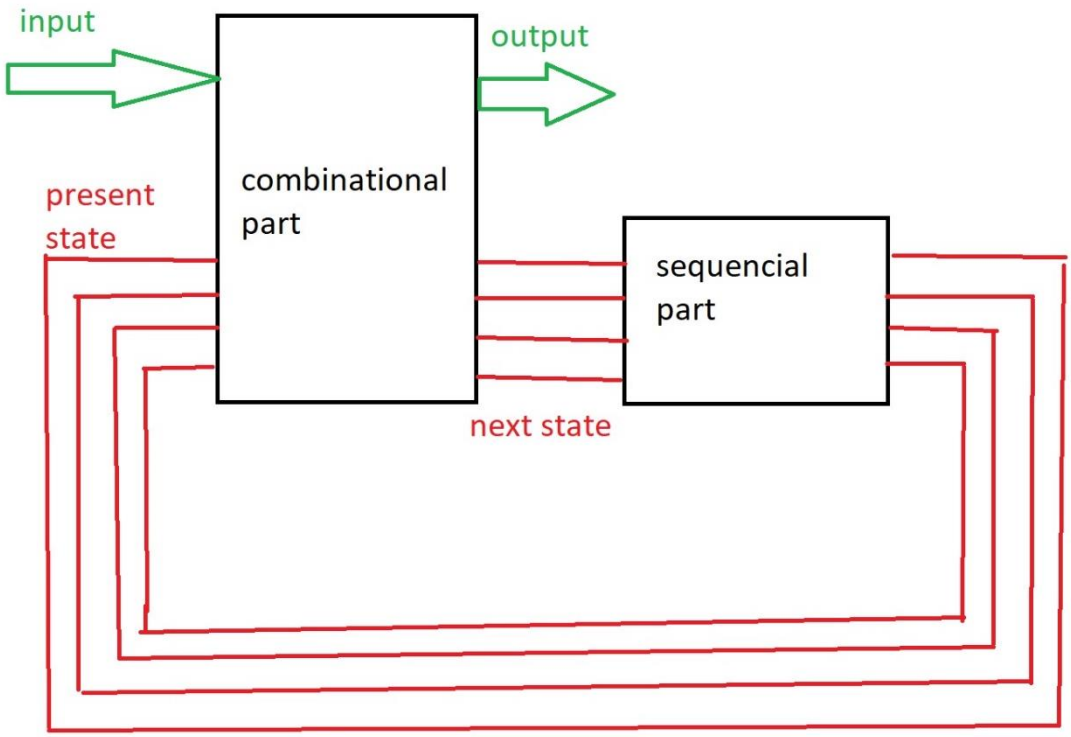
The details of LUT are same as below:

1/2 -> 10000000	K = 0
1/12 -> 00010101	K = 1
1/30 -> 00001000	K = 2
1/56 -> 00000100	K = 3
1/90 -> 00000010	K = 4
1/132 -> 00000001	K = 5
1/182 -> 00000001	K = 6
1/240 -> 00000001	K = 7

The left column is decimal, it should be converted to binary. The last three rows have differences in further bits.

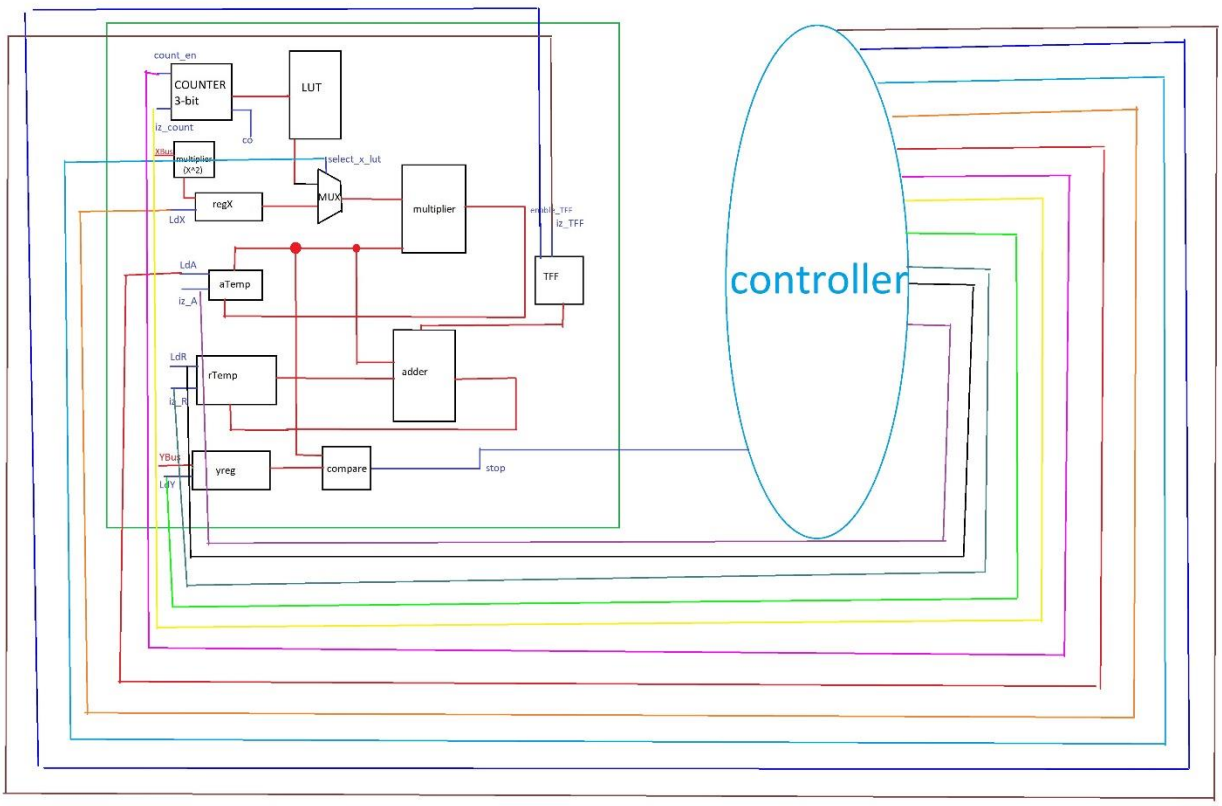
Question 4)

The Huffman model of the controller is as below:



The Huffman model of the circuit is same as other circuits under the blue sky. The combinational part has its own data input/output that based on the decision of controller, output will be output bus for usage of other parts of the system. Based on the present state the controller will choose the next state and send it back to combinational part.

Question 5)



Huffman model of the circuit shoed in last question. Now, the output of controller in connected to the combinational part which is the green bordered part. Controller will send signals based on the signals that it gets from the combinational part.

Implementation phase:

Here it's possible to write codes or use ready components of quartus. Based on the complicated design, I prefer to use codes as it's exactly what I want. So, all components such as adder, multiplier, register, etc. are written in code format.

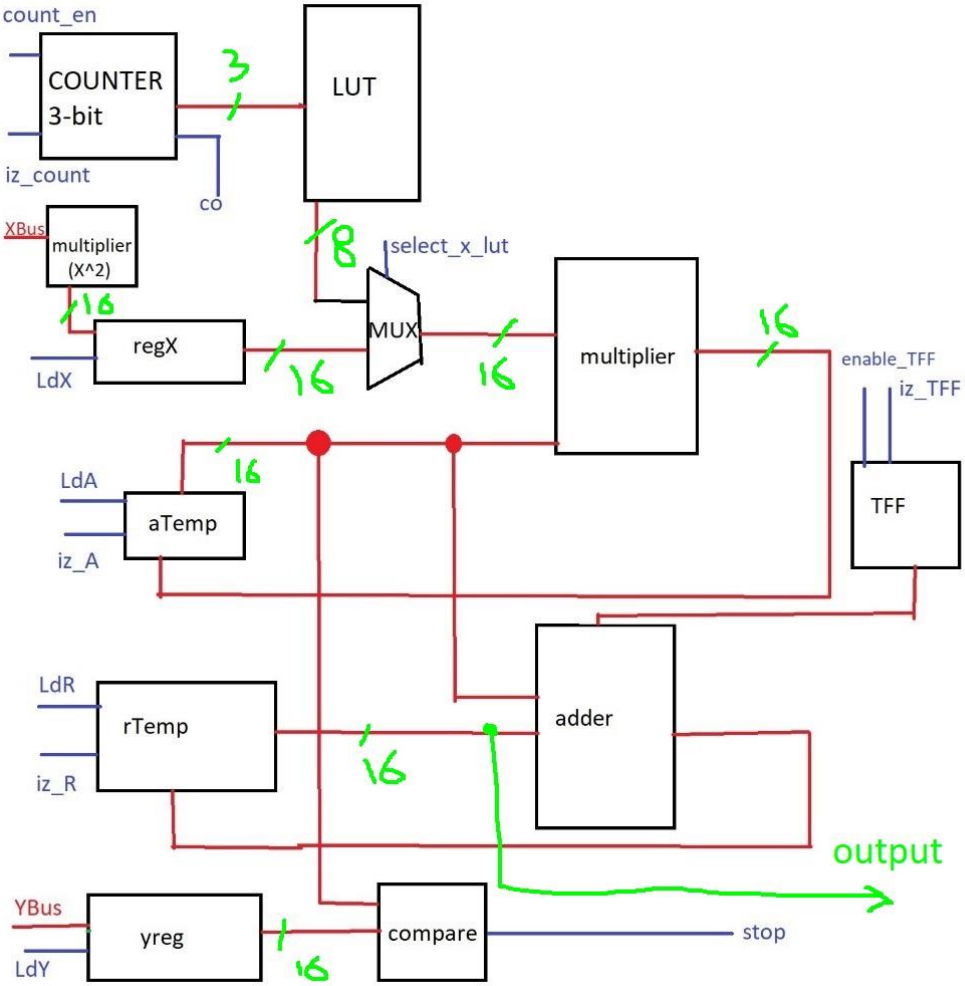
In data path file, all components are connected to each other.

```
`timescale 1ns/1ns
module datapath(input clk, rst, count_en, iz_count, LdX, select_lut, select_x2,
                LdA , iz_A, enable_TFF, iz_TFF, LdR , iz_R , LdY,
                input [15:0] x, input [7:0] y,
                output stop, co, output[15:0] total);
    reg [2:0] address;
    wire [7:0] lut_data;
    wire [15:0] x_2, input1_multi, output_multiplier, adder_output;
    reg [15:0] xreg, areg, Rreg, yreg;
    reg sign;
    wire [15:0] in_multi, y_comp;
    wire [7:0]zero = 8'b00000000;
    assign in_multi = {zero,lut_data};
    assign y_comp = {zero, yreg};
    counter_3bit counter_address_lut(.count_en(count_en), .clk(clk), .rst(rst),
.co(co), .po(address));
    lut lut_ROM(.address(address), .data(lut_data));
    multiplier16 multix2(.x1(x), .x2(x), .result(x_2));
    register16bit xregister(.ParIn(x_2), .clk(clk), .rst(rst), .load(LdX),
.ParOut(xreg));
    mux2to1 multiplexer(.a(in_multi), .b(xreg), .select_1(select_lut),
.select_2(select_x2),.data(input1_multi));
    multiplier16 multiplier(.x1(input1_multi), .x2(areg),
.result(output_multiplier));
    register16bit aregister(.ParIn(output_multiplier), .clk(clk), .rst(rst),
.load(LdA), .ParOut(areg));
    tff sign_chooser(.clk(clk), .iz(iz_TFF), .rst(rst), .en(enable_TFF),
.q(sign));
    register16bit Rregister(.ParIn(adder_output), .clk(clk), .rst(rst),
.load(LdR), .ParOut(Rreg));
    adder16 adder(.x1(Rreg), .x2(areg), .data(adder_output));
    register8bit yregister(.ParIn(y), .clk(clk), .rst(rst), .load(LdY),
.ParOut(yreg));
    comparator comparator_y_a(.y(y_comp), .a(areg), .result(stop));
    assign total = {zero,Rreg[7:0]};



endmodule
```

this is the data path based on this schematic circuit:



The size of each wire is clear.

After creating data path, controller should be connected to it. The controller is:

```
`timescale 1ns/1ns
module controller(input start, co ,stop, clk, rst,
        output reg counter_enable , iz_count, select_lut , select_x2 , LdX ,
        LdA , iz_A , LdR , iz_R , LdY , enable_TFF , iz_TFF , ready);

    parameter Idle = 3'b000, init = 3'b001, prepare_data = 3'b010, multi1 =
3'b011,
            multi2 = 3'b100, adding = 3'b101, checking = 3'b110;

    reg [2:0] ps, ns;

    always @(ps, start, co, stop)begin
        ns = Idle;
        {counter_enable , iz_count , select_lut , select_x2 ,
        LdX , LdA , iz_A , LdR , iz_R , LdY , enable_TFF , iz_TFF} = 12'b0;
        case(ps)
            Idle: begin
                ns = start ? init: Idle;
                ready = 1;
                end
```

```
            init: begin
                 ns = start ? init: prepare_data;
                {iz_count , iz_A , iz_R , iz_TFF} = 4'b1111;
                end
            prepare_data: begin
                ns = multi1;
                {LdY, LdX} = 2'b11;
                end
            multi1: begin
                ns = multi2;
                {select_x2 , LdA} = 2'b11;
                end
            multi2: begin
                ns = adding;
                {counter_enable , select_lut , LdA} = 3'b111;
                end
            adding: begin
                ns = checking;
                {enable_TFF , LdR}= 2'b11;
             end
            checking: begin
                ns = co ? Idle:(stop) ? Idle : multi1;
            end
            default: ns = Idle;
        endcase
    end
    always @(posedge clk, posedge rst) begin
        if (rst)
            ps <= 3'b000;
        else
            ps <= ns;
    end
endmodule
```

which is same as state machine.At the end we have the top-level design of $cos(x)$ which is created by connecting controller to data path.

```
`timescale 1ns/1ns
module cosx(input clk ,rst, start, ready, input[15:0] x, input[7:0]y,
            output [15:0]total);
        wire co ,stop, counter_enable , iz_count, select_lut , select_x2 , LdX,
                LdA , iz_A , LdR , iz_R , LdY , enable_TFF , iz_TFF;
        controller controllerCos(start, co ,stop, clk, rst, counter_enable ,
iz_count, select_lut , select_x2 , LdX ,
        LdA , iz_A , LdR , iz_R , LdY , enable_TFF , iz_TFF , ready);

        datapath datapathCos(clk, rst, counter_enable, iz_count, LdX, select_lut,
select_x2,
                LdA , iz_A, enable_TFF, iz_TFF, LdR , iz_R , LdY,
                x, y,
                stop, co, total);

endmodule
```