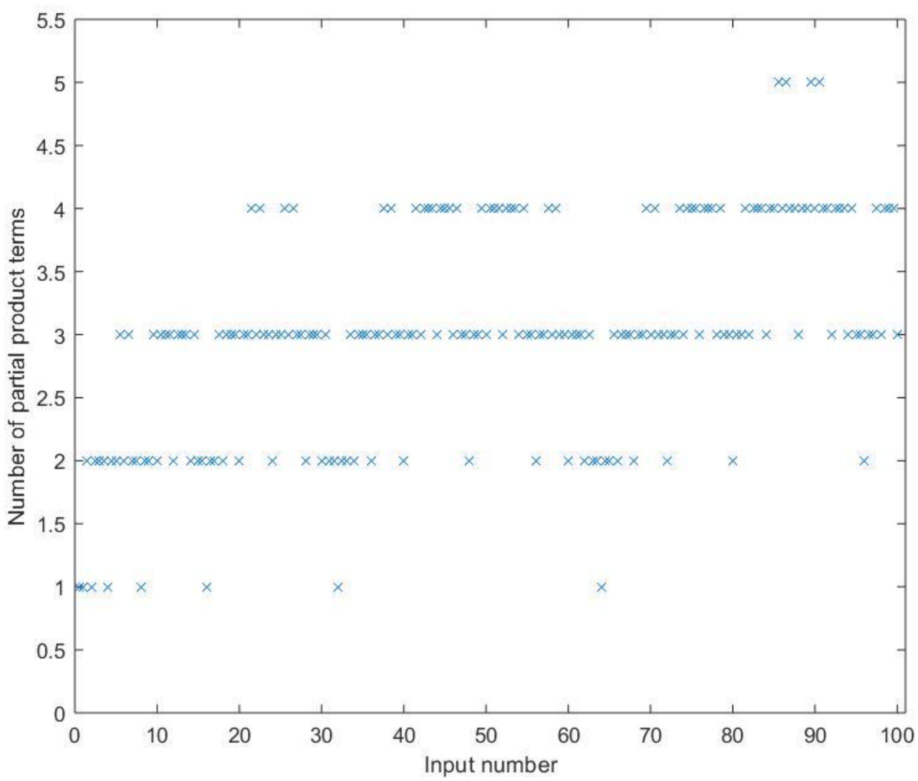
Problem 1

Result

Total sum for +0.5 - +100.00 = 605



numppterms.m

```
function [mpp] = numppterms(num)
    mpp = 0;
    rem = num;
    while (rem ~= 0 && mpp < 10)
        for i = -1:10
            if 2^i >= abs(rem)
                mpp = mpp + 1;
                if (2^i - abs(rem) < abs(rem) - 2^(i-1))
                    rem = abs(rem) - 2^i;
                else
                    rem = abs(rem) - 2^(i-1);
                end
            break;
            end
        end
    end
end
```

Problem 2

Matlab output for unscaled coefficients

```
Scale 1.000000 Using coefficients [ 17 -90 241 902 241 -90 17]
coef #1 17, 2 partial products
coef #2 -90, 4 partial products
coef #3 241, 3 partial products
coef #4 902, 4 partial products
coef #5 241, 3 partial products
coef #6 -90, 4 partial products
coef #7 17, 2 partial products
In total, need 22 partial products
```

Matlab output for scaled coefficients

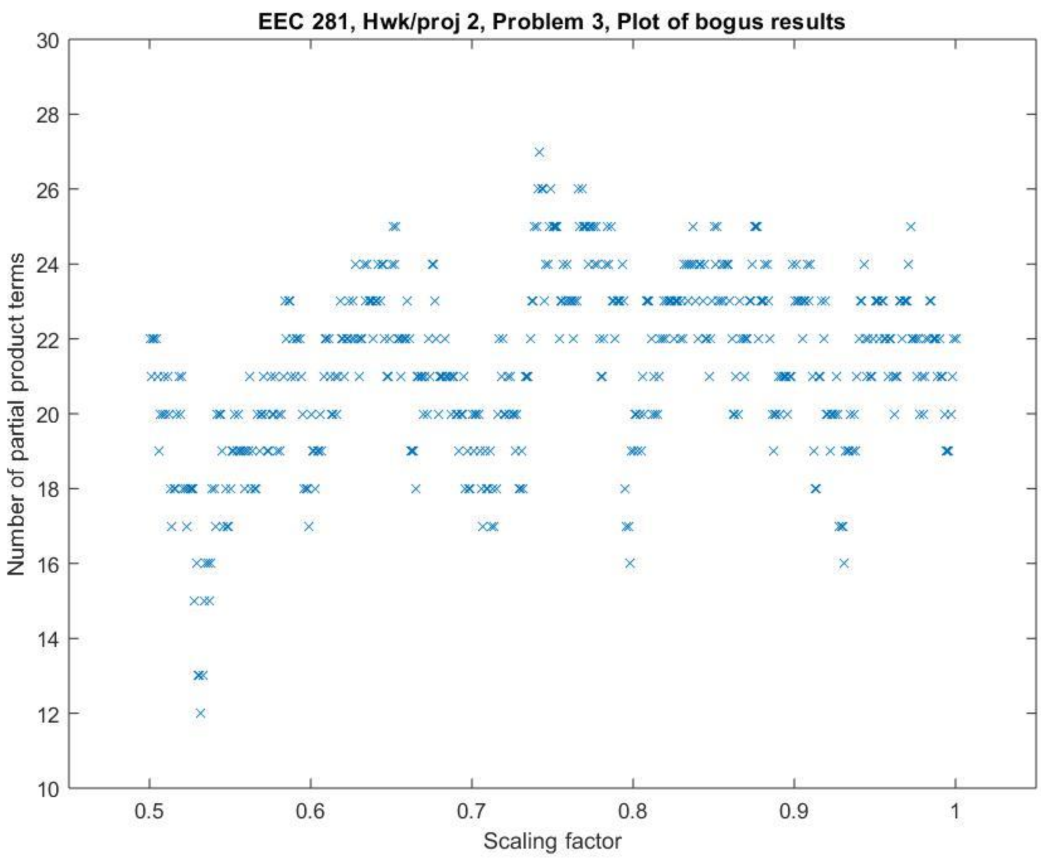
```
Scale 0.532000 Using coefficients [ 9 -48 128 480 128 -48 9]
coef #1 9, 2 partial products
coef #2 -48, 2 partial products
coef #3 128, 1 partial products
coef #4 480, 2 partial products
coef #5 128, 1 partial products
coef #6 -48, 2 partial products
coef #7 9, 2 partial products
In total, need 12 partial products
```

Matlab code

```
clear;
addpath('../part1/') %add numppterms
%fir
%coef = [17 -90  241   902  241 -90  17]
%pps      2   4    3     4    3   4   2
%       +16 -64 +256 +1024 +256 -64 +16
%        +1 -32  -16  -128  -16 -32  +1
%            +4   -1    +4   -1  +4
%            +2         -2       +2

%original coefficients
coef = [17 -90 241 902 241 -90 17];

%scaling factor
%scale = 1;
scale = 0.532; %12 pps, coef_r will be [9 -48 128 480 128 -48 9]

%rounded coedfficients
coef_r = round( coef * scale );
fprintf("Scale %f Using coefficients [", scale);
fprintf(" %d", coef_r);
fprintf("]\n");

%total number of partial products for FIR
total = 0;
```

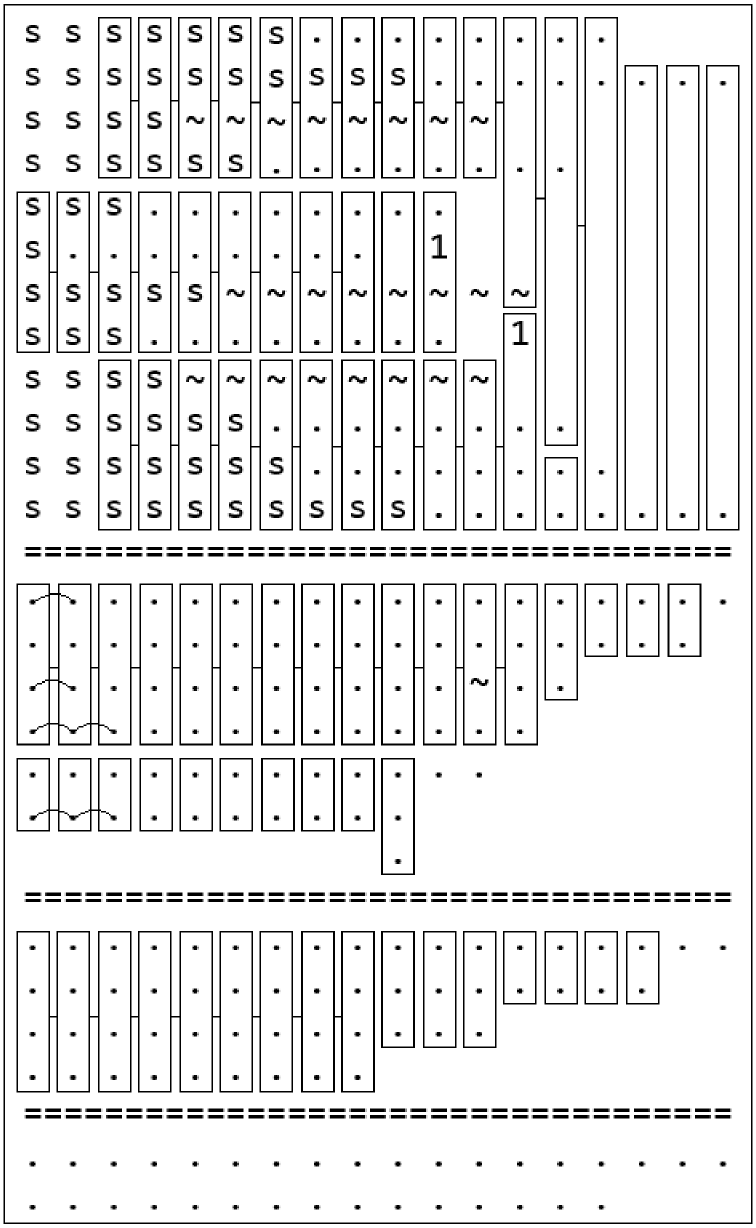
```
%calculation
for fircoef = 1:7
    total = total + numppterms(coef_r(fircoef));
    fprintf("coef #%d %d, %d partial products\n", fircoef, coef_r(fircoef),
numppterms(coef_r(fircoef)));
end

fprintf("In total, need %d partial products\n", total);
```

Bogus plot



Dot diagram

[ ] – 4:2 adder, [ ] – 3:2 adder, [ ] – half adder, "." – signal, "~" – inverted signal, "s" – sign extend, ark – right-to-left signal copy to reduce number of adders due to same inputs.

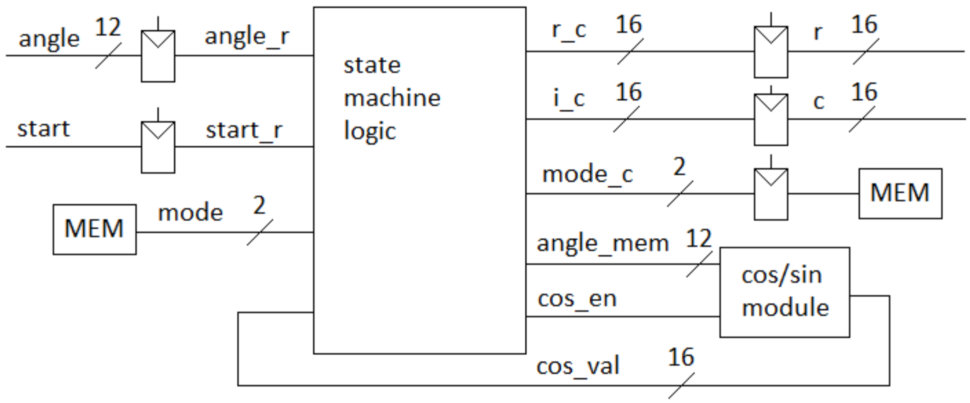


Problem 3

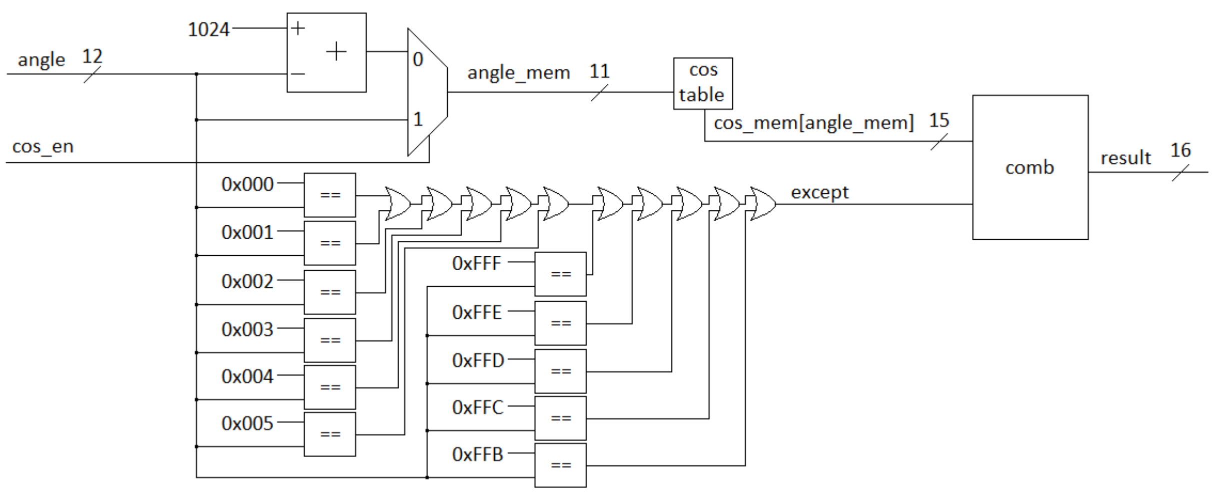
Part 1

Clock period	Total cell area
4ns	345.268
2ns	345.268
20ns	345.268

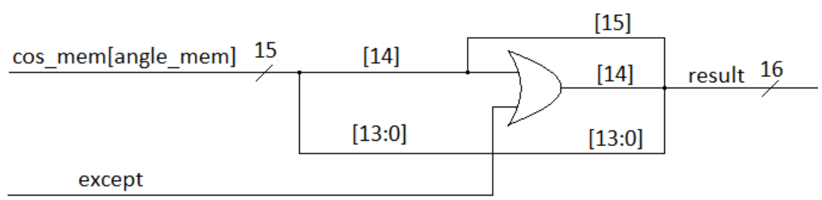
Top-level diagram:



“cos/sin module” diagram:



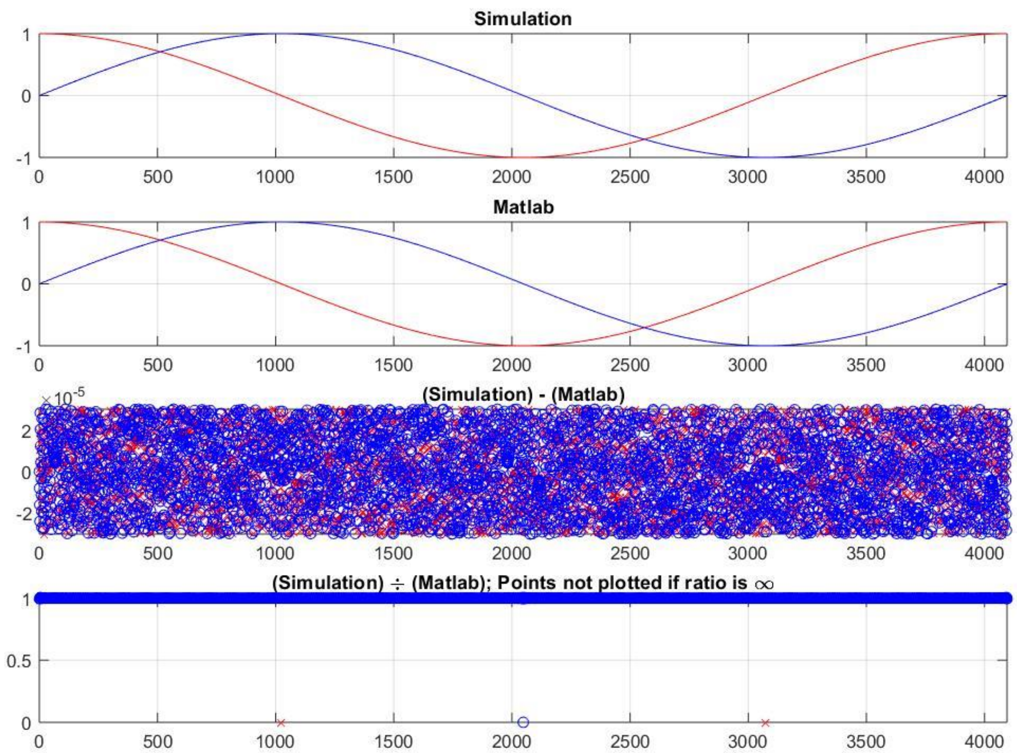
“comb” module diagram (applies exception for result):



Energy_diff / Energy_data0 = 10*log(0.000002/4096.020943) = -83.11dB

Difff.m output:

```
max(data0-data1) =  +0.000031 +0.000031i =  +3.05034e-05 +3.05034e-05i
min(data0-data1) =  -0.000031 -0.000031i =  -3.05034e-05 -3.05034e-05i
max(data0/data1) = -Inf -Infi
min(data0/data1) = +Inf +Infi
approx separated mean(data0/data1) = +0.999508+ +0.999508i
Energy_data0 = 4096.020943
Energy_diff  = 0.000002
Energy_data0/Energy_diff = 1653952521.828711 = 92.185230dB
```



Code compl.v:

```
/*
Top level for complex number generator
*/
```

```
`timescale 1ns/10ps

module compl(
    input [11:0] angle,
    input start,
    input clk,
    input reset,
    output reg [15:0] r,
    output reg [15:0] i
);

parameter IDLE = 2'b00;
parameter COS = 2'b01;
parameter SIN = 2'b10;

reg [1:0] mode, mode_c;
reg [11:0] angle_mem, angle_r;
reg [15:0] r_c, i_c;
wire [15:0] cos_val;
reg cos_en, start_r;

initial begin
    mode = IDLE;
    mode_c = IDLE;
    angle_mem = 12'b0000_0000_0000;
//    angle_r = 12'b0000_0000_0000;
    r_c = 16'b0000_0000_0000_0000;
    i_c = 16'b0000_0000_0000_0000;
    cos_en = 1'b0;
//    start_r = 1'b0;
end

cos cos_mem (.angle (angle_r), .cos_en(cos_en), .result(cos_val));

//State Machine
always @(angle_r or start_r or mode) begin
    mode_c = mode;
    case(mode)
        IDLE: begin
            if (start_r == 1'b1) begin
                mode_c = COS;
                cos_en = mode_c[0];
                angle_mem = angle_r;
            end
        end
        COS: begin
            r_c = cos_val;
            mode_c = SIN;
            cos_en = mode_c[0];
        end
```

```
        SIN: begin
            i_c = cos_val;
            mode_c = IDLE;
        end
        default: begin
            mode_c = IDLE;
        end
    endcase
end

//output FFlop
always @(posedge clk or posedge reset) begin
    if (reset == 1'b1) begin
        r <= #1 16'b0000_0000_0000_0000;
        i <= #1 16'b0000_0000_0000_0000;
        mode <= #1 IDLE;
        angle_r <= #1 12'b0000_0000_0000;
        start_r <= #1 1'b0;
    end else begin
        r <= #1 r_c;
        i <= #1 i_c;
        mode <= #1 mode_c;
        angle_r <= #1 angle;
        start_r <= #1 start;
    end
end

endmodule
```

Code cos.v

```
//cos function lookup table
//full precision for angle [0 .. 360) degrees
`timescale 1ns/10ps

module cos(
    input cos_en,
    input [11:0] angle,
    output reg [15:0] result
);

reg except;
reg [14:0] cos_mem [4095:0];
reg [11:0] angle_mem;

initial begin
    result = 16'b0000_0000_0000_0000;
    //load values to memory
    cos_mem[12'b000000000000] = 15'b000000000000000;
    cos_mem[12'b000000000001] = 15'b000000000000000;
    cos_mem[12'b000000000010] = 15'b000000000000000;
```

```
    cos_mem[12'b000000000011] = 15'b000000000000000;
    cos_mem[12'b000000000100] = 15'b000000000000000;
    cos_mem[12'b000000000101] = 15'b000000000000000;
    cos_mem[12'b000000000110] = 15'b011111111111111;
    cos_mem[12'b000000000111] = 15'b011111111111111;
    cos_mem[12'b000000001000] = 15'b011111111111111;
    cos_mem[12'b000000001001] = 15'b011111111111110;
    cos_mem[12'b000000001010] = 15'b011111111111110;
    cos_mem[12'b000000001011] = 15'b011111111111110;
    cos_mem[12'b000000001100] = 15'b011111111111101;
    cos_mem[12'b000000001101] = 15'b011111111111101;
    cos_mem[12'b000000001110] = 15'b011111111111100;
    cos_mem[12'b000000001111] = 15'b011111111111100;

      //<Many lines removed>

    cos_mem[12'b111111110001] = 15'b011111111111100;
    cos_mem[12'b111111110010] = 15'b011111111111100;
    cos_mem[12'b111111110011] = 15'b011111111111101;
    cos_mem[12'b111111110100] = 15'b011111111111101;
    cos_mem[12'b111111110101] = 15'b011111111111110;
    cos_mem[12'b111111110110] = 15'b011111111111110;
    cos_mem[12'b111111110111] = 15'b011111111111110;
    cos_mem[12'b111111111000] = 15'b011111111111111;
    cos_mem[12'b111111111001] = 15'b011111111111111;
    cos_mem[12'b111111111010] = 15'b011111111111111;
    cos_mem[12'b111111111011] = 15'b000000000000000;
    cos_mem[12'b111111111100] = 15'b000000000000000;
    cos_mem[12'b111111111101] = 15'b000000000000000;
    cos_mem[12'b111111111110] = 15'b000000000000000;
    cos_mem[12'b111111111111] = 15'b000000000000000;
end

always @(angle or cos_en) begin
    //cos/sin selection
    if (cos_en == 1'b1)
        //compute cos(angle)
        angle_mem = angle;
    else
        //compute sin(angle) = cos (pi/2 - angle)
        angle_mem = 12'b0100_0000_0000 - angle;

    //exception (when need to return == +1.0)
    except = (angle_mem == 12'b0000_0000_0000);
    except = except || (angle_mem == 12'b0000_0000_0001);
    except = except || (angle_mem == 12'b0000_0000_0010);
    except = except || (angle_mem == 12'b0000_0000_0011);
    except = except || (angle_mem == 12'b0000_0000_0100);
    except = except || (angle_mem == 12'b0000_0000_0101);
```

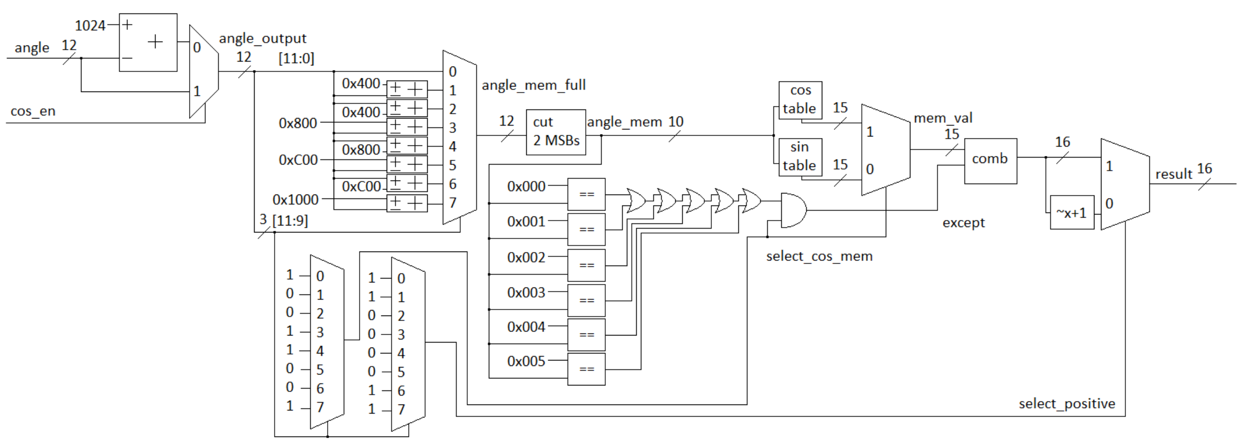
```
    except = except || (angle_mem == 12'b1111_1111_1011);
    except = except || (angle_mem == 12'b1111_1111_1100);
    except = except || (angle_mem == 12'b1111_1111_1101);
    except = except || (angle_mem == 12'b1111_1111_1111);
    except = except || (angle_mem == 12'b1111_1111_1110);

    //output
    result = {cos_mem[angle_mem][14], cos_mem[angle_mem][14] | except,
cos_mem[angle_mem][13:0]};
end

endmodule
```

Part 2

Clock period	Total cell area
4ns	345.268
2ns	345.268
20ns	345.268

Top-level design is the same as in part 1.

“cos/sin module” diagram:



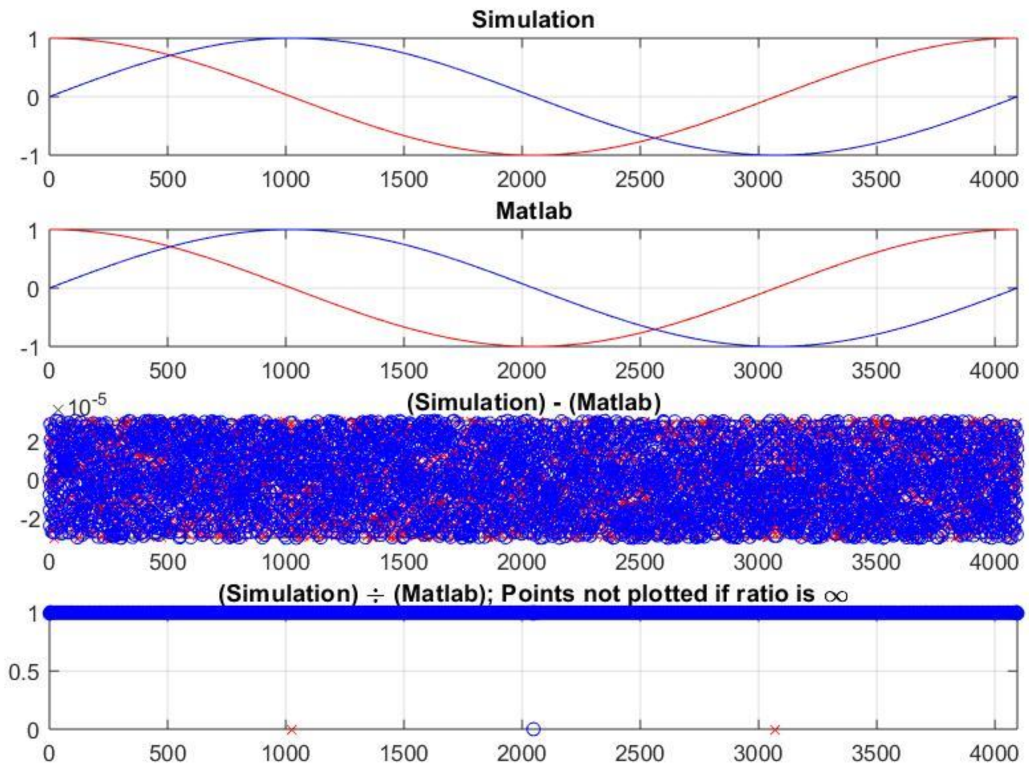
“Comb” module is the same as in part 1.

Energy_diff / Energy_data0 = 10*log(0.000002/4096.020943) = -83.113dB

Difff.m output:

```
max(data0-data1) =  +0.000031 +0.000031i =  +3.05034e-05 +3.05034e-05i
min(data0-data1) =  -0.000031 -0.000031i =  -3.05034e-05 -3.05034e-05i
max(data0/data1) = -Inf -Infi
min(data0/data1) = +Inf +Infi
approx separated mean(data0/data1) = +0.999508+ +0.999508i
Energy_data0 = 4096.020943
Energy_diff  = 0.000002
```

```
Energy_data0/Energy_diff = 1653952521.828711 = 92.185230dB
```



Code compl.v

```
/*
Top level for complex number generator
*/

`timescale 1ns/10ps

module compl(
    input [11:0] angle,
    input start,
    input clk,
    input reset,
    output reg [15:0] r,
    output reg [15:0] i
);

parameter IDLE = 2'b00;
parameter COS = 2'b01;
parameter SIN = 2'b10;

reg [1:0] mode, mode_c;
reg [11:0] angle_mem, angle_r;
```

```
reg [15:0] r_c, i_c;
wire [15:0] cos_val;
reg cos_en, start_r;

initial begin
    mode = IDLE;
    mode_c = IDLE;
    angle_mem = 12'b0000_0000_0000;
//    angle_r = 12'b0000_0000_0000;
    r_c = 16'b0000_0000_0000_0000;
    i_c = 16'b0000_0000_0000_0000;
    cos_en = 1'b0;
//    start_r = 1'b0;
end

cos cos_mem (.angle (angle_r), .cos_en(cos_en), .result(cos_val));

//State Machine
always @(angle_r or start_r or mode) begin
    mode_c = mode;
    case(mode)
        IDLE: begin
            if (start_r == 1'b1) begin
                mode_c = COS;
                cos_en = mode_c[0];
                angle_mem = angle_r;
            end
        end
        COS: begin
            r_c = cos_val;
            mode_c = SIN;
            cos_en = mode_c[0];
        end
        SIN: begin
            i_c = cos_val;
            mode_c = IDLE;
        end
        default: begin
            mode_c = IDLE;
        end
    endcase
end

//output FFlop
always @(posedge clk or posedge reset) begin
    if (reset == 1'b1) begin
        r <= #1 16'b0000_0000_0000_0000;
        i <= #1 16'b0000_0000_0000_0000;
        mode <= #1 IDLE;
        angle_r <= #1 12'b0000_0000_0000;
```

```
        start_r <= #1 1'b0;
    end else begin
        r <= #1 r_c;
        i <= #1 i_c;
        mode <= #1 mode_c;
        angle_r <= #1 angle;
        start_r <= #1 start;
    end
end

endmodule
```

Code cos.v

```
//cos function lookup table
//full precision and [0 .. 45) degrees
`timescale 1ns/10ps

module cos(
    input cos_en,
    input [11:0] angle, //full angle
    output reg [15:0] result
);

reg except;
reg [14:0] cos_mem [512:0];
reg [14:0] sin_mem [512:0];
reg [9:0] angle_mem; //memory address

//logic variables
reg select_cos_mem;
reg select_positive;
reg [11:0] angle_output, angle_mem_full;
reg [14:0] mem_val;

initial begin
    //defaults
    result = 16'b0000_0000_0000_0000;
    mem_val = 15'b000_0000_0000_0000;
    angle_output = 12'b0000_0000_0000;
    angle_mem_full = 12'b0000_0000_0000;
    select_cos_mem = 1'b0;
    select_positive = 1'b1;
    //load values to memory;
    cos_mem[10'b0000000000] = 15'b000000000000000; sin_mem[10'b0000000000] =
15'b000000000000000;
    cos_mem[10'b0000000001] = 15'b000000000000000; sin_mem[10'b0000000001] =
15'b000000000011001;
    cos_mem[10'b0000000010] = 15'b000000000000000; sin_mem[10'b0000000010] =
15'b000000000110010;
```

```
    cos_mem[10'b0000000011] = 15'b000000000000000; sin_mem[10'b0000000011] =
15'b000000001001011;
    cos_mem[10'b0000000100] = 15'b000000000000000; sin_mem[10'b0000000100] =
15'b000000001100101;
    cos_mem[10'b0000000101] = 15'b000000000000000; sin_mem[10'b0000000101] =
15'b000000001111110;
    cos_mem[10'b0000000110] = 15'b011111111111111; sin_mem[10'b0000000110] =
15'b000000010010111;
    cos_mem[10'b0000000111] = 15'b011111111111111; sin_mem[10'b0000000111] =
15'b000000010110000;
    cos_mem[10'b0000001000] = 15'b011111111111111; sin_mem[10'b0000001000] =
15'b000000011001001;
    cos_mem[10'b0000001001] = 15'b011111111111110; sin_mem[10'b0000001001] =
15'b000000011100010;
    cos_mem[10'b0000001010] = 15'b011111111111110; sin_mem[10'b0000001010] =
15'b000000011111011;
    cos_mem[10'b0000001011] = 15'b011111111111110; sin_mem[10'b0000001011] =
15'b000000100010100;
    cos_mem[10'b0000001100] = 15'b011111111111101; sin_mem[10'b0000001100] =
15'b000000100101110;
    cos_mem[10'b0000001101] = 15'b011111111111101; sin_mem[10'b0000001101] =
15'b000000101000111;
    cos_mem[10'b0000001110] = 15'b011111111111100; sin_mem[10'b0000001110] =
15'b000000101100000;

      //<Many lines removed>

    cos_mem[10'b0111110010] = 15'b010111000110111; sin_mem[10'b0111110010] =
15'b010110001000110;
    cos_mem[10'b0111110011] = 15'b010111000100110; sin_mem[10'b0111110011] =
15'b010110001011000;
    cos_mem[10'b0111110100] = 15'b010111000010101; sin_mem[10'b0111110100] =
15'b010110001101010;
    cos_mem[10'b0111110101] = 15'b010111000000011; sin_mem[10'b0111110101] =
15'b010110001111100;
    cos_mem[10'b0111110110] = 15'b010110111110010; sin_mem[10'b0111110110] =
15'b010110010001110;
    cos_mem[10'b0111110111] = 15'b010110111100000; sin_mem[10'b0111110111] =
15'b010110010100000;
    cos_mem[10'b0111111000] = 15'b010110111001111; sin_mem[10'b0111111000] =
15'b010110010110010;
    cos_mem[10'b0111111001] = 15'b010110110111101; sin_mem[10'b0111111001] =
15'b010110011000100;
    cos_mem[10'b0111111010] = 15'b010110110101011; sin_mem[10'b0111111010] =
15'b010110011010110;
    cos_mem[10'b0111111011] = 15'b010110110011010; sin_mem[10'b0111111011] =
15'b010110011101000;
    cos_mem[10'b0111111100] = 15'b010110110001000; sin_mem[10'b0111111100] =
15'b010110011111010;
```

```
    cos_mem[10'b0111111101] = 15'b010110101110110; sin_mem[10'b0111111101] =
15'b010110100001100;
    cos_mem[10'b0111111110] = 15'b010110101100101; sin_mem[10'b0111111110] =
15'b010110100011110;
    cos_mem[10'b0111111111] = 15'b010110101010011; sin_mem[10'b0111111111] =
15'b010110100101111;
    cos_mem[10'b1000000000] = 15'b010110101000001; sin_mem[10'b1000000000] =
15'b010110101000001;
end

always @(angle or cos_en) begin
    //cos/sin selection for output
    if (cos_en == 1'b1)
        //compute cos(angle)
        angle_output = angle;
    else
        //compute sin(angle) = cos (pi/2 - angle)
        angle_output = 11'b100_0000_0000 - angle;

    //choose cos/sin mem bank
    case (angle_output[11:9])
        3'b000: begin //0 - 45
            angle_mem_full = angle_output;
            select_cos_mem = 1'b1;
            select_positive = 1'b1;
        end
        3'b001: begin //45 - 90
            angle_mem_full = 12'h400 - angle_output;
            select_cos_mem = 1'b0;
            select_positive = 1'b1;
        end
        3'b010: begin //90 - 135
            angle_mem_full = angle_output[9:0] - 12'h400;
            select_cos_mem = 1'b0;
            select_positive = 1'b0;
        end
        3'b011: begin //135 - 180
            angle_mem_full = 12'h800 - angle_output;
            select_cos_mem = 1'b1;
            select_positive = 1'b0;
        end
        3'b100: begin //180 - 225
            angle_mem_full = angle_output - 12'h800;
            select_cos_mem = 1'b1;
            select_positive = 1'b0;
        end
        3'b101: begin //225 - 270
            angle_mem_full = 12'hC00 - angle_output;
            select_cos_mem = 1'b0;
            select_positive = 1'b0;
```

```
        end
        3'b110: begin //270 - 315
            angle_mem_full = angle_output - 12'hC00;
            select_cos_mem = 1'b0;
            select_positive = 1'b1;
        end
        3'b111: begin //315 - 360
            angle_mem_full = 13'h1000 - angle_output;
            select_cos_mem = 1'b1;
            select_positive = 1'b1;
        end
    endcase

    //exception (when need to return +1.0)
    angle_mem = angle_mem_full[9:0];
    except = (angle_mem == 10'b00_0000_0000);
    except = except || (angle_mem == 10'b00_0000_0001);
    except = except || (angle_mem == 10'b00_0000_0010);
    except = except || (angle_mem == 10'b00_0000_0011);
    except = except || (angle_mem == 10'b00_0000_0100);
    except = except || (angle_mem == 10'b00_0000_0101);
    //except only work for cos bank near angle 0
    except = except & select_cos_mem;

    //read memory
    if (select_cos_mem == 1'b1) begin
        mem_val = cos_mem[angle_mem];
    end else begin
        mem_val = sin_mem[angle_mem];
    end

    //make output
    if (select_positive == 1'b1) begin
        result = {mem_val[14], mem_val[14] | except, mem_val[13:0]};
    end else begin
        result = -{mem_val[14], mem_val[14] | except, mem_val[13:0]};
    end

end

endmodule
```

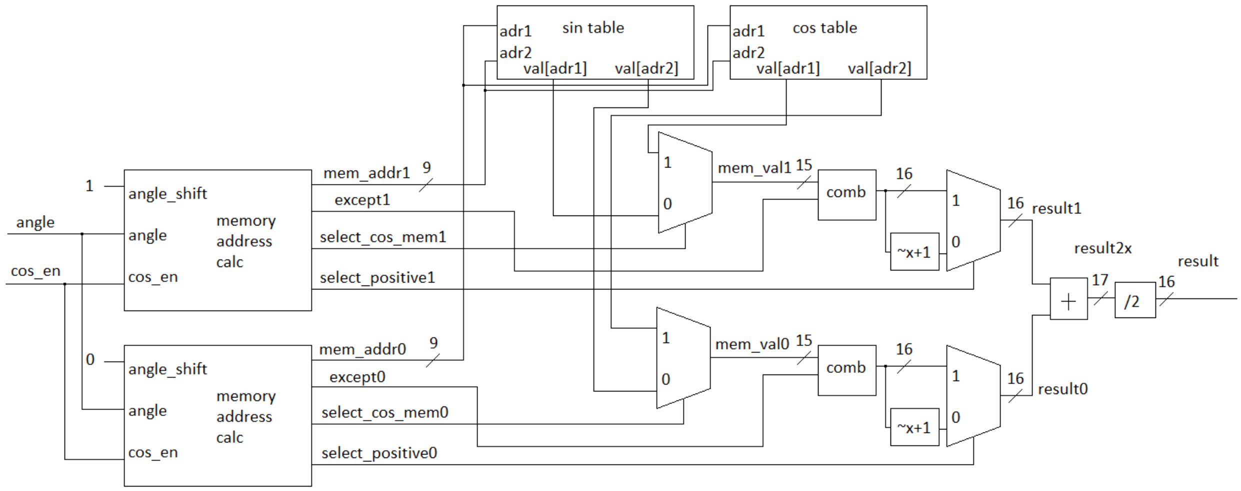
Part 3

Clock period	Total cell area
4ns	345.268
2ns	345.268
20ns	345.268

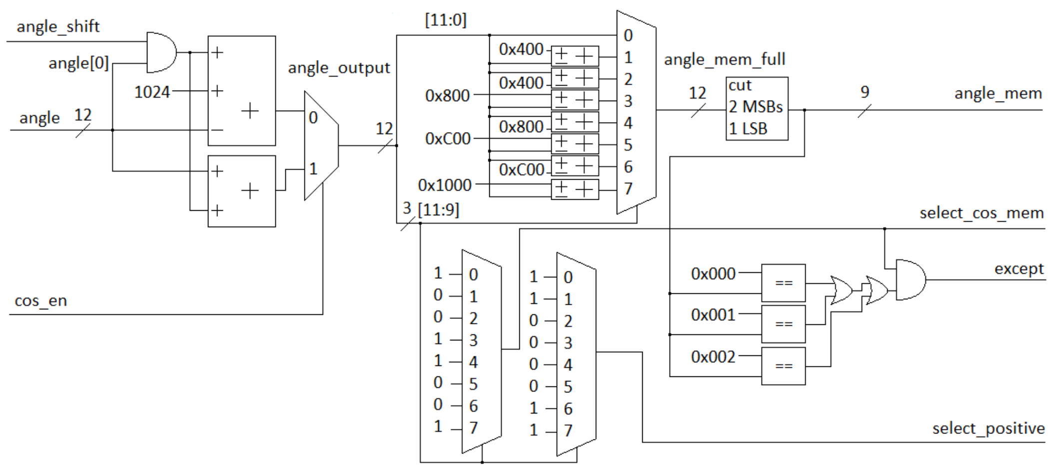
Top –level diagram is the same as in part 1 and 2.

Maxim Shepovalov HW3 EEC281

Cos/sin module:



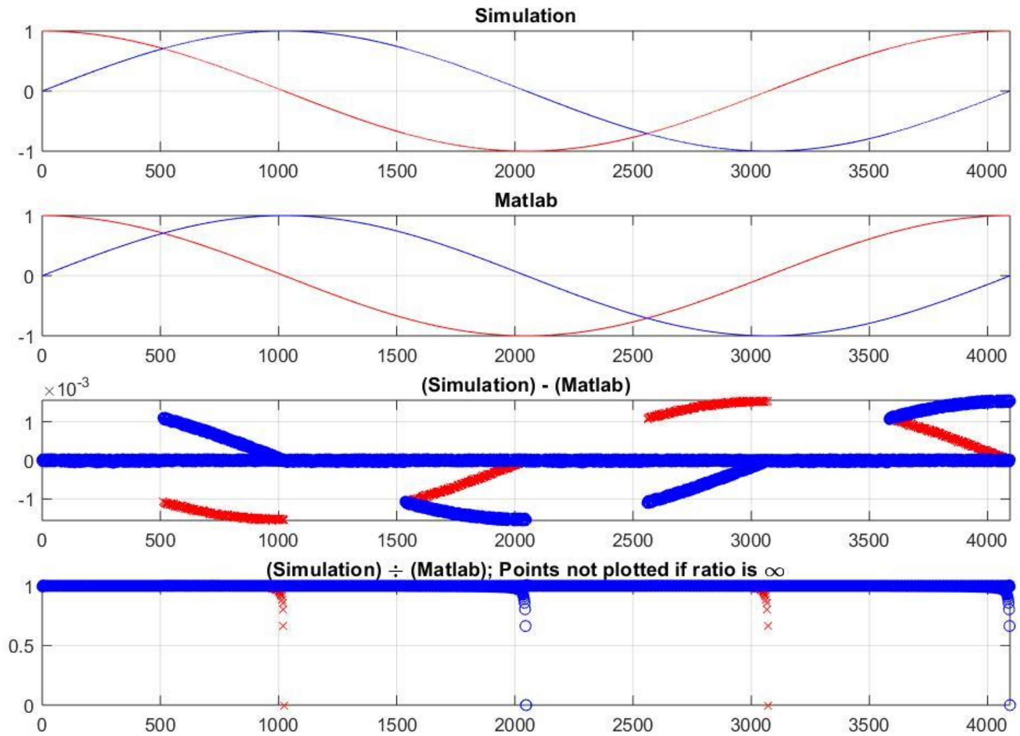
Memory address module:



Energy_diff / Energy_data0 = 10*log(0.002413/4096.016469) = -62.298dB

Difff.m output:

```
max(data0-data1) =  +0.001554 +0.001554i =  +0.00155412 +0.00155412i
min(data0-data1) =  -0.001554 -0.001554i =  -0.00155412 -0.00155412i
max(data0/data1) = -Inf -Infi
min(data0/data1) = +Inf +Infi
approx separated mean(data0/data1) = +0.997796+ +0.997796i
Energy_data0 = 4096.016469
Energy_diff  = 0.002413
Energy_data0/Energy_diff = 1697518.136491 = 62.298144dB
```



Code compl.v

```
/*
Top level for complex number generator
*/

`timescale 1ns/10ps

module compl(
    input [11:0] angle,
    input start,
    input clk,
    input reset,
    output reg [15:0] r,
    output reg [15:0] i
);

parameter IDLE = 2'b00;
parameter COS = 2'b01;
parameter SIN = 2'b10;

reg [1:0] mode, mode_c;
reg [11:0] angle_mem, angle_r;
reg [15:0] r_c, i_c;
wire [15:0] cos_val;
```

```
reg cos_en, start_r;

initial begin
    mode = IDLE;
    mode_c = IDLE;
    angle_mem = 12'b0000_0000_0000;
//    angle_r = 12'b0000_0000_0000;
    r_c = 16'b0000_0000_0000_0000;
    i_c = 16'b0000_0000_0000_0000;
    cos_en = 1'b0;
//    start_r = 1'b0;
end

cos cos_mem (.angle (angle_r), .cos_en(cos_en), .result(cos_val));

//State Machine
always @(angle_r or start_r or mode) begin
    mode_c = mode;
    case(mode)
        IDLE: begin
            if (start_r == 1'b1) begin
                mode_c = COS;
                cos_en = mode_c[0];
                angle_mem = angle_r;
            end
        end
        COS: begin
            r_c = cos_val;
            mode_c = SIN;
            cos_en = mode_c[0];
        end
        SIN: begin
            i_c = cos_val;
            mode_c = IDLE;
        end
        default: begin
            mode_c = IDLE;
        end
    endcase
end

//output FFlop
always @(posedge clk or posedge reset) begin
    if (reset == 1'b1) begin
        r <= #1 16'b0000_0000_0000_0000;
        i <= #1 16'b0000_0000_0000_0000;
        mode <= #1 IDLE;
        angle_r <= #1 12'b0000_0000_0000;
        start_r <= #1 1'b0;
    end else begin
```

```
        r <= #1 r_c;
        i <= #1 i_c;
        mode <= #1 mode_c;
        angle_r <= #1 angle;
        start_r <= #1 start;
    end
end

endmodule
```

Code cos.v

```
//cos function lookup table
//full precision and [0 .. 45) degrees
`timescale 1ns/10ps

module addr_selector(
    input [11:0] angle,
    input angle_shift,
    input cos_en,
    output reg [8:0] mem_address,
    output reg do_except,
    output reg select_cos_mem,
    output reg select_positive
);

//logic variables
reg [11:0] angle_output, angle_mem_full;

always @(angle or cos_en) begin
    //cos/sin selection for output
    if (cos_en == 1'b1)
        //compute cos(angle)
        angle_output = angle + angle[0]*angle_shift;
    else
        //compute sin(angle) = cos (pi/2 - angle)
        angle_output = 11'b100_0000_0000 - angle - angle[0]*angle_shift;

    //choose cos/sin mem bank
    case (angle_output[11:9])
        3'b000: begin //0 - 45
            angle_mem_full = angle_output;
            select_cos_mem = 1'b1;
            select_positive = 1'b1;
        end
        3'b001: begin //45 - 90
            angle_mem_full = 12'h400 - angle_output;
            select_cos_mem = 1'b0;
            select_positive = 1'b1;
        end
        3'b010: begin //90 - 135
```

```
            angle_mem_full = angle_output[9:0] - 12'h400;
            select_cos_mem = 1'b0;
            select_positive = 1'b0;
        end
        3'b011: begin //135 - 180
            angle_mem_full = 12'h800 - angle_output;
            select_cos_mem = 1'b1;
            select_positive = 1'b0;
        end
        3'b100: begin //180 - 225
            angle_mem_full = angle_output - 12'h800;
            select_cos_mem = 1'b1;
            select_positive = 1'b0;
        end
        3'b101: begin //225 - 270
            angle_mem_full = 12'hC00 - angle_output;
            select_cos_mem = 1'b0;
            select_positive = 1'b0;
        end
        3'b110: begin //270 - 315
            angle_mem_full = angle_output - 12'hC00;
            select_cos_mem = 1'b0;
            select_positive = 1'b1;
        end
        3'b111: begin //315 - 360
            angle_mem_full = 13'h1000 - angle_output;
            select_cos_mem = 1'b1;
            select_positive = 1'b1;
        end
    endcase

    //exception (when need to return +1.0)
    mem_address = angle_mem_full[9:1];
    do_except = (mem_address == 10'b00_0000_0000);
    do_except = do_except || (mem_address == 10'b00_0000_0001);
    do_except = do_except || (mem_address == 10'b00_0000_0010);
    //except only work for cos bank near angle 0
    do_except = do_except & select_cos_mem;
end

endmodule

/////////////////////////////////////////////////////////////////////

module cos(
    input cos_en,
    input [11:0] angle,
    output reg [15:0] result
);
```

```
reg [15:0] result0, result1;
reg [16:0] result2x;
reg [14:0] cos_mem [256:0];
reg [14:0] sin_mem [256:0];

reg [14:0] mem_val0, mem_val1;
wire [8:0] mem_adr0, mem_adr1;
wire select_cos_mem0, select_cos_mem1, select_positive0, select_positive1,
except0, except1;

initial begin
    //defaults
    result = 16'b0000_0000_0000_0000;
    mem_val0 = 15'b000_0000_0000_0000;
    mem_val1 = 15'b000_0000_0000_0000;
    //load values to memory;
    cos_mem[9'b000000000] = 15'b000000000000000; sin_mem[9'b000000000] =
15'b000000000000000;
    cos_mem[9'b000000001] = 15'b000000000000000; sin_mem[9'b000000001] =
15'b000000000110010;
    cos_mem[9'b000000010] = 15'b000000000000000; sin_mem[9'b000000010] =
15'b000000001100101;
    cos_mem[9'b000000011] = 15'b011111111111111; sin_mem[9'b000000011] =
15'b000000010010111;
    cos_mem[9'b000000100] = 15'b011111111111111; sin_mem[9'b000000100] =
15'b000000011001001;
    cos_mem[9'b000000101] = 15'b011111111111110; sin_mem[9'b000000101] =
15'b000000011111011;
    cos_mem[9'b000000110] = 15'b011111111111101; sin_mem[9'b000000110] =
15'b000000100101110;
    cos_mem[9'b000000111] = 15'b011111111111100; sin_mem[9'b000000111] =
15'b000000101100000;
    cos_mem[9'b000001000] = 15'b011111111111011; sin_mem[9'b000001000] =
15'b000000110010010;
    cos_mem[9'b000001001] = 15'b011111111111010; sin_mem[9'b000001001] =
15'b000000111000100;
    cos_mem[9'b000001010] = 15'b011111111111000; sin_mem[9'b000001010] =
15'b000000111110111;
    cos_mem[9'b000001011] = 15'b011111111110111; sin_mem[9'b000001011] =
15'b000001000101001;
    cos_mem[9'b000001100] = 15'b011111111110101; sin_mem[9'b000001100] =
15'b000001001011011;
    cos_mem[9'b000001101] = 15'b011111111110011; sin_mem[9'b000001101] =
15'b000001010001101;
    cos_mem[9'b000001110] = 15'b011111111110001; sin_mem[9'b000001110] =
15'b000001011000000;

      //<Many lines removed>
```

```
    cos_mem[9'b011110010] = 15'b010111100101000; sin_mem[9'b011110010] =
15'b010101101000101;
    cos_mem[9'b011110011] = 15'b010111100000110; sin_mem[9'b011110011] =
15'b010101101101010;
    cos_mem[9'b011110100] = 15'b010111011100100; sin_mem[9'b011110100] =
15'b010101110001111;
    cos_mem[9'b011110101] = 15'b010111011000010; sin_mem[9'b011110101] =
15'b010101110110100;
    cos_mem[9'b011110110] = 15'b010111010011111; sin_mem[9'b011110110] =
15'b010101111011000;
    cos_mem[9'b011110111] = 15'b010111001111101; sin_mem[9'b011110111] =
15'b010101111111101;
    cos_mem[9'b011111000] = 15'b010111001011010; sin_mem[9'b011111000] =
15'b010110000100001;
    cos_mem[9'b011111001] = 15'b010111000110111; sin_mem[9'b011111001] =
15'b010110001000110;
    cos_mem[9'b011111010] = 15'b010111000010101; sin_mem[9'b011111010] =
15'b010110001101010;
    cos_mem[9'b011111011] = 15'b010110111110010; sin_mem[9'b011111011] =
15'b010110010001110;
    cos_mem[9'b011111100] = 15'b010110111001111; sin_mem[9'b011111100] =
15'b010110010110010;
    cos_mem[9'b011111101] = 15'b010110110101011; sin_mem[9'b011111101] =
15'b010110011010110;
    cos_mem[9'b011111110] = 15'b010110110001000; sin_mem[9'b011111110] =
15'b010110011111010;
    cos_mem[9'b011111111] = 15'b010110101100101; sin_mem[9'b011111111] =
15'b010110100011110;
    cos_mem[9'b100000000] = 15'b010110101000001; sin_mem[9'b100000000] =
15'b010110101000001;
end

//address calculation for bottom value
addr_selector ADS0 (
    .angle (angle),
    .angle_shift (1'b0),
    .cos_en (cos_en),
    .mem_address (mem_adr0),
    .do_except (except0),
    .select_cos_mem (select_cos_mem0),
    .select_positive (select_positive0)
);

//address calculation for upper value
addr_selector ADS1 (
    .angle (angle),
    .angle_shift (1'b1),
    .cos_en (cos_en),
    .mem_address (mem_adr1),
    .do_except (except1),
```

```
    .select_cos_mem (select_cos_mem1),
    .select_positive (select_positive1)
);

//calculate upper value
always @(mem_adr1 or mem_adr1 or select_cos_mem1 or select_positive1) begin
    //read memory
    if (select_cos_mem1 == 1'b1) begin
        mem_val1 = cos_mem[mem_adr1];
    end else begin
        mem_val1 = sin_mem[mem_adr1];
    end
    if (select_positive1 == 1'b1) begin
        result1 = {mem_val1[14], mem_val1[14] | except1, mem_val1[13:0]};
    end else begin
        result1 = -{mem_val1[14], mem_val1[14] | except1, mem_val1[13:0]};
    end
end

//calculate bottom value
always @(mem_adr0 or mem_adr0 or select_cos_mem0 or select_positive0) begin
    //read memory
    if (select_cos_mem0 == 1'b1) begin
        mem_val0 = cos_mem[mem_adr0];
    end else begin
        mem_val0 = sin_mem[mem_adr0];
    end
    if (select_positive0 == 1'b1) begin
        result0 = {mem_val0[14], mem_val0[14] | except0, mem_val0[13:0]};
    end else begin
        result0 = -{mem_val0[14], mem_val0[14] | except0, mem_val0[13:0]};
    end
end

//write output
always @(result1 or result0) begin
    result2x = {result1[15], result1} + {result0[15], result0};
    result = result2x[16:1]; //devide by 2
end

endmodule
```