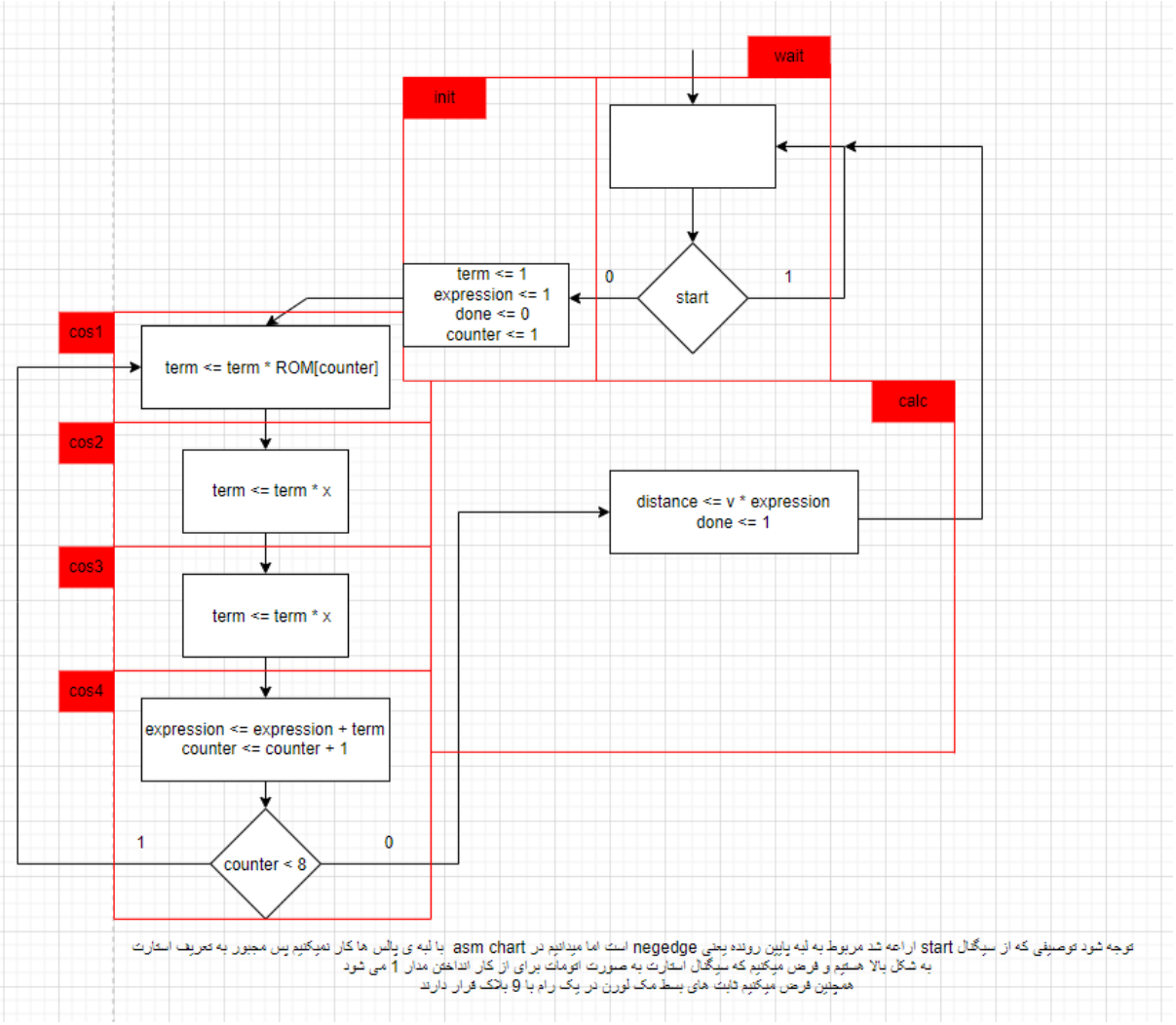
در این تمرین قصد داشتیم تا یک lateral distance calculator برای اندازه گیری فاصله طولی که یک موشک در یک واحد زمانی طی میکند را طراحی و شبیه سازی کنیم. در ادامه به شرح قسمت های مختلف ماژول LateralDistanceCalc و بعد test bench مربوط به آن می پردازیم.

Asm chart :

تصویر پایین asm chart مدار مورد نظر است که از 7 asm block تشکیل شده است.
در استیت اول منتظر میمانیم تا استفاده کننده سیگنال start را صفر کند. توجه شود در asm chart با لبه پالس کار نمیکنیم بنابراین ناچار بودیم به این گونه استیت wait را در **asm chart** نشان دهیم. در کد وریلاگ از negedge بهره برده ایم.
در استیت init مقدار های اولیه را سِت میکنیم و done را صفر میکنیم. دلیل اینکه دو استیت اول مجزا هستند به دلیل تداخل پیدا کردن سیگنال های کنترلی با فرایند بازگشت به استیت wait است.
در مراحل cosi به محاسبه ی کسینوس میپردازیم. تا زمانی که 7 مرحله تکرار نداشته باشیم(توجه شود یک مرحله با مقدار دهی در استیت init طی میشود) باید به محاسبه ی کسینوس بپردازیم.
بعد از محاسبه ی کسینوس به محاسبه ی distance میپردازیم و سیگنال done را فعال میکنیم.
توجه شود فرض میکنیم حافظه ی ROM ضرایب مورد نیاز در هر مرحله حلقه را دارد.
بعد از اتمام به استیت wait بازگشته و منتظر آمدن پالس start مورد نظر میشویم. در واقع فرض میشود که استفاده کننده به صورت خودکار در یک پالس نامعین قبل شروع مدار start را یک میکند که با توجه به محدودیت asm chart بجای لبه با مقدار آن را نشان دادیم.



ماژول LateralDistanceCalc :

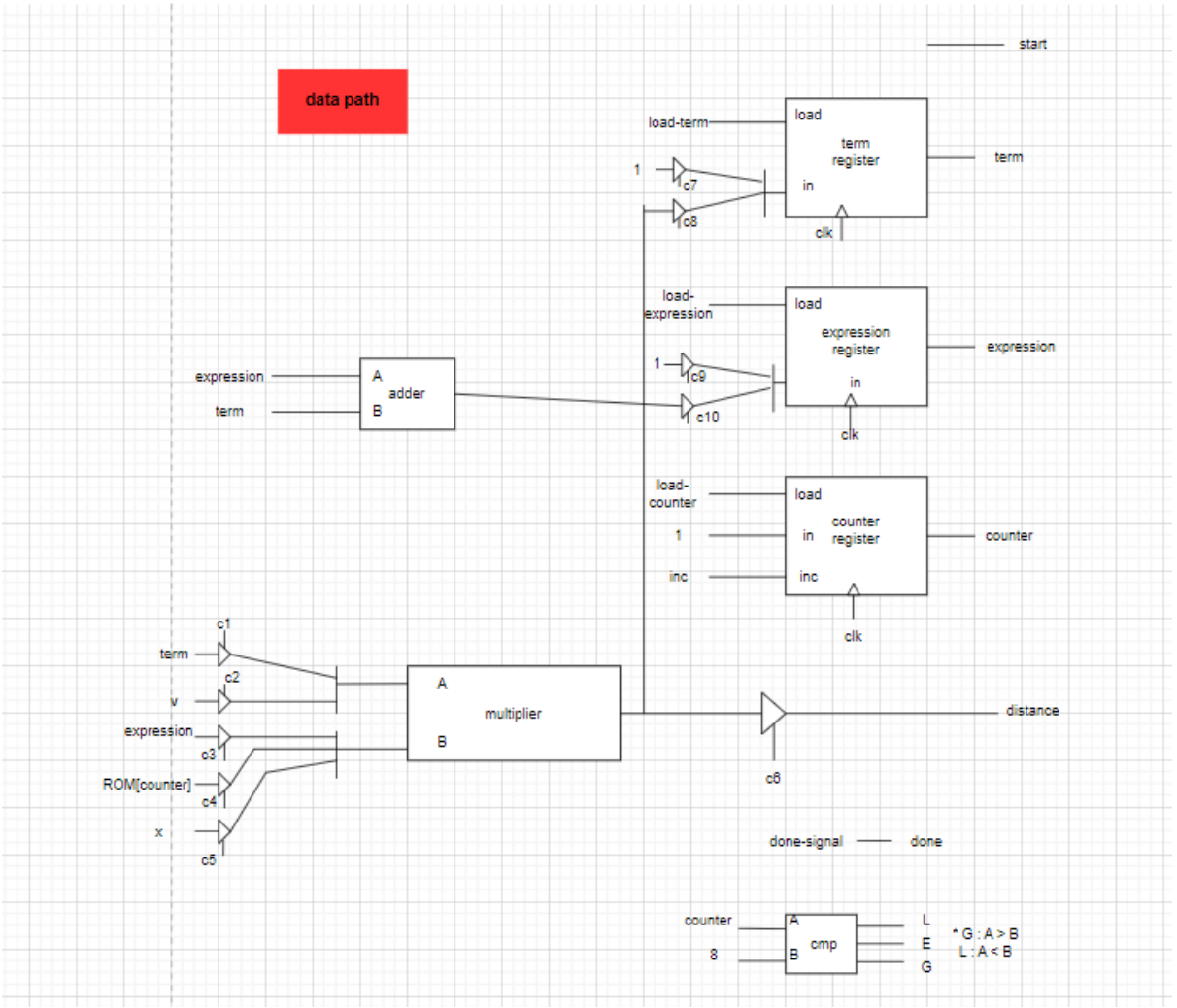
ماژول اصلی برنامه ی ما این ماژول است. در واقع بلاکی که به عنوان محصول نهایی است این بلاک است.

این ماژول از یک data path ساختاری و یک controll unit رفتاری تشکیل شده است.

- Data path : برای بخش data path چون ساختاری بود از ماژول جدیدی استفاده نکردم(نحوه ی خاصی در داک تمرین ذکر نشده است) و مستقیم در بدنه ی ماژول LDC آمدم و از ماژول های مورد نیازم instance گرفتم و با استفاده از wire آن ها را به هم وصل کردم. با توجه به تصویری که مربوط به data path من در asm است ، من از ماژول های custom برای tri state buffer استفاده کردم چون میخواستم ورودی و خروجی بافر vector باشد.

نحوه ی عملکرد DP نیز مشابه تصویر است و به این شکل است که با استفاده از سیگنال های command در هر کلاک یکی از استیت های مشخص شده در asm chart را اجرا می کند. این سیگنال ها در بخش کنترل یونیت توضیح داده خواهند شد. تنها سیگنال status مربوط به سیگنال L است که مقدار موجود در شمارنده ی counter را با عدد 8 مقایسه

میکند تا ببیند همچنان باید در حلقه بماند و term حساب کند یا میتواند وارد مرحله ی محاسبه ی نهایی شود.
ورودی های adder همواره شامل term ,expression است چون تنها در مرحله cos4 باید جمع بزنیم. همچنین خروجی آن تنها به bus ورودی expression register وصل میشود. ورودی دیگر آن میتواند 1 برای استیت init باشد.
اما ورودی های multiplier بسته به اینکه در کدام استیت هستیم متفاوت است و مجبور به استفاده از tri state buffer هستیم.(در asm block های cos1, cos2, cos3, calc نیاز به ضرب کننده داریم). خروجی ضرب کننده به یک tri state بافر وصل میشود تا اگر در مرحله ای غیر init باشیم ، حاصل ضرب را در صورت فعال بودن سیگنال لود در term قرار دهد. همچنین یک سر دیگر خروجی به distance وصل است و تنها زمانی مقدار خروجی را میدهد که high Z نباشد و این یعنی باید در استیت calc قرار بگیرد.
شمارنده ی ما نیز به غیر از استیت init که در آن 1 لود میشود، دارای یک سیگنال inc است که مقدار درون آن را یکی زیاد میکند.
بلاک cmp برای مقایسه ی counter و عدد 8 برای تکرار حلقه است.
هرگاه جواب حاضر شود یک سیگنال از cu می آید و done را فعال میکند(در طراحی done از فلیپ فلاپ استفاده نکردیم و سیگنال به صورت پیوسته از cu می آید)



ماژول TriState :

همانطور که گفتم به دلیل محدودیتی که ماژول primitive bufif1 دارد و نمیگذارد وکتور به عنوان ورودی داده شود این ماژول را تعریف کردم. نحوه توصیف نیز دیتافلو است.

```
module TriState(out, in, control);
    input [15:0] in;
    output [15:0] out;
    input control;

    assign out = control ? in : 16'Bzzzzzzzzzzzzzzz;
endmodule
```

ماژول ROM :

این ماژول ضرایب (2i-1) * (2i) را در خود نگه میدارد.

```
module ROM(address, out);
    input [3:0] address;
    output reg [15:0] out;

    always @(address)
        begin
            if (address == 1) out = 16'Hfc00;
            else if (address == 2) out = 16'Hff55;
            else if (address == 3) out = 16'Hffbc;
            else if (address == 4) out = 16'Hffdb;
            else if (address == 5) out = 16'Hffe9;
            else if (address == 6) out = 16'Hfff0;
            else if (address == 7) out = 16'Hfff5;
        end
endmodule
```

ماژول FixedPointAdder :

جمع در اعداد fixed point مشابه جمع عادی است. بنابراین صرفا با یک توصیف جریان داده آن را طراحی کردم.

```
module FixedPointAdder(in1, in2, out);
    input [15:0] in1;
    input [15:0] in2;
    output [15:0] out;

    assign out = in1 + in2;
endmodule
```

ماژول FixedPointMultiplier :

در این ماژول توجه به دو نکته ضروری است :

1. حاصل ضرب دو عدد 16 بیتی ، 32 بیت است. 5 بیت با ارزش به عنوان overflow و 11 بیت کم ارزش به عنوان underflow باید دور ریخته شوند تا 16بیت حاصل شود. این 16 بیت با slicing بدست می آید.
2. چون داده ها به صورت signed هستند نیاز است تا ابتدا هر دو ورودی ماژول مثبت شوند(مکمل2) و سپس ضرب انجام گیرد. همچنین باید علامت اولیه این دو عدد xor شود تا علامت حاصل نهایی تعیین شود. برای مثبت کردن هر ورودی از علامت آن عدد و replication operator برای xor استفاده می شود. برای ضرب قابل سنتز از الگوریتم شیفت و جمع استفاده کردیم. در انتها نیز علامت out را از طریق sign مشخص میکنیم.

```
module FixedPointMultiplier(A, B, out);
    input [15:0] A;
    input [15:0] B;
    output reg [15:0] out;

    reg [15:0] op1;
    reg [15:0] op2;

    reg sign;

    reg [31:0] multiplicand;
    reg [31:0] accumulator;

    reg [5:0] i;

    always @(A or B)
        begin
            sign = A[15] ^ B[15];
            op1 = (A ^ {16{A[15]}}) + A[15];
            op2 = (B ^ {16{B[15]}}) + B[15];

            multiplicand = op1;
            accumulator = 0;

        for (i = 0; i < 16; i = i + 1)
            begin
                if (op2[0] == 1'b1)
                    accumulator = accumulator + multiplicand;

                    multiplicand = multiplicand << 1;
                    op2 = op2 >> 1;
            end
            out = (accumulator[26:11] ^ {16{sign}}) + sign;
        end
endmodule
```

ماژول Register :

دو رجیستر term , expression برای نگه داری داده های 16بیتی داریم. سنکرون است و دارای ورودی آسنکرون reset است. تنها زمانی مقدارش آپدیت میشود که load فعال شود. همچنین اولویت reset در آن بیشتر از load است(با توجه به ترتیب بررسی شروط).همچنین توصیف آن رفتاری است.

```
module Register(clk, load, reset, in, out);
    parameter bandwidth = 16;

    input clk;
    input load;
    input reset;
    input [bandwidth-1:0] in;
    output reg [bandwidth-1:0] out;

    always @(posedge clk or posedge reset)
        begin
            if (reset)
                out = 0;
            else if(load)
                out = in;
        end
endmodule
```

ماژول FourBitCounter یا Counter :

یک شمارنده 4بیتی است(یک بیت احتیاطی برای 8 شدن).سنکرون است و دارای ورودی reset آسنکرون است.دارای سیگنال inc است تا در استیت 1 ، cos4 واحد زیاد شود(در واقع index حلقه را نگه میدارد).

اولویت reset از inc و load بیشتر است.توصیف آن رفتاری است.

```
module FourBitCounter(clk, load, reset, inc, in, out);
    input clk;
    input load;
    input reset;
    input inc;
    input [3:0] in;
    output reg [3:0] out;

    always @(posedge clk or posedge reset)
        begin
            if (reset)
                out = 0;
            else if (inc)
                out = out + 1;
            else if(load)
                out = in;
        end
endmodule
```

ماژول CMP :

این ماژول وظیفه ی مقایسه ی دو ورودی را دارد که ما تنها با سیگنال L به معنای کمتر کار میکنیم.توصیف آن جریان داده است.

```
module CMP(A, B, L, E, G);
    input [3:0] A;
    input [3:0] B;
    output L, E, G;

    assign L = A < B ? 1 : 0;
endmodule
```

ماژول ControlUnit :

ابتدا به توضیح CU مطابق asm chart میپردازیم.

7 تا asm block داریم پس به روش one's hot باید 7 فلیپ فلاپ داشته باشیم، این فلیپ فلاپ ها را در قالب یک رجیستر 7 بیتی پیاده سازی کردیم.

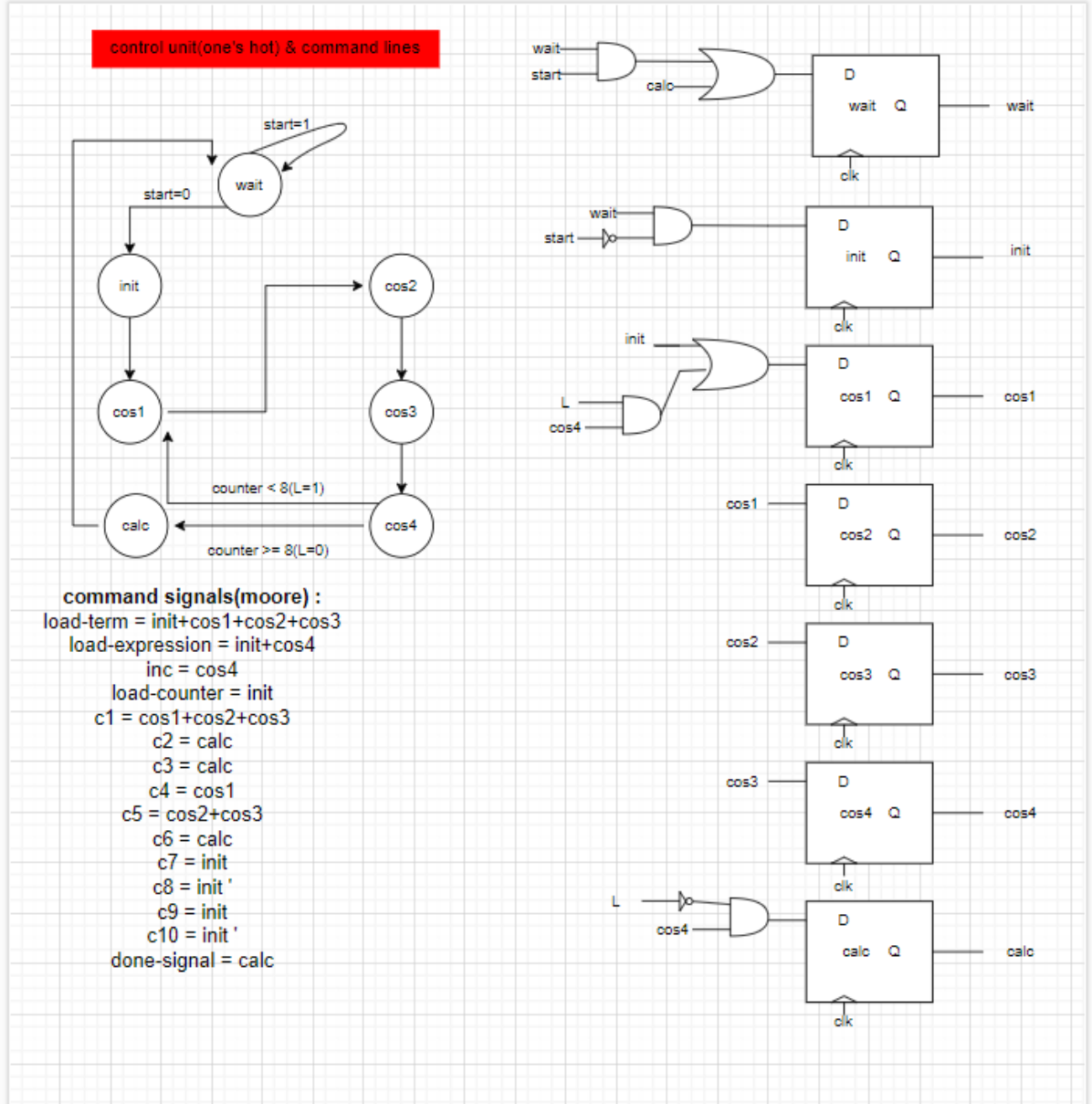
طبق state diagram میتوان ورودی D هر فلیپ فلاپ را بدست آورد.مجددا یادآور میشویم که فرض بر این است که بعد انجام عملیات یا زمانی که سیگنال آسنکرون rst فعال شد ، به استیت wait میرویم تا کاربر سیگنال start با لبه پایین رونده را ارسال کند.

سیگنال های command ، خروجی های مدار هستند و بنابر داک تمرین CU باید موور باشد پس همانطور که میبینیم خروجی ها تنها وابسته به استیتی که در آن هستیم هستند.

سیگنال های ci در واقع کنترلر های tri state buffer هستند. در کد وریلاگ سیگنال های , c9 c10 را نداریم چون مشابه c7 ,c8 هستند.

همچنین سیگنال هایی برای load رجیستر ها و inc موجود است که بر اساس اینکه در کدام استیت عمل لود انجام میشود ، عبارات منطقی را نوشتیم.(توجه شود inc شمارنده به load نیاز ندارد)

- **توضیح کد وریلاگ :** دو بلاک always داریم ، یکی حساس به لبه پایین رونده سیگنال start و دیگری حساس به لبه بالارونده clk و rst. در بلاک اول هرگاه سیگنال start از 1 به 0 تغییر وضعیت داد سیگنال های مناسب برای استیت دوم یعنی init حاضر میشوند. در بلاک دیگر اگر rst فعال شود(دارای اولویت بالا) آنگاه همه سیگنال ها غیرفعال و به استیت اول(wait) میرویم و منتظر لبه پایین رونده ی start میشویم. در غیر اینصورت بر اساس شرط در توصیف رفتاری ، از طریق وضعیت فعلی ، استیت بعدی و سیگنال های مورد نیازش را تعیین میکنیم. در استیت cos4 از طریق تنها ورودی status یعنی L متوجه میشویم که حلقه تمام شده یا نه.




```
module ControlUnit(clk, rst, start, L, load_term, load_expression, inc, load_counter, c1, c2, c3, c4, c5, c6, c7, c8, done_signal);
    input start, L, clk, rst;
    output reg load_counter, load_expression, load_term;
    output reg c1, c2, c3, c4, c5, c6, c7, c8, done_signal, inc;

    reg [6:0] state = 7'B0000001; //one's hot

    // start for CU
    always @(negedge start)
        begin
            c1 = 0;
            c2 = 0;
            c3 = 0;
            c4 = 0;
            c5 = 0;
            c6 = 0;
            c7 = 1;
            c8 = 0;
            load_term = 1;
            load_expression = 1;
            inc = 0;
            load_counter = 1;
            done_signal = 0;
            state = 7'B0000010;
        end
```

```
// control unit
always @(posedge clk or posedge rst)
    begin
        //if (state != 1) $display("state : %d -> ",state);
        if (rst)
            begin
                c1 = 0;
                c2 = 0;
                c3 = 0;
                c4 = 0;
                c5 = 0;
                c6 = 0;
                c7 = 0;
                c8 = 0;
                load_term = 0;
                load_expression = 0;
                inc = 0;
                load_counter = 0;
                done_signal = 0;
                state = 7'B0000001;
            end
        else
            begin
                if (state == 7'B0000010)
                    begin
                        c1 = 1;
                        c2 = 0;
                        c3 = 0;
                        c4 = 1;
                        c5 = 0;
                        c6 = 0;
                        c7 = 0;
                        c8 = 1;
                        load_term = 1;
                        load_expression = 0;
                        inc = 0;
                        load_counter = 0;
                        done_signal = 0;
                        state = 7'B0000100;
                    end
                else if (state == 7'B0000100)
                    begin
                        c1 = 1;
                        c2 = 0;
                        c3 = 0;
                        c4 = 0;
                        c5 = 1;
                        c6 = 0;
                        c7 = 0;
                        c8 = 1;
                        load_term = 1;
                        load_expression = 0;
                        inc = 0;
                        load_counter = 0;
                        done_signal = 0;
                        state = 7'B0001000;
                    end
                else if (state == 7'B0001000)
                    begin
                        c1 = 1;
                        c2 = 0;
                        c3 = 0;
                        c4 = 0;
                        c5 = 1;
                        c6 = 0;
                        c7 = 0;
                        c8 = 1;
                        load_term = 1;
                        load_expression = 0;
                        inc = 0;
                        load_counter = 0;
                        done_signal = 0;
                        state = 7'B0010000;
                    end
                else if (state == 7'B0010000)
                    begin
                        c1 = 0;
                        c2 = 0;
                        c3 = 0;
                        c4 = 0;
                        c5 = 0;
```

```
                        c6 = 0;
                        c7 = 0;
                        c8 = 1;
                        inc = 1;
                        load_term = 0;
                        load_expression = 1;
                        load_counter = 0;
                        done_signal = 0;
                        state = 7'B0100000;
                    end
                else if (state == 7'B0100000)
                    begin
                        if (L)
                            begin
                                c1 = 1;
                                c2 = 0;
                                c3 = 0;
                                c4 = 1;
                                c5 = 0;
                                c6 = 0;
                                c7 = 0;
                                c8 = 1;
                                inc = 0;
                                load_term = 1;
                                load_expression = 0;
                                load_counter = 0;
                                done_signal = 0;
                                state = 7'B0000100;
                            end
                        else
                            begin
                                c1 = 0;
                                c2 = 1;
                                c3 = 1;
                                c4 = 0;
                                c5 = 0;
                                c6 = 1;
                                c7 = 0;
                                c8 = 1;
                                load_term = 0;
                                load_expression = 0;
                                inc = 0;
                                load_counter = 0;
                                done_signal = 1;
                                state = 7'B1000000;
                            end
                    end
                else if (state == 7'B1000000)
                    begin
                        c1 = 0;
                        c2 = 1;
                        c3 = 1;
                        c4 = 0;
                        c5 = 0;
                        c6 = 1;
                        c7 = 0;
                        c8 = 1;
                        load_term = 0;
                        load_expression = 0;
                        inc = 0;
                        load_counter = 0;
                        done_signal = 1;
                        state = 7'B0000001;
                    end
            end
    end
endmodule
```

Test bench و خطا :

یک کلاک داریم که بعد هر 10 واحد زمانی not میشود و به تعداد 100000 بار این اتفاق می افتد(با تسک finish شک موجم ناقص می افتد). در همان بلاک initial استارت را 1 می کنیم. در یک بلاک initial دیگر بعد از 10 واحد زمانی start صفر میشود(شروع کار مدار).

```
initial
    begin
        clk = 0;
        rst <= 0;
        start <= 1;
        repeat(100000) #10 clk = ~clk;
    end

initial #10 start <= 0;
```

در ورودی دادن ابتدا زوایای کوچکتر مساوی 90 درجه را به عنوان ورودی دادیم و سرعت 1 است.

```
// x <= 90 , v = 1
        x <= 16'B0000100001010010;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(60 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B0000010000101111;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(30 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B0000110010001111;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(90 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B0000011001001000;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(45 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B0000000000000000;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(0 degree) * v(1m/s) => distance : %b , done = %b", distance, done);
```

سپس زاویه های بزرگتر از 90 درجه و سرعت 1 ، زاویه های بین منفی90 درجه تا 0 و سرعت 1 و زاویه های بین منفی 180 تا منفی 90 با سرعت 1 را به عنوان ورودی میدهیم.

```
        // x > 90 , v = 1
        start <= 1;
        #10 start <= 0;
        x <= 16'B0001000010111000;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(120 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B0001010011110000;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(150 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B0001100100100001;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(180 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        // -90 <= x <= 0 , v = 1
        start <= 1;
        #10 start <= 0;
        x <= 16'B1111101111010001;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(-30 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B1111100110111000;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(-45 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B1111011110101110;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(-60 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B1111001101110001;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(-90 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        // -180 <= x < -90 , v = 1
        start <= 1;
        #10 start <= 0;
        x <= 16'B1110111100111111;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(-120 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B1110101100010000;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(-150 degree) * v(1m/s) => distance : %b , done = %b", distance, done);

        start <= 1;
        #10 start <= 0;
        x <= 16'B1110011011011111;
        v <= 16'B0000100000000000;
        #1000
        $display("cos(-180 degree) * v(1m/s) => distance : %b , done = %b", distance, done);
```

در مرحله بعد ورودی هایی می دهیم که سرعت در آن ها 1 نیست و سرعت منفی نیز تست میکنیم.

```
            // x = 60 , v = 1.5 => different volacity from 1m/s
            start <= 1;
            #10 start <= 0;
            x <= 16'B0000100001010010;
            v <= 16'B0000110000000000;
            #1000
            $display("cos(60 degree) * v(1.5 m/s) => distance : %b , done = %b", distance, done);

            // x = 30 , v = 3 => different volacity from 1m/s
            start <= 1;
            #10 start <= 0;
            x <= 16'B0000010000101111;
            v <= 16'B0001100000000000;
            #1000
            $display("cos(30 degree) * v(3 m/s) => distance : %b , done = %b", distance, done);

            // negative volacity
            start <= 1;
            #10 start <= 0;
            x <= 16'B0000100001010010;
            v <= 16'B1111000000000000;
            #1000
            $display("cos(60 degree) * v(-2 m/s) => distance : %b , done = %b", distance, done);
```

همچنین تاثیر سیگنال آسنکرون rst را بررسی میکنیم.

```
            // asynchronous rst signal , reset LDC and set done signal to 0 and high impedance distance as i used a
tri state buffer for output
            start <= 1;
            #10 start <= 0;
            x <= 16'B0000100001010010;
            v <= 16'B1111000000000000;
            #10 rst = 1;
            rst = 0;
            $display("rst effect => distance : %b , done = %b", distance, done);
```

پالس هایی با طول متفاوت از start را بررسی میکنیم.

```
          // different start pulse
          start <= 1;
          #20 start <= 0;
          x <= 16'B0000011001001000;
          v <= 16'B0010100000000000;
          #1000
          $display("checking different start pulse : cos(45 degree) * v(5 m/s) => distance : %b , done = %b", distance, done);

          // different start pulse and x , v < 0
          start <= 1;
          #15 start <= 0;
          x <= 16'B0001010011110000;
          v <= 16'B1111100000000000;
          #1000
          $display("checking different start pulse : cos(150 degree) * v(-1 m/s) => distance : %b , done = %b", distance, done);

          // different start pulse and unnormal x
          start <= 1;
          #13 start <= 0;
          x <= 16'B0000101000001100;
          v <= 16'B0000100000000000;
          #1000
          $display("checking different start pulse and x = 72 degree : cos(72 degree) * v(1 m/s) => distance : %b , done = %b", distance,
done);
```

و در انتها زاویه های غیر معمول را بررسی میکنیم(یک مورد در بخش قبلی بود)

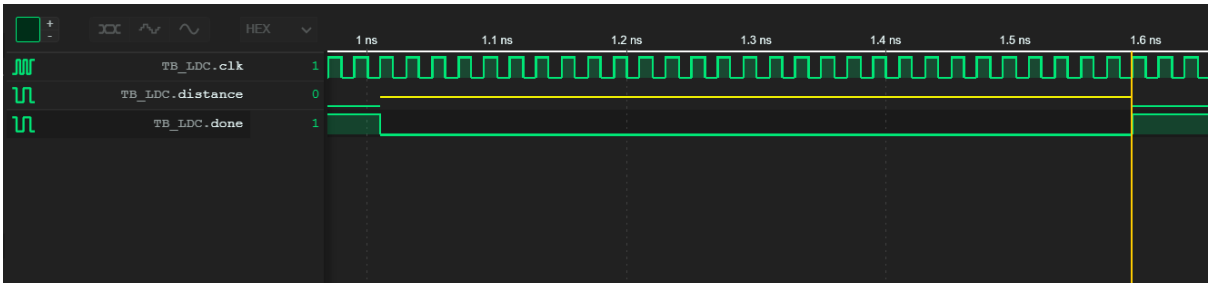
```
// unnormal x
start <= 1;
#10 start <= 0;
x <= 16'B0000001100010000;
v <= 16'B0000100000000000;
#1000
$display("x = 22 degree : cos(22 degree) * v(1 m/s) => distance : %b , done = %b", distance, done);
```

خطا :

خطای محاسبات جدید(آنهایی که برای تست کردن موارد دیگر غیر عملکرد محاسباتی نبودند). در جدول زیر موجود است :

x(degree)	v	output	Actual value	error
60	1	0.505859375	0.5	0.058
30	1	0.86669921875	0.86602540378	0.0006
90	1	0.0009765625	0	0.0009
45	1	0.70703125	0.70710678118	0.0001
0	1	1	1	0
120	1	-0.49560546875	-0.5	0.005
150	1	-0.86328125	-0.86602540378	0.003
180	1	-1	-1	0
-30	1	0.86669921875	0.86602540378	0.0006
-45	1	0.70703125	0.70710678118	0.0001
-60	1	0.505859375	0.5	0.058
-90	1	0.0009765625	0	0.0009
-120	1	-0.49951171875	-0.5	0.0005
-150	1	-0.86328125	-0.86602540378	0.003
-180	1	-0.9892578125	-1	0.0108
60	1.5	0.7587890625	0.75	0.008
30	3	2.60009765625	2.59807621135	0.002
60	-2	-1.01171875	-1	0.01
22	1	0.92724609375	0.92718385456	0.0001
72	1	0.31005859375	0.30901699437	0.001

تعداد سیکل ها : بر اساس شکل موج سیگنال done میتوان تعداد سیکل برای خروجی درست را فهمید به این شکل که تا زمانی که done صفر است یعنی در حال انجام محاسبات هستیم. با توجه به تصویر زیر 30 سیکل تا تولید خروجی درست طول می کشد.(دلیل اینکه طراحی asm یک سیکل(31) بیشتر از شکل زیر دارد این است که در شکل زیر و به طور کلی در اجرا سیکل(ها) مربوط به wait را در نظر نمیگیریم)



پایتون :

برنامه زیر را بر پردازنده intel core-i7 با فرکانس 2.6GHz اجرا کردیم و به مدت زمان 1.9200201034545898 ثانیه زمان اجرا طول کشید.

```
import math
import time
import random

t1 = time.time()
for _ in range(1000000):
    x = random.uniform(-math.pi, math.pi)

    term = 1
    expression = 1
    for i in range(1, 8):
        term = -1 * term / (2 * i * (2 * i - 1))
        term = term * x * x
        expression = expression + term

print(time.time() - t1)
```

FPGA :

با توجه به اینکه هر بار اجرای برنامه به مدت 30 سیکل طول می کشد، یک میلیون اجرا به مدت 30 میلیون سیکل طول میکشد. مدت زمان هر سیکل 5ns است(طبق صورت سوال) پس در مجموع 0.15 ثانیه اجرا برنامه فوق روی FPGA طول می کشد. با توجه به تفاوت 2 ثانیه با 15صدم ثانیه میتوان فهمید که FPGA برای کاربرد های realTime و embedded مناسب تر است.
همچنین چون CPU در یک زمان تنها مشغول به انجام محاسبات مربوط به یک برنامه نیست ، مدت زمان اجرای آن متغیر است درحالی که FPGA که خاص این هدف برنامه ریزی شده است ثبات بیشتری دارد و شرط real time بودن را بهتر پوشش میدهد.

خروجی test bench :

```
VSIM 119> run
# cos(60 degree) * v(1m/s) => distance : 0000010000001100 , done = 1
# cos(30 degree) * v(1m/s) => distance : 0000011011101111 , done = 1
# cos(90 degree) * v(1m/s) => distance : 0000000000000010 , done = 1
# cos(45 degree) * v(1m/s) => distance : 0000010110101000 , done = 1
# cos(0 degree) * v(1m/s) => distance : 0000100000000000 , done = 1
# cos(120 degree) * v(1m/s) => distance : 1111110000001001 , done = 1
# cos(150 degree) * v(1m/s) => distance : 1111100100011000 , done = 1
# cos(180 degree) * v(1m/s) => distance : 1111100000010110 , done = 1
# cos(-30 degree) * v(1m/s) => distance : 0000011011101111 , done = 1
# cos(-45 degree) * v(1m/s) => distance : 0000010110101000 , done = 1
# cos(-60 degree) * v(1m/s) => distance : 0000010000001100 , done = 1
# cos(-90 degree) * v(1m/s) => distance : 0000000000000010 , done = 1
# cos(-120 degree) * v(1m/s) => distance : 1111110000000001 , done = 1
# cos(-150 degree) * v(1m/s) => distance : 1111100100011000 , done = 1
# cos(-180 degree) * v(1m/s) => distance : 1111100000010110 , done = 1
# cos(60 degree) * v(1.5 m/s) => distance : 0000011000010010 , done = 1
# cos(30 degree) * v(3 m/s) => distance : 0001010011001101 , done = 1
# cos(60 degree) * v(-2 m/s) => distance : 1111011111101000 , done = 1
# rst effect => distance : zzzzzzzzzzzzzzz , done = 0
# checking different start pulse : cos(45 degree) * v(5 m/s) => distance : 0001110001001000 , done = 1
# checking different start pulse : cos(150 degree) * v(-1 m/s) => distance : 0000011011101000 , done = 1
# checking different start pulse and x = 72 degree : cos(72 degree) * v(1 m/s) => distance : 0000001001111011 , done = 1
# x = 22 degree : cos(22 degree) * v(1 m/s) => distance : 0000011101101011 , done = 1
```