بسم الله الرحمن الرحیم

پروژه ۶ درس مدار های منطقی

دکتر نوابی

مهدی وجهی

۸۱۰۱۰۱۵۵۸

فهرست

قدم اول: فهم مسئله 3
قدم دوم: طراحی الگوریتم 4
قدم سوم: طراحی Data path 5
مشخص کردن RTL های مورد استفاده 5
کشیدن مدار 6
نوشتن کد system verilog 7
قدم چهارم: طراحی controller 10
کشیدن state machine 10
نوشتن کد 11
قدم پنجم: اتصال دو مدار 12
قدم ششم: بررسی مدار و آزمایش آن 13
قدم هفتم: synthesize 15
پیاده سازی مدار 16
قطعات استفاده شده در مدار 18
آزمایش مدار 19

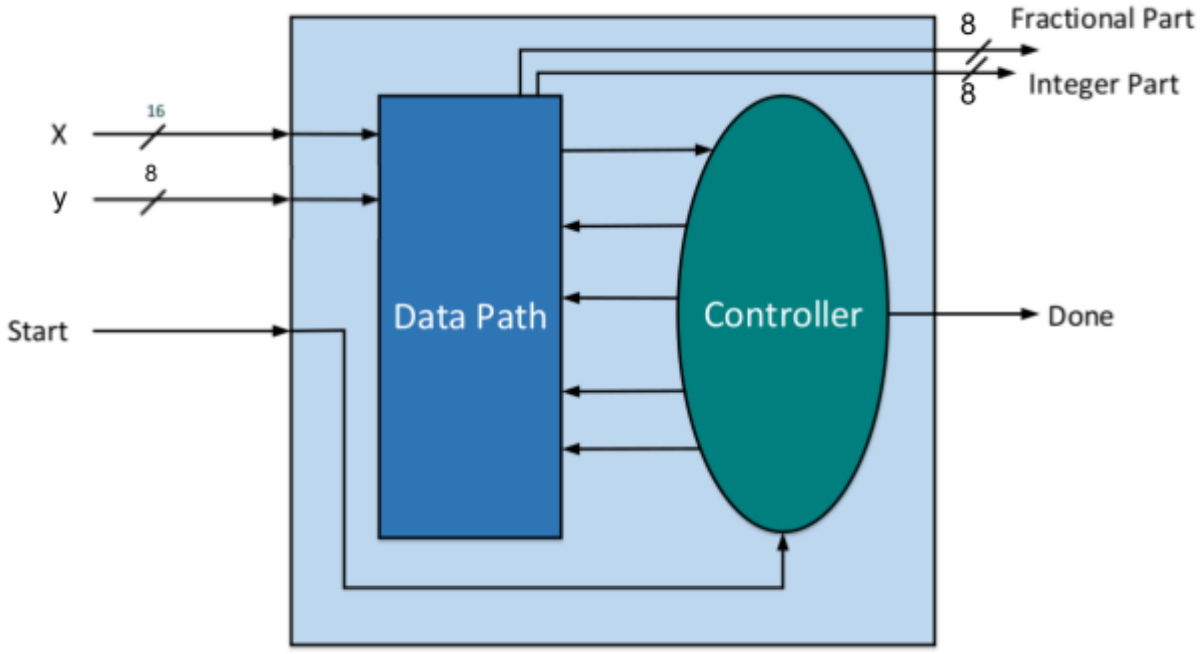
قدم اول: فهم مسئله

در این پروژه خواسته شده که یک مدار محاسبه کننده کسینوس با استفاده از سری تیلور پیاده سازی شود.
برای این کار ابتدا به سری تیلور آن نگاهی می اندازیم:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

$$= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

با توجه به محدودیت های سوال مدار ما به طور کلی به شکل زیر است:



قدم دوم: طراحی الگوریتم

باید سری تیلور بالا را به شکل الگوریتمی تبدیل کنیم که بتوان آن را تبدیل به سخت افزار کرد.

الگوریتم آن به صورت زیر است:

```
tmp = 1
ans = 1
for i in range(8):
      tmp *= rom[i]
      tmp *= x ** 2
      ans += tmp
      if tmp <= y:
            break
```

قدم سوم: طراحی Data path

مشخص کردن RTL های مورد استفاده

در مدار بالا چند چیز مشاهده می شود که در مدار لازم است:

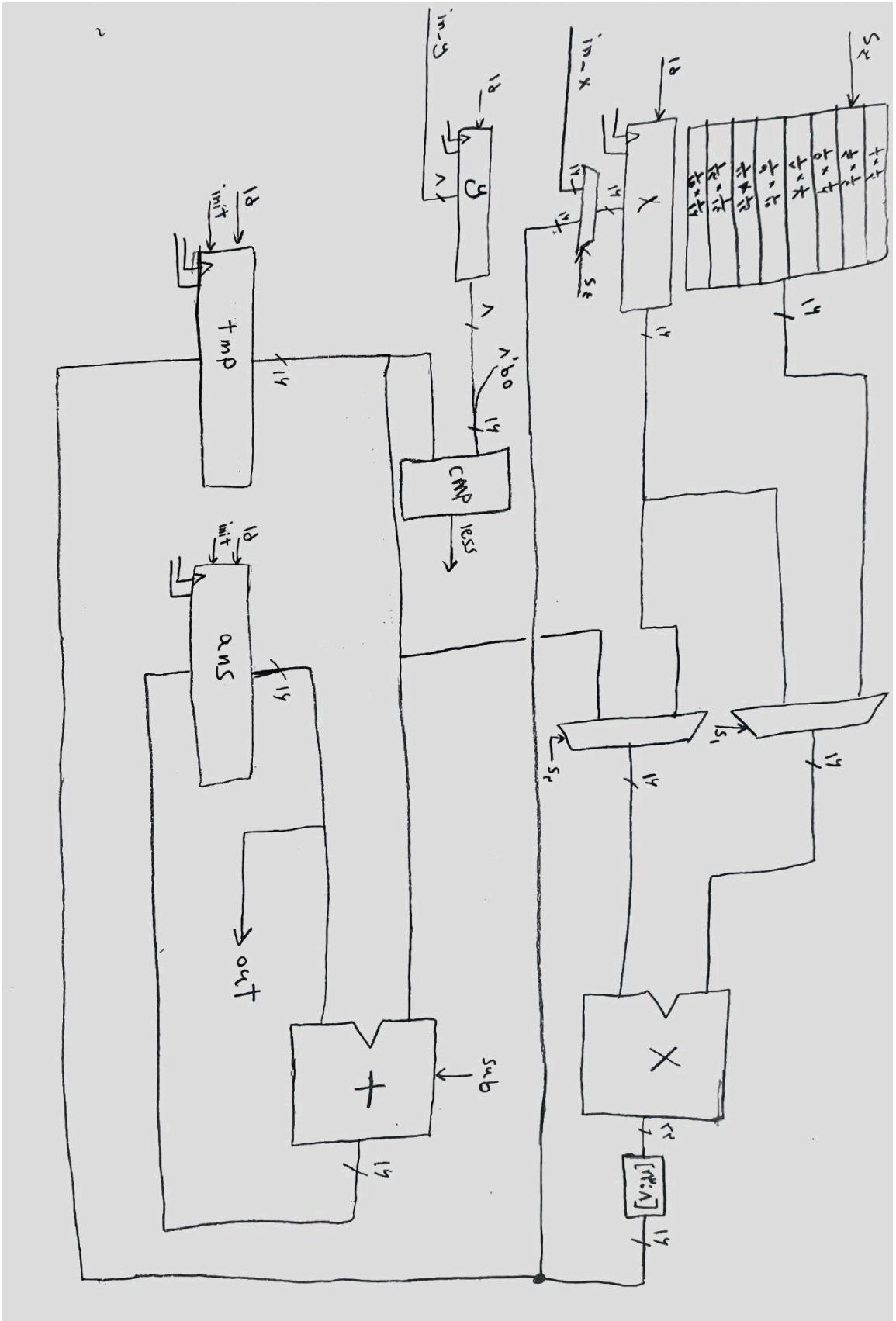
- حافظه برای ذخیره x , y
- حافظه برای tmp, ans
- مدار ضرب برای محاسبه `x ** 2` ، `tmp *= rom[i]` و `tmp *= x ** 2`
- مدار جمع برای محاسبه `ans += tmp`
- مدار مقایسه برای `tmp <= y`
- شمارنده برای i (این مورد در کنترلر قرار داده شده)

با ادغام موارد بالا و محدودیت های سوال به فهرست زیر میرسیم:

نام عنصر	موارد استفاده	اندازه ورودی اول	اندازه ورودی دوم	اندازه خروجی	توضیحات
DFF	x,tmp,ans	۱۶	-	۱۶	سیگنال لود و مقدار دهی اولیه
DFF	y	۸	-	۸	سیگنال لود
ROM	LUT	۳	-	۸	-
ADD	ADD/SUB	۱۶	۱۶	۱۶	سیگنالی برای تعیین این که مدار جمع کند یا تفریق
MULT	به توان رساندن x و ضرب tmp برای تولید مقادیر	۱۶	۱۶	۳۲	در نهایت باید خروجی از ۳۲ بیت به ۱۶ بیست تبدیل شود که با انتخاب ۱۶ بیت وسط این کار انجام می شود
CMP	مقایسه y, tmp	۱۶	۱۶	۱	در صورت کمتر یا مساوی بودن علامت می دهد

البته که در مواردی که یک عنصر چندین استفاده دارد لازم است که این انتخاب ها با استفاده از یک مالتی پلکسر انجام شود.

کشیدن مدار



نوشتن کد system verilog

با توجه به مدار کشیده شده کد برنامه را می نویسیم: (عملکرد هر قسمت از کد با کامنت مشخص شده)

```
module data_path (
   input clk,
   input [15:0] x,
   input s1_rom, s1_x,
   input s2_tmp, s2_x,
   input [2:0] s3,
   input s4_in, s4_mult,
   input ld_tmp, init_tmp,
   input ld_ans, init_ans,
   input ld_x,
   input sub,
   input ld_y,
   input [7:0] in_y,
   output logic [15:0] out_ans,
   output logic [15:0] out_tmp,
   output logic less_cmp
);
   parameter [15:0] default_tmp = 16'b0000000100000000,
                    default_ans = 16'b0000000100000000;
   logic[15:0] rom_out;
   logic [15:0] out_x;
   logic [7:0] out_y;

   //MULT
   wire [31:0] out_mult_32;
   wire [15:0] out_mult;
   wire [15:0] in_mult_1, in_mult_2;

   assign out_mult_32 = in_mult_1 * in_mult_2;
   assign out_mult = out_mult_32[23:8];
   assign in_mult_1 = s1_rom ? rom_out : out_x;
   assign in_mult_2 = s2_x ? out_x : out_tmp;

   //ADD/SUB
   wire [15:0] in_add_1, in_add_2;
   wire [15:0] out_add;
   assign out_add = sub ? in_add_2 - in_add_1 :
                          in_add_2 + in_add_1;
   assign in_add_1 = out_tmp;
   assign in_add_2 = out_ans;

   //CMP
   wire [15:0] in_cmp_1, in_cmp_2;
   assign in_cmp_1 = {8'b0, out_y};
```

```
assign in_cmp_2 = out_tmp;
assign less_cmp = in_cmp_2 <= in_cmp_1;

// reg x
wire [15:0] in_x;
assign in_x = s4_in ? x : out_mult;
always @(posedge clk) begin
    if (ld_x)
        out_x <= in_x;
end

//reg tmp
wire [15:0] in_tmp;
assign in_tmp = out_mult;
always @(posedge clk, posedge init_tmp) begin
    if (init_tmp)
        out_tmp <= default_tmp;
    else if (ld_tmp)
        out_tmp <= in_tmp;
end

//reg ans
wire [15:0] in_ans;
assign in_ans = out_add;
always @(posedge clk, posedge init_ans) begin
    if (init_ans)
        out_ans <= default_ans;
    else if (ld_ans)
        out_ans <= in_ans;
end

//ROM
always @(s3) begin
    case(s3)
        0: rom_out = 16'h0080;
        1: rom_out = 16'h0019;
        2: rom_out = 16'h0008;
        3: rom_out = 16'h0004;
        4: rom_out = 16'h0002;
        5: rom_out = 16'h0001;
        6: rom_out = 16'h0001;
        7: rom_out = 16'h0001;
    endcase
end

//reg y
always @(posedge clk) begin
    if (ld_y)
```

```
            out_y <= in_y;
    end

endmodule
```

ایرادی که این قسمت دارد این است که rom نوشته شده به صورت logic پیاده می شود نه یک عنصر rom برای رفع این مشکل می توان یک hex فایل نوشت که اطلاعات rom در آن ذخیره شود و در کد نیز با دستوری خاص می توان به آن ارجاع داد.

فایل hex فایلی با فرمت .mif و محتوای زیر است:

```
WIDTH=16;
DEPTH=8;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT BEGIN
0 : 0080;                -- memory address : data
1 : 0019;
2 : 0008;
3 : 0004;
4 : 0002;
5 : 0001;
6 : 0001;
7 : 0001;
END;
```

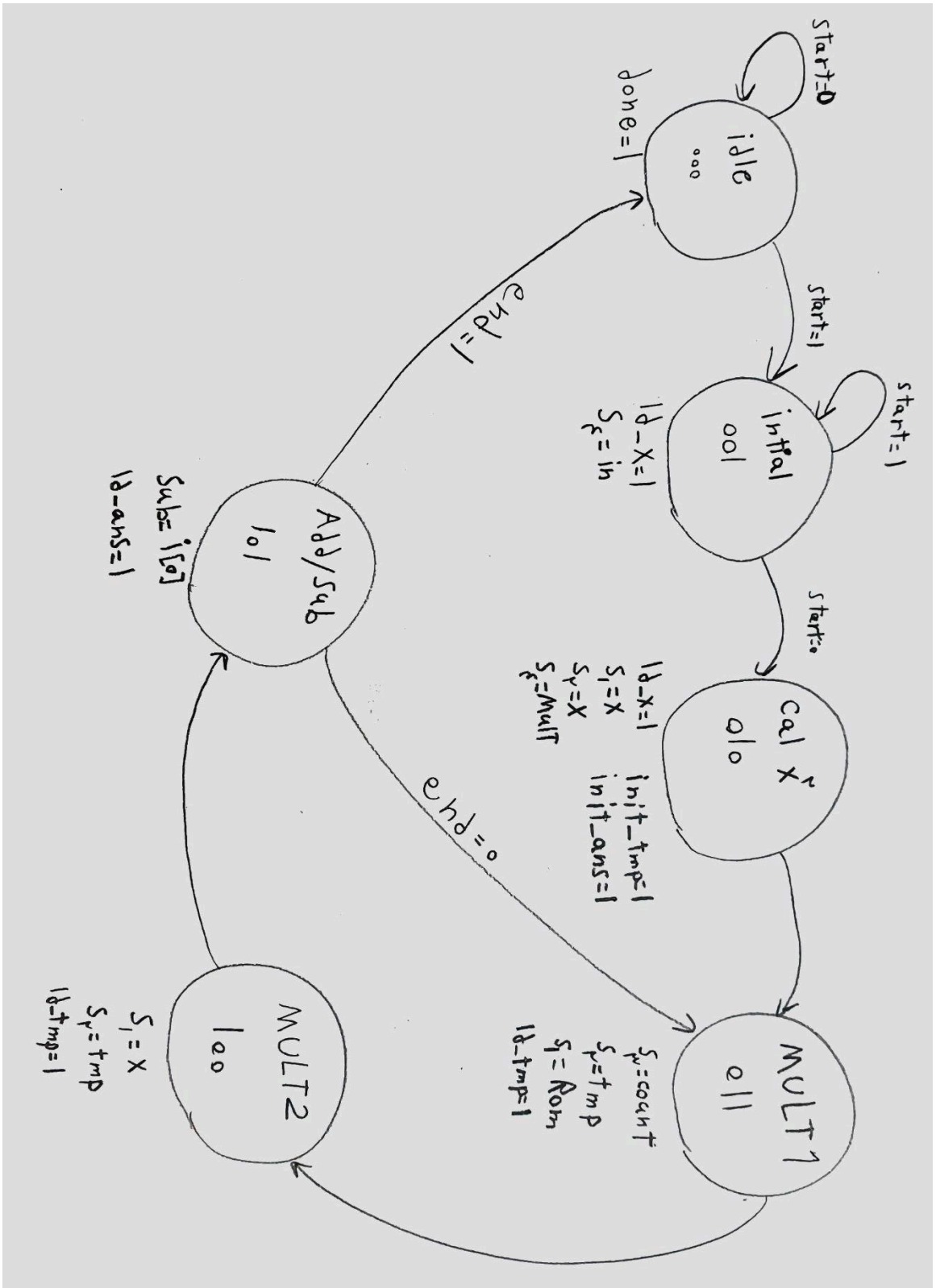
سپس به شکل زیر در کد استفاده می کنیم:

```
(* romstyle = "M9K" *)(* ram_init_file = "NAME.mif" *)  logic  [2:0] rom
[15:0];

      always @ (posedge clk)  q <= rom[addr];
```

قدم چهارم: طراحی controller

کشیدن state machine



نوشتن کد

```
module state (
   input start,
   input[15:0] out_tmp,
   input less_cmp,
   input clk,
   output s1_rom, s1_x,
   output s2_tmp, s2_x,
   output logic [2:0] s3,
   output s4_in, s4_mult,
   output ld_tmp, init_tmp,
   output ld_ans, init_ans,
   output ld_x,
   output sub,
   output ld_y,
   output done
);
   parameter [2:0] s_idle = 3'b000,
                   s_intial = 3'b001,
                   s_cal_x = 3'b010,
                   s_mult_1 = 3'b011,
                   s_mult_2 = 3'b100,
                   s_add = 3'b101;
   logic end_flag;
   logic [2:0] ps, ns;

   // transfer part
   always @(ps, start, end_flag) begin
       case(ps)
       s_idle:     ns = start ? s_intial : s_idle;
       s_intial:   ns = start ? s_intial : s_cal_x;
       s_cal_x:    ns = s_mult_1;
       s_mult_1:   ns = s_mult_2;
       s_mult_2:   ns = s_add;
       s_add:      ns = end_flag ? s_idle : s_mult_1;
       default:    ns = s_idle;
       endcase
   end

   always @(posedge clk) begin
       ps <= ns;
   end

   // out signals
   assign s1_rom = (ps == s_mult_1) ? 1'b1 : 1'b0;
   assign s1_x = (ps == s_cal_x || ps == s_mult_2) ? 1'b1 : 1'b0;
   assign s2_tmp = (ps == s_mult_1 || ps == s_mult_2) ? 1'b1 : 1'b0;
```

```
    assign s2_x = (ps == s_cal_x) ? 1'b1 : 1'b0;
    assign s4_in = (ps == s_intial) ? 1'b1 : 1'b0;
    assign s4_mult = (ps == s_cal_x) ? 1'b1 : 1'b0;
    assign ld_tmp = (ps == s_mult_1 || ps == s_mult_2) ? 1'b1 : 1'b0;
    assign init_tmp = (ps == s_cal_x) ? 1'b1 : 1'b0;
    assign init_ans = (ps == s_cal_x) ? 1'b1 : 1'b0;
    assign ld_ans = (ps == s_add) ? 1'b1 : 1'b0;
    assign ld_x = (ps == s_intial || ps == s_cal_x) ? 1'b1 : 1'b0;
    assign sub = s3[0];
    assign ld_y = (ps == s_intial) ? 1'b1 : 1'b0;
    assign done = end_flag;

    // other parts
    assign end_flag = (s3 == 3'b111 || less_cmp == 1'b1) ? 1'b1 : 1'b0;
    always @(ps) begin
        if (ps == s_idle)
            s3 <= 8'b0;
        else if (ps == s_add)
            s3 <= s3 + 1'b1;
    end
endmodule
```

قدم پنجم: اتصال دو مدار

به آسانی سیگنال های مربوطه را به هم متصل می کنیم:

```
module ca6 (
    input start,
    input clk,
    input [15:0] x,
    input [7:0] in_y,
    output [15:0] out_ans,
    output done
);
    wire s1_rom, s1_x,
         s2_tmp, s2_x,
         s4_in, s4_mult,
         ld_tmp, init_tmp,
         ld_ans, init_ans,
         ld_x,
         sub,
         ld_y,
         less_cmp;
    wire[2:0] s3;
    wire[15:0] out_tmp;
```

```
   data_path dp (clk, x,
                 s1_rom, s1_x,
                 s2_tmp, s2_x,
                 s3,
                 s4_in, s4_mult,
                 ld_tmp, init_tmp,
                 ld_ans, init_ans,
                 ld_x,
                 sub,
                 ld_y,
                 in_y,
                 out_ans,
                 out_tmp,
                 less_cmp
   );

   state s (start,
            out_tmp,
            less_cmp,
            clk,
            s1_rom, s1_x,
            s2_tmp, s2_x,
            s3,
            s4_in, s4_mult,
            ld_tmp, init_tmp,
            ld_ans, init_ans,
            ld_x,
            sub,
            ld_y,
            done
   );
endmodule
```

قدم ششم: بررسی مدار و آزمایش آن

تست بنچ زیر را می نویسیم و شبیه سازی می کنیم:

```
`timescale 1ns/1ns
module tb();
   logic start = 0, clk = 0;
   // pi/3 16'h010c
   // pi/2 16'h0191
   // pi/4 16'h00c8
   // pi/1 16'h0324
   // pi/5 16'h00a0
   logic[15:0] x = 16'h0191;
```

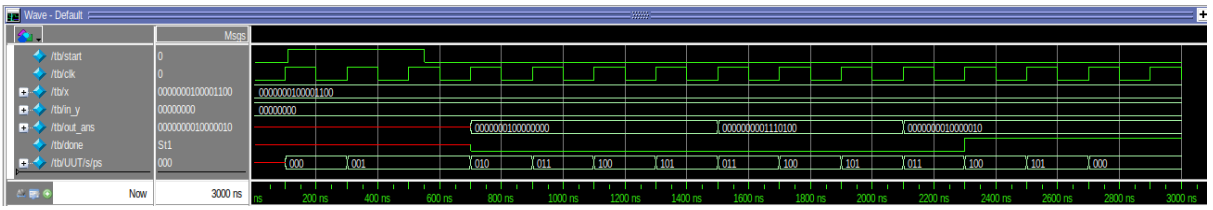
```
    logic[7:0] in_y = 8'b0;
    wire[15:0] out_ans;
    wire done;

    ca6 UUT (start,
             clk,
             x,
             in_y,
             out_ans,
             done);

    always #100 clk = ~clk;

    initial begin
        #110 start = 1;
        #440 start = 0;
    end
endmodule
```

خروجی مدار را برای pi/3 به صورت زیر است:



مقدار pi/3 به hex برابر است با 010c و نتیجه برابر است با ۱/۲ که این مقدار در نتیجه مدار قرار دارد.

بررسی باقی مقادیر نیز صحت عملکرد مدار را نشان می دهد.

قدم هفتم: synthesize

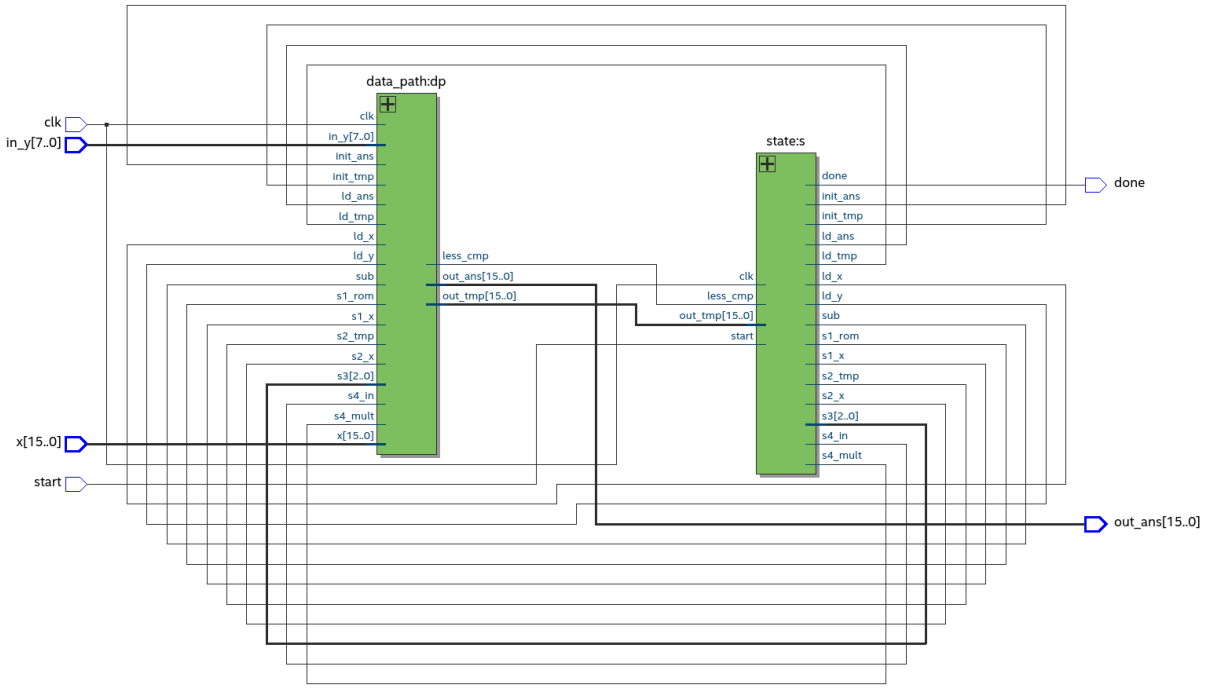
پروژه خود را در کوارتز ایجاد می کنیم سپس فایل های خود را اضافه می کنیم و تنظیمات مربوطه از جمله خروجی sdo و همچنین پیاده سازی state machine را تنظیم می کنیم. سپس خروجی می گیریم.

Flow Summary

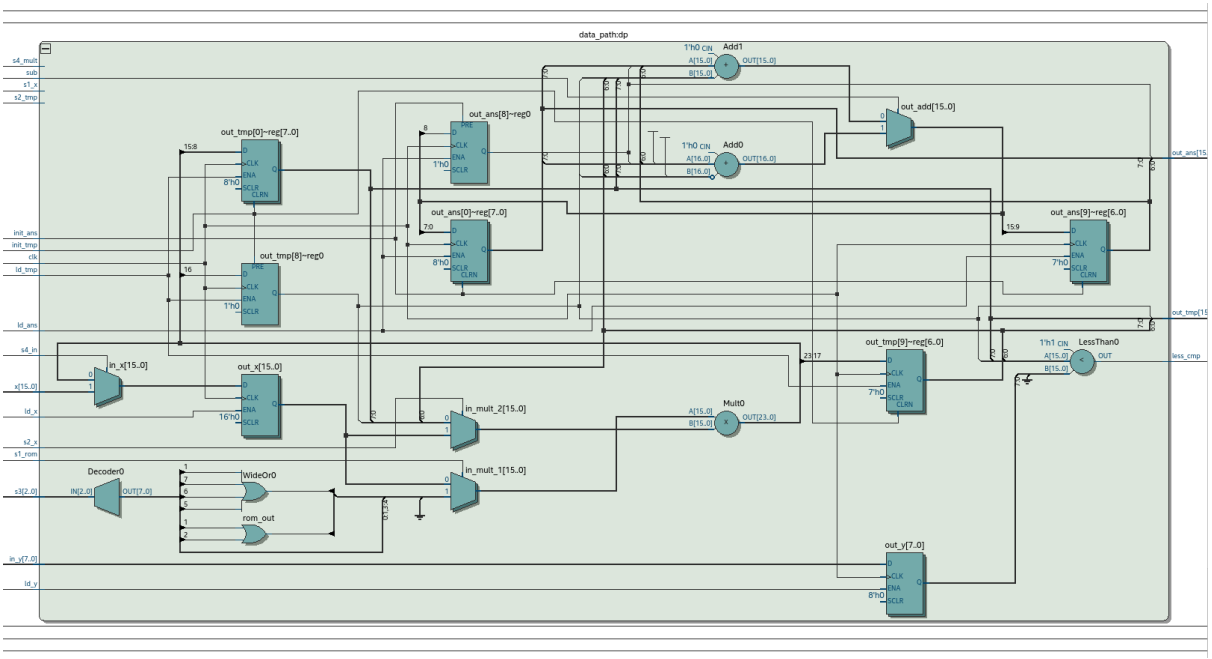
<<Filter>>

Flow Status	Successful - Wed Jan 10 00:55:30 2024
Quartus Prime Version	23.1std.0 Build 991 11/28/2023 SC Lite Edition
Revision Name	ca6
Top-level Entity Name	ca6
Family	Cyclone IV GX
Device	EP4CGX15BF14A7
Timing Models	Final
Total logic elements	396 / 14,400 (3 %)
Total registers	59
Total pins	43 / 81 (53 %)
Total virtual pins	0
Total memory bits	0 / 552,960 (0 %)
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0 / 2 (0 %)
Total GXB Receiver Channel PMA	0 / 2 (0 %)
Total GXB Transmitter Channel PCS	0 / 2 (0 %)
Total GXB Transmitter Channel PMA	0 / 2 (0 %)
Total PLLs	0 / 3 (0 %)

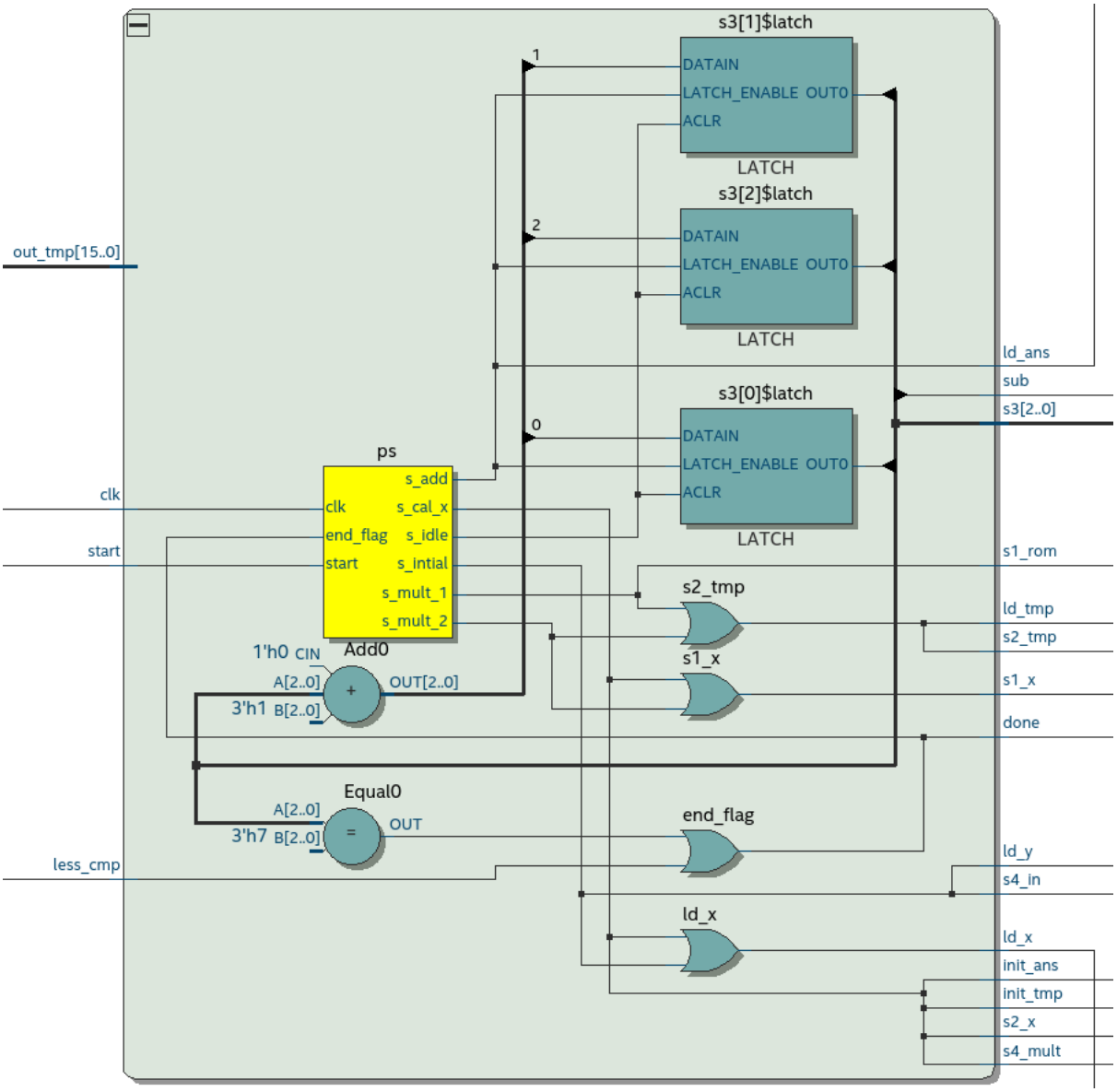
پیاده سازی مدار



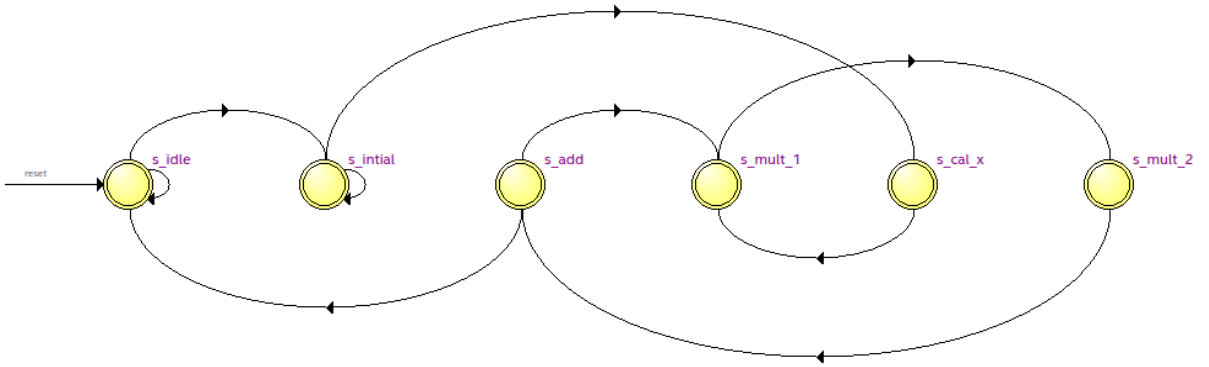
همانطور که انتظار می رفت پیاده سازی مدار به صورت بالا است.



پیاده سازی dp به صورت بالا است همانطور که مشاهده می شود از یک مدار مقایسه استفاده شده و از یک مدار ضرب. اما از دو مدار جمع استفاده شده که به این دلیل است که یکی جمع می کند و دیگری تفریق و همچنین مشاهده می شود که rom به صورت logic پیاده سازی شده است. باقی موارد هم از مدار ما پیروی می کند.



برای بخش controller هم مشاهده می کنید که متشکل از state machine ، یک شمارنده و یک مقایسه کننده است. دلیل استفاده از مقایسه کننده این است که ببیند شمانده ۱ شده است یا خیر که می شد با استفاده از and پیاده شود و نیازی به استفاده از جز دیگر نبود. لازم به ذکر است که با کمی دقت می توان متوجه نحوه پیاده سازی سیگنال های خروجی شد.



	Name	ps~4	ps~3	ps~2
1	s_idle	0	0	0
2	s_intial	0	0	1
3	s_cal_x	0	1	0
4	s_mult_1	0	1	1
5	s_mult_2	1	0	0
6	s_add	1	0	1

	Source State	Destination State	Condition
1	s_add	s_mult_1	(lend_flag)
2	s_add	s_idle	(end_flag)
3	s_cal_x	s_mult_1	
4	s_idle	s_idle	(!start)
5	s_idle	s_intial	(start)
6	s_intial	s_cal_x	(!start)
7	s_intial	s_intial	(start)
8	s_mult_1	s_mult_2	
9	s_mult_2	s_add	

State machine دقیقا از طراحی ما پیروی می کند و نیازی به توضیح ندارد

قطعات استفاده شده در مدار

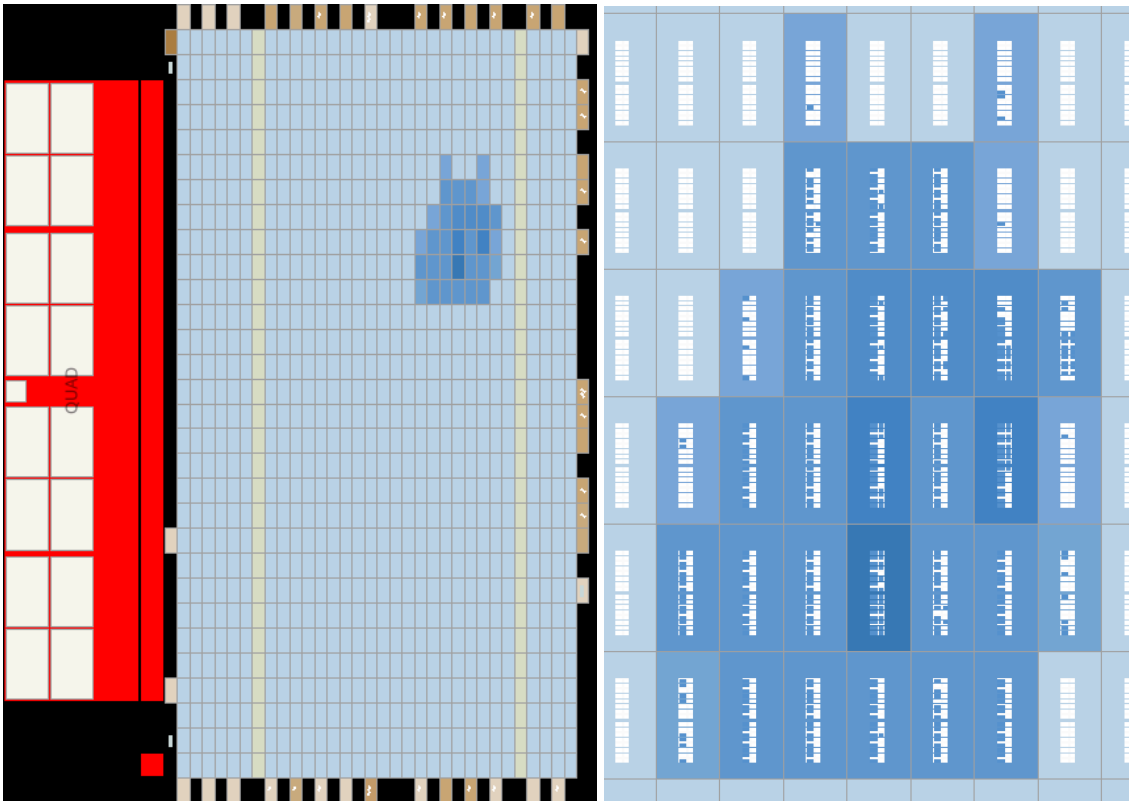
Analysis & Synthesis Summary

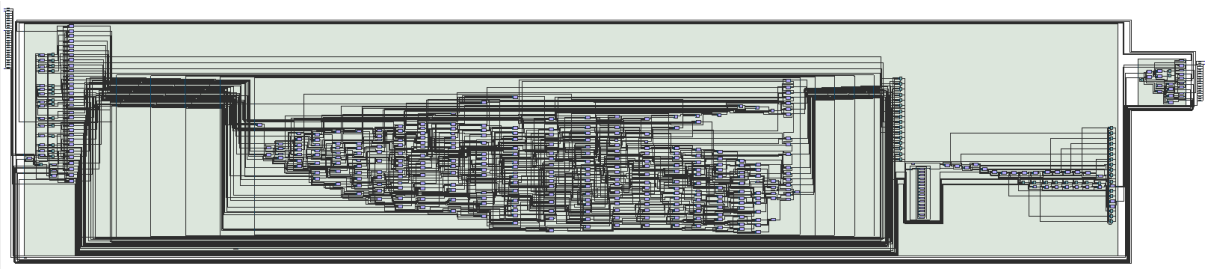
<<Filter>>

Analysis & Synthesis Status	Successful - Wed Jan 10 00:55:10 2024
Quartus Prime Version	23.1std.0 Build 991 11/28/2023 SC Lite Edition
Revision Name	ca6
Top-level Entity Name	ca6
Family	Cyclone IV GX
Total logic elements	403
Total registers	59
Total pins	43
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0
Total GXB Receiver Channel PMA	0
Total GXB Transmitter Channel PCS	0
Total GXB Transmitter Channel PMA	0
Total PLLs	0

می توان دید که از ۵۹ رجیستر استفاده شده که در واقع ۱۶ تا برای x و ۱۶ تا tmp و ۱۶ تا ans و ۸ تا y و ۳ تا برای شمارش است که برابر با این مقدار است. ۴۳ پین هم i/o های دستگاه را نشان می دهند.

مدار روی چیپ به صورت زیر است:





Slow 1200mV -40C Model Fmax Summary

<<Filter>>

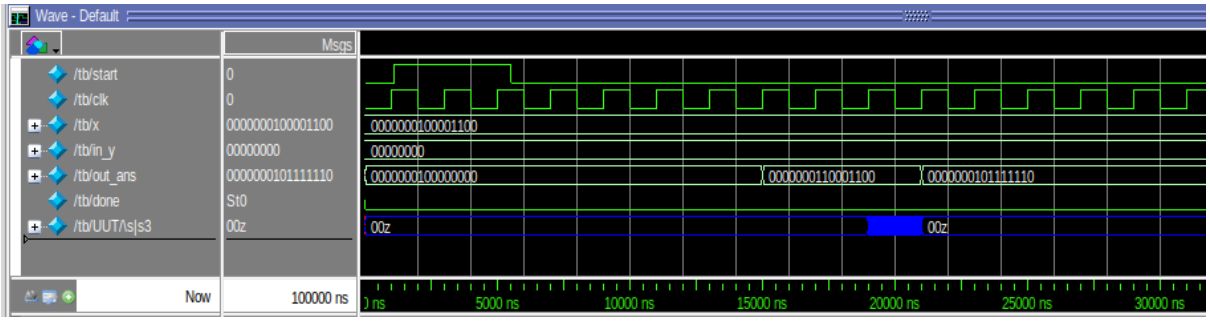
	Fmax	Restricted Fmax	Clock Name
1	93.43 MHz	93.43 MHz	clk
2	216.08 MHz	216.08 MHz	state:s|ps~2

Slow 1200mV 125C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name
1	80.57 MHz	80.57 MHz	clk
2	225.23 MHz	225.23 MHz	state:s|ps~2

آزمایش مدار



مشاهده می شود که کد عملکرد مشابهی ندارد.