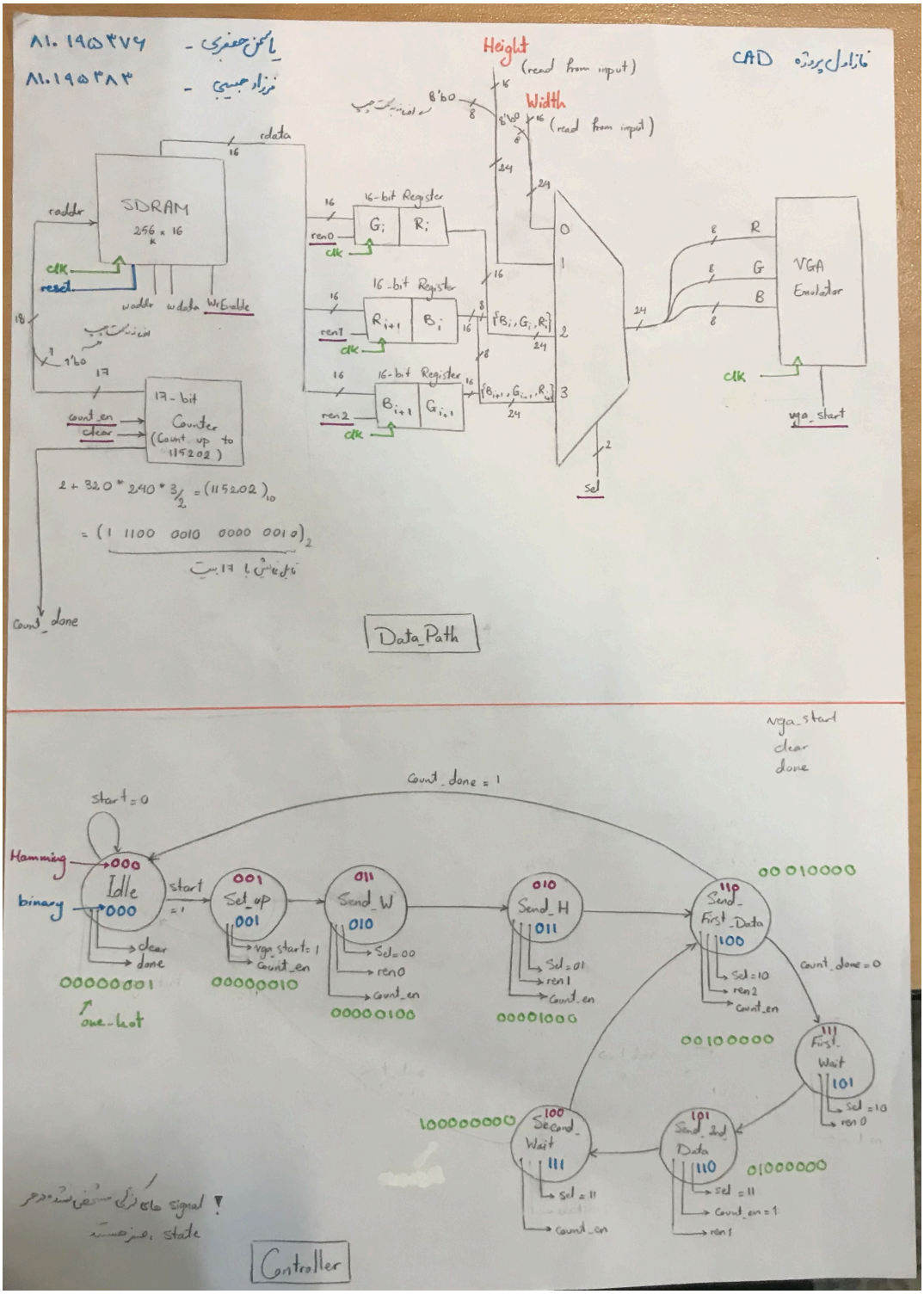
تمرین مقدماتی دوم

فرزاد حبیبی (۸۱۰۱۹۵۳۸۳) ، یاسمن جعفری (۸۱۰۱۹۵۳۷۶)

- برای حل این بخش از کنترلر باینری و مسیرداده‌ی فاز اول خودمان استفاده کردیم که در زیر عکس آن‌ها آمده است.



شبیه‌سازی :

در قسمت شبیه‌سازی ابتدا کنترلر و مسیرداده را در ماژول‌های جداگانه پیاده‌سازی کردیم.

ماژول‌هایی که در مسیرداده به کار رفته‌اند به شکل زیر می‌باشند.

شمارشگر ۱۸ بیتی
وظیفه‌ی این شمارشگر این است که تا زمانی که اطلاعات در sram به پایان نرسیده است در صورت enable بودن آدرس اطلاعات را در sram تولید کند.

```
/*********CounterInterface*******/
input wire clk, clear, count_en;
input wire [15:0] width, height;
output reg [17:0] r_addr;
output reg count_done;
/*******************************/

wire[17:0] limit;
assign limit = width*height*3/2 + 2;
always@(posedge clk)
begin
  if(clear)
    begin
      r_addr <= 18'b0;
      count_done <= 0;
    end

  if(r_addr == limit)
    count_done <= 1;
  else if(count_en)
    r_addr <= r_addr + 1;
end
```

رجیسترهای ۱۶ بیتی
در مسیر داده ۳ رجیستر ۱۶ وجود دارد که کد هر کدام از آن‌ها به شکل زیر می‌باشد.

```
/*****register_16_bit Interface *******/
input wire load_en , clk ;
input wire [15:0] data_in;
output reg [15:0] data_out;
/**************************************/

always@(posedge clk)
begin
  if(load_en)
    data_out <= data_in;
  else
    data_out <= data_out;
end
```

این رجیسترها وظیفه‌ی ذخیره‌ی خروجی sram را دارند.

مالتی‌پلکسر
این مالتی‌پلکسر از بین ۴ ورودی ۲۳ بیتی یک خروجی ۲۳ بیتی می‌دهد.

```
input wire [23:0] in0, in1, in2, in3;
input wire [1:0] sel;
output wire [23:0] out;

genvar index;
generate
for (index=0; index < 24; index=index+1)
  begin : mult
    multiplexer_4_to_1 m(in0[index], in1[index], in2[index], in3[index], sel, out[index]);
  end
endgenerate
```

این مالتی‌پلکسر در واقع از ۲۴ مالتی پلکسر یک‌بیتی به وجود آمده است که کد وریلاگ هر کدام به شکل زیر می‌باشد.

```
/*********Multiplexer Interface*******/
input wire in0, in1, in2, in3;
input wire [1:0] sel;
output wire out;
/*************************************/

wire [1:0] sel_not;
not(sel_not[0], sel[0]);
not(sel_not[1], sel[1]);

wire [3:0] q;
and(q[0], in0, sel_not[0], sel_not[1]);
and(q[1], in1, sel[0], sel_not[1]);
and(q[2], in2, sel_not[0], sel[1]);
and(q[3], in3, sel[0], sel[1]);

or(out, q[0], q[1], q[2], q[3]);
```

و در نهایت مسیرداده به شکل زیر می‌باشد.

```
/******DataPath Interface*****/
input wire clk, count_en, clear, len0, len1, len2, vga_start;
input wire [1:0] sel;
input wire [15:0] width, height;
input wire [15:0] rdata;
output wire cnt_done;
output wire [17:0] r_addr;
output wire [23:0] mux_out;
/*****************************/
counter_18bit cn(.clk(clk), .count_en(count_en), .clear(clear),
                 .width(width), .height(height), .r_addr(r_addr), .count_done(cnt_done));

wire [15:0] rout0;
register_16_bit r0(.load_en(len0), .clk(clk), .data_in(rdata), .data_out(rout0));

wire [15:0] rout1;
register_16_bit r1(.load_en(len1), .clk(clk), .data_in(rdata), .data_out(rout1));

wire [15:0] rout2;
register_16_bit r2(.load_en(len2), .clk(clk), .data_in(rdata), .data_out(rout2));

multiplexer mux(.in0({8'b0, width}), .in1({8'b0, height}), .in2({rout1[7:0], rout0}),
                .in3({rout2, rout1[15:8]}), .sel(sel), .out(mux_out));
```

تنها ماژول رفتاری‌ای که در کنترلر به کار رفته است فیلپ‌فلاپ می‌باشد که به شکل زیر نوشته شده است :

```
module flipflop(input wire in, clk, rst , output reg out);
  always@(posedge clk)
  begin
    if(rst)
      out <= 1'b0;
    else
      out <= in;
  end
endmodule
```

در نهایت ماژول کنترلر براساس فاز قبلی به شکل زیر پیاده‌سازی می‌شود :

```
/********Controller Interface*******/
input wire clk, rst;
input wire start, count_done;
output wire clear, done, vga_start, count_en, len0, len1, len2;
output wire [1:0] sel;
/*********************************/


//Combinantional
wire [2:0] ps;
wire A, B, C, D, E;
assign A = ps[2];
assign B = ps[1];
assign C = ps[0];
assign D = start;
assign E = count_done;

wire Ap, Cp, Bp, Ep;
not(Ap, A);
not(Cp, C);
not(Bp, B);
not(Ep, E);

wire ApC, ACp, BC, AB, ApCp, BpC, BpCp, BCp, AEp, AC;
wire ABCp, ApBpCp, ApBpC, ApCpD, ACpEp;

// count_en
and(ApC, Ap, C);
and(ACp, A, Cp);
or(count_en, B, ApC, ACp);

//sel[0]
and(BC, B, C);
and(AB, A, B);
or(sel[0], BC, AB);

//sel[1]
assign sel[1] = A;

//len0
and(ApCp, Ap, Cp);
and(BpC, Bp, C);
or(len0, ApCp, BpC);

//len1
and(ABCp, AB, Cp);
or(len1, ApC, ABCp);

//len2
and(BpCp, Bp, Cp);
assign len2 = BpCp;

//done / clear
and(ApBpCp, Ap, Bp, Cp);
assign done = ApBpCp;
assign clear = ApBpCp;

//vga_start
and(ApBpC, Ap, Bp, C);
assign vga_start = ApBpC;

//ns :
wire [2:0] ns;

//ns[0]
and(BCp, B, Cp);
and(ApCpD, Ap, Cp, D);
and(ACpEp, A, Cp, Ep);
or(ns[0], BCp, ApCpD, ACpEp);

//ns[1]
or(ns[1], BpC, BCp);

//ns[2]
and(AEp, A, Ep);
and(AC, A, C);
or(ns[2], BC, AEp, AC, AB);
//=>26 gates<=//


//Sequential
flipflop f1(ns[0], clk, rst, ps[0]);
flipflop f2(ns[1], clk, rst, ps[1]);
flipflop f3(ns[2], clk, rst, ps[2]);
```

در نهایت در ماژول sram_vga_controller هر دو مسیرداده و کنترلر را اضافه کردیم :

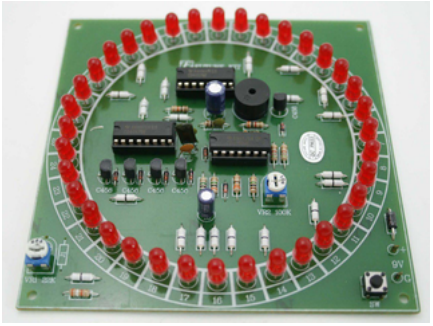
```
wire count_en, count_done, clear, len0, len1, len2;
wire [1:0] sel;
wire [23:0] mux_out;
controller c(.clk(clk), .rst(reset), .start(start), .count_done(count_done),
             .clear(clear), .count_en(count_en), .done(done), .vga_start(vgastart),
             .len0(len0), .len1(len1),  .len2(len2), .sel(sel));

datapath dp(.clk(clk), .count_en(count_en), .clear(clear), .len0(len0),
            .len1(len1), .len2(len2), .sel(sel), .width(IMAGE_WIDTH[15:0]),
            .height(IMAGE_HEIGHT[15:0]), .vga_start(vgastart), .cnt_done(count_done),
            .r_addr(raddr), .rdata(rdata), .mux_out(mux_out));

assign r = mux_out[7:0];
assign g = mux_out[15:8];
assign b = mux_out[23:16];
```

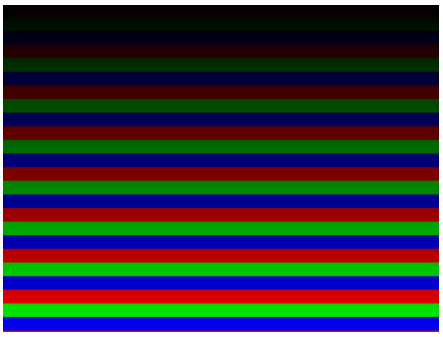
برای شبیه‌سازی فایل‌های وریلاگ را در sim_top.tcl اضافه کردیم.

```
vlog    +acc -incr -source  +define+SIM     $hdl_path/*.v
```

در پایان در برنامه‌ی model sim با اجرای دستور do sim_top.tcl تست‌بنچ را اجرا کرده و خروجی را به‌صورت یک تصویر ذخیره شده در دایرکتوری sim به دست می‌آوریم.

خروجی‌ها هر کدام به شکل زیر می‌باشند :





سنتز :

برای سنتز کردن، تمامی فایل‌های وریلاگ را به نرم‌افزار ISE اضافه کردیم و مدار را سنتز کردیم.
بعد از تمام شدن فرآیند سنتز مدار را مپ کرده و اعداد گزارش شده به شکل زیر می‌باشند.

AVAILABLE	USED	LOGIC UTILIZATION	
27,648	65	NUMBER OF OCCUPIED SLICES	
55,296	78	TOTAL	NUMBER OF SLICE LUTS
-	61	NUMBER USED AS LOGIC	
-	17	NUMBER USED AS MEMORY	
55,296	70	NUMBER OF SLICE REGISTERS	
		NUMBER OF MULT18X18S	

در این مدار تعداد ۷۰ رجیستر به کار رفته است که تحلیل آن‌ها به صورت زیر می‌باشد :

- ۱۸ رجیستر برای شمارشگر ۱۸ بیتی
- ۴۸ رجیستر برای ۳ عدد رجیستری که در مسیر داده وجود دارد
- ۳ رجیستر برای ۳ عدد فلیپ‌فلاپی که در کنترلر وجود دارند
- ۱ رجیستر برای

زمان‌بندی :

با استفاده از Create Timing Constraints ، تایم را ۲۰ نانو ثانیه قرار می‌دهیم تا به فرکانس ۵۰ مگا هرتز برسیم. سپس Rising duty cycle را برابر با ۵۰ درصد قرار می‌دهیم.

اطلاعات زیر را در قسمت Design Overview > Timing constraints قرار داشتند.

Setup slack time	9.447 ns
Hold slack time	1.027 ns
Maximum Frequency	137.589 MHz

سپس با کلیک برروی لینک Constraint و با سپس راست کلیک Maximum Data Path و انتخاب گزینه‌ی showing technology viewr می‌توانیم به critical path دسترسی داشته باشیم که در صفحه‌ی بعد آمده است.

7