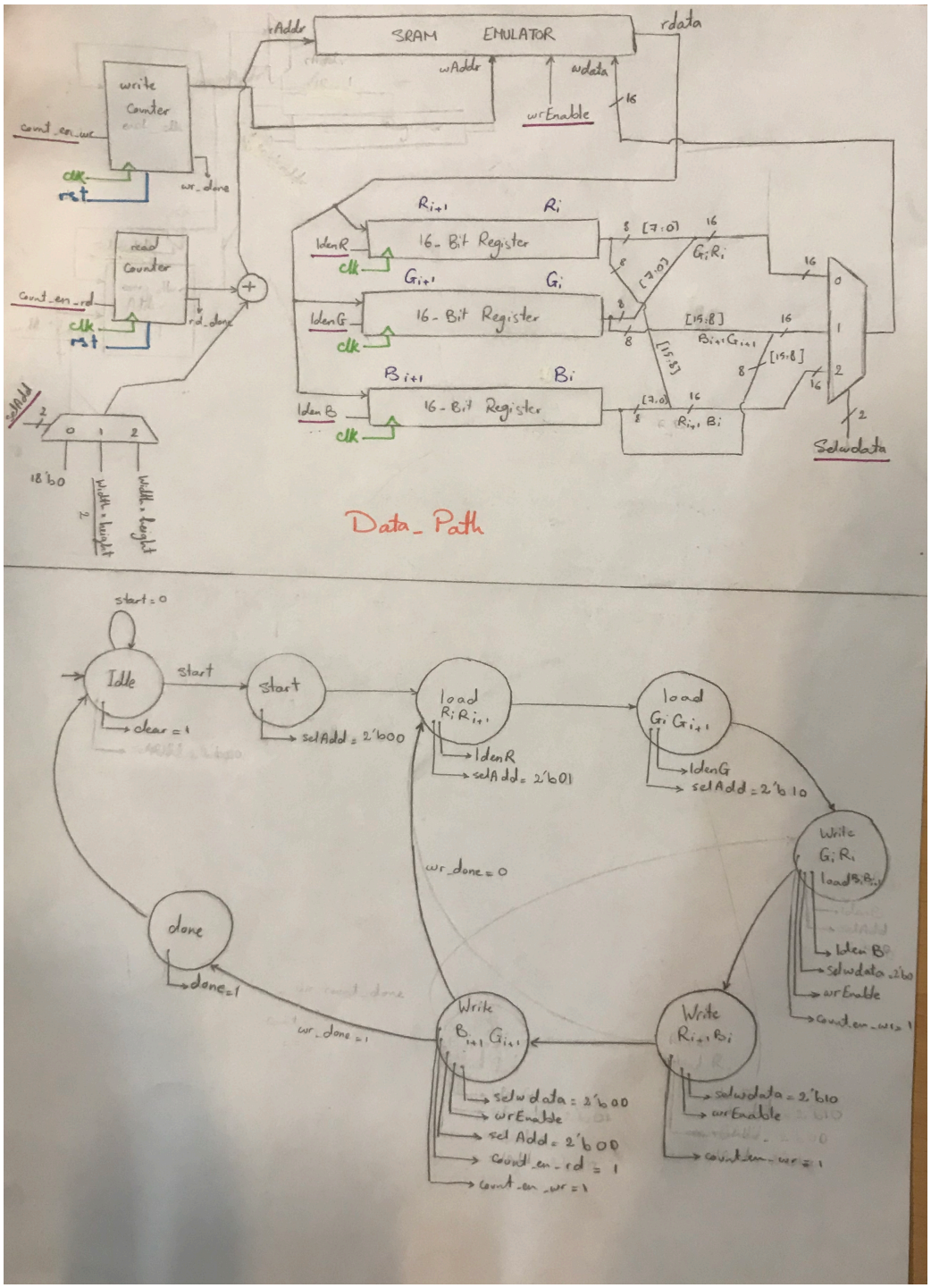
تمرین مقدماتی دوم

گروه ۳ ـ 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.

دو شمارشگر ۱۸ بیتی

وظیفه‌ی یکی از این شمارشگرها ایجاد کردن آدرس نوشتن و یکی دیگر آدرس خواندن را دارد. شمارشگری که آدرس خواندن را ایجاد می‌کند به ازای هر بار بالا رفتن شمارشگر نوشتن یک بار بالا می‌رود.

```
  wire[17:0] start_addr = width*height*3/2;
  wire[17:0] wlimit = width*height*3 ;
  wire[17:0] rlimit = width*height/2 ;
  counter_18bit write_counter(.start_addr(start_addr), .limit(wlimit),.clk(clk),
   .count_en(count_en_wr), .clear(rst), .r_addr(w_addr), .count_done(wr_done));

  wire [17:0] r_addr_base;
  counter_18bit read_counter(.start_addr(18'b0), .limit(rlimit),.clk(clk),
   .count_en(count_en_rd), .clear(rst), .r_addr(r_addr_base), .count_done(rd_done));
```

هرکدام از این شمارشگرها یک آدرس شروع و یک آدرس نهایی دارند که از آدرس شروع، شروع به شماردن می‌کنند و تا آدرس نهایی می‌شمارند.

کد این شمارشگرها به شکل زیر می‌باشند.

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

always@(posedge clk)
begin
  if(clear)
    begin
      r_addr <= start_addr;
      count_done <= 0;
    end

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

سه رجیستر ۱۶ بیتی

هرکدام از این رجیسترها وظیفه‌ی نگهداری Ri,Ri+1 و Gi,Gi+1 و Bi,Bi+1 را دارند.

```
wire [15:0] r_out, g_out, b_out;
register_16_bit r_reg(.load_en(ldenR), .clk(clk), .data_in(r_data), .data_out(r_out));
register_16_bit g_reg(.load_en(ldenG), .clk(clk), .data_in(r_data), .data_out(g_out));
register_16_bit b_reg(.load_en(ldenB), .clk(clk), .data_in(r_data), .data_out(b_out));
```

کد رجیسترها 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
```

دو مولتی‌پلکسر

یک مولتی پلکسر برای انتخاب اطلاعاتی که قرار است نوشته شوند

```
assign wdata = (selWdata == 2'b0) ? {g_out[7:0],r_out[7:0]} : (selWdata == 2'b01) ? {b_out[15:8],
                {r_out[15:8], b_out[7:0]};
```

یک مولتی پلکسر برای انتخاب آدرس خواندن استفاده کرده‌ایم

```
assign r_addr =((selAdd == 2'b0) ? 18'b0 : (selAdd == 2'b01) ? g_offset : b_offset) + r_addr_base
```

در نهایت اینترفیس کل ماژول مسیرداده به شکل زیر می‌باشد.

```
//****************** reorder datapath interface*******************
    input wire clk, rst, ldenB, ldenR, ldenG, count_en_rd, count_en_wr;
    input wire [1:0] selAdd, selWdata;
    input wire [15:0] r_data;
    input wire [15:0] width, height;
    output wire wr_done, rd_done;
    output wire[17:0] w_addr, r_addr;
    output wire [15:0] wdata;
//****************************************************************
```

کنترلر این ماژول از سه بخش تشکیل شده است.

بخش اول که وظیفه‌ی انتقال استیت‌های قبلی به استیت جدید را دارد.

```
always@(posedge clk, posedge rst) begin
    if(rst)
      ps <= 3'd0;
    else if(clk)
      ps <= ns;
end
```

بخش دوم که وظیفه‌ی مشخص کردن استیت بعدی را دارد.

```
always@(ps or start or wr_done)
begin
   case (ps)
    IDLE : ns <= start ? START : IDLE ;
    START : ns <= LOAD_R ;
    LOAD_R : ns <= LOAD_G ;
    LOAD_G : ns <= WR_GR_LD_B ;
    WR_GR_LD_B : ns <= WR_RB ;
    WR_RB : ns <= WR_BG ;
    WR_BG : ns <= wr_done ? DONE : LOAD_R ;
    DONE : ns <= IDLE;
    default: ns <= ps ;
   endcase
end
```

و بخش سوم که وظیفه‌ی مشخص کردن خروجی‌ها را دارد.

```
always@(ps)
begin
    done <= 1'b0;
    clear <= 1'b0;
    ldenB <= 1'b0;
    ldenR <= 1'b0;
    ldenG <= 1'b0;
    count_en_rd <= 1'b0;
    count_en_wr <= 1'b0;
    rd_enable <= 1'b0;
    wr_enable <= 1'b0;
    selAdd <= 2'b00;
    selwdata <= 2'b00;
    case (ps)
      IDLE : begin clear <= 1'b1; end
      START : selAdd <= 2'b00;
      LOAD_R : begin ldenR <= 1'b1; selAdd <= 2'b01; end
      LOAD_G : begin ldenG <= 1'b1; selAdd <= 2'b10; end
      WR_GR_LD_B : begin ldenB <= 1'b1; selwdata <= 2'b00 ; wr_enable <= 1'b1; count_en_wr <= 1'b1; end
      WR_RB : begin selwdata <= 2'b10; wr_enable <= 1'b1; count_en_wr <= 1'b1; end
      WR_BG : begin selwdata <= 2'b01; wr_enable <= 1'b1; selAdd <= 2'b00; count_en_rd <= 1'b1; count_en_wr <=1'b1; end
      DONE: done <= 1'b1;
      default: begin  done <= 1'b0; clear <= 1'b0; ldenB <= 1'b0; ldenR <= 1'b0; ldenG <= 1'b0; count_en_rd <= 1'b0;
                      count_en_wr <= 1'b0; rd_enable <= 1'b0; wr_enable <= 1'b0; selAdd <= 2'b00; selwdata <= 2'b00; end
    endcase
end
```

در نهایت اینترفیس ماژول کنترلر به شکل زیر می‌باشد.

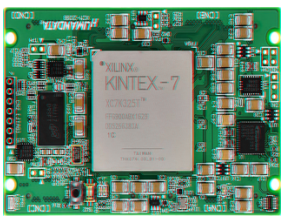
```
//***********reoder controller interface***************//
input wire clk, rst;
input wire wr_done, start;
output reg done , clear, ldenB, ldenR, ldenG, count_en_rd, count_en_wr, rd_enable, wr_enable;
output reg [1:0] selAdd, selwdata;
//*****************************************************//
```

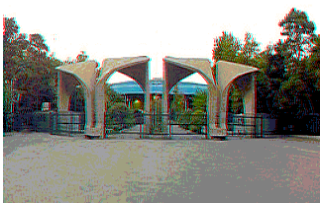
برای شبیه‌سازی فایل‌های وریلاگ را در sim_top.tcl اضافه کردیم.

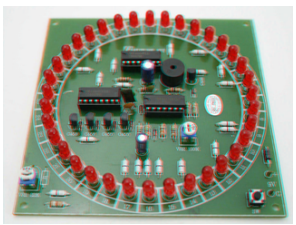
```
vlog    +acc -incr -source  +define+SIM     $hdl_path/*.v
```

در پایان در برنامه‌ی مادل‌سیم 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.

هر کدام از خروجی‌ها به شکل زیر می‌باشد.







سنتز :

برای سنتز کردن، تمامی فایل‌های وریلاگ را به نرم‌افزار ISE اضافه کردیم و مدار را سنتز کردیم.

با کلیک برروی Module Level Utilization به مقادیر زیر دست پیدا می‌کنیم .

Module Name	Partition	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MAP_MULT18X18	BUFG	DCM
decompressor_top		30/214	3/167	55/274	0/0	0/0	0/0	1/1	0/0
decompressor_top		0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0
pixel_reorder		0/102	0/94	0/138	0/0	0/0	0/0	0/0	0/0
c		8/8	9/9	5/5	0/0	0/0	0/0	0/0	0/0
dp		21/94	0/85	41/133	0/0	0/0	0/0	0/0	0/0
b_reg		8/8	16/16	0/0	0/0	0/0	0/0	0/0	0/0
g_reg		8/8	16/16	0/0	0/0	0/0	0/0	0/0	0/0
r_reg		8/8	16/16	0/0	0/0	0/0	0/0	0/0	0/0
read_coun...		24/24	18/18	46/46	0/0	0/0	0/0	0/0	0/0
write_cou...		25/25	19/19	46/46	0/0	0/0	0/0	0/0	0/0
sram_vga_controller		0/82	0/70	0/81	0/0	0/0	0/0	0/0	0/0
c		6/9	0/3	11/11	0/0	0/0	0/0	0/0	0/0
f1		1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0
f2		1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0
f3		1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0
dp		0/73	0/67	0/70	0/0	0/0	0/0	0/0	0/0
cn		25/25	19/19	46/46	0/0	0/0	0/0	0/0	0/0
mux		0/24	0/0	0/24	0/0	0/0	0/0	0/0	0/0
r0		8/8	16/16	0/0	0/0	0/0	0/0	0/0	0/0
r1		8/8	16/16	0/0	0/0	0/0	0/0	0/0	0/0
r2		8/8	16/16	0/0	0/0	0/0	0/0	0/0	0/0

- در decompressor_top سه رجیستر برای ذخیره‌ی state استفاده شده است .
- در pixel_reorder نود و چهار رجیستر استفاده شده است که در بالا هر کدام مشخص شده است.
  - ۱۹ بیت برای شمارشگر wirte. که یک شمارشگر ۱۸ بیتی می‌باشد و یک بیت برای ذخیره سازی پایان شمارش نیاز دارد.
  - ۱۸ بیت بری شمارشگر read که یک شمارشگر ۱۸ بیتی می‌باشد.
  - ۴۸ بیت برای ۳ رجیستر ۱۶ بیتی.
  - ۶ بیت در کنترلر برای ns و ps
  - ۳ بیت در کنترلر برای
- در sram_vga_controller نیز هشتاد و یک رجیستر به کار رفته است که در بالا مشخص شده است.
  - ۴۸ بیت برای ۳ رجیستر ۱۶ بیتی در مسیر داده
  - ۱۹ بیت برای شمارشگر ۱۸ بیتی که یک بیت برای ذخیره‌ی نتیجه‌ی اتمام شمارش دارد.
  - ۳ بیت برای ۳ عدد فلیپ‌فلاپی که در کنترلر قرار دارند.

در نتیجه در نهایت ۱۶۷ تا رجیستر داریم .

بعد از تمام شدن فرآیند سنتز هر کدام از ماژول‌ها مدار را مپ کرده و اعداد گزارش شده به شکل زیر می‌باشند.

AVAILABLE	USED TOTAL	USED PIXEL_REORDER	USED SRAM_VGA_CONTROLLER	LOGIC UTILIZATION	
27,648	214	102	80	NUMBER OF OCCUPIED SLICES	
55,296	274	138	77	TOTAL	NUMBER OF SLICE LUTS
-	205	86	60	NUMBER USED AS LOGIC	
-	0	0	0	NUMBER USED AS MEMORY	
55,296	167	94	70	NUMBER OF SLICE REGISTERS	
0	0	0	0	NUMBER OF MULT18X18S	

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.