تمرین ۶ درس طراحی کامپیوتری سیستم‌های دیجیتال

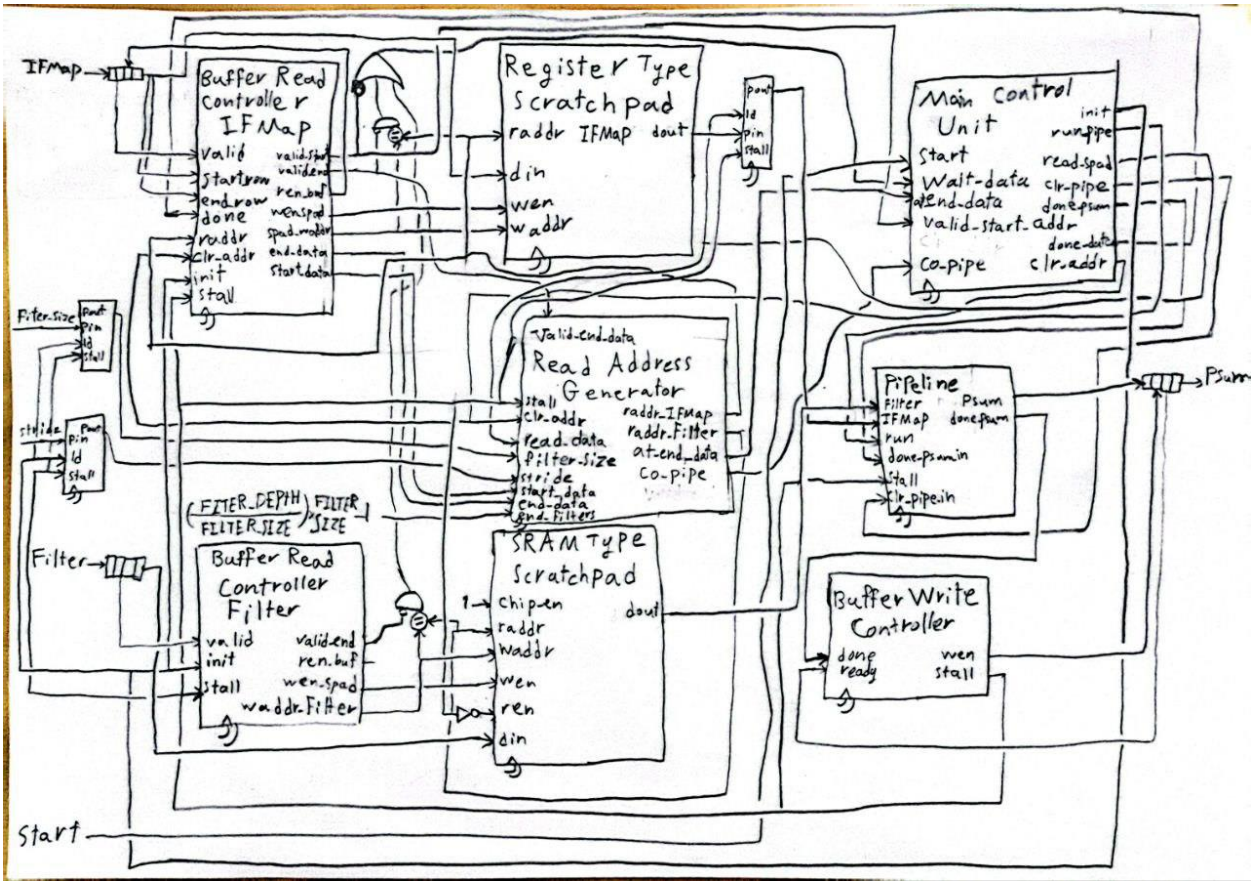
بابک حسینی محتشم ۸۱۰۱۰۱۴۰۸

کیارش خراسانی ۸۱۰۱۰۱۴۱۳

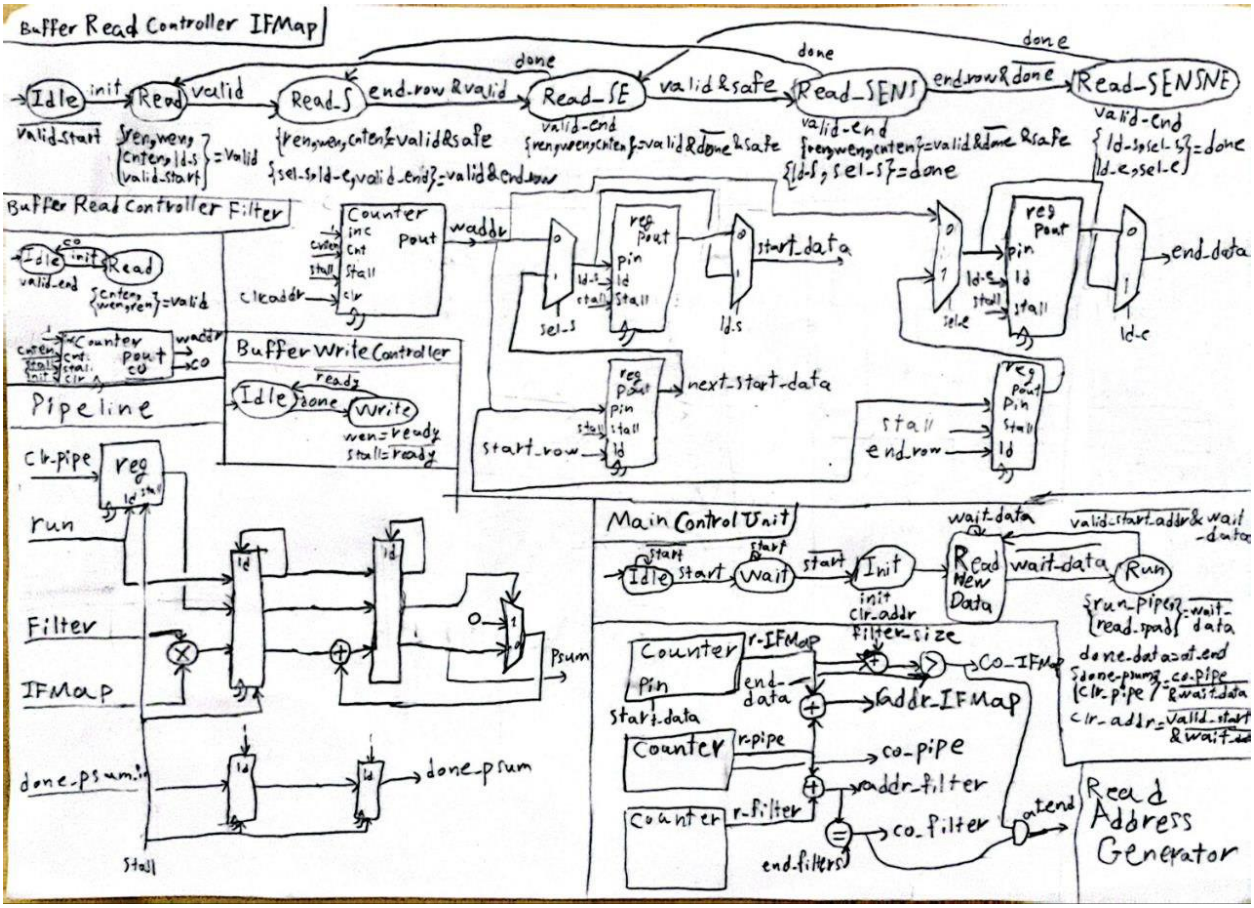
در این تمرین هدف، کامل کردن قسمت محاسباتی ماژول eyeris بود. در این تمرین، ماژول PE_generator را ایجاد کردیم که این ماژول با گرفتن پارامتر N، به همان تعداد از ماژول‌های PE تمرین قبل تولید و به هم متصل می‌کند و در نهایت خروجی آخرین واحد محاسباتی را در بافری به نام global buffer ذخیره می‌کند.

ابتدا طرح کلی را که طراحی کردیم مشاهده میکنیم.

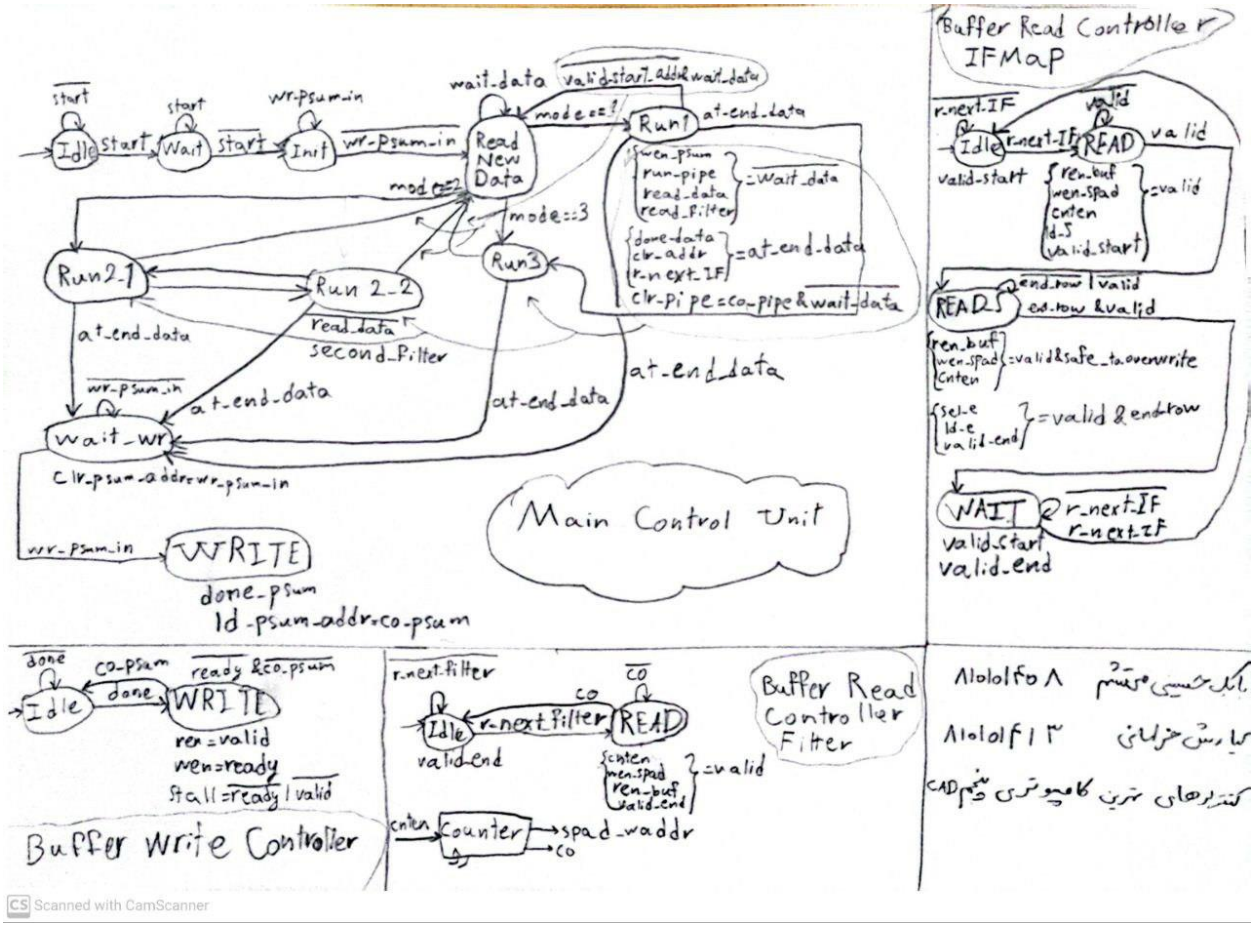
طرح مسیر داده تمرین ۴:



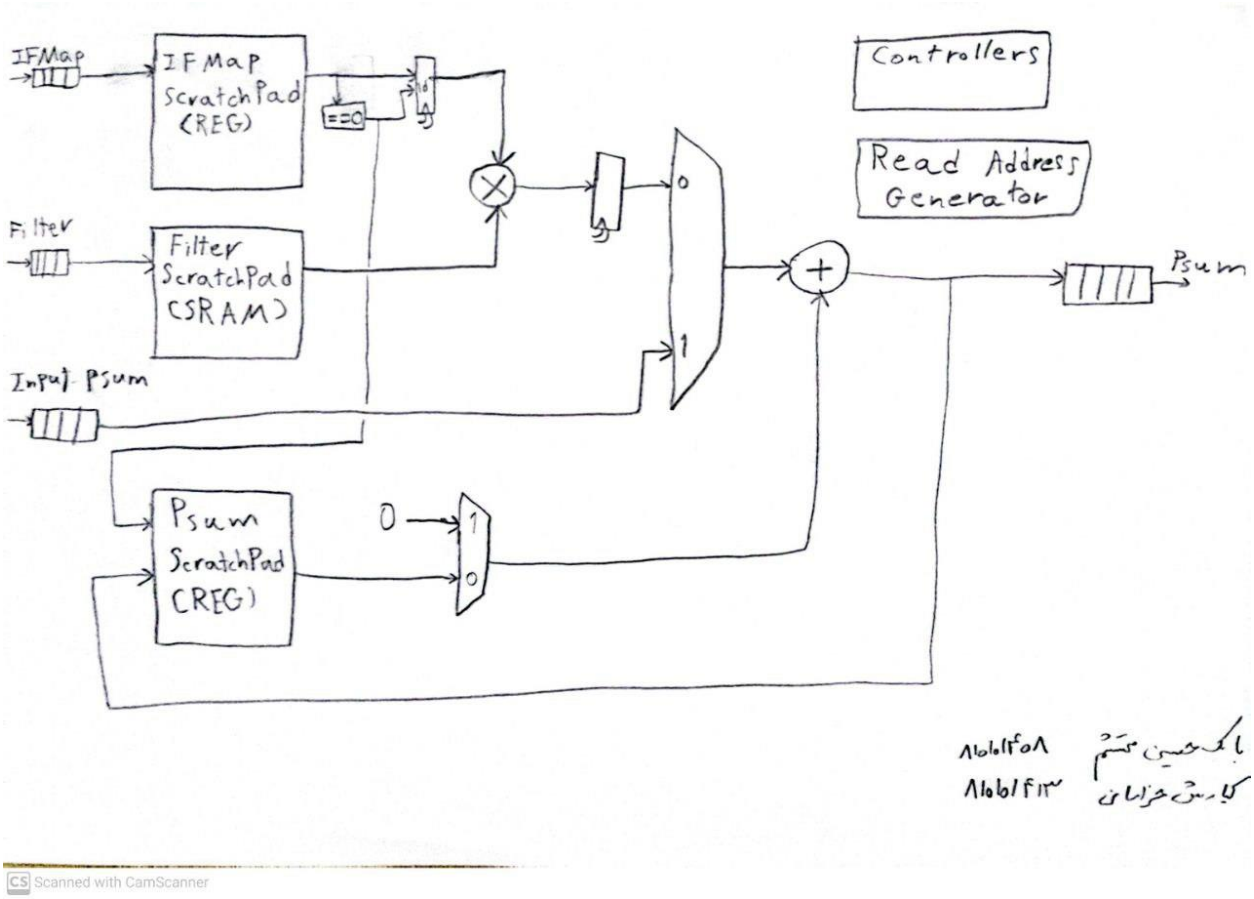
طرح هر یک از ماژول ها در تمرین ۴:



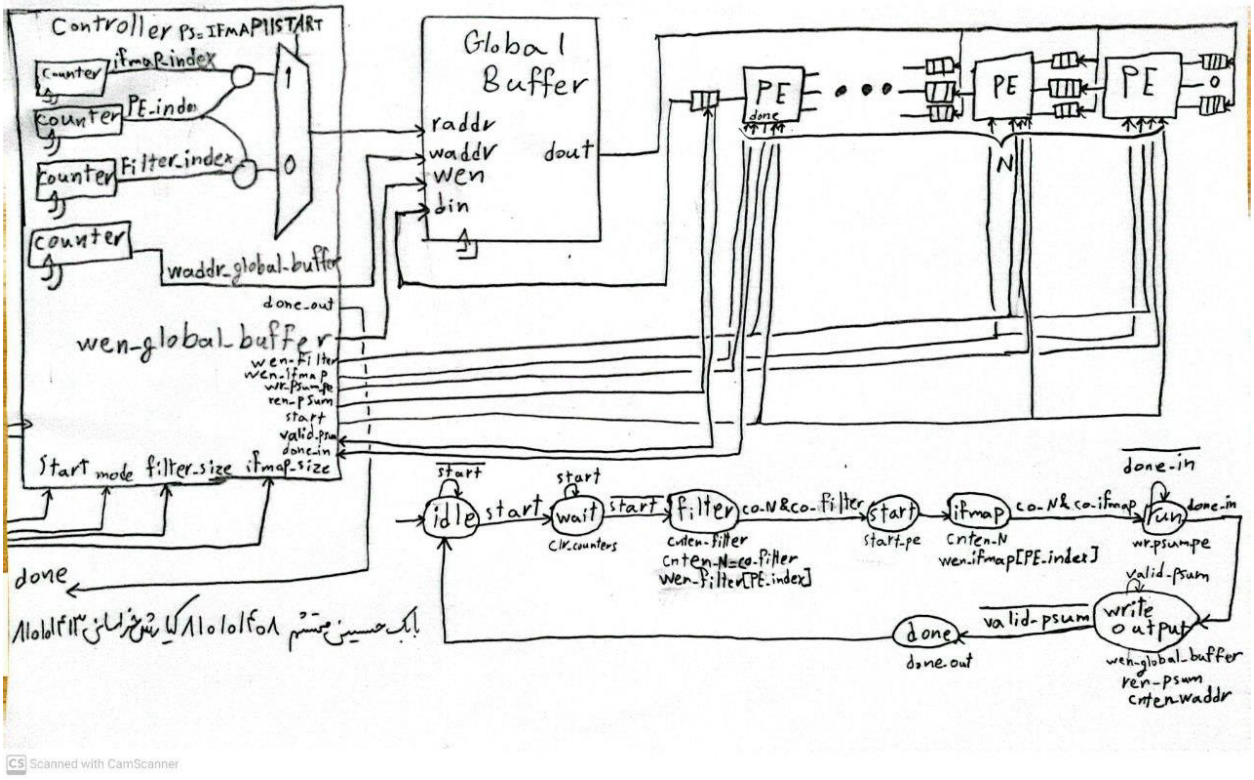
طرح کنترلرهای تمرین ۵:



طرح مسیرداده تمرین ۵:



طرح این تمرین:



در این تمرین برای اینکه بتوانیم ورودی را از فایل بخوانیم و با دادن اندازه فیلتر و داده ورودی بدون داشتن start row و end row، ماژول‌ها به درستی کار کنند، تغییر جزئی در ماژول‌های تمرین قبل ایجاد کردیم. به طوری که کد زیر را به ماژول PE اضافه کردیم که اولین یا آخرین داده را مشخص می‌کند. بدین صورت که اگر اولین داده یا آخرین داده باشد به ترتیب، start row و end row یک می‌شوند. همچنین اگر مود اول باشد، دو سری داده داریم پس در این صورت برای دو داده میانی نیز start row و end row یک می‌شود.

```
assign {start_row,end_row} = (waddr_IFMap==0) ? 2'b10 : (waddr_IFMap==ifmap_size-1) ? 2'b01 :
(mode==1) ? ((waddr_IFMap==ifmap_size/2-1) ? 2'b01 : (waddr_IFMap==ifmap_size/2) ? 2'b10 : 2'b00) : 2'b00;
```

ماژول اولی که برای این پروژه ایجاد کردیم، ماژول global buffer است. این ماژول دقیقا مشابه register type scratchpad کار می‌کند با این تفاوت که با ریست شدن آن، داده و فیلتر را از فایل مشخصی که داده در آن قرار گرفته می‌خواند.

```
module global_buffer#(parameter DATA_WIDTH,ADDR_WIDTH,DEPTH)(raddr,waddr,wen,din,clk,rst, dout);
    input [ADDR_WIDTH-1:0] raddr,waddr;
    input wen,clk,rst;
    input [DATA_WIDTH-1:0] din;
    output [DATA_WIDTH-1:0] dout;
    reg [DATA_WIDTH-1:0] mem [0:DEPTH-1];
    integer file, i, value;

    assign dout = mem[raddr];
    always @(posedge clk) begin
        if (rst) begin
            file = $fopen("mode3-3.txt", "r");
            for (i = 0; i < DEPTH; i = i + 1) begin
                if ($fscanf(file, "%d\n", value) == 1) begin
                    mem[i] = value;
                end else begin
                    mem[i] = 0;
                end
            end
            $fclose(file);
        end
        else begin
            if (wen)
                mem[waddr] <= din;
        end
    end
endmodule
```

ماژول بعدی کنترلر برای تولید کننده واحد محاسباتی است.

```
module controller_PE_generator#(parameter N_WIDTH,N,GLOBAL_BUFFER_ADDR_WIDTH,GLOBAL_BUFFER_DEPTH,IFMAP_SIZE_WIDTH,FILTER_SIZE_WIDTH)
(Start,mode,filter_size,ifmap_size,done_in,valid_psum,clk,rst,raddr_global_buffer,waddr_global_buffer,Start_pe,wen_filter,
wen_global_buffer,ren_psum,wen_ifmap,done_out,wr_psum_pe);
    input Start,done_in,valid_psum,clk,rst;
    input [FILTER_SIZE_WIDTH-1:0] filter_size;
    input [IFMAP_SIZE_WIDTH-1:0] ifmap_size;
    input [1:0] mode;
    output reg Start_pe,wen_global_buffer,ren_psum,done_out,wr_psum_pe;
    output reg [N-1:0] wen_ifmap=0,wen_filter=0;
    output [GLOBAL_BUFFER_ADDR_WIDTH-1:0] raddr_global_buffer,waddr_global_buffer;

    wire [IFMAP_SIZE_WIDTH-1:0] ifmap_index;
    wire [FILTER_SIZE_WIDTH:0] filter_index;
    wire [GLOBAL_BUFFER_ADDR_WIDTH-1:0] filter_addr,ifmap_addr;
    wire [N_WIDTH-1:0] PE_index;
    wire co_N,co_filter,co_ifmap;
    wire [FILTER_SIZE_WIDTH:0] mode_filter_size;
    reg cnten_waddr=1'b0,clr_counters=1'b0,cnten_ifmap=1'b0,cnten_N=1'b0,cnten_filter=1'b0;

    assign mode_filter_size = (mode==2) ? {filter_size[FILTER_SIZE_WIDTH-1:0],1'b0} : {1'b0,filter_size};
    assign filter_addr = filter_index + (PE_index * mode_filter_size);
    assign ifmap_addr = ifmap_index + (PE_index * ifmap_size) + mode_filter_size * N;

    parameter [2:0] IDLE=0,WAIT=1,FILTER=2,START=3,IFMAP=4,RUN=5,WRITE_OUTPUT=6,DONE=7;
    reg [2:0] ns=IDLE,ps=IDLE;
    always@(posedge clk) begin
        if(rst)
        ps<=IDLE;
        else
            ps<=ns;
    end
    assign raddr_global_buffer = (ps==IFMAP || ps==START) ? ifmap_addr : filter_addr;
```

در این ماژول، تعدادی wire وجود دارد:

- Mode_filter_size: همان اندازه فیلتر است مگر این که مود دوم باشیم که در آن صورت دو برابر اندازه فیلتر می‌شود زیرا در این مود، در اصل دو فیلتر وجود دارد.
- Filter_index: نشان دهنده اندیسی از فیلتر فعلی است که داریم از بافر می‌خوانیم.
- IFMAP_index: نشان دهنده اندیسی از داده فعلی است که داریم از بافر می‌خوانیم.
- PE_index: نشان دهنده اندیس واحد محاسباتی فعلی است که قصد داریم در بافرهای آن بنویسیم.
- Filter_addr: نشان دهنده آدرس خانه‌ای از فیلتر است که از بافر میخوانیم.
- IFMAP_addr: نشان دهنده آدرس خانه‌ای از داده است که از بافر میخوانیم.
- Raddr_global_buffer: در صورتی که در استیت خواندن فیلتر باشیم برابر Filter_addr وگرنه برابر IFMAP_addr است.

```
counter#(.WIDTH(FILTER_SIZE_WIDTH+1),.WIDTH_INC(1))counter_filter(.max_count(mode_filter_size),.inc(1'b1),
.stall(1'b0),.cnt(cnten_filter),.clr(clr_counters),.clk(clk),.rst(rst), .pout(filter_index),.co(co_filter));
counter#(.WIDTH(IFMAP_SIZE_WIDTH),.WIDTH_INC(1))counter_ifmap(.max_count(ifmap_size),.inc(1'b1),
.stall(1'b0),.cnt(cnten_ifmap),.clr(clr_counters),.clk(clk),.rst(rst), .pout(ifmap_index),.co(co_ifmap));
counter#(.WIDTH(N_WIDTH),.WIDTH_INC(1))counter_PE(.max_count(N[N_WIDTH-1:0]),.inc(1'b1),.stall(1'b0),.cnt(cnten_N),
    .clr(clr_counters),.clk(clk),.rst(rst), .pout(PE_index),.co(co_N));
wire [GLOBAL_BUFFER_ADDR_WIDTH-1:0] waddr;
assign waddr = ((GLOBAL_BUFFER_DEPTH + 1) / 2);
counter_with_load#(.WIDTH(GLOBAL_BUFFER_ADDR_WIDTH),.WIDTH_INC(1))counter_waddr(.pin(waddr),.ld(clr_counters),.cnt(cnten_waddr),.clr(1'b0),.inc(1'b1),
    .max_count(GLOBAL_BUFFER_DEPTH[GLOBAL_BUFFER_ADDR_WIDTH-1:0]),.stall(1'b0),.clk(clk),.rst(rst), .pout(waddr_global_buffer));
```

همچنین در این ماژول تعدادی شمارنده وجود دارد:

- شمارنده فیتر، اندیس فیلتر را می‌شمارد.
- شمارنده داده، اندیس داده را می‌شمارد.
- شمارنده واحد محاسباتی، اندیس واحد محاسباتی را می‌شمارد.
- شمارنده آدرس نوشتن، آدرس نوشتن در حافظه global buffer می‌شمارد. این شمارنده از نیمه‌ی دوم بافر شروع به نوشتن می‌کند و حداکثر تا انتهای بافر را می‌تواند بشمارد.

```
always@(*) begin
    ns=IDLE;
    case (ps)
        IDLE: begin ns = Start ? WAIT : IDLE; end
        WAIT: begin ns = Start ? WAIT : FILTER; end
        FILTER: begin ns = (co_N & co_filter) ? START : FILTER; end
        START: begin ns = IFMAP; end
        IFMAP: begin ns = (co_N & co_ifmap) ? RUN : IFMAP; end
        RUN: begin ns = done_in ? WRITE_OUTPUT : RUN; end
        WRITE_OUTPUT: begin ns = valid_psum ? WRITE_OUTPUT : DONE; end
        DONE: begin ns = IDLE; end
        default: ns = IDLE;
    endcase
end
always@(*) begin
    {cnten_filter,cnten_N,Start_pe,ren_psum,cnten_ifmap,done_out,wr_psum_pe,wen_global_buffer,cnten_waddr,clr_counters} = 10'd0;
    case (ps)
        IDLE:;
        WAIT: {clr_counters} = {1'b1};
        FILTER: begin
            {cnten_filter,cnten_N} = {1'b1,co_filter};
            wen_filter={N{1'b0}};
            wen_filter[PE_index]=1'b1;
        end
        START: {Start_pe,wen_filter} = {1'b1,{N{1'b0}}};
        IFMAP: begin
            {cnten_N,cnten_ifmap} = {1'b1,co_N};
            wen_ifmap = {N{1'b0}};
            wen_ifmap[PE_index] = 1'b1;
        end
        RUN: {wr_psum_pe,wen_ifmap} = {1'b1,{N{1'b0}}};
        WRITE_OUTPUT: {wen_global_buffer,ren_psum,cnten_waddr} = {3'b111};
        DONE: {done_out} = {1'b1};
        default: {cnten_filter,cnten_N,Start_pe,ren_psum,cnten_ifmap,done_out,wr_psum_pe,wen_global_buffer,cnten_waddr,clr_counters} = 10'd0;
    endcase
end
```

استیت‌های این کنترلر بدین صورت هستند:

- IDLE: در این استیت کار خاصی انجام نمی‌شود و منتظر سیگنال Start می‌مانیم و سپس به استیت WAIT می‌رویم.
- WAIT: در این استیت تنها شمارنده‌ها را ریست می‌کنیم و منتظر غیرفعال شدن Start می‌مانیم و سپس به استیت Filter می‌رویم.
- Filter: در این استیت، به ازای هر یک از واحدهای محاسباتی، تمام فیلترهای آن را از global buffer می‌خوانیم و در بافر مربوطه می‌نویسیم و سپس به استیت START می‌رویم.
- START: در این استیت سیگنال شروع به کار را برای تمام واحدهای محاسباتی فعال می‌کنیم. البته آن‌ها تا آمدن اولین داده‌شان منتظر می‌مانند.
- IFMAP: در این استیت، به ازای هر اندیس داده، داده مربوط به هر واحد محاسباتی را در بافر مربوط به آن می‌نویسیم. بدین صورت چون پس از N کلاک، تمام واحدهای محاسباتی یک داده دارند، تقریبا هم زمان شروع به کار می‌کنند.
- RUN: در این استیت سیگنال تغییر مود به psum را می‌دهیم و هر یک از واحدهای محاسباتی پس از اتمام محاسباتش به این مود می‌رود و تا جایی که داده از واحد محاسباتی قبلی داشته باشد، خروجی‌های مورد نظر را تولید می‌کند. پس از اینکه از آخرین واحد محاسباتی سیگنال done را دریافت کردیم یعنی کار تمام واحدهای محاسباتی تمام شده و می‌توانیم داده را از بافر آخر بخوانیم در بافر خروجی بنویسیم.
- WRITE_OUTPUT: در این استیت هر یک از داده‌های موجود در بافر واحد محاسباتی آخر را خوانده و در بافر خروجی می‌نویسیم و سپس به استیت DONE می‌رویم.
- DONE: در این استیت، به طول یک کلاک سیگنال Done را به نشانه اتمام تمام محاسبات فعال کرده و سپس به استیت ابتدایی یعنی IDLE برمی‌گردیم.

آخرین ماژولی که در این تمرین نیاز بود، ماژول PE_generator است.

```
module PE_generator#(parameter N_WIDTH,N,GLOBAL_BUFFER_ADDR_WIDTH,GLOBAL_BUFFER_DEPTH,STRIDE_WIDTH,FILTER_SIZE_WIDTH,IFMAP_SIZE_WIDTH,IFMAP_ADDR_WIDTH,IFMAP_DEPTH,PSUM_ADDR_WIDTH
,PSUM_DEPTH,FILTER_ADDR_WIDTH,FILTER_DEPTH,DATA_WIDTH,PSUM_BUF_DEPTH,FILTER_BUF_DEPTH,IFMAP_BUF_DEPTH)(clk,rst,Start,mode,Done,stride,filter_size,ifmap_size);
    input clk,rst,Start;
    input [1:0] mode;
    input [STRIDE_WIDTH-1:0] stride;
    input [FILTER_SIZE_WIDTH-1:0] filter_size;
    input [IFMAP_SIZE_WIDTH-1:0] ifmap_size;
    output Done;

    wire Start_pe,wr_psum_pe;
    wire [N-1:0] ren_Psum,wen_IFMap,wen_Filter,ren_IFMap,valid_IFMap,ready_psum,ren_Filter,valid_Filter,valid_buffer_Psum,wen_Psum,ready,done;
    wire [DATA_WIDTH-1:0] global_buffer_out;
    wire [DATA_WIDTH-1:0] IFMap_out [0:N-1];
    wire [DATA_WIDTH-1:0] Filter_out [0:N-1];
    wire [DATA_WIDTH-1:0] PE_Psum [0:N-1];
    wire [DATA_WIDTH-1:0] buffer_Psum [0:N-1];
    wire [N_WIDTH-1:0] PE_index;

    wire wen_global_buffer;
    wire [GLOBAL_BUFFER_ADDR_WIDTH-1:0] raddr_global_buffer,waddr_global_buffer;

    global_buffer#(.DATA_WIDTH(DATA_WIDTH),.ADDR_WIDTH(GLOBAL_BUFFER_ADDR_WIDTH),.DEPTH(GLOBAL_BUFFER_DEPTH))gb(
    .raddr(raddr_global_buffer),.waddr(waddr_global_buffer),.wen(wen_global_buffer),.din(buffer_Psum[N-1]),.clk(clk),.rst(rst), .dout(global_buffer_out));

    controller_PE_generator#(.N_WIDTH(N_WIDTH),.N(N),.GLOBAL_BUFFER_ADDR_WIDTH(GLOBAL_BUFFER_ADDR_WIDTH),.GLOBAL_BUFFER_DEPTH(GLOBAL_BUFFER_DEPTH),
    .IFMAP_SIZE_WIDTH(IFMAP_SIZE_WIDTH),.FILTER_SIZE_WIDTH(FILTER_SIZE_WIDTH))ctrl_pe_gen(.Start(Start),.done_in(done[N-1]),.filter_size(filter_size),.mode(mode),
    .ifmap_size(ifmap_size),.valid_psum(valid_buffer_Psum[N-1]),.clk(clk),.rst(rst),.PE_index(PE_index),.raddr_global_buffer(raddr_global_buffer),
    .waddr_global_buffer(waddr_global_buffer),.Start_pe(Start_pe),.wen_filter(wen_Filter),.wen_global_buffer(wen_global_buffer),.ren_psum(ren_Psum[N-1]),
    .wen_ifmap(wen_IFMap),.done_out(Done),.wr_psum_pe(wr_psum_pe));
```

در این ماژول کنترلر و global buffer را ایجاد می‌کنیم.

```
circular_buffer#(.PAR_WRITE(1),.PAR_READ(1),.DEPTH(IFMAP_BUF_DEPTH),.BITS(DATA_WIDTH))fifo_IFMap0
(.clk(clk),.rst(rst),.read_en(ren_IFMap[0]),.write_en(wen_IFMap[0]),.din(global_buffer_out), .valid(valid_IFMap[0]),.ready(),.dout(IFMap_out[0]));

circular_buffer#(.PAR_WRITE(1),.PAR_READ(1),.DEPTH(FILTER_BUF_DEPTH),.BITS(DATA_WIDTH))fifo_Filter0
(.clk(clk),.rst(rst),.read_en(ren_Filter[0]),.write_en(wen_Filter[0]),.din(global_buffer_out[DATA_WIDTH-1:0]), .valid(valid_Filter[0]),.ready(),.dout(Filter_out[0]));

Processing_element#(.IFMAP_SIZE_WIDTH(IFMAP_SIZE_WIDTH),.STRIDE_WIDTH(STRIDE_WIDTH),.FILTER_SIZE_WIDTH(FILTER_SIZE_WIDTH),.IFMAP_ADDR_WIDTH(IFMAP_ADDR_WIDTH),.IFMAP_DEPTH(IFMAP_DEPTH),
.FILTER_ADDR_WIDTH(FILTER_ADDR_WIDTH),.FILTER_DEPTH(FILTER_DEPTH),.DATA_WIDTH(DATA_WIDTH),.PSUM_ADDR_WIDTH(PSUM_ADDR_WIDTH),.PSUM_DEPTH(PSUM_DEPTH))pe0
(.clk(clk),.rst(rst),.Start(Start_pe),.IFMap(IFMap_out[0]),.ready(ready[0]),.done(done[0]),.valid_IFMap(valid_IFMap[0]),.Filter(Filter_out[0]),.valid_Filter(valid_Filter[0]),
.stride_in(stride),.mode_in(mode),.input_Psum({{DATA_WIDTH}{1'b0}}),.valid_input_Psum(1'b1),.wr_psum_in(wr_psum_pe),.filter_size_in(filter_size),.ifmap_size_in(ifmap_size),
.ready_Psum(ready_psum[0]),.Psum(PE_Psum[0]),.ren_buf_Filter(ren_Filter[0]),.ren_buf_IFMap(ren_IFMap[0]),.ren_buf_input_Psum(),.wen_buf_Psum(wen_Psum[0]));

circular_buffer#(.PAR_WRITE(1),.PAR_READ(1),.DEPTH(PSUM_BUF_DEPTH),.BITS(DATA_WIDTH))fifo_Psum0
(.clk(clk),.rst(rst),.read_en(ren_Psum[0]),.write_en(wen_Psum[0]),.din(PE_Psum[0]), .valid(valid_buffer_Psum[0]),.ready(ready_psum[0]),.dout(buffer_Psum[0]));

genvar i;
generate
    for (i=1; i<N; i=i+1) begin
        circular_buffer#(.PAR_WRITE(1),.PAR_READ(1),.DEPTH(IFMAP_BUF_DEPTH),.BITS(DATA_WIDTH))fifo_IFMap
        (.clk(clk),.rst(rst),.read_en(ren_IFMap[i]),.write_en(wen_IFMap[i]),.din(global_buffer_out), .valid(valid_IFMap[i]),.ready(),.dout(IFMap_out[i]));

        circular_buffer#(.PAR_WRITE(1),.PAR_READ(1),.DEPTH(FILTER_BUF_DEPTH),.BITS(DATA_WIDTH))fifo_Filter
        (.clk(clk),.rst(rst),.read_en(ren_Filter[i]),.write_en(wen_Filter[i]),.din(global_buffer_out[DATA_WIDTH-1:0]), .valid(valid_Filter[i]),.ready(),.dout(Filter_out[i]));

        Processing_element#(.IFMAP_SIZE_WIDTH(IFMAP_SIZE_WIDTH),.STRIDE_WIDTH(STRIDE_WIDTH),.FILTER_SIZE_WIDTH(FILTER_SIZE_WIDTH),.IFMAP_ADDR_WIDTH(IFMAP_ADDR_WIDTH),.IFMAP_DEPTH(IFMAP_DEPTH),
        .FILTER_ADDR_WIDTH(FILTER_ADDR_WIDTH),.FILTER_DEPTH(FILTER_DEPTH),.DATA_WIDTH(DATA_WIDTH),.PSUM_ADDR_WIDTH(PSUM_ADDR_WIDTH),.PSUM_DEPTH(PSUM_DEPTH))pe
        (.clk(clk),.rst(rst),.Start(Start_pe),.IFMap(IFMap_out[i]),.ready(ready[i]),.done(done[i]),.valid_IFMap(valid_IFMap[i]),.Filter(Filter_out[i]),.valid_Filter(valid_Filter[i]),
        .stride_in(stride),.mode_in(mode),.input_Psum(buffer_Psum[i-1]),.valid_input_Psum(valid_buffer_Psum[i-1]),.wr_psum_in(wr_psum_pe),.filter_size_in(filter_size),.ifmap_size_in(ifmap_size),
        .ready_Psum(ready_psum[i]),.Psum(PE_Psum[i]),.ren_buf_Filter(ren_Filter[i]),.ren_buf_IFMap(ren_IFMap[i]),.ren_buf_input_Psum(ren_Psum[i-1]),.wen_buf_Psum(wen_Psum[i]));

        circular_buffer#(.PAR_WRITE(1),.PAR_READ(1),.DEPTH(PSUM_BUF_DEPTH),.BITS(DATA_WIDTH))fifo_Psum
        (.clk(clk),.rst(rst),.read_en(ren_Psum[i]),.write_en(wen_Psum[i]),.din(PE_Psum[i]), .valid(valid_buffer_Psum[i]),.ready(ready_psum[i]),.dout(buffer_Psum[i]));
    end
endgenerate
```

همچنین N واحد محاسباتی را نیز ایجاد کرده و اتصالات مورد نیاز را انجام می‌دهیم.

در نهایت ماژول را با ترکیب دو تست کیس 3-2 و 3-3 تمرین قبل، آزمایش می‌کنیم. انتظار داریم که پاسخی

که به دست می‌آید برابر حاصل جمع پاسخ این دو تست کیس شود که همین طور هم می‌شود.

```
//filter&ifmap channels: 2, ifmap size: 5, filter size: 3, filter lines: 1 - 6, ifmap lines: 7 - 16, output: 17 - 21
-42
151
88
-44
-68
41
19
-16
17
-65
34
-32
13
-34
21
-5
-2482
-1268
-8407
7649
-5253
```

[0]	-42
[1]	151
[2]	88
[3]	-44
[4]	-68
[5]	41
[6]	19
[7]	-16
[8]	17
[9]	-65
[10]	34
[11]	-32
[12]	13
[13]	-34
[14]	21
[15]	-5
[16]	-130
[17]	-53
[18]	177
[19]	-120
[20]	-121
[21]	25
[22]	0
[23]	0
[24]	-1
[25]	2
[26]	-1
[27]	-2
[28]	2
[29]	0
[30]	1
[31]	1

[500]	-2828
[501]	-449
[502]	-8562
[503]	6873
[504]	-4759

```
module Processing_element_tb3_1();
    parameter N=3,N_WIDTH=2,IFMAP_SIZE_WIDTH=4,GLOBAL_BUFFER_ADDR_WIDTH=10,GLOBAL_BUFFER_DEPTH=1000,STRIDE_WIDTH=2,FILTER_SIZE_WIDTH=4,IFMAP_ADDR_WIDTH=5,IFMAP_DEPTH=15,
    FILTER_ADDR_WIDTH=4,FILTER_DEPTH=15,DATA_WIDTH=20,IFMAP_BUF_DEPTH=60,PSUM_BUF_DEPTH=60,FILTER_BUF_DEPTH=60,PSUM_ADDR_WIDTH=5,PSUM_DEPTH=20;
    reg clk=0,rst=1,Start=0;
    reg [1:0] mode=3;
    reg [STRIDE_WIDTH-1:0] stride=1;
    reg [FILTER_SIZE_WIDTH-1:0] filter_size=6;
    reg [IFMAP_SIZE_WIDTH-1:0] ifmap_size=10;
    wire Done;

    PE_generator#(.N(N),.N_WIDTH(N_WIDTH),.GLOBAL_BUFFER_ADDR_WIDTH(GLOBAL_BUFFER_ADDR_WIDTH),.GLOBAL_BUFFER_DEPTH(GLOBAL_BUFFER_DEPTH),.STRIDE_WIDTH(STRIDE_WIDTH),.IFMAP_SIZE_WIDTH(IFMAP_SIZE_WIDTH),
    .FILTER_SIZE_WIDTH(FILTER_SIZE_WIDTH),.IFMAP_ADDR_WIDTH(IFMAP_ADDR_WIDTH),.IFMAP_DEPTH(IFMAP_DEPTH),.PSUM_ADDR_WIDTH(PSUM_ADDR_WIDTH),.PSUM_DEPTH(PSUM_DEPTH),
    .FILTER_ADDR_WIDTH(FILTER_ADDR_WIDTH),.FILTER_DEPTH(FILTER_DEPTH),.DATA_WIDTH(DATA_WIDTH),.PSUM_BUF_DEPTH(PSUM_BUF_DEPTH),.FILTER_BUF_DEPTH(FILTER_BUF_DEPTH),
    .IFMAP_BUF_DEPTH(IFMAP_BUF_DEPTH))PE_gen(.clk(clk),.rst(rst),.Start(Start),.mode(mode),.Done(Done),.stride(stride),.filter_size(filter_size),.ifmap_size(ifmap_size));

    always begin #19;clk=~clk;end
    initial begin
        #38;
        #38; rst=1'b0;
        #38; Start = 1'b1;
        #38; Start = 1'b0;
        #38000;
        $stop;
    end
endmodule
```