تمرین ۴ درس طراحی کامپیوتری سیستم های دیجیتال

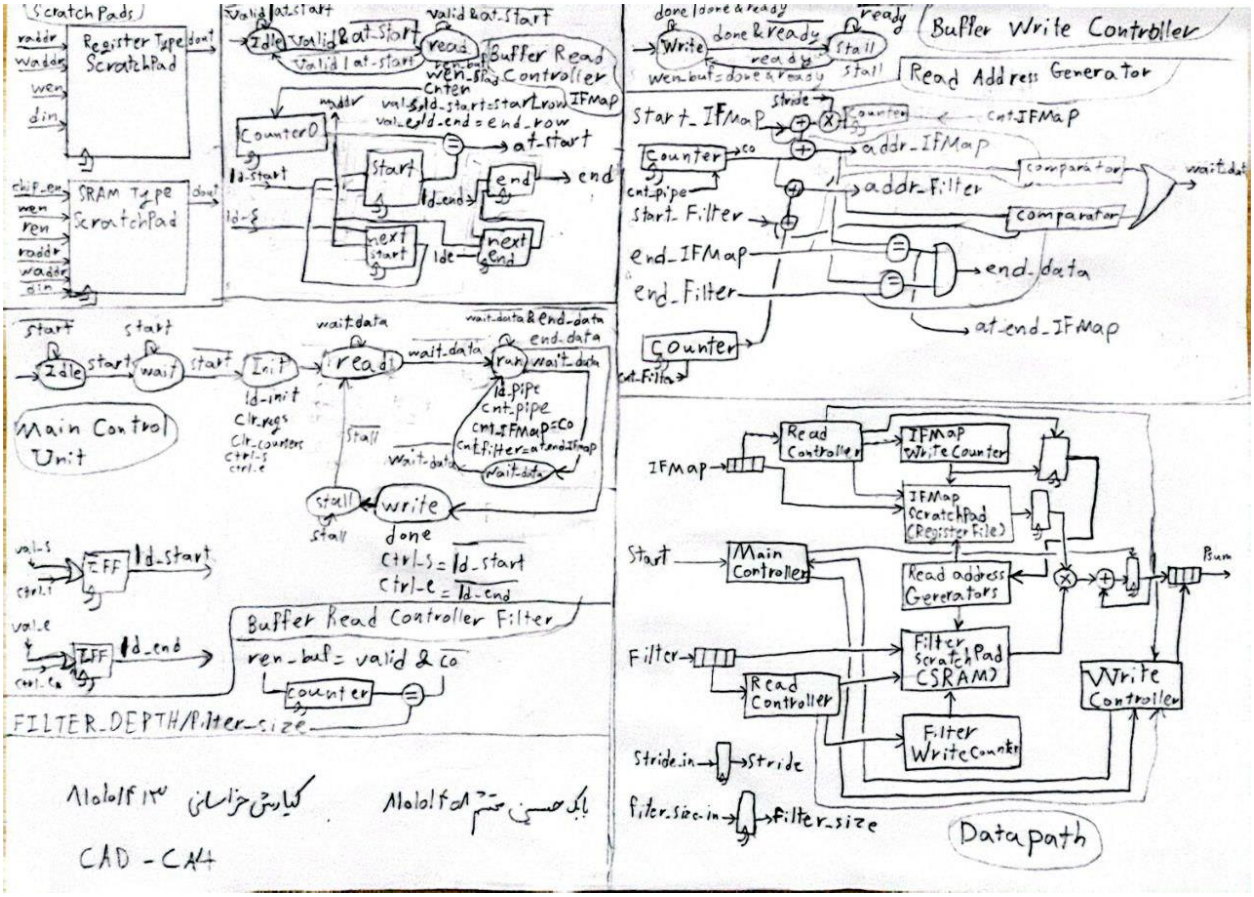
بابک حسینی محتشم ۸۱۰۱۰۱۴۰۸

کیارش خراسانی ۸۱۰۱۰۱۴۱۳

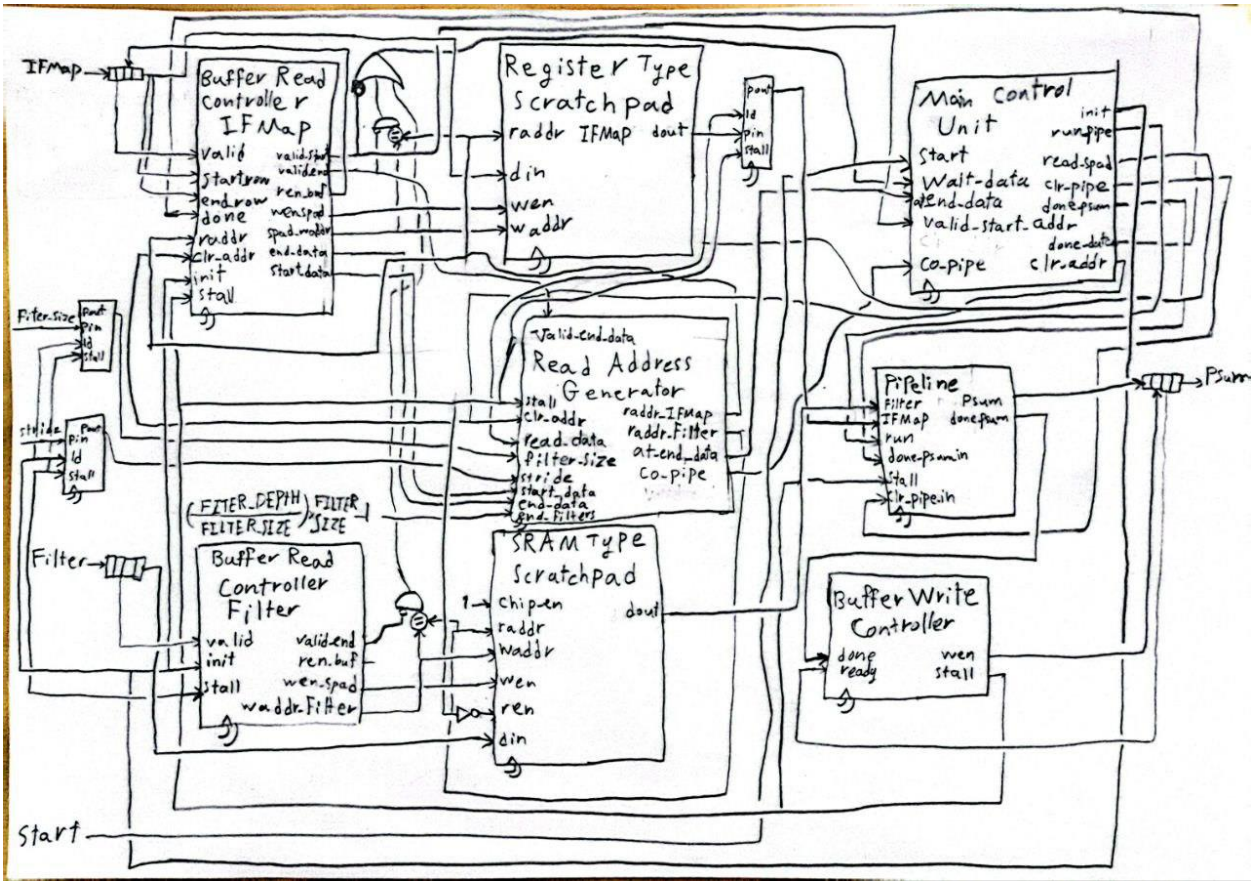
در این تمرین هدف، پیاده سازی قسمت محاسباتی ماژول eyeris بود. خود قسمت محاسباتی از ۶ بخشی اصلی تشکیل شده است که در ادامه هر یک را به طور جداگانه توضیح خواهیم داد.

ابتدا طرح کلی را که طراحی کردیم مشاهده میکنیم.

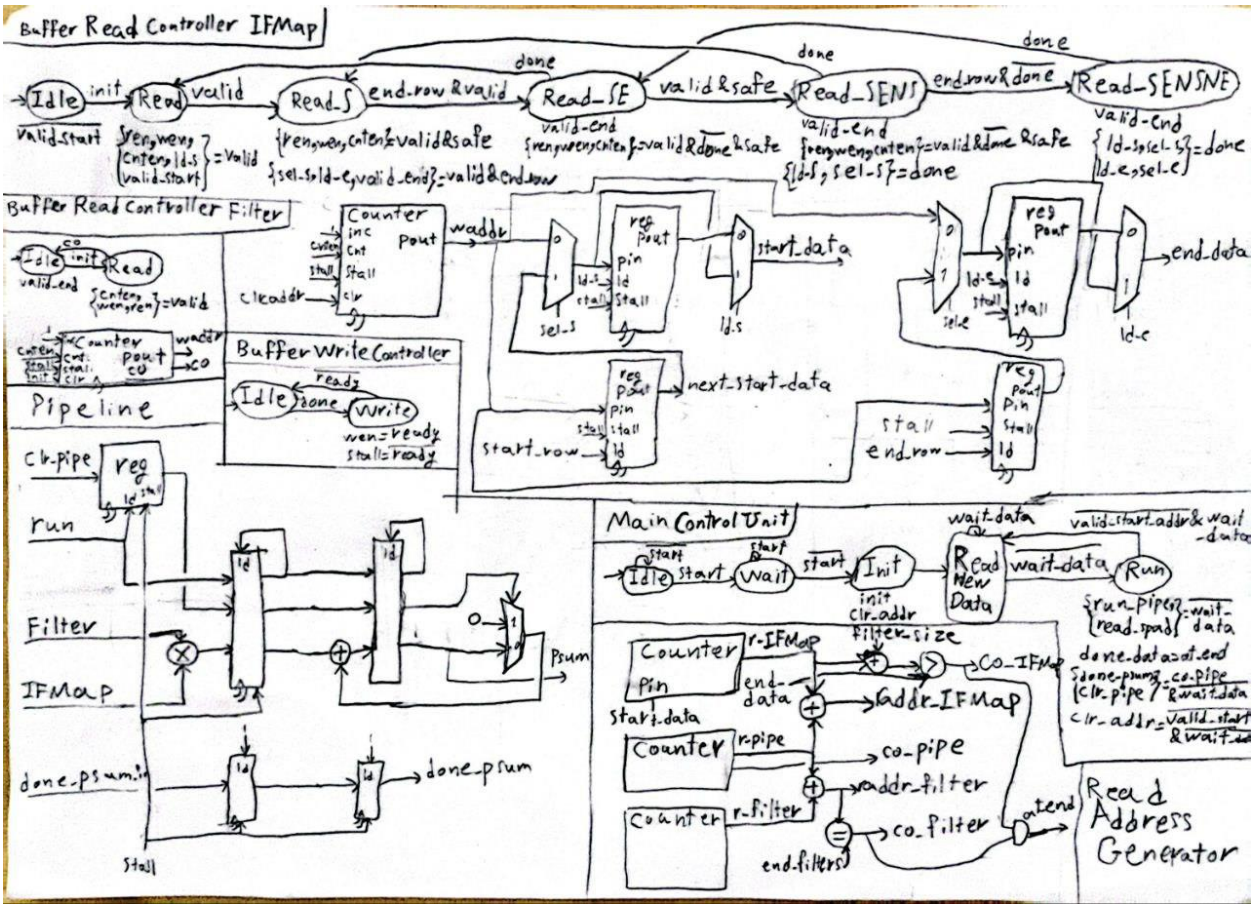
طرح اولیه:



طراح نهایی مسیر داده کلی:



طرح نهایی هر یک از ماژول ها:

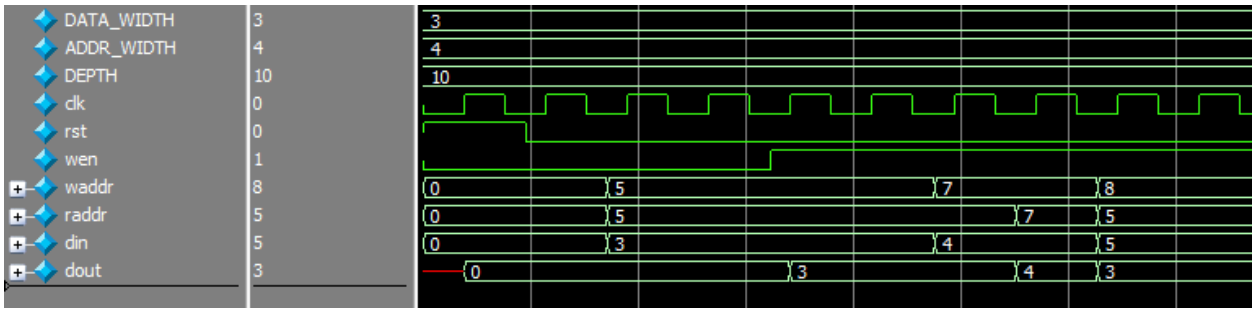


۱. در این تمرین از دو نوع حافظه مختلف برای ذخیره سازی داده و فیلتر استفاده می‌کنیم.

نوع اول Register Type ScratchPad است که از آن برای ذخیره کردن داده‌های ورودی IFMap استفاده می‌کنیم. کد این ماژول مانند یک حافظه مموری ساده است.

```
module register_type_scratchpad#(parameter DATA_WIDTH,ADDR_WIDTH,DEPTH)(raddr,waddr,wen,din,clk,rst, dout);
input wen,clk,rst;
input [ADDR_WIDTH-1:0] raddr,waddr;
    input [DATA_WIDTH-1:0] din;
    output [DATA_WIDTH-1:0] dout;
    reg [DATA_WIDTH-1:0] mem [0:DEPTH-1];
    assign dout = mem[raddr];
    integer i=0;
    always @(posedge clk) begin
        if (rst) begin
            for (i = 0; i < DEPTH; i = i + 1) begin
                mem[i] <= {DATA_WIDTH{1'b0}};
            end
        end
        else if (wen) begin
            mem[waddr] <= din;
        end
    end
endmodule
```

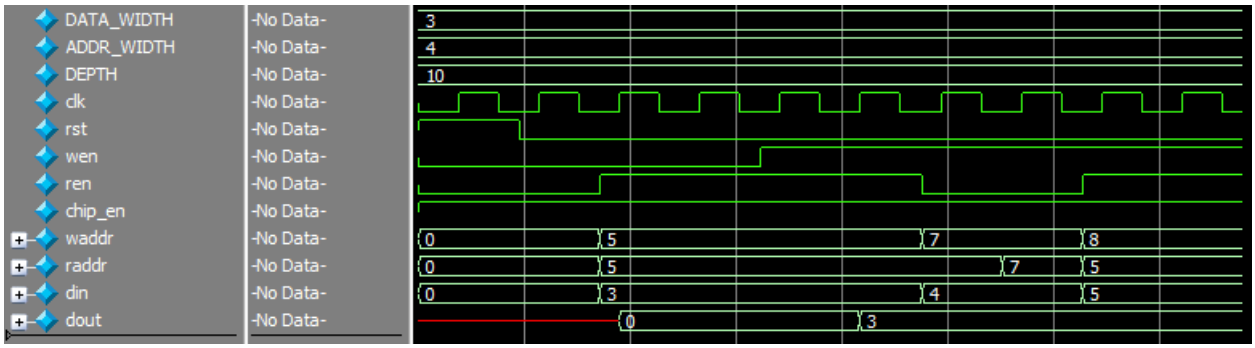
این ماژول را با کمک تست بنچ درستی آزمایی می‌کنیم و خروجی زیر را در شبیه سازی میبینم که با خواسته ما مطابقت دارد.



نوع دوم SRAM Type ScratchPad است که از آن برای ذخیره کردن فیلترهای کانولوشن استفاده می‌کنیم. کد این ماژول مانند ماژول قبل است با این تفاوت که chip_en و ren دارد و همچنین خواندن از آن به صورت سنکرون صورت میگیرد.

```
module SRAM_type_scratchpad#(parameter DATA_WIDTH,ADDR_WIDTH,DEPTH)(chip_en,raddr,waddr,wen,ren,din,clk,rst, dout);
    input [ADDR_WIDTH-1:0] raddr,waddr;
    input chip_en,wen,ren,clk,rst;
    input [DATA_WIDTH-1:0] din;
    output reg [DATA_WIDTH-1:0] dout;
    reg [DATA_WIDTH-1:0] mem [0:DEPTH-1];
    integer i=0;
    always @(posedge clk) begin
        if (rst) begin
            for (i = 0; i < DEPTH; i = i + 1) begin
                mem[i] <= {DATA_WIDTH{1'b0}};
            end
        end
        else if (chip_en) begin
            if (wen)
                mem[waddr] <= din;
            if (ren)
                dout <= mem[raddr];
        end
    end
endmodule
```

با تست بنچی مشابه قبل این ماژول را راستی آزمایی میکنیم.



۲. دو ماژول بعدی برای خواندن داده از بافرهای ورودی و نوشتن داده‌ها در scratchpad ها هستند.

ماژول buffer_read_controller_IFMap این کار را برای دادهای IFMap انجام میدهد.

استیت های این کنترلر بدین صورت است:

IDLE: هنوز شروع به کار نکرده است.

READ: شروع به کار کرده ولی هنوز داده ای نخوانده.

READ_S: اولین داده سطر را خوانده ولی هنوز آخرین داده سطر فعلی را نخوانده.

READ_SE: هم کل داده های سطر فعلی را خوانده ولی هنوز داده های سطر بعد را نخوانده.

READ_SENS: کل داده های سطر فعلی و تعدادی از داده های سطر بعدی را خوانده ولی هنوز تمام داده های سطر بعد را نخوانده.

READ_SENSNE: کل داده های سطر فعلی و کل داده های سطر بعدی را هم کامل خوانده و در نتیجه scratchpad کامل پر شده.

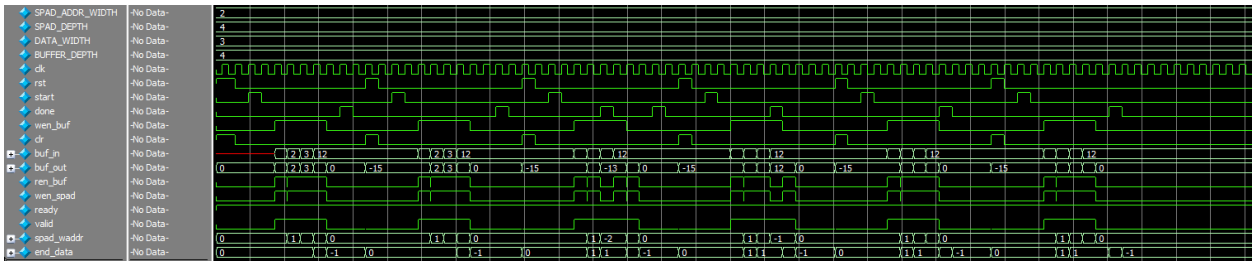
در این ماژول از یک شمارنده برای نگه داری آدرس نوشتن در scratchpad استفاده کردیم. همچنین چهار رجیستر داریم. دو تا برای ذخیره آدرس اولین و آخرین داده سطر فعلی و یکی برای ذخیره آدرس اولین و آخرین داده سطر بعدی.

تصویر کد ماژول:

```
module buffer_read_controller_IFMap#(parameter SPAD_ADDR_WIDTH = 3, SPAD_DEPTH = 7)(stall,clr_addr,valid,start_row,end_row,done,spad_raddr,init,clk,rst,
    output reg ren_buf,wen_spad,valid_end,valid_start;
    output [SPAD_ADDR_WIDTH-1:0] spad_waddr,start_data,end_data;
    wire safe_to_overwrite;
    wire [SPAD_ADDR_WIDTH-1:0] start_data_reg,end_data_reg,next_start_data,next_end_data,start_in,end_in;
    reg ld_s=0,sel_s=0,sel_e=0,ld_e=0,cnten=0;
    parameter [2:0] IDLE=0,READ=1,READ_S=2,READ_SE=3,READ_SENS=4,READ_SENSNE=5;
    reg [2:0] ns=IDLE,ps=IDLE;
    always@(posedge clk) begin
        if(rst)
            ps<=IDLE;
        else
            ps<=ns;
    end
    always@(*) begin
        ns=IDLE;
        case (ps)
            IDLE: begin ns = init ? READ : IDLE; end
            READ: begin ns = valid ? READ_S : READ; end
            READ_S: begin ns = (end_row && valid) ? READ_SE : READ_S; end
            READ_SE: begin ns = done ? READ : (valid && safe_to_overwrite) ? READ_SENS : READ_SE; end
            READ_SENS: begin ns = done ? READ_S : end_row ? READ_SENSNE : READ_SENS; end
            READ_SENSNE: begin ns = done ? READ_SE : READ_SENSNE; end
            default: ns= IDLE;
        endcase
    end
    always@(*) begin
        {ren_buf,wen_spad,cnten,ld_s,sel_s,ld_e,sel_e,valid_end,valid_start} = 9'd1;
        case (ps)
            IDLE: valid_start=1'b0;
            READ: {ren_buf,wen_spad,cnten,ld_s,valid_start,sel_s,ld_e,sel_e,valid_end} = {{5{valid}},4'b0000};
            READ_S: {ren_buf,wen_spad,cnten,ld_s,sel_s,ld_e,sel_e,valid_end} =
                {{3{valid && safe_to_overwrite}},2'b00,(valid && end_row),1'b0,(valid && end_row)};
            READ_SE: {ren_buf,wen_spad,cnten,ld_s,sel_s,ld_e,sel_e,valid_end} = {{3{valid && !done && safe_to_overwrite}},4'b0000,1'b1};
            READ_SENS: {ren_buf,wen_spad,cnten,ld_s,sel_s,ld_e,sel_e,valid_end} = {{3{valid && !done && safe_to_overwrite}},{2{done}},2'b00,1'b1};
            READ_SENSNE: {ren_buf,wen_spad,cnten,ld_s,sel_s,ld_e,sel_e,valid_end} = {3'b000,{4{done}},1'b1};
            default: {ren_buf,wen_spad,cnten,ld_s,sel_s,ld_e,sel_e,valid_end,valid_start} = 9'd1;
        endcase
    end
    assign start_data = (ld_s) ? start_in : start_data_reg;
    assign end_data = (ld_e) ? end_in : end_data_reg;
    assign safe_to_overwrite = (spad_waddr != start_data);
    assign start_in = sel_s ? next_start_data : spad_waddr;
    assign end_in = sel_e ? next_end_data : spad_waddr;
    counter #(.WIDTH(SPAD_ADDR_WIDTH),.WIDTH_INC(1))cnter(.max_count(SPAD_DEPTH[SPAD_ADDR_WIDTH-1:0]),.inc(1'b1),.cnt(cnten),
        .stall(stall),.clr(clr_addr),.clk(clk),.rst(rst), .pout(spad_waddr),.co());
    register #(.WIDTH(SPAD_ADDR_WIDTH))start_reg(.pin(start_in),.ld(ld_s),.stall(stall),.clk(clk),.rst(rst), .pout(start_data_reg));
    register #(.WIDTH(SPAD_ADDR_WIDTH))next_start_reg(.pin(spad_waddr),.stall(stall),.ld(start_row),.clk(clk),.rst(rst), .pout(next_start_data));
    register #(.WIDTH(SPAD_ADDR_WIDTH))end_reg(.pin(end_in),.ld(ld_e),.stall(stall),.clk(clk),.rst(rst), .pout(end_data_reg));
    register #(.WIDTH(SPAD_ADDR_WIDTH))next_end_reg(.pin(spad_waddr),.ld(end_row),.stall(stall),.clk(clk),.rst(rst), .pout(next_end_data));
endmodule
```

برای شبیه سازی این و برخی ماژول‌های دیگر نیاز به استفاده از بافر چرخشی بود که ما از بافر چرخشی تمرین قبلی استفاده کردیم.

خروجی شبیه سازی:



ماژول مشابه برای فیلترها بسیار ساده تر است چرا که فیلترها تغییر نمیکنند و تنها نیاز است به تعداد مورد نیاز خوانده شوند و بعد از آن ثابت خواهند ماند. استیت های این ماژول به طور زیر هستند:

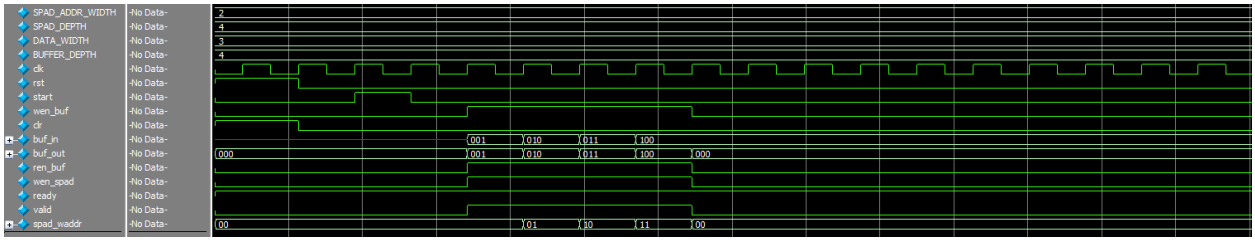
IDLE: هنوز شروع به کار نکرده یا کل فیلترها را خوانده و کارش را تمام کرده است.

READ: در حال خواندن فیلتر ها است.

تصویر کد ماژول:

```
module buffer_read_controller_Filter#(parameter SPAD_ADDR_WIDTH = 3, SPAD_DEPTH = 7)(valid,stall,init,clk,rst, valid_end,ren_buf,wen_spad,spad_waddr);
    input valid,stall,init,clk,rst;
    output reg valid_end,ren_buf,wen_spad;
    output [SPAD_ADDR_WIDTH-1:0] spad_waddr;
    reg cnten;
    wire co;
    parameter IDLE=0,READ=1;
    reg ns=IDLE,ps=IDLE;
    always@(posedge clk) begin
        if(rst)
            ps<=IDLE;
        else
            ps<=ns;
    end
    always@(*) begin
        ns=IDLE;
        case (ps)
            IDLE: begin ns = init ? READ : IDLE; end
            READ: begin ns = co ? IDLE : READ; end
            default: ns= IDLE;
        endcase
    end
    always@(*) begin
        {cnten,wen_spad,ren_buf,valid_end} = 4'd0;
        case (ps)
            IDLE: valid_end = 1'b1;
            READ: {cnten,wen_spad,ren_buf} = {3{valid}};
            default: {cnten,wen_spad,ren_buf,valid_end} = 4'd0;
        endcase
    end
    counter#(.WIDTH(SPAD_ADDR_WIDTH),.WIDTH_INC(1))cnter(.max_count(SPAD_DEPTH[SPAD_ADDR_WIDTH-1:0]),.inc(1'b1),
        .stall(stall),.cnt(cnten),.clr(init),.clk(clk),.rst(rst), .pout(spad_waddr),.co(co));
endmodule
```

تصویر خروجی شبیه سازی:



۳. ماژول بعدی، کنترل کننده نوشتن در بافر خروجی است که وظیفه دارد در صورت وجود نتیجه، آن را در بافر خروجی بنویسد و در صورتی که نتواند در بافر خروجی بنویسد سیگنال stall را فعال می‌کند که

این سیگنال به ماژول های مختلف میرود و ماژول های sequential را متوقف میکند تا نتیجه کانولوشن بعدی نیاید و نتیجه فعلی از بین نرود.

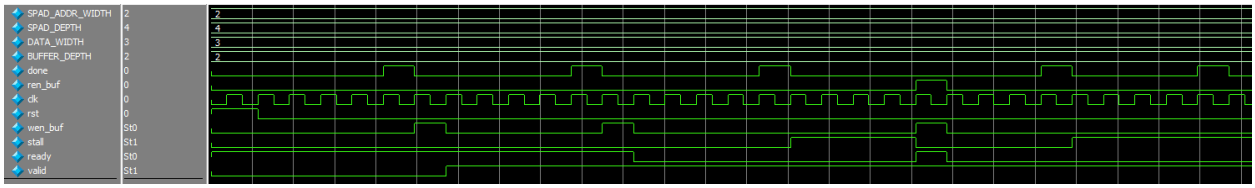
IDLE: هنوز خروجی برای نوشتن روی بافر چرخشی آماده نشده است.

WRITE: خروجی آماده نوشتن است و در این استیت آن قدر می‌مانیم که بافر چرخشی به ما اجازه نوشتن دهد و تا زمانی که اجازه نوشتن نداده، سیگنال stall را فعال نگه می‌داریم.

تصویر کد ماژول:

```
module buffer_write_controller(done,ready,clk,rst, wen,stall);
    input done,ready,clk,rst;
    output reg wen,stall;
    parameter IDLE=0,WRITE=1;
    reg ns=IDLE,ps=IDLE;
    always@(posedge clk) begin
        if(rst)
            ps<=IDLE;
        else
            ps<=ns;
    end
    always@(*) begin
        ns=IDLE;
        case (ps)
            IDLE: begin ns = (done) ? WRITE : IDLE; end
            WRITE: begin ns = (!ready) ? WRITE : IDLE; end
            default: ns= IDLE;
        endcase
    end
    always@(*) begin
        {wen,stall} = 2'd0;
        case (ps)
            IDLE:;
            WRITE: {wen,stall} = {ready,!ready};
            default: {wen,stall} = 2'd0;
        endcase
    end
endmodule
```

تصویر خروجی شبیه سازی:



۴. ماژول بعدی، تولید کننده آدرس خواندن از scratchpadها است. این ماژول آدرس خواندن از دو scratchpad را تولید میکند تا داده ها به pipeline بروند و محاسبات انجام شود.

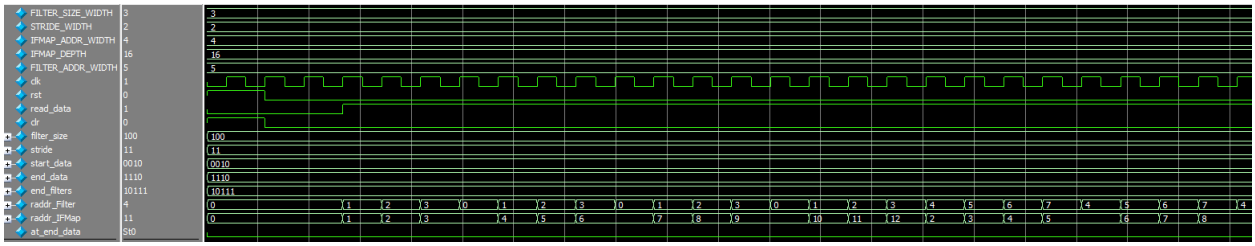
در این ماژول سه شمارنده وجود دارد: یک شمارنده نشان دهنده مکان ابتدای window مورد محاسبه در IFMap است. این شمارنده هر دفعه که کار یک پنجره تمام شد، به اندازه stride افزایش پیدا میکند. شمارنده بعد نگه دارنده آدرس اولین خانه فیلتر فعلی است. این شمارنده با هر بار پیمایش داده ورودی تا انتها، سپس به اندازه سایز فیلتر افزایش میابد. شمارنده آخر هم نشان دهنده خانه است که نسبت به این دو شمارنده باید به جلو برویم و مقادیر آن ها را به pipeline بفرستیم به همین جهت مقدار این شمارنده را با دو شمارنده دیگر جمع میکنیم تا آدرس خواندن از هر یک از scratchpadها به دست آید.

همچنین در این ماژول مشخص میشود که کار داده فعلی تمام شده یا نه یعنی تمامی فیلترها در داده فعلی ضرب شده اند و میتوان داده جدید را جایگزین کرد یا خیر. برای این کار، اگر به انتهای پنجره رسیدیم بررسی میکنیم که اگر پنجره بعدی جلوتر از انتهای داده فعلی بود پس دیگر نیازی به ادامه نیست و سیگنال at_end_data را فعال میکنیم. همچنین سیگنال co_pipe نیز با هر بار رسیدن به انتهای پنجره فعال میشود که این سیگنال مهم نیز به کنترلر اصلی میرود.

تصویر کد ماژول:

```
module read_address_generator#(parameter FILTER_SIZE_WIDTH,STRIDE_WIDTH,IFMAP_ADDR_WIDTH,FILTER_ADDR_WIDTH,IFMAP_DEPTH)
(stall,clr_addr,read_data,filter_size,stride,start_data,end_data,end_filters,valid_end,clk,rst,
raddr_Filter,raddr_IFMap,at_end_data,co_pipe);
    input stall,clr_addr,read_data,valid_end,clk,rst;
    input [STRIDE_WIDTH-1:0] stride;
    input [FILTER_SIZE_WIDTH-1:0] filter_size;
    input [IFMAP_ADDR_WIDTH-1:0] start_data,end_data;
    input [FILTER_ADDR_WIDTH-1:0] end_filters;
    output at_end_data,co_pipe;
    output [IFMAP_ADDR_WIDTH-1:0] raddr_IFMap;
    output [FILTER_ADDR_WIDTH-1:0] raddr_Filter;
    wire co_IFMap,co_Filter;
    wire [FILTER_SIZE_WIDTH-1:0] r_pipe;
    wire [IFMAP_ADDR_WIDTH-1:0] r_IFMap,addr_next_window;
    wire [FILTER_ADDR_WIDTH-1:0] r_Filter;
    counter #(.WIDTH(FILTER_SIZE_WIDTH),.WIDTH_INC(1))counter_pipe(.max_count(filter_size),.inc(1'b1),
        .stall(stall),.cnt(read_data),.clr(clr_addr),.clk(clk),.rst(rst), .pout(r_pipe),.co(co_pipe));
    counter_with_load #(.WIDTH(IFMAP_ADDR_WIDTH),.WIDTH_INC(STRIDE_WIDTH))counter_IFMap(.max_count(end_data),.inc(stride),
        .stall(stall),.pin(start_data),.cnt(co_pipe),.clr(clr_addr),.ld(co_IFMap | clr_addr),.clk(clk),.rst(rst), .pout(r_IFMap),.co());
    counter #(.WIDTH(FILTER_ADDR_WIDTH),.WIDTH_INC(FILTER_SIZE_WIDTH))counter_Filter(.max_count(end_filters),.inc(filter_size),
        .stall(stall),.cnt(co_IFMap),.clr(clr_addr | at_end_data),.clk(clk),.rst(rst), .pout(r_Filter),.co());
    adder #(.WIDTH_NUM1(FILTER_ADDR_WIDTH),.WIDTH_NUM2(FILTER_SIZE_WIDTH),.WIDTH_SUM(FILTER_ADDR_WIDTH))add_Filter
        (.num1(r_Filter),.num2(r_pipe), .sum(raddr_Filter));
    add_and_mod#(.WIDTH_NUM1(IFMAP_ADDR_WIDTH),.WIDTH_NUM2(FILTER_SIZE_WIDTH),.WIDTH_SUM(IFMAP_ADDR_WIDTH),
        .MOD(IFMAP_DEPTH[IFMAP_ADDR_WIDTH-1:0]))add_IFMap(.num1(r_IFMap),.num2(r_pipe), .sum(raddr_IFMap));
    assign co_Filter = (raddr_Filter==end_filters);
    assign at_end_data = co_Filter & co_IFMap;
    add_and_mod#(.WIDTH_NUM1(IFMAP_ADDR_WIDTH),.WIDTH_NUM2(STRIDE_WIDTH),.WIDTH_SUM(IFMAP_ADDR_WIDTH),
        .MOD(IFMAP_DEPTH[IFMAP_ADDR_WIDTH-1:0]))add_end_window(.num1(raddr_IFMap),.num2(stride), .sum(addr_next_window));
    assign co_IFMap = co_pipe & (addr_next_window > end_data || raddr_IFMap==end_data) & valid_end;
endmodule
```

تصویر خروجی شبیه سازی:



۵. آخرین کنترلر این تمرین، واحد کنترل اصلی است. این واحد کنترلی وظیفه فعال کردن سیگنال‌های شروع به کار و متوقف شدن ماژول۲های مختلف را بر عهده دارد. توصیف استیت‌های این واحد کنترلی:

IDLE: هنوز واحد کنترلی شروع به کار نکرده است و منتظر فعالی شدن سیگنال Start است.

WAIT: واحد کنترلی سیگنال Start فعال را دیده و الان منتظر غیر فعال شدن این سیگنال است.

INIT: سیگنال Start غیرفعال شده و واحد کنترلی با فعال کردن سیگنال init، ماژول‌های دیگر را آماده به کار می‌کند.

READ_NEW_DATA: واحد کنترلی منتظر دریافت داده‌های جدید از بیرون است. و تا وقتی داده ای ندارد در این استیت میماند و کاری نمیکند.

RUN: واحد کنترلی در حال فعالیت است و با توجه به سیگنال های دریافت از بخش های مختلف، سیگنال های لازم را ارسال میکند. این سیگنال ها بدین صورت هستند:

run_pipe: سیگنالی است که باعث لود شدن مقادیر جدید در رجیسترها و در نتیجه کارکرد پایپ لاین میشود.

read_spad: با فعال بودن این سیگنال، داده ها از scratchpadها خوانده میشوند.

clr_pipe: این سیگنال باعث صفر شدن مقادیر پایپ میشود تا داده جدید بتواند وارد شود.

done_psum: این سیگنال نوشتن نتیجه در بافر چرخشی خروجی است و کنترلر نوشتن میرود.

done_data: با فعال شدن این سیگنال کنترلر اصلی به کنترلر خواندن داده اعلام میکند که کارش با داده های فعلی تموم شده و میتوان داده های جدید را جایگزین کرد.

clr_addr: پس از تمام شدن کار یک سطر از داده در صورتی که داده جدیدی در بافر چرخشی ورودی نباشد این سیگنال فعال میشود تا شمارنده ها را صفر کند تا از اول در scratchpadها بنویسند.

تصویر کد این ماژول:

```
module main_control_unit(Start,wait_data,at_end_data,co_pipe,valid_start_addr,clk,rst,
init,run_pipe,read_spad,clr_pipe,done_psum,done_data,clr_addr);
    input Start,wait_data,at_end_data,co_pipe,valid_start_addr,clk,rst;
    output reg init,run_pipe,read_spad,clr_pipe,done_psum,done_data,clr_addr;
    parameter [2:0] IDLE=0,WAIT=1,INIT=2,READ_NEW_DATA=3,RUN=4;
    reg [2:0] ns=IDLE,ps=IDLE;
    always@(posedge clk) begin
        if(rst)
            ps<=IDLE;
        else
            ps<=ns;
    end
    always@(*) begin
        ns=IDLE;
        case (ps)
            IDLE: begin ns = Start ? WAIT : IDLE; end
            WAIT: begin ns = Start ? WAIT : INIT; end
            INIT: begin ns = READ_NEW_DATA; end
            READ_NEW_DATA: begin ns = wait_data ? READ_NEW_DATA : RUN; end
            RUN: begin ns = (!valid_start_addr & wait_data) ? READ_NEW_DATA : RUN; end
            default: ns= IDLE;
        endcase
    end
    always@(*) begin
        {init,run_pipe,read_spad,clr_pipe,done_psum,done_data,clr_addr} = 7'd0;
        case (ps)
            IDLE:;
            WAIT:;
            INIT: {clr_addr,init} = 2'b11;
            READ_NEW_DATA:;
            RUN: {run_pipe,read_spad,done_data,done_psum,clr_pipe,clr_addr} =
                {!wait_data,!wait_data,at_end_data,co_pipe & !wait_data,
                co_pipe & !wait_data,(!valid_start_addr & wait_data)};
            default: {init,run_pipe,read_spad,clr_pipe,done_psum,done_data,clr_addr} = 7'd0;
        endcase
    end
endmodule
```

۶. بقیه مسیرداده و پایپ لاین هم بیشتر کارهای محاسباتی را انجام میدهند.

تصویر کد مسیر داده:

```
module Processing_element#(parameter STRIDE_WIDTH,FILTER_SIZE_WIDTH,IFMAP_ADDR_WIDTH,IFMAP_DEPTH,FILTER_ADDR_WIDTH,FILTER_DEPTH,DATA_WIDTH)
    input clk,rst,Start,valid_IFMap,valid_Filter,ready_Psum;
    input [DATA_WIDTH+1:0] IFMap;
    input [DATA_WIDTH-1:0] Filter;
    input [STRIDE_WIDTH-1:0]stride_in;
    input [FILTER_SIZE_WIDTH-1:0]filter_size_in;
    output ren_buf_Filter,ren_buf_IFMap,wen_buf_Psum;
    output [DATA_WIDTH-1:0] Psum;

    wire wen_Filter,wen_IFMap,init,done_psum_ctrl,done_psum,done_data,stall,at_end_data,co_pipe,run_pipe,
    read_spad,wait_data,valid_start_data,valid_end_data,valid_end_filter,clr_pipe,clr_addr;
    wire [STRIDE_WIDTH-1:0]stride;
    wire [FILTER_SIZE_WIDTH-1:0]filter_size;
    wire [DATA_WIDTH-1:0] Filter_pipe,IFMap_to_reg,IFMap_pipe;
    wire [IFMAP_ADDR_WIDTH-1:0] raddr_IFMap,waddr_IFMap,start_data_addr,end_data_addr;
    wire [FILTER_ADDR_WIDTH-1:0] raddr_Filter,waddr_Filter,end_filters;
    assign end_filters = (FILTER_DEPTH / filter_size) * filter_size - 1;

    register #(.WIDTH(STRIDE_WIDTH))stride_reg(.pin(stride_in),.ld(init),.stall(stall),.clk(clk),.rst(rst), .pout(stride));
    register #(.WIDTH(FILTER_SIZE_WIDTH))filter_size_reg(.pin(filter_size_in),.ld(init),.stall(stall),.clk(clk),.rst(rst), .pout(filter_size));

    buffer_read_controller_IFMap#(.SPAD_ADDR_WIDTH(IFMAP_ADDR_WIDTH), .SPAD_DEPTH(IFMAP_DEPTH))read_controller_IFMap
        (.valid(valid_IFMap),.start_row(IFMap[DATA_WIDTH+1]),.end_row(IFMap[DATA_WIDTH]),.done(done_data),.spad_raddr(raddr_IFMap),
        .clr_addr(clr_addr),.init(init),.stall(stall),.clk(clk),.rst(rst), .valid_start(valid_start_data),.valid_end(valid_end_data),
        .ren_buf(ren_buf_IFMap),.wen_spad(wen_IFMap),.spad_waddr(waddr_IFMap),.end_data(end_data_addr),.start_data(start_data_addr));
    register_type_scratchpad #(.DATA_WIDTH(DATA_WIDTH),.ADDR_WIDTH(IFMAP_ADDR_WIDTH),.DEPTH(IFMAP_DEPTH))IFMapScratchPad
        (.raddr(raddr_IFMap),.waddr(waddr_IFMap),.wen(wen_IFMap),.din(IFMap[DATA_WIDTH-1:0]),.clk(clk),.rst(rst), .dout(IFMap_to_reg));
    register #(.WIDTH(DATA_WIDTH))reg_IFMap(.pin(IFMap_to_reg),.ld(read_spad),.stall(stall),.clk(clk),.rst(rst), .pout(IFMap_pipe));

    buffer_read_controller_Filter#(.SPAD_ADDR_WIDTH(FILTER_ADDR_WIDTH), .SPAD_DEPTH(FILTER_DEPTH))read_controller_Filter
        (.stall(stall),.valid(valid_Filter),.init(init),.clk(clk),.rst(rst), .valid_end(valid_end_filter),.ren_buf(ren_buf_Filter),.wen_spad(wen_Filter),.s
    SRAM_type_scratchpad #(.DATA_WIDTH(DATA_WIDTH),.ADDR_WIDTH(FILTER_ADDR_WIDTH),.DEPTH(FILTER_DEPTH))FilterScratchPad
        (.chip_en(1'b1),.raddr(raddr_Filter),.waddr(waddr_Filter),.wen(wen_Filter),.ren(!stall),.din(Filter),.clk(clk),.rst(rst), .dout(Filter_pipe));

    buffer_write_controller write_controller(.done(done_psum),.ready(ready_Psum),.clk(clk),.rst(rst), .wen(wen_buf_Psum),.stall(stall));

    read_address_generator#(.FILTER_SIZE_WIDTH(FILTER_SIZE_WIDTH),.STRIDE_WIDTH(STRIDE_WIDTH),.IFMAP_ADDR_WIDTH(IFMAP_ADDR_WIDTH),.IFMAP_DEPTH(IFMAP_DEPTH)
        .FILTER_ADDR_WIDTH(FILTER_ADDR_WIDTH))r_addr_gen(.stall(stall),.clr_addr(clr_addr),.read_data(read_spad),.filter_size(filter_size),.stride(stride),
        .raddr_Filter(raddr_Filter),.raddr_IFMap(raddr_IFMap),.start_data(start_data_addr),.end_data(end_data_addr),.end_filters(end_filters),
        .clk(clk),.rst(rst), .at_end_data(at_end_data),.co_pipe(co_pipe),.valid_end(valid_end_data));

    pipeline#(.WIDTH(DATA_WIDTH))pipe(.Filter(Filter_pipe),.IFMap(IFMap_pipe),.run(run_pipe),.done_psum_in(done_psum_ctrl),.stall(stall),
        .clr_pipe_in(clr_pipe),.clk(clk),.rst(rst), .Psum(Psum),.done_psum(done_psum));

    main_control_unit m_c_u(.Start(Start),.wait_data(wait_data),.at_end_data(at_end_data),.valid_start_addr(valid_start_data),.clr_addr(clr_addr),
        .co_pipe(co_pipe),.clk(clk),.rst(rst), .init(init),.run_pipe(run_pipe),.read_spad(read_spad),.clr_pipe(clr_pipe),.done_psum(done_psum_ctrl),.done_
    assign wait_data = ((!valid_end_filter & raddr_Filter == waddr_Filter) | (!valid_end_data & raddr_IFMap == waddr_IFMap)) | !valid_start_data;
endmodule
```

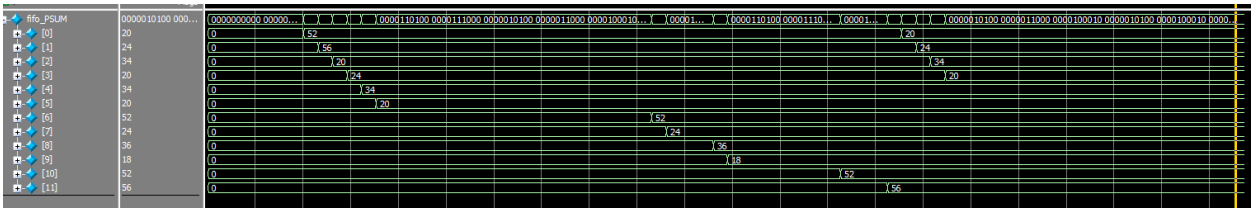
تصویر کد پایپ لاین:

```
module pipeline#(parameter WIDTH)(Filter,IFMap,run,clr_pipe_in,done_psum_in,stall,clk,rst, Psum,done_psum);
    input run,clr_pipe_in,done_psum_in,stall,clk,rst;
    input [WIDTH-1:0] Filter, IFMap;
    output done_psum;
    output [WIDTH-1:0] Psum;
    wire clr_pipe0,clr_pipe1,clr_pipe2;
    wire run1,run2,done_psum_reg;
    wire [WIDTH-1:0] mult_to_reg,in_reg_mult,num2_adder,mult_to_add,Psum,next_Psum;
    register #(.WIDTH(1))run_reg0(.pin(run),.ld(1'b1),.stall(stall),.clk(clk),.rst(rst), .pout(run1));
    register #(.WIDTH(1))run_reg1(.pin(run1),.ld(1'b1),.stall(stall),.clk(clk),.rst(rst), .pout(run2));

    assign num2_adder = clr_pipe2 ? {WIDTH{1'b0}} : Psum;
    multiplier #(.WIDTH(WIDTH))mult(.pin1(Filter),.pin2(IFMap), .pout(mult_to_reg));
    register #(.WIDTH(WIDTH))mult_reg(.pin(mult_to_reg),.ld(run1),.stall(stall),.clk(clk),.rst(rst), .pout(mult_to_add));

    register #(.WIDTH(1))reg_clr_pipe0(.pin(clr_pipe_in),.ld(run),.stall(stall),.clk(clk),.rst(rst), .pout(clr_pipe0));
    register #(.WIDTH(1))reg_clr_pipe1(.pin(clr_pipe0),.ld(run1),.stall(stall),.clk(clk),.rst(rst), .pout(clr_pipe1));
    register #(.WIDTH(1))reg_clr_pipe2(.pin(clr_pipe1),.ld(run2),.stall(stall),.clk(clk),.rst(rst), .pout(clr_pipe2));

    register#(.WIDTH(1))done_psum_reg0(.pin(done_psum_in),.ld(1'b1),.stall(stall),.clk(clk),.rst(rst), .pout(done_psum_reg));
    register#(.WIDTH(1))done_psum_reg1(.pin(done_psum_reg),.ld(1'b1),.stall(stall),.clk(clk),.rst(rst), .pout(done_psum));

    adder #(.WIDTH_NUM1(WIDTH),.WIDTH_NUM2(WIDTH),.WIDTH_SUM(WIDTH))add
        (.num1(mult_to_add),.num2(num2_adder), .sum(next_Psum));
    register #(.WIDTH(WIDTH))add_reg(.pin(next_Psum),.ld(run2),.stall(stall),.clk(clk),.rst(rst), .pout(Psum));
endmodule
```

پایپ لاین مشابه پایپ لاین داده شده در صورت پروژه است و تفاوت اصلی در این است که سیگنال های کنترلر نیز وارد پایپ و رجیسترها میشوند. در مسیر داده هم ماژول ها به هم متصو شده اند. هم چنین سیگنال wait_data ساخته میشود که در صورتی که آدرس خواندن و نوشتن فیلتر یا IFMap یکسان باشند این سیگنال فعال میشود که به معنی نبود داده است. همچنین از دو رجیستر برای ذخیره مقدار ورودی stride و filter_size استفاده میشود.

برای تست کلی ماژول تست بنچی طراحی کردیم و به صورت دستی مقادیر مورد انتظارمان را حسا کرده و با مقادیر خروجی مقایسه کردیم. همچنین از صحت عملکرد ماژول از لحاظ تاخیر زمانی نیز اطمینان حاص کردیم به طوریکه مشاهده کردیم ماژول پس از گذشتن تعدادی سیکل به اندازه filter_size داده بعدی را در خروجی مینویسد.

تصویر خروجی شبیه سازی:



تصویر تست بنچ کلی:

```
    always begin #19;clk=~clk;end
    initial begin
        #38;
        #38; rst=1'b0;
        #38; IFMap_in = {2'b10,10'd1};Filter_in = {10'd0};wen_IFMap = 1'b1;wen_Filter = 1'b1;
        #38; IFMap_in = {2'b00,10'd2};Filter_in = {10'd7};
        #38; IFMap_in = {2'b00,10'd3};Filter_in = {10'd6};
        #38; IFMap_in = {2'b00,10'd4};Filter_in = {10'd5};
        #38; IFMap_in = {2'b00,10'd3};Filter_in = {10'd4};
        #38; IFMap_in = {2'b00,10'd2};Filter_in = {10'd3};
        #38; IFMap_in = {2'b00,10'd1};Filter_in = {10'd2};
        #38; IFMap_in = {2'b01,10'd0};Filter_in = {10'd2};
        #38; IFMap_in = {2'b10,10'd1};Filter_in = {10'd1};
        #38; IFMap_in = {2'b00,10'd2};Filter_in = {10'd2};
        #38; IFMap_in = {2'b00,10'd3};Filter_in = {10'd3};
        #38; wen_IFMap = 1'b0;wen_Filter = 1'b0;
        #38; Start = 1'b1;
        #38; Start = 1'b0;
        #3800;
        #38; IFMap_in = {2'b00,10'd4};Filter_in = {10'd5};wen_IFMap = 1'b1;wen_Filter = 1'b1;
        #38; IFMap_in = {2'b01,10'd6};Filter_in = {10'd6};
        #38; wen_IFMap = 1'b0;wen_Filter = 1'b0;
        #380;
        #38; IFMap_in = {2'b10,10'd1};wen_IFMap = 1'b1;
        #38; IFMap_in = {2'b00,10'd2};
        #38; IFMap_in = {2'b00,10'd2};
        #38; IFMap_in = {2'b01,10'd2};
        #38; wen_IFMap = 1'b0;
        #1140;
        #38; IFMap_in = {2'b10,10'd1};wen_IFMap = 1'b1;
        #38; IFMap_in = {2'b00,10'd2};
        #38; IFMap_in = {2'b00,10'd3};
        #38; IFMap_in = {2'b00,10'd4};
        #38; IFMap_in = {2'b00,10'd3};
        #38; IFMap_in = {2'b00,10'd2};
        #38; IFMap_in = {2'b00,10'd1};
        #38; IFMap_in = {2'b01,10'd0};
        #38; IFMap_in = {2'b10,10'd1};
        #38; IFMap_in = {2'b00,10'd2};
        #38; IFMap_in = {2'b00,10'd3};
        #38; wen_IFMap = 1'b0;
        #380; ren_Psum=1;
        #3800;
        $stop;
    end
```