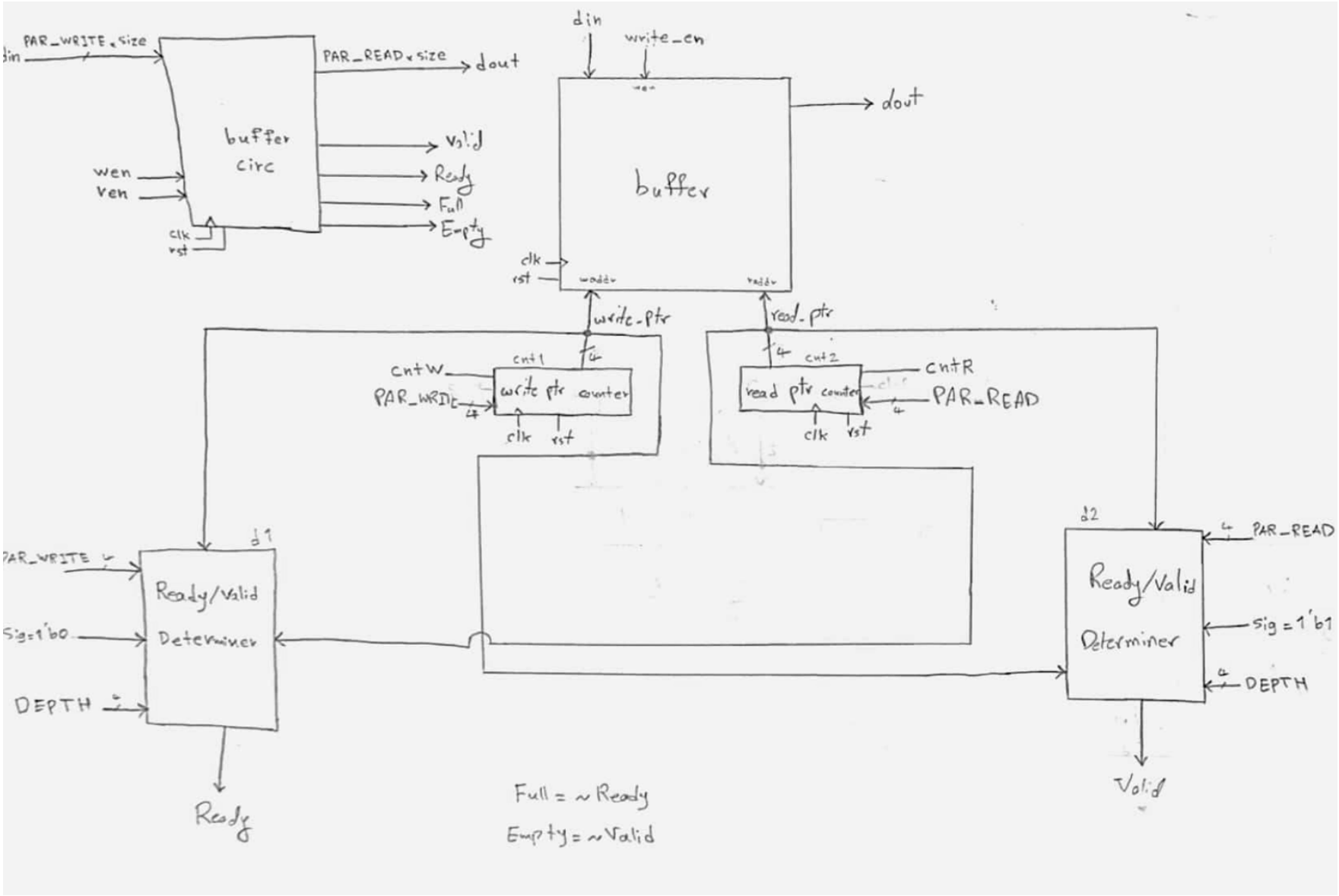
In the name of God

CAD CA2-Phase2 Report

Soroush Esfahanian 810101376

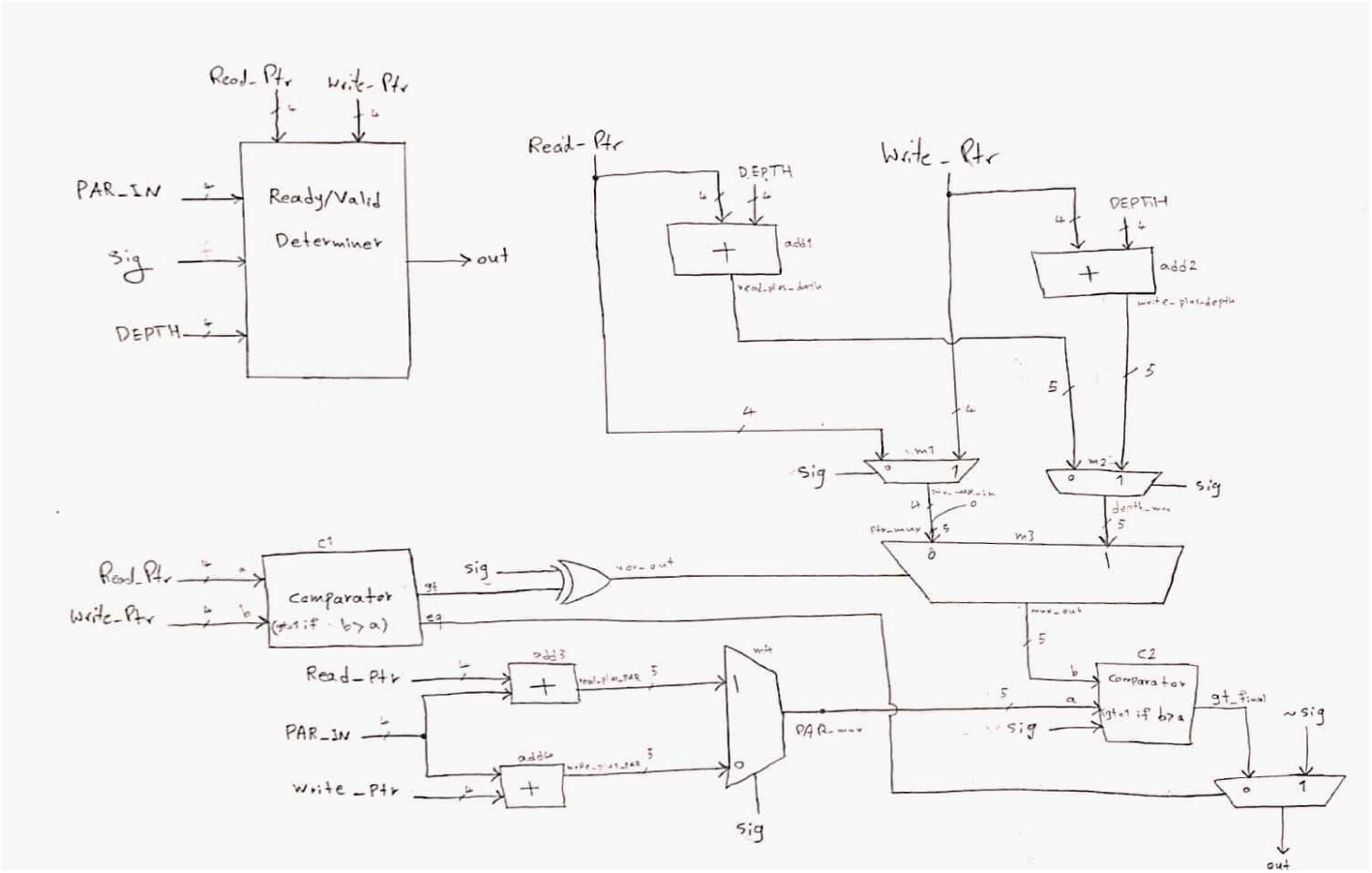
Kasra Kashani 810101490

اجزای مسیر داده:



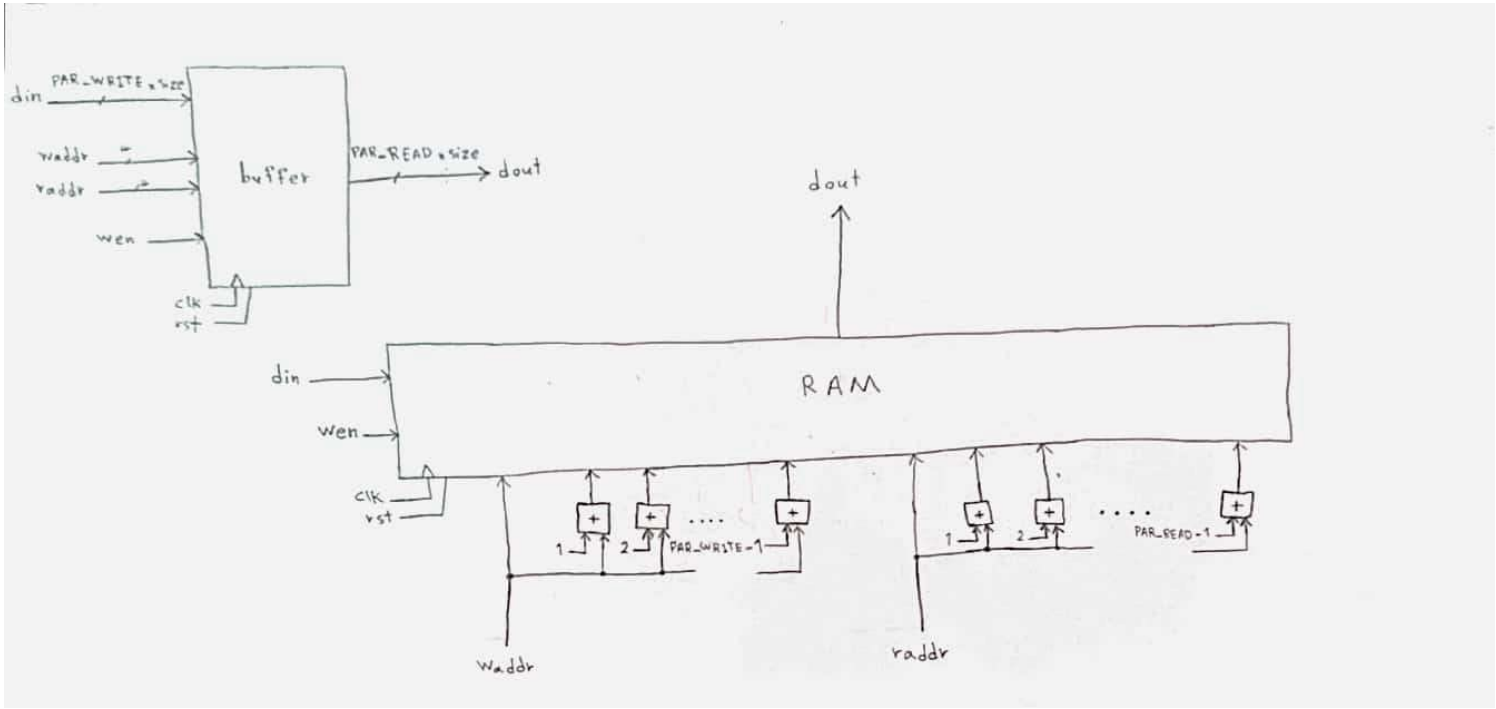
شکل بالا نماینگر مسیر داده کلی برای بافر حلقوی است. برای بافر حلقوی، ابتدا یک نمونه از بافر اصلی را که در ادامه توضیح خواهیم داد را قرار می‌دهیم که دو مقدار write_addr و read_ptr، ورودی‌های ما از مسیر داده به بافر هستند. برای کم و زیاد کردن نشانگرها، از یک counter برای زیاد کردن مقادیر آن‌ها استفاده می‌کنیم و با توجه به سیگنال ورودی که از کنترلر به مسیر داده می‌آید، مقادیر آن‌ها را به اندازهٔ PAR_READ و PAR_WRITE افزایش می‌دهیم.

گام اصلی بعدی در مسیر داده، مشخص کردن دو سیگنال ready و valid می‌باشد. برای اینکار، ابتدا یک ماژول determiner را تعریف می‌کنیم و از آن در دو موقعیت استفاده می‌کنیم. ساختار ماژول determiner به صورت زیر است:



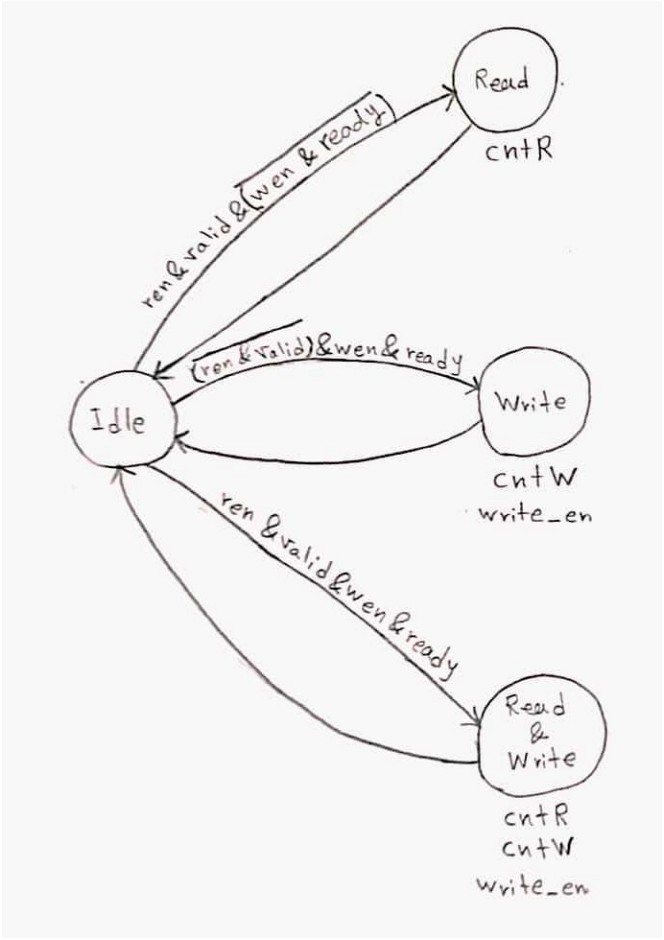
هدف کلی استفاده از ماژول determiner، مشخص نمودن سیگنال‌های ready و valid می‌باشد. برای اینکار، در دو مرحله مقایسه انجام می‌دهیم. ابتدا چک می‌کنیم مقدار نشانگر نوشتن از خواندن بیشتر می‌باشد یا خیر و با توجه به آن، و سیگنال ورودی sig که برای مشخص نمودن سیگنال ready/valid می‌باشد، به مقایسه‌های دیگر می‌پردازیم. مقایسه بعدی که مقایسه اصلی این ماژول نیز می‌باشد، بین یکی از مقادیر نشانگرها یا مقادیر آن‌ها به علاوهٔ عمق بافر، با مقدار نشانگر دیگر به علاوهٔ مقدار PAR مرتبط است. در واقع، در این مرحله چک می‌کنیم آیا به اندازه کافی برای نوشتن یا خواندن در بافر فضا وجود دارد یا خیر. همانطور که در شکل دیده می‌شود، با استفاده از سیگنال ورودی sig، مشخص می‌کنیم که مقایسه مربوطه برای به‌دست آوردن مقدار ready است یا مقدار سیگنال valid؛ همچنین در صورتی که مقادیر دو نشانگر با یکدیگر برابر باشند، به صورت مستقیم خروجی را به صورت نقیض سیگنال sig در نظر می‌گیریم.

در ادامهٔ به توضیح بافر معمولی می‌پردازیم:



در این ماژول، از یک RAM به عنوان هسته اصلی حافظه استفاده می‌کنیم که در واقع به صورت یک آرایهٔ دوبعدی در وریلاگ پیاده‌سازی می‌شود. از آن‌جایی که باید امکان خواندن و نوشتن از چند آدرس حافظه فراهم باشد، آدرس‌های حافظه ورودی برای هرکدام را به صورت مقادیر از صفر تا PAR – 1 با مقدار اولیهٔ آدرس در نظر می‌گیریم و بدین صورت خواندن و نوشتن به صورت همزمان میسر می‌شود.

توضیح کنترلر:



کنترلر مورد نظر تنها با استفاده از چهار استیت پیاده‌سازی شده‌است. استیت READ تنها برای خواندن و استیت READ تنها برای نوشتن و استیت READ & WRITE برای انجام همزمان هر دو عمل با رعایت ترتیب اولویت خواندن و نوشتن می‌باشد و همانطور که مشخص شده‌است، برای عبور از استیت اولیه به هرکدام از استیت‌ها، مقادیر سیگنال‌های ready و valid که نشان‌دهندهٔ وضعیت بافر هستند به همراه مقادیر wen و ren که مقادیر ورودی کاربر می‌باشند، در نظر گرفته شده‌است.

توضیح کلی پیاده‌سازی:

در این قسمت به طور مختصر، کدهای پیاده‌سازی پروژه را توضیح می‌دهیم. ابتدا آرگومان‌ها و ورودی‌های ماژول مسیر داده را می‌بینیم:

```
module datapath
    #(parameter SIZE,
    parameter DEPTH,
    parameter PAR_WRITE,
    parameter PAR_READ)
    (input clk, rst, write_en, cntW, cntR, input[(PAR_WRITE * SIZE)-1:0] din, output valid, ready, empty, full, output [(PAR_READ * SIZE)-1:0] dout);
    wire [3:0] write_ptr, read_ptr;
    wire [(PAR_READ * SIZE)-1:0] out_temp;
```

پارامترهای موردنظر برای مسیر داده به صورت parameter در وریلاگ تعریف می‌کنیم تا مسیر داده با توجه به پارامترهای موردنظر پیاده‌سازی شود. همچنین علاوه بر سیگنال های کلاک و ریست، din به عنوان دیتایی که باید نوشته شود به ماژول داده می‌شود که سایز آن همانطور که دیده می‌شود برابر با حاصلضرب PAR_WRITE در SIZE می‌باشد که مطابق با خواسته مسئله است. همچنین چهار سیگنال موردنظر صورت پروژه به عنوان خروجی مسیر داده در نظر گرفته می‌شود. علاوه بر آن‌ها، سیگنال dout به عنوان داده‌ای که باید از آن خوانده شود، به عنوان ورودی آخر به همراه سایز مربوطه در نظر گرفته شده‌است.

در ادامه کنترلر پیاده‌سازی شده را مشاهده می‌کنیم:

```
module controller(input clk, rst, ren, valid, wen, ready, output reg cntR, cntW, write_en);
```

همانطور که دیده می‌شود، ورودی ماژول علاوه بر کلاک و ریست، سیگنال‌های مربوطهٔ کنترلر که در قبل توضیح داده شد، به ماژول داده می‌شود. خروجی کنترلر هم شامل سیگنال کنترلی counterهای استفاده شده در مسیر داده موردنظر می‌باشد. همچنین سیگنال write_en به عنوان خروجی کنترلر به بافر داده می‌شود که به عنوان سیگنال کنترلی برای نوشتن در بافر استفاده خواهد شد. مورد بعدی در پیاده‌سازی کنترلر این است که با توجه به خواسته پروژه که باید نوشتن و خواندن در یک کلاک انجام شود، و ما این مورد را در دو استیت پیاده‌سازی کرده بودیم، به جای عوض کردن استیت‌ها در هر کلاک، به ازای هر نیم کلاک، یعنی posedge و negedge استیت‌ها را عوض می‌کنیم تا در انتها، به ازای هر یک کلاک، عملیات موردنظر انجام شود. پیاده‌سازی این مورد به صورت زیر است:

```
always @(posedge clk, negedge clk, posedge rst) begin
    if (rst) begin
        ps <= 3'b0;
        ns <= 3'b0;
    end
    else
        ps <= ns;
end
```

در ادامه فرم کلی ورودی و خروجی بافر کلی را خواهیم دید:

```
module buffer_circ
    #(parameter SIZE,
    parameter DEPTH,
    parameter PAR_WRITE,
    parameter PAR_READ)
    (input clk, rst, wen, ren, input [(PAR_WRITE * SIZE)-1:0] din, output valid, ready, full, empty, output [(PAR_READ * SIZE)-1:0] dout);
    wire write_en, cntW, cntR;

    datapath #(
        .SIZE(SIZE),
        .DEPTH(DEPTH),
        .PAR_WRITE(PAR_WRITE),
        .PAR_READ(PAR_READ))
        dp(clk, rst, write_en, cntW, cntR, din, valid, ready, empty, full, dout);
    controller cl(clk, rst, ren, valid, wen, ready, cntR, cntW, write_en);
endmodule
```

همانطور که دیده می‌شود، سیگنال‌های موردنظر که قبلا توضیح داده شده است، به عنوان ورودی و خروجی بافر حلقوی در نظر گرفته شده اند؛ همچنین پارامترهای موردنظر صورت پروژه، به صورت مقدار parameter در وریلاگ تعریف شده‌اند. در پیاده‌سازی ماژول کلی، از کنترلر و مسیر داده نمونه گرفته شده است و پارامترهای مربوطه به همراه مقادیر آرگومان‌ها به هرکدام داده شده‌اند.

پایان