

پروژه‌ی درس طراحی سیستم‌های دیجیتال

نام استاد: مهندس فصحتی

۱۴۰۳

دانشگاه صنعتی شریف

حورا عابدین

شماره دانشجویی: ۴۰۱۱۰۶۲۰۹

در این پروژه قصد داریم یک پردازنده‌ی آرایه‌ای ۵۱۲ بیتی مطابق توضیحات زیر پیاده سازی کنیم:

یک پردازنده آرایه‌ای^۱ ۵۱۲ بیتی برای اعداد صحیح طراحی کنید که دارای ۳ بخش زیر است:

- ۱) یک رجیستر فایل با قابلیت ذخیره‌سازی ۴ آرایه ۵۱۲ بیتی با نامهای A1 تا A4
- ۲) یک واحد ریاضی که قابلیت انجام ضرب و جمع را دارد. ورودی این واحد ریاضی A1 و A2 و خروجی کم‌ارزش آن در A3 و پردازش آن در A4 است.
- ۳) دارای یک حافظه با عمق ۵۱۲ و عرض ۳۲ بیت. این پردازنده امکان بارگزاری/ذخیره‌سازی ۱۶ خانه پشت‌سر هم از حافظه را دارد.

مجموعه دستورات این پردازنده شامل: ۱) بارگزاری از حافظه در یکی از ثبات‌ها ۲) ذخیره‌سازی از یکی از ثبات‌ها به حافظه^۳ جمع واحد ریاضی^۴ ضرب واحد ریاضی است.

این پردازنده را با زبان وریل‌اگ طراحی کنید و برای اطمینان از صحت طراحی مدار خود را برای حالات مرزی مورد آزمون قرار دهید. دقت کنید اگر ورودی‌ها/خروجی‌های دیگری مانند پرچم‌های وضعیت نیاز است به پردازنده بیفزایید (۶۰ نمره).

با توجه به مسئله‌ی مطرح شده، نیاز است ۴ مازول طراحی کنیم:

:RegisterFile .1

از این مازول و رجیسترها ۴ A1, A2, A3, A4 جهت انجام محاسبات و نگهداری مقادیر جهت نمایش و ... استفاده خواهیم کرد.

:ALU .2

درون این مازول، منطق ریاضی مورد نظر پیاده سازی می‌شود. مانند پردازنده‌های موجود، برای هر یک از عملیات‌های جمع و ضرب خواسته شده یک opcode در نظر گرفته شده است.

:Memory .3

این مازول جهت خواندن و نوشتمن از روی حافظه مورد استفاده قرار می‌گیرد.

:Vector Processor .4

در نهایت، این مازول با استفاده از ۳ مازول اول که در واقع بخش‌های تشکیل‌دهنده‌ی پردازنده هستند، خواسته‌ی مسئله را برآورده می‌کند.

همچنین نیاز است یک TB برای پردازنده‌ی خود بنویسیم که کارکرد آن را در حالت‌های مختلف مورد بررسی قرار دهد. در ادامه به طراحی هر یک از مازول‌ها و توضیحاتی در مورد هر یک می‌پردازیم.

طراحی مازولها:

RegisterFile

module:

```
≡ RegisterFile.v ×
C: > Users > voizo > OneDrive > Desktop > DSD_PROJ > ≡ RegisterFile.v
1   module RegisterFile (
2     input wire clk,
3     input wire [1:0] read_from_a_address,
4     input wire [1:0] read_from_b_address,
5     input wire [1:0] address_to_write,
6     input wire [511:0] data_to_write,
7     input wire write_enable,
8     output reg [511:0] data_from_a,
9     output reg [511:0] data_from_b
10    );
11    reg [511:0] A1, A2, A3, A4;
12
13    always @(posedge clk) begin
14      if (write_enable) begin
15        case (address_to_write)
16          2'b00: A1 <= data_to_write;
17          2'b01: A2 <= data_to_write;
18          2'b10: A3 <= data_to_write;
19          2'b11: A4 <= data_to_write;
20        endcase
21      end
22    end
23
24    always @(*) begin
25      case (read_from_a_address)
26        2'b00: data_from_a = A1;
27        2'b01: data_from_a = A2;
28        2'b10: data_from_a = A3;
29        2'b11: data_from_a = A4;
30      endcase
31      case (read_from_b_address)
32        2'b00: data_from_b = A1;
33        2'b01: data_from_b = A2;
34        2'b10: data_from_b = A3;
35        2'b11: data_from_b = A4;
36      endcase
37    end
38  endmodule
```

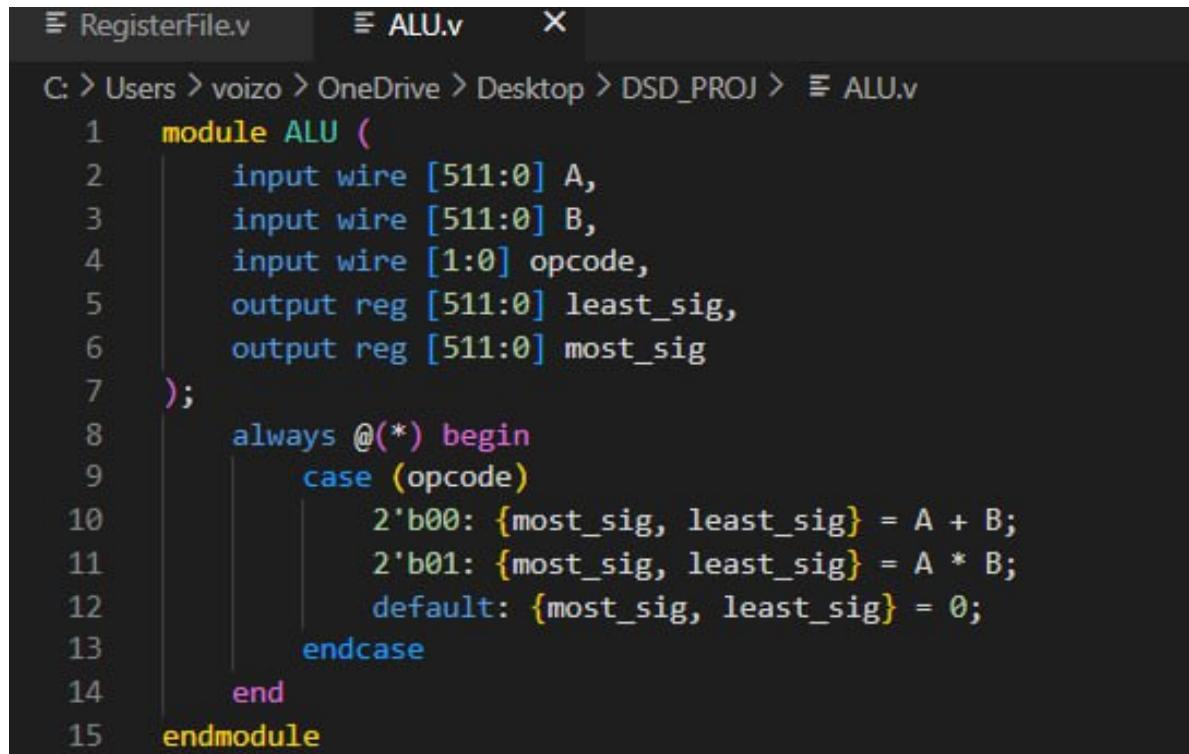
توضیحات:

مطابق با توصیف وریلگ بالا، مازول RegisterFile دارای چندین ورودی و ۲ خروجی است.
ورودی اول (clk) سیگنال کلک را مشخص می‌کند. -

- از آن جایی که در پردازنده با دو ورودی و عملیات روی آن مواجهیم، ورودی دوم و سوم آدرس رجیستری است که باید دیتا از آن خوانده شود. (طبق خواست سوال، ۴ رجیستر با نامهای A4, A3, ..., A1 داریم که برای هر یک آدرسی دوبیتی معادل شماره آن (۰ تا ۱۱) در نظر گرفته شده است.)
- ورودی چهارم آدرس رجیستر مقصد است که می‌خواهیم حاصل درون آن ریخته شود.
- ورودی پنجم مقداری است که قرار است در رجیستر مقصد نوشته شود.
- ورودی ششم مشخص می‌کند که آیا نوشتمن خواهیم داشت یا نه (۰ یا ۱)
- دو خروجی نیز مقدار خوانده شده از آدرس‌های داده شده هستند.

همچنین در بدنه‌ی مازول، ۴ رجیستر ۵۱۲ بیتی تعریف شده‌اند و با گذاشتن switch برای تعیین شماره‌ی رجیستر، خواندن و نوشتمن بر رجیسترها پیاده‌سازی شده است.

ALU module:



```

RegisterFile.v      ALU.v      X
C: > Users > voizo > OneDrive > Desktop > DSD_PROJ > ALU.v
1   module ALU (
2     input wire [511:0] A,
3     input wire [511:0] B,
4     input wire [1:0] opcode,
5     output reg [511:0] least_sig,
6     output reg [511:0] most_sig
7   );
8   always @(*) begin
9     case (opcode)
10       2'b00: {most_sig, least_sig} = A + B;
11       2'b01: {most_sig, least_sig} = A * B;
12       default: {most_sig, least_sig} = 0;
13     endcase
14   end
15 endmodule

```

توضیحات:

- مطابق با توصیف وریلگ بالا، مازول ALU دارای ورودی و خروجی‌های زیر است:
- ورودی اول و دوم، مقادیر A و B هستند که می‌خواهیم روی آن عملیات انجام دهیم.
- ورودی سوم، opcode است که تعیین می‌کند کدام عملیات باید انجام شود.
- ضرب: 01 / جمع: 00
- خروجی اول بخش کم ارزش جواب و خروجی دوم بخش پرازش جواب است.

همچنین در بدنه‌ی مازول، منطق محاسباتی پیاده‌سازی شده است که با کدام opcode چه عملیاتی انجام شود و حاصل برگردانده شود.

Memory module:

```

RegisterFile.v   ALU.v      Memory.v X
C:\> Users > voizo > OneDrive > Desktop > DSD_PROJ > Memory.v
1  module Memory (
2    input wire clk,
3    input wire [8:0] address,
4    input wire [511:0] data_to_write,
5    input wire write_enable,
6    output reg [511:0] rd_data
7  );
8    reg [31:0] memory [511:0];
9
10   always @ (posedge clk) begin
11     if (write_enable)
12       {memory[address+15], memory[address+14], memory[address+13], memory[address+12], memory[address+11], memory[address+10], memory[address+9], memory[address+8],
13      memory[address+7], memory[address+6], memory[address+5], memory[address+4], memory[address+3], memory[address+2], memory[address+1], memory[address]) <= data_to_write;
14   end
15
16   always @ (*) begin
17     rd_data = {memory[address+15], memory[address+14], memory[address+13], memory[address+12], memory[address+11], memory[address+10], memory[address+9], memory[address+8],
18                  memory[address+7], memory[address+6], memory[address+5], memory[address+4], memory[address+3], memory[address+2], memory[address+1], memory[address]};
19   end
20 endmodule

```

توضیحات:

مطابق با توصیف وریلگ بالا، مازول Memory دارای ورودی و خروجی‌های زیر است:

ورودی اول، آدرس خانه مورد برای شروع در حافظه را مشخص می‌کند.

ورودی دوم، مقداری است که می‌خواهیم در حافظه نوشته شود.

ورودی سوم مشخص می‌کند که آیا نوشتمن روی حافظه داریم یا خیر. (۰ و ۱)

خروجی نیز مقدار خوانده شده از خانه مشخص شده در حافظه است.

همچنین در بدنی مازول، مطابق خواست سوال، حافظه‌ای با عمق ۳۲ بیت و قابلیت نوشتمن تا ۱۶ خانه پشت سر هم پیاده‌سازی شده است.

VectorProcessor module:

```

RegisterFile.v      ALU.v       Memory.v      VectorProcessor.v X
C: > Users > voizo > OneDrive > Desktop > DSD_PROJ > VectorProcessor.v
1   module VecotrProcessor (
2     input wire clk,
3     input wire [8:0] memmory_address,
4     input wire [1:0] resister_a_address,
5     input wire [1:0] register_b_address,
6     input wire [1:0] register_address_to_write,
7     input wire [511:0] data_to_write,
8     input wire write_enable,
9     input wire [1:0] alu_opcode,
10    input wire mem_write_enable,
11    output wire [511:0] result
12  );
13  wire [511:0] register_a;
14  wire [511:0] register_b;
15  wire [511:0] alu_least_sig;
16  wire [511:0] alu_most_sig;
17  wire [511:0] memmory_data;
18  RegisterFile registerFile (
19    .clk(clk),
20    .read_from_a_address(resister_a_address),
21    .read_from_b_address(register_b_address),
22    .address_to_write(register_address_to_write),
23    .data_to_write(data_to_write),
24    .write_enable(write_enable),
25    .data_from_a(register_a),
26    .data_from_b(register_b)
27  );
28  ALU alu (
29    .A(register_a),
30    .B(register_b),
31    .opcode(alu_opcode),
32    .least_sig(alu_least_sig),
33    .most_sig(alu_most_sig)
34  );
35  Memory memory (
36    .clk(clk),
37    .address(memmory_address),
38    .data_to_write(data_to_write),
39    .write_enable(mem_write_enable),
40    .data_from_memmory(memmory_data)
41  );
42
43  assign result = memmory_data;
44 endmodule

```

توضیحات:

این مازول از به هم پیوستن زیرماژول‌ها به دست آمده که خواسته‌ی سوال را به طور کامل برآورده می‌کند. ورودی و خروجی‌ها دقیقاً منطبق بر خواست پروژه است. همچنین درون آن، از هر یک از زیرماژول‌های طراحی شده به ترتیب گرفته شده است تا وظیفه‌ی خود را جهت عملکرد صحیح پردازنده انجام داده و خروجی تولید شده را به بخش بعدی بدهند.

تست کردن و مورد آزمون قرار دادن پردازنده در حالات مختلف:

پیش از نوشتن TB، مأژولهای نوشته شده را در Modelsim کامپایل می‌کنیم تا از صحت آن‌ها اطمینان پیدا کنیم:

Name	Status	Type	Order	Modified
Memory.v	✓	Verilog	2	06/29/2024 03:19:52 ...
RegisterFile.v	✓	Verilog	0	06/29/2024 02:40:45 ...
ALU.v	✓	Verilog	1	06/29/2024 02:56:41 ...
VectorProcessor.v	✓	Verilog	3	06/29/2024 03:21:27 ...

اکنون یک TB می‌نویسیم که فرایند ورودی دادن و خروجی مورد نظر را شبیه سازی کند. در این تست، خروجی مورد انتظار را در کنار خروجی حاصل از پردازنده چاپ می‌کنیم تا بتوانیم صحت خروجی را چک کنیم.

* اعداد 512 بیتی به صورت hex در تست ورودی داده شده اند اما جهت سهولت خواندن، در پرینت به صورت decimal چاپ شده‌اند.

* از آن جایی که کد TB طولانی است، آن را به بخش‌های کوچک‌تر تقسیم کرده و توضیح داده‌ایم. سپس نتایج حاصل از شبیه‌سازی آورده شده است.

بخش اول: نمونه سازی از processor و مقداردهی‌های اولیه:

```

TB.v      X  RegisterFile.v  VectorProcessor.v
C: > Users > voizo > OneDrive > Desktop > DSD_PROJ >  TB.v
1  module TB;
2    reg clk;
3    reg [8:0] memmory_address;
4    reg [1:0] resister_a_address;
5    reg [1:0] register_b_address;
6    reg [1:0] register_address_to_write;
7    reg [511:0] data_to_write;
8    reg write_enable;
9    reg [1:0] alu_opcode;
10   reg mem_write_enable;
11   wire [511:0] result;
12
13  Processor processor (
14    .clk(clk),
15    .memmory_address(memmory_address),
16    .resister_a_address(resister_a_address),
17    .register_b_address(register_b_address),
18    .register_address_to_write(register_address_to_write),
19    .data_to_write(data_to_write),
20    .write_enable(write_enable),
21    .alu_opcode(alu_opcode),
22    .mem_write_enable(mem_write_enable),
23    .result(result)
24  );
25
26  initial begin
27    clk = 0;
28    forever #5 clk = ~clk;
29  end
30
31  initial begin
32    memmory_address = 0;
33    resister_a_address = 0;
34    register_b_address = 1;
35    register_address_to_write = 0;
36    data_to_write = 0;
37    write_enable = 0;
38    alu_opcode = 0;
39    mem_write_enable = 0;
40

```

یخشش دوم: تست اول برای جمع و ضرب برای مقادیر مشخص شده و مقابسه با حاصل مورد نظر:

بخش سوم: تست دوم برای جمع و ضرب برای مقادیر مشخص شده و مقابیسه با حاصل مورد نظر:

پیش چهارم: تست خواندن از حافظه و نوشتن روی آن:

نتیجه‌ی شبیه سازی:

با انجام شبیه سازی می بینیم که فرایند با موفقیت انجام می شود و نتایج تست ها به درستی نمایش داده می شود.

- * از آن جایی که اعداد بسیار بزرگ هستند، Wave نمایش خوبی برای نتیجه نیست و به نتیجه‌ی display اکتفا می‌کنیم.
 - * از آن جایی که اعداد بسیار بزرگ هستند، در عکس نتیجه کاملاً واضح نیست اما با run و مشاهده‌ی دقیق به صحت نتایج می‌رسیم.