بسم الله الرحمن الرحیم

پروژه درس طراحی سیستم های دیجیتال (DSD)

رسول الله صلی الله علیه وآله :

طَلَبُ العِلمِ فَریضَةٌ عَلی کُلِّ مُسلِمٍ أَلا إِنَّ الله یُحِبُّ بُغاةَ العِلمِ؛

طلب دانش بر هر مسلمانی واجب است. خداوند جویندگان دانش را دوست دارد.

مصباح الشریعه،ص۱۳

استاد: فرشاد بهاروند

افراد گروه:

امیر نژادملایری

مهدی شریف خانی

امین حسن‌زاده

محمد صالح شجاعی

محمد جواد علاءالدینی

تیر ۱۴۰۰

فهرست مطالب

مقدمه

در این پروژه قصد داریم به وسیله الگوریتم divide and conquer (تقسیم و غلبه) ضرب دو ماتریس را در زمانی کمتر از الگوریتم بدیهی بدست آوریم.

الگوریتم بدیهی ضرب دو ماتریس بدین صورت تعریف میشود که خانه سطر i ام و ستون j ام ماتریس نتیجه برابر مجموع ضرب خانه های سطر i ام ماتریس اول و ستون j ام ماتریس دوم میباشد. شبه کد این الگوریتم برای ضرب دو ماتریس دو در دو، به صورت زیر میباشد.

```
for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
                c[i][j]=0;
                for(int k = 0; k < 3; k++){
                        c[i][j] += a[i][k] * b[k][j];
                }
        }
}
```

اردر زمانی این الگوریتم با توجه به سه for تو در تو برابر O(n^3) میشود.

طبق الگوریتم پیشنهادی برای حل این سوال با استفاده از الگوریتم Strassen's به شرح زیر عمل میکنیم.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

A B C

ابتدا هر ماتریس را به چهار بخش تقسیم میکنیم. میدانیم مطابق الگوریتم بدیهی مقادیر خانه های زیر چگونه محاسبه میشوند.

اگر بخواهیم پس از تقسیم به چهار بخش همان محاسبات را انجام دهیم اردر زمانی محاسبه پاسخ به این صورت میشود. $T(N) = 8T(N/2) + O(N2)$

که میدانیم این مقدار همان O(N ^3) است و تفاوتی با الگوریتم اولیه ندارد. اما اگر متغییر ها را مطابق زیر تعریف کنیم اردر زمانی متفاوتی ایجاد میشود.

$p1 = a(f - h)$ $p2 = (a + b)h$
$p3 = (c + d)e$ $p4 = d(g - e)$
$p5 = (a + d)(e + h)$ $p6 = (b - d)(g + h)$
$p7 = (a - c)(e + f)$

و برای محاسبه مقادیر ماتریس C از p1 تا p7 استفاده کنیم.

$$C = \begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}$$

قسمت اصلی شبه کد این قسمت نیز مطابق زیر است.

```
# Computing the 7 products, recursively (p1, p2...p7)
p1 = strassen(a, f - h)
p2 = strassen(a + b, h)
p3 = strassen(c + d, e)
p4 = strassen(d, g - e)
p5 = strassen(a + d, e + h)
p6 = strassen(b - d, g + h)
p7 = strassen(a - c, e + f)

# Computing the values of the 4 quadrants of the final matrix c
c11 = p5 + p4 - p2 + p6
c12 = p1 + p2
c21 = p3 + p4
c22 = p1 + p5 - p3 - p7
```

که در این صورت داریم: $T(N) = 7T(N/2) + O(N2)$ که از $O(N^{Log7}) = O(N^{2.8074})$ است.

از بین دو الگوریتم مطرح شده gap بین $O(n^2)$ آن ها قابل بررسی می باشد که به بررسی آن در مثال های زیر میپردازیم.

تعداد کلاک های هر عمل جمع برابر است با ۳ و تعداد کلاک های هر عمل ضرب برابر است با ۱۳.

در نتیجه با محاسبه complexity time در دو الگوریتم زیر به این نتیجه میرسیم که الگوریتم ابتدایی بهینه تر است زیرا $O(n^2)$ کمتری تولید میکند و پس gap بین $O(n^2)$ دو عدد اینقدر زیاد است که باعث میشود مرتبه زمانی الگوریتم دوم کمتر از الگوریتم دوم شود. پس داریم:

a) $8*140 + 16*3 = 1168$
b) $7*145 + 18*4*3 = 1231$

در نتیجه واضح است که الگوریتم ابتدایی جواب بهتری میدهد.

توصیف معماری سیستم (Architecture Description of system and)

سیستم به طور کلی از ۵ ماژول تشکیل شده است. این چهار واحد هر کدام وظیفه انجام محاسبه خاصی را بر عهده داردند که به شرح زیر می باشد:

- Ram
- Single multiplier
- Adder
- Based multi matrix
- Matrix multiplier

برای تقسیم بندی ماتریس ها به روش devide and conquer مراحل زیر را طی میکنیم:

- ماتریس اول را به صورت ماتریس های ۲*۲ تقسیم بندی میکنیم.
- ماتریس دوم را به صورت ماتریس های ۲*۲ تقسیم بندی میکنیم.
- ماتریس های ۲*۲ متناظر را در هر ماتریس می یابیم و در هم ضرب میکنیم
- پاسخ ضرب های انجام شده را طبق الگوریتم تقسیم و غلبه با جملات مرتبط جمع میکنیم و درایه های ماتریس حاصل را به دست می آوریم.

اینترفیس های سیستم (ورودی‌ها، خروجی‌ها، مشخصات آنها، کلاک سیستم)

در پیاده سازی اینترفیس های این مدار از سه پارامتر n,s و r استفاده شده است که به ترتیب 2^n، ماکزیمم تعداد سطر ها و ستون های ماتریس های ورودی و خروجی، 2^s تعداد رجیستر های رم و از خانه r+1 ام رم ذخیره سازی ماتریس حاصله را شروع میکنیم.

- RAM (رم)

این واحد وظیفه ذخیره سازی اطلاعات را دارد. کلمات رم ۳۲بیتی میباشند و هر کلمه آدرس یکتایی دارد. تعداد خانه های رم به صورت پارامتری تعریف میشوند و به اندازه 2^s خانه فضا دارد. این رم قابلیت نوشتن یا خواندن در یک کلاک را دارد و با کلاک ورودی کار میکند. در صورتی که سیگنال write فعال باشد مقدار

ورودی در آدرس ورودی نوشته میشود. همچنین این رم همواره محتویات آدرس ورودی را به خروجی منتقل میکند.

- single multiplier

این واحد که به عنوان واحد آماده در اختیار ما قرار گرفته است و وظیفه آن ضرب اعداد اعشاری با قائده IEEE745 است.

- Adder

برای جمع کردن دو عدد واحد آماده ای در اختیار ما قرار داده شده است که دارای ۶ ورودی می باشد، number1 و number2 ورودی های ۳۲ بیتی است و result هم خروجی حاصل از مجموع این دو عدد می باشد.

- Based multi matrix

در این قسمت می خواهیم ماتریس پایه خود را درست کنیم که در آن دو عدد ماتریس ۲*۲ در هم ضرب می شوند.

در این واحد پس از ۱۶ کلاک خروجی مطلوب تولید می شود، همچنین در این واحد ۸ عدد ضرب کننده و ۴ عدد جمع کننده استفاده شده است. پس از ۱۳ کلاک، عملیات ضرب های موازی به اتمام می رسد و بعد از آن ۳ کلاک دیگر صورت می گیرد که ۴ عمل جمع به صورت موازی در آن زمان انجام می شود.

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}_{C}$$

- Matrix multiplier

محوری ترین ماژول طراحی شده در سیستم این ماژول است که وظیفه اصلی برقراری ارتباط بین ماژول ها را بر عهده دارد و تمام ورودی ها و خروجی های اصلی مدار در این ماژول تعریف میشود.

درون این ماژول یک Casestatemant وجود دارد که حالت های مختلف را دسته بندی میکند و state های آن به شرح زیر میباشد:

- State 0: منتظر StartBit می مانیم و پس از فعالسازی این سیگنال وارد وضیعت بعدی می شویم. اگر startbit فعال نبود در همین وضعیت می مانیم که اصطلاحا به این حالت، حالت wating می گوییم.
- State 1: در این مرحله conf را ست میکنیم که حاوی ابعاد ماتریس ها می باشد. همچنین پس از ست شدن conf بیت پنجم status برابر ۱ می شود.
- State 2: علمیات گرفتن ماتریس اول شروع می شود. بعد از آن بیت چهارم status برابر ۱ می شود. به اندازه تعداد درایه های ماتریس در این وضعیت می مانیم و بعد از پیمایش کامل ماتریس وارد مرحله بعدی می شویم.
- State 3: عملیات گرفتن ماتریس دوم شروع می شود. بعد از آن بیت سوم status برابر ۱ می شود. به اندازه تعداد درایه های ماتریس در این وضعیت می مانیم و بعد از پیمایش کامل ماتریس وارد مرحله بعدی می شویم
- State 4: عملیات تقسیم بندی به ماتریس های پایه ای انجام میشود و عملیات های محاسبه ای در این استیت انجام می شود، مانند فراخواندن ماژول های انجام دهنده ضرب و جمع. بعد از آن بیت دوم status برابر ۱ می شود. این وضعیت اصلی ترین بخش مدار می باشد که محوری ترین محاسبات عددی ماتریس ها در این وضعیت انجام می شود. هر کدام از بعد های ماتریس ها فرد بود به اندازه یک سطر یا ستون ماتریس را با درایه های صفر گسترش می دهیم. برای محاسبه هر ماتریس ۲*۲ از ماتریس حاصل ابتدا ماتریس صفر را جای آن گذاشته و مرحله به مرحله به روش شبه pipline ضرب ها را انجام داده و با مقادیر به دست آمده مرتبط از محاسبات قبلی جمع میکنیم.(با توجه به الگوریتم)
- State 5: ماتریس حاصله از مراحل قبلی را داخل رم ذخیره میکند. بعد از آن بیت اول status برابر ۱ می شود.

ورودی ها و خروجی های این واحد به صورت زیر می باشد:

- conf: در این ثبات ابعاد ماتریس های ما نگهداری میشود به این صورت که از خانه ۰ تا ۹ تعداد سطرهای ماتریس اول، ده خانه بعدی تعداد ستون های ماتریس اول و سطرهای ماتریس دوم، همچینن ده خانه بعدی تعداد سطون های ماتریس دوم می باشد.
- Status: خانه اول رم می باشد که مرحله به مرحله آپدیت می شود، برای اینکه سخت افزار کار خود را آغاز کند، cpu بیت ۳۱ ام از status را برابر با ۱ قرار می دهد. مراحل مختلف فرایند ضرب کردن با استفاده از بقیه بیت های این رجیستر مشخص می شود.
- State: مشخص کننده مرحله در casestatement مدار می باشد که در هر مرحله با توجه به شرایط وضعیت آن را مشخص میکنیم.
- Matrix1: رجیستر ماتریس اولی می باشد که در عملیات ضرب شرکت میکند.
- Matrix2: رجیستر ماتریس دومی می باشد که در عملیات ضرب شرکت میکند.
- MatrixRes: رجیستر ماتریس جواب می باشد که در عملیات ضرب تولید می شود.
- statusUpadate: این سیگنال مشخص می کند که چه هنگام باید مرحله تغییر کند. اگر سیگنال یک بود state مدار عوض می شود و اگر صفر بود مدار در همان state می ماند.
- Counterr,counterc: این دو رجیستر موقتی هستند که هر دفعه تعداد سطرها و ستون های ماتریس هایمان را ست میکند تا در درون case ها از آن استفاده کنیم.
- Regadder(i): ورودی جمع کننده می باشد.
- Adderack(i): پس از اماده سازی خروجی و فرستادن سیگنال result ready که فرایند را به پایان برساند و آماده جمع بعدی شود.
- Lastadderss: برای نگهداری آدرس قبلی از آن استفاده می شود.
- bmmIn1_i: ست کردن ورودی ماتریس اول برای انتقال به ماتریس پایه ای می باشد.
- bmmIn2_i: ست کردن ورودی ماتریس دوم برای انتقال به ماتریس پایه ای می باشد.
- bmmOut(i): خروجی های ماتریس پایه در این وایر ها وارد می شوند.
- Bmmstart: سیگنالی است که شروع شدن عملیات در ماژول پایه ای را اعلام می کند.

- BmmAck: سیگنالی است که به ماتریس پایه ای ارسال میشود و به معنی این است که ماتریس پایه ای اجازه دارد دستور بعدی را انجام دهد.
- bmmFinished: سیگنالی است که ماتریس پایه ای اعلام میکند عملیات تمام شده است.

سنتز (synthesis)

پس از تکمیل فرایند طراحی ماژول ها برای شبیه سازی مدار طراحی شده از ابزار vivado استفاده شده است که گزارش سنتز آن در تصویر زیر قابل مشاهده می باشد.

Report	Type	Options	Modified	Size
Synthesis				
Synth Design (synth_design)				
synth_1_synth_report_utilization_0	report_utilization		7/14/21, 10:54 PM	6.9 KB
synth_1_synth_synthesis_report_0			7/14/21, 10:54 PM	132.9 KB
Implementation				
impl_1				
Design Initialization (init_design)				
impl_1_init_report_timing_summary_0	report_timing_summary	max_paths = 10;		
Opt Design (opt_design)				
impl_1_opt_report_drc_0	report_drc			
impl_1_opt_report_timing_summary_0	report_timing_summary	max_paths = 10;		
Power Opt Design (power_opt_design)				
impl_1_power_opt_report_timing_summary_0	report_timing_summary	max_paths = 10;		
Place Design (place_design)				
impl_1_place_report_io_0	report_io			
impl_1_place_report_utilization_0	report_utilization			
impl_1_place_report_control_sets_0	report_control_sets	verbose = true;		
impl_1_place_report_incremental_reuse_0	report_incremental_reuse			
impl_1_place_report_incremental_reuse_1	report_incremental_reuse			
impl_1_place_report_timing_summary_0	report_timing_summary	max_paths = 10;		
Post-Place Power Opt Design (post_place_power_opt_design)				
impl_1_post_place_power_opt_report_timing_summary_0	report_timing_summary	max_paths = 10;		
Post-Place Phys Opt Design (phys_opt_design)				
impl_1_phys_opt_report_timing_summary_0	report_timing_summary	max_paths = 10;		
Route Design (route_design)				
impl_1_route_report_drc_0	report_drc			
impl_1_route_report_methodology_0	report_methodology			
impl_1_route_report_power_0	report_power			
impl_1_route_report_route_status_0	report_route_status			
impl_1_route_report_timing_summary_0	report_timing_summary	max_paths = 10;		
impl_1_route_report_incremental_reuse_0	report_incremental_reuse			
impl_1_route_report_clock_utilization_0	report_clock_utilization			
impl_1_route_report_bus_skew_0	report_bus_skew	warn_on_violation = true;		
impl_1_route_implementation_log_0				
Post-Route Phys Opt Design (post_route_phys_opt_design)				
impl_1_post_route_phys_opt_report_timing_summary_0	report_timing_summary	max_paths = 10; warn_on_violation = true;		
impl_1_post_route_phys_opt_report_bus_skew_0	report_bus_skew	warn_on_violation = true;		
Write Bitstream (write_bitstream)				
impl_1_bitstream_report_webtalk_0				
impl_1_bitstream_implementation_log_0				