دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارشکار آزمایشگاه معماری کامپیوتر – شماره ۸

عنوان آزمایش: بررسی ضرب کننده شیفت و جمع

نام و نام خانوادگی گردآورندگان : حسنا اویارحسینی و پویا محمدی

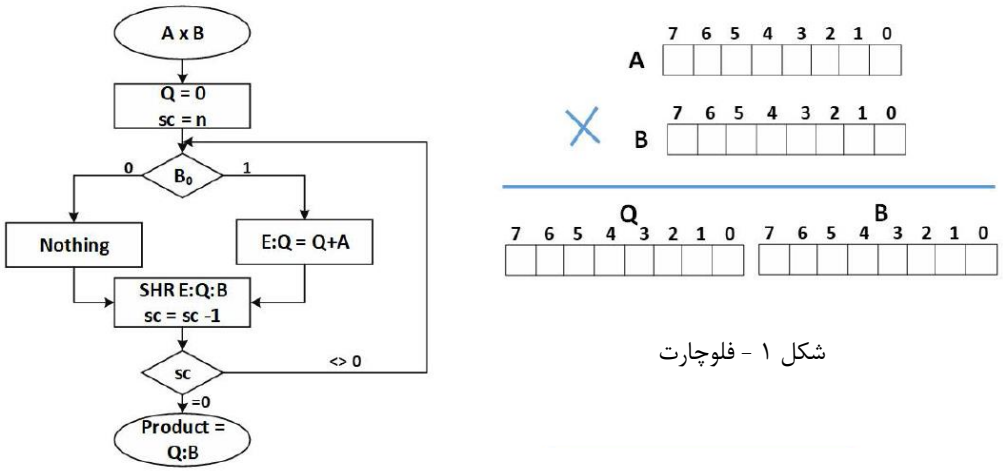
استاد آزمایشگاه: جناب آقای مهندس عاروان

تاریخ آزمایش: ۱۴۰۰/۹/۳

آزمایش۱)

نام آزمایش: پیاده سازی و بررسی ضرب کننده شیفت و جمع

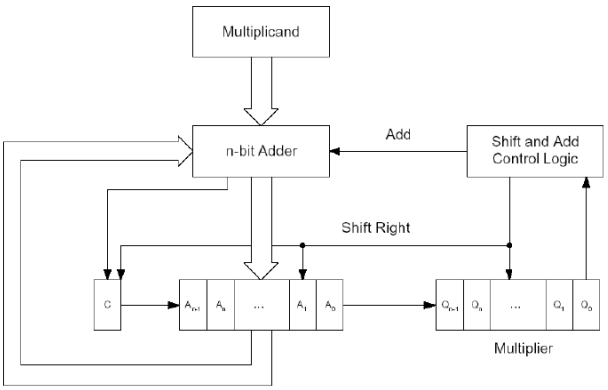
شرح آزمایش: در این آزمایش یک ضرب کننده شیفت و جمع ۴ بیت پیاده سازی و تست شده است. اساس کار این ضرب کننده به این صورت است که ابتدا به سمت راست ترین رقم عدد دوم (B) نگاه می شود و اگر صفر بود خروجی صرفا یک واحد به سمت راست شیفت پیدا میکند و اگر ۱ بود مقدار فعلی خروجی را یکبار با عدد اول جمع میزند و سپس حاصل را یک واحد به راست شیفت می دهد، و این کار را تا جایی ادامه میدهیم تا تمام بیت های B یکبار پیمایش شوند. لازم به ذکر است که برای خروجی نیاز به 2n بیت داریم که n بیت سمت چپ آن را در ثبات جدیدی به نام Q ، و n بیت سمت راست را در بخش خالی شده B قرار میدهیم(زیرا در هر مرحله که B یکی شیفت میابد و یک بیت از آن خالی میشود یک بیت به خروجی نهایی اضافه میشود) در واقع خروجی Q:B خواهد بود. در ادامه فلوچارت این ضرب کننده را مشاهده میکنید:



شکل ۱ - فلوچارت

برای پیاده سازی این جمع کننده مدار را به چند ماژول اصلی تقسیم میکنیم و همانند شکل ۲ آن ها را به هم مرتبط میسازیم :

۱- controller ۲- multiplicand ۳- multiplier ۴- جمع کننده.



شکل ۲ – مدار سطح بالا

در ادامه به توضیح عملکرد و پیاده سازی هر بخش می پردازیم:

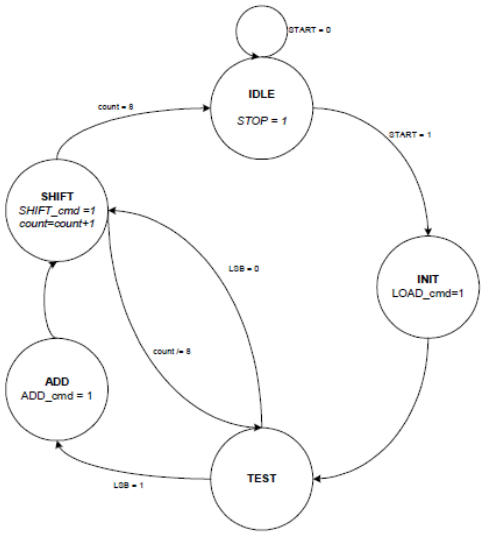
۱- Controller: Controller وظیفه کنترل ماژول های دیگر و تعیین عملیات لازم در هر مرحله را دارد. کنترلر در واقع یک ماشین حالت است که با هر کلاک با توجه به ورودی ها، state و خروجی های آن عوض میشود. ورودی ها و خروجی های این ماژول به شکل زیر می باشد:

خروجی ها		ورودی ها	
SHIFT_cmd	دستور شیفت به راست	ریست	reset
ADD_cmd	دستور جمع	کلاک	clk
LOAD_cmd	دستور لود کردن اعداد	شروع عملیات ضرب	START
STOP	پایان عملیات ضرب	کم ارزش ترین رقم multiplier	LSB

در ادامه باتوجه به شکل۳ به بررسی حالت های مختلف میپردازیم:

a. کنترلر در ابتدا در حالت IDLE می باشد و خروجی stop صفر می باشد، کار آن زمانی شروع میشود که سیگنال START را دریافت می کند و به حالت INIT تغییر وضعیت می دهد، در این زمان خروجی stop صفر شده و سیگنال خروجی LOAD_cmd فعال میشود که باعث میشود اطلاعات در ماژول های دیگر لود شود.
b. سپس کنترلر از وضعیت INIT به TEST تغییر میکند.
c. در مرحله بعد با توجه به مقدار LSB(که در واقع م ارزش ترین رقم multiplier می باشد) تصمیم میگیرد که به کدام حالت برود:
   c.1) اگر LSB = 1 باشد همانطور که در فلوچارت توضیح داده شد خروجی باید با multiplicand جمع شود و سپس مرحله بعد یعنی شیفت اتفاق بیافتد پس فعلا حالت کنترلر به ADD تغییر میکند و خروجی ADD_cmd فعال میشود.
   c.2) اگر LSB = 0 باشد نیاز به جمع نیست و مستقیم وارد حالت SHIFT میشویم.

d. اگر در وضعیت ADD قرار داشته باشیم (یعنی c.1 رخ داده باشد) در این مرحله کنترلر به SHIFT تغییر وضعیت می دهد.
e. هنگامی که در وضعیت SHIFT قرار داریم SHIFT_cmd فعال شده و به count یک واحد اضافه میشود سپس بررسی میشود اگر count برابر با n شده باشد (یعنی الگوریتم بر روی تمام بیت ها انجام شده و عملیات ضرب پایان یافته) پس به وضعیت IDLE میرویم و خروجیstop فعال میشود، در غیر این صورت به حالتTEST برمیگردیم تا الگوریتم بر روی بیت های باقی مانده هم انجام شود.



شکل ۳ – دیاگرام حالت کنترلر

۲- Multiplicand: Multiplicand یا همان عدد اول در ضرب را کافیست در یک ثبات ذخیره کنیم و در مواقع مورد نیاز(زمانی که میخواهیم خروجی را با آن جمع بزنیم) این عدد را از ثبات بخوانیم. برای پیاده سازی نیز کافیست به کمک ۴ DFF یک ثبات بسازیم. DFF را همان طور که در جلسات قبل پیاده سازی کرده بودیم به صورت behavioral تشکیل میدهیم، سپس برای کنار هم قرار دادن آنها از قطعه کد زیر استفاده میکنیم:

```
DFFs: for i in 3 downto 0 generate
    DFFReg:DFF port map (reset, LOAD_cmd, A_in(i), RA(i));
end generate;
```

توضیح دستور generate:

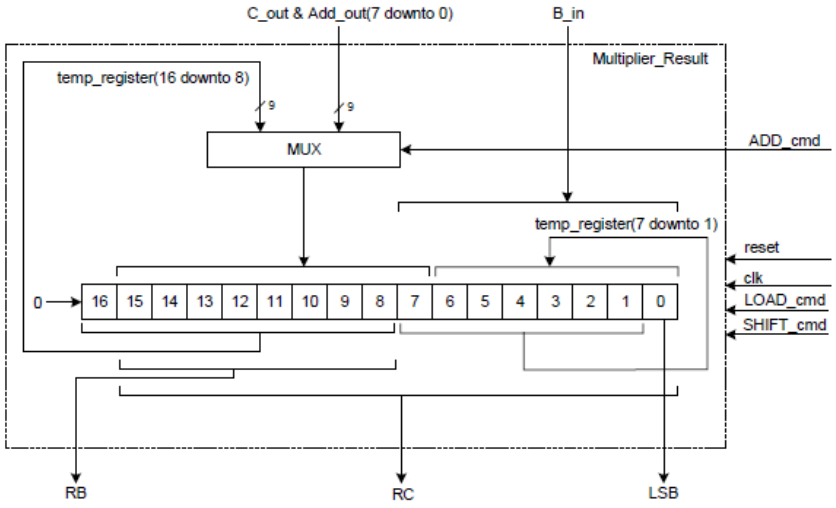
دستور for ...generate معمولاً برای ساخت آرایه ای از component ها استفاده می شود. برای مثال در مورد قطعه کد بالا ما به کمک generate ، ۴ DFF را که ورودی و خروجی هر کدام یک بیت از آرایه ورودی و خروجی است را ساخته ایم و در کنار هم نگهداری میکنیم.

- سینتکس generate به صورت زیر می باشد:

```
label: for parameter in range generate
        concurrent statements
end generate label;
```

۳- Multiplier: بلوک Multiplier/Result مقدار multiplier و همچنین خروجی adder را در یک ثبات ۹ بیتی ذخیره میکند و در هنگام نیاز تغییرات را بر روی آن اعمال میکند و در واقع نتیجه نهایی ضرب در این ثبات قرار خواهد گرفت. این طراحی که multiplier و خروجی جمع کننده را در کنار هم قرار داده ایم کمک میکند تا با شیفت این ثبات، هم ارزش عددی که در نیمه پر ارزش تر ثبات قرار

دارد یک بیت زیاد شود و در نتیجه به عنوان ورودی جمع بعدی مقدار صحیحی داشته باشد و هم باعث می شود تا LSB به بیت بعدی از multiplier اشاره کند و همچنین با هر بار شیفت یک بیت از multiplier دور ریخته میشود و به جای آن یک بیت به نتیجه نهایی افزوده میشود. همانطور که در شکل ۴ مشاهده میکنید بلوک Multiplier/Result از یک شیفت ثبات ۹ بیتی و یک مالتی پلکسر تشکیل شده است. که ثبات برای ذخیره و اعمال شیفت بر روی multiplier , result می باشد و ماکس نیز برای اینکه تصمیم بگیریم آیا نیاز است نتیجه نهایی را باmultiplicand جمع کنیم یا همان مقدار قبلی را نگهداریم (یعنی آیا به ازای بیت فعلی از multiplier یا همان LSB حاصل ضرب میانی غیر صفری تولید می شود یا خیر). البته لازم به ذکر است ما در پیاده سازی نهایی از آرایه به عنوان ثبات ۹ بیتی و از دستور شرطی if به جای مالتی پلکسر استفاده کرده ایم.



شکل ۴ – مدار داخلی multiplier

خروجی ها			ورودی ها		
به جمع کننده داده میشود	RC	بیت های ۰ تا ۷ ثبات	ریست	reset	
به کنترلر داده می شود	LSB	کم ارزش ترین بیت multiplier که همان بیت 0 ثبات است	کلاک	clk	
	RB	نیمه دوم ثبات (بیت ۴ تا ۷)	دستور شیفت به راست	SHIFT_cmd	از کنترلر دریافت می شود
			دستور جمع	ADD_cmd	از کنترلر دریافت می شود
			دستور لود کردن اعداد	LOAD_cmd	از کنترلر دریافت می شود
			multiplier	B_in	
			خروجی adder	Add_out	از جمع کننده دریافت میشود
				C_out	

توضیح بیشتر در رابطه با ورودی ها و خروجی ها:

LSB برای تعیین state مرحله بعدی به کنترل کننده برگشت داده می شود، در حالی که RB به عنوان ورودی به جمع کننده داده می شود تا با multiplicand جمع شود. RC نتیجه ضرب نهایی است و تنها زمانی معتبر تلقی می شود که کنترلر سیگنال STOP را فعال کند.

فعالیت هایی که در multiplier انجام میشود:

- هنگامی که ورودی LOAD_cmd که از کنترلر دریافت میشود، فعال باشد مقدار multiplier در بیت های کم ارزش ثبات ذخیره میشود و بقیه بیت ها صفر میشوند.

```
temp_register (8 downto 4) <= (others => '0');
temp_register(3 downto 0) <= B_in;  -- load B_in into register
```

- اگر در Result/Multiplier ورودی SHIFT_cmd یک باشد و ADD_cmd صفر باشد ، ثبات به سمت راست شیفت* داده می شود.

```
temp_register <= '0' & temp_register (8 downto 1);
```

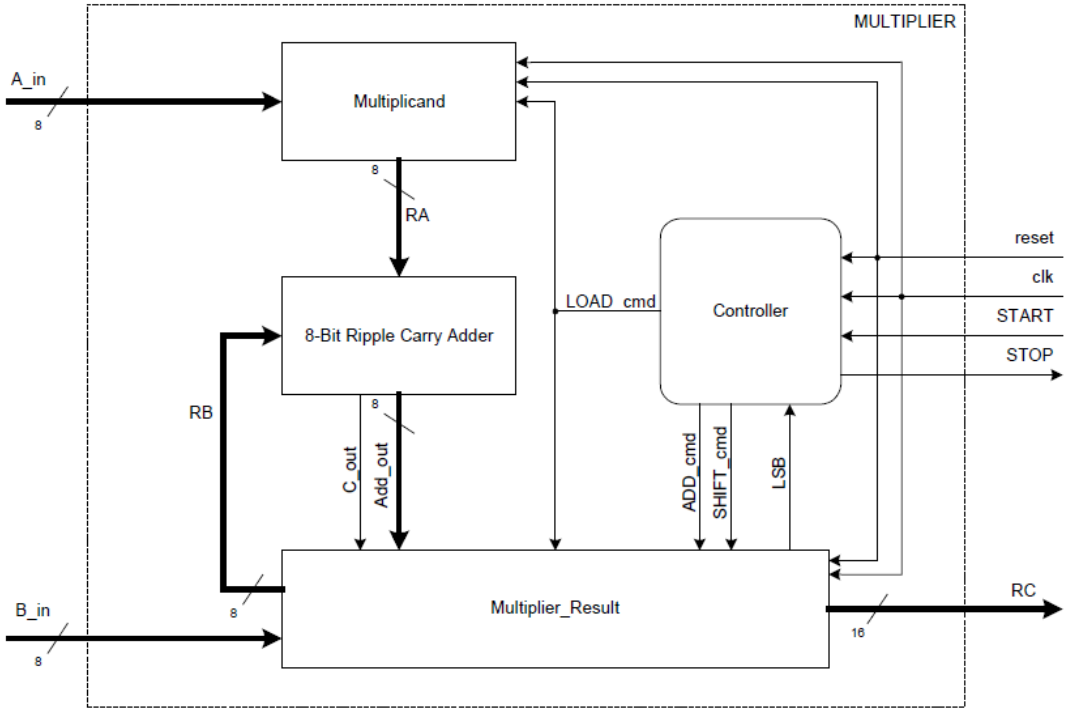
- اگر هر دو بیت SHIFT_cmd, ADD_cmd یک باشند ابتدا بیت ۴ تا ۷ ثبات به جمع کننده منتقل میشود و با multiplicand جمع میشود و حاصل دوباره روی بیت ۴ تا ۷ ذخیره میشود و سپس کل ثبات به راست شیفت* داده میشود.

```
temp_register <= '0' & C_out & Add_out & temp_register (3downto 1);
```

* عملیات شیفت به کمک مقدار دهی دوباره آرایه temp_register با مقدار شیفت یافته انجام شده است.

۴- جمع کننده: برای جمع کردن multiplicand و نیمه پرارزش جواب نهایی از یک جمع کننده ۴ بیت باید استفاده کنیم که ما در اینجا از 4bit-RCA استفاده کرده ایم که آن را به کمک ۴ FA و دستور generate ساخته ایم.

در نهایت مدار توصیف شده به صورت زیر خواهد بود:



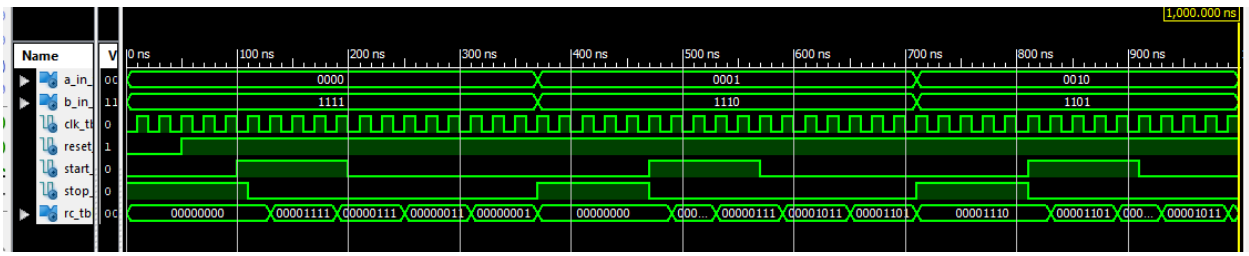
شکل۵ – مدار نهایی

نتایج شبیه سازی:

در این شبیه سازی برای برسی نتایج تمام حالات ممکن ضرب دوعدد ۴ بیتی در تست بنچ آورده شده است ولی با توجه به اینکه نمودار خروجی تا ns۱۰۰۰ را نمایش میدهد میتوانیم حاصل ضرب اعداد زیر را در خروجی مشاهده می کنیم:

multiplicand	multiplier	result
0000	1111	00000000
0001	1110	00001110
0010	1101	۰۰۰۱۱۰۱۰

خروجی شبیه سازی را در ادامه مشاهده میکنید:



شکل۶ – خروجی شبیه سازی