پروژه دوم - طراحی کامپیوتری سیستم های دیجیتال

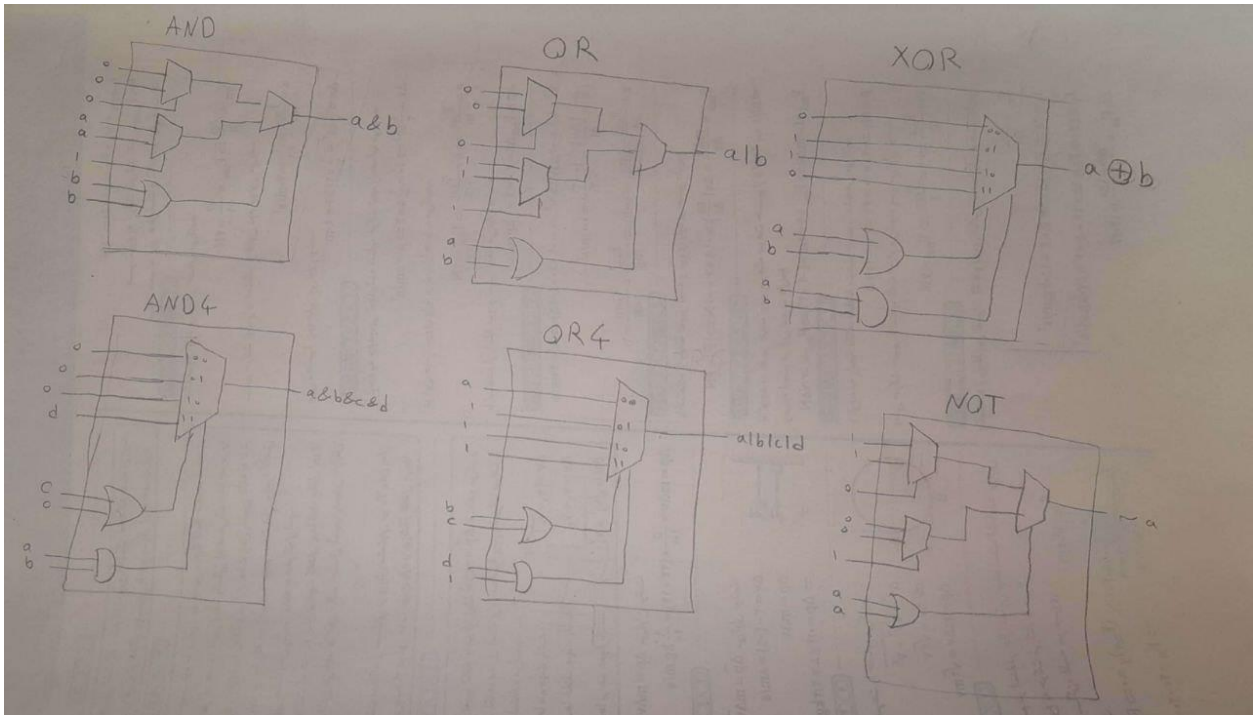
علی حمزه‌پور – ۸۱۰۱۰۰۱۲۹

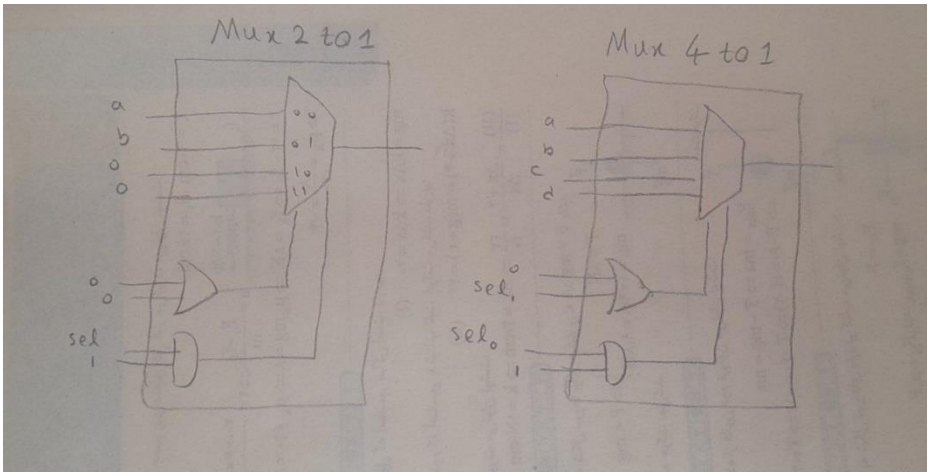
الهه خداوردی – ۸۱۰۱۰۰۱۳۲

طراحی گیت‌های اولیه

طرح‌های کلی کنترلر و دیتاپث نسبت به پروژه‌ی قبل هیچ تغییری نکردند و صرفا در این پروژه هر کدام از ماژول‌ها را با استفاده از ماژول‌های گفته شده پیاده سازی کردیم.

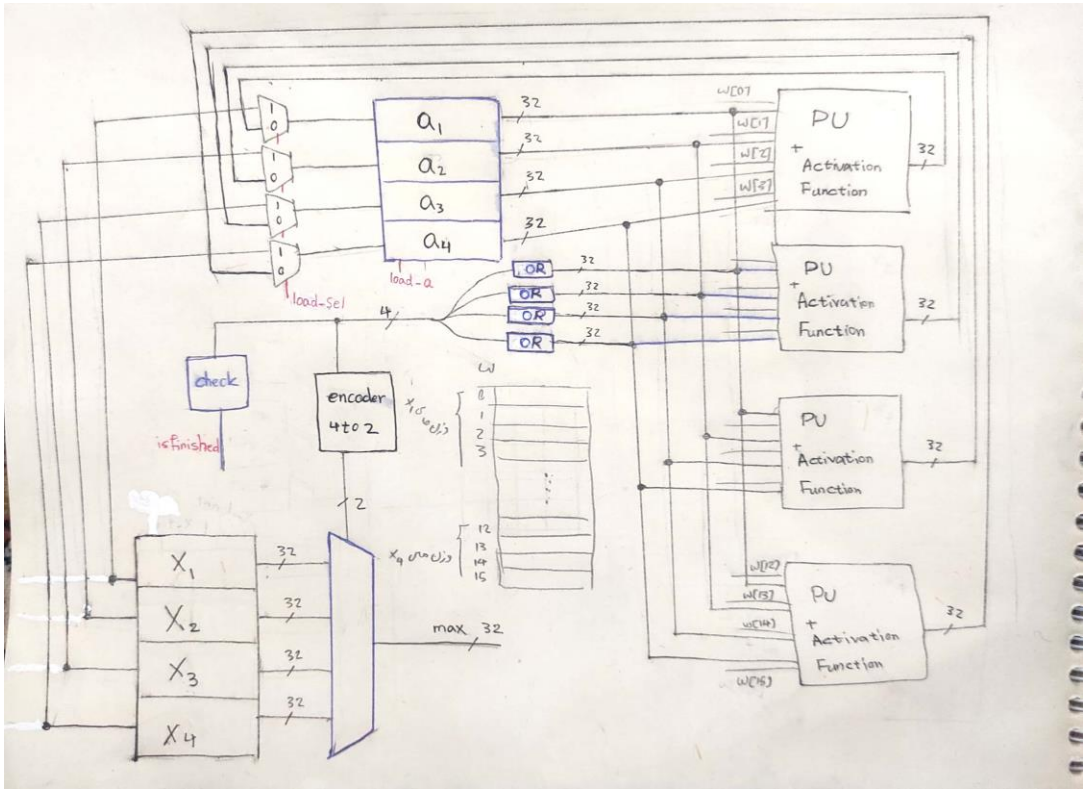
ابتدا قبل از طراحی ماژول‌های دیتاپث و کنترلر، گیت‌ها و ماکس‌های موردنیاز را با استفاده از ماژول‌های actel پیاده‌سازی کردیم:



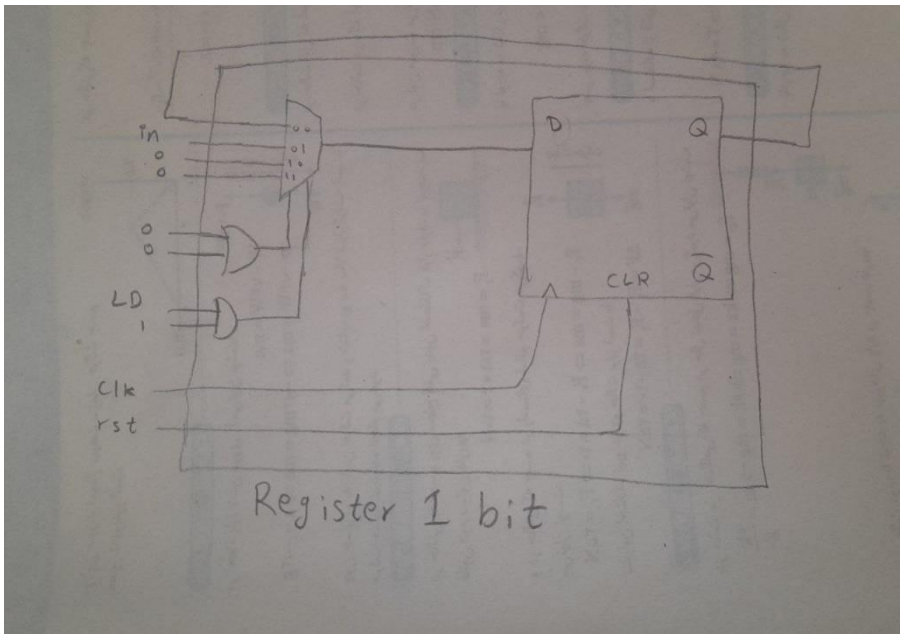


طراحی datapath

همانطور که گفته شد، طرح کلی دیتاپث نسبت به پروژه‌ی قبلی تغییری نکرده است:



- رجیستر را به صورت ترکیبی از چند ماژول S2 کنار هم طراحی کردیم.



- واحد check هم که در حقیقت بررسی می‌کند که آیا تمام اعداد جز یک عدد صفر شده‌اند یا خیر، با گیت‌ها قابل پیاده سازی هستند، زیرا در پروژه‌ی قبل هم به صورت ترکیبی از گیت‌ها پیاده‌سازی شده بود:

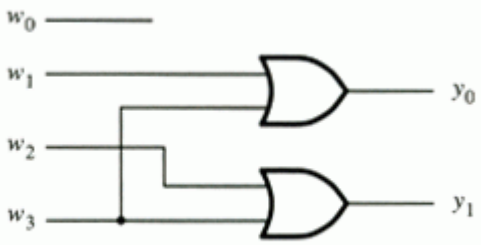
```
assign is_finished = (a == 4'b0001 || a == 4'b0010 || a == 4'b0100 || a == 4'b1000);
```

برای پیاده‌سازی عملگر "==" هم از چهار گیت xor استفاده کردیم.

- واحد encoder هم مانند شکل زیر با استفاده از گیت‌ها قابل پیاده‌سازی است:

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

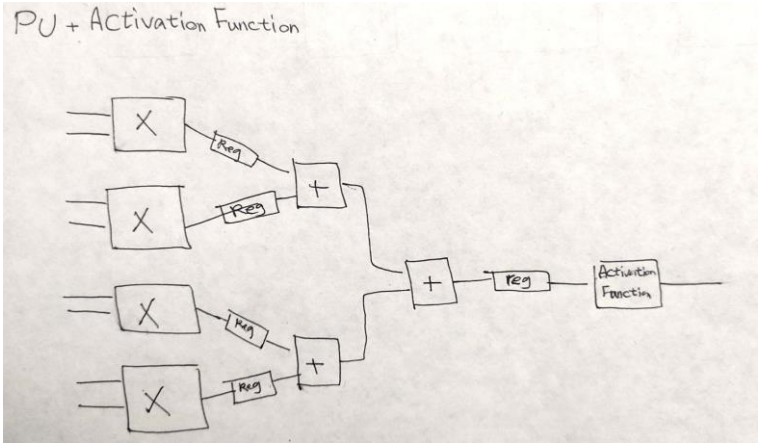
(a) Truth table



(b) Circuit

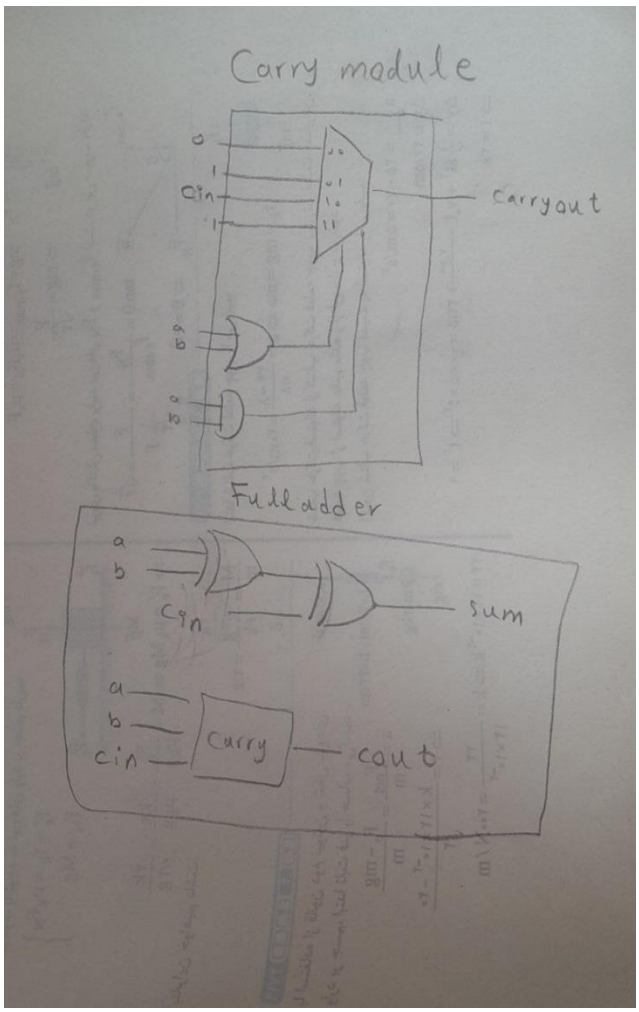
- واحد OR هم که در حقیقت چک می‌کند که عدد ورودی صفر است یا نه، صرفا یک or ۵تایی‌ست که با استفاده از تعدادی گیت or ساده قابل پیاده‌سازی است.

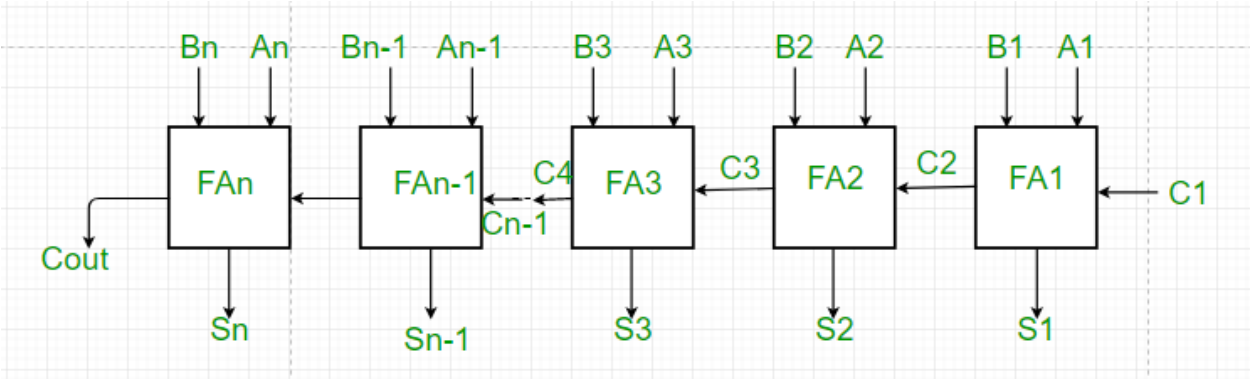
- طراحی کلی واحد PU نیز تغییری نکرده و صرفا طراحی داخلی جمع‌کننده، ضرب‌کننده و واحد فعال‌سازی را تغییر دادیم:



ماژول‌های PU

- جمع‌کننده: ابتدا یک full adder با استفاده از گیت‌های ساده ساختیم و سپس با کنار هم قرار دادن full adderها یک جمع‌کننده‌ی n بیتی ساختیم:





- ضرب‌کننده: با استفاده از array multiplier که با full adder و گیت‌های ساده قابل پیاده‌سازیست یک ضرب کننده ساختیم. برای اعداد منفی هم ابتدا هر ورودی را تبدیل به مقدار مثبت آن می‌کنیم، ضرب را انجام می‌دهیم و سپس در صورتی که جواب ضرب باید منفی می‌شد، جواب ضرب را منفی می‌کنیم. برای قرینه کردن نیز یک واحد 2's complement ساختیم.

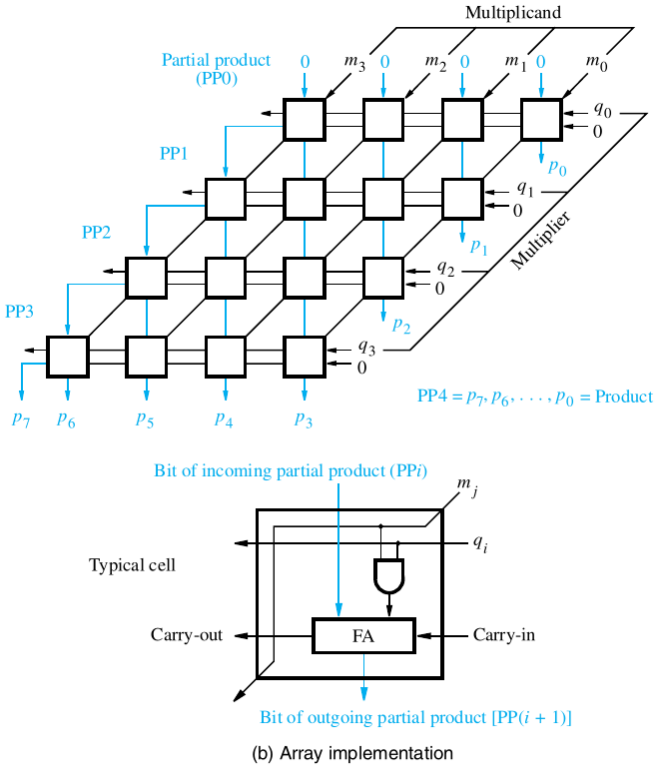
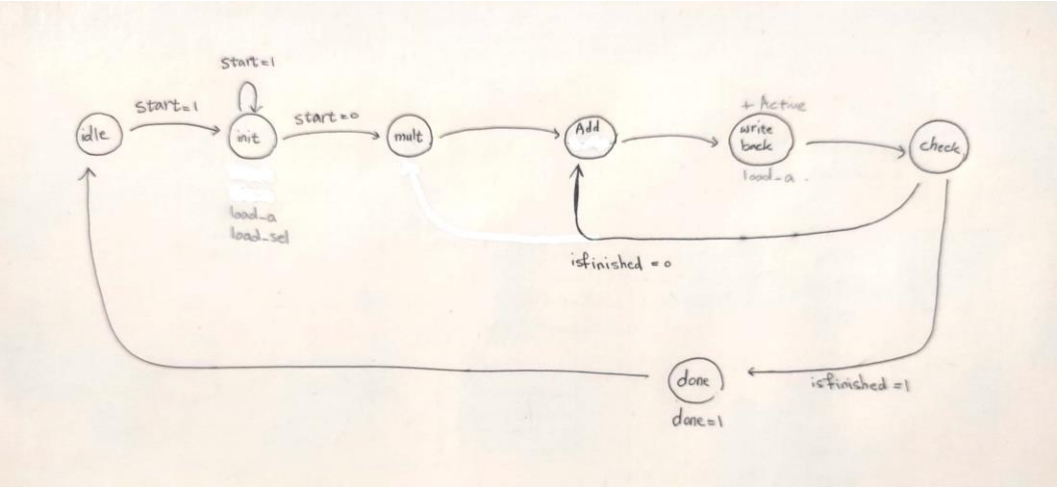


Figure 9.6 Array multiplication of unsigned binary operands.

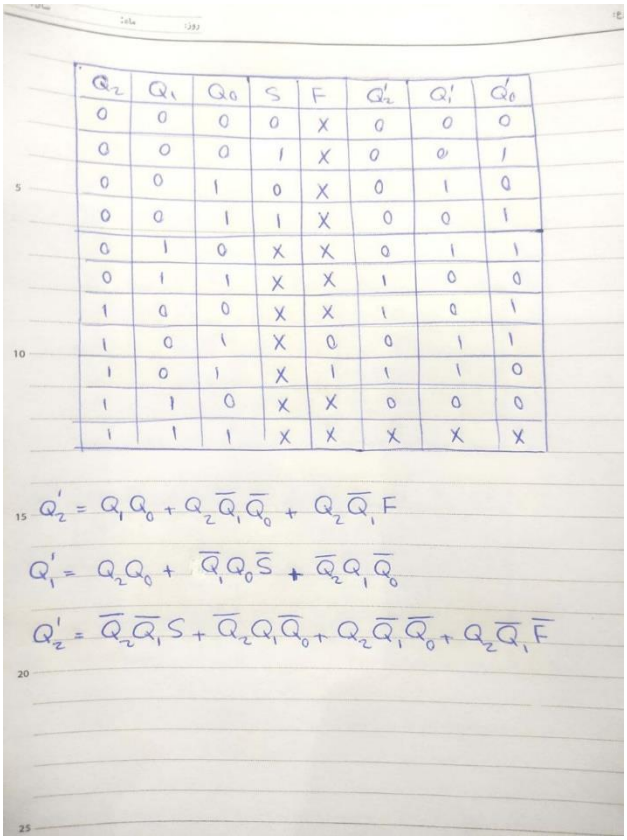
- فعال‌سازی: واحد فعال‌سازی یک ورودی ۱۲ بیتی که خروجی جمع لایه‌ی آخر هست را می‌گیرد و در صورتی که ورودی مثبت بود، پر ارزش‌ترین و کم ارزش‌ترین بیت‌های صحیح و ۳ بیت پرارزش اعشاری آن را خروجی می‌دهد و در غیر این صورت ۰ را خروجی می‌دهد. این ساختار با یک mux قابل پیاده‌سازی است.

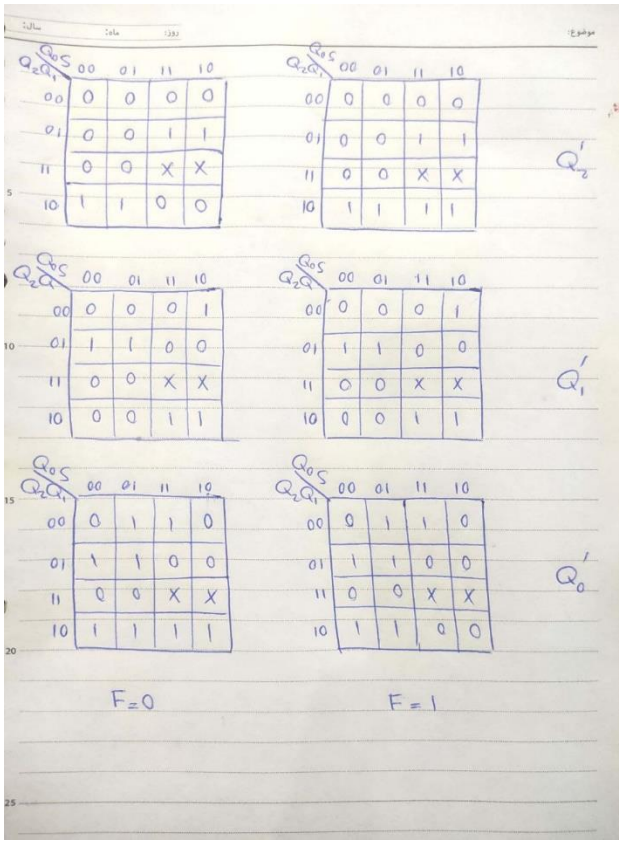
طراحی کنترلر

واحد کنترلر از لحاظ استیت‌ها و state digram هیچ تغییری نکرده است:



برای پیاده‌سازی کنترلر ابتدا جدول درستی استیت‌های بعدی را بر حسب استیت‌های فعلی و ورودی‌ها کشیدیم و سپس کارنو مپ هر بیت از استیت‌های بعدی را حل کردیم تا به یک عبارت بولی بهینه برای هر بیت از استیت برسیم:





در ادامه، عبارات بولی بدست آمده به راحتی با استفاده از گیت‌ها قابل پیاده‌سازی هستند و برای ذخیره بیت‌های هر استیت از D Flip Flop استفاده می‌کنیم که مانند یک رجیستر تک بیتی‌ست.

تست

ورودی زیر را تست می‌کنیم:

```
1  01101
2  00110
3  00101
4  01010
```

در این تست‌کیس، اعداد به ترتیب 1.625، 0.75، 0.375 و 1.25 هستند.

همانطور که می‌بینیم 01101 (1.625) به عنوان ماکسیمم خروجی داده شد که صحیح است.