گزارش پروژه معماری

# ALU

محمدمهدی آقاجانی

استاد زرندی

# توضیحات کلی

## Control unit

در این ماژول از سه ماژول واحد محاسبات ، واحد منطق و واحد رمزگشا استفاده شده است.

ورودی های a , b به هر یک از دو ماژول محاسبات و منطق داده میشوند و سیگنال های کنترلی توسط رمز گشا به این ماژول ها تحویل داده می شوند سپس با توجه به همین سیگنال ها ، خروجی مشخص شده و پرچم ها نیز معین می گردند.

## Arithmatic_unit

از چهار ماژول جمع کننده و تفریق کننده و تقسیم کننده و ضرب کننده تشکیل شده است که هر یک همزمان محاسبات را انجام می دهند و بعد نتیجه آنها با توجه به سیگنال کنترلی انتخاب میشود و در خروجی قرار می گیرد . در این ماژول با توجه به اینکه ضرب کننده و تقسیم کننده به صورت همگام پیاده سازی شده اند ورودی کلاک نیز وجود دارد

## Adder

در این ماژول از ماژول carry_look_ahead_adder دو بار نمونه سازی می شود و به صورت آبشاری به هم متصل می شوند هم چنین پرچم ها نیز در این ماژول معین می گردند.

## Carry_look_ahead_adder

در این ماژول با توجه به منطق look_ahead از فرمول های زیر استفاده شده است :

$$g(i) = a(i)\ and\ b(i)$$

$$p(i) = a(i)\ xor\ b(i)$$

$$c(i) = g(i) + p(i).c(i-1)$$

که با حدس زدن cin ها آن ها و ورودی ها را به FA ها می دهیم و s(i) ها را به خروجی وصل می کنیم.

## Full adder

پیاده سازی این ماژول نیز بار ها در کلاس مورد بحث قرار گرفته و کد آن نیز بسیار ساده می باشد.

## Subtractor

در این ماژول از ماژول borrow_look_ahead_subtractor چهار بار نمونه سازی شده است زیرا دوتا از این ماژول ها برای محاسبه a-b و دوتای دیگر برای محاسبه همزمان b-a است . همچنین از ماژول comprator نیز نمونه سازی شده تا a , b با هم مقایسه شوند و بر مبنای خروجی همین ماژول خروجی کلی تفریق کننده معین می شود. پرچم ها نیز بر مبنای همین خروجی همین مقایسه کننده انتخاب میشوند.

## Borrow_look_ahead_subtractor

در این ماژول با توجه به منطق look_ahead از فرمول های زیر استفاده شده است :

$$g(i) = not\ a(i) \quad and \quad b(i)$$

$$p(i) = a(i)\ xnor\ b(i)$$

$$c(i) = g(i) + p(i).c(i-1)$$

که در ادامه همانند جمع کننده عمل می کند.

## Full subtractor

در پیاده سازی آن از فرمول زیر استفاده شده است :

```
subtract  =  a xor b xor bin;
bout  =  ((not a) and (b or bin) ) or ( b and bin );
```

## multiplier

در این ماژول از ماژول adder نمونه سازی می شود برای اینکه برای محاسابت داخلی از آن استفاده می شود. در این ماژول از روش شیفت و جمع استفاده می شود که مطالب آن در کلاس مطرح شد که برای این کار ماژول جمع کننده در بیرون از process تعریف می شود و خروجی آن یک سیگنال میانی است که با توجه به ناهمگام بودن این جمع کننده در هر لحظه مجموع سیگنال های مورد نظر را بر روی سیگنال خروجی خود قرار می دهد و می توان از آن سیگنال درون process استفاده کرد.

برای کنترل ضرب کننده یک سیگنال do استفاده شده است که هر بار که کلاک بیاید و برابر یک باشد شروع به ضرب کردن می کند و در غیر این صورت اگر عمل ضرب تمام شده باشد و do صفر باشد سیگنال ها را مقدار دهی اولیه کرده و آماده دستور بعدی برای ضرب می شود.

## Divider

در این قسمت از الگوریتم تقسیم با بازیابی که در سر کلاس مطرح شد استفاده می شود . برای اینکار با توجه به اجازه از آقای دهباشی از عملیات جمع و تفریق استفاده می شود . برای این تقسیم کننده ابتدا شرایط سرریز بررسی می شود و اگر سرریز رخ می داد سیگنال مربوطه یک میشود و عملیات متوقف می گردد و در غیر این صورت وارد عملیات تقسیم می شود به این صورت که در هر کلاک rq را به سمت چپ شیفت داده و هر بار تفریق r-q را در سیگنال temp میرزد که طولش یکی بیشتر از طول سیگنال های نامبرده است بنابر این بیت سمت چپ temp معین می کند که حاصل تفریق انجام شده منفی است یا مثبت .

برای تقسیم کننده نیز همانند ضرب کننده یک سیگنال do تعریف شده که کار کنترل این ماژول را برعهده دارد و دقیقا همانند do در ضرب کننده عمل میکند.

## Logic_unit

در این ماژول از ماژول های and_maker , or_maker , shift_to_right , shift_to_left نمونه سازی می شود و به صورت موازی همه این چهار ماژول عملیات مربوط به خود را انجام می دهند و بعد توسط سیگنال control خروجی انتخاب مشود.

## And_maker

در این ماژول عملیات and منطقی به صورت کاملا ساده پیاده سازی شده است.پرچم ها نیز در این ماژول تعیین وضعیت می شوند.

## Or_maker

در این ماژول عملیات or منطقی به صورت کاملا ساده پیاده سازی شده است.پرچم ها نیز در این ماژول تعیین وضعیت می شوند.

## Shift_to_right

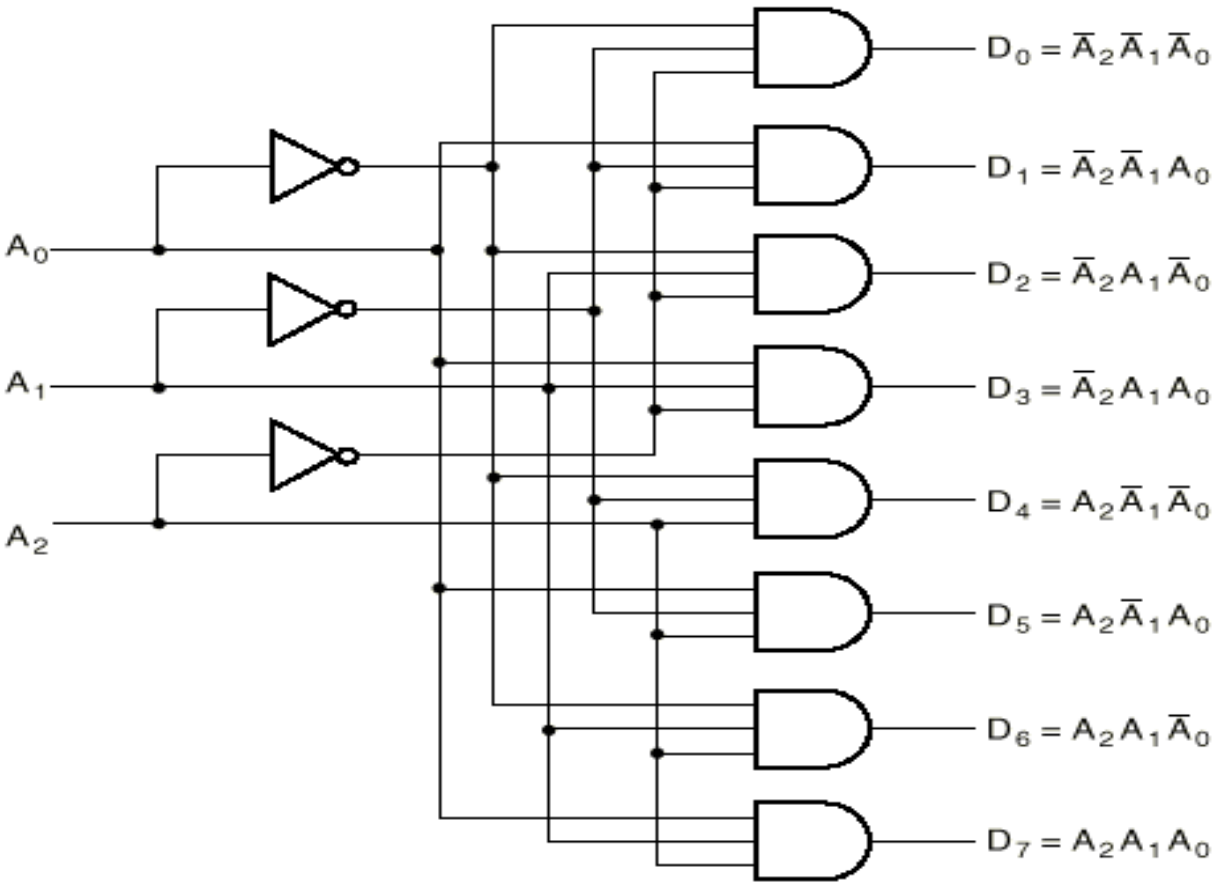
در این ماژول دو سیگنال میانی یکی برای ورودی و یکی برای خروجی تعیین شده است تا عملیات شیفت ممکن شود زیرا در غیر این صورت ممکن بود همزمان با نوشتن بر روی یک سیگنال از همان سیگنال بخوانیم و اطلاعات غلط بگیریم.

## Shift_to_left

همانند shift_to_right پیاده سازی می شود.

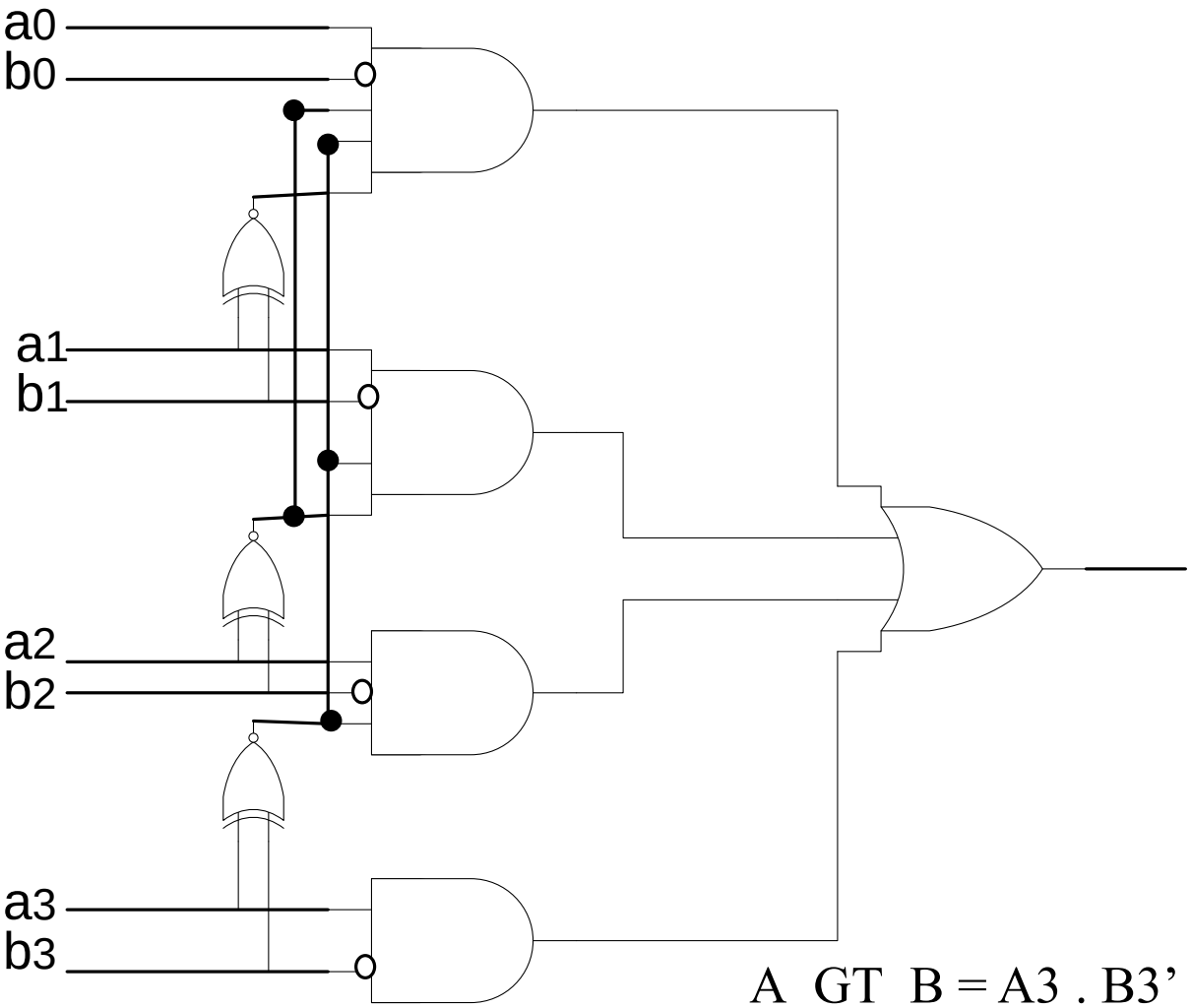
## Decoder

این ماژول از روی مدار زیر پیاده سازی شده است :




## Comprator

این ماژول از روی مدار زیر پیاده سازی شده است ( البته ماژول زیر یک مقایسه کننده برای چهار بیت است که دراین ماژول ما برای هشت بیت آن را پیاده سازی کرده ایم )




$$A\_GT\_B = A3 \cdot B3' + C3 \cdot A2 \cdot B2' + C3 \cdot C2 \cdot A1 \cdot B1' + C3 \cdot C2 \cdot C1 \cdot A0 \cdot B0'$$

# محاسبه تاخیر ها

## Adder

هر look ahead با تاخیر 7 گیت cout آن آماده می شود که با توجه به آبشاری بستن دو تا look ahead تاخیر جمع کننده می شود 14 گیت .

## Subtractor

هر look_ahead در اینجا با تاخیر 8 گیت آماده می شود که با توجه به آبشاری بستن آن ها تاخیر می شود 16گیت .

## Multiplier

در 7 کلاک عمل ضرب را انجام میدهد

## Divider

در 4 کلاک عمل ضرب را انجام میدهد

## Add_maker

2گیت تاخیر دارد

## Or_maker

2 گیت تاخیر دارد

## Shift_to_right

2 گیت تاخیر دارد

## Shift_to_left

2گیت تاخیر دارد.

# محاسبه مساحت ها

## Adder

هر look_ahead 47 گیت مساحت دارد پس adder دارای 94 + 8 گیت است که می شود 102 گیت

## Subtractor

هر look_ahead 53 گیت مساحت دارد پس دارای 53 * 4 برابر با 212 گیت است

## Multiplier

دارای مساحت 9 گیت است البته اگر adder به کار رفته در آن را در نظر نگیریم.

## Divider

10 گیت مساحت دارد . البته این ماژول در سطح گیت پیاده سازی نشده است.