

## 積體電路設計 Homework #4 Report

B06602035 生工三 李晴妍

### 一、Simulation

Minimum half-cycle time:3.59

### 二、Discussion

#### 1. Introduce your design

##### I. Multiplier

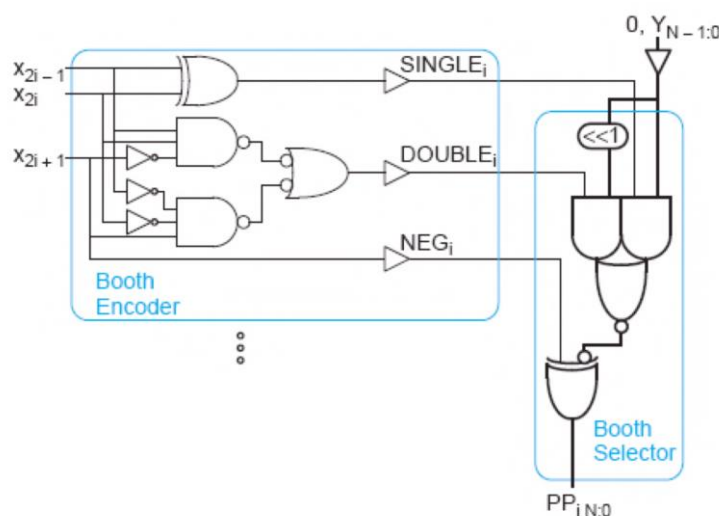
- module square(a,result,number); // calculate  $x \cdot x$
- module boothencoder(a,b,c,single,double,neg,number);
- module boothselector(y,single,double,neg,p,number);
- module add10(sum, a, b, number); // 10 bits ripple carry adder

乘法器用的是 Booth Encoding。Input 是 5 bits，multiplier 做三次後會產生三個 10 bits 的 partial product P0,P1,P2。

$\{x_1, x_0, 1'b0\} : P0$	$S_1$	$S_1$	$S_1$	$S_1$	●	●	●	●	●	●
$\{x_3, x_2, x_1\} : P1$	$S_2$	$S_2$	●	●	●	●	●	●	0	$S_1$
$\{1'b0, x_4, x_3\} : P2$	●	●	●	●	●	●	●	0	$S_2$	0

$\{x_1, x_0, 1'b0\}$ 和 $\{1'b0, x_4, x_3\}$ 因為有位是 0 bit，在 booth encoder 時可以簡化。再用 Wallace tree 把 partial product 加起來。

一開始是用 carry safe adder 相加得平方結果，但發現這樣 area 比起 ripple carry adder 要大，但時間跟 cycle 並不會減少很多，所以最後改用 ripple carry adder。而為了避免 overflow，所以 partial product 選用 10 bits。



## II. Adder

$$D = BC \cos \angle A = \frac{B^2 + C^2 - A^2}{2}$$

- module hadd10(a, b, c, sum, number); // 10 bits half adder
- module add11(sum, a, b, number); // 11 bits ripple carry adder

這裡用的是 2's complement。得到  $A^2$ 、 $B^2$ 、 $C^2$  後，要先 invert  $A^2$  的每個 bits，再用 half adder 加上 1'b1，就會得到  $-A^2$ 。

因為相加過後有些值會超過 10 bits 2's complement 的範圍(-512～511)，ex:  $30^2 + 30^2 = 900$ ，所以在這邊先轉成 11 bits 再用 ripple carry adder 將  $-A^2$ 、 $B^2$ 、 $C^2$  相加得到結果 final[10:0]。

## III. Shift

取 final[10:1]，無條件捨棄。

## IV. Register

- module reg10(Q, DD, clk, reset, reg3\_num);

用 FD2 做一個 10 bits register，為了控制讀取的時間有讓 valid\_out 與其中一個 multiplier 的結果作 or。

## 2. How do you cut your pipeline?

我把 pipeline 放在做完 multiplier 之後。也有試著多增加一個 pipeline 在做平方相加的地方，但 area 增加很多 cycle 卻沒辦法降多少，所以捨棄。

Pipeline 加上後會讓 D 讀取的 clk 改變，那時候找了很久以為是後面加法器的問題。最後有讓 valid\_out 跟乘法器的 output 做 or 去拉回 clk。

## 3. How do you improve your critical path and the number of transistors?

像是做 adder 的時候，一開始用的其實是 CSA，但後來去算裡面實際 FA 的 area，再去跟 ripple carry adder 的 area 比較其實差蠻多的，因為多增加一個 FA 就要多 26 個面積，而 adder 又會用在很多地方就會不斷增加，所以最後採用的是後者。

除此之外因為已經考慮過 overflow 的問題，最後的進位其實都可以捨去，這樣就不用用 FA 可以改用 XOR 去計算，雖然這樣 cycle 會降不太下來但 area 可以少很多。

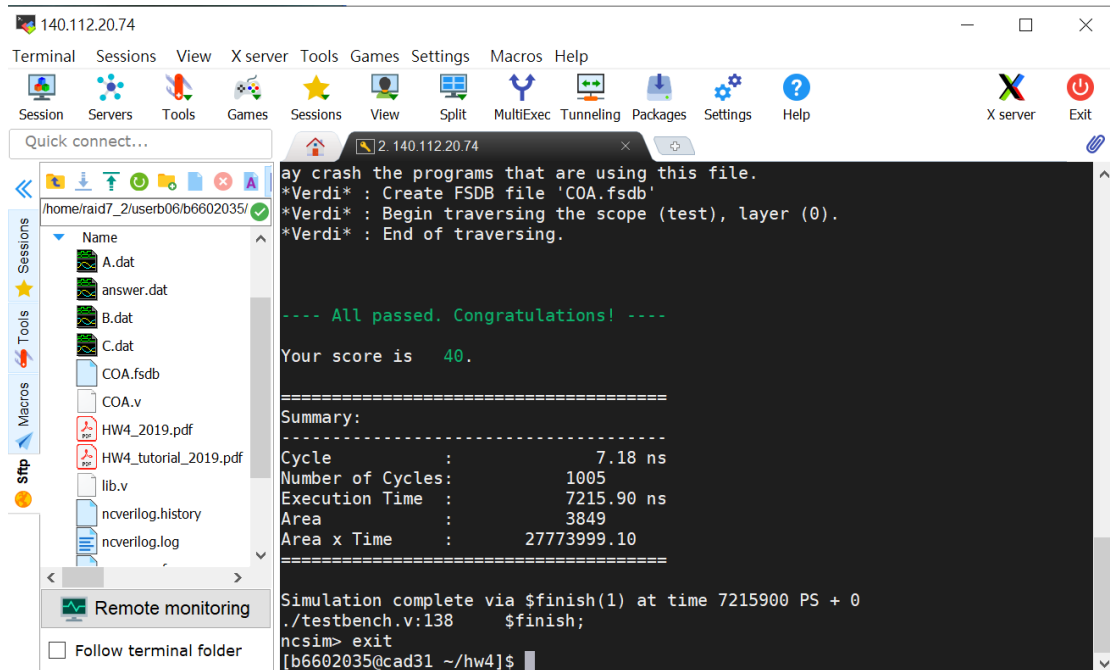
而為了減少 critical path，在做完乘法後有加入 pipeline，其他地方有試過再加但沒有甚麼好處。

## 4. How do you trade off between area and speed?

我會先大概算 area 會增加多少，主要是考慮 FA 的 area 就好。然後看增加的比例去估 half cycle 降到多少比例以下才有利。如果跑 testbench 失敗的話就不會改算法。

5. Compare with other architectures you have designed(if any)

基本上沒太大改變，就是用 Booth algorithm + Wallace tree 做的乘法器。加法器從 CSA 換成 ripple carry adder。後來有在乘法器做完後加上 pipeline，最後用 flip-flop 跟 valid\_out 去算 D 讀取的時間。



The screenshot shows a terminal window with a file explorer on the left. The file explorer lists files like A.dat, answer.dat, B.dat, C.dat, COA.fsdb, COA.v, HW4\_2019.pdf, HW4\_tutorial\_2019.pdf, lib.v, nverilog.history, and nverilog.log. The terminal window displays the following text:

```
ay crash the programs that are using this file.
*Verdi* : Create FSDB file 'COA.fsdb'
*Verdi* : Begin traversing the scope (test), layer (0).
*Verdi* : End of traversing.

---- All passed. Congratulations! ----

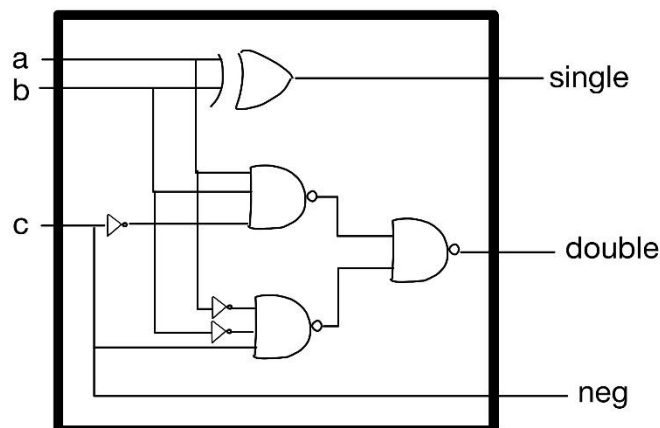
Your score is 40.

Summary:
-----
Cycle       :          7.18 ns
Number of Cycles:      1005
Execution Time :      7215.90 ns
Area        :        3849
Area x Time  :      27773999.10
-----

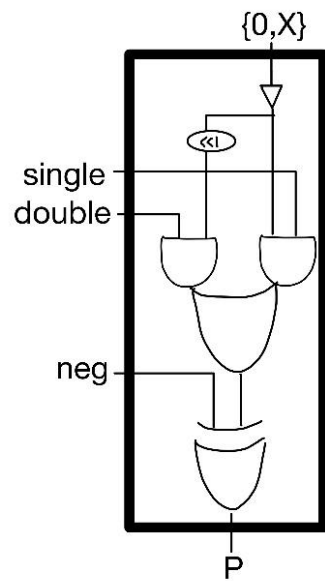
Simulation complete via $finish(1) at time 7215900 PS + 0
./testbench.v:138 $finish;
ncsim> exit
[b6602035@cad31 ~/hw4]$
```

### 三、Circuit diagram

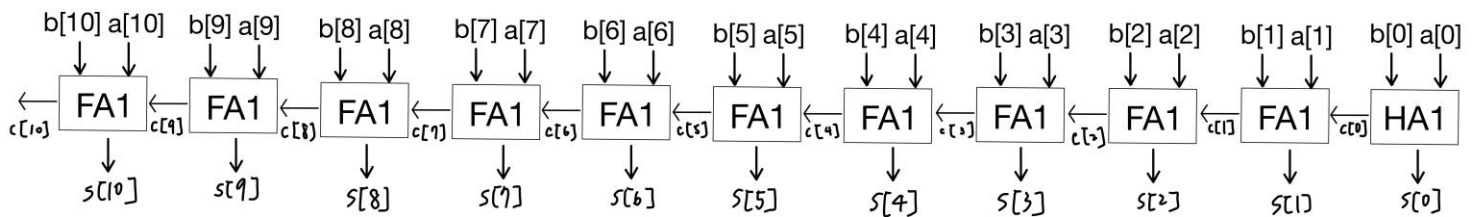
#### ■ Booth Encoder



#### ■ Booth Selector



#### ■ Ripple Carry Adder(11 bits)



#### ■ Critical path

