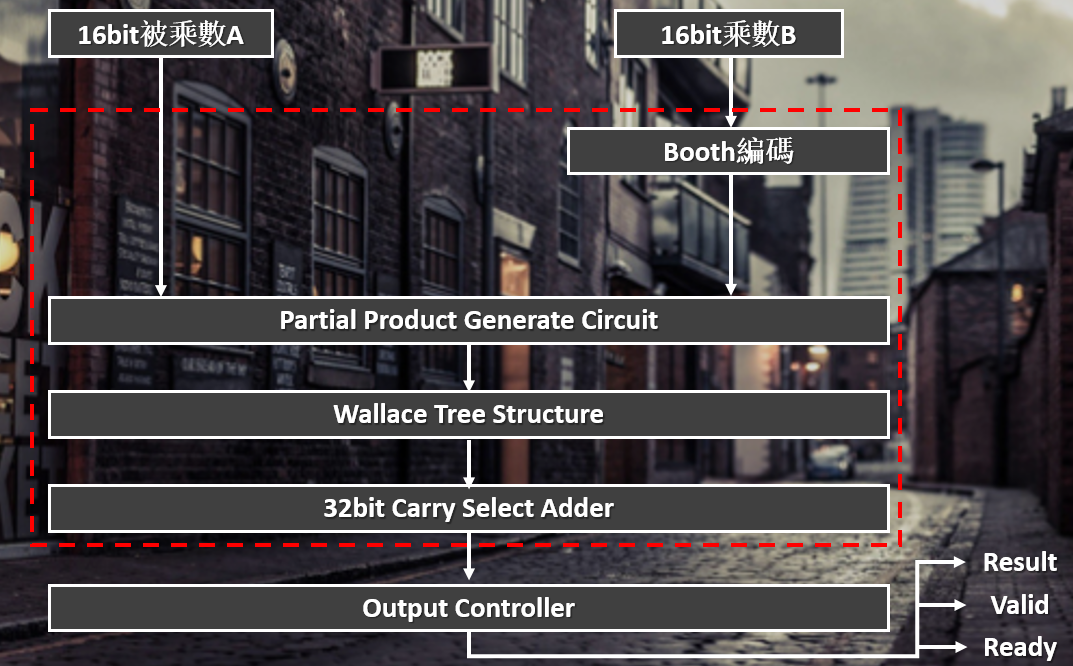
**106-2實用數位系統設計期末專題報告**

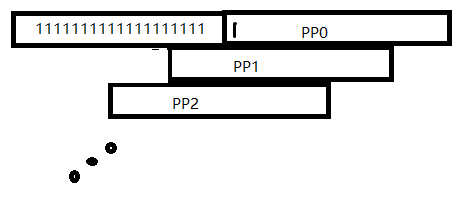
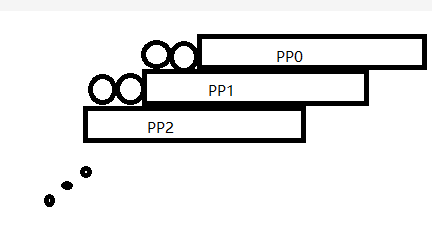
**<16-bit 有號數乘法器之設計與實現>**

**組別: 03**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Member | | | | |
|  | **學號** | **姓名** | **負責項目** | **貢獻百分比** | **簽名** |
| 1 | B035020026 | 李冠霖 | Wallace架構  報告 | 50% |  |
| 2 | B032040036 | 廖偉驊 | SRCSA加法器  Booth編碼 | 50% |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |

架構說明

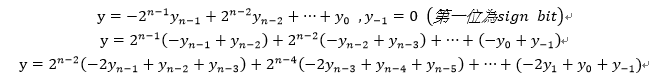
圖一、乘法器架構

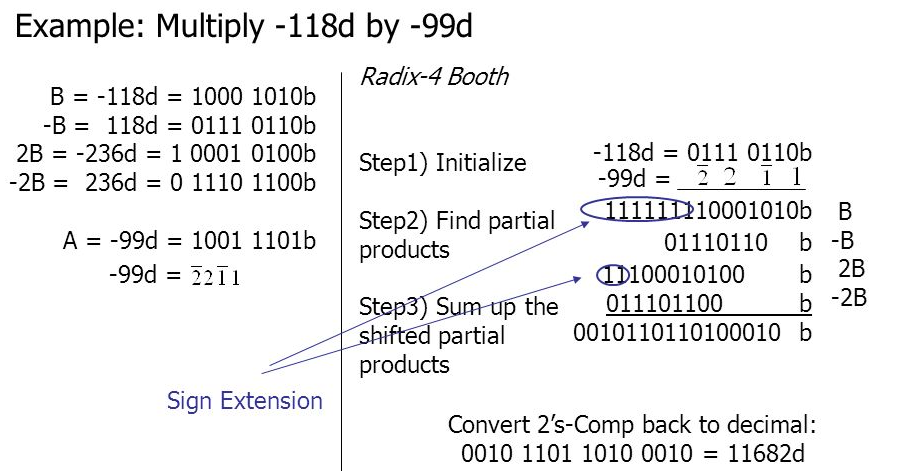
此架構可接受有號數計算，原先的架構是將有號數事先提出來存放，但為了節省每個部分乘積(以下簡稱PP)都要以32bit下去計算浪費許多資源，以下圖二為例，產生的PP為17bit，但進行PP相加的時候必須採用32bit去計算，高位元數字完全相同而為了省去這些重複的計算，我們將每個PP的Sign事先存起來以利在Wallace Tree計算完畢後將Sign放回。

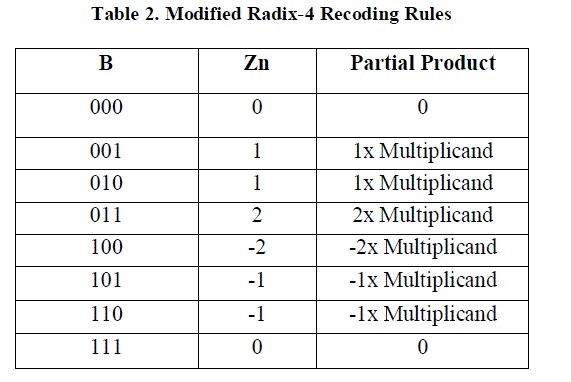
圖二、PP有號數處理

本架構採用Booth的Radix-4編碼方式，每隔兩個bit編碼一次，重疊(overlay)一個bit，以利快算計算出PP並減少PP的數量，可以將PP數量減少為原本的(N+1)/2。Booth編碼共有三種模式，Radix-2、Radix-4、Radix-8，2過於簡單，8對16bit的乘法來說過於複雜沒有達到太多效益，因此選擇了Radix-4實作。詳細過程如下圖三、圖四。圖三中第一列為原始的乘法相加流程，經過第三式的改寫後可以減少相加的次數。

圖三、Radix-4的PP編碼公式(

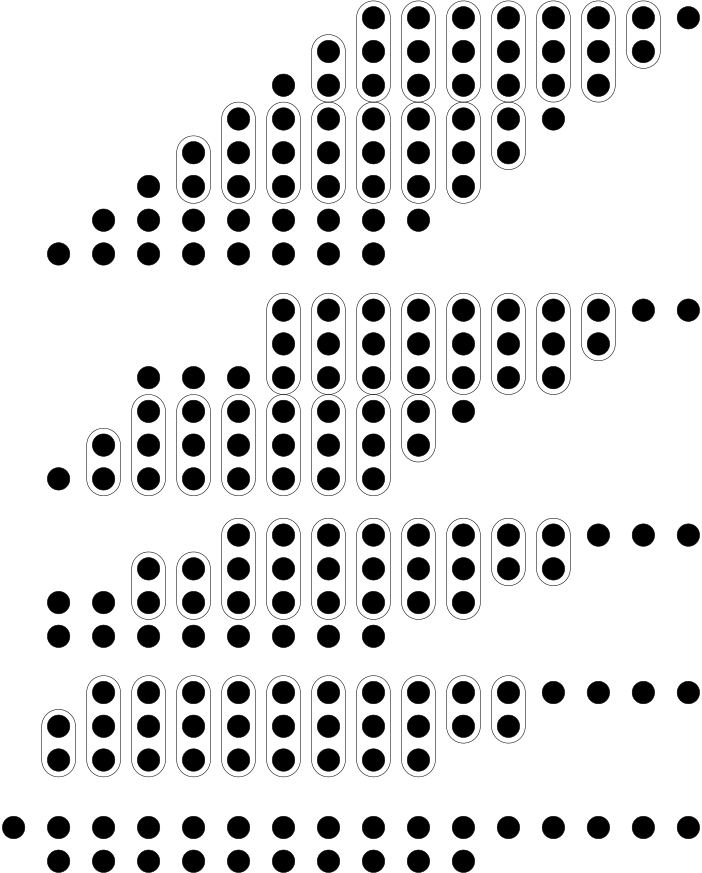




圖四、Radix-4編碼過程

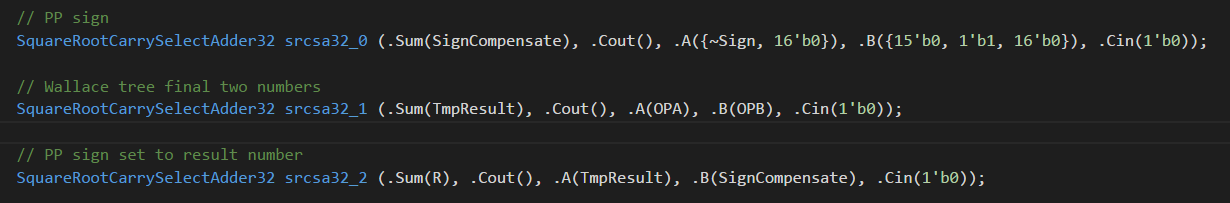
圖五、編碼表

圖四為Radix-4的編碼過程，由左至右每三個bit進行一次編碼，編碼結果參見圖五，以圖四的乘數10011101為例，編碼後為-2 2 -1 1，因此簡化後的PP為四個，而16bit簡化後為8個。

有了PP之後再來就是使用壓縮樹Wallace tree把8個PP壓縮成兩個數字相加，如圖六所示。

圖六、Wallace Tree壓縮樹

因為此結構極為複雜，人工畫圖容易錯誤，因此我們撰寫了一個python小程式自動化的產生出了Wallace Tree的結構代碼，節省了人工輸入的時間，且確保了變數命名不會有錯誤。(這個問題一直導致合成失敗，python代碼可參見壓縮檔內的verilog.py)

最後使用三個加法器(Squre Root Carry Select Adder)將結果運算出來，如圖七。

圖七、SRSCA加法器

第一個加法器用於把PP擴展為32bit，第二個加法器將壓縮後的兩個數字相加，最後一個就是把之前拿出去的Sign放回去，輸出最終運算結果。

最後為了配合ready和valid訊號，我們撰寫了一個控制器，利用cycle數控制輸出結果，以確保可以正確地配合testbench。(可參見modules.v的Outputer)因為已知我們的乘法器最慢需要8個cycle輸出結果，因此控制器裡有一個計數器負責計算cycle數量，在第1個cycle時ready=1之後皆為0，而在第8個cycle時valid=1，其餘皆為0，就算乘法器提前算完也不會提早輸出結果，計算完的數值會被存放在buffer中。

驗證流程

在合成過程中使用到了以下指令：

合成前：

**current\_design multiplier** (指定Top module，預設為寫在最上面的module)

**set\_fix\_multiple\_port\_nets -all -buffer\_constants** (解決port同時in-out)

**remove\_unconnected\_ports -blast\_buses [get\_cells \* -hier]** (移除空的腳位)

**create\_clock -name "Clk" -period 1.66 -waveform { 0.83 1.66 } { Clk }** (指定clock)

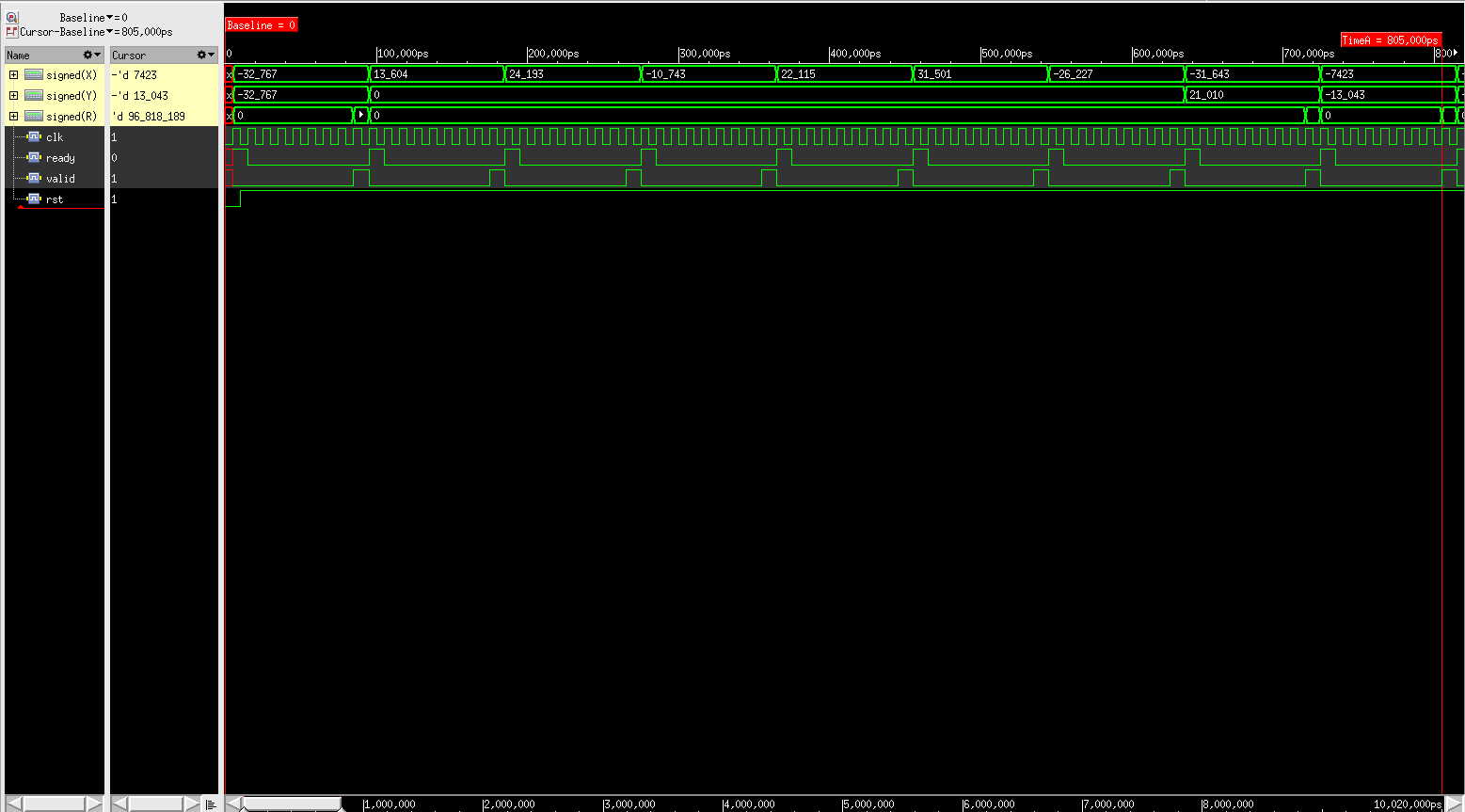
合成後：

**change\_names -rules verilog –hierarchy** (解決 Verilog 'assign' or 'tran' statements are written out. (VO-4))

**write\_sdf -version 1.0 -context verilog multiplier\_syn.sdf** (將時間資訊寫入檔案)

**`include "/data/software/PROCESS/CBDK\_IC\_Contest\_v2.1/Verilog/tsmc13.v"** (引入製程到testbench中)

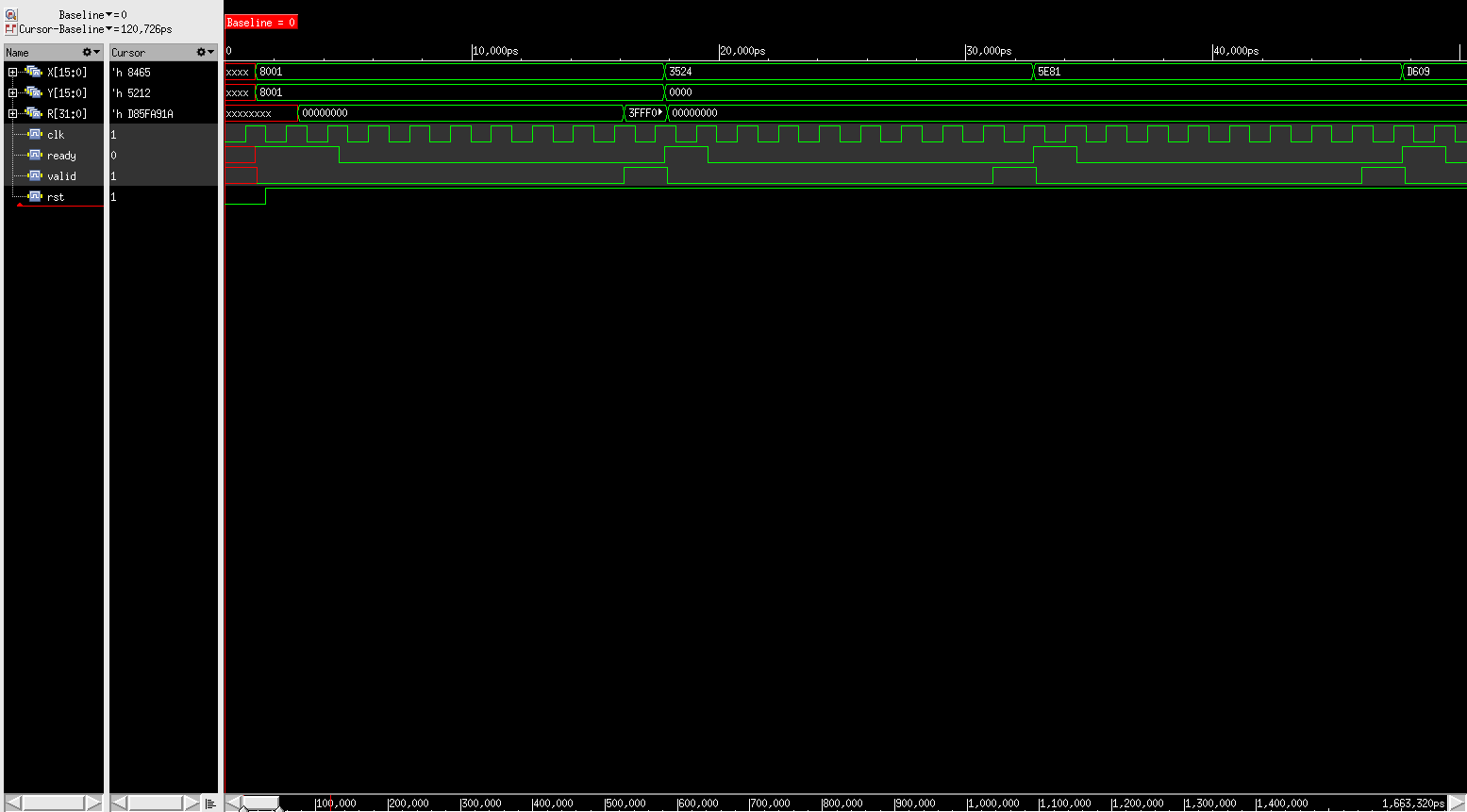
**initial $sdf\_annotate("multiplier\_syn.sdf", multiplier);** (引入時間資訊到testbench中)



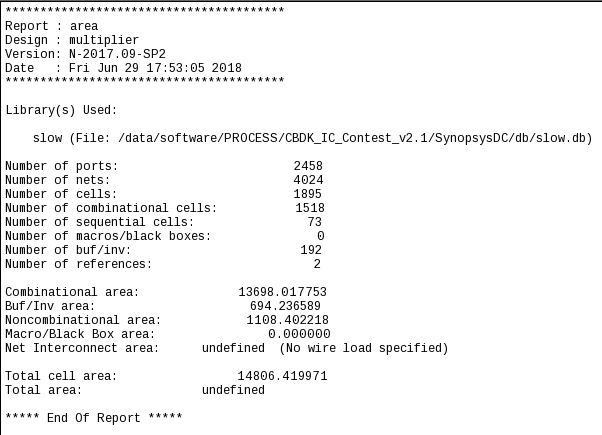
圖八、合成前波形圖

驗證方法採用(Random)隨機生成數字，並規定上限為1000個進行驗證，另外加上了本次教授所提供的參考測資。

波形圖中X代表被乘數，Y代表乘數，結果為R，從波形圖可得知在valid=1的結果皆為正確(人工確認前10個以及教授的給定測資，假定全部皆為正確)。

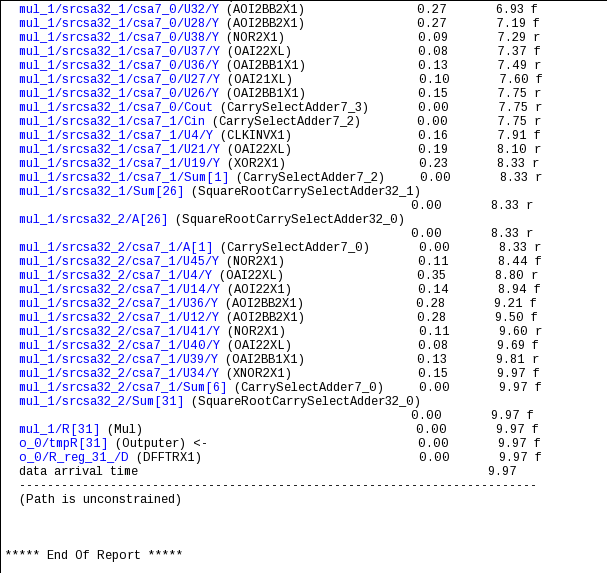
圖九、合成後波形圖(時脈602MHz，週期1.66ns)

我們將局部的結果放大可得知，因為加入了clk並且以最高速度運行(實測後的結果)，第一次結果的輸出遠比合成前所需要的時間縮短了將近1/6的時間(原始為100MHz)，測資不變，而得出的結果也和合成前相同，因此推斷合成後的結果為正確的。

電路面積：

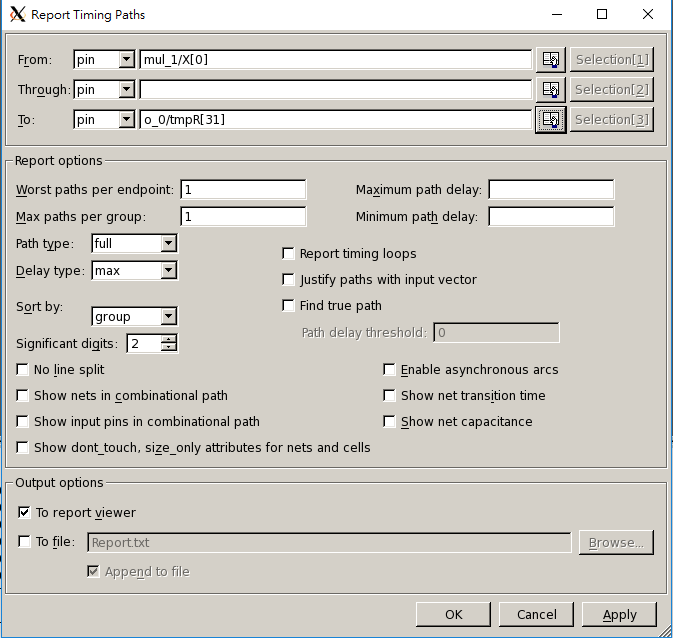
圖十、電路面積

大約為

乘法運算時間：

圖十一、乘法運算時間(部分)

上圖十一為乘法運算時間的部分截圖，詳情可見delay.txt，我們利用指定port的方式讓軟體計算出乘法的delay，設定如右圖十二。共需要9.97ns



圖十二、計算乘法運算時間設定

心得討論

廖偉驊：

這次期末專題，我做的部分是Square root carry select adder以及Booth coding。在決定做Booth乘法器之後，我們首先去了解Booth乘法器的運作原理，網路上有各種運作原理，Booth編碼有Radix-2、Radix-4等，最終的加法器可以使用各種加法器，當然它們的運算時間與面積各有好壞。而Booth編碼完後的不等數量的數字可以用各種Tree來進行相加，例如Dadda Tree或是Wallace Tree，當然也有好壞。在研究相關原理之後，我們無意間在被微軟收購的GitHub上找到了使用Booth Radix-2、Wallace Tree、Square root carry select adder的開放原始碼，我們當然不能直接使用，而是參考後改用Radix-4編碼下去做。Radix-2編碼後會產生16個16-bit的數字，之後用Wallace Tree進行平行相加，需花費6個單位相加時間。我們使用的Radix-4會產生8個17-bit的數字，為什麼是17-bit呢？因為Booth編碼表上有兩倍的被乘數，如果使用了16-bit兩倍後肯定是不夠存的。在第一次實作的時候並沒有發現，而直接使用16-bit去實作，並手動設計Wallace Tree後下去跑testbench，結果當然是錯的。後來經過10分鐘的思考，回顧整個流程，才發現需要使用17-bit去存編碼後的結果。第二次實作時，完全不想在手動規劃Wallace Tree還得打一堆會忘記的變數，因此我們直接編寫Python程序去自動產生Wallace Tree壓縮過程的Verilog code，才能讓第二次實作如此快速。這次整個編寫時間花費了大約16個小時包含成功合成與模擬波型，比預估的還慢，我們會努力增加自己的能力的。

李冠霖：

在這次的專題中，我們起初對於乘法器的架構一知半解，也不知道從哪裡著手，因此上網找了許多現有的架構進行比較，挑選了效能不差也容易實作的，最後就決定使用booth編碼搭配Wallace Tree的架構，而在最後的加法器部分，起初考慮到Carry Lookahead的面積過大，因此找了另一個折衷方案SRCSA，不過因為是沒有看過的架構，因此考慮到實作上可能會有問題，我們並行開發了兩個加法器，最後再根據組合後的效能決定最終要採用的架構。而經過比較我們還是決定使用新的架構Squre Root Carry Select Adder，而Wallace Tree因為極為複雜，加上我們第一次實作的時候發生了點人為疏失導致要整個重寫，而第二次為了避免這個問題，於是撰寫了python來根據理論自動產生code。因為這次專案比較大，合成的時候也碰到了不少問題，所幸有CSDN論壇，和國外的Verilog專門討論版，解決了一些古怪的問題，諸如報告中提到的部分。這堂課幾乎算是從頭開始，這次的專案能順利完成可以說是運氣也很好。