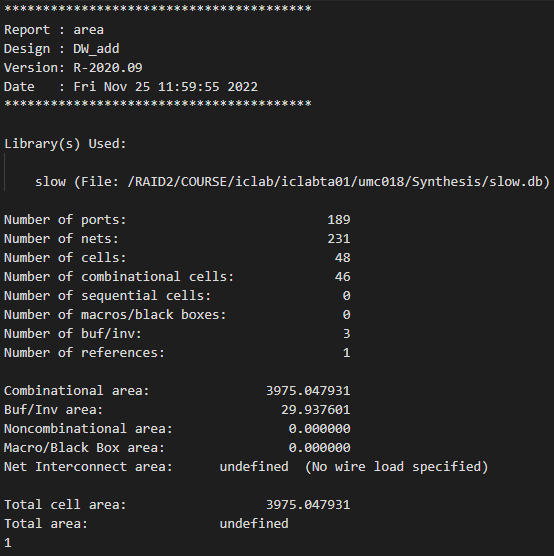
2022 NYCU EE VLSI Lab Report

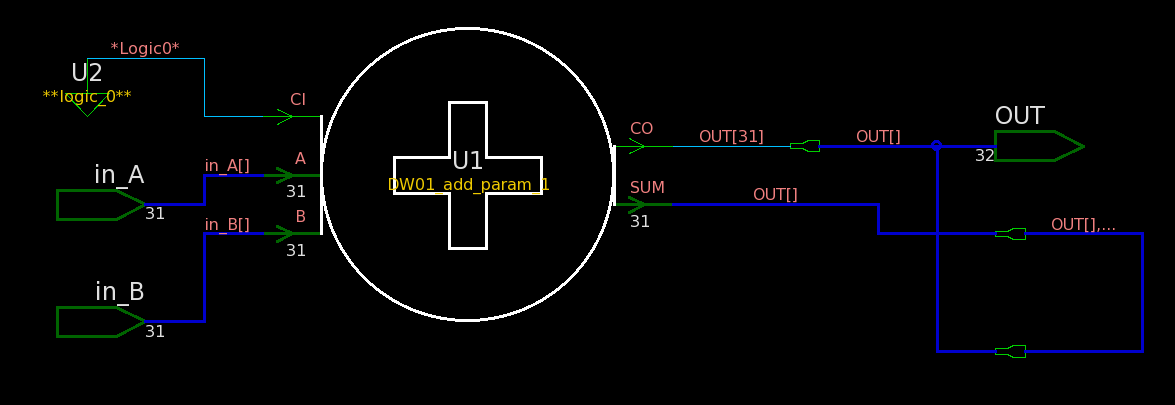
Lab04 Logic synthesis

Student ID: 109611070 Name: 郭家均 Date: 2022/11/25

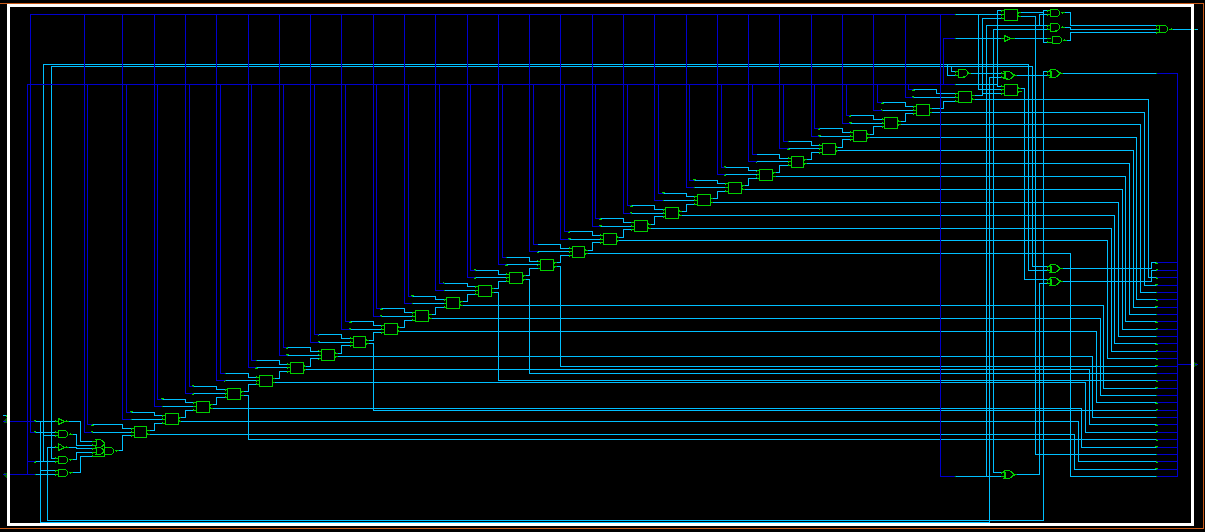
1. **Adder**
2. **31bits carry-ripple adder**
3. **the smallest clock period: 7.06**
4. **area report (picture)**



1. **pre-synthesis schematic**



1. **post-synthesis schematic**

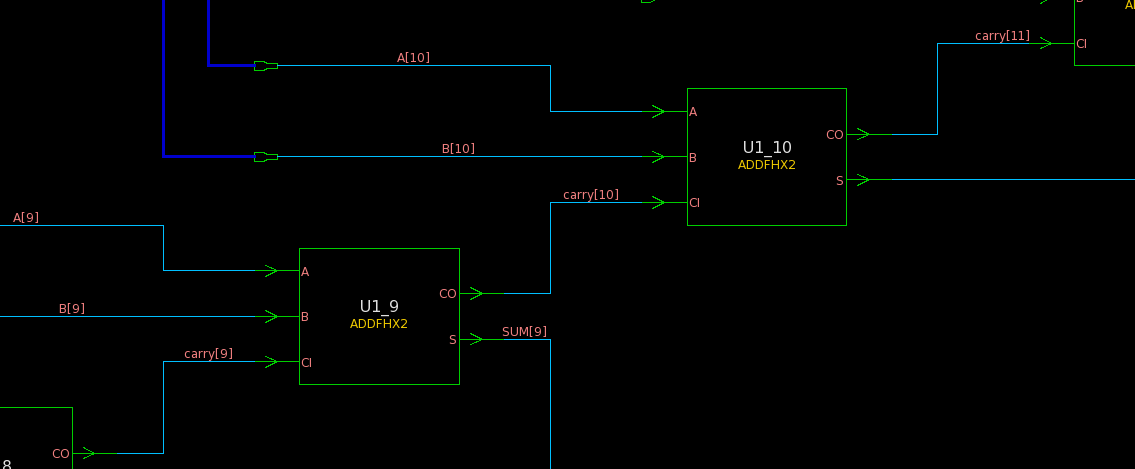


1. **Please explain the difference between pre-synthesis and post-synthesis**

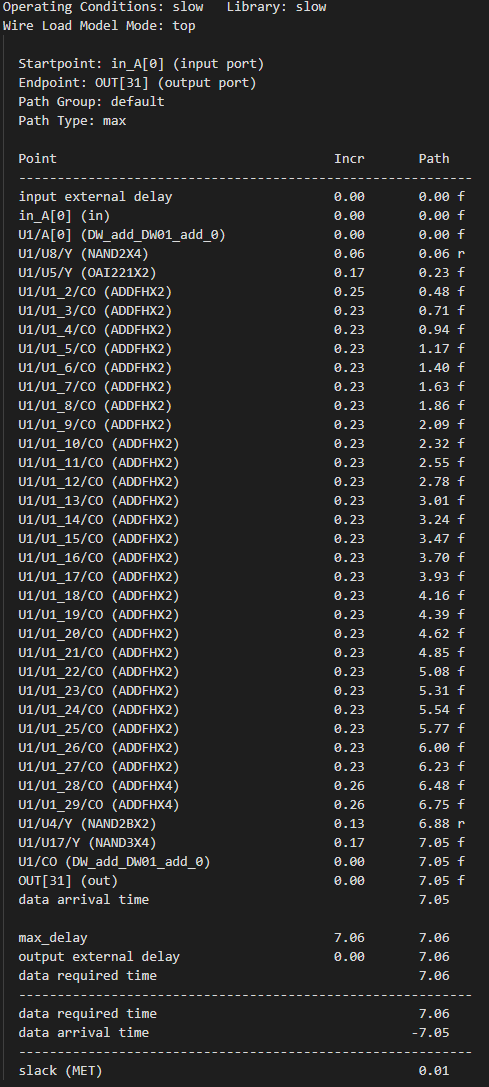
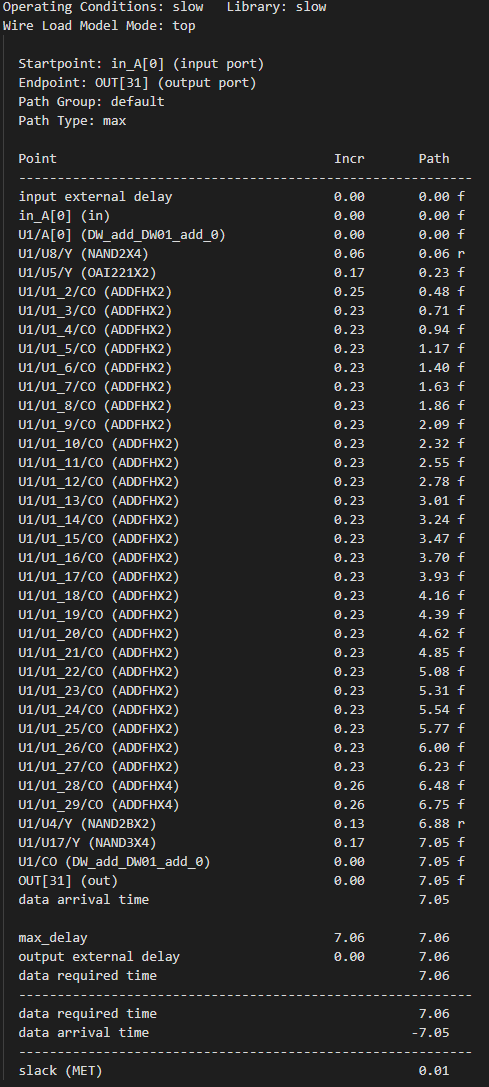
Pre-synthesis在電路設計過程能verify自己的電路在functionality的部 分是正確的。可想成在instance中有一個叫DW01\_add的block，而 我可以直接在HDL code中呼叫。Pre-synthesis的結果跟pattern對比 看結果是否正確。

Post-synthesis中我們可以修改adder block的種類(RCA, CLA...)並且 02模擬過程中，可以將每個instance中的架構(gate-level)實現。看到 schematic就可以看到adder的完整架構已經由stdcell拼接好，且最 後synthesis是考慮所有stdcell的delay、area。

rpl adder的架構很簡單。以一個FA為單位，最初的CI = 0。每級 carry-out推到下一級的carry-in。以下放大看schematic的部分。可以 看到U1\_9的carry[10]接到U1\_10的CI port。



1. **timing report**

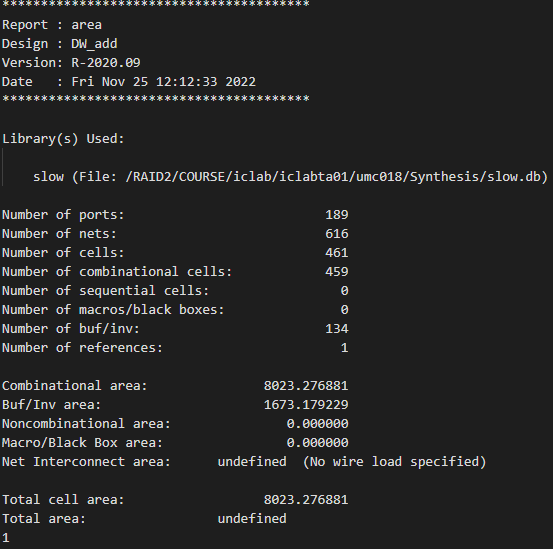


1. **Describe and explain the critical path**

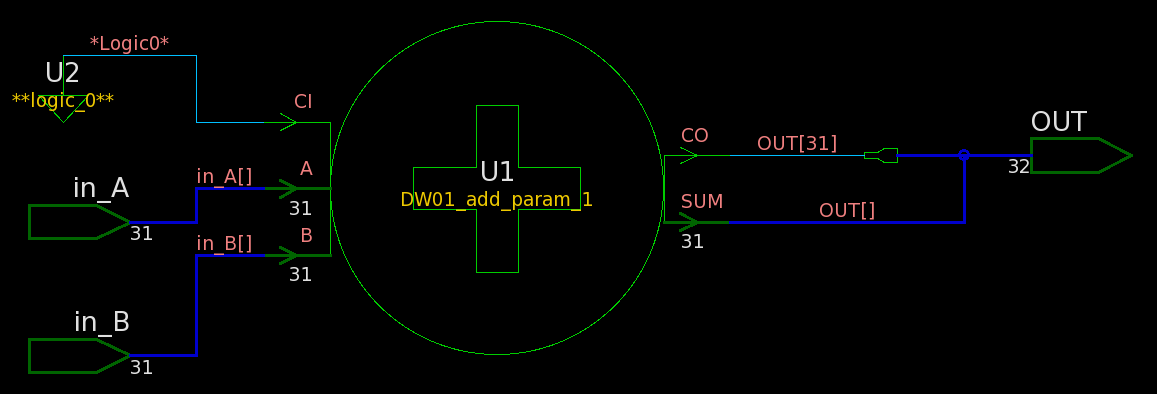
根據timing report，可以看到critical path經過每一個block的CO port。 且最後一級為OUT[31]。從pre-synthesis schematic可以看出OUT[31] 為最終的COUT port。

可想而知，critical path從第一級的input推到下一級FA的CI port， 一路串接到最後的carry-out。也可以得知最後一級FA的output中， SUM[30]的delay比COUT還小 (NAND2 + NAND3 > XOR)。

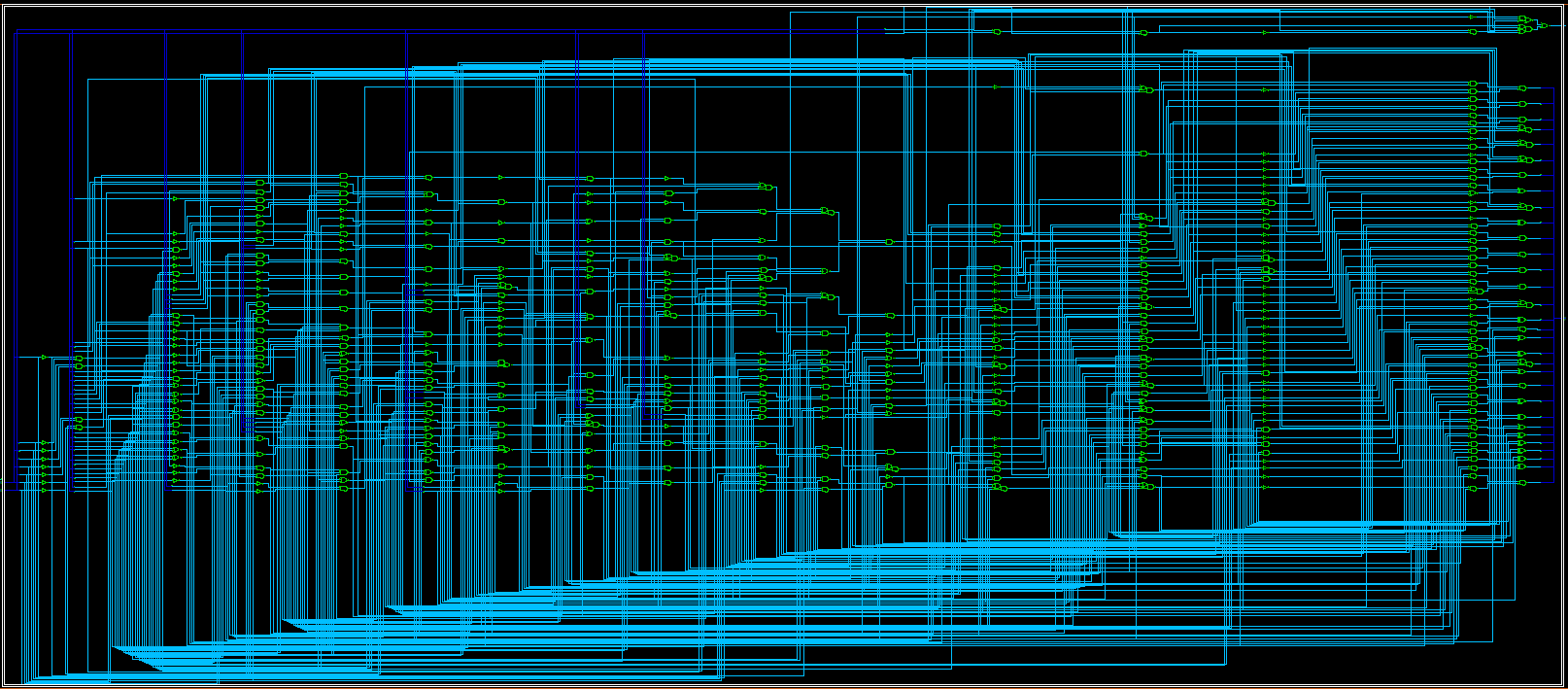
1. **31 bits carry-lookahead adder**
2. **the smallest clock period: 1.44**
3. **area report**



1. **pre-synthesis schematic (picture)**



1. **post-synthesis schematic (picture)**



1. **Please explain the difference between pre-synthesis and post-synthesis**

Pre-synthesis、Post-synthesis的差別和**i. 5.**雷同。

cla架構 :

從post-synthesis電路可以看到整個area大了兩倍以上。為了加速先 前rpl的critical path，每一級的SUM不再單純由前一級的carry-out 提供。每一級的CI可以由以下的判斷式提前得知

CI (ith) = (AiBi = 1) +

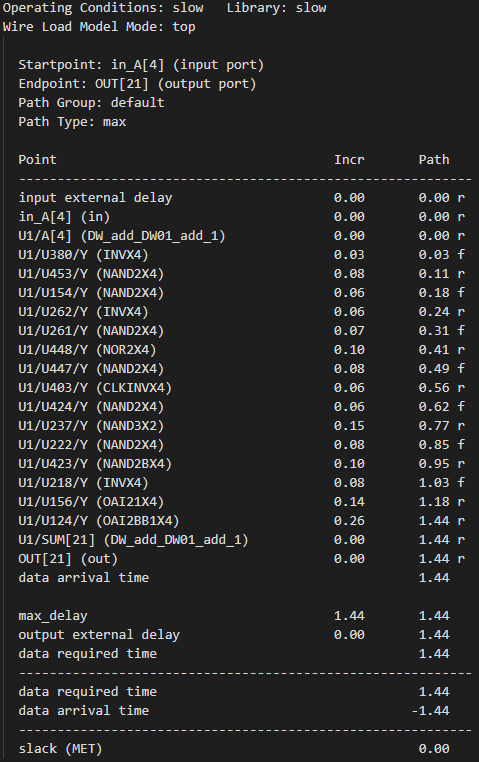
(Ai+Bi = 1) & (carry(i-1) = 1) +

(Ai+Bi = 1) & (A(i-1)+B(i-1) = 1) & (carry(i-2) = 1) +.........

越後級的等式會越長，代表其propagation也會過大，不切實際，因 此推估應該是使用n bit為一組的CLA架構 + tree架構。如此time complexity為O(log n)比較符合和rpl adder之間時間的比較。更詳細 的原因寫在critical path的analysis中。

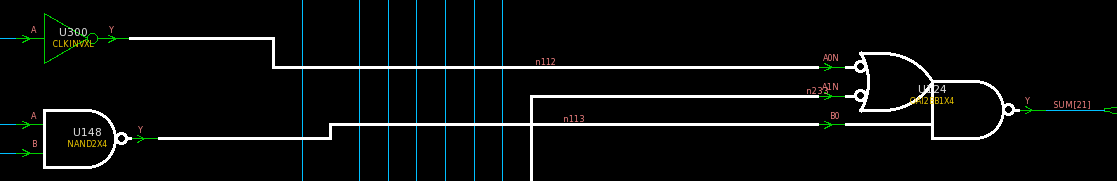
只要其中一行的logic = 1，CI就為1。如此加快carry-in的速度，因 此最後可以發現fastest clock整個快了4、5倍左右。

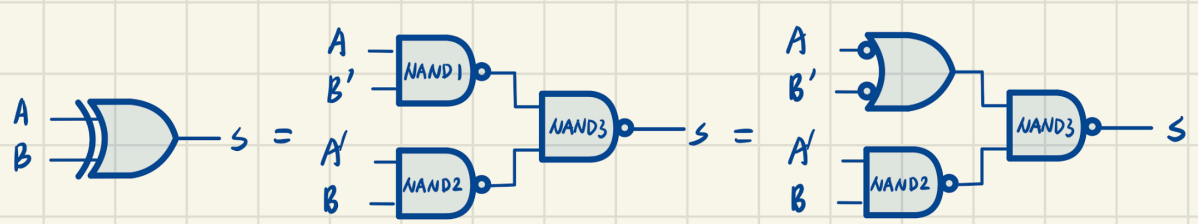
1. **timing report (picture)**



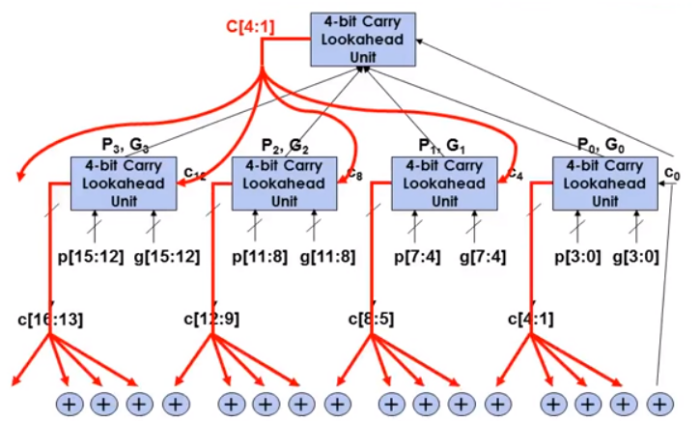
1. **Describe and explain the critical path**

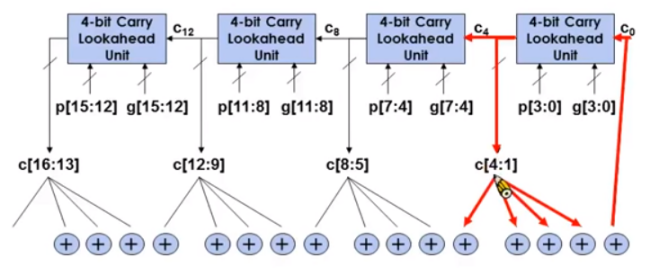
根據timing report，發現整條critical path是從input端 (A[4])經過了 15級的gate最後output為SUM[21]。觀察SUM[21]output最後兩級 簡化後可以發現就是一個簡單的XOR的邏輯如下圖所示。





又因為critical path的起點在A[4]，可以推得主要造成delay的是 A[21] + B[21] + CI(21)中carry-in的部分。由此可以討論幾個可能的 CLA架構。一個是最基礎的每級CI都由一塊combinational logic運 算。但後級的CI會因為式子過長所以propagation太長，不切實際。 第二種架構為採n-bit唯一單元的CLA，一次計算完n-bit的CI，n+1 級的CI由n COUT提供。像rpl adder和CLA組合如下圖左。但此架 構也不是很合理，照理來說若為此架構，critical path會位於最後一級 的SUM，但結果不是如此，因此猜想第三種架構。 第三種架構是第二種的延伸，將CI的邏輯做成tree的架構，time complexity為O(log n)，類似右下的圖。如此critical path的終點就可 以位在SUM[21]而不會有問題。且根據公式可以算31約為 2^5。因 此猜測CLA以兩個bit為單位，做了五層的tree。





1. **Compare the pros and cons of architectures of carry ripple adder and carry-lookahead adder.**

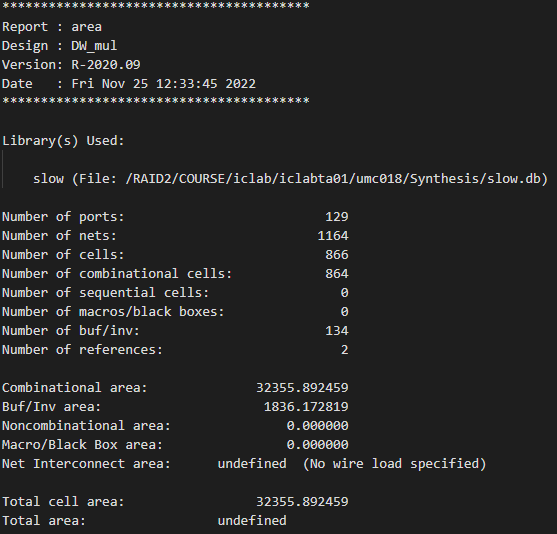
Carry ripple adder Pros :面積小，結構方便切pipeline去做加速。

Cons : Critical path很長，不切pipeline會latch住CLK

Carry look ahead Pros : 速度大大優於RCA。有多種架構、活性高。

Cons : 面積相較RCA大許多，成本高。

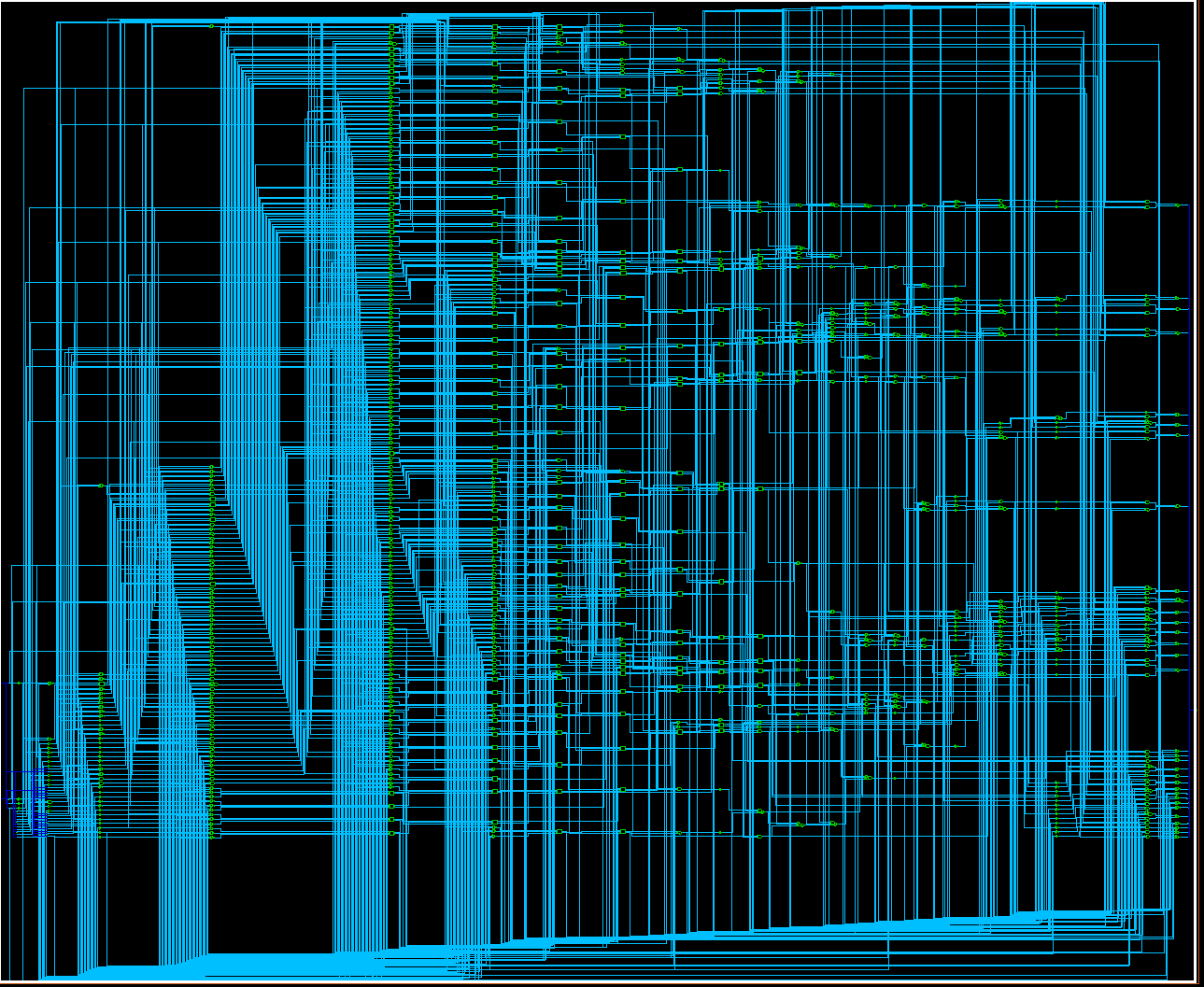
1. **Multiplier**
2. **31bits and-multiplier**
3. **the smallest clock period: 3.40**
4. **area report (picture)**



1. **pre-synthesis schematic (picture)**



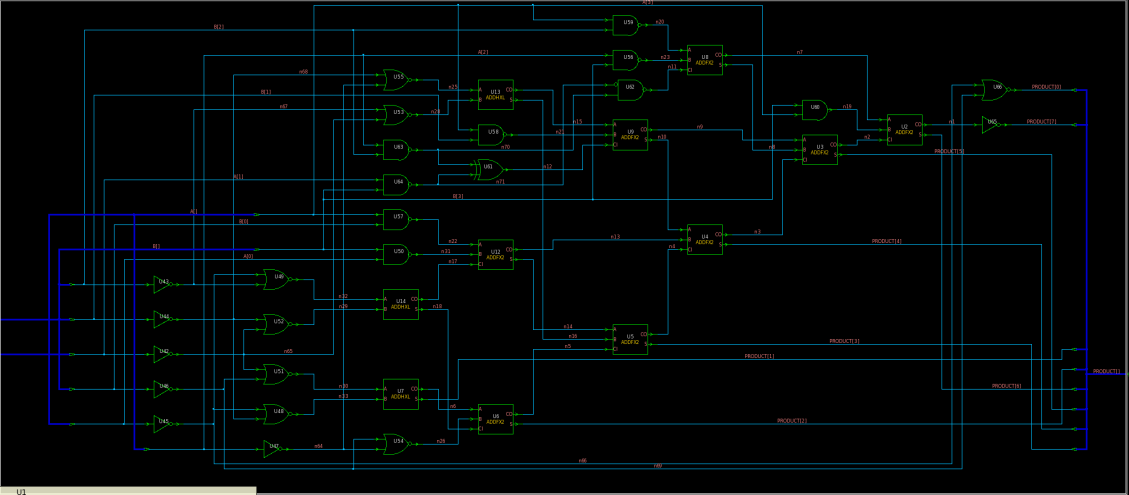
1. **post-synthesis schematic (picture)**



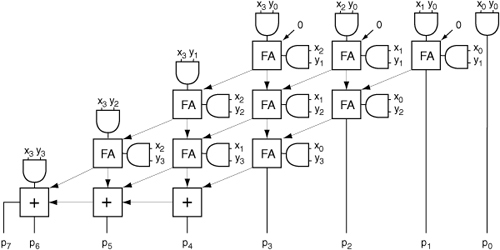
1. **Please explain the difference between pre-synthesis and post-synthesis**

Pre-synthesis看不出電路的架構，只是判斷function正確。

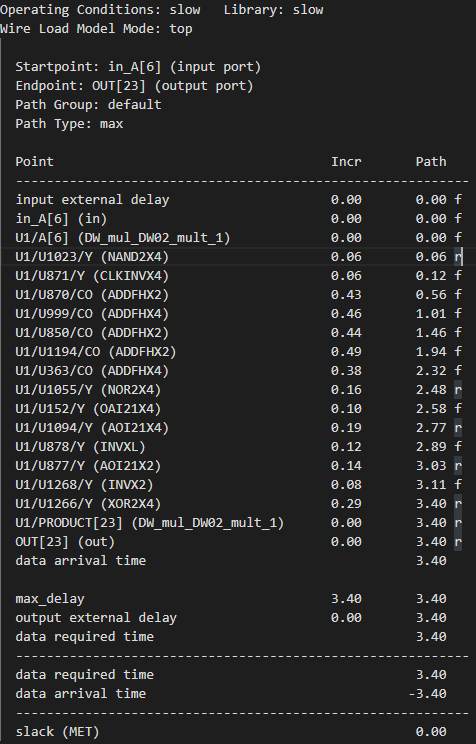
Post-synthesis架構 :

And multiplier的架構上很直觀，和直式乘法的概念很像。我有叫出 4-bit\*4-bit的And multiplier觀察其架構如下圖 :

可以發現每個對應的A B input都接上了AND gate (INV + NOR)。算 出兩個bit相乘的數值後，由一些HA、FA去計算product sum和每 一級的進位，整個架構初步推估類似下圖(完整架構分析看7.):

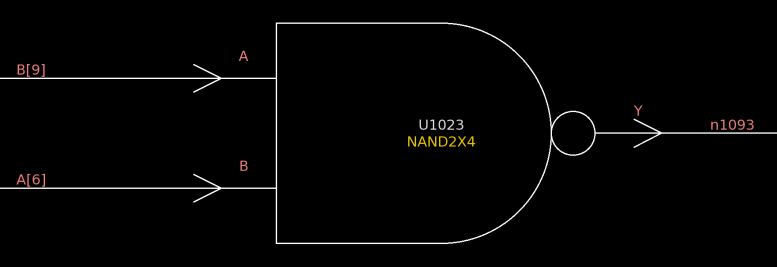


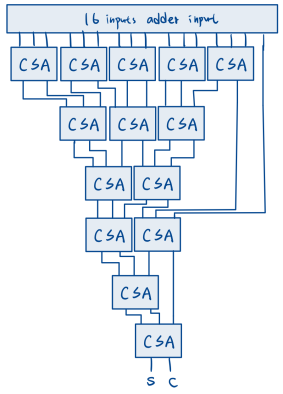
1. **timing report (picture)**

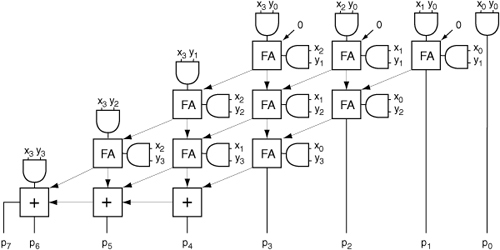


1. **Describe and explain the critical path**

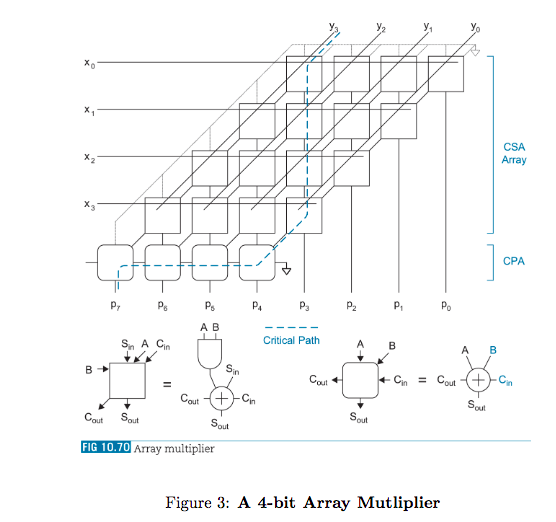
根據5.分析的架構，可得知critical path必經的路段是中間partial product重疊最多的部分，可以看成一個multiple input adder。 我們觀察timing report critical path的input可以看到是發生在NAND U1023且兩個輸入為A[6]、B[9]如下圖

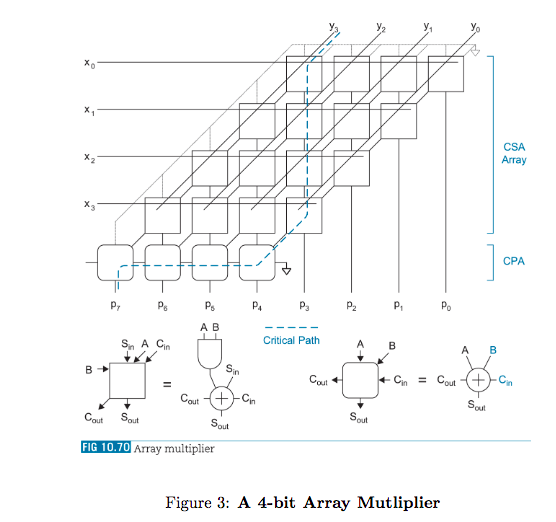


可以推得是最中間16 input multiple adder的其中一個input，驗證先 前的架構。不過終點可以看到在product[23]，不是product[31]，而且 critical path經過的HA、FA數遠遠不及16個，因此可以推測最下方 的FA應該不是單純的RCA架構，如下圖(用少bit架構代替表示)。 在16\*16的multiplier中，紅框部分為類似16-bit RCA的架構，藍+ 紅框為critical path。

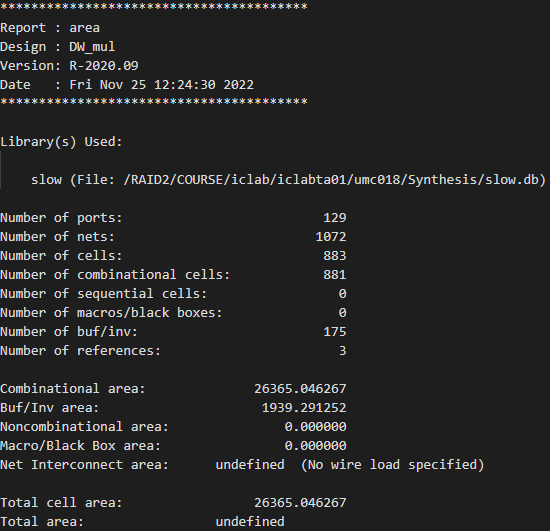
  
 上圖藍框的部分可以改為CSA的作法，如上右圖。可以發現用CSA 的作法，藍框的propagation delay最多過六級的adder。

至於紅框部分，可以不採用RCA的方式，改成CLA tree的架構，如 此將 16 bit adder的delay再繼續下壓，且critical path的output出現不 在MSB就合理了。整體架構如下圖(紅框部分改成CLA ):

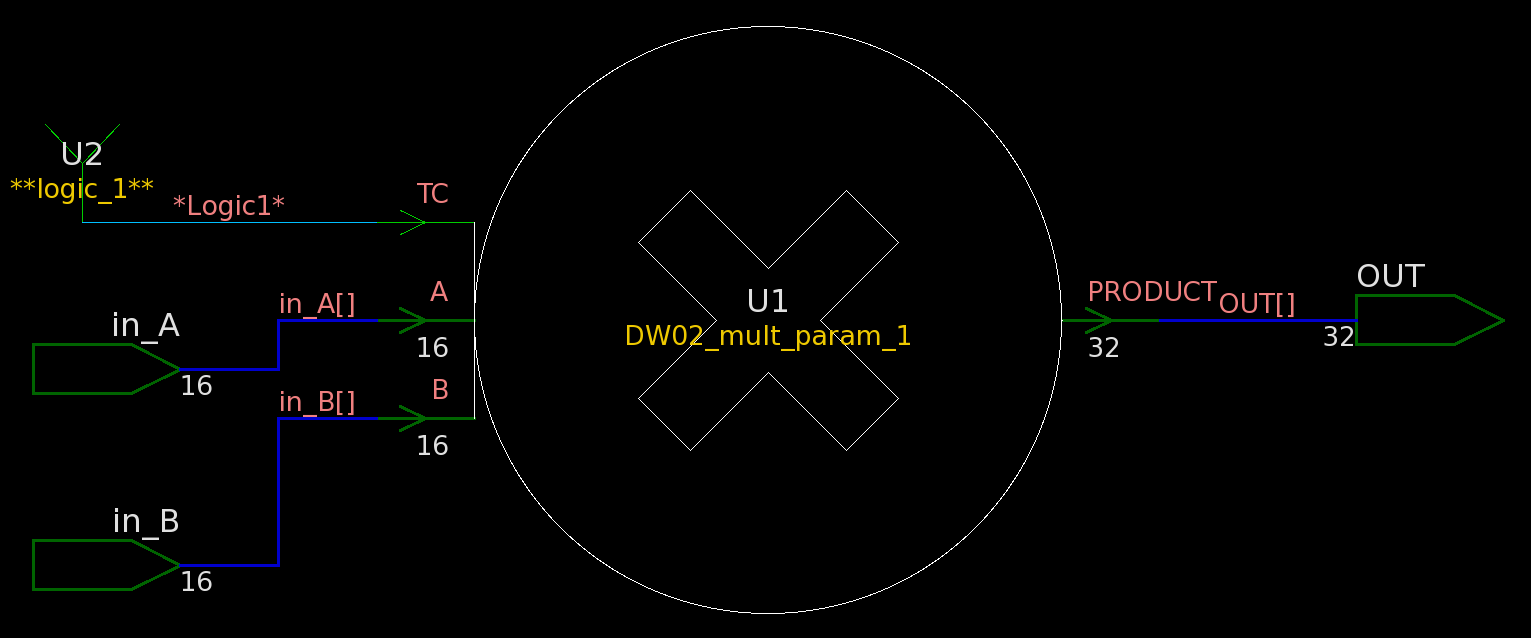




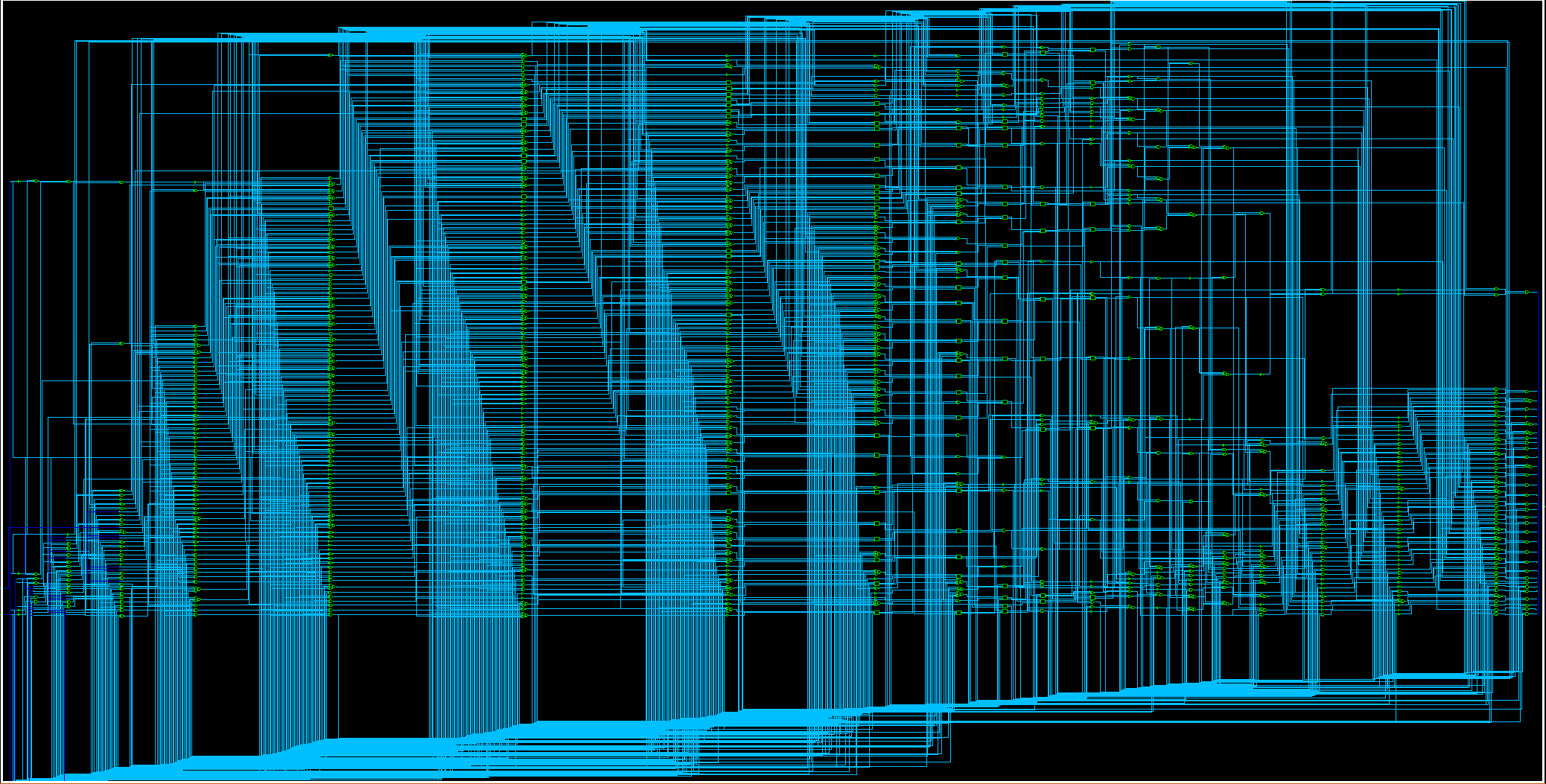
1. **31bits radix-4 booth multiplier**
2. **the smallest clock period: 3.26**
3. **area report (picture)**



1. **pre-synthesis schematic (picture)**



1. **post-synthesis schematic (picture)**



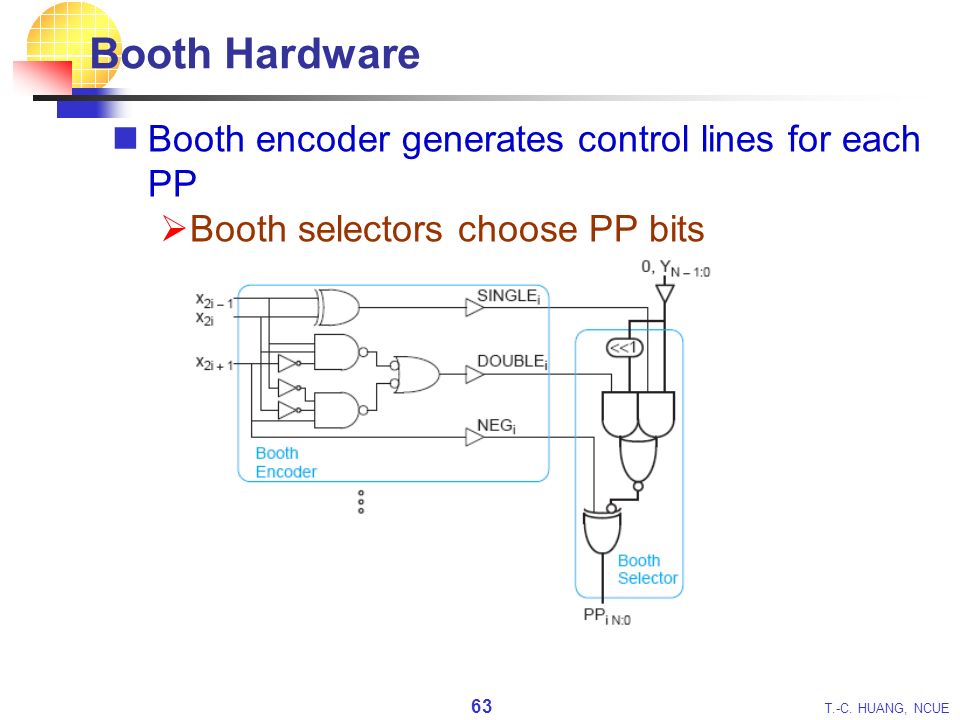
1. **Please explain the difference between pre-synthesis and post-synthesis**

Pre-synthesis看不出電路的架構，只是判斷function正確。

Post-synthesis架構 : Radix-4 booth multiplier為了解決AND multiplier partial products過多 的問題而引入的。Radix-4代表以4為基底，去替換掉原本以二進制 表示的multiplier。因此multiplicand的倍數可能有0, 1, 2, 3。其中3 倍的運算會很複雜，因此使用Booth encoding如下表取代3 :

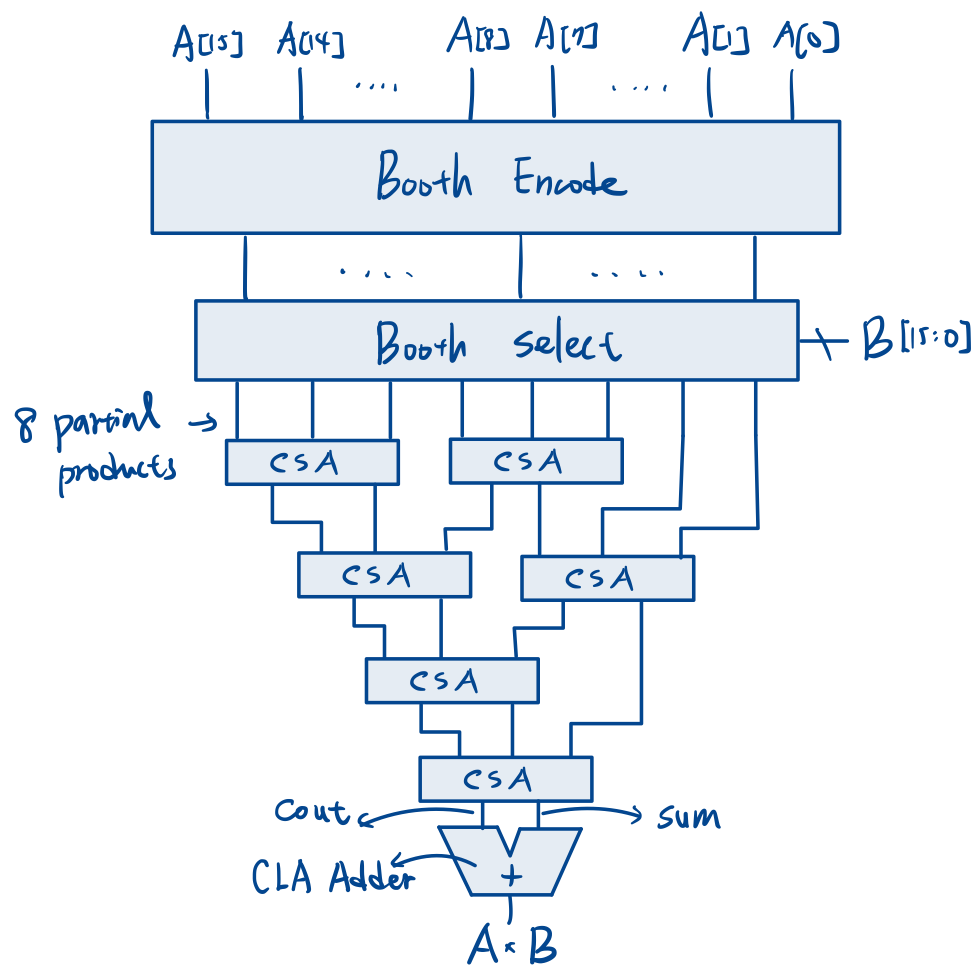
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Inputs | | | Partial product | Booth Select | | |
| X2i+1 | X2i | X2i-1 | PPi | Si | Di | Ni |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | Y | 1 | 0 | 0 |
| 0 | 1 | 0 | Y | 1 | 0 | 0 |
| 0 | 1 | 1 | 2Y | 0 | 1 | 0 |
| 1 | 0 | 0 | -2Y | 0 | 1 | 1 |
| 1 | 0 | 1 | -Y | 1 | 0 | 1 |
| 1 | 1 | 0 | -Y | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

其中，X為multiplier。可以藉由multiplier得到Booth select的三個數 值。其中Si = 1代表PPi為1倍的multiplicand ; Di = 1代表PPi為2倍 的multiplicand ; Ni = 1代表PPi是負數。結果對應可以由上表看出。 以下是Booth encoding和Booth select的電路 :

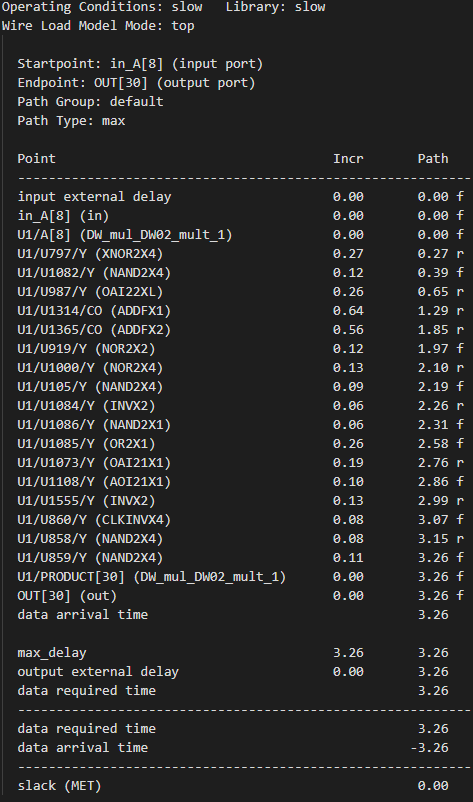


因此這個multiplier的架構在經過booth後只需要加8組的partial product (每個partial product是32 bits)，相當於砍半。partial product 也可以用AND-Multiplier最後的CSA + CLA架構去加速。

以下為我推估該乘法器的大概架構 :

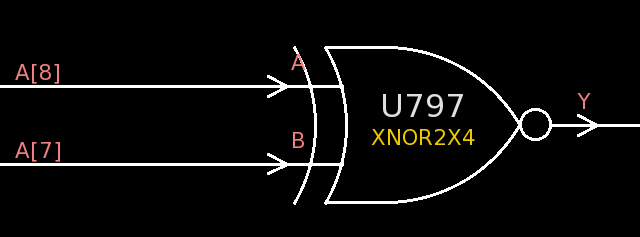


1. **timing report**



1. **Describe and explain the critical path**

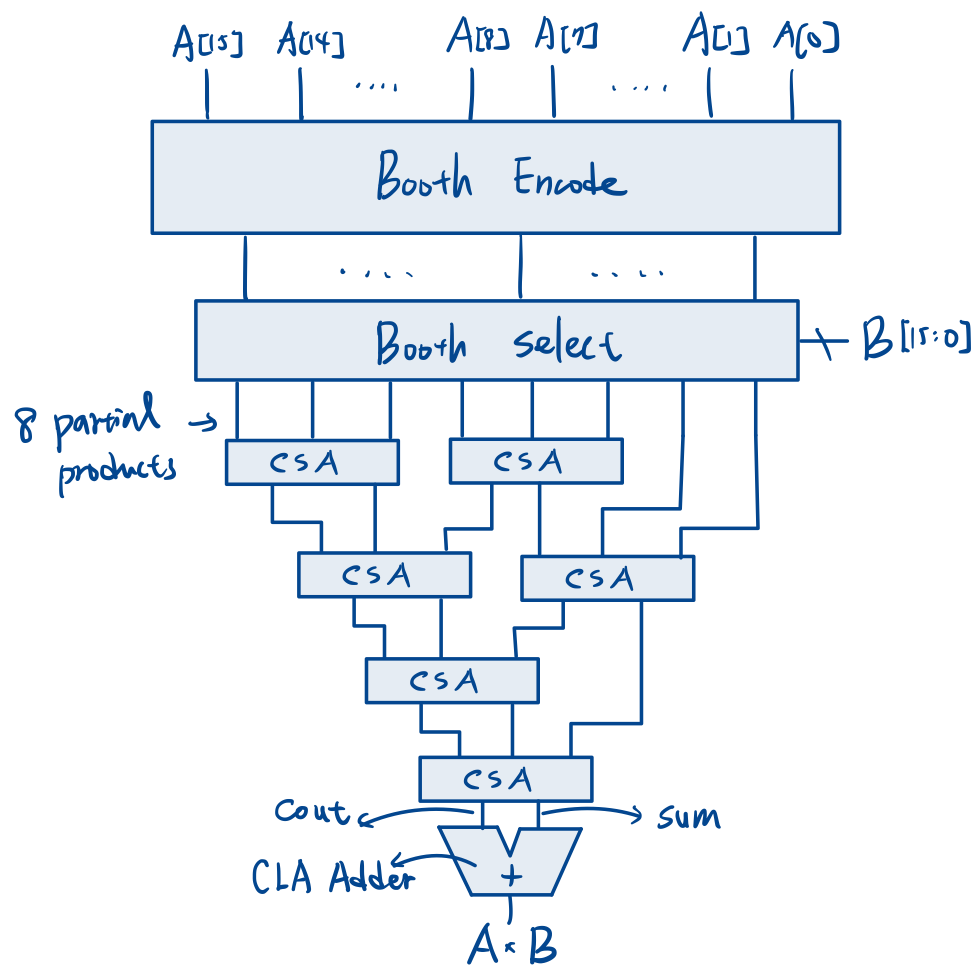
根據timing report可以看到critical path起點是下圖的XNOR。輸入 是A[8] A[7]。此為Booth encoder邏輯的一部分 (用於生成Booth select的訊號)。因此可得知此架構將A訂為multiplier，B訂 為multiplicand。且A[8]、A[7]訊號剛好是16\*16乘法的正中間，因 此最後面做CSA時會有多多input要處理 (更多level的CSA)， propagation delay會是最長的。



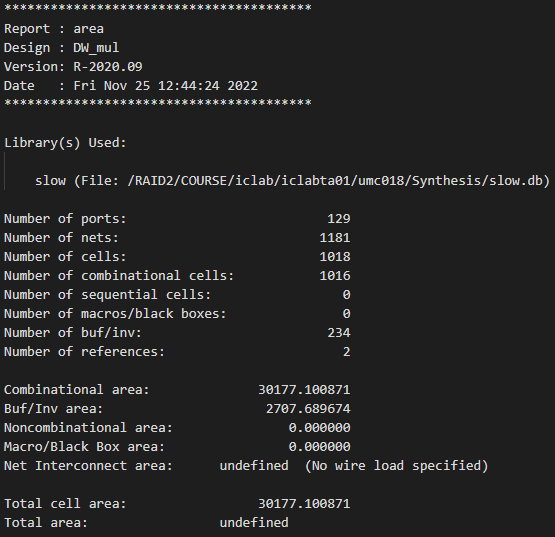
以下是critical path的流程 :

Input (A[8] A[7])=> Booth Encode => Booth select

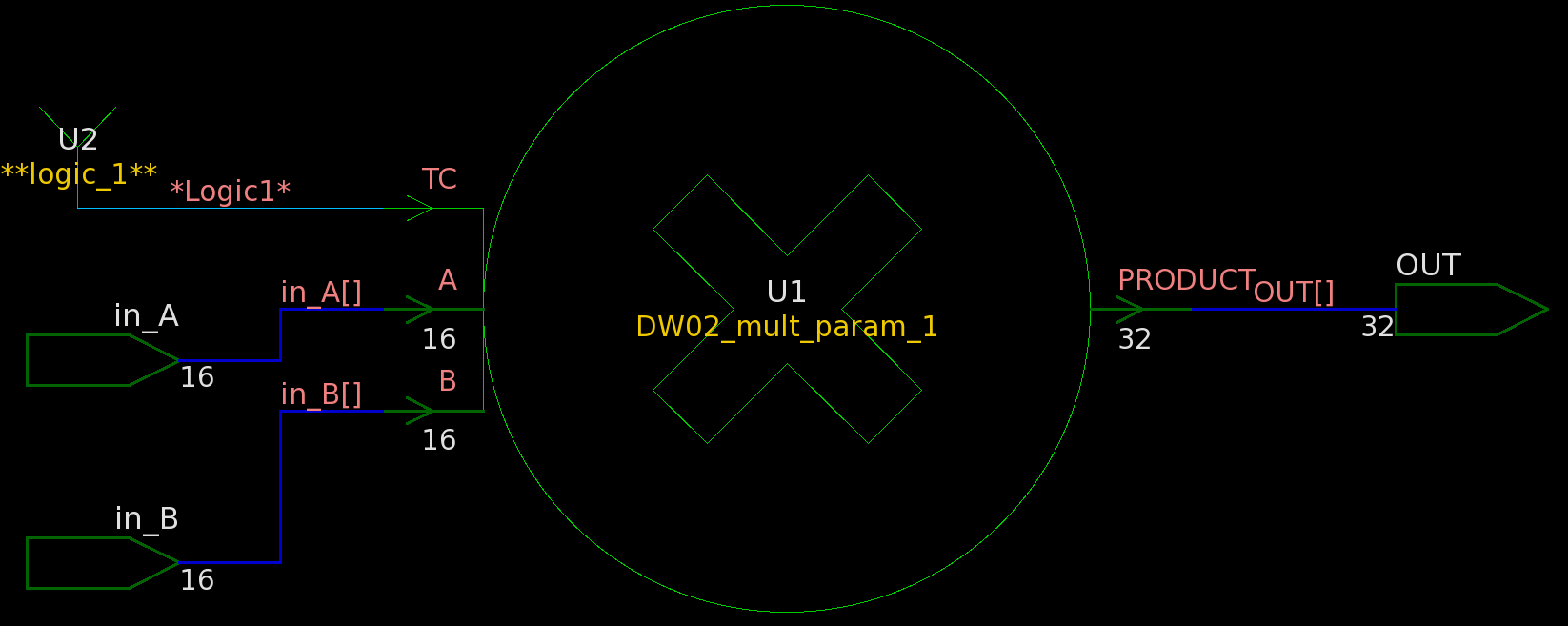
=> multiple CSA Level => CLA adder => output



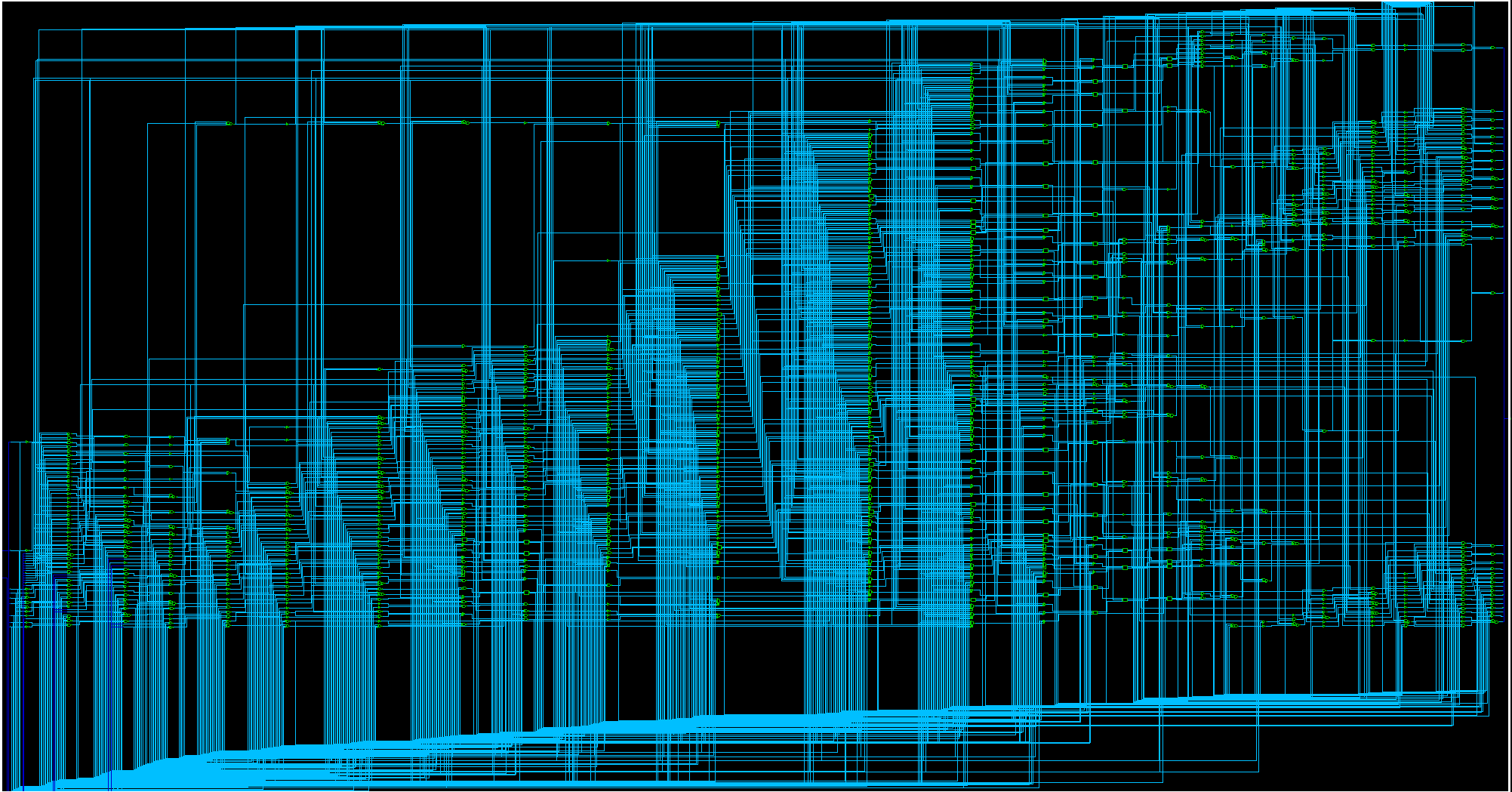
1. **31bits radix-8 booth multiplier**
2. **the smallest clock period: 3.73**
3. **area report (picture)**



1. **pre-synthesis schematic (picture)**



1. **post-synthesis schematic (picture)**

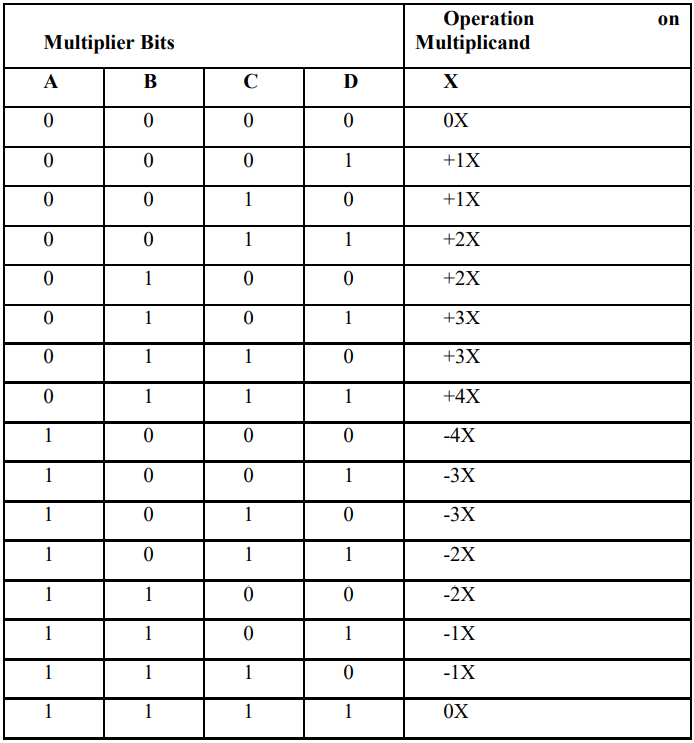


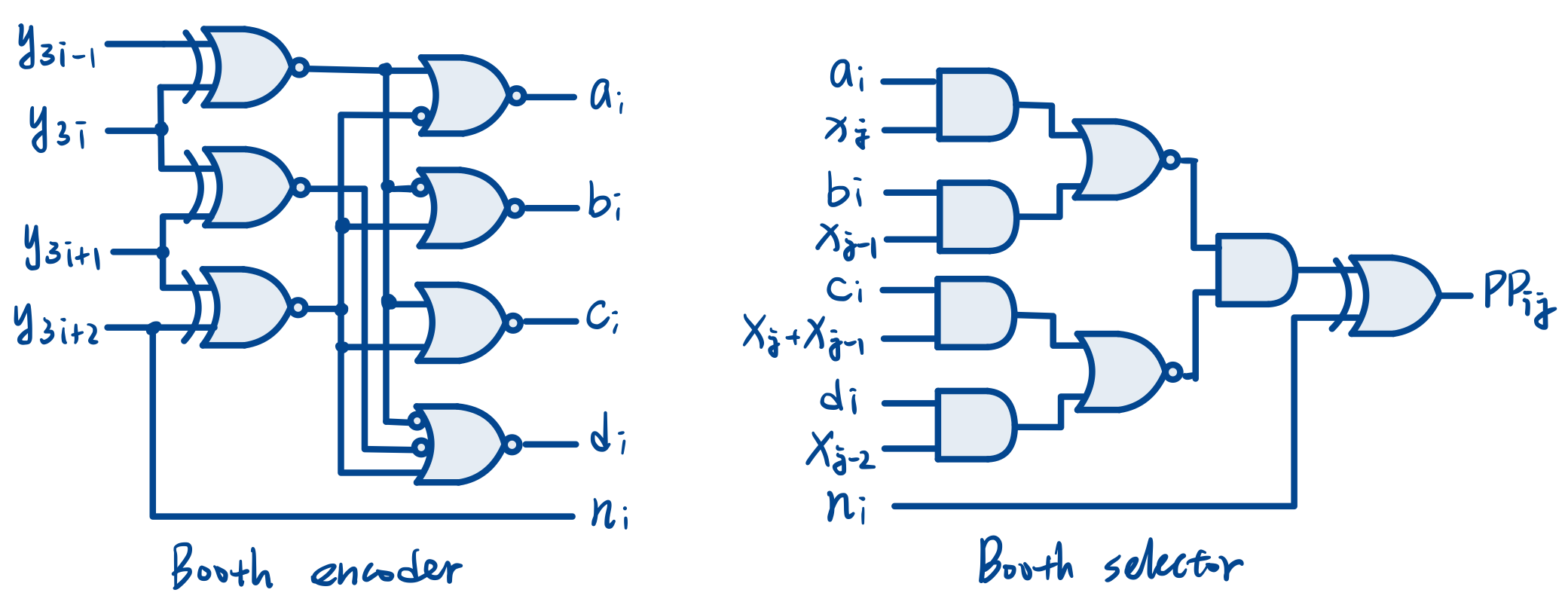
1. **Please explain the difference between pre-synthesis and post-synthesis**

Pre-synthesis看不出電路的架構，只是判斷function正確。

Post-synthesis架構 : 和radix-4 booth multiplier結構雷同，但radix-8的encode又更複雜。 去替換掉原本以二進制 表示的multiplier。因此multiplicand的倍數可 能有0 ~ 7，其中5, 6, 7我們可以用-3, -2, -1去encode。 根據下表的encode可以將binary based的multiplier轉成octave based

i.e. 2510 = 011001(0)2 => radix-8 encode (0110) (0010) =>1\*80 + 3\*81



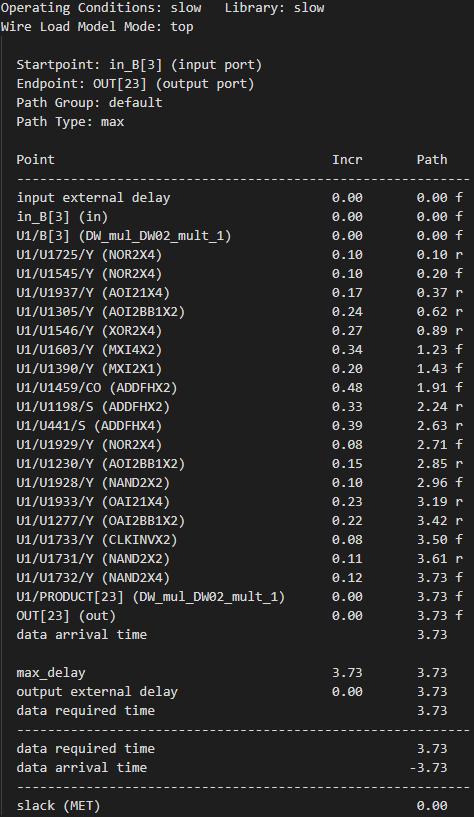


上方有booth encoder和booth select的邏輯，以下是booth encoder的 output對應的意思。Selector output ( PPij )則為partial product。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ai | bi | ci | di | ni |
| | PPi | = 1 | | PPi | = 2 | | PPi | = 3 | | PPi | = 4 | PPi < 0 |

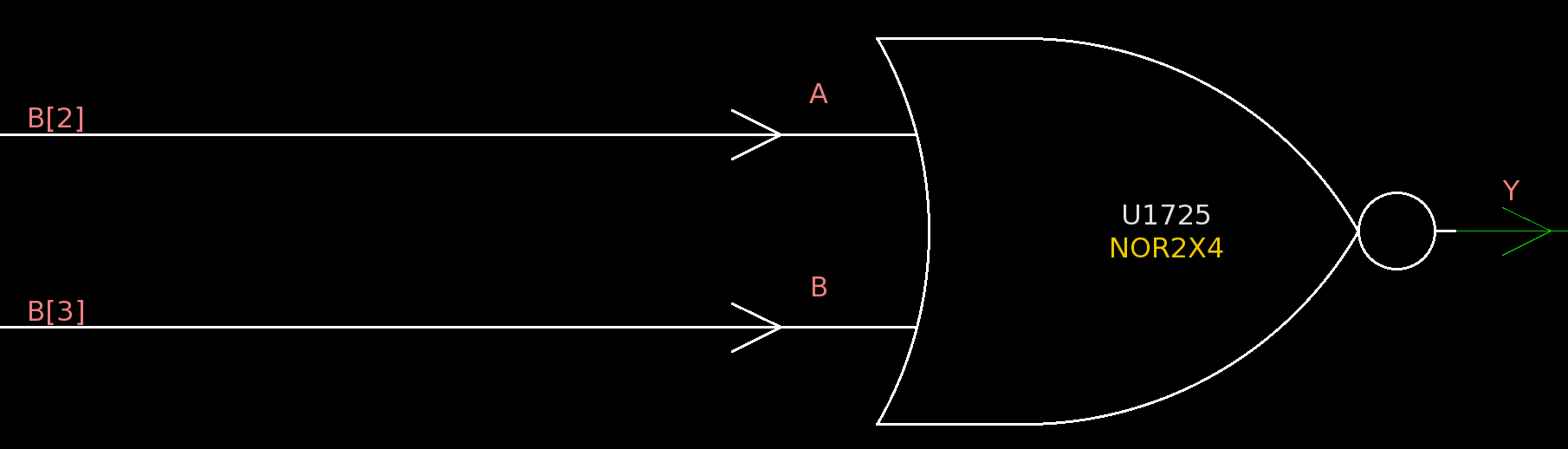
在此radix-8乘法器中，partial product的總數會比radix-4來的更少。 最終只有16/3取上界 = 6個partial product要做相加。因此後方所需 的CSA level和CLA的面積部分會減少。整體結構上和radix-4 booth multiplier 差不多。(更細部比較寫在後方討論)

1. **Timing report**



1. **Describe and explain the critical path**

根據timing report可以看到critical path起點是下圖的NOR。輸入 是B[2] B[3]。此為Booth encoder邏輯的一部分 (用於生成Booth select的訊號)。因此可得知此架構將B訂為multiplier，A訂為 multiplicand。



以下是critical path的流程(大架構和radix-4雷同) :

Input (B[2] B[3])=> Booth Encode => Booth select

=> multiple CSA Level => CLA adder => output

1. **Compare the pros and cons of architectures of and-multiplier, radix-4 booth multiplier and radix-8 booth multiplier.**

先將and-multiplier 和booth multiplier (radix-4, 8)之間做比較 :

And-multiplier Pros : 面積小、硬體架構整齊

Cons: partial product很多，速度會隨bit數上升大幅變慢。

Booth-multiplier Pros : partial product少非常多，在後級加法的計算上較快。

Cons:面積上因為多出booth encoder、booth select等電路， 因此cost較大。

接著我們細究radix-4和radix-8在16\*16的case有甚麼差別 :

我們可以根據最後的結果看到就速度來說，radix-4相對and-multiplier有 稍微的提升，但radix-8的case不但沒有變快，反而還變慢了。在radix-4 的部分，因為Booth encode select的input訊號可以直接由multiplicand本 身或是shift過(\*2)來表示，因此硬體上部會有太大的prop. delay。大概只 多了五, 六級的gate delay，而因為partial product相較and-mult.而言是砍 半的，因此綜合起來最終critical path有稍微縮減。 Radix-8的booth selector input訊號有一個xj+xj-1項，代表多了Adder。且 booth encode、selector的電路上比radix-4複雜、龐大，且最終partial product 只比radix-4少了兩組，在後方加法以CSA實現的前提下並沒有優勢。因 此此次Lab做下來的結果radix-4的速度是最好的，radix-8面積最大，效 能也不出色。 在此值得注意的是，如果bit數不多的情況下，將booth encoding radix的 base number增大並沒有好處。如果今天增加radix-number，減少的partial product delay可以大於我booth encoder selector複雜化帶來的prop. delay， 那速度上才會再提升。不過考慮到radix-num上升會使面積增加不少，這 兩者速度、面積之間的tradeoff也是選擇multiplier要考慮的點。

1. **Questions**
2. **What is the difference between cell-based design and full-custom design? Please list out their advantages and disadvantages.**

Full-custom design : Logic cells、circuit到最後layout全部由工程師設計。

Pros : 設計上有較高的彈性，可以達到最optimal的面積和performance

Cons: 設計流程繁瑣、成本高，大規模晶片整合time-to-market不切實際

Cell-based design :使用已經預先設計好的component (stdcell)作為building block。而layout的拉線部分是customize (大規模由EDA做routing)。

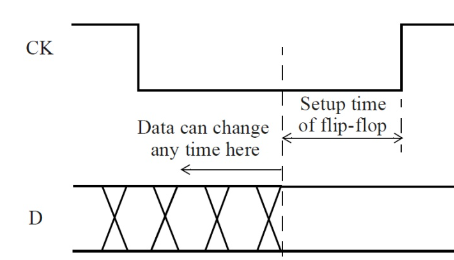
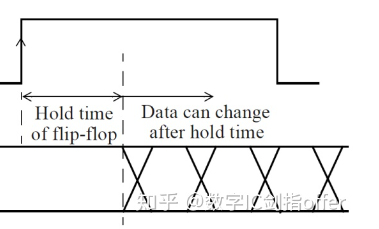
Pros : 設計速度快，較符合time-to-market。由於stdcell通常高度固定， 很方便由EDA tool去做place & route。因此對於大規模的晶片設 計會使用cell-based design + EDA。

Cons: 由於stdcell已固定，因此無法達到最優化的performance和面積。 整合度不會像full-custom那麼高。

1. **What is set-up time and hold-up time? If there are violation, how to fix it?**

在數位電路中，電路的速度和CLK rate有關。因此如何在極高的CLK freq.下不違反其timing是很重要的觀念。

今天register (DFF)在clk正緣處發前，D input必須達到穩定，而這兩者之間的最短時間就是set-up time，如左下圖所示 :



hold-up time則是正緣觸發後，D input須維持穩定的最短時間。(右上圖) 簡而言之，DFF的input要確保在posedge的前後，共維持穩定set-up time+hold-up time，如此才可以確保DFF的output是我們想要的值。 產生violation即代表D input在posedge觸發前set-up time內，或是posedge 觸發後hold-up time內改變了數值，如此DFF output Q是unknown狀態。 解決方法有 : 將clk freq降低、加快DFF前方combination circuit速度(即 為降低數位電路的critical path delay)、或是將前方circuit中 間加入pipeline design (加入更多級DFF)

**Summary:**

本次Lab相較乍看比之前的Lab稍微輕鬆一些，但其實不簡單。要分析大 量bit的加法器、乘法器不同架構，非常的耗費時間。要不斷上網爬閱資 料，還要用design vision重複檢視。中間很多分析並不適百分之百確定是 否和他生成的電路相同。不過我想大架構會是正確的方向，這次Lab學到 了乘法器進階的架構。我想是為了之後的project做準備吧QQ