[1] 設計者姓名及連絡電話

學生姓名：潘世軒、黃元培、吳隆暉

連絡電話：0978186136

[2] 專題名稱

中文專題名稱： 32 點快速傅立葉轉換處理器

英文專題名稱：32-point Fast Fourier Transform Processor

[3] 全新設計或改版說明

此案件為設計者全新設計

[4] 原理及架構說明

離散傅立葉變換(DFT)是一個在訊號處理的領域中相當常用的工具，他可以幫助我們從頻域的角度去分析或設計一個系統，而我們研究的主題：快速傅立葉變換(FFT)，是其中一類計算離散傅立葉變換的快速演算法，相較於直接計算 DFT，FFT 有效減少運算量，降低了計算複雜度。而本設計的研究目的，即是設計出能夠計算 32 點快速傅立葉變換的晶片。

我們選擇實作 FFT 的方法，是 Radix-II DIF FFT 演算法，此演算法利用離散傅立葉變換在複數平面上的對稱性質，將具有對稱性質的多個乘法合併，以大幅減少數學運算量，在相同的數學模型架構下，較有效率的算出結果。此演算法可大致分為兩種不同架構，時間點分組架構(DIT, Decimation in time)以及頻率點分組架構(DIF, Decimation in frequency)，本設計選擇 DIF 架構。以下分別介紹兩種分組架構。

DIT 架構之 Radix-II FFT 數學推導如下：

N 點離散傅立葉變換(DFT)的數學公式（假設 N 為偶數）為

$$X[k] = \sum_{n=0}^{N-1} x[n]\, w_N^{kn}, k = 0,1,2,\dots,N-1$$

（其中$w_N = e^{-\frac{j2\pi}{N}}$，稱作$twiddle\ factor$）

將奇數點與偶數點分開運算，可得

$$X[k] = \sum_{n(even)} x[n]\, w_N^{kn} + \sum_{n(odd)} x[n]\, w_N^{kn}$$

令偶數點的$\mathrm{n} = 2\mathrm{r}$，奇數點的$\mathrm{n} = 2\mathrm{r} + 1$，則

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r]\, w_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1]\, w_N^{(2r+1)k}$$

$$= \sum_{r=0}^{(N/2)-1} x[2r]\, w_N^{2rk} + w_N^k \sum_{r=0}^{(N/2)-1} x[2r+1]\, w_N^{2rk}$$

$$= \sum_{r=0}^{(N/2)-1} x[2r]\, w_{N/2}^{rk} + w_N^k \sum_{r=0}^{(N/2)-1} x[2r+1]\, w_{N/2}^{rk}$$

$$= \mathrm{G[k]} + w_N^k \mathrm{H[k]}$$

其中$\mathrm{G[k]} = \sum_{r=0}^{(N/2)-1} x[2r]\, w_{N/2}^{rk}$，$H[k] = \sum_{r=0}^{(N/2)-1} x[2r+1]\, w_{N/2}^{rk}$

化簡後 G[k]和 H[k]又可視為對奇數點與偶數點分別做 N/2 點的離散傅立葉轉換，再遞迴地對兩式繼續操作下去，即可計算完畢。

考慮 8 個點（N=8）的 FFT，其數據流程圖為

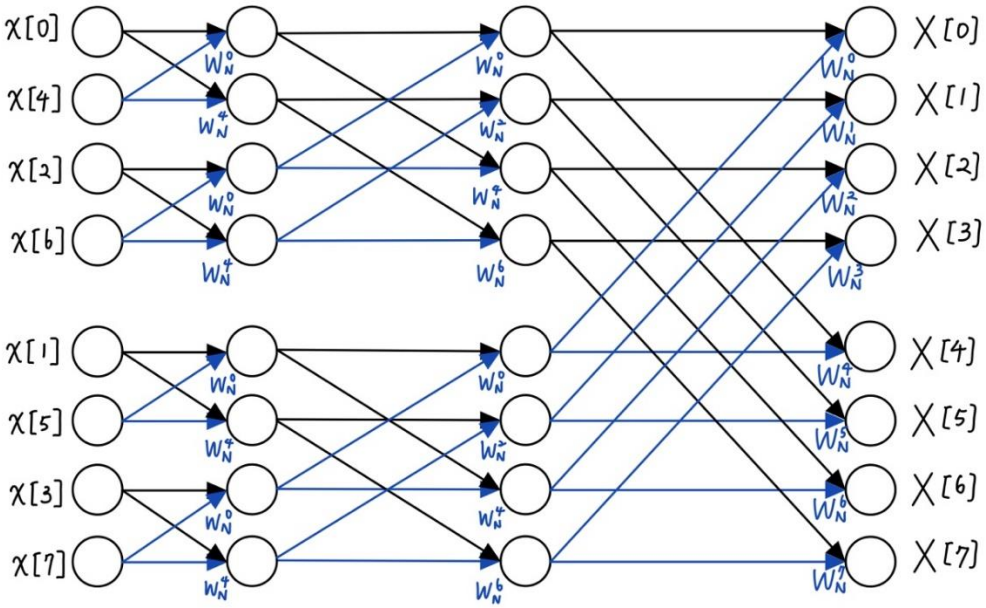


Fig.(1) N=8 Radix-II DIT FFT 數據流程圖

可觀察到上圖中有許多形似蝴蝶的運算架構，在此架構中被稱為蝴蝶圖

（DIT butterfly），觀察每一個蝴蝶圖，可以發現運算過程中乘上的兩個 twiddle factor 均有相位差 180 度，亦即正負號相反，因此我們可以將蝴蝶圖的結構化簡如下圖

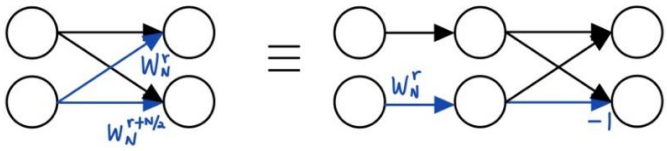


Fig.(2) DIT 蝴蝶圖化簡

並將數據流程圖化簡為

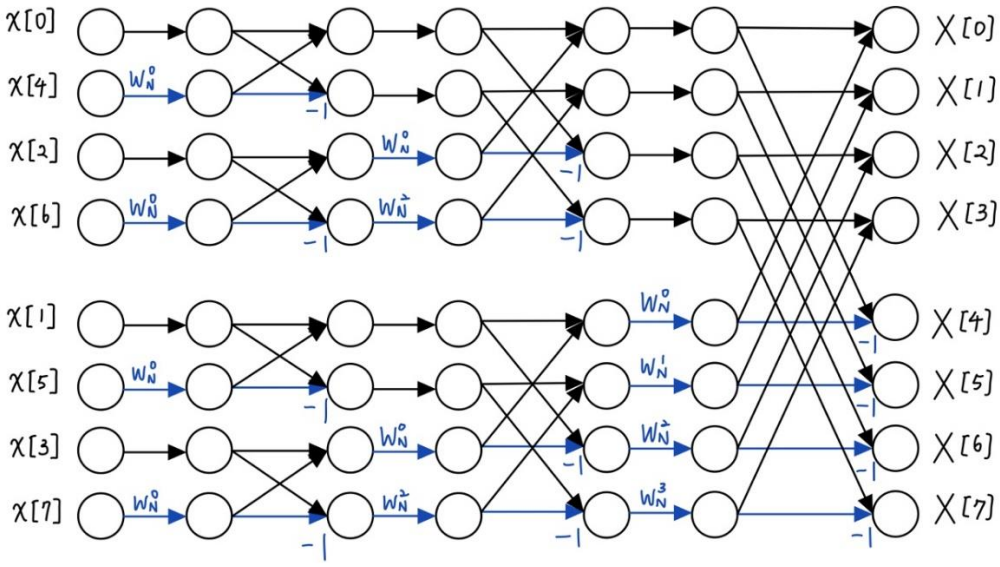


Fig.(3) 化簡過後的 DIT 數據流程圖

DIF 架構以頻率點切割，數學推導如下：

N 點離散傅立葉變換(DFT)的數學公式（假設 N 為偶數）為

$$X[k] = \sum_{n=0}^{N-1} x[n]\, w_N^{kn}, k = 0,1,2,\dots,N-1$$

（其中$w_N = e^{-\frac{j2\pi}{N}}$，稱作$twiddle\ factor$）

令偶數點的$k = 2r$，其中 r = 0, 1, 2, …, (N/2)-1，則

$$X[2r] = \sum_{n=0}^{N-1} x[n]\, w_N^{2rn} = \sum_{n=0}^{N/2-1} x[n]\, w_N^{2rn} + \sum_{n=N/2}^{N-1} x[n]\, w_N^{2rn}$$

$$= \sum_{n=0}^{N/2-1} x[n]\, w_N^{2rn} + \sum_{n=0}^{N/2-1} x[n+(N/2)]\, w_N^{2rn}$$

$$= \sum_{n=0}^{N/2-1} (x[n] + x[n + \frac{N}{2}])\, w_{N/2}^{rn}$$

此即將上半平面與下半平面的資料相加後，做 N/2 點的離散傅立葉變換。

令，奇數點的$k = 2r + 1$，其中 r = 0, 1, 2, …, (N/2)-1，則

$$X[2r+1] = \sum_{n=0}^{N-1} x[n]\, w_N^{(2r+!)n} = \sum_{n=0}^{N/2-1} x[n]\, w_N^{(2r+1)n} + \sum_{n=N/2}^{N-1} x[n]\, w_N^{(2r+1)n}$$

$$= \sum_{n=0}^{N/2-1} x[n]\, w_N^{(2r+1)n} + \sum_{n=0}^{N/2-1} x[n + (N/2)]\, w_N^{(2r+1)(n+\frac{N}{2})}$$

$$= \sum_{n=0}^{N/2-1} x[n]\, w_N^{(2r+1)n} + w_N^{(2r+1)(N/2)} \sum_{n=0}^{N/2-1} x[n + (N/2)]\, w_N^{(2r+1)n}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] - x\left[n + \frac{N}{2}\right] \right) w_N^{(2r+1)n}$$

$$= \sum_{n=0}^{N/2-1} (x[n] - x[n + \frac{N}{2}])\, w_{N/2}^{rn} w_N^n$$

此即將上半平面與下半平面的資料相減，乘上 twiddle factor 後，做 N/2 點的離散傅立葉變換。

考慮 8 個點（N=8）的 FFT，將分組後的訊號遞迴地往下分組運算，蝴蝶圖的化簡方式與化簡過後數據流程圖為

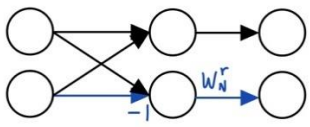


Fig.(4) DIF 蝴蝶圖化簡

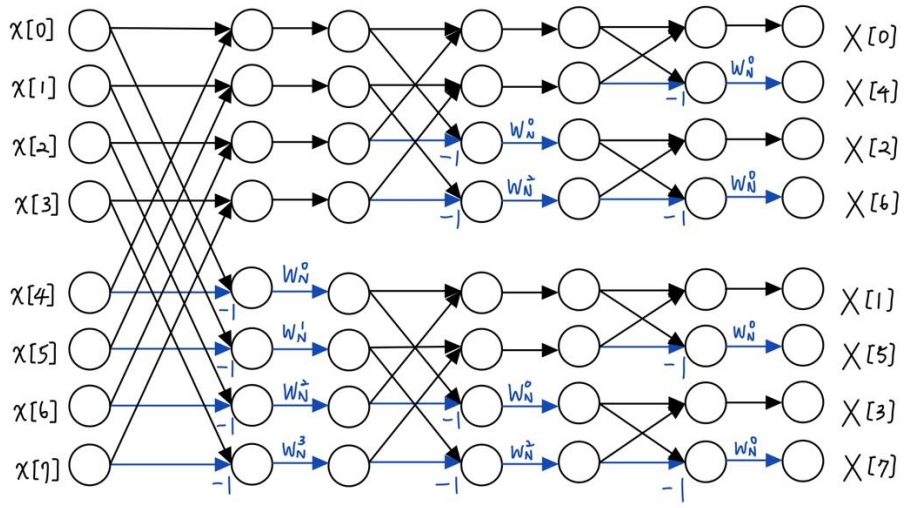


Fig.(5)　化簡後的 DIF 數據流程圖

此架構即為我們最終採用的演算法架構。[2]

考慮原本的離散傅立葉變換，假設輸入為 N 個點的訊號，則直接用公式計算約需要使用 N 次複數乘法以及 N-1 次的複數加法，每次複數乘法需要 4 次實數乘法以及 2 次實數加法，每次複數加法需要 2 次實數加法，因此合併來看 N 個輸出點共需要$4N^2$次實數乘法以及$4N^2-2N$次實數加法，運算次數複雜度為$O(N^2)$。而 Radix-II DIT FFT 演算法將計算切為log_2N個 stage，每個 stage 包含 N 次複數加法、N/2 次複數乘法以及 N/2 次變號，總共需要$2Nlog_2N$次實數乘法、$3Nlog_2N$次實數加法以及$Nlog_2N/2$次變號，運算次數複雜度為$O(Nlog_2N)$，相較於原本的計算量降低了許多。

在採用 DIF 架構的 Fast Fourier Transform 時，我們也要考慮如何表示小數，考量到 floating point 在做加法或者乘法的時候的運算單元複雜度高，以及面積非常大，因此我們選擇採用二補數的 fixed-point 表示法，此表示法的基本使用方法如下：

舉例來說，考慮一 binary 的數字 x：001100 01010，其整數部分占六位、小數部分占五位，這樣的話，x 在我們二補數 fixed point 表示法中的值就是值 394 除以 32，因此在十進位的表示法為 12.3125。其中 394 為 00110001010 的值，而因為小數點占五位，因此實際上他表示的值為 394 除以 32。

透過此種 fixed-point 的表示法有幾項好處，首先就是在做這些數字相加的時候，我們只要透過 signed extension 把兩個數字的小數點對齊之後就可以直接透過加法器將兩數相加，乘法亦然，我們可以使用乘法器直接將兩數相乘。再來就是 fixed-point 表示法對我們來說比較容易理解，我們便可以將心思放在電路的設計而非那些運算單元的設計。因此最後我們採用 fixed-point 表示法來表示運算過程中的數值以及 twiddle factor 的值。

本設計採用 32 點輸入，每個輸入點之實部有 11bit，小數點前有五位，小數點後有六位，我們假設時域訊號不會有複數，因此輸入點虛部為零，電路架構如下圖

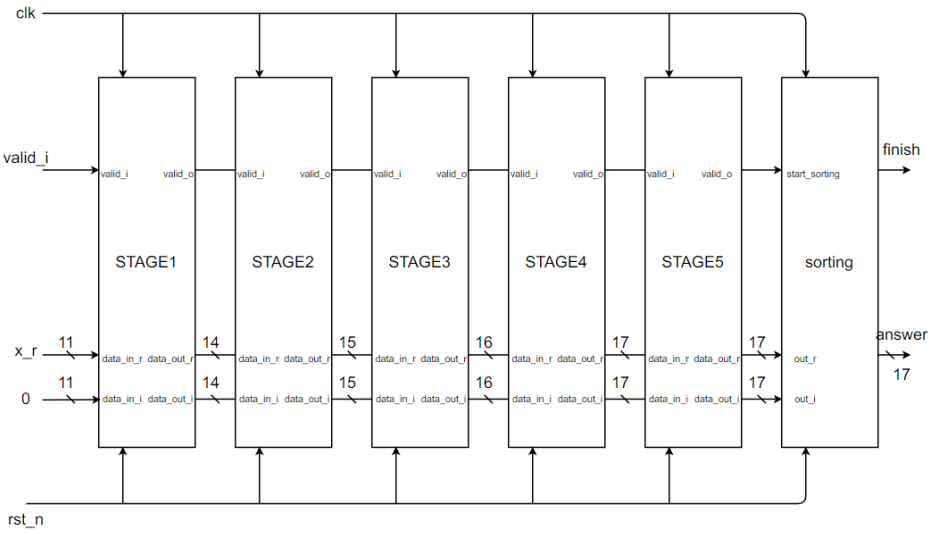


Fig.(6)　整體架構

其中每個 stage 皆為 Serial-in-serial-out，使用 shift register 來暫存數據，並使用 counter-based control unit 來控制 twiddle factor、state 以及 output，stage 的工作流程可以劃分成 4 個 state：IDLE, WAITING, FIRST, SECOND，以下講解各個 state 工作內容：

1. IDLE：等待輸入。
2. WAITING：從吃進第一個輸入開始，會依序將輸入值填進 shift register，此 state 恰好填滿時結束，即進入下一個 state FIRST。
3. FIRST：此 state 將 shift register 暫存的值讀出，與當下的輸入值相加輸出（輸出值為推導中的 G[k]），並將新輸入的值減去 shift register 吐出的值再填入 shift register，當輸出一半的值後，即進入下一個 state SECOND。
4. SECOND：將 shift register 吐出的值與 control unit 給的 twiddle factor 相乘並輸出，輸出完畢回到 IDLE。相乘之後的結果我們一律保留小數點後 7 位，避免運算到後方的 stage 時計算量太大。

除此之外，我們在每個 stage 之中傳遞資料的時候每次用來表示整數的 bit 數會增加 1，因為考量到在運算的過程中會經過加法和乘法，其中乘法皆是乘以不大於 1 的數字，因此不必考慮 overflow，但加法可能會導致 overflow 的情形發生，因此我們在每一個 stage 之間將 data 的總 bit 數增加 1 來避免 overflow。

在經過五個 stage 之後，理論上應該已經將完成快速傅立葉變換的數據輸

出，但實際上出來的順序並不是按照 X[0],X[1]…X[31]的數據排列的(可以參考上方 Fig.(5))，因此我們需要做 bit reversal 的排列使得最後的輸出可以按照 X[0],X[1]…X[31]的順序排列，Fig.(6)中的 sorting 即是用來實現此 bit reversal 排列的電路。

此外，輸出腳位為 answer，共 17bit，其中小數部分占 6bit，整數部分占 11bit，考量到晶片以及量測用的儀器腳位限制，我們必須將此晶片的總腳位數量控制在 40 以內，因此我們最後的輸出按照順序會是 X[0]到 X[31]的實部，再來是 X[0]到 X[31]的虛部，最後總共會花 64 個 cycle 來將結果順序以及實部虛部輸出。Fig.(6)中的 sorting 除了解決上一段中的 bit reversal 問題，也用來控制將實部和虛部依序輸出。

具體作法如下：

首先，我們觀察 Fig.(5)當中的簡化模型，我們可以發現，輸出的 X[n]並不是按照順序的結果，不過，我們可以經由完整的 DIF FFT 流程圖，整理出實際上的 output 順序，實際的作法為，一步一步照著演算法的設計，觀察每個 stage 的 output 順序，得到最後的輸出順序如下圖

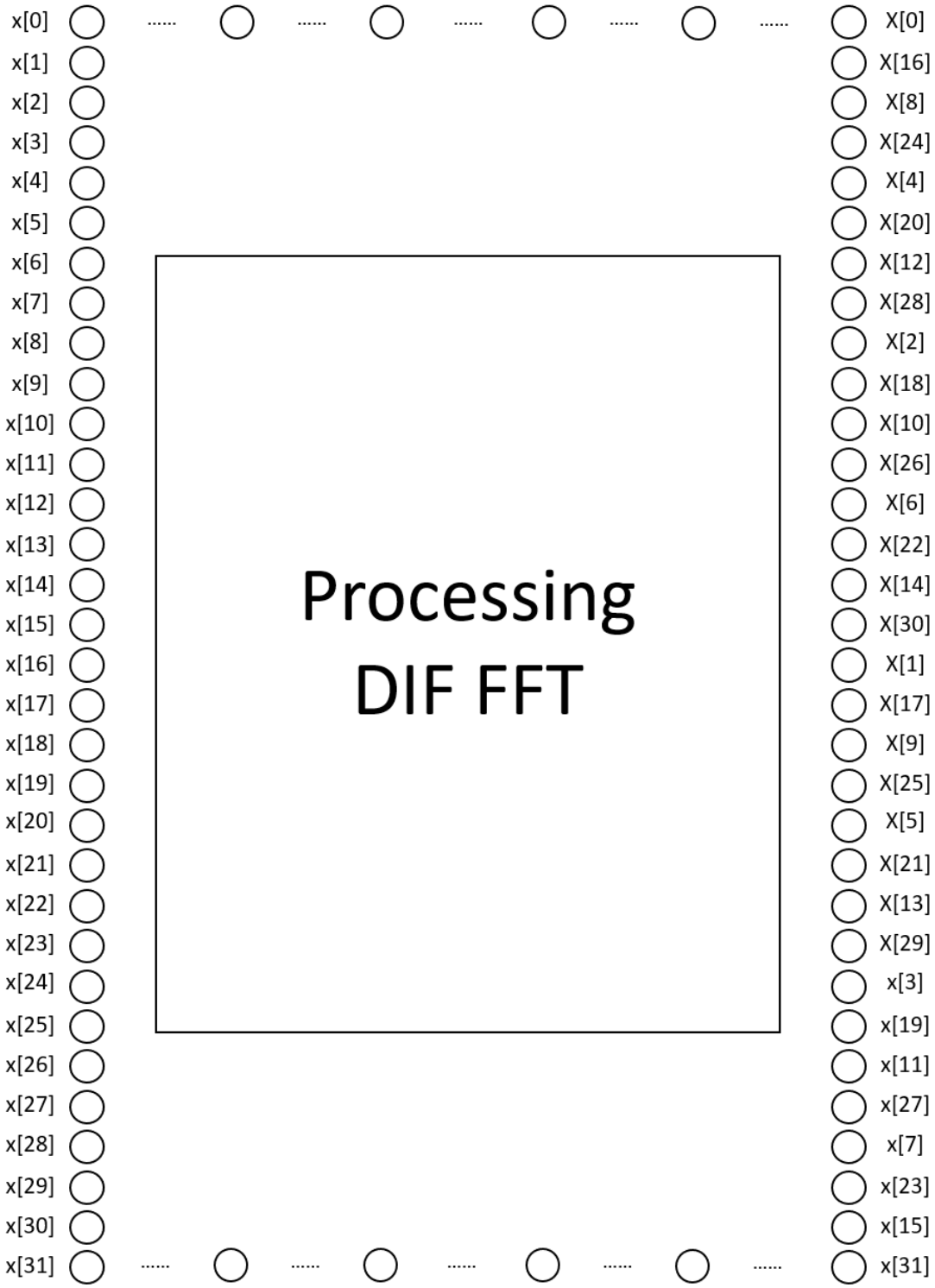


Fig.(7) 完整 DIF FFT 輸入順序及輸出順序示意圖

其中，左邊直行由上到下代表 input 的輸入順序，右邊直行從上到下代表 output 的輸出順序。因此，我們在得知 output 的資訊後，我們設計了一個做 bit reversal 的 module，將所有的 output 暫存至一個 64(個)*17(位)的 register 中，再將這些 output 按照正確的順序一個一個輸出，先輸出 X[0]~X[31]的實部，再輸出 X[0]~X[31]的虛部。

最後，下表是本晶片的輸入及輸出腳位配置：

Name	I/O	Width	Description
clk	I	1	Positive edge trigger clock
valid_i	I	1	Active-high signal that tells if input is valid
rst_n	I	1	Asynchronous active-low reset signal
x_r	I	11	Serial-in time domain input signal
finish	O	1	Asynchronous active-high signal that indicates the data output is done
answer	O	17	Serial out frequency domain output signal

表一 輸入及輸出腳位

在設計電路之外，我們另外寫了 python 檔來驗證電路的正確性，其具體功能包括將數值轉換為二進位 fixed-point、以及按照電路順序生出每一個 stage 過後的 output(binary)，還有將 binary 數據轉化為我們常看到的小數格式並劃出 spectrum 功能，透過 python 輔助產生的 Expected output，我們可以在 testbench 中讀取這些檔案，將其與電路的 output 做比較來驗證電路的正確性。總共使用十組數據來驗證電路的正確性，最後的結果是十組數據皆能成功，畫出來的 spectrum 也和 matlab 內建的 FFT 畫出來的 spectrum 雷同。

[5] 設計流程

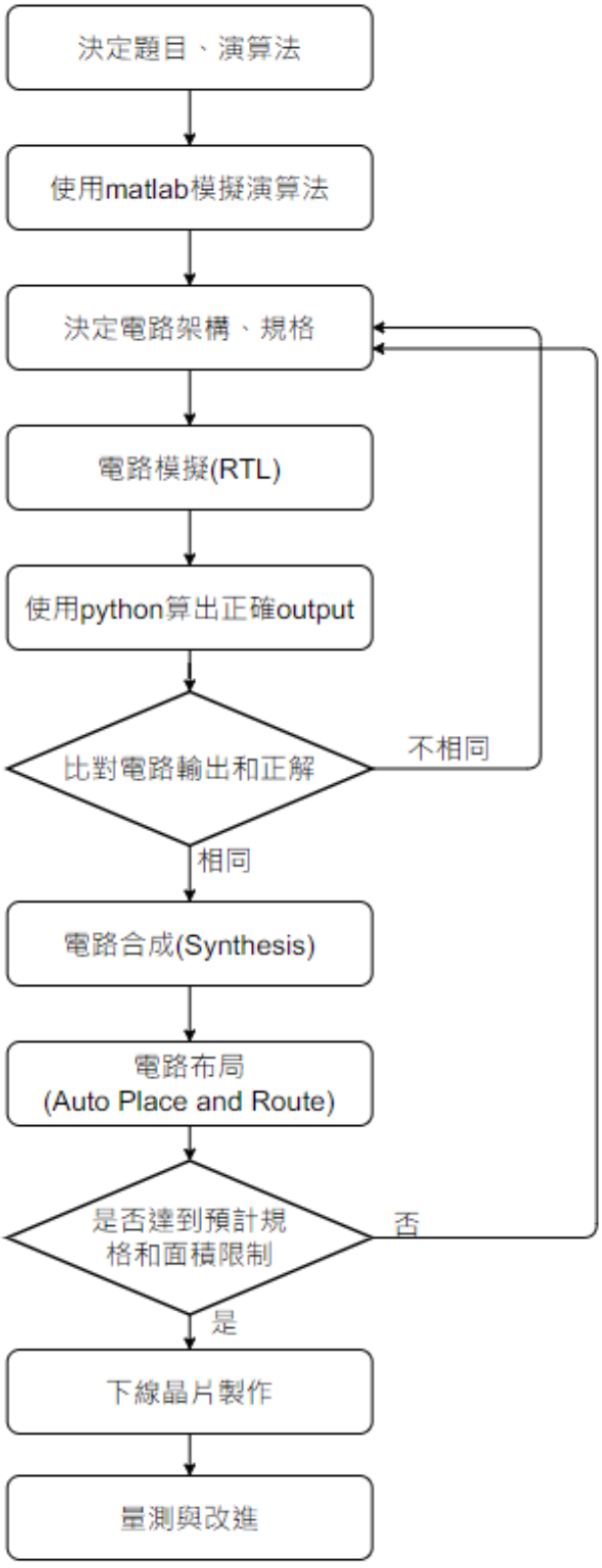


Fig.(8) 設計流程圖

[6] 電路詳圖

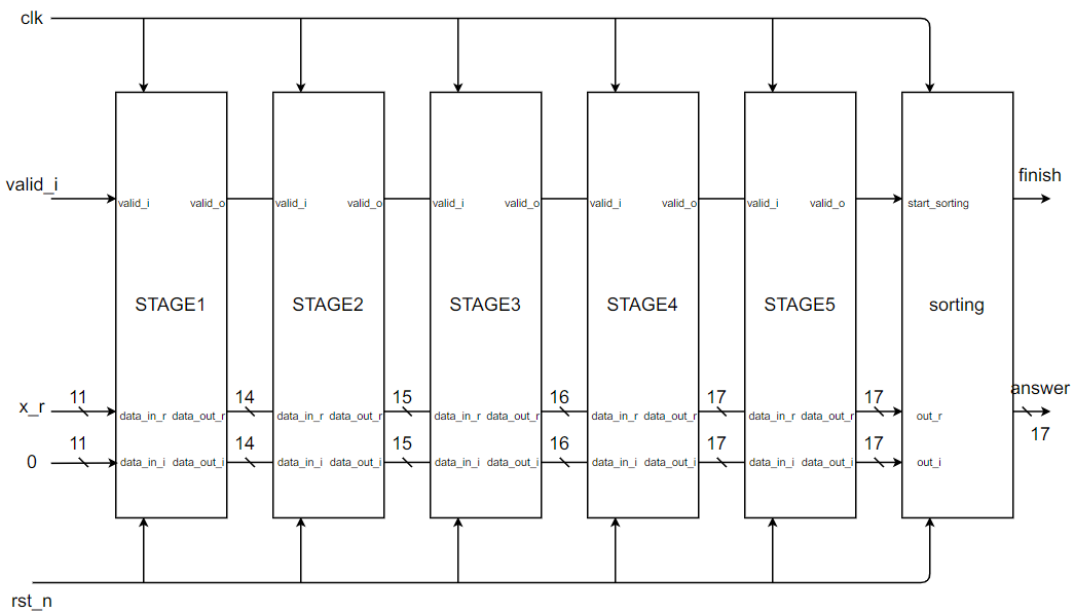


Fig.(9) 整體架構

其中，每一 stage 電路圖為 butterfly 架構，如下(以 stage1 為例)，其中 SR16 為長度 16 的 shift register 而下方的 CTRL16 則是用來控制 state(state 控制了送進 shift register 的數是從上一級的 output 還是此 stage 中的減法結果、以及此 stage 的 output 是加法結果還是減法和 twiddle factor 的乘積)以及送出 twiddle factor 給上方之乘法器

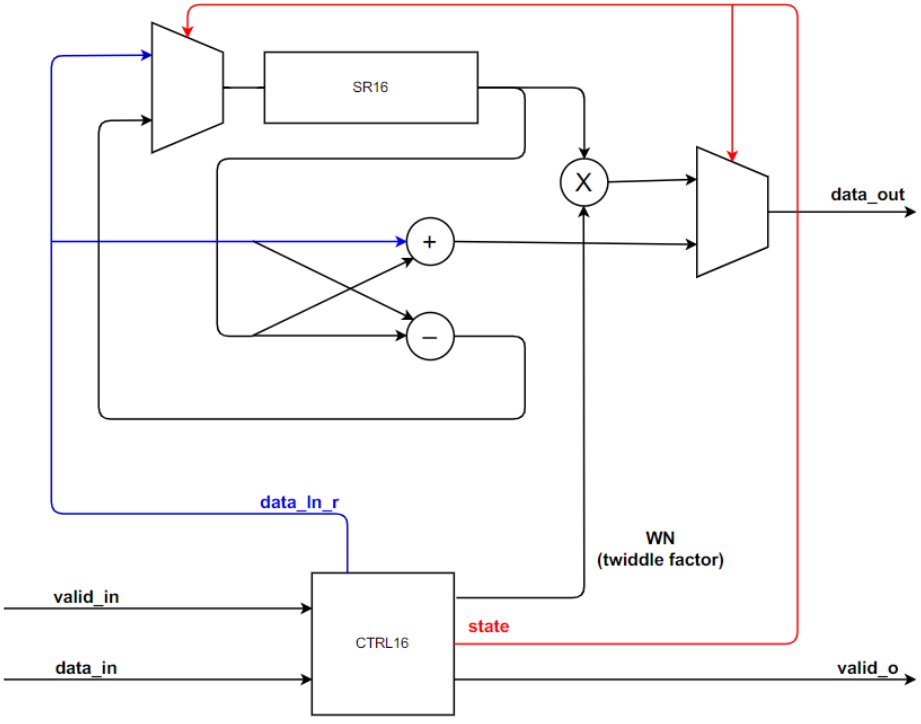


Fig.(10) stage1 架構圖

[7] 模擬結果

波形圖：(時間單位 ps，週期為 10ns)

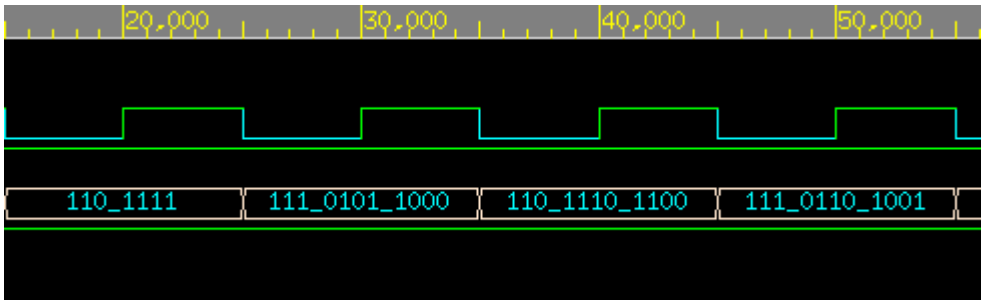


Fig.(11)　輸入波形圖(節錄)

上圖為 Input 波形圖，每一個 cycle 會輸入一組以二補數表示，小數部分占 6bit，整數部分占 5bit 的數值(第三個波形)，連續輸入 32 個 cycle，分別代表 x[0], x[1]…x[31]，當 32 個 cycle 結束後，會將 valid 訊號拉為 0，代表輸入結束。

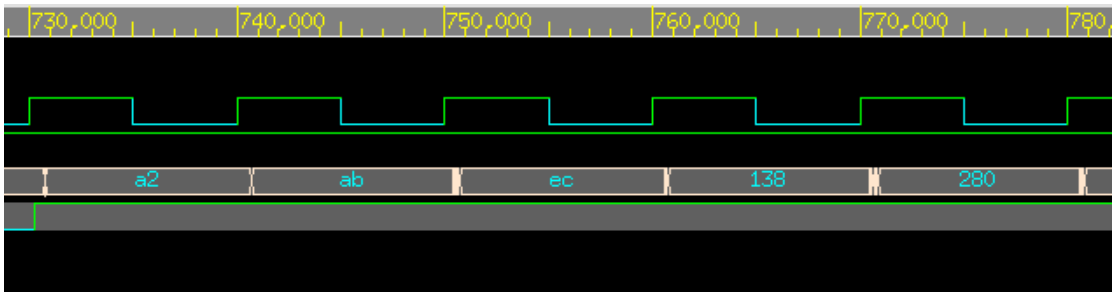


Fig.(12)　輸出波形圖(節錄)

上圖為 output 波形圖，當計算完成後，會依據傅立葉變換之後的結果，將 X[0], X[1]…X[32]的實部輸出，接著再將 X[0], X[1]…X[32]的虛部輸出，其數值以以二補數表示，小數部分占 6bit，整數部分占 11bit(第三個波形)，在輸出的同時將 finish 拉至 1，代表有效的輸出。完成整個快速傅立葉變換(包含排列輸出次序)需時約 72 個 cycle。

在完成快速傅立葉變換之後我們可以將晶片的輸出透過 python 來畫出 spectrum，並和 matlab 模擬出來的結果相比較，我們共準備了十組數據，對每一組數據，晶片都能確實輸出正確答案，以下提供了其中兩組數據，可以看見圖形幾乎完全雷同。

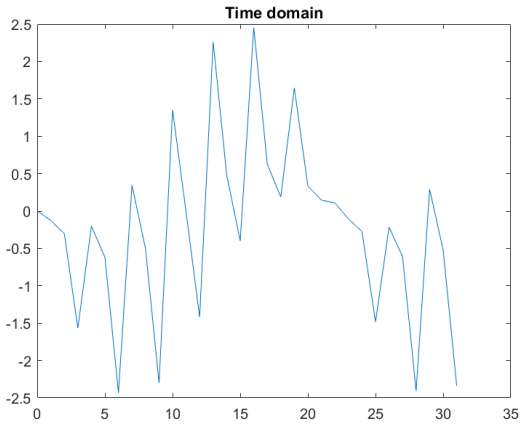


Fig.(13) x = sin(2t)+sin(4t)+sin(6t)

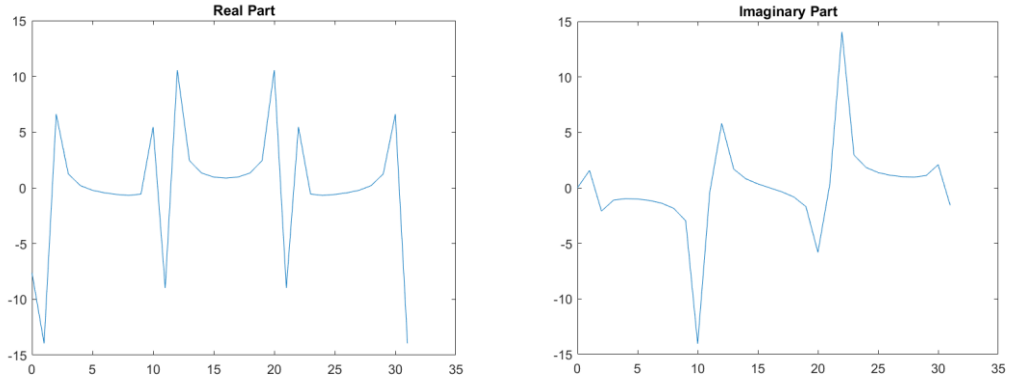


Fig.(14) matlab 模擬出之頻譜

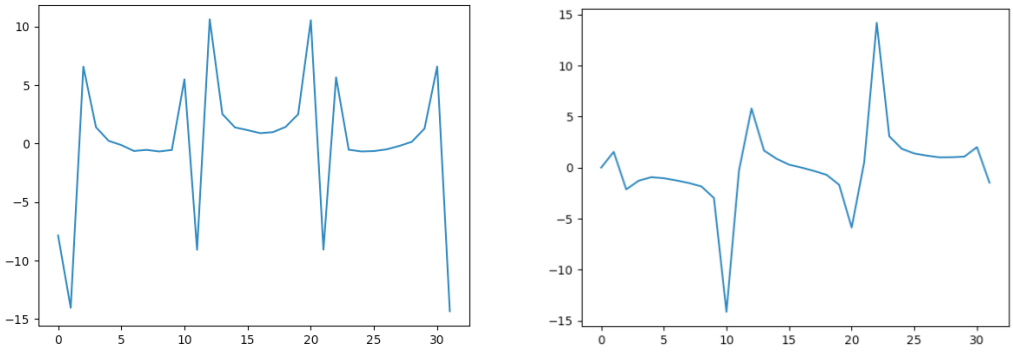


Fig.(15) 晶片輸出之頻譜

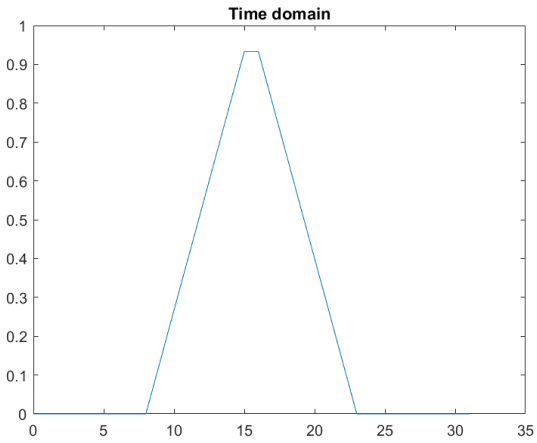


Fig.(16) x = traingle(8,23,t)

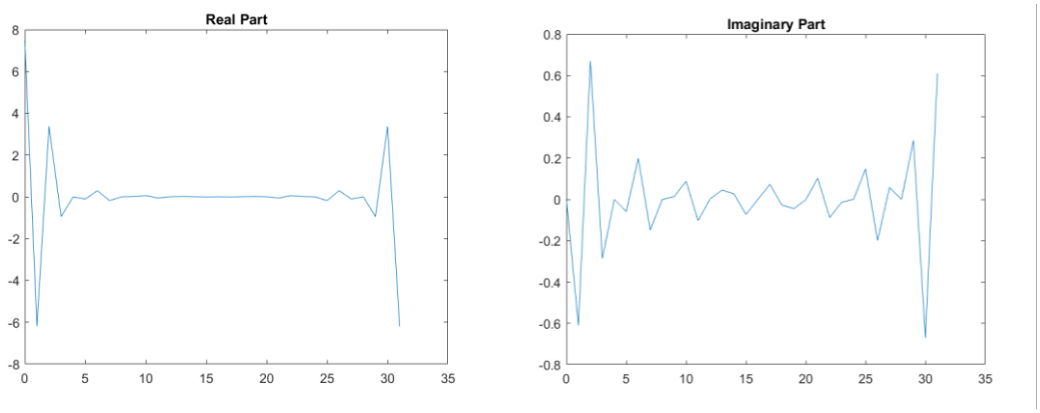


Fig.(17) matlab 模擬出之頻譜

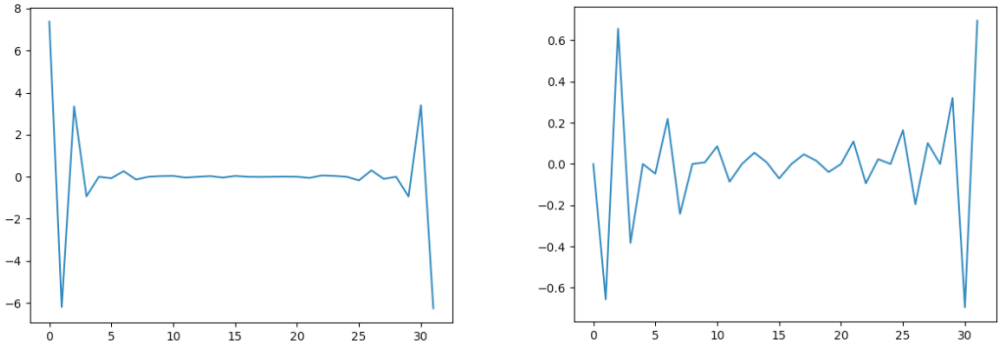


Fig.(18)晶片輸出之頻譜

[8] 量測考量

預計量測流程

(1) 將準備好之 input pattern 轉換為 fixed point，輸入按照順序準備好，利用訊號產生器向晶片輸出，此為測試資料

(2) 儀器設置：電源供應器調整為 1.8V 直流電，並接上晶片的電源腳位

(3) 儀器設置：訊號產生器接到晶片的輸入腳位，並產生 100MHz 的方波，輸入 clk 腳位

(4) 儀器設置：將邏輯分析儀接制晶片的輸出腳位

(5) 使用訊號產生器像晶片輸入測試資料

(6) 將晶片輸出與 python 模擬出來的 output binary pattern 做比對

(7) 確認比對結果是否一致

[9] 佈局驗證結果錯誤說明

(A) DRC 驗證結果：(到時候要用 Qserver 的截圖)

所有錯誤如下圖，皆為可容許之錯誤

```
RULECHECK Latch.4.1 ..................... TOTAL Result Count = 18   (336)
RULECHECK Latch.4.2 ..................... TOTAL Result Count = 34   (368)
RULECHECK Latch.4.4 ..................... TOTAL Result Count = 0    (0)
RULECHECK Latch.4.4.pick ................ TOTAL Result Count = 5    (84)
RULECHECK Latch.4.5 ..................... TOTAL Result Count = 0    (0)
RULECHECK Latch.4.5.pick ................ TOTAL Result Count = 23   (514)
RULECHECK Latch.4.6 ..................... TOTAL Result Count = 0    (0)
RULECHECK Latch.4.6.guard ............... TOTAL Result Count = 17   (428)
RULECHECK Latch.4.7 ..................... TOTAL Result Count = 53   (662)
RULECHECK Latch.4.7.guard ............... TOTAL Result Count = 5    (84)
RULECHECK Latch.4.8__Latch.4.9__Latch.5.2 ... TOTAL Result Count = 1000 (101518)
RULECHECK Latch.4.10 .................... TOTAL Result Count = 8    (42)
RULECHECK Latch.5.1 ..................... TOTAL Result Count = 4    (72)
RULECHECK Latch.5.3 ..................... TOTAL Result Count = 0    (0)
RULECHECK Latch.5.4 ..................... TOTAL Result Count = 0    (0)
RULECHECK Latch.5.5 ..................... TOTAL Result Count = 181  (1077)
RULECHECK Latch.5.6 ..................... TOTAL Result Count = 96   (1480)
```

(LATCH_UP.rep)

```
RULECHECK sanity_1 ..... TOTAL Result Count = 16 (512)
RULECHECK sanity_2 ..... TOTAL Result Count = 0  (0)
RULECHECK sanity_3 ..... TOTAL Result Count = 0  (0)
RULECHECK sanity_4 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.1.EN1 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.1.W1 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.1.W2 ..... TOTAL Result Count = 2  (64)
RULECHECK IO5.1.R1 ..... TOTAL Result Count = 32 (1024)
RULECHECK IO5.2.1.W1.a ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.W1.b ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.L1.a ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.L1.b ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.L1.c ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.L1.d ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.L1.e ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.L1.g ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.S1 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.S2 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.EN1 .... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.S3 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.S4 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.EN2 .... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.EN3 .... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.1.S5 ..... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.2.L1.a ... TOTAL Result Count = 16 (512)
RULECHECK IO5.2.2.L1.b ... TOTAL Result Count = 0  (0)
RULECHECK IO5.2.2.L1.c ... TOTAL Result Count = 16 (512)
```

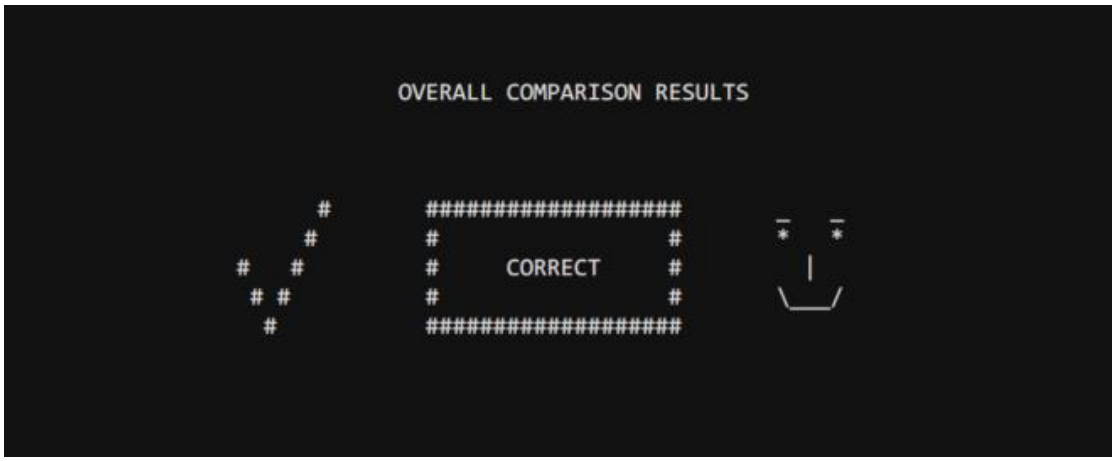
(ESD.rep)

```
RULECHECK RECOMMEND_4.14L ..... TOTAL Result Count = 41 (1743)
RULECHECK RECOMMEND_4.14M ..... TOTAL Result Count = 0  (0)
```

```
RULECHECK 4.01Z.NO_IND_OD ..... TOTAL Result Count = 0  (0)
RULECHECK 4.14Z.NO_IND_PO1 ..... TOTAL Result Count = 1  (1)
RULECHECK 4.20F.NO_IND_M1 ..... TOTAL Result Count = 1  (1)
RULECHECK 4.20G ..... TOTAL Result Count = 1  (1)
RULECHECK 4.22F.NO_IND_M2 ..... TOTAL Result Count = 1  (1)
RULECHECK 4.22G ..... TOTAL Result Count = 1  (1)
RULECHECK 4.24F.NO_IND_M3 ..... TOTAL Result Count = 1  (1)
RULECHECK 4.24G ..... TOTAL Result Count = 1  (1)
RULECHECK 4.26F.NO_IND_M4 ..... TOTAL Result Count = 0  (0)
RULECHECK 4.26G ..... TOTAL Result Count = 1  (1)
RULECHECK 4.28F.NO_IND_M5 ..... TOTAL Result Count = 0  (0)
RULECHECK 4.28G ..... TOTAL Result Count = 1  (1)
RULECHECK 4.31E.NO_IND_M6 ..... TOTAL Result Count = 0  (0)
RULECHECK 4.31F ..... TOTAL Result Count = 1  (1)
RULECHECK 4.20C ..... TOTAL Result Count = 0  (0)
RULECHECK 4.22C ..... TOTAL Result Count = 0  (0)
RULECHECK 4.24C ..... TOTAL Result Count = 0  (0)
RULECHECK 4.26C ..... TOTAL Result Count = 0  (0)
RULECHECK 4.28C ..... TOTAL Result Count = 0  (0)
```

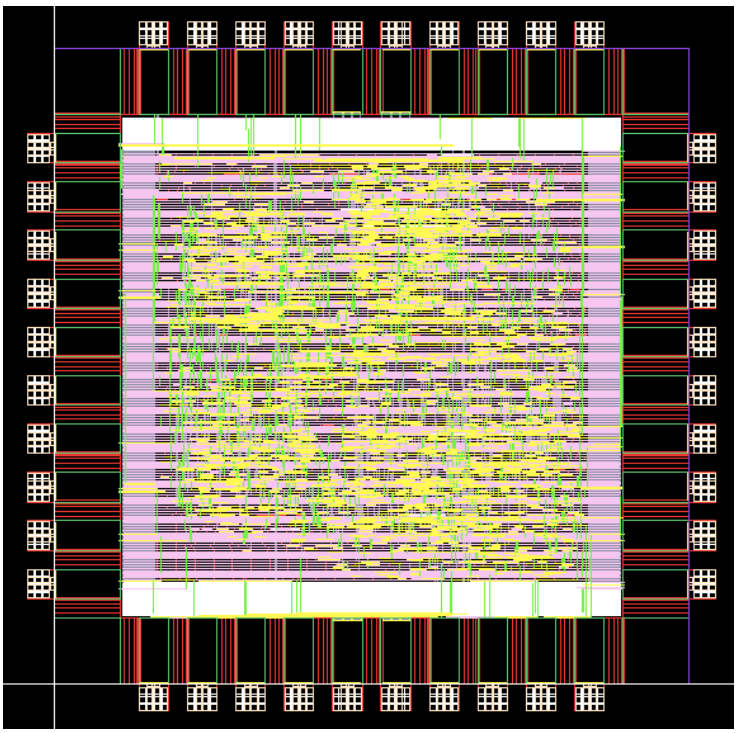
(DRC.rep)

(B) LVS 驗證結果：通過



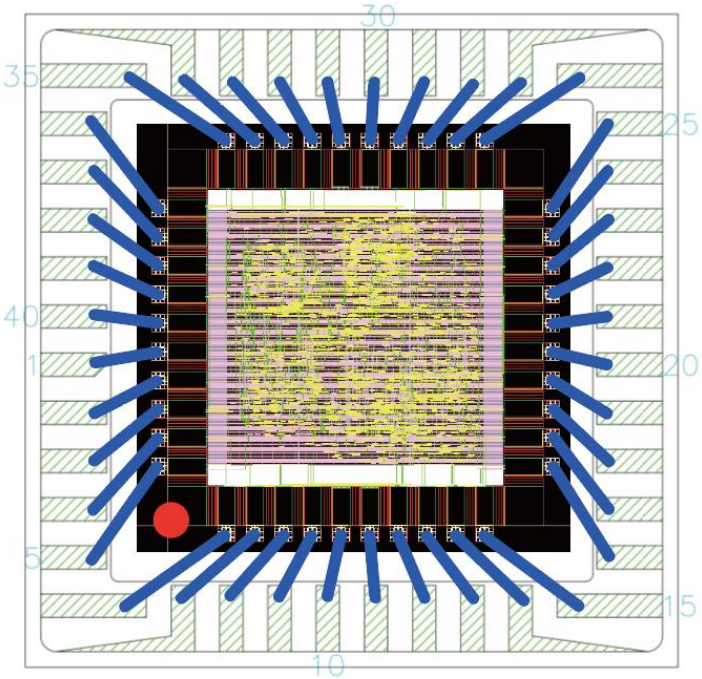
[10] 佈局平面圖

Chip Size: 15000x1500 μm^2
Power Dissipation: 32.8 mW
Max. Frequency: 100MHz



[11] 打線圖

SB40



[12] 預計規格列表

Specification	Spec.	Pre-sim(tt)	Post-sim(tt)
Supply Voltage	**1.8V**	**1.8V**	
Freqeuncy	**100MHz**	**100MHz**	
Chip size (μm²)	**1500x1500**	**1500x1500**	
Power	**-**	**32.8 mW**	
PADs	**40**	**40**	

[13] 參考文獻

[1] Ying, G. (2015). *Hardware Implementation of a 32-point Radix-2 FFT Architecture.* (Master's Thesis, Lung University, Department of Electrical and Information Technology). Retrieved from https://www.eit.lth.se/sprapport.php?uid=856

[2] 陳振模 (2004)。快速傅立葉轉換之複數乘法器設計及其系統整合（碩士論文）。取自 https://ir.nctu.edu.tw/bitstream/11536/70434/4/360104.pdf