CVSD Final Project: 5G MIMO Demodulation – ML Demodulator Design

Group 20

電子所 D11943001 廖威騏、材料所 R10527053 黃竑智

Jun. 2023

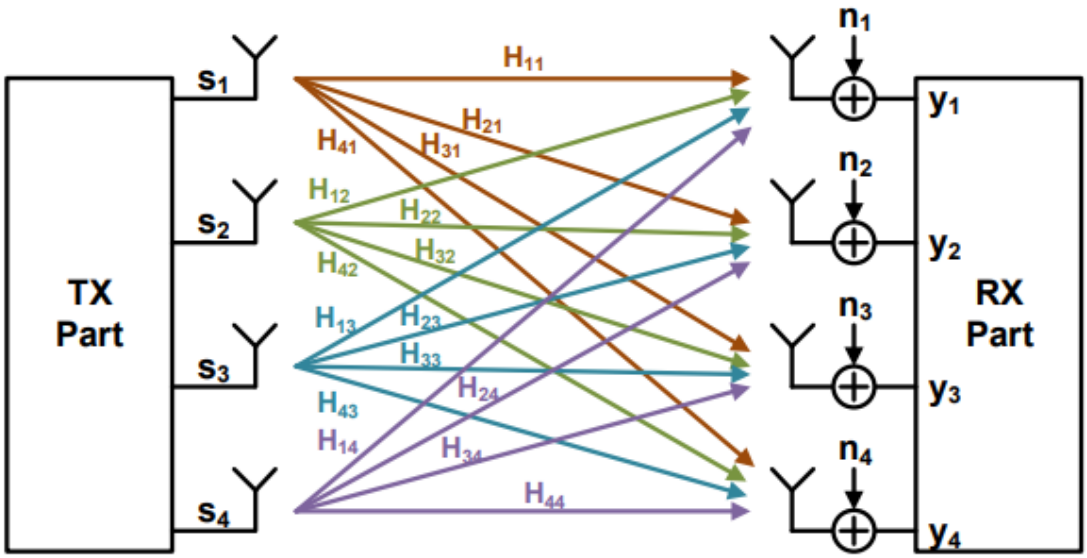
Outline

1. Algorithm
   i. ML demod. algorithm introduction
   ii. FXP setting
2. Performance
   i. The plot with Data Error Rate vs. SNR
3. HW implementation
   i. HW scheduling
   ii. HW block diagram
   iii. Area / Power / Latency report
      - Technique sharing for HW improvement

1. Algorithm

1-1. ML demod. algorithm introduction

本次 CVSD Final Project 的主題為 5G MIMO Demodulation，實踐的算法為 Maximum Likelihood (ML) Demodulation。MIMO 的全稱為 Multi-Input, Multi-Output，其示意圖如下圖：



REF：CVSD 111-2 1112_final_introduction.pdf

傳輸訊號s_1~s_4會由發送端(TX Part)經過不同的通道 H 傳輸到接收端(RX Part)。在訊號抵達 RX 端前，各自會受到不同的噪音作用(以n_1~n_4表示)。整個過程可以用矩陣寫下：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{21} & H_{22} & H_{23} & H_{24} \\ H_{31} & H_{32} & H_{33} & H_{34} \\ H_{41} & H_{42} & H_{44} & H_{44} \end{bmatrix} \begin{bmatrix} \widetilde{s_1} \\ \widetilde{s_2} \\ \widetilde{s_3} \\ \widetilde{s_4} \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{bmatrix} \Leftrightarrow y = H\tilde{s} + n$$

其中$\tilde{s}$的數值可能為$\pm\frac{1}{\sqrt{2}} \pm \frac{1}{\sqrt{2}}j$、H 矩陣的任一元素是依據常態分佈 N(0, 0.25)產生的結果、噪音 n 則滿足常態分佈 N(0, 1)。具體的產生方式是透過 matlab 的函數 awgn()、取不同 SNR(信噪功率比)而產生。

給定 y、H 的前提下，若欲求出實際傳輸的訊號$\tilde{s}$，可以依據如下關係式求出：

$$\tilde{s} = \underset{s\epsilon A}{\operatorname{argmin}} \left(||y - Hs||^2 \right)$$

其中 A 是四個傳輸信號的所有組合可能組合成的集合。

假如把 H 進行 QR 分解，可以把$y = H\tilde{s} + n$近一步簡化運算量。

$$y = H\tilde{s} + n = (QR)\tilde{s} + n$$
$$\Rightarrow Q^H y = Q^H (QR)\tilde{s} + Q^H n$$
$$\Rightarrow Q^H y = R\tilde{s} + Q^H n$$
$$\Rightarrow \hat{y} = R\tilde{s} + v$$

展開寫成矩陣型式：

$$\begin{bmatrix} \widehat{y_1} \\ \widehat{y_2} \\ \widehat{y_3} \\ \widehat{y_4} \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix} \begin{bmatrix} \widetilde{s_1} \\ \widetilde{s_2} \\ \widetilde{s_3} \\ \widetilde{s_4} \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \Leftrightarrow \hat{y} = R\tilde{s} + v$$

關於求出實際傳輸訊號$\tilde{s}$的關係式也可以改寫成：

$$\tilde{s} = \underset{s\epsilon A}{\operatorname{argmin}} \left(||\hat{y} - R\tilde{s}||^2 \right)$$

具體來說，括號內的算式可以寫成：

$$||\hat{y} - R\tilde{s}||^2 = \left(\begin{bmatrix} \widehat{y_1} \\ \widehat{y_2} \\ \widehat{y_3} \\ \widehat{y_4} \end{bmatrix} - \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix} \begin{bmatrix} \widetilde{s_1} \\ \widetilde{s_2} \\ \widetilde{s_3} \\ \widetilde{s_4} \end{bmatrix} \right)^2 = \sum_{i=1}^{4} \left\| \left[\widehat{y_i} - \sum_{i=1}^{4} R_{ij} \widetilde{s_j} \right] \right\|^2$$

在把傳輸訊號$\tilde{s}$寫成行矩陣後，假如要求出第 k 列的實部，則須計算$L(x_{k0}|y)$，此計算又稱為 soft-bit calculation、為 LLR(Log-Likelihood Ratio)的結果，滿足：

$$L(x_{k0}|y) \approx \min_{x\epsilon X_{k01}} ||y - Hs||^2 - \min_{x\epsilon X_{k00}} ||y - Hs||^2$$

由於輸入訊號$\widetilde{s_1}$~$\widetilde{s_4}$的每一個信號都可能$\pm\frac{1}{\sqrt{2}} \pm \frac{1}{\sqrt{2}}j$的其中一個，所以行矩陣 s 總共有 4*4*4*4=256 種組合。其中X_{k01}代表所有第 k 列數值的實部為負的組合(為與代碼一致，於此 k 從 0 開始算)、而X_{k00}代表所有第 k 列數值的實部為正的組合。因此例如：

$$\widehat{s_{74}} = \begin{bmatrix} \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j \\ \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j \\ -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j \\ \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j \end{bmatrix} \Rightarrow \widehat{s_{74}} \subset X_{000}, X_{011}, X_{100}, X_{110}, X_{201}, X_{211}, X_{310}, X_{320}$$

(為何$\widehat{s_{74}}$的內容是那樣的，這個等到後面會說明。)

同理，假如要算出第 k 列的虛部，則須計算$L(x_{k1}|y)$，其中$L(x_{k1}|y)$滿足：

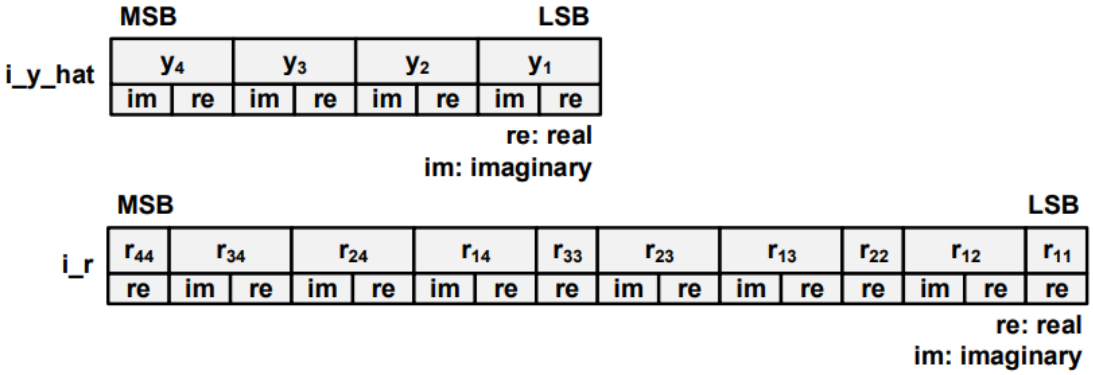
$$L(x_{k1}|y) \approx \min_{x \epsilon X_{k11}} ||y - Hs||^2 - \min_{x \epsilon X_{k10}} ||y - Hs||^2$$

相較於上面的 Soft-Bit Calculation，Hard-Bit Calculation 代表取 LLR 符號位的結果，也就是說：

$$HB\big(L(x_{kb}|y)\big) = \text{sgn}\big(L(x_{kb}|y)\big) = \begin{cases} 1, & L(x_{kb}|y) < 0 \\ 0, & L(x_{kb}|y) \geq 0 \end{cases}$$

對於任何一筆 y、R，都可以算出 LLR 以及 HB 各自 8 筆。這 8 筆就是$L(x_{00}|y)$、$L(x_{01}|y)$、$L(x_{10}|y)$、$L(x_{11}|y)$、$L(x_{20}|y)$、$L(x_{21}|y)$、$L(x_{30}|y)$、$L(x_{31}|y)$ 這 8 個 LLR 以及對應的 8 個 HB。

在題目的要求中，自第一筆輸入起，每間隔 64 週期就會輸入下一筆資料。輸入的資料為 160 bits 的 i_y_hat、320 bits 的 i_r、以及輸入是否有效的 1 bit i_trig。當 i_trig=1 時代表輸入有效。i_y_hat 自高位到低位各自表示 y 行矩陣的第 4~1 列元素的虛部與實部；i_r 自高位到低位則表示 R 矩陣中有值的數值內容、以 reverse raster-scan 的次序呈現，如下表所示：



每個部分都是 20 bits，以 S3.16 的形式表示。

外部將會用 i_rd_rdy 來表示是否要讀取輸出。當代表輸出有效的 o_rd_rdy=1 且 i_rd_rdy=1 時，testbench 將會檢驗資料正確性。i_rd_rdy 何時為 1 何時為 0 並無肯定，但每 640 週期必然會有 128 週期為 1。因此，最長將會有 1024 週期不讀取輸出。由於每 64 週期會餵入一筆資料，所以可能會有 1024/64=16 筆輸入資料的計算結果沒有被輸出。

鑑於每筆輸入都要輸出 8 筆，所以最多會需要暫存 16*8=128 筆輸出資料。在設計中是以 reg signed [7:0] LLR_pool[127:0]; 暫存資料。128 是 2 的 7 次方，所以定義讀指針的寬度要是 7 bits：

```
reg [6:0] read_pter;
```

而寫指針只需要 4 bits，因為寫入操作是一次寫 8 筆、8 是 2 的 3 次方。例如當寫入指針為 3 時，由於 3*2**3=24，就會寫入 LLR_pool[24]~ LLR_pool[31]。

```
reg             [3:0]     write_pter;
```

且經由分析，當寫指針與讀指針指向相同位置時，應令此時輸出應該是無效的、其餘都是有效，前者的狀況只可能發生於起始狀態以及讀取速度過快之時。舉凡當 o_rd_rdy=1 且 i_rd_rdy=1，就讓讀指針往下一位；而寫指針則是每 64 週期算完資料、填入 LLR_pool[i]~ LLR_pool[i+7] 後移動一位。值得一提的是，在此設計中不會發生寫指針倒追讀指針的狀況。

在硬體層面上計算$L(x_{kb}|y)$的過程分析如下。
由於輸入每 64 週期來一次，假如計算某組 y, R 的 256 種$\tilde{s}$需要花超過 64 週期，則需要非常大的資源暫存非常大的輸入，且隨輸入資料數增加而線性增加。因此應確保需要在 64 週期內算完 256 種$\tilde{s}$的結果、也就是至少每週期要算 4 種$\tilde{s}$的結果。於此，設計計數器用於計時、也用於標識當前要算哪些$\tilde{s}$。

```
reg             [6:0]     Ccnt, Ncnt;
```

本來計數 0~63 只需要 6 bits，但是以最高 bit 用於區別是否經歷過有效輸入。在第一筆有效輸入來之前，計數器恆為 0；而後計數器開始累加，當加到 63 後順理成章變成 64；但是當加到 127 後下一時刻會變成 64。計數器的結果純 0 代表此電路尚未開始有效工作。
當第一筆有效輸入來臨時，由於本設計的輸入與輸出都有使用 reg，所以需要等到下一週期才會開始計算、計數器也要在下一週期才會開始工作。在剛開始工作的週期，應有 Ccnt=0, Ncnt=1。

在計算開始前，要確定要計算哪些。因為一個週期內要計算 4 種$\hat{s}_i$對應的 8*4=32 個 LLR。所以設計當 Ccnt 為某 Cnt0 時，就計算 $\tilde{s}$={Ccnt, 2'd0}、{Ccnt, 2'd1}、{Ccnt, 2'd2}、{Ccnt, 2'd3}。對於拼湊完的 8 bits $\tilde{s}$ 來說：
$\tilde{s}$[7] 代表行矩陣 s 的$\tilde{s_4}$的虛部是正還是負(各自對應 0 以及 1)；
$\tilde{s}$[6] 代表行矩陣 s 的$\tilde{s_4}$的實部是正還是負(各自對應 0 以及 1)；
$\tilde{s}$[5] 代表行矩陣 s 的$\tilde{s_3}$的虛部是正還是負(各自對應 0 以及 1)；
…以此類推。
因此例如當 Ccnt=5 時，就計算 $\widehat{s_{20}}$、$\widehat{s_{21}}$、$\widehat{s_{22}}$、$\widehat{s_{23}}$ 的 8*4=32 個 LLR，因為把 {6'd5, 2'd0} 看成二進位數就是 00_01_01_00，對應十進位的 20。

此時回顧我們要計算的算式：

$$\left|\left|\hat{y}-R\tilde{s}\right|\right|^2=\left(\begin{bmatrix}\widehat{y_1}\\ \widehat{y_2}\\ \widehat{y_3}\\ \widehat{y_4}\end{bmatrix}-\begin{bmatrix}R_{11} & R_{12} & R_{13} & R_{14}\\ 0 & R_{22} & R_{23} & R_{24}\\ 0 & 0 & R_{33} & R_{34}\\ 0 & 0 & 0 & R_{44}\end{bmatrix}\begin{bmatrix}\widetilde{s_1}\\ \widetilde{s_2}\\ \widetilde{s_3}\\ \widetilde{s_4}\end{bmatrix}\right)^2$$

可以展開括號內的各列：

$$\begin{bmatrix}\widehat{y_1}\\ \widehat{y_2}\\ \widehat{y_3}\\ \widehat{y_4}\end{bmatrix}-\begin{bmatrix}R_{11} & R_{12} & R_{13} & R_{14}\\ 0 & R_{22} & R_{23} & R_{24}\\ 0 & 0 & R_{33} & R_{34}\\ 0 & 0 & 0 & R_{44}\end{bmatrix}\begin{bmatrix}\widetilde{s_1}\\ \widetilde{s_2}\\ \widetilde{s_3}\\ \widetilde{s_4}\end{bmatrix}=\begin{bmatrix}\widehat{y_1}\\ \widehat{y_2}\\ \widehat{y_3}\\ \widehat{y_4}\end{bmatrix}-\begin{bmatrix}R_{11}\widetilde{s_1}+R_{12}\widetilde{s_2}+R_{13}\widetilde{s_3}+R_{14}\widetilde{s_4}\\ R_{22}\widetilde{s_2}+R_{23}\widetilde{s_3}+R_{24}\widetilde{s_4}\\ R_{33}\widetilde{s_3}+R_{34}\widetilde{s_4}\\ R_{44}\widetilde{s_4}\end{bmatrix}$$

由於所求是$\left|\left|\hat{y}-R\tilde{s}\right|\right|^2$，是各列複數的 Norm 總和，所以即使同一列，都再拆成實部與虛部考慮。另

外，乍看需要極大量的乘法運算，但其實 $\tilde{s_i}$ 不外乎是 $\pm\frac{1}{\sqrt{2}}\pm\frac{1}{\sqrt{2}}j$，所以可以把$\frac{1}{\sqrt{2}}$提出到外面去、先不考慮。綜合考量複數的乘法特性，可以把$R\tilde{s}$矩陣的第一列寫成：

$$\begin{aligned}&\mathrm{Re}[R_{11}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_1}])-\mathrm{Im}[R_{11}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_1}])+\mathrm{Im}[R_{11}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_1}])-\mathrm{Re}[R_{11}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_1}])\\&+\mathrm{Re}[R_{12}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_2}])-\mathrm{Im}[R_{12}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_2}])+\mathrm{Im}[R_{12}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_2}])-\mathrm{Re}[R_{12}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_2}])\\&+\mathrm{Re}[R_{13}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_3}])-\mathrm{Im}[R_{13}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_3}])+\mathrm{Im}[R_{13}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_3}])-\mathrm{Re}[R_{13}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_3}])\\&+\mathrm{Re}[R_{14}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Im}[R_{14}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])+\mathrm{Im}[R_{14}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Re}[R_{14}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])\end{aligned}$$

同理，第二列可以寫成：

$$\begin{aligned}&\mathrm{Re}[R_{22}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_2}])-\mathrm{Im}[R_{22}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_2}])+\mathrm{Im}[R_{22}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_2}])-\mathrm{Re}[R_{22}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_2}])\\&+\mathrm{Re}[R_{23}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_3}])-\mathrm{Im}[R_{23}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_3}])+\mathrm{Im}[R_{23}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_3}])-\mathrm{Re}[R_{23}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_3}])\\&+\mathrm{Re}[R_{24}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Im}[R_{24}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])+\mathrm{Im}[R_{24}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Re}[R_{24}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])\end{aligned}$$

第三列可以寫成：

$$\begin{aligned}&\mathrm{Re}[R_{33}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_3}])-\mathrm{Im}[R_{33}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_3}])+\mathrm{Im}[R_{33}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_3}])-\mathrm{Re}[R_{33}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_3}])\\&+\mathrm{Re}[R_{34}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Im}[R_{34}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])+\mathrm{Im}[R_{34}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Re}[R_{34}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])\end{aligned}$$

第四列可以寫成：

$$\begin{aligned}&\mathrm{Re}[R_{44}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Im}[R_{44}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])+\mathrm{Im}[R_{44}]\,\mathrm{sgn}(\mathrm{Re}[\tilde{s_4}])-\mathrm{Re}[R_{44}]\,\mathrm{sgn}(\mathrm{Im}[\tilde{s_4}])\end{aligned}$$

在剔除$\mathrm{Im}[R_{11}]$ 、$\mathrm{Im}[R_{22}]$ 、$\mathrm{Im}[R_{33}]$ 、$\mathrm{Im}[R_{44}]$之後(因為他們都是 0)，可以發現乘積總共會有 32 個，所以取暫存 reg／中間變數為：

reg signed [19:0] tmp[31:0];

而 $\mathrm{Re}[\tilde{s_i}]$ 、$\mathrm{Im}[\tilde{s_i}]$ 的正負就可以從 {Ccnt, 2'b??} 得知。在計算剛開始，就會依據 {Ccnt, 2'b??} 、把 i_r 的不同部分取正/負後賦值給每個 tmp[k] 、k 的編號則是按上面列出的先後次序編號。

在賦值完 tmp[k] 後，就可以把每一列的實部與虛部各自總合起來，賦值給 Rs[m]。m=0 代表第一列的實部總和、m=1 代表第一列的虛部總和、m=2 代表第二列的實部總和、… 以此類推。Rs[m] 的寬度是 23 bits ，因為他最多收納 7 個 tmp[k] 的相加結果，最大值也不會超過 8 個 tmp[k]的相加結果，也就是左移三位的結果。

reg signed [22:0] Rs[7:0];

```
Rs[0] = ( (tmp[0]  + tmp[2])  + (tmp[3]  + tmp[6]) )  + ( (tmp[7] + tmp[10]) + tmp[11] );
Rs[1] = ( (tmp[1]  + tmp[4])  + (tmp[5]  + tmp[8]) )  + ( (tmp[9] + tmp[12]) + tmp[13] );
Rs[2] =   (tmp[14] + tmp[16]) + (tmp[17] + tmp[20])   + tmp[21];
Rs[3] =   (tmp[15] + tmp[18]) + (tmp[19] + tmp[22])   + tmp[23];
Rs[4] =   (tmp[24] + tmp[26]) + tmp[27];
Rs[5] =   (tmp[25] + tmp[28]) + tmp[29];
Rs[6] = tmp[30];
Rs[7] = tmp[31];
```

接著把 Rs[m] 乘以 $1/\sqrt{2}$。乘完之後讓相對應的 y 與之相減。例如應以 y[0][0] 減去 Rs[0]、以 y[0][1] 減去 Rs[1]、以 y[1][0] 減去 Rs[2]、以 y[1][1] 減去 Rs[3]…以此類推。
而 y[0][0]對應 y 行矩陣的第一列實部、y[1][1]對應 y 行矩陣的第二列虛部等等。

reg signed [22:0] Rs_sqrt2;

```
for (i=0 ; i<4 ; i=i+1 ) begin
    for (j=0; j<2 ; j=j+1) begin
        Rs_sqrt2    = $signed( Rs[2*i+j][ 22 -: 15 ] ) * $signed( {{ 9'b01011_0101 }} );
        y_Rs[2*i+j] = y[i][j] - $signed( Rs_sqrt2 );
    end
end
```

求得 y-Rs 的各列實部與虛部後，把實部與虛部都平方然後加總，結果記為 A。

reg signed [25:0] Squared[7:0];

```
for (i=0 ; i<8 ; i=i+1) begin
    Squared[i] = $signed( y_Rs[i][23 -: 13] ) * $signed ( y_Rs[i][23 -: 13] ) ;
end
A = ( (Squared[0] + Squared[1]) + (Squared[2] + Squared[3]) ) +
    ( (Squared[4] + Squared[5]) + (Squared[6] + Squared[7]) ) ;
```

由於每一個 $\hat{s}_i$ 都會被$\{X_{kbd}|k=0\sim4,\quad b=0,1,\quad d=0,1\}$集合中的其中 8 個子集合包含，所以每算完一個 A 就要依據 {Ccnt, 2'b??} 選出特定的 $\{X_{kbd}\}$，並直接更新這些組合的最小值。

例如當算完 $\hat{s_{128}}$ ，由於 128 以二進位表示是 10_00_00_00 ，因此就把X_{000}的最小值更新成 min(min(X_{000}), A)、把X_{010}的最小值更新成 min(min(X_{010}), A)、…以此類推。硬體代碼中的變數假設為 xkb。

reg signed [26:0] xkb[3:0][1:0][1:0];

reg signed [26:0] xkb_n[3:0][1:0][1:0];

當計算完 $\hat{s_{255}}$ 後，便可以得知 $\{X_{kbd}\}$ 內所有子集合的最小值，此時就讓 X_{kb1} 減去 X_{kb0} 即可得到 $L(x_{kb}|y)$。然而需要注意的是，因為一個週期要計算 4 種 $\hat{s}_i$ 的 A、$\hat{s}_i$ 與 $\hat{s_{i+1}}$ 總會更新到同一個 $\{X_{kbd}\}$ 的最小值。

因此在算完{Ccnt, 2'd0}的 A 時，需要更新的 8 個$\{X_{kbd}\}$子集合會先把 min(min(X_{kbd}), A) 的結果存給 surrogate[0][k][b][d]、另外 8 個$\{X_{kbd}\}$子集合的數值直接給另外 8 個 surrogate[0][k][b][d]。

reg signed [27:0] surrogate[3:0][3:0][1:0][1:0];

至此，就開始計算{Ccnt, 2'd1}的 A，重複上面的過程。在得到{Ccnt, 2'd1}的 A 後，比較對象就是 surrogate[0][k][b][d]而非X_{kbd}、而賦值對象是 surrogate[1][k][b][d]。

計算完{Ccnt, 2'd2}的 A 同理。但是當計算完{Ccnt, 2'd3}的 A、並且賦值給 surrogate[3][k][b][d]後，一般狀況會在下一個 posedge 依據 surrogate[3][k][b][d]的值更新給 xkb 並結束計算。但是當 Ccnt=6'd63 時，會讓 surrogate[3][k][b][1]減 surrogate[3][k][b][0]，結果就可以得到 8 筆 LLR 的結果、並且把$\{X_{kbd}\}$的內容重置成{1'b0, {26{1'b1}}}。

由於輸出緩衝 LLR_pool 每 64 週期才會有其中的 8 筆需要更新內容，所以當要更新 LLR_pool 內容時，讓 EN_LLR_pool[write_pter]=1，posedge 時才依據哪個 EN_LLR_pool[write_pter]為 1 來更新指定的 8 個 LLR_pool。

reg EN_LLR_pool[15:0];

1-2. FXP setting

在計算的過程中需要同時注意位寬不能太寬以避免延遲與硬體資源消耗太大、也不能太窄損失過多的精準度。在第一步決定 tmp[i] 為何時 tmp[i] 位寬同R_{ij}相同是{S3.16}。

接著 tmp[i]相加時形成 Rs[i]時因為 Rs[i]最多由 7 個 tmp[i]組成，故讓 Rs 為{S6.16}。

計算 Rs 除以根號 2 時，取 Rs 為{S6.8}、根號 2 為{1.8}，相乘得到{S7.16}的 Rs_sqrt2。這樣的取法是因為讓乘積結果的小數點後有 16 位，能與{S3.16}的 y 運算。

y-Rs 後為{S7.16}，下一步要平方再相加總合。一樣為了位寬避免過大，取 y-Rs 寬{S7.5}，平方就是{S7.5}*{S7.5}={S14.10}。

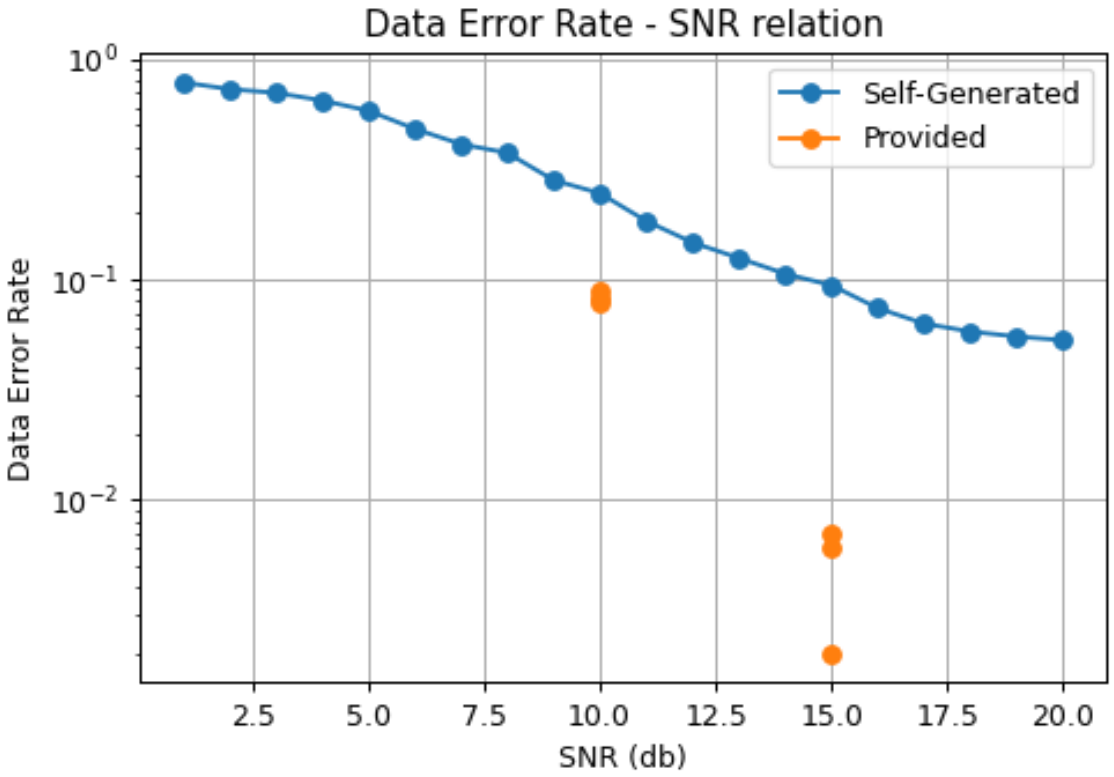
8 個$(y - Rs)^2$便為{S17.10}、寬 28 bits。後面的 xkb、xkb_n、surrogate 也相同設定。

而在最後計算 A 的時候，因應輸出是 S3.4，所以 surrogate 相減後會右移 6 位，以得到計算結果。

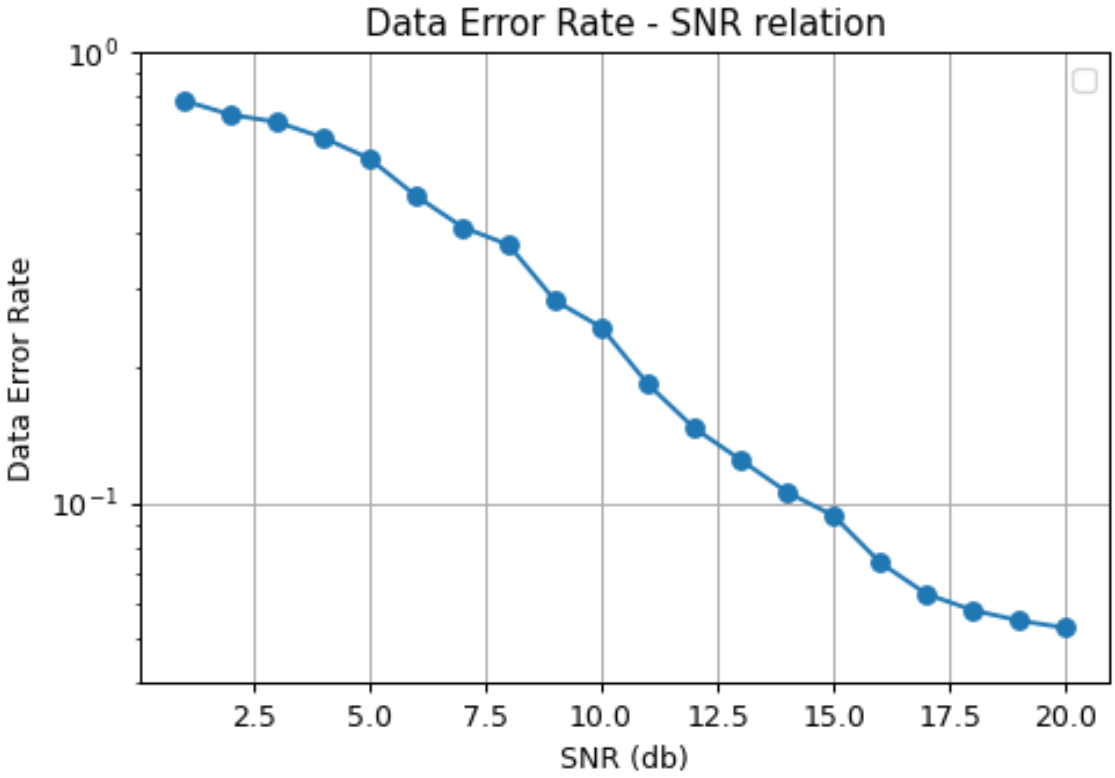
2. Performance

i. The plot with Data Error Rate vs. SNR

或許是對 Noise 的生成方式或/和 H 的生成方式有誤解，自己生成的 pattern 在同一 SNR 分貝下總是有較多筆有 LLR 的正負狀態不等於 HB 的數值的現象。因此以硬體實現算法時，Error Rate 也一定會更大。以自行生成的 pattern 依據自己的硬體設計去算出的 Error Rate 以藍線表示、另將「依助教提供的 pattern 以自己的硬體設計去算出的 Error Rate」 以橘典表示。



以下是純藍線的 Data Error Rate vs. SNR 圖。

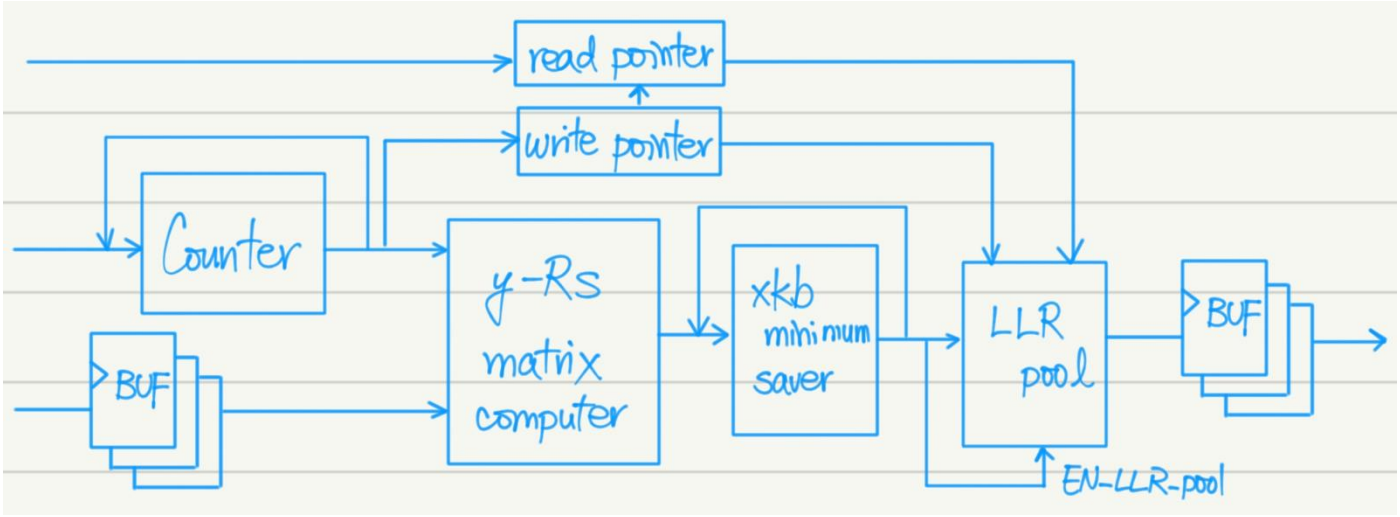


3. HW implementation

i. HW scheduling (list chronologically with raster-scan order)

演算法考察	軟體驗證	編寫 testbench	編寫 core RTL 與 debug	RTL 優化	電路合成與合成後模擬
黃竑智 廖威騏	黃竑智	黃竑智 廖威騏	黃竑智	廖威騏	廖威騏
APR 與 APR 後模擬	**功耗分析**	**報告撰寫(前期)**	**報告撰寫(後期)**	**Pattern 生成與測試**	
廖威騏	廖威騏	黃竑智	廖威騏	黃竑智	

ii. HW block diagram



iii. Area / Power / Latency report

- **Technique sharing for HW improvement**

(1) RTL Optimization

在 Power 方面，我們對於 xkb、LLR_pool、i_y_hat、i_r 皆有使用 clock gating 以降低 dynamic power。

(2) Gate-level Synthesis

在合成方面我們做了一點 trick，我們將 synthesis constraint 的 CLOCK_PERIOD 稍微放寬為 22.0ns，以合成較小之 area。但在 gate-sim 和 post-sim 用 cycle time 21.0ns 跑都還是可以通過且無任何 timing violation。

(3) APR

我們 core 的 Aspect Ratio (H/W)設定為 1，Core Utilization 設定為 0.7。在 Floorplan 的部分，由於此次 spec 規定之 VDD 與 VSS Power Ring 寬度只設為 2um 且只須做一組，因此 core margin 不需像 Lab 預留那麼寬，core 到 4 個 boundary 的距離只需留 7μm 即可，如此可大幅降低 die area。Halo 在 Top、Bottom、Left、Right 皆設定為 30μm。在整個 APR 過程中皆沒有發生 DRC 或 LVS 的 violation，並且 WNS 也皆≥ 0，因此無特別需要修正的地方。