Error Control Coding - Decoding of Reed-Muller Codes

電通所　Q361142221　蘇沛錦

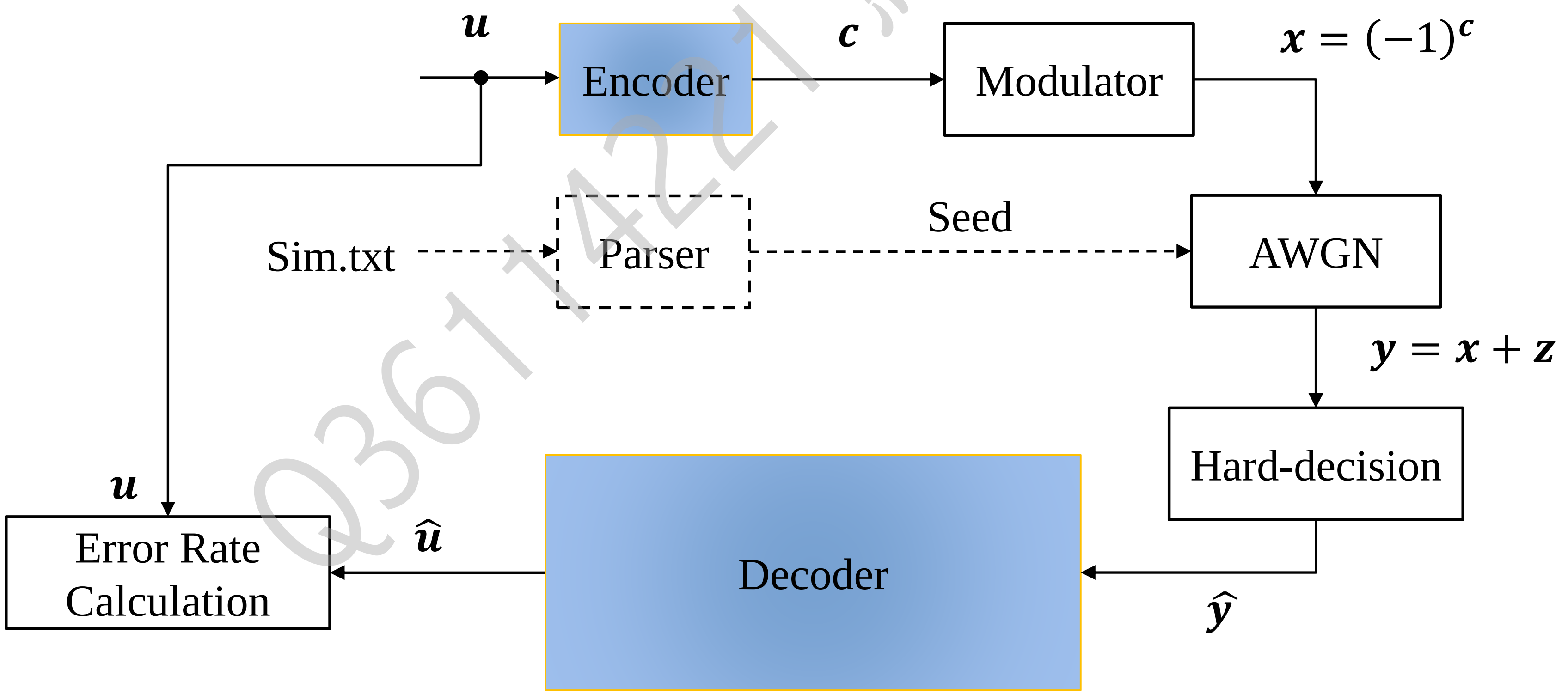
2023/06/19 (Mon.)

Outline

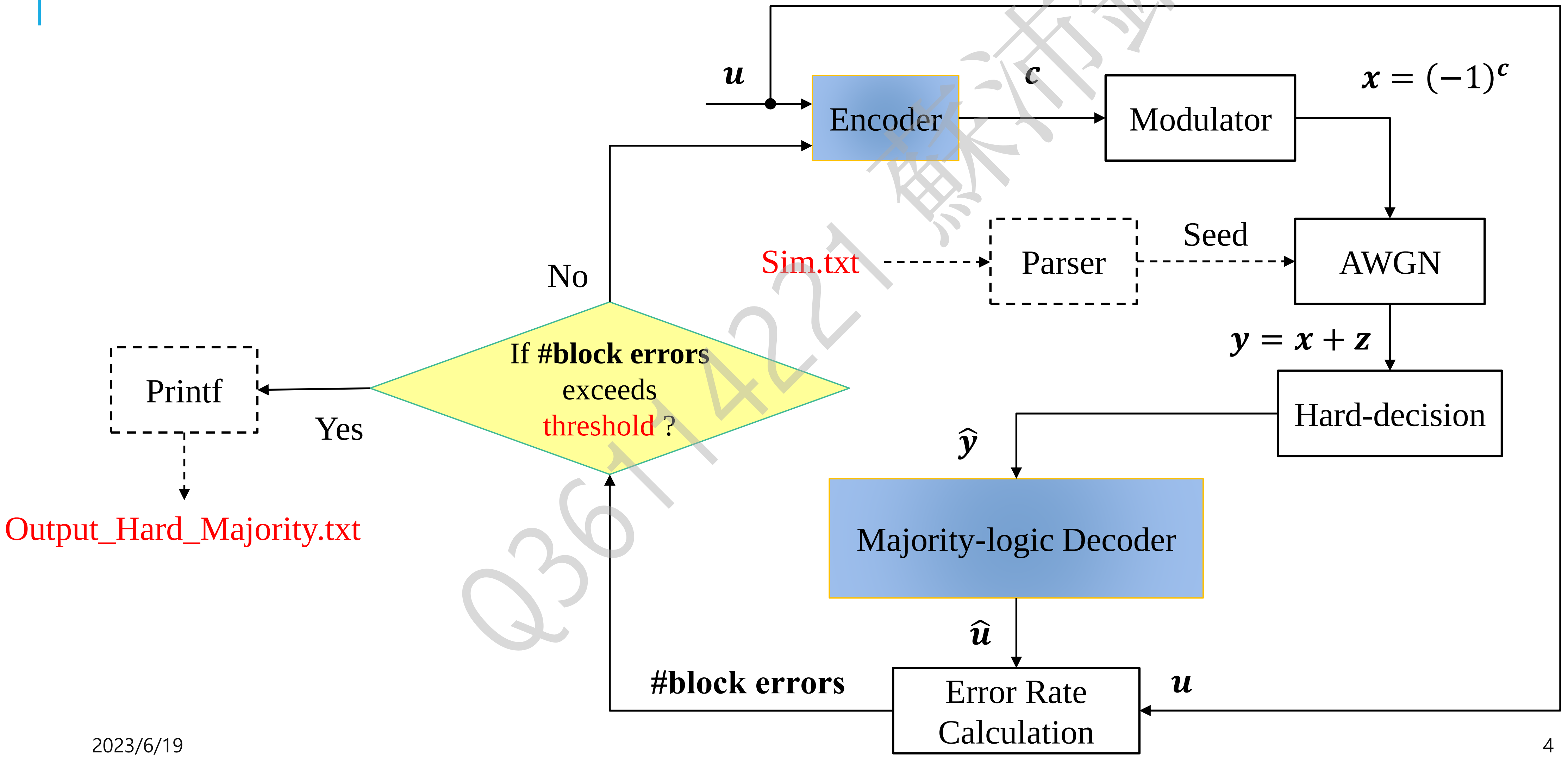
1. 系統架構圖
2. 程式流程解釋
3. 參數設定
4. 模擬數據
5. 模擬圖

系統架構圖

- Consider RM(1, 5) with code length $n = 2^5 = 32$ and dimension $K = 6$.
- Block Diagram:

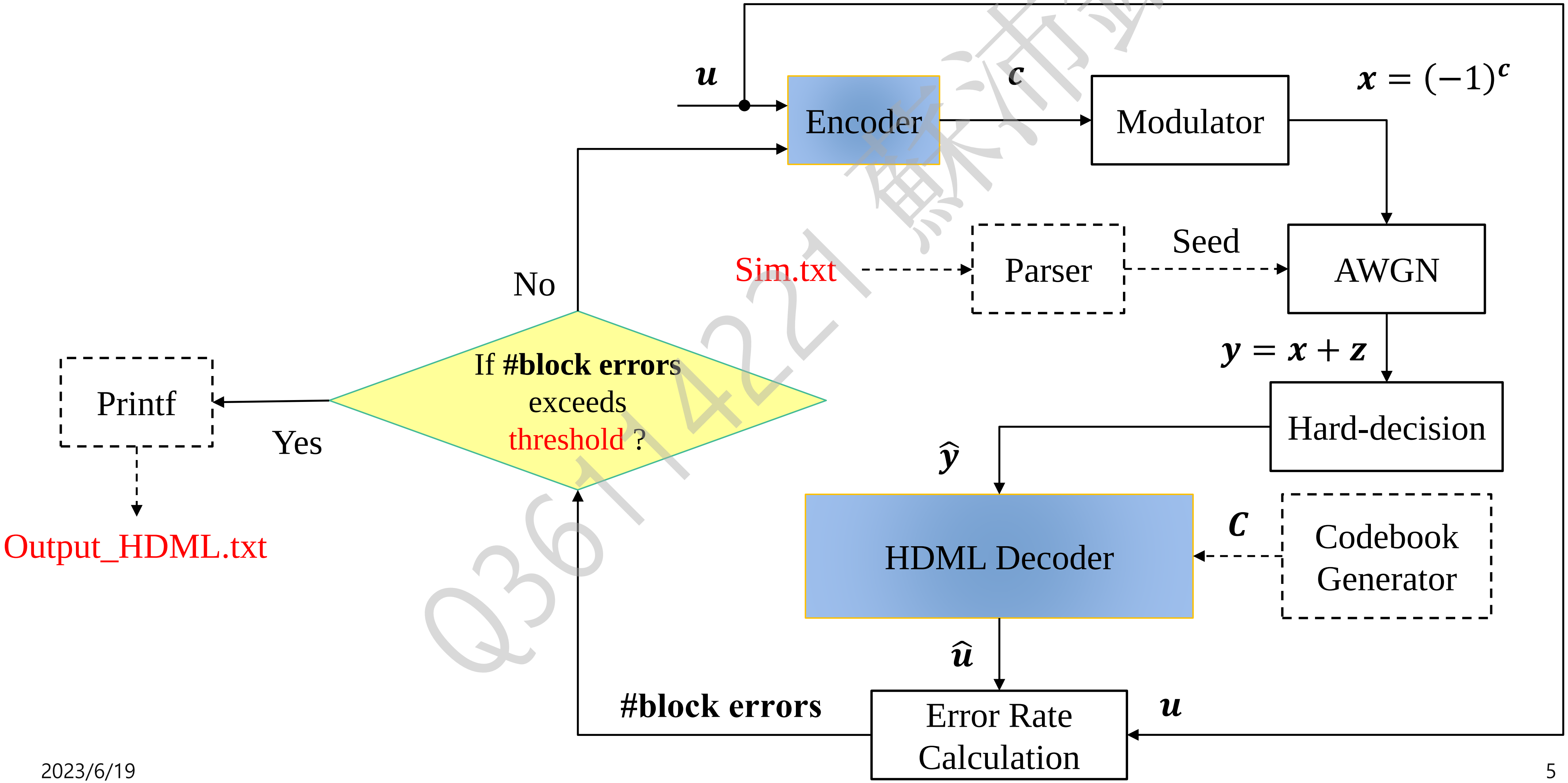


系統架構圖



u
Encoder
c
Modulator
$x = (-1)^c$
Seed
AWGN
Sim.txt
Parser
No
$y = x + z$
Hard-decision
Printf
If #block errors
exceeds
threshold ?
Yes
$\hat{y}$
Output_Hard_Majority.txt
Majority-logic Decoder
$\hat{u}$
#block errors
Error Rate
Calculation
u

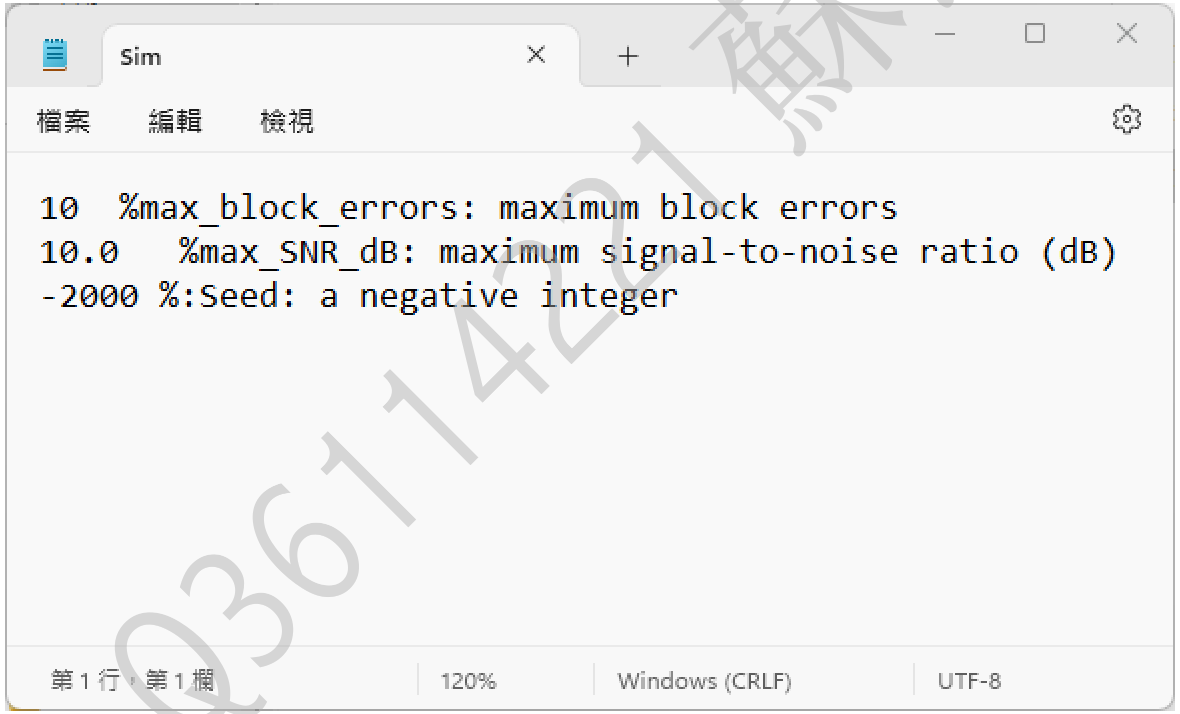
系統架構圖



$\boldsymbol{u}$
Encoder
$\boldsymbol{c}$
Modulator
$\boldsymbol{x} = (-1)^{\boldsymbol{c}}$
Sim.txt
Parser
Seed
AWGN
$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{z}$
Hard-decision
No
If #block errors exceeds threshold ?
Printf
Yes
Output_HDML.txt
$\hat{\boldsymbol{y}}$
HDML Decoder
$\boldsymbol{C}$
Codebook Generator
$\hat{\boldsymbol{u}}$
#block errors
Error Rate Calculation
$\boldsymbol{u}$

Decoding of Reed-Muller Codes

- Input Sim.txt:




```
10  %max_block_errors: maximum block errors
10.0   %max_SNR_dB: maximum signal-to-noise ratio (dB)
-2000 %:Seed: a negative integer
```

Decoding of Reed-Muller Codes

- Output “Output_Hard_Majority.txt”:

Output_Hard_Majority

檔案 編輯 檢視

```
SNR(dB): 5.0
No. decoded blocks: 238, No. decoded block errors: 10

SNR(dB): 6.0
No. decoded blocks: 1240, No. decoded block errors: 10

SNR(dB): 7.0
No. decoded blocks: 2796, No. decoded block errors: 10

SNR(dB): 8.0
No. decoded blocks: 13732, No. decoded block errors: 10

SNR(dB): 9.0
No. decoded blocks: 437925, No. decoded block errors: 10

SNR(dB): 10.0
No. decoded blocks: 9908596, No. decoded block errors: 10

Runtime = 16389.640 seconds
```

第 1 行，第 1 欄 120% Windows (CRLF) UTF-8

Decoding of Reed-Muller Codes

- Use the recursion

$$u_{l+6} = u_{l+1} \oplus u_l, \qquad \text{for } l \geq 0$$

with the initial conditions

$$u_0 = 1, u_1 = u_2 = u_3 = u_4 = u_5 = 0$$

to generate $k = 6$ information bits.

- The generated sequence is $100000100001 \cdots$ with period 63.

Decoding of Reed-Muller Codes

Additive white Gaussian noise (AWGN) channel:

- For a binary code of rate R with BPSK modulation, the noise variance σ^2 is given by

$$\sigma^2 = \left(2R\frac{E_b}{N_0}\right)^{-1}$$

  where E_b/N_0 is the bit signal-to-noise ratio (SNR).

- The code rate is $R = 6/32$ here.

Decoding of Reed-Muller Codes

- Use the following pseudo code:

```
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

main()
{
    ...
   long *idum;
   idum = (long *)malloc(sizeof(long));
   *idum = SEED;  //SEED must be a negative integer.
   ...
   ...
}
```

Decoding of Reed-Muller Codes

- Use **normal()** to output two independent normal random variables, n_1 and n_2.

$$
\begin{array}{l}
\texttt{normal}(n_1, n_2, \sigma) \\
\{ \\
\quad \texttt{do}\{ \\
\qquad x_1 = \texttt{ran1(idum)}; \\
\qquad x_2 = \texttt{ran1(idum)}; \\
\qquad x_1 = 2x_1 - 1; \\
\qquad x_2 = 2x_2 - 1; \\
\qquad s = x_1^2 + x_2^2; \\
\quad \}\ \texttt{while}\ (s \geq 1.0) \\
\quad n_1 = \sigma x_1 \sqrt{-2 \ln s / s}; \\
\quad n_2 = \sigma x_2 \sqrt{-2 \ln s / s}; \\
\}
\end{array}
$$

Decoding of Reed-Muller Codes

- Use **ran1()** to generate a random variable uniformly distributed in the interval $(0, 1)$.

```
ran1(long *idum)
{
    int j;
    long k;
    static long iy=0;
    static long iv[NTAB];
    double temp;
    if (*idum <= 0 || !iy){
       if (-(*idum) < 1) *idum=1;
       else *idum = -(*idum);
       for (j=NTAB+7;j>=0;j--){
           k=(*idum)/IQ;
           *idum=IA*(*idum-k*IQ)-IR*k;
           if (*idum < 0) *idum += IM;
           if (j < NTAB) iv[j] = *idum;
       }
       iy=iv[0];
    }
    k=(*idum)/IQ;
    *idum=IA*(*idum-k*IQ)-IR*k;
    if (*idum < 0) *idum += IM;
    j=iy/NDIV;
    iy=iv[j];
    iv[j] = *idum;
    if ((temp=AM*iy) > RNMX) return RNMX;
    else return temp;
}
```

程式流程解釋 (Encoder)

- Let us consider RM(1, 5) with the generator matrix given by

$$
\boldsymbol{G}_{RM}(1,5)=\begin{array}{c}\begin{bmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}\end{array}
\begin{array}{c}\boldsymbol{x}_0\\ \boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{x}_3\\ \boldsymbol{x}_4\\ \boldsymbol{x}_5\end{array}
$$

- Its dimension is $K=\binom{5}{0}+\binom{5}{1}=6$ and code length is $n=2^5=32$.
- $\boldsymbol{u}=(u_0,u_1,u_2,u_3,u_4,u_5)$ is the information vector.

程式流程解釋 (Encoder)

- $\boldsymbol{c} = (c_0, c_1, \dots, c_{31})$ is the codeword.

$$\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G}_{RM}$$

$$= [u_0 \quad u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5] \cdot \begin{bmatrix} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{bmatrix}$$

$$= u_0\boldsymbol{x}_0 + u_1\boldsymbol{x}_1 + u_2\boldsymbol{x}_2 + u_3\boldsymbol{x}_3 + u_4\boldsymbol{x}_4 + u_5\boldsymbol{x}_5$$

- $(u_1, u_2, u_3, u_4, u_5)$ are the information bits corresponding to vectors of degree 1 and u_0 is the information bit corresponding to the all-one vector.

程式流程解釋 (Encoder)

- The codeword $\boldsymbol{c}$ can be written as follows:

$c_0 = u_0;$
$c_1 = u_0 + u_1;$
$c_2 = u_0 + u_2;$
$c_3 = u_0 + u_1 + u_2;$
$c_4 = u_0 + u_3;$
$c_5 = u_0 + u_1 + u_3;$
$c_6 = u_0 + u_2 + u_3;$
$c_7 = u_0 + u_1 + u_2 + u_3;$
$c_8 = u_0 + u_4;$
$c_9 = u_0 + u_1 + u_4;$

$c_{10} = u_0 + u_2 + u_4;$
$c_{11} = u_0 + u_1 + u_2 + u_4;$
$c_{12} = u_0 + u_3 + u_4;$
$c_{13} = u_0 + u_1 + u_3 + u_4;$
$c_{14} = u_0 + u_2 + u_3 + u_4;$
$c_{15} = u_0 + u_1 + u_2 + u_3 + u_4;$
$c_{16} = u_0 + u_5;$
$c_{17} = u_0 + u_1 + u_5;$
$c_{18} = u_0 + u_2 + u_5;$
$c_{19} = u_0 + u_1 + u_2 + u_5;$

$c_{20} = u_0 + u_3 + u_5;$
$c_{21} = u_0 + u_1 + u_3 + u_5;$
$c_{22} = u_0 + u_2 + u_3 + u_5;$
$c_{23} = u_0 + u_1 + u_2 + u_3 + u_5;$
$c_{24} = u_0 + u_4 + u_5;$
$c_{25} = u_0 + u_1 + u_4 + u_5;$
$c_{26} = u_0 + u_2 + u_4 + u_5;$
$c_{27} = u_0 + u_1 + u_2 + u_4 + u_5;$
$c_{28} = u_0 + u_3 + u_4 + u_5;$
$c_{29} = u_0 + u_1 + u_3 + u_4 + u_5;$
$c_{30} = u_0 + u_2 + u_3 + u_4 + u_5;$
$c_{31} = u_0 + u_1 + u_2 + u_3 + u_4 + u_5.$

程式流程 (Encoder)

```c
/* Encoder */
int *Encoder(int *u)
{
    int *c = (int *)calloc(power(2, m), sizeof(int));    // Codeword

    c[0] = u[0];
    c[1] = u[0]^u[1];
    c[2] = u[0]^u[2];
    c[3] = u[0]^u[1]^u[2];
    c[4] = u[0]^u[3];
    c[5] = u[0]^u[1]^u[3];
    c[6] = u[0]^u[2]^u[3];
    c[7] = u[0]^u[1]^u[2]^u[3];
    c[8] = u[0]^u[4];
    c[9] = u[0]^u[1]^u[4];

    c[10] = u[0]^u[2]^u[4];
    c[11] = u[0]^u[1]^u[2]^u[4];
    c[12] = u[0]^u[3]^u[4];
    c[13] = u[0]^u[1]^u[3]^u[4];
    c[14] = u[0]^u[2]^u[3]^u[4];
    c[15] = u[0]^u[1]^u[2]^u[3]^u[4];
    c[16] = u[0]^u[5];
    c[17] = u[0]^u[1]^u[5];
    c[18] = u[0]^u[2]^u[5];
    c[19] = u[0]^u[1]^u[2]^u[5];

```

程式流程 (Encoder)

```c
    c[20] = u[0]^u[3]^u[5];
    c[21] = u[0]^u[1]^u[3]^u[5];
    c[22] = u[0]^u[2]^u[3]^u[5];
    c[23] = u[0]^u[1]^u[2]^u[3]^u[5];
    c[24] = u[0]^u[4]^u[5];
    c[25] = u[0]^u[1]^u[4]^u[5];
    c[26] = u[0]^u[2]^u[4]^u[5];
    c[27] = u[0]^u[1]^u[2]^u[4]^u[5];
    c[28] = u[0]^u[3]^u[4]^u[5];
    c[29] = u[0]^u[1]^u[3]^u[4]^u[5];

    c[30] = u[0]^u[2]^u[3]^u[4]^u[5];
    c[31] = u[0]^u[1]^u[2]^u[3]^u[4]^u[5];

    return c;
}
```

```c
    while (num_blocks_error < max_block_errors)
    {
        num_tx_blocks = num_tx_blocks + 1;
        printf("\nSNR(dB): %.1lf, No. decoded blocks: %d\n", SNR_dB, num_tx_blocks);

        /* Information Bits Generator */
        u = Info_Bits_Gen((r+m));

        /* Encoder */
        c = Encoder(u);

        for (int i = 0; i < power(2, (m - 1)); i++)…

        /* Decoder (Majority-Logic Decoding Algorithm) */
        u_est = Decoder(y);

        /* BLER measurement */
        for (int i = 0; i < (r+m); i++)…

        /* 釋放先前配置的記憶體區塊 */
        free(u);
        free(c);
        free(u_est);
    }
```

程式流程解釋 (Majority-logic Decoder)

Based on the above 32 equations, we have the relationship with respect to u_1 by

$$
\begin{aligned}
u_1 &= c_0 + c_1 & \qquad u_1 &= c_{16} + c_{17} \\
&= c_2 + c_3 & &= c_{18} + c_{19} \\
&= c_4 + c_5 & &= c_{20} + c_{21} \\
&= c_6 + c_7 & &= c_{22} + c_{23} \\
&= c_8 + c_9 & &= c_{24} + c_{25} \\
&= c_{10} + c_{11} & &= c_{26} + c_{27} \\
&= c_{12} + c_{13} & &= c_{28} + c_{29} \\
&= c_{14} + c_{15}; & &= c_{30} + c_{31}.
\end{aligned}
$$

- Let $\boldsymbol{y} = (y_0, y_1, \dots, y_{31})$ be the hard-decision codeword.
- k denotes the index of vote.
- $\hat{u}_{j,k}$ represents the estimation of u_j in the kth vote.

程式流程解釋 (Majority-logic Decoder)

- **Step 1.** Decode the information bits corresponding to vectors of degree 1, i.e., $(u_1, u_2, u_3, u_4, u_5)$.

$$
\begin{aligned}
\hat{u}_{1,0} &= y_0 + y_1; & \hat{u}_{1,8} &= y_{16} + y_{17};\\
\hat{u}_{1,1} &= y_2 + y_3; & \hat{u}_{1,9} &= y_{18} + y_{19};\\
\hat{u}_{1,2} &= y_4 + y_5; & \hat{u}_{1,10} &= y_{20} + y_{21};\\
\hat{u}_{1,3} &= y_6 + y_7; & \hat{u}_{1,11} &= y_{22} + y_{23};\\
\hat{u}_{1,4} &= y_8 + y_9; & \hat{u}_{1,12} &= y_{24} + y_{25};\\
\hat{u}_{1,5} &= y_{10} + y_{11}; & \hat{u}_{1,13} &= y_{26} + y_{27};\\
\hat{u}_{1,6} &= y_{12} + y_{13}; & \hat{u}_{1,14} &= y_{28} + y_{29};\\
\hat{u}_{1,7} &= y_{14} + y_{15}; & \hat{u}_{1,15} &= y_{30} + y_{31}.
\end{aligned}
$$

The estimation of u_1 is obtained by majority voting based on the above sixteen equations.

程式流程解釋 (Majority-logic Decoder)

- **Step 1.**(cont'd) Similarly, the estimation of u_2 is obtained by majority voting according to the following sixteen equations.

$$
\begin{aligned}
\hat{u}_{2,0} &= y_0 + y_2; & \hat{u}_{2,8} &= y_{16} + y_{18};\\
\hat{u}_{2,1} &= y_1 + y_3; & \hat{u}_{2,9} &= y_{17} + y_{19};\\
\hat{u}_{2,2} &= y_4 + y_6; & \hat{u}_{2,10} &= y_{20} + y_{22};\\
\hat{u}_{2,3} &= y_5 + y_7; & \hat{u}_{2,11} &= y_{21} + y_{23};\\
\hat{u}_{2,4} &= y_8 + c_{10}; & \hat{u}_{2,12} &= y_{24} + y_{26};\\
\hat{u}_{2,5} &= y_9 + y_{11}; & \hat{u}_{2,13} &= y_{25} + y_{27};\\
\hat{u}_{2,6} &= y_{12} + y_{14}; & \hat{u}_{2,14} &= y_{28} + y_{30};\\
\hat{u}_{2,7} &= y_{13} + y_{15}; & \hat{u}_{2,15} &= y_{29} + y_{31}.
\end{aligned}
$$

程式流程解釋 (Majority-logic Decoder)

- **Step 1.**(cont'd) The estimation of u_3 is obtained by majority voting according to the following sixteen equations.

$$\hat{u}_{3,0} = y_0 + y_4;\qquad \hat{u}_{3,8} = y_{16} + y_{20};$$
$$\hat{u}_{3,1} = y_1 + y_5;\qquad \hat{u}_{3,9} = y_{17} + y_{21};$$
$$\hat{u}_{3,2} = y_2 + y_6;\qquad \hat{u}_{3,10} = y_{18} + y_{22};$$
$$\hat{u}_{3,3} = y_3 + y_7;\qquad \hat{u}_{3,11} = y_{19} + y_{23};$$
$$\hat{u}_{3,4} = y_8 + y_{12};\qquad \hat{u}_{3,12} = y_{24} + y_{28};$$
$$\hat{u}_{3,5} = y_9 + y_{13};\qquad \hat{u}_{3,13} = y_{25} + y_{29};$$
$$\hat{u}_{3,6} = y_{10} + y_{14};\qquad \hat{u}_{3,14} = y_{26} + y_{30};$$
$$\hat{u}_{3,7} = y_{11} + y_{15};\qquad \hat{u}_{3,15} = y_{27} + y_{31}.$$

程式流程解釋 (Majority-logic Decoder)

- **Step 1.**(cont'd) The estimation of u_4 is obtained by majority voting according to the following sixteen equations.

$$\hat{u}_{4,0} = y_0 + y_8; \qquad \hat{u}_{4,8} = y_{16} + y_{24};$$
$$\hat{u}_{4,1} = y_1 + y_9; \qquad \hat{u}_{4,9} = y_{17} + y_{25};$$
$$\hat{u}_{4,2} = y_2 + y_{10}; \qquad \hat{u}_{4,10} = y_{18} + y_{26};$$
$$\hat{u}_{4,3} = y_3 + y_{11}; \qquad \hat{u}_{4,11} = y_{19} + y_{27};$$
$$\hat{u}_{4,4} = y_4 + y_{12}; \qquad \hat{u}_{4,12} = y_{20} + y_{28};$$
$$\hat{u}_{4,5} = y_5 + y_{13}; \qquad \hat{u}_{4,13} = y_{21} + y_{29};$$
$$\hat{u}_{4,6} = y_6 + y_{14}; \qquad \hat{u}_{4,14} = y_{22} + y_{30};$$
$$\hat{u}_{4,7} = y_7 + y_{15}; \qquad \hat{u}_{4,15} = y_{23} + y_{31}.$$

程式流程解釋 (Majority-logic Decoder)

- **Step 1.**(cont'd) The estimation of u_5 is obtained by majority voting according to the following sixteen equations.

$$\begin{aligned}
\hat{u}_{5,0} &= y_0 + y_{16}; & \hat{u}_{5,8} &= y_8 + y_{24};\\
\hat{u}_{5,1} &= y_1 + y_{17}; & \hat{u}_{5,9} &= y_9 + y_{25};\\
\hat{u}_{5,2} &= y_2 + y_{18}; & \hat{u}_{5,10} &= y_{10} + y_{26};\\
\hat{u}_{5,3} &= y_3 + y_{19}; & \hat{u}_{5,11} &= y_{11} + y_{27};\\
\hat{u}_{5,4} &= y_4 + y_{20}; & \hat{u}_{5,12} &= y_{12} + y_{28};\\
\hat{u}_{5,5} &= y_5 + y_{21}; & \hat{u}_{5,13} &= y_{13} + y_{29};\\
\hat{u}_{5,6} &= y_6 + y_{22}; & \hat{u}_{5,14} &= y_{14} + y_{30};\\
\hat{u}_{5,7} &= y_7 + y_{23}; & \hat{u}_{5,15} &= y_{15} + y_{31}.
\end{aligned}$$

程式流程解釋 (Majority-logic Decoder)

- **Step 2.** To decode u_0, remove the estimated values of u_1, u_2, u_3, u_4 and u_5, from the original received codeword $\boldsymbol{y} \to \boldsymbol{y}'$:

$$\begin{aligned}
y'_0 &= y_0 = \hat{u}_{0,0}; \\
y'_1 &= y_1 + \hat{u}_1 = \hat{u}_{0,1}; \\
y'_2 &= y_2 + \hat{u}_2 = \hat{u}_{0,2}; \\
y'_3 &= y_3 + \hat{u}_1 + \hat{u}_2 = \hat{u}_{0,3}; \\
y'_4 &= y_4 + \hat{u}_3 = \hat{u}_{0,4}; \\
y'_5 &= y_5 + \hat{u}_1 + \hat{u}_3 = \hat{u}_{0,5}; \\
y'_6 &= y_6 + \hat{u}_2 + \hat{u}_3 = \hat{u}_{0,6}; \\
y'_7 &= y_7 + \hat{u}_1 + \hat{u}_2 + \hat{u}_3 = \hat{u}_{0,7}; \\
y'_8 &= y_8 + \hat{u}_4 = \hat{u}_{0,8}; \\
y'_9 &= y_9 + \hat{u}_1 + \hat{u}_4 = \hat{u}_{0,9};
\end{aligned}$$

$$\begin{aligned}
y'_{10} &= y_{10} + \hat{u}_2 + \hat{u}_4 = \hat{u}_{0,10}; \\
y'_{11} &= y_{11} + \hat{u}_1 + \hat{u}_2 + \hat{u}_4 = \hat{u}_{0,11}; \\
y'_{12} &= y_{12} + \hat{u}_3 + \hat{u}_4 = \hat{u}_{0,12}; \\
y'_{13} &= y_{13} + \hat{u}_1 + \hat{u}_3 + \hat{u}_4 = \hat{u}_{0,13}; \\
y'_{14} &= y_{14} + \hat{u}_2 + \hat{u}_3 + \hat{u}_4 = \hat{u}_{0,14}; \\
y'_{15} &= y_{15} + \hat{u}_1 + \hat{u}_2 + \hat{u}_3 + \hat{u}_4 = \hat{u}_{0,15}; \\
y'_{16} &= y_{16} + \hat{u}_5 = \hat{u}_{0,16}; \\
y'_{17} &= y_{17} + \hat{u}_1 + \hat{u}_5 = \hat{u}_{0,17}; \\
y'_{18} &= y_{18} + \hat{u}_2 + \hat{u}_5 = \hat{u}_{0,18}; \\
y'_{19} &= y_{19} + \hat{u}_1 + \hat{u}_2 + \hat{u}_5 = \hat{u}_{0,19};
\end{aligned}$$

程式流程解釋 (Majority-logic Decoder)

$$y'_{20} = y_{20} + \hat{u}_3 + \hat{u}_5 = \hat{u}_{0,20};$$
$$y'_{21} = y_{21} + \hat{u}_1 + \hat{u}_3 + \hat{u}_5 = \hat{u}_{0,21};$$
$$y'_{22} = y_{22} + \hat{u}_2 + \hat{u}_3 + \hat{u}_5 = \hat{u}_{0,22};$$
$$y'_{23} = y_{23} + \hat{u}_1 + \hat{u}_2 + \hat{u}_3 + \hat{u}_5 = \hat{u}_{0,23};$$
$$y'_{24} = y_{24} + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,24};$$
$$y'_{25} = y_{25} + \hat{u}_1 + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,25};$$
$$y'_{26} = y_{26} + \hat{u}_2 + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,26};$$
$$y'_{27} = y_{27} + \hat{u}_1 + \hat{u}_2 + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,27};$$
$$y'_{28} = y_{28} + \hat{u}_3 + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,28};$$
$$y'_{29} = y_{29} + \hat{u}_1 + \hat{u}_3 + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,29};$$
$$y'_{30} = y_{30} + \hat{u}_2 + \hat{u}_3 + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,30};$$
$$y'_{31} = y_{31} + \hat{u}_1 + \hat{u}_2 + \hat{u}_3 + \hat{u}_4 + \hat{u}_5 = \hat{u}_{0,31}.$$

- **Step 3.** The estimation of u_0 is determined through majority voting based on the above 32 equations.

程式流程 (Majority-logic Decoder)

```c
/* Decoder (Majority-Logic Decoding Algorithm) */
int *Decoder(int *y)
{
    /* 票數統計變數 */
    int vote_0_count = 0;
    int vote_1_count = 0;

    /* 配置儲存票數的記憶體空間 */
    int *v = (int *)calloc(power(2, m), sizeof(int));
    /* 配置記憶體空間儲存解碼器估計出來的位元資料 (u_est) */
    int *u_est = (int *)calloc((r+m), sizeof(int));     // the estimated information bits

    /* Obtain the estimation of 'u1' by majority voting */
    for (int i = 0; i < power(2, (m - 1)); i++)⋯

    if (vote_1_count >= vote_0_count)⋯
    else⋯

    /* Similarly, obtain the estimation of 'u2' by majority voting */
    vote_0_count = 0;    // 票數歸零
    vote_1_count = 0;    // 票數歸零
    for (int i = 0; i < power(2, (m - 2)); i++)⋯

    if (vote_1_count >= vote_0_count)⋯
    else⋯

```

程式流程 (Majority-logic Decoder)

```
/* Obtain the estimation of 'u1' by majority voting */
for (int i = 0; i < power(2, (m - 1)); i++)
{
    v[i] = y[2*i]^y[2*i + 1];
    if (v[i] == 0)
    {
        vote_0_count = vote_0_count + 1;
    }
    else
    {
        vote_1_count = vote_1_count + 1;
    }
}

if (vote_1_count >= vote_0_count)
{
    u_est[1] = 1;
}
else
{
    u_est[1] = 0;
}
```

程式流程 (Majority-logic Decoder)

```c
/* Similarly, obtain the estimation of 'u2' by majority voting */
vote_0_count = 0;   // 票數歸零
vote_1_count = 0;   // 票數歸零
for (int i = 0; i < power(2, (m - 2)); i++)
{
    v[2*i] = y[4*i]^y[4*i + 2];
    v[2*i + 1] = y[4*i + 1]^y[4*i + 3];
    if (v[2*i] == 0)
    {
        vote_0_count = vote_0_count + 1;
    }
    else
    {
        vote_1_count = vote_1_count + 1;
    }

    if (v[2*i + 1] == 0)
    {
        vote_0_count = vote_0_count + 1;
    }
    else
    {
        vote_1_count = vote_1_count + 1;
    }
}

if (vote_1_count >= vote_0_count) ···
else ···
```

程式流程 (Majority-logic Decoder)

```c
/* Similarly, obtain the estimation of 'u3' by majority voting */
vote_0_count = 0;   // 票數歸零
vote_1_count = 0;   // 票數歸零
for (int i = 0; i < power(2, (m - 3)); i++)
{
    v[4*i] = y[8*i]^y[8*i + 4];
    v[4*i + 1] = y[8*i + 1]^y[8*i + 5];
    v[4*i + 2] = y[8*i + 2]^y[8*i + 6];
    v[4*i + 3] = y[8*i + 3]^y[8*i + 7];
    if (v[4*i] == 0)
    {
        vote_0_count = vote_0_count + 1;
    }
    else
    {
        vote_1_count = vote_1_count + 1;
    }

    if (v[4*i + 1] == 0) ···
    else ···

    if (v[4*i + 2] == 0) ···
    else ···

    if (v[4*i + 3] == 0) ···
    else ···
}

if (vote_1_count >= vote_0_count) ···
else ···
```

程式流程 (Majority-logic Decoder)

```c
/* Similarly, obtain the estimation of 'u4' by majority voting */
vote_0_count = 0;   // 票數歸零
vote_1_count = 0;   // 票數歸零
for (int i = 0; i < power(2, (m - 2)); i++)     // the 0 ~ 7th vote for the estimated 'u4'
{
    v[i] = y[i]^y[i + 8];
    if (v[i] == 0)
    {
        vote_0_count = vote_0_count + 1;
    }
    else
    {
        vote_1_count = vote_1_count + 1;
    }
}

for (int i = 0; i < power(2, (m - 2)); i++)     // the 8 ~ 15th vote for the estimated 'u4' …

if (vote_1_count >= vote_0_count)
{
    u_est[4] = 1;
}
else
{
    u_est[4] = 0;
}
```

程式流程 (Majority-logic Decoder)

```c
/* Similarly, obtain the estimation of 'u5' by majority voting */
vote_0_count = 0;   // 票數歸零
vote_1_count = 0;   // 票數歸零
for (int i = 0; i < power(2, (m - 1)); i++)
{
    v[i] = y[i]^y[i + 16];
    if (v[i] == 0)
    {
        vote_0_count = vote_0_count + 1;
    }
    else
    {
        vote_1_count = vote_1_count + 1;
    }
}

if (vote_1_count >= vote_0_count)
{
    u_est[5] = 1;
}
else
{
    u_est[5] = 0;
}
```

程式流程 (Majority-logic Decoder)

```c
/* To decode 'u0', remove the estimated values of 'u1', u2, u3, u4, and u5' from "y" */
v[0] = y[0];
v[1] = y[1]^u_est[1];
v[2] = y[2]^u_est[2];
v[3] = y[3]^u_est[1]^u_est[2];
v[4] = y[4]^u_est[3];
v[5] = y[5]^u_est[1]^u_est[3];
v[6] = y[6]^u_est[2]^u_est[3];
v[7] = y[7]^u_est[1]^u_est[2]^u_est[3];
v[8] = y[8]^u_est[4];
v[9] = y[9]^u_est[1]^u_est[4];

v[10] = y[10]^u_est[2]^u_est[4];
v[11] = y[11]^u_est[1]^u_est[2]^u_est[4];
v[12] = y[12]^u_est[3]^u_est[4];
v[13] = y[13]^u_est[1]^u_est[3]^u_est[4];
v[14] = y[14]^u_est[2]^u_est[3]^u_est[4];
v[15] = y[15]^u_est[1]^u_est[2]^u_est[3]^u_est[4];
v[16] = y[16]^u_est[5];
v[17] = y[17]^u_est[1]^u_est[5];
v[18] = y[18]^u_est[2]^u_est[5];
v[19] = y[19]^u_est[1]^u_est[2]^u_est[5];

v[20] = y[20]^u_est[3]^u_est[5];
v[21] = y[21]^u_est[1]^u_est[3]^u_est[5];
v[22] = y[22]^u_est[2]^u_est[3]^u_est[5];
v[23] = y[23]^u_est[1]^u_est[2]^u_est[3]^u_est[5];
v[24] = y[24]^u_est[4]^u_est[5];
v[25] = y[25]^u_est[1]^u_est[4]^u_est[5];
v[26] = y[26]^u_est[2]^u_est[4]^u_est[5];
v[27] = y[27]^u_est[1]^u_est[2]^u_est[4]^u_est[5];
v[28] = y[28]^u_est[3]^u_est[4]^u_est[5];
v[29] = y[29]^u_est[1]^u_est[3]^u_est[4]^u_est[5];
```

程式流程 (Majority-logic Decoder)

```c

    v[30] = y[30]^u_est[2]^u_est[3]^u_est[4]^u_est[5];
    v[31] = y[31]^u_est[1]^u_est[2]^u_est[3]^u_est[4]^u_est[5];

    /* Similarly, obtain the estimation of 'u0' by majority voting */
    vote_0_count = 0;   // 票數歸零
    vote_1_count = 0;   // 票數歸零
    for (int i = 0; i < power(2, m); i++)
    {
        if (v[i] == 0)
        {
            vote_0_count = vote_0_count + 1;
        }
        else
        {
            vote_1_count = vote_1_count + 1;
        }
    }

    if (vote_1_count >= vote_0_count)
    {
        u_est[0] = 1;
    }
    else
    {
        u_est[0] = 0;
    }

    /* 釋放先前配置的記憶體區塊 */
    free(v);

    return u_est;
}

```

程式流程解釋 (HDML Decoder)

- Maximum-Likelihood (ML) decoding rule:
  - Choose $\hat{x}$ as the codeword which maximizes

$$P(\boldsymbol{y}|\boldsymbol{x}). \qquad \text{(likelihood function)}$$

- Maximizing $P(\boldsymbol{y}|\boldsymbol{x})$ is equivalent to minimizing $d_H(\boldsymbol{x},\boldsymbol{y})$, which is called the minimum distance (MD) decoding.

- **Hard-decision maximum likelihood (HDML)** decoding algorithm
  - Consider $\mathrm{RM}_2(1,5)$, let $\boldsymbol{C} = \{\boldsymbol{0}, \boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, (\boldsymbol{x}_0 + \boldsymbol{x}_1), \dots, (\boldsymbol{x}_0 + \boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3 + \boldsymbol{x}_4 + \boldsymbol{x}_5)\}$
  - D_l denotes the distance between the hard-decision codeword $\hat{\boldsymbol{y}}$ and the valid codeword $\boldsymbol{c}_l$ in $\boldsymbol{C}$, i.e., $\boldsymbol{c}_l \in \boldsymbol{C} = \{\boldsymbol{c}_0, \boldsymbol{c}_1, \dots, \boldsymbol{c}_{M-1}\}$ where $M = 2^{(5+1)} = 64$ and $\boldsymbol{c}_0 = (0, 0, \dots, 0)$.

程式流程解釋 (HDML Decoder)

- The code $\boldsymbol{C}$ of $\mathrm{RM}_2(1,5)$ is given by

$$
\boldsymbol{C}=\begin{Bmatrix}\boldsymbol{c}_0\\ \boldsymbol{c}_1\\ \boldsymbol{c}_2\\ \boldsymbol{c}_3\\ \boldsymbol{c}_4\\ \boldsymbol{c}_5\\ \boldsymbol{c}_6\\ \vdots\\ \boldsymbol{c}_{62}\\ \boldsymbol{c}_{63}\end{Bmatrix}=\begin{Bmatrix}
\text{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0}\\
\text{1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1}\\
\text{0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1}\\
\text{0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1}\\
\text{0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1}\\
\text{0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1}\\
\text{0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1}\\
\vdots\\
\text{1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1}\\
\text{0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0}
\end{Bmatrix}=\begin{Bmatrix}\boldsymbol{0}\\ \boldsymbol{x}_0\\ \boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{x}_3\\ \boldsymbol{x}_4\\ \boldsymbol{x}_5\\ \vdots\\ \boldsymbol{x}_1+\boldsymbol{x}_2+\boldsymbol{x}_3+\boldsymbol{x}_4+\boldsymbol{x}_5\\ \boldsymbol{x}_0+\boldsymbol{x}_1+\boldsymbol{x}_2+\boldsymbol{x}_3+\boldsymbol{x}_4+\boldsymbol{x}_5\end{Bmatrix}
$$

程式流程解釋 (HDML Decoder)

- **Step 1.** Compute the hard-decision of $\boldsymbol{y}$ and obtain the information $\hat{\boldsymbol{y}}$, i.e.,

$$\boldsymbol{y} \xrightarrow{\text{Hard-decision}} \hat{\boldsymbol{y}}.$$

- **Step 2.** For $0 \leq l \leq 63$, calculate the distance between the hard-decision codeword $\hat{\boldsymbol{y}}$ and the valid codeword $\boldsymbol{c}_l$ in $\boldsymbol{C} = \{\boldsymbol{c}_0, \boldsymbol{c}_1, \dots, \boldsymbol{c}_{63}\}$.

$$D_l \leftarrow d_H(\boldsymbol{x}, \hat{\boldsymbol{y}}), \forall \boldsymbol{c}_l \in \boldsymbol{C} = \{\boldsymbol{c}_0, \boldsymbol{c}_1, \dots, \boldsymbol{c}_{63}\}.$$

- **Step 3.** Find l' with the minimum value of D_l, i.e.,

$$l' = \operatorname*{argmin}_{0 \leq l \leq 63} D_l\,.$$

  Decide $\boldsymbol{c}_{l'}$ as the estimated codeword $\hat{\boldsymbol{c}}$.

$$\hat{\boldsymbol{c}} \leftarrow \boldsymbol{c}_{l'}.$$

- **Step 4.** Output the estimated information vector $\hat{\boldsymbol{u}}$ corresponding to the estimated codeword $\boldsymbol{c}_{l'}$.

$$\hat{\boldsymbol{u}} \leftarrow \boldsymbol{u}_{l'}, \boldsymbol{c}_{l'} = \boldsymbol{u}_{l'} \cdot \boldsymbol{G}_{RM}(1,5).$$

程式流程 (HDML Decoder)

- 產生 $RM_2(1,5)$ 的碼簿 (Codebook)

```c
/* 讀取模擬參數 */
int max_block_errors;   // Number of decoded bits
double max_SNR_dB;      // Bit signal-to-noise ratio (dB) [SNR(dB) 模擬最大上限]
long SEED;              // Seed: a negative integer

FILE *fp;
char filename[30] = "Sim.txt";
char buffer[50];
int buffer_flag;

// Input Sim.txt:
if((fp=fopen(filename,"r"))==NULL){ …
else{ …

/* 產生 RM(1,5) 的 Codebook */
int **Codebook = Generate_Codebook();

// Prerequisite for AWGN channel
long *idum;
idum = (long *)malloc(sizeof(long));
*idum = SEED;   // SEED must be a negative integer
double SNR_dB;  // SNR: bit signal-to-noise ratio (dB)
double SNR;     // Bit signal-to-noise ratio
double sigma;   // standard deviation of the noise
int SNR_dB_Idx;

/* 配置統計不同 SNR(dB) 條件下的 BLER 所需之記憶體空間 */
int num_tx_blocks = 0;      // No. decoded blocks
int num_blocks_error = 0;   // No. decoded block errors
```

```c
/* 產生 RM(1,5) 的 Codebook */
int **Generate_Codebook(void)
{
    int **Codebook = (int **)calloc(power(2,(r+m)), sizeof(int *));
    for (int i = 0; i < power(2,(r+m)); i++)
    {
        Codebook[i] = (int*)calloc(power(2, m), sizeof(int));
    }
    int *temp = (int *)calloc((r+m), sizeof(int));

    for (int i = 0; i < power(2,(r+m)); i++)
    {
        for (int j = 0; j < (r+m); j++)
        {
            temp[j] = (i/power(2, j))%2;
        }

        Codebook[i][0] = temp[0];
        Codebook[i][1] = temp[0]^temp[1];
        Codebook[i][2] = temp[0]^temp[2];
        Codebook[i][3] = temp[0]^temp[1]^temp[2];
        Codebook[i][4] = temp[0]^temp[3];
        Codebook[i][5] = temp[0]^temp[1]^temp[3];
        Codebook[i][6] = temp[0]^temp[2]^temp[3];
        Codebook[i][7] = temp[0]^temp[1]^temp[2]^temp[3];
        Codebook[i][8] = temp[0]^temp[4];
        Codebook[i][9] = temp[0]^temp[1]^temp[4];

```

```c
        Codebook[i][10] = temp[0]^temp[2]^temp[4];
        Codebook[i][11] = temp[0]^temp[1]^temp[2]^temp[4];
        Codebook[i][12] = temp[0]^temp[3]^temp[4];
        Codebook[i][13] = temp[0]^temp[1]^temp[3]^temp[4];
        Codebook[i][14] = temp[0]^temp[2]^temp[3]^temp[4];
        Codebook[i][15] = temp[0]^temp[1]^temp[2]^temp[3]^temp[4];
        Codebook[i][16] = temp[0]^temp[5];
        Codebook[i][17] = temp[0]^temp[1]^temp[5];
        Codebook[i][18] = temp[0]^temp[2]^temp[5];
        Codebook[i][19] = temp[0]^temp[1]^temp[2]^temp[5];

        Codebook[i][20] = temp[0]^temp[3]^temp[5];
        Codebook[i][21] = temp[0]^temp[1]^temp[3]^temp[5];
        Codebook[i][22] = temp[0]^temp[2]^temp[3]^temp[5];
        Codebook[i][23] = temp[0]^temp[1]^temp[2]^temp[3]^temp[5];
        Codebook[i][24] = temp[0]^temp[4]^temp[5];
        Codebook[i][25] = temp[0]^temp[1]^temp[4]^temp[5];
        Codebook[i][26] = temp[0]^temp[2]^temp[4]^temp[5];
        Codebook[i][27] = temp[0]^temp[1]^temp[2]^temp[4]^temp[5];
        Codebook[i][28] = temp[0]^temp[3]^temp[4]^temp[5];
        Codebook[i][29] = temp[0]^temp[1]^temp[3]^temp[4]^temp[5];

        Codebook[i][30] = temp[0]^temp[2]^temp[3]^temp[4]^temp[5];
        Codebook[i][31] = temp[0]^temp[1]^temp[2]^temp[3]^temp[4]^temp[5];
    }

    /* 釋放先前配置的記憶體區塊 */
    free(temp);
    return Codebook;
}
```

C:\Users\P.J. SU\Desktop\ECC_Project_RM\RM_CodeBook_20230617.exe

```
The codebook of RM(1,5):
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0
1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1
0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0
1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0
1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1
0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1
1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1
1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0
0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0
0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1
1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0
0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0
0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1
1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0
0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0
1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0
1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1
0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0
```

程式流程 (HDML Decoder)

```c
while (num_blocks_error < max_block_errors)
{
    num_tx_blocks = num_tx_blocks + 1;
    printf("\nSNR(dB): %.1lf, No. decoded blocks: %d\n", SNR_dB, num_tx_blocks);

    /* Information Bits Generator */
    u = Info_Bits_Gen((r+m));

    /* Encoder */
    c = Encoder(u);

    for (int i = 0; i < power(2, (m - 1)); i++)
    {
        /* Modulator */
        x[0] = Modulator(c[2*i]);
        x[1] = Modulator(c[2*i + 1]);

        /* AWGN channel */
        normal(&z[0], &z[1], sigma, idum);

        /* Hard decision */
        y[2*i] = ((x[0] + z[0]) >= 0.0) ? 0 : 1;
        y[2*i + 1] = ((x[1] + z[1]) >= 0.0) ? 0 : 1;
    }

    /* HDML Decoder (Hard-decision Maximum Likelihood) */
    u_est = HDML_Decoder( y, Codebook);

    /* BER measurement */
    for (int i = 0; i < (r+m); i++)···
```

程式流程 (HDML Decoder)

- HDML Decoder 副函式

```c
/* HDML Decoder (Hard-decision Maximum Likelihood) */
int *HDML_Decoder(int *y, int **Codebook)
{
    /* 配置記憶體空間，用來儲存解碼器估計出來的位元資料 (u_est) */
    int *u_est = (int *)calloc((r+m), sizeof(int));      // the estimated information bits

    /* 配置記憶體空間，用來儲存接收碼字(y)和碼簿內所有合法碼字之間的距離 */
    int *d = (int *)calloc(power(2,(r+m)), sizeof(int));

    /* 計算接收碼字(y)和碼簿內所有合法碼字之間的距離 */
    for (int i = 0; i < power(2,(r+m)); i++)
    {
        for (int j = 0; j < power(2,m); j++)
        {
            if (y[j] != Codebook[i][j])
            {
                d[i] ++;
            }
        }
    }

```

程式流程 (HDML Decoder)

```c

    /* 找出最短距離的合法碼字 */
    int codeword_Idx = 0;
    int reg_temp = d[codeword_Idx];
    for (int i = 1; i < power(2,(r+m)); i++)
    {
        if (d[i] < reg_temp)
        {
            reg_temp = d[i];
            codeword_Idx = i;
        }
    }

    /* 解碼器估計出來的位元資料 (u_est) */
    for (int i = 0; i < (r+m); i++)
    {
        u_est[i] = (codeword_Idx/power(2, i))%2;
    }

    /* 釋放先前配置的記憶體區塊 */
    free(d);

    return u_est;
}
```

模擬參數設定

- 解碼演算法:
  1. Majority-Logic Decoding Algorithm
  2. Hard-Decision Maximum Likelihood (HDML) Decoding Algorithm
- SNR (dB): 5, 6, 7, 8, 9, 10 (Note: 因時間關係 HDML 只畫到 SNR = 9 dB)
- Seed (a negative integer): -2000
- maximum number of block errors = 50

模擬數據

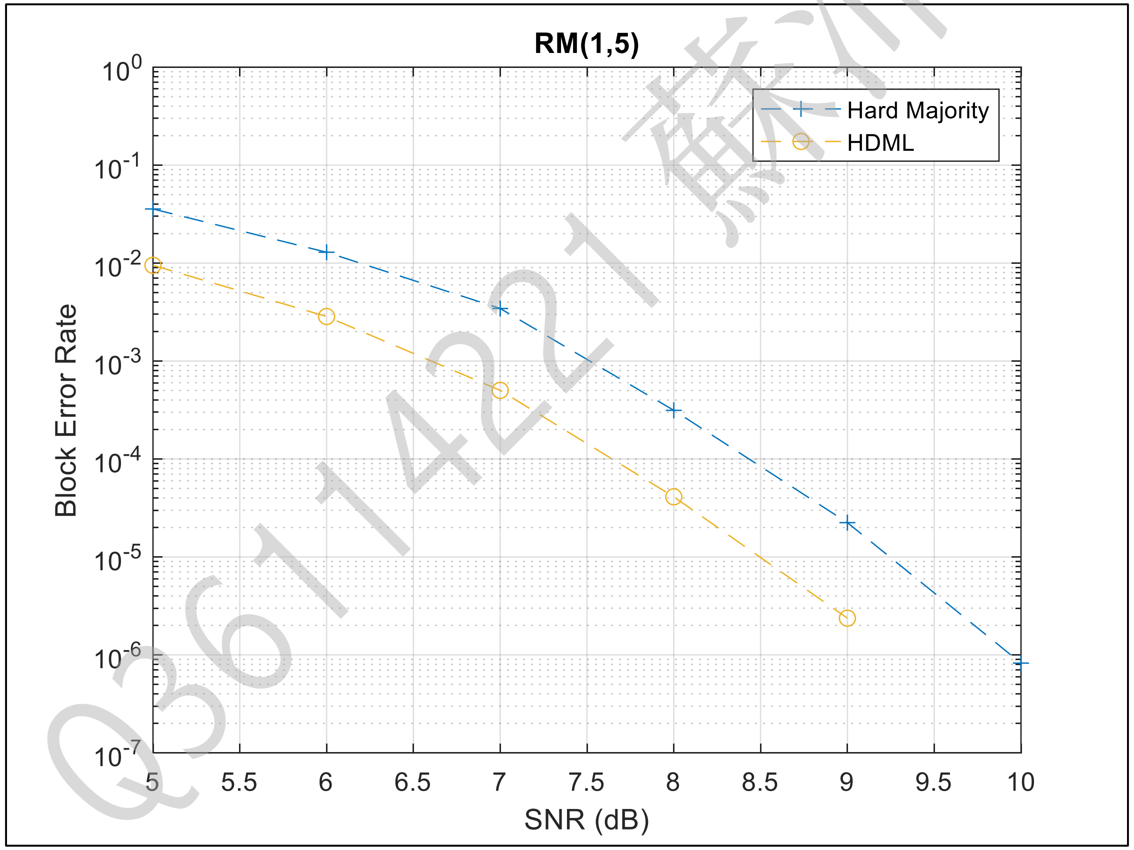
1. Majority-Logic Decoding Algorithm

SNR (dB)	5	6	7	8	9	10
No. decoded block errors	50					
No. decoded blocks	1399	3880	14541	159235	2230083	60703678
Block error rate (BLER)	3.57E-02	1.29E-02	3.44E-03	3.14E-04	2.24E-05	8.24E-07

2. Hard-Decision Maximum Likelihood (HDML) Decoding Algorithm

SNR (dB)	5	6	7	8	9
No. decoded block errors	50				
No. decoded blocks	5268	17552	99561	1217628	21072680
Block error rate (BLER)	9.49E-03	2.85E-03	5.02E-04	4.11E-05	2.37E-06

模擬效能圖



BLER performance of RM(1, 5) decoded with the HDML and the majority decoding algorithms.

模擬效能圖 (Ref. 老師上課講義)

Error Cotrol Coding - Reed-Muller Codes

Simulation Results

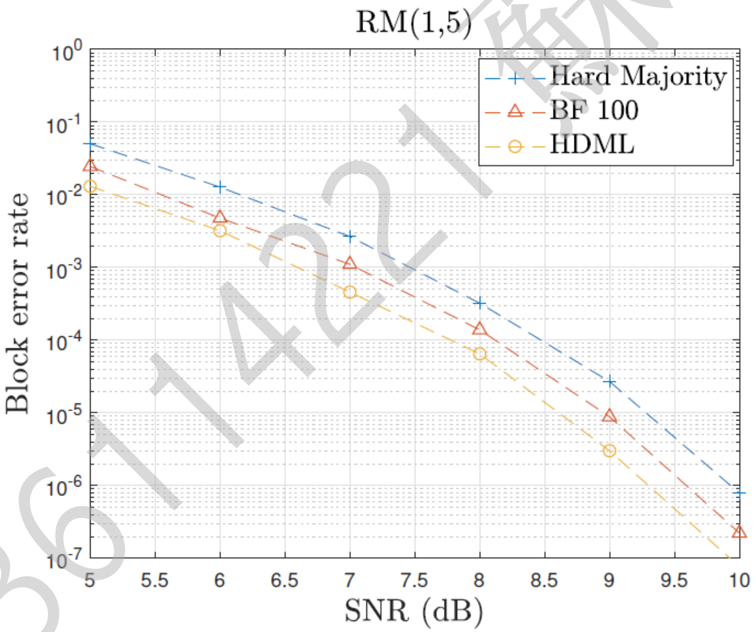


Figure 4: BLER performance of RM$(1,5)$ decoded with the BF, the hard-decision maximum likelihood (HDML), and the majority decoding algorithms.

不同 block errors 個數的比較

模擬參數設定

- 解碼採用 Majority-Logic Decoding Algorithm
- SNR (dB): 5, 6, 7, 8, 9, 10
- Seed (a negative integer): 2000
- maximum number of block errors = 1, 3, 5, 10, 30, 50

不同 block errors 個數的比較

模擬數據 (Majority-Logic Decoding Algorithm)

- maximum number of block errors = 1

SNR (dB)	5	6	7	8	9	10
No. decoded block errors	1					
No. decoded blocks	91	5	313	15655	21502	2902143
Block error rate (BLER)	1.10E-02	2.00E-01	3.19E-03	6.39E-05	4.65E-05	3.45E-07

- maximum number of block errors = 3

SNR (dB)	5	6	7	8	9	10
No. decoded block errors	3					
No. decoded blocks	116	293	1034	8309	71367	3510270
Block error rate (BLER)	2.59E-02	1.02E-02	2.90E-03	3.61E-04	4.20E-05	8.55E-07

不同 block errors 個數的比較

模擬數據 (Majority-Logic Decoding Algorithm)

- maximum number of block errors = 5

SNR (dB)	5	6	7	8	9	10
No. decoded block errors	5					
No. decoded blocks	156	275	2828	5193	312304	5175901
Block error rate (BLER)	3.21E-02	1.82E-02	1.77E-03	9.63E-04	1.60E-05	9.66E-07

- maximum number of block errors = 10

SNR (dB)	5	6	7	8	9	10
No. decoded block errors	10					
No. decoded blocks	238	1240	2796	13732	437925	9908596
Block error rate (BLER)	4.20E-02	8.06E-03	3.58E-03	7.28E-04	2.28E-05	1.01E-06

不同 block errors 個數的比較

模擬數據 (Majority-Logic Decoding Algorithm)

- maximum number of block errors = 30

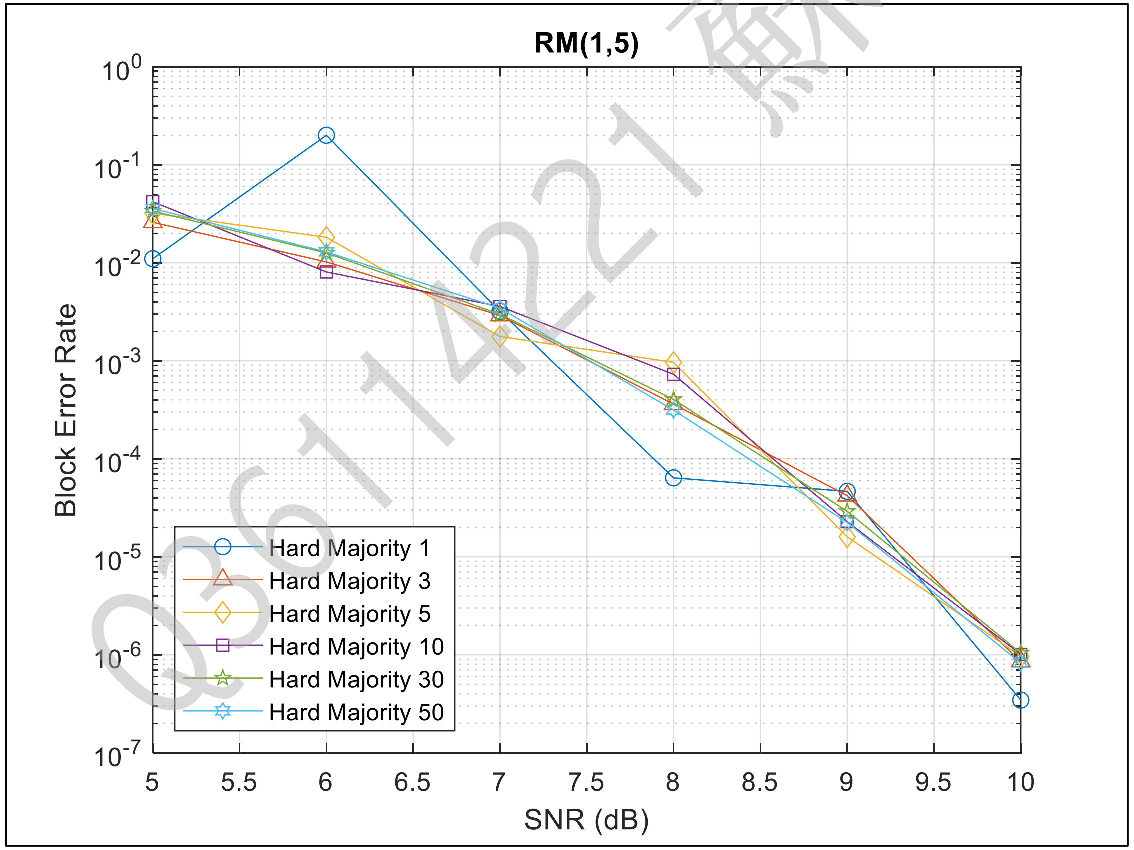
SNR (dB)	5	6	7	8	9	10
No. decoded block errors	30					
No. decoded blocks	894	2386	10060	74352	1032425	28790153
Block error rate (BLER)	3.36E-02	1.26E-02	2.98E-03	4.03E-04	2.91E-05	1.04E-06

- maximum number of block errors = 50

SNR (dB)	5	6	7	8	9	10
No. decoded block errors	50					
No. decoded blocks	1399	3880	14541	159235	2230083	60703678
Block error rate (BLER)	3.57E-02	1.29E-02	3.44E-03	3.14E-04	2.24E-05	8.24E-07

不同 block errors 個數的比較

模擬效能圖 (Majority-Logic Decoding Algorithm)



不同 block errors 個數的比較

模擬參數設定

- 解碼採用 Maximum Likelihood (HDML) Decoding Algorithm
- SNR (dB): 5, 6, 7, 8, 9
- Seed (a negative integer): 2000
- maximum number of block errors = 1, 3, 5, 10, 30, 50

不同 block errors 個數的比較

模擬數據 (HDML Decoding Algorithm)

- maximum number of block errors = 1

SNR (dB)	5	6	7	8	9
No. decoded block errors			1		
No. decoded blocks	156	102	294	2612	994678
Block error rate (BLER)	6.41E-03	9.80E-03	3.40E-03	3.83E-04	1.01E-06

- maximum number of block errors = 3

SNR (dB)	5	6	7	8	9
No. decoded block errors			3		
No. decoded blocks	268	2744	3812	54818	513368
Block error rate (BLER)	1.12E-02	1.09E-03	7.87E-04	5.47E-05	5.84E-06

不同 block errors 個數的比較

模擬數據 (HDML Decoding Algorithm)

- maximum number of block errors = 5

SNR (dB)	5	6	7	8	9
No. decoded block errors	5				
No. decoded blocks	384	848	20097	53288	589423
Block error rate (BLER)	1.30E-02	5.90E-03	2.49E-04	9.38E-05	8.48E-06

- maximum number of block errors = 10

SNR (dB)	5	6	7	8	9
No. decoded block errors	10				
No. decoded blocks	1309	2037	17434	173691	4999537
Block error rate (BLER)	7.64E-03	4.91E-03	5.74E-04	5.76E-05	2.00E-06

不同 block errors 個數的比較

模擬數據 (HDML Decoding Algorithm)

- maximum number of block errors = 30

SNR (dB)	5	6	7	8	9
No. decoded block errors	30				
No. decoded blocks	3523	11634	47627	483451	14023599
Block error rate (BLER)	8.52E-03	2.58E-03	6.30E-04	6.21E-05	2.14E-06

- maximum number of block errors = 50

SNR (dB)	5	6	7	8	9
No. decoded block errors	50				
No. decoded blocks	5268	17552	99561	1217628	21072680
Block error rate (BLER)	9.49E-03	2.85E-03	5.02E-04	4.11E-05	2.37E-06

不同 block errors 個數的比較

模擬效能圖 (HDML Decoding Algorithm)