Error Control Coding - Low-Density Parity-Check Codes

電通所　Q361142221　蘇沛錦

2023/06/07 (Wednesday)

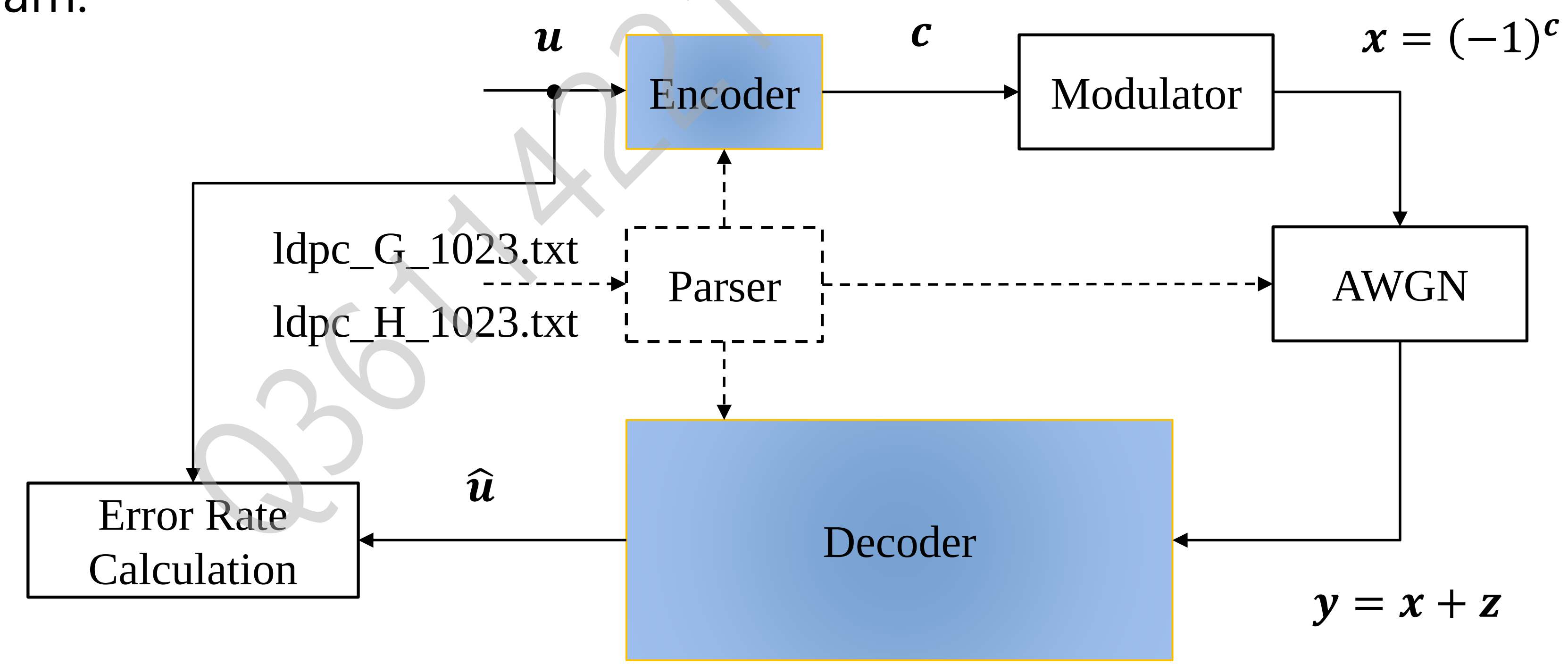
Outline

1. 系統架構圖 (細部Block diagram)
2. 程式流程解釋 (流程/Pseudo code)
3. 模擬數據 (表格)
4. 模擬效能圖

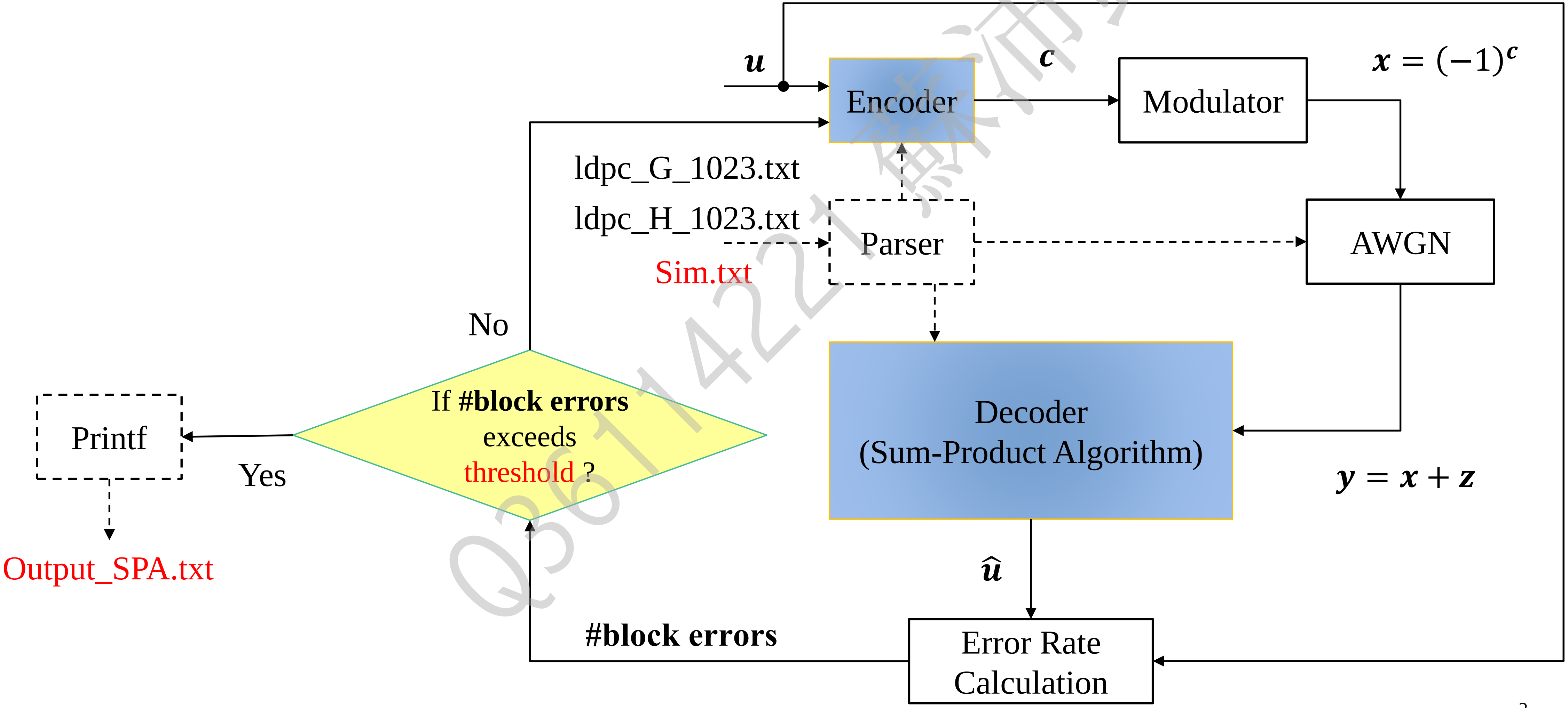
系統架構圖

Consider the (1023, 781) low-density parity-check (LDPC) code with block length $n = 1023$, dimension $k = 781$, row weight $\rho = 32$, and column weight $\gamma = 32$.

- Block diagram:

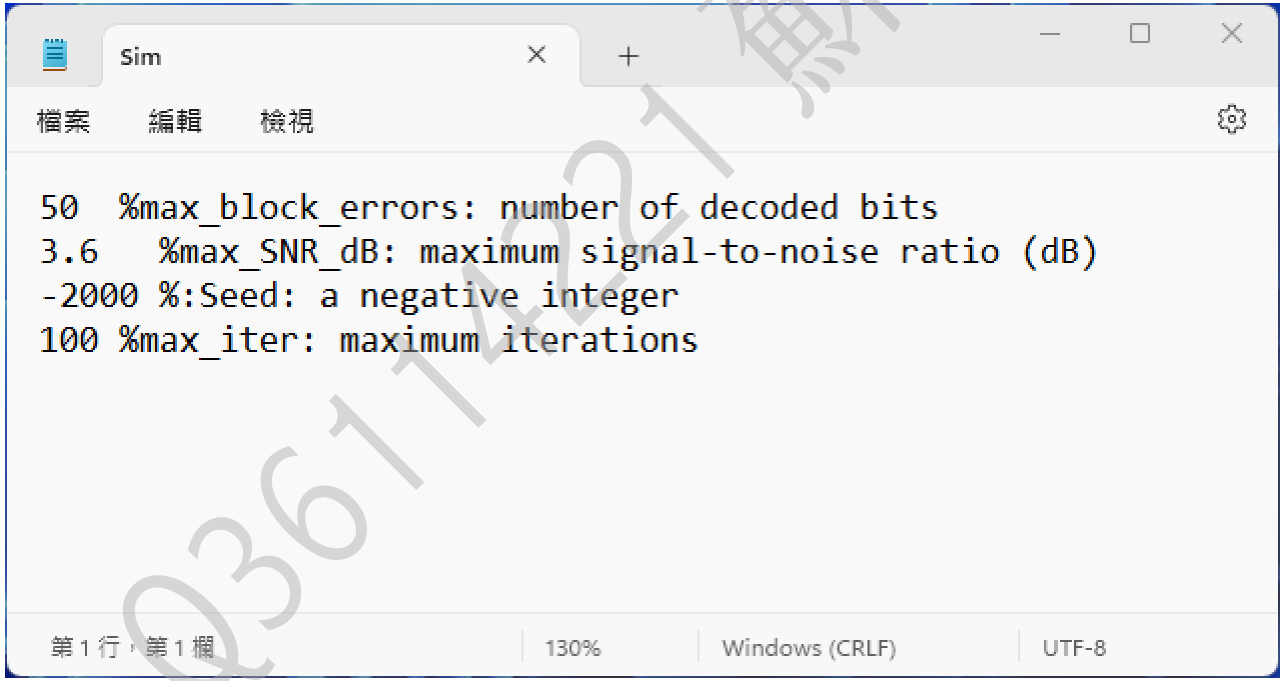


細部Block diagram



Decoding of Low-Density Parity-Check (LDPC) Codes

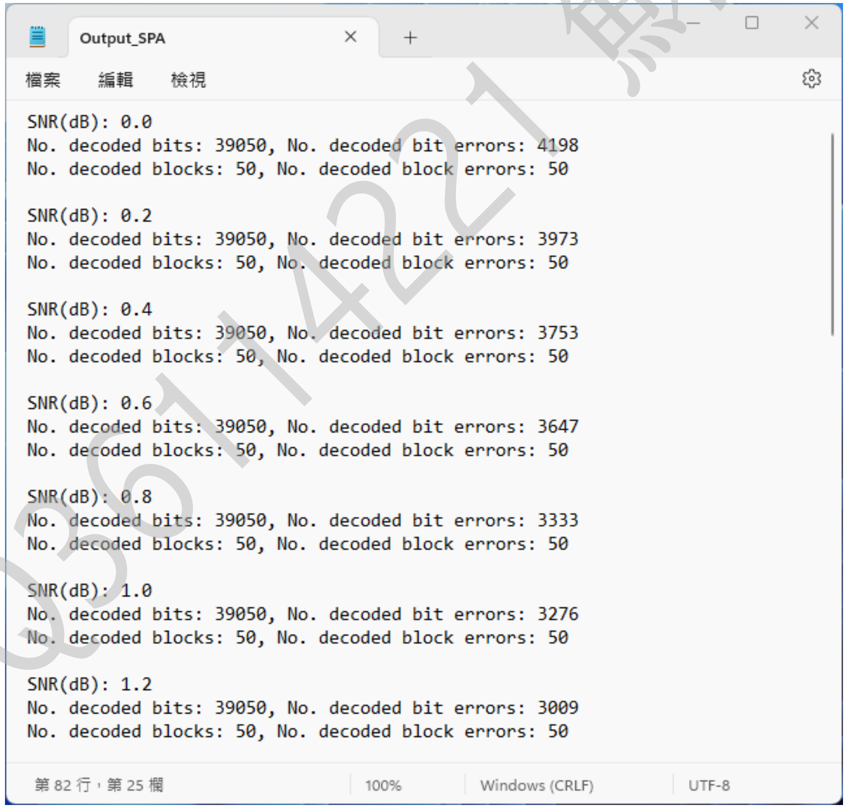
Input Sim.txt:



```
50  %max_block_errors: number of decoded bits
3.6   %max_SNR_dB: maximum signal-to-noise ratio (dB)
-2000 %:Seed: a negative integer
100 %max_iter: maximum iterations
```

Decoding of Low-Density Parity-Check (LDPC) Codes

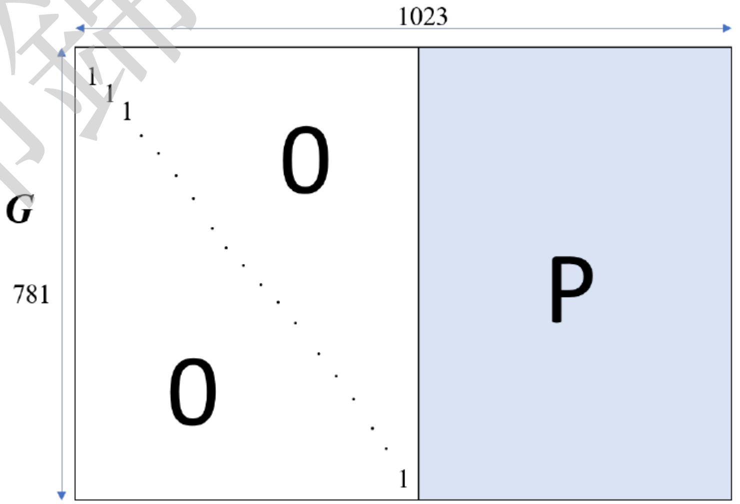
Output “Output_SPA.txt”:

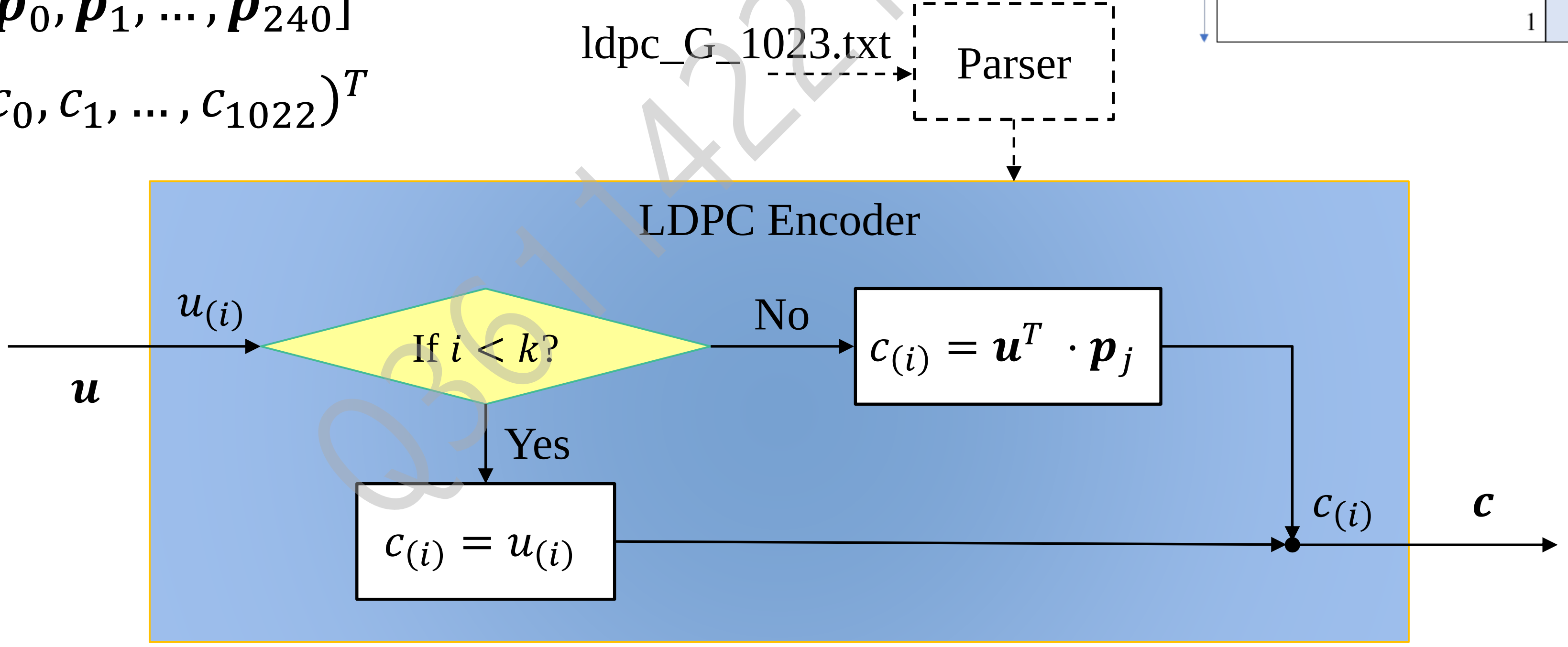


```
SNR(dB): 0.0
No. decoded bits: 39050, No. decoded bit errors: 4198
No. decoded blocks: 50, No. decoded block errors: 50

SNR(dB): 0.2
No. decoded bits: 39050, No. decoded bit errors: 3973
No. decoded blocks: 50, No. decoded block errors: 50

SNR(dB): 0.4
No. decoded bits: 39050, No. decoded bit errors: 3753
No. decoded blocks: 50, No. decoded block errors: 50

SNR(dB): 0.6
No. decoded bits: 39050, No. decoded bit errors: 3647
No. decoded blocks: 50, No. decoded block errors: 50

SNR(dB): 0.8
No. decoded bits: 39050, No. decoded bit errors: 3333
No. decoded blocks: 50, No. decoded block errors: 50

SNR(dB): 1.0
No. decoded bits: 39050, No. decoded bit errors: 3276
No. decoded blocks: 50, No. decoded block errors: 50

SNR(dB): 1.2
No. decoded bits: 39050, No. decoded bit errors: 3009
No. decoded blocks: 50, No. decoded block errors: 50
```

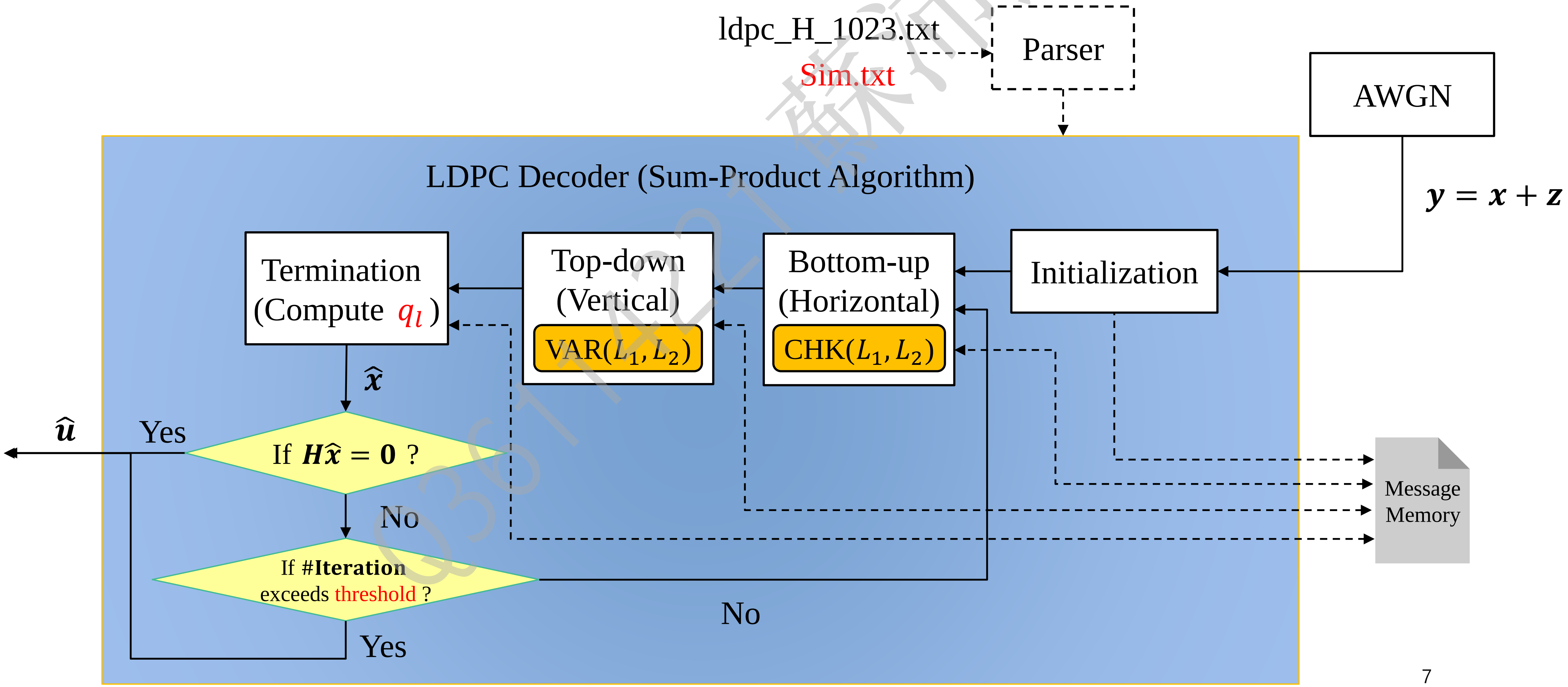
細部Block diagram

- $\boldsymbol{u} = (u_0, u_1, \dots, u_{780})^T$
- $\boldsymbol{G} = [\boldsymbol{I}_{781} \quad \boldsymbol{P}]$
- $\boldsymbol{P} = [\boldsymbol{p}_0, \boldsymbol{p}_1, \dots, \boldsymbol{p}_{240}]$
- $\boldsymbol{c} = (c_0, c_1, \dots, c_{1022})^T$





細部Block diagram



ldpc_H_1023.txt

Sim.txt

Parser

AWGN

$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{z}$

LDPC Decoder (Sum-Product Algorithm)

Termination (Compute q_l)

Top-down (Vertical)

VAR(L_1, L_2)

Bottom-up (Horizontal)

CHK(L_1, L_2)

Initialization

$\widehat{\boldsymbol{x}}$

$\widehat{\boldsymbol{u}}$

Yes

If $\boldsymbol{H}\widehat{\boldsymbol{x}} = \boldsymbol{0}$?

No

If #**Iteration** exceeds threshold ?

No

Yes

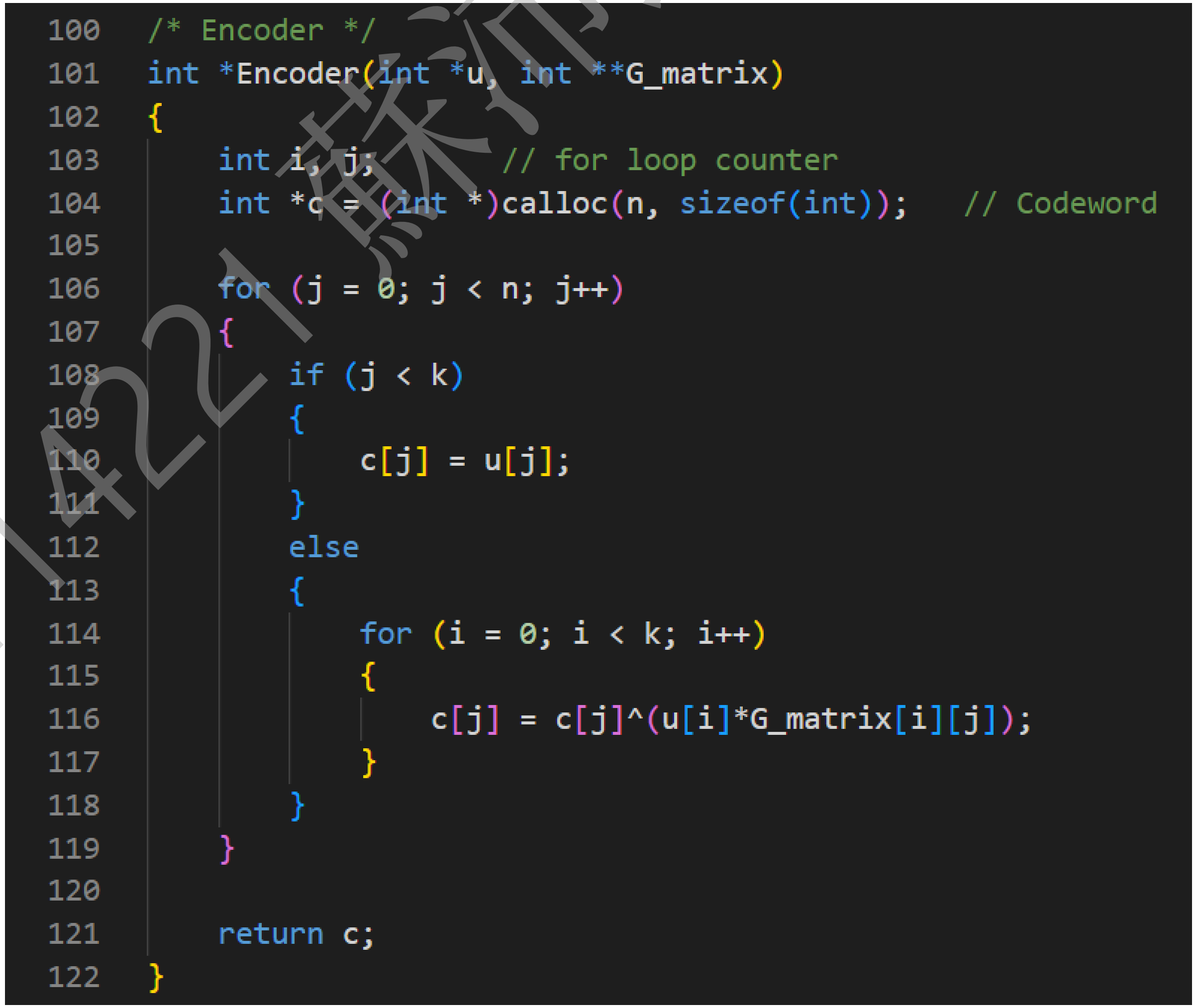
Message Memory

程式流程解釋 (Encoder)

- $\boldsymbol{u} = (u_0, u_1, \dots, u_{780})^T$
- $\boldsymbol{G} = [\boldsymbol{g}_0, \boldsymbol{g}_1, \dots, \boldsymbol{g}_{1021}]$
- $\boldsymbol{c} = (c_0, c_1, \dots, c_{1022})^T$

- Pseudo code:

```
for $i = 0, \dots, k-1$ do
    if $(i < k)$
        $c_i \leftarrow u_i$
    then
        $c_i \leftarrow \boldsymbol{u}^T \cdot \boldsymbol{g}_i$
  end for
  return $\boldsymbol{c}$
```



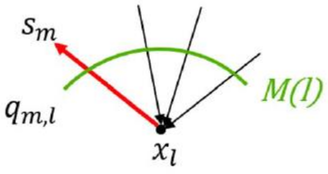
```c
/* Encoder */
int *Encoder(int *u, int **G_matrix)
{
    int i, j;       // for loop counter
    int *c = (int *)calloc(n, sizeof(int));   // Codeword

    for (j = 0; j < n; j++)
    {
        if (j < k)
        {
            c[j] = u[j];
        }
        else
        {
            for (i = 0; i < k; i++)
            {
                c[j] = c[j]^(u[i]*G_matrix[i][j]);
            }
        }
    }

    return c;
}
```

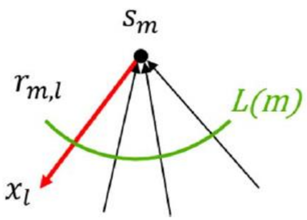
程式流程解釋 (Decoder)

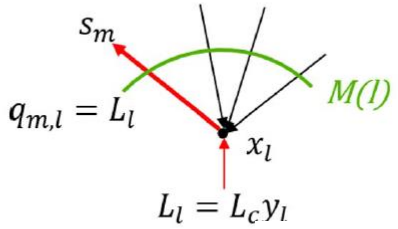
- Pseudo code:

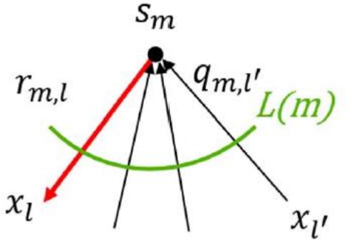
```
for $\forall\, m \in M(l), 1 \le l \le n$                         // Initialization
    $q_{m,l} \leftarrow L_l = 2 \cdot y_l/\sigma^2$
end for
repeat
    for $\forall\, l \in L(m), 1 \le m \le J$                     // Step 1: Bottom-up (Horizontal)
        $r_{m,l} \leftarrow \mathrm{CHK}_{l' \in L(m)\backslash l}(q_{m,l'})$
    end for
    for $\forall\, m \in M(l), 1 \le l \le n$                     // Step 2: Top-down (Vertical)
        $q_{m,l} \leftarrow \mathrm{VAR}(\mathrm{VAR}_{m' \in M(l)\backslash m}(r_{m',l}), L_l) = L_l + \sum_{m' \in M(l)\backslash m} r_{m',l}$
    end for
    for $1 \le l \le n$                                           // Step 3: Termination
        $q_l \leftarrow \mathrm{VAR}(\mathrm{VAR}_{m \in M(l)}(r_{m,l}), L_l) = L_l + \sum_{m \in M(l)} r_{m,l}$
        if $q_l > 0$
            $\hat{x}_l = 0$
        then
            $\hat{x}_l = 1$
    end for
    if $H\hat{x} = 0$
        break
until $\#iterations \ge threshold$
return $x$
```

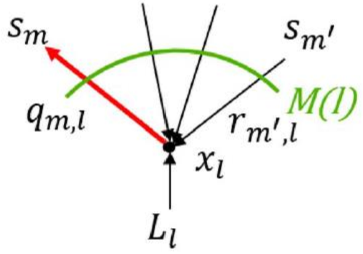
- $L(m) = \{l : H_{m,l} = 1\}, 1 \le m \le J$
- $M(l) = \{m : H_{m,l} = 1\}, 1 \le l \le n$

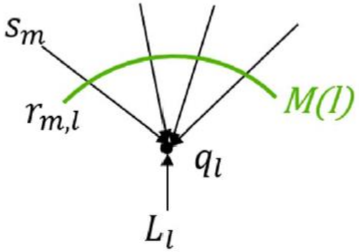






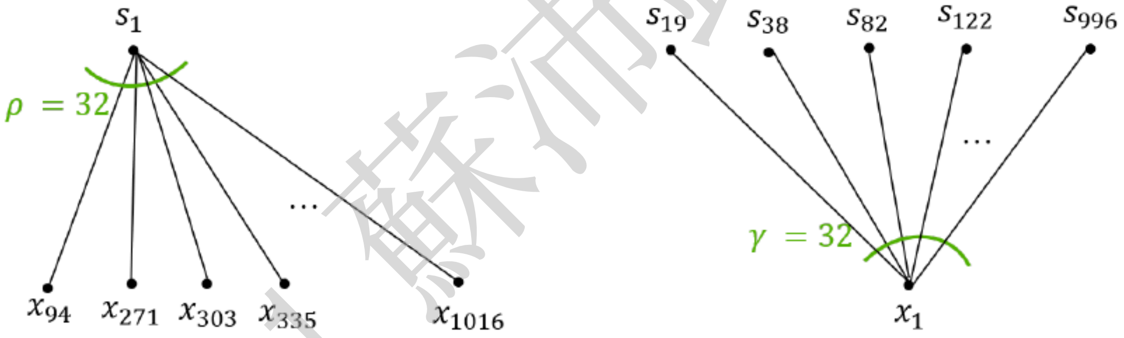




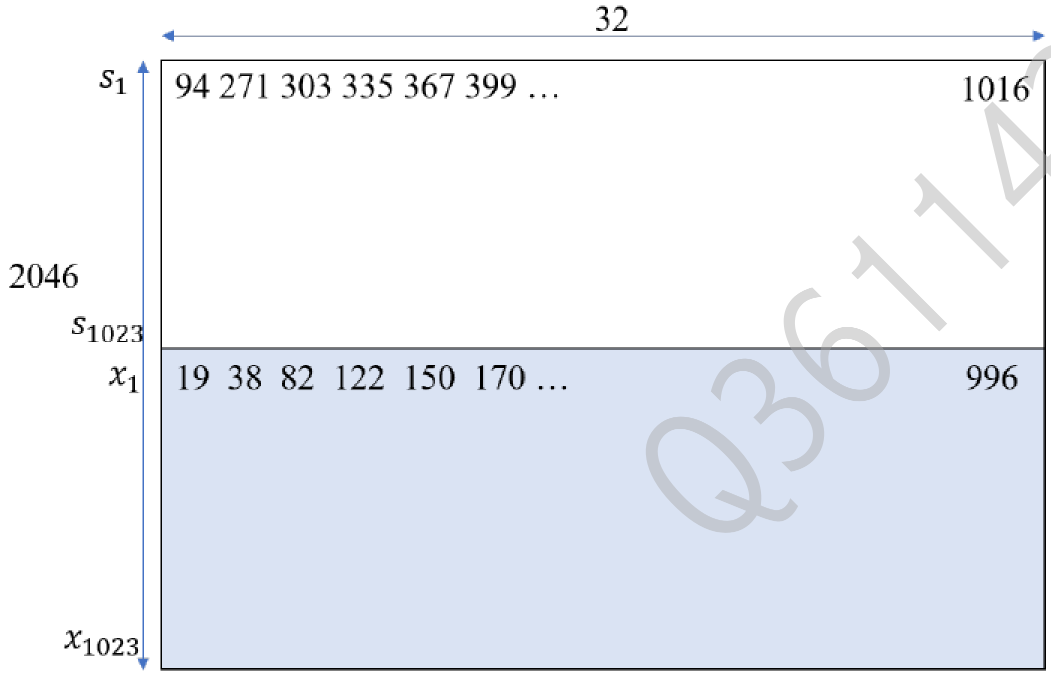


程式流程解釋 (Decoder)

- Message Memory



ldpc_H_1023.txt



32

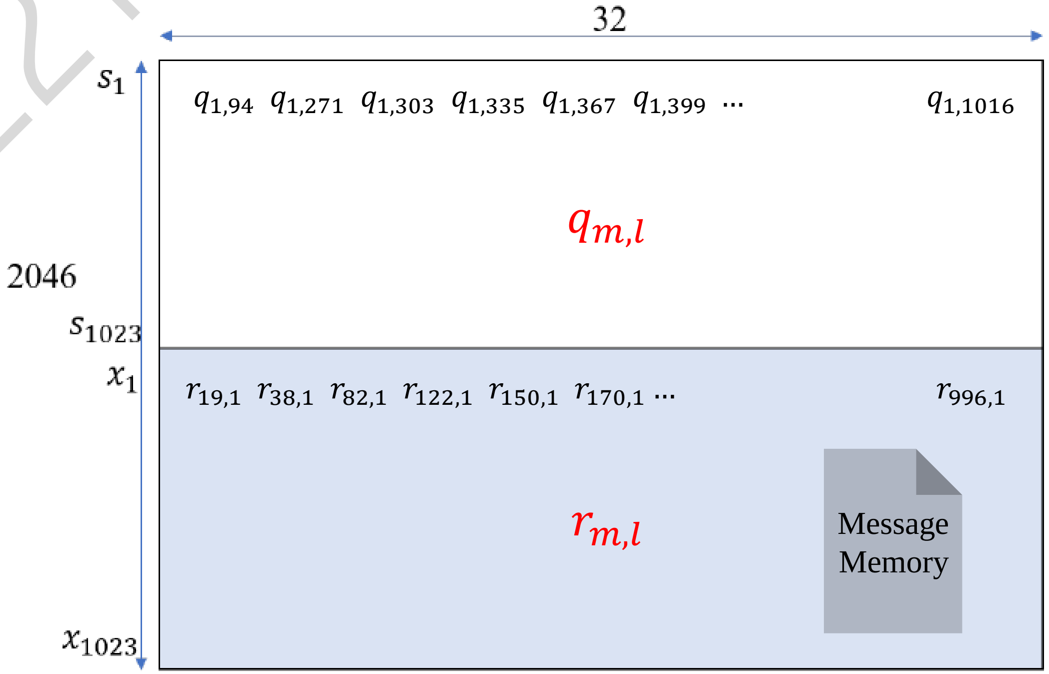
2046

s_1 94 271 303 335 367 399 … 1016

s_{1023}

x_1 19 38 82 122 150 170 … 996

x_{1023}



32

2046

s_1 $q_{1,94}$ $q_{1,271}$ $q_{1,303}$ $q_{1,335}$ $q_{1,367}$ $q_{1,399}$ … $q_{1,1016}$

$q_{m,l}$

s_{1023}

x_1 $r_{19,1}$ $r_{38,1}$ $r_{82,1}$ $r_{122,1}$ $r_{150,1}$ $r_{170,1}$ … $r_{996,1}$

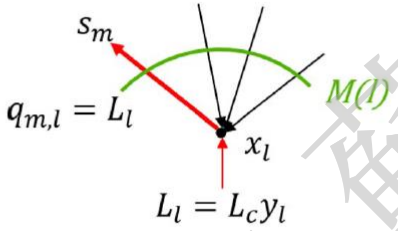
$r_{m,l}$

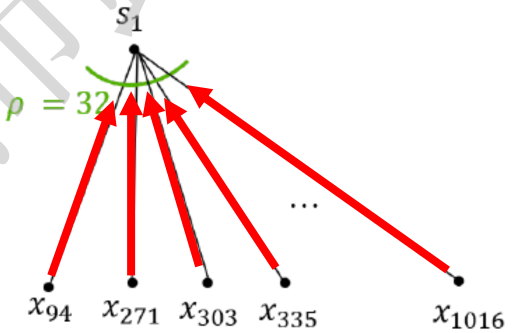
Message Memory

x_{1023}

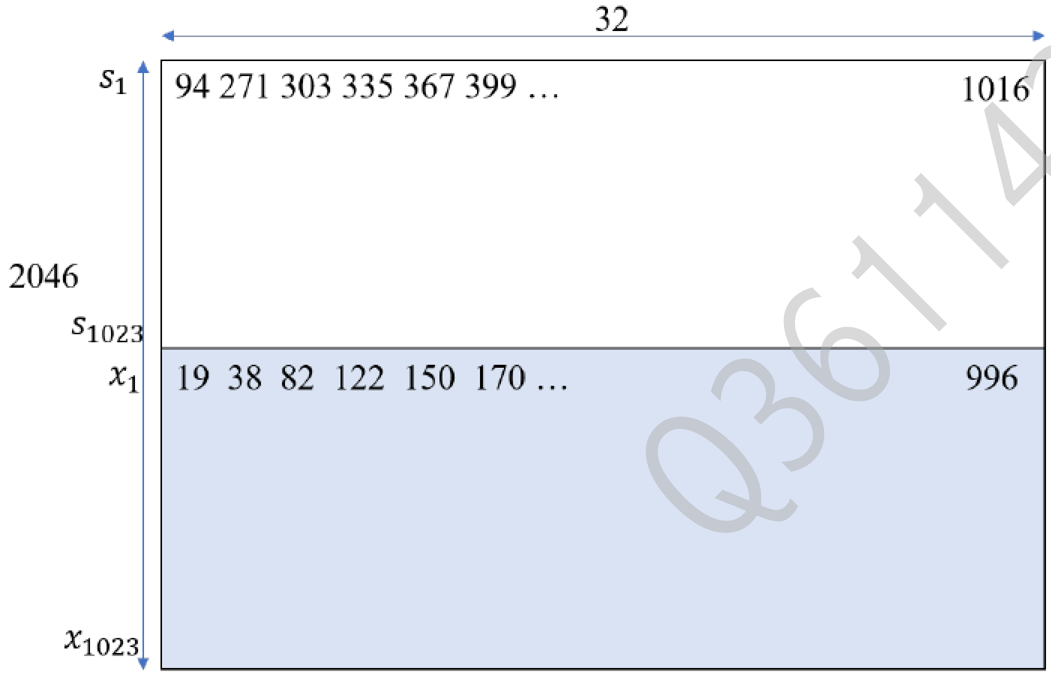
程式流程解釋 (Decoder)

- Initialization:



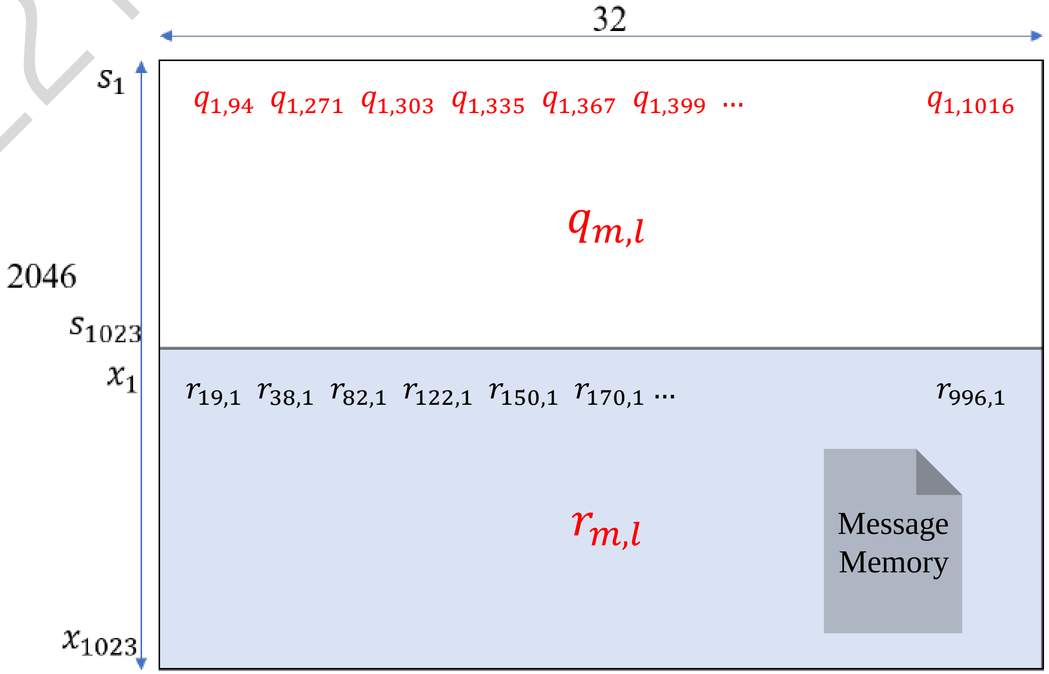


ldpc_H_1023.txt



94 271 303 335 367 399 … 1016

19 38 82 122 150 170 … 996



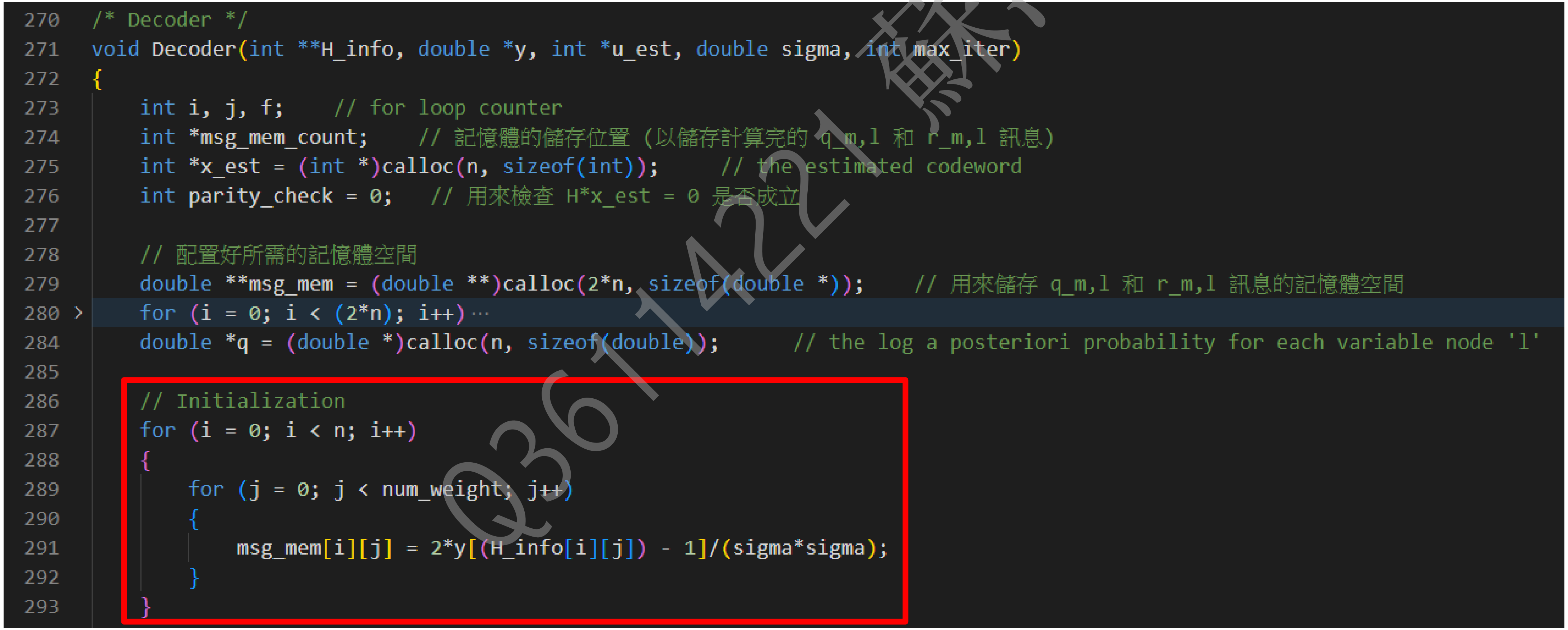
$q_{1,94}$ $q_{1,271}$ $q_{1,303}$ $q_{1,335}$ $q_{1,367}$ $q_{1,399}$ ⋯ $q_{1,1016}$

$q_{m,l}$

$r_{19,1}$ $r_{38,1}$ $r_{82,1}$ $r_{122,1}$ $r_{150,1}$ $r_{170,1}$ ⋯ $r_{996,1}$

$r_{m,l}$

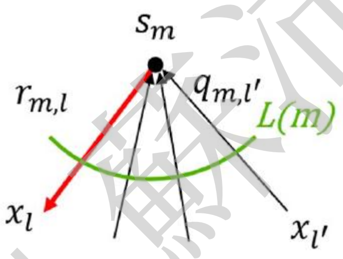
程式流程解釋 (Decoder)

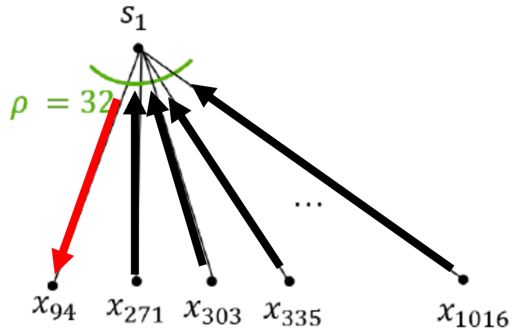


```
/* Decoder */
void Decoder(int **H_info, double *y, int *u_est, double sigma, int max_iter)
{
    int i, j, f;     // for loop counter
    int *msg_mem_count;    // 記憶體的儲存位置 (以儲存計算完的 q_m,l 和 r_m,l 訊息)
    int *x_est = (int *)calloc(n, sizeof(int));     // the estimated codeword
    int parity_check = 0;   // 用來檢查 H*x_est = 0 是否成立

    // 配置好所需的記憶體空間
    double **msg_mem = (double **)calloc(2*n, sizeof(double *));    // 用來儲存 q_m,l 和 r_m,l 訊息的記憶體空間
    for (i = 0; i < (2*n); i++) ···
    double *q = (double *)calloc(n, sizeof(double));      // the log a posteriori probability for each variable node 'l'

    // Initialization
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < num_weight; j++)
        {
            msg_mem[i][j] = 2*y[(H_info[i][j]) - 1]/(sigma*sigma);
        }
    }
```

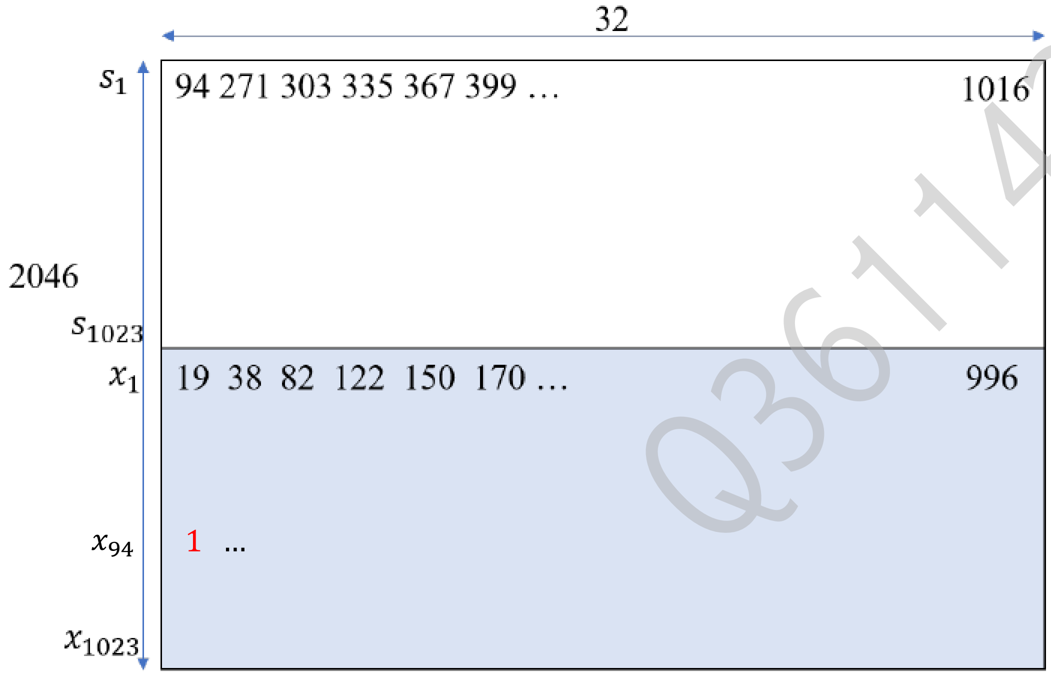
程式流程解釋 (Decoder)

- Step 1: Bottom-up (Horizontal)





ldpc_H_1023.txt



32

2046

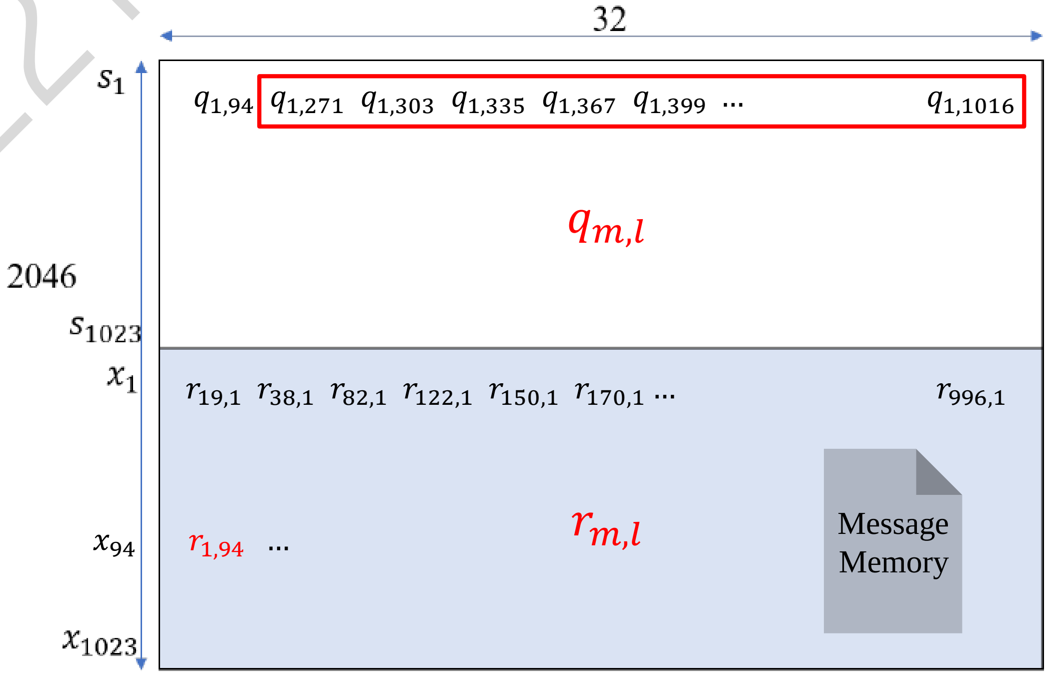
s_1: 94 271 303 335 367 399 … 1016

s_{1023}

x_1: 19 38 82 122 150 170 … 996

x_{94}: 1 …

x_{1023}



32

2046

s_1: $q_{1,94}$ $q_{1,271}$ $q_{1,303}$ $q_{1,335}$ $q_{1,367}$ $q_{1,399}$ … $q_{1,1016}$

$q_{m,l}$

s_{1023}

x_1: $r_{19,1}$ $r_{38,1}$ $r_{82,1}$ $r_{122,1}$ $r_{150,1}$ $r_{170,1}$ … $r_{996,1}$

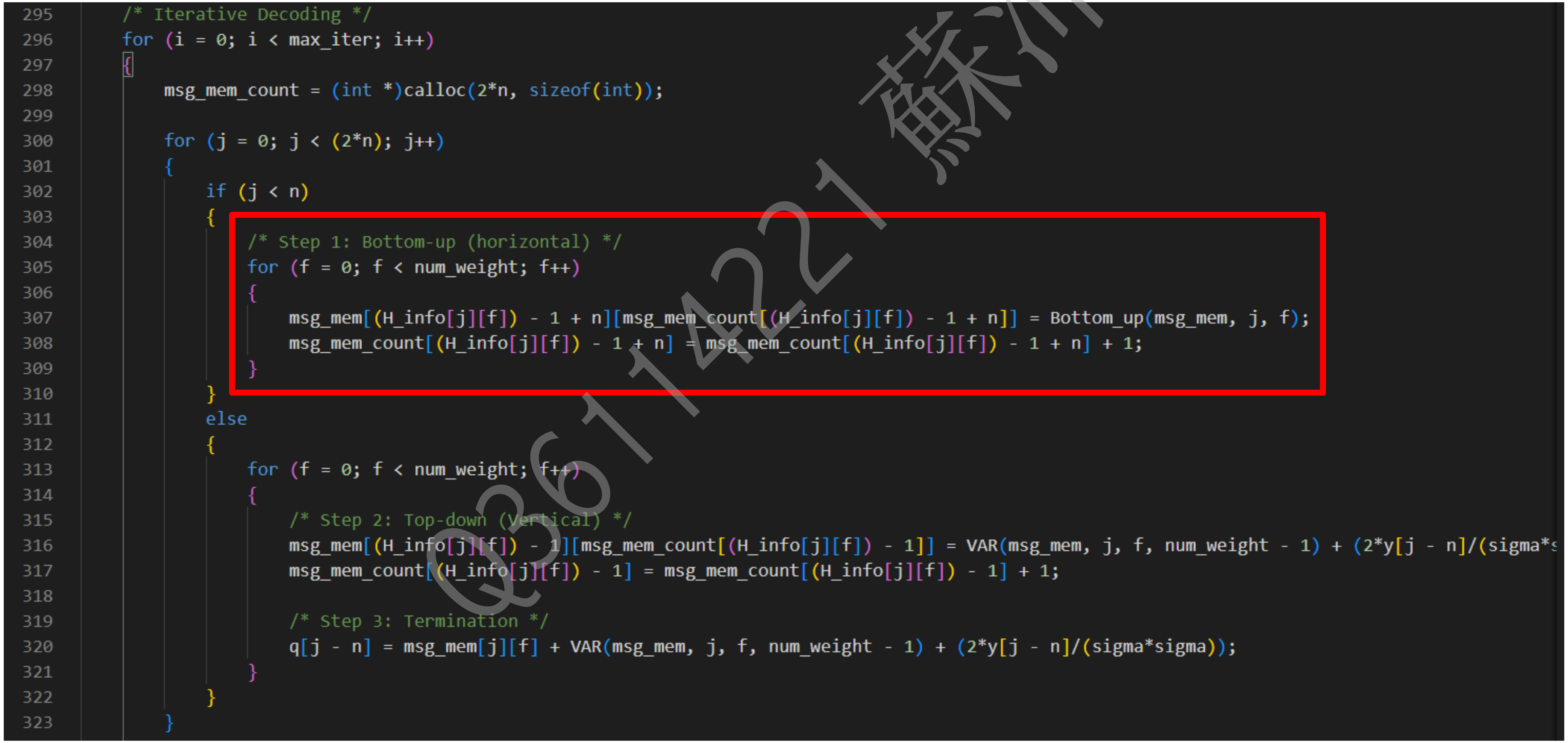
x_{94}: $r_{1,94}$ …

$r_{m,l}$

Message Memory

x_{1023}

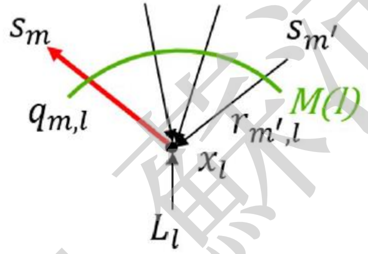
程式流程解釋 (Decoder)

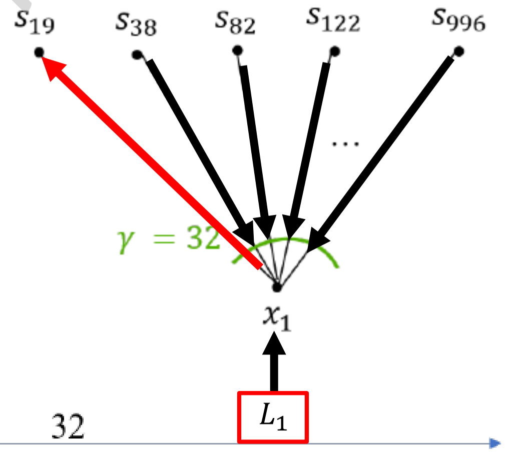


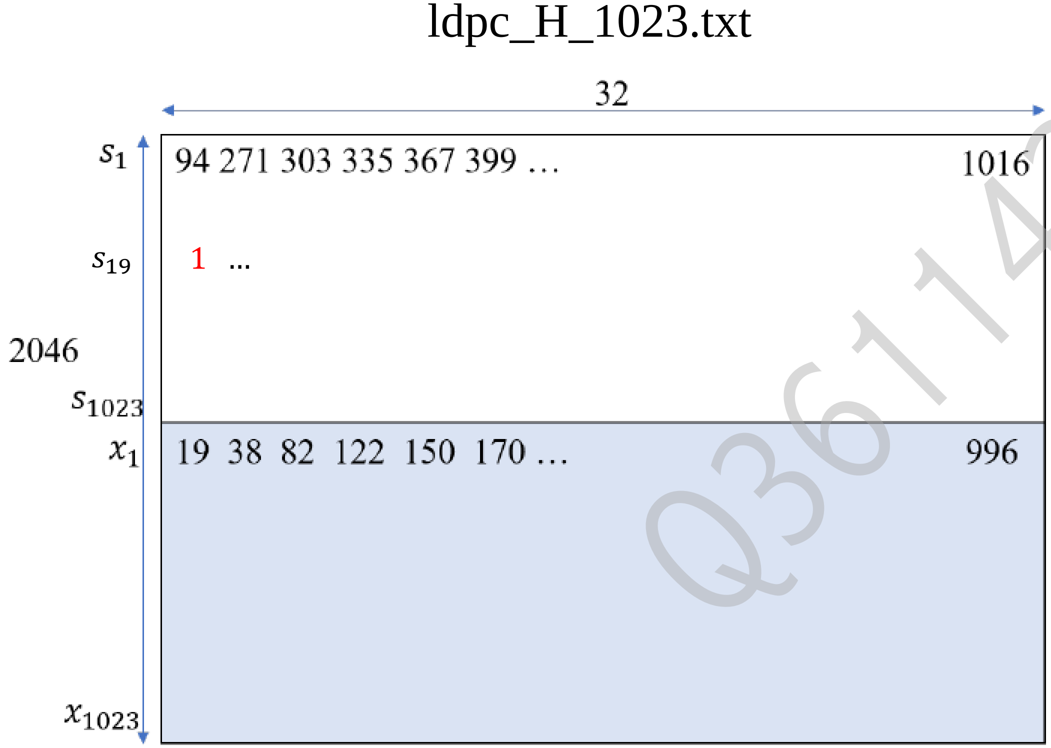
```c
    /* Iterative Decoding */
    for (i = 0; i < max_iter; i++)
    {
        msg_mem_count = (int *)calloc(2*n, sizeof(int));

        for (j = 0; j < (2*n); j++)
        {
            if (j < n)
            {
                /* Step 1: Bottom-up (horizontal) */
                for (f = 0; f < num_weight; f++)
                {
                    msg_mem[(H_info[j][f]) - 1 + n][msg_mem_count[(H_info[j][f]) - 1 + n]] = Bottom_up(msg_mem, j, f);
                    msg_mem_count[(H_info[j][f]) - 1 + n] = msg_mem_count[(H_info[j][f]) - 1 + n] + 1;
                }
            }
            else
            {
                for (f = 0; f < num_weight; f++)
                {
                    /* Step 2: Top-down (Vertical) */
                    msg_mem[(H_info[j][f]) - 1][msg_mem_count[(H_info[j][f]) - 1]] = VAR(msg_mem, j, f, num_weight - 1) + (2*y[j - n]/(sigma*s
                    msg_mem_count[(H_info[j][f]) - 1] = msg_mem_count[(H_info[j][f]) - 1] + 1;

                    /* Step 3: Termination */
                    q[j - n] = msg_mem[j][f] + VAR(msg_mem, j, f, num_weight - 1) + (2*y[j - n]/(sigma*sigma));
                }
            }
        }
```

程式流程解釋 (Decoder)

- Step 2: Top-down (Vertical)







ldpc_H_1023.txt

32

2046

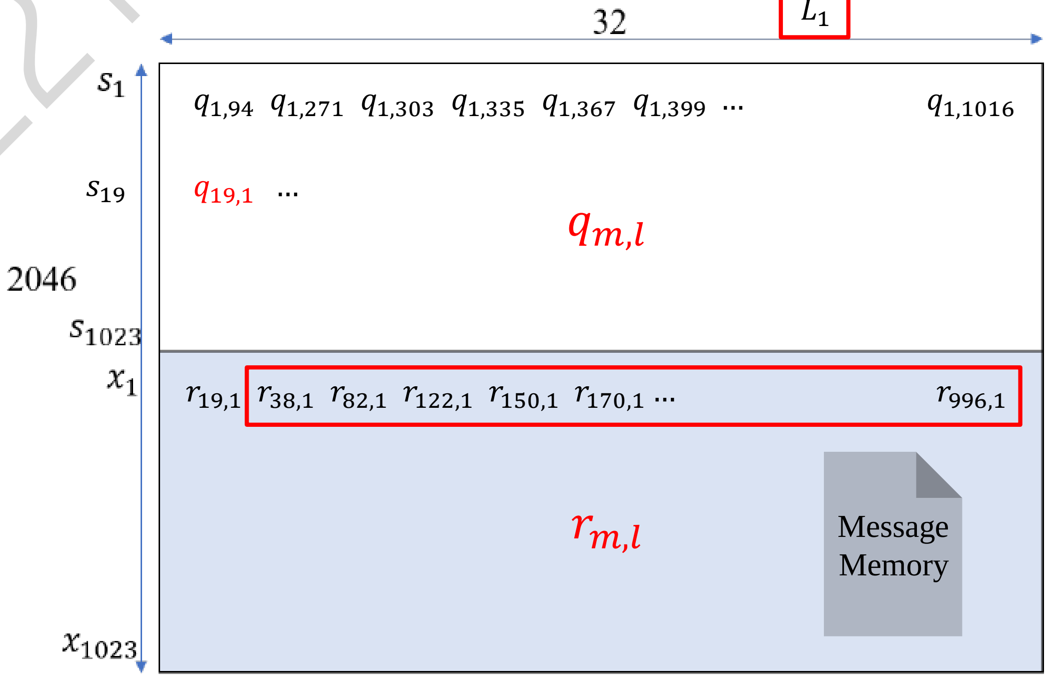
s_1 94 271 303 335 367 399 … 1016

s_{19} 1 …

s_{1023}

x_1 19 38 82 122 150 170 … 996

x_{1023}



32

2046

s_1 $q_{1,94}$ $q_{1,271}$ $q_{1,303}$ $q_{1,335}$ $q_{1,367}$ $q_{1,399}$ ⋯ $q_{1,1016}$

s_{19} $q_{19,1}$ ⋯

$q_{m,l}$

s_{1023}

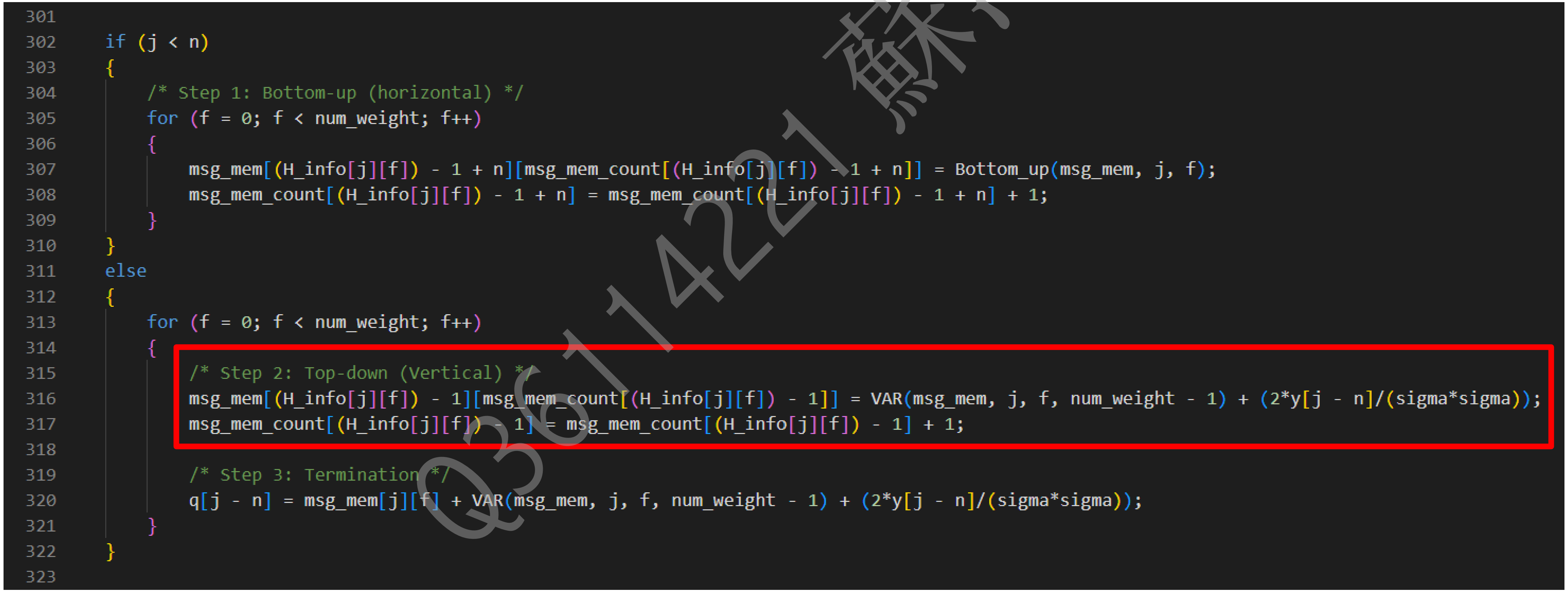
x_1 $r_{19,1}$ $r_{38,1}$ $r_{82,1}$ $r_{122,1}$ $r_{150,1}$ $r_{170,1}$ ⋯ $r_{996,1}$

$r_{m,l}$

Message Memory

x_{1023}

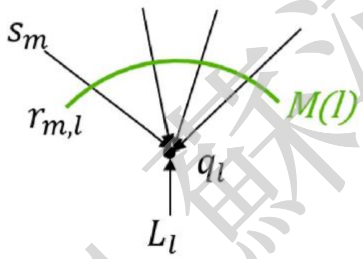
程式流程解釋 (Decoder)

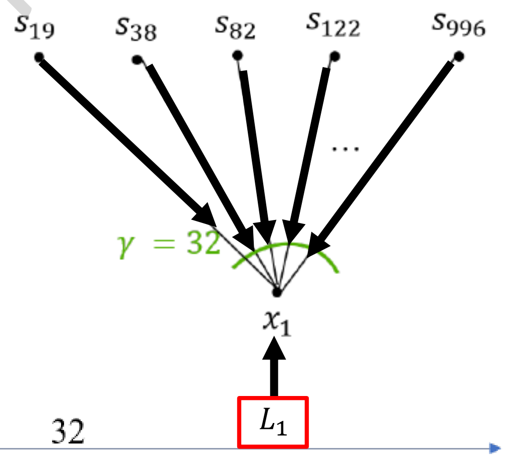


```

    if (j < n)
    {
        /* Step 1: Bottom-up (horizontal) */
        for (f = 0; f < num_weight; f++)
        {
            msg_mem[(H_info[j][f]) - 1 + n][msg_mem_count[(H_info[j][f]) - 1 + n]] = Bottom_up(msg_mem, j, f);
            msg_mem_count[(H_info[j][f]) - 1 + n] = msg_mem_count[(H_info[j][f]) - 1 + n] + 1;
        }
    }
    else
    {
        for (f = 0; f < num_weight; f++)
        {
            /* Step 2: Top-down (Vertical) */
            msg_mem[(H_info[j][f]) - 1][msg_mem_count[(H_info[j][f]) - 1]] = VAR(msg_mem, j, f, num_weight - 1) + (2*y[j - n]/(sigma*sigma));
            msg_mem_count[(H_info[j][f]) - 1] = msg_mem_count[(H_info[j][f]) - 1] + 1;

            /* Step 3: Termination */
            q[j - n] = msg_mem[j][f] + VAR(msg_mem, j, f, num_weight - 1) + (2*y[j - n]/(sigma*sigma));
        }
    }

```

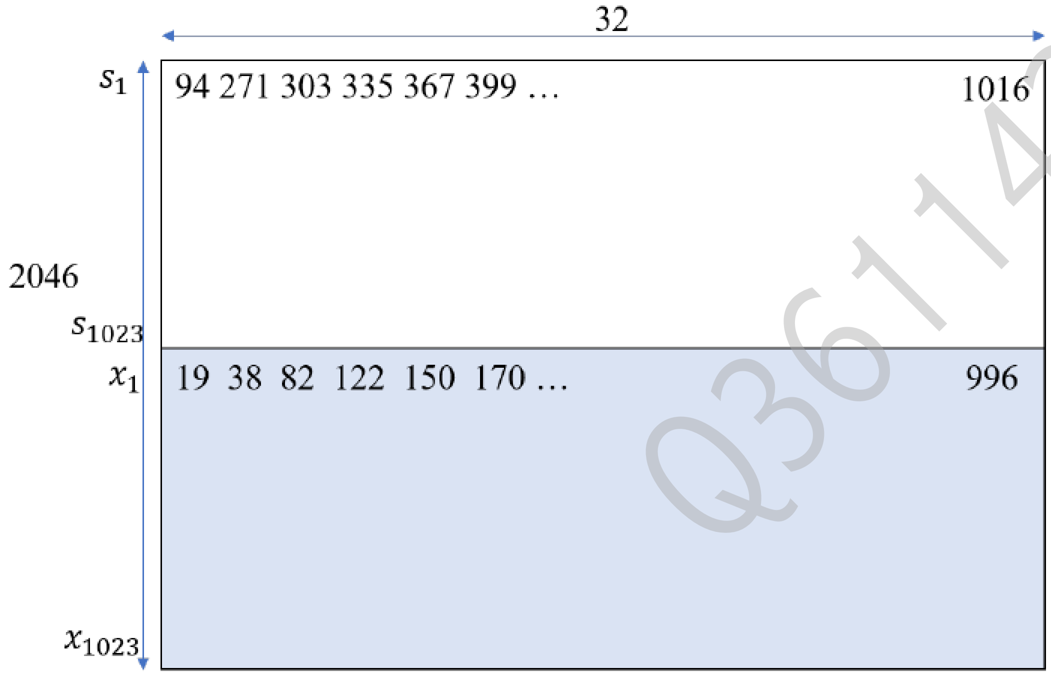
程式流程解釋 (Decoder)

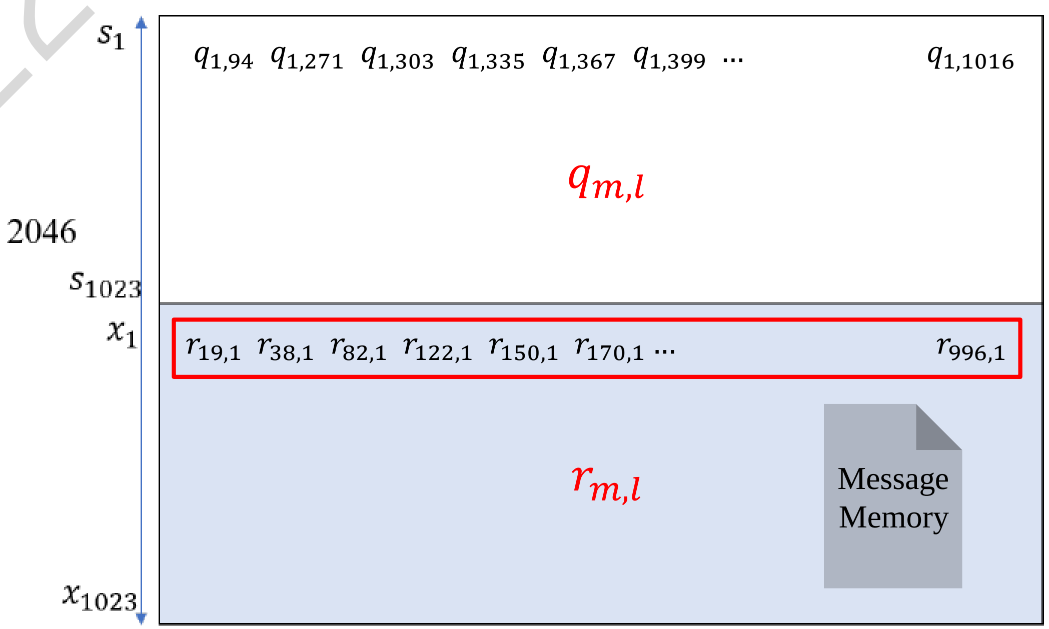
- Step 3: Termination





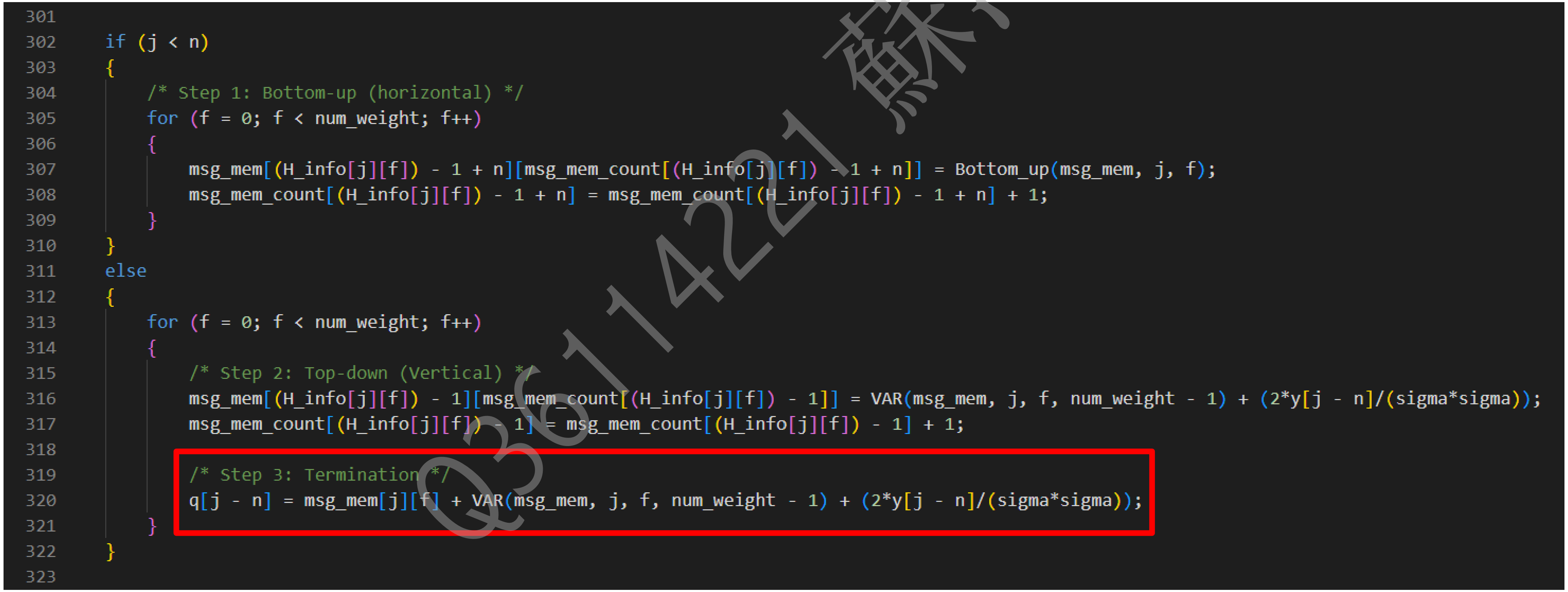
ldpc_H_1023.txt





32
2046
s_1
$q_{1,94}$ $q_{1,271}$ $q_{1,303}$ $q_{1,335}$ $q_{1,367}$ $q_{1,399}$ ⋯ $q_{1,1016}$
$q_{m,l}$
s_{1023}
x_1
$r_{19,1}$ $r_{38,1}$ $r_{82,1}$ $r_{122,1}$ $r_{150,1}$ $r_{170,1}$ ⋯ $r_{996,1}$
$r_{m,l}$
Message Memory
x_{1023}

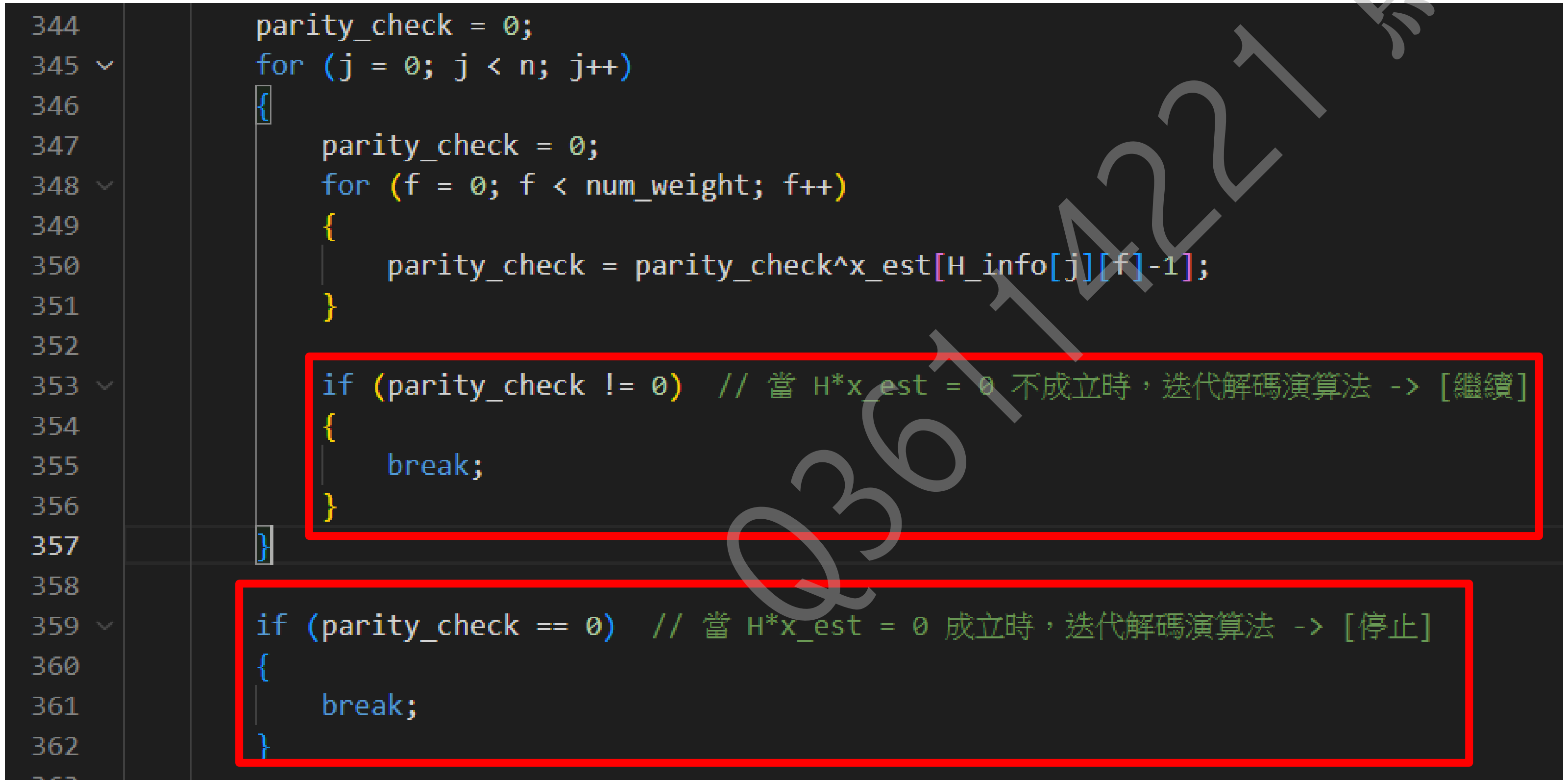
程式流程解釋 (Decoder)



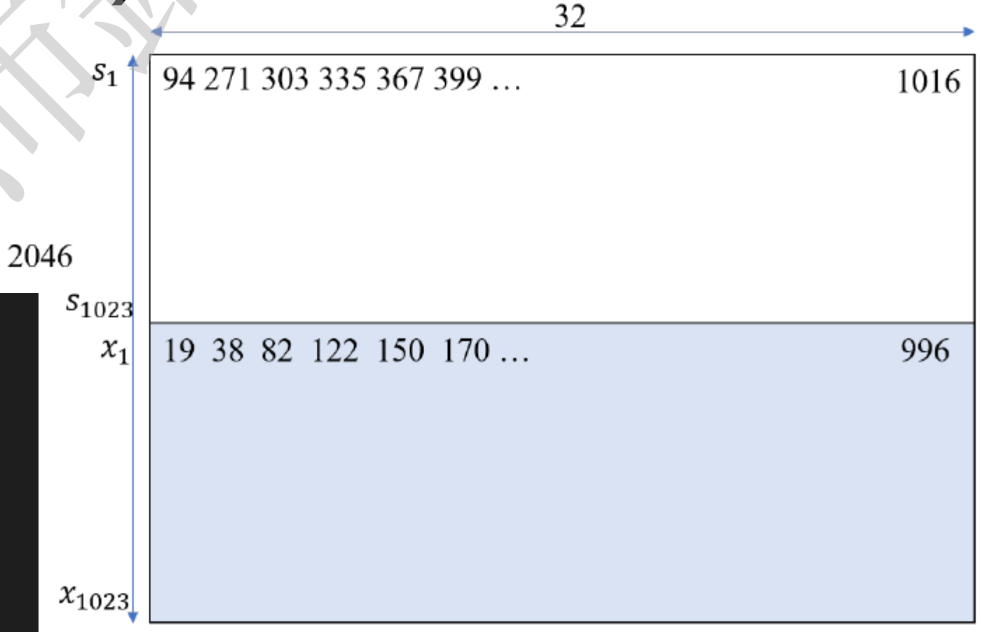
```
if (j < n)
{
    /* Step 1: Bottom-up (horizontal) */
    for (f = 0; f < num_weight; f++)
    {
        msg_mem[(H_info[j][f]) - 1 + n][msg_mem_count[(H_info[j][f]) - 1 + n]] = Bottom_up(msg_mem, j, f);
        msg_mem_count[(H_info[j][f]) - 1 + n] = msg_mem_count[(H_info[j][f]) - 1 + n] + 1;
    }
}
else
{
    for (f = 0; f < num_weight; f++)
    {
        /* Step 2: Top-down (Vertical) */
        msg_mem[(H_info[j][f]) - 1][msg_mem_count[(H_info[j][f]) - 1]] = VAR(msg_mem, j, f, num_weight - 1) + (2*y[j - n]/(sigma*sigma));
        msg_mem_count[(H_info[j][f]) - 1] = msg_mem_count[(H_info[j][f]) - 1] + 1;

        /* Step 3: Termination */
        q[j - n] = msg_mem[j][f] + VAR(msg_mem, j, f, num_weight - 1) + (2*y[j - n]/(sigma*sigma));
    }
}
```

程式流程解釋 (Decoder)

- If $\boldsymbol{H}\hat{\boldsymbol{x}} = \boldsymbol{0}$, then $\hat{\boldsymbol{x}}$ is the codeword.
  - The algorithm stops.



```
parity_check = 0;
for (j = 0; j < n; j++)
{
    parity_check = 0;
    for (f = 0; f < num_weight; f++)
    {
        parity_check = parity_check^x_est[H_info[j][f]-1];
    }

    if (parity_check != 0)  // 當 H*x_est = 0 不成立時，迭代解碼演算法 -> [繼續]
    {
        break;
    }
}

if (parity_check == 0)  // 當 H*x_est = 0 成立時，迭代解碼演算法 -> [停止]
{
    break;
}
```

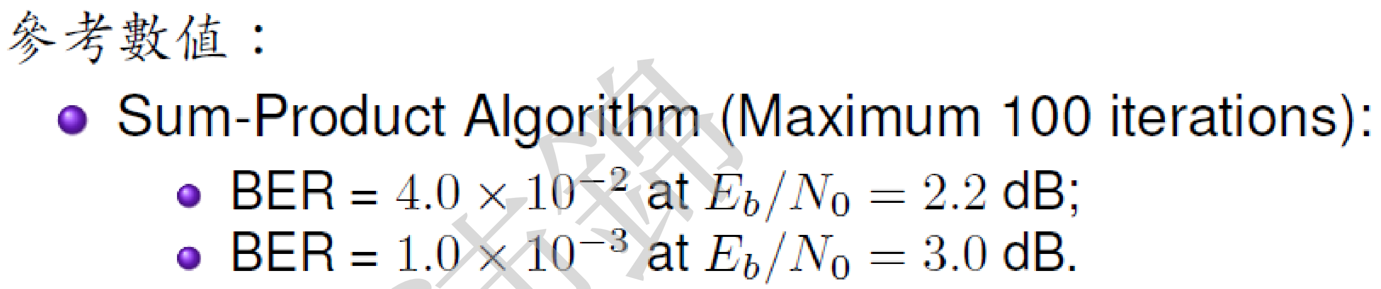


ldpc_H_1023.txt

模擬參數設定

- 解碼採用 Sum-Product Algorithm (SPA)
- SNR (dB) → 0.0 : 0.2 : 3.6
- Seed (a negative integer): -2000
- number of maximum block errors = 50 or 100
- number of maximum iterations = 50 or 100

模擬數據



參考數值：

- Sum-Product Algorithm (Maximum 100 iterations):
  - BER = 4.0×10^{-2} at $E_b/N_0 = 2.2$ dB;
  - BER = 1.0×10^{-3} at $E_b/N_0 = 3.0$ dB.

- number of maximum block errors = 50
- number of maximum iterations = 100

SNR (dB)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6
No. decoded bit errors	4198	3973	3753	3647	3333	3276	3009	2749	2690	2541	2253	2165	2025	2112	2080	2079	2171	2148	2070
No. decoded bits	39050	39050	39050	39050	39050	39050	39050	39050	39050	41393	45298	57013	107778	177287	450637	1946252	5609923	18160593	72659554
bit error rate (BER)	1.08E-01	1.02E-01	9.61E-02	9.34E-02	8.54E-02	8.39E-02	7.71E-02	7.04E-02	6.89E-02	6.14E-02	4.97E-02	3.80E-02	1.88E-02	1.19E-02	4.62E-03	1.07E-03	3.87E-04	1.18E-04	2.85E-05

SNR (dB)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6
No. decoded block errors	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
No. decoded blocks	50	50	50	50	50	50	50	50	50	53	58	73	138	227	577	2492	7183	23253	93034
block error rate (BLER)	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	9.43E-01	8.62E-01	6.85E-01	3.62E-01	2.20E-01	8.67E-02	2.01E-02	6.96E-03	2.15E-03	5.37E-04

模擬數據

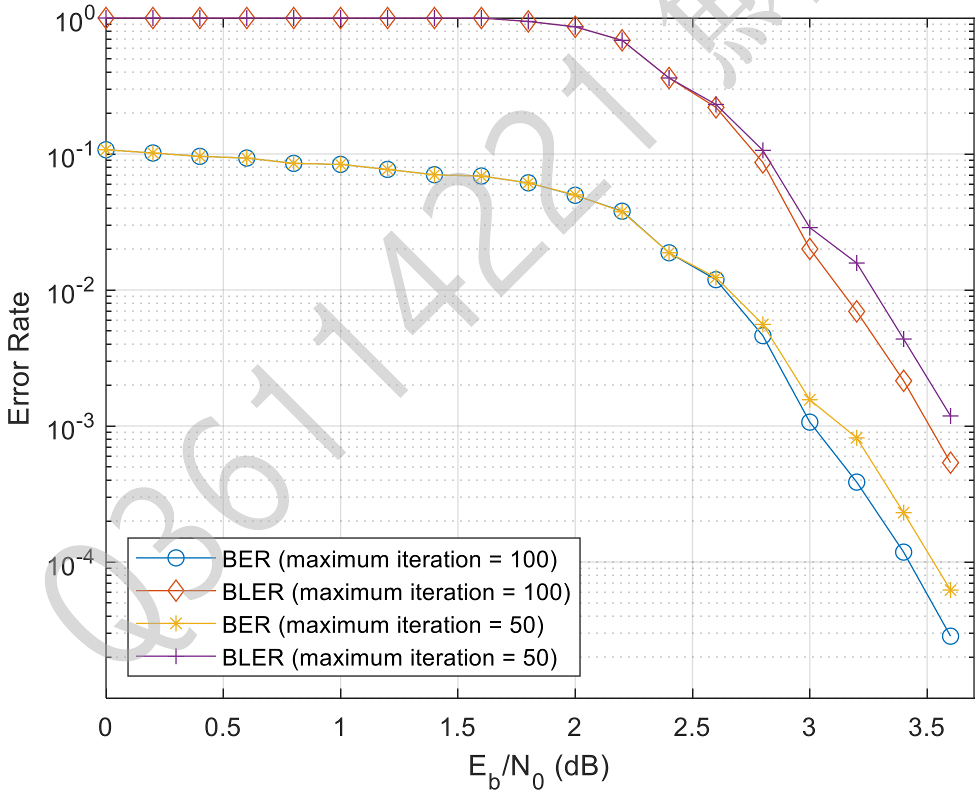
- number of maximum block errors = 50 (fixed)
- number of maximum iterations = 50

SNR (dB)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6
No. decoded bit errors	4198	3973	3753	3647	3333	3276	3009	2749	2687	2541	2265	2143	2026	2085	2051	2118	2024	2066	2049
No. decoded bits	39050	39050	39050	39050	39050	39050	39050	39050	39050	41393	45298	57013	107778	168696	367070	1357378	2473427	8954946	32931646
bit error rate (BER)	1.08E-01	1.02E-01	9.61E-02	9.34E-02	8.54E-02	8.39E-02	7.71E-02	7.04E-02	6.88E-02	6.14E-02	5.00E-02	3.76E-02	1.88E-02	1.24E-02	5.59E-03	1.56E-03	8.18E-04	2.31E-04	6.22E-05

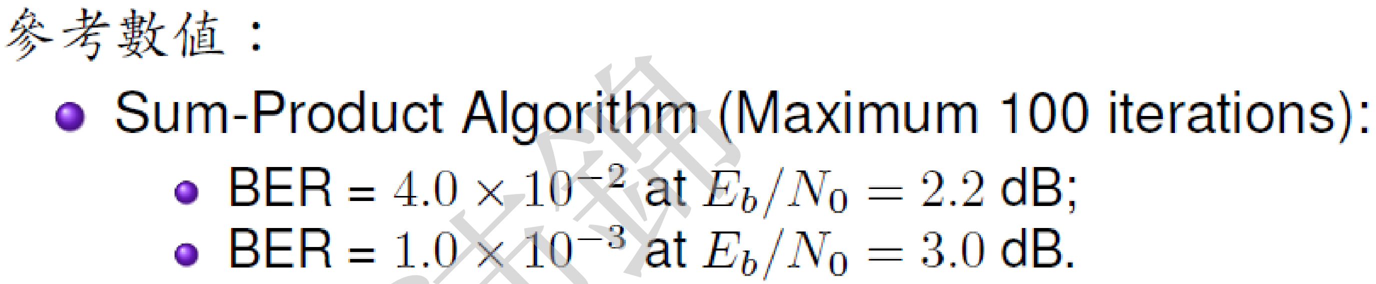
SNR (dB)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6
No. decoded block errors	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
No. decoded blocks	50	50	50	50	50	50	50	50	50	53	58	73	138	216	470	1738	3167	11466	42166
block error rate (BLER)	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	9.43E-01	8.62E-01	6.85E-01	3.62E-01	2.31E-01	1.06E-01	2.88E-02	1.58E-02	4.36E-03	1.19E-03

模擬效能圖

- number of maximum block errors = 50 (fixed)



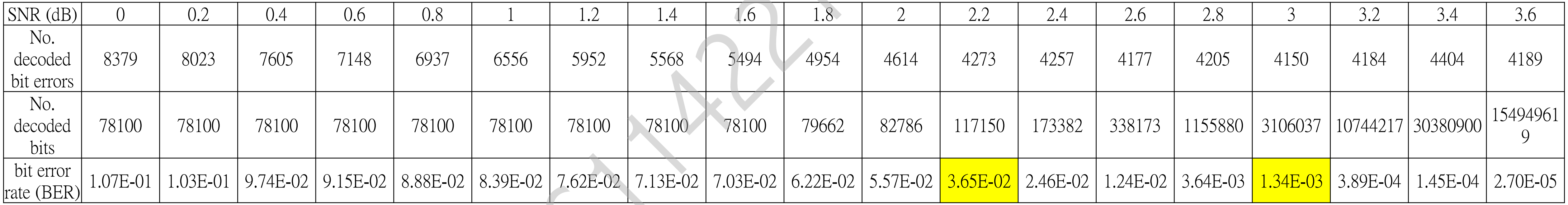
模擬數據



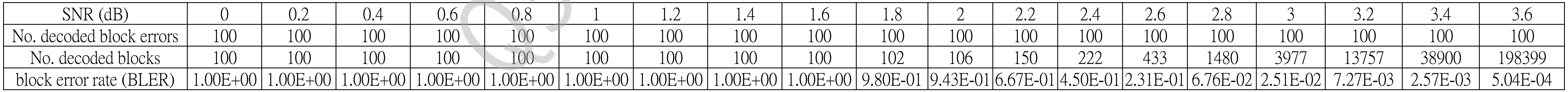
參考數值：

- Sum-Product Algorithm (Maximum 100 iterations):
  - BER = 4.0×10^{-2} at $E_b/N_0 = 2.2$ dB;
  - BER = 1.0×10^{-3} at $E_b/N_0 = 3.0$ dB.

- number of maximum block errors = 100
- number of maximum iterations = 100 (fixed)



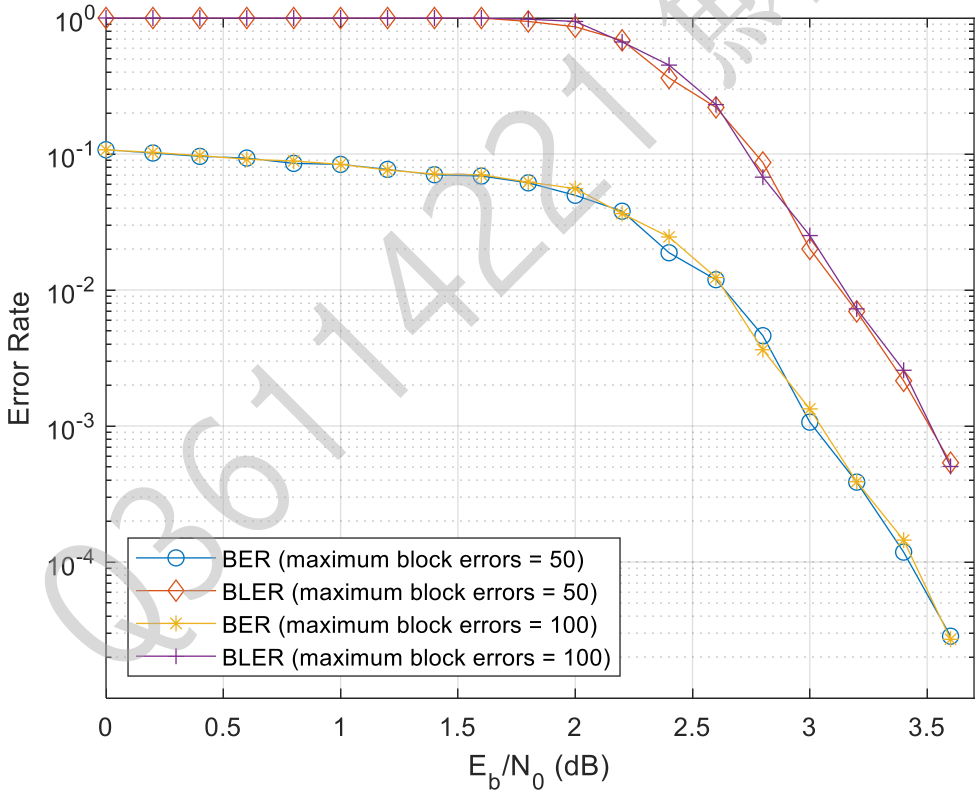
SNR (dB)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6
No. decoded bit errors	8379	8023	7605	7148	6937	6556	5952	5568	5494	4954	4614	4273	4257	4177	4205	4150	4184	4404	4189
No. decoded bits	78100	78100	78100	78100	78100	78100	78100	78100	78100	79662	82786	117150	173382	338173	1155880	3106037	10744217	30380900	154949619
bit error rate (BER)	1.07E-01	1.03E-01	9.74E-02	9.15E-02	8.88E-02	8.39E-02	7.62E-02	7.13E-02	7.03E-02	6.22E-02	5.57E-02	3.65E-02	2.46E-02	1.24E-02	3.64E-03	1.34E-03	3.89E-04	1.45E-04	2.70E-05



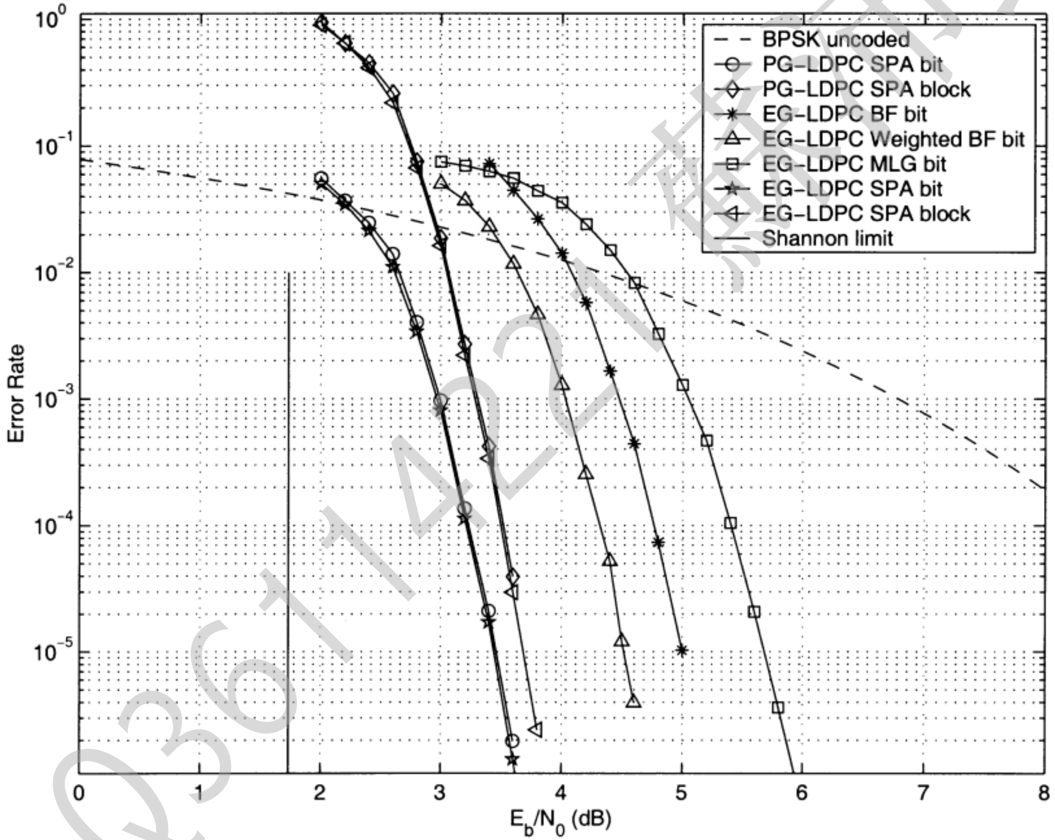
SNR (dB)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6
No. decoded block errors	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
No. decoded blocks	100	100	100	100	100	100	100	100	100	102	106	150	222	433	1480	3977	13757	38900	198399
block error rate (BLER)	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	9.80E-01	9.43E-01	6.67E-01	4.50E-01	2.31E-01	6.76E-02	2.51E-02	7.27E-03	2.57E-03	5.04E-04

模擬效能圖

- number of maximum iterations = 100 (fixed)



模擬效能圖 (Reference)*



Bit- and block-error probabilities of the type-I 2-D (1023,781) EG-LDPC code and (1057,813) PG-LDPC code based on different decoding algorithms. *

* Y. Kou, S. Lin and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," in *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2711-2736, Nov. 2001, doi: 10.1109/18.959255.