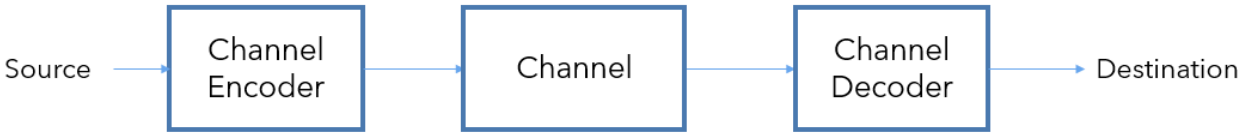
Convolutional Code Decoder

109061217 林峻霆

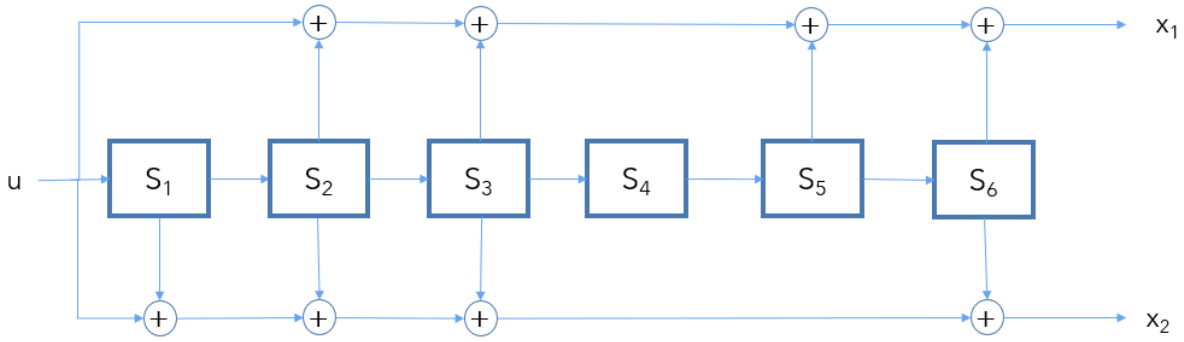
1. System Design

本次模擬的系統流程圖如下：



- Encoder

Encoder 的設計如下圖所示，我們需要一個 6-bit shift register



另外，在將 x = [x_1, x_2]輸入 channel 前需要先經過調變，本次模擬採用的是 BPSK，也就是 $0 \rightarrow 1,\ 1 \rightarrow -1$。

- Channel：AWGN channel

- Decoder

在 Decoder 端我們要先進行解調，可以粗略分為兩種：Hard decision 和 Soft decision。Hard Decision 就是根據收到的訊號進行分類，若大於 0 則歸類為 1，小於 0 則歸類為-1，在傳入 Viterbi Decoder；Soft decision 則直接將收到的傳入 Viterbi Decoder，接著就按照 Viterbi Algorithm 去進行 Decoding。要注意的是，為了防止 overflow，我在每一輪更新每個 state 的 metric 後，計算所有 state 的 metric 平均值，並將其減去。

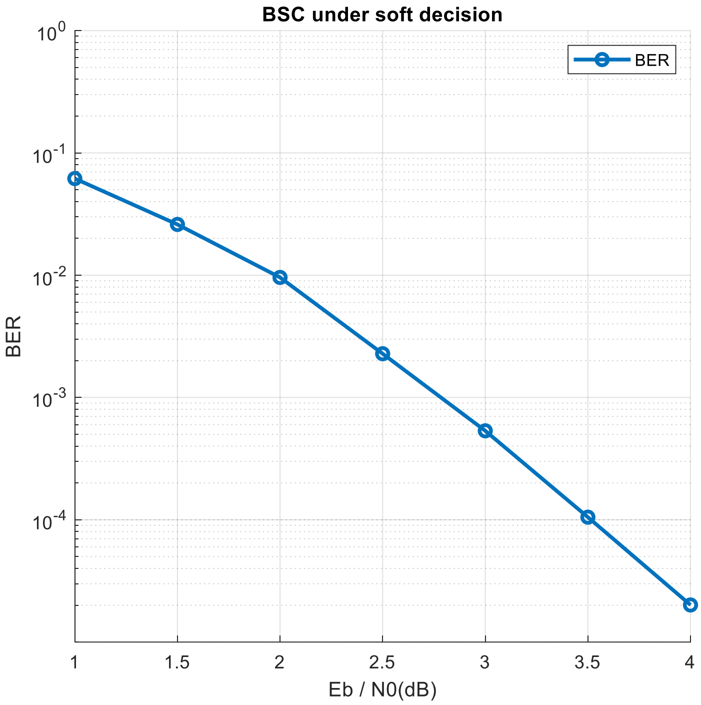
2. Result

以下考慮在不同狀況下，進行解碼成效的比較：

a. Best State

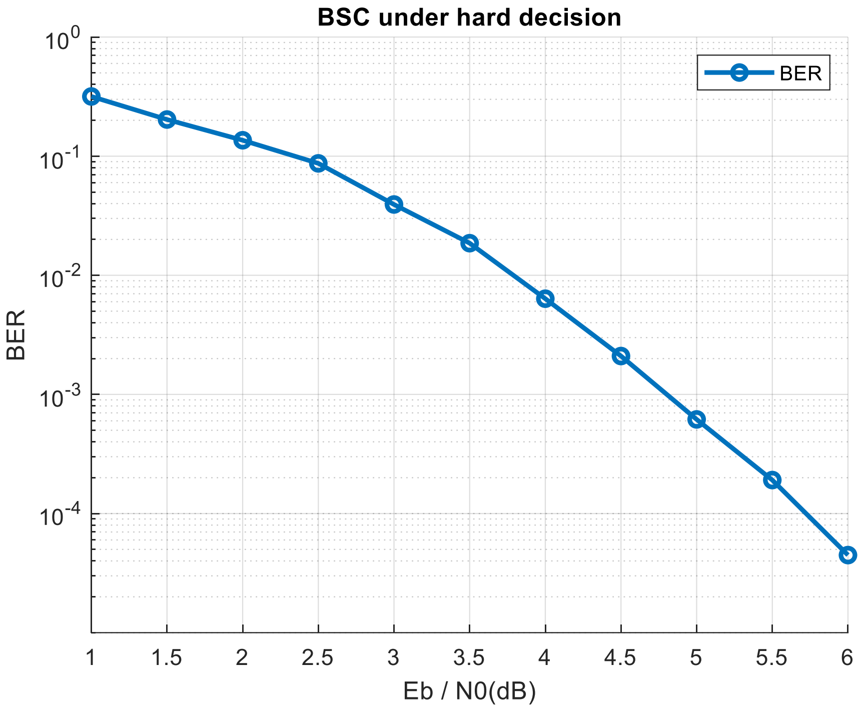
- Soft Decision

soft decision			
SNR(dB)	# of bits	# of error bit	BER
1	16203	1000	6.17E-02
1.5	38478	1000	2.60E-02
2	104273	1000	9.59E-03
2.5	438056	1000	2.28E-03
3	1872272	1000	5.34E-04
3.5	9525619	1000	1.05E-04
4	49765296	1000	2.01E-05



- Hard Decision

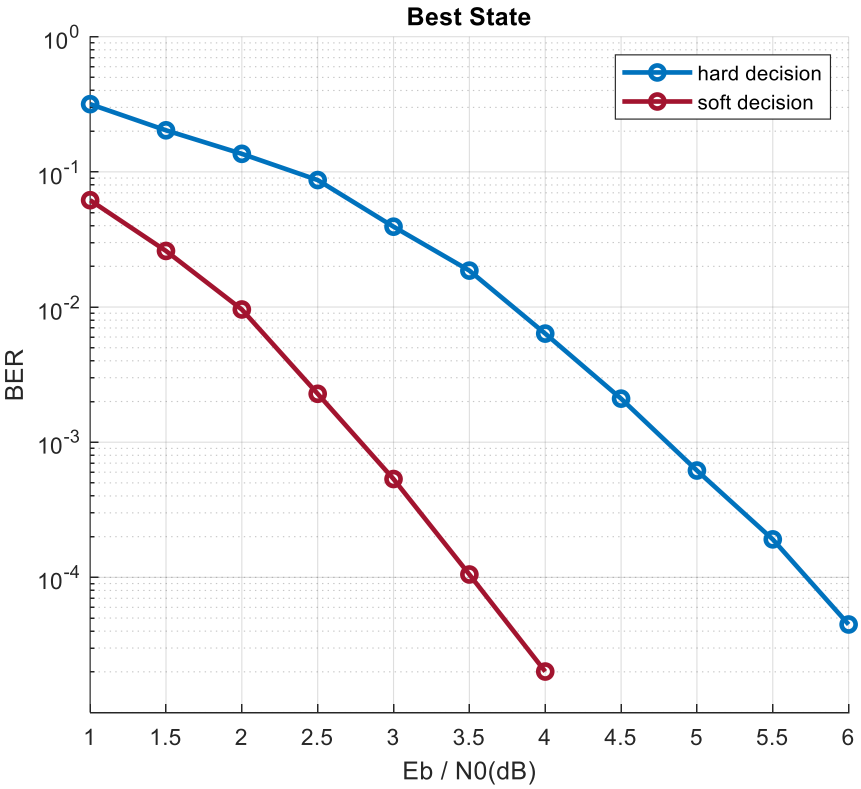
soft decision			
SNR(dB)	# of bits	# of error bit	BER
1	3153	1000	3.17E-01
1.5	4909	1000	2.04E-01
2	7368	1000	1.36E-01
2.5	11488	1000	8.70E-02
3	25477	1000	3.93E-02
3.5	53659	1000	1.86E-02
4	157130	1000	6.36E-03
4.5	476734	1000	2.10E-03
5	1621436	1000	6.17E-04
5.5	5228549	1000	1.91E-04
6	22333422	1000	4.48E-05



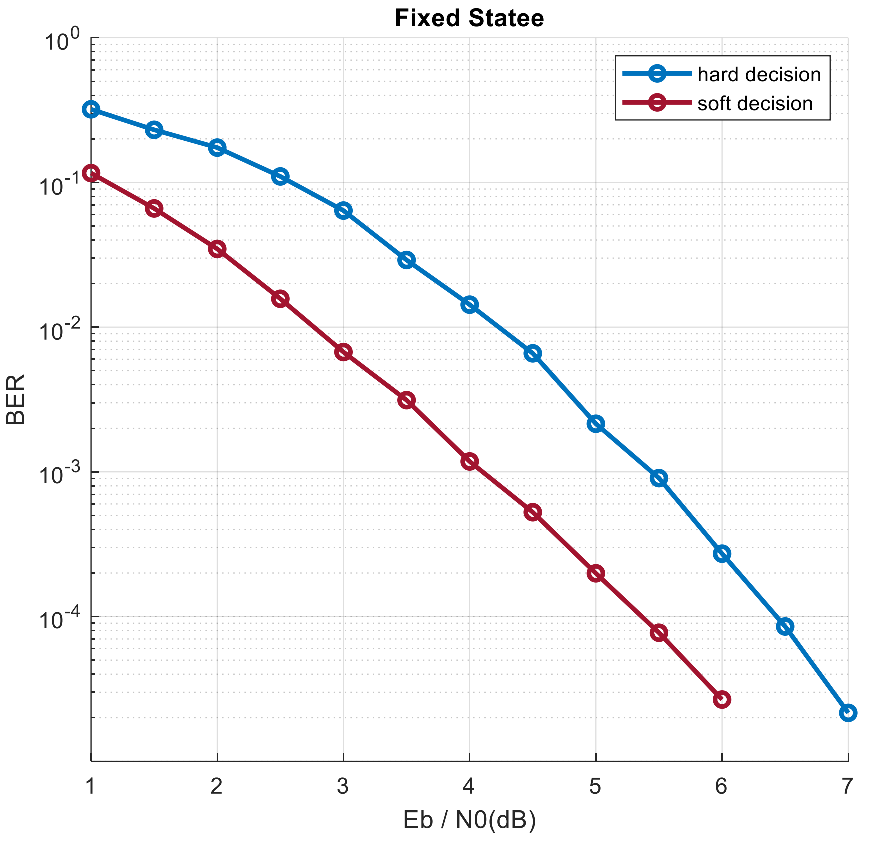
比較 Hard decision 與 Soft decision，兩者在達到 BER = 10^{-5} 時所需的 SNR 約有 2dB 差距。

b. Best State vs. Fixed State vs. Majority Vote

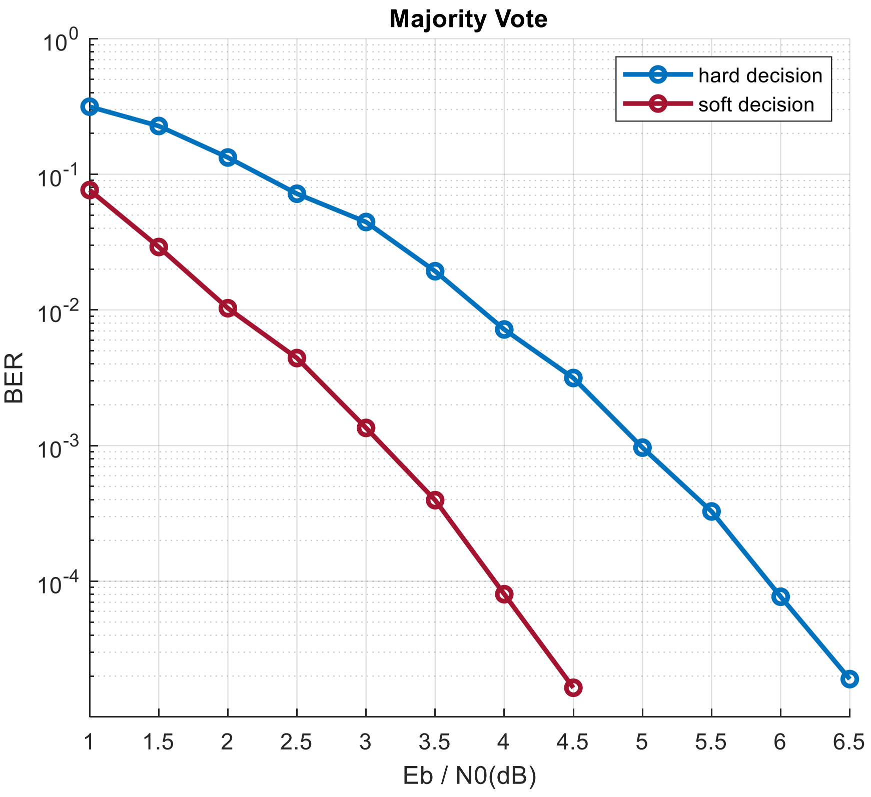
- Best State



- Fixed State (always pick the all zero state)



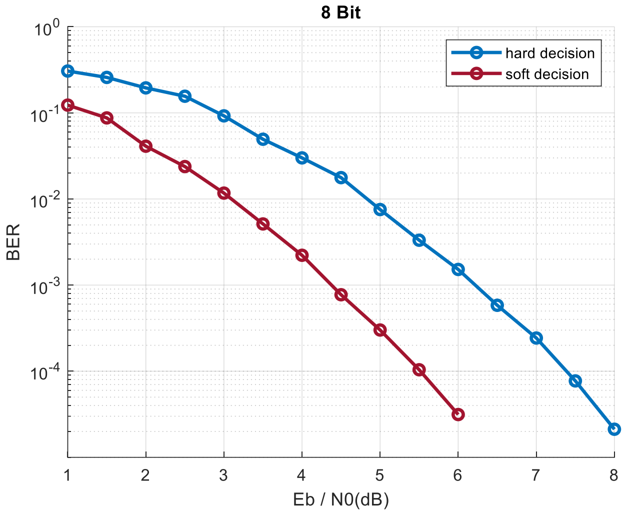
- Majority Vote



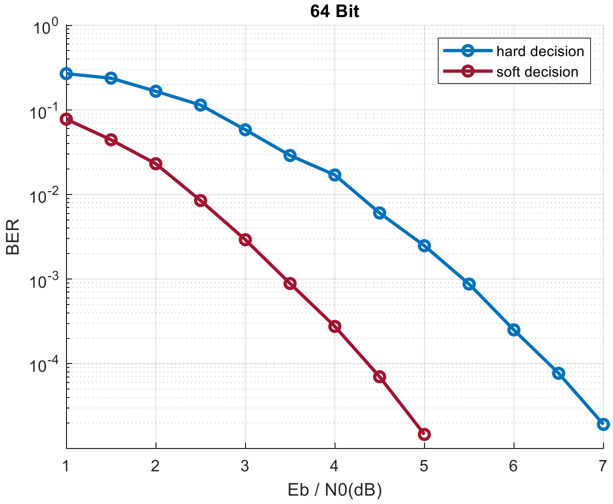
比較三者，可以發現三者的 performance 相比 Best State > Majority Vote > Fixed State。另外在採用 Best State 和 Majority Vote 時，Hard Decision 都比 Soft Decision 高 2dB；在 Fixed State 時，Soft Decision 與 Hard Decision 僅有 1dB 的差距。

c. Truncated Bits

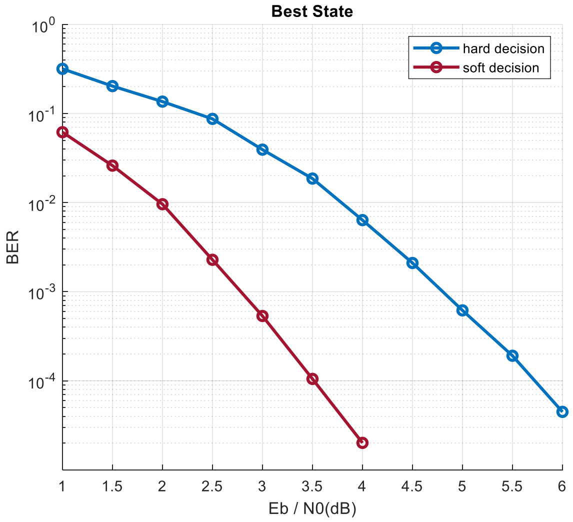
- 8 bits



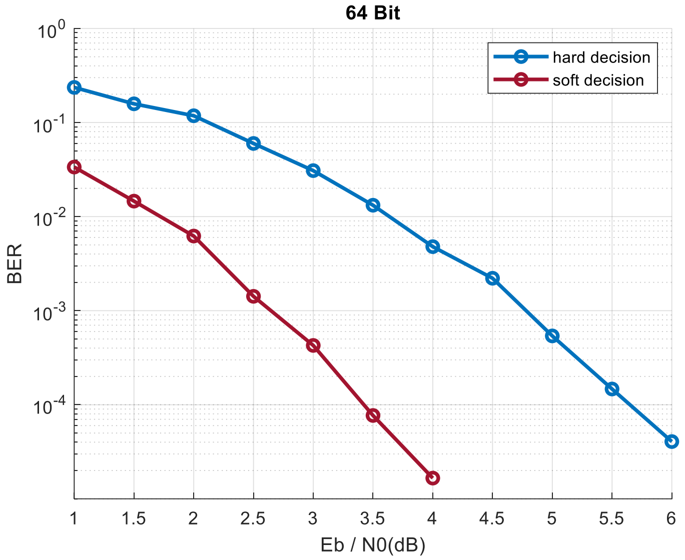
- 16 bits



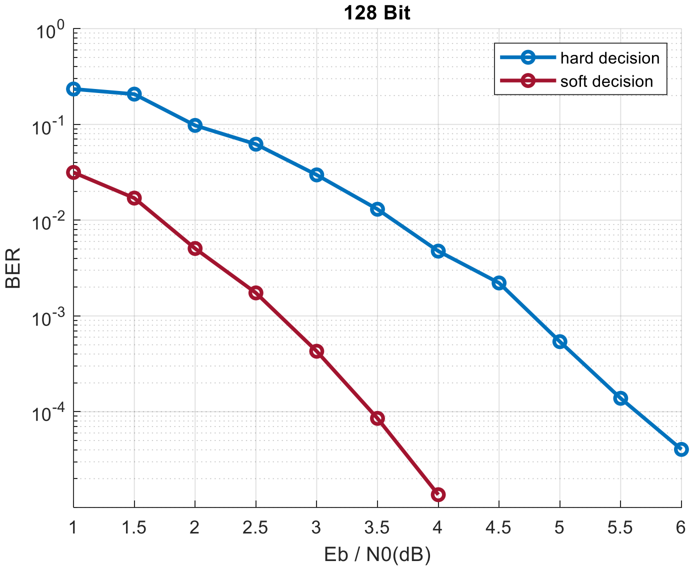
- 32 bits



- 64 bits



- 128 bits

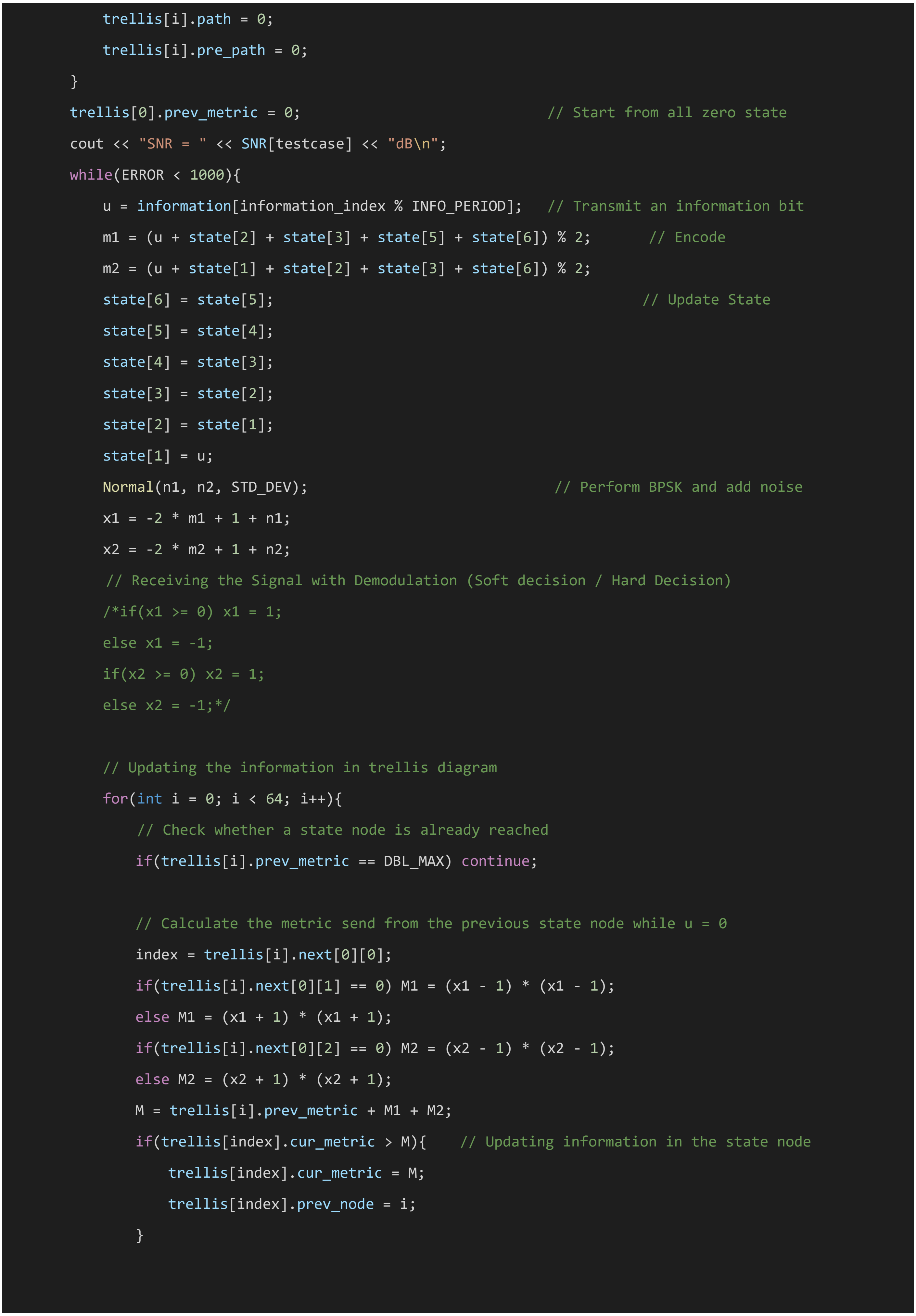


理論上來說我們應該等到整個 bit stream 都傳完才進行 output，但在現實應用上，transmitter 可能會不斷傳送訊號過來，因此實際上我們會對收到的資訊做 truncated。Truncated length 也會影響到解碼的成效，觀察上述幾張圖可發現，當 truncation length 越長，解碼的成效越好，8-bit 與 32-bit truncation length 有約 2dB 的差異。另外，也可以拉線，當 truncation length 持續上升，performance 的優化幅度越小，32-bit、64-bit 以及 128-bit 的 performance 近乎沒有區別。

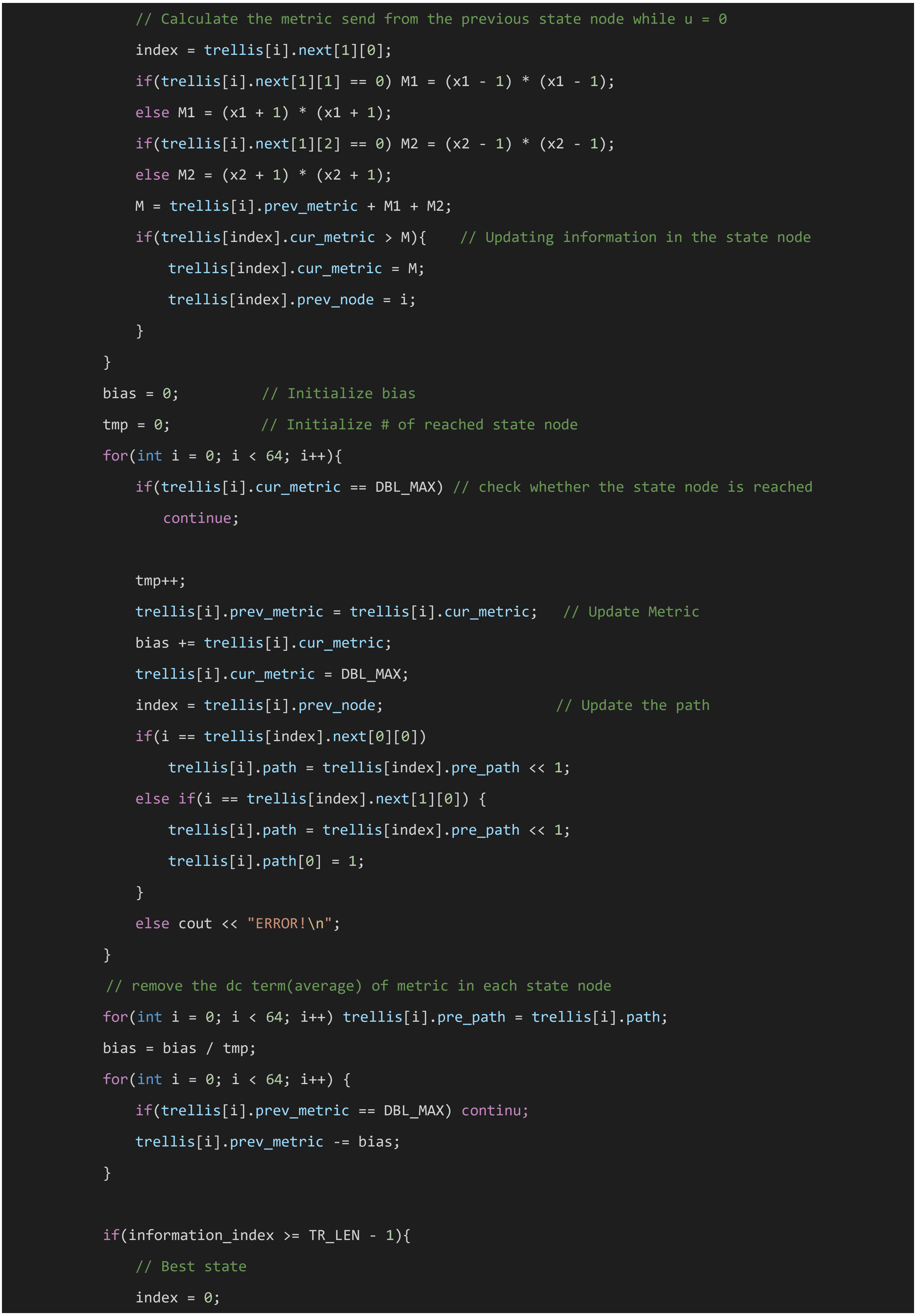
3. Program

```
#include <iostream>

#include <iomanip>

#include <bitset>

#include <cmath>

#include <limits>

#include <float.h>

#include <stdlib.h>

#include <vector>

#include <fstream>


#define TR_LEN 32

#define INFO_PERIOD 63


using namespace std;


const long long int para_1 = 4101842887655102017LL;

const long long int para_2 = 2685821657736338717LL;

const double para_3 = 5.42101086242752217E-20;

unsigned long long int SEED = 14;

unsigned long long int RANV;

int RANI = 0;


double Ranq1(){

    if(RANI == 0){

        RANV = SEED ^ para_1;

        RANV ^= RANV >> 21;

        RANV ^= RANV << 35;

        RANV ^= RANV >> 4;

        RANV *= para_2;

        RANI++;

    }

    RANV ^= RANV >> 21;

    RANV ^= RANV << 35;

    RANV ^= RANV >> 4;

    return RANV * para_2 * para_3;

}

void Normal(double& n1, double& n2, double std_dev){

    double x1, x2, s;

    do{

        x1 = Ranq1();
        x2 = Ranq1();
        x1 = 2 * x1 - 1;
        x2 = 2 * x2 - 1;
        s = x1 * x1 + x2 * x2;
    } while (s >= 1.0);
    n1 = std_dev * x1 * sqrt(-2 * log(s) / s);
    n2 = std_dev * x2 * sqrt(-2 * log(s) / s);
}

// each node represent a state, storing the current information and previous information
typedef struct Node{
    double prev_metric = DBL_MAX;              // metric in previous state
    double cur_metric = DBL_MAX;               // metric in current state
    int prev_node = 100000;                    // index of previous node
    vector<vector<int>> next;                  // next[0][0] = index of next state when u = 0
                                               // next[1][0] = index of next state when u = 1
    bitset<TR_LEN> pre_path {0};              // each bit represent u that was sent previously, in previous state
    bitset<TR_LEN> path {0};                   // each bit represent u that was sent previously, in current state
} node;

double SNR[20] = {1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8};

int main(void){
    int ERROR = 0;                            // # of bit error
    int u, tmp;                               // temporary storage
    int m1, m2;                               // [m1, m2] = uG
    int num_of_0, num_of_1;                    // # of 0 and 1 while we using the Majority Vote
    int truncated_len = 0;
    double x1, x2;                            // Signal point after BPSK and adding noise
    double STD_DEV;                           // standard deviation correspond to SNR to generate Normal r.v
    double bias;                              // the average of the metric to prevent overflow
    int index = 0;
    long long int information_index = 0;      // # of transmitted information bit
    double n1, n2;                            // Noise
    double M1, M2, M;
    ofstream outFile;
    node* trellis = new node[64];
    vector<int> information;                   // information bits
    vector<int> state;                         // current state
```

```
// [m1, m2] = uG
// m1 = u + s[2] + s[3] + s[5] + s[6]
// m2 = u + s[1] + s[2] + s[3] + s[6]
for(int i = 0; i < 64; i++){
    m1 = 0;
    m2 = 0;
    if(i & 1) m2 = m2 + 1;
    if(i & 2) {
        m1 = m1 + 1;
        m2 = m2 + 1;
    }
    if(i & 4) {
        m1 = m1 + 1;
        m2 = m2 + 1;
    }
    if(i & 16) m1 += 1;
    if(i & 32) {
        m1 = m1 + 1;
        m2 = m2 + 1;
    }
    index = i * 2;
    if(index >= 64) index -= 64;
    trellis[i].next.push_back({index, m1 % 2, m2 % 2});
    trellis[i].next.push_back({index + 1, (m1 + 1) % 2, (m2 + 1) % 2});
}
state.assign(7, 0);
information.assign(63, 0);

// generate information
// u[l + 6] = u[l + 1] + u[l] for l >= 0. The period of the sequence is 63
information[0] = 1;
for(int i = 6; i < INFO_PERIOD; i++) information[i] = information[i - 6] ^ information[i - 5];

// Start Testing for each SNR
for(int testcase = 0; testcase < 15; testcase++){
    STD_DEV = sqrt(pow(10, -SNR[testcase] / 10));         // Calculate standard deviation
    ERROR = 0;                                           // Initialize # of error
    information_index = 0;                              // Initialize # of transmitted bit
    for(int i = 0; i < 64; i++){                         // Initialize each state node
        trellis[i].prev_metric = DBL_MAX;
        trellis[i].cur_metric = DBL_MAX;
```



```
        trellis[i].path = 0;
        trellis[i].pre_path = 0;
    }
    trellis[0].prev_metric = 0;                             // Start from all zero state
    cout << "SNR = " << SNR[testcase] << "dB\n";
    while(ERROR < 1000){
        u = information[information_index % INFO_PERIOD];   // Transmit an information bit
        m1 = (u + state[2] + state[3] + state[5] + state[6]) % 2;       // Encode
        m2 = (u + state[1] + state[2] + state[3] + state[6]) % 2;
        state[6] = state[5];                                           // Update State
        state[5] = state[4];
        state[4] = state[3];
        state[3] = state[2];
        state[2] = state[1];
        state[1] = u;
        Normal(n1, n2, STD_DEV);                             // Perform BPSK and add noise
        x1 = -2 * m1 + 1 + n1;
        x2 = -2 * m2 + 1 + n2;
        // Receiving the Signal with Demodulation (Soft decision / Hard Decision)
        /*if(x1 >= 0) x1 = 1;
        else x1 = -1;
        if(x2 >= 0) x2 = 1;
        else x2 = -1;*/

        // Updating the information in trellis diagram
        for(int i = 0; i < 64; i++){
            // Check whether a state node is already reached
            if(trellis[i].prev_metric == DBL_MAX) continue;

            // Calculate the metric send from the previous state node while u = 0
            index = trellis[i].next[0][0];
            if(trellis[i].next[0][1] == 0) M1 = (x1 - 1) * (x1 - 1);
            else M1 = (x1 + 1) * (x1 + 1);
            if(trellis[i].next[0][2] == 0) M2 = (x2 - 1) * (x2 - 1);
            else M2 = (x2 + 1) * (x2 + 1);
            M = trellis[i].prev_metric + M1 + M2;
            if(trellis[index].cur_metric > M){    // Updating information in the state node
                trellis[index].cur_metric = M;
                trellis[index].prev_node = i;
            }
```



```
        // Calculate the metric send from the previous state node while u = 0
        index = trellis[i].next[1][0];
        if(trellis[i].next[1][1] == 0) M1 = (x1 - 1) * (x1 - 1);
        else M1 = (x1 + 1) * (x1 + 1);
        if(trellis[i].next[1][2] == 0) M2 = (x2 - 1) * (x2 - 1);
        else M2 = (x2 + 1) * (x2 + 1);
        M = trellis[i].prev_metric + M1 + M2;
        if(trellis[index].cur_metric > M){    // Updating information in the state node
            trellis[index].cur_metric = M;
            trellis[index].prev_node = i;
        }
    }
    bias = 0;          // Initialize bias
    tmp = 0;           // Initialize # of reached state node
    for(int i = 0; i < 64; i++){
        if(trellis[i].cur_metric == DBL_MAX) // check whether the state node is reached
           continue;

        tmp++;
        trellis[i].prev_metric = trellis[i].cur_metric;   // Update Metric
        bias += trellis[i].cur_metric;
        trellis[i].cur_metric = DBL_MAX;
        index = trellis[i].prev_node;                    // Update the path
        if(i == trellis[index].next[0][0])
            trellis[i].path = trellis[index].pre_path << 1;
        else if(i == trellis[index].next[1][0]) {
            trellis[i].path = trellis[index].pre_path << 1;
            trellis[i].path[0] = 1;
        }
        else cout << "ERROR!\n";
    }
    // remove the dc term(average) of metric in each state node
    for(int i = 0; i < 64; i++) trellis[i].pre_path = trellis[i].path;
    bias = bias / tmp;
    for(int i = 0; i < 64; i++) {
        if(trellis[i].prev_metric == DBL_MAX) continu;
        trellis[i].prev_metric -= bias;
    }

    if(information_index >= TR_LEN - 1){
        // Best state
        index = 0;
```

```
                // Find the best state result
                double best_metric = trellis[0].prev_metric;
                for(int i = 1; i < 64; i++){
                    if(trellis[i].prev_metric < best_metric){
                        index = i;
                        best_metric = trellis[i].prev_metric;
                    }
                }
                // Check if there's error
                if(trellis[index].path[TR_LEN - 1] != information[(information_index - TR_LEN + 1) %
INFO_PERIOD]) ERROR++;

                // Fixed State
                //if(trellis[0].path[TR_LEN - 1] != information[(information_index - TR_LEN + 1) %
INFO_PERIOD]) ERROR++;

                // Majority vote
                /*num_of_0 = 0;
                num_of_1 = 1;
                for(int i = 0; i < 64; i++){
                    if(trellis[i].path[31] == 0) num_of_0++;
                    else num_of_1++;
                }
                if(num_of_0 >= num_of_1 && information[(information_index - 31) % INFO_PERIOD] == 1)
ERROR++;
                else if(num_of_0 < num_of_1 && information[(information_index - 31) % INFO_PERIOD] == 0)
ERROR++;*/
            }
            information_index = (information_index + 1);
        }
        cout << "ERROR = " << ERROR << " Number of bits = " << information_index - 30 << endl;
        cout << "BER = " << ERROR * 1.0 / (information_index - 30) << endl;
    }
    return 0;
}
```