Hamming Code

Project #3 | NCUE CSIE|
S0954010 謝宥宣 | S0954016戴育琪 | S0954017 蔡蔓萱 | S0954020 邱靖云 | S0954026 李妍瑄

Project Goals

- To encode a 4-bit binary data.
- To transfer the encoded data.
- To decode the transferred data back to the original 4-bit data
  - If error occurred during transfer, correct the error

Hamming code

- A set of error-correction codes that can be used to detect and correct the errors that may occur when data is transferred from sender to receiver.

Source: Hamming Code in Computer Network - GeeksforGeeks

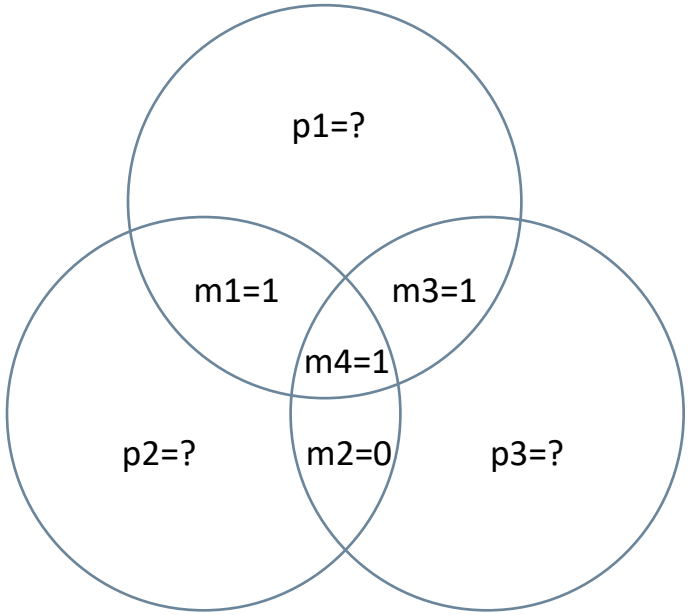
How does hamming code work- Encoding (Even parity)

- Eg: Data bit = (1, 0, 1, 1)

m1	m2	m3	m4	p1	p2	p3
1	**0**	**1**	**1**	**?**	**?**	**?**

  - p1 = 1
  - P2 = 0
  - P3 = 0

- Codeword = (1, 0, 1, 1, 1, 0, 0)



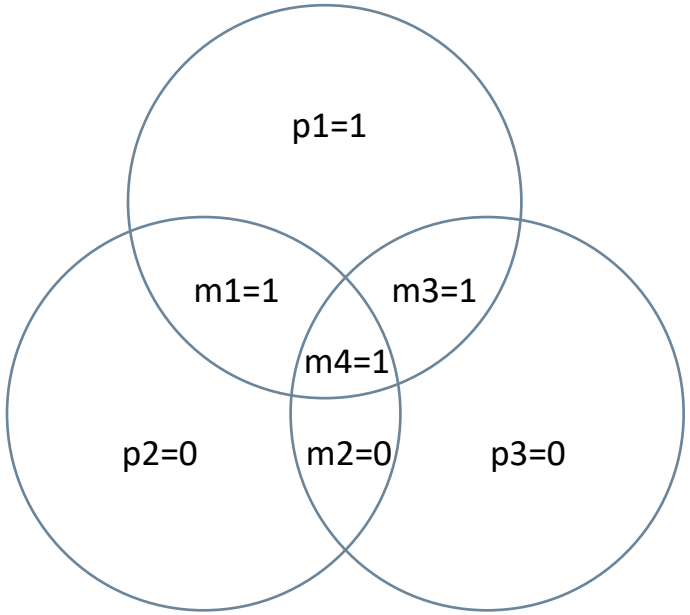
Source: Hamming Codes Encoding and Decoding - YouTube

How does hamming code work- Decoding (Even parity)

- Eg: Codeword = (1, 0, 1, 1, 1, 0, 0)

m1	m2	m3	m4	p1	p2	p3
1	**0**	**1**	**1**	**1**	**0**	**0**

- All three circles are OK.



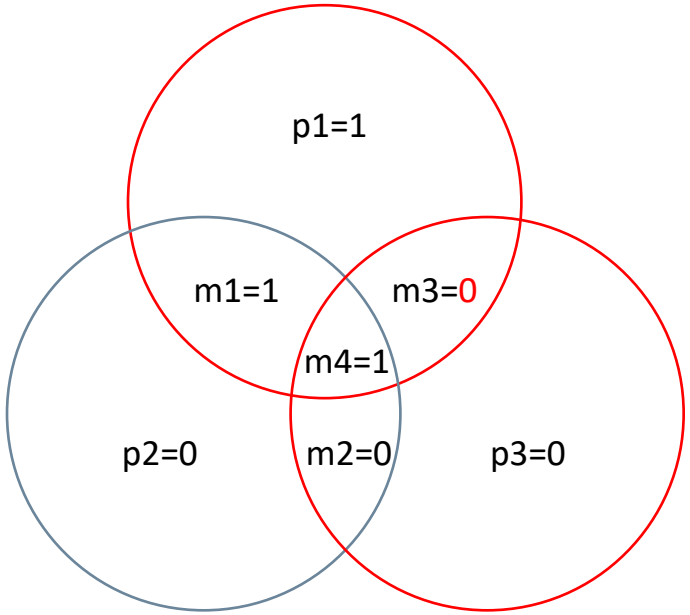
Source: Hamming Codes Encoding and Decoding - YouTube

How does hamming code work- Decoding (Even parity)

- Eg: Codeword = (1, 0, 1, 1, 1, 0, 0)

m1	m2	m3	m4	p1	p2	p3
1	**0**	**0**	**1**	**1**	**0**	**0**

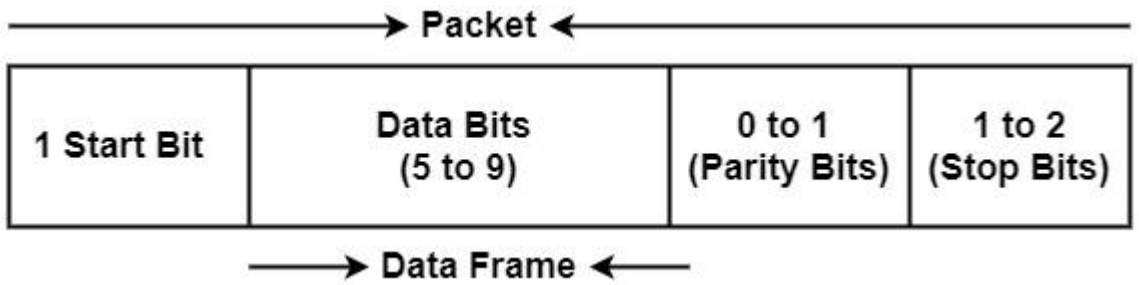
- 1st, 3rd circle are NG
- 2nd circle is OK.
- If we invert m3, all three circles are OK.
- Error corrected!



Source: Hamming Codes Encoding and Decoding - YouTube

Transfer- UART

- UART (Universal Asynchronous Receiver/Transmitter)
- a popular device-to-device communication protocol



- Includes a start bit, a stop bit, a parity bit(optional), and the data bits.

Source: What is UART? (tutorialspoint.com)

信號框架	位元數	位元值(正邏輯)	功　能
標誌位元 Mask Bit	-	1(Mask)	表示電路停滯中。
起始位元 Start Bit	1	0(Space)	通知接收者的同步訊號，計時由1➔0開始，每次接收皆由此位元取得同步。
資料位元 Data Bit	5、6、7、8	依資料而定	要傳送或接收的資料位元，依系統目的的不同，資料長度有5、6、7、8位元可選用。
同位位元 Parity Bit	1或0	-	(None)爲加快傳輸速度，通常會選不使用同位位元。
		0或1	使用奇同位(Odd Parity)，即資料位元與同位位元所含的位元1的個數爲奇數個；或偶同位(Even Parity)，即資料位元與同位位元所含的位元1的個數爲偶數個，以爲做位元檢查之用，可簡易判斷位元傳輸中是否有受干擾。
		0(Space)	此位元永遠設0(Space)，較少用。
		1(Mask)	此位元永遠設1(Mask)，較少用。
停止位元 Stop Bit	1、1.5、2	1(Mask)	此次傳輸已結束，一方面做爲下一筆傳輸之間隔符號。其中1.5位元是爲資料長度爲5時所專用。

Encoder

```
module encoder ( data, ham );
    input [3:0] data;
    output [6:0] ham;
    wire w1,w2,w3;
    assign w1 = data[0] ^ data[1] ^ data[3]; //根據文氏圖在輸出中的第二個bit與第三、四、六個bit的1總數需為偶數
    assign w2 = data[0] ^ data[2] ^ data[3]; //根據文氏圖在輸出中的第一個bit與第三、五、六個bit的1總數需為偶數
    assign w3 = data[1] ^ data[2] ^ data[3]; //根據文氏圖在輸出中的第零個bit與第四、五、六個bit的1總數需為偶數
    assign ham = {data, w1, w2, w3};
endmodule
```

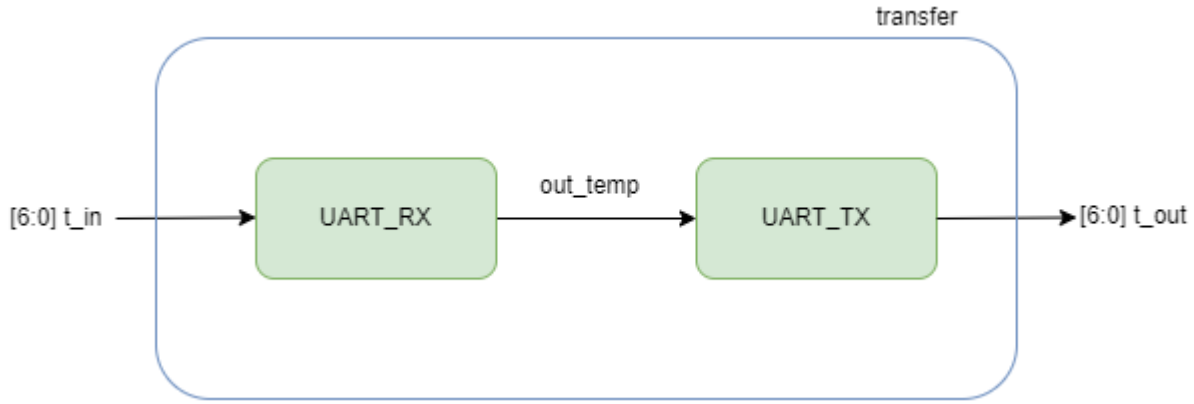
Decoder (1/2)

```
module decoder ( input clk, input [6:0] codeword, output reg [3:0] correctword );

 //收到的codeword :{data[3],data[2],data[1],data[0],h1,h2,h3}

 wire [2:0] check; //codeword 有1的index做XOR

 assign check[0] = codeword[4] ^ codeword[3] ^codeword[2] ^ codeword[6];  //3‘d1~3’d7 之間 [0]是1的 1.3.5.7 依解碼格式對照input順序為 2.3.4.6

 assign check[1] = codeword[3] ^ codeword[5] ^ codeword[1] ^ codeword[6];  //3'd1~3'd7 之間 [1]是1的 2.3.6.7  =>1.3.5.6

 assign check[2] = codeword[5] ^ codeword[4] ^ codeword[6] ^ codeword[0];  //3'd1~3'd7 之間 [2]是1的 4.5.6.7  =>0.4.5.6
```

Decoder (2/2)

```
always@(posedge clk) begin  //更正錯誤的 bit
case (check)  //check=3 data[0]有誤 ，check=5 data[1]有誤 ，check=6 data[2]有誤 ，check=7 data[3]有誤，將其反轉
            3'b111 :   correctword = {~codeword[6],codeword[5:3]};
            3'b110 :   correctword = {codeword[6],~codeword[5],codeword[4:3]};
            3'b101 :   correctword = {codeword[6:5],~codeword[4],codeword[3]};
            3'b011 :   correctword = {codeword[6:4],~codeword[3]};
            default :  correctword = codeword[6:3];
        endcase
  end
endmodule
```

Transfer



Transfer - UART_TX.v (1/5)

```
module UART_TX( input i_clk, input i_rst, input [6:0]i_TX_Byte, output o_TX_Bit );

    parameter [1:0] TX_START_ST=2'd0;  //信號傳輸時，電路之狀態 1. TX_START_ST: 傳送start信號

    parameter [1:0] TX_DATA_ST=2'd1;  //2. TX_DATA_ST: 傳送資料信號

    parameter [1:0] TX_STOP_ST=2'd2;  //3. TX_STOP_ST: 傳送stop信號

    reg [1:0] r_SM_Main=TX_START_ST;  //1. r_SM_Main: 紀錄電路之狀態

    reg [2:0] r_Byte_Idx=3'd0;  //2. r_Byte_Idx: 資料信號之bit index

    reg r_TX_Bit=1'b1;     //3. r_TX_Bit: 傳送至輸出o_TX_Bit之信號
```

Transfer - UART_TX.v (2/5)

```
assign o_TX_Bit=r_TX_Bit;

always@(posedge i_clk or posedge i_rst) begin

 if(i_rst==1'b1) begin

  r_Byte_Idx=3'd0;

  r_TX_Bit=1'b1;

  r_SM_Main=TX_START_ST;

 end
```

Transfer - UART_TX.v (3/5)

```
else begin
  case(r_SM_Main)
   TX_START_ST: begin
    r_Byte_Idx=3'd0;
    r_TX_Bit=1'b0;   //啟動資料傳輸，start信號設為0
    r_SM_Main=TX_DATA_ST; //跳至資料傳輸狀態
   end
```

Transfer - UART_TX.v (4/5)

```
TX_DATA_ST: begin
   r_TX_Bit=i_TX_Byte[r_Byte_Idx];  //傳送資料信號之某個bit
   if(r_Byte_Idx<3'd6) begin      //若資料信號尚未傳完，則在下一週期繼續傳送下一個bit,
    r_Byte_Idx=r_Byte_Idx+1'b1;   //電路狀態維持在TX_DATA_ST
    r_SM_Main=TX_DATA_ST;
   end
   else
    r_SM_Main=TX_STOP_ST;     //若資料信號已傳完 則在下一週期進入TX_STOP_ST狀態
  end
```

Transfer - UART_TX.v (5/5)

```
TX_STOP_ST:begin
    r_TX_Bit=1'b1;          //1. 結束資料傳輸，stop信號設為1
    r_SM_Main=TX_START_ST;   //2. 重新回到TX_START_ST狀態，開啟下一次資料傳輸
   end
default:
    r_SM_Main=TX_START_ST;
 endcase
 end
end
endmodule
```

Transfer - UART_RX.v (1/5)

```
module UART_RX( input i_clk, input i_rst, input i_RX_Bit, output [6:0] o_RX_Byte);

 parameter [1:0] RX_START_ST = 2'd0;   //信號傳輸時，電路之狀態1. RX_START_ST: 接收start信號

 parameter [1:0] RX_DATA_ST = 2'd1;    //2. RX_DATA_ST: 接收資料信號

 parameter [1:0] RX_STOP_ST = 2'd2;    //3. RX_STOP_ST: 接收stop信號

reg [1:0] r_SM_Main = RX_START_ST;     //1. r_SM_Main: 紀錄電路之狀態

reg [2:0] r_Byte_Idx = 3'd0;           //2. r_Byte_Idx: 資料信號之bit index

reg [6:0] r_RX_Byte = 7'd0;          //3. r_RX_Byte: 傳送至輸出o_RX_Byte之信號
```

Transfer - UART_RX.v (2/5)

```
assign o_RX_Byte = r_RX_Byte;

always@ (posedge i_clk or posedge i_rst) begin

 if (i_rst  ==  1'b1) begin

  r_Byte_Idx = 3'd0;

  r_RX_Byte = 7'd0;

  r_SM_Main = RX_START_ST;

 end
```

Transfer - UART_RX.v (3/5)

```
else begin
  case(r_SM_Main)
   RX_START_ST:  begin
       r_Byte_Idx = 3'd0;
       if( i_RX_Bit == 1'b0)    //若接收到start信號為0，資料傳輸啟動
            r_SM_Main = RX_DATA_ST;  //跳至資料傳輸狀態
       else
            r_SM_Main = RX_START_ST;  //否則維持在等待start的狀態
   end
```

Transfer - UART_RX.v (4/5)

```
RX_DATA_ST: begin
    r_RX_Byte[r_Byte_Idx] = i_RX_Bit;  //接收資料信號之某個bit
    if( r_Byte_Idx < 3'd6) begin     //若資料信號尚未傳完
        r_Byte_Idx=r_Byte_Idx+1'b1;  //則在下一週期繼續接收下一個bit
        r_SM_Main=RX_DATA_ST;       //電路狀態維持在RX_DATA_ST
    end
    else                        //若資料信號已傳完
       r_SM_Main=RX_STOP_ST;      //則在下一週期進入RX_STOP_ST狀態
   end
```

Transfer - UART_RX.v (5/5)

```
RX_STOP_ST: begin          //結束資料傳輸，重新回到RX_START_ST狀態，等待下一次傳輸
    r_SM_Main = RX_START_ST;
   end
   default:
        r_SM_Main=RX_START_ST;
  endcase
 end
end
endmodule
```

Transfer top module- transfer.v

```
module transfer( input rst, input clk, input [6:0]t_in, output reg[6:0]t_out );
 wire out_temp;
 wire[6:0]find_correct;
 UART_TX u0(clk,rst,t_in,out_temp);
 UART_RX u1(clk,rst,out_temp,find_correct);
 always @(find_correct)begin
     if(t_in==find_correct)    //只輸出全部bit傳送完畢與input完全相同的結果
          t_out=find_correct;
 end
endmodule
```

Frequency divider – freq_div (1/3)

```
module freq_div(clk, rst, clk_out);

        input clk, rst;

        output clk_out;

        parameter N = 7;

        parameter WIDTH = 3;

        reg [WIDTH-1:0] cnt_pos, cnt_neg;

        reg clk_pos, clk_neg;

        assign clk_out = (N == 1)? clk: (N[0] == 0)? clk_pos:clk_neg;
```

Frequency divider – freq_div (2/3)

```
always@ (posedge clk or posedge rst)
        begin
                if(rst)
                        cnt_pos = 0;
                else if(cnt_pos == (N-1))
                        cnt_pos = 0;
                else
                        cnt_pos = cnt_pos+ 1'b1;
        end
```

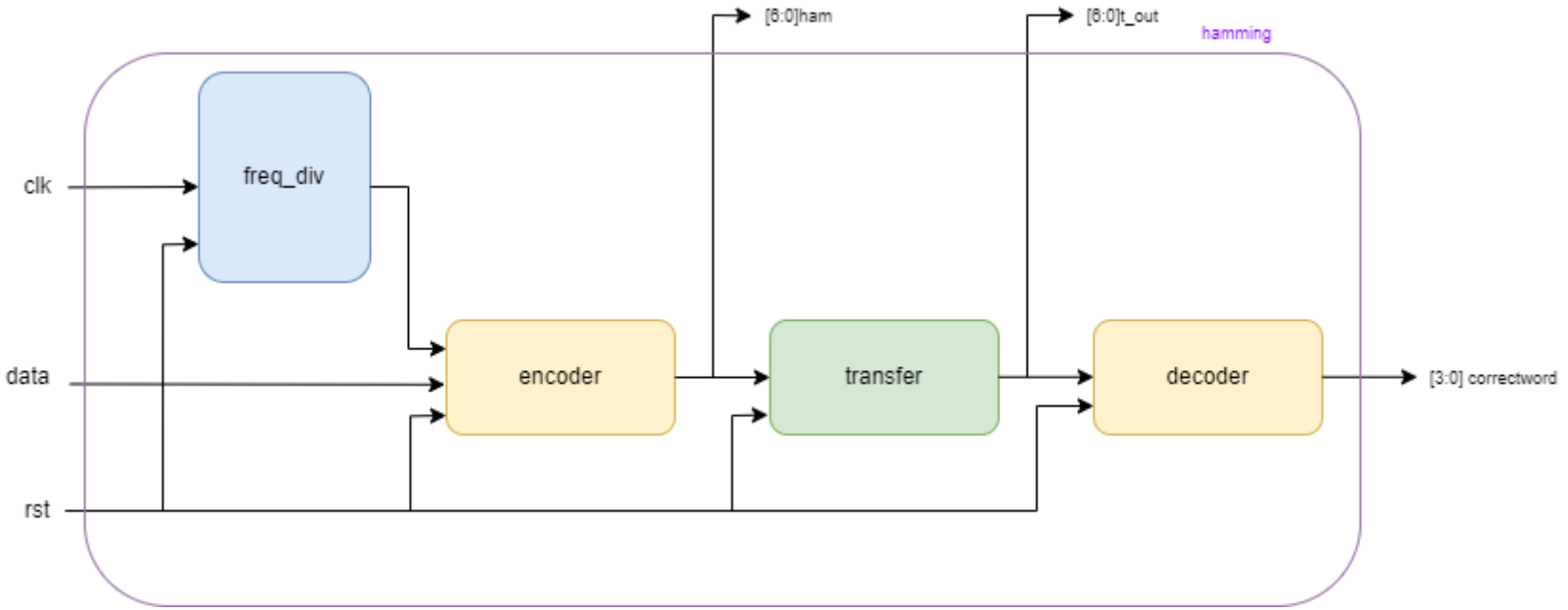
```
always@ (posedge clk or posedge rst)
        begin
                if(rst)
                        clk_pos = 0;
                else if(cnt_pos < N/2)
                        clk_pos = 1;
                else
                        clk_pos = 0;
        end
```

Frequency divider – freq_div (3/3)

```
always@ (negedge clk or posedge rst)
        begin
                if(rst)
                        cnt_neg = 0;
                else if(cnt_neg == (N-1))
                        cnt_neg = 0;
                else
                        cnt_neg = cnt_neg+ 1'b1;
        end
```

```
always@ (negedge clk or posedge rst)
        begin
                if(rst)
                        clk_neg = 0;
                else if(cnt_neg < (N-1)/2)
                        clk_neg = 1;
                else
                        clk_neg = 0;
        end
endmodule
```

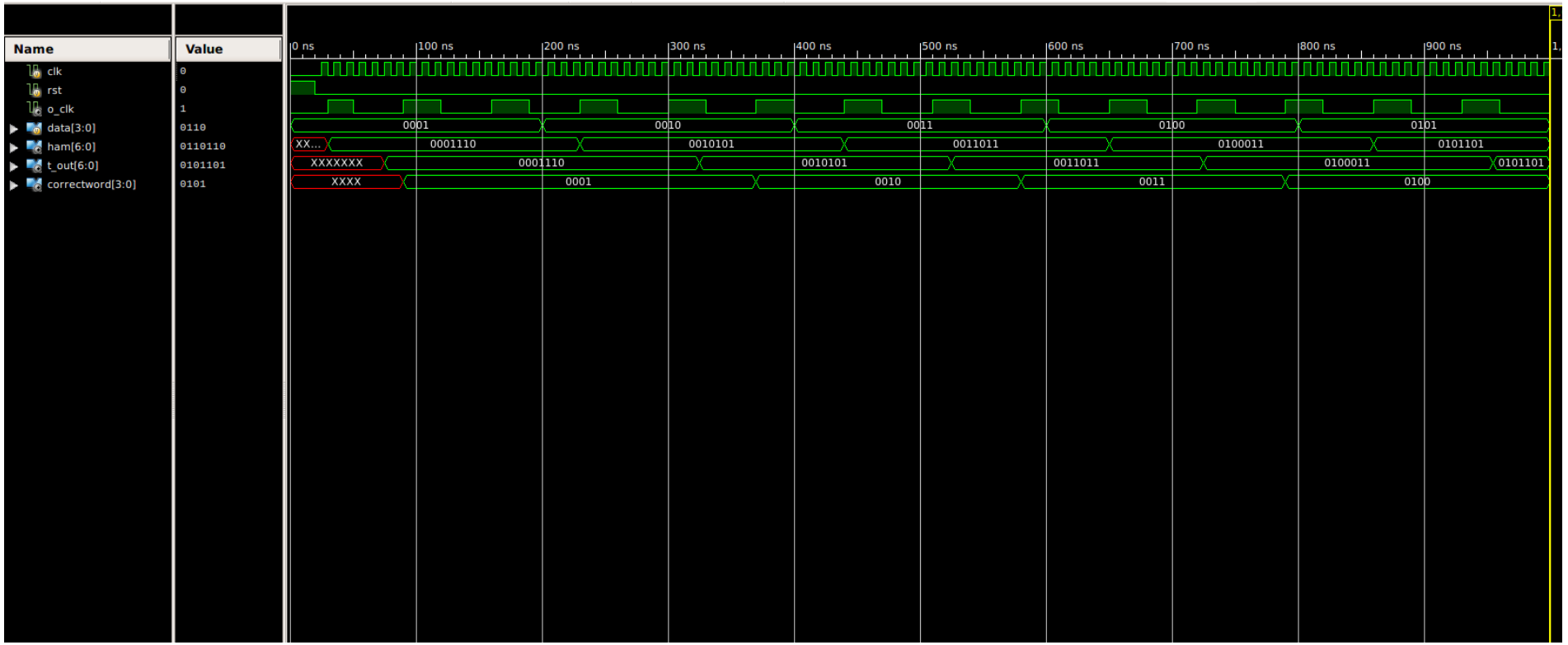
Top module- hamming.v



Top module- hamming.v

```
module hamming(input rst, input clk, input [3:0] data, output [6:0] ham, output [6:0]t_out, output [3:0] correctword, output o_clk );


        freq_div f0(clk, rst, o_clk);

        encoder e0(rst, o_clk, data, ham);

        transfer t0(rst, clk, ham, t_out);

        decoder d0(o_clk, t_out, correctword);


endmodule
```

Wave


Name
Value
clk
rst
o_clk
data[3:0]
ham[6:0]
t_out[6:0]
correctword[3:0]
0
0
1
0110
0110110
0101101
0101
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
0001
0010
0011
0100
0101
XX...
0001110
0010101
0011011
0100011
0101101
XXXXXXX
XXXX

Work Distribution Chart

Work	Responsible Member
Encoder	蔡蔓萱
Transfer (UART_RX.v, UART_TX.v, transfer.v)	李妍瑄
Decoder	邱靖云
Top module, frequency divider & testbench	戴育琪
Presentation & Organization	謝宥宣