# 实验一 通信信号同步电路

## 一、系统方案

1、同步电路原理

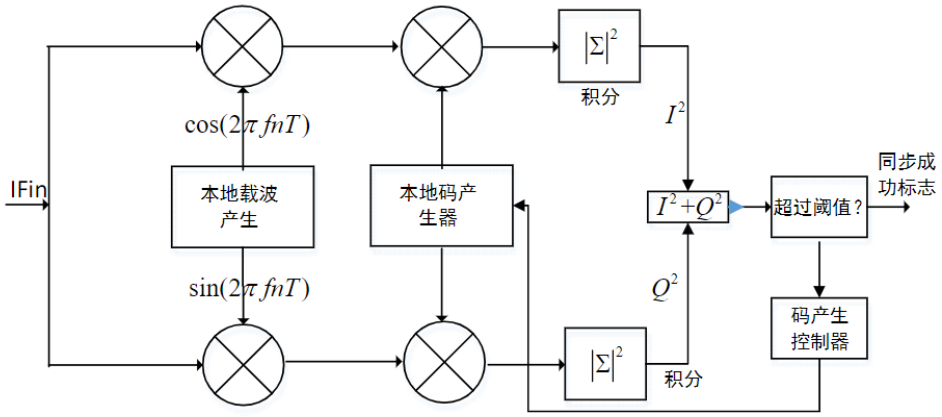



图 1 下变频与码同步电路结构

图 1 为下变频与码同步电路结构图，该电路实现了扩频通信中的中频信号的下变频和码同步。以下是关于该电路结构的详细说明。

**输入信号：**

输入中频信号为 IFin，使用两位有符号数表示，即 1、0、-1，其表达式为：

$$IFin = f(D \oplus M)\cos(2\pi fnT + \phi)$$

其中，符号⊕表示异或，$f(x)$为极化函数，

$$f(x) = \begin{cases} 1, & x = 1 \\ -1, & x = 0 \end{cases}$$

$D$为信号中调制的二进制数据序列，取值为 0、1，数据速率$1kbps$。$M$为用于扩频的伪随机码序列，由循环周期为 31 的 m 码产生，$M$的速率为$31kbps$，每一位所占的时间称为一个码片时间。数据位$D$的每一位都与$M$序列的一个循环周期对齐，即一个$D$对应 31 个 m 码。 $f$ 为载波频率， $f$ =124$kHz$，$n$为采样序列标号，$T$是采样周期，使用载波频率的 4 倍采样。

**信号同步原理：**

在信号接收端，中频信号$IFin$经采样后输入该同步电路，在下变频操作中，使用本地载波产生电路产生的两路本地载波信号$s1(n)$、$s2(n)$与输入信号

$IFin$相乘，得到两路下变频信号$I$和$Q$（分别为同相支路、正交支路），其中本地载波信号为

$$s_1(n) = \cos(2\pi fnT)$$

$$s_2(n) = \sin(2\pi fnT)$$

随后，将两路下变频信号与本地码产生器产生的本地码序列进行相关操作(若本地码为 1，则乘以 1，若本地码为 0，则乘以-1)，并在$T_S = 1ms$的时长内对相关后的两路信号进行积分，两路信号的积分结果分别为$I$ 、$Q$，并求能量信号$S = I^2 + Q^2$。

由于 m 码的互相关和自相关特性，当本地码产生器的相位与发送端码产生器产生的序列相位一致时，积分结果$S$可达到最大值，当两者不一致时，$S$保持一个较小值。因此，可设定一个门限值$S_{th}$与积分结果$S$相比较，若$S < S_{th}$，可认为本地 m 码产生器与发送端 m 码产生器未达到同步状态，此时应将本地 m 码产生器的相位延后一个码片周期，并重新开始积分过程，直至达到$S \geq S_{th}$，即本地 m 码产生器与发送端 m 码产生器同步的状态。

2、总体设计框图及说明

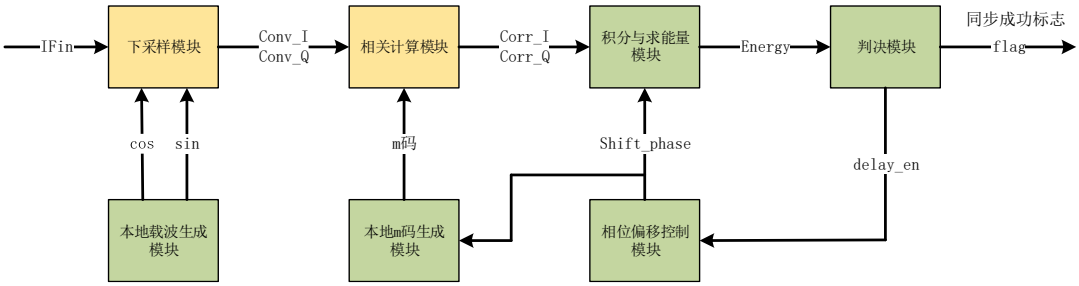



图 2 总体设计框图

按照图 1 的电路结构，进行总体框图设计，总体设计框图如图 2 所示，共有七个模块，具体如下：

- 组合逻辑：下采样模块、相关计算模块
- 时序逻辑：本地载波生成模块、本地 m 码生成模块、积分与求能量模块、相位偏移控制模块、判决模块

- 本地载波生成模块：生成两路本地载波；
- 下采样模块：将输入信号与生成的本地载波进行相乘，得到下变频信号$I$和$Q$；
- 本地 m 码生成模块：生成循环周期 31 的 m 码，具有延迟码片周期的功能；
- 相关计算模块：将下变频信号$I$和$Q$与本地 m 码进行相关运算；

- 积分与求能量模块：在$T_S = 1ms$时间内，即在一个$D$信号的时间内，对相关后的两路信号进行积分；
- 判决模块：将计算得到的能量与阈值进行判决，输出同步标识 flag 与相位延迟信号；
- 相位偏移控制模块：产生一个相位偏移控制信号；

3、理论分析

1) 输入信号采用 2bit 有符号数，结合表达式容易知道，$IFin$信号取值为$-1$、$0$、$1$。
2) 采样频率为载波频率的 4 倍，因此本地载波表达式如下：

$$s_1(n) = \cos(2\pi fnT) = \cos\left(\frac{\pi}{2}n\right)$$

$$s_2(n) = \sin(2\pi fnT) = \sin(\frac{\pi}{2}n)$$

即本地载波仅有 3 个值$-1$、$0$、$1$。因此，本地载波生成模块只生成 4 点余弦或正弦信号即可，因此使用 2bit 有符号数表示。

3) 只有三种取值的 2bit 有符号数$IFin$与只有三种取值的 2bit 有符号数$s_1(n)$相乘，结果也只有三种取值，因此为了减小位宽，结果可以用 2bit 有符号数表示，需要考虑截位。
4) 相关运算中，输入为只有三种取值的 2bit 有符号数，因此相关运算的结果也为三种取值的 2bit 有符号数。
5) D 的数据速率为$1kbps$，M 的速率为$31kbps$，载波频率$f = 124kHz$，$\frac{1}{T} = 4f = 496kHz$。因此，使用最高的频率$496kHz$作为系统时钟。
6) 本地 m 码生成模块中，由于 M 的速率为$31kbps$，为系统时钟的$1/16$。可以采用计数器实现，也可以使用分频器实现。此处我采用计数器实现该模块。
7) 由各信号的频率关系，可知道 1 个 D 对应 31 个 m 码，1 个 m 码对应 16 个载波的值（1 个 m 码对应 4 个载波周期）。因此，在$T_S = 1ms$的时间里，积分就等于对 $31 \times 16 = 496$ 个值进行求和。因此求和结果使用 10bit 有符号数表示，能量使用 20bit 有符号数可以完全表示

## 二、电路与程序设计

本地载波生成模块：

| 端口 | 位宽 | 方向 | 说明 |
| --- | --- | --- | --- |
| clk | 1 bit | input | 时钟 |
| rst_n | 1 bit | input | 复位信号 |
| cos_wave | 2 bit | output | 本地余弦载波 |
| sin_wave | 2 bit | output | 本地正弦载波 |

```
module local_carry_wave (
    input clk,
    input rst_n,

    output reg [1:0]cos_wave,
    output reg [1:0]sin_wave
);
    reg [1:0] count;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            count <= 2'b0;
        end
        else begin
            count <= count+1'b1;
        end
    end

    //使用查表法生成
    always @(posedge clk) begin
        if (!rst_n) begin
            cos_wave <= 2'b01;
            sin_wave <= 2'b00;
        end
        else begin
            case (count)
                2'b00: {cos_wave, sin_wave} <= {2'b01, 2'b00};
                2'b01: {cos_wave, sin_wave} <= {2'b00, 2'b01};
                2'b10: {cos_wave, sin_wave} <= {2'b11, 2'b00};
                2'b11: {cos_wave, sin_wave} <= {2'b00, 2'b11};
            endcase
        end
    end

endmodule
```

下采样模块：

| 端口 | 位宽 | 方向 | 说明 |
| --- | --- | --- | --- |
| IFin | 2 bit | input | 输入的 IFin 信号 |
| cos_in | 2 bit | input | 本地余弦载波 |
| sin_in | 2 bit | input | 本地正弦载波 |
| I_out | 2 bit | output | 下变频信号同向分量 |
| Q_out | 2 bit | output | 下变频信号正交分量 |

```
module down_converter (
    input wire signed [1:0] IFin,
    input wire signed [1:0] cos_in,
    input wire signed [1:0] sin_in,
    output wire signed [1:0] I_out,
    output wire signed [1:0] Q_out
);
    wire signed [2:0] temp_I;
    wire signed [2:0] temp_Q;

    assign temp_I = IFin * cos_in;
    assign temp_Q = IFin * sin_in;

    assign I_out = temp_I[1:0];
    assign Q_out = temp_Q[1:0];
endmodule
```

本地 m 码生成模块：

| 端口 | 位宽 | 方向 | 说明 |
| --- | --- | --- | --- |
| clk | 1 bit | input | 时钟 |
| rst_n | 1 bit | input | 复位信号 |
| shift_parse | 1 bit | input | 延迟码片信号 |
| m_code | 1 bit | output | 本地 m 码 |

```
module m_code_31_generate (
    input clk,
    input rst_n,
    input shift_parse,  //延迟一个码片,
    output reg m_code
);
    //计数，一个 M 码，16 个 clk
    //延迟时，计数器清零
    reg [3:0] count;
```

```
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            count<=4'b0;
        end
        else if (shift_parse) begin
            count<=4'b0;
        end
        else begin
            count<=count +1;
        end
    end

    reg [4:0] Q;//5 级寄存器

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            Q <= 5'b11111;
        end
        else if (count == 4'b1111) begin
            Q <= {Q[3]^Q[0],Q[4:1]};
            m_code = Q[0];
        end
    end

endmodule
```

相关计算模块：

| 端口 | 位宽 | 方向 | 说明 |
|---|---|---|---|
| I_in | 2 bit | input | 下变频信号同向分量 |
| Q_in | 2 bit | input | 下变频信号正交分量 |
| local_code | 1 bit | input | 本地 m 码 |
| I_out | 2 bit | output | 同向分量相关后结果 |
| Q_out | 2 bit | output | 正交分量相关后结果 |

```
module correlator (
    input wire signed [1:0] I_in,   // 2 位有符号 I 路输入信号
    input wire signed [1:0] Q_in,   // 2 位有符号 Q 路输入信号
    input wire local_code,          // 1 位本地码
```

```
    output reg signed [1:0] I_out, // 2 位有符号 I 路输出信号
    output reg signed [1:0] Q_out  // 2 位有符号 Q 路输出信号
);
    always @(*) begin
        if (local_code) begin
            I_out = I_in;
        end else begin
            I_out = -I_in;
        end
    end

    always @(*) begin
        if (local_code) begin
            Q_out = Q_in;
        end else begin
            Q_out = -Q_in;
        end
    end

endmodule
```

积分与求能量模块：

| 端口 | 位宽 | 方向 | 说明 |
|---|---|---|---|
| clk | 1 bit | input | 时钟 |
| rst_n | 1 bit | input | 复位信号 |
| shift_parse | 1 bit | input | 延迟码片信号 |
| I_in | 2 bit | input | 同向分量相关后结果 |
| Q_in | 2 bit | input | 正交分量相关后结果 |
| I_out | 10 bit | output | 同向分量积分值 |
| Q_out | 10 bit | output | 同向分量积分值 |
| energy | 20 bit | output | 能量 |
| result_ok | 1 bit | output | 能量值结果有效 |

```
module integrator (
    input wire clk,
    input wire rst_n,
    input wire shift_parse,
    input wire signed [1:0] I_in, // 2 位有符号 I 路输入信号
    input wire signed [1:0] Q_in, // 2 位有符号 Q 路输入信号
    output reg signed [9:0] I_out, // 6 位有符号 I 路积分输出信号
```

```
    output reg signed [9:0] Q_out, // 6 位有符号 Q 路积分输出信号
    output  [19:0] energy,
    output result_ok
);

    reg [8:0] counter; // 9 位计数器，用于计数 496 个值

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            I_out <= 10'b0;
            Q_out <= 10'b0;
            counter <= 9'b0;
        end
        else if (shift_parse) begin
            I_out <= 10'b0;
            Q_out <= 10'b0;
            counter <= 9'b0;
        end
        else if (counter == 9'd496) begin
            I_out <= 10'b0;
            Q_out <= 10'b0;
            counter <= 9'b0;
        end
        else begin
            I_out <= I_out+I_in;
            Q_out <= Q_out+Q_in;
            counter <= counter+1;
        end
    end

assign energy = I_out*I_out+Q_out*Q_out;
assign result_ok = (counter == 9'd496)?1'b1:1'b0;

endmodule
```

相位偏移控制模块：

| 端口 | 位宽 | 方向 | 说明 |
| --- | --- | --- | --- |
| clk | 1 bit | input | 时钟 |
| rst_n | 1 bit | input | 复位信号 |
| delay_en | 1 bit | input | 延迟码片使能信号 |
| shift_parse | 1 bit | output | 延迟码片信号 |

```
module control (
    input clk,
    input rst_n,
    input delay_en,

    output reg shift_parse
);

    reg [4:0] counter; // 4 位计数器，用于计数 16 个时钟周期

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // 异步复位，当 rst_n 为低电平时，计数器清零，shift_parse 置为低电平
            counter <= 5'b0;
            shift_parse <= 1'b0;
        end else if (delay_en && counter == 5'b0) begin
            // 当检测到 delay_en 脉冲且计数器为 0 时，开始计数
            counter <= counter + 1;
            shift_parse <= 1'b1;
        end else if (counter > 5'b0 && counter < 5'b10000) begin
            // 计数器大于 0 且小于 16 时，继续计数，shift_parse 保持高电平
            counter <= counter + 1;
            shift_parse <= 1'b1;
        end else if (counter == 5'b10000) begin
            // 计数器达到 16 时，计数结束，计数器清零，shift_parse 置为低电平
            counter <= 5'b0;
            shift_parse <= 1'b0;
        end
    end

endmodule
```

判决模块：

| 端口 | 位宽 | 方向 | 说明 |
|---|---|---|---|
| clk | 1 bit | input | 时钟 |
| rst_n | 1 bit | input | 复位信号 |
| result_ok | 1 bit | input | 能量值结果有效 |
| energy | 20 bit | input | 能量 |
| flag | 1 bit | output | 同步成功标志 |
| delay_en | 1 bit | output | 延迟码片使能信 |

| | | | |
|---|---|---|---|
| | | | 号 |

```
/*
    flag 变成 1 后，delay_en 就永远失效
*/
module detect (
    input clk,
    input rst_n,
    input result_ok,
    input [19:0] energy,

    output reg flag,
    output reg delay_en
);

    reg flag_set;        //用于实现：当 flag 变成 1 后，delay_en 就永远失效。完成同步后，就不需要进行码片延迟了

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            flag <= 1'b0;
            delay_en<=1'b0;
            flag_set <= 1'b0;
        end
        else if (result_ok) begin
            if (energy >= 19'd10000) begin
                flag <=1'b1;
                delay_en<=1'b0;
                flag_set <= 1'b1;
            end
            else if (!flag_set) begin
                delay_en <= 1'b1;
            end
            else begin
                delay_en<=1'b0;
            end
        end
        else begin
            //flag <= 1'b0;
            delay_en<=1'b0;
        end
    end
```

```
endmodule
```

译码模块：

| 端口 | 位宽 | 方向 | 说明 |
| --- | --- | --- | --- |
| flag | 1 bit | input | 同步成功标志 |
| sum_I | 10 bit | input | 同向分量积分值 |
| sum_Q | 10 bit | input | 正交分量积分值 |
| result_ok | 1 bit | input | 能量值结果有效 |
| decode_D | 1 bit | output | 译码结果 |

```
module decode (
    input flag,

    input signed [9:0] sum_I,
    input signed [9:0] sum_Q,
    input result_ok,

    output reg decode_D
);
    always @(posedge result_ok) begin
        if (flag) begin
            if ((sum_I + sum_Q) > 0) begin
                decode_D <= 1'b1;
            end
            else begin
                decode_D <= 1'b0;
            end
        end
        else begin
            decode_D <= 1'bx;
        end
    end

endmodule
```

## 三、测试结果

顶层：

```
module top (
```

```
    input clk,
    input rst_n,

    input signed   [1:0]IFin,

    output flag,
    output decode_D
);
    wire [1:0]cos_wave;
    wire [1:0]sin_wave;
    local_carry_wave u_1_local_carry_wave(
        .clk(clk),
        .rst_n(rst_n),
        .cos_wave(cos_wave),
        .sin_wave(sin_wave)
    );

    wire [1:0] conv_I;
    wire [1:0] conv_Q;
    down_converter u_1_down_converter(
        .IFin(IFin),
        .cos_in(cos_wave),
        .sin_in(sin_wave),
        .I_out(conv_I),
        .Q_out(conv_Q)
    );
    wire m_code;
    m_code_31_generate u_m_code_31_generate (
       .clk(clk),
       .rst_n(rst_n),
       .shift_parse(shift_parse),
       .m_code(m_code)
    );

    wire [1:0] corr_I;
    wire [1:0] corr_Q;
    correlator u_correlator (
       .I_in(conv_I),
       .Q_in(conv_Q),
       .local_code(m_code),
       .I_out(corr_I),
       .Q_out(corr_Q)
    );
```

```
    wire [9:0] sum_I;
    wire [9:0] sum_Q;
    wire [19:0] energy;
    wire result_ok;
    integrator u_integrator (
       .clk(clk),
       .rst_n(rst_n),
       .shift_parse(shift_parse),
       .I_in(corr_I),
       .Q_in(corr_Q),
       .I_out(sum_I),
       .Q_out(sum_Q),
       .energy(energy),
       .result_ok(result_ok)
    );

    detect u_detect(
         .clk(clk),
         .rst_n(rst_n),
         .result_ok(result_ok),
         .energy(energy),
         .flag(flag),
         .delay_en(delay_en)
    );

    //wire shift_parse;
    control u_control(
         .clk(clk),
         .rst_n(rst_n),
         .delay_en(delay_en),
         .shift_parse(shift_parse)
    );

    //译码模块
    wire decode_D;
    decode u_decode(
         .flag(flag),
         .sum_I(sum_I),
         .sum_Q(sum_Q),
         .result_ok(result_ok),
         .decode_D(decode_D)
    );
endmodule
```

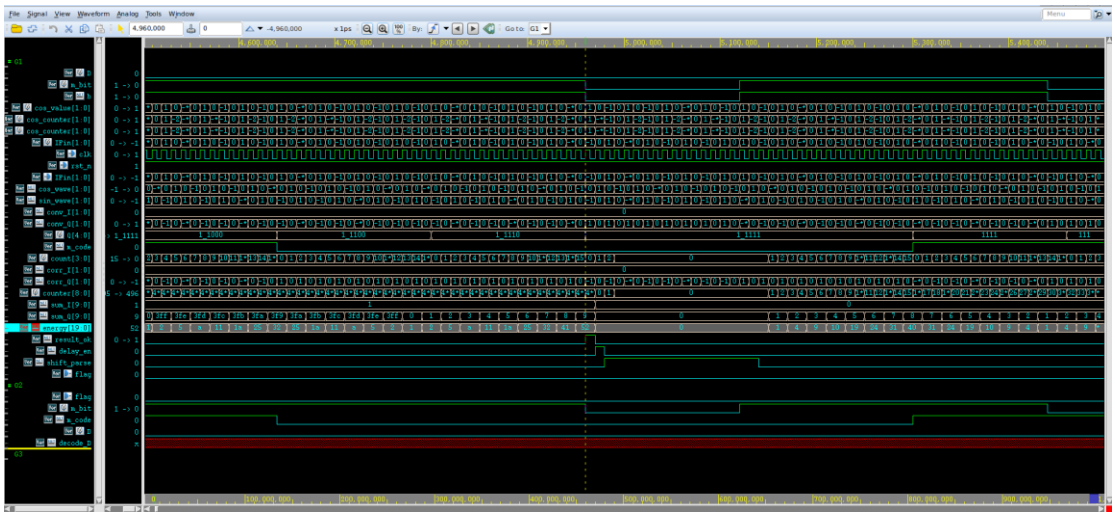
test_bench：

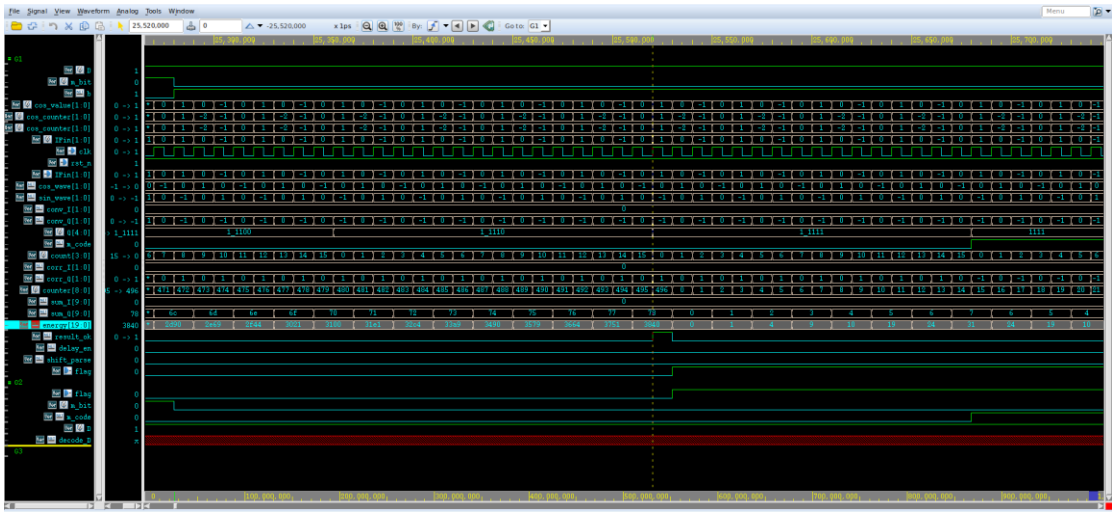
```
`timescale 1ns / 1ps

module tb_top;

    // 参数定义
    parameter DURATION_PER_D = 496;  // 每个 D 持续的时钟周期数
    parameter M_CHIP_DURATION = 16;  // 每个 M 码片持续的时钟周期数
    parameter NUM_D_SEQUENCES = 100;    // 定义 D 序列的数量

    // 信号声明
    reg clk;
    reg rst_n;
    wire flag;
    wire decode_D;
    reg signed [1:0] IFin;

    // 实例化被测试模块
    top uut (
        .clk(clk),
        .rst_n(rst_n),
        .IFin(IFin),
        .flag(flag),
        .decode_D(decode_D)
    );

    // 时钟生成（10ns 周期）
    initial begin
         clk = 1;
         forever #5 clk = ~clk;
    end

    // 复位信号生成
    initial begin
         rst_n = 0;
         #10;
         rst_n = 1;
    end

    // 生成余弦载波信号（1,0,-1,0 循环）
    reg [1:0] cos_value;
    reg [1:0] cos_counter;
    initial begin
         cos_counter = 2'b0;
```

```
end
always @(posedge clk) begin
    cos_counter <= cos_counter + 1;
    case (cos_counter)
        2'b00: cos_value <= 2'b01; // 1
        2'b01: cos_value <= 2'b00; // 0
        2'b10: cos_value <= 2'b11; // -1
        2'b11: cos_value <= 2'b00; // 0
    endcase
end

// 生成 M 序列（31 位 m 码）
reg [4:0] m_lfsr;
reg m_bit;
reg [3:0] m_chip_counter; // 每个码片持续 16 个时钟周期
initial begin
    m_lfsr = 5'b00110;
    m_chip_counter = 4'd0;
    m_bit = 1'b0;
    forever begin
        #(M_CHIP_DURATION * 10); // 每个码片持续 16 个时钟周期
        m_lfsr = {m_lfsr[3] ^ m_lfsr[0], m_lfsr[4:1]};
        m_bit = m_lfsr[0];
    end
end

// 生成 D 序列（0/1 交替，每个持续 496 个时钟周期）
reg D;
reg [9:0] d_counter; // 496 个时钟周期计数
integer d_sequence_count; // 记录 D 序列的数量
initial begin
    D = 1'b0;
    d_counter = 10'd0;
    d_sequence_count = 0;
    forever begin
        #(DURATION_PER_D * 10); // 等待 496 个时钟周期
        D = $random % 2;
        d_sequence_count = d_sequence_count + 1;
        if (d_sequence_count == NUM_D_SEQUENCES * 2) begin
            $finish; // 当生成指定数量的 D 序列后结束仿真
        end
    end
end
```

```
    // 计算 IFin 信号（D^M 后应用极化函数和载波相乘）
    wire b = D ^ m_bit;
    always @(*) begin
        case (cos_value)
            2'b01: IFin = (b) ? 2'b01 : 2'b11; // cos=1 时，b=1→1，b=0→-1
            2'b11: IFin = (b) ? 2'b11 : 2'b01; // cos=-1 时，b=1→-1，b=0→1
            default: IFin = 2'b00; // cos=0 时输出 0
        endcase
    end

    // 产生 fsdb 文件
    initial begin
        $fsdbDumpfile("tb.fsdb");
        $fsdbDumpvars("+all");
    end

endmodule
```

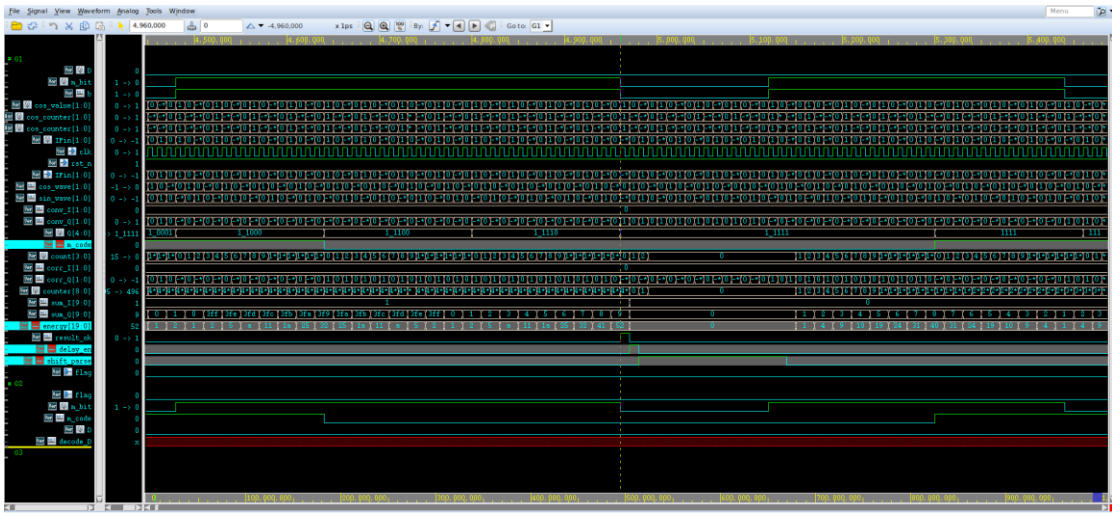
1、能量计算与阈值判别





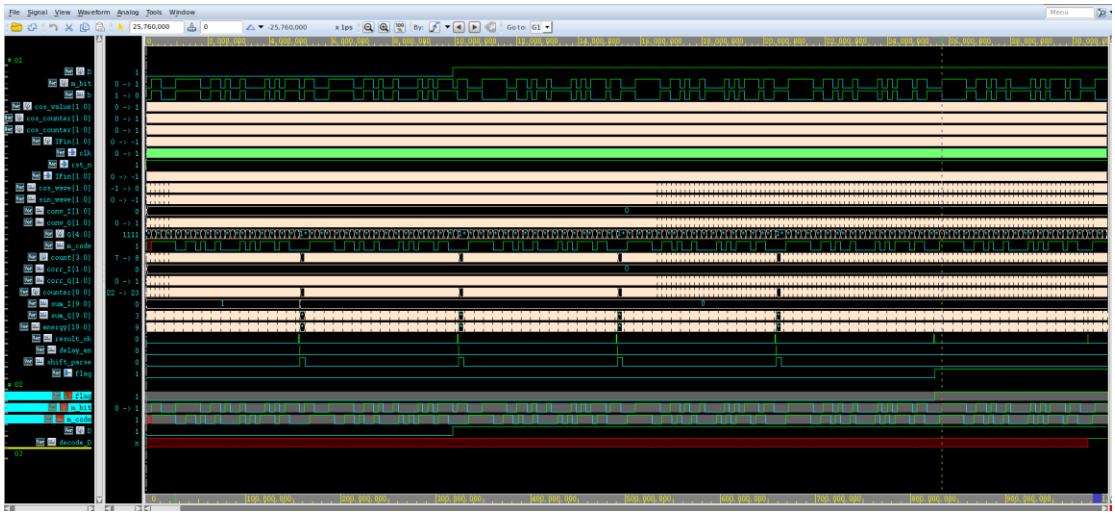
可以看到，没有码同步的状态下，计算出的能量值非常小；码同步后，计算出的能量值非常大。这符合理论推导，满足了 m 序列的自相关和互相关特性。

2、码片延迟



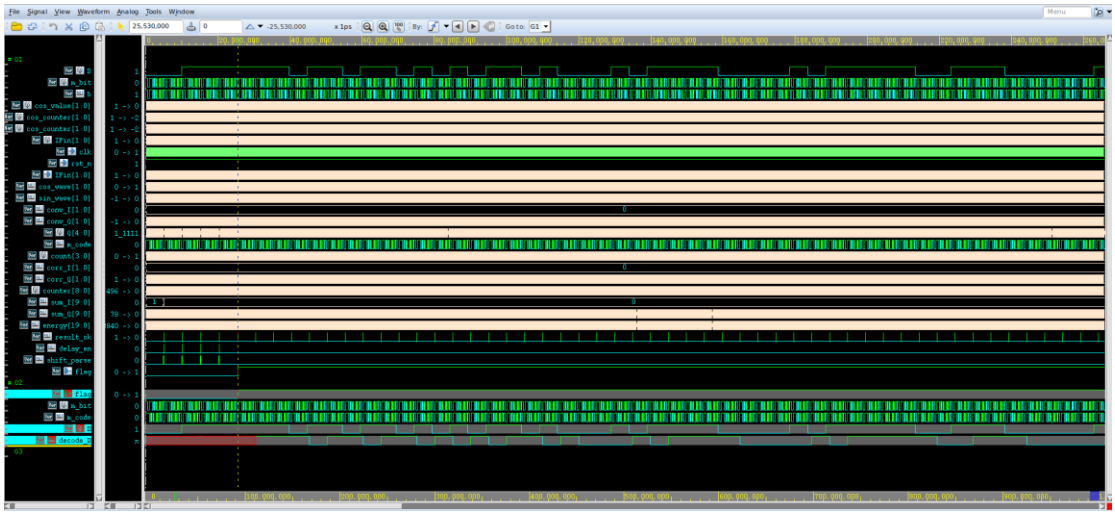
可以看到，当能量不满足阈值时，会产生一个码片延迟信号 shift_parse，使得 m 序列延迟一个码片，并且积分值和能量值都重新开始计算。

3、码同步的结果：



m_bit 为 testbench 中生成 IFin 时使用的 m 序列，m_code 为本地 m 序列生成器产生的 m 序列。可以看到，两个信号没有对齐。经过同步电路后，由于电路中的码片延迟模块，使得两个信号逐渐对齐，最终趋于同步，完成了码同步。完成码同步后，flag 信号被拉高，开始译码。

4、译码



当码同步成功后，flag 信号被拉高，开始进行译码。D 信号为 test_bench 中生成的调制电文，decode_D 为译码出的电文，可以看到，信号被正确译码，且 decode_D 延迟于 D 一个码片周期。

# 附加部分：

要求：阅读北斗 ICD 及北斗信号捕获参考文献，在任务一、任务二的基础上。借助 AI 工具，设计北斗 B1I 信号的捕获电路及仿真代码，完成信号同步。

## 北斗信号的信号规格

$B1I$信号由载波频率、测距码和导航电文组成。测距码与导航电文调制在载波上，北斗信号$B1I$的表达式如下：

$$S_{B1I}^{j}(t) = A_{B1I}C_{B1I}^{j}(t)D_{B1I}^{j}(t)\cos(2\pi f_1 t + \varphi_{\mathrm{B1I}}^{j})$$

其中，各参数意义如下：

$j$：卫星编号

$A_{B1I}$：$S_{B1I}$信号幅度

$C_{B1I}$：$S_{B1I}$信号测距码

$D_{B1I}$：调制在$B1I$测距码上的数据

$f_1$：载波频率

$\varphi_{B1I}$：载波初始相位

其余参数参考文档，此处不再列出

## 1、系统设计

北斗信号$B1I$的信号捕获电路与之前系统大致类似。首先，接收信号需要与本地载波进行相乘，得到同向分量$I$与正交分量$Q$，之后与本地产生的 PRN 码进行相关预算，之后进行积分运算，再进行峰值判决，输出信号同步成功的标志。若不同步，需要控制本地 PRN 码的生成，进行码片偏移。

## 2、各模块设计

（1）输入信号分析

$$S_{B1I}^{j}(t) = A_{B1I}C_{B1I}^{j}(t)D_{B1I}^{j}(t)\cos(2\pi f_1 t + \varphi_{\mathrm{B1I}}^{j})$$

我们假设信号经过天线接收后，进行了信号放大，所以我们不关心$A_{B1I}$的值为多少。$C_{B1I}$信号为伪随机码，其取值为 0 与 1，由于$B1I$信号为 BPSK 信号，因此，在发送端会将$C_{B1I}$的值进行映射，即 1 映射为 1，0 映射为-1。D 信

号同理。因此，系统输入信号可以简单的使用 2bit 有符号数来表示。采样频率使用 4 倍的载波频率。

（2）信号速率分析

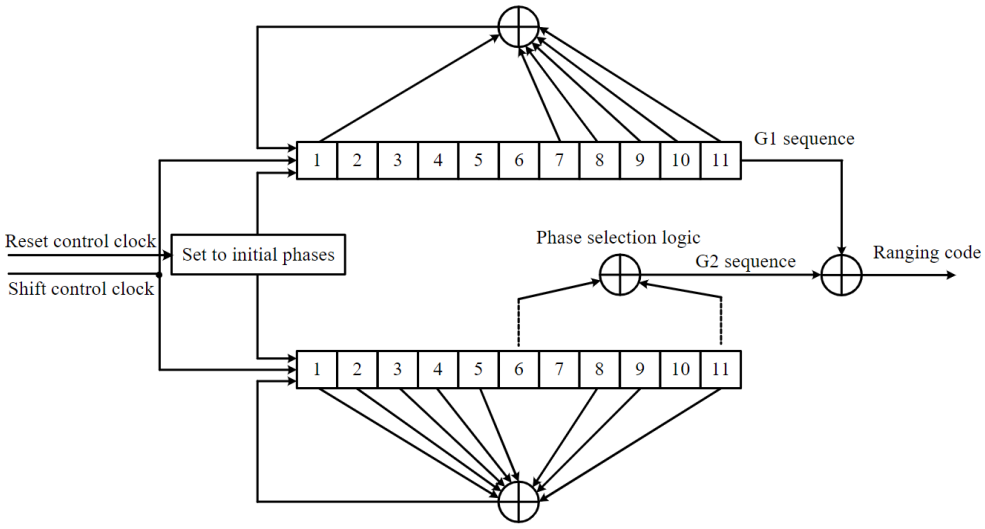
载波频率$f_1 = 1561.098\ MHz$

测距码$C_{B1I}$的码片速率为$2.046Mcps$，码长为 2046 个码片，即 2046 个码片为一个周期。1 秒钟有 1000 个周期。

数据$D_{B1I}$的速率为：电文分 D1 和 D2 两种结构，D1 的速率为 50bps，D2 的速率为 500bps。D1 由 MEO/IGSO 卫星发送，D2 由 GEO 卫星发送。我们选择接收 D2 电文，原因说明见下部分——$C_{B1I}$的生成电路。

（3）$C_{B1I}$的生成电路

$C_{B1I}$的生成电路如下图所示，由对平衡 Gold 码截去最后一个码片生成。




$C_{B1I}$的生成电路

如下图所示（来自文档中的部分截图），文档中提到，卫星的测距码$C_{B1I}$的生成电路中，G2 序列的相移方式共有 63 种，因此，若要实现能够捕获全部卫星的捕获电路，需要生成 63 路 G2 的序列，系统会非常庞大。因此为了简化，我们只捕获一路信号（即一颗卫星的信号），所以我选择下图中第一种方式，即选择 G2 移位寄存器中的 1 和 3 抽头。由图可知，该 G2 序列的相位分配给了 GEO 卫星，由于 GEO 卫星发送的电文为 D2，因此，我们这个系统中的电文速率 D 为 500bps

Table 4-1 Phase assignment of G2 sequence

| SV ID | Satellite type | Ranging code number* | Phase assignment of G2 sequence |
|---|---|---|---|
| 1 | GEO satellite | 1 | 1⊕3 |
| 2 | GEO satellite | 2 | 1⊕4 |
| 3 | GEO satellite | 3 | 1⊕5 |
| 4 | GEO satellite | 4 | 1⊕6 |
| 5 | GEO satellite | 5 | 1⊕8 |
| 6 | MEO/IGSO satellite | 6 | 1⊕9 |
| 7 | MEO/IGSO satellite | 7 | 1⊕10 |
| 8 | MEO/IGSO satellite | 8 | 1⊕11 |
| 9 | MEO/IGSO satellite | 9 | 2⊕7 |
| 10 | MEO/IGSO satellite | 10 | 3⊕4 |
| 11 | MEO/IGSO satellite | 11 | 3⊕5 |
| 12 | MEO/IGSO satellite | 12 | 3⊕6 |
| 13 | MEO/IGSO satellite | 13 | 3⊕8 |
| 14 | MEO/IGSO satellite | 14 | 3⊕9 |
| 15 | MEO/IGSO satellite | 15 | 3⊕10 |
| 16 | MEO/IGSO satellite | 16 | 3⊕11 |
| 17 | MEO/IGSO satellite | 17 | 4⊕5 |
| 18 | MEO/IGSO satellite | 18 | 4⊕6 |
| 19 | MEO/IGSO satellite | 19 | 4⊕8 |
| 20 | MEO/IGSO satellite | 20 | 4⊕9 |
| 21 | MEO/IGSO satellite | 21 | 4⊕10 |
| 22 | MEO/IGSO satellite | 22 | 4⊕11 |
| 23 | MEO/IGSO satellite | 23 | 5⊕6 |
| 24 | MEO/IGSO satellite | 24 | 5⊕8 |
| 25 | MEO/IGSO satellite | 25 | 5⊕9 |
| 26 | MEO/IGSO satellite | 26 | 5⊕10 |

G2 序列的相位分配表（部分）

（4）各信号的关系

数据$D_{B1I}$的码率为 500bps，测距码$C_{B1I}$的码率为$2.046Mcps$，因此一个$D_{B1I}$对应 4092 个$C_{B1I}$（两个循环周期）。载波频率为$f_1 = 1561.098\,MHz$，使用 4 倍载波频率作为采样频率，因此本地载波为 4 点余弦与正弦信号。一个$C_{B1I}$对应$1561.098/2.046 = 763$个周期，也即一个$C_{B1I}$对应$763 \times 4 = 3052$个载波值。我们以频率$1561.098 \times 4 = 6{,}244.392MHz$作为系统时钟。

（5）利用 AI 对先前模块进行对应修改

- 本地载波生成模块：保持不变
- 下采样模块：组合逻辑，保持不变
- 相关运算模块：组合逻辑，保持不变
- 本地测距码生成模块：

测距码为 GOLD 码，由两组码长为 31 的 m 码异或运算而成。上文已给出其生成电路。下方给出其代码：

```
module ranging_code_generate (
    input clk,
    input rst_n,
    input shift_parse,  // 延迟一个码片
    output reg ranging_code
);

    wire g1,g2;

    m_g1_generate u_m_g1_generate(
       .clk(clk),
       .rst_n(rst_n),
       .shift_parse(shift_parse),
       .m_code(g1)
    );

    m_g2_generate u_m_g2_generate(
       .clk(clk),
       .rst_n(rst_n),
       .shift_parse(shift_parse),
       .m_code(g2)
    );

    always @(posedge clk) begin
        if (!rst_n) begin
            ranging_code <= 1'b0;
        end
        else begin
            ranging_code <= g1 ^ g2;
        end
    end

endmodule
```

```
module m_g1_generate (
    input clk,
    input rst_n,
    input shift_parse,  // 延迟一个码片
    output reg m_code
);

    // 计数，一个 M 码，3052 个 clk
    // 延迟时，计数器清零
    reg [11:0] count;  // 因为要计数到 3052，至少需要 12 位
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            count <= 12'b0;
        end
        else if (shift_parse) begin
            count <= 12'b0;
        end
        else begin
            count <= count + 1;
        end
    end

    reg [10:0] Q; // 11 级寄存器
    wire feedback;

    // 合适的反馈抽头，对于 2046 长度的 M 序列需要根据理论确定
    assign feedback = Q[10] ^ Q[9] ^ Q[8] ^ Q[7] ^ Q[6] ^ Q[0];

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            Q <= 11'b01010101010;
        end
        else if (count == 12'd3051) begin
            Q <= {Q[9:0],feedback};
            m_code <= Q[10];
            count <= 12'b0; //  计数到 3052 后，计数器清零
        end
    end

endmodule
```

```
module m_g2_generate (
    input clk,
```

```
    input rst_n,
    input shift_parse,  // 延迟一个码片
    output reg m_code
);

    // 计数，一个 M 码，3052 个 clk
    // 延迟时，计数器清零
    reg [11:0] count;  // 因为要计数到 3052，至少需要 12 位
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            count <= 12'b0;
        end
        else if (shift_parse) begin
            count <= 12'b0;
        end
        else begin
            count <= count + 1;
        end
    end

    reg [10:0] Q; // 11 级寄存器
    wire feedback;

    // 合适的反馈抽头，对于 2046 长度的 M 序列需要根据理论确定
    assign feedback = Q[10] ^ Q[9] ^ Q[8] ^ Q[7] ^ Q[4] ^ Q[3] ^ Q[2] ^ Q[1] ^
Q[0];

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            Q <= 11'b01010101010;
        end
        else if (count == 12'd3051) begin
            Q <= {Q[9:0],feedback};
            m_code <= Q[0] ^ Q[2];    //抽头 1 和 3
            count <= 12'b0; //  计数到 3052 后，计数器清零
        end
    end

endmodule
```

■ 积分与求能量模块：

由信号分析可知，积分与能量的位宽需要进行修改，计数器的值需要修改，其逻辑不变，实现如下：

```
module integrator_beidou (
    input wire clk,
    input wire rst_n,
    input wire shift_parse,
    input wire signed [1:0] I_in, // 2 位有符号 I 路输入信号
    input wire signed [1:0] Q_in, // 2 位有符号 Q 路输入信号
    output reg signed [24:0] I_out, // 10 位有符号 I 路积分输出信号
    output reg signed [24:0] Q_out, // 10 位有符号 Q 路积分输出信号
    output  [49:0] energy,
    output result_ok
);

    reg [23:0] counter; // 24 位计数器，用于计数 12488784 个值

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            I_out <= 25'b0;
            Q_out <= 25'b0;
            counter <= 24'b0;
        end
        else if (shift_parse) begin
            I_out <= 25'b0;
            Q_out <= 25'b0;
            counter <= 24'b0;
        end
        else if (counter == 24'd12488784) begin
            I_out <= 25'b0;
            Q_out <= 25'b0;
            counter <= 24'b0;
        end
        else begin
            I_out <= I_out + I_in;
            Q_out <= Q_out + Q_in;
            counter <= counter + 1;
        end
    end

assign energy = I_out * I_out + Q_out * Q_out;
assign result_ok = (counter == 24'd12488784)? 1'b1 : 1'b0;

endmodule
```

■ 判决模块：

判决模块需修改对应数据位宽与门限值，具体如下：

```
/*
    flag 变成 1 后，delay_en 就永远失效
*/
module detect_beidou (
    input clk,
    input rst_n,
    input result_ok,
    input [49:0] energy,

    output reg flag,
    output reg delay_en
);

    reg flag_set;        //用于实现：当 flag 变成 1 后，delay_en 就永远失效。完成同步后，就不需要进行码片延迟了

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            flag <= 1'b0;
            delay_en<=1'b0;
            flag_set <= 1'b0;
        end
        else if (result_ok) begin
            if (energy >= 50'd19000000000) begin
                flag <=1'b1;
                delay_en<=1'b0;
                flag_set <= 1'b1;
            end
            else if (!flag_set) begin
                delay_en <= 1'b1;
            end
            else begin
                delay_en<=1'b0;
            end
        end
        else begin
            delay_en<=1'b0;
        end
    end

endmodule
```

■ 相位偏移控制模块：

相位偏移控制模块需要修改相位延迟输出的时间：

```
module control_beidou (
    input clk,
    input rst_n,
    input delay_en,

    output reg shift_parse
);

    reg [11:0] counter; // 12 位计数器，用于计数 3052 个时钟周期

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // 异步复位，当 rst_n 为低电平时，计数器清零，shift_parse 置为低电平
            counter <= 12'b0;
            shift_parse <= 1'b0;
        end else if (delay_en && counter == 12'b0) begin
            // 当检测到 delay_en 脉冲且计数器为 0 时，开始计数
            counter <= counter + 1;
            shift_parse <= 1'b1;
        end else if (counter > 12'b0 && counter < 12'd3052) begin
            // 计数器大于 0 且小于 3052 时，继续计数，shift_parse 保持高电平
            counter <= counter + 1;
            shift_parse <= 1'b1;
        end else if (counter == 12'd3052) begin
            // 计数器达到 3052 时，计数结束，计数器清零，shift_parse 置为低电平
            counter <= 12'b0;
            shift_parse <= 1'b0;
        end
    end

endmodule
```

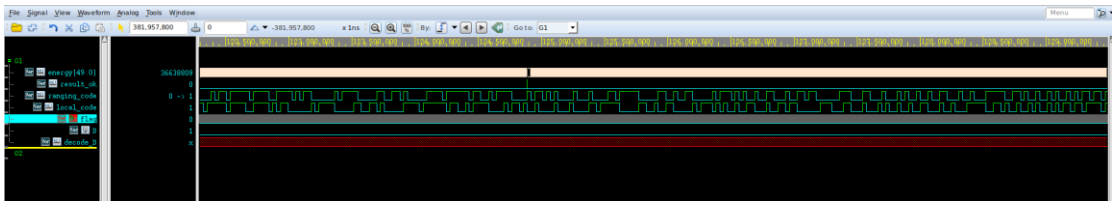
■ 译码模块：

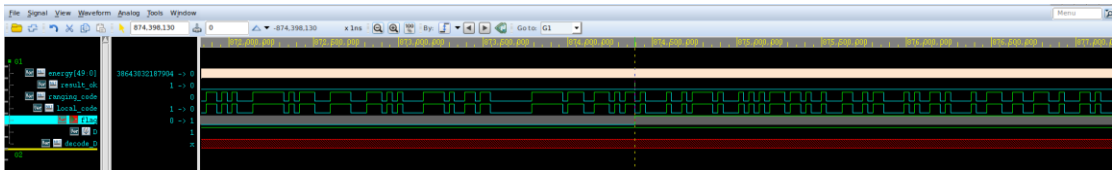
使用能量来作为译码的判断依据

```
module decode_beidou (
    input flag,
```

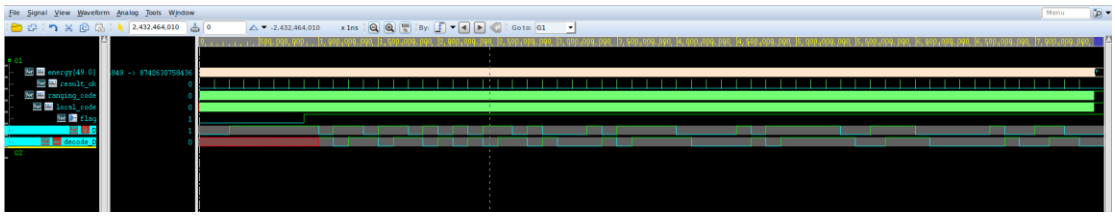
```
    input signed [49:0] energy,
    input result_ok,

    output reg decode_D
);
    always @(posedge result_ok) begin
        if (flag) begin
            if (energy < 50'd15000000000) begin
                decode_D <= 1'b0;
            end
            else begin
                decode_D <= 1'b1;
            end
        end
        else begin
            decode_D <= 1'bx;
        end
    end

endmodule
```

3、结果展示：





ranging_code 为 tb 中生成的测距码，local 为本地生成的测距码，可以看到，两者匹配了，即码同步。



D 为原卫星信息，decode_D 为译码出的结果。可以看到，该电路成功捕获活到了信号，decode_D 比 D 延迟一个码片。