### 实验一 通信信号同步电路

#### 一、系统方案

##### 1、同步电路原理

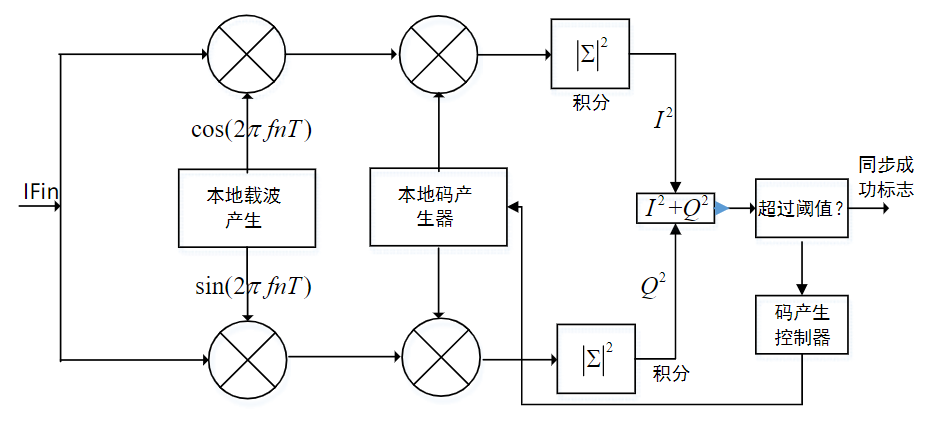


图1 下变频与码同步电路结构

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：

其中，符号表示异或，为极化函数，

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

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 、，并求能量信号。

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，可设定一个门限值与积分结果相比较，若，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，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：在时间内，即在一个信号的时间内，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，信号取值为、、。
2. 采样频率为载波频率的4倍，因此本地载波表达式如下：

即本地载波仅有3个值、、。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。

1. 只有三种取值的2bit有符号数与只有三种取值的2bit有符号数相乘，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。
2. 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。
3. D的数据速率为，M的速率为，载波频率，。因此，使用最高的频率作为系统时钟。
4. 本地m码生成模块中，由于M的速率为，为系统时钟的。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。
5. 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，在的时间里，积分就等于对 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

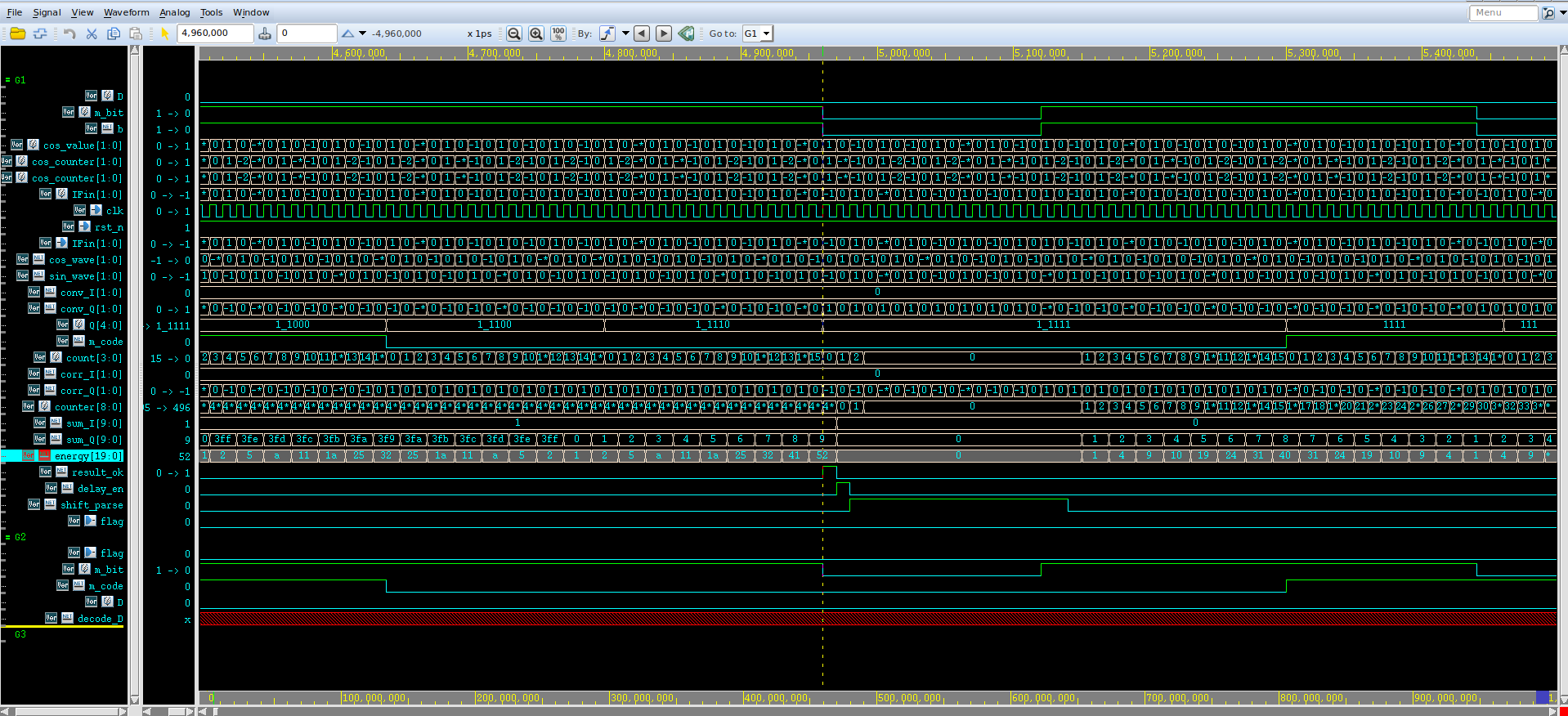
##### 顶层：

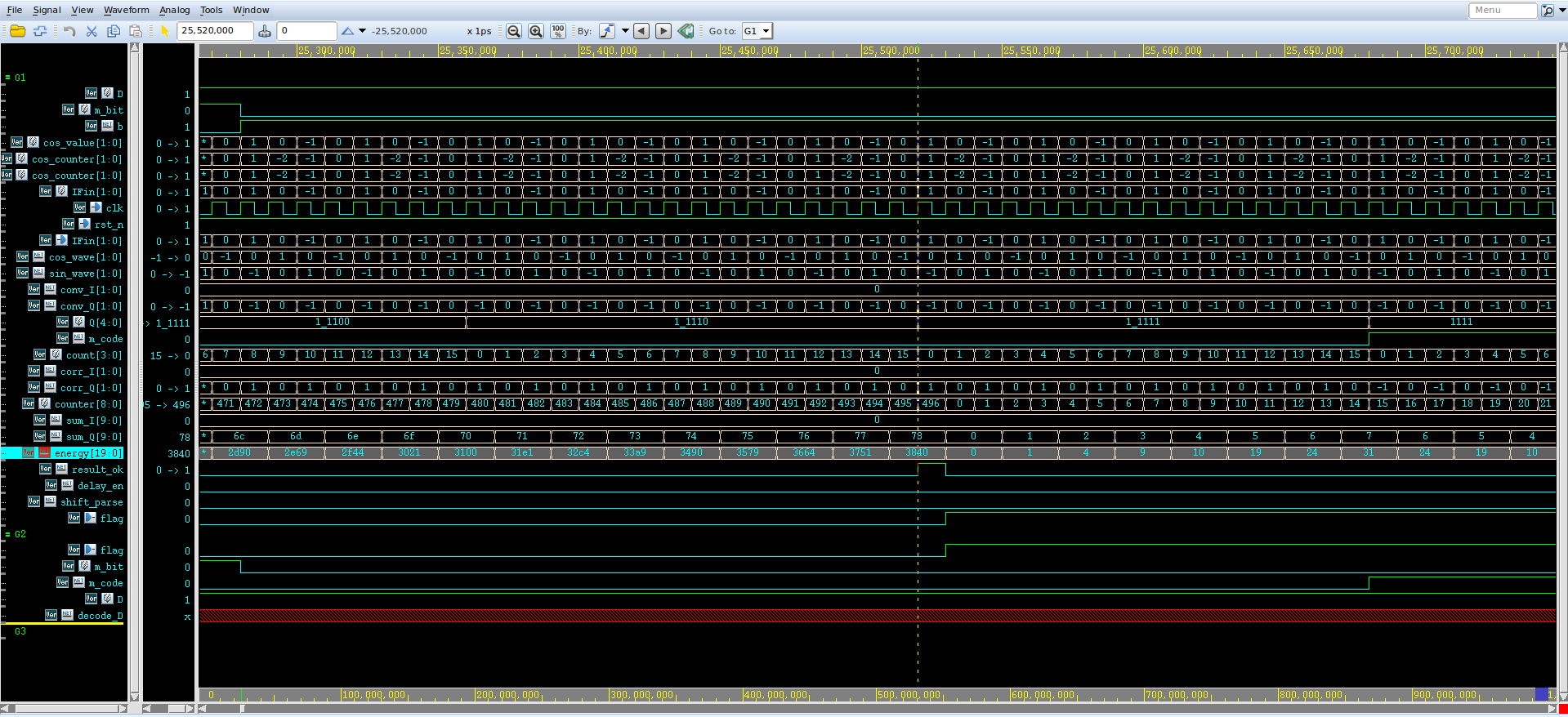
|  |
| --- |
| module top (      input clk,      input rst\_n,      input signed  [1:0]IFin,      output flag,      output decode\_D  );      wire [1:0]cos\_wave;      wire [1:0]sin\_wave;      local\_carry\_wave u\_1\_local\_carry\_wave(          .clk(clk),          .rst\_n(rst\_n),          .cos\_wave(cos\_wave),          .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 |

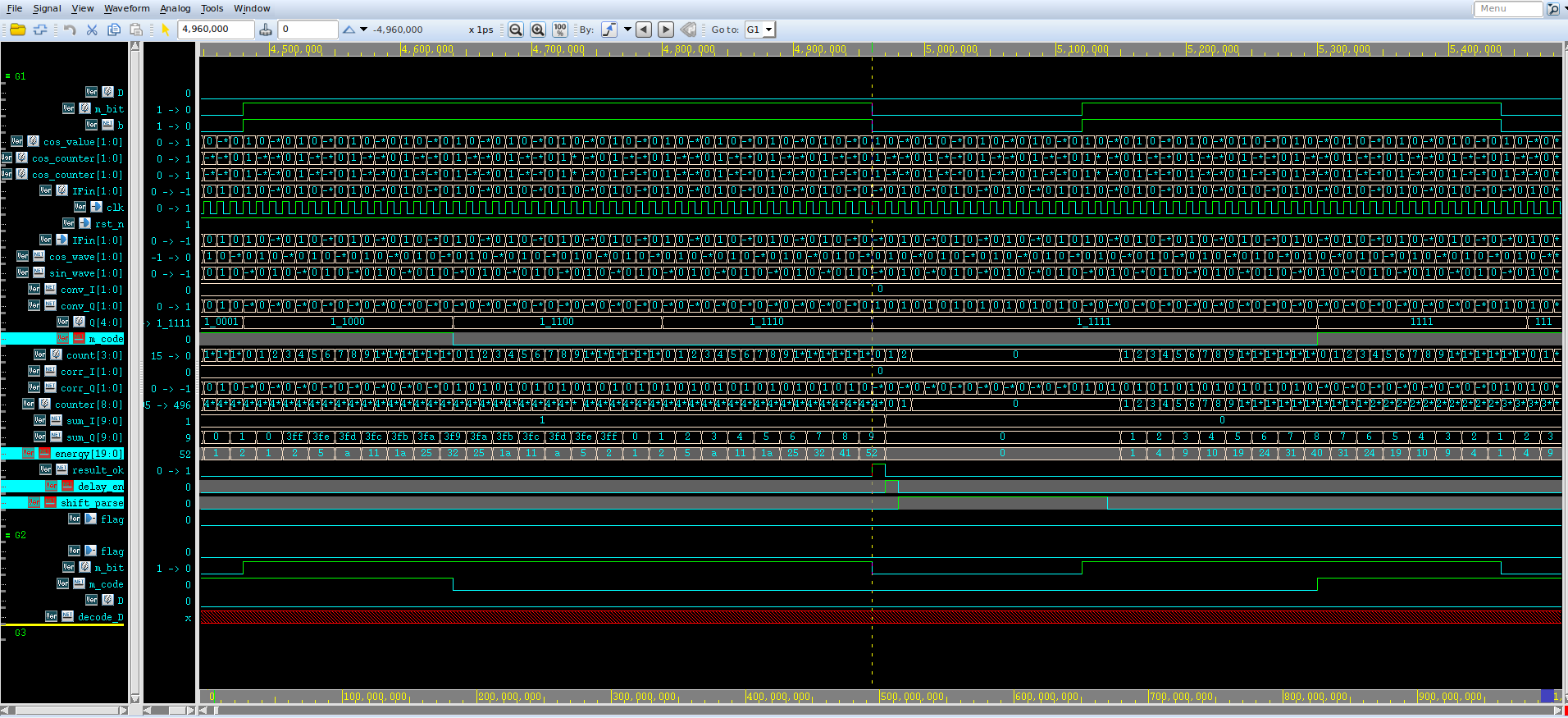
##### 1、能量计算与阈值判别





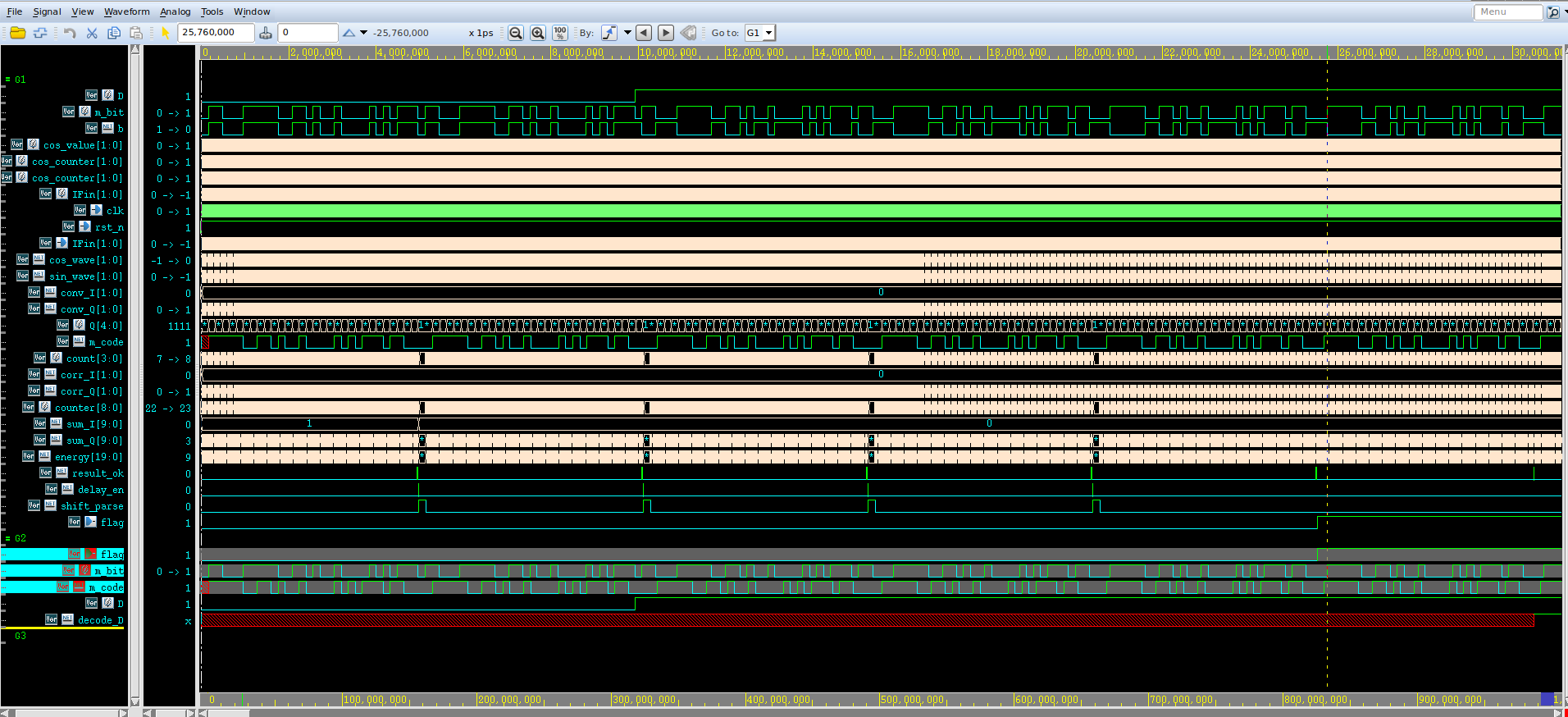
可以看到，没有码同步的状态下，计算出的能量值非常小；码同步后，计算出的能量值非常大。这符合理论推导，满足了m序列的自相关和互相关特性。

##### 2、码片延迟



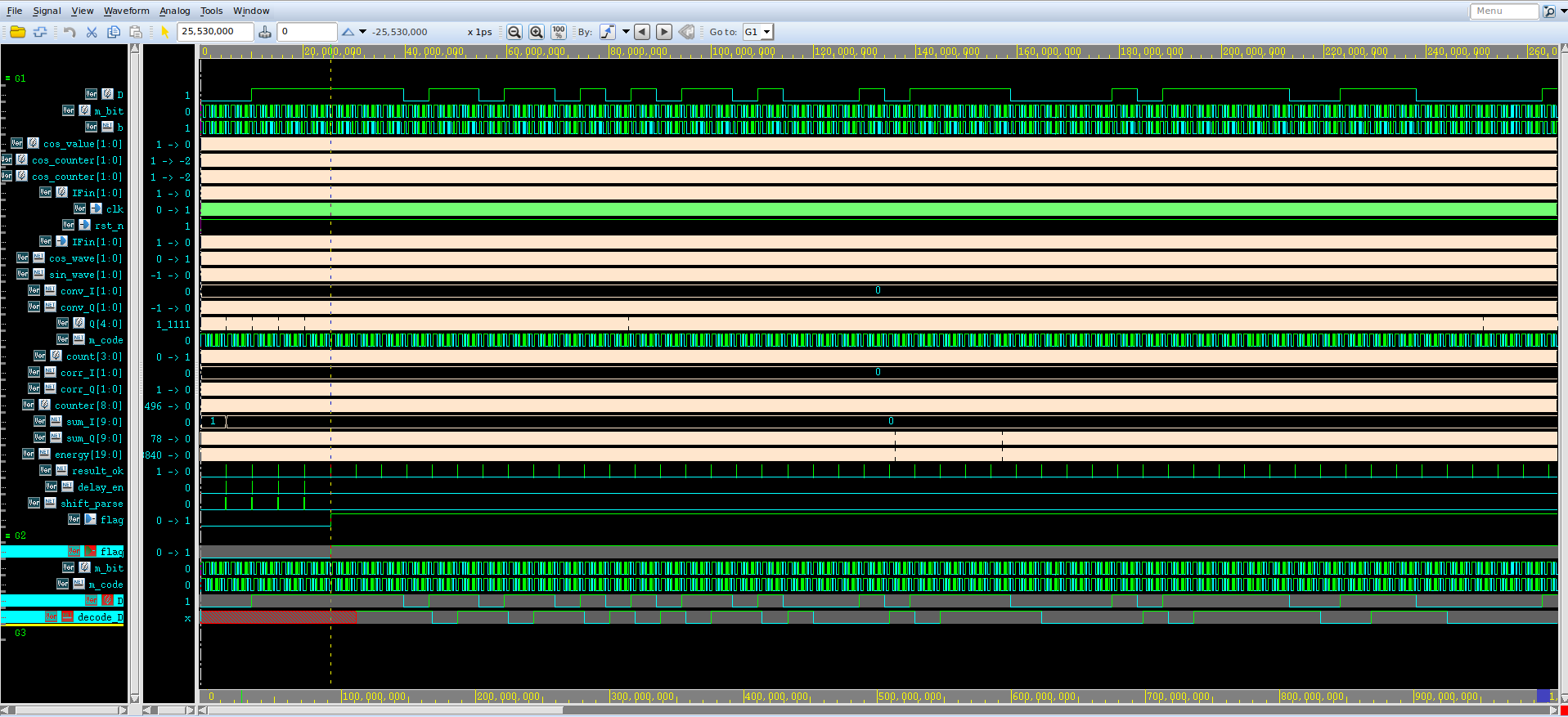
可以看到，当能量不满足阈值时，会产生一个码片延迟信号shift\_parse，使得m序列延迟一个码片，并且积分值和能量值都重新开始计算。

##### 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

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，北斗信号的表达式如下：

其中，各参数意义如下：

：卫星编号

：信号幅度

：信号测距码

：调制在测距码上的数据

：载波频率

：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，得到同向分量与正交分量，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

我们假设信号经过天线接收后，进行了信号放大，所以我们不关心的值为多少。信号为伪随机码，其取值为0与1，由于信号为BPSK信号，因此，在发送端会将的值进行映射，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

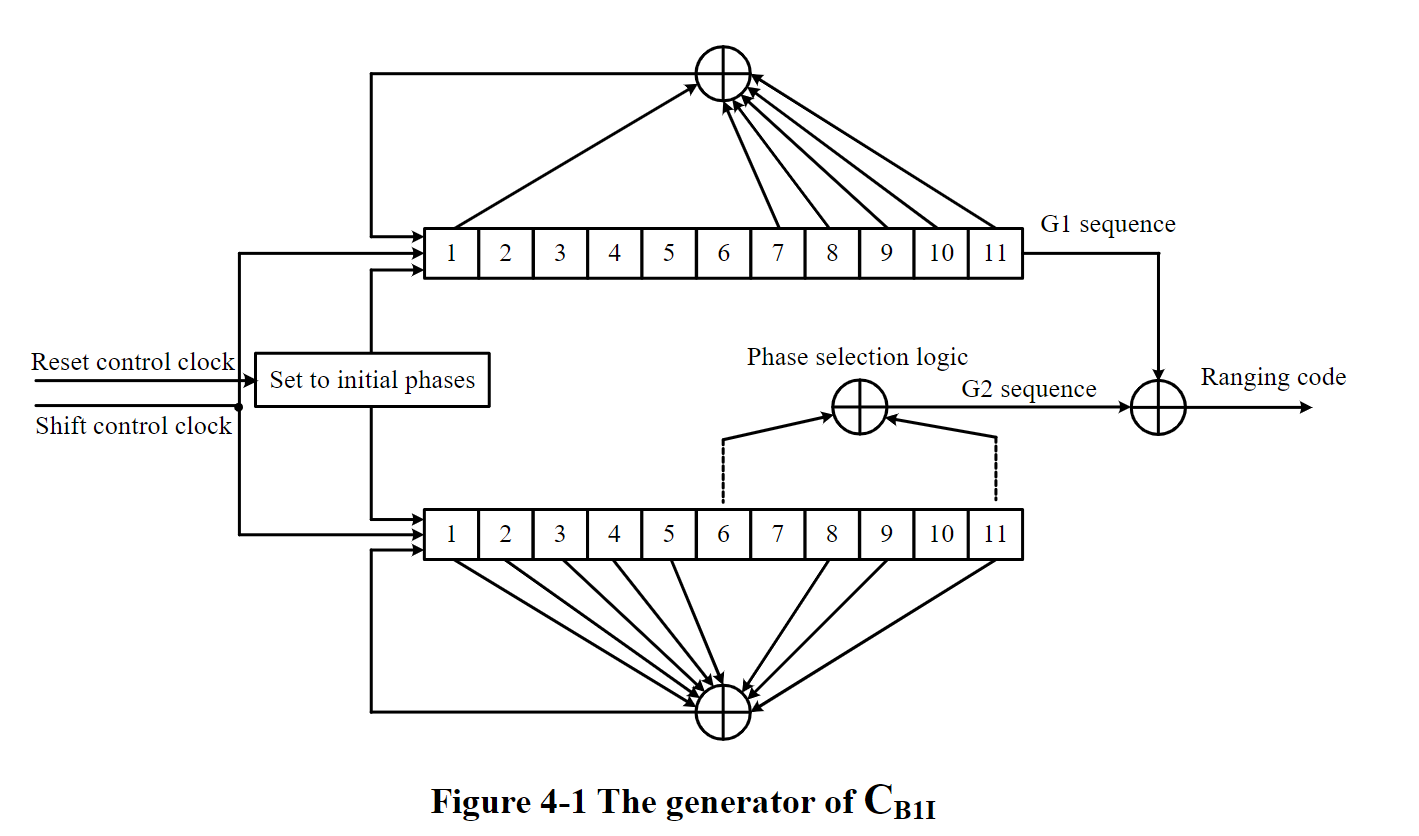
载波频率

测距码的码片速率为，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——的生成电路。

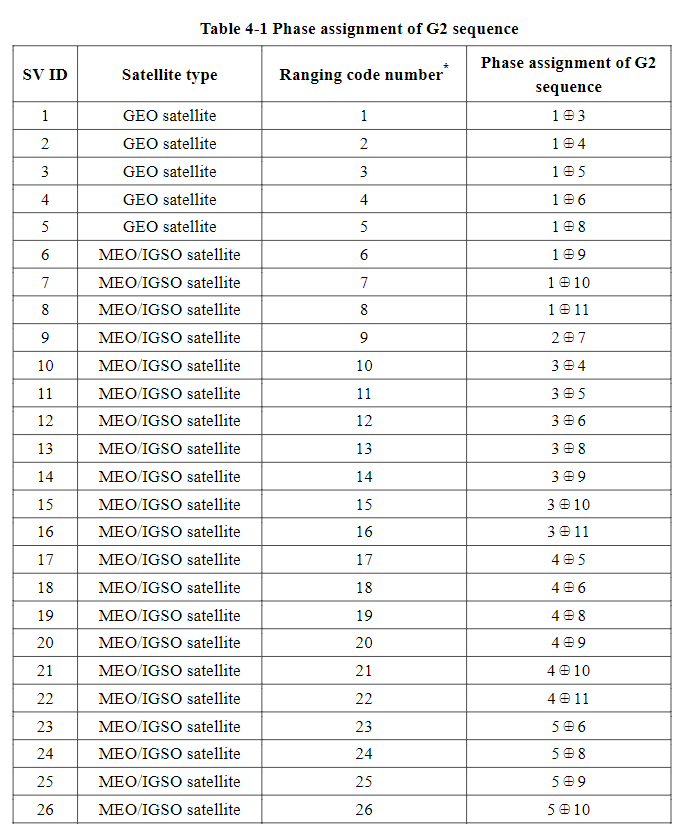
（3）的生成电路

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。



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



G2序列的相位分配表（部分）

（4）各信号的关系

数据的码率为500bps，测距码的码率为，因此一个对应4092个（两个循环周期）。载波频率为，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。一个对应个周期，也即一个对应个载波值。我们以频率作为系统时钟。

（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：

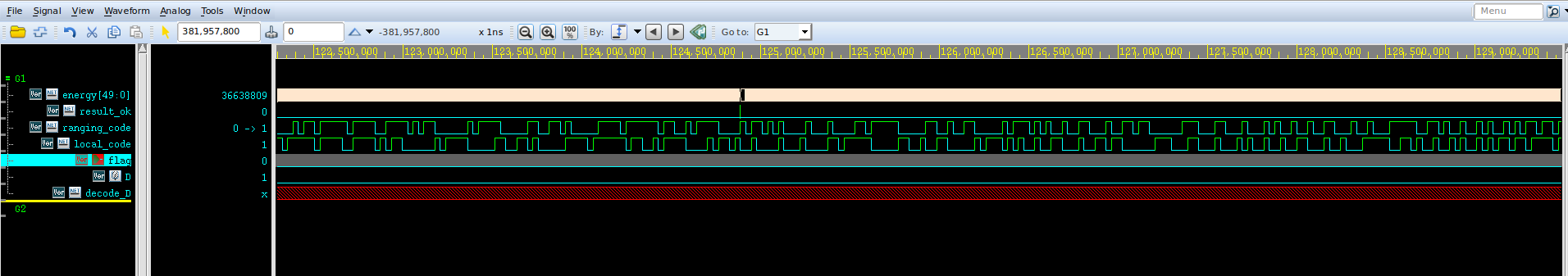
|  |
| --- |
| module control\_beidou (      input clk,      input rst\_n,      input delay\_en,      output reg shift\_parse  );        reg [11:0] counter; // 12 位计数器，用于计数 3052 个时钟周期      always @(posedge clk or negedge rst\_n) begin          if (!rst\_n) begin              // 异步复位，当 rst\_n 为低电平时，计数器清零，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 |

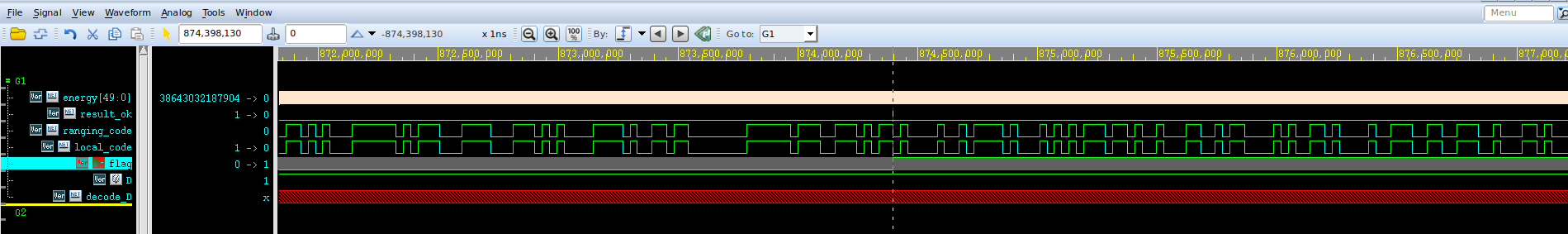
* 译码模块：

使用能量来作为译码的判断依据

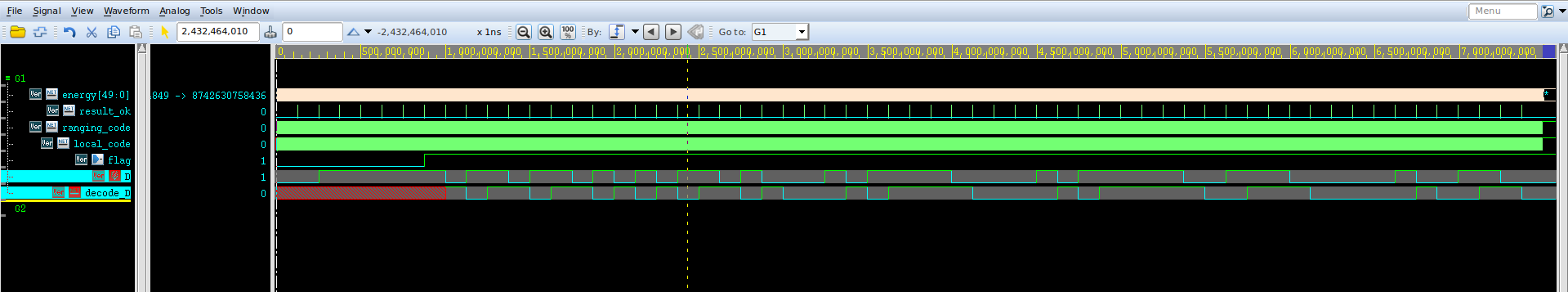
|  |
| --- |
| module decode\_beidou (      input flag,      input signed [49:0] energy,      input result\_ok,      output reg decode\_D  );      always @(posedge result\_ok) begin          if (flag) begin              if (energy < 50'd15000000000) begin                  decode\_D <= 1'b0;              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。