对程序的描述：

SPI_Slave.v:

介绍：是一个Slave文件。

MOSI上面，接受一次一位数字

MISO上面，一次输出一位

CS_n 保持0 时， 可以支持 multiple bytes per transaction（一次传输多个bit ）

i_Clk 必须 至少 比 i_SPI_Clk 快 4x

MISO 不传输的时候为三态（z）

并且有 SPI_MODE

对程序的描述：

module部分：

I/O内容1：

i_Rst_L 用于rst，0是rst

i_Clk 整体的clk

reg o_RX_DV 数据脉冲内容，1是有效，0无效数据，与o_RX_Byte一起

reg o_RX_Byte[7:0] MOSI接收到的数据存储

i_TX_Byte[7:0] MISO的数据流

i_TX_DV MISO的数据流的接口,1是有效，0无效数据

相对应的各个部分在程序中的描述：

i_SPI_Clk SPI的clk

o_SPI_MISO MISO的输出

i_SPI_MOSI MOSI的输入

i_SPI_CS_n SPI的CS， 0是选择，1是不选择

rst : i_Rst_L

clk : i_Clk, i_SPI_Clk

MOSI : reg o_RX_Byte[7:0] reg o_RX_DV i_SPI_MOSI

MISO : i_TX_DV i_TX_Byte o_SPI_MISO

CS_n : i_SPI_CS_n

RX 为 MOSI

TX 为 MISO

对程序的描述：

wire:
w_CPOL
w_CPHA
w_SPI_Clk (根据设定会进行更改)
w_SPI_MISO_Mux MISO的线（一根）

reg:
MOSI:
[2:0] r_RX_Bit_Count MOSI
[7:0] r_Temp_RX_Byte MOSI
[7:0] r_RX_Byte
r_RX_Done, r2_RX_Done, r3_RX_Done;
MISO:
[2:0] r_TX_Bit_Count
[7:0] r_TX_Byte
r_SPI_MISO_Bit,
r_Preload_MISO

CPHA=0 先进后出 CPOL=0 leading edge is rising edge.
CPHA=1 先出后进 CPOL=1 leading edge is falling edge.

测试中为了方便测试使用，又增加了input [1:0]　SPI_MODE

测试：

测试时候需要注意的：

i_Rst_L　　用于rst，0是rst
i_Clk　　整体的clk
i_SPI_Clk　　SPI的clk

MOSI:

reg o_RX_DV　　数据脉冲内容，1是有效，0无效数据，与o_RX_Byte一起
reg o_RX_Byte[7:0]　　MOSI接收到的数据存储
i_SPI_MOSI　　MOSI的输入

MISO:

i_TX_Byte[7:0]　　MISO的数据流
i_TX_DV　　MISO的数据流的接口,1是有效，0无效数据
o_SPI_MISO　　MISO的输出

i_SPI_CS_n　　SPI的CS， 0是选择，1是不选择

RX 为 MOSI
TX 为 MISO

改写：

鉴于该程序编写过于复杂，个人在分析与思考后，个人进行了一定的修改：
文件： SPISlave.v
各项输入输出端口

```
input [1:0]          SPI_MODE,
input                rst_i,                 // rst
output reg           EN_MOSI_ro,
        // 指示是否可以使用Slave内MOSI的数据了，可以使用的时候会输出一个脉冲高电平
input                EN_MOSI_i,             // 输入，指示此时MOSI的数据是否有效，有效时候应该是高电平
output reg [7:0]     MOSI_DT_ro,            // MOSI的数据输出，与上面的EN_MOSI_ro配套使用
input                EN_MISO_i,             // 指示是否接受 MISO,1是可以，0是不可以
input  [7:0]         MISO_DT_i,             // MISO 的 输入内容
output               EN_MISO_o,             // 输出，指示此时MISO线的数据是否可用，可以使用的时候是高电平
input                SPI_Clk_i,             // Clk
output               MISO_o,                // 两根线:MOSI与MISO
input    MOSI_i,
input    CS_i,                              // CS: 1为选择，0为不选择。 这里我们先指让CS对MISO有影响；

output reg [2:0] MISO_cnt_r,   // MISO cnt, 用于指示此时的MISO到第几个数字了,改成了output方便调试
output reg MISO_ro  //输出的reg,改成了output方便调试
```

改写：

```
// 各项寄存器
    reg [2:0] MOSI_cnt_r;  // MOSI cnt，用于接收数据时候计数，从 000 到 111
    reg [7:0] MOSI_DT_r;   // 用于存储已经接收到的MOSI信号
    reg MOSI_EN_r;
 // 用于确定确定此时的数据是否可以使用，并且这个是一个只会持续一个周期的小的脉冲,与上面的
EN_MOSI_o配套使用

    //reg [2:0] MISO_cnt_r;  // MISO cnt, 用于指示此时的MISO到第几个数字了,改成了output方便调试
    reg [7:0] MISO_DT_r;   // MISO DT,用于存储MISO这一轮的八个数字
    reg MISO_EN_r;         // MISO 是否可以使用。可以使用的时候会是一个高电平持续，EN_MISO_o配套用
    //reg MISO_ro;          //输出的reg,改成了output方便调试

    // 各项线路
    wire CPOL_w;
    wire CPHA_w;
    wire Clk_p_w; //这个依据不同mode进行定义
    wire MOSI_i_w; // 由CS控制
```

改写：

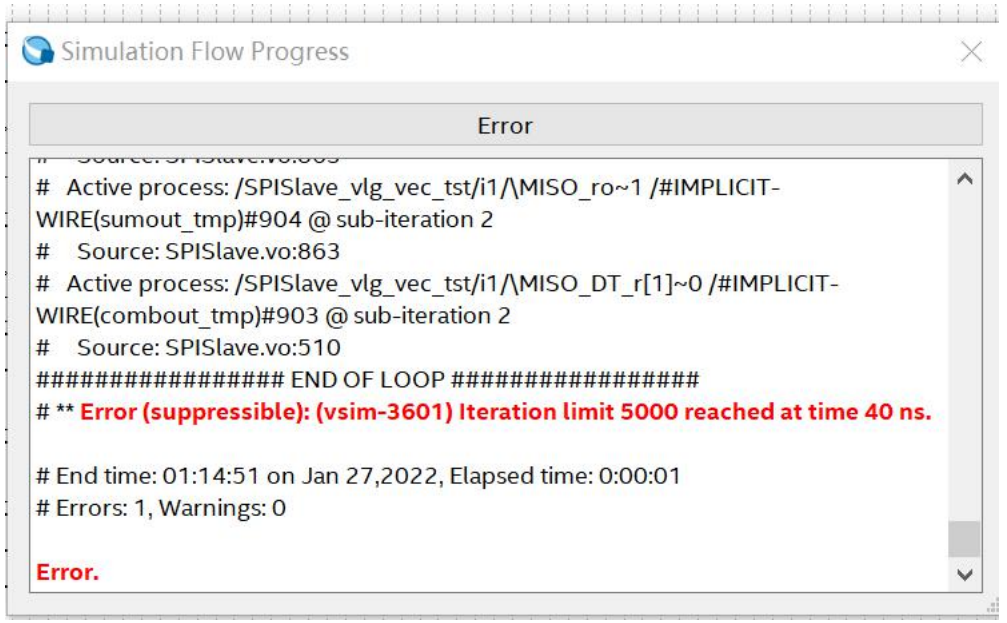
```
// 各项连线项目
        assign EN_MISO_o = MISO_EN_r;      //将输出和寄存器相连接
// 如果EN 或者 CS 为0，输出高阻态
        assign MISO_o = (MISO_EN_r && CS_i) ?  MISO_ro  : 1'bz;
        assign MOSI_i_w = (EN_MOSI_i && CS_i )  ? MOSI_i : 1'bz;


//时钟
// 1. CLK:
   // SPI_MODE有两位,第一位代表CPOL,第二位代表CPHA
   assign CPOL_w = SPI_MODE[1];
   assign CPHA_w = SPI_MODE[0];
   // 在设置完CPOL后可以对Clk_p_w定义
   assign Clk_p_w = CPOL_w ? ( CPHA_w? SPI_Clk_i : ~SPI_Clk_i ) : ( CPHA_w? ~SPI_Clk_i : SPI_Clk_i );
```

改写：

在程序中，分为两个组件：MOSI部分与MISO部分
具体程序执行方式在程序中已经以注释方式进行详细描述。
具体操作方式见后文ppt

其他说明:
1. MISO_cnt_r 在递增过程中，如果直接选择MISO_cnt_r <= MISO_cnt_r + 1; 会导致在模拟过程中发生无限震荡的错误（见右部），因此增加了MISO_cnt_after_r，在 posedge 的clk时候，MISO_cnt_r <= MISO_cnt_after_r



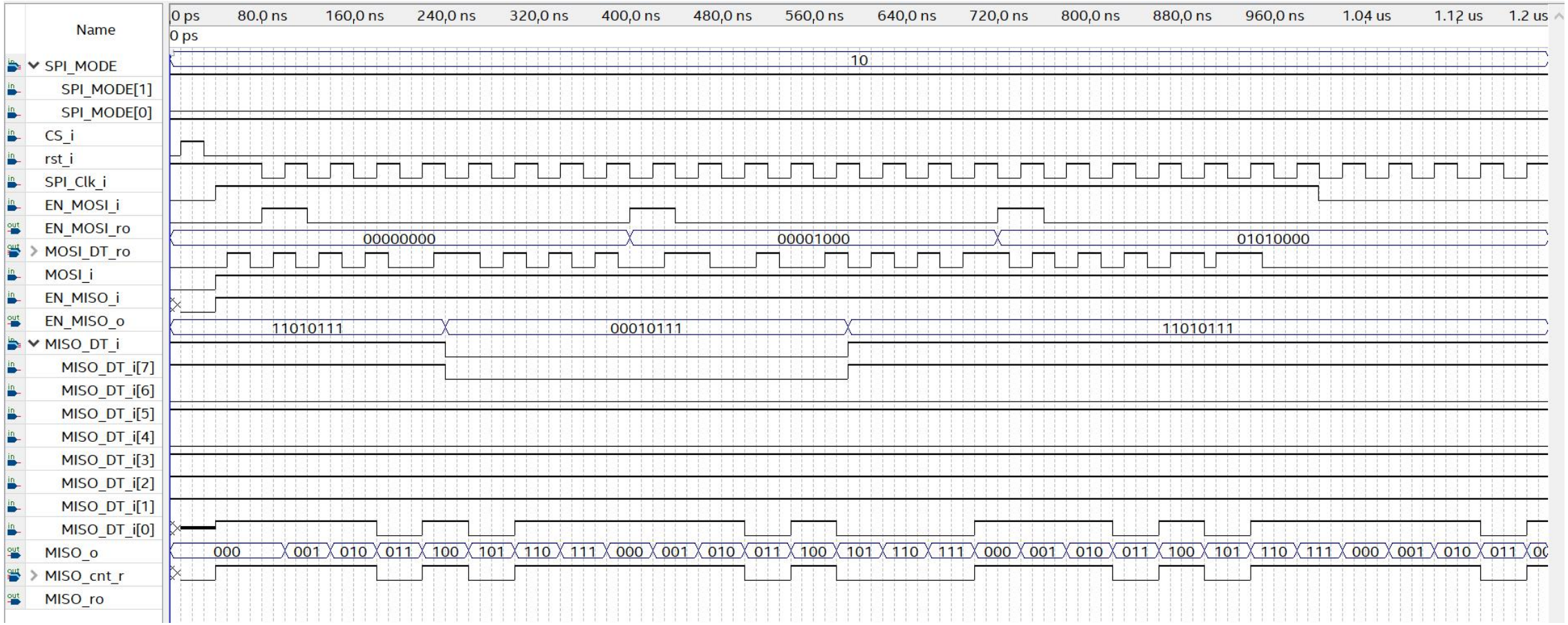
2.在测试中发现：MISO中理论上应该是posedge Clk_p_w时，修改cnt（根据上文中1的情况），negedege Clk_p_w修改其他寄存器，**但是实际操作时候却需要相反：如修改内部输出的为posedge rst_i or posedge Clk_p_w or posedge EN_MISO_CPHA,** 这一项没有解决，但是目前按符合要求的情况进行编写

3. 各类信号操作需要按照要求进行否则可能会出问题。同时如果有必要将来可以进行其他时钟信号控制

4.MISO的数据输出与输入相反（也就是输入时[0:7]会变成[7:0]),但是可以容易解决，在这里不进行重复修改

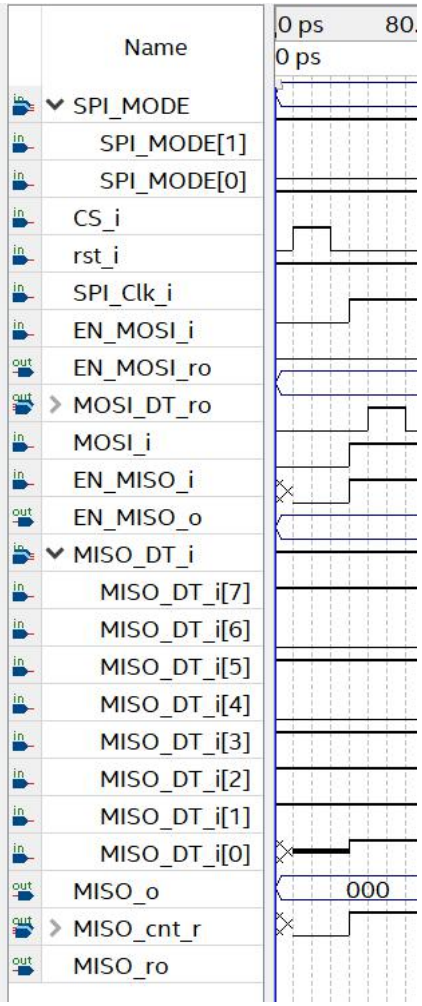
操作方式：

接下来以各种不同的模式下实现对于操作模式的演示：
1. 各类输入信号
信号范例：



操作方式：

1. 各类输入信号



所有i后缀为输入（包括SPI_MODE),所有o后缀为输出

SPI_MODE:两位数字，第一位为CPOL,第二位为CPHA （一般不修改）；

CS_i: 1. CS_i为高电平时候，MOSI才会收取当前数字；

2. CS_i 为低电平，MISO_o为高阻（但是对内部寄存器不进行影响）

rst_i: 重置信号，需要输入一个高电平脉冲以重置

SPI_Clk_i: （以CPOL为标准）

MOSI:

EN_MOSI_i: 和CS_i 一同指示当前数据是否收取入MISO中

EN_MOSI_ro: 指示当前的MOSI_DT_ro是否可以使用，当可以使用时，会输出一高电平脉冲

MOSI_DT_ro: 接收到的上8个MOSI数据所组成的信号

MOSI_i: MOSI线

MISO:

EN_MISO_i: 指示此时我们输入的8个数字是否可以使用，可以时候需要保持高电平；

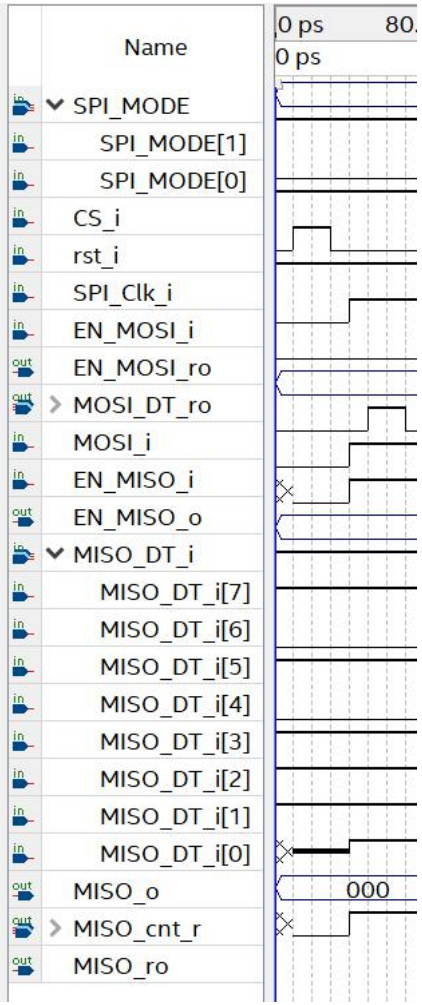
为了在不同模式下都可以正常运行，需要在SPI_Clk_i的第一个边缘之前设置为高电平，否则会导致第一个数据的时间过短

EN_MISO_o: 指示此时的MISO_o是否可以使用，可以时候是高电平

MISO_DT

操作方式：

1. 各类输入信号



所有i后缀为输入（包括SPI_MODE),所有o后缀为输出

MISO:

EN_MISO_i: 指示此时我们输入的8个数字是否可以使用，可以时候需要保持高电平；

为了在不同模式下都可以正常运行，需要在SPI_Clk_i的第一个边缘之前设置为高电平，否则会导致第一个数据的时间过短

EN_MISO_o: 指示此时的MISO_o是否可以使用，可以时候是高电平

MISO_DT_i: 次数需要输出的8个数字

因为程序中在第一个周期就已经完成了输入至寄存器的操作，在后面的过程中我们不需要再用该数据，并且如果修改过于缓慢还会导致下一周期中的数据错误，因此需要在第七个周期以及以前进行下一组（8个数字）的输入

MISO_o: 输出的线路MISO

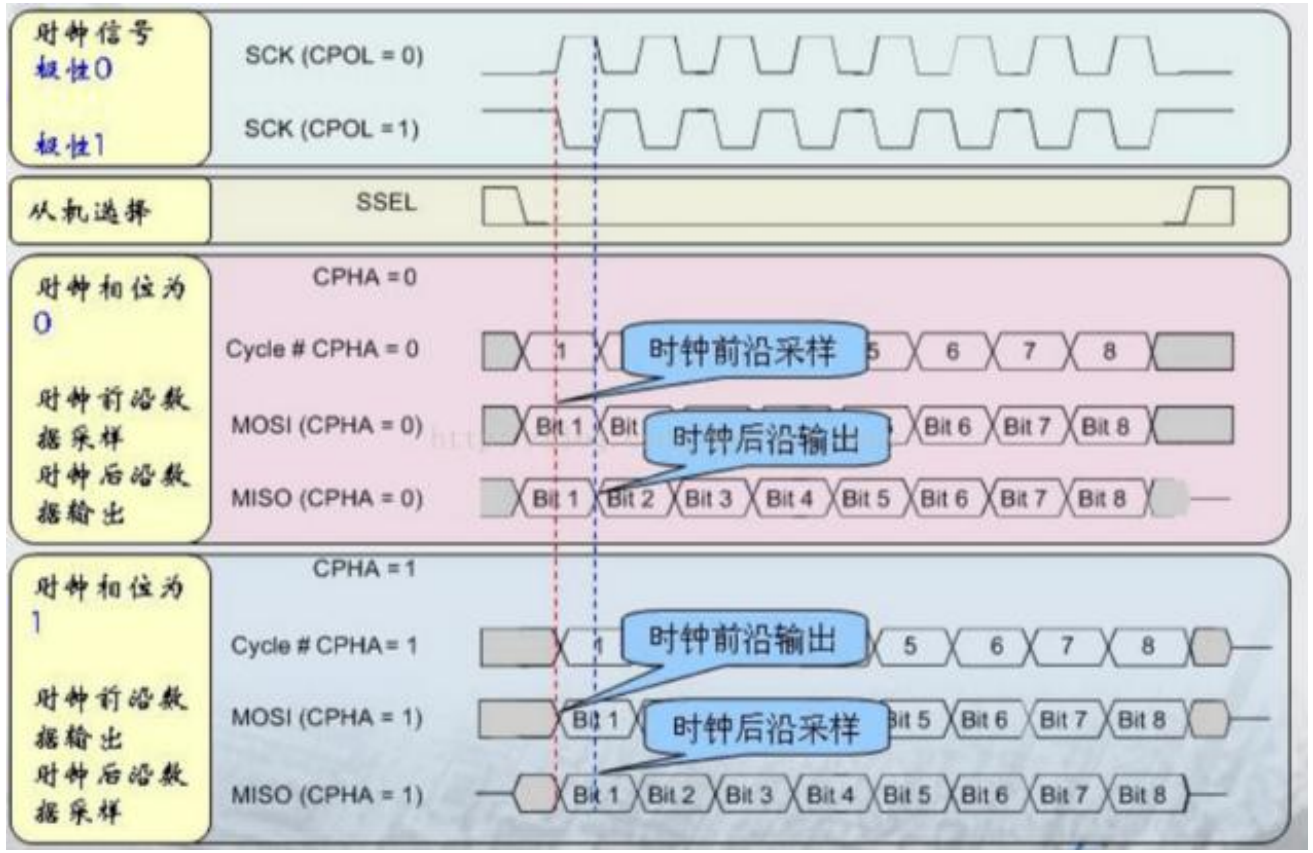
MISO_cnt_r: MISO当前输出的数据的编号（0到7）

MISO_ro: 与MISO_o相连的寄存器

操作方式：

2.使用范例

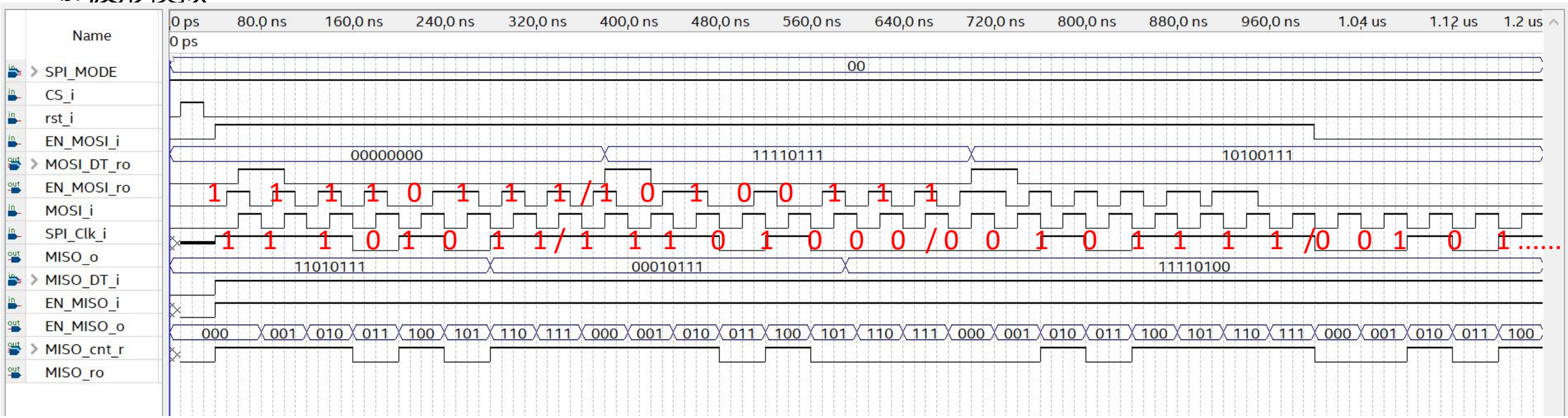
a. 模式解释



操作方式：

2.使用范例

b. 波形模拟

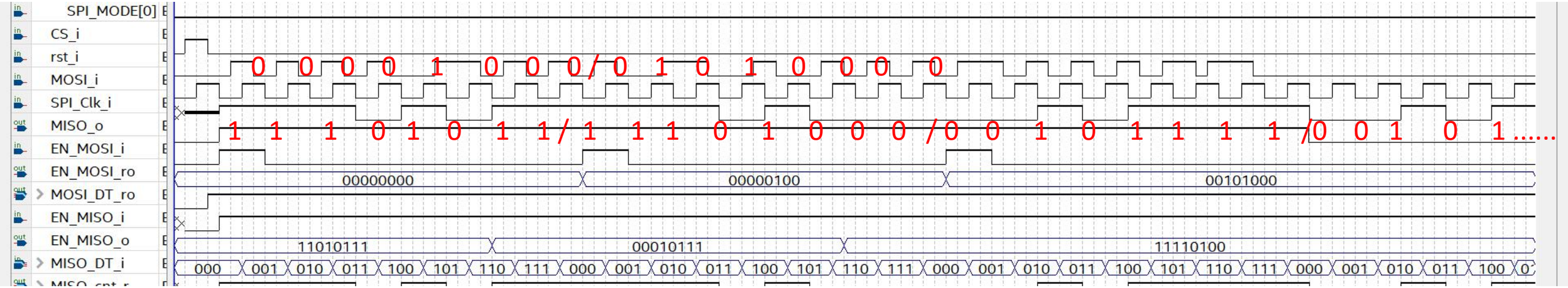


CPOL=0,CPHA=0,时钟前沿采样（MOSI），后沿输出（MISO），
前沿为上升，后沿为下降

操作方式：

2.使用范例

b. 波形模拟

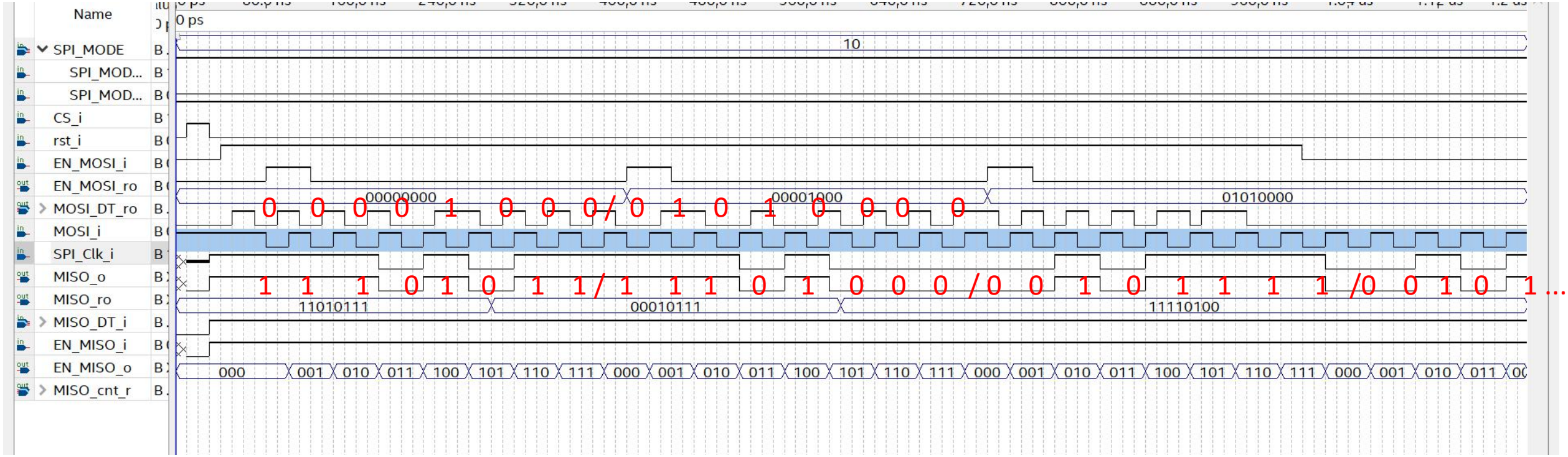


CPOL=0,CPHA=1,时钟前沿输出（MISO），后沿采样（MOSI），
前沿为上升，后沿为下降(在前沿时候数据是下一个数字的)

操作方式：

2.使用范例

b. 波形模拟

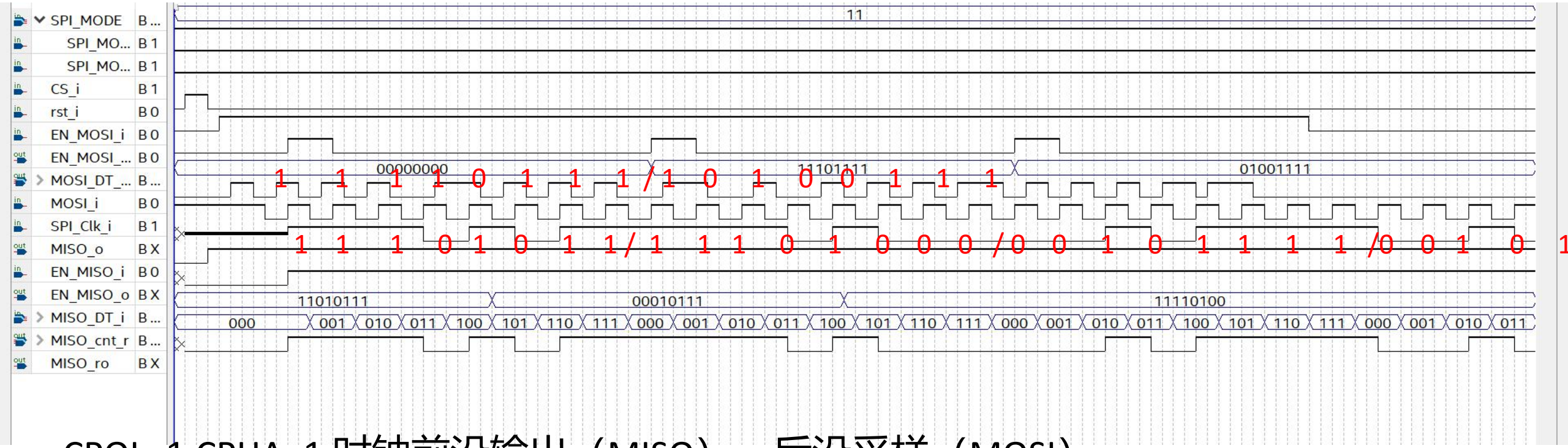


CPOL=1,CPHA=0,时钟前沿采样（MOSI），后沿输出（MISO），
前沿为下降，后沿为上升

操作方式：

2.使用范例

b. 波形模拟



CPOL=1,CPHA=1,时钟前沿输出（MISO），后沿采样（MOSI），
前沿为下降，后沿为上升
(在前沿时候数据是下一个数字的)