使用fpga进行huffman编码

步骤：

1. 收集并统计【0-9】中每个数字【4位】出现的次数（256次）【9位】（包含0和256）
2. 进行哈夫曼编码
3. 找出次数最小的两个数字
4. 合成两个数字
5. 重复（1）（2）构成哈夫曼树
6. 用哈夫曼树进行编码
7. 进行并行输出【1位】

Module getnum

输入信号：Clk\_in,nRst,Start, Data\_in(输入数字)

输出信号：Num0-9(用于统计数字0-9出现次数)

内部信号：Gn\_en（使能信号）,Num\_time（数字出现总次数最多256次）

Start下降沿触发，Gn\_en开始使能

每个clk\_in

~nRst初始化

Gn\_en为1，对对应数字进行计数，若不为0-9直接结束计数（使能信号置0）（自由性）

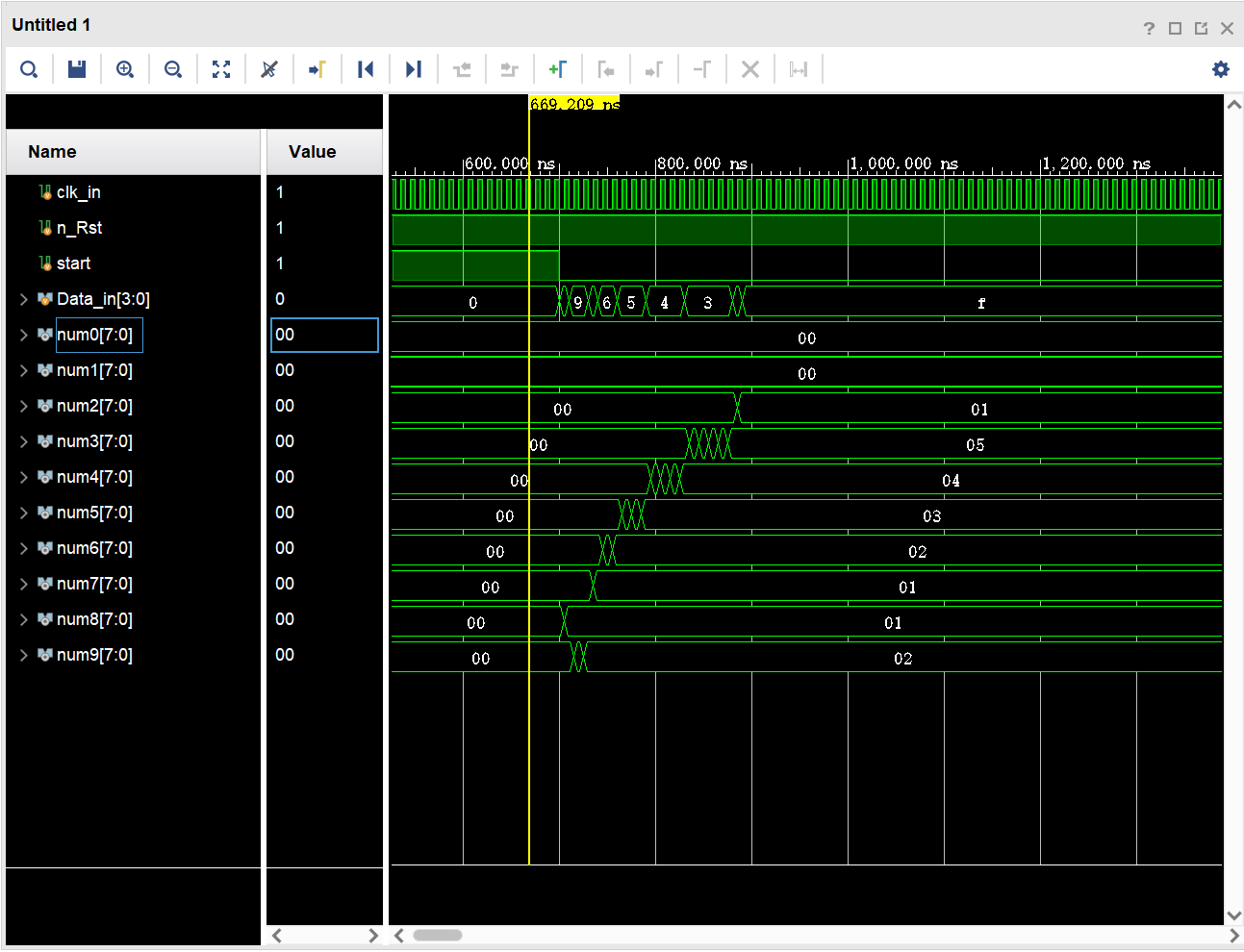
每个clk\_in

Num\_time不为256时，Num\_time值加1

Num\_time为256时，使能信号置0结束计数

测试成功

已修改成9位



2.哈夫曼编码（9位）

（\*）特殊情况：一个数为256，其他为0

查找num0-9中第一位是否有1

中间通过m1=11111进行标记

输出预先设计的代码

Max-0 0001 0000

其他按0-9的顺序序为

0 0000 1110

0 0000 1100

0 0000 1010

0 0000 1000

0 0000 0110

0 0000 0100

0 0000 0010

0 0000 0001

0 0000 0000

👇已重构别看

设置min1，min2【9位】初始化为1 0000 0000（最大值）,m1,m2=0000

遍历寻找min1 然后记录位置（0-9）进m1 遍历时忽略首位为1的数

修改m1位置的记录，首位改为1（特殊情况已经除外，首位必为0）

M2与m1同方法。

比大小必须使用阻塞赋值

新增8个中间状态（位数调整为5）num10-num17初始化为1 0000 0000

使用min1，min2（9位）提取m1，m2位置的数字后相加记录进num10-17位置（5位）（用一个额外寄存器count计数初始为10每次加1）（首位1+1=0，若首位出现1说明min1+min2=1 0000 0000数字全部统计完了，直接终止），正常情况m2位置上的数首位为0，可以被正常读取。构建表 格式：[count][m1][m2](15位),共8行。最后终止时m1和m2分别为256的左枝和右枝//导致可拓展性不高的问题

👆构造哈夫曼树（已重构，别看了）

构造哈夫曼树

输入信号：Clk\_in,n\_Rst,Start\_tree,Num0-9

输出信号：Tree0-7 保存哈夫曼树的表，格式为[count][m2][m1](min2>min1)

m1作为输出时为根节点的右孩子，运行时为节点的右孩子

m2作为输出时为根节点的左孩子，运行时为节点的左孩子

内部信号：min用于记录找到的频率最小值

mm用于记录频率最小值的地址（对应的数值）

min1最小值1

min2最小值2 min2>min1

count记录循环次数，同时作为表头地址

Tr\_en使能信号

NNum0-9 Num0-9的对应寄存器，用于修改其中的值

min1\_2 min1+min2的值

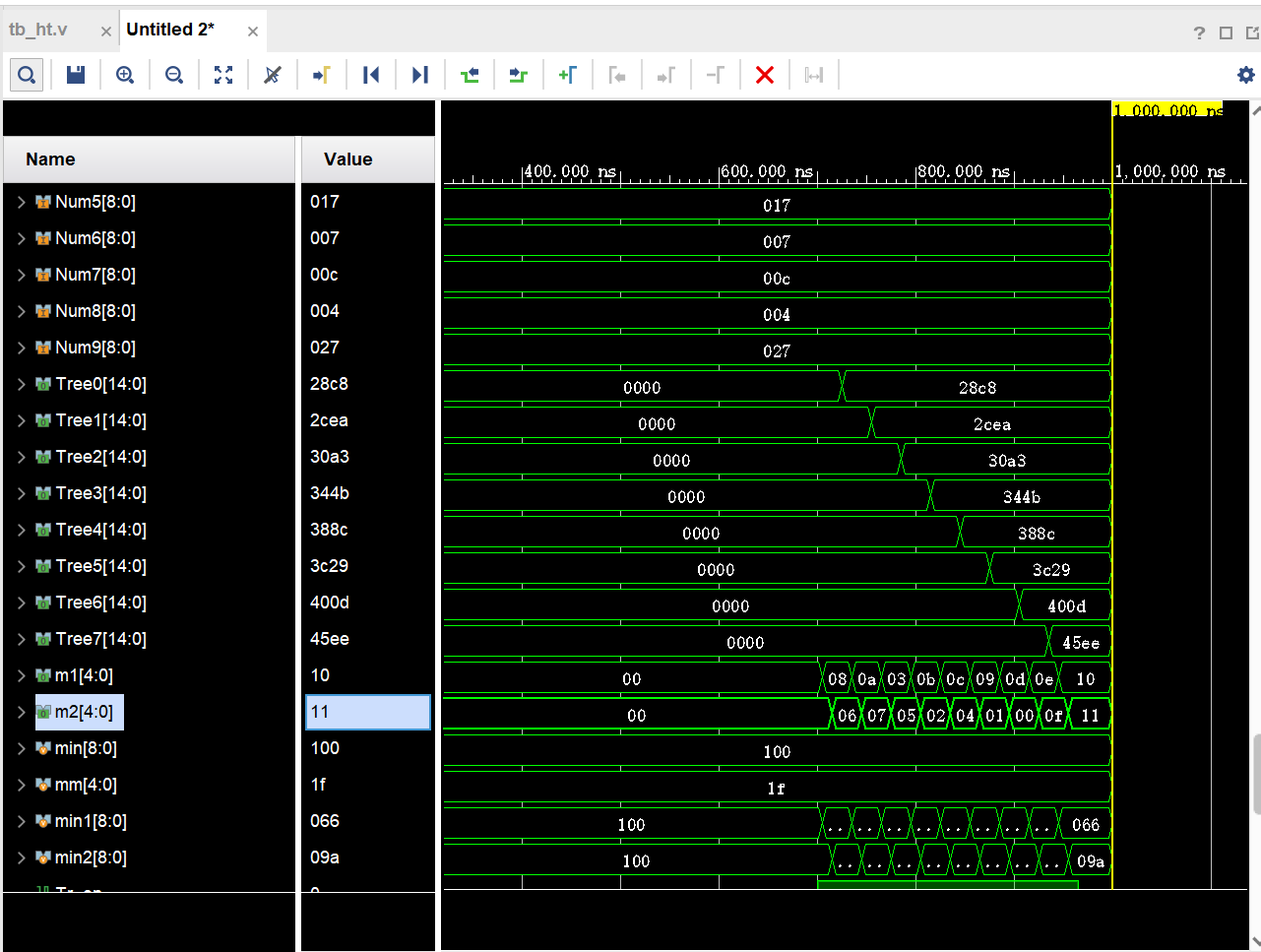
state表示哈夫曼树构造时的状态，0表示寻找min1，1表示寻找min2，2表示生成哈夫曼树，构建表的过程

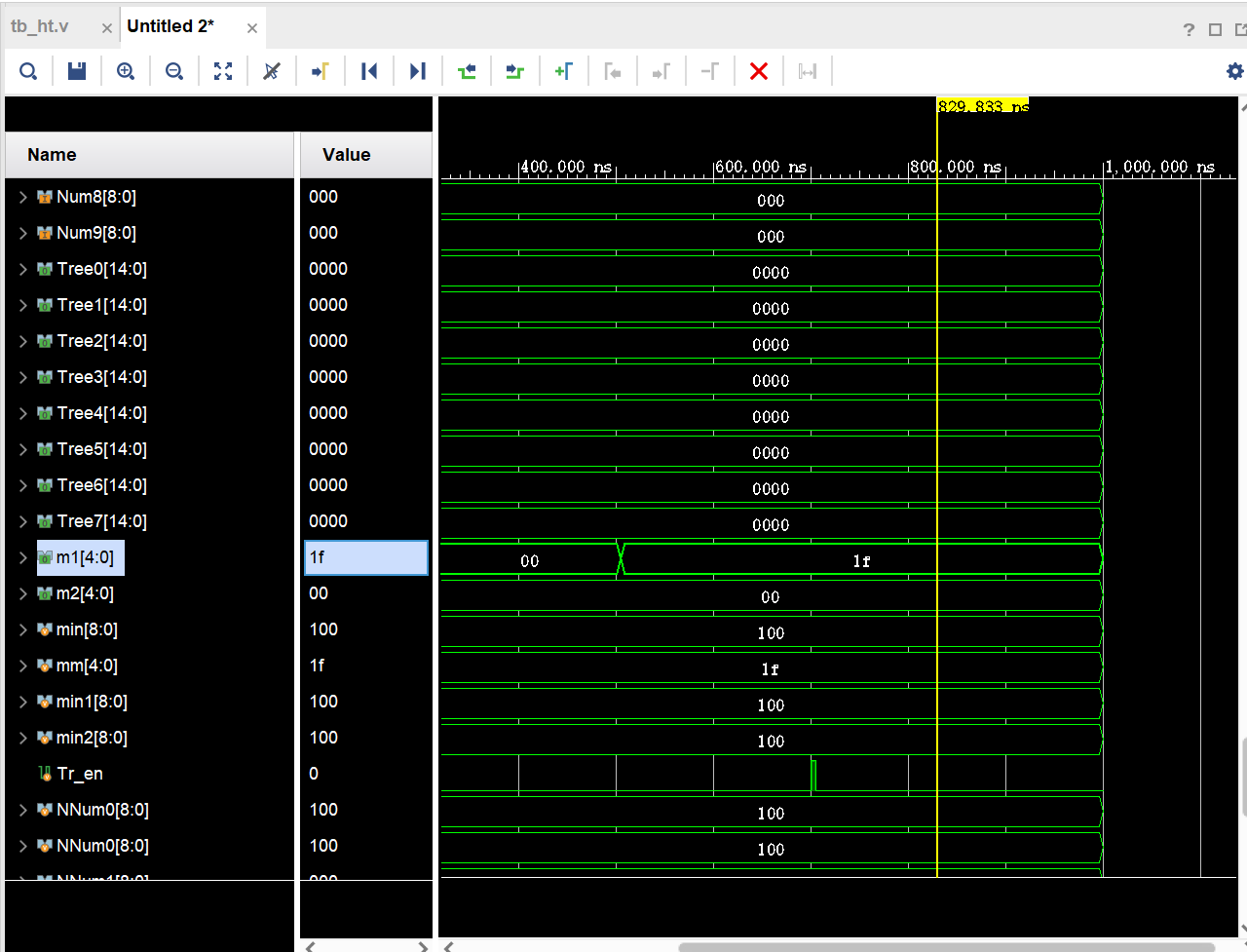
总体框架为0-2三个阶段，交替运行，总共运行9次完成哈夫曼树的构建（10个叶子节点，二叉树，需要构建九次树，8个表加最后256的两个分枝可以完成树的构建）。

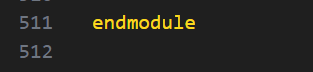
阶段0：寻找min1，通过遍历NNum0-9,Num10-17寻找其中的最小数，因为中间过程的min被初始化为0’h100，所以min不会寻找到第九位为1的数，通过给第九位置1表示已经被作为最小数提出过。

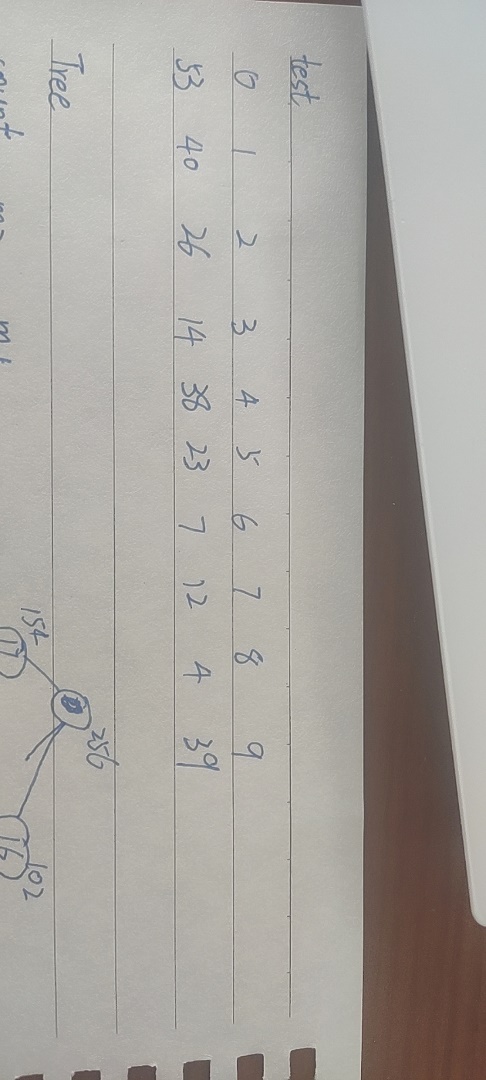
阶段1：寻找min2，方法同上。

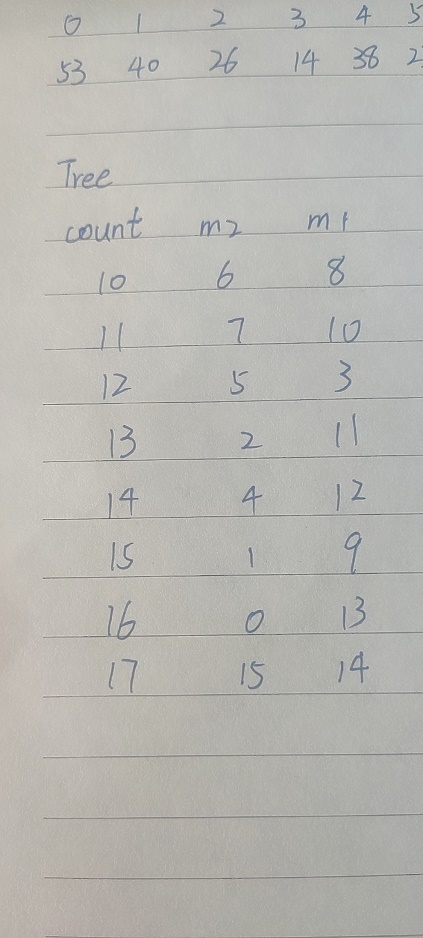
阶段3：计算min1\_2=min1+min2，通过count的值进行表的构建，表构建为[count][m2][m1]，这个阶段完成后count的值+1，代表进入下个循环，循环九次，第九次不构建表只进行两次最小值检索。

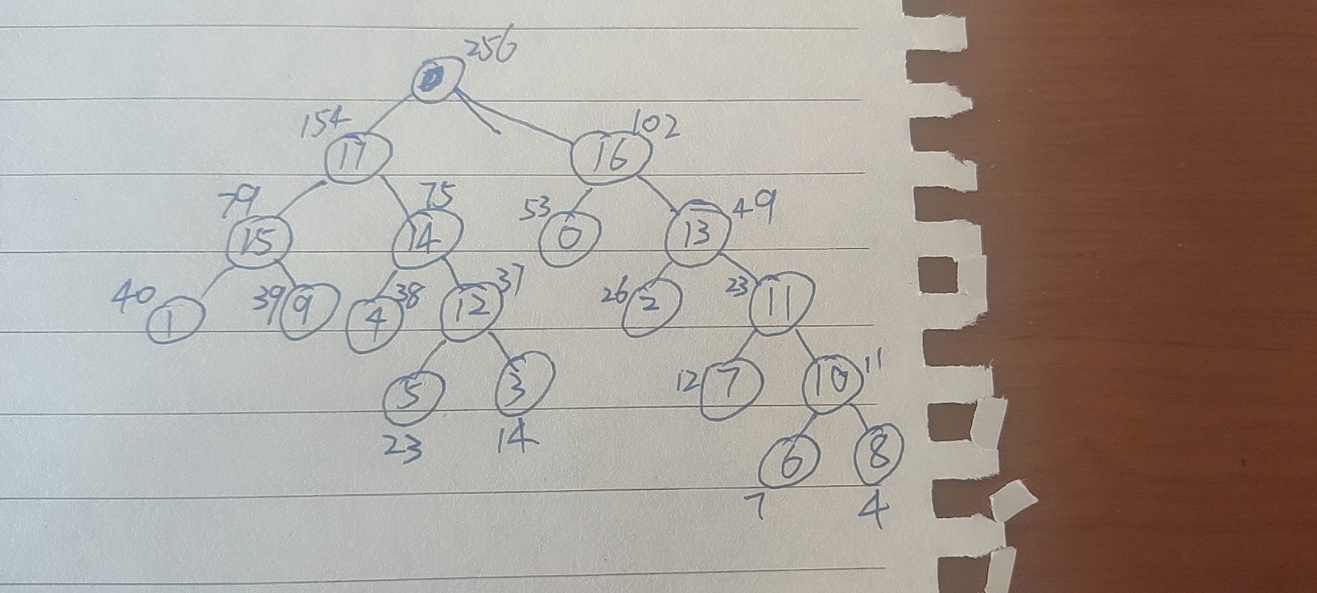




 512行代码太完美了



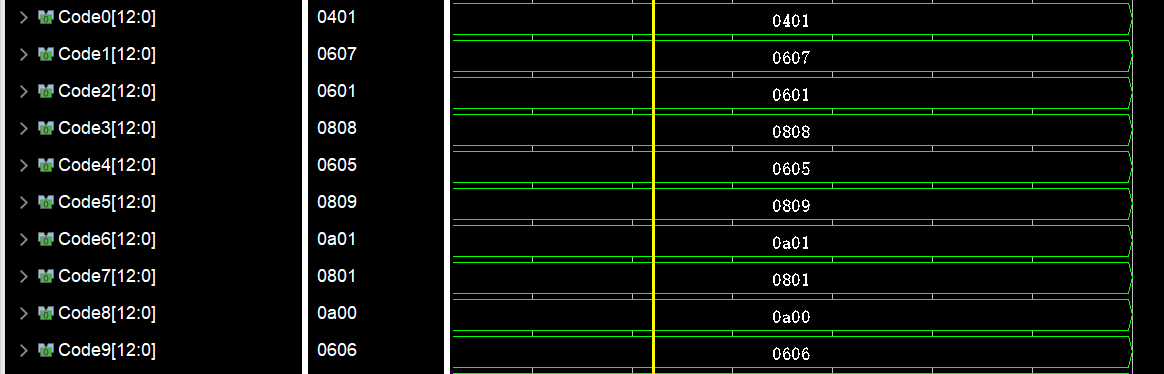




👇哈夫曼树转换成哈夫曼编码

使用前序遍历遍历二叉树

成啦兄弟



输入信号：Clk\_in,n\_Rst,Start\_code,Tree0-7,m1,m2

输出信号：Code0-7 （[code\_bit][code]）

内部信号：code(用于哈夫曼编码)，code\_bit（用于记录编码长度），code\_num用于记录code对应的值（暂时），Hc\_en(使能信号)（逻辑导致关闭使能信号的代码位置在状态1内），Node[8:0]（Node8=[18][m2][m1]根节点，剩下node与tree对应），state(状态0，准备状态和调整状态，准备状态为从根节点出发开始遍历，状态1左子树状态，状态2右子树状态)，flag(标记，用于区分准备状态的)（其实可以再补一个状态但是我写的时候头昏了）,temp用于装载数字

收到开始信号使能信号置1

初始化数值，包含特殊情况

若使能信号为1进入状态循环：

状态0：准备状态和调整状态

调整状态：flag==1，遍历所有左孩子右孩子，在表中删去与temp记录相同的值， 表现为将左右孩子位置的这个值改为5‘h1f，完成后flag置0。

准备状态：读取Node[8]，左孩子没读取完成则从左孩子开始，进入状态1，temp改为左孩子的值；左孩子已经读完则从右孩子开始，进入状态1，temp改为右孩子的值；调整状态完成后也会立刻进入准备状态。

状态1：左孩子

因为在所有code都被正确记录输出后，state会停止在1状态，所以1状态开始

要进行一次判断来关闭使能信号。

若Node[temp]即准备状态给出的值对应的表头，这个值对应的左右孩子若均为1f则说明左右孩子都被读取过，进入调整状态（state=0，flag=1），将表头值传给temp，在调整状态中删去存在于左右子树的表头，说明这个表头被读取过。

若Node[temp]的左子树为0x1f，此时右子树不为0x1f，进入状态2

若Node[temp]的左子树为【0x0a-0x11】之间的值，说明这个左子树依然为中间状态，这个左子树拥有左右子树，可以进入这个左子树，code左移一位并加一，code\_bit+1，state=1，temp更新

若Node[temp]的左子树为【0x00-0x09】之间的值，说明这个左子树的左子树没有左右孩子了，可以进行编码并将编码输出。完成编码输出后将这个编码完成的数字从邻接表中移去，即置为0x1f。回到准备状态

状态2：右孩子

右孩子操作大致于左孩子相同，只是修改了编码的方式。

输出数据