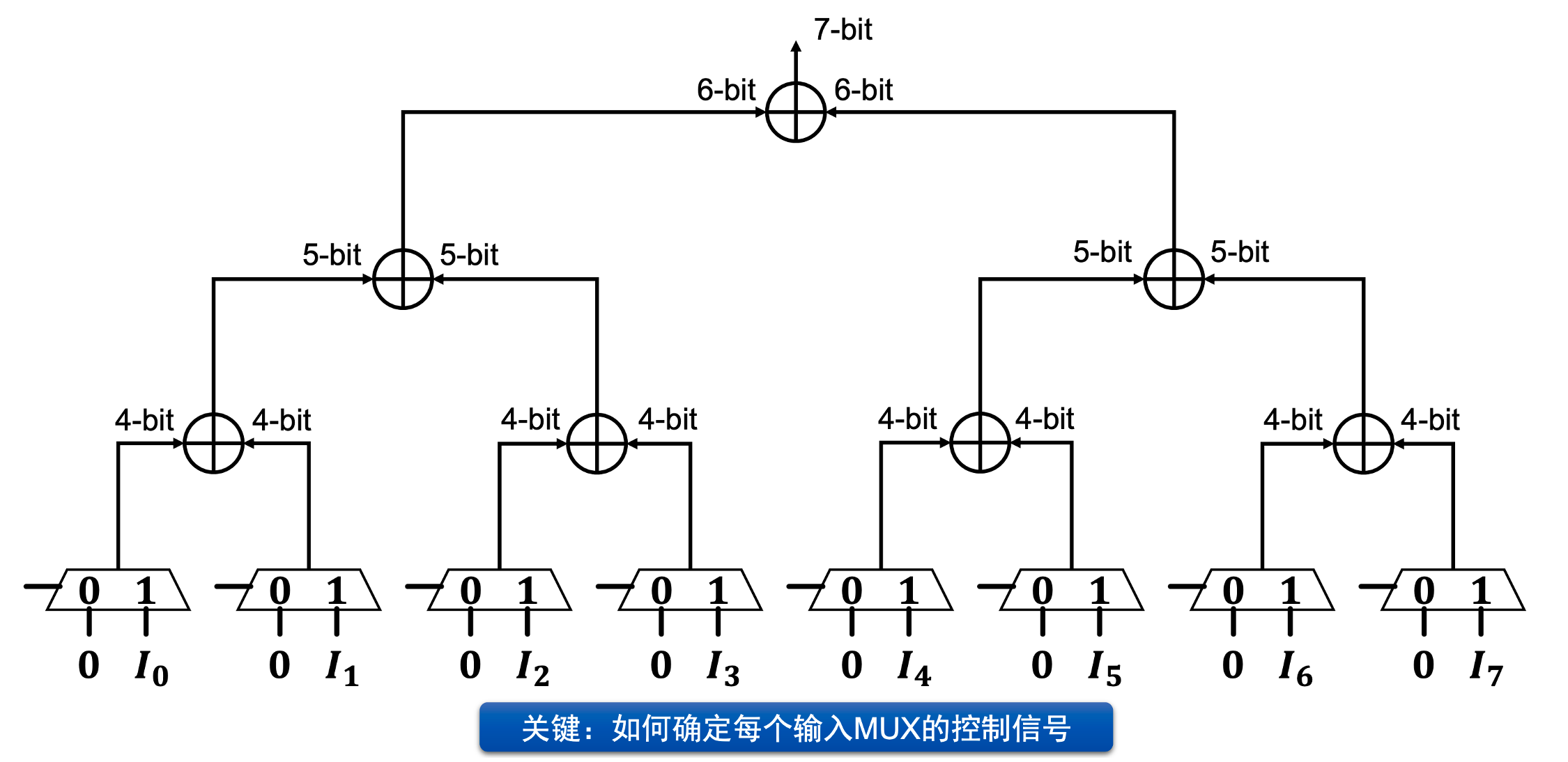
**实验报告**

**实验内容一**

**设计思路**

使用提示中的八输入加法树设计，根据MUX输入编号为MUX0~MUX7，MUX控制信号设置如下：

**MUX0：**M和m有一个是0则选择，否则选择0，因此

**MUX1：**M和m一个小于等于1一个大于1，或者有一个是1时，选，因此

**MUX2：**M和m一个小于等于2一个大于2，或者有一个是2时，选，因此

**MUX3：**M和m一个小于等于3一个大于3，或者有一个是3时，选，因此

**MUX4：**M和m一个小于等于4一个大于4，或者有一个是4时，选，因此

**MUX5：**M和m一个小于等于5一个大于5，或者有一个是5时，选，因此

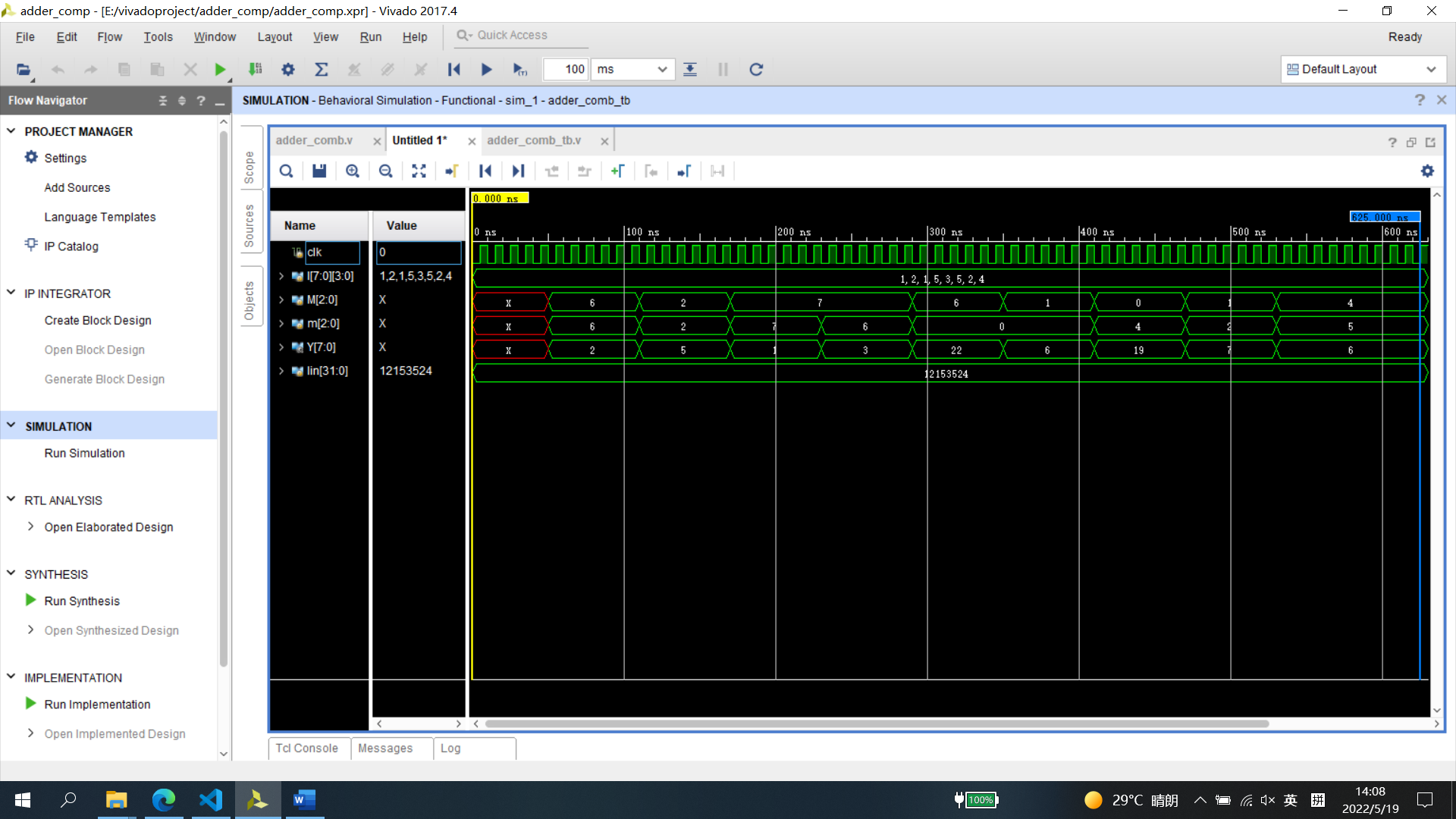
**MUX6：**M和m一个小于等于6一个大于6，或者有一个是6时，选，因此

**MUX7：**M和m有一个是7则选择，否则选择0，因此

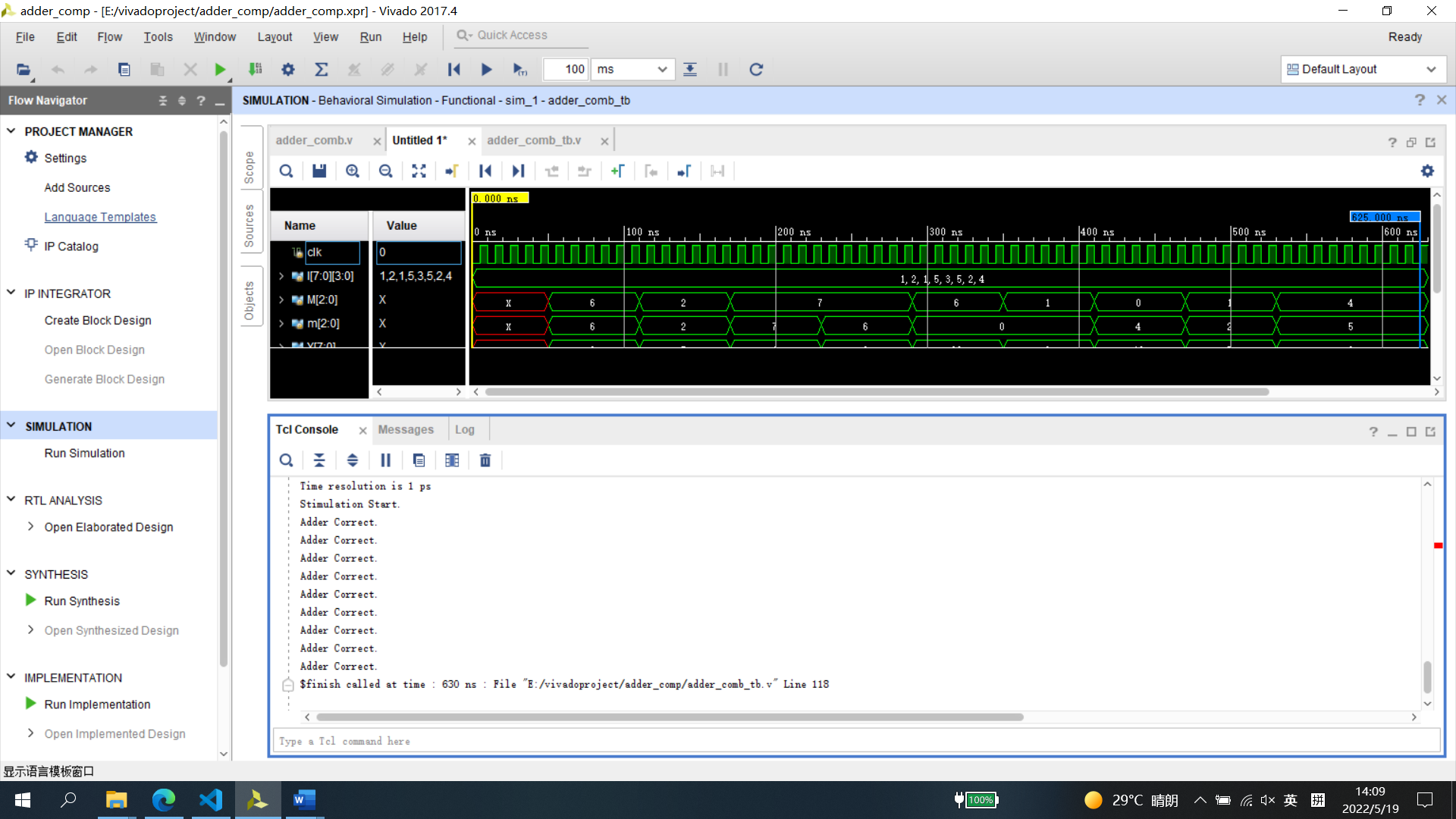
**代码逻辑**

32位输入Iin，两个3位控制信号M和m，首先将输入的32位Iin拆分成8个四位输入，接入到以上述信号控制的8个MUX中，在always语句中完成第一级4bit加法得到MUX0，MUX1输出的和s01，MUX2，MUX3输出的和s23，MUX4，MUX5输出的和s45，MUX6，MUX7输出的和s67，之后输入第二级加法器得到结果s0123和s4567，用assign语句赋值Y=s0123+s4567输出。

**验证结果**



仿真波形图



仿真输出结果，对3×3个测试样例设计的加法器都能输出正确结果

**实验内容二**

**设计思路**

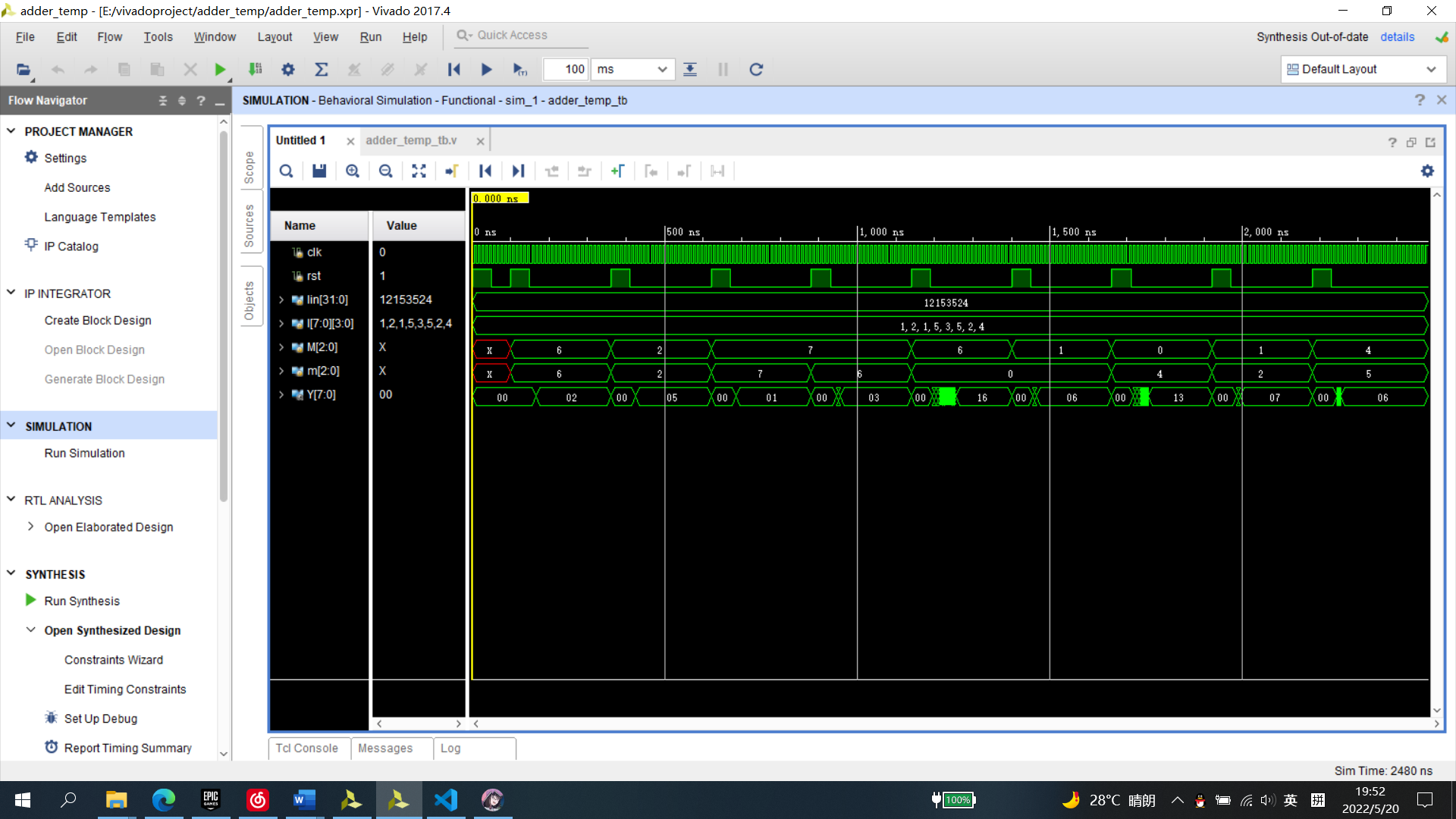
设计的关键是有限状态机的状态和状态转移，共设计了1+7+7个状态，其中包含一个复位态，7个M<m的状态，7个M≥m的状态，考虑到控制信号3位，加上一位表示M和m大小关系，所以用四位寄存器current\_state记录现态，用四位寄存器next\_state记录次态。复位之后根据M和m大小，决定次态的最高位是0还是1，次态的后三位由M和m的最小值提供。特殊情况M=m=0，则输出I[0]结束运算，M和m中只有一个为0，说明结果一定包含I[1],根据M和m大小跳到4'b0001(M<m)或4'b1001(M≥m)，之后继续进行运算。

每个状态都要判断max{M，m}和当前state的后三位是否相等也就是运算是否已经结束决定次态回到复位态还是向state+1转移继续运算。

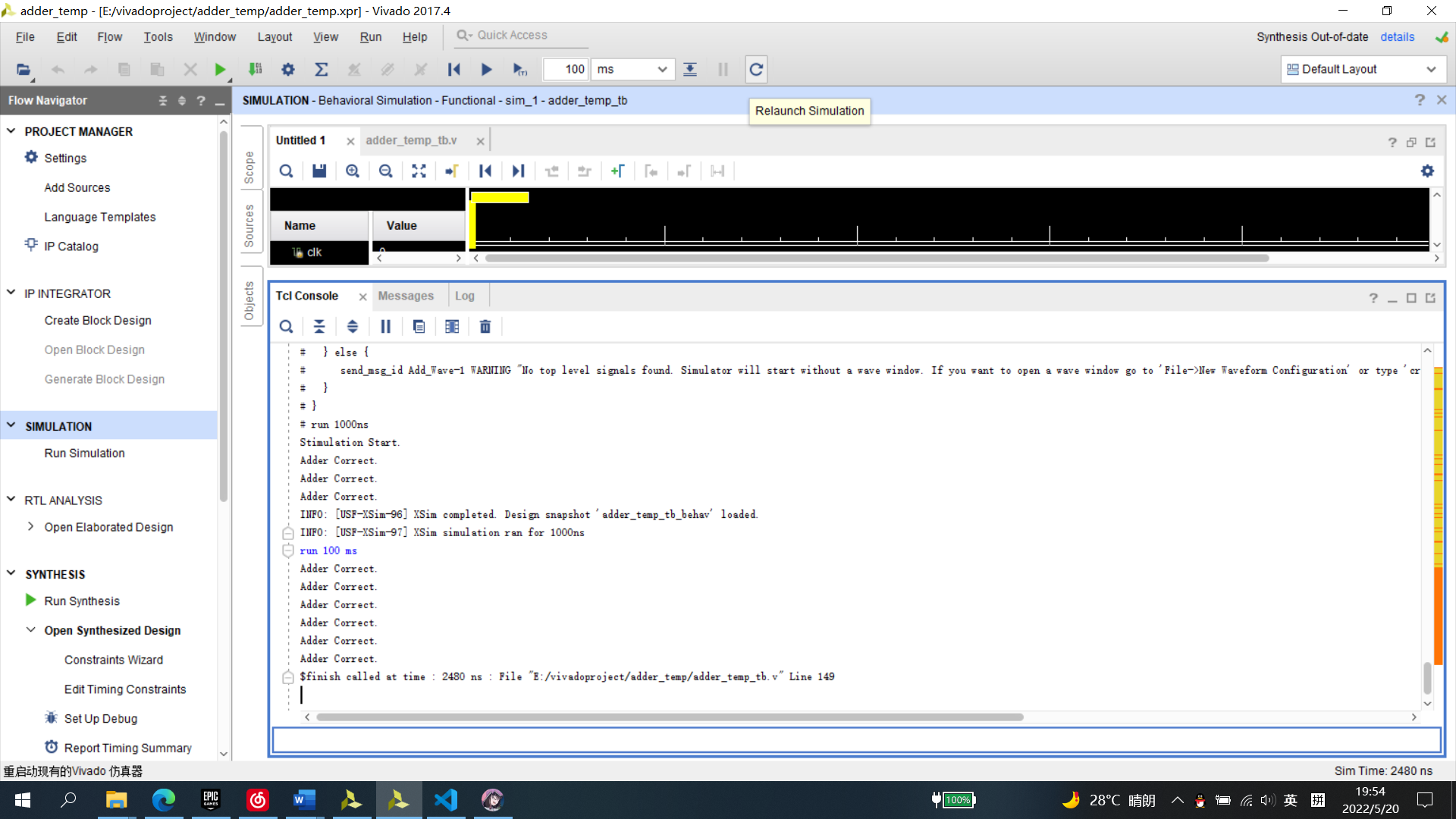
**代码逻辑**

32位输入Iin，两个3位控制信号M和m，一位reg add\_done指示运算是否已经结束，用宏定义将输入的32位Iin拆分成8个四位输入，两个4bit指示状态转移变量current\_state和next\_state。1bit时钟输入以及1bit异步复位信号输入。如果复位信号为1，则现态和次态都赋4‘b0000，输出Y赋0，add\_done赋0。使用case语句实现状态转移，如果复位信号为0并且add\_done也是0则进入状态转移语句，复位态4‘b0000，进入之后先按一般情况比较M和m大小，M<m则次态设置为{1’b0，M}，否则次态设置为{1’b1，m}，之后检查M和m中是否有0，M=0说明输出结果要加I[0]，直接把I[0]赋给Y，之后判断m是不是0，若是0则add\_done置1，直接结束运算，否则说明m≥1，次态设置为4‘b0001；如果M不是0，检查m是不是0，是0则把I[0]赋给Y，次态设置为4’b1001。其余状态编码的最高位指示M和m的大小关系，此状态应该加的I由后三位决定，每个状态在加和之后都要检查max{M，m}是否等于当前加和的I编号，以决定是否结束运算。Case语句之后将next\_state的值赋给current\_state，完成状态转移。

**验证结果**



仿真波形图



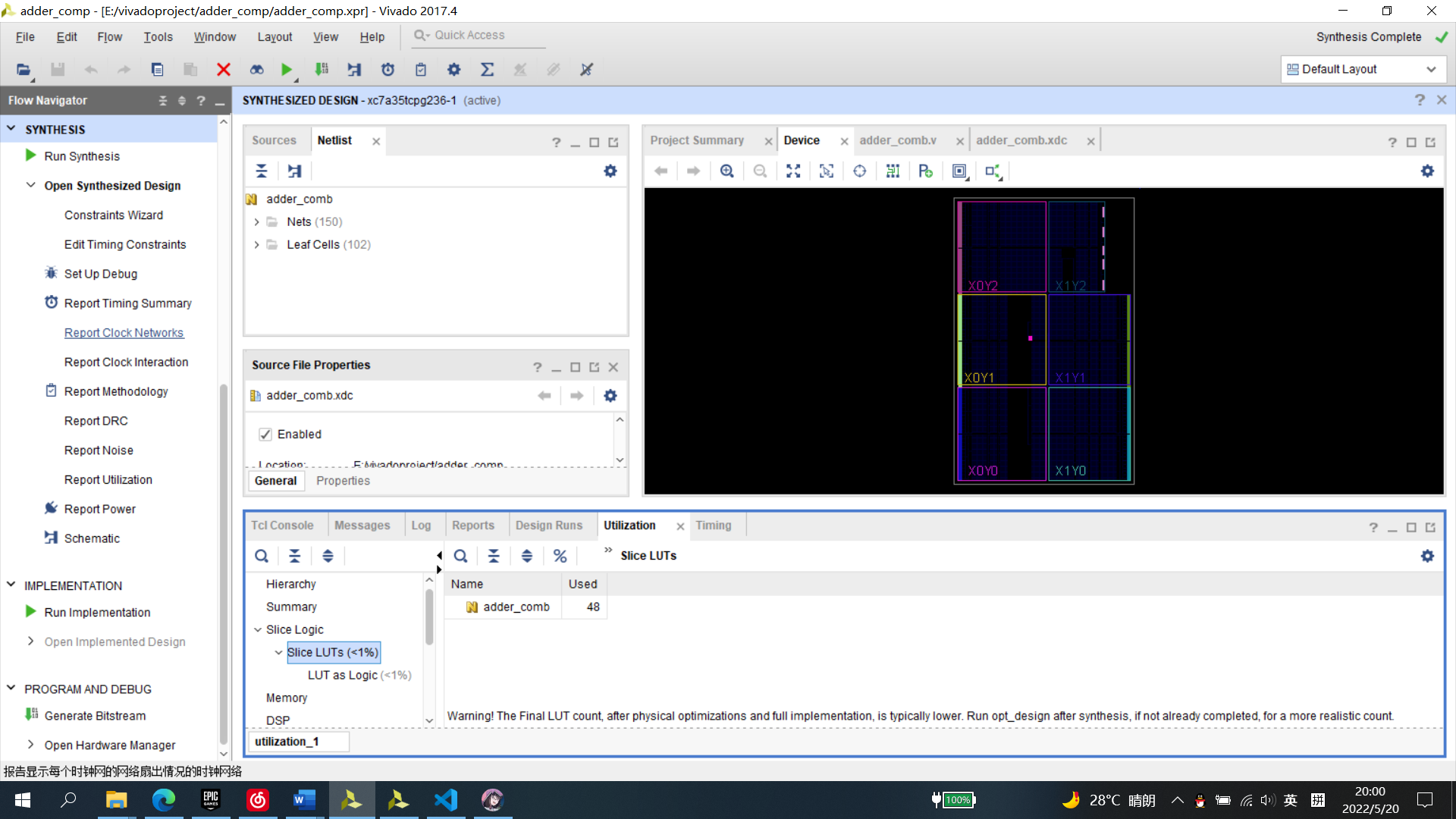
仿真输出结果，对3×3个测试样例时序逻辑设计的加法器都能输出正确结果

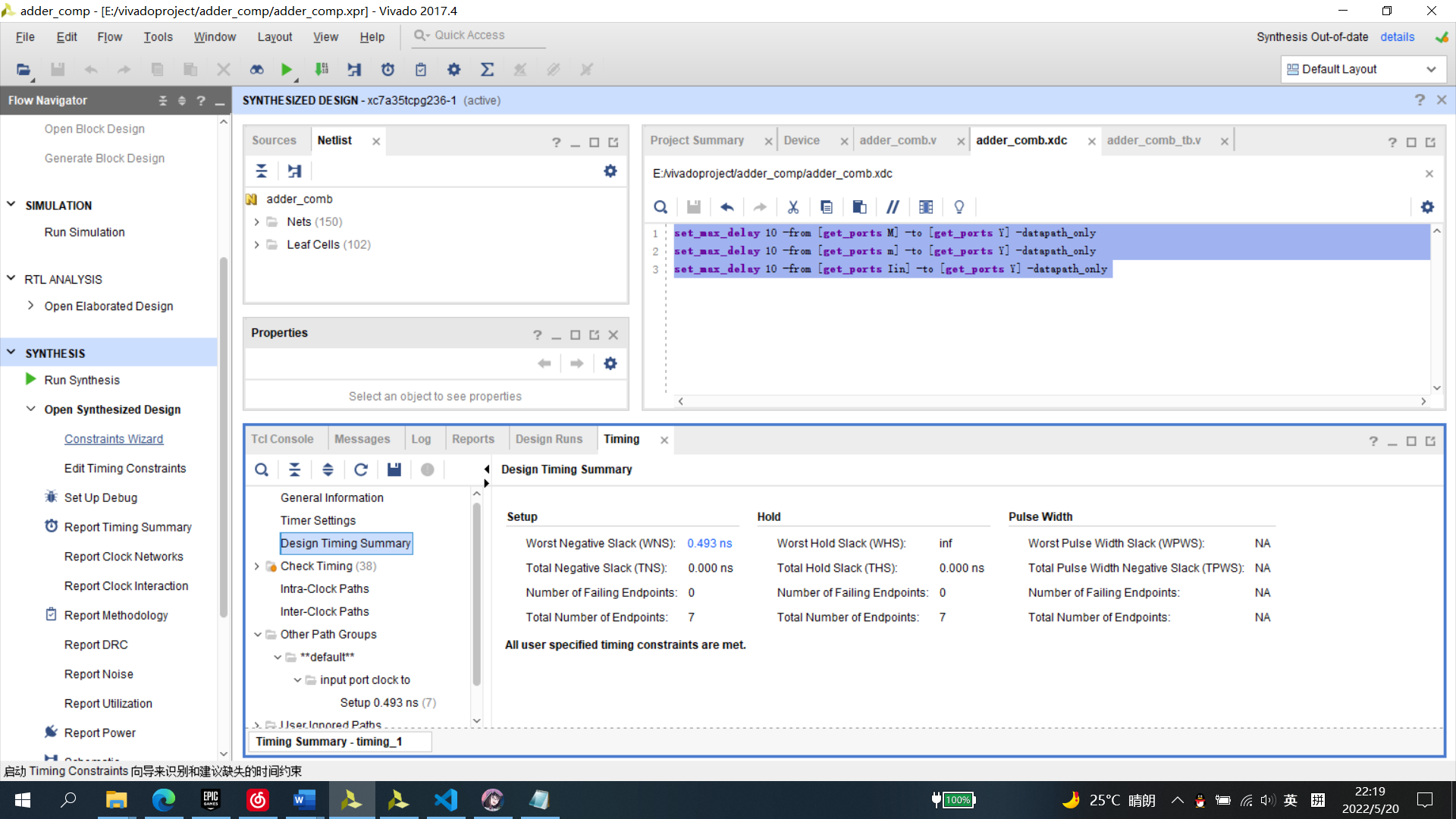
**对两种方式设计加法器的STA**

1. **组合逻辑**

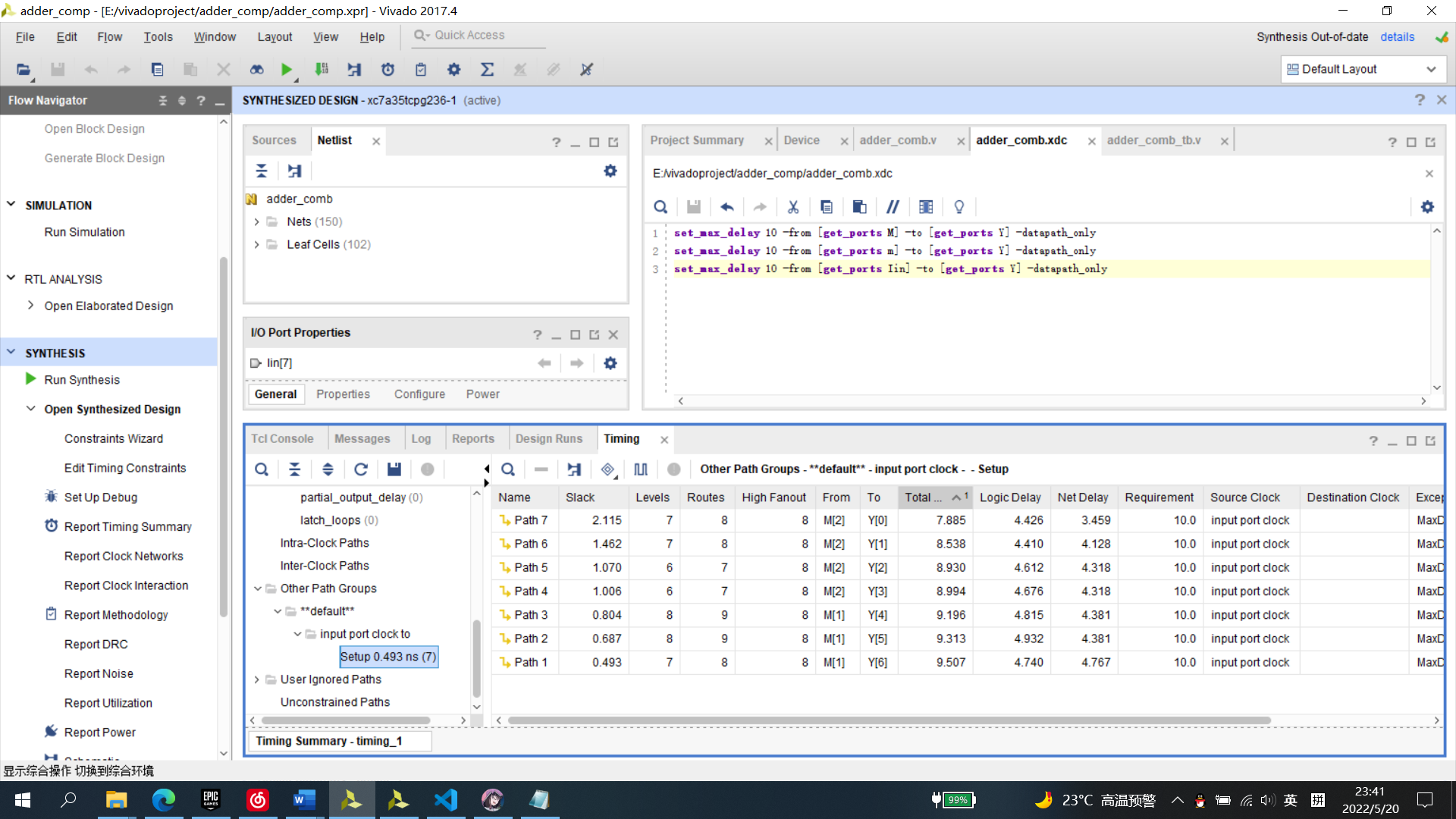
**资源占用**

逻辑资源：





设置M，m和Iin到Y的max\_delay为10ns后的时序性能



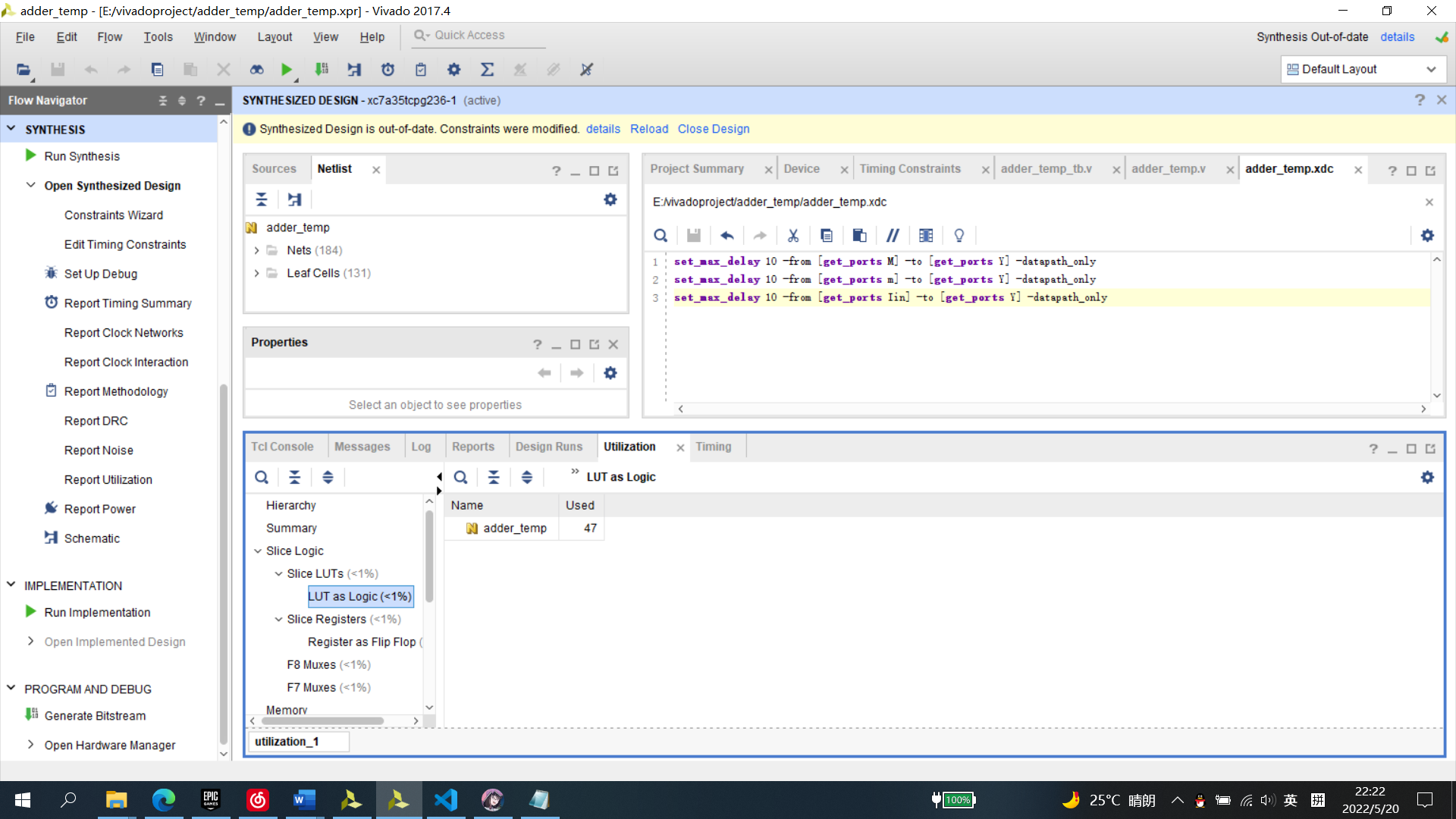
单次计算最低延时

最大，最大，，由建立时间约束，最大时间频率

1. **时序逻辑**

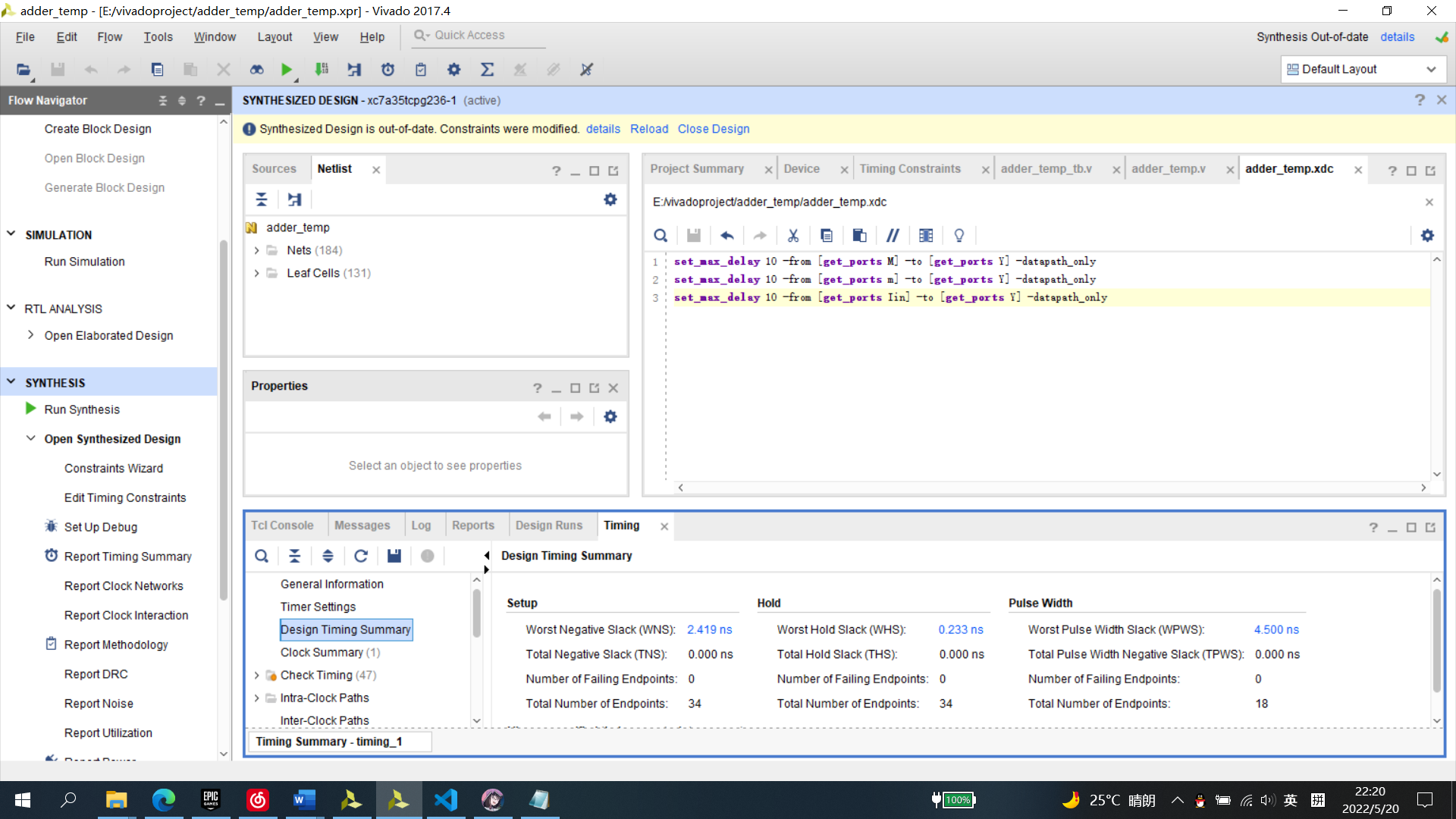
**资源占用**

逻辑资源：

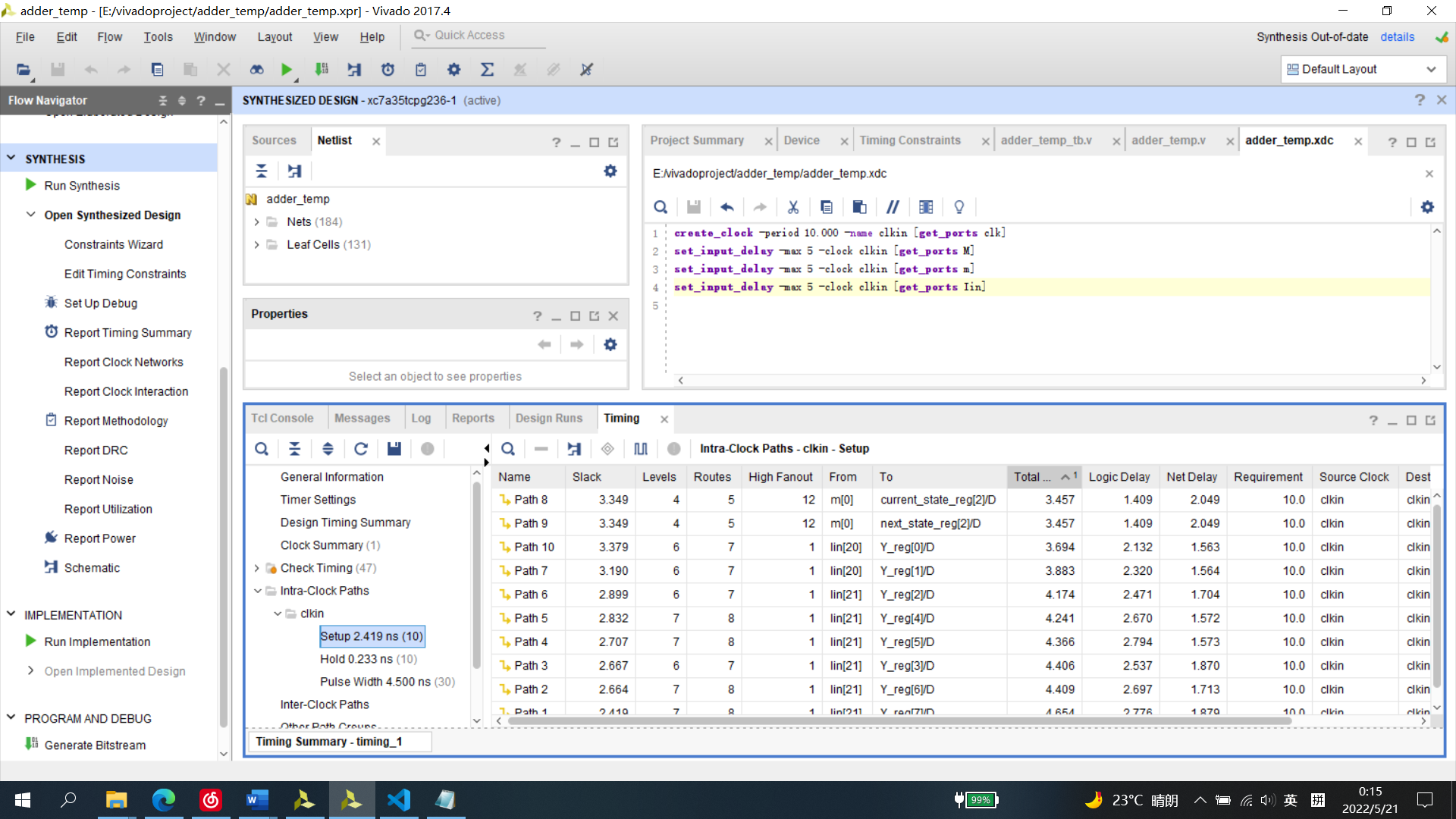




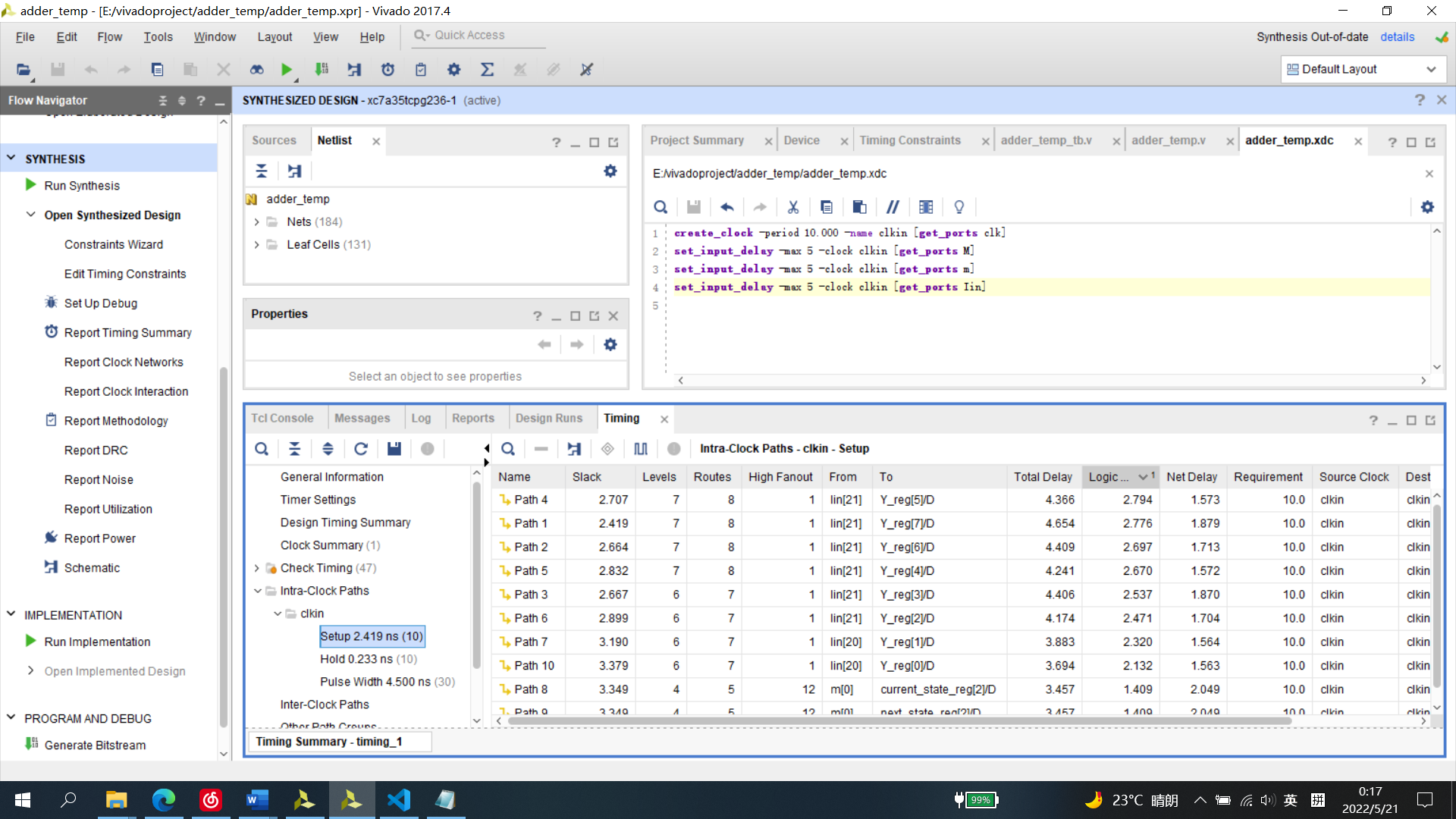
寄存器使用：



设置M，m和Iin到Y的max\_delay为5ns后的时序性能



单次计算最低延时，最高



最大，，由建立时间约束，最大时间频率

综上，组合逻辑设计使用的LUT更少，但是wns和最高时钟频率方面较差；相对的，时序逻辑用的LUT更多但是wns和最高时钟频率方面较组合逻辑设计优秀。

**实验内容三**

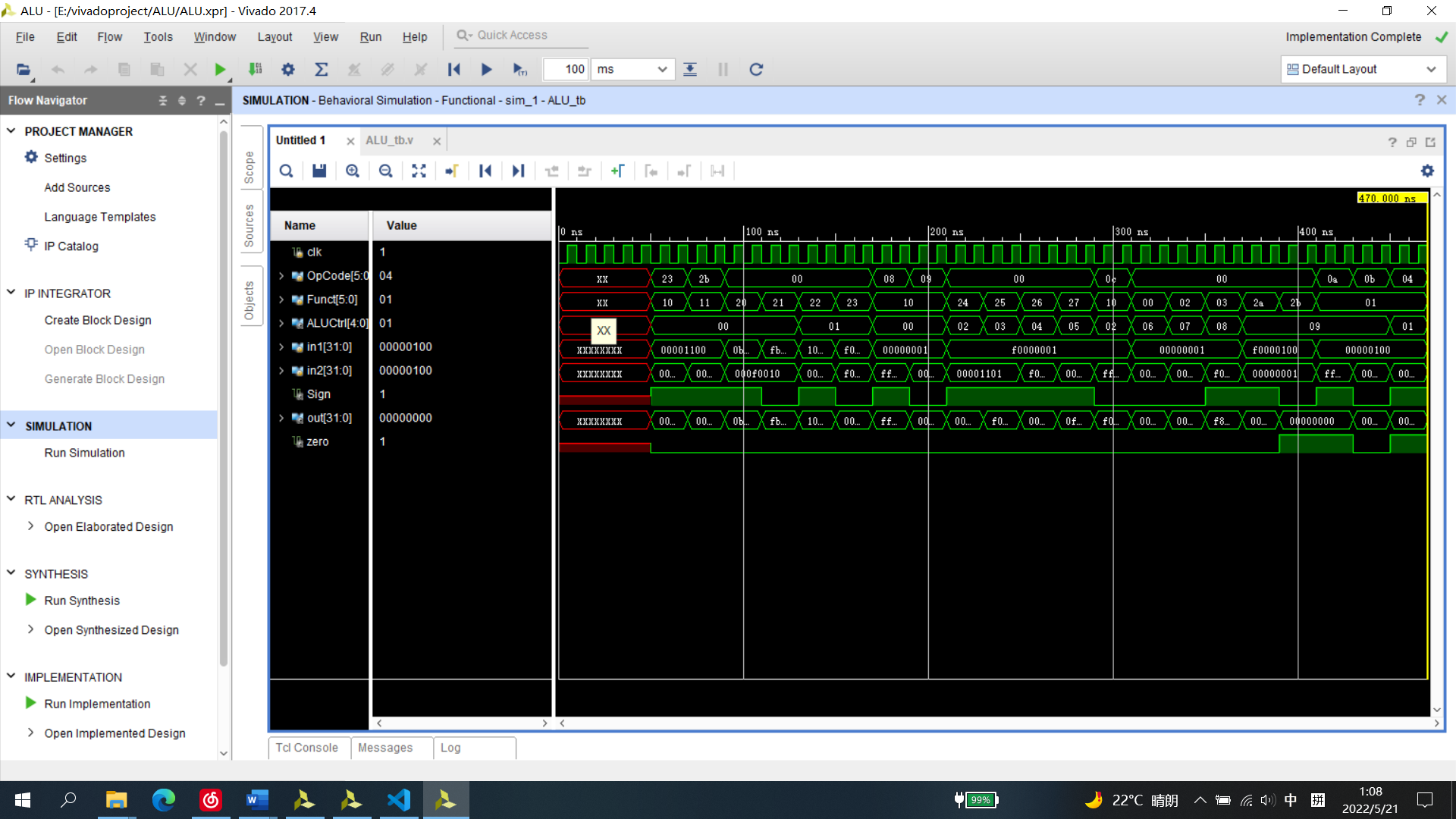
**设计思路**

对经过ALU的指令进行了分类，即add、addu、sub、subu、and、er、xor、noe、sll、srl、sra、slt、sltu、lw、sw、addi、addiu、andislti、sltiu和beq。以及这些指令对ALU的功能需求，包括加减法（包括有无符号运算，虽然不考虑溢出的情况有无符号数的加减法都是一样的，区别主要在i形指令需要的立即数扩展上，但为了testbench中区分有无符号数结果在输出时区分了有符号和无符号的情况）；按位与；按位或；按位异或；按位或非；左移；逻辑右移；算数右移和slt系列的四个语句，根据输入的Sign信号0还是1决定有还是无符号比较。ALU功能的选择依赖于上一级ALUController输出的控制信号ALUCtrl，决定ALU功能。ALUCtrl由输入的Opcode和Funct决定，根据指令类型决定控制信号赋值。

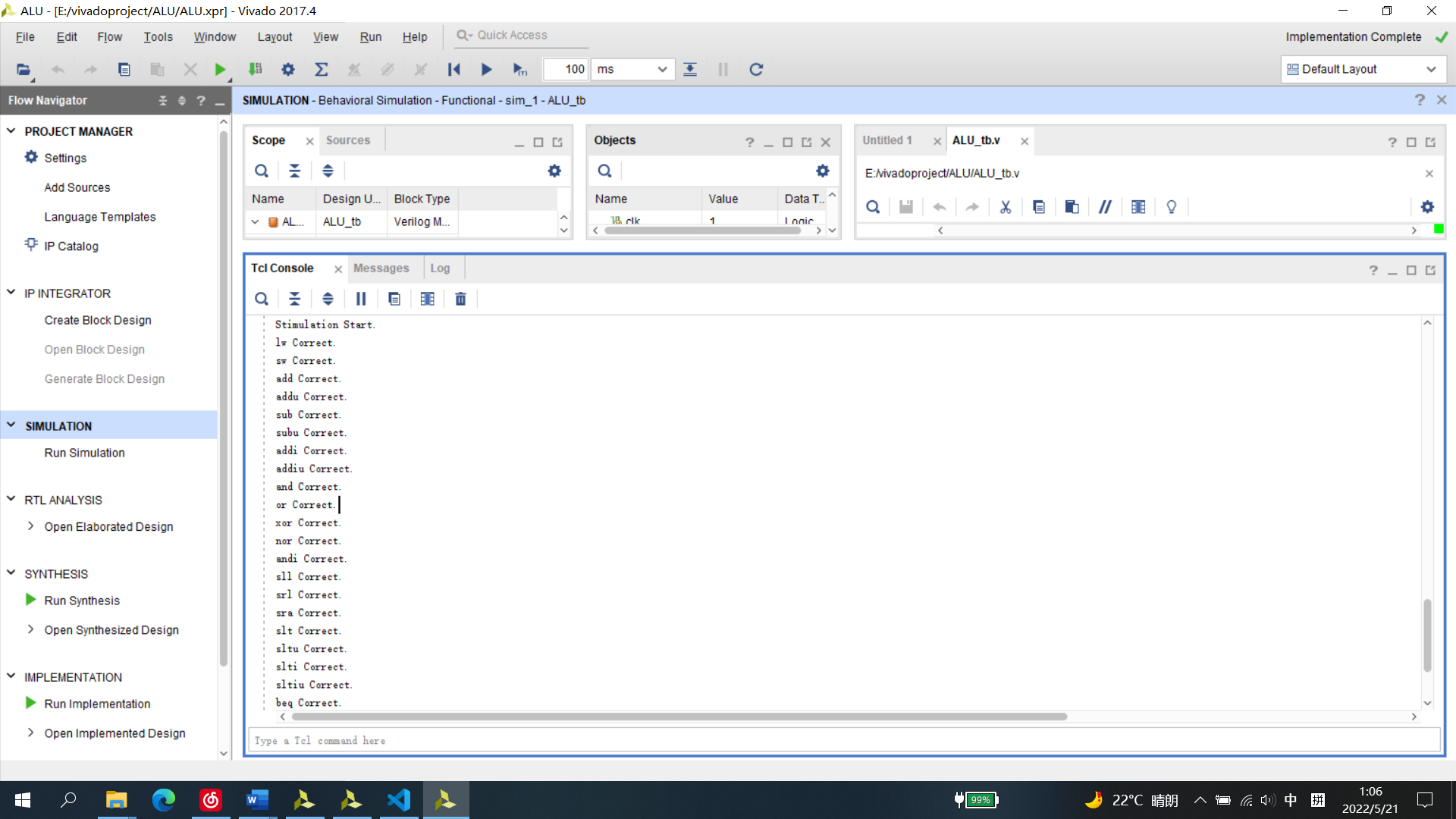
**代码逻辑**

ALUController输入6bit Opcode和6bit Funct，输出5bit ALUCtrl和1bit Sign。用case语句给ALUCtrl和Sign赋值。Opcode等于零说明是R型指令，具体功能要看Funct的值，因此再加一个case语句选择Funct。其他指令可以根据Opcode确定。进入下一级ALU根据ALUCtrl选择对应的功能，根据Sign选择有无符号运算，用assign语句持续赋值zero，输出out=0就赋给zero 1‘b1，否则赋1’b0。

**验证结果**



仿真波形图



仿真结果，对设计的各条指令能正确输出