

（深圳）

实验报告

开课学期： 2020秋季

课程名称：数字逻辑设计（实验）

实验名称： 记忆游戏

实验性质： 综合设计型

实验学时： 6 地点： T2614

学生班级： 计科5班

学生学号： 190110509

学生姓名： 王铭

评阅教师：

报告成绩：

实验与创新实践教育中心制

2020年12月

|  |
| --- |
| 设计的功能描述 |
| 基础功能：当玩家按下S0键后，能够自动产生五个随机数并显示到数码管上，按下S1后，玩家选择想要匹配的随机数地址，按下S2确定选择。后用户需要输入五次一个八进制数，每次按下S3确认输入，当输入够五次后，若用户输入的数字与选择的随机数相同，则数码管显示对应的“随机数地址-随机数”，否则，数码管显示浮动的零，表示匹配失败。若用户想要重新开始游戏，需按下S4复位键，则游戏重新恢复初始状态，等待用户按下S0开始产生新一轮的随机数。 |
| 系统功能详细设计 |
| 主要设计分块：  分频器，随机数，消抖，存储，选择，匹配，数据选择，输出八个模块。  主要功能：  分频功能：divider，divider1，divider2，divider3  消抖：fangdou  随机数：Random  存储：ram  选择：MUX  数据选择：readS1、readSelect  匹配：readS2  输出：convert和Print  相互关系图：    状态描述：  本设计未采用状态机表示按键的状态，选择用select、res、flag、i等标记标明程序执行到的步骤。 |
| 模块描述 |
| 主模块：    MemoryGame模块为文件顶层模块，输入5个按键信号s0~s5，时钟信号clk，以及拨码sw，输出控制信号DK，左端7段数码管和右端7段数码管的信号编码led\_lo和led\_ro。  主要设计：  MemoryGame模块通过调用消抖模块，将按键信号过滤。S0按下时调用随机数模块，生成5个随机数，同时调用Ram模块，存储生成的随机数。S1或S2按下时，调用readS1完成用户的输入选择功能，同时对Ram中存储的随机数进行读取。按下S3时，由ReadS2模块进行匹配，输出匹配的结果。每次输出时用control模块控制打印（DK、led\_lo、led\_ro变化）的频率，并用MUX模块选择当前游戏进行到的模式，选择出应该显示的内容。最后将相关内容通过Print模块显示在7段数码管中，从而完成该游戏的设计。  该模块的关键变量有：  rand:当前随机数生成器正在生成的随机数  rand1~rand5:生成的5个随机数  set\_0~set\_5:消抖后的s0~s5  i:控制生成随机数的变量，i为1~5代表生成第1~5个随机数  select:用户按下S1、S2后的选择，由readS1决定  select\_num:用户选择的随机数  res:用户匹配之后的结果，1表示失败，2表示成功，由readS2选择  buffer:要输出到数码管上显示的内容，由MUX选择  注：上述变量i、res、select均从1开始表示有效，0为初始值，程序中不会用到。  Random模块：    输入变量有时钟信号clk，顶层模块的i，输出为15位的随机数。  该模块的设计思路：  采用给移位寄存器增加反馈，完成状态转移，同时取状态机的最后一位作为随机数的第i位输出。  关键变量：  i:控制状态机的最后一位赋值给生成随机数的第几位，初值为0，在生成随机数阶段，随时钟的上升沿每次加一，直到f。  flag:1表示随机数模块正常工作  x:存储上一个状态的输入变量t  输入的t变量，由于t（即顶层文件的i）为0表示初始状态，故该模块判断随机数是否开始生成的标准为，x< t。当用户在未生成5个随机数时再次按下S0时，x会大于t，此时将i变成0，表示重新生成。  now\_state:状态机的现态  next\_state:状态机的次态  状态机转换的逻辑关系如下：  next\_state[7] <= now\_state[7]^now\_state[5]^now\_state[4]^now\_state[3];  next\_state[6] <= now\_state[5]^now\_state[4];  next\_state[5] <= now\_state[4]^now\_state[3];  next\_state[4] <= now\_state[3]^now\_state[2];  next\_state[3] <= now\_state[2]^now\_state[1];  next\_state[2] <= now\_state[3];  next\_state[1] <= now\_state[2];  next\_state[0] <= now\_state[1];  divider：    输入要分频的时钟信号clk\_i,输出分频为1ms的时钟信号clk\_o，采用计数器分频，较为简单  divider1:    输入要分频的时钟信号clk\_i,输出分频为1s的时钟信号clk\_o，采用计数器分频，用于控制1s生成1个随机数（clk\_o每次上升沿时，顶层模块的i加1）  divider2:    输入要分频的时钟信号clk\_i,输出分频为1ms的时钟信号clk\_o，采用计数器分频，用于利用人眼的特性，在短暂时间内人眼分辨不出来数码管显示时发生的快速变化，可以用来控制DK的输出。  divider3:    输入要分频的时钟信号clk\_i,输出分频为0.25s的时钟信号clk\_o，采用计数器分频，主要用于在玩家匹配失败时，输出浮动的0。  control:    输入信号：  clk\_dk1为1ms的时钟信号  clk\_dk2为0.25s的时钟信号  res为匹配的结果  set\_4为消抖过滤后的复位键。  输出cnt为控制DK变化的计数器  主要设计：  根据res判断cnt变化的频率，当res不是1（即匹配未失败时），控制cnt变化的应该为clk\_dk1,否则为clk\_dk2  MUX：    输入信号：  cs1为顶层文件中的flag，表示生成随机数的信号  i为生成的随机数序号  rand为正在生成的随机数  select为用户的选择  store为存储第五个随机数的7段数码管表示  res为匹配的结果  输出信号：  buffer为8个数码管该输出的内容  主要设计：  分3个大的情况，①res为0，此时根据select是否为0判断是否处于随机数的生成阶段，输出相应的随机数编码，否则输出选择。②res为1，匹配失败，输出8个f的编码。③res为2，匹配成功，输出“地址-随机数”的编码。  Print：    输入信号：  cnt为控制显示的频率的计数器  buffer[63:0]对应8个数码管的输出  输出信号：  DK为选择信号，led\_lo为左四个的输出，led\_ro为右四个的输出  主要设计：  根据cnt的值，DK的值和led\_lo、led\_ro的值。  Ram：    输入信号:  en\_i为写使能信号，为顶层模块的flag（当处于生成随机数阶段时为1）  select为用户的选择，根据select进行读随机数  i为当前生成的随机数次序  rand为当前生成的随机数  输出信号：  rand1~rand5为生成的5个随机数  select\_num为用户选择的随机数  store用于向数码管打印第5个随机数  主要设计：  当使能端有效时，ram进行写功能，根据i的值，将当前生成的随机数赋值给randi。当i为5时，由于数码管要保留显示第5个随机数，故用store保存这5位随机数转换为7段数码管输出时的值。  否则，判断select的值，确定用户选择的数字，赋值给select\_num  fangdou:    输入信号：  clk为时钟信号，s0为要消抖的信号  输出信号：  s1为消抖后的信号  主要设计：  用一个1ms的时钟信号，在检测到s0变化后的第5ms对信号采样，若信号仍发生了变化，则此次按键有效，输出s1，否则认为此次按键无效。  convert:    输入信号:  en\_i为使能信号  num为要转换成7段数码管输出的数字  输出信号：  输出该数字转换为7段数码管输出后的结果。  主要设计：  主要用“查表法”，判断输入的数字，选择相应输出，较简单。  readS1:    输入信号：  s1,s2,s4为消抖后的按键，sw为选择时的输入  输出信号：  select为做出的选择  关键变量：  flag：为0表示未按下，为1表示s1按下，为2表示s2按下  主要设计：  根据按键情况设置flag的值，后调用readSelect模块，选择select的输出值。  readS2:    输入信号：  s4、s3为消抖后的按键s4、s3,select\_num为选择的数字，sw为每次输入的一位8进制数。  输出信号：  匹配的结果，成功为2，失败为1，否则为0  主要设计：  用s3为时钟，控制一个计数器，5次输入后，进行匹配，返回匹配结果res  readSelect:    输入信号：  sw为输入的选择，flag为记录按键的标记  输出信号：  select为做出的选择。  主要设计：  根据flag的值判断s2是否按下，按下后select不变，否则判断s1是否按下，按下后select若小于4则输出选择，否则输出6表示此时应该输出8个f，若未按下s1、s2，flag为0则select为0，否则select不变。 |
| 管脚分配表 |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | clk | s0 | s1 | s2 | s3 | s4 | | P17 | R11 | R17 | R15 | V1 | U4 | | sw[0] | sw[1] | sw[2] | sw[3] | sw[4] | sw[5] | | R1 | N4 | M4 | R2 | P2 | P3 | | sw[6] | sw[7] | DK[0] | DK[1] | DK[2] | DK[3] | | P4 | P5 | G6 | E1 | F1 | G1 | | DK[4] | DK[5] | DK[6] | DK[7] | led\_lo[0] | led\_lo[1] | | H1 | C1 | C2 | G2 | B4 | A4 | | led\_lo[2] | led\_lo[3] | led\_lo[4] | led\_lo[5] | led\_lo[6] | led\_lo[7] | | A3 | B1 | A1 | B3 | B2 | D5 | | led\_ro[0] | led\_ro[1] | led\_ro[2] | led\_ro[3] | led\_ro[4] | led\_ro[5] | | D4 | E3 | D3 | F4 | F3 | E2 | | led\_ro[6] | led\_ro[7] |  |  |  |  | | D2 | H2 |  |  |  |  | |
| 调试报告 |
| 仿真代码：  `timescale 1ms / 1ns  module MemoryGame\_sim();  reg clk = 0;  reg set\_0 = 0;  reg set\_1 = 0;  reg set\_2 = 0;  reg set\_3 = 0;  reg set\_4 = 0;  reg [7:0]sw;  wire [7:0]DK;  wire [7:0]led\_lo;  wire [7:0]led\_ro;  MemoryGame game(clk,set\_0,set\_1,set\_2,set\_3,set\_4,sw[7:0],DK[7:0],led\_lo[7:0],led\_ro[7:0]);  always #0.000005 clk = ~clk;  initial  begin  #0 set\_0 = 0;  #6 set\_0 = 1;  #6 set\_0 = 0;//模拟按下s0  #50 sw[7:0] = 8'b0000\_0001;//生成随机数并输入选择  #5000 set\_1 = 1;  #6 set\_1 = 0;//按下set1  #6 set\_2 = 1;  #6 set\_2 = 0;//按下S2确定选择  #6 sw[2:0] = 3'b011;//输入5次8进制数，匹配不成功  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b100;  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b101;  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b110;  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b010;  #6 set\_3 = 1;  #6 set\_3 = 0;  #20 set\_4 = 1;  #6 set\_4 = 0;  #10 set\_0 = 0;  #6 set\_0 = 1;  #6 set\_0 = 0;//按下s0  #50 sw[7:0] = 8'b0000\_0011;//生成随机数并输入选择  #5000 set\_1 = 1;  #6 set\_1 = 0;//按下set1  #6 set\_2 = 1;  #6 set\_2 = 0;//按下S2确定选择  #6 sw[2:0] = 3'b001;//输入5次8进制数，匹配成功  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b110;  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b100;  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b111;  #6 set\_3 = 1;  #6 set\_3 = 0;  #6 sw[2:0] = 3'b010;  #6 set\_3 = 1;  #6 set\_3 = 0;  end  endmodule  仿真分析：    图1    图2  如图1所示，12ms时，模拟S0按下后，随机数模块开始生成随机数。随机数的生成选择使用状态机生成，按照表达式给出的逻辑，控制现态转换成次态的方式，每次时钟的上升沿会导致状态机的一次变化。同个上升沿时，取状态机现态的末位赋值给随机数的一位，当开始生成一个随机数时，取状态机的状态的前15次变化（用i反映），当i处于[0,14]区间时，把状态机的现态末位赋给随机数的第i位，当i取15（f）时，不再赋值。图2中，now\_state和next\_state分别表示现态和次态，每个时钟上升沿时，状态发生状态转移，其中i用来控制赋值，结果正确。    图3  如图3所示，在生成随机数的同时，将此时生成的随机数转换成7段数码管的表示形式，并打印到数码管上，其中DK[7:0]控制选择数码管。led\_lo和led\_ro显示7段数码管输出的内容，其中根据设计，DK为80,40,20,10时，led\_lo显示生成随机数的前4位，DK为08时，led\_ro显示生成随机数的第5位，而数码管的后3位不亮，即DK为04,02,01时，led\_ro为0仿真结果均正确。    图4  如图4所示，在1s左右，当下一个随机数要生成时，i会自动变0。    图5  DK及led\_lo,led\_ro的控制信号由cnt的值控制，即DK变化的cnt的频率相同，如图5所示。  重复上述过程，直至5s后生成5个随机数。    图6  如图6所示，生成随机数时，由顶层文件的i信号和flag（写使能信号）控制Ram模块将随机数写入rand1~rand5。    图7  如图7所示，在生成5个随机数后，模拟S1按键，当按下S1后（5068ms），数码管显示SW[7:0]的选择,select[3:0]表示选择，仿真的led\_lo,led\_ro显示均正确。    图8  如图8所示，按下S2后确认选择，此时select和数码管的显示均保持不变，仿真正确。    图9  如图9所示，按下S2后，根据select（即读信号）的值，在ram中读取选择的随机数到select\_num中，select为1，则选择的是第二位随机数，故select\_num取rand2（1d3a）以供匹配。    图10  接着，如图10所示，模拟按下5次S3，每次由SW[2:0]输入一位八进制数，与上一步选择的随机数进行匹配。    图11  如图11所示，在第一次匹配时，由于输入的5个八进制数组成的5位数与选择的随机数不同，res变为1，表示匹配失败。此时，cnt以及DK、led\_lo,led\_ro的显示频率变慢，显示浮动的0，结果正确。    图12  如图12，按下复位键S4（5190ms），开始第二次游戏，顶层文件的i置0，select置0    图13  如图13所示，5s后生成新一轮随机数。    图14  如图14所示，选择第四个随机数，仿真模拟按下S1，S2，select变为4，根据select从Ram中读取第4个随机数赋给select\_num(1d3a)，仿真正确。    图15  如图15所示，同样地，输入5位八进制数，与选择的随机数进行匹配，此次匹配成功，输出res为2。    图16  如图16所示，此时数码管显示为“选择-随机数”，仿真结果正确。 |
| 设计过程中遇到的问题及解决方法 |
| 仿真中遇到的问题：  未考虑消抖功能的影响，模拟按键的时间太短，导致按键被消抖，即模拟按键失败，随机数无法生成。  解决方法：  消抖模块采用延时消抖的方法，其中取样的时间为5ms，则在  仿真文件中模拟按键按下时，信号按下的时间置为6ms  错误的仿真截图：    仿真较为顺利，除出现此错误外并未出现其他问题。 |
| 课程设计总结 |
| 包括设计的总结和还需改进的内容以及收获  设计总结：  本次设计采用的思想偏编程化，在完成设计时选用了标志表示状态，初版设计时仿真、上板均较为顺利，初版的耦合度较高，故我选择优化程序结构，增加多个模块使该设计更偏模块化。  还需改进的内容：  由于设了许多标志位，使得代码看起来更为冗杂，可以考虑用状态机表示按键的各个状态。同时，在分频模块时，并未集成一个模块，在需要不同频率的时钟信号时，选择用了多个模块，可以改进。在随机数的生成模块中，我选择用带反馈的移位状态机生成随机数，这种随机数的伪随机的程度太高，可以进一步改进。最后，模块的命名规范上仍待改进，如readS1可以改成Select，readS2可改为Match。  收获：  本门实验课程对我的理论学习有很大的帮助，通过实验上班，将较为抽象的理论知识变得具体化，使我掌握的更牢固。同时，提高了自己的解决难题的能力，锻炼了自己的编程与硬件相结合的思维模式。 |