P1课下作业个人小结

1. Verilog题目注意事项。
   1. 审题贯穿整个做题的过程，理解题-分析题-调试题，一般出错了绝大部分是审题出现了偏差。
   2. 低级错误：模块名错误，端口名错误，未初始化错误，阻塞与非阻塞赋值错误。
   3. 系统调用的使用：使用display和monitor，可以分辨某个块是否被进入过。
   4. 重视题目所给的具体案例和时序图案例：将自己设计的方法带进去看能否生效。
2. Verilog书写总线时的注意事项。
   1. [m:n]的位置问题：定义时在变量名前面；部分选择时（全选可不写）在变量名后面。
   2. 声明与定义阶段：[size-1:0] or [size:1]（行为规范，如果反着过来也可以，但是外部接口在调用时便发生了错位）。
   3. 调用阶段：必须遵循声明定义时的形式，左右两边数的大小关系需要和最初的一致。
   4. 赋值阶段：书写要求同调用，同时在“默认”写法下赋值，线上的内容和书写时顺序是一致的（计算机不考虑高低位的问题），index高的存高位，index低的存低位，a[7:0] = 8’b1000\_0000. -> a[7] = 1;
   5. ？逐位颠倒如何处理：目前来看只能将其视为“总线选择和拼接总线的问题”，因为在声明a[7:0],b[7:0]情况下不可用b[0:7] = a[7:0]。

reg [7:0] a;

reg [7:0] b;

reg [0:7] c;

a[7:5] = b[5:3]; // Correct

a[7:5] = b[3:5]; // Wrong

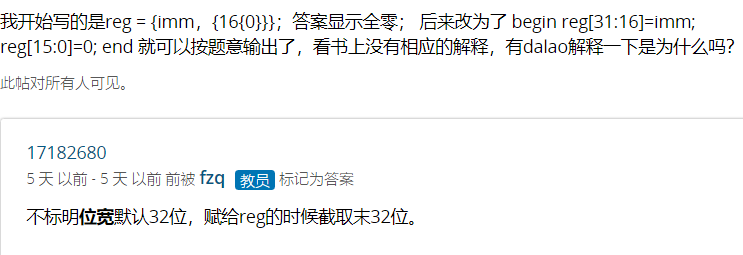
a[7:5] = c[0:2]; // Correct

a[7:0] = b[0:7]; // Wrong

a[7:0] = {b[7:4],b[3],b[2],b[1],b[0]}; // Correct

a[7:0] = {b[7],b[6],b[5],b[4],b[3],b[2],b[1],b[0]}; // Correct

1. 数制表达时的注意事项（条件语句，{}拼接符）
   1. 可移植性模块使用parameter确定位数数量。
   2. 条件语句（case，if-else等）写明比较的数的位数和进制。
   3. Verilog通用：未声明进制-默认十进制，未声明位数-默认32位。
   4. {}拼接时务必注意每一元素的大小必须给出
      1. 被赋值对象位数不足：从低位截取拼接结果赋值。
      2. 被赋值对象位数多余：赋值给低位部分。



1. 符号位扩展方法总结与$signed使用方法。
   1. 符号位扩展方法1：判断最高位，而后使用拼接。
   2. 符号位扩展方法2：将16位直接赋予32位数，默认无符号扩展，$signed(x)会进行符号扩展。
   3. Signed目的是告诉系统这是一个算术数，并且比较类运算符需要两者都为算术数才会执行算术比较（其中一个没有声明便是无符号比较）。
   4. 加减法运算遵循补码形式的运算规则，只对这个序列操作，并不管实际是否有符号。

    reg signed [7:0] a;

    reg [7:0] b;

a = 8'b11111111;

    b = 8'b00000001;

    c = (b < a); // c = 1;

reg signed [7:0] a;

    reg signed [7:0] b;

a = 8'b11111111;

    b = 8'b00000001;

c = (b < a); // c = 0;

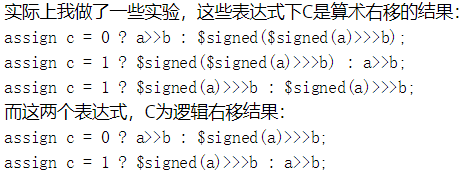
reg [7:0] a;

    reg signed [7:0] b;

a = -2;

b = 8'b00000001;

    c = (b < a); // c = 1, a=-2系统会按照补码的形式赋予a，但是a为非符号数的默认性质并没有改变，因此比较的时候还是会按照非符号数来。

1. 三目运算符使用的坑与复合使用方法。
   1. 三目运算符的规则探究还不清晰：目前的理论——两个运算都需要为算术类型，程序才会执行算术类型处理，另外整个表达式若被强制转化为算术类型，也输出算术类型。
   2. 
   3. 复合使用方法：h? choice\_a : g? choice\_b : f? choice\_C : choice\_D.
   4. = h? choice\_a : (g?( choice\_b (f? choice\_C : choice\_D))).
2. Gray编码的编码方法与技巧。
   1. Gray编码方法：
      1. t xor t(左移一位，去掉最高位),
      2. \*n->n+1位归纳法：(0~2^(n-1)-1)前n-1位与n-1时相同，(2^(n-1)~2^n-1)第n位为1，其余的为(0~2^(n-1)-1)倒序填充。
   2. 技巧：少量编码情况可采用parameter定值法。
3. 混合时序与组合逻辑的赋值方法。

混合时序逻辑与组合逻辑时请务必全部使用“非阻塞赋值”（杜绝需要及时变化的中间变量，直接操作最终变量）。实际操作时，若组合逻辑涉及到输入输出的边缘情况，可将两部分时序和组合分拆执行

（时序逻辑是简要的核心，仅需要保留最精髓的，形象公关交给组合逻辑）。

1. Verilog的层次化事件队列（从上到下优先级递减）。
   1. 动态事件队列（其中顺序若为顺序块则顺序执行，否则随意安排）（真正执行队列）
      1. 阻塞赋值
      2. 计算非阻塞赋值语句右边表达式
      3. 连续赋值
      4. 执行display命令
      5. 源于的输入和输出的变化
   2. 停止运行的时间队列（#0）（不推荐使用）
   3. 非阻塞事件队列：更新非阻塞赋值语句LHS（左边变量）的值。
   4. 监控事件队列（monitor，strobe）
2. $monitor, $strobe, $display三种方法检测赋值的问题。
   1. Display优先于非赋值语句执行，不可用于always模块中检测非赋值语句，strobe、monitor可以。
   2. $display($time, , “the value of x is %b”, x); // 一般是时间+字符串表示（逗号间隔的内容可以有多个东西。）
   3. $monitor("value of a and b is %b %b",a, clk);// 检测了a和clk信号的变化。
3. 寄存器初始化问题。
   1. ISE初始定义的寄存器值呈现高阻值Z（？），请务必在initial块中对其初始化。
4. 两个Always块赋同样的值，出现的问题。
   1. 两个不同的always块对同一个变量赋值，在仿真时不会出现报错，但是若同时触发赋值可能发生难以发现的错误；而如果进行综合，则会提示报错，因为同一个输入端接入了多个端口。
5. Verilog-iSim仿真使用方法。（Relunch， 查看中间变量值）。
   1. 在iSim中可以直接点集查看uut和testbench代码，可以直接进行修改并进行relunch仿真。
   2. 点击uut等对应的子模块，可将模块中定义的中间变量进行监测。
6. Verilog-testbench调试程序的书写方法。
   1. 多而简单的功能：在一个代码中分别实现即可。
   2. 时序变化功能：
      1. 在稿纸上画好clk信号，分割出参考区间。（不画高低情况，写数字即可）
      2. 按照模拟好的数据将输入信号变化写在对应的区格中。
      3. 用testbench代码实现。
      4. ！正常输入信号尽量避免在跳变区测试；复位、置位信号一定要在跳变区尝试。
7. 异步复位在同一个always块中的实现方法。

