

（深圳）

实验报告

开课学期： 2021秋季

课程名称：数字逻辑设计（实验）

实验名称： 十六进制计算器设计

实验性质： 综合设计型

实验学时： 6 地点： T2506

学生班级： 计算机类4班

学生学号： 200110428

学生姓名： 杨杰睿

评阅教师： 郑海刚

报告成绩：

实验与创新实践教育中心制

2021年12月

|  |
| --- |
| 设计的功能描述 |
| **基本功能**  实现十六进制计算器基本功能，包含加、减、乘、除、取模、平方。输入数值num1和num2由[15:8], [7:0]位拨码开关给出，运算的种类由[23:21]位拨码开关给定。  **扩展功能**  按键消抖模块key\_filter.v，该模块功能为实现按键消抖，防止按键时持续的周期造成运算不受控制的进行。设计逻辑经两轮迭代，初版消抖模块基本实现消抖功能，但设计粗糙，对于长按没有进行处理；终版提交的消抖模块，正确完整的实现了消抖功能，避免了按键时信号的毛刺对结果的影响，也避免了按键持续时非期望连续计算的进行。  该按键消抖模块具有通用性，可方便的复用到其他应用到的模块中。 |
| 系统功能详细设计 |
| **硬件框图** |
| 各模块描述 |
|  |
| 设计过程中遇到的问题及解决方法 |
| **遇到的问题及解决方案**  事实上，本实验难度适中，仅代码量相比此前稍大（总计约260行），需要提前梳理好电路运行的逻辑，按部就班写完代码仿真上板即可完成实验。  在理解题意的方面上，出现了一点小的偏差，导致1-2h在对一小部分testbench的异常进行排错。事实上，所给定的testbench只对前6次进行了测试，之后是没有测试的，如果强行运行，会出现波形异常的情况，虽然此前尝试利用同步时序变化滞后一个周期的特点刻意制造延迟等手段来满足异常处的波形的计算，但后了解到是没有必要的部分，最终删去相关代码。  在上板时，比较典型的问题就是按下button的瞬间，会导致计算器进行上千万次的计算，而这显然不是我们所期望的，所以消抖模块对于上板正确是必要的，而对于仿真过程中，由于仿真设置的button信号没有异常抖动，仅需要调整消抖模块的CNT\_MAX即可正常运行仿真。  起初的消抖模块未能够处理在持续按键时不进行多次运算的问题，这一点在最终提交的代码上已经得到修改。当按下button时进行运算，持续按键不改变运算结果，只有释放之后再次按下才会使用先前的计算结果进行连续计算。消抖模块的实现逻辑，基本思路是限定两次button按下有设定的最小时间间隔并且持续按下时计数器值不发生改变或为0，这里我设置的最小时间间隔是1E-4s，也就是1000个分频时钟（10MHz）周期，经测试可以达到所期望的执行效果。 |
| 课程设计总结 |
| **总结**  实验6十六进制计算器设计，可以视为是此前多次实验的复合产物，综合的锻炼并复习了此前实验的功能实现。  在实验1中，我们学习使用了3-8译码器的verilog代码实现，在那之后，实验4数码管控制器设计实验中同样的用到了译码器代码，并且此次实验6，归功于此前实验4代码风格良好，直接复用了近50行数码管扫描代码，大大的提高了代码编写效率。  本次实验中要求使用时钟IP，可以推测目的在于对使用IP核进行verilog编程的再一次熟悉。除此以外，大量使用的计数器用于时间控制，也是实验2熟悉的结果。  **收获与待改进的方面**  代码风格相比于最初几次的实验有了较大的改进。从第2次实验开始，因将所有的逻辑语句放在同一个always块中，当代码出错后难以排查，甚至逻辑无异常但是确实无法正确运行。那之后，在学长和同学以及老师的提醒下，养成了对于同一always块尽可能的只对一个变量进行赋值的习惯，偶尔的特例也是在严格思考过代码运行逻辑的情况下，以提高代码简洁程度来进行的少量改动（如几个always块有完全相同的判断条件）。  学会阅读了波形并调出自己模块的波形。因testbench并非自己编写，对于其代码逻辑的不熟悉往往不方便对代码进行调试，在学会将自己模块的信号波形调出进行观察以前，通常使用上板的方法进行调试，这样效率极其低下，浪费大量时间在等待比特流生成的过程中。  学会了报错信息阅读。仿真报错信息路径已经较为熟悉，elaborate文件xvlog日志信息等已经深入脑海。对于日志的阅读能高效的帮助自己排除错误。  复用xdc文件。这一点是在实验3之后才意识到的，对于同样的开发板，尽管使用不同的电路元件进行编程，在信号命名习惯相同的情况下，通用的xdc文件是可以被总结出的，为此，经过前面几次的总结梳理得出了目前为止的通用Exp\_General.xdc文件来提高约束文件编写效率。  使用generate来简化代码编写。Generate不同于for语句的作用是，generate语句相当于是将代码段展开硬嵌到最终的代码中，实际上并不会执行循环语句，所以，既起到了免于对重复部分编写代码的麻烦和单调工作，也避免了不可综合的风险。而对于verilog中一些小技巧的使用，也让我对向量的控制更加灵活，如assign result[i] = cal\_result[i\*4 +: 4]; (出处为generate语句块中, genvar i;)，来控制特定连续4位的向量块赋值。  但是一些使用习惯有待改进。对于毛刺信号，据了解，利用任何计数器进行的时间控制都会有毛刺的产生，导致时间控制不完全准确，更建议的做法是只使用时钟IP而不做计数器，但由于使用计数器进行时钟控制几乎已经成为代码习惯的一部分，而当前的任务并没有如此高的精度要求，所以暂时搁置这一点不好的习惯。对于使用rst\_n而不是rst是被建议的做法，理由是对生成实际电路更友好，这一点由于在写最后一次实验代码的中途被告知，因此也只在最后一次做出了相应的改动，仍然需要日后对这一意识的巩固。  本次实验收获颇多，很好的起到了最后一次大实验对于verilog代码能力的考查，在经过此前多次实验的打磨之后，本次实验得以顺利的完成。 |