实验3：32位MIPS流水线处理器的设计

完成日期：2022/5/19 周训哲 20307110315

一、设计原理

1.体系结构状态和指令集

R类型算数/逻辑指令：add、sub、and、or、slt

存储器指令：lw、sw

分支指令：beq

拓展指令：addi、j、bne、ori、andi

2.MIPS结构设计

可将微体系结构分为两个互相作用的部分：数据路径（datapath）和控制（control）。

其中数字路径具体执行控制的指令，并对数据进行操作；而控制又接受到数字路径传来的指令，再控制数据路径执行。

数字路径包含了寄存器（regfile）、ALU、复用器（mux）、移位器（sl）、加法器（adder）、扩展器（符号扩展/零扩展）、计数器（flopr/flopenr）等结构元件。

控制包含了主译码器（maindec）和alu译码器（aludec）两个部分。

3.流水线微体系结构设计

流水线微体系结构（pipeline microarchitecture）将单周期微体系结构流水线化，使得可以同时执行多条指令，显著提高了吞吐量。流水线结构必须增加一些逻辑来处理多条正在执行指令之间的相关性。同时，还需要增加非体系结构流水线寄存器。增加这些逻辑和寄存器是值得的，虽然每条指令的延迟时间并没有改变，甚至由于加入寄存器反而增加了，但是流水线的吞吐量有了极大的提升，由于将一条指令分为，取指、译码、执行、存储器、写回五个阶段，其吞吐量可以提高几乎五倍。当前，所有的商业高性能处理器都使用流水线结构。

4.增加I/O接口

对于数据存储译码器模块进行更改，除了数据存储器以外，增加了I/O接口，用来处理NEXYS4 DDR开发板的外部输入，并且引入七段数码管模块对计算结果进行led显示的输出。

其中数据存储译码器（DataMemoryDecoder）模块对控制器传入的write信号进行选择，选择写入的数据传入数据存储器还是I/O接口，对处理器读入的数据也进行选择，是外部输入的数据还是数据存储器存储的数据，最终计算结果由七段数码管进行输出。

二、实验方案

（1）指令存储器（imem）：

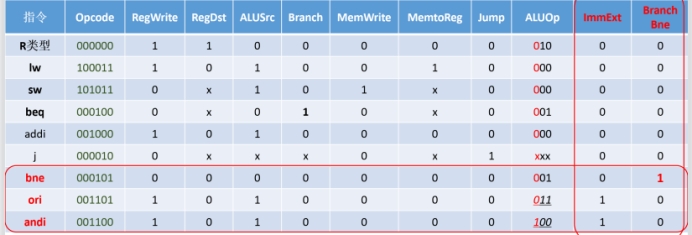
有一个读取端口，用来读取已有文件编写的汇编语言，并且将数据传输到rd上。

（2）数据存储器（dmem）：

有一个读/写端口，如果使能we为1，则在时钟的上升沿将数据wd写入地址a，如果使能为0，则将地址a读到rd。

2.指令的执行：

根据opcode，将控制指令进行拆解，得到主译码器真值表，分别对应各种控制指令，输出到aludec与datapath模块中进行计算和执行。



3.controller：

（1）maindec主译码器：

输入op地址，根据op地址的真值表，对

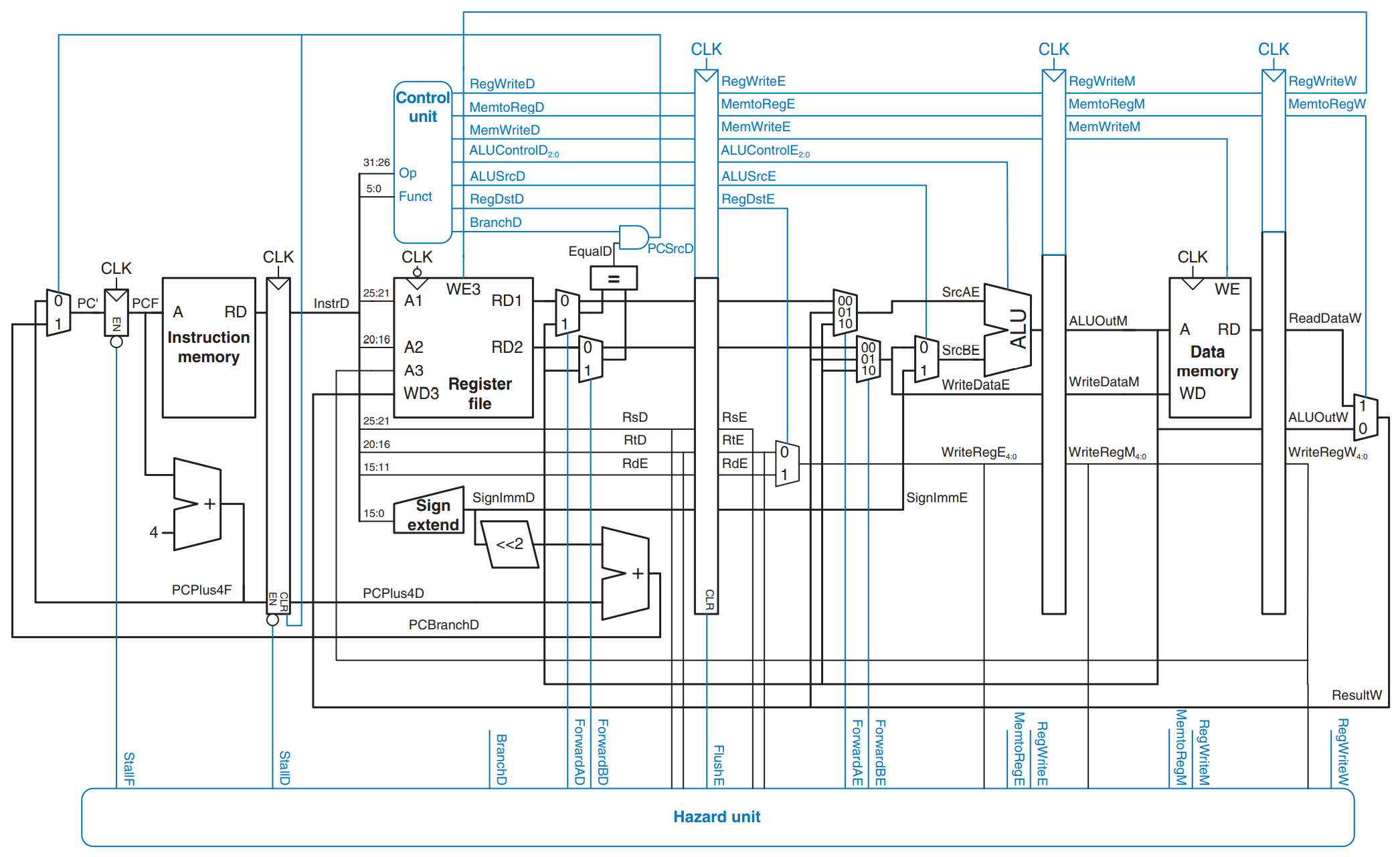
regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump,aluop,immext,branchbne进行赋值，并将数据输出。

（2）aludec ALU译码器：

输入功能指令及aluop指令，得到alucontrol指令对alu模块的计算进行控制。

（3）增加bne指令后，还要对分支指令进行处理。

4.datapath：主要的数据处理部分



读/写存储器和寄存器文件、使用 ALU通常构成处理器中的最大的延迟。我们选择5个流水线阶段，这样每一个阶段只完成一个慢操作。具体地，我们称这 5个阶段为∶取指令（Fetch）、译码（Decode）、执行（Execute）、存储器（Memory）和写回（Writeback）。它们类似于多周期处理器中执行lw指令的5个步骤。在取指阶段，处理器从指令存储器中读取指令。在译码阶段，处理器从寄存器文件中读取源操作数并对指令译码以便产生控制信号。在执行阶段，处理器使用ALU 执行计算。在存储器阶段，处理器读或写数据存储器。最后。在写回阶段，如果需要，处理器将结果写回到寄存器文件。

每个阶段结束后都会经历一个寄存器，会在一个时钟周期的上升沿将数据传入下一个阶段。

4.冲突（Hazard）

流水线系统中的核心问题是化解冲突（Hazard）。在后一条指令需要前一条指令的计算结果，而前一条指令还没有执行完时就会发生冲突。

(1)使用重定向解决冲突

有些数据冲突可以通过将存储器访问阶段或写回阶段的结果重定向或旁路到执行阶段的相关指令来解决。这需要在 ALU的前面增加复用器以便选择来自寄存器文件来的操作数，或存储器阶段或写回阶段的结果。当执行阶段中的指令有一个与存储器阶段或写回阶段中的目的寄存器相匹配的源寄存器时，需要重定向。

(2)使用阻塞解决冲突

当在执行阶段计算指令结果时，使用重定向解决 RAW数据冲突就足够了，因为它的结果可以重定向到后一条指令的执行阶段。但是，1w指令直到存储器阶段后才能完成读数据，因此它的结果不能重定向到下一条指令的执行阶段。我们说1w指令有两个周期延迟。因为相关指令直到两个周期后才能使用它的结果。1w指令在周期4的结尾才从存储器中接收数据。但是 and指令周期4 的开始时就需要这个数据作为源操作数。使用重定向无法解决这种冲突。

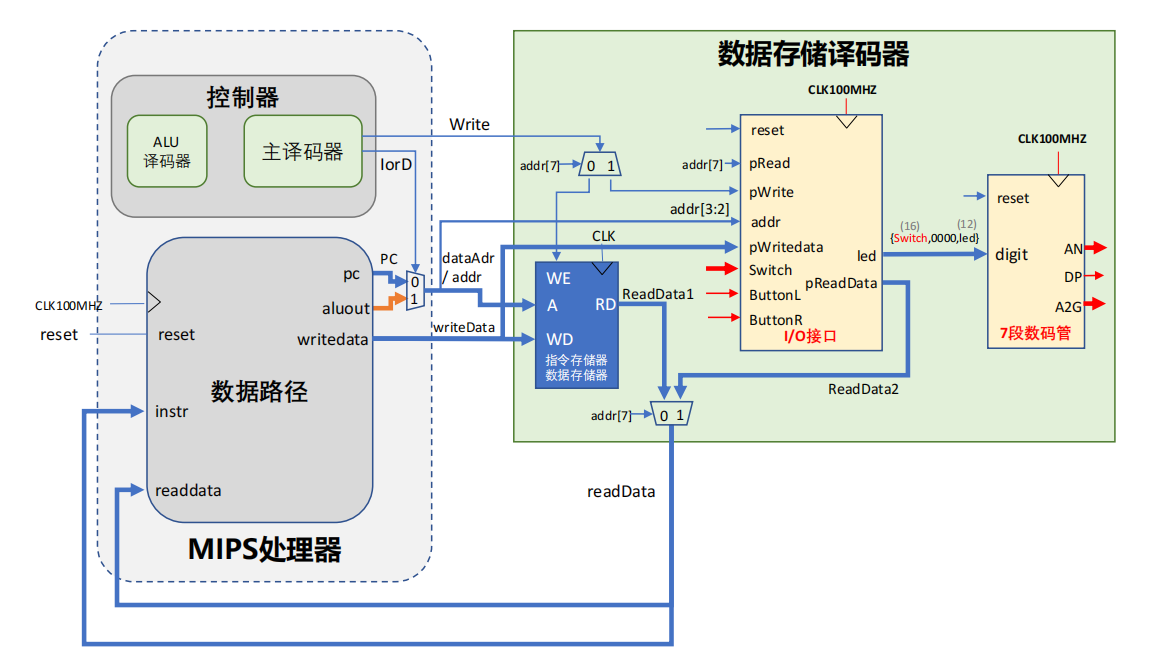
可以通过禁止流水线寄存器来阻塞某个阶段，使得寄存器的内容不改变。当某阶段流水线被阻塞时，所有前面的各阶段也都应该被阻塞，这样后续的指令就不会丢失。在阻塞阶段后的流水线寄存器必须清除，防止错误信息传播重定向。阻塞降低性能，因此它们只有在必需时才能使用。

(3)解决控制冲突

beq指令将产生控制冲突，因为在取下一条指令时分支是否发生还尚未确定，所以流水线处理器不知道取哪条指令。处理控制冲突的一种机制是阻塞流水线直到确定分支是否发生为止（计算出 PCSrc）。因为确定分支是在存储器阶段完成的，所以流水线将在每个分支阻塞3个周期。这将严重降低系统的性能。

另一种解决方法是预测分支是否发生。并基干该预测来执行指令。日确定发生分支。如果预测是错误的，则处理器将抛弃这条错误的指令。尤其是，假设我们预测所有的分支都不会发生而只是简单地按顺序执行程序。如果分支的确发生了，则分支指令后的3条指令将通过清除这些指令的流水线寄存器来刷新（抛弃）。这些浪费的指令周期称为分支错误预测代价。

6.数据存储译码器



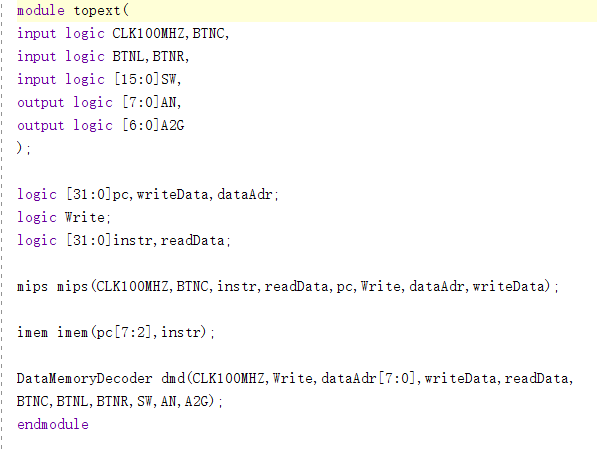
主要分为三个模块，数据存储器对传输的数据进行存储及运算输出，I/O接口接收外部接口（switch）的数据，将数据进行输出，同时led数据对七段数码管进行控制，使开发板显示出结果。

Datapath中alu模块输出aluout和pc在iord的选择结果作为addr,addr[7]作为pread控制读入操作,控制器传出的write在addr[7]的控制下,选择输入到idmem与io控制读入的模块。Switch，btnl，btnr作为外部信号控制读入与写出操作。Writedata信号分别传入idmem和io模块进行写入操作，最终输入为readdata1和readdata2，

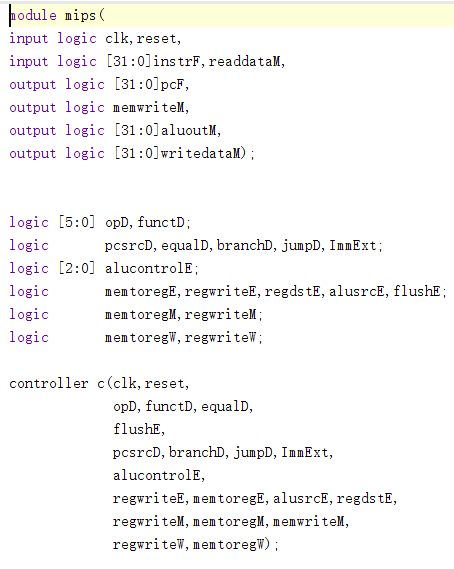
在addr[7]的控制下选择输出结果readdata作为instr和readdata信号传入数据路径中。

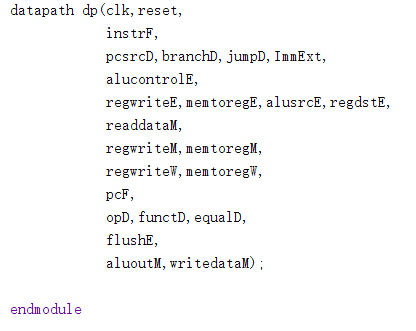
三、关键代码

**Top：**

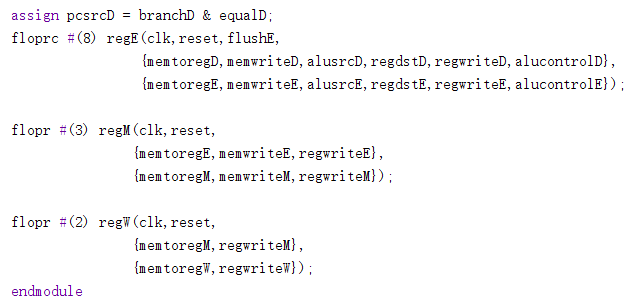
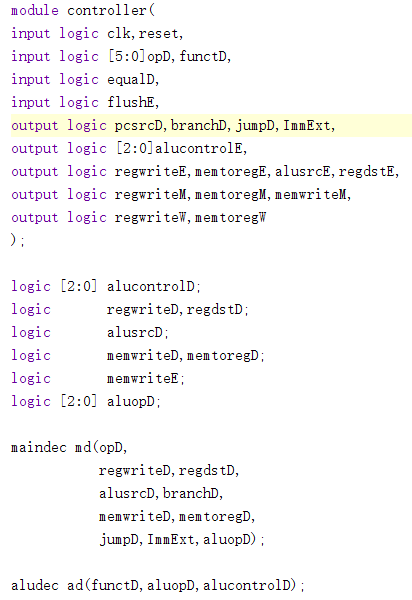


**Mips**

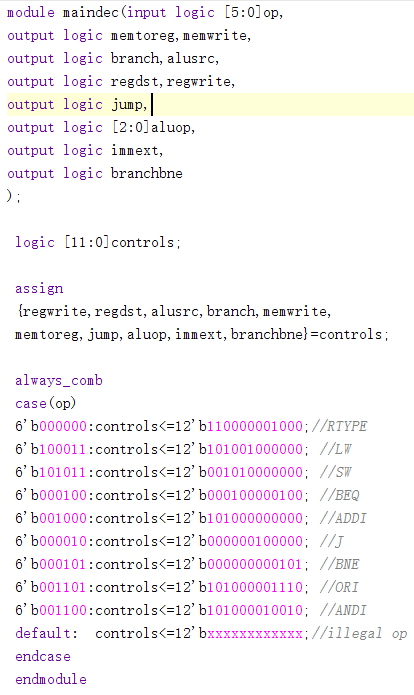




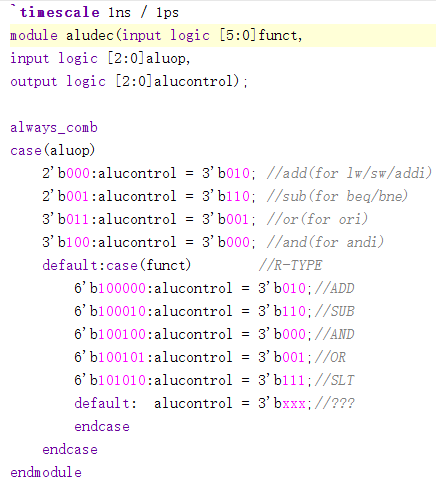
**Controller**

****

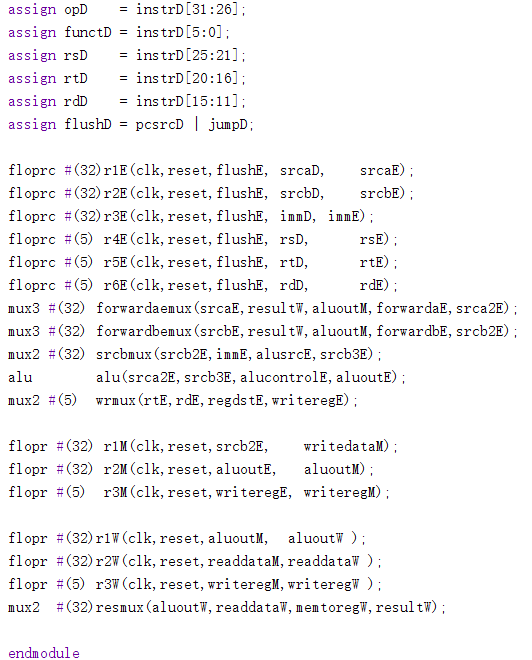
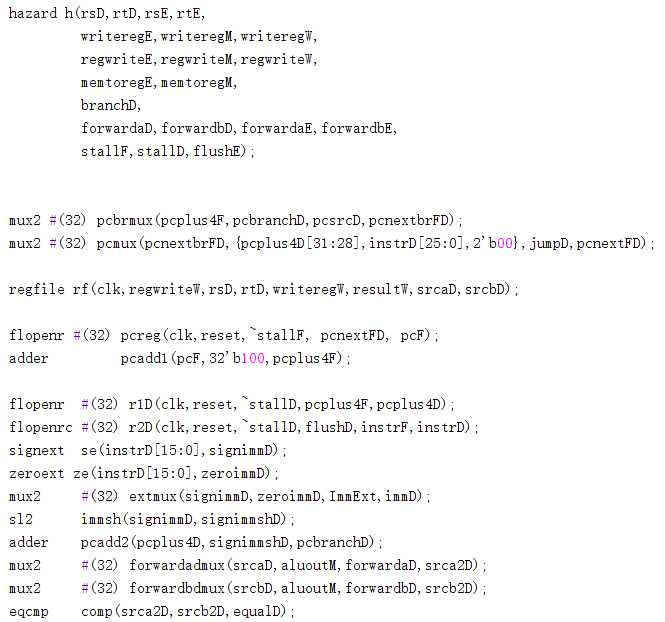
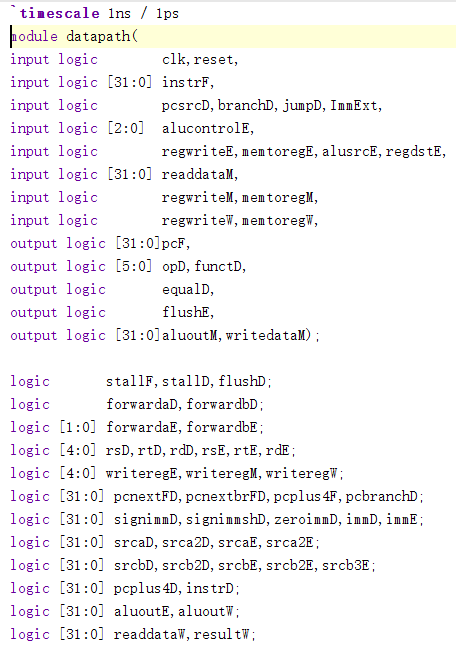
**Maindec**



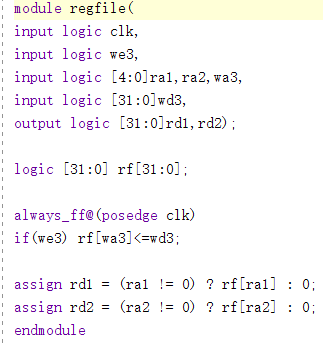
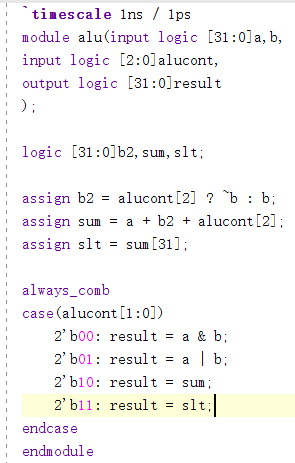
**Aludec**



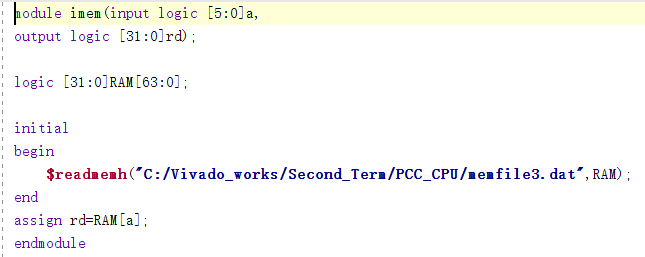
**Datapath**



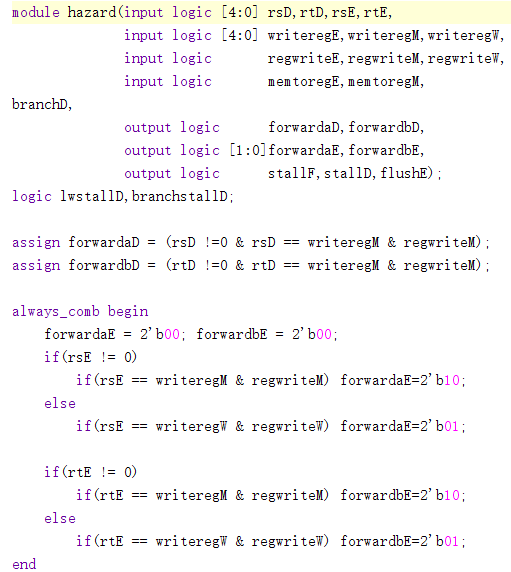
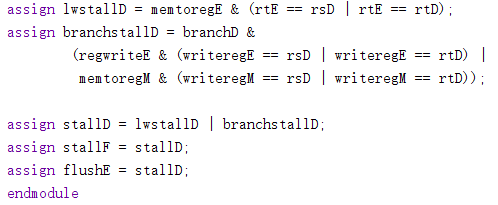
**Regfile Alu**

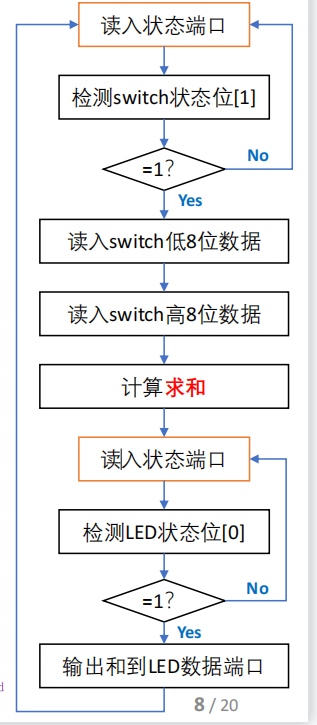
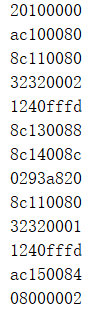
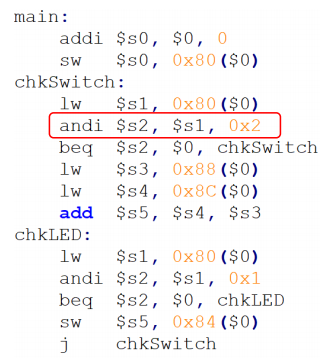
**Imem**



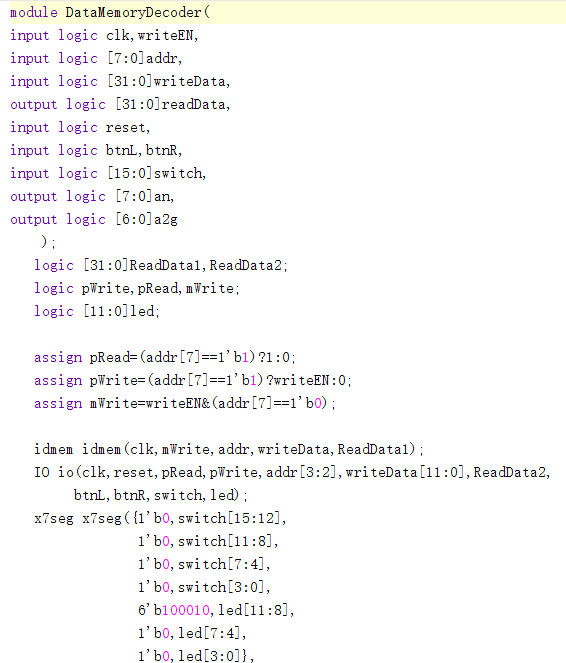
**Hazard**

**** ****

**Memfile3.dat**

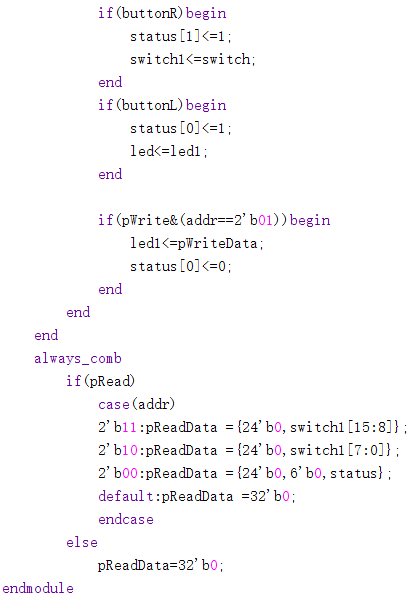
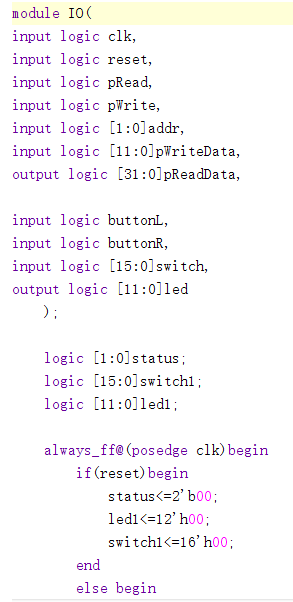


**Datamemorydecoder**

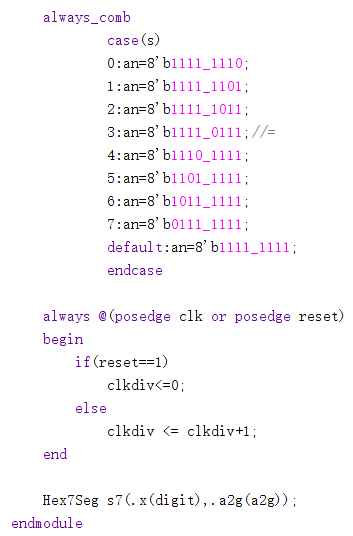
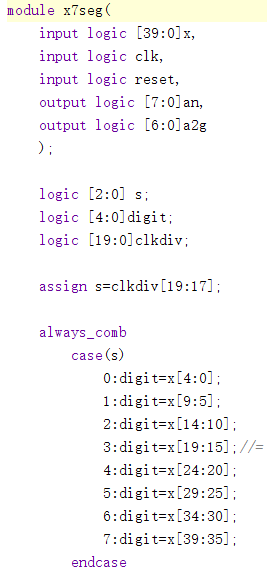


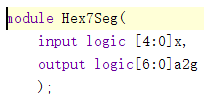


**IO**



**X7seg**

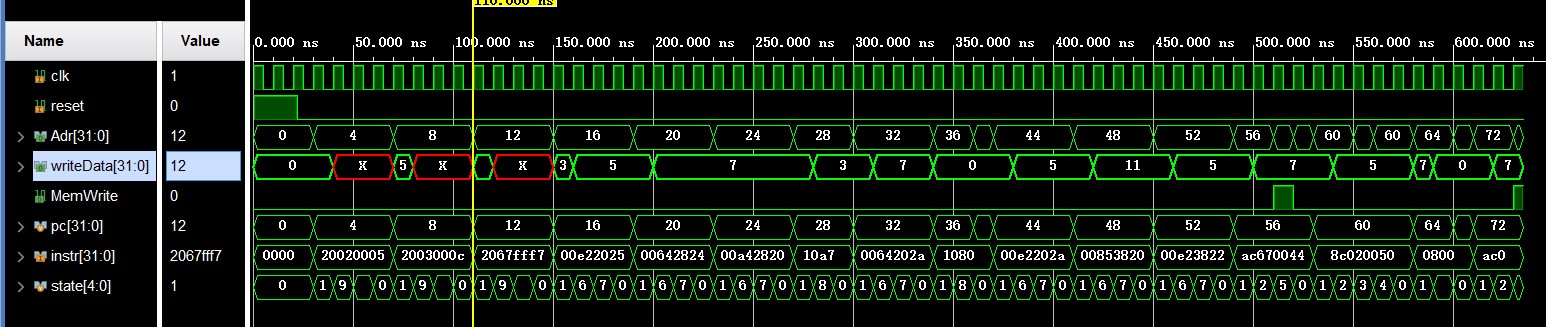




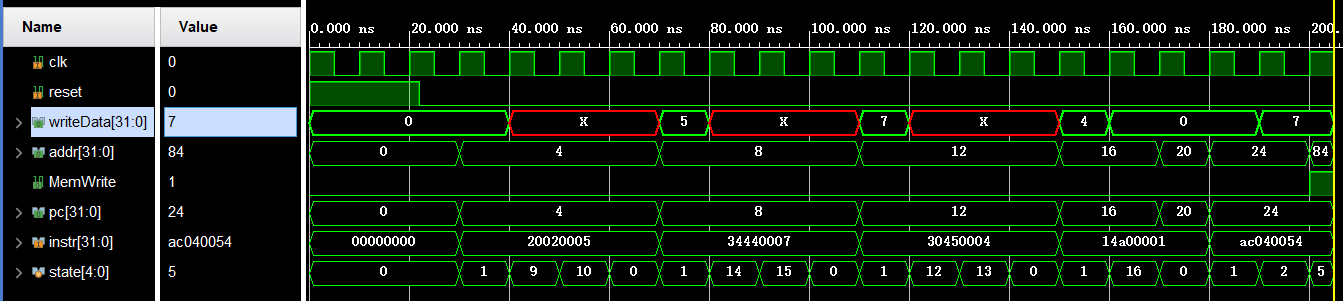


四、仿真截图

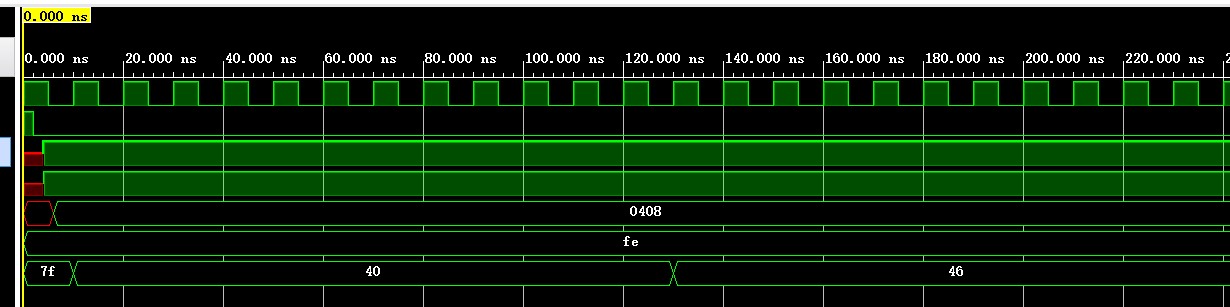
1.拓展前仿真测试



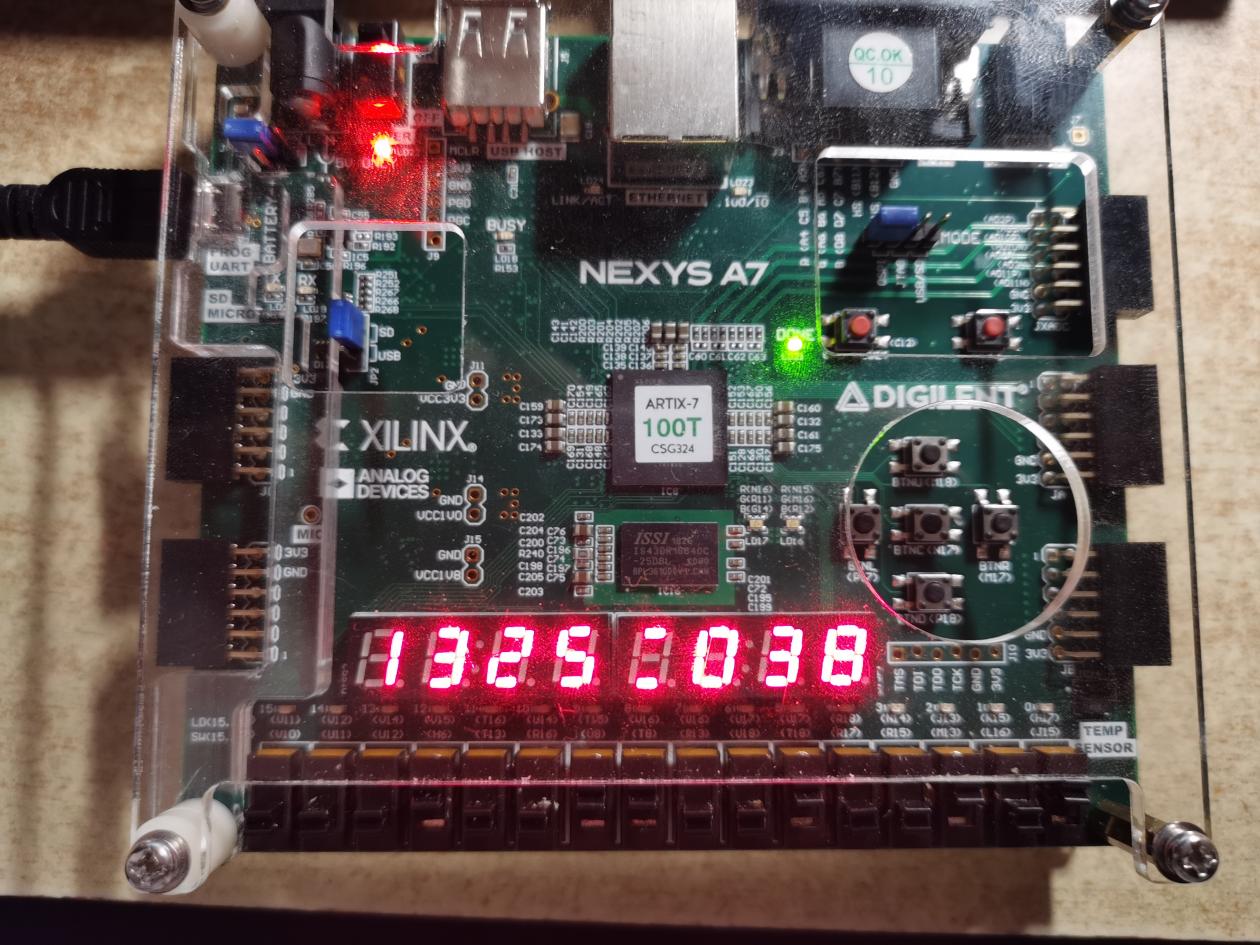
2.扩展后仿真测试

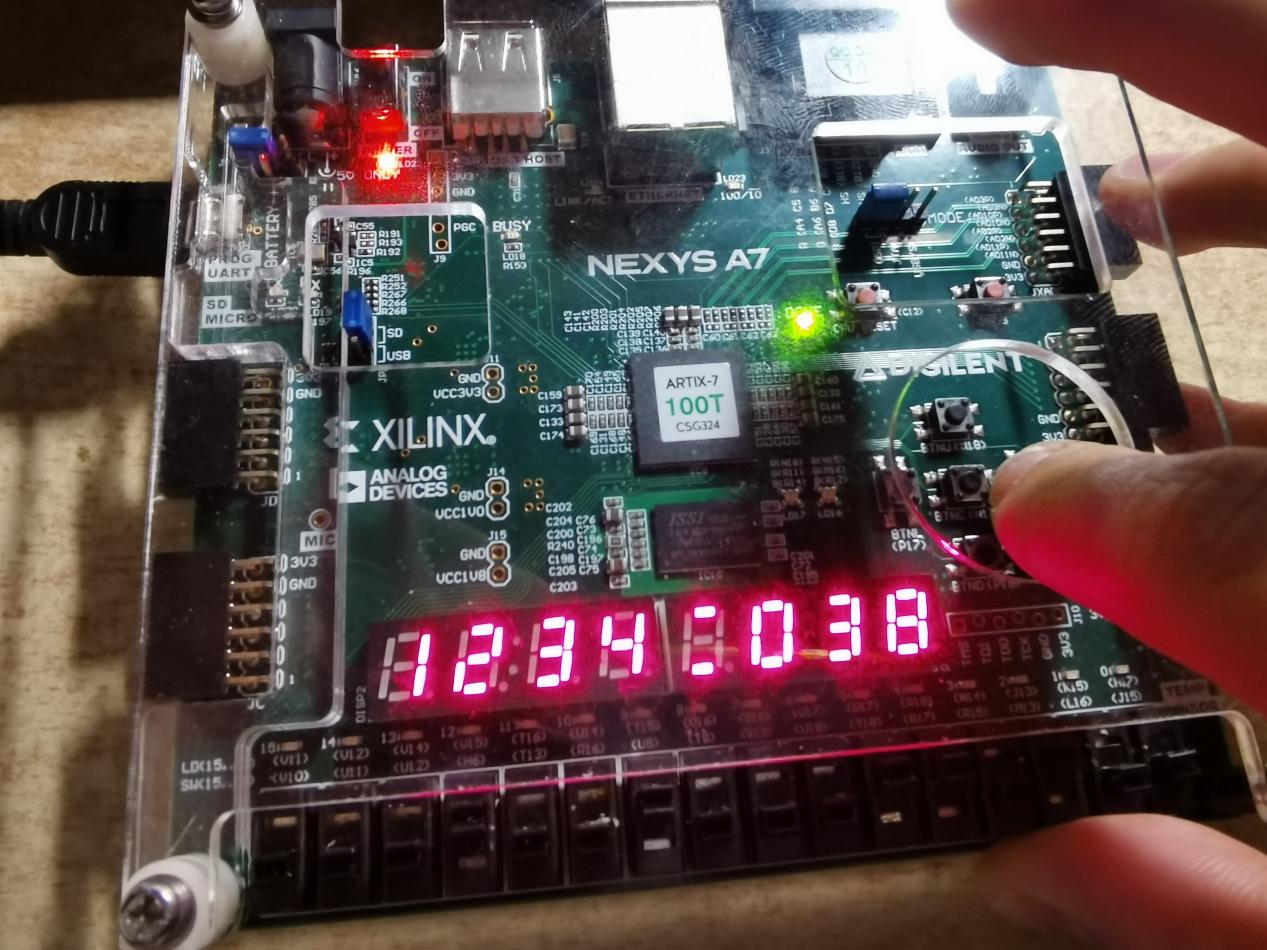


3.增加I/O接口仿真测试

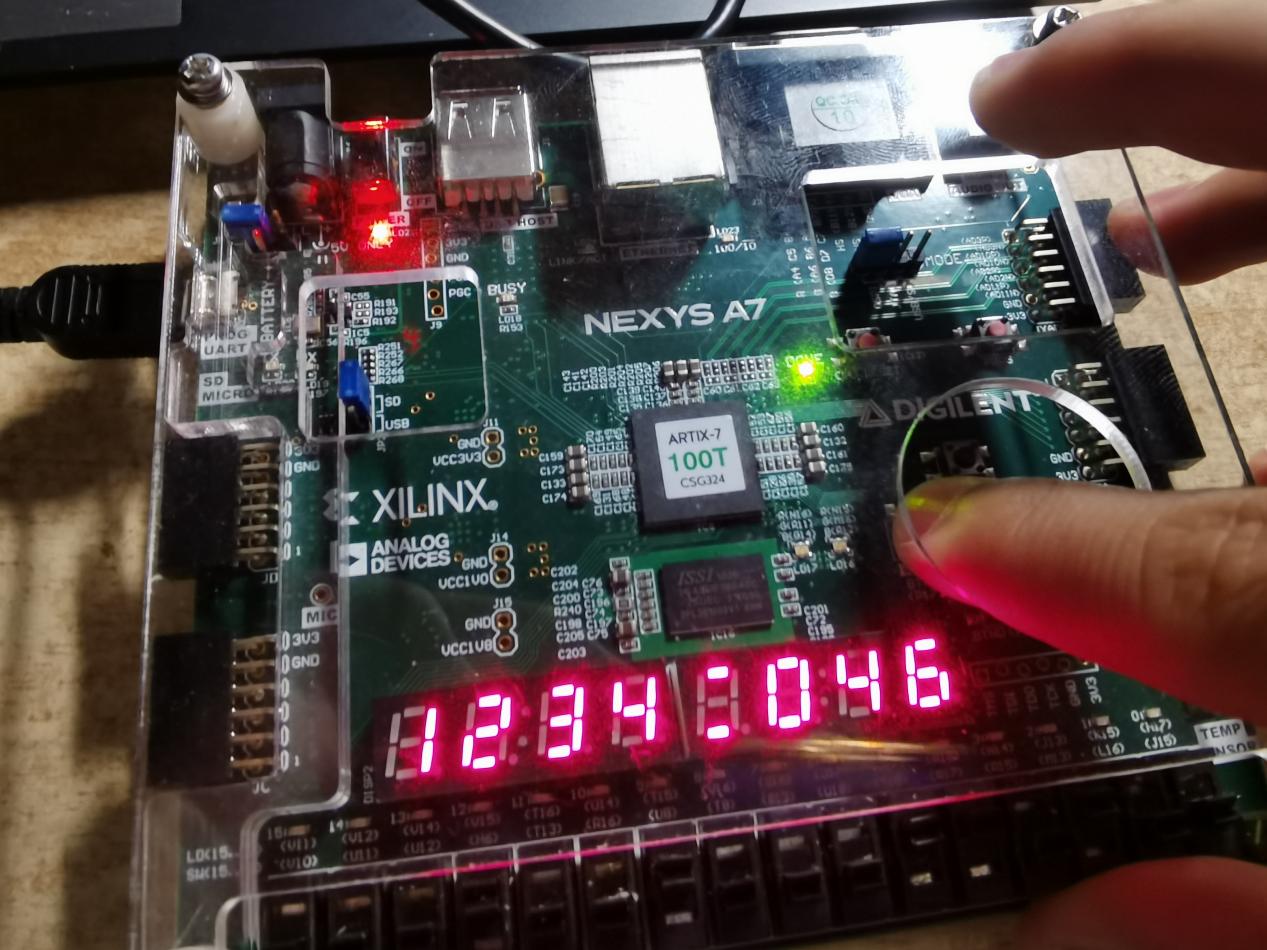


五、实验开发板照片

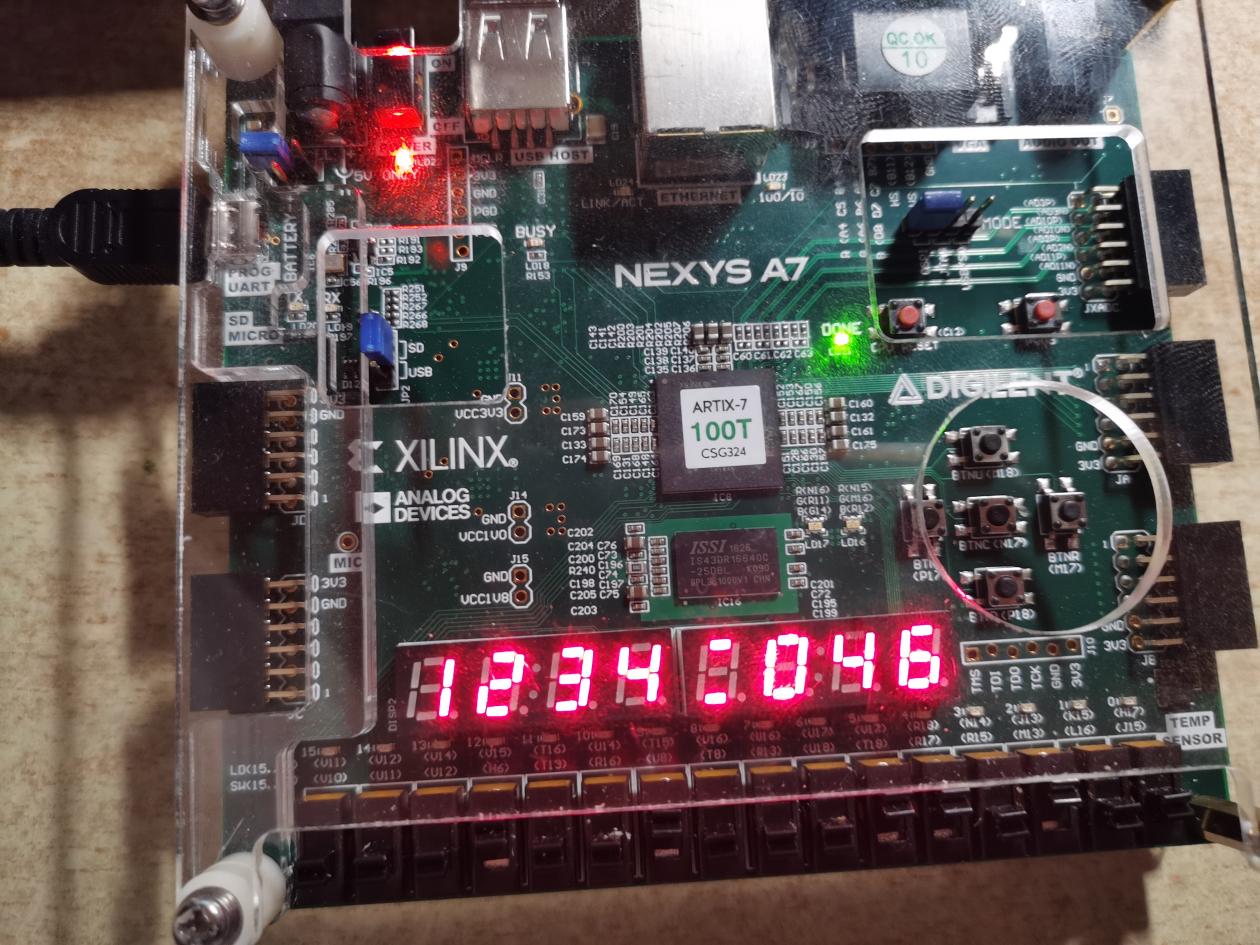




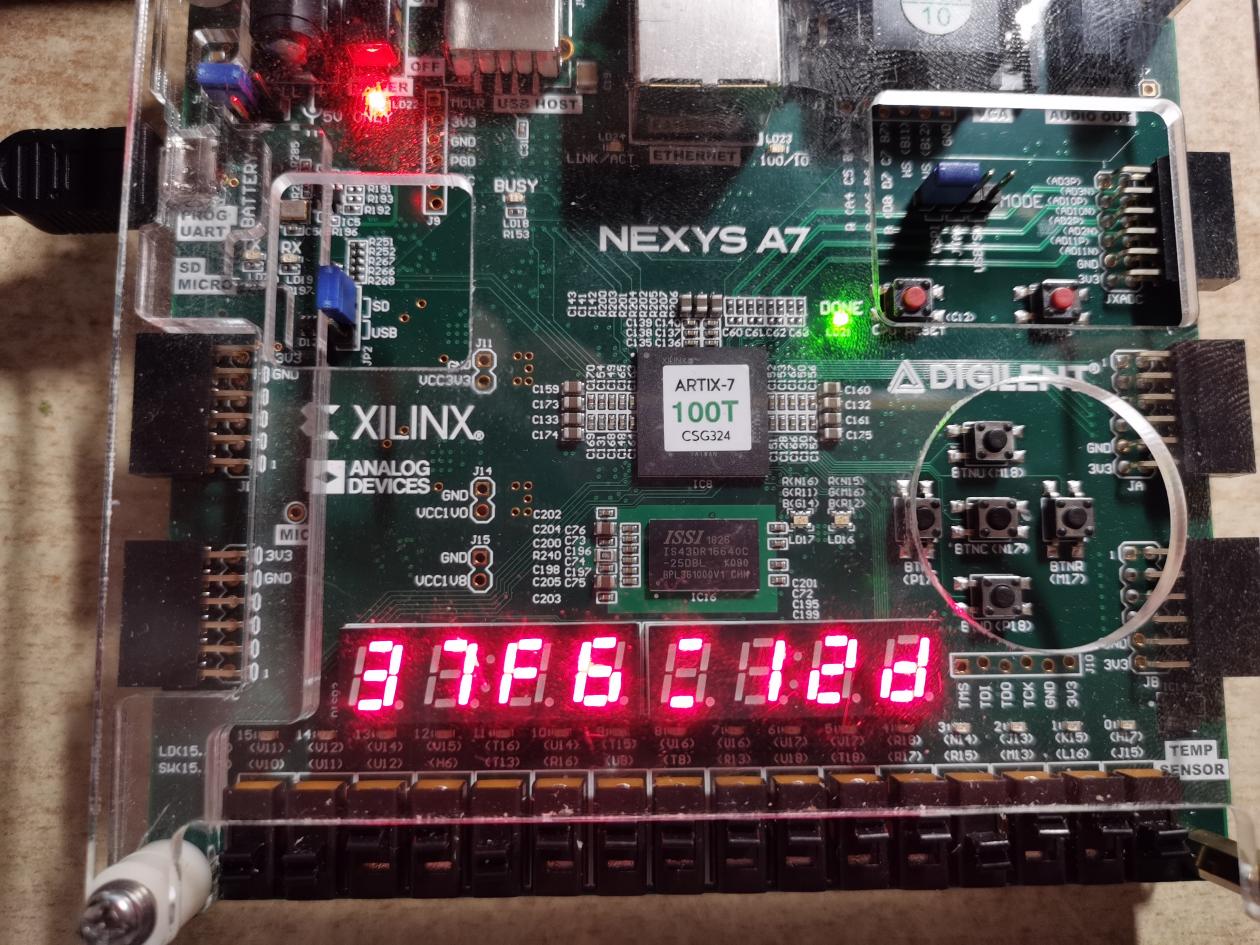
使用btnr读取数据

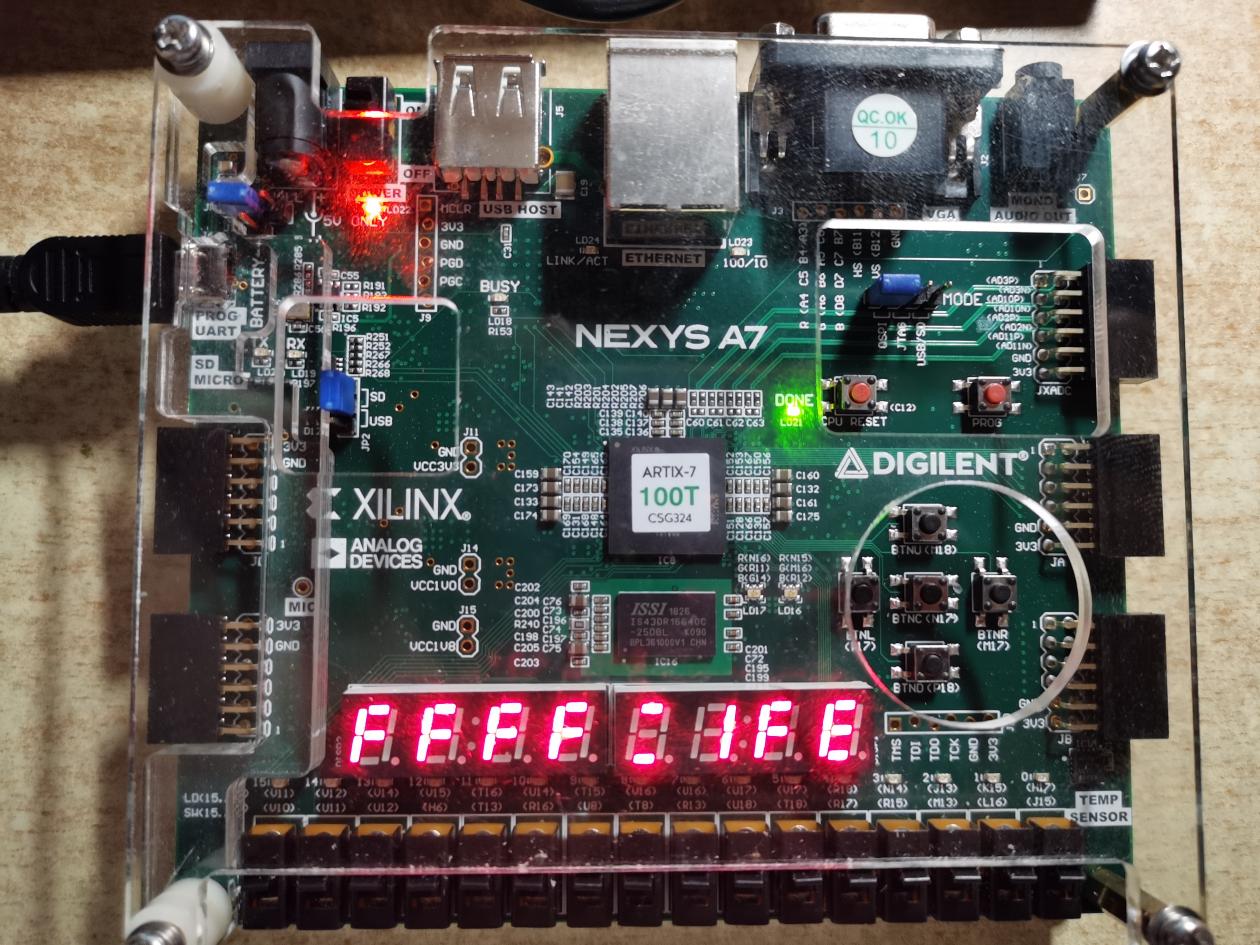


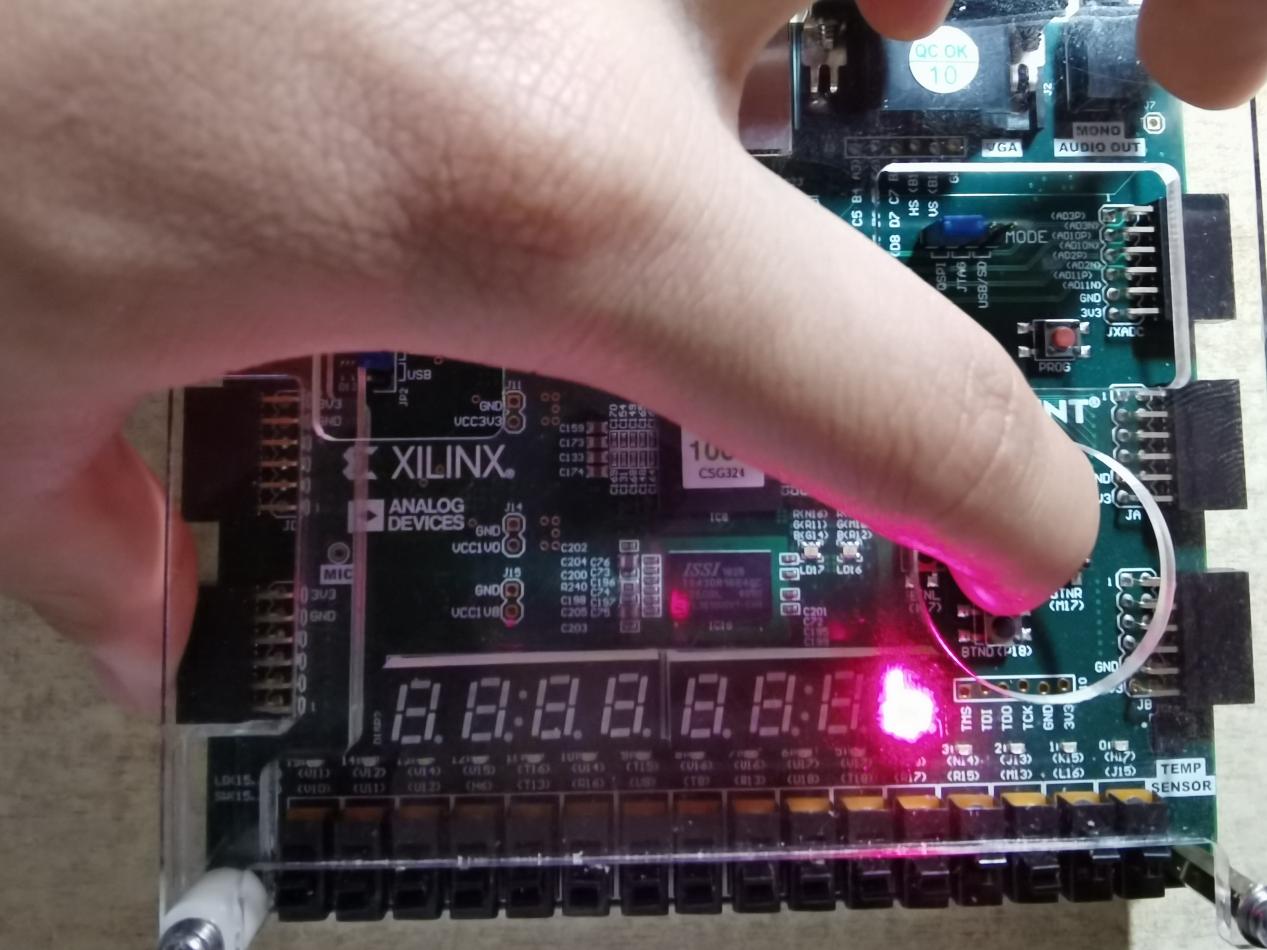
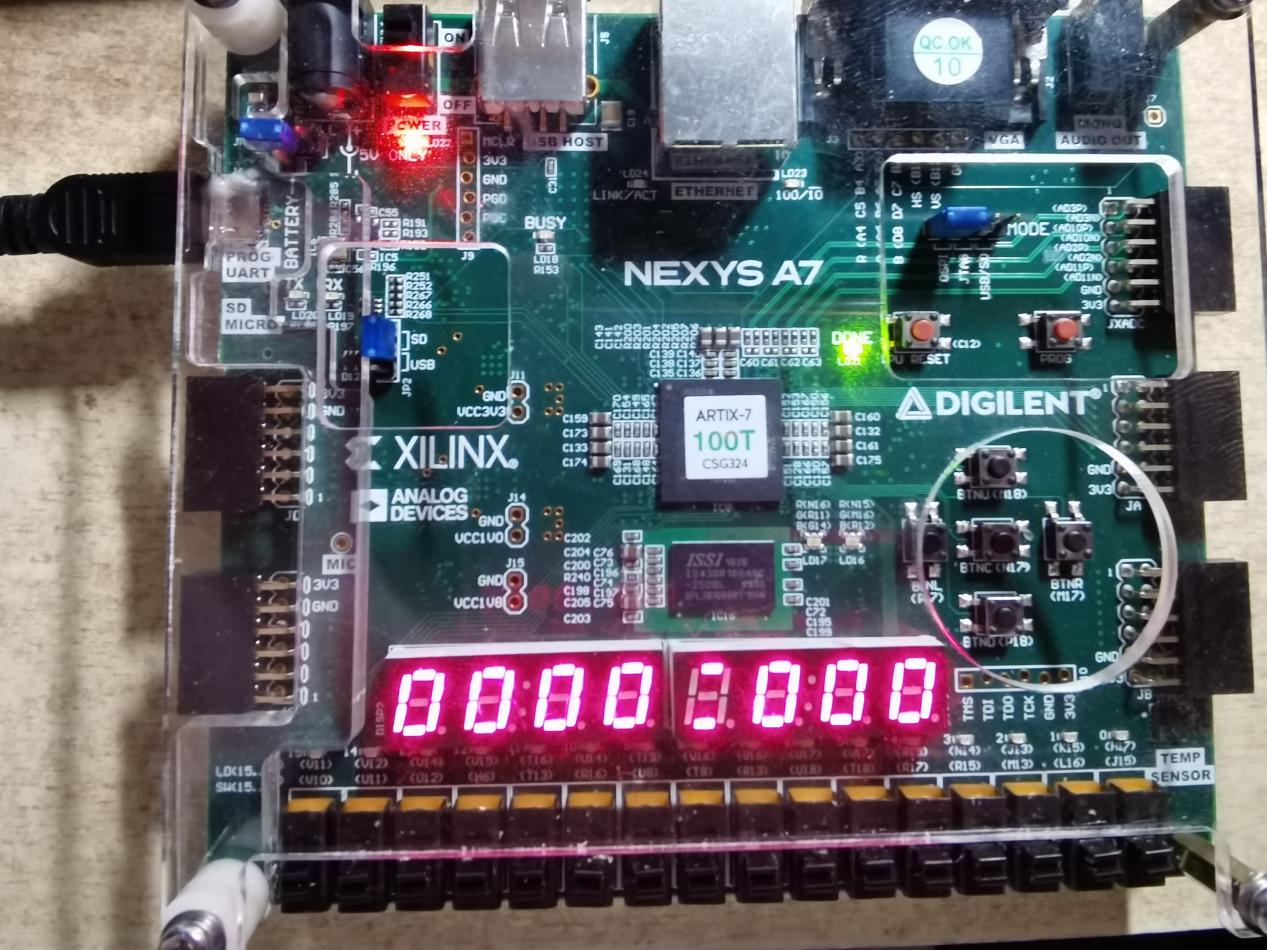
使用btnl计算并显示结果



最终结果展示







使用btnc实现重置操作

六、总结

1.调试发生的问题与对应解决方案

1.运行结果进入到default，仿真显示结果全为x

解决方案：利用仿真数据，根据电路图，逐层向上查找，最终寻找到发生错误的最初模块，进行检查与修改，最后发现顶层文件设置错误导致。

2.运行结果有浮空值z

解决方案：说明有结果并未读入，检查输入端是否有数据输入，若无，则检查上一层模块的数据是否正常，若正常，这检查浮空值模块代码是否有误，若异常，则检查上一层代码是否存在问题，检查方式为逐层检查。

3.仿真时得到的最终地址为84正确，但数据结果并不是7：

解决方案：调用查看在执行指令时的状态图，根据状态图以及对应结果对代码进行修改，最终解决了问题。

4.io接口时，仿真文件无法得到结果，在开发板上运行时结果显示为0

解决方案：先检查是否有数据出错，发现数据无误，最终发现在idmem读入汇编代码时就无法读入，更改文件格式从.txt到.dat后结果运行正常。

5. Memwrite没有写入dmem，原因是没有区分大小写，导致dmem的使能we为浮空值。

2.收获与体会

由于本次实验为设计一个完整的mips流水线处理器，相比于单周期设计，本次实验在其基础上进行设计，有了之前的经验，难度减小了不少。这次实验除了让我了解了流水线cpu的工作原理之外，最重要的还是让我更加熟练了调试的技巧，以及报错之后的修改方式。本次实验，不仅会加深我对流水线cpu构造的认识，还会对我今后的硬件语言学习与调试有着极大的帮助！