实验2：32位MIPS多周期处理器的设计

完成日期：2022/4/20 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：beq1

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.多周期微体系结构设计

多周期微体系结构与单周期微体系结构的对比：

第一，单周期在一个时钟周期内执行完一条完整的指令，需要足够长的周期来完成最慢的指令（1w）；多周期每条指令分成多个阶段，每个阶段在一个时钟周期内完成，极大节约了运行时间。第二，单周期需要3个加法器（一个用干 ALU。两个用干 PC 的逻辑），而加法器是相对占用芯片面积的电路；多周期将多个加法器复用到一个ALU中，极大节约了芯片空间。第三，它采用独立的指令存储器和数据存储器，而这在实际系统中是不现实的。大多数计算机有一个单独的大容量存储器来存储指令和数据，并且支持读和写操作，多周期将指令存储器和数据存储器合并为一个存储器。

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。

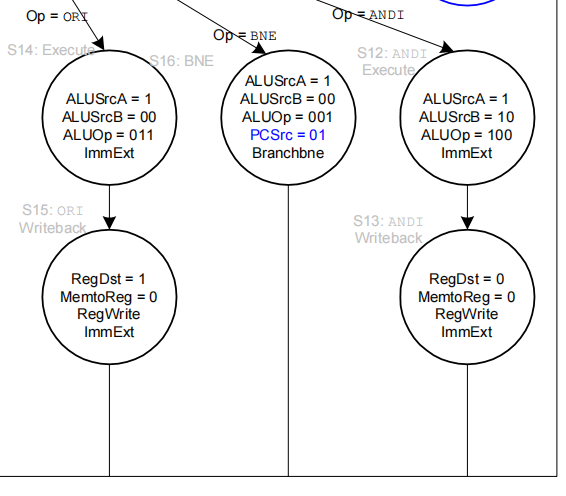
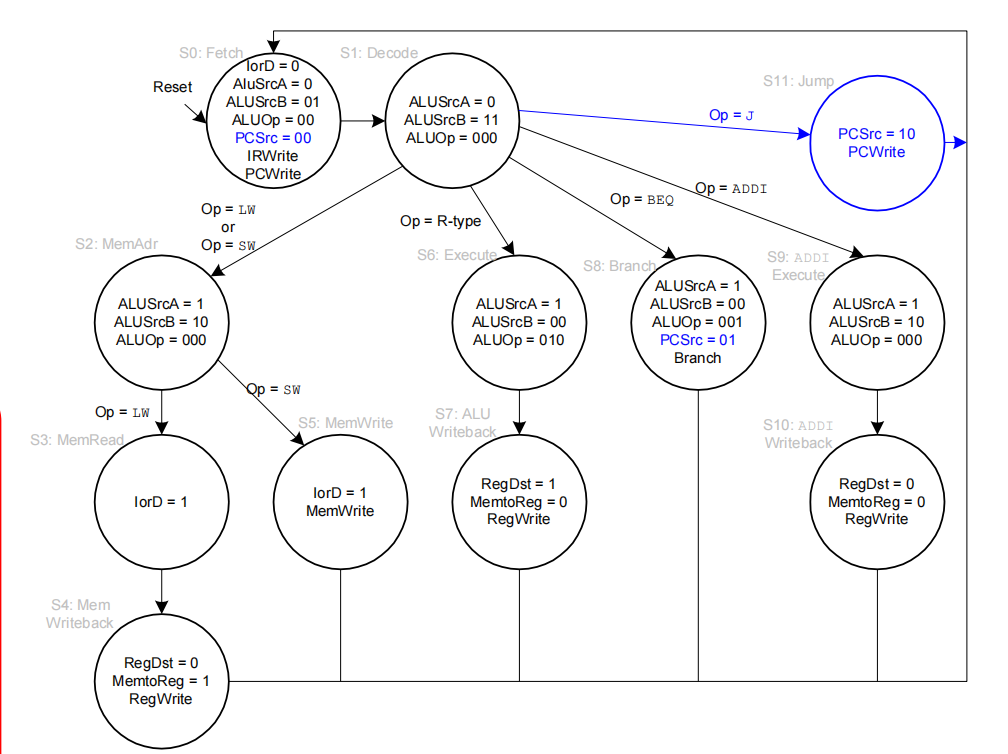
二、实验方案

1.存储器模块（idmem）

由于需要在一个周期内读指令存储器且读或写数据存储器，所以选择使用一个指令和数据组合的存储器。在一个周期内读指令。然后在单独的周期内读或写数据。

2.控制信号：

根据opcode字段和funct字段计算控制信号，主控制器在合适的周期应用合适的控制信号的FSM，具体的控制信号与当前执行的指令有关，最后将maindec的控制信号输出到aludec与datapath模块中进行计算和执行。



（1）第一步是根据PC中保存的地址从存储器中取出指令。为了读存储器，令 lorD=0，这样可以从 PC获得地址。IRWrite 有效以便将指令写入指令寄存器IR中。同时，PC应该递增4 以便指向下一条指令。由于 ALU 此时没有被使用，所以处理器可以在 ALU取出指令的同时使用它计算 PC+4。ALUSrcA=0，所以SrcA=PC。ALUSrcB=01， 所以SrcB=4。ALUOP=000使 ALU执行加法。为了用这个值更新 PC，令 PCSrc =0并设置 PCWrite 为有效。

（2）下一步将读寄存器文件，并对指令译码。寄存器文件总是读由指令的 rs和 rt 字段指定的两个源。同时，对立即数进行符号扩展。译码操作包括检查指令的 opcode 字段以便决定后续的操作。译码指令不需要控制信号，但是需要等待一个周期来完成读和译码。为了兼顾beq和bne指令，处理器必须计算目的地址，并比较两个源寄存器来确定是否发生转移。这需要使用ALU 两次。然而，我们注意到在状态 S1期间在读寄存器时，没有使用ALU。处理器也可以在此时使用ALU 通过将递增的 PC（PC+4）与 Imm×4相加来计算目的地址。ALUSrcA=0，选择递增的 PC;ALUSrcB=11，选择 lmm×4;ALUOp=000，完成加法。目的地址存储在 ALUOut中。如果指令不是beq或bne，计算的地址将不在随后的周期中使用。

（3）存储器的装入和存储（lw或sw）

多周期处理器通过将基地址和符号扩展的立即数相加来计算地址。这需要 ALUSrcA=1以便选择寄存器A，ALUSrcB=10 以便选择 Signlmm。ALUOp=000，因此 ALU执行加法。有效地址将存储在ALUOut 寄存器中为下一步使用做准备。

若指令为lw，多周期处理器必须紧接着从存储器中读取数据并将数据写入寄存器文件中。为了读取存储器。令IorD=1 以便选择刚刚计算的并存入ALUOut 中的存储器地址。读存储器地址，并将结果在步骤 S3存入Data 寄存器中。在下一步 S4 中，将Data写入寄存器文件。MemtoReg =1，选择 Data。RegDst =0，从指令的 rt字段获得目的寄存器地址。RegWrite有效，执行写操作，执行lw指令。最后，FSM将返回到初始状态 SO，取出下一条指令。

若指令为sw，从寄存器文件的第二个端口读取的数据简单地写入存储器中。在状态 S3，IorD=1，选择S2中计算的并保存在 ALUOut 中的地址。将 MemWrite 将设置为有效以便写存储器。此外，FSM返回到状态 S0 以便取出下一条指令。

（4）R类型指令和立即数指令

多周期处理器使用 ALU计算结果并将结果写入寄存器文件。如果是R类型指令，通过选择寄存器A和 B（ALUSrcA=1，ALUSrcB=00），如果是立即数指令，选择寄存器A和扩展数imm（ALUSrcA=1，ALUSrcB=10）并根据指令的 funct 字段执行的 ALU操作来执行指令。对于R类型指令，ALUOp =010。ALUResult存储器在 ALUOut中。然后在下一状态，将 ALUOut 写人寄存器文件，RegDst=1，因为目的寄存器由指令的rd字段指定。MemtoReg=0，因为写数据WD3来自ALUOut。RegWrite 设置为有效以便写寄存器文件。

（5）beq和bne指令

处理器通过减法并判断结果是否为0来比较两个寄存器。ALUSrcA=1，选择寄存器A;ALUSrcB=00，选择寄存器B;ALUOp=001，执行减法;PCSrc=1，从 ALUOut 取出目的地址;如果ALU结果为0，则 Branch=1，如果ALU结果不为0，则Branchbne=1，得到branch和branchbne用来计算pcen对PC的赋值的寄存器进行使能，并用ALUOut地址更新PC。

（5）jump指令

修改数据路径，为主指令计算下一个 PC值。然后，我们给主控制器增加一个新的状态来处理这条指令。跳转目的地址由指令的26 位 addr 字段左移2位得到，接着预先设计已经递增的 PC的最高4 位。扩展 PCSrc 复用器将此地址作为第三个输入，PCSrc=10控制写入PC的是跳转的指令。

3.controller：

（1）maindec主译码器：

输入op地址，根据不同状态对应的FSM，对

pcwrite,memwrite,irwrite,regwrite,

alusrcA,branch,iord,memtoreg,regdst,

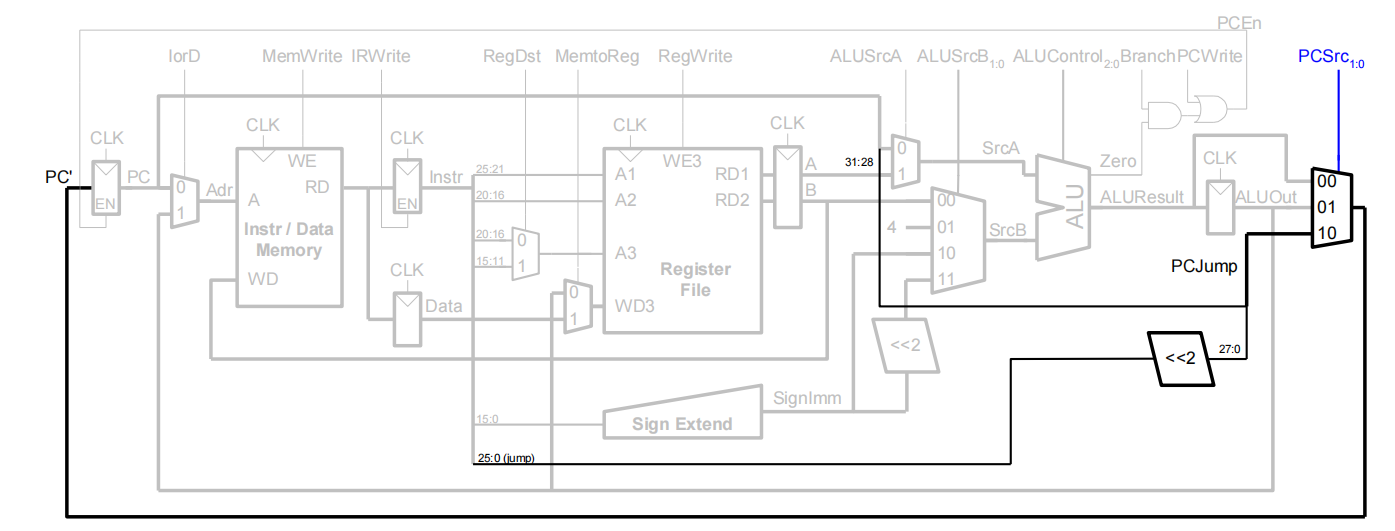
alusrcB,pcsrc,aluop,immext,branchbne进行赋值，并将数据输出。

（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



（1）首先，由存储器读入数据，将RD首先存入指令寄存器，在irwrite使能信号和clk时钟信号的控制下输出为instr，将RD存入数据寄存器，在时钟信号控制下输入为Data。

（2）32个单元×32位寄存器文件有两个读端口和一个写端口。将instr[25:21],instr[20:16] 输入到读端口，具有5位地址输入 A1和A2，每个用于指定2^5=32个寄存器中的一个作为源操作数。instr[15:11],instr[20:16]由regdst选择的结果传入到寄存器文件写端口具有5位地址输入A3中。它们可以读32 位寄存器的值并分别传送到 RD1和RD2 上。32位数据为data和aluout在memtoreg选择下的结果输入WD3。写入使能 WE3为regwrite和时钟信号 CLK。如果写入使能为1，则寄存器文件将在时钟的上升沿将数据写入特定寄存器。

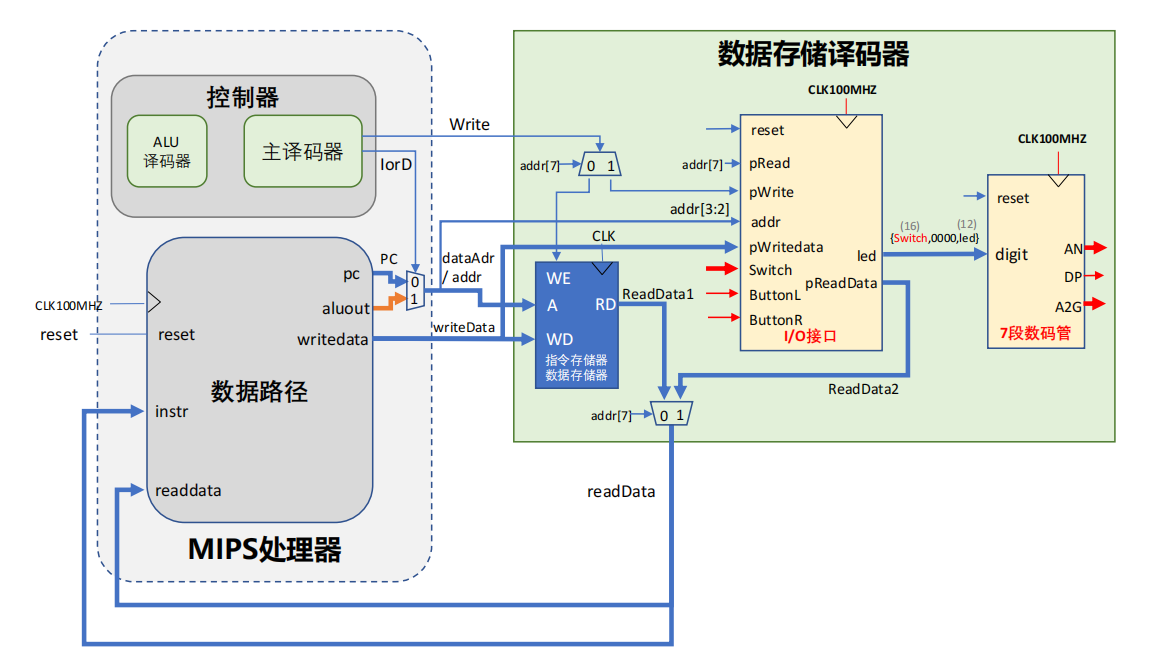
（3）RD1和RD2分别存入寄存器A和B，在时钟信号控制下输出。RD1和pc在alusrcA的选择下输出为srcA， instr[15:0]进行位扩展,符号扩展及零扩展的由immext选择的结果,再与RD2、立即数4、位扩展\*4的结果在复用器下由alusrcB选择的结果选择得到srcB。

将scra与srcb在alucontrol指令下在alu模块进行运算输出结果为aluresult,并且alu模块返回zero传入控制单元，在控制单元，zero信号与branch和branchbne进行处理输出为pcen。

（3）alu模块输出计算结果aluresult存到寄存器aluout中，同时计算跳转结果pcjump，aluresult，aluout，pcjump在pcsrc信号的选择下输出为pc’信号

（4）pc’信号存到寄存器中，在pcen使能信号和时钟信号控制下输出为pc，同时与aluout寄存器结果经iord信号选择为地址adr传入到存储器单元中。

5.数据存储译码器



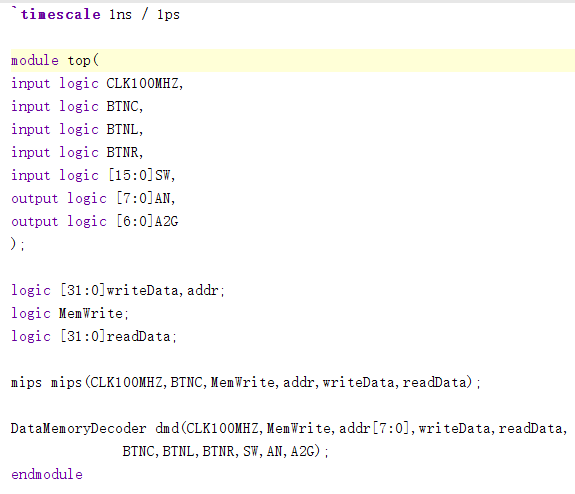
主要分为三个模块，数据存储器对传输的数据进行存储及运算输出，I/O接口接收外部接口（switch）的数据，将数据进行输出，同时led数据对七段数码管进行控制，使开发板显示出结果。

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，

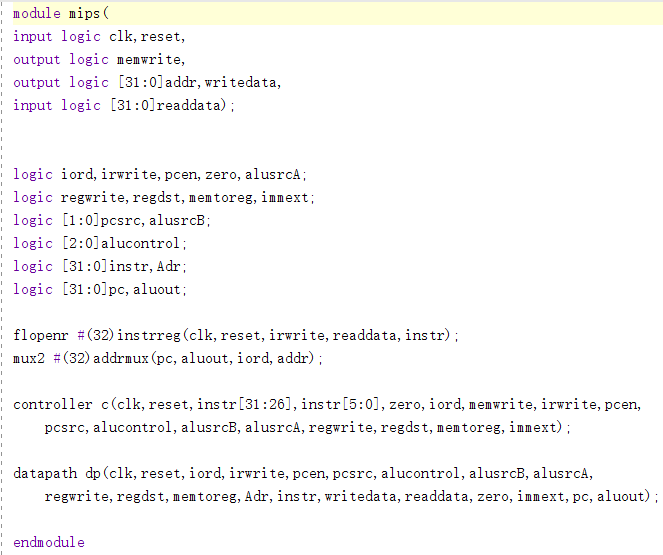
在addr[7]的控制下选择输出结果readdata作为instr和readdata信号传入数据路径中。

三、关键代码

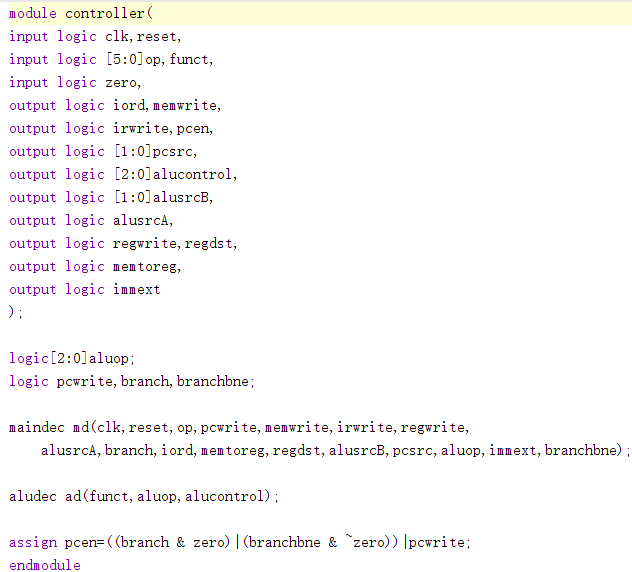
**Top：**



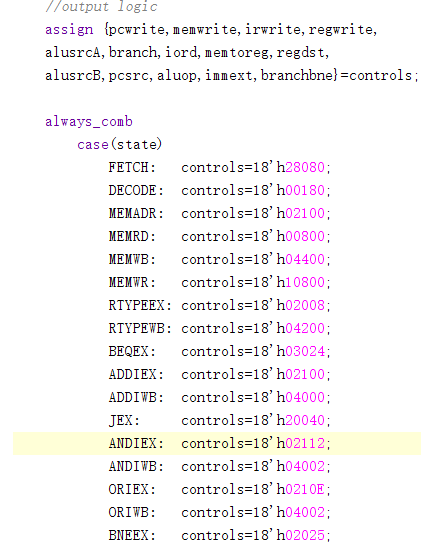
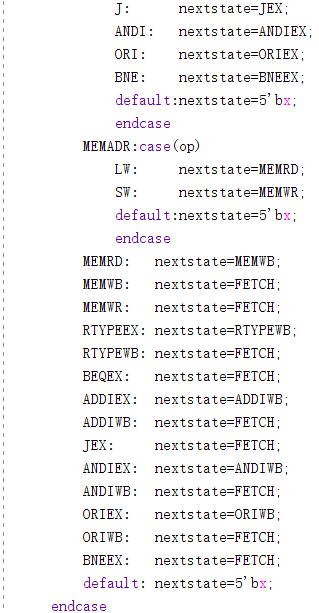
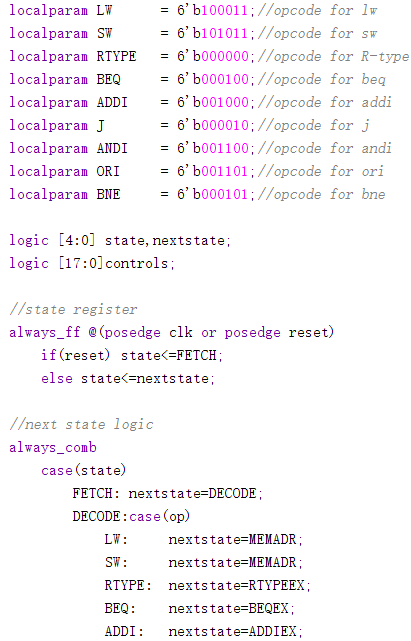
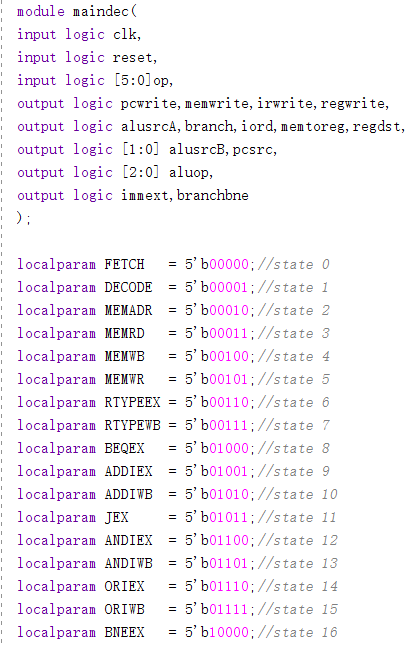
**Mips**



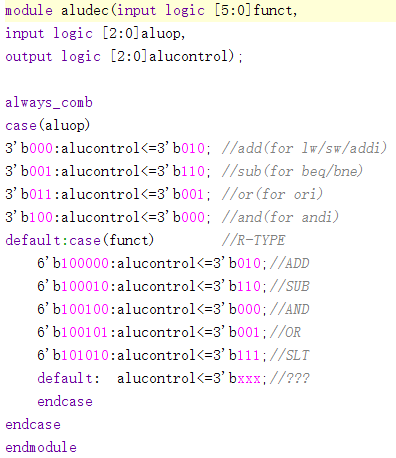
**Controller**

****

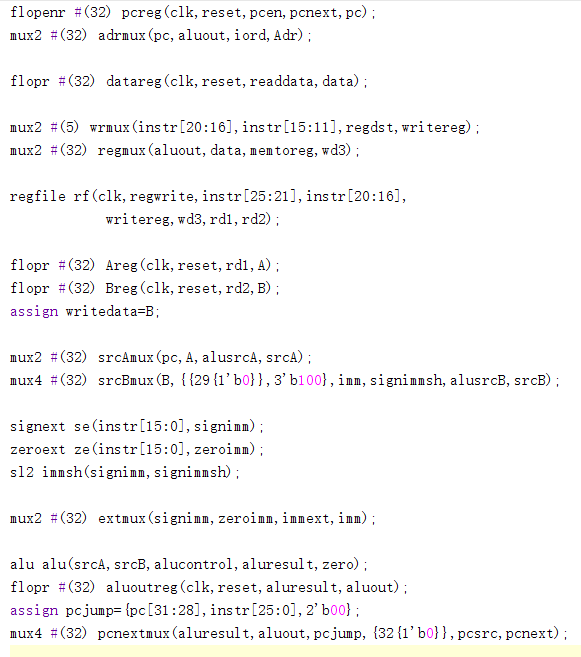
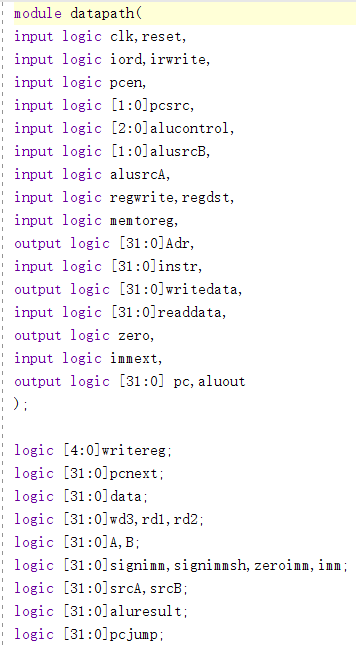
**Maindec**



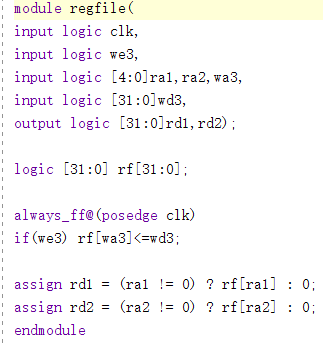
**Aludec**



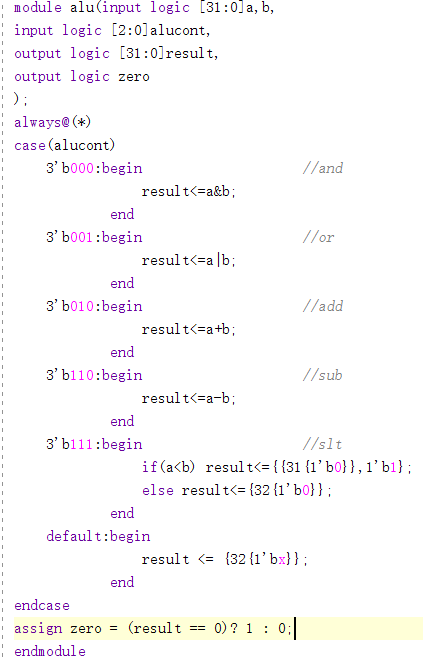
**Datapath**



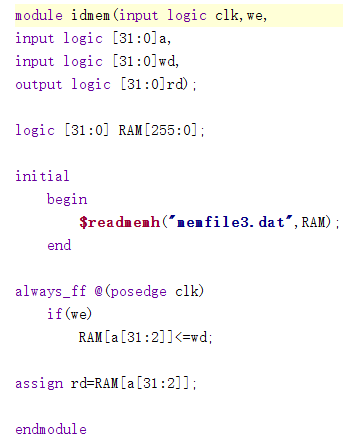
**Regfile**



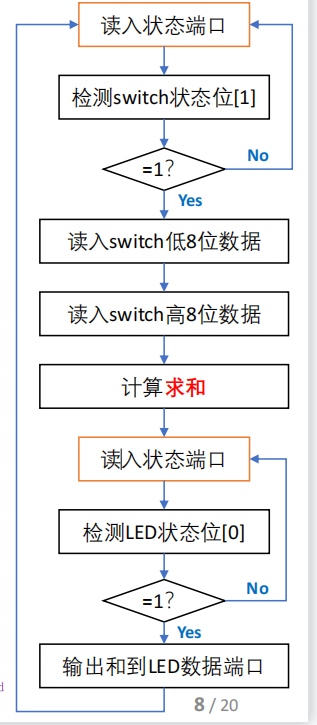
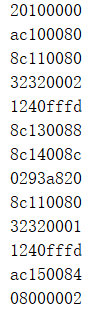
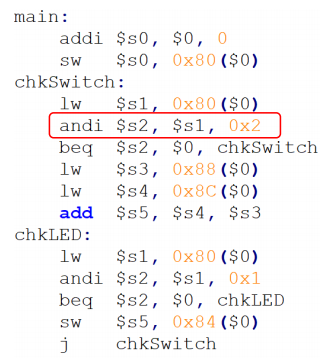
**Alu**



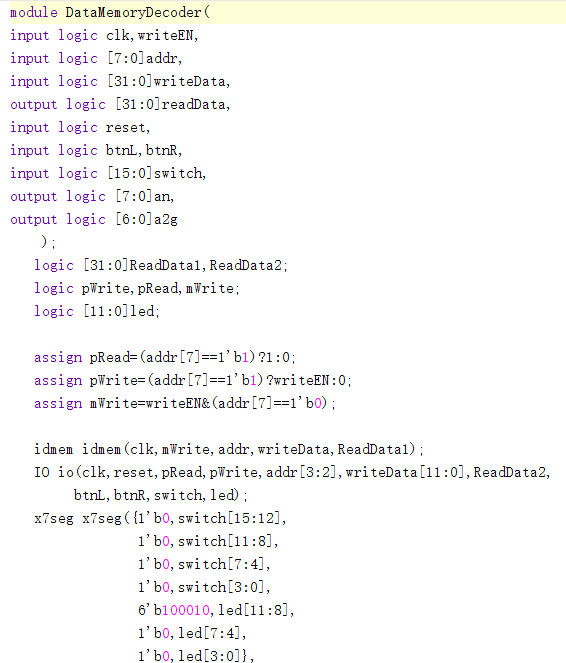
**Idmem**



**Memfile3.dat**

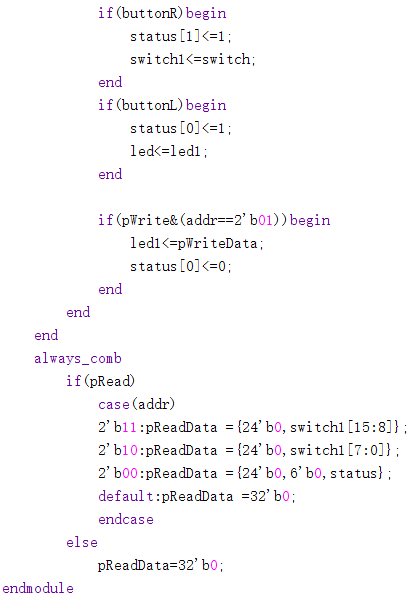
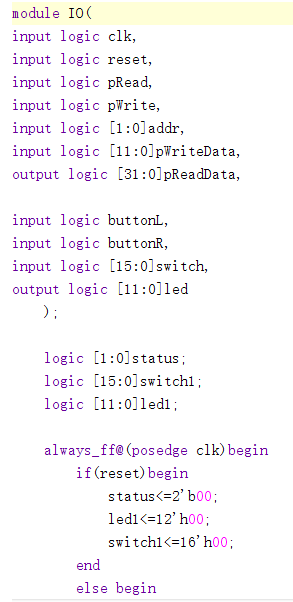


**Datamemorydecoder**

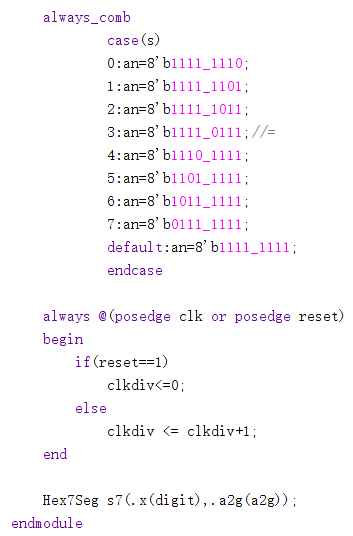
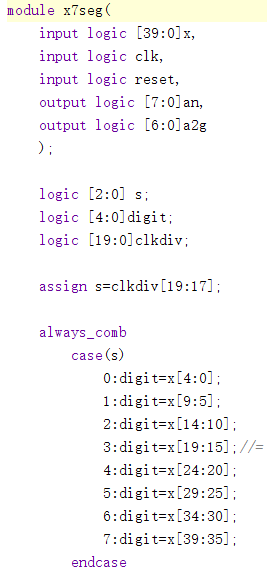


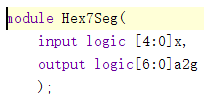


**IO**



**X7seg**

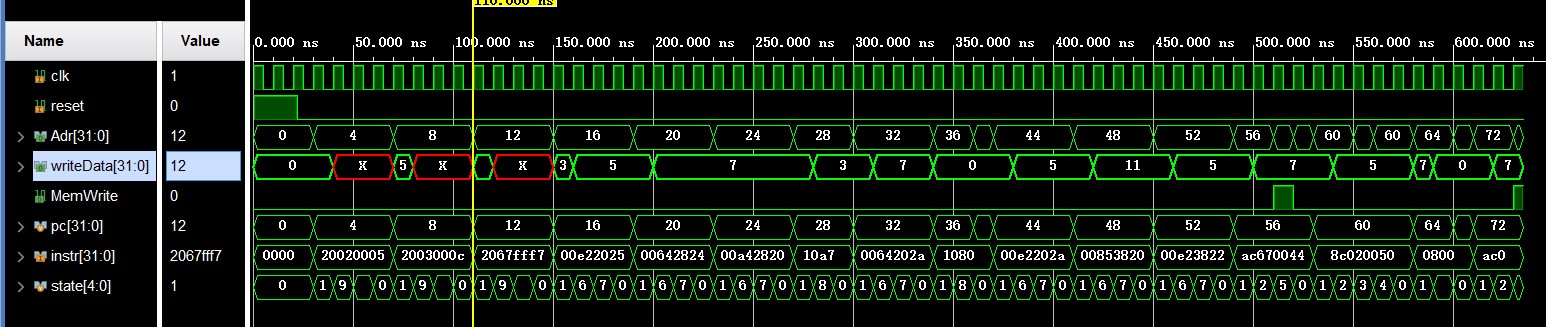




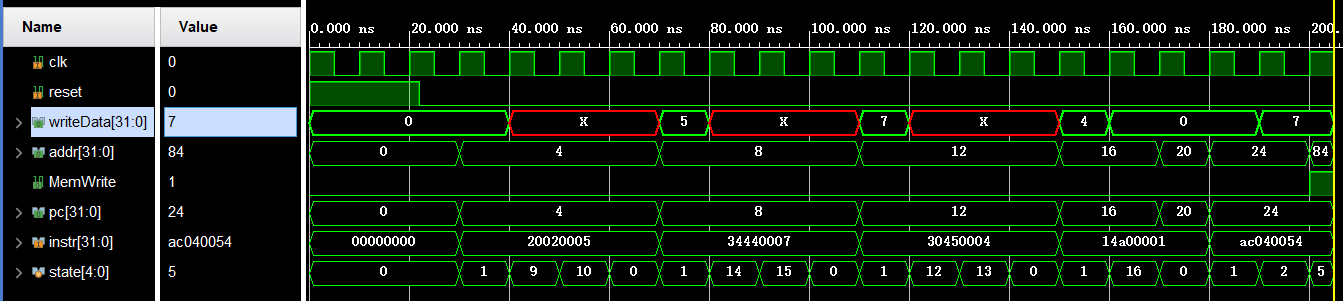


四、仿真截图

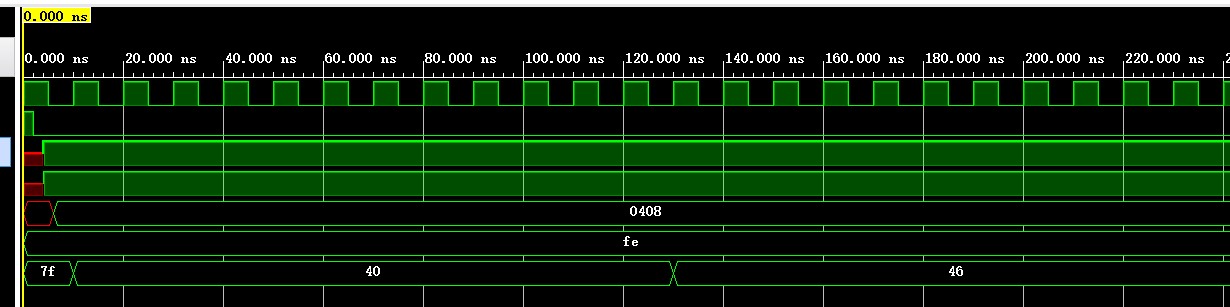
1.拓展前仿真测试



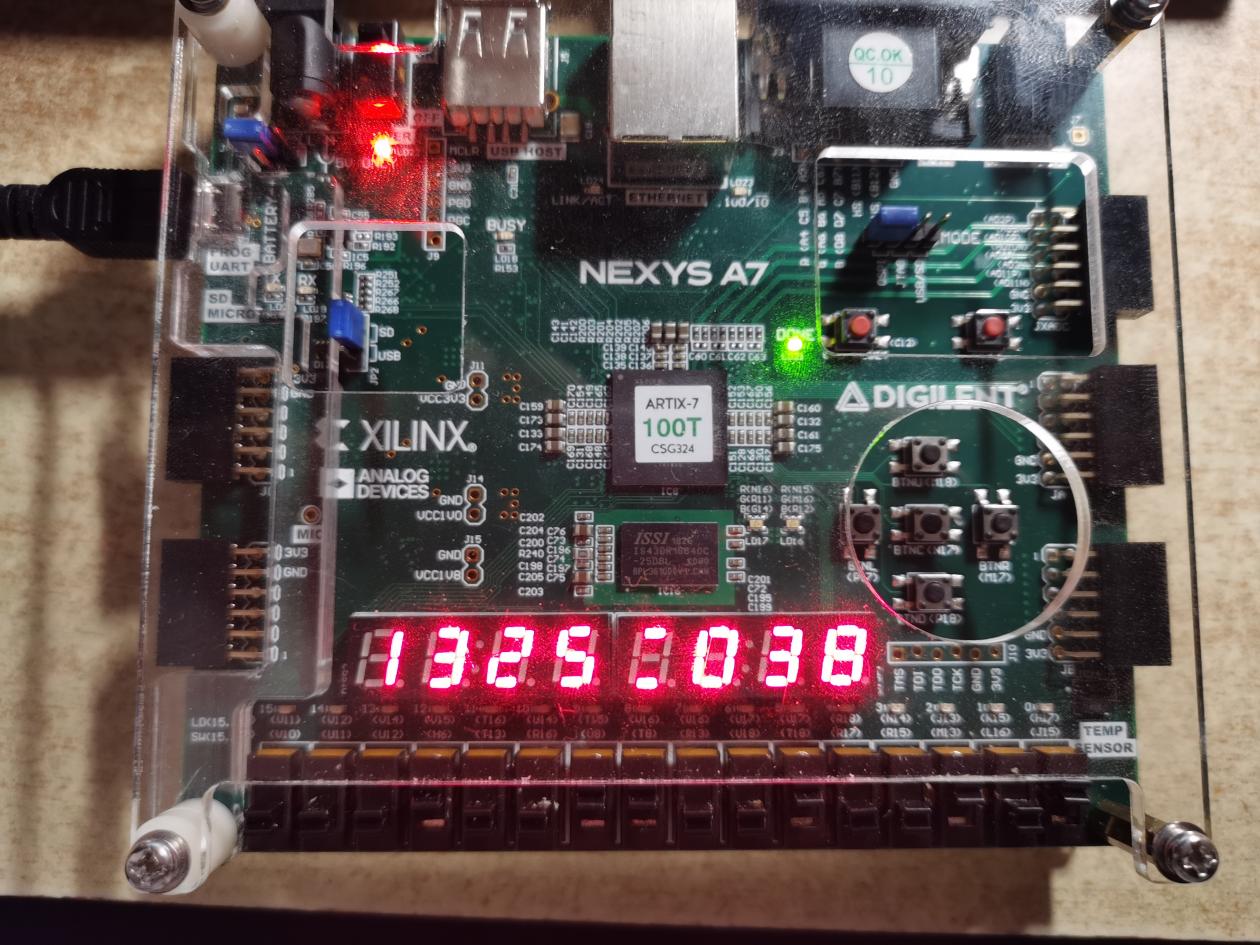
2.扩展后仿真测试

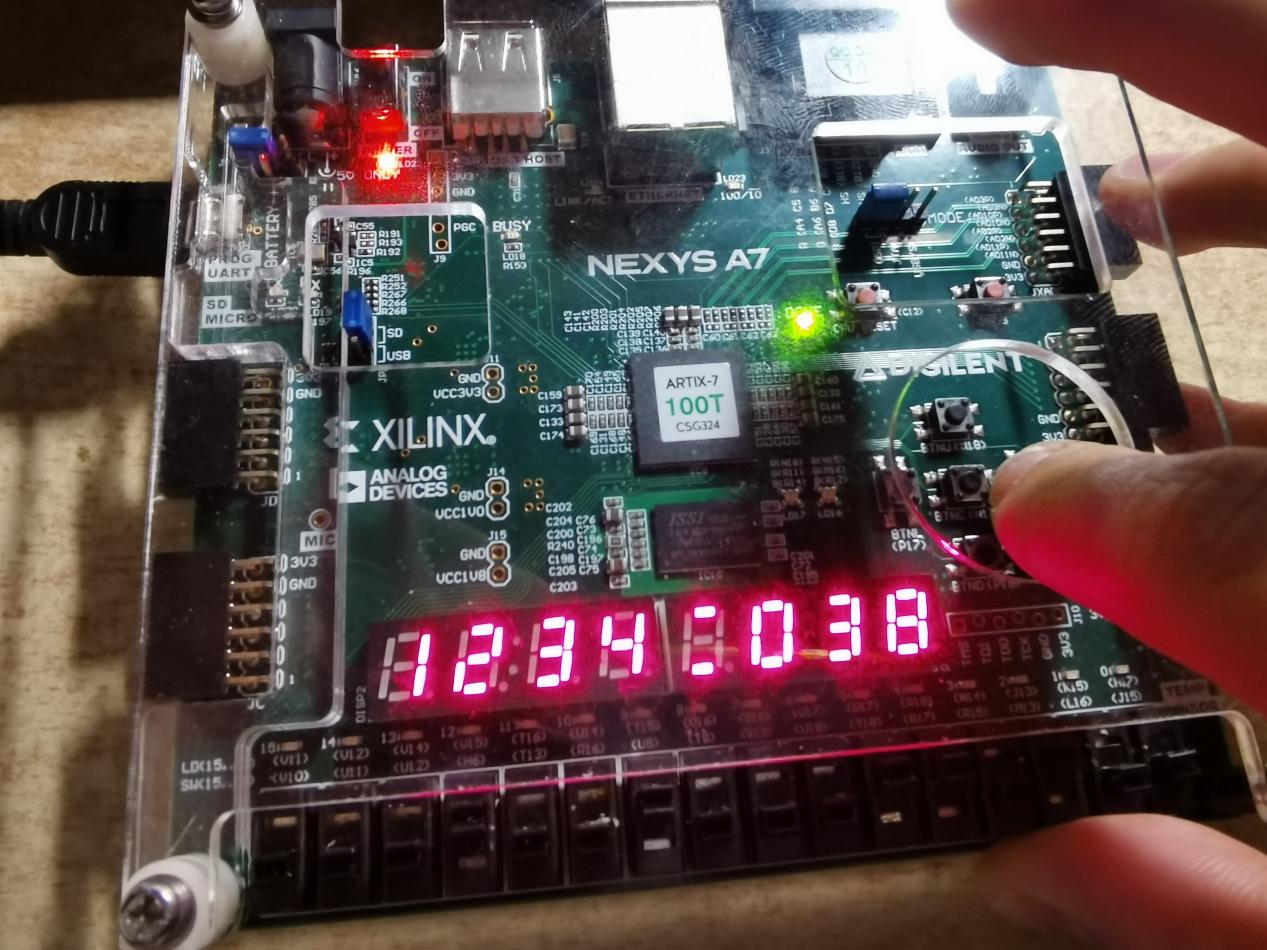


3.增加I/O接口仿真测试

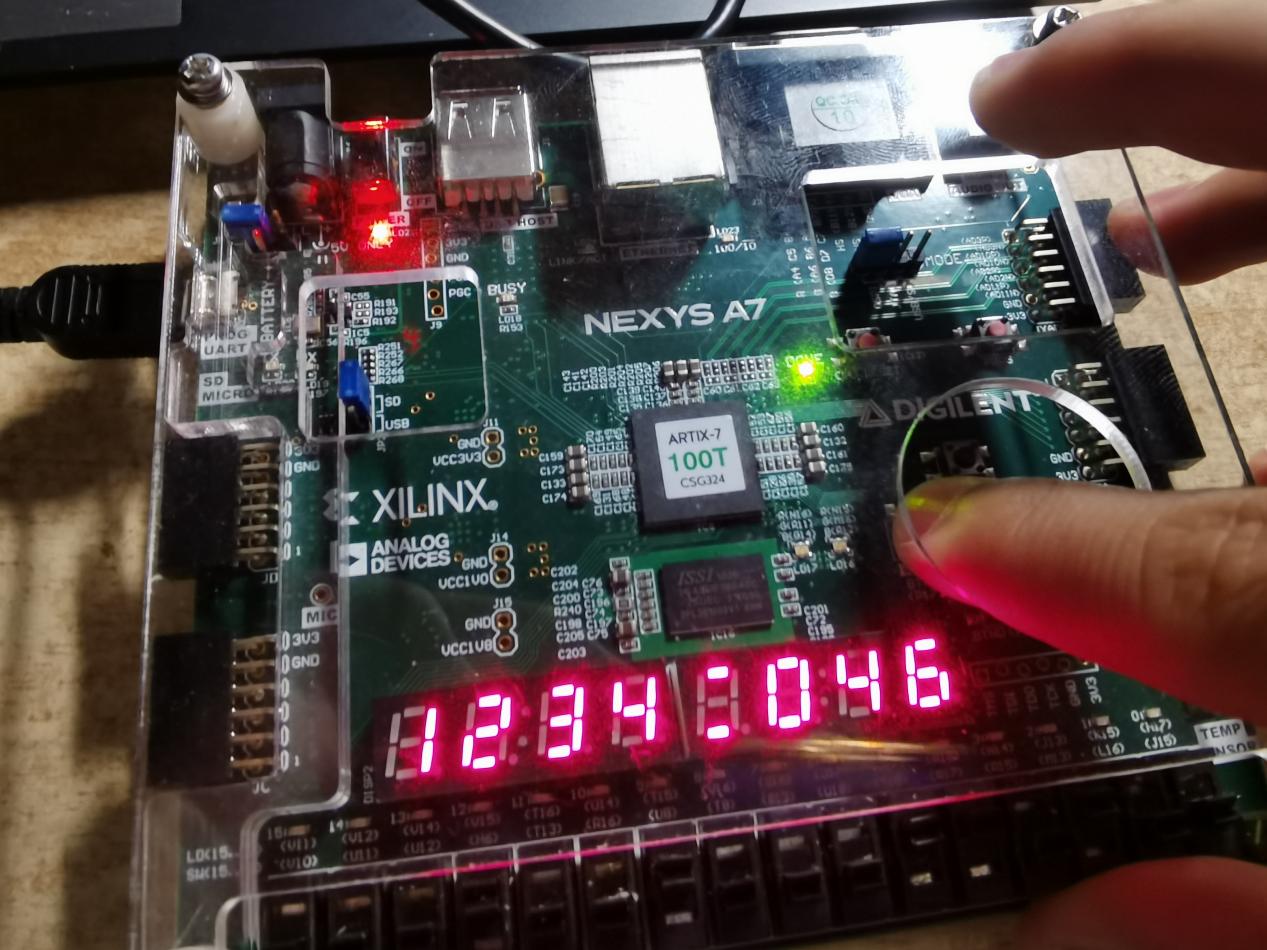


五、实验开发板照片

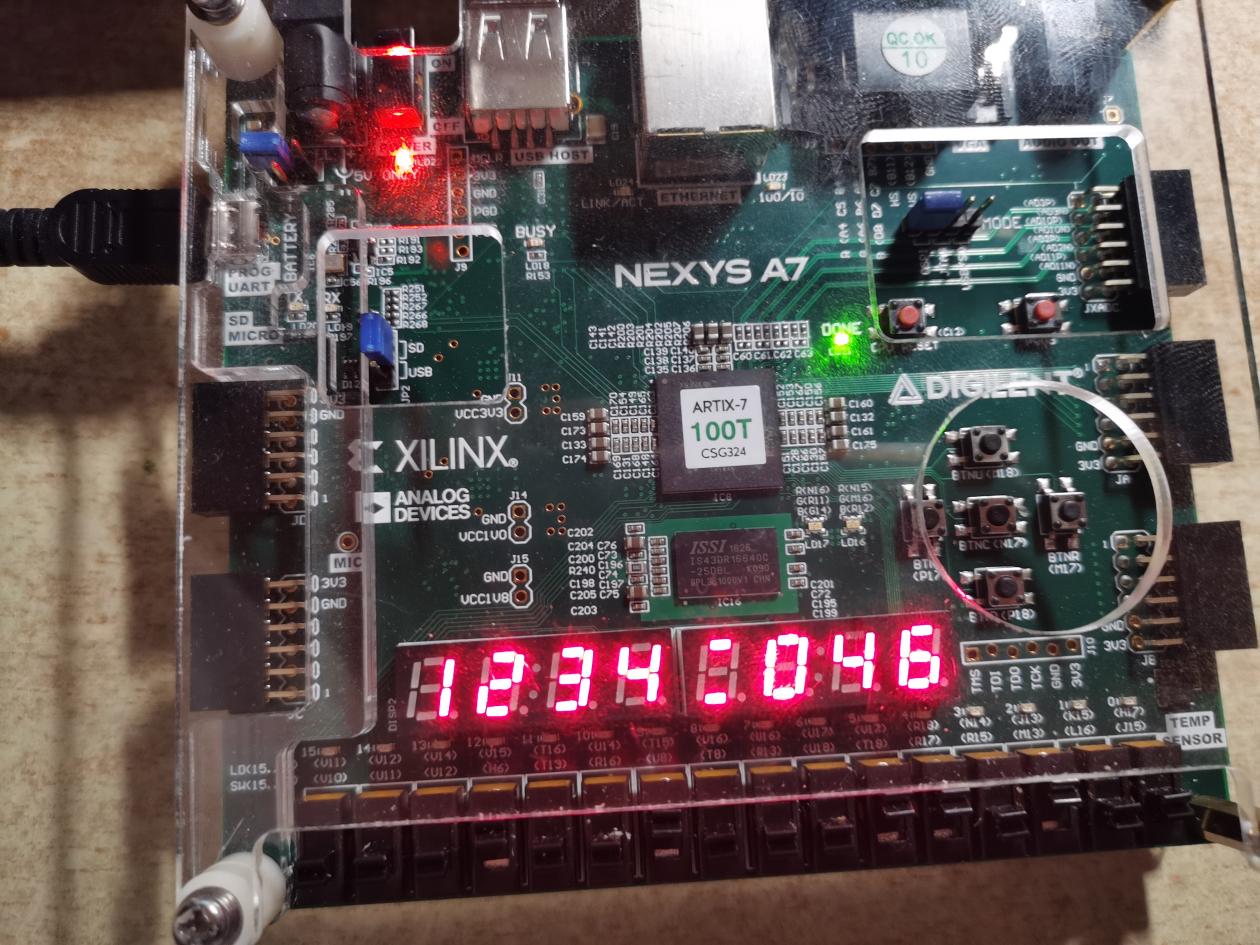




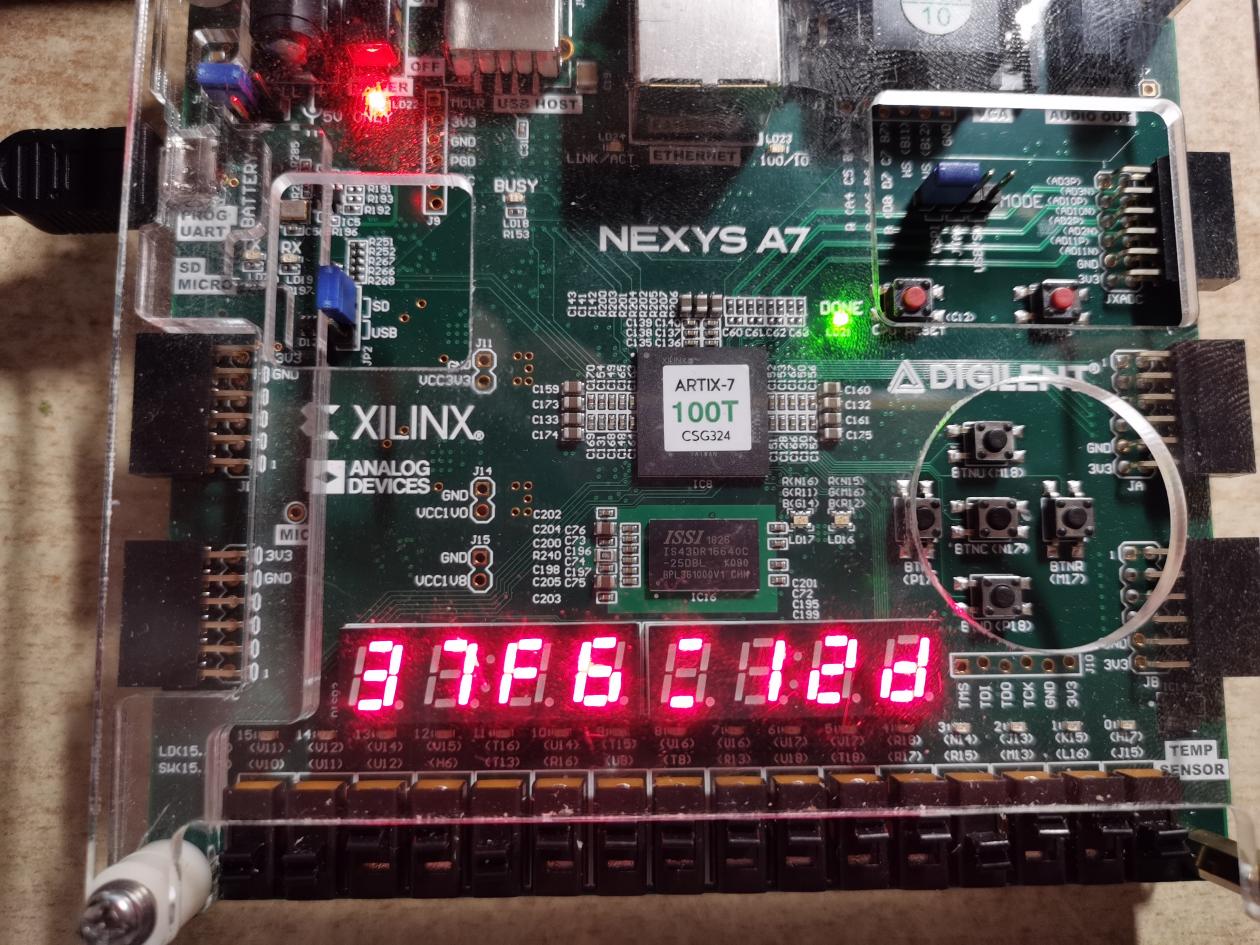
使用btnr读取数据

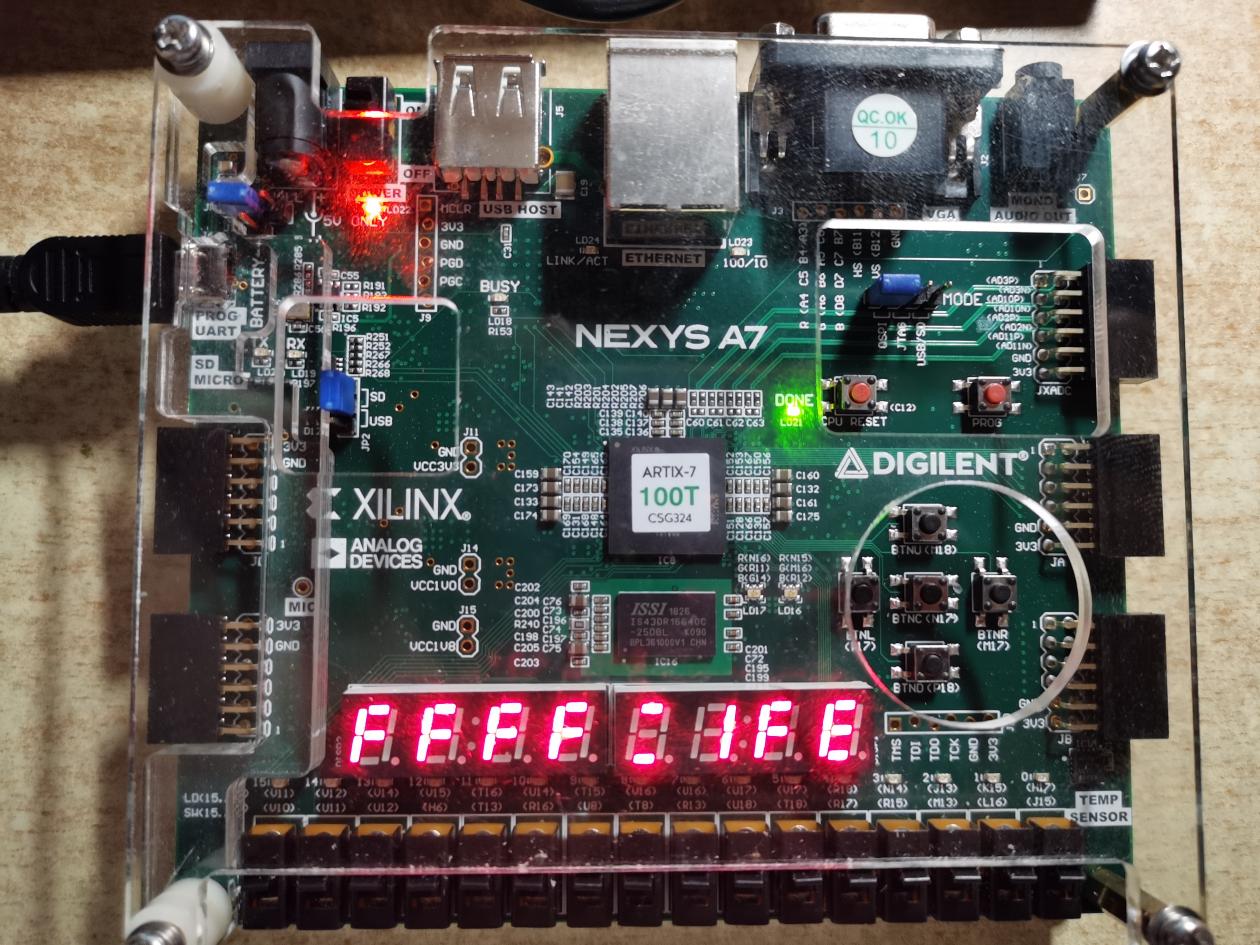


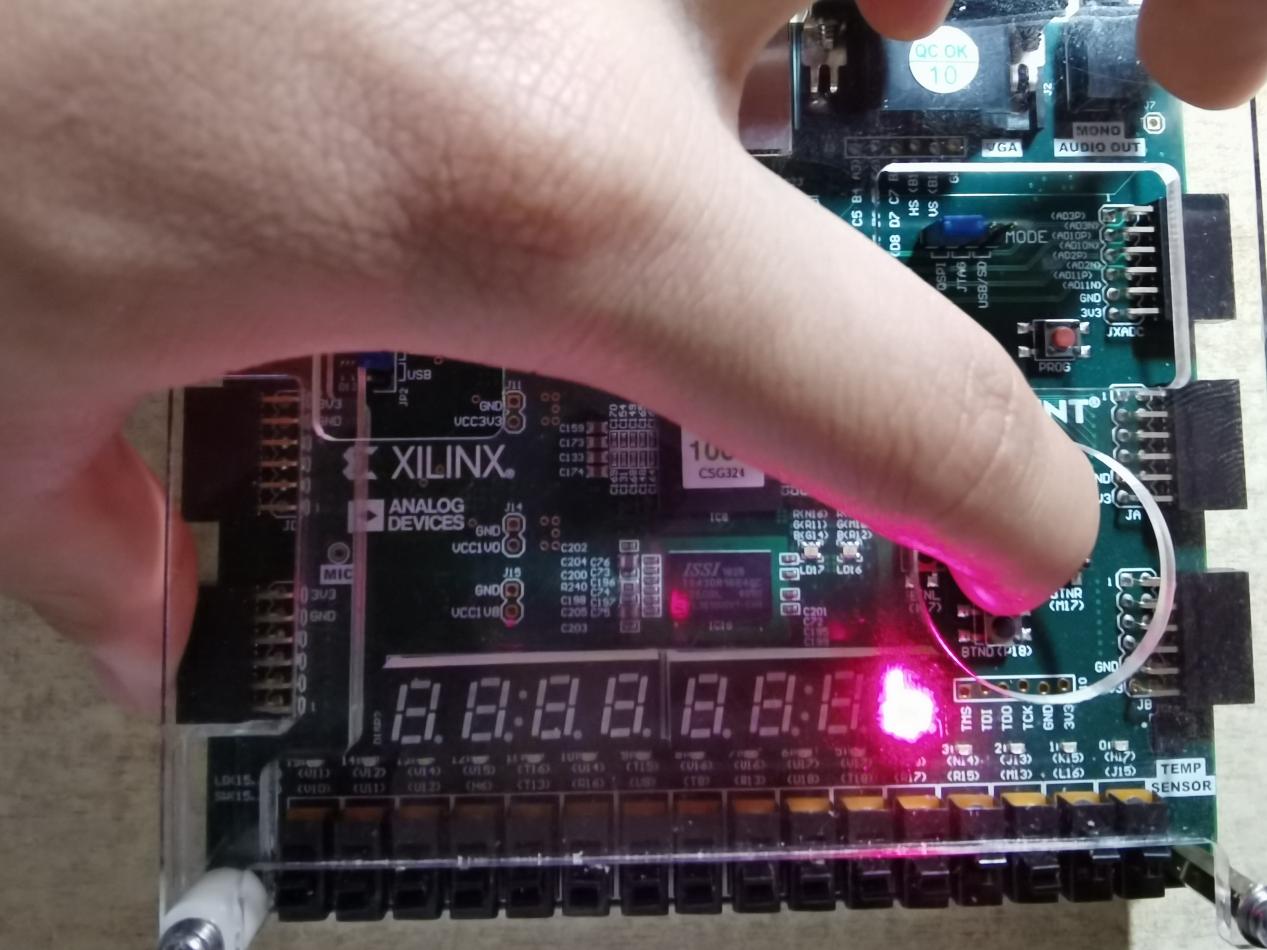
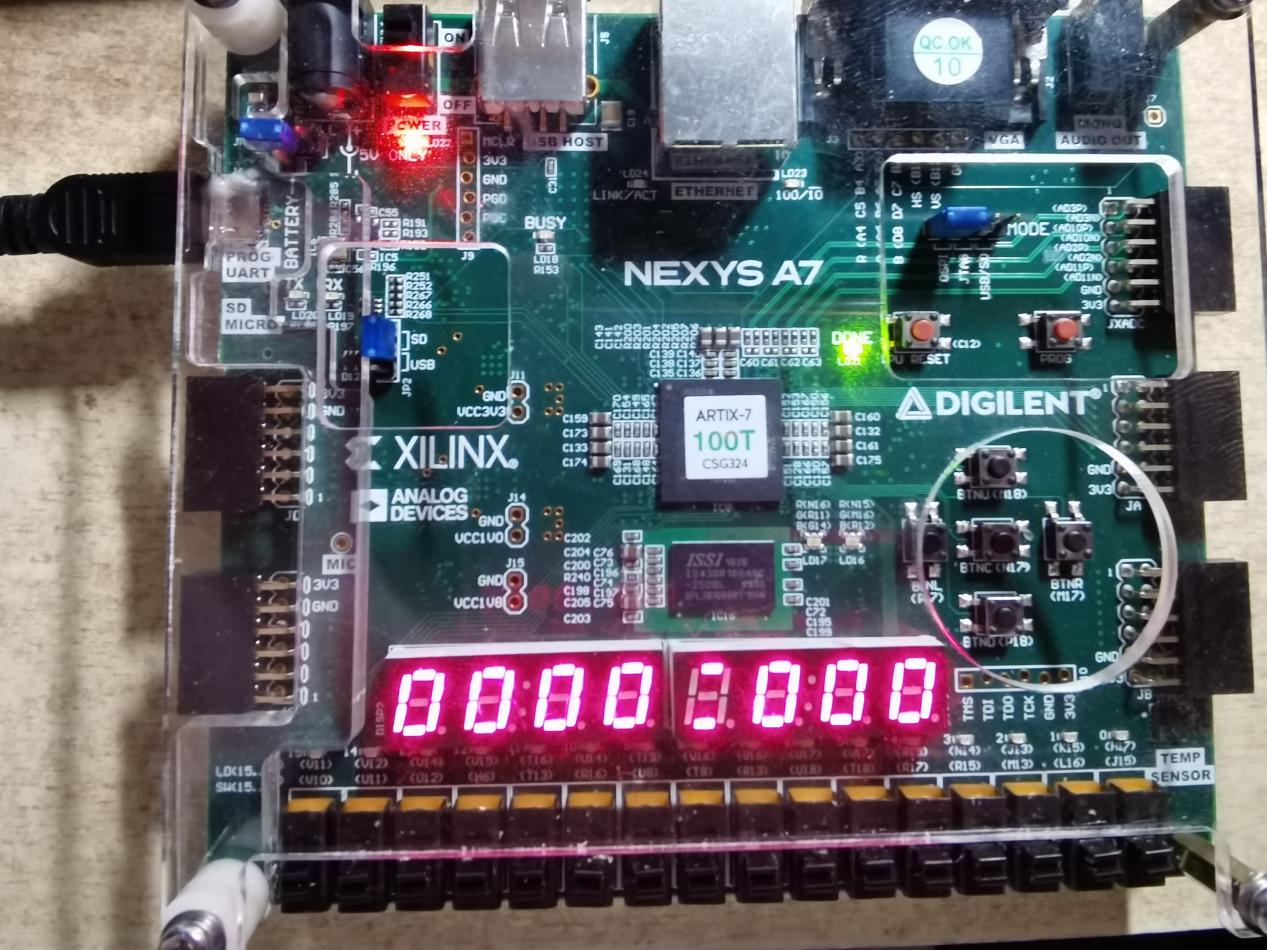
使用btnl计算并显示结果



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。

2.收获与体会

由于本次实验为设计一个完整的mips多周期处理器，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多周期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，不仅会加深我对多周期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！