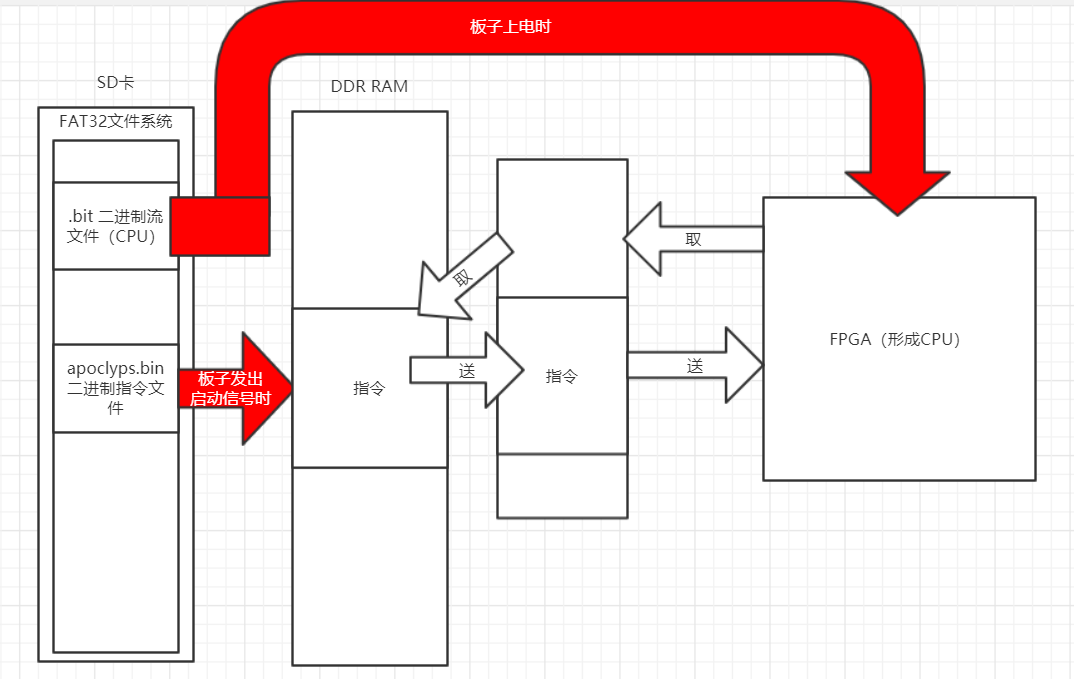
三级存储提升实验报告

1652270 冯舜

## 总体框架



## 要求

SD卡中存放流水线CPU的二进制流，以及用户程序，N4板上电自动完成如下的任务：

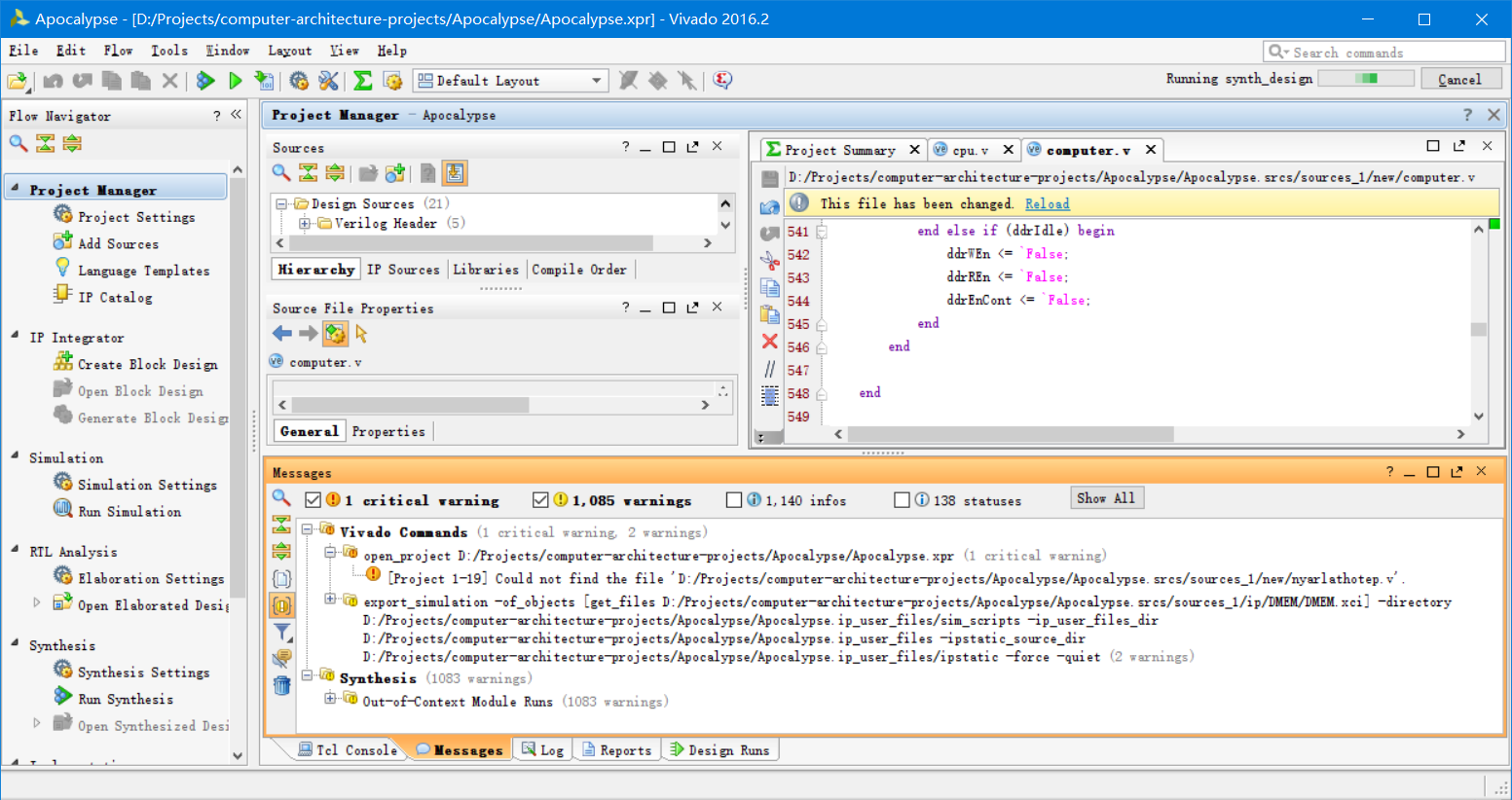
1） 采用跳线的方式，FPGA自动从SD卡中获取流水线CPU的二进制流,并运行该二进制流，使FPGA成为CPU。

2） CPU再按照三级存储的方式访问SDRAM，再由SDRAM从SD卡中把用户程序的目标代码调入到SDRAM，再由CPU把SDRAM中的用户程序目标代码调入到片内CACHE加以运行。

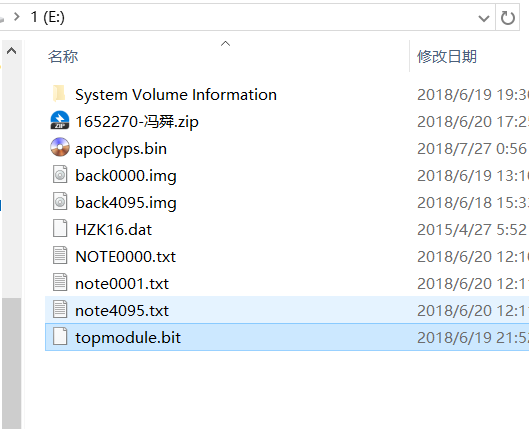
## 实现方法

### SD中存放流水线CPU的二进制流

使用Vivado软件综合、实现出CPU代码的二进制流文件topmodule.bit。



将其放入SD卡（FAT32文件系统）的根目录下。



将用户程序apoclyps.bin也放进去。

### 板子上电，自动加载bit文件为CPU

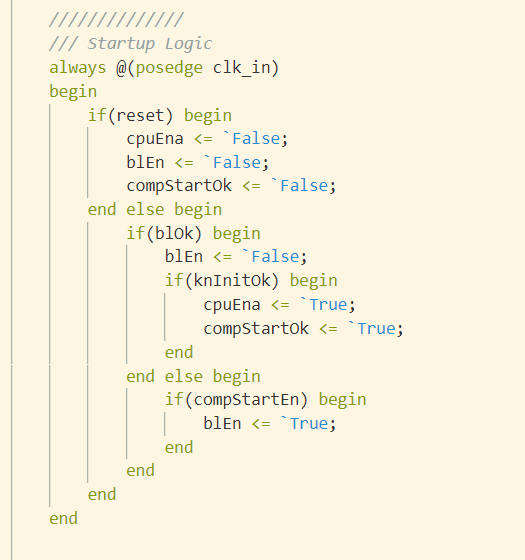
将板子的右上角JP1 Programming Mode跳线，从JTAG跳到USB/SD。

这样，板子在加电运行时，会自动加载SD卡根目录的.bit文件。

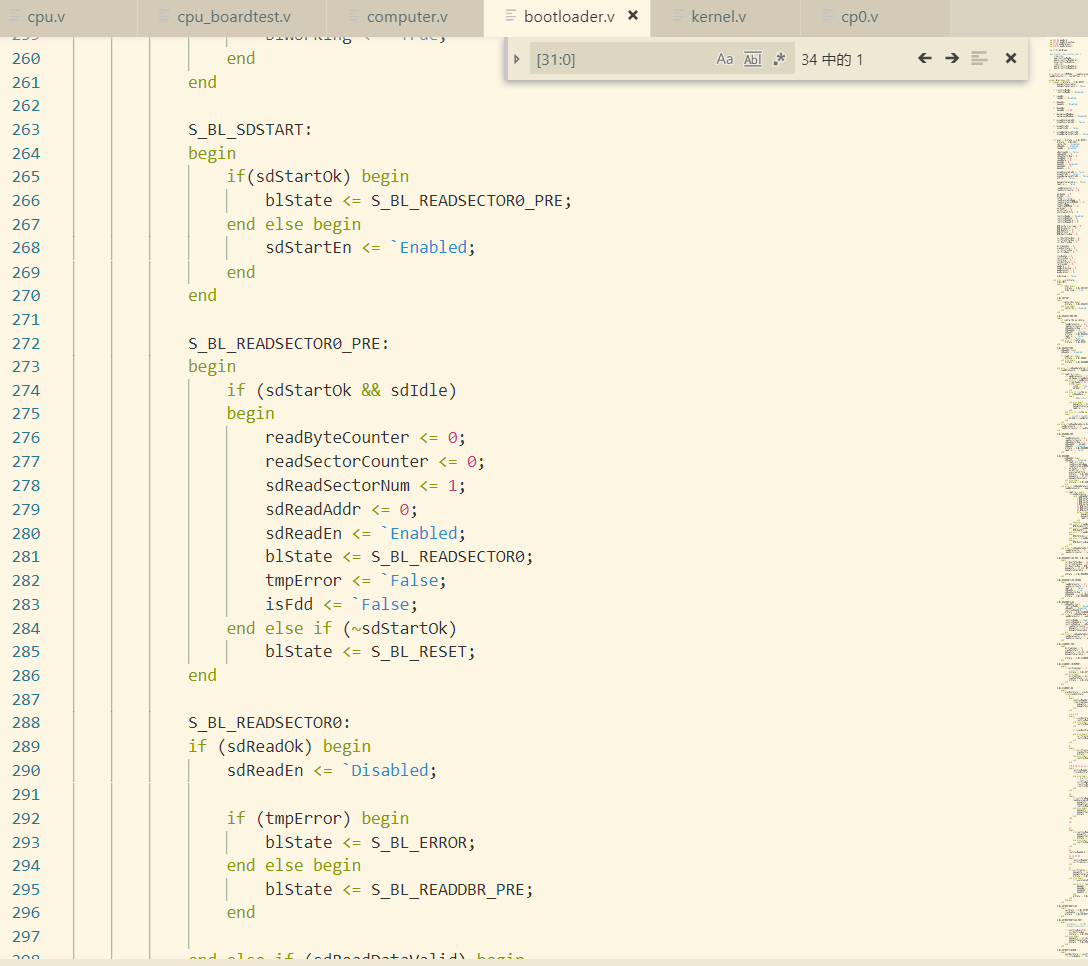
### CPU三级存储访问

#### 板子启动，加载用户程序

computer.v的Startup Logic块中，会在板子reset后发出一个blEn信号，指示bootloader程序从SD卡加载apoclyps.bin。

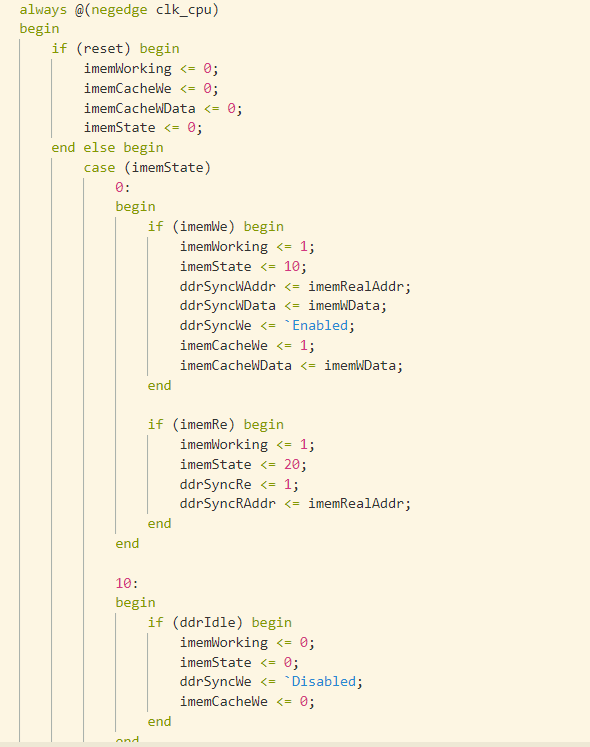


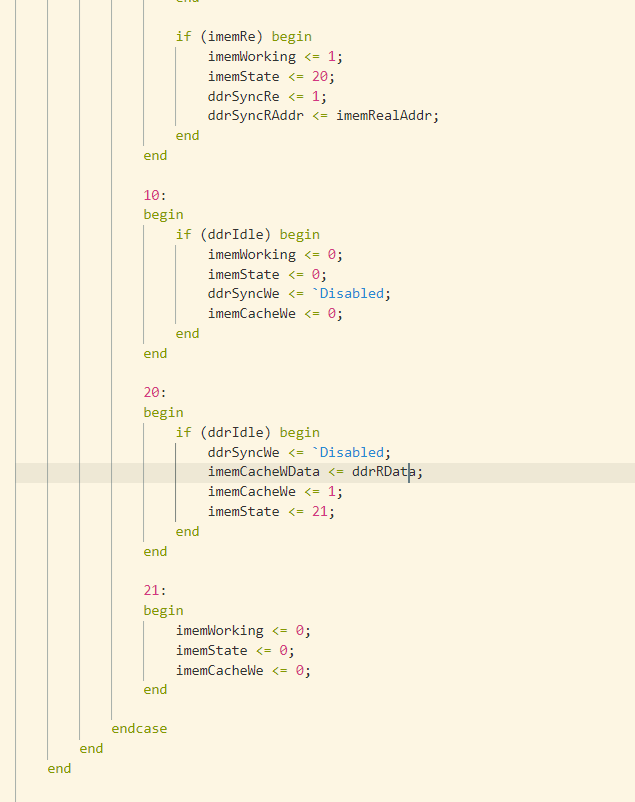
bootloader是一个大状态机，处理了SD卡的文件系统逻辑，再操控sd\_controller.v模块，完成SD卡根目录下apoclyps.bin 文件的读入。由于代码复杂，只展示一小部分。



#### CPU访问指令存储

CPU访问指令存储时，会触发指令存储逻辑电路。这个电路会使CPU挂起，并自己运行一个状态机，从DDR SDRAM获取数据，存入到Block RAM Cache中。

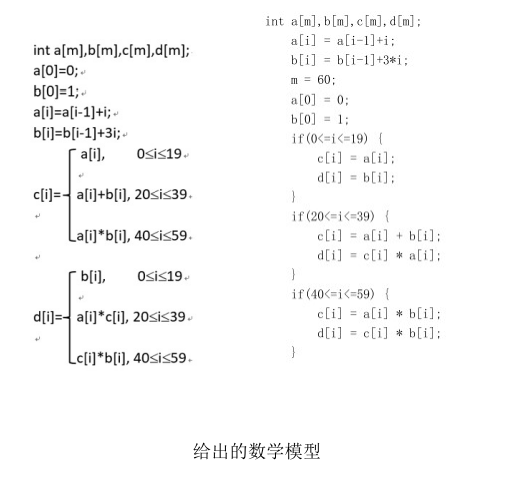


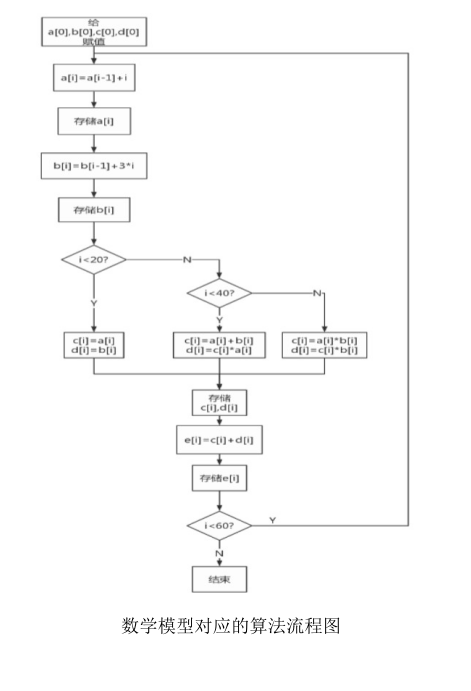


这一系列操作完成后，指令存储取消对CPU的挂起（IMEMWORKING=0），并在输出端给出仿存结果，供CPU读入。

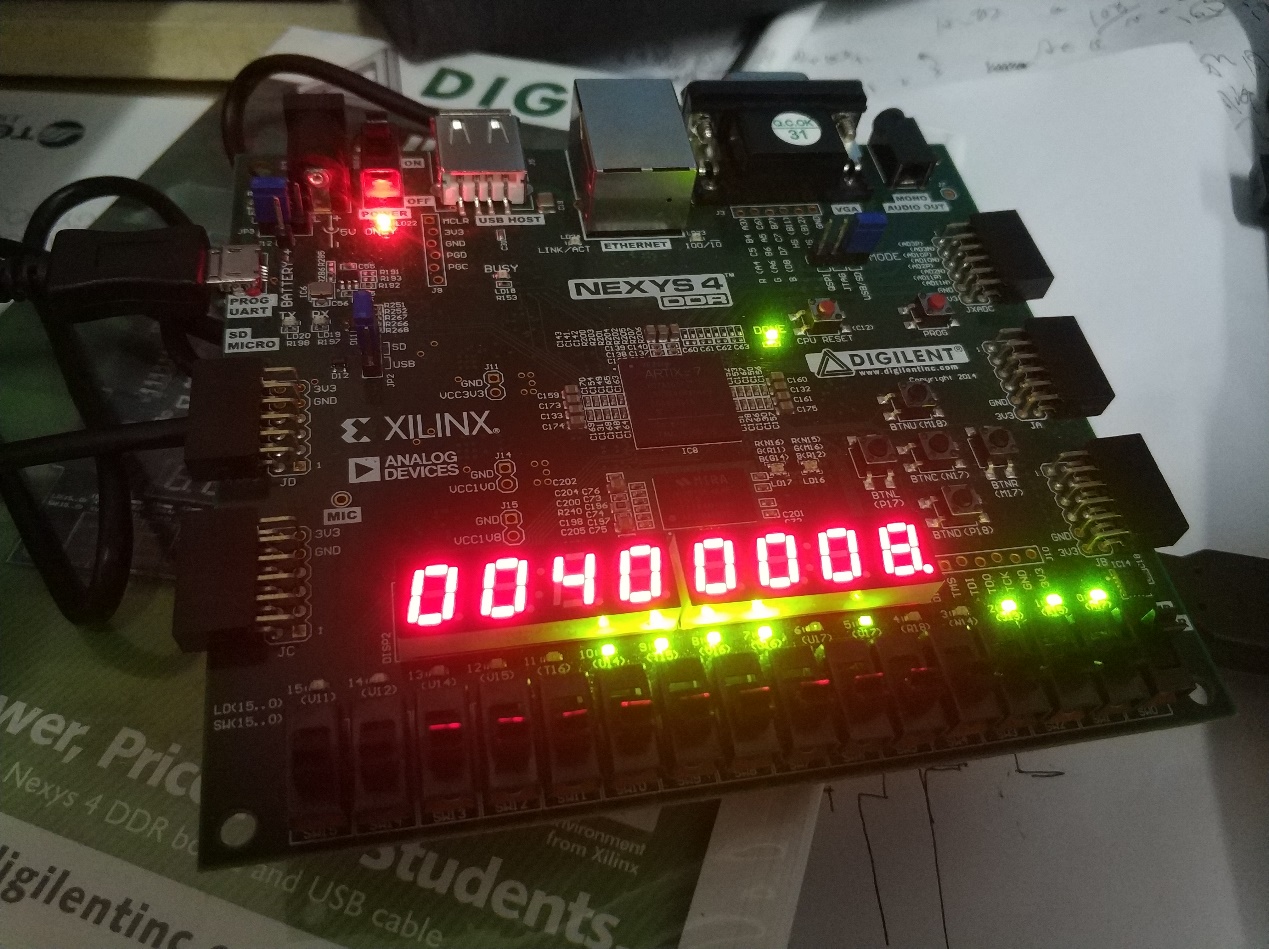
### 实际运行验证

实际使用的程序是本学期的静/动态流水线测试模型：





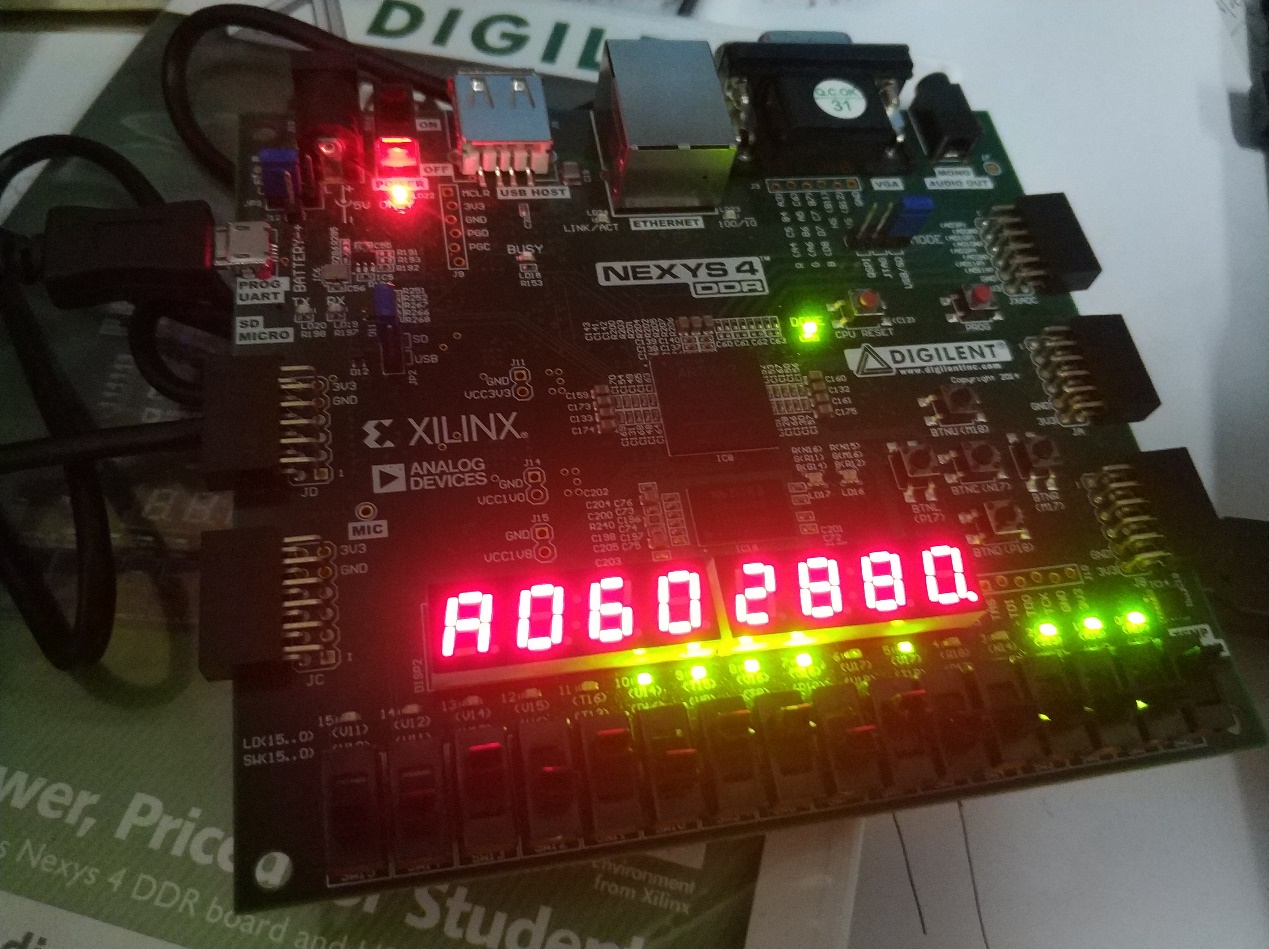
将apoclyps.bin和topmodule.bit放入SD卡，给板子上电，等待其出现12ABCDEF。按几下CPU/RESET键，再按“中”键开始程序的运行。运行完毕后，将十六个开关打为“关关关关关关关关，关关关关关关关开”，观测到PC值应当停留在00400008，说明程序运行到最后一条指令break后，跳到异常入口，并停留在0x00400008的指令。



将十五个开关从左到右依此拨为“关关开关开关开开，关开关关关关关开”（左 8位 00101011 为 E[59]所在字的序号 0x4ac/4 的低 8 位，右侧的开关表示在调试输出模

式下输出内存序号为 0x1XXXXXXXX（X 表示左侧开关对应的低八位）的字的值）

，七段数码管显示 E[59]的值 A0602880。验证了数学模型，CPU 制作正确。



### 源程序代码

这个代码是对上述数学模型的计算。

|  |
| --- |
| .data  A:.space 240  B:.space 240  C:.space 240  D:.space 240  E:.space 240    .text  j main  nop  exc:  j exc  nop  main:  addi $2,$0,0 #a[i]  addi $3,$0,1 #b[i]  addi $4,$0,0 #c[i]  addi $13,$0,0 #d[i]  addi $5,$0,4 #counter  addi $6,$0,0 #a[i-1]  addi $7,$0,1 #b[i-1]  addi $10,$0,0 #flag for i<20 || i<40  addi $11,$0,240 #sum counts  addi $14,$0,3  addi $30,$0,0  # 把 0 1 0 0 ($2,...,$13) 分别存入 A B C D  lui $27,0x0000  addu $27,$27,$0  sw $2,A($27)  lui $27,0x0000  addu $27,$27,$0  sw $3,B($27)  lui $27,0x0000  addu $27,$27,$0  sw $2,C($27)  lui $27,0x0000  addu $27,$27,$0  sw $3,D($27)  # 循环  loop:  ## $5(4) 除以 4 (=i) 存入 $12  srl $12,$5,2  # $6 加 i. 自此, $6 就是 a[i] 而不是 a[i-1] 了  add $6,$6,$12  # 把 a[i] 的内容存入 A[i] 中  lui $27,0x0000  addu $27,$27,$5  sw $6,A($27)    # $14 (3) 乘以 $5/4 (i) ( = 3i )  mul $15,$14,$12  # 把 $7 (b[i-1]) 的内容加上 3i, 存入 B[i]. 自此, $7 就是 b[i] 而不是 b[i-1] 了  add $7,$7,$15  lui $27,0x0000  addu $27,$27,$5  sw $7,B($27)    # $5 是否小于 80 (i 是否小于 20)? 记入 $10  slti $10,$5,80  # 若不是, 跳转  bne $10,1,c1  # (0<=i<=19)  # 把 $6 的内容存入 C[i] 中 (c[i] = a[i])  lui $27,0x0000  addu $27,$27,$5  sw $6,C($27)  # 把 $7 的内容存入 D[i] 中 (d[i] = b[i])  lui $27,0x0000  addu $27,$27,$5  sw $7,D($27)  addi $15,$6,0 # $15 $16 分别赋值为 c[i] d[i]  addi $16,$7,0  j endc  c1: # (20<=i<=39)  # i 是否小于 40 ？ 若不是，跳转到 c2  slti $10,$5,160  addi $27,$0,1  bne $10,$27,c2  # C[i] = a[i] + b[i]  add $15,$6,$7  lui $27,0x0000  addu $27,$27,$5  sw $15,C($27)    # D[i] = a[i] \* b[i]  mul $16, $15,$6  lui $27,0x0000  addu $27,$27,$5  sw $16,D($27)    j endc  c2: # (i>=40)  # C[i] = a[i] \* b[i]  mul $15,$6,$7  lui $27,0x0000  addu $27,$27,$5  sw $15,C($27)    # D[i] = c[i] \* b[i]  mul $16,$15,$7  lui $27,0x0000  addu $27,$27,$5  sw $16,D($27)    endc:  add $28,$15,$16 #$28 = c[i] + d[i]  lui $27,0x0000  addu $27,$27,$5  sw $28,E($27) # 将 c[i] + d[i] 存入 E[i]  addi $5,$5,4 # i = i + 1  bne $5,$11,loop # i = 60 不跳转  break  # 最后 E[i] = c[i] + d[i], 可通过验证 E[59] 的正确性来验证 c[i] 和 d[i] 正确性 |