# 一、示例机的设计和开发板上验证

## (一)、数据通路

CPU具有以下寄存器：

* 2个8位通用寄存器R0[7：0]和R1[7：0]。
* 16位的地址寄存器 AR[15：0]
* 16位的程序计数器 PC[15：0]
* 8位的数据寄存器 DR[7：0]
* 8位的指令寄存器 IR[7：0]
* 8位的数据暂存器 TR[7：0]，用于存储地址的低8位
* 8位的数据暂存器 X[7：0]，用于暂存ALU输入端的数据
* 1位的“0”标志寄存器Z，用于标志运算结果。如果运算结果为0，则Z=1；否则Z=0。

数据通路如图1所示。

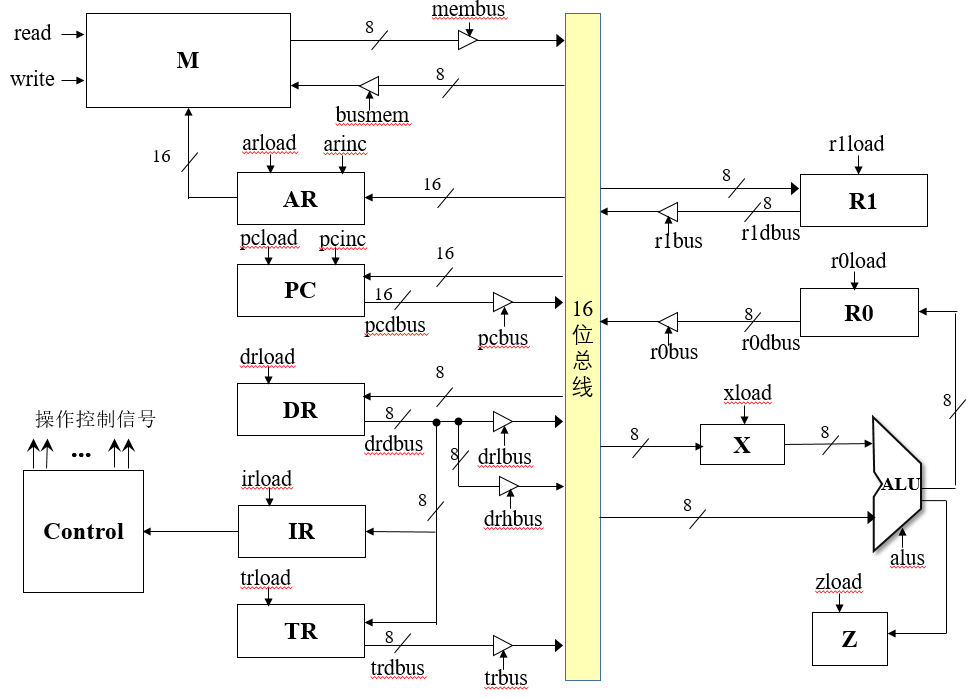


图1数据通路

说明：

（1）AR、PC是16位寄存器，直接连接到16位总线，其余8位寄存器连接到16位总线的低8位。

（2）由于IR仅仅从DR接受数据，所以建立一条DR到IR的直接的数据路径，而去掉总线到IR的连接。

（3）TR仅仅从DR接受数据，所以建立一条DR到TR的直接数据路径，去掉总线到TR的输入，保留TR到总线的输出，因为其中的内容要经过总线送到AR或者PC。

（4）如果是转移指令的地址或者是读写存储器的地址送到总线，由于地址是16位的，而通用寄存器都是8位的，所以地址存放在DR和TR中（DR提供高8位，TR提供低8位）。需要DR和TR同时把其中的内容放到总线上形成16位的地址，必须是DR为高8位TR为低8位，解决的办法就是把DR的数据同时连接到总线的高8位和低8位上，然后增加控制信号drhbus和drlbus。

（5）存储器M的输出是CPU的输入，CPU的输出是存储器M的输入。

数据通路中的控制信号说明如表1所示：（注：pcload类似于课上讲的pcin信号，pcbus类似于课上讲的pcout信号）

表1 控制信号

|  |  |  |
| --- | --- | --- |
| **序号** | **控制信号** | **说明** |
| 1 | arload | 将总线上的内容送入ar |
| 2 | arinc | ar内容自增 |
| 3 | pcload | 将总线上的内容送入pc |
| 4 | pcinc | pc内容自增 |
| 5 | pcbus | 打开三态门，pc中的内容送总线 |
| 6 | drload | 将总线上的内容送入dr |
| 7 | drhbus | 打开三态门，dr中的内容送总线的高8位 |
| 8 | drlbus | 打开三态门，dr中的内容送总线的低8位 |
| 9 | irload | 将总线上的内容送入ir |
| 10 | trload | 将总线上的内容送入tr |
| 11 | trbus | 打开三态门，tr中的内容送总线 |
| 12 | r1load | 将总线上的内容送入r1 |
| 13 | r1bus | 打开三态门，r1中的内容送总线 |
| 14 | r0load | 将总线上的内容送入r0 |
| 15 | r0bus | 打开三态门，r0中的内容送总线 |
| 16 | xload | 将总线上的内容送入x |
| 17 | alus | 选择alu运算的控制信号 |
| 18 | zload | 将alu的运算结果送入z（运算结果=0，送入1，否则送入0） |
| 19 | read | 读存储器 |
| 20 | write | 写存储器 |
| 21 | membus | 打开三态门，存储器中的内容送总线 |
| 22 | busmem | 打开三态门，总线上的内容送存储器 |

## (二)、指令系统

指令系统示例如表2所示。

表2 指令系统示例

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 指令 | 格式 | 功能 |
| 1 | NOP | 00000000 | No Operation |
| 2 | ADD | 00010000 | R0<— R0+R1 |
| 3 | INC | 01100000 | R0<— R0+1 |
| 4 | MVR | 10100000 | R1<— R0 |
| 5 | LAD | 11100000 A | R0<— M[A] |
| 6 | JMP | 10110000 A | GOTO A |

说明：

（1）表2中包含单字指令和三字指令。

（2）单字指令都是必做指令，除了MVR指令外，目的操作数都是R0寄存器。操作码8位，高4位用于区分指令，低4位为0。

（3）三字指令中A为16位的地址。

## (三)、取指和译码

这个CPU的取指周期具有3个节拍，如表3所示：

表3 取指周期的数据流和控制流

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T0 | fetch1 | AR<—PC | pcbus、arload |
| T1 | fetch2 | DR<—M，PC<—PC+1 | read、membus、drload、pcinc |
| T2 | fetch3 | IR<—DR，AR<—PC | irload、pcbus、arload |

取指周期和各指令的设计如下（图3，这是表2中的指令，需要换成自己设计的指令）。取指周期是所有指令公有的。

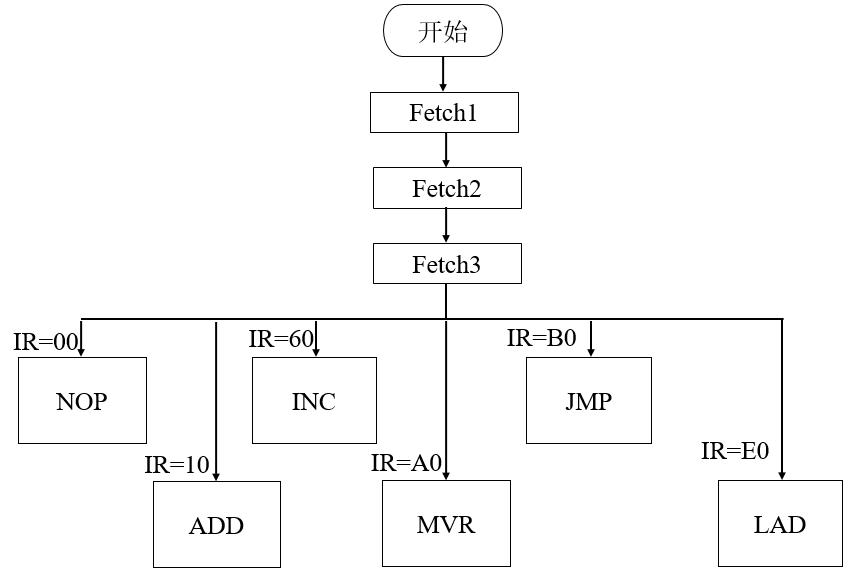


图3 各指令对应的取指和译码

## (四)、指令的执行（只列举表2指令的执行，其余指令自行设计）

### 1、NOP指令

NOP指令是最容易实现的一条指令，CPU什么都不作，而只是取下一条指令就可以了。设计一个空状态NOP1，什么都不做。

### 2、ADD指令

ADD指令在两个状态完成，功能：R0<—R0+R1。（注意如果设计条件转移指令，需要对标志寄存器Z的值进行设置）。指令执行状态有两个状态ADD1和ADD2，该指令执行周期的节拍、状态、数据流和控制流如表4所示：

表4 ADD指令执行周期的数据流和控制流

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | ADD1 | X<—R0 | r0bus、xload |
| T4 | ADD2 | R0<—X+R1 | r1bus、alus、r0load、zload（如果需要z标志寄存器，则有该控制信号） |

### 3、INC指令

INC指令在两个状态完成的，功能：R0<—R0+1。（注意如果设计条件转移指令，需要对标志寄存器Z的值进行设置）。指令执行状态有两个状态INC1和INC2，该指令执行周期的节拍、状态、数据流和控制流如表5所示：

表5 INC指令执行周期的数据流和控制流

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | INC1 | X<—R0 | r0bus、xload |
| T4 | INC2 | R0<—X+1 | alus、r0load、zload（如果需要z标志寄存器，则有该控制信号） |

### 4、MVR指令

该指令的功能是将R0的内容送到R1，只需要1个节拍即可。该指令执行周期的节拍、状态、数据流和控制流如表6所示：

表6 MVR指令执行周期的数据流和控制流

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | MVR1 | R1<—R0 | r0bus、r1load |

### 5、LAD指令

LAD指令包括三个部分：操作码、低位地址、高位地址。执行时必须从主存中取出地址然后再从相应的地址取数据到R0中。

在取指周期的fetch2中，存储地址A的低8位已经放入PC中，并在fetch3的时候load到AR中，所以此时CPU要做的事情就是：

* PC+1，以便取下一条指令（在这里还是A的高8位地址）
* AR+1，取A的高8位地址

即LAD1：DR<—M，PC<—PC+1，AR<—AR+1

接下来CPU可以取A的高8位地址了，同时它必须用数据暂存器TR暂存低8位地址，并且PC+1：

即LAD2：TR<—DR，DR<—M，PC<—PC+1

这时CPU得到了A的地址，可以进行从存储器读取数据的操作了，首先把地址复制到AR，然后把要取的数载入到DR，最后复制数据到R0中。

LAD3：AR<—DR，TR

LAD4：DR<—M

LAD5：R0<—DR

指令执行状态有5个状态LAD1——LAD5，该指令执行周期的节拍、状态、数据流和控制流如表7所示：

表7 LAD指令执行周期的数据流和控制流

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | LAD1 | DR<—M，PC<—PC+1，  AR<—AR+1 | read、membus、drload、pcinc、arinc |
| T4 | LAD2 | TR<—DR，DR<—M，  PC<—PC+1 | trload、read、membus、drload、pcinc |
| T5 | LAD3 | AR<—DR，TR | drhbus、trbus、arload |
| T6 | LAD4 | DR<—M | read、membus、drload |
| T7 | LAD5 | R0<—DR | drlbus、r0load、alus |

### 6、JMP指令

JMP指令执行时，CPU先按照LAD的取地址方式取出要跳转的地址，然后送PC。该指令执行周期的节拍、状态、数据流和控制流如表8所示：

表8 JMP指令执行周期的数据流和控制流

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | JMP1 | DR<—M，AR<—AR+1 | read、membus、drload、arinc |
| T4 | JMP 2 | TR<—DR，DR<—M | trload、read、membus、drload |
| T5 | JMP 3 | PC<—DR，TR | drhbus、trbus、pcload |

### 指令的状态图

这些指令的状态图如图4：（需要自行补充所设计指令的状态图）

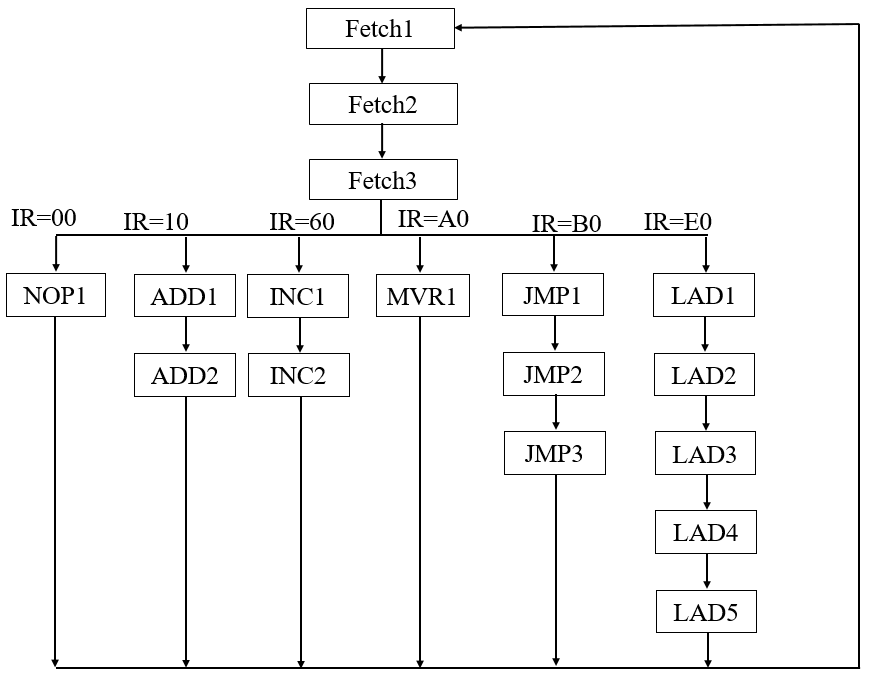


图4 指令状态图

## (五)、ALU的设计

根据数据通路可知：ALU的一个输入来自于暂存器X的输出，另一个输入来自于总线BUS，ALU的运算结果送入了R0。所以要设计ALU，我们首先来看那些指令修改了R0的值（即使用R0进行了运算）：（给出的典型指令中只有以下3条指令修改了R0，根据自行设计的具体指令进行添加）

ADD2：R0<—X+R1

INC2：R0<—X+1

LAD5：R0<—DR

根据ALU的两个输入和输出的对应关系可得出如下关系：

ADD2：R0<—X+BUS，（将总线上来的数据和X相加后的结果送入R0）

INC2：R0<—X+1，（将X自增后送入R0）

LAD5：R0<—BUS，（将总线上来的数据后送入R0）

由于在数据通路中，R0只是将输出接到了总线上，而总线上的数据不能直接送入R0中，只能借助于ALU做加0运算。因此

LAD5：R0<—BUS+0，（将总线上来的数据+0送入R0）

由此可知，以上3条指令都要在ALU中做运算。

提示：在必做指令中有9条指令都要在ALU中完成，因此选择运算的控制信号alus至少是4位的。alus的值需要在译码时根据指令赋值成不同的值！

## (六)、控制器的设计

指令译码的设计：因为操作码就存储在IR寄存器中，所以控制单元用IR寄存器的输出作为译码器的输入，由于指令格式为xxxx0000所以只需要译码高4位就可以了，NOR低4位可以用于使能译码器。

所有指令的前3个状态都是fetch1、fetch2和fetch3（这是取指周期的3个节拍），接下来的状态才是各指令执行周期的状态。

指令的执行状态要靠译码器和计数器两者组合才能得知。结构如图5：

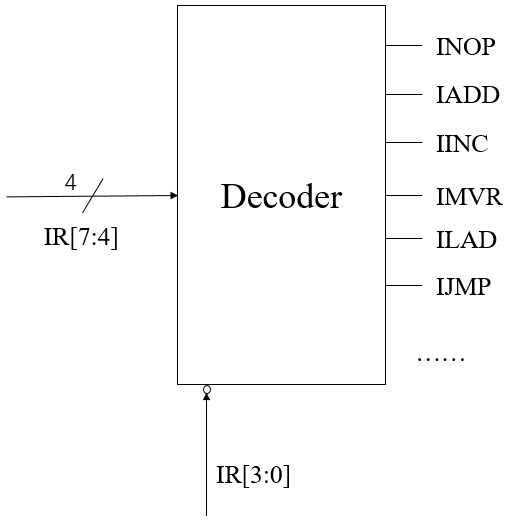
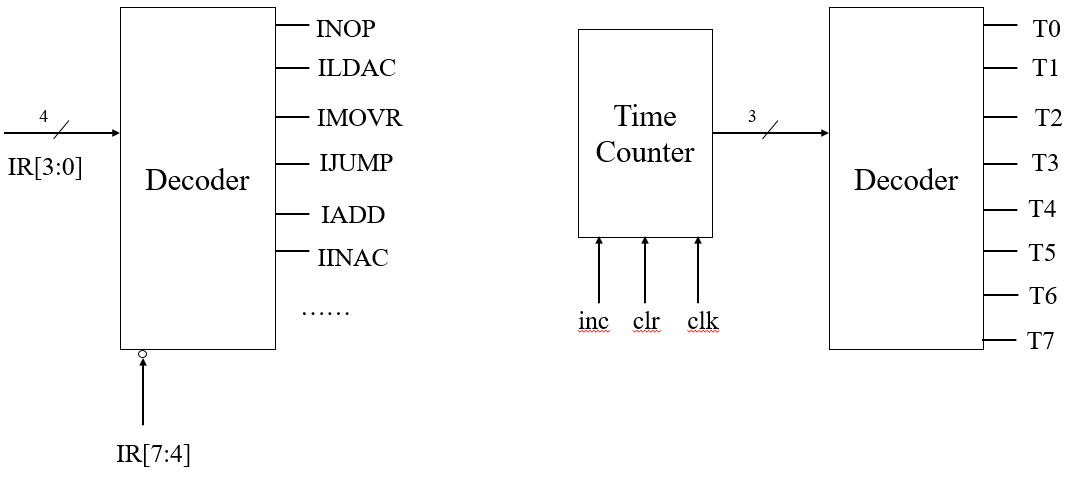
 

图5 译码器和计数器

图5中经过译码后的指令分别叫做INOP、IADD、IINC、IMVR、ILAD、IJMP。

控制器把两者的输出进行AND操作，来选择正确的指令执行。例如：LAD的执行周期如下：

LAD1=ILAD^T3

LAD2=ILAD^T4

LAD3=ILAD^T5

LAD4=ILAD^T6

LAD5=ILAD^T7

典型指令的状态及其表达式如表9所示：（根据自行设计的具体指令进行添加）

表9 典型指令的状态及其表达式

|  |  |  |
| --- | --- | --- |
| 机器周期 | 状态 | 表达式 |
| 取指周期 | fetch1 | T0 |
| fetch2 | T1 |
| fetch3 | T2 |
| NOP执行周期 | NOP1 | INOP^T3 |
| ADD执行周期 | ADD1 | IADD^T3 |
| ADD2 | IADD^T4 |
| INC执行周期 | INC1 | IINC^T3 |
| INC2 | IINC^T4 |
| MVR执行周期 | MVR1 | IMVR^T3 |
| LAD执行周期 | LAD1 | ILAD^T3 |
| LAD2 | ILAD^T4 |
| LAD3 | ILAD^T5 |
| LAD4 | ILAD^T6 |
| LAD5 | ILAD^T7 |
| JMP执行周期 | JMP1 | IJMP^T3 |
| JMP2 | IJMP^T4 |
| JMP3 | IJMP^T5 |

生成这些状态之后，还要生成计数器的CLR和INC信号。把每个执行周期的最后一个状态进行OR得到了CLR信号，再把CLR信号取反得到了INC信号。例如现在设计了6条指令，则clr = NOP1 || LAD5||MVR1||JMP3||ADD2||INC2（如果还有其他指令，则需要与其他指令的最后一个状态进行或运算），inc = ~clr

接下来需要产生数据路径中的控制信号。（还是以表2指令为例，将前面取指周期和指令的执行周期都集中在表10中）。

表10 取指周期和各指令执行周期的数据流和控制流

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 周期 | 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| 取  指  周  期 | T0 | fetch1 | AR<—PC | pcbus、arload |
| T1 | fetch2 | DR<—M，PC<—PC+1 | read、membus、drload、pcinc |
| T2 | fetch3 | IR<—DR，AR<—PC | irload、pcbus、arload |
| ADD执行周期 | T3 | ADD1 | X<—R0 | r0bus、xload |
| T4 | ADD2 | R0<—X+R1 | r1bus、alus、r0load、zload（如果需要z标志寄存器，则有该控制信号） |
| INC执行周期 | T3 | INC1 | X<—R0 | r0bus、xload |
| T4 | INC2 | R0<—X+1 | alus、r0load、zload（如果需要z标志寄存器，则有该控制信号） |
| MVR执行周期 | T3 | MVR1 | R1<—R0 | r0bus、r1load |
| LAD执行周期 | T3 | LAD1 | DR<—M，PC<—PC+1，  AR<—AR+1 | read、membus、drload、pcinc、arinc |
| T4 | LAD2 | TR<—DR，DR<—M，  PC<—PC+1 | trload、read、membus、drload、pcinc |
| T5 | LAD3 | AR<—DR，TR | drhbus、trbus、arload |
| T6 | LAD4 | DR<—M | read、membus、drload |
| T7 | LAD5 | R0<—DR | drlbus、r0load、alus |
| JMP执行周期 | T3 | JMP1 | DR<—M，AR<—AR+1 | read、membus、drload、arinc |
| T4 | JMP 2 | TR<—DR，DR<—M | trload、read、membus、drload |
| T5 | JMP 3 | PC<—DR，TR | drhbus、trbus、pcload |

接下来产生数据路径中的控制信号。需要找到控制信号所在的所有状态，将这些状态进行或运算。例如pcbus，该控制信号出现在fetch1和fetch3，则pcbus的逻辑表达式如下：

pcbus = fetch1 || fetch3

再举个例子，控制信号arload出现在fetch1、fetch3和LAD3，则其逻辑表达式为：

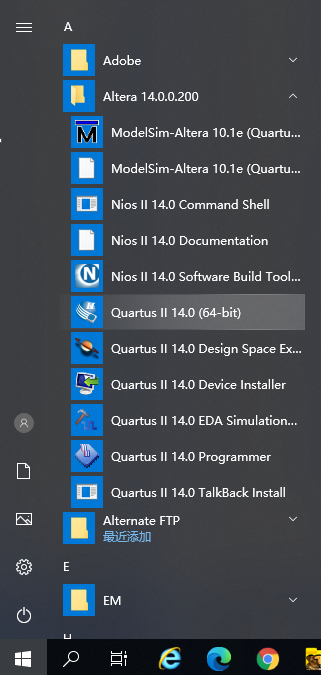
arload = fetch1 || fetch3|| LAD3

其他控制信号的生成采用类似的方法（注意：alus是在译码时根据指令不同赋值为不同的值！与其他控制信号的逻辑表达式生成方式不同！），将所有控制信号的逻辑表达式写出来（注意，表10中是表2的指令，需要包含自行设计的指令！）

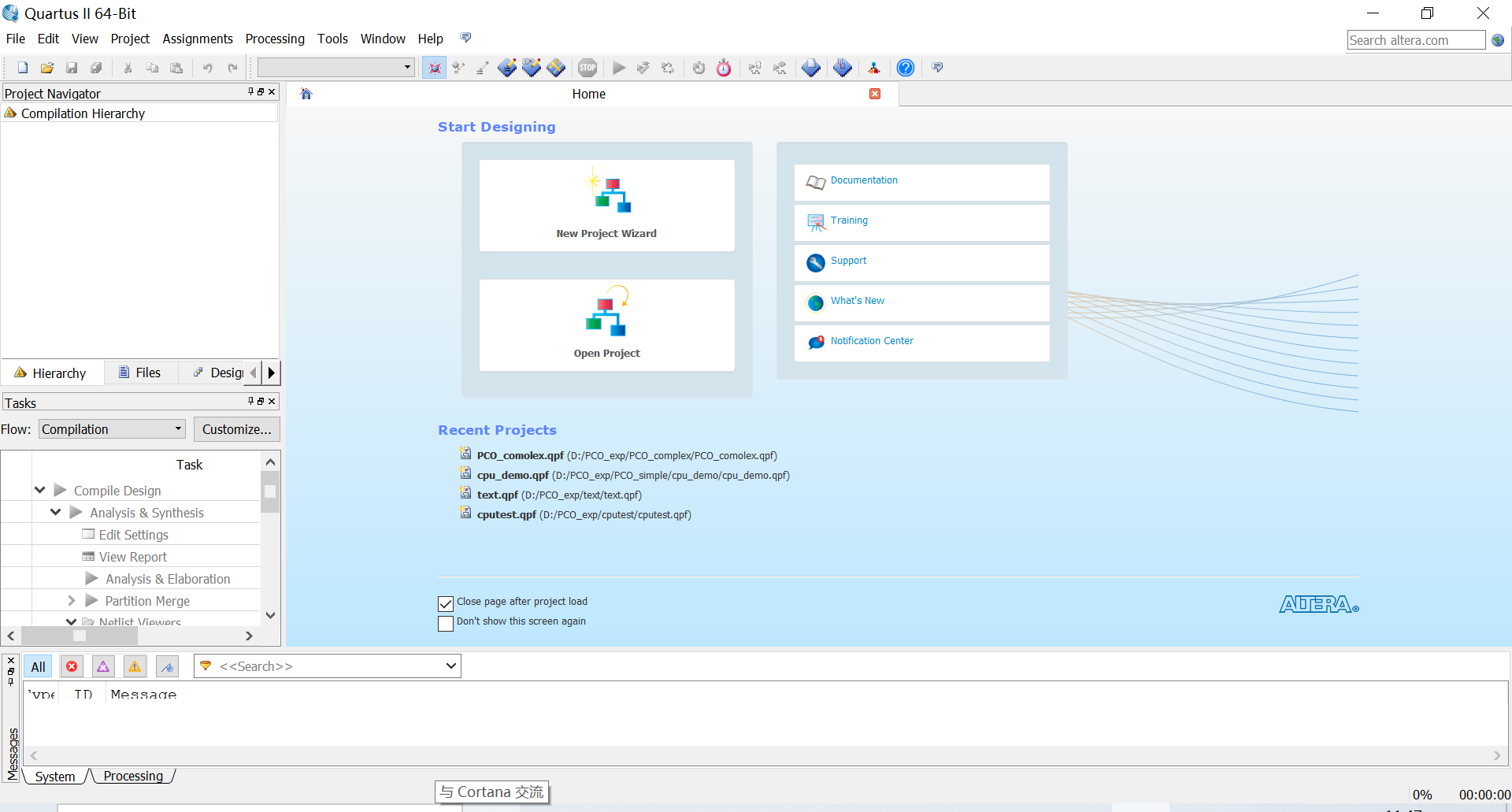
# 二、Quartus II使用说明及步骤

## (一)、建立工程项目并导入.v文件

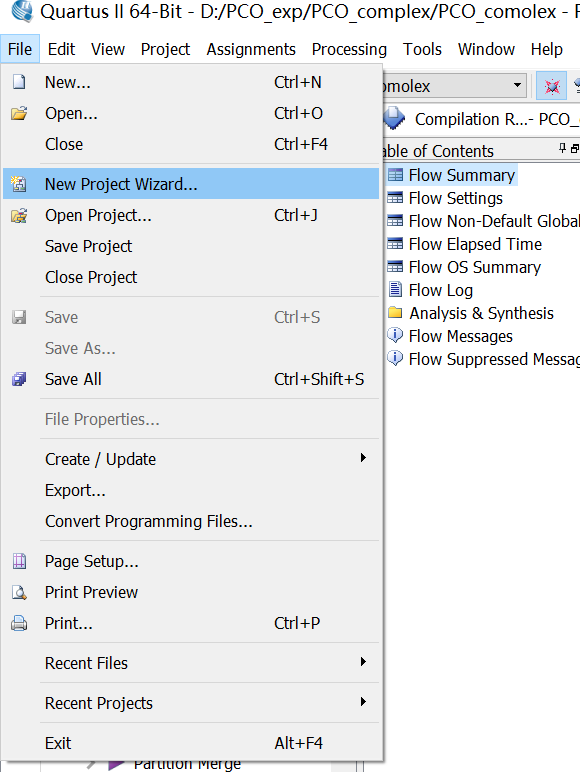
1、点击“开始”菜单，选择“Altera 14.0.0.200”—>“Quartus II 14.0（64-bit）”，如下图所示（以T09机房为例，其他机房系统不一样请自行查找该软件）。



打开Quartus II软件，界面如下图所示。

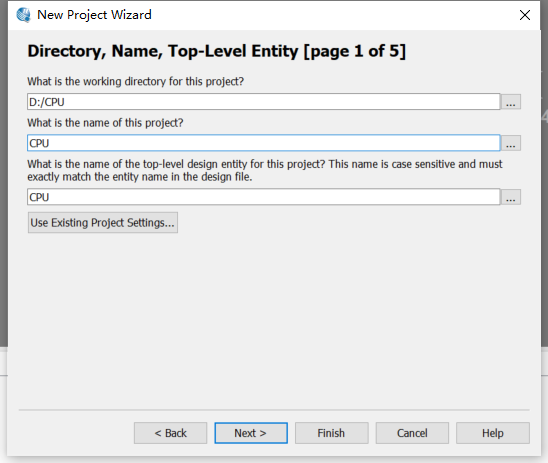


2、新建工程项目。点击上方菜单栏“File”->“New Project Wizard...”进行新建项目工程操作，界面如下图所示。

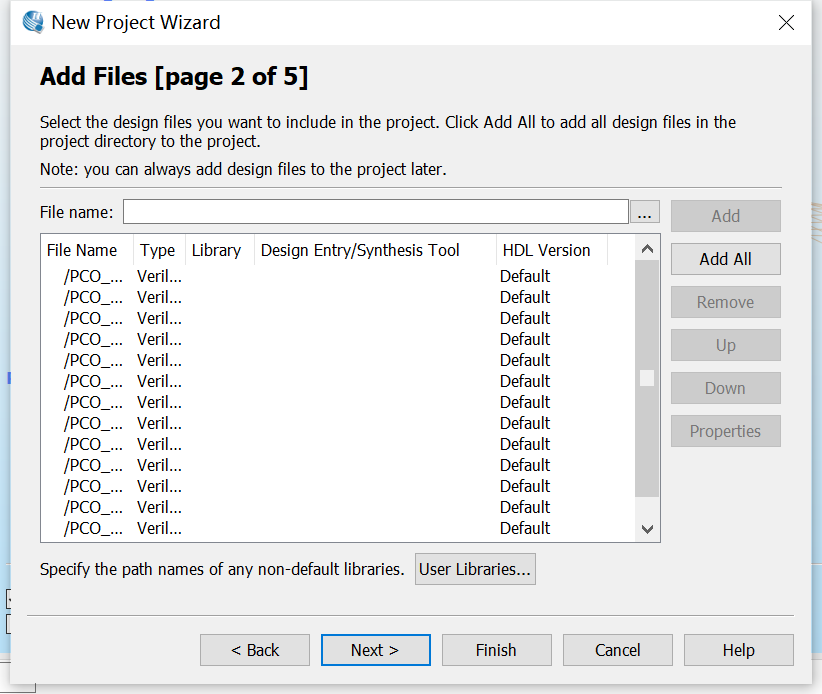


**注意：如果使用老师提供的工程文件（一定是全英文路径），则在上图中选择“Open Project…”，找到相应的工程文件（扩展名为qpf）打开即可，以下步骤省略直接跳到第17页（二）**

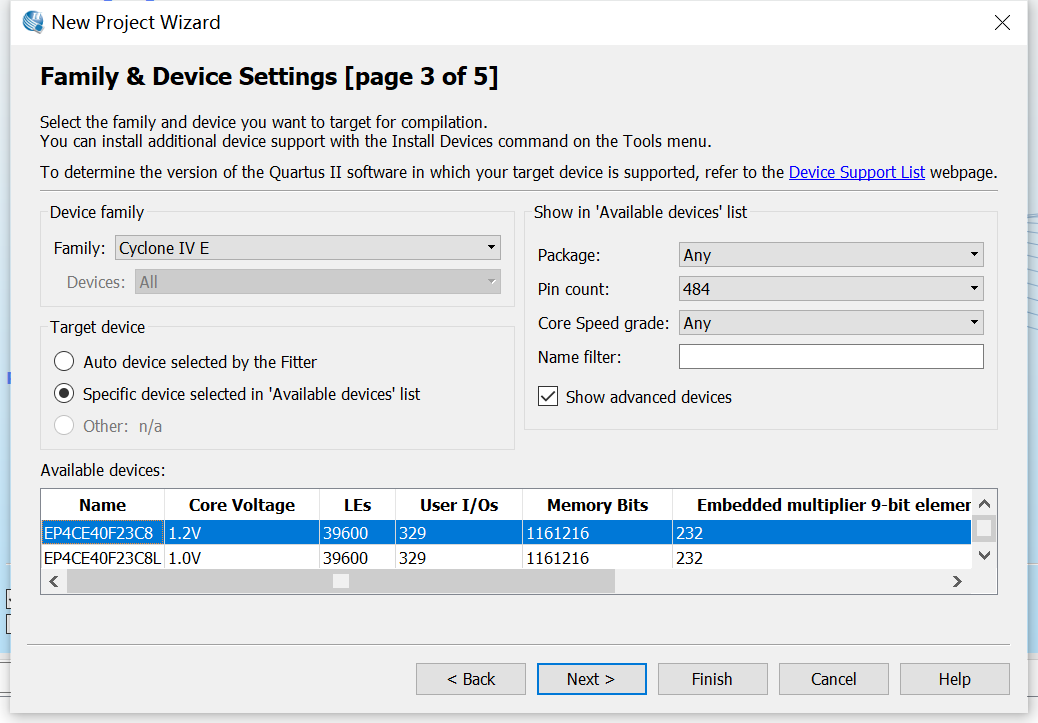
3、为工程安排存储路径，并为工程起名(全英文路径)，完成后点击“next”。界面如下图所示。注意：将工程命名为姓名拼音首字母缩写（例如张三，这里的工程名字就叫做zs）



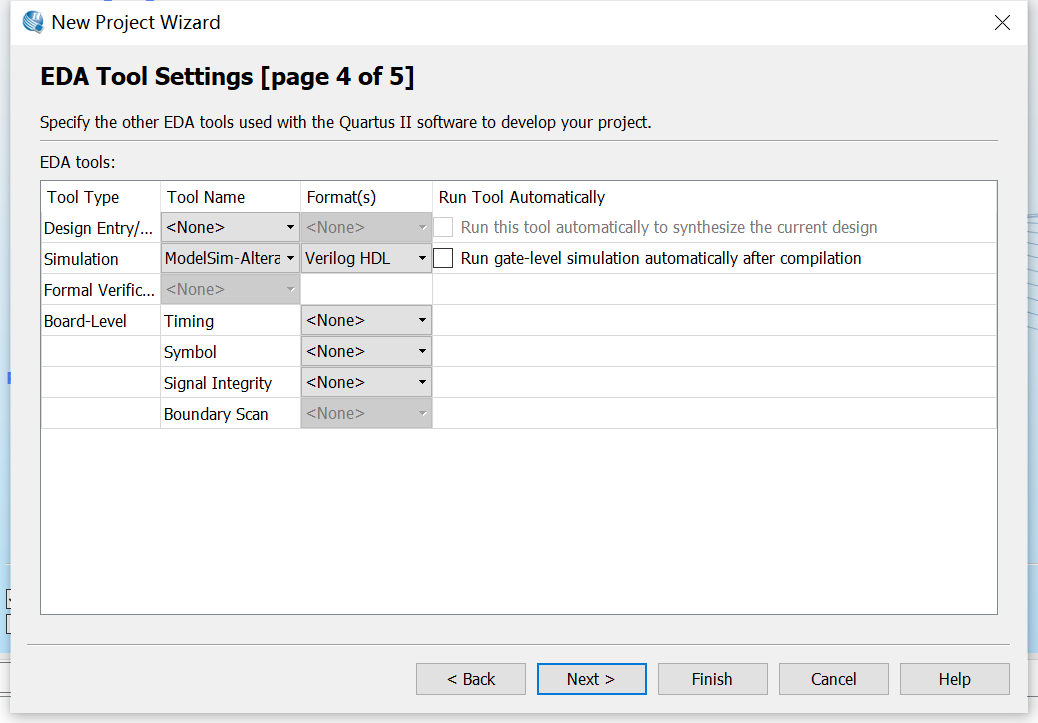
4、将从FTP上或微信群里下载的.v文件导入到里面后（存放.v文件的路径必须为全英文），导入文件后的界面如下图所示，点击“next”。

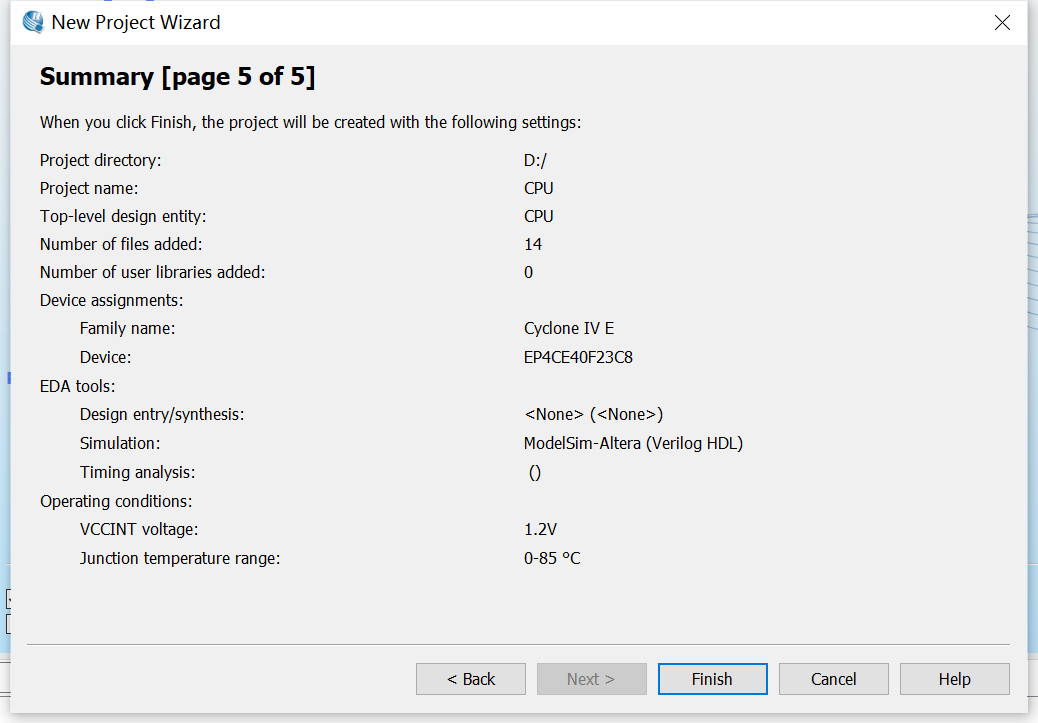


5、选择该工程的设备。将“Family”选为Cyclone IV E，“Pin Count”选为484，在“Available devices”的列表中，找到EP4CE40F23C8并选择后，点击“next”。界面如下图所示。

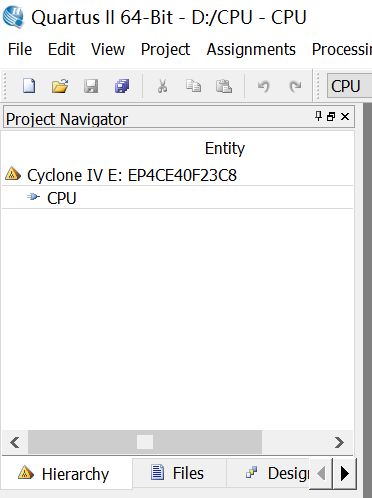


6、将标签为“Simulation”行的第二个选项框选择为Verilog HDL后，点击“next”，再点击“Finish”。界面如下图所示。

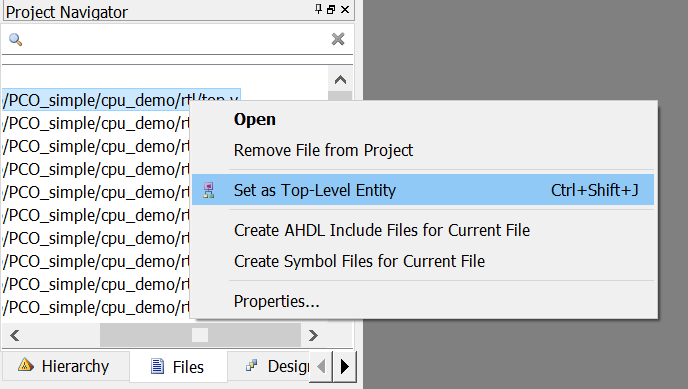




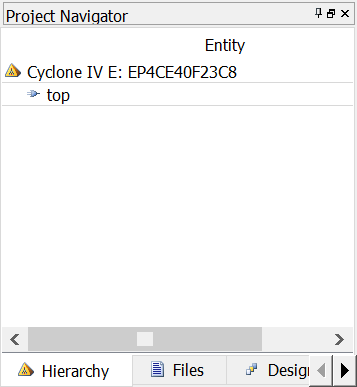
7、点击“Finish”后，在左上角可以看到“Project Navigator”界面，在里面可以看到当前的设备型号“Cyclone IV E：EP4CE40F23C8”和顶层模块“CPU”。界面如下图所示。



8、设置顶层模块。点击“Project Navigator”图框下菜单栏“Files”，可以看到导入的文件，右键单击“top.v”文件，在弹出的菜单中选择“Set as Top-Level Entity”。



选择后可以看到“Project Navigator”界面中的顶层模块中的内容由“CPU”变成了“top”。将“top”设为顶层模块的目的是：为之后的仿真文件指定模板，方便成功生成testbench仿真测试文件。界面如下图所示。



## (二)、示例程序中的CPU模块

### 1、启停电路——qtsj.v（已实现）

功能：启停电路是对输入时钟脉冲源产生的主脉冲信号进行完整、有效控制。

（1）为方便在课设验证时观察结果，设置以按钮A1按下作为RUN状态时的运行启动标志，以clr信号的出现作为运行结束标志。

（2）为方便进行课设验证时观察结果，在此设置两种演示时钟（clk\_quick,clk\_slow），使用SW\_choose滑动开关进行选择。

输入端：

* clk\_quick和clk\_slow分别为输入的快慢时钟，上升沿有效，通过输入端SW\_choose的值进行选择，由输出端口clk\_choose进行输出；
* clk\_delay为A1按钮的检测时钟，上升沿有，通过此时钟完成了A1按钮的延迟检测和防抖功能；
* clr为控制信号，其值来源于control.v模块，当clr有效时，时钟clk\_run输出为0；
* rst为复位信号，0为有效，rst与cpustate共同组成了reset的值。
* SW\_choose：选择快慢时钟；1位，为1是快时钟，否则是慢时钟；top模块的输入，即SW15的状态；
* A1：按钮key1；1位，按下为0，弹起为1；top模块的输入
* cpustate：CPU的状态；2位，CPU\_Controller模块的输出；

输出端：

* clk\_run：运行时的时钟，1位，送入控制器control的时钟；
* clk\_choose：时钟；1位，是寄存器ar、pc等的时钟；

### 2、分频模块——clk\_div.v（已实现）

功能：采用硬件平台所提供的 32.768MHz 时钟脉冲源提供主脉冲信号。教学实验的硬件操作一般是在 s 级节拍上进行的，因而对脉冲信号进行分频，使其产生 s 级脉冲周期。将输入的32.768MHZ进行分频。具体分频参数在top.v会详细说明。

### 3、显示输出模块——light\_show.v（已实现）

功能：通过HEX0-7数码管和LED灯将微控制信号、运算结果、指令地址和存储器中的指令输出。

### 4、状态转换模块——CPU\_Controller.v（已实现）

功能：通过SW2和SW1将CPU分为三个状态：IN状态（01），CHECK状态（10）和RUN状态（11）。

* IN状态为输入状态：从switch开关上输入指令并保存；
* CHECK状态为检查状态：检查输入的指令是否正确；
* RUN状态：依次执行之前输入的指令。

输入端：

* SW1：模拟按键SW17；1位，拨到上方为1，拨到下方为0；
* SW2：模拟按键SW16；1位，拨到上方为1，拨到下方为0；

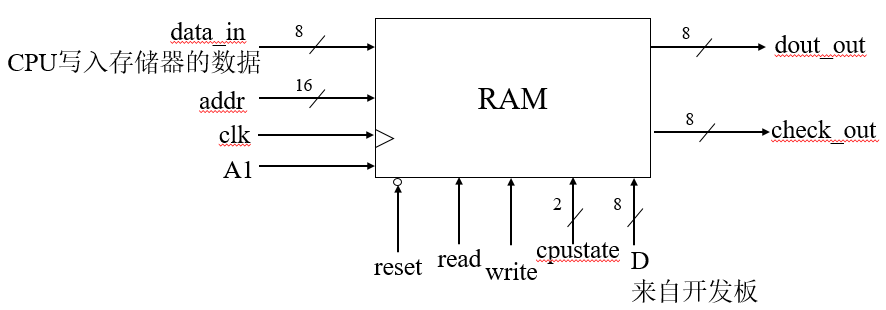
输出端：

* CPU\_state：2位，SW1SW2=01时输入(IN状态)，SW1SW2=10时检查（CHECK状态），SW1SW2=11时运行（RUN状态）；

### 5、存储器——ram.v（已实现）

功能：在IN状态下存储指令和数据，在CHECK状态下输出存储的指令和数据，在RUN状态下将存储的指令传给数据总线DB。

存储器的电路如下所示：



输入端：

* data\_in，8位，来自于cpu模块的data\_out，即cpu写入存储器的数据
* clk为时钟，只在IN和CHECK状态时，其上升沿有效；
* addr为16位，其值来源于地址寄存器ar模块的输出；
* A1为按钮，按下时（置0）有效，在IN状态下进行存储操作，在CHECK状态下进行检查操作；
* reset为清零信号，为零时有效，将内部计数器清零，将内部延时信号置1；
* read为控制信号，1位，1为有效，读出存储器ram[addr2]或memory[addr1]的值；
* write：为控制信号，1位，1为有效，将data\_in写入ram[addr2]；
* cpustate：cpu的状态，2位，01表示输入（IN状态），10表示检查（CHECK状态），11表示运行（RUN状态）
* D：8位，由FPGA开发板上SW7-SW0拨动开关的值组成，IN状态下按下A1按钮时进行存储操作，将指令存在内部存储器memory中；

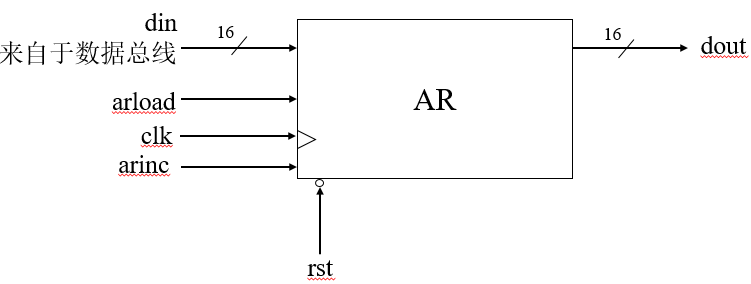
输出端：

* data\_out：8位，只在RUN状态下有效，其余状态时为高阻态，有效时如果addr[15:5]=0（即当前地址在0-31）则输出memeory[addr1]的内容，如果addr[15:5]!=0，则输出ram[addr2] 的内容；
* check\_out：8位，只在CHECK状态下有效，其余状态为高阻态，有效时输出当前地址cnt在memory存储的指令。

### 6、地址寄存器——ar.v（已实现）

功能：保存16位地址，可以自动加1

AR的电路如下所示：



输入端：

* din为16位，其值来源于数据总线DB；
* clk为时钟，上升沿有效；
* rst为清零信号，为零时有效，将输出dout清零；
* arload为控制信号，1为有效，其有效且clk有上升沿来临时，将输入din直接由dout输出。
* arinc：AR+1；该信号为1，AR++；1位，来自控制器

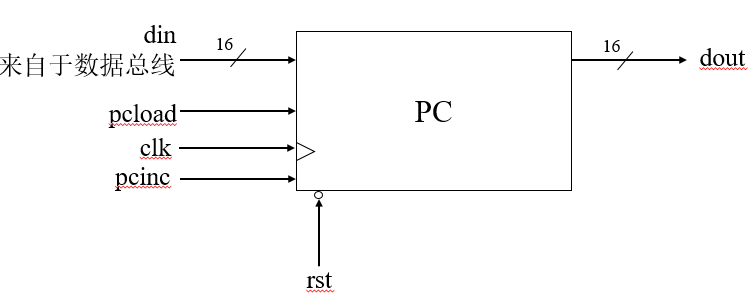
输出端：

* dout作为ar地址寄存器的值，可以送到存储器或连接到数码管。

### 7、程序计数器——pc.v（未实现）

功能：保存指令的地址，可以自动加1

PC的电路如下所示：



输入端：

* din为16位，其值来源于数据总线DB；
* clk为时钟，上升沿有效；
* rst为清零信号，为零时有效，将输出dout清零；
* pcload为控制信号，1为有效，其有效且clk有上升沿来临时，将输入din直接由dout输出；
* pcinc为控制信号，1为有效，其有效且clk有上升沿来临时，将输出的值+1后再输出。

输出端：

* dout为16位，送数据总线DB；

注：可仿照ar.v来写，注意替换相应的控制信号。

### 8、控制模块——control.v（需要修改）

功能：在不同的时刻生成不同的控制信号。

示例机涉及到的控制信号共22个：read, write, arload, arinc, pcinc, pcload, drload, trload, irload, r1load, alus, r0load, xload, zload, pcbus, drhbus, drlbus, trbus, r1bus, r0bus, membus, busmem (如上控制信号根据自己设计的指令系统而定，如果只是必做指令就不需要这么多控制信号)，以及2个内部控制信号inc, clr（inc在control模块内部定义，不在control模块的端口）。

输入端：

* din：指令，8位，来自IR；
* clk：时钟信号，1位，上升沿有效；
* rst：复位信号，1位，与cpustate共同组成reset信号；
* cpustate：当前CPU的状态（IN，CHECK，RUN），2位；
* z：零标志，1位，零标志寄存器的输出，如果指令中涉及到z，可加上，否则可去掉；

输出端：

* clr：清零
* 其他需要的控制信号

### 9、CPU模块——cpu.v（需要修改）

功能：将各模块（qtsj, ar, pc, dr, tr, ir, r0, r1, x, alu, z, control）实例化（已给出前3个实例化语句，其他的需要自己补充。如果没有条件转移指令可以没有z）。并对数据总线进行赋值，已给出了多个寄存器的输出，可根据具体情况进行删减。

### 10、顶层模块——top.v（需要修改）

功能：将clk\_div，cpu，mem，light\_show实例化，并且其输入和输出端作为引脚绑定和生成仿真文件的依据。需要补充cpu实例化的语句。

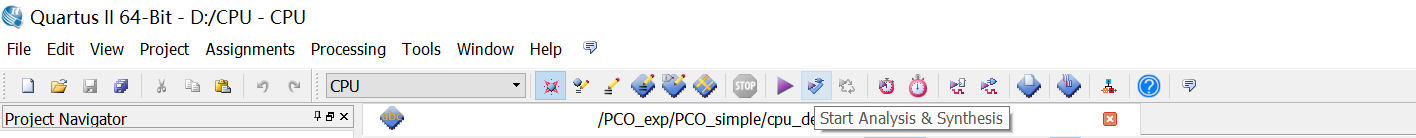
### 11、其他模块——根据需要自行设计，示例工程中给出了只有模块说明的代码，需要自行编写代码

### 12、testbench文件——top.vt（需要修改）

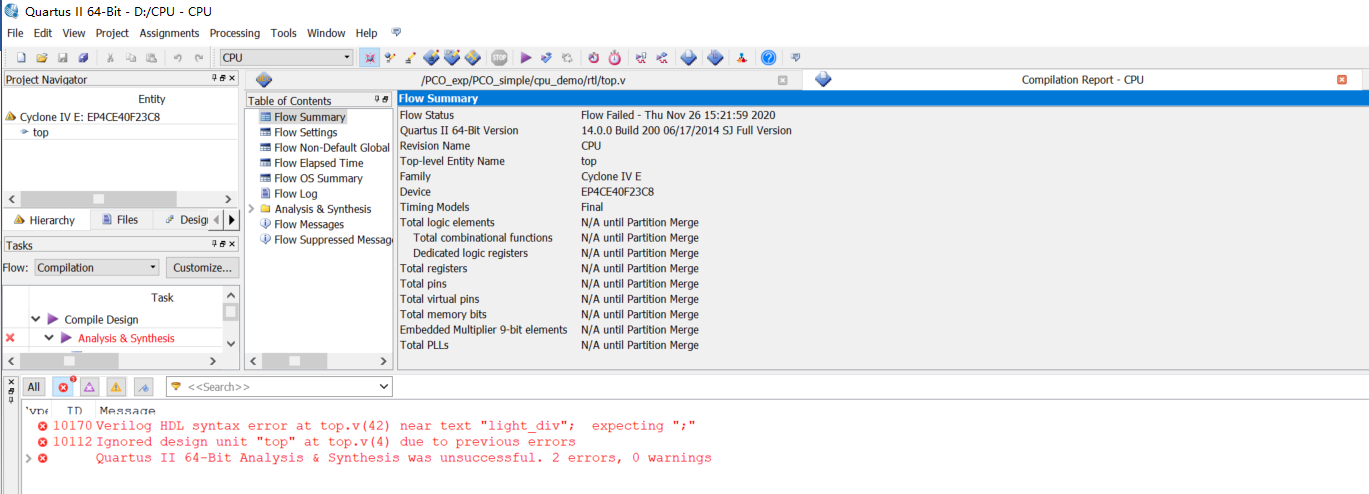
## (三)、编写代码和测试文件并进行仿真

1、完成所有模块后，点击界面上方的“”按钮进行“Analyzes and synthesizes”代码分析，对分析后出现在“Message”窗口的报错进行排查和修改。

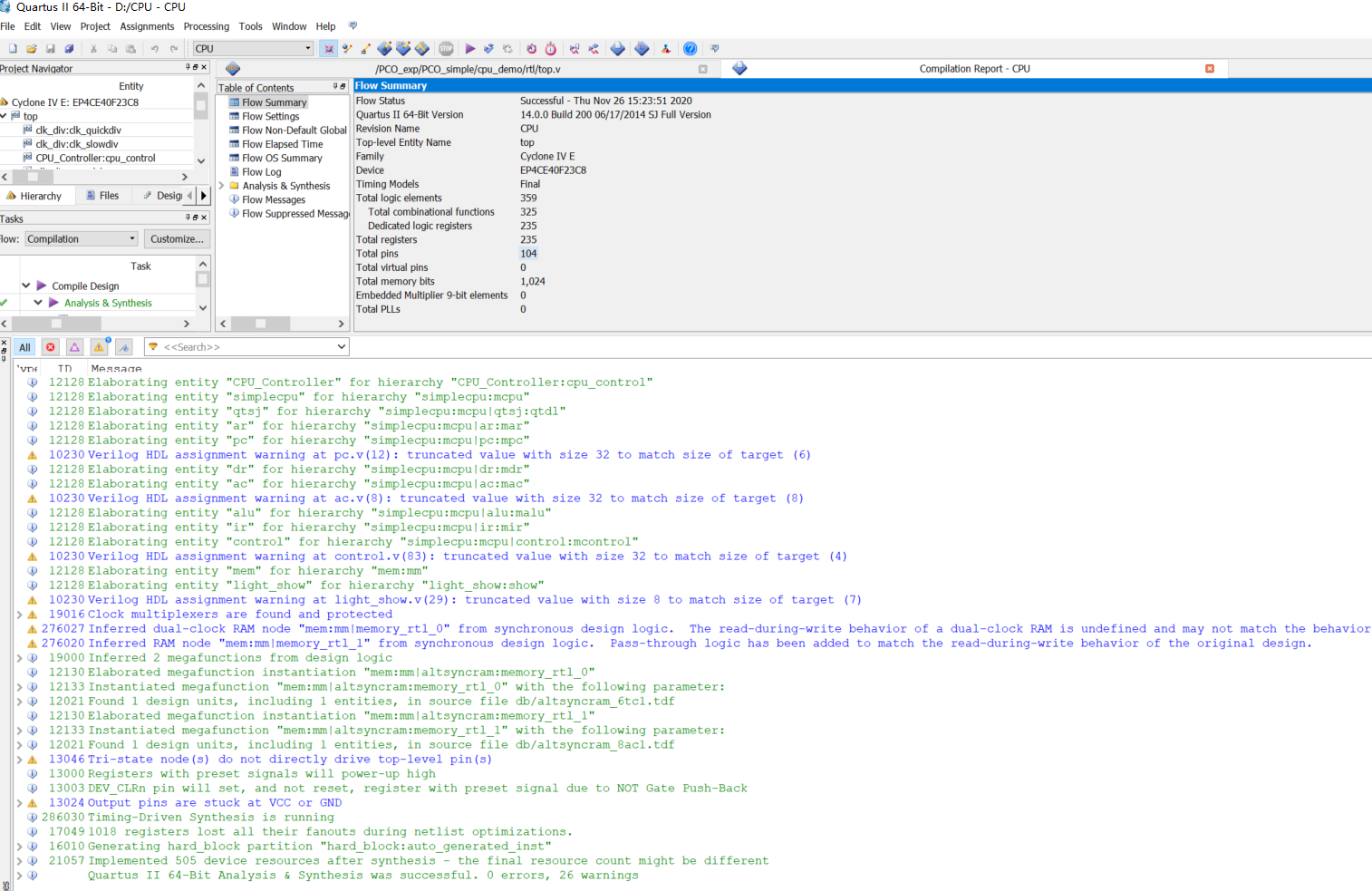
“Analyzes and synthesizes”按钮位置如下图所示。



2、报错界面如下图所示。

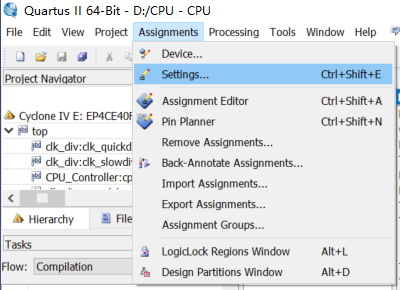


3、无报错，分析成功后界面如下图所示。

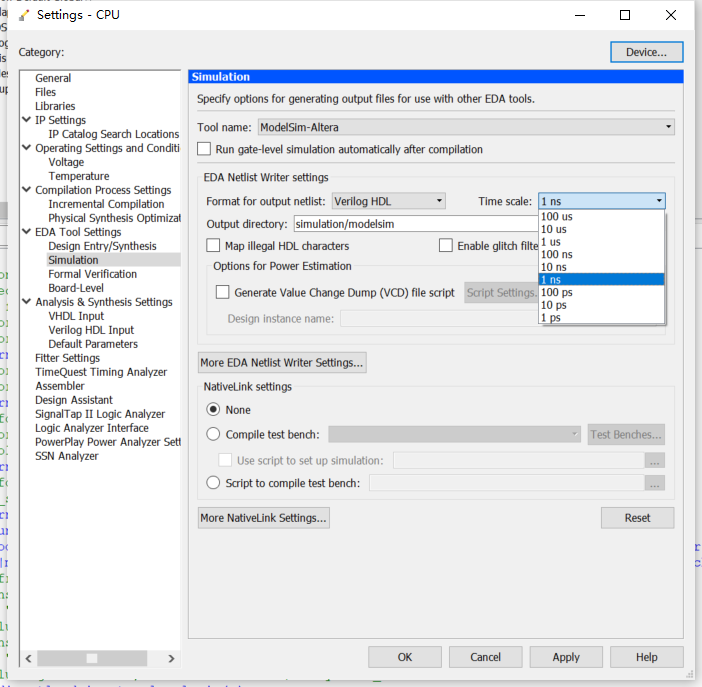


**注意：如果使用老师提供的工程文件，则以下第4-14步可以省略。 在进行第15步仿真前，需要修改top.vt，改成自己所设计指令的仿真文件**。

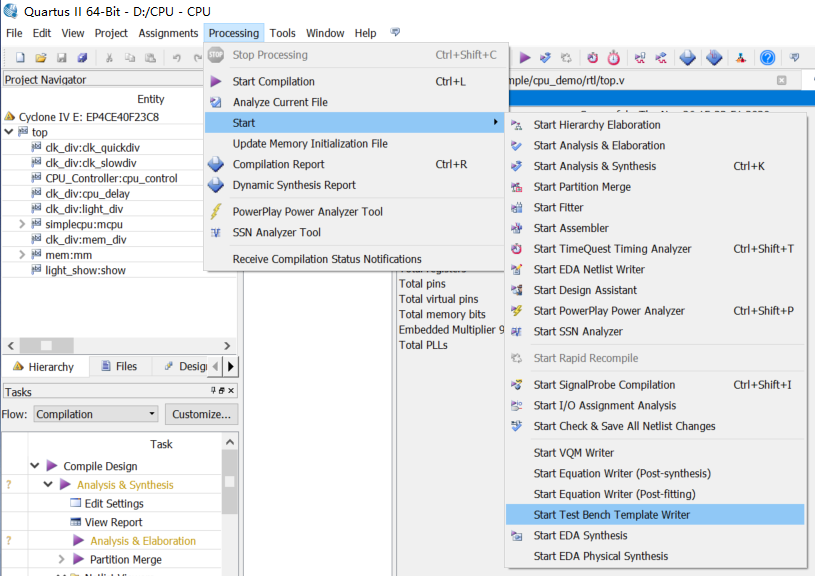
4、在界面顶部菜单点击“Assignments”—>“Settings”



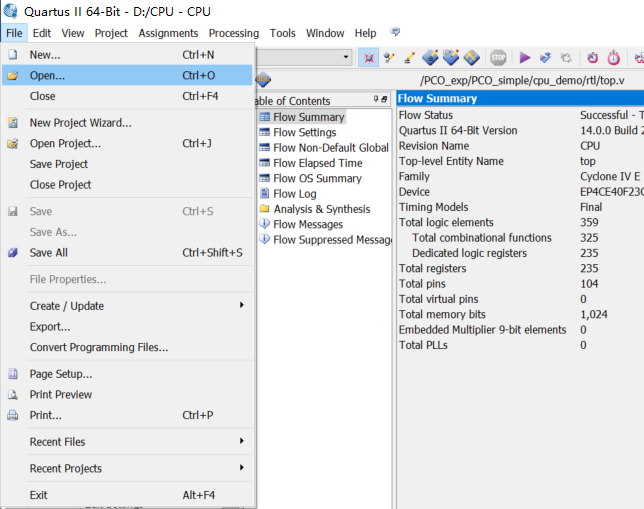
5、进入“Settings”界面后，点击左侧选择栏“Simulation”，将右侧“Time scale”下拉菜单栏选为“1ns”（选其他的大于1ns的也可以）。点击“Apply”—>“ok”进行保存退出。



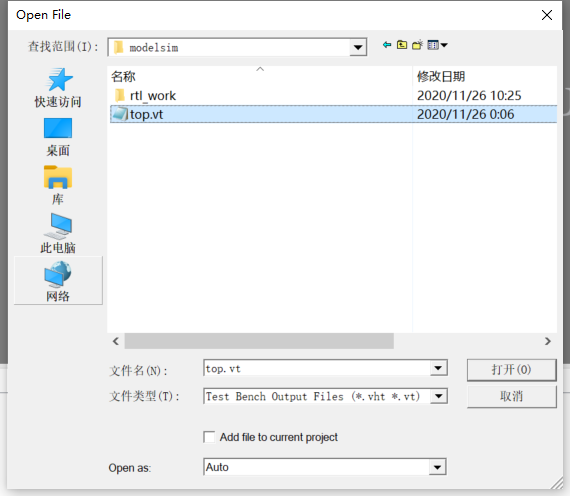
6、生成testbench仿真文件。分析排查无误后，在界面顶部菜单点击“Processing”—>“Start”—>“Start Test Bench Template Writer”生成testbench仿真测试模板文件。



7、寻找testbech仿真文件的路径。在界面顶部菜单点击“File”—>“Open”

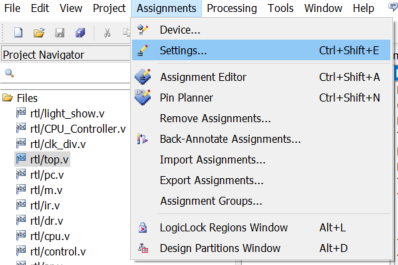


8、打开生成的testbench仿真文件。打开“存放工程项目的路径/simulation/modelsim/top.vt”，具体界面如下图所示，注意文件类型。

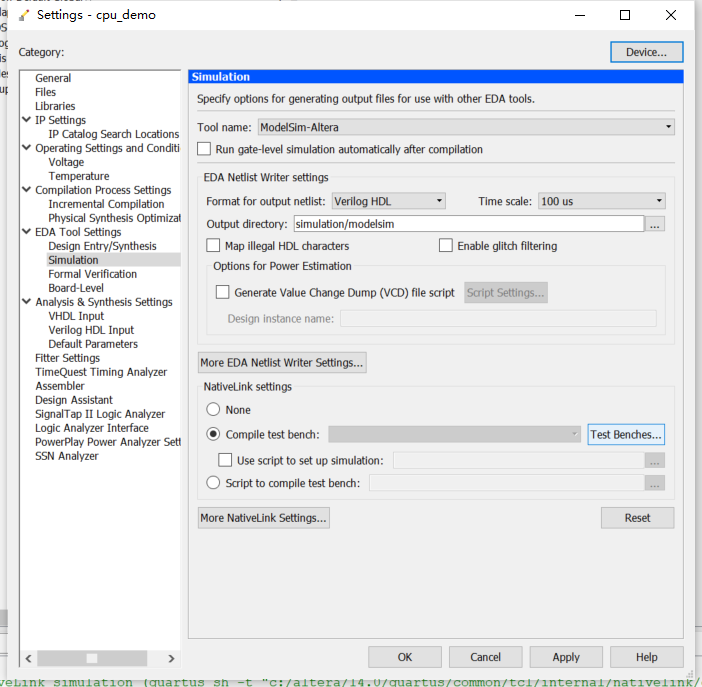


9、打开后将FTP中所给的.vt文件中内容全部复制到当前仿真文件之中，保存

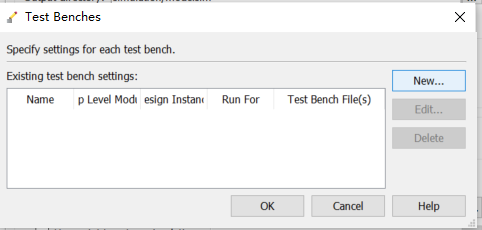
10、设置quartus工程仿真文件。点击顶部菜单选项“Assignments”—>“Settings”，点击左侧选项栏中的“Simulation”。



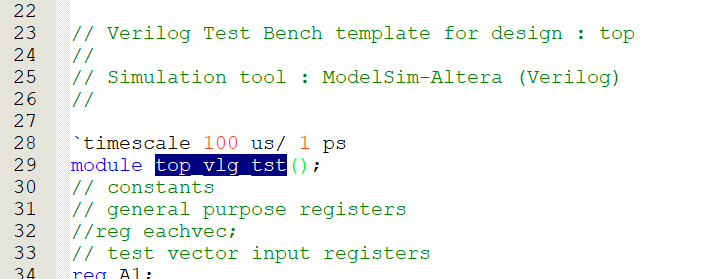
11、将右侧“NativeLink settings”由“None”选择为“Compile test bench”，点击“Test Benches…”选项。

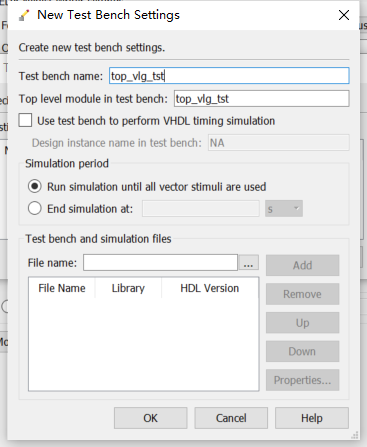


12、在“Test Benches”中点击“New”选项。

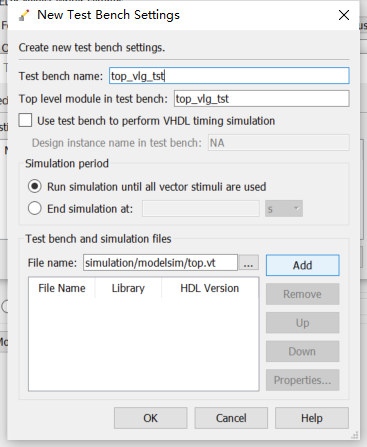


13、在“Test bench name”中起名（与testbench中的实例模块同名，一般都为top\_vlg\_tst），在File name在添加路径（路径为top.vt文件的存储路径）。

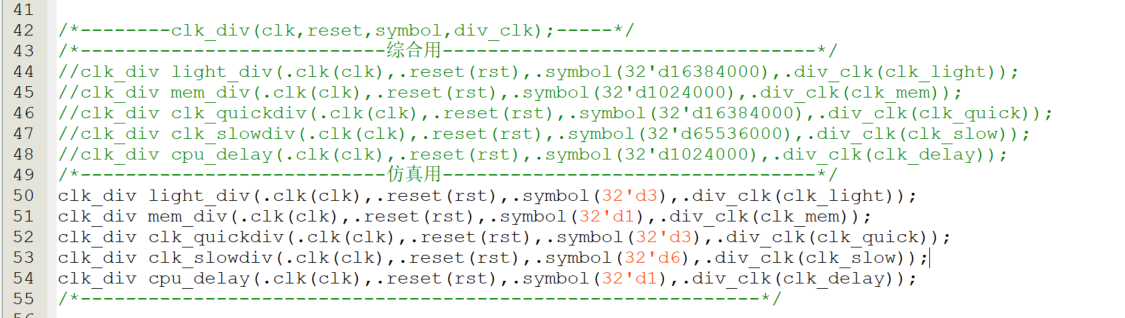


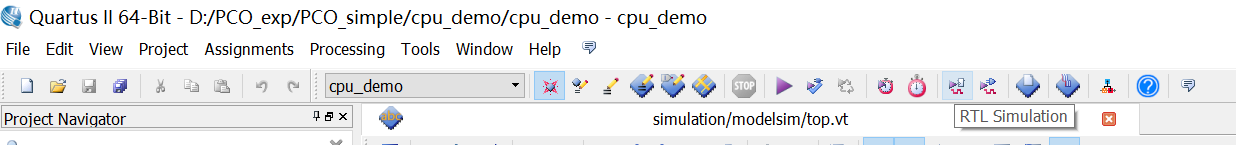


14、点击“Add”—>“ok”，一路全点“ok”。

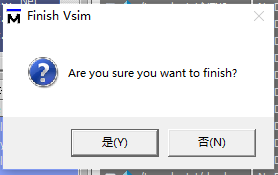


15、仿真。在“top.v”文件中，将综合部分注释掉，将仿真部分解除注释后，保存。点击分析，无报错后，点击界面顶部的“”按钮，进行“RTL simulation”仿真。





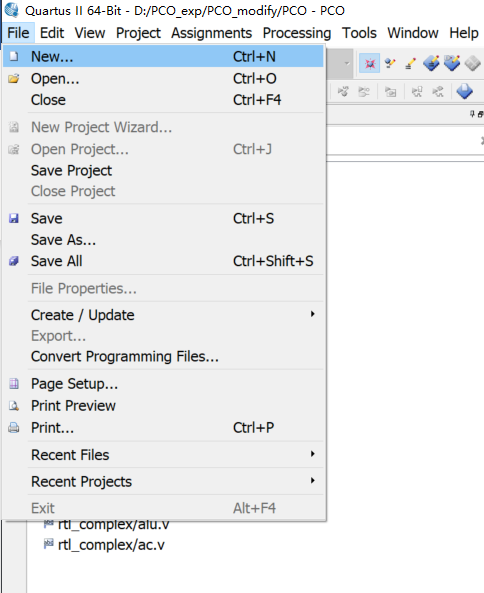
16、仿真波形显示。进入ModelSim界面，对弹出框“Finish Vsim”选择“否”，观察波形是否符合此次课程设计CPU的结果。若不符合则继续修改代码直到仿真波形与预期相同。

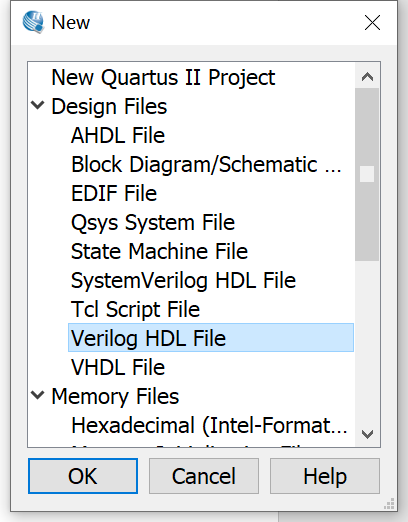


## (四)、注意事项及常见问题

### 1、如何新建.v文件

点击主页面左上角菜单栏选项“File”—>“New”，进入“New”界面后，选择“Design Files”列表下的“Verilog HDL File”。

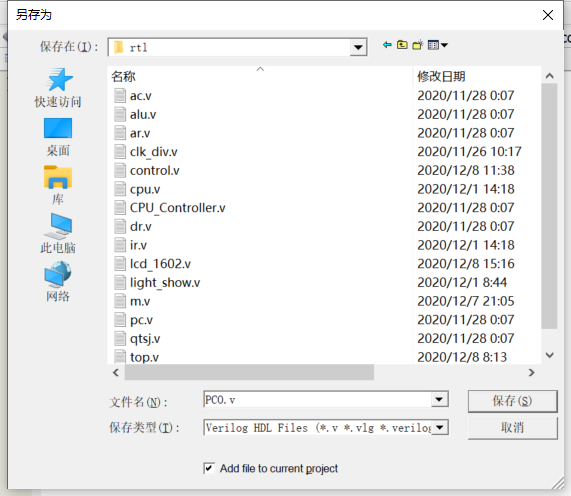




点击后，可在首页文件栏中出现“Verilogx.v”。



在其中写好代码后，点击保存，将文件放在需要的位置，取好名（英文）后保存。

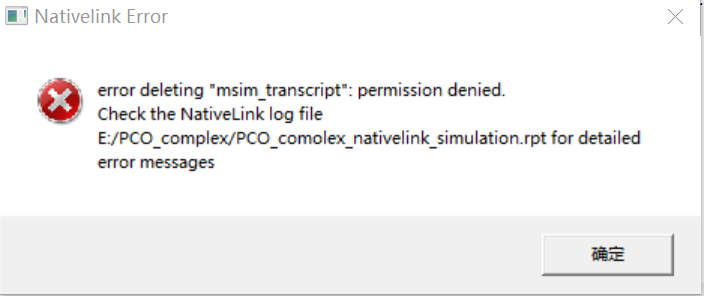


保存成功后，即可在左侧Files选项栏中看到文件列表中看到刚才保存的具体.v文件。



### 2、仿真时出现以下错误

点击分析，无报错后，点击界面顶部的“”按钮，进行“RTL simulation”仿真时出现以下错误：



说明已经打开了ModelSim，现在要再次打开ModelSim，会出现如上错误。

解决方法：关掉ModelSim，重新进行仿真。

### 3、无法生成pof综合文件或无法进行仿真，需要先破解

出现无法生成pof综合文件或无法进行仿真的主要原因为2个：①未完全成功破解quartus；②安装后选择了30天试用版。请务必按照以下的破解步骤破解！！！

1、破解前请先关闭杀毒软件和防火墙，防止破解文件被误判为病毒。

2、解压破解压缩包“Crack\_QII\_14.0\_Windows密码12345.rar”，点击程序“Quartus\_II\_14.0破解器.exe”。

图形用户界面, 文本

描述已自动生成

3、点击查找，到“C:\altera\14.0\quartus\bin64”路径下，点击“gcl\_afcq.dll”文件后，选择“打开”。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

4、点击程序“Quartus\_II\_14.0破解器.exe”中的“下一步”。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

5、点击查找，到“C:\altera\14.0\quartus\bin64”路径下，点击“sys\_cpt.dll”文件后，选择“打开”。

图形用户界面, 文本

描述已自动生成

6、点击程序“Quartus\_II\_14.0破解器.exe”中的“完成”。

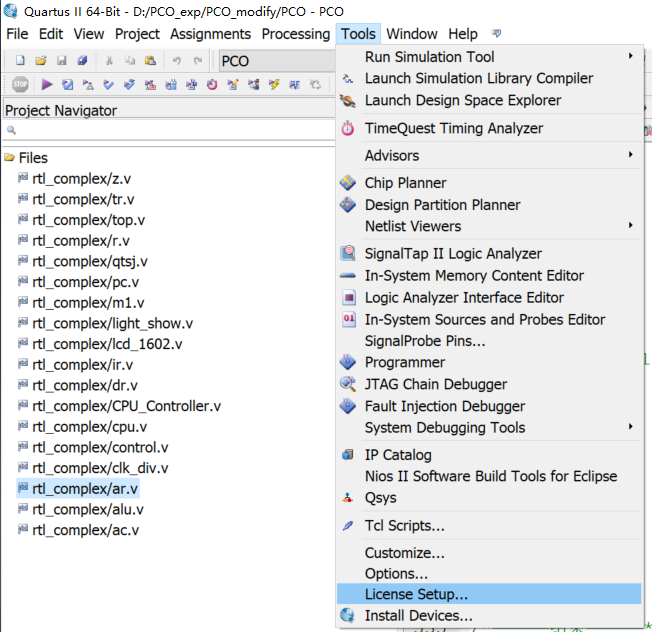
图形用户界面, 文本, 电子邮件

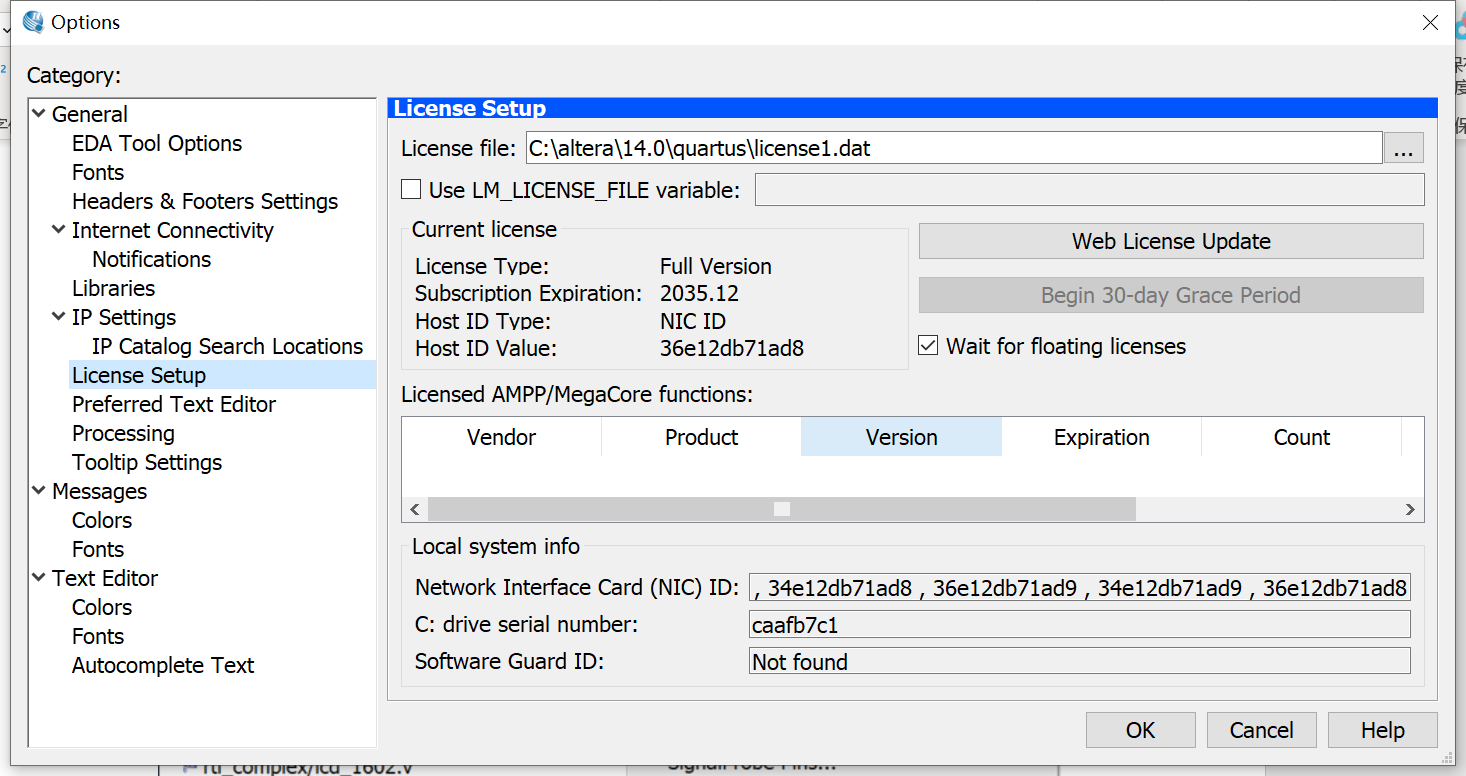
描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

7、验证破解是否成功，打开Quartus软件，在左上角菜单栏总选择“Tools”—>“License Setup”，查看Licence Type是否为Full Version。





# 三、提交内容方式和提交时间：

1、课程设计报告模板从<ftp://202.204.121.156/张海燕/授课材料/>计算机组成原理A课程设计/中下载，课程设计报告一律写成word文档。

2、将课程设计报告和整个工程文件打包压缩（命名要求：学号姓名，例如你叫张三，学号为050101，则压缩包名称为“050101张三.rar”），上传到<ftp://202.204.121.156/张海燕/作业/>计算机组成原理A课程设计/各班文件夹。

3、提交时间：第19周周五前