# 示例机的设计和开发板上验证

完成一条指令：

以MOVAC为例

1. 取指：F1（t0）

F2(t1)

F3(t2)

MOVAC(t3)

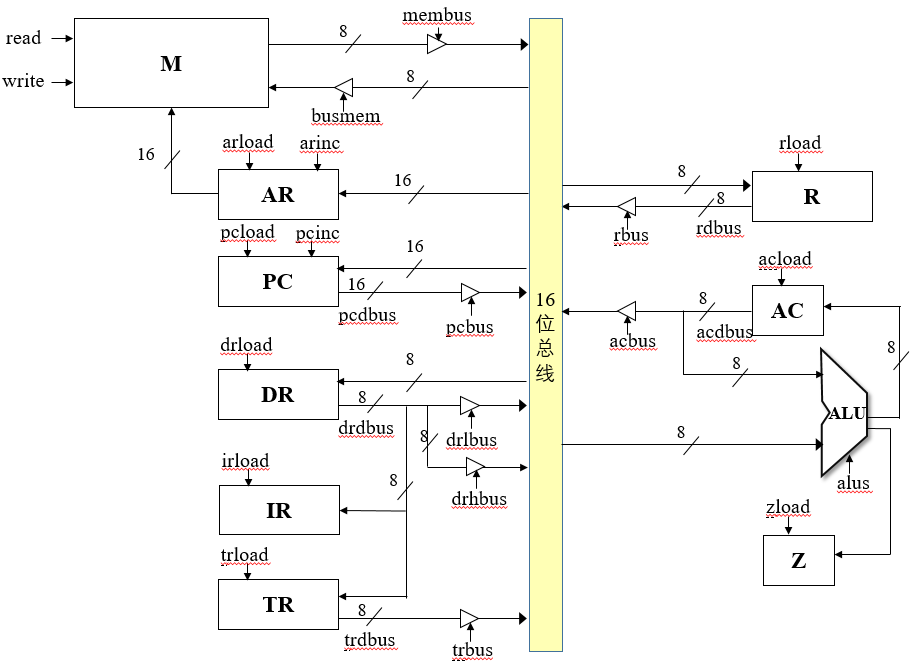
1. 执行

## 一)、数据通路

CPU具有以下寄存器：

* 2个8位通用寄存器AC[7：0]和R[7：0]，AC用作目的寄存器，R用作源寄存器。
* 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 （选做会用）
* 1位的“0”标志寄存器Z，用于标志运算结果0。

数据通路如图1所示。



irout

其余8位寄存器连到低八位

16’程序计数器 PC

16‘地址寄存AR

Z寄存器 标志0

运算器 ALU

目的寄存器AC

源寄存器 R

数据暂存器TR

指令寄存器IR

数据寄存器 DR

存储 M

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）ALU的其中一个输入是AC的输出，还要将ALU的运算结果送AC。如果自行设计的指令系统中包含条件转移指令，还需要设置标志寄存器Z的值。

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 | rload | 将总线上的内容送入r |
| 13 | rbus | 打开三态门，r中的内容送总线 |
| 14 | acload | 将总线上的内容送入ac |
| 15 | acbus | 打开三态门，ac中的内容送总线 |
| 16 | alus | 选择alu运算的控制信号 |
| 17 | read | 读存储器 |
| 18 | write | 写存储器 |
| 19 | membus | 打开三态门，存储器中的内容送总线 |
| 20 | busmem | 打开三态门，总线上的内容送存储器 |
| 21 | zload | 将alu的运算结果送入z（运算结果=0，送入1，否则送入0） |

## (二)、指令系统

指令系统如表2所示。表2中的T为16位的地址（注：该表中的地址都是16位的，因此有的指令是单字指令，有的指令是三字指令，例如第1-2，5-7条指令是三字指令）。可参考以下指令格式（也可自行设计指令格式），自行设计的指令须符合课设任务书的要求。

表2 指令系统

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 指令 | 格式 | 功能 |
| 0 | NOP | 00000000 | No Operation |
| 1 | LDAC | 00000001 T | AC<— M[T] |
| 2 | STAC | 00000010 T | M[T] <— AC |
| **3 √** | **MOVAC** | **00000011** | **R<— AC** |
| **4** | **MOVR** | **00000100** | **AC<— R** |
| 5 | JUMP  无条件转移 | 00000101 T | GOTO T |
| 6 | JMPZ（条件跳转） | 00000110 T | IF(Z=1) GOTO T |
|  | JPNZ（条件跳转） | 00000111 T | IF(Z=0) GOTO T |
| **8 √** | **ADD** | **00001000** | **AC<— AC+R** |
| **9 √** | **SUB** | **00001001** | **AC<— AC-R** |
| **10 √** | **INAC** | **00001010** | **AC<— AC+1** |
| **11 √** | **CLAC** | **00001011** | **AC<— 0** |
| **12 √** | **AND** | **00001100** | **AC<— AC^R** |
| **13 √** | **OR** | **00001101** | **AC<— AC or R** |
| **14 √** | **XOR** | **00001110** | **AC<— AC xor R** |
| **15 √** | **NOT** | **00001111** | **AC<— ~AC** |

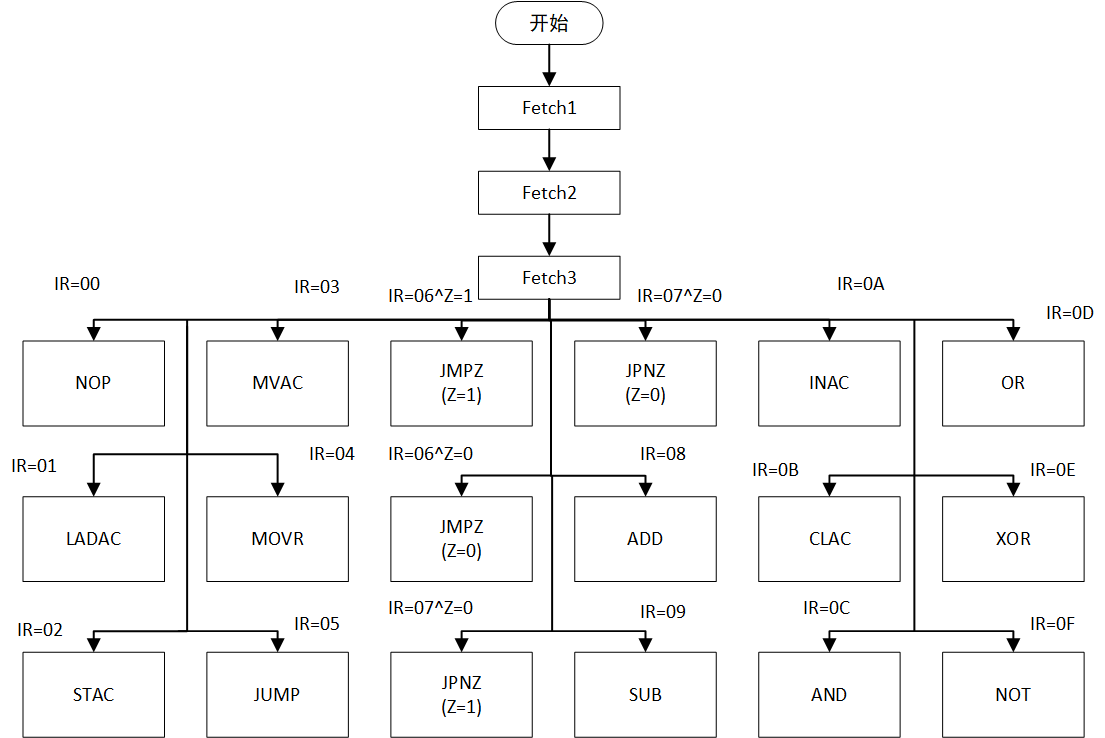
## (三)、取指和译码

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 |

各指令周期的状态转化图如下（这是该机所有指令，需要换成自己设计的指令）。取指周期是所有指令公有的。

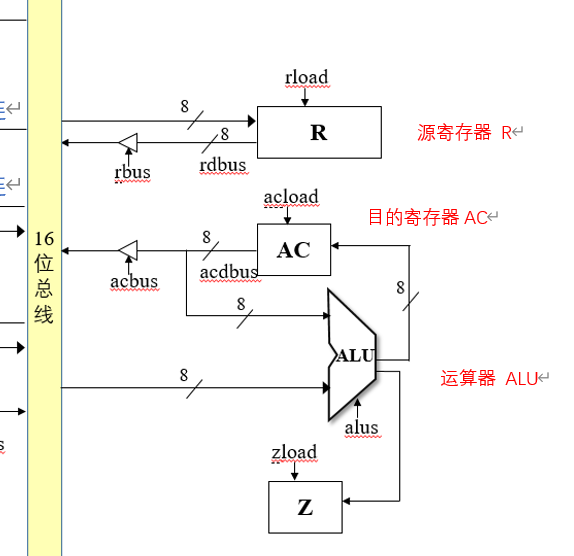


## (四)、指令的执行（只举几条典型指令的执行，其余指令自行设计）

### 1、NOP指令

NOP指令是最容易实现的一条指令，CPU什么都不作，而只是去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 加法★



**1.R->总线->ALU**

**2.AC->ALU**

**3.ALU执行加法**

**4.结果送回AC**



ADD指令在一个状态完成的，功能：AC<—AC+R。（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）。指令执行状态只有一个状态ADD1，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 | AC<—AC+R | rbus、alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |

### 3、INAC指令 自加 ★

INAC指令在一个状态完成的，功能：AC<—AC+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）。指令执行状态只有一个状态INAC1，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 INAC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 | INAC1 | AC<—AC+1 | alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |

### 4、LDAC指令 取主存的数据

LDAC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AC中。

在取指周期的Fetch2中，存储地址T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T的高8位地址）
* AR+1，取T的高8位地址

即LDAC1：DR<—M，PC<—PC+1，AR<—AR+1

接下来CPU可以取T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LDAC2：TR<—DR，DR<—M，PC<—PC+1

这时CPU得到了T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，然后把要去的数载入到DR，最后复制数据到AC中。

LDAC3：AR<—DR，TR

LDAC4：DR<—M

LDAC5：AC<—DR

指令执行状态有5个状态LDAC1——LDAC5，该指令执行周期的节拍、状态、数据流和控制流如表6所示：

低八位、高八位、一起给ar、数据给DR、DR给AC

表6 LDAC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 | LDAC1 | DR<—M，PC<—PC+1，  AR<—AR+1  低八位地址给DR，AR取高八位 | read、membus、drload、pcinc、arinc |
| T4 | LDAC2 | TR<—DR，DR<—M，  DR存的低八位转存TR，高八位给DR  PC<—PC+1 | trload、read、membus、drload、pcinc |
| T5 | LDAC3 | AR<—DR，TR  低八位和高八位均给AR | drhbus、trbus、arload |
| T6 | LDAC4 | DR<—M  通过AR的地址，M把需要的数据给DR | read、membus、drload |
| T7 | LDAC5 | AC<—DR  DR存的数据给AC，在这里自行设计了busac，直接给ac | drlbus、busac、alus |

### 5、MOVR指令: R的内容送到AC ★

该指令的功能是将R的内容送到AC，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7所示：

表7 MOVR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 | MOVR1 | AC <—R | rbus、busac、alus |

### 6、MOVAC：AC的内容到R ★

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | MOVAC1 | R<—AC | acbus、rload、alus |

该指令的功能是将AC的内容送到R，只需要1个节拍即可。

### 7、JUMP指令

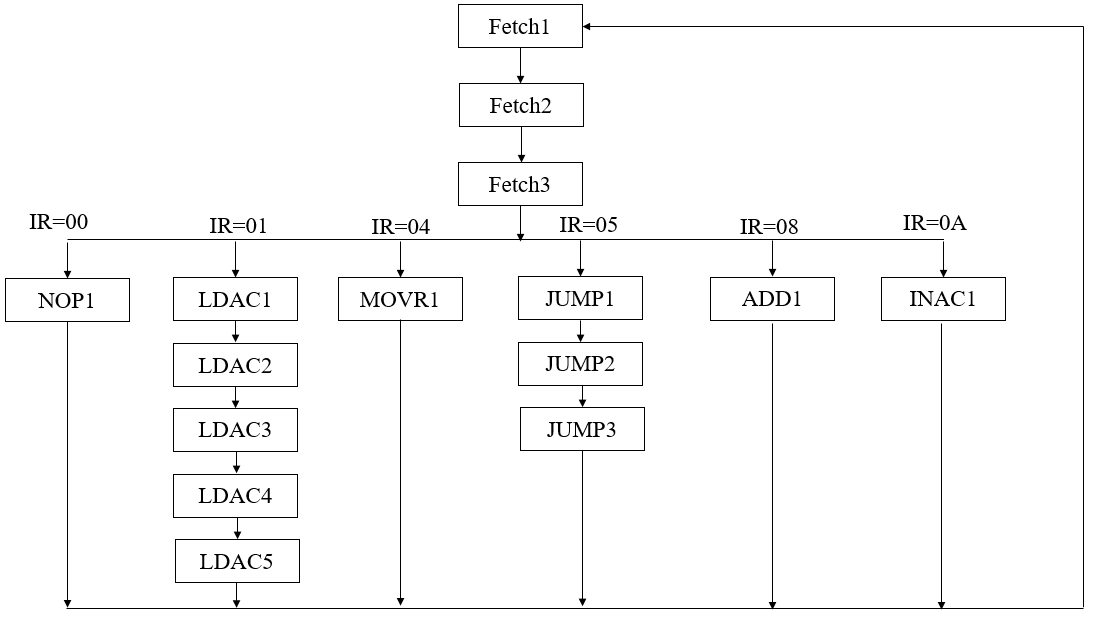
JUMP指令执行时，CPU先按照LDAC的取址方式取出要跳转的地址，然后送PC。该指令执行周期的节拍、状态、数据流和控制流如表8所示：

低八位、高八位、进pc

表6 JUMP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 | JUMP1 | DR<—M，AR<—AR+1  低八位给DR 地址加一，为了获取下一个高八位 | read、membus、drload、arinc |
| T4 | JUMP 2 | TR<—DR，DR<—M  低八位存到TR，高八位存入DR | trload、read、membus、drload |
| T5 | JUMP 3 | PC<—DR，TR  DR、TR的地址同时进入PC待命 | drhbus、trbus、pcload |

这些指令的状态图如下：（需要自行补充所设计指令的状态图）



### 8、SUB减法指令 ★

该指令的功能是完成两个数相减，只需要一个状态SUB1，状态表如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | SUB1 | AC<—AC-R | rbus、alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |

### 9、CLAC 清零指令　　★

该指令的功能是完成AC的清零。只需要一个状态CLAC１，状态表如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | ＣＬＡＣ１ | ＡＣ＜－０ | ｒｓｔ |

### １０、AND操作　与　★

该指令的功能是完成AC和R的与作。只需要一个状态AND１，状态表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | AND | AC<—AC　AND　R | rbus、alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |

### １１、OR　或操作　★

该指令的任务是完成AC和R数据的相或，状态表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | OR１ | AC<—AC　OR　R | rbus、alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |

### １２、XOR操作　异或操作　★

该指令的任务是完成R和AC的与操作，只需一个节拍。

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | XOR１ | AC<—AC　XOR　R | rbus、alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |

### １３、NOT　非操作　★

该指令的任务是完成AC的取非操作，只需一个节拍。

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T3 | NOT１ | AC<—～ＡＣ | rbus、alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |

**１4.STAC 操作**

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 状态 | 微操作序列（数据流） | 控制信号（控制流） |
| T0 | Fetch1 | AR<—PC  PC的地址给AR，AR指导M取指令 | pcbus、arload |
| T1 | Fetch2 | DR<—M，PC<—PC+1  M里面的指令地址给DR，PC++ | read、membus、drload、pcinc |
| T2 | Fetch3 | IR<—DR，AR<—PC  DR存的指令数据给IR指令寄存器，PC里的地址给AR准备下一条指令。 | irload、pcbus、arload |
| T3 | STAC1 | DR<—M，PC<—PC+1，  AR<—AR+1  低八位地址给DR，PC++，AR++准备取下一地址 | read、membus、drload、pcinc、arinc |
| T4 | STAC2 | TR<—DR，DR<—M，  PC<—PC+1  低八位给TR，高八位给DR，PC++准备下一条指令 | trload、read、membus、drload、pcinc |
| T5 | STAC3 | AR<—DR，TR  低八位高八位送给AR，获取地址 | drhbus、trbus、arload |
| T6 | STAC4 | DR<—AC  AC的数据传给DR | read、membus、drload |
| T7 | STAC5 | M<—DR  DR的数据给M存放 | drlbus（?）、acload、alus |

## (五)、ALU的设计

根据数据通路可知：ALU的一个输入来自于AC的输出，另一个输入来自于总线BUS，ALU的运算结果送入了AC。所以要设计ALU，我们首先来看那些指令修改了AC的值（即使用ALU进行了运算）：（给出的典型指令中只有以下4条指令修改了AC，根据自行设计的具体指令进行添加）

LDAC5：AC<—DR

MOVR1：AC<—R

ADD1：AC<—AC+R

INAC1：AC<—AC+1

根据ALU的两个输入和输出的对应关系可得出如下关系：

LDAC5：AC<—BUS，（将总线上来的数据送入AC）

MOVR1：AC<—BUS，（将总线上来的数据送入AC）

ADD1：AC<—AC+BUS，（将总线上来的数据和AC相加后的结果送入AC）

INAC1：AC<—AC+1，（将AC自增后送入AC）

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

LDAC5：AC<—BUS+0，（将总线上来的数据+0送入AC）

MOVR1：AC<—BUS+0，（将总线上来的数据+0送入AC）

由此可知，以上4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

//状态表达式

assign pcbus = fetch1||fetch3;

assign read = fetch2||ldac1||stac1||ldac2||stac2||ldac4||jump1||(z&&jmpz1)||(!z&&jpnz1)||jump2||(z&&jmpz2)||(!z&&jpnz2);

assign write = stac5;

assign pcinc = fetch2 ||ldac1||ldac2||stac1||stac2||(!z&&jmpz2)||(!z&&jmpz3)||(z&&jpnz2)||(z&&jpnz3);

//条件不满足，pcinc++跳过此跳转指令

assign pcload = jump3||(z&&jmpz3)||(!z&&jpnz3);

assign irload = fetch3;

assign rload = movac1;

assign busmem = stac5;

assign drlbus = ldac5||stac5;

assign acbus = movac1||stac4;

assign arload = fetch1||fetch3||ldac3||stac3;

assign rbus = add1||sub1||and1||or1||xor1||movr1;

assign arinc = ldac1||stac1||jump1||(z&&jmpz1)||(!z&&jpnz1);

assign trload = ldac2||stac2||jump2||(z&&jmpz2)||(!z&&jpnz2);

assign trbus = ldac3||stac3||jump3||(z&&jmpz3)||(!z&&jpnz3);

assign drhbus = ldac3||stac3||jump3||(z&&jmpz3)||(!z&&jpnz3);

assign zload = add1||sub1||and1||or1||xor1||inac1||clac1||not1||movr1||jump3||(z&&jmpz3)||(!z&&jpnz3)||ldac5||stac5;

assign acload = movr1||add1||sub1||and1||or1||xor1||inac1||clac1||not1||ldac5;

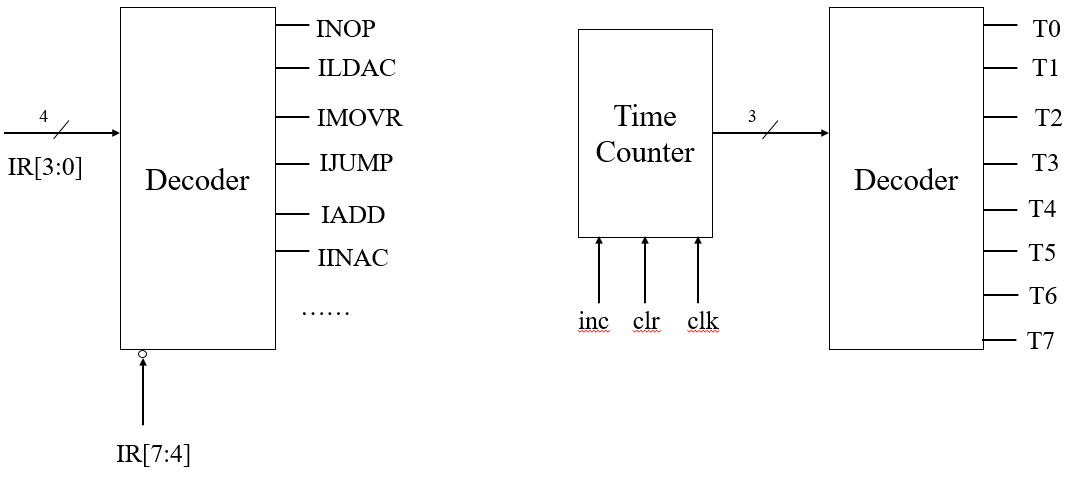
assign membus = fetch2||ldac1||stac1||ldac2||stac2||ldac4 ||jump1||(z&&jmpz1)||(!z&&jpnz1)||jump2||(z&&jmpz2)||(!z&&jpnz2);

assign drload = fetch2 ||ldac1||stac1||ldac2||stac2||ldac4||stac4||jump1||(z&&jmpz1)||(!z&&jpnz1)||jump2||(z&&jmpz2)||(!z&&jpnz2);

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0000xxxx所以只需要译码低4位就可以了，NOR高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（这是取指令的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。结构如下图：



图中经过译码后的指令分别叫做INOP、ILDAC、IMOVR、IJUMP、IADD、IINAC

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：LDAC的执行周期如下：

LDAC1=ILDAC^T3

LDAC2=ILDAC^T4

LDAC3=ILDAC^T5

LDAC4=ILDAC^T6

LDAC5=ILDAC^T7

典型指令的状态及其表达式如表7所示：（根据自行设计的具体指令进行添加）

表7 典型指令的状态及其表达式

|  |  |
| --- | --- |
| 状态 | 表达式 |
| Fetch1 | T0 |
| Fetch2 | T1 |
| Fetch3 | T2 |
| NOP1 | INOP^T3 |
| LDAC1 | ILDAC^T3 |
| LDAC2 | ILDAC^T4 |
| LDAC3 | ILDAC^T5 |
| LDAC4 | ILDAC^T6 |
| LDAC5 | ILDAC^T7 |
| MOVR1 | IMOVR^T3 |
| JUMP1 | IJUMP1^T3 |
| JUMP2 | IJUMP^T4 |
| JUMP3 | IJUMP^T5 |
| ADD1 | IADD^T3 |
| INAC1 | IINAC^T3 |

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 || LDAC5||MOVR1||JUMP3||ADD1||INAC1（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。（还是以典型指令为例，将前面取指周期和指令的执行周期都集中在表8中）。

表8 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 | AC<—AC+R | rbus、alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |
| INAC执行周期 | T3 | INAC1 | AC<—AC+1 | alus、acload、zload（如果需要z标志寄存器，则有该控制信号） |
| LDAC执行周期 | T3 | LDAC1 | DR<—M，PC<—PC+1，  AR<—AR+1 | read、membus、drload、pcinc、arinc |
| T4 | LDAC2 | TR<—DR，DR<—M，  PC<—PC+1 | trload、read、membus、drload、pcinc |
| T5 | LDAC3 | AR<—DR，TR | drhbus、trbus、arload |
| T6 | LDAC4 | DR<—M | read、membus、drload |
| T7 | LDAC5 | AC<—DR | drlbus、acload、alus |
| MOVR执行周期 | T3 | MOVR1 | AC <—R | rbus、acload、alus |
| JUMP执行周期 | T3 | JUMP1 | DR<—M，AR<—AR+1 | read、membus、drload、arinc |
| T4 | JUMP 2 | TR<—DR，DR<—M | trload、read、membus、drload |
| T5 | JUMP 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LDAC3，则其逻辑表达式为：

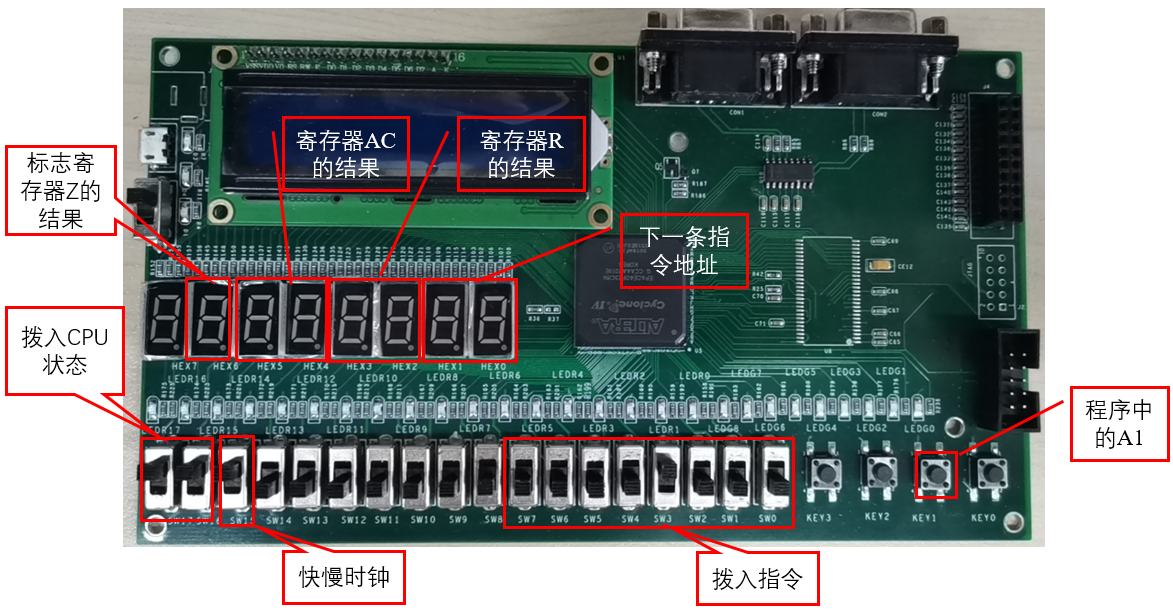
arload = Fetch1 || Fetch3|| LDAC3

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8中是典型指令，需要换成自行设计的指令！）

## (七)、在开发板上验证指令

### 1、开发板

开发板如下图所示：



### 2、步骤

CPU分为三个状态：IN状态，CHECK状态和RUN状态。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。

例如验证如下指令：

0： NOP 00001010

1： INAC 00001010

验证多条指令的方法过程是一样的。步骤如下：

（1）选择演示时钟：拨弄SW15，上为快速演示，下为慢速演示。

（2）输入指令

将SW17置0，SW16置1，按下KEY0复位。在开关上置第一条程序代码 00000000，按下KEY1将该指令存入地址 0000 0000 中，继续在数据开关上输入下一条指令00000001，按 KEY1 进行指令存储，依次输入上述程序。

（3）检查指令

将SW17置1，SW16置0，按 KEY0复位，按 KEY1 观察数据灯显示的值是否为保存的指令，若不正确，返回第一步重新输入正确指令，若正确则进入下一步。

（4）运行指令

按 KEY0 复位，然后将SW17置1，SW16置1，按 KEY1按键一条一条进行执行。观察数码管的显示。

### 3、指令输入及结果显示

#### （1）指令输入：

SW0-7：输入开关

KEY1：输入按钮

KEY0：清零按钮

SW17 、SW16：状态输入

* IN： SW17=0，SW16=1
* CHECK： SW17=1，SW16=0
* RUN： SW17=1，SW16=1

SW15：快慢时钟选择，上拨为快时钟，下拨为慢时钟

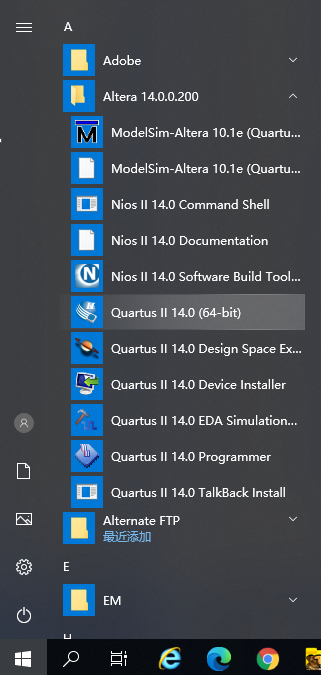
#### （2）结果显示：

* LEDR17，LEDR16：当前状态显示灯
* IN： LEDR17灭， LEDR16亮
* CHECK： LEDR17亮， LEDR16灭
* RUN： LEDR17亮， LEDR16亮
* LEDR15：表示当前演示的快慢时钟。灭：慢时钟；亮：快时钟
* HEX1，HEX0：下一条执行指令的地址
* HEX3，HEX2：通用寄存器R的结果显示
* HEX5，HEX4：通用寄存器AC的运行结果显示
* HEX6：标志寄存器Z的值（没有条件转移指令的话，该值可以没有）

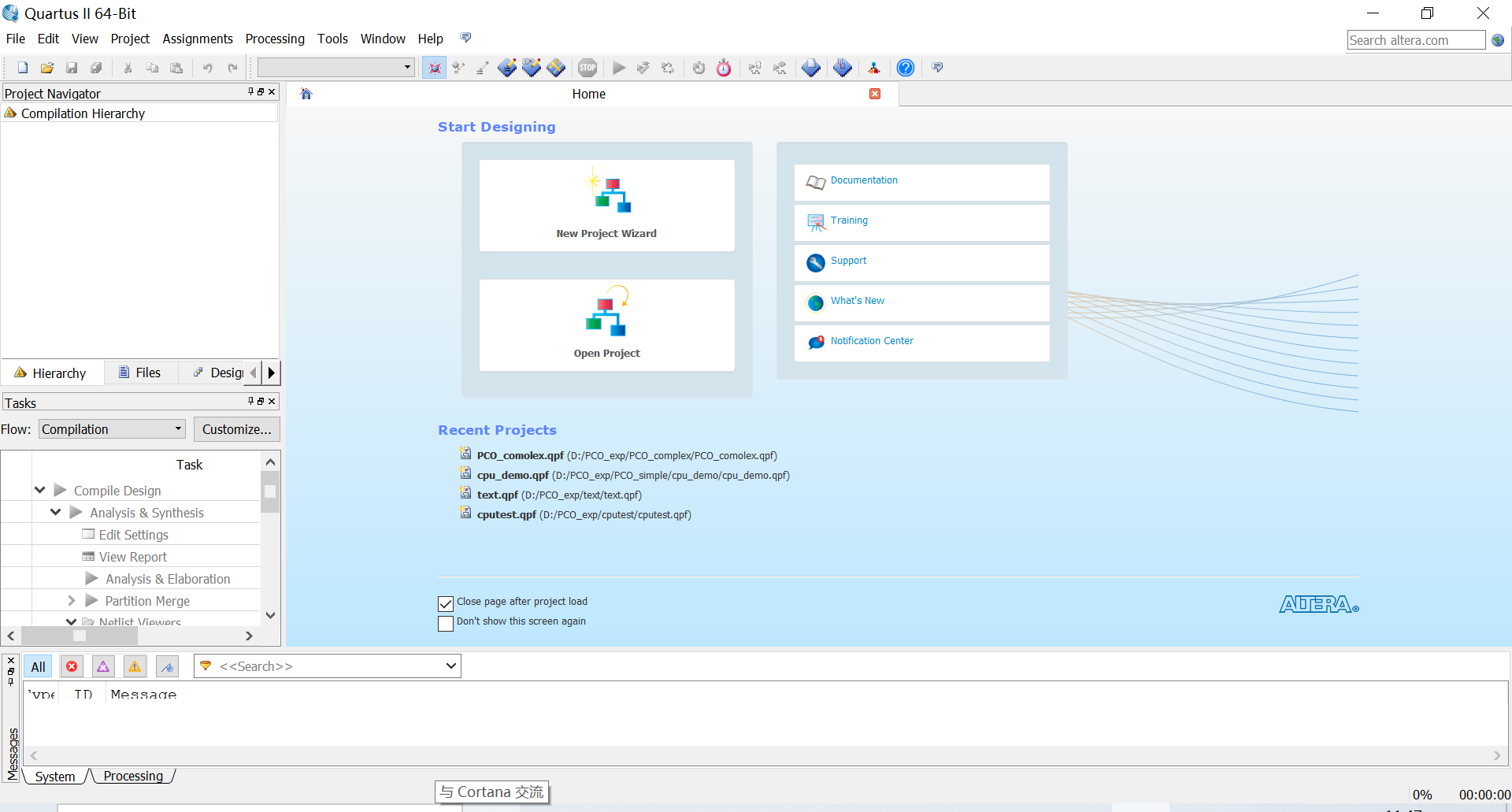
# 二、Quartus II使用说明及步骤

## (一)、建立工程项目并导入.v文件

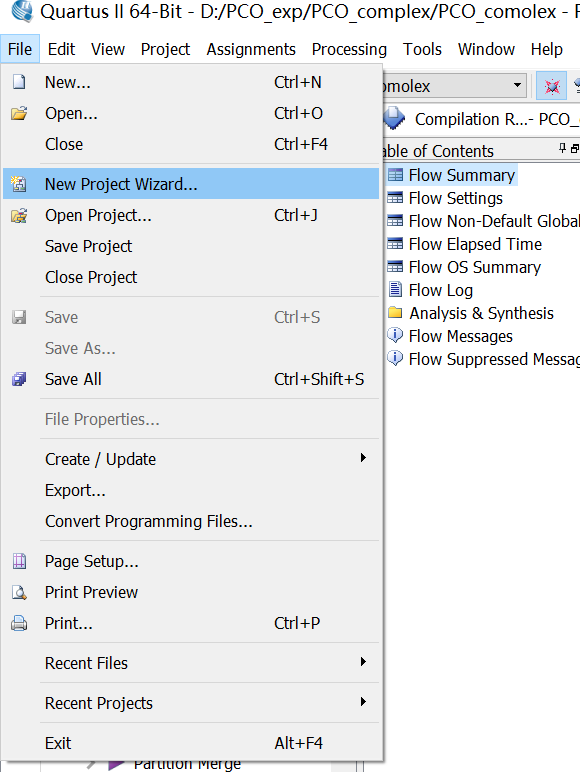
1、点击“开始”菜单，选择“Altera 14.0.0.200”—>“Quartus II 14.0（64-bit）”，如下图所示（以T09机房为例，其他机房系统不一样请自行查找该软件）。



打开Quartus II软件，界面如下图所示。

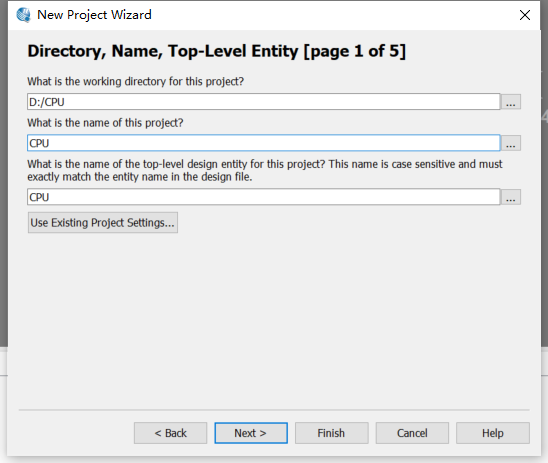


2、新建工程项目。点击上方菜单栏“File”->“New Project Wizard...”进行新建项目工程操作，界面如下图所示。

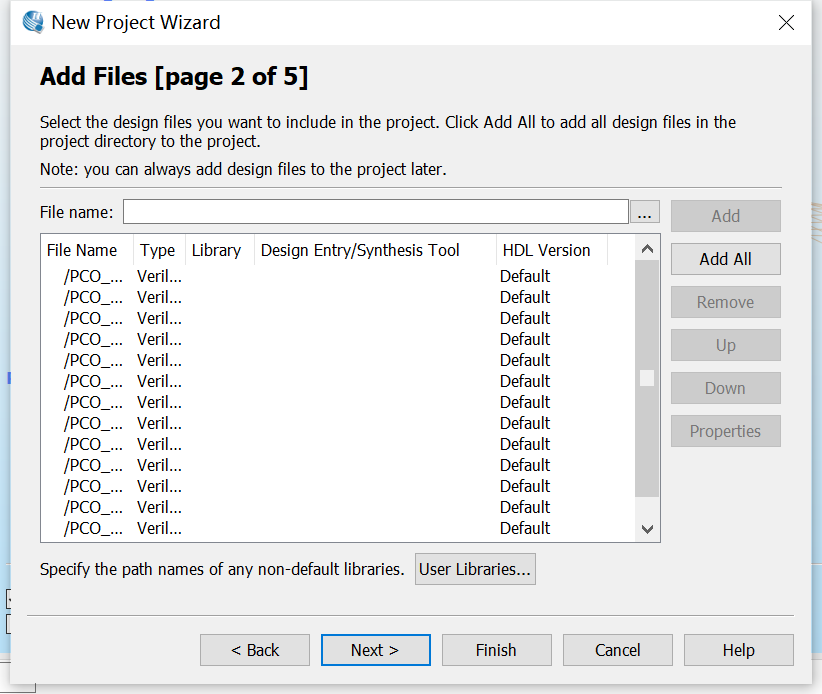


注意：如果已有工程文件（一定是全英文路径），则在上图中选择“Open Project…”，找到相应的工程文件（扩展名为qpf）打开即可。

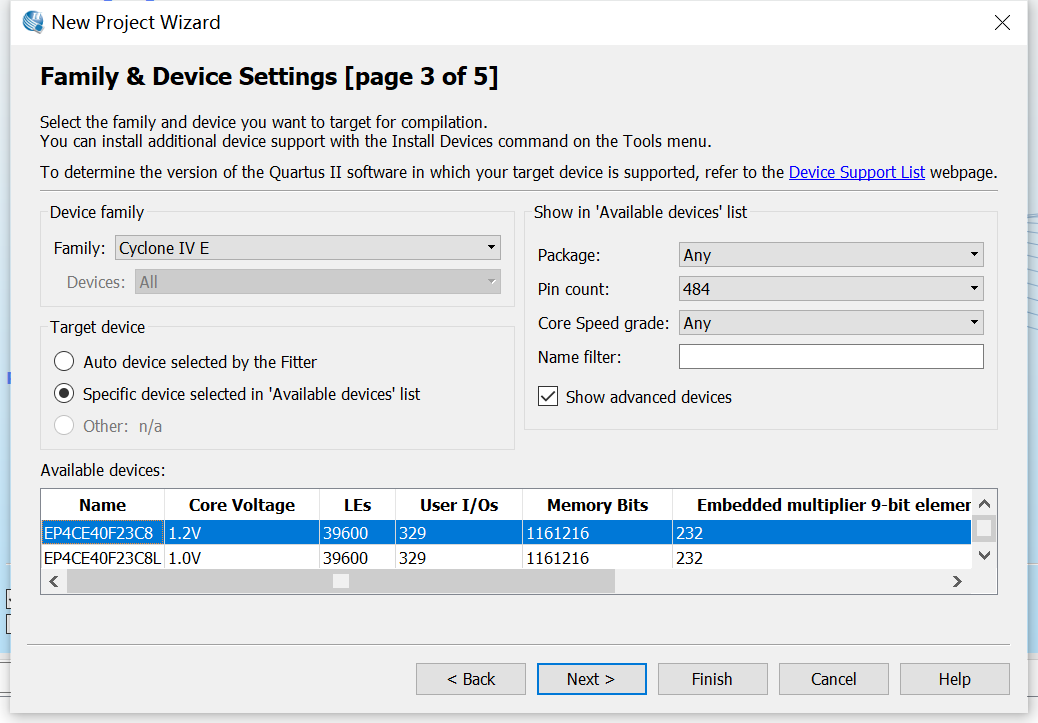
3、为工程安排存储路径，并为工程起名(全英文路径)，完成后点击“next”。界面如下图所示。注意：将工程命名为姓名拼音首字母缩写（例如张三，这里的工程名字就叫做zs）



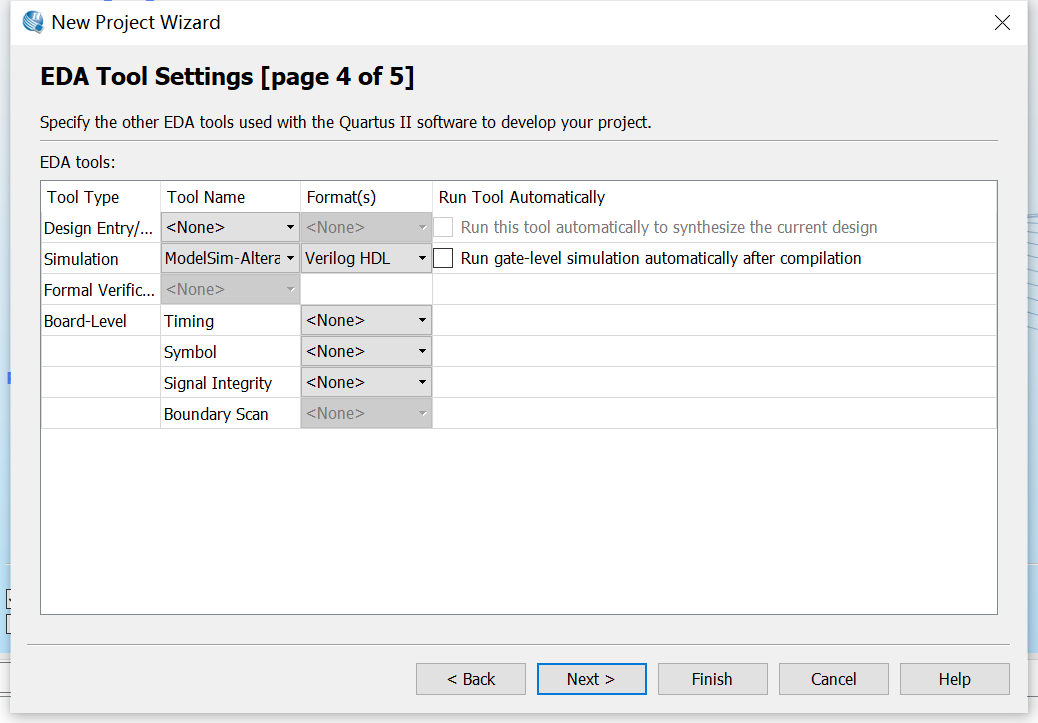
4、将从FTP上下载的.v文件导入到里面后（存放.v文件的路径必须为全英文），导入文件后的界面如下图所示，点击“next”。

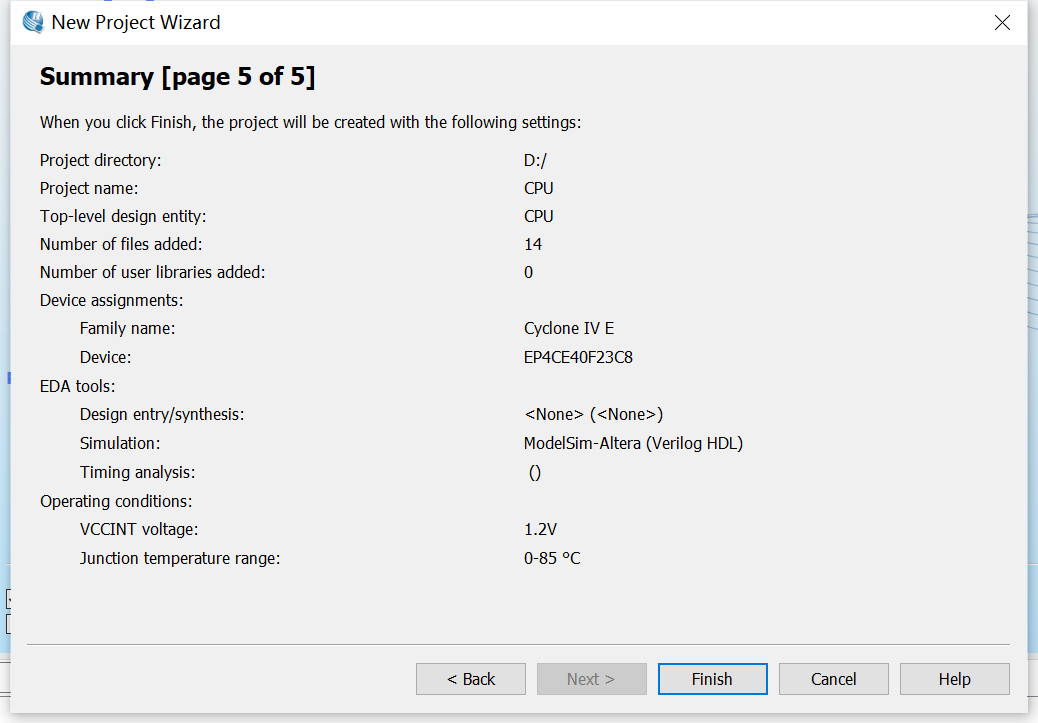


5、选择该工程的设备。将“Family”选为Cyclone IV E，“Pin Count”选为484，在“Available devices”的列表中，找到EP4CE40F23C8并选择后，点击“next”。界面如下图所示。

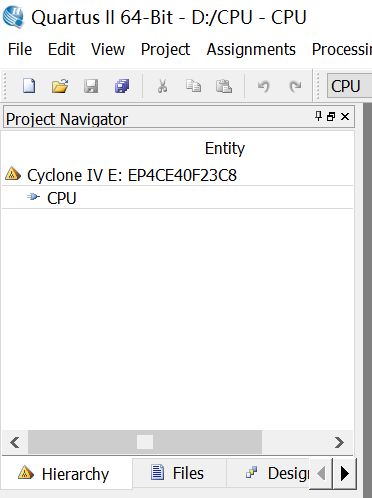


6、将标签为“Simulation”行的第二个选项框选择为Verilog HDL后，点击“next”，再点击“Finish”。界面如下图所示。

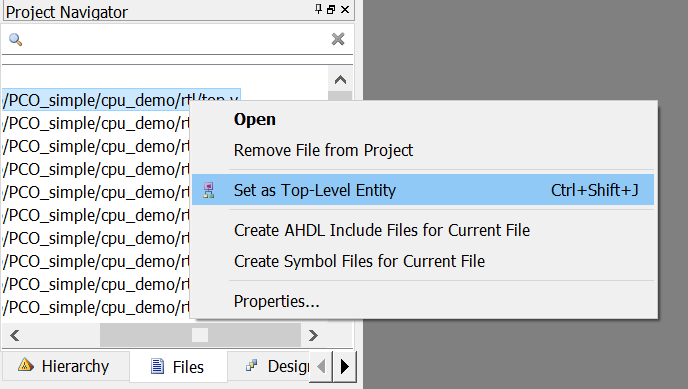




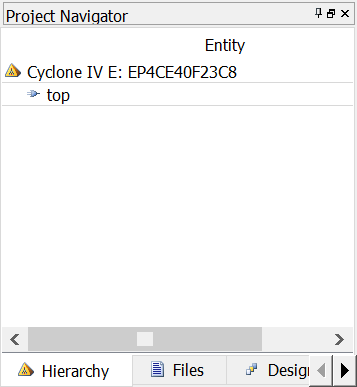
7、点击“Finish”后，在左上角可以看到“Project Navigator”界面，在里面可以看到当前的设备型号“Cyclone IV E：EP4CE40F23C8”和顶层模块“CPU”。界面如下图所示。



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。

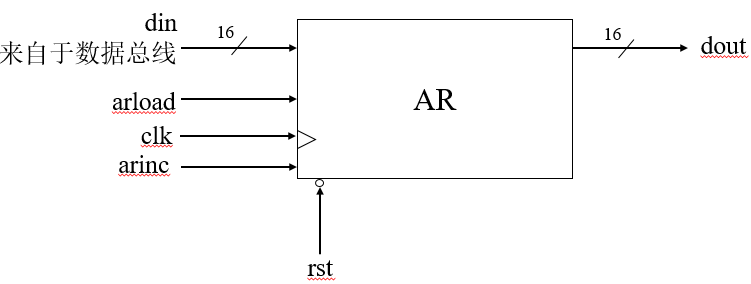


## (二)、示例程序中的CPU模块

### 1、地址寄存器——ar.v

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。

### 3、状态转换模块——CPU\_Controller.v

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、控制模块——control.v

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21个：read, write, pcload, pcinc, pcbus, drhbus, drlbus, arload, arinc, drload, acload, acbus, alus, irload, trload, trbus, zload, rload, rdbus, 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

### 6、启停电路——qtsj.v

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；

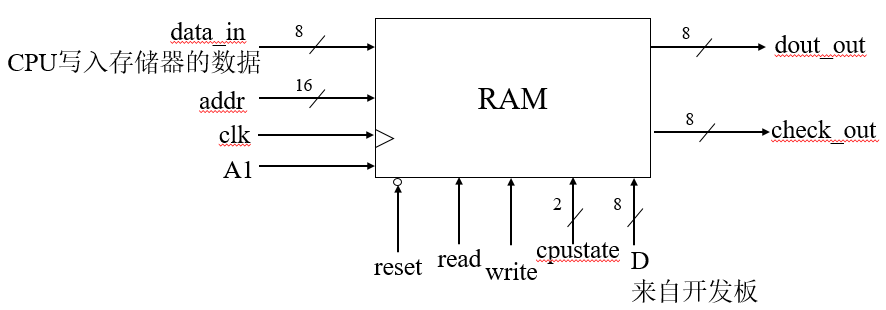
### 7、分频模块——clk\_div.v

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。

### 4、存储器——ram.v

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，由实验开发板上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，将指令存在内部寄存器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。

### 8、显示输出模块——light\_show.v

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。

### 9、CPU模块——cpu.v

功能：将各模块（qtsj, ar, pc, dr, tr, ir, r, ac, 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，但是如果需要tr的话，补充dbus接收tr的输出trdbus。

ar mar(.din(dbus),.clk(clk\_choose),.rst(rst),.arload(arload),.arinc(arinc),.dout(addr));

//pc(din, clk, rst,pcload, pcinc, dout);pc实例化

pc mpc(.din(dbus),.clk(clk\_choose),.rst(rst),.pcload(pcload),.pcinc(pcinc),.dout(pcdbus));

//dr(din, clk,rst, drload, dout);补充dr实例化语句

dr mdr(.din(dbus[7:0]),.clk(clk\_choose), .rst(rst), .drload(drload), .dout(drdbus));

//tr(din, clk,rst, trload, dout);补充tr实例化语句，如果需要tr的话

tr mtr(.din(drdbus),.clk(clk\_choose), .rst(rst), .trload(trload), .dout(trdbus));

//ir(din,clk,rst,irload,dout);补充ir实例化语句

ir mir(.din(drdbus),.clk(clk\_choose), .rst(rst), .irload(irload), .dout(irout));

//r(din, clk, rst,rload, dout);补充r实例化语句

r mr(.din(dbus[7:0]),.clk(clk\_choose), .rst(rst), .rload(rload), .dout(rdbus));

//dbus[7:0] rdbus

//ac(din, clk, rst,acload, dout);补充ac实例化语句

ac mac(.din(aluout),.clk(clk\_choose), .rst(rst), .acload(acload),.busac(busac),.dout(acdbus));

//alu(alus,ac, bus, dout);补充alu实例化语句

alu malu(.alus(alus),.ac(acdbus) , .bus(dbus[7:0]), .dout(aluout));

//z(din,clk,rst, zload,dout);补充z实例化语句，如果需要的话

z mz(.din(aluout),.clk(clk\_choose), .rst(rst),

.zload(zload), .dout(zout));

### 10、顶层模块——top.v

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

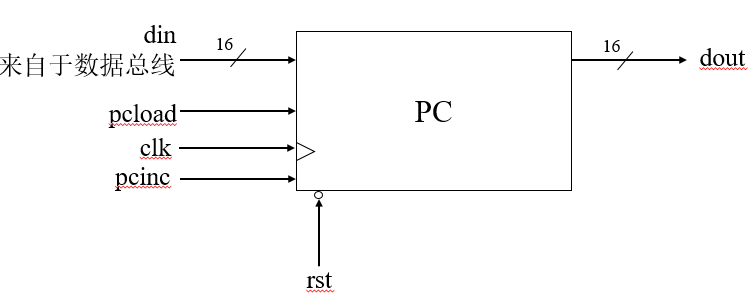
### 11、其他模块——根据需要自行设计，示例工程中给出了只有模块说明的代码，需要自行编写代码

### 12、testbench文件——根据需要进行修改

### 2、程序计数器——pc.v

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；

1. ac:

module ac(din, busin, clk, rst, acload, busac, dout);

//din:从alu来的数据

//busin:从总线来的数据

input [7:0] din;

input [7:0] busin;

input clk,rst,acload,busac;

output reg [7:0] dout;

always @(posedge clk or negedge rst)

begin

if (!rst)

dout <= 0;

else if (acload)

dout <= din;

else if (busac)

dout <= busin;

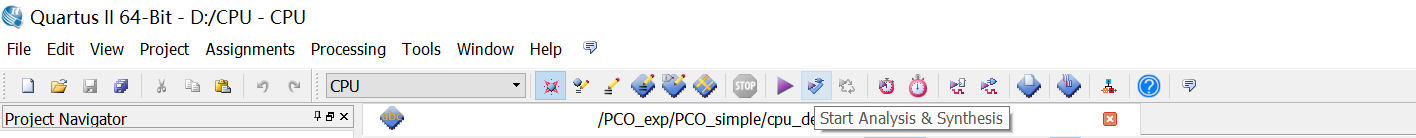
end

endmodule

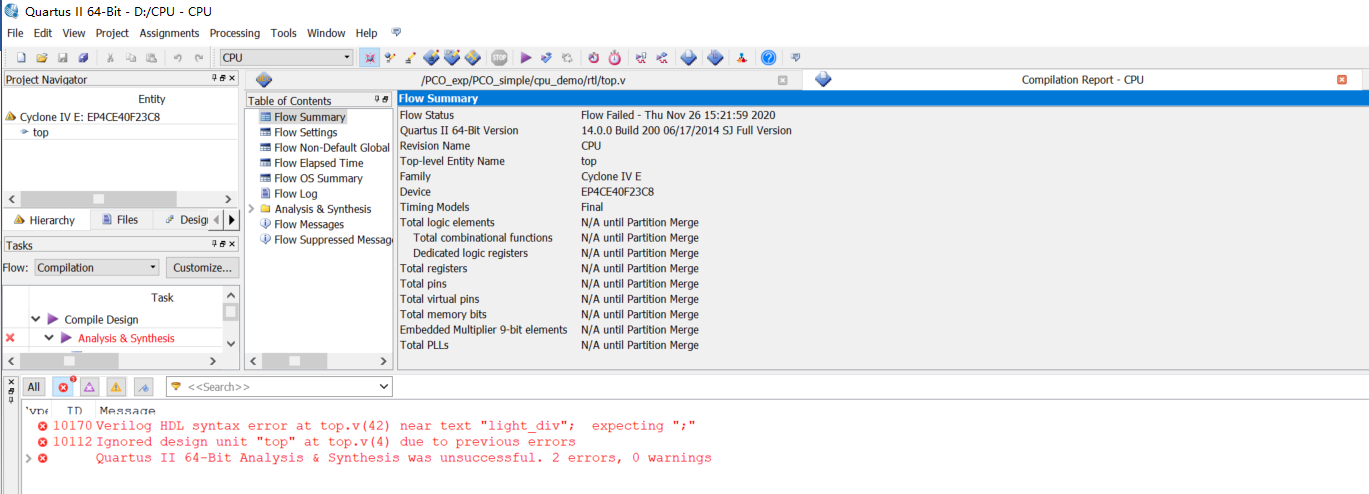
## (三)、编写代码和测试文件并进行仿真

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。

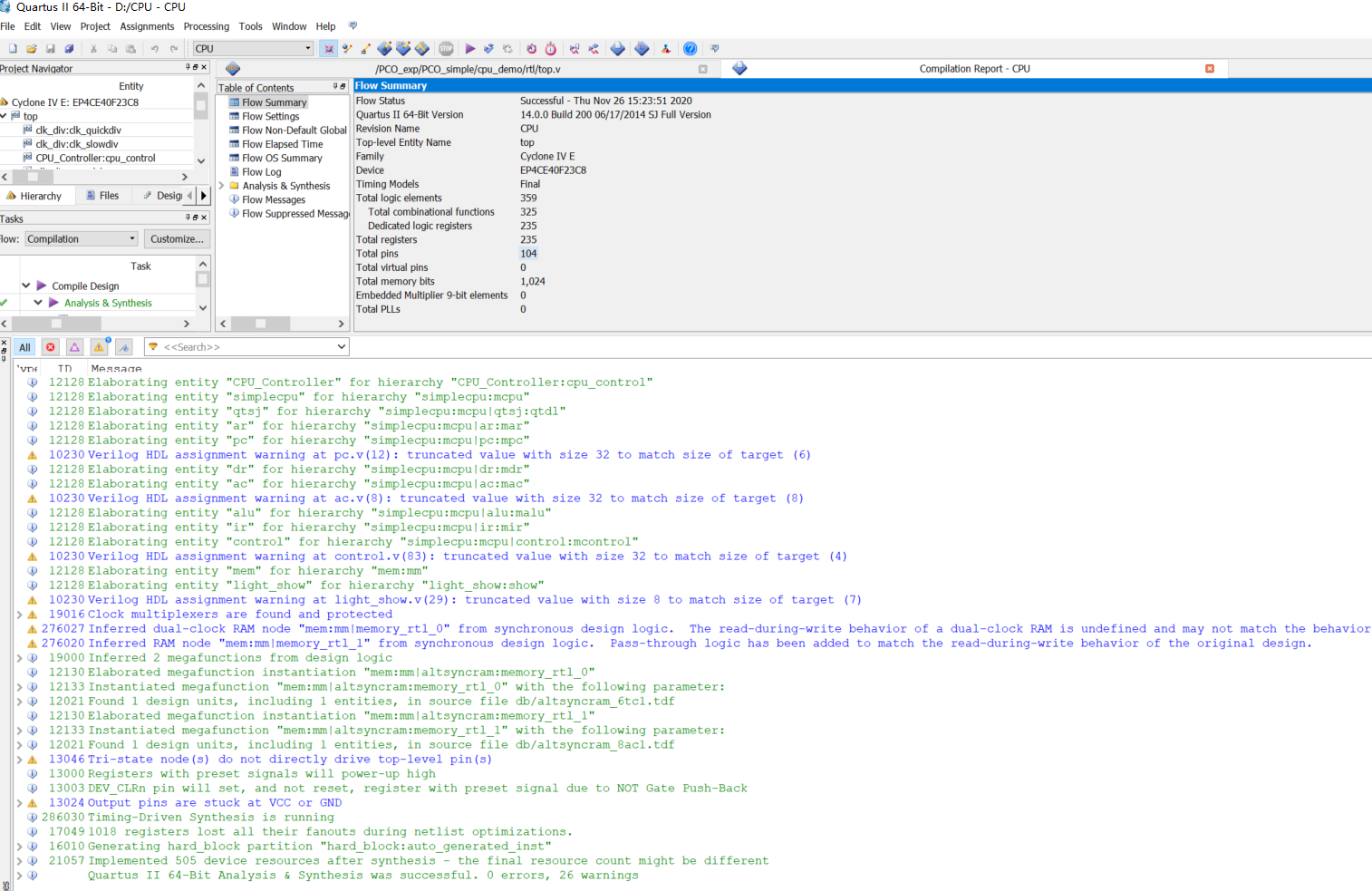
“Analyzes and synthesizes”按钮位置如下图所示。



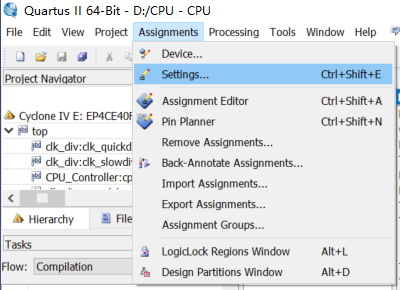
2、报错界面如下图所示。



3、无报错，分析成功后界面如下图所示。



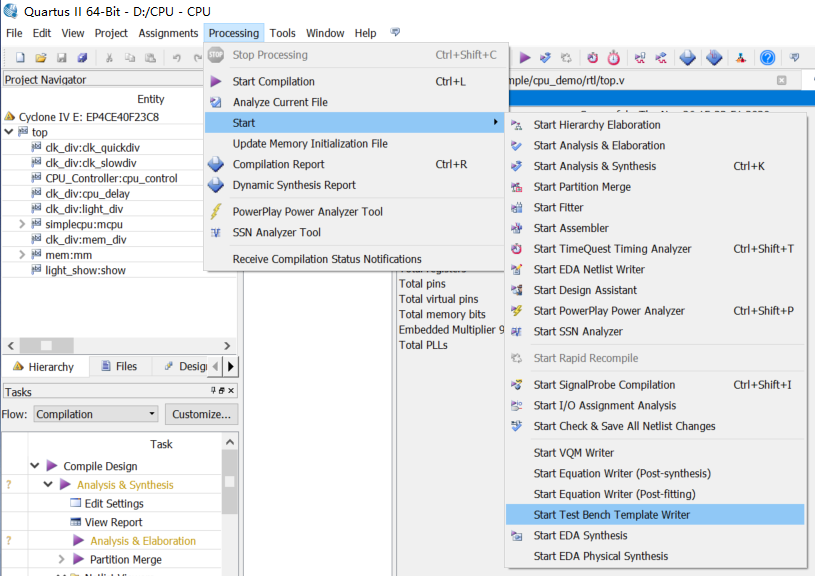
4、在界面顶部菜单点击“Assignments”—>“Settings”



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。



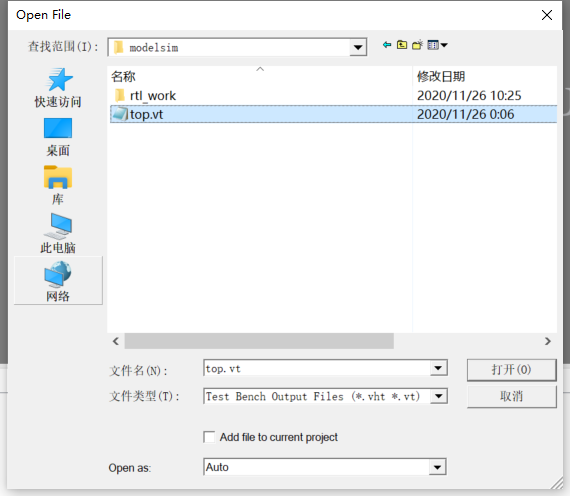
6、生成testbench仿真文件。分析排查无误后，在界面顶部菜单点击“Processing”—>“Start”—>“Start Test Bench Template Writer”生成testbench仿真测试模板文件。



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

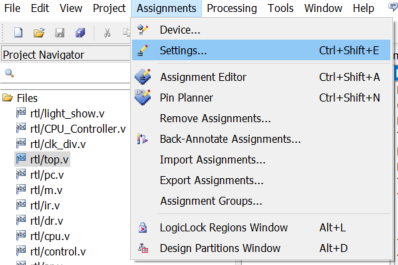


8、打开生成的testbench仿真文件。打开“存放工程项目的路径/simulation/modelsim/top.vt”，具体界面如下图所示，注意文件类型。

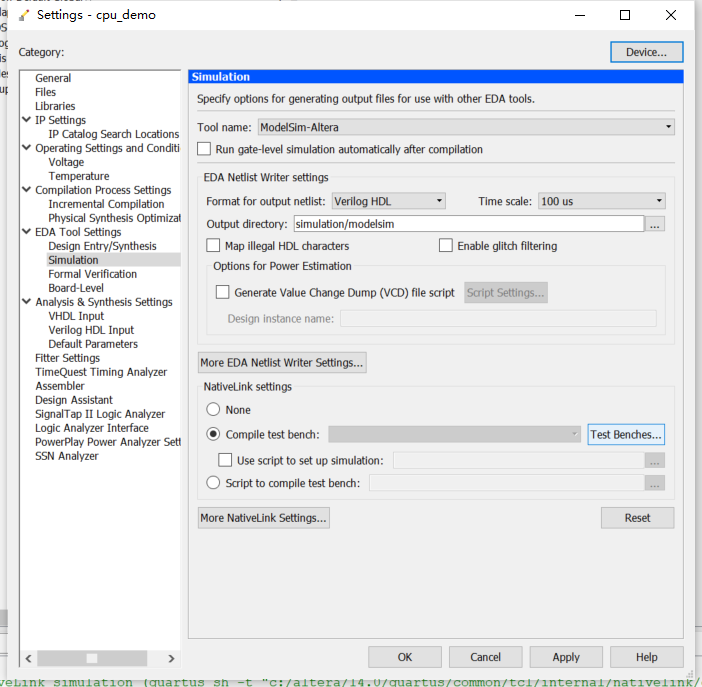


9、打开后将FTP中所给的.vt文件中内容全部复制到当前仿真文件之中，保存。注意：FTP中的.vt文件是nop指令的仿真文件，需要修改成自己所设计指令的仿真文件。

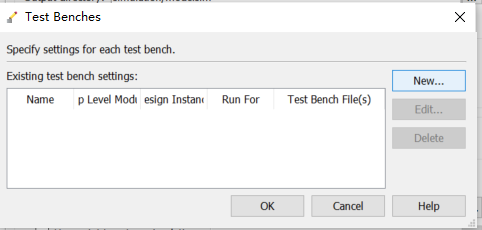
10、设置quartus工程仿真文件。点击顶部菜单选项“Assignments”—>“Settings”，点击左侧选项栏中的“Simulation”。



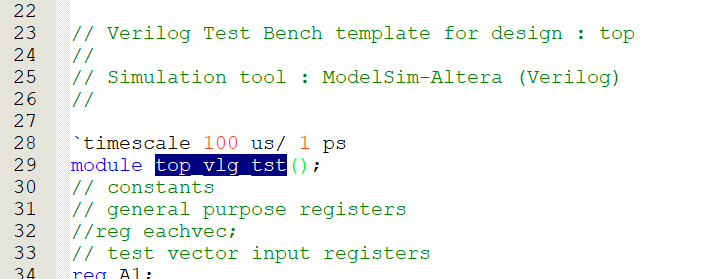
11、将右侧“NativeLink settings”由“None”选择为“Compile test bench”，点击“Test Benches…”选项。

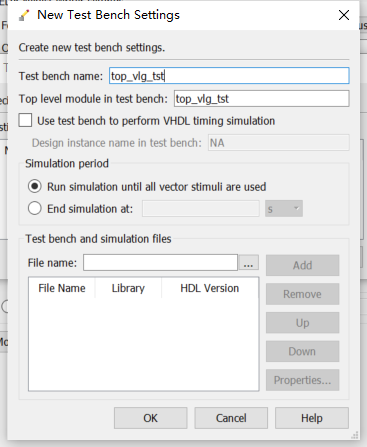


12、在“Test Benches”中点击“New”选项。

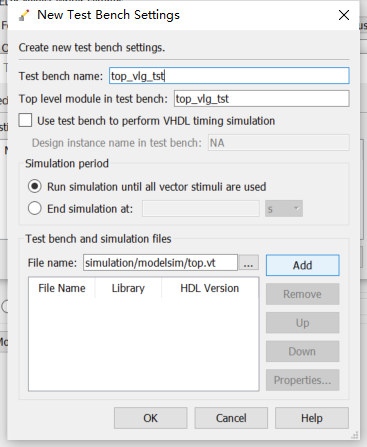


13、在“Test bench name”中起名（与testbench中的实例模块同名，一般都为top\_vlg\_tst），在File name在添加路径（路径为top.vt文件的存储路径）。

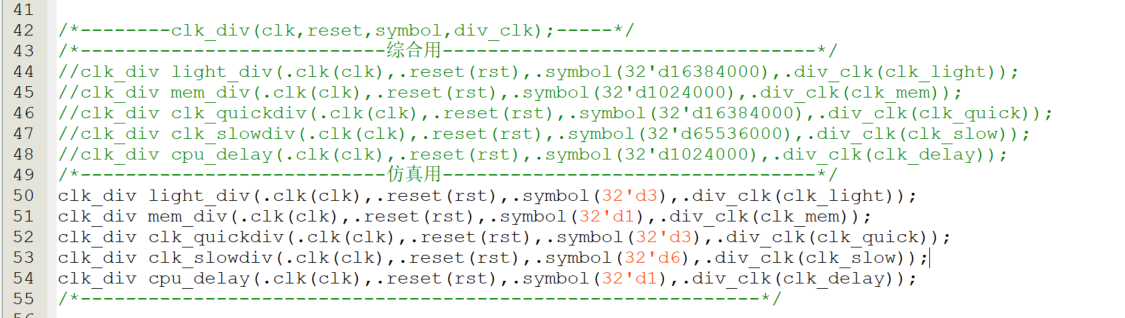


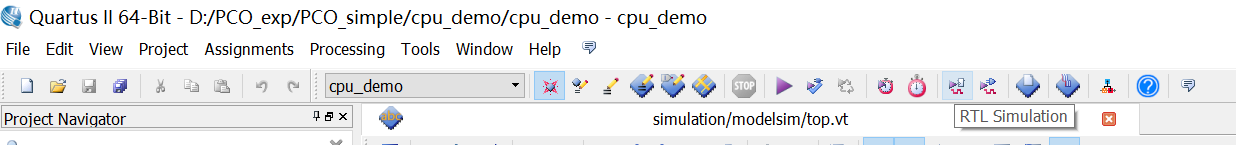


14、点击“Add”—>“ok”，一路全点“ok”。

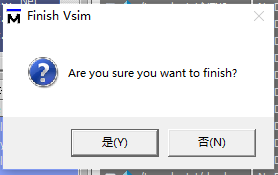


15、仿真。在“top.v”文件中，将综合部分注释掉，将仿真部分解除注释后，保存。点击分析，无报错后，点击界面顶部的“”按钮，进行“RTL simulation”仿真。



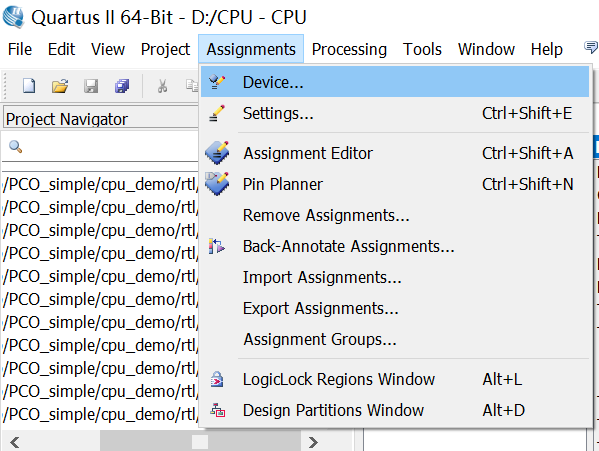


16、仿真波形显示。进入ModelSim界面，对弹出框“Finish Vsim”选择“否”，观察波形是否符合此次课程设计CPU的结果。若不符合则继续修改代码直到仿真波形与预期相同。

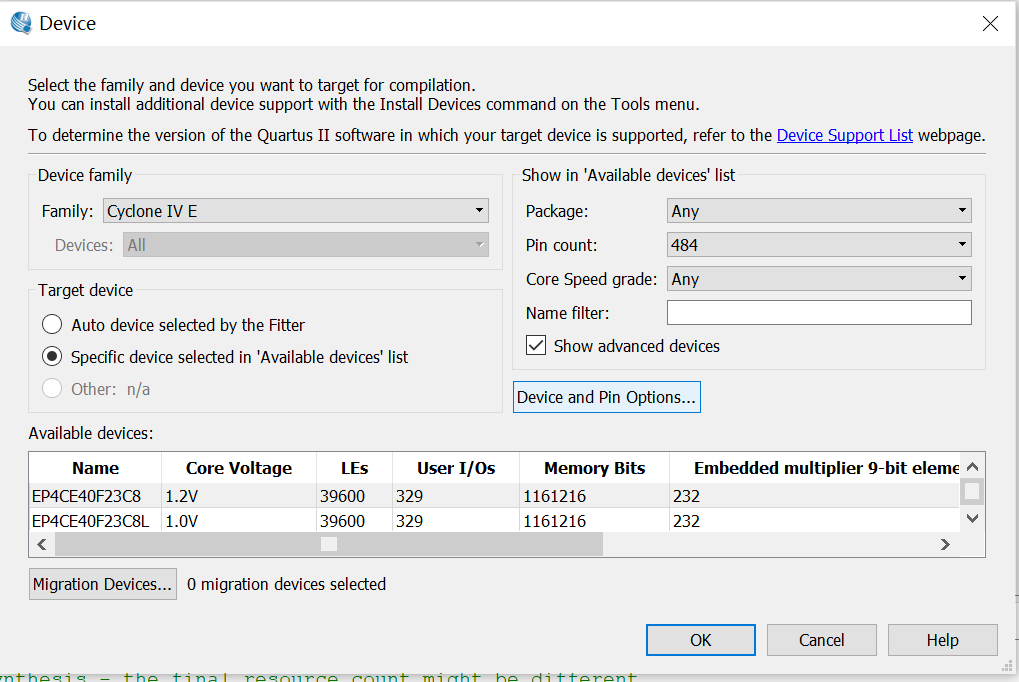


## (四)、Quartus部分参数设置及USB驱动安装

1、设置生成pof文件。综合后，需要生成pof文件才能被Quartus软件识别并正确烧到实验开发板中，所以需要在Quartus中设置相关内容——点击顶部菜单栏中“Assignments”->“Device”，如下图所示：



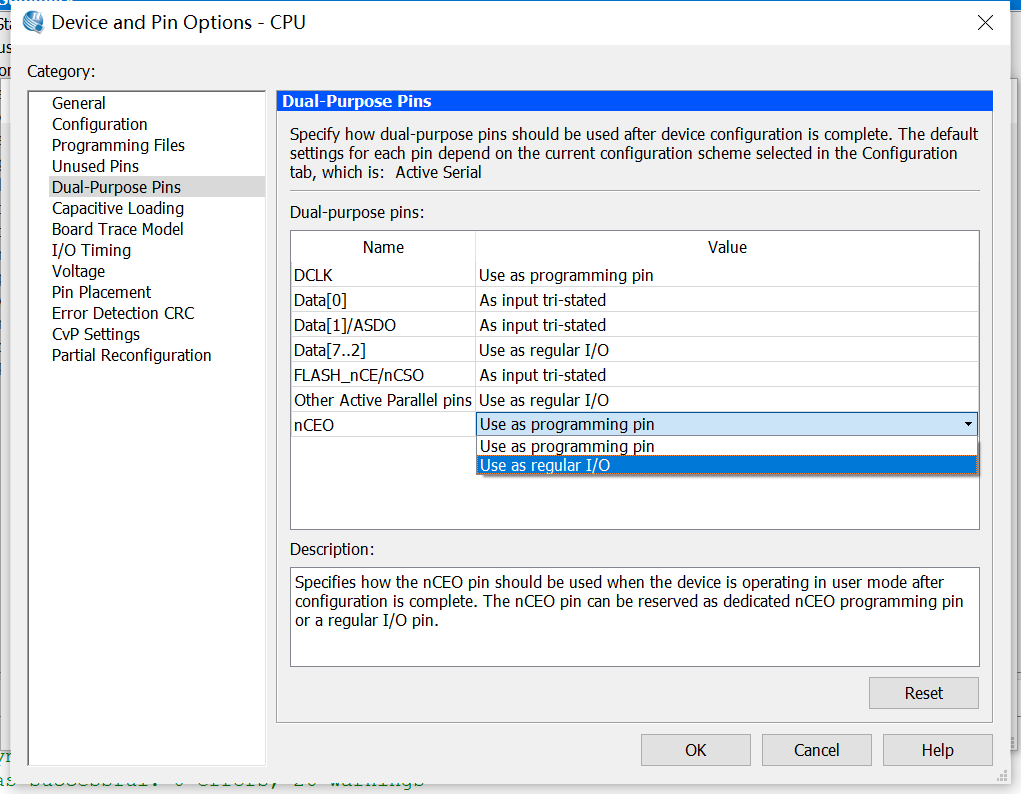
进入“Device”界面后，点击“Device and Pin Options...”。界面如下图所示。



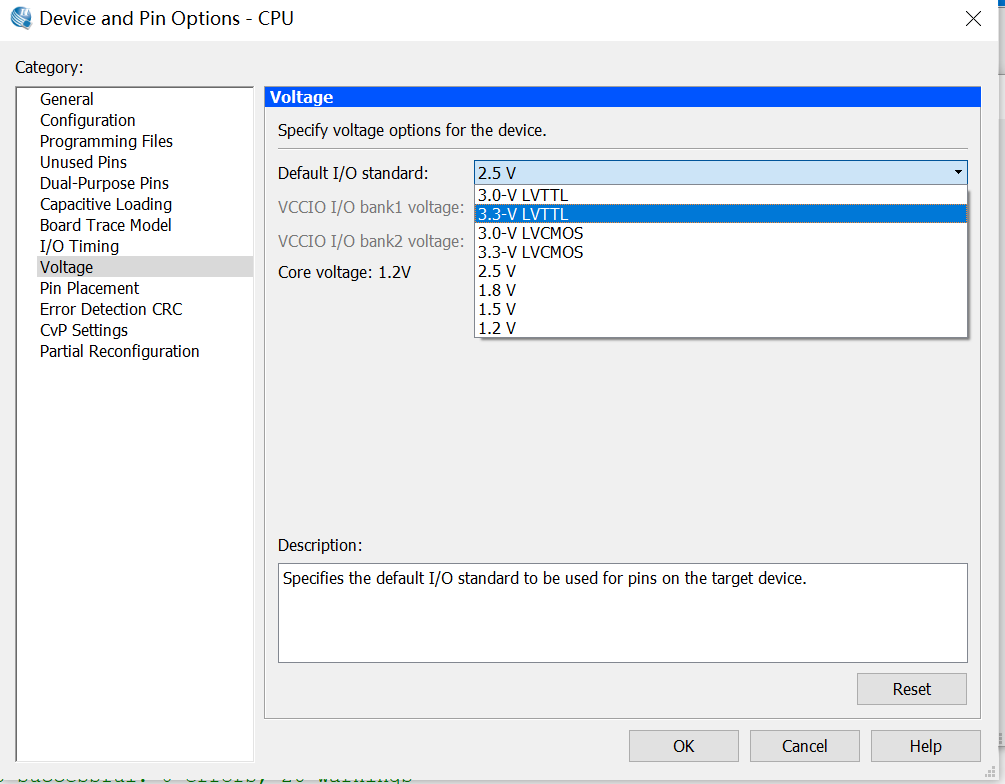
2、进入“Device and Pin Options...”界面后，在其左侧的选项栏中选择“Configuration”，选择右侧显示出的选项栏中的“Use configuration device”，在下拉菜单中选择“EPCS64”，点击“ok”。界面如下图所示。



3、将部分端口指定为输入输出类型。绑定引脚时，Quartus默认一些端口为专用引脚（如nCEO），如果不指定其为常规I/O引脚，那么在绑定引脚后的综合操作时会报错，所以需要设置常规的I/O引脚。在“Device and Pin Options”界面的左侧选项栏中选择“Dual-Purpose Pins”，将右侧显示出的选项栏中的“nCEO”一栏改选为“Use as regular I/O”，点击“ok”。界面如下图所示。“nCEO”默认是“Use as programming pin”。双击它，出现下拉的箭头，然后进行选择即可。



4、设置引脚的默认电压。该设备引脚默认为3.3V LVTTL。在“Device and Pin Options”界面的左侧选项栏中选择“Voltage”，在右侧“Default I/O standard”的下拉菜单中选择3.3-V LVTTL。界面如下图所示。

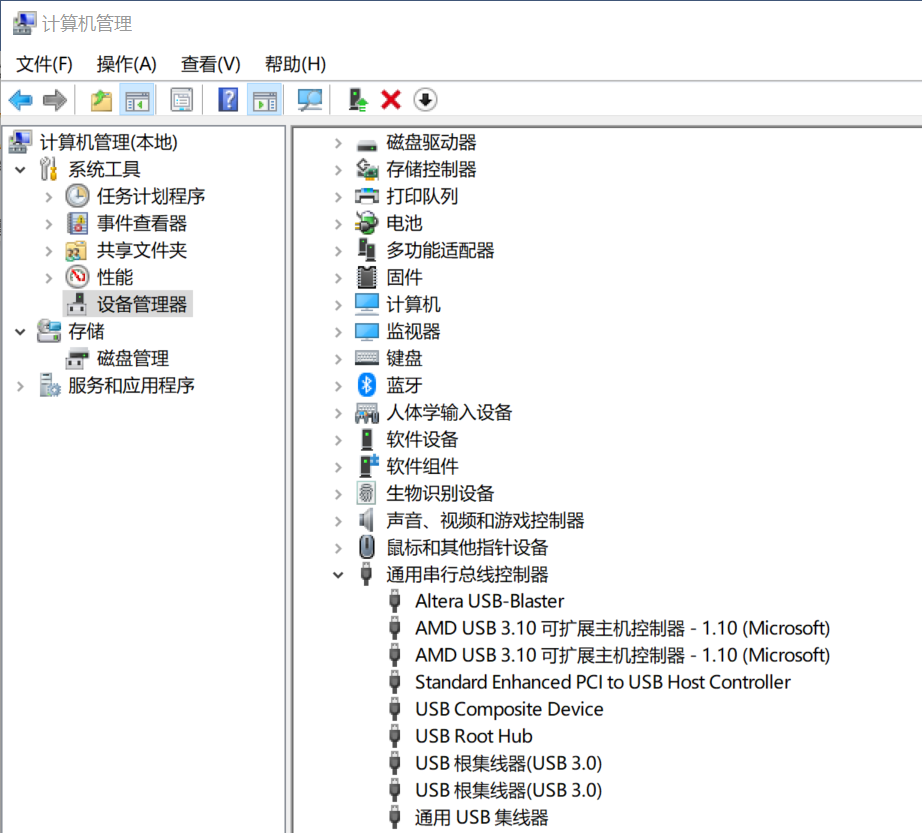


5、安装USB驱动。在实验开发板中烧录代码进行验证时，为了让计算机成功识别到实验开发板已经连接到USB端口，需要安装对应的USB驱动。

将实验开发板插入到电脑USB端口后，在电脑“桌面”中选择“计算机”，右键打开“管理”，进入“设备管理器”，找到带黄色叹号的图标“USB-Blaster”，双击后选择“浏览计算机查找驱动软件”，驱动软件的位置位于“C:\altera\14.0\quartus\drivers\usb-blaster”，点击“下一步”进行安装，安装后重启Quartus II。

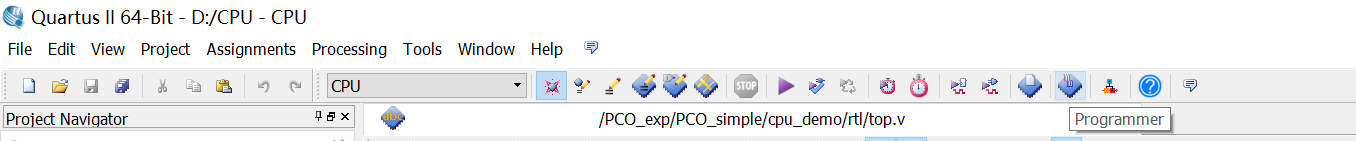
注：一定不能是usb-blaster的子文件夹（如C:\altera\14.0\quartus\drivers\usb-blaster-ii\x32或C:\altera\14.0\quartus\drivers\usb-blaster-ii\x64），否则会提示找不到驱动程序。



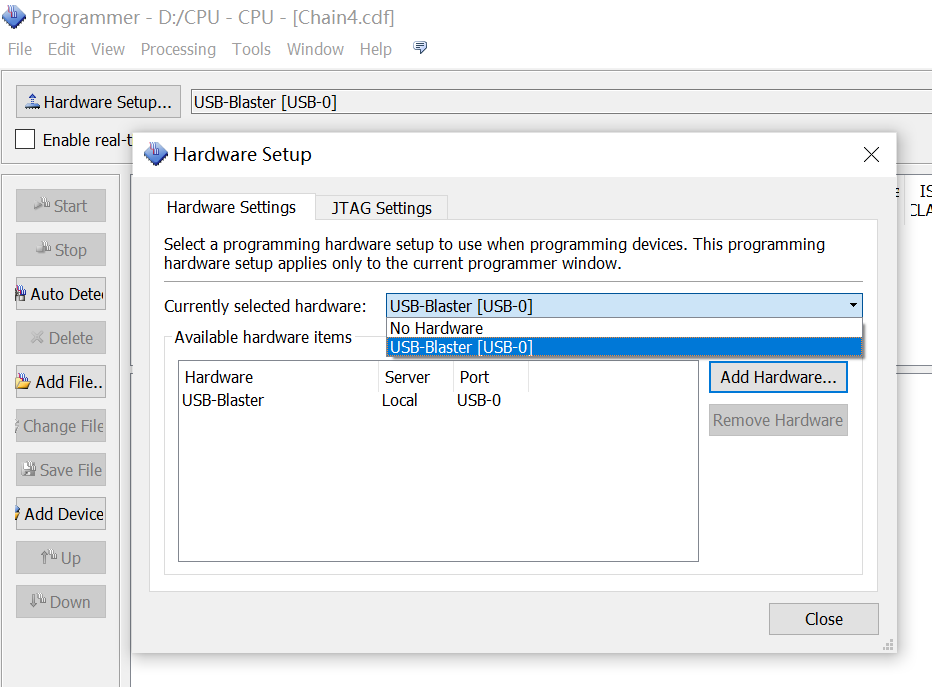




6、验证驱动是否已经装上：连接好实验开发板，打开Quartus II，点击界面上方的“”按钮，进入“Programmer”界面。



7、进入“Programmer”界面后，点击左上方“Hardware Setup”，在点击后出现的界面中下拉“Currently selected hardware”菜单，看是否出现“USB-Blaster [USB-0]”，如果出现，则安装成功。

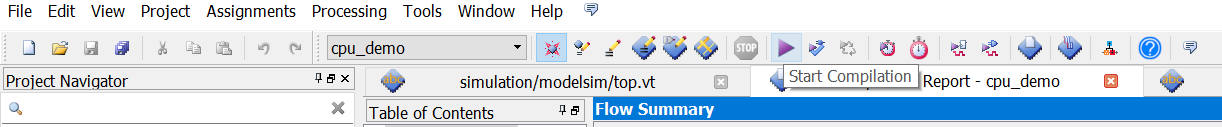


## (五)、综合

1、在“top.v”文件中，将仿真部分注释掉，将综合部分解除注释后，保存。点击分析。



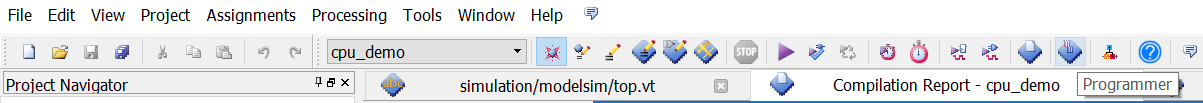
2、分析无报错后，代码分析无报错且引脚绑定好，点击顶部菜单栏的“”进行“Compilation”综合。注：综合时间较长，请耐心等待数分钟



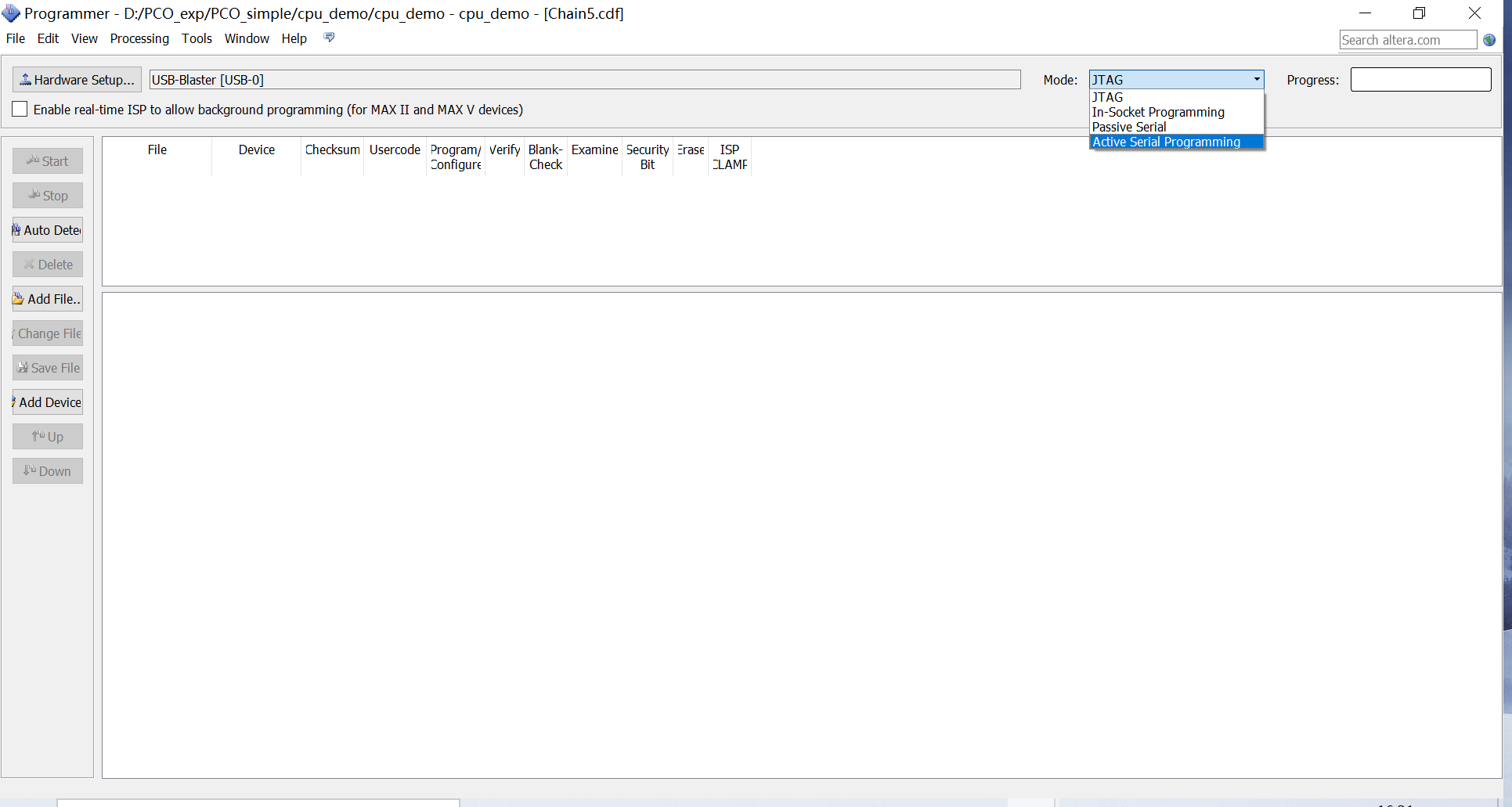
## (六)、烧写文件及实验开发板验证

1、连接实验开发板，先插USB数据线，后插电源线。

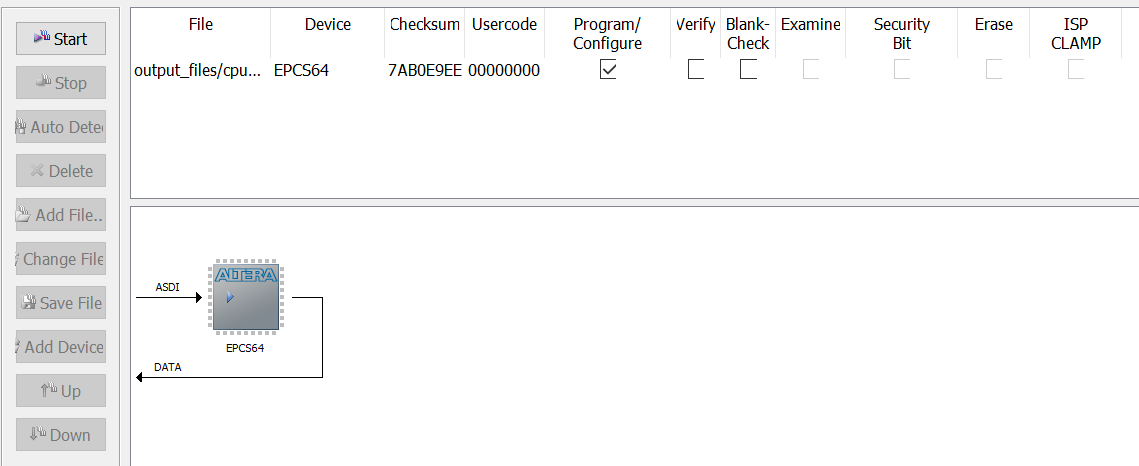
2、综合完毕后，点击顶部菜单栏的“”



3、进入“Programmer”界面，确认左上部分为USB-Blaster后，将右侧的“Mode”下拉菜单选为“Active Serial Programming”



4、在“Programmer”界面中，点击左侧“Add File”，选择“存放工程项目路径/output\_files/xxx.pof文件”。勾选“Program/Configure”选项，点击“Start”进行烧写。

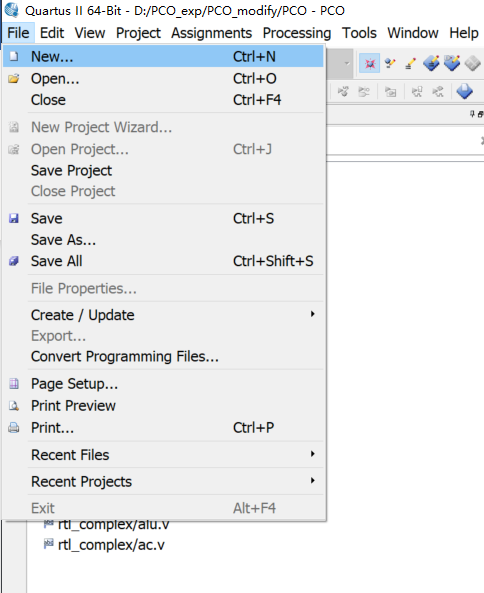


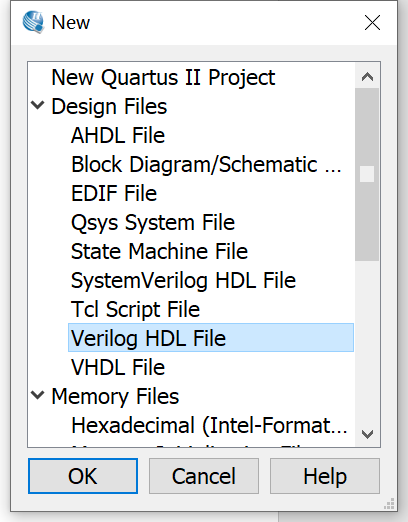
5、验证CPU是否能成功运行。

## (七)、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”。

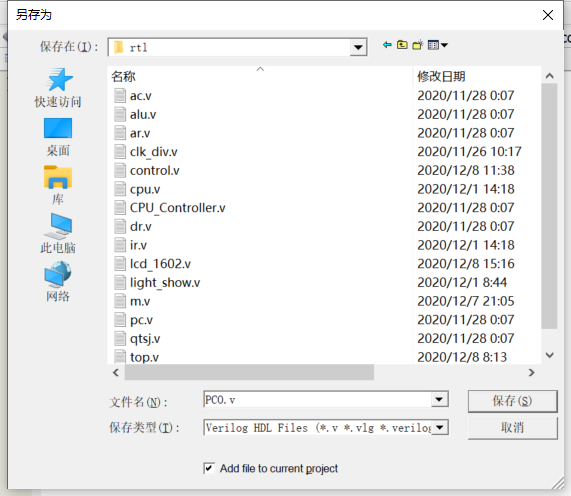




点击后，可在首页文件栏中出现“Verilogx.v”。



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。

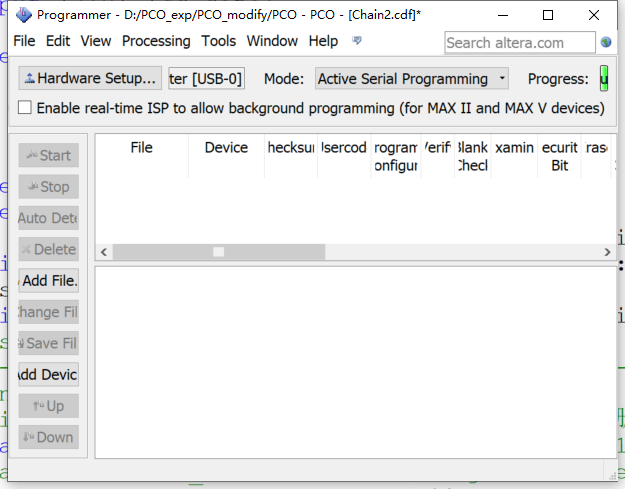


保存成功后，即可在左侧Files选项栏中看到文件列表中看到刚才保存的具体.v文件。

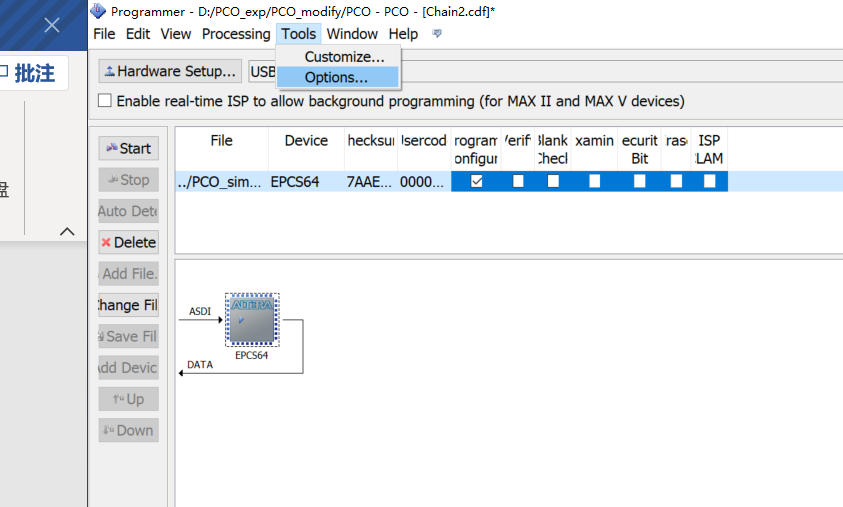


### 2、烧写时出现failed,不能成功烧入

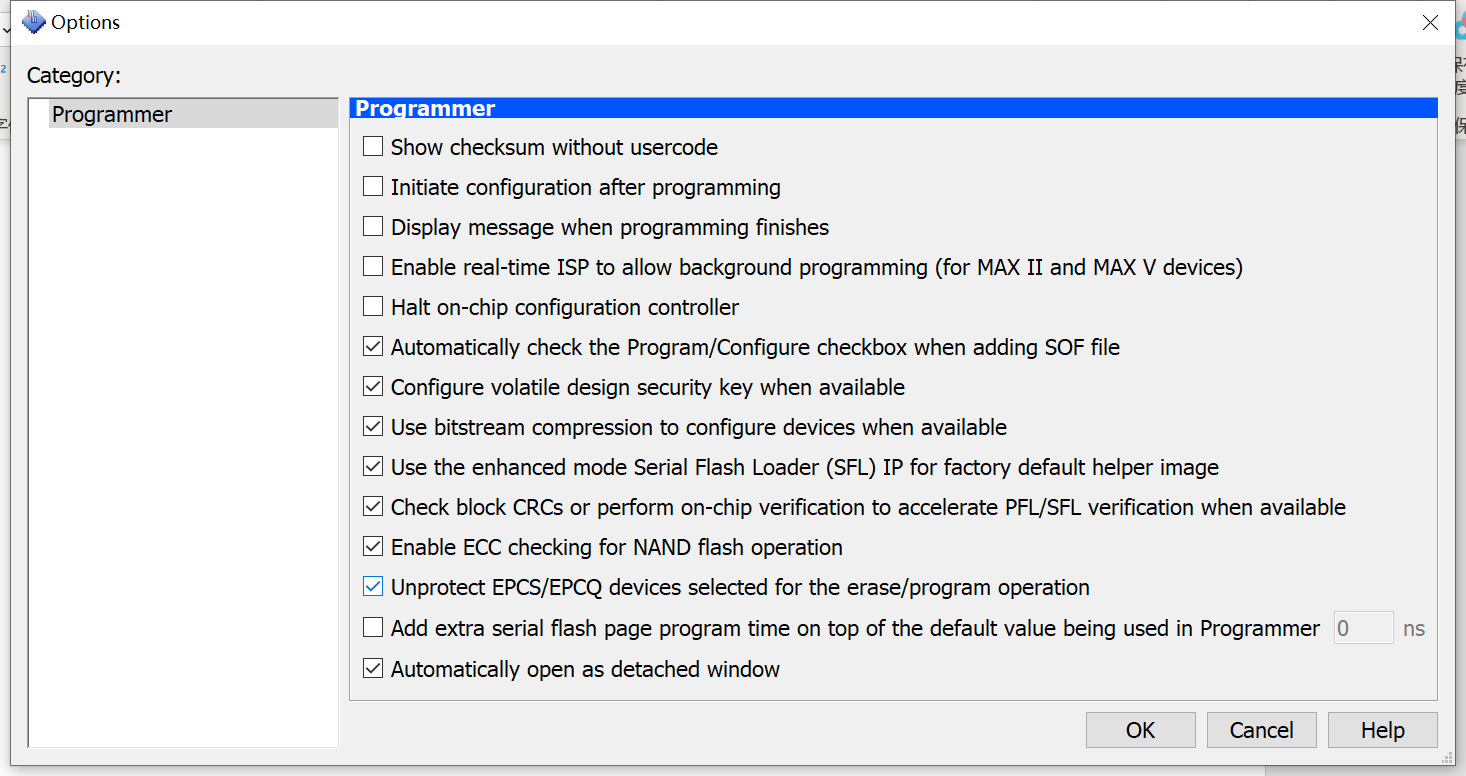
生成pof文件后，再烧写的时候，右上角经常会出现烧入failed情况。出现这种情况时，一般有两个原因：①usb数据线连接不牢问题，解决方法：插紧或换一根线；②“Active Serial Programming”写保护，解决办法如下所示。



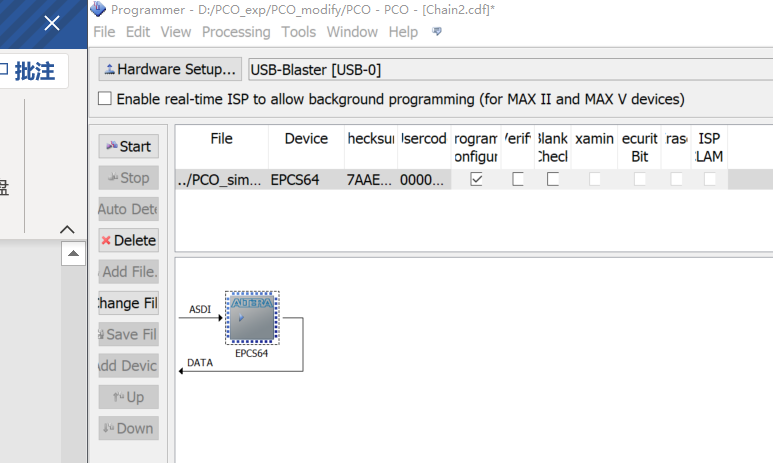
连接好板子，插上电源，打开开关后，在“programming”界面中将.pof文件添加好，勾选“Programme/Configure”后，选择上侧菜单栏“Tools”选项，点击“Options”。



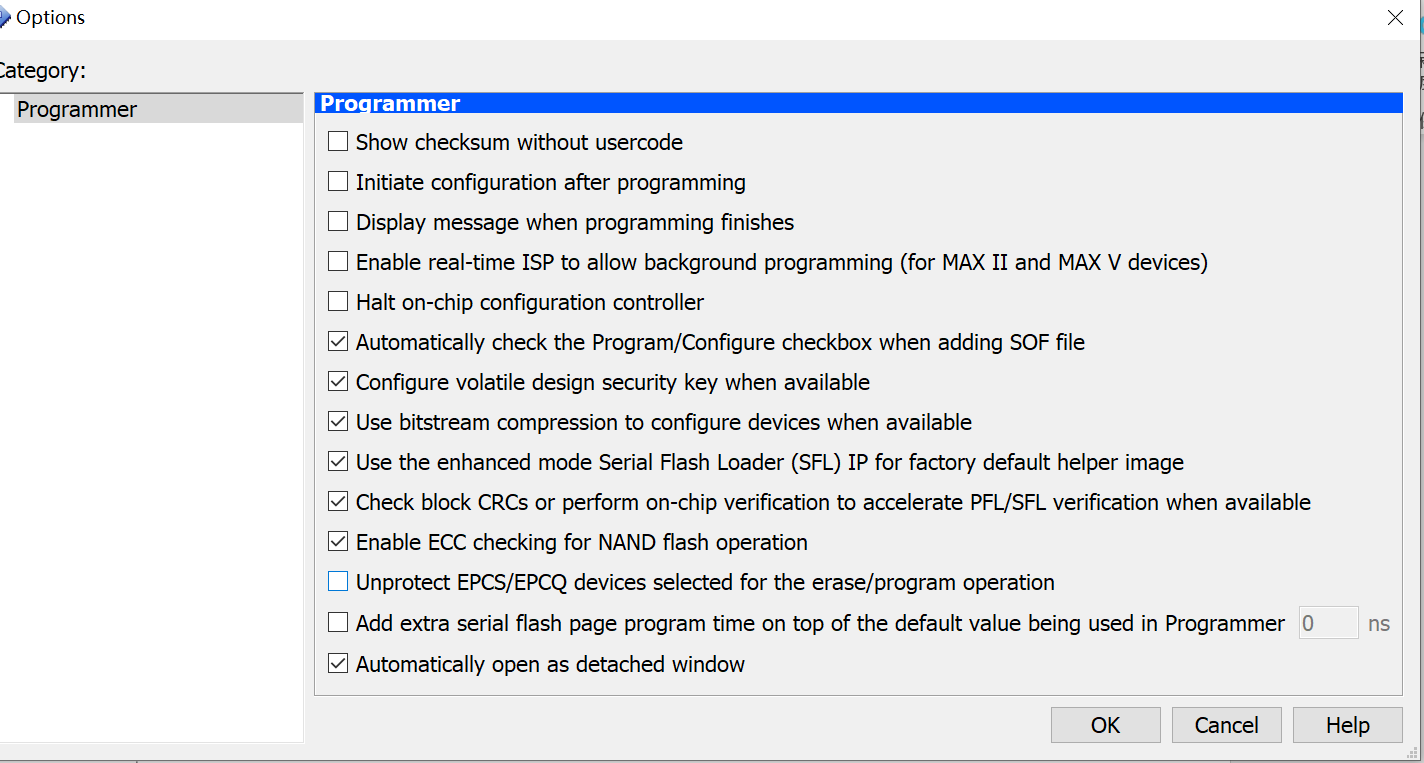
在“Options”界面，勾选倒数第三个选项“Unprotect EPCS/EPCQ devices selected for the erase/program operation”后，点击“OK”。



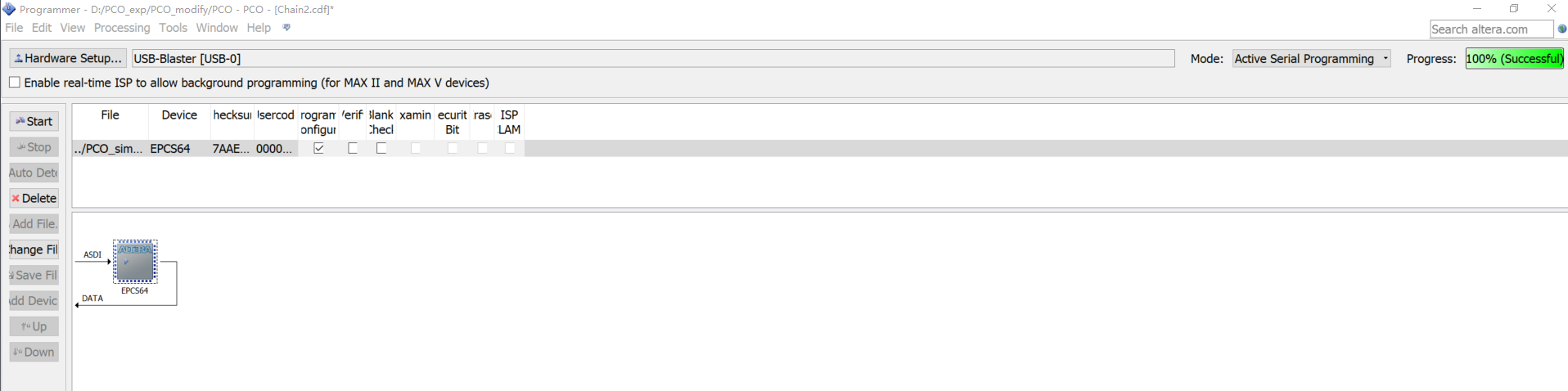
点击“Start”进行烧写，显示100%（successful）。



再次进入刚才的“Programmer”界面，将刚才勾选的内容取消勾选，点击“OK”。



再进行一次烧写，成功后即可将代码烧写到实验开发板中。



### 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。



