Tiny CPU设计报告

一、 项目概述

名称： Tiny CPU （微型处理器）

说明： 设计一个微型的 CPU，包含运算器/控制器/寄存器/总线接口逻辑。

使用软件： MaxPlusII （Altera 公司）

第九项目组成员： 肖剑 （040855）

饶驰 （040867）

丁於麟 （040874）

董涛 （040881）

二、 项目目标

通过学习 VHDL 硬件描述语言的一些基础知识，并且通过调用以及修改课外书籍上的程序实现 8 位微型简单处理器。使用 Altera 公司的 MaxPlusII 软件进行模拟仿真，在指令周期中，按照读取指令、分析指令和执行指令的顺序运行的。能够完成加法、减法和乘法的算术运算，也能够完成左移位和右移位的功能。通过本次项目来理解处理器的工作原理。

三、 项目设计思路

1. **简单处理器的组成**
   a) **程序计数器**——我们用一个 4 位计数器，计数范围是由 0~15。主要功能是记录下每个执行的指令地址，并把这个地址传送到 MAR 寄存器存放。
   b) **输入与 MAR（Memory Access Register）**——由两个部分组成，一是接受由“输入”部分输入到 RAM 内存的外部程序和数据；另一“MAR”是用来在 CPU 执行上述所加载的程序时，暂存下一个要执行的指令地址。
   c) **16×8 RAM**——这个内存大小共有 16 地址×8 位。
   d) **指令寄存器**——CPU 内的控制单元，将 RAM 的 8 位数据通过 Wbus 后读入指令寄存器，然后把数据一分为二，较高的 4 位输入指令部分，送至下一级的“控制器”，而较低的 4 位属于数据部分，将会被送至 Wbus。
   e) **控制器**（Controller）
   f) **累加器**（Accumulator）——8 位缓冲寄存器，存放目前计算机执行的实时数据。
   g) **运算器**（Calculater）——运算器会进行加、减、乘法的运算，运算结果放回到累加器中去。
   h) **B 寄存器**（B Register）
   i) **输出寄存器**（Output Register）——CPU 执行到“输出结果”指令时，便将“累加器”的结果传送到“输出寄存器”。
2. **简单处理器的寻址方式**
   本设计中，我们采用“类似直接寻址”的方式。由于本设计中使用的是 16*8 的 ROM，不是寻址范围很大的内存，所以采用此方式寻址，即指令中的地址直接就是操作数的地址，而且操作数就占用一个存储单元。
   指令分类：
   数据传送指令——寄存器之间传送指令、寄存器与存储器之间传送指令、堆栈操作指令和输入/输出指令。
   数据处理指令——算术运算指令、逻辑运算指令、移位与循环指令和比较与操作指令。
   程序控制指令——无条件转移指令、条件转移指令、子程序调用与返回指令、中断及中断返回指令。
   CPU 控制指令——标志或状态控制指令、特权指令、暂停指令和空操作指令。

类似直接寻址法	
指令	操作码
LDA	0000
ADD	0001
SUB	0010
MUL	0011
SHL	0100
SHR	0101

详细操作意义见“指令执行周期”

3. **简单处理器的指令周期**

a) 指令读取周期

状态 s0（寻址状态）

状态 s1（增加状态）

状态 s2（记忆状态）

b) 指令执行周期

LDA 指令

- 状态 s3：在这个状态下，LDA 后面的数据将传入 MAR，以便下个状态能取出该数值代表的地址里的内容值。
- 状态 s4：这个状态是将存放在“MAR”里的数据，通过 ROM 读出地址内的数据。
- 状态 s5：这个状态下没有动作。

ADD 指令

- 状态 s3：在这个状态下，上述的 AH 数据将传入 MAR，以便下个状态能取出该数值带边的地址里的内容值。
- 状态 s4：这个状态将存放在“MAR”里的 AH 数据，通过 ROM 读出 AH 地址内的数据。
- 状态 s5：这个状态是将存放在“累加器”和“B 寄存器”的数值内容”放如“运算器“相加后，再将相加结果放回”累加器“。

SUB 指令

- 状态 s3：在这个状态下，上述的 BH 数据将传入 MAR，以便下个状态能取出该数值代表的内容值。
- 状态 s4：这个状态将存放放在“MAR“里的 BH 数据，通过 ROM 读出 BH 地址内的数据。
- 状态 s5：这个状态是将存放在“累加器“和”B 寄存器“的数值内容放入”运算器“相减后，再将相减结果放会”累加器“。

MUL 指令

- 状态 s3：在这个状态下，上述的 BH 数据将传入 MAR，以便下个状态能取出该数值代表的内容值。
- 状态 s4：这个状态将存放放在“MAR“里的 BH 数据，通过 ROM 读出 BH 地址内的数据。
- 状态 s5：这个状态是将存放在“累加器“和”B 寄存器“的数值内容放入”运算器“相乘后，再将相乘结果放会”累加器“。

SHL 指令

- 状态 s3：在这个状态下，上述的 CH 数据将传入 MAR，以便下个状态能取出该数值代表的内容值。
- 状态 s4：这个状态将存放放在“MAR“里的 CH 数据，通过 ROM 读出 CH 地址内的数据。
- 状态 s5：在这个状态下，如果“B 寄存器“中的数据不为 0H，那么将”累加器“中的 数据左移一位，最右边的一位添 0，同时”B 寄存器“中的数据减 1。也就是说，每次指令周期，只能移位一次，要多次移位，那么就需要多个指令周期。

SHR 指令

- 状态 s3：在这个状态下，上述的 CH 数据将传入 MAR，以便下个状态能取出该数值代表的内容值。
- 状态 s4：这个状态将存放放在“MAR“里的 CH 数据，通过 ROM 读出 CH 地址内的数据。
- 状态 s5：在这个状态下，如果“B 寄存器“中的数据不为 0H，那么将”累加器“中的 数据右移一位，最左边的一位添 0，同时”B 寄存器“中的数据减 1。也就是说，每次指令周期，只能移位一次，要多次移位，那么就需要多个指令周期。

OUT 指令

- 状态 s3：这个状态下，“累加器“的内容将传送至”输出寄存器“，然后显示在二进制显示装置中。
- 状态 s4：这个状态下 OUT 指令没有动作。
- 状态 s5：这个状态下 OUT 指令没有动作。

HLT 指令

- 这个状态下，“控制器/序列发生器“将停止送出脉冲信号 CLK，这时 CPU 停止工作。
- 状态 s4：这个状态下 HLT 指令没有动作。
- 状态 s5：这个状态下 HLT 指令没有动作。

四、 程序代码说明

简单 CPU 的硬件描述语言代码

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity cpu is port (outport: out std_logic_vector (7 downto 0);——二进制显示输出端口
reset: in std_logic;
clk: in std_logic——时钟输入端口
);
end cpu;

architecture m of cpu is type state is (s0,s1,s2,s3,s4,s5);——指令周期的状态机类型
signal pstate:state;  ——定义状态机类型的信号 pstate
signal nstate:state;  ——定义状态机类型的信号 nstate
```

在状态机中使用两个状态机类型的信号，目的是使得指令执行周期的动作不放入 CASE_WHEN 中，而改用 IF_ELSE 语句将 S3 到 S5 独立出来，使 CASE 语句不致太过冗长。

```
signal pc:unsigned (3 downto 0);  ——程序计数器
signal mar:std_logic_vector (3 downto 0);  ——在 CPU 执行加载程序前地址暂存
signal acc:std_logic_vector (7 downto 0);  ——累加器
signal ir:std_logic_vector (7 downto 0);  ——指令寄存器
signal tmp:std_logic_vector (3 downto 0);  ——暂存器
signal breg:std_logic_vector (7 downto 0);  ——B  寄存器
signal outreg:std_logic_vector (7 downto 0);  ——输出寄存器
signal run:std_logic;  ——运行信号 run
signal cs:std_logic;——允许 ROM 读出信号
signal ramdata:std_logic_vector (7 downto 0);  ——数据存储器
signal addrbus:std_logic_vector (7 downto 0);  ——地址总线
signal databus:std_logic_vector (7 downto 0);  ——数据总线

component ram16_8 is  ——调用 16×8 ROM
 port (dataout:out std_logic_vector (7 downto 0);
address:in std_logic_vector (3 downto 0);
ce:in std_logic
);
end component;

begin
```

```
ctrstate:process (clk, reset) ——核心状态机进程
variable flag,f1:boolean; ——变量标志 flag 和 f1
variable keepacc:std_logic_vector (7 downto 0);

begin
if reset='1' then ——异步置位
 pc<="0000"; ——程序计数器置 0
 acc<="00000000"; ——累加器置 0
 breg<="00000000"; ——B 寄存器置 0
 run<='1'; ——运行信号置 1
 pstate<=s0; ——pstate 置状态 s0
 flag:=true; ——标志 flag 置 1
else 否则
if clk'event and clk='0' then ——时钟的下降触发沿
 if run='1' then ——如果运行信号为 1
 case pstate is
when s0 => ——当 pstate 处于状态 s0 的时候
nstate<=s1; ——nstate 被置 s1
f1:=true; ——标志 f1 置 true
if breg="00000000" then ——如果 B 寄存器值为 1
mar<=std_logic_vector (pc); ——mar 地址暂存为程序计数器
end if;

when s1=> ——当 pstate 处于 s1 时
nstate<=s2; ——nstate 被置 s2
if breg="00000000" then
if flag=true then ——如果 B 寄存器值为 0 且标志 flag 为 true
 pc<=pc+1; ——程序计数器加 1
 flag:=false; ——标志 flag 置 flase
end if;
end if;

when s2=> ——当 pstate 处于 s2 时
nstate<=s3; ——nstate 被置 s3
if breg="00000000" then
flag:=true; ——如果 B 寄存器值为 0 且标志 flag 为 true
ir<=databus; ——指令寄存器从数据总线读出指令
end if;

when s3 => ——当 pstate 处于 s3 时
nstate<=s4; ——nstate 被置 s4
if breg="00000000" then——如果 B 寄存器值为 0
tmp<=ir (7 downto 4); ——将指令寄存器中数的高 4 位即操作码放到暂存器
```

```
end if;

when s4 =>  ——当 pstate 处于 s4 时
nstate<=s5;  ——nstate 被置 s5

when s5=>  ——当 pstate 处于 s5 时
nstate<=s0;  ——nstate 被置 s0
 end case;
pstate<=nstate;    ——pstate 的值由 nstate 的值决定
 end if;

if pstate=s3 then  ——当 pstate 处于 s3 时
 if breg="00000000" then  ——如果 B  寄存器值为 0
c3:case tmp is  ——暂存器内存放的是指令中的操作码部分

when "0000" => 0000——代表 LDA 操作
mar<=ir(3 downto 0);  ——指令寄存器中的后 3 位即操作数存放到 mar 中去
when "0001" => 0001——代表 ADD 操作
mar<=ir(3 downto 0);
when "0010" => 0010——代表 SUB 操作
mar<=ir(3 downto 0);
when "0011" => 0011——代表 MUL 操作
mar<=ir(3 downto 0);
when "0100" => 0100——代表 SHL 操作
mar<=ir(3 downto 0);
when "0101" => 0101——代表 SHR 操作
mar<=ir(3 downto 0);
when "1110" => 1110——代表 OUT 操作
outreg<=acc;  ——累加器中的内容放到输出寄存器中

when "1111" => 1111——代表 HLT 操作
run<='0';  ——运行信号置 0

when others =>
null;
end case c3;
 end if;

elsif pstate=s4 then  ——当 pstate 处于 s4 时
 if breg="00000000" then  ——如果 B  寄存器值为 0
c4:case tmp is——暂存器内存放的是指令中的操作码部分

when "0000" =>    0000——代表 LDA 操作
acc<=databus;  ——将数据总线上的数据存放到累加器中去作为被操作数以及所要移位
```

```
的数
when "0001" => 0001——代表 ADD 操作
breg<=databus; ——将数据总线上的数据存放到 B 寄存器中去作为加数
when "0010" => 0010——代表 SUB 操作
breg<=databus; ——将数据总线上的数据存放到 B 寄存器中去作为减数
when "0011" =>  0011——代表 MUL 操作
breg<=databus; ——将数据总线上的数据存放到 B 寄存器中去作为乘数
when "0100" => 0100——代表 SHL 操作
breg<=databus; ——将数据总线上的数据存放到 B 寄存器中去作为移位的次数
when "0101" => 0101——代表 SHR 操作
breg<=databus; ——将数据总线上的数据存放到 B 寄存器中去作为移位的次数
when others =>
null;
 end case c4;
end if;

elsif pstate=s5 then——当 pstate 处于 s5 时
 if f1=true then ——如果标志 f1 为 true
 c5:case tmp is——暂存器内存放的是指令中的操作码部分

when "0001" =>0001——代表 ADD 操作
acc<=acc+breg; ——将累加器中的被加数和 B 寄存器中的加数相加，结果放回累加器
f1:=false;
breg<="00000000";

when "0010" => 0010——代表 SUB 操作
acc<=acc-breg; ——将累加器中的被减数和 B 寄存器中的减数相减，结果放回累加器
f1:=false;
breg<="00000000";

when "0011" => 0011——代表 MUL 操作
acc<=acc*breg; ——将累加器中的被乘数和 B 寄存器中的乘数相乘，结果放回累加器
f1:=false;
breg<="00000000";
when "0100" => 0100——代表 SHL 操作
if breg>"0000" then ——如果 B 寄存器大于 0 表示移位没有完成，继续移位
keepacc:=acc;
acc (7 downto 1) <=keepacc (6 downto 0);
acc(0)<='0'; ——完成一次左移操作，最右边一位置 0
breg<=breg-1; ——B 寄存器移位次数减 1
f1:=false; ——标志 f1 置 false
else null;
end if;
```

```
when "0101" => 0101——代表 SHR 操作
if breg>"0000" then
keepacc:=acc;
acc (6 downto 0) <=keepacc (7 downto 1);
acc (7)<='0'; ——完成一次右移操作，最左边一位置 0
breg<=breg-1; ——B 寄存器移位次数减 1
f1:=false; ——标志 f1 置 false
else null;
end if;

when others =>
null;
end case c5;
  end if;
end if;
end if;
end if;
end process ctrstate;

sysconnect:block ——系统连接块
begin

u1:ram16_8 port map (ramdata,addrbus,cs); ——ROM 的连接，输入是地址总线和允许输出信号，输出是 ramdate

ad_bus: ——地址总线的连接
with pstate select
addrbus<=mar when s2|s4, ——当 pstate 处于状态 s2 或 s4 的时候，地址总线有具体值，为 mar 中的内容
"0000" when others; ——否则地址总线输出 0

da_bus: ——数据总线连接
databus<=ramdata when cs='0' else ——当允许 ROM 输出信号为 0 时，数据总线上才是 ramdate 的值
"00000000"; ——否则数据总线上没有值，为 0

chip_enab: ——控制 ROM 的读出
with pstate select
cs<='0' when s2|s4, ——只有当 pstate 处于状态 s2 或 s4 的时候，允许 ROM 读出信号置 0，即允许 ROM 读出
'1' when others; ——其他时候不能读出
outport<=outreg;
end block sysconnect;
end m;
```

ROM 解释
（所有运算所用的都是 16 进制代码）

1．10+15+17-20=1C

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ram16_8 is
      port (dataout:out std_logic_vector (7 downto 0);
address:in std_logic_vector(3 downto 0);
ce:in std_logic
);
end ram16_8;

architecture r of ram16_8 is
begin
dataout<="00001001" when address="0000" and ce='0' else
"00011010" when address="0001" and ce='0' else
"00011011" when address="0010" and ce='0' else
"00101100" when address="0011" and ce='0' else
"11100000" when address="0100" and ce='0' else
"11110000" when address="0101" and ce='0' else
"00010000" when address="1001" and ce='0' else
"00010101" when address="1010" and ce='0' else
"00010111" when address="1011" and ce='0' else
"00100000" when address="1100" and ce='0' else
"00000000";
end r;
```

2．3×4=C

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ram16_8 is
      port (dataout:out std_logic_vector (7 downto 0);
address:in std_logic_vector(3 downto 0);
ce:in std_logic
```

```
);
end ram16_8;

architecture r of ram16_8 is
begin
dataout<="00001001" when address="0000" and ce='0' else
"00111010" when address="0001" and ce='0' else
"11100000" when address="0010" and ce='0' else
"11110000" when address="0011" and ce='0' else
"0000011" when address="1001" and ce='0' else
"0000100" when address="1010" and ce='0' else
"00000000";
end r;
```

3．连续移动

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ram16_8 is
      port (dataout:out std_logic_vector (7 downto 0);
address:in std_logic_vector(3 downto 0);
ce:in std_logic
);
end ram16_8;

architecture r of ram16_8 is
begin
dataout<="00001001" when address="0000" and ce='0' else
"00001001" when address="0000" and ce='0' else
"01001010" when address="0001" and ce='0' else
"01011011" when address="0010" and ce='0' else
"11100000" when address="0011" and ce='0' else
"11110000" when address="0100" and ce='0' else
"0001111" when address="1001" and ce='0' else
"00000100" when address="1010" and ce='0' else
"00000010" when address="1011" and ce='0' else
"00000000";
end r;
```

指令	操作码	范例	说明
OUT	1110	OUT	将累加器的内容输入到"输出寄存器"
HLT	1111	HLT	结束 CPU 执行
LDA	0000	LDA 9H	将 9H 的内存的内容加载到累加器中
ADD	0001	ADD AH	将 AH 内存内容值和累加器内容值相加，并将运算结果放回到累加器中去
SUB	0010	SUB BH	将 BH 内存内容值和累加器内容值相加，并将运算结果放回到累加器中去
MUL	0011	MUL CH	将 CH 内存内容值和累加器内容值相加，并将运算结果放回到累加器中去
SHL	0100	SHL BH	将累加器中的内容依次向左位移位，最右边的位添零，移位的次数和 BH 内存内容值相等
SHR	0101	SHR CH	将累加器中的内容依次向右位移位，最左边的位添零，移位的次数和 CH 内存内容值相等

五、 程序调试与模拟仿真

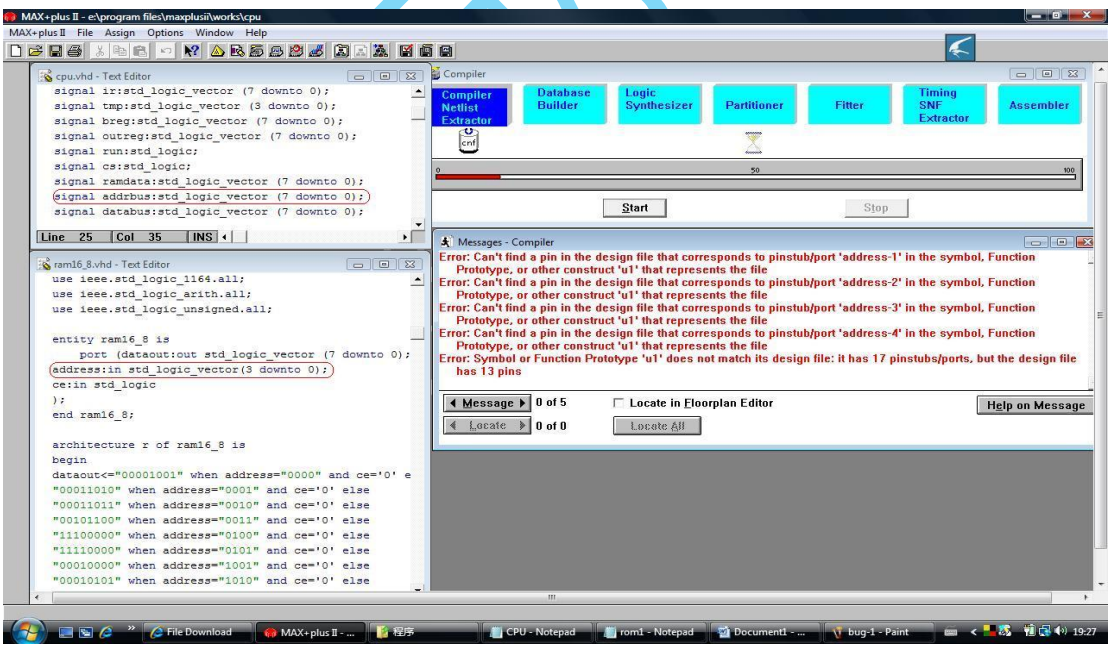
我们使用 Altera Max+plus II 软件来对设计的 CPU 进行仿真。建立两个*.vhd 文件，分别命名为 cpu.vhd 和 ram16_8.vhd，其中 cpu.vhd 为 CPU 的主体文件，ram16_8.vhd 为其内存文件。

然后将第四部分程序代码说明中的：

分别输入到两个文件中。

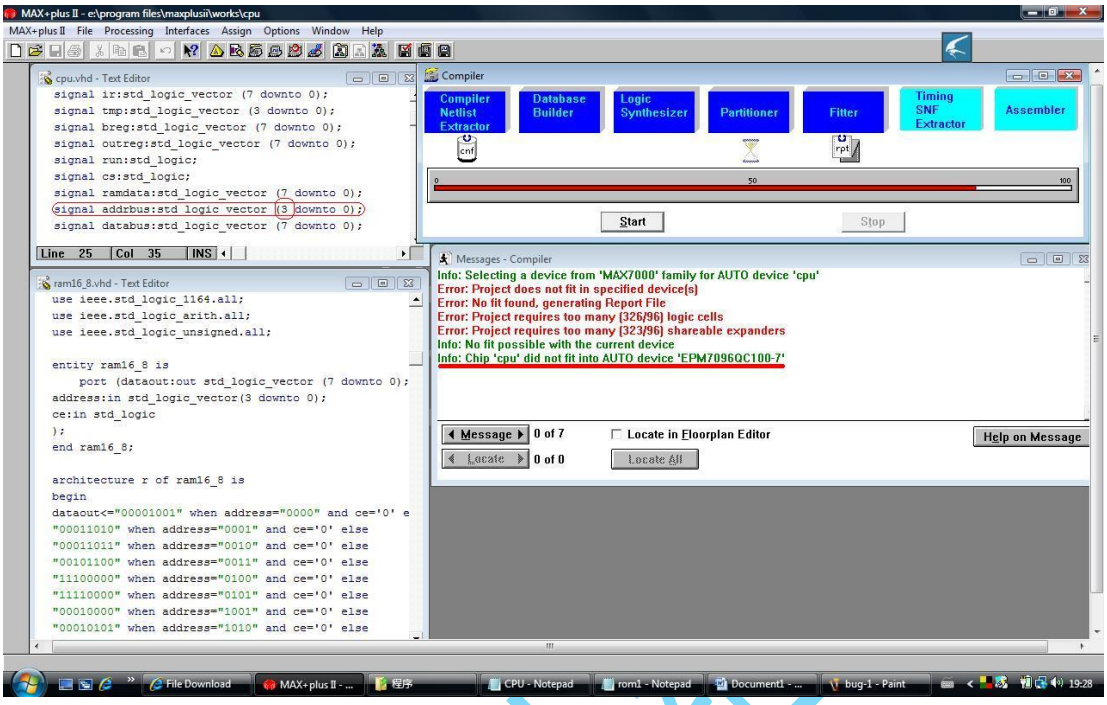
现在我们开始 Debug 阶段。

1. 当程序输入完成后我们点击 Compiler。Max+plus II 给了我们如下的信息：



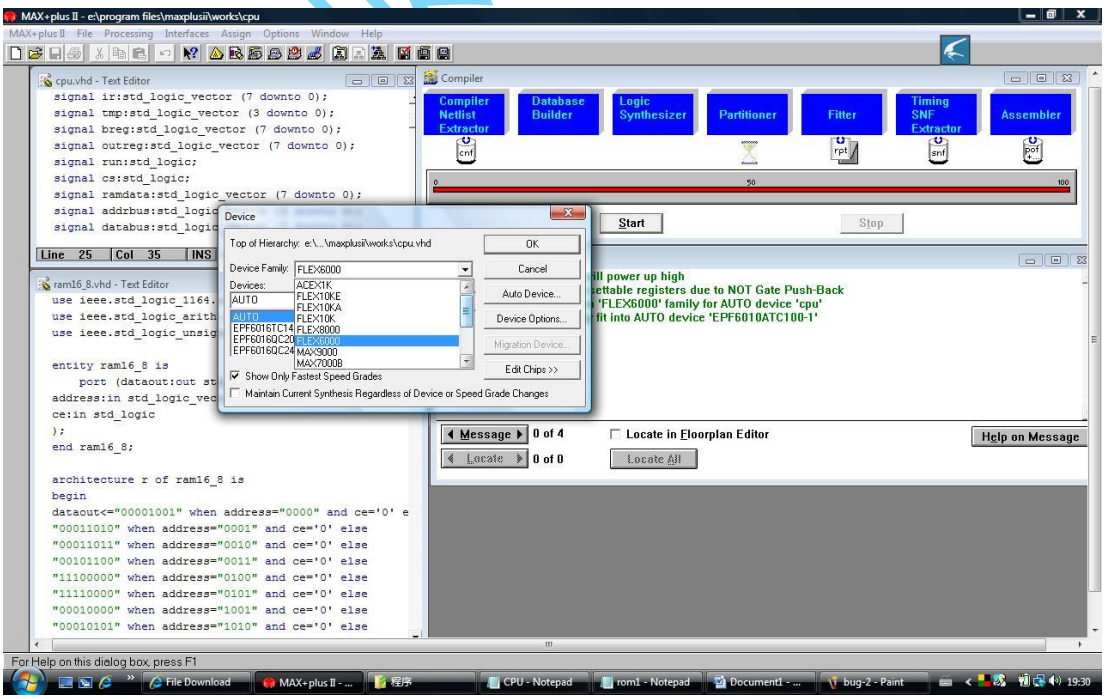
很明显，ram 中的地址总线的个数与 CPU 中的地址总线数不同，有 4 个针脚未定义。所以我们修改 CPU 的地址总线个数，使其和内存中相同。

2.　再次运行 Compiler，出现新的错误。如图。



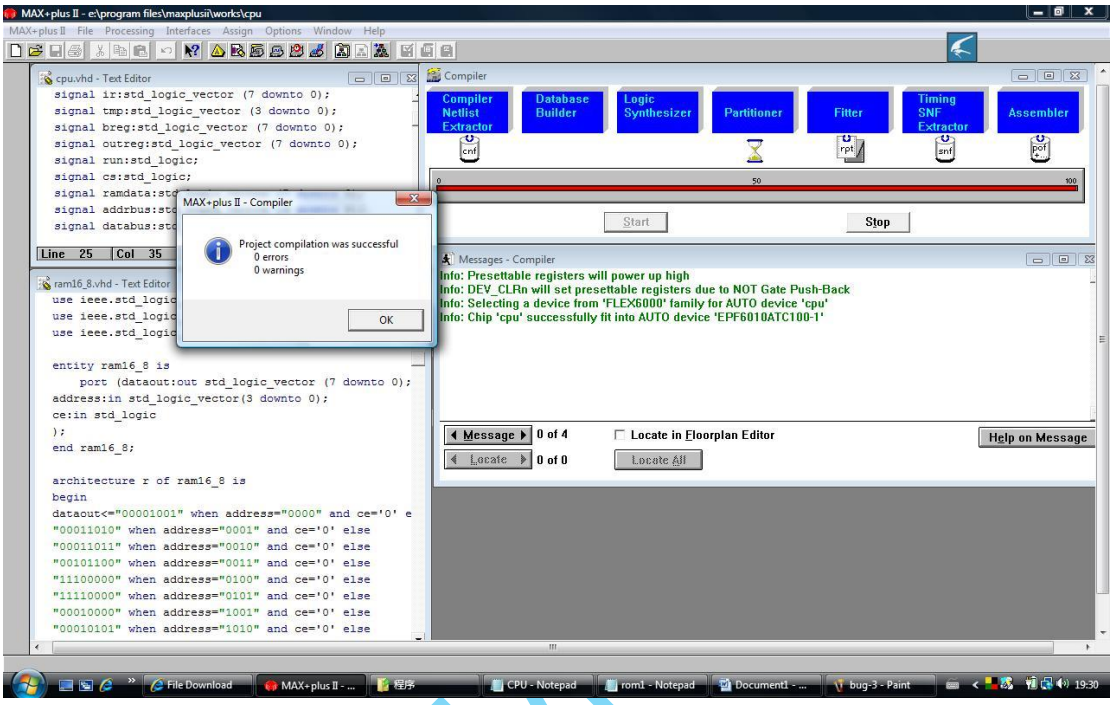
Max+plus II 提示当前选择的可编程逻辑器件无法支持此 CPU 所有的针脚定义。

3. .点击 Assign→Device…选择程序提供的可编程逻辑器件。我们的选择如图：



选择 FLEX6000 芯片，此芯片支持此 CPU 所需的功能。

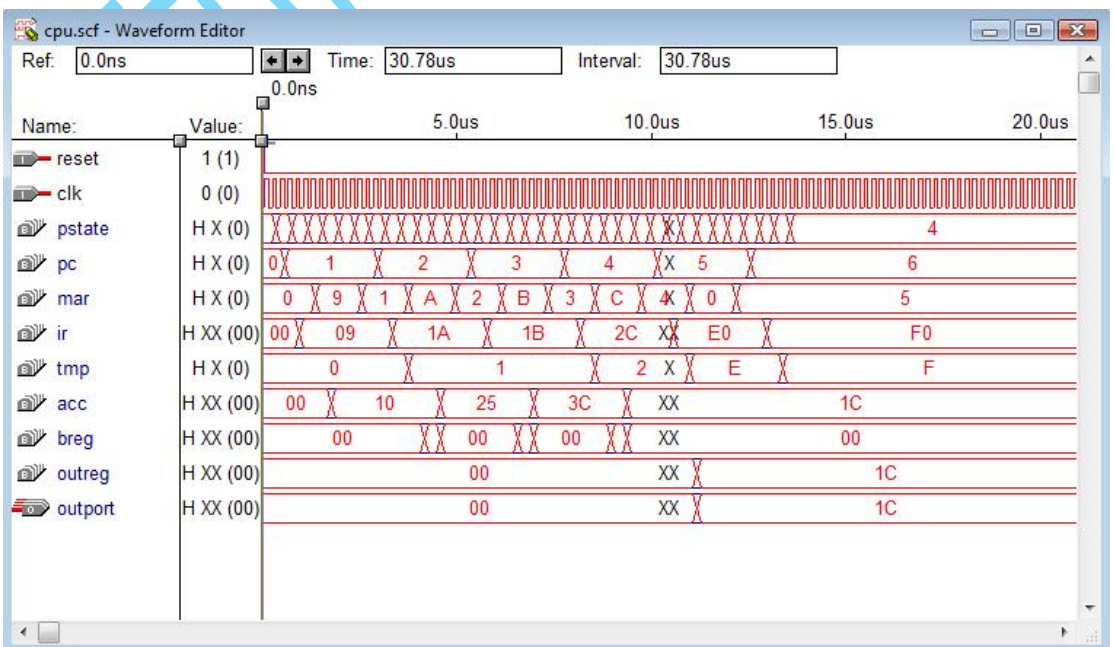
4. 再次运行 Compiler，结果如图：



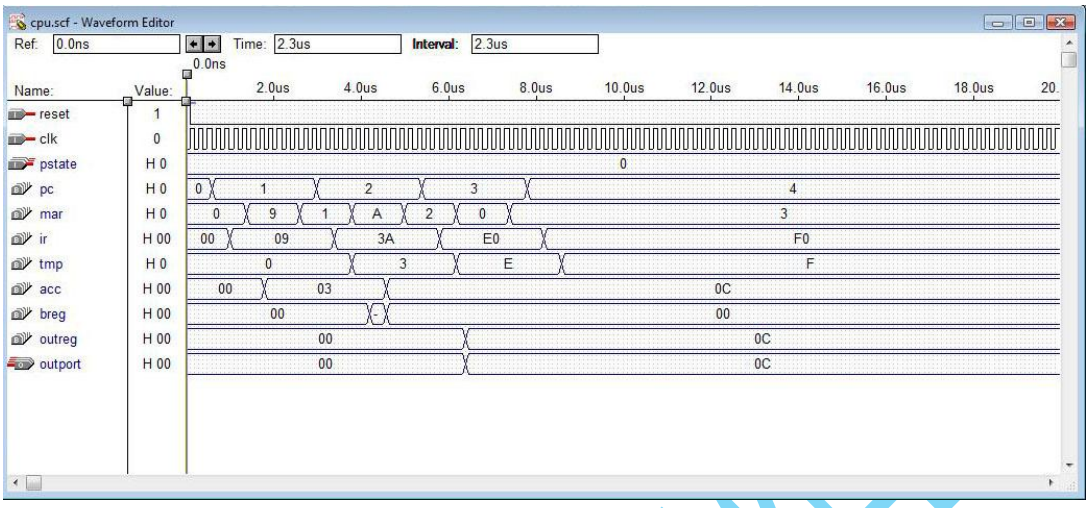
Bingo!

在 CPU 正常运行时，我们分析了它的输出信号波形。

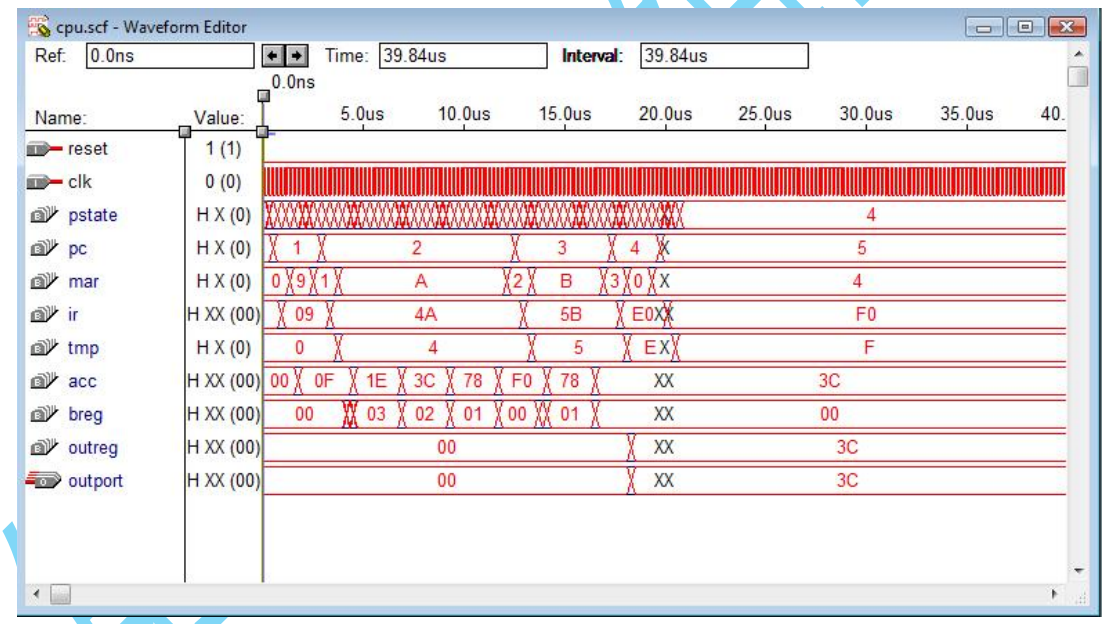
1. 加减法“10+15+17-20=1C”运算波形图。



2. 乘法“3×4=C”波形图。



3. 连续移动波形图。



本次模拟仿真实验对三个 ROM 进行了波形仿真分别是：

- **10+15+17-20=1C**
- **3×4=C**
- **连续移动（将 0F 左移 4 位再右移 2 位）**

实测波形与理论波形相一致。

模拟仿真测试成功!!!

六、 项目总结

1. 简单 CPU 的不足之处
   - 没有直接实现除法
   - 算术运算没有优先级
   - 移位动作的效率较低
   - 没有 RAM

2. 参考书目
   - 甘历——《VHDL 应用与开发实践》——科学出版社 ,2003 年
   - 卢毅，赖杰——《VHDL 与数字电路设计》——科学出版社，2001 年
   - FLEX 10K Embedded Programmable Logic Device Family Data Sheet, March 2001, Version4.1

3. 项目成员任务分配

丁於麟：课题选定、资料搜集与整理、代码调试、程序录入（部分）、软件搜集与破解、操作演示、屏幕录像、代码说明（部分）、演示视频制作、项目报告撰写（部分）。

饶驰： 程序调试及其仿真、项目报告调试部分撰写、软件搜集（部分）。

董涛： 代码说明、视频讲解。

肖剑： 程序录入、指令解释。