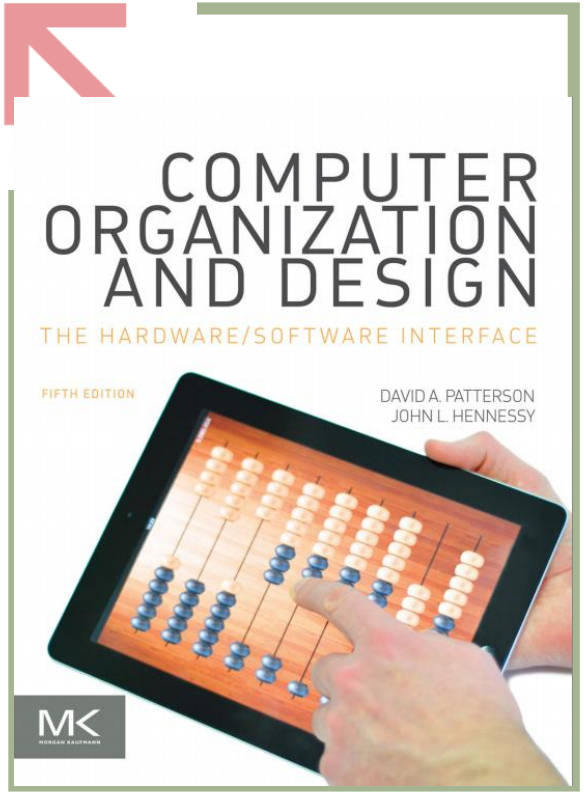
单周期CPU

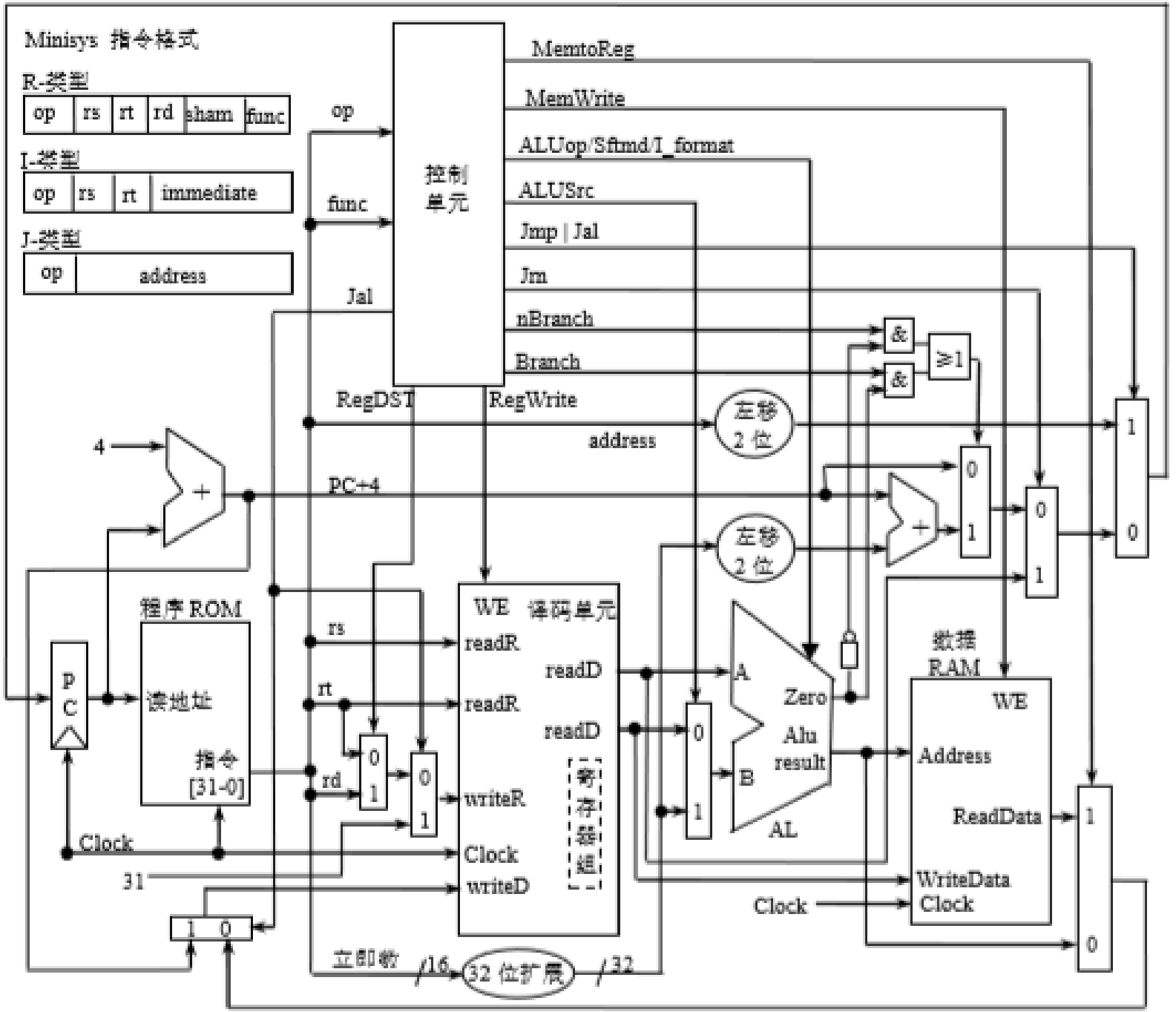

COMPUTER
ORGANIZATION
AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
FIFTH EDITION
DAVID A. PATTERSON
JOHN L. HENNESSY
MK

单周期CPU

1. 创建时钟

2. 创建CPU顶层模块
1）对已完成的时钟单元，译码单元，执行单元，取址单元，控制单元以及存储单元进行实例化

2） 完成CPU内部的模块间连线以及与CPU端口的绑定

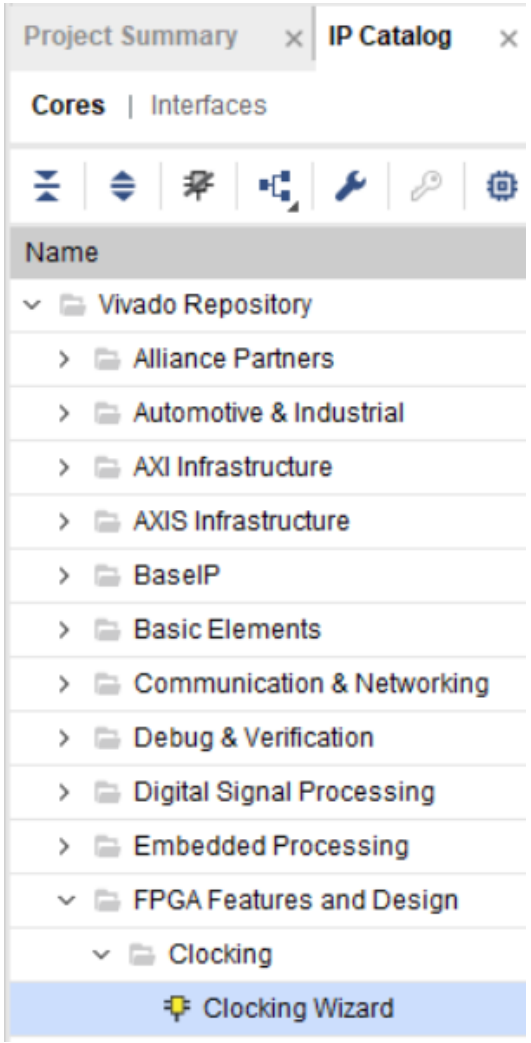


时钟

1.添加PPL时钟IP核，进行分频

2.Minisys开发板上时钟为100Mhz

3.PPL分频后，产生23Mhz左右的时钟



功能验证
1）创建verilog设计模块，对该IP核进行实例及端口绑定
2）搭建testbench，验证在输入信号为100Mhz的时钟时，输出信号是否为23Mhz的时钟信号

时钟

Component Name cpuclk

Clocking Options | Output Clocks | Port Renaming | PLLE2 Settings | Summary

Clock Monitor

☐ Enable Clock Monitoring

Primitive

○ MMCM ◉ PLL

Clocking Features

☑ Frequency Synthesis ☐ Minimize Power

☑ Phase Alignment

☐ Dynamic Reconfig

☐ Safe Clock Startup

Jitter Optimization

◉ Balanced

○ Minimize Output Jitter

○ Maximize Input Jitter filtering

Clocking Options | Output Clocks | Port Renaming | PLLE2 Settings

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)	
		Requested	Actual
☑ clk_out1	clk_out1	23.000	23.000

Component Name cpuclk

Clocking Options | Output Clocks | Port Renaming | PLLE2 Settings | Summary

Enable Optional Inputs / Outputs for MMCM/PLL

☐ reset ☐ power_down

☐ locked

Reset Type

◉ Active High ○ Active Low

CPU顶层模块设计

- 确定CPU对外的输入输出端口
  - 时钟信号、复位信号
  - 输入端口
  - 输出端口
- 确定CPU内部的基本模块
  - 时钟模块
  - 取址、控制、译码、存储及IO处理、执行
- 确定CPU内部基本模块之间以及基本模块与顶层模块的关系
- 使用 verilog的structure描述 或者 vivado中的block design 完成cpu顶层模块设计
  - verilog 语法中大小写敏感

单周期CPU添加程序代码及初始化寄存器

1）**直接使用coe文件**

将prgmip32.coe（初始化rom） 和 dmem32.coe（初始化ram） 放入以下目录，并指定为对应IP的初始化文件

a) 工程文件夹\工程名.srcs\sources_1\ip\prgrom （初始化prgrom）

b) 工程文件夹\工程名.srcs\sources_1\ip\RAM （初始化RAM）

2）**编写汇编程序，通过汇编器生成coe文件，再使用其作为对应IP的初始化文件**

a) 基于minisys的汇编代码：符合minisys语法规范(可参考: lab14_汇编器\Minisys1Av2.2汇编器\cputest.asm)

b) 汇编器： lab14_汇编器\Minisys1Av2.2汇编器\ MinisysAv2.0.exe，生成的coe文件在同级的子目录output下

提醒：coe文件被更新后要重新产生项目中的 prgrom 和 RAM IP 封包，功能需要重新验证
具体指令及minisys汇编器的使用请参考后页tips

CPU顶层模块的仿真(1)

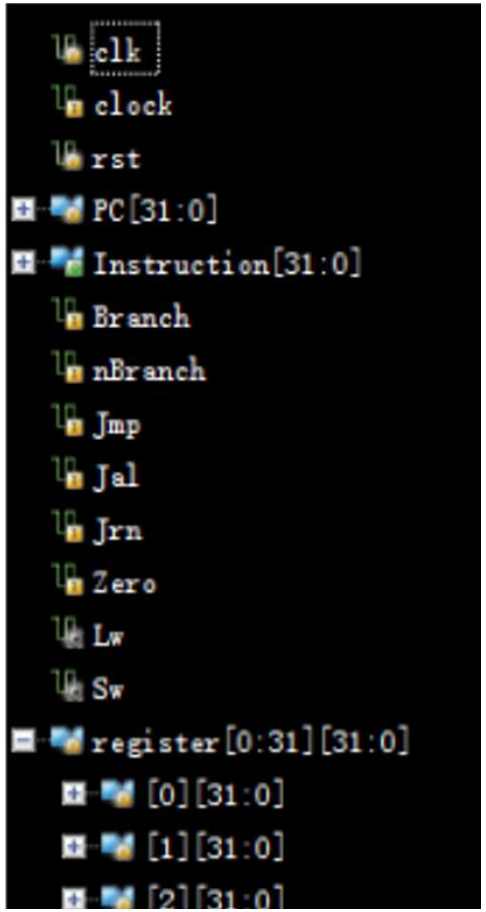
仿真文件参考：

```
`timescale 1ns / 1ps
module minisys_sim(  );
    // input
    reg clk = 0;
    reg rst = 1;
    reg switch2N4 = 8'b10101100;

    //  output
    wire led2N4;

    minisys u (.clk(clk),.rst(rst),.led2N4(led2N4),.switch2N4(switch2N4));
    initial begin
        #7000 rst = 0;
    end
    always #10 clk=~clk;
   Endmodule
```

待观测的波形参考：



CPU顶层模块的仿真(2)

7000ns 复位信号后撤销，程序中的指令行为如下:

(指令行为由下发到rom中的coe文件决定，对应的指令请参考: ’ lab14_汇编器\Minisys1Av2.2汇编器\ cputest.asm’)

大约时间（ns）	执 行 指 令	结果现象
7000~9710	一组 ori 将$i 寄存器赋值为 i	$1=1;$2=2;……$31=31
9800	ori $v0,$zero,0x55	$2=55H
9970~10060	lw $v1,buf($a0)	LW=1； $3=AAH
10146	sw $at,8($zero)	SW=1
10150	add $at,$v0,$v1	$1=FFH (55H+AAH)
10320	subu $a0,$v1,$v0	$4=55H (AAH-55H)
10400	slt $a0,$v0,$at	$4=1 (55H<AAH)
10500	and $at,$v1,$a3	$1=2 (AAH&7H)
10580	or $a2,$v0,$at	$a6=57h (55H|02H)
10670	xor $a3,$v0,$v1	$7=FFH (55H^AAH)
10680~10760	nor $a2,$a1,$at	$6=FFFFFFF8H (~(5|2))
10760	lop: beq $v1,$v0,lop	Branch=1;Zero=0 跳转不成立
10850~13624	lop1: sub $v0,$v0,$a1 bne $a1,$v0,lop1	$2 由 55A 每次减去 5，不断循环直到$2=5

CPU顶层模块的仿真(3)

13624	beq $at,$at,lop2	Branch=1;Zero=1;PC 从 B8H 跳到了 C0H,跳过了 BCH 地址的 NOP 指令
13710~13800	lop2: jal subp	Jal=1;$31=31H (C4H>>2); 从 C0H 跳转到 C8H，跳过 C4H
13840	subp: jr $ra	Jrn=1 跳回到 C4H 地址
13920	j next	Jmp=1；跳到地址为 CCH 的地方（next）
14050	next: addi $v0,$zero,0x99	$2=99H
14150	ori $v1,$zero,0x77	$3=77H
14240	sll $v1,$v0,4	$3=990H (99H<<4)
14330	srl $v1,$v0,4	$3=09H (99H>>4)
14420	srlv $v1,$v0,$at	$3=26H (99H>>2)
14500	lui $a2,0x9988	$6=99880000H
14580	sra $a3,$a2,4	$7=F9988000H (99880000H>>>4)
14670	addi $v0,$zero,0	$2=0
14760	addi $v1,$zero,2	$3=2
14850	sub $at,$v0,$v1	$1=FFFFFFFE0H (0-2)
14890	j start	Jmp=1；跳回到 0 地址重复执行

单周期CPU下板测试

1. 下发测试场景对应的coe文件，给程序CPU下发程序及初始化寄存器。有以下两种情况：
    1. coe文件已准备好
        1. 将coe文件拷贝到sources_1\IP\IP名 目录下，重新打包
    2. 根据测试场景编写汇编程序，使用’Minisys1Av2.2汇编器’完成对该汇编程序的处理。在编译通过的前提下本页1-1-1中的coe文件拷贝至目的地，重新打包的过程

2. 根据单周期CPU对外接口及minisys开发板的特性实现约束文件

3. 对该项目进行综合、实现、生成比特流文件

4. 连接好 Minisys实验板，进行下载验证，判断测试结果是否符合预期

综合检查

- 要求：
  - 1.实现单周期CPU
    - 熟悉对核心模块的代码实现
    - 熟悉重要控制信号、数据与各模块之间的关系
  - 2.根据测试场景进行仿真
    - 时钟信号是否正确
    - 单周期CPU的行为符合预期（拨码开关的行为在testbench中进行模拟）
  - 3. 根据测试场景进行下板测试
    - 单周期CPU的行为符合预期
  - 4. 测试场景
    - 1）基本场景（使用提供的coe文件进行测试，课件p12）
    - 2）扩展场景（加分项，完成从汇编程序编写、汇编处理、coe文件到测试的全过程；课件p13）

综合检查1（测试场景）

请使用 sakai站点资源目录下的 ‘**lab14_汇编器\Minisys1Av2.2汇编器\testcase1’下的prgmip32.coe及 dmem32.coe** 分别实现程序下发和寄存器初始化，在minisys开发板上测试单周期CPU的行为是否符合下表描述的行为

（ SW23/SW22/SW21 为功能选择；VAL位宽16， GLD7~0~YLD7~0 始终输出 VAL 的值)

SW23	SW22	SW21	动作
X	0	0	无
0	0	1	将SW15~SW0这16位作为输入赋值给VAL
0	1	0	VAL=VAL+1 （每隔约1秒动作一次）
0	1	1	VAL=VAL-1（每隔约1秒动作一次）
1	0	1	VAL左移1位（每隔约1秒动作一次）
1	1	0	VAL算术右移1位（每隔约1秒动作一次）
1	1	1	VAL逻辑右移1位（每隔约1秒动作一次）

综合检查2（测试场景）

请**自行编写汇编源代码**，实现下表行为；通过**Minisys1Av2.2**汇编器汇编处理后，使用**output**目录下生成的prgmip32.coe 及 dmem32.coe 分别实现程序下发和寄存器初始化，在minisys开发板上测试单周期CPU的行为是否符合下表描述的行为
（ SW23/SW22/SW21 为功能选择；VAL位宽16， GLD7~0~YLD7~0 始终输出 VAL 的值)

SW23	SW22	SW21	动作
X	0	0	编号连续的16盏led灯中奇数编号灯与偶数编号灯交替闪烁（每隔约1秒动作一次）
0	0	1	将SW15~SW0这16位作为输入赋值给VAL
0	1	0	VAL=VAL+1 （每隔约1秒动作一次）
0	1	1	VAL=VAL-1（每隔约1秒动作一次）
1	0	1	VAL左移1位（每隔约1秒动作一次）
1	1	0	VAL算术右移1位（每隔约1秒动作一次）
1	1	1	VAL逻辑右移1位（每隔约1秒动作一次）

Tips1：CPU顶层文件的设计参考

在元件例化的时候特别注意字母大小写一定要一致。

```
cpuclk cpuclk(
    .clk_in1(clk),     //100MHz
    .clk_out1(clock)     //cpuclock
);

Ifetc32 ifetch(
    .Instruction(instruction),
    .PC_plus_4_out(pc_plus_4),
    ? ? ?
);

Idecode32 idecode(
    .read_data_1(read_data_1),
    .read_data_2(read_data_2),
    ? ? ?
);

control32 control(
    .Opcode(instruction[31:26]),
    .Function_opcode(instruction[5:0]),
    ? ? ?
);

Executs32 execute(
    .Read_data_1(read_data_1),
    .Read_data_2(read_data_2),
    ? ? ?
 );

dmemory32 memory(
    .read_data(read_data),
    .address(address),
    .write_data(write_data),
    .Memwrite(memwrite),
    .clock(clock)
);
```

Tips1：CPU顶层文件的设计参考

```
memorio memio(
    .caddress(alu_result),
    .address(address),
    .memread(memread),
    .memwrite(memwrite),
    .ioread(ioread),
    .iowrite(iowrite),
    .mread_data(read_data),
    .ioread_data(ioread_data),
    .wdata(read_data_2),
    .rdata(rdata),
    .write_data(write_data),
    .LEDCtrl(ledctrl),
    .SwitchCtrl(switchctrl)
);
```

```
ioread multiioread(
    .reset(rst),
    .ior(ioread),
    .switchctrl(switchctrl),
    .ioread_data(ioread_data),
    .ioread_data_switch(ioread_data_switch)
);
```

```
leds led24(
    .led_clk(clock),
    .ledrst(rst),
    ? ? ?
);

switchs switch24(
    .switclk(clock),
    .switrst(rst),
    ? ? ?
);
```

Tips2: 基于MIPS的minisys（1）

- 数据段定义伪指令

  .data　　[addr]

  定义数据段，程序的变量需要在该伪指令下定义。汇编程序应分配和初始化变量的存储空间。如果定义了addr，则该数据段从这个addr地址开始。如果未定义，默认从0地址开始。

- 代码段定义伪指令

  .text [addr]

  定义代码段，如果定义了addr，则该代码段从这个addr地址开始。如果未定义，默认从0地址开始。

Tips2: 基于MIPS的minisys（2）

- Minisys的语法基于MIPS32做了适当裁剪
- 支持
  - lw，sw （访问memory 或者对IO设备的读写访问）
  - 基本算术、逻辑、移位运算
  - slt指令
  - 跳转指令：j，jr， jal，beq，bne
- 不支持
  - move
  - 基于byte，halfword的相关操作

- 语法约束
  - #”到行末的是注释部分，注意：“#”不要出现在一行的最左列，因为最左列的“#”表示要启用C预编译器

Tips2: 基于MIPS的minisys（3）

- 读拨码开关的输入
  - 第一步，$28中放端口地址的高20位
    - `lui  $28,0xFFFF`
    - `ori  $28,$28,0XF000`
  - 第二步，读拨码开关端口
    - `lw $s1,0XC70H($28)`
    - `lw $s1,0XC72H($28)`

接口部件	首地址
拨码开关低16位	0xFFFFFC70
拨码开关高8位	0xFFFFFC72

Tips2: 基于MIPS的minisys（4）

- 对LED输出
  - 第一步，$28中放端口地址的高20位
    - `lui  $28,0xFFFF`
    - `ori  $28,$28,0XF000`
  - 第二步，读拨码开关端口
    - `sw $s1,0XC60H($28)`
    - `sw $s1,0XC60H($28)`

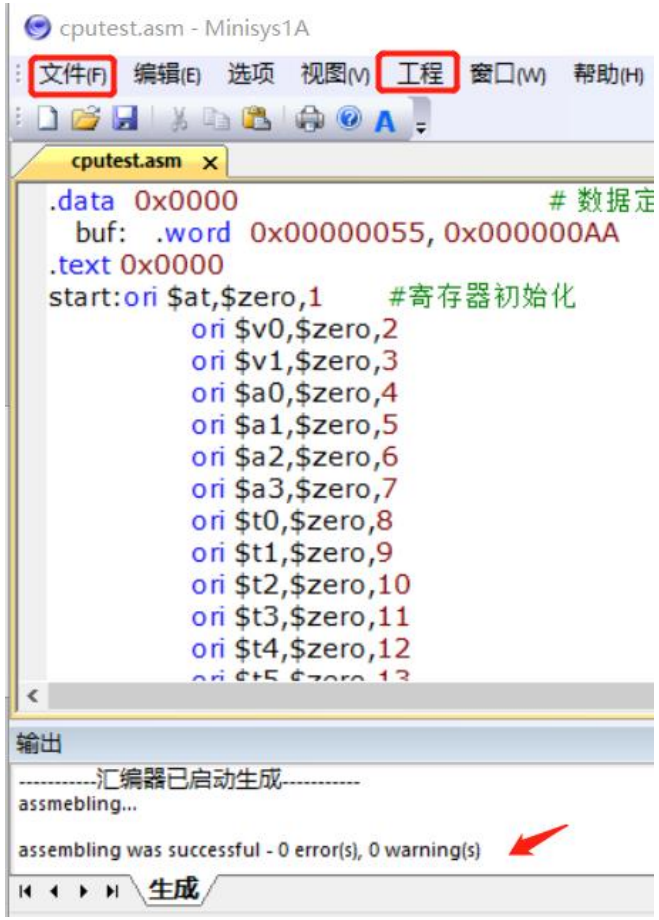
接口部件	首地址
GLD-YLD	**0xFFFFFC60**
RLD	**0xFFFFFC62**

Tips3: Minisys 汇编器的使用（1）

功能：
编辑汇编源代码文件
汇编处理生成coe文件
（coe文件用于vivado 对rom/ram ip核进行初始化）

运行:
双击 MinisysAv2.0 图标打开汇编器
（sakai站点的lab14目录下)

重要菜单：‘文件’，‘工程’



Tips3: Minisys 汇编器的使用（2）

- 创建汇编程序
  - 菜单栏‘文件’->‘新建’asm文件
- 打开已有的汇编源代码文件
  - 菜单栏‘文件’->‘打开’已有asm文件
- 汇编处理
  - 菜单栏‘工程’-> 64KB 设置rom和ram的大小
  - 菜单栏‘工程’->‘汇编’
- 输出文件请参考 output目录