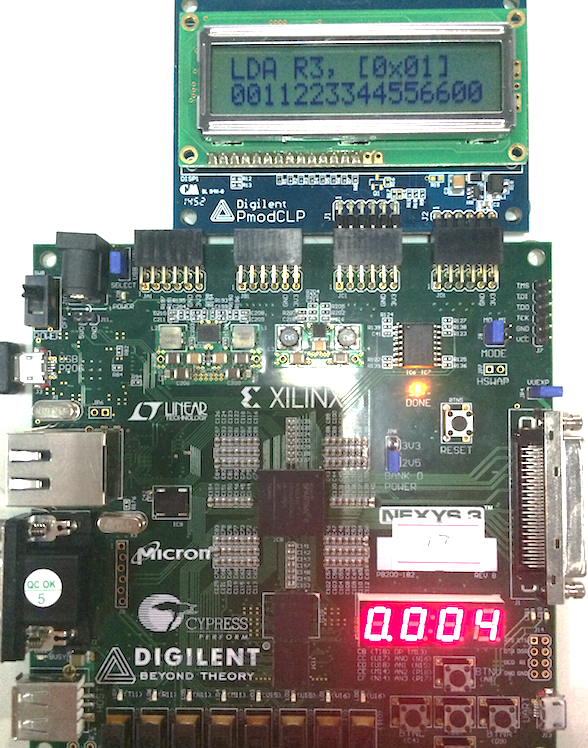
**基于Digilent® Nexys 3™开发板的**

**RISC指令集CPU设计**



**作者：杨志飞**

**学号：1130310217**

**2013级计算机科学试验班（1336101）**

**计算机科学与技术学院**

**哈尔滨工业大学**

目录

[前言 3](#_Toc426736281)

[CPU结构介绍 3](#_Toc426736282)

[1. 时钟管理模块CLK 3](#_Toc426736283)

[2. 取指管理模块IF 3](#_Toc426736284)

[3. 运算管理模块EX 4](#_Toc426736285)

[4. 存储管理模块MEM 5](#_Toc426736286)

[5. 回写管理模块WB 5](#_Toc426736287)

[6. 访存控制模块AC 6](#_Toc426736288)

[7. 主模块CPU 6](#_Toc426736289)

[8. 按键防抖模块btnDebounce 7](#_Toc426736290)

[9. IO转换模块IOConv 7](#_Toc426736291)

[10. 七段数码管显示模块segDisplay 8](#_Toc426736292)

[11. PmodCLP显示模块PmodCLP 9](#_Toc426736293)

[12. 总模块N3Adapter 9](#_Toc426736294)

[实现与仿真 10](#_Toc426736295)

[1. 时钟管理模块CLKctrl 10](#_Toc426736296)

[2. 取指管理模块IFctrl 10](#_Toc426736297)

[3. 运算管理模块EXctrl 11](#_Toc426736298)

[4. 存储管理模块MEMctrl 13](#_Toc426736299)

[5. 回写管理模块WBctrl 13](#_Toc426736300)

[6. 访存控制模块ACctrl 14](#_Toc426736301)

[7. 主模块CPU 16](#_Toc426736302)

[8. 按键防抖模块btnDebounce 20](#_Toc426736303)

[9. IO转换模块IOConv 20](#_Toc426736304)

[10. 七段数码管显示模块segDisplay 21](#_Toc426736305)

[11. PmodCLP显示模块PmodCLP 22](#_Toc426736306)

[12. 总模块N3Adapter 30](#_Toc426736307)

[问题与解决方案 33](#_Toc426736308)

# 前言

一直以来，提到CPU，我们都会觉得这是一个复杂而又神奇的东西，能够自行设计和实现一个CPU，对我们来说更是想都不敢想的事。但是现在，VHDL（VHSIC hardware description language）和FPGA（Field-Programmable Gate Array）的发展，使得计算机组成原理的初学者自己设计并实现一个CPU有了可能。我们只需要使用硬件描述语言在寄存器、门电路层面上描述我们想实现的硬件功能，并将其综合、编译，下载到FPGA上运行，就可以得到我们想要的功能。本文以Digilent® Nexys 3™开发板为开发平台，介绍一个简单的RISC指令集单进程CPU的设计与实现。

# CPU结构介绍

CPU是一个复杂的硬件，它由多个模块协同工作来完成一件事（一条指令）。回顾计算机组成原理中所讲的内容，CPU执行一条指令可以分为四个周期：取指周期、间址（间接寻址）周期、执行周期、中断周期。本次要设计的CPU为简单起见，采用自定义的RISC指令集，不设间接寻址指令，也不支持中断，所以只需设取指周期和执行周期。如果按节拍再细化，执行周期又可以分为运算阶段（给出相应的运算结果、访存地址等）、存储阶段（进行访存，读写数据）和回写阶段（更新寄存器和PC）。因此，我们将每个指令周期设计成包含四个机器周期，每个机器周期包含一个节拍，并且每个节拍在一个特定的模块进行，其余模块配合其工作。

抛开Nexys 3开发板，我们先来看这个CPU设计方案的核心部分（详细设计框图见下页）。

这个CPU主要由以下几个模块构成：

1. 时钟管理模块CLK

这个模块是一个节拍发生器，用它来把时钟信号转化成控制四个模块分别工作的控制信号。4-节拍发生器有多种实现方法，比如用一个计数器和一个2-4译码器来做，也可以用一个四位串行移位寄存器来做。为简单起见，这里用四位串行移位寄存器实现。当rst信号有效时，节拍发生器复位到0001，即回写节拍；rst信号无效时，在每次时钟上升沿，从一个节拍进入下一个节拍。时钟管理模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 时钟管理模块CLK | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **CLK** | 1 | I | 处理器板 | 系统时钟 |
| **RST** | 1 | I | 处理器板 | 高电平复位 |
| **T0** | 1 | O | 取指管理模块 | 工作信号 |
| **T1** | 1 | O | 运算管理模块 | 工作信号 |
| **T2** | 1 | O | 存储管理模块 | 工作信号 |
| **T3** | 1 | O | 回写管理模块 | 工作信号 |

1. 取指管理模块IF

此模块主要负责取指令、维护与输出PC值。在T0节拍，它会向访存控制模块发送读指令的命令（RDIR），在T0的前半拍（CLK=’1’）时接收访存传过来的指令，然后将读到的指令锁存并送到IRout，供其他模块分析使用。在T0节拍的上升沿，它会将PC更新为从回写管理模块送过来的PCnew。PCnew可能是PC+1，也可能是复位后的初始值，也可能是跳转后的新地址。取指管理模块的输入输出接口定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 取指管理模块IF | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **T0** | 1 | I | 时钟管理模块 | 工作节拍信号 |
| **PCnew** | 16 | I | 回写管理模块 | T0上升沿PC更新为Pcnew |
| **CLK** | 1 | I | 处理器板 | 时钟 |
| **IRdata** | 16 | I | 访存控制模块 | 指令字 |
| **RST** | 1 | I | 处理器板 | 复位 |
| **PCout** | 16 | O | 访存、回写 | 当前指令地址 |
| **RDIR** | 1 | O | 访存控制模块 | 命令读指令字 |
| **IRout** | 16 | O | 运算、存储、回写 | 提供指令 |

1. 运算管理模块EX

这个模块中有8个通用寄存器，可供程序使用，还有Addr、A、B、Cy暂存器，供CPU执行运算或访存操作时使用。在T1节拍的上升沿，该模块进行初始化操作，根据指令中的地址码，将A置为Reg(Ad1(IR))的值，B置为Reg(Ad2(IR))的值，Addr置为R7//Ad(IR)的值（//代表扩充寻址，即把R7的值连接在Ad（IR）的前面）。在T1节拍中间CLK的下降沿，该模块执行运算操作，根据指令操作码来执行不同操作，如Cy置位、加减法以及把要回写或保存的值输出到ALUOUT等。该模块还接受Rupdate、Raddr和Rdata信号，在Rupdate的上升沿将Raddr代表的寄存器的值更新为Rdata。运算管理模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 运算管理模块EX | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **T1** | 1 | I | 时钟管理模块 | 工作节拍信号 |
| **IR** | 16 | I | 取指管理模块 | 指令字 |
| **CLK** | 1 | I | 处理器板 | 时钟 |
| **Rupdate** | 1 | I | 回写管理模块 | 回写寄存器信号 |
| **Raddr** | 3 | I | 回写管理模块 | 回写寄存器地址 |
| **Rdata** | 8 | I | 回写管理模块 | 回写寄存器数据 |
| **Addr** | 16 | O | 存储管理模块 | Reg(R7)//Ad(IR) |
| **ALUOUT** | 8 | O | 访存、回写 | 运算结果、欲存数据 |
| **Cout** | 1 | O | 七段数码管显示 | 进位 |
| **R0~R7** | 8\*8 | O | PmodCLP显示 | 通用寄存器 |

1. 存储管理模块MEM

这个模块主要负责在T2节拍向访存控制模块送访存地址，根据不同的指令向访存控制模块发相应的访存信号（访存、访IO、读、写），并将访存控制模块传来的数据暂存到Rtemp中。存储管理模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 存储管理模块MEM | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **T2** | 1 | I | 时钟管理模块 | 工作节拍信号 |
| **OP** | 5 | I | 取指管理模块 | 操作码 |
| **CLK** | 1 | I | 处理器板 | 时钟 |
| **DATA** | 8 | I | 访存控制模块 | 取到的数据 |
| **Addrin** | 16 | I | 运算管理模块 | 实际Addr（Port） |
| **Addr** | 16 | O | 访存、回写 | Addr、Port |
| **Rtemp** | 8 | O | 回写管理模块 | 送Rdata的数据 |
| **WR** | 1 | O | 访存控制模块 | 高电平写 |
| **RD** | 1 | O | 访存控制模块 | 高电平读 |
| **nMEM** | 1 | O | 访存控制模块 | 低电平访主存 |
| **nIO** | 1 | O | 访存控制模块 | 低电平访IO |

1. 回写管理模块WB

回写管理模块，顾名思义，就是管理回写寄存器和回写PC操作的。在T3节拍，对于需要更新寄存器值的指令，它负责将存储管理模块暂存的Rtemp送到运算管理模块，并向运算管理模块发送寄存器地址Raddr和更新信号Rupdate。它还负责生成接下来要执行的指令的地址PCnew，由取指管理模块在T0上升沿更新到PC。当有复位信号时，PCnew为初始值；当需要进行跳转时，PCnew为跳转后的值；否则PCnew为PC+2，代表当前指令的下一个指令字的地址（按字节编址，一个字有两个字节）。回写管理模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 回写管理模块WB | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **T3** | 1 | I | 时钟管理模块 | 工作节拍信号 |
| **OP** | 5 | I | 取指管理模块 | 操作码 |
| **RST** | 1 | I | 处理器板 | 高电平复位 |
| **Rtemp** | 8 | I | 存储管理模块 | 访存回写寄存器的数据 |
| **PC** | 16 | I | 取指管理模块 | 当前指令地址 |
| **Addr** | 16 | I | 存储管理模块 | 用于与PC相加 |
| **ALUOUT** | 8 | I | 运算管理模块 | 运算结果送Rdata回写 |
| **AD1** | 3 | I | 取指管理模块 | IR(10~8)，目标寄存器 |
| **Raddr** | 3 | O | 运算管理模块 | 回写寄存器的地址 |
| **Rdata** | 8 | O | 运算管理模块 | 回写寄存器的数据 |
| **Rupdate** | 1 | O | 运算管理模块 | 回写寄存器信号 |
| **PCnew** | 16 | O | 取指管理模块 | 下一个PC |

1. 访存控制模块AC

这个模块不受节拍控制，它接受存储管理模块的命令，进行访存或访IO的操作。当读指令时，访存地址为PC值，否则为Addr值。在进行读操作时，要把数据总线置为高阻，在读一个字时，nBLE和nBHE要同时置为0，在读一个字节时，nBLE和nBHE的值视地址（按字节编址）的最低位而定（存储器是按字编址，支持选择高低字节）。访存控制模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 访存控制模块AC | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **nIO** | 1 | I | 存储管理模块 | 低电平访IO |
| **RD** | 1 | I | 存储管理模块 | 高电平读 |
| **WR** | 1 | I | 存储管理模块 | 高电平写 |
| **nMEM** | 1 | I | 存储管理模块 | 低电平访主存 |
| **RDIR** | 1 | I | 取指管理模块 | 取指令 |
| **PC** | 16 | I | 取指管理模块 | 要取的指令地址 |
| **ALUOUT** | 8 | I | 运算管理模块 | 要写入的数据 |
| **Addr** | 16 | I | 存储管理模块 | 访存地址 |
| **DBUS** | 16 | inout | 主存 | 数据总线 |
| **IODB** | 8 | inout | IO管理 | 数据总线 |
| **IOAD** | 2 | O | IO管理 | 地址总线 |
| **nBLE** | 1 | O | 主存 | 低字节访问允许 |
| **Rtemp** | 8 | O | 存储管理模块 | 取出的数据 |
| **IR** | 16 | O | 取指管理模块 | 取出的指令 |
| **nMREQ** | 1 | O | 主存 | 访存 |
| **nPREQ** | 1 | O | IO管理 | 访IO |
| **nRD** | 1 | O | 主存 | 读 |
| **nWR** | 1 | O | 主存 | 写 |
| **nPRD** | 1 | O | IO管理 | 读 |
| **nPWR** | 1 | O | IO管理 | 写 |
| **ABUS** | 16 | O | 主存 | 地址总线 |
| **nBHE** | 1 | O | 主存 | 高字节访问允许 |

1. 主模块CPU

这个模块并不执行实质性的功能，只是用元件例化的方法把各个分模块连接在一起，并且向开发板提供必要的输出供用户观察、调试。这个模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 主模块CPU | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **RST** | 1 | I | 处理器板 | 高电平复位 |
| **CLK** | 1 | I | 处理器板 | 系统时钟 |
| **IR** | 16 | O | 取指管理模块 | 指令 |
| **PC** | 16 | O | 取指管理模块 | 指令地址 |
| **R0~R7** | 8\*8 | O | 运算管理模块 | 8个通用寄存器 |
| **T** | 4 | O | 时钟管理模块 | 四个节拍 |
| **DBUS** | 16 | inout | 主存 | 数据总线 |
| **IODB** | 8 | inout | IO管理 | 数据总线 |
| **IOAD** | 2 | O | IO管理 | 地址总线 |
| **nBLE** | 1 | O | 主存 | 低字节访问允许 |
| **Rtemp** | 8 | O | 存储管理模块 | 取出的数据 |
| **IR** | 16 | O | 取指管理模块 | 取出的指令 |
| **nMREQ** | 1 | O | 主存 | 访存 |
| **nPREQ** | 1 | O | IO管理 | 访IO |
| **nRD** | 1 | O | 主存 | 读 |
| **nWR** | 1 | O | 主存 | 写 |
| **nPRD** | 1 | O | IO管理 | 读 |
| **nPWR** | 1 | O | IO管理 | 写 |
| **ABUS** | 16 | O | 主存 | 地址总线 |
| **nBHE** | 1 | O | 主存 | 高字节访问允许 |

为了使这个CPU能更好地运行在Nexys 3开发板上，以及显示出必要的信息供观察和调试用，我除了给CPU增加了IR、PC、T、Cout（进位）、R0~R7的输出外，又增加了以下模块：

1. 按键防抖模块btnDebounce

Nexys 3开发板上的按钮是弹性按钮，很容易抖动，导致按一下可能会执行数个节拍，影响观察，抖动剧烈时甚至会影响时序。这里我用了一个时钟计数电路，实现了按键的防抖。经过它处理的按钮，平常都处于高电平，只有在按下的瞬间会产生一个时钟周期的低电平。这个模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 按键防抖模块btnDebounce | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **btn** | 1 | I | 处理器板 | 按钮 |
| **CLK** | 1 | I | 处理器板 | 系统时钟 |
| **btn\_deb** | 1 | O | N3Adapter | 消抖后的按钮 |

1. IO转换模块IOConv

由于Nexys 3开发板上的输入输出接口有限，调试和观察IO指令不太方便，我增加了一个IO转换模块，专门处理IO操作。这个模块使用了开发板上的8个开关和8个led灯，总共支持4个外设，0号和1号外设是输入外设，代表switch(7 downto 0)和switch(0 to 7)，2号和3号是输出外设，代表led(7 downto 0)和led(0 to 7)。当nPREQ信号和nPWR信号有效时，led灯亮，显示出OUT指令输出的数据；当nPREQ和nPRD信号有效时，该控制器会将8个开关的值送到数据总线上，被CPU取进寄存器。这个模块的输入输出接口信号定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IO转换模块IOConv | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **IOAD** | 2 | I | CPU | IO地址总线 |
| **IODB** | 8 | inout | CPU | IO数据总线 |
| **nPREQ** | 1 | I | CPU | 访IO控制信号 |
| **nPRD** | 1 | I | CPU | 读信号 |
| **nPWR** | 1 | I | CPU | 写信号 |
| **sw** | 8 | I | 开发板 | 拨码开关 |
| **led** | 8 | O | 开发板 | led灯 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IO转换模块IOConv | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **IOAD** | 2 | I | CPU | IO地址总线 |
| **IODB** | 8 | inout | CPU | IO数据总线 |
| **nPREQ** | 1 | I | CPU | 访IO控制信号 |
| **nPRD** | 1 | I | CPU | 读信号 |
| **nPWR** | 1 | I | CPU | 写信号 |
| **sw** | 8 | I | 开发板 | 拨码开关 |
| **led** | 8 | O | 开发板 | led灯 |

1. 七段数码管显示模块segDisplay

这个模块负责PC、IR、节拍、Cy等信号显示到四个七段数码管上。当按住五个按钮中最上方的那个按键btnu时，显示的是IR，且四个小数点亮表示Cy为1，松开btnu时，显示的是PC，且四个小数点从左到右分别代表T0、T1、T2、T3。为了将四位二进制数显示成十六进制字符，我实现了一个通用的转换函数conv\_seg。另外，该七段数码管需要一个频率合适的时钟来控制刷新，所以其中还包含一个分频电路。这个模块的输入输出接口定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 七段数码管显示模块segDisplay | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **PC** | 16 | I | CPU | 指令地址 |
| **IR** | 16 | I | CPU | 指令 |
| **btnu** | 1 | I | 开发板 | 最上的按键 |
| **T** | 4 | I | CPU | 节拍 |
| **clk** | 1 | I | 开发板 | 时钟 |
| **Cy** | 1 | I | CPU | 进位 |
| **an** | 4 | O | 开发板 | 数码管选择信号 |
| **seg** | 8 | O | 开发板 | 数码管显示数据 |

1. PmodCLP显示模块PmodCLP

由于Nexys 3开发板的输出接口太少，不容易观察CPU内部运行情况，我使用了PmodCLP扩展显示模块，显示出了当前运行的指令的汇编代码和8个通用寄存器的值。这个模块修改自PmodCLP的官方示例程序，采用了状态机，控制PmodCLP转换状态并用循环的方式依次输出每个字符。这个模块的输入输出接口定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PmodCLP显示模块PmodCLP | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **RESET** | 16 | I | 开发板 | 复位显示屏 |
| **clk** | 16 | I | 开发板 | 时钟 |
| **IR** | 1 | I | CPU | 指令 |
| **R0~R7** | 8\*8 | I | CPU | 通用寄存器 |
| **JC** | 8 | O | 开发板 | 数据线 |
| **JD** | 3 | O | 开发板 | 控制线 |

1. 总模块N3Adapter

这个模块比较简单，就是把Nexys 3开发板上的按钮、输入输出接口以及访存接口和本次设计的CPU输入输出相适应。主要难点在于N3使用的Ram是按字编址，支持高低字节访问，而CPU是按字节编址的设计。解决方法就是把CPU地址线的高15位连到Ram地址线上，最低位连到RamLB（低电平有效，控制取低字节），最低位的非连到RamUB上。这个模块的输入输出接口定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 总模块N3Adapter | | | | |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **CLK** | 1 | I | 开发板 | 板载时钟 |
| **sw** | 8 | I | 开发板 | 拨码开关 |
| **led** | 8 | O | 开发板 | led灯 |
| **btnr** | 1 | I | 开发板 | 右按键 |
| **btns** | 1 | I | 开发板 | 中按键 |
| **btnu** | 1 | I | 开发板 | 上按键 |
| **an** | 4 | O | 开发板 | 七段数码管控制 |
| **seg** | 8 | O | 开发板 | 七段数码管控制 |
| **JC** | 8 | O | 开发板 | PmodCLP |
| **JD** | 3 | O | 开发板 | PmodCLP |
| **MemOE** | 1 | O | 开发板 | 读信号 |
| **MemWR** | 1 | O | 开发板 | 写信号 |
| **MemAdv** | 1 | O | 开发板 | 这里没用，置0 |
| **MemWait** | 1 | O | 开发板 | 这里没用，置0 |
| **RamCS** | 1 | O | 开发板 | 片选 |
| **RamCRE** | 1 | O | 开发板 | 这里没用，置0 |
| **RamUB** | 1 | O | 开发板 | 高位 |
| **RamLB** | 1 | O | 开发板 | 低位 |
| **MemAdr** | 26 | O | 开发板 | 地址总线 |
| **MemDB** | 16 | inout | 开发板 | 数据总线 |

# 实现与仿真

这里介绍以上所有模块的VHDL实现，以及CPU及其分模块的仿真波形。为N3开发板而额外加的模块没有仿真。

1. 时钟管理模块CLKctrl

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity CLKctrl is

Port ( CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

T : out STD\_LOGIC\_VECTOR (3 downto 0));

end CLKctrl;

architecture Behavioral of CLKctrl is

signal tmp : STD\_LOGIC\_VECTOR (3 downto 0);

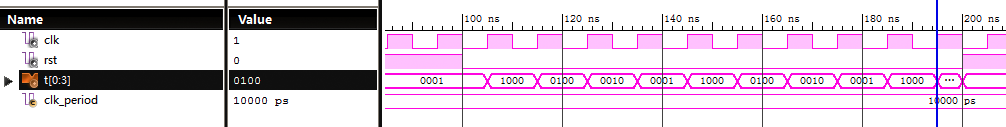
begin

T <= tmp;

tmp <= "1000" WHEN rst = '1' ELSE

tmp(2 downto 0) & tmp(3) WHEN rising\_edge(clk);

end Behavioral;



1. 取指管理模块IFctrl

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity IFctrl is

Port ( T0 : in STD\_LOGIC;

CLK : in STD\_LOGIC;

PCnew : in STD\_LOGIC\_VECTOR (15 downto 0);

IRdata : in STD\_LOGIC\_VECTOR (15 downto 0); -- 从访存控制模块发来的指令字

PCout : out STD\_LOGIC\_VECTOR (15 downto 0); -- 发送指令地址

RDIR : out STD\_LOGIC; -- 高电平令访存控制模块读指令字

IRout : out STD\_LOGIC\_VECTOR (15 downto 0)); -- 指令送往其他模块

end IFctrl;

architecture Behavioral of IFctrl is

signal PC, IR : STD\_LOGIC\_VECTOR(15 downto 0) := X"0000";

signal nextPC : STD\_LOGIC\_VECTOR(15 downto 0) := X"0000";

begin

RDIR <= T0; -- 控制读指令

process (T0)

begin

if T0 = '1' and T0'event then

PC <= PCnew;

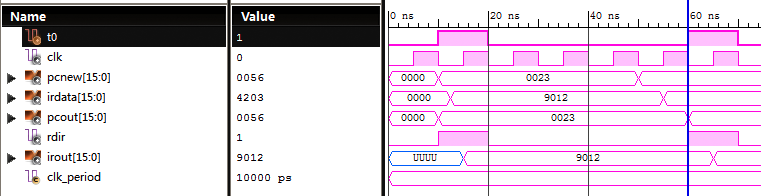
end if;

end process;

PCout <= PC;

IRout <= IRdata when (T0 and CLK) = '1';

end Behavioral;



1. 运算管理模块EXctrl

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity EXctrl is

Port ( CLK : in STD\_LOGIC;

T1 : in STD\_LOGIC;

Rupdate : in STD\_LOGIC;

Raddr : in STD\_LOGIC\_VECTOR (2 downto 0);

Rdata : in STD\_LOGIC\_VECTOR (7 downto 0);

IR : in STD\_LOGIC\_VECTOR (15 downto 0);

Cout : out std\_logic;

R0 : out STD\_LOGIC\_VECTOR (7 downto 0);

R1 : out STD\_LOGIC\_VECTOR (7 downto 0);

R2 : out STD\_LOGIC\_VECTOR (7 downto 0);

R3 : out STD\_LOGIC\_VECTOR (7 downto 0);

R4 : out STD\_LOGIC\_VECTOR (7 downto 0);

R5 : out STD\_LOGIC\_VECTOR (7 downto 0);

R6 : out STD\_LOGIC\_VECTOR (7 downto 0);

R7 : out STD\_LOGIC\_VECTOR (7 downto 0);

Addr : out STD\_LOGIC\_VECTOR (15 downto 0) := X"0000";

ALUOUT : out STD\_LOGIC\_VECTOR (7 downto 0) := X"00");

end EXctrl;

architecture Behavioral of EXctrl is

type REGARR is array(7 downto 0) of STD\_LOGIC\_VECTOR(7 downto 0);

signal Reg : REGARR := (X"77", X"66", X"55", X"44", X"33", X"22", X"11", X"00");

signal A, B : STD\_LOGIC\_VECTOR(7 downto 0) := X"00";

signal Cy : STD\_LOGIC := '0';

begin

Cout <= Cy;

R0 <= Reg(0);

R1 <= Reg(1);

R2 <= Reg(2);

R3 <= Reg(3);

R4 <= Reg(4);

R5 <= Reg(5);

R6 <= Reg(6);

R7 <= Reg(7);

-- Prepare A, B, Addr

process (T1, IR)

begin

if T1'event and T1 = '1' then

A <= Reg(conv\_integer(IR(10 downto 8)));

B <= Reg(conv\_integer(IR(2 downto 0)));

Addr(7 downto 0) <= IR(7 downto 0);

if IR(15 downto 11) = "00010" then -- JZ

Addr(15 downto 8) <= (others => IR(7)); -- 符号位扩展

else

Addr(15 downto 8) <= Reg(7); -- R7扩展寻址

end if;

end if;

end process;

-- Calculate at falling\_edge

process (A, B, Cy, IR, T1, CLK)

variable result : STD\_LOGIC\_VECTOR(8 downto 0);

begin

if T1 = '1' and falling\_edge(CLK) then

case IR(15 downto 11) is

when "10100" => Cy <= '0'; -- CLRC(A)

when "10110" => Cy <= '1'; -- SETC(B)

when "00110" => result := (A(7)&A) + (B(7)&B) + Cy; -- ADC(3)

ALUOUT <= result(7 downto 0);

Cy <= result(8); -- 第8位判断进位

when "00100" => result := (A(7)&A) - (B(7)&B) - Cy; -- SBB(2)

ALUOUT <= result(7 downto 0);

Cy <= result(8);

when "01000" => ALUOUT <= IR(7 downto 0); -- MVI(4)

when "01010" => ALUOUT <= B; -- MOV(5)

when "01100" => ALUOUT <= A; -- STA(6)

when "00010" => ALUOUT <= A; -- JZ(1)

when "10010" => ALUOUT <= A; -- OUT(9)

when others => null;

end case;

end if;

end process;

-- Writeback reg

process (Rupdate, Raddr, Rdata)

begin

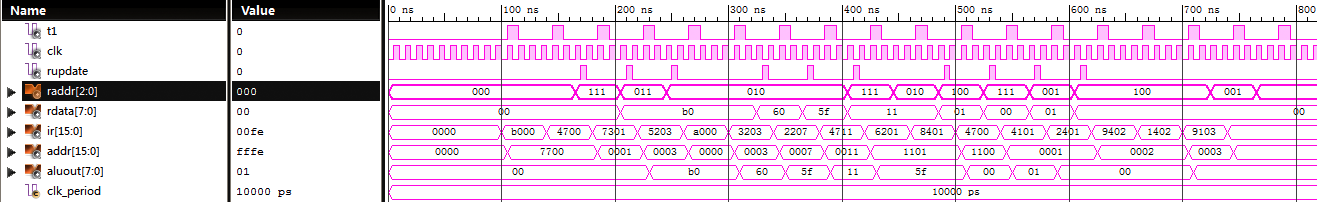
if Rupdate'event and Rupdate = '1' then

Reg(conv\_integer(Raddr)) <= Rdata;

end if;

end process;

end Behavioral;



1. 存储管理模块MEMctrl

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity MEMctrl is

Port ( CLK : in STD\_LOGIC;

Addrin : in STD\_LOGIC\_VECTOR (15 downto 0);

Addr : out STD\_LOGIC\_VECTOR (15 downto 0);

OP : in STD\_LOGIC\_VECTOR (15 downto 11); -- IR(15 downto 11)

DATA : in STD\_LOGIC\_VECTOR (7 downto 0);

T2 : in STD\_LOGIC;

Rtemp : out STD\_LOGIC\_VECTOR (7 downto 0);

nMEM : out STD\_LOGIC;

nIO : out STD\_LOGIC;

RD : out STD\_LOGIC;

WR : out STD\_LOGIC);

end MEMctrl;

architecture Behavioral of MEMctrl is

begin

-- 送出地址

Addr <= Addrin;

-- 读写控制

nMEM <= '0' when (T2 = '1' and (OP = "01110" or OP = "01100")) else '1';

nIO <= '0' when (T2 = '1' and (OP = "10000" or OP = "10010")) else '1';

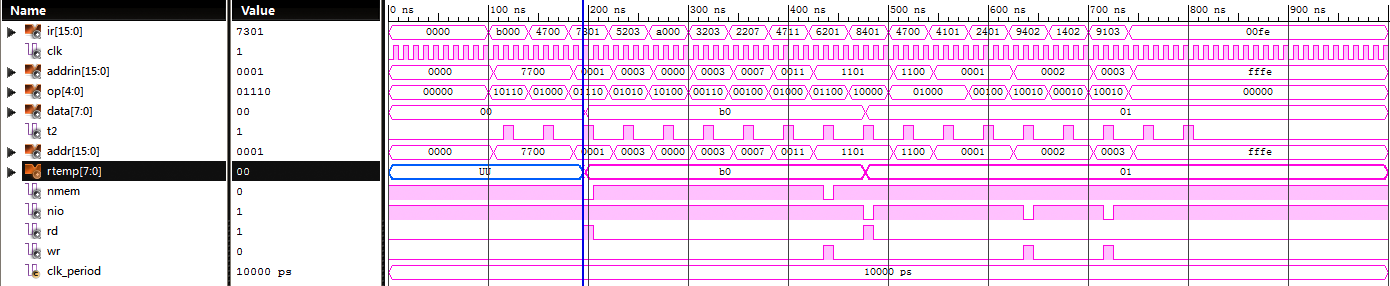
WR <= '1' when (T2 = '1' and (OP = "01100" or OP = "10010")) else '0';

RD <= '1' when (T2 = '1' and (OP = "01110" or OP = "10000")) else '0';

-- 更新Rtemp

Rtemp <= DATA when (T2 = '1' and (OP = "01110" or OP = "10000")) else unaffected;

end Behavioral;



1. 回写管理模块WBctrl

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity WBctrl is

Port ( RST : in STD\_LOGIC;

Rtemp : in STD\_LOGIC\_VECTOR (7 downto 0);

PC : in STD\_LOGIC\_VECTOR (15 downto 0);

Addr : in STD\_LOGIC\_VECTOR (15 downto 0);

ALUOUT : in STD\_LOGIC\_VECTOR (7 downto 0);

T3 : in STD\_LOGIC;

OP : in STD\_LOGIC\_VECTOR (15 downto 11); -- IR(15 downto 11)

AD1 : in STD\_LOGIC\_VECTOR (10 downto 8); -- IR(10 downto 8)

Raddr : out STD\_LOGIC\_VECTOR (2 downto 0);

Rdata : out STD\_LOGIC\_VECTOR (7 downto 0);

Rupdate : out STD\_LOGIC;

PCnew : out STD\_LOGIC\_VECTOR (15 downto 0));

end WBctrl;

architecture Behavioral of WBctrl is

begin

-- 提供回写内容

Rdata <= Rtemp when (OP = "10000" or OP = "01110") else -- IN / LDA

ALUOUT;

Raddr <= AD1;

process (RST, OP, ALUOUT, T3)

begin

if RST = '1' then

PCnew <= X"0000";

elsif T3 = '1' and T3'event then

if OP = "00000" then

PCnew <= Addr;

elsif (OP = "00010" and ALUOUT = X"00") then

PCnew <= PC + Addr + 2;

else

PCnew <= PC + 2;

end if;

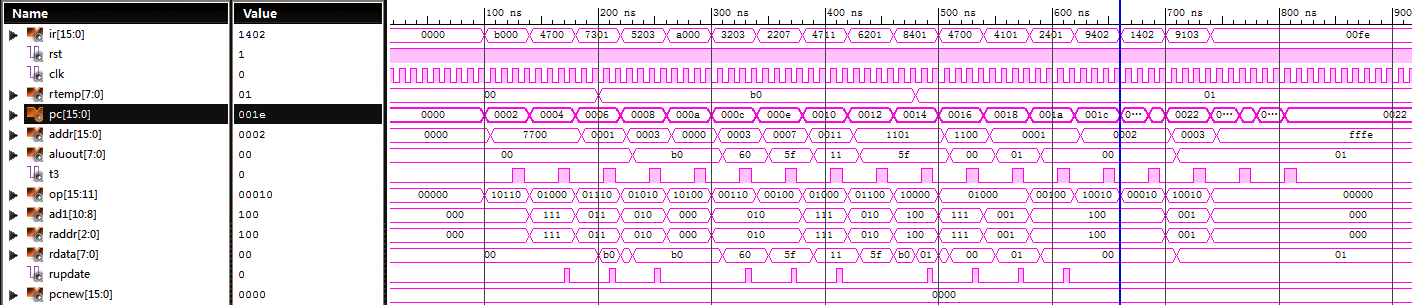
end if;

end process;

-- 回写控制信号

Rupdate <= '1' when (T3 = '1' and (OP = "10000" or OP = "01110" or OP = "00110" or OP = "00100" or OP = "01010" or OP = "01000")) else '0'; -- IN, LDA, ADC, SBB, MVI, MOV

end Behavioral;



1. 访存控制模块ACctrl

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ACctrl is

Port ( nIO : in STD\_LOGIC;

nMEM : in STD\_LOGIC;

RD : in STD\_LOGIC;

WR : in STD\_LOGIC;

RDIR : in STD\_LOGIC;

PC : in STD\_LOGIC\_VECTOR (15 downto 0);

Addr : in STD\_LOGIC\_VECTOR (15 downto 0);

ALUOUT : in STD\_LOGIC\_VECTOR (7 downto 0);

nBLE : out STD\_LOGIC;

nBHE : out STD\_LOGIC;

ABUS : out STD\_LOGIC\_VECTOR (15 downto 0);

nRD : out STD\_LOGIC;

nWR : out STD\_LOGIC;

nMREQ : out STD\_LOGIC;

DBUS : inout STD\_LOGIC\_VECTOR (15 downto 0);

IOAD : out STD\_LOGIC\_VECTOR (1 downto 0);

IODB : inout STD\_LOGIC\_VECTOR (7 downto 0);

nPRD : out STD\_LOGIC;

nPWR : out STD\_LOGIC;

nPREQ : out STD\_LOGIC;

IR : out STD\_LOGIC\_VECTOR (15 downto 0);

Rtemp : out STD\_LOGIC\_VECTOR (7 downto 0));

end ACctrl;

architecture Behavioral of ACctrl is

signal address : std\_logic\_vector (15 downto 0);

begin

-- 形成访存/访IO的地址

address <= Addr when (nMEM = '0' or nIO = '0') else

PC when RDIR = '1' else

address;

process (RDIR, WR, RD, nIO, nMEM, DBUS, ALUOUT, IODB, address)

begin

if RDIR = '1' then

nMREQ <= '0';

nBLE <= '0';

nBHE <= '0';

nRD <= '0';

nWR <= '1';

ABUS <= address;

DBUS <= (others => 'Z');

IR <= DBUS;

elsif nMEM = '0' then

nMREQ <= '0';

ABUS <= address;

nBLE <= address(0);

nBHE <= not address(0);

nRD <= not RD;

nWR <= not WR;

if RD = '1' and address(0) = '0' then

DBUS <= (others => 'Z');

Rtemp <= DBUS(7 downto 0);

elsif RD = '1' and address(1) = '0' then

DBUS <= (others => 'Z');

Rtemp <= DBUS(15 downto 8);

elsif WR = '1' then

DBUS <= ALUOUT&ALUOUT;

end if;

elsif nIO = '0' then

nPREQ <= '0';

IOAD <= address(1 downto 0);

nPRD <= not RD;

nPWR <= not WR;

if RD = '1' then

IODB <= (others => 'Z');

Rtemp <= IODB;

elsif WR = '1' then

IODB <= ALUOUT;

end if;

else

nMREQ <= '1';

nPREQ <= '1';

ABUS <= address;

DBUS <= (others => 'Z');

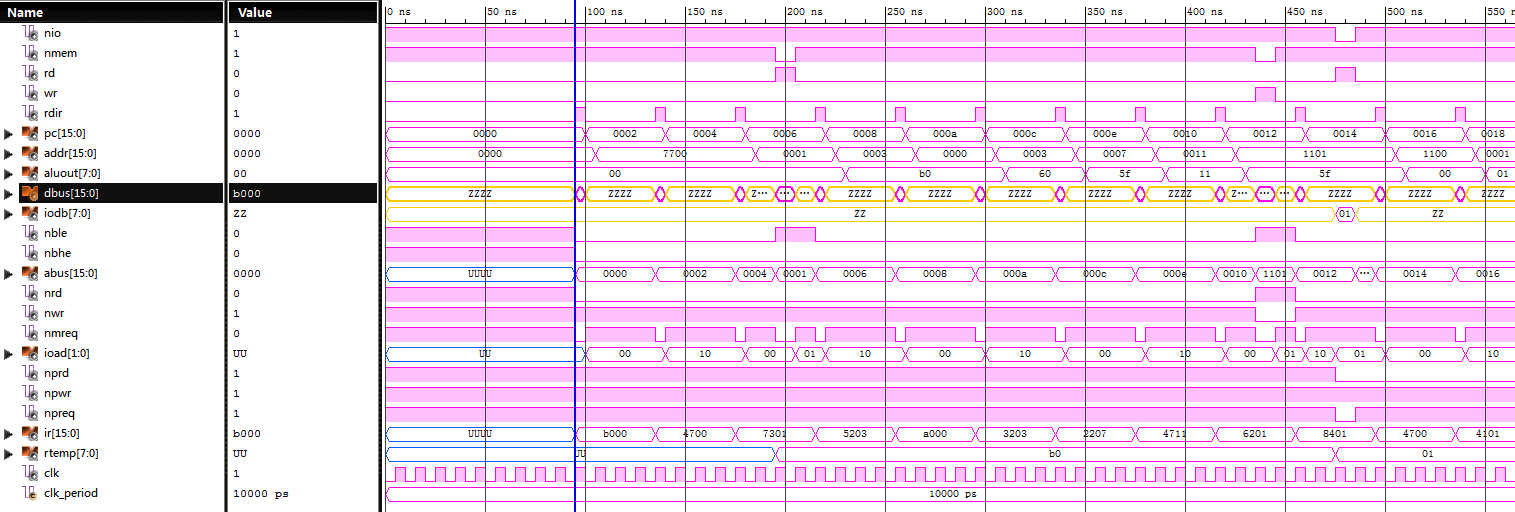
IOAD <= address(1 downto 0);

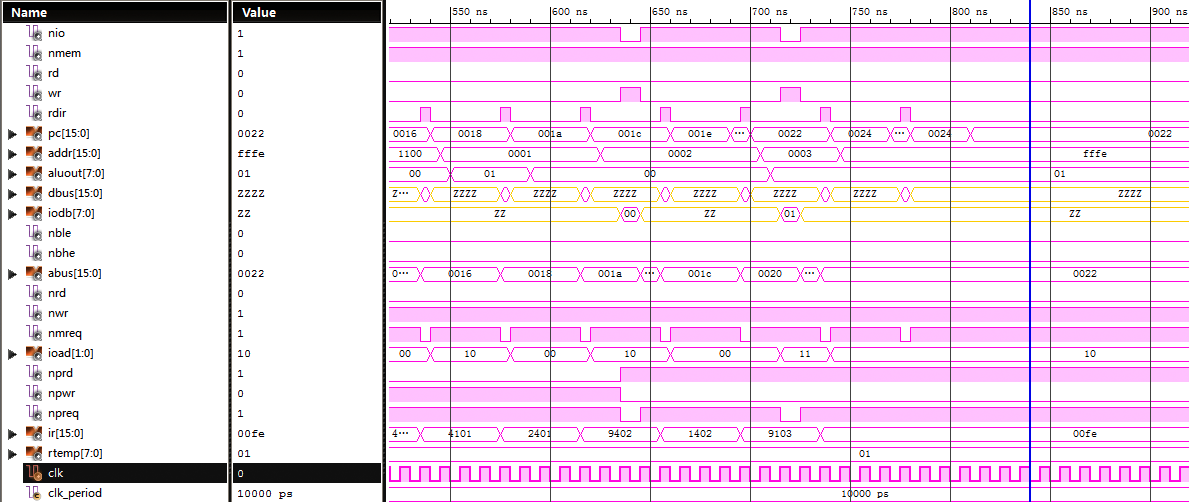
IODB <= (others => 'Z');

end if;

end process;

end Behavioral;





1. 主模块CPU

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity CPU is

Port ( RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

DBUS : inout STD\_LOGIC\_VECTOR (15 downto 0);

IODB : inout STD\_LOGIC\_VECTOR (7 downto 0);

ABUS : out STD\_LOGIC\_VECTOR (15 downto 0);

IOAD : out STD\_LOGIC\_VECTOR (1 downto 0);

nMREQ : out STD\_LOGIC;

nPREQ : out STD\_LOGIC;

nRD : out STD\_LOGIC;

nWR : out STD\_LOGIC;

nBHE : out STD\_LOGIC;

nBLE : out STD\_LOGIC;

nPRD : out STD\_LOGIC;

nPWR : out STD\_LOGIC;

Cout : out STD\_LOGIC;

IR : out STD\_LOGIC\_VECTOR (15 downto 0);

PC : out STD\_LOGIC\_VECTOR (15 downto 0);

R0 : out STD\_LOGIC\_VECTOR (7 downto 0);

R1 : out STD\_LOGIC\_VECTOR (7 downto 0);

R2 : out STD\_LOGIC\_VECTOR (7 downto 0);

R3 : out STD\_LOGIC\_VECTOR (7 downto 0);

R4 : out STD\_LOGIC\_VECTOR (7 downto 0);

R5 : out STD\_LOGIC\_VECTOR (7 downto 0);

R6 : out STD\_LOGIC\_VECTOR (7 downto 0);

R7 : out STD\_LOGIC\_VECTOR (7 downto 0);

T : out STD\_LOGIC\_VECTOR (3 downto 0));

end CPU;

architecture Behavioral of CPU is

COMPONENT CLKctrl

PORT(

CLK : IN std\_logic;

RST : IN std\_logic;

T : OUT std\_logic\_vector(3 downto 0)

);

END COMPONENT;

--///////////////////////////////////////////////

COMPONENT IFctrl

PORT(

T0 : IN std\_logic;

CLK : IN std\_logic;

PCnew : IN std\_logic\_vector(15 downto 0);

IRdata : IN std\_logic\_vector(15 downto 0);

PCout : OUT std\_logic\_vector(15 downto 0);

RDIR : OUT std\_logic;

IRout : OUT std\_logic\_vector(15 downto 0)

);

END COMPONENT;

--///////////////////////////////////////////////

COMPONENT EXctrl

PORT(

T1 : IN std\_logic;

CLK : IN std\_logic;

Rupdate : IN std\_logic;

Raddr : IN std\_logic\_vector(2 downto 0);

Rdata : IN std\_logic\_vector(7 downto 0);

IR : IN std\_logic\_vector(15 downto 0);

Cout : out std\_logic;

R0 : out STD\_LOGIC\_VECTOR (7 downto 0);

R1 : out STD\_LOGIC\_VECTOR (7 downto 0);

R2 : out STD\_LOGIC\_VECTOR (7 downto 0);

R3 : out STD\_LOGIC\_VECTOR (7 downto 0);

R4 : out STD\_LOGIC\_VECTOR (7 downto 0);

R5 : out STD\_LOGIC\_VECTOR (7 downto 0);

R6 : out STD\_LOGIC\_VECTOR (7 downto 0);

R7 : out STD\_LOGIC\_VECTOR (7 downto 0);

Addr : OUT std\_logic\_vector(15 downto 0);

ALUOUT : OUT std\_logic\_vector(7 downto 0)

);

END COMPONENT;

--///////////////////////////////////////////////

COMPONENT MEMctrl

PORT(

CLK : IN std\_logic;

Addrin : IN std\_logic\_vector(15 downto 0);

Addr : OUT std\_logic\_vector(15 downto 0);

OP : IN std\_logic\_vector(4 downto 0);

DATA : IN std\_logic\_vector(7 downto 0);

T2 : IN std\_logic;

Rtemp : OUT std\_logic\_vector(7 downto 0);

nMEM : OUT std\_logic;

nIO : OUT std\_logic;

RD : OUT std\_logic;

WR : OUT std\_logic

);

END COMPONENT;

--///////////////////////////////////////////////

COMPONENT WBctrl

PORT(

RST : IN std\_logic;

Rtemp : IN std\_logic\_vector(7 downto 0);

PC : IN std\_logic\_vector(15 downto 0);

Addr : IN std\_logic\_vector(15 downto 0);

ALUOUT : IN std\_logic\_vector(7 downto 0);

T3 : IN std\_logic;

OP : IN std\_logic\_vector(15 downto 11);

AD1 : IN std\_logic\_vector(10 downto 8);

Raddr : OUT std\_logic\_vector(2 downto 0);

Rdata : OUT std\_logic\_vector(7 downto 0);

Rupdate : OUT std\_logic;

PCnew : OUT std\_logic\_vector(15 downto 0)

);

END COMPONENT;

--///////////////////////////////////////////////

COMPONENT ACctrl

PORT(

nIO : IN std\_logic;

nMEM : IN std\_logic;

RD : IN std\_logic;

WR : IN std\_logic;

RDIR : IN std\_logic;

PC : IN std\_logic\_vector(15 downto 0);

Addr : IN std\_logic\_vector(15 downto 0);

ALUOUT : IN std\_logic\_vector(7 downto 0);

nBLE : OUT std\_logic;

nBHE : OUT std\_logic;

ABUS : OUT std\_logic\_vector(15 downto 0);

nRD : OUT std\_logic;

nWR : OUT std\_logic;

nMREQ : OUT std\_logic;

DBUS : INOUT std\_logic\_vector(15 downto 0);

IOAD : OUT std\_logic\_vector(1 downto 0);

IODB : INOUT std\_logic\_vector(7 downto 0);

nPRD : OUT std\_logic;

nPWR : OUT std\_logic;

nPREQ : OUT std\_logic;

IR : OUT std\_logic\_vector(15 downto 0);

Rtemp : OUT std\_logic\_vector(7 downto 0)

);

END COMPONENT;

--///////////////////////////////////////////////

signal Tout : std\_logic\_vector(3 downto 0) := "1000";

signal RDIR : std\_logic;

signal Addrin : std\_logic\_vector(15 downto 0) := (others => '0');

signal ALUOUT : std\_logic\_vector(7 downto 0) := (others => '0');

signal Addr : std\_logic\_vector(15 downto 0) := (others => '0');

signal Rtemp : std\_logic\_vector(7 downto 0) := (others => '0');

signal nMEM : std\_logic := '1';

signal nIO : std\_logic := '1';

signal RD : std\_logic := '0';

signal WR : std\_logic := '0';

signal Raddr : std\_logic\_vector(2 downto 0) := (others => '0');

signal Rdata : std\_logic\_vector(7 downto 0) := (others => '0');

signal Rupdate : std\_logic := '0';

signal PCnew : std\_logic\_vector(15 downto 0) := (others => '0');

signal IRdata, IRout, PCout : std\_logic\_vector(15 downto 0) := (others => '0');

signal Rtempdata : std\_logic\_vector(7 downto 0) := (others => '0');

begin

comCLK: CLKctrl PORT MAP (

CLK => CLK,

RST => RST,

T => Tout

);

comIF: IFctrl PORT MAP (

T0 => Tout(0),

CLK => CLK,

PCnew => PCnew,

IRdata => IRdata,

PCout => PCout,

RDIR => RDIR,

IRout => IRout

);

comEX: EXctrl PORT MAP (

T1 => Tout(1),

CLK => CLK,

Rupdate => Rupdate,

Raddr => Raddr,

Rdata => Rdata,

IR => IRout,

Cout => Cout,

R0 => R0,

R1 => R1,

R2 => R2,

R3 => R3,

R4 => R4,

R5 => R5,

R6 => R6,

R7 => R7,

Addr => Addrin,

ALUOUT => ALUOUT

);

comMEM: MEMctrl PORT MAP (

CLK => CLK,

Addrin => Addrin,

Addr => Addr,

OP => IRout(15 downto 11),

DATA => Rtempdata,

T2 => Tout(2),

Rtemp => Rtemp,

nMEM => nMEM,

nIO => nIO,

RD => RD,

WR => WR

);

comWB: WBctrl PORT MAP (

RST => RST,

Rtemp => Rtemp,

PC => PCout,

Addr => Addr,

ALUOUT => ALUOUT,

T3 => Tout(3),

OP => IRout(15 downto 11),

AD1 => IRout(10 downto 8),

Raddr => Raddr,

Rdata => Rdata,

Rupdate => Rupdate,

PCnew => PCnew

);

comAC: ACctrl PORT MAP (

nIO => nIO,

nMEM => nMEM,

RD => RD,

WR => WR,

RDIR => RDIR,

PC => PCout,

Addr => Addr,

ALUOUT => ALUOUT,

nBLE => nBLE,

nBHE => nBHE,

ABUS => ABUS,

nRD => nRD,

nWR => nWR,

nMREQ => nMREQ,

DBUS => DBUS,

IOAD => IOAD,

IODB => IODB,

nPRD => nPRD,

nPWR => nPWR,

nPREQ => nPREQ,

IR => IRdata,

Rtemp => Rtempdata

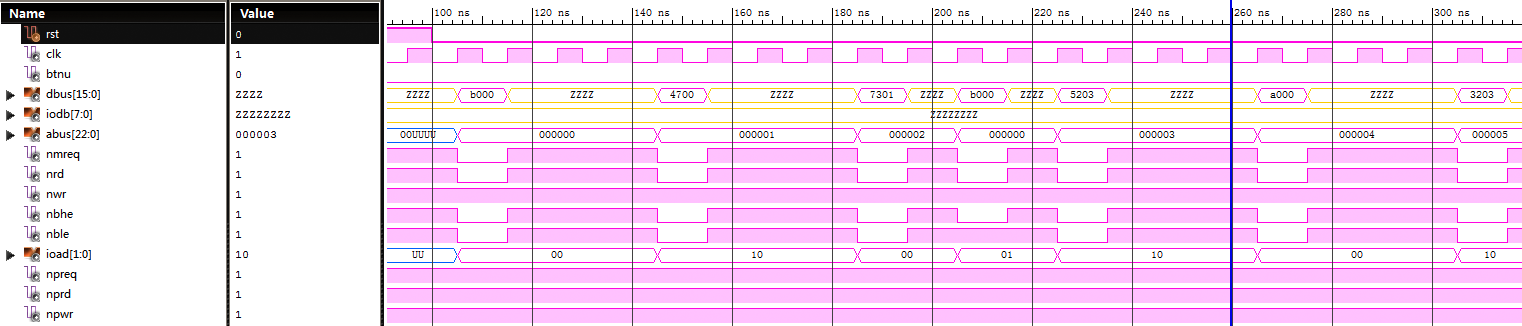
);

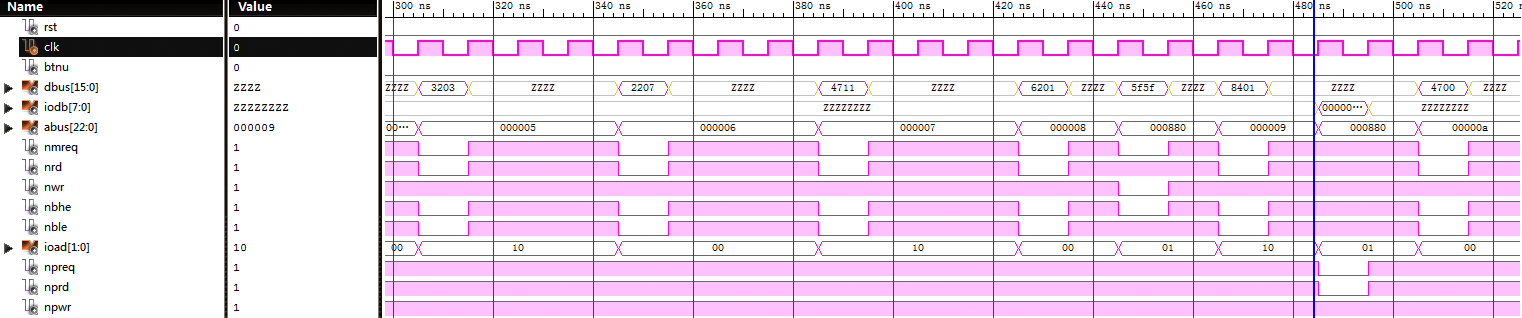
PC <= PCout;

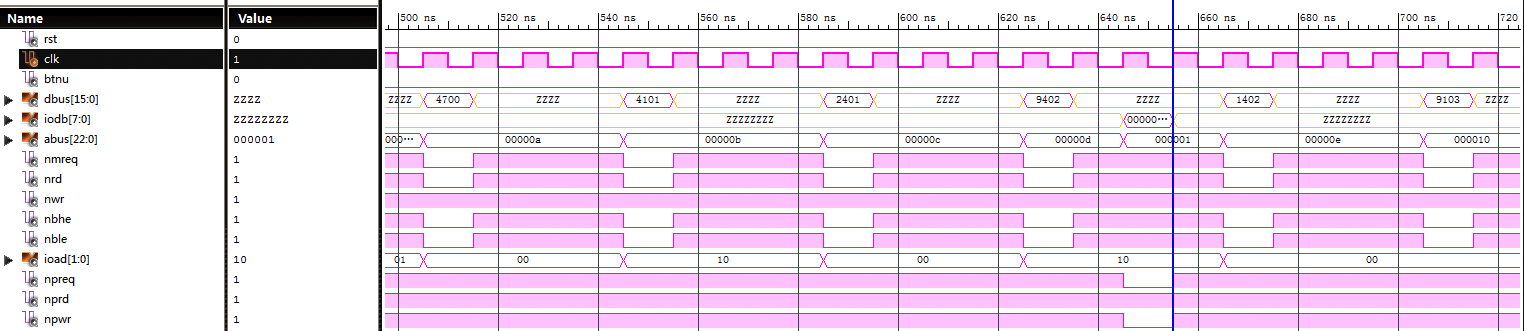
IR <= IRout;

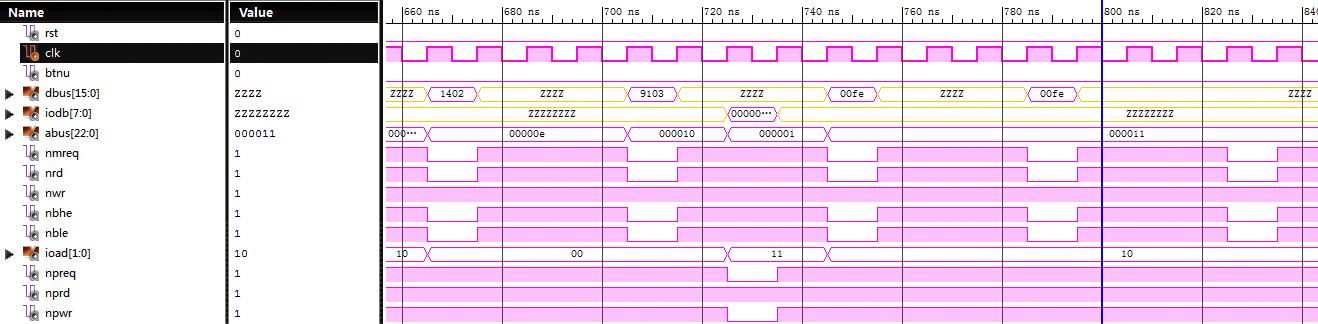
T <= Tout;

end Behavioral;









1. 按键防抖模块btnDebounce

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity btnDebounce is

Port ( CLK : in STD\_LOGIC;

btn : in STD\_LOGIC;

btn\_deb : out STD\_LOGIC);

end btnDebounce;

architecture Behavioral of btnDebounce is

constant CNTR\_MAX : std\_logic\_vector(15 downto 0) := (others => '1');

begin

btn\_debounce: process (CLK, btn)

variable count: INTEGER := 0;

begin

if (rising\_edge(CLK)) then

if btn = '0' then

if (count = CNTR\_MAX) then

count := count;

else

count := count + 1;

end if;

if count = CNTR\_MAX - 1 then

btn\_deb <= '0';

else

btn\_deb <= '1';

end if;

else

count := 0;

end if;

end if;

end process;

end Behavioral;

1. IO转换模块IOConv

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity IOConv is

Port ( IOAD : in STD\_LOGIC\_VECTOR (1 downto 0);

IODB : inout STD\_LOGIC\_VECTOR (7 downto 0);

nPREQ : in STD\_LOGIC;

nPRD : in STD\_LOGIC;

nPWR : in STD\_LOGIC;

sw : in STD\_LOGIC\_VECTOR (7 downto 0);

led : out STD\_LOGIC\_VECTOR (7 downto 0));

end IOConv;

architecture Behavioral of IOConv is

begin

-- This is a IO device simulator adapted on Nexys3.

-- Address -> device:

-- 00 -> sw(7 downto 0)

-- 01 -> sw(0 to 7)

-- 10 -> led(7 downto 0)

-- 11 -> led(0 to 7)

process (IOAD, IODB, nPREQ, nPRD, nPWR, sw)

begin

IODB <= (others => 'Z');

led <= (others => '0');

if nPREQ = '0' then

if nPRD = '0' and nPWR = '1' then

case IOAD is

when "00" => IODB <= sw;

when "01" =>

rev1: for i in 7 downto 0 loop

IODB(i) <= sw(7-i);

end loop rev1;

when others => null;

end case;

elsif nPRD = '1' and nPWR = '0' then

case IOAD is

when "10" => led <= IODB;

when "11" =>

rev2: for i in 7 downto 0 loop

led(i) <= IODB(7-i);

end loop rev2;

when others => null;

end case;

end if;

end if;

end process;

end Behavioral;

1. 七段数码管显示模块segDisplay

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity segDisplay is

Port ( PC : in STD\_LOGIC\_VECTOR (15 downto 0);

IR : in STD\_LOGIC\_VECTOR (15 downto 0);

btnu : in STD\_LOGIC;

T : in STD\_LOGIC\_VECTOR (3 downto 0);

clk : in STD\_LOGIC;

Cy : in STD\_LOGIC;

an : out STD\_LOGIC\_VECTOR (3 downto 0);

seg : out STD\_LOGIC\_VECTOR (7 downto 0));

end segDisplay;

architecture Behavioral of segDisplay is

-- seg display control

signal fpclock : STD\_LOGIC;

signal js : std\_logic\_vector(15 downto 0):= (others => '0');

signal shift : STD\_LOGIC\_VECTOR(3 downto 0) :="0111";

signal digit1, digit2, digit3, digit4 : STD\_LOGIC\_VECTOR (7 downto 0) := (others => '1');

-- convert 4-bit data into Hexadecimal 7-seg display character

function conv\_seg(X : STD\_LOGIC\_VECTOR) return STD\_LOGIC\_VECTOR is

begin

case X is

when "0000" => return "1000000"; -- 0

when "0001" => return "1111001";

when "0010" => return "0100100";

when "0011" => return "0110000";

when "0100" => return "0011001";

when "0101" => return "0010010";

when "0110" => return "0000010";

when "0111" => return "1111000";

when "1000" => return "0000000";

when "1001" => return "0010000"; -- 9

when "1010" => return "0001000"; -- A

when "1011" => return "0000011"; -- b

when "1100" => return "1000110"; -- C

when "1101" => return "0100001"; -- d

when "1110" => return "0000110"; -- E

when "1111" => return "0001110"; -- F

when others => return "1111111";

end case;

end conv\_seg;

begin

-- Seg display

process(clk)

begin

if(rising\_edge(clk)) then

js<=js+'1';

end if;

end process;

fpclock <= js(15);

process(fpclock) -- 7 seg display control

begin

if(fpclock'event and fpclock = '1') then

shift(1)<=shift(0);

shift(2)<=shift(1);

shift(3)<=shift(2);

shift(0)<=shift(3);

an <= shift;

case shift is

when "0111" => seg <= digit1;

when "1011" => seg <= digit2;

when "1101" => seg <= digit3;

when others => seg <= digit4;

end case;

end if;

end process;

digit1 <= (not Cy)&conv\_seg(IR(15 downto 12)) when btnu = '1' else (not T(0))&conv\_seg(PC(15 downto 12));

digit2 <= (not Cy)&conv\_seg(IR(11 downto 8)) when btnu = '1' else (not T(1))&conv\_seg(PC(11 downto 8));

digit3 <= (not Cy)&conv\_seg(IR(7 downto 4)) when btnu = '1' else (not T(2))&conv\_seg(PC(7 downto 4));

digit4 <= (not Cy)&conv\_seg(IR(3 downto 0)) when btnu = '1' else (not T(3))&conv\_seg(PC(3 downto 0));

end Behavioral;

1. PmodCLP显示模块PmodCLP

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity PmodCLP is

Port ( RESET: in std\_logic; --use RESET as reset input

clk: in std\_logic; --100 MHz clock input

IR: in std\_logic\_vector(15 downto 0);

R0: in std\_logic\_vector(7 downto 0);

R1: in std\_logic\_vector(7 downto 0);

R2: in std\_logic\_vector(7 downto 0);

R3: in std\_logic\_vector(7 downto 0);

R4: in std\_logic\_vector(7 downto 0);

R5: in std\_logic\_vector(7 downto 0);

R6: in std\_logic\_vector(7 downto 0);

R7: in std\_logic\_vector(7 downto 0);

--lcd input signals

--signal on connector JC

JC: out std\_logic\_vector(7 downto 0); --output bus, used for data transfer (DB)

-- signal on connector JD

--JD(4)register selection pin (RS)

--JD(5)selects between read/write modes (RW)

--JD(6)enable signal for starting the data read/write (E)

JD: out std\_logic\_vector (6 downto 4)

);

end PmodCLP;

architecture Behavioral of PmodCLP is

------------------------------------------------------------------

-- Local Type Declarations

-----------------------------------------------------------------

-- Symbolic names for all possible states of the state machines.

--LCD control state machine

type mstate is (

stFunctionSet, --Initialization states

stDisplayCtrlSet,

stDisplayClear,

stPowerOn\_Delay, --Delay states

stFunctionSet\_Delay,

stDisplayCtrlSet\_Delay,

stDisplayClear\_Delay,

stInitDne, --Display characters and perform standard operations

stActWr,

stCharDelay --Write delay for operations

);

------------------------------------------------------------------

-- Signal Declarations and Constants for PmodCLP

------------------------------------------------------------------

--These constants are used to initialize the LCD pannel.

--FunctionSet:

--Bit 0 and 1 are arbitrary

--Bit 2: Displays font type(0=5x8, 1=5x11)

--Bit 3: Numbers of display lines (0=1, 1=2)

--Bit 4: Data length (0=4 bit, 1=8 bit)

--Bit 5-7 are set

--DisplayCtrlSet:

--Bit 0: Blinking cursor control (0=off, 1=on)

--Bit 1: Cursor (0=off, 1=on)

--Bit 2: Display (0=off, 1=on)

--Bit 3-7 are set

--DisplayClear:

--Bit 1-7 are set

signal clkCount: std\_logic\_vector (6 downto 0);

signal count: std\_logic\_vector (20 downto 0):= "000000000000000000000"; --21 bit count variable for timing delays

signal delayOK: std\_logic:= '0'; --High when count has reached the right delay time

signal OneUSClk: std\_logic; --Signal is treated as a 1 MHz clock

signal stCur: mstate:= stPowerOn\_Delay; --LCD control state machine

signal stNext: mstate;

signal writeDone: std\_logic:= '0'; --Command set finish

type LCD\_CMDS\_T is array(0 to 37) of std\_logic\_vector(9 downto 0);

signal LCD\_CMDS : LCD\_CMDS\_T := ( 0 => "00"&X"3C", --Function Set

1 => "00"&X"0C", --Display ON, Cursor OFF, Blink OFF

2 => "00"&X"01", --Clear Display

3 => "00"&X"02", --return home

4 => "10"&X"20", --blank

5 => "10"&X"20",

6 => "10"&X"20",

7 => "10"&X"20",

8 => "10"&X"20",

9 => "10"&X"20",

10 => "10"&X"20",

11 => "10"&X"20",

12 => "10"&X"20",

13 => "10"&X"20",

14 => "10"&X"20",

15 => "10"&X"20",

16 => "10"&X"20",

17 => "10"&X"20",

18 => "10"&X"20",

19 => "10"&X"20",

20 => "00"&X"C0", --Set DDRAM location to 40H (second row)

21 => "10"&X"20",

22 => "10"&X"20",

23 => "10"&X"20",

24 => "10"&X"20",

25 => "10"&X"20",

26 => "10"&X"20",

27 => "10"&X"20",

28 => "10"&X"20",

29 => "10"&X"20",

30 => "10"&X"20",

31 => "10"&X"20",

32 => "10"&X"20",

33 => "10"&X"20",

34 => "10"&X"20",

35 => "10"&X"20",

36 => "10"&X"20",

37 => "00"&X"02"); --return home

signal lcd\_cmd\_ptr : integer range 0 to LCD\_CMDS'HIGH + 1 := 0;

-- convert 4-bit data into Hexadecimal ascii display character

function conv\_ascii(X : STD\_LOGIC\_VECTOR) return STD\_LOGIC\_VECTOR is

begin

case X is

when "0000" => return X"30"; -- 0

when "0001" => return X"31";

when "0010" => return X"32";

when "0011" => return X"33";

when "0100" => return X"34";

when "0101" => return X"35";

when "0110" => return X"36";

when "0111" => return X"37";

when "1000" => return X"38";

when "1001" => return X"39"; -- 9

when "1010" => return X"41"; -- A

when "1011" => return X"42"; -- b

when "1100" => return X"43"; -- C

when "1101" => return X"44"; -- d

when "1110" => return X"45"; -- E

when "1111" => return X"46"; -- F

when others => return X"58"; -- X

end case;

end conv\_ascii;

begin

--This process counts to 100, and then resets. It is used to divide the clock signal.

--This makes oneUSClock peak aprox. once every 1microsecond

process (CLK)

begin

if (CLK = '1' and CLK'event) then

if(clkCount = "0000001") then -- 修改clkCount可以调节PmodCLP屏幕刷新速度

clkCount <= "0000000";

oneUSClk <= not oneUSClk;

else

clkCount <= clkCount + 1;

end if;

end if;

end process;

--This process increments the count variable unless delayOK = 1.

process (CLK, delayOK)

begin

if (CLK = '1' and CLK'event) then

if delayOK = '1' then

count <= "000000000000000000000";

else

count <= count + 1;

end if;

end if;

end process;

--Determines when count has gotten to the right number, depending on the state.

delayOK <= '1' when ((stCur = stPowerOn\_Delay and count = "111101000010010000000") or --2000000 -> 20 ms

(stCur = stFunctionSet\_Delay and count = "000000000111110100000") or --4000 -> 40 us

(stCur = stDisplayCtrlSet\_Delay and count = "000000000111110100000") or --4000 -> 40 us

(stCur = stDisplayClear\_Delay and count = "000100111000100000000") or --160000 -> 1.6 ms

(stCur = stCharDelay and count = "000111111011110100000")) --260000 -> 2.6 ms - Max Delay for character writes and shifts

else '0';

--writeDone goes high when all commands have been run

writeDone <= '1' when (lcd\_cmd\_ptr = LCD\_CMDS'HIGH)

else '0';

--Increments the pointer so the statemachine goes through the commands

process (lcd\_cmd\_ptr, oneUSClk)

begin

if (oneUSClk = '1' and oneUSClk'event) then

if ((stNext = stInitDne or stNext = stDisplayCtrlSet or stNext = stDisplayClear) and writeDone = '0') then

lcd\_cmd\_ptr <= lcd\_cmd\_ptr + 1;

elsif ((stNext = stInitDne or stNext = stDisplayCtrlSet or stNext = stDisplayClear) and writeDone = '1') then

lcd\_cmd\_ptr <= 4;

elsif stCur = stPowerOn\_Delay or stNext = stPowerOn\_Delay then

lcd\_cmd\_ptr <= 0;

else

lcd\_cmd\_ptr <= lcd\_cmd\_ptr;

end if;

end if;

end process;

--This process runs the LCD state machine

process (oneUSClk, RESET)

begin

if oneUSClk = '1' and oneUSClk'Event then

if RESET = '1' then

stCur <= stPowerOn\_Delay;

else

stCur <= stNext;

end if;

end if;

end process;

--This process generates the sequence of outputs needed to initialize and write to the LCD screen

process (stCur, delayOK, writeDone, lcd\_cmd\_ptr)

begin

case stCur is

--Delays the state machine for 20ms which is needed for proper startup.

when stPowerOn\_Delay =>

if delayOK = '1' then

stNext <= stFunctionSet;

else

stNext <= stPowerOn\_Delay;

end if;

--This issues the function set to the LCD as follows

--8 bit data length, 1 lines, font is 5x8.

when stFunctionSet =>

stNext <= stFunctionSet\_Delay;

--Gives the proper delay of 37us between the function set and

--the display control set.

when stFunctionSet\_Delay =>

if delayOK = '1' then

stNext <= stDisplayCtrlSet;

else

stNext <= stFunctionSet\_Delay;

end if;

--Issuse the display control set as follows

--Display ON, Cursor OFF, Blinking Cursor OFF.

when stDisplayCtrlSet =>

stNext <= stDisplayCtrlSet\_Delay;

--Gives the proper delay of 37us between the display control set

--and the Display Clear command.

when stDisplayCtrlSet\_Delay =>

if delayOK = '1' then

stNext <= stDisplayClear;

else

stNext <= stDisplayCtrlSet\_Delay;

end if;

--Issues the display clear command.

when stDisplayClear =>

stNext <= stDisplayClear\_Delay;

--Gives the proper delay of 1.52ms between the clear command

--and the state where you are clear to do normal operations.

when stDisplayClear\_Delay =>

if delayOK = '1' then

stNext <= stInitDne;

else

stNext <= stDisplayClear\_Delay;

end if;

--State for normal operations for displaying characters, changing the

--Cursor position etc.

when stInitDne =>

stNext <= stActWr;

when stActWr =>

stNext <= stCharDelay;

--Provides a max delay between instructions.

when stCharDelay =>

if delayOK = '1' then

stNext <= stInitDne;

else

stNext <= stCharDelay;

end if;

end case;

end process;

--Generate display content according to IR and R0~R7

process (IR, R0, R1, R2, R3, R4, R5, R6, R7)

begin

LCD\_CMDS(4) <= "10"&X"20"; -- blank

LCD\_CMDS(5) <= "10"&X"20";

LCD\_CMDS(6) <= "10"&X"20";

LCD\_CMDS(7) <= "10"&X"20";

LCD\_CMDS(8) <= "10"&X"20";

LCD\_CMDS(9) <= "10"&X"20";

LCD\_CMDS(10) <= "10"&X"20";

LCD\_CMDS(11) <= "10"&X"20";

LCD\_CMDS(12) <= "10"&X"20";

LCD\_CMDS(13) <= "10"&X"20";

LCD\_CMDS(14) <= "10"&X"20";

LCD\_CMDS(15) <= "10"&X"20";

LCD\_CMDS(16) <= "10"&X"20";

LCD\_CMDS(17) <= "10"&X"20";

LCD\_CMDS(18) <= "10"&X"20";

LCD\_CMDS(19) <= "10"&X"20";

LCD\_CMDS(21) <= "10"&conv\_ascii(R0(7 downto 4)); -- R0

LCD\_CMDS(22) <= "10"&conv\_ascii(R0(3 downto 0));

LCD\_CMDS(23) <= "10"&conv\_ascii(R1(7 downto 4)); -- R1

LCD\_CMDS(24) <= "10"&conv\_ascii(R1(3 downto 0));

LCD\_CMDS(25) <= "10"&conv\_ascii(R2(7 downto 4)); -- R2

LCD\_CMDS(26) <= "10"&conv\_ascii(R2(3 downto 0));

LCD\_CMDS(27) <= "10"&conv\_ascii(R3(7 downto 4)); -- R3

LCD\_CMDS(28) <= "10"&conv\_ascii(R3(3 downto 0));

LCD\_CMDS(29) <= "10"&conv\_ascii(R4(7 downto 4)); -- R4

LCD\_CMDS(30) <= "10"&conv\_ascii(R4(3 downto 0));

LCD\_CMDS(31) <= "10"&conv\_ascii(R5(7 downto 4)); -- R5

LCD\_CMDS(32) <= "10"&conv\_ascii(R5(3 downto 0));

LCD\_CMDS(33) <= "10"&conv\_ascii(R6(7 downto 4)); -- R6

LCD\_CMDS(34) <= "10"&conv\_ascii(R6(3 downto 0));

LCD\_CMDS(35) <= "10"&conv\_ascii(R7(7 downto 4)); -- R7

LCD\_CMDS(36) <= "10"&conv\_ascii(R7(3 downto 0));

case IR(15 downto 11) is

when "00000" => -- JMP [R7//0xff]

LCD\_CMDS(4)(7 downto 0) <= X"4a";

LCD\_CMDS(5)(7 downto 0) <= X"4d";

LCD\_CMDS(6)(7 downto 0) <= X"50";

LCD\_CMDS(8)(7 downto 0) <= X"5b"; -- [

LCD\_CMDS(9)(7 downto 0) <= conv\_ascii(R7(7 downto 4)); -- R7

LCD\_CMDS(10)(7 downto 0) <= conv\_ascii(R7(3 downto 0)); -- R7

LCD\_CMDS(11)(7 downto 0) <= X"2f"; -- /

LCD\_CMDS(12)(7 downto 0) <= X"2f"; -- /

LCD\_CMDS(13)(7 downto 0) <= X"30"; -- 0

LCD\_CMDS(14)(7 downto 0) <= X"78"; -- x

LCD\_CMDS(15)(7 downto 0) <= conv\_ascii(IR(7 downto 4));

LCD\_CMDS(16)(7 downto 0) <= conv\_ascii(IR(3 downto 0));

LCD\_CMDS(17)(7 downto 0) <= X"5d";

when "00010" => -- JZ Rx, 0xffff

LCD\_CMDS(4)(7 downto 0) <= X"4a";

LCD\_CMDS(5)(7 downto 0) <= X"5a";

LCD\_CMDS(7)(7 downto 0) <= X"52";

LCD\_CMDS(8)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(9)(7 downto 0) <= X"2c"; -- ,

LCD\_CMDS(11)(7 downto 0) <= X"30"; -- 0

LCD\_CMDS(12)(7 downto 0) <= X"78"; -- x

LCD\_CMDS(13)(7 downto 0) <= conv\_ascii(IR(7)&IR(7)&IR(7)&IR(7)); -- sign

LCD\_CMDS(14)(7 downto 0) <= conv\_ascii(IR(7)&IR(7)&IR(7)&IR(7)); -- sign

LCD\_CMDS(15)(7 downto 0) <= conv\_ascii(IR(7 downto 4));

LCD\_CMDS(16)(7 downto 0) <= conv\_ascii(IR(3 downto 0));

when "00100" => -- SBB Rx, Rx

LCD\_CMDS(4)(7 downto 0) <= X"53";

LCD\_CMDS(5)(7 downto 0) <= X"42";

LCD\_CMDS(6)(7 downto 0) <= X"42";

LCD\_CMDS(8)(7 downto 0) <= X"52";

LCD\_CMDS(9)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(10)(7 downto 0) <= X"2c";

LCD\_CMDS(12)(7 downto 0) <= X"52";

LCD\_CMDS(13)(7 downto 0) <= "00110"&IR(2 downto 0);

when "00110" => -- ADC Rx, Rx

LCD\_CMDS(4)(7 downto 0) <= X"41";

LCD\_CMDS(5)(7 downto 0) <= X"44";

LCD\_CMDS(6)(7 downto 0) <= X"43";

LCD\_CMDS(8)(7 downto 0) <= X"52";

LCD\_CMDS(9)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(10)(7 downto 0) <= X"2c";

LCD\_CMDS(12)(7 downto 0) <= X"52";

LCD\_CMDS(13)(7 downto 0) <= "00110"&IR(2 downto 0);

when "01000" => -- MVI Rx, 0xff

LCD\_CMDS(4)(7 downto 0) <= X"4d";

LCD\_CMDS(5)(7 downto 0) <= X"56";

LCD\_CMDS(6)(7 downto 0) <= X"49";

LCD\_CMDS(8)(7 downto 0) <= X"52";

LCD\_CMDS(9)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(10)(7 downto 0) <= X"2c";

LCD\_CMDS(12)(7 downto 0) <= X"30";

LCD\_CMDS(13)(7 downto 0) <= X"78";

LCD\_CMDS(14)(7 downto 0) <= conv\_ascii(IR(7 downto 4));

LCD\_CMDS(15)(7 downto 0) <= conv\_ascii(IR(3 downto 0));

when "01010" => -- MOV Rx, Rx

LCD\_CMDS(4)(7 downto 0) <= X"4d";

LCD\_CMDS(5)(7 downto 0) <= X"4f";

LCD\_CMDS(6)(7 downto 0) <= X"56";

LCD\_CMDS(8)(7 downto 0) <= X"52";

LCD\_CMDS(9)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(10)(7 downto 0) <= X"2c";

LCD\_CMDS(12)(7 downto 0) <= X"52";

LCD\_CMDS(13)(7 downto 0) <= "00110"&IR(2 downto 0);

when "01100" => -- STA Rx, [0xff]

LCD\_CMDS(4)(7 downto 0) <= X"53";

LCD\_CMDS(5)(7 downto 0) <= X"54";

LCD\_CMDS(6)(7 downto 0) <= X"41";

LCD\_CMDS(8)(7 downto 0) <= X"52";

LCD\_CMDS(9)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(10)(7 downto 0) <= X"2c";

LCD\_CMDS(12)(7 downto 0) <= X"5b";

LCD\_CMDS(13)(7 downto 0) <= X"30";

LCD\_CMDS(14)(7 downto 0) <= X"78";

LCD\_CMDS(15)(7 downto 0) <= conv\_ascii(IR(7 downto 4));

LCD\_CMDS(16)(7 downto 0) <= conv\_ascii(IR(3 downto 0));

LCD\_CMDS(17)(7 downto 0) <= X"5d";

when "01110" => -- LDA Rx, [0xff]

LCD\_CMDS(4)(7 downto 0) <= X"4c";

LCD\_CMDS(5)(7 downto 0) <= X"44";

LCD\_CMDS(6)(7 downto 0) <= X"41";

LCD\_CMDS(8)(7 downto 0) <= X"52";

LCD\_CMDS(9)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(10)(7 downto 0) <= X"2c";

LCD\_CMDS(12)(7 downto 0) <= X"5b";

LCD\_CMDS(13)(7 downto 0) <= X"30";

LCD\_CMDS(14)(7 downto 0) <= X"78";

LCD\_CMDS(15)(7 downto 0) <= conv\_ascii(IR(7 downto 4));

LCD\_CMDS(16)(7 downto 0) <= conv\_ascii(IR(3 downto 0));

LCD\_CMDS(17)(7 downto 0) <= X"5d";

when "10000" => -- IN Rx, [00]

LCD\_CMDS(4)(7 downto 0) <= X"49";

LCD\_CMDS(5)(7 downto 0) <= X"4e";

LCD\_CMDS(7)(7 downto 0) <= X"52";

LCD\_CMDS(8)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(9)(7 downto 0) <= X"2c";

LCD\_CMDS(11)(7 downto 0) <= X"5b";

LCD\_CMDS(12)(7 downto 0) <= conv\_ascii("000"&IR(1));

LCD\_CMDS(13)(7 downto 0) <= conv\_ascii("000"&IR(0));

LCD\_CMDS(14)(7 downto 0) <= X"5d";

when "10010" => -- OUT Rx, [00]

LCD\_CMDS(4)(7 downto 0) <= X"4f";

LCD\_CMDS(5)(7 downto 0) <= X"55";

LCD\_CMDS(6)(7 downto 0) <= X"54";

LCD\_CMDS(8)(7 downto 0) <= X"52";

LCD\_CMDS(9)(7 downto 0) <= "00110"&IR(10 downto 8);

LCD\_CMDS(10)(7 downto 0) <= X"2c";

LCD\_CMDS(12)(7 downto 0) <= X"5b";

LCD\_CMDS(13)(7 downto 0) <= conv\_ascii("000"&IR(1));

LCD\_CMDS(14)(7 downto 0) <= conv\_ascii("000"&IR(0));

LCD\_CMDS(15)(7 downto 0) <= X"5d";

when "10100" => -- CLRC

LCD\_CMDS(4)(7 downto 0) <= X"43";

LCD\_CMDS(5)(7 downto 0) <= X"4c";

LCD\_CMDS(6)(7 downto 0) <= X"52";

LCD\_CMDS(7)(7 downto 0) <= X"43";

when "10110" => -- SETC

LCD\_CMDS(4)(7 downto 0) <= X"53";

LCD\_CMDS(5)(7 downto 0) <= X"45";

LCD\_CMDS(6)(7 downto 0) <= X"54";

LCD\_CMDS(7)(7 downto 0) <= X"43";

when others => -- Undefined... >\_<

LCD\_CMDS(4)(7 downto 0) <= X"55";

LCD\_CMDS(5)(7 downto 0) <= X"6e";

LCD\_CMDS(6)(7 downto 0) <= X"64";

LCD\_CMDS(7)(7 downto 0) <= X"65";

LCD\_CMDS(8)(7 downto 0) <= X"66";

LCD\_CMDS(9)(7 downto 0) <= X"69";

LCD\_CMDS(10)(7 downto 0) <= X"6e";

LCD\_CMDS(11)(7 downto 0) <= X"65";

LCD\_CMDS(12)(7 downto 0) <= X"64";

LCD\_CMDS(13)(7 downto 0) <= X"2e";

LCD\_CMDS(14)(7 downto 0) <= X"2e";

LCD\_CMDS(16)(7 downto 0) <= X"3e";

LCD\_CMDS(17)(7 downto 0) <= X"5f";

LCD\_CMDS(18)(7 downto 0) <= X"3c";

end case;

end process;

JD(4) <= LCD\_CMDS(lcd\_cmd\_ptr)(9);

JD(5) <= LCD\_CMDS(lcd\_cmd\_ptr)(8);

JC <= LCD\_CMDS(lcd\_cmd\_ptr)(7 downto 0);

JD(6) <= '1' when stCur = stFunctionSet or stCur = stDisplayCtrlSet or stCur = stDisplayClear or stCur = stActWr

else '0';

end Behavioral;

1. 总模块N3Adapter

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity N3Adapter is

Port ( CLK: in STD\_LOGIC;

sw : in STD\_LOGIC\_VECTOR (7 downto 0);

led : out STD\_LOGIC\_VECTOR (7 downto 0);

btnr : in STD\_LOGIC;

btns : in STD\_LOGIC;

btnu : in STD\_LOGIC;

an : out STD\_LOGIC\_VECTOR (3 downto 0);

seg : out STD\_LOGIC\_VECTOR (7 downto 0);

--signal on connector JC

JC: out std\_logic\_vector(7 downto 0); --output bus, used for data transfer (DB)

-- signal on connector JD

--JD(4)register selection pin (RS)

--JD(5)selects between read/write modes (RW)

--JD(6)enable signal for starting the data read/write (E)

JD: out std\_logic\_vector (6 downto 4);

MemOE : out STD\_LOGIC;

MemWR : out STD\_LOGIC;

MemAdv : out STD\_LOGIC;

MemWait : out STD\_LOGIC;

MemClk : out STD\_LOGIC;

RamCS : out STD\_LOGIC;

RamCRE : out STD\_LOGIC;

RamUB : out STD\_LOGIC;

RamLB : out STD\_LOGIC;

MemAdr : out STD\_LOGIC\_VECTOR (26 downto 1);

MemDB : inout STD\_LOGIC\_VECTOR (15 downto 0));

end N3Adapter;

architecture Behavioral of N3Adapter is

------------------------------------------------------------------

-- Component Declarations

------------------------------------------------------------------

component CPU

Port ( RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

DBUS : inout STD\_LOGIC\_VECTOR (15 downto 0);

IODB : inout STD\_LOGIC\_VECTOR (7 downto 0);

ABUS : out STD\_LOGIC\_VECTOR (15 downto 0);

IOAD : out STD\_LOGIC\_VECTOR (1 downto 0);

nMREQ : out STD\_LOGIC;

nPREQ : out STD\_LOGIC;

nRD : out STD\_LOGIC;

nWR : out STD\_LOGIC;

nBHE : out STD\_LOGIC;

nBLE : out STD\_LOGIC;

nPRD : out STD\_LOGIC;

nPWR : out STD\_LOGIC;

Cout : out std\_logic;

IR : out STD\_LOGIC\_VECTOR (15 downto 0);

PC : out STD\_LOGIC\_VECTOR (15 downto 0);

R0 : out STD\_LOGIC\_VECTOR (7 downto 0);

R1 : out STD\_LOGIC\_VECTOR (7 downto 0);

R2 : out STD\_LOGIC\_VECTOR (7 downto 0);

R3 : out STD\_LOGIC\_VECTOR (7 downto 0);

R4 : out STD\_LOGIC\_VECTOR (7 downto 0);

R5 : out STD\_LOGIC\_VECTOR (7 downto 0);

R6 : out STD\_LOGIC\_VECTOR (7 downto 0);

R7 : out STD\_LOGIC\_VECTOR (7 downto 0);

T : out STD\_LOGIC\_VECTOR (3 downto 0));

end component;

component IOConv

Port ( IOAD : in STD\_LOGIC\_VECTOR (1 downto 0);

IODB : inout STD\_LOGIC\_VECTOR (7 downto 0);

nPREQ : in STD\_LOGIC;

nPRD : in STD\_LOGIC;

nPWR : in STD\_LOGIC;

sw : in STD\_LOGIC\_VECTOR (7 downto 0);

led : out STD\_LOGIC\_VECTOR (7 downto 0));

end component;

component btnDebounce

Port ( CLK : in std\_logic;

btn : in std\_logic;

btn\_deb : out std\_logic);

end component;

component PmodCLP

Port ( RESET: in std\_logic; --use BTNS as reset input

CLK: in std\_logic; --100 MHz clock input

IR: in std\_logic\_vector(15 downto 0);

R0: in std\_logic\_vector(7 downto 0);

R1: in std\_logic\_vector(7 downto 0);

R2: in std\_logic\_vector(7 downto 0);

R3: in std\_logic\_vector(7 downto 0);

R4: in std\_logic\_vector(7 downto 0);

R5: in std\_logic\_vector(7 downto 0);

R6: in std\_logic\_vector(7 downto 0);

R7: in std\_logic\_vector(7 downto 0);

--lcd input signals

--signal on connector JC

JC: out std\_logic\_vector(7 downto 0); --output bus, used for data transfer (DB)

-- signal on connector JD

--JD(4)register selection pin (RS)

--JD(5)selects between read/write modes (RW)

--JD(6)enable signal for starting the data read/write (E)

JD: out std\_logic\_vector (6 downto 4)

);

end component;

component segDisplay

Port ( PC : in STD\_LOGIC\_VECTOR (15 downto 0);

IR : in STD\_LOGIC\_VECTOR (15 downto 0);

btnu : in STD\_LOGIC;

T : in STD\_LOGIC\_VECTOR (3 downto 0);

clk : in STD\_LOGIC;

Cy : in STD\_LOGIC;

an : out STD\_LOGIC\_VECTOR (3 downto 0);

seg : out STD\_LOGIC\_VECTOR (7 downto 0));

end component;

------------------------------------------------------------------

-- Local Type Declarations

-----------------------------------------------------------------

-- For CPU

signal T : std\_logic\_vector(3 downto 0);

signal IR : STD\_LOGIC\_VECTOR (15 downto 0);

signal PC : STD\_LOGIC\_VECTOR (15 downto 0);

signal Cout : std\_logic;

signal R0 : STD\_LOGIC\_VECTOR (7 downto 0);

signal R1 : STD\_LOGIC\_VECTOR (7 downto 0);

signal R2 : STD\_LOGIC\_VECTOR (7 downto 0);

signal R3 : STD\_LOGIC\_VECTOR (7 downto 0);

signal R4 : STD\_LOGIC\_VECTOR (7 downto 0);

signal R5 : STD\_LOGIC\_VECTOR (7 downto 0);

signal R6 : STD\_LOGIC\_VECTOR (7 downto 0);

signal R7 : STD\_LOGIC\_VECTOR (7 downto 0);

signal ABUS : STD\_LOGIC\_VECTOR (15 downto 0);

signal nRD : STD\_LOGIC;

signal nWR : STD\_LOGIC;

-- This signals are for IO controller.

signal IODB : std\_logic\_vector(7 downto 0);

signal IOAD : std\_logic\_vector(1 downto 0);

signal nPREQ, nPRD, nPWR : std\_logic;

-- Debounce btnr

signal btnr\_deb : std\_logic;

begin

comBtnrDeb: btnDebounce port map(

CLK => CLK,

btn => btnr,

btn\_deb => btnr\_deb);

comPmodCLP: PmodCLP port map(

RESET => btns,

CLK => CLK,

IR => IR,

R0 => R0,

R1 => R1,

R2 => R2,

R3 => R3,

R4 => R4,

R5 => R5,

R6 => R6,

R7 => R7,

JC => JC,

JD => JD);

comCPU: CPU port map(

RST => btns,

CLK => btnr\_deb,

DBUS => MemDB,

IODB => IODB,

ABUS => ABUS,

IOAD => IOAD,

nMREQ => RamCS,

nPREQ => nPREQ,

nRD => MemOE,

nWR => MemWR,

nBHE => RamUB,

nBLE => RamLB,

nPRD => nPRD,

nPWR => nPWR,

Cout => Cout,

IR => IR,

PC => PC,

R0 => R0,

R1 => R1,

R2 => R2,

R3 => R3,

R4 => R4,

R5 => R5,

R6 => R6,

R7 => R7,

T => T);

comIOConv: IOConv port map(

IOAD => IOAD,

IODB => IODB,

nPREQ => nPREQ,

nPRD => nPRD,

nPWR => nPWR,

sw => sw,

led => led);

comsegDisplay: segDisplay port map(

T => T,

PC => PC,

IR => IR,

btnu => btnu,

Cy => Cout,

clk => clk,

an => an,

seg => seg);

MemAdv <= '0';

MemWait <= '0';

MemClk <= '0';

RamCRE <= '0';

MemAdr(26 downto 16) <= (others => '0');

MemAdr(15 downto 1) <= ABUS(15 downto 1);

end Behavioral;

# 问题与解决方案

下面介绍一下在设计、调试、波形、下载过程中遇到的问题及解决方法。

1. 一个节拍内可能要做多件事，比如T1节拍，既要准备好A、B、Addr的值，又要根据A、B、Cy的值计算出结果，这两件事有先后顺序，而如果只由T1一个信号控制的话，先后顺序不好处理。

解决办法：将CLK时钟信号引入，一个节拍就可以分成前半拍（CLK=’1’）和后半拍（CLK=’0’）。可以在节拍的上升沿做前一件事，在CLK的下降沿做另一件事，顺序就很清楚了。

1. 取指模块取到的指令不对；访存模块在写内存时改变了不应改变的（前一次访问的）数据。

解决办法：取到的指令不对，可以把IR的赋值从边沿触发改为电平触发，在T0 and CLK高电平时让IR跟随IRdata，这样就不会出现要取的数据还没出现在数据线上就被取走的情况。也可以提前送地址，后取数据。写数据时将数据写乱也可能是这个原因，写信号的给出先于地址的给出，导致先写到了上一次地址总线所指的地方，又写到了正确的地方。

1. 系统仿真时看到波形中数据线上出现一堆红色的“X”。

解决办法：问题的原因有很多，如读写的时序有问题，但最有可能也最容易解决的是，在读数据时没有将数据总线赋高阻。数据总线是三态总线，使用一个方向时，另一个方向必须赋高阻。如果输出不赋高阻，还要读取数据，那么数据总线上的数据就是0、1混淆的，也就是X。

1. 仿真一切都对，但是下载到板子上PC就是不加一。

解决办法：这是一个非常常见也非常棘手的问题，据我了解，几乎所有人，包括我，最后都采用了同一个解决办法，就是把PC+1从取指模块改到回写模块。我取消了PCupdate信号，让回写模块中的PCnew来表示加一后的PC或跳转到的PC值，然后让取指模块在T0上升沿把PCnew更新到PC。这样就避开了取指模块既修改PC又送PC访存的麻烦，改完以后，程序运行马上就变得一切正常了。

1. 下载到板子上以后，发现读到的指令或数据有时是正确的，有时又是0x0006或0x6666。

解决办法：这是使用N3开发板的同学很容易遇到的问题，究其原因，还是访存的时序没有控制好，数据有时是对的，有时是错的，说明逻辑没错，但实现得不好，不稳定。建议把访存模块换一种方法重写一遍，对于解决疑难bug有奇效。

# 总结

本文设计并实现了一个基于Digilent® Nexys 3™开发板的单进程、精简指令集CPU。该CPU采用四个节拍控制指令周期完成一条指令，结构清晰，实现起来并不复杂，但要真正使其正常工作，还是需要大量的调试工作。在这次开发过程中，我经历了巨大的困难，连续几天熬到凌晨四五点，只为了改好一个个的bug。奋斗的过程是痛苦的，但成功后的喜悦是巨大的。当我成功在开发板上运行了自己设计的程序，并将每条指令和寄存器数据显示在PmodCLP上的时候，那种欣慰告诉我，这些付出是值得的。我不仅学到了硬件设计的入门知识，还极大地加深了对于计算机底层设计和运行机制的理解，这些收获，如果不亲自做一个CPU的话，是不可能有的。最后我想说，虽然这个CPU做完了，看起来也运行得很好，但是我知道，他一定还存在着许多缺陷，存在着很多我没有发现的问题。越学习，就越明白自己的浅薄，接下来，我会以更加虚心的态度，不仅在计算机这个领域中，也要在各种知识和人文素养上，做一个不那么浅薄的人。这篇报告在上交老师的同时，也发在了我的博客中，文中有诸多不当之处，还请不吝赐教。

