**《硬件综合实践》报告书**

**简易处理器的设计与实现**

班 级：

学 号：

姓 名：

系 别：

指导教师：

简易处理器的设计与实现

# 设计要求

设计一个简易的处理器，该处理机能够处理两个8bit数据的CPU，该运算器可以实现数据的加法、减法运算、与运算、或运算、取非、与非运算、与或运算和异或运算8种算术与逻辑运算，对运算结果进行处理后，进行输出一个8bit结果。

在寻址方式上，采用了存储器寻址的方式得到相应的数据。

设计指令集，指令长度为8位，高3位作为操作码，控制，低4位作为通用寄存器的地址选择,剩下一位舍弃,指令集中存8条指令。指令和数据分别存储在指令存储器和通用寄存器中。

存储器和通用寄存器采用RAM实现指令存储器和通用寄存器,分别存储指令和数据。

根据CPU的执行流程进行顶层设计，将各个部件按照所需要求和功能进行具体的设计与实现。综合各个部分,设计CPU的顶层框图,将各部分连接起来,形成完整的简易CPU结构。

采用自顶向下的设计以及自底向上的实现方式，设计实现该CPU。

# 系统总体设计

1. CPU 顶层设计

该简易CPU目的是能在时钟脉冲的控制下，通过取指令的控制获取一条指令，将该指令进行存储，通过取出的指令，从外部输入获取相应的数据，并根据指令的操作码进行相应的操作，对所取数据进行相应处理，将运算结果输出保存到寄存器中，将运算结果输出。

CPU 设计的顶层图如下:



**图 2-1 CPU设计顶层图**

1. 指令设计

在该 CPU 的设计中，共 8 种算术运算和逻辑运算，但为了方便以后的扩展，因此，操作码字段 OP 用4位二进制数进行控制；寻址方式为：寄存器寻址，通用寄存器预设计为8个，因此，寄存器的选择需要4位二进制数进行控制。设计指令宽度为8位，高3位作为操作码，低四位作为寄存器地址寻址，剩余一位废弃。

指令格式如下：

|  |  |
| --- | --- |
| **操作码 OP** | **寄存器地址** |
| 3位 | 4位 |

1. 各部件设计

四节拍脉冲发生器：因为 CPU 每个部件执行的动作的时间都是有严格要求的，因此需要一个四节拍脉冲发生器，对各种操作信号实施时间进行控制。

运算器 ALU：通过指令的操作码S[7..5]，对传入的 A 和 B 数据进行相应的操作，并对 A 和 B 运算的结果进行输出，将 8 位的二进制结果输出到数据缓存寄存器DR中。

指令存储器：只读的存储器，内置一些指令。通过 PC 传来的地址，在控制信号到来时，将存储的指令 IR\_data 传出到指令寄存器IR 中。

程序计数器 PC：为了取出每一条指令，需要通过一个程序计数器作为地址取出指令。PC 将值 AD 作为地址传递到指令存储器中，在控制信号到来的时候，程序计数器的值会+1。

指令寄存器 IR：在控制信号 T2 到来时，接收处理指令存储器传来的指令，用于对后面部件的控制。

通用存储器（寄存器）：用于存储一些数据。使用ROM存储器作为寄存器使用，共设计四个寄存器R0-R3，首先通过外部 data 数据对寄存器进行初始化，通过地址 S[1..0]和S[3..2]选出两个数据，作为 A 和 B 传给运算器。

数据缓冲寄存器 DR：在控制信号的控制下，接收运算器的运算结果，对其进行缓冲，对结果输出。使用D触发器实现。

# 系统详细设计

通过顶层的设计后，对底层的各个部件进行实现，具体的实现如下：

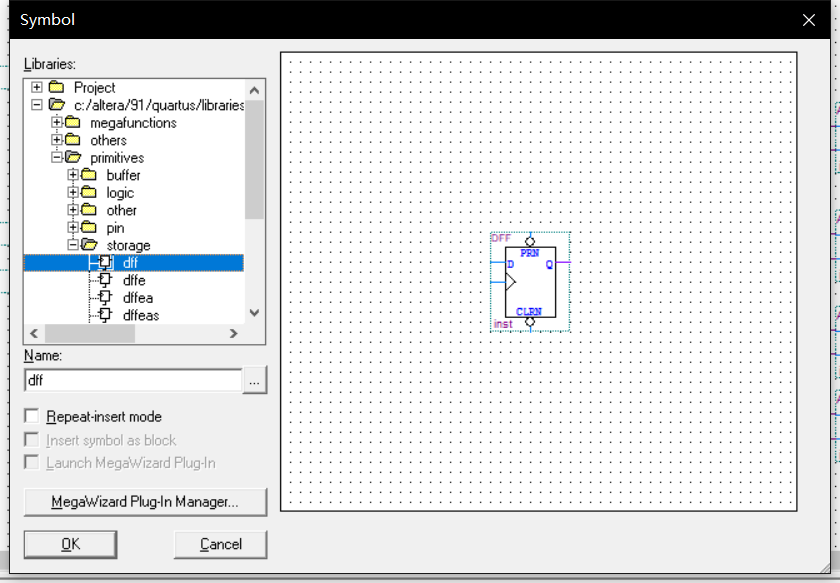
1. 四节拍脉冲发生器

四节拍脉冲发生器可产生 4 个等间隔的时序节拍信号 T1~T4，CLK 为时钟信号，当START高电平变为低电平时，四节拍脉冲发生器开始工作，T1~T4 将在 CLK 的输入脉冲作用下，周期性地轮流输出正脉冲；

当STOP高电平变为低电平时，四节拍脉冲发生器暂停工作；当 RESET 为高电平时，T1~T4可以正常工作；当 RESET由高电平变为低电平后，T1~T4置为低电平。

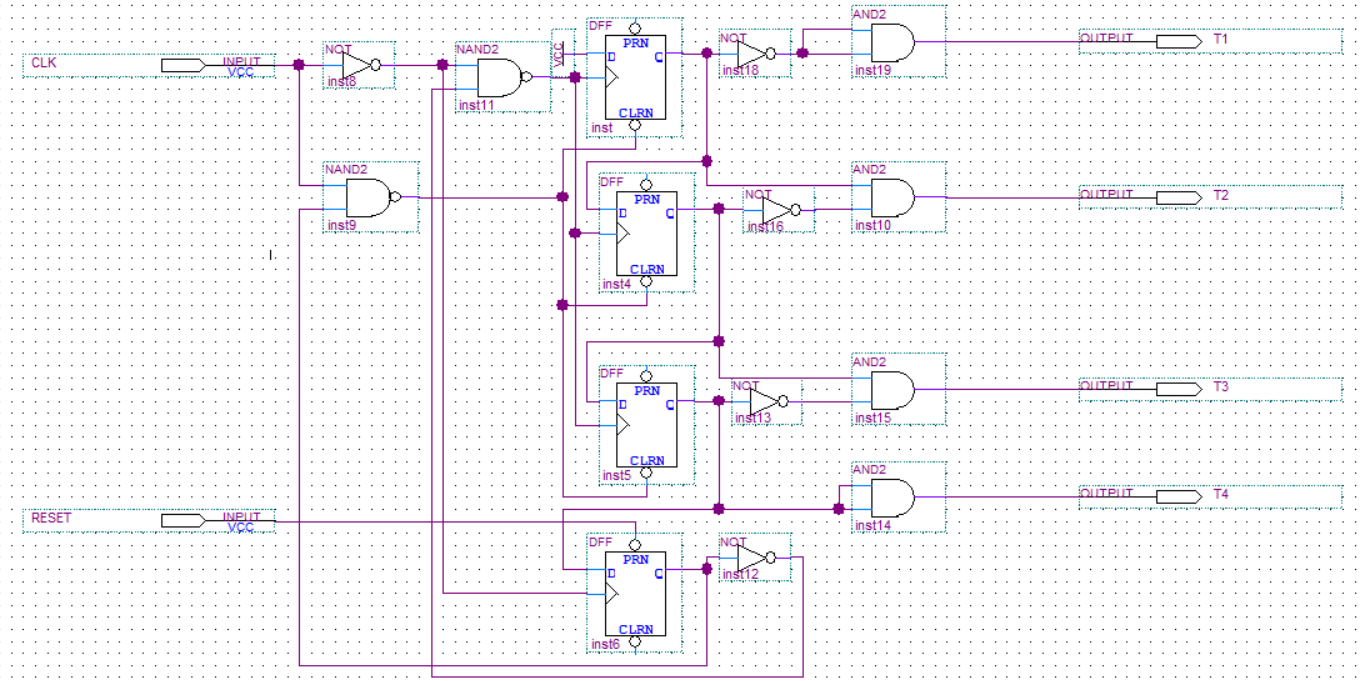
通过产生节拍信号，用于对各个部件的发生时间进行控制。

创建D触发器:



**图3-1-1 创建D触发器**

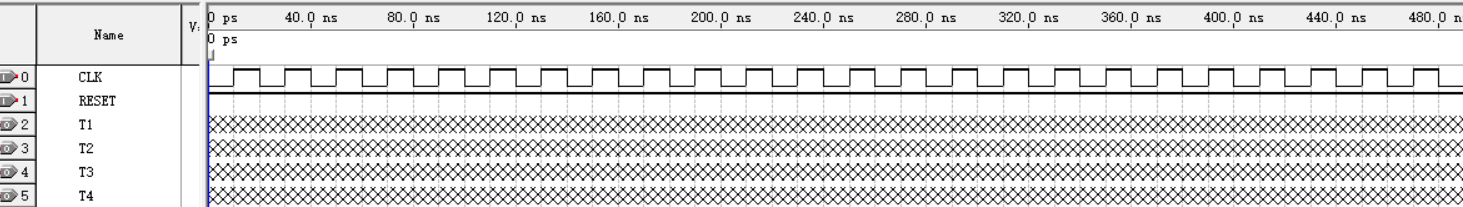
四节拍脉冲的电路图如下：



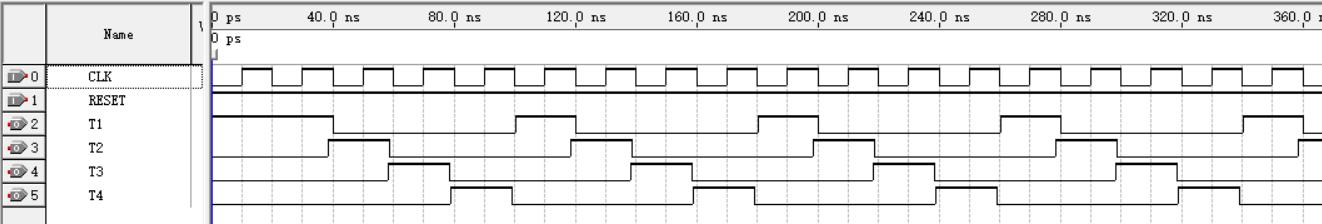
**图 3-1-2 四节拍脉冲发生器电路图**

CLK：触发D触发器, CP脉冲RESET：复位键，使脉冲发生器复位

T1~T4：分别输出对应时序的节拍脉冲



**图3-1-3 四节拍脉冲发生器波形图**



**图3-1-4 四节拍脉冲发生器仿真波形图**

由仿真图可知道，这个四节拍脉冲发生器存在些许缺点，我接下来进行了改进；

由于图1-2的四节拍脉冲不能控制启动和停止，所以有了下面的带启停时序四相节拍脉冲发生器

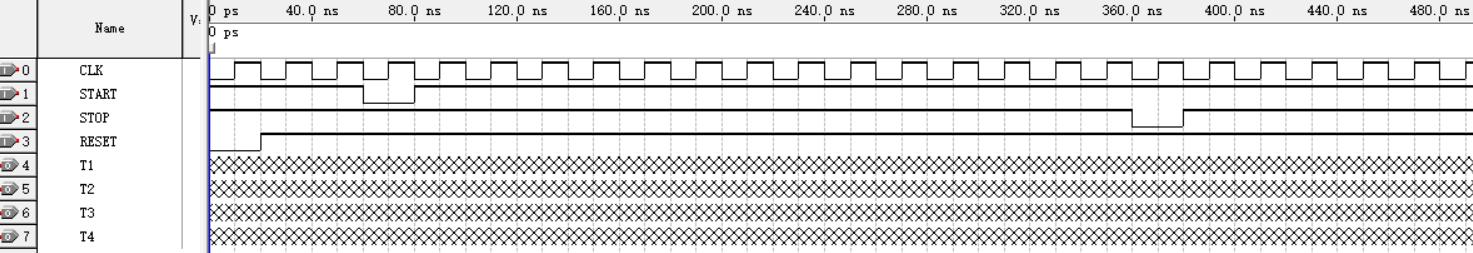
**图3-1-5 带启停时序四相节拍脉冲发生器原理图**

CLK：触发D触发器, CP脉冲RESET：复位键，使脉冲发生器复位

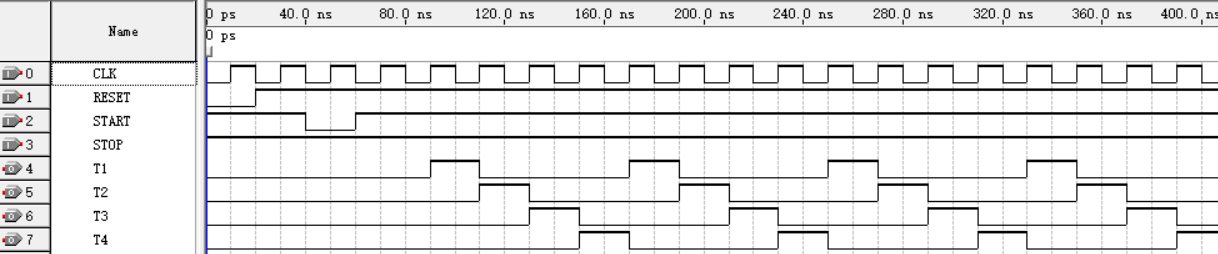
START：启动四节拍脉冲发生器

STOP：停止四节拍脉冲发生器

T1~T4：分别输出对应时序的节拍脉冲



**图3-1-6 带启停时序四节拍脉冲发生器波形图**

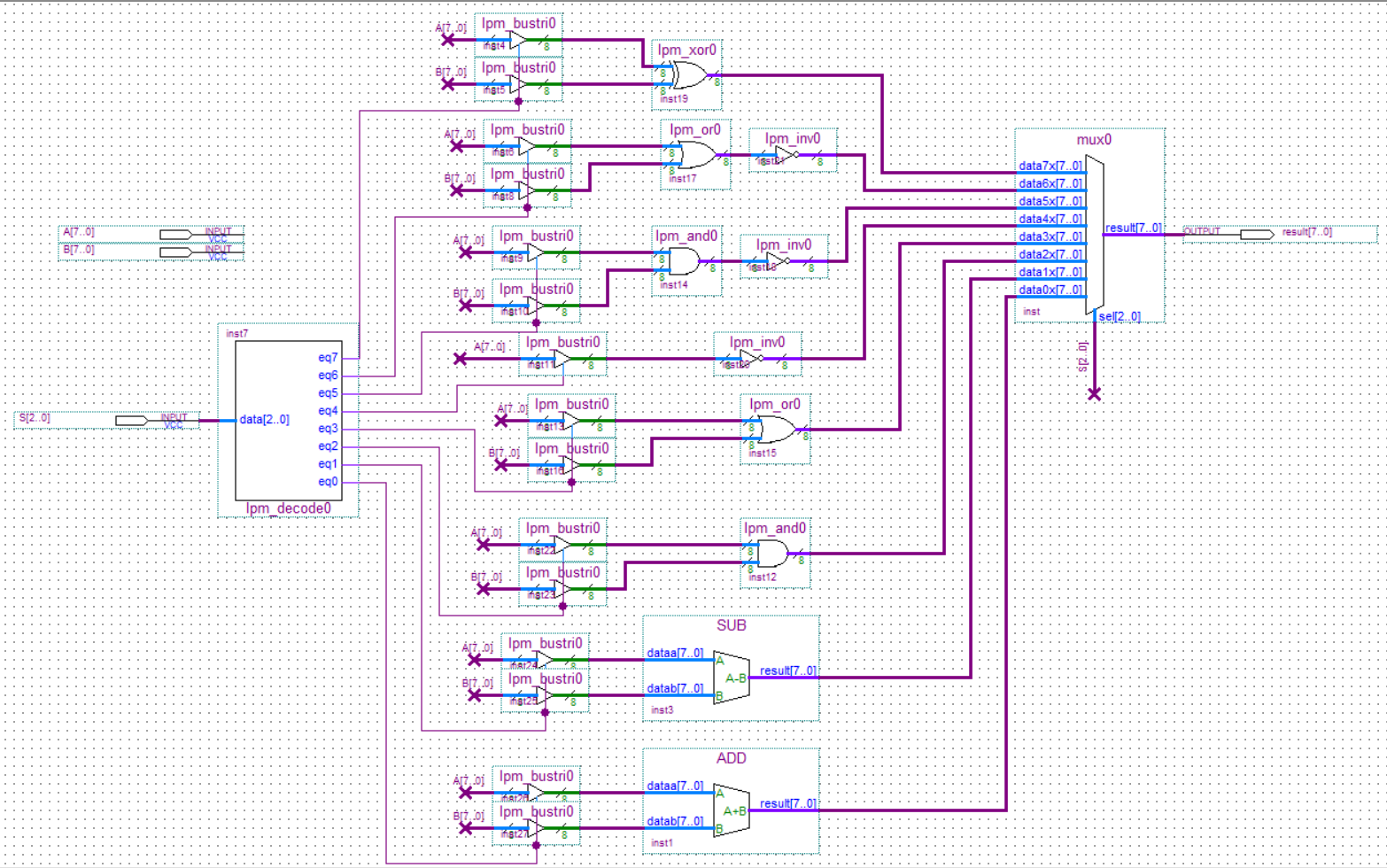


**图3-1-7 带启停时序四节拍脉冲发生器仿真波形图**

1. 运算器 ALU

ALU 用于对相应数据进行算术运算及逻辑运算。

运算器AUL电路图如下:



**图3-2-1 运算器ALU电路图**

A[7..0]：8位二进制数；B[7..0]：8位二进制数；

S[2..0]：3位二进制数，作为选择ALU运算方式

ALU的八种预算指令如下：

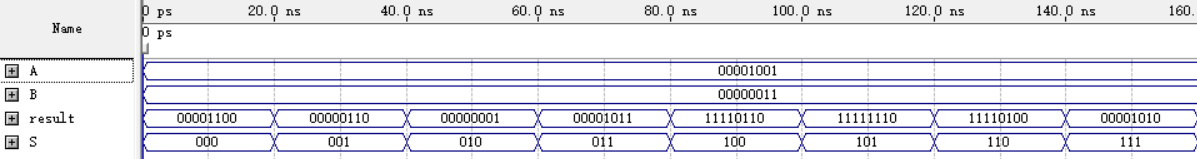
**表 3-2-1 指令集**

|  |  |  |
| --- | --- | --- |
| **指令** | **操作** | **说明** |
| 000 | 算术加 | Y=A+B |
| 001 | 算术减 | Y=A-B |
| 010 | 逻辑与 | Y=AB |
| 011 | 逻辑或 | Y=A+B |
| 100 | 逻辑非 | Y=A |
| 101 | 逻辑与非 | Y=AB |
| 110 | 逻辑与或 | Y=A+B​ |
| 111 | 逻辑异或 | Y=A⨁B |

通过相应的操作码控制相应的运算，输入8位的数据A和B，对其进行运算，并将结果输出到八位的COUT

**表 3-2-2 运算结果参考表**

|  |  |  |
| --- | --- | --- |
| **操作码** | **运算A=1001B B=0011B** | **结果** |
| 000 | A+B | 0000 1100B |
| 001 | A-B | 0000 0110B |
| 010 | A AND B | 0000 0001B |
| 011 | A OR B | 0000 0110B |
| 100 | NOT A | 1111 0110B |
| 101 | NOT (A AND B) | 1111 1110B |
| 110 | NOT (A OR B)​ | 1111 0100B |
| 111 | A XOR B | 0000 1010B |



**图3-2-2 ALU仿真图**

仿真结果与所预想结果一致。

1. 指令存储器ROM

指令存储器 ROM 用于存放指令集，将指令预存到存储文件中，根据地址来获取指令。为了实现八种算术逻辑运算的操作，对寻址方式进行选择，表示寄存器地址、数据存储器地址及数据，需要设计8位的指令，剩下一位无作用，其表示如下：

|  |  |
| --- | --- |
| **操作码 OP** | **寄存器地址** |
| 3位 | 4位 |

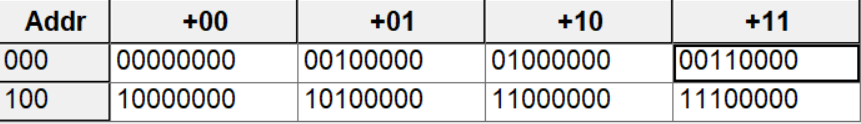
* 1. **指令存储文件**

首先，根据指令格式，对指令进行预存，指令集如下：

**表 3-3-1 指令集表**

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **OP** | **寄存器地址** | **含义** |
| 1 | 000 | 0000 | R1+R3 |
| 2 | 001 | 0000 | R1-R3 |
| 3 | 010 | 0000 | R1 AND R3 |
| 4 | 011 | 0000 | R1 OR R3 |
| 5 | 100 | 0000 | NOT R1 |
| 6 | 101 | 0000 | NOT (R1 AND R3) |
| 7 | 110 | 0000 | NOT (R1 OR R3)​ |
| 8 | 111 | 0000 | R1 XOR R3 |

将以上指令存到存储文件中：

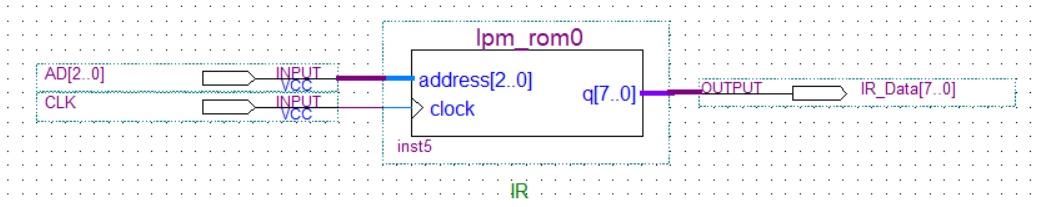


**图 3-3-1指令存储器存储文件展示图**

* 1. **指令存储器的实现**

由于指令存储器只需要从存储器中读出指令，不需要写操作，因此，可以使用只读存储器ROM对指令存储器进行实现。

由于指令集中存放了8条指令，因此需要3位二进制位作为地址选择端，在控制信号到来时，输出该指令。



**图 3-3-2指令存储器电路图**

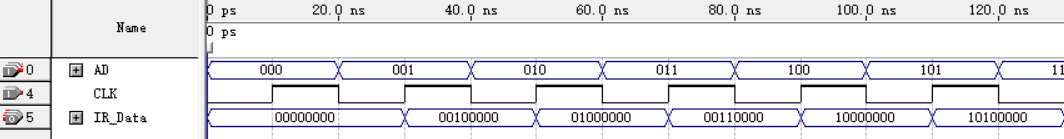
AD[2..0]：3位地址，进行指令选择；

CLK：CP脉冲；

IR\_Data[7..0]：8位地址，输出指令；

* 1. **指令存储器的仿真测试**

仿真结果可看出，通过地址取出的指令与所存储指令一致，该结果符合预期结果。



**图 3-3-3 指令存储器仿真图**

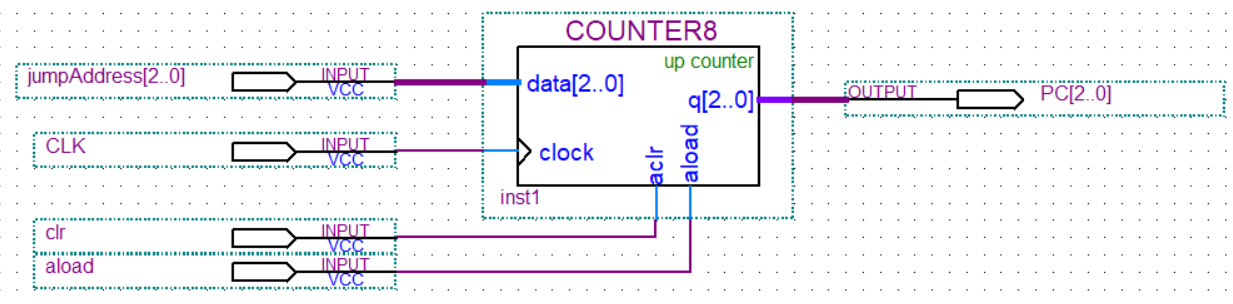
1. 程序计数器PC

程序计数器用于作为地址取出指令存储器的指令。前面所设计的指令存储器中共存储8条指令，需要3位二进制作为地址，因此，需要设计模8的程序计数器 PC，其能在控制信号到来时进行+1操作，从而取出下一条指令。

为了方便后续需要时可以进行程序跳转，在设计程序计数器时，给其设置了同步置数端。当置数端为1时，在控制信号到来时，会将传入的数据置数到程序计数器中，并且在之后的控制信号到来，该程序计数器会从所置的数据继续+1 操作，从而实现程序的跳转。

同时，为了在特殊情况下，能够对程序计数器清零操作，需要给一个清零端 clr，当 clr 为高电平时，能够对其进行清零。

程序计数器的电路图如下：



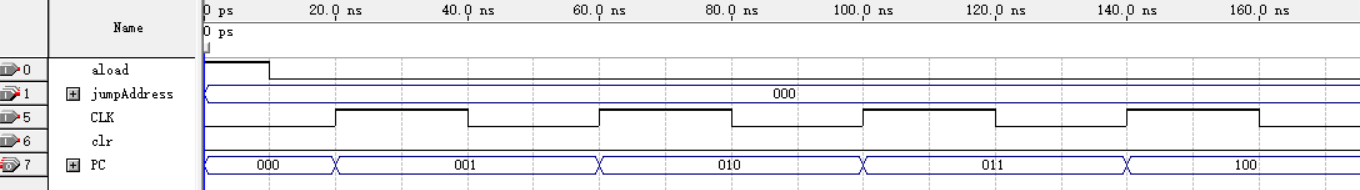
**图 3-4-1 程序计数器PC电路图**

jumpAddress [2..0]：8位二进制数，用于PC跳转相应的地址；CLK：触发PC-COUNTERT计数器自增1，CP脉冲

clr:异步清零PC值,上升沿有效

aload:使设置PC初始值有效,上升沿有效

在控制信号CLK到来时，会进行+1 操作。在aload为高电平时会将jumpAddress的值置数到程序计数器中。在clr为高电平时，在控制信号到来时，会进行同步清零操作。其仿真结果如下：



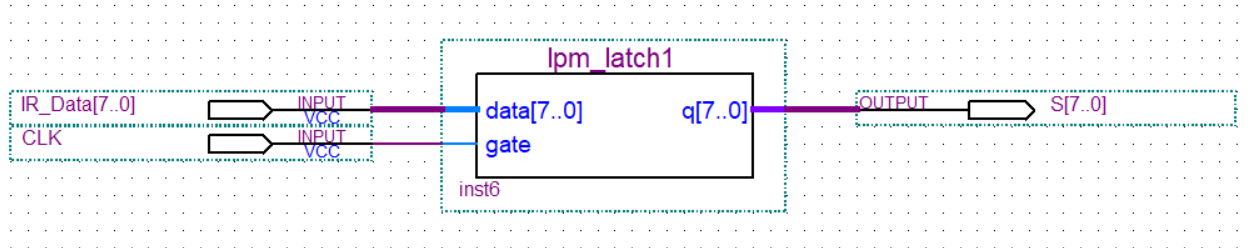
**图3-4-2 程序计数器仿真图**

从仿真结果可看出，该程序计数器符合预期结果。

1. 指令寄存器IR

指令寄存器用于存放从指令存储器中取出的指令，在控制信号到来时，将指令存储到指令存储器中。为了能够在特殊情况下对指令寄存器进行清零操作。

通过调用 lam\_latch 存放8位指令，其电路图如下：



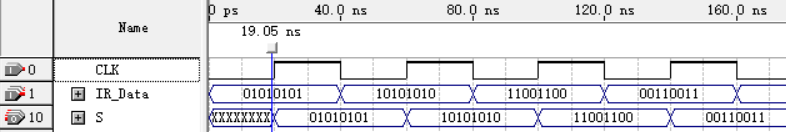
**图 3-5-1 指令寄存器IR电路图**

IR\_Data[7..0]：8位二进制，输入指令；

CLK：CP脉冲，寄存器有效；

S[7..0]：8位二进制，输出指令；

对指令寄存器进行仿真，通过仿真结果可看出，该指令寄存器符合预期结果。

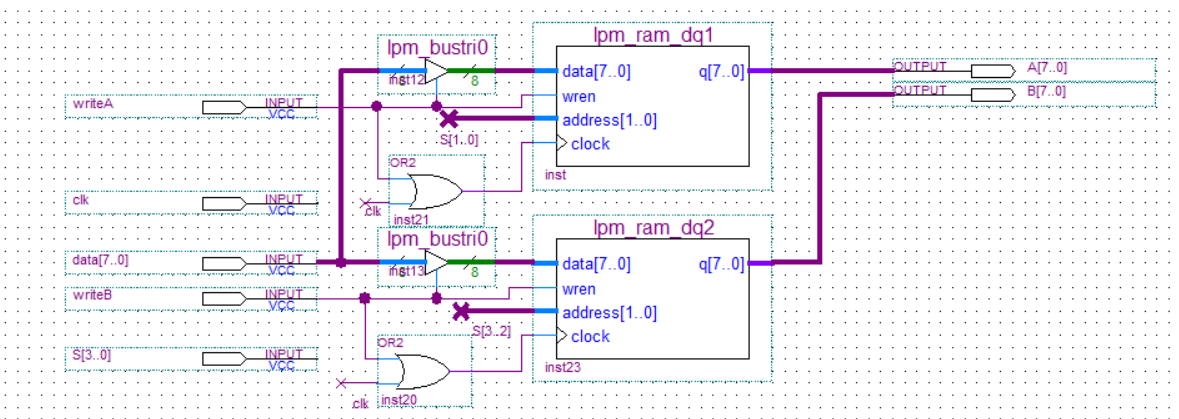


**图 3-5-2 指令寄存器仿真图**

1. 通用寄存器

本实验中，寄存器组设定为8个，即 R0-R7。在本实验中，我使用分别使用了两个RAM存储器作为A、B的数据寄存器，两个RAM可以作为输入存储输出数据，每个存储器可以存储4个数据，既可以事先写入数据在存储器中，也可以调用写入功能，进而实现寄存器的功能。

此处调用了 8 位的 lpm\_ram\_dq 模块作为寄存器，其能在wren为高电平以及clock为上升沿的时候，将外部数据存储到寄存器中。通过调用两块lpm\_ram\_dq 模块，构成八个寄存器作为通用寄存器。通用寄存器的电路图如下：



**图 3-6-1 通用寄存器电路图**

data[7..0]：8位二进制数，数据输入；

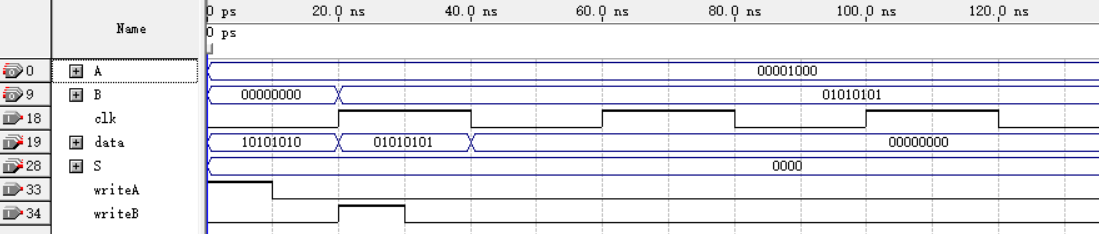
writeA：触发对A寄存器写入操作，高电平有效

writeB：触发对B寄存器写入操作，高电平有效

clk：CP脉冲,输出寄存器的数据；

S[3..0]：通用寄存器地址选择

通过对通用寄存器仿真，可看出其结果符合预期。通用寄存器仿真图如下：

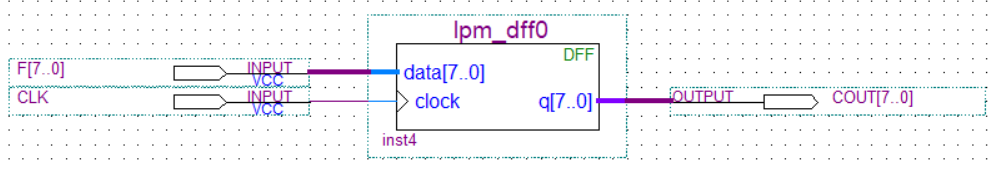


**图 3-6-2 通用寄存器仿真图**

1. 数据缓冲寄存器DR

数据缓冲寄存器用于接收运算器的 8 位处理结果，当控制信号到来时，数据缓冲寄存器将运算器的结果保存到寄存器中。这里使用D触发器实现寄存器功能。

数据缓冲寄存器的电路图如下：

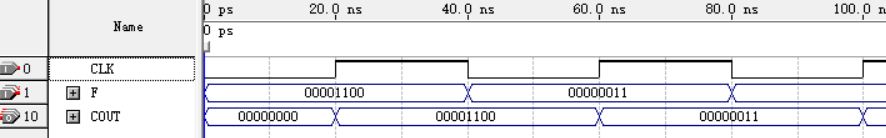


**图 3-7-1 数据缓冲寄存器电路图**

F[7..0]：8位二进制数，作为数据输入；

CLK：CP脉冲；

COUT[7..0]：8位二进制数，输出结果；



**图 3-7-2 数据缓冲寄存器仿真图**

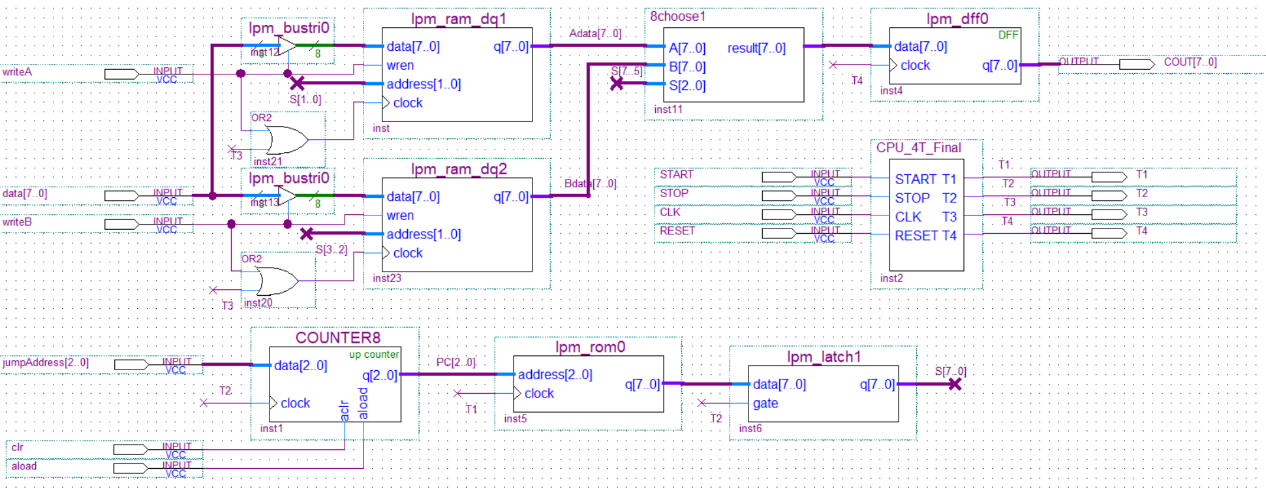
# 系统的实现与测试

基于顶层设计图，对以上部件进行连接，基于顶层设计图，对以上部件进行连接，程序计数器在控制信号T1控制下产生指令地址，作为指令存储器的地址输入端；指令存储器根据地址取出指令，传到指令寄存器 IR 中，指令寄存器在时钟脉冲 T1 的控制下，将指令存放到寄存器中；

在T2节拍进行将IR指令发送到对应的地方，通过指令的S[7..5]作为操作码连接到 ALU 中控制相应的操作；通用寄存器RAM1中的address地址连接 S[1..0]，通用寄存器RAM2中的address地址连接 S[3..2]，用于选出所需的寄存器；寄存器输入端连接data[7..0]，分别在writeA、writeB信号下对A、B寄存器进行数据外部输入，也可以在T3信号控制下通过根据S[3..0]进行地址选择里面存有的MIF文件数据，进行输入到ALU运算器中；

通过指令的操作控制，寻址选择获取数据并进行相应的运算，在 T4 信号控制下将数据存储到 DR 中，并进行结果输出。

总体连接如下：



**图 4-1 顶层连接图**

data[7..0]：8位二进制数，数据输入；

writeA：触发对A寄存器写入操作，高电平有效

writeB：触发对B寄存器写入操作，高电平有效

jumpAddress [2..0]：8位二进制数，用于PC跳转相应的地址；CLK：触发PC-COUNTERT计数器自增1，CP脉冲address[5..0]：6位二进制数，用来选取储存相对应的地址

clr:异步清零PC值,上升沿有效

aload:使设置PC初始值有效,上升沿有效

COUT[7..0]：8位二进制数，输出结果

RESET：复位键，使四节拍脉冲发生器复位

START：启动四节拍脉冲发生器

STOP：停止四节拍脉冲发生器

T1~T4：分别输出对应时序的节拍脉冲

可通过对以下数据进行相应操作，测试其结果。

**表 4-1 预设数据表**

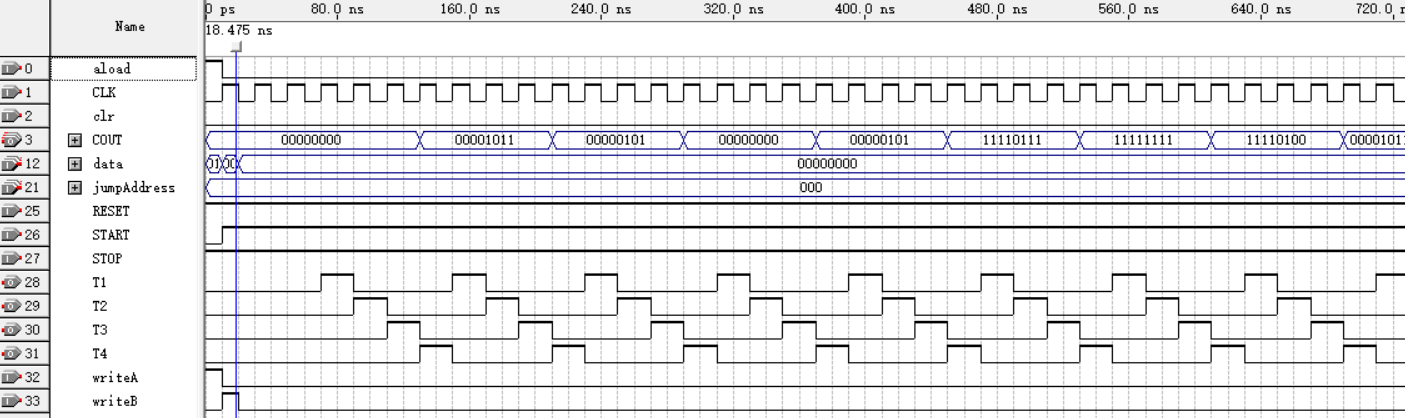
|  |  |
| --- | --- |
| **R1** | **R4** |
| 0000 1001 | 0000 0011 |

执行的指令及结果查看如下表：

**表 4-2 指令执行结果参考表**

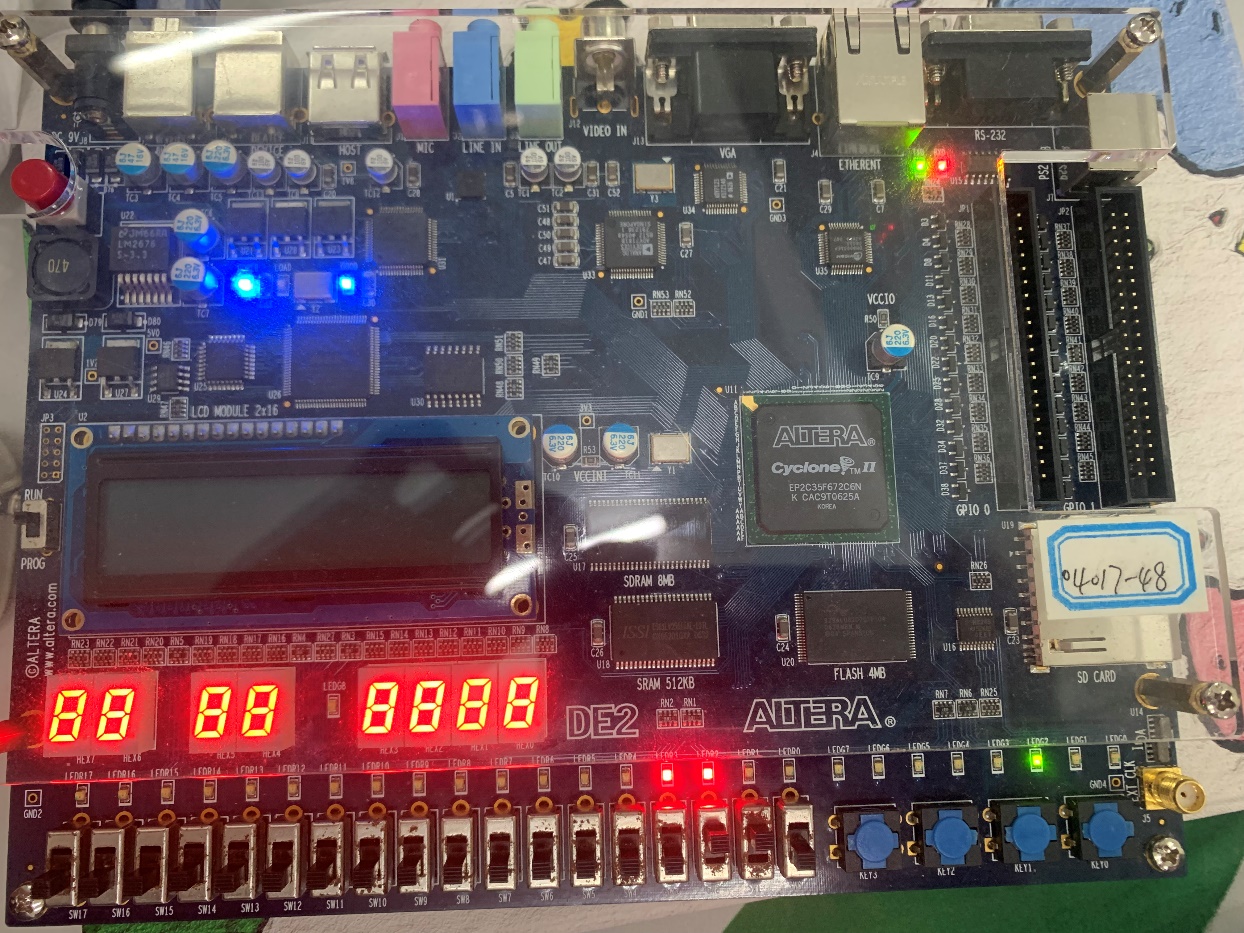
|  |  |  |
| --- | --- | --- |
| **指令** | **含义** | **结果** |
| 0000 0000 | R1+R4 | 0000 1100B |
| 0010 0000 | R1-R4 | 0000 0110B |
| 0100 0000 | R1 AND R4 | 0000 0001B |
| 0110 0000 | R1 OR R4 | 0000 0110B |
| 1000 0000 | NOT R1 | 1111 0110B |
| 1010 0000 | NOT (R1 AND R4) | 1111 1110B |
| 1100 0000 | NOT (R1 OR R4)​ | 1111 0100B |
| 1110 0000 | R1 XOR R4 | 0000 1010B |

对总体进行仿真，将仿真结果与以上参考表进行比较。通过仿真结果可知，该实现符合预期结果。

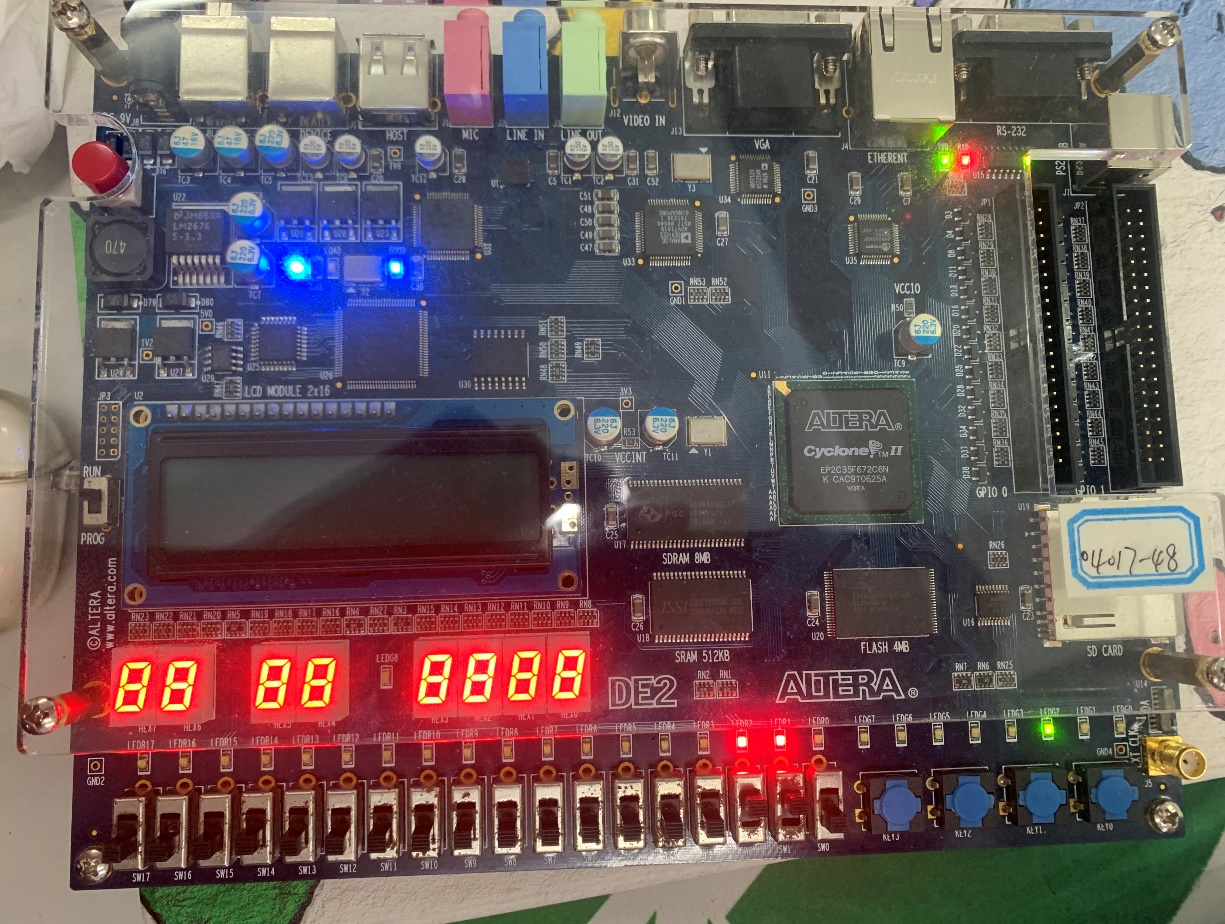


**图 4-2 总体仿真图**

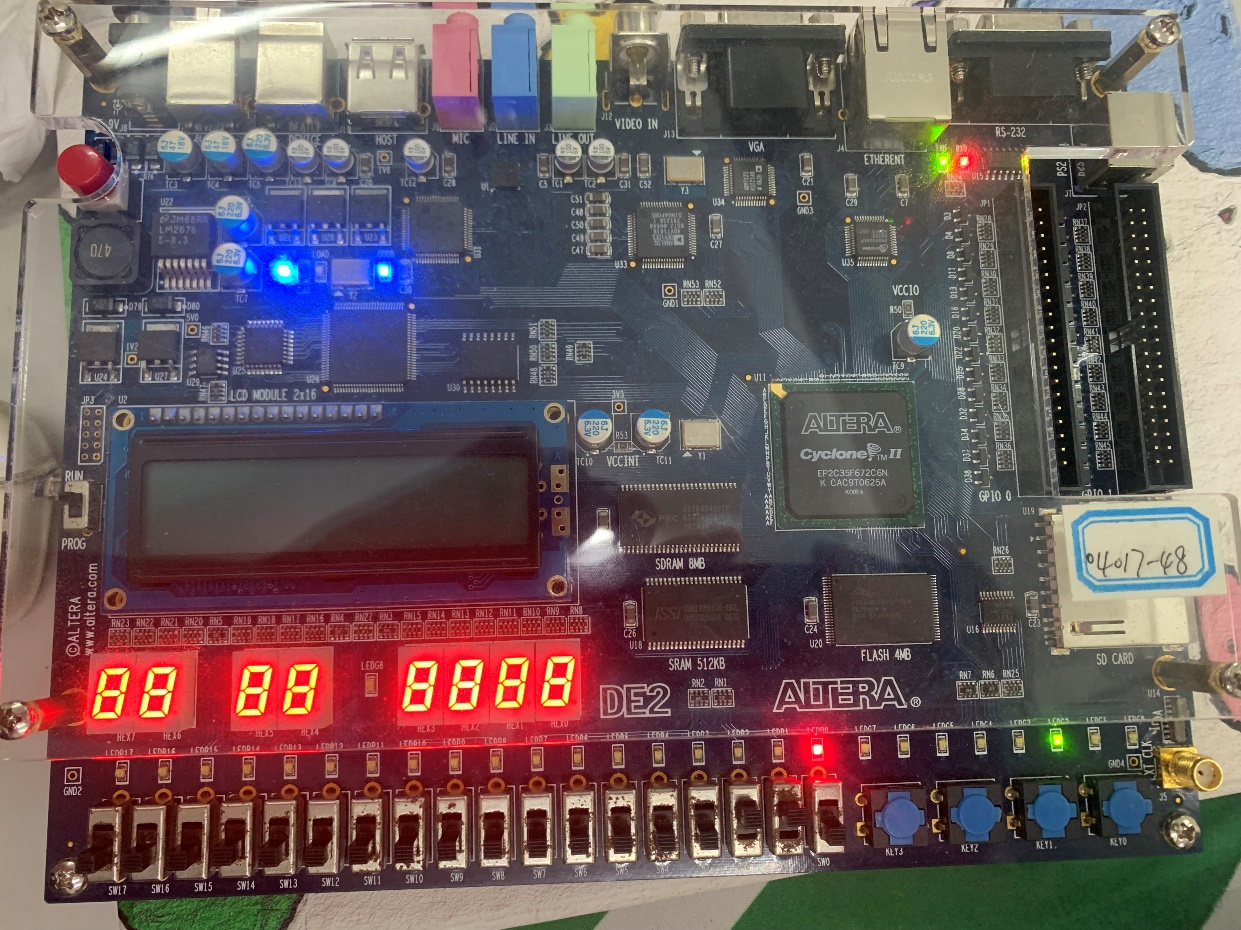
将以上操作实现到DE2开发板上：



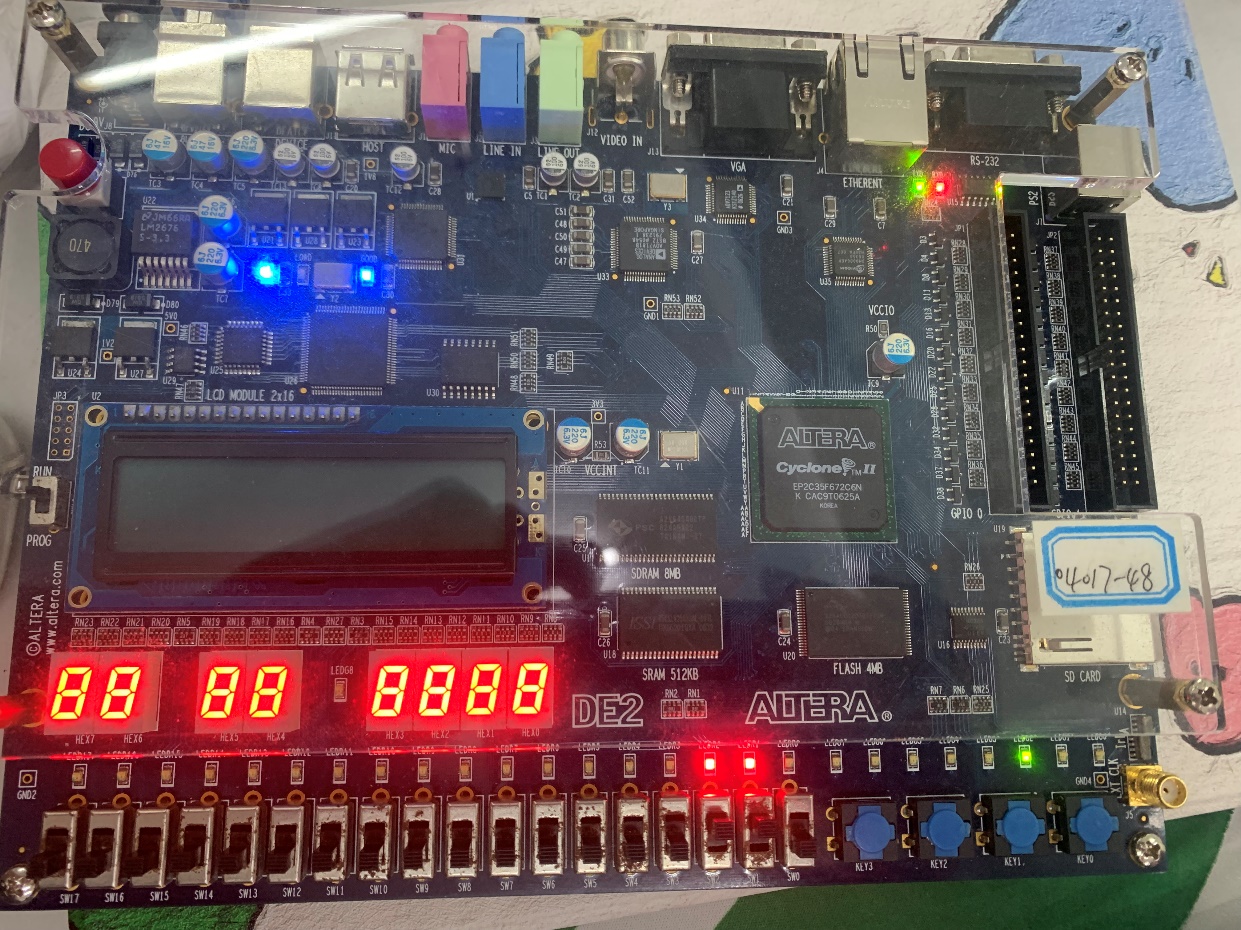
**图 4-3 R1+R3**

****

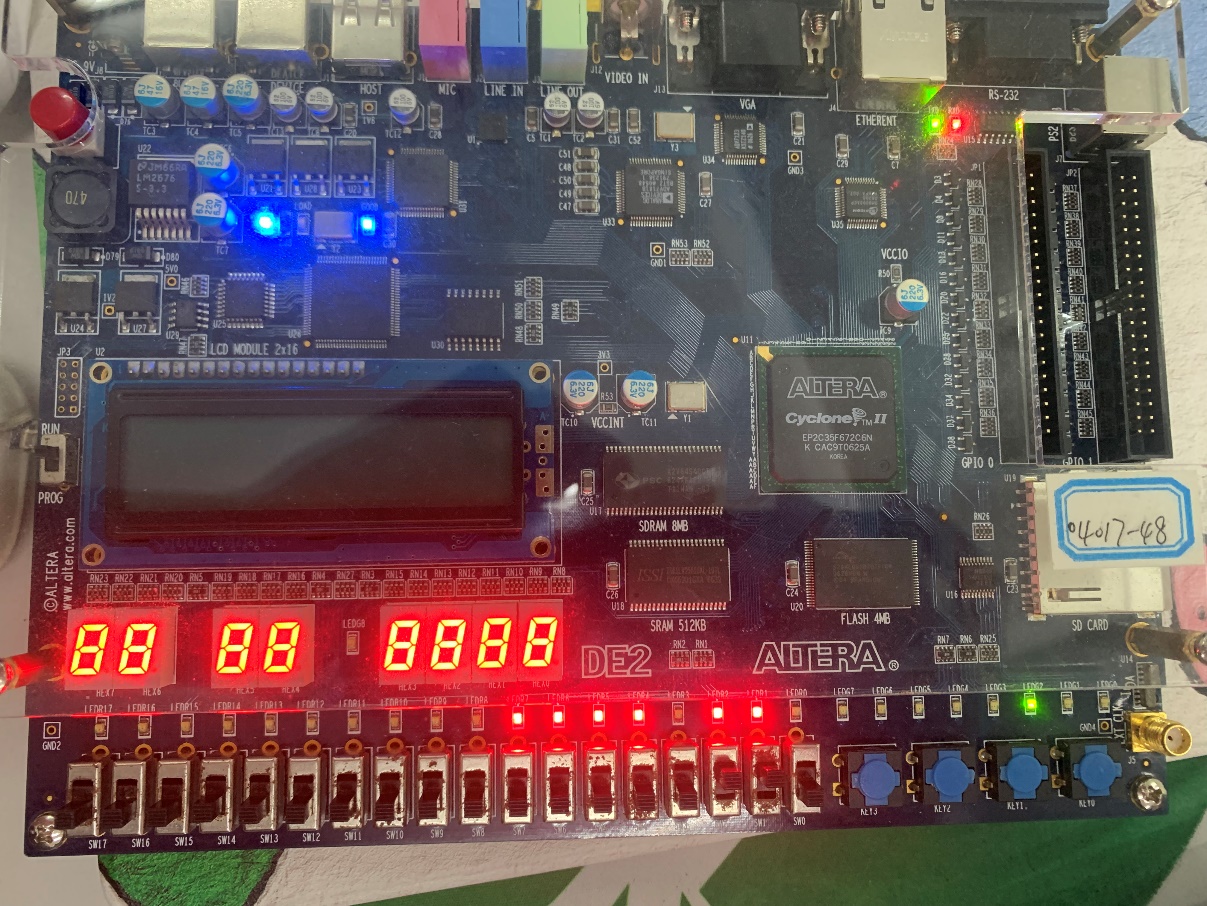
**图 4-4 R1-R3**

****

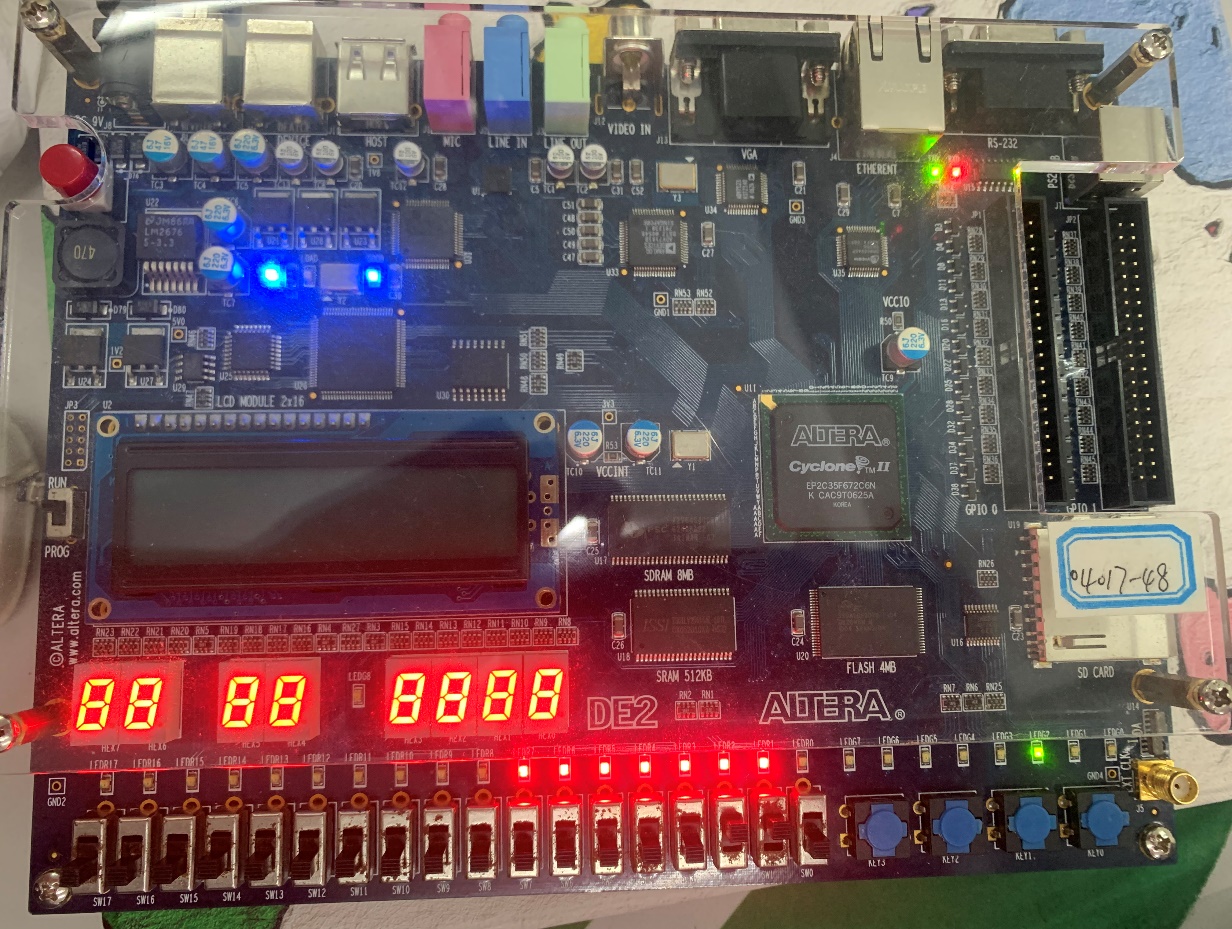
**图 4-5 R1 AND R3**

****

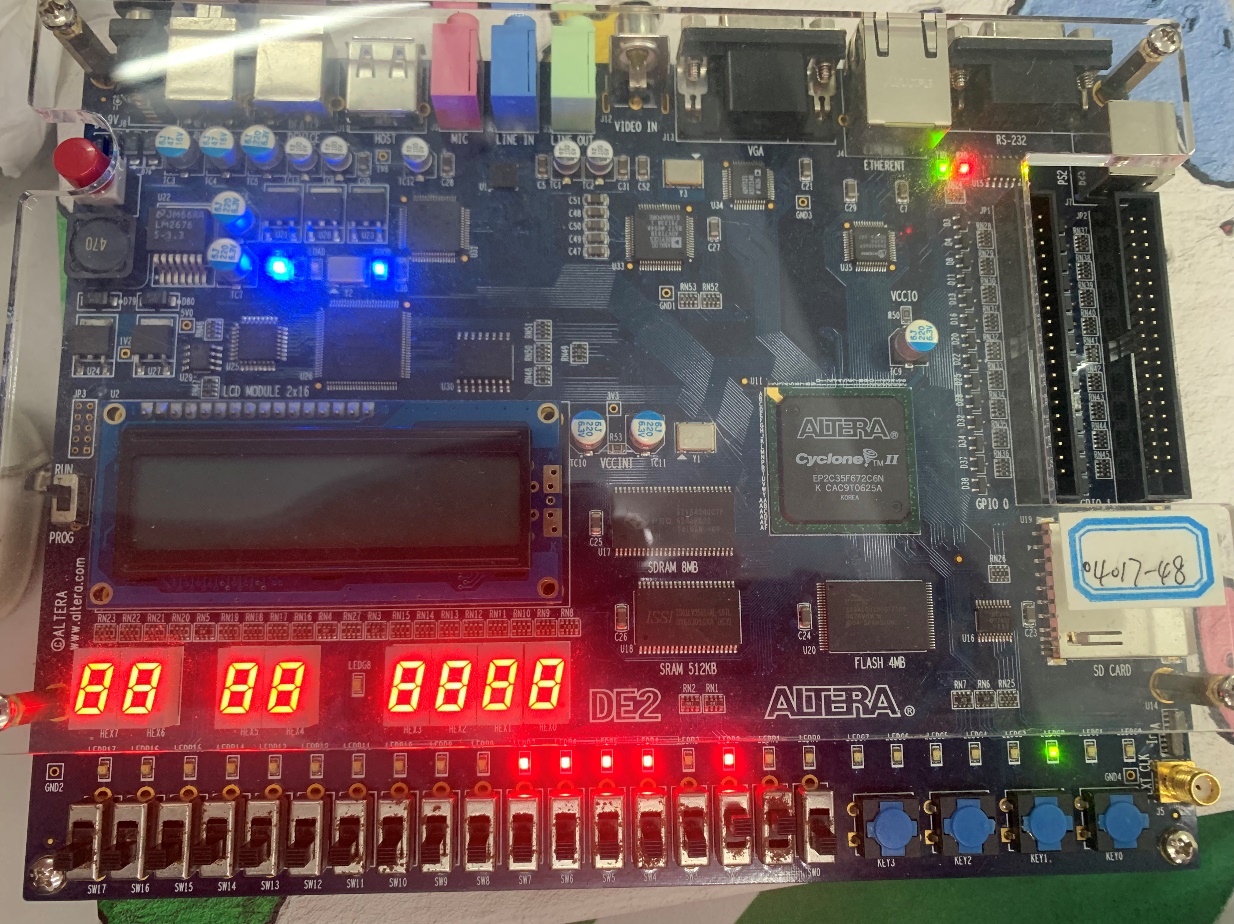
**图 4-6 R1 OR R3**

****

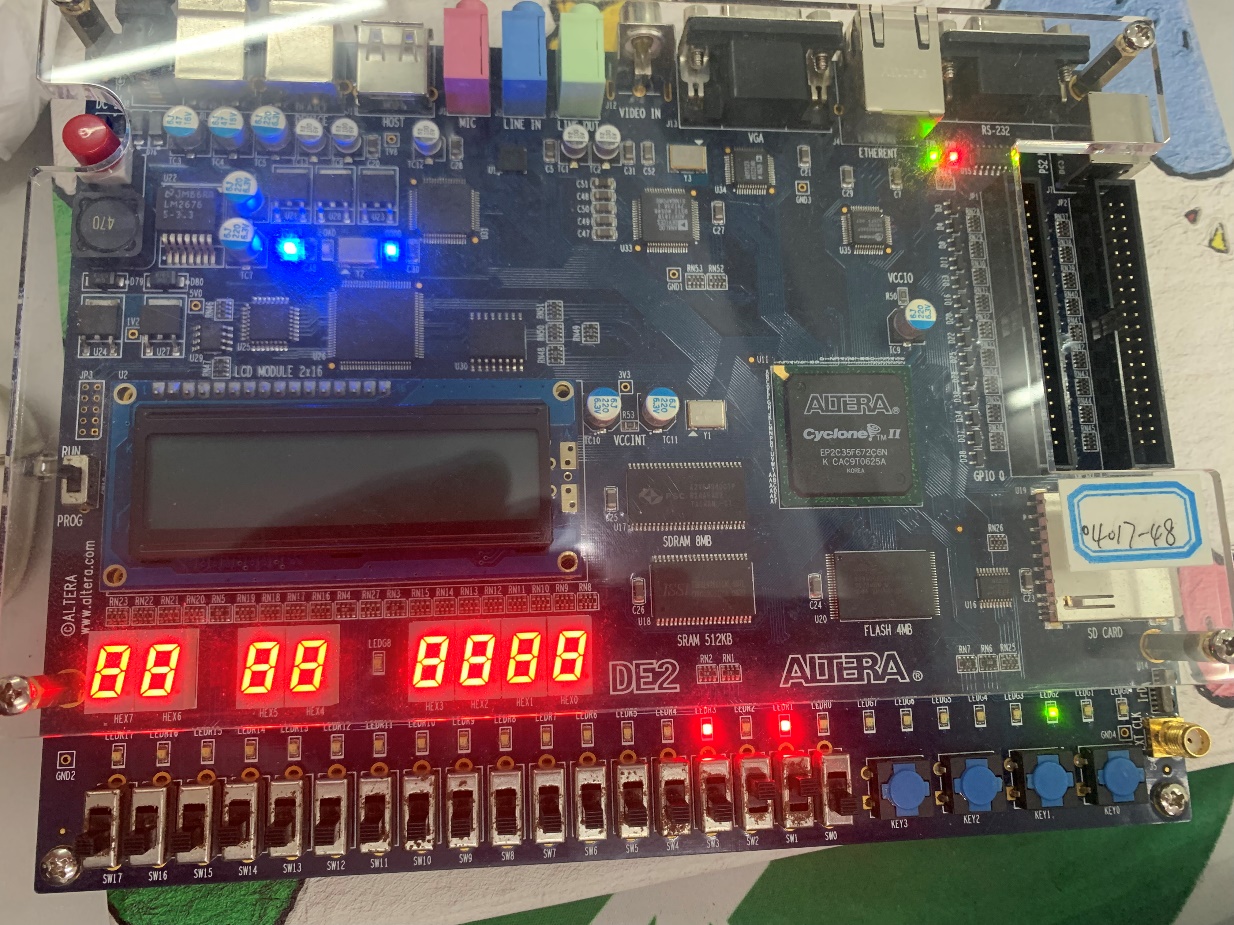
**图 4-7 NOT R1**

****

**图 4-8 NOT (R1 AND R3)**

****

**图 4-9 NOT (R1 OR R3)**

****

**图 4-10 R1 XOR R3**

在DE2开发板上运行的结果与表4-2的结果相符，本次硬件综合实践成功！

# 总结

1. 系统优点:

该系统实现了基本的CPU功能,包括指令获取、运算和结果输出全过程。采用模块化设计,各个部件功能明确,方便调试。通过四节拍脉冲发生器产生的时序信号控制各部件协同工作。实现了8种算术逻辑运算,可以通过指令选择相应的运算。指令格式简单,通过3位操作码选择运算,4位进行寻址。

2. 系统缺点:

指令集和数据宽度较小,只有8条指令,数据宽度8位,不具备扩展性。没有实现更复杂的功能如循环、分支等。输入输出功能不足。中断响应能力缺失。

3. 后续发展方向:

(1)增加更多类型指令:如循环指令、跳转指令、调用栈操作指令等;

(2)扩展数据宽度和地址总线宽度,如扩展为16位或32位;

(3)增加中断控制器,实现中断响应功能;

(4)增加堆栈结构,实现子程序调用等更复杂的功能。

(5)增加存储设备，该系统只能将结果数据进行输出展示，未能将其进行存储或利用，后续也可对其进行完善，将输出的数据进行存储。

4. 个人收获:

1. 对CPU顶层设计有了理解,指令获取、运算、存储输出是CPU 的三大步骤。我们按部就班完成了这三大步骤的设计,理解了时序控制的重要性。
2. 对指令获取过程有了深入理解。我们设计了指令存储器和程序计数器来完成指令获取。理解了程序计数器产生地址和指令存储器根据地址取指令的过程。
3. 对运算过程有了具体的实现。我们设计了算术逻辑运算器来完成运算,根据指令的操作码选择相应的运算,并取出输入数据进行运算,真正地实现了运算过程。
4. 对结果输出有了具体设计。我们使用数据缓冲寄存器DR来存储运算结果并输出,完成了结果输出这最后一步。

所以,通过本次CPU设计,我们不但理解了CPU的抽象工作过程,更重要的是对每一步的细节都有了具体的实现和认识。这为我们后续在学习各种CPU结构和执行过程打下了很好的基础。我们会由此更加深刻地理解和掌握更为复杂的知识点。

5. 评价:

1. 采用自顶向下的系统设计方法,我们按照CPU的工作流程设计了顶层框图,明确了各个模块的功能和连接,这有助于我们理解系统设计的方法。然后采用自底向上的方式逐层设计实现每个模块,这加深了我们对系统分层设计与实现的理解。
2. 硬件设计实践对比软件设计,我们理解了二者之间的差异与联系。硬件设计更注重时序控制和部件连接,软件设计更关注算法和功能实现。但两者都适用类似的设计方法与测试方式。这加深了我们对工程设计与实现的认识。
3. 在设计实现过程中遇到问题是难免的,我们需要学会如何系统地分析问题,查找资料并解决问题。这有助于培养我们的工程思维和解决问题的能力。解决问题的方法和过程也是本次学习的重要收获之一。
4. 我们对各个模块进行了独立测试,以确定其正确性。然后对系统顶层进行仿真测试,从而测试各模块之间的连接与协作。这类似于软件工程中的单元测试和集成测试,有助于我们理解测试在系统设计中的重要意义。通过测试可以验证设计的正确性和找出存在的问题。

本次CPU设计实践不仅加深了我们对CPU的理解,也培养了我们的系统设计与实现能力。让我们领会到工程设计的方法与流程,学会解决问题和测试设计的重要性。这些都是软件工程和系统设计不可或缺的能力,所以本次实践对我们产生了深远的影响,我们受益匪浅。