课程设计报告

课程名称：数字逻辑与数字系统课程设计

设计题目：16 位模型计算机设计与仿真

指导教师：李晶皎

学生姓名：寇凯淇　　学号：20184446

班　　级：计算机 1802

专　　业：计算机科学与技术

设计日期：2020 年 6 月 29 日~2020 年 7 月 24

课程设计任务书

课程设计题目： 16 位模型计算机设计与仿真

一、设计目的

1.掌握 16 位模型计算机的工作原理。

2.在 QUARTUS II 环境下，熟练使用 VHDL 语言完成功能模块和系统编程。

3.在 ModelSim 环境下，熟练使用 VHDL 语言完成功能模块和系统的仿真。

二、设计任务和要求

用所学数字逻辑和数字系统的知识，用 VHDL 语言设计“16 位模型计算机”。因疫情原因，用 ModelSim 实现功能模块仿真和系统仿真。

第一阶段完成选题，规划系统功能；

第二阶段完成功能模块设计和仿真，阶段验收；

第三阶段完成系统设计和仿真，系统验收；

第四阶段书写课程设计报告。

摘要

本设计将自顶向下地对 16 位模型计算机设计，完成系统设计、功能模块和仿真、系统顶层设计与仿真，加深了对“数字逻辑与数字系统”知识的理解，强化了理论知识，掌握了的实践和应用。

在 Quartus II环境下，采用 VHDL 语言构建算术逻辑运算单元、累加器、控制器、地址寄存器、程序计数器、数据寄存器、存储器、节拍发生器、时钟信号源、指令寄存器、指令译码器等功能模块，以及模型计算机系统。在 ModelSim 仿真环境下，完成功能模块，以及模型系统仿真。

功能模块主要有：CTRL 控制器、ALU 算术逻辑单元、IREG 指令寄存器、PC 程序计数器、MAREG 地址寄存器、DREG 数据寄存器、RAM 存储器、COUNTER 节拍发生器、CLOCK 时钟信号产生器。系统驱动时钟由时钟信号产生器分频得到，并通过节拍脉冲发生器对各功能模块的微操作实现有序控制。

本系统的设计在满足基本要求的前提下，进行了进一步的功能扩充。为保证运行更多有意义的，对指令集做到了尽可能的丰富，例如：指令集包含算术指令、逻辑指令、存取指令和停机指令等共计 10 条。

关键词：Quartus II, 16 位模型机, ModelSIM, VHDL 语言

目录

第 1 章 绪论

1.1 16 位模型计算机简介

为了更好地理解数字逻辑与数字系统课程的基础知识，进一步学习计算机的基本结构和原理。在本次课程设计中，选择了模型计算机的设计与实现题目，为了进一步提高自己的能力，增强自己的水平，对该题目做出的改进有提高模型机位数至 16 位，增加减法、自增、自减、逻辑与、或、非、异或共 7 条运算功能指令。

模型计算机，顾名思义，就是以实际的计算机结构为基础，对其进行抽象和简化，使之具备计算机的基本结构和功能，可以对数据或指令进行处理和执行。

模型计算机应该具备以下模块：CPU、存储单元、输入和输出、以及总线，可以使本系统具有更好的演示性和可操作性[1]。

根据汇编语言的知识，CPU 模块的核心器件是控制器，数据寄存器(DR，ACC)，专用寄存器(例如：指令寄存器，程序计数器和地址寄存器等)，以及用于驱动各器件协同有序工作的时序发生器。

1.2 设计主要内容

1.2.1 设计指标

本模型计算机具备以下功能、模块，以及参数指标：

1. 模块：存储器模块、CPU 模块；
2. 总线：包括地址总线和数据总线，总线位宽为 16 位；
3. 指令集：涵盖基本的汇编指令，如算术运算指令、逻辑运算指令、存取指令和停机指令，共计 10 条；
4. 存储器容量：16 × 16位；
5. 输入输出方式：在仿真文件通过 Add_IN 和 Input 端口输入。模型机的信息在相应的输出端口展示。

1.2.2 设计思路

1. 模块化设计

(1)根据确立的设计指标，对各个模块进行单独设计和仿真，对该模块可能涉及的作用和功能有清晰地认识，同时需要注意各模块协同工作时的时序关系和控制信号；

(2)模块设计的方式可以更为多样化，以 VHDL 语言编程为主、电路原理图设计和状

态转换图设计等方式为辅，尽可能多地熟悉 Quartus II 软件的使用方法和功能；

(3)自顶向下，逐层设计。对各模块的设计应从顶层的应用/功能入手，分解其在执行指令中选通信号的控制情况和模块工作的先后情况，对可能出现的问题要及早发现及早改正，以免后续综合时，对系统产生不确定性影响。

2. 顶层设计与描述

当功能模块设计完成后，可由 VHDL 代码生成元件符号，根据模型机的结构，将各元器件用总线和控制线连接起来，连接的顺序由小及大、由内到外；每连完一个模块就测试一个模块，确保模块内部可以正常工作。

3. 仿真

在设计过程中，及时对小的模块进行仿真验证，通过才能继续设计。若只在最后进行仿真验证，则对于出现的问题将可能很难发现或解决，从而增大了程序调试的任务量，迟滞了课程设计的进度。仿真工具使用 ModelSim，编写 VHDL 测试文件。

第 2 章 系统设计

2.1 模型计算机原理

所谓模型计算机就是以计算机实际结构为基础，将其简化，能对输入的信息进行处理运算，更便于分析设计。

计算机主要由运算器、控制器、存储器三大部分组成。计算机能按照用户要求、完成提前设计好的指令，指令是计算机执行具体操作的命令。一条指令就是机器语言的一个语句，用来说明机器硬件要完成什么样的基本操作。

在设计整体结构时，依据的是各指令的数据通路。然后采用自顶向下，逐步分解细化的方法进行设计。先整体模块，后局部模块。

在本设计中，把模型计算机划分成十余个基本模块，分别是存储器、时钟信号源、节拍发生器、操作控制器、程序计数器、地址寄存器、累加器、算术逻辑单元、数据寄存器、指令寄存器和指令译码器。

让预设指令在这些部件中按顺序执行达到预期目的。计算机执行一条指令分为三步进行：

第 1 步是取指令，将要执行的指令从内存取到控制器中；

第 2 步是分析指令，对所取的指令通过译码器进行分析判断，判断该指令要完成的操作；

第 3 步是执行指令，根据分析结果向各部件发出操作信息，执行该指令相应的操作功能。

2.2 模型计算机组成

在设计整体结构时，依据的是各指令的数据通路。然后采用自顶向下，逐步分解细化的方法进行设计。先整体模块，后局部模块。

从整体上看，模型计算机主要分为：CPU 模块和存储模块。模型计算机结构框图如图 2-2-1 所示，具体模块有：节拍发生器模块 CT、指令寄存器模块 IR、算术逻辑单元模块 ALU、数据寄存器模块 DR、程序计数器模块 PC、地址寄存器模块 MAR、操作控制器模块 CTRLM、累加器 ACC 等[2]。

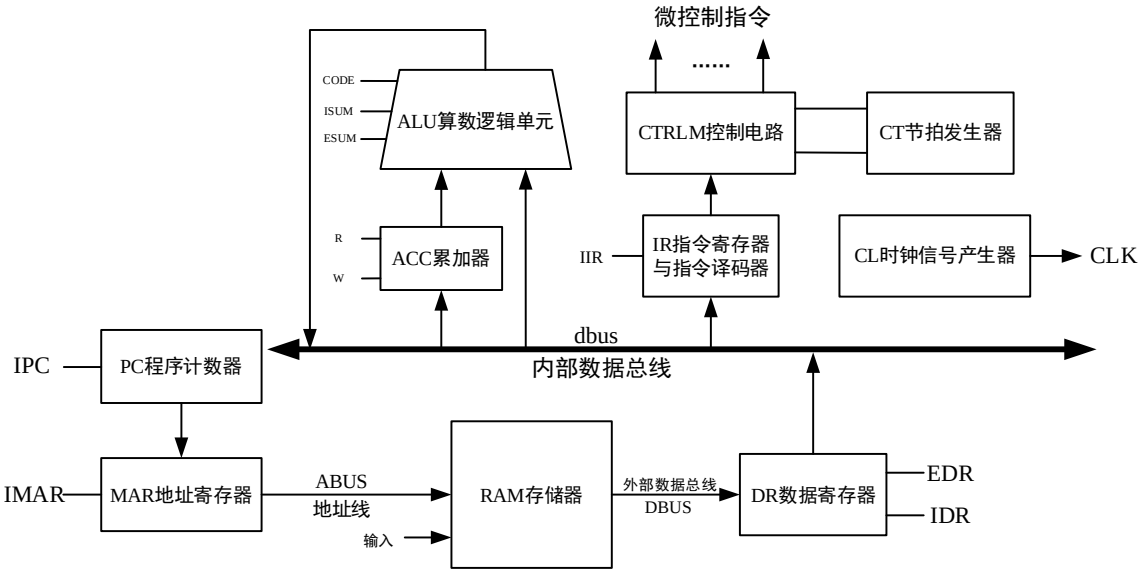

图 2-2-1 模型计算机结构框图

2.3 模型计算机的指令系统设计

在模型计算机中，指令用于验证模型机能否正常工作。模型计算机可完成加法、减法、自增、自减、逻辑与、或、非、异或共 8 种运算指令，并将结果保存至累加器中。

指令动作	16 位指令代码	4 位指令信号
移数 LD	0000000000111110	0001
加法 ADD	0000000011000110	0010
减法 SUB	0001001100011110	0011
自增 INC	0010111011100111	0100
自减 DEC	0000111100000001	0101
逻辑与 AND	1011100010000011	0110
逻辑或 OR	1000011111111110	0111
逻辑非 NOT	0101010101011110	1000
逻辑异或 XOR	0111100001010000	1001
停机 HALT	0000000001110110	1111

为了方便描述，累加器以“A”表示，操作数以“num”表示，具体如下：

(1) LD A, num；A ← num，把 num 送入累加器 A，操作码是 0000000000111110；

(2) ADD A, num；A ← A + num，把 A 中数字与 num 相加，结果送入累加器 A，操作码是 0000000011000110；

(3) SUB A, num；A ← A - num，把 A 中数字与 num 相减，结果送入累加器 A，操作码是 0001001100011110；

(4) INC A；A ← A + 1，把 A 中数字自增，结果送入累加器 A，操作码是 0010111011100111；

(5) DEC A；A ← A - 1，把 A 中数字自减，结果送入累加器 A，操作码是 0000111100000001；

(6) AND A, num；A ← A and num，把 A 中数字与 num 相与，结果送入累加器 A，操作码是 1011100010000011；

(7) OR A, num；A ← A or num，把 A 中数字与 num 相或，结果送入累加器 A，操作码是 1000011111111110；

(8) NOT A；A ← not A，把 A 中数字求非，结果送入累加器 A，操作码是 0101010101011110；

(9) XOR A, num；A ← A xor num，把 A 中数字与 num 相异或，结果送入累加器 A，操作码是 0111100001010000；

(10) HALT；停机，操作码是 0000000001110110。

总线结构是单总线，数据总线位数 16 位，存储器容量是 16×16 位；地址总线是 4 位。算术逻辑单元实现相应运算操作。

根据模型计算机的结构框图，可设计指令系统中每条指令的执行流程。一条指令从存储器中取出到执行完，需要若干个机器周期，任何指令的第一个机器周期都是“取指令周期”，一条指令一共需要几个机器周期，取决于指令在机内实现的复杂程度。

(1) 本模型计算机指令“LD A，num”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=0001

T3：(PC)→MAR→ABUS, IMAR=1

T4：DBUS→DR, IDR=1, EDR=1

T5：(PC)+1→PC, IPC=1

T6：dbus→A, W=1

T7：空

(2) 本模型计算机指令“ADD A，num”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=0010

T3：(PC)→MAR→ABUS, IMAR=1

T4：DBUS→DR, IDR=1, EDR=1

T5：(PC)+1→PC, IPC=1

A + num, R=1, ISUM=1

T6：ALU→dbus, ESUM=1

dbus→A, W=1

T7：NULL

(3) 本模型计算机指令“SUB A，num”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=0011

T3：(PC)→MAR→ABUS, IMAR=1

T4：DBUS→DR, IDR=1, EDR=1

T5：(PC)+1→PC, IPC=1

A - num, R=1, ISUM=1

T6：ALU→dbus, ESUM=1

dbus→A, W=1

T7：NULL

(4) 本模型计算机指令“INC A”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=0100

T3：A + 1, R=1, ISUM=1

T4：ALU→dbus, ESUM=1

dbus→A, W=1

T5：NULL

T6：NULL

T7：NULL

(5) 本模型计算机指令“DEC A”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=0101

T3：A - 1, R=1, ISUM=1

T4：ALU→dbus, ESUM=1

dbus→A, W=1

T5：NULL

T6：NULL

T7：NULL

(6) 本模型计算机指令“ADD A，num”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=0110

T3：(PC)→MAR→ABUS, IMAR=1

T4：DBUS→DR, IDR=1, EDR=1

T5：(PC)+1→PC, IPC=1

A and num, R=1, ISUM=1

T6：ALU→dbus, ESUM=1

dbus→A, W=1

T7：NULL

(7) 本模型计算机指令“OR A，num”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=0111

T3：(PC)→MAR→ABUS, IMAR=1

T4：DBUS→DR, IDR=1, EDR=1

T5：(PC)+1→PC, IPC=1

A or num, R=1, ISUM=1

T6：ALU→dbus, ESUM=1

dbus→A, W=1

T7：NULL

(8) 本模型计算机指令“NOT A”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=1000

T3：not A, R=1, ISUM=1

T4：ALU→dbus, ESUM=1

dbus→A, W=1

T5：NULL

T6：NULL

T7：NULL

(9) 本模型计算机指令“XOR A，num”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(PC)+1→PC, IPC=1

(DR)→IR, IIR=1，Code=1001

T3：(PC)→MAR→ABUS, IMAR=1

T4：DBUS→DR, IDR=1, EDR=1

T5：(PC)+1→PC, IPC=1

A xor num, R=1, ISUM=1

T6：ALU→dbus, ESUM=1

dbus→A, W=1

T7：NULL

(10)本模型计算机指令“HALT”流程如下：

T0：(PC)→MAR→ABUS, IMAR=1

T1：DBUS→DR, IDR=1, EDR=1

T2：(DR)→IR, IIR=1，Code=1111

IPC=0, EDR=0

T3：NULL

T4：NULL

T5：NULL

T6：NULL

T7：NULL

第 3 章 功能模块设计与仿真

3.1 节拍发生器

节拍发生器用于产生八个节拍脉冲信号 T0~T7，以便控制计算机按固定节拍有序地工作。构成节拍发生器的关键在于环形移位寄存器的初始状态要置成 00000001，在 CLR=1 信号作用下，节拍发生器置为初始状态，CLR=0 时，每经过一个时钟信号上升沿，产生下个节拍信号，如此循环。

3.1.1 节拍发生器的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-1-1 所示，编译成功后得到 RTL Viewer 如图 3-1-2 所示。

```vhdl
--COUNTER节拍发生器
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity COUNTER is
    port (CLK : in std_logic;  --时钟信号
          CLR : in std_logic;  --初始
          T : out std_logic_vector(7 downto 0)
          );
end COUNTER;
architecture vhd_counter of COUNTER is
signal temp : std_logic_vector(7 downto 0);
begin
    T <= temp;
    process(CLK, CLR)
    begin
        if(CLR = '1') then    --初始
            temp <= "00000001";
        elsif(rising_edge(CLK)) then  --时钟信号上升沿
            temp(0)<=temp(7);
            temp(1)<=temp(0);
            temp(2)<=temp(1);
            temp(3)<=temp(2);
            temp(4)<=temp(3);
            temp(5)<=temp(4);
            temp(6)<=temp(5);
            temp(7)<=temp(6);
        end if;
    end process;
end vhd_counter;
```

图 3-1-1 节拍发生器 VHDL 代码

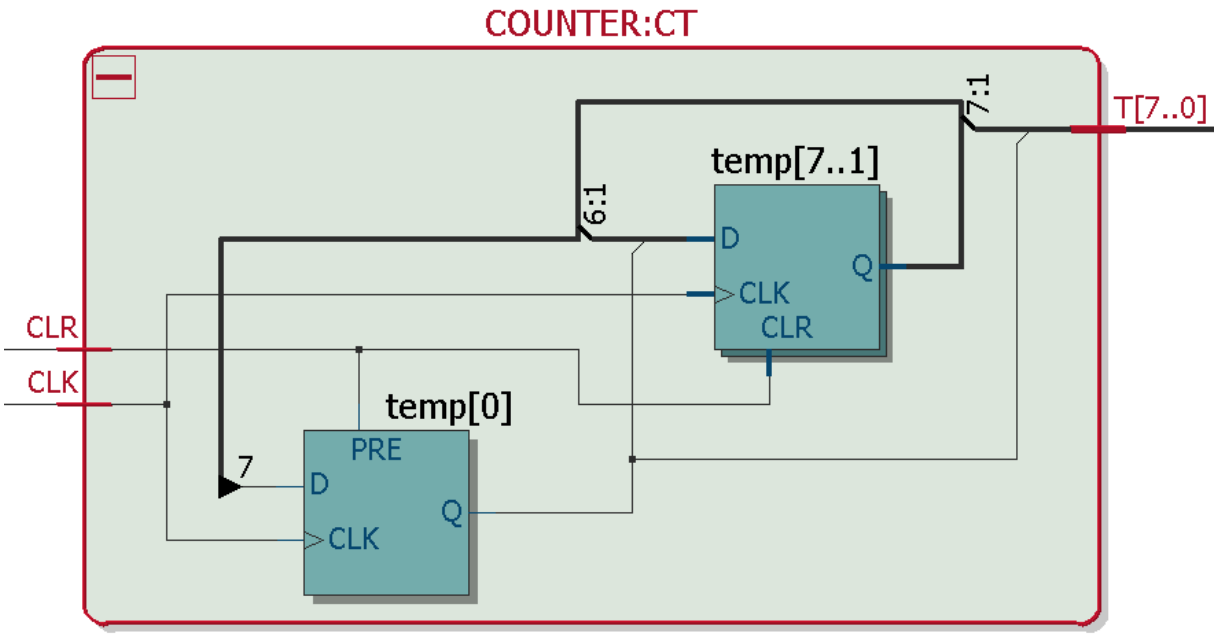

图 3-1-2 节拍发生器的 RTL 图

3.1.2 节拍发生器仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-1-3 所示，运行结果如图 3-1-4。

```
1  --节拍发生器仿真文件
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity ms_counter is
7  end ms_counter;
8
9  architecture modelsim of ms_counter is
10 component COUNTER --节拍发生器
11    port (CLK : in std_logic;  --时钟信号
12          CLR : in std_logic;  --初始
13          T : out std_logic_vector(7 downto 0)
14          );
15    end component;
16
17 signal CLK : std_logic; --时钟信号
18 signal CLR : std_logic; --初始
19 signal T : std_logic_vector(7 downto 0);
20 begin
21 CT : COUNTER port map(CLK, CLR, T);
22 process
23    begin
24       CLK<= '0';
25       wait for 20 ns;
26       CLK<='1';
27       wait for 20 ns;
28    end process;
29 process
30    begin
31       CLR <= '1';
32       wait for 60 ns;
33       CLR <= '0';
34       wait;
35    end process;
36
37 end modelsim;
```

图 3-1-3 节拍发生器测试向量代码

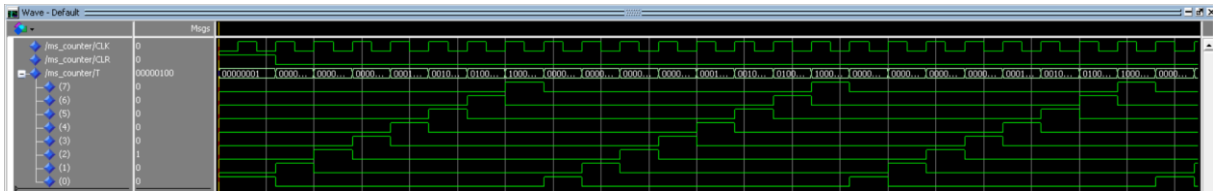

图 3-1-4 节拍发生器 ModelSim 仿真结果

3.2 程序计数器 PC

程序计数器的作用是确定下一条指令的地址。由于模型计算机有 16 个字节的机器码，所以程序计数器 PC 的输出只使用 4 位。当 IPC=0 时，计数器保持原状态；当 IPC=1 时，计数器处于计数状态，当时钟信号 CLK 上升沿到来时，做加 1 运算。

3.2.1 程序计数器 PC 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-2-1 所示，编译成功后得到 RTL Viewer 如图 3-2-2 所示。

```
1  --PC程序计数器
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity PC is
7     port (CLK : in std_logic;  --时钟信号
8           CLR : in std_logic;
9           IPC : in std_logic;  --控制信号
10          PC_OUT : out std_logic_vector(3 downto 0)
11          );
12 end PC;
13
14 architecture vhd_pc of PC is
15 signal temp : std_logic_vector(3 downto 0);
16 begin
17    process(CLK, CLR, IPC)
18    begin
19       if(CLR = '1') then    --清零
20          temp <= "0000";
21       elsif(rising_edge(CLK)) then  --时钟上升沿
22          if(IPC = '1')then --控制信号有效
23             temp <= temp + 1; --PC + 1
24          end if;
25       end if;
26    end process;
27 PC_OUT <= temp;
28 end vhd_pc;
```

图 3-2-1 程序计数器的 VHDL 代码

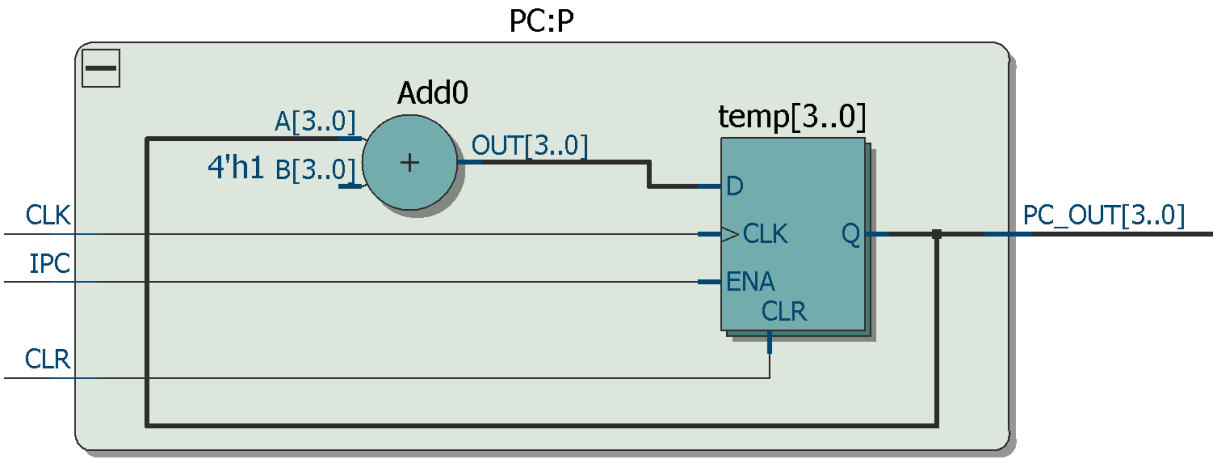

图 3-2-2 程序计数器的 RTL 图

3.2.2 程序计数器的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-2-3 所示，运行结果如图 3-2-4。

```
1  --程序计数器仿真文件
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity ms_pc is
7  end ms_pc;
8
9  architecture modelsim of ms_pc is
10 component PC   --程序计数器
11    port (CLK : in std_logic;  --时钟信号
12          CLR : in std_logic;
13          IPC : in std_logic;  --控制信号
14          PC_OUT : out std_logic_vector(3 downto 0)
15          );
16    end component;
17
18    signal CLK : std_logic; --时钟信号
19    signal CLR : std_logic;
20    signal IPC : std_logic; --控制信号
21    signal PC_OUT : std_logic_vector(3 downto 0);
22 begin
23 P : PC port map(CLK, CLR, IPC, PC_OUT);
24 process
25    begin
26       CLK<= '0';
27       wait for 20 ns;
28       CLK<='1';
29       wait for 20 ns;
30    end process;
31
32 process
33    begin
34       CLR<= '1';
35       wait for 40 ns;
36       CLR<='0';
37       wait;
38    end process;
39
40 process
41    begin
42       IPC<= '0';
43       wait for 60 ns;
44       IPC<='1';
45       wait for 60 ns;
46    end process;
47 end modelsim;
```

图 3-2-3 程序计数器仿真文件

图 3-2-4 程序计数器仿真结果

3.3 地址寄存器 MAR

地址寄存器的作用是保存 4 位地址信息。当时钟信号上升沿来临且 IMAR=1 时，地址寄存器保存输入的 4 位地址信息数据，并输出。

3.3.1 地址寄存器 MAR 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-3-1 所示，编译成功后得到 RTL Viewer 如图 3-3-2 所示。

```
1  --MAREG地址寄存器
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity MAREG is
7     port (CLK : in std_logic;  --时钟信号
8           AI : in std_logic_vector(3 downto 0);
9           IMAR : in std_logic; --寄存命令
10          AO : out std_logic_vector(3 downto 0)
11          );
12 end MAREG;
13
14 architecture vhd_marge of MAREG is
15 signal temp : std_logic_vector(3 downto 0):="0000";
16 begin
17 AO <= temp;
18    process(CLK, IMAR)
19    begin
20       if(rising_edge(CLK)) then  --时钟上升沿
21          if(IMAR = '1') then  --寄存信号有效
22             temp <= AI;
23          end if;
24       end if;
25    end process;
26 end vhd_marge;
```

图 3-3-1 地址寄存器的 VHDL 代码

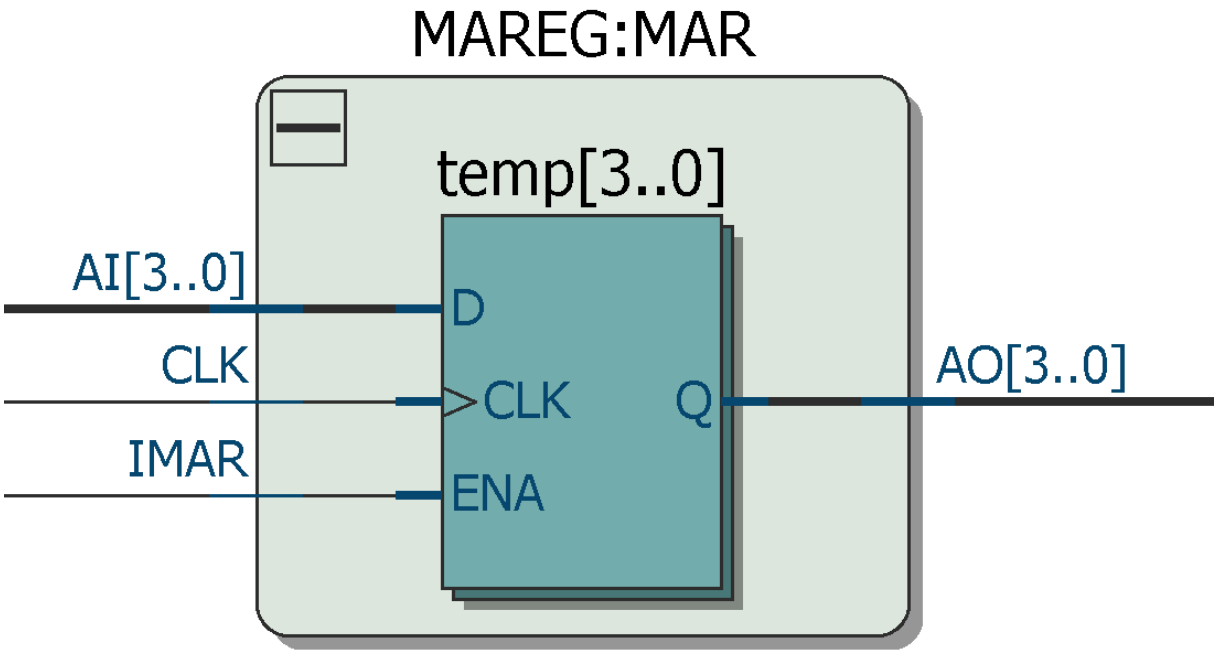

图 3-3-2 地址寄存器的 RTL 图

3.3.2 地址寄存器的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-3-3 所示，运行结果如图 3-3-4。

```
1  --地址寄存器仿真文件
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity ms_mar is
7  end ms_mar;
8
9  architecture modelsim of ms_mar is
10 component MAREG   --地址寄存器
11    port (CLK : in std_logic;  --时钟信号
12          AI : in std_logic_vector(3 downto 0);
13          IMAR : in std_logic; --寄存命令
14          AO : out std_logic_vector(3 downto 0)
15          );
16    end component;
17
18    signal CLK : std_logic; --时钟信号
19    signal AI : std_logic_vector(3 downto 0);
20    signal IMAR : std_logic;   --寄存命令
21    signal AO : std_logic_vector(3 downto 0);
22 begin
23 MAR : MAREG port map(CLK, AI, IMAR, AO);
24 process
25    begin
26       CLK<= '0';
27       wait for 20 ns;
28       CLK<='1';
29       wait for 20 ns;
30    end process;
31 process
32    begin
33       AI<= "1001";
34       wait for 40 ns;
35       AI<= "1010";
36       wait for 40 ns;
37       AI<= "0011";
38       wait for 40 ns;
39       AI<= "1111";
40       wait for 40 ns;
41       AI<= "0101";
42       wait for 40 ns;
43    end process;
44
45 process
46    begin
47       IMAR<= '0';
48       wait for 60 ns;
49       IMAR<='1';
50       wait for 60 ns;
51    end process;
52 end modelsim;
```

图 3-3-3 地址寄存器仿真文件

图 3-3-4 地址寄存器仿真结果

3.4 指令寄存器 IR

指令寄存器的作用是保存并处理 16 位指令数据信息，并通过译码器翻译生成指令代码，输出给控制电路。当时钟信号上升沿来临且 IIR=1 时，指令寄存器接收来自总线 dbus 的 16 位指令数据信息，并保存译码输出相应的指令代码。

3.4.1 指令寄存器 IR 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-4-1 所示，编译成功后得到 RTL Viewer 如图 3-4-2 所示。

```
--IR指令寄存器
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity IREG is
   port (CLK : in std_logic;  --时钟信号
         DI : in std_logic_vector(15 downto 0); --指令数据
         IIR : in std_logic;  --寄存信号
         Code : out std_logic_vector(3 downto 0);  --指令信号
         ERROR,STOP : out std_logic
         );
end IREG;

architecture vhd_ireg of IREG is
signal temp : std_logic_vector(15 downto 0);
begin
   process(CLK, IIR)
   begin
      if(rising_edge(CLK)) then  --时钟信号上升沿
         if(IIR = '1') then   --寄存信号有效
            temp <= DI;
         end if;
      end if;
   end process;

   process(CLK, temp)
   begin
      case temp is   --产生指令信号（相当于一个译码器）
         when "0000000000111110" => Code <= "0001";ERROR<='0';STOP<='0';    --移数信号
         when "0000000011000110" => Code <= "0010";ERROR<='0';STOP<='0';    --加法信号
         when "0001001100011110" => Code <= "0011";ERROR<='0';STOP<='0';    --减法信号
         when "0010110111100111" => Code <= "0100";ERROR<='0';STOP<='0';    --自增信号
         when "0000111100000001" => Code <= "0101";ERROR<='0';STOP<='0';    --自减信号
         when "1011100010000011" => Code <= "0110";ERROR<='0';STOP<='0';    --逻辑与信号
         when "1000011111111110" => Code <= "0111";ERROR<='0';STOP<='0';    --逻辑或信号
         when "0101010101011110" => Code <= "1000";ERROR<='0';STOP<='0';    --逻辑非信号
         when "0111100001010000" => Code <= "1001";ERROR<='0';STOP<='0';    --逻辑异或信号
         when "0000000001110110" => Code <= "1111";ERROR<='0';STOP<='1';    --停机信号
         when others => Code <= "0000";ERROR<='1';STOP<='1'; --异常错误停机
      end case;
   end process;
end vhd_ireg;
```

图 3-4-1 指令寄存器的 VHDL 代码

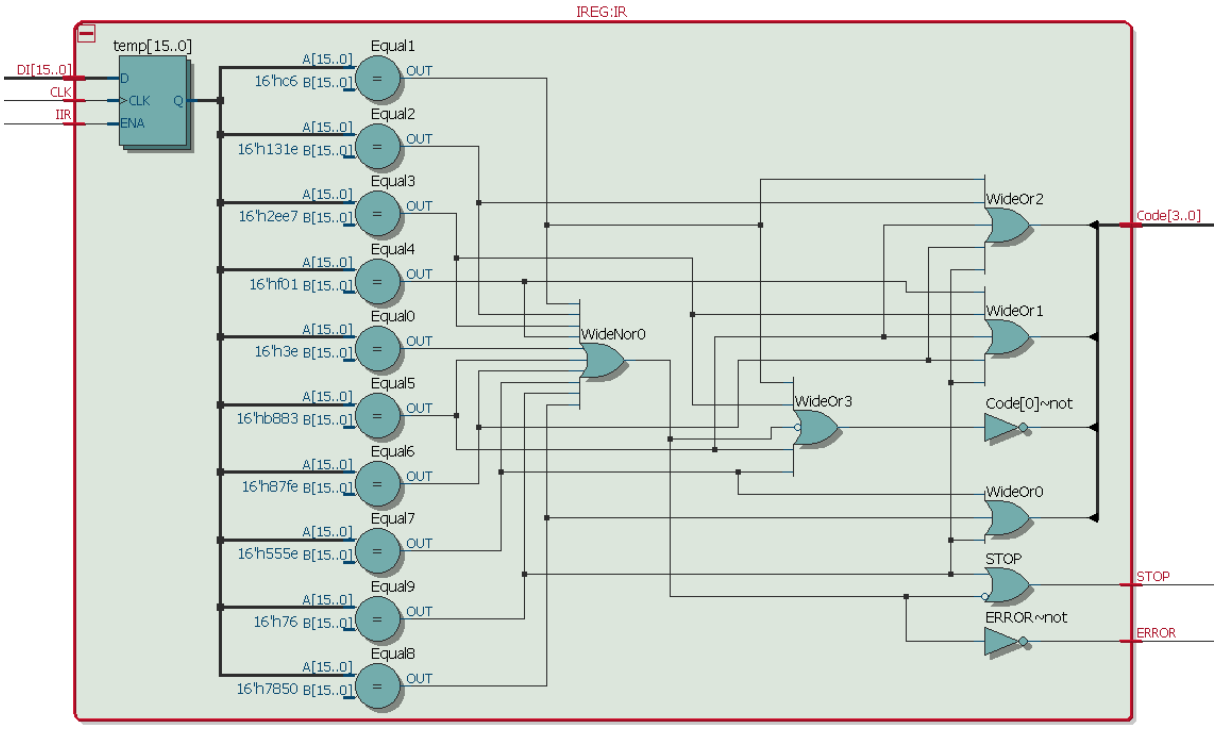

图 3-4-2 指令寄存器的 RTL 图

3.4.2 指令寄存器的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-3-3 所示，运行结果如图 3-4-4。

```
--指令寄存器仿真文件
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ms_ir is
end ms_ir;

architecture modelsim of ms_ir is
component IREG --指令寄存器
   port (CLK : in std_logic;  --时钟信号
         DI : in std_logic_vector(15 downto 0); --指令数据
         IIR : in std_logic;  --寄存信号
         Code : out std_logic_vector(3 downto 0)    --指令信号
         );
   end component;

   signal CLK : std_logic; --时钟信号
   signal DI : std_logic_vector(15 downto 0);    --指令数据
   signal IIR : std_logic; --寄存命令
   signal Code : std_logic_vector(3 downto 0);  --指令信号
begin
IR : IREG port map(CLK, DI, IIR, Code);
process
   begin
      CLK<= '0';
      wait for 20 ns;
      CLK<='1';
      wait for 20 ns;
   end process;
process
   begin
      DI<= "0000000000111110";
      wait for 40 ns;
      DI<= "0000000001110110";
      wait for 40 ns;
      DI<= "0000000011000110";
      wait for 40 ns;
      DI<= "0000000001110110";
      wait for 40 ns;
      DI<= "1111111111111111";
      wait for 40 ns;
      DI<= "0000000000111110";
      wait for 40 ns;
   end process;

process
   begin
      IIR<= '0';
      wait for 60 ns;
      IIR<='1';
      wait for 60 ns;
   end process;
end modelsim;
```

图 3-4-3 指令寄存器仿真文件

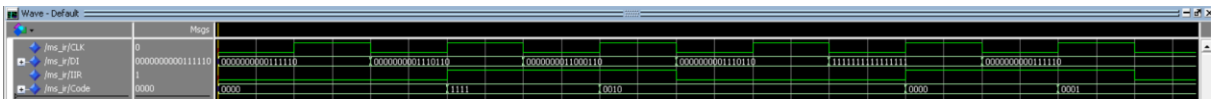

图 3-4-4 指令寄存器仿真结果

3.5 算数逻辑运算单元 ALU

算数逻辑运算单元的作用是按照控制电路的微控制指令对来自累加器 ACC 和总线 dbus 的 16 位数据进行算数逻辑运算，并输出到总线 dbus 中。当 ISUM=1 时，算数逻辑运算单元接收来自累加器 ACC 和总线 dbus 的 16 位数据，依据指令 Code 可执行加法、减法、自增、自减、逻辑与、或、非、异或共 8 中运算。

3.5.1 算数逻辑运算单元 ALU 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-5-1 所示，编译成功后得到 RTL Viewer 如图 3-5-2 所示。

```
--ALU 算数逻辑单元
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
--16位数据加法
entity ALU is
   port (DA,DB : in std_logic_vector(15 downto 0); --输入数据
         Code : in std_logic_vector(3 downto 0);   --输入命令
         ISUM, ESUM : in std_logic; --输入、输出控制
         ALU_OUT : out std_logic_vector(15 downto 0); --ALU输出数据
         Zero, Over, Nege : out std_logic:='0' --标志位
         );
end ALU;

architecture vhd_alu of ALU is
signal DO : std_logic_vector(15 downto 0);
begin
process(DA, DB, ISUM)
variable D1, D2, temp : std_logic_vector(15 downto 0);
begin
   if(ISUM = '1')then
      D1 := DA;
      D2 := DB;
      Zero <= '0';
      Over <= '0';
      Nege <= '0';
      if (Code = "0010") then --加法
         temp := D1 + D2;
         if (temp < D1 or temp < D2) then
            Over <= '1';
         else
            Over <= '0';
         end if;
      elsif (Code = "0011") then --减法
         temp := D1 - D2;
         if (temp > D1) then
            Nege <= '1';
         else
            Nege <= '0';
         end if;
      elsif (Code = "0100") then --自增
         temp := D1 + 1;
         if (temp < D1) then
            Over <= '1';
         else
            Over <= '0';
         end if;
      elsif (Code = "0101") then --自减
         temp := D1 - 1;
         if (temp > D1) then
            Nege <= '1';
         else
            Nege <= '0';
         end if;
      elsif (Code = "0110") then --逻辑与
         temp := D1 and D2;
      elsif (Code = "0111") then --逻辑或
         temp := D1 or D2;
      elsif (Code = "1000") then --逻辑非
         temp := not D1;
      elsif (Code = "1001") then --逻辑异或
         temp := D1 xor D2;
      else
         temp := temp;
      end if;
      if (temp = "0000000000000000") then
         Zero <= '1';
      else
         Zero <= '0';
      end if;
      DO <= temp;
   end if;
end process;
ALU_OUT <= DO when ESUM = '1' else "ZZZZZZZZZZZZZZZZ";   --输出有效 否则高阻态
end vhd_alu;
```

图 3-5-1 算数逻辑运算单元的 VHDL 代码

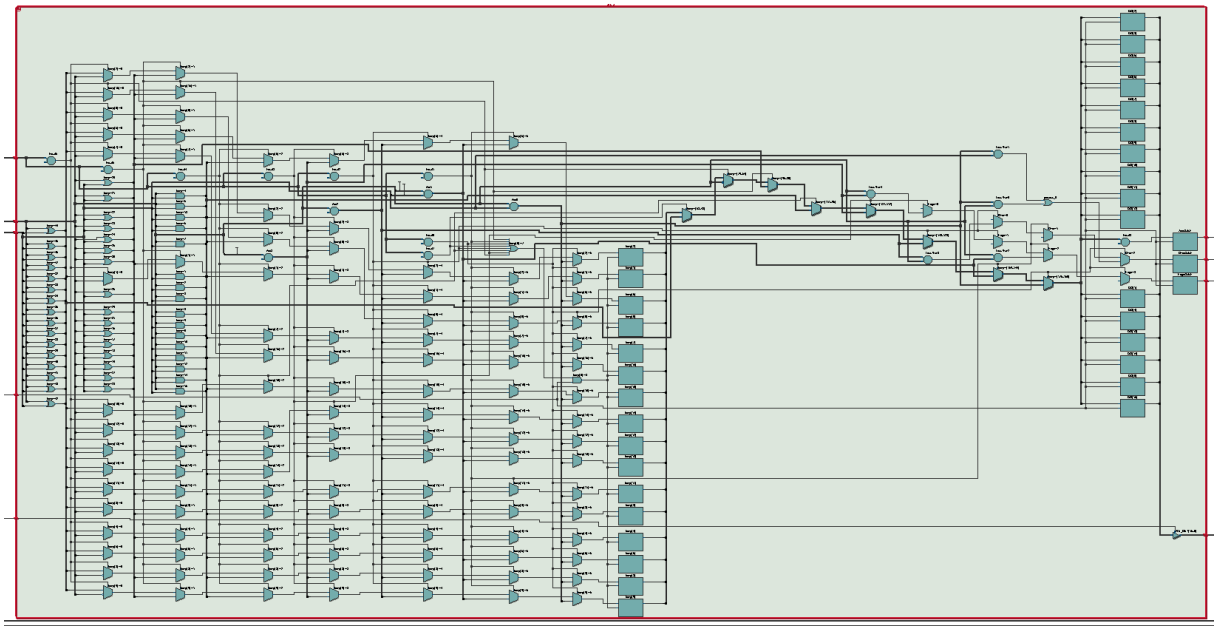

图 3-5-2 算数逻辑运算单元的 RTL 图

3.5.2 算数逻辑运算单元的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-5-3 所示，运行结果如图 3-5-4。

```
--算术逻辑单元仿真文件
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ms_alu is
end ms_alu;

architecture modelsim of ms_alu is
component ALU  --算术逻辑单元
   port (DA,DB : in std_logic_vector(15 downto 0); --输入数据
         Code : in std_logic_vector(3 downto 0);   --输入命令
         ISUM, ESUM : in std_logic; --输入、输出控制
         ALU_OUT : out std_logic_vector(15 downto 0); --ALU输出数据
         Zero, Over, Nege : out std_logic --标志位
         );
   end component;

signal DA,DB : std_logic_vector(15 downto 0);   --输入数据
signal Code : std_logic_vector(3 downto 0):="0000";   --输入命令
signal ISUM,ESUM : std_logic; --输出控制
signal ALU_OUT : std_logic_vector(15 downto 0); --ALU输出数据
signal Zero, Over, Nege : std_logic:='0'; --标志位
begin
A : ALU port map(DA, DB, Code, ISUM, ESUM, ALU_OUT, Zero, Over, Nege);
process
begin
   DA <= "1001110110011111";
   wait for 40 ns;
   DA <= "1000100011110000";
   wait for 40 ns;
   DA <= "0000011100000001";
   wait for 40 ns;
   DA <= "1000000000110000";
   wait for 40 ns;
end process;
process
begin
   DB <= "0000000001111111";
   wait for 60 ns;
   DB <= "1111110000011111";
   wait for 60 ns;
   DB <= "0100011100011111";
   wait for 60 ns;
   DB <= "1010101010101010";
   wait for 60 ns;
end process;
process
begin
   Code <= Code + 1;
   wait for 20 ns;
end process;

process
begin
   ISUM <= '0';
   wait for 10 ns;
   ISUM <= '1';
   wait for 10 ns;
end process;

process
begin
   ESUM <= '0';
   wait for 50 ns;
   ESUM <= '1';
   wait for 50 ns;
end process;

end modelsim;
```

图 3-5-3 算数逻辑运算单元仿真文件

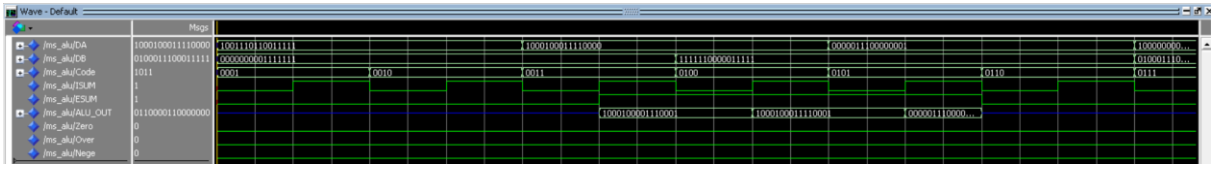

图 3-5-4 算数逻辑运算单元仿真结果

3.6 数据寄存器 DR

数据寄存器的作用是保存 16 位数据信息。当 IDR=1 时接收输入 16 位数据，当 EDR=1 时，输出内部保存的 16 位数据。此结构应用于 DR 数据寄存器与 ACC 累加器两个模块。

3.6.1 数据寄存器 DR 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-6-1 所示，编译成功后得到 RTL Viewer 如图 3-6-2 所示。

```
1  --DREG16位累加器/数据寄存器
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity DREG is
7     port (CLK : in std_logic;  --时钟信号
8           DI : in std_logic_vector(15 downto 0); --输入数据
9           W : in std_logic; --写命令
10          R : in std_logic; --读命令
11          DO : out std_logic_vector(15 downto 0) --输出数据
12          );
13 end DREG;
14
15 architecture vhd_dreg of DREG is
16 signal temp : std_logic_vector(15 downto 0):="ZZZZZZZZZZZZZZZZ";
17 begin
18    process(CLK, W, R)
19    begin
20       if(rising_edge(CLK))then    --时钟上升沿
21          if(W='1') then --写入
22             temp <= DI;
23          end if;
24       end if;
25    end process;
26 DO <= temp when R='1' else "ZZZZZZZZZZZZZZZZ";  --读命令 否则高阻态
27 end vhd_dreg;
```

图 3-6-1 数据寄存器的 VHDL 代码

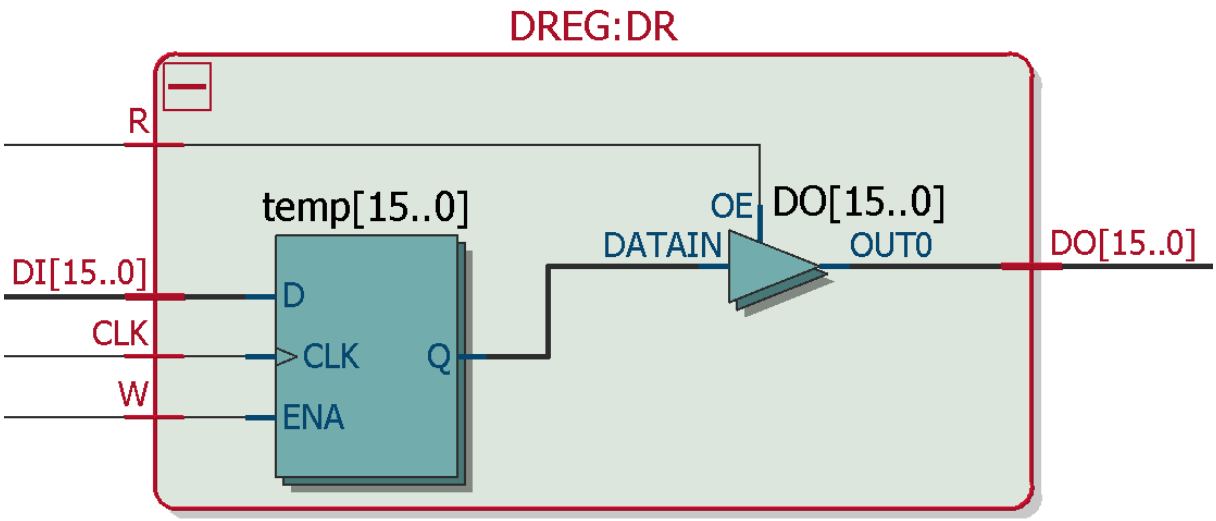

图 3-6-2 数据寄存器的 RTL 图

3.6.2 数据寄存器的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-6-3 所示，运行结果如图 3-6-4。

```
--数据寄存器仿真文件
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ms_dr is
end ms_dr;

architecture modelsim of ms_dr is
component DREG --数据寄存器
   port (CLK : in std_logic;  --时钟信号
         DI : in std_logic_vector(15 downto 0); --输入数据
         W : in std_logic; --写命令
         R : in std_logic; --读命令
         DO : out std_logic_vector(15 downto 0) --输出数据
         );
   end component;

   signal CLK : std_logic; --时钟信号
   signal DI : std_logic_vector(15 downto 0);   --输入数据
   signal W : std_logic;   --写命令
   signal R : std_logic;   --读命令
   signal DO : std_logic_vector(15 downto 0);   --输出数据
begin
DR : DREG port map(CLK, DI, W, R, DO);
process
   begin
      CLK<= '0';
      wait for 20 ns;
      CLK<='1';
      wait for 20 ns;
   end process;
process
   begin
      DI<= "1000111001111111";
      wait for 40 ns;
      DI<= "1001000000011000";
      wait for 40 ns;
      DI<= "1111111100000000";
      wait for 40 ns;
      DI<= "1000011110000000";
      wait for 40 ns;
      DI<= "0100001111000011";
      wait for 40 ns;
   end process;
process
   begin
      W<= '0';
      wait for 60 ns;
      W<='1';
      wait for 60 ns;
   end process;
process
   begin
      R<= '0';
      wait for 120 ns;
      R<='1';
      wait for 120 ns;
   end process;
end modelsim;
```

图 3-6-3 数据寄存器仿真文件

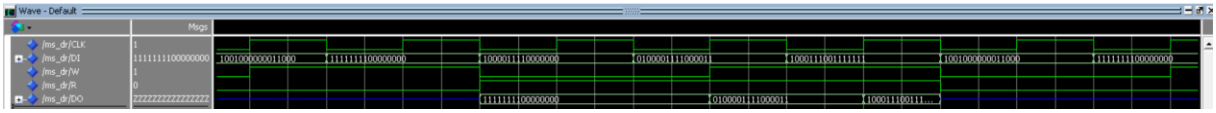

图 3-6-4 数据寄存器仿真结果

3.7 控制电路 CTRL

控制电路的作用是接收来自指令寄存器的指令代码并根据节拍信号生成为控制信号控制模型机各个模块工作。

3.7.1 控制电路 CTRL 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-7-1 所示，编译成功后得到 RTL Viewer 如图 3-7-2 所示。

```
--CTRL控制电路
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity CTRL is
   port (CLK : in std_logic;  --时钟信号
         Code : in std_logic_vector(3 downto 0);   --指令信号   from IREG
         T : in std_logic_vector(7 downto 0);   --节拍信号   from COUNTER

         IPC : out std_logic:='0';  --PC控制信号
         IMAR : out std_logic:='0'; --MAREG寄存信号
         IDR, EDR : out std_logic:='0';   --DREG寄存、输出信号
         W, R : out std_logic:='0'; --ALU中DREG输入输出信号
         Code_A : out std_logic_vector(3 downto 0);
         ISUM, ESUM : out std_logic:='0'; --ALU控制、输出信号
         IIR : out std_logic:='0'   --IREG寄存信号
         );
end CTRL;

architecture vhd_ctrl of CTRL is
signal Code1,Code2,Code3,Code4,Code5,Code6,Code7,Code8,Code9,Code15 : std_logic:='0';  --命令信号中间变量
signal T0, T1, T2, T3, T4, T5, T6, T7 : std_logic:='0';  --节拍信号中间变量
signal C1, C2, C3 : std_logic:='0'; --指令分类
begin
--命令信号保存-------------------------------------------------------------------------
Code1 <= (not Code(3)) and (not Code(2)) and (not Code(1)) and (Code(0));  --0001
Code2 <= (not Code(3)) and (not Code(2)) and (Code(1)) and (not Code(0));  --0010
Code3 <= (not Code(3)) and (not Code(2)) and (Code(1)) and (Code(0));   --0011
Code4 <= (not Code(3)) and (Code(2)) and (not Code(1)) and (not Code(0));  --0100
Code5 <= (not Code(3)) and (Code(2)) and (not Code(1)) and (Code(0));   --0101
Code6 <= (not Code(3)) and (Code(2)) and (Code(1)) and (not Code(0));   --0110
Code7 <= (not Code(3)) and (Code(2)) and (Code(1)) and (Code(0)); --0111
Code8 <= (Code(3)) and (not Code(2)) and (not Code(1)) and (not Code(0));  --1000
Code9 <= (Code(3)) and (not Code(2)) and (not Code(1)) and (Code(0));   --1001
Code15 <= (Code(3)) and (Code(2)) and (Code(1)) and (Code(0)); --1111

C1 <= Code1;
C2 <= Code2 or Code3 or Code6 or Code7 or Code9;
C3 <= Code4 or Code5  or Code8;
--节拍信号保存-------------------------------------------------------------------------
T0 <= (not T(7)) and (not T(6)) and (not T(5)) and (not T(4)) and (not T(3)) and (not T(2)) and (not T(1)) and (T(0));  --00000001
T1 <= (not T(7)) and (not T(6)) and (not T(5)) and (not T(4)) and (not T(3)) and (not T(2)) and (T(1)) and (not T(0));  --00000010
T2 <= (not T(7)) and (not T(6)) and (not T(5)) and (not T(4)) and (not T(3)) and (T(2)) and (not T(1)) and (not T(0));  --00000100
T3 <= (not T(7)) and (not T(6)) and (not T(5)) and (not T(4)) and (T(3)) and (not T(2)) and (not T(1)) and (not T(0));  --00001000
T4 <= (not T(7)) and (not T(6)) and (not T(5)) and (T(4)) and (not T(3)) and (not T(2)) and (not T(1)) and (not T(0));  --00010000
T5 <= (not T(7)) and (not T(6)) and (T(5)) and (not T(4)) and (not T(3)) and (not T(2)) and (not T(1)) and (not T(0));  --00100000
T6 <= (not T(7)) and (T(6)) and (not T(5)) and (not T(4)) and (not T(3)) and (not T(2)) and (not T(1)) and (not T(0));  --01000000
T7 <= (T(7)) and (not T(6)) and (not T(5)) and (not T(4)) and (not T(3)) and (not T(2)) and (not T(1)) and (not T(0));  --10000000
--微控制指令-------------------------------------------------------------------------
IPC <= (not Code15) and (T2 or ((C1 and T5) or (C2 and T5)));  --PC控制信号
IMAR <= (not Code15) and (T0 or ((C1 and T3) or (C2 and T3)));--MAREG寄存信号
IDR <= (not Code15) and (T1 or ((C1 and T4) or (C2 and T4)));  --DREG寄存信号
EDR <= (not Code15) and (T1 or T2 or (C1 and T4) or (C1 and T5) or (C1 and T6) or (C2 and T4) or (C2 and T5) or (C3 and T3)); --DREG输出信号
W <= (not Code15) and ((C1 and T6) or (C2 and T6) or (C3 and T4));    --ALU中DREG输入信号
R <= (not Code15) and ((C2 and T5) or (C3 and T3));   --ALU中DREG输出信号
Code_A <= Code;   --ALU操作指令
ISUM <= (not Code15) and ((C2 and T5) or (C3 and T3));      --ALU控制信号
ESUM <= (not Code15) and ((C2 and T6) or (C3 and T4));   --ALU输出信号
IIR <= (not Code15) and (T2);--IREG寄存信号
end vhd_ctrl;
```

图 3-7-1 控制电路的 VHDL 代码

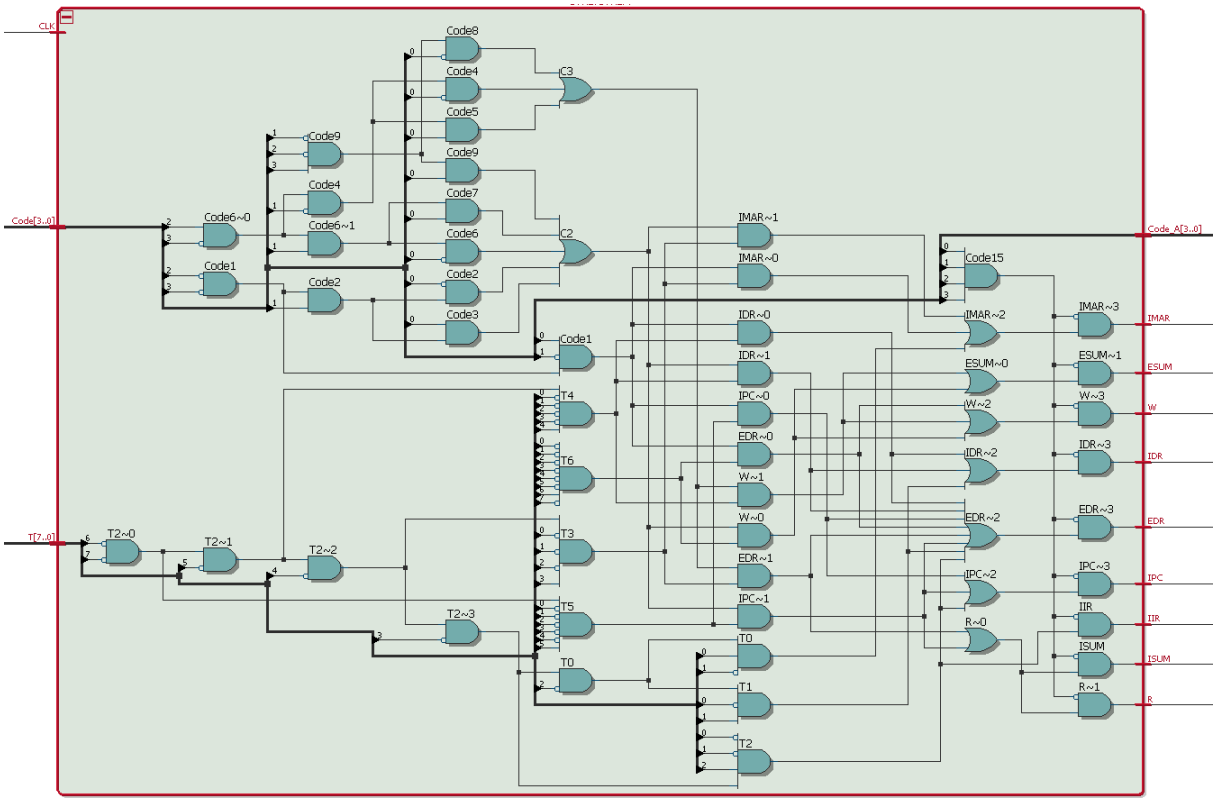

图 3-7-2 控制电路的 RTL 图

3.7.2 控制电路的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-7-3 所示，运行结果如图 3-7-4。

```
--CTRL控制电路仿真文件
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ms_ctrl is
end ms_ctrl;

architecture modelsim of ms_ctrl is
component CTRL --程序计数器
   port (CLK : in std_logic;  --时钟信号
         Code : in std_logic_vector(3 downto 0);   --指令信号   from IREG
         T : in std_logic_vector(7 downto 0);   --节拍信号   from COUNTER
         IPC : out std_logic; --PC控制信号
         IMAR : out std_logic;   --MAREG寄存信号
         IDR, EDR : out std_logic;  --DREG寄存、输出信号
         W, R : out std_logic;   --ALU中DREG输入输出信号
         Code_A : out std_logic_vector(3 downto 0);
         ISUM, ESUM : out std_logic;   --ALU控制、输出信号
         IIR : out std_logic  --IREG寄存信号
         );
   end component;

   signal CLK : std_logic; --时钟信号
   signal Code : std_logic_vector(3 downto 0);  --指令信号   from IREG
   signal T : std_logic_vector(7 downto 0);  --节拍信号   from COUNTER
   signal IPC : std_logic; --PC控制信号
   signal IMAR : std_logic;   --MAREG寄存信号
   signal IDR, EDR : std_logic;  --DREG寄存、输出信号
   signal W, R : std_logic;   --ALU中DREG输入输出信号
   signal Code_A : std_logic_vector(3 downto 0);
   signal ISUM, ESUM : std_logic;   --ALU控制、输出信号
   signal IIR : std_logic; --IREG寄存信号
begin
CL : CTRL port map(CLK, Code, T, IPC, IMAR, IDR, EDR, W, R, Code_A, ISUM, ESUM, IIR);

process
   begin
      CLK <= '0';
      wait for 20 ns;
      CLK <= '1';
      wait for 20 ns;
   end process;
process
   begin
      T <= "00000001";
      wait for 40 ns;
      T <= "00000010";
      wait for 40 ns;
      T <= "00000100";
      wait for 40 ns;
      T <= "00001000";
      wait for 40 ns;
      T <= "00010000";
      wait for 40 ns;
      T <= "00100000";
      wait for 40 ns;
      T <= "01000000";
      wait for 40 ns;
      T <= "10000000";
      wait for 40 ns;
   end process;


process
   begin
      Code <= "0001";
      wait for 320 ns;
      Code <= "0010";
      wait for 320 ns;
      Code <= "0011";
      wait for 320 ns;
   end process;


end modelsim;
```

图 3-7-3 控制电路仿真文件

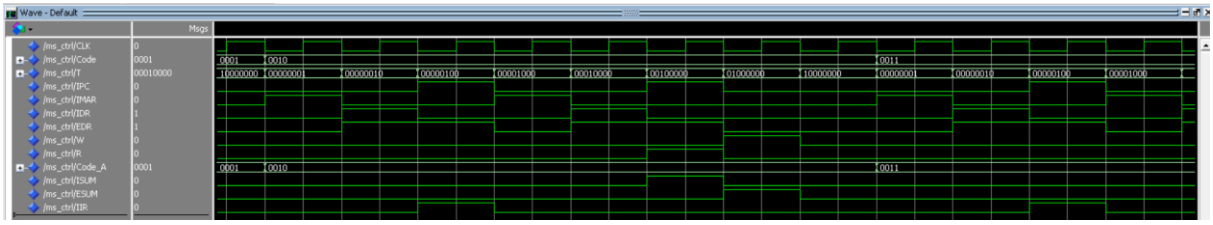

图 3-7-4 控制电路仿真结果

3.8 时钟信号产生器 CLOCK

时钟信号产生器的作用对系统时钟信号进行分频输出，调节模型机内部时钟信号，控制指令操作速度。

3.8.1 时钟信号产生器 CLOCK 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-8-1 所示，编译成功后得到 RTL Viewer 如图 3-8-2 所示。

```
1  --CLOCK时钟信号产生器
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity CLOCK is
7     port (CLK : in std_logic;  --50MHz T=20ns 5e7T= 1s
8           CLK_OUT : out std_logic
9           );
10 end CLOCK;
11
12 architecture vhd of CLOCK is
13 signal temp : std_logic:='0';
14 begin
15 CLK_OUT <= temp;
16    process(CLK)
17    variable T : integer := 10;    --25000000
18    variable counter : integer range 0 to T;
19    begin
20       if(counter = T) then
21          counter := 0;
22          temp <= not temp;
23       elsif(rising_edge(CLK))then
24          counter := counter + 1;
25       end if;
26    end process;
27 end;
```

图 3-8-1 时钟信号产生器的 VHDL 代码

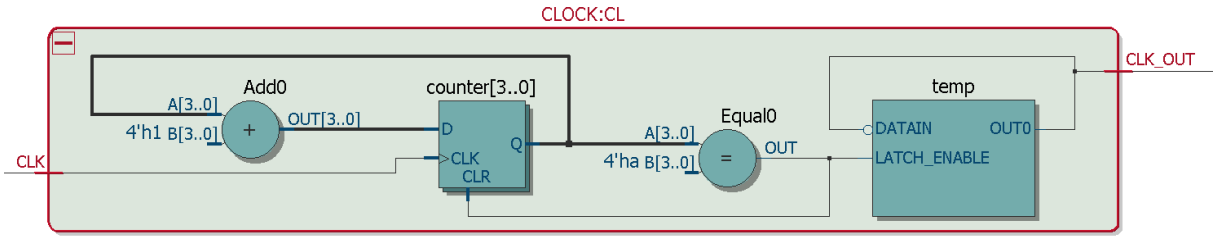

图 3-8-2 时钟信号产生器的 RTL 图

3.8.2 时钟信号产生器的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-8-3 所示，运行结果如图 3-8-4。

```
1  --时钟信号产生器仿真文件
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity ms_clk is
7  end ms_clk;
8
9  architecture modelsim of ms_clk is
10 component CLOCK    --时钟信号产生器
11    port (CLK : in std_logic;  --50MHz T=20ns 5e7T= 1s
12          CLK_OUT : out std_logic
13          );
14    end component;
15
16    signal CLK : std_logic; --时钟信号
17    signal CLK_OUT : std_logic;
18 begin
19 CL : CLOCK port map(CLK, CLK_OUT);
20 process
21    begin
22       CLK<= '0';
23       wait for 20 ns;
24       CLK<='1';
25       wait for 20 ns;
26    end process;
27 end modelsim;
```

图 3-8-3 时钟信号产生器仿真文件

图 3-8-4 时钟信号产生器仿真结果

3.9 程序计数器 PC

程序计数器的作用控制指令操作进程，输出地址数据信息。当时钟信号上升沿来临时，若 CLR=1 则程序计数器 PC 恢复初始状态，地址信息恢复为 0000。当 CLR=0 且 IPC=1 时程序计数器执行加一动作，否则状态不改变。

3.9.1 程序计数器 PC 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-9-1 所示，编译成功后得到 RTL Viewer 如图 3-9-2 所示。

```vhdl
--PC程序计数器
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity PC is
    port (CLK : in std_logic;  --时钟信号
          CLR : in std_logic;
          IPC : in std_logic;  --控制信号
          PC_OUT : out std_logic_vector(3 downto 0)
          );
end PC;

architecture vhd_pc of PC is
signal temp : std_logic_vector(3 downto 0);
begin
    process(CLK, CLR, IPC)
    begin
       if(CLR = '1') then    --清零
          temp <= "0000";
       elsif(rising_edge(CLK)) then  --时钟上升沿
          if(IPC = '1')then --控制信号有效
             temp <= temp + 1; --PC + 1
          end if;
       end if;
    end process;
PC_OUT <= temp;
end vhd_pc;
```

图 3-9-1 程序计数器的 VHDL 代码

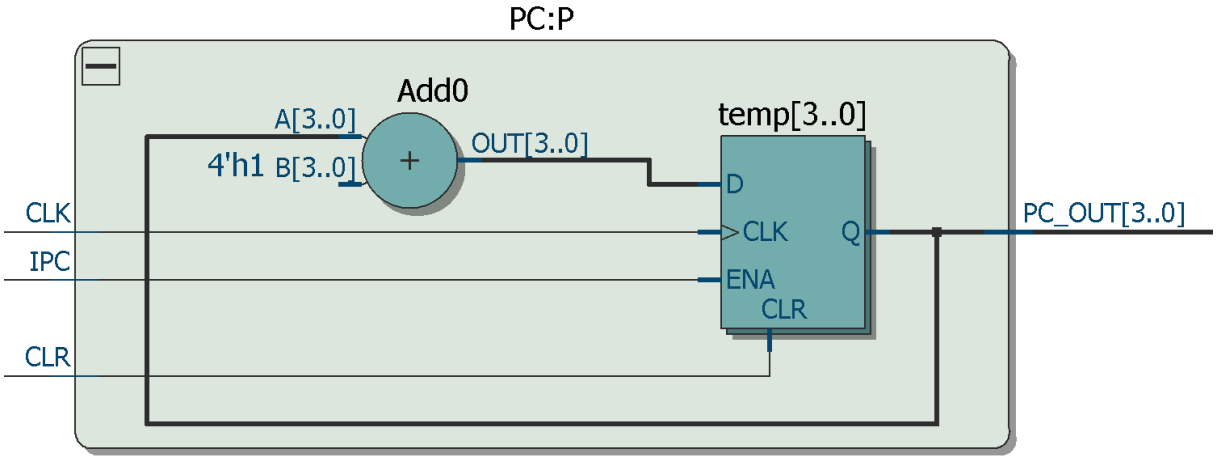

图 3-9-2 程序计数器的 RTL 图

3.9.2 程序计数器的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-9-3 所示，运行结果如图 3-9-4。

```
1  --程序计数器仿真文件
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity ms_pc is
7  end ms_pc;
8
9  architecture modelsim of ms_pc is
10 component PC   --程序计数器
11    port (CLK : in std_logic;  --时钟信号
12          CLR : in std_logic;
13          IPC : in std_logic;  --控制信号
14          PC_OUT : out std_logic_vector(3 downto 0)
15          );
16    end component;
17
18    signal CLK : std_logic; --时钟信号
19    signal CLR : std_logic;
20    signal IPC : std_logic; --控制信号
21    signal PC_OUT : std_logic_vector(3 downto 0);
22 begin
23 P : PC port map(CLK, CLR, IPC, PC_OUT);
24 process
25    begin
26       CLK<= '0';
27       wait for 20 ns;
28       CLK<='1';
29       wait for 20 ns;
30    end process;
31
32 process
33    begin
34       CLR<= '1';
35       wait for 40 ns;
36       CLR<='0';
37       wait;
38    end process;
39
40 process
41    begin
42       IPC<= '0';
43       wait for 60 ns;
44       IPC<='1';
45       wait for 60 ns;
46    end process;
47 end modelsim;
```

图 3-9-3 程序计数器仿真文件

图 3-9-4 程序计数器仿真结果

3.10 存储器 RAM

存储器的作用是保存 16 位数据信息，存储空间为 16×16 位。当 CS=1 是存储器处于工作状态，此时若 WR=0，则为写状态，输入 16 位信息保存于 4 位地址信息所指空间内；若 WR=1.则为读状态，输出保存于 4 位地址信息所指空间内 16 位数据信息。

3.10.1 存储器 RAM 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-10-1 所示，编译成功后得到 RTL Viewer 如图 3-10-2 所示。

```
1  --RAM存储器
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity RAM is
7     port (CS : in std_logic;    --工作使能端
8           WR : in std_logic;    --读写使能端
9           Add_IN : in std_logic_vector(3 downto 0);
10          Address : in std_logic_vector(3 downto 0);
11          DI : in std_logic_vector(15 downto 0);
12          DO : out std_logic_vector(15 downto 0)
13          );
14 end RAM;
15 --16 * 16 bit
16 architecture vhd_ram of RAM is
17 TYPE matrix_index is array (15 downto 0) of std_logic_vector(15 downto 0);
18 begin
19    process(CS, WR, Add_IN, Address, DI)
20    variable Data : matrix_index;
21    begin
22       if(CS='1') then    --使能端有效
23          if(WR = '0') then --写有效
24             Data(conv_integer(ADD_IN(3 downto 0))) := DI;
25          elsif(WR = '1') then --读有效
26             DO <= Data(conv_integer(Address(3 downto 0)));
27          end if;
28       end if;
29    end process;
30 end vhd_ram;
```

图 3-10-1 存储器的 VHDL 代码

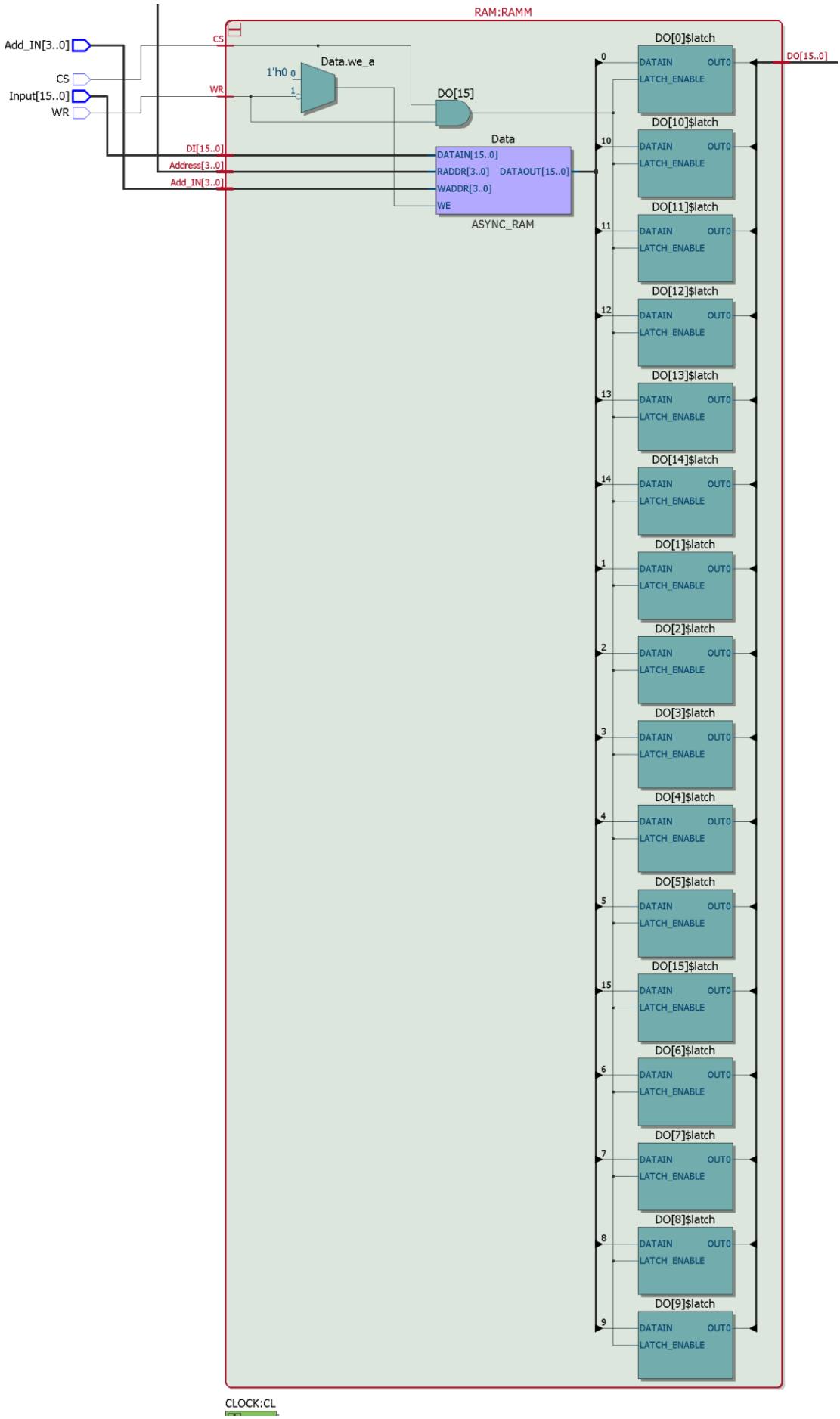
RAM:RAMM
Add_IN[3..0]
CS
Input[15..0]
WR
CS
WR
DI[15..0]
Address[3..0]
Add_IN[3..0]
1'h0
Data.we_a
DO[15]
Data
DATAIN[15..0]
RADDR[3..0]
DATAOUT[15..0]
WADDR[3..0]
WE
ASYNC_RAM
DO[0]$latch
DO[10]$latch
DO[11]$latch
DO[12]$latch
DO[13]$latch
DO[14]$latch
DO[1]$latch
DO[2]$latch
DO[3]$latch
DO[4]$latch
DO[5]$latch
DO[15]$latch
DO[6]$latch
DO[7]$latch
DO[8]$latch
DO[9]$latch
DATAIN
OUT0
LATCH_ENABLE
DO[15..0]
CLOCK:CL

图 3-10-2 存储器的 RTL 图

3.10.2 存储器的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-10-3 所示，运行结果如图 3-10-4。

```
--RAM存储器仿真文件
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ms_ram is
end ms_ram;

architecture modelsim of ms_ram is
component RAM  --程序计数器
   port (CS : in std_logic;   --工作使能端
         WR : in std_logic;   --读写使能端
         Add_IN : in std_logic_vector(3 downto 0);
         Address : in std_logic_vector(3 downto 0);
         DI : in std_logic_vector(15 downto 0);
         DO : out std_logic_vector(15 downto 0)
         );
   end component;

   signal CS : std_logic;  --时钟信号
   signal WR : std_logic;  --读写使能端
   signal Add_IN : std_logic_vector(3 downto 0);
   signal Address : std_logic_vector(3 downto 0);
   signal DI : std_logic_vector(15 downto 0);
   signal DO : std_logic_vector(15 downto 0);
begin
R : RAM port map(CS, WR, Add_IN, Address, DI, DO);
process
   begin
      CS<= '1';
      wait;
   end process;

process
   begin
      WR<= '0';
      wait for 100 ns;
      WR<='1';
      wait for 100 ns;
   end process;
process
   begin
      Address <= "0000";
      wait for 20 ns;
      Address <= "0001";
      wait for 20 ns;
      Address <= "0010";
      wait for 20 ns;
      Address <= "0011";
      wait for 20 ns;
      Address <= "0100";
      wait for 20 ns;
   end process;
process
   begin
      Add_IN<="0000";DI<="0100010011111001";
      wait for 20 ns;
      Add_IN<="0001";DI<="1000111000011110";
      wait for 20 ns;
      Add_IN<="0010";DI<="0000000011111110";
      wait for 20 ns;
      Add_IN<="0011";DI<="1111100111110001";
      wait for 20 ns;
      Add_IN<="0100";DI<="0000001111110000";
      wait for 20 ns;
   end process;
end modelsim;
```

图 3-10-3 存储器仿真文件

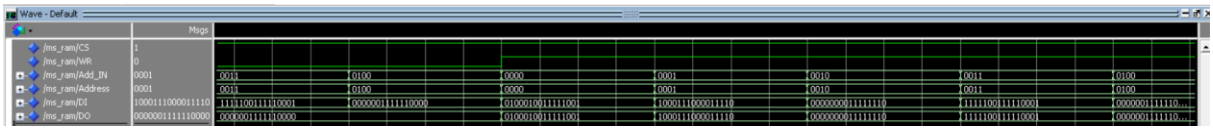

图 3-10-4 存储器仿真结果

3.11 总线 dbus

总线的作用接收保存来自于数据寄存器 DR 与算数逻辑运算单元 ALU 的 16 位数据，并输出。

3.11.1 总线 dbus 的 VHDL 设计

用 VHDL 语言实现，具体代码如图 3-11-1 所示，编译成功后得到 RTL Viewer 如图 3-11-2 所示。

```
1  --dbus
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5  entity dbus is
6     port (D_ALU : in std_logic_vector(15 downto 0);
7           D_DR : in std_logic_vector(15 downto 0);
8           D_OUT : out std_logic_vector(15 downto 0)
9           );
10 end dbus;
11
12 architecture vhd of dbus is
13 signal temp : std_logic_vector(15 downto 0):="ZZZZZZZZZZZZZZZZ";
14 begin
15    process(D_ALU, D_DR)
16    begin
17       if not (D_ALU = "ZZZZZZZZZZZZZZZZ") then
18          temp <= D_ALU;
19       else
20          temp <= D_DR;
21       end if;
22    end process;
23 D_OUT <= temp;
24 end;
```

图 3-11-1 总线的 VHDL 代码

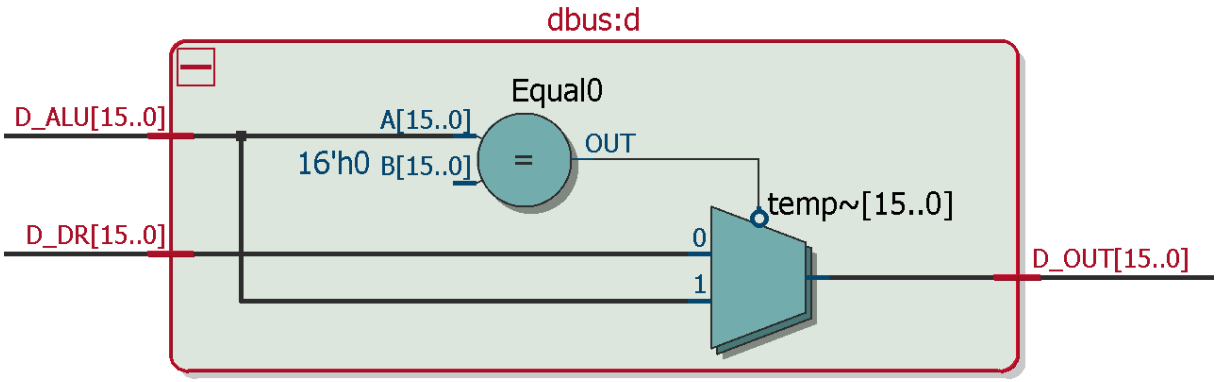

图 3-11-2 总线的 RTL 图

3.11.2 总线的仿真

要用 ModelSim 实现节拍发生器脉冲波形图，编写的测试向量如图 3-11-3 所示，运行结果如图 3-11-4。

```
1  --dbus仿真文件
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity ms_dbus is
7  end ms_dbus;
8
9  architecture modelsim of ms_dbus is
10 component dbus --程序计数器
11    port (D_ALU : in std_logic_vector(15 downto 0);
12          D_DR : in std_logic_vector(15 downto 0);
13          D_OUT : out std_logic_vector(15 downto 0)
14          );
15    end component;
16
17    signal D_ALU : std_logic_vector(15 downto 0);
18    signal D_DR : std_logic_vector(15 downto 0);
19    signal D_OUT : std_logic_vector(15 downto 0);
20 begin
21 d : dbus port map(D_ALU, D_DR, D_OUT);
22 process
23    begin
24       D_ALU <= "ZZZZZZZZZZZZZZZ";
25       wait for 30 ns;
26       D_ALU <= "1001100011100010";
27       wait for 30 ns;
28       D_ALU <= "1000100000000000";
29       wait for 30 ns;
30       D_ALU <= "ZZZZZZZZZZZZZZZ";
31       wait for 30 ns;
32       D_ALU <= "1011110111100110";
33       wait for 30 ns;
34       D_ALU <= "1111111110000000";
35       wait for 30 ns;
36    end process;
37 process
38    begin
39       D_DR <= "ZZZZZZZZZZZZZZZ";
40       wait for 80 ns;
41       D_DR <= "1111111111111111";
42       wait for 80 ns;
43       D_DR <= "0000000000000000";
44       wait for 80 ns;
45       D_DR <= "ZZZZZZZZZZZZZZZ";
46       wait for 80 ns;
47       D_DR <= "ZZZZZZZZZZZZZZZ";
48       wait for 80 ns;
49       D_DR <= "0101010101010111";
50       wait for 80 ns;
51    end process;
52 end modelsim;
```

图 3-11-3 总线仿真文件

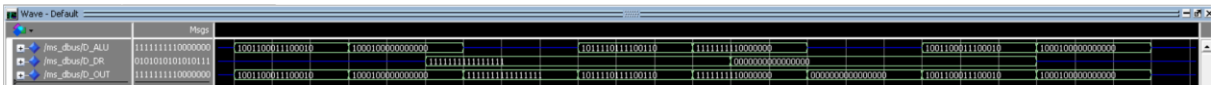

图 3-11-4 总线仿真结果

第 4 章 系统 VHDL 设计与仿真

4.1 顶层模块设计

16 位模型机由 CPU、存储器两大模块组成。CPU 包含时间信号产生器、节拍信号发生器、程序计数器、地址寄存器、数据寄存器、指令寄存器、控制电路、累加器、算数逻辑运算单元共 9 部分组成。程序计数器记录程序运行进度，输出 4 位地址信号，通过地址寄存器访问存储器。存储器根据地址信号查找 16 位数据信息输出至数据寄存器并输入至内部总线。指令寄存器获取内部总线中的指令代码信息，经过内部译码器翻译生成指令信号。节拍信号发生器接收时间信号，循环生成节拍信号。控制电路接收节拍信号与指令信号生成微指令信号，控制模型机各个模块。算数逻辑单元接收累加器与内部总线中的 16 位数据，执行相应的运算动作，输出运算结果。

```
--16位模型机
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Sixteen_Bit_CPU is
    port (CLK : in std_logic;  --时钟信号
          CLR : in std_logic;  --复位信号
          CS, WR : in std_logic;  --RAM的控制信号
          Add_IN : in std_logic_vector(3 downto 0); --RAM输入数据存放地址
          Input : in std_logic_vector(15 downto 0); --RAM输入数据
          ALU_OUT : out std_logic_vector(15 downto 0); --ALU输出数据
          Zero, Over, Nege : out std_logic; --标志位信号
          ERROR,STOP : out std_logic;

          Tap : out std_logic_vector(7 downto 0);    --节拍信号
          Code_IR : out std_logic_vector(3 downto 0);  --IR输出指令
          IPC_O : out std_logic;  --PC控制信号
          IMAR_O : out std_logic; --MAR寄存信号
          IDR_O, EDR_O : out std_logic; --DR寄存、输出信号
          W_O, R_O : out std_logic;  --ALU中DR输入输出信号
          ISUM_O : out std_logic; --ALU控制信号
          ESUM_O : out std_logic; --ALU输出信号
          IIR_O : out std_logic;  --IREG寄存信号
          ABUS_O : out std_logic_vector(3 downto 0);    --地址总线
          dbus_O : out std_logic_vector(15 downto 0)    --内部总线
          );
end Sixteen_Bit_CPU;

architecture vhd of Sixteen_Bit_CPU is
component MAREG   --地址寄存器
    port (CLK : in std_logic;  --时钟信号
          AI : in std_logic_vector(3 downto 0);
          IMAR : in std_logic; --寄存命令
          AO : out std_logic_vector(3 downto 0)
          );
end component;

component IREG --指令寄存器
    port (CLK : in std_logic;  --时钟信号
          DI : in std_logic_vector(15 downto 0); --指令数据
          IIR : in std_logic;  --寄存信号
          Code : out std_logic_vector(3 downto 0);  --指令信号
          ERROR,STOP : out std_logic
          );
end component;

component COUNTER --节拍发生器
    port (CLK : in std_logic;  --时钟信号
          CLR : in std_logic;  --初始
          T : out std_logic_vector(7 downto 0)
          );
end component;

component ALU  --算术逻辑单元
    port (DA,DB : in std_logic_vector(15 downto 0); --输入数据
          Code : in std_logic_vector(3 downto 0);    --输入命令
          ISUM, ESUM : in std_logic; --输入、输出控制
          ALU_OUT : out std_logic_vector(15 downto 0); --ALU输出数据
          Zero, Over, Nege : out std_logic --标志位
          );
    end component;

component DREG --数据寄存器
    port (CLK : in std_logic;  --时钟信号
          DI : in std_logic_vector(15 downto 0); --输入数据
          W : in std_logic; --写命令
          R : in std_logic; --读命令
          DO : out std_logic_vector(15 downto 0) --输出数据
          );
    end component;

component CLOCK   --时钟信号产生器
    port (CLK : in std_logic;  --50MHz T=20ns 5e7T= 1s
          CLK_OUT : out std_logic
          );
    end component;

component PC   --程序计数器
    port (CLK : in std_logic;  --时钟信号
          CLR : in std_logic;
          IPC : in std_logic;  --控制信号
          PC_OUT : out std_logic_vector(3 downto 0)
          );
    end component;
```

图 4-2-1 顶层模块 VHDL 代码(1)

```
component CTRL --控制电路
   port (CLK : in std_logic;  --时钟信号
         Code : in std_logic_vector(3 downto 0);   --指令信号   from IREG
         T : in std_logic_vector(7 downto 0);   --节拍信号   from COUNTER
         IPC : out std_logic; --PC控制信号
         IMAR : out std_logic;   --MAREG寄存信号
         IDR, EDR : out std_logic;  --DREG寄存、输出信号
         W, R : out std_logic;   --ALU中DREG输入输出信号
         Code_A : out std_logic_vector(3 downto 0);
         ISUM : out std_logic;   --ALU控制信号
         ESUM : out std_logic;   --ALU输出信号
         IIR : out std_logic  --IREG寄存信号
         );
end component;

component RAM  --存储器
   port (CS : in std_logic;   --工作使能端
         WR : in std_logic;   --读写使能端
         Add_IN : in std_logic_vector(3 downto 0);
         Address : in std_logic_vector(3 downto 0);
         DI : in std_logic_vector(15 downto 0);
         DO : out std_logic_vector(15 downto 0)
         );
end component;

component dbus --内部数据总线
   port (D_ALU : in std_logic_vector(15 downto 0);
         D_DR : in std_logic_vector(15 downto 0);
         D_OUT : out std_logic_vector(15 downto 0)
         );
end component;

signal CLK_Used : std_logic;  --时钟信号   CL时钟信号产生器 --> 各个电路
signal T : std_logic_vector(7 downto 0);  --节拍信号   CT节拍发生器 --> CTRLM控制电路
signal Code : std_logic_vector(3 downto 0);  --指令信号   IR指令寄存器 --> CTRLM控制电路
signal Code_A : std_logic_vector(3 downto 0);   --指令信号   CTRLM控制电路 --> ALU
signal IPC : std_logic; --PC信号  CTRLM控制电路 --> PC程序计数器
signal IMAR : std_logic;   --MAR寄存信号   CTRLM控制电路 --> MAR地址寄存器
signal IDR, EDR : std_logic;  --DR寄存、输出信号  CTRLM控制电路 --> DR数据寄存器
signal W, R : std_logic;   --ALU中DR输入输出信号  CTRLM控制电路 --> ALU数据寄存器
signal ISUM : std_logic;   --ALU指令信号   CTRLM控制电路 --> ALU算数逻辑单元
signal ESUM : std_logic;   --ALU输出信号   CTRLM控制电路 --> ALU算数逻辑单元
signal DA : std_logic_vector(15 downto 0);   --ALU输入数据
signal IIR : std_logic; --IREG寄存信号  CTRLM控制电路 --> IR指令寄存器
signal Address : std_logic_vector(3 downto 0);  --地址数据   MAR数据寄存器 --> RAM存储器
signal A_OUT : std_logic_vector(15 downto 0);
signal DR_OUT : std_logic_vector(15 downto 0);
signal dbus_OUT : std_logic_vector(15 downto 0);

signal ABUS : std_logic_vector(3 downto 0);  --地址线
signal D_BUS : std_logic_vector(15 downto 0);   --外部数据总线

begin
--输出信号赋值
Tap <= T;
Code_IR <= Code;
IPC_O <= IPC;
IMAR_O <= IMAR;
IDR_O <= IDR;  EDR_O<=EDR;
W_O <= W;
R_O <= R;
ISUM_O <= ISUM;
ESUM_O <= ESUM;
IIR_O <= IIR;
ABUS_O <= ABUS;
dbus_O <= dbus_OUT;
ALU_OUT <= A_OUT;

CL : CLOCK port map(CLK, CLK_Used); --时钟信号产生器
CT : COUNTER port map(CLK_Used, CLR, T); --节拍发生器
CTRLM : CTRL port map(CLK_Used, Code, T, IPC, IMAR, IDR, EDR, W, R, Code_A, ISUM, ESUM, IIR);   --控制电路
IR : IREG port map(CLK_Used, dbus_OUT, IIR, Code, ERROR, STOP);   --指令寄存器
DR : DREG port map(CLK_Used, D_BUS, IDR, EDR, DR_OUT);   --数据寄存器
ACC : DREG port map(CLK_Used, dbus_OUT, W, R, DA); --累加器
P : PC port map(CLK_Used, CLR, IPC, Address);   --程序寄存器
MAR : MAREG port map(CLK_Used, Address, IMAR, ABUS);  --地址寄存器
RAMM : RAM port map(CS, WR, Add_IN, ABUS, Input, D_BUS); --存储器
A : ALU port map(DA, dbus_OUT, Code_A, ISUM, ESUM, A_OUT, Zero, Over, Nege);  --算数逻辑单元
d : dbus port map(A_OUT, DR_OUT, dbus_OUT);  --总线
end vhd;
```

图 4-2-2 顶层模块 VHDL 代码(2)

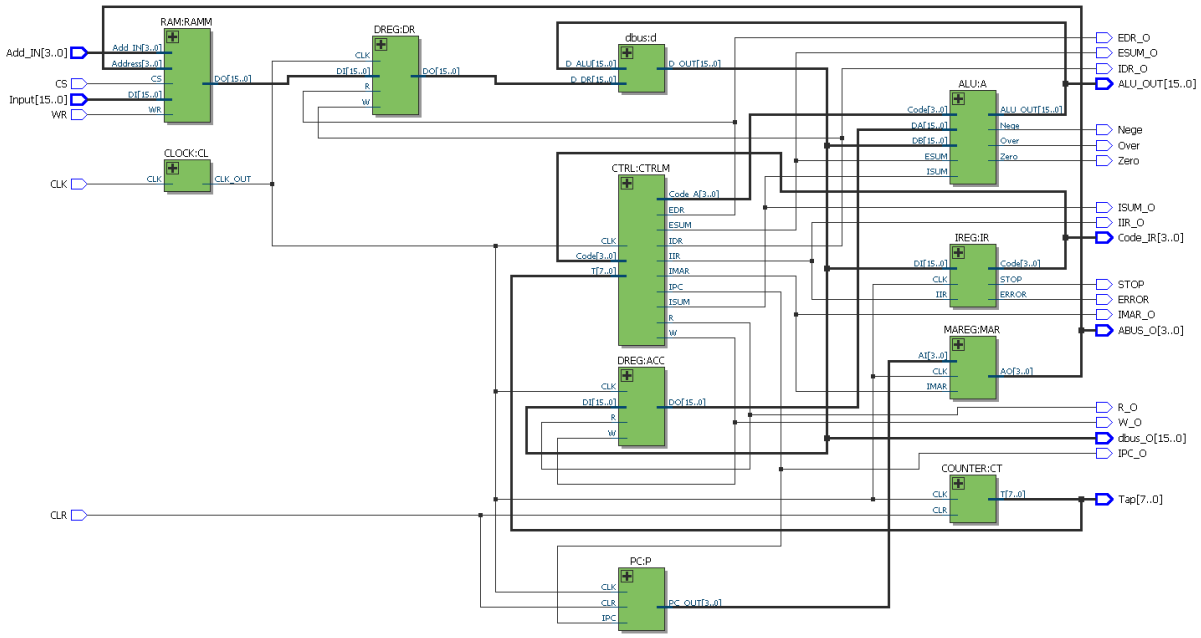

图 4-2-3 顶层模块 RTL 图

4.2 顶层模块仿真

编写 testbenth 文件对顶层模块测试。

通过仿真文件输入代码指令 LD 6; ADD A,7;NOT A; LD 392; XOR A,29152; INC A; SUB A,28799; HALT; 获得输出分别为 0000000000001101、1111111111110010、0111000001101000、0111000001101001 和 1111111111101010，结果完全正确。

```
--程序计数器仿真文件
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ms_cpu is
end ms_cpu;

architecture modelsim of ms_cpu is
component Sixteen_Bit_CPU
   port (CLK : in std_logic;  --时钟信号
         CLR : in std_logic;  --复位信号
         CS, WR : in std_logic;  --RAM的控制信号
         Add_IN : in std_logic_vector(3 downto 0); --RAM输入数据存放地址
         Input : in std_logic_vector(15 downto 0); --RAM输入数据
         ALU_OUT : out std_logic_vector(15 downto 0); --ALU输出数据
         Zero, Over, Nege : out std_logic; --标志位信号
         ERROR,STOP : out std_logic;

         Tap : out std_logic_vector(7 downto 0);
         Code_IR : out std_logic_vector(3 downto 0);
         IPC_O : out std_logic;  --PC信号  CTRLM控制电路 --> PC程序计数器
         IMAR_O : out std_logic; --MAR寄存信号   CTRLM控制电路 --> MAR地址寄存器
         IDR_O, EDR_O : out std_logic; --DR寄存、输出信号  CTRLM控制电路 --> DR数据寄存器
         W_O, R_O : out std_logic;  --ALU中DR输入输出信号  CTRLM控制电路 --> ALU数据寄存器
         ISUM_O : out std_logic; --ALU指令信号   CTRLM控制电路 --> ALU算数逻辑单元
         ESUM_O : out std_logic; --ALU输出信号   CTRLM控制电路 --> ALU算数逻辑单元
         IIR_O : out std_logic;  --IREG寄存信号  CTRLM控制电路 --> IR指令寄存器
         ABUS_O : out std_logic_vector(3 downto 0);   --地址线
         dbus_O : out std_logic_vector(15 downto 0) --内部总线
         );
   end component;

   signal CLK : std_logic; --时钟信号
   signal CLR : std_logic; --复位信号
   signal CS, WR : std_logic; --RAM的控制信号
   signal Add_IN : std_logic_vector(3 downto 0); --RAM输入数据存放地址
   signal Input : std_logic_vector(15 downto 0);   --RAM输入数据
   signal ALU_OUT : std_logic_vector(15 downto 0); --ALU输出数据
   signal Zero, Over, Nege : std_logic; --标志位信号
   signal ERROR,STOP : std_logic;

   signal Tap : std_logic_vector(7 downto 0);
   signal Code_IR : std_logic_vector(3 downto 0);
   signal IPC_O : std_logic;  --PC信号  CTRLM控制电路 --> PC程序计数器
   signal IMAR_O : std_logic; --MAR寄存信号   CTRLM控制电路 --> MAR地址寄存器
   signal IDR_O, EDR_O : std_logic; --DR寄存、输出信号  CTRLM控制电路 --> DR数据寄存器
   signal W_O, R_O : std_logic;  --ALU中DR输入输出信号  CTRLM控制电路 --> ALU数据寄存器
   signal ISUM_O : std_logic; --ALU指令信号   CTRLM控制电路 --> ALU算数逻辑单元
   signal ESUM_O : std_logic; --ALU输出信号   CTRLM控制电路 --> ALU算数逻辑单元
   signal IIR_O : std_logic;  --IREG寄存信号  CTRLM控制电路 --> IR指令寄存器
   signal ABUS_O : std_logic_vector(3 downto 0);   --地址线
   signal dbus_O : std_logic_vector(15 downto 0); --内部总线
begin
CPU : Sixteen_Bit_CPU port map  (CLK, CLR, CS, WR, Add_IN, Input, ALU_OUT, Zero, Over, Nege, ERROR, STOP,
                                 Tap, Code_IR, IPC_O, IMAR_O, IDR_O, EDR_O, W_O, R_O, ISUM_O, ESUM_O, IIR_O, ABUS_O, dbus_O);
process
   begin
      CLK<= '0';
      wait for 5 ns;
      CLK<='1';
      wait for 5 ns;
   end process;

process
   begin
      CS<='1';
      wait;
   end process;

process
   begin
      CLR <= '1';
      wait for 400 ns;
      CLR <= '0';
      wait;
   end process;

process
   begin
      WR<='0'; Add_IN<="0000";   Input<="0000000000111110"; --指令   LD
      wait for 30 ns;
      WR<='0'; Add_IN<="0001";   Input<="0000000000000110"; --操作数  6
      wait for 30 ns;

      WR<='0'; Add_IN<="0010";   Input<="0000000011000110"; --指令   ADD
      wait for 30 ns;
      WR<='0'; Add_IN<="0011";   Input<="0000000000000111"; --操作数  7              结果   0000000000001101(13)
      wait for 30 ns;

      WR<='0'; Add_IN<="0100";   Input<="0101010101011110"; --指令   NOT            结果   1111111111110010
      wait for 30 ns;

      WR<='0'; Add_IN<="0101";   Input<="0000000000111110"; --指令   LD
      wait for 30 ns;
      WR<='0'; Add_IN<="0110";   Input<="0000000110001000"; --操作数  392
      wait for 30 ns;

      WR<='0'; Add_IN<="0111";   Input<="0111100001010000"; --指令   XOR
      wait for 30 ns;
      WR<='0'; Add_IN<="1000";   Input<="0111000111100000"; --操作数  29152          结果   0111000001101000
      wait for 30 ns;

      WR<='0'; Add_IN<="1001";   Input<="0010111011100111"; --指令   INC            结果   0111000001101001(28777)
      wait for 30 ns;

      WR<='0'; Add_IN<="1010";   Input<="0001001100011110"; --指令   SUB
      wait for 30 ns;
      WR<='0'; Add_IN<="1011";   Input<="0111000001111111"; --操作数  28799          结果   1111111111101010(65536-22) Nege 1   溢出
      wait for 30 ns;

      WR<='0'; Add_IN<="1100";   Input<="0000000001110110"; --指令   HALT
      wait for 30 ns;
      WR<='1';
      wait;
   end process;
end modelsim;
```

图 4-2-4 顶层模块仿真文件

从仿真结果可以看出，控制系统输出为控制指令正确，ALU 计算满足代码书写，可以正确完成程序执行。综上，满足设计要求。

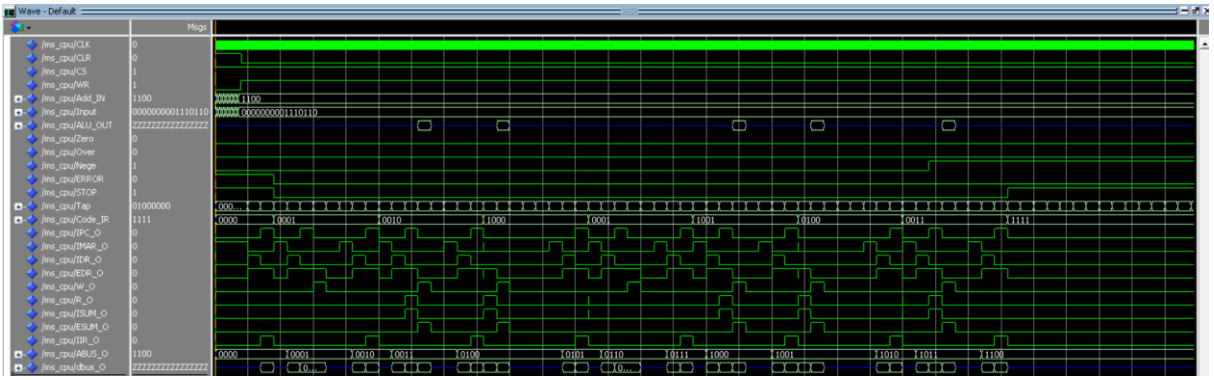

图 4-2-5 顶层模块仿真结果总览

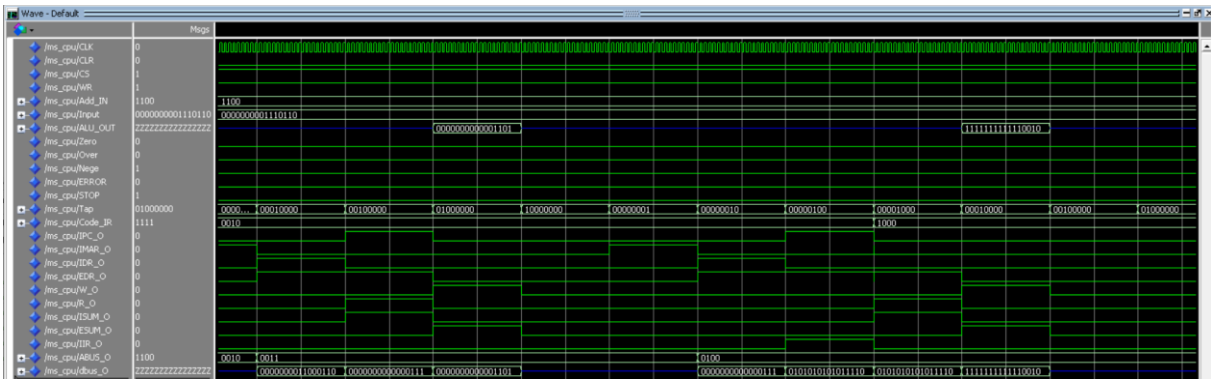

图 4-2-6 顶层模块仿真结果局部放大

第 5 章 结论

本次课设题目是“16 位模型机的设计与实现”，该课题从软件实现平台的学习到程序的设计与仿真，实现了对 16 位模型计算机各部分功能的仿真。

在开发期间，有目的的去学习一些将要用到的东西，仔细的考虑工作流程的规律和步骤充分的利用手中的开发工具，对 VHDL 语言有了深刻地了解与掌握，并对 QuartusII 软件有了深入的了解。

运用 QuartusII 软件，基于 VHDL 的 16 位模型计算机，完成了以下工作：

针对 8 位模型计算机的总体框图划分成几个相对独立的模块，然后对各个模块在开发环境中进行编程和调试。

对每个模块单独进行分析设计，并在 Quartus II 中进行了功能仿真。

在设计的过程中，将一些常用的模块定义为相应的逻辑元件符号，以便共享和复用，使设计具有可重用性和可移植性，提高工作效率。

各个模块都调试成功后将各个模块连接起来总体调试，按照总体的设计图进行集成调试。

16 位模型计算机的仿真实验结果表明波形正确，达到了设计的功能要求。

通过本课题的设计，让我对数字逻辑与系统设计和计算机组成原理这两门课程有了更深的体会，并更好的学会了使用Quartus Ⅱ软件进行设计和运用VHDL语言进行编程。在完成此设计中也遇到过许多困难，设计中也存在着一些不足之处，希望在日后的学习中慢慢得到改善和提高。

参考文献

[1] 李晶皎, 李景宏, 曹阳. 逻辑与数字系统设计(M) .北京: 清华大学出版社, 2018.

[2] 李景宏，王永军等著. 数字逻辑与数字基础(第五版)[M]. 北京: 电子工业出版社, 2017.

心得体会

历时 3 周的数字逻辑与数字系统课程设计随着报告的书写完毕结束了。内心复杂。通过这次课程设计，我成功的实现了 16 位模型机的设计。在设计与实现的过程中，遇到了很多困难与问题。由于在平时几乎不适用 Modelsim 进行仿真，所以在使用它的最初就给我造成了很大的麻烦，在一个个 bug 中不断试错、挑错、改错，使得现在的我能够简单的使用 Modelsim 仿真完成此次课设的内容。其次的一大难题就是对整个模型机进行顶层的设计。尤其是节拍发生器与控制电路的逻辑关系，让我百思不得解，经过课本的翻阅，网上资料的查询以及各种 CPU 相关教学视频，我才理解整个模型机的工作原理并确定了自己模型机的设计思路与方向。深入了解了模型机的实现原理和掌握了 Modelsim 的使用，之后的代码实现过程中困扰我的就只剩源源不断地 bug。调试 bug 无疑是一种煎熬与折磨，是对自己认真、耐心与毅力地考验。而最终获得正确仿真结果时的地快乐也是不言而喻的。写代码最吸引我的也便是编译通过时的那份创造的喜悦。通过这次的课程设计，我更深入的了解了课堂所教的各个知识点，提升了我的动手能力，更锻炼了我的耐心。在之后的学习生活中，我也会以此次课程设计为宝贵经验，不断提升自己的知识深度与代码书写能力。