实验报告

智能 1602　　侯云鹏　　201608010703

实验名称

相对简单 CPU 电路设计

实验目标

利用 VHDL 设计相对简单 CPU 的电路并验证

实验要求

- 采用 VHDL 描述电路及其测试平台
- 采用时序逻辑设计电路
- 采用从 1 累加到 n 的程序进行测试

实验内容

相对简单 CPU 的规格说明

- 地址总线 16 位，数据总线 8 位
- 有一个 8 位累加寄存器 AC，一个 8 位通用寄存器 R，一个 1 位的零标志
- 有一个 16 位 AR 寄存器，一个 16 位程序计数器 PC，一个 8 位数据寄存器 DR，一个 8 位指令寄存器 IR，一个 8 位临时寄存器 TR
- 有 16 条指令，每条指令 1 个或 3 个字节，其中操作码 8 位。3 字节的指令有 16 位的地址

相对简单 CPU 设计方案

1. 指令执行过程分为取指、译码、执行三个阶段
2. 取指包括三个状态，FETCH1，FETCH2，FETCH3
3. 译码体现为从 FETCH3 状态到各指令执行状态序列的第一个状态
4. 执行根据指令的具体操作分为若干状态
5. 执行的最后一个状态转移到 FETCH1 状态
6. 控制器根据每个状态需要完成的操作产生相应的控制信号

设计内容

取指令一共分为 3 个步骤，fetch3 之后就跳转到对应的指令进行执行操作，指令执行结束后再跳转到 fetch1。

```
when fetch1 =>        -- ar<-pc
when fetch2 =>        -- ir<-M  pc++
when fetch3 =>        -- ar<-pc    ir<-bus
```

在 cpu.vhd 文件中，声明了 cpu 的内部总线、寄存器、控制信号和所有状态，以及对 alu 相关运算进行设计。设计了在什么状态做什么事情。做出下一状态的赋值，在另一个进程中 gen_controls 规定所处状态下所有控制信号的 0、1 状态。

rscpu.vhd：

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.rsisa.all;

entity rscpu is
    port(
            clk: in std_logic;
            reset: in std_logic;
            addrbus: out std_logic_vector(15 downto 0);
            databus: inout std_logic_vector(7 downto 0);
            en_read: out std_logic;
            en_write: out std_logic
        );
end entity;
```

```vhdl
architecture rscpu_behav of rscpu is
    --reg
    signal pc: std_logic_vector(15 downto 0);
    signal ac: std_logic_vector(7 downto 0);
    signal r: std_logic_vector(7 downto 0);
    signal ar: std_logic_vector(15 downto 0);
    signal ir: std_logic_vector(7 downto 0);
    signal dr: std_logic_vector(7 downto 0);
    signal tr: std_logic_vector(7 downto 0);
    signal z: std_logic;

    -- control signal
    signal alu_select: std_logic_vector(2 downto 0);
    signal arload, arinc, pcload, pcinc, drload, trload, irload, rload, acload, zload: std_logic;
    signal pcbus, drhbus, drlbus, trbus, rbus, acbus, membus: std_logic;
    signal bus_enable: std_logic_vector(6 downto 0);

    --other signal
    signal alu_result: std_logic_vector(7 downto 0);
    signal main_bus: std_logic_vector(15 downto 0);
    signal acinc, accl, write_tmp:std_logic;
```

```vhdl
    -- address and data bus
    addrbus <= ar;
    databus <= dr when write_tmp='1' else "ZZZZZZZZ";

    main_bus <= pc when pcbus='1' else
                (dr & tr) when (drhbus='1' and trbus='1') else
                (X"00" & dr) when drlbus='1' else
                (X"00" & r) when rbus='1' else
                (X"00" & ac) when acbus='1' else
                (X"00" & databus) when membus='1' else
                "ZZZZZZZZZZZZZZZZ";

    alu_result <= main_bus(7 downto 0) when alu_select="000" else
                  std_logic_vector(unsigned(ac)+unsigned(main_bus(7 downto 0))) when alu_select="001" else
                  std_logic_vector(unsigned(ac)-unsigned(main_bus(7 downto 0))) when alu_select="010" else
                  ac and main_bus(7 downto 0) when alu_select="011" else
                  ac or main_bus(7 downto 0) when alu_select="100" else
                  ac xor main_bus(7 downto 0) when alu_select="101" else
                  not ac when alu_select="110" else
                  main_bus(7 downto 0);
```

```
    --update the values of register,bus...
    process_reg:process(clk)
    begin
        if (rising_edge(clk)) then
            --when reset
            if(reset='1') then
                pc <= X"0000";
                state <= fetch1;
            else
                state <= nextstate;
            end if;
            --reg update reffers to control signal
            if(arload = '1') then
                ar <= main_bus;
            end if;
            if(arinc = '1') then
                ar <= std_logic_vector(unsigned(ar)+1);
            end if;
            if(pcload = '1') then
                pc <= main_bus;
            end if;
            if(pcinc = '1') then
                pc <= std_logic_vector(unsigned(pc)+1);
            end if;
            if(drload = '1') then
                dr <= main_bus(7 downto 0);
            end if;
            if(trload = '1') then
                tr <= dr;
            end if;

--generate next state
for_nextstate: process(state, ir)
begin
    case state is
        when fetch1 =>
            nextstate <= fetch2;
        when fetch2 =>
            nextstate <= fetch3;
        when fetch3 =>
            case ir is
                when RSNOP =>
                    nextstate <= nop1;
                when RSLDAC =>
                    nextstate <= ldac1;
                when RSSTAC =>
                    nextstate <= stac1;
                when RSMVAC =>
                    nextstate <= mvac1;
                when RSMOVR =>
                    nextstate <= movr1;
                when RSJUMP =>
                    nextstate <= jump1;
                when RSJMPZ =>
                    if(z='1') then
                        nextstate <= jmpzy1;
                    elsif(z='0') then
                        nextstate <= jmpzn1;
                    end if;
                when RSJPNZ =>
                    if(z='1') then
                        nextstate <= jpnzn1;
```

```
--generate control signal
control_signal:process(state)
begin
    case state is
        when fetch1 =>      -- ar<-pc
            arload <= '1';  arinc <= '0';   pcload <= '0';  pcinc <= '0';   drload <= '0';  alu_select <= "000";
            trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
            pcbus <= '1';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '0';
            acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
        when fetch2 =>      -- ir<-M    pc++
            arload <= '0';  arinc <= '0';   pcload <= '0';  pcinc <= '1';   drload <= '0';  alu_select <= "000";
            trload <= '0';  irload <= '1';  rload <= '0';   acload <= '0';  zload <= '0';
            pcbus <= '0';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '1';
            acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
        when fetch3 =>      -- ar<-pc
            arload <= '1';  arinc <= '0';   pcload <= '0';  pcinc <= '0';   drload <= '0';  alu_select <= "000";
            trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
            pcbus <= '1';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '0';
            acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
        when nop1 =>        -- do nothing
            arload <= '0';  arinc <= '0';   pcload <= '0';  pcinc <= '0';   drload <= '0';
            trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
            pcbus <= '0';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '0';
            acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
        when ldac1 =>       -- dr<-M    pc++    ar++
            arload <= '0';  arinc <= '1';   pcload <= '0';  pcinc <= '1';   drload <= '1';  alu_select <= "000";
            trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
```

rsisa.vhd:

在 rsisa.vhd 文件中，对下一个状态进行了指令的一个编码，并且给它一个相关的变量名，这是为了在 cpu.vhd 中对变量名的判断从而执行下一个状态。

```
package rsisa is

    -- RS prefix is used to avoid tautonym such like AND, OR, XOR, NOT
    constant RSNOP: std_logic_vector(7 downto 0) := "00000000";
    constant RSLDAC: std_logic_vector(7 downto 0) := "00000001";
    constant RSSTAC: std_logic_vector(7 downto 0) := "00000010";
    constant RSMVAC: std_logic_vector(7 downto 0) := "00000011";
    constant RSMOVR: std_logic_vector(7 downto 0) := "00000100";
    constant RSJUMP: std_logic_vector(7 downto 0) := "00000101";
    constant RSJMPZ: std_logic_vector(7 downto 0) := "00000110";
    constant RSJPNZ: std_logic_vector(7 downto 0) := "00000111";

    constant RSADD: std_logic_vector(7 downto 0) := "00001000";
    constant RSSUB: std_logic_vector(7 downto 0) := "00001001";
    constant RSINAC: std_logic_vector(7 downto 0) := "00001010";
    constant RSCLAC: std_logic_vector(7 downto 0) := "00001011";
    constant RSAND: std_logic_vector(7 downto 0) := "00001100";
    constant RSOR: std_logic_vector(7 downto 0) := "00001101";
    constant RSXOR: std_logic_vector(7 downto 0) := "00001110";
    constant RSNOT: std_logic_vector(7 downto 0) := "00001111";

end package;
```

rsmem.vhd:

在 rsmem.vhd 文件中，设计了内存中的指令和操作数等，实现了 1 加到 n 的操作。

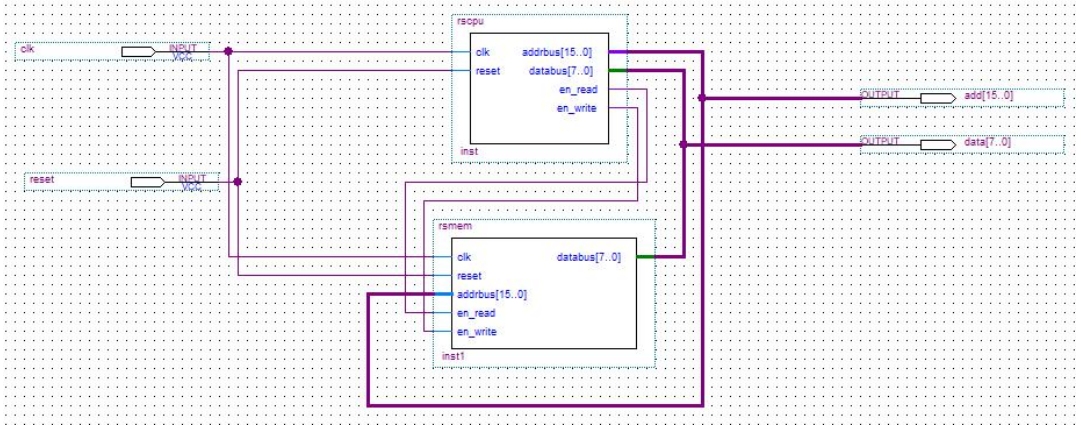
```
0 => RSCLAC,
1 => RSSTAC,
2 => std_logic_vector(to_unsigned(total_addr, 8)),
3 => X"00",
4 => RSSTAC,
5 => std_logic_vector(to_unsigned(i_addr, 8)),
6 => X"00",
7 => RSLDAC,  -- loop
8 => std_logic_vector(to_unsigned(i_addr, 8)),
9 => X"00",
10 => RSINAC,
11 => RSSTAC,
12 => std_logic_vector(to_unsigned(i_addr, 8)),
13 => X"00",
14 => RSMVAC,
15 => RSLDAC,
16 => std_logic_vector(to_unsigned(total_addr, 8)),
17 => X"00",
18 => RSADD,
19 => RSSTAC,
20 => std_logic_vector(to_unsigned(total_addr, 8)),
21 => X"00",
22 => RSLDAC,
23 => std_logic_vector(to_unsigned(n_addr, 8)),
24 => X"00",
25 => RSSUB,
26 => RSJPNZ,
27 => std_logic_vector(to_unsigned(loop_addr, 8)),
28 => X"00",
29 => X"00",  -- total
30 => X"00",  -- i
31 => X"04",  -- n
```

0、将 AC 清 0
1、将 AC 存到 29 号地址
2、将 AC 存到 30 号地址
3、将 30 号地址的值传到 AC
4、AC++
5、将 AC 存到 30 号地址
6、AC——>R
7、将 29 号地址的值传到 AC
8、AC=R+AC

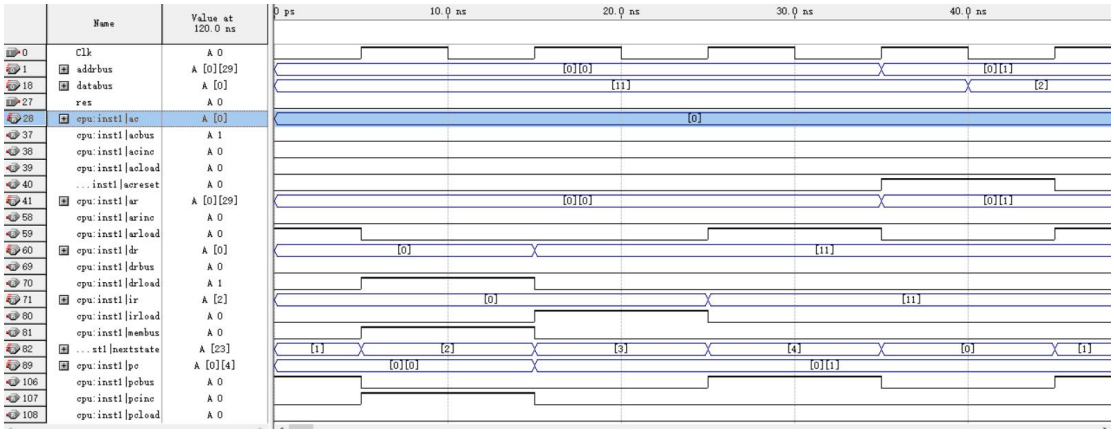
9、将 AC 存到 29 号地址
10、将 31 号地址的值传到 AC
11、AC=AC-R
12、IF　AC！=0　　跳转到第 3 步

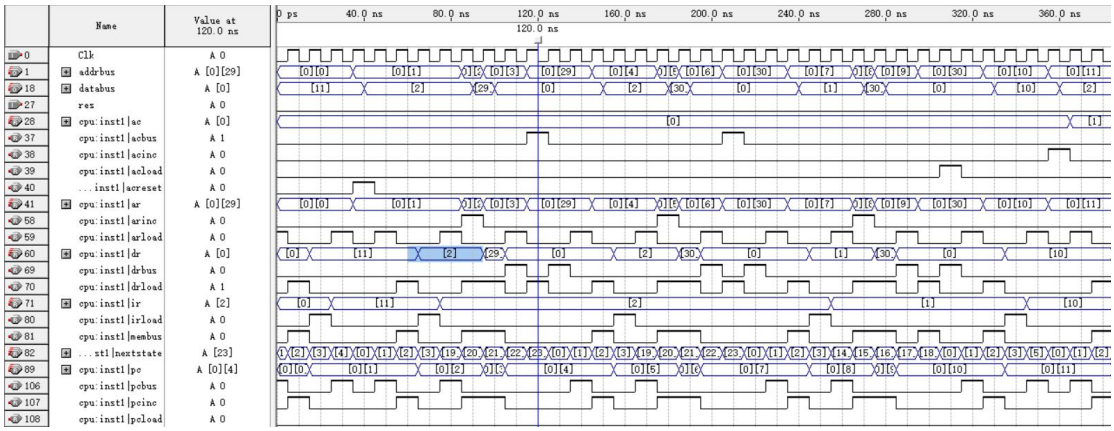
最终实现 1 累加到 n

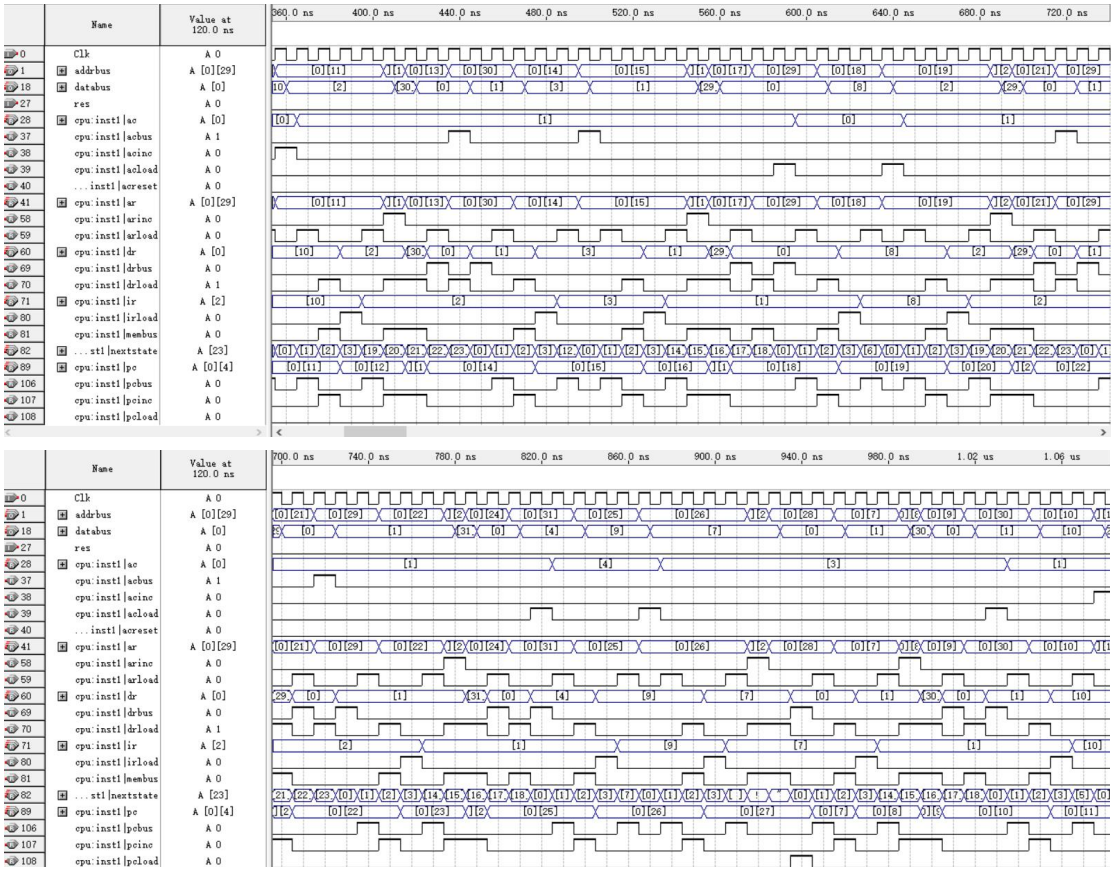
顶层文件：



仿真结果






Name
Value at 120.0 ns
360.0 ns
400.0 ns
440.0 ns
480.0 ns
520.0 ns
560.0 ns
600.0 ns
640.0 ns
680.0 ns
720.0 ns
0 Clk A 0
1 addrbus A [0][29]
18 databus A [0]
27 res A 0
28 cpu:inst1|ac A [0]
37 cpu:inst1|acbus A 1
38 cpu:inst1|acinc A 0
39 cpu:inst1|acload A 0
40 ...inst1|acreset A 0
41 cpu:inst1|ar A [0][29]
58 cpu:inst1|arinc A 0
59 cpu:inst1|arload A 0
60 cpu:inst1|dr A [0]
69 cpu:inst1|drbus A 0
70 cpu:inst1|drload A 1
71 cpu:inst1|ir A [2]
80 cpu:inst1|irload A 0
81 cpu:inst1|membus A 0
82 ...st1|nextstate A [23]
89 cpu:inst1|pc A [0][4]
106 cpu:inst1|pcbus A 0
107 cpu:inst1|pcinc A 0
108 cpu:inst1|pcload A 0

Name
Value at 120.0 ns
700.0 ns
740.0 ns
780.0 ns
820.0 ns
860.0 ns
900.0 ns
940.0 ns
980.0 ns
1.02 us
1.06 us
0 Clk A 0
1 addrbus A [0][29]
18 databus A [0]
27 res A 0
28 cpu:inst1|ac A [0]
37 cpu:inst1|acbus A 1
38 cpu:inst1|acinc A 0
39 cpu:inst1|acload A 0
40 ...inst1|acreset A 0
41 cpu:inst1|ar A [0][29]
58 cpu:inst1|arinc A 0
59 cpu:inst1|arload A 0
60 cpu:inst1|dr A [0]
69 cpu:inst1|drbus A 0
70 cpu:inst1|drload A 1
71 cpu:inst1|ir A [2]
80 cpu:inst1|irload A 0
81 cpu:inst1|membus A 0
82 ...st1|nextstate A [23]
89 cpu:inst1|pc A [0][4]
106 cpu:inst1|pcbus A 0
107 cpu:inst1|pcinc A 0
108 cpu:inst1|pcload A 0