实验报告

物联 1601　　刘小芬　　201608010107

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

增加了 FETCH4 状态一共四个状态。取址包括四个状态，而译码体现为从 FETCH4 状态到各指令执行状态序列的第一个状态。执行根据指令的具体操作分为不同的状态，并且其最后一个状态转移到 FETCH1 状态。

```
when fetch1=>--ar<=pc
when fetch2=>--dr<=m  pc<=pc+1
when fetch3=>--ir<=dr
when fetch4=>--ar<=pc
```

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所对应的编码，以及对 alu 相关运算进行设计，并声明了所有会执行的指令。在时钟信号的上升沿，对信号 0、1 时刻下对应的操作做了规定。在进程 for_nextstate 中判断所处状态的下一个状态是什么，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。

其中部分代码：

```
    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;
    signal thebus: std_logic_vector(15 downto 0);

    if(rising_edge(clk)) then
            --update registers
            if(acreset='1') then
                    ac<="00000000";
                    z<='0';
            end if;
            if(reset='1') then
                    pc <= "0000000000000000";
                    state <= fetch1;
            else
                    state <= nextstate;
            end if;
            if(pcinc='1')then
                    pc<=std_logic_vector(unsigned(pc) + 1);
            end if;
            if(acinc='1')then
                    ac<=std_logic_vector(unsigned(ac) + 1);
                    if(ac="11111111")then
                    z<='1';
                    else
                    z<='0';
                    end if;
            end if;


case state is
        when fetch1=>
                nextstate<=fetch2;
        when fetch2=>
                nextstate<=fetch3;
        when fetch3=>
                nextstate<=fetch4;
        when fetch4=>
                case ir is
                        when RSCLAC=>
                                nextstate<=clac1;
                        when RSINAC=>
                                nextstate<=incac1;
                        when RSADD=>
                                nextstate<=add1;
                        when RSSUB=>
                                nextstate<=sub1;
                        when RSAND=>
                                nextstate<=and1;
                        when RSOR=>
                                nextstate<=or1;
                        when RSXOR=>
                                nextstate<=xor1;

case state is
        when fetch1=>--ar<=pc
                arload<='1';pcbus<='1';pcinc<='0';
                drload<='0';membus<='0';irload<='0';
                acreset<='0';acinc<='0';rbus<='0';
                s<="000";acload<='0';rload<='0';
                acbus<='0';arinc<='0';trbus<='0';
                drbus<='0';trload<='0';read<='0';
                write1<='0';write<='0';pcload<='0';
        when fetch2=>--dr<=m  pc<=pc+1
                arload<='0';pcbus<='0';pcinc<='1';
                drload<='1';membus<='1';irload<='0';
                acreset<='0';acinc<='0';rbus<='0';
                s<="000";acload<='0';rload<='0';
                acbus<='0';arinc<='0';trbus<='0';
                drbus<='0';trload<='0';read<='1';
                write1<='0';write<='0';pcload<='0';
```

在 rsisa.vhd 文件中，对 FETCH4 所要进行的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。

```
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";
```

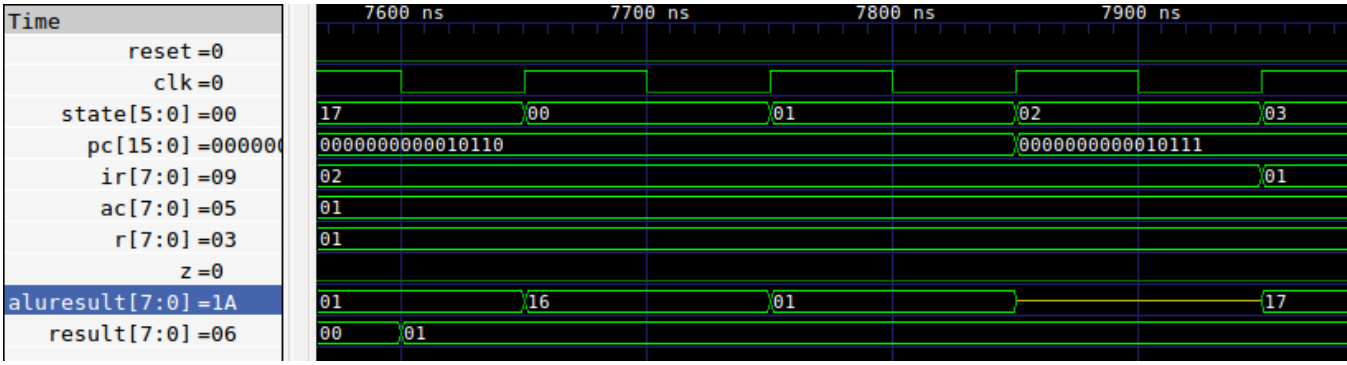
在 mem.vhd 文件中，对内存大小进行了规定为 0 到 65535，之前的 4096 太小了会溢出。对内存所要执行的东西进行了设计，可以从 1 到 n 进行加法运算，n 暂时设为 8，result 变量对应没进行一次运算的结果，可以很直观的看出结果是否正确。

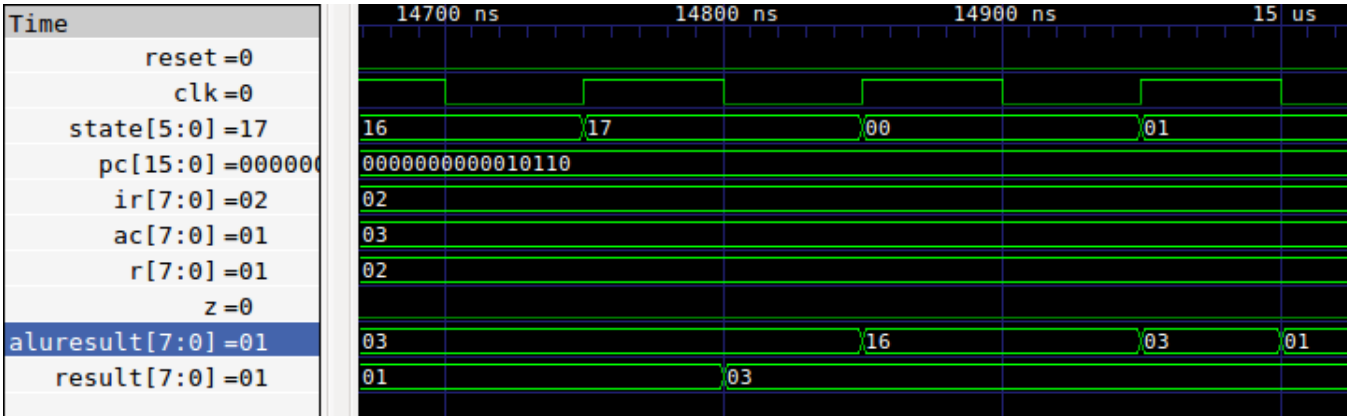
```
signal addr: std_logic_vector(15 downto 0);
signal result: std_logic_vector(7 downto 0);

begin
        if(falling_edge(clk)) then
                if(reset='1') then
                        addr <= (others=>'0');
                else
                        addr <= addrbus;
                end if;

                if(write='1') then
                        memdata(to_integer(unsigned(addr))) <= databus;
                end if;
        end if;
end process;

databus <= memdata(to_integer(unsigned(addr))) when (write='0') else "ZZZZZZZZ";
result<=memdata(29);
```

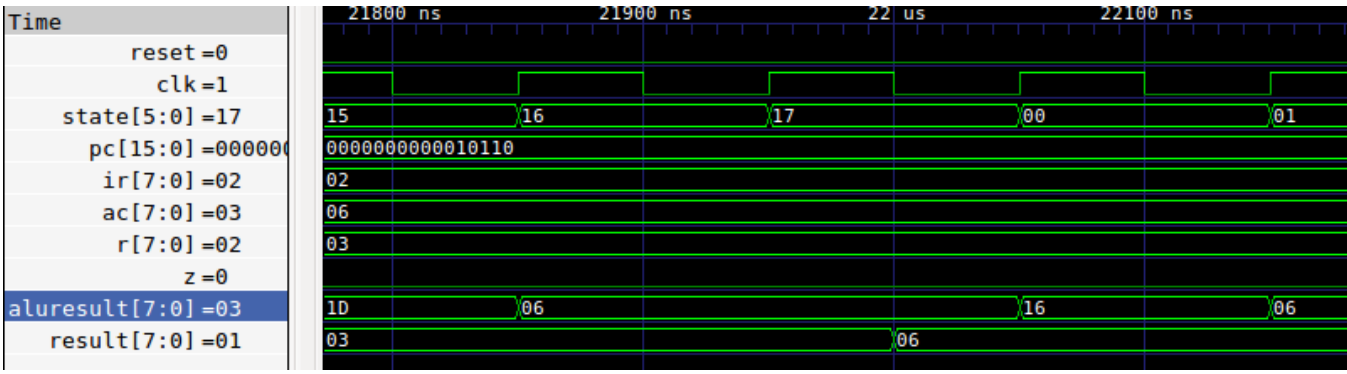
在 comp.vhd 文件中，对 cpu 和 mem 端口进行了连接，并规定了时钟周期和 reset 的时间大小。

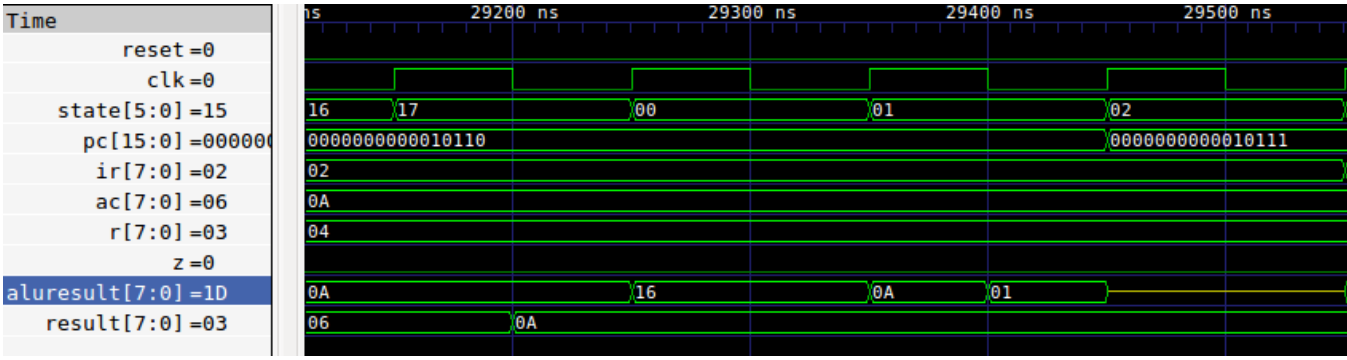
```
reset <= '1', '0' after 300 ns;
clk <= not clk after 50 ns;
```

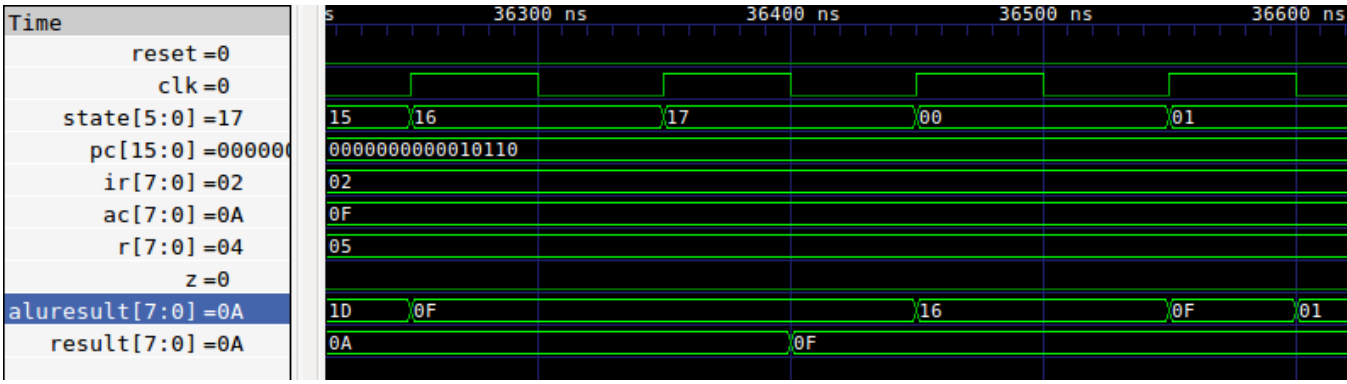
仿真结果

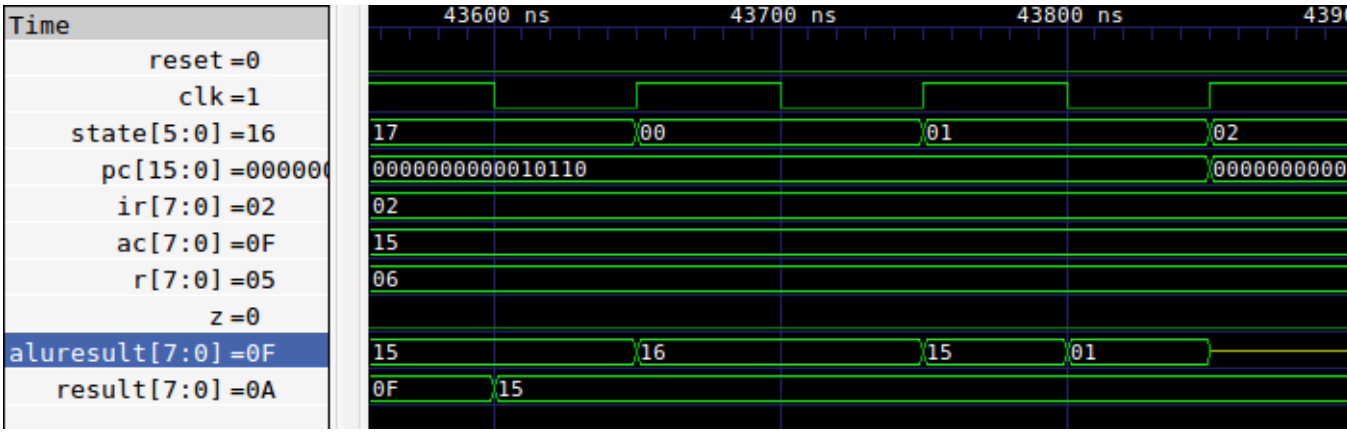

Time
reset =0
clk =0
state[5:0] =00
pc[15:0] =000000
ir[7:0] =09
ac[7:0] =05
r[7:0] =03
z =0
aluresult[7:0] =1A
result[7:0] =06
7600 ns
7700 ns
7800 ns
7900 ns
17
00
01
02
03
0000000000010110
0000000000010111
02
01
01
01
01
16
01
17
00
01

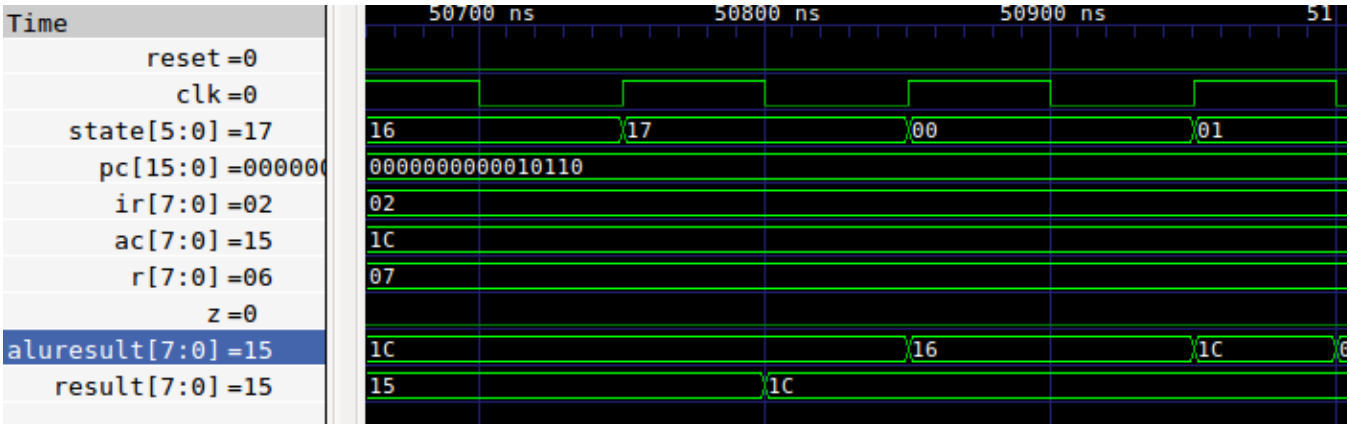

Time
reset =0
clk =0
state[5:0] =17
pc[15:0] =000000
ir[7:0] =02
ac[7:0] =01
r[7:0] =01
z =0
aluresult[7:0] =01
result[7:0] =01
14700 ns
14800 ns
14900 ns
15 us
16
17
00
01
0000000000010110
02
03
02
03
16
03
01
01
03

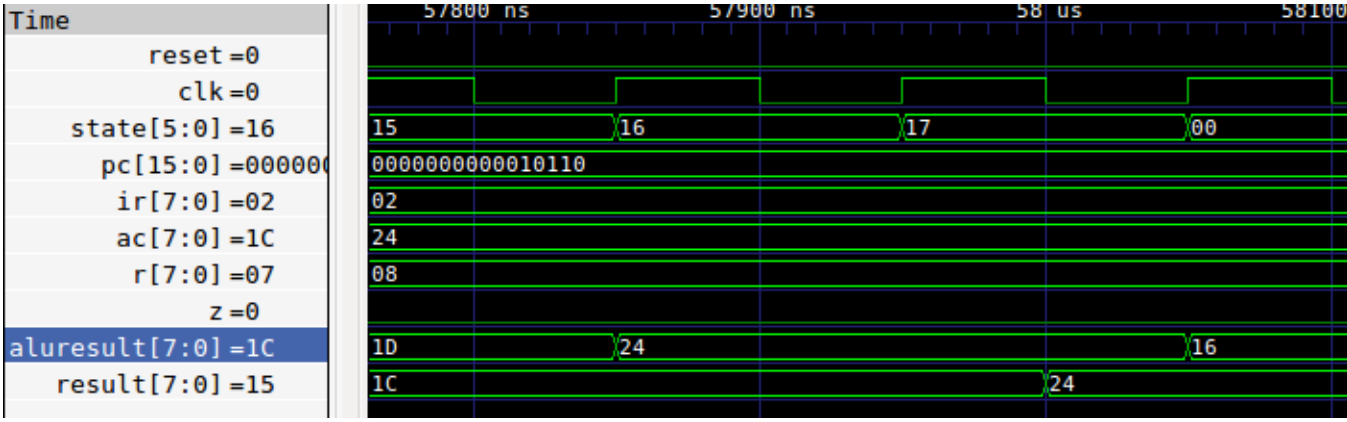

Time
reset =0
clk =1
state[5:0] =17
pc[15:0] =000000
ir[7:0] =02
ac[7:0] =03
r[7:0] =02
z =0
aluresult[7:0] =03
result[7:0] =01
21800 ns
21900 ns
22 us
22100 ns
15
16
17
00
01
0000000000010110
02
06
03
1D
06
16
06
03
06


Time
reset =0
clk =0
state[5:0] =15
pc[15:0] =000000
ir[7:0] =02
ac[7:0] =06
r[7:0] =03
z =0
aluresult[7:0] =1D
result[7:0] =03
29200 ns
29300 ns
29400 ns
29500 ns
16
17
00
01
02
0000000000010110
0000000000010111
02
0A
04
0A
16
0A
01
06
0A









可以看到仿真结果是正确的，到了 0x24 就不再变化了也就是 36。