第9章

16位CPU创新设计

9.1 KX9016的结构与特色

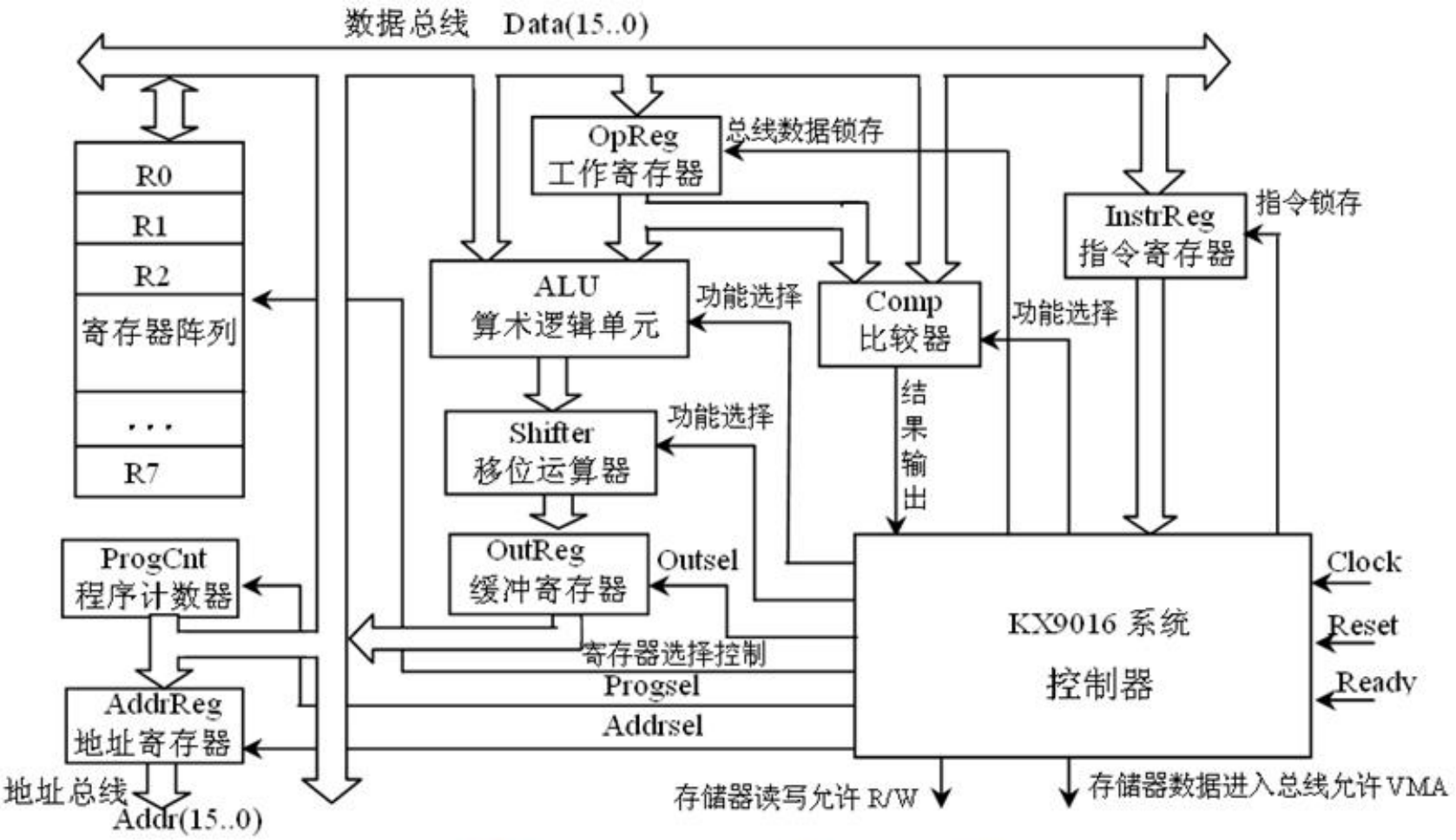

图 9-1 16 位 KX9016v1 CPU 结构图

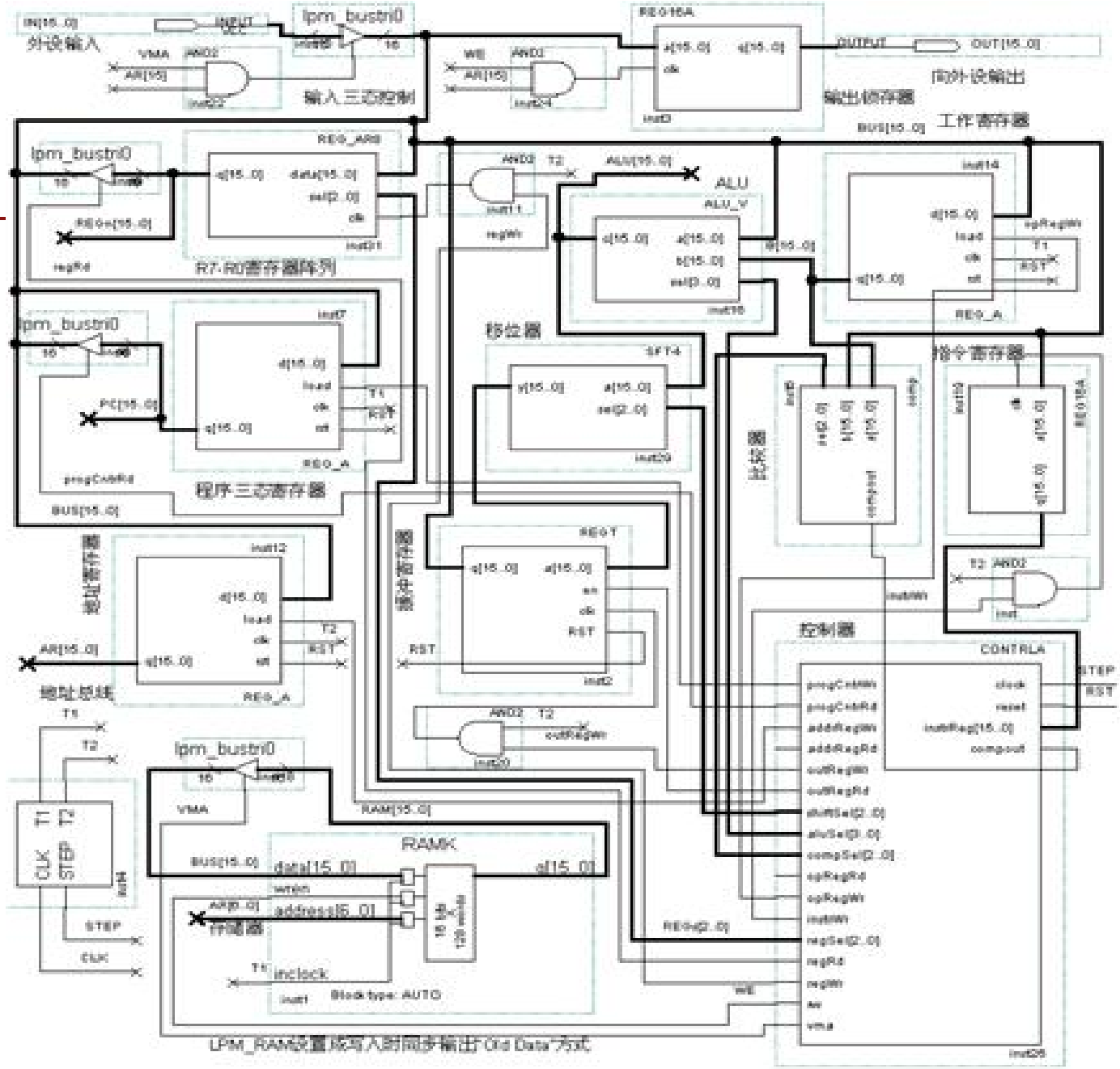

图 9-2　KX9016v1 顶层结构图

9.2 KX9016基本硬件系统设计

9.2.1 单步节拍发生模块

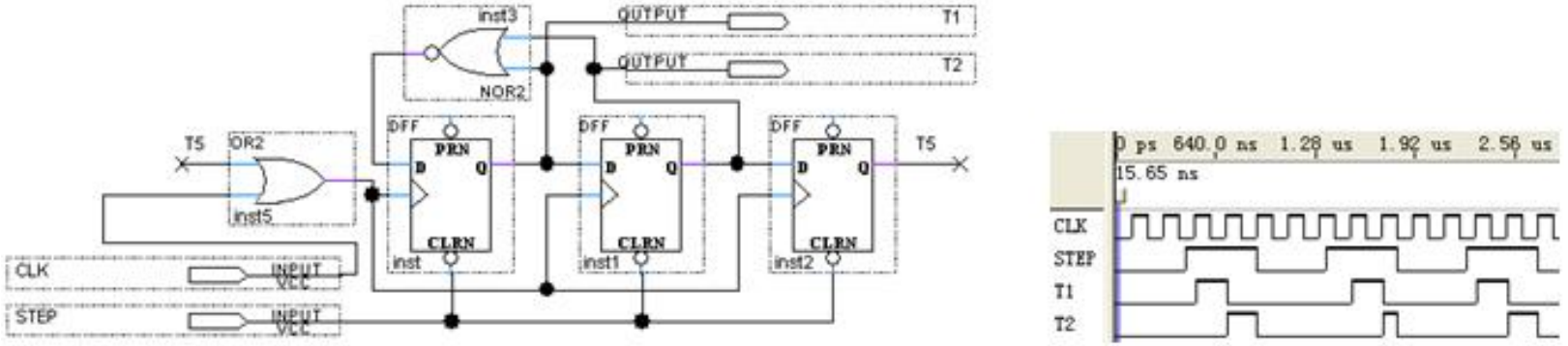

图 9-3　节拍脉冲发生器 STEP2 的电路及其仿真波形图

9.2 KX9016基本硬件系统设计

9.2.2 运算器

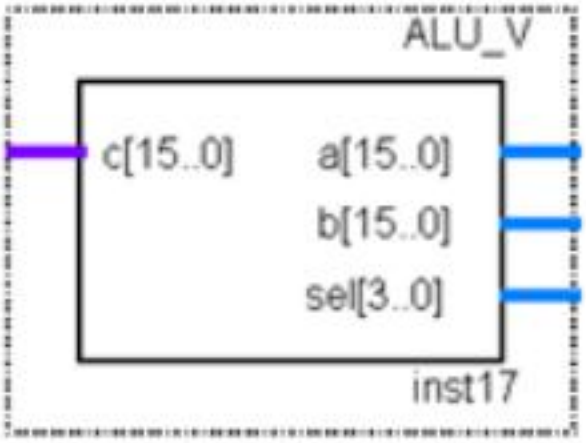

图 9-4 ALU 模块符号

【例 9-1】

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity ALU_V is
   port( a, b : in std_logic_vector(15 downto 0);
   sel : in std_logic_vector(3 downto 0);
    c : out std_logic_vector(15 downto 0));
end ALU_V;
architecture rtl of ALU_V is
    constant alupass : std_logic_vector(3 downto 0) := "0000";
    constant andOp : std_logic_vector(3 downto 0) := "0001";
    constant orOp : std_logic_vector(3 downto 0) := "0010";
    constant notOp : std_logic_vector(3 downto 0) := "0011";
    constant xorOp : std_logic_vector(3 downto 0) := "0100";
    constant plus : std_logic_vector(3 downto 0) := "0101";
    constant alusub : std_logic_vector(3 downto 0) := "0110";
    constant inc : std_logic_vector(3 downto 0) := "0111";
    constant dec : std_logic_vector(3 downto 0) := "1000";
    constant zero : std_logic_vector(3 downto 0) := "1001";
begin
    process(a, b, sel)  begin
     case sel is
      when alupass => c <= a;                          --总线数据直通ALU
      when andOp =>   c <= a and b;                    --逻辑与操作
      when orOp =>    c <= a or b;                     --逻辑或操作
      when xorOp =>   c <= a xor b;                    --逻辑异或操作
      when notOp =>   c <= not a;                      --取反操作
      when plus =>    c <= a + b;                      --算术加操作
      when alusub =>  c <= a - b;                      --算术减操作
      when inc =>     c <= a + "0000000000000001";     --加1操作
      when dec =>     c <= a - "0000000000000001";     --减1操作
      when zero =>    c <= "0000000000000000";         --输出清零
      when others =>  c <= "0000000000000000";
    end case;   end process;
end rtl;
```

9.2 KX9016基本硬件系统设计

9.2.3 比较器

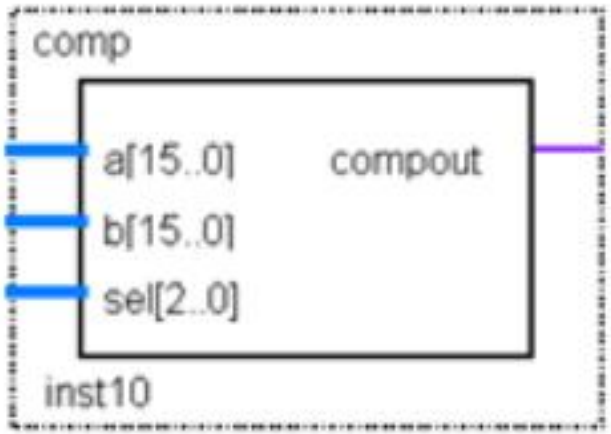

图 9-5 比较器模块符号

【例 9-2】

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
 entity comp is
  port( a, b: in std_logic_vector(15 downto 0);
       sel : in std_logic_vector ( 2 downto 0 );
       compout : out std_logic);
 end comp;
 architecture rtl of comp is
    constant    eq: std_logic_vector (2 downto 0):= "000";
    constant   neq: std_logic_vector (2 downto 0):= "001";
    constant    gt: std_logic_vector (2 downto 0):= "010";
    constant   gte: std_logic_vector (2 downto 0):= "011";
    constant    lt: std_logic_vector (2 downto 0):= "100";
    constant   lte: std_logic_vector (2 downto 0):= "101";
 begin
  process(a, b, sel)  begin
   case sel is
   when  eq => if  a=b then compout<='1'; else compout<='0'; end if;
   when neq => if a/=b then compout<='1'; elsecompout<='0'; end if;
   when  gt => if a >b then compout<='1'; else compout<='0'; end if;
   when gte => if a>=b then compout<='1'; elsecompout<='0'; end if;
   when  lt => if  a<b then compout<='1'; else compout<='0'; end if;
   when lte => if a<=b then compout<='1'; elsecompout<='0'; end if;
   when  others => compout<='0' ;
   end case;
  end process;
end rtl;
```

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

1. 基本寄存器

（1）只有锁存控制时钟的寄存器。

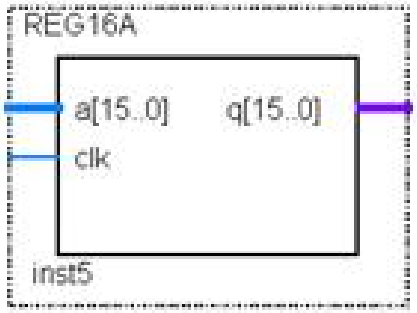

图 9-6 基本寄存器

【例 9-3】

```
library IEEE;
use IEEE.std_logic_1164.all;
entity REG16A is
    port( a : in std_logic_vector(15 downto 0);
        clk : in std_logic;
          q : out std_logic_vector(15 downto 0));
end REG16A;
architecture BHV of REG16A is
begin
  process(clk,a)  begin
    if rising_edge(clk) then q <= a; end if;
  end process;
end BHV;
```

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

（2）含三态输出控制的寄存器

【例 9-4】

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity REGT is
  port( a : in std_logic_vector ( 15 downto 0 );
    clk,RST, en : in std_logic;
     q : out std_logic_vector( 15 downto 0 ));
end REGT;
architecture rtl of REGT is
   signal vl : std_logic_vector ( 15 downto 0 );
    begin
  process(clk, a,RST)  begin
    IF RST='1' THEN vl <= "0000000000000000";
    ELSIF rising_edge(clk) THEN vl <= a; END IF;
  end process;
  process(en, vl) begin
    if en = '1' then q<=vl; else   q <= "ZZZZZZZZZZZZZZZZ"; end if;
  end process;
end rtl;
```

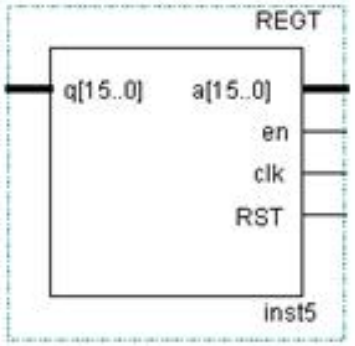

图 9-7 含三态门的寄存器

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

（3）含清零和数据锁存同步使能控制的寄存器

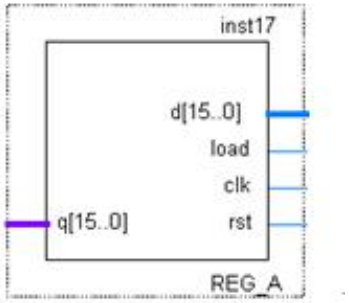

图 9-8 含加载使能的寄存器

【例 9-5】

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned. ALL;
ENTITY REG_A IS
      PORT ( rst,clk, load: IN std_logic;
          d: IN std_logic_vector (15 downto 0);
          q: OUT std_logic_vector (15 downto 0) );
END REG_A;
ARCHITECTURE Behavioral OF REG_A IS
BEGIN
   PROCESS (clk,rst )     BEGIN
    IF rst='1' THEN   q<=(OTHERS=>'0');
     ELSIF rising_edge(CLK) THEN
       IF load='1' THEN  q<=d;  END IF;
    END IF;
   END PROCESS;
END Behavioral;
```

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

（3）含清零和数据锁存同步使能控制的寄存器

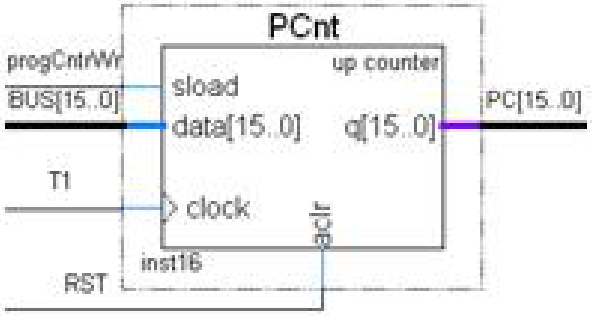

图 9-9 PC 替代电路

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

1. 基本寄存器

（1）只有锁存控制时钟的寄存器。

（2）含三态输出控制的寄存器

（3）含清零和数据锁存同步使能控制的寄存器

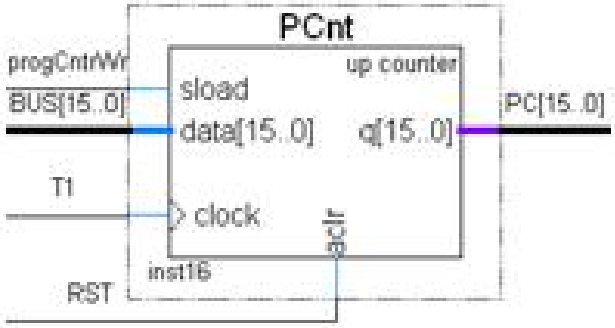

图 9-9 PC 替代电路

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

2. 寄存器阵列

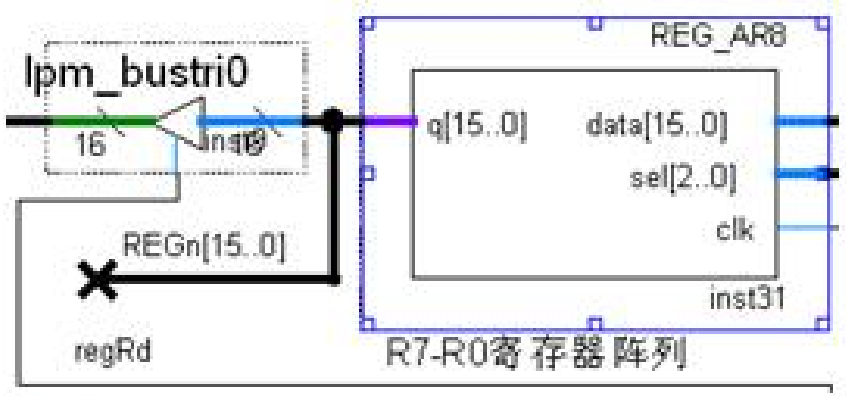

图 9-10 寄存器阵列元件与三态控制门电路

9.2 KX9016基本硬件系统设计

【例 9-6】

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity REG_AR8 is
   port( data : in  std_logic_vector(15 downto 0);
         sel : in std_logic_vector(2 downto 0);
         clk : in std_logic;  q : out std_logic_vector(15 downto 0));
end REG_AR8;
  architecture rtl of REG_AR8 is
    type t_ram is array (0 to 7) of  std_logic_vector(15 downto 0);
    signal  ramdata : t_ram;
    signal temp_data :  std_logic_vector(15 downto 0);
  begin
   process(clk,sel)  begin
    if rising_edge(clk) then ramdata(conv_integer(sel))<=data; end if;
   end process;
  process(sel)  begin
    temp_data<=ramdata(conv_integer(sel)) ; end process;
    q <= temp_data ;
end rtl;
```

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

2. 寄存器阵列

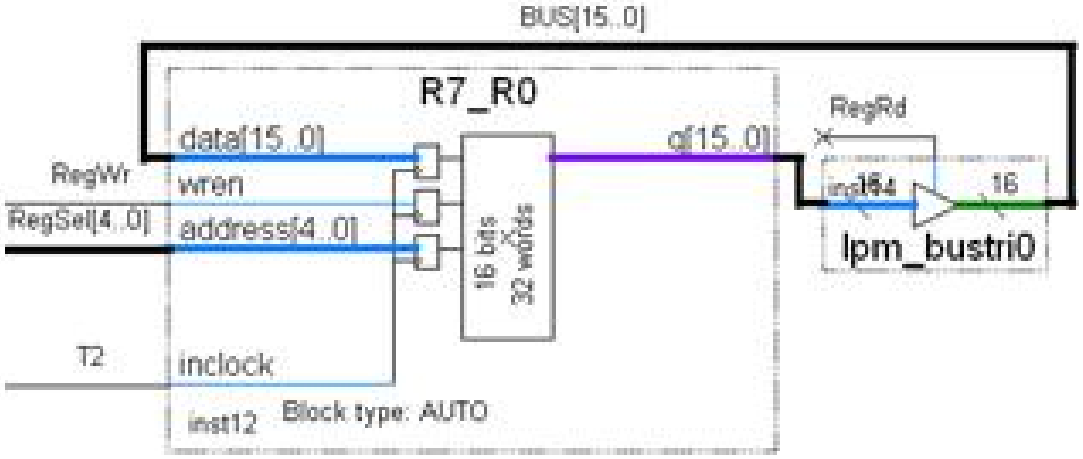

图 9-11 用 LPM_RAM 替代寄存器阵列的电路

9.2 KX9016基本硬件系统设计

9.2.5 移位器

图 9-12 移位器符号

【例 9-7】

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity SFT4 is
port ( a : in std_logic_vector (15 downto 0);
       sel : in std_logic_vector (2 downto 0);
         y : out std_logic_vector (15 downto 0));
end SFT4;
architecture rtl of SFT4 is
constant shftpass : std_logic_vector (2 downto 0) := "000";
constant    sftl : std_logic_vector (2 downto 0) := "001";
constant    sftr : std_logic_vector (2 downto 0) := "010";
constant    rotl : std_logic_vector (2 downto 0) := "011";
constant    rotr : std_logic_vector (2 downto 0) := "100";
begin
    process(a, sel) begin
     case sel is
      when shftpass =>  y<= a ;                          --数据直通
      when sftl =>      y<= a(14 downto 0) & '0' ;  --左移
      when sftr =>      y<= '0' & a(15 downto 1) ;  --右移
      when rotl =>      y<= a(14 downto 0) & a(15) ; --循环左移
      when rotr =>      y<= a(0) & a(15 downto 1) ;  --循环右移
      when others =>   y<= "0000000000000000" ;
end case;
end process;
end rtl;
```

9.2 KX9016基本硬件系统设计

9.2.6 程序与数据存储器

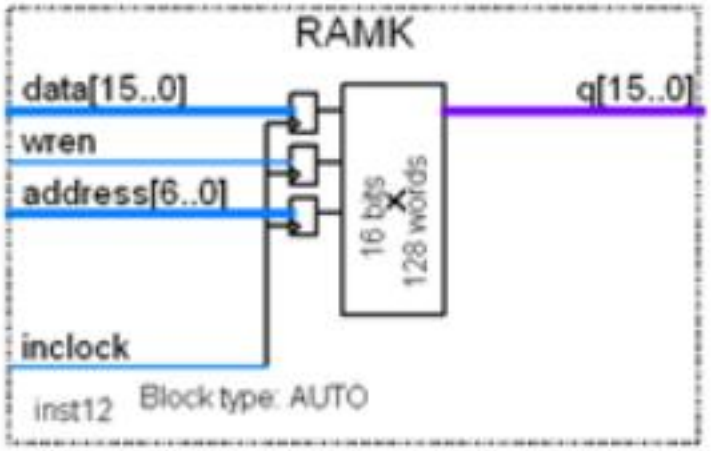

图 9-13　存储器符号

9.3 KX9016v1指令系统设计

9.3.1 指令格式

（1）单字指令。

操作码						源操作数			目的操作数		
Opcode						SRC			DST		
15	14	13	12	11		5	4	3	2	1	0

图 9-14　单字指令格式

（2）双字指令。

操作码									目的操作数		
Opcode									DST		
15	14	13	12	11					2	1	0

16 位 操作数															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

图 9-15　双字指令格式

操作码							目的操作数		
0	0	1	0	0			0	0	1

0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
0				0				1				5			

图 9-16　双字指令

9.3 KX9016v1指令系统设计

9.3.2 指令操作码

表 9-1 KX9016 预设指令及其功能表

操作码	指令	功能	操作码	指令	功能
00000	NOP	空操作	01111	IN	外设数据输出指令
00001	LD	装载数据到寄存器	10000	JMPLTI	小于时转移到立即数地址
00010	STA	将寄存器的数存入存储器	10001	JMPGT	大于时转移
00011	MOV	在寄存器间传送操作数	10010	OUT	数据输出指令
00100	LDR	将立即数装入寄存器	10011	MTAD	16 位乘法累加
00101	JMPI	转移到由立即数指定的地址	10100	MULT	16 位乘法
00110	JMPGTI	大于转移至立即数地址	10101	JMP	无条件转移
00111	INC	加 1 后放回寄存器	10110	JMPEQ	等于时转移
01000	DEC	减 1 后放回寄存器	10111	JMPEQI	等于时转移到立即地址
01001	AND	两个寄存器间与操作	11000	DIV	32 位除法
01010	OR	两个寄存器间或操作	11001	JMPLTE	小于等于时转移
01011	XOR	两个寄存器间异或操作	11010	SHL	左逻辑移位
01100	NOT	寄存器求反	11011	SHR	右逻辑移位
01101	ADD	两个寄存器加运算	11100	ROTR	循环右移
01110	SUB	两个寄存器减运算	11101	ROTL	循环左移

9.3 KX9016v1指令系统设计

9.3.2 指令操作码

表 9-2 示例程序

指令	机器码	字长	操作码	闲置码	源操作数	目的操作数	功能说明
LDR R1，0025H	2001H 0025H	2	00100	xxxxx	xxx	001 0000 0000 0010 0101	立即数 0025H 送 R1
LDR R2，0047H	2002H 0047H	2	00100	Xxxxx	xxx	010 0000 0000 0100 0111	立即数 0047H 送 R2
LDR R6，0036H	2006H 0036H	2	00100	Xxxxx	xxx	110 0000 0000 0011 0110	立即数 0036H 送 R6
LD R3，[R1]	080BH	1	00001	Xxxxx	001	011	从 R1 指定的 RAM 存储单元取数送 R3
STA [R2]，R3	101AH	1	00010	Xxxxx	011	010	将 R3 的内容存入 R2 指定 RAM 单元
JMPGTI [0000]	300EH 0000H	2	00110	Xxxxx	001	110 0000 0000 0000 0000	若 R1>R6，则转向地址[0000H]
INC R1	3801H	1	00111	Xxxxx	xxx	010	R1+1→R1
INC R2	3802H	1	00111	Xxxxx	xxx	010	R2+1→R2
JMPI [0006]	2800H 0006H	2	00101	Xxxxx	xxx	xxx 0000 0000 0000 0110	绝对地址转移指令：转向地址 0006H

9.3 KX9016v1指令系统设计

9.3.3 软件程序设计实例

表 9-3　7 条指令的汇编程序示例

地　址	机　器　码	指　令	功 能 说 明
0000H 0001H	2001H 0032H	LDR　R1, 0032H	将立即数 0032H 送寄存器 R1
0002H 0003H	2002H 0011H	LDR　R2, 0011H	将立即数 0011H 送寄存器 R2
0004H	680AH	ADD　R1, R2, R3	将寄存器 R1 和 R2 的内容相加后送 R3
0005H	1819H	MOV　R1, R3	将寄存器 R3 的内容送入 R1
0006H	3802H	INC　R2	R2 + 1→R2
0007H	101AH	STA　[R2],　R3	将 R3 的内容存入 R2 指定地址的 RAM 单元
0008H	080BH	LD　R3, [R1]	将 R1 指定地址的 RAM 单元的数据送 R3
0009H	0000H	NOP	空操作

9.3 KX9016v1指令系统设计

9.3.3 软件程序设计实例

表 9-4 存储器初始化文件 RAM_16.mif 的内容

WIDTH = 16;	03 : 0011;	0B : 0000;	13 : 0000;
DEPTH = 256;	04 : 680A;	0C : 0000;	.....
ADDRESS_RADIX = HEX;	05 : 1819;	0D : 0000;	41 : 0000;
DATA_RADIX = HEX;	06 : 3802;	0E : 0000;	42 : 0000;
CONTENT BEGIN	07 : 101A;	0F : 0000;	43 : A6C7;
00 : 2001;	08 : 080B;	10 : 0000;	.....;
01 : 0032;	09 : 0000;	11 : 0000;	4F : 0000;
02 : 2002;	0A : 0000;	12 : 1524;	END;

9.3 KX9016v1指令系统设计

9.3.4 KX9016 v1控制器设计

1．程序结构

（1）CPU复位模块。

（2）指令辨认分支模块。

（3）指令处理状态序列。

2．指令的语句结构

【例 9-8】

```
library IEEE;
use IEEE.std_logic_1164.all;--以下的加粗文字是加入加法指令后的程序变化，以上有示例
entity CONTRLA is   --这里的clock对应图中的STEP
   port( clock : in std_logic;  reset : in std_logic; --时钟和复位
     instrReg : in std_logic_vector(15 downto 0);--指令寄存器操作码输入
     compout : in std_logic;                          --比较器结果输入
     progCntrWr : out std_logic;  --程序寄存器同步加载允许，但需T1的上升沿有效
     progCntrRd : out std_logic;  --程序寄存器数据输出至总线三态开关允许控制
     addrRegWr : out std_logic;   --地址寄存器允许总线数据锁入，但需T2有效
     addrRegRd : out std_logic;   --地址寄存器读入总线允许
     outRegWr : out std_logic;    --输出寄存器允许总线数据写入，但需T2有效
     outRegRd : out std_logic;    --输出寄存器数据进入总线允许，即打开三态门
     shiftSel : out std_logic_vector(2 downto 0);   --移位器功能选择
     aluSel : out std_logic_vector ( 3 downto 0 );  --ALU功能选择
     compSel : out std_logic_vector(2 downto 0);    --比较器功能选择
     opRegRd : out std_logic;  --工作寄存器读出允许
     opRegWr : out std_logic;  --总线数据允许锁入工作寄存器，但需T1有效
     instrWr : out std_logic;  --总线数据允许锁入指令寄存器，但需T2有效
     regSel : out std_logic_vector(2 downto 0);  --寄存器阵列选择
     regRd : out std_logic;     --寄存器阵列数据输出至总线三态开关允许控制
     regWr : out std_logic;     --总线上数据允许写入寄存器阵列，但需T2有效
     rw : out std_logic;         --rw=1，RAM写允许；rw=0，RAM读允许；
     vma : out std_logic);       --存储器RAM数据输出至总线三态开关允许控制;
 end CONTRLA;
architecture rtl of CONTRLA is
 constant shftpass: STD_LOGIC_VECTOR(2 DOWNTO 0) := "000"; --移位器直通
 constant alupass : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";--ALU直通
 constant    zero : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1001";--寄存器清零
```

```
constant     inc : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0111";--加 1
constant     plus : STD_LOGIC_VECTOR(3 DOWNTO 0)  :=  "0101"; --设定一个常数做加法
   type state is (reset1, reset2, reset3, execute, nop, load, store,
    load2, load3, load4, store2, store3, store4, incPc, incPc2, incPc3,
    loadI2,loadI3, loadI4,loadI5, loadI6, inc2, inc3,inc4,move1,move2,
     add2,add3,add4);  -- 在状态机中增加三个作加法微操作的状态变量元素。
    signal current_state, next_state : state; --定义现态和次态状态变量
    begin
 COM: process( current_state, instrReg, compout)   begin --组合进程
  progCntrWr<='0'; progCntrRd<='0'; addrRegWr<='0'; addrRegRd<='0';
  outRegWr<='0'; outRegRd<='0'; shiftSel<=shftpass; aluSel<=alupass;
  opRegRd<='0';  opRegWr<='0';  instrWr<='0'; regSel<="000";
   regRd<='0'; regWr<='0'; rw<='0'; vma<='0';
case current_state is
    when reset1=> aluSel<=zero; shiftSel<=shftpass;
                    outRegWr<='1'; next_state<=reset2;
    when reset2=> outRegRd<='1';  progCntrWr<='1';
                   addrRegWr<='1'; next_state<=reset3;
    when reset3=> vma<='1'; rw<='0'; instrWr<='1'; next_state<=execute;
    when execute=>
        case instrReg(15 downto 11) is      --不同指令识别分支处理
          when "00000" => next_state <= incPc;-- NOP指令
          when "00001" => next_state <= load2; -- LD指令
          when "00010" => next_state <= store2;-- STA指令
          when "00100" => progcntrRd <= '1'; alusel <= inc ;
                  shiftsel <= shftpass; next_state<=loadI2; --LDR指令
          when "00111" => next_state <= inc2; -- INC指令
          when "01101" => next_state <= add2; --增加一个加法 ADD指令分支
          when "00011" => next_state <= move1; -- MOVE指令
```

```
        when others =>next_state <= incPc;    --转 PC 加 1
      end case;
    when load2=> regSel<=instrReg(5 downto 3); regRd<='1';
                 addrregWr<='1'; next_state<=load3;
    when load3=>  vma<='1'; rw<='0'; regSel<=instrReg(2 downto 0);
                  regWr<='1'; next_state<=incPc;
    WHEN add2 => regSel<=instrReg(5 downto 3); --选择寄存器阵列的 R1;
                 regRd<='1';  --允许 R1 寄存器数据进入总线;
next_state<=add3; opRegWr<='1';--将此数据锁入工作寄存器。此 4 步在一个 STEP 脉冲完成
    WHEN add3 => regSel<=instrReg(2 downto 0); --选择寄存器阵列的 R2
 regRd<='1'; alusel<=plus; --允许 R2 寄存器数据进入总线，同时选择 ALU 作加法;
shiftsel<=shftpass; outRegWr<='1';--使 ALU 输出直通移位器，同时将数据锁入输出寄存器
 next_state<=add4;          --此时相加结果尚未进入总线。此 5 步在一个 STEP 脉冲完成。
    WHEN add4 => regSel<="011"; --选择寄存器阵列的 R3;
 outRegRd<='1'; regWr<='1';--允许输出寄存器的数据进入总线，将此数据锁入工作寄存器 R3。
      next_state<= incPc; --加法操作结束，最后转入作 PC 加 1 操作的状态。
    when move1 => regSel<=instrReg(5 downto 3); regRd<='1'; alusel <= alupass;
                  shiftsel<=shftpass; outregWr<='1'; next_state<=move2;
    when move2 => regSel<=instrReg(2 downto 0); outRegRd<='1';
                  regWr<='1'; next_state<=incPc;
    when store2 => regSel <= instrReg(2 downto 0); regRd <= '1';
                   addrregWr <= '1'; next_state <= store3;
    when store3 => regSel <= instrReg(5 downto 3); regRd <= '1';
                   rw <= '1'; next_state <= incPc;
    when loadI2 => progcntrRd <= '1'; alusel<=inc; shiftsel<=shftpass;
                   outregWr <= '1'; next_state<=loadI3;
    when loadI3 => outregRd <= '1'; next_state<=loadI4;
    when loadI4 => outregRd <= '1';  progcntrWr<='1'; addrregWr<='1';
                   next_state <= loadI5;
```

```
    when loadI5 => vma <= '1'; rw <= '0'; next_state <= loadI6;
    when loadI6 => vma <= '1'; rw <= '0';  regSel<=instrReg(2 downto 0);
                   regWr <= '1'; next_state <= incPc;
    when inc2 => regSel<=instrReg(2 downto 0); regRd<='1'; alusel<=inc;
                 shiftsel<=shftpass; outregWr<='1'; next_state<=inc3;
    when inc3 => outregRd <= '1'; next_state <= inc4;
    when inc4 => outregRd <= '1'; regsel <= instrReg(2 downto 0);
                regWr <= '1'; next_state <= incPc;
    when incPc => progcntrRd<='1'; alusel<=inc; shiftsel<=shftpass;
                  outregWr<='1'; next_state<=incPc2;
    when incPc2 => outregRd<='1'; progcntrWr <= '1'; addrregWr<='1';
                   next_state <= incPc3;
    when incPc3 => outregRd<='0'; vma<='1'; rw<='0';  instrWr<='1';
                   next_state<=execute;
    when others => next_state <= incPc;
    end case;
  end process;
REG: process(clock, reset)  begin   --时序进程
       if reset = '1' then  current_state <= reset1 ;
     elsif  rising_edge(clock) then  current_state<=next_state ; end if;
     end process;
end rtl;
```

9.3 KX9016v1指令系统设计

9.3.4 KX9016 v1控制器设计

3. CPU复位操作

（1）reset1。

（2）reset2

（3）reset3。

9.3 KX9016v1指令系统设计

9.3.5 指令设计实例详解

（1）确定功能。

```
ADD   Rs1, Rs2, R3
```

（2）确定指令的操作码。

（3）设定相关常数。

（4）增加状态元素。

（5）加入指令操作码译码语句。

（6）加入完成实际指令功能的状态转换语句。

（7）处理PC。

9.4 KX9016的时序仿真与硬件测试

9.4.1 时序仿真与指令执行波形分析

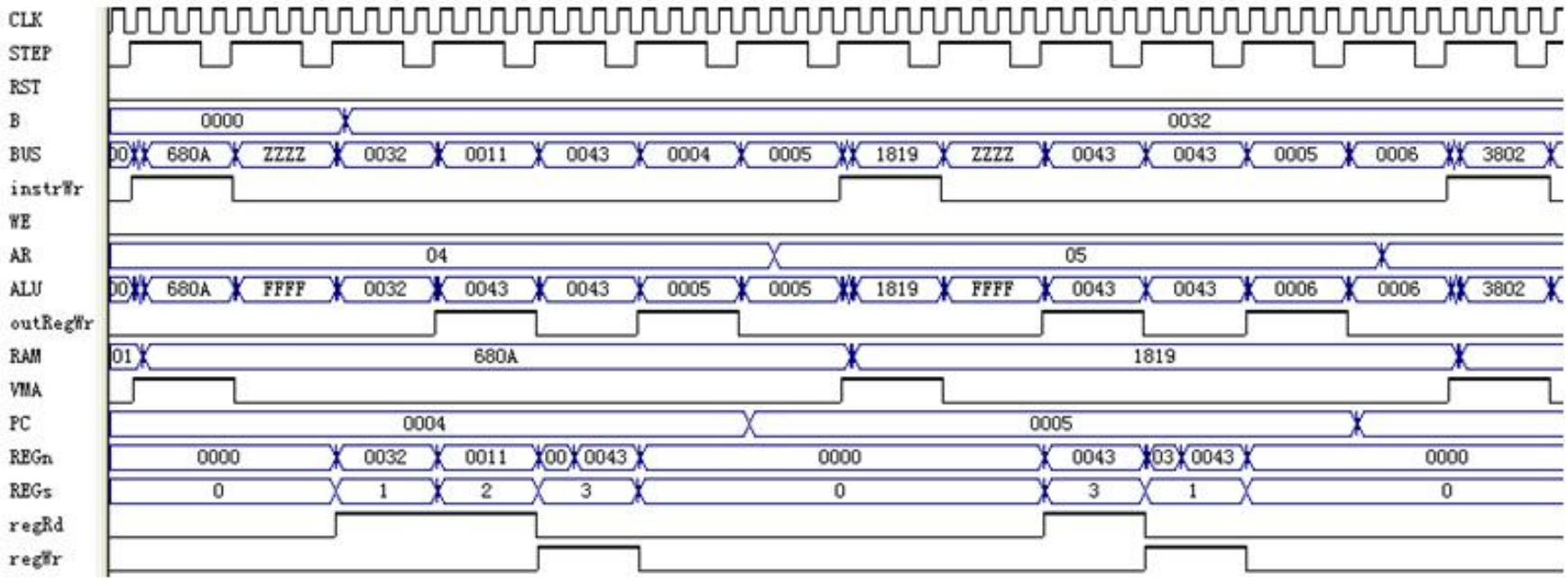

图 9-17　KX9016 的仿真波形，含 ADD 指令和 MOV 指令的时序

9.4 KX9016的时序仿真与硬件测试

9.4.1 时序仿真与指令执行波形分析

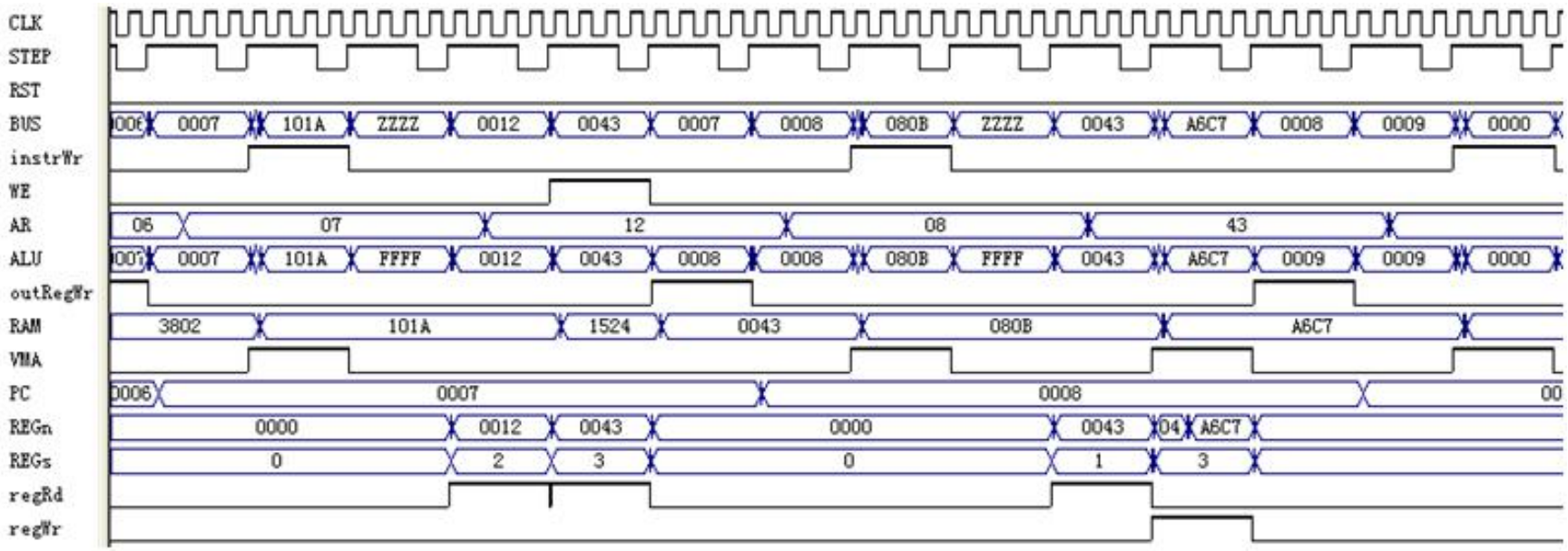

图 9-18　KX9016 的仿真波形，含 STA 指令和 LD 指令的时序

9.4 KX9016的时序仿真与硬件测试

9.4.2 CPU工作情况的硬件测试

1. 用SignalTap II测试与分析

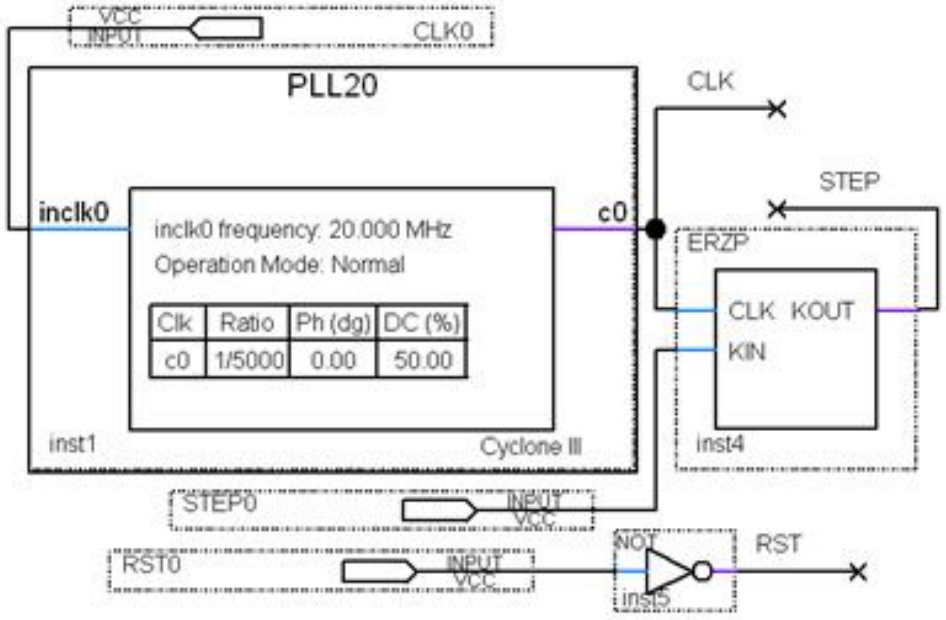

图 9-19　加入锁相环的电路方案

9.4 KX9016的时序仿真与硬件测试

9.4.2 CPU工作情况的硬件测试

1 用SignalTap II测试与分析

Instance	Status	LEs: 1385	Memory: 11264	M512,MLAB: 0/0	M4K,M9K: 4/260	M-RAM,M144K: 0/0
CPU16B	Waiting for trigger	1385 cells	11264 bits	0 blocks	3 blocks	0 blocks

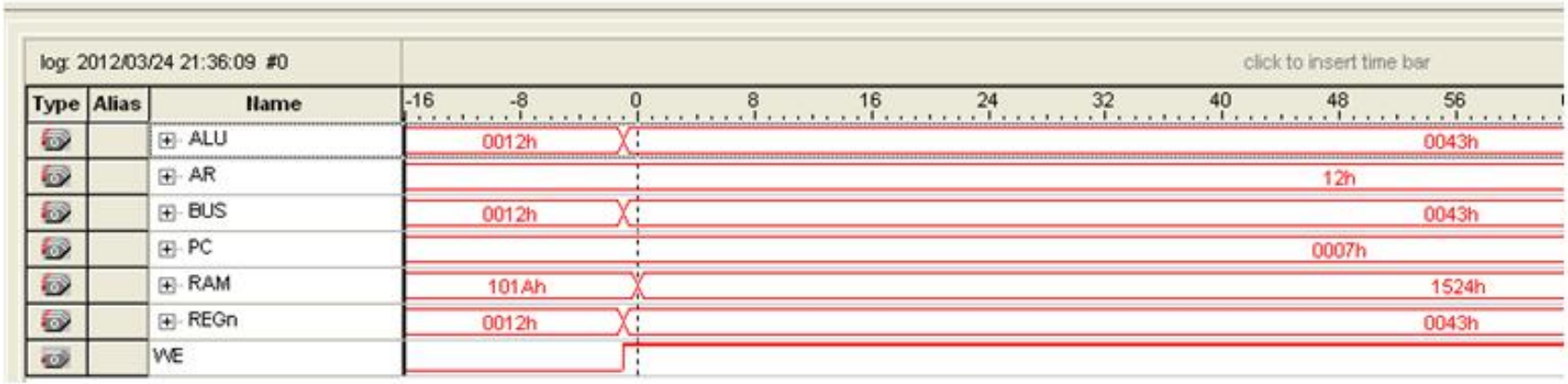

图 9-20 嵌入式锁相环对 KX9016 执行从 RAM 写数据指令的实时测试波形

9.4 KX9016的时序仿真与硬件测试

9.4.2 CPU工作情况的硬件测试

2．利用In-System Memory Content Editor进行实时测试

0 RAM8:

000000	20	01	00	32	20	02	00	11	68	0A	18	19	38	02	10	1A	08	0B	00	00	00	00	00	00	00	00	00	00	00	00
000010	00	00	00	00	00	43	00	00	00	00	00	00	00	00	00	00	00	00	00	00	B3	B4	B5	B6	B7	B8	D1	D2	D3	D4
000020	D5	D6	D7	D8	D9	DA	E1	E2	E3	E4	E5	E6	20	08	20	09	20	0A	20	0B	20	0C	20	0D	20	0E	20	0F	20	10	20	11
000030	D5	D6	D7	D8	D1	23	E1	E2	E3	E4	E5	E6	20	08	20	09	20	0A	20	0B	20	0C	00	00	00	00	00	00	00	00	00	00
000040	00	00	00	00	00	00	A6	C7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

图 9-21　In-System Memory Content Editor 对 KX9016 内 RAM 数据变化情况的实测情况

9.4 KX9016的时序仿真与硬件测试

9.4.2 CPU工作情况的硬件测试

3．利用In-System Sources & Probes进行实时测试

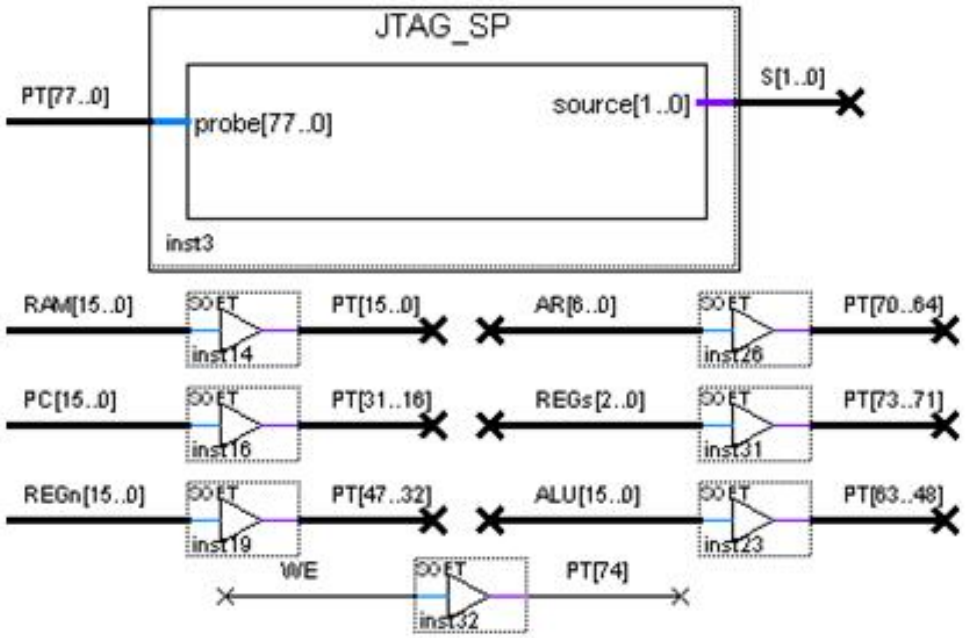

图 9-22　S/P 模块对 KX9016 端口的连接情况

9.4 KX9016的时序仿真与硬件测试

9.4.2 CPU工作情况的硬件测试

3. 利用In-System Sources & Probes进行实时测试

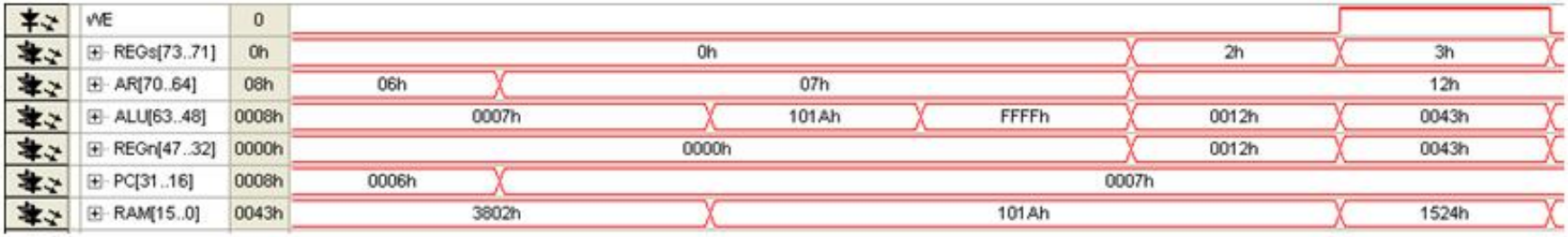

图 9-23　在系统 S/P 模块对 KX9016 执行从 RAM 写数据指令 STA 的实时测试波形

9.5 KX9016应用程序设计实例和系统优化

9.5.1 除法算法及其硬件实现

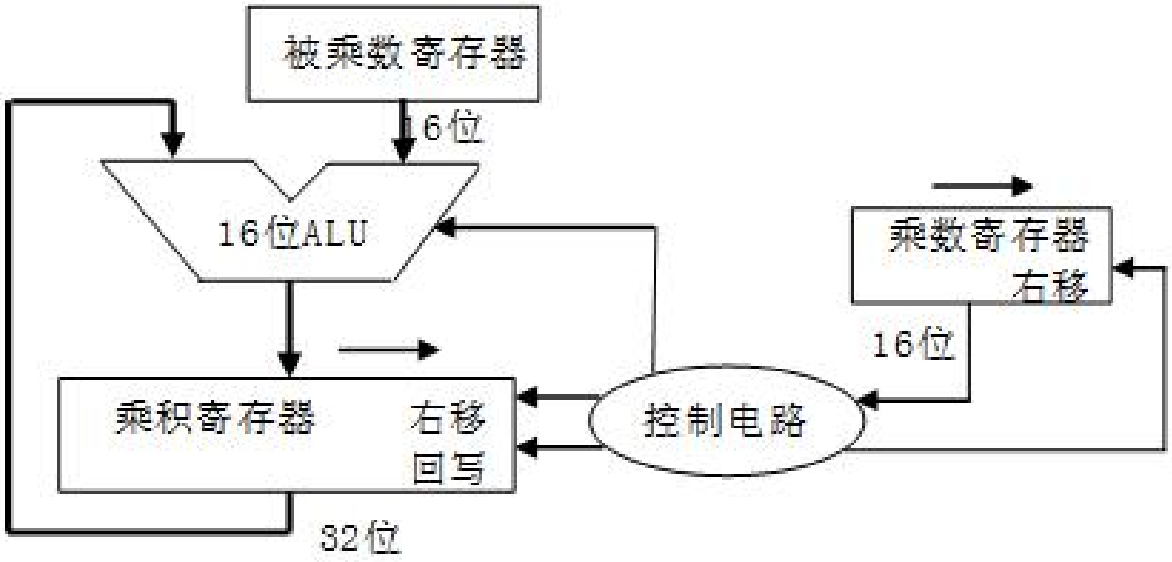

图 9-24 乘法算法 1 的硬件实现

9.5 KX9016应用程序设计实例和系统优化

9.5.1 除法算法及其硬件实现

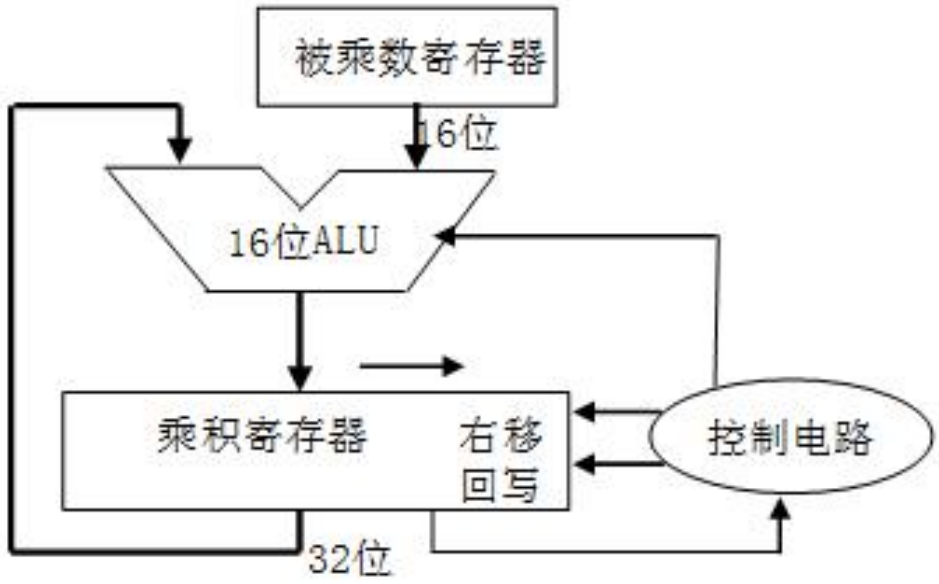

图 9-25 改进后的乘法算法 2 的硬件实现

9.5 KX9016应用程序设计实例和系统优化

9.5.1 除法算法及其硬件实现

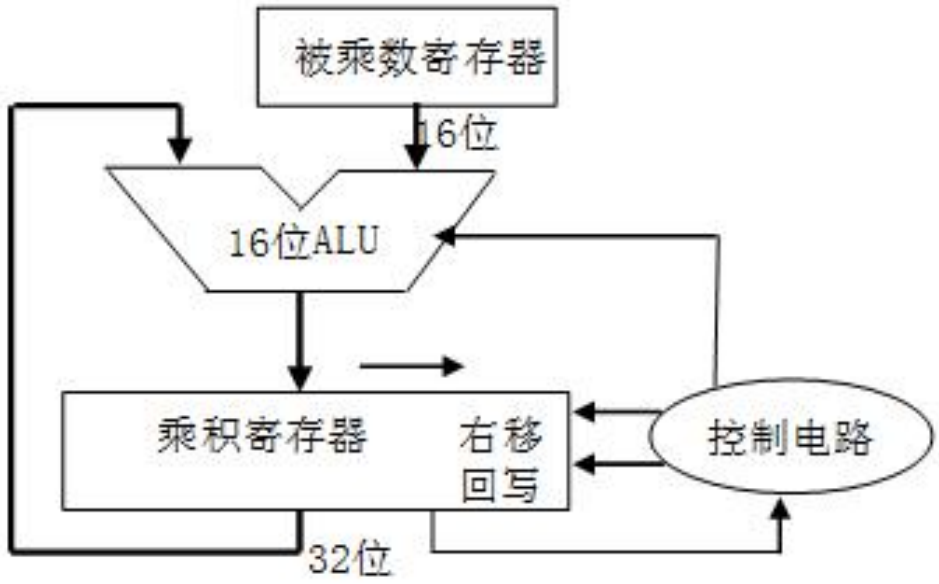

图 9-25 改进后的乘法算法 2 的硬件实现

9.5 KX9016应用程序设计实例和系统优化

9.5.1 除法算法及其硬件实现

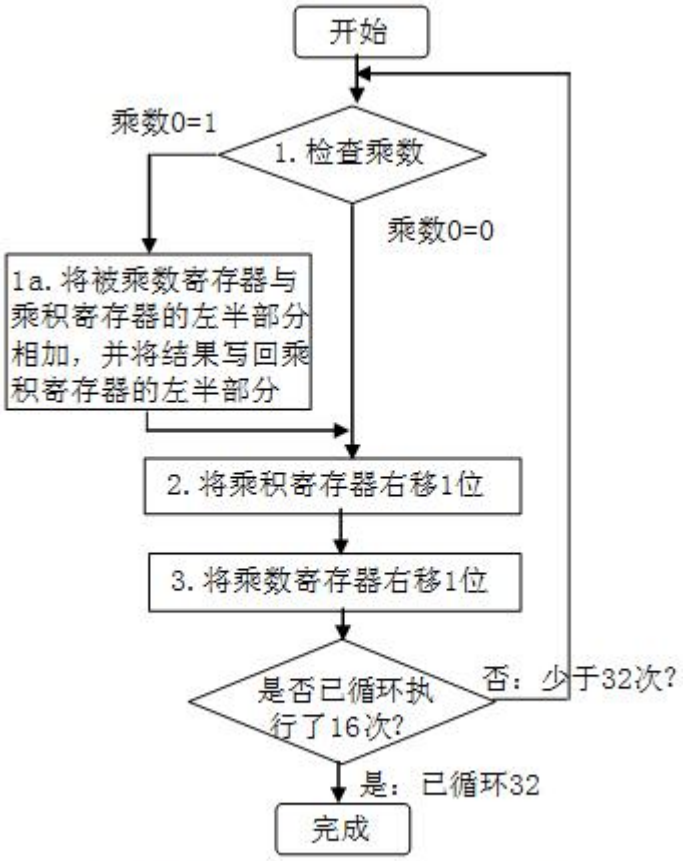

图 9-26 乘法算法 1 的流程图

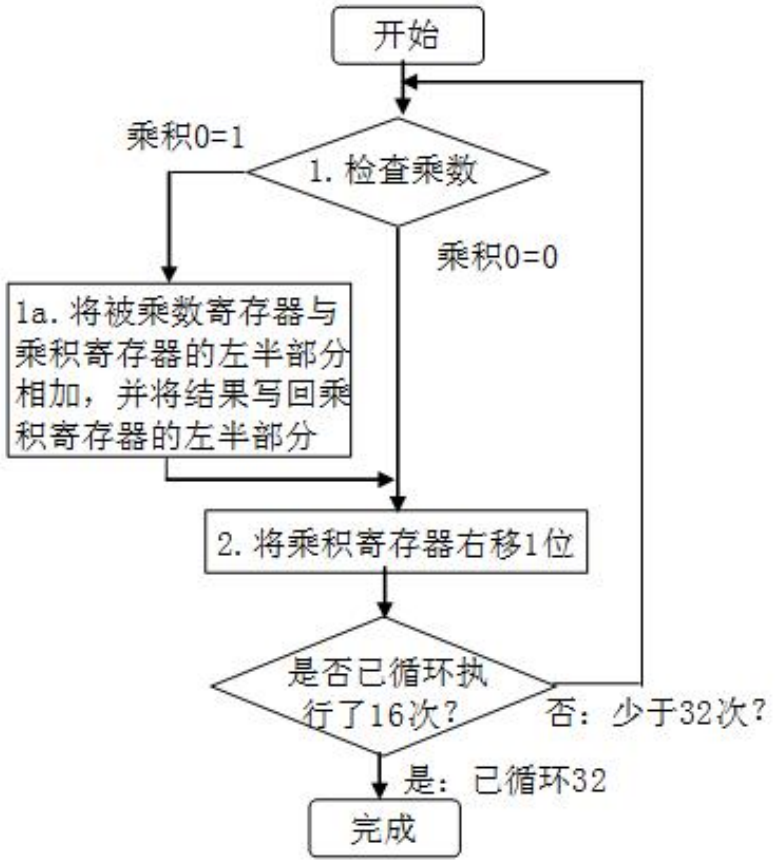

图 9-27 乘法算法 2 的流程图

9.5 KX9016应用程序设计实例和系统优化

9.5.2 除法算法及其硬件实现

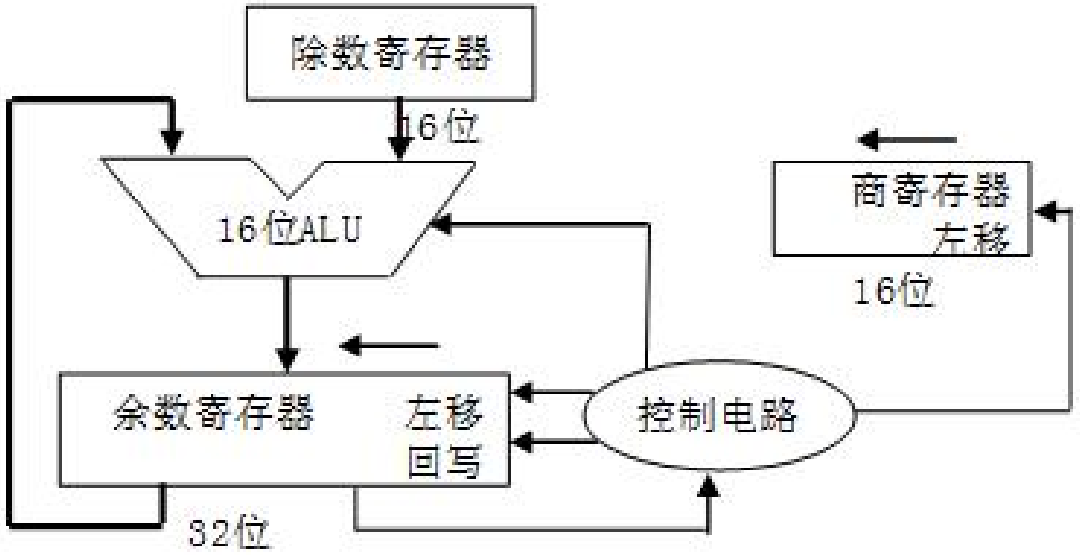

图 9-28 除法算法 1 的硬件结构

9.5 KX9016应用程序设计实例和系统优化

9.5.2 除法算法及其硬件实现

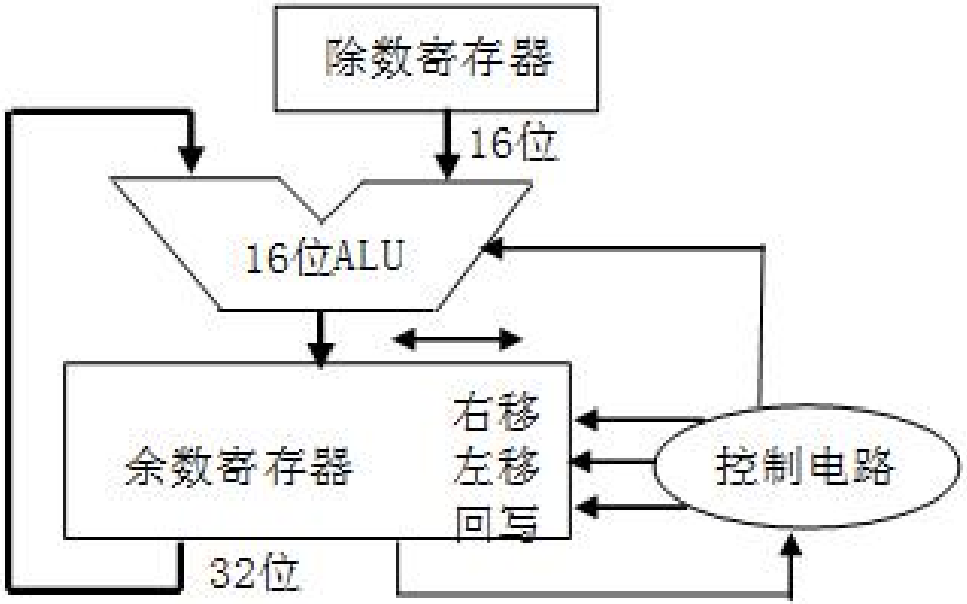

图 9-29 除法算法 2 的硬件结构

9.5 KX9016应用程序设计实例和系统优化

9.5.3 KX9016v1的硬件系统优化

算法优化。

① 软件方案。

② 硬件指令替代软件程序的S2H方案。

③ 调用专用乘法器硬件模块。

实验与设计

实验9-1 16位CPU验证性设计综合实验

实验9-2 新指令设计及程序测试实验

RAM_16.mif

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	2001	0021	2002	0058	2006	0040	080B	101A
08	300E	0000	3801	3802	2800	0006	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000
18	0000	0000	0000	0000	0000	0000	0000	0000
20	0000	FF00	4321	5432	6543	4433	5544	2DFF
28	1234	2345	3456	2030	3033	3068	2D2D	3466
30	A1A2	B2B3	C3C4	D5D6	E6E7	F8F9	ABCD	EF01
38	1212	2323	3434	5656	7878	8989	ABAB	CDCD
40	EFEF	0000	0000	0000	0000	0000	0000	0000
48	0000	0000	0000	0000	0000	0000	0000	0000
50	0000	0000	0000	0000	0000	0000	0000	0000
58	0000	0000	0000	0000	0000	0000	0000	0000

图 9-30 编辑 ram_16.mif 文件

实验与设计

实验9-2 新指令设计及程序测试实验

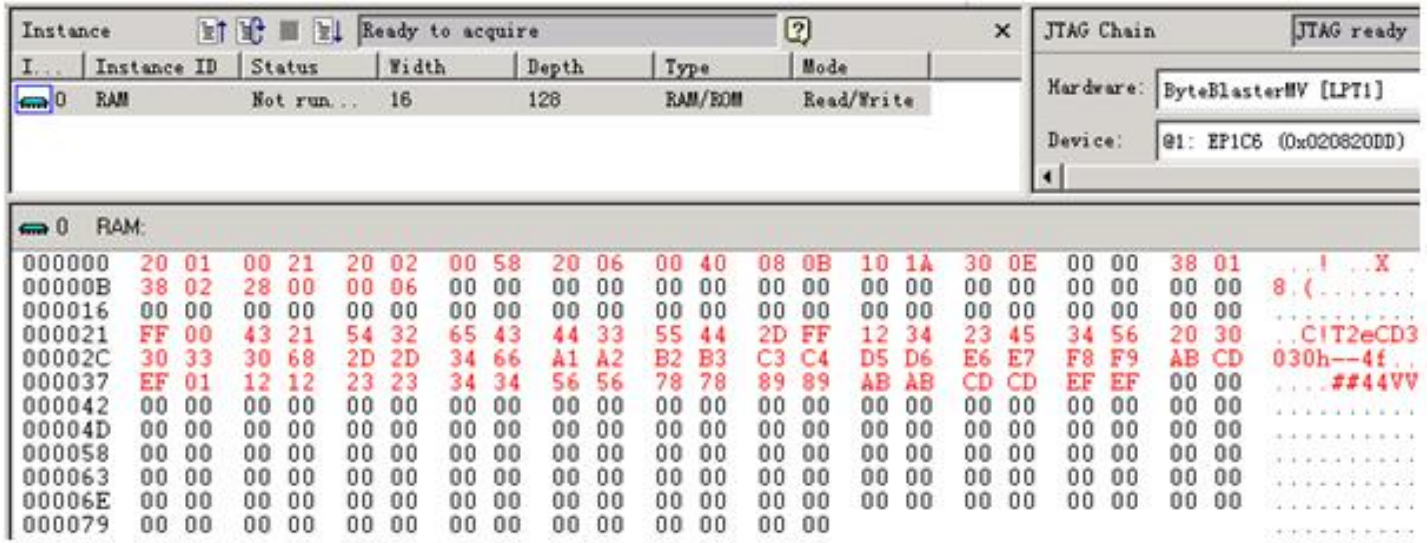

图 9-31 用 In-System Memory Content Editor 读取的数据

实验9-3 16位CPU的优化设计与创新

实验与设计

9-3. 16位CPU的优化设计与创新

9-4. CPU创新设计竞赛