第9章

16位CPU创新设计

9.1 KX9016的结构与特色

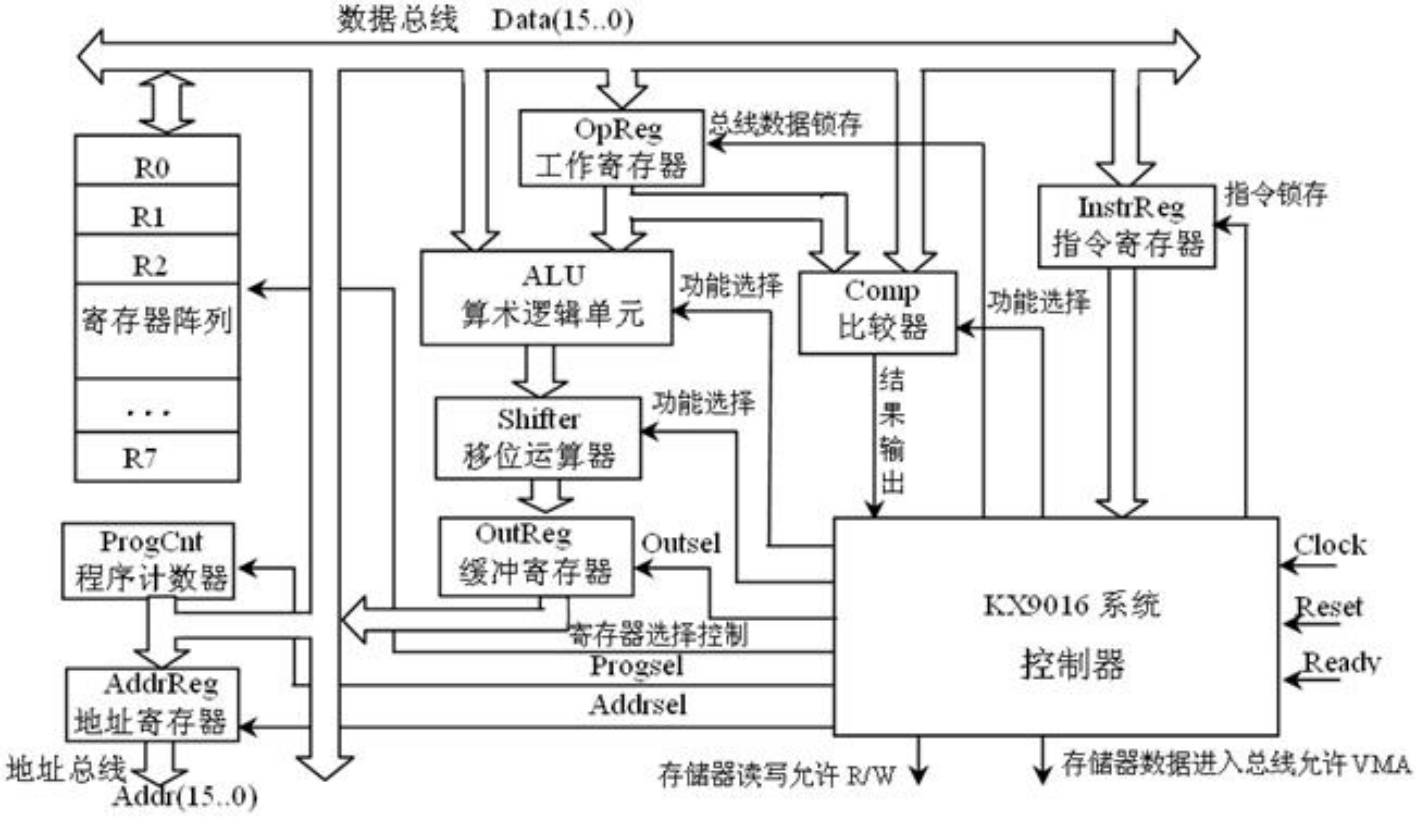

图 9-1 16 位 KX9016v1 CPU 顶层结构图

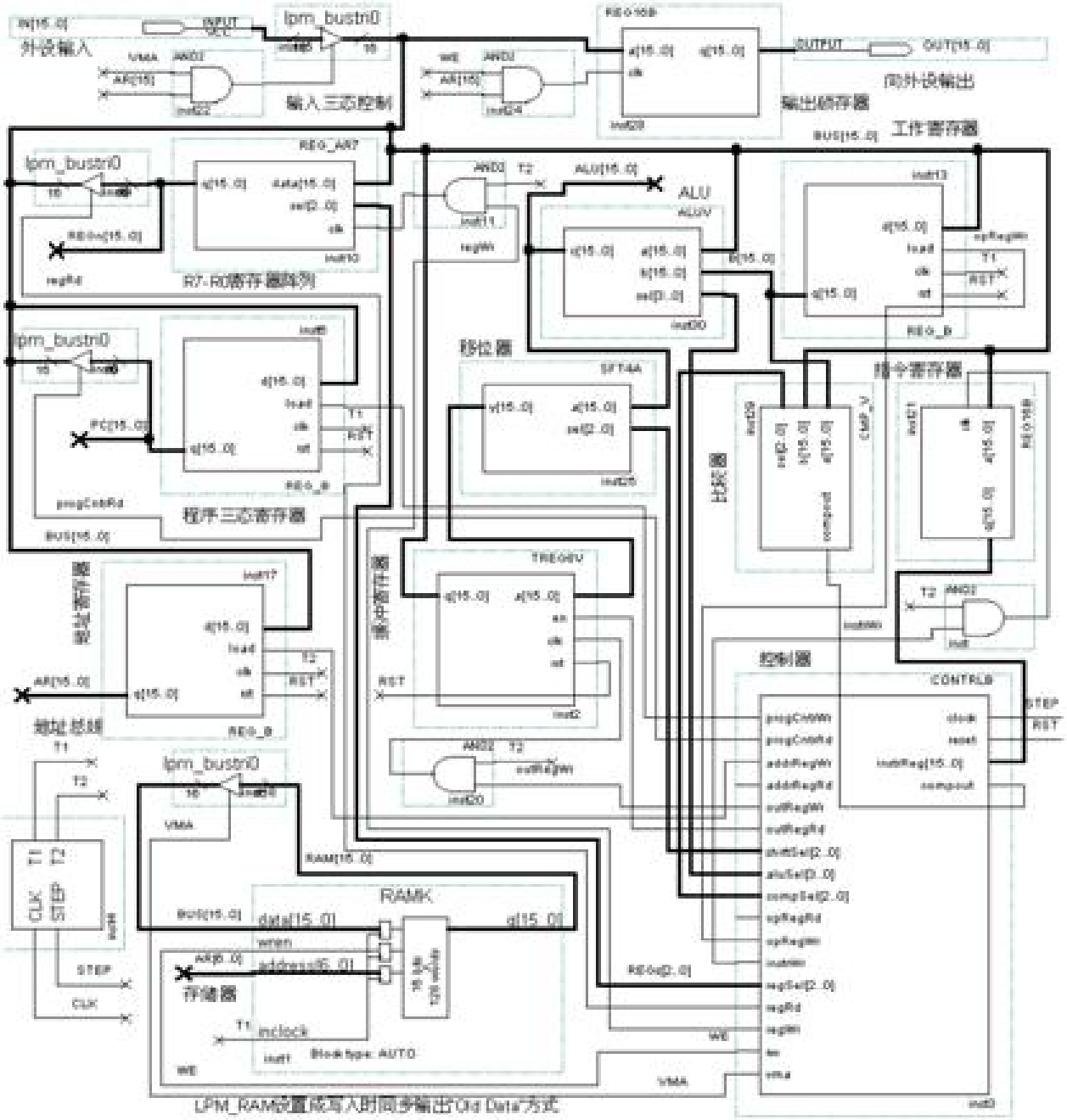

图 9-2　KX9016v1 顶层电路原理结构图。

9.2 KX9016基本硬件系统设计

9.2.1 单步节拍发生模块

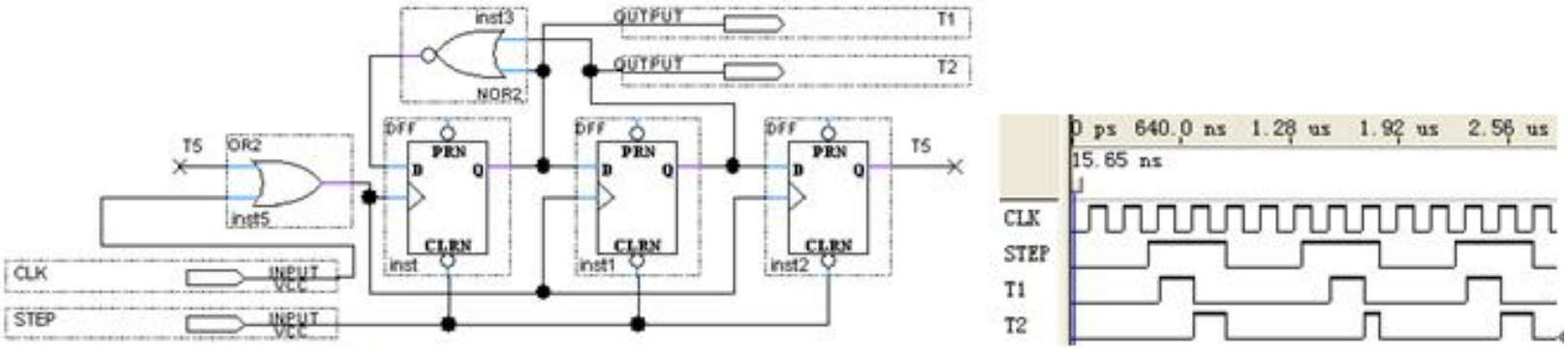

图 9-3 节拍脉冲发生器 STEP2 的电路及其仿真波形图

9.2 KX9016基本硬件系统设计

9.2.2 ALU模块

【例 9-1】

```
module ALUV (input[15:0] a, b, input[3:0] sel, output reg [15:0] c);
  parameter  alupass=0, andOp=1, orOp=2, notOp=3, xorOp=4, plus=5,
 alusub=6,  inc=7, dec=8, zero=9;
  always @(a or b or sel)
    case (sel)
      alupass : c <= a ;          //总线数据直通ALU
       andOp : c <= a & b ;       //逻辑与操作
        orOp : c <= a | b ;       //逻辑或操作
       xorOp : c <= a ^ b ;       //逻辑异或操作
       notOp : c <= ~a ;          //取反操作
        plus : c <= a + b ;       //算术加操作
      alusub : c <= a - b ;       //算术减操作
         inc : c <= a + 1 ;       //加1操作
         dec : c <= a - 1 ;       //减1操作
        zero : c <= 0 ;           //输出清0
      default : c <= 0 ;
  endcase          endmodule
```

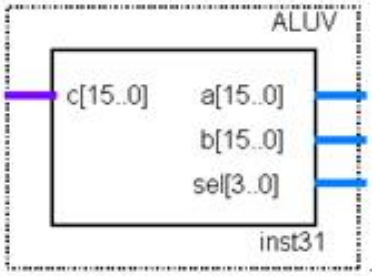

图 9-4 ALU 模块

9.2 KX9016基本硬件系统设计

9.2.3 比较器模块

【例 9-2】

```
module CMP_V (input[15:0] a, b,  input[2:0] sel,  output reg compout);
   parameter eq=0, neq=1, gt=2, gte=3, lt=4, lte =5;
  always @(a or b or sel)
    case (sel)
      eq : if (a==b)  compout<=1; else  compout<=0; //a等于b，输出为1，负责是0
     neq : if (a!=b)  compout<=1; else  compout<=0; //a不等于b，输出为1
      gt : if (a>b)   compout<=1; else  compout<=0;  //a大于b，输出为1
     gte : if (a>=b)  compout<=1; else  compout<=0; //a大于等于b，输出为1
      lt : if (a<b)   compout<=1; else  compout<=0; //a小于b，输出为1
     lte : if (a<=b)  compout<=1; else  compout<=0; //a小于等于b，输出为1
     default :        compout<=0;
     endcase        endmodule
```

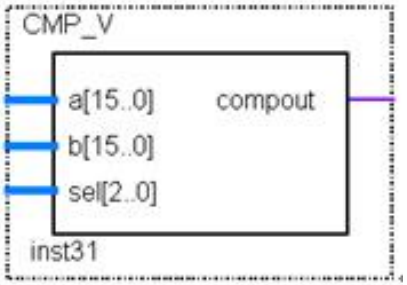

图 9-5 比较器模块符号

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

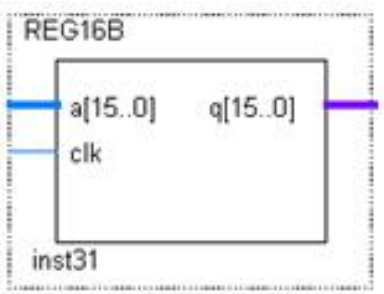

图 9-6 基本寄存器

（1）只含锁存控制时钟的寄存器

（2）含三态输出控制的寄存器

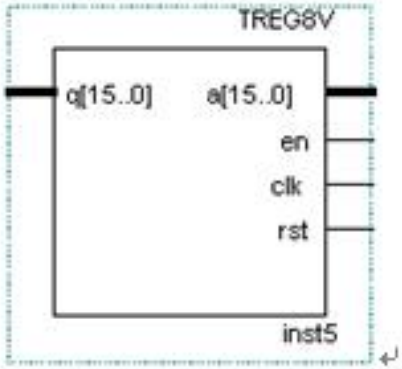

图 9-7 含三态门的寄存器

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

1. 基本寄存器

（3）含清零和数据锁存同步使能控制的寄存器

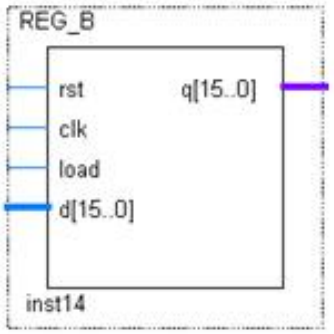

图 9-8 含加载使能的寄存器

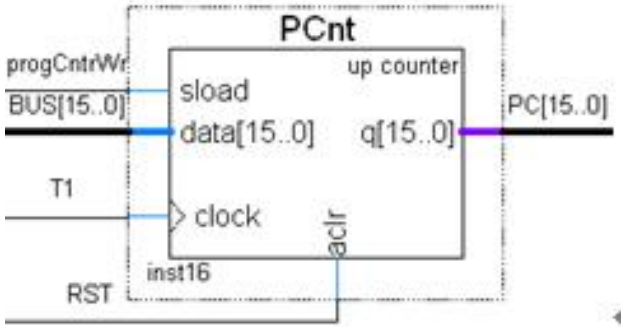

图 9-9 PC 替代电路

9.2 KX9016基本硬件系统设计

（3）含清零和数据锁存同步使能控制的寄存器

【例 9-3】

```
module REG16B (input[15:0] a, input clk, output reg [15:0] q);
  always @(posedge clk)     q <= a ;   endmodule
```

【例 9-4】

```
module TREG8V (a, en, clk,rst, q);
  input rst,en, clk; input[15:0] a;   output[15:0] q;  reg[15:0] q, val;
   always @(posedge clk or posedge rst)
     if (rst==1'b1)  val <= {16{1'b0}} ;  else  val <= a ;
   always @(en or val)
     if (en == 1'b1)  q <= val ;  else  q <= 16'bZZZZZZZZZZZZZZZZ ;
     endmodule
```

【例 9-5】REG_B.v

```
module REG_B (input rst, clk, load, input[15:0]d, output reg [15:0] q);
   always @(posedge clk or posedge rst)
     if (rst==1'b1)  q <= {16{1'b0}} ;    else
     begin  if (load == 1'b1)   q <= d ;  end       endmodule
```

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

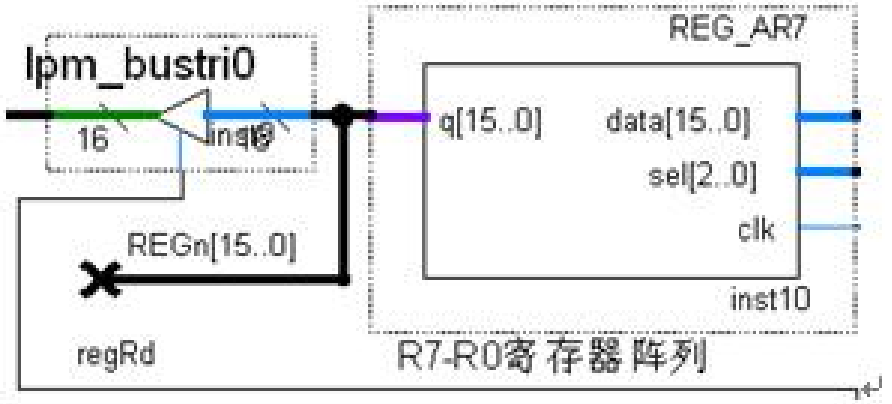

图 9-10 寄存器阵列元件与三态控制门电路

9.2 KX9016基本硬件系统设计

9.2.4 基本寄存器与寄存器阵列组

2. 寄存器阵列

【例 9-6】

```
module REG_AR7 (data, sel, clk, q);
   input[15:0] data;  input[2:0] sel; input clk;   output[15:0] q;
   reg[15:0] ramdata[0:7];
   always @(posedge clk)    ramdata[sel] <= data;
   assign q = ramdata[sel] ;     endmodule
```

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

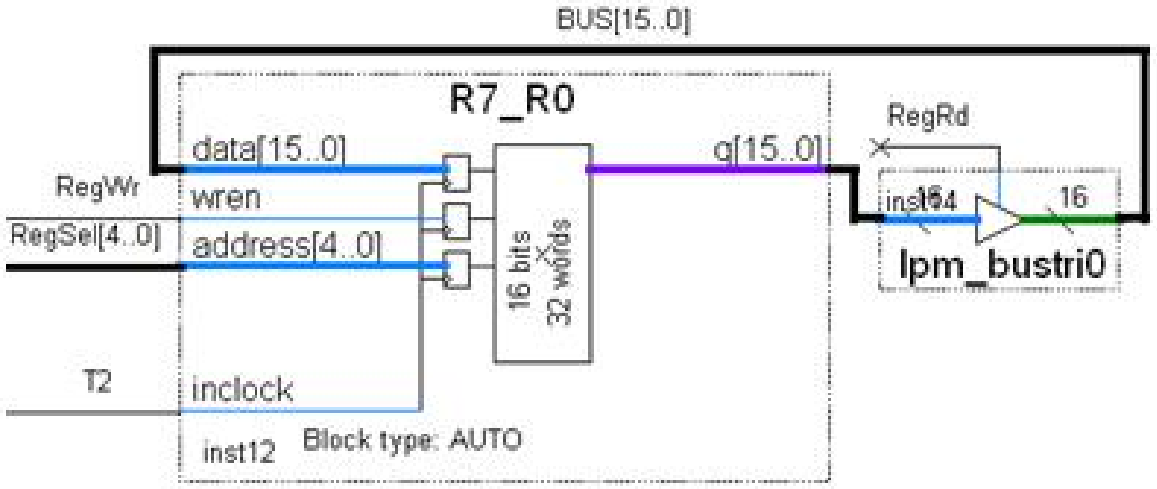

图 9-11 用 LPM_RAM 替代寄存器阵列的电路

9.2 KX9016基本硬件系统设计

9.2.5 移位器模块

【例 9-7】

```
module SFT4A (input[15:0] a, input[2:0] sel, output reg[15:0] y);
   parameter shftpass=0, sftl=1, sftr=2, rotl=3, rotr=4;
   always @(a or sel)
    case (sel)
    shftpass :  y<=a ;                    //数据直通
        sftl :  y<={a[14:0], 1'b0};       //左移
        sftr :  y<={1'b0, a[15:1]};       //右移
        rotl :  y<={a[14:0], a[15]};      //循环左移
        rotr :  y<={a[0], a[15:1]};       //循环右移
     default :  y<=0 ;
    endcase        endmodule
```

图 9-12 移位器符号

9.2 KX9016基本硬件系统设计

9.2.6 程序与数据存储器模块

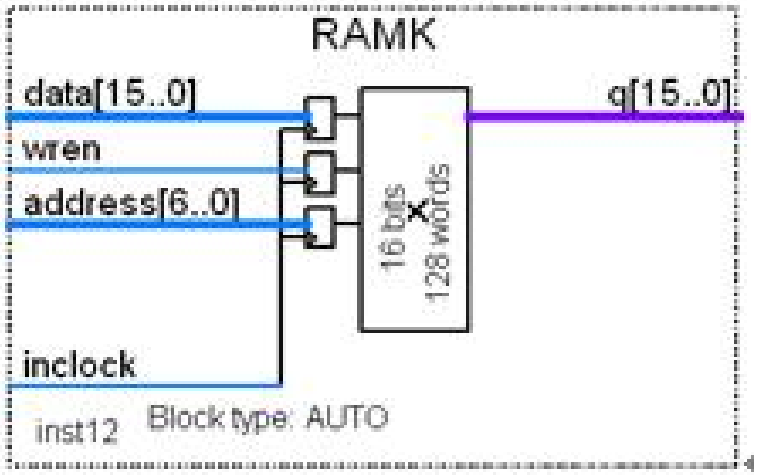

图 9-13 存储器符号

9.3 KX9016v1指令系统设计

9.3.1 指令格式

（1）单字指令。

表 9-1 单字指令格式

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

9.3 KX9016v1指令系统设计

9.3.1 指令格式

（2）双字指令。

表 9-2 双字指令格式

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

表 9-3 双字指令

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			

9.3 KX9016v1指令系统设计

9.3.2 指令操作码

表 9-4 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 9-5 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 0000000000000000	110	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 0000000000000110	xxx	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 软件程序设计示例

表 9-6 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 软件程序设计示例

表 9-7 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 KX9016v1控制器设计

1. 程序代码结构

2. 指令的语句结构

3. CPU复位操作

【例 9-8】

```
module CONTRLB (clock, reset, instrReg, compout, progCntrWr, progCntrRd,
 addrRegWr, addrRegRd, outRegWr, outRegRd, shiftSel, aluSel, compSel,
 opRegRd, opRegWr, instrWr, regSel, regRd, regWr, rw, vma);
  input clock;   input reset;      //时钟和复位信号
  input[15:0] instrReg; input compout; //指令寄存器操作码输入及比较器结果输入
  output progCntrWr;                //程序寄存器同步加载允许，但需 T1 的上升沿有效
  output progCntrRd;               //程序寄存器数据输出至总线三态开关允许控制
  output addrRegWr;                //地址寄存器允许总线数据锁入，但需 T2 有效
  output addrRegRd;                //地址寄存器读入总线允许
  output outRegWr;                 //输出寄存器允许总线数据写入，但需 T2 有效
  output outRegRd;                 //输出寄存器数据进入总线允许，即打开三态门
  output[2:0] shiftSel; output[3:0] aluSel;    //移位器功能选择和 ALU 功能选择
  output[2:0] compSel;  output opRegRd;    //比较器功能选择和工作寄存器读出允许
  output opRegWr;                   //总线数据允许锁入工作寄存器，但需 T1 有效
  output instrWr;                   //总线数据允许锁入指令寄存器，但需 T2 有效
  output[2:0] regSel;               //寄存器阵列选择
  output regRd;                     //寄存器阵列数据输出至总线三态开关允许控制
  output regWr;                    //总线上数据允许写入寄存器阵列，但需 T2 有效
  output rw;                        //rw=1，RAM 写允许；rw=0，RAM 读允许；
  output vma;                      //存储器 RAM 数据输出至总线三态开关允许控制;
```

接下页

```
  reg progCntrWr, progCntrRd, addrRegWr, addrRegRd, outRegWr, outRegRd,
      opRegRd, opRegWr, instrWr, regRd, vma, regWr, rw;
  reg[2:0] shiftSel, regSel;  wire[2:0] compSel;   reg[3:0] aluSel;
  parameter shftpass=0,  alupass=0, zero=9, inc=7, plus=5; //参见程序例 8-1
 parameter reset1=0, reset2=1, reset3=2, execute=3, nop=4, load=5, store=6,
load2=7, load3=8, load4=9, store2=10, store3=11, store4=12, incPc=13, incPc2=14,
incPc3=15, loadI2=19, loadI3=20, loadI4=21, loadI5=22, loadI6=23, inc2=24,
inc3=25, inc4=26, move1=27, move2=28, add2=29, add3=30, add4=31;
    //在状态机中增加三个作加法微操作的状态变量元素 add2, add3, add4。
  reg[4:0] current_state, next_state; //定义现态和次态状态变量
  always @(current_state or instrReg or compout)   begin : COM //组合过程
    progCntrWr<=0; progCntrRd<=0; addrRegWr<=0; addrRegRd<=0; outRegWr<=0;
 outRegRd<=0; shiftSel<=shftpass; aluSel<=alupass;   opRegRd<=0; opRegWr<=0;
 instrWr<=0; regSel<=0;  regRd<=0;  regWr<=0; rw<=0; vma<=0;
    case (current_state)
    reset1 :  begin  aluSel<=zero; shiftSel<=shftpass;
                     outRegWr<=1'b1; next_state<=reset2;    end
    reset2 :  begin  outRegRd<=1'b1; progCntrWr<=1'b1;
                     addrRegWr<=1'b1; next_state<=reset3;   end
    reset3 :  begin  vma<=1'b1; rw<=1'b0; instrWr<=1'b1;
                     next_state<=execute;                   end
   execute : begin
```

接下页

```
      case (instrReg[15:11])                          //不同指令识别分支处理
        5'b00000 : next_state <= incPc ;              // 转 nop 指令处理
        5'b00001 : next_state <= load2 ;              // 转 load 指令处理
        5'b00010 : next_state <= store2 ;             // 转 store 指令处理
        5'b00100 : begin progCntrRd<=1'b1; aluSel<=inc; shiftSel<=shftpass;
                   next_state<=loadI2; end            //转 loadI 指令处理
        5'b00111 : next_state <= inc2 ;               //转 inc 指令处理
        5'b01101 : next_state <= add2 ;               //增加一个加法 ADD 指令分支
        5'b00011 : next_state <= move1 ;              // 转 move 指令处理
         default : next_state <= incPc ;              //转 PC 加 1
       endcase              end
    load2 :  begin  regSel<=instrReg[5:3]; regRd<=1'b1;
                    addrRegWr<=1'b1; next_state<=load3; end
    load3 :  begin  vma<=1'b1; rw<=1'b0; regSel<=instrReg[2:0];
                    regWr<=1'b1; next_state<=incPc; end
  add2 :  begin  regSel <= instrReg[5:3];     //选择寄存器阵列的 R1;
                 regRd <= 1'b1 ;                //允许 R1 寄存器数据进入总线;
           next_state<=add3; opRegWr<=1'b1;  end//将此数据锁入工作寄存器。
                                    //以上 4 步在一个 STEP 脉冲完成，以下同样。
  add3 :  begin  regSel <= instrReg[2:0] ; //选择寄存器阵列的 R2
        regRd<=1'b1; 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'b1;//使 ALU 输出直通移位器，同时将数据锁入输出寄存器
     next_state<=add4; end  //此时相加结果尚未进入总线。此 5 步在一个 STEP 脉冲完成。
  add4 :   begin  regSel <= 3'b011 ; //固定选择寄存器阵列的 R3;
outRegRd<=1'b1; regWr<=1'b1;//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。
```

接下页

```
next_state <= incPc ; end //加法操作结束，最后转入作 PC 加 1 操作的状态。
move1 : begin  regSel<=instrReg[5:3]; regRd<=1'b1; aluSel<=alupass;
          shiftSel<=shftpass; outRegWr<=1'b1; next_state<=move2; end
move2 : begin  regSel<=instrReg[2:0]; outRegRd<=1'b1;
               regWr<=1'b1; next_state<=incPc;  end
store2 : begin  regSel<=instrReg[2:0]; regRd<=1'b1;
                addrRegWr<=1'b1;  next_state<=store3; end
store3 : begin  regSel<=instrReg[5:3]; regRd<=1'b1;
                rw<=1'b1;  next_state<=incPc;  end
loadI2 : begin  progCntrRd<=1'b1;  aluSel<=inc ; shiftSel<=shftpass;
               outRegWr <= 1'b1 ;  next_state <= loadI3 ;   end
loadI3 : begin  outRegRd<=1'b1;  next_state<=loadI4;       end
loadI4 : begin  outRegRd <= 1'b1 ;  progCntrWr <= 1'b1 ;
               addrRegWr <= 1'b1 ;   next_state <= loadI5 ; end
loadI5 : begin  vma<=1'b1; rw<=1'b0;  next_state<=loadI6;  end
loadI6 : begin  vma <= 1'b1;  rw <= 1'b0 ;  regSel <= instrReg[2:0] ;
               regWr <= 1'b1 ;  next_state <= incPc ;       end
inc2 : begin  regSel <= instrReg[2:0] ; regRd <= 1'b1 ; aluSel <= inc ;
           shiftSel<=shftpass;  outRegWr<=1'b1; next_state<=inc3;  end
inc3 : begin  outRegRd <= 1'b1 ; next_state <= inc4 ;  end
inc4 : begin  outRegRd<=1'b1; regSel<=instrReg[2:0];
                regWr<=1'b1; next_state<=incPc; end
incPc : begin  progCntrRd<=1'b1;  aluSel<=inc;  shiftSel<=shftpass;
               outRegWr <= 1'b1 ;  next_state <= incPc2 ; end
```

接下页

9.3 KX9016v1指令系统设计

9.3.4 KX9016 v1控制器设计

```
  incPc2 : begin  outRegRd <= 1'b1 ;  progCntrWr <= 1'b1 ;
                  addrRegWr <= 1'b1 ;   next_state <= incPc3 ;  end
  incPc3 : begin  outRegRd <= 1'b0 ;  vma <= 1'b1 ; rw <= 1'b0 ;
                 instrWr <= 1'b1 ;  next_state <= execute ;    end
  default :      next_state <= incPc ;
  endcase   end
 always @(posedge clock or posedge reset)   //时序过程
    if (reset==1) current_state<=reset1; else current_state<=next_state;
endmodule
```

9.3 KX9016v1指令系统设计

9.3.5 指令设计示例

（1）确定功能。

（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

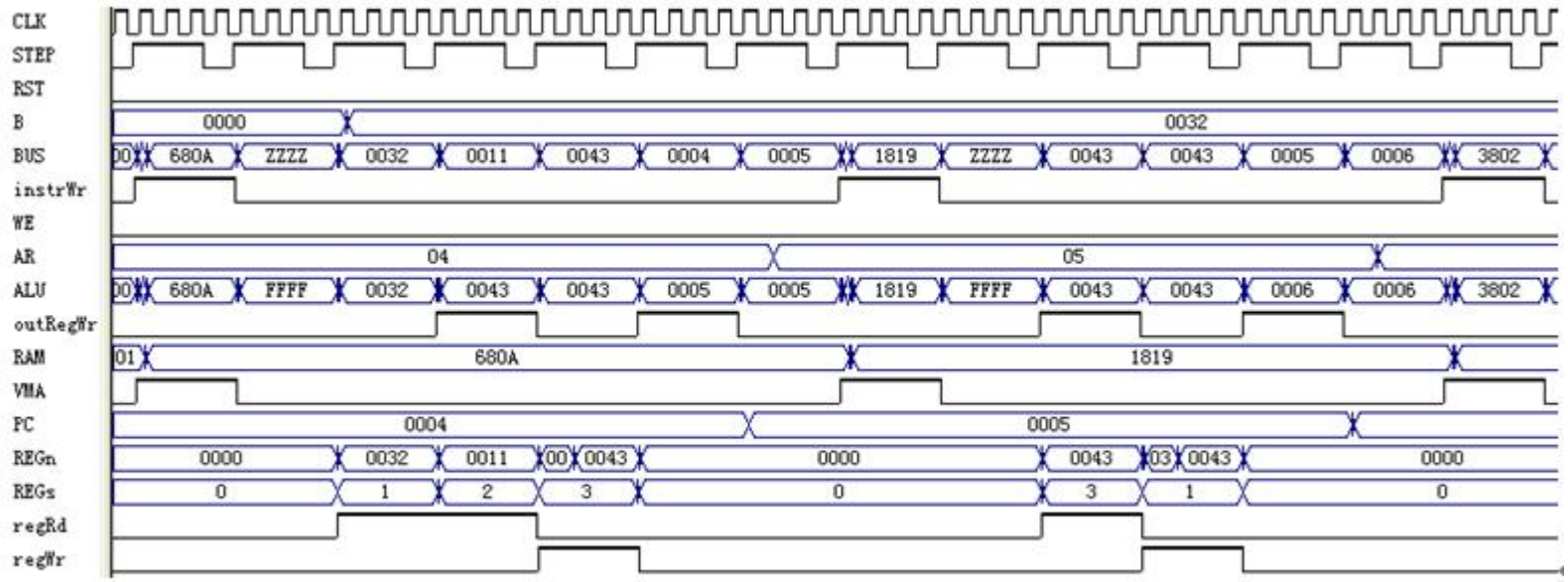

图 9-14 KX9016 的仿真波形，含 ADD 指令和 MOV 指令的时序

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

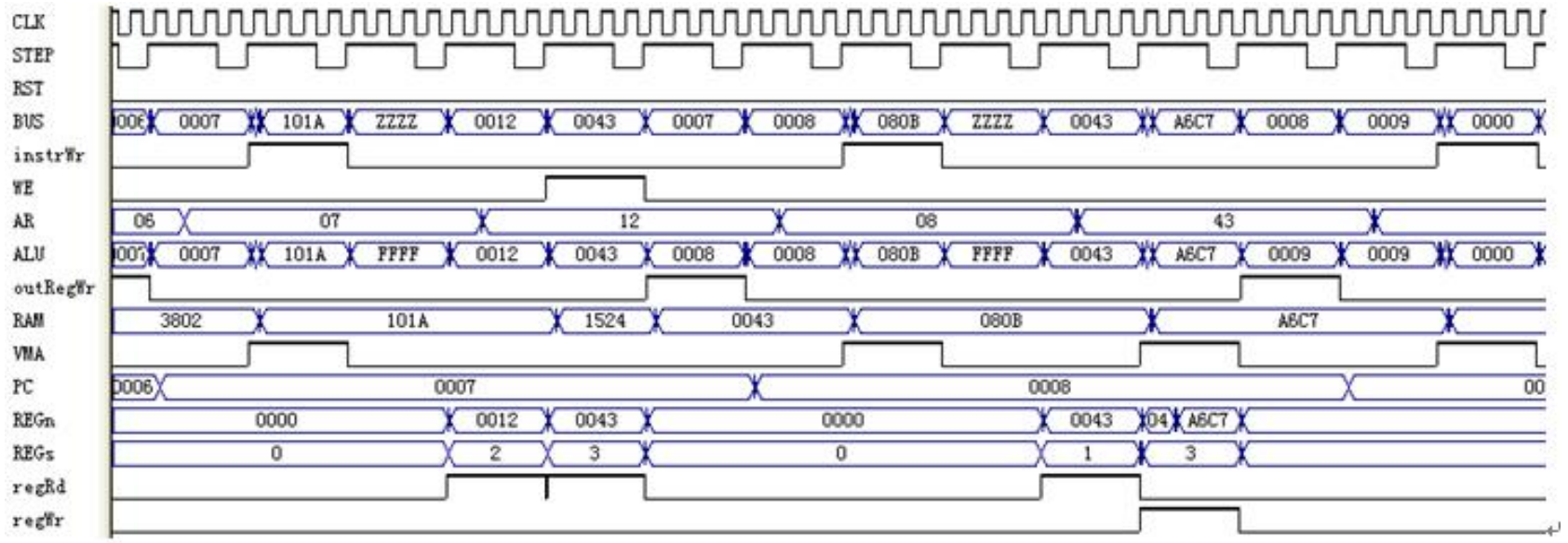

图 9-15　KX9016 的仿真波形，含 STA 指令和 LD 指令的时序

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

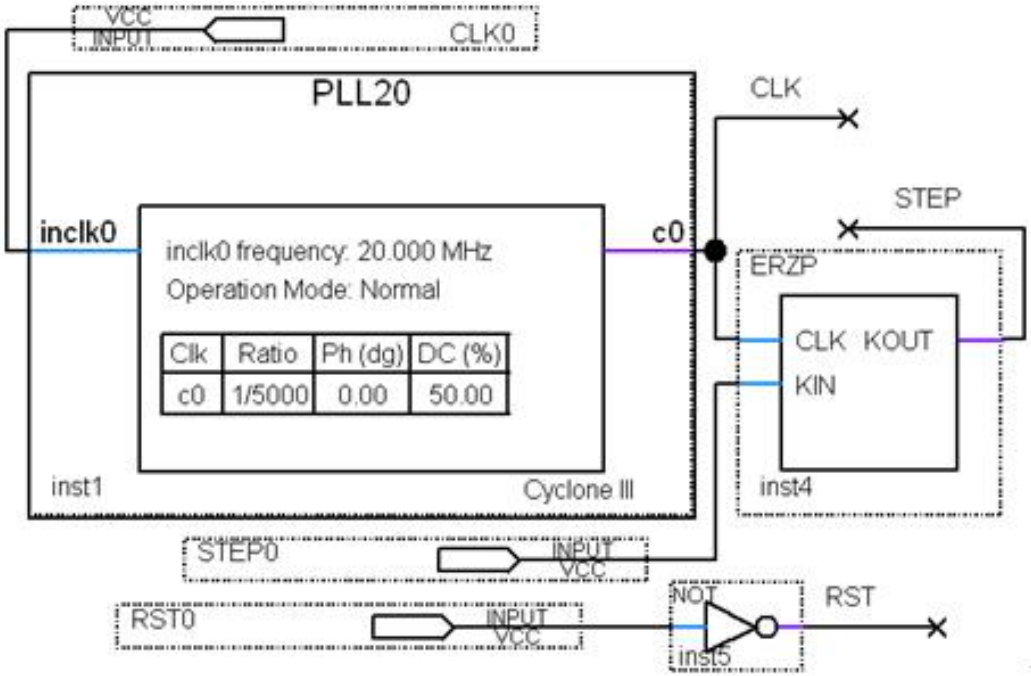

图 9-16 加入锁相环的电路方案

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

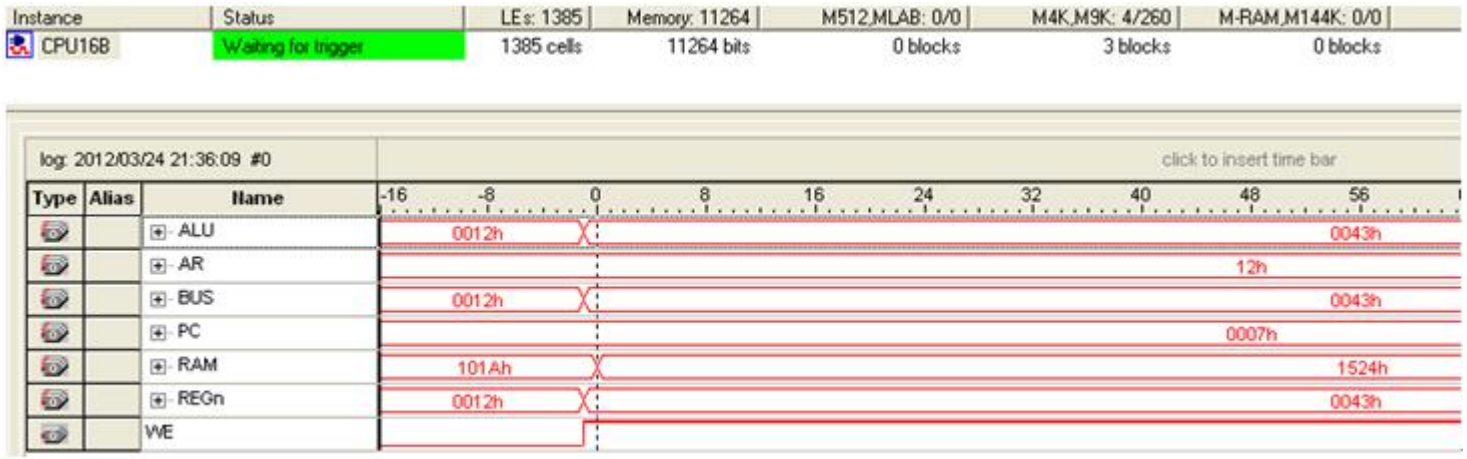

图 9-17 SignalTap II 对 KX9016 执行从 RAM 写数据指令的实时测试波形

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	00	00
000010	00	00	00	00	00	43	00	00	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 9-18　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

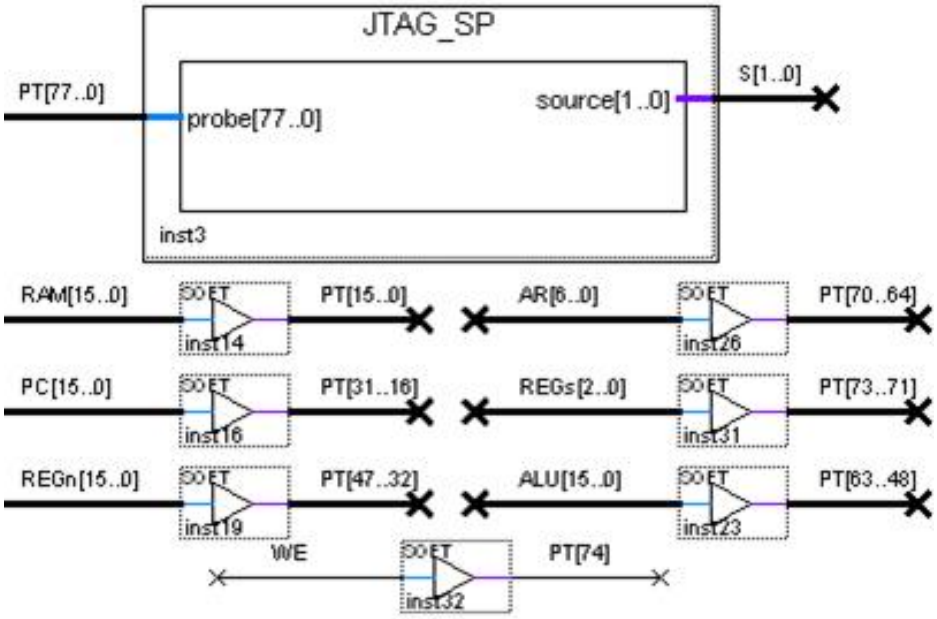

图 9-19　S/P 模块对 KX9016 端口的连接情况

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 9-20　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应用程序设计示例和系统优化

9.5.1 乘法算法及其硬件实现

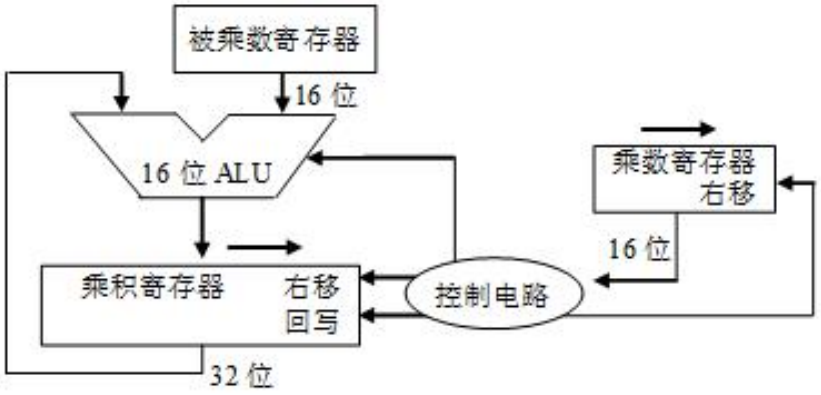

图 9-21 乘法算法 1 的硬件实现

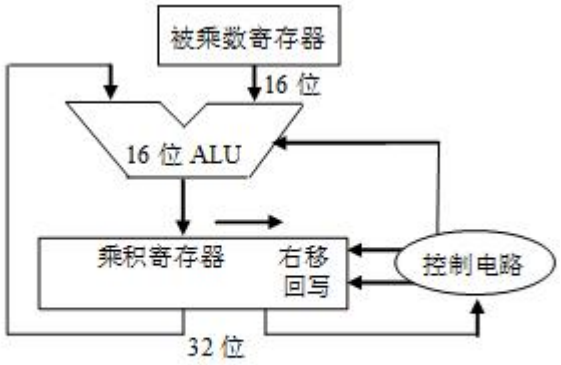

图 9-22 改进后的乘法算法 2 的硬件实现

9.5 KX9016应用程序设计示例和系统优化

9.5.1 乘法算法及其硬件实现

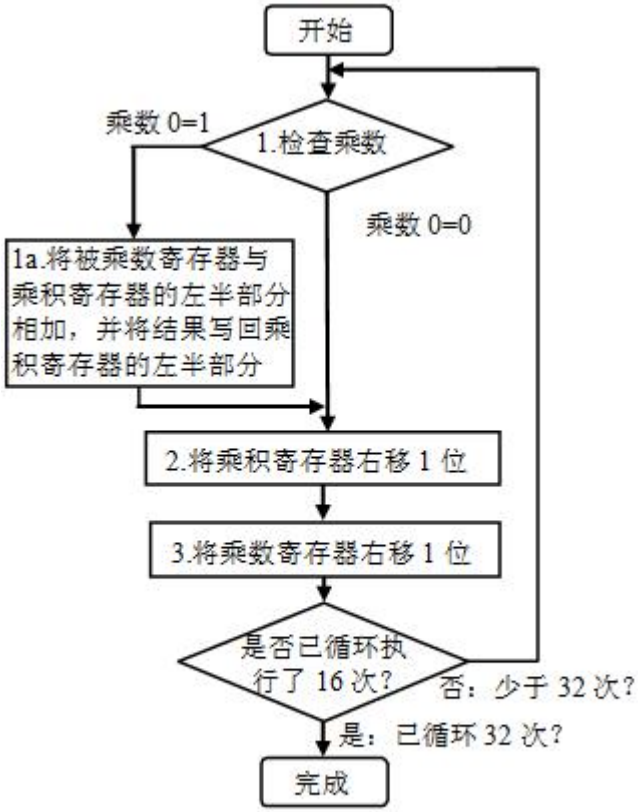

图 9-23 乘法算法 1 的流程图

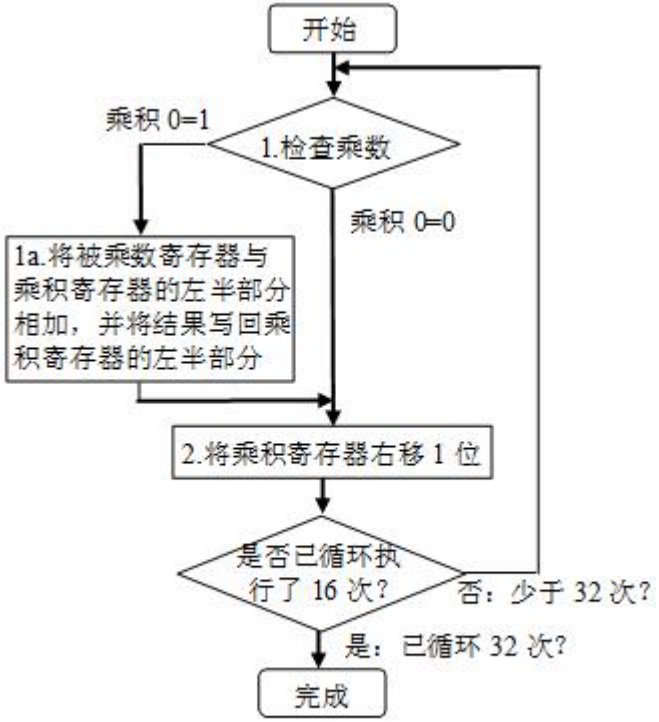

图 9-24 乘法算法 2 的流程图

9.5 KX9016应用程序设计示例和系统优化

9.5.2 除法算法及其硬件实现

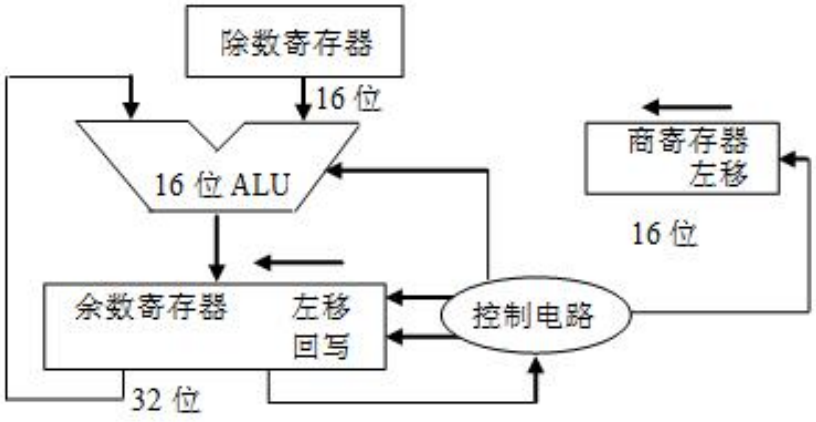

图 9-25 除法算法 1 的硬件结构

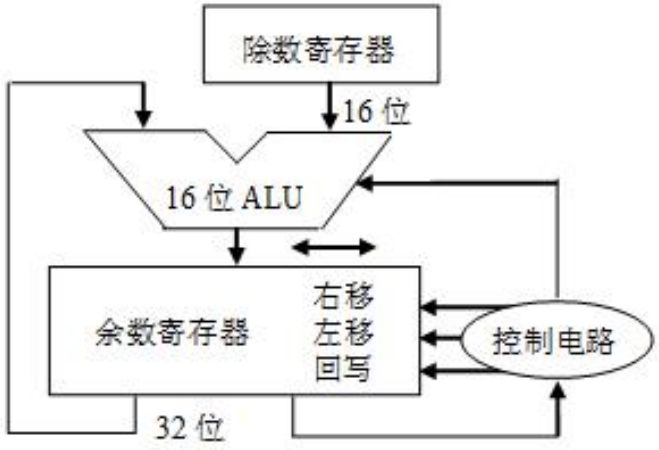

图 9-26 除法算法 2 的硬件结构

9.5 KX9016应用程序设计示例和系统优化

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 9-27 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

0 RAM:

000000	20 01	00 21	20 02	00 58	20 06	00 40	08 0B	10 1A	30 0E	00 00	38 01	..! ..X .
00000B	38 02	28 00	00 06	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	8.(.....
000016	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000021	FF 00	43 21	54 32	65 43	44 33	55 44	2D FF	12 34	23 45	34 56	20 30	..C!T2eCD3
00002C	30 33	30 68	2D 2D	34 66	A1 A2	B2 B3	C3 C4	D5 D6	E6 E7	F8 F9	AB CD	030h--4f..
000037	EF 01	12 12	23 23	34 34	56 56	78 78	89 89	AB AB	CD CD	EF EF	00 00	....##44VV
000042	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
00004D	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000058	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000063	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
00006E	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000079	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....

图 9-28 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