**武汉大学计算机学院**

**本科生实验报告**

**计算机组成原理实验**

**运算器实验**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：计算机组成原理

指 导 教 师 ：计组组实验老师

学 生 学 号 ：2019302070035

学 生 姓 名 ：刘涛榕

二○二○年11月

**1. 实验内容**

一、实验目的

1. 进一步掌握运算器的组成及工作原理；

2. 熟悉运算器的数据传输通路；

3. 熟悉运算器执行算术运算与逻辑运算的具体实现过程；

4. 掌握双端口寄存器的运算器进行算术与逻辑运算的控制方法；

5. 按给定数据，完成几种指定的算术和逻辑运算，验证双端口寄存器的运算器的功能。

二、 实验原理

算术逻辑单元（ALU）的主要功能是对二进制数据进行定点算术运算、逻辑运算和各种移位操作。算术运算包括定点加减乘除运算；逻辑运算主要有逻辑与、逻辑或、逻辑异或和逻辑非操作。ALU通常有两个数据输入端A和B，一个数据输出端Y以及标志位等。

AM2901是一种典型的4位ALU器件。图1-1是AM2901芯片的内部逻辑电路图。表1-1是AM2901的9个控制信号（I8~I0）的作用，这9个控制信号决定了AM2901进行相应运算所需要的数据来源、运算功能和数据去向。

二选一

运算部件ALU

二选一

三选一

A锁存器

B锁存器

16个通用寄存器

三选一

Q寄存器

三选一

A地址

B地址

输入D

CIN

S

V

Z

输出**Y**

**R**

**S**

**F**

A

B

RAM0

RAM3

Q3

Q0

/OE 接地

**I3**

**I4**

**I5**

**I0**

**I1**

**I2**

**I6**

**I7**

**I8**

4

4

4

4

运算功能

数据来源

数据去向

C

**B**

F3

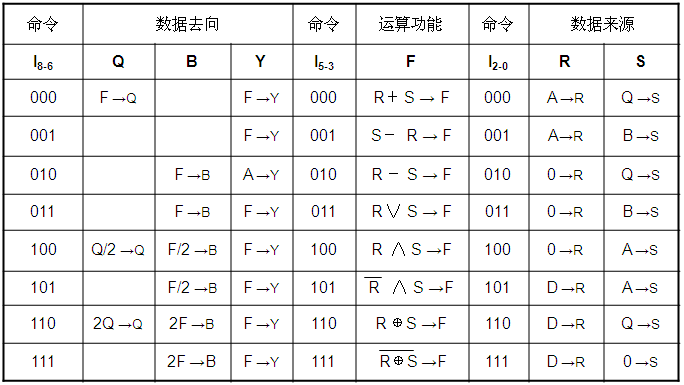
OVR

F0

Cy

图1-1 AM2901芯片的内部逻辑结构

表1-1 AM2901的数据去向、运算功能和数据来源



20世纪80年代计算机中的算术逻辑单元有许多是用现成的算术逻辑器件连接起来构成的，如第4章提到的74181和这一章提到的Am2901等就是著名的算术逻辑器件。使用这些4位的算术逻辑器件，能够构成8位、16位等长度的算术逻辑单元。现在由于超大规模器件的广泛应用，使用这种方法构成算术逻辑单元已经不多见，代之以直接用硬件描述语言设计算术逻辑单元。本章实验就采用VHDL语言来描述一个仿AM2901算术逻辑单元的实现过程，并简化了其控制信号和操作流程，调整了其运算功能。

**2**．**实验步骤与分析**

1. 实验设计目标

设计一个16位算术逻辑单元，满足以下要求。

（1） 16位算术逻辑单元能够进行下列运算：加法、减法、与、或、异或、左移、右移。用3位运算操作码alu\_fun[2…0]控制ALU进行相应的运算，控制方式如表1-2所示。

（2） 设立四个标志寄存器CZVS。这四个标志寄存器会根据ALU部件执行的运算结果进行改变。当运算结束后，运算结果改变标志寄存器CZVS的情况如表1-2所示。其中，对应列下面打“\*”的表示此种运算对这个标志位有影响，否则不影响此标志位。

（3） 为了为ALU提供源操作数A和B，设计了一个4×16位双端口寄存器组。DR和SR两个端口可同时读，这样ALU部件需要的源操作数都可以从通用寄存器文件中取出来，加快了指令的执行速度。但只有DR端口即可以读也可以写，SR端口只能读不能写。

当寄存器写使能信号DRWr=1时，则在时钟信号clk的上升沿将数据输入d\_input[15..0]写入到DR[1..0]端口所指定的寄存器中。当寄存器写使能信号DRWr=0时，则在时钟信号clk的上升沿将DR[1..0]端口所指定的寄存器内容送入ALU部件的alu\_a[15..0]，同时将SR[1..0]端口所指定的寄存器内容送入ALU部件的alu\_b[15..0]，供ALU完成相应的运算。

（4） 算术逻辑单元用一个设计实体完成。

表1-2 算术逻辑单元的运算

|  |  |  |
| --- | --- | --- |
| ALU功能控制信号 | 运算功能 | 标志位 |
| alu\_fun[2…0] | C Z V S |
| 000 | Result ← A+B+CIN | \* \* \* \* |
| 001 | Result ← B-A-CIN | \* \* \* \* |
| 010 | Result ← A and B | \* \* |
| 011 | Result ← A or B | \* \* |
| 100 | Result ← A xor B | \* \* |
| 101 | Result ← B左移一位，低位补0，最高位进C标志位 | \* \* \* |
| 110 | Result ←B右移一位，高位补0，最低位进C标志位 | \* \* \* |
| 其他 | 全零 | 全零 |

2. 顶层设计实体的引脚要求

（1）sw\_clk对应实验台上的时钟（手动时钟）；

（2）sw\_rst对应实验台上的CPU复位信号（复位按钮）；

（3）DRWr写使能信号对应实验台上的开关I5；

（4）运算功能控制信号alu\_fun[2..0]对应实验台上的开关I8~I6；

（5）进位信号cin对应实验台上的开关I4；

（6）提供运算操作数的ROM地址rom\_aa[3..0]对应实验台上的开关A3~A0；

（7）寄存器文件DR[1..0]端口地址对应实验台上的开关B3B2；

（8）寄存器文件SR[1..0]端口地址对应实验台上的开关B1B0；

（9） ROM对通用寄存器文件提供的源操作数对应实验台上的指示灯B3~B0和A3~A0以及I8~I1，共计16个LED显示灯；

（10）算术逻辑部件alu\_a[15..0]端口的源操作数（同时也是通用寄存器文件DR[1..0]端口读出的数据），通过实验台上的指示灯Y7~Y0和D7~D0进行显示；

（11）算术逻辑部件alu\_b[15..0]端口的源操作数（同时也是通用寄存器文件SR[1..0]端口读出的数据），通过实验台上的指示灯IR15~IR0进行显示；

（12）16位的运算结果alu\_out[15..0]对应实验台上的指示灯D15~D0；

（13）标志位CZVS的结果对应实验台上的指示灯SCC3~SCC0。

3．实验代码

1. 算术逻辑部件alu.vhd参考源代码

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity alu is

port(cin:in std\_logic;

alu\_a,alu\_b:in std\_logic\_vector(15 downto 0);

alu\_func:in std\_logic\_vector(2 downto 0);

alu\_out:out std\_logic\_vector(15 downto 0);

c,z,v,s:out std\_logic);

end alu;

architecture behave of alu is

begin

process(alu\_a,alu\_b,cin,alu\_func)

variable temp1,temp2,temp3 : std\_logic\_vector(15 downto 0) ;

begin

temp1 := "000000000000000"&cin;

case alu\_func is

when "000"=>

temp2 := alu\_b+alu\_a+temp1;

when "001"=>

temp2 := alu\_b-alu\_a-temp1;

when "010"=>

temp2 := alu\_a and alu\_b;

when "011"=>

temp2 := alu\_a or alu\_b;

when "100"=>

temp2 := alu\_a xor alu\_b;

when "101"=>

temp2(0) := '0';

for I in 15 downto 1 loop

temp2(I) := alu\_b(I-1);

end loop;

when "110"=>

temp2(15) := '0';

for I in 14 downto 0 loop

temp2(I) := alu\_b(I+1);

end loop;

when others=>

temp2 := "0000000000000000";

end case;

alu\_out <= temp2;

if temp2 = "0000000000000000" then z<='1';

else z<='0';

end if;

if temp2(15) = '1' then s<='1';

else s<='0';

end if;

case alu\_func is

when "000" | "001"=>

if (alu\_a(15)= '1' and alu\_b(15)= '1' and temp2(15) = '0') or

(alu\_a(15)= '0' and alu\_b(15)= '0' and temp2(15) = '1') then

v<='1';

else v<='0';

end if;

when others=>

v<='0';

end case;

case alu\_func is

when "000"=>

temp3 := "1111111111111111"-alu\_b-temp1;

if temp3<alu\_a then

c<='1';

else c<='0';

end if;

when "001"=>

if alu\_b<alu\_a then

c<='1';

else c<='0';

end if;

when "101"=>

c <= alu\_b(15);

when "110"=>

c <= alu\_b(0);

when others=>

c<='0';

end case;

end process;

end behave;

2. 通用寄存器文件regfile.vhd参考源代码

通用寄存器文件采用了层次结构的设计方法，设计方案如下：

（1） 低层设计实体reg：完成寄存器复位和读写功能；

（2） 低层设计实体mux\_4\_to\_1:完成选择哪一个寄存器的值送寄存器组的输出，这是一个4选1选择器；

（3） 低层设计实体decoder\_2\_to\_4:完成选择写哪一个寄存器。这是一个2-4译码器；

（4） 高层设计实体regfile：负责3个底层设计实体的连接，完成寄存器文件的全部功能。

低层设计实体reg参考源代码如下：

library ieee;

use ieee.std\_logic\_1164.all;

entity reg is

port

(

clr: in std\_logic;

D: in std\_logic\_vector(15 downto 0);

clock: in std\_logic;

write: in std\_logic;

sel: in std\_logic;

Q: out std\_logic\_vector(15 downto 0)

);

end reg;

architecture behav of reg is

begin

process(clr,clock)

begin

if clr = '0' then

Q <= x"0000";

elsif (clock'event and clock = '1') then

if sel = '1' and write = '1' then

Q <= D;

end if;

end if;

end process;

end behav;

底层设计实体mux\_4\_to\_1参考源代码如下：

library ieee;

use ieee.std\_logic\_1164.all;

entity mux\_4\_to\_1 is

port (

input0,

input1,

input2,

input3: in std\_logic\_vector(15 downto 0);

sel: in std\_logic\_vector(1 downto 0);

out\_put: out std\_logic\_vector(15 downto 0));

end mux\_4\_to\_1;

architecture behav of mux\_4\_to\_1 is

begin

mux: process(sel, input0, input1, input2, input3)

begin

case sel is

when "00" =>

out\_put <= input0;

when "01" =>

out\_put <= input1;

when "10" =>

out\_put <= input2;

when "11" =>

out\_put <= input3;

end case;

end process;

end behav;

低层设计实体decoder\_2\_to\_4参考源代码如下：

Library ieee;

use ieee.std\_logic\_1164.all;

entity decoder\_2\_to\_4 is

port (

sel: in std\_logic\_vector(1 downto 0);

sel00: out std\_logic;

sel01: out std\_logic;

sel02: out std\_logic;

sel03: out std\_logic

);

end decoder\_2\_to\_4;

architecture behav of decoder\_2\_to\_4 is

begin

sel00 <= (not sel(1)) and (not sel(0));

sel01 <= (not sel(1)) and sel(0) ;

sel02 <= sel(1) and (not sel(0)) ;

sel03 <= sel(1) and sel(0) ;

end behav;

高层设计实体regfile参考源代码如下：

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity regfile is

Port ( DR: in std\_logic\_vector(1 downto 0);

SR: in std\_logic\_vector(1 downto 0);

reset: in std\_logic;

DRWr: in std\_logic;

clk: in std\_logic;

d\_input: in std\_logic\_vector(15 downto 0);

DR\_data: out std\_logic\_vector(15 downto 0);

SR\_data: out std\_logic\_vector(15 downto 0)

);

end regfile;

architecture struct of regfile is

-- components

-- 16 bit Register for register file

component reg

port (

clr: in std\_logic;

D: in std\_logic\_vector(15 downto 0);

clock: in std\_logic;

write: in std\_logic;

sel: in std\_logic;

Q: out std\_logic\_vector(15 downto 0)

);

end component;

-- 2 to 4 Decoder

component decoder\_2\_to\_4

port(

sel: in std\_logic\_vector(1 downto 0);

sel00: out std\_logic;

sel01: out std\_logic;

sel02: out std\_logic;

sel03: out std\_logic

);

end component;

-- 4 to 1 line multiplexer

component mux\_4\_to\_1

port (

input0,

input1,

input2,

input3: in std\_logic\_vector(15 downto 0);

sel: in std\_logic\_vector(1 downto 0);

out\_put: out std\_logic\_vector(15 downto 0));

end component;

signal reg00, reg01, reg02, reg03

:std\_logic\_vector(15 downto 0);

signal sel00 ,sel01 ,sel02 ,sel03

: std\_logic;

begin

Areg00: reg port map(

clr => reset,

D => d\_input ,

clock => clk ,

write => DRWr ,

sel => sel00 ,

Q => reg00

);

Areg01: reg port map(

clr => reset,

D => d\_input ,

clock => clk ,

write => DRWr ,

sel => sel01 ,

Q => reg01

);

Areg02: reg port map(

clr => reset,

D => d\_input ,

clock => clk ,

write => DRWr ,

sel => sel02 ,

Q => reg02

);

Areg03: reg port map(

clr => reset,

D => d\_input ,

clock => clk ,

write => DRWr ,

sel => sel03 ,

Q => reg03

);

-- decoder

des\_decoder: decoder\_2\_to\_4 port map

(

sel => DR,

sel00 => sel00 ,

sel01 => sel01 ,

sel02 => sel02 ,

sel03 => sel03

);

mux1: mux\_4\_to\_1 PORT MAP(

Input0 => reg00 ,

Input1 => reg01 ,

Input2 => reg02 ,

Input3 => reg03 ,

sel => DR ,

out\_put => DR\_data

);

mux2: mux\_4\_to\_1 PORT MAP(

input0 => reg00 ,

input1 => reg01 ,

input2 => reg02 ,

input3 => reg03 ,

sel => SR ,

out\_put => SR\_data

);

end struct;

实验步骤：

1. 设计ALU元件

在Quartus II 环境下，用文本输入编辑器Text Editor输入alu.vhd算术逻辑单元文件，编译VHDL文件，并将alu.vhd文件制作成一个可调用的原理图元件。

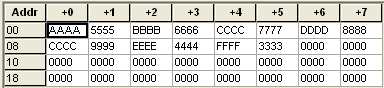
2. 设计寄存器

在Quartus II 环境下，用文本输入编辑器Text Editor输入高层设计实体regfile.vhd，以及regfile的三个底层调用文件reg.vhd、decoder\_2\_to\_4.vhd和mux\_4\_to\_1.vhd，编译VHDL文件，并将regfile.vhd文件制作成一个可调用的通用寄存器组原理图元件。

3. 建立一个ROM及其初始化文件

在Quartus II 环境下，建立一个16字×16位的ROM,并对其进行初始化，初始化文件名为alu\_2.mif（必须和顶层设计实体同名），建立好后改成以十六进制显示其地址和数据，要填充的数据如表1-3所示，此数据为ALU进行相应运算所需要的操作数。

表1-3 ROM初始化文件内容



4. 以原理图方式建立顶层文件工程

选择图形方式。根据图1-2输入实验电路图，从Quartus II的基本元件库中将各元件调入图形编辑窗口、连线，添加输入输出引脚。

将所设计的图形文件alu\_2.bdf保存到原先建立的文件夹中，将当前文件设置成工程文件，以后的操作就都是对当前工程文件进行的。

5. 器件选择

选择Cyclone系列，在Devices中选择器件EP2C20Q240。编译，引脚锁定，再编译。引脚锁定后需要再次进行编译，才能将锁定信息确定下来，同时生成芯片编程/配置所需要的各种文件。

顶层设计实体的引脚要求如图1-3所示。

6. 芯片编程Programming

打开编程窗口。将配置文件alu\_2.sof下载到实验台系统中的FPGA中。

7. 实验验证

根据引脚绑定要求，设置相应的控制信号，验证设计的双端口寄存器文件和ALU的算术运算和逻辑运算功能。

实验任务：

1. 做好实验预习。

2. 按照理论分析值填写表1-4、表1-5、表1-6、表1-7

3. 按照以下操作说明进行实验：

**任务一、将ROM存储器0000单元的内容打入R0，0001单元的内容打入R1，并进行相关功能的运算。**

（1）复位；

（2）置开关“I5”=1 (往寄存器组里写数,使DRWr信号为1)；

（3）开关“A3..A0”=0000, “B3B2”=00(ROM单元通过DR端口给R0送数)；

（4）按下“手动时钟”给脉冲(如果一下不行再按)，此时看到LED显示灯B3~B0 A3~A0 I8~I1为”1010101010101010” (ROM读出)；

（5）按下“手动时钟”给脉冲，此时看到LED显示灯Y7~Y0D7~D0为”1010101010101010” (打入R0的值)；

（6）开关“A3..A0”=0001, “B3B2”=01(ROM单元通过DR端口给R1置数)；

（7）按下“手动时钟”给脉冲， 此时看到LED显示灯B3~B0A3~A0I8~I1为”0101010101010101” (ROM读出)；

（8）按下“手动时钟”给脉冲，此时看到LED显示灯Y7~Y0D7~D0为”0101010101010101” (打入R1的值)；

经过（1）~（8）步，此时R0=AAAA，R1=5555，将开关B3B2=00（即DR选R0=AAAA），B1B0=01（即SR选R1=5555 ），并选择其他相关控制信号，观察运算结果D15~D0和SCC3~SCC0(CZVS)，记录实验结果到表1-4和表1-5中。

**任务二、同任务一的操作方法，将ROM存储器0002单元的内容打入R3， 0003单元的内容打入R2，并进行相关功能的运算。**

任务二的实验结果记录到表1-6和表1-7中。



图1-2 顶层设计实体连接图

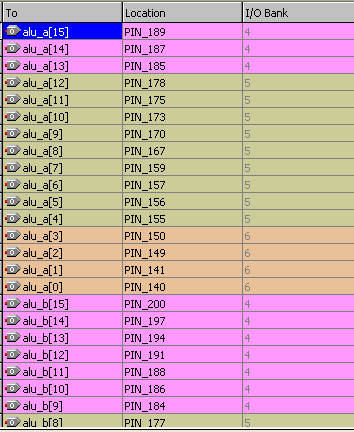


图1-3（1） 顶层设计实体的引脚要求

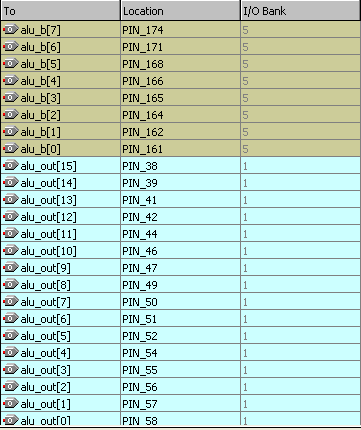


图1-3（2） 顶层设计实体的引脚要求

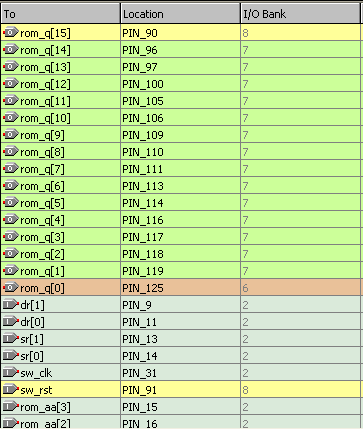


图1-3（3） 顶层设计实体的引脚要求

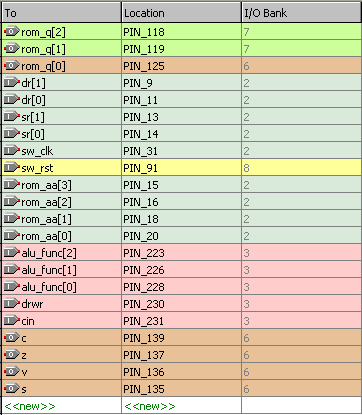


图1-3（4） 顶层设计实体的引脚要求

**3运算器实验结果与总结**

实验结果 ：

在Quartus II软件中设计好相应的底层文件和顶层文件之后，将其下载到下载到实验箱中，在实验箱中进行上述步骤中的操作，拨动相应的开关，观察LED显示灯的显示结果。正确操作之后，实验箱中的LED对于运算器的运算结果显示正确，分别实现了对操作数相加、相减、相与、或非、左移等运算。记录实验现象，填入下表中。

表5-4 ALU\_2运算测试数据（CIN=0）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 运算功能 | ALU\_FUN | DR(1..0) | SR(1..0) | ALU输出 | C | Z | V | S |
| B+A+CIN | 000 | 00 | 01 | FFFF | 0 | 0 | 0 | 1 |
| B-A-CIN | 001 | 00 | 01 | AAAB | 1 | 0 | 0 | 1 |
| A and B | 010 | 00 | 01 | 0000 | 0 | 1 | 0 | 0 |
| A or B | 011 | 00 | 01 | FFFF | 0 | 0 | 0 | 1 |
| A xor B | 100 | 00 | 01 | FFFF | 0 | 0 | 0 | 1 |

表5-5 ALU\_2运算测试数据（CIN=1）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 运算功能 | ALU\_FUN | DR(1..0) | SR(1..0) | ALU输出 | C | Z | V | S |
| B+A+CIN | 000 | 11 | 10 | 0000 | 1 | 1 | 0 | 0 |
| B-A-CIN | 001 | 11 | 10 | AAAA | 1 | 0 | 0 | 1 |
| A and B | 010 | 11 | 10 | 0000 | 0 | 1 | 0 | 0 |
| A or B | 011 | 11 | 10 | FFFF | 0 | 0 | 0 | 1 |
| A xor B | 100 | 11 | 10 | FFFF | 0 | 0 | 0 | 1 |

表5-6 ALU\_2运算测试数据（CIN=0）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 运算功能 | ALU\_FUN | DR(1..0) | SR(1..0) | ALU输出 | C | Z | V | S |
| B+A+CIN | 000 | 11 | 10 | 2221 | 1 | 0 | 0 | 0 |
| B-A-CIN | 001 | 11 | 10 | AAAB | 1 | 0 | 0 | 1 |
| A and B | 010 | 11 | 10 | 2222 | 0 | 0 | 0 | 0 |
| A or B | 011 | 11 | 10 | FFFF | 0 | 0 | 0 | 1 |
| A xor B | 100 | 11 | 10 | DDDD | 0 | 0 | 0 | 1 |

表5-7 ALU\_2运算测试数据（CIN =1）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 运算功能 | ALU\_FUN | DR(1..0) | SR(1..0) | ALU输出 | C | Z | V | S |
| B+A+CIN | 000 | 11 | 10 | 2222 | 1 | 0 | 0 | 0 |
| B-A-CIN | 001 | 11 | 10 | AAAA | 1 | 0 | 0 | 1 |
| A and B | 010 | 11 | 10 | 2222 | 0 | 0 | 0 | 0 |
| A or B | 011 | 11 | 10 | FFFF | 0 | 0 | 0 | 1 |
| A xor B | 100 | 11 | 10 | DDDD | 0 | 0 | 0 | 1 |

实验体会：

本次实验相较于以往的两次实验难度加大了许多，任务量也增加了许多。在详细地听完老师的讲解后，自己操作还是会出很多错误。往往会出现不知道下一步该做什么的情况，说明对实验步骤还不是很熟悉，要多练习，争取达到熟练。

这一次的运算器实验，是在存储器ROM和通用寄存器组实验的基础上进行的，所以要能正确做出这次实验，需要先对之前两个实验进行比较熟悉的操作，并且要知道这两个实验的原理，数据的处理过程。所以对于每一个实验，都需要以认真的态度进行。一个实验可能是另外的实验的基础，而且相近的实验之间的原理是相通的，当真正理解了一个实验的原理之后，等到再遇到相近似的实验，可以更容易理解并且进行相应的操作。相反，如果在每个实验中都只是不求甚解地做实验，最终做完还是不明白实验的原理，只能简单看到一个实验的结果，但是不知道是怎么得出来的。

做实验的过程中，对实验现象有不明白的地方，要及时对实验过程进行分析，找出这个实验结果对应的实验操作，弄清楚其相应的原理，最终才可以是做实验达到最佳的效果。在对实验原理有更深入一步的了解的同时，还可以增强自己的实际动手能力。