AI2618 电子电路系统实验 课程综合作业
简易元件参数及三极管输出特性测试仪

薛翔元 卜家梓 李奇睿

目录

1 实验内容

1.1 实验任务

设计元件参数测量电路，通过面包板搭建电路，并以 Arduino Nano 单片机为控制芯片，利用 C++ 编程实现简易元件参数及三极管输出特性测试仪。实现的元件测试仪应当支持常见三引脚以内的元件特性检测，包括电阻、电容、三极管。此外，元件测试仪应当自动识别待测元件的类型，并进行相应测量。

- 对于电阻，测量电阻大小 R（基本任务）
- 对于电容，测量电容大小 C（基本任务）
- 对于三极管，确定其类型和管脚（基本任务）
- 对于三极管，测量其电流放大倍数 β（附加任务）

1.2 实验材料

搭建电路的材料及工具包括：

- 单片机一块及数据线一条
- 面包板一块及硬导线 1m
- 680Ω、510kΩ 电阻 6 个
- 饭盒与剥线钳各一个

1.3 待测元件

待测元件如下：

- 电阻 360Ω、5.6kΩ、100kΩ
- 电容 47pF、0.1μF（104）、10μF
- PNP 三极管（9012）、NPN 三极管（9013）

2 实现思路

2.1 元件识别

2.1.1 三极管的识别

三极管是待测元件中唯一的三端元件，具有其他元件所不具备的特性，因此最容易进行区分。我们首先对三极管的识别方案进行设计：利用三端特性，在三个测量端口中选择一个输入高电压 V_{cc}，其余两个端口输入低电压 0V，依次遍历三种接入情况。得益于三极管中 PN 结具有良好的单向导电性，在某一种接入情况下，将会出现三个测量端口电压读数均不为零的情况。实验表明，三个电压均在 1.5V 附近，而其他二端元件均不具有此特性，由此我们可以通过这种测试方法识别出三极管元件。

2.1.2 电阻的识别

电阻是一个无向的二端元件，因此我们可以从接入的两个接口中任选其一，一端接入高电压 V_{cc}，并且选用 680Ω 的电阻（等效为内阻 680Ω 的电压源），另一端接入低电压 0V，并且选用 680Ω 的电阻。在一定时间的延迟后（这是为了避免与未充满的电容发生混淆），测量目标测试端口的电位差 U_x，经过理论计算和实验测试，对于 5V 的总电势差，电阻的高电压测电位不应过大（≤ 4.95V），低点压测电位不应过小（≥ 0.01V）。按此方法，电阻的识别效果比较理想。

2.1.3 电容的识别

在排除三极管和电阻的情形后，由于电容的特殊时域性质，我们可以利用电位差建立后，元件两侧的电压变化情况进行判断。因此，我们在充分放电以后，对目标端口进行充电，并测量达到稳态所需的时间，若该时间超过一定阈值，则目标端口间一定为电容元件。由于此过程与电容大小的测量完全相同，我们复用测量函数，直接测量目标端口之间的电容大小，若测得电容超过阈值，则可以确定目标端口间为电容元件。经过实验测试，我们将该阈值设定为 $10^{-5}\mu$F。

2.1.4 空载状态的识别

空载状态下，目标端口之间开路，既可以看作一个无穷大的电阻，也可以看作无穷小的电容。但由于我们此前已经可以分辨电阻、电容、三极管，因此其余状态均可视为空载，不产生任何输出。

2.2 电阻的测量

2.2.1 测量原理

电阻测量的电路如下图所示。我们利用一个已知大小的电阻和待测电阻串联，在已知电阻侧施加高电压 V_{cc}，在待测电阻侧施加低电压 0V，形成一个简易的分压电路。测量待测电阻的分压，根据串联分压公式即可计算出待测电阻的阻值。

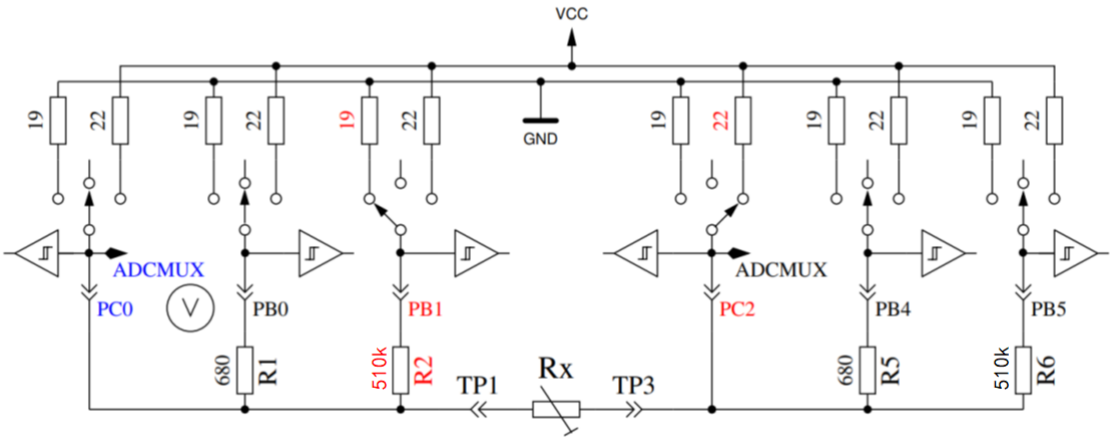


图 1: 电阻测量原理

为了提高电阻测量的精确度，我们设置两个量程，按照测量情况选择合适的量程。我们要确定一个合适的阈值，作为两个量程的分界点。当两个量程具有相同的误差时，电路满足相同的分压关系，已知电阻与待测电阻的比值互为倒数，也即：

$$\frac{R_x}{R_H} = \frac{R_L}{R_x}$$

因此选用 $R_0 = \sqrt{R_H R_L}$ 比较合理，这一结果也在实验中得到了验证。

2.2.2 大电阻的测量

我们首先假定待测电阻阻值很大，串联 510kΩ 大电阻，测量待测电阻两端的电压 U_x，根据分压公式有:

$$\frac{V_{cc} - U_x}{R_H} = \frac{U_x - 0}{R_x}$$

其中 R_x 为待测电阻的阻值，$R_H = 510\text{k}\Omega$，可知:

$$R_x = \frac{U_x}{V_{cc} - U_x} R_H$$

若测得 $R_x \geq R_0$，我们认为 R_x 落在大量程之内，接受这一测量结果。

2.2.3 小电阻的测量

如果测得阻值 R_x 过小，表明 $R_H >> R_x$，此时测量结果不够精确，我们改用 680Ω 小电阻，在小量程下重新对 R_x 进行测量，可以得到更为精确的结果，类似可得公式:

$$\frac{V_{cc} - U_x}{R_L} = \frac{U_x - 0}{R_x}$$

因此:

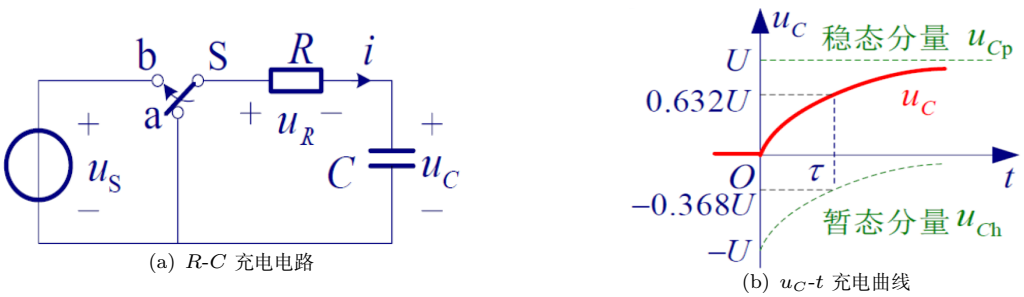
$$R_x = \frac{U_x}{V_{cc} - U_x} R_L$$

由于大量程已经被排除，R_x 必然落在小量程之内，我们直接接受这一测量结果。

2.3 电容的测量

2.3.1 测量原理

针对电容的测量，参考文档中使用脉冲输入，再对端口电压的波形进行分析，这种方法虽然精度较高，但是需要实现波形发生器和运算电路，我们缺少关键的电路元件，因此这种方案并不可行。作为替代，我们利用时域下电容的充电特性，通过测量时间常数的方法间接测量电容。为保证测量的准确性，我们首先需要对电容进行充分的放电，然后在一侧输入高电压 V_{cc}，另一侧输入低电压 0V。同时，我们将一个电阻与电容进行串联，构成 R-C 充电电路。



(a) R-C 充电电路　　(b) u_C-t 充电曲线

图 2: 电容测量原理

根据电容的特性方程和基尔霍夫定律，列出如下方程组：

$$\begin{cases} i = C\dfrac{\mathrm{d}u_C}{\mathrm{d}t} \\ u_S = iR + u_C \end{cases}$$

在充分放电的前提下有初始条件 $u_C(0) = 0$，求解微分方程可知：

$$u_C = u_S\left(1 - e^{-\frac{t}{\tau}}\right), \quad t \geq 0$$

其中 $\tau = RC$ 为时间常数。因此在建立电位差以后，我们反复测量电容两端的电压，根据 R-C 充电电路的特性曲线可知，经过一个时间常数 τ 以后，电容两端的电压 $u_C = V_{cc}(1 - e^{-1})$。近似地，当电容两端电压达到 $0.632V_{cc}$ 时，经过的时间恰好为 τ，根据公式 $\tau = RC$ 可以计算电容大小。

2.3.2 大电容的测量

对于大电容的测量，我们不希望充电过程耗费过多的时间，因此选用 680Ω 小电阻构成 R-C 充电电路，由于电容较大，我们认为测量仍然是精确的。在程序实现中，使用 `micros` 函数获取微秒级起始时间戳，通过循环不断地读取端口电压，并判断电容两端的电压是否达到 $0.632V_{cc}$，循环结束时再次获取终止时间戳，其与起始时间戳的差值即为消耗的微秒数，通过公式可以计算得到电容大小 C_x。

如果测得电容大小 $C_x \geq 1\mu F$，我们可以认为待测电容为大电容，从而接受测量结果。否则我们认为电容过小，该方法在此情况下并不精确。

2.3.3 小电容的测量

在大电容测量的结果被拒绝后，我们确定待测电容为小电容。为了提高精度，我们应该使充电时间尽量长，因此选用 510kΩ 大电阻构成 R-C 充电电路，采用类似的方法测量充电时间常数，并计算得到电容大小 C_x。在此情况下，我们必然接受测量结果。由于该函数还用于电容的判定，为了避免短接等情况造成死循环，我们为循环设置次数上限，若超过次数上限则立即结束测量。

此外，我们必须指出该方案测量微小电容（47pF）的困难性。虽然 Arduino Nano 的时钟精度高达 4μs，但用于读入模拟量的 `analogRead` 函数性能较差，在默认采样率下其延迟高达 100μs，而微小电容的时间常数则远小于该延迟，因此无法准确进行测量。为了解决这一问题，我们在 Arduino 网站和技术论坛上查阅相关资料，了解到可以通过修改采样率来提高 `analogRead` 函数的测量速度。于是我们设计了 `setPrescaler` 函数，通过修改 `ADCSRA` 比特位的方式将采样率修改为 128，使得测量延迟降低至约 20ms，从而将微小电容的测量误差控制在可以接受的范围内。值得主义的是，修改模拟端口的采样率会降低测量的准确性，因此在电容测量完成后，我们需要将采样率改回 16，保证后续测量的准确性。

2.4 三极管的测量

2.4.1 三极管类型的判断

通过对 PNP 和 NPN 型三极管导通条件的分析，我们发现可以通过如下方法确定待测三极管的类型：

1. 控制三个测量端口 TP1、TP2、TP3 中有且仅有一个接入高电压 V_{cc}
2. 其余两个端口接入低电压 0V

3. 遍历三种接法，观察端口电压的输出情况

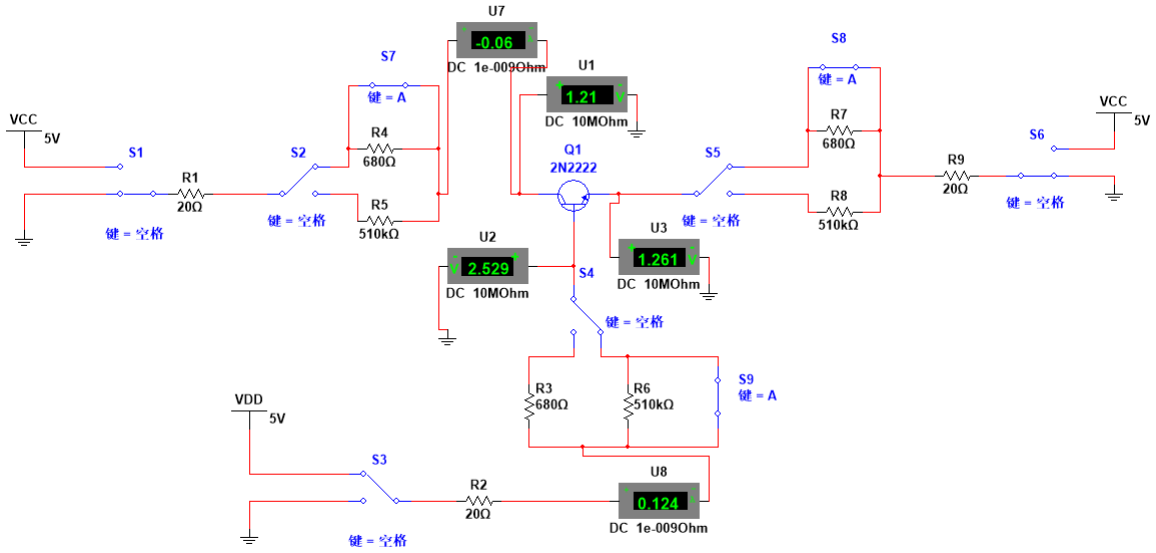


图 3: NPN 型三极管类型测试

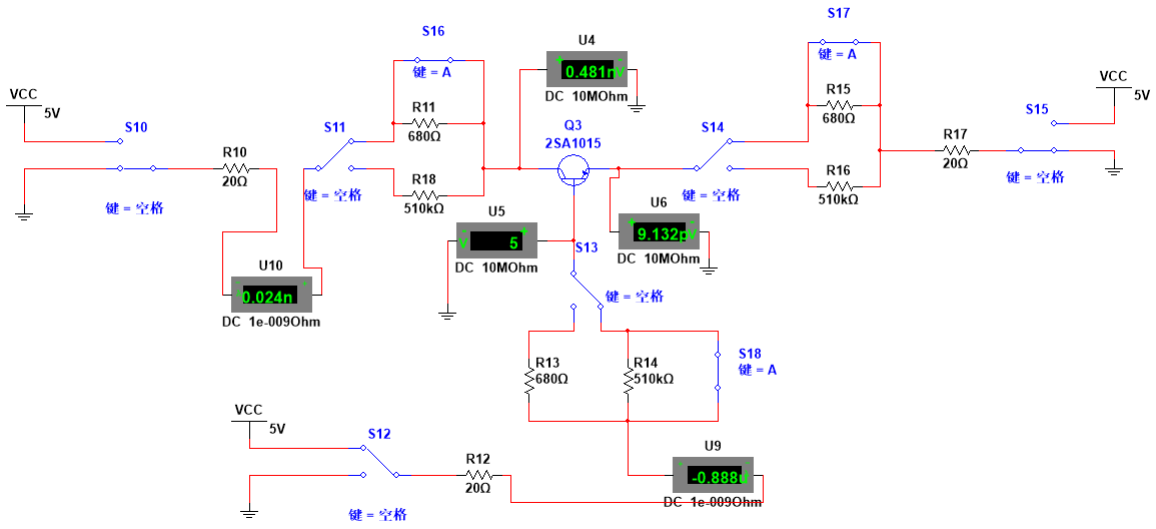


图 4: PNP 型三极管类型测试

为使实验效果更加明显，在三个端口均采用低电阻档。(实际的万用表电路为保护三极管不被击穿，通常会加上一定的限流电阻）此时，PNP 型三极管和 NPN 型三极管将会出现不同的实验现象：

- 测量三个端口的电压，如果其中两个端口的电压接近 0V，另一个端口的电压接近 V_{cc}，我们称此情形为 “奇异状态”
- PNP 型三极管：在三种接入情况中出现且仅出现一次 “奇异状态”
- NPN 型三极管：在三种接入情况中出现且仅出现两次 “奇异状态”

2.4.2 基极的确定

通过分析2.4.1中导通状态，我们可以一并得知：

- 对于 PNP 型三极管，当 “奇异状态” 出现时，正在输入高电压的测量端口对应基极
- 对于 NPN 型三极管，当 “奇异状态” 未出现时，正在输入高电压的测量端口对应基极。

于是我们可以准确判断 PNP 和 NPN 两种三极管的基极位置。

2.4.3 发射极与集电极的确定

在上述实验确定了待测三极管类型以及基极所在测量端口以后，经过大量模拟仿真和实验测试，我们选择采用两种不同的方法分别确定 PNP 和 NPN 型三极管的发射极和集电极。

- 对于 PNP 型三极管：类似地，我们仍然采用控制三个测量端口 TP1、TP2、TP3 中有且仅有一个接入高电压 V_{cc}，其余两个端口接入低电压的方法。不妨假定事先确定的基极所对应的测量端口为 TP1，此时我们将 TP1 接入高电压 V_{cc}，将 TP2 和 TP3 两个测量端口接入低电压 0V，观察除 TP2 和 TP3 测量电压的读数。实验表明，两个端口 TP2 和 TP3 的读数有显著差异，读数显著较大者对应集电极，而显著较小者对应发射极。
- 对于 NPN 型三极管：不妨假定事先确定的基极所对应的测量端口为 TP1，控制端口 TP1 始终接入高电压，将待确定的 TP2 和 TP3 其中一个端口接入高电压 V_{cc}，另一个接入低电压 0V，测量端口电压读数后，将两者电压互换，重新测量端口读数。通过分析基极的电压读数可以准确分辨：取两种情况下，TP1 端口电压读数较高者，此时，正在接入高电压的测试端口对应集电极，正在接入低电压的测试端口对应发射极。

2.4.4 放大系数的测量

根据《模拟电子技术》课程教材中基本共射放大电路的设计思路，我们可以设计出 NPN 型三极管正向电流放大倍数 β 的测量电路。经过实际测试，该方法测量结果比较合理。

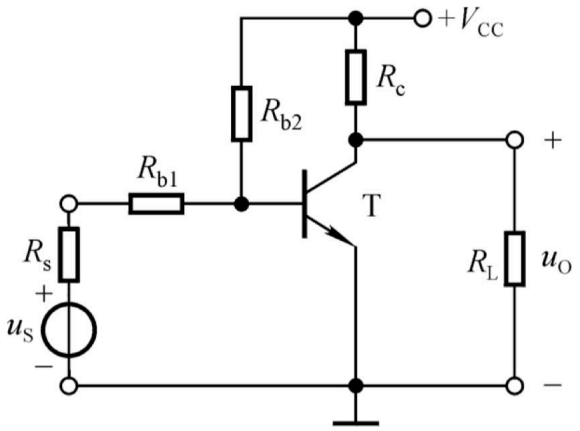


图 5: 基本共射放大电路

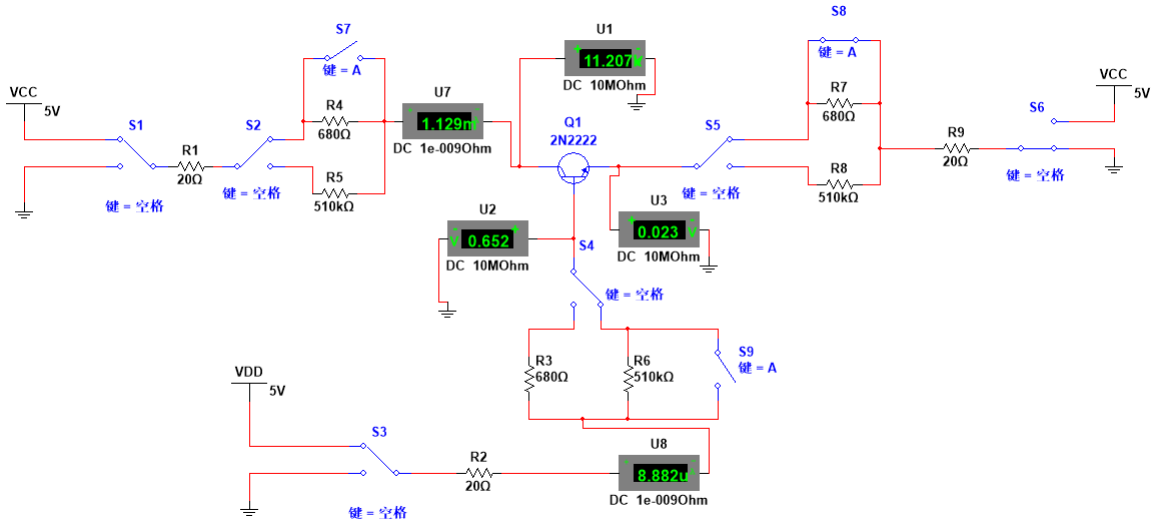


图 6: 放大倍数 β 仿真测量

对于 NPN 型三极管，仿照基本共射放大电路的设计，在基极对应的测量端口接入高电压 V_{cc}，并选取较大的 510kΩ 电阻档，以限制基极电流 I_b。在集电极对应的测量端口接入高电压 V_{cc}，并选取较小的 680Ω 电阻档，以保证集电极电流 I_c 足以体现对 I_b 的放大。在发射极对应的测量端口接入低电压 0V，并选取零电阻档。由于外电路参数已知，我们可以测量三个端口的电压，并根据欧姆定律可以计算出基极电流 I_b 和集电极电流 I_c：

$$I_b = \frac{V_{cc} - V_b}{R_H}$$
$$I_c = \frac{V_{cc} - V_c}{R_L}$$

根据正向电流放大倍数的定义式，即可计算出 β 的值：

$$\beta = \frac{I_c}{I_b}$$

而针对 PNP 型三极管，上述方法在实际测试中难以完全满足三极管工作在放大区的约束条件，因此效果欠佳。为了实现 PNP 型三极管的放大倍数 β 的测量，我们采用参考文档给出的测量方法。

对于 PNP 型三极管，在发射极对应的测量端口接入高电压 V_{cc}，并选取较大的 510kΩ 电阻档，在集电极和基极对应的测量端口接入低电压，并选取较小的 680Ω 电阻档。观察基极对应测量端口电压的读数，选用 2V 电压作为阈值，当基极电压 U_b 不低于阈值时，采用公式：

$$\beta = \frac{U_e - U_b}{U_b}$$

计算正向电流放大倍数 β。当基极电压 U_b 低于阈值时，采用公式：

$$\beta = \frac{R_H}{R_L} \cdot \frac{U_e}{U_b}$$

计算正向电流放大倍数 β。

3 电路构造

3.1 管脚构造

元件测试仪的每个测试管脚对应单片机内部三个端口，其中一个为模拟端口，可以准确测量端口电压，其余两个为数字端口，只能输出高电压 V_{cc} 或低电压 0V。

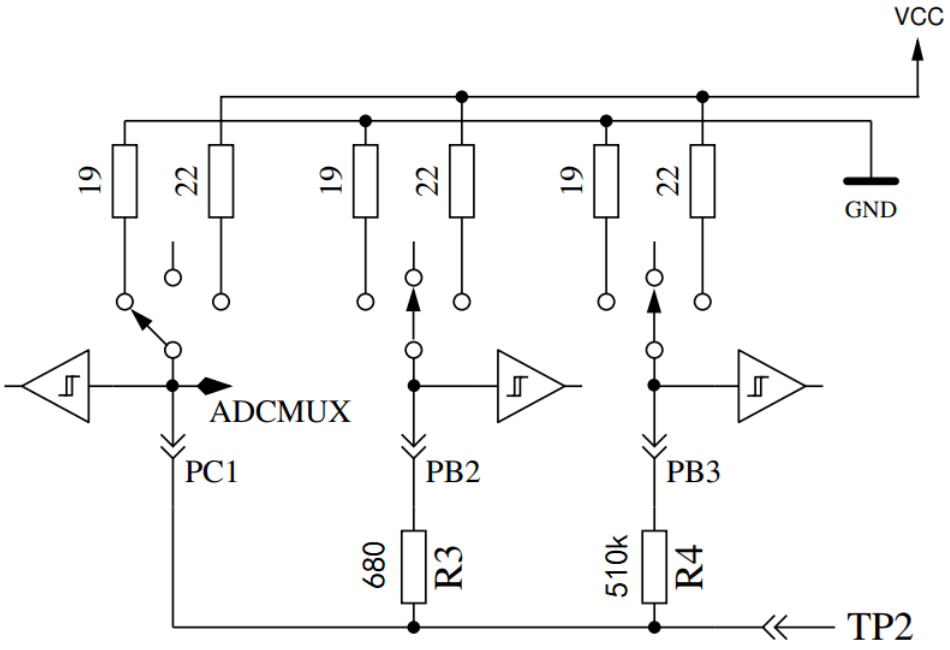


图 7: 元件测试仪管脚结构

以测试管脚 TP2 为例，其内部包含两个数字端口和一个模拟端口，每个端口可以存在高电压、低电压、悬空三种状态，两个数字端口的端口电阻不同，一个为 680Ω 小电阻，另一个为 510kΩ 大电阻。通过单片机编程，可以实现端口连接状态的设置。其余两个测试管脚 TP1 和 TP3 具有与 TP2 完全相同的结构。

3.2 电路构造

元件测试仪的电路通过面包板连接。其中，Arduino Nano 单片机插在面包板的左侧，方便串口数据线的连接。高电压端口 V_{cc} 和低电压端口 0V 通过两根导线引出，方便外部调试。需要用到的单片机端口均用导线引出，与相应大小的电阻串联后接在一起，构成三个测试管脚。

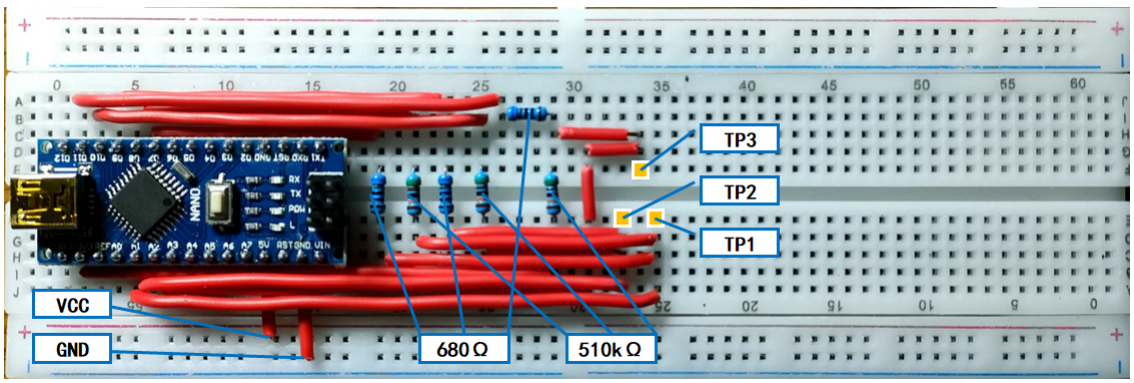


图 8: 元件测试仪电路构造

上图标出了电路中各个电阻的阻值以及测试管脚的位置。其中，三个测试管脚距离较近，近似构成一个等边三角形，这种设计是为了方便待测三极管元件的接入。所有导线和电阻均被裁剪为合适的长度，使布线紧贴面包板，视觉上比较美观，容易调试，且不容易出现接触不良等问题。

实际布线中，单片机各个端口与测试管脚的对应关系如下表所示：

表 1: 端口命名及对应关系

测试端口	端口类型	单片机端口	端口电阻	程序命名	程序端口
TP1	Analog	A0	0	TP[1].M	A0
TP1	Digital	D8	680	TP[1].L	8
TP1	Digital	D9	510k	TP[1].H	9
TP2	Analog	A1	0	TP[2].M	A1
TP2	Digital	D10	680	TP[2].L	10
TP2	Digital	D11	510k	TP[2].H	11
TP3	Analog	A2	0	TP[3].M	A2
TP3	Digital	D12	680	TP[3].L	12
TP3	Digital	D13	510k	TP[3].H	13

4 程序设计

4.1 主控程序

首先，我们定义一些全局变量，方便统一程序的编写。将 `unsigned long long` 类型缩写为 `ulong`，便于声明保存时间的变量。定义 `TestPin` 类，并构造 TP1、TP2、TP3 三个实例，将电路中对应管脚进行常量编码，提高程序可读性。`Point` 数组保存输入点对信息，用于简化代码。`Result` 数组和 `Buffer` 数组均为缓冲数组，用于电压读取和字符串输出。

```
#pragma once

// short for unsigned type
typedef unsigned long ulong;

// define test pin type
struct TestPin
{
    int M, L, H;
    TestPin(int _M = 0, int _L = 0, int _H = 0): M(_M), L(_L), H(_H) {}
};

// test pins
const TestPin TP[4] = {
    TestPin(0, 0, 0),
    TestPin(A0, 8, 9),
    TestPin(A1, 10, 11),
    TestPin(A2, 12, 13),
};

// connection information
```

```
const int Point[6][2] = {
   {1, 2}, {2, 1},
   {1, 3}, {3, 1},
   {2, 3}, {3, 2},
};

// measure result
int Result[4];

// output buffer
char Buffer[100];
```

主控程序包括 setup 和 loop 两个函数，其中 setup 用于程序的初始化，我们在此将单片机串口的传输速率设置为 9600，以便我们从控制台正确获取输出，同时将所有管脚设置为悬空状态。

```
// initialization
void setup()
{
    Serial.begin(9600);   // start serial
    clearPinMode();       // reset pin mode
}
```

loop 函数是程序的主函数，负责控制整个测量流程。根据前文所述的测量设计，我们首先检测接入元件类型。若元件为三极管，则进行三极管相关参数的测量，否则该元件为二端元件。我们首先需要判断其所处的管脚位置，确定位置以后，按照预先设计的方案对目标位置进行类型检测，若为电阻则进行电阻的测量，若为电容则进行电容的测量。若所有位置均未检测到元件，则认为测试仪处于空载状态，不产生任何输出。最后利用 delay 函数产生 1000 毫秒的延时，保证测试仪以合理且恒定的频率进行测量。

```
// main program
void loop()
{
    if (isTransistor())
        measureTransistor();
    else
    {
        for (int i = 0; i < 3; i++)
        {
            int x = Point[i][0], y = Point[i][1];
            if (isResistor(x, y) && isResistor(y, x))
            {
                measureResistor(x, y);
                break;
            }
            else if (isCapacitor(x, y) && isCapacitor(y, x))
            {
                measureCapacitor(x, y);
                break;
```

```
            }
        }
    }
    delay(1000);
}
```

4.2 工具函数

为方便程序实现，我们将常用的代码封装为工具函数，保存在 `utils.h` 中以供调用。根据前文所述，由于 `analogRead` 函数速度过慢，我们通过修改采样率的方法提高测量速度，`setPrescaler` 函数用于修改采样率，以提高时间的测量精度。

```
// set prescaler
void setPrescaler(char type)
{
    ADCSRA &= ~(bit (ADPS0) | bit (ADPS1) | bit (ADPS2));
    if (type == 'H')      // 128
        ADCSRA |= bit (ADPS0) | bit (ADPS1) | bit (ADPS2);
    else if (type == 'L') // 16
        ADCSRA |= bit (ADPS2);
}
```

`clearPinMode` 函数用于重置管脚状态，保证测量的实现的正确性。

```
// reset pin mode
void clearPinMode()
{
    for (int i = 1; i <= 3; i++)
    {
        pinMode(TP[i].M, INPUT);
        pinMode(TP[i].L, INPUT);
        pinMode(TP[i].H, INPUT);
    }
    delay(1);
}
```

`setPinMode` 函数用于设置指定管脚状态，集成了 `pinMode` 和 `digitalWrite` 两个函数，简化代码，提高可读性。

```
void setPinMode(int x, char t, int v)
{
    if (t == 'M')
    {
        pinMode(TP[x].M, OUTPUT);
        digitalWrite(TP[x].M, v);
    }
    else if (t == 'L')
    {
        pinMode(TP[x].L, OUTPUT);
```

```
        digitalWrite(TP[x].L, v);
    }
    else if (t == 'H')
    {
        pinMode(TP[x].H, OUTPUT);
        digitalWrite(TP[x].H, v);
    }
}
```

dischargeCapacitor 函数用于指定管脚间电容的放电，通过两侧接地并给予足够延迟的方式实现。

```
// discharge capacitor
void dischargeCapacitor(int x, int y)
{
    clearPinMode();
    setPinMode(x, 'L', LOW);
    setPinMode(y, 'M', LOW);
    Result[0] = analogRead(TP[x].M);
    while (Result[0] > 1)
        Result[0] = analogRead(TP[x].M);
    delay(10);
}
```

isFeature 函数用于2.4.1中所述 “奇异状态” 的判断，简化代码逻辑。

```
// transistor feature
bool isFeature()
{
    if (Result[1] >= 1000 && Result[2] <= 10 && Result[3] <= 10)
        return true;
    if (Result[1] <= 10 && Result[2] >= 1000 && Result[3] <= 10)
        return true;
    if (Result[1] <= 10 && Result[2] <= 10 && Result[3] >= 1000)
        return true;
    return false;
}
```

readTripleResult 函数用于同时读取三个测试管脚的电压。

```
// read three fixed results
void readTripleResult()
{
    Result[1] = analogRead(TP[1].M);
    Result[2] = analogRead(TP[2].M);
    Result[3] = analogRead(TP[3].M);
}
```

4.3 电阻测量的实现

isResistor 函数用于测试指定管脚之间是否为电阻元件，具体方法如前文所述。

```
// try resistor
bool isResistor(int x, int y)
{
    int last = 0;
    clearPinMode();
    dischargeCapacitor(x, y);
    clearPinMode();
    setPinMode(x, 'L', HIGH);
    setPinMode(y, 'L', LOW);
    delay(200);
    Result[1] = analogRead(TP[x].M);
    Result[0] = analogRead(TP[y].M);
    return Result[0] > 2 || Result[1] < 1015;
}
```

measureResistor 函数用于测量指定管脚之间电阻的阻值，如前文所述，首先选用大电阻 R_H 与待测电阻 R_x 进行串联，按照分压公式计算阻值。若测得阻值过小，则选用小电阻 R_L 与待测电阻 R_x 进行串联，重新测量，最后输出测量结果。

```
// measure resistor
void measureResistor(int x, int y)
{
    long resistance = 0;
    // large scale
    clearPinMode();
    setPinMode(x, 'H', HIGH);
    setPinMode(y, 'M', LOW);
    delay(1);
    Result[0] = analogRead(TP[x].M);
    if (Result[0] > 1000)
    {
        sprintf(Buffer, "[Resistor]\tInfinity");
        Serial.println(Buffer);
        return;
    }
    resistance = double(510000) / (double(1023) / Result[0] - double(1));
    if (Result[0] <= 20)
    {
        // tiny scale
        clearPinMode();
        setPinMode(x, 'L', HIGH);
        setPinMode(y, 'M', LOW);
        delay(1);
        Result[0] = analogRead(TP[x].M);
```

```
        resistance = double(680) / (double(1023) / Result[0] - double(1));
    }
    sprintf(Buffer, "[Resistor]\t%ld ohm", resistance);
    Serial.println(Buffer);
}
```

4.4 电容测量的实现

isCapacitor 函数用于测试指定管脚之间是否为电容元件，具体方法如前文所述。

```
// try capacitor
bool isCapacitor(int x, int y)
{
    double capacitance = calculateCapacitance(x, y);
    return capacitance > 1e-5;
}
```

calculateCapacitance 函数用于测量指定管脚之间的充电时间常数 τ，计算并返回相应的电容大小。该函数首先将小电阻 R_L 与电容串联，测量时间常数。若时间常数大于指定阈值，则待测电容为大电容，直接采用该结果。否则待测电容为小电容，调用 setPrescaler 调高采样率以获得更精确的时间测量结果，将大电阻 R_H 与电容串联，测量时间常数，计算实际电容大小。在函数返回结果之前，我们需要将采样率调低，以保证其他函数电压测量的精确性。

```
// calculate capacitance
double calculateCapacitance(int x, int y)
{
    ulong start = 0, finish = 0;
    long last = 0, count = 0;
    int value = 0;
    clearPinMode();
    dischargeCapacitor(x, y);
    clearPinMode();
    setPinMode(x, 'L', HIGH);
    setPinMode(y, 'M', LOW);
    count = 0;
    value = analogRead(TP[x].M);
    start = micros();
    while (++count <= 500 && value < 647)
        value = analogRead(TP[x].M);
    finish = micros();
    if (count > 500)
        return double(0);
    last = max(long(finish - start), 0);
    if (last > 500)
        return last / double(720);
    clearPinMode();
    dischargeCapacitor(x, y);
```

```
    clearPinMode();
    setPrescaler('L');
    setPinMode(x, 'H', HIGH);
    setPinMode(y, 'M', LOW);
    count = 0;
    value = analogRead(TP[x].M);
    start = micros();
    while (++count <= 5000 && value < 647)
        value = analogRead(TP[x].M);
    finish = micros();
    setPrescaler('H');
    if (count > 5000)
        return double(0);
    last = max(long(finish - start) - 20, 0);
    return last / double(510000);
}
```

measureCapacitor 函数用于测量指定管脚之间的电容大小，该函数在确定元件类型的情况下直接调用 calculateCapacitance 函数，并格式化输出结果。

```
// measure capacitor
void measureCapacitor(int x, int y)
{
    double capacitance = calculateCapacitance(x, y);
    Serial.print("[Capacitor]\t");
    Serial.print(capacitance, 6);
    Serial.println(" uF");
}
```

4.5 三极管测量的实现

isTransistor 函数用于测试指定管脚之间是否为三极管元件，具体方法如前文所述。由于测试管脚 TestPin 类的定义，我们可以方便地遍历三种接法并设置管脚状态，这使代码简洁美观。

```
// try transistor
bool isTransistor()
{
    int count = 0;
    for (int i = 1; i <= 3; i++)
    {
        clearPinMode();
        for (int j = 1; j <= 3; j++)
            setPinMode(j, 'L', LOW);
        setPinMode(i, 'L', HIGH);
        delay(1);
        if (analogRead(TP[1].M) >= 100 && analogRead(TP[2].M) >= 100 &&
    analogRead(TP[3].M) >= 100)
```

```
            return true;
    }
    return false;
}
```

measureTransistor 函数用于控制三极管元件的测量流程，并完成三极管的类型判断和基极位置的确定，具体方法如前文所述。通过遍历三种接法，统计“奇异状态”出现的次数。若恰好出现一次，我们断言三极管为 PNP 型，若恰好出现两次，我们断言三极管为 NPN 型。根据三极管的类型和基极的位置，我们调用相应的函数进行其余参数的测量。

```
// measure transistor
void measureTransistor()
{
    bool feature[4];
    int count = 0;
    for (int i = 1; i <= 3; i++)
    {
        clearPinMode();
        for (int j = 1; j <= 3; j++)
            setPinMode(j, 'L', LOW);
        setPinMode(i, 'L', HIGH);
        delay(1);
        readTripleResult();
        if (feature[i] = isFeature())
        count++;
    }
    if (count == 1)
    {
        Serial.println("[Transistor]\tPNP Type");
        if (feature[1])
            analyseTransistorPNP(1, 2, 3);
        else if (feature[2])
            analyseTransistorPNP(2, 1, 3);
        else
            analyseTransistorPNP(3, 1, 2);
    }
    else if (count == 2)
    {
        Serial.println("[Transistor]\tNPN Type");
        if (!feature[1])
            analyseTransistorNPN(1, 2, 3);
        else if (!feature[2])
            analyseTransistorNPN(2, 1, 3);
        else if (!feature[3])
            analyseTransistorNPN(3, 1, 2);
    }
}
```

`analyseTransistorPNP` 函数用于分析 PNP 型三极管发射极和集电极的位置，具体方法如前文所述。确定位置后，输出相应的管脚信息，并将这些信息作为参数，调用 `calculateTransistorPNP` 函数测量相应的放大倍数 β。

```
// analyse PNP transistor
void analyseTransistorPNP(int x, int y, int z)
{
    sprintf(Buffer, "\tBase:\t\tTP%d", x);
    Serial.println(Buffer);
    // high on TPx, low on TPy and TPz
    clearPinMode();
    setPinMode(x, 'H', HIGH);
    setPinMode(y, 'H', LOW);
    setPinMode(z, 'H', LOW);
    delay(10);
    readTripleResult();
    if (Result[y] >= 100)
    {
        sprintf(Buffer, "\tCollector:\tTP%d", y);
        Serial.println(Buffer);
        sprintf(Buffer, "\tEmitter:\tTP%d", z);
        Serial.println(Buffer);
        calculateTransistorPNP(x, y, z);
    }
    else
    {
        sprintf(Buffer, "\tCollector:\tTP%d", z);
        Serial.println(Buffer);
        sprintf(Buffer, "\tEmitter:\tTP%d", y);
        Serial.println(Buffer);
        calculateTransistorPNP(x, z, y);
    }
}
```

`calculateTransistorPNP` 函数用于测量 PNP 型三极管的放大倍数 β，具体方法如前文所述。该函数根据指定阈值划分两种测量范围，选取更精确的测量结果并格式化输出。

```
// calculate PNP transistor factor
void calculateTransistorPNP(int base, int collector, int emitter)
{
    double beta = 0.0;
    clearPinMode();
    setPinMode(base, 'L', LOW);
    setPinMode(collector, 'M', LOW);
    setPinMode(emitter, 'L', HIGH);
    delay(1);
    Result[0] = analogRead(TP[base].M);
```

```
    Result[1] = analogRead(TP[emitter].M);
    if (Result[0] >= 2)
        beta = double(Result[1] - Result[0]) / Result[0];
    else
    {
        clearPinMode();
        setPinMode(base, 'H', LOW);
        setPinMode(collector, 'M', LOW);
        setPinMode(emitter, 'L', HIGH);
        delay(1);
        Result[0] = analogRead(TP[base].M);
        Result[1] = analogRead(TP[emitter].M);
        beta = Result[1] * double(510000) / Result[0] / double(700);
    }
    Serial.print("\tBeta:\t\t");
    Serial.println(beta, 3);
}
```

`analyseTransistorNPN` 函数用于分析 NPN 型三极管发射极和集电极的位置，具体方法如前文所述。确定位置后，输出相应的管脚信息，并将这些信息作为参数，调用 `calculateTransistorNPN` 函数测量相应的放大倍数 β。

```
// analyse NPN transistor
void analyseTransistorNPN(int x, int y, int z)
{
    sprintf(Buffer, "\tBase:\t\tTP%d", x);
    Serial.println(Buffer);
    // high on TPy, low on TPz, high on TPx with resistor
    clearPinMode();
    setPinMode(x, 'L', HIGH);
    setPinMode(y, 'M', HIGH);
    setPinMode(z, 'M', LOW);
    delay(1);
    Result[y] = analogRead(TP[x].M);
    // high on TP3, low on TP2, high on TP1 with resistor
    clearPinMode();
    setPinMode(x, 'L', HIGH);
    setPinMode(y, 'M', LOW);
    setPinMode(z, 'M', HIGH);
    delay(1);
    Result[z] = analogRead(TP[x].M);
    if (Result[y] > Result[z])
    {
        sprintf(Buffer, "\tCollector:\tTP%d", y);
        Serial.println(Buffer);
        sprintf(Buffer, "\tEmitter:\tTP%d", z);
        Serial.println(Buffer);
```

```
        calculateTransistorNPN(x, y, z);
    }
    else
    {
        sprintf(Buffer, "\tCollector:\tTP%d", z);
        Serial.println(Buffer);
        sprintf(Buffer, "\tEmitter:\tTP%d", y);
        Serial.println(Buffer);
        calculateTransistorNPN(x, z, y);
    }
}
```

calculateTransistorNPN 函数用于测量 NPN 型三极管的放大倍数 β，该函数通过设置测试管脚的状态构造基本共射放大电路，按照定义式计算放大倍数 β 并格式化输出。

```
// calculate NPN transistor factor
void calculateTransistorNPN(int base, int collector, int emitter)
{
    double ib = 0.0, ic = 0.0, beta = 0.0;
    clearPinMode();
    setPinMode(base, 'H', HIGH);
    setPinMode(collector, 'L', HIGH);
    setPinMode(emitter, 'M', LOW);
    delay(10);
    ib = (1023 - analogRead(TP[base].M)) * double(5) / double(1023) / double
    (510000);
    ic = (1023 - analogRead(TP[collector].M)) * double(5) / double(1023) /
    double (700);
    beta = ic / ib;
    Serial.print("\tBeta:\t\t");
    Serial.println(beta, 3);
}
```

5 创新点

5.1 空载状态检测

在常见的工程项目设计中，通常采用 GIGO（Garbage In Garbage Out）原则，只针对合法的输入进行正确的运算操作，对于设计之外的不合理输入，并不关心结果如何。在本实验的元件测试仪中，只存在电阻、电容、三极管三种元件，而对于测试仪空载的状态，输出结果并没有严格的规定。对于无元件相连的两个管脚，我们既可以认为是一个无穷大的电阻，也可以认为是一个无穷小的电容。但我们设计的元件测试仪通过合理的流程和方案，保证三种元件都可以被独立且正确地检测，因此经过排除可以确定空载状态，并不产生任何输出。这种对空载状态的检测使元件测试仪的输出结果更易读，对产品的使用者更加友好。

5.2 测试管脚强兼容性

本实验的元件测试仪包含三个测试管脚，支持二端和三端两类元件的测试。对于二端元件的测试，我们可以要求使用者选择指定的两个管脚，这样可以大大简化流程设计和程序编写。然而，区分三个等价的管脚对于使用者而言是极不友好的，因此我们通过思考反复修改设计，使得三个管脚具有等价的测试性能。对于电阻和电容的，接在任意两个测试端口之间均可以正确识别和准确测量。这种测试管脚的强兼容性为实现增加了难度，但具有较高的实用价值。

5.3 放大系数测量

本次实验中我们对于 NPN 和 PNP 两种类型的三极管的正向电流放大倍数 β 采取了不同的测量方式，其中 NPN 型三极管的测量方式基于模拟电子技术课程中的共射放大电路，完全由我们自己设计，并且测量误差在可接受范围以内，体现课程要求的了良好的跨学科科学思维能力和课本知识应用能力。对于 PNP 型三极管的测量，则通过仔细研读文档并多次实践摸索得出，在有限的电压和电阻档选择中实现了测量误差较小的实验方案，同样体现了对书面知识的实际应用能力以及文档文献查阅能力。

6 分工介绍

我们小组的任务分工如下表所示：

表 2: 任务分工安排表

分工内容	薛翔元	卜家梓	李奇睿
电路设计和实现	✓		
电阻测量方案	✓		
电容测量方案	✓		✓
三极管测量方案		✓	
电路仿真实验		✓	
程序编写和调试	✓		
实际运行和测试	✓		
报告撰写和排版	✓	✓	✓

参考文献

[1] Karl-Heinz Kübbeler. TransistorTester with AVR microcontroller and a little more Version 1.12k. https://www.academia.edu, 2017.

[2] Sedra A S, Smith K C, Carusone T C, et al. Microelectronic circuits[M]. New York: Oxford university press, 2004.

[3] Arduino Website. Reference. https://www.arduino.cc/reference.

[4] 童诗白, 华成英. 模拟电子技术基础 [M]. 北京: 高等教育出版社, 2015.