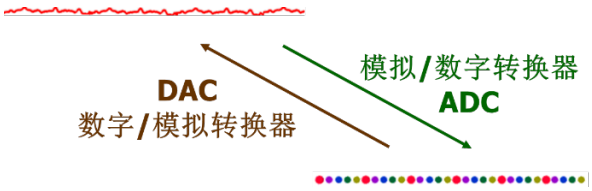
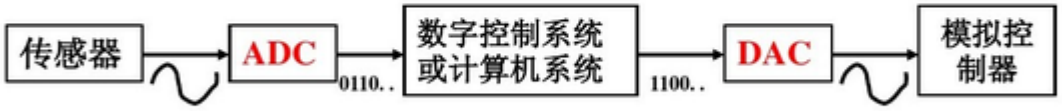
12. DAC 和 ADC

1) DAC 和 ADC 的基本概念

有了异步串行通信，就可以通过虚拟串口，轻松实现嵌入式系统和 PC、服务器的数据交互。可以通过微控制器采集传感器的数据，发送给 PC 进行记录、显示和处理。
我们已经学习过使用 IIC 总线读取加速度传感器的数据。除了加速度传感器外，还有很多传感器的信号是以模拟信号呈现的。
模拟信号的特点是时间连续、幅值连续。由于现在的计算机几乎已经占领了各个领域，因此这些领域对信号的处理都要转换成计算机能看得懂的信号，然后再交给计算机来处理，这些计算机能看懂的信号就是数字信号。那么将模拟信号转换成数字信号需要一种器件，它叫做模数转换器(ADC)。以此类推，既然 ADC 是将模拟量转换成数字量，那么也还有一种器件能把数字量转换成模拟量，它叫数模转换器(DAC)。



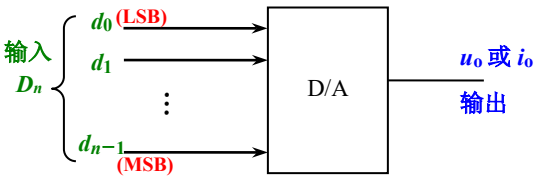


2) 数字/模拟转换器 DAC

DAC 是将数字量转换成模拟量输出的设备。
DAC 将输入的二进制数字量转换成模拟量，以电压或电流的形式输出。
DAC 实质上是一个译码器（解码器），提供了数字量到模拟量的映射功能。一般常用的线性 DAC，其输出模拟电压 u_O 和输入数字量 D_n 之间成正比关系：

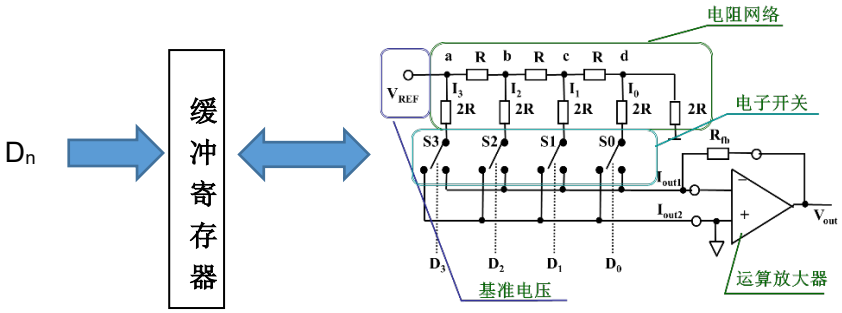
$$u_O = D_n \lozenge U_{REF}$$

U_{REF} 为参考电压。

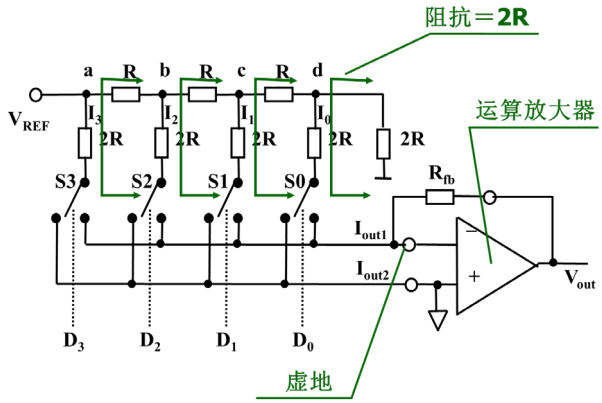


D/A 转换的基本原理

DAC 一般由缓冲寄存器、模拟电子开关、参考电压、解码网络和求和电路等组成。一个四位 DAC 的原理示意图如图所示。



其中数字输入 D_n 写入到缓冲寄存器后，由寄存器的每一个位，控制一个模拟电子开关。
本 DAC 的解码网络由倒 T 型电阻网络组成，是 DAC 转换的关键。由运放组成求和电路。
电阻网络设计的十分巧妙，由电阻值为 R 和 2R 的电阻组成，因此这种结构也被称作 R-2R 结构。
在 a、b、c、d 每一个点，向右看，阻抗都是 2R，



而运放的同相输入端接地，根据运放虚短的特点，无论 S0、S1、S2、S3 四个模拟电子开关向哪个方向拨，开关后的电位都是零。a、b、c、d 四个点的电压为：

Va＝VREF

Vb＝VREF/2

Vc＝VREF/4

Vd＝VREF/8

那么，电流 I_0、I_1、I_2、I_3 可以计算为：

I_0＝Vd/2R＝VREF/（8×2R）

I_1＝Vc/2R＝VREF/（4×2R）

I_2＝Vb/2R＝VREF/（2×2R）

I_3＝Va/2R＝VREF/（1×2R）

这时，输入量 Dn 的每一位，控制一个电子开关。

Dn 的第 0 位 Dn(0)控制 S0，如果 Dn(0)为 1，那么 S0 向右拨，I_0流向运放的反向输入端，如果 Dn(0)为 0，那么 S0 向左拨，I_0流向运放的同向输入端，即流入地。
Dn 的第 1 位 Dn(1)控制 S1，如果 Dn(1)为 1，那么 S1 向右拨，I_1流向运放的反向输入端，如果 Dn(1)为 0，那么 S1 向左拨，I_1流向运放的同向输入端，即流入地。
Dn 的第 2 位 Dn(2)控制 S2，如果 Dn(2)为 1，那么 S2 向右拨，I_2流向运放的反向输入端，如果 Dn(2)为 0，那么 S2 向左拨，I_2流向运放的同向输入端，即流入地。
Dn 的第 3 位 Dn(3)控制 S3，如果 Dn(3)为 1，那么 S3 向右拨，I_3流向运放的反向输入端，如果 Dn(3)为 0，那么 S3 向左拨，I_3流向运放的同向输入端，即流入地。
那么，流向运放反向输入端的电流为：

I_{out1}＝Dn(0)◊I_0＋Dn(1)◊I_1＋Dn(2)◊I_2＋Dn(3)◊I3

＝V_{REF}/2R×（Dn(0)◊1/8＋Dn(1)◊ 1/4＋Dn(2)◊1/2＋Dn(3)◊1）

那么运放的输出电压为：

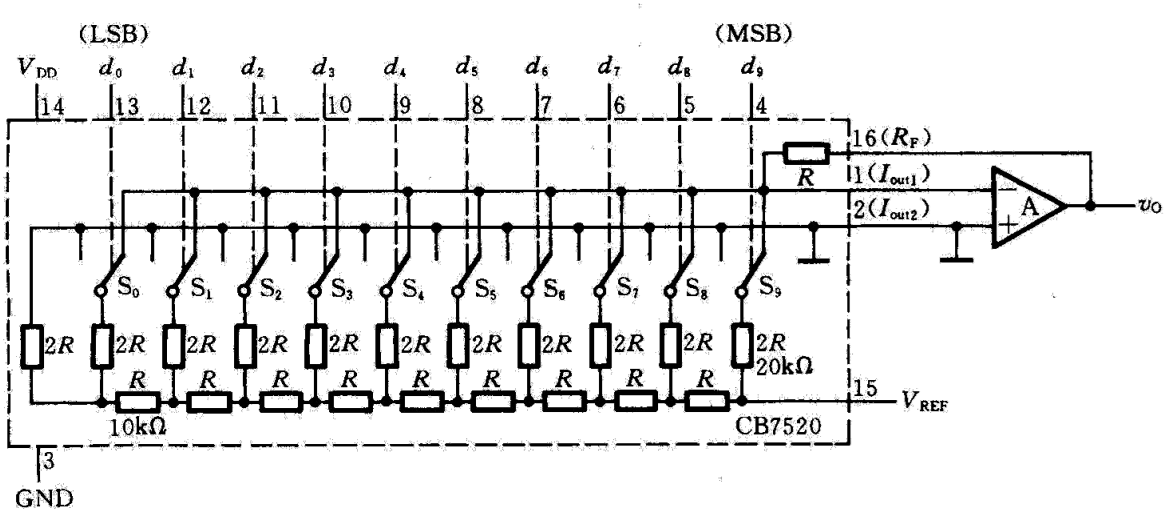
V_{out}＝－I_{out1}×R_{fb}

如果 R_{fb} =R，则

V_{out}＝－V_{REF}×[（Dn(0)◊2^0＋Dn(1)◊2^1＋Dn(2)◊2^2＋Dn(3)◊2^3）/2^4]

可见，DAC 将输入的每一位二进制代码，按其权值大小转换成相应的模拟量，然后将代表各位的模拟量相加，则所得的总模拟量就与数字量成正比，这样便实现了从数字量到模拟量的转换。

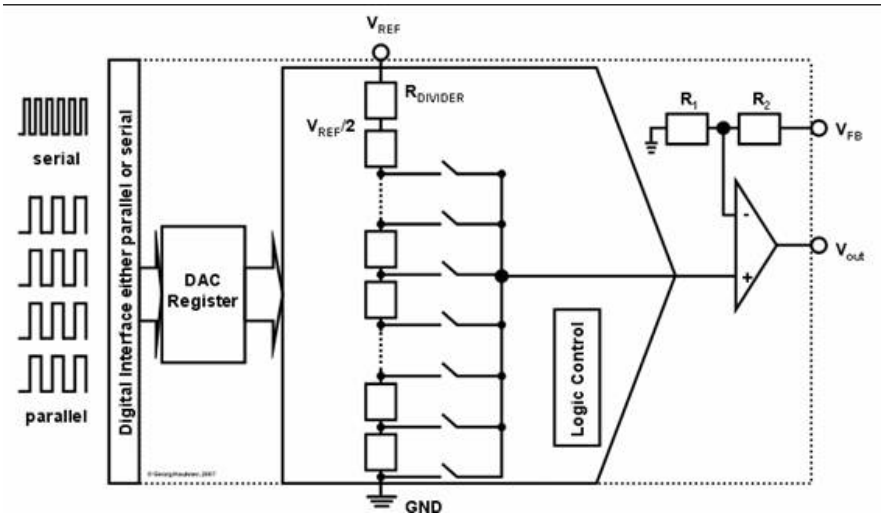
应用示例(10 位 DAC CB7520)：



DAC——CB7520 电路原理图

其他结构的 DAC：

倒 T 型网络 DAC 也被称作 **R2R 架构** DAC。除此之外，常用的还有电阻串(R-String)结构的 DAC。



顾名思义，电阻串架构就是一个以串联形式放置的一串电阻，以构建一个电阻串。从理论上来说，可能会需要 256 个电阻才能构建一款 8 位 DAC (28 = 256)，因此，设计人员推出了其它更小的电路设计方案，如可降低电阻串上所需电阻数量以及接触点的内插式放大器，从而实现了功耗更低且更节省空间的设计。

DAC 的主要技术指标

1. **转换精度**

   输出模拟电压的实际值与理想值之差，也称最大静态转换误差。

2. **分辨率**

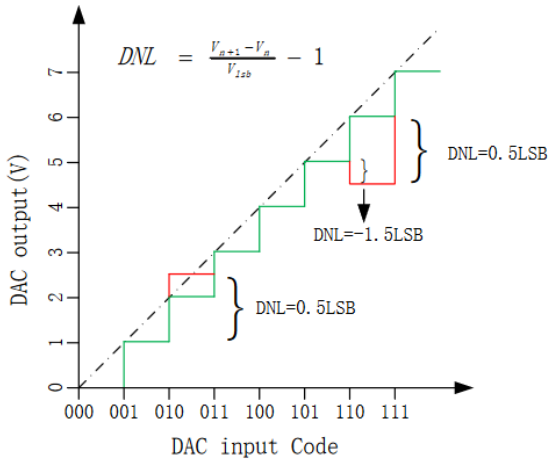
   DAC 模拟输出电压可能被分离的等级数。

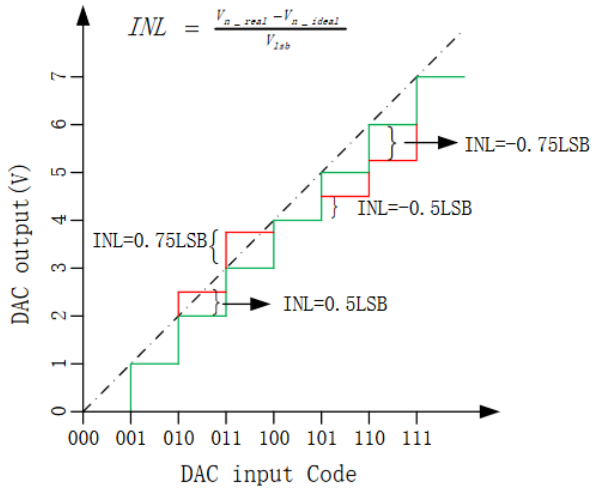
   输入数字量位数越多，分辨率越高。所以，在实际应用中，常用字量的位数表示 DAC 的分辨率，如 CB7520 为 10 位 DAC

   此外，也可用 DAC 的最小输出电压与最大输出电压之比来表示分辨率，N 位 DAC 的分辨率可表示为 1/（2n-1）。

3. **线性度**

   通常用非线性误差的大小表示 DAC 的线性度。

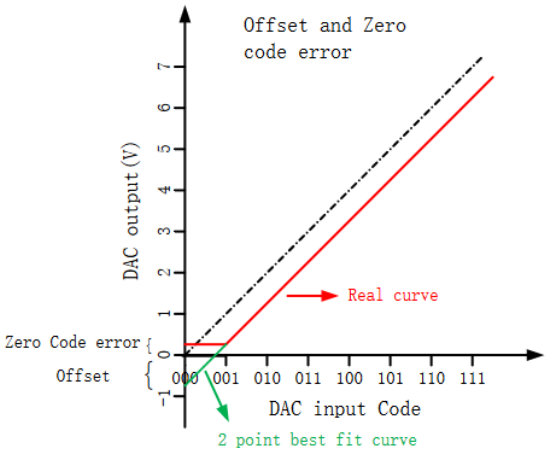


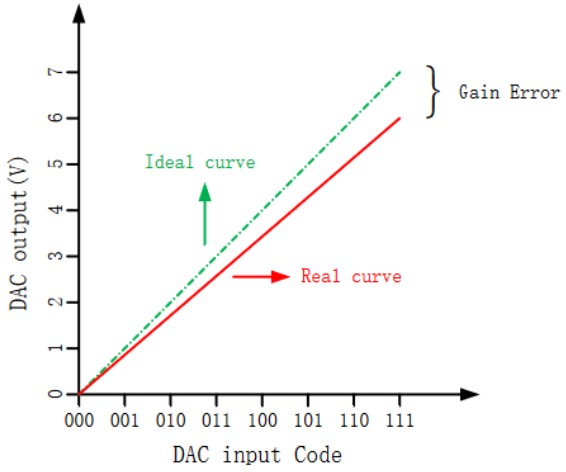


DNL 是微分线性度，指相邻两个输出电平的差相对于理想值（1LSB）的偏差。 datasheet 中的 DNL 代表所有台阶中最大的偏差值。由上图可以看到，如果出现 DNL<-1LSB 的现象，则 DAC 的输出肯定是非单调的，也就是说数字编码增加 1，输出不增加反而会下降。这一点在很多闭环系统应用中是不能接受的，如果 DAC 出现非单调的情况，则控制环路无法收敛。这时一般会选择 DNL < ±1LSB 的器件。

INL 是积分线性度，指实际的输出相对理想 DAC 的输出之间的差异，所以也叫 relative accuracy，用满量程的百分比或者 LSB 来表示。理论上，某个编码对应输出的 INL 就是从第一个编码到这个编码所有的 DNL 的积分，也印证了“积分”这个名称的含义。Datasheet 中的 INL（或者 relative accuracy）代表所有输出值最大的 INL。这个指标用来衡量 DAC 输出的准确度如何，应用比较广。特别是在开环应用中，应当关注 INL 的指标。

4. 偏移



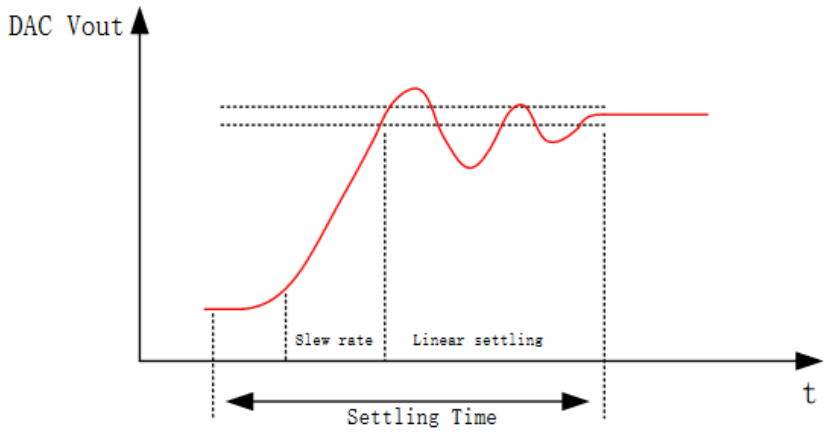


一个无限分辨率的理想 DAC 输出特性应该是通过原点的一条直线，y=x (这里我们把 DAC 增益相对理想值归一化成 1), 但实际的 DAC 输出特性，用靠近首尾两端的两点拟合一条直线， 特性一般是 y=ax+b。其中，a 代表 DAC 实际的输出增益，即 gain。其相对理想增益的偏差，即 gain error。b 代表这条直线整体相对原点向上或者向下偏移的幅度，即 offset error.

但实际 DAC 在 code 为 0 附近，输出电压也很低时，由于内部电路接近饱和（特别是带输出 buffer 的 DAC），会出现一定的非线性。所以 DAC 会有一个额外的参数来标定 code 为 0 时输出的偏差，叫 zero code error. 另外 gain，offset 在不同温度下也会产生变化，即 gain shift, offset error shift。

5. 转换速度

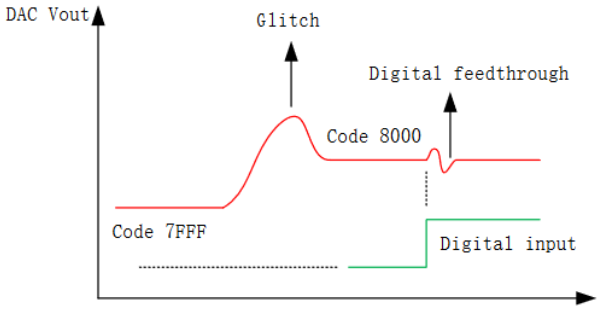
建立时间（Settling time）



输出从 0 到满摆幅变化（或者特定的两个差异较大的值）的总时间，称为 settling time。输出主要经历两个阶段，一是 slew rate， 二是 linear settling。slew rate 反映了输出大摆幅下的极限驱动能力，一般决定了输出 10%~90%变化的时间，。而 linear settling 则主要取决于输出节点的 RC 常数或者输出 buffer 的带宽。

Settling time 是用户考虑精密 DAC 速度的重点参数。

6. 毛刺（Glitch）



Glitch 主要与 DAC 核心部分的开关有关。 当内部开关从一个点切换到另一个点时， 会受到寄生电荷以及开关切换不能理想同步的影响，从而造成输出跳动。 跳动的幅度和时间都是我们关注的对象，所以 Glitch 用 nV*S 这个二者相乘的单位来表示其能量大小。从其产生原理可见， glitch 与具体切换的开关位置有关。Code 的高位 MSB 变化时一般会产生较大的 glitch，所以 datasheet 中普遍定义 major carry 处的 glitch。 Glitch 也和结构有关， R-string 的 glitch 一般比 R2R 结构的 glitch 小。

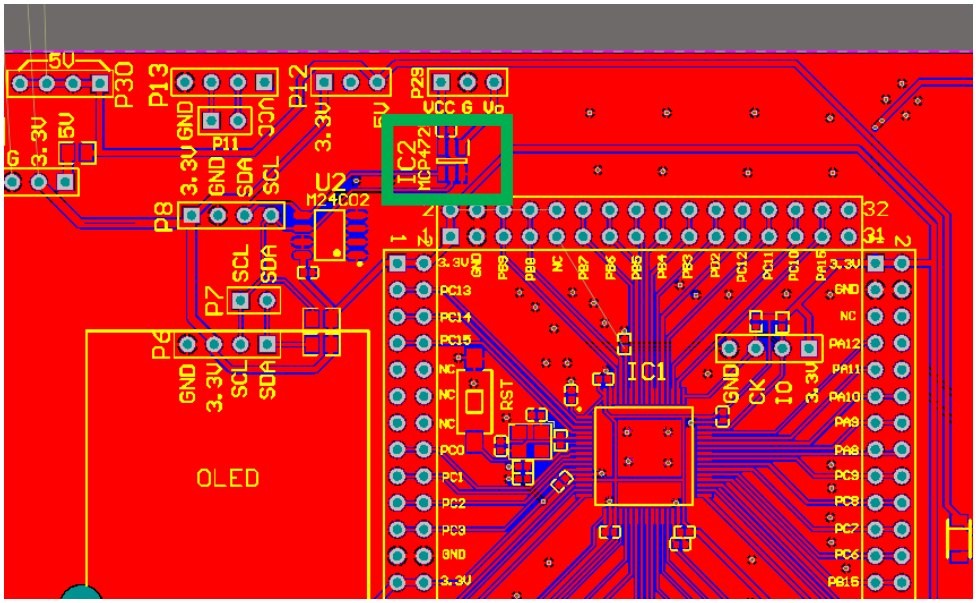
Digital feedthrough 则代表了模拟输出与数字输入的隔离程度。 即使 DAC 没有被选中进行通信，总线上的数字 IO 信号或时钟跳动通过内部信号通路或者电源地的耦合也会造成 DAC 输出的跳动，即为 digital feedthrough。 良好的设计可以保证这个值很小。

7. 温度系数

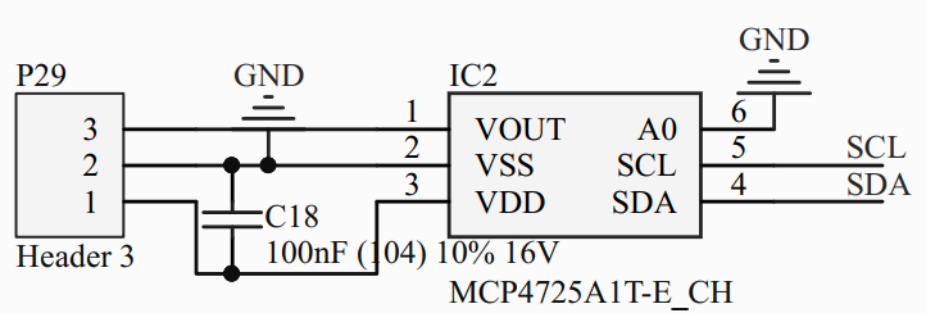
在输入不变的情况下，输出模拟电压随温度变化产生的变化量。一般用满刻度输出条件下温度每升高 1℃，输出电压变化的百分数作为温度系数。

3) 实验板上的 DAC

实验板上采用 MCP4725A1T 芯片，可将数字信号转换为模拟信号，他在实验板上的位置如下：

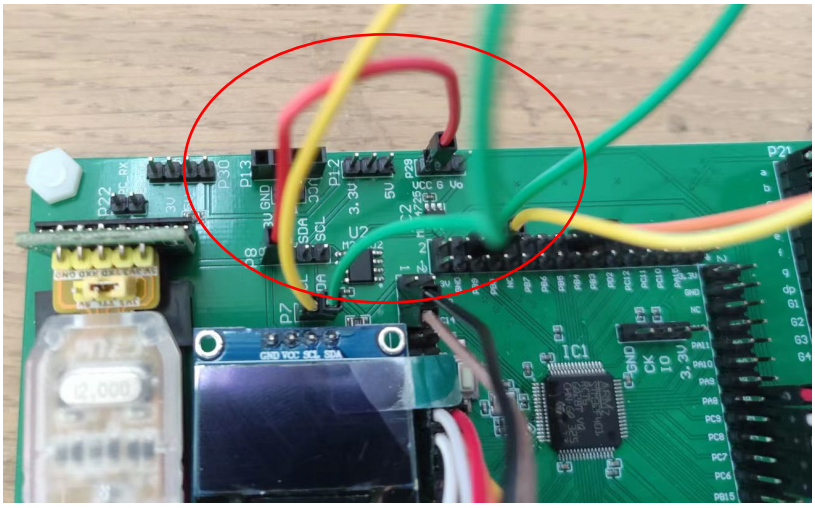


MCP4725A1T 芯片在实验板的电路原理图如下图所示：



从上图可知，该芯片需要通过 I2C 接口与 STM32 芯片相连。若要使用 I2C 向 MCP4725 芯片发送输出，需要完成下列操作：

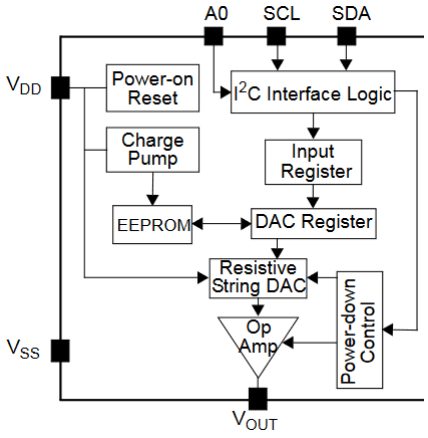
1. 线路连接



如上图所示，将 PB8 连接至 P7 端口的 SCL，将 PB9 连接至 P7 的 SDA 端口，将 P29 的 VCC 端口连接至 P8 的 3.3V，给 MCP4725 芯片供电。

2. MCP4725 芯片

其芯片结构如下：



可见，该芯片的数据接口为 I2C 总线，从 I2C 总线获取的数据存入 DAC 数据寄存器，内部有一个 EEPROM 可以存储 DAC 数据寄存器的值。其 DAC 的类型为电阻串类型，其供电电源为 VDD。内置了一个运算放大器用于输出。

在对 MCP4725 芯片进行进一步的配置之前，需要了解该芯片的基本特征。

MCP4725 DA 转换芯片有如下特点：

- 该芯片是 12 bit 的 DA 转换芯片
- 内部含有 EEPROM 存储
- 通过 I2C 接口和外部连接
- 提供了 8 个可选地址
- 可配置为 100kbps、400kbps 或 3.4Mbps 等不同通讯速率
- 使用 I2C 接口和外部通讯时，MCP4725 芯片工作为从机模式，需要主机提供时钟、开始位、结束位等信息。

MCP4725 芯片转换的 LSB 定义如下，其含义为数字输入变化 1 时，模拟输出变化的多少，某种程度上反映了 DAC 芯片的精度。由于芯片的 VCC 连接到 3.3V，该芯片是 12 位的，因此 LSB 近似为 8.06*10^-4V。

EQUATION 4-1:

$$LSB_{Ideal} = \frac{V_{REF}}{2^n} = \frac{(V_{Full\ Scale} - V_{Zero\ Scale})}{2^n - 1}$$

Where:

V_{REF} = The reference voltage = V_{DD} in the MCP4725. This V_{REF} is the ideal full scale voltage range

n = The number of digital input bits. (n = 12 for MCP4725)

MCP4725 芯片有 normal 和 power-down 两种工作模式，由内部寄存器的 PD0 和 PD1 两个位决定。当 PD0 和 PD0 均为 0 时，芯片正常运行；否则芯片工作于低功耗模式，芯片关闭除了 I2C 接口的其他电路，没有数据转换，V0 也没有输出，输出从运放的输出切换为阻性负载，其阻值见下表：

TABLE 5-2: POWER-DOWN BITS

PD1	PD0	Function
0	0	Normal Mode
0	1	1 kΩ resistor to ground (1)
1	0	100 kΩ resistor to ground (1)
1	1	500 kΩ resistor to ground (1)

Note 1: In the power-down mode: V_{OUT} is off and most of internal circuits are disabled.

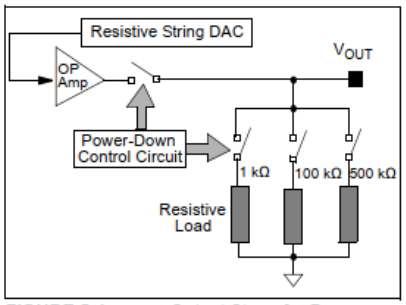


FIGURE 5-1: *Output Stage for Power-Down Mode.*

MCP4725 芯片的地址由 3 部分组成：device code + address code + R/W bit。其中 Device code 始终都是 1100；address code 由 A2、A1、A0 3 位组成，其中 A2 和 A1 在本芯片中为 01（见下图），A0 位由 A0 管脚的逻辑状态决定，由于 A0 在电路上被接地，所以 A0=0。因此，MCP4725 芯片的地址为 1100010b。

Part Number	Address Option	Code
MCP4725A0T-E/CH	A0 (00)	AJNN
MCP4725A1T-E/CH	A1 (01)	APNN
MCP4725A2T-E/CH	A2 (10)	AQNN
MCP4725A3T-E/CH	A3 (11)	ARNN

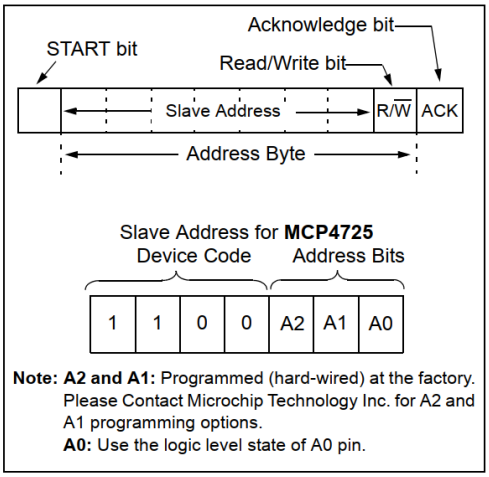


FIGURE 7-1: *Device Addressing.*

MCP4725 的写操作：可将配置位和 DAC 输入加载到 DAC 寄存器中，或者 EEPROM 中。
常用的写入操作如下所示：
起始位+从机地址（写）+字节 1+字节 2+停止位。

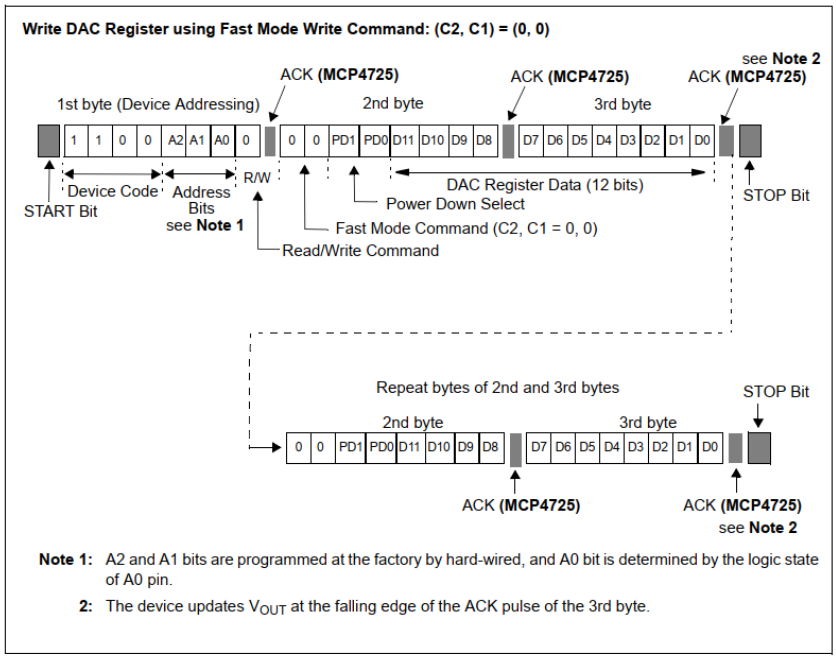


FIGURE 6-1: *Fast Mode Write Command.*

当 C2，C1 均为 0 时，芯片写操作工作于 Fast mode。该模式用于改变 DAC 寄存器，EEPROM 不受影响。可以用该模式设置下电模式选择位（PD1 和 PD0），以及 DAC 寄存器中的 12 位输入数据。在该模式下，需要主机先向 MCP4725 芯片发送起始位+从机地址（写），在 DA 芯片应答后，主机发送一个字节，包括 PD1 和 PD0 的配置位，以及数据信息的高 4 位（D11、D10、D9、D8）；最后，主机再发送一个字节，全部为数据信息（D7~D0）。当三个字节信息发送完毕后，MCP4725 芯片的 Vo 会在第三个字节的 ACK 下降沿处更新数据。具体见下图：

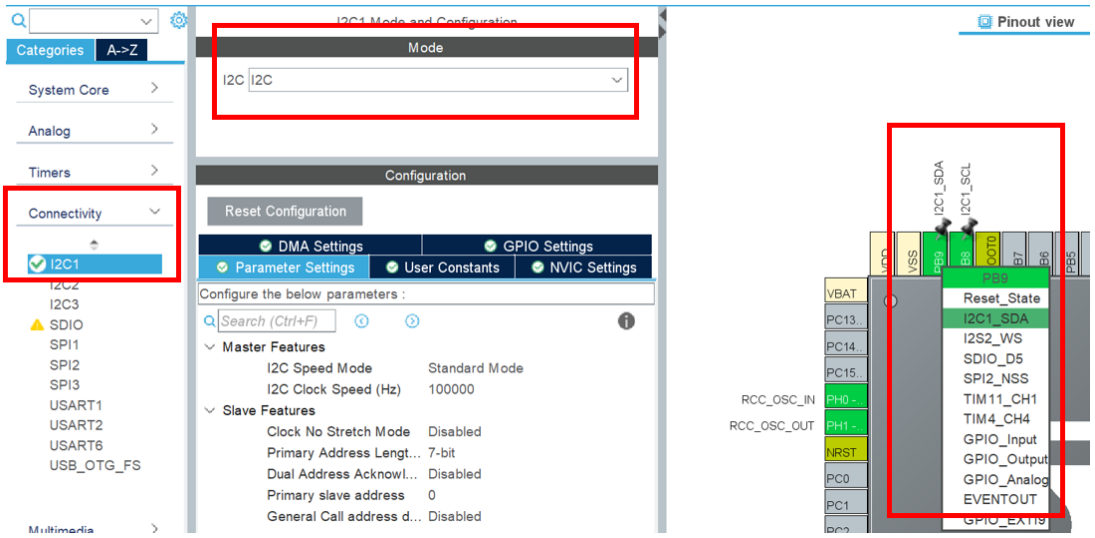
DAC 芯片的输入数字值和输出模拟值之间的对应关系如下表所示：

TABLE 6-1: INPUT DATA CODING

Input Code	Nominal Output Voltage (V)
111111111111 (FFFh)	V_{DD} - 1 LSB
111111111110 (FFEh)	V_{DD} - 2 LSB
000000000010 (002h)	2 LSB
000000000001 (001h)	1 LSB
000000000000 (000h)	0

3. 使用 STM32F401 芯片操作 MCP4725 芯片

首先复制已有的 HelloWorld 工程，修改好配置后，打开.ioc 文件，将 PB8 配置为 SCL，将 PB9 配置为 SDA。



使用 Fast mode 将要转换的数据发送至 DAC 芯片中，具体可参考如下代码：

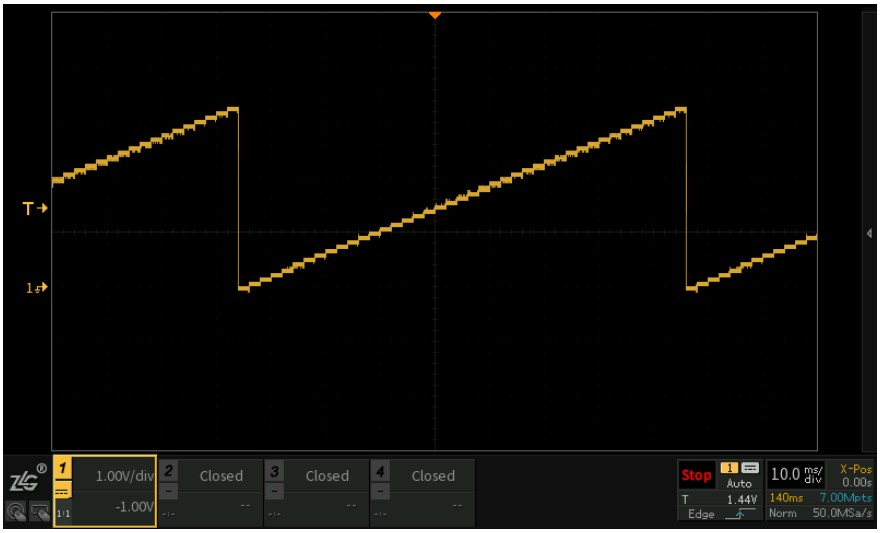
```
void WriteMCP4245(uint32_t val){

    uint32_t MCP4245Addr;
    MCP4245Addr=0x62;
    uint8_t dacData[2];

    dacData[0]=val>>8;
    dacData[1]=val&0xff;

    HAL_I2C_Master_Transmit(&hi2c1,MCP4245Addr<<1,dacData,2,100);

}
```

向 DAC 芯片发送一个递增变量（三角波），

```
  a+=10;
```

```
if(a>0x0fff){
    a=0;
}
WriteMCP4245(a);
HAL_Delay(10);
```

用示波器观察如下：



当然也可以使用寄存器操作完成发数据的功能。

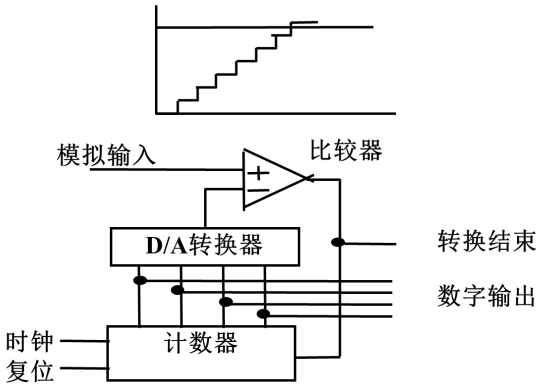
4) 模拟/数字转换器 ADC

要实现将连续变化的模拟量变为离散的数字量，有多种转换技术，各有特点，分别应用于不同的场合。以下 5 种是比较常用的 AD 转换技术：

- 计数器式
- 逐次逼近式
- 双积分式
- 并行式
- Delta-sigma

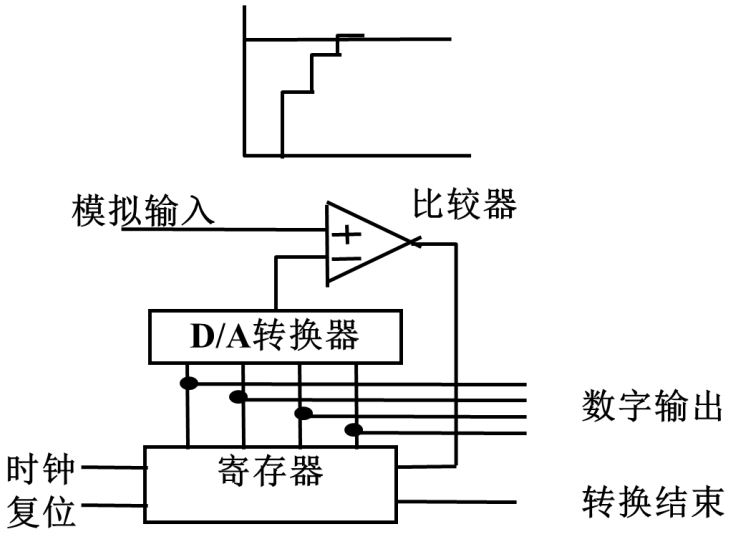
1. 计数器式

以最低位为增减量单位的逐步计数法



2. 逐次逼近式

从最高位开始的逐位试探法，类似于查找算法中的二分法。



其工作原理可用天平秤重过程作比喻来说明。若有四个砝码共重 15 克，每个重量分别为 8、4、2、1 克。设待秤重量 Wx = 13 克，可以用下表步骤来秤量：

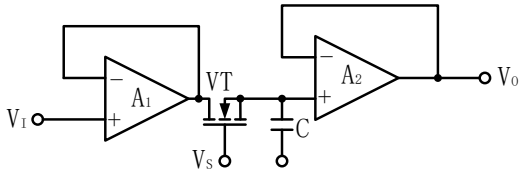
顺序	砝码重	比较判断	暂时结果
1	8 g	8g < 13g，保留	8 g
2	8 g + 4 g	12g < 13g，保留	12 g
3	8 g + 4 g + 2 g	14g > 13g，撤去	12 g
4	8 g + 4 g + 1 g	13g = 13g，保留	13g

例：参考电压 U_{REF}= 8V，输入电压 U_I = 5.52V

顺序	d_3 d_2 d_1 d_0	U_A(V)	比较判断	"1"留否
1	1 0 0 0	4V	$U_A < U_I$	留
2	1 1 0 0	6V	$U_A > U_I$	去
3	1 0 1 0	5V	$U_A < U_I$	留
4	1 0 1 1	5. 5V	$U_A \approx U_I$	留

由于逐次逼近式 ADC 转换过程中，需要多次比较才能得到最终结果。在比较的过程中，输入电压 Ui 需要保持不变，因此逐次逼近式 ADC 的模拟-数字转换需经过四个步骤：采样、保持、量化、编码，一般前两步由采样-保持电路完成，量化和编码由 ADC 完成。

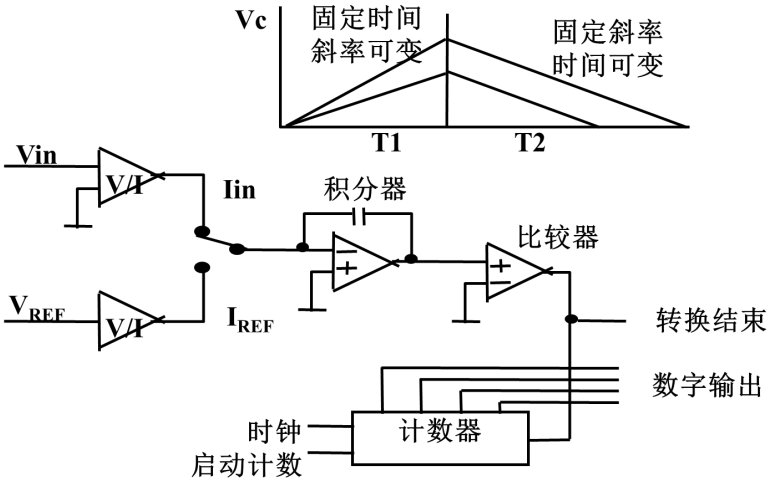
常用的采样保持电路：



3. 双积分式

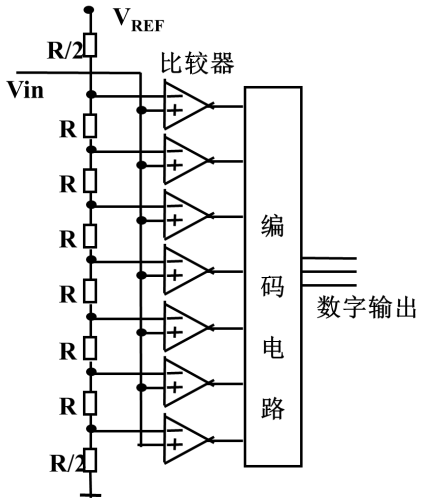
分两个积分阶段，先固定积分时间 T1，对输入信号 Vin 积分。然后对参考信号进行反向积分，计算积分到零的时间 T2。就可以根据 T2 与 T1 的关系以及 VREF 的大小，算出 Vin。

实质是电压/时间变换



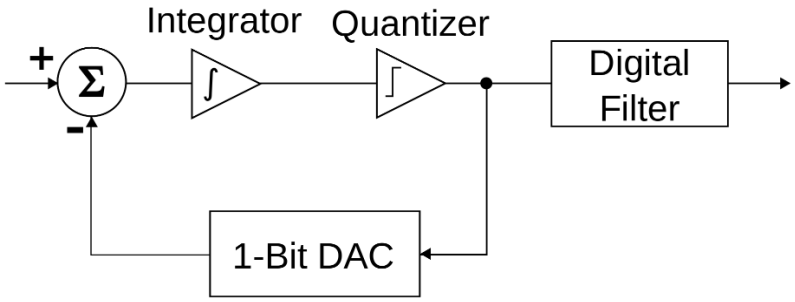
4. 并行式

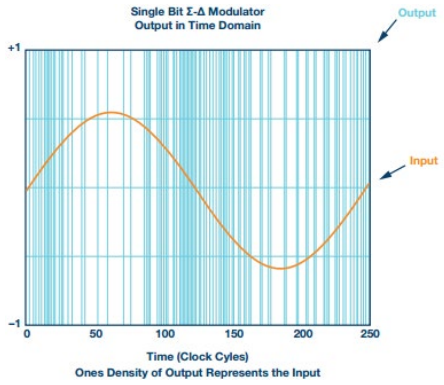
用电阻分压实现多个参考电压，然后使输入信号与每个参考电压比较。



5. Delta-sigma 型

Delta-Sigma（ΔΣ）调制（或称 Sigma-Delta（ΣΔ）调制、SDM，中文译作积分-微分调制）是一种数字模拟互相转换的方法，它是把高比特清晰度低频率信号用脉冲密度调制编码为低比特清晰度高频率信号的一种方法。





A/D 转换器的主要技术指标

1. 分辨率

以输出二进制数的位数表示分辨率。位数越多，误差越小，转换精度越高。

2. 转换速度

完成一次 A/D 转换所需要的时间，即从它接到转换控制信号起，到输出端得到稳定的数字量输出所需要的时间。

3. 精度

精度是反映转换器的实际输出接近理想输出的精确程度的物理量。实际转换值和理想特性之间的最大偏差。

4. 量化误差 (Quantizing Error)

由于 AD 的有限分辨率而引起的误差，即有限分辨率 AD 的阶梯状转移特性曲线与无限分辨率 AD（理想 AD）的转移特性曲线（直线）之间的最大偏差。通常是 1 个或半个最小数字量的模拟变化量，表示为 1LSB、1/2LSB。

5. 偏移误差(Offset Error)

输入信号为零时输出信号的值。

6. 满刻度误差(Full Scale Error)

满度输出时对应的输入信号与理想输入信号值之差。

7. 线性度(Linearity)

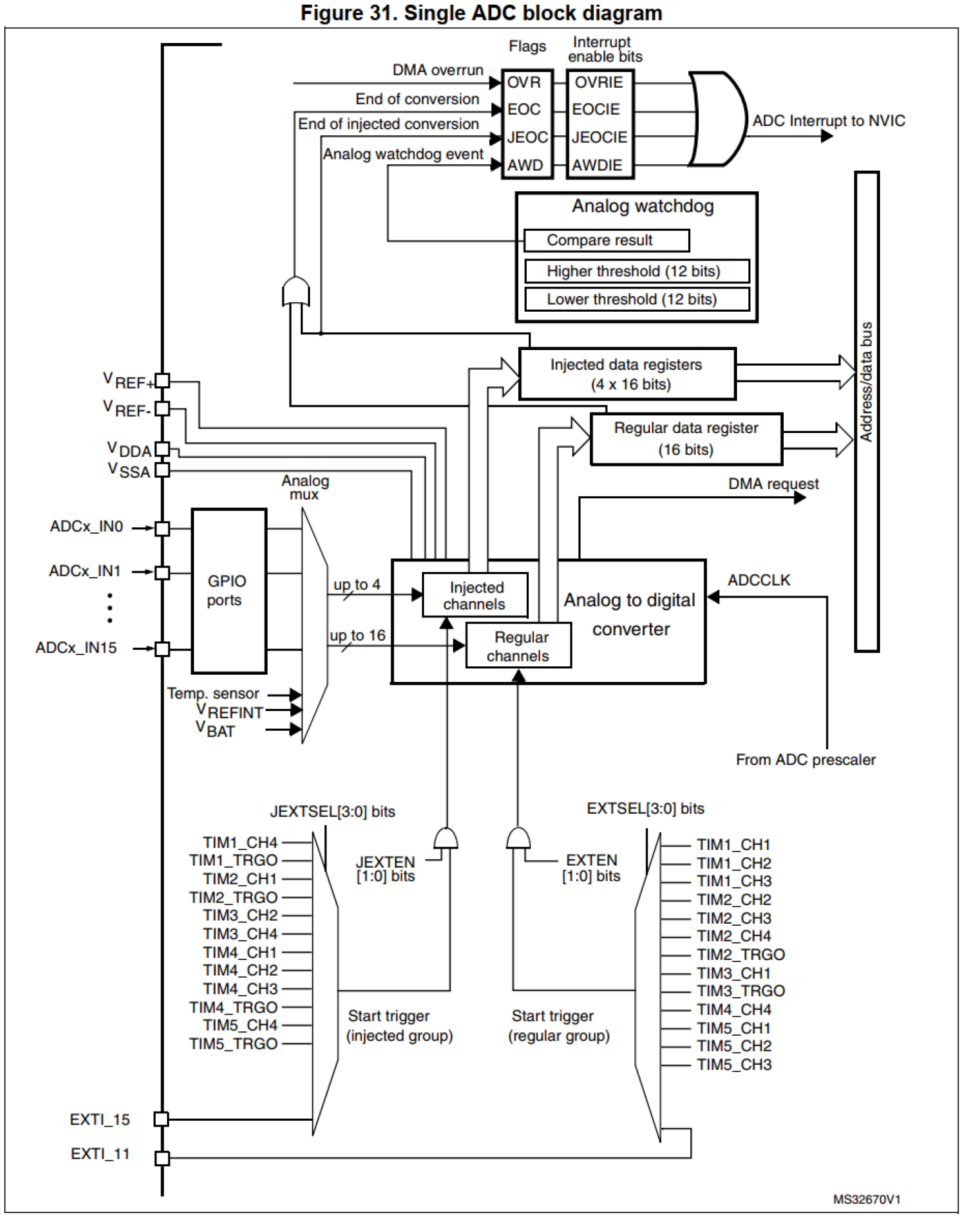
实际转换器的转移函数与理想直线的最大偏移。

5) STM32F401 的 ADC 模块及其使用方法

STM32F401 的 ADC 模块

STM32F401 微控制器有 1 个逐次逼近型 ADC：ADC1

Figure 31. Single ADC block diagram



其特性为：

- 逐次逼近型 ADC
- 可配置为 12bit、10bit、8bit 或 6bit 模式
- 单次或连续转换模式
- 在转换结束、注入转换结束以及发生模拟看门狗或溢出事件时产生中断
- 用于从通道 0 到通道“n”自动转换的扫描模式
- 数据对齐（16 位储存），以保持内置数据一致性
- 不连续模式

- ADC 供电：2.04V 至 3.6V 全速运行，1.8V 低速运行
- ADC 输入电压范围：VREF- ≤ VIN ≤ VREF+
- 16 个外部源，2 个内部源，VBAT 通道

根据 ADC 的框图可以看出：

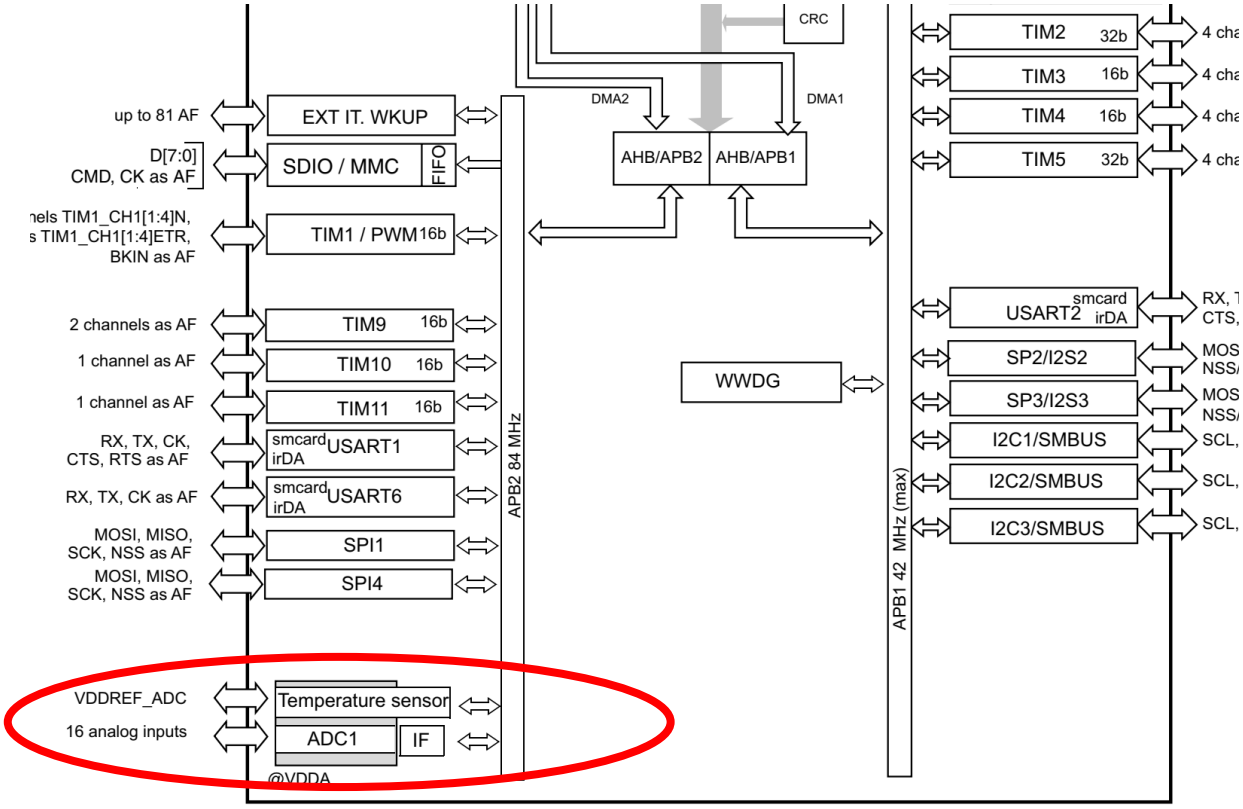
1. ADC 模块的核心是一个模拟数字转换器（analog to digital converter）。
2. MCU 的 16 个引脚，对应 ADC 的 16 个输入通道（channel）。每一时刻，可以由模拟复选器（Analog mux）选择 1 个引脚（输入通道）上的模拟信号输入到模拟数字转换器中进行转换。
3. 转换的速度由 ADCCLK 控制。
4. 转换的参考电压是 VREF- 和 VREF+。由于引脚的限制，在 STM32F401CB 这个芯片中，VREF-在内部连接到了 VSSA，而 VREF+在内部连接到了 VDDA。
5. 可以由软件启动转换，也可以由其他定时器模块的信号启动转换。
6. 转换的模式为成组转换。组即是一次需要集中转换的通道及转换的先后顺序的集合。比如，将 ADC1_IN0，ADC1_IN1，ADC1_IN2 放到一个组里面。启动该组的转换后，就可以按照 ADC1_IN0，ADC1_IN1，ADC1_IN2 的顺序来对这三个通道逐个进行转换。当然转换的顺序和数量是可以任意设置的，即也可以将 ADC1_IN6，ADC1_IN1 放到一个组里面，启动该组的转换后，就可以按照 ADC1_IN6，ADC1_IN1 的顺序来对这两个通道逐个进行转换。
7. ADC 模块中，有两个组，一个是常规组（Regular Group），还有一个是插入组（Injected Group）。一个常规组里面，可以容纳 16 个通道。一个插入组里面可以容纳 4 个通道。常规组和插入组有几个基本的区别。1：加入组的优先级高于常规组。即在常规组进行转换的过程中，如果启动了插入组的转换，则当前的常规组转换暂停，立刻开始插入组的转换。插入组转换完成后，再继续常规组的转换。2：常规组可以容纳 16 个转换通道，而插入组只能容纳 4 个。3：常规组只有一个数据寄存器，组里的每个通道转换完成后，得到的数据都会存到 Regular Data Register 中，要及时将数据读出，不然转换的数据会丢失。而插入组有 4 个数据寄存器 Injected Data Register，插入组中的每个通道转换完成后，数据会自动存到各自的数据寄存器中，读取数据比较方便。如果需要转换的通道不超过 4 个，可以只使用插入组，如果需要转换的数据多余 4 个，可以使用常规组。
8. 有两个寄存器 ADC_CR1 和 ADC_CR2 来控制 ADC 模块。有一个 ADC_SR 寄存器来指示 ADC 的状态。

ADC 模块的使用：

开发板上有一个变阻器模块，可以通过旋转旋钮改变输出电压，我们以采集变阻器模块的输出电压为例，讲解 ADC 模块的使用过程。

1. 使能 ADC 模块的时钟：

首先查看手册的 MCU 结构框图，确认 ADC1 模块挂载在 APB2 总线上，总线最高频率为 84MHz：



因此需要在 RCC_APB2ENR 寄存器中使能 ADC1 模块，即把第 8 位 ADC1 EN 设置为 1。

6.3.12 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TIM11 EN	TIM10 EN	TIM9 EN
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reser-ved	SYSCFG EN	SPI4EN	SPI1 EN	SDIO EN	Reserved		ADC1 EN	Reserved		USART6 EN	USART1 EN	Reserved			TIM1 EN
	rw	rw	rw	rw			rw			rw	rw				rw

可以使用__HAL_RCC_ADC1_CLK_ENABLE()语句，将 RCC_APB2ENR 寄存器的第 8 位 ADC1 EN 设置为 1。

2. 将 GPIO 的引脚连接到 ADC 模块的通道上

例如，我们要使用 ADC1 模块的第 14 个通道，即 ADC1_IN14。测量这个信号的电压，需要首先将 ADC1_IN14 连接到指定的引脚上。查看芯片手册：

-	-	24	33	K5		PC4	I/O	FT	-	EVENTOUT	ADC1_IN14

发现 ADC1_IN14 可以与 PC4 连接在一起。

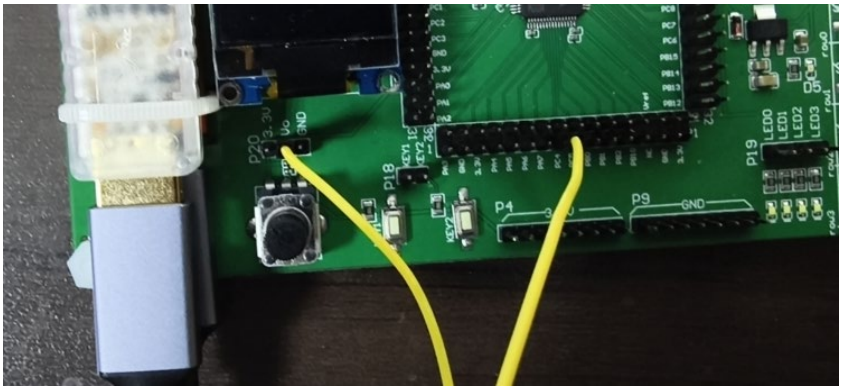
因此首先需要使能 PORTC 端口：

__HAL_RCC_GPIOC_CLK_ENABLE();

然后需要将 PC4 的输入模式改为模拟输入，即将 GPIOC 的 MODER 寄存器的第 8、9 位设置为 11.

```
temp = GPIOC->MODER;
temp &= ~(0x03 << (4 * 2));
temp |= (0x03 << (4 * 2));
GPIOC->MODER = temp;
```

这样，PC4 引脚就与 ADC 模块的 ADC1_IN14 信号连接起来了。
然后需要将开发板上变阻器模块的输出电压连接到 PC4 上。



3. 使能 ADC 模块

ADC_CR2 寄存器的第 0 位，ADON 控制 ADC 模块的电源，将这一位设置位 1，可以开启 ADC 模块。

```
ADC1->CR2|=0x01; //ADON=1
```

ADC_CR2 寄存器：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved	SWSTART	EXTEN		EXTSEL[3:0]				reserved	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON
				rw	rw	rw	rw							rw	rw

4. 设置 ADC 模块的时钟频率

在使用 ADC 模块之前，需要确认 ADC 模块最大的工作频率。根据手册可知，ADC 模块的工作频率 fADC 与 VDDA 的电压有关系，如果 VDDA 为 3.3V，那么 fADC 的最高频率为 36Mhz。

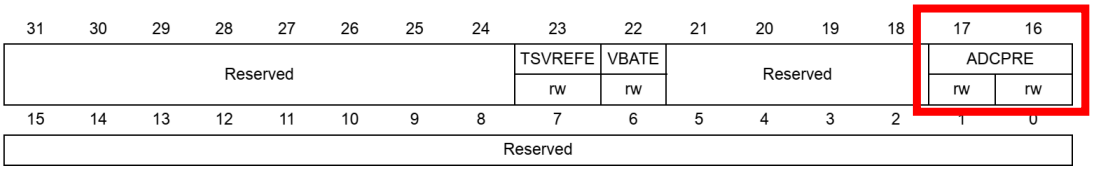
Table 66. ADC characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDA}	Power supply	$V_{DDA} - V_{REF+} < 1.2$ V	1.7(1)	-	3.6	V
V_{REF+}	Positive reference voltage		1.7(1)	-	V_{DDA}	
V_{REF-}	Negative reference voltage	-	-	0	-	
f_{ADC}	ADC clock frequency	V_{DDA} = 1.7(1) to 2.4 V	0.6	15	18	MHz
		V_{DDA} = 2.4 to 3.6 V	0.6	30	36	MHz
f_{TRIG}(2)	External trigger frequency	f_{ADC} = 30 MHz, 12-bit resolution	-	-	1764	kHz
		-	-	-	17	$1/f_{ADC}$

而 APB2 总线的时钟频率时 84MHz。因此我们需要对 APB2 总线进行分频。

ADC_CCR 寄存器中的第 16、17 位控制了 APB2 总线到 fADC 的分频系数：

ADC_CCR 寄存器



Bits 17:16 **ADCPRE:** ADC prescaler

Set and cleared by software to select the frequency of the clock to the ADC. .

Note: 00: PCLK2 divided by 2
01: PCLK2 divided by 4
10: PCLK2 divided by 6
11: PCLK2 divided by 8

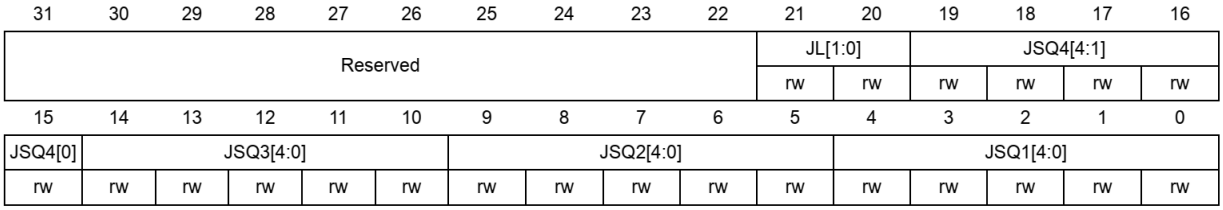
根据 fADC 的限制，我们需要将分频系数设置为 01b，即 4 分频，才能得到 21MHz 的 fADC，即将 fADC 设置在 36Mhz 以下。

```
ADC1_COMMON->CCR=(0x01<<16); //ADCPRE=0x01 /4
```

5. 将要转换的通道填入到组中

由于我们这次只转换 1 个通道，因此既可以选择使用常规组，也可以选择使用插入组。我们以使用插入组为例。控制插入组转换的寄存器是 ADC_JSQR 寄存器。

ADC_JSQR 寄存器：



这个寄存器第 20、21 两位 JL 用来控制这一组里要转换多少个通道，插入组最多转换 4 个通道。这里我们将 JL 设置为 0，值转换一个通道。

Bits 21:20 **JL[1:0]:** Injected sequence length

These bits are written by software to define the total number of conversions in the injected channel conversion sequence.
00: 1 conversion
01: 2 conversions
10: 3 conversions
11: 4 conversions

除此之外，还有四个区域，JSQ1、JSQ2、JSQ3、JSQ4 用于控制本组第 1、2、3、4 次转换的通道编号。

这里需要注意的是：

如果 JL 为 11b，即转换四个通道，则按照 SQ1、JSQ2、JSQ3、JSQ4 的顺序转换。

如果 JL 为 10b，即转换三个通道，则按照 JSQ2、JSQ3、JSQ4 的顺序转换。

如果 JL 为 01b，即转换二个通道，则按照 JSQ3、JSQ4 的顺序转换。

如果 JL 为 01b，即转换 1 个通道，则只转换 JSQ4 区域指定的通道。

由于我们本示例只转换一个通道，因此只需要设置 JSQ4 即可。JSQ4 的值既是通道的编号。因此我们将 JSQ4 设置位 14，同时将 JL 设置为 0。

```
ADC1->JSQR=((0<<20)+(14<<(3*5))); //  JSQR4=14 JL=0
```

6. 设置采样时间

ADC 的每个通道，默认有 3 个 fADC 周期的采样保持时间，用于确保本次采样的电压信号不受上一次采样信号的影响。如果要修改采样时间，需要设置 ADC_SMPR1 寄存器和 ADC_SMPR2 寄存器。

ADC_SMPR1 寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					SMP18[2:0]			SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ADC_SMPR1 寄存器中的 SMP14 区域，用于设置 ADC_IN14 通道的采样时间。

其值与采样时间的对应关系为：

Bits 26:0 **SMPx[2:0]:** Channel x sampling time selection

These bits are written by software to select the sampling time individually for each channel.
During sampling cycles, the channel selection bits must remain unchanged.

Note: 000: 3 cycles
001: 15 cycles
010: 28 cycles
011: 56 cycles
100: 84 cycles
101: 112 cycles
110: 144 cycles
111: 480 cycles

本次示例，我们将采样时间设置为 15 个 fADC 周期。

```
ADC1->SMPR1=0x01<<12;
```

7. 启动组的转换并读取转换结果

ADC_CR1 寄存器的第 8 位 SCAN，需要设置为 1，才能进行成组转换。

ADC_CR1 寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
					rw	rw	rw	rw	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	EOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

因此首先需要将 SCAN 位设置为 1：

```
ADC1->CR1|=1<<8;
```

然后就可以启动组转换了。

ADC_CR2 寄存器：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved	SWSTART	EXTEN		EXTSEL[3:0]				reserved	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON
				rw	rw	rw	rw							rw	rw

将 ADC_CR2 寄存器的第 22 位，即 JSWSTART 设置为 1，就可以开始插入组的转换。

```
ADC1->CR2|=1<<22;
```

开始转换后，可以通过 ADC_SR 寄存器查看 ADC 模块的状态

ADC_SR 寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										OVR	STRT	JSTRT	JEOC	EOC	AWD
										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

ADC_SR 寄存器的第 2 位 JEOC，指示插入组是否转换完成。如果 JEOC 为 1，表示插入组的转换已经完成。因此在转换前要先清除 JEOC：

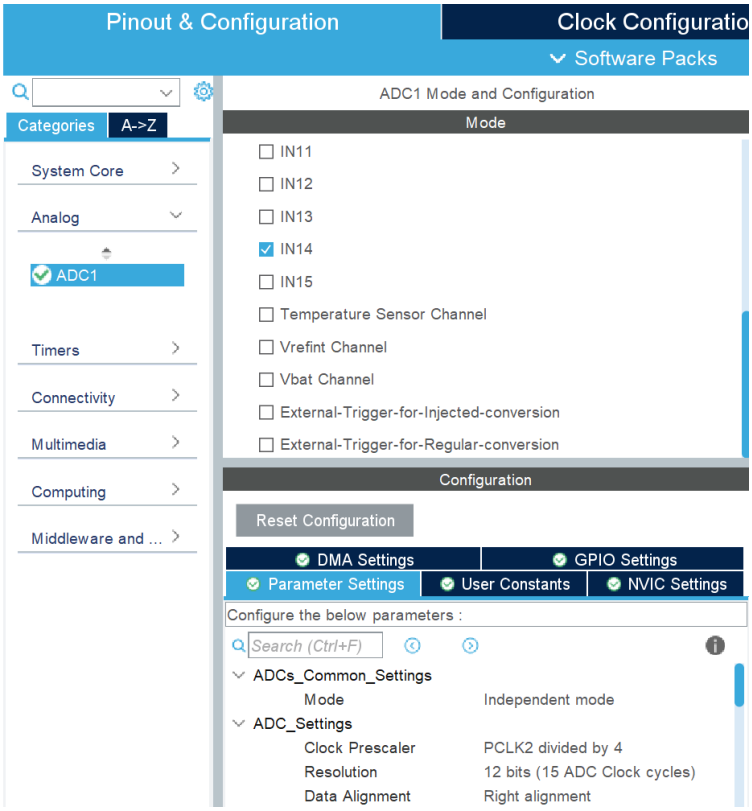
```
ADC1->SR&=~(0x04);//JEOC=0;
```

启动 ADC 转换后，再查看 JEOC，如果 JEOC 为 1，就从插入组数据寄存器中读取转换值，插入组第一个转换通道的转换结果存在 JDR1 寄存器中。同理第 2、3、4 个通道的转换结果存在 JDR2、JDR3、JDR4 寄存器中。这里需要注意的是，虽然只采样一个通道的情况下，转换的通道是由 ADC_JSQR 寄存器的 JSQ4 区域指定的，但是转换结果存在 JDR1 寄存器中。

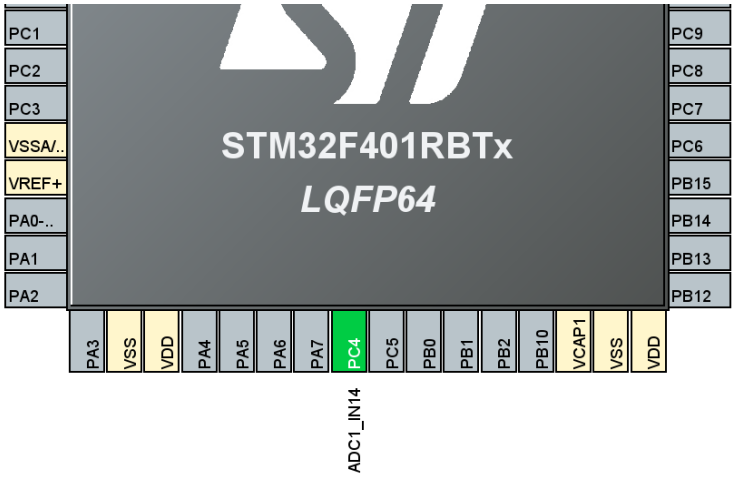
```
while((ADC1->SR & 0x04)==0);


ADCdata14=ADC1->JDR1;
```

使用 HAL 库操作 ADC 模块：

首先复制 HelloWorld 工程，修改相关配置文件。在左侧 Analog 条目中选择 ADC1 模块。在右边上方的 Mode 选项中，勾选 IN14，即 ADC_IN14 通道，如图所示。



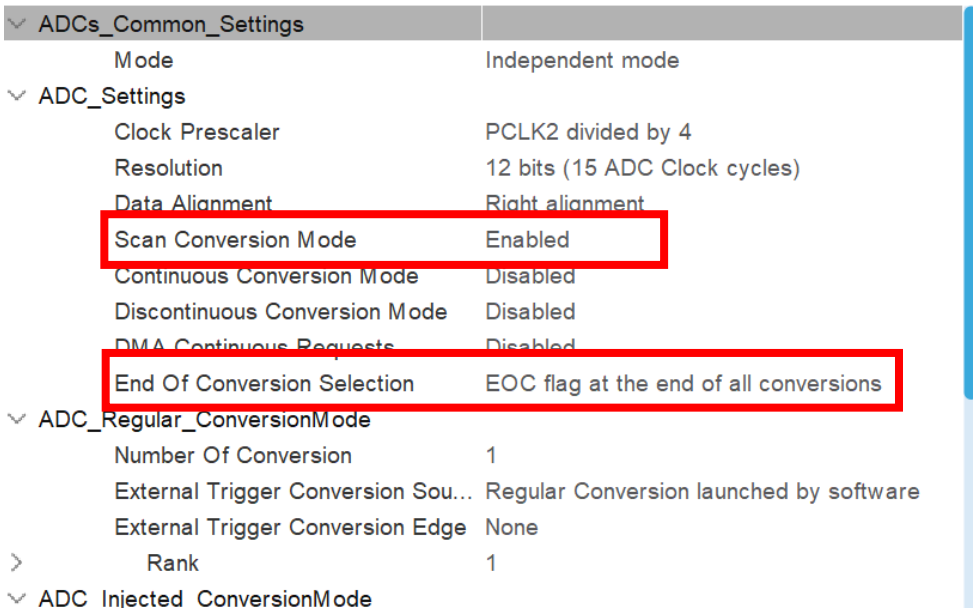
此时，PC4 引脚会变成绿色，其功能会自动被设置为 ADC_IN14：



然后对 ADC 进行设置;

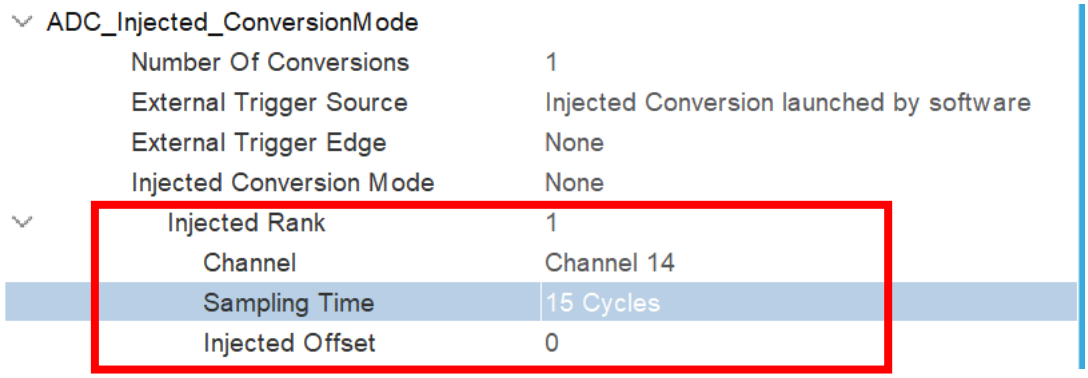
转到 ADC1 模块的 Parameter Settings 选项卡中，将 Scan Conversion Mode 设置为 Enabled。

将 End Of Conversion Selection 改成 EOC flag at the end of all conversions。



由于我们没有使用常规组，所以 ADC_Regular_conversionMode 中的设置保持不变。

展开 ADC Injected Conversion Mode 菜单，将 Number Of Conversions 设置为 1，Injected Rank 1 中的 Channel 设置为 Channel 14。Sampling Time 设置为 15 Cycles。



设置完毕后，保存并生成代码。

生成代码后，main.c 中会多了一个 MX_ADC1_Init 函数，用于设置 ADC 模块和采样通道。并同时生成一个全局变量 hadc1 用于操作 ADC1 模块。

调用 HAL_ADCEx_InjectedStart(&hadc1);语句可以启动插入组的转换。

调用 HAL_ADCEx_InjectedPollForConversion 函数可以不断的查询 ADC_SR 寄存器，查询转换是否完成，在设定的时间（第二个参数）内完成了转换，该函数返回 HAL_OK。

然后调用 HAL_ADCEx_InjectedGetValue 函数可以读取插入通道的转换结果，其第二个参数可以选择组内第几个通道，使用方法如下：

```
HAL_ADCEx_InjectedStart(&hadc1);

if(HAL_ADCEx_InjectedPollForConversion(&hadc1,2)==HAL_OK){
    adcdata14=HAL_ADCEx_InjectedGetValue(&hadc1,1);
}
```

ADC 中断：

除了使用轮询的方法查询 ADC 是否转换完成外，也可以使用中断的方式获取 ADC 的转换结果。

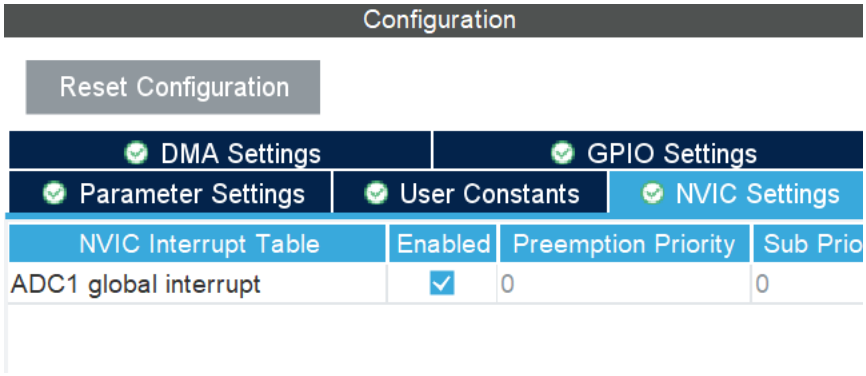
ADC_CR1 寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
					rw	rw	rw	rw	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

在 ADC_CR1 寄存器中，如果将第 7 位 JEOCIE 设置为 1，则插入组转换完成后，ADC 模块会发送中断请求给 NVIC。响应此中断，可以第一时间获取 ADC 的转换结果。

在上面工程的基础上，在 ADC1 模块的配置界面中，点击 NVIC Settings 选项卡，勾选 ADC1 global interrupt。



保存并生成代码。

生成代码后，在 stm32f4xx_it.c 文件中，会出现 ADC1 的中断服务函数，用于处理 ADC1 的中断事务。

```
void ADC_IRQHandler(void)
{
  /* USER CODE BEGIN ADC_IRQn 0 */

  /* USER CODE END ADC_IRQn 0 */
  HAL_ADC_IRQHandler(&hadc1);
  /* USER CODE BEGIN ADC_IRQn 1 */

  /* USER CODE END ADC_IRQn 1 */
}
```

这时，在 main 函数中，调用 HAL_ADCEx_InjectedStart_IT(&hadc1);函数可以启动 ADC1 的插入组转换，并在转换完成后产生中断，进入中断服务函数。在中断服务函数中，会调用弱函数 HAL_ADCEx_InjectedConvCpltCallback 用于处理转换数据。

因此在 main.c 中重新定义 HAL_ADCEx_InjectedConvCpltCallback 函数，获取转换数据：

```
/* USER CODE BEGIN 0 */
void HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef* hadc){
   adcdata14=HAL_ADCEx_InjectedGetValue(hadc,1);
```

```
}
/* USER CODE END 0 */
```