本文档为 TI 数字温度传感器 TMP104 的应用入门，介绍如何利用 MSP430 及 TMP104 来进行温度的读取及系统设计过程中的注意事项。

文中涉及源代码仅供参考，如需完整代码，请邮件索取

准备文档

TMP104 Datasheet http://www.ti.com/lit/ds/symlink/tmp104.pdf

TMP104 评估板用户手册 http://www.ti.com/lit/ug/sbou118/sbou118.pdf

MSP430x2xx User's Guide http://www.ti.com/lit/ug/slau144j/slau144j.pdf

软件

CCS http://www.ti.com/tool/CCSTUDIO-MSP

登陆 TI Store 免费申请评估板所需芯片 https://store.ti.com/default.aspx

TMP104 是 TI 推出的的低功耗数字温度传感器。其分辨率为 1 摄氏度，精度为±0.5 摄氏度。供电电压范围为 1.4V – 3.6V。典型应用场景为手机及笔记本电脑。TMP104 提供支持菊花链的 SMAART 接口，目前公开的 TMP104 版本支持 16 个设备相连。同时，TMP104 通讯支持多设备访问（Multiple device access），也就意味着主机可以单独对链路上的每个 TMP104 进行控制访问。

如图所示为 TMP104 的评估板，采用 MSP430F2112 作为控制芯片，链路上共有 4 个 TMP104，PCB 丝印标注为 TMP104A，TMP104B，TMP104C 和 TMP104D。可以看出 TMP104 的芯片封装很小，为 4 引脚 WCSP（DSBGA）封装。

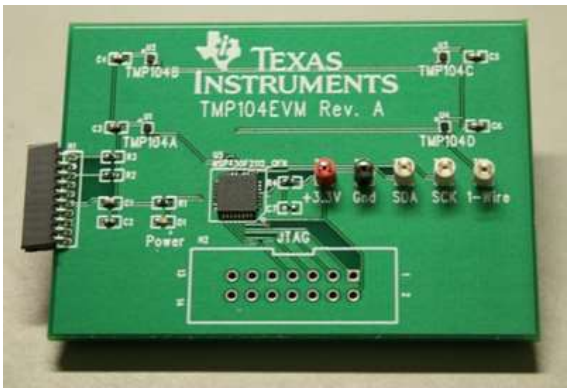

Figure 1 TMP104 评估板

硬件设计

TMP104 只有 4 个引脚，分别为用于通信的 RX，TX 以及供电 VCC 和 GND。其中通信引脚按照图示方法与主机相连。这种单线串行通信方式为 SMAART 通信，其通信协议在后文中进行阐述。

VCC 和 GND 之间最短回路加上旁路电容，容值应大于 0.1uF，以达到进一步降低噪声的目的。

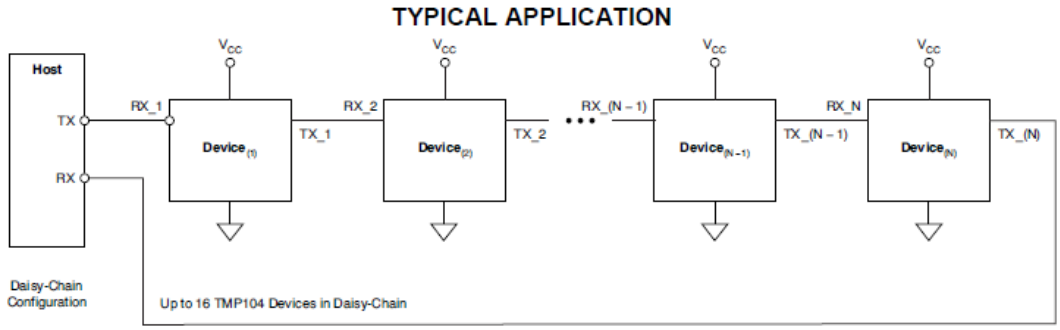

Figure 2 硬件连接

注意事项：TMP104 芯片本身即为温度传感器，其检测的外界温度通过焊锡盘以及封装感知，所以在进行电路设计时，需要特别注意热传导回路，以确保待测点的温度能够尽可能小损耗地到达芯片内部，同时减少非测量点温度对温度传感器的影响。

软件设计

通过硬件 UART 对菊花链中的 TMP104 进行控制和温度的读取。首先来看一线 UART，即 SMAART 通信协议。

Table 1. Communication Format

Start bit	Calibration	Stop bit	Start bit	Command byte	Stop bit	Start bit	Data byte	Stop bit

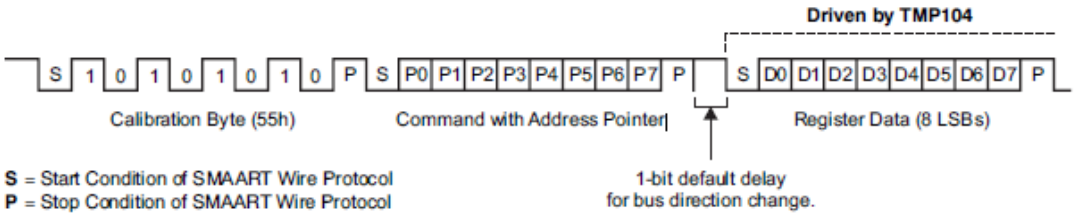

Figure 3 SMAART 通信协议

从图中看到 SMAART 通信协议的基础是 UART，但略有不同。

每次通讯的开始，由主机首先发送一段校准字节段来对通讯波特率进行自动同步，校准字节为 0x55h（1010,1010）。和串口通讯相同，在每个字节传输的开始和结束都分别有一个 bit 来标志开始和结束，同样，开始位为低电平，结束位为高电平。

紧接着，由主机开始发送指令字节。在指令字节发送结束位之后，总线会有一个 bit 的延迟，用于总线方向调整。根据发送的指令字节类型，对应的 TMP104 会响应命令，或写入寄存器，或读取寄存器中的数值。无论写入还是读取，仍遵循之前的通讯协议。

一次通讯中，波特率必须保持一致，但上一次通信和下一次通信的波特率可以发生改变。SMAART 通信协议支持的波特率范围在 4.8k 至 114k 之间。对抖动的容忍度在±1%。

TMP104 寄存器

TMP104 包括四个读写寄存器，分别为温度寄存器，配置寄存器，低温阈值寄存器和高温阈值寄存器。可以通过向指令寄存器中写入不同的指令对这四个寄存器进行访问和控制。

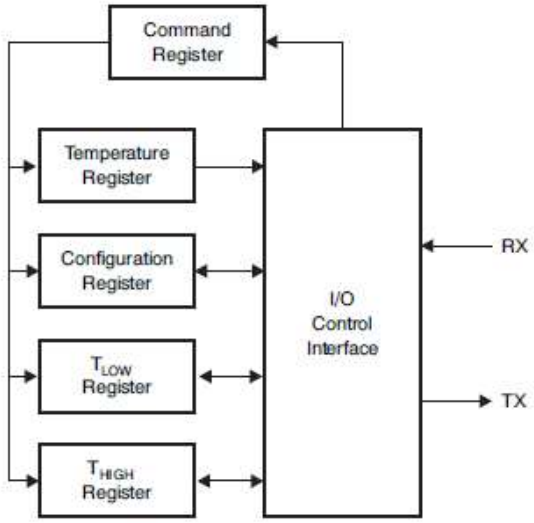

Figure 4 TMP104 内部寄存器

分别看这几个寄存器：

1. 温度寄存器

温度寄存器为一个 8 位只读寄存器，存储最近一次转换的芯片传感器温度数字值。温度与数字量的对应关系如图所示。

TEMPERATURE (°C)	DIGITAL OUTPUT	
	BINARY	HEX
128	0111 1111	7F
127	0111 1111	7F
100	0110 0100	64
80	0101 0000	50
75	0100 1011	4B
50	0011 0010	32
25	0001 1001	19
0	0000 0000	00
–1	1111 1111	FF
–25	1110 0111	E7
–55	1100 1001	C9

Figure 5 温度寄存器数据格式

温度的模拟值和数字量是一一对应的，例如 100℃，将 100 转换为二进制为 0110,0100，该值即为寄存器中读取到的数值。需要注意的是对于零下温度，采用的转换方式遵循以下

法则：例如零下 55℃（-55℃），55 对应的二进制为 0011,0111，在转换为负数时，首先将所有位取反，即 1100,1000，然后加 1，最终为 1100,1001。

由此也可以理解为什么说 TMP104 的分辨率为 1 摄氏度。

2. 配置寄存器

配置寄存器为一个 8 位的可读写寄存器，通过该寄存器可完成对温度传感器的相关配置。下表中第三行为上电时传感器寄存器中的默认值。

D7	D6	D5	D4	D3	D2	D1	D0
INT_EN	**CR1**	**CR0**	**FH**	**FL**	**LC**	**M1**	**M0**
0	0	0	0	0	0	1	0

- **INT_EN**

中断使能位，每个 TMP104 都可以通过断开与总线的连接，向主机提示中断。TMP104 拉低总线提示中断必须满足以下几个条件：

- 寄存器该位，即中断使能标志位被置为“1”；
- 温度传感器检测到的温度值不在低温阈值寄存器和高温阈值寄存器所限制的范围内；
- 总线超时（处于高电平超过 28ms）。

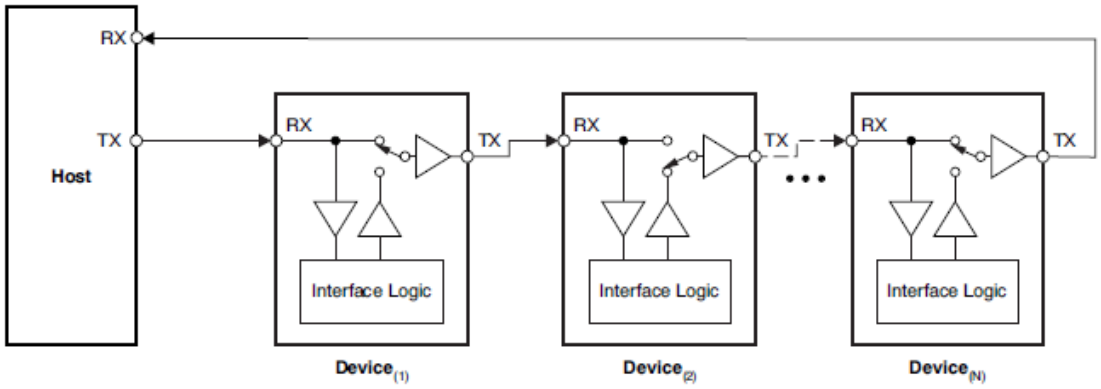

Figure 6 设备 2 通过断开总线提示中断

以下操作之一可以使设备从中断中恢复：

- 全局中断清除指令；
- 全局软件复位指令；
- 上电复位。

在使能中断情况下，有可能发生主机正在发送指令的同时，TMP104 产生中断。为避免此种情况的发生，最好在发送校准字节后，主机确认总线是否为低电平

- **CR1, CR0**

转换率位。通过配置寄存器中的 CR1 和 CR0 两位，可以设置 TMP104 的转换速率。默认上电时，TMP104 会以 4s 一次的速率更新温度寄存器内的数值。通过配置，可设置转换速率为 1Hz，4Hz 或 8Hz。

CR1	CR0	CONVERSION RATE
0	0	0.25 Hz (default)
0	1	1 Hz
1	0	4 Hz
1	1	8 Hz

注意：上述的转换率是指 TMP104 工作在连续模式下的速度。实际上温度传感器在每次转换的时序为下图所示，其时间消耗为：启动时间需占据 26ms，在第一次转换和第二次转换之间会有一个延迟。当在连续模式下，这个延迟会由 CR0 和 CR1 两个寄存器内的配置值决定。而如果工作在非连续模式下，如后文提及的单次转换模式，单次转换所需的最小时间为：26ms 加上 300us（单次读取所耗费的最短时间），由此可推算得到每秒可以实现的转换次数可达 30 多次。

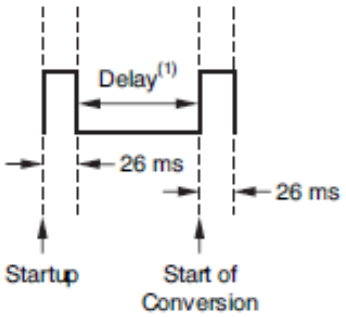

Figure 7 TMP104 温度转换时序

- **FH, FL 温度看门狗**

该两位为温度阈值看门狗标志位。当温度寄存器中的数值高于高温阈值寄存器中的数值时，FH（Flag High）位会置“1”，如果温度寄存器中的数值低于低温阈值寄存器中的数值时，FL（Flag Low）位会置“1”。通过这两位可以判断待测温度是否在阈值中。

- **LC**

ElecMuse

缓存位（Latch Bit），LC 位用于缓存 FH 和 FL 的数值。

- **M1, M0**

转换模式配置位。这两位一起实现了转换模式的配置，在 TMP104 中，共可工作在三种模式下：一种是掉电模式，此时配置 M1 和 M0 为“0”，在掉电模式下，TMP104 可以工作在最小电流消耗下，典型电流消耗仅 0.5uA；一种是单次转换模式，当 TMP104 处在掉电模式下，向 M1 和 M0 写入“01”将触发单次温度转换，在结束转换后，M1 和 M0 的值会自动切换到“00”，重新会到掉电模式。如前文所述，采用该种方式进行温度转换读取，每秒最多可进行 30 余次的温度转换；最后一种模式是连续转换模式，这也是 TMP104 上电默认的模式，此时寄存器 M1 位为“1”。

3. 温度阈值寄存器

在高温阈值寄存器和低温阈值寄存器中分别存放高低温阈值。在每次温度转换结束后，TMP104 会将转换得到的温度与温度阈值寄存器中的阈值进行比较，若测得的温度值高于高温阈值，配置寄存器中的 FH 位会置“1”，同样，若测得的温度低于低温阈值，配置寄存器中的 FL 位会置“1”，从而进一步触发中断。

默认上电时，高温阈值寄存器中的值为+60℃，低温阈值寄存器的值为-10℃。

4. 指令寄存器

P7	P6	P5	P4	P3	P2	P1	P0
GLB	IN3/ID3	IN2/ID2	IN1/ID1	IN0/ID0	P1	P0	R/W

其中，

P7 标识该指令为全局指令还是针对总线上的某个设备。P7 置“0”则为针对单个设备的指令，若 P7 置“1”，则为全局指令，针对总线上所有的设备。

全局地址分配（P7-P0 = 1000,1100）

在初始化阶段，主机需发送 1000,1100 指令，对总线上的设备进行地址分配。紧接着，主机发送地址分配指令（1001,0000），其中 P3-P0 为首设备的地址。总线上的设备依次被分配地址（逐次+1），最后反馈至主机为最后一个设备的地址+1。通过这个过程，主机就可以分辨总线上有多少设备。

全局读/写

主机可以对总线上所有的 TMP104 进行读或写操作，全局读写指令格式为

P7	P6	P5	P4	P3	P2	P1	P0

1	1	1	1	0	X	X	X

其中 P2 和 P1 决定读/写的寄存器类型

P2	P1	寄存器
0	0	温度寄存器（仅读）
0	1	配置寄存器
1	0	低温阈值寄存器
1	1	高温阈值寄存器

P0 标识随后总线方向是读或写，若 P0 为“0”，则为写指令，若 P0 为“1”，则为读指令。

全局中断清除指令（1010,1001）；全局软件复位（1011,0100）

CCS 例程解析

在 TMP104EVM 中，使用 MSP430F2112 作为上位机控制 4 个 TMP104.

基本时钟模块配置

在 EVM 板中 MSP430 未外接晶振，故使用内部 DCO 及 VLO 作为时钟源。DCO 配置在 1MHz，提供 MCLK 和 SMCLK。

```
WDTCTL = WDTPW + WDTHOLD;              // Stop WDT

if (CALBC1_1MHZ==0xFF)                // If calibration constant erased
  {
    while(1);                          // do not load, trap CPU!!
  }
DCOCTL = 0;                            // Select lowest DCOx and MODx settings
BCSCTL1 = CALBC1_1MHZ;                 // Set DCO
DCOCTL = CALDCO 1MHZ;
```

UART 模块配置

USCI_A0 模块，对应引脚为 P3.4 和 P3.5。使用 SMCLK 作为时钟信号，配置波特率为 9.6k

```
P3SEL = 0x30;                             // P3.4,5 = USCI_A0 TXD/RXD
UCA0CTL1 |= UCSSEL_2;                     // SMCLK
UCA0BR0 = 104;                            // 1MHz 9600
UCA0BR1 = 0;                              // 1MHz 9600
UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 1
UCA0CTL1 &= ~UCSWRST;
```

TMP104 初始化

根据数据手册描述，在开始使用 TMP104 之前应对链路上的设备进行初始化。注意，这里定义了 echodata 数组用于存储传输线短接情况下直接回传至 MSP430RX 端口的数据。作者不确定是否一定需要这样的操作，如果有更合理的解释和操作方式，欢迎提出宝贵意见。

经过初始化后，在 RX 端会接收到 0x94 的数据，这个数据为链路中最后一个设备，即第四个设备的地址+1（初始设备为 0x90）。

```
char GlobalInit()
{
      while (!(IFG2&UCA0TXIFG));
      UCA0TXBUF = 0x55;

       while (!(IFG2&UCA0TXIFG));
       UCA0TXBUF = 0x8C;

       while (!(IFG2&UCA0TXIFG));
       UCA0TXBUF = 0x90;

       while (!(IFG2&UCA0RXIFG));
       echodata[0] = UCA0RXBUF;

       while (!(IFG2&UCA0RXIFG));
       echodata[1] = UCA0RXBUF;

        while (!(IFG2&UCA0RXIFG));
       return UCA0RXBUF;

}
```

以下将简单演示如何读取设备中的温度信息。一种是单个设备读取。

根据指令寄存器的数据格式，可以给出对链路中第一个设备，设备编号为 0，进行温度寄存器数据的读取，其指令应为 0x01。

```
/* dev 1*/
Tempdev[0] = ReadFromTmp (0x01);
```

定义单个读取子函数，根据通讯协议，首先同步波特率，然后写入指令，最后从 RXBUF 中读取对应的数值，即为当前传感器检测到的摄氏温度值。和初始化一样，会接收到两个 echodata，同上处理。重复该步骤即可得到链路上所有设备的温度值。

```
……

char ReadFromTmp(char RRegAddr)
{

       while (!(IFG2&UCA0TXIFG));
         UCA0TXBUF = 0x55;

         while (!(IFG2&UCA0TXIFG));
         UCA0TXBUF = RRegAddr;

         while (!(IFG2&UCA0RXIFG));
         echodata[0] = UCA0RXBUF;

         while (!(IFG2&UCA0RXIFG));
         echodata[1] = UCA0RXBUF;

         while (!(IFG2&UCA0RXIFG));
         return UCA0RXBUF;


}
```

类似的，可以对所有设备进行一次性温度读取，只需更改指令为 0xF1 即可。

```
void GlobalTempRead(char DevNum)
{
       while (!(IFG2&UCA0TXIFG));
       UCA0TXBUF = 0x55;

       while (!(IFG2&UCA0TXIFG));
       UCA0TXBUF =0xF1;

       while (!(IFG2&UCA0RXIFG));
       echodata[0] = UCA0RXBUF;

       while (!(IFG2&UCA0RXIFG));
       echodata[1] = UCA0RXBUF;

       for (i=0;i<DevNum;i++)
         {
                while (!(IFG2&UCA0RXIFG));
                GlobalTemp [i] = UCA0RXBUF;
         }

}
```

- 欢迎发送邮件 regina-cui@ti.com 或 cuimeng1919@163.com 讨论交流
- 登陆 TI Store 免费申请评估板所需芯片 https://store.ti.com/default.aspx

附录

MSP430F2 系列时钟模块简介

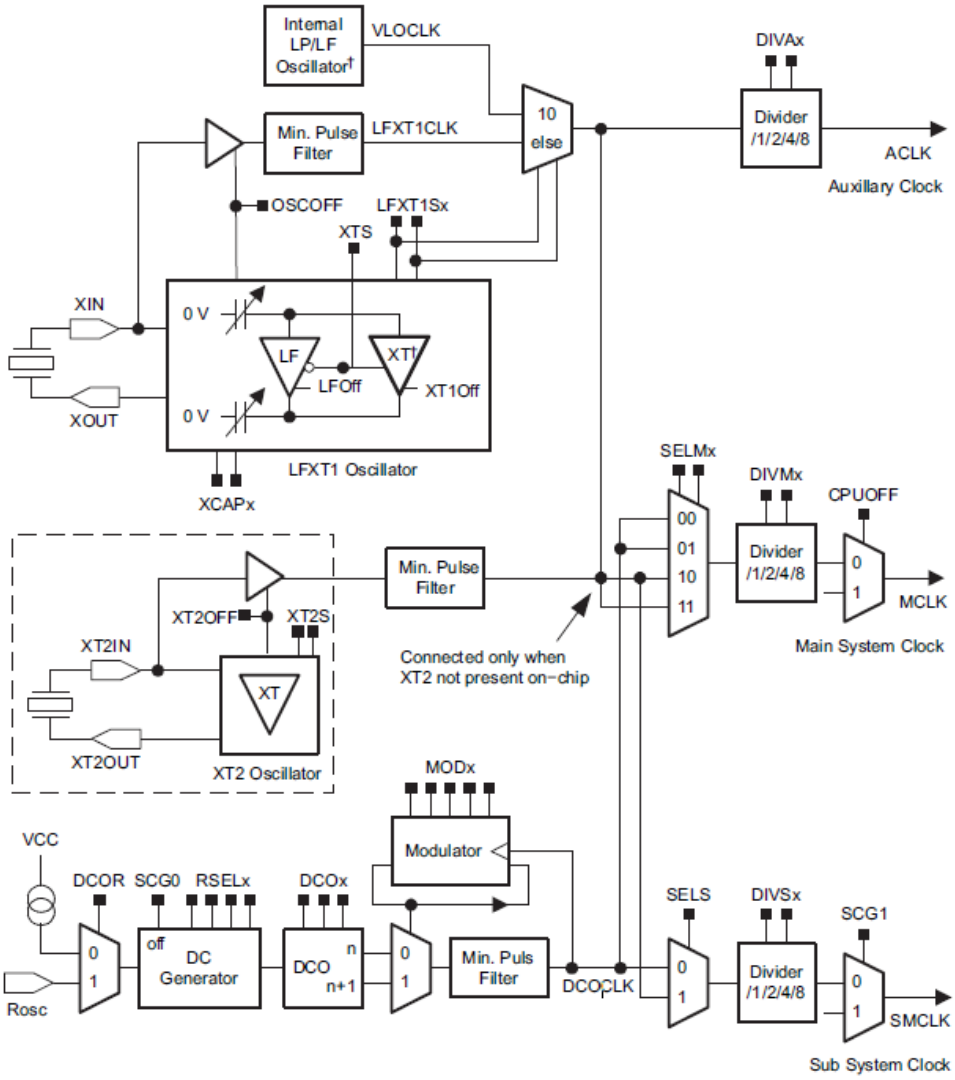

Figure 5-1. Basic Clock Module+ Block Diagram – MSP430F2xx

Figure 8 F2112 时钟模块

ElecMuse

如图所示，可以看到 F2112 的基本时钟模块框图，大多数 MSP430 家族的时钟模块也与之相类似。

首先看输出，基本时钟模块提供到 CPU 及外设的有三种时钟信号，在图中从上至下分别为 ACLK，MCLK 和 SMCLK。其中 MCLK 为 CPU 时钟，SMCLK 为高速外设时钟，而 ACLK 为低速外设时钟（这点所有的 MSP430 均适用）。通俗的来讲，在后面使用到的各个外设，编程者都需要为这些数字外设选择时钟信号，在每个模块都有相应的寄存器控制字来进行时钟源的选择。

接下来看这三种时钟信号的时钟源。同样在图中从上至下可以看到有四种不同的时钟源，分别为 VLOCLK，LFXT1, XT2 和 DCO。其中 VLO 和 DCO 为内部时钟，VLO 的典型频率为 12kHz。LFXT1 和 XT2（某些型号有）支持外部晶振。

注意：ACLK 时钟信号的时钟源仅支持 LFXT1 和 VCOCLK。

MSP430F2 系列在上电时初始的时钟配置如下所述：MCLK 和 SMCLK 时钟信号来源于 DCOCLK（大约为 1.1MHz），ACLK 来源于 LFXT1CLK，其配置为 LF 模式，内部电容为 6pF。

看上去对初学者略为复杂的时钟模块其实主要在功耗上让用户更多地受益。低频时钟能够最大限度地降低系统功耗，而高频时钟则能满足事件快速响应需求。在 MSP430 基本时钟系统中，ACLK 可以选择 32k 手表时钟，从而提供稳定同时低功耗的时钟源；MCLK 选择芯片内部的 DCO；SMCLK 则可以根据需要选择外部时钟或 DCO。