Application Note

TI 湿度传感器：编程和集成指南

Harry Gill

摘要

相对湿度 (RH) 传感器在当今的电子系统中发挥着关键作用，可在各种应用中支持精确的环境监测——从服务器机房和工业自动化到电动汽车和智能基础设施。通过提供环境条件下的实时数据，这些传感器有助于保护敏感系统、提高性能并保持整体可靠性。本应用手册为在以下三代产品选择和集成德州仪器 (TI) 湿度传感器提供了实用指导：HDC1x（第一代）、HDC2x（第二代）和 HDC3x（第三代，也是最新一代）。虽然每个器件系列都提供温度和湿度测量功能，但每个系列都具有独特的接口协议和配置选项。本应用手册中提供的内容旨在简化针对给定系统设计评估和实现 TI 湿度传感器的过程。

备注

除非另有说明，否则本应用手册引用的*数字接口/协议*严格限定于基于 I2C 的数字通信。本应用手册将重点介绍使用 Arduino™ 平台的代码示例——一个基于 C 语言的开源原型设计的理想平台，它简单并且可快速实现。可通过本文档末尾提供的链接获取其他 C 代码。

另外，在 HDC302x 或 HDC2x 等器件名称末尾使用字母“x”表示以下说明适用于“x”之后的所有适用型号：

HDC1x = HDC1010 / HDC1080

HDC2x = HDC2010 / HDC2021 / HDC2022 / HDC2080

HDC3x =HDC3020 / HDC3021 / HDC3022 / HDC3120

内容

1 简介.....2
2 数字 I2C 接口概述.....3
2.1 寄存器映射协议.....3
2.1.1 I2C 寄存器映射协议的快速概览.....3
2.1.1.1 HDC1x.....4
2.1.1.2 HDC2x.....6
2.2 命令协议.....12
2.2.1 HDC302x.....12
2.2.1.1 按需触发模式下的连接（单次触发）.....14
2.2.1.2 自动测量模式 (AMM) 下连接.....15
如何使用 CRC 校验测量数据.....16
3 模拟接口概述.....18
3.1 HDC3120.....18
4 总结.....24
5 开发支持和文档.....24
5.1 软件支持.....24
5.2 参考资料.....24

商标

1 简介

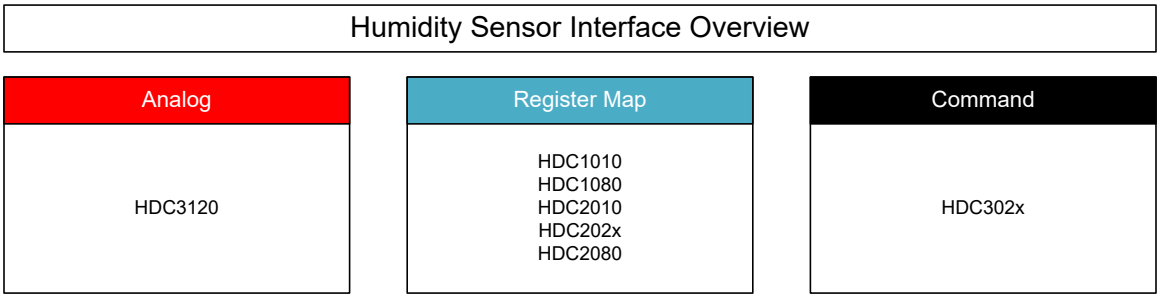

图 1-1. 湿度传感器接口概述

德州仪器 (TI) 提供具有两种主要接口类型的湿度传感器产品系列：模拟比例式和数字 I2C 通信。这些接口类型可分为三组：

1. **数字 I2C – 寄存器映射协议**
   - 通信基于寄存器（类似于 TI 的温度传感器）。
   - 通过对特定寄存器地址进行写入和读取来触发和读取测量值。
   - 器件：HDC1x 和 HDC2x（例如 HDC1080、HDC2022）
2. **数字 I2C – 基于命令的协议**
   - 通信基于命令序列。
   - 主机发送一条命令来开始测量，然后使用单独的读取命令检索结果。
   - 器件：HDC302x（例如，HDC3020、HDC3022）
3. **模拟输出 – 比例式电压**
   - 传感器输出与温度和湿度成比例的电压信号。
   - 这些信号可以直接馈送到模拟系统中，也可以使用外部 ADC 进行数字化。
   - 器件：HDC3120

每种接口类型都需要不同的方法来读取传感器数据。以下各节将详细介绍这些方法，并提供支持 Arduino 的微控制器的代码示例。

2 数字 I2C 接口概述

TI 目前提供两种 I2C 接口样式的数字湿度传感器：寄存器映射或基于命令的访问。以下各节介绍了对每个数字湿度传感器系列的传感器进行编程的一般过程。

2.1 寄存器映射协议

HDC1x 和 HDC2x 传感器系列都使用基于寄存器映射的数字接口，其中通过写入特定的寄存器来开始温度和湿度测量和读取测量值。

虽然通信概念相似，但两个系列在功能和配置选项上有所不同。

HDC2x 系列提供更高级的功能集，包括：

- 警报功能
- 数据就绪或中断引脚支持
- 两种测量模式：
  - 按需触发，实现用户控制的实时采样
  - 自动测量模式 (AMM)，实现低功耗周期性采样
- 拆分了温度 MSB/LSB（温度高或低）和湿度 MSB/LSB（湿度高或低）的寄存器

HDC1x 系列旨在实现更简单的实施。这仅支持按需触发，并使用基本寄存器结构，每个结构有一个寄存器用于温度和湿度测量。

总之，对于需要高级功能、可配置测量模式或低功耗工作的设计，建议使用 HDC2x 系列而不是 HDC1x。HDC1x 系列专为需要简单直接的数字 I2C 接口和更简单编码要求的应用而设计。

2.1.1 I2C 寄存器映射协议的快速概览

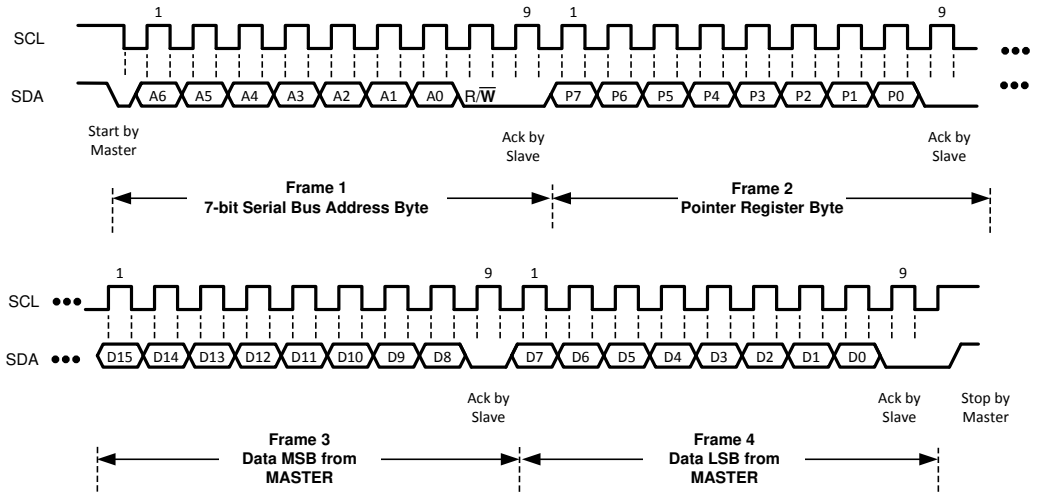

图 2-1. HDC1080 数据帧示例（配置寄存器）

在基于 I2C 的传感器中，寄存器映射是存储器位置（寄存器）的结构化表，用于定义如何控制和访问器件。每个寄存器都有一个主机可以读取或写入的唯一地址，从而能够与传感器的配置、状态位和数据进行直接交互。这种有序的布局使调整设置、检索测量值和监控标志位变得简单。对于 HDC1x 和 HDC2x 器件，数据结构类似于 图 2-1 中的结构，其中发送地址字节后，控制器必须先发送指针寄存器字节，然后才能从传感器读取数据。

有关 I2C 的更多信息，请参阅以下关于 I2C 基本知识的文档。

2.1.1.1 HDC1x

TI 的第一代 HDC1x 器件都共享同一个寄存器映射（如 表 2-1 中所示）。因此，以下说明适用于 HDC1x 系列。

表 2-1. HDC1x 寄存器映射

指针	名称	复位值	说明
0x00	温度	0x0000	温度测量输出
0x01	湿度	0x0000	相对湿度测量输出
0x02	配置	0x1000	HDC1080 配置和状态
0xFB	串行 ID	视器件而定	器件串行 ID 的前 2 个字节
0xFC	串行 ID	视器件而定	器件串行 ID 的中间 2 个字节
0xFD	串行 ID	视器件而定	器件串行 ID 的最后一个字节位
0xFE	制造商 ID	0x5449	德州仪器 (TI) 的 ID
0xFF	器件 ID	0x1050	器件的 ID

首先，用户必须向 表 2-2 (0x02) 中的寄存器写入一个 16 位值，以便按顺序定义测量序列——温度和/或湿度。

表 2-2. HDC1x 配置寄存器 (0x02)

名称	位			说明
RST	[15]	软件复位位	0	正常运行，该位会自行清除
			1	软件复位
保留	[14]	保留	0	保留，必须为 0
加热	[13]	加热器	0	加热器已禁用
			1	加热器已使能
模式	[12]	采集模式	0	采集温度或湿度。
			1	按顺序获取温度和湿度，先获取温度。
BTST	[11]	电池状态	0	电池电压 > 2.8V（只读）
			1	电池电压 < 2.8V（只读）
TRES	[10]	温度测量分辨率	0	14 位
			1	11 位
HRES	[9:8]	湿度测量分辨率	00	14 位
			01	11 位
			10	8 位
保留	[7:0]	保留	0	保留，必须为 0

在以下示例中，HDC1x 配置为通过将 0x10 (MSB) 和 0x00 (LSB) 写入寄存器来按顺序测量温度和湿度。图 2-2 显示了设置的寄存器位。

当 HDC1x 设置为按顺序输出温度和湿度时，用户必须开始从温度寄存器 (0x00) 的 4 个字节读取。

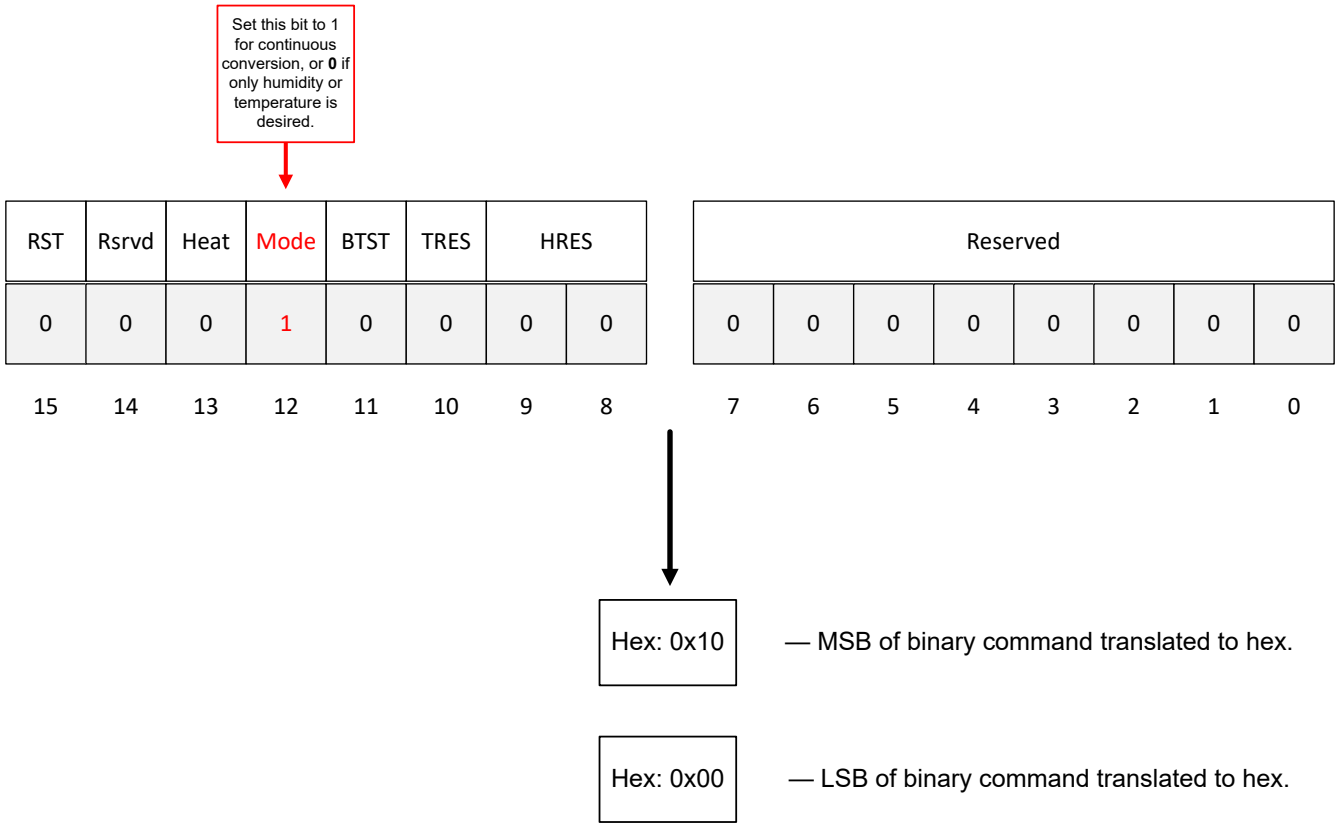

图 2-2. 用于温度和湿度采集的配置寄存器位

用于设置配置的代码如下：

```
Wire.beginTransmission(0x40); // initiate communication with HDC1x
                Wire.write(0x02); // point to configuration register Wire.write(0x10); // write
                8-bit configuration to config register (MSB) Wire.write(0x00); // write 8 0s to
                Reserved bits (LSB) Wire.endTransmission();
```

接下来，通过写入器件地址 (0x40) 来触发测量过程。

```
Wire.beginTransmission(0x40); // initiate communication with HDC1x
                Wire.write(0x00); // start measurements Wire.endTransmission(); delay(20); // wait
                20ms for conversion to complete.
```

由于按顺序测量温度和湿度，因此需要从寄存器 0x00 的 4 个字节读取。前两个字节对应温度，之后的两个字节对应湿度数据。

```
Wire.requestFrom(0x40, 4); // requesting 4 bytes from device // once 4 bytes
                are received, store this in appropriate variables if (Wire.available() == 4) { //
                stores raw temperature and humidity data // reads/stores first byte (MSB), then
                reads/stores second byte // combines each pair of bytes into a 16-bit integer
                uint16_t tempBytes = (Wire.read() << 8) | Wire.read(); uint16_t humBytes =
                (Wire.read() << 8) | Wire.read(); }
```

最后，应用 HDC1x 数据表中的标准转换公式：

```
// equation for converting temperature output in Celsius temp = (tempBytes /
                65536.0) * 165.0 - 40.0; // equation for converting humidity output hum = (humBytes
                / 65536.0) * 100.0;
```

此 Arduino 示例演示了在按顺序采集测量值时如何从 HDC1x 传感器读取和存储温度和湿度数据。存储原始数据后，可以使用 HDC1x 数据表中提供的公式将其转换为实际温度和湿度值。

此处的 TI GitHub 环境传感器存储库上提供了的完整工作例子。

2.1.1.2 HDC2x

HDC2x 系列（HDC2010、HDC2021、HDC2022、HDC2080）也使用基于寄存器映射的数字接口，类似于 HDC1x 系列。如 表 2-3 中所示，所有 HDC2x 器件共享一个通用的寄存器布局，以下过程适用于该系列。本节概述了如何在按需触发（单次触发）和自动测量（连续转换）模式下与这些器件连接。

表 2-3. HDC2x 寄存器映射

指针	名称	复位值	说明
0x00	TEMPERATURE LOW	0x00	温度 [7:0]
0x01	TEMPERATURE HIGH	0x00	温度 [15:8]
0x02	HUMIDITY LOW	0x00	湿度 [7:0]
0x03	HUMIDITY HIGH	0x00	湿度 [15:8]
0x04	INTERRUPT/DRDY	0x00	DataReady 和中断配置
0x05	TEMPERATURE MAX	0x00	测得的最高温度 （在自动测量模式下不受支持）
0x06	HUMIDITY MAX	0x00	测得的最高湿度 （在自动测量模式下不受支持）
0x07	INTERRUPT ENABLE	0x00	中断启用
0x08	TEMP_OFFSET_ADJUST	0x00	温度偏移调整
0x09	HUM_OFFSET_ADJUST	0x00	湿度偏移调整
0x0A	TEMP_THR_L	0x00	温度阈值低
0x0B	TEMP_THR_H	0xFF	温度阈值高
0x0C	RH_THR_L	0x00	湿度阈值低
0x0D	RH_THR_H	0xFF	湿度阈值高
0x0E	RESET&DRDY/INT CONF	0x00	软复位和中断配置
0x0F	MEASUREMENT CONFIGURATION	0x00	测量配置
0xFC	MANUFACTURER ID LOW	0x49	制造商 ID 低
0xFD	MANUFACTURER ID HIGH	0x54	制造商 ID 高
0xFE	DEVICE ID LOW	0xD0	器件 ID 低
0xFF	DEVICE ID HIGH	0x07	器件 ID 高

备注

在以下 HDC2x 示例中，使用了配置为地址 0x40（ADDR 引脚连接到 GND）的 HDC2010，但可以使用一个全局变量，以根据您的设置方便地调整地址。

与 HDC1x 系列的一个关键区别是 HDC2x 器件使用单独的 8 位寄存器来存储每次测量的最高有效位和最低有效位：

- 温度：TEMP_LOW (LSB)、TEMP_HIGH (MSB)
- 湿度：HUM_LOW (LSB)、HUM_HIGH (MSB)

除了这些数据寄存器外，HDC2x 系列还包括一个**测量配置寄存器**，允许用户定义测量参数。

测量过程首先写入配置寄存器，该寄存器控制一些关键功能，例如（但不限于）：

- 加热器使能 (HEAT_EN)
- 自动测量模式 (AMM)
- 软复位 (SOFT_RES)

后续步骤涉及设置测量参数和启动转换，后续各小节将对此进行详细介绍。

2.1.1.2.1 按需触发模式下的连接

在本例中，已将 HDC2010 传感器配置为**按需触发（单次触发）模式**，并禁用了自动测量模式 (AMM)。该过程首先写入配置寄存器 (0x0E)，以将器件设置为按需触发模式。图 2-3 中提供了配置寄存器设置的图示。

表 2-4. 配置寄存器 (0x0E)

位	字段	类型	复位	说明
7	SOFT_RES	R/W	0	0 = 正常工作模式，此位会自行清除 1 = 软复位 EEPROM 值重新加载且寄存器复位
[6:4]	AMM[2:0]	R/W	000	自动测量模式（AMM） 000 = 禁用。通过 I2C 启动测量 001 = 1/120Hz（每 2 分钟进行 1 次采样） 010 = 1/60Hz（每分钟进行 1 次采样） 011 = 0.1Hz（每 10 秒钟进行 1 次采样） 100 = 0.2Hz（每 5 秒钟进行 1 次采样） 101 = 1Hz（每秒钟进行 1 次采样） 110 = 2Hz（每秒钟进行 2 次采样） 111 = 5Hz（每秒钟进行 5 次采样）
3	HEAT_EN	R/W	0	0 = 加热器关闭 1 = 加热器开启
2	DRDY/INT_EN	R/W	0	DRDY/INT_EN 引脚配置 0 = 高阻抗 1 = 启用
1	INT_POL	R/W	0	中断极性 0 = 低电平有效 1 = 高电平有效
0	INT_MODE	R/W	0	中断模式 0 = 电平敏感型 1 = 比较器模式

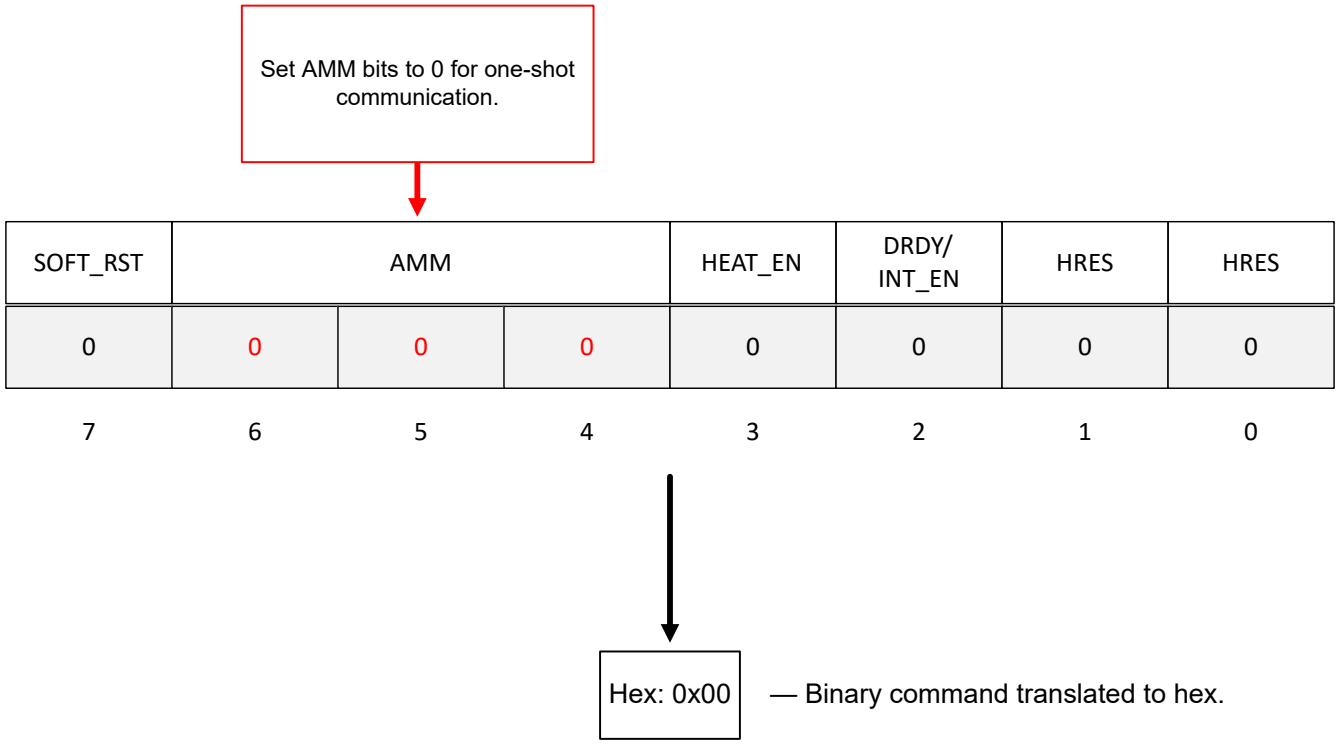

图 2-3. 用于按需触发（单次触发）的配置寄存器

```
Wire.beginTransmission(0x40); // initiate communication with HDC2x sensor
Wire.write(0x0E); // write to Config Register
Wire.write(0x00); // configure device to Trigger-On Demand
Wire.endTransmission();
```

接下来，按 图 2-4 中所示配置测量配置寄存器 (0x0F)。

测量配置寄存器定义以下内容：

- 温度和湿度分辨率 (TRES 和 HRES)
- 测量类型（仅温度，仅湿度，或两者）
- 测量触发器 (MEAS_TRIG)

表 2-5. 测量配置寄存器 (0x0F)

位	字段	类型	复位	说明
7:6	TRES[1:0]	R/W	00	温度分辨率 00：14 位 01：11 位 10：9 位 11：不适用
5:4	HRES[1:0]	R/W	00	湿度分辨率 00：14 位 01：11 位 10：9 位 11：不适用
3	RES	R/W	0	保留
2:1	MEAS_CONF[1:0]	R/W	00	测量配置 00：湿度 + 温度 01：仅温度 10：NA 11：不适用

表 2-5. 测量配置寄存器 (0x0F) （续）

位	字段	类型	复位	说明
0	MEAS_TRIG	R/W	0	测量触发 0：无操作 1：开始测量 测量完成时自行清除位

图 2-4 提供了测量配置寄存器的图示。

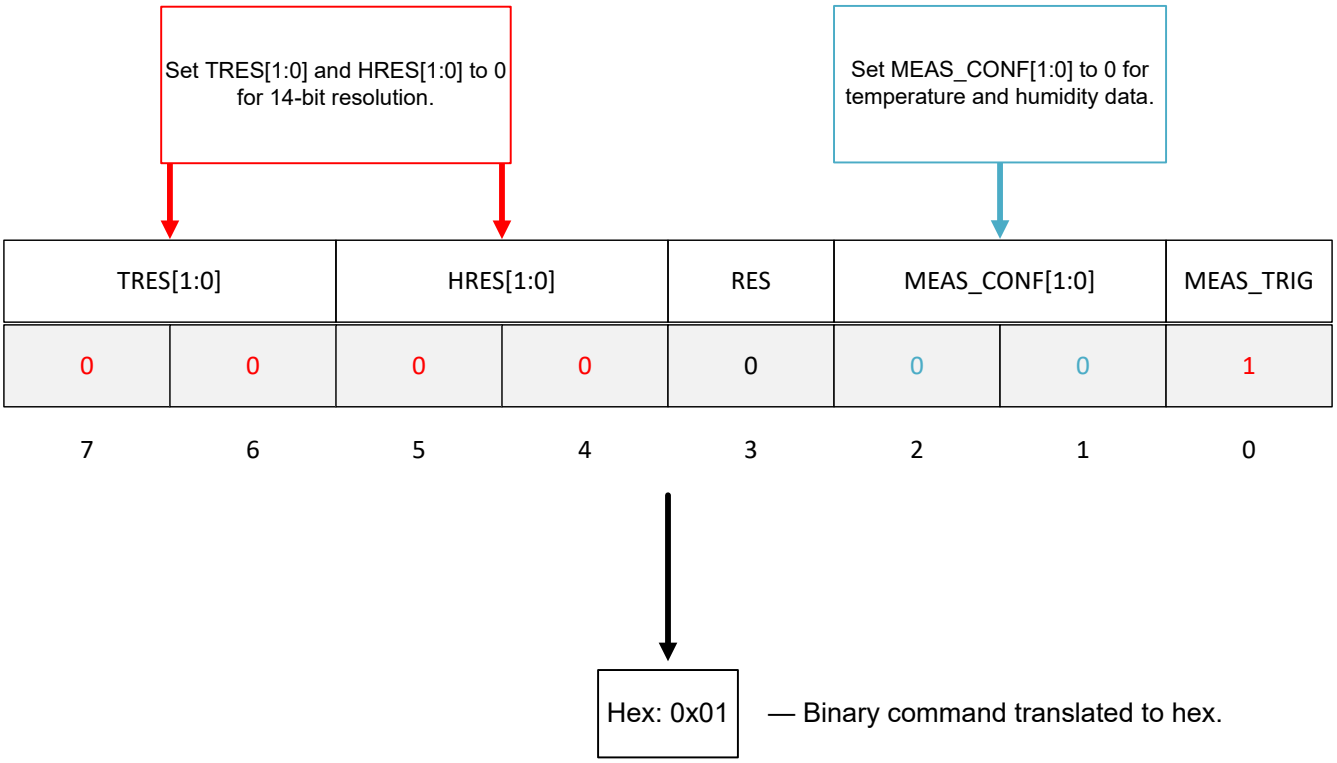

图 2-4. 测量配置寄存器设置

```
Wire.beginTransmission(0x40); // initiate communication with HDC2x sensor

Wire.write(0x0F); // write to Measurement Config Register

Wire.write(0x01); // set output to 14-bit temperature/humidity data
                  // and trigger measurements

Wire.endTransmission();
```

在按需触发模式下，可以根据系统的需求将此序列放置在循环内以进行定期轮询。

要读取 HDC2x 的温度和湿度测量值，用户可以在两种可能的方法中进行选择。第一种方法是使用 TEMP_LOW/HUM_LOW 和 TEMP_HIGH/HUM_HIGH 分别读取/存储温度和湿度字节，并将 MSB 和 LSB 位组合为一个 16 位值，或在一个通信帧中突发读取多个字节，如以下湿度数据采集代码中所示：

读取测量数据

HDC2x 通过两个 8 位寄存器存储测量结果：

- 湿度：HUM_LOW (0x02)、HUM_HIGH (0x03)
- 温度：TEMP_LOW (0x00)、TEMP_HIGH (0x01)

一种方法是分别读取每个字节并将它们组合在一起：

```
uint16_t getHum() {

    // RH LSB Acquisition

    Wire.beginTransmission(0x40); // start communication with HDC2x
    Wire.write(0x02); // set a pointer for Humidity Low register (0x02)
    Wire.requestFrom(0x02, 2); // request 2 bytes from HDC2x
    uint8_t humLow = Wire.read(); // store Humidity LSB data
    Wire.endTransmission();

    // RH MSB Acquisition

    Wire.beginTransmission(0x40); // start communication with HDC2x
    Wire.write(0x03); // set a pointer for Humidity High register (0x03)
    Wire.requestFrom(0x40, 2); // request 2 bytes from HDC2x
    uint8_t humHigh = Wire.read(); // store Humidity MSB data
    Wire.endTransmission();

    // combine MSB and LSB into 16-bit integer and return value

    return ((uint16_t) humHigh << 8) | humLow;

}
```

不过，更有效的方法是从 LSB 寄存器一次突发读取两个字节：

```
uint16_t getHum2() {

    Wire.beginTransmission(0x40); // start communication with HDC2x
    Wire.write(0x02); // set a pointer for Humidity Low register (0x02)
    Wire.endTransmission(false);
    Wire.requestFrom(0x40, 2); // request 2 bytes from HDC2x

    uint8_t lsb = Wire.read(); // read and store LSB
    uint8_t msb = Wire.read(); // read and store MSB

    // adds MSB of data to an empty 16-bit variable
    // shifts 8 bits left, then "or" with LSB for final value

    return ((uint16_t) msb << 8) | lsb;

}
```

此方法利用 HDC2x 的内部指针行为：从 HUMIDITY_LOW 寄存器读取 LSB 后，指针自动递增到 MSB 的 HUMIDITY_HIGH 寄存器。同样的机制也适用于读取温度。

请访问以下链接，查看 HDC2x 在按需触发模式下的完整示例代码。

2.1.1.2.2 使用自动测量模式 (AMM) 连接

本节概述了如何将 HDC2x 器件配置为在自动测量模式 (AMM) 下运行，并重点介绍了与按需触发模式相比的主要差异。

在 AMM 中，器件会自动以用户定义的采样频率执行测量，无需从 MCU 手动触发测量。与按需触发 (每次测量都必须手动启动) 不同，AMM 只需一个触发器即可开始周期性转换。

在本例中，HDC2010 配置为每 5 秒 (0.2Hz) 采样一次。这通过将适当的设置写入配置寄存器 (0x0E) 来实现。图 2-5 中提供了配置寄存器设置的图示。

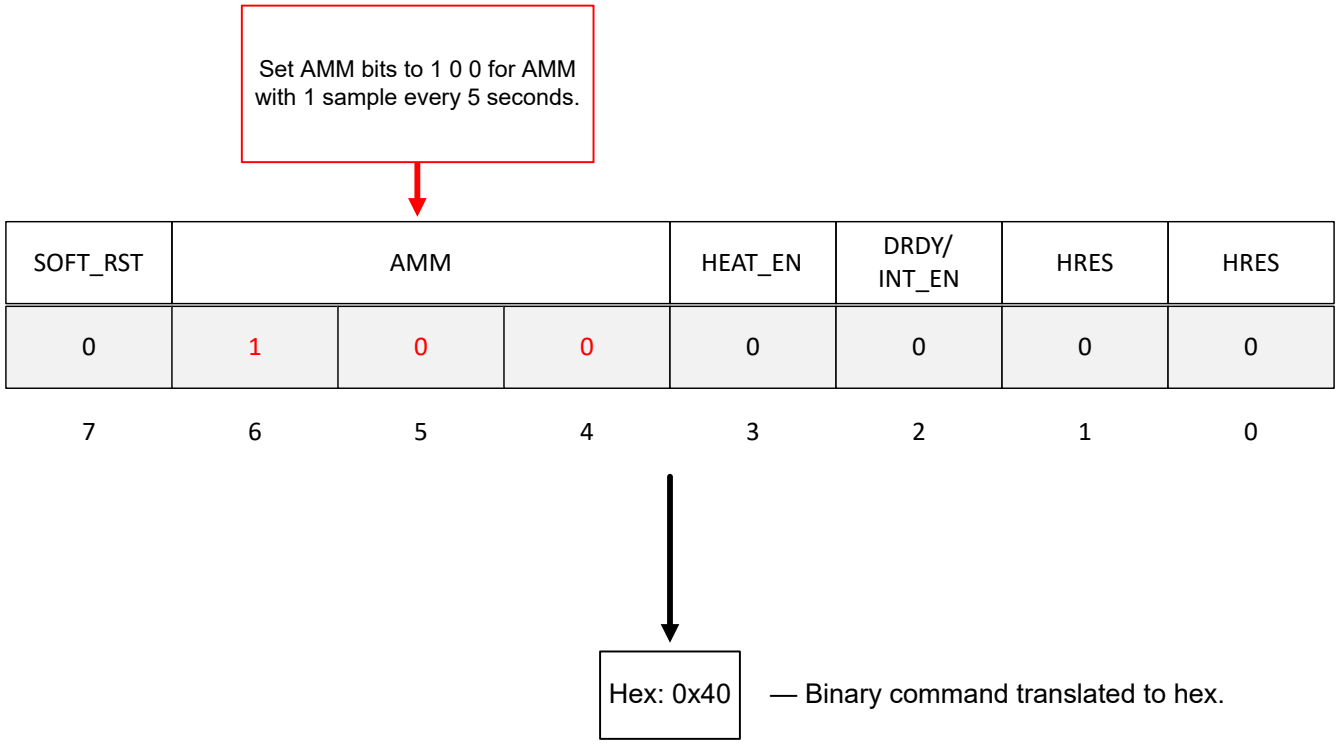

图 2-5. 自动测量模式 (AMM) 的配置寄存器

```
// set device to Auto Measurement Mode for 0.2Hz (1 sample/5 seconds)

Wire.beginTransmission(0x40); // start communication with HDC2x

Wire.write(0x0E); // point to register 0x0E (Measurement Config)
Wire.write(0x40); // write value to register

Wire.endTransmission(); // end communication
```

测量配置寄存器使用“按需触发”一节中所述的相同配置。

请访问以下链接，查看自动测量模式下 HDC2x 的完整示例代码。

2.2 命令协议

2.2.1 HDC302x

HDC302x 系列引入了基于命令的接口，不再沿用前几代产品使用的基于寄存器映射的方案。HDC302x 器件不会写入特定寄存器，而是响应明确定义的命令代码，以开始测量、配置设置或检索数据。表 2-6 显示了 HDC302x 器件支持的命令示例。

这种方法简化了接口并减少了所需 I2C 事务的数量，尤其是在只需要基本测量的应用中。

后续几节将演示如何在按需触发和自动测量模式下与 HDC302x 连接，并举例说明了在每种情况下如何实现基于命令的通信。

表 2-6. HDC302x 命令表片段

十六进制代码 (MSB)	十六进制代码 (LSB)	命令	命令详细信息
24	00	按需触发模式 单个温度 (T) 测量 和相对湿度 (RH) 测量	低功耗模式 0（最低噪声）
24	0B		低功耗模式 1
24	16		低功耗模式 2
24	FF		低功耗模式 3（最低功耗）
20	32	自动测量模式 每 2 秒测量 1 次。	低功耗模式 0（最低噪声）
20	24		低功耗模式 1
20	2F		低功耗模式 2
20	FF		低功耗模式 3（最低功耗）
21	30	自动测量模式 每秒测量 1 次。	低功耗模式 0（最低噪声）
21	26		低功耗模式 1
21	2D		低功耗模式 2
21	FF		低功耗模式 3（最低功耗）
22	36	自动测量模式 每秒测量 2 次。	低功耗模式 0（最低噪声）
22	20		低功耗模式 1
22	2B		低功耗模式 2
22	FF		低功耗模式 3（最低功耗）
23	34	自动测量模式 每秒测量 4 次。	低功耗模式 0（最低噪声）
23	22		低功耗模式 1
23	29		低功耗模式 2
23	FF		低功耗模式 3（最低功耗）
27	37	自动测量模式 每秒测量 10 次。	低功耗模式 0（最低噪声）
27	21		低功耗模式 1
27	2A		低功耗模式 2
27	FF		低功耗模式 3（最低功耗）
2C	06	按需触发模式 单个温度 (T) 测量 和相对湿度 (RH) 测量	低功耗模式 0（最低噪声）
2C	0D		低功耗模式 1
2C	10		低功耗模式 2

表 2-6. HDC302x 命令表片段（续）

十六进制代码 (MSB)	十六进制代码 (LSB)	命令	命令详细信息
30	93	自动测量模式	退出，然后返回按需触发模式。
E0	00		T 和 RH 的测量读数 （注意：如果 RH 和 T 未更新，则数据读出所有 FF）
E0	01		仅 RH 的测量读数
E0	02		最小 T 的测量历史读数。
E0	03		最大 T 的测量历史读数。
E0	04		最小 RH 的测量历史读数。
E0	05		最大 RH 的测量历史读数。

2.2.1.1 按需触发模式下的连接（单次触发）

本节演示了如何使用超高分辨率设置来配置 HDC302x 器件并从中读取测量值。所有示例代码都对 I2C 地址使用全局变量，并可根据用户的器件配置轻松更改。

备注

本示例中使用的 HDC302x 传感器配置为地址 0x44（ADDR 和 ADDR1 引脚连接到 GND），但代码中提供了一个全局变量，可根据您的布局方便地调整地址。

在读取测量值之前，必须先对器件进行配置。可以通过向 HDC302x 发送特定的命令序列来完成该操作。图 2-6 中显示了此配置的图示。

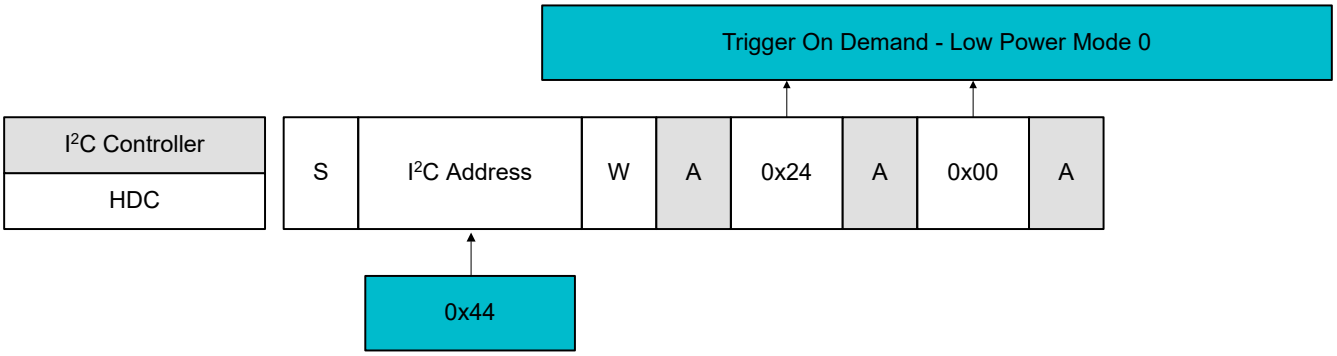

图 2-6. 按需触发命令选择

```
Wire.beginTransmission(0x44); // Initiate communication with HDC302x

Wire.write(0x24); // Write Command MSB to device.

Wire.write(0x00); // Write Command LSB to device.

Wire.endTransmission();

delay(25); //wait 25ms before reading
```

需要延迟一段时间来确保测量转换已完成，然后再开始读取。在本例中，由于测量配置基于最高分辨率和可重复性，因此使用了 15ms 延迟。尽管应实现 15ms 的最小延迟，但在设置适当的延迟之前，工程师应参考数据表。

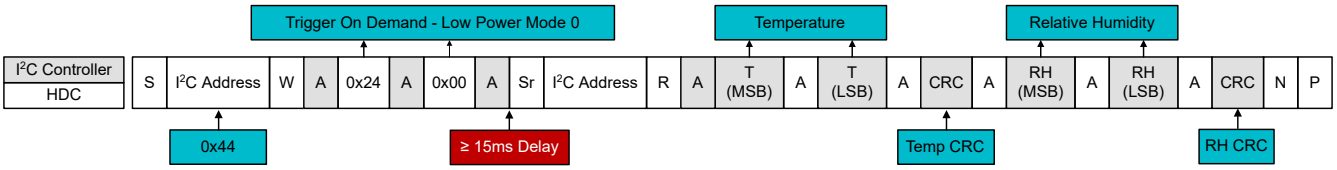

图 2-7. HDC302x 按需触发通信结构

```
void loop() {

    float humidity;
    float temp;

    // send device command for highest repeatability
    Wire.beginTransmission(0x44);
    Wire.write(0x24); //send MSB of command
    Wire.write(0x00); //command LSB
    Wire.endTransmission();
    delay(15); //wait 15ms before reading

    Wire.requestFrom(0x44, 6); //request 6 bytes from HDC device
```

```
    Wire.readBytes(HDC_DATA_BUFF, 6); //move 6 data bytes into
buffer

    temp = getTemp(HDC_DATA_BUFF);
    Serial.print("Temp (C): "); // print final temp value
    Serial.println(temp);

    delay(1000); // wait 1 second (optional)

    humidity = getHum(HDC_DATA_BUFF);
    Serial.print("Humidity (RH): "); // print final humidity value
    Serial.print(humidity);
    Serial.println("%");

    delay(1000); // wait 1 second (optional)

}
```

数据转换功能

使用 HDC302x 数据表中提供的公式计算温度和湿度：

```
// function processes raw temperature values and returning final value
float getTemp(uint8_t humBuff[]) {

    float tempConv;
    float celsius;

    TEMP_MSB = humBuff[0] << 8 | humBuff[1]; //shift 8 bits of data in
                                             //first array index to get
                                             //MSB then OR with LSB

    tempConv = (float)(TEMP_MSB); // convert uint8_t temp value
    celsius = ((tempConv / 65535) * 175) - 45; // calculate celcius

    return celsius;

}

// function for processing raw humidity values and returning final value
float getHum(uint8_t humBuff[]) {

    float humConv;
    float humidity;

    HUM_MSB = (humBuff[3] << 8) | humBuff[4]; //shift 8 bits of data in
                                              //first array index to get
                                              //MSB then OR with LSB

    humConv = (float)(HUM_MSB); // convert uint8_t humidity value
    humidity = (humConv / 65535) * 100; // calculate humidity

    return humidity;

}
```

缓冲器结构的注释

六字节缓冲区 HDC_DATA_BUFF 包含：

- 字节 0-1：原始温度数据（MSB、LSB）
- 字节 2：温度 CRC（可选）
- 字节 3-4：原始湿度数据（MSB、LSB）
- 字节 5：湿度 CRC（可选）

尽管本示例不使用 CRC 字节，但这些字节包含在缓冲区中以确保完整，如果需要，可以检查这些字节以确保数据完整性。

请访问以下链接，查看 HDC302x 在按需触发模式下的完整示例代码。

2.2.1.2 自动测量模式 (AMM) 下连接

本节概述了如何为 AMM 配置 HDC302x，并说明了与按需触发模式相比的主要差异。

自动测量模式和按需触发之间的主要区别在于：如果用户想要以固定间隔从 HDC302x 传感器读取数据，则自动测量模式更适合其可编程输出间隔。图 2-8 显示了对 HDC302x 进行编程的命令序列，每秒输出一次测量，具有较低噪声和较高可重复性。

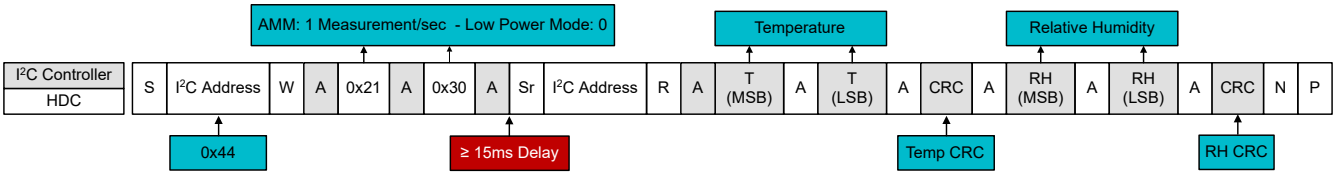

图 2-8. HDC302x 自动测量模式 (AMM) 通信结构

```
// configure HDC302x for Auto Measurement Mode (1 measurement/sec)
// lowest noise, highest repeatability
void deviceInit() {

    Wire.beginTransmission(0x44);
    Wire.write(0x21); //send MSB of command
    Wire.write(0x30); //command LSB
    Wire.endTransmission();
    delay(15); //wait 15ms before reading

}
```

在器件配置为自动测量模式并且经过足够的时间来完成转换（本例中为一秒）后，使用以下函数来请求存储的测量数据：

```
// Helper function for requesting data when in Auto Measurement Mode
void requestData() {

    Wire.beginTransmission(DEVICE_ADDR); // initiate communication
    Wire.write(0xE0); // send MSB of read command
    Wire.write(0x00); // send LSB of read command
    Wire.endTransmission();

}
```

若要在 AMM 中持续轮询器件，您可以发出读取命令以在配置的采样率（每秒测量次数）下检索最新的测量数据。

此处的 TI GitHub™ 环境传感器存储库提供了演示 HDC302x 在 AMM 中的工作的完整 Arduino 示例——包括器件配置、测量轮询和数据读数。

如何使用 CRC 校验测量数据

对 HDC302x 的测量输出执行 CRC 校验是确保数据完整性和防止错误读数的重要步骤，尤其是在医疗设备、冷链或气象站等关键应用中。本节提供了一个简单的代码示例，可轻松实现到 HDC302x 的现有代码中。本示例特别重点介绍对温度和湿度读数执行 CRC 校验。

HDC3020 遵循 CRC-8 标准，因为它为温度和湿度测量输出唯一的 8 位 CRC 值。上面 图 2-8 说明了这一点。根据用户希望检查湿度还是温度数据，以下算法总共接受三个字节——MSB、LSB 和 CRC 作为参数以及发送的总字节数。然后，它使用数据表中指定的多项式值 0x31 进行计算。方程式 1 显示了用于 CRC 校验计算的多项式。

$$0x31 = x^8 + x^5 + x^4 + 1 \quad (1)$$

在处理了测量数据和 CRC 字节后，如果最终值为 0x00，则 CRC 校验已通过，程序会继续输出测量读数；否则，它会在控制台中输出错误消息。

```
// function for checking CRC for HDC measurements
uint8_t checkMeasurementCRC(uint8_t data[], uint8_t dataLength){

uint8_t crc = 0xFF; // initial value per HDC302x datasheet
uint8_t byte;
uint8_t bit;
```

```
for (byte =0; byte < dataLength; byte++){ // loops through each byte of input data
    crc ^= data[byte]; // XOR next data byte into current CRC value
    for (bit = 0; bit < 8; bit++){ // process each bit from the data byte
        if (crc & 0x80) // if MSB of CRC is 1
            crc = (crc << 1) ^ 0x31; // shift left and apply polynomial
        else
            crc = (crc << 1); // else shift left, but no polynomial application
    }
}
Serial.print("CRC Check Result: "); // optional; prints CRC value for debugging
Serial.println(crc);
return crc; // return final CRC value
}
```

然后，可以在读取温度或湿度时调用此函数，以检查数据完整性。

```
Humidity Read Example w/ CRC Check:

uint8_thumCheck[3] = {HDC_DATA_BUFF[3], HDC_DATA_BUFF[4], HDC_DATA_BUFF[5]};

// if algorithm output equals final byte value of 0x00, CRC check passes, else output error message
if ((checkMeasurementCRC(humCheck, 3)) == 0x00){

Serial.println("Humidity CRC check passed.");
humidity = getHum(HDC_DATA_BUFF);
Serial.print("Humidity (RH): ");
Serial.print(humidity);
Serial.println("%");

} else {

    Serial.println("Error: Humidity CRC Check Failed.");

}
```

下面 图 2-9 中提供了示例输出。可以采用相同的方法来检查温度数据完整性。

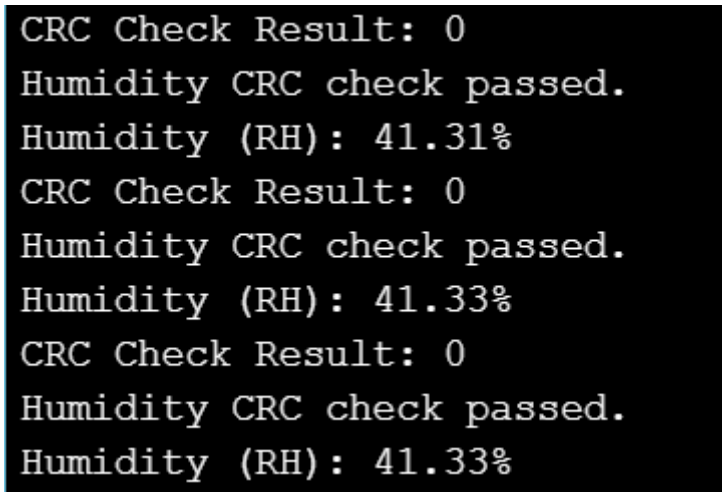

图 2-9. CRC 校验示例输出

在 TI 基于 GUI 的代码生成器 ASC Studio 中可以找到在 C 语言中检查警报 CRC 的示例。

3 模拟接口概述

3.1 HDC3120

HDC3120 是德州仪器 (TI) 具有模拟比例式输出的首款湿度传感器。HDC3120 提供与温度和相对湿度对应的连续电压信号，因此设计得非常适合低噪声模拟前端系统。

与需要通信协议的数字传感器不同，HDC3120 允许直接访问传感器输出，无需写入 I2C 命令或配置寄存器。这简化了在已有模数转换器 (ADC) 的系统中的集成。

要解读输出信号，用户可以应用适当的转换公式（如下提供），将电压电平转换为温度（°C）和相对湿度 (%RH)。

HDC3120 的一个重要特性是比例式行为。温度和湿度的输出电压随器件的电源电压 (VDD) 线性调整，后者也用作内部基准。该设计提供电源噪声和漂移抗扰，可实现可靠的测量。以下温度和湿度输出曲线图显示了该关系。

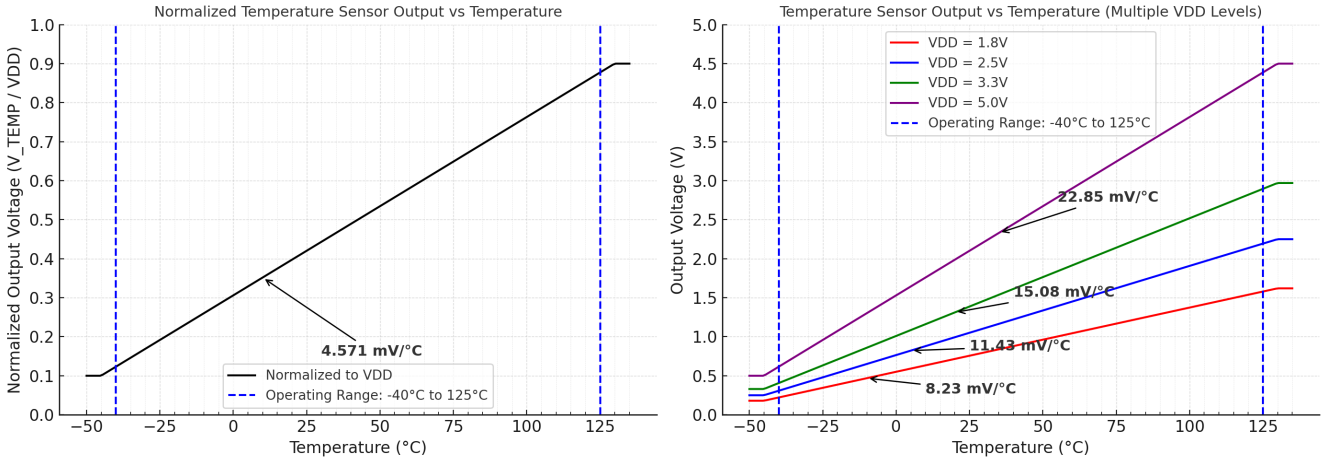

图 3-1. 比例式温度输出曲线和转换公式

$$T(°C) = 218.75 \times \frac{V_{OUT}}{V_{DD}} - 66.875 \quad (2)$$

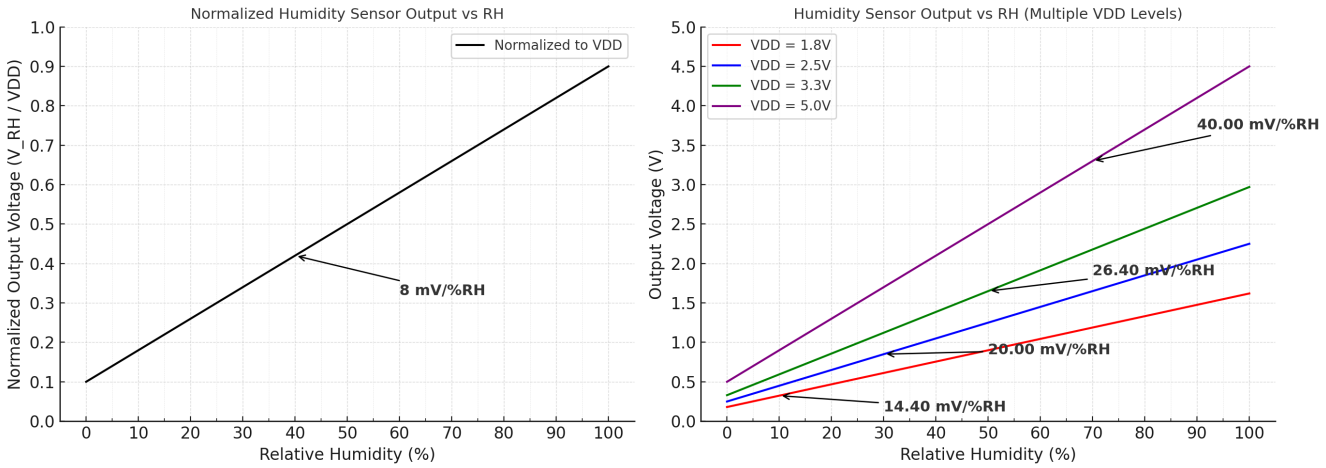

图 3-2. 比例式相对湿度输出曲线和转换公式

$$\%RH = 125 \times \frac{V_{OUT}}{V_{DD}} - 12.3 \quad (3)$$

了解 HDC3120 输出的比例式性质至关重要——尤其是在连接 ADC 时，因为传感器的输出电压与电源电压 (VDD) 成正比。如果所选 ADC 也使用其电源作为参考，则在电压电源内出现噪声或漂移时，可以保持测量一致性。这种对齐方式可消除增益不匹配问题，并有助于保持传感器读数一致。

如何为 HDC3120 选择 ADC：

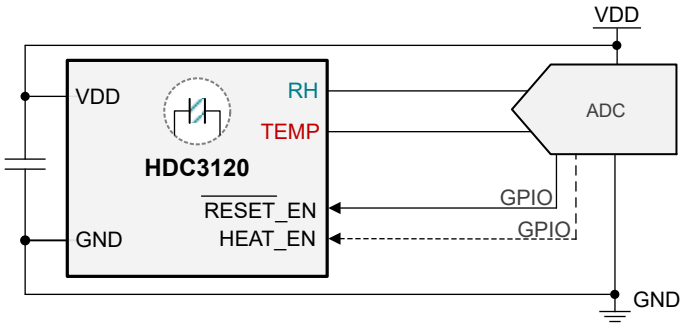

图 3-3. HDC3120 至 ADC

表 3-1 为 HDC3120 提供了一些 ADC：

表 3-1. HDC3120 的 ADC

ADC	分辨率	比例式	汽车级	电源电压范围	何时选择
ADS7142	12 位	是	Q100	1.65-3.6V	集成超出系统限制时的警报功能；最多两个单端输入
ADS7138	12 位	是	Q100	2.35V 至 5.5V	通过 GPIO 引脚进行加热器/使能控制；最多八个单端输入
ADS7066	16 位	是	—	3V-5.5V	通过 GPIO 引脚进行加热器/使能控制；最多八个单端输入
ADS1015	12 位	否	Q100	2V-5.5V	可编程增益放大器 (PGA) 可实现更宽的输入电压范围；最多四个单端输入

在为 HDC3120 选择 ADC 时，必须考虑以下过程。

1. 确定 ADC 的最低有效位 (LSB)
2. 计算 HDC3120 温度 LSB
3. 计算 HDC3120 的湿度 LSB
4. 比较 LSB 值

选择小于 HDC3120 温度 LSB 的 ADC LSB。但是，湿度 LSB 可以使用相同的考虑因素，因为 HDC3120 的温度传感器具有更高的精度，所以可使用它来确定 ADC 所需的最小 LSB。

ADS1015 无法执行比例式测量，因为它只有一个内部基准，不提供外部基准的选项。此外，需要使用以下公式求解 LSB：

$$LSB = \frac{FSR}{2^n}; \quad FSR_{ADS1015} = \frac{2 \times V_{REF}}{gain} \tag{4}$$

其中 FSR (满标量程) 表示比例因子和 V_{REF} 是 ADC 的参考电压。如果是 ADS1015，使用 ADS1015 数据表中的表 7-1 通过设置可编程增益放大器 (PGA) 来确定增益。

有关将 HDC3120 与 ADC 配对的其他信息，请参阅 HDC3120 数据表中的第 8.2.2 节。

示例场景：

在本示例中，BOOSTXL-ADS7142-Q1 EVM 与 HDC3120EVM 配对。两个器件均使用相同的 3.3V 电源。从此处，可以使用上述步骤比较两个器件的 LSB：

1. 使用以下公式确定 ADC 的最低有效位 (LSB)：

$$\frac{FSR}{Resolution} = LSB_{ADC} \tag{5}$$

   a. 由于工作电压为 3.3V 且 ADC 具有 12 位分辨率，所以 LSB 为：

$$\frac{3.3V}{2^{12}} = 0.805mV \tag{6}$$

2. 使用以下公式计算 HDC3120 的温度 LSB：

$$V_{TEMP} = V_{DD} \times \left[T(°C) \times 4.571\frac{mV}{°C}\right] \tag{7}$$

$$Temp_{LSB} = 3.3V \times \left[\left(0.1\right) \times 4.571\frac{mV}{°C}\right] \quad = \quad 1.508 \quad mV/°C \tag{8}$$

温度值 (T(°C)) 为 0.1，反映 HDC3120 的典型温度检测精度。

3. 使用 方程式 9 计算 HDC3120 的湿度 LSB：

$$V_{RH} = V_{DD} \times \left[(\%\,RH) \times 8\frac{mV}{\%\,RH}\right] \tag{9}$$

$$RH_{LSB} = 3.3V \times \left[(1.0) \times 8\frac{mV}{\%RH}\right] = 26.4 \quad mV/\%\,RH \tag{10}$$

湿度值 (%RH) 为 1.0，反映 HDC3120 的典型湿度检测精度。

4. 比较 LSB 值
   a. 由于在 3.3V 下 HDC3120 的温度 LSB 大小为 1.508mV，而 ADS7142 的 LSB 为 0.805mV；这意味着用户保留至少 1°C 的测量精度。如果需要更高的精度，例如 0.5°C，则应使用 ADS7066 等分辨率更高的 ADC。

在此示例设置中，HDC3120EVM 已连接到 BOOSTXL-ADS7142-Q1 EVM。两个器件共用同一个 VDD 电源导轨。设置过程如下文所示：

1. 使用跳线将 HDC3120EVM 的 TEMP 输出连接到 BOOSTXL-ADS7142-Q1 EVM BoosterPack™ 上的 AIN0 输入，如 图 3-4 所示。

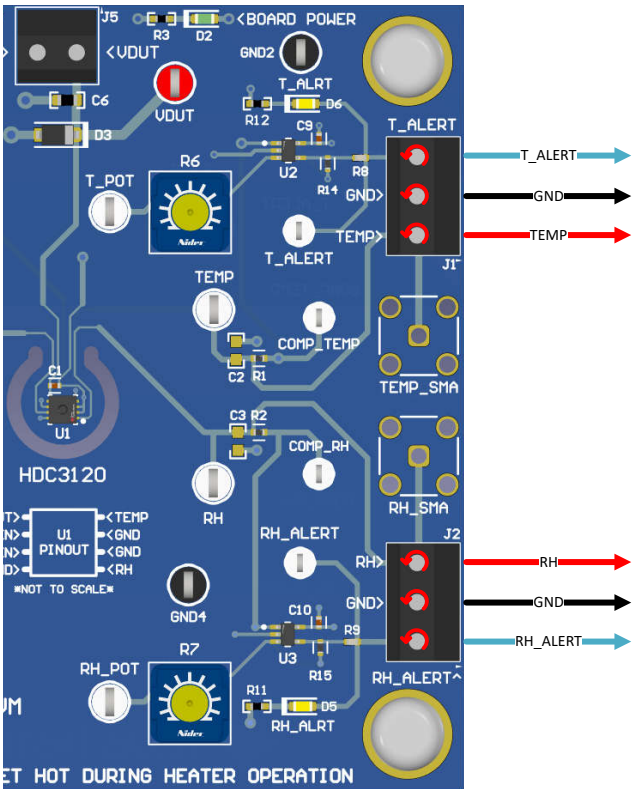

图 3-4. HDC3120EVM 输出

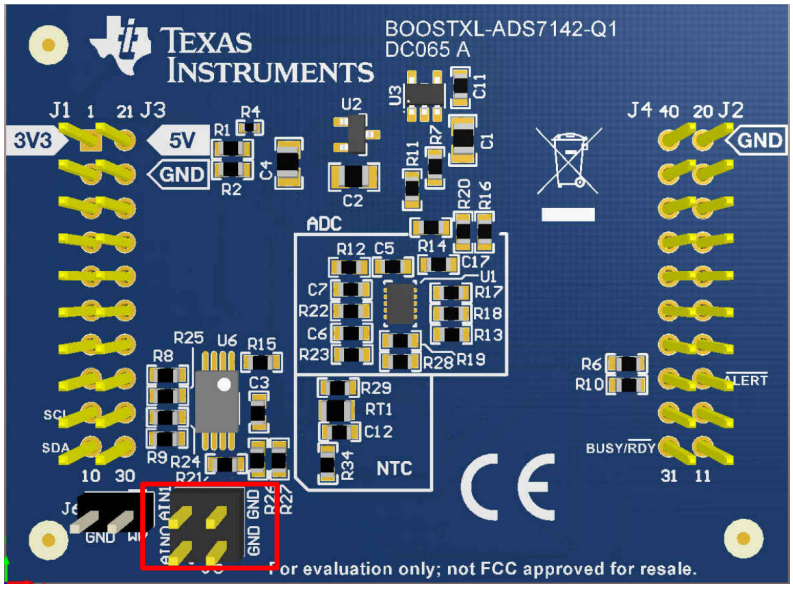

图 3-5. BOOSTXL-ADS7142-Q1 EVM BoosterPack™ 引脚连接位置

2. 将 HDC3120EVM 的 RH 输出连接到 BOOSTXL-ADS7142-Q1 EVM BoosterPack™ 上的 AIN1。
3. 启动 ADS7142EVM GUI 并导航到主页。
4. 点击齿轮图标以访问配置菜单，然后在“命令模式”下选择“I2C 命令模式”。
5. 从*操作模式*下拉菜单中选择*自动 SEQ 模式*，然后点击红色*设置*按钮。
6. 配置完后，点击*启动序列*，开始以 12 位十进制格式捕获测量值。

备注

以下用于转换 HDC3120 温度和湿度输出的公式专为采用 12 位 BOOSTXL-ADS7142-Q1 EVM 的应用而设计。

为了解读结果，用户可以应用 HDC3120 的转换公式。方程式 11 和 方程式 12 显示了通过 HDC3120 转换公式重新解读比率。方程式 13 提供了变量的其他定义。

$$\frac{V_{RH}}{V_{DD}} = \frac{RH_{ADC}}{2^{12}} \tag{11}$$

$$\frac{V_{TEMP}}{V_{DD}} = \frac{TEMP_{ADC}}{2^{12}} \tag{12}$$

$$\begin{aligned} &\boldsymbol{TEMP_{ADC}} \rightarrow \text{HDC3120 to ADC temperature decimal output} \\ &\boldsymbol{RH_{ADC}} \rightarrow \text{HDC3120 to ADC relative humidity decimal output} \\ &\boldsymbol{2^{12}} \rightarrow \text{represents the 12 – bit resolution of the ADC outputs} \end{aligned} \tag{13}$$

备注

由于本示例的接线配置，引脚 AIN0（温度）代表 GUI 中的通道 0，而引脚 AIN1（湿度）代表通道 1。确保两者都打开以接收输出数据。

图 3-6 HDC3120 的温度和湿度测量图。通道 0（上图）表示温度读数，而通道 1（下图）表示湿度读数。每个通道的“最新数据”旁边提供了最新数据。图中的曲线表示大约 3 秒的间隔，其中 HDC3120 加热器在通过 HDC3120EVM 上的加热器使能开关禁用之前使能了三秒。因此，每次使能加热器时，都会观察到温度峰值，而在器件因加热而干燥时观察到湿度谷值（与峰值相反）（这是具有集成加热元件的湿度传感器中的正常现象）。

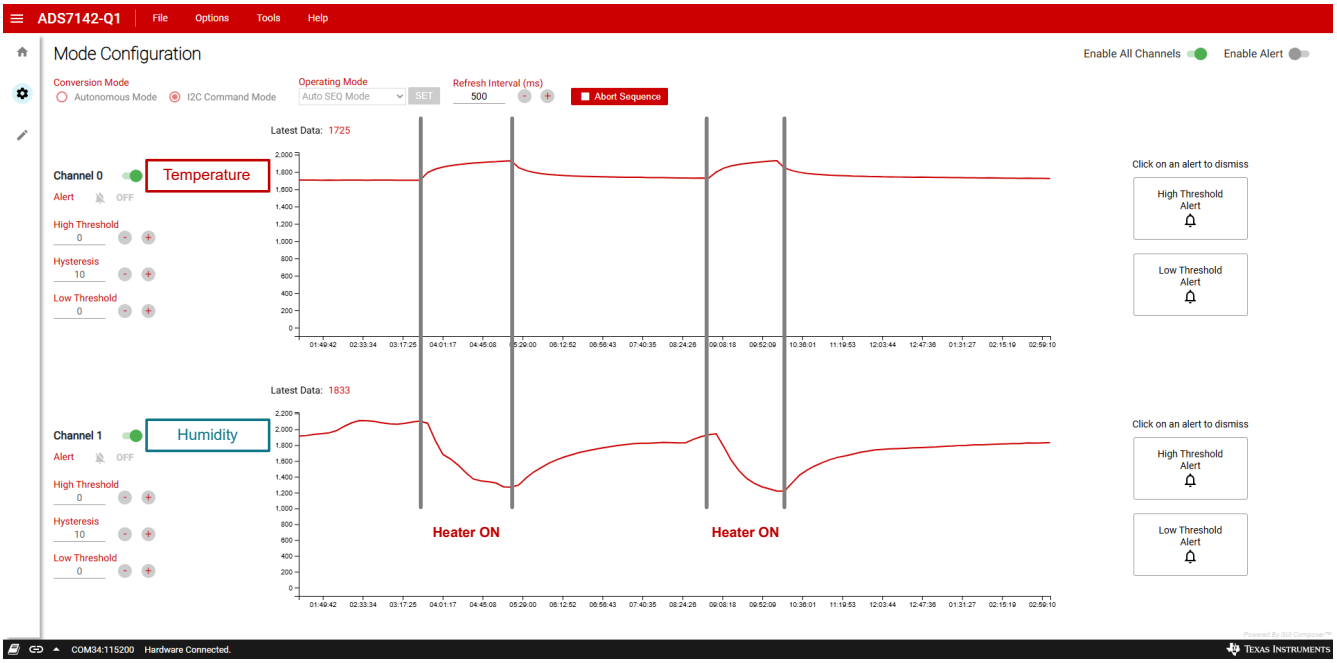

图 3-6. 使用 BOOSTXL-ADS7142-Q1 EVM 进行 HDC3120 输出解读

下面的计算利用最近收集的温度和湿度数据点。

温度：

ADC 温度输出（通道 0）：1725

HDC3120 至 ADS7142 温度转换公式：

$$\mathrm{T}\left(^{\circ}\mathrm{C}\right) = -66.875 + 218.75 \times \left(\frac{V_{TEMP}}{V_{DD}}\right) = -66.875 + 218.75 \times \left(\frac{TEMP_{ADC}}{2^{12}}\right) \quad (14)$$

$$\mathrm{T}\left(^{\circ}\mathrm{F}\right) = -88.375 + 393.75 \times \left(\frac{V_{TEMP}}{V_{DD}}\right) = -88.375 + 393.75 \times \left(\frac{TEMP_{ADC}}{2^{12}}\right) \quad (15)$$

使用 方程式 14，转换的温度结果为：25.2°C

湿度：

ADC 湿度输出（通道 1）：1833

HDC3120 至 ADS7142 湿度转换公式：

$$\%\,\mathrm{RH} = -12.5 + 125 \times \left(\frac{V_{RH}}{V_{DD}}\right) = -12.5 + 125 \times \left(\frac{RH_{ADC}}{2^{12}}\right) \quad (16)$$

使用 方程式 16，转换的湿度结果为：43.4%RH

4 总结

德州仪器 (TI) 提供了不断增长的湿度传感器系列，这些传感器可为终端系统提供关键环境信息。了解如何对 TI 的湿度传感器进行适当编程可以帮助设计人员在先进系统中实现最佳性能。为每个湿度传感器系列提供的代码可用作参考，帮助工程师了解该过程并避免常见的调试问题。有关传感器应用和器件特性的更多信息，请参阅下面链接的文档。

5 开发支持和文档

5.1 软件支持

如需使用基于 Arduino™ 的控制器进行快速原型设计，请访问 TI 的 GitHub™ 环境传感器存储库以开始设计。此存储库提供所有可用湿度传感器的示例代码。

如需更深入的基于 C 语言的驱动程序级支持，请访问 TI 基于 GUI 的代码生成器 ASC Studio 以开始设计。

如需其他帮助，请访问 TI E2E 传感器支持论坛。

5.2 参考资料

德州仪器 (TI)，*如何调试 RH 传感器中的 RH 精度问题*，应用手册。

- 提供了有关 RH 精度误差的解释和预防策略。

德州仪器 (TI)，*基于湿度传感器的汽车电子产品进水监测*，应用手册。

- 提供了一种基于湿度传感器的检测方法，利用 HDC3020 传感器快速识别密封或通风电子外壳中的进水情况。

德州仪器 (TI)，*HDC3x 器件用户指南*，用户指南。

- HDC3x 系列传感器的有用存储和处理指南。

德州仪器 (TI)，*HDC2x 器件用户指南*，用户指南。

- HDC2x 系列传感器的有用存储和处理指南。

德州仪器 (TI)，*温度和湿度传感器的 NIST 可追溯性*，产品概述

- 重点介绍了 NIST 可追溯器件，并讲解了在现代应用中的重要性。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月