



Harry Gill

ABSTRACT

Relative humidity (RH) sensors play a critical role in today’s electronic systems, supporting accurate environmental monitoring in a wide range of applications—from server rooms and industrial automation, to electric vehicles and smart infrastructure. By providing real-time data on ambient conditions, these sensors help protect sensitive systems, improve performance, and maintain overall reliability. This Application Note provides practical guidance for selecting and integrating Texas Instruments’ humidity sensors across three product generations: HDC1x (first generation), HDC2x (second generation), and HDC3x (third and latest generation). While each device family offers temperature and humidity measurement capabilities, each series features unique interface protocols and configuration options. The content presented in this application note is intended to simplify the process of evaluating and implementing TI’s humidity sensors for a given system design.

Note

Unless specified otherwise, this application note’s reference to *digital interface/protocol* strictly pertains to I2C-based digital communication. This application note will focus on code examples using the Arduino™ platform — a C-based open-source prototyping platform ideal for its simplicity and quick implementation. Additional C code is available through links provided at the end of this document.

Furthermore, any use of the letter ‘x’ at the end of a device name such as HDC302x or HDC2x means the following explanation applies to all applicable variants from the ‘x’ onwards:

HDC1x = [HDC1010](#) / [HDC1080](#)

HDC2x = [HDC2010](#) / [HDC2021](#) / [HDC2022](#) / [HDC2080](#)

HDC3x = [HDC3020](#) / [HDC3021](#) / [HDC3022](#) / [HDC3120](#)

Table of Contents

1 Introduction	2
2 Digital I2C Interface Overview	3
2.1 Register Map Protocol.....	3
2.1.1 A Quick Overview of I2C Register Map Protocol.....	3
2.1.1.1 HDC1x.....	4
2.1.1.2 HDC2x.....	6
2.2 Command Protocol.....	12
2.2.1 HDC302x.....	12
2.2.1.1 Interfacing in Trigger-On Demand Mode (One-Shot).....	14
2.2.1.2 Interfacing in Auto Measurement Mode (AMM).....	16
How to Check Measurement Data With CRC.....	16
3 Analog Interface Overview	18
3.1 HDC3120.....	18
4 Summary	24
5 Development Support and Documentation	24
5.1 Software Support.....	24
5.2 References.....	24

Trademarks

Arduino™ is a trademark of Arduino AG.

GitHub™ is a trademark of GitHub, Inc.
 BoosterPack™ is a trademark of Texas Instruments.
 All trademarks are the property of their respective owners.

1 Introduction

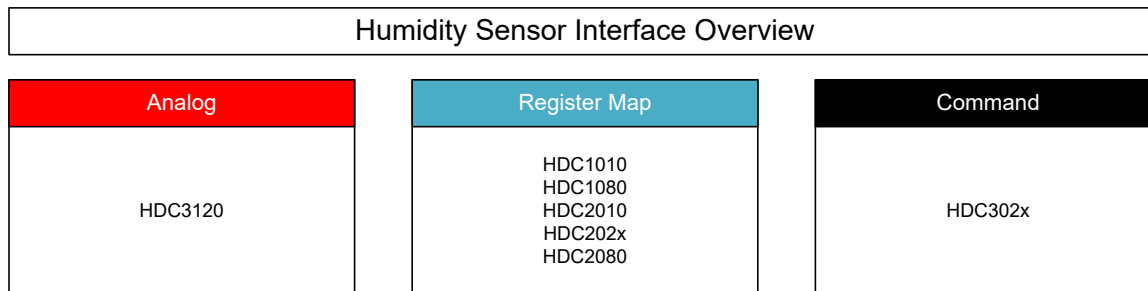


Figure 1-1. Humidity Sensor Interface Overview

Texas Instruments offers a portfolio of humidity sensors with two main interface types: analog ratiometric and digital I2C communication. These interface types can be broken up into three groups:

1. Digital I2C – Register Map Protocol

- Communication is register-based (similar to TI's Temperature sensors).
- Measurements are triggered and read by writing to and reading from specific register addresses.
- Devices: HDC1x and HDC2x (ex. HDC1080, HDC2022)

2. Digital I2C – Command-Based Protocol

- Communication is based on command sequences.
- The host sends a command to initiate a measurement and then retrieves the result using a separate read command.
- Devices: HDC302x (for example, HDC3020, HDC3022)

3. Analog Output – Ratiometric Voltage

- Sensor outputs voltage signals proportional to temperature and humidity.
- These can be fed directly into an analog system or digitized using an external ADC.
- Device: HDC3120

Each interface type requires a different approach to reading sensor data. The following sections detail these methods and provide code examples for Arduino-enabled microcontrollers.

2 Digital I2C Interface Overview

TI currently offers digital humidity sensors in two I2C interface styles: register map or command-based access. The following sections explain the general procedure for programming sensors from each digital humidity sensor family.

2.1 Register Map Protocol

Both HDC1x and HDC2x sensor families use a register map-based digital interface, where temperature and humidity measurements are initiated and read by writing to specific registers.

While the communication concept is similar, the two families differ in functionality and configuration options.

The **HDC2x series** provides a more advanced feature set including:

- Alert Functionality
- Data ready or interrupt pin support
- Two Measurement Modes:
 - Trigger-On Demand for user-controlled, real-time sampling
 - Auto Measurement Mode (AMM) for low-power, periodic sampling
- Split registers for Temperature MSB/LSB (Temperature High or Low) and Humidity MSB/LSB (Humidity High or Low)

The **HDC1x series** is designed for simpler implementations. This supports only Trigger-On Demand and uses a basic register structure with one register each for temperature and humidity measurements.

In summary, the HDC2x series is recommended over the HDC1x for designs requiring advanced features, configurable measurement modes, or low-power operation. The HDC1x series is designed for applications where a straightforward, digital I2C interface and simpler coding requirements are sufficient.

2.1.1 A Quick Overview of I2C Register Map Protocol

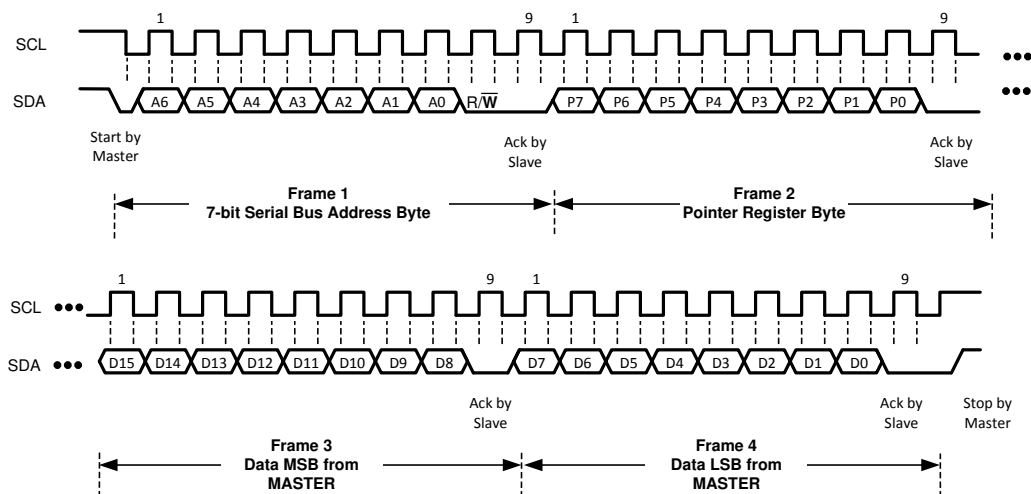


Figure 2-1. HDC1080 Data Frame Example (Configuration Register)

In I2C-based sensors, the register map is a structured table of memory locations (registers) that define how the device is controlled and accessed. Each register has a unique address that the host can read from or write to, allowing direct interaction with the configuration of the sensor, status bits, and data. This organized layout makes it straightforward to adjust settings, retrieve measurements, and monitor flag bits. For the HDC1x and HDC2x devices, the data structure looks similar to that in Figure 2-1, where after sending the address byte, the controller must send the pointer register byte before the controller can read data from the sensor.

For more information on I2C, see the following document on [I2C Basics](#).

2.1.1.1 HDC1x

TI's first-generation HDC1x devices all share the same register map (as illustrated in [Table 2-1](#)). Therefore, the following explanation applies across the HDC1x family.

Table 2-1. HDC1x Register Map

Pointer	Name	Reset value	Description
0x00	Temperature	0x0000	Temperature measurement output
0x01	Humidity	0x0000	Relative Humidity measurement output
0x02	Configuration	0x1000	HDC1080 configuration and status
0xFB	Serial ID	device dependent	First 2 bytes of the serial ID of the part
0xFC	Serial ID	device dependent	Mid 2 bytes of the serial ID of the part
0xFD	Serial ID	device dependent	Last byte bit of the serial ID of the part
0xFE	Manufacturer ID	0x5449	ID of Texas Instruments
0xFF	Device ID	0x1050	ID of the device

To begin, users must write a 16-bit value to the register in [Table 2-2](#) (0x02) to define the measurement sequence – temperature, humidity or both sequentially.

Table 2-2. HDC1x Configuration Register (0x02)

NAME	Bits	DESCRIPTION		
RST	[15]	Software reset bit	0	Normal Operation, this bit self clears
			1	Software Reset
Reserved	[14]	Reserved	0	Reserved, must be 0
HEAT	[13]	Heater	0	Heater Disabled
			1	Heater Enabled
MODE	[12]	Mode of acquisition	0	Temperature or Humidity is acquired.
			1	Temperature and Humidity are acquired in sequence, Temperature first.
BTST	[11]	Battery Status	0	Battery voltage > 2.8V (read only)
			1	Battery voltage < 2.8V (read only)
TRES	[10]	Temperature Measurement Resolution	0	14 bit
			1	11 bit
HRES	[9:8]	Humidity Measurement Resolution	00	14 bit
			01	11 bit
			10	8 bit
Reserved	[7:0]	Reserved	0	Reserved, must be 0

In the example below, the HDC1x is configured to measure both temperature and humidity in sequence by writing 0x10 (MSB) and 0x00 (LSB) to the register. [Figure 2-2](#) illustrates the register bits that are set.

When the HDC1x is set to output temperature and humidity sequentially, users must initiate a 4-byte read from the Temperature Register (0x00).

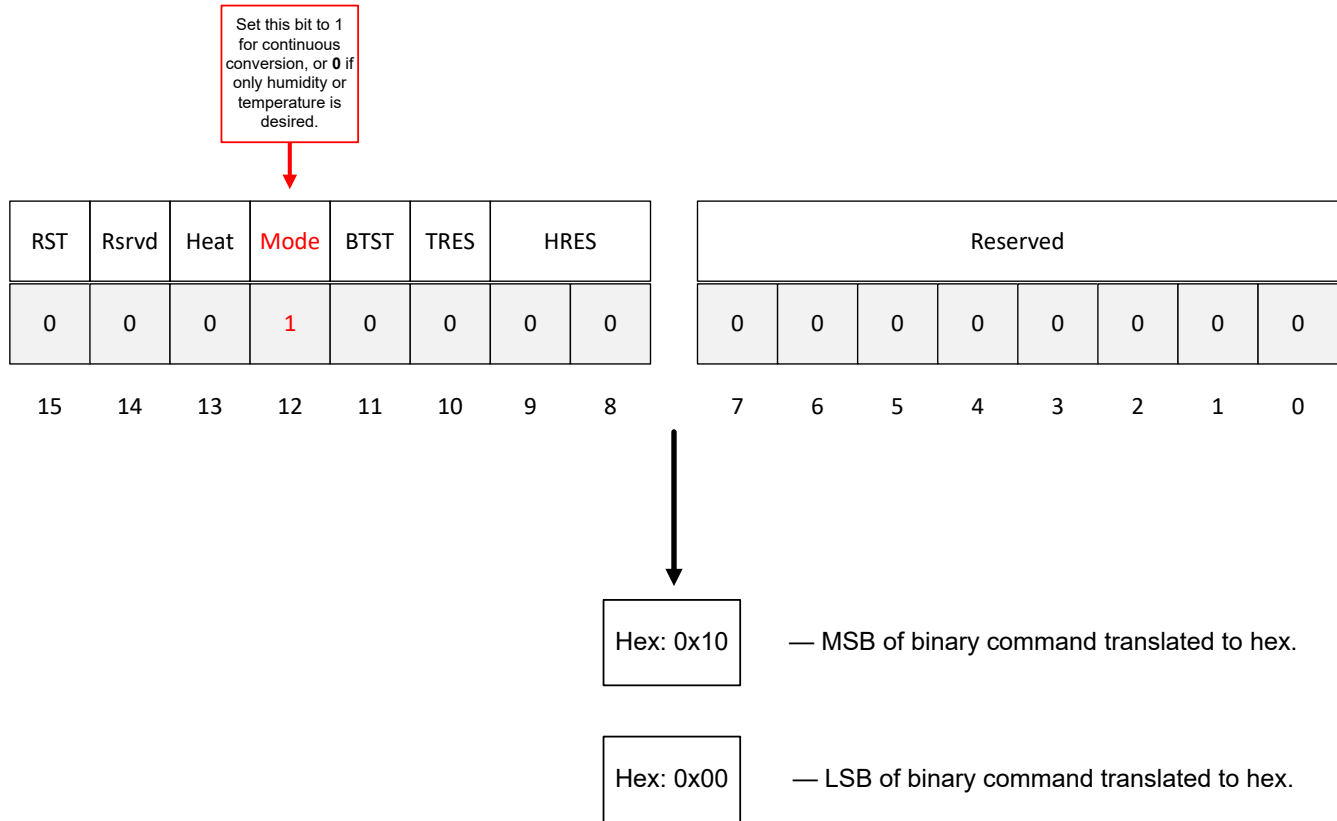


Figure 2-2. Configuration Register Bits for Temperature and Humidity Acquisition

The code for setting the configuration is as follows:

```

wire.beginTransaction(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();
  
```

Next, trigger the measurement process by writing to the device address (0x40).

```

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

Since temperature and humidity are measured in sequence, a 4-byte read from register 0x00 is required. The first two bytes correspond to the temperature while the next two bytes correspond to humidity data.

```

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();
}
  
```

Finally, apply the standard conversion formulas from the HDC1x data sheet:

```
// 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;
```

This Arduino example demonstrates how to read and store temperature and humidity data from the HDC1x sensor when measurements are acquired sequentially. Once the raw data is stored, this can be converted to physical temperature and humidity values using the formulas provided in the HDC1x data sheet.

A complete working sample is available on the TI GitHub repository for environmental sensors [here](#).

2.1.1.2 HDC2x

The HDC2x family ([HDC2010](#), [HDC2021](#), [HDC2022](#), [HDC2080](#)) also uses a register map-based digital interface, similar to the HDC1x series. As illustrated in [Table 2-3](#), all HDC2x devices share a common register layout, and the following procedure applies across the family. This section outlines how to interface with these devices in both Trigger-On Demand (One-Shot) and Auto Measurement (Continuous Conversion) modes.

Table 2-3. HDC2x Register Map

Pointer	NAME	RESET VALUE	DESCRIPTION
0x00	TEMPERATURE LOW	0x00	Temperature [7:0]
0x01	TEMPERATURE HIGH	0x00	Temperature [15:8]
0x02	HUMIDITY LOW	0x00	Humidity [7:0]
0x03	HUMIDITY HIGH	0x00	Humidity [15:8]
0x04	INTERRUPT/DRDY	0x00	DataReady and interrupt configuration
0x05	TEMPERATURE MAX	0x00	Maximum measured temperature (Not supported in Auto Measurement Mode)
0x06	HUMIDITY MAX	0x00	Maximum measured humidity (Not supported in Auto Measurement Mode)
0x07	INTERRUPT ENABLE	0x00	Interrupt Enable
0x08	TEMP_OFFSET_ADJUST	0x00	Temperature offset adjustment
0x09	HUM_OFFSET_ADJUST	0x00	Humidity offset adjustment
0x0A	TEMP_THR_L	0x00	Temperature Threshold Low
0x0B	TEMP_THR_H	0xFF	Temperature Threshold High
0x0C	RH_THR_L	0x00	Humidity threshold Low
0x0D	RH_THR_H	0xFF	Humidity threshold High
0x0E	RESET&DRDY/INT CONF	0x00	Soft Reset and Interrupt Configuration
0x0F	MEASUREMENT CONFIGURATION	0x00	Measurement configuration
0xFC	MANUFACTURER ID LOW	0x49	Manufacturer ID Low
0xFD	MANUFACTURER ID HIGH	0x54	Manufacturer ID High
0xFE	DEVICE ID LOW	0xD0	Device ID Low
0xFF	DEVICE ID HIGH	0x07	Device ID High

Note

For the following HDC2x examples, an HDC2010 configured to address 0x40 (ADDR pin connected to GND) was used, however a global variable is available to conveniently adjust the address according to your setup.

One key difference from the HDC1x series is the HDC2x devices use separate 8-bit registers for storing the most significant and least significant bits of each measurement:

- Temperature: TEMP_LOW (LSB), TEMP_HIGH (MSB)
- Humidity: HUM_LOW (LSB), HUM_HIGH (MSB)

In addition to these data registers, the HDC2x series includes a **Measurement Configuration Register**, which allow users to define measurement parameters.

The measurement process begins by writing to the Configuration Register, which controls key functions such as (but not limited to):

- Heater Enable (HEAT_EN)
- Auto Measurement Mode (AMM)
- Soft Reset (SOFT_RES)

Subsequent steps involve setting measurement parameters and initiating conversions, which can be detailed in the following subsections.

2.1.1.2.1 Interfacing in Trigger-On Demand Mode

In this example, an HDC2010 sensor is configured for **Trigger-On Demand (One-Shot) mode**, with Auto Measurement Mode (AMM) disabled. The procedure begins by writing to the Configuration Register (0x0E) to set the device in Trigger-On Demand mode. An illustration of the Configuration Register setup is provided in [Figure 2-3](#).

Table 2-4. Configuration Register (0x0E)

BIT	FIELD	TYPE	RESET	DESCRIPTION
7	SOFT_RES	R/W	0	0 = Normal Operation mode, this bit is self-clear 1 = Soft Reset EEPROM value reload and registers reset
[6:4]	AMM[2:0]	R/W	000	Auto Measurement Mode (AMM) 000 = Disabled. Initiate measurement via I2C 001 = 1/120Hz (1 samples every 2 minutes) 010 = 1/60Hz (1 samples every minute) 011 = 0.1Hz (1 samples every 10 seconds) 100 = 0.2Hz (1 samples every 5 second) 101 = 1Hz (1 samples every second) 110 = 2Hz (2 samples every second) 111 = 5Hz (5 samples every second)
3	HEAT_EN	R/W	0	0 = Heater off 1 = Heater on
2	DRDY/INT_EN	R/W	0	DRDY/INT_EN pin configuration 0 = High Z 1 = Enable
1	INT_POL	R/W	0	Interrupt polarity 0 = Active Low 1 = Active High
0	INT_MODE	R/W	0	Interrupt mode 0 = Level sensitive 1 = Comparator mode

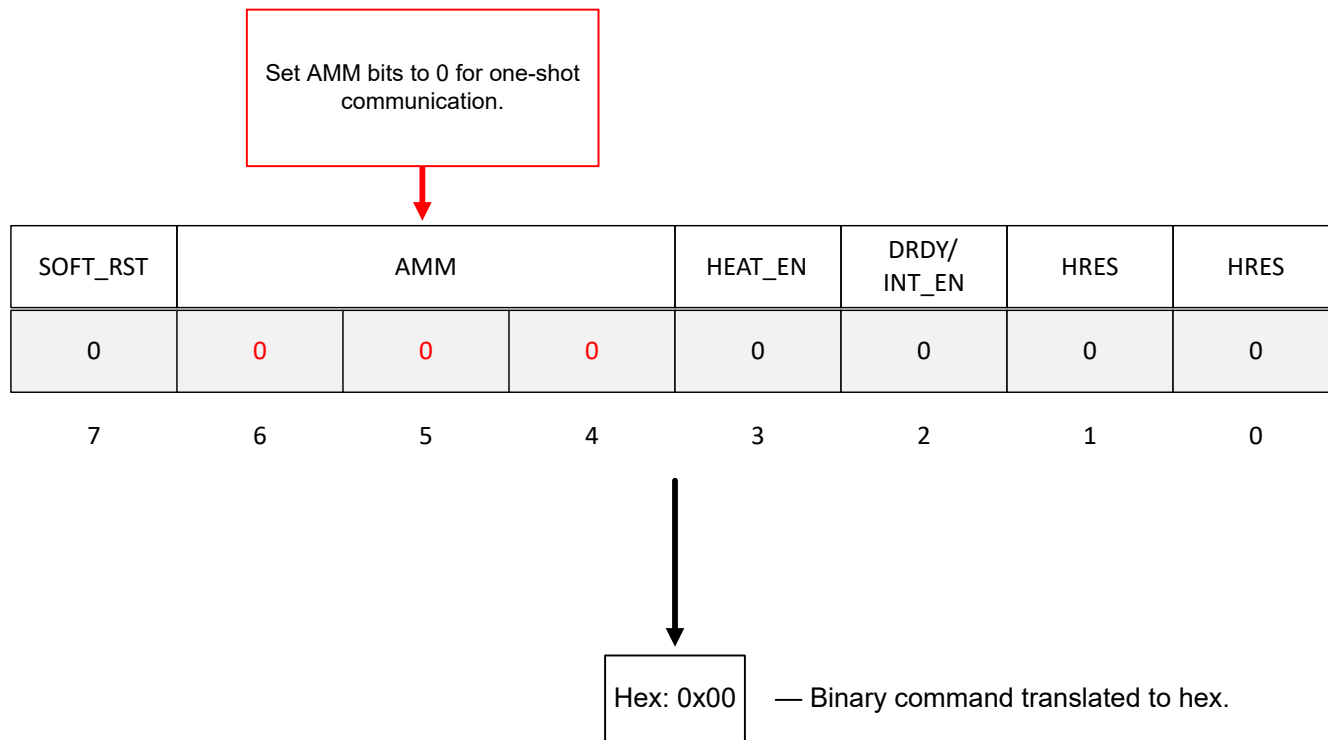


Figure 2-3. Configuration Register for Trigger-On Demand (One-Shot)

```

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

Next, configure the Measurement Configuration Register (0x0F) as shown in [Figure 2-4](#).

The Measurement Configuration Register defines the following:

- Temperature and Humidity resolution (TRES and HRES)
- Measurement Type (temperature only, humidity only, or both)
- Measurement Trigger (MEAS_TRIG)

Table 2-5. Measurement Configuration Register (0x0F)

BIT	FIELD	TYPE	RESET	DESCRIPTION
7:6	TRES[1:0]	R/W	00	Temperature resolution 00: 14 bit 01: 11 bit 10: 9 bit 11: NA
5:4	HRES[1:0]	R/W	00	Humidity resolution 00: 14 bit 01: 11 bit 10: 9 bit 11: NA
3	RES	R/W	0	Reserved
2:1	MEAS_CONF[1:0]	R/W	00	Measurement configuration 00: Humidity + Temperature 01: Temperature only 10: NA 11: NA

Table 2-5. Measurement Configuration Register (0x0F) (continued)

BIT	FIELD	TYPE	RESET	DESCRIPTION
0	MEAS_TRIG	R/W	0	Measurement trigger 0: no action 1: Start measurement Self-clearing bit when measurement completed

Figure 2-4 provides an illustration of the Measurement Configuration Register.

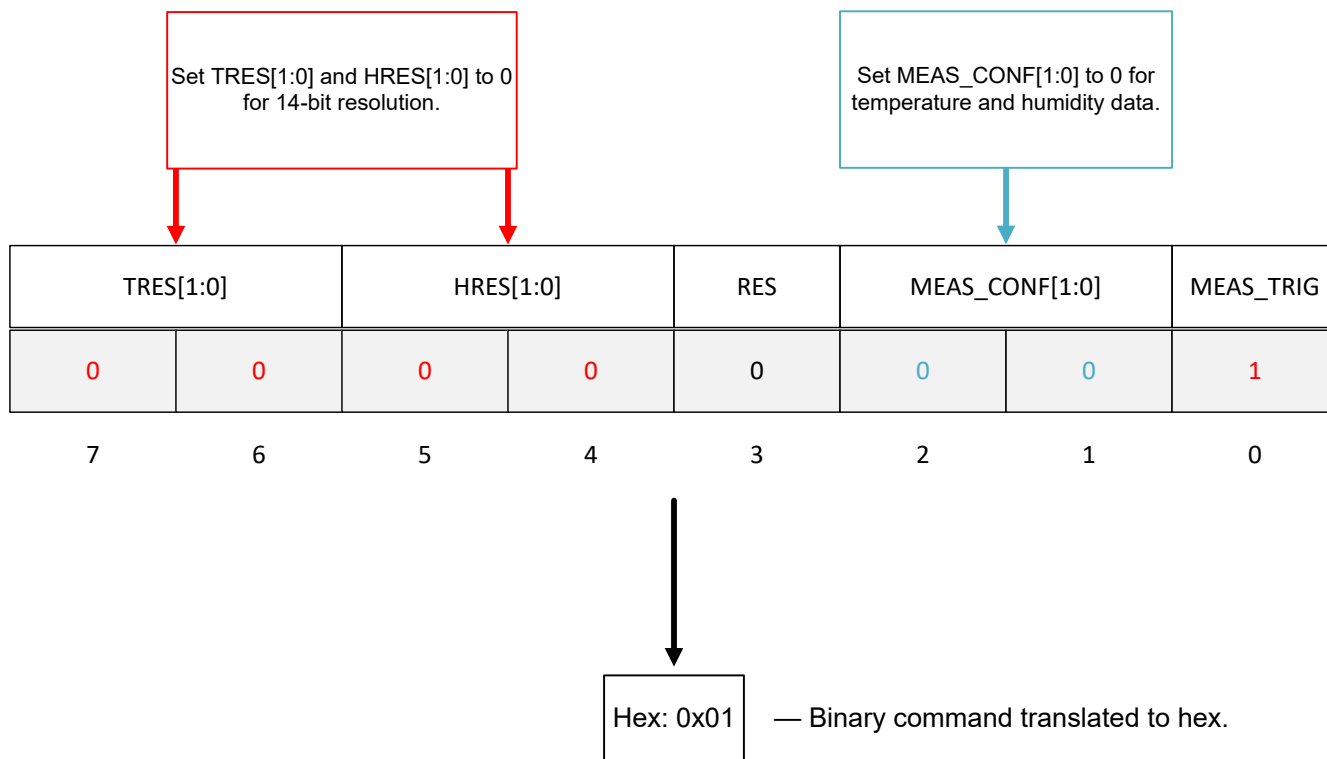


Figure 2-4. Measurement Configuration Register Setup

```
wire.beginTransaction(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();
```

In Trigger-On Demand mode, this sequence can be placed inside a loop for periodic polling, depending on the system's needs.

To read the temperature and humidity measurements for the HDC2x, users can choose between two possible methods. The first method is to read/store the temperature and humidity bytes separately using the TEMP_LOW/HUM_LOW and TEMP_HIGH/HUM_HIGH and combine the MSB and LSB bits into one 16-bit value, or burst read multiple bytes in one communication frame as illustrated in the following code for humidity data acquisition:

Reading Measurement Data

The HDC2x stores measurement results across two 8-bit registers:

- Humidity: HUM_LOW (0x02), HUM_HIGH (0x03)
- Temperature: TEMP_LOW (0x00), TEMP_HIGH (0x01)

One approach is to read each byte separately and combine them:

```
uint16_t getHum() {
    // RH LSB Acquisition

    wire.beginTransaction(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.beginTransaction(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;
}
```

However, a more efficient method reads both bytes in a burst from the LSB registers:

```
uint16_t getHum2() {
    wire.beginTransaction(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;
}
```

This approach leverages the HDC2x's internal pointer behavior: after reading the LSB from the HUMIDITY_LOW register, the pointer auto-increments to the HUMIDITY_HIGH register for the MSB. The same mechanism applies for reading temperature.

Visit the following [link](#) to view the full sample code for the HDC2x in Trigger-On Demand Mode.

2.1.1.2.2 Interfacing Using Auto Measurement Mode (AMM)

This section outlines how to configure HDC2x devices to operate in Auto Measurement Mode (AMM) and highlights key differences compared to Trigger-On Demand mode.

In AMM, the device automatically performs measurements at a user-defined sampling frequency, eliminating the need for manual measurement triggers from the MCU. Unlike Trigger-On Demand, where each measurement must be initiated manually, AMM requires only a single trigger to start periodic conversions.

In this example, the HDC2010 is configured to sample once every 5 seconds (0.2Hz). This is accomplished by writing the appropriate setting to the Configuration Register (0x0E). An illustration of the Configuration Register setup is provided in [Figure 2-5](#).

Set AMM bits to 1 0 0 for AMM with 1 sample every 5 seconds.



SOFT_RST	AMM			HEAT_EN	DRDY/ INT_EN	HRES	HRES
0	1	0	0	0	0	0	0
7	6	5	4	3	2	1	0



Hex: 0x40

— Binary command translated to hex.

Figure 2-5. Configuration Register for Auto Measurement Mode (AMM)

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

The Measurement Configuration Register uses the same configuration explained in the [Trigger-On Demand section](#).

Visit the following [link](#) to view the full sample code for the HDC2x in Auto Measurement Mode.

2.2 Command Protocol

2.2.1 HDC302x

The **HDC302x** family introduces a command-based interface, marking a departure from the register map-based scheme used in prior generations. Instead of writing to specific registers, HDC302x devices respond to well-defined command codes to initiate measurements, configure settings, or retrieve data. [Table 2-6](#) shows an example of supported commands by the HDC302x devices.

This approach simplifies the interface and reduces the number of required I2C transactions, particularly in applications where only basic measurements are needed.

The forthcoming sections demonstrate how to interface with the HDC302x in both Trigger-On Demand and Auto Measurement modes, providing examples of how command-based communication is implemented in each case.

Table 2-6. HDC302x Command Table Snippet

HEX CODE (MSB)	HEX CODE (LSB)	COMMAND	COMMAND DETAIL
24	00	Trigger-On Demand Mode Single Temperature (T) Measurement and Relative Humidity (RH) Measurement	Low Power Mode 0 (lowest noise)
24	0B		Low Power Mode 1
24	16		Low Power Mode 2
24	FF		Low Power Mode 3 (lowest power)
20	32	Auto Measurement Mode 1 measurement per 2 seconds.	Low Power Mode 0 (lowest noise)
20	24		Low Power Mode 1
20	2F		Low Power Mode 2
20	FF		Low Power Mode 3 (lowest power)
21	30	Auto Measurement Mode 1 measurement per second.	Low Power Mode 0 (lowest noise)
21	26		Low Power Mode 1
21	2D		Low Power Mode 2
21	FF		Low Power Mode 3 (lowest power)
22	36	Auto Measurement Mode 2 measurements per second.	Low Power Mode 0 (lowest noise)
22	20		Low Power Mode 1
22	2B		Low Power Mode 2
22	FF		Low Power Mode 3 (lowest power)
23	34	Auto Measurement Mode 4 measurements per second.	Low Power Mode 0 (lowest noise)
23	22		Low Power Mode 1
23	29		Low Power Mode 2
23	FF		Low Power Mode 3 (lowest power)
27	37	Auto Measurement Mode 10 measurements per second.	Low Power Mode 0 (lowest noise)
27	21		Low Power Mode 1
27	2A		Low Power Mode 2
27	FF		Low Power Mode 3 (lowest power)
2C	06	Trigger-On Demand Mode Single Temperature (T) Measurement and Relative Humidity (RH) Measurement	Low Power Mode 0 (lowest noise)
2C	0D		Low Power Mode 1
2C	10		Low Power Mode 2

Table 2-6. HDC302x Command Table Snippet (continued)

HEX CODE (MSB)	HEX CODE (LSB)	COMMAND	COMMAND DETAIL
30	93	Auto Measurement Mode	Exit, then return to Trigger-on Demand Mode.
E0	00		Measurement Readout of T and RH (Note: if RH and T are not updated, data read outs all FFs)
E0	01		Measurement Readout of RH only
E0	02		Measurement History Readout of Minimum T.
E0	03		Measurement History Readout of Maximum T.
E0	04		Measurement History Readout of Minimum RH.
E0	05		Measurement History Readout of Maximum RH.

2.2.1.1 Interfacing in Trigger-On Demand Mode (One-Shot)

This section demonstrates how to configure and read measurements from an HDC302x device using highest resolution settings. All example codes utilize a global variable for the I2C address and can be easily changed based off the user's device configuration.

Note

The HDC302x sensor used in this example is configured for address 0x44 (ADDR and ADDR1 pin connected to GND), but a global variable is available in the code to conveniently adjust the address according to your layout.

Before measurements can be read, the device must first be configured. This is done by sending a specific command sequence to the HDC302x. An illustration of the configuration is shown in [Figure 2-6](#).

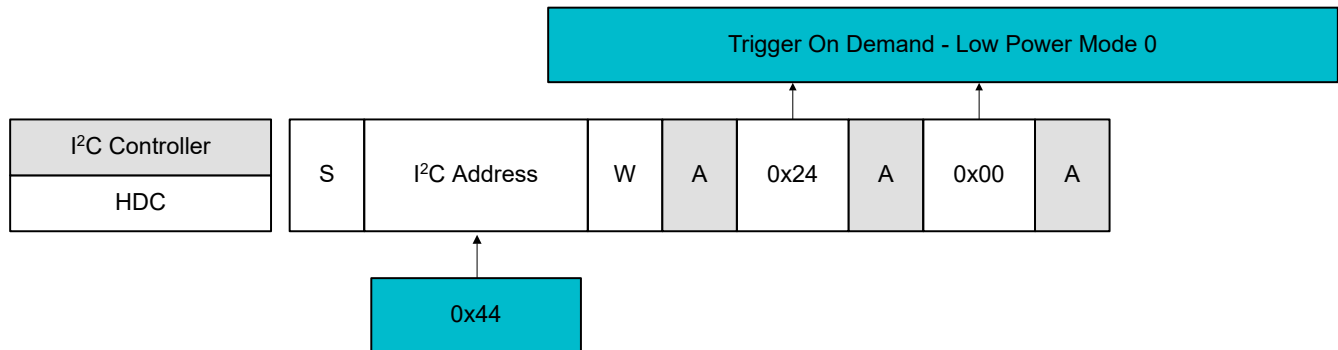


Figure 2-6. Trigger-On Demand Command Selection

```

wire.beginTransaction(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
    
```

A delay is required to ensure the measurement conversion has completed before initiating a read. For this example, a 15ms delay was used since the measurement configuration is based on the highest resolution and repeatability. Engineers should consult the data sheet prior to setting the appropriate delay, though a minimum delay of 15ms should be implemented.



Figure 2-7. HDC302x Trigger-On Demand Communication Structure

```

void loop() {
    float humidity;
    float temp;

    // send device command for highest repeatability
    wire.beginTransaction(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)
}

```

Data Conversion Functions

Temperature and humidity are calculated using the formulas provided in the HDC302x data sheet:

```

// 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 celsius

    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;
}

```

Note on Buffer Structure

The six-byte buffer `HDC_DATA_BUFF` contains:

- Bytes 0-1: Raw temperature data (MSB, LSB)
- Byte 2: Temperature CRC (optional)
- Bytes 3-4: Raw humidity data (MSB, LSB)
- Byte 5: Humidity CRC (optional)

Although this example does not use the CRC bytes, the bytes are included in the buffer for completeness and can be checked for data integrity if needed.

Visit the following [link](#) to view the full sample code for the HDC302x in Trigger-On Demand Mode.

2.2.1.2 Interfacing in Auto Measurement Mode (AMM)

This section outlines how to configure the HDC302x for AMM and explains the key differences compared to Trigger-On Demand mode.

The main distinction between Auto Measurement Mode and Trigger-On Demand is if users intend to read from the HDC302x sensor in a fixed interval, Auto Measurement Mode would be more suitable for its programmable output interval. Figure 2-8 illustrates the command sequence for programming an HDC302x to output a single measurement every second with the lowest noise and highest repeatability.

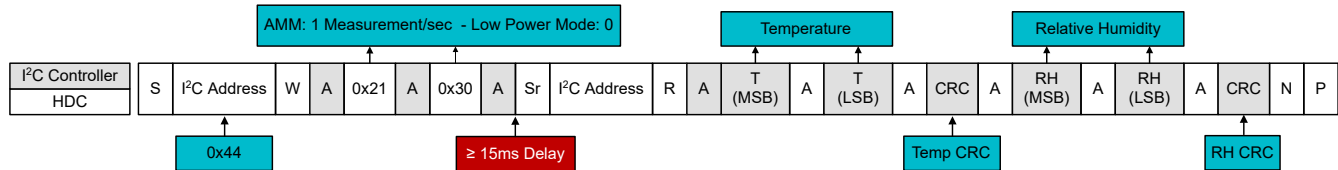


Figure 2-8. HDC302x Auto Measurement Mode (AMM) Communication Structure

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

    wire.beginTransaction(0x44);
    wire.write(0x21); //send MSB of command
    wire.write(0x30); //command LSB
    wire.endTransmission();
    delay(15); //wait 15ms before reading
}
}
```

After the device is configured for Auto Measurement Mode, and sufficient time has passed for a conversion to complete (one second in this case), the following function is used to request the stored measurement data:

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

    wire.beginTransaction(DEVICE_ADDR); // initiate communication
    wire.write(0xE0); // send MSB of read command
    wire.write(0x00); // send LSB of read command
    wire.endTransmission();
}
}
```

To continuously poll the device in AMM, you can issue the read command to retrieve the latest measurement data at the configured sampling rate (measurements per second).

A complete Arduino example demonstrating HDC302x operation in AMM – including device configuration, measurement polling, and data readout is available in the TI GitHub™ repository for environmental sensors [here](#).

How to Check Measurement Data With CRC

Performing CRC checks on the HDC302x's measurement output can be an important step to ensure data integrity and prevent false readings, particularly in critical applications such as medical devices, cold chain, or weather stations. This section provides a simple code example that can be readily implemented into the existing code for the HDC302x. This example specifically focuses on performing a CRC check on the temperature and humidity readings.

The HDC3020 follows the CRC-8 standard as it outputs a unique 8-bit CRC value for temperature and humidity measurements. This is illustrated above in Figure 2-8. Depending on whether the user wants to check humidity or temperature data, the algorithm below accepts a total of three bytes—MSB, LSB and CRC as a parameter and the total number of bytes sent. It then performs a calculation using the specified polynomial value from the datasheet, 0x31. Equation 1 illustrates the polynomial used for the CRC check calculation.

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

Once both measurement data and the CRC byte are processed, if the final value is 0x00, the CRC check has passed and the program proceeds to output the measurement readings; else, it outputs an error message in the console.

```
// 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
}
```

This function can then be called while temperature or humidity is read to check data integrity.

```
Humidity Read Example w/ CRC Check:

uint8_t humCheck[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){

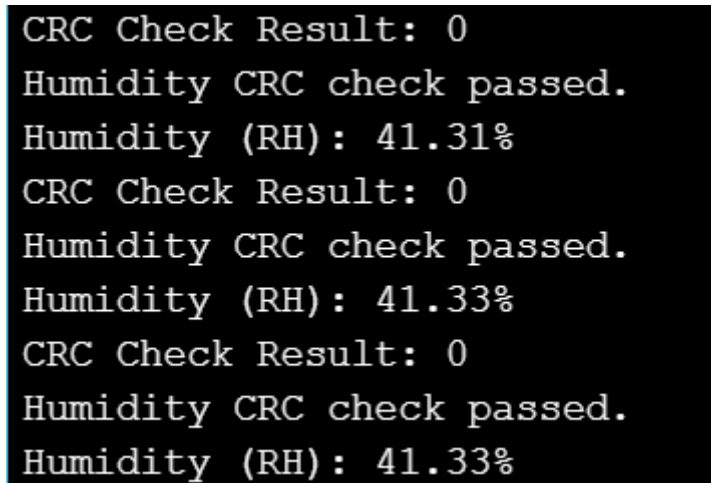
Serial.println("Humidity CRC check passed.");
humidity = getHum(HDC_DATA_BUFF);
Serial.print("Humidity (RH): ");
Serial.print(humidity);
Serial.println("%");

} else {

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

}
```

A sample output is provided below in [Figure 2-9](#). The same method can be implemented for checking temperature data integrity.



```
CRC Check Result: 0
Humidity CRC check passed.
Humidity (RH): 41.31%
CRC Check Result: 0
Humidity CRC check passed.
Humidity (RH): 41.33%
CRC Check Result: 0
Humidity CRC check passed.
Humidity (RH): 41.33%
```

Figure 2-9. CRC Check Example Output

An example for checking Alert CRC in C can be found in TI's GUI-based code generator, [ASC Studio](#).

3 Analog Interface Overview

3.1 HDC3120

The [HDC3120](#) is Texas Instruments' first humidity sensor with an analog ratiometric output. The HDC3120 provides continuous voltage signals corresponding to temperature and relative humidity, making this well-designed for low-noise, analog front-end systems.

Unlike digital sensors which require communication protocols, the HDC3120 allows direct access to sensor outputs without writing I2C commands or configuring registers. This simplifies integration in systems where an analog-to-digital converter (ADC) is already available.

To interpret the output signals, users can apply the appropriate conversion equations (provided below) to translate voltage levels into temperature (°C) and relative humidity (%RH).

An important characteristic of the HDC3120 is the ratiometric behavior. The output voltages for temperature and humidity scale linearly with the device's supply voltage (VDD), which also serves as the internal reference. This design provides immunity from noise and drift in the power supply, enabling reliable measurements. This relationship is illustrated in the following graphs for both temperature and humidity output curves.

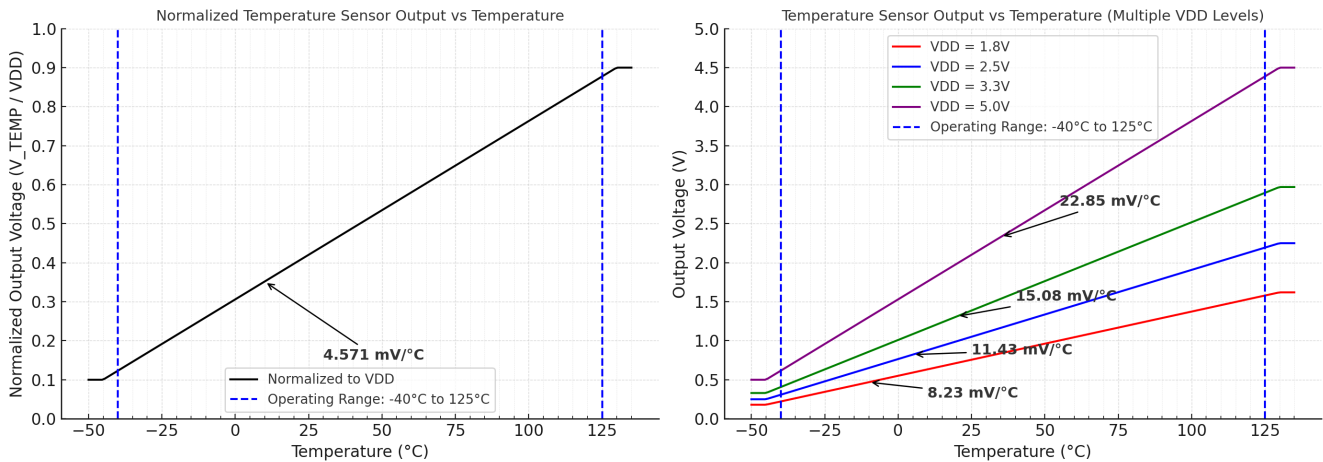


Figure 3-1. Ratiometric Temperature Output Profile and Conversion Equation

$$T(^{\circ}\text{C}) = 218.75 \times \frac{V_{\text{OUT}}}{V_{\text{DD}}} - 66.875 \tag{2}$$

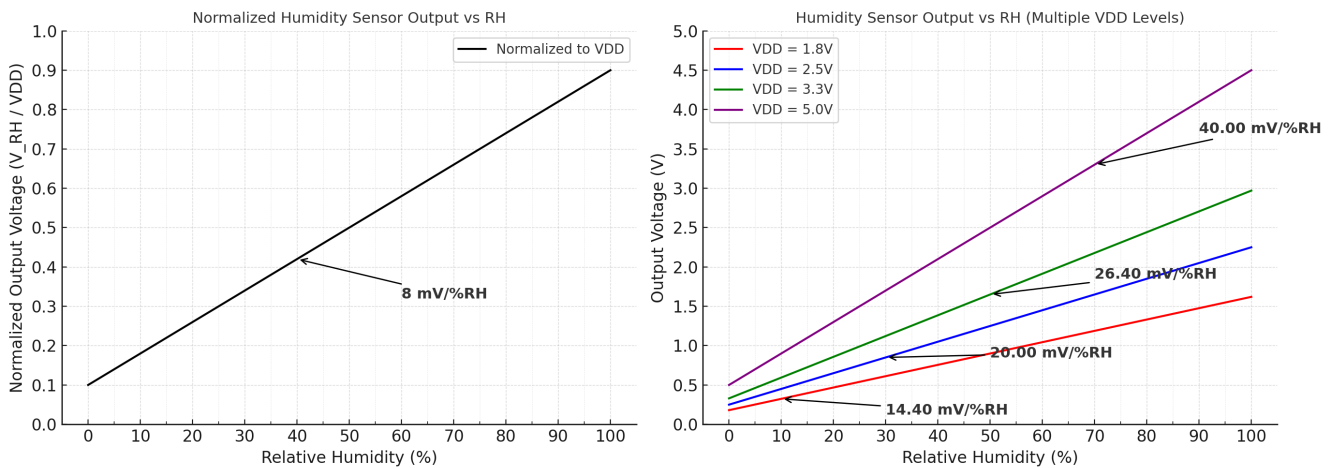


Figure 3-2. Ratiometric Relative Humidity Output Profile and Conversion Equation

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

Understanding the ratiometric nature of the HDC3120 output is critical – especially when interfacing with an ADC since the sensor’s output voltage directly scales with the supply voltage (VDD). If the chosen ADC also uses its supply as a reference, measurement consistency is preserved in the event noise or drift appears within the voltage supply. This alignment eliminates gain mismatch issues and helps maintain consistent sensor readings.

How to Choose an ADC for the HDC3120:

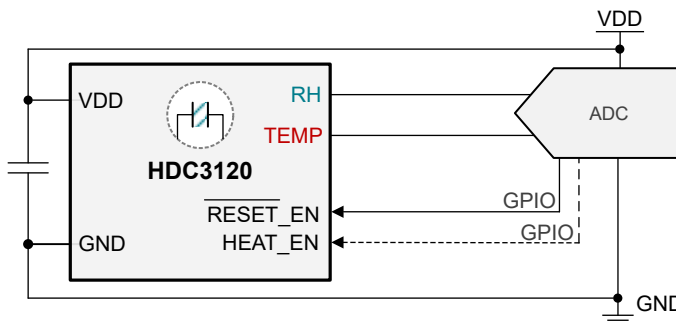


Figure 3-3. HDC3120 to ADC

Table 3-1 provides some ADCs for the HDC3120:

Table 3-1. ADCs for the HDC3120

ADC	Resolution	Ratiometric	Automotive Grade	Supply Range	When to Select
ADS7142	12-bit	Yes	Q100	1.65-3.6V	Integrated alert functionality for exceeding system limits; up to two single-ended inputs
ADS7138	12-bit	Yes	Q100	2.35V-5.5V	Heater/enable control through GPIO pins; up to eight single-ended inputs
ADS7066	16-bit	Yes	—	3V-5.5V	Heater/enable control via GPIO pins; up to eight single-ended inputs
ADS1015	12-bit	No	Q100	2V-5.5V	Programmable Gain Amplifier (PGA) for wider input voltage range; up to four single-ended inputs

When selecting an ADC for the HDC3120, the following process must be considered.

1. Determine the ADC’s least significant bit (LSB)
2. Calculate the HDC3120 temperature LSB
3. Calculate the HDC3120’s humidity LSB
4. Compare LSB Values

Choose an ADC LSB that is smaller than the HDC3120’s temperature LSB. The same consideration can be used for humidity LSB, however, since the HDC3120’s temperature sensor has a higher accuracy, it is used to determine the minimum LSB required for an ADC.

The ADS1015 cannot perform ratiometric measurements since it only has an internal reference, and offers no option for an external reference. Additionally, the LSB will need to be solved for using the following equation:

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

Where FSR (Full Scale Range) represents the scaling factor and V_{REF} is the ADC's reference voltage. In the case of the ADS1015, the gain is determined by setting of the programmable gain amplifier (PGA) using Table 7-1 in the ADS1015 datasheet.

Additional information on pairing the HDC3120 with an ADC can be found in section 8.2.2 in the [HDC3120 datasheet](#).

Example Scenario:

For this example, the BOOSTXL-ADS7142-Q1 EVM was paired with the HDC3120EVM. Both devices were powered using the same 3.3V supply. From here the LSB's of both devices can be compared using the steps above:

1. Determine the ADC's least significant bit (LSB) using the following equation:

$$\frac{FSR}{Resolution} = LSB_{ADC} \quad (5)$$

- a. Since the operating voltage is 3.3V and the ADC features a 12-bit resolution, the LSB is:

$$\frac{3.3V}{2^{12}} = 0.805mV \quad (6)$$

2. Calculate the HDC3120's Temperature LSB using the following equation:

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

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

The value for temperature ($T(^{\circ}C)$) is 0.1 to reflect the HDC3120's typical temperature sensing accuracy.

3. Calculate the HDC3120's Humidity LSB using [Equation 9](#):

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

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

The value for humidity (%RH) is 1.0 to reflect the HDC3120's typical humidity sensing accuracy.

4. Compare LSB Values

- a. Since the Temperature LSB size of the HDC3120 is 1.508mV at 3.3V, and the ADS7142 has an LSB of 0.805mV; this means users retains measurement precision of at least 1°C. If a higher degree of precision is desired, such as 0.5°C, a higher resolution ADC such as the [ADS7066](#) should be used.

In this example setup, the [HDC3120EVM](#) was connected to the [BOOSTXL-ADS7142-Q1 EVM](#). Both devices shared the same VDD rail. The setup procedure is as follows:

1. Connect the TEMP output from the HDC3120EVM to the AIN0 input on the BOOSTXL-ADS7142-Q1 EVM BoosterPack™ using a jumper wire as illustrated in [Figure 3-4](#).

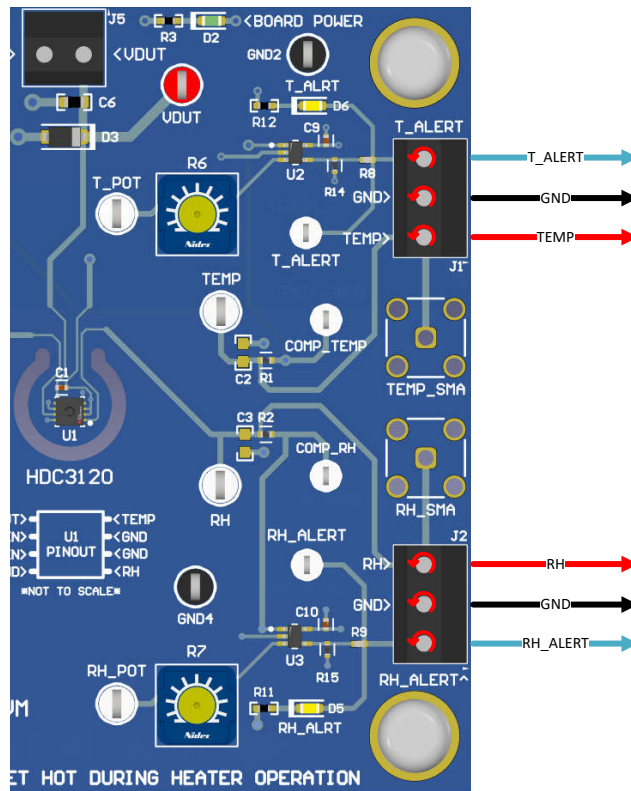


Figure 3-4. HDC3120EVM Outputs

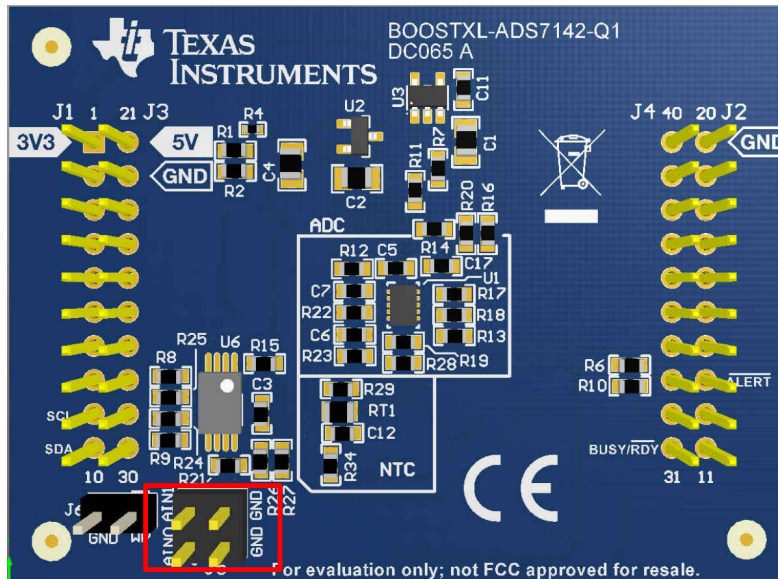


Figure 3-5. BOOSTXL-ADS7142-Q1 EVM BoosterPack™ Pin Connection Location

2. Connect RH output from the HDC3120EVM to AIN1 on the BOOSTXL-ADS7142-Q1 EVM BoosterPack™.
3. Launch the [ADS7142EVM GUI](#) and navigate to the homepage.
4. Click the gear icon to access the configuration menu, then under Conversion Modes, select I2C Command Mode.
5. Select *Auto SEQ Mode* from the *Operating Mode* drop-down menu and click the red *SET* button.
6. Once configured, click *Start Sequence* to begin capturing measurements in 12-bit decimal format.

Note

The equations for converting the HDC3120's temperature and humidity outputs below are specifically for the application with the 12-bit BOOSTXL-ADS7142-Q1 EVM.

To interpret the results, users can apply the HDC3120's conversion equations. Equation 11 and Equation 12 illustrate the reinterpretation of the ratios from the HDC3120's conversion equations. Equation 13 provides additional definitions for the variables.

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

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

- $TEMP_{ADC}$ → HDC3120 to ADC temperature decimal output (13)
- RH_{ADC} → HDC3120 to ADC relative humidity decimal output
- 2^{12} → represents the 12-bit resolution of the ADC outputs

Note

Due to the wiring configuration for this example, pin AIN0 (Temperature) represents Channel 0 in the GUI while pin AIN1 (Humidity) represents Channel 1. Make sure both are toggled on to receive output data.

Figure 3-6 plots temperature and humidity measurements of the HDC3120. Channel 0 (top graph) represents the temperature reading, while Channel 1 (bottom graph) represents the humidity reading. The latest data is provided next to "Latest Data" for each channel. The curves in the graph represent approximately 3-second intervals where HDC3120 heater was enabled for three seconds before being disabled via the heater enable switch on the HDC3120EVM. Therefore, a peak is observed for temperature each time the heater was enabled, while a trough (opposite peak) was observed for humidity as the device dries from heating (this is normal behavior within humidity sensors with integrated heating elements).



Figure 3-6. HDC3120 Output Interpretation With the BOOSTXL-ADS7142-Q1 EVM

The calculations below utilize the most recently collected data point for temperature and humidity.

Temperature:

ADC Temperature Output (Channel 0): 1725

HDC3120 to ADS7142 Temperature Conversion Equations:

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

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

Using [Equation 14](#), the converted temperature result is: 25.2°C

Humidity:

ADC Humidity Output (Channel 1): 1833

HDC3120 to ADS7142 Humidity Conversion Equation:

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

Using [Equation 16](#) the converted humidity result is: 43.4 %RH

4 Summary

Texas Instruments offers a growing family of humidity sensors which provide end systems with critical environmental information. Understanding how to appropriately program TI's humidity sensors can help designers achieve the best performance for in cutting-edge systems. The provided code for each family of humidity sensor aims to serve as a reference to help engineers understand the process and avoid common debugging issues. For more information on sensor applications and device characteristics, please see the linked documents below.

5 Development Support and Documentation

5.1 Software Support

For rapid prototyping with Arduino™-based controllers, visit TI's [GitHub™](#) repository for environmental sensors to get started. This repository offers sample code for all available humidity sensors.

For deeper, C-based driver-level support, visit TI's GUI-based code generator, [ASC Studio](#) to get started.

For additional assistance, visit the [TI E2E Sensors Support Forum](#).

5.2 References

Texas Instruments, [How to Debug RH Accuracy Issues in RH Sensors](#), application note.

- Offers explanations and prevention strategies for RH accuracy errors.

Texas Instruments, [Humidity Sensor-Based Water Ingress Monitoring for Automotive Electronics](#), application note.

- Offers a humidity sensor-based detection method utilizing the HDC3020 sensor to rapidly identify water ingress in sealed or vented electronic enclosures.

Texas Instruments, [HDC3x Silicon User's Guide](#), user's guide.

- Helpful storage and handling guidelines for HDC3x family of sensors.

Texas Instruments, [HDC2x Silicon User's Guide](#), user's guide.

- Helpful storage and handling guidelines for HDC2x family of sensors.

Texas Instruments, [NIST Traceability for Temperature and Humidity Sensors](#), product overview

- Highlights NIST traceable devices and explains importance in modern applications.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#), [TI's General Quality Guidelines](#), or other applicable terms available either on [ti.com](#) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2025, Texas Instruments Incorporated

Last updated 10/2025