Pawel Gburzynski

HDC1000 humidity/temperature sensor

(preliminary) driver

September 18, 2017

Introduction

HDC1000 is a temperature and humidity sensor manufactured by TI. Documentation available from the manufacturer (data sheet + specification) covers all the technical intricacies.

In the present implementation of the driver, the sensor triggers no events.

Summary of the sensor's functionality

The sensor can be set to return just the temperature, just the humidity, or both. In principle, the sensor can also trigger an event when a new readout becomes available. We take no advantage of this feature in the present driver. First, it isn't particularly useful; second, the wiring of the sensor on the CC1350 SENSORTAG (which device has inspired the driver) ignores the interrupt pin (leaving it NC).

The philosophy of the measurement/data acquisition process is that data cannot be read until the conversion is done. Also, conversions must be explicitly started. This is in contrast to some other sensors which convert continuously and use shadow registers to keep the old data available for readout at any time (so readouts can be carried out independently of the conversions). In the present driver, a readout (**`read_sensor`**) starts a conversion and waits until it is completed. Thus, between explicit requests for data the sensor remains in the idle state.

The temperature is returned as the number of degrees Celsius multiplied by 100, i.e., the LSB corresponds to 1/100th of a degree. The driver produces this value from a 14-bit unsigned representation of the temperature actually returned by the sensor. The resolution of the result is 14 or 11 bits (an option), which formally means about 0.01 and 0.1 degree. The accuracy reported in the data sheet is between ±0.2 and ±0.4 degree.

The humidity is returned as the unsigned number of percent multiplied by 100, i.e., the resolution is 1/100th of a percent. The formal resolution options are 14, 11, or 8 bits translating formally into 0.006%, 0.05%, and 0.4%. Needless to say, the actual accuracy is much lower, of order ±3% RH.

The resolution options translate into the amount of time needed to carry out the measurement and, consequently into the power budget of the sensor. In the idle state (aka sleep mode), the sensor drains between 110 and 200 nA. The conversion times are listed below:

Resolution	Conversion time (ms)
Temperature	
14 bits	6.35
11 bits	3.65
Humidity	
14 bits	6.50
11 bits	3.85
8 bits	2.50

When both temperature and humidity readouts are selected, the conversion times stack together. The current drain during a conversion is in low microamps, so the sensor takes very little current no matter what it does. One exception is the 10-15 ms startup time

after power up, which is only relevant when the sensor is powered from a pin and often switched on and off, amounting to ca. 300 μA.

One somewhat exotic feature of the sensor is the *heater* which can be turned on and then it becomes active during conversion. The datasheet says that it may make sense to turn it on after a prolonged period of high humidity to eliminate the possible condensation. The heater takes about 60 μA extra current during a measurement.

Driver interface

The sensor must be explicitly turned on, which operation determines its mode and parameters. This is accomplished by invoking:

```
hdc1000_on (word options);
```

where **options** is a collection of fields composed by or'ring these constants defined in hdc1000.h:

Constant name	Meaning
HDC1000_MODE_HEATER	Turns the heater on
HDC1000_MODE_HUMID	Selects humidity readings
HDC1000_MODE_TEMP	Selects temperature readings
HDC1000_MODE_TR14	14-bit resolution for temperature; this is the default
HDC1000_MODE_TR11	11-bit resolution for temperature
HDC1000_MODE_HR14	14-bit resolution for humidity (the default)
HDC1000_MODE_HR11	11-bit resolution for humidity
HDC1000_MODE_8	8-bit resolution for humidity

If neither humidity nor temperature has been explicitly selected, then the default is humidity only.

Reading values

The sensor values can be read in the standard way:

```
read_sensor (word st, address val);
```

Note that the operation *always* has to wait (because data is only available a while after the conversion is started), so the state argument is important. If **st** is **WNONE**, the operation will busy-wait for the result, which may take more than 10 ms in some cases.

The readings come as words. If both conversion have been selected, the **val** argument should point to a two-element array of words; the first word is filled with the temperature, the second with humidity. Only one word is needed in the single-conversion case.