Pawel Gburzynski

TMP007
infrared temperature sensor

(preliminary) driver

September 16, 2017

Introduction

TMP007 is an infrared thermopile sensor manufactured by TI. Documentation available from the manufacturer (data sheet + specification) covers all the technical intricacies.

The sensor can be used to measure the temperature of a remote object located in front of it and, at the same time, take the temperature of the chip. Thus, the sensor returns two temperature values denoted T_{OBJ} and T_{DIE}. The sensor can perform an adjustment of the returned value of T_{OBJ} based on T_{DIE} and its gradient to improve the accuracy. The details are in the official data sheet.

In addition to returning values upon demand, the sensor can also trigger events on low and high thresholds settable independently for T_{OBJ} and T_{DIE}.

Summary of the sensor's functionality

Two options affect the sensor's functionality as far as taking the measurement is concerned:

1. whether the transient correction of T_{OBJ} should be applied (this is a binary option)

2. how many takes should be averaged into the returned value (the options are 1, 2, 4, 8, and 16)

Both temperatures are returned as 16-bit signed numbers representing the temperature in degrees Celsius times 32 (i.e., the value should be divided by 32 to yield the number of degrees).[1] The resolution (and the nominal accuracy of the results) is 1/32rd of a degree. The range is ±256 degrees. When active, the sensor takes measurements at the fixed intervals of 0.25 s (known as the conversion time). A new value is available for readout every conversion time interval multiplied by the number of averaged takes in a sample. For example, for 8 takes, the values are refreshed about every 2 s. Values can be read from the sensor more often than that, but they won't change more often.

The selection of the number of takes to average includes three additional options corresponding to 1, 2, and 4 takes where extra delay is inserted between the takes to reduce current usage. With these (slow) options, the selection of 1 take translates into a 1 s measurement interval (0.75 s extra delay), and 2 and 4 takes need 4 s to complete each (3.5 and 3 s extra delay, respectively).

The events (interrupts) generated by the sensor are described by four thresholds representing: high limit for T_{OBJ} , low limit for T_{OBJ}, high limit for T_{DIE}, and low limit for T_{DIE}. Each of those limits is independently enabled. Additionally, a binary flag selects one of two options for generating events (interrupts). With one option, known as the *INT* mode, the event status, once raised, is only cleared when the chip's status register is read (which in our driver happens on every value read). This means, for example, that when a limit is exceeded, the alert condition becomes pending and it will remain so (no matter what happens to the temperature) until data is read from the sensor. Then, the condition will be cleared (even if the previous condition persists) until the nearest moment when a threshold is reached again. With the second option, the *COMP* mode, the event status strictly follows the actual condition (and is not cleared by reading the sensor value).

[1] I am not sure, if we shouldn't try to normalize this somehow across the numerous temperature sensors that we use. Perhaps it wouldn't be a big deal to have it in 1/100's of a degree. Nonetheless, normalizing the format of returned values would be difficult, because different sensors return different kinds of additional information.

Note: the sensor also offers a voltage readout, although our present driver doesn't implement no direct interface to that readout. It can be implemented trivially, if needed.

Driver interface

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

```
tmp007_on (word options, word enable);
```

where **options** is a collection of fields composed by or'ring a few constants defined in tmp007.h. In particular, the number of takes to average a sample is selected by one of these constants:

Constant name	Takes	Conversion time (s)	Current (µA)
TMP007_CONFIG_AV1	1	0.26	270
TMP007_CONFIG_AV2	2	0.51	270
TMP007_CONFIG_AV4	4	1.01	270
TMP007_CONFIG_AV8	8	2.01	270
TMP007_CONFIG_AV16	16	4.01	270
TMP007_CONFIG_AV1_LP	1	1	85
TMP007_CONFIG_AV2_LP	2	4	60
TMP007_CONFIG_AV4_LP	4	4	85

The last column gives the current drain by the sensor when operanting in the corresponding sampling mode.

The remaining components of **options** are single bit flags represented by these constants:

Constant name	Meaning
TMP007_CONFIG_CORRECT	turns on advanced transient correction for T_{OBJ}
TMP007_CONFIG_COMP	selects the COMP option for triggering threshold events, as opposed to the default INT option

The **enable** argument of **tmp007_on** selects which conditions should be reported as events, i.e., should trigger interrupts. It can be composed of the following binary flags:

Constant name	Meaning
TMP007_ENABLE_OH	high limit for T_{OBJ}
TMP007_ENABLE_OL	low limit for T_{OBJ}
TMP007_ENABLE_LH	high limit for T_{DIE}
TMP007_ENABLE_LL	low limit for T_{DIE}
TMP007_ENABLE_CRT	conversion ready

The last selection will cause an event whenever a new data becomes available. Note that data can be read from the sensor at any time, so there is no particular need to pay

attention to the ”conversion ready” moments, but it is only updated at the conversion intervals prescribed by the averaging option.

The temperature limits for events can be set with this function:

```
void tmp007_setlimits (wint oh, wint ol, wint lh, wint ll);
```

where the arguments are word-sized signed integers (temperature can be negative). Note that a limit can be disabled either by setting it to an unreachable value or by clearing the corresponding **enable** flag. The former method is more convenient when the sensor has been already initialized.

The temperature format is the same for the measurements returned by the sensor and for the limits, i.e., they are signed integer values representing the number of degrees times 32 (the LSB corresponds to 1/32 of a degree Celsius, but the four least significant bits are ignored in a limit specification. In other words, the limits are specified with 0.5 degree resolution.

```
void tmp007_off ();
```

The operation turns the sensor off, i.e., puts it into the (inactive) sleep mode. On a board where the sensor is powered from a pin, this operation sets the pin to logical 0 completely removing the sensor from the power budget. In the connected sleep mode, the current drain by the sensor is between 2 and 4 μA.

Reading values

The sensor value can be read in the standard way:

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

The state argument is ignored (the sensor is always ready for data extraction). The second argument should point to an array of two words. The first word is filled with T_{DIE} and the second with T_{OBJ}.

Sometimes a value returned by the sensor is invalid (this is diagnosed by the sensor and can happen, e.g., when the supply voltage is low). In such a case, our driver fills both **val** words with zeros. Zeros are also returned when the sensor is switched off.

Receiving events

As usual:

```
wait_sensor (SENSOR_TMP007, TARGET_STATE);
```

Normally, the function immediately blocks (forces release), so it should be the last statement in the current state. The issuing FSM is suspended and will be resumed in **TARGET_STATE** when the event occurs (as described above).

Note that in the COMP mode, the event will persist after the sensor values have been read, for as long as the condition holds. So the application willing to operate in this mode should disable or reset the limits (or verify the condition otherwise) before resubmitting the wait request.