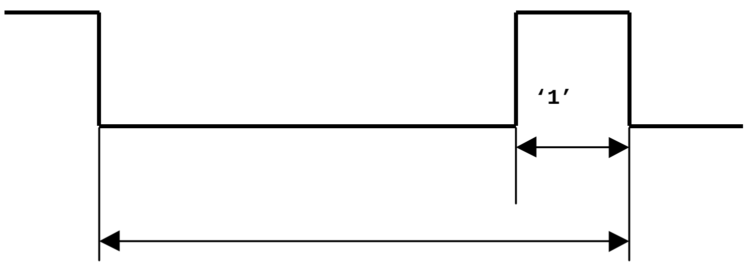
Connecting a SMARTEC temperature sensor to a 68HC11 type of microcontroller

by H. Liefting

This application note describes how to connect the Smartec temperature sensor to a 68HC11 microcontroller. Two types of inputs are considered: The capture input and the regular input.

A. Using the capture input

The SMARTEC temperature sensor has a duty-cycle output:



Full cycle

To be able to calculate the duty cycle, two measurements must be taken. One is the time that a full cycle takes, the other one is the time the signal is high ('1'). Both periods can be measured with an input capture timer.
The timer starts at the moment the input changes from logical '1' to '0'. The moment the signal goes from '0' to '1', the timer content is stored. At the end of the period, when the signal changes from '1' to '0' again, the timer content is stored once more. Now we can calculate the duty cycle.
The time that a full cycle takes (t_c) and the time the signal is high (t_h) are available now in units of 0.5ms. These times must be used to calculate the sensor temperature. The data sheet of the sensor gives us the formula:

$$\text{duty cycle (d.c.)} = 0.31924 + 0.00472 \times \text{Temperature } (^{o}C)$$

So the temperature is: 211.9 x (d.c. - 0.31924). To keep things simple, we will eliminate the decimals by multiplying with 2^{16} (65536) on both sides of the formula:

$$\text{Temperature} \times 65536 = 211.9 \times (2^{16} \times \text{d.c.} - 20922),$$

or:

$$309 \times \text{Temperature} = 2^{16} \times \text{d.c.} - 20922$$

The variable d.c. x 2^{16} can be calculated by: the time that the signal is high divided by the time a full cycle takes, and use the 'fdiv' instruction for the division operation. The 'fdiv' instruction will divide two 16-bit numbers after multiplying the divident with 2^{16}. This is exactly what we need. From the result of this calculation we must subtract 20922. Then we have 309*temperature. A temperature

of 25 degrees would yield the number 25 x 309 = 7725. It is simple to use these numbers for further calculations.
The frequency of the output signal of the sensor lies between 1 and 4 kHz. This means that every ms there is a new measured value available. That is much more than usually required. We benefit from this by measuring not a single period, but i.e. 100 periods and taking the average.

```
**************************************************
* Temperature measurement
**************************************************

* During 'still_to_do' periods of the inputsignal the measurement accumulates:
* - the periodtimes in 'periodsum'
* - the time the signal is a '1' in 'signalsum'

* At the end of each period, 'still_to_do' is lowered by 1.
* If 'still_to_do' reaches 0, the measurement is done and the sums will no
* longer be adjusted. The flag 'meas_on' will be reset.

* To start a measurement:
* - the flag 'meas_on' has to be set to '1'
* - 'still_to_do' has to be initialized with the number of periods
*               that have to be averaged during the measurement
* - the sums 'periodsum' and 'signalsum' must be set to 0
PROGRAM     space

* data area for the temperature measurement
DATA            space
still_to_do     rmb 1           |number of periods to accumulate
meas_on         rmb 1           |flag to indicate the measurement is running
periodsum       rmb 3           |accumulates period times
signalsum       rmb 3           |accumulates '1' time of the signal

* storage for internal use
periodstart     rmb 2           |startingtime of a period
starttime1      rmb 2           |time at which the signal became '1'

* initialization of the temperature measurement
PROGRAM         space
                clr meas_on
                clr still_to_do |no measurement active
                ldab #$7E       |initialize the interrupt vector
                stab tic1int
                ldd #sensorint
                std tic1int+1
                ldx #regsbeg    |start measuring a falling edge
                bclr tctl2-regsbeg,x,edg1a
                bset tctl2-regsbeg,x,edg1b
                ldab #ic1f      clear possibly pending interrupt
                stab tflg1-regsbeg,x
```

```
                bset tmsk1-regsbeg,x,ic1i
                jmp tempend     |end of the initialization

* subroutines for the temperature measurement
PROGRAM         space

******* input capture interrupt routine *******
sensorint       equ $
                ldx #regsbeg    |let IX point at the I/O registers
* first reset the interrupt-flag
                ldab #ic1f
                stab tflg1-regsbeg,x
* find out if it was a rising or a falling edge
                brclr tctl2-regsbeg,x,edg1a,sensorint1

* if it is a rising edge, we're in the middle of a measurement
* make a note of the time at which the edge occurred
                ldd tic1-regsbeg,x
                std starttime1
* then wait for a falling edge
                bclr tctl2-regsbeg,x,edg1a
                bset tctl2-regsbeg,x,edg1b
                bra sensorint9

* if it is a falling edge, we're at the end of a period
* the end of one period is also the start of the new period
sensorint1      equ $
                tst meas_on     |check if the measurement should be taken
                beq sensorint4
* when the measurement must be taken:
* accumulate the total time the signal was '1'
                ldd tic1-regsbeg,x
                subd starttime1
                addd signalsum+1
                std signalsum+1
                ldaa signalsum
                adca #0
                staa signalsum
* accumulate the total period time
                ldd tic1-regsbeg,x
                subd periodstart
                addd periodsum+1
                std periodsum+1
                ldaa periodsum
                adca #0
                staa periodsum
* one more period done, one less to do
                dec still_to_do
                bne sensorint4
```

```
* if no more periods have to be measured, the measurement is ready
                clr meas_on
* make a note of the time at which the new period started
sensorint4      ldd tic1-regsbeg,x
                std periodstart
* wait for a rising edge
                bset tctl2-regsbeg,x,edg1a
                bclr tctl2-regsbeg,x,edg1b
sensorint9      rti

* start a measurement of 100 periods
startmeas       equ $
                tpa             |save the condition-code register on the stack
                psha            |(using A)
                sei             |temporarily block the interrupts
                ldaa #100
                staa still_to_do
                ldd #0
                std periodsum   |clear the results
                staa periodsum+2
                std signalsum
                staa signalsum+2
                ldab #1         |start the measurement
                stab meas_on
                pula            |retrieve the condition-code register
                tap             |from the stack (using A)
                rts

***** routines for use by the main program
PROGRAM         space

* Measure the temperature, and leave the result in D. This result is
* the number of degrees (in Celcius) * 309
grad            equ 309
meas_temp       equ $
                pshx            |save IX on the stack
                bsr startmeas   |start a measurement of 256 periods
meas_temp0      tst meas_on     |wait until the measurement is ready
                bne meas_temp0
mess_temp1      ldab periodsum          |scale both times back to values that
                orab signalsum          |will fit in 16 bits
                beq meas_temp2          |do this by dividing both values by 2
                lsr periodsum           |until both are within the 16-bit range
                ror periodsum+1
                ror periodsum+2
                lsr signalsum
                ror signalsum+1
                ror signalsum+2
                bra mess_temp1
```

```
meas_temp2   ldx periodsum+1
             ldd signalsum+1
             fdiv          |calculate the duty-cycle
             xgdx          |put the result into D
             subd #20922   |then correct (see explanation in handbook)
             pulx          |retrieve IX from the stack
             rts
```

B. Using the regular input

An input capture timer is the preferred input for a SMARTEC temperature sensor. However the SMARTEC sensor can also be connected to a regular input. The program will then have to scan the input, and determine the signal/period ratio from the scanned signal. Do this by accumulating the number of samples, and the number of times the signal was '1', and dividing one by the other.

There are a few things to take into account when this approach is used:

The resolution of the measurement is determined by the number of samples that is used to calculate the duty cycle. Since the scanning process is comperatively slow, it is not possible to accumulate a large number of samples in a reasonable amount of time. A small number of samples will yield a less accurate calculation of the duty cycle, and hence of the temperature. A larger number of samples can be accumulated from a number of scans. In that case, the measured value will be less responsive to changes in the temperature.

An important factor to keep in mind, is that there should be absolutely no relation between the sensor signal and the sampling process. If the frequency of the sensor signal is (some multiple of) the frequency with which the signal is scanned, you will get false results. Note, that the frequency of the sensor signal varies with temperature. Make sure that the input is scanned at a random time within the sensor signal period.

The following code can be called by a main program to measure the temperature. In this example it is assumed that the measurement routine is called as part of some main scanning loop that is called at fixed intervals. To make sure the measurement routine is called at a random time within the smartec's cycle, a random delay was introduced in the measurement routine. If you can be sure you call the measurement routine at random times, you can ommit the internal random delay.

The measured values from all the measurements are accumulated, so the reading of the temperature will be an average of a relatively large number of samples, and will not respond quickly to changes in the temperature. If you need to detect changes quickly, do not accumulate the samples from consequetive measurements, but clear the accumulators before each measurement.

```
**************************************************
* Maximum length random generator
**************************************************

* The random generator is implemented as a 7-bit shift
* register that has the XOR of bit6 and bit5 fed back to bit0.

DATA        space
seed        rmb 1          |the shift register's seed

PROGRAM     space
** initialisation
             ldaa #1
```

```
                staa seed       |init. the register to anything but 0
                jmp randomend

******* get a random number (1..127) from the generator into (B)
getrandom       equ $
                ldab seed       |get current seed
                aslb            |shift left to XOR bit6 with bit5
                eorb seed
                aslb
                aslb            |resulting bit is now in the carry flag
                rol seed        |rotate new bit into shift register
                ldab seed       |get the new random number
                andb #$7F
                rts

randomend       equ $


**************************************************
* Temperature measurement
**************************************************
* The measurement accumulates:
* - the number of times a sample was taken in 'periodsum'
* - the number of times the signal was a '1' in 'signalsum'
* Whenever one of these two accumulators grow beyond 65535 (16 bits)
*   both will be divided by 2.
PROGRAM         space
tempstart       equ $           |the starting address of this module

* data area for the temperature measurement
DATA            space
periodsum       rmb 3           |counts samples taken
signalsum       rmb 3           |counts samples valued '1'

* initialization of the temperature measurement
PROGRAM         space
                ldd #0
                std periodsum   |clear the counters
                staa periodsum+2
                std signalsum
                staa signalsum+2
                jmp tempend     |end of the initialization

* subroutines for the temperature measurement
PROGRAM         space

******* take one sample from the sensor
take_sample     equ $
                ldab porta      |get the signal into bit0 of accumulator (B)
```

```
                asrb
                asrb
                andb #bit0      |mask out only the smartec signal
                clra            |add sample to accumulator
                addd signalsum+1
                std signalsum+1
                ldaa signalsum
                adca #0
                staa signalsum
                ldd #1          |add one more sample to the sample counter
                addd periodsum+1
                std periodsum+1
                ldaa periodsum
                adca #0
                staa periodsum
                rts

******* take a number of consequetive samples from the sensor
******* the number of samples to take is expected in (B)
take_samples    equ $
                pshb            |take the required number of samples
                bsr take_sample
                pulb
                decb
                bne take_samples
take_samples0   ldab periodsum          |scale both accumulators back to values that
                orab signalsum          |will fit in 16 bits
                beq take_samps9         |do this by dividing both values by 2
                lsr periodsum           |until both are within the 16-bit range
                ror periodsum+1
                ror periodsum+2
                lsr signalsum
                ror signalsum+1
                ror signalsum+2
                bra take_samples0
take_samps9     rts

* Measure the temperature, and leave the result in D. This result is
* the number of degrees (in Celcius) * 309
meas_temp       equ $
                pshx
                jsr getrandom   |get a random number into (B)
meas_temp0      decb            |and use that to generate a random delay
                bne meas_temp0
                ldab #50
                bsr take_samples
                ldx periodsum+1
                ldd signalsum+1
                fdiv            |calculate the duty-cycle
```

```
                xgdx            |put the result into D
                subd #20922     |then correct (see explanation in handbook)
                pulx            |retrieve IX from the stack
                rts

tempend         equ $           |end of this module

* the number of bytes this module requires in the PROGRAM area
tempsize        equ tempend-tempstart
```