Advanced Engineering Forum *Online: 2012-09-26*
ISSN: 2234-991X, Vols. 6-7, pp 294-298
doi:10.4028/www.scientific.net/AEF.6-7.294

The Programming Algorithm Based on Embedded System for the Output Conversion of the Humidity & Temperature Sensor SHTxx

Zou Jiupeng[1, a], Dai Yuqiang[1, b], Liu Xuewu[1, c], Zhang Liming[1, d] and Liu Fengxia[1, e]

[1]Dalian University of Technology, Dalian, P. R. China

[a]zoujp@dlut.edu.cn, [b]daiyuqiang@dlut.edu.cn, [c]liuxuewu@dlut.edu.cn, [d]zlmdl718@sina.com, [e]liufx@dlut.edu.cn

Keywords: Sensor SHTxx; Programming algorithm; CRC; Embedded system;

Abstract. In order to avoid the digital humidity & temperature sensor SHTxx's output values conversion consume many quantity of storage location, and spend more operation time, a group new type of conversion polynomials, and corresponding programming algorithm were deduced and tested. The polynomials are precise equivalent to the conversion formulas provided by the manufacturers, but contain only Binary fixed-point integer, fractional part, and 2^N. Using fixed-point calculations and shift operations instead of floating-point calculations, the results of program code reduction amount of 60 percent, and computing speed faster nearly 4 times than the original algorithm are obtained. Furthermore, a kind of speedy and unified CRC algorithm for read-out data of the sensor is proposed. The novel programming algorithm makes the output conversion more simplified, so it could pave the way for the low-end embedded applications of SHTxx.

Introduction

SHTxx series is Switzerland SENSIRION's produces of high integrated digital Humidity & Temperature Sensor, with high measurement precision (temperature 14 bit and Humidity 12 bit resolution ratio), it can be widely applied to many domain such as HVAC (Heating Ventilation Air Conditioning), weather station, workshop humidity and temperature control, medical treatment, automobile and home appliance etc.

However, SHTxx does not like the DS18B20 which directly outputs the Binary integer and decimal complement form of measurement data, but outputs unsigned binary integer: corresponding to the measuring temperature is -41.08℃, 14 bit output data is x000H; when 100℃, in between 371FH to 3720H. Another not enough is that its output value and measured value don't be the linear relationship. So only the conversion and non-linearity compensation of the output data are done by receiving party, the real temperature value Tture and humidity value RHture can be solved.

It is significant to research simple and accurate conversion algorithm for decreasing program code and speeding up the computing, so the SHTxx can be applied widely.

Defects of Conversion and Calibration by Floating Point Calculating

SENSIRION manufacturer propose the conversion formulas as follows [1]:
Temperature conversion:

$$T_{ture}(℃) = d1 + d2 \times T_{out} + d3 \times (T_{out} - g)^2 \tag{1}$$

Humidity conversion:

$$RH_{line}(\%RH) = c1 + c2 \times RH_{out} + c3 \times RH_{out}^2 \tag{2}$$

Besides the conversion, the Humidity compensation associated to temperature is also required:

$$RH_{ture}(\%RH) = (T_{ture}(℃) - 25) \times (t1 + t2 \times RH_{out}) + RH_{line} \tag{3}$$

In the aforesaid formulas, d1 is determined by the operating voltage, d2, d3 and g are conversion coefficients determined by the number of bits of the temperature output. To convert into Celsius temperature, the d1 for 5V and 3V are -40.1 and -39.6 respectively (offset hot temperature rise). Corresponding to 14 bit and 12 bit output, the d2, d3 and g are given in the table 1.

Table 1. The temperature conversion coefficients

T output	d2	d3	g
14 bit	0.01	-2E-8	7000
12 bit	0.04	-3.2E-7	1750

The conversion coefficients c1, c2 and c3, compensation coefficients t1 and t2 are determined by the number of bits of the humidity output. There is some difference between V4 and V3 version, corresponding to 12 bit and 8 bit of the output, these coefficients please see the table 2.

Table 2. The humidity conversion and compensation coefficients

RH% output output	Version V4 c3			Version V3* c3*				
	c1	c2	c3	c1*	c2*	c3*	t1	t2
12 bit	-2.0468	0.0367	-1.5955E-6	-4.0000	0.0405	-2.8000E-6	0.01	0.00008
8 bit	-2.0468	0.5872	-4.0845E-4	-4.0000	0.6480	-7.2000E-4	0.01	0.00128

Because of the coefficients contain decimal part, and the conversion formulas are non-linear (Contains quadratic terms), multi-byte floating-point calculations must be done. Obviously, the calculated quantity of the receiving procedures is larger, that spent a lot of time so the real-time performance of multi-sensors processing system, transmission unit or instrument, will be deteriorated. Furthermore, the more disturbing problem is that floating-point program looks simple, but after compiled the code is greater amount (total program generally more than 7KB). It is too large to the ROM of most base types MCU, such as 80C51, PIC16 etc, so SHTxx would be limited to use in the low-end embedded system. Although the SENSIRION's datasheet has given the linearization conversion formulas for the humidity, and two fold line for the temperature to avoid square operators, it is at the cost of loss of accuracy (the maximum error of humidity conversion is increased from 0.1% to 2.2%, and 0.1% to 0.8% the temperature conversion), the converting errors is several times more than the sensor's measurement error.

The Fixed-Point Transform of Floating Point Conversion Formulas

For the embedded microprocessor or microcontroller which does not have float-point unit, it is more quick and has less code size by several times using fixed-point calculation and shift operation to substitute for running a float Point program. For this reason, it is the key to transform all the items in the before-mentioned fomulas to the fixed-point binary number, and multiply by or divide by 2^N. For this purpose, the transformed expressions are deduced. For the 5V working voltage and 14 bits output, the temperature conversion formula (1) can be transformed to:

$$\begin{aligned} & T_{ture}(℃) = -40.1 + 0.01\times T_{out} - 2E\text{-}8\times(T_{out}-7000)^2 \\ \approx\ & (-40-0.1) + [(41/4096)\times T_{out}] - (1/(5E7)\times(T_{out}-7000)^2 \\ \approx\ & (-28H-0.00011010B) + (T_{out}\times 29H/2^{12}) - [(T_{out}-1B58H)^2/(2FAFH\times 2^{12})] \end{aligned} \tag{4}$$

In the expression, the first constant term - 40.1 is broken down into Single-Byte binary integer and binary fractional part, so the item can be calculated with fixed-point addition and subtraction, the conversion error of fractional part relative to the quondam formula (1) is only 1.5‰. The coefficient 0.01 in second item is changed to a form of the integer 29H dividing by 2^{12}, so the floating-point arithmetic can be replaced by fixed-point multiplication and shift operation. The

-2E-8 in third item is changed to the fraction form, and the denominator can be transformed into two bytes binary integer multiply by 2^{12}, so these can be solved by Double-Byte fixed-point multiplication and division, and N times right shift operation to substitute for divide by 2^N.

In the same way, corresponding to the 12 bit output of the relative humidity, formula (2) can be written as:

$$\begin{aligned} RH_{line}(\%RH) &= -2.0468 + 0.0367 \times RH_{out} - 1.5955E\text{-}6 \times RH_{out}^2 \\ &\approx (-2 - 0.0468) + [(301/8192) \times RH_{out}] - [(1/626763) \times RH_{out}^2] \\ &\approx (-02H - 0.00001100\,B) + (RH_{out} \times 12DH / 2^{13}) - (RH_{out}^2 / 9905H / 2^4) \end{aligned} \tag{5}$$

And formula (3) can be written as:

$$\begin{aligned} RH_{ture}(\%RH) &= (T_{ture} - 25) \times (0.01 + 0.00008 \times RH_{out}) + RH_{line} \\ &\approx (T_{ture} - 25) \times (0.00000011\,B + RH_{out} / 30D4H) + RH_{line} \end{aligned} \tag{6}$$

After transformation, the conversion expressions are (4), (5), and (6), they have high accuracy, the errors relative to the original formulas (1)-(3) are all under 2‰.

Operation Process and Solving Fractional Part

There are only double-byte binary integer, single-byte binary fractional part, and 2^N in the transformation expressions. Consequently, it can be compute by using the existing fixed-point multiplication and division standard procedures, and right shift operation.

In order to facilitate the integer operations, the multiplication would first be done, then is division, and last Right shift operation for every item. Because fractional part does not appear in the product of multiplication, and 4 bytes are required to store the product.

For every conversion expressions, an integer division must be done. The quotient is integer part, if it is 0, the integer part is 0. To get the decimal part, the remainder would be left shift 8 times (to complement zero on the right), again divide by the same divisor. It is a byte binary fractional part that new 8 bits quotient. Only a byte binary decimal is retained, the truncation error is less than $1/2^8 \approx 3.93‰$. The second remainder could be discarded.

Instead of dividing by 2^N, it has less computation workload to use right shifting N times (vacancy on the left side is filled with zeros). If the shifting times N is more than 8, one byte fractional part (second quotient) would be directly wiped off (see Fig. 1 with red mark), then N-8 bits are erased from the right of the integer (if the leftmost bit wiped off is 1, it would carry). Lastly, 8 bit from right to left of remainder integer (if the bits are not enough to complement zero on the left) are taken as the fractional part (see Fig.1). It looks more complex, but the real program is simpler.

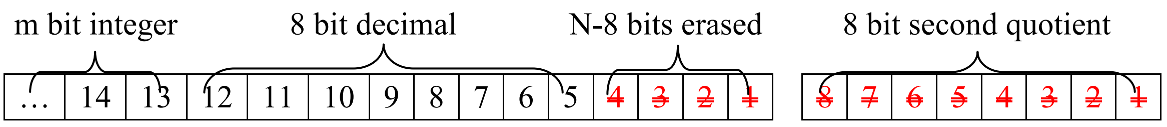

Fig. 1 The partition of integer and decimal for calculated digits

Finally, the integer and fractional part of every item would be algebraically added, the carry of fractional part is added to the integer. It is conformity with the rule of multi-bytes addition and subtraction, so both can be computed together, the result's bits number of fractional part does not change. For all of the calculations are true form, it is relatively simple.

The novel algorithm program compares with the floating-point calculation program, the code size decreases from 1384 bytes to 550 bytes, computing time reduces from 4483 to 1242 machine cycle. Even with CRC, the key operating, digital display, serial communication, and other functions, the program total code size is only about 3 KB in assembler programming language. So, even the low-grade MCU only 4 KB program memory and dozens of bytes of RAM (as 80C51, ATmega48, etc.)，also qualified for these tasks. And for multitasking system, the efficiency and real-time performance may be significantly improve because consumed resources for conversion of the sensor data and CRC are reduced.

CRC (cyclic redundancy check) of Read-out Data of the Sensor

The read timing sequence of SHTxx is that a read command (03H for temperature and 05H for humidity) must be inputted firstly. After 11/55/210mS (8/12/14-bit resolution ratio), it serially outputs two bytes of measurement data and one byte of CRC checksum [2, 3]. Noted two important points: 1. the command bytes (03H or 05H) must first be bring into calculation, follow by the measurement data to obtain the correct CRC checksum; 2. The CRC checksum must be reversed that can compare with another CRC checksum from SHTxx, if the both are equal, the transmission is right.

SENSIRION manufacturer does not provide CRC polynomial, only a 256 byte CRC lookup table. It consumes more program storage units for looking-up table and calculating. Referencing SHTxx internal CRC generating circuit (see Fig.2) [4], it may be computed by bit operation, as is more feasible for low-grade MCU, but the operation is slightly slower.

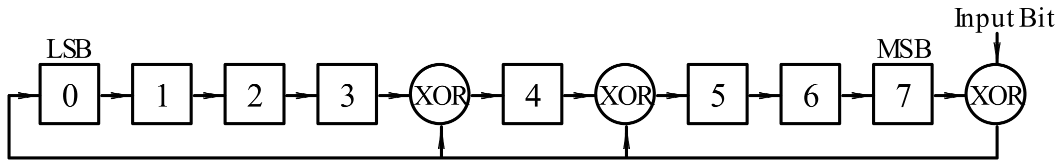

Fig. 2 SHTxx CRC-8 hardware generating circuit

Compare with the internal CRC generating circuit of digital temperature sensor DS18B20 (see Fig.3), the difference between both is only inverted sequence to generate CRC, that is D7＝D0*, D6＝D1*, ···, D0＝D7*. So as long as every inputted one byte data is reversed order, the CRC algorithms of DS18B20 [6] can be borrowed, the CRC checking polynomial as follows:

$$g(x) = x^8 + x^5 + x^4 + 1$$

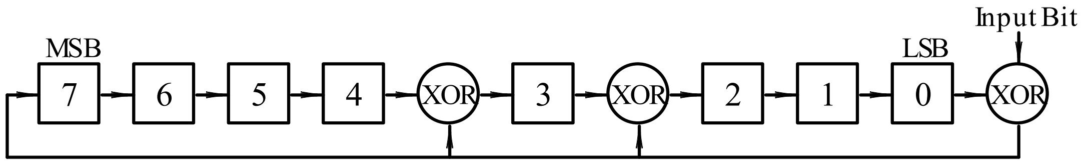

Fig. 3 DS18B20 CRC – 8 hardware generating circuit

That can decrease again program code, and speed up the operation. To reverse order the compute result, the CRC is equivalent to that obtained by bit operation. Because the CRC from SHTxx must be reversed order, then can compare with the computed CRC, so the CRC by borrowing the algorithms of DS18B20 does not need be reversed order, can compare with the CRC from SHTxx directly.

Conclusion

For the conversion of SHTxx output value, the above developed programming algorithm can markedly decrease program code size and consumed resources of the CPU, and can speed up the computing. So not only the low-grade MCU can be used for the conversion, also the efficiency and real-time performance may be significantly improved for multitasking system.

The programming algorithm could provide convenience and reduce cost for the multi-points and follow measurement of humidity and temperature, multi-sensors information fusion, and sensor communication node of Internet of Things. Furthermore, it provides the possibility that the sensor is connected to the low-end of consumer electronics.

The fixed-point Transform and shift operation method could also provide reference for similar embedded programming.

References

[1] SENSERION THE SENSOR COMPANY. SHTxx humidity & temperature sensmitter application note, Non-linearity compensation. Rev 1.5. http://www.sensirion.com

[2] SENSERION THE SENSOR COMPANY. Datasheet SHT1x (SHT10, SHT11, SHT15) humidity and temperature sensor, Version 4.1. Sept. 2008. http://www.sensirion.com

[3] SENSERION THE SENSOR COMPANY. SHT1x /SHT7x humidity & temperature sensor system, Evaluation kit available, V2.02. July 2004. http://www.sensirion.com

[4] SENSERION THE SENSOR COMPANY. SHTxx Humidity & Temperature Sensmitter Application Note CRC. Rev 1.07, http://www.sensirion.com

[5] Li Xiaowei. The temperature and humidity monitoring system based on the technology of embedded system. Dalian University of Technology, Master Dissertation, 2012

[6] Zou Jiupeng, Lin Yiaoyiao, Zhou Jian. Discussion of CRC coding and hardware fast check [J], Microcontrollers & Embedded Systems. 100 (2009) 76-78.