Chapter 7 SHT10 Relative Humidity & Temperature Sensor

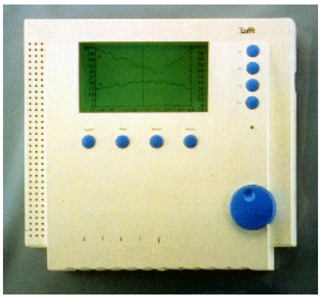
7.1 SHT10 Sensor

The SHT10 (U7) is a single chip relative humidity (RH) and temperature multi sensor module comprising a calibrated digital output. We use it to measure ambient temperature and RH values without using individual sensors. There are various articles on SHT1X sensor available from the Internet:

- Data sheet, application notes, and a demo program in Keil C available from Sensirion (www.sensirion.com), the manufacturer of this sensor.
- Sensirion SHT11 Sensor Module (#28018) by Parallax Inc (www.Parallax.com)
- Build a digital thermo-hygrograph (Part 1) at www.TechToys.com.hk ⇒ Components⇒SHT10 humidity & temperature sensor

RH and temperature recording devices are called thermo hygrograph. There are a lot of commercial devices in the market. Making a search on "thermo hygrograph" from the Internet gives a long list of commercial products like these:

- Simultaneous mechanical recordings of temperature and RH values on paper (consumables, need to buy it and refill)
- Battery powered
- Bi-metal sensors for temperature, and Nylon film for humidity
- Temperature: -10°C to 40°C
- Relative humidity: 0% to 100%RH
- Accuracy : ±2°C and ±5%RH between 30% and 90%
- Price: US$260



- Automatic and Precise Readings Transmitted from Data Logger to Your Computer when Required
- Graphic LCD-display (5" x 2 3/4")
- Printer interface
- Battery powered
- Any measurement interval, any storage interval
- Long term recording without service
- Table installation or wall mounting
- Alarm function
- PC-interface RS232
- Expandable memory (memory card)
- Processing range: -20°C to 50°C or (-4 to 122°F)
  Temperature probe: NTC
  Accuracy: ±0.2°C
  Resolution: ±0.1°C
  Humidity probe: HC200
  Accuracy: ±2.5%
  Resolution 0.5%
- Price: US$2,800!

Why bother to re-invent the wheel when there are similar devices in the market? First-of-all, working on a real-life application is a good exercise on embedded system design. Besides, it is from our point-of-view (embedded system engineers) the performance of those commercial products not any big deal. Let's look at the data sheet and compare:

	Relative Humidity %	Temperature °C
SHT10	±4.5	±0.5
SHT11	±3.0	±0.4
SHT75	±1.8	±0.3
US$260 commercial device	±5	±2
US$2,800 commercial device	±2.5	±0.2

Our SHT10 sensor(US$10) is not too bad even when it is comparing with the device of US$2,800, right? We also have a graphical LCD onboard. It is possible to use this LCD for graphing. We also have a RS232 port for data download to PC. What we don't have yet is a SD card or MMC card socket for mass data storage, which leaves room for a new version. If higher accuracy required, we may replace SHT10 with SHT11, which is completely pin-compatible and software driver the same too. Who knows, some day we may come up with a completely new design to compete. However, let's concentrate on a prototype first.

PCB layout of SHT10 sensor is shown in Figure 7.1.1.

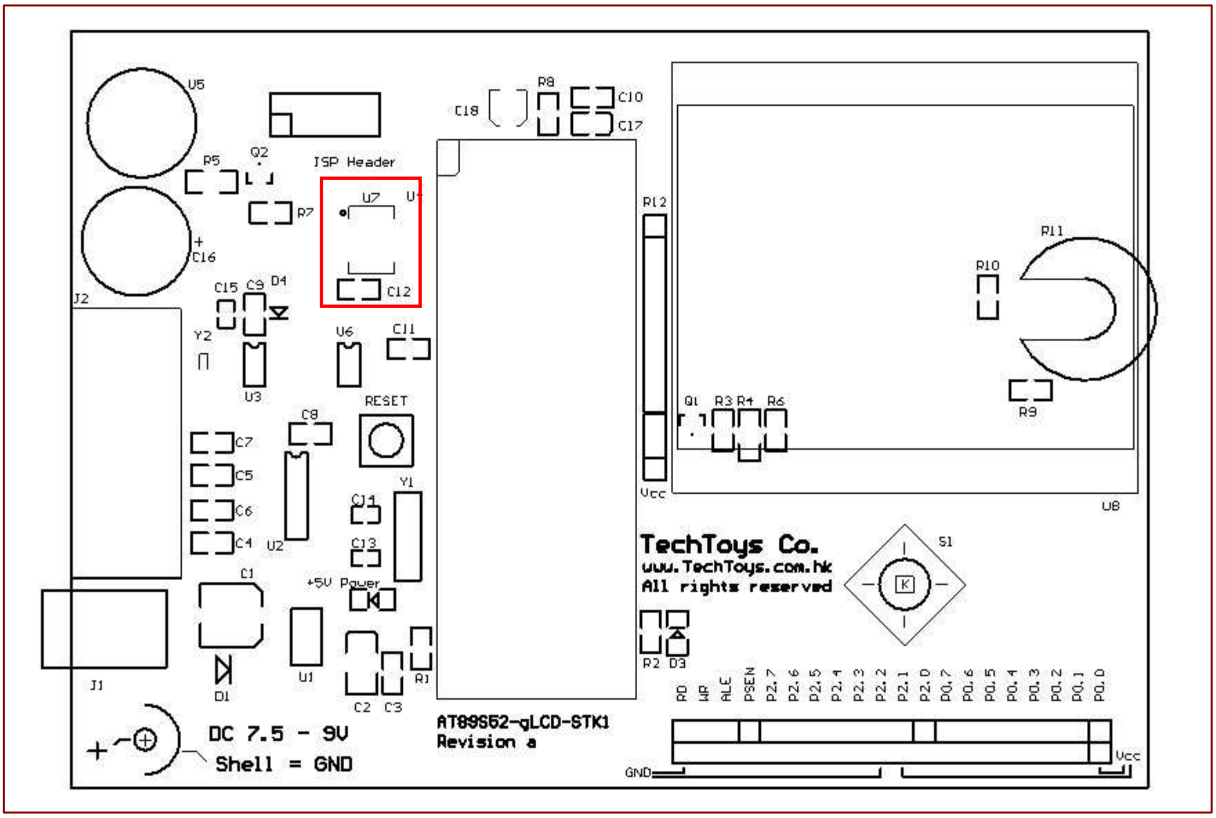


Figure 7.1.1 SHT10 digital sensor PCB layout

There is a working demonstration program on Keil C available from Sensirion web site at www.sensirion.com (Sample Code humidity sensor SHTxx Rev2.05). Unfortunately, this program uses floating point math and the linker output also exceeds 2K limit of our eval Keil C copy. We need to make some changes.

First we need to look at the schematic (Figure 7.1.2).

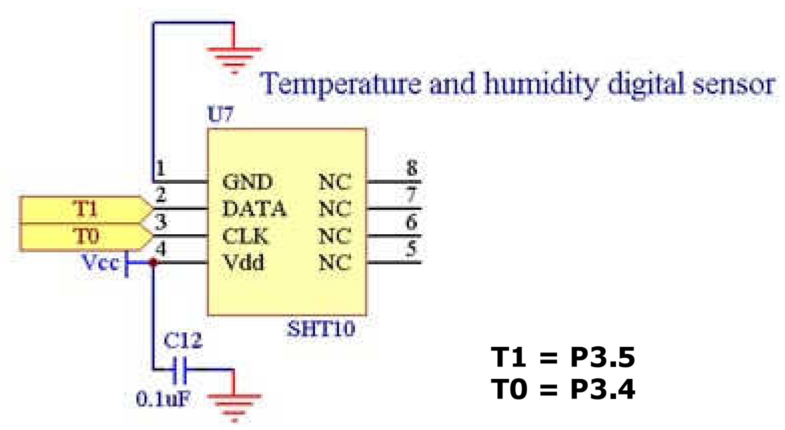


Figure 7.1.2 SHT10 interface with microcontroller

Because the sensor interface of SHT10 is not compatible with I^2C protocol, we have allocated P3.5 and P3.4 for its interface with the microcontroller.

Now, the aim is to minimize the program released by Sensirion to fit in 2k limit of eval Keil C, and at the same time, allow us to look at temperature and RH readings on a PC. There is not a lot of function to change. We need not doing the job all over again from zero but a simple modification on calc_sht11(float *p_humidity, float *p_temperature).

Let’s look at the original functions provided by Sensirion:

```
void calc_sth11(float *p_humidity ,float *p_temperature)
//This function converts raw data obtained from s_measure() to temperature in °C
and relative humidity in linearized, temperature-compensated %. This function is a
serious problem for us because it uses floating point calculation. Codes like these

rh_lin=C3*rh*rh + C2*rh + C1;
rh_true=(t_C-25)*(T1+T2*rh)+rh_lin;

with C1=-4.0, C2=+0.0405, C3=-0.0000028, T1=+0.01, T2=+0.00008, eat all 2k
program code space.

Heavy modification required!

float calc_dewpoint(float h,float t)
//This function calculates the dewpoint, again floating point required. Because we
are not giving data on dew point, therefore this function has been deleted.
```

Listing 7.1.1 Original Sensirion functions with floating point

Listing 7.1.2 shows the modified version using 2nd order approximation method as stated in the Application Note Non-Linearity compensation in Sensirion's web site. The source code is found under cd:\src\chp7\src7_1\SHTxx_Demo1.Uv2. This program does not print real-life reading on RH and temperature yet. Dummy data (raw_data) is used. Computational result is compared against the original result calculated by floating point method using an Excel worksheet. It is possible to use **Serial Window #1** under **View** of **Debug Windows**, or Docklight for data coming out of the UART port. We will use **Serial Window#1** to show the result. Interested parties may print via UART port as well. They give the same data!

The purpose of this program is to verify if our 'home-made" function calc_sht10() works as expected and it is not giving funny result such as humidity in 120% or temperature in 200.5°C. Figure 7.1.3 shows **Serial Window #1** after pressing **F5** under DEBUG window.

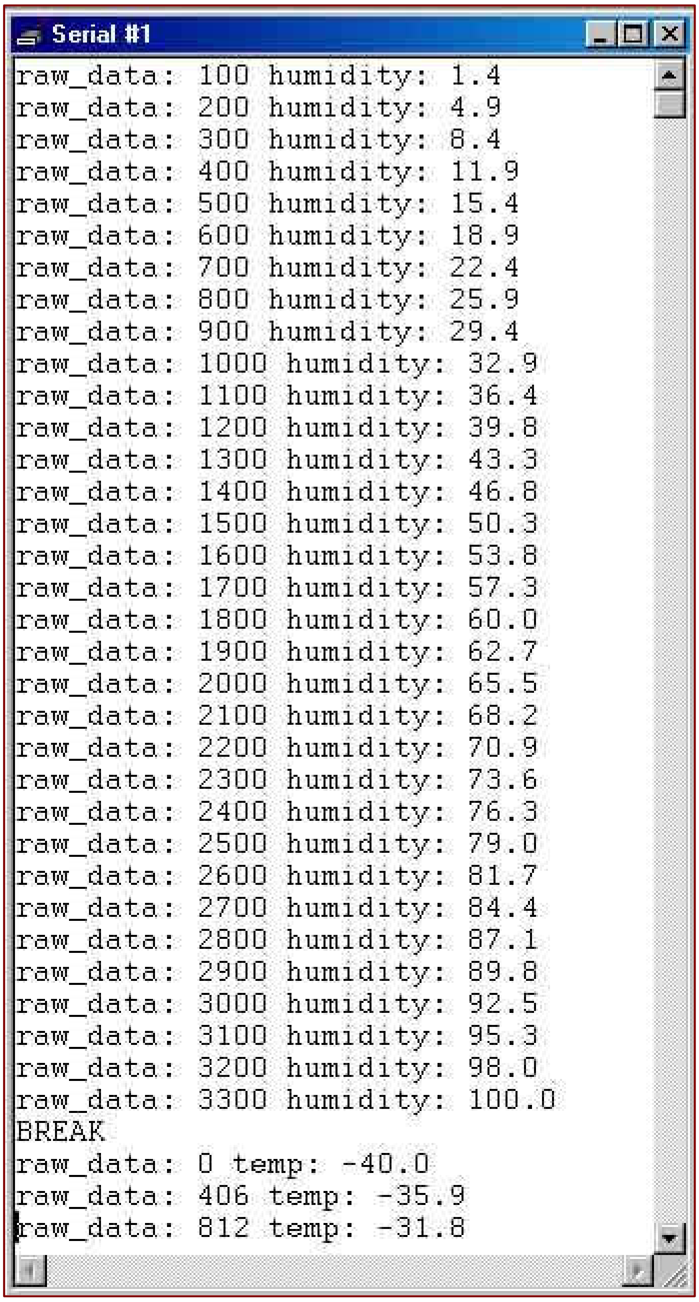


Figure 7.1.3 Simulation results on humidity (RH)

Figure 7.1.4 shows the Excel worksheet comparing floating point calculation (original Sensirion's example) against calc_sht10() computation.

Computational result by calc_sht10() using 2nd linear approximation

$RH_{linear}=C_1+C_2*raw\_data+C_3*raw\_data^2$
$C_1 = -4$
$C_2 = 0.0405$
$C_3 = -2.8*10^{-6}$

RH 12-bit raw data, dummy data for direct sensor readout

raw_data	rh_lin (floating point) %	uP computation %	% error
100	0.0	1.4	NA
200	4.0	4.9	-23%
300	7.9	8.4	-6%
400	11.8	11.9	-1%
500	15.6	15.4	1%
600	19.3	18.9	2%
700	23.0	22.4	3%
800	26.6	25.9	3%
900	30.2	29.4	3%
1000	33.7	32.9	2%
1100	37.2	36.4	2%
1200	40.6	39.8	2%
1300	43.9	43.3	1%
1400	47.2	46.8	1%
1500	50.5	50.3	0%
1600	53.6	53.8	0%
1700	56.8	57.3	-1%
1800	59.8	60.0	0%
1900	62.8	62.7	0%
2000	65.8	65.5	0%
2100	68.7	68.2	1%
2200	71.5	70.9	1%
2300	74.3	73.6	1%
2400	77.1	76.3	1%
2500	79.8	79.0	1%
2600	82.4	81.7	1%
2700	84.9	84.4	1%
2800	87.4	87.1	0%
2900	89.9	89.8	0%
3000	92.3	92.5	0%
3100	94.6	95.3	-1%
3200	96.9	98.0	-1%
3300	99.2	100.0	-1%

Figure 7.1.4 EXCEL sheet comparing rh_lin (floating point) with uP computation

As shown in Figure 7.1.4, we would be able to conclude that 2nd linear approximation alone induces a percentage error of ±1% in 40% - 100%RH range. If we look back on SHTxx data sheet (Figure 7.1.5), we may conclude the accuracy of our final product to be around ±5.5% due to a lack of floating point computational power plus the intrinsic measurement error of the SHT10 sensor. It is important to notice that the RH value has not been temperature compensated yet. For more serious application, we also need to take care of the following equation, especially when t_C deviates much from 25°C.

```
rh_true = (t_C-25)*(rh*0.00008+0.01)+rh_lin
```

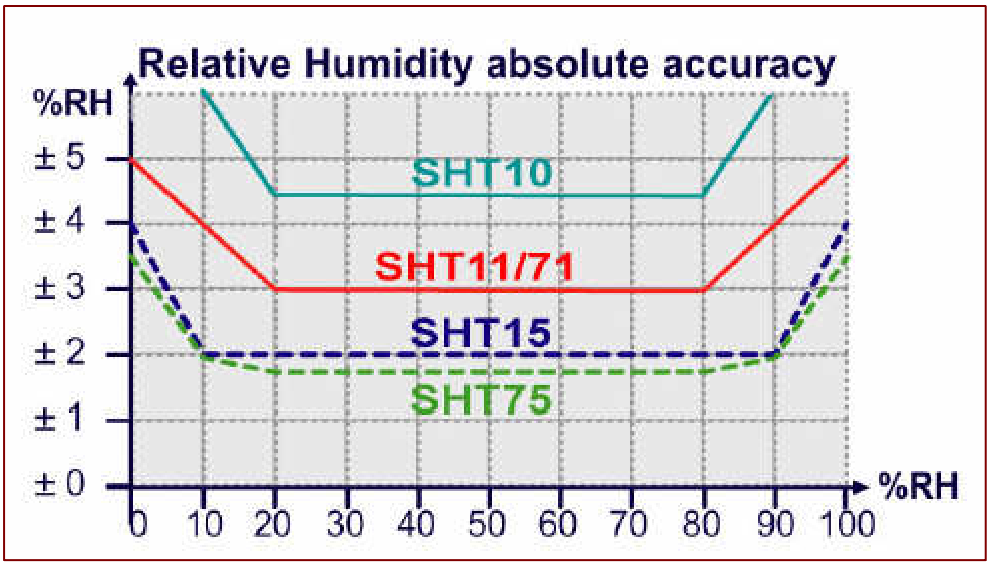


Figure 7.1.5 Relative Humidity absolute accuracy

What about temperature? Scroll down Serial Window #1 and see. This one is less difficult than RH. The original formula with floating point is:

t_C=t*0.01-40;

t_C = temperature in °C,
t=raw 14-bit reading from SHT10 sensor.

We cannot handle a multiplication factor of 0.01 as it involves floating point calculation. In order to facilitate print-out in 1 decimal place, we just need to give the equation a factor of hundred (x100) and handle decimal point printing in printf() as below.

```
t_C (hundred) = t - 4000
….
….
printf("raw_data: %d temp: %d.%u\n", raw_data, temp_val/100, abs(temp_val)%100/10)
```

We may compare the computational result with the original result calculated by floating point method (Figure 7.1.6). An error of ±0.1°C induced. I would accept that!

t_C = raw_data - 4000
decimal place printing
handled by printf() function

t_C = raw_data*0.01 - 40

Dummy temperature
14-bit raw data

raw_data	temp (floating) °C	uP computation °C
0	-40.0	-40.0
406	-35.9	-35.9
812	-31.9	-31.8
1218	-27.8	-27.8
1624	-23.8	-23.7
2030	-19.7	-19.7
2436	-15.6	-15.6
2842	-11.6	-11.5
3248	-7.5	-7.5
3654	-3.5	-3.4
4060	0.6	0.6
4466	4.7	4.6
4872	8.7	8.7
5278	12.8	12.7
5684	16.8	16.8
6090	20.9	20.9
6496	25.0	24.9
6902	29.0	29.0
7308	33.1	33.0
7714	37.1	37.1
8120	41.2	41.2
8526	45.3	45.2
8932	49.3	49.3
9338	53.4	53.3
9744	57.4	57.4
10150	61.5	61.5
10556	65.6	65.5
10962	69.6	69.6
11368	73.7	73.6
11774	77.7	77.7
12180	81.8	81.8
12586	85.9	85.8
12992	89.9	89.9
13398	94.0	93.9
13804	98.0	98.0
14210	102.1	102.1
14616	106.2	106.1
15022	110.2	110.2
15428	114.3	114.2
15834	118.3	118.3
16240	122.4	122.4

Figure 7.1.6 Excel sheet comparing temp (floating point) with uP computation

```
#include <REGX52.h>
#include <stdio.h>
#include <math.h>

void init_uart()
{SCON  = 0x52;
 TMOD  = 0x20;
 TCON  = 0x69;
 TH1   = 0xfd;
}

void calc_sht10(unsigned int *p_humidity ,int *p_temperature)
{
	unsigned int rh = *p_humidity;
	int t      = *p_temperature;
	int t_C;
	unsigned int rh_lin;
	//t_C=t*0.01-40, multiplied by 100, t_C=t-4000
	t_C = t - 4000;
	//58<=rh<=1720,		(1430*rh-5120*16)/4096,		in x10
	//1721<=rh<=3273,	(1110*rh+28930*16)/4096,	in x10
	//From : Application Note Non-Linearity compensation, Sensirion
	if(rh<=1720)
	{
		rh_lin=(1430UL*(long)rh)>>12;
		(rh_lin>=20)?(rh_lin = rh_lin-20):(rh_lin=0);
	}
	else
	{
		//rh_lin = (1110*rh + 12320 + 4096*110)/4096, in x10
		rh_lin=((1110UL*(long)rh+12320UL)>>12)+110;
		if(rh_lin>1000) rh_lin=1000;
	}
	//rh_true = (t_C-25)*(rh*0.00008+0.01)+rh_lin
	//~0.12 %RH /deg C, this factor has been ignored at this moment
	*p_temperature = t_C;
	*p_humidity = rh_lin;
}

void main()
{	unsigned int	humi_val;
	int		temp_val;
	unsigned int	raw_data;
	init_uart();

	//test on humidity, 12-bit range
	for(raw_data=100;raw_data<3400;raw_data+=100)
	{
		humi_val=raw_data;
		calc_sht10(&humi_val, &temp_val);
		printf("raw_data: %d humidity: %d.%u \n", raw_data, humi_val/10, humi_val%10);
	}
	printf("BREAK \n");

	//test on temperature, 14-bit range
	for(raw_data=0;raw_data<16384;raw_data+=406)
	{
		temp_val=raw_data;
		calc_sht10(&humi_val, &temp_val);
		printf("raw_data: %d temp: %d.%u\n", raw_data, temp_val/100,
		abs(temp_val)%100/10);
	}

	while(1);
}
```

Listing 7.1.2 SHTxx_dummy_code.c

Now we know calc_sht10() is at least working, and it is not giving funny results except when the relative humidity is lower than 40%. At this stage of development, we would make a compromise between accuracy and a limitation with our tool.

Listing 7.1.3 shows the final version with hardware dependent functions to read/write SHT10 sensor combined with calc_sht10() function. Only the main() portion has been shown because all other functions like s_measure(), s_connectionreset() are the same as the original Sensirion's example. Please refer to the source code for details if interested.

```
……

void main()
// Sample program that shows how to use SHT10 functions
// 1. connection reset
// 2. measure humidity [ticks](12 bit) and temperature [ticks](14 bit)
// 3. calculate humidity x10 [%RH] and temperature x100 [Deg Cel]
// 4. print temperature, humidity in 1 decimal place

{ unsigned int humi_val;
   int temp_val;
   unsigned char error,checksum;
   unsigned int i;

   init_uart();

  s_connectionreset();
  while(1)
  {
   error=0;
   error+=s_measure((unsigned char*) &humi_val,&checksum,HUMI);  //measure humidity
   error+=s_measure((unsigned char*) &temp_val,&checksum,TEMP);  //measure temperature
   if(error!=0) s_connectionreset();                 //in case of an error: connection reset
   else
   {
         calc_sht10(&humi_val,&temp_val);      //calculate humidity, temperature
         printf("temp: %d.%u  humidity: %d.%u\n", temp_val/100, (temp_val)%100/10,
         humi_val/10, humi_val%10);
   }

   //----------wait approx. 0.8s to avoid heating up SHTxx-----------------------------
   for (i=0;i<40000;i++);     //(be sure that the compiler doesn't eliminate this line!)
   //-----------------------------------------------------------------------------------
  }

}
```

Listing 7.1.3 SHTxx_Sample_Code.c extract

The program size just makes it. It is 2,128Byte, stepping on the 2K margin. It would be sensible to check what if we are not printing out the results via UART. Comment out the ***printf*** function and re-compile. Program size becomes 851 byte. The function printf() is demanding in program size.

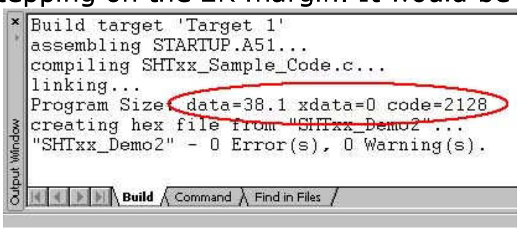


Figure 7.1.7 Compilation result

Figure 7.1.8 shows a Docklight screenshot. Temperature and humidity increase from 12:30:35:71 because I was holding my finger near the sensor opening at that moment. In around 1-second interval, both readings increase due to my body temperature and moisture. For more exaggerate result, breathe air to the sensor. You will see humidity soars from your mouth.

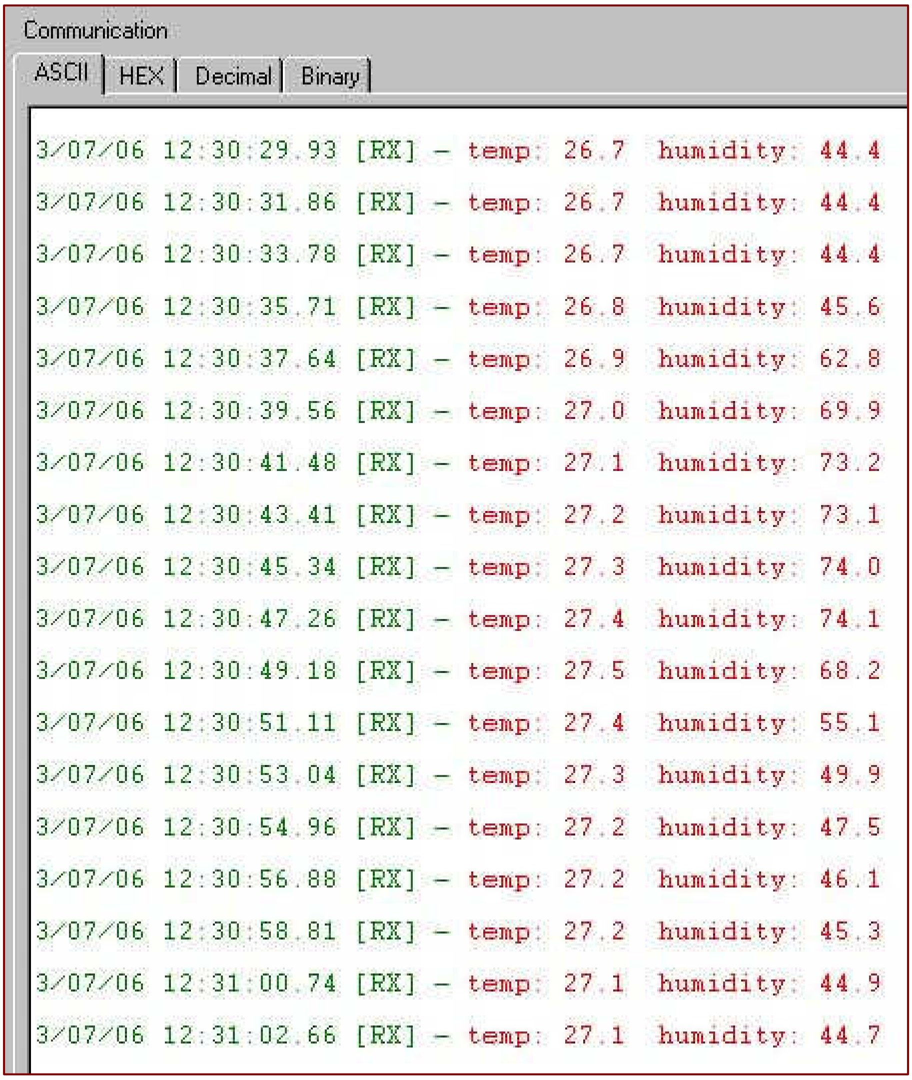


Figure 7.1.8 Docklight screenshot on actual data

⇒ Chapter 8