Z- Gingl, A photoplethysmograph experiment for microcontroller labs
INTERNATIONAL JOURNAL OF ELECTRICAL ENGINEERING EDUCATION 49:(1) pp. 42-60. (2012)
http://dx.doi.org/10.7227/IJEEE.49.1.4

Final manuscript

A photoplethysmograph experiment for microcontroller labs

Zoltan Gingl

Department of Technical Informatics
University of Szeged, H-6720 Árpád tér 2, Szeged, Hungary
e-mail: gingl@inf.u-szeged.hu.

Abstract

An easy-to-build photoplethysmograph experiment for microcontroller labs is presented. Students learn about direct sensor interfacing to modern mixed-signal microcontrollers and signal processing during an exciting biomedical measurement which is general and important enough to be included in various engineering courses. The full development is open source and the low-cost hardware is available worldwide. The experiment is highly transparent and scalable, accessible to beginners while providing many challenges to even the most qualified students. It can also be used as a demonstration tool in oral lecturing.

Keywords: photoplethysmograph, sensor, mixed-signal microcontroller, laboratory experiment, open source

Introduction

The broad range of microcontroller (MCU) application fields includes consumer electronics, household machines, battery-powered portable and wearable healthcare products, automotive industry, industrial sensing and embedded control, smart and wireless sensors and many more. Today's mixed-signal MCUs offer high integration of both digital and analogue peripherals like timers, communication ports, supervisory circuits, comparators, data converters, voltage references and temperature sensors. In addition, low power requirement, ease of use and high efficiency make MCUs very popular in solving many multidisciplinary engineering problems. It is obvious that training in MCU software and hardware, interfacing and embedding plays a key role in engineering education. Courses of different levels exist to aid undergraduate and graduate students of mechanical, biomedical or electrical engineering in learning about MCUs and their applications. It is important to note that besides the review of the theoretical background laboratory practices are needed to enhance the students' ability to successfully use MCUs to solve real world problems.

In the current paper a scalable MCU-based LED-transistor photoplethysmograph[1] laboratory practice experiment is presented aiming to demonstrate and teach how an analogue-intensive modern MCU can be applied to effectively provide information about the cardiovascular system of the human body. In a typical case such a lecture includes building analogue signal conditioning circuitry based on operational amplifiers, application of data converters and digital processing units and can also require the use of wireless networks[2-5]. These solutions either rely on preassembled parts of the development or need quite a long time and comprehensive knowledge available likely only in a limited environment.

Here an alternative approach is shown as an addition to the above mentioned methods. A highly integrated microcontroller incorporating a high-resolution data converter reduces the complexity of analogue preprocessing circuitry dramatically. At the same time the code can also be kept very simple without any loss of accuracy. Although the basic application requires small effort, many additions are possible; therefore a wide range of education levels can be covered.

The whole experiment is open source, the tested demonstration code is shared on a dedicated web

page[6], the development tools are available for free, and the required hardware is also very easy and cheap to purchase worldwide, so any lecturer can straightforwardly add it to an existing course. The experiment can also be ported to alternative hardware platforms.

The principle of finger photoplethysmography

The photoplethysmograph measures blood volume changes in an organ, most commonly in the finger. In a cardiac cycle the heart contracts and relaxes, making a change in the blood pressure and consequently a change in the momentary blood volume in the finger. Since hemoglobin absorbs the infrared light, this change can be easily monitored by using an infrared LED and a photodiode or phototransistor in either transmission or reflection mode as shown in Fig.1. The same principle is applied in pulseoximetry, when red light is used additionally due to its sensitivity of oxygen content in the haemoglobin, which allows the estimation of this value compared to the saturation level[3]. Note that the plethysmograph signal contains information about the heart rate and blood pressure changes, blood vessel pulsations, breathing and even more. The ease of use and installation, non-invasive nature and rich information content make photoplethysmography a very useful tool of monitoring the processes of the human body.

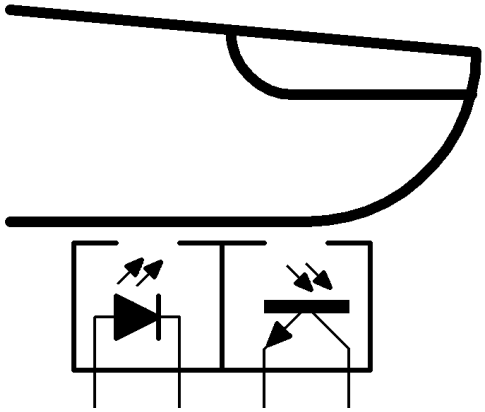

Fig. 1. Principle of the reflective finger photoplethysmograph. The intensity of the reflected light depends on the momentary volume of blood in the finger.

Hardware platform and experimental setup

The sensor and actuator

In one of the simplest experiments a separate infrared LED (SFH487) and phototransistor (SFH309FA) are used in a transmission measurement arrangement[7]. However, it is significantly simpler to apply the reflective method especially if the photodetector is integrated with the actuator[1]. In this work the TCRT1000 sensor-actuator pair was applied, therefore no additional mechanical work is needed to prepare the measurement head. The TCRT1000 has high sensitivity, while its housing style and built-in infrared filter provide easy installation, protection against interfering light sources and reliable operation during the experiment.

The MCU development kit

There are several high performance mixed signal MCUs on the market with integrated analog components. The measurement of photoplethysmograph signals requires handling a small AC component in the presence of large DC component. In most cases, the DC component is removed by an analogue high-pass filter, then the AC signal is amplified by operational amplifiers to match the range and input impedance of the analogue-to-digital converter (ADC) integrated into the MCU. However, if the MCU has a wide dynamic range and low noise ADC (resolution range from 16 to 24

bits), the required signal processing can be done in the digital domain and the external analogue signal conditioning can be omitted, so the sensor can be directly interfaced. This approach has a significant advantage in education, since the processing appears much less as a black box; students can simply implement the needed operations in the MCU and can easily explore the impact of parameter changes on the results.

This experiment is based on a C8051F350DK development kit from Silicon Laboratories[8]. The 8051 is one of the most popular, successful and longest living 8-bit MCU architectures well suited for sensors, actuators and most embedded control. Modern 8051 MCUs have fast core and extensive integration of analogue and digital peripherals. One example is the C8051F350, whose single-cycle core can be operated up to 50MIPS and integrates a highly configurable 24-bit sigma-delta ADC. The low cost development kit contains the target board with prototyping area, a USB debug adapter and emulator, code size limited compiler and integrated development environment — everything needed for the work. Note that the integrated development environment (IDE) and a configuration wizard software supporting easy setup of the MCUs registers are free as well as an unlimited and open source alternative compiler, the popular Small Device C Compiler (SDCC)[9].

Experimental setup

The block diagram of the system setup is shown on Fig.2. The sensor-actuator device is connected to the MCU without any analogue signal conditioning hardware – only two resistors are needed. One sets the current flowing through the LED and the other serves as a current-to-voltage converter of the photocurrent of the light detector. A recommended front-end input filter in for the ADC is preinstalled on the target board.

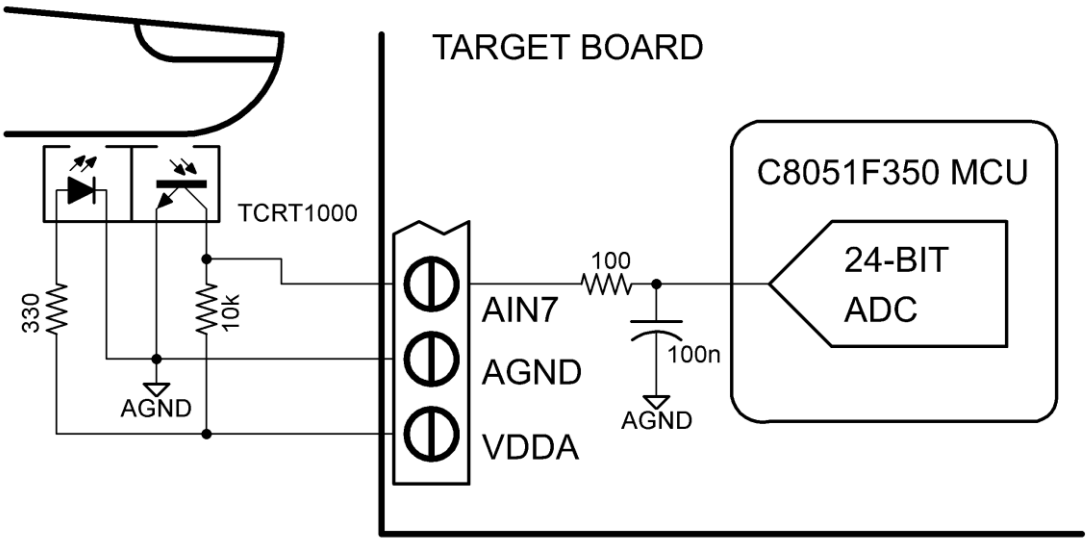

Fig. 2. The infrared LED and phototransistor can be connected to the MCU board using only two additional resistors, no further analogue signal conditioning is needed. The LED is powered from the VDDA supply line (nominally 3,3V) using a series resistor to set the current flowing through the LED.

The block diagram of the whole system setup is shown on Fig.3. The MCU code is developed on the host personal computer and the USB debug adapter allows downloading the code into the target, supports non-intrusive debugging, single-stepping and it even powers the board. Serial communication is provided between the MCU and the computer, therefore real-time processing and display of measured data are possible. The conventional RS232 port may be missing in newer computers, however reliable USB-RS232 converters are available. Note, that newer target boards have USB-UART interface rather than the conventional RS232 serial port. The communication protocol remains the same using the virtual serial port feature of the operating system.

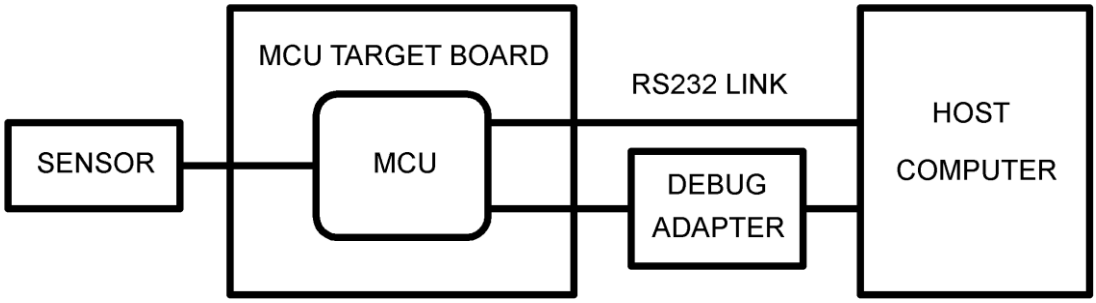

Fig.3. The block diagram of the experiment including the host computer connection.

Teaching the signal processing by scalable experiments

In this section experiment examples are reported that help students to understand and practice the signal processing typically required for the analysis of the photoplethysmographic and other similar signals. Fig. 4 shows a photo of the experimental setup. The sensor can be directly connected to the board using three wires, only two external resistors are needed.

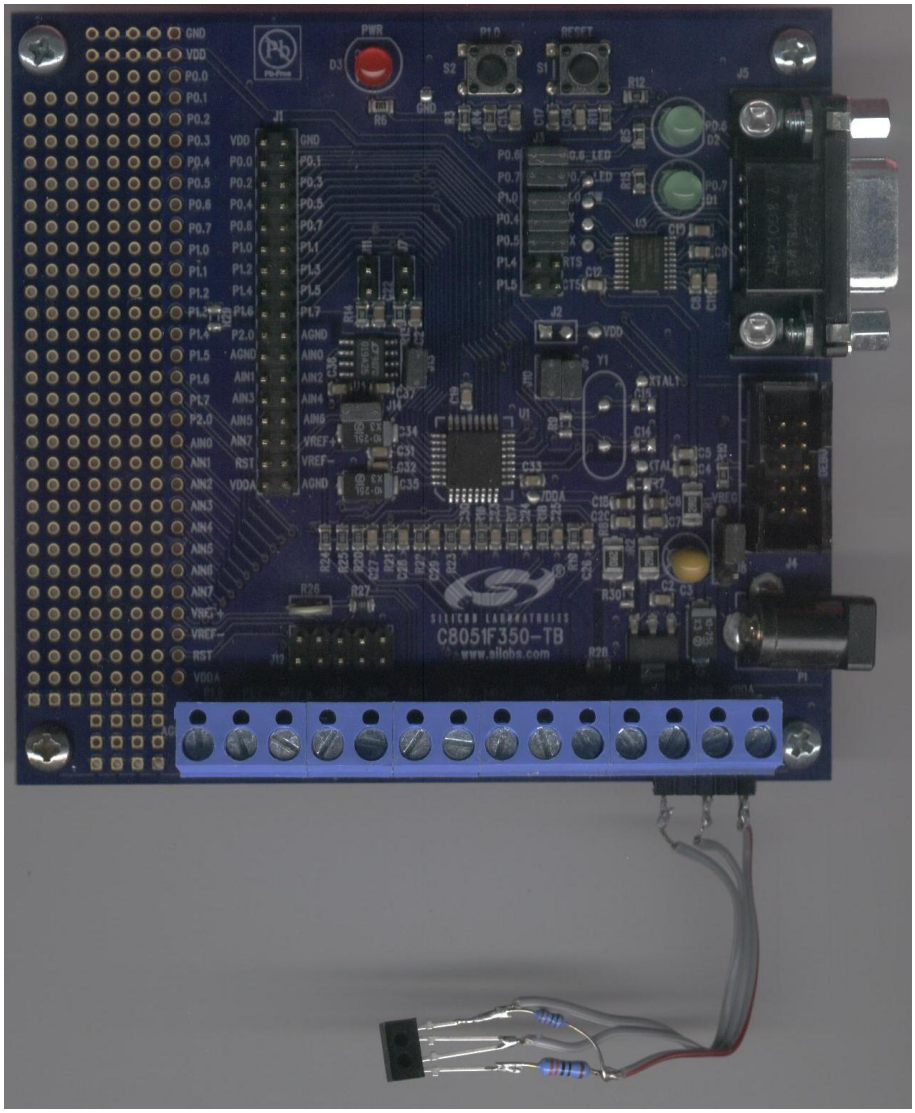

Fig. 4. The reflective optical sensor (TCRT1000) is connected to the microcontroller target board via three wires, only two additional resistors are needed.

Watching the raw output signal of the sensor

After installing the hardware, the students may try to check the output waveform of the sensor using an oscilloscope by touching AIN7 (see Fig.2.) with the probe. A typical result measured by a Tektronix DPO2024 oscilloscope is illustrated on Fig. 5.

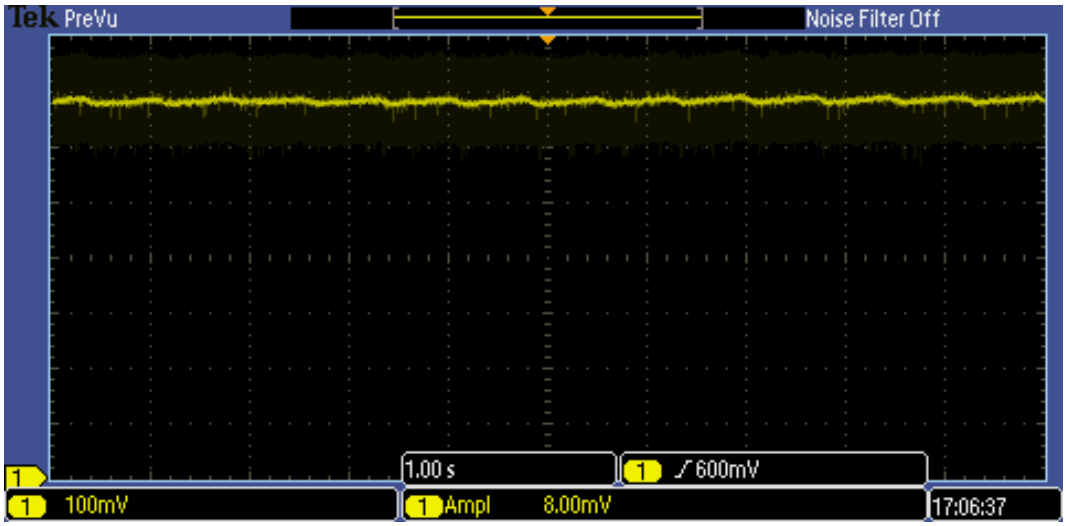

Fig. 5. Typical raw output signal of the sensor measured in DC coupled mode. The amplitude of the AC component of the signal is only a few percent compared to the DC magnitude.

Students can clearly see that the AC component is rather small, only a few percent of the DC component. Although the blood pressure can vary about 40% in a cardiac cycle, the blood volume change in the finger is smaller and significant part of the light is reflected from the bone, from the tissue, therefore the changes in the reflected light intensity falls in the range of a few percents.

The typical way to measure small AC signals in the presence of a large DC component is to use the AC coupling mode of the oscilloscope. However, the AC coupling in oscilloscopes means high pass filtering with corner frequency well above 1Hz, therefore most of the slowly changing AC component caused by the cardiac activity will be attenuated and buried completely in the high frequency noise, as shown on Fig. 6.

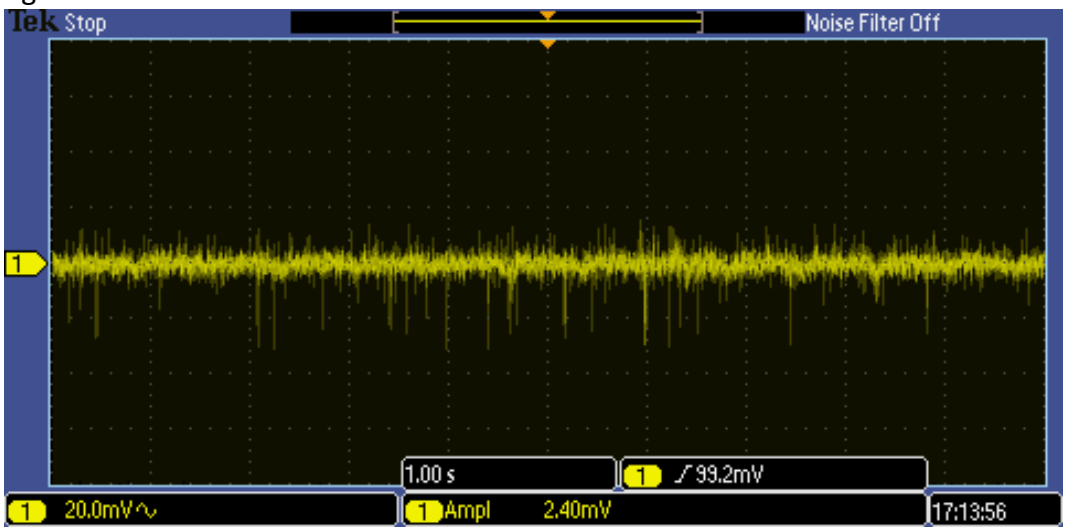

Fig. 6. If the student uses AC coupling mode, most of the photoplethysmograph signal will be attenuated by the high pass filtering and the signal will be buried in the high frequency noise.

After performing these tests the properties of the signal will be clear to the students and they will also see that an oscilloscope can’t be used to effectively extract information from the measured signal, further signal processing will be certainly needed:

- high pass filtering with sub-1Hz corner frequency to remove the DC component, but keep the slowly varying AC component,
- low pass filtering with corner frequency of about a 20-30 Hz to remove high frequency noise,
- amplification to get large enough signal, gain of 100 is sufficient according to the oscilloscope measurement presented on Fig.5.

Possible analogue signal processing

Students of electrical or biomedical engineering will probably suggest a simple analogue active filter to fulfill the above mentioned requirements. An often used simple two stage circuit example[7] can be further simplified using a single operational amplifier and some passive components as shown on Fig. 7.

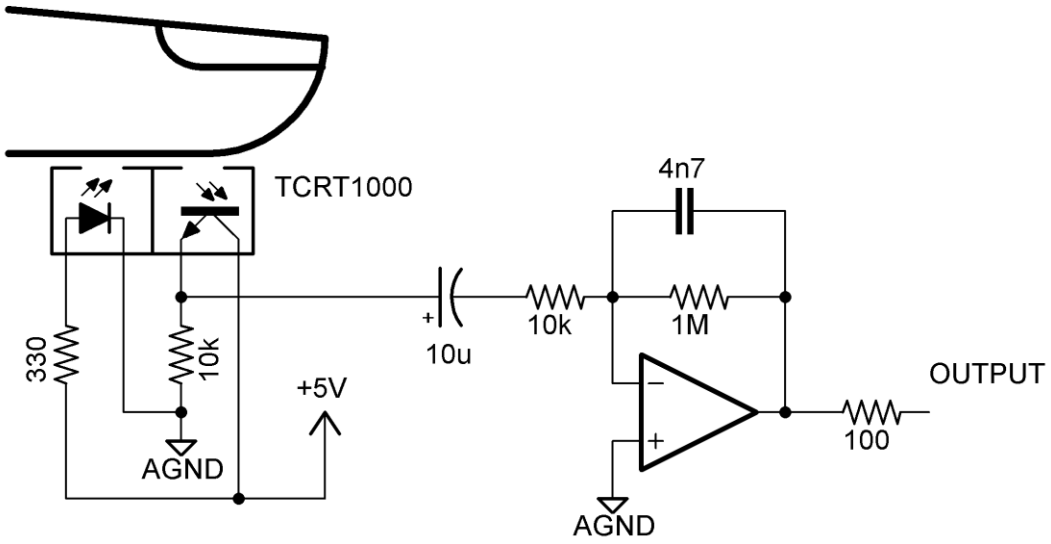

Fig. 7. This simple circuit removes the DC and high frequency noise components and provides signal amplification. General purpose operational amplifiers like the LM358, TL071 can be used.

This circuit can be built on a simple prototyping board and it is a good chance for students to learn that care must be taken to put 100nF supply decoupling capacitors close to the supply pins and a series resistor at the output of the operational amplifier to isolate the capacitive load caused by the oscilloscope probe to prevent possible oscillations. This active bandpass filter provides an overall gain of 100 in the passband range of about 1Hz to 30Hz, while both the DC and high frequency components are removed. Students can check the output signal again with an oscilloscope, also may vary the component values to see the effect of changing the filter corner frequencies both for the low and high pass part. Fig. 8. shows a typical waveform recorded by an oscilloscope.

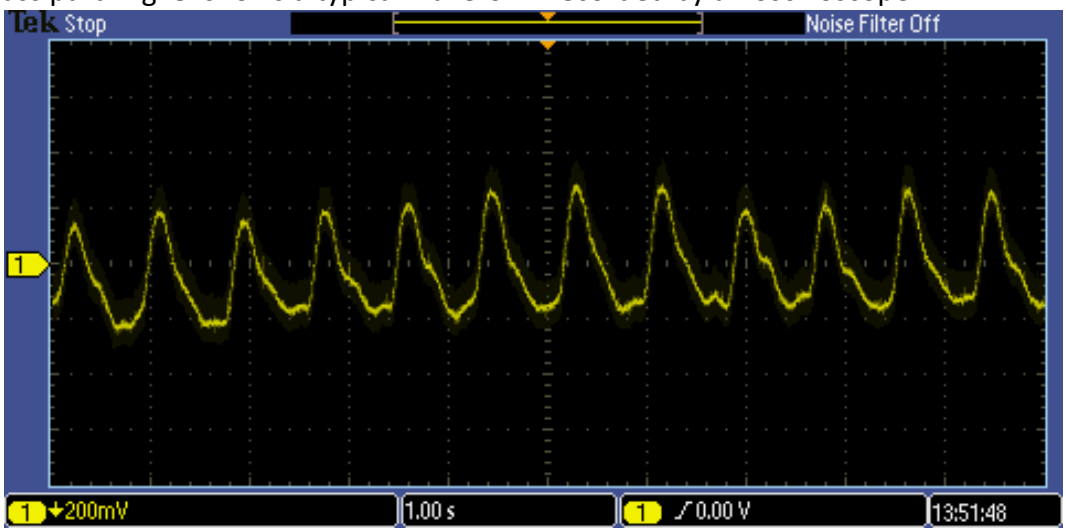

Fig. 8. Amplified and band-pass filtered photoplethysmograph signal. The circuit depicted on Fig. 7. was used for the experiment.

Digitising with the microcontroller's built-in high-resolution delta-sigma ADC

The output signal of the analogue circuit presented on Fig. 7 could be digitised with a commonly used 12-bit ADC, what could be the reason to use higher resolution ADCs? What could be the advantages and disadvantages?

Today's medium speed (up to about 10^6 samples per second) 12-bit ADCs typically have a built-in sample and hold circuit followed by a successive approximation ADC architecture (a.k.a. sampling ADC). According to the sampling theorem, the signal must not contain significant components above the half of the sampling frequency. Since the circuit of Fig. 7 forms only a first order low-pass filter, either a high sample rate or more complex analogue filtering will be needed. On the other hand, the DC removal also causes signal distortion depending on the corner frequency of the high-pass filter, that can't be easily optimised and changing the corner frequency of the high-pass filter needs component replacement.

All of these problems can be solved by applying a high-resolution delta-sigma ADC, sometimes called single-bit ADC[12,13]. This revolutionary ADC architecture is very common (they are used in sound cards, mobile phones), because it has several advantages including:

- the internal sample rate if much higher than the data rate, therefore the anti-aliasing filter can be much simpler (in several cases first order will be sufficient);
- very high resolutions can be realised (16-24 bits) with inherently high linearity and precision, therefore the dynamic range is very high and the noise is also very low;
- most of the circuit is digital, therefore it is easy to integrate with digital circuitry like processors and the price is also low (for example the C8051F350 microcontroller with built-in 24-bit delta-sigma ADC has almost the same price as the C8051F410 microcontroller with built-in 12-bit successive approximation ADC, close to £6; see at www.farnell.com).

In summary, the delta-sigma ADC allows the use of very simple external analogue circuitry, most of the signal processing can be done in the digital domain, therefore flexibility, complexity accuracy are dramatically improved while the price is kept low.

Since the delta-sigma ADC allows direct digitisation of the sensor's output signal without the need for DC removal, students can deal with the raw sensor data, therefore:

- they can implement the required processing, therefore they learn and understand more about such processing;
- they can tune the processing parameters very easily and see the effect in real time;
- they learn also about today's approach of highly integrated mixed signal (both analogue and digital) hardware and embedded programming.

In the following chapter signal processing software examples will be presented using the experimental arrangement shown by Fig 2.

Configuring the MCU and its peripherals

Since the C8051Fxxx series MCUs are among the most advanced 8051 family MCUs, care should be taken during initialization of the registers. A useful tool, the Config Wizard can help to generate the source code; however the students must understand first the basics of the C8051Fxxx features.

The setup process should include the configuration of the watchdog timer, the port input/output, the on-chip oscillator, the universal asynchronous communication port (UART) and related timers, the voltage reference and the ADC.

The MCU code

After setting up the registers the MCU is ready to acquire the signal coming from the sensor. Three typical operating modes are recommended to be implemented by the students. The tested and annotated C source code was developed using the free and open-source SDCC compiler and can be found in the Appendix.

Continuous data transfer to the host computer

This is the simplest operating mode, where the ADC does the conversions continuously and at the end of the conversion the microcontroller sends the 24-bit raw data to the host computer in either polling or interrupt mode. The data samples can be represented as text including a termination character or can remain in their binary representation.

In the first case the data can be retrieved and viewed using a simple terminal software like Hyperterminal, can be logged into a text file or the data can even be copied into the students' favourite spreadsheet software. Fig. 9 shows a typical raw data waveform. For an example code, see the *RawSensorData()* function in the Appendix.

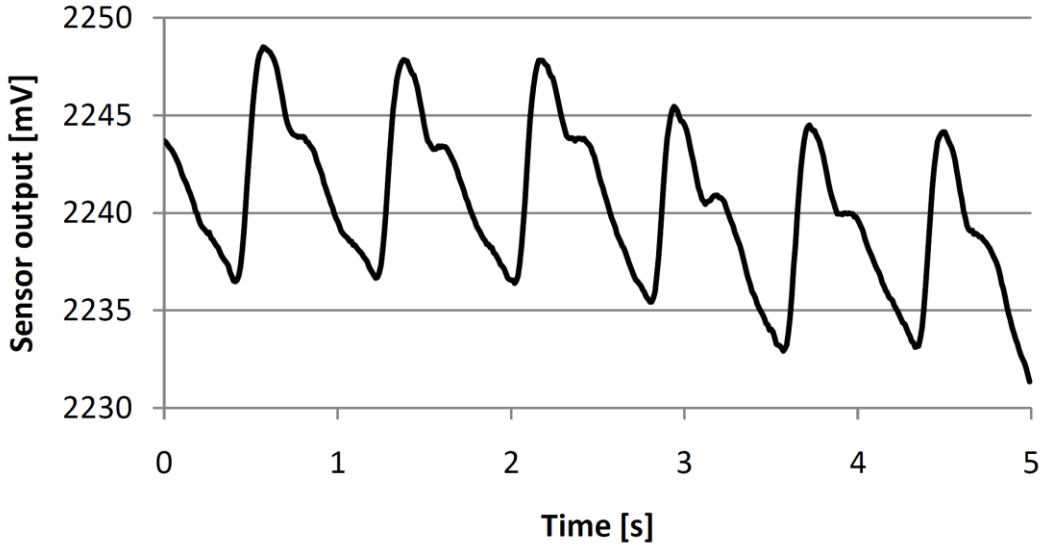

Fig. 9. Typical raw sensor signal. The data has been received from the microcontroller using the Hyperterminal software and copied into a spreadsheet program to make the plot of data. A slowly varying component can be easily seen.

Given that only one byte at a time can be sent over the serial port, students have to make their decision about using little endian or big endian format if they decide to use binary data representation. They should also take care of using signed or unsigned integer numbers. Consequently it is a very useful way to help the students to properly use data representation, where sometimes even practiced developers may make related mistakes. However, in this case a terminal software is not enough, the students must develop their own software to receive and process the data stream.

High-pass digital filtering

This mode is also based on continuous data conversion, but the raw data is first passed through a digital high-pass filter before the stream is forwarded to the host computer. Since the signal contains large DC and slowly varying components[10], these should be removed first to ease heart beat detection — it is the same job as in the case of analogue signal processing as was discussed above. At this point students are facing the problem how to implement a high-pass filter in the digital domain. In general, the recursive (a.k.a. infinite impulse response or IIR) digital equivalent of an analogue filter can be found by using the bilinear transform[13]. If *A(s)* is the analogue filter's transfer function, the

corresponding digital filter's *A(z)* transfer function can be obtained by substituting *s* as follows:

$$s \leftarrow 2 \cdot f_s \frac{(1 - z^{-1})}{(1 + z^{-1})}, \tag{1}$$

where f_s is the sampling frequency and $z=\exp(s/f_s)$. This way *A(z)* will be a polinom of z^{-1}:

$$A(z) = \frac{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2} + \ldots}{1 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} + \ldots}, \tag{2}$$

and the corresponding recursive filter will have the form:

$$y_i = a_0 \cdot x_i + a_1 \cdot x_{i-1} + a_2 \cdot x_{i-2} + \ldots + b_1 \cdot y_{i-1} + b_2 \cdot y_{i-2} + \ldots, \tag{2}$$

where x_i and y_i represent the input and the filtered sampled data, respectively.

However, for single pole (first order) filters students can use a simple method to easily implement a first order infinite impulse response high-pass digital filter using the following expressions[13]:

$$\begin{aligned} y_i &= a_0 \cdot x_i + a_1 \, x_{i-1} + b_1 \cdot y_{i-1}, \\ a_0 &= \frac{1 + e^{-2\pi \frac{f_c}{f_s}}}{2}, \\ a_1 &= -\frac{1 + e^{-2\pi \frac{f_c}{f_s}}}{2}, \\ b &= e^{-2\pi \frac{f_c}{f_s}}, \end{aligned} \tag{2}$$

where f_s is the sampling frequency and the corner frequency of the high-pass filter is f_c. Fig. 10 shows a typical waveform obtained with 1Hz corner frequency. For an example code, see the *FilteredSensorData()* function in the Appendix.

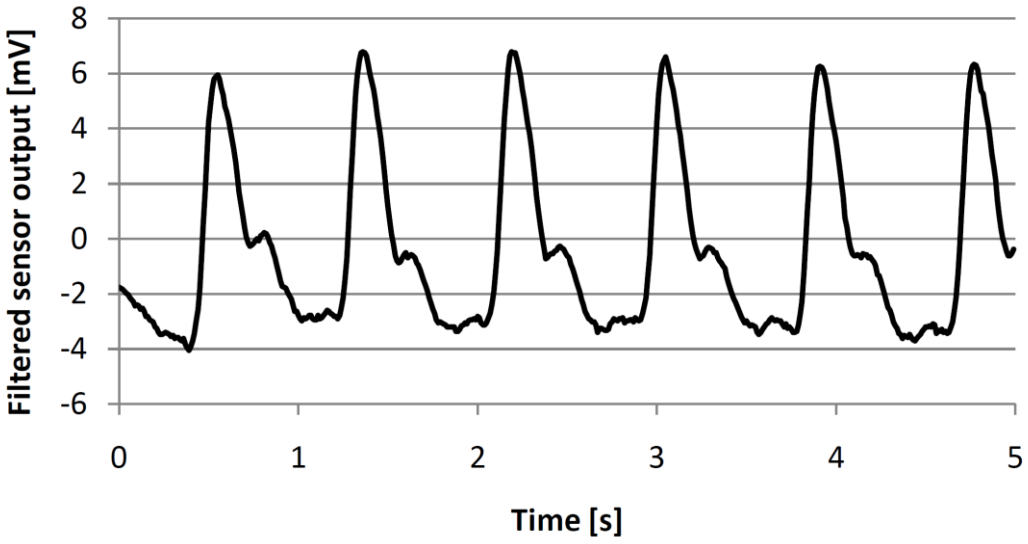

Fig. 10. Typical digital high-pass filtered sensor signal with 1Hz corner frequency. The DC component is removed, however the waveform is somewhat changed compared to the signal shown on Fig. 9.

Heart beat detection, heart rate calculation

After high-pass filtering the mean value of the signal becomes zero, therefore a simple way to visualize the heart beats can be realized by turning on an LED on the C8051F350 target board if the

signal is positive.

However, if students want to measure the time between successive heartbeats, simple zero-crossing detection won't be sufficient, since some noise on the data can cause false heart beat detections. A common technique can be exercised by the students here: add some hysteresis to the level-crossing detection to prevent false pulse counting. This can be easily implemented by using two thresholds: if the signal goes above the higher level, the pulse is detected and subsequent detection is blocked until the signal goes below the lower level. The lecturer can draw the students' attention to the analogue equivalent, a comparator with added hysteresis called Schmitt-trigger.

Measuring the time between successive pulse detections allows the students to calculate the beat-to-beat 'instantaneous' heart rate and send these values to the host computer at every heart beat. This can be easily implemented by just counting of the number of conversions between successive heart beat detections, since the conversions are performed at a given and known rate, 100Hz. In this mode the students can record the beat-to-beat heart rate and can observe that the heart rate is fluctuating around its mean (it is often referred and measured as heart rate variability), since the healthy nervous system controls the rate according to the fluctuating external and internal stimuli. A typical result is shown on Fig. 11. For an example code, see the *HeartBeatDetection()* function in the Appendix.

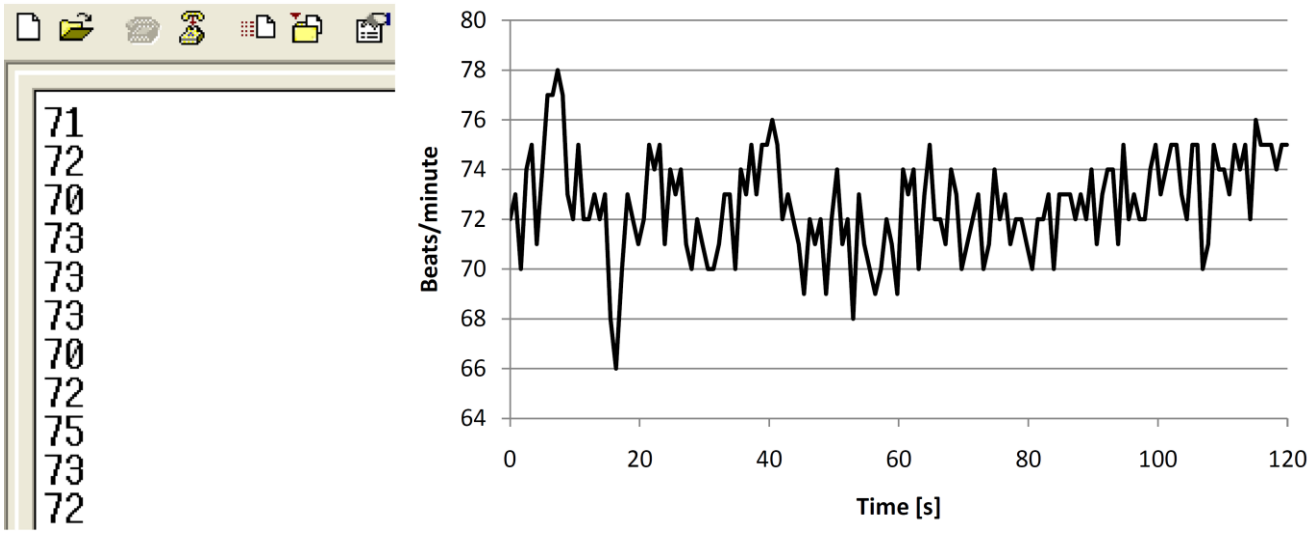

Fig. 11. The microcontroller performs the digital high-pass filtering, heart beat detection and sends the calculated heart rate values for each beat. The data can be received from the microcontroller using Hyperterminal (see the screenshot on the left hand side) software and can be copied into a spreadsheet program to make the plot of data.

The host computer software

The host computer should be able to read and display data continuously to capture the signal coming from the sensor. As it was pointed out, the simplest way is to use a terminal software like Hyperterminal to control the system and retrieve and display data. After setting up the communication port (the provided code in the Appendix assumes 115200 bit/s, 8 data bits, no parity, no handshaking), the students can easily check if the system is working, since pressing any key will cause the corresponding character to be sent to the system, and it will respond with sending the same character back to the personal computer (see the *while* loop in the *main()* function of the source code in the Appendix). The raw data, filtered data and heart beat detection modes can be started by pressing 'r', 'f' and 'b', respectively. In all modes the microcontroller will send the numbers representing the selected data: the digitised sensor data, the digitally filtered sensor data or the

instantaneous heart rate (beats per minute). The data can be logged into a text file or can be copied into the clipboard to allow displaying or further processing using the students' favourite spreadsheet software.

Depending on the level of education the lecturer may ask the students to write their own programs to receive and plot the data even in real time. Experience in serial communication basics and real-time graphical data plotting with automatic scaling, zooming might be required; therefore, such a code may have too high degree of complexity if it is to be written in Java, C++ or C#.

However a powerful virtual instrumentation environment like LabVIEW can be easily used to do the job and Matlab also has easily accessible serial communication and data displaying capabilities, therefore these tools are recommended to develop more advanced applications.

Conclusion

In this paper a mixed-signal MCU based finger photoplethysmograph laboratory experiment has been presented. This greatly transparent experiment helps students to see an example how a modern analogue-intensive MCU allows direct sensor interfacing, eliminates the need for extensive analogue signal conditioning and improves flexibility, integration and reliability. The key points are listed below.

- Students have their MCU training by making an inspiring and exciting cardiovascular experiment; they measure their own blood volume pulsations.
- The experiment helps students to understand the basic principles of operation of modern healthcare devices, signal conditioning and embedded processing.
- Besides laboratory practice, real time demonstrations in lectures are supported as well.
- Most part of the processing takes place in the digital domain, students make most of the whole system themselves, this way they can understand the job better. They learn about sampling, delta-sigma ADC basics, digital signal processing, embedded and host computer programming.
- The low price allows students to have a separate kit, therefore every student do the same experiment in the lab. The lecturer can introduce the basics, can easily help students, draw their attention to the specific problems while more skilled students are free to go faster, to do and learn more.
- The experiment is scalable: it is suitable for beginners, while allowing more experienced students to exploit their creativity in developing more advanced code, playing with the processing parameters. It is also easy to add new tasks including heart rate average and variability calculations, beat-to-beat statistical analysis, frequency domain analysis, estimating breathing frequency and many more[11]. Adding hardware like an alphanumeric LCD display is also easy.
- The low cost hardware and free development tools are available worldwide, students can easily install one at home and can use it for many other problems. Additionally, university education is supported by the manufacturer of the hardware.
- The experiment has been tested and demonstrated. By sharing the associated code and information on a dedicated web page, we made it fully open source and easily reproducible[6].
- The experiment can be easily ported to other MCU platforms. The main requirement is the presence of a wide dynamic range low noise ADC. Candidates with high resolution integrated analogue components include the PIC18F87J72 8-bit MCU, the MSP430FG47x 16-bit MCU and the ADuC7060 32-bit ARM core MCU.
- Similar experiments can be carried out with other small output signal sensors which can also be interfaced directly to precision analogue MCUs. Thermocouples, pressure sensors, load cells, magnetic field sensors, strain gauges are good examples. These broaden the education applications even further[14].

Acknowledgment

The author thanks Peter Makra for helpful discussions.

References

1 A. L. Lee, Albert J. Tahmoush and J. R. Jennings, 'An LED-Transistor Photoplethysmograph', *IEEE Trans. Biomedical Engineering*, **BME-22** (3) (1975) 248-250
2 K. A. Reddy, J. Rezuana Bai, B. George, N. M. Mohan and V. J. Kumar, 'Virtual Instrument for the Measurement of Haemo-dynamic Parameters Using Photoplethysmograph', in *Proc. IEEE Instrumentation and Measurement Technology Conference*, (2006) 1167-1171
3 K. Ashoka Reddy, Boby George, N. Madhu Mohan, Student Member and V. Jagadeesh Kumar, 'A Novel Calibration-Free Method of Measurement of Oxygen Saturation in Arterial Blood', *IEEE Trans. Instrumentation And Measurement*, **58** (5) (2009) 1699-1705
4 K-I. Wong, N. Barsoum and C. Z. Myint, 'Teaching the Electronic Design and Embedded System Course with Body Sensor Nodes', *Proc. 2nd International Conference on Education Technology and Computer (ICETC)*, (2010) 158-162
5 S. Rhee, B.-H. Yang and H. H. Asada, 'Artifact-Resistant Power-Efficient Design of Finger-Ring Plethysmographic Sensors', *IEEE Trans. Biomedical Engineering*, **48** (7) (2001) 795-805
6 http:// www.inf.u-szeged.hu/noise/edudev/PPGonF350
7 Application note AN2944, 'Plethysmograph based on the TS507', *STMicroelectronics*, http://www.st.com
8 http://www.silabs.com
9 http://sdcc.sourceforge.net
10 J. Allen and A. Murray, 'Effects of filtering on multi-site photoplethysmography pulse waveform characteristics', *Computers in Cardiology*, **31** (2004) 485-488,
11 K. H. Shelley, 'Photoplethysmography: Beyond the Calculation of Arterial Oxygen Saturation and Heart Rate', *Anesthesia & Analgesia*, **105** (6) (2007) S31-S36
12 Walt Kester, ed. 'Data Conversion Handbook', Newnes, (2004), ISBN-13: 978-0750678414, http://www.analog.com/library/analogDialogue/archives/39-06/data_conversion_handbook.html
13 Steven W. Smith, 'The Scientist & Engineer's Guide to Digital Signal Processing', California Technical Pub.; 1st edition (1997), ISBN-13: 978-0966017632, http://www.dspguide.com/
14 Z. Gingl, 'Fabricate a high-resolution sensor-to-USB interface', EDN, **56** (2011) 54-57, http://www.inf.u-szeged.hu/~gingl/edaq24/

Appendix

SDCC compatible sample source code for the C8051F350 microcontroller used to demonstrate the experiment.

```
#include <stdio.h>
#include <stdlib.h>

#include "c8051f350.h"

// Peripheral specific initialization functions,
// Called from the Init_Device() function
void PCA_Init()
{
    PCA0MD    &= ~0x40;                    // disable watchdog timer
    PCA0MD    = 0x00;
}

void Timer_Init()
```

```
{
    TCON    = 0x40;
    TMOD    = 0x20;
    CKCON   = 0x08;                 // Timer base = system clock for high resolution
    TH1     = 0xCB;                 // 115200 bit/s (closest match)
}

void UART_Init()
{
    SCON0   = 0x10;                 // 8-bit, variable baud mode
}

// Configure the ADC:
// 100Hz sample rate, unipolar mode
// gain=1, external voltage reference
// analog input buffers are bypassed
// ADC input is AIN7 and GND
void ADC_Init()
{
    ADC0CN  = 0x00;                 // disable ADC, will be enabled later
    ADC0CF  = 0x04;
    ADC0MD  = 0x80;
    ADC0CLK = 0x04;
    ADC0DECH = 0x00;
    ADC0DECL = 0xBE;
    ADC0MUX = 0x78;
}

void Port_IO_Init()
{
    P1MDIN  = 0xBF;                 // P1. is analog
    P0MDOUT = 0x90;                 // P0.4 (UART TX) and P0.7 (LED) push-pull
    P1SKIP  = 0x40;
    XBR0    = 0x01;
    XBR1    = 0x40;
}

void Oscillator_Init()
{
    OSCICN  = 0x82;                 // internal oscillator, 12.25MHz
}

void Init_Device(void)              // Initialization function for the device
{
    PCA_Init();
    Timer_Init();
    UART_Init();
    ADC_Init();
    Port_IO_Init();
    Oscillator_Init();
}

unsigned char SIn(void)             // serial input function
{
    while (!RI);                    // wait for a byte
    RI=0;
    return SBUF;                    // return the byte
}
```

```
// serial output function
void SOut(char a)
{
    TI=0;
    SBUF=a;                         // transmit a byte
    while (!TI);                    // wait for end of transmission
}

void putchar(char c)
{
    SOut(c);
}

void CalibrateADC(void)
{
    ADC0MD = 0x81;                  // full internal calibration
    while(AD0CBSY);                 // end of cal step
    while(ADC0MD & 0x07);           // wait return to IDLE!!!
    AD0INT=0;
}

// Do continuous sampling
// every sample will be sent to the host
void RawSensorData(void)
{
    unsigned char c;
    unsigned int x;

    CalibrateADC();                 // ADC calibration is required
    c=0;                            // initialize UART input variable
    ADC0MD = 0x83;                  // enable ADC
    do {
        while(!AD0INT);             // wait for end of conversion
        AD0INT=0;                   // clear flag
        x=(ADC0H<<8) | ADC0M;       // most significant 16-bits are enough
        printf("%u\r",x);           // send to PC
        if (RI) c=SIn();            // check if character received
    } while (P1_0 && c!=27);        // stop if button is pressed or ESC received
    ADC0MD = 0x80;                  // disable the ADC
}

// Do continuous sampling, apply digital high-pass filtering
// every sample will be sent to the host
void FilteredSensorData(void)
{
    unsigned char c;
    int x,xprev;
    int y;

    CalibrateADC();                 // ADC calibration is required
    xprev=0;                        // initial value of data
    y=0;                            // initial value of filtered data
    c=0;                            // initialize UART input variable
    ADC0MD = 0x83;                  // enable ADC
    do {
        while(!AD0INT);             // wait for end of conversion
        AD0INT=0;                   // clear flag
```

```
            x=(ADC0H<<8) | ADC0M;          // most significant 16-bits are enough
            y=((x-xprev)*248L+y*240L)>>8;  // do digital high-pass filtering
            xprev=x;                       // shift data for digital filtering
            printf("%d\r",y);              // send to the PC
            if (RI) c=SIn();               // check if character received
      } while (P1_0 && c!=27);             // stop if button is pressed or ESC received
      ADC0MD = 0x80;                       // disable the ADC
}

// Do continuous sampling, high-pass filtering
// detect heart beats and flash the LED accordingly
// send calculated heart rate t the host
void HeartBeatDetection(void)
{
      unsigned char c;
      int x2,x1;
      int y,threshold1,threshold2;
      unsigned char b,t;

      threshold1=10;                       // lower threshold
      threshold2=20;                       // upper threshold

      CalibrateADC();                      // ADC calibration is required
      x1=0;                                // initial value of data
      y=0;                                 // initial value of filtered data
      c=0;                                 // initialize UART input variable
      b=0;                                 // bat detection flag, used to implement hysteresis
      t=0;                                 // measures time (number of samples) between beats
      ADC0MD = 0x83;                       // enable ADC
      do {
            while(!AD0INT);                // wait for end of conversion
            AD0INT=0;                      // clear flag
            x2=(ADC0H<<8) | ADC0M;         // most significant 16-bits enough
            y=x2-x1;                       // do digital high-pass filtering
            if (y>threshold2 && !b) {
              b=1;
              printf("%d\r", 6000/t);      // send heart rate (t is in 10ms units)
              t=0;
            }
            if (y<threshold1 && b) b=0;
            P0_7=b;                        // drive LED to visualize heart beats
            x1=x2;                         // shift data for digital filtering
            t++;                           // increment time
            if (RI) c=SIn();               // check if character received
      } while (P1_0 && c!=27);             // stop if button is pressed or ESC received
      ADC0MD = 0x80;                       // disable the ADC
}

void main(void)
{
      unsigned char i;

      Init_Device();

      while (1)                            // loop forever
      {
            i=SIn();                       // wait for a byte
            SOut(i);                       // return to the host for "software handshaking"
```

```
            if (i=='r') {                   // if 'm' is received, start measurement
               RawSensorData();             // detect hear beats, stream data to host
            }
            if (i=='f') {                   // if 'm' is received, start measurement
               FilteredSensorData();        // detect hear beats, stream data to host
            }
            if (i=='b') {                   // if 'm' is received, start measurement
               HeartBeatDetection();        // detect hear beats, stream data to host
            }
      }
}
```