DIGITAL ACQUISITION DEVICE WITH COMPUTER INTERFACE SUITABLE FOR ANALOG ELECTROCARDIOGRAPH FRONT-END

DIGITALA PROJEKT, GRUPP 1

LÄRARE: BERTIL LINDVALL

STUDENTER: GIUSEPPE LIPPOLIS
FRANCESCO PALMISANO

INTRODUCTION

The idea underlying the realization of the project is the attempt to create an analog to digital front-end device able to acquire an analog signal from the external environment and to sample it in order to have the information in a format suitable for following treatments and elaborations.
Specifically the case of study concerns the acquisition of a biological signal and in particular of a signal generated by the human body like the electrocardiographic one. This represents the electric activity (as a cause of the mechanical activity we can easily be aware of) of the heart, that is, the waves of polarization and depolarization involving the myocardial cells that can be detected by some electrodes put on appropriate places of the body surface.
It turns out that the first stage of the work which anyway goes over the requirements and the aims of the project should be the correct acquisition of this signal through specific analog equipment like an ECG device able to "extract" the signal and to present it after an early analog filtering and amplification.
Leaving the inner problems of this first stage aside, the front-end whose core is a microcontroller, thanks to its internal AD converter, samples the signal and produces a digital one. The sampling rate chosen is 500 Hz, enough to have an optimal band of the signal of circa 250 Hz (indeed this is only an ideal limit to prevent aliasing, the real value should be circa 200 Hz). The microcontroller is programmed through a firmware written in C code in order to handle this signal and extract some information from it, like for example the heart frequency, which is visualized through an LCD display. The information is also sent to a second processing station, like for example a PC which can exploit its larger computing power to storage or to further elaborate the signal and highlight some other features by implementing different algorithms of the signal processing theory.

HARDWARE

The components that have been used are:

- Atmega16 AVR microcontroller running at 1 Mhz, internal clock
- Sharp LCD display
- Max232 voltage level converter
- Personal computer equipped with COM port

The Atmega16 has already an internal ADC that represents perhaps the most important part of the system. It gives the possibility to sample an analog signal with a 10 bits precision, but for the specific purposes it has been enough an 8 bits precision that allows us to have 256 different levels to sample the signal. Considering that the electrocardiographic signal coming out from the analog equipment has amplitude of circa 2 volt and the voltage reference is 2.56, the least significant bit is equal to 0.01 V. The signal source has been connected to the pin 0 of the port A of the atmega16.
The LCD display gives the possibility to show important information. The fulfilled implementation produces as result the most common parameter that is the heart frequency. Of

course it can be decided to visualize whichever parameter we consider significant. The LCD display has been connected to the port B and D to receive data and commands respectively.
As it often happened the use of general-purpose processors like the PCs can let a series of other elaborations of the signal such as filtering for noise reduction or for extraction of other important parameters through for example frequency domain methods. A PC has much more computing power than a microcontroller, which on the other hand is the optimal choice for signal acquisition and sampling, thanks to its predictable behavior. That is way the microcontroller has been connected to the PC through the USART peripheral, which allows a serial communication (transmitting and receiving). The connection is asynchronous, that is, no clock signal is transmitted from the Atmega16 to the PC and the two devices need a stable clock in order to communicate effectively.
The USART interface has been used together with the module max232, which is the responsible for the conversion of the voltage levels. In fact the logic 0 and 1 values in output from the microcontroller are in fact 0 and 5V. In order to be connected to the computer the have to be translated to -10 and +10 to represent 0 and 1 respectively. The max232 is connected to the pins RX and TX of the microcontroller (pin 0 and pin 1 of the port D) and to the pins of the RS232 connector.

FIRMWARE

The program is structured in many function used to initialize and control the peripherals. The flow of the program, described in the *main* method, has basically 2 parts: the initialization of the peripherals (LCD display, USART and ADC) and the execution of the specific tasks.
The LCD display is set to work in 8 bits mode on 2 lines and with 5x7 font size. The cursor is set to be incremented automatically when a character is written on the display. When the initializing commands are passed to the LCD display, we have to wait until the busy flag of the display is ok.
The ADC settings include the ADMUX and ADCSRA registers. The ADMUX is set to sample only one signal from the pin 0 of the port A with an internal reference of 2.56 V and to use 8 bits. The ADCSRA is set in order to have the interrupt enable and with a prescaler of clock/8. Moreover the interrupt bit of the SREG register is enabled in order to enable the interrupt service routine for the ADC.
We will use a single ended mode conversion, that is, once the ADC is enabled it will sample the signal only one time. We have to start the ADC periodically in order to sample the signal at the desired rate. When the data is converted and is ready in the ADCH register, an interrupt is called and the interrupt service routine for the ADC is loaded.
The USART is initialized in 8 bits mode with 1 stop bit and no parity bits. We have to be very careful in setting the speed of the connection since it is asynchronous and it requires a precise divider of the clock frequency. Since the sampling rate of the ADC should be set at no more than 500 Hz and we acquire 8 bits per sample, we do not require the USART to run faster than 4800 bps. With a clock frequency of 1 Mhz, we get an error of 0.2%.
Once all the peripherals are initialized the second block can begin: it is the infinite cycle where the microcontroller will execute its function, what it is programmed for. Every time the code comes back to the beginning of this cycle, the ADC is set to start the conversion of the analog value. Hence the interrupt service routine is executed and the result of the conversion is put in the variable *ad_data* and sent in the register UDR of the USART. The program waits until the UDR is ready to write data inside it and this is the main timing process of the program since the major restriction is the connection speed set to 600 bytes/second.

Through the value of a flag variable called *go*, the entire execution of the program is controlled by the interrupt service routine of the ADC. It means that when an ADC conversion is launched the program cannot start another conversion until the previous data has been converted and sent through the serial interface.
After the start of the ADC conversion comes the part of the code that acts in order to measure the period of the signal. Since the ECG has a quite stable, periodic and recognizable shape, it is helpful to identify a reference point whose repetition during the time can give the information of the period of this signal. The experiment performed on a simple sinusoidal or a saw-tooth signal clearly proves the situation.
In the case of the ECG the QRS complex (representing the depolarization of the ventricles) is the most recognizable part of the signal and the R wave has the maximum amplitude. For this reason the lapse of time between an R peek and the following one is chosen to represent the period of the signal. As a consequence the heart frequency has been calculated as its reciprocal. Of course at least 2 problems occur in the fulfillment of the process: the first is the need to create a "time axis" in order to create a time reference which otherwise would not be explicit. What we can get in fact is the number of samples between two R peeks (the variable *count* has this function). By means of experimental settings, a "conversion constant" has been estimated and this seems to relate in the proper way the number of samples and the actual time expressed in seconds. That is way the frequency *freq* equals *1/(RR*const).*
The second is a problem of different nature and is related to the unavoidable presence of noise that vitiates the ECG signal. The detrimental consequence is clear: if we had simply decided that the occurrence of an R event is registered when the signal reaches certain voltage amplitude with certain slope (positive for example), we could have had an error if an unpredictable noise had generated the same conditions even not being a real R event. Setting two thresholds as it has been done in the code can overtake this kind of problem. The counting starts only after the signal has encountered the first and the second threshold in a row and stops after the repetition of these two events.
Besides it has been decided to update the value of the frequency measured by calculating its value every 1200 cycles (equal to c.a. 2 seconds) by means of a counter *j* and an *if condition*. This value is sent to the LCD display to update the previous one.

THE PC INTERFACE

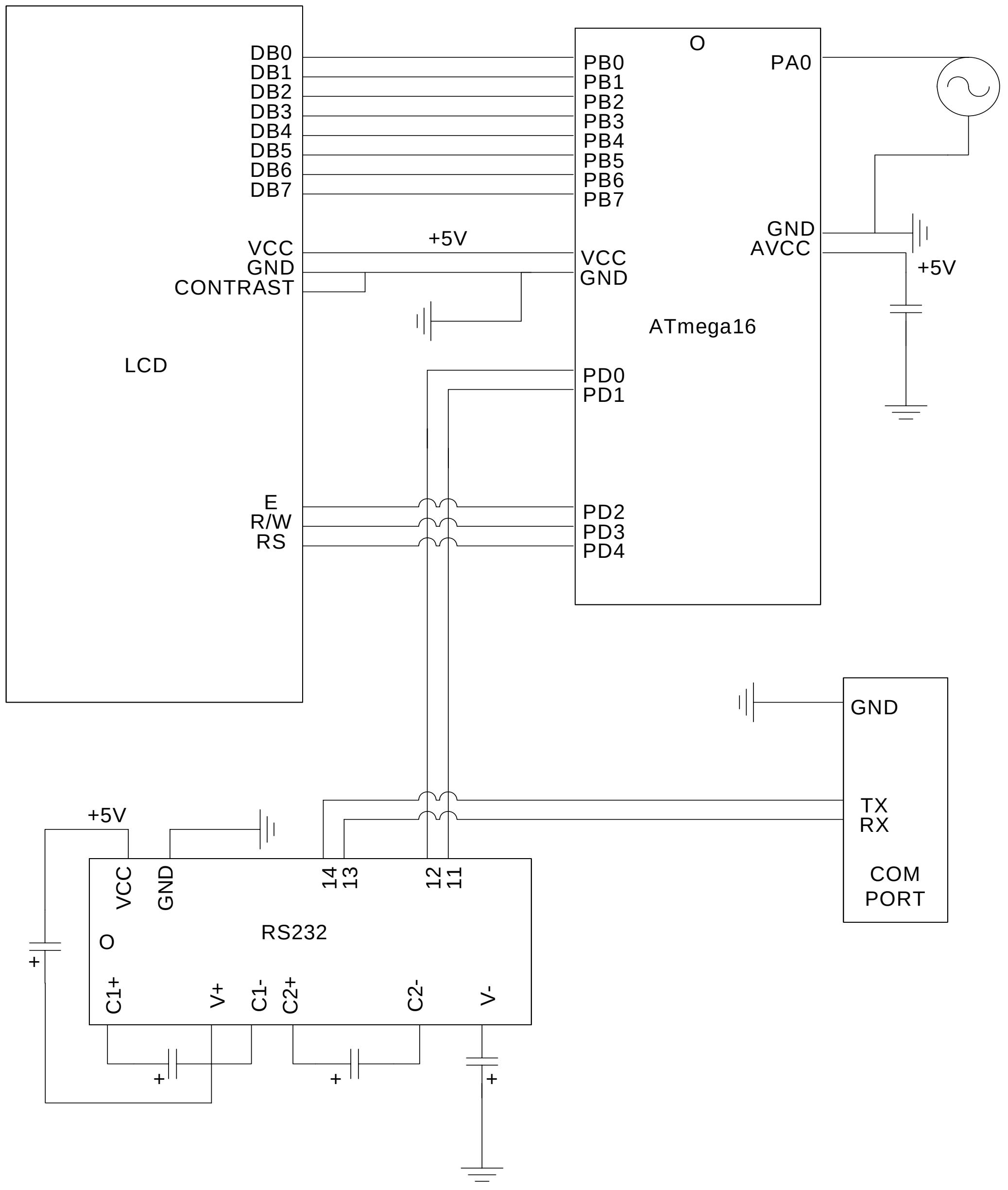
The PC application interface uses an external library to control the serial port COM1. The application initializes the peripheral and starts a cycle in which it acquires the data (one byte per cycle) from the serial port and plots it on a graphic control.
The graphic window is designed to show the signal in a large size to be easily analyzed by the physicians. A period of a heart signal of 60 beats per minute occupies the entire graphic window.

DISCUSSION

The most critical part of the project is to coordinate and temporize the different parts of the system. Evidently the main part and the reference for the other is the ADC. But, since the USART has typically a slower bit rate than the ADC, it has been used as general timing of the

program. This is perfect for our purposes since we need to sample at no more than 500 Hz, each sample is 8 bit and the USART is set to a bit rate of 4800 bps (600 bytes per second).
As we said before in this way we are limiting the band of the signal to 250 Hz: other applications, like the acquisition of high frequency signals, could have required a different timing strategy.
A second point is that when we write the frequency on the LCD display we lose some time in which we do not sample and send data through the USART. This problem is not so important when the sampled signal has low frequency but could become a serious problem increasing the frequency.
A third point is that, since we are using the internal clock of the microcontroller, we have not a perfectly stable reference to use in the serial communication. Naturally greater accuracy can be achieved by using an external oscillator.

APPENDIX 1: SCHEMATICS



APPENDIX 2: FIRMWARE CODE

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>

//RS high
#define   RShigh  0b00010000
//RW high
#define   RWhigh  0b00001000
//E high
#define   Ehigh  0b00000100

//time constant for detecting RR time distance
#define cost 0.002001

short ad_data;
int j=0;
int count=1;
double freq;
int bpm=1;
short flag=0;
int RR=1;
short go=1;

// interrupt service routine for the ADC
ISR(ADC_vect)
{
  // take the data from the ADCH register
  ad_data=ADCH;
  // Do nothing until UDR is ready for more data to be written to it
  while ((UCSRA & (1 << UDRE)) == 0)
  {}
  // Put the acquired data in the UDR register and send it
  UDR = ad_data;
  go=1;
}

// wait until the for cycle is complete
void wait(int times)
{
  int i;
  for(i=0;i<times;i++)
  {}
}

// initialize the ADC
void inADC(void)
{
  ADMUX = 0b11100000;
  ADCSRA = 0b10001011;
}

// start the conversion
void startADC(void)
{
  ADCSRA = 0b11001011;
}

// initialize the USART
```

```
void inUSART(void)
{
  UCSRA = 0;
  // Turn on the transmission and reception circuitry
  UCSRB |= (1 << RXEN) | (1 << TXEN);
  // Use the 8 bit mode  - URSEL bit set to select the UCRSC register
  UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);
  // Put the lower 8 bits of the BRV in the UBRRL register
  UBRRL = 12;
  // Put the upper 8 bits of the BRV in the UBRRH register
  UBRRH = (12 >> 8);
}

// write a char to the LCD display
void writeLCD(char data)
{
  DDRB = 0xFF;
  PORTB = data;

  //put RW low and E low, leave RS in its state
  PORTD = PORTD & ~(RWhigh | Ehigh);

  //put RS high and E high, leave RW in its state
  PORTD = PORTD | (RShigh|Ehigh);

  wait(5);

  //put RS low and E low, leave RW in its state
  PORTD = PORTD & ~(RShigh | Ehigh);
}

// take the flag from the LCD display through the port B
char flagLCD(void)
{
  char address;
  DDRB = 0x00;
  //put RW high and E high, leave RS in its state
  PORTD = PORTD | (RWhigh | Ehigh);

  wait(3);

  // get the value the flag through the B port
  address = PINB;

  //put RW low and E low, leave RS in its state
  PORTD = PORTD & ~(RWhigh | Ehigh);

  return address;
}

// wait until DB7 is high (LCD display ready)
void waitflag(void)
{
  while((flagLCD() & 0x80) == 0x80) {}
}

// set a command to the LCD display
void commandLCD(char command)
{
  DDRB = 0xFF;
  PORTB = command;
  PORTD = PORTD & ~(RShigh | RWhigh | Ehigh);
  PORTD = PORTD | Ehigh;
  wait(5);
  PORTD = PORTD & ~(Ehigh);
  DDRB = 0;
}

// initialize LCD
void inLCD(void)
{
  DDRA = 0xFE;
  DDRD = 0xFF;
  PORTA = 0x00;
  PORTD = 0x00;

  //8 bit interface, 5x7 font, 2 lines.
  commandLCD(0x38);
  waitflag();

  //display on, cursor on blinking
  commandLCD(0b00001110);
  waitflag();

  //clear the display, cursor home
  commandLCD(0x01);
  waitflag();

  //cursor auto increment
  commandLCD(0x06);
}

// write heart frequency to the LCD display
void stamp(void)
{
  int x;
  char c1=48;
  char c2=48;
  char c3=48;
  int resto1;

  // transform the variable bpm in 3 different char
  for(x=100;x<=bpm;x=x+100)
  {
    c1++;
  }
  resto1=bpm-((c1-48)*100);

  for(x=10;x<=resto1;x=x+10)
  {
    c2++;
  }
  c3=48+resto1-((c2-48)*10);

  writeLCD(c1);
  writeLCD(c2);
  writeLCD(c3);
  commandLCD(0b00010000);
  commandLCD(0b00010000);
  commandLCD(0b00010000);
```

```
}


void main(void)
{
   // LCD display connections with PORTB and PORTD
                      // PB7-DB7, PB6-DB6,PB5-DB5,PB4-DB4,PB3-DB3,PB2-DB2,PB1-DB1, PB0-DB0
   // PD4-RS, PD3-R/W, PD2-E

                      // initialize the LCD display and write to the default string
                      wait(30);
   inLCD();
   writeLCD('H');
   writeLCD('e');
   writeLCD('a');
   writeLCD('r');
   writeLCD('t');
   writeLCD(' ');
   writeLCD('r');
   writeLCD('a');
   writeLCD('t');
   writeLCD('e');
   writeLCD(' ');

   // initialize the ADC
                      inADC();

                      // set to 1 the I bit of the SREG register in order to get an interrupt for the ADC
                      SREG = 0x80;

   // initialize the USART
   inUSART();


                      while(1)
                      {
                        //wait until the ADC interrupt service routine has been completed
                        if(go==1)
                        {
                          // acquire one sample
                          startADC();
                          go=0;

        // every 1200 cycles update the value of frequency
                          if(j == 1200)
                          {
                            j = 0;
                                          // get the heart frequency as the inverse of the time between two R peaks
                            freq = 1/(RR*cost);
                                          bpm = freq*60;
                                          // update the heart frequency on the LCD display
                                          stamp();
                           }

        // when the voltage is > 150 we are in the QRS complex
                          if (ad_data>150)
        {
                                          flag=1;
                          }
        // when the voltage is < 120 and it was > 150 it is sure that we detected a QRS pattern
        if((ad_data<120) & (flag==1))
                          {
                            RR=count;
                            count=0;
                flag=0;
                          }

                          // count how many samples there are from a R peak in the QRS pattern to the other one
                          count++;
                          j++;
                         }
                      }

   return;
}
```

REFERENCES

Atmega16 manual
Sharp LCD display manual
Some code for the LCD initialization has been taken from the example provided by *http://www.avrbeginners.net*
Some code for the USART initialization has been taken from the example provided by *http://www.avrfreaks.net*