Appendix 1

Code vision code:

```
/*****************************************************
This program was created by the
CodeWizardAVR V3.14 Advanced
Automatic Program Generator
© Copyright 19982014- Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    :    122023/31/
Author  :
Company :
Comments:


Chip type                : ATmega32A
Program type             : Application
AVR Core Clock frequency : 8.000000 MHz
Memory model             : Small
External RAM size        : 0
Data Stack size          : 512
*****************************************************/

#include <mega32a.h>

#include <delay.h>

// Declare your global variables here

#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<DOR)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE <= 256
unsigned char rx_wr_index=0,rx_rd_index=0;
#else
unsigned int rx_wr_index=0,rx_rd_index=0;
```

```
#endif

#if RX_BUFFER_SIZE < 256
unsigned char rx_counter=0;
#else
unsigned int rx_counter=0;
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer[rx_wr_index++]=data;
#if RX_BUFFER_SIZE == 256
   // special case for receiver buffer size=256
   if (++rx_counter == 0) rx_buffer_overflow=1;
#else
   if (rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
   if (++rx_counter == RX_BUFFER_SIZE)
      {
      rx_counter=0;
      rx_buffer_overflow=1;
      }
#endif
   }
}

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index++];
#if RX_BUFFER_SIZE != 256
if (rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#endif
#asm("cli")
--rx_counter;
#asm("sei")
```

```
return data;
}
#pragma used-
#endif

// USART Transmitter buffer
#define TX_BUFFER_SIZE 8
char tx_buffer[TX_BUFFER_SIZE];

#if TX_BUFFER_SIZE <= 256
unsigned char tx_wr_index=0,tx_rd_index=0;
#else
unsigned int tx_wr_index=0,tx_rd_index=0;
#endif

#if TX_BUFFER_SIZE < 256
unsigned char tx_counter=0;
#else
unsigned int tx_counter=0;
#endif

// USART Transmitter interrupt service routine
interrupt [USART_TXC] void usart_tx_isr(void)
{
if (tx_counter)
   {
   --tx_counter;
   UDR=tx_buffer[tx_rd_index++];
#if TX_BUFFER_SIZE != 256
   if (tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
#endif
   }
}

#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the USART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")
if (tx_counter || ((UCSRA & DATA_REGISTER_EMPTY)==0))
   {
   tx_buffer[tx_wr_index++]=c;
#if TX_BUFFER_SIZE != 256
   if (tx_wr_index == TX_BUFFER_SIZE) tx_wr_index=0;
#endif
   ++tx_counter;
```

```
   }
else
   UDR=c;
#asm("sei")
}
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>

// Voltage Reference: AREF pin
#define ADC_VREF_TYPE ((0<<REFS1) | (0<<REFS0) | (0<<ADLAR))

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | ADC_VREF_TYPE;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=(1<<ADSC);
// Wait for the AD conversion to complete
while ((ADCSRA & (1<<ADIF))==0);
ADCSRA|=(1<<ADIF);
return ADCW;
}
    long int adc;
    float v;

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2) | (0<<DDA1) |
(0<<DDA0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3) | (0<<PORTA2) |
(0<<PORTA1) | (0<<PORTA0);

// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=Out
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) | (0<<DDB1) |
(1<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=0
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) |
(0<<PORTB1) | (0<<PORTB0);
```

```
// Port C initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) | (0<<DDC1) |
(0<<DDC0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) |
(0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) |
(0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) |
(0<<PORTD1) | (0<<PORTD0);

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=0xFF
// OC0 output: Disconnected
TCCR0=(0<<WGM00) | (0<<COM01) | (0<<COM00) | (0<<WGM01) | (0<<CS02) | (0<<CS01) | (0<<CS00);
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=0xFFFF
// OC1A output: Disconnected
// OC1B output: Disconnected
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11) | (0<<WGM10);
TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (0<<CS12) | (0<<CS11) | (0<<CS10);
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
```

```
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=0<<AS2;
TCCR2=(0<<PWM2) | (0<<COM21) | (0<<COM20) | (0<<CTC2) | (0<<CS22) | (0<<CS21) | (0<<CS20);
TCNT2=0x00;
OCR2=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=(0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) | (0<<OCIE1B) | (0<<TOIE1) |
(0<<OCIE0) | (0<<TOIE0);

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=(0<<ISC11) | (0<<ISC10) | (0<<ISC01) | (0<<ISC00);
MCUCSR=(0<<ISC2);

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600 (Double Speed Mode)
UCSRA=(0<<RXC) | (0<<TXC) | (0<<UDRE) | (0<<FE) | (0<<DOR) | (0<<UPE) | (1<<U2X) | (0<<MPCM);
UCSRB=(1<<RXCIE) | (1<<TXCIE) | (0<<UDRIE) | (1<<RXEN) | (1<<TXEN) | (0<<UCSZ2) | (0<<RXB8) |
(0<<TXB8);
UCSRC=(1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) | (0<<USBS) | (1<<UCSZ1) | (1<<UCSZ0) |
(0<<UCPOL);
UBRRH=0x00;
UBRRL=0x67;

// Analog Comparator initialization
// Analog Comparator: Off
// The Analog Comparator's positive input is
// connected to the AIN0 pin
// The Analog Comparator's negative input is
// connected to the AIN1 pin
ACSR=(1<<ACD) | (0<<ACBG) | (0<<ACO) | (0<<ACI) | (0<<ACIE) | (0<<ACIC) | (0<<ACIS1) |
(0<<ACIS0);

// ADC initialization
// ADC Clock frequency: 62.500 kHz
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: ADC Stopped
ADMUX=ADC_VREF_TYPE;
```

```
ADCSRA=(1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2) | (1<<ADPS1) |
(1<<ADPS0);
SFIOR=(0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);

// SPI initialization
// SPI disabled
SPCR=(0<<SPIE) | (0<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) | (0<<CPHA) | (0<<SPR1) |
(0<<SPR0);

// TWI initialization
// TWI disabled
TWCR=(0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWEN) | (0<<TWIE);

// Global enable interrupts
#asm("sei")

while (1)
      {
      adc = read_adc(0);
      v = adc*5.01023/;
      printf("%f\r\n", v);
      PORTB.0=1;

      }
}
```

Appendix 2

Matlab:

```
clc; clear all; close all;
Samples=60e3;
FirstAndLastData=[100,Samples/10-100-1];
k=8;
for i=1:1:12
    if i<=6
SaveFileName=['DataReport ',num2str(i),'.txt'];
[CleanedData,Time] = DataReader(Samples,FirstAndLastData,SaveFileName);
LData=length(CleanedData);
PCTimer=clock;b=['Date: ',num2str(PCTimer(1)),'/',...
    num2str(PCTimer(2)),'/',num2str(PCTimer(3)),...
    newline,'EndTime: ',num2str(PCTimer(4)),':',...
    num2str(PCTimer(5)),':',num2str(PCTimer(6))];
Data2Excell(1,i) =  {['RawData ',num2str(i),newline,...
    'ReceivingTime: ',num2str(Time),' Second',newline,b]};
Data2Excell(2:LData+1,i)=mat2cell(CleanedData,ones(LData,1));
writecell(Data2Excell,'Data.xls','Sheet',k);
    end
    if i>6
        if i==7
        TCF='on'
        end
SaveFileName=['DataReport ',num2str(i),'.txt'];
[CleanedData,Time] = DataReader(Samples,FirstAndLastData,SaveFileName);
LData=length(CleanedData);
PCTimer=clock;b=['Date: ',num2str(PCTimer(1)),'/',...
    num2str(PCTimer(2)),'/',num2str(PCTimer(3)),...
    newline,'EndTime: ',num2str(PCTimer(4)),':',...
    num2str(PCTimer(5)),':',num2str(PCTimer(6))];
Data2Excell(1,i+2) =  {['RawData ',num2str(i),newline,...
    'ReceivingTime: ',num2str(Time),' Second',newline,b]};
Data2Excell(2:LData+1,i+2)=mat2cell(CleanedData,ones(LData,1));
writecell(Data2Excell,'Data.xls','Sheet',k);
    end


end



function [CleanedData,Time] = DataReader(Samples,FirstAndLastData,SaveFileName)
instrfind;
delete(instrfind)
device = serialport("COM8",9600);
```

```
tic
data = read(device,Samples,'string');
Time = toc;
[token, ~] = strtokPYA(data{1}, '↲↲');
fid = fopen(SaveFileName, 'w');
fprintf(fid, '%s', token);

fclose(fid);
fileID = fopen(SaveFileName,'r');
A = textscan(fileID,'%s');
B=A{1};
C=str2double(B);
D= C(  ~isnan(C) );
CleanedData=D(FirstAndLastData(1):FirstAndLastData(2));
end



function [token, remainder] = strtokPYA(str, delimiters)
%STRTOK Split string into tokens.
%   [TOKEN,REMAIN] = STRTOK(STR) returns the first token in STR delimited
%   by whitespace characters and the rest of STR in REMAIN. STRTOK ignores
%   any leading whitespace. If STR is a cell array of character vectors,
%   TOKEN is a cell array of tokens. If STR is a string array, TOKEN is a
%   string array.
%
%   TOKEN = STRTOK(STR,DELIMITER) returns the first token delimited by one
%   of the characters in DELIMITER. STRTOK ignores any leading delimiters.
%   Do not use escape sequences as delimiters.  For example, use char(9)
%   rather than '\t' for tab.
%
%   If the input does not contain any delimiter characters, STRTOK returns
%   the entire input in TOKEN (excluding any leading delimiter characters),
%   and REMAIN contains text with no characters.
%
%   NOTE: Inputs STR and DELIMITER can be string arrays, character vectors
%   or cell arrays of character vectors. When STR is a string array outputs
%   TOKEN and REMAIN are string arrays. Otherwise TOKEN and REMAIN are cell
%   arrays of character vectors.
%
%   Example:
%
%       s = '  This is a simple example.';
%       [token, remain] = strtok(s)
%
%   returns
%
%       token =
%       This
```

```
%       remain =
%        is a simple example.
%
%   See also SPLIT, extractBefore, extractAfter, extractBetween, REGEXP,
%   ISSPACE, STRFIND, STRCMP, TEXTSCAN


%   Copyright 1984-2016 The MathWorks, Inc.


    if nargin < 1 || nargin > 2
        narginchk(1, 2);
    end

    if nargin < 2
        delimiters = char([9:13, 32]); % White space characters
    elseif iscell(delimiters)
        delimiters = char([delimiters{:}]);
    elseif isstring(delimiters)
        delimiters(ismissing(delimiters)) = [];
        delimiters = char([delimiters{:}]);
    end


    computeRemainder = (nargout > 1);

    if iscell(str)
        token = str;
        remainder = str;
        for idx = 1:numel(str)
            [token{idx}, remainder{idx}] = doStrtok(str{idx}, delimiters, computeRemainder);
        end
    elseif isstring(str)
        token = str;
        remainder = str;
        for idx = 1:numel(str)
            if ismissing(str(idx))
                remainder(idx) = '';
                continue;
            end
            [token{idx}, remainder{idx}] = doStrtok(str{idx}, delimiters, computeRemainder);
        end
    else
        [token, remainder] = doStrtok(str, delimiters, computeRemainder);
    end


end
```

```
function [token, remainder] = doStrtok(str, delimiters, computeRemainder)


    token = str([]);
    remainder = token;


    len = length(str);
    if len == 0
        return;
    end


    idx = 1;
    while (any(str(idx) == delimiters))
        idx = idx + 1;
        if (idx > len)
           return;
        end
    end


    start = idx;
    while (~any(str(idx) == delimiters))
        idx = idx + 1;
        if (idx > len)
           break;
        end
    end
    finish = idx - 1;


    token= str(start:finish);
    if computeRemainder && finish < len
        remainder = str(finish + 1:len);
    end


end
```