Scientific Bulletin of Chełm
Section of Mathematics and
Computer Science
No. 1/2009

POMIAR I PRZETWARZANIE TEMPERATURY Z WYKORZYSTANIEM MIKROKONTROLERA ATMEGA16

BOGUMIŁ ZARZYCKI, MARIUSZ HOLUK

Państwowa Wyższa Szkoła Zawodowa w Chełmie

Streszczenie. Artykuł ma na celu przybliżenie problematyki związanej z pomiarem wielkości analogowych (na przykładzie procesu pomiaru temperatury) oraz programowej implementacji zaprojektowanego algorytmu z wykorzystaniem mikrokontrolera AVR (ATmega16). Proponowany układ przetwarzania może pracować niezależnie, natomiast funkcje wizualizacji przebiegu temperatury oraz obróbki danych pomiarowych zapewnione są przy pomocy dodatkowego oprogramowania zainstalowanego na komputerze klasy PC.

1. Wstęp

Procesy pomiaru i cyfrowego przetwarzania spotykane są dziś niemal na każdym kroku. Choć nie zdajemy sobie często z tego sprawy, codziennie korzystamy z urządzeń, które wykorzystują te funkcje. W większości z nich pracują już urządzenia cyfrowe, jednak jeszcze nie tak dawno były one urządzeniami w pełni analogowymi (np. telewizory, pralki, wzmacniacze, telefony i wiele więcej).

Jakie argumenty zatem przemawiają za układami cyfrowymi i dlaczego są dziś one tak powszechnie stosowane? Powodów jest kilka.

Po pierwsze, analogowe obwody przetwarzania w układach elektronicznych były złożone z zestawu rezystorów, kondensatorów, cewek, a w obwodach pneumatycznych m.in. z zaworów, zbiorników i dysz. Raz zbudowany układ mógł realizować tylko jeden algorytm, a możliwości jego ewentualnego strojenia były niewielkie. Każda zmiana algorytmu działania niosła ze sobą zwykle konieczność przebudowy całego układu, co nie było wygodne ani oszczędne. Natomiast układy cyfrowe do zmiany algorytmu potrzebują jedynie przeprogramowania, które (nie wliczając czasu potrzebnego do napisania nowej wersji oprogramowania) zajmuje do kilkunastu sekund.

Kolejnym argumentem przemawiającym za stosowaniem układów cyfrowych (a może przeciw stosowaniu analogowych) są gabaryty i koszty. Układy analogowe charakteryzujące się wysoką dokładnością były kosztowne w wykonaniu, a przede wszystkim sprawiały problemy przy próbach miniaturyzacji. Układy cyfrowego przetwarzania są właśnie

Key words and phrases. pomiar temperatury, AVR, ATmega16, wizualizacja, przetwarzanie cyfrowe.
Treść artykułu była prezentowana w czasie *VIII Konferencji Informatyki Stosowanej* (Chełm 29 - 30 maja 2009 r.)

jej efektem. Są też tańsze od analogowych, gdyż ich masowa produkcja jest łatwiejsza do wdrożenia.

W końcu, sygnały cyfrowe posiadają bardzo korzystne właściwości: m.in. dużą odporność na zakłócenia i są wygodne w przetwarzaniu.

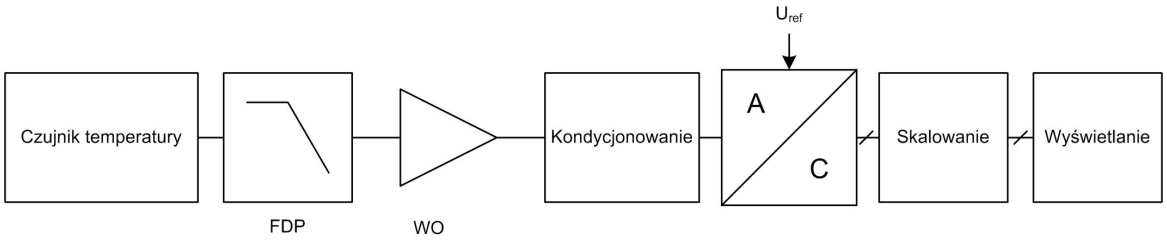
Głównie z tych powodów analogowe przetwarzanie sygnałów jest zredukowane do niezbędnego minimum, m.in. do kondycjonowania sygnałów (przygotowania sygnałów analogowych) dla przetworników analogowo-cyfrowych (np. wzmocnienie małych sygnałów).

Przetwornikiem analogowo-cyfrowym nazywa się urządzenie przetwarzające wielkość analogową w sygnał cyfrowy pomiarowy. Przetwarzaniu analogowo-cyfrowemu towarzyszą na ogół: próbkowanie (dyskretyzacja), kwantowanie (kwantyzacja) oraz kodowanie. Próbkowanie polega na pobieraniu wartości dyskretnych, odpowiadających wartościom chwilowym wielkości ciągłej w określonych chwilach. Kwantowanie wielkości ciągłej polega na przyporządkowaniu każdej próbce skończonej liczby poziomów amplitudy, odpowiadających dyskretnym wartościom od zera do pełnego zakresu. Natomiast kodowanie polega na przypisaniu poziomom reprezentacji wartości w kodzie cyfrowym.

2. Pomiar temperatury i przetwarzanie

Pomiar temperatury w swym założeniu nie różni się zasadniczo od pomiaru innej wielkości analogowej. Cały problem polega na przetworzeniu wielkości nieelektrycznej w elektryczną (w naszym przypadku temperatury na napięcie). Jest to wynikiem tego, że układy cyfrowe nie mierzą bezpośrednio wielkości nieelektrycznych; należy tak je zaprojektować, by rzeczywista wielkość mierzona (napięcie) była w pewnej skali odwzorowaniem wielkości pierwotnej (temperatury).

Istnieją różne układy pomiarowe wykorzystujące konwersję temperatury do wielkości elektrycznej (najczęściej napięcia) i dalszego przetwarzania. Ogólny schemat blokowy układu cyfrowego realizującego taki algorytm znajduje się na rysunku 1.



Rysunek 1. Ogólny schemat blokowy przetwarzania temperatury

Sygnał napięciowy z czujnika jest podawany na filtr dolnoprzepustowy (FDP) w celu wyeliminowania składowych o wyższych częstotliwościach – głównie szumów. Tak odfiltrowany sygnał jest wzmacniany przez wzmacniacz operacyjny (WO) jeśli sygnały są bardzo małe lub w przypadku większych wartości – jedynie odseparowywany od czujnika, by nie powodować zbędnego obciążenia, które mogłoby znacznie pogorszyć wyniki pomiarów. Wzmocniony przebieg kierowany jest następnie do bloku kondycjonowania, gdzie jest formowany w taki sposób, by spełniał wymagania przetwornika A/C. W tym miejscu kończy się droga sygnału analogowego. Napięcie referencyjne U_{ref} doprowadzone do przetwornika służy mu jako punkt odniesienia – po przetworzeniu najmniej znaczący

bit liczby wyjściowej (LSB - *Least Significant Bit*) będzie odpowiadał kwantowi napięcia wejściowego o wartości:

$$Q = \frac{U_{\text{ref}}}{2^n}\,, \tag{1}$$

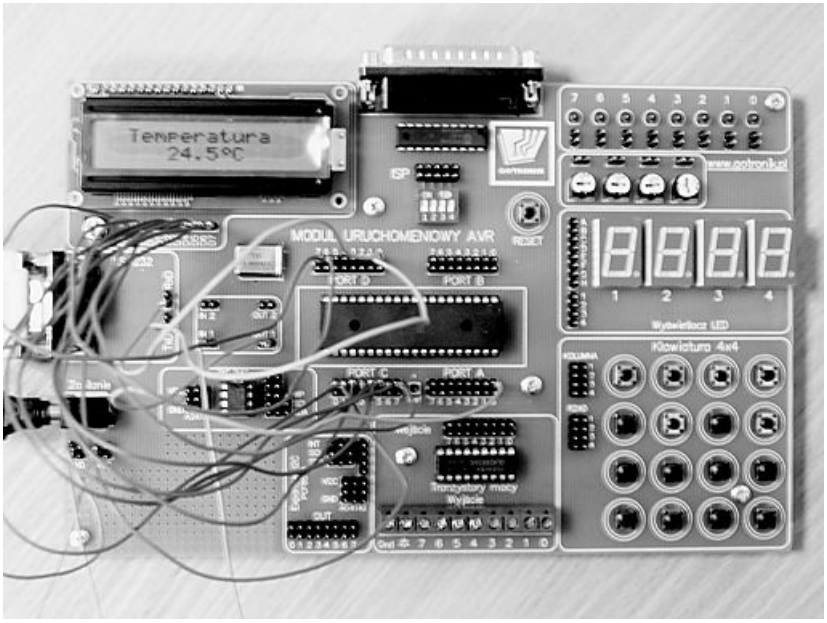
gdzie n – rozdzielczość przetwornika (w bitach).

Przetwornik A/C przeprowadza konwersję napięcia wejściowego U_{we} na postać cyfrową n-bitową, według zależności:

$$\text{ADC} = \frac{U_{\text{we}} \cdot 2^n}{U_{\text{ref}}} = \frac{U_{\text{we}}}{Q} \tag{2}$$

Po konwersji analogowego sygnału napięciowego na sygnał cyfrowy, uzyskana wartość jest odpowiednio przetwarzana przez system mikroprocesorowy – przede wszystkim skalowana programowo w jednostkach wielkości pierwotnej (temperatury) według charakterystyki przetwarzania czujnika. Tak przeskalowana wartość jest wyświetlana do odczytu w sposób zrozumiały dla użytkownika systemu. Dzięki temu zwolniony jest on całkowicie z obowiązku znajomości procesów przetwarzania sygnałów – dostaje od razu wartość, która go interesuje.

3. Opis stanowiska

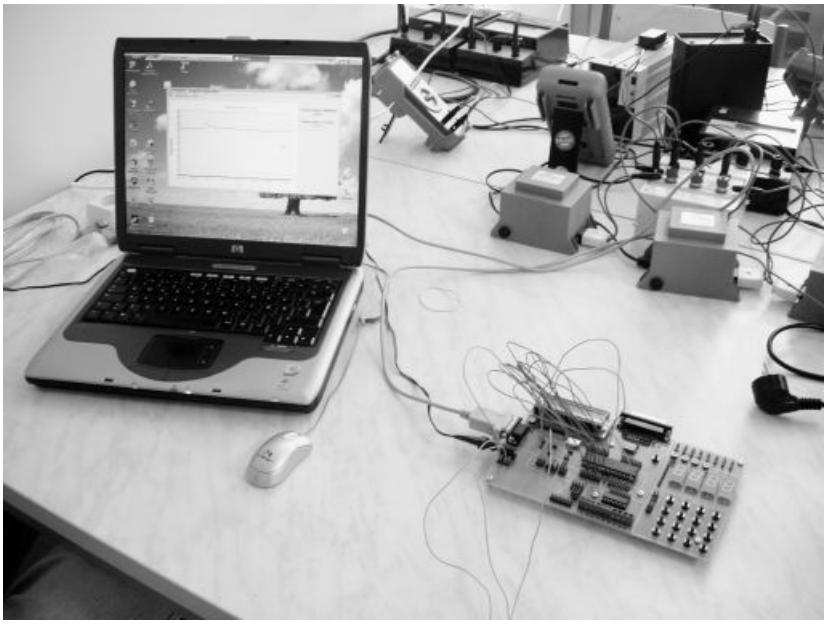


Rysunek 2. Fotografia modułu z mikrokontrolerem

Stanowisko do pomiarów temperatury (Rys. 3) zbudowane zostało [5] w oparciu o moduł uruchomieniowy AVR mogący współpracować z mikrokontrolerami AT90S8535, ATmega8535, ATmega16 oraz ATmega32 firmy ATMEL. W proponowanym układzie pomiarowym wykorzystywane są (Rys. 2):

- mikrokontroler ATmega16 ze zintegrowanym przetwornikiem A/C,
- wzmacniacz operacyjny LM358,
- czujnik temperatury LM35,

- wyświetlacz zgodny ze standardem HD44780,
- konwerter poziomów napięć TTL-CMOS/RS232 MAX232.



Rysunek 3. Stanowisko do pomiaru temperatury

3.1. **Mikrokontroler ATmega16.** Jednym z powodów, dla których mikrokontrolery AVR zyskały popularność, jest łatwy dostęp do niezbędnych narzędzi uruchomieniowych. W szczególności chodzi tu o zintegrowane środowisko rozwojowe (IDE - *Integrated Development Environment*) i urządzenia programująco-testujące.

Wykorzystywany mikrokontroler wyposażony jest w 16kB pamięci programowalnej typu FLASH do przechowywania kodu programu, 1kB statycznej pamięci RAM oraz 512B pamięci EEPROM. Posiada także wiele różnorodnych układów wewnętrznych i peryferyjnych, system obsługi przerwań i układy czasowo-licznikowe, jednakże w tej aplikacji najważniejszą rolę spełnia przetwornik analogowo-cyfrowy.

Wybrane parametry przetwornika A/C:

- rozdzielczość: 10 bitów,
- dokładność bezwzględna: ± 2 LSB
- czas konwersji: $13 \div 260$ μs
- zakres napięć wejściowych: $0 \div V_{cc}$
- możliwość wybrania wewnętrznego napięcia odniesienia 2.56 V
- przerwanie na zdarzeniu zakończenia przetwarzania
- tryb uśpienia Noise Canceler (redukcja szumów)

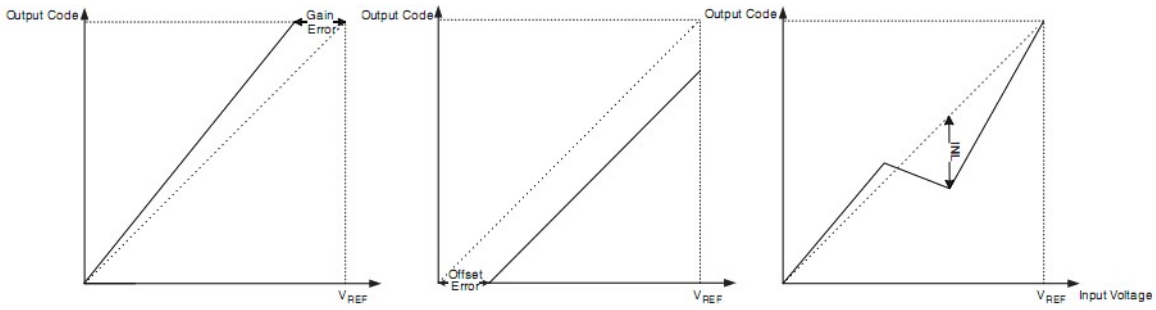
3.2. **Czujnik temperatury.** Jako główny element wykonawczy do pomiaru temperatury zastosowano bezpośredni przetwornik temperatura-napięcie LM35DZ firmy National Semiconductors. Jego najważniejsze parametry są następujące:

- zakres pomiarowy: $0 \div 150$°C
- współczynnik skali: 10 mV/°C
- niski pobór prądu (< 60 μA)

4. Błędy przetwarzania

Każda ingerencja w układ wnosi zakłócenia i pewne błędy pomiaru. Podczas pomiaru temperatury w układzie opartym na mikrokontrolerze ATmega16 współpracującym z czujnikiem temperatury LM35DZ występują błędy:

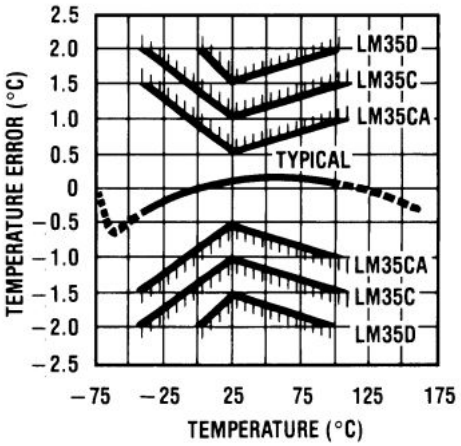
- przetwarzania A/C, wśród których można wyszczególnić - błędy wynikające z rozdzielczości, kwantyzacji, offsetu (przesunięcia zera), nieliniowości oraz wzmocnienia (współczynnika skali)



Rysunek 4. Źródła błędów przetwarzania A/C: a) wzmocnienie, b) offset, c) nieliniowość [2]

Błędy przesunięcia zera są określane jako różnica pomiędzy nominalnym a rzeczywistym położeniem punktu zerowego. Błąd wzmocnienia jest to różnica pomiędzy nominalną maksymalną wartością sygnału wejściowego, a środkową wartością w ostatnim kroku kwantowania.

- błędy czujnika - źródłem których jest nieliniowość charakterystyki przetwarzania temperatury na napięcie, wzmocnienie, wpływ obciążenia, samonagrzewanie



Rysunek 5. Przedstawienie całkowitej dokładności czujnika, będącej wynikiem współdziałania błędów [4]

Na Rys. 5. przedstawiono charakterystykę całkowitej dokładności czujnika względem temperatury otoczenia. Z wykresu można odczytać, że w temperaturze pokojowej maksymalny błąd czujnika LM35DZ wynosi ok. 1,5°C, natomiast przy innych wzrasta on do 2,0°C.

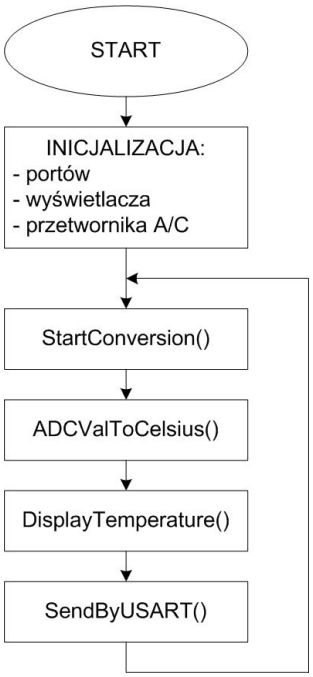
5. Programowanie mikrokontrolera w języku C

Rdzeń rodziny AVR zaprojektowano z myślą o językach wysokiego poziomu konkretnie C. Przy pracach nad układem konsultowano się z ekspertami w dziedzinie implementacji tego języka, czego wynikiem jest elastyczny i wygodny dla programisty rdzeń, którego wydajności nie można wiele zarzucić.

Na potrzeby zaprogramowania mikrokontrolera wybrane zostało środowisko AVR Studio, współpracujące z kompilatorem AVR-GCC. AVR Studio to darmowe środowisko IDE, którego twórcą jest firma ATMEL. Środowisko to tworzą następujące moduły: edytor, translator asemblera, symulator, programator oraz moduł uruchomieniowy (debugger).

5.1. **Algorytm pomiaru.** Program ma na celu obliczenie i wyświetlenie temperatury na ekranie LCD, a następnie wysłanie tej wartości przez interfejs szeregowy. Poszczególne etapy programu przedstawiono następująco:

- Inicjalizacja portów, przetwornika A/C, interfejsu szeregowego, wyświetlacza, układów czasowych, trybu uśpienia
- rozpoczęcie przetwarzania sygnału (wejście w tryb uśpienia - redukcja szumów),
- skalowanie otrzymanej wartości z przetwornika do skali Celsjusza,
- Konwersja wartości liczbowej temperatury do postaci tekstowej,
- Wyświetlenie wartości na wyświetlaczu,
- Przesłanie pakietu z danymi poprzez interfejs szeregowy USART,
- Uaktywnienie opóźnienia czasowego (1s),
- Powrót do punktu 2.



Rysunek 6. Diagram blokowy algorytmu

Opis głównych funkcji algorytmu:

- StartConversion() ustawia mikrokontroler w tryb redukcji szumów dla przetwarzania A/C (jeden z trybów uśpienia) i rozpoczyna konwersję sygnału,
- ADCValToCelsius() przelicza uzyskaną wartość z przetwornika A/C na odpowiadającą jej w skali Celsjusza,
- DisplayTemperature() wyświetla bieżącą wartość temperatury na wyświetlaczu LCD,
- SendByUSART() wysyła pakiet z wartością temperatury poprzez interfejs szeregowy.

5.2. **Listing programu realizującego pomiar temperatury.**

```
#define F_CPU     8000000UL // częstotliwość taktowania CPU
#define NEW_VAL   1
#define SLEEP_REQ 2
#define BAUD_REG  51
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include <stdlib.h>
#include "lcd.h"
unsigned char __attribute__ ((progmem)) text[] =
    {"  Temperatura\0"};
unsigned char __attribute__ ((progmem)) deg[] =
    {0x06,0x09,0x09,0x06,0x00,0x00,0x00,0x00};
    // znak stopnia Celsjusza
unsigned char i;
volatile char state=0;
uint16_t ADCValue=0;
double tempvalue=0.0;
char ttemp[7]="", *ptr;
ISR(ADC_vect)
{
  ADCValue = ADCL;
  ADCValue += (ADCH << 8);
  state |= NEW_VAL;
  state &= ~SLEEP_REQ;
}
ISR(TIMER1_OVF_vect)
{
  TCCR1B &= ~(1<<CS12);
  TCNT1H = 0x7F;
  TCNT1L = 0x00;
  state |= SLEEP_REQ;
}
```

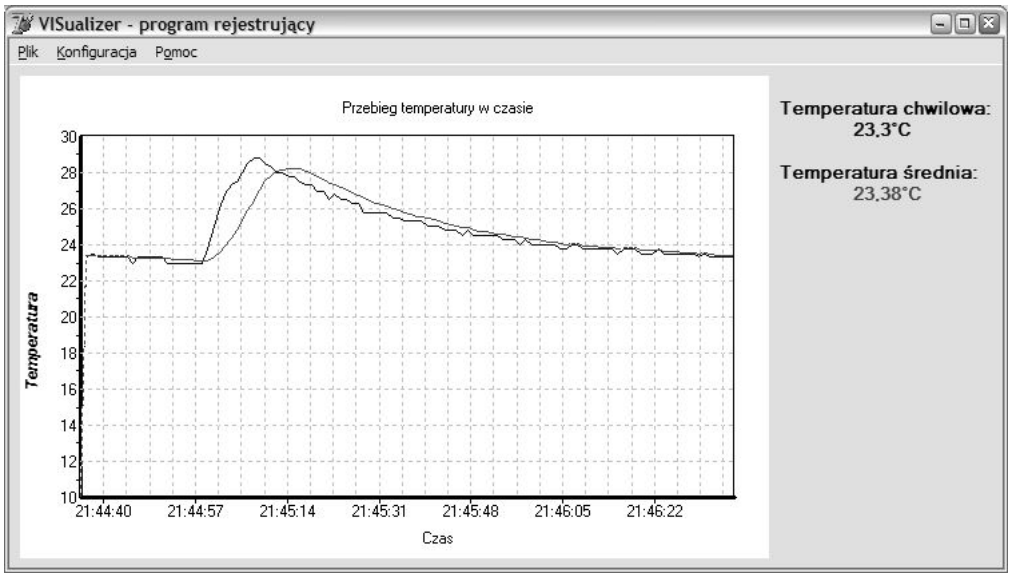
```
void init(void)
{
  PORTC = 0x00; // ustaw PORTC w stan niski
  DDRC = 0xFF; // PORTC cały jako wyjścia
  PORTD = 0x00; // podobnie PORTD
  DDRD = 0xFF;
  ADC_init(); // inicjalizacja przetwornika ADC
  USART_init(); // inicjalizacja interfejsu szeregowego
  LCD_init(); // inicjalizacja wyświetlacza LCD
  TIMER_init();
  set_sleep_mode(SLEEP_MODE_ADC);
  sei();
}

void ADC_init(void)
{
  ADCSRA = (1<<ADEN) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1);
  // włącz ADC, zezwól na przerwanie, włącz preskaler
  ADMUX = (1<<REFS1) | (1<<REFS0);
  // tryb Single-Ended, wewnętrzne napięcie odniesienia 2.56V,
  kanał pomiarowy ADC0
}

void LCD_init(void)
{
  _delay_ms(30);
  LCD_WriteCommand(0x38); // 8bit, 2 linie, czcionka 5x8
  _delay_ms(5);
  LCD_WriteCommand(0x38);
  _delay_ms(1);
  LCD_WriteCommand(0x38);
  LCD_WriteCommand(0x38);
  LCD_WriteCommand(0x06); // zwiększaj adres, bez przesuwania
  LCD_WriteCommand(0x0C); // włącz wyświetlanie, bez kursora
                             i migotania
  for (i=0; i<10; i++)
    _delay_ms(20);
  LCD_PDefineChar(0, (uint16_t)deg);
  // załaduj znak stopnia do pamięci wyświetlacza
  LCD_Clear(); // wyczyść wyświetlacz
}

void USART_init(void)
{
  UBRRH = (unsigned char)(BAUD_REG>>8);
  UBRRL = (unsigned char) BAUD_REG;
  UCSRB = (1<<TXEN); // włącz nadajnik
}
```

```
void TIMER_init(void)
{
  TIMSK |= (1<<TOIE1);
}
void USART_Transmit(unsigned char data)
{
  while (!(UCSRA & (1<<UDRE))) ;
  UDR = data;
}
void StartConversion(void)
{
  sleep_enable();
  sleep_cpu();
  sleep_disable();
}
double ADCValToCelsius(uint16_t value)
{
  return ((double)value*(256/1024.0));
}
void DisplayTemperature(char *s)
{
  LCD_Cursor(2,5);
  LCD_SendString(s);
  LCD_DisplayCharacter(0);
  LCD_DisplayCharacter('C');
}
void SendPacketByUSART(char *s)
{
  USART_Transmit('@');
  for (i=0; *(ptr+i)!=0; i++)
    USART_Transmit(*(ptr+i));
  USART_Transmit('#');
}
int main(void)
{
  init();
  LCD_Clear();
  LCD_PSendString((uint16_t)text);
  ptr = ttemp;
  state |= SLEEP_REQ;
  while (1){
  if (state & SLEEP_REQ)
    StartConversion();
  if (state & NEW_VAL){
    state &= !NEW_VAL;
```

```
    tempvalue = ADCValToCelsius(ADCValue);
    ptr = dtostrf(tempvalue, 5, 1, ptr);
    DisplayTemperature(ptr);
    SendPacketByUSART(ptr);
    TCCR1B |= (1<<CS12);
    }
  }
}
```

6. Wizualizacja pomiarów temperatury

Program, którego zadaniem jest wyświetlenie przebiegu temperatury w funkcji czasu w postaci wykresu, został napisany w środowisku Delphi.



Rysunek 7. Wizualizacja pomiaru temperatury

6.1. **Algorytm wizualizacji.** Kolejne etapy działania programu:

(1) inicjalizacja zmiennych i stworzenie interfejsu,
(2) zdarzenie odebrania pakietu danych z portu RS-232,
(3) konwersja odebranych danych tekstowych do formatu liczbowego i wyświetlenie uzyskanej wartości na wykresie,
(4) obliczenie wartości średniej z 10 ostatnich pomiarów i wyświetlenie na wykresie jako osobna seria danych,
(5) oczekiwanie na zdarzenie 2.

6.2. **Listing programu przedstawiającego wizualizację temperatury.**

```
unit uMain;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, JvExControls, JvChart, Menus, JvMenus, CPort, uConfig,
```

```
  XPMan, StatsClasses, DateUtils, ExtCtrls, JvLabel, JvStaticText,
  StrUtils, StdCtrls;

type
  TfrmMain = class(TForm)
    MainMenu1: TJvMainMenu;
    Plik1: TMenuItem;
    Konfiguracja1: TMenuItem;
    Pomoc1: TMenuItem;
    ComPort1: TComPort;
    ComDataPacket1: TComDataPacket;
    XPManifest1: TXPManifest;
    Chart: TJvChart;
    T_chwil: TJvStaticText;
    T_sred: TJvStaticText;
    JvLabel1: TJvLabel;
    JvLabel2: TJvLabel;
    procedure FormCreate(Sender: TObject);
    procedure Konfiguracja1Click(Sender: TObject);
    procedure ChartUpdate(NewVal: Single);
    procedure ComDataPacket1Packet(Sender: TObject; const Str: String);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    function ReplaceDP(const Str: String): String;
  private
    t_stat: TStatArray;
  public
  end;

var
  frmMain: TfrmMain;

implementation
uses Math;
{$R *.dfm}

function TfrmMain.ReplaceDP(const Str: String): String;
var s: String;
    i: Integer;
begin
  s := Trim(Str);
  for i:=0 to Length(s)-1 do
    if s[i] = '.' then begin
      s[i] := ',';
      break;
    end;
  Result := s;
end;
```

```
procedure TfrmMain.ChartUpdate(NewVal: Single);
begin
  Chart.Data.Value[0,Chart.Data.ValueCount] := NewVal;
  Chart.Data.Timestamp[Chart.Data.ValueCount-1] := Now;
  t_stat.AddValue(NewVal);
  Chart.Data.Value[1,Chart.Data.ValueCount-1] := t_stat.Average;
  Chart.PlotGraph;
  T_chwil.Caption := FloatToStr(RoundTo(NewVal,-2)) + 'C';
  T_sred.Caption := FloatToStr(RoundTo(t_stat.Average,-2)) + 'C';
end;

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  t_stat := TStatArray.Create(10);
  if Assigned(Chart) then begin
    Chart.Data.Clear;
    Chart.Options.PenColor[0] := clNavy;
    Chart.Data.Value[0,0] := 0;
    Chart.Data.Value[1,0] := 0;
    Chart.Data.Timestamp[0] := Now;
  end;
  T_chwil.Caption := '';
  T_sred.Caption := '';
  ComPort1.Connected := true;
end;

procedure TfrmMain.Konfiguracja1Click(Sender: TObject);
begin
  frmConfig.ShowModal;
end;

procedure TfrmMain.ComDataPacket1Packet(Sender: TObject;
const Str: String);
var temp: String;
begin
  temp := ReplaceDP(Str);
  ChartUpdate(StrToFloatDef(temp,0.0));
end;

procedure TfrmMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if ComPort1.Connected then
    ComPort1.Connected := false;
end;

end.
```

7. Planowane działania

W przyszłości planowane jest wykorzystanie w pełni możliwości mikrokontrolera, m.in. przesyłanie aktualnej temperatury na żądanie poprzez sieć GSM, czy informowanie użytkownika o bieżącej temperaturze online. Jednym z wariantów powyższych zamierzeń może być stworzenie całej stacji pogodowej z dostępem do aktualnych warunków atmosferycznych przez Internet lub GSM. Ponadto planuje się opracowanie zestawu ćwiczeń laboratoryjnych dla studentów, bazujących na wykorzystywanym w pracy module uruchomieniowym.

8. Podsumowanie

Celem artykułu było przedstawienie aspektów związanych z pomiarem i cyfrowym przetwarzaniem wielkości analogowych. Potrzeba przetwarzania analogowo-cyfrowego wynika stąd, że wartość wielkości mierzonych ma postać analogową oraz z bardzo korzystnych właściwości sygnałów cyfrowych: ich odporności na zakłócenia, łatwości i większej dokładności ich przetwarzania. Zaprojektowany układ pracuje bez zastrzeżeń, wykazuje się zadowalającą dokładnością i pozostawia jeszcze wiele możliwości odnośnie rozbudowy.

Literatura

[1] R. Baranowski, *Mikrokontrolery AVR ATmega w praktyce*, wyd. BTC, Warszawa 2005.
[2] ATMEL, *http://www.atmel.com/*
[3] GOTRONIK, *http://www.gotronik.pl/*
[4] National Semiconductors, *http://www.national.com/*

THE MEASUREMENT AND PROCESSING OF TEMPERATURE WITH USE OF THE AVR ATMEGA16 MICROCONTROLLER

BOGUMIŁ ZARZYCKI, MARIUSZ HOLUK

Abstract. The main purpose of this article is to describe the problems related to measurement of analog signals (according to temperature measurement) and its software implementation by using the AVR ATmega16 microcontroller. The proposed processing circuit can work autonomously. Further processing functions and time plot visualization are delivered by additional software installed on a PC.