Sprawozdanie z laboratorium

Architektury Komputerów

Ćwiczenie numer: 7

Temat: W i r e

Wykonujący ćwiczenie:

- Zaborowska Magda
- Wójtowicz Patryk

Studia dzienne I stopniaKierunek:

Informatyka

Semestr: III Grupa zajęciowa: Lab 15

Prowadzący ćwiczenie:

Dr inż. Mirosław Omieljanowicz

.....

OCENA

Data wykonania ćwiczenia

17.12.2021r.

.....

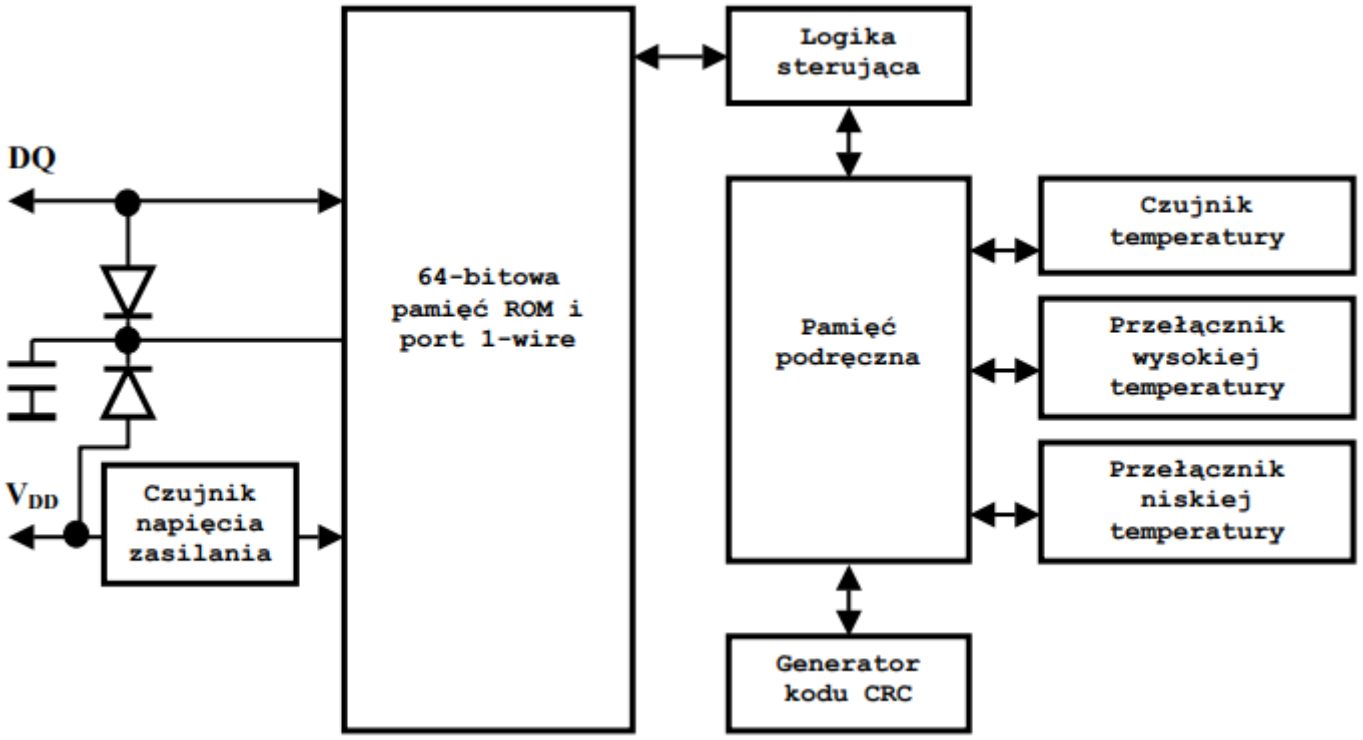
Data i podpis prowadzącego

Cel zadania

Zapoznanie się z działaniem magistrali 1-Wire i sposobem programowania przykładowego układu pomiaru temperatury, komunikując się przy tym z otoczeniem.

Teoria

Układ DS1820 umożliwia odczyt temperatury w postaci 9 bitowego słowa. Informacja przesyłana jest za pośrednictwem jedno przewodowego interfejsu. Zasilanie niezbędne jest dla operacji odczytu, zapisu i przetwarzania temperatury, aczkolwiek możliwe jest pobieranie jej z samej linii danych (zasilanie pasożytnicze). Strukturę typowego układu DS1820 prezentuje rysunek poniżej:



Ważnym jest, że każdy układ DS1820 posiada swój indywidualny 64-bitowy numer ifentyfikacyjny. Umożliwia on komunikacje z właściwym układem w przypadku kiedy do magistrali jest podpiętych więcej urządzeń. Aby móc połączyć wtedy urządzenia do magistrali 1-wire każda sekwencja komunikacji wymaga podania tego unikalnego adresu. Wyjątek stanowi obecność tylko jednego urządzenia, wtedy nie trzeba komunikować się przy użyciu unikalnego adresu jest to kierowane automatycznie na jedyne urządzenie.

Układ DS1820 zawiera pamięć podręczną typu RAM oraz pamięć stałą typu EEPROM. Pamięć podręczna zawiera następujące wpisy:

Pamięć podręczna	Bajt
Bajt temperatury (LSB)	0
Bajt temperatury (MSB)	1
TH lub bajt użytkownika 1	2
TL lub bajt użytkownika 2	3
Zarezerwowany	4
Zarezerwowany	5
Reszta zliczania (Count_Remain)	6
Liczba zliczeń na °C (Count_Per_C)	7
CRC	8

Pamięć EEPROM zawiera wartości przerzutników temperatur wysokeij (TH) oraz niksiej (TL):

EEPROM

TH lub bajt użytkownika 1
TL lub bajt użytkownika 2

Najważniejszymi poleceniami funkcji pamięci są

Polecenie i KOD	Opis
Zapisz do pamięci podręcznej [4Eh]	Polecenie to powoduje zapis dwóch bajtów do pamięci o adresach 2 i 3.
Odczytaj pamięć podręczną [BEh]	Polecenie to powoduje odczyt całej pamięci podręcznej od adresu 0 do 8.
Kopiuj pamięć podręczną [48h]	Kopiuje dwa bajty pamięci podręcznej o adresach 2 i 3 do pamięci EEPROM.
Zmierz temperaturę T [44h]	Polecenie rozpoczęcia pomiaru i przetwarzania temperatury. Po realizacji pomiaru układ przechodzi w stan bezczynności. Pomiar może trwać do 500ms.
Wywołaj (odczytaj) EEPROM [B8h]	Przepisuje zawartość bajtów TH i TL pamięci EEPROM do pamięci podręcznej.

Pamięć podręczna układu DS18B20 różni się od standardowej i przedstawiona została poniżej:

Pamięć podręczna	
Bajt temperatury (LSB)	0
Bajt temperatury (MSB)	1
TH lub bajt użytkownika 1	2
TL lub bajt użytkownika 2	3
Rejestr konfiguracyjny	4
Zarezerwowany	5
Zarezerwowany	6
Zarezerwowany	7
CRC	8

TEMPERATURA:

Zmierzona temperatura zapisywana jest w dwóch bajtach pamięci podręcznej układu pod adresami 0 oraz 1. Liczba ta stanowi 16-bitową wartość temperatury zapisaną w kodzie uzupełnień do dwóch (U2) z rozszerzonym do pełnego bajtu znakiem np.: 0000000011110000

Pierwsze 8 bitów stanowi znak:
„**00000000**" -temperatura dodatnia
„**11111111**" -temperatura ujemna

Pozostałe 8 bitów dają wielokrotność najmniej znaczącego bitu (LSB) dla którego temperatura wynosi 0.5°C. W naszym przykładzie to:
11110000 = 240 dziesiętnie = 240 * 0.5°C = 120°C

Format rejestrów przechowujących wartość temperatury:

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LSB	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
MSB	S	S	S	S	S	S	S	S

gdzie: S – znak temperatury. Jeżeli S=0 (S=1) to temperatura dodatnia (ujemna).

Większa rozdzielczość niż 0.5°C może być uzyskana z wykorzystaniem poniższej procedury obliczeniowej:

$$TEMP = TEMP\_Read - 0.25 + \frac{Count\_Per\_C - Count\_Remain}{Count\_Per\_C}$$

TEMP_Read – temperatura odczytana z pamięci podręcznej po odcięciu najmniej znaczącego bitu.
Count_Per_C – liczba zliczeń przypadająca na 1°C.

Count_Remain - wartość pozostała w liczniku (zawartość licznika w chwili upływu czasu otwarcia bramki).

Jeżeli wykorzystywany jest układ DS18B20 zapis temperatury w pamięci podręcznej jest inny niż opisany powyżej. Wynika to z faktu, że w układzie DS18B20 możliwa jest zmiana rozdzielczości pomiaru temperatury poprzez zmianę wartości bitów R0 i R1 rejestru konfiguracyjnego. Temperatura skalowana jest w stopniach Celsjusza. Jeżeli rozdzielczość wynosi:

- 11-bitów - bit 0 rejestru LSB jest nieokreślony,
- 10-bitów - bity 0,1 rejestru LSB są nieokreślone,
- 9-bitów - bity 0,1,2 rejestru LSB są nieokreślone.

Format rejestrów przechowujących wartość temperatury:

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LSB	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
MSB	S	S	S	S	S	2^6	2^5	2^4

gdzie: S – znak temperatury. Jeżeli S=0 (S=1) to temperatura dodatnia (ujemna).

Konwersja wartości temperatury zapisanej w bitach na °C dla 12-bitowej rozdzielczości pomiaru:

Temperatura [°C]	Wartość cyfrowa
+125	0000 0111 1101 0000
+85	0000 0101 0101 0000
+25,0625	0000 0001 1001 0001
+10,125	0000 0000 1010 0010
+0,5	0000 0000 0000 1000
0	0000 0000 0000 0000
-0,5	1111 1111 1111 1000
-10,125	1111 1111 0101 1110
-25,0625	1111 1110 0110 1111
-55	1111 1100 1001 0000

Rejestr konfiguracyjny, którego strukturę pokazano poniżej decyduje o rozdzielczości pomiaru temperatury przez układ. Należy zwrócić uwagę, że im wyższa rozdzielczość tym dłuższy czas konwersji temperatury.

Rejestr konfiguracyjny (DS18B20):

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	R1	R0	1	1	1	1	1

Znaczenie bitów R1 i R0 podano w tabeli 6. Zawiera ona również maksymalne czasy konwersji temperatury.

Konfiguracja parametrów pracy DS18B20:

R1	**R0**	**Rozdzielczość**	**Maksymalny czas konwersji**
0	0	9-bit	93.75 ms
0	1	10-bit	187.5 ms
1	0	11-bit	375 ms
1	1	12-bit	750

Właściwości magistrali 1-Wire:

- Wykorzystuje jedną linię do dwustronnej komunikacji,
- Wymaga rezystora podciągającego na linii danych (ang. *pull-up resistor*) najczęściej 4.7-5kW,
- System z magistralą 1-Wire wymaga jednego urządzenia typu *Master* oraz jednego lub więcej urządzeń typu *Slave*,
- Każde urządzenie podłączone jest do magistrali poprzez końcówkę typu „open drain” lub 3-state port,
- Wszystkie dane i rozkazy przesyłane są poprzez magistralą począwszy od najmłodszego bitu (ang. *LSB*).

Każda transakcja (wymiana danych) powinna obejmować trzy etapy:

1. **Inicjalizacja**. Zawiera etap zerowania układu *Slave* oraz etap potwierdzenia przez *Slave*'a obecności na magistrali.

2. **Przesłanie komendy typu ROM**. Każda układ *Slave* posiada unikalny 64-bitowy kod. Komendy typu ROM umożliwiają zaadresowanie konkretnego układu, identyfikację układu, wyszukanie alarmu lub pominięcie sprawdzania 64-bitowego kodu.

3. **Przesłanie funkcji sterującej układu**. Sekwencja tego typu umożliwia realizację wszystkich dostępnych operacji na wybranym układzie np. DS1820.

Ogólny schemat komunikacji z układami podłączonymi do magistrali 1-Wire na podstawie układu DS1820:

Lp	Komenda	Opis
1	RESET	Inicjalizacja przez urządzenie nadrzędne (*Master*) łączności z DS1820.
2	„Przeskocz ROM" [CCh]	Informacja dla DS1820, że można pominąć sprawdzanie i wysyłanie 64-bitowego unikalnego numeru identyfikacyjnego.
3	„Zmierz temperaturę T" [44h]	Inicjalizacja pomiaru temperatury przez DS1820.
4	Utrzymuj stan wysoki (500ms)	Przy zasilaniu układu z linii danych DQ wymagane do przeprowadzenia etapu pomiaru temperatury. Przy zasilaniu zewnętrznym (linia V_{DD}) czas opóźnienia wymagany do zakończenia cyklu pomiarowego.
5	RESET	Ponowna inicjalizacja przez urządzenie nadrzędne łączności z DS1820.
6	„Przeskocz ROM" [CCh]	Jak wyżej
7	„Odczytaj pamięć podręczną" [BEh]	Polecenie to powoduje odczyt całej pamięci podręcznej od adresu 0 do 8 (w tym zmierzonej temperatury).
8	Etap transmisji danych	Układ DS1820 wysyła kolejno 9 bajtów danych z pamięci podręcznej począwszy od adresu 0 i bitu LSB.

Realizacja zadania

```
1  #include <msp430x14x.h>
2  #include <stdlib.h>
3  #include "portyLcd.h"
4  #include "lcd.h"
5  #define B1 BIT4 &P4IN
6  #define DALLAS BIT7
7  #define WE 0
8  #define PORT_1Wire DALLAS
9  #define SET_1Wire    P1DIR |= DALLAS;P1OUT |= DALLAS
10 #define CLEAR_1Wire  P1DIR |= DALLAS;P1OUT &= ~DALLAS
11 unsigned int cntr;
12 int wait_flag = 0;
```

```
int bit_is_set(int a, int b)
{
    P1DIR &= ~DALLAS;
    __delay_cycles(80);
    return P1IN & DALLAS;
}
int bit_is_set2(int a, int b)
{
    P1DIR &= ~DALLAS;
    return P1IN & DALLAS;
}
unsigned char RESET_PULSE(void)
{
    unsigned char PRESENCE;
    CLEAR_1Wire;          //ustawiamy magistrale w poziom niski
    __delay_cycles(4000); // >480us
    SET_1Wire;            //ustawiamy magistrale w poziom wysoki
    __delay_cycles(240);  //oczekiwanie na ustawienie linii w stan niski przez DS 15-60us
    //sprawdzamy poziom linii (czy w stanie niskim)
    if (bit_is_clear(PORT_1Wire, WE))
        PRESENCE = 1;
    else
        PRESENCE = 0;

    //1-odebrano bit PRESENCE, 0- stan nieaktywnooci
    __delay_cycles(3840); //odczekanie rzez mastera 480us i spr. czy DALLAS podciagnal magistrale
    //sprawdzamy posiom linii (czy ustawiona)
    if (bit_is_set(PORT_1Wire, WE))
        PRESENCE = 1;
    else
        PRESENCE = 0;
    return PRESENCE; //zwracamy wartosc do funkcji
}
```

```
void send(char bit)
{
    CLEAR_1Wire; //ustawienie w stan niski magistralii
    __delay_cycles(40);
    if (bit == 1)
    {
        SET_1Wire; // zwolnienie magistralii - wyslanie jedynki
    }
    __delay_cycles(640); // przetrzymanie - wyslanie zera
    SET_1Wire;
}

unsigned char read(void)
{
    unsigned char PRESENCE = 0;
    CLEAR_1Wire;         //ustawienie w stan niski DQ
    __delay_cycles(16);  //odczekanie 2us
    SET_1Wire;           //zwolnienie magistrali
    __delay_cycles(120); //delay 15us
    if (bit_is_set2(PORT_1Wire, WE))
        PRESENCE = 1;
    else
        PRESENCE = 0; //odbior jedynki lub zera
    return (PRESENCE);
}
```

```
int celsiusToFarenheit(int t)
{
    int u = t % 100;
    float f = 32 + 9 * ((t - u) / 100.f + u / 100.f) / 5.f; // obliczanie temp
    float x = (f - (int)f) * 100;
    if (x - (int)x >= 0.5)
        f += 0.01f;
    return (int)(f * 100);
}

```

```
void send_byte(char wartosc)
{
    unsigned char i;   //zmienna licznikowa
    unsigned char pom; //zmienna pomocnicza

    for (i = 0; i < 8; i++)
    {
        pom = wartosc >> i; //przesuniecie bitowe w prawo
        pom &= 0x01;        //skopiowanie bitu do zmiennej pomocniczej
        send(pom);          //wyslanie bitu na magistrale
    }
    __delay_cycles(800); //odczekanie 100us
}

char read_byte(void)
{
    unsigned char i;           // zmienna licznikowa
    unsigned char wartosc = 0; //zczytywana wartosc

    for (i = 0; i < 8; i++) //petla wykonywana 8 razy
    {
        if (read())
            wartosc |= 0x01 << i;
        __delay_cycles(120); //odczekanie 15us
    }
    return (wartosc); //zwrot wartosci do funkcji
}
```

```
void show(long int t)
{
    int cyfra, waga = 10;
    Delayx100us(2);
    if (t < 0) // wypisanie -  przed wartosci1 ujemna
    {
        SEND_CHAR('-');
        t *= -1;
    }
    if (t == 0)
    {
    }
    if (t < 10)
        SEND_CHAR('0');

    if (t >= 10000) // wypisanie ?  jezeli temperatura przekroczy dopuszczalny poziom
    {
        SEND_CHAR('?');
        return;
    }
    while (waga <= t)
        waga *= 10;
    while ((waga /= 10) > 10)
    {
        cyfra = t / waga; // liczba kt ra zostanie wyowietlona jako temperatura
        SEND_CHAR((int)('0' + cyfra));
        t -= cyfra * waga;
    }
    SEND_CHAR('.');
    cyfra = t / waga; // pierwsza cyfra po przecinku
    SEND_CHAR((int)('0' + cyfra));
    t -= cyfra * waga; // druga cyfra po przecinku
    SEND_CHAR((int)('0' + t));
}
```

```c
int main(void)
{
    unsigned char sprawdz;
    char temp1 = 0, temp2 = 0;
    int temp;
    unsigned int k;
    WDTCTL = WDTPW + WDTHOLD;
    InitPortsLcd(); // inicjalizacja port�w LCD
    InitLCD();      // inicjalizacja LCD
    clearDisplay(); // czyszczenie wyowietlacza
    //Mierzenie temperatury wewn�trznej
    ADC12CTL0 = ADC12ON | REFON | SHT0_15; // wlaczenie rdzenia i generatora napiecia odniesienia oraz wyb�r napiecia odniesienia
    ADC12CTL1 = SHP | CSTARTADD_0;         // pr�bkowanie impulsowe, wynik skladany w ADC12MEM0
    ADC12MCTL0 = INCH_10 | SREF_1;         // kanal 10, zr�dlo napiecia odniesienia - wewnetrzny generator 1,5 V
    for (k = 0; k < 0x3600; k++);          // czas na ustabilizowanie generatora napiecia odniesienia
    clearDisplay();                 // czyszczenie wyowietlacza
    clearDisplay();                 // czyszczenie wyowietlacza
    CCR0 = 50000;                   // ustala nowy okres licznika
    TACTL = TASSEL_2 | ID_3 | MC_1; // zr�dlo taktowania SMCLK dzielone przez 8, tryb UP
    CCTL0 = CCIE;                   // uaktywnienie przerwania od TACCR0
    _BIS_SR(GIE);                   // wlaczenie przerwan
    ADC12CTL0 |= ENC;               // uaktywnienie konwersji
    int i;

    P2DIR |= BIT1;

    BCSCTL1 |= XTS; // ACLK = LFXT1 = HF XTAL 8MHz
    do
    {
        IFG1 &= ~OFIFG; // Czyszczenie flgi OSCFault
        for (i = 0xFF; i > 0; i--)
            ;                           // odczekanie
    } while ((IFG1 & OFIFG) == OFIFG); // dop�ki OSCFault jest ci1gle ustawiona

    BCSCTL1 |= DIVA_0;        // ACLK=8 MHz
    BCSCTL2 |= SELM0 | SELM1; // MCLK= LFTX1 =ACLK
     for (;;)
     {

         if (0 == 0)
         {
             wcisniety = 1;
             zerowy = 0;

             sprawdz = RESET_PULSE(); //impuls resetu
             if (sprawdz == 1)
             {

                 send_byte(0xCC); //SKIP ROM
                 send_byte(0x44); //CONVERT T
                 for (k = 0; k < 9; k++)
                     __delay_cycles(600000); //odczekaj 750ms - czas konwersji
                 sprawdz = RESET_PULSE();    //wyslanie impulsu reset
                 send_byte(0xCC);            //SKIP ROM
                 send_byte(0xBE);            //READ SCRATCHPAD

                 temp1 = read_byte(); //odczytanie LSB
                 temp2 = read_byte(); //odczytanie MSB

                 sprawdz = RESET_PULSE(); //zwolnienie magistrali

                 float temp = 0; //zmienna do obliczen

                 temp = (float)(temp1 + (temp2 * 256)) / 16; //obl. temp

                 SEND_CMD(DD_RAM_ADDR + 0x00);

                 if (temp < 0) // wypisanie -
                 {
                     SEND_CHAR('-');
                     temp *= -1;
                 }
```

```
            if (((int)(temp / 10) != 0) && (((int)(temp) % 10) != 0) && (((int)(temp * 10) % 10) != 0))
            {
                clearDisplay();
                SEND_CHAR((int)(temp / 10) + 0x30);
                SEND_CHAR((int)(temp) % 10 + 0x30);
                SEND_CHAR('.');
                SEND_CHAR((int)(temp * 10) % 10 + 0x30);
                SEND_CHAR('C');
                __delay_cycles(16000);
            }

            else //jezli nie wykryto PRESENCE_PULSE
            {
                if (((int)(temp / 10) == 0) && (((int)(temp) % 10) == 0) && (((int)(temp * 10) % 10) == 0))
                {
                    SEND_CMD(DD_RAM_ADDR + 0x00);

                    SEND_CHAR(' ');
                    SEND_CHAR('B');
                    SEND_CHAR('R');
                    SEND_CHAR('A');
                    SEND_CHAR('K');
                    SEND_CHAR(' ');
                    SEND_CHAR('C');
                    SEND_CHAR('Z');
                    SEND_CHAR('U');
                    SEND_CHAR('J');
                    SEND_CHAR('N');
                    SEND_CHAR('I');
                    SEND_CHAR('K');
                    SEND_CHAR('A');
                    SEND_CHAR(' ');
                }
            }
        }
```

```
            else //jezli nie wykryto PRESENCE_PULSE
            {
                SEND_CMD(DD_RAM_ADDR + 0x00);

                SEND_CHAR(' ');
                SEND_CHAR('Z');
                SEND_CHAR('L');
                SEND_CHAR('E');
                SEND_CHAR(' ');
                SEND_CHAR('W');
                SEND_CHAR('P');
                SEND_CHAR('I');
                SEND_CHAR('E');
                SEND_CHAR('T');
                SEND_CHAR('Y');
                SEND_CHAR(' ');
                SEND_CHAR(' ');
                SEND_CHAR(' ');
                SEND_CHAR(' ');
                SEND_CHAR(' ');
            }
        }
        else
            wcisniety = 0;

        ADC12CTL0 |= ADC12SC; // start konwersji
        while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY)
            ; // czekanie na koniec konwersji
        temp = (int)(ADC12MEM0 * 1.0318 - 2777.4647) * 10;
```

```
         if (wait_flag == 0 && zerowy == 0)
         {
             SEND_CHAR(' ');
             SEND_CHAR(' ');
             SEND_CHAR('T');
             SEND_CHAR('E');
             SEND_CHAR('M');
             SEND_CHAR('P');
             SEND_CHAR('.');
             SEND_CHAR('O');
             SEND_CHAR('U');
             SEND_CHAR('T');
             wait_flag = 1;
         }
```

```
306         else if (zerowy == 1)
307         {
308             SEND_CHAR(' ');
309             SEND_CHAR('B');
310             SEND_CHAR('R');
311             SEND_CHAR('A');
312             SEND_CHAR('K');
313             SEND_CHAR(' ');
314             SEND_CHAR(' ');
315             SEND_CHAR('T');
316             SEND_CHAR('E');
317             SEND_CHAR('M');
318             SEND_CHAR('P');
319             SEND_CHAR('.');
320             SEND_CHAR('O');
321             SEND_CHAR('U');
322             SEND_CHAR('T');
323             SEND_CHAR(' ');
324         }
325         if (!wcisniety || zerowy == 1)
326         {
327             SEND_CMD(DD_RAM_ADDR2);
328             show(temp); // wyswietla na wyswietlaczu
329             SEND_CHAR('C');
330             SEND_CHAR(' ');
331             SEND_CHAR('T');
332             SEND_CHAR('E');
333             SEND_CHAR('M');
334             SEND_CHAR('P');
335             SEND_CHAR('.');
336             SEND_CHAR('I');
337             SEND_CHAR('N');
338             SEND_CHAR('P');
339             SEND_CHAR(' ');
340             SEND_CHAR(' ');
341         }
342          for (int i = 0; i < 10000; i++)
343          {
344          }
345     }
346 }
347
348 #pragma vector = TIMERA0_VECTOR
349 __interrupt void Timer_A()
350 {
351     if (++cntr == 10)
352     {
353         _BIC_SR_IRQ(LPM0_bits);
354         cntr = 0;
355     }
356 }
```