Wielokanałowy termometr

Wydawałoby się, że nie ma bardziej banalnego projektu niż termometr elektroniczny. Sam kilka zaprojektowałem i wykonałem. Jednak do skonstruowania kolejnego termometru skłoniła mnie potrzeba – potrzebowałem termometru mającego możliwość dołączenia różnej liczby czujników. Dodatkowo była wymagana funkcja termostatu z ustawianą histerezą i progiem zadziałania, elastycznie włączana dla jednego z dostępnych kanałów pomiarowych.

Zgodnie z założeniami, termometr miał być wielokanałowy, więc trzeba było zapewnić możliwość dołączenia kilku czujników do mikrokontrolera. Liczbę używanych czujników można zaprogramować, ale lepiej, gdyby termometr sam potrafił automatycznie wykryć ile kanałów pomiarowych jest aktywnych. Na rynku jest dostępny olbrzymi wybór elektronicznych czujników temperatury z interfejsami cyfrowymi. Są one łatwe w aplikacji i co ważne – kalibrowane w procesie produkcyjnym. Ponadto, są wyposażone w cyfrowy interfejs przeznaczony do przesyłania zmierzonej temperatury w „gotowej" postaci do mikrokontrolera.

Wiele z dostępnych czujników spełniało założenia projektu, ale jeden z nich dało się zastosować szczególnie łatwo – znany i popularny czujnik DS18B20. Interfejs 1-Wire, w który jest wyposażony ten układ, pozwala na dołączenie wielu czujników do jednej linii portu mikrokontrolera. Każdy z nich ma unikalny 64-bitowy numer seryjny zapisany w trakcie produkcji. Można napisać procedurę, która w czasie inicjowania systemu automatycznie wykryje i zidentyfikuje wszystkie dołączone czujniki. W trakcie eksploatacji termometru jest możliwe dołączanie i odłączanie czujników. Wystarczy po zmianie ich liczby wyłączyć i włączyć termometr, aby oprogramowanie jest wykryło i aktywowało.

DS18B20 jest umieszczany miedzy innymi w popularnej, „tranzystorowej" obudowie TO-92. Termometr ma tylko 3 wyprowadzenia: masę, plus zasilania i linia DQ – dwukierunkowa magistrala 1-Wire. W wersji uproszonej układ można zasilić z linii DQ – wtedy potrzebne są tylko 2 przewody. Niestety, układ ma też swoje wady. Uproszczona magistrala wymaga skomplikowanej obsługi z koniecznością precyzyjnego odmierzania czasów rzędu kilku mikrosekund. Zazwyczaj obsługa 1-Wire blokuje mikrokontroler i nie jest możliwe wykonywanie w trakcie odczytywania pomiarów realizowanie innych czynności. Dla szybkich mikrokontrolerów 16- lub 32-bitowych można napisać program obsługi magistrali będący automatem stanu wykonywanym w obsłudze przerwań, ale jest on bardziej skomplikowany i trudny do uruchomienia. W termometrach wykonujących tylko pomiar temperatury prędkość wykonywanie pomiarów nie jest problemem i można stosować typowe procedury blokujące czas procesora.

Sterownik i wyświetlacz

Schemat termometru pokazano na **rysunku 1**. Po wyborze czujnika temperatury następnym krokiem był wybór mikrokontrolera. Pierwsze próby i wstępne testy oprogramowania wykonałem na płytce *Nucleo* w środowisku *Mbed*. Taki zestaw jest rewelacyjny do testów, ale w tym wypadku trudno go użyć w wymaganej aplikacji. Do budowy prototypu termometru wybrałem 16-bitowy mikrokontroler PIC24FJ64GA002 w obudowie do montażu przewlekanego. 16-bitowa jednostka jest dosyć szybka i ma wystarczającą pamięć programu do realizacji urządzenia. Wbrew pozorom, jak potem zobaczymy, program sterujący jest dość rozbudowany, a zastosowanie co prawda małego, ale jednak graficznego wyświetlacza z trzema różnymi wielkościami czcionek wymaga sporej pamięci i określonej szybkości działania. Szybkość ta jest niezbędna do płynnego wyświetlania informacji na ekranie.

Rdzeń mikrokontrolera jest taktowany częstotliwością 32 MHz za pomocą wewnętrznego oscylatora RC o częstotliwości 8 MHz powielonej przez 4 w układzie PLL i zasilany napięciem +2,5 V. Aby można było zasilać mikrokontroler napięciem +3,3 V w strukturę układu wbudowano odpowiedni regulator. Jest on załączany poziomem na wejściu DISVREG. Wyzerowanie wejścia DISVREG powoduje włączenie stabilizatora. Do wyprowadzenia VCAP/VCORE trzeba dołączyć kondensator o pojemności 10 µF i małej rezystancji ESR. Jeżeli wyprowadzenie DSVREG będzie ustawione, to do wejścia VCAP/VCORE trzeba dołączyć źródło napięcia +2,5 V zasilające rdzeń mikrokontrolera. Ja zasiliłem mikrokontroler napięciem +3,3 V i wykorzystałem wbudowany stabilizator.

Pamięć mikrokontrolera można programować w układzie wykorzystując do tego celu złącze ICSP z wyprowadzeniami zgodnymi z programatorem/debugerem PICKit3. Ponieważ mikrokontroler nie ma wbudowanej pamięci nieulotnej przeznaczonej dla użytkownika, to do zapisywania nastaw termometru wykorzystałem pamięć EEPROM typu 24C04 z interfejsem I²C. Komunikację mikrokontrolera pracującego w trybie master z pamięcią zapewnia sprzętowy interfejs I2C2 (linie SCL2 i SDA2). Rezystory R6 i R7 realizują wymagane przez standard I²C podciąganie do plusa zasilania linii danych i zegarowej.

W interfejsie użytkownika przeznaczonym do odczytu temperatury i ustawiania termostatu zastosowałem mały, graficzny wyświetlacz OLED i impulsator ze stykiem zwieranym po wciśnięciu ośki. Wyświetlacz jest zasilany napięciem +3,3V i komunikuje się z mikrokontrolerem za pomocą interfejsu SPI z dodatkową linią C/D. Impulsator ma typowe 2 styki: „A" (dołączone do port RB15) i „B" (do portu RB14). Linie A i B są podciągane

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 75421, pass: tkuyg3b9

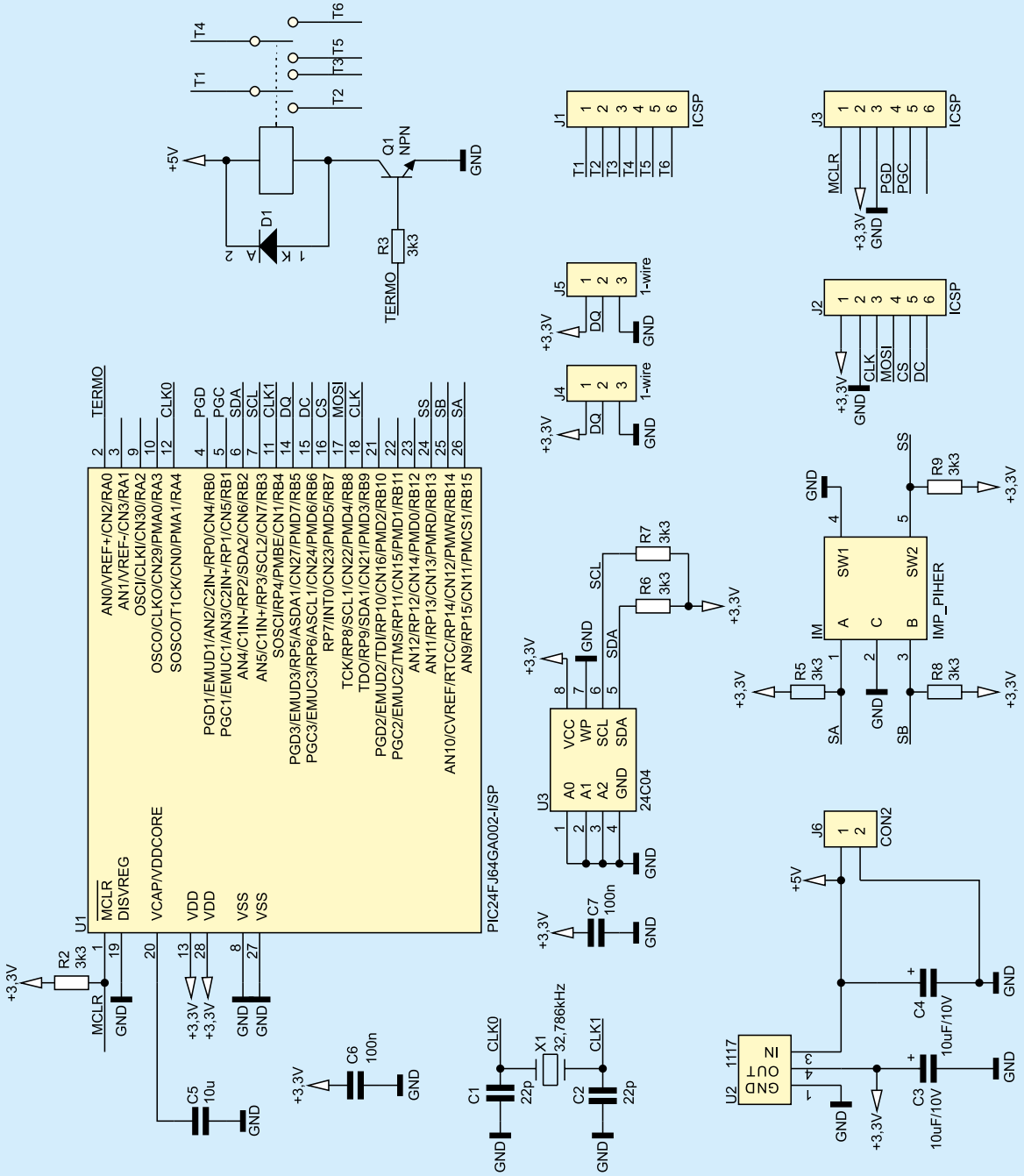
do plusa zasilania za pomocą rezystorów R5 i R8. Styk zwierny impulsatora jest połączony z linią RB13. Jeśli styk nie jest zwarty, to na RB13 jest poziom wysoki wymuszony przez rezystor R9. Po zwarciu styku (wciśnięciu ośki) na RB13 jest wymuszany poziom niski.

Układem wykonawczym termostatu jest przekaźnik typu AZ822 z cewką na napięcie 5 VDC. Po ustawieniu linii RA0 tranzystor Q1 przechodzi w stan przewodzenia powodując zadziałanie przekaźnik R1. AZ822 ma 2 pary styków przełączanych wyprowadzonych na listwę J1. Zależnie od potrzeb można wykorzystać styki zwarte lub rozwarte.

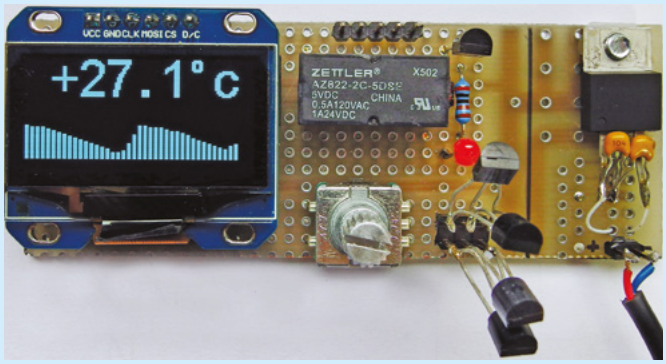
Cały układ jest zasilany ze stabilizatora LM1117-3.3. Na płytce nie przewidziano mostka prostowniczego i kondensatorów filtrujących. Całość powinna być zasilana z zasilacza o napięciu +5 V. Można do tego celu wykorzystać niepotrzebną ładowarkę do telefonu komórkowego. Starsze ładowarki maja napięcie wyjściowe ok. 7…8 V, nowsze zazwyczaj 5 V. Ja wykorzystałem ładowarkę od starego telefonu Alcatel, w której zamontowałem stabilizator 78L05. Oczywiście, można zasilić

Wykaz elementów

R2…R7 3,3 kΩ (1206)
C1, C2: 22 pF (0805)
C3…C5: 10 µF/10 V (3216)
C6, C7: 100 nF (1206)
T1: BC237
D1: BAV21
U1: PIC24FJ64GA002-I/SP
U2 LM1117-3.3 (TO-252)
U3: 24C04
Listwa goldpin
R1: przekaźnik A822-5VDC
Impulsator Piher



Rysunek 1. Schemat ideowy termometru



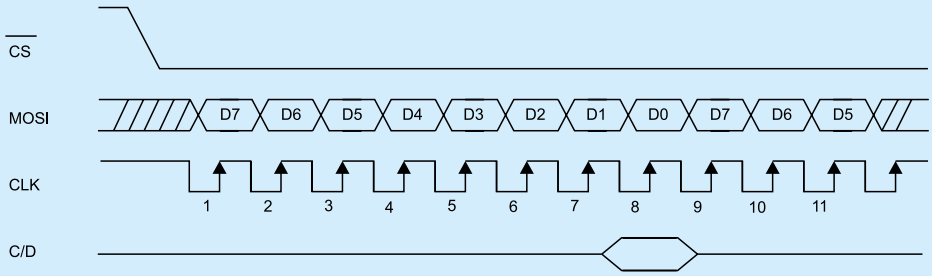
Fotografia 2. Moduł wyświetlacza

płytkę wyższym napięciem, ale wtedy trzeba zmienić przekaźnik R1.

Wyświetlacz OLED

Prezentacja wyników kilku pomiarów temperatury wymaga zastosowania wyświetlacza. Początkowo rozważałem użycie standardowego wyświetlacza alfanumerycznego ze sterownikiem HD44780. Potrzebowałem takiego o rozdzielczości przynajmniej 4×16 linii, jednak był on za duży. Zastosowałem mały, stosunkowo tani wyświetlacz graficzny o przekątnej 1,3", rozdzielczości 128×64 piksele, wykonany w technologii OLED. Mimo niewielkich rozmiarów okazało się, że matryca OLED zapewnia bardzo dobrą czytelność nawet z większej odległości. Wyświetlacz składa się z matrycy OLED i sterownika SH1106 umieszczonych na płytce szklanej z wyprowadzeniami elektrycznymi w formie taśmy elastycznej. Całość przymocowano na stałe do płytki drukowanej, na której umieszczono wszystkie niezbędne dodatkowe elementy: pola lutownicze do przylutowania taśmy wyświetlacza, stabilizator LDO napięcia +3,3 V, oraz goldpiny do podłączenia zasilania i sygnałów magistrali SPI przeznaczonej do podłączenia mikrokontrolera – hosta (**fotografia 2**).

Układ SH1106 jest specjalizowanym sterownikiem matryc OLED/PLED o maksymalnej rozdzielczości 132×64 piksele. Wbudowany driver steruje 132 segmentami i 64 kolumnami matrycy typu *Common Cathode*.



Rysunek 3. Przebiegi czasowe na magistrali SPI wyświetlacza.

Listing 1. Programowa obsługa SPI

```
//definicje linii interfejsu
define CS           LATBbits.LATB7
#define CS_TRIS     TRISBbits.TRISB7
#define MOSI        LATBbits.LATB8
#define MOSI_TRIS   TRISBbits.TRISB8
#define CLK         LATBbits.LATB9
#define CLK_TRIS    TRISBbits.TRISB9
#define DC          LATBbits.LATB6
#define DC_TRIS     TRISBbits.TRISB6

//procedura wyslania 8 bitowej danej przez SPI
void PutSpi(unsigned char data)
{
unsigned char i=0;
    CS=0;Nop();Nop();
    CLK=0;Nop();Nop();
    for(i=0;i<8;i++)
    {
        if((data&0x80)==0x80) MOSI=1;
        else MOSI=0;
        Nop();Nop();
        CLK=1;Nop();Nop();
        CLK=0;Nop();Nop();
        data<<=1;
    }
    CLK=0;Nop();Nop();
    CS=1;Nop();Nop();
}
```

Listing 2. Zapis danej do rejestru sterującego

```
void OledData(unsigned char  data)
{
    DC=1;
    CS=0;
    PutSpi(data);
}
```

Listing 2. Zapis danej do pamięci obrazu

```
void OledCmd(unsigned char  cmd)
{
    DC=0;
    CS=0;
    PutSpi(cmd);
}
```

Sterownik ma tryb oszczędzania energii *Sleep mode* z poborem prądu nieprzekraczającym 5 µA i może pracować w szerokim zakresie temperatury od -40 do +85°C.

Do komunikacji z mikrokontrolerem można wybrać równoległą, 8-bitowa magistralę mogącą pracować w trybach przemysłowych zgodnych z Intel8080 lub Motorola 6800, lub dwa interfejsy szeregowe: SPI i I²C. Wyboru aktywnego interfejsu dokonuje się przez wymuszenie poziomów logicznych na wejściach konfiguracyjnych IM0…IM2. W naszym wypadku na stałe wybrano interfejs SPI i nie ma innej możliwości. Magistrala sterująca składa się z linii:

- Danych wyjściowych MOSI (z punktu widzenia mikrokontrolera).
- Zegara taktującego transmisją CLK.
- Wyboru układu na magistrali CS.

Dodatkowo, interfejs uzupełniono o linie sterującą C/D. Przebiegi czasowe podczas przesyłania danej 8-bitowej pokazano na **rysunku 3**. Dane są przesyłane od najstarszego bitu i wpisywane do wyświetlacza narastającym zboczem sygnału zegarowego CLK. Linia !CS musi być wtedy wyzerowana. Stan linii C/D określa czy dane trafią do pamięci obrazu (C/D=1), czy do rejestrów sterujących (C/D=0).

Obsługę magistrali SPI zrealizowano programowo – odpowiednią procedurę pokazano na **listingu 1**. Dwie kolejne procedury to procedura zapisu danej do pamięci obrazu wyświetlacza *OledData* (**listing 2**) i zapisu danej do rejestru sterującego SH1106 *OledCmd* (**listing 3**).

Sterownik wyświetlacza graficznego wymaga zainicjowania. Konieczność inicjalizacji wynika na przykład

z możliwości ustalania współrzędnej początkowej (0,0) zależnie od mocowania mechanicznego panelu, różnych źródeł zasilania driverów matrycy (wewnętrzna przetwornica lub napięcie zewnętrzne), poziomu kontrastu, częstotliwości odświeżania itp. Inicjalizacja polega na wysłaniu szeregu komend ustalających początkowe parametry pracy. W **tabeli 1** wymieniono ważniejsze komendy sterownika SH1106.

Dysponując wykazem komend musimy tak zainicjować sterownik, aby pracował poprawnie z panelem wyświetlacza. Ponieważ część komend wymaga dodatkowego argumentu, to wszystkie bajty komend inicjalizacji łącznie z niezbędnymi argumentami umieszczono w buforze *Buffer_Init*. Zawartość bufora z pierwszym bajtem kontrolnym 0x80 jest wysyłana do sterownika wyświetlacza przez funkcję *I2C_Master_BufferWrite*. Zawartość bufora została pokazano na **listingu 3**, a procedurę inicjalizacji na **listingu 4**.

Inicjalizacje można podzielić na kilka etapów:

- Wyzerowanie liczników: kolumn, stron pamięci i linii początkowej. Domyślnie, po włączeniu zasilania (POR) te liczniki są wyzerowane.
- Ustawienie powiązania zawartości pamięci RAM obrazu z pozycją na panelu OLED (orientacja wyświetlania).
- Ustawienie kontrastu, przetwornicy DC/DC zasilającej drivery i częstotliwości taktowania oraz włączenie wyświetlenia.

Matryca wyświetlacza jest monochromatyczna i jednemu pikselowi odpowiada pojedynczy bit w pamięci

Listing 4. Inicjalizacja wyświetlacza

```
uint8_t Buffer_Init[255]={0x80,    //control
byte
0xae,  //Display OFF
0x00,  //Low Column
0x10,  //High Column
0xB0,  //Page
0x40,  //Start line

0xA1,  //remap
0xDA,  //com pins
0x12,
0xD3,  //display offset
0x00,  //NO offset
0xc0,  //scan direction
0xc8,
0xA6,  //normal display
0xA4,  //display ON

0x81,  //set contrast
0x50,  //contrast DATA

0xa8,  //multiplex ratio
0x3f,  //1/64 duty

0xD5,  //Display clock divide
0x80,

0xd9,  //precharge period
0xF1,
0xDB,  //VCOM deselect
0x40,
0x8d,  //charge pump
0x14,

0xAF,  //display ON
};
```

Listing 3. Bufor Buffer_Init z komendami inicjalizacji

```
void InitOled(void){
    unsigned char i;
    CS_TRIS=0;MOSI_TRIS=0;CLK_TRIS=0;DC_
TRIS=0;
    DC=0;
    for(i=0;i<29;i++){
    CS=0;
    PutSpi(Buffer_Init[i]);
    }
}
```

Tabela 1. Ważniejsze komendy sterownika SH1106

Komenda	B7	B6	B5	B4	B3	B2	B1	B0	Funkcja
Adres kolumny 4 młodsze bity	0	0	0	0	Młodsza część licznika kolumn				Ustawienie 4 młodszych bitów licznika kolumn
Adres kolumny 4 starsze bity	0	0	0	1	Starsza część licznika kolumn				Ustawienie 4 starszych bitów licznika kolumn
Napięcie wyjściowe wbudowanej przetwornicy DC/DC	0	0	1	1	0	0	napięcie		Ustawienie napięcia wyjściowego wbudowanej przetwornicy DC/DC
Linia początkowa wyświetlacza	0	1		Licznik linii początkowej wyświetlacza					Przypisanie pozycji w pamięci obrazu do COM0
Ustawienie kontrastu	1	0	0	0	0	0	0	1	Kod komendy
	Poziom kontrastu 0…255								Wartość kontrastu
Remapowanie segmentów	1	0	1	0	0	0	0	ADC	ADC=0
Włączenie całego wyświetlacza	1	0	1	0	0	1	0	D	D=0 wyświetlacz wyłączony D=1 wyświetlacz włączony
Wyświetlanie w negatywie	1	0	1	0	0	1	1	D	D=0 wyświetlanie normalne D=1 negatyw
Multiplex ration	1	0	1	0	1	0	0	0	Ilość aktywnych kolumn skanowanych w jednej ramce odświeżania
	*	*	Ilość kolumn 1…64 (domyślnie 64)						
Sterownie wewnętrzną przetwornicą DC/DC	1	0	1	0	1	1	0	1	D=0 DC/DC wyłączona D=1 DC/DC wyłączona
	1	0	0	0	1	0	1	D	
Włączenie/wyłączenie wyświetlacza	1	0	1	0	1	1	1	D	D=0 wyświetlacz wyłączony D=1 wyświetlacz włączony
Ustawienie licznika stron	1	0	1	1	Licznik stron				Ustawienie licznika stron
Kierunek skanowania kolumn	1	1	0	0	D	*	*	*	D=0 COM0->COM63 D=1 COM63->COM0
Offset wyświetlania	1	1	0	1	0	0	1	1	Ustawia pierwszą linię przypisaną początkowi pamięci obrazu
	*	*	COMx						
Częstotliwość wewnętrznego zegara	1	1	0	1	0	1	0	1	B3..B0 – dzielnik czestotliwości 1…16 B7…B4 regulacja częstotliwości -25%..._50%
	Regulacja częstotliwości				Dzielnik				
Sprzętowa konfiguracja wyprowadzeń	1	1	0	1	1	0	1	0	Ustawia 2 sekwencje sterowania wyświetlaczem
	0	0	0	D	0	0	1	0	
Ustawienie napięcia VCOM	1	1	0	1	1	0	1	1	Ustawienie napięcia VCOM
	VCOM=(B[7:0]Xv ref)								

Listing 5 Ustawienie licznika kolumn

```
#define COLADDLOW 0
#define COLADDHIGH 0x10
#define PAGEADD 0xB0
#define LINEADD 0x40
#define OLEDON 0xAE
unsigned char  SetColumn(int column)
{
    if(column>128) return(1);
    ++column;
    ++column;
    OledCmd(COLADDLOW|(column&0x0f));
    OledCmd(COLADDHIGH|(column>>4));
    return(0);
}
```

Listing 6. Ustawienie licznika stron

```
unsigned char SetPage(unsigned char page)
{
    if(page>7) return(1);
    OledCmd(PAGEADD|page);
    return(0);
}
```

Listing 7. Zapis danej do komórki pamięci EEPROM

```
void EEWr(unsigned char addr, unsigned char data)
{
    UINT config1;
    CloseI2C2(); //zamknij interfejs I2C
    ConfigIntI2C2(MI2C_INT_OFF);  //zablokuj przerwania
od I2C2
    config1 = (I2C_ON  | I2C_7BIT_ADD );//konfiguruj
interfejs I2C2
    OpenI2C2(config1,39);   //
    IdleI2C2();
    StartI2C2();                    //sekwencja START
    while(I2C2CONbits.SEN );  //czekaj na zakończenie
sekwencji START
    MI2C2_Clear_Intr_Status_Bit; //Zeruj flagi przerwania
    IdleI2C2();
    MasterWriteI2C2(0xA0);        //Zapisz adres SLAVE
    while(I2C2STATbits.TBF);      //Czekaj na wysłanie 8
bitów
    while(!IFS3bits.MI2C2IF);     //Czekaj na 9-ty takt
zegara
    MI2C2_Clear_Intr_Status_Bit; //Zeruj flage przerwania
od I2C2
    while(I2C2STATbits.ACKSTAT); //czekaj ba bit
potwierdzenia
    IdleI2C2();
    MasterWriteI2C2(addr);  //wyślij adres komórki eeprom
    while(I2C2STATbits.TBF);
    while(!IFS3bits.MI2C2IF);
    MI2C2_Clear_Intr_Status_Bit;
    while(I2C2STATbits.ACKSTAT);
    IdleI2C2();
    MasterWriteI2C2(data);  //wyślij daną do zapisu
    while(I2C2STATbits.TBF);
    while(!IFS3bits.MI2C2IF);
    MI2C2_Clear_Intr_Status_Bit;
    while(I2C2STATbits.ACKSTAT);
    IdleI2C2();
    StopI2C2();  //sekwencja STOP
    while(I2C2CONbits.PEN);  //czekaj na zakończenie
sekwencji STOP
    IdleI2C2();
    __delay_ms(6);  // czekaj na zakończenie zapisu
do komórki eeprom
    CloseI2C2();  //zamknij interfejs I2C2
}
```

obrazu. Pamięć ma wielkość 132×64 bity, ale w rzeczywistości jest zorganizowana bajtowo. Host wysyła kolejne bajty po magistrali SPI pod lokacje określone przez liczniki kolumn i stron. Jeden bajt w pamięci obrazu odpowiada pionowej linijce o długości 8 pikseli. Najmłodszy bit tego bajta jest pikselem położonym najwyżej w linijce, a bit najstarszy pikselem położonym najniżej. Położenie linijki w poziomie określa licznik kolumn zmieniający się w zakresie 0…131, a położenie w pionie określa licznik stron zmienia się w zakresie 0…7. Po ustaleniu numeru strony kolejne zapisywane bajty tworzą pasek o szerokości 8 pikseli. Każdy zapis danej powoduje inkrementację licznika kolumn i nowa dana jest zapisywana po kolejna lokację. Kiedy licznik kolumn osiągnie wartość 131, to po następnym zapisie danych jest zerowany. Przepełnienie licznika kolumn nie powoduje inkrementacji licznika stron i jeżeli ni zmodyfikujemy go wykorzystując do tego celu komendę „ustawienie licznika stron", to kolejne wpisy do pamięci będą nadpisywać dane począwszy od pozycji zerowej (wyzerowanie licznika kolumn). To ważna właściwość, o której trzeba pamiętać. Na **listingach 5** i **6** zostały pokazane procedury ustawienia licznika kolumn i licznika stron.

Pamięć EEPROM

Pamięć EEPROM typu 24C04 ma wbudowany interfejs I²C. Do komunikacji z pamięcią użyto sprzętowego interfejsu I2C2 wbudowanego w mikrokontroler. Dostęp jest realizowany przez 2 sekwencje – zapisu i odczytu pojedynczej komórki pamięci.

Zapisanie komórki pamięci zaczyna się sekwencją *Start*. Po niej jest wysyłany adres slave z bitem R/W=0. Kolejne 2 bajty to adres komórki i bajt do zapisania w pamięci. Jako ostatnia jest wysyłana sekwencja *Stop*. Całość pokazano na **rysunku 4**. Na **listingu 7** pokazano procedurę zapisywania danej do komórki pamięci pod konkretny adres. Dla linii adresowych A0=A1=A2=0 adres slave jest równy 0xA0.

Odczytywanie danej z konkretnej lokalizacji rozpoczyna się od wysłania sekwencji *Start*, adresu slave z bitem RW=0, a po nim bajtu z adresem w pamięci EEPROM. Po tym na magistralę jest ponownie wysyłana sekwencja *Start*, a po niej adres slave z bitem RW=1 informującym pamięć, że ma odesłać dane. Po wysłaniu tego adresu mikrokontroler wywołuje sekwencję odczytania danej z pamięci i kończy wszystko wysłaniem sekwencji *Stop*.

Czujnik temperatury DS18B20

Czujnik temperatury DS18B20 był wielokrotnie opisywany na łanach Elektroniki Praktycznej. Nie będę tu powielał szeroko dostępnych informacji o wewnętrznej budowie czujnika i interfejsie 1-Wire. W naszym urządzeniu do jednej magistrali 1-Wire dołączamy wiele czujników. Program najpierw musi wykryć ile czujników jest do niej dołączonych oraz odczytać i zapamiętać ich identyfikatory. Potem, na podstawie tablicy identyfikatorów, cyklicznie odczytywać i wyświetlać temperaturę każdego z czujników.

Napisanie procedury identyfikacji nie jest zadaniem łatwym, ale firma Maxim udostępnia notę katalogową numer 162 „*Interfacing DS18X20/DS1822 1-wire Temperature Sensor In a Microcontroller Enviroment*". Są w niej dokładnie opisane czynności, które należy wykonać żeby identyfikacja została wykonana prawidłowo. Oprócz wyczerpującego opisu zamieszczono też przykładowe fragmenty programów. Procedury obsługi magistrali 1-Wire i wyszukiwania termometrów na magistrali są napisane w oparciu o informacje zawarte w tej nocie.

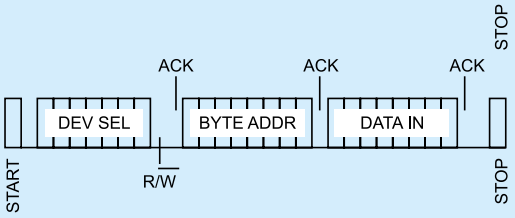
Na **listingu 9** pokazano procedury: zerowania magistrali *DS._reset*, zapisania bitu na magistrali *write_bit* oraz odczytanie bitu *red_bit*. Opóźnienia czasowe są generowane przez funkcję biblioteczną kompilatora MPLAB XC16.

Do realizacji magistrali 1-Wire wykorzystano właściwości portów mikrokontrolerów PIC. Kierunek przesyłania danych poprzez linię portu jest określony za pomocą ustawienia/wyzerowania odpowiedniego bitu w rejestrze TRISx. Jeżeli na przykład bit TRISB4 dla linii portu RB4 jest wyzerowany, to linia jest wyjściowa, a jeżeli ustawiony to linia jest wejściowa. Drugi rejestr PORT odpowiada

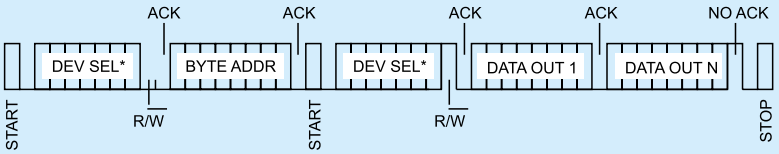
za odczytywanie poziomu na linii wejściowej lub zapisywanie wyjściowej. Załóżmy, że do rejestru PORTB4 linii RB4 zapiszemy zero, a linia będzie podciągnięta do plusa zasilania przez rezystor. Po wpisaniu do TRISB zera linia staje się wyjściową i wystąpi na niej poziom niski. Kiedy do TRISB4 wpiszemy jedynkę, to linia staje się wejściową i rezystor wymusza na niej poziom wysoki. Manipulowanie stanem linii jest realizowane przez zapisywanie rejestru TRISB4. Inicjalizacja portów pokazano na **listingu 10**.

Pokazana na **listingu 11** procedura *Find devices* wyszukuje czujniki dołączone do magistrali 1-Wire i tworzy tablicę *FoundROM* z odczytanymi numerami seryjnymi DS18B20. Procedury *First* i *Next* używane do wyszukiwania zostały pokazane na **listingach 12** i **13**.

Kiedy czujniki są zidentyfikowane, to trzeba z każdego z nich odczytać temperaturę i przekonwertować ją na wartość w stopniach Celsjusza. Odczytywanie temperatury realizuje funkcja *ReadRawTemp* pokazana na **listingu 14**. Jej argumentem jest kolejny numer urządzenia.



Rysunek 4. Zapis danej do pamięci



Rysunek 5. Sekwencja odczytania danych z pamięci

Działanie termometru

Zgodnie z założeniami, do magistrali 1-Wire można dołączać kilka czujników. Teoretycznie, ich liczba

Listing 8. Sekwencja odczytywania danej z pamięci EEPROM

```
unsigned char EERead(unsigned char addr)
{
    unsigned char  buf[15];
    IdleI2C2();
    StartI2C2();
    while(I2C2CONbits.SEN );  //Wait till Start sequence is completed
    MI2C2_Clear_Intr_Status_Bit; //Clear interrupt flag
     IdleI2C2();
    MasterWriteI2C2(0xA0);  //Write Slave address and set master for transmission
    while(I2C2STATbits.TBF);  //Wait till address is transmitted
    while(!IFS3bits.MI2C2IF);  //Wait for ninth clock cycle
    MI2C2_Clear_Intr_Status_Bit; //Clear interrupt flag
    while(I2C2STATbits.ACKSTAT);
    IdleI2C2();
    MasterWriteI2C2(addr);
    while(I2C2STATbits.TBF);  //Wait till address is transmitted
    while(!IFS3bits.MI2C2IF);  //Wait for ninth clock cycle
    MI2C2_Clear_Intr_Status_Bit;  //Clear interrupt flag
    while(I2C2STATbits.ACKSTAT);
    IdleI2C2();
    StartI2C2();
    while(I2C1CONbits.SEN );  //Wait till Start sequence is completed
    MI2C2_Clear_Intr_Status_Bit; //Clear interrupt flag
    IdleI2C2();
    MasterWriteI2C2(0xA1);  //Write Slave address and set master for receive
    while(I2C2STATbits.TBF);     //Wait till address is transmitted
    while(!IFS3bits.MI2C2IF);    //Wait for ninth clock cycle
    MI2C2_Clear_Intr_Status_Bit; //Clear interrupt flag
    while(I2C2STATbits.ACKSTAT);
    IdleI2C2();
    MastergetsI2C2(2,buf,1000); //Master recieves from Slave upto 10 bytes
    IdleI2C2();  //wait for the I2C to be Idle
    StopI2C2();  //Terminate communication protocol with stop signal
    while(I2C2CONbits.PEN);  //Wait till stop sequence is completed
    return(buf[0]);
    //CloseI2C1();  //Disable I2C
}
```

Listing 9. Procedury zerowania magistrali, oraz zapisu i odczytu bitu z magistrali

```
unsigned char DS_reset(void) {
    unsigned char presence;
    DQPORT=0;
    DQ=OUTPUT;
    DQ = 0; //pull DQ line low
    __delay_us(480); // stan niski przez  480us
    DQ=INPUT; // allow line to return high
    __delay_us(70); // czakaj na stan  presence
    presence = DQPORT; // odczytaj sygnał presence
    __delay_us(410); // czekaj na koniec szczeliny czasowej
    return(presence); // 0=presence, 1 = nie ma układu na magistrali
}

// WRITE_BIT - zapis bitu  bitval na magistrlę 1-wire
void write_bit(unsigned char bitval) {
    __delay_us(1);
    DQ=OUTPUT; //DQ w stan niski
    __delay_us(10);
    if(bitval==1) DQ =1; // DQ w stan wysoki, jeżeli zapisujemy jedynkę
    __delay_us(50); // stan zapisywanego bitu na magistrali przez całą szczelinę czasową
    DQ=INPUT; // zwolnienie DQ ( stan wysoki)
}

// READ_BIT - odczytanie bitu z magistrali 1-wire
unsigned char read_bit(void) {
    unsigned char retval;
    __delay_us(1);
    DQ=OUTPUT;//DQ=0 - start szczeliny czasowej
    __delay_us(2);
    DQ=INPUT; // zwolnienie magistrali
    __delay_us(10);
    retval=DQPORT;//odczytaj stan linii magistrali
    __delay_us(48); // dokończenie szceliny czasowej odczytu (min 60usek)
    return(retval);
}
```

Listing 10. Inicjalizacja linii DQ i wyszukiwanie czujników

```
#define OUTPUT 0
#define INPUT 1
#define DQ TRISBbits.TRISB4
#define DQPORT PORTBbits.RB4
void DS18B20Init(void)
{
    DQ=OUTPUT;
    DQPORT = 0;//zapisanie do RB4 stanu niskiego
    DQ=INPUT;
    DS_reset();     //zerowanie magistrali
    FindDevices(); //procedura wyszukiwania czujników temperatury
}
```

może być duża, ale ze względu na możliwości wyświetlania na zastosowanym wyświetlaczu nie powinno ich być więcej niż 4. Program dopuszcza zastosowanie maksymalnie 20 czujników. Po uruchomieniu mikrokontrolera są wykonywane inicjalizacje: wyświetlacza OLED, interfejsu I²C do komunikacji z pamięcią, interfejsu 1-Wire. Następnie jest wykonywana procedura *FindDevices* wyszukująca czujniki na magistrali i program przechodzi do wyświetlania jednego z trzech ekranów temperatury:

Listing 11. Wyszukiwanie czujników

```
// FIND DEVICES
void FindDevices(void) {
    unsigned char m;
    if(!DS_reset()) { // początek wyszukiwania, kiedy  wykryto presence
        if(First()) {
            numROMs=0;
            do {
                numROMs++;
                for(m=0;m<8;m++) {
                    FoundROM[numROMs][m]=ROM[m]; //Identifies ROM
                }

            }
            while (Next()&&(numROMs<MaxROMs)); //Continues until no additional devices are found
        }
    }
}
```

Listing 12. Wyszukiwanie pierwszego czujnika na magistrali

```
//wyszukiwanie pierwszego czujnika na magistrali
unsigned char First(void) {
    lastDiscrep = 0;
    doneFlag = FALSE;
    return Next();
}
```

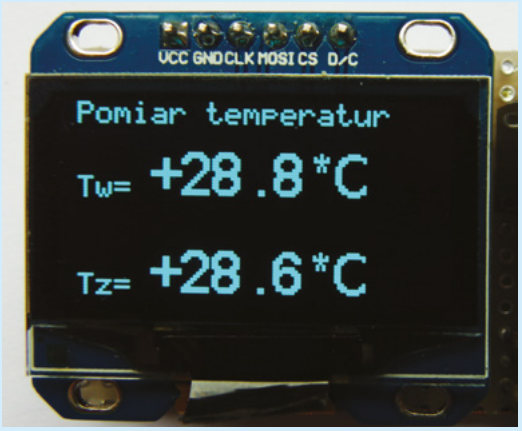
Listing 13. Wyszukiwanie kolejnych czujników

```
// NEXT
//wyszukiwanie kolejnych czujników na magistrali
//jeżeli nie ma czujników, to funkcja zwraca FALSE
unsigned char Next(void) {
    unsigned char m = 1;
    unsigned char n = 0;
    unsigned char k = 1;
    unsigned char x = 0;
    unsigned char discrepMarker = 0;
    unsigned char g;
    unsigned char nxt;
    int flag;
    nxt = FALSE;
    dowcrc = 0;
    flag = DS_reset(); // reset the 1-Wire
    if(flag||doneFlag) { //nie ma czujnika - powrót z FALSE
        lastDiscrep = 0; //
        return FALSE;
    }
    write_byte(0xF0); // wyślij komendę SearchROM
    do { // dla wszyskich bajtów
        x = 0;
        if(read_bit()==1) x = 2;
          __delay_us(120);
        if(read_bit()==1) x |= 1;
        if(x ==3) // nie ma czujników na magistrali 1-wire
            break;
        else {
            if(x>0)
                g = x>>1;
            else {
                    if(m<lastDiscrep) g = ((ROM[n]&k)>0);
                    else g = (m==lastDiscrep);
                    if (g==0) discrepMarker = m;
            }
            if(g==1)
                ROM[n] |= k;
            else
                ROM[n] &= ~k;
            write_bit(g);
            m++;
            k = k<<1;
            if(k==0) {
                ow_crc(ROM[n]);
                n++; k++;
            }
        }
    }
    while(n<8);
    if(m<65||dowcrc)
        lastDiscrep=0;    else {
        lastDiscrep = discrepMarker;
        doneFlag = (lastDiscrep==0);
        nxt = TRUE; //
    }
    return nxt;
}
```

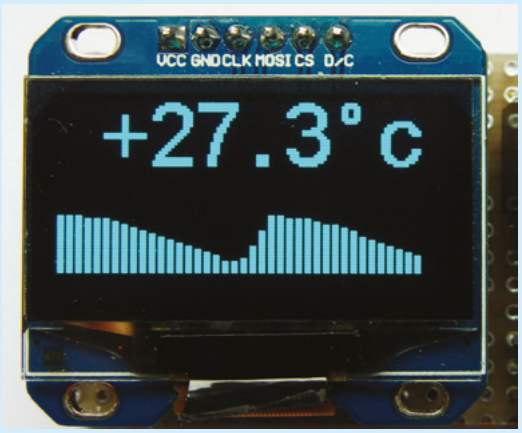
1. Ekran 1 (**fotografia 6**), na którym jest wyświetlana liczba wykrytych czujników, kanał pomiarowy, do którego dołączono funkcję termostatu, a w kolejnych wierszach temperatury mierzone przez dołączone czujniki.
2. Ekran 2 (**fotografia 7**), który jest przeznaczony do realizowania funkcji termometru domowego wyświetlającego dwie temperatury: wewnętrzna i zewnętrzną. Temperatura wewnętrzna jest odczytywana z pierwszego czujnika na liście, a temperatura zewnętrzna z drugiego czujnika na liście. Żeby „ekran 2" działał poprawnie, jest potrzebne dołączenie przynajmniej dwóch czujników do magistrali.
3. Ekran 3 (**fotografia 8**) spełnia funkcję termometru pokojowego z linijką przedstawiającą graficznie tendencje zmian temperatury w pomieszczeniu.

Po włączeniu zasilania z pamięci EEPROM jest odczytywany numer ekranu, który był aktywny przed wyłączeniem i zostaje on przywrócony. Przełączanie pomiędzy ekranami jest wykonywane za pomocą naciskania ośki impulsatora.

Fotografia 6. Ekran 1



Fotografia 7. Ekran 2



Fotografia 8. Ekran3

Do wyświetlania informacji na „ekranie 1” jest wykorzystana najmniejsza czcionka – 8×6 pikseli. Komunikaty nie są przez to dobrze czytelne, ale na ekranie można zmieścić najwięcej informacji. Na **listingu 15** pokazano procedurę realizująca funkcje „ekranu 1”. Na początku, z pamięci EEPROM są odtwarzane nastawy termostatu: temperatura progu, histereza i kanał termostatu. Potem jest wyświetlana liczba wykrytych czujników i kanał pomiarowy, do którego jest dołączony termostat. Pomiar temperatury i jej wyświetlanie są realizowane w pętli nieskończonej. Pomiar temperatury jest wykonywany co 2 sekundy. Po każdym z cykli pomiarowych sprawdzany jest warunek zadziałania termostatu.

„Ekran 2” do wyświetlania mierzonych temperatur używa czcionki o średniej wielkości 16×10 pikseli. Widoczność tak wyświetlanych informacji jest zadawalająca nawet z odległości kilku metrów. W założeniu „ekran 2” ma pełnić funkcję domowego tremometru do pomiaru temperatury wewnętrznej i zewnętrznej. Rozdzielczość pomiaru ograniczono do jednego miejsca po przecinku. Procedurę realizującą „ekran 2” zamieszczono na **listingu 16**.

„Ekran 3” używa największej czcionki do wyświetlania temperatury z pierwszego czujnika na liście. Oprócz wartości cyfrowej w dolnej części wyświetlacza jest wyświetlany rodzaj analogowego wskaźnika tendencji zmian temperatury. Po każdym pomiarze odczytana wartość jest reprezentowana przez słupek o szerokości 2 pikseli i wysokości od 1 do 24 pikseli.

Założyłem, że temperatura pokojowa będzie się zmieniała w zakresie od +18…30°C z rozdzielczością

Listing 14. Odczyt temperatury z rejestrów czujnika

```
unsigned int ReadRawTemp(unsigned char device) {
    int HighByte, LowByte;
    Send_MatchRom(device); // wybór urządzenia .
    write_byte( 0xbe); // komenda Read Scratchpad
    LowByte=read_byte(); //odczytanie  rejestrów
temperatury
    HighByte=read_byte();
    return (HighByte << 8) + LowByte;
}
//14.1 Odczyt temperatury
//wybór urządzenia
unsigned char Send_MatchRom(unsigned char DeviceNo)
{
    unsigned char i;
    if(DS_reset())
        return FALSE;
    write_byte(0x55); // match ROM
    for(i=0;i<8;i++) {
        write_byte(FoundROM[DeviceNo][i]); //send ROM
code
    }
    return TRUE;
}
```

Listing. 15 Funkcja ekran 1

```
void ekran1(void){
    char buf[20];
    short i;
    unsigned char chn, tst, kod;
    poc:tg=td=1;//bity sterownia termostatem
    EEWr(11,1);//aktywny ekran
    RestoreTermostat();//odtworzenie histerezy i progu
termostatu
    chn=EERead(21);//aktywny kanał termostatu

    OledCls();
    sprintf(buf,"ilosc czujnikow = %u",numROMs);//
wyświetlenie ilości czujników
    OledTxtRam(buf,1,0);
    sprintf(buf,"termostat kanal = %u",chn);//
wyświetlenie kanału termostatu
    OledTxtRam(buf,1,1);
    while (1) {//pętla główna
        ConvT(); // Konwersja temperatury
        tmsek=750;//oczekiwanie 750msek na zakończenie
pomiaru
        while(1){//pętla oczekiwania na akcję
            if(tmsek<=0)//zakończenie oczekiwania
750msek
            break;
        if(ST==0)//naciśnięcie ośki impulsatora
        {
            while(ST==0);//naciśnięty przycisk - koniec
funkcji
            __delay_ms(50);
            return;
        }
        kod=GetEncoder();
        if(kod==KOD_IMP_UP||kod==KOD_IMP_DWN)//obrót
oski - ustawianie  termostatu
        {  ekran4();//ustaw termostat
            OledCls();
            goto poc;}
        }
        for(i=1;i<=numROMs;i++) { // odczytanie pomiarów
ze wszystkich czujników
            temperature[i] = Get_Temp(i);
            sprintf (Temperatura,"T%u = %-2.2f
*C",i,temperature[i]);//wyświetlanie temperatur
            OledTxt(„              „,1,i+2);
            OledTxtRam(Temperatura,1,i+2);
            if(i==chn)//sprawdzenie warunku termostatu
            {
                tst=CheckTermostat(temperature[chn]);//
sprawdzanie termostatu
                if(tst==1)
                OledTxt(„(*)",16,chn+2);//wyświetlanie
wskaźnika działania termostatu
                if(tst==0)
                OledTxt(„   „,16,chn+2);
            }
        }
        tmsek=2000;//pomiar co 2 sekundy
        while(1){
        if(tmsek<=0) break;
        if(ST==0){
            while(ST==0);//naciśniecie oski - koniec
funkcji
            __delay_ms(50);
            return;}
        kod=GetEncoder();
        if(kod==KOD_IMP_UP||kod==KOD_IMP_DWN)//obrót
oski - ustawiane termostatu
        {ekran3();
        OledCls();
        goto poc;}
        }
    }
}
```

0,5°C. Daje to 12 stopni z rozdzielczością 0,5 stopnia, czyli 24 zmiany co 0,5°C. Przyrost o 0,5 °C powoduje zwiększenie wysokości słupka o 1 piksel. W podobny sposób można dobrać zmianę długości słupków dla innego zakresu temperatury.

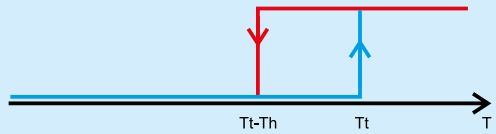
Maksymalna wysokość słupków (24 piksele) jest podyktowana rozdzielczością i wielkością matrycy wyświetlacza. Zwiększenie maksymalnej wysokości o kolejne 8 pikseli, a co za tym idzie zwiększenie dokładność wskazań, nie było możliwe do wyświetlenia.

Funkcja termostatu

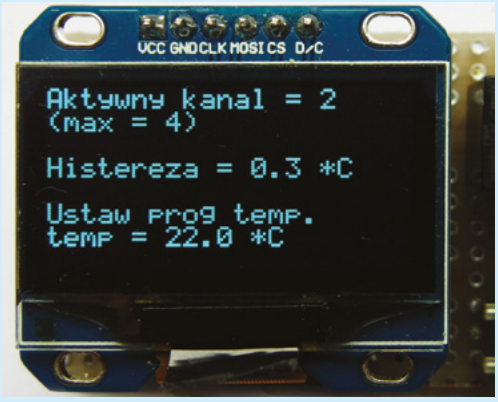
Termometr ma wbudowaną funkcję termostatu z histerezą. Działanie termostatu określają: kanał pomiarowy, dla którego włączono funkcję termostatu, temperatura progowa Tt i wartość histerezy. Jeżeli podczas inicjalizacji

Listing 16. Funkcja ekran 2

```
void ekran2(void){
    short i;
    unsigned char termostatus,chn, kod;
    poc:   td=tg=1;
    EEWr(11,2);  //ekran 2
    chn=EERead(21);  //aktywny kanał termostatu
    RestoreTermostat();  //odtworzenie z eeprom progu i histerezy
    OledCls();
    OledTxt(„Pomiar temperatur”,1,0);
    if(numROMs>1)  //więcej niż 1 czujnik
        OledTxt(„Tz=”,1,7);
    OledTxt(„Tw=”,1,3);
    while (1){
        ConvT(); // konwersja temperatury dla wszystkich czujników.
        tmsek=750; //750msek dla zakończenia konwersji
        while(1) {
            if(tmsek<=0) break;
            if(ST==0)//naciśnięcie ośki impulsatora
            {
                while(ST==0);
                __delay_ms(50);
                return;
            }
            kod=GetEncoder();
            if(kod==KOD_IMP_UP||kod==KOD_IMP_DWN)//obrot oski - ustawiane termostatu
            {
                ekran4(); //ustaw termostat
                OledCls();
                goto poc;}
            }
            for(i=1;i<=numROMs;i++) { odczytanie wszystkich temperatur
                temperature[i] = Get_Temp(i);
            }
            termostatus=CheckTermostat(temperature[chn]);//sprawdzenie warunku termostatu
            if(termostatus==1)
            {
                if(chn==1)
                OledTxt(„(*)”,18,3);
                if(chn==2) OledTxt(„(*)”,18,7);
            }
            if(termostatus==0){
                if(chn==1)
                OledTxt(„   „,18,3);
                if(chn==2) OledTxt(„   „,18,7);
            }
            sprintf(Temperatura, „T1=%-2.2f *C”, temperature[1]);
            if(Temperatura[4]==’.’){
                Temperatura[6]=Temperatura[5];
                Temperatura[5]=Temperatura[4];
                Temperatura[4]=Temperatura[3];
                Temperatura[3]=’0’;
            }
            DispTempMid(Temperatura+3,3,1);
            if(numROMs>1){
                sprintf(Temperatura, „T1=%-2.2f *C”, temperature[2]);
                if(Temperatura[4]==’.’) {
                    Temperatura[6]=Temperatura[5];
                    Temperatura[5]=Temperatura[4];
                    Temperatura[4]=Temperatura[3];
                    Temperatura[3]=’0’;
                }
                DispTempMid(Temperatura+3,3,3);
            }
            tmsek=2000;
            while(1){
                if(tmsek<=0) break;
                if(ST==0)
                {
                    while(ST==0);
                    __delay_ms(50);
                    return;
                }
                kod=GetEncoder();
                if(kod==KOD_IMP_UP||kod==KOD_IMP_DWN)//obrot oski - ustawiane termostatu
                {
                   ekran4();//ustaw termostat
                   OledCls();
                   goto poc;
                }
            }
        }
    }
}
```



Rysunek 9. Działanie termostatu



Fotografia 10. Ekran ustawień termostatu

temperatura jest niższa od temperatury progowej Tt, to przekaźnik jest załączony. Załóżmy, że temperatura rośnie i osiąga wartość Tt. Wtedy przekaźnik się wyłącza i jest wyłączony do momentu, kiedy temperatura nie spadnie do wartości Tt-Th (**rysunek 9**). Wprowadzenie histerezy jest niezbędne, bo w przeciwnym przypadku niewielkie zmiany temperatury w pobliżu Tt powodowałyby nieprzerwane załączanie i wyłączanie przekaźnika.

Termostat jest w pełni programowany. Można ustawić histerezę od 0,1°C do 2,5 °C z krokiem co 0,1 °C oraz temperaturę progową w całym zakresie pomiarowym. Do programowania jest przeznaczony „ekran 4" – **fotografia 10**. Można go wywołać z każdego z ekranów pomiaru temperatury przez obrót ośki impulsatora w dowolna stronę.

Programowanie termostatu rozpoczynamy od wybrania kanału pomiarowego. Termometr wykrył wcześniej ile czujników jest dołączonych i nie można wybrać kanału wyższego, niż maksymalny numer wykrytego kanału. Wszystkie parametry ustawia się przez obracanie, a zatwierdza przyciśnięciem ośki impulsatora. Po ustawieniu kanału przechodzimy do ustawiania histerezy, a następnie do ustawiania temperatury progowej. Nastawy są zapisywane do pamięci EEPROM i odtwarzane po włączeniu zasilania. Procedurę sprawdzania termostatu *CheckTermostat* pokazano na **listingu 17**. Jest ona wywoływana po każdym cyklu pomiaru temperatury. Jej argumentem jest wartość temperatury zmierzona w zaprogramowanym kanale pomiarowym. Termostat działa w zaprogramowanym kanale pomiarowym niezależnie od wybranego ekranu wyświetlania temperatury. Jeżeli wybierzemy „ekran 1" ze wszystkimi mierzonymi temperaturami, to po zadziałaniu termostatu (przekaźnik załączony) w wierszu z temperaturą kanału, do którego przypisano funkcje termostatu pojawi się symbol gwiazdki – **fotografia 11**.

Jeżeli do magistrali 1-Wire mamy dołączone 4 czujniki i termostat jest przypisany do kanału 4, a wyświetlamy pierwsze dwie temperatury „ekranem 2", to termostat będzie działał, ale na ekranie nie będzie żadnej informacji o tym czy przekaźnik zadziałał, czy nie. Jeżeli termostat będzie w kanale 1 lub 2, to symbol gwiazdki pojawi się przy wyświetlanych temperaturach.

Listing 17. Testowanie funkcji termostatu

```
unsigned char CheckTermostat(double temp)
{
    double pom;
    pom=temptermo-histermo;
    if(td)
    {
        if(temp>temptermo)
        {
            td=0;
            tg=1;//osiągnięto górny próg temperatury
            TERMO=0;//ustawić wyłączenie przekaźnika
            return(0);
        }
    }
    if(tg)
    {
        if(temp<pom)
        {
            td=1;//osiągnięto dolny próg temperatury
            tg=0;
            TERMO=1;//ustawić włączenie przekaźnika
            return(1);
        }
    }
    return(2);//powrot z bledem
}
```

Fotografia 11. Ekran 1 z sygnalizacja zadziałania termostatu

Montaż i uruchomienie

Prototyp termometru został zmontowany na kawałku uniwersalnej płytki drukowanej i w takiej postaci jest obecnie eksploatowany. Połączeń do wykonania jest sporo i dlatego zaprojektowałem płytkę drukowaną, której projekt można znaleźć w materiałach dodatkowych dołączonych do artykułu. Jest ona przystosowana do zamontowania w obudowie Z7AP. Montaż nie jest trudny, większość elementów jest przystosowana do montażu przewlekanego.

Mikrokontroler najlepiej zamontować w podstawce. Po zmontowaniu płytki trzeba ją zasilić napięciem stabilizowanym +5 V ze źródła o wydajności min 100 mA i sprawdzić poprawność napięcia +3,3 V. Jeżeli wszystko jest w porządku, to umieszczamy w podstawce zaprogramowany mikrokontroler, dołączamy wyświetlacz i czujniki DS18B20. Poprawnie zmontowany układ nie wymaga uruchamiania i regulacji. Czujniki temperatury dołączone do magistrali identyfikujemy poprzez ich podgrzewanie i obserwowanie wskazań zmian temperatury na „ekranie 1". Zidentyfikowane czujniki można oznaczyć (na przykład na kablu połączeniowym). Maksymalną długość kabla łączącego czujnik z termometrem trzeba wyznaczyć eksperymentalnie. Trzeba zawsze dążyć do tego, aby kabel łączący czujniki z termometrem był jak najkrótszy.

Tomasz Jabłoński, EP