W ofercie AVT*

AVT6047

Najważniejsze parametry:
- Konstrukcja oparta na mikrokontrolerze ATmega4808
- Pomiar temperatury otoczenia: czujnik DS18B20
- Dokładność pomiaru temperatury: ±0,5°C
- Sterowanie: enkoder obrotowy
- Wyświetlacz: LCD TFT 128×160 px
- Wyjścia przekaźnikowe: 2×DPDT 250 V/8 A
- Wbudowany zegar czasu rzeczywistego (DS3231) z podtrzymaniem (CR2032)

* **Uwaga!** Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania! Podstawową wersją zestawu jest wersja **[B]** nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji **[B]** zawiera elementy elektroniczne (w tym **[UK]** – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu. Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:

- **wersja [C]** – zmontowany, uruchomiony i przetestowany zestaw **[B]** (elementy wlutowane w płytkę PCB),
- **wersja [A]** – płytka drukowana bez elementów i dokumentacji.

Kity, w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- **wersja [A+]** – płytka drukowana **[A]** + zaprogramowany układ **[UK]** i dokumentacja,
- **wersja [UK]** – zaprogramowany układ.

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik PDF! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! **http://sklep.avt.pl**

W przypadku braku dostępności na stronie sklepu osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: **kity@avt.pl**.

Termostat do pieca gazowego

Projekt termostatu powstał z potrzeby chwili. W piecu gazowym uszkodził się stary, prosty termostat bazujący na bimetalu i trzeba było go czymś zastąpić. W nowoczesnych piecach stosuje się zaawansowane algorytmy regulacji z modulacją płomienia (mocy). W takim przypadku fabryczne regulatory komunikują się ze sterownikiem w piecu za pomocą własnościowych protokołów. Samodzielne konstruowanie zamienników takich urządzeń jest bardzo trudne, jeżeli w ogóle możliwe – dlatego właśnie powstał prezentowany w artykule projekt uniwersalnego, mikroprocesorowego termostatu.

Sterowanie naszym piecem odbywa się na prostej zasadzie włącz/wyłącz. Nie jest to zbyt efektywna metoda sterowania (piec pracuje albo z pełną mocą, albo nie pracuje wcale), lecz do podstawowych zastosowań okazuje się ona w zupełności wystarczająca. Termostat mierzy temperaturę w reprezentatywnym pomieszczeniu i jeżeli ta spadnie poniżej ustawionej wartości, to piec jest włączany. Po osiągnięciu docelowej temperatury piec jest wyłączany. Żeby zapobiec zbyt częstemu uruchamianiu pieca, w procesie sterowania wprowadzono programowaną histerezę. Zastosowanie sterownika mikroprocesorowego pozwoliło na dodanie możliwości programowania czasowego temperatur w trybie dzień/noc.

Założenia projektowe

Pomiar temperatury

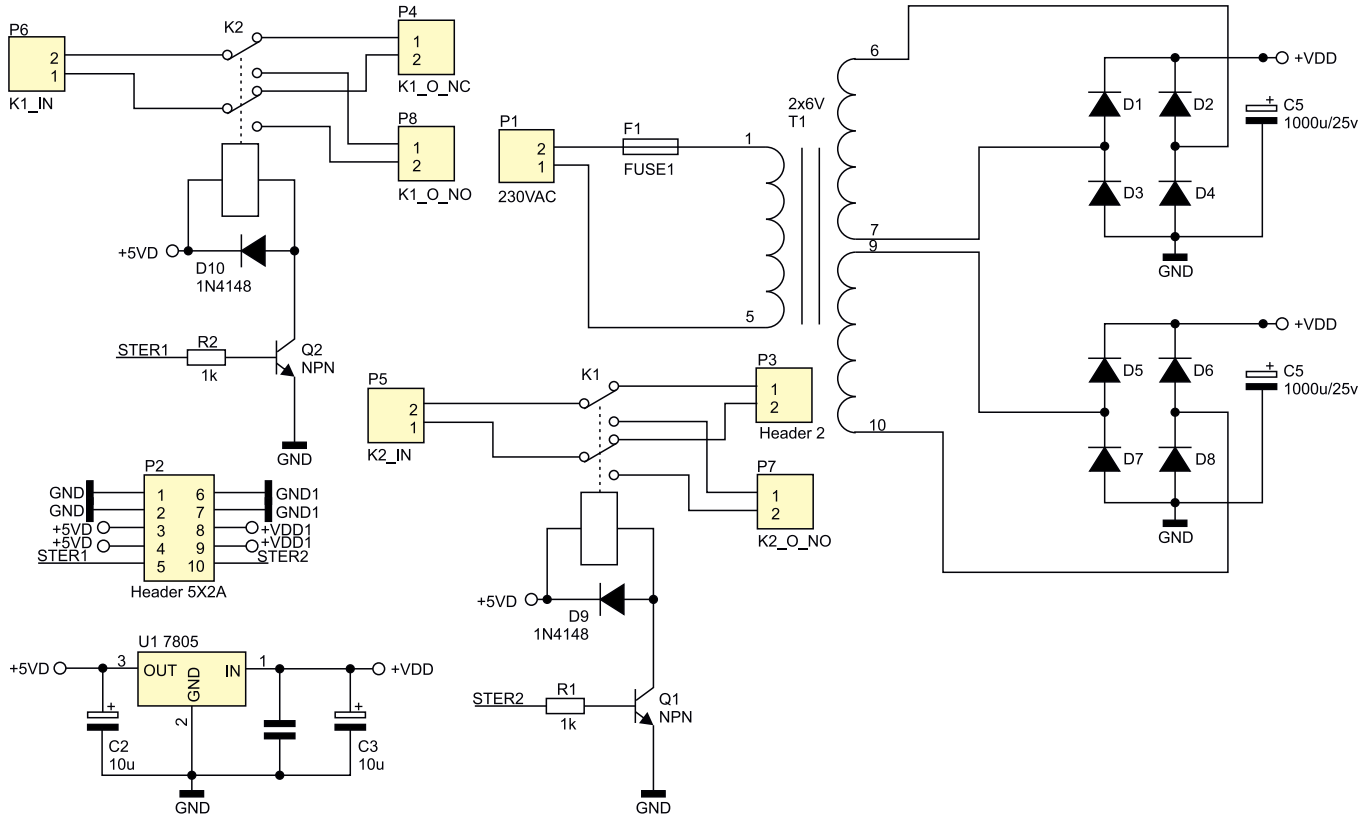
Pomiar temperatury otoczenia przez sterownik mikroprocesorowy wydaje się rzeczą banalną. Na rynku jest dostępnych wiele czujników elektronicznych komunikujących się za pomocą magistrali szeregowych SPI czy I²C. Stosowane są też czujniki konwertujące mierzoną temperaturę na napięcie, mierzone następnie przez przetwornik analogowo-cyfrowy mikrokontrolera i przeliczane na wartość temperatury. Ja wybrałem popularny i niezawodny czujnik DS18B20 z magistralą 1-Wire. Czujnik jest wystarczająco dokładny do realizacji pomiaru temperatury w pomieszczeniu (±0,5°C). Magistrala 1-Wire umożliwia umieszczenie czujnika w odległości nawet powyżej 1 metra od sterownika, co jest pożądaną cechą w tym zastosowaniu. Zbyt blisko umieszczony czujnik będzie wskazywał temperaturę zawyżoną przez ciepło wytwarzane przez układy termostatu. Przy odpowiednim oprogramowaniu do jednej magistrali można podłączyć wiele czujników i identyfikować je po numerach seryjnych.

Sterownik mikroprocesorowy

Wybór mikrokontrolera do budowy sterownika został podyktowany głównie zastosowanym interfejsem użytkownika. Ponieważ zdecydowałem się użyć kolorowego, graficznego wyświetlacza LCD, procesor musiał mieć odpowiednie zasoby – przede wszystkim sporą pamięć Flash, by móc pomieścić wzorce wyświetlanych znaków i dość rozbudowane oprogramowanie. Wybrany wyświetlacz jest sterowany przez magistralę SPI. Płynne wyświetlanie na ekranie zmieniających się danych wymaga pewnej minimalnej szybkości wykonywania operacji. Zdecydowałem się na szybki, 8-bitowy mikrokontroler ATmega4808 produkowany przez firmę Microchip (dawniej Atmel). Spełniał on wszystkie wymagania co do zasobów i szybkości. Poza tym, po przejęciu Atmela przez Microchip, można już pracować nad oprogramowaniem w znanym mi środowisku Microchip MPLAB X IDE, korzystając z bezpłatnej wersji kompilatora języka C (MPLAB XC8) oraz sprzętowego programatora-debuggera MPALB PicKit 4.

Zegar czasu rzeczywistego

Pomiar czasu rzeczywistego jest wykorzystywany do realizacji programu sterowania dzień/noc. Zegar RTC można zrealizować na kilka sposobów, zazwyczaj używając wbudowanego w mikrokontroler modułu RTC lub stosując zewnętrzny układ RTC komunikujący się z mikrokontrolerem przez magistralę I²C lub SPI (tak jest np. w przypadku układu DS1703). W każdej tego typu implementacji musimy zmierzyć się z problemem dokładności taktowania zegara przez układ rezonatora kwarcowego. To dość trudne zadanie do wykonania w warunkach



Rysunek 2. Schemat ideowy płytki zasilacza

amatorskich ze względu na trudności z zakupem odpowiedniego rezonatora kwarcowego. Zazwyczaj kupujemy rezonatory o nominalnej częstotliwości, nie wiedząc, jaka jest jego dokładność, dryft temperaturowy czy pojemność. W rezultacie zegar działa, ale jego dokładność jest w praktyce niezbyt dobra i trudno ją poprawić. Żeby uniknąć opisywanych problemów, zastosowałem układ zegara czasu rzeczywistego DS3231 z wbudowanym w strukturę kalibrowanym i kompensowanym termicznie oscylatorem kwarcowym. Bez żadnych zabiegów układ zapewnia dokładność na poziomie ±2ppm w temperaturze od 0°C do 40°C, co daje w teorii błąd ±1 min na rok!

Konstrukcja układu

Układ termostatu został umieszczony na dwóch płytkach: PCB sterownika mikroprocesorowego i PCB zasilacza z układem wykonawczym sterującym piecem. Schemat tej pierwszej został pokazany na **rysunku 1**.

Podstawowym elementem urządzenia jest mikrokontroler ATmega4808 (układ U1) sterujący działaniem wyświetlacza graficznego za pomocą magistrali SPI. Magistrala sterująca, oprócz standardowych linii zegara LCD_SCK, LCD_MOSI (dane wyjściowe z mikrokontrolera) i linii wyboru interfejsu LCD_CS, jest uzupełniona o linie zerowania sterownika wyświetlacza LCD_RES, linie wyboru rodzaju przesyłanych danych (dane wyświetlane lub komendy) LCD_DC i linię wygaszania ekranu LCD_BLK. Zastosowany w projekcie moduł wyświetlacza z kolorową matrycą TFT o rozmiarze 128×160 pikseli ma wbudowany sterownik ST7735. Interfejs użytkownika – oprócz wyświetlacza – zawiera także impulsator (enkoder) obrotowy ze stykiem zwieranym przez przyciśnięcie pokrętła. Rezystory R2 i R4 wymuszają stan wysoki na wejściowych liniach PD0 i PD1 (odczytujących stany wyprowadzeń A i B enkodera), a rezystor R1 wymusza stan wysoki na linii wejściowej PD2, odpowiedzialnej za odczyt stanu przycisku.

Zegar RTC typu DS3231 (U3) komunikuje się z mikrokontrolerem przez magistralę I²C. Rezystory R5 i R6 zapewniają wymagane przez standard I²C podciąganie linii SDA i SCL do plusa zasilania 3,3 V. Po zaniku głównego napięcia +3,3 V układ zegara jest zasilany z baterii litowej CR2032 o napięciu 3 V. Na płytce sterownika jest umieszczone specjalne gniazdo do zamocowania tej baterii, co pozwala podtrzymać bieg układu czasowego i niweluje problem konieczności ponownego ustawiania aktualnej godziny po zaniku napięcia sieciowego. Napięcie o wartości 3,3 V, służące do zasilania układów mikrokontrolera, wyświetlacza graficznego i zegara RTC, jest wytwarzane przez scalony stabilizator LDO (U2) typu 1117-3.3 Na wejście stabilizatora jest podawane napięcie +5 V z płytki zasilacza przez złącze szpilkowe P3. Czujnik temperatury jest podłączany do wyprowadzeń złącza P4, a rezystor R8 realizuje wymagane przez standard 1-Wire podciąganie linii danych DQ do plusa zasilania. Złącze J2 umożliwia podłączenie programatora/debuggera PicKit4. Do programowania układu jest wykorzystywana jedna dwukierunkowa linia UPDI.

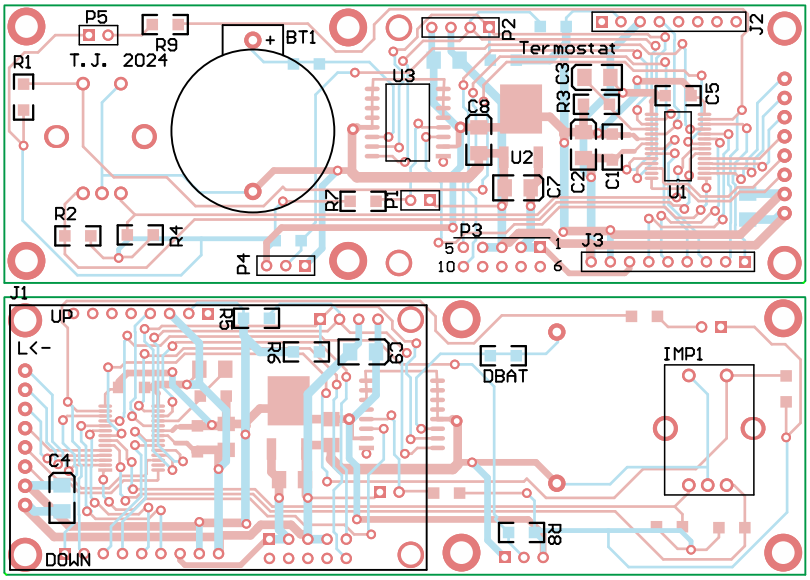
Schemat płytki zasilacza został pokazany na **rysunku 2**.

Napięcie sieciowe 230 V jest podłączone przez złącze P1 i bezpiecznik F1 do uzwojenia pierwotnego transformatora sieciowego T1. Napięcie wtórne o nominalnej wartości skutecznej 6 V jest prostowane w mostku Greatza zbudowanym z diod D5...D8 typu 1N4007 i filtrowane kondensatorem C5 o pojemności 1000 µF. Wyprostowane i odfiltrowane napięcie trafia na wejście stabilizatora U1 typu uA7805. Napięcie +5 V z jego wyjścia zasila (poprzez złącze P2) układy płytki sterownika oraz cewki przekaźników wykonawczych K1 i K2, które odpowiedzialne są za sterowanie załączaniem/wyłączaniem pieca. Przewidziano dwa kanały sterujące z dwoma przekaźnikami, ale w obecnej wersji w użyciu jest tylko jeden kanał. Cewki przekaźników są sterowane tranzystorami Q1 i Q2. Podanie napięcia +3,3 V z portu mikrokontrolera (przez rezystory ograniczające R1 lub R2) na bazy tranzystorów sterujących powoduje wprowadzenie ich w stan nasycenia i zadziałanie przekaźnika. Diody D9 i D10 tłumią przepięcia indukowane w cewkach przekaźników w momencie wyłączania napięcia i zaniku prądu płynącego przez uzwojenia cewek.

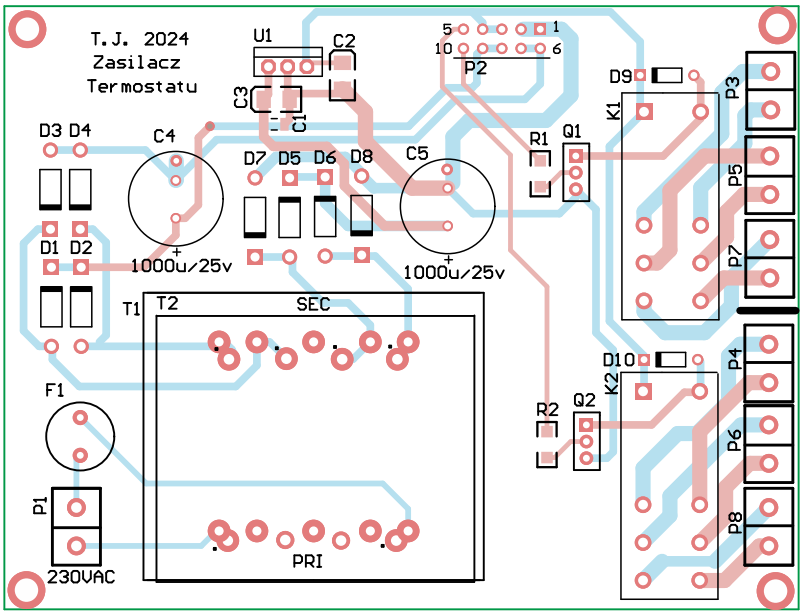
Zastosowany transformator ma dwa uzwojenia wtórne o napięciach nominalnych 6 V każde. Jedno z nich nie jest wykorzystywane w termostacie, ale na płytce przewidziano miejsce na jeszcze jeden mostek prostowniczy i kondensator filtrujący. Tych elementów nie trzeba montować w standardowej wersji opisywanego urządzenia.

Montaż układu

Jak już wspomniałem, termostat jest montowany na dwóch dwustronnych płytkach: sterownika (**rysunek 3**) i zasilacza (**rysunek 4**).



Rysunek 3. Schemat montażowy płytki sterownika termostatu



Rysunek 4. Schemat montażowy płytki zasilacza

Płytka sterownika

Montaż płytki sterownika rozpoczynamy od najniższych elementów na stronie elementów (warstwa top): mikrokontrolera, układu zegara RTC i elementów biernych: rezystorów i kondensatorów. Wszystkie te komponenty są w obudowach do montażu powierzchniowego. Ze względu na mały rozstaw wyprowadzeń należy zachować ostrożność przy lutowaniu mikrokontrolera ATmega4808. Wyświetlacz, impulsator i kilka elementów biernych należy zamontować na warstwie dolnej (bottom). Lutujemy je po wmontowaniu wszystkich elementów na warstwie top. Zaczynamy od elementów biernych, bo większość z nich jest umieszczona pod wyświetlaczem i po jego zamocowaniu nie będzie do nich dostępu. Wyprowadzenia wyświetlacza wlutowujemy w złącze J3, a następnie przykręcamy go do płytki za pomocą wkrętów M2,5 i tulejek dystansowych. Zmontowaną płytkę sterownika można zobaczyć na **fotografii 1**.

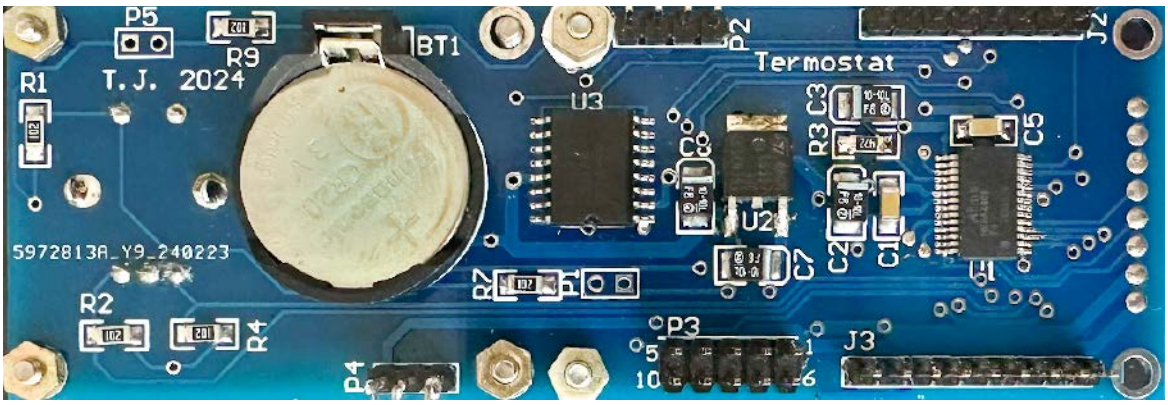
Płytka zasilacza

Montaż płytki zasilacza jest stosunkowo prosty. Większość elementów jest przeznaczona do montażu przewlekanego. Wyjątkiem są kondensatory blokujące przy stabilizatorze 7805 oraz rezystory R1 i R2 ograniczające prąd tranzystorów Q1 i Q2. Zmontowana płytka zasilacza została pokazana na **fotografii 2**.

Gotowe płytki zostały umieszczone w małej, plastikowej obudowie. Połączone są za pomocą złączy IDC10 zaciskanych na płaskim kablu AWG28 (**fotografia 3**). To połączenie umożliwia zasilanie płytki sterownika napięciem +5 V i przesyłanie sygnału sterującego ze sterownika do tranzystora sterującego przekaźnikiem wykonawczym.

Uruchomienie i działanie układu

Uruchomienie układu zaczynamy od płytki zasilacza. Po podłączeniu sieci 230 V sprawdzamy obecność napięcia wyjściowego +5 V (wyprowadzenia 1 i 2 złącza P2) względem masy (wyprowadzenia 3 i 4 złącza P2). Jeżeli napięcie to jest prawidłowe, łączymy obie płytki złączami IDC. Napięcie na wyjściu stabilizatora U2 – umieszczonego na płytce sterownika – powinno wynosić +3,3 V. W następnym kroku do złącza J2 wpinamy programator umożliwiający zaprogramowanie mikrokontrolera Atmega4808 za pomocą jednej linii UPDI. Ja użyłem programatora MPLAB PICkit4 współpracującego ze środowiskiem projektowym MPLABX IDE. Po podłączeniu zewnętrznego czujnika DS18B20 do złącza P4 układ jest gotowy do weryfikacji działania oraz rozpoczęcia programowania ustawień.



Fotografia 1. Zmontowana płytka sterownika

Wykaz elementów:

Płytka sterownika

Rezystory:
R1, R2, R4, R5...R9: 1 kΩ
R3: 10 kΩ

Kondensatory:
C1, C5: 100 nF
C2...C4, C7...C9: 10 µF

Półprzewodniki:
U1: ATmega4808_28
U2: SPX1117
U3: DS3231
DS18B20
dioda LED

Pozostałe:
BT1: bateria CR2032 + podstawka THT pozioma 2-pin.
P1, P5: listwa goldpin 1×2
P2: listwa goldpin 1×4
P4: listwa goldpin 1×3
P3: listwa goldpin 2×5
J1, J2: goldpin 1×8
J3: goldpin 1×9
wyświetlacz LCD TFT 128×160 px ze sterownikiem ST7735

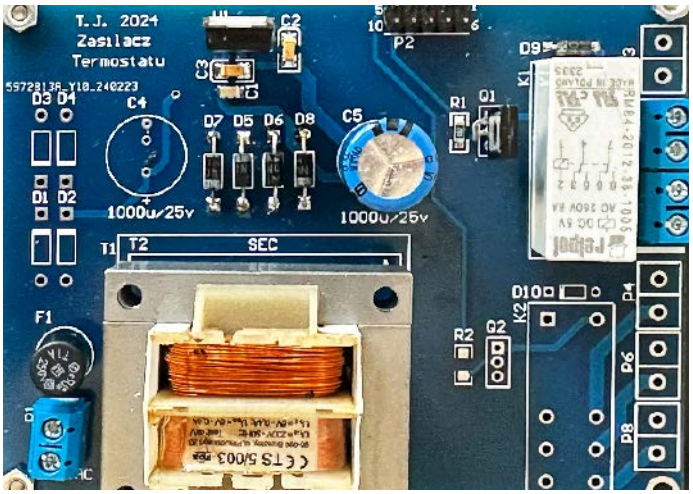
Płytka zasilacza

Rezystory:
R1, R2: 1 kΩ

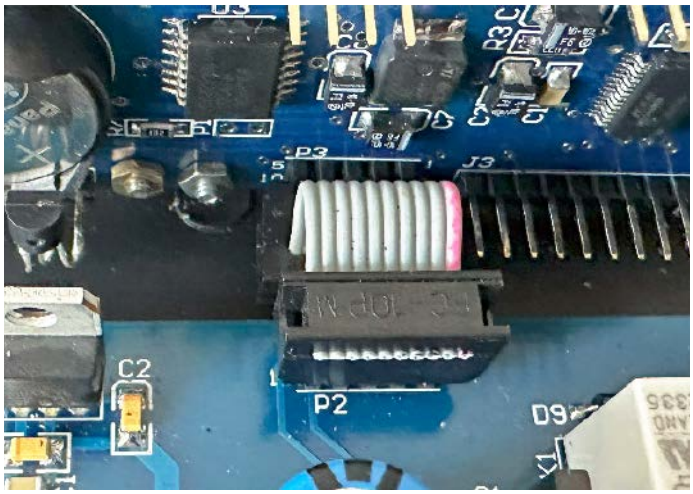
Kondensatory:
C1: 100 nF
C2, C3: 10 µF
C4, C5: 1000 µF/25V

Półprzewodniki:
D1...D10: 1N4007
U1: 7805 (TO-220)
Q1, Q2: tranzystor NPN (TO-126)

Pozostałe:
F1: bezpiecznik
K1, K2: przekaźnik RM84-2012-36-1005
P1, P3...P8: złącze ARK2 5 mm 2 pin.
P2: listwa goldpin 2×5
T1: transformator sieciowy 2×6 V



Fotografia 2. Zmontowana płytka zasilacza



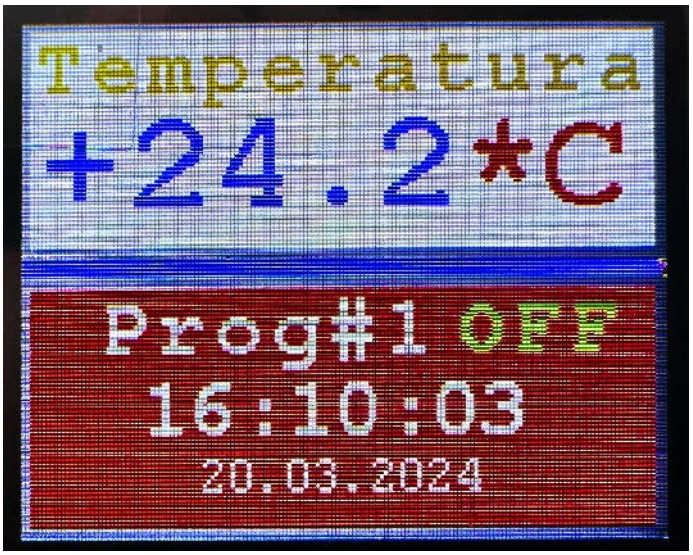
Fotografia 3. Połączenie obu płytek taśmą z wtykiem IDC10

Po włączeniu zasilania pojawia się ekran główny podzielony na dwie strefy. W pierwszej z nich wyświetla się temperatura mierzona przez czujnik DS18B20. Wynik jest aktualizowany co 1 sekundę (co wynika z czasu potrzebnego na dokonanie pomiaru). W drugiej strefie wyświetlane są natomiast: rzeczywisty czas w godzinach, minutach i sekundach, data, a także bieżący program termostatu.

Ten ostatni działa w dwóch przedziałach czasowych, umownie nazwanych: „dzień" i „noc". Do każdego z tych przedziałów przypisana została wartość temperatury. Z założenia w czasie „dzień" ustawiana jest wyższa temperatura, właściwa dla aktywności domowników (może to być na przykład +21°C). W czasie „noc" ustawiamy temperaturę niższą, na przykład +18°C. Przekłada się to niższe zużycie gazu, a dodatkowo w niższej temperaturze lepiej się śpi.

Przedział czasowy „dzień" zaczyna się o godzinie określonej przez Czas1 i kończy się o godzinie określonej przez Czas2. W tym przedziale wykonywany jest program o nazwie „Program #1" z przypisaną temperaturą regulacji Temp1.

Przedział czasowy „noc" zaczyna się o godzinie Czas2, a kończy o godzinie Czas1. W tym przedziale z kolei wykonywany jest czas regulacji „Program #2" z przypisaną Temp2. Wszystkie parametry programu: Czas1, Czas2, Temp1 i Temp2 są programowane. Jedyne ograniczenie bieżącej wersji oprogramowania stanowi warunek, że Czas1 musi być mniejszy od Czas2. Na przykład Czas1=07:00, Czas2=22:15, ale Czas2 nie powinien mieć na przykład wartości 0:30. Wtedy program regulacji będzie działał, ale po zaniku zasilania może błędnie określić, w którym przedziale czasowym się znajduje, aż do momentu osiągnięcia jednego z czasów: Czas1, lub Czas2. Na **rysunku 5** pokazano zasadę czasowego podziału doby na dwa programy Program #1 i Program #2. W czasie wykonywania programu #1 termostat utrzymuje temperaturę Temp1, a w czasie trwania programu #2 – temperaturę Temp2.



Fotografia 4. Ekran główny termostatu

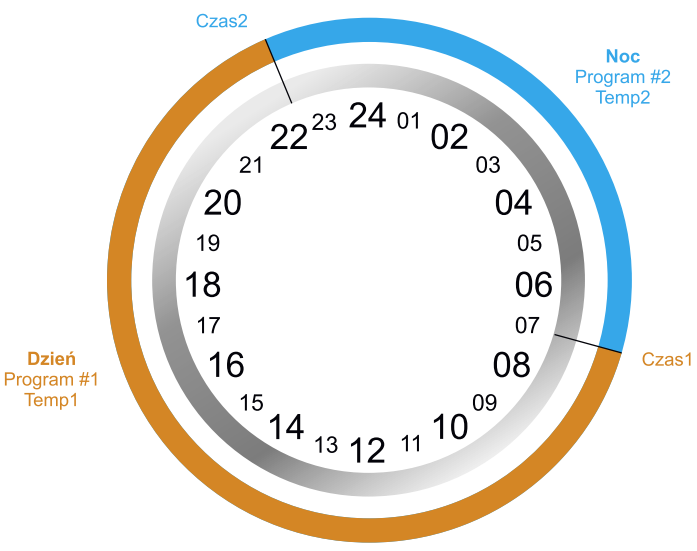
Temperatura jest regulowana przy zastosowaniu pętli histerezy. Załóżmy, że temperatura otoczenia okazuje się wyższa od temperatury Temp1. Termostat wyłącza wówczas grzanie pieca. Temperatura otoczenia spada, osiąga wartość Temp1. Termostat nadal nie włączy pieca – tak długo, aż temperatura otoczenia spadnie do wartości Temp1 – Hist, gdzie Hist jest wartością programowaną w zakresie od 0,1°C do 2°C z krokiem 0,1°C. Jeżeli temperatura otoczenia jest niższa od Temp1 – Hist, to termostat włącza grzanie. Temperatura w pomieszczeniu rośnie, aż osiągnie wartość wyższą od Temp1. Wtedy termostat wyłącza grzanie i cykl się powtarza. Takie rozwiązanie zapobiega częstym cyklom załącz-wyłącz, kiedy temperatura jest bliska wartości Temp1. Temperatura w pomieszczeniu będzie się w przybliżeniu wahać od wartości Temp1-Hist do Temp1. Wartość histerezy należy dobrać w zależności od warunków w ogrzewanych pomieszczeniach. Zbyt duża powoduje duże wahania temperatury, zbyt mała natomiast – częste cykle załącz-wyłącz. Prototyp termostatu pracował z histerezą 0,8°C. Histereza pozostaje wspólna dla obu programów.

Menu programowania jest dostępne po przyciśnięciu osi impulsatora. Można w nim ustawić czas, datę i termostat – **fotografia 5**.

Wybór jednej z funkcji w menu ustawień (czas, data, termostat) realizowany jest przez obrót gałki. Wybrana funkcja wyświetlana się na biało. Przyciśnięcie osi impulsatora sprawia, że rozpoczyna się wykonywanie wybranej funkcji.

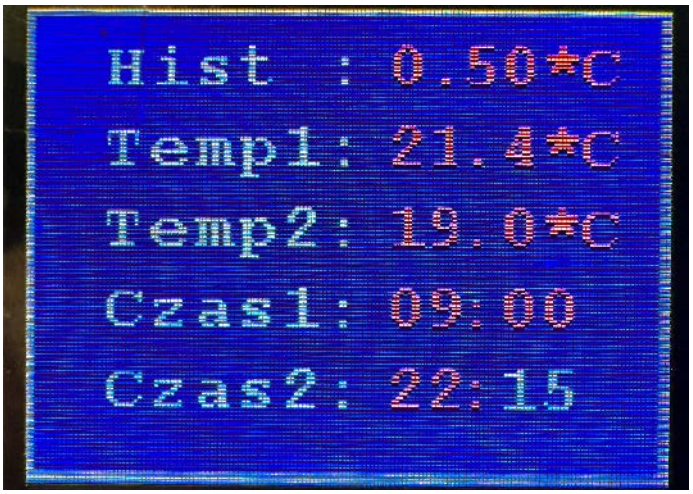
Najbardziej rozbudowana jest funkcja ustawień termostatu. Można w niej zaprogramować:

- histerezę w zakresie od 0,1°C do 2,0°C z krokiem 0,1°C,
- temperaturę Temp1 w zakresie od 0°C do 31°C z krokiem 0,1°C,
- temperaturę Temp2 w zakresie od 0°C do 31°C z krokiem 0,1°C,
- czas 1 (godziny, minuty),
- czas 2 (godziny, minuty).



Rysunek 5. Zasada działania programu termostatu

Fotografia 5. Menu ustawień



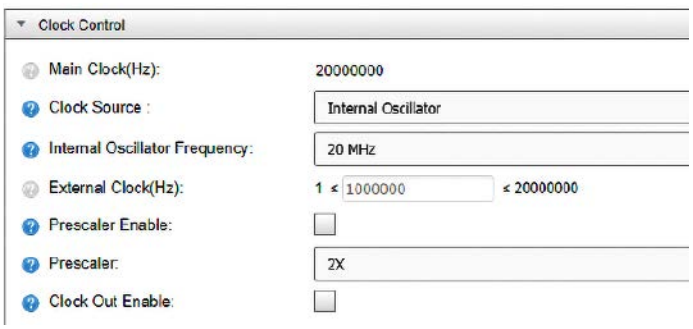
Fotografia 6. Przykładowy ekran ustawień termostatu

Nastawy zmienia się przez obrót osi impulsatora, a zatwierdza jej przyciśnięciem. Ustawiana wartość wyświetlana się na biało, a po zatwierdzeniu zmienia kolor na niebieski – **fotografia 6.**

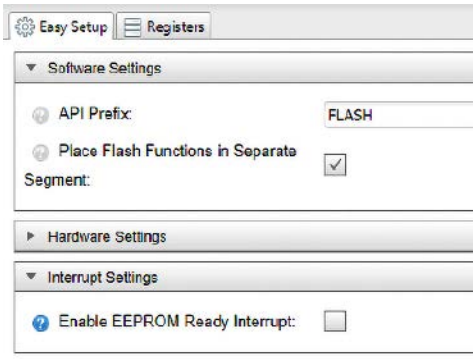
Po wejściu w menu ustawień termostatu użytkownik nie ma możliwości wybrania tylko jednej z nastaw. Musi „przeklikać" wszystkie pozycje. Sterownik pamięta każdą nastawę i jeżeli jakiejś nie chcemy modyfikować, powinniśmy sekwencyjnie przyciskać oś impulsatora, żeby przejść do kolejnej nastawy bez wprowadzania zmian. To uproszczenie nie wpływa znacząco na komfort użytkowania, ponieważ programowania termostatu nie wykonuje się często. Wszystkie nastawy są zachowywane w pamięci nieulotnej mikrokontrolera i odtwarzane po każdym zaniku napięcia zasilania. Wraz z podtrzymaniem bateryjnym zegara RTC zapewnia to w miarę niezakłócony cykl sterowania przy chwilowych zanikach napięcia sieciowego. Sterownik – po ponownym włączeniu zasilania – na podstawie ustawień Czas1 i Czas2 oraz bieżącego wskazania zegara oblicza czy aktywny jest Program #1, czy Program #2 i zaczyna regulować nastawione temperatury właściwe aktywnemu programowi.

Ustawienia czasu oraz daty nie wymagają szerszego komentarza. W trybie ustawiania czasu programujemy kolejno: godziny i minuty. Po ustawieniu minut oraz przyciśnięciu osi impulsatora, zerowany jest licznik sekund – i wartości godzin, minut i sekund są wpisywane do rejestrów układu DS3231, a zegar zaczyna odliczać nowy czas. W funkcji ustawienia daty programowane są: dzień miesiąca, miesiąc, a także rok.

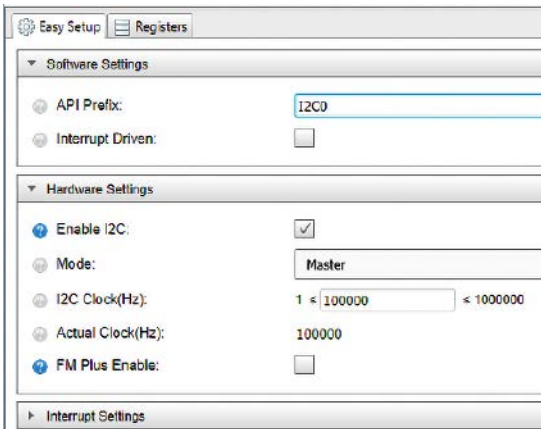
Jak już wspomniałem, oprogramowanie zostało napisane w środowisku MPLAB firmy Microchip. Oprócz oczywistych elementów – takich jak środowisko IDE i kompilator C – istotnym wsparciem dla programisty jest wtyczka konfiguracyjna MCC. Oprócz konfiguracji urządzeń peryferyjnych, dostarcza kody źródłowe procedur do ich obsługi, co znacznie ułatwia i skraca pisanie programu. Trzeba przy tym pamiętać, że większość procedur komunikacji sprawdza warunek zakończenia transmisji w nieskończonych pętlach (procedury blokujące). Warto je nieco zmodyfikować, tak by – po określonym czasie oczekiwania w pętli na spełnienie warunku – tę pętlę przerwać. Jeżeli tego nie zrobimy ryzykujemy, że program prędzej czy później trwale się zawiesi. Można również pozostawić go tak jak jest – i skonfigurować moduł watchdoga.



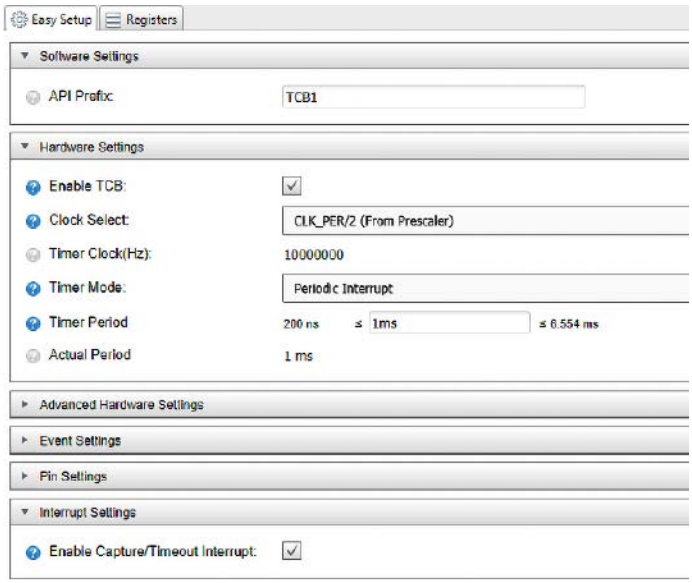
Rysunek 6. Konfiguracja układu taktowania



Rysunek 7. Moduł NVMCTRL



Rysunek 8. Konfiguracja TWI (I²C master)



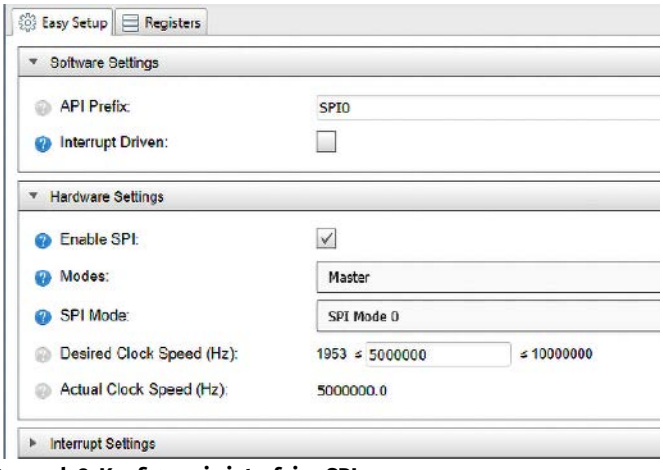
Rysunek 10. Konfiguracja Timera TCB

```
/***************************************************
zapisanie bajtu do pamięci EEPROM
***************************************************/
uint8_t HMI_WriteEEPROM(uint8_t address,uint8_t data)
{
    uint8_t status;
    status = FLASH_WriteEepromByte(address, data);
    return status;
}
/***************************************************
zapisanie bajtu do pamięci EEPROM
-procedura wygenerowana przez MMC
***************************************************/
/**
* \brief Write a byte to eeprom
*
* \param[in] eeprom_adr The byte-address in eeprom to write to
* \param[in] data The byte to write
*
* \return Status of write operation
*/
nvmctrl_status_t FLASH_WriteEepromByte(eeprom_adr_t eeprom_adr, uint8_t data)
{
/* Wait for completion of previous write */
while (NVMCTRL.STATUS & NVMCTRL_EEBUSY_bm)
;

/* Clear page buffer */
ccp_write_spm((void *)&NVMCTRL.CTRLA, NVMCTRL_CMD_PAGEBUFCLR_gc);

/* Write byte to page buffer */
*(uint8_t *)(EEPROM_START + eeprom_adr) = data;

/* Erase byte and program it with desired value */
ccp_write_spm((void *)&NVMCTRL.CTRLA, NVMCTRL_CMD_PAGEERASEWRITE_gc);

        if (NVMCTRL.STATUS & NVMCTRL_WRERROR_bm)
        {
            return NVM_ERROR;
        }
        else
        {
            return NVM_OK;
        }
}
```

Listing 1. Zapisanie danej do pamięci EEPROM

```
static void LCD_WriteData_NLen16Bit(uint16_t Data,uint32_t DataLen)
{
uint32_t i;
PORTC.OUT |= (1 << DC);                          //zapis danej
    PORTC.OUT &= ~(1 << CS);                     //SPI aktywny
for(i = 0; i < DataLen; i++){
SPI0_ExchangeByte( (uint8_t)(Data >> 8) );
SPI0_ExchangeByte( (uint8_t)(Data & 0XFF) );
}
PORTC.OUT |= (1 << CS);                          //SPI nie aktywny
}
```

Listing 3. Przesłanie dwu bajtów do pamięci wyświetlacza



Rysunek 9. Konfiguracja interfejsu SPI

```
uint8_t SPI0_ExchangeByte(uint8_t data)
{
    SPI0.DATA = data;
    while (!(SPI0.INTFLAGS & SPI_RXCIF_bm));
    return SPI0.DATA;
}
```

Listing 4. Funkcja wysyłająca jeden bajt przez SPI

Mikrokontroler jest taktowany wewnętrznym oscylatorem o częstotliwości 20 MHz. Przy zasilaniu +3,3 V nominalna częstotliwość taktowania powinna być ograniczona do 10 MHz, ale w praktyce – w temperaturach pokojowych – mikrokontroler pracuje poprawnie przy 20 MHz (okazuje się to istotne dla w miarę płynnej pracy wyświetlacza). Układ taktowania jest konfigurowany w oknie Clock Control wtyczki konfiguratora MCC – **rysunek 6**.

Mikrokontroler ma wbudowane 256 bajtów nieulotnej pamięci EEPROM przeznaczonej do przechowywania danych systemowych. Dostęp do pamięci nieulotnej – w tym EEPROM (ale też i Flash) – zapewnia specjalny moduł NVMCTRL (Nonvolatile Memory Controller). Po zaznaczeniu *Place Flash Functions in Separate* w oknie NVMCTRL (**rysunek 7**), wtyczka MCC wygeneruje funkcje inicjalizujące moduł NVMCTRL oraz funkcje zapisu i odczytu bajtu z pamięci EEPROM (funkcje te używane są do zapisywania i odczytywania ustawień termostatu – **listing 1**).

Zegar RTC DS3231 komunikuje się z mikrokontrolerem za pomocą magistrali I²C. Transmisję obsługuje sprzętowy moduł TWI konfigurowany w MCC – **rysunek 8**. Pracuje on jako master I²C z zegarem o częstotliwości 100 kHz.

W przypadku interfejsu TWI, MCC dostarcza plik twi0_master_example.c z działającymi funkcjami transferu danych po magistrali I²C. Na **listingu 2** pokazano funkcję przesyłania danych do zegara RTC DS3231, korzystającą z „przykładowej" funkcji I2C0_example_write1ByteRegister.

Mikrokontroler przesyła dane do sterownika wyświetlacza poprzez magistralę SPI. Wyświetlacz ma rozmiar 160×128 pikseli – żeby określić jasność i kolor każdego z pikseli, mikrokontroler musi przesłać do sterownika 2 bajty. Zapisanie całego wyświetlacza wymaga zatem transferu 128×160×2=40960 bajtów. Ponadto trzeba często wysłać sekwencje komend sterujących, na przykład do ustalenia adresu piksela. Wszystko to sprawia, że transfer danych powinien być szybki,

```
#define SPI_FRAME 0xD00
uint8_t SPI0_ExchangeByte(uint8_t data)
{
uint16_t frame = 0;
    SPI0.DATA = data;
    while (1)
   {
     if!(SPI0.INTFLAGS & SPI_RXCIF_bm)
return SPI0.DATA;                          //dana prawidłowo wysłana

    ++ frame;
    if (frame >= SPI_FRAME)
{
SPI SPI0_Initialize ();
return(0);
    }
    }
    return SPI0.DATA;
}
```

Listing 5. Zmodyfikowana procedura wysyłania bajtu

```
/***************************************************
zapisz danej do rejestru o adresie reg
***************************************************/
void RTC_SendData(uint8_t reg, uint8_t data)
{
    I2C0_example_write1ByteRegister(RTC_I2C_ADD, reg, data);
}
void I2C0_example_write1ByteRegister(twi0_address_t address, uint8_t reg, uint8_t data)
{
    while(!I2C0_Open(address));                                          //sit here until we get the bus.
    I2C0_SetDataCompleteCallback(wr1RegCompleteHandler_example,&data);
    I2C0_SetBuffer(&reg,1);
    I2C0_SetAddressNackCallback(I2C0_SetRestartWriteCallback,NULL);      //NACK polling?
    I2C0_MasterWrite();
    while(I2C0_BUSY == I2C0_Close());                                    //sit here until finished.
}
```

Listing 2. Funkcja przesłania danej do zegara RTC

```
void Termostat(double temperature)
{
double hist, temp1, temp2;
      HMI_GetTermostat();                       //pobierz dane ustawień
                                                //termostatu
   hist = (double)termo.hist/100;
   temp1 = (double)termo.temp1/10;
   temp2 = (double)termo.temp2/10;


   Termostat_Check_Time ();                     //sprawdź czy nie zmienił
                                                //się program
    Termostat_Check_Temperature (temperature);
}
```

Listing 6. Funkcja termostatu

```
//odczytanie czasu i sprawdzenie który program mam być aktywny
//przy zmianie progów w czasowych

void Termostat_Check_Time (void)
{
    uint8_t min, hour;
    RTC_GetTime ();                                     //pobierz czas z rejestrów DS3231
    hour = HMI_ConvHexDec (time.hour);
    min =  HMI_ConvHexDec (time.min);
    if (hour == termo.t1_hour && min == termo.t1_min)
    {
        termo.program = PR1;
        return ;
    }
    else
    if (hour >termo.t1_hour && hour < termo.t2_hour)
    {
        termo.program = PR1;
        return ;
    }
    else
    if (hour == termo.t2_hour && min < termo.t2_min)
    {
        termo.program = PR1;
        return ;
    termo.program = PR2
}
```

Listing 7. Identyfikacja aktywnego programu czasowego

```
void Termostat_Check_Temperature (double temperature)
{
   if(termo.program == PR1)
   {
       if(temperature < (temp1 - hist))
       {
           Heater = 1; //wlacz grzanie
           if(termo.dpr1p)
           {
               LCD_DisplayString(20,70,"Prog#1",&Font20,COL6,WHITE);
               LCD_DisplayString(108,70,"ON ",&Font20,COL6,BLACK);
               termo.dpr1p = 0;
               termo.dpr1np = 1;
           }

       }
       if(temperature > temp1)
       {
           Heater = 0;
       if(termo.dpr1np)
           {
               LCD_DisplayString(20,70,"Prog#1",&Font20,COL6,WHITE);
               LCD_DisplayString(108,70,"OFF",&Font20,COL6,GREEN);
               termo.dpr1np = 0;
               termo.dpr1p = 1;
           }
       }
   }
   if (termo.program == PR2)
   {
        if(temperature < (temp2 - hist))
        {
           Heater = 1; //wlacz grzanie
         if(termo.dpr2p)
           {
               LCD_DisplayString(20,70,"Prog#2",&Font20,COL6,WHITE);
               LCD_DisplayString(108,70,"ON ",&Font20,COL6,BLACK);
               termo.dpr2p = 0;
               termo.dpr2np = 1;
           }
        }
       if(temperature > temp2)
       {
           Heater = 0;
        if(termo.dpr2np)
           {
               LCD_DisplayString(20,70,"Prog#2",&Font20,COL6,WHITE);
               LCD_DisplayString(108,70,"OFF",&Font20,COL6,GREEN);
               termo.dpr2np = 0;
               termo.dpr2p = 1;
           }
       }

   }

}
```

Listing 8. Funkcja termostatu

co z kolei wymaga wydajnego mikrokontrolera oraz – oczywiście – szybkiego interfejsu SPI. W naszym przypadku zegar taktujący przesyłaniem bitów ma częstotliwość 5 MHz przy taktowaniu mikrokontrolera częstotliwością 20 MHz. Konfiguracja interfejsu SPI została pokazana na **rysunku 9**.

Procedury przesłania danych i komend wymagają również ustawienia linii CD wyboru interfejsu SPI oraz linii wyboru rodzaju przesyłanych danych DC (data/command). Na **listingu 3** pokazano procedurę przesyłającą 2 bajty dotyczące jednego piksela do pamięci obrazu wyświetlacza. MMC wygenerowała procedurę wysyłania jednego bajtu przez interfejs SPI – **listing 4**.

Przy dużej ilości przesyłanych danych zdarza się, że procedura oczekiwania na wysłanie bajtów ulega zawieszeniu, choć teoretycznie przy zastosowaniu SPI nie powinna. Drobna modyfikacja powoduje, że problem ten praktycznie znika. W pętli oczekiwania inkrementowany jest licznik. Jeżeli nie zostanie sprzętowo ustawiona flaga SPI_RXCIF oznaczająca, że bajt został wysłany (i odebrany), to wykonywana jest ponowna inicjalizacja modułu SPI SPI0_Initialize () i pętla kończy działanie – **listing 5**.

Ostatni moduł sprzętowy używany przez program to timer TCB (**rysunek 10**). Został on skonfigurowany tak, by zgłaszać cykliczne przerwania co jedną milisekundę. W procedurze obsługi tego przerwania jest realizowana obsługa impulsatora.

Główne procedury regulacji nie są zbyt skomplikowane. Po każdym pomiarze temperatury – wykonywanym co sekundę – wywołaniu ulega procedura Termostat, której argumentem jest temperatura w formacie zmiennoprzecinkowym – **listing 6**.

Funkcja HMI_GetTermostat() odczytuje z pamięci EEPROM wszystkie ustawienia termostatu: histerezę, temperatury programów temp1 i temp2 oraz oba czasy. Następnie wywoływana jest funkcja Termostat_Check_Time () – **listing 7**. Ma ona za zadanie porównać bieżący czas (odczytany z rejestrów zegara RTC) z nastawami Czas1 i Czas2, a na podstawie wyniku tego porównania – określić, który program czasowy ma się wykonywać.

Na podstawie wartości zmiennej termo.program funkcja Termostat_Check_Temperature (temperature) identyfikuje, która z temperatur ma być użyta w procesie regulacji. Procedura ta została pokazana na **listingu 8**. Opisana funkcja załącza lub wyłącza przekaźnik sterujący piecem, oraz wyświetla na ekranie głównym informację o aktywnym programie, a także wskazuje, czy grzanie jest włączone, czy wyłączone.

Program termostatu zajmuje 69% z 48 kB dostępnej pamięci Flash, zaś zdecydowana większość tej przestrzeni przypada na procedury związane z obsługą graficznego wyświetlacza, w tym także wzorce znaków alfanumerycznych. Również podczas pisania programu oprogramowanie interfejsu użytkownika HMI okazało się najbardziej pracochłonne. Można zadać sobie pytanie, czy warto poświęcać tyle czasu na bardziej lub mniej atrakcyjny interfejs użytkownika. Skoro jednak mamy dzisiaj możliwości używania tanich, dobrej jakości wyświetlaczy, wydajnych małych mikrokontrolerów, czy dobrych bezpłatnych narzędzi programistycznych, szkoda byłoby tego nie wykorzystać.

Tomasz Jabłoński, EP