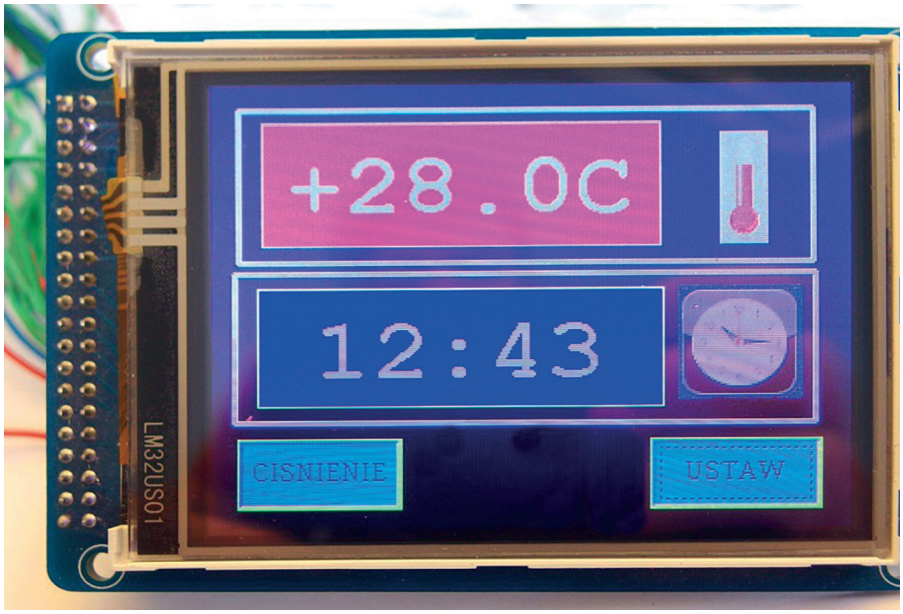
Biblioteka graficzna Microchip'a – przykładowa aplikacja

W poprzednich artykułach opisałem bezpłatną bibliotekę graficzna firmy Microchip, oraz integrację z wyświetlaczem LCD z wbudowanym sterownikiem SSD1289. Teraz postaram się pokazać jak w praktyce można zaprojektować i wykonać graficzny interfejs użytkownika wykorzystując bibliotekę i wspomniany wyświetlacz ze sterownikiem SSD1289.



Przyjąłem, że będzie to termometr z zegarem czasu rzeczywistego. Pomiar temperatury będzie wykonywał termometr DS18B20, a pomiar czasu moduł RTCC wbudowany w mikrokontroler. Platforma sprzętowa to moduł PI32 USB Starter Kit II połączony z płytka rozszerzeń Starter Kit I/O Expansion Board. Połączenia pomiędzy wyprowadzeniami płytki a wyświetlaczem i termometrem DS18B20 są wykonane przewodami z końcówkami do łączenia z wyprowadzeniami goldpinów.

Projekt

Pracę na projektem zaczniemy od sprecyzowania, co będzie zwierał ekran główny. Ponieważ ma to być zegar z termometrem, to zostaną tam umieszone pola z temperaturą i czasem. Pomiar temperatury w zasadzie nie wymaga niczego więcej poza odczytaniem wartości z DS18B20 i jej wyświetleniem na ekranie wyświetlacza, natomiast zegar musi mieć możliwość ustawienia minut i godzin. Dlatego trzeba zaprojektować i wykonać ekran z możliwością nastawiania zegara oraz elementy pozwalające przechodzić do tego ekranu i powracać do ekranu głównego po wprowadzeniu nastaw.

W trakcie prac nad projektem dodałem jeszcze jeden ekran z wyświetlaniem pomiaru ciśnienia atmosferycznego. Ciśnienie w projekcie nie jest mierzone, bo nie miałem czujnika odpowiedniego do prób, ale po uzupełnieniu aplikacji oraz napisanie procedur pomiarowych można w stosunkowo prosty sposób dodać taką funkcję.

Program sterujący zostanie napisany w środowisku MPLAB X IDE i skompilowany kompilatorem MPLAB X C32. Do projektu interesu graficznego wykorzystamy wtyczkę GDDX.

Wyświetlanie temperatury

Temperatura odczytywana z DS18B20 ma postać liczby ze znakiem zapisanej w kodzie U2. Do wyświetlania wartości liczbowych w GDDX jest przeznaczony obiekt *Digital Meter*. Wydaje się, że naturalnym zastosowaniem tego elementu będzie wyświetlanie wartości w rodzaju temperatury i dlatego użyłem go w pierwszej wersji projektu. Jednak było z tym trochę kłopotu.

Temperatura powinna być wyświetlana ze znakiem plus lub minus, a z prawej strony powinno być miano pomiaru. Oczywiście, można bez problemu wykorzystać obiekt *Static Text* i uzupełnić brakujące elementy pola wyświetlanej temperatury. Kłopot polegał na tym, że w czasie przesuwania pola pomiaru temperatury po to, aby znaleźć optymalne położenie, trzeba było przesuwać trzy niezależne elementy i po przesunięciu dokładnie zgrać je ze sobą. W trakcie prób przyszedł mi do głowy pewien pomysł. Ponieważ temperatura ma być wyświetlana w postaci znaków ASCII, to może lepszym rozwiązaniem byłoby konwertowanie wyniku pomiaru na postać łańcucha znakowego, na przykład „+24.5C" i wyświetlanie wyniku pomiaru w takiej postaci. Mamy wtedy kompleksowo załatwioną sprawę wyświetlania znaku i miana pomiaru. Można te elementy wpisać do bufora znakowego na etapie konwersji, w procedurze odczytu z termometru. Wykonanie takiej konwersji jest bardzo łatwe, ale jak potem wyświetlić wynik pomiaru?

Do wyświetlania tekstu na ekranie są przeznaczone 2 elementy: *Text* z warstwy *Primitive Layer* i obiekt (widżet) *Static Text*. Pierwszy z nich na pewno nie może być używany do wyświetlania w tym samym miejscu zmieniającego się tekstu. Pozostaje *Static Text*. Nazwa tego obiektu jest trochę myląca, bo sugeruje, że można wyświetlić tekst, którego nie można zmienić. W rzeczywistości wyświetlany tekst jest powiązany przez wskaźnik do bufora w pamięci RAM lub Flash. Wystarczy zmieniać zawartość bufora znakowego i zlecić funkcjom biblioteki ponowne narysowanie obiektu, by w taki sposób zmieniać informację wyświetlaną przez obiekt *Static Text*.

Zmianie mogą ulegać też elementy struktury stylu obiektu. W naszym przypadku można zmieniać na przykład kolory tła *CommonBkColor* na przykład w zależności od mierzonej temperatury. Dla zbyt niskiej może to być kolor niebieski, dla komfortowej żółty lub zielony, a dla zbyt wysokiej czerwony.

Bufor z tekstem wartości temperatury jest zapisywany przez procedury odczytu temperatury. Oprócz wielkości liczbowej jest tam też wpisywany na początku znak „+" lub „–", a na końcu litera „C". Na **listingu 1** pokazano procedurę odczytującą i konwertującą temperaturę oraz zapisującą ją do bufora *temp[]*. Na końcu tej procedury jest wyliczana wartość temperatury*10. Na przykład dla „22,5" będzie to „225". Zawartość zmiennej *temper* można wykorzystać do wykrywania,

czy wartość temperatury uległa zmianie od ostatniego pomiaru i wyświetlać tylko jej zmiany.

Pracę nad projektem w GDDX rozpoczynamy od postawienia na ekranie początkowym obiektu *Static Text*. GDDX używa domyślnie do wyświetlania tekstu czcionki *GENTIUM_16*. Dal naszych potrzeb ta czcionka jest zbyt mała i trzeba ją zmienić na większą, aby mierzoną temperaturę można było odczytać z większej odległości. Zdefiniowałem nowy styl obiektu z czcionką *Monospace Bold 48*. Jest to pogrubiona czcionka o wysokości 48 pikseli, bardzo dobrze widoczna na ekranie wyświetlacza z większej odległości. Domyślne tło (*CommonBkColor*) ma kolor czerwony, a czcionka (*Color1*) jest biała (**rysunek 1**).

Ponieważ dysponujemy kolorowym wyświetlaczem graficznym i możemy dzięki temu wyświetlić dowolny obraz, to po prawej stronie obok wyniku pomiaru zostanie umieszczona ikonka z symbolem termometru. Ikonka jest umieszczana na ekranie jako obiekt *Picture* (kolorowa bitmapa). Ikonki można znaleźć np. w Internecie lub wykonać je własnoręcznie. Dla naszych celów zmieniłem wielkość gotowego elementu do 19×59 pikseli za pomocą programu *Picture Manager* wchodzącego w skład pakietu *Office*. Obiekt *Static Text* oraz ikona zostały obramowane elementem *Rectangle* z warstwy *Primitive Layer*. Moim Pozwala to na pogrupowanie na ekranie powiązanych ze sobą elementów.

Modyfikacja wyświetlanej temperatury będzie polegała na zapisaniu bufora *temp*, a potem przypisaniu wartości tego bufora do elementu *Static Text*. Biblioteka wykorzystuje do tego celu funkcję *StSetText* z dwoma argumentami. Pierwszy jest wskaźnikiem do modyfikowanego obiektu, a drugi wskaźnikiem do bufora z ciągiem wyświetlanych znaków. Jak widać, do jednego obiektu *StaticText* można „podłączyć" różne bufory, ale też jeden bufor można „podłączać" do różnych obiektów *Static Text*. Samo przypisanie bufora nie spowoduje zmian na ekranie – trzeba jeszcze zlecić funkcjom bibliotecznym ponowne narysowanie obiektu. W przypadku obiektu *Digital Meter* jest możliwe narysowanie tylko samej wartości tekstowej (*DM_UPDATE*). *Static Text* może być również przerysowany cały lub modyfikacji podlega tylko wyświetlany tekst. Ma to swoje konsekwencje, przynajmniej w naszej konfiguracji sprzętowej. Każde przerysowanie powoduje widoczne miganie obiektu i dlatego trzeba się starać, by było to wykonywane tylko wtedy, gdy jest niezbędne. Lepszym rozwiązaniem jest modyfikacja wyświetlania tylko tekstu.

Rysunek 1. Pole pomiaru temperatury w projekcie GDDX z domyślnymi ustawieniami

W tej aplikacji przerysowanie jest wykonywane tylko w czasie, kiedy odczytywane temperatura zmieni swoją wartość. Zlecenie ponownego rysowania obiektu *Static Text* wykonuje funkcja *StDraw* z argumentem, którym jest wskaźnik do rysowanego obiektu. Zlecenie przerysowania tylko tekstu obiektu wykonuje funkcja *Set State* również z argumentem, którym jest wskaźnik do rysowanego obiektu i drugim argumentem (*ST_UPDATE*) wskazującym, że przerysowanie dotyczy tylko obszaru tekstu, na przykład *SetState(((STATICTEXT*) (GOLFindObject(STE_12))), ST_UPDATE);* Na **listingu 2** pokazano procedurę wywoływaną cyklicznie w pętli głównej i realizującą:

- Odczytywanie temperatury z DS18B20 i konwersję do ciągu znaków ASCII, wartości liczbowej i mnożenie t*10 (zmienna temper).
- Porównywanie poprzednio odczytanej wartości temperatury z bieżącą.
- Wyświetlanie wartości temperatury, jeśli wynik porównania jest różny od zera.

Możemy w prosty sposób uatrakcyjnić wyświetlanie temperatury przez zmianę koloru *Static Text* zależnie od mierzonej wartości. Przy każdej zmianie jest testowana wartość zmiennej *temper* odpowiadająca temperaturze *10.

Listing 1. Pomiar i konwersja temperatury
```
void DSReadTemp(void){
    unsigned int tmp,tmp1,Temp;
    Temp=ReadDS();//odczytaj temperaturę z DS18B20
    temp[0]=0;
    if((!Temp&0x0100)==0)  //detekcja znaku
        temp[0]='+';
    else
        temp[0]='-';
    tmp=KonwTemp(Temp);   konwersja z formatu U2
    tmp1=KonwDec(tmp);//konwersja na 2 znaki ascii
    temp[1]=tmp1>>8;     //dziesiątki temperatury
    temp[2]=tmp1;//jednostki temperatury
    temp[3]='.';
    if((tmp&0xf00)>0)
        temp[4]='5';           //część ułamkowa
    else
        temp[4]='0';
    temp[5]='C';
    temp[6]=0;
    temper=((temp[1]-0x30)*100)+((temp[2]-0x30)*10)+(temp[4]-0x30); //wartość  dziesiętna
}
```

Listing 2. Pomiar i wyświetlanie temperatury
```
DSInit();
DSReadTemp();//odczytanie temperatury
if(TMP!=temper)//czy pomiar jest różny od ostatnio odczytanego
{
    TMP=temper;//nowa wartość do porównania
//przypisanie bufora temp do Static Text STE_11 STE_11
    StSetText((STATICTEXT*)(GOLFindObject(STE_11)),temp);
    SetState(((STATICTEXT*)(GOLFindObject(STE_11))), ST_UPDATE);
//lub alternatywnie
//StDraw((STATICTEXT*)(GOLFindObject(STE_11)));//przerysowanie STE_11
}
```

Listing 3. Struktura stylu _static
```
if(_static != NULL) free(_static);
    _static = GOLCreateScheme();
    _static->Color0 = RGBConvert(32, 168, 224);
    _static->Color1 = RGBConvert(248, 252, 248);
    _static->TextColor0 = RGBConvert(248, 252, 248);
    _static->TextColor1 = RGBConvert(248, 252, 48);
    _static->EmbossDkColor = RGBConvert(248, 204, 0);
    _static->EmbossLtColor = RGBConvert(24, 116, 184);
    _static->TextColorDisabled = RGBConvert(128, 128, 128);
    _static->ColorDisabled = RGBConvert(208, 224, 240);
    _static->CommonBkColor = RGBConvert(200, 0, 0);
    _static->pFont = (void*)&Monospaced_bold_Bold_48_1;
```

Listing 4. Procedura wyświetlania temperatury z modyfikacją kolorów
```
DSInit();
DSReadTemp();
if(TMP!=temper)
{
    TMP=temper;
    if(temper<180)
    _static->CommonBkColor =BLUE;
    if(temper>=180&&temper<=270) _static->CommonBkColor =GREEN;
    if(temper>270) _static->CommonBkColor =RED;
    StSetText((STATICTEXT*)(GOLFindObject(STE_11)),temp);
    StDraw((STATICTEXT*)(GOLFindObject(STE_11)));
}
```

Temperaturę podzielono na 3 progi:

- Poniżej 18°C – kolor niebeski.
- W zakresie 18…27°C – kolor zielony.
- Powyżej 27°C – kolor czerwony.

Kolory zmieniamy przez zapisywanie składowej *CommonBkColor* struktury stylu *_static* przypisanego do obiektu *Static Text STE_11*. Styl jest definiowany przez GDDX, co pokazano na **listingu 3**.

Na **listingu 4** pokazano zmodyfikowaną wersję procedury wyświetlania temperatury z modyfikacją koloru *Static Text*. Tu jednak użyliśmy funkcji *StDraw*, bo musimy przerysować cały obiekt, a nie tylko pole tekstowe. W ten sposób można modyfikować wszystkie kolory struktury stylu, a także czcionkę. To ostatnie pod warunkiem, że nowa czcionka jest wcześniej zdefiniowana przez GDDX. Modyfikacje są wyświetlane po wywołaniu funkcji *StDraw*. Biblioteka graficzna GDD w opisywanej wersji wykrywa zdarzenia na zasadzie odpytywania (poolingu). Odbywa się to w pętli głównej w pliku main.c – **listing 5**. Umieszczenie funkcji związanych z przerysowaniem obiektów, mimo że działa, to nie jest do końca prawidłowe. Wszelkie zlecenia przerysowywania obiektów powinny być umieszczone albo w funkcji wykrywania zdarzeń *GDDDemoGOLMsgCallback*, albo w funkcji *GOLDrawCallback*. W programie docelowym przeniesiono tam procedurę wyświetlania temperatury, co zostanie omówione później.

Jeżeli chcemy, aby termometr mierzył temperaturę i wyświetlał ją na bieżąco, to trzeba w tej pętli wywoływać procedurę *DSReadTemp*. I tu pojawia się spory problem. Emulacja programowa magistrali 1-Wire i odczytywanie temperatury powoduje blokowanie programu na około 1 sekundę. Potem są wykonywane czynności związane z testowaniem obsługi panelu dotykowego, sprawdzenie czy trzeba coś narysować i ponownie pomiar blokuje mikrokontroler na 1 sekundę. Czynności wykonywane poza pomiarami zajmują zazwyczaj o wiele mniej czasu niż sam pomiar.

To „wstrzymanie pracy" jest chyba najgorsze w układach z poolingiem. Powoduje ono, że praktycznie nie można wykryć zdarzenia z panelu dotykowego wyświetlacza i przejść do następnego ekranu. Mało tego. Gdy już to się uda, to wykorzystanie jakiegokolwiek obiektu współpracującego z panelem dotykowym (*buton*, *round dial*, *radio buton* itp.) nie jest możliwe. Rozwiązaniem mogłoby być zastosowanie systemu RTOS, ale w naszym projekcie z zasady tego nie przewidujemy. Jeżeli chcemy poruszać się dalej w obrębie przyjętych rozwiązań sprzętowo-programowych, to można mierzyć temperaturę na przykład co kilka minut. Dla pomiarów temperatury otoczenia to wystarczy. Dużo lepszym rozwiązaniem byłoby zastosowanie czujnika umożliwiającego szybkie odczytywanie statusu bez czekania na zakończenie pomiaru. Po stwierdzeniu zakończenia pomiaru można szybko odczytać jego wartość poprzez szeregową magistralę SPI lub I²C, zależnie od zastosowanego typu czujnika temperatury.

Ponieważ rozpatrujemy tutaj przykład działającej aplikacji z wykorzystaniem biblioteki graficznej, to zastosowałem pomiar temperatury wykonywany z interwałem 5 minut. Program jest blokowany przez 1 sekundę raz na 300 sekund i można przyjąć, że nie stanowi to problemu.

Listing 5. Pętla główna programu

```
while(1)
{
    if(GOLDraw())  // wykrycie konieczności rysowania obiektów i rysowanie obiektów
    {
        TouchGetMsg(&msg);  // wykrycie zdarzenia generowanego przez panel dotykowy
        GOLMsg(&msg);  // obsługa zdarzenia
    }
    DSInit();  //pomiar temperatury
    DSReadTemp();
    if(TMP!=temper)
    {
        TMP=temper;
        if(temper<180)
        {
            _static->CommonBkColor =BLUE;
            _static->TextColor0 =WHITE;
        }
        if(temper>=180&&temper<=240)
        {
            _static->CommonBkColor =GREEN;
            _static->TextColor0 =WHITE;
        }
        if(temper>=240&&temper<=270)
        {
            _static->CommonBkColor =ORANGE;
            _static->TextColor0 =RED;
        }
        if(temper>270)
        {
             _static->CommonBkColor =RED;
             _static->TextColor0 =WHITE;
        }
        StSetText((STATICTEXT*)(GOLFindObject(STE_11)),temp);
        StDraw((STATICTEXT*)(GOLFindObject(STE_11)));
    }
}//end while
```

Zegar czasu rzeczywistego – pomiar i wyświetlanie czasu

Do wyświetlania czasu wykorzystamy również obiekt *Static Text* z czcionką *Monospaced Bold 48* (**rysunek 2**). Obiekt jest powiązany z buforem znakowym *timer*. W odróżnieniu od termometru, zegar czasu rzeczywistego musi mieć możliwość ustawiania czasu. Do tego celu został zaprojektowany osobny ekran (**rysunek 3**). W skład tego ekranu wchodzą obiekty:

- *Static Text* z ustawianym czasem,
- *Static Text* z informacją, co jest w danej chwili ustawiane,
- *Round Dial* do zmiany wartości,
- przyciski ACC i Return.

W oknie „Ustaw czas" jest wyświetlana bieżąca wartość ustawianego czasu. Po wyświetleniu ekranu jest domyślnie włączone ustawianie minut – informacja o tym jest pokazywana w zielonym oknie. Zmiana wartości minut jest wykonywana przez „kręcenie" obiektem *Round Dial*. Po ustawieniu minut naciskamy przycisk *Acc* i program przechodzi do ustawiania godzin – informacja o tym jest wyświetlana w zielonym oknie. Kolejne przyciśnięcie *Acc* powoduje powrót do ustawiania minut. Ustawienie zegara i zakończenie wyświetlania ekranu ustawiania czasu następuje po przyciśnięciu przycisku *Return*.

Przejście do ekranu ustawiania czasu jest wykonywane po naciśnięciu przycisku *Ustaw* umieszczonego na dole ekranu głównego (wyświetlającego temperaturę i czas). Biblioteka z każdym elementem manipulacyjnym wiąże zdarzenie generowane w momencie wykrycia przyciśnięcia panelu dotykowego obszaru zajmowanego przez ten

Rysunek 2. Pole wyświetlania czasu

Rysunek 3. Ekran ustawiania zegara

element. Programowe wykrywanie tego zdarzenia może być generowane przez GDDX, jeżeli w oknie *Events/Project Explorer* wybierzemy element, a następnie akcję – na przykład przyciśnięciu przycisku *BTN_6* są przypisane 4 akcje:

Listing 6. Kod wykonywany po naciśnięciu „USTAW"
```
// <START_ID_BTN_MSG_PRESSED_(BTN_6)>
if(objMsg == BTN_MSG_PRESSED && pObj->ID == (BTN_6))
{
    GDDDemoGoToScreen(1);
    ekran=1;
}
```

Listing 7. Obsługa zdarzenia obrotu w kierunku wskazówek zegara obiektu Round Dial
```
if(objMsg == RD_MSG_CLOCKWISE && pObj->ID == (RDI_6) )
{
    if(set.set_min==1)//ustawianie minut
    {
        ++set.min;
        if(set.min==60) set.min=0;
        val=KonwDec(set.min);//konwersja na znaki ASCII
        timer[3]=val>>8;
        timer[4]=val;
        //wyświetlanie ustawionej wartości
        StSetText((STATICTEXT*)(GOLFindObject(STE_12)),timer);
        SetState(((STATICTEXT*)(GOLFindObject(STE_12))), ST_UPDATE);
//Lub alternatywnie
       //StDraw((STATICTEXT*)(GOLFindObject(STE_12)));
    }
    if(set.set_godz==1)//ustawianie godzin
    {
        ++set.godz;
        if(set.godz==24) set.godz=0;
        val=KonwDec(set.godz);  //konwersja na znaki ASCII
        timer[0]=val>>8;
        timer[1]=val;
//wysietlanie ustawionej wartości
        StSetText((STATICTEXT*)(GOLFindObject(STE_12)),timer);
        SetState(((STATICTEXT*)(GOLFindObject(STE_12))), ST_UPDATE);
//Lub alternatywnie
        //  StDraw((STATICTEXT*)(GOLFindObject(STE_12)));
    }
}
// <END_ID_RD_MSG_CLOCKWISE_(RDI_6)>
```

- BTN_MSG_PRESSED – przycisk został przyciśnięty,
- BTN_MSG_RELEASED – przycisk został zwolniony,
- BTN_MSG_STILLPRESSED – przycisk jest nadal przyciśnięty,
- BTN_MSG_CANCELPRESS – przyciśnięcie anulowane.

Każdemu zdarzeniu wtyczka GDDX może przypisać akcję związaną z innymi elementami projektu, na przykład przejście do innego ekranu – opisywałem to w poprzednich częściach cyklu poświęconego bibliotece. Można też – poprzez wprowadzenie polecenia *Add code* – dodać puste wywołania detekcji zdarzenia. Dla przycisku *USTAW (BTN_6)* i zdarzenia *BTN_MSG_PRESSED* przypisałem w GDDX akcję: przejdź do ekranu 1 (*Goto Screen(1)*). Po wygenerowaniu kodu otrzymałem w pliku *GDD_X_Event_Handler.c* fragment pokazany na **listingu 6**.

Po przejściu do ekranu pierwszego można używać wszystkich elementów tam umieszczonych i programowo wykrywać oraz obsługiwać zdarzenia od przyciśnięcia panelu dotykowego. Jednym z głównych obiektów tego ekranu jest *Round Dial*.

Round Dial jest wirtualnym odpowiednikiem impulsatora (enkodera obrotowego) z gałką i może generować dwa zdarzenia: obrót zgodny z kierunkiem obrotu wskazówek zegara

Rysunek 4. Ekran w trakcie ustawiania zegara

i obrót przeciwny z ruchem wskazówek zegara. Dla każdego z tych zdarzeń można modyfikować liczniki minut i godzin w trakcie procesu nastawiania zegara. Na **listingu 7** pokazano obsługę wykrycia zdarzenia ruchu zgodnego z kierunkiem ruchu wskazówek zegara.

O tym, co jest aktualnie ustawiane – minuty czy godziny – decydują składowe *set.set_min* i *set.set_godz* struktury *set*. Po restarcie mikrokontrolera domyślnie jest ustawione *set.set_min=1* i *set.set_godz=0* (ustawianie minut). Każde wykrycie zdarzenia obrotu powoduje zwiększenie licznika ustawianej wielkości, potem jest wykonywana konwersja modyfikowanej wartości na dwa znaki ASCII w buforze *timer*. Wyświetlanie zawartości bufora jest wykonywane przez dwie już opisywane funkcje: *StSetText* i *StDraw*. Analogicznie jest obsługiwane zdarzenie wykrycia obrotu w kierunku przeciwnym do ruchu wskazówek zegara, co pokazano na **listingu 8**.

Na **rysunku 4** pokazano ekran wyświetlacza w trakcie ustawiania zegara. Ustawiona wartość musi być zapisana do rejestrów modułu RTTC, ale jednocześnie musi się wyświetlić na ekranie głównym po przyciśnięciu *Return*. W trakcie ustawiania czasu zawartość bufora *timer* jest przypisana i wyświetlana przez obiekt *Static Text* STE_12 umieszczony na ekranie ustawiania. Po przejściu do ekranu głównego bufor *timer* musi zostać przypisany do obiektu *Static Text STE_22*.

W trakcie prac nad tym projektem okazało się, że nie można w dowolnym momencie przypisać bufora do obiektu i wywołać funkcję rysowania *StDraw*. Jeżeli program obsługuje ekran ustawień, to wywołanie *StDraw((STATICTEXT*)(GOLFindObject(STE_22)))* dla obiektu *Static Text STE_22* w funkcji *GOLDrawCallback()* powoduje błąd i w najlepszym wypadku mikrokontroler się wyzeruje, bo *STE_22* jest umieszczona na ekranie głównym. Program musi „wiedzieć" czy obiekt może być przerysowany ponownie. Do identyfikacji ekranów wprowadziłem zmienną *ekran* inicjowaną na wartość 0. Kiedy program wykonuje polecenie przejścia do ekranu zerowego *GDDDemoGoToScreen(0)*, to zmienna ekran jest zerowana (*ekran=0;*). Po przejściu do ekranu ustawiania czasu *GDDDemoGoToScreen(1);* do zmiennej *ekran* jest wpisywana 1. Na **listingu 9** pokazano funkcję *GOLDrawCallback()*, która przypisuje i wyświetla zawartość bufora *timer* do obiektu *STE_22*, lub *STE_12* zależnie od zawartości zmiennej *ekran* oraz sprawdza czy poprzedni pomiar temperatury różni się od bieżącego. Jeżeli tak, to zawartość bufora *temp* jest wyświetlana przez obiekt *STE_11*. Jednocześnie jest testowany zakres temperatury i modyfikowany kolor obiektu *STE_11*.

Listing 8. Obsługa zdarzenia obrotu w kierunku przeciwnym do ruchu wskazówek zegara obiektu Round Dial

```
if(objMsg == RD_MSG_CTR_CLOCKWISE && pObj->ID == (RDI_6) )
{
        if(set.set_min==1)
        {
            --set.min;
            if(set.min<0) set.min=59;
            val=KonwDec(set.min);
            timer[3]=val>>8;
            timer[4]=val;
            StSetText((STATICTEXT*)(GOLFindObject(STE_12)),timer);
            SetState(((STATICTEXT*)(GOLFindObject(STE_12))), ST_UPDATE);
//Lub alternatywnie
            //StDraw((STATICTEXT*)(GOLFindObject(STE_12)));
        }
        if(set.set_godz==1)
        {
            --set.godz;
            if(set.godz<0) set.godz=23;
            val=KonwDec(set.godz);
            timer[0]=val>>8;
            timer[1]=val;
            StSetText((STATICTEXT*)(GOLFindObject(STE_12)),timer);
            SetState(((STATICTEXT*)(GOLFindObject(STE_12))), ST_UPDATE);
//Lub alternatywnie
            //StDraw((STATICTEXT*)(GOLFindObject(STE_12)));
        }
    }
}
```

Listing 9. Wyświetlanie temperatury i ekranów z czasem

```
WORD GOLDrawCallback(void)
{
    GDDDemoGOLDrawCallback();
//wyświetlanie czasu na ekranie głównym
    if(ekran==0)
    {
        StSetText((STATICTEXT*)(GOLFindObject(STE_22)),timer);  //czas
        StDraw((STATICTEXT*)(GOLFindObject(STE_22)));
        TMP=0;
        ekran=0xff;
    }
//wyświetlanie czasu na ekranie ustawiania czasu
    if(ekran==1)
    {
        StSetText((STATICTEXT*)(GOLFindObject(STE_12)),timer);  //temperatura
        StDraw((STATICTEXT*)(GOLFindObject(STE_12)));
        ekran=0xff;
    }
    if(TMP!=temper)
    {
        TMP=temper;
        if(temper<180)
        {
            _static->CommonBkColor =BLUE;
            _static->TextColor0 =WHITE;
        }
        if(temper>=180&&temper<=240)
        {
            _static->CommonBkColor =GREEN;
            _static->TextColor0 =WHITE;
        }
        if(temper>=240&&temper<=270)
        {
            _static->CommonBkColor =BROWN;
            _static->TextColor0 =WHITE;
        }
        if(temper>270)
        {
            _static->CommonBkColor =RED;
            _static->TextColor0 =WHITE;
        }
        StSetText((STATICTEXT*)(GOLFindObject(STE_11)),temp);
        StDraw((STATICTEXT*)(GOLFindObject(STE_11)));
    }
    return (1);
}
```

Podsumowanie

Z założenia miał to być projekt nie tyle pokazujący jak najwięcej możliwości opisywanego narzędzia, ale będący przykładem jak za pomocą elementów biblioteki można w stosunkowo łatwy i szybki sposób zbudować bardziej lub mniej atrakcyjny interfejs wykorzystujący kolorowy wyświetlacz LCD i panel dotykowy. Program zajmuje ok 24% całej pamięci programu Flash i tylko 520 bajtów pamięci RAM przy kompilacji z wyłączonymi opcjami optymalizacji kodu dla bezpłatnej wersji kompilatora MPLAB XC-32. Można zatem rozwijać go bez obawy o to, że szybko zabraknie pamięci dla głównych procedur urządzenia.

Tomasz Jabłoński, EP