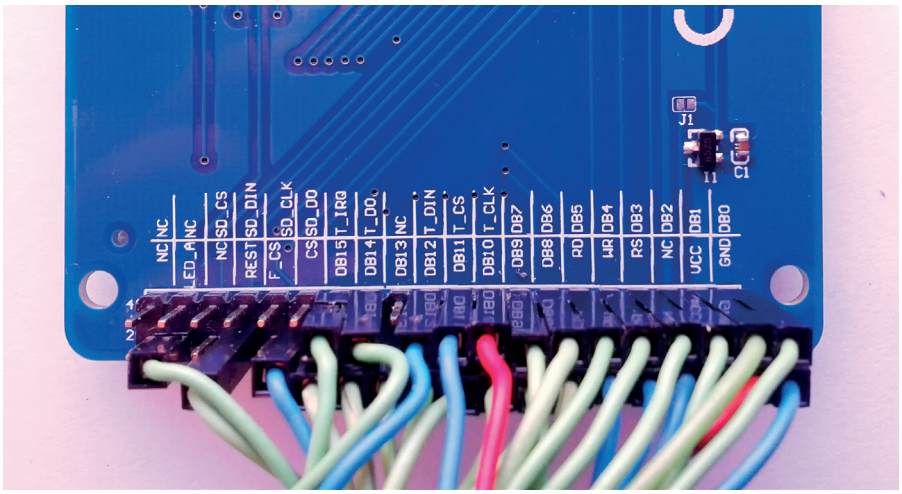
Użycie biblioteki graficznej Microchipa z wyświetlaczem LCD ze sterownikiem SSD1289

Na łamach EP3/2014 opisałem możliwości bezpłatnej biblioteki graficznej firmy Microchip na podstawie przykładów uruchamianych na firmowym module ewaluacyjnym Multimedia Expansion Board (MEB). Microchip przygotował dla tego modułu kompletną konfigurację wszystkich niezbędnych modułów sprzętowych mikrokontrolera oraz definicję linii portów niezbędną do prawidłowego sterowania wbudowanego modułu TFT i panelu dotykowego. Wykorzystanie modułu umożliwiło mi skupienie się na możliwościach samej biblioteki bez konieczności zajmowania się problemami sprzętu. To bardzo komfortowa sytuacja, ale po zapoznaniu się z biblioteką prędzej czy później przyjdzie moment, kiedy zechcemy wykorzystać na przykład wtyczkę Graphic Designer Display X do zaprojektowania i wykonania interfejsu graficznego we własnym urządzeniu.

Aby zastosować wyświetlacz kolorowy TFT, trzeba go sterować za pomocą przeznaczonego do tego celu sterownika lub mikrokontrolerem z odpowiednimi układami peryferyjnymi. Jak wiemy, biblioteka graficzna wspiera sporą liczbę sterowników i może być stosowana z jednym mikrokontrolerów rodziny PIC24 lub PIC32 w roli sterownika wyświetlacza TFT. Ponieważ w module MEB jest używany sterownik SSD1926, to na początku najbardziej naturalnym wydaje się użycie wyświetlacza TFT z takim sterownikiem. Niestety, po sprawdzeniu oferty krajowych dystrybutorów nie znalazłem takiego wyświetlacza. Sięgnąłem po raz kolejny do dokumentacji biblioteki i znalazłem informację, że możliwe jest jej proste skonfigurowanie do pracy z alternatywnym sterownikiem S1D13517 firmy Epson. I tu również poszukiwania nie dały pozytywnego wyniku. Pozostało znalezienie wyświetlacza TFT ze sterownikiem z listy wspieranych przez bibliotekę lub w ostateczności z dowolnym sterownikiem i napisanie własnych procedur warstwy *Device Driver Layer*. To ostatnie, przy pewnym doświadczeniu, nie powinno być specjalnie trudne.



Fotografia 1. Wyprowadzenia wyświetlacza

W trakcie dalszych poszukiwań kupiłem na aukcji wyświetlacz TFT o rozdzielczości 240×320 pikseli, z układem SSD1289 będącym na liście wspieranych sterowników. Dodatkowo, wyświetlacz był wyposażony w podświetlenie LED, rezystancyjny panel dotykowy ze sterownikiem XPT2046 oraz złącze dla karty SD. Jeżeli dodamy do tego cenę ok. 70 złotych, to wydaje się on idealny do stosowania we własnych urządzeniach. Niska cena być może będzie skutkowała gorszymi parametrami matrycy, ale to nie jest przesądzone.

Sterowanie wyświetlaczem – podstawowe funkcje warstwy *Device Driver Layer*

Biblioteka graficzna jest przeznaczona do używania z mikrokontrolerami rodzin PIC24 i PIC32. Do testów z wyświetlaczem użyłem modułu *PIC32 USB Stater kit II* z mikrokontrolerem PIC32MX795F512L. Jest to dokładnie taki sam moduł, jaki sterował wyświetlaczem w module *Multimedia Expansion Board*. Do połączenia modułu z wyświetlaczem jest potrzebna specjalna płytka-przejściówka *Starter Kit I/O Expansion Board*. Oczywiście, taki zestaw nie jest niezbędny. Wyprowadzenia wyświetlacza można połączyć z płytką mikrokontrolera w dowolny sposób.

Na płytce wyświetlacza jest zamontowane złącze szpilkowe IDC40. Każdy z pinów jest opisany (**fotografia 1**), więc wykonanie połączeń nie powinno sprawiać problemu. Sterownik SST1289 jest produkowany przez firmę Solomon Sysytech i przeznaczony do sterownia kolorowymi matrycami TFT o rozdzielczości 320×240 pikseli RGB i 18-bitowej głębi kolorów. Jest wyposażony w pamięć obrazu RAM o pojemności 172800 bajtów (240×320×18/8). Sterownik ma również wbudowaną przetwornicę DC-DC wytwarzającą napięcie do zasilania driverów matrycy LCD. Zapisywanie i odczytywanie pamięci obrazu oraz rejestrów konfiguracyjnych

Listing 1. Modyfikacja pliku HardwareProfile.h

```
#elif defined (__PIC32MX__) || defined (__dsPIC33E__) || defined(__PIC24E__)
#include "Configs/MY_BOARD.h"
```

Listing 2. Plik konfiguracyjny *MY_BOARD.h*

```
#define MY_BOARD//nazwa mojej "platformy" sprzętowej
#define PIC32_USB_SK//definicje właściwe dla użytego mikrokontrolera
#define GFX_USE_DISPLAY_CONTROLLER_SSD1289//użyty sterownik

#ifdef MY_BOARD
    #define MY_TFT_PANEL//mój wyświetlacz
#endif

#if defined (MY_TFT_PANEL)//definicje właściwości wyświetlacza
    #define DISP_ORIENTATION       0
    #define DISP_HOR_RESOLUTION    240
    #define DISP_VER_RESOLUTION    320
    #define DISP_DATA_WIDTH        18
    #define DISP_INV_LSHIFT
    #define DISP_HOR_PULSE_WIDTH   25
    #define DISP_HOR_BACK_PORCH    5
    #define DISP_HOR_FRONT_PORCH   10
    #define DISP_VER_PULSE_WIDTH   4
    #define DISP_VER_BACK_PORCH    0
    #define DISP_VER_FRONT_PORCH   2

#defined (MY_BOARD)
//definicje linii sterujących magistrali równoległej
 #define LCD_CS                 PORTGbits.RG15
 #define LCD_CS_TRIS            TRISGbits.TRISG15
 #define LCD_RS                 PORTGbits.RG14
 #define LCD_RS_TRIS            TRISGbits.TRISG14
 #define LCD_RD                 PORTGbits.RG12
 #define LCD_RD_TRIS            TRISGbits.TRISG12
 #define LCD_WR                 PORTGbits.RG13
 #define LCD_WR_TRIS            TRISGbits.TRISG13
 #define LCD_RES                PORTBbits.RB15
 #define LCD_RES_TRIS           TRISBbits.TRISB15

#endif
```

umożliwiają równoległe interfejsy pracujące w standardzie Motorola 6800 lub Intel 8080 oraz szybki interfejs szeregowy SPI. Interfejs komunikacyjny wybiera się poprzez wymuszenie kombinacji poziomów na wejściach konfiguracyjnych. Producent panelu wyświetlacza może udostępnić te wejścia lub nie. W naszym wypadku interfejs jest ustalony na stałe i nie można go zmienić. Jest to 16-bitowa, równoległa magistrala danych z sygnałami sterującymi w standardzie Intel 8080.

Wiemy już jak musimy komunikować się ze sterownikiem. Wróćmy teraz na chwilę do biblioteki graficznej. Projekt wygenerowany przez GDD X zawiera podkatalog *Drivers* z plikiem *TCON_SSD1289.c*. W opisie biblioteki podano, że dla sterownika SSD1289 plikiem drivera jest *drvTFT2.c*. Na początku te dwie informacje wprowadzają trochę zamieszania. Intuicyjnie chcielibyśmy wykorzystać do sterowania plik o nazwie odpowiadającej nazwie sterownika. Po dokładniejszej analizie obu plików okazuje się, że *TCON_SSD1289.c* zawiera obsługę tego wyświetlacza, lecz za pomocą SPI. Z tego powodu, niestety, nie możemy użyć go do naszych celów.

W pliku *drvTFT2.c* znajdziemy parę niezbędnych procedur, ale nie ma funkcji komunikacji poprzez magistralę równoległą. Microchip ma do sterowania interfejsami równoległymi specjalny moduł peryferyjny PMP (*Parallel Master Port*). Może on pracować z 8- lub 16-bitową szyną danych i jest elastycznie konfigurowany. Ja do celów testowych nie użyłem PMP, a zastosowałem programową emulację standardu Intel 8080 z 16-bitową magistralą danych.

Programowe integrowanie biblioteki z wyświetlaczem zaczniemy od napisania procedur emulujących magistralę równoległą oraz zapisywania oraz odczytywania danych z/do pamięci obrazu i oraz rejestrów sterujących. Procedury te zostaną umieszczone w pliku *drvTFT2.c*. Zanim to zrobimy, trzeba będzie zdefiniować środowisko sprzętowe. Tradycyjnie w projektach Microchipa jest do tego przeznaczony plik *HardwareProfile.h*. W pliku wygenerowanym przez GDDX należy zawrzeć definicje, jak na **listingu 1**.

Listing 3. Zapisanie pamięci danych poprzez magistralę Intel 8080

```
//zapisanie danych
void LcdWriteData(unsigned int data)
{
    LCD_RD=1;Nop();Nop();
    LCD_RS = 1;Nop();Nop();
    LCD_CS = 0; Nop();Nop();
    PORTE=(data>>8);Nop();
    PORTD=data;Nop();Nop();
    LCD_WR = 0;Nop();Nop();
    LCD_WR = 1;Nop();Nop();
    LCD_CS = 1;Nop();Nop();
}
```

Listing 4. Zapisanie rejestru kontrolnego poprzez magistralę Intel 8080

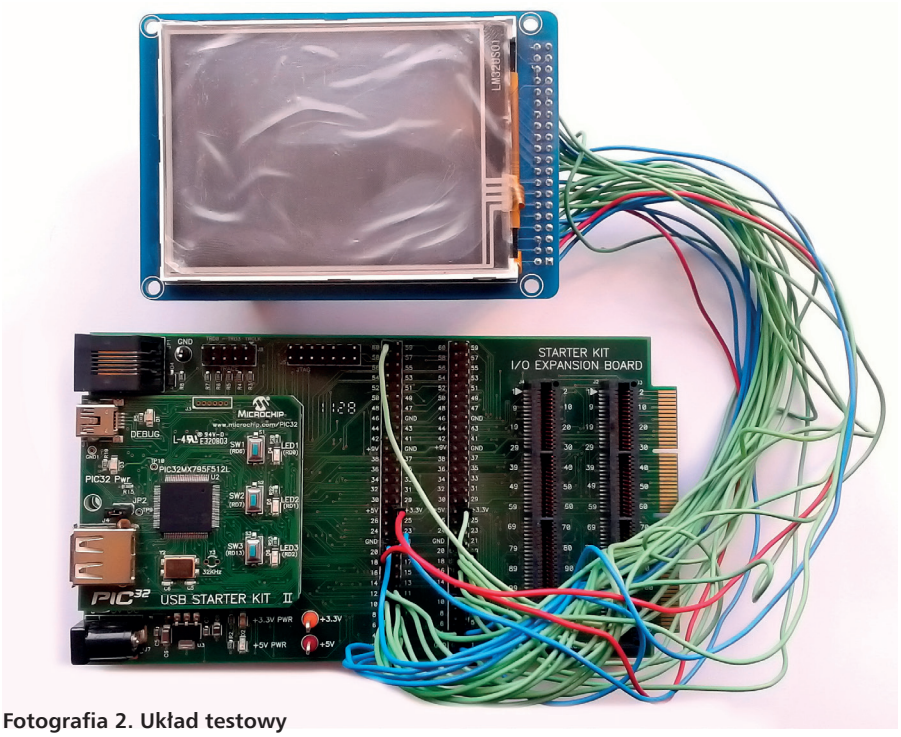
```
void LcdWriteControl(unsigned int data)
{
    LCD_RD=1;Nop();Nop();
    LCD_RS = 0;Nop();Nop();
    LCD_CS = 0;Nop();Nop();
    PORTE=(data>>8);Nop();
    PORTD=data;Nop();Nop();
    LCD_WR = 0;Nop();Nop();
    LCD_WR = 1;Nop();Nop();
    LCD_RS = 1;Nop();Nop();
    LCD_CS = 1;Nop();Nop();
}
```

Listing 5. Odczytanie słowa danych z pamięci obrazu

```
unsigned short LcdReadData(void)
{
    Unsigned short Data;
    LCD_WR=1;Nop();Nop();
    LCD_RS = 0;Nop();Nop();
    LCD_CS = 0;Nop();Nop();
    LCD_RD = 0;Nop();Nop();Nop();Nop();
    Nop();Nop();
    Data=PORTE;
    Data=(data>>8);
    Data=Data|PORTD;
    LCD_RD = 1;Nop();Nop();
    LCD_RS = 1;Nop();Nop();
    LCD_CS = 1;Nop();Nop();
    return(data);
}
```

Listing 6. Zapis rejestrów sterownika

```
void SetReg(WORD index, WORD value)
{
    LcdWriteControl(index);
    LcdWriteData(value);
}
```



Fotografia 2. Układ testowy

Listing 7. Inicjalizacja sterownika SSD1289

```
void ResetDevice(void)
{
    TRISD=0xff00;//RD0…RD7 wyjściowy
    TRISE=0xff00;//RE0…RE7 wyjściowy
    LCD_CS_TRIS=0;  //sterujące linie wyjściowe
    LCD_RS_TRIS=0;
    LCD_RD_TRIS=0;
    LCD_WR_TRIS=0;
    LCD_RES_TRIS=0;
    LCD_RES=1;//cykl zerowania sterownika
    DelayMs(50);
    LCD_RES=0;
    DelayMs(50);
    LCD_RES=1;
    LCD_RS = 0;//stany  początkowe linii sterujących
    LCD_WR = 1;
    LCD_RD = 1;
    LCD_CS = 1;
    SetReg(0x00, 0x0001);
    SetReg(0x03, 0xAAAC);
    SetReg(0x0C, 0x0002);
    DelayMs(15);
    SetReg(0x0D, 0x000A);
    SetReg(0x0E, 0x2D00);
    SetReg(0x1E, 0x00BC);
    SetReg(0x01, 0x1A0C);
    DelayMs(15);
    #if (DISP_ORIENTATION == 0)
        SetReg(0x01, 0x293F);
    #else
        SetReg(0x01, 0x2B3F);
    #endif
    SetReg(0x02, 0x0600);
    SetReg(0x10, 0x0000);
    #if (DISP_ORIENTATION == 0)
        SetReg(0x11, 0x60B0);
    #else
        SetReg(0x11, 0x60B0);
    #endif
    SetReg(0x05, 0x0000);
    SetReg(0x06, 0x0000);
    DelayMs(100);
    SetReg(0x16, 0xEF1C);
    SetReg(0x17, 0x0003);
    SetReg(0x07, 0x0233);
    SetReg(0x0B, 0x0000);
    SetReg(0x0F, 0x0000);
    SetReg(0x41, 0x0000);
    SetReg(0x42, 0x0000);
    SetReg(0x48, 0x0000);
    SetReg(0x49, 0x013F);
    SetReg(0x44, 0xEF00);
    SetReg(0x45, 0x0000);
    SetReg(0x46, 0x013F);
    SetReg(0x4A, 0x0000);
    SetReg(0x4B, 0x0000);
    SetReg(0x30, 0x0707);
    SetReg(0x31, 0x0704);
    SetReg(0x32, 0x0204);
    SetReg(0x33, 0x0502);
    SetReg(0x34, 0x0507);
    SetReg(0x35, 0x0204);
    SetReg(0x36, 0x0204);
    SetReg(0x37, 0x0502);
    SetReg(0x3A, 0x0302);
    SetReg(0x3B, 0x1f00);
    SetReg(0x23, 0x0000);
    SetReg(0x24, 0x0000);
DelayMs(50);
}
```

Definicja sprzętu zostanie umieszczona w pliku *MY_BOARD.h*. Jest to zmodyfikowany do potrzeb testów plik konfiguracyjny pierwotnie przeznaczony dla modułu MEB (**listing 2**). Znajdziemy tu definicje stałych odpowiednich dla użytego sterownika i definicje linii sterujących magistrali równoległej.

Magistrala Intel 8080 składa się z linii danych i linii sterujących (opcjonalnie z linii adresowych). Linie sterujące to *!WR* (zapis) i *!RD* (odczyt). Cykl zapisu danych rozpoczyna się od wyzerowania linii *!WR*, a następnie wystawienia ważnych danych na magistrali. Narastające zbocze sygnału *!WR* zapisuje dane do rejestru sterownika. Interfejs sterujący jest uzupełniony o 2 dodatkowe linie: wyboru interfejsu *CS* i źródła zapisywanych danych *D/!C* (w opisie wyprowadzeń wyświetlacza ta linia nazywa się RS). Jeżeli linia *D/!C* jest ustawiona, to jest zapisywana pamięć danych, jeżeli wyzerowana, to jest zapisywany rejestr (adres rejestru). Na **rysunku 3** pokazano przebiegi czasowe w czasie zapisu danych. Na **listingu 3** pokazano procedurę zapisu danej (zaadresowanej komórki pamięci obrazu), a na **listingu 4** zapisu rejestru sterownika.

Mikrokontroler jest taktowany z częstotliwością 80 MHz, a układy peryferyjne – w tym linie portów – z częstotliwością 40 MHz. Aby zapewnić poprawne zbocza sygnałów sterujących, po każdej zmianie ich poziomu jest wykonywanych klika rozkazów *NOP* (dodatkowe opóźnienie). W czasie zapisu danych linia *!RD* musi być ustawiona. Odczytywanie danych również nie jest skomplikowane. Na **listingu 5** pokazano procedurę odczytu słowa danych z pamięci obrazu.

Każdy sterownik wyświetlacza graficznego wymaga inicjalizacji polegającej na zapisaniu rejestrów konfiguracyjnych. W sterowniku SSD1289 zapisywanie rejestrów polega na wykonaniu cyklu zapisu adresu, a po nim cyklu zapisu 16-bitowej danej (**listing 6**). Kompletny cykl przykładowej inicjalizacji pokazano na **listingu 7**.

Opisywanie tutaj wszystkich rejestrów sterujących nie jest konieczne. Te dane można sobie znaleźć w dokumentacji sterownika i na ich podstawie zobaczyć jak jest wykonywana inicjalizacja. W Internecie można znaleźć sekwencje inicjalizacji SSD1289 różniące się nieco do tej z list. 7. W trakcie testów wypróbowałem kilka alternatywnych i efekt działania był taki sam. Zainicjowany wyświetlacz jest gotowy do wyświetlania zawartości pamięci obrazu.

W dokumentacji biblioteki, w części

Listing 8. Procedura PutPixel

```
void PutPixel(SHORT x, SHORT y)
{
    if(_clipRgn)
    {
        if(x < _clipLeft) return;
        if(x > _clipRight) return;
        if(y < _clipTop) return;
        if(y > _clipBottom) return;
    }
    SetAddress(x, y);
    WritePixel(_color);
}
```

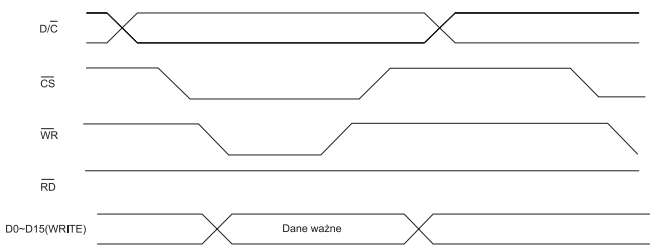
Listing 9. Procedura ustawienia adresu piksela

```
inline void SetAddress(WORD x, WORD y)
{
    #if (DISP_ORIENTATION == 0)
        SetReg(0x004f,x);
    #else
        SetReg(0x004e,x);
    #endif
    #if (DISP_ORIENTATION == 0)
        SetReg(0x004e,y);
    #else
        SetReg(0x004f,y);
    #endif
    LcdWriteControl(0x0022);
}
```

poświęconej dodawaniu nowego sterownika jest opisany zestaw funkcji warstwy *Device Driver Layer* niezbędnych dla prawidłowego działania wyświetlacza. Ponieważ SSD1289 jest wspierany przez bibliotekę, to teoretycznie wszystkie funkcje powinny być gotowe. W praktyce jednak konieczne były zmiany w kilku kluczowych procedurach. Jedną z najważniejszych procedur w tej warstwie jest z „zapalenie" jednego piksela matrycy – *PutPixel*. Każdy piksel ma dwa podstawowe atrybuty: adres (wyznaczany na postawie współrzędnych *x, y*) oraz kolor. Standardowa procedura PutPixel ma dwa argumenty: współrzędną *x* i współrzędną *y* (**listing 8**). Kolor piksela jest zawarty w zmiennej globalnej *_color*. Przed ustaleniem adresu i zapisaniem danej sprawdzane jest czy współrzędne *x* i *y* nie mają wartości spoza dopuszczalnego zakresu określonego przez wielkość matrycy. Procedurę *SetAdress* pokazano na **listingu 9**. Zależnie od przyjętej orientacji (pionowa lub pozioma) współrzędne są zapisywane zamiennie komendami o adresie 0x4e, lub 0x4f. Procedurę kończy wysłanie komendy 0x22 (*Write data to GRAM*). Funkcja WritePiksel to makro:

#define WritePixel(color) {LcdWriteData(color);}

Jest ona używana w dwóch kolejnych ważnych procedurach: czyszczącej ekran *ClearDevice()* – **listing 10** oraz rysującej prostokąt *Bar()* – **listing 11**. Obie procedury w wersji oryginalnej nie działały poprawnie.



Rysunek 3. Zapis danych na magistralę

Listing 11. Funkcja rysowania prostokąta

```
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    register SHORT  x, y;
    #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);
    #else
    if(IsDeviceBusy() != 0) return (0);
    #endif
    if(_clipRgn)
    {
        if(left < _clipLeft) left = _clipLeft;
        if(right > _clipRight)right = _clipRight;
        if(top < _clipTop) top = _clipTop;
        if(bottom > _clipBottom) bottom = _clipBottom;
    }
    {
        for(y = top; y < bottom + 1; y++)
            for(x = left; x < right + 1; x++)
                PutPixel(x, y);
    }
    return (1);
}
```

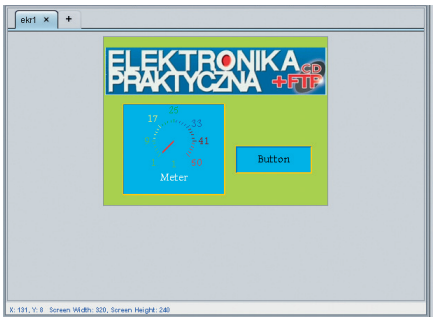
Listing 10. Czyszczenie ekranu

```
void ClearDevice(void)
{
    DWORD   counter;
    SetAddress(0, 0);
    for(counter = 0; counter < (DWORD) (GetMaxX() + 1) * (GetMaxY() + 1); counter++)
    {
        WritePixel(_color);
    }
}
```

Tabela 1. Połączenie wyświetlacza z liniami portów mikrokontrolera

Sygnał	Wyprowadzenie wyświetlacza	Port
CS – aktywacja interfejsu, aktywny poziom niski	CS	RG15
RS (D/!C) – wybór rejestru: poziom niski – rejestr konfiguracyjny, poziom wysoki – dane pamięci obrazu	RS	RG14
RD – odczyt danych z magistrali, aktywny poziom niski	RD	RG12
WR – zapis danych na magistrali, aktywny poziom niski	WR	RG13
RES – zerowanie sterownika, aktywny poziom niski	RES	RB15
DATAH – 8 starszych bitów danych	DB8…DB15	RE0….RE7
DATAL – 8 młodszych bitów danych	DB0…DB7	RD0…RD7

Szczególnie *Bar()* przysporzyła mi trochę kłopotów. Wynikały one z tego, że biblioteka ma możliwość konfigurowania orientacji, a ja przyjąłem, że wyświetlacz będzie zorientowany poziomo. Oryginalna procedura z *drvTFT2.c* nie chciała w tej orientacji pracować poprawnie. Na początku funkcji *Bar()* następuje sprawdzenie czy współrzędne wierzchołków prostokąta nie mają wartości większych, niż wynika to z wielkości matrycy wyświetlacza. Po napisaniu i uruchomieniu procedur komunikacji ze sterownikiem wyświetlacza, poprawnym zainicjowaniu sterownika i korekty procedur *Bar* i *ClearDevice* można by było w zasadzie przystąpić do prób działania biblioteki. Jednak GDD-X generuje projekt, w którym jest definiowana obsługa rezystancyjnego panelu dotykowego, którym to zajmiemy się później. Na tym etapie ta obsługa musiała być wyłączona przez usunięcie definicji *#define USE_TOUCHSCREEN*, żeby projekt mógł się skompilować bez błędów. Nie są wtedy kompilowane funkcje obsługi paneli z pliku *TouchScreen.c*.



Rysunek 4. Okno projektu ekranu w Graphics Display Designer

Graphic Display Designer X – testy działania

Mamy już wszystko gotowe by rozpocząć testy działania biblioteki. W tym celu tworzymy projekt w środowisku MPALB X i uruchamiamy wtyczkę Graphic Display-Designer. Ponieważ jeszcze nie mamy obsługi panelu dotykowego, to nie można testować działania interakcji z użytkownikiem. Testy będą polegały tylko na wyświetlaniu obiektów (głównie widżetów). W GDD-X utworzyłem projekt ekranu (**rysunek 4**) z umieszczonymi w nim bitmapą z logo Elektroniki Praktycznej oraz obiektami *Meter* i *Button*. Po skompilowaniu projektu i zaprogramowaniu mikrokontrolera te elementy zostały wyświetlone prawidłowo na ekranie wyświetlacza (**rysunek 5**).

Teraz nadszedł czas na ocenę jakości matrycy wyświetlacza. Pomimo niskiej ceny wyświetlany obraz jest kontrastowy i wyraźny. Nie widziałem zauważalnej różnicy pomiędzy tym wyświetlaczem, a wyświetlaczem z fabrycznego modułu MEB. Ekran wyświetla się bez dużych opóźnień. To zapewne zasługa szybkiego mikrokontrolera, ale też 16-bitowej magistrali równoległej (pomimo zastosowania emulatora programowego).

W tym momencie możemy uznać, że integracja wyświetlacza z biblioteką zakończyła się częściowym sukcesem. Częściowym, bo przed nami jeszcze ważny element: obsługa interfejsu dotykowego, pozwalająca na interakcję graficznego interfejsu z użytkownikiem.

Rysunek 5. Efekt działania projektu ekranu z GDD-X

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
S	A2	A1	A0	MODE	SER/DFR	PD1	PD0

S – bit startu pomiaru (konwersji ADC) – aktywna jedynka
A2…A0 – wybór kanału pomiarowego przetwornika A/C
MODE – rozdzielczość pomiaru: jedynka pomiar 12-bitowy, zero pomiar 8-bitowy
SER/DFR topologia układu wejściowego: jedynka wejście ***Single Ended***, zero wejście różnicowe
PD1..PD0 – tryb poboru mocy w stanie pomiędzy pomiarami

Rysunek 6. Rejestr komendy

Panel dotykowy – obsługa sterownika XPT2046

Jak już wspomniałem, wyświetlacz jest wyposażony w 4-przewodowy, rezystancyjny panel dotykowy. Taki panel może być obsłużony przez mikrokontroler z wbudowanym przetwornikiem analogowo-cyfrowym lub za pomocą wyspecjalizowanego kontrolera – tu jest to XPT2046. Biblioteka Microchipa wspiera obsługę za pomocą przetwornika (plik *TouchScreenResistive.c*) oraz specjalizowanego kontrolera AR1020 (plik *AR1020.c*). Jak widać, nie ma wsparcia dla XPT2046 i w zasadzie prawie identycznego układu ADS7843. Dlatego trzeba napisać obsługę od podstaw.

Zasada działania pomiaru zmiany rezystancji rezystancyjnego panelu dotykowego jest opisana w wielu źródłach. Można tez o tym poczytać w dokumentacji układu. XPT2046 komunikuje się z mikrokontrolerem z pomocą szeregowego interfejsu SPI. Podobnie jak przy komunikacji ze sterownikiem wyświetlacza, obsługa magistrali została wykonana programowo, bo w trakcie uruchamiania w programowych emulacjach jest łatwiej znaleźć błędy. Po uruchomieniu, kiedy wszystko działa prawidłowo, można zamienić programową obsługę na transmisję wykonywaną przez sprzętowe moduły komunikacyjne.

Z punktu widzenia XPT2046, magistrala SPI składa się z następujących linii: danych wyjściowych DOUT, danych wejściowych DIN, zegara taktującego DCLK oraz linii wyboru interfejsu CS. Interfejs komunikacyjny jest uzupełniony o linię sygnalizacyjną PENIRQ. Na **listingu 12** pokazano definicję linii portów, a w tabeli 1 zamieszczono sposób połączenia mikrokontrolera z wyświetlaczem.

Po zdefiniowaniu linii trzeba je z zainicjować przez zdefiniowanie kierunku (wejście/wyjście) oraz stanu początkowego linii – **listing 13**. Do komunikacji z układem potrzebne będą dwie procedury: zapisu 8-bitowej danej (**listing 14**) i odczytu 12-bitowej danej (**listing 15**). Do sterownika będziemy zapisywali komendy sterujące jego pracą, a potem odczytywali 12-bitową wartość rezystancji zmierzoną przez przetwornik A/C.

W trakcie uruchamiania transmisji napotkałem na nieoczekiwane problemy.

Tabela 1. Połączenie mikrokontrolera z wyświetlaczem za pomocą SPI

Nazwa sygnału	Wyprowadzenie modułu wyświetlacza	Port
TP_DIN Dane wejściowe DIN	T_DIN	RG1
TP_DOUT Dane wyjściowe DOUT	T_DOUT	RG0
TP_CLK Zegar	T_CLK	RC3
TP_CS Wybór interfejsu	T_CS	RC4
TP-PENIRQ sygnalizacja naciśnięcia	T_IRQ	RA6

Procedury zostały sprawdzone ze sprzętowym emulatorem modułu PIC32 Starter Kit i zgodnie z dokumentacją powinny działać prawidłowo. Jednak po normalnym uruchomieniu komunikacja nie działała. Obserwacja przebiegu transmisji na oscyloskopie cyfrowym pokazała, że przyczyną problemów były zbyt małe opóźnienia po zmianach stanów na liniach interfejsu SPI. Dodanie dodatkowego opóźnienia *Delay()* wyeliminowało ten problem.

Pomiar położenia naciskanego punktu powinien się rozpocząć po wykryciu momentu naciśnięcia folii panelu dotykowego. Wykrycie naciśnięcia można wykonać programowo testując wartości odczytane z modułu. XPT2046 ma wyprowadzenie PENIRQ, które w czasie naciśnięcia folii przechodzi w stan niski. Wystarczy testować stan PENIRQ by wiedzieć czy panel został przyciśnięty, czy nie. Ja skorzystałem z tego mechanizmu, ponieważ jest dużo prostszy w realizacji i pewny w działaniu.

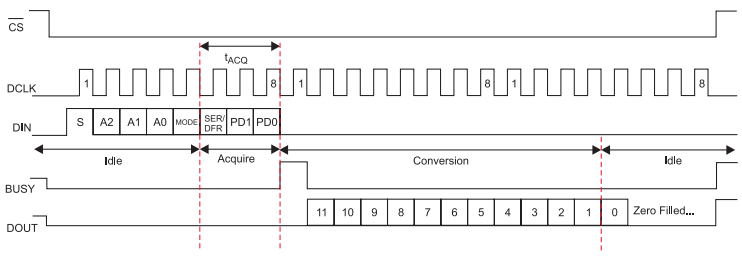
Po wykryciu naciśnięcia można przystąpić do odczytania położenia miejsca nacisku. Jest ono identyfikowane przez współrzędne *x* oraz *y*. Jeżeli wyświetlacz ma rozdzielczość 240×320 pikseli, to *x* zmienia się w zakresie 0<x<319, a y w zakresie 0<y<239 dla poziomej orientacji ekranu. Programowa identyfikacja położenia będzie polegała na odczytaniu współrzędnej *x*, a następnie współrzędnej *y*. O tym, która współrzędna będzie czytana i z jaka rozdzielczością decyduje 8-bitowy kod komendy (**rysunek 6**). Żeby wyzwolić pomiar trzeba ustawić bit startu S, wybrać kanał pomiarowy i skonfigurować przetwornik A/C. Dla wyświetlacza o rozdzielczości 320×240 pikseli wybieramy pomiar o rozdzielczości 12-bitowej, z wejściem różnicowym. Bity PD01…PD0 będą zerowane, tak aby pomiędzy pomiaramiXPT2046 przechodził w stan oszczędzania energii *Power Down*. Odczytanie współrzędnej *x* będzie wymagało wysłania komendy 0x90, a odczytanie współrzędnej *y* komendy 0xD0. Po wysłaniu komendy za pomocą *WriteTpSpi(cmd)* można odczytać wynik konwersji przez wywołanie procedury *ReadTpSpi()*. Ponieważ SPI jest emulowa-

Listing 12. Definicja linii interfejsu SPI XPT2046
```
#define TP_DCLK          PORTCbits.RC3
#define TP_DCLK_TRIS     TRISCbits.TRISC3
#define TP_DIN           PORTGbits.RG1
#define TP_DIN_TRIS      TRISGbits.TRISG1
#define TP_DOUT          PORTGbits.RG0
#define TP_DOUT_TRIS     TRISGbits.TRISG0
#define TP_CS            PORTCbits.RC4
#define TP_CS_TRIS       TRISCbits.TRISC4
#define TP_PENIRQ        PORTAbits.RA6
#define TP_PENIRQ_TRIS   TRISAbits.TRISA6
```

Listing 13. Inicjalizacja interfejsu SPI dla XPT2046
```
void InitTpSpi(void)
{
    TP_CS_TRIS=0;
    TP_DCLK_TRIS=0;
    TP_DIN_TRIS=0;
    TP_DOUT_TRIS=1;
    TP_PENIRQ_TRIS=1;
    TP_CS=1;
    TP_DCLK=1;
    TP_DIN=1;
    TP_CS=1;
}
```

Listing14. Zapis 8-bitowej danej do XPT2046
```
void WriteTpSpi(unsigned char data)
{
    unsigned char i=0;
    TP_DCLK=0;
    for(i=0;i<8;i++)
    {
        if((data&0x80)==0) TP_DIN=0;
        else TP_DIN=1;
        Delay();
        TP_DCLK=0; Delay();
        TP_DCLK=1; Delay();
        data<<=1;
    }
    Delay();
}
```

Listing 15. Odczyt 12-bitowej danej z XPT2046
```
W short ReadTpSpi(void)
{
    unsigned char i=0;
    unsigned short data=0;
    TP_DOUT=0;
    for(i=0;i<12;i++)
    {
        TP_DCLK=1; ;Delay();
        TP_DCLK=0; ;Delay();
        if(TP_DOUT) data|=1;
        data<<=1;
        Delay();
    }
    return(data);
}
```



Rysunek 7. Przebiegi czasowe sekwencji odczytu współrzędnej

ne programowo, to procedura wysyła tylko 12 taktów zegara i odczytuje 12 bitów wyniku. W modułach sprzętowych trzeba będzie wysłać 16 taktów zegara i z wyniku odczytu wziąć 12 młodszych bitów. Przebiegi czasowe sekwencji odczytu współrzędnej zostały pokazane na **rysunku 7**. Trzeba pamiętać, że po wysłaniu komendy układ potrzebuje czasu na wykonanie pomiaru i zmierzona wartość można odczytać po pewnym czasie o wysłania komendy. Do sygnalizowania tego stanu jest przeznaczone wyprowadzenie BUSY. My z niego nie skorzystamy, a potrzebne opóźnienie zostanie odliczone programowo.

Do odczytania współrzędnych są używane dwie funkcje: *TouchGetXRAW()* – **listing 16** i *TouchGetYRAW* – **listing 17**. Obie podają zmierzone wartości rezystancji w rozdzielczości 12 bitowej, a więc ich wartości zmieniają się w zakresie 0.....4095. Żeby uzyskać rzeczywiste współrzędne z zakresu 0<x<319 i 0<y<219 trzeba odczytany wynik przekonwertować. Ale najpierw trzeba ustalić, gdzie znajduje się punkt początkowy o współrzędnych (0,0), bo od tego zależy, jak będzie wykonywana konwersja. Ja do ustalenia współrzędnych tego punktu dla posłużyłem się wtyczką GDD-X. Na ekranie umieściłem obiekt *button* i tak go przesunąłem, by dwie współrzędne były zerowe. Okazało się, ze współrzędne (0,0) są w lewym górnym rogu wyświetlacza.

Listing 16. Odczytanie współrzędnej x

```
int TouchGetXRAW(void)
{
    TP_CS=0;
    Delay();
    WriteTpSpi(0x90);
    Delay();
    X=ReadTpSpi();
    TP_CS=1;
    return((X&0x0fff));
}
```

Listing 17. Odczytanie współrzędnej y

```
int TouchGetYRAW(void)
{
    TP_CS=0;
    Delay();
    WriteTpSpi(0xD0);
    Delay();
    Y=ReadTpSpi();
    TP_CS=1;
    return((Y&0x0fff));
}
```

Listing 18. Odczytanie przekonwertowanej współrzędnej x

```
SHORT TouchGetY(void)
{
    int x;
    float xp;
    if(TP_PENIRQ==1) return(-1);
    DelayMs(30);
    x=TouchGetYRAW();
    if (x > 3900) x = 3900;
    else if (x < 200)
    x = 200;
    xp=(x - 200) / 16.25;
    x=xp;
    if (x > 240)
    x = 240;
    return(x);

}
```

Listing 19. Odczytanie przekonwertowanej współrzędnej y

```
SHORT TouchGetX(void)
{
    int y;
    float yp;
    if(TP_PENIRQ==1) return(-1);
    DelayMs(30);
    y=TouchGetXRAW();
    if (y > 3850) y = 3850;
    else if (y < 150) y = 150;
    yp=((y - 150) / 11.56);
    y=yp;
    if(y>320) y=320;
    return(y);
}
```

Na podstawie tej informacji zostały napisane 2 procedury odczytujące pomiary z XPT2046 i zwracające po przekonwertowaniu współrzędne *x* i *y* punktu dotyku na ekranie. Te procedury zostały pokazane na **listingach 18** i **19.** Na początku obu procedur jest sprawdzany stan wejścia PENIRQ i jeżeli jest on wysoki, to procedura kończy działanie i zwraca wartość *-1*. Jeżeli ten stan jest niski, to odczytywana jest wartość z XTP2046 dla konkretnych kanałów. Każda z procedur zwraca po konwersji swoja współrzędną.

Wróćmy teraz do biblioteki i projektu generowanego w GDD-X. Jak już wspomniałem, taki projekt, w którym wybrano MEB standardowo generuje pliki ze skonfigurowaną obsługą panelu rezystancyjnego za pomocą przetworników A/C mikrokontrolera. Musimy teraz go tak przekonfigurować, aby biblioteka korzystała z obsługi napisanej przez nas. Po pierwsze trzeba wyłączyć kompilację dla procedur umieszczonych *TouchScreenResistive.c* przez usuniecie definicji *#define USE_TOUCHSCREEN_RESISTIVE* w pliku konfiguracji *MY_BOARD.h*. Obsługa panelu dotykowego sprowadza się głównie do funkcji *void TouchGetMsg(GOL_MSG *pMsg)* umieszczonej w pliku *TouchScreen.c*. Ta funkcja korzysta z wcześniej opisanych procedur *TouchGetX()* i *TouchGetY()*. Ponieważ po usunięciu *#define USE_TOUCHSCREEN_RESISTIVE* nie będą funkcje z *TouchScreenResistive.c*, to projekt użyje naszych procedur. Funkcje *TouchGetMsg* na podstawie odczytanych współrzędnych x, oraz y modyfikuje składowe struktury *GOL_MSG* dotyczące obsługi panelu dotykowego.

Jak widać z powyższego opisu, żeby dodać własną obsługę panelu dotykowego trzeba utworzyć dwie procedury, *TouchGetX()* i *TouchGetY()*, zwracające współrzędne dotkniętego punktu i tak skonfigurować bibliotekę, aby z tych procedur korzystała. Zależnie od użytych rozwiązań sprzętowych, trzeba też zmodyfikować procedurę *TouchInit* inicjującą (w naszym przypadku) programowy interfejs SPI.

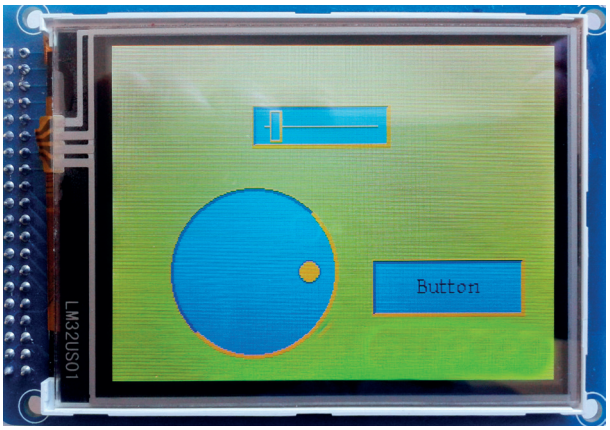
Testowanie działania ekranu dotykowego można szybko sprawdzić przez postawienie na ekranie obiektów wykorzystujących ekran dotykowy. Ja użyłem trzech: przycisku *Button*, suwaka *Slider* i pokrętła *Round Dial* (**rysunek 8**). Testy wykazały, ze wszystkie te obiekty współpracują prawidłowo z obsługa ekranu dotykowego, a ich działanie nie odbiega od działania takich samych obiektów w projekcie przeznaczonym dla firmowego modułu MEB.

Podsumowanie

Pokazałem jak można w praktyce zintegrować dostępny na rynku wyświetlacz LCD z panelem dotykowym z graficzna biblioteką firmy Microchip. Jak już wspomniałem wyświetlacz jest dostępny przynajmniej w czasie pisania tego artykułu i jego cena jest stosunkowo niska. W trakcie pracy nad artykułem miałem możliwość porównania efektów działania z wyświetlaczem modułu Multimedia Expansions Board. Oba wyświetlacze zachowywały się identycznie, poza szybkością rysowania niektórych obiektów. Nieznacznie szybszy był wyświetlacz ze sterownikiem SSD1926 modułu MEB. Wynika to najprawdopodobniej z różnicy w budowie sterowników. Poza tym nie starałem się optymalizować czasowo obsługi komunikacji z SS1289.

Integracja biblioteki z nowym sterownikiem, szczególnie wykonywana po raz pierwszy, nie jest zadaniem łatwym, chociaż być może na takie wygląda. Jest to spowodowane dość sporym skomplikowaniem projektu, dużą liczbą plików źródłowych i nagminnym stosowaniem kompilacji warunkowej. Przykładowe projekty Microchipa maja tę właściwość, że bardzo dobrze działają z modułami formowymi, ale ich modyfikacja dla innego sprzętu wymaga trochę wysiłku. W przypadku biblioteki graficznej naprawdę warto tę pracę wykonać, bo efekty są znakomite.

Tomasz Jabłoński, EP



Rysunek 8. Ekran testowania panelu dotykowego