Biblioteka grafiki firmy Microchip

Stosunkowo niewielkie ceny mikrokontrolerów 16- i 32-bitowych sprawiają, że są one coraz częściej stosowane przez elektroników, nawet tych, którzy „wychowali się" na 8-bitowcach. Duża wydajność obliczeniowa i spora pamięć programu umożliwia tworzenie efektownych interfejsów użytkownika wykorzystujących kolorowe wyświetlacze graficzne. Tworzenie tanich interfejsów graficznych stało się możliwe dzięki drastycznemu obniżeniu cen kolorowych wyświetlaczy o dużej rozdzielczości, a także....

Graficzne interfejsy użytkownika w mikroprocesorowych systemach sterowania można projektować i tworzyć za każdym razem od nowa, na przykład w formie łączenia bitmap z tekstem. Jest to dobre rozwiązanie dla niewielkich projektów. Jeżeli interfejs jest rozbudowany, lub planujemy tworzenie wielu interfejsów w różnych projektach, najbardziej korzystne jest stworzenie lub skorzystanie z gotowej biblioteki zawierającej funkcje graficzne. Firma Microchip opracowała dla swoich mikrokontrolerów 16-bitowych (PIC24 i dsPIC33) oraz 32-bitowych (PIC32) bezpłatną bibliotekę funkcji graficznych Microchip Graphic Library o imponujących możliwościach:

- tworzenie grafiki dla ekranów o rozdzielczości do 320×240 pikseli (QVGA),
- 16-bitowe kodowanie informacji o kolorze (65 k kolorów),
- tworzenie dwuwymiarowych obiektów graficznych (linie, okręgi, tekst itp.),
- tworzenie 3-wymiarowych obiektów graficznych (przyciski, suwaki, mierniki, bargrafy itp.),
- umieszczanie obrazków(bitmap) i animacji,
- obsługa ekranów dotykowych.

Biblioteka oferuje interfejs API (*Application Programming Interface*) składający się z warstw (**rys. 1**):

- aplikacji,
- obiektów graficznych,
- prostych elementów grafiki,
- driverów wyświetlacza.

Warstwa aplikacji to program, który używa graficznych funkcji bibliotecznych. Inaczej mówiąc jest to program, który jako interfejs użytkownika wykorzystuje kolorowy wyświetlacz graficzny. Może to być oprogramowanie sterownika, w którym są graficznie przedstawione bieżące stany kontrolowanych procesów, wprowadzane nastawy itp. Interfejs użytkownika składa się również z elementów manipulacyjnych, na przykład myszy, klawiatury stykowej lub ekranu dotykowego.

Polecenia są przesyłane z warstwy aplikacji przez interfejs poleceń do warstwy obiektów graficznych (*Graphics Object Layer*). W tej warstwie są zaimplementowane bardziej zaawansowane funkcje graficzne. Mogą to być funkcje rysowania i pozycjonowania na ekranie przycisków trójwymiarowych 3D, funkcje rysowania przyrostów (bargrafy), zdefiniowane okna itp. Wykorzystując funkcje tej warstwy można budować jak z klocków skomplikowane reprezentacje graficzne. W obecnie dostępnej wersji biblioteki warstwa Graphic Object Layer oferuje następujące obiekty: przyciski, suwaki, panele kontrolne, przewijane bargrafy, statyczny tekst, obrazki (bitmapy) i mierniki.

Funkcje z warstwy obiektów graficznych wykorzystują funkcje rysowania prostych elementów grafiki: linii, okręgów, tekstu. Z kolei te elementy wykorzystują proste funkcje manipulacji pikselami ekranu z warstwy driverów wyświetlacza.

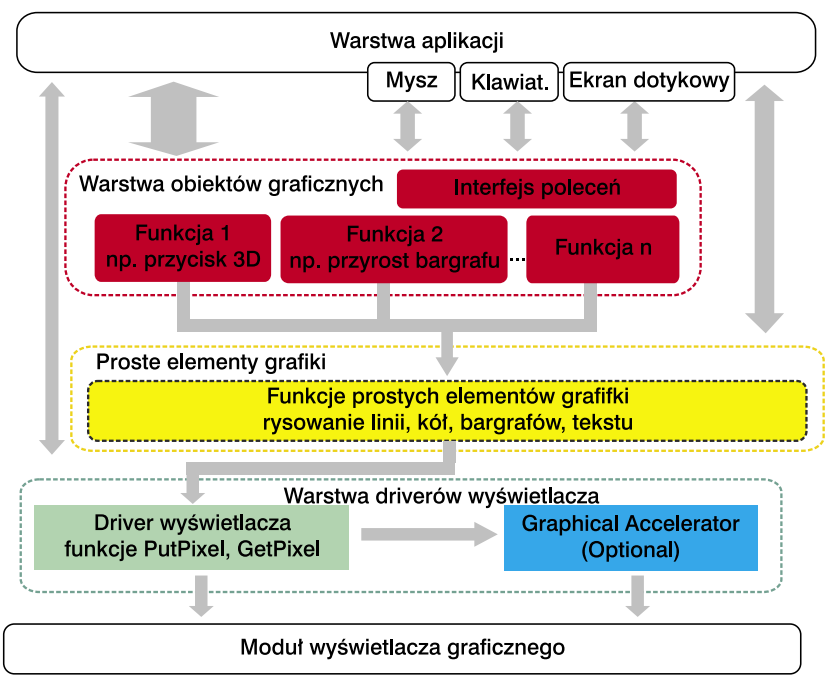
Warstwa driverów wyświetlacza umożliwia komunikację z konkretnym typem sterownika wbudowanego w wyświetlacz. W wersji V1 biblioteki graficznej obsługiwane są sterowniki:

- Samsung S6D0129, S6D0139
- LG LGDP4531
- Densitron HIT1270
- Solomon Systech SSD1339

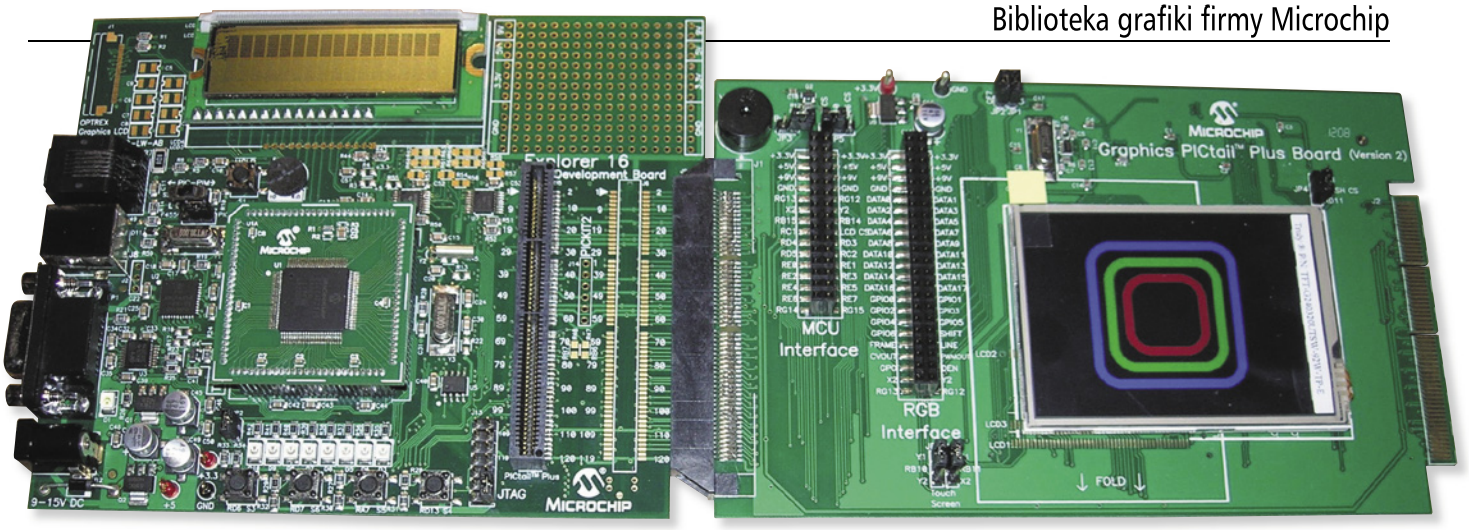
W nowszych wersjach są planowane drivery dla sterowników Solomon Systech, Himax i Ilitek.

Wymagania systemowe biblioteki nie są wygórowane jak na operacje graficzne. Potrzebny będzie 16-, lub 32-bitowy mikrokontroler firmy Microchip z odpowiednio dużą pamięcią oraz opcjonalną zewnętrzną pamięcią Flash do zapisywania dodatkowych elementów grafiki. Standardowa biblioteka graficzna zajmuje ok. 8 kB pamięci programu Flash i 60 bajtów RAM. Dodatkowo każdy obiekt, taki jak linia, okrąg, przycisk wymaga 1 kB pamięci Flash i 10 bajtów pamięci RAM plus ok. 25 bajtów pamięci przydzielanej dynamicznie. Każdy krój czcionek zajmuje ok. 7 kB pamięci Flash. Magistrala łącząca mikrokontroler ze sterownikiem wyświetlacza ma szerokość 8 lub 16 bitów i mikrokontroler musi mieć do tego celu zarezerwowane linie portów.

Bibliotekę można pobrać ze strony producenta – *www.microchip.com/graphics* w formie spakowanego pliku. Po rozpakowaniu otrzymujemy pliki licencji i plik instalacyjny. W trakcie



Rys. 1. Warstwowy model interfejsu API



Fot. 2. Moduły uruchomieniowe Explorer16 i Graphics PICtail Plus Daughter Board

instalacji, która tak naprawdę jest procesem rozpakowywania archiwów, wszystkie pliki są standardowo umieszczane na dysku C w katalogu *Microchip Solutions* w pięciu podkatalogach: *Graphic Primitive Layer Demo, Graphic Object Layer Demo, Graphic External memory Demo, Graphic AN1136* i *Graphic AN1182.*

Żeby ułatwić potencjalnemu użytkownikowi poznawanie nowego oprogramowania programy demonstracyjne każdej z warstw i not aplikacyjnych są gotowymi projektami środowiska MPLAB IDE przeznaczonymi do skompilowania kompilatorem MPLAB C30. Programy demonstracyjne są tak napisane, że można je natychmiast uruchomić w zestawie składającym się z modułu uruchomieniowego Explorer16 połączonego z modułem Graphics PICtail Plus Daughter Board (**fot. 2**) zawierającego wyświetlacz QVGA oraz pamięć Flash o pojemności 512 kB. Pamięć ta jest przeznaczona do zapisywania elementów grafiki.

Umieszczony w katalogu *Graphic Primitive Layer Demo* projekt pozwala na zapoznanie się z funkcjami dostępnymi w tej warstwie. Po skompilowaniu projektu i zaprogramowaniu mikrokontrolera PIC24FJ128GA010 w module Explorer16, na ekranie wyświetlacza można zobaczyć możliwości, jakie dają procedury zaimplementowane w tej warstwie. Kolejno są wyświetlane linie proste, okręgi, koła wypełniane różnymi kolorami, figury będące połączeniem prostych i łuków, kwadraty, romby. Oprócz figur geometrycznych wyświetlany jest tekst i bitmapa w różnych rozdzielczościach (**fot. 3**).

Sam projekt jest bardzo przydatny, bo pozwala na poznanie praktycznych implementacji funkcji warstwy, ale nieodzowne jest też przestudiowanie pliku pomocy *Graphic Library Help* zawartego w katalogu *Microchip Solutions\Microchip\Help*. Funkcje *Primitive Layer Demo* są opisane w grupach:

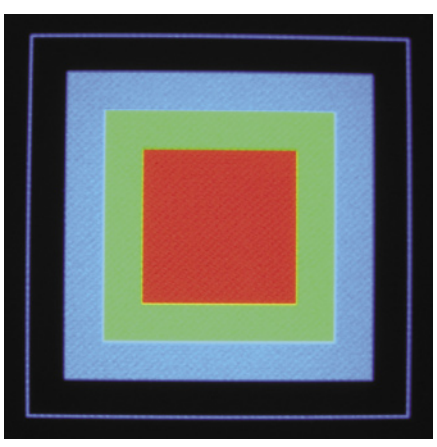
- 2 funkcje *Set Up*. Pierwsza z nich - *Clear Device* czyści ekran i ustawia kursor grafiki na pozycję (0,0). Druga funkcja *InitGraph* inicjalizuje sterownik wyświetlacza.
- Funkcje *Text Functions* są związane z wyświetlaniem tekstu. Są to: ustawienie wzorca znaku (*SetFont*), wyświetlenie znaku (*OutChar*), wyświetlenie ciągu znaków (*OutText*) itp.
- Funkcje rysowania linii (*Line, DrawPoly*).
- Funkcje rysownia prostokątów (*Rectangle Functions*).
- Funkcje rysownia okręgów (*Circle Functions*).
- Funkcje pozycjonowania graficznego kursora (*Graphic Cursor*).
- Funkcje związane z wyświetlaniem bitmap (*Bitmap Functions*).
- Funkcje obsługi zewnętrznej pamięci Flash.

Warstwa obiektów graficznych *Graphic Object Layer GOL* jest zbudowana z bardziej zaawansowanych funkcji niż zaimplementowane w warstwie *Primitive Layer*. Ich poznanie ułatwia analiza projektu zapisanego w katalogu *Graphic Object Layer Demo*. Działanie tej aplikacji polega na wyświetlaniu okien z działającymi obiektami. Nawigację po projekcie i sprawdzanie działania obiektów umożliwia zaimplementowana funkcja obsługi ekranu dotykowego. Po pierwszym zaprogramowaniu mikrokontrolera aplikacja przeprowadza kalibrację ekranu dotykowego, który dopiero wtedy jest gotowy do działania. Jako pierwszy, wyświetlany jest ekran z bitmapą z logo Microchip. Kolejne ekrany są wyświetlane po naciśnięciu umieszczonych na krawędziach ekranu, specjalnie do tego celu przygotowanych graficznych elementów. Trzeba przyznać, że inżynierowie firmy Microchip zrobili wszystko, by w tej dość rozbudowanej aplikacji pokazać duże możliwości tworzenia graficznego interfejsu wykorzystującego ekran dotykowy.

Funkcje stosowane do inicjalizacji i definiowania obiektów warstwy obiektów graficznych są również opisane w pliku pomocy *Graphic Library Help*. Dokumentacja każdej z nich jest dość obszerna. Jako przykład mogą służyć funkcje obiektu *Meter*, przeznaczone do definiowania graficznego miernika na ekranie wyświetlacza. Można zdefiniować trzy style wyświetlanego miernika określane parametrami `MTR_WHOLE_TYPE`, `MTR_HALF_TYPE` lub `MTR_QUARTER_TYPE`. Definicja może wyglądać na przykład tak:

```
#define METER_TYPE MTR_HALF_TYPE
```

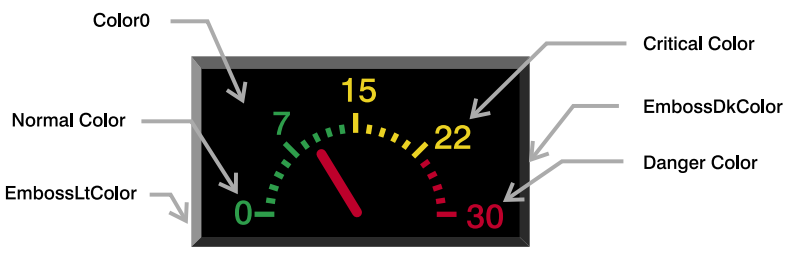
Na **rys. 4**. pokazano miernik typu MTR_HALF_TYPE. Kolory skali miernika są definiowane przez makro *MtrScaleColors* (**list. 1**),
gdzie `pMtr` jest wskaźnikiem do obiektu, `normal` określa kolor zakresu normalnych wartości, `critical` określa kolor krytycznych wartości, a `danger` określa kolor krytycznego przekroczenia wartości.



Rys. 3. Przykładowe figury wyświetlane na ekranie

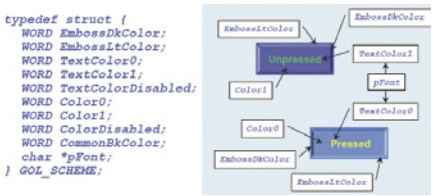
List. 1. Makro *MtrScaleColors* definiujące kolor skali miernika

```
#define MtrSetScaleColors(pMtr, normal, critical, danger)     \
                    {     pMtr->normalColor=normal;           \
                          pMtr->criticalColor=critical;       \
                          pMtr->dangerColor=danger;     }
```

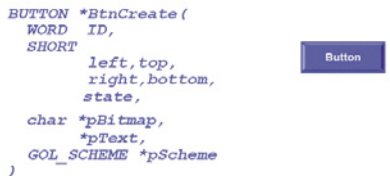


Rys. 4. Graficzna prezentacja obiektu Meter

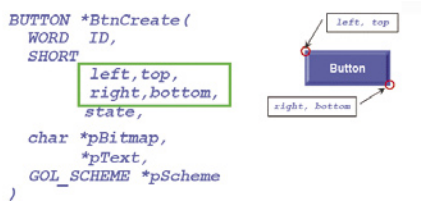
List. 2. Tworzenie miernika funkcją *MtrCreate*
```
define ID_METER 101

GOL_SCHEME *pMeterScheme;
METER *pMtr;

 pMeterScheme = GOLCreateScheme();

 pMtr=MtrCreate(
       ID_METER,           // przydzielenie identyfikatora  ID
       30, 50, 150, 180,   // ustawienie rozmiarów
       MTR_DRAW|MTR_RING,  // rysuje obiekt po stworzeniu
       0,                  // ustawienie wartości początkowej
            100,           // ustawienie zakresu
            „Speed”,       // tekst etykiety
            pMeterScheme); // styl miernika

    // sprawdzenie, czy miernik został utworzony
 if (pMtr == NULL)
  return 0;
 MtrSetScaleColors(pMtr, WHITE, BLUE, RED); //ustawienie koloru skali
z predefiniowanymi wartościami

 while(!MtrDraw(pMtr);//rysuj miernik
 return 1;
```



Rys. 6. Struktura stylu obiektów graficznych



Rys. 7. Funkcja definiowania przycisku BtnCreate



Rys. 8. Określanie współrzędnych przycisku

Tworzenie miernika (**fot. 5**) zaczyna się od wywołania funkcji *MtrCreate* z parametrami określającymi rozmiar, identyfikator, zakres pomiarowy, wartość początkową, etykietę pod miernikiem i jego typ. Funkcja typu *Create* tworzy wzór obiektu w pamięci, ale go nie wyświetla. Przykład wywołania funkcji tworzenia obiektu przedstawiono na **list. 2**.

Narysowanie miernika na ekranie jest realizowane wywołaniem funkcji *MtrDraw* z parametrami określonymi w strukturze zapisanej przez funkcję *MtrCreate*. Definicję struktury jest następująca:

```
typedef struct {
  OBJ_HEADER hdr;
  XCHAR * pText;
  SHORT value;
  SHORT range;
  SHORT xCenter;
  SHORT yCenter;
  SHORT radius;
  SHORT xPos;
  SHORT yPos;
  WORD normalColor;
  WORD criticalColor;
  WORD dangerColor;
} METER;
```

Został tu pokazany ogólny opis jednej z funkcji obiektu warstwy obiektów graficznych. Wykorzystanie w praktyce każdej z funkcji wymaga poznania szczegółów niezbędnych do prawidłowego ich zastosowania.

W warstwie driverów wyświetlacza są umieszczone 2 funkcje: *PutPixel* i *GetPixel*. Funkcja *PutPixel* zapala piksel wyświetlacza o współrzędnych określonych w argumentach funkcji, której prototyp przedstawiono niżej:

```
void PutPixel(
    SHORT x,
    SHORT y
);
```

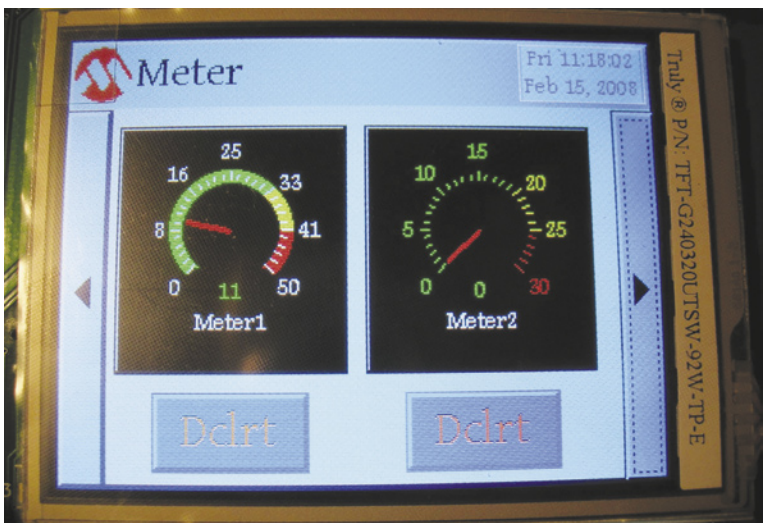
Przed zapaleniem punktu trzeba określić kod koloru funkcją *SetColor*. Kolor jest zakodowany w 16-bitowej liczbie w formacie 5:6:5 RGB. Kolor B (niebieski) jest zakodowany na 5 najmłodszych bitach, zielony na 6 kolejnych, a czerwony na 5 najstarszych bitach.

Funkcja *GetPixel* zwraca 16-bitowy kod koloru piksela o współrzędnych określonych w argumentach funkcji. Dodatkowo, w warstwie driverów są zdefiniowane makra pozwalające odczytać kolor bieżącego piksela (określonego przez kursor graficzny), i odczytać maksymalne współrzędne wyświetlacza (jego rozmiar).

Korzystanie z biblioteki graficznej rozpoczyna się od wywołania funkcji inicjalizacji *InitGraph()* zerującej sterownik wyświetlacza, ustawiającej kursor na pozycję (0,0) i wygaszającą wyświetlacz. Kolejną czynnością jest określenie stylu przez funkcję *GOLCreateScheme()*. Funkcja inicjalizuje strukturę określającą styl obiektów graficznych tworzonych w warstwie *Graphic Object Layer*. Na **rys. 6** pokazano strukturę stylu i wpływ wartości jej składowych na styl obiektów. Jeżeli struktura stylu nie zostanie jawnie zmieniona, to ważne będą ustawienia domyślne.

Kolejnym krokiem będzie stworzenie obiektu warstwy obiektów graficznych. Do tego celu są wykorzystywane funkcje *ObjCreate*. Na przykład przycisk jest definiowany funkcją *BtnCreate* (**rys. 7**). Parametry `left`, `top`, `right` i `bottom` określają rozmiar przycisku (**rys. 8**). Parametr `state` definiuje stan przycisku w czasie inicjalizacji (**list. 3**).

Parametry `pBitmap` i `pText` powinny być ustawione na `NULL` jeżeli nie będą używane. Tekst określony przez `*pText` jest wyświetlany na przycisku i może być umieszczony centralnie, z lewej strony, z prawej strony, na dole lub na górze przycisku. Jeżeli struktura stylu `pScheme` nie będzie użyta, parametr musi mieć wartość `NULL`. Na **list. 4** pokazano tworzenie trzech różnych przycisków



Fot. 5. Wygląd ekranu miernika z artykułu

List. 3. Przykładowe parametry definiujące przycisk
```
#define BTN_FOCUSED 0x0001  //przycisk wybrany
#define BTN_PRESSED 0x0002  //przycisk w stanie przyciśniętym
#define BTN_DISABLED 0x0010 //przycisk nieaktywny
#define BTN_DRAW 0x4000     //bit informujący, że przycisk będzie ponownie rysowany
#define BTN_REMOVE 0x8000   //bit informujący, że przycisk będzie usunięty z ekranu
```

Jak zostało wcześniej powiedziane, funkcja *Create* tworzy obiekt w pamięci, ale go nie rysuje na ekranie wyświetlacza. Do rysowania stworzonego obiektu przeznaczone są funkcje *Draw*. Wcześniej zdefiniowany przycisk funkcją *BtnCreate* można narysować przez wywołanie funkcji *BtnDraw(pB)*, gdzie `pB` jest strukturą opisującą obiekt stworzoną przez funkcję *BtnCreate*.

Parametr `state` może być wpisany bezpośrednio w argument funkcji lub ustawiany funkcją *SetState*.

W opisywanym już modelu warstwowym powiedzieliśmy, że warstwa aplikacji przekazuje graficznej warstwie obiektów polecenia przez interfejs poleceń (rys. 1). Polecenia są przekazywane przez strukturę przedstawioną niżej:

```
typedef struct {
BYTE type;
BYTE event;
SHORT param1;
SHORT param2;
} GOL_MSG;
```

Parametr `type` określa rodzaj elementu manipulacyjnego, np. ekran dotykowy lub klawiaturę stykową. W zależności od wartości parametru `type` interpretowana jest wartość parametrów `param1` i `param2`. Dla ekranu dotykowego będą to współrzędne (x,y) wyrażone w pikselach. Parametr `event` precyzuje rodzaj wykonywanej operacji. Dla ekranu dotykowego są zdefiniowane 3 wartości parametru `event`:

List. 4. Tworzenie trzech różnych przycisków

```
GOL_SCHEME *pScheme;
BUTTON *buttons[3];
WORD state;

 pScheme=GOLCreateScheme();
 state=BTN_DRAW;

 buttons[0]=BtnCreate(1,20,64,50,118,0, state, NULL, "ON", pScheme);
 // check if button 0 is created
 if (buttons[0] == NULL)
  return 0;

 buttons[1]=BtnCreate(2,52,64,82,118,0, state, NULL, "OFF", pScheme);
 // check if button 1 is created
 if (buttons[1] == NULL)
  return 0;

 buttons[2]=BtnCreate(3,84,64,114,118,0, state, NULL, "HI", pScheme);
 // check if button 2 is created
 if (buttons[2] == NULL)
  return 0;
 return 1;
```

- `EVENT_PRESS`
- `EVENT_RELEASE`
- `EVENT_MOVE`

Łatwo sobie wyobrazić, jak będzie wyglądała struktura polecenia naciśnięcia ekranu dotykowego w punkcie o współrzędnych (x,y).

Ten, z konieczności krótki, opis jest próbą zwrócenia uwagi na niesłychanie interesujące narzędzie służące do tworzenia graficznych interfejsów użytkownika. Jego zaletą jest warstwowy model, doskonale przygotowana dokumentacja i przykładowe projekty pozwalające prawie natychmiast przetestować działanie w praktyce. Nie bez znaczenia jest też fakt, że biblioteka jest całkowicie bezpłatna, co jeszcze nie tak dawno temu nie było do pomyślenia. O tym, jak poważnie firma traktuje potencjalnych klientów, niech świadczy fakt umieszczenia w zasobach biblioteki programu *Font & bitmap converter* pozwalającego konwertować bitmapy na kod tablicy języka C. Dodatkowo, program może tworzyć tablice wzorców znaków alfanumerycznych (fontów) dostępnych w systemie Windows. Oczywiście wykorzystanie narzędzi wymaga pewnego doświadczenia w programowaniu w języku C i raczej nie jest przeznaczone dla początkujących.

Tomasz Jabłoński, EP
tomasz.jabłoński@ep.com.pl