Biblioteka graficzna firmy Microchip

Wyświetlacze TFT LCD o „rozsądnej” rozdzielczości i odpowiednich wymiarach nie są zbyt drogie, a poza tym mają (lub mogą być wyposażone w) ekrany dotykowe podnoszące atrakcyjność interfejsu i jednocześnie ułatwiające jego użytkowanie. Zaprojektowanie i wykonanie interfejsu graficznego w urządzeniach wymagających interakcji z użytkownikiem, to jedna z najbardziej czasochłonnych czynności i to do tego stopnia, że nawet dużym firmom bardziej opłaca się zakup gotowych bibliotek, niż opracowywanie wszystko do podstaw.

Dla małych firm i hobbystów kupowanie drogiego oprogramowania jest nie do przyjęcia. Wyjściem jest samodzielne zaprojektowanie własnej biblioteki na miarę możliwości, lub skorzystanie z rozwiązań darmowych. O tym, że darmowe nie oznacza gorsze możemy się przekonać poznając budowę i właściwości biblioteki graficznej firmy Microchip.

Testowanie funkcji biblioteki – moduł Multimedia Expansion Board

Do testowania funkcji bibliotecznych użyłem modułu Multimedia Expansion Board oraz modułu *PIC32 USB Starter Kit II* z mikrokontrolerem PIC32MX795F521L. Oba moduły są połączone ze sobą specjalnym złączem (**rysunek 1**). Taki zestaw jest bardzo wygodny do przeprowadzania testów, bo firmowe programy przykładowe mają zdefiniowaną miedzy innymi tą konfigurację sprzętową. Wybór konfiguracji sprzętowej jest standardowo wykonywany w pliku nagłówkowym *HardwareProfile.h*. Fragment tego pliku zamieszczono na **listingu1**.

Po usunięciu komentarza z linii *#include „Configs/HWP_MEB_PIC32_USB_SK_16PMP.h* jest dołączany plik konfiguracyjny *Configs/HWP_MEB_PIC32_USB_SK_16PMP.h* zawierający wszystkie niezbędne definicje 16-bitowego interfejsu komunikacyjnego, stałych używanych przez procedury obsługi kontrolera wyświetlacza, linii interfejsu dotykowego itp.

Wyświetlacz modułu MEB (**fotografia 1**) ma rozdzielczość 320×240 pikseli, 16-bitową głębię koloru i rezystancyjny panel dotykowy. Matryca LCD jest sterowana przez układ SSD1926 produkowany przez firmę Solomon Systech.

Model warstwowy

Graficzna biblioteka Microchipa jest przeznaczona dla mikrokontrolerów rodzin PIC24 i PIC32. Mniejsze 8-bitowe mikrokontrolery mają zbyt małe zasoby, by funkcje biblioteczne mogły poprawnie działać.

Przy realizacji skomplikowanego, rozbudowanego oprogramowania dobrze sprawdza się model warstwowy. Dobrze znanym przykładem takiego podejścia jest stos protokołu TCP/IP. Wykonywane zadania są funkcjonalnie podzielone na części i umieszczane w warstwach.

Biblioteka graficzna ma również budowę warstwową i składa się z 3 warstw, od najniższej do najwyższej: *device driver layer*, *primitive layer* i *object layer* (**rysunek 2**). Najniżej umieszczony element to fizyczny wyświetlacz LCD ze sterownikiem matrycy. Warstwa *Device Driver Layer* jest ściśle powiązana z zastosowanym wyświetlaczem, a właściwie z zastosowanym sterownikiem matrycy LCD. Procedury rysowania podstawowych

Fotografia 1. Platforma sprzętowa: moduł MEB z PIC32USB Stater kit II – widok od strony dołączonego modułu Starter Kit II

elementów graficznych są zaimplementowane w warstwie *Graphics Primitive Layer*.

Nad warstwą *Graphic Primitive* jest umieszczona warstwa rysowania obiektów graficznych (widżetów) *Graphics Object Layer*. Korzysta ona z funkcji umieszczonych w warstwie *Graphics Primitive* i jest przeznaczona do rysowania obiektów typu przycisk (*button*), miernik (*meter*), suwak (*slider*), przycisk wyboru (*radio button*) itp.

Na rys. 2 pokazano jeszcze jedną warstwę – aplikacji. Nie jest to warstwa biblioteki, ale tworzona przez użytkownika, który musi zapewnić sobie komunikację pomiędzy warstwą obiektów graficznych, a warstwą aplikacji wykorzystując zestaw funkcji przesyłających polecenia (*Message interface*).

Konfiguracja biblioteki

Biblioteka może być skonfigurowana do pracy w 2 trybach: *Blocking* lub *Non Blocking*. W trybie *Blocking* wszystkie funkcje rysowania obiektów graficznych blokują wykonywanie innych funkcji przez program użytkownika. Inaczej mówiąc, cały program jest zatrzymywany aż funkcja biblioteki narysuje obiekt. W trybie *Non Blocking* funkcje

Listing 1. Konfiguracja sprzętowa

```
/*********************************************************
     * Hardware Configuration for
     * Starter Kit
     * MultiMedia Development Board
     * Display TFT-G240320LTSW-118W-E
*********************************************************/
    //#include „Configs/HWP_MEB_PIC32_STK_8PMP.h”
    //#include „Configs/HWP_MEB_PIC32_USB_SK_8PMP.h”
    //#include „Configs/HWP_MEB_PIC32_ETH_SK_8PMP.h”
    //#include „Configs/HWP_MEB_PIC32_GP_SK_16PMP.h”
    #include „Configs/HWP_MEB_PIC32_USB_SK_16PMP.h”  //konfiguracja sprzętowa
MEB+PIC32USB
    //#include „Configs/HWP_MEB_PIC32_ETH_SK_16PMP.h”
    //#include „Configs/HWP_MEB_dsPIC33E_SK_8PMP.h”
```

rysowania nie czekają na zakończenie rysowania, tylko przekazują sterowanie rysowaniem do programu użytkownika, który musi sam stwierdzić przez cykliczne sprawdzanie warunku wykonania procesu rysowania czy rysowanie zostało zakończone.

Warstwa *Device Driver Layer*

Jest to warstwa dostarczająca funkcji odpowiadających za obsługę sterownika wyświetlacza. Sterowniki kolorowych matryc LCD TFT są produkowane przez różne firmy. W ich ofertach można spotkać wiele sterowników różniących się między sobą. Te różnice wynikają z zastosowanego interfejsu komunikacyjnego, możliwości obsłużenia matryc o innych rozdzielczościach i głębiach koloru, funkcjami obsługi palety koloru itp.

Każdy ze sterowników, nawet z obrębie jednego producenta, ma własne procedury inicjalizacji i funkcje odpowiadające za podstawowe operacje na pikselach matrycy. Biblioteka standardowo obsługuje zestaw popularnych układów scalonych, ale jeżeli wyświetlacz ma inny sterownik, to użytkownik musi sobie samodzielnie napisać oprogramowanie tej warstwy.

Firma Microchip również dostarcza dwa własne, programowe sterowniki matrycy LCD. Jeden wykorzystujący mikrokontroler PIC24FJ256DA210, a drugi mikrokontrolery z rodziny PIC32MX. Piny mikrokontrolerów sterują bezpośrednio wyprowadzeniami RGB wyświetlacza, a funkcje biblioteki graficznej są uruchamiane bezpośrednio na mikrokontrolerze sterownika. W tabeli 1 umieszczono wykaz obsługiwanych sterowników.

Na **rysunku 3** pokazano strukturę plików wykorzystywanych przez warstwę *Device Driver Layer*. Jeśli używamy sterownika wyświetlacza z listy z tab. 1, to ta warstwa z punktu widzenia programisty nie zawiera niczego interesującego. Przy tworzeniu aplikacji trzeba zainicjować sterownik wyświetlacza i wszystko powinno działać pod warunkiem, że sprawny jest sprzęt: wyświetlacz ze sterownikiem, magistrala sterująca (połączenia) i układ mikrokontrolera. Jeżeli mamy wyświetlacz z innym sterownikiem, to trzeba dokładnie znać działanie i programowanie sterownika wyświetlacza i samemu napisać szereg funkcji stanowiących warstwę *Device Driver Layer*. Najważniejsze z nich, to:

- Reset Device() – inicjalizuje wyświetlacz. Inicjalizacja polega na zapisaniu rejestrów sterownika z niezbędnymi ustawieniami.
- GetMaxX() – zwraca maksymalną współrzędną X wyświetlacza.
- GetMaxY() – zwraca maksymalną współrzędną Y wyświetlacza.
- SetColor() – ustawienie koloru rysowania.
- GetColor() – zwraca aktualnie ustawiony kolor rysowania.

Tabela 1. Sterowniki obsługiwane przez bibliotekę graficzną

Układ scalony sterownika	Interfejs	Plik drivera
Microchip Graphics Display Driver – PIC24FJ256DA210	RGB	mchpGfxDrv.c, mchpGfxDrv.h
Microchip Low Cost Controllers Graphic Display Driver – PIC32MX.	RGB	mchpGfxLCC.c, mchpGfxLCC.h
LG LGDP4531	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Renesas R61505U/R61580	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Orise Technology SPDF5408	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Epson S1D13517	8/16 bit PMP	S1D13517.c, S1D13517.h
Epson S1D13522	8/16 bit PMP SPI	S1D13517.c, S1D13517.h
Solomon Systech SSD1926	8/16 bit PMP	SSD1339.c, SSD1339.h
Solomon Systech SSD1289	8/16 bit PMP	SSD1926.c, SSD1926.h
Solomon Systech SSD1339 for OLED displays	8 bit PMP	SSD1339.c, SSD1339.h
Solomon Systech SSD1303 for OLED displays	8 bit PMP	SH1101A_SSD1303.c, SH1101A_SSD1303
Solomon Systech SSD1305 for OLED displays	8 bit PMP	SSD1305.c, SSD1305.h
Solomon Systech SSD2119	8/16 bit PMP	drvTFT002.c, drvTFT002.h
Sino Wealth SH1101A for OLED displays	8 bit PMP	SH1101A_SSD1303.c, SH1101A_SSD1303.h
Sitronix ST7529	8 bit PMP	ST7529.c, ST7529.h
Hitech Electronics HIT1270	8 bit PMP	HIT1270.c, HIT1270.h
Ilitek ILI9320	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Himax HX8347	8/16 bit PMP	HX8347.c, HX8347.h
UltraChip UC1610	8 bit PMP	UC1610.c, UC1610.h

- PutPixel – ustawia parametry piksela na ekranie.
- GetPixel() – zwraca kolor piksela.
- PutImage() – przetwarzanie grafiki i jej rysowanie na ekranie – działanie tej funkcji jest zależne od użytego formatu koloru.
- IsDeviceBusy() – sprawdzenie czy można do sterownika wysłać nową komendę.
- SetPalette() – ustawia zawartość rejestrów palety kolorów.

Wszystkie funkcje są dokładnie opisane w pliku pomocy. Można też obejrzeć pliki źródłowe funkcji warstwy Device driver dla wspieranych wyświetlaczy i na ich podstawie napisać własne.

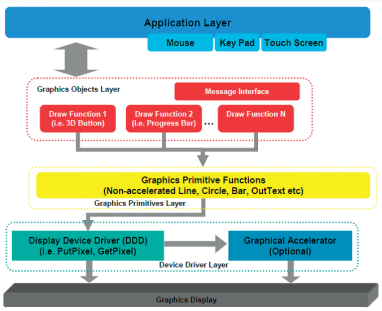
Warstwa Graphic Primitive Layer

W tej warstwie są zawarte funkcje rysowania podstawowych elementów graficznych: linii, prostokątów, okręgów, ustawianie kursora, operacji na bitmapach i definiowania gradientu wypełnienia obszaru kolorem. Bardzo ważnym elementem tej warstwy jest wyświetlanie znaków alfanumerycznych (tekstu). W pliku pomocy są opisane wszystkie funkcje warstwy *Graphic Primitive Layer*. Trzeba jednak pamiętać, że w pewnych okolicznościach mogą one nie dawać pełnej programowej kontroli nad rysowanymi elementami.

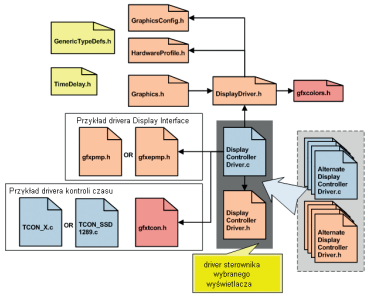
Funkcje rysowania linii

Warstwa Graphics Primitive Layer zawiera funkcje do rysowania 3 rodzajów linii:

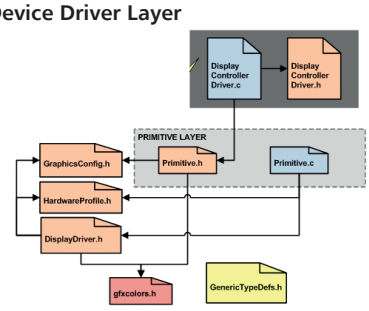
- Linii ciągłej (*SOLID_LINE*).



Rysunek 2. Model warstwowy biblioteki graficznej



Rysunek 3. Struktura plików warstwy Device Driver Layer



Rysunek 4. Struktura plików warstwy Primitive Layer

- Linii punktowej (*DOTTED_LINE*).
- Linii z odcinków (*DASHED_LINE*).

Do wybrania rodzaju linii zdefiniowano makro *SetLineType(lnType)*. Podstawowa funkcja rysująca linie, to *Line(x1, y1, x2, y2)*, gdzie:

– *x1, y1* – współrzędne początku linii (odcinka),
– *x2, y2* współrzędne końca odcinka.

Funkcja w trybie *Non Blocking* zwraca „0”, gdy rysowanie nie jest zakończone lub „1”, gdy rysowanie zakończy się. W trybie *Blocking* funkcja zawsze zwraca „1”. Linie mogą być rysowane liniami o dwóch grubościach: 1 piksel (*NORMAL_LINE*) i 3 pikseli (*THICK_LINE*). Wyboru grubości linii dokonuje się za pomocą makra *SetlineThickness(lnThickness)*. Na **listingu 2** zamieszczono fragment programu rysującego 3 linie w 3 różnych kolorach przy włączonym trybie *Blocking*.

Wynik działania procedur z listingu 2 pokazano na **fotografii 5**.

Listing 2. Rysowanie 3 linii na ekranie
```
SetLineType(4);//DASHED LINE
    SetLineThickness(1); //3 piksele
    SetColor(RGBConvert(96, 252, 0));//ustawienie koloru - Zielony
    while(!Line(  29, //x1                  //rysuj linię
                  103, //y1
                  130, //x2
                 193 //y2
                )
         );

    SetLineType(0); SOLID LINE
    SetLineThickness(1);//3 piksele
    SetColor(RGBConvert(48, 48, 248));//niebieski
    while(!Line(  151, //x1   //rysuj linię
                   31, //y1
                  151, //x2
                  172 //y2
                )
         );

    SetLineType(1); DOTTED LINE
    SetLineThickness(1);//3 piksele
    SetColor(RGBConvert(248, 0, 0));//czerwony
    while(!Line(  202, //x1   //rysuj linię
                   70, //y1
                  251, //x2
                   70   //y2
                 )
         );
```

Listing 3. Funkcje rysowania okręgów
```
SetLineType(4); // okrąg z odcinków łuków
SetLineThickness(1);
SetColor(RGBConvert(0, 0, 0));//kolor czarny
while(!Circle(  174, //x  rysowanie okręgu
                 69, //y
                 38 //radius
             )
     );

SetLineType(0);//
SetLineThickness(1);
SetColor(RGBConvert(48, 252, 48));//kolor zielony
while(!FillCircle(  192, //x  rysowanie koła
                    171, //y
                     40 //radius
                 )
     );

SetLineType(1);// okrąg z punktów
SetLineThickness(1);
SetColor(RGBConvert(0, 0, 248));
while(!Circle(  73, //x  rysowanie okręgu.
               105, //y
                25 //radius
             )
     );
```

Listing4. Wyświetlenie tekstu „Elektronika Praktyczna”
```
const XCHAR text_OTE_25text[ ] = „Elektronika Praktyczna”;

SetLineType(0);
SetLineThickness(0);
SetColor(RGBConvert(0, 0, 0));
SetFont((void*)&Calibri_24);
while(!OutTextXY(  24, //x
                   15, //x
                   (XCHAR*)text_OTE_25text //text
                 )
     );
```

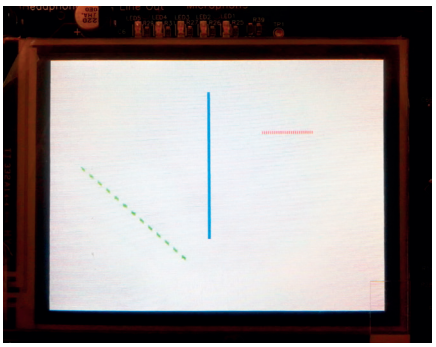
Funkcje rysowania okręgów

Okręgi można rysować trzema różnymi stylami linii: ciągłą, kropkowaną i składającą się z odcinków łuków. Dodatkowo, można wypełnić wnętrze okręgu dowolnym kolorem, czyli inaczej mówiąc narysować koło o wybranym kolorze.

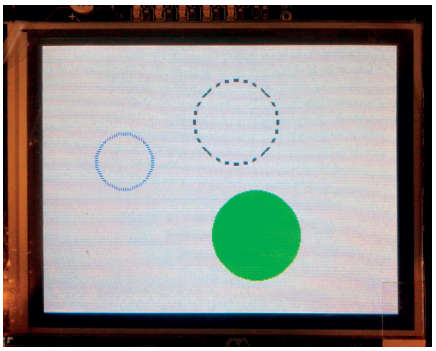
Do rysowania okręgu zdefiniowano makro *Circle(x, y, r)*, gdzie *x, y* to współrzędne środka okręgu, a *r* promień w pikselach. Do rysowania koła jest przeznaczone makro *FillCircle(x, y, r)*. Przed wysyłaniem tych makr trzeba ustawić kolor rysowania okręgu lub koła funkcją *SetColor* i styl rysowania funkcją *SetLineType* (dokładnie tak samo, jak w przypadku rysowania linii). Na **listingu 3** pokazano fragment programu rysującego 2 okręgi i koło, a na **fotografii 6** wynik jego działania.

Funkcje wyświetlania tekstu

Funkcje wyświetlania tekstu powinny być jednymi z podstawowych funkcji warstwy



Rysunek 5. Działanie funkcji rysowania linii



Rysunek 6. Wynik działania programu z listingu 3

REKLAMA

Listing 5. Struktura nagłówka
```
typedef struct {
  WORD ID;
  void * pNxtObj;
  GOL_OBJ_TYPE type;
  WORD state;
  SHORT left;
  SHORT top;
  SHORT right;
  SHORT bottom;
  GOL_SCHEME * pGolScheme;
  DRAW_FUNC DrawObj;
  FREE_FUNC FreeObj;
  MSG_FUNC MsgObj;
  MSG_DEFAULT_FUNC MsgDefaultObj;
} OBJ_HEADER;
```

Listing 6. Fragment funkcji tworzenia obiektu przycisku
```
BUTTON *BtnCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    SHORT       radius,
    WORD        state,
    void        *pBitmap,
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    BUTTON  *pB = NULL;
    pB = (BUTTON *)GFX_malloc(sizeof(BUTTON));
    if(pB == NULL)
        return (NULL);

    pB->hdr.ID = ID;                        // unique id assigned for
referencing
    pB->hdr.pNxtObj = NULL;                 // initialize pointer to NULL
    pB->hdr.type = OBJ_BUTTON;              // set object type
    pB->hdr.left = left;                    // left position
    pB->hdr.top = top;                      // top position
    pB->hdr.right = right;                  // right position
    pB->hdr.bottom = bottom;                // bottom position
    pB->radius = radius;                    // radius
    pB->pBitmap = pBitmap;                  // location of bitmap
    pB->pText = pText;                      // location of the text
    pB->hdr.state = state;                  // state
    pB->hdr.DrawObj = BtnDraw;              // draw function
    pB->hdr.MsgObj = BtnTranslateMsg;       // message function
    pB->hdr.MsgDefaultObj = BtnMsgDefault;  // default message function
    pB->hdr.FreeObj = NULL;                 // free function

    // Set the color scheme to be used
    if(pScheme == NULL)
        pB->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pB->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    pB->textWidth = 0;
    pB->textHeight = 0;
    if(pB->pText != NULL)
    {
            BtnSetText(pB, pText);
    }
```

Promitive Layer. I rzeczywiście. Jeżeli chcemy wyświetlić jakiś komunikat, to można je z powodzeniem stosować, jednak kiedy różne napisy maja się pojawiać w tej samej lokalizacji zależnie od wyniku działania programu użytkownika, to pojawiają się problemy. W takich sytuacjach dobrze sprawdza się funkcja nadpisywania (*overwrite*). Na istniejącym tekście wyświetlamy nowy.

Funkcje z warstwy *Primitive Layer* działają tak, jakbyśmy pisali ołówkiem w tym samym miejscu, bez uprzedniego wytarcia poprzedniego napisu gumką. Przyznam, że początkowo długo szukałem ustawień biblioteki pozwalających na nadpisywanie tekstu i dopiero konsultant pomocy technicznej Microchip wyjaśnił mi, że takie działanie jest normalne i inaczej nie można w tej warstwie w prosty sposób wyświetlać w jednym miejscu różnych tekstów.

Tekst jest wyświetlany za pomocą funkcji *OutText* lub *OutTextXY*. Argumentem funkcji *OutText* jest wskaźnik na bufor z łańcuchem znaków do wyświetlania. Jako argument funkcji *OutTextXY* dodatkowo są podawane 2 współrzędne początku wyświetlania znaków.

Przed wywołaniem *OutText* trzeba mieć zdefiniowana tablicę z wzorcami znaków i ustawić kolor znaków. Na **listingu 4** pokazano przykład wyświetlania tekstu „Elektronika Praktyczna”, a wynik działania zamieszczono na **rysunku 7**.

Próba ponownego wyświetlenia tekstu w tym samym miejscu powoduje zlewanie się obu napisów (**rysunek 8**). Oczywiście można temu zapobiec wymazując poprzedni tekst przez „narysowanie” na nim prostokąta o kolorze tła, a potem wysyłając na czysty obszar nowy napis. Jest to jednak niewygodne, bo wymaga precyzyjnego definiowania obszaru „kasowania” o wielkości zależnej od wielkości czcionki i długości napisu. Biblioteka oferuje inne rozwiązanie, które nie ma tych wad. Tekst można wyświetlać posługując się obiektem Static Text umieszczonym w wyższej warstwie Objest Layer. Staic Text zostanie później opisany przy okazji omawiania obiektów graficznych.

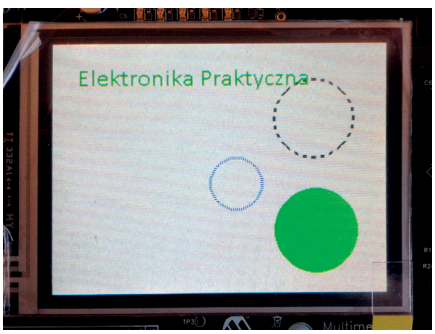
Osobnym problemem jest generowanie wzorca znaków dla funkcji wyświetlania napisów. Definiowanie różnych krojów i wielkości czcionek „na piechotę” jest zajęciem bardzo żmudnym. Istnieje narzędzie, które potrafi to zrobić za nas bardzo szybko. O nim również powiemy przy okazji omawiania warstwy obiektów.

Omówiłem większość funkcji warstwy *Primitive Layer*. Z oczywistych względów jest to opis pokazujący możliwości rysowania, a nie szczegóły działania wszystkich funkcji i makr łącznie z mechanizmami konfigurującymi działanie tej warstwy. Te szczegółowe informacje można znaleźć w pliku pomocy, który jest dołączony biblioteki.

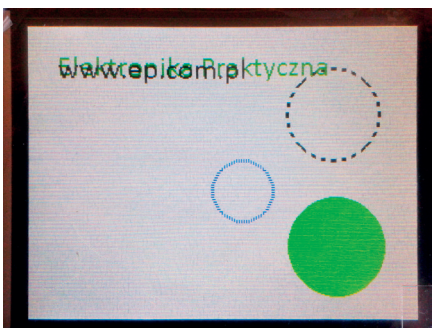
Warstwa *Object Layer*

Poprzednio opisane funkcje rysowania podstawowych elementów graficznych, łącznie z wyświetlaniem tekstu, nie są trudne do napisania dla średniozaawansowanego programisty. A jeszcze ważniejsze jest to, że ich implementacja nie jest bardzo czasochłonna, może poza ręcznym definiowaniem wzorców znaków alfanumerycznych. Ale warstwa *Pirimitive Layer* to dopiero wstęp do prawdziwych możliwości biblioteki ukrytych w warstwie obiektów *Object Layer*.

Warstwa *Object Layer* zawiera szereg bardziej lub mniej skomplikowanych elementów graficznych zwanych obiektami lub



Rysunek 7. Wyświetlenie tekstu funkcją warstwy Primitive

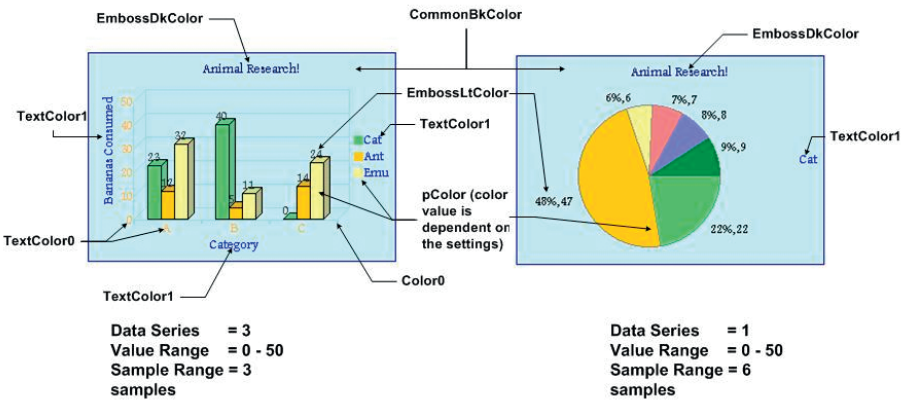


Rysunek 8. Efekt nadpisania dwu napisów

Listing 7. Utworzenie obiektu BUTTON1

```
const XCHAR text_BUTTON1[ ] = „MENU”;
BUTTON *pBUTTON1;
    pBUTTON1 = BtnCreate(  BUTTON1, //nazwa
                        135, //left
                        110, //top
                        245, //right
                        150, //bottom
                        0, //radius
                        BTN_DRAW, //state
                        NULL, //bitmap
                        (XCHAR*)text_BUTTON1, //text
                       defscheme //scheme
                      );

    if(pBUTTON1==NULL)
    {
      CreateError(0);
      while(1); //Fatal Error, trzeba sprawdzić dostępność pamięci lub
zajętość sterty
    }
```



Rysunek 9. Przykład działania obiektu Chart

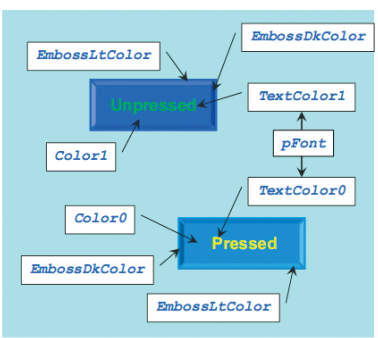
widżetami. W obecnej wersji biblioteki są dostępne następujące obiekty:

- Button – przycisk, który może być w 3 stanach: przyciśnięty, zwolniony i zablokowany. Każdy ze stanów ma swój zdefiniowany kolor w strukturze GOL_SCHEME.
- Chart – element wykorzystywany do graficznego przedstawiania wielkości w formie trójwymiarowych słupków lub okręgu podzielonego na sektory (**rysunek 9**).
- Check box – pole wyboru.
- Radio button – przyciski opcji.
- Round dial – wirtualne pokrętło.
- Digital Meter – cyfrowe wyświetlanie wartości.
- Edit Box – obiekt przeznaczony do wprowadzania tekstu.
- List box – obiekt do wybierania elementów z listy.
- Meter – obiekt przeznaczony do wyświetlania wielkości w formie skali analogowej.
- Slider/Scroll Bar – obiekt w formie „suwaka” przeznaczony do ustawiania wartości.
- Static Text – przeznaczony do wyświetlania komunikatów tekstowych.
- Text Entry – klawiatura do wprowadzania znaków alfanumerycznych.
- Window – wyświetlanie okna.

Każdy taki element ma zdefiniowany nagłówek w formie struktury OBJ_HEADER (**listing 5**).

Składowe *left*, *right*, *top* i *bottom* nagłówka OBJ_HEADER określają współrzędne pozycji obiektu i rozmiar. Bardzo ważną składową struktury nagłówka jest struktura stylu GOL_SCHEME. Funkcja *GOLCreateScheme()* tworzy i inicjalizuje tę strukturę określającą styl obiektów graficznych tworzonych w warstwie *Graphic Object Layer*. Na **rysunku 10** pokazano przykład znaczenia składowych struktury GOL_SCHEME dla obiektu przycisku *Button*.

```
typedef struct {
   WORD EmbossDkColor;
   WORD EmbossLtColor;
   WORD TextColor0;
   WORD TextColor1;
   WORD TextColorDisabled;
   WORD Color0;
   WORD Color1;
   WORD ColorDisabled;
   WORD CommonBkColor;
   char *pFont;
} GOL_SCHEME;
```



Rysunek 10. Składowe struktury stylu na przykładzie obiektu Button

Składowe EmbossDkColor i EmbossLtColor definiują kolory wykorzystywane dla uzyskania efektu przestrzennego przycisku. *Color1* jest kolorem przycisku w stanie „zwolniony”, a *Color0* w stanie „wciśnięty”. Kiedy przycisk jest w stanie zablokowanym (*disabled*), to zabarwia się na kolor zdefiniowany w składowej *TextColorDisabled*. Całkowite ukrycie przycisku jest możliwe, gdy zdefiniujemy *CommonBkColor* taki sam, jak kolor tła. Argument **pFont* jest wskaźnikiem na bufor zawierający tekst wyświetlany na przycisku. Kolor tego tekstu jest określany składowymi *TextColor0 (przycisk wciśnięty)* i *TextColor1(przycisk zwolniony)*. Składowa *DRAW_FUNC* jest wskaźnikiem do funkcji rysowanego obiektu, *FREE_FUNC* wskaźnikiem do funkcji zwalniania obiektu, a *MSG_FUNC* do funkcji wysyłania komunikatów. Przykładowy obiekt Button jest tworzony przez funkcję **BtnCreate*. Fragment tej funkcji pokazano na **listingu 6**.

Jeżeli chcemy utworzyć własny przycisk o nazwie BUTTON1, to wywołujemy tę funkcję z wymaganymi argumentami, tak jak to zostało pokazane na **listingu 7**.

Funkcja zwraca wskaźnik do obiektu, jeżeli został prawidłowo utworzony lub NULL, jeżeli z jakichś powodów utworzenie się nie udało. Na **rysunku 11** pokazano przycisk utworzony w ten sposób. Użyto tu domyślnego stylu *drfscheme* (**listing 8**). Jeżeli użytkownik chciałby zmienić styl, to nic nie stoi na przeszkodzie, aby zdefiniować własny i w argumencie funkcji *BtnCreate* podać wskaźnik do jego struktury.

Składowa *state* struktury nagłówka (listing 5) określa stan obiektu graficznego. Może on być przypisany do jednej z dwóch grup: *Property States* lub *Drawing States*. Stany z grupy *Property States* definiują wykonywane działania i wygląd obiektu. Stany z grupy *Drawing States* określają czy obiekt ma być ukryty, zmieniony częściowo lub narysowany od nowa.

REKLAMA

W bibliotece zdefiniowano pięć stanów:
- OBJ_FOCUSED – zazwyczaj używany do wskazania wybrania obiektu (jest zdefiniowany dla części obiektów).
- OBJ_DISABLED – obiekt jest zablokowany i ignoruje wszystkie komunikaty.
- OBJ-DRAW_FOCUS – wybrany obiekt musi być narysowany w jakieś części (na przykład zmieniony kolor).
- OBJ_DRAW – obiekt musi być narysowany od nowa.
- OBJ_HIDE – obiekt jest ukryty. Ten stan ma najwyższy priorytet ze wszystkich z grupy *Drawing States*.

Stany dla każdego z obiektów można ustawiać makrem *SetState*. Jako argumenty są używane: wskaźnik do struktury nagłówka obiektu i wartość bitów stanu. Opis znaczenia poszczególnych bitów stanu można znaleźć w pliku pomocy biblioteki.

Do zarządzania obiektami wykorzystywanych jest szereg funkcji API i zdefiniowanych makr. Trudno tu wymieniać wszystkie, bo jest ich sporo. Zajmę się kilkoma ważniejszymi, a opisy reszty można znaleźć w pliku pomocy:
- Funkcja *GOLInit()* inicjuje bibliotekę i definiuje domyślny styl (*style scheme*) z domyślnymi ustawieniami kolorów. *GOLInit()* musi być wywołana przed wywołaniem jakichkolwiek funkcji bibliotecznych.
- Funkcja *GOLDraw()* przeszukuje listę aktywnych obiektów i jeżeli natrafi na stan obiektu graficznego wymagający rysowania to wykonuje to rysowanie.
- *GOLDrawCallback()* jest funkcją definiowana przez użytkownika i jest wywoływana zawsze po kompletnym narysowaniu obiektu.
- Funkcja *GOLAddObject()* dodaje utworzony obiekt do listy aktywnych obiektów. Argumentem jest wskaźnik do struktury nagłówka.
- Funkcja *GOLFree()* zwalnia pamięć zarezerwowaną dla używanych obiektów, zapisuje do wskaźnika obiektów w liście zero i startuje z pustą listą obiektów.
- Funkcja GOLDeleteObject kasuje obiekt z listy obiektów bieżącego ekranu.

Interfejs Message Interface

Rysowanie obiektów na nie jest jedyna czynnością wykonywaną przez funkcje biblioteczne. Graficzny interfejs użytkownika ma zapewnić interakcję pomiędzy użytkownikiem, a sterowanym urządzeniem. Załóżmy, że na ekranie mamy narysowany przycisk Button, taki jak na rys. 12. Taki obiekt ma sens, jeśli aplikacja może określić, czy przycisk jest naciśniety, lub zwolniony. Jeżeli wyświetlacz ma wbudowany panel dotykowy, to można bezpośrednio połączyć dotknięcie/puszczenie obszaru, na którym jest narysowany przycisk z podjęciem odpowiedniej akcji przez aplikację. Zapewnia to *Message Interface*. Działanie tego mechanizmu (**rysunek 12**) wygląda następująco:

Listing 8. Definicja domyślnego stylu
```
GOL_SCHEME* defscheme;
    defscheme->Color0 = RGBConvert(32, 168, 224);
    defscheme->Color1 = RGBConvert(16, 132, 168);
    defscheme->TextColor0 = RGBConvert(24, 24, 24);
    defscheme->TextColor1 = RGBConvert(248, 252, 248);
    defscheme->EmbossDkColor = RGBConvert(248, 204, 0);
    defscheme->EmbossLtColor = RGBConvert(24, 116, 184);
    defscheme->TextColorDisabled = RGBConvert(128, 128, 128);
    defscheme->ColorDisabled = RGBConvert(208, 224, 240);
    defscheme->CommonBkColor = RGBConvert(208, 236, 240);
    defscheme->pFont = (void*)&Gentium_16;
```

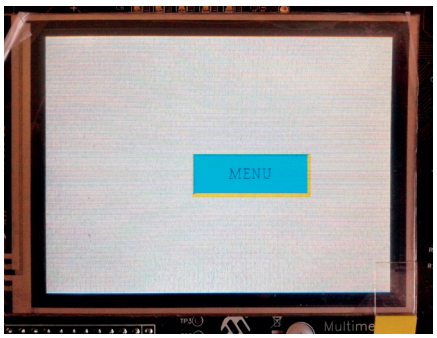
- Interfejs użytkownika (ekran dotykowy, klawiatura) wysyła *GOL Message* za pośrednictwem wpisu do struktury *GOL_MSG* (**listing 9**). Na list. 9:
  - *Type* – definiuje typ urządzenia, w którym wygenerowane wiadomość (*message*). Możliwe typy urządzeń, to: *TYPE_UNKNOW*, *TYPE_KEYBOARD*, *TYPE_TOUCHSCREEN* i *TYPE_MOUSE*.
  - *uiEvent* – rodzaj zdarzenia generowanego przez obiekt: *EVENT_INVALID* (zdarzenie błędne), *EVENT_MOVE* (przesunięcie), *EVENT_PRESS* (naciśnięcie), *EVENT_STILLPRESS* (ciągłe naciskanie), *EVENT_RELEASE* (zwolnienie), *EVENT_KEYSCAN* (skanowanie stanów klawiszy), *EVENT_CHARCODE* (kod znaku).
  - Definicja składowych *param1* i *param2* jest zależna od typu urządzenia. Na przykład dla ekranów dotykowych *TYPE_TOUCHSCREEN* zawierają współrzędne *x, y*, a dla typu *TYPE_KEYBOARD param1* zawiera ID obiektu, a *param2* znak kodu przycisku.
- Wewnątrz funkcji *GOLMsg* ze wskaźnikiem do struktury *GOL_MSG* jako argumentem (**listing 10**) jest wykonywana pętla wykrywająca obiekt generujący wiadomość.
- Wykryty obiekt zwraca przekonwertowaną wiadomość w oparciu o parametry *GOL Messager*.
- Użytkownik może zmienić domyślną akcję zdefiniowaną przez bibliotekę w funkcji *GOLMsgCallBack*.
- Po wykonaniu akcji obiekt powinien być narysowany ponownie w zależności od swojego nowego stanu.

Na **listingu 11** pokazano pętlę, w której jest wywoływana funkcja *GOLGraw* sprawdzająca stany obiektów i ewentualnie ry-

Listing 9 struktura GOL_MSG
```
typedef struct {
    BYTE  type;
    BYTE  uiEvent;
    int   param1;
    int   param2;
} GOL_MSG;
```

Listing 10. Funkcja GOLMsg
```
void GOLMsg(GOL_MSG *pMsg)
{
    OBJ_HEADER  *pCurrentObj;
    WORD        translatedMsg;
    if(pMsg->uiEvent == EVENT_INVALID)
        return;
    pCurrentObj = _pGolObjects;
    while(pCurrentObj != NULL)
    {
        if(pCurrentObj->MsgObj)
        {
            translatedMsg = pCurrentObj->MsgObj(pCurrentObj, pMsg);
            if(translatedMsg != OBJ_MSG_INVALID)
            {
                if(GOLMsgCallback(translatedMsg, pCurrentObj, pMsg))
                    if(pCurrentObj->MsgDefaultObj)
                        pCurrentObj->MsgDefaultObj(translatedMsg, pCurrentObj,
pMsg);
            }
        }
        pCurrentObj = (OBJ_HEADER *)pCurrentObj->pNxtObj;
    }
}
```

Listing 11. Pętla obsługi graficznego interfejsu użytkownika
```
while(1)
{
    if(GOLDraw())  // Rysuje obiekt z warstwy Obejct layer (GOL)
    {
        TouchGetMsg(&msg);  //odbierz informacje (message) z panelu dotykowego
        GOLMsg(&msg);
    }
}//end while
```



Rysunek 11. Utworzony obiekt BUTTON1

sująca je. Funkcja sprawdza czy z drivera ekranu dotykowego nie została przesłana informacja (*message*) o niewykonanej akcji (funkcja *TouchGetMsg*) i na końcu, na podstawie tej informacji, funkcja *GOLMsg* interpretuje tę wiadomość i przekazuje sterowanie do funkcji *GOLMsgCallback* (**listing 12**). Przed wywołaniem tej pętli wszystkie używane obiekty graficzne muszą być utworzone.

Pobranie informacji o wykonanej akcji przez użytkownika z wykorzystaniem ekranu dotykowego wykonuje funkcja *TouchGetMsg* (**listing 13**). Zależnie od odczytanych współrzędnych *x, y* oraz poprzednio odczytanych współrzędnych *prevX* i *prevY* jest modyfikowana składowa *uiEvent* struktury *GOL_MSG*.

Projekt MPALB X – wtyczka Graphic Display Designer X

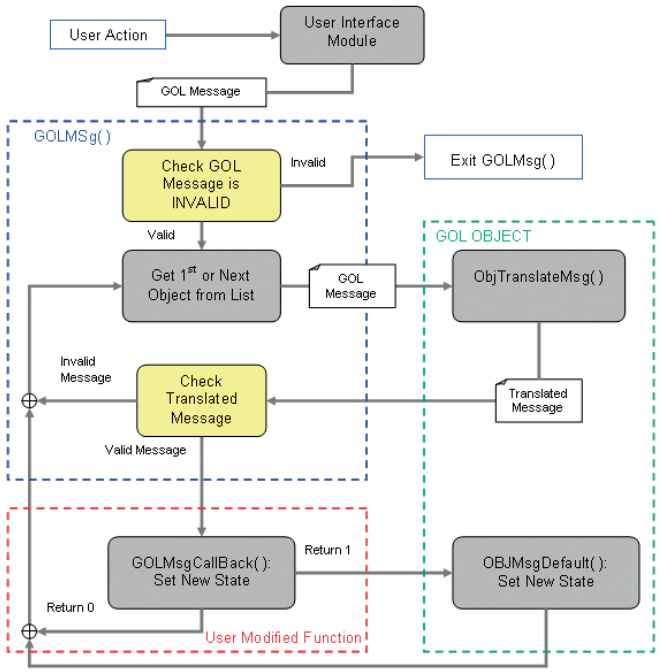
Do tej pory przedstawiłem wybrane przez siebie właściwości biblioteki graficznej. Oczywiście wiele szczegółów zostało pominiętych, bo ze względu na obszerność tematu trudno byłoby zmieścić większość w artykule.

Bibliotekę można pobrać ze strony *www.microchip.com*. Ostatnia wersja, dostępna w momencie pisania artykułu, to V2013-06-15. Biblioteka domyślnie się instaluje w katalogu *\microchip_solutions_v2013-06-15*. W katalogu *\microchip_solutions_v2013-06-15\Microchip\Help* jest umieszczony plik pomocy *Graphic Library Help*, w którym można znaleźć kompletny opis wszystkich warstw, obiektów, funkcji, makr oraz informacje dodatkowe, niezbędne do swobodnego używania biblioteki. W katalogu *\microchip_solutions_v2013-06-15\Graphics* zostało umieszczonych szereg programów przykładowych, oczywiście z kompletnymi plikami źródłowymi, przeznaczonych do uruchomienia z wykorzystaniem firmowych modułów ewaluacyjnych.

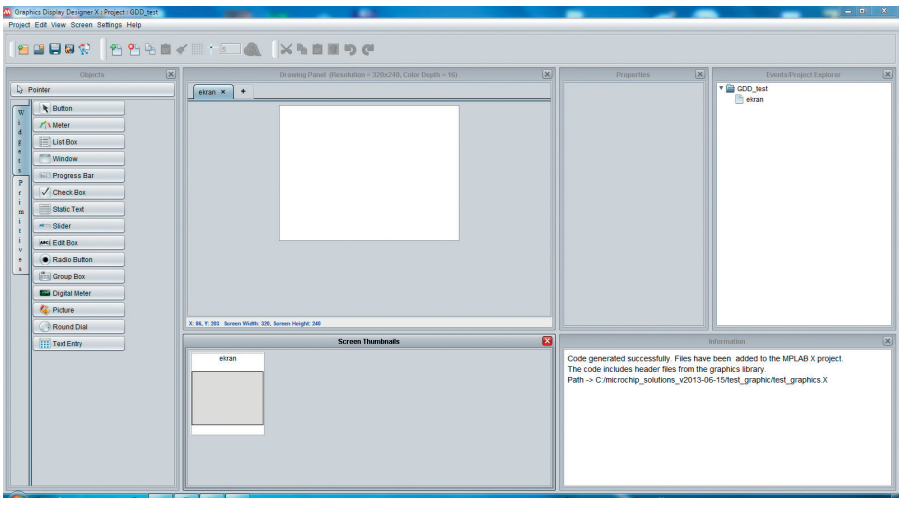
Jeżeli chcemy użyć biblioteki w swoim projekcie, to trzeba go umieścić we własnym katalogu umieszczonym w głównym katalogu biblioteki – może to być na przykład *\ microchip_solutions_v2013-06-15\ art_graf*. Nazwa katalogu i projektu może być dowolna. Wszystkie pliki biblioteczne są umieszczone w *\microchip_solutions_v2013-06-15\Microchip\Graphics* i *\microchip_solutions_v2013-06-15\Microchip\Include\Graphics*. W katalogu Graphics są umieszczone wszystkie pliki źródłowe.

W pliku pomocy, w części „*Miscellaneous Topics*", w rozdziale „*Starting a New Project*" jest dokładnie opisany sposób tworzenia nowego projektu i dołączanie do niego wszystkich plików źródłowych oraz konfigurowanie ścieżek dostępu do plików nagłówkowych. Na samym końcu tego rozdziału opisano sposób wstępnego konfigurowania projektu i biblioteki.

Taki sposób postępowania: ręczne tworzenie projektu, dodawanie do niego plików źródłowych i potem pisanie własnych procedur obsługi wyświetlacza z wykorzystaniem funkcji bibliotecznych jest dobrym sposobem na stworzenie własnego efektownego interfejsu graficznego. Jednak nawet przy tak dobrze przygotowanym wsparciu jest to dalej zadanie pracochłonne. Oczywiście, nie tak, jak napisanie wszystkiego samodzielnie, ale jednak trzeba poświęcić trochę czasu na poznanie funkcji API i zasad ich stosowania. Inżynierowie Microchipa pomyśleli również o tych, którzy nie chcą lub nie mogą poświęcić tego dodatko-

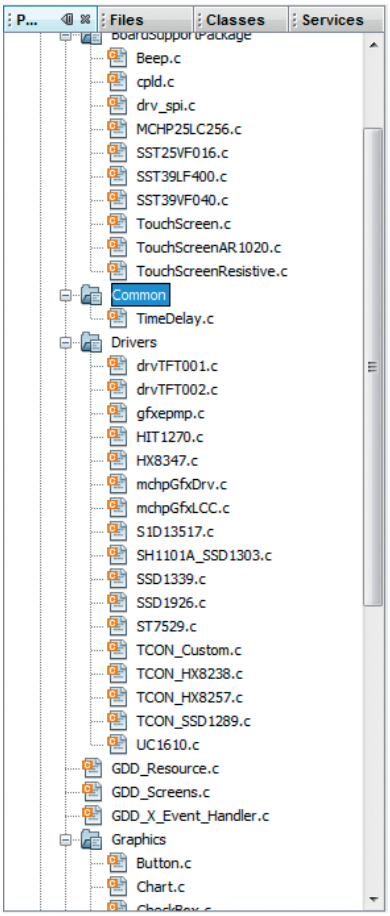


Rysunek 12. Działanie Message Interface



Rysunek 13. Ekran główny GDD-X

REKLAMA

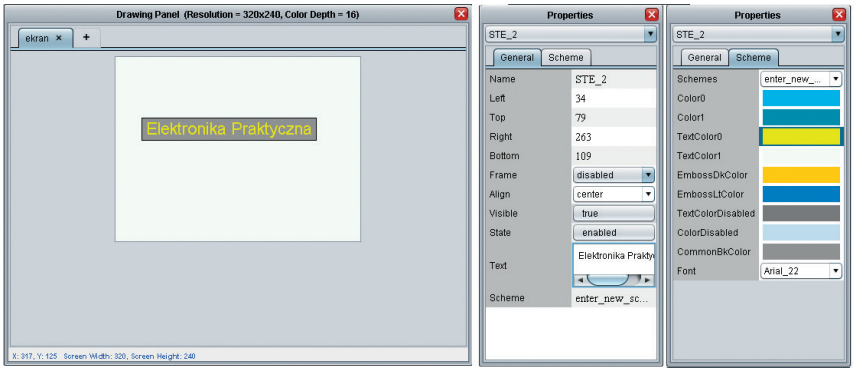


Rysunek 14. Fragment okna Project z plikami dołączonymi przez GDD-X

wego czasu i zaoferowali im moim zdaniem rewelacyjne narzędzie *Graphics Display Designer X – GDD X* (wersja X współpracuje z MPLAB X).

Graphics Dispaly Designer jest programem narzędziowym, który umożliwia programiście bardzo szybkie zaprojektowanie graficznego interfejsu użytkownika z wykorzystaniem elementów warstw *Primitives Layer* i *Object Layer*. GDD może być uruchamiany jako niezależny program lub wtyczka dla pakietów MPALB v8.x lub MPLAB-X. GDD pozwala na szybkie i łatwe projektowanie tzw. ekranów z umieszczonymi na nich elementami graficznymi, ale też wspiera interakcję pomiędzy tymi elementami oraz edycję własnego kodu. Wynik działania GDD X pracującego jako wtyczka (plug-in) pakietu MPLAB-X jest automatycznie zapisywany w plikach źródłowych otwartego projektu. Można po modyfikacjach natychmiast go skompilować i wynik przesłać do mikrokontrolera.

Żeby pokazać jak GDD X współpracuje z MPLAB-X utworzymy nowy projekt. Jak wiemy ten projekt musi być umieszczony w katalogu głównym biblioteki. Do testów zostanie użyty moduł *Multimedia Expansion Board* (wyświetlacz 320×240, sterownik Solomon Systech SSD1926) z modułem *PIC32 USB Starter KIT II* i mikrokontrolerem PIC32MX795F512L. Projekt jest kompilowany kompilatorem XC32 V1.30.



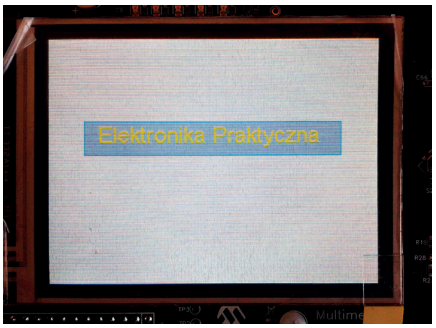
Rysunek 15. Element Static Text z oknem właściwości

Wtyczka GDD nie jest dołączana do pakietu instalacyjnego MPALB-X – trzeba ją pobrać i zainstalować. Wykonuje się to z poziomu MPLAB-X wybierając *Tools* → *Plugins* → *Available Plugins* → *Graphics Display Designer X*. Jeżeli mamy połączenie z siecią, to wtyczka zostanie automatycznie pobrana i zainstalowana. Po zainstalowaniu wtyczkę można uruchomić z menu *Tools* → *Embedded* → *Graphics Display Designer X*. Podobnie jak w przypadku MPLAB-X, również GDD-X pracuje w oparciu o utworzony projekt z menu *Projekt* → *New Project*. W pierwszym kroku tworzenia nowego projektu nadajemy mu dowolną nazwę. Ja do celów testowych nazwałem projekt *GDD_test*. Potem kreator projektu pyta o właściwości wyświetlacza: rozmiary matrycy i głębię kolorów (w bitach). Dla modułu MEB będzie to wyświetlacz 320×240 pikseli z 16-bitową głębią koloru. Tak zdefiniowanemu ekranowi głównemu nadajemy nazwę na przykład „ekran” i kreator kończy swoje działanie.

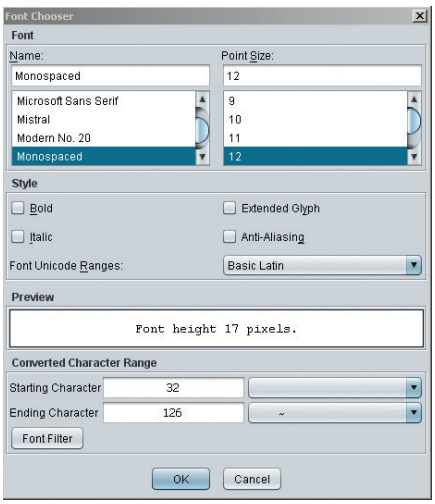
Zrzut ekranu głównego programu GDD-X po utworzeniu projektu został pokazany na **rysunku 13**. Po utworzeniu projektu w MPLAB-X nie są do niego dołączone żadne pliki źródłowe. Możemy je dołączyć ręcznie, tak jak to zostało opisane w pliku pomocy lub zrobi to za nas GDD-X po kliknięciu na ikonę *Generate code* (*Project* → *Generate Code*). Nawet gdy projekt GDD-X nie zawiera żadnych elementów graficznych poza zdefiniowanym ekranem głównym, to i tak po kliknięciu na *Generate code* otwarty projekt w MPLAB-X, w którym uruchomiono GDD-X, zostanie uzupełniony o wszystkie potrzebne pliki biblioteki graficznej (**rysunek 14**).

Kolor ekranu można dowolnie ustawi ć klikając na niego prawym przyciskiem w oknie *Drawing Panel*. Można również dodawać kolejne ekrany po kliknięciu na belkę oznaczoną „+”.

Praca z GDD X jest łatwa i intuicyjna. Z lewej strony okna głównego jest umieszczone okno obiektów *Objects* z dwoma zakładkami *Widgets* (elementy z warstwy obiektów, czyli widżety) oraz zakładka *Primitives* z elementami graficznymi z warstwy *Primitives*. Jeżeli chcemy umieścić jakiś obiekt na ekranie, to po prostu klikamy na jeden z listy w oknie *Objects*, a potem klikamy w obszarze ekranu element tam umieszczany. Po umieszczeniu można go dowolnie przesuwać w obszarze ekranu i zmieniać jego wielkość. Z każdym obiektem umieszczonym w projekcie na ekranie jest związane okno właściwości *Properties* oraz okno informacji *Informations*.



Rysunek 16. Element Static Text na ekranie wyświetlacza



Rysunek 17. Wybór czcionki z opcjami

Wyświetlanie tekstu i bitmap

Wyświetlanie tekstu to jedno z głównych zadań wykonywanych przez interfejs użytkownika. O tym, jak można wyświetlać tekst i o problemach z tym zwianych, pisałem przy okazji omawiania funkcji warstwy *Pirimitives*. Warstwa obiektów zawiera element *Static Text*, którego użycie nie powo-

Listing 12. Funkcja GOLMsgCallback
```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg)
{
    WORD    objectID;
    objectID = GetObjID(pObj);
    GDDDemoGOLMsgCallback(objMsg, pObj, pMsg);  //funkcja obsługi zdarzeń
z obiektów
    // Tutaj można dodać dodatkowy kod użytkownika ...
    return (1);
}
```

duje takich problemów, jak w przy użyciu funkcji warstwy *Primitives Layer*. *Static Text* wyświetla tekst w ramce o definiowanych wymiarach i kolorze.

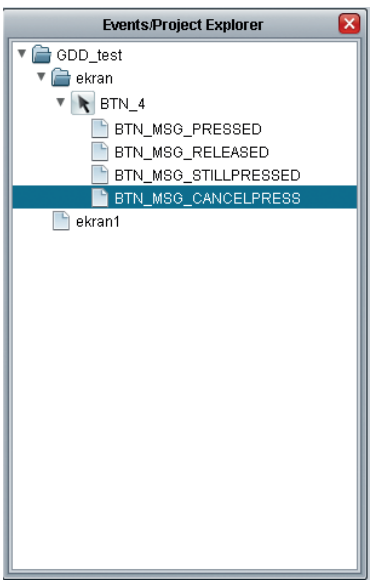
Z listy okna obiektów wybieramy element *Static Text* i umieszczamy go na ekranie. Domyślnie jest wyświetlany komunikat „*Static Text*". W oknie *Properties*, w zakładce *General* możemy zmienić wyświetlany napis w okienku *Text*. Można tu też zmienić wyrównanie w obszarze okna tekstu (wypośrodkowane, wyrównane do lewej i wyrównane do prawej), wyświetlanie ramki i status (widzialny, niewidzialny). W zakładce stylu obiektu (*Scheme*) można zmieniać kolory tekstu oraz kolory tła ramki (*background*), jak pokazano na **rysunku 15**. W oknie *Information*, w zakładce *Scheme* dokładnie opisano znaczenie poszczególnych składowych stylu.

Jeżeli chcemy zmienić wartości domyślne stylu, to program poprosi nas o podanie nowej nazwy stylu (okno *Create New Scheme*) i zmienione wartości zostaną zapisane

Listing 13. Funkcja odczytu wiadomości (message) z interfejsu ekranu dotykowego
```
void TouchGetMsg(GOL_MSG *pMsg)
{
    static SHORT    prevX = -1;
    static SHORT    prevY = -1;
    SHORT           x, y;
    x = TouchGetX();
    y = TouchGetY();
    pMsg->type = TYPE_TOUCHSCREEN;
    pMsg->uiEvent = EVENT_INVALID;
    if((x == -1) || (y == -1))
    {
        y = -1;
        x = -1;
    }
    if((prevX == x) && (prevY == y) && (x != -1) && (y != -1))
    {
        pMsg->uiEvent = EVENT_STILLPRESS;
        pMsg->param1 = x;
        pMsg->param2 = y;
        return;
    }
    if((prevX != -1) || (prevY != -1))
    {
        if((x != -1) && (y != -1))
        {
            // Move
            pMsg->uiEvent = EVENT_MOVE;
        }
        else
        {
            // Released
            pMsg->uiEvent = EVENT_RELEASE;
            pMsg->param1 = prevX;
            pMsg->param2 = prevY;
            prevX = x;
            prevY = y;
            return;
        }
    }
    else
    {
        if((x != -1) && (y != -1))
        {
            // Pressed
            pMsg->uiEvent = EVENT_PRESS;
        }
        else
        {
            // No message
            pMsg->uiEvent = EVENT_INVALID;
        }
    }
    pMsg->param1 = x;
    pMsg->param2 = y;
    prevX = x;
    prevY = y;
}
```

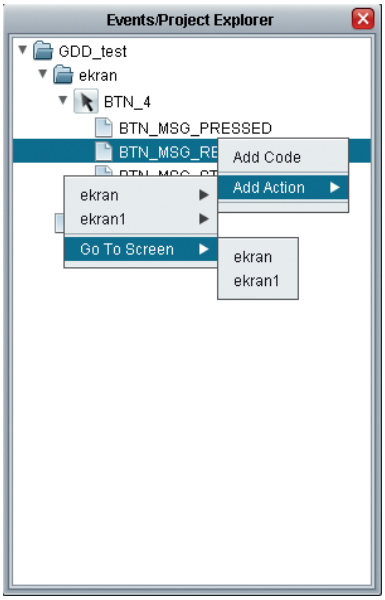
Rysunek 18. Wyświetlanie bitmapy

pod tą nazwą. Olbrzymim ułatwieniem jest możliwość wyboru czcionek zdefiniowanych w systemie Windows z rozmiarami od 8 do 72. Każdy, kto używał wyświetlaczy graficznych wie jak dużo pracy pochłania ręczne generowanie tablicy wzorca znaków. Dlatego powstało sporo dobrych programów robiących to automatycznie, ale w większości są to wersje płatne lub o bardzo ograniczonych możliwościach. Tu mamy wszystko za darmo. Trzeba jednak pamiętać, że każdy nowozdefiniowany zestaw czcionek znacząco zapełnia pamięć programu mikrokontrolera – szczególnie dla dużych znaków. Żeby to ograniczyć,



Rysunek 19. Okno Events/Project Explorer

REKLAMA



Rysunek 20. Wybór akcji przypisanej do zdarzenia

można wybrać części znaków poprzez wybranie zakresu (*starting, ending character*) lub przez wybór filtru (**rysunek 17**). Filtrem jest plik „.txt" z wybranymi znakami, na przykład z samymi czcionkami cyfr.

Wyświetlanie bitmap również łączy się koniecznością konwertowania na tablicę zapisaną w języku C lub w asemblerze. Do tego celu są niezbędne programy konwertujące. GDD-X rozwiązuje problem bitmap – z listy elementów wybieramy *Picture* i stawiamy go na ekranie. W oknie *Properties* klikamy na belkę „...". W okienku *Bitmap* i otwiera się okno eksploratora wyszukiwania plików. W obecnej wersji GDD-X mogą to być wyłącznie pliki z rozszerzeniem „.bmp". Ja przygotowałem sobie bitmapę z logo Elektroniki Praktycznej o długości nieprzekraczającej 320 pikseli. Skalowanie bitmap można wykonać w wielu programach graficznych. Wystarczający do tego jest np. Microsoft Office Picture Manager. Na **rysunku 18** pokazano wyświetloną bitmapę, łącznie z wcześniej zdefiniowanym elementem *Static Text*.

Interakcja z Użytkownikiem

Rysowanie tekstu i bitmap, to dopiero początek możliwości GDD-X. Zajmiemy się teraz przykładowymi elementami, które są przeznaczone do interakcji z użytkownikiem.

Oprócz ekranu wyświetlacza podstawowym elementem interfejsu użytkownika są elementy manipulacyjne: klawiatura, enkodery obrotowe (impulsatory), myszka itp. Wprowadzenie ekranów dotykowych znacznie uprościło implementację manipulatorów jednocześnie podnosząc atrakcyjność interfejsu. Podstawowym elementem manipulacyjnym jest przycisk *Button*. Jak wiemy, interfejs *Message Interface* korzysta ze struktury *GOL_MSG*, jednym z jej składników jest *uiEvent*. Jest w nim zapisane zdarzenie generowane przez element graficzny (taki, który może generować zdarzenie). Działanie interfejsu *Message Interface* najlepiej pokazać na przykładzie przycisku. Element *Button* umieszczony na ekranie wyświetlacza z interfejsem dotykowym może być w stanie: naciśnięty (*PRESSED*), nadal naciśnięty (*STILLPRESSED*), zwolniony (*RELEASED*) i skasowany (*CANCELPRESSED*).

Na naszym ekranie „ekran" ustawiamy element *Button* i w oknie *Properties* zmieniamy w polu *Text* napis z *Button* na *Ekran1*. Następnie, klikamy na zakładkę „+" w panelu *Drawing Panel* i dodajemy nowy ekran o nazwie *„ekran1"*. Interfejs elementów graficznych *Message Interface* jest obsługiwany w okienku *Events/Project Explorer*. Są tam umieszczone nazwy ekranów i identyfikatory obiektów umieszczanych na poszczególnych ekranach, łączenie z możliwymi zdarzeniami (**rysunek 19**). Jak widać, są tu umieszczone identyfikatory dwóch ekranów (*ekran* i *ekran1*) oraz identyfikator *BTN_4* przycisku *Button*. Dla przycisku są zdefiniowane cztery zdarzenia, zgodnie z tym, co powiedzieliśmy sobie przed chwilą. Jeżeli teraz klikniemy prawym przyciskiem myszki na opis zdarzenia, to możemy wybrać dwa działania: dodaj akcję (*Add Action*) i dodaj kod (*Add Code*).

Pierwsze działanie polega na dodaniu akcji wykonywanej po zaistnieniu zdarzenia. Dostępne akcje są automatycz-

Listing 14. Przejście do ekranu ekran1 po zwolnieniu przycisku BTN_4

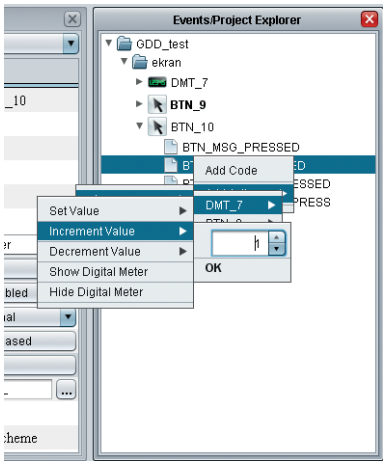
```
// <START_ID_BTN_MSG_RELEASED_(BTN_4)>
if(objMsg == BTN_MSG_RELEASED && pObj->ID == (BTN_4) )
{
    GDDDemoGoToScreen(1);
    // Add code
}
// <END_ID_BTN_MSG_RELEASED_(BTN_4)>
```

Listing 15. Obsługa modyfikacji I wyświetlania wartosci przez obiekt Digital Meter

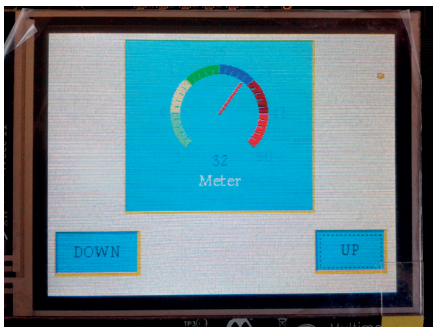
```
// <START_ID_BTN_MSG_RELEASED_(BTN_9)>
if(objMsg == BTN_MSG_RELEASED && pObj->ID == (BTN_9) )
{
    DmDecVal(((DIGITALMETER*)(GOLFindObject(DMT_7))), (SHORT)1);
    SetState(((DIGITALMETER*)(GOLFindObject(DMT_7))), DM_UPDATE);
}
// <END_ID_BTN_MSG_RELEASED_(BTN_9)>
// <START_ID_BTN_MSG_RELEASED_(BTN_10)>
if(objMsg == BTN_MSG_RELEASED && pObj->ID == (BTN_10) )
{
    DmIncVal(((DIGITALMETER*)(GOLFindObject(DMT_7))), (SHORT)1);
    SetState(((DIGITALMETER*)(GOLFindObject(DMT_7))), DM_UPDATE);
}
// <END_ID_BTN_MSG_RELEASED_(BTN_10)>
```

Listing 16. Modyfikacja wskazań obiektu Meter

```
if(objMsg == BTN_MSG_RELEASED && pObj->ID == (BTN_9) )
{
    MtrDecVal(((METER*)(GOLFindObject(MTR_6))), (SHORT)1);
    SetState(((METER*)(GOLFindObject(MTR_6))), MTR_DRAW_UPDATE);
}
// <END_ID_BTN_MSG_RELEASED_(BTN_9)>
// <START_ID_BTN_MSG_RELEASED_(BTN_10)>
if(objMsg == BTN_MSG_RELEASED && pObj->ID == (BTN_10) )
{
    MtrIncVal(((METER*)(GOLFindObject(MTR_6))), (SHORT)1);
    SetState(((METER*)(GOLFindObject(MTR_6))), MTR_DRAW_UPDATE);
}
```



Rysunek 21. Dodanie akcji inkrementacji wyświetlanej wartości o 1



Rysunek 22. Użycie obiektu Digital Meter

nie określane przez GDD-X na podstawie właściwości elementów umieszczonych w oknie z rys. 19. Na **rysunku 20** pokazano możliwość wyboru akcji przy kliknięciu na zdarzenie *RELEASE*. Można wybrać jakiś element z ekranu *ekran* lub sam *ekran*, albo wykonać skok do ekranu *ekran1*. Ponieważ na ekranach nie ma elementów, którymi można byłoby manipulować po naciśnięciu i zwolnieniu przycisku, to wybierzemy akcję *Go To Screen1 → ekran1*. Po takim zdefiniowaniu akcji *BTN_4 RELEASED* program powinien przejść do ekranu *ekran1*, kiedy naciśniemy i zwolnimy przycisk nazwany „*ekran1*". Nie dość, że GDD-X sam skonfiguruje wykonanie akcji po zdarzeniu, to jeszcze automatycznie uaktualni plik źródłowy w projekcie MPLAB-X. Obsługa zdarzeń jest wykonywana w funkcji *GDDDemoGOLMsgCallback* umieszczonej w pliku *GDD_X_Event_Handler.c*. Ta funkcja jest z kolei wywoływana z funkcji *GOLMsgCallback*. Obie są automatycznie generowane przez GDD-X. Na list. 13 pokazano fragment akcji przejścia do ekranu *ekran1* po zwolnieniu przycisku *BTN_4*.

Oczywiście zamiast wywoływania funkcji GDDDemoGoToScreen(1) można tutaj wywołać własną funkcję użytkownika i wykonywać działania przypisane do naciśnięcia i zwolnienia przycisku. Wsparcie GDD-X i możliwości własnej konfiguracji są tutaj bardzo pomocne. Podobnie działają pozostałe elementy manipulacyjne typu *Round Dial*, *List Box* czy *Check box*.

Następną ważną grupą są elementy wyświetlające wartości w postaci cyfrowej, lub analogowej. Do wyświetlania wartości w postaci cyfrowej jest przeznaczony element *Digital Meter*. Po postawieniu na ekranie GDD-X nadał mu identyfikator *DMT_7*. W oknie właściwości *Properties* można ustawić styl obiektu (*Scheme*). Rozdzielczość wyświetlanej danej ustawia się przez modyfikację zmiennych *Noofdigits* (liczba wszystkich cyfr) i *Dotpos* (liczba cyfr po przecinku).

Modyfikacja wyświetlanej wartości odbywa się z wykorzystaniem funkcji i makr przypisanych do obiektu *Digital Meter*. Funkcja *DmSetValue* ustawia warto wyświetlanej zmiennej. Jej argumentami są: wskaźnik do modyfikowanego obiektu i nowa wartość. Uzupełnieniem tej funkcji są dwa makra wykonujące inkrementacje i dekrementację wyświetlanej wartości o określoną w argumencie wartość. Zdefiniujmy dodatkowe dwa przyciski i nazwijmy je *UP* i *DOWN*. Za pomocą GDD-X można napisać program, który będzie modyfikował i wyświetlał wartość w postaci cyfrowej. W oknie *Events/Projects Explorer* klikamy na identyfikator *DMT_7* i wykonujemy *Add Code*. Powoduje to uaktywnienie obiektu. Teraz dla przycisku nazwanego *UP* definiujemy akcję inkrementacji wyświetlanej wartości o 1 przy każdym przyciśnięciu i zwolnieniu. Klikamy prawym przyciskiem myszy na *BTN_10->BTN_MSG_RELEASED*, wybieramy *Add Action*. Potem wybieramy element *Digital Meter DMT_7* z ekranu „*ekran*", a następnie akcję *Increment Value* i na koniec wartość inkrementacji (tutaj o 1). Zostało to pokazane na **rysunku 21**. Podobne czynności wykonujemy dla przycisku *BTN_9* z tym, że tam definiujemy dekrementację wartości o 1. Po tych definicjach GDD X wygeneruje kod obsługi pokazany na **listingu 15**.

Inkrementację i dekrementację wykonują makra *DmDrcVal* i *DmIncVal* obiektu *Digitalmeter*. Nowa wartość jest wyświetlana po wykonaniu opisywanego już makra *SetState*. Wywołanie tego makra z argumentem *DM_UPDATE* informuje aplikację, że w obiekcie *Digital Meter* trzeba na nowo narysować tylko element *text*. Wszystkie stany obiektu *Digital Meter* są dokładnie opisane w pliku pomocy.

Na **rysunku 22** pokazano ekran z opisywanym przykładem. Żeby wyświetlana wartość była dobrze widoczna, wybrałem w definicji stylu pogrubioną czcionkę Arial o wielkości 24, dla której są zdefiniowane wzorce znaków tylko dla cyfr 0…9. Teraz zmienimy obiekt *Digital Meter* na analogowo/cyfrowy miernik *Meter* – **rysunek 23**. Ten obiekt pokazuje wartość w postaci analogowej (wskazówka na skali) oraz w postaci cyfrowej u dołu skali. W zakładce *Properties → General* ustawiamy parametr *Minval=1* i *Maxval=50*. Wyświetlane wartości będą się mogły zmieniać w zakresie 1…50 z krokiem co 1. Na **listingu 16** pokazano fragment programu obsługującego zmianę wyświetlanej wartości przyciskami *UP* i *DOWN*.

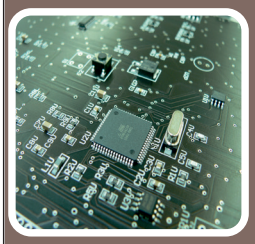
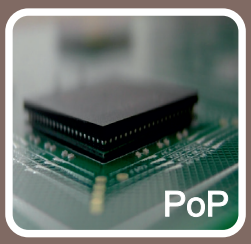
Podsumowanie

Trudno byłoby opisać szczegóły wszystkich możliwości biblioteki i programu *Graphics Display Designer X*, bo byłby to materiał na sporą książkę. Ale nawet tych kilka przykładów pokazuje olbrzymie możliwości samej biblioteki i rewelacyjnego wsparcia w postaci wtyczki GDD-X. Powoduje to znakomite skrócenie czasu przygotowania interfejsu i daje możliwość szybkiego modyfikowania projektów. Jeżeli do tego dodamy dobrze przygotowany plik pomocy, to dostajemy idealne narzędzie do tworzenia atrakcyjnych interfejsów użytkownika.

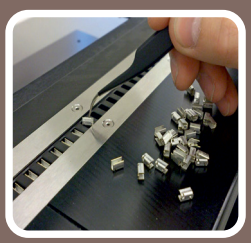
Tomasz Jabłoński, EP

MONTAŻ PCB

NOWA LINIA PRODUKCYJNA SMT:
Pick & Place - JUKI KE-3020VE-XL
- max wymiary PCB 500 x 800 mm
- zmieniacz tacek, fluxer do montażu PoP
Sitodrukarka EKRA X5-36
- pole zadruku do 610 x 915 mm
- wbudowany dispenser
Montaż SMT na giętkich płytkach drukowanych (PCB-flex)
Montaż PoP

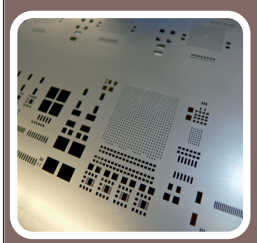
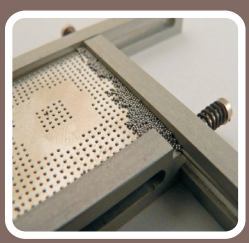



T&R Pakowanie elementów SMD w taśmę



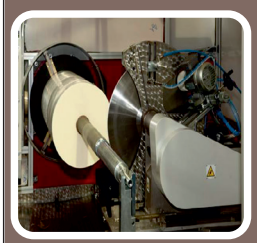
SZABLONY SMT WYCINANE LASEROWO

Szablony w ramach aluminiowych
Szablony stopniowane
Szablony w ramach typu Vector Guard
Mikroszablony do napraw BGA, QFN, QFP

USŁUGA CIĘCIA TAŚM PRZEMYSŁOWYCH

Cięcie logroli na rolki o żądanej szerokości
Min szerokość taśmy 2mm, dokładność cięcia 0,1mm
Max wymiary ciętej logroli - Ø 460 x 1700 mm




TARGI AUTOMATICON
25 - 28 marca 2014
Zapraszamy na nasze stoisko
Stoisko F14, hala 3

SEMICON SP. Z O.O.
ul. Zwoleńska 43/43a, 04-761 Warszawa
tel: 22 615 64 31, 22 615 73 71
DZIAŁ USŁUG DLA ELEKTRONIKI
ul. Ezopa 71a, 04-805 Warszawa
tel: 22 612 67 92, 22 825 24 64, 22 615 27 05