Komputer samochodowy Mee 2.0 (1)

Współcześnie czas życia urządzenia jest krótszy, niż kiedykolwiek wcześniej, co tylko potwierdza obiegową opinię, iż rozwój elektroniki użytkowej jest teraz szybszy. Wystarczy zauważyć, że większość producentów urządzeń z tego obszaru, aktualizuje swoje produkty co najmniej raz w roku fundując przy okazji swoim klientom niezłą łamigłówkę decyzyjną. Z jednej strony wydaje się to irracjonalne, z drugiej, dzięki temu dostajemy coraz to nowocześniejsze i zwykle znacznie ładniejsze urządzenia. Dobrym przykładem może być tutaj motoryzacja, która jest świetną platformą prezentacji możliwości technologicznych producentów różnych gałęzi przemysłu, w tym elektroniki w szczególności. Wszak dzisiejsze samochody są syntezą wielu, czasami ekstremalnie nowoczesnych, rozwiązań elektronicznych, które to integrują w sobie systemy, o jakich jeszcze do niedawna mogliśmy wyłącznie pomarzyć.
Rekomendacje: estetyczny komputer, który może usprawnić korzystanie ze starszego modelu samochodu lub być dobrym przykładem, jak takie rozwiązanie zaimplementować samodzielnie.

Kilku uznanych producentów samochodów (i nie tylko) deklaruje, iż w ciągu kliku lat wprowadzi do sprzedaży pojazdy samodzielnie poruszające się po drogach, zaś już teraz wyposaża swoje produkty w systemy, które pozwalają nazwać je autonomicznymi. Jako, że ostatnimi czasy miałem okazję „liznąć" tej nowoczesnej motoryzacji, muszę stwierdzić, że rzeczywiście robi ona piorunujące wrażenie, choć trzeba przyznać, że ta autonomiczność wymaga pewnego przyzwyczajenia. Z mojego punktu widzenia, osoby, która zajmowała się nieco grafiką komputerową i która to zwraca szczególną uwagę na formę prezentacji, szczególne wrażenie zrobiły wszelkiego rodzaju graficzne interfejsy użytkownika, prezentujące w piękny sposób zarówno

DODATKOWE MATERIAŁY NA FTP:

ftp://ep.com.pl
USER: 77642, PASS: 3220ppmm

W ofercie AVT*
AVT-5545

Podstawowe informacje:

- Napięcie zasilające: 8...15 V DC.
- Prąd zasilający:
  - Maksymalny prąd obciążenia (z napięcia +12 V): 10 mA.
  - Prąd podtrzymania zegara RTC (z napięcia BATT): 1 mA.
  - Maksymalny prąd podświetlenia (z napięcia ILL+): 75 mA.
- Dokładność pomiaru temperatury: 0,5°C.
- Zakres pomiarowy temperatury zewnętrznej i wewnętrznej: -55...99°C.
- Zakres pomiarowy prędkości pojazdu: 0...255 km/godz.
- Zakres pomiarowy paliwa:
  - Zużycie chwilowe: 0...99,9 l/100 km.
  - Zużycie średnie: 0...25,5 l/100 km.
  - Dostępnego w baku: 0...99,9 l.
- Zakres pomiarowy przejechanego dystansu: 0...9999 km.
- Zakres pomiarowy dystansu do przejechania na dostępnym paliwie: 0...999 km.
- Zakresy regulacji parametrów konfiguracyjnych:
  - Stała wtryskiwacza: 1...999 ml/min.
  - Stała przetwornika drogi: 1...99 impulsów/obrót.
  - Obwód opony: 50...255 cm.
  - Liczba cylindrów: 2...8.
  - Pojemność baku: 25...99 l.

Projekty pokrewne na FTP:
(wymienione artykuły są w całości dostępne na FTP)

AVT-5495	Uniwersalny komputer samochodowy Mee (EP 3/2015)
AVT-3095	Komputer samochodowy (EdW 4-5/2014)
AVT-5405	TripCo – komputer samochodowy (EP 7/2013)
AVT-5395	TIDex – komputer dla samochodów z silnikiem Diesla (EP 5/2013)
AVT-5397	Komputer pokładowy z funkcją tempomatu (EP 5/2013)
AVT-1664	Transceiver CAN (EP 2/2012)
AVT-5280	Urządzenie diagnostyczne do sieci CAN (EP 3/2011)
AVT-5271	VAGlogger – Przyrząd diagnostyczny dla samochodów z grupy VW – Audi (EP 1/2011)
AVT-5260	Obrotomierz cyfrowy (EP 10/2010)
AVT-2799	Mikroprocesorowy obrotomierz stroboskopowy (EdW 9/2006)
AVT-434	Komputer samochodowy (EP 9-10/2005)
AVT-2711	Obrotomierz (EdW 2/2004)
AVT-482	Obrotomierz z czujnikiem optycznym (EP 1/1999)
Projekt 117	Wskaźnik optymalnych obrotów silnika samochodowego (EP 3/2004)
Projekt 116	Cyfrowy obrotomierz/prędkościomierz samochodowy (EP 2/2004)
---	Obrotomierz cyfrowo-analogowy (EdW 6/2010)
AVT-286	„Komputerek” pokładowy do samochodu (EP 5-6/1996)

* Uwaga:
Zestawy AVT mogą występować w następujących wersjach:
AVT xxxx UK to zaprogramowany układ. Tylko i wyłącznie. Bez elementów dodatkowych.
AVT xxxx A płytka drukowana PCB (lub płytki drukowane, jeśli w opisie wyraźnie zaznaczono), bez elementów dodatkowych.
AVT xxxx A+ płytka drukowana i zaprogramowany układ (czyli połączenie wersji A i wersji UK) bez elementów dodatkowych.
AVT xxxx B płytka drukowana (lub płytki) oraz komplet elementów wymieniony w załączniku pdf
AVT xxxx C to nic innego jak zmontowany zestaw B, czyli elementy wlutowane w PCB. Należy mieć na uwadze, że o ile nie zaznaczono wyraźnie w opisie, zestaw ten nie ma obudowy ani elementów dodatkowych, które nie zostały wymienione w załączniku pdf
AVT xxxx CD oprogramowanie (nieczęsto spotykana wersja, lecz jeśli występuje, to niezbędne oprogramowanie można ściągnąć, klikając w link umieszczony w opisie kitu)
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! (UK, A, A+, B lub C). http://sklep.avt.pl

Tabela 1. Rozmieszczenie wyprowadzeń taśmy ZIF wyświetlacza RVT28AET-NWN00

Nr wypr.	Oznaczenie	Opis
1	LEDK	Katoda diod podświetlenia
2	LEDA1	Anoda pierwszej diody podświetlenia
3	LEDA2	Anoda drugiej diody podświetlenia
4	LEDA3	Anoda trzeciej diody podświetlenia
5	LEDA4	Anoda czwartej diody podświetlenia
6	IM0	Interfejs wyboru aktywnej magistrali sterującej
7	IM1	
8	IM2	
9	IM3	
10	RESET	Sygnał zerowania sterownika ekranu
11	VSYNC	Sygnały interfejsu RGB
12	HSYNC	
13	DOTCLK	
14	DE	
15...32	DB17...DB0	Równoległa magistrala sterująca/danych
33	SDO	Wyjście szeregowej magistrali sterującej SPI
34	SDI/SDA	Wejście szeregowej magistrali sterującej SPI lub linia danych szeregowej magistrali sterującej I²C
35	RDX	Sygnał odczytu magistrali sterującej
36	WRX	Sygnał zapisu magistrali sterującej
37	DCX (SCL)	Sygnał wyboru rodzaju danych: dane obrazu (1)/rozkaz sterujący (0) lub linia sygnału zegarowego szeregowej magistrali sterującej I²C
38	CSX	Sygnał wyboru sterownika ekranu
39	TE	Wyjście synchronizacji ramki dla MCU
40	VDDI	Zasilanie
41	VDDI	
42	VCI	
43	GND	Masa zasilania
44...47	NC	Nieużywane (interfejs panelu dotykowego w wersjach ze zintegrowanym panelem)
48...50	GND	Masa zasilania

dane pokładowe, jak i zawartości multimedialne, włączając w to coraz to częściej stosowane wirtualne zespoły zegarów w konsoli centralnej. Niby to nic nowego, bo zwykle nie pozyskujemy w ten sposób jakiś nowych, interesujących nas informacji, jednakże sposób ich „podania” budzi zachwyt i daje możliwość dowolnej personalizacji. Idąc tym tropem postanowiłem „zaktualizować” jeden ze swoich ostatnich projektów, a mianowicie uniwersalny komputer pokładowy Mee (EP 03/2015), wyposażając go w nowoczesny, piękny interfejs użytkownika zbudowany z wykorzystaniem kolorowego wyświetlacza TFT.

W tym momencie musiałem pokonać jeden z głównych problemów, na które napotykamy się za każdym razem, gdy sięgamy po element TFT, jego cenę! Moim głównym założeniem, podobnie, jak to miało miejsce, przy projekcie Mee, było przecież zbudowanie urządzenia dostępnego dla każdego, więc docelowy koszt jego budowy był głównym kryterium wyboru. Nie pozostało więc nic innego, jak znaleźć dobrej jakości, niewielki wyświetlacz TFT, którego cena pozwalałaby sprostać przyjętym założeniom. Na szczęście nie szukałem zbyt długo, gdyż jak zwykle mogłem liczyć na pomoc pana Sławomira Szwedy z firmy Unisystem, który dostarczył

Listing 1. Podstawowe funkcje narzędziowe odpowiedzialne za wysyłanie rozkazów sterujących lub danych do wyświetlacza TFT

```
void writeCommand(uint8_t Command)
{
    RESET_DCX;  //Command
    RESET_WRX;
    TFT_DATA_PORT = Command;
    SET_WRX;  //TFT reads data at the rising edge
   SET_DCX;  //DCX is 1 by default to increase DRAM data transfer
}


void writeData(uint8_t Data)
{
    RESET_WRX;
    TFT_DATA_PORT = Data;
    SET_WRX;  //TFT reads data at the rising edge
}
```

Tabela 2. Wybór aktywnego interfejsu za pomocą wyprowadzeń IM3...IM0

IM3	IM2	IM1	IM0	Rodzaj interfejsu	Magistrala sterująca	Magistrala danych obrazu GRAM	Dodatkowe, aktywne sygnały sterujące
0	0	0	0	Równoległy, 8-bitowy	D7...D0	D7...D0	WRX,RDX,CSX,DCX
0	0	0	1	Równoległy, 16-bitowy	D7...D0	D15...D0	WRX,RDX,CSX,DCX
0	0	1	0	Równoległy, 9-bitowy	D7...D0	D8...D0	WRX,RDX,CSX,DCX
0	0	1	1	Równoległy, 18-bitowy	D7...D0	D17...D0	WRX,RDX,CSX,DCX
0	1	0	1	Szeregowy I²C, 9-bitowy, 3-przewodowy	SDA		CSX
0	1	1	0	Szeregowy I²C, 8-bitowy, 4-przewodowy	SDA		DCX, CSX
1	0	0	0	Równoległy, 16-bitowy	D8...D1	D17...D10, D8...D1	WRX,RDX,CSX,DCX
1	0	0	1	Równoległy, 8-bitowy	D17...D10	D17...D10	WRX,RDX,CSX,DCX
1	0	1	0	Równoległy, 18-bitowy	D8...D1	D17...D0	WRX,RDX,CSX,DCX
1	0	1	1	Równoległy, 9-bitowy	D17...D10	D17...D9	WRX,RDX,CSX,DCX
1	1	0	1	Szeregowy SPI, 9-bitowy, 3-przewodowy	SDI/SDO		CSX
1	1	1	0	Szeregowy SPI, 8-bitowy, 4-przewodowy	SDI/SDO		DCX, CSX

moduł RVT28AETNWN00 firmy Rivierdi wyposażony w popularny kontroler ILI9341, który to idealnie wpisuje się w nasze wymagania zarówno, jeśli chodzi o parametry techniczne, jak i bardzo niską cenę.

Zacznę nietypowo, bo od opisu wspomnianego modułu. Dlaczego warto poznać ten element? Po pierwsze, dlatego, że z uwagi na swoje parametry stał się dość popularny i łatwo kupić gotowe moduły z wygodnym złączem, a po drugie, jego niska cena powoduje, że znacznie chętniej będziemy sięgać po to nowoczesne rozwiązanie. Wyświetlacz firmy Rivierdi charakteryzuje się następującymi, wybranymi cechami funkcjonalnymi:

- Przekątna ekranu: 2,83" (obszar aktywny 43,2 mm×57,6 mm), bardzo mała grubość modułu – rzędu 3 mm.
- Rozdzielczość: 320x240 pikseli (możliwość pracy w trybie portret 240×320 pikseli lub pejzaż 320×240 pikseli).
- Liczba kolorów: 65 tys. /262 tys.
- Wbudowany filtr antyodblaskowy, podświetlenie LED.
- Interfejsy: 8-, 9-, 16- i 18-bitowa szyna danych, RGB, SPI/I²C (3- i 4-przewodowy), możliwość oddzielenia szyny danych od szyny rozkazów dla interfejsu równoległego.
- Możliwość wyposażenia w opcjonalny pojemnościowy lub rezystancyjny panel dotykowy,
- Taśma podłączeniowa typu ZIF 50 pinów (raster 0,5 mm).
- Napięcie zasilające 2,5…3,3 V.

Wyświetlacz RVT28AETNWN00 doskonale spełnia nasze wymagania, a dodatkowo zastosowany w nim sterownik ILI9341 ułatwia wykonanie oprogramowania sterującego. W **tabeli 1** umieszczono opis rozmieszczenia wyprowadzeń taśmy ZIF.

Moduł TFT wyposażono w wiele interfejsów sterujących, przez co jego zastosowanie w docelowym systemie mikroprocesorowym jest ułatwione. Wybór aktywnego interfejsu

Listing 2. Funcja inicjalizacyjna sterownika ekranu ILI9341 wyświetlacza Rivierdi RVT28AETNWN00

```
void TFTinit(void)
{
    TFT_DATA_DDR = 0xFF;  //Data port as output with 0x00
    TFT_CTRL_PORT|= (1<<WRX_PIN)|(1<<DCX_PIN)|(1<<RESX_PIN)|(1<<RDX_PIN);  //
ChipSelect (CSX) is 0 by default
    TFT_CTRL_DDR |= (1<<WRX_PIN)|(1<<CSX_PIN)|(1<<DCX_PIN)|(1<<RESX_
PIN)|(1<<RDX_PIN);  //All control ports are outputs
    _delay_ms(5);
    //Hardware reset
    RESET_RESX;
    _delay_ms(5);
    SET_RESX;
    _delay_ms(50);
    //Software reset and configuration
    writeCommand(CMD_SOFTWARE_RESET);
    _delay_ms(5);
    writeCommand(CMD_DISPLAY_OFF);
    writeCommand(CMD_PWR_CTRL_B);
    writeData(0x00);  //Default values
    writeData(0x83);
    writeData(0x30);
    writeCommand(CMD_PWR_ON_SEQ_CTRL);
    writeData(0x64);  //Default values
    writeData(0x03);
    writeData(0x12);
    writeData(0x81);
    writeCommand(CMD_DRIVER_TIM_CTRL_A);
    writeData(0x85);  //Default values
    writeData(0x01);
    writeData(0x79);
    writeCommand(CMD_PWR_CTRL_A);
    writeData(0x39);  //Default value
    writeData(0x2c);  //Default value
    writeData(0x00);  //Default value
    writeData(VCORE_1_60);
    writeData(DDVDH_5_6);
    writeCommand(CMD_PUMP_RATIO_CTRL);
    writeData(0x20);  //Default value
    writeCommand(CMD_DRIVER_TIM_CTRL_B);
    writeData(0x00);  //Default value
    writeData(0x00);  //Default value
    //Power Control
    writeCommand(CMD_PWR_CTRL_1);
    writeData(0x26);  //Default value
    writeCommand(CMD_PWR_CTRL_2);
    writeData(0x11);  //Default value
    //VCOM setting
    writeCommand(CMD_VCOM_CTRL_1);
    writeData(0x35);  //Default value
    writeData(0x3e);  //Default value
    writeCommand(CMD_VCOM_CTRL_2);
    writeData(0xbe);  //Default value
    //Memory Access Control
    writeCommand(CMD_MEM_ACCESS_CTRL);
    writeData(ROW_COLUMN_EXCHANGE_ON|RGB_ORDER_BGR);
    writeCommand(CMD_PIXEL_FORMAT);
    writeData(RGB_INTERFACE_16BITS|MCU_INTERFACE_16BITS);
    //Frame Rate
    writeCommand(CMD_FRAME_RATE);
    writeData(DIVISION_RATIO_FOSC);
    writeData(FRAME_RATE_119KHZ);
    //Gamma
    writeCommand(CMD_GAMMA3_FUNCTION_DISABLE);
    writeData(0x08);  //Default value
    writeCommand(CMD_SET_GAMMA_CURVE_NR);
    writeData(0x01);  //One curve
    writeCommand(CMD_POSITIVE_GAMMA_CORR);
    writeData(0x1f);
    writeData(0x1a);
    writeData(0x18);
    writeData(0x0a);
```

sterującego jest wykonywany za pomocą wyprowadzeń IM0…IM3 modułu (**tabela 2**).

Jak widać, jest w czym wybierać! Jednak z uwagi na fakt, że nie dysponujemy bardzo szybkim mikrokontrolerem (AVR ATmega), posłużymy się równoległym, 8-bitowym interfejsem danych/rozkazów sterujących, dla którego aktywowania piny IM3…IM0 powinny być wyzerowane. Oczywiście, można by zastosować 16-bitową magistralę danych, co zwiększyłoby szybkość transmisji (w procesorze 8-bitowym tylko nieznacznie), jednak w ten sposób zajmiemy cenne wyprowadzenia mikrokontrolera, które będą potrzebne do realizacji innych funkcjonalności. Zastosujemy jednak dwa inne rozwiązania, które pozwolą na zwiększenie prędkości przesyłania danych z mikrokontrolera do sterownika ekranu ILI9341. Po pierwsze, sygnał wyboru sterownika ekranu CSX przyłączymy na stałe do masy zasilania aktywując „na stałe" sterownik ILI9341, przez co funkcje odpowiedzialne za transfer danych nie muszą „obsługiwać" tegoż sygnału. Po drugie, przyjmiemy, że poziomem spoczynkowym sygnału wyboru rodzaju danych DCX będzie logiczna „1", co oznacza, że przesyłane dane są domyślnie danymi treści obrazu lub danymi towarzyszącymi rozkazom sterującym. Pozwoli to na dalsze przyśpieszenie stosownych funkcji graficznych. Nasza biblioteka nie będzie również obsługiwać sygnału odczytu magistrali sterującej RDX, gdyż nie korzystamy w niej z możliwości odczytu danych ze sterownika ekranu. Sygnał ten na stałe ma poziom wysoki.

No dobrze, mamy już garść niezbędnych informacji dotyczących sterownika ekranu, więc pora zacząć „zabawę" w programowanie! Niezbędne są do tego dwie podstawowe funkcje, które pozwolą na wysłanie do wyświetlacza TFT rozkazu sterującego lub danych obrazu (w analogiczny sposób jak dane obrazu wysyłamy dane towarzyszące rozkazom sterującym). Interpretacja rodzaju danych, które są odbierane przez sterownik ekranu, jest zdeterminowana poziomem wyprowadzenia DCX. Poziom niski na tym wyprowadzeniu decyduje o tym, iż przesyłana wartość zinterpretowana zostanie jako komenda sterująca, zaś wysoki, iż dana zostanie potraktowana jako argument dla tegoż rozkazu sterującego lub dana pamięci ekranu (w zależności od przesłanego wcześniej polecenia). Wspomniane, podstawowe funkcje narzędziowe zamieszczono na **listingu 1**.

Teraz możemy przystąpić do inicjalizacji sterownika naszego modułu TFT, ponieważ wymaga on ustawienia szeregu rejestrów konfiguracyjnych. Funkcję inicjalizacyjną pokazano na **listingu 2**. Brak przeprowadzenia inicjalizacji sterownika ekranu w zasadzie uniemożliwia poprawne

Listing 2. cd.

```
    writeData(0x0f);
    writeData(0x06);
    writeData(0x45);
    writeData(0x87);
    writeData(0x32);
    writeData(0x0a);
    writeData(0x07);
    writeData(0x02);
    writeData(0x07);
    writeData(0x05);
    writeData(0x00);
    writeCommand(CMD_NEGATIVE_GAMMA_CORR);
    writeData(0x00);
    writeData(0x25);
    writeData(0x27);
    writeData(0x05);
    writeData(0x10);
    writeData(0x09);
    writeData(0x3a);
    writeData(0x78);
    writeData(0x4d);
    writeData(0x05);
    writeData(0x18);
    writeData(0x0d);
    writeData(0x38);
    writeData(0x3a);
    writeData(0x1f);
    //DISPLAY RAM
    writeCommand(CMD_COLUMN_ADDR_SET);
    writeData(0x00);
    writeData(0x00);
    writeData((TFT_WIDTH-1)>>8);
    writeData((uint8_t) TFT_WIDTH-1);
    writeCommand(CMD_PAGE_ADDR_SET);
    writeData(0x00);
    writeData(0x00);
    writeData((TFT_HEIGHT-1)>>8);
    writeData(TFT_HEIGHT-1);
    writeCommand(CMD_ENTRY_MODE);
    writeData(GON_DTE_NORMAL_DISPLAY|LOW_VOLTAGE_DETECT_DISBLE);
    //Display
    writeCommand(CMD_DISPLAY_FUNCTION_CTRL);
    writeData(INTERVAL_SCAN|0b10);
    writeData(LCD_NORMALLY_WHITE|GATE_OUT_SCAN_DIR_NORMAL|SOURCE_OUT_SCAN_DIR_
NORMAL|GATE_OUT_SEQUENCE_NORMAL|SCAN_CYCLE_INTERVAL_5FRAME);
    writeData(NUMBER_OF_LINES_320);
    writeData(0x00);  //Default value
    writeCommand(CMD_SLEEP_OUT);
    _delay_ms(5);
    writeCommand(CMD_DISPLAY_ON);
    _delay_ms(5);
}
```

Listing 3. Plik nagłówkowy sterownika ekranu ILI9341 wyświetlacza Rivierdi RVT28AETNWN00

```
//PHYSICAL PARAMETERS - LANDSCAPE MODE
#define TFT_WIDTH 320
#define TFT_HEIGHT 240
//PORTS CONFIGURATION. We assume that RDX (read signal) is permanently pulled
high
#define TFT_DATA_PORT PORTA
#define TFT_DATA_DDR DDRA
#define TFT_CTRL_PORT PORTB
#define TFT_CTRL_DDR DDRB
#define WRX_PIN PB1  //Write signal
#define RDX_PIN PB7  //Read signal
#define CSX_PIN PB3  //Chip select signal
#define DCX_PIN PB2  //Command/Data signal: 1->DRAM data, 0->Command
#define RESX_PIN PB4  //Reset signal
//MACROS
#define SET_WRX TFT_CTRL_PORT |= (1<<WRX_PIN)
#define RESET_WRX TFT_CTRL_PORT &= ~(1<<WRX_PIN)
#define SET_CSX TFT_CTRL_PORT |= (1<<CSX_PIN)
#define RESET_CSX TFT_CTRL_PORT &= ~(1<<CSX_PIN)
#define SET_DCX TFT_CTRL_PORT |= (1<<DCX_PIN)
#define RESET_DCX TFT_CTRL_PORT &= ~(1<<DCX_PIN)
#define SET_RESX TFT_CTRL_PORT |= (1<<RESX_PIN)
#define RESET_RESX TFT_CTRL_PORT &= ~(1<<RESX_PIN)
//ILI9341 COMMANDS AND PARAMETERS
#define CMD_SOFTWARE_RESET 0x01  //5ms delay is needed after this command
#define CMD_SLEEP_OUT 0x11  //5ms delay is needed after this command
#define CMD_SET_GAMMA_CURVE_NR 0x26  //Selects gamma curve number
#define CMD_DISPLAY_OFF 0x28
#define CMD_DISPLAY_ON 0x29
#define CMD_COLUMN_ADDR_SET 0x2A  //Sets current column address
#define CMD_PAGE_ADDR_SET 0x2B  //Sets current page address
#define CMD_MEMORY_WRITE 0x2C  //Starts frame memory writting
#define CMD_MEM_ACCESS_CTRL 0x36  //Defines read/write scanning direction of
frame memory
    #define ROW_ADDR_ORDER_NORMAL (0<<7)
    #define ROW_ADDR_ORDER_REVERSE (1<<7)
    #define COLUMN_ADDR_ORDER_NORMAL (0<<6)
    #define COLUMN_ADDR_ORDER_REVERSE (1<<6)
    #define ROW_COLUMN_EXCHANGE_OFF (0<<5)
    #define ROW_COLUMN_EXCHANGE_ON (1<<5)
    #define VERTICAL_REFRESH_ORDER_NORMAL (0<<4)
    #define VERTICAL_REFRESH_ORDER_REVERSE (1<<4)
    #define RGB_ORDER_RGB (0<<3)
    #define RGB_ORDER_BGR (1<<3)
    #define HORIZONTAL_REFRESH_ORDER_NORMAL (0<<2)
```

Listing 3. cd.

```
    #define HORIZONTAL_REFRESH_ORDER_REVERSE (1<<2)
#define CMD_PIXEL_FORMAT 0x3A  //Sets pixel data format
    #define RGB_INTERFACE_16BITS (0b101<<4)
    #define RGB_INTERFACE_18BITS (0b110<<4)
    #define MCU_INTERFACE_16BITS (0b101)
    #define MCU_INTERFACE_18BITS (0b110)
#define CMD_FRAME_RATE 0xB1  //Sets the division ratio for internal clocks
    #define DIVISION_RATIO_FOSC 0x00  //1st parameter
    #define DIVISION_RATIO_FOSC_2 0x01
    #define DIVISION_RATIO_FOSC_4 0x02
    #define DIVISION_RATIO_FOSC_8 0x03
    #define FRAME_RATE_119KHZ 0b10000  //2nd parameter
    #define FRAME_RATE_112KHZ 0b10001
    #define FRAME_RATE_106KHZ 0b10010
    #define FRAME_RATE_100KHZ 0b10011
    #define FRAME_RATE_95KHZ 0b10100
    #define FRAME_RATE_90KHZ 0b10101
    #define FRAME_RATE_85KHZ 0b10110
    #define FRAME_RATE_83KHZ 0b10111
#define CMD_DISPLAY_FUNCTION_CTRL 0xB6
    #define NORMAL_SCAN (0b00<<2)  //1st parameter
    #define INTERVAL_SCAN (0b10<<2)
    #define LCD_NORMALLY_BLACK (0<<7)  //2nd parameter
    #define LCD_NORMALLY_WHITE (1<<7)
    #define GATE_OUT_SCAN_DIR_NORMAL (0<<6)
    #define GATE_OUT_SCAN_DIR_REVERSE (1<<6)
    #define SOURCE_OUT_SCAN_DIR_NORMAL (0<<5)
    #define SOURCE_OUT_SCAN_DIR_REVERSE (1<<5)
    #define GATE_OUT_SEQUENCE_NORMAL (0<<4)
    #define GATE_OUT_SEQUENCE_REVERSE (1<<4)
    #define SCAN_CYCLE_INTERVAL_1FRAME (0b0000)
    #define SCAN_CYCLE_INTERVAL_3FRAME (0b0001)
    #define SCAN_CYCLE_INTERVAL_5FRAME (0b0010)
    #define SCAN_CYCLE_INTERVAL_7FRAME (0b0011)
    #define SCAN_CYCLE_INTERVAL_9FRAME (0b0100)
    #define SCAN_CYCLE_INTERVAL_11FRAME (0b0101)
    #define SCAN_CYCLE_INTERVAL_13FRAME (0b0110)
    #define SCAN_CYCLE_INTERVAL_15FRAME (0b0111)
    #define NUMBER_OF_LINES_320 0x27  //3rd parameter
    #define NUMBER_OF_LINES_240 0x1D
#define CMD_ENTRY_MODE 0xB7
    #define LOW_VOLTAGE_DETECT_ENABLE (0x00)
    #define LOW_VOLTAGE_DETECT_DISBLE (0x01)
    #define GON_DTE_VGH (0x00<<1)
    #define GON_DTE_VGH2 (0x01<<1)
    #define GON_DTE_VGL (0x02<<1)
    #define GON_DTE_NORMAL_DISPLAY (0x03<<1)
#define CMD_PWR_CTRL_1 0xC0  //Sets the GVDD level, which is a reference level
for the VCOM
#define CMD_PWR_CTRL_2 0xC1  //Sets the factor used in the step-up circuits.
#define CMD_VCOM_CTRL_1 0xC5  //Sets the VCOM voltage.
#define CMD_VCOM_CTRL_2 0xC7  //Set the VCOM offset voltage
#define CMD_PWR_CTRL_A 0xCB
    #define VCORE_1_55 0x30
    #define VCORE_1_40 0x31
    #define VCORE_1_50 0x32
    #define VCORE_1_65 0x33
    #define VCORE_1_60 0x34
    #define VCORE_1_70 0x35
    #define DDVDH_5_8 0x00
    #define DDVDH_5_7 0x01
    #define DDVDH_5_6 0x02
    #define DDVDH_5_5 0x03
    #define DDVDH_5_4 0x04
    #define DDVDH_5_3 0x05
    #define DDVDH_5_2 0x06
#define CMD_PWR_CTRL_B 0xCF
#define CMD_POSITIVE_GAMMA_CORR 0xE0  //Set the gray scale voltage to adjust
the gamma of the TFT
#define CMD_NEGATIVE_GAMMA_CORR 0xE1  //Set the gray scale voltage to adjust
the gamma of the TFT
#define CMD_DRIVER_TIM_CTRL_A 0xE8  //EQ timing for Internal clock
#define CMD_DRIVER_TIM_CTRL_B 0xEA
#define CMD_PWR_ON_SEQ_CTRL 0xED
#define CMD_GAMMA3_FUNCTION_DISABLE 0xF2
#define CMD_PUMP_RATIO_CTRL 0xF7
//Auxiliary macros - used in selected functions
#define SOLID_TEXT 0
#define TRANSPARENT_TEXT 1
```

Listing 4. Funkcja narzędziowa odpowiedzialna za ustawienie aktywnego obszaru ekranu, w ramach którego przeprowadzany jest zapis do pamięci ekranu sterownika ILI9341

```
void TFTsetActiveWindow(uint16_t X1, uint8_t Y1, uint16_t X2, uint8_t Y2)
{
    writeCommand(CMD_COLUMN_ADDR_SET);
    writeData(X1 >> 8);
    writeData(X1 & 0xFF);
    writeData(X2 >> 8);
    writeData(X2 & 0xFF);
    writeCommand(CMD_PAGE_ADDR_SET);
    writeData(Y1 >> 8);
    writeData(Y1);
    writeData(Y2 >> 8);
    writeData(Y2);
}
```


ELEKTRONIKA PRAKTYCZNA

teraz zawsze z Tobą w wersji mobilnej

m.ep.com.pl

REKLAMA

Listing 5. Funkcje umożliwiające wyświetlanie wypełnionego i „pustego" prostokąta na ekranie wyświetlacza TFT

```
void TFTdrawFilledBox(uint16_t X1, uint8_t Y1, uint16_t X2, uint8_t Y2)
{
    uint32_t pixelsToSend = (X2-X1+1)*(Y2-Y1+1);  //We calculate how many pixels we need to send (2 bytes/pixel)
    //We define display active area to simplify writing
    TFTsetActiveWindow(X1, Y1, X2, Y2);
    //We start memory writing
    writeCommand(CMD_MEMORY_WRITE);
    while(pixelsToSend --) {writeData(Colour>>8); writeData(Colour&0xFF);}  //(2 bytes/pixel)
}

void TFTdrawRectangle(uint16_t X1, uint8_t Y1, uint16_t X2, uint8_t Y2)
{
    TFTdrawFilledBox(X1 , Y1 , X2 , Y1);
    TFTdrawFilledBox(X1 , Y2 , X2 , Y2);
    TFTdrawFilledBox(X1 , Y1 , X1 , Y2);
    TFTdrawFilledBox(X2 , Y1 , X2 , Y2);
}
```

Listing 6. Funkcja odpowiedzialna za wyświetlanie obrazków na ekranie wyświetlacza TFT

```
void TFTdrawPicture(uint16_t X1, uint8_t Y1, const uint16_t *Picture)
{
    register uint16_t pixelData, pixelsToSend;
    register uint8_t Width, Height;
    //We read the first word that holds the picture width and height (MSB and
LSB)
    pixelData = pgm_read_word(Picture++);
    //We calculate the picture width and height
    Width = pixelData >> 8; Height = pixelData & 0xFF;
    //We calculate how many pixels we need to send
    pixelsToSend = Width * Height;
    //We define display active area to simplify writing
    TFTsetActiveWindow(X1, Y1, X1+Width-1, Y1+Height-1);
    //We start memory writing
    writeCommand(CMD_MEMORY_WRITE);
    while(pixelsToSend--)
    {
        pixelData = pgm_read_word(Picture++);
        writeData(pixelData >> 8); writeData(pixelData & 0xFF);
    }
}
```

Listing 7. Funkcja odpowiedzialna za wyświetlanie skompresowanych obrazków na ekranie wyświetlacza TFT

```
void TFTdrawCompressedPicture(uint16_t X1, uint8_t Y1, const uint16_t
*Picture)
{
    register uint16_t pixelsToSend, pixelA, pixelB;
    register uint8_t Width, Height;
    //We read the first word that holds the picture width and height (MSB and
LSB)
    pixelA = pgm_read_word(Picture++);
    //We calculate the picture width and height
    Width = pixelA >> 8; Height = pixelA & 0xFF;
    //We calculate how many pixels we need to send
    pixelsToSend = Width * Height;
    //We define display active area to simplify writing
    TFTsetActiveWindow(X1, Y1, X1+Width-1, Y1+Height-1);
    //We start memory writing
    writeCommand(CMD_MEMORY_WRITE);
    while(pixelsToSend)
    {
        //We read pixel n and n+1
        pixelA = pgm_read_word(Picture++);
        pixelB = pgm_read_word(Picture);
        //If the pixel n is different than the pixel n+1 or if it is the last
        //pixel, we send it to the TFT
        if(pixelA != pixelB || pixelsToSend == 1)
        {
            writeData(pixelA >> 8); writeData(pixelA & 0xFF);
            pixelsToSend--;
        }
        else
        {
            //Otherwise we read how many the same pixels we need to send
(third word)
            pixelB = pgm_read_word(++Picture);
            pixelsToSend -= pixelB;
            Picture++;
            //We sent them to the TFT
            while(pixelB--) {writeData(pixelA >> 8); writeData(pixelA &
0xFF);}
        }
    }
}
```

Listing 8. Definicja nowego typu danych odpowiedzialnego za przechowywanie parametrów bieżącej czcionki ekranowej

```
//Deklaracja struktury przechowującej parametry bieżącej czcionki ekranowej
typedef struct
{
    uint8_t Width;  //Current font widht (px)
    uint8_t Height;  //Current font height (px)
    uint8_t Interspace;  //Font interspace (px)
    uint8_t BytesPerChar;  //Bytes per char definition
    uint8_t FirstCharCode;  //First char ASCII code
    const uint8_t *Bitmap;  //Pointer to the font table
} fontDescription;
```

funkcjonowanie modułu TFT, gdyż domyślne ustawienia rejestrów sterujących układu ILI9341 zwykle odbiegają od tych, wymaganych przez producenta wyświetlacza. Aby jednak w pełni zrozumieć znaczenie poszczególnych ustawień sterownika ekranu, niezbędna jest znajomość zawartości pliku nagłówkowego zamieszczonego na **listingu 3**.

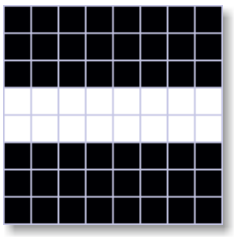
Dalej, na **listingu 4**,pokazano funkcję odpowiedzialną za ustawienie aktywnego obszaru ekranu, w którego ramach jest przeprowadzany jest zapis do pamięci ekranu sterownika ILI9341 – jej użycie upraszcza manipulacje w zakresie pamięci obrazu.

Czas na funkcje odpowiedzialne za rysowanie prostych elementów graficznych, to jest funkcje umożliwiające wyświetlanie wypełnionego i „pustego" prostokąta na ekranie wyświetlacza TFT – pokazano je na **listingu 5**. Te funkcje korzystają z dwóch zmiennych globalnych przechowujących bieżący kolor treści ekranu oraz kolor tła. Specyfikacja tych zmiennych przedstawia się następująco: *uint16_t Colour, Background*.

Na **listingu 6** zamieszczono nieskomplikowaną funkcję umożliwiającą wyświetlanie plików graficznych na ekranie wyświetlacza. Jednym z argumentów jest wskaźnik na tablicę (elementów 16-bitowych) umieszczoną w pamięci programu (Flash), która zawiera wielkość obrazu (pierwszy element: szerokość → MSB, wysokość → LSB) oraz jego zawartość (pozostałe elementy, przy czym każdy z nich zawiera kolor kolejnego piksela obrazu w formacie RGB565). Jest to rozwiązanie bardzo proste, jednak obarczone pewną wadą. Jak łatwo się domyślić, tak przygotowane obrazki zajmują dość sporo cennej pamięci programu. Jak temu zaradzić? Odpowiedź wydaje się dość prosta! Należy zastosować jakąś metodę kompresji powtarzających się danych, co pozwoli na ograniczenie wynikowego rozmiaru tablicy przechowującej treść obrazu. Ideę działania zastosowanego algorytmu kompresji najlepiej prześledzić analizując przykładowe ciągi słów 16-bitowych reprezentujących kolory kolejnych pikseli (założono powtarzanie się pikseli o kolorze 0xAAAA), co przedstawiono w **tabeli 3**.

Tabela 3. Idea działania algorytmu kompresującego dane obrazków

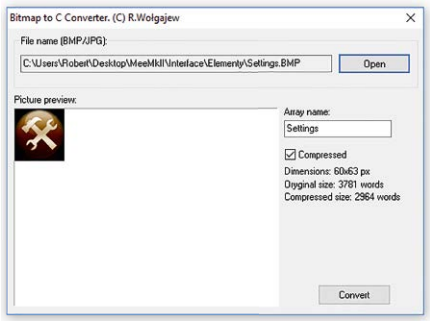
Liczba powtórzeń	Dane przed kompresją	Dane po kompresji
1	0xAAAA	0xAAAA
2	0xAAAA 0xAAAA	0xAAAA 0xAAAA 0x0002
3	0xAAAA 0xAAAA 0xAAAA	0xAAAA 0xAAAA 0x0003
4	0xAAAA 0xAAAA 0xAAAA 0xAAAA	0xAAAA 0xAAAA 0x0004
5	0xAAAA 0xAAAA 0xAAAA 0xAAAA 0xAAAA	0xAAAA 0xAAAA 0x0005
65535	65535 x 0xAAAA	0xAAAA 0xAAAA 0xFFFF



```
const uint16_t Uncompressed[] PROGMEM =
{
0x0808,  //Width (MSB)& Height (LSB)
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000};

const uint16_t Compressed[] PROGMEM =
{
0x0808,  //Width (MSB)& Height (LSB)
0x0000,0x0000,0x0018,0xFFFF,0xFFFF,0x0010,0x0000,0x0000,
0x0018};
```

Rysunek 1. Wynik zastosowania algorytmu kompresji obrazu w świetle rozmiaru wynikowej tablicy danych



Rysunek 2. Wygląd aplikacji BMPConverter realizującej konwersję i kompresję pliku BMP do postaci tablicy języka C

Zasada działania wspomnianego algorytmu jest bardzo prosta, a pomimo tego, dla obrazków o dużych obszarach jednolitych kolorów, osiągany stopień kompresji jest całkiem spory. Wymownym dowodem powyższej tezy jest **rysunek 1**, na którym pokazano obrazek i wynikową tablicę elementów 16-bitowych, reprezentującą jego treść w przypadku stosowania jak i niestosowania wspomnianego algorytmu kompresji obrazu.

Listing 9. Funkcja odpowiedzialna za ustawienie bieżącej czcionki ekranowej

```
void OLEDsetFont(constfontDescription *Font)
{
    CurrentFont.Width = pgm_read_byte(&Font->Width);  //Szerokość czcionki
    CurrentFont.Height = pgm_read_byte(&Font->Height);  //Wysokość czcionki
    CurrentFont.Interspace = pgm_read_byte(&Font->Interspace);  //Odstęp
    CurrentFont.BytesPerChar = pgm_read_byte(&Font->BytesPerChar);  //Liczba bajtów na definicję pojedyńczego znaku
    CurrentFont.FirstCharCode = pgm_read_byte(&Font->FirstCharCode);  //Kod ASCII definicji pierwszego znaku
    CurrentFont.Bitmap = (uint8_t*)pgm_read_word(&Font->Bitmap);  //Wskaźnik do tablicy wzorców tej czcionki
}
```

Listing 10. Funkcja odpowiedzialna za rysowanie znaków, przy użyciu bieżącej czcionki ekranowej

```
void TFTdrawChar(uint16_t X1, uint8_t Y1, char Character, const uint8_t Transparency)
{
    register uint8_t widthIndex, heightIndex, readByte, pixelsNr, i;
    const uint8_t *dataPointer;
    if(Transparency != TRANSPARENT_TEXT)
    {
        //We define display active area to simplify writing
        TFTsetActiveWindow(X1, Y1, X1+CurrentFont.Width-1, Y1+CurrentFont.Height-1);
        //We start memory writing
        writeCommand(CMD_MEMORY_WRITE);
    }
    //Now we calculate start address of the current character definition
    dataPointer = &CurrentFont.Bitmap[(CurrentFont.BytesPerChar*(Character-CurrentFont.FirstCharCode))];
    for(heightIndex = 0; heightIndex < CurrentFont.Height; heightIndex++)
    {
        for(widthIndex = 0; widthIndex < CurrentFont.Width; widthIndex += 8)
        {
//We read character definition byte by byte
            readByte = pgm_read_byte(dataPointer++);
//For fonts which width is not a multiple of 8 we need to calculate
//useful number of pixels to be sent
            pixelsNr = widthIndex+8 <= CurrentFont.Width ? 8 : CurrentFont.Width - widthIndex;
            for(i=0; i<pixelsNr; ++i)
            {
//We check if the text background is transparent
                if(Transparency == TRANSPARENT_TEXT)
                {
//We check the pixel presence
                    if(readByte & 0x80)
                    {
//We define display active area for one active pixel to simplify writing
                        TFTsetActiveWindow(X1+widthIndex+i, Y1+heightIndex, X1+widthIndex+i, Y1+heightIndex);
                        writeCommand(CMD_MEMORY_WRITE);
                        writeData(Colour >> 8); writeData(Colour & 0xFF);
                    }
                }
                else
                {
//Pixel color depends on the pixel presence
                    if(readByte & 0x80) {writeData(Colour >> 8); writeData(Colour & 0xFF);}
                    else {writeData(Background >> 8); writeData(Background & 0xFF);}
                }
                readByte<<=1;
            }
        }
    }

}
```

Listing 11. Funkcje umożliwiające wyświetlenie ciągu znaków z pamięci RAM, jak i pamięci programu (Flash)

```
void TFTdrawString(uint16_t X1, uint8_t Y1, char *String, const uint8_t Transparency)
{
    while(*String)
    {
        TFTdrawChar(X1, Y1, *String++, Transparency);
        X1 += CurrentFont.Width + CurrentFont.Interspace;
    }
}

void TFTdrawString_P(uint16_t X1, uint8_t Y1, const char *String, const uint8_t Transparency)
{
    register char Character;
    while((Character = pgm_read_byte(String++)))
    {
        TFTdrawChar(X1, Y1, Character, Transparency);
        X1 += CurrentFont.Width + CurrentFont.Interspace;
    }
}
```

Listing 12. Sposób obliczania zmiennych przez oprogramowanie

```
spentFuelPer1s = ((1UL*Config.Cylinders*injectionTime*Config.CcPerMin)/2880UL);
Accu.spentFuel += spentFuelPer1s;
if(Accu.remainingFuel >= spentFuelPer1s) Accu.remainingFuel -= spentFuelPer1s;
Accu.Distance += ((1UL*WEGpulses*Config.Wheel) / (100UL*Config.PulsPerRot));
Speed = ((36UL*WEGpulses*Config.Wheel) / (1000UL*Config.PulsPerRot));
if(Speed<=5) Consum = ((5UL*Config.Cylinders*injectionTime*Config.CcPerMin) / 400000UL);
else Consum = ((5UL*Config.Cylinders*injectionTime*Config.CcPerMin*Config.PulsPerRot) / (144UL*WEGpulses*Config.Wheel));
SpeedAvg = ((36UL*Accu.Distance)/(10UL*Accu.Measurements));
if(Accu.Distance>999)
{
    ConsumAvg = (Accu.spentFuel/Accu.Distance);  //l/100km *10
    availableDistance = (((Accu.remainingFuel/1000)*(Accu.Distance/10)) / Accu.spentFuel)*10;  //km
}
else {ConsumAvg = 0; availableDistance = 0;}

/*
InjectionTime - sumaryczny czas wtrysku zliczony w czasie 1s [ms*48]
WEGpulses - liczba impulsów z przetwornika drogi zliczona w czasie 1 sekundy

spentFuelPer1s - paliwo spalone w czasie ostatniej sekundy [ul]
Accu.Measurements - akumulator liczby interwałów pomiarowych [s]
Accu.spentFuel - akumulator ilości spalonego paliwa [ul]
Accu.remainingFuel - akumulator ilości paliwa pozostającego w baku [ul]
Accu.Distance - akumulator przejechanego dystansu [m]
Consum - chwilowe zużycie paliwa [l*10/h], dla prędkości≤5 km/h lub [l*10/100km], dla prędkości>5 km/h
ConsumAvg - średnie zużycie paliwa [l*10/100km]
Speed - prędkość chwilowa [km/h]
SpeedAvg - prędkośc średnia [km/h]
availableDistance - orientacyjny, dostępny dystans na paliwie pozostającym w baku [km]

Config.CcPerMin - stała wtryskiwacza [ml/min]
Config.PulsPerRot - stała przetwornika drogi [imp/obr]
Config.Cylinders - liczba wtryskiwaczy paliwa
Config.Wheel - obwód opony [cm]
*/
```

Do „pełni szczęścia” brakuje nam dedykowanej aplikacji, za której pomocą dokonamy konwersji typowego pliku BMP do postaci tablicy języka C (o wspomnianej wcześniej organizacji danych) i która to dodatkowo umożliwi nam skompresowanie wyjściowego obrazka wedle powyższego algorytmu. Na szczęście napisanie takiej aplikacji nie jest rzeczą nazbyt skomplikowaną i nawet mi, mimo że dawno nie zajmowałem się językiem programowania innym, niż C, nie nastręczyło wielu problemów. Specjalnie na potrzeby tego projektu wykonałem aplikację, która realizuje wymaganą funkcjonalność (**rysunku 2**). Myślę, że przedstawione oprogramowanie jest na tyle proste i czytelne, iż nie wymaga dodatkowego słowa komentarza. Jedyne, czego potrzebujemy to funkcja, która umożliwia wyświetlenie na ekranie wyświetlacza skompresowanego obrazka – pokazano ją na **listingu 7**.

Uff, przyszedł czas na obsługę ostatniego elementu interfejsów graficznych, czcionek ekranowych! Aby jednak umożliwić wygodną obsługę wielu czcionek ekranowych, konieczne było wprowadzenie nowego typu danych, którego definicję pokazano na **listingu 8**. Bazując na zdefiniowanej powyżej strukturze, wprowadzono funkcję, która korzystając z globalnej zmiennej **static** *fontDescription* CurrentFont pozwala na ustawienie bieżącej czcionki ekranowej (**listing 9**). W tym momencie w końcu przyszedł czas na przedstawienie funkcji umożliwiającej rysowanie znaków, przy użyciu bieżącej czcionki ekranowej – **listing 10**. Ta funkcja korzysta z argumentu *uint8_t Transparency*, który to decyduje o tym, czy tło bieżącej czcionki ekranowej będzie określone wartością globalnej zmiennej *uint16_t* Background (wartość argumentu równa w takim przypadku **SOLID_TEXT**), czy też tło wyświetlanej czcionki będzie niejako przeźroczyste (wartość argumentu równa tym razem **TRANSPARENT_TEXT**)! Na bazie tejże funkcji wprowadzono dwie, nowe, proste funkcje, które umożliwiają wyświetlenie ciągu znaków z pamięci RAM, jak i pamięci programu (Flash) – pokazano je na **listingu 11**.

Robert Wołgajew, EP

HTTP://WWW.EP.COM.PL/KAP