Obsługa wyświetlacza TFT320240C256

Wydaje się, że minęły całe wieki od momentu, gdy alfanumeryczny wyświetlacz LCD ze sterownikiem HD44780 zaczął być powszechnie stosowany w interfejsach użytkownika urządzeń mikroprocesorowych. W rzeczywistości nie było to tak dawno i takie wyświetlacze są nadal bardzo popularne. Jednak w bardziej rozbudowanych interfejsach zaczęto stosować wyświetlacze graficzne. Zależnie od ceny były to albo małe wyświetlacze monochromatyczne i kolorowe od telefonów komórkowych lub większe – moduły monochromatyczne. Duże kolorowe wyświetlacze LCD wykonane w technologii TFT były drogie. Jak należało się spodziewać, rozwój technologii spowodował, że ceny spadły i warto zainteresować się wyświetlaczami o dużej rozdzielczości. Przykładem takiego produktu jest wyświetlacz kolorowy TFT320240C256 z matrycą TFT o rozdzielczości 320×240 pikseli.

Wbudowany sterownik wyświetlacza zapewnia uzyskanie 8-bitowej głębi kolorów. Ekran jest podświetlany za pomocą wbudowanych białych diod LED i ma zintegrowany panel dotykowy. Obszar wyświetlanych informacji ma wymiary 56 mm×73 mm.

Sterownik matrycy jest zbudowany w oparciu o układ Actel Igloo. Jest to programowany układ logiczny FPGA z pamięcią Flash. Producent wyświetlacza nie podaje właściwie żadnych technicznych szczegółów pracy sterownika poza opisem wyprowadzeń i przebiegów czasowych na magistrali sterującej. Do tych przebiegów dodano tabelkę z liczbowymi zależnościami czasowymi przy przesyłaniu danych. Zamiast klasycznej dokumentacji w danych technicznych umieszczono krótki program napisany w języku C dla mikrokontrolerów rodziny 8051. To dość nietypowe, ponieważ szczegółowa dokumentacja była tym, czego się spodziewałem i co w praktyce jest bardzo potrzebne. Jednak z drugiej strony, funkcje w C wyjaśniają właściwie wszystkie wątpliwości dotyczące sterowania wyświetlaczem na poziomie komunikacji z mikrokontrolerem. Ideałem byłoby połączenie opisu sterownika z przykładowymi procedurami. Jednak w końcowym efekcie analiza prostych procedur w C dała wyczerpująca odpowiedź na wszystkie wątpliwości i można uznać, że dokumentacja spełnia swoje zadanie.

Interfejs Komunikacyjny

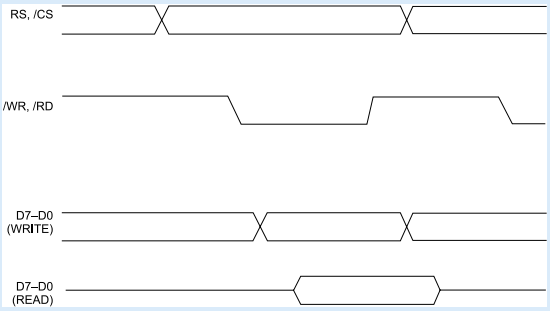
Sterownik wyświetlacza do komunikacji z mikrokontrolerem wykorzystuje równoległą 8-bitową magistralę pracującą w przemysłowym standardzie Intel 8080. Przebiegi czasowe na magistrali pokazano na **rysunku 1**. Interfejs jest aktywowany po wyzerowaniu linii CS. Kierunkiem przesyłu danych na magistrali sterują linie WR (zapis do sterownika) i RD (odczyt ze sterownika). Dane są wpisywane z mikrokontrolera do sterownika wyświetlacza przy narastającym zboczu na linii WR, a odczytywane z pamięci obrazu sterownika przy opadającym zboczu na linii RD. Stan dodatkowej linii sterującej RS określa czy dane są przesyłane do pamięci obrazu sterownika (RS=1), czy do rejestrów sterujących (RS=0).

Tabela 1. Połączenia magistrali sterującej

Port	Funkcja
GPIOB8	RESET (aktywny stan L)
GPIOB9	WR
GPIOB10	RD
GPIOB11	CS
GPIOB12	RS (oznaczona jako A0)
GPIO0….GPIOC7	D0….D7

Programowa obsługa wyświetlacza

Trzy 8-bitowe rejestry adresowe (RS=0) określają współrzędne *x*, *y* na ekranie wyświetlacza (rejestr adresu *x* jest 2-bajtowy bo zawiera liczbę z zakresu 0…. 319). Brak innych rejestrów oznacza, że nie trzeba sterownika wstępnie konfigurować. Wynika to pewnie z tego, że jest to rozwiązanie zoptymalizowane dla zastosowanej matrycy i wszystko zoptymalizowano pod kątem właściwości użytego wyświetlacza. Z drugiej strony, nie ma tu na przykład sprzętowego wsparcia wyświetlania ograniczonych bitmap. To wsparcie przydaje do wyświetlania znaków w trybie tekstowym. Jednak jak pokażę dalej, można sobie bez problemu z tym poradzić programowo.



Rysunek 1. Przebiegi czasowe na magistrali wyświetlacza.

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
R2	R1	R0	G2	G1	G0	B1	B0
MSB LSB			MSB LSB			MSB LSB	
RED (000~111)			GREEN(000~111)			BLUE(00~11)	

Rysunek 2. Format danej określającej kolor piksela

Listing 1. Inicjalizacja portu GPIOC
```
void BusInit(void)
{
GPIO_InitTypeDef  GPIO_InitStructure;
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,
ENABLE);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 |
GPIO_Pin_1 |GPIO_Pin_2 | GPIO_Pin_3 | GPIO_
Pin_4 | GPIO_Pin_5 |GPIO_Pin_6 | GPIO_Pin_7 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_
OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_
PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_
Speed_50MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_
NOPULL;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

Listing 2. Inicjalizacja linii sterujących
```
void CtrlInit(void)
{
   GPIO_InitTypeDef  GPIO_InitStructure;
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,
ENABLE);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 |
GPIO_Pin_9 |GPIO_Pin_10 | GPIO_Pin_11 | GPIO_
Pin_12 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_
OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_
PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_
Speed_50MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_
NOPULL;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

Listing 3. Wystawienie 8 bitowej danej na GPIOC
```
void BusWrite(uint8_t PortVal)
{
 GPIOC->ODR = PortVal;
```

Listing 4. Funkcje służące do manipulowania poziomami linii I/O
```
//definicje linii
GPIO_TypeDef* GPIO_PORT[LCDn] = {RST_GPIO_
PORT, WR_GPIO_PORT, RD_GPIO_PORT, CS_GPIO_
PORT, A0_GPIO_PORT};
const uint16_t GPIO_PIN[LCDn] = {RST_PIN, WR_
PIN, RD_PIN, CS_PIN, A0_PIN};

//ustaw port
void SetP(Port_TypeDef Port)
{
  GPIO_PORT[Port]->BSRR = GPIO_PIN[Port];
}
//zeruj port
void ClrP(Port_TypeDef Port)
{
  GPIO_PORT[Port]->BRR = GPIO_PIN[Port];
}
```

Listing 5. Funkcje zapisujące komendy i dane
```
//zapisz komende
void WrCmd(unsigned char cmd){
  ClrP(CS);
  SetP(RD);
  ClrP(A0); //RS=0
  BusWrite(cmd);
  ClrP(WR);
  SetP(WR);
  SetP(CS);
}

//zapisz dana
void WrData(unsigned char data){
  ClrP(CS);
  SetP(RD);
  SetP(A0); //RS=1
  BusWrite(data);
  ClrP(WR);
  SetP(WR);
  SetP(CS);
}
```

Do testów wyświetlacza użyłem modułu ewaluacyjnego STM32F0 Discovery. Jeżeli nie chcemy wyświetlać pełnowymiarowych bitmap, to jest to zestaw zupełnie wystarczający. Dla bitmap o pełnym wymiarze 240×320 pikseli potrzeba pamięci o pojemności 76800 bajtów i jest to więcej, niż całkowita pojemność pamięci Flash użytego mikrokontrolera. W aplikacjach wymagających wielu bitmap najlepiej jest je umieścić w zewnętrznej pamięci np. na karcie SD.

Obsługa magistrali jest emulowana programowo. Linie magistrali danych D0…D7 są połączone z liniami GPIOC0….GPIOC7. Jako linie sterujące są wykorzystano porty GPIOB8….GPIOB12 (**tabela 1**). Oprócz linii interfejsu komunikacyjnego niezbędny jest sygnał zerowania sterownika *RESET* (aktywny poziom niski).

Wszystkie linie portów w STM32 muszą być przed użyciem zainicjowane. Na **listingu 1** pokazano inicjalizację linii GPIO0….GPIOC7, a na **listingu 2** – inicjalizację linii sterujących.

Do emulowania magistrali będą potrzebne procedury wysyłania 8-bitowej danej na port (**listing 3**) i do manipulowania pojedynczymi wyjściami I/O (**listing 4**).

Teraz można napisać właściwe procedury emulowania magistrali. Będziemy potrzebowali funkcji zapisu do sterownika komendy *WrCmd* i zapisu danej *WrData*. Funkcje te można zobaczyć na **listingu 5**.

Funkcję ustalającą pozycję zapisu pozycji na ekranie (adres piksela) pokazano na **listingu 6**. Jej argumentami są współrzędne *x* (z zakresu 0…319; liczba 16-bitowa) i *y* (z zakresu 0…239; liczba 8 bitowa).

Nie ma tu zabezpieczeń przed przekroczeniem zakresu i użytkownik sam musi zadbać o poprawność określenia pozycji piksela.

Dana określająca kolor piksela ma format pokazany na **rysunku 2**. Kolor czerwony i zielony są kodowane na 3 bitach, a kolor niebieski na 2 bitach. Mając do dyspozycji funkcje zapisu danych i ustalenia pozycji na ekranie zapisywanej danej, można napisać funkcje zapisującą bitmapę do wyświetlacza. W Elektronice Praktycznej opisywałem już narzędzia i sposób konwertowania bitmap na tablice w języku C. W czasie testów użyłem programu *bmp2c.exe* do konwersji bitmapy z 8-bitową głębią koloru. Na **listingu 7** zamieszczono funkcję umożliwiającą wyświetlenie bitmapy o wymiarach 240×320 pikseli zapisanej w tablicy *bmp[]*.

Wynik działania tej procedury został pokazano na **fotografii 3**.

Listing 6. Ustalenie pozycji na ekranie
```
void WrAddr(unsigned int x, unsigned char y)
{
  unsigned char xh, xl;
  xh=x>>8;
  xl=x;
  WrCmd(xh);
  WrCmd(xl);
  WrCmd(y);
}
```

Listing 7. Wyświetlenie pełnowymiarowej bitmapy
```
void SendFullBmp(void)
{
  unsigned short x, y;
  for(y=0;y<240;y++)
  {
    WrAddr(0,y);
    for(x=0;x<320;x++)
    WrData(bmp[(y*320)+x]);
  }
}
```

Fotografia 3. Wyświetlenie pełnowymiarowej bitmapy

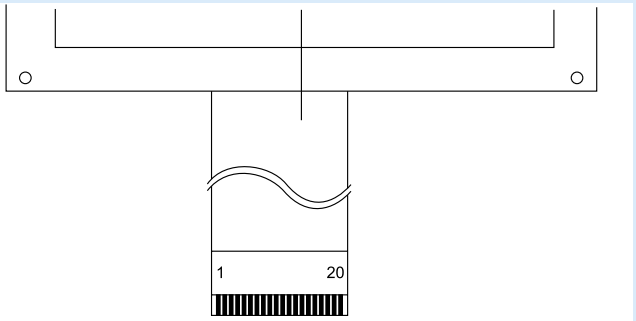
Fotografia 4. Wyświetlanie bitmapy o mniejszych wymiarach

Fotografia 5. Wyświetlacz w trybie tekstowym

Pełnowymiarowe bitmapy są używane rzadko. Dużo częściej trzeba wyświetlać mniejsze obiekty graficzne. Dlatego potrzebujemy procedur wyświetlających bitmapy o dowolnym rozmiarze i w dowolnym miejscu wyświetlacza. Oczywiście ta dowolność jest ograniczona rozdzielczością matrycy. Procedura z **listingu 8** wyświetla bitmapę od pozycji określonej w argumentach *x* i *y*, o wymiarach określonych w argumentach *dx* i *dy*. Ponieważ na ekranie można umieścić kilka bitmap, to argument bmp przekazuje wskaźnik tablicę z wyświetlaną bitmapą.

Ważnym elementem graficznego interfejsu użytkownika jest wyświetlanie informacji tekstowych. Jak napisano na początku, sterownik nie zapewnia żadnego wsparcia poza możliwością zapalenia/zgaszenia pojedynczego pikseli o zadanym kolorze. Dlatego wyświetlanie znaków alfanumerycznych trzeba wykonać całkowicie programowo. Zaczniemy od generatora znaków. Tablicę z wzorcami znaków można sobie samemu zdefiniować, ale jest to żmudne zajęcie. Ja wykorzystałem gotową tablicę wykonaną przez Jamesa P. Lyncha dla wyświetlacza telefonu Nokia 6100. Tablica zawiera wzorce znaków o 3 wielkościach: 6×8 pikseli (*small*), 8×8 pikseli (*medium*) i 8×16 pikseli (*large*). W praktyce, najbardziej przydatne okazały się największe znaki. Nic nie stoi na przeszkodzie, aby sobie zdefiniować jeszcze większe znaki.

Pierwsze 3 bajty w tablicy generatora dla każdej z wielkości znaków zawierają informację o liczbie kolumn, wierszy i bajtów na znak. Na tej podstawie jedna uniwersalna procedura potrafi wyświetlić znaki o różnych wielkościach. Procedurę umożliwiającą wyświetlanie pojedynczego znaku pokazano na **listingu 9**.



PIN	1	2	3	4	5	6	7	8	9	10
CONNECTOR	VSS	VDD	LED+	/RST	/WR	/RD	/CS	A0	D0	D1
PIN	11	12	13	14	15	16	17	18	19	20
CONNECTOR	D2	D3	D4	D5	D6	D7	XL	YD	XR	YU

Rysunek 6. Wyprowadzenia wyświetlacza

Listing 8. Wyświetlanie dowolnej bitmapy

```
void SendBmp(const unsigned char *bmp, short
x, short y, short dx, short dy)
{
  unsigned short i, j;
  for(i=y;i<y+dy;i++) //wyświetlanie wierszy
  {
    WrAddr(x,i);
    for(j=x;j<x+dx;j++)//wyświetlanie pikseli
w wierszu
    WrData(bmp[((i-y)*dx)+j-x]);
  }
}
```

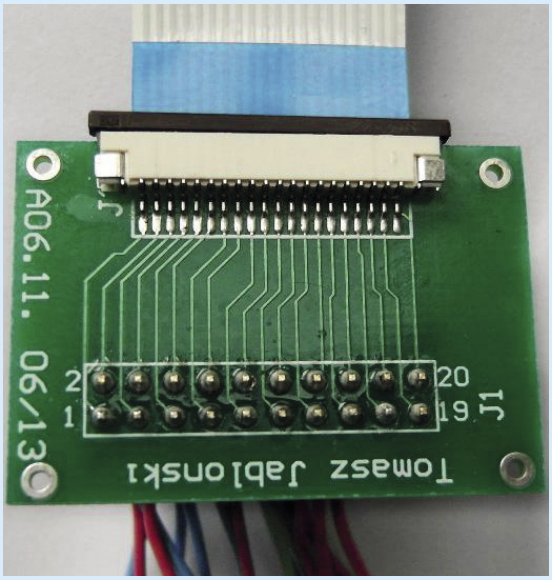
Listing 9. Procedura wyświetlająca pojedynczy znak

```
void LCDPutChar(char c, int x, int y, int
size, unsigned char  fColor, unsigned char
bColor) {
extern const unsigned char FONT6x8[97][8];
extern const unsigned char FONT8x8[97][8];
extern const unsigned char FONT8x16[97][16];
int i,j;
unsigned char nCols;
unsigned char nRows;
unsigned char nBytes;
unsigned char PixelRow;
unsigned char Mask;
unsigned char *pFont;
unsigned char *pChar;
unsigned char *FontTable[] = {(unsigned char
*)FONT6x8, (unsigned char *)FONT8x8,
(unsigned char *)FONT8x16};
//wskaznik na poczatek tablicy generatora
znakow
  pFont = (unsigned char *)FontTable[size];
  nCols = *pFont;                  //ilosc
kolumn wzorca znaku
  nRows = *(pFont + 1); //ilosc wierszy wzorca
znaku
  nBytes = *(pFont + 2);//ilosc bajtow wzorca
znaku
//wskaznik na ostatni bajt z wzorca naszego
znaku
  pChar = pFont + (nBytes * (c - 0x1F)) +
nBytes - 1;

//petla wykonywana dla kazdego wiersza
  for (i = nRows - 1; i >= 0; i--) {
//pobierz bajt z wzorca znaków
  PixelRow = *pChar--;
  WrAddr(x,y+i);
  Mask=0x80;
  for (j = 0; j < nCols; j ++) {
    if ((PixelRow & Mask) == 0)
    WrData (bColor);//jezeli piksel = 0 kolor
tla
    else
    WrData(fColor);//jezeli piksel = 1 kolor
znaku
    Mask=Mask>>1; }
  }
}
```

Listing 10. Wyświetlanie tekstu

```
void LCDPutStr(const char *pString, int x, int
y, int Size, char fColor, char  bColor) {
//w petli do znalezienia końca ciagu
  while (*pString != 0x00) {
//wyswietl znak alfanumeryczny
    LCDPutChar(*pString++, x, y, Size, fColor,
bColor);
//korekcja pozycji y w zaleznosci od wielkosci
znaku
    if (Size == SMALL)
    x = x + 6;
    else if (Size == MEDIUM) x = x + 8;
    else x = x + 8;
    if (x > 320) break;
  }
}
```



Fotografia 7. Płytka-przejściówka wykonana na potrzebę artykułu

Argumentami procedury są : kod ASCII wyświetlanego znaku, położenie na ekranie (współrzędne *x* i *y*), rozmiar znaku, kolor znaku i kolor tła. W zmiennej *pFont* jest umieszczany wskaźnik na początek tablicy generatora znaków. Do zmiennych *nCols*, *nRows* i *nBytes* są zapisywane pierwsze 3 bajty z tablicy generatora znaków. Ponieważ wzorce znaków nie są umieszczane w tablicy zgodnie z kodami ASCII, to trzeba kod ASCII wyświetlanego znaku przekształcić na pozycję w tablicy generatora. Po przekształceniu wskaźnik *pChar* wskazuje na ostatni bajt wzorca znaku w tablicy generatora. Program pobiera kolejne bajty z tablicy (od ostatniego do pierwszego). Każdy bit tego bajtu jest analizowany. Jeżeli jest ustawiony, to bit jest wyświetlany w kolorze znaku określonym argumentem *fColor*. Gdy bit jest wyzerowany, to jest wyświetlany w kolorze określonym argumentem *bColor*. W ten sposób można wyświetlać znaki o dowolnym kolorze na tle dowolnego koloru o głębi 8-bitowej.

Tryb tekstowy uzupełnimy procedurą wyświetlania tekstu w jednym z trzech rozmiarów(argument size). Pozycja początku ciągu znaków jest ustalana argumentami *x* i *y*, a kolor znaków oraz tła argumentami *fColor* i *bColor*.

Wynik działania tej funkcji pokazano na **fotografii 5**.

Sposób dołączenia wyświetlacza

Wszystkie sygnały elektryczne łącznie z zasilaniem sterownika, diod podświetlania i 4-rzewodowym wyprowadzeniem panelu dotykowego są dołączone 20-przewodowa taśmą zakończoną płaskimi złoconymi stykami. Podłączenie wyświetlacza do układu wymaga specjalnego złącza zaciskowego do styków kończących tasiemkę (**fotografia 6**). Do celów testowych zaprojektowałem płytkę-przejściówkę pozwalającą na połączenie wyprowadzeń wyświetlacza do modułu STM32F0 Discovery (**fotografia 7**).

Tomasz Jabłoński, EP