Animacje na LCD

Tworzenie animacji na wyświetlaczu LCD za pomocą STM32

Chociaż w handlu jest dostępnych coraz więcej kolorowych wyświetlaczy LCD, to jednak przeważnie nie są one przeznaczone na rynek masowy i dlatego są dosyć drogie. Z tego powodu ogromną popularnością cieszą się wyświetlacze z telefonów komórkowych, ponieważ są produkowane w ogromnych ilościach i dlatego dosyć tanie. Można je również łatwo pozyskać na rynku wtórnym, więc gdy zachodzi potrzeba zastosowania w urządzeniu niewielkiego wyświetlacza LCD w projekcie, warto rozważyć użycie takiego przeznaczonego dla telefonu komórkowego lub odtwarzacza multimedialnego. W artykule opisano sposób dołączenia i sterowania wyświetlaczem LCD od telefonu komórkowego Nokia E51.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 15031, pass: 40nep417

Wyświetlacz z telefonu Nokia E51 ma przekątną ekranu 2,2". Część jego powierzchni zajmuje ramka, więc obszar aktywny ma przekątną 51 mm, rozdzielczość 320×240 pikseli przy 24-bitowej głębi koloru. Wyświetlacz ten jest stosowany również w innych telefonach Nokii np.: 6500c, 6500, 6300/6301. 6120, 6555, 5310, 7310, 8600, 3120. Do podświetlania matrycy służą cztery białe diody LED. Dzięki ich połączeniu szeregowemu uzyskuje się tę samą jasność ich świecenia. W mojej aplikacji są one zasilane napięciem 12 V. Na **rysunku 1** pokazano rozmieszczenie wyprowadzeń wyświetlacza.

Opis interfejsu wyświetlacza

Sterownikiem opisywanego wyświetlacza jest Leadis LDS285. Pomimo wsparcia dla wielu interfejsów, w LCD od Nokii E51 sterownik jest na stałe ustawiony w trybie komunikacji za pomocą interfejsu równoległego w standardzie Intel 8080 o magistrali 8-bitowej. Do poprawnej komunikacji wymagane są sygnały: DC, WRB, RDB.

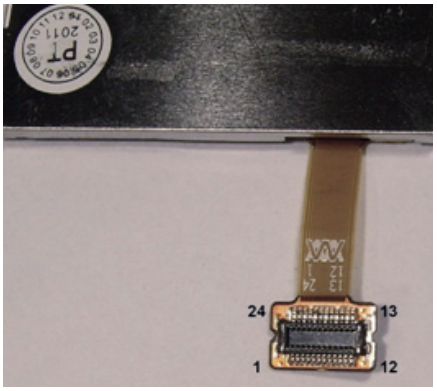
Wejście DC informuje sterownik wyświetlacza czy wysłano komendę, czy dane. Wyzerowanie wejścia DC oznacza komendę, natomiast ustawienie – dane. Wejście WRB służy do zatrzaskiwania danych D0…D7 – aktywne jest zbocze narastające. Aby wysłać do wyświetlacza komendę lub dane, najpierw należy wyzerować linię WRB, przesłać bajt na magistralę, odpowiednio ustawić stan wejścia DC, a następnie zmienić poziom WRB z niskiego na wysoki. Wejście RDB zezwala na odczyt danych z układu. Istotną rolę pełni również wejście CSB. Służy ono do aktywowania układu. Aktywny jest poziom niski.

Szczegóły nt. funkcjonowania można znaleźć w dokumentacji sterownika LDS285. Bez trudu można ją znaleźć w Internecie.

Wyświetlacz jest zasilany napięciami 1,8 V i 2,5 V. W aplikacji uzyskuje się je dzięki zastosowaniu dwóch stabilizatorów: **TC1185-1.8VCT** i **TC1185-2.5VCT**. Napięcie 1,8 V zasila układy cyfrowe, zatem przy sterowaniu za pomocą mikrokontrolera zasilanego wyższym napięciem, należy wykonać odpowiednie układy dopasowujące poziomy napięć. W naszym wypadku: 3,3 V <–> 1,8 V. Tę rolę pełnią dwa układy typu 74LVC4245AD.

Sterowanie wyświetlaczem

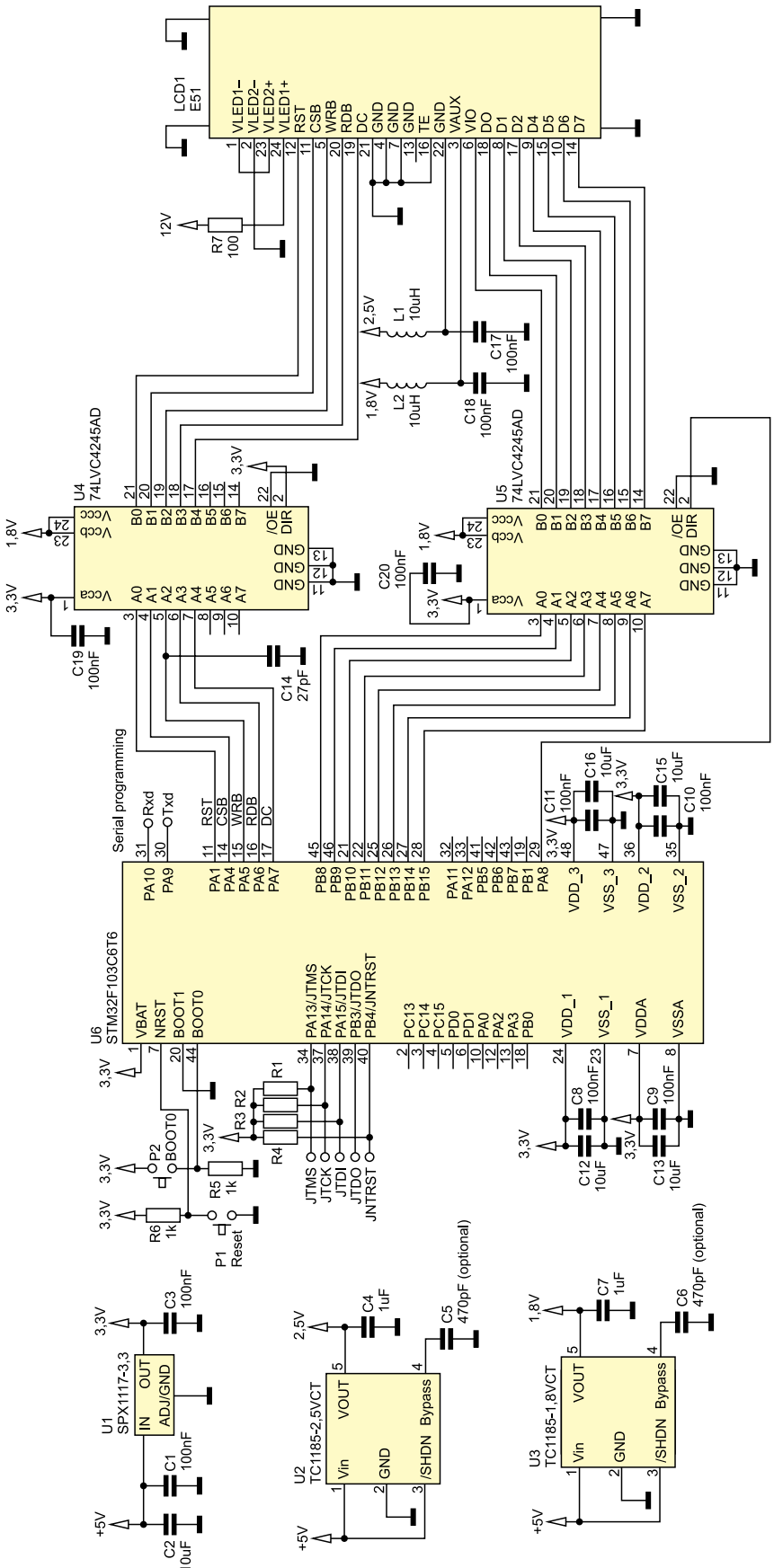
Obsługę wyświetlacza można wykonać na dwa sposoby. W pierwszym trzeba samodzielnie generować sygnały interfejsu Intel 8080 (emulować go) wykorzystując



Nr wyprowadzenia	Funkcja
1	VLED1–
2	VLED2–
3	VIO
4	GND
5	WRB
6	D0
7	GND
8	D2
9	D4
10	D6
11	CSB
12	RST
13	TE
14	D7
15	D5
16	GND
17	D3
18	D1
19	DC
20	RDB
21	GND
22	VAUX
23	VLED2+
24	VLED1+

Rysunek 1. Rozmieszczenie wyprowadzeń wyświetlacza od telefonu Nokia E51

porty GPIO. Użyjemy do tego mikrokontroler STM32F103C6T6 (CORTEX M3). Jest to stosunkowo wolne sterowanie, ponieważ maksymalna częstotliwość odświeżania, którą udało się osiągnąć podczas testów to 1,5 klatki na sekundę, przy czym „klatka" oznacza wypełnienie całej powierzchni wyświetlacza jednakowym kolorem. Adekwatny kod nie powinien sprawiać trudności przy przenoszeniu go na inną platformę sprzętową. Jako ciekawostkę dodam, że wypełnienie całego ekranu jednym kolorem można przyspieszyć dzięki sterowaniu pinu WRB nie z portu GPIO, a z sygnałów zegarowych np. z szybkich peryferii bądź generatorów sygnałów.



Rysunek 2. Schemat ideowy sterownika wyświetlacza Nokii E51

W drugiej metodzie LCD jest sterowany przez układ FSMC dostępny w mikrokontrolerze STM32F103VCT6. Dzięki temu udaje się uzyskać częstotliwość odświeżania 12,5 klatki na sekundę. FSMC (Flexible Static Memory Controller) jest układem peryferyjnym, niedostępnym w tańszych wersjach mikrokontrolerów STM32. Zwykle służy on do obsługi pamięci: SRAM, ROM, NOR, PSRAM, NAND, 16-bit PC Card. Jednak jego możliwości są na tyle duże, że z powodzeniem można go użyć także do sterowania wyświetlaczem z interfejsem Intel 8080.

Sterowanie LCD za pomocą emulowanie interfejsu Intel 8080. Schemat ideowy modelowego układu sterującego za pomocą emulowania interfejsu Intel 8080 pokazano na **rysunku 2**. Sterowanie wyświetlaczem odbywa się przez wysyłanie do niego komend (*LCD_write_command*) z parametrami (*LCD_write_data*). Nie wszystkie komendy wymagają parametrów. W **tabeli 1** zamieszczono komendy sterownika LDS285. Ten jest ograniczony jedynie do objaśnienia instrukcji umożliwiających podstawowe operacje związane z wyświetlaniem obrazów. Więcej informacji należy szukać w dokumentacji sterownika.

Funkcja *LCD_write_parallel* służy do przesłania liczby na magistralę 8-bitową. O sposobie jej interpretacji informują wcześniej wspomniane sygnały sterujące. Dana jest dostępna na wyprowadzeniach PB8… PB15 mikrokontrolera. Niezależnie od stosowanego mikrokontrolera nie polecam używania pinów należących do różnych portów lub podłączonych przypadkowo, ponieważ związane z tym dodatkowe operacje zabiorą czas mikrokontrolera i tym samym będą miały istotny wpływ na szybkość pracy wyświetlacza.

Funkcje *set_LCD* i *clear_LCD* są kontrolują stan wyprowadzeń połączonych z wejściami wyświetlacza: LCD_RST, LCD_CS, LCD_WR, LCD_RD, LCD_DC. Funkcja *LCD_hard_reset()* realizuje zerowanie sprzętowe wyświetlacza LCD. Minimalny czas trwania poziomu niskiego linii RST powinien wynosić 10 μs. Procedura restartu kontrolera LCD trwa około 120 ms.

LCD_write_command i *LCD_write_data* służą do wysłania komendy bądź danych. W ciele tych funkcji, na początku, jest uaktywniany wyświetlacz niskim poziomem linii CSB. Następnie, w zależności od rodzaju danych, jest sterowania linia DC: w wypadku komendy jest ona zerowana, dla danych jest ona ustawiana. Następnie jest ustawiana linia RDB i dane są zatrzaskiwane narastają-

cym zboczem sygnału WRB. Po przesłaniu danych, linia WRB jest zerowana. Funkcje zapisu komendy i danej pokazano na **listingu 1**.

Funkcja *LCD_init* realizuje inicjalizuje sterownik wyświetlacza. Zamieszczono ją na **listingu 2**. Wykonuje ona zerowanie sprzętowe sterownika wyświetlacza (*LCD_hard_reset*), przesyła komendę *SWRESET* wywołującą tzw. miękki restart sterownika. Podczas niego są przywracane wartości domyślne rejestrów, a wyświetlacz jest wyłączany. „Miękkie" zerowanie trwa ok. 5 ms. Następnie komenda *SLPOUT* wyprowadza wyświetlacz z trybu *sleep mode*. Włączany jest konwerter DC/DC, uruchamiane są wewnętrzny oscylator wyświetlacza i panel skanujący. Wszystko to trwa kolejne 5 ms. W tym czasie sterownik wykonuje również diagnostykę. Ostatnim krokiem inicjalizacji jest wysłanie komendy *DISPON*. Włącza ona wyjście pamięci DDRAM, co skutkuje pojawieniem się różnych kolorowych pikseli na ekranie. Zdarza się, że niektóre egzemplarze LCD wyświetlają równe paski. Są to przypadkowe wartości pamięci ustalone po włączeniu LCD. Od tego momentu wyświetlacz jest gotowy do pracy.

Teraz dobrze jest wyczyścić obszar wyświetlania. Polega to na wypełnieniu pamięci obrazu stałą np. dla białego koloru. W przykładzie wykorzystano do tego celu zamieszczoną na **listingu 3** funkcję *LCD_paint*. Najlepiej ją wywołać zaraz po inicjalizacji, by „nieeleganckie", różnokolorowe piksele nie były widoczne. Argumentem funkcji jest liczba odpowiadająca kolorowi. Powinna ona zawierać 3 bajty, znaczenie poszczególnych bitów pokazano na **rysunku 3**.By móc wypełnić cały obszar wyświetlania jednolitym kolorem najpierw trzeba wysłać komendę *RAMWR*. Parametrem tej komendy są dane odpowiadające za kolor pikseli. Dzięki niej wysyłane dane aktualizują wewnętrzną pamięć DDRAM w sterowniku, czego skutkiem jest natychmiastowa zmiana koloru zapisywanych pikseli. Po wysłaniu tej komendy początkowa wartość liczników kolumn i wierszy pikseli jest przywracana do wartości początkowych. W tym wypadku, gdy inne rejestry nie zostały jeszcze zmienione, wypełnianie matrycy następuje od 0 wiersza i kolumny aż po 319 wiersz i 239 kolumnę, w kolejności jak na **rysunku 4**. Jeżeli wyślemy więcej potrójnych bajtów odpowiadających za jeden piksel, niż mamy dostępnych pikseli w wyświetlaczu, liczniki wierszy i kolumn przyjmą wartości początkowe (0 wiersz, 0 kolumna). Liczba bajtów danych jest równoważna iloczynowi rozdzielczości pionowej i poziomej LCD i głębi koloru (320×240×3). Jako pierwszy zawsze wysyłany jest kolor czerwony, potem zielony i na końcu niebieski.

Pokazana na **listingu 4** funkcja *LCD_flash* została wykonana w celu zaprezentowania najszybszego, możliwego z użyciem danej metody, odświeżania ekranu wyświetlacza, na przemian wypełniając całą matrycę trzema kolorami. Jej działanie jest tak same jak funkcji *LCD_paint*. Jak widać, wystarczy tylko raz wysłać komendę RAMWR.

Funkcja *put_pixel* (**listing 5**) ilustruje zasadę wypełniania części matrycy. Jej parametrami wejściowymi są współrzędne i kolor piksela, który chcemy umieścić. Komendy CASET i RASET pozwalają ustalić

Tabela 1. Wykaz komend sterujących wyświetlacza LCD Nokii E51

Instrukcja	Kod (notacja języka C)	Opis
NOP	0x00	No Operation
SWRESET	0x01	Software reset
RDDID	0x04	Read Display ID
RDDST	0x09	Read Display Status
RDDPM	0x0A	Read Display Power Mode
RDDMADCTR	0x0B	Read Display MADCTR
RDDCOLMOD	0x0C	Read Display Pixel Format
RDDIM	0x0D	Read Display Image Mode
RDDSM	0x0E	Read Display Signal Mode
RDDSDR	0x0F	Read Display Self-diagnostic result
SLPIN	0x10	Sleep in & booster off
SLPOUT	0x11	Sleep out & booster on
PTLON	0x12	Partial mode on
NORON	0x13	Partial off (Normal)
INVOFF	0x20	Display inversion off (normal)
INVON	0x21	Display inversion on
GAMSET	0x26	Gamma curve select
DISPOFF	0x28	Display off
DISPON	0x29	Display on
CASET	0x2A	Column address set
RASET	0x2B	Row address set
RAMWR	0x2C	Memory write
RAMRD	0x2E	Memory read
PTLAR	0x30	Partial start/end address set
TEOFF	0x34	Tearing effect line off
TEON	0x35	Tearing effect mode set & on
MADCTR	0x36	Memory data access control
IDMOFF	0x38	Idle mode off
IDMON	0x39	Idle mode on
COLMOD	0x3A	Interface pixel format
WRDISBV	0x51	Write Display Brightness
RDDISBV	0x52	Read Display Brightness value
WRCTRLD	0x53	Write Control Display
RDCTRLD	0x54	Read Control Display
WRCABC	0x55	Wrtie Content Adaptive Brightness
RDCABC	0x56	Read Content Adaptive Brightness
RDID1	0xDA	Read ID1
RDID2	0xDB	Read ID2
RDID3	0xDC	Read ID3
IFMODE	0xB0	Set display interface mode
DISCLK	0xB1	Display clock set
INVCTR	0xB2	Display inversion control
REGCTR	0xC0	Regulator control
VCOMCTR	0xC1	VCOML/VCOMH voltage control
GAMCTR1	0xC8	Set gamma correction characteristics
GAMCTR2	0xC9	Set gamma correction characteristics
GAMCTR3	0xCA	Set gamma correction characteristics
GAMCTR4	0xCB	Set gamma correction characteristics
EPPGMDB	0xD0	Write ID2,VCOM Offset for EEPROM program
EPERASE	0xD1	EEPROM erase
EPPROG	0xD2	EEPROM program
EPRDVRF	0xD3	EEPROM read, verify register set
RDVCOF	0xD9	VCOM offset bits read
LEDCTRL	0xEF	Write LED control value

Listing 1. Funkcje zapisu komendy i danych do wyświetlacza

```
void LCD_write_command(unsigned char cmd){
  clear_LCD(LCD_CS);              //choose LCD
  clear_LCD(LCD_DC);              //write command
  set_LCD(LCD_RD);                //page 19 in datasheet
  clear_LCD(LCD_WR);              //low on WRB pin
  LCD_write_parallel(cmd);        //set 8bit parallel data
  set_LCD(LCD_WR);                //high on WRB pin
}

void LCD_write_data(unsigned char cmd){
  clear_LCD(LCD_CS);              //choose LCD
  set_LCD(LCD_DC);                //write data
  set_LCD(LCD_RD);                //page 19 in datasheet
  clear_LCD(LCD_WR);              //low on WRB pin
  LCD_write_parallel(cmd);        //set 8bit parallel data
  set_LCD(LCD_WR);                //high on WRB pin
}
```

Listing 2. Funkcja inicjalizująca sterownik wyświetlacza
```
void LCD_init(){
  //delay_ms(250);                 //optional - wait for stabilization lcd
                                   //power
  LCD_hard_reset();
  LCD_write_command(SWRESET);      //software reset
  delay_ms(5);                     //delay 5ms
  LCD_write_command(SLPOUT);       //This command turns off sleep mode
  delay_ms(5);                     //delay 5ms
  LCD_write_command(DISPON);       //Display On
}
```

Listing 3. Funkcja wypełniająca obszar wyświetlania kolorem
```
void LCD_paint(unsigned int color){
  unsigned int i;
  LCD_write_command(RAMWR);
  for(i=0; i<320*240; i++){
    LCD_write_data((unsigned char)color);
    LCD_write_data((unsigned char)(color >> 8));
    LCD_write_data((unsigned char)(color >> 16));
  }
}
```

Rysunek 3. Rozmieszczenie bajtów kolorów w słowie-argumencie funkcji *LCD_paint*

początkowe i końcowe wartość kolumn oraz wierszy. Obie komendy wymagają podania 4 parametrów. Jeżeli wartości tych rejestrów nie będą zmieniane, za każdym razem po wysłaniu komendy RAMWR, cała matryca będzie wypełniana. Komenda CASET ustala numer kolumny początkowej i końcowej, jak pokazano na **rysunku 4**. Argumentami wywołania są dwa bajty wartości początkowej i dwa bajty wartości końcowej. Jako pierwsze podaje się starszy i młodszy bajt wartości początkowej, następnie starszy i młodszy bajt wartości końcowej. Komenda RASET umożliwia ustalenie startowej i końcowej wartości wierszy, podobnie jak CASET, jednak w kierunku pionowym. Jak z poprzednio, jako pierwsze argumenty podaje się starszy i młodszy bajt wartości początkowej, następnie starszy i młodszy bajt wartości końcowej.

W przypadku omawianej funkcji *put_pixel* wartości początkowe i końcowe są takie same, odpowiednio dla kolumn i wierszy, ponieważ naszym celem jest zaświecenie pojedynczego piksela. Po ustaleniu współrzędnych piksela wysyłamy 3 bajty danych odpowiedzialnych za jego kolor, poprzedzając je komendą RAMWR. Jeżeli chcemy zmienić kolor większego fragmentu ekranu niż jeden piksel, należy podać nierówne sobie wartości początkowe i końcowe dla wierszy i kolumn.

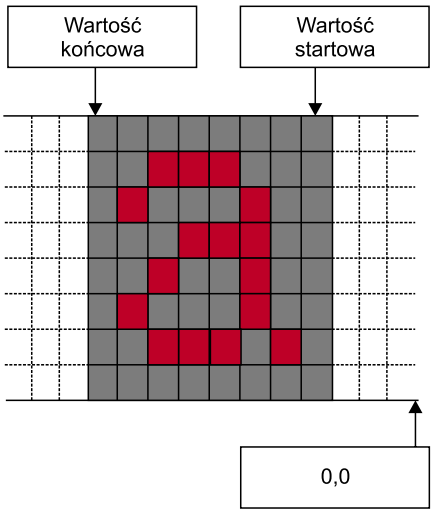
Listing 4. Funkcja powodująca migotanie wyświetlacza
```
void LCD_flash(){
  unsigned int i;
  LCD_write_command(RAMWR);
  while(3){
    for(i=0; i<320*240; i++){
      LCD_write_data(0xFF);
      LCD_write_data(0x00);
      LCD_write_data(0x00);
    }
    for(i=0; i<320*240; i++){
      LCD_write_data(0x00);
      LCD_write_data(0xFF);
      LCD_write_data(0x00);
    }
    for(i=0; i<320*240; i++){
      LCD_write_data(0x00);
      LCD_write_data(0x00);
      LCD_write_data(0xFF);
    }
  }
}
```

Listing 5. Wyświetlanie pojedynczego piksela
```
void put_pixel(unsigned int x, unsigned int y, unsigned int color){
  LCD_write_command(CASET);
  LCD_write_data((unsigned char)((x>>8) & 0x000000FF));
  LCD_write_data((unsigned char)(x & 0x000000FF));
  LCD_write_data((unsigned char)((x>>8) & 0x000000FF));
  LCD_write_data((unsigned char)(x & 0x000000FF));
  LCD_write_command(RASET);
  LCD_write_data((unsigned char)((y>>8) & 0x000000FF));
  LCD_write_data((unsigned char)(y & 0x000000FF));
  LCD_write_data((unsigned char)((y>>8) & 0x000000FF));
  LCD_write_data((unsigned char)(y & 0x000000FF));
  LCD_write_command(RAMWR);
  LCD_write_data((unsigned char)(color >> 16));
  LCD_write_data((unsigned char)(color >> 8));
  LCD_write_data((unsigned char)(color));
}
```

Listing 6. Inicjalizacja jednostki FSMC
```
void FSMC_NAND_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  FSMC_NANDInitTypeDef FSMC_NANDInitStructure;
  FSMC_NAND_PCCARDTimingInitTypeDef  p;
  RCC_APB2PeriphClockCmd(
    RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|
    RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|
    RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|
    RCC_APB2Periph_GPIOG,ENABLE);
/* CLE, D0-D3, NOE, NWE - NAND pin configuration*/
  GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_11|GPIO_Pin_14|GPIO_Pin_15|
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_4|GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
/* D4->D7 - NAND pin configuration  */
  GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
  /* FSMC Configuration */
  p.FSMC_SetupTime = 0x1;
  p.FSMC_WaitSetupTime = 0x2;
  p.FSMC_HoldSetupTime = 0x1;
  p.FSMC_HiZSetupTime = 0x0;
  FSMC_NANDInitStructure.FSMC_Bank = FSMC_Bank2_NAND;
  FSMC_NANDInitStructure.FSMC_Waitfeature = FSMC_Waitfeature_Enable;
  FSMC_NANDInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_8b;
  FSMC_NANDInitStructure.FSMC_ECC = FSMC_ECC_Disable;
  FSMC_NANDInitStructure.FSMC_ECCPageSize = FSMC_ECCPageSize_8192Bytes;
  FSMC_NANDInitStructure.FSMC_TCLRSetupTime = 0x00;
  FSMC_NANDInitStructure.FSMC_TARSetupTime = 0x00;
  FSMC_NANDInitStructure.FSMC_CommonSpaceTimingStruct = &p;
  FSMC_NANDInitStructure.FSMC_AttributeSpaceTimingStruct = &p;
  FSMC_NANDInit(&FSMC_NANDInitStructure);
  /* FSMC NAND Bank Cmd Test */
  FSMC_NANDCmd(FSMC_Bank2_NAND, ENABLE);
}
```

Sterowanie LCD za pomocą FSMC

Na **rysunku 5** pokazano schemat połączeń dla sterowania wyświetlaczem za pomoca FSMC. W tym wypadku, inaczej niż poprzednio, nie wolno zmienić połączeń



Rysunek 4. Zasada działania komendy CASET

pomiędzy mikrokontrolerem a LCD, ponieważ do obsługi wyświetlacza użyto FSMC, które dołącza się do układów zewnętrznych w ściśle określony sposób. Dzięki użyciu jednostki FSMC nie dosyć, że wzrasta prędkość obsługi wyświetlacza, to jeszcze zwalnia nas ona od konieczności programowej emulacji interfejsu Intel 8080. Funkcje: *LCD_write_command*, *LCD_write_data* i *LCD_write_parallel* nie są już potrzebne.

Środowisko programistyczne, którego używam to *Cortex-IAR Embedded Workbench for ARM*. Wraz z nim producent dostarcza mnóstwo programów przykładowych. Wśród przykładów obsługi FSMC udostępniono procedury obsługi pamięci NOR, SRAM, SRAM i NAND. Dla nas najbardziej użytecznym jest przykład prezentujący sposób sterowania pamięcią NAND Flash. Na **listingu 6** zamieszczono zmodyfikowaną procedurę inicjalizacji FSMC, która pierwotnie służyła do obsługi pamięci NAND Flash, a teraz można posłużyć się nią do obsługi wyświetlacza LCD z telefonu Nokia E51.

Na początku są włączane sygnały zegarowe, które będą występowały na wyprowadzeniach GPIO. Następnie są konfigurowane poszczególne piny mikrokontrolera. My użyjemy – zgodnie z nazewnictwem interfejsu pamięci NAND – tylko wyprowadzeń oznaczonych CLE (=LCD_DC), NOE (=LCD_RDB), NWE (=LCD_WRB) i 8-bitowej szyny danych D0…D7. W dalszej kolejności są ustawiane parametry czasów dostępu do pamięci (bo tak będzie „widziany" wyświetlacz), aby zwiększyć szybkość komunikacji z wyświetlaczem LCD, który jest szybszy od pamięci Flash. Teraz są ustawiane parametry samego FSMC, tutaj najważniejszym jest 8-bitowa szerokość magistrali. Na koniec, następuje ustawienie rejestrów zgodnie z wcześniej wypełnioną strukturą i włączenie drugiego banku pamięci NAND.

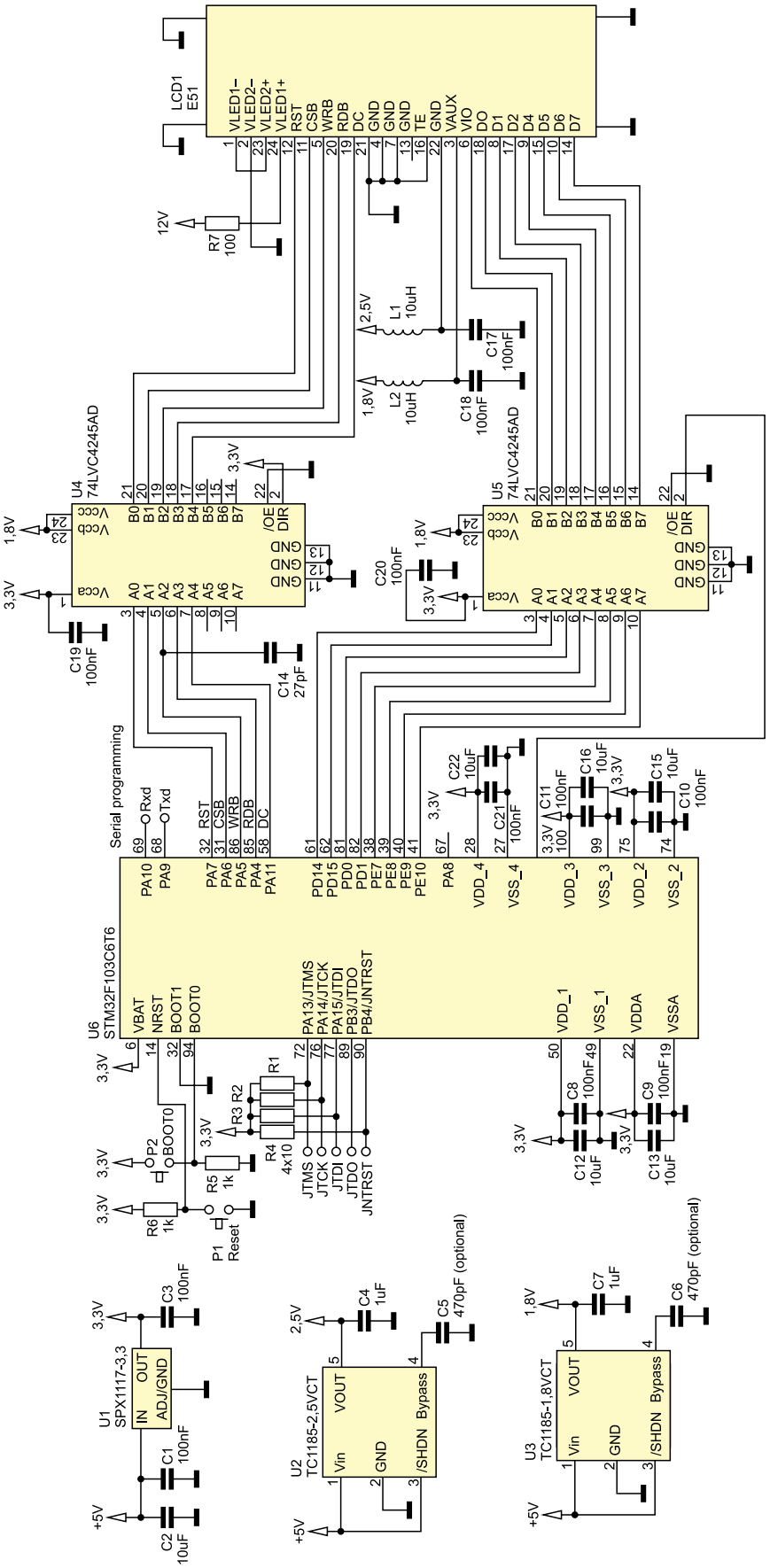
W nowym programie obsługi, który wykonano odpowiednio modyfikując prezentowany wcześniej, oprócz usunięcia wspomnianych funkcji, wprowadzono nowe definicje:

```
#define LCD_WRITE_COMMAND*(vu8*)
(Bank_NAND_ADDR|DATA_AREA)
#define LCD_WRITE_DATA*(vu8*)
(Bank_NAND_ADDR|CMD_AREA.
```

Zastępują one funkcje *LCD_write_command*, *LCD_write_data*, które służyły do wysyłania komend lub danych. W wypadku wysłania komendy bajt będzie zapisany pod inny adres niż w wypadku wysłania danych. W ten nieskomplikowany sposób jednostka FSMC może odróżnić komendę od danej i odpowiednio sterować sygnałem CLE. Wyprowadzenie CLE jest połączony z pinem wyświetlacza DC. Jeżeli chcemy wysłać np. komendę

```
LCD_WRITE_COMMAND = SWRESET;
```

FSMC zapisuje ją pod adresem danych zeruje wyprowadzenie CLE (*Command Latch Enable*). Dlaczego zeruje? Jednostkę FSMC skonfigurowałem w taki sposób, aby jak najbardziej uprościć program obsługi, jednak – jak pamiętamy – posłużyłem się przykładem sterownika pamięci NAND Flash. W wypadku jej obsługi zapis komendy następuje,



Rysunek 5. Schemat ideowy obsługi wyświetlacza Noki E51 przez FSMC

Listing 7. Funkcja wyświetlająca napisy

```
//przykład użycia: putc_enter (4,160 , czarny, "Dariusz Rzedowski");
#include "fonts.h"                                                    //tablica ze znakami
void putc_enter(unsigned char x, unsigned char y, unsigned int color, unsigned char *napis){
  int i,j, k=0, l=0, u=-1;
  LCD_WRITE_COMMAND = MADCTR;
  LCD_WRITE_DATA = ((unsigned char)(0x40));
  for(k=0; k<2000;k++){                                               //maksymalna liczba liter
                                                                      //na ekranie
    if(napis[k] == 0) break;                                          //jeżeli koniec tablicy
    u++;  //pozycja x-owa
    if((x+6*u) > 230){                                                //sprawdzenie czy juz
                                                                      //trzeba „enter"
      y=y-10;                                                         //"enter"
      u=0;                                                            //zerowanie pozycji
                                                                      //x-owych
    }
    l = (napis[k] - 32)*5;
    for(i=0;i<5;i++)
      for(j=0;j<8;j++)
        if(((fonts[l+i]>>j) & 0x01) == 1) put_pixel(x+i+6*u, y-j, color);
  }
}
```

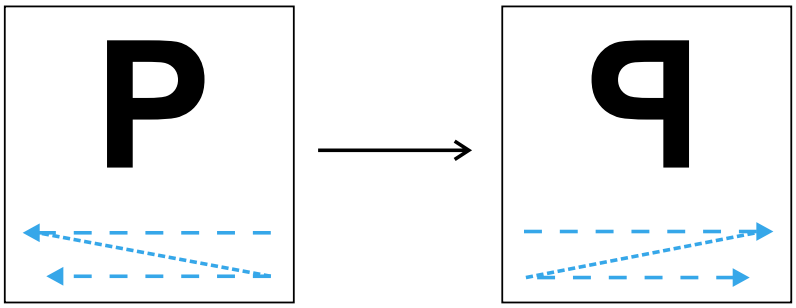
gdy sygnał CLE ma poziom wysoki. Zapis danych lub adresu do pamięci NAND jest wykonywany, gdy sygnał CLE ma poziom niski (pomijam poziomy innych wyprowadzeń interfejsu pamięci NAND Flash, ponieważ są one istotne). W wyświetlaczu, jeżeli wyprowadzenie DC połączone z wejściem CLE wyświetlacza ma poziom niski, to zapisywane są komendy. Stąd na pozór w definicjach błędne przypisanie nazwy LCD_WRITE_COMMAND pod adres DATA_AREA.

Jeżeli chcemy wysłać daną, to należy wykonać przypisanie

```
LCD_WRITE_DATA = ((u8)0x1A);
```

Wtedy liczba 0x1A zostanie zapisana pod adres komend w FSMC, co będzie oznaczało ustawienie sygnału CLE, ponieważ właśnie tak wysyła się komendę do pamięci NAND Flash. Jeżeli CLE (czyli pin DC) jest jedynką, w wypadku obsługi LCD oznacza to wysyłanie danych.

Podczas pracy nad uruchomieniem obsługi wyświetlacza za pomocą jednostki FSMC długo szukałem błędu powodującego wysyłanie dwóch bajtów zamiast jednego. Rozwiązaniem okazało się rzutowanie zmiennych typu *unsigned char* przy zapisie danych i komend. Jeżeli w konfiguracji FSMC jest ustawione słowo 8-bitowe a używane są 16-bitowe zmienne typu *unsigned short*, to FSMC automatycznie wysyła dwa bajty. Jest to wygodne rozwiązanie przy pracy z 16-bitowymi słowami danych, ale nasze słowa są 8-bitowe.



Rysunek 6. Działanie komendy MADCTR

Funkcja wyświetlająca napisy

Funkcja widoczna na **listingu 7** wyświetla napisy na ekranie LCD. Rysowanie znaków zrealizowane jest za pomocą wcześniej opisanej funkcji *put_pixel*. Funkcja automatycznie przenosi wyrazy do kolejnego wiersza, gdy znaki je tworzące nie mieszą się. Pierwotnie funkcja była przeznaczona do obsługi innego wyświetlacza, w którym współrzędne zerowego piksela znajdowały się w lewym dolnym rogu, a nie jak w E51 – w prawym dolnym. Można uwzględnić to modyfikując funkcję, ale można też wysłać komendę *MADCTR*, która służy do ustalenia kierunku skanowania wewnętrznej pamięci DDRAM wyświetlacza. W praktyce, zgodnie z podanym parametrem, powoduje to przerzucenie obrazu w pionie (**rysunek 6**).

Podsumowanie

W artykule zaprezentowano podstawy obsługi wyświetlacza z telefonu Nokia E51 będące bazą do wykonania programu wyświetlającego animacje, który będzie zaprezentowany w kolejnym numerze Elektroniki Praktycznej. Niestety, aby biegać trzeba nauczyć się najpierw chodzić, a to zajmuje troszkę czasu. Nie jest to jednak czas zmarnowany.

Dariusz Rzędowski
dariusz.rzedowski@gmail.com