POLITECHNIKA CZĘSTOCHOWSKA, **WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI**
KATEDRA INŻYNIERII KOMPUTEROWEJ

Laboratorium Systemów Wbudowanych

Ćwiczenie nr 2:

TEMAT: Sterowanie pracą wyświetlacza LCD w mikrokontrolerze AT91SAM7X256

1. Wprowadzenie

Miliony telefonów komórkowych firmy Nokia zostały wyposażone w wyświetlacze typu: Nokia 6100. Wyświetlacze te, co prawda, zostały zastąpione przez nowe, większe, o wyższej rozdzielczości, ale dostępność i cena tych starych powoduje, że cieszą się one dużą popularnością dla hobbystów i eksperymentatorów.
Podstawowe parametry tego wyświetlacza to:

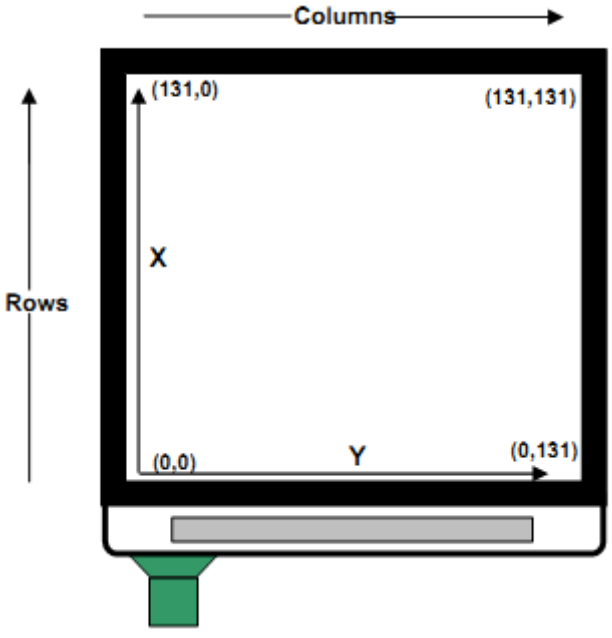
- rozdzielczość 132x132
- 12-bitowa rozdzielczość kolorów (4 dla czerwonego, 4 dla zielonego i 4 dla niebieskiego)
- zasilanie 3,3 V
- 9-bitowy interfejs SPI

Pierwszą trudnością, na jaką napotkamy, jest rozpoznanie kontrolera wyświetlacza. Na rynku dostępne są dwa: Epson S1D15G00 i Philips PCF8833. Na płytkach ewaluacyjnych w laboratorium wyświetlacza mają ten pierwszy kontroler.

2. Programowanie wyświetlacza

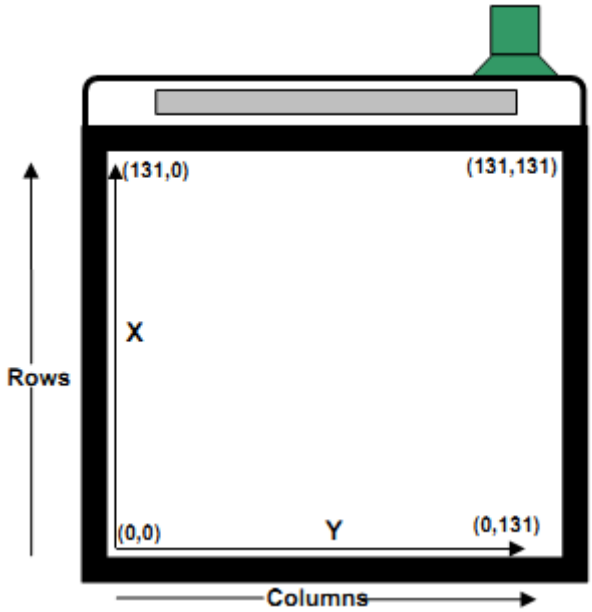
2.1. Orientacja wyświetlacza

Domyślną orientację wyświetlacza przedstawia rys. 1.



Rys. 1. Typowy układ wyświetlacza

Praktycznie, nie widać pierwszego i ostatniego rzędu i kolumny, co oznacza, że można tylko wykorzystać rozdzielczość 130x130 pikseli. Na płytce ewaluacyjne wyświetlacz jest usytuowany „do góry nogami”. Wymaga to, w związku z tym, zastosowania poleceń odbicia lustrzanego obu osi (mirror x i mirror y), by odwrócić wyświetlacz o 180 stopni (rys. 2).

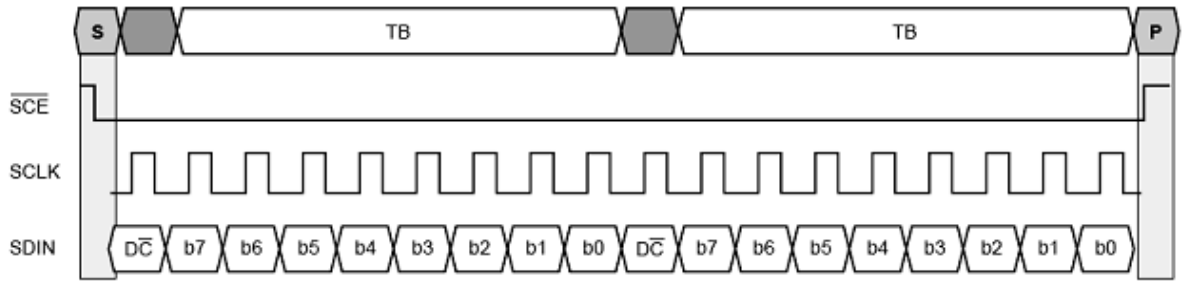


Rys. 2. Wyświetlacz po odwróceniu o 180 stopni

2.2 Komunikacja z wyświetlaczem

Wyświetlacz ten wykorzystuje 2-przewodowy interfejs SPI. Układ peryferyjny mikrokontrolera ARM 7 generuje sygnał zegarowy i przekazuje dane po linii MOSI (Master Out Slave In) a wyświetlacz pracuje wyłącznie jako SLAVE. Oznacza to, że linia MISO (Master In Slave Out) jest nieaktywna (nie ma tego połączenia). W związku z tym nie jest możliwe odczytanie danych z wyświetlacza takich jak: kody identyfikacyjne, status, temperatura itp.

Do wyświetlacza przesyłanych jest 9 bitów danych, przy czym 9. bit (najstarszy) decyduje o tym czy przesyłane są dane, czy polecenia (0 na tym bicie oznacza polecenie).



Rys.3. Przesyłanie danych i poleceń poprzez interfejs SPI

Poniżej znajduje się fragment kodu przesyłającego polecenie:

```
unsigned int command;   //  bajt polecenia

while ((pSPI->SPI_SR & AT91C_SPI_TXEMPTY) == 0); // oczekiwanie na zakończenie pop. przesyłu
command = (command & (~0x0100));   // zerowanie 9. bitu dla polecenia
```

```
pSPI->SPI_TDR = command;   // wysłanie polecenia
```

i dane:

```
unsigned int data;    //  bajt danych

while ((pSPI->SPI_SR & AT91C_SPI_TXEMPTY) == 0); // oczekiwanie na zakończenie pop. przesyłu
data = (data | 0x0100);    // ustawienie 9. bitu dla danych
pSPI->SPI_TDR = data;    // wysłanie danej
```

Oba fragmenty kodu zawierają bezpieczne testowanie flagi TXEMPTY, co gwarantuje oczekiwanie na zakończenie poprzedniej transmisji zanim rozpoczniemy nową.
Program sterowania wyświetlaczem zawiera 3 podstawowe funkcje:

InitSpi( )	- ustawia interfejs 1 SPI do komunikacji z LCD
WriteSpiCommand(command)	- wysyła polecenie do LCD
WriteSpiData(data)	- wysyła dane do LCD

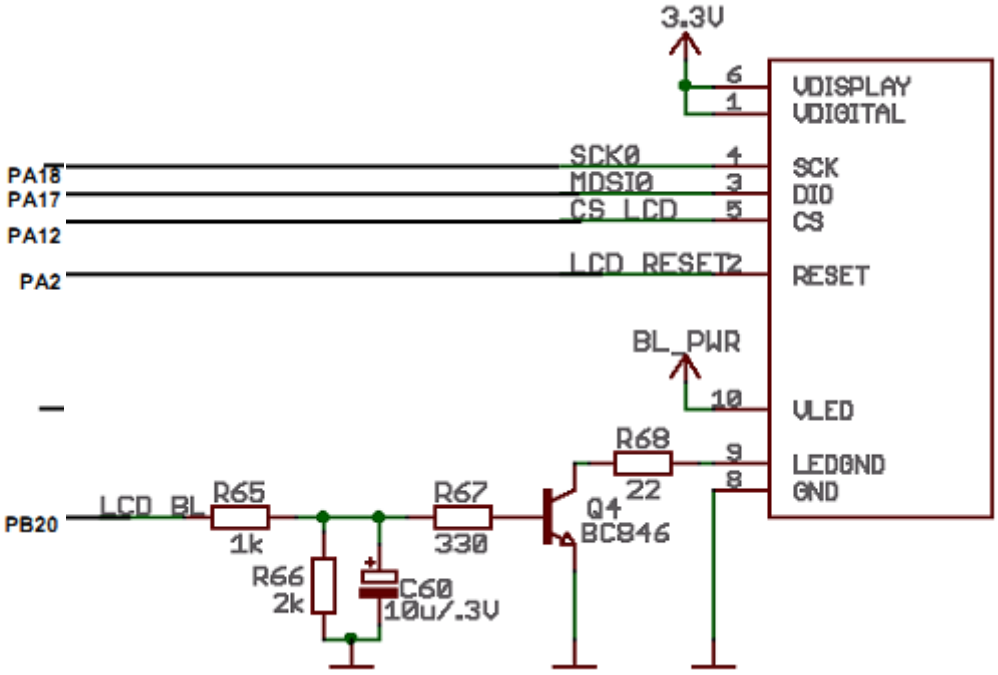
Wykorzystanie tych funkcji jest dość proste, np. ustawienie kontrastu odbywa się następująco:

```
InitSpi( );                          //  Inicjalizuje interfejs SPI dla LCD
WriteSpiCommand(SETCON);             //  wysyła polecenie dot. kontrastu  (command 0x25)
WriteSpiData(0x30);                  //  ustawienie kontrastu 0x30 (zakres jest od -63 do +63)
```

Połączenia sprzętowe wykorzystują następujące linie portów:

PA2	LCD Reset (set to low to reset)
PA12	LCD chip select (set to low to select the LCD chip)
PA16	SPI0_MISO Master In - Slave Out (not used in LCD interface)
PA17	SPI0_MOSI Master Out - Slave In pin (Serial Data to LCD slave)
PA18	SPI0_SPCK Serial Clock (to LCD slave)
PB20	backlight control (normally PWM control, 1 = full on)

Rys. 4. Linie portów wykorzystane do sterowanie LCD



Rys. 5. Schemat połączeń elektrycznych

2.3 Adresowanie pamięci

Sterownik wyświetlacza zawiera 17424 słowa pamięci (132x132), przy czym każde słowo jest 12-bitowe (po 4 bity na każdy podstawowy kolor). Adresowanie każdego piksela polega na podaniu polecenia adresu rzędu – PAS (Page Address Set) i kolumny – CAS (Column Address Set). Oba polecenia specyfikują adres piksela początkowego i końcowego. Daje to efekt rysowania prostokąta. W celu uzyskania tego efektu wykorzystuje się dwie możliwości rysowania prostokąta: zawijania i auto-inkrementacji, które znakomicie ułatwiają wykonanie zadania. Możliwości te daje się również wykorzystać do wypisywania znaków.
W celu zapalenia pojedynczego piksela należy określić ten sam adres początkowy i końcowy dla rzędu i kolumny. Na przykład, w celu zapalenia piksela o współrzędnych (2,7) wykonujemy następującą sekwencję:

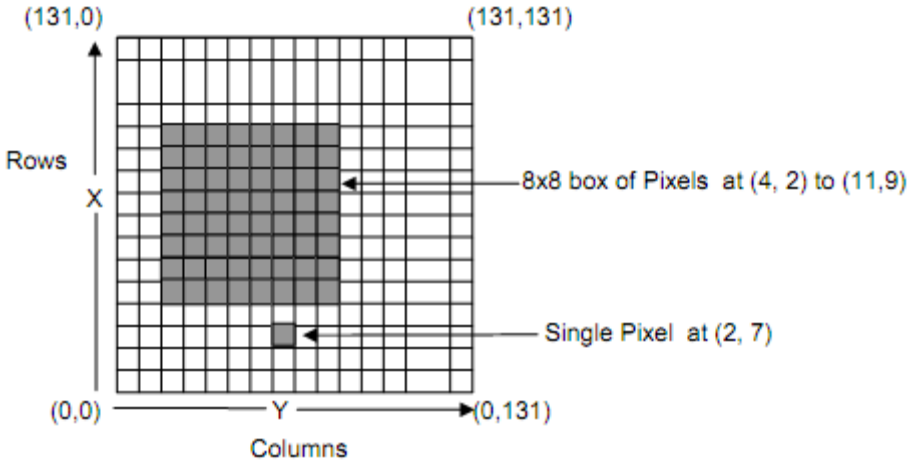
```
WriteSpiCommand(PASET);     // ustawienie adresu rzędu (polecenie 0x2B)
WriteSpiData(2);            // adres początkowy x
WriteSpiData(2);            // adres końcowy x (taki sam, jak początkowy)

WriteSpiCommand(CASET);     // // ustawienie adresu kolumny (polecenie 0x2A)
WriteSpiData(7);            // adres początkowy y
WriteSpiData(7);            // adres końcowy y (taki sam, jak początkowy)
```

Natomiast, w celu zaświecenia kwadratu 8x8 począwszy od piksela (4,2) a skończywszy na pikselu (11,9), wykonujemy następującą sekwencję (rys. 6):

```
WriteSpiCommand(PASET);     // ustawienie adresu rzędu (polecenie 0x2B)
WriteSpiData(4);            // adres początkowy x
WriteSpiData(11);           // adres końcowy x

WriteSpiCommand(CASET);     // // ustawienie adresu kolumny (polecenie 0x2A)
WriteSpiData(2);            // adres początkowy y
WriteSpiData(9);            // adres końcowy y
```

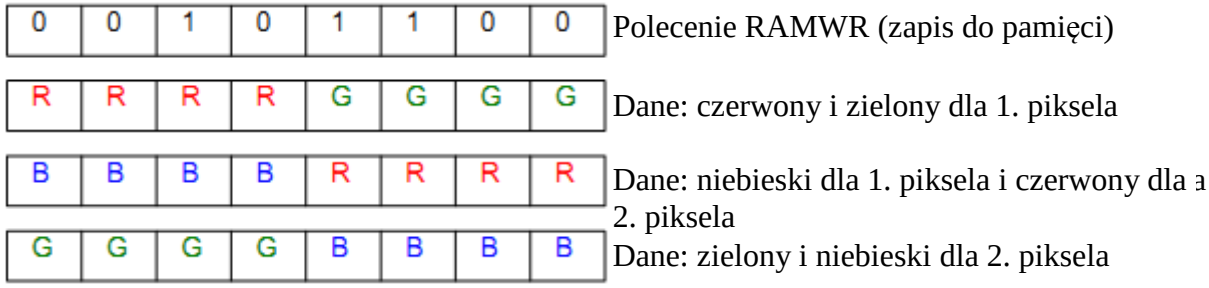


Rys. 6. Adresowanie pikseli

Po określeniu adresów pikseli jakikolwiek dostęp do pamięci sterownika wyświetlacza odnosi się wyłącznie do wcześniej zdefiniowanego zakresu. Oznacza to, że jakakolwiek próba zapisu do innego, niż zdefiniowany, obszaru pamięci nie zostanie wykonana.

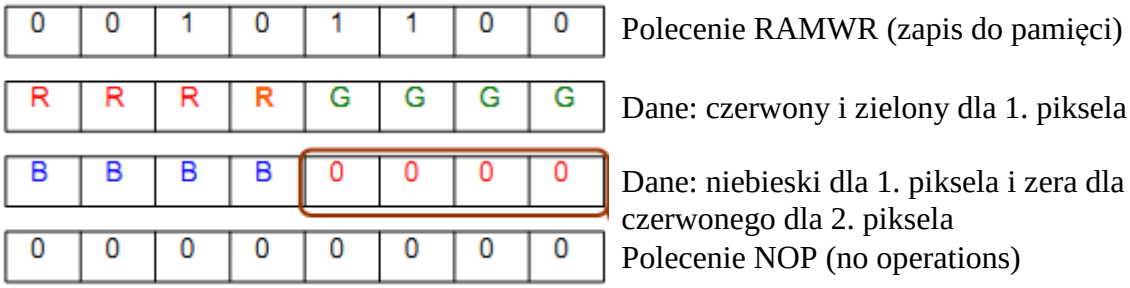
2.4. Dane dotyczące koloru (12-bitowy kolor)

Ten natywny tryb związany z rozdzielczością koloru uzyskuje się poprzez wysłanie polecenia: Color Interface Pixel Format (0x3A), po czym następuje wysłanie danej równej 3. Określenie koloru dla pojedynczego piksela uzyskuje się poprzez wysłanie 1,5 bajtu (12 bitów). Bajty mają postać spakowaną, więc dane o kolorach dla dwóch pikseli zajmują trzy kolejne bajty pamięci sterownika wyświetlacza. Rysunek 7 ilustruje rozkład bitów związanych z kolorem pikseli.



Rys. 7. Rozkład bitów dla kolorów

W związku z powyższym opisem może pojawić się pytanie: co się stanie, gdy chcemy określić kolor tylko jednego piksela? Co się stanie z 4. młodszymi bitami drugiego bajtu? Właściwość sterownika wyświetlacza jest taka, że zostanie wykonany zapis do pamięci wyświetlacza tylko wtedy, gdy zostaną dostarczone kompletne dane dla danego piksela. Oznacza to, że cztery bity czerwonego koloru dla drugiego piksela będą czekać na pozostałe. Jeżeli to nie nastąpi, gdyż pojawiło się następne polecenie, to zapoczątkowana operacja zapisu koloru drugiego piksela zostanie skasowana. Dla pewności można wykonać zapis do pamięci, zgodnie z rys. 8.



Rys. 8. Wysłanie danych o kolorze dla jednego piksela

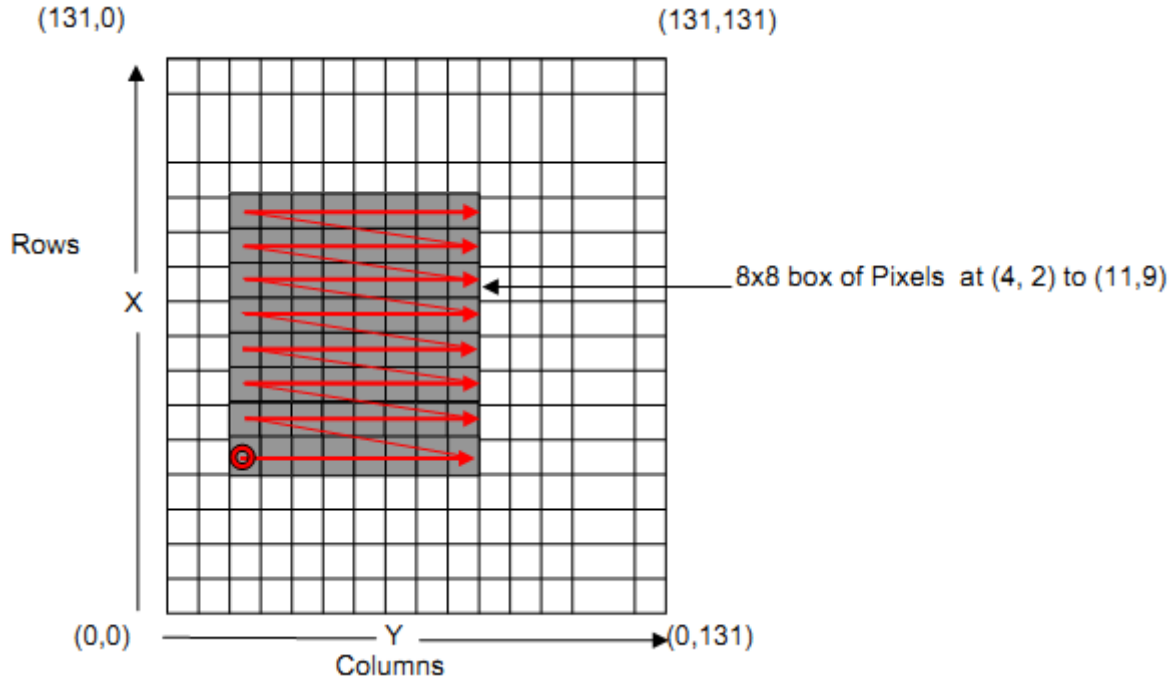
2.5. Zawijanie i auto-inkrementacja

Właściwość ta pozwala na efektywne rysowanie znaku alfanumerycznego lub wypełnianie prostokąta kolorem. Uzyskuje się to poprzez wykorzystanie właściwości zawijania po zapisaniu piksela w ostatniej kolumnie i auto-inkrementacji w następnym rzędzie. Na przykład, dla kwadratu 8x8 wypełnienie 64 pikseli polega na wykonaniu prostej pętli. Adresy rzędów i kolumn będą wówczas automatycznie inkrementowane i zawijane na końcu każdego rzędu (rys. 9).

Zasady związane z auto-inkrementacją i zawijaniem są następujące:

- ustaw adres kolumny i rzędu na lewy dolny róg rysowanego prostokąta

- ustaw pętle, by obsłużyć wszystkie piksele w prostokącie. Biorąc pod uwagę fakt, że dla 2 pikseli potrzeba 3 bajtów określających kolor, pętla będzie wykonywana 1,5 razy więcej niż liczba pikseli
- każdy zapisany piksel automatycznie zwiększa adres kolumny. Po osiągnięciu maksymalnego adresu kolumny następuje zawinięcie adresu, tzn. adres kolumny wraca na wartość początkową, a adres rzędu jest inkrementowany.



Rys. 9. Auto-inkrementacja i zawijanie dla rysowanego prostokąta

Przykład funkcji realizującej powyżej opisane zadanie jest następujący:

```
// Ustawienie adresu rzędu  (polecenie 0x2B)
 WriteSpiCommand(PASET);
 WriteSpiData(4);
 WriteSpiData(11);

// Ustawienie adresu kolumny (polecenie 0x2A)
 WriteSpiCommand(CASET);
 WriteSpiData(2);
 WriteSpiData(9);

// Zapis do pamięci  (polecenie  0x2C)
 WriteSpiCommand(RAMWR);

// pętla wykonywana jest – liczba pikseli/2
for (i = 0; i < ((((11 - 4 + 1) * (9 - 2 + 1)) / 2) + 1); i++) {
  // podaj wartości kolorów w trzech bajtach – odpowiadających 2. pikselom
  WriteSpiData((color >> 4) & 0xFF);
  WriteSpiData(((color & 0xF) << 4) | ((color >> 8) & 0xF));
  WriteSpiData(color & 0xFF);
 }
```

W ten sposób wykonujemy 33 zapisy do pamięci (po 3 bajty). Daje to 106 transmisji SPI. W przypadku nie korzystania z możliwości auto-inkrementacji i zawijania, trzeba wykonać 576 transmisji SPI.
W podobny sposób, jak dla prostokąta, realizujemy wyświetlanie dowolnego znaku ale wówczas dla każdego piksela należy określić kolor znaku i tła.

2.6 Inicjalizacja wyświetlacza

Niestety, sterownik wyświetlacza nie jest gotowy do normalnej pracy po resecie. Minimalny zestaw poleceń/danych do uruchomienia wyświetlacza w trybie 12-bitowego koloru jest następujący.
Po pierwsze wykonujemy reset sprzętowy:

```
// Sprzętowy reset
  LCD_RESET_LOW;
  Delay(20000);
  LCD_RESET_HIGH;
  Delay(20000);
```

Następnie wykonujemy kolejne ustawienia parametrów pracy wyświetlacza: domyślny timing (P1=0), 132 linie (P2=0x20), bez inwersyjnie podświetlanych linii (P3=0):

```
// Parametry pracy wyświetlacza
 WriteSpiCommand(DISCTL);
 WriteSpiData(0x00); // P1: 0x00
 WriteSpiData(0x20); // P2: 0x20
 WriteSpiData(0x00); // P3: 0x00
```

Po czym ustawiamy wspólny kierunek skanowania wyjścia (dobrany eksperymentalnie):

```
// Kierunek skanowania
 WriteSpiCommand(COMSCN);
 WriteSpiData(0x01);  // P1: 0x01
```

I włączamy oscylator i wyłączamy tryb uśpienia:

```
// Włączamy oscylator
 WriteSpiCommand(OSCON);

 // Wyłączamy tryb uśpienia
 WriteSpiCommand(SLPOUT)
```

Oraz włączamy regulatory napięcia:

```
// Włączanie regulatorów
 WriteSpiCommand(PWRCTR);
 WriteSpiData(0x0f);
```

Jak również włączamy inwersję kolorów:

```
// Włączenie inwersji kolorów
 WriteSpiCommand(DISINV);
```

A w odniesieniu do rejestru DATCTL ustawiamy tryb koloru 12-bitowy (P1=0x01), sekwencję RGB (P2=0x00) – domyślna i skalę szarości (P3=0x02):

```
// Kolejne parametry
 WriteSpiCommand(DATCTL);
 WriteSpiData(0x01); // P1: 0x01 = adres strony w inwersji, adres kolumny normalny, skanowanie adresu w kierunku kolumny
 WriteSpiData(0x00); // P2: 0x00 = sekwencja RGB (wartość domyślna)
 WriteSpiData(0x02); // P3: 0x02 = skala szarości -> 16 (wybór 12-bitowego koloru, typu A)
```

Wreszcie kontrast i wartość rezystancji:

```
// Ustawienie kontrastu
 WriteSpiCommand(VOLCTR);
 WriteSpiData(32); // P1 = 32  (kontrast ustawiony z zakresu  0 .. 63)
 WriteSpiData(3); // P2 = 3    wartość rezystancji  (dobrana eksperymentalnie)
```

W końcu opóźnienie i włączenie wyświetlacza:

```
// by pozwolić na ustabilizowanie się napięć
 Delay(100000);

 // włączenie wyświetlacza
 WriteSpiCommand(DISON);
```

3. Moduły programowe przykładowego projektu

W przykładzie, stanowiącym bazę do wykonania zadań w ćwiczeniu, główne moduły programu to:

- lcd.c – zawiera kod umożliwiający komunikację poprzez interfejs SPI i komplet funkcji do wyświetlania podstawowych elementów graficznych takich jak: zapalanie pojedynczego piksela, kreślenie linii, prostokąta, okręgu i wyświetlanie obrazu. Dołączone tabele fontów pozwalają na wyświetlanie znaków alfanumerycznych w trzech rozmiarach.
- lcd.h – zawiera polecenia sterownika wyświetlacza i specyfikację kodów kolorów.

4. Program ćwiczenia

4.1. Uruchomić środowisko Eclipse, utworzyć nowy projekt a następnie zaimportować przykładowy projekt (P2).
4.2. Przeanalizować pliki źródłowe projektu.
4.3. Skompilować przykładowy program i przesłać kod wynikowy do pamięci flash mikrokontrolera poprzez interfejs JTAG. Zaobserwować działanie programu.
4.4. Zaobserwować działanie programu dla innych parametrów funkcji graficznych (kolor, położenie wyświetlanego: piksela, linii, prostokąta, okręgu, znaku, łańcucha znaków). Zrobić próby dla znaków o różnej wielkości fontu.
4.5. Zaprojektować i wykonać proste menu 2-poziomowe na wyświetlaczu LCD.