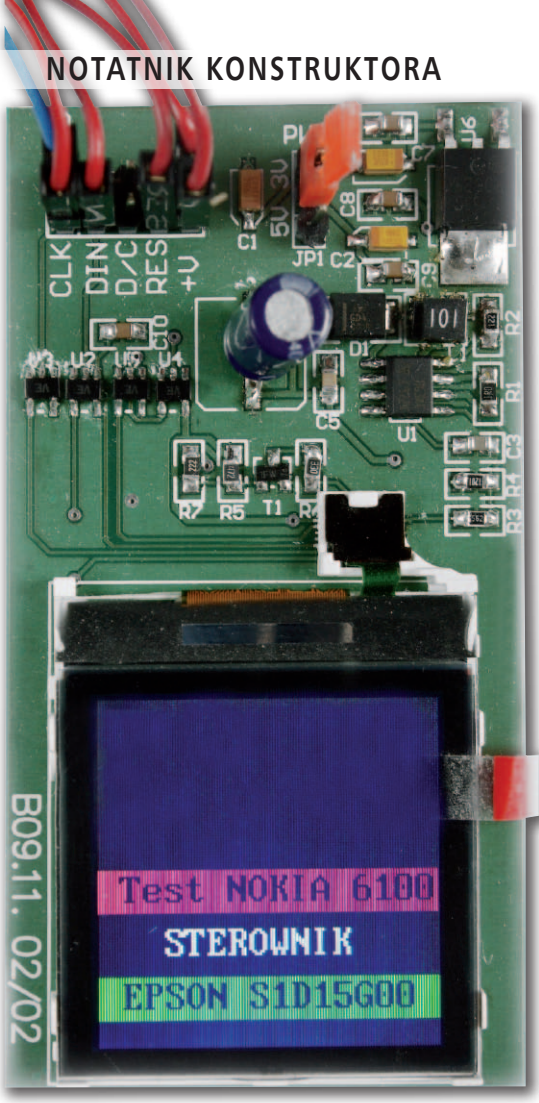
Wyświetlacz telefonu Nokia 6100

Obsługa sterownika Epson S1D15G00



Mimo, że ceny graficznych wyświetlaczy LCD ciągle spadają, to koszt wyświetlacza kolorowego jest nadal stosunkowo wysoki. Jeżeli zależy nam na tym, aby w aplikacji był użyty kolorowy wyświetlacz graficzny o niewielkich rozmiarach ekranu, to zawsze można rozważyć zastosowanie kolorowego wyświetlacza od telefonu komórkowego.

Wyświetlacze przeznaczone do telefonów komórkowych są tanie, a część z nich jest dość dobrze udokumentowana. Jednak trzeba pamiętać, że wyświetlacze tego samego modelu telefonu mogą mieć różne typy sterowników, a sterowniki montowane w serwisowych zamiennikach wyświetlaczy, nie zawsze są wierną kopią sterownika firmowego. Również elektryczne połączenie wyprowadzeń wyświetlacza z układem sterowania często stanowi nie lada wyzwanie. Różnorodne wyprowadzenia oferowanych wyświetlaczy: złącza, styki, taśmy itp. wymagają zaprojektowania specjalnych płytek drukowanych lub stosowania niestandardowych złącz, co jest dość kłopotliwe, szczególnie w układach prototypowych.

Pin	Funkcja
1	Vcc
2	!RESET
3	DATA
4	CLK
5	!CS
6	Vcc
7	NC
8	GND
9	VLED-
10	VLED+

Fotografia 1. Wyświetlacz telefonu Nokia 6100

Na rynku jest bardzo dużo modeli telefonów używających różnych rodzajów wyświetlaczy. Do stosowania we własnych urządzeniach warto wybrać taki, z którym będzie jak najmniej problemów. Powinien być tani, łatwo dostępny i ze znanym sterownikiem. Mój wybór padł na wyświetlacz od telefonu Nokia 6100 (ale również paru innych modeli telefonów tego producenta).

Wyświetlacz Nokii 6100

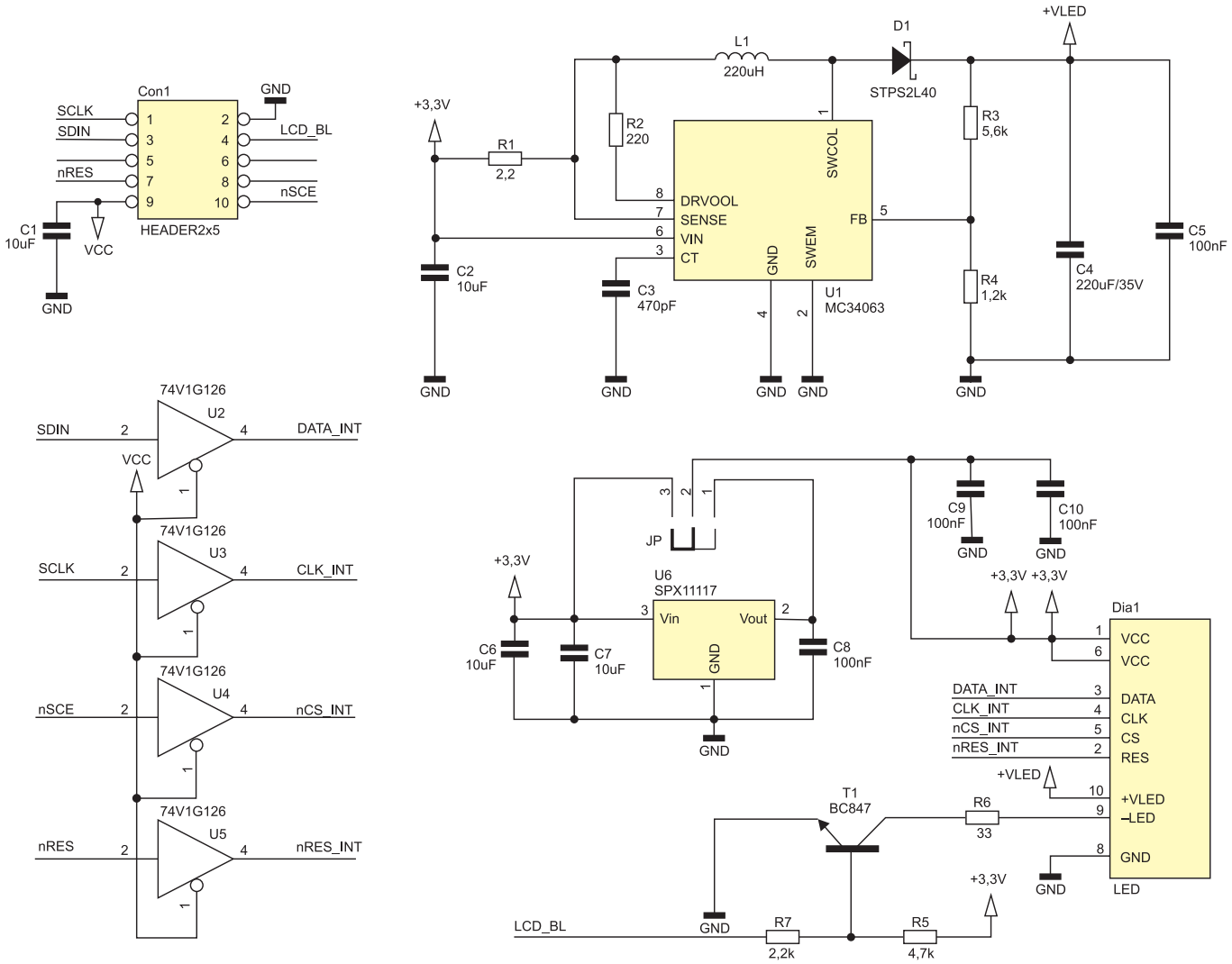
Jest to kolorowy wyświetlacz z matrycą o rozmiarze 132×132 piksele. Może być zasilany napięciem +3,3 V, a kolory są wyświetlane z 12-bitową głębią (4096 kolorów). Sterownik wyświetlacza komunikuje się z zewnętrznym sterownikiem (hostem) przez 3-przewodową magistralę SPI. Słowo przesyłane w czasie pojedynczej transakcji ma długość 9 bitów. Moduł ma wbudowane diody podświetlające. Wszystkie sygnały i linie zasilające są wyprowadzone elastyczną taśmą zakończoną specjalnym wtykiem, do którego można kupić gniazdo przeznaczone do montażu SMD.

Wyświetlacz ma jednak wadę z punktu widzenia przewidywanych zastosowań – są w nim montowane dwa różne typy sterowników: Epson S1D15G00 lub Philips PCF8833. Nie byłoby to jakimś wielkim problemem, gdyby sterowniki były ze sobą kompatybilne. Niestety tak nie jest. Mało tego, kupując wyświetlacz nie jesteśmy w stanie rozpoznać jaki ma sterownik. Jedynym wyjściem jest napisanie obsługi dla obu typów sterowników, bo oba są znane i dostępna jest do nich dokumentacja.

Żeby rozwiązać problem z podłączeniem wyświetlacza do zewnętrznego sterownika – hosta zaprojektowano płytkę drukowaną, na której umieszczono gniazdo do podłączenia wyświetlacza, bufory linii sterujących, akceptujące poziomy logiczne +5 V, stabilizator +3,3 V i układ przetwornicy podwyższający napięcie do zasilania układu podświetlania. Zewnętrzne sygnały sterujące i zasilania są podłączene do złącza IDC10. Schemat układów z płytki pokazano na **rysunku 2**.

Zworka J1 pozwala na włączenie lub wyłączenie (ominięcie) stabilizatora SPX-1117-3,3V. Stabilizator jest niezbędny, jeżeli system jest zasilany napięciem +5 V i nie ma dostępnego napięcia +3,3 V. Obwód z tranzystorem T1 pozwala na włączanie, wyłączanie podświetlenia lub sterowanie jego jasnością przebiegiem PWM.

Ponieważ matryca wyświetlacza ma rozmiar 132×132 piksele, to patrząc na wyświetlacz nie możemy na podstawie jego wyglądu określić orientacji wyświetlanej informacji. Domyślną orientację wyświetlacza pokazano na **rysunku 3**.



Rysunek 2. Schemat układów z płytki wyświetlacza

Czasami mechaniczne mocowanie wyświetlacza wymusza odwrócenie obrazu o 180°. Można to zrobić wysyłając do wyświetlacza odpowiednie komendy. Na **rysunku 4** pokazano orientację po odwróceniu obrazu o 180°.

Organizacja pamięci

Sterownik S1D15G00 może sterować kolorowym wyświetlaczem o organizacji 396 segmentów i 168 wierszy. Na każdy piksel przypadają 3 segmenty koloru czerwonego, zielonego i niebieskiego. Daje to możliwość sterowania wyświetlaczem o rozdzielczości 168×132 (168=396/3) pikseli. Znając możliwości sterowania można określić potrzebną wielkość pamięci RAM obrazu. Intensywność świecenia każdego z segmentów koloru (R, G, B) jest kodowana 4 bitami. Trzy składowe koloru po 4 bity dają 12-bitową głębię kolorów. Zatem pamięć RAM musi mieć pojemność 396×168×4=266112 bitów.

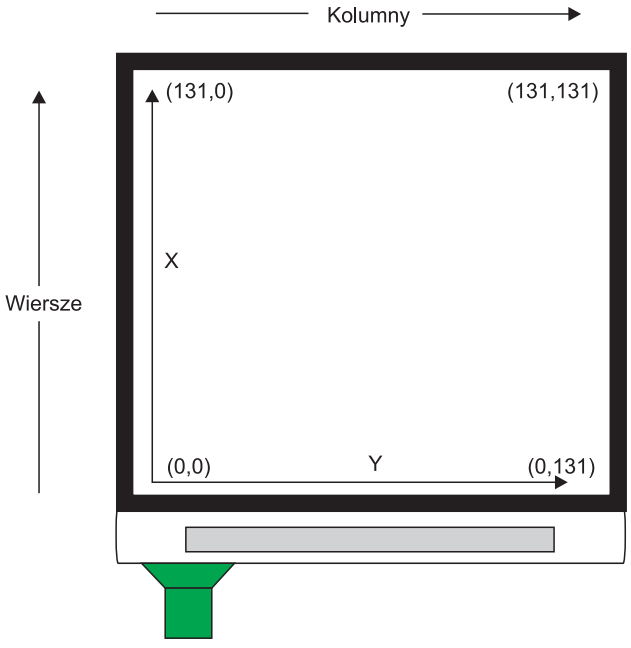
Na **rysunku 5** pokazano położenie zakodowanych 4 bitami składowych koloru dla 2 kolejnych pikseli. Informacja o kolorze pierwszego piksela zajmuje półtora bajta. Podobnie informacja o kolorze drugiego piksela zajmuje półtora bajta. Jest to niezbyt wygodna organizacja pamięci z punktu widzenia zorganizowanych bajtowo (8, 16, lub 32-bitowych) układów sterowania i wymaga dodatkowych zabiegów programowych.

Jeżeli nie potrzeba 12-bitowej głębi kolorów, można użyć trybu z 8-bitowa głębią. Informacja o kolorze jest zapisywana na 8 bitach: D7, D6, D5 dla składowej R, D4, D3, D2 dla składowej G oraz D1, D0 dla składowej B (**rysunek 6**). Po zapisaniu bajtu, sterownik przekształca 2- i 3-bitowe składowe na 4-bitowe i zapisuje w pamięci RAM, tak jakby to była głębia 12-bitowa.

Komendy SD15G00

Sterowanie wyświetlaczem odbywa się przez zapisywanie do niego komend z parametrami. W **tabeli 1** pokazano zbiorcze zestawienie wszystkich komend S1D15G00

Cześć komend sterownika jest oczywista i nie będziemy ich szczegółowo opisywać. Zostaną natomiast opisane komendy



Rysunek 3. Domyślna orientacja wyświetlacza

bardziej skomplikowane i wymagające szerszego wyjaśnienia.

Adresowanie pamięci obrazu jest powiązane ze sposobem wyświetlania informacji na ekranie. Sterownik może kontrolować wyświetlacz z matrycą mającą 168 linii i 132 kolumny. Pamięć obrazu sterownika można traktować jako obszar o organizacji 168×132 słów 12-bitowych. Pozycja danej w pamięci jest określana przez 2 liczniki: kolumn zmieniający się od 0 do 131 i wierszy (stron pamięci) zmieniający się od 0 do 167. Z modyfikacją liczników adresowych są związane komendy *PASET* i *CASET*. Obie mają po 2 argumenty określające początek zakresu zmiany i koniec zakresu zmiany liczników.

Na **rysunku 7** pokazano komendę *PASET*, a na **rysunku 8** komendę *CASET*.

Używając tych komend programuje się zakres adresów w obszarze których będą się zmieniały liczniki stron i kolumn po każdym zapisaniu danej do pamięci obrazu.

Żeby pokazać mechanizm działania komend modyfikacji adresów, zdefiniujemy obszar w kształcie kwadratu 8×8 pikseli. Do tego celu wykorzystamy komendy *PASET* z argumentem 4 (początek) i 11(koniec) i *CASET* z argumentem 2 (początek) i 9 (koniec) – **rysunek 9**.

Po wysłaniu tych komend wyświetlanie rozpocznie się od pozycji określonej przez argumenty adresu początku obu komend (4, 2). Każde wpisanie danej do sterownika będzie powodowało zwiększanie zawartości licznika kolumn od wartości początkowej określonej przez pierwszy argument komendy *CASET* do wartości końcowej kreślonej przez drugi argument tej komendy. Po osiągnięciu wartości końcowej, zwiększana jest zawartość licznika stron (wierszy), a licznik kolumn jest zerowany. W ten sposób kolejne zapisanie 64 danych tworzy obraz z 64 pikseli (kwadrat 8×8). Pokazano to na **rysunku 10**. Dalsze wpisywanie danych będzie zapełniało kwadrat od początku. Jest to bardzo użyteczny mechanizm, pozwalający w prosty sposób wyświetlać znaki alfanumeryczne o dowolnej wielkości oraz bitmapy o różnych wielkościach. Można również zdefiniować wyświetlanie pojedynczego piksela przez wpisanie do komendy takich samych adresów początku i końca.

Powiązanie pomiędzy danymi zapisywanymi do pamięci wyświetlacza, a informacją wyświetlaną na ekranie określają parametry komendy *DATCTL* (**rysunek 11**).

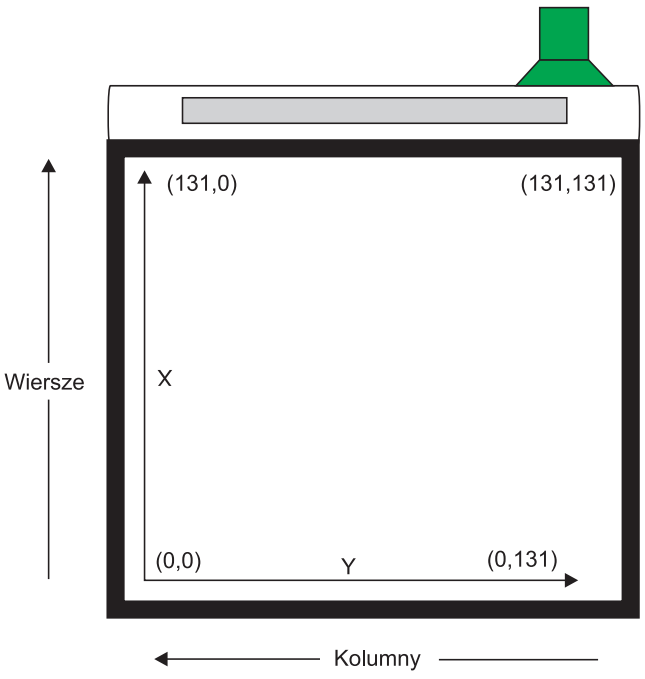
Pierwszy parametr określa sposób modyfikacji liczników adresowych w zdefiniowanym komendami PASET i CASET obszarze. Na rysunku 11 jest pokazana modyfikacja liczników w trybie normalnym, to znaczy po każdym wpisie do pamięci liczniki są inkrementowane.

Tabela 1. Zestawienie komend sterownika S1D15G00

Komenda	Opis	Kod (hex)	parametr
DISON	Włącz sterowanie wyświetlaczem	AF	brak
DISOFF	Wyłącz sterowanie wyświetlaczem	AE	brak
DISNOR	Sterowanie normalne	A6	brak
DISINV	Inwersja wyświetlania	A7	brak
COMSCN	Kierunek skanowania	BB	1bajt
DISCTL	Sterowanie wyświetlaniem	CA	3bajty
SLPIN	Włączenie uśpienia	95	Brak
SLPOUT	Wyłączenie uśpienia	94	Brak
PASET	Ustawienie adresu strony	75	2bajty
CASET	Ustawienia dresu kolumny	15	2bajty
DATCTL	Ustawienia sposobu zapisu danych	BC	3bajty
RGBSET8	Zapisanie tablicy konwersji kolorów	CE	20bajtów
RAMWR	Zapisanie pamięci RAM	5C	Dana
RAMRD	Odczytanie pamięci RAM	5D	Dana
PTLIN	Włączenie trybu partial	A8	2bajty
PLTOUT	Wyłączenie trybu partial	A9	brak
RMWIN	Start odczyt i zapis z modyfikacją	E0	brak
RMWOUT	Koniec trybu odczyt i zapis z modyfikacją	EE	brak
ASCSET	Ustawienie obszaru skrolowania	AA	4bajty
SCSTART	Start skrolowania	AB	1bajt
OSCON	Start wewnętrznego oscylatora	D1	Brak
OSCOFF	Zatrzymanie wewnętrznego oscylatora	D2	brak
PWCTRL	Sterowanie zasilaniem	20	1bajt
VOLCTRL	Sterowanie napięciem przetwornicy	81	2bajty
VOLUP	Zwiększenie sterowania przetwornicy o 1	D6	brak
VOLDOWN	Zmniejszenie sterowania przetwornicy o 1	D7	brak
TMPGRD	Współczynnik temperaturowy	82	1bajt
EPCTIN	Sterowanie EEPROM	CD	1bajt
EPCOUT	Koniec sterowania EEPROM	CC	brak
EPMWR	Zapis do EEPROM	FC	brak
EPMRD	Odczyt z EEPROM	FD	brak
EPSRRD1	Odczyt rejestru 1	7C	brak
EPSRRD2	Odczyt rejestru 2	7D	brak
NOP	NOP	25	brak
STREAD	Odczytanie rejestru statusowego	-------	-------

Kierunek zmiany licznika stron (wierszy) określa bit P10 pierwszego parametru komendy DATCTL. Dla P10=0 licznik stron jest inkrementowany, a dla P10 jest dekrementowany. Dokładnie tak samo z pomocą bitu P11 określa się kierunek modyfikacji licznika kolumn.

Bit P12 określa czy po zapisaniu danej zmienia się zawartość licznika kolumn, czy licznika stron. W przykładzie pokazanym na rysunku 11 po zapisaniu danej inkrementowany jest licznik kolumn. Tak dzieje się, kiedy P12=0. Kiedy P12=1, to po wpisaniu danej jest inkrementowany licznik wierszy, a po osiągnięciu wartości granicznej jest dopiero modyfikowany licznik kolumn. Wtedy na rysunku 10



Rysunek 4. Alternatywna orientacja wyświetlacza

R	R	R	R	G	G	G	G

Składowa czerwona i zielona pierwszego piksela

B	B	B	B	R	R	R	R

Składowa niebieska pierwszego i czerwona drugiego piksela

G	G	G	G	B	B	B	B

Składowa zielona i niebieska drugiego piksela

Rysunek 5. Położenie składowych koloru w pamięci RAM obrazu dla dwóch pikseli

R	R	R	R	G	G	B	B

Rysunek 6. Położenie składowych koloru w bajcie dla trybu 8-bitowego

Kod	75
parametr1	Adres strony początkowej
parametr2	Adres strony końcowej

Rysunek 7. Komenda PASET ustawienia zakresu numerów stron

Kod	15
Parametr1	Adres kolumny początkowej
Parametr2	Adres kolumny końcowej

Rysunek 8. Komenda CASET ustawienia zakresu numerów kolumn

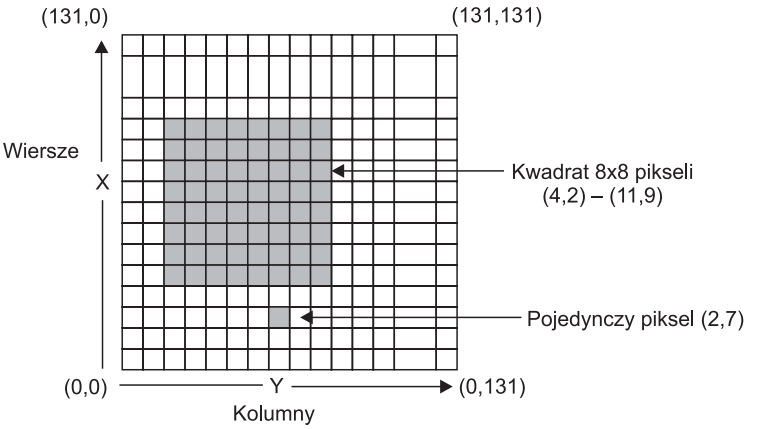
czerwone strzałki byłaby zwrócone pionowo poczynając od punktu początkowego.

W drugim parametrze komendy DACTL jest określone przypisanie składowych koloru do bitów słowa danych. W domyślnym trybie normalnym to przypisanie wygląda tak jak na rysunku 5.

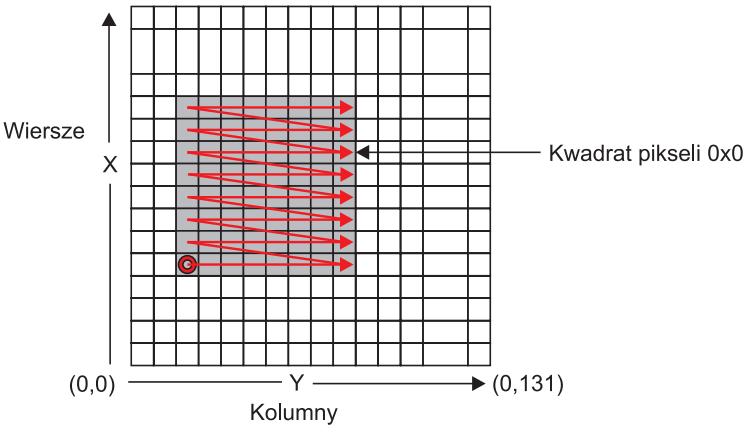
Parametr 3 definiuje głębię kolorów 8-bitową lub 12-bitową. Dla P32=0, P31=0 i P30=1 do sterownika wpisuje się dane 8-bitowe. Dla P32=0, P31=1 i P30=0 do sterownika wpisuje się dane 16-bitowe mając na uwadze pokazane na rysunku 6 położenie 12 bitów składowych koloru w 16-bitowym słowie.

Komenda *COMSCAN* umożliwia odwrócenie wyświetlanej informacji o 180 stopni bez konieczności zmiany mocowania wyświetlacza. Kierunek skanowania linii określa 1-bajtowy parametr komendy – **rysunek 12**.

Trójparametrowa komenda *DISCTL* (**rysunek 13**) jest używana do ustawiania funkcji związanych z zależnościami czasowymi sygnałów sterujących segmentami wyświetlacza. Pierwszy parametr określa zależności czasowe sygnałów sterujących CL, F1 i F2 dostępnych na wyprowadzeniach sterownika (**rysunek 14**). Drugi parametr określa zależności pomiędzy współczynnikiem multipleksowania a liczbą linii wyświetlacza. Trzeci parametr określa liczbę linii wyświetlanych inwersyjnie (od 2 do 16). Inwersyjne wyświetlanie jest wyłączone po wyzerowaniu tego parametru (wartość domyślna). Rozbudowana komenda *ACSET* (**rysunek 15**) pozwala na elastyczne ustawienie obszaru i rodzaju skrolowania. Dostępne są 4 tryby skrolowania ustawiane bitami P41 i P40 czwartego parametru komendy (**rysunek 16**). Dla współczynnika multipleksowania 1/132 zdefiniowany w sterowniku S1D15G00 blok ma szerokość 4 linii. Dla ekranu o wysokości 132 linii mamy do dyspozycji 32 bloki. Parametry komendy *ASCSET* definiują adres górnego bloku, adres dolnego bloku oraz liczbę bloków skrolowanego obszaru. Żeby w ogóle operację przesuwania można było wykonać, to obszar pamięci musi być większy niż pamięć obrazu. W tym celu definiuje się dodatkowy obszar nazywany *backgrund area* o wielkości 10 bloków, tak że cały obszar ma w sumie 42 bloki. Na przykład przyjmijmy, że będzie skrolowana centralna część ekranu. Obszar skrolowany ma 112 linii czyli 28 bloków. Od góry zarezerwujemy na nie-



Rysunek 9. Ustawianie zakresów adresów komendami



Rysunek 10. Zapisywanie wcześniej zdefiniowanego obszaru

Kod komendy DATCL BC									
Parametr 1	*	*	*	*	*	*	P12	P11	P10
Parametr 2	*	*	*	*	*	*	P22	P21	P20
Parametr 3	*	*	*	*	*	*	P32	P31	P30

Rysunek 11. Format komendy DATCTL

Kod Komendy COMSCAN BB								
Parametr	*	*	*	*	*	P12	P11	P10

P12	P11	P10	Kierunek skanowania	
			COM1 COM80	COM81 COM160
0	0	0	1 → 80	81 → 160
0	0	1	1 → 80	160 → 81
0	1	0	80 ← 1	81 → 160
0	1	1	80 ← 1	160 ← 81

Rysunek 12. Komenda COMSTAN

Kod komendy DISCTL CA									
Parametr 1	*	*	*	*	*	*	P12	P11	P10
Parametr 2	*	*	*	*	*	*	P22	P21	P20
Parametr 3	*	*	*	*	*	*	P32	P31	P30

Rysunek 13. Format komendy DISCTL

P13	P12	Współczynnik podziału sygnału CL
0	0	2
0	1	4
1	0	8
1	1	1

P13	P12	Okres przełączania F1, F2
0	0	8H
0	1	4H
1	0	16H
1	1	pole

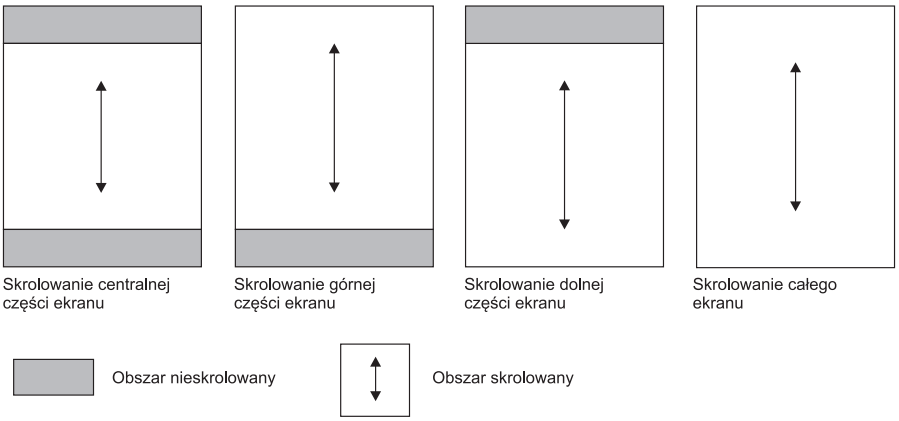
Rysunek 14. Znaczenie bitów pierwszego argumentu komendy DISCTL

skrolowany obszar 8 linii, czyli 2 bloki i od dołu 8 linii, czyli również 2 bloki. Komenda *ASCSET* musi mieć następujące parametry:

– P1 (adres górnego bloku) =2.
– P2 (Adres dolnego bloku) =39. Ta wartość wynika z dodania wielkości *background area* (10 bloków) i adresu 29. Adres 29, to 2 bloki górnego obszaru nieskrolowanego i 28 bloków obszaru skrolowanego z zastrzeżeniem, że adresy są liczone od 0.
– Liczba bloków równa 28

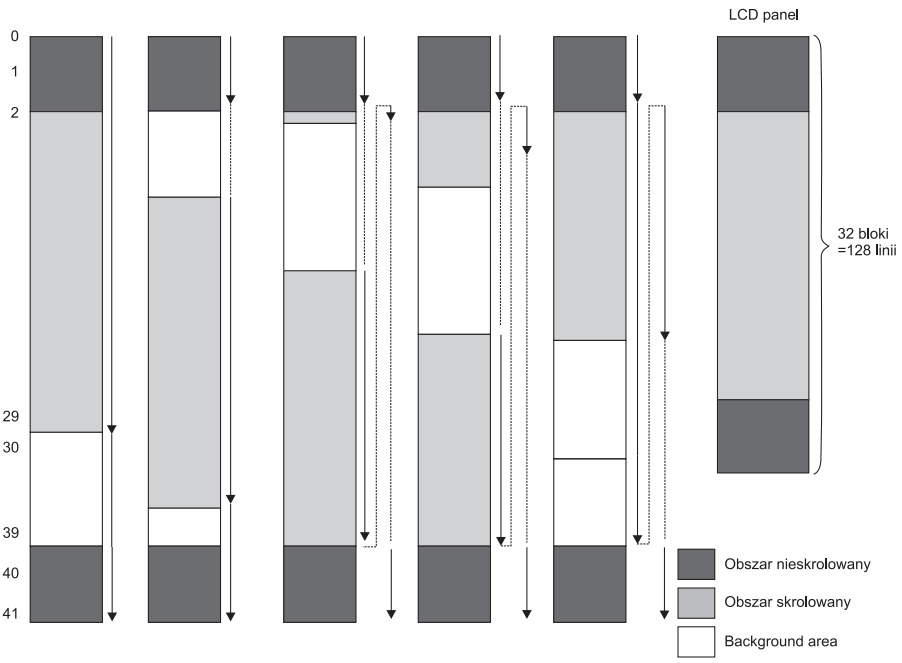
Kod komendy ASCSET AA										
Parametr 1	*	*	*	P15	P14	P13	P12	P11	P10	Adres górnego bloku
Parametr 2	*	*	*	P25	P24	P23	P22	P21	P20	Adres dolnego bloku
Parametr 3	*	*	*	P35	P34	P33	P32	P31	P30	Liczba bloków
Parametr 4	*	*	*	*	*	*	*	P41	P40	Tryb skrolowania

Rysunek 15. Komenda ustawienia skrolowania ASCSET



P41	P40	
0	0	Skrolowanie centralnej części ekranu
0	1	Skrolowanie górnej części ekranu
1	0	Skrolowanie dolnej części ekranu
1	1	Skrolowanie całego ekranu

Rysunek 16. Dostępne tryby skrolowania ekranu



Rysunek 17. Działanie mechanizmu skrolowania

Działanie tak zdefiniowanego przesuwania zostało pokazane na **rysunku 17**. Uruchomienie przesuwania następuje po wysłaniu komendy *SCSTART*. Jej parametrem jest adres bloku, od którego rozpoczyna się skrolowanie.

Komenda *RMWIN* działa w połączeniu z komendami *CASET* i *PASET*. Używa się jej do wprowadzenia w sterowniku mechanizmu: odczytaj/modyfikuj/zapisz. W pierwszym kroku jest odczytywana dana z lokalizacji określonej przez bieżącą zawartość liczników kolumn i stron. Odczyt nie powoduje modyfikacji liczników adresowych. Dopiero po zmodyfikowaniu danej i jej zapisaniu są modyfikowane liczniki adresowe (**rysunek 18**). Działanie trybu odczytaj/modyfikuj/zapisz kończy komenda RMWOUT.

Rozpoczęcie zapisu danych jest wykonywane po wysłaniu komendy *RMWIN*. Wykonanie kodu komendy zawsze powoduje ustawienie liczników adresowych na wartości początkowe, ustawione komendami *PASET* i *CASET*. Dane są kierowane do sterownika do momentu wysłania dowolnej komendy.

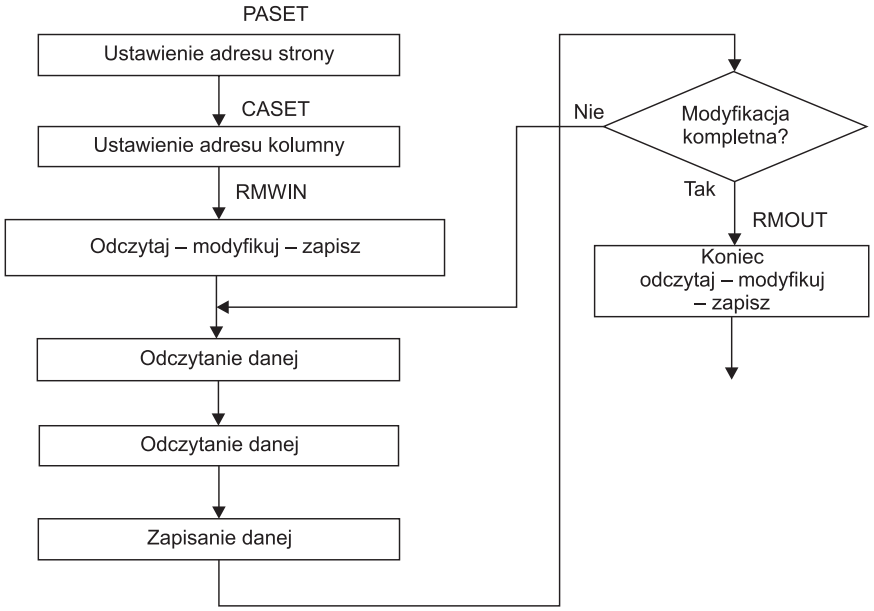
Wyświetlacze LCD ze sterownikiem S1D15G00 są głównie przeznaczone do pracy w urządzeniach przenośnych, zasilanych bateryjnie. Matryce LCD wymagają do prawidłowej pracy napięć wyższych, niż napięcie baterii zasilające sterownik (+3,3...3,6 V). Dlatego w układ sterownika wbudowano przetwornicę podwyższająca napięcie i programowany układ regulacji napięcia zasilania driverów matrycy. Każdy z układów przetwornicy może być programowo włączany lub wyłączany komendą *PWRCTR* (**rysunek 19**).

Układ regulacji napięcia zasilającego segmenty matrycy V2 pokazano na **rysunku 20**. Napięcie wyjściowe zależy od napięcia podawanego na nieodwracające wejście wzmacniacza, dzielnika rezystorowego Rb i Ra oraz programowanego współczynnika *alfa*. Obie te wartości są programowane komendą *VOLCTR* (**rysunek 21**).

Współczynnik *alfa* jest związany ze stopniem podziału elektronicznego potencjometru podającego napięcie na wyjście elektronicznego układu regulacji. Wyjście to jest połączone z nieodwracającym wejściem wzmacniacza z rysunku 21.

Parametr1 komendy *VOLCTR* można modyfikować przez wysyłanie komendy *VOULP* (inkrementacja) i komendy *VOLDN* (dekrementacja). Modyfikacja odbywa się modulo 256.

Korekcja napięcia zasilania driverów matrycy w funkcji temperatury otoczenia jest programowana komendą *TMPGRD* (**rysunek 22**).

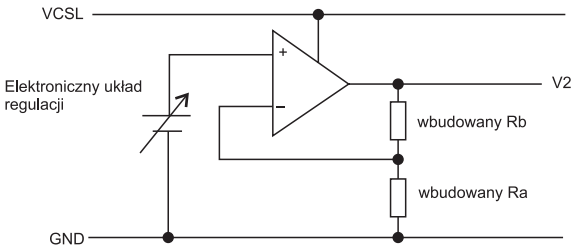


Rysunek 18. Działanie komendy RMWIN

Kod komendy PWCTR 20								
Parametr	*	*	*	*	P13	P12	P11	P10

P10 – sterowanie układem generowania napięcia referencyjnego
P10 = 1 załączony P10=0 wyłączony
P11- sterowanie układem regulatora napięcia wyjściowego
P11 = 1 załączony P11=0 wyłączony
P12 – sterowanie drugim stopniem boostera
P12 = 1 załączony P12=0 wyłączony
P13 – sterowanie pierwszym stopniem boostera
P13 = 1 załączony P13=0 wyłączony

Rysunek 19. Komenda PWCTR



Rysunek 20. Układ regulacji napięcia zasilającego matrycę

Programowa obsługa wyświetlacza

Sterownik S1D15G00 może się komunikować z hostem z pomocą 8- lub 16-bitowej równoległej magistrali pracującej w przemysłowym standardzie Intel8080 lub Motorola 6800. Takie połączenie zapewnia dużą prędkość przesyłanych danych, ale jest kłopotliwe w implementacji ze względu na dużą liczbę linii sterujących. Dlatego sterownik wyposażono też w 3- lub 4-przewodowy interfejs SPI. Do przesyłania danych używane są linie danych DATA, zegarowa CLK i wyboru CS. W interfejsie 4-przewodowym dodano jeszcze linię wyboru dane/komendy – D/C. Można wtedy przesyłać interfejsem standardowe dane 8-bitowe i wykorzystać w tym celu sprzętowe interfejsy SPI.

W interfejsie 3-przewodowym wyboru dane/komendy dokonuje się przesyłając dodatkowy, dziewiąty bit. Upraszcza to interfejs (jedna linia mniej), ale trochę komplikuje obsługę programową. W wyświetlaczu z telefonu Nokia 6100 w sterowniku S1D15G00 jest wybrany na stałe interfejs SPI w wersji 3-przewodowej (**rysunek 23**).

Nietypowy interfejs SPI, którym przesyła się 9 bitów danych, najwygodniej jest zaimplementować programowo. Na **listingu 1** pokazano funkcję *WriteSpi* wysyłającą przez hosta 8 bitów danych. Przesyłanie kompletnego 9-bitowego słowa można łatwo zrealizować przez poprzedzenie wywołania *WriteSpi* wysłaniem zera (zapis komendy) lub wysłaniem jedynki (zapis danej 8-bitowej). Procedury realizujące wysyłanie komend i danych zostały pokazane na **listingach 2** i **3**.

Po włączeniu zasilania sterownik wyświetlacza nie jest gotowy do pracy i wymaga zerowania oraz programowego zainicjalizowania. Pierwszą czynnością jaką należy wykonać jest sprzętowe zerowanie sterownika przez podanie na wejście zerowania (doprowadzenie *Reset*) poziomu niskiego, a następnie wymuszenie na tej linii poziomu wysokiego (**listing 4**). Programową inicjalizację sterownika pokazano na **listingu 5**. Rozpoczynamy ją od wysłania komendy *Display Control*. Parametr P1 komendy jest wyzerowany, co oznacza współczynnik podziału wynoszący 2 i okres przełączania równy 8 (wartość domyślna). Parametr P2 ma wartość 0x20 (132 dzies.). Ponieważ nie chcemy aby jakieś linie były wyświetlane inwersyjnie, to trzeci parametr zerujemy. Kolejna komenda *COMSCAN* określa orientacje wyświetlanej informacji. Po wpisaniu do parametru komendy wartości 0x01 będzie to skanowanie 1->80, 160<-81. Następne dwie komendy włączają wewnętrzny oscylator (*OSCON*) i wybudzają sterownik (drivery) z domyślnego stanu uśpienia (*SLPOUT*).

W trakcie prób z wyświetlaczem okazało się, że aby wyświetlane kolory były prawidłowe, trzeba wysłać komendę *DISINV*. W opisie sterownika nie znalazłem informacji o tym, w jakiej sytuacji należy używać tej komendy. Być może jej działanie jest związane z konstrukcją samej matrycy LCD.

Pierwszy parametr następnej komendy *DACTL* określa sposób modyfikacji liczników stron i kolumn. Dokładnie zostało to opisane przy okazji omawiania samej komendy. W inicjalizacji pierwszy parametr ma wartość 0x01. Adres stron

Kod komendy VOLCTR 81									
Parametr1	*	*	P15	P14	P13	P12	P11	P10	Parametr alfa
Parametr2	*	*	*	*	*	P22	P21	P20	1+Rb/Ra

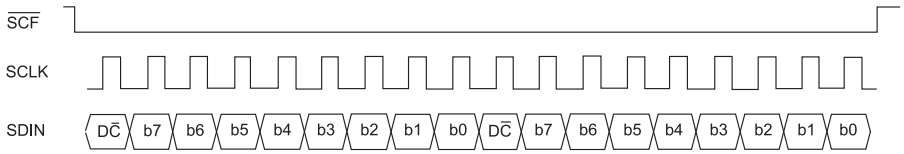
Parametr			1+Rb/Ra	Napięcie
P22	P21	P20		
0	0	0	3.95	Małe
0	0	1	4.27	
0	1	0	4.60	
0	1	1	4.93	
1	0	0	4.26	
1	0	1	5.59	
1	1	0	5.92	Duże
1	1	1	6.25	

Rysunek 21. Komenda VOLCTR

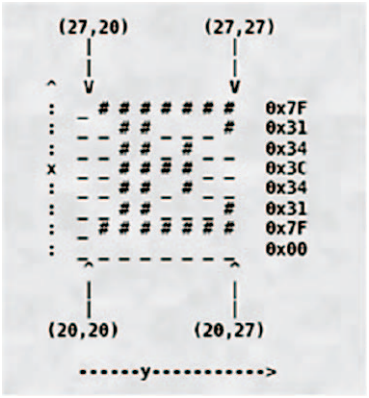
Kod komendy TMPGRD 82								
Parametr	*	*	*	*	*	*	P11	P10

P11	P10	Współczynnik korekcji %/C
0	0	-0,05
0	1	-0,1
1	0	-0,15
1	1	-0,2

Rysunek 22. Komenda TMPGRD



Rysunek 23. Przesłanie danych 3-przewodową magistralą SPI



Rysunek 24. Przykład wyświetlania znaku E

jest dekrementowany, a adres kolumn inkrementowany. Po każdym zapisaniu danych są modyfikowane liczniki kolumn. Drugi parametr jest wyzerowany dla uzyskania domyślnej wartości sekwencji kolorów RGB. Trzeci parametr programuje tryb wyświetlania 8- lub 12-bitowy. Po wpisaniu wartości 0x02 wybrany zostaje tryb 16-bitowej skali szarości równoważny 12-bitowemu kolorowi.

Kontrast wyświetlacza jest regulowany parametrami komendy *VOLCTR*. Pierwszy parametr to opisywany wcześniej parametr *alfa*. Można go ustawiać w zakresie 0...63.

Fotografia 25. Wielkości zdefiniowanych znaków

Drugi parametr określa współczynnik podziału dzielnika rezystancyjnego Ra/Rb (wzmocnienie wzmacniacza). Dobierając eksperymentalnie wartości tych parametrów można ustawić optymalny kontrast wyświetlacza. W trakcie prób u okazało się, że po podaniu jako drugi parametr komendy VOCTR wartości innej niż 2, regulacja kontrastu nie działa prawidłowo

Po włączeniu układu przetwornicy i ustawieniu napięcia wyjściowego trzeba odczekać czas potrzebny na ustabilizowanie się napięcia zasilającego drivery i można włączyć komendą *DISON* sterowanie matrycą wyświetlacza.

Ostatnią czynnością wykonywaną w trakcie inicjalizacji jest czyszczenie pamięci obrazu sterownika, bo po włączeniu zasilania w pamięci są zapisane wartości przypadkowe. Czyszczenie będzie polegało na zapisie wartości odpowiadającej jednakowemu kolorowi wszystkich pikseli wyświetlacza. Kolor inicjalizacji jest argumentem procedury *ClsLCD* umieszczonej na **listingu 6**.

Do czyszczenia (i nie tylko tutaj) wykorzystamy mechanizm pokazany na rysunkach 10 i 11. Najpierw komendami *PASET* i *CASET* definiuje się zakresy zmian adresów liczników stron i kolumn. Ponieważ zapisywanie będzie dotyczyło całego ekranu, to liczniki muszą się zmieniać w zakresie od 0 do 131 dla stron (wierszy) i kolumn. Zapisywanie danych do pamięci wyświetlacza należy poprzedzić komendą *RAMWR*. Jej wykonanie powoduje wpisanie do liczników adresowych wartości początkowych ustawionych komendami *PASET* i *CASET*. Każde zapisanie danej będzie powodowało modyfikację liczników w sposób określony komendą *DATCTL* z procedury inicjalizacji.

Jako argument komendy trzeba podać 12-bitową wartość odpowiadającą oczekiwanemu kolorowi. Na **listingu 7** pokazano definicje 12-bitowych liczb odpowiadających kolorom w systemie RGB. Jednemu pikselowi odpowiada 12 bitów w pamięci obrazu. Jednak dane są zapisywane do sterownika bajtowo. Kiedy popatrzymy na rysunek 5 to możemy dojść do wniosku, że najlepiej

Listing 1. Wysyłanie 8 bitów danych przez SPI
```
void WriteSpi(unsigned char data) {
unsigned char i;
         SetBit(CLK); //linia zegarowa stan wysoki
          for (i=0;i<8;i++) {
         ClrBit(CLK);              //linia zegarowa stan niski
         if ((data&0x80)==0)
          ClrBit(DATA);   //na linii danych „0”
         else
          SetBit(DATA);   //na linii danych „1”
         SetBit(CLK);              //linia zegarowa stan wysoki
        data<<=1;         //kolejny bit danych
    }
         SetBit(CS);               //linia SCF w stan nieaktywny ( wysoki)
}
```

Listing 2. Wysłanie komendy
```
void WriteSpiCommand(unsigned char cmd){
         ClrBit(CS);       //SCF w stan aktywny niski
         ClrBit(CLK);
         ClrBit(DATA);//pierwszy bit =0 - komendy
         WriteSpi(cmd);
}
```

Listing 3. Wysłanie danej
```
void WriteSpiData(unsigned char data){
         ClrBit(CS);       //SCF w stan aktywny niski
         ClrBit(CLK);
         SetBit(DATA);//pierwszy bit =1 - dane
         WriteSpi(data);
}
```

Listing 4. Inicjalizacja interfejsu SPI i zerowanie sterownika
```
SetBit(CS);//linie interfejsu SPI inicjowane w stan wysoki
SetBit(CLK);
SetBit(DATA);
ClrBit(RES);//reset wyświetlacza aktywny
for(i=0;i<8000;i++)
delay();
SetBit(RES);//reset nieaktywny
for(i=0;i<8000;i++)
delay();
```

Listing 5. przykładowa inicjalizacja sterownika S1D15G00

```
// Display control
   WriteSpiCommand(DISCTL);
   WriteSpiData(0x00); // P1: 0x00 = 2 współczynnik podziału i  period=8
   WriteSpiData(0x20); // P2: 0x20  = 132/4 - 1 = 32 linii
   WriteSpiData(0x00); // P3: 0x00 = bez inwersji
// COM scan
   WriteSpiCommand(COMSCN);
   WriteSpiData(1); // P1: 0x01 = skanowanie 1->80, 160<-81
// Właczenie wewnętrznego oscylatora
   WriteSpiCommand(OSCON);
// Wyjście ze stanu uśpienia
   WriteSpiCommand(SLPOUT);
// programowanie układu zasilania  ( przetwornicy)
   WriteSpiCommand(PWRCTR);
   WriteSpiData(0x0f); // wł. Regulatora nap. ref., wł układu regulatora nap.
wyjściowego, wł układu boostera
// Wyświetlanie inwersyjne
   WriteSpiCommand(DISINV);
// Data control
   WriteSpiCommand(DATCTL);
   WriteSpiData(0x01); // P1: 0x01 = dekrementowanie licznika stron,
inkrementowanie  licznika kolumn, zmiana licznika kolumn
   WriteSpiData(0x00); // P2: 0x00 = sekwencja RGB  (wartość domyślna)
   WriteSpiData(0x02); // P3: 0x02 = 16 bitowa skala szarości (12 bitowy kolor
A)
// sterowanie kontrastem
   WriteSpiCommand(VOLCTR);
   WriteSpiData(32); // P1 = 32 współczynnik alfa
   WriteSpiData(3); // P2 = 3 współczynnik podziału Ra/Rb
   for(i=0;i<8000;i++) //opóźnienie na ustabilizowanie się napięcia
wyjściowego
   delay();;
// włączenie sterowania wyświetlaczem
   WriteSpiCommand(DISON);
```

Listing 6. Czyszczenie wyświetlacza

```
Void ClsLCD(short color) {
int i;
// ustawienie zakresu zmian licznika adresowego stron pamięci
    WriteSpiCommand(PASET);
    WriteSpiData(0); //adres początkowy
    WriteSpiData(131);//adres końcowy
// ustawienie zakresu zmian licznika adresowego kolumn
    WriteSpiCommand(CASET);
    WriteSpiData(0);//adres początkowy
    WriteSpiData(131);//adres końcowy
// zapisanie pamięci kolorem z
    WriteSpiCommand(RAMWR);
    for(i = 0; i < ((131 * 131) / 2); i++) {
        WriteSpiData((color >> 4) & 0xFF); //kopiowanie 12 bitów dla
                          //1 piksela na 24 bity dla 2 pikseli
        WriteSpiData(((color & 0xF) << 4) | ((color >> 8) & 0xF));
        WriteSpiData(color & 0xFF);
    }
}
```

Listing 7. Definicja 12- bitowych kolorów

```
// definicja kolorów 12-bitowych
#define WHITE 0xFFF
#define BLACK 0x000
#define RED 0xF00
#define GREEN 0x0F0
#define BLUE 0x00F
#define CYAN 0x0FF
#define MAGENTA 0xF0F
#define YELLOW 0xFF0
#define BROWN 0xB22
#define ORANGE 0xFA0
#define PINK 0xF6A
```

jest przesyłać dane w 3-bajtowych porcjach (3×8=24 bity) odpowiadających kolejnym dwóm pikselom po 12 bitów każdy. Dla zapisania całej pamięci wyświetlacza trzeba zapisać 131×131 12-bitowych słów. Ale mając ciągle na uwadze rysunek 5 lepiej jest zapisać 131×131/2 24-bitowych słów. Tak też jest to robione w procedurze *ClsLCD* z listingu 7. W pętli, która wykonywana się (131×131/2) razy jest realizowane kopiowanie 12-bitów argumentu koloru dla jednego piksela na 24 bity dla dwóch pikseli.

Na prawidłowo zainicjowanym wyświetlaczu możemy wyświetlać informacje tekstowe lub bitmapy. Wyświetlanie tekstu wymaga zdefiniowania w pamięci mikrokontrolera hosta tablicy z generatorem znaków. Graficzna natura wyświetlacza powoduje, że można definiować znaki o różnych wielkościach, elastycznie dostosowując je do potrzeb. Dodatkowo, w kolorowym wyświetlaczu można definiować kolor znaku i kolor tła podnosząc czytelność i atrakcyjność wyświetlanej informacji.

Pierwszą bardzo żmudną czynnością, którą należy wykonać, jest zdefiniowanie tablicy generatora znaków. Pierwsza moja próba z wyświetlaniem znaków alfanumerycznych polegała na wykorzystaniu tablicy generatora znaków o wielkości 8×6 pikseli z monochromatycznego wyświetlacza od telefonu Nokia 3310. Jednak okazało się, że znaki o tej wielkości nie są zbyt czytelne i w większości przypadków przydałyby się większe. Przy definiowaniu większych znaków postanowiłem posiłkować się zawartością Internetu. Znalazłem tam gotową tablicę znaków o trzech wymiarach: 6×8 pikseli, 8×8 pikseli i 8×16 pikseli autorstwa Jamesa P. Lyncha. Do tablicy została dołączona procedura *LCDPutChar* (również tego autora) służąca do wyświetlania pojedynczych znaków. Po testach okazało się, że procedura działa doskonale i jest elastyczna, dlatego zdecydowałem się ją tutaj przedstawić (**listing 8**).

Argumentami funkcji *LCDPutChar* są:
- znak do wyświetlania *c*,
- współrzędne *x, y* początku umieszczenia znaku,
- parametr *size* określający wielkość znaku,
- definicja koloru znaku *fColor*,
- definicja koloru tła *bColor*.

Aby wyświetlanie mogło dobrze działać dla znaków o różnych wielkościach, na początku każdej tablicy generatora znaków zostały umieszczone informacje o liczbie kolumn i wierszy w znaku oraz liczbie bajtów do pobrania z tablicy niezbędnych do jego wyświetlenia. Te informacje są pobierane z tablicy i zapisywane do zmiennych *nColumns*, *nRows* i *nBytes*. Na podstawie *nColumns* i *nRows* oraz argumentów współrzędnych *x* i *y* wyliczane są adresy początku wyświetlania napisu i zapisywane komendami *PASET* i *CASET*. Do samego wyświetlania wykorzystano taki sam mechanizm automatycznej zmiany adresów w obszarze zdefiniowanym przez *PASET* i *CASET*.

Podobnie jak w procedurze czyszczenia ekranu do pamięci wyświetlacza są zapisywane za każdym razem 3 bajty określające kolor dwóch pikseli.

Listing 8. Wyświetlanie znaku na ekranie LCD (autor James P. Lynch)

```
void LCDPutChar(char c, int x, int y, int size, int fColor, int bColor) {
//deklaracje tablic z generatorami znaków
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 int nCols;
unsigned int nRows;
unsigned int nBytes;
unsigned char PixelRow;
unsigned char Mask;
unsigned int Word0;
unsigned int Word1;
unsigned char *pFont;
unsigned char *pChar;
unsigned char *FontTable[] = {(unsigned char *)FONT6x8, (unsigned char *)
FONT8x8,
(unsigned char *)FONT8x16};
//pobranie wskaźnika wybranej argumentem size jednej z trzech tablicy  znaków
        pFont = (unsigned char *)FontTable[size];
// pobranie  nColumns, nRows and nBytes - ilość kolumn, wierszy i bajtów na
znak
        nCols = *pFont;
        nRows = *(pFont + 1);
        nBytes = *(pFont + 2);
```

Trzeba pamiętać, że argument *c* zawiera kod ASCII znaku, a tablica z wzorcami znaków zaczyna się od znaku spacji. Z tego powodu, od kodu zawartego w zmiennej *c* trzeba odjąć 0x20. Jednak pierwsze 8 lub 16 bajtów tablicy są zajęte na informacje o wielkości znaku i liczbie bajtów na znak. Dlatego od kodu ASCII odejmuje się wartość 0x1F.

Na **rysunku 24** pokazano przykład wyświetlania znaku „E" o wielkości 8×8 pikseli, od współrzędnej (20, 20). Najpierw definiujemy obszar 8×8 pikseli od współrzędnych początkowych komendami PASET i CASET:

```
WriteSPICommand (PASET);
WriteData(20);
WriteData(27); //limit 20,27
WriteSPICommand (CASET);
WriteData(20); //limit 20,27
WriteData(27);
```

W czasie wpisywania danych licznik kolumn jest inkrementowany i kiedy osiągnie wartość 27 zostanie wyzerowany, a zwiększy się zawartość licznika wierszy. W ten sposób po wpisaniu 64 słów 12-bitowych wyświetlony zostanie znak „E". Trzeba wpisać 64 słowa, bo każdy bajt z generatora odpowiada 8 pikselom, a bajtów jest 8.

Po analizie list. 8 można zauważyć, że napisanie uniwersalnej procedury wyświetlania znaków alfanumerycznych nie jest banalne. Jeżeli dołączymy do tego konieczność zdefiniowania wzorów znaków w tablicy generatora znaków, to okaże się, że trzeba wykonać niemałą pracę tylko dla samego wyświetlania napisów. Jednak warto to zrobić dobrze, bo z mojego doświadczenia wynika, że wyświetlanie tekstów to najczęściej wykorzystywana funkcja w obsłudze wyświetlaczy graficznych.

Funkcję wyświetlania pojedynczego znaku użyjemy do napisania funkcji wyświetlającej napisy (**listing 9**). Jej argumentami są: wskaźnik do początku tablicy z napisem, współrzędne *x*, *y*, wielkość znaku i kody koloru znaku oraz tła.

Wyświetlanie pełnowymiarowych bitmap jest bardzo proste pod warunkiem, że mamy przygotowaną przez odpowiedni program tablicę z zapisanymi 8-bitowymi danymi RGB dla każdego z pikseli lub tablicę, w której na trzech bajtach są zapisane dwa piksele. W zasadzie zapisywanie bitmapy niewiele różni się od czyszczenia ekranu. Zamiast wpisywania stałej wartości koloru czyszczenia zapisuje się do pamięci dane z tablicy bitmapy. Na **listingu 10** pokazano wyświetlanie bitmapy w trybie koloru 8-bitowego.

Listing 8. cd.

```
//pobranie wskaźnika do ostatniego znaku do wyświetlania i
//korekcja z kodu ASCII do kodów tablicy generatora
        pChar = pFont + (nBytes * (c - 0x1F)) + nBytes - 1;
// zapisanie zakresu adresowania stron ( wierszy) - komenda PASET
        WriteSpiCommand(PASET);
        WriteSpiData(x);
        WriteSpiData(x + nRows - 1);
// zapisanie zakresu adresowania kolumn- komenda CASET
        WriteSpiCommand(CASET);
        WriteSpiData(y);
        WriteSpiData(y + nCols - 1);
// komenda RAMWR
        WriteSpiCommand(RAMWR);
//pętla dla każdego wiersza od góry do dołu
        for (i = nRows - 1; i >= 0; i--) {
//kopiowanie wiersza pikseli z tablicy generatora i dekrementacja wiersza
        PixelRow = *pChar--;
//pętla dla każdego piksela w wierszu ( z lewej do prawej) - każda pętla to 2
piksele
        Mask = 0x80;
        for (j = 0; j < nCols; j += 2) {
//jeżeli bit piksela jest ustawiony, to używamy koloru znaku
//w przeciwnym przypadku koloru tła
        if ((PixelRow & Mask) == 0)
        Word0 = bColor;
        else
        Word0 = fColor;
        Mask = Mask >> 1;
        if ((PixelRow & Mask) == 0)
        Word1 = bColor;
        else
        Word1 = fColor;
        Mask = Mask >> 1;
//zapisanie 3 bajtów dla 2 pikseli
        WriteSpiData((Word0 >> 4) & 0xFF);
        WriteSpiData(((Word0 & 0xF) << 4) | ((Word1 >> 8) & 0xF));
        WriteSpiData(Word1 & 0xFF);
        }
}
```

Listing 9. Funkcja wyświetlająca napisy

```
void LCDPutStr(char *pString, int x, int y, int Size, int fColor, int bColor)
{
// w pętli do napotkania znacznika końca łańcucha znaków 0x00
        while (*pString != 0x00) {
// wyświetlanie znaku
        LCDPutChar(*pString++, x, y, Size, fColor, bColor);
// wyliczenie pozycji następnego znaku w zależności od jego wielkości
        if (Size == SMALL)
        y = y + 6;        //dla 6x8
        else if (Size == MEDIUM)
        y = y + 8;        //dla 8x8
        else
        y = y + 8;        //dla 8x16
        // współrzędna poza zakresem
        if (y > 131) break;
        }
}
```

Listing 10. Zapisanie pełnowymiarowej bitmapy w trybie koloru 8-bitowego.

```
void LCDWriteBmp(void) {
        long j;
        WriteSpiCommand(DATCTL);
        WriteSpiData(7); // P1: 0x00 = adres strony dekr., adres
                                  //kolumn dekr.  ,mod. licznika stron
        WriteSpiData(0x00); // P2: 0x00 = RGB (wartosc domyslna)
        WriteSpiData(0x01); // P3:tryb 8 bitowy
        WriteSpiCommand(CASET); //zakres licznika kolumn
        WriteSpiData(0);
        WriteSpiData(131);
        WriteSpiCommand(PASET);//zakres licznika stron
        WriteSpiData(0);
        WriteSpiData(131);
        WriteSpiCommand(RAMWR);   //zapis do pamieci 17424 (132*132)bajtow
        for(j = 0; j < 17424; j++) {
                WriteSpiData(bmp[j]);
        }
// powrót do ustawień dla wyświetlania tekstu
        WriteSpiCommand(DATCTL);
        WriteSpiData(0x01);       //P1: 0x01 = adres strony dekr.,
                                        //adres kolumn inkr ,mod. licznika
stron
        WriteSpiData(0x00); // P2: 0x00 = RGB
        WriteSpiData(0x02); // P3: 0x02 = tryb 12 bitowy
        // Display On
        WriteSpiCommand(DISON);
}
```

Podsumowanie

Przedstawione w artykule informacje pozwalają na użycie wyświetlacza od telefonu Nokia 6100 ze sterownikiem Epson S1D15G00. Programowanie sterownika NXP PCF8833 jest podobne, a nawetnie prostsze. W przyszłości postaram się uzupełnić opis sterowania wyświetlaczem z tym sterownikiem

Tomasz Jabłoński, EP
tomasz.jablonski@ep.com.pl