Tomasz Francuz

Sterowanie wyświetlaczami LCD

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Maciej Mazak

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/miklcd
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
ftp://ftp.helion.pl/przyklady/miklcd.zip

ISBN: 978-83-283-2846-4

Printed in Poland.

Spis treści

Rozdział 13.

Akceleratory graficzne FT8xx

Poznaliśmy do tej pory różnego typu układy kontrolerów graficznych, których wspólną cechą jest możliwość sterowania wyświetlonym obrazem dzięki przechowywaniu jego bezpośredniej kopii w pamięci GRAM kontrolera. Takie rozwiązanie jest proste, lecz ma kilka wad:

- Wymaga dużej ilości pamięci GRAM, aby możliwe było przechowywanie kopii obrazu. Niestety pamięć zajmuje sporo miejsca w strukturze scalonej, co zwiększa koszty wykonania takiego układu.
- Kolejny problem związany jest z koniecznością renderowania obrazu przez CPU, który zamienia obiekty graficzne na ich reprezentację w pamięci GRAM. Proces renderowania, jak się przekonaliśmy, wymaga przeprowadzania intensywnych obliczeń i jest czasochłonny.
- Renderowanie obrazu wymaga przesyłania olbrzymich ilości danych między CPU a kontrolerem graficznym. W efekcie, aby zapewnić odpowiednią wydajność przesyłu, musimy stosować równoległe magistrale danych, nawet o szerokości 16 i więcej bitów.

Czy można w jakiś sposób powyższe problemy rozwiązać? Częściowe rozwiązanie poznaliśmy w poprzednich rozdziałach, a było nim przerzucenie części pracy związanej z renderowaniem obrazu na kontroler graficzny, na przykład *RA8875*. Spowodowało to dramatyczny wzrost szybkości przetwarzania danych, jednak możliwości wymienionego kontrolera w tym zakresie były mocno ograniczone, a ze względu na buforowanie obrazu w pamięci GRAM ciągle dosyć często zachodziła konieczność przesyłania surowych danych między CPU a kontrolerem. Zupełnie inne podejście do problemu tworzenia obrazu pokazała firma FTDI. Inżynierowie wyszli z ciekawego założenia: skoro wyświetlany obraz składa się z prostych elementów, tak zwanych prymitywów graficznych, to CPU zamiast zajmować się ich renderowaniem, a następnie przesyłaniem efektów tej obróbki do GRAM, może przesyłać po prostu dane o obiektach do kontrolera graficznego, który zajmie się ich wyświetlaniem. Co więcej, kontroler zamiast przekształcać je do formatu rastrowego, przechowywanego następnie w GRAM, może

bezpośrednio wyświetlać je na LCD. Dzięki temu nie będzie nam potrzebny bufor przechowujący rastrową kopię wyświetlanego obrazu, a to z kolei spowoduje, że nie będziemy potrzebowali dużej ilości pamięci GRAM. Obniży to koszty produkcji układu, a także znacznie ograniczy liczbę danych transferowanych między CPU a kontrolerem graficznym. W tym momencie procesor przesyła do kontrolera polecenia odpowiedzialne za rysowanie konkretnych obiektów, co wymaga przesłania tylko niewielkiej liczby danych. Tak niewielkiej, że CPU możemy połączyć z kontrolerem za pomocą magistrali szeregowej — I2C, SPI lub QSPI. Upraszcza to połączenia — w najprostszym przypadku do połączenia CPU z kontrolerem potrzebować będziemy tylko dwóch przewodów (I2C), a w najgorszym sześciu (QSPI). To naprawdę niewiele, zważywszy że do realizacji magistrali 8-bitowej *I8080* potrzebujemy co najmniej jedenastu połączeń.

Firma FTDI do tej pory wypuściła na rynek sześć układów serii *FT8xx*, będących akceleratorami graficznymi dostosowanymi do pracy z matrycami o różnej rozdzielczości. Krótkie podsumowanie podstawowych różnic między nimi zawiera tabela 13.1.

Tabela 13.1. *Podstawowe różnice między przedstawicielami rodziny akceleratorów FT8xx*

Typ	Głębia kolorów	Maksymalna rozdzielczość	Kontroler panelu dotykowego
FT800	18-bitowa, 6 bitów na kanał	512 x 512 pikseli	rezystancyjny
FT801	18-bitowa, 6 bitów na kanał	512 x 512 pikseli	pojemnościowy
FT810	18-bitowa, 6 bitów na kanał	800 x 600 pikseli	rezystancyjny
FT811	18-bitowa, 6 bitów na kanał	800 x 600 pikseli	pojemnościowy
FT812	24-bitowa, 8 bitów na kanał	800 x 600 pikseli	rezystancyjny
FT813	24-bitowa, 8 bitów na kanał	800 x 600 pikseli	pojemnościowy

Jak widzimy, główne różnice związane są z typem obsługiwanego panelu dotykowego (rezystancyjny/pojemnościowy) oraz maksymalną rozdzielczością i głębią kolorów. Na wstępie warto powiedzieć o pewnej „wadzie" układów *FT8xx*. Otóż dostępne są one wyłącznie w obudowach VQFN o 48 lub 56 wyprowadzeniach, z rozstawem 0,5 mm. Ich lutowanie stanowi więc nie lada wyzwanie dla elektronika amatora, jednak w sprzedaży dostępne są gotowe moduły z tymi kontrolerami, które możemy połączyć z MCU za pomocą interfejsu szeregowego. Niestety gotowe moduły mają wadę, jaką jest ich cena… Na szczęście coraz więcej firm oferuje moduły z *FT8xx* w postaci samej elektroniki lub płytki w połączeniu z LCD — tego typu moduły można kupić już w cenie zaczynającej się od około 100 złotych. Jest to całkiem niezła oferta, gdyż dają one naprawdę duże możliwości i bardzo dobrą jakość.

Przyjrzyjmy się więc nieco bliżej akceleratorom serii *FT8xx*. Są to układy integrujące w sobie kontroler panelu LCD-TFT, układ audio umożliwiający odtwarzanie muzyki oraz kontroler panelu dotykowego (rezystancyjnego lub pojemnościowego). Zacznijmy od możliwości graficznych. Układ umożliwia sprzętowe wyświetlanie punktów, linii, prostokątów, bitmap (skompresowanych lub nie, także w formatach JPEG i PNG), tekstu, a niektóre układy pozwalają także dekompresować wideo. Dodatkowo w stosunku do wyświetlanych obiektów możemy zastosować antyaliasing, alfablending i maskowanie, dzięki czemu możemy stworzyć skomplikowane sceny graficzne. Wszystkie obiekty

przed wyświetleniem podlegają zabiegowi antyaliasingu, co gwarantuje najwyższą możliwą jakość wyświetlanego obrazu. Każdy obiekt może być poddany różnym transformacjom, na przykład obrazy mogą być rozciągane, obracane o zadany kąt itd. Układy *FT81x* mogą dodatkowo dekompresować i wyświetlać na LCD przesyłane informacje wideo. Wszystkie układy są wyposażone w tak zwany koprocesor graficzny, który potrafi wyświetlać różnego typu bardziej złożone obiekty związane z tworzeniem graficznego menu i systemów okienkowych: przyciski, paski przewijania, paski postępu, przełączniki on/off, pokrętła, klawisze, zegary itd. Dzięki temu operacje graficzne, które wymagałyby przesłania znacznej ilości danych, realizowane są za pomocą prostego, 4-bajtowego polecenia. Ponadto z każdym obiektem graficznym możemy powiązać zdefiniowany znacznik, dzięki czemu wbudowany kontroler panelu dotykowego nie tylko przekaże nam informację o miejscu dotyku, ale także powiąże ją z konkretnym obiektem. Znacznie ułatwia to budowanie GUI, gdyż właściwie wszystkie główne funkcje realizowane są sprzętowo przez kontroler. Kontroler ma również bogaty zestaw wbudowanych czcionek z możliwością ich rozszerzenia o dodatkowe czcionki zdefiniowane przez użytkownika. Ta ostatnia cecha jest istotna, ponieważ wbudowane zestawy czcionek nie obejmują polskich liter. Dlatego jeśli chcemy z nich korzystać, musimy zdefiniować własne zestawy czcionek.

Ceną, jaką płacimy za rozbudowane funkcje układów graficznych serii *FT8xx*, jest ich duży stopień złożoności. Niestety, aby rozpocząć z nimi pracę, musimy sporo się dowiedzieć, a i napisanie programu będącego interfejsem między aplikacją a kontrolerem (drivera) nie jest tak banalne, jak to było w przypadku kontrolerów pokazanych w poprzednich rozdziałach. Mogę jednak zapewnić, że czas poświęcony na naukę bez wątpienia nie będzie zmarnowany, a kto raz zasmakował w możliwościach kontrolerów *FT8xx*, nie wróci już do innych rozwiązań. Duży stopień złożoności układu powoduje, że te same problemy możemy rozwiązać na różne sposoby. W rezultacie wiele będzie zależało od sposobu, w jaki napiszemy sterownik umożliwiający komunikację z kontrolerem LCD. Tu jednak warto skupić się na przykładowych rozwiązaniach dostarczonych przez firmę FTDI — nie są one idealne, w wielu miejscach kod jest przesadnie skomplikowany i można go znacznie zoptymalizować, lecz mamy od razu gotowy uniwersalny kod obsługujący wszystkie typy układów *FT8xx*. Jest jeszcze jeden powód, dla którego warto trzymać się kodu referencyjnego, a mianowicie FTDI dostarcza oprogramowanie do projektowania i budowania grafiki z elementów wspieranych przez układy *FT8xx*, a gotowy projekt możemy wyeksportować w postaci kodu w języku C dla mikrokontrolera. Oczywiście domyślamy się, że wygenerowany kod współpracuje poprawnie z kodem referencyjnym sterownika. **Warto jednak uspokoić osoby, które od razu chciałyby ten kod optymalizować — nie ma to większego sensu.** Cały kod po skompilowaniu zajmuje 2 – 4 kB (w zależności od tego, czy korzystamy tylko z wybranych czy z wszystkich funkcji) i jego skrócenie nie odbije się znacznie na zajętości pamięci FLASH mikrokontrolera. Również optymalizacja pod względem szybkości działania niewiele daje — każda funkcja jest realizowana sprzętowo przez kontroler, zatem jej wykonanie trwa tyle co przesłanie odpowiednich poleceń (zwykle kilku bajtów) do kontrolera graficznego. Nawet jeśli byśmy proces przesyłania danych znacznie zoptymalizowali, to nie wpłynie to istotnie na wydajność całego systemu. Dlatego w przykładach opisujących kontrolery *FT8xx* zdecydowałem się na użycie oryginalnych kodów FTDI z drobnymi modyfikacjami umożliwiającymi ich dostosowanie do współpracy z mikrokontrolerami XMEGA lub ARM (albo po drobnych modyfikacjach z dowolnym mikrokontrolerem). W pokazanych kodach usunięte zostały także zbędne elementy związane z możliwością ich kompilacji na różnych platformach sprzętowych, w tym

na komputerach klasy PC. W praktyce nigdy z tego nie będziemy korzystali, a zawarte w kodzie dyrektywy warunkowej kompilacji znacznie zmniejszały jego czytelność.

W kolejnych rozdziałach poznamy bliżej rejestry kontrolera i jego możliwości. Na początek jednak musimy poznać pewną ogólną koncepcję ich działania i komunikacji z MCU, dzięki czemu lepiej zrozumiemy, w jaki sposób tworzony jest obraz.

Połączenia elektryczne

Jak w przypadku każdego układu, musimy zacząć od podłączenia go z wykorzystywanym MCU. Tu sprawa jest prosta. Zacznijmy od zasilania. Układy *FT8xx* możemy zasilać napięciem z zakresu od 2,97 do 3,63 V, a więc idealnie nadają się do zasilania napięciem 3,3 V, tak jak wykorzystywane przez nas mikrokontrolery. Dodatkowo mamy do dyspozycji pin *VCCIO*, na który możemy podać napięcie z zakresu od 1,62 do 3,63 V, które będzie zasilać bufory wyjściowe akceleratora. Dzięki temu nie ma problemu z połączeniem akceleratora z mikrokontrolerami pracującymi z niższym napięciem, nawet 1,8 V.

> Niestety w większości modułów zawierających układy *FT80x* pin *VCCIO* nie jest wyprowadzony i jest połączony na stałe z pinem *VCC* zasilającym układ. Korzystając z takich modułów, będziemy mogli pracować wyłącznie w logice dostosowanej do poziomu 3,3 V.

A co mają zrobić osoby korzystające na przykład z AVR8 czy innych mikrokontrolerów zasilanych napięciem 5 V? Znakomita większość modułów zawierających te akceleratory ma konwertery poziomów logicznych z wejściami tolerującymi 5 V, dlatego stosując tego typu moduły, możemy dostępne piny IO łączyć bezpośrednio z mikrokontrolerem, bez pośrednictwa jakichkolwiek układów. Jest to z pewnością bardzo wygodne, szczególnie w przypadku posiadaczy modułów Arduino.

> Pamiętaj jednak, że takie bezpośrednie połączenie możliwe jest wyłącznie w przypadku modułów z konwerterami napięć (na przykład oryginalnych modułów oferowanych przez firmę FTDI), w żadnym razie nie można bezpośrednio łączyć MCU zasilanego napięciem 5 V z samych chipem *FT80x*.

Gotowy moduł czy samoróbka?

Układy *FT80x* dostępne są w obudowach QFN48, które są raczej trudne do lutowania w warunkach amatorskich. Tu w zdecydowanie lepszej sytuacji są posiadacze stacji *hot-air* — lutowanie tego typu obudowy z użyciem pasty i gorącego powietrza jest banalnie proste. Przy pewnej wprawie QFN można polutować także zwykłą lutownicą, a nawet lutownicą transformatorową[1], lecz wymaga to pewnych ćwiczeń. Jak ominąć

[1] Jest to jedyna lutownica, jaką posiada autor, dlatego wszystkie prezentowane układy zostały zlutowane z użyciem lutownicy transformatorowej, czego jednak autor nie poleca, gdyż jest ona źródłem ciągłych frustracji i problemów.

ten problem? Najprościej jest kupić gotowy moduł z już przylutowanym układem oraz przylutowanym złączem LCD i panelu dotykowego, a także wyprowadzonymi potrzebnymi sygnałami dla MCU. Tego typu moduły mają też przetwornicę generującą napięcie wymagane do podświetlenia LCD i układ wzmacniacza audio. Moduły takie można kupić w cenie poniżej 100 złotych, co jest ofertą bardzo atrakcyjną, i to z co najmniej dwóch powodów. Pierwszym jest brak konieczności lutowania małych elementów SMD, druga zaleta wynika z możliwości wykorzystania „gołych" wyświetlaczy LCD. Do takiego modułu podłączamy samą matrycę LCD, bez kontrolera. Tego typu matryce są zdecydowanie tańsze, a matrycę LCD-TFT z panelem dotykowym o rozdzielczości 480 na 272 pikseli można kupić w cenie nawet poniżej 30 złotych za sztukę. Taka matryca z wbudowanym kontrolerem (na przykład układem z serii SSD) może kosztować 100 i więcej złotych.

Dla osób, które nie boją się lutowania elementów SMD, dobrą wiadomością będzie zapewne informacja, że układy *FT80x* można bez problemu zakupić w cenie tylko trochę wyższej niż 20 złotych za sztukę, na dodatek w przeciwieństwie do innych kontrolerów LCD są łatwo dostępne.

Matryca LCD-TFT

Mamy więc załatwione zasilanie, czas na dobór odpowiedniej matrycy LCD-TFT. Tu wybór mamy spory. Najprościej jest oczywiście zakupić moduł firmy FTDI od razu z odpowiednim wyświetlaczem, lecz nie jest to konieczne. Układy *FT80x* współpracują ze wszystkimi LCD o rozdzielczości do 512 na 512 pikseli (a układy serii *FT81x* także z matrycami o wyższej rozdzielczości), dlatego możemy wykorzystać bardzo popularne, a co za tym idzie tanie wyświetlacze o rozdzielczości 480 na 272 lub 320 na 240 pikseli. Musimy jednak zwrócić uwagę na sposób wyprowadzenia sygnałów z LCD. Jeśli nie korzystamy z modułu, lecz sami projektujemy płytkę pod *FT80x*, to nie ma problemu — prowadzone sygnały możemy dostosować do ich rozkładu w zakupionej matrycy LCD. Jeśli jednak korzystamy z gotowego modułu, to matryca musi mieć taki sam rozkład sygnałów, jaki mamy do dyspozycji na złączu ZIF dostępnym w module. Niezgodności uniemożliwią bezpośrednie połączenie LCD z modułem i zmuszą nas do wykonania odpowiedniej przejściówki. Jest to duży problem, gdyż sygnały z matryc LCD wyprowadzone są za pomocą elastycznych taśm pasujących do złączy ZIF. Zwykle raster wyprowadzeń nie przekracza 1 mm.

Na rynku dostępne są matryce LCD z wyprowadzeniami w różnych standardach, lecz pewne rozkłady sygnałów są częściej spotykane. W tabeli 13.2 pokazany został rozkład sygnałów dostępnych na złączu ZIF modułów firmy FTDI dla wyświetlaczy o przekątnej 3,5 cala oraz 4,3/5 cali.

Decydując się na zakup matrycy LCD, należy porównać rozkład dostępnych sygnałów z rozkładami pokazanymi w tabeli. Jeśli nie są one zgodne, to lepiej zakupić inną matrycę. Uważnie przyglądając się pokazanemu rozkładowi sygnałów, zauważymy, że zamiast 24 sygnałów koloru RGB mamy tylko 18. Gdzie podziało się pozostałych 6? Otóż kontrolery *FT80x* używają tylko 18 linii sygnału koloru, stąd mamy do dyspozycji „tylko" 6-bitową głębię dla każdego koloru podstawowego, co daje nam łącznie

Tabela 13.2. *Rozkład sygnałów na złączu ZIF dla modułów o przekątnej 3,5 cala i 5,0 cali. Dla modułów o przekątnej 3,5 cala stosowane są złącza 54-stykowe, a dla modułów 4,3/5 cali złącza 40-stykowe*

Nr pinu	Sygnał na złączu ZIF54	Sygnał na złączu ZIF40	Nr pinu	Sygnał na złączu ZIF54	Sygnał na złączu ZIF40
1	LED_K	LED_K	28	GND	B7
2	LED_K	LED_A	29	GND	GND
3	LED_A	GND	30	B2	DCLK
4	LED_A	3V3	31	B3	DISP
5	NC	GND	32	B4	HSYNC
6	NC	GND	33	B5	VSYNC
7	NC	R2	34	B6	DE
8	DISP	R3	35	B7	NC
9	3V3	R4	36	HSYNC	GND
10	3V3	R5	37	VSYNC	XP
11	3V3	R6	38	DCLK	YM
12	GND	R7	39	NC	XM
13	GND	GND	40	NC	YP
14	R2	GND	41	3V3	–
15	R3	G2	42	3V3	–
16	R4	G3	43	NC	–
17	R5	G4	44	NC	–
18	R6	G5	45	NC	–
19	R7	G6	46	NC	–
20	GND	G7	47	NC	–
21	GND	GND	48	XP	–
22	G2	GND	49	YM	–
23	G3	B2	50	XM	–
24	G4	B3	51	YP	–
25	G5	B4	52	DE	–
26	G6	B5	53	GND	–
27	G7	B6	54	GND	–

262 144 możliwe do wyświetlenia kolory zamiast normalnie oczekiwanych ponad 16 milionów. Czy jest to problem? Otóż nie — dostępne matryce LCD-TFT pomimo 24-bitowego interfejsu RGB i tak nie są w stanie wyświetlić 16 milionów kolorów, zazwyczaj właśnie wyświetlają 262 tysiące, a brakujące kolory symulowane są za pomocą różnych technik, między innymi ditheringu. Dlatego na pokazanych złączach bity *R0 – R1*, *G0 – G1* i *B0 – B1* połączone są na stałe z masą, a układ steruje tylko bardziej znaczącymi 6 bitami określającymi każdy z kolorów RGB.

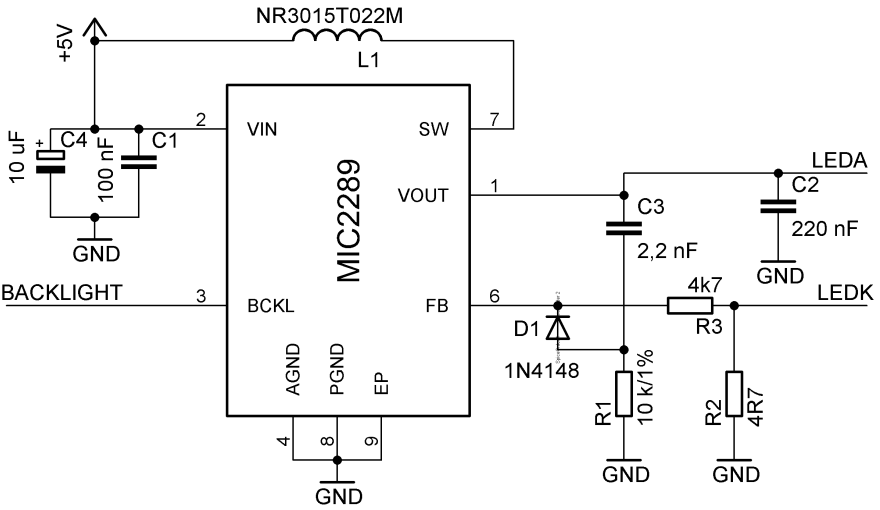
W przypadku, gdy sami projektujemy PCB, musimy pamiętać, aby niewykorzystane sygnały kolorów połączyć z masą w celu wymuszenia na nich określonego stanu logicznego.

Warto też pamiętać, że jeśli kupiona matryca LCD nie ma panelu dotykowego, to sygnały *XP*, *YM*, *XM* i *YP* pozostawiamy niepodłączone.

Dla osób, które potrzebują z jakichś powodów 24-bitowej głębi koloru, dostępne są układy serii *FT81x* dysponujące taką możliwością. Jeśli jednak matryca LCD istotnie ma prawidłowo wyświetlać taką liczbę kolorów, to musimy przygotować się na spory wydatek. Nie ma sensu kupować w takim przypadku tanich matryc, które nie spełnią naszych oczekiwań.

Podświetlenie

Moduły zawierające akcelerator *FT80x* wyposażone są także w układ przetwornicy generującej wymagane napięcia do podświetlenia modułu LCD. Sam akcelerator ma rejestry umożliwiające wybór częstotliwości sterowania i wypełnienia przebiegu PWM sterującego przetwornicą, dzięki czemu możliwe jest programowe sterowanie jasnością podświetlenia i na przykład dostosowanie jej do warunków otoczenia. Przebieg PWM do sterowania przetwornicą wyprowadzony jest na wyprowadzeniu *BACKLIGHT* akceleratora. Przetwornicę możemy zrealizować w dowolny sposób, w zależności od wymogów układu podświetlenia zastosowanej matrycy. To, co nas interesuje, to napięcie i prąd, jaki pobiera podświetlenie. Ponieważ matryce TFT mają podświetlenie w postaci LED-ów, tak naprawdę interesuje nas stabilizacja prądu. Schemat przetwornicy zastosowanej w układach FTDI został pokazany na rysunku 13.1.



Rysunek 13.1. *Schemat przetwornicy zasilającej podświetlenie panelu LCD-TFT*

Pokazana przetwornica znajduje się na modułach firmy FTDI. Jej zadaniem jest dostosowanie napięcia zasilania (z zakresu 2,5 – 10 V) do wymogów podświetlenia panelu LCD-TFT. W tym celu podwyższa ona napięcie maksymalnie do 34 V, jednocześnie kontrolując prąd podświetlenia[2]. Ponieważ stosowane matryce mają różnie rozwiązane podświetlenie i może ono wymagać różnych prądów, wymagany prąd podświetlenia dobiera się rezystorem *R2* — dla wartości 4,7 Ω prąd podświetlenia zostanie ograniczony do 20 mA. Jest to prąd typowy dla paneli LCD-TFT o przekątnej do około 3,5 cala, natomiast panele o dłuższych przekątnych wymagają nieco większego prądu. Dla panelu o przekątnej 4,3 cala może to być około 32 mA, dlatego *R2* powinniśmy zmniejszyć do 3 Ω. Dowolny prąd możemy ustawić, korzystając z następującej zależności:

$$I_{LED} = \frac{95mV}{R2}$$

Dane na temat wymaganego prądu podświetlenia zawsze znajdziemy w nocie katalogowej użytej matrycy LCD. Jeśli korzystamy z modułów firmy FTDI, a prąd użytej matrycy znacznie odbiega o powyższych wartości, możemy wymienić rezystor pomiarowy (zawsze należy sprawdzić schemat stosowanego modułu w celu jego lokalizacji).

W ten sposób mamy elegancko załatwione podświetlenie i sterowanie nim. Przetwornica impulsowa jest o tyle fajna, że cechuje się ona wysoką sprawnością, przekraczającą 80%, a jej zastosowanie eliminuje konieczność stosowania nietypowych napięć (zasilanie podświetlenia paneli LCD wymaga napięć rzędu 24 – 34 V) i umożliwia poprawne, prądowe sterowanie podświetleniem z możliwością regulacji intensywności programowo, poprzez modulację współczynnika wypełnienia PWM przebiegu generowanego przez *FT80x*.

Interfejs MCU – akcelerator

Zacznijmy od połączenia akceleratora z MCU. Ponieważ akcelerator wykonuje polecenia graficzne, co nie wymaga przesyłania dużej liczby danych, może on być połączony z mikrokontrolerem za pomocą jednego z trzech interfejsów:

- I2C — wymaga on tylko dwóch przewodów. Jednak stopień złożoności jego programowej obsługi i niewygoda z tego wynikająca powodują, że w większości aplikacji z niego nie korzystamy. W tej książce pominiemy tę możliwość, ale jeśli Czytelnik mimo wszystko postanowi go wykorzystać, to może posłużyć się kodami obsługi I2C pokazanymi w poprzednich rozdziałach.
- SPI — jest to podstawy interfejs komunikacyjny, relatywnie szybki i bardzo wygodny w obsłudze ze względu na jego prostotę. W poprzednich rozdziałach bardzo szeroko z niego korzystaliśmy, więc jego użycie nie powinno już przedstawiać żadnych problemów. Układy *FT8xx* obsługują SPI taktowane z częstotliwością do 10 MHz przed inicjalizacją układu lub do 30 MHz po skonfigurowaniu zegara taktującego akcelerator.

[2] Jak pamiętamy, w przypadku LED-ów ważny jest prąd płynący przez diodę.

- QSPI — jest on obsługiwany wyłącznie przez akceleratory serii *FT81x*, a więc nowsze układy FTDI. Jest to związane z tym, że mają one większy bufor RAM (1 MB), a także obsługują transfery wideo wymagające nieco większej przepustowości. Niestety sprzętowa obsługa QSPI jest dostępna tylko w niektórych mikrokontrolerach ARM, nie jest dostępna dla 8-bitowych mikrokontrolerów. Jeśli nie dysponujemy interfejsem QSPI, to również nowsze układy możemy obsługiwać za pomocą interfejsu SPI lub I2C.

W dalszych przykładach będziemy korzystali głównie ze znanego nam interfejsu SPI. Jak wiemy, wymaga on podłączenia sygnałów *CS*, *MISO*, *MOSI* i *SCK*, dzięki którym następuje wymiana danych. W przypadku interfejsu QSPI wykorzystywane są sygnały *CS* i *SCK*, natomiast zamiast *MISO* i *MOSI* mamy cztery dwukierunkowe sygnały *IO0 – IO3*. Domyślnie po aktywacji *CS* układy *FT81x* utrzymują piny *IO0 – IO3* jako wejścia, oczekując na dane z MCU. W przypadku odczytu danych z *FT81x* procesor najpierw wysyła polecenie odczytu danych, potem przesyła tak zwany *dummy byte* (bajt bez znaczenia) — w tym czasie sygnały *IO0 – 3* nie powinny być sterowane ani przez procesor, ani przez układ *FT81x*, a następnie sygnały *IO0 – 3* stają się wyjściami układu *FT81x*, a procesor może odczytać ich stan. Z tego schematu transmisji wynika, że okresowo linie *IO0 – 3* nie są sterowanie ani przez procesor, ani przez akcelerator, dlatego aby ich stan nie „pływał", powinny być podpięte przez rezystory podciągające o wartości około 4,7 kΩ do *Vcc*. Widzimy, że także obsługa programowa QSPI jest nieco bardziej skomplikowana, stąd warto ją wykorzystać wyłącznie w sytuacji, gdy planujemy odtwarzać filmy, a w efekcie przepustowość interfejsu SPI może okazać się niewystarczająca. Domyślnie akceleratory *FT81x* pracują w trybie SPI, który możemy zamienić na tryb dual-SPI lub quad-SPI programowo[3].

Kontroler wymaga jeszcze dwóch dodatkowych sygnałów: *PDN* i *INT_N*. Sygnał *PDN* powoduje dezaktywację kontrolera i jego uśpienie. W trakcie normalnej pracy powinien mieć stan wysoki. Możemy go wykorzystać do resetowania układu. Jeśli nie jest nam potrzebny, to możemy go podpiąć poprzez rezystor 47 kΩ do *Vcc*. Drugi sygnał to *INT_N* — jak się domyślamy, jest to sygnał przerwania wysyłany przez kontroler dla poinformowania MCU, że zaszło jakieś zdarzenie wymagające jego interwencji. Nie jest on niezbędny, lecz jego wykorzystanie czasami ułatwia obsługę zdarzeń generowanych przez użytkownika (na przykład dotknięcie panelu dotykowego). Sygnał ten domyślnie jest skonfigurowany jako wyjście typu otwarty dren, ale w układach *FT81x* programowo możemy zamienić go na sygnał typu push-pull. Warto podłączyć go przez rezystor 1 – 10 kΩ do *Vcc* w celu wymuszenia określonego stanu tego sygnału także w czasie resetu kontrolera.

W układach serii *FT80x* wyjście *INT_N* jest typu otwarty dren. Aby w stanie nieaktywnym sygnał ten miał określony stan, musimy podpiąć go przez rezystor o wartości kilku kiloomów do *Vcc*. W układach *FT81x* to samo możemy osiągnąć programowo. W obu przypadkach możemy wykorzystać także wbudowane w pin IO mikrokontrolera programowo włączane rezystory podciągające.

[3] Dla trybu QSPI maksymalna częstotliwość magistrali wynosi 25 MHz, dla pozostałych trybów 30 MHz.

Aby łączenie MCU z układami *FT8xx* było łatwiejsze, te ostatnie wyposażono w pin *VCCIO* — napięcie podane na ten pin wykorzystywane jest do sterowania interfejsem SPI. Dlatego napięcie *VCCIO* powinno być takie samo jak napięcie zasilania MCU, dzięki temu bowiem unikniemy konieczności konwersji napięć. Ponieważ układy *FT8xx* mogą być zasilane napięciem 3,3 V, to zazwyczaj napięcie zasilania MCU, *FT8xx* i *VCCIO* podłączamy do napięcia 3,3 V. Jeśli MCU pracuje z niższym napięciem, to *VCCIO* podłączamy do napięcia zasilającego MCU (pin toleruje napięcia w zakresie 1,8 – 3,3 V). Na modułach z układami *FT8xx* często znajdziemy przełącznik lub rezystor podłączony do pinu *MODE*. Wejście to umożliwia wybór typu interfejsu łączącego *FT8xx* z MCU. Stan niski tego sygnału wybiera interfejs SPI, a stan wysoki — interfejs I2C. Różne typy interfejsu SPI (SPI, dual-SPI lub quad-SPI) wybierane są programowo. **Warto pamiętać, że SPI zawsze pracuje w trybie 0 (MODE 0).**

Pamiętaj, że układ może być zasilany napięciem z zakresu od 2,97 do 3,63 V, typowo 3,3 V. Układ nie może być zasilany napięciem 5 V, ponieważ spowoduje to jego nieodwracalne uszkodzenie. Dlatego łącząc kontrolery *FT8xx* z mikrokontrolerem zasilanym napięciem wyższym niż 3,63 V, musimy zastosować odpowiedni translator poziomów napięć.

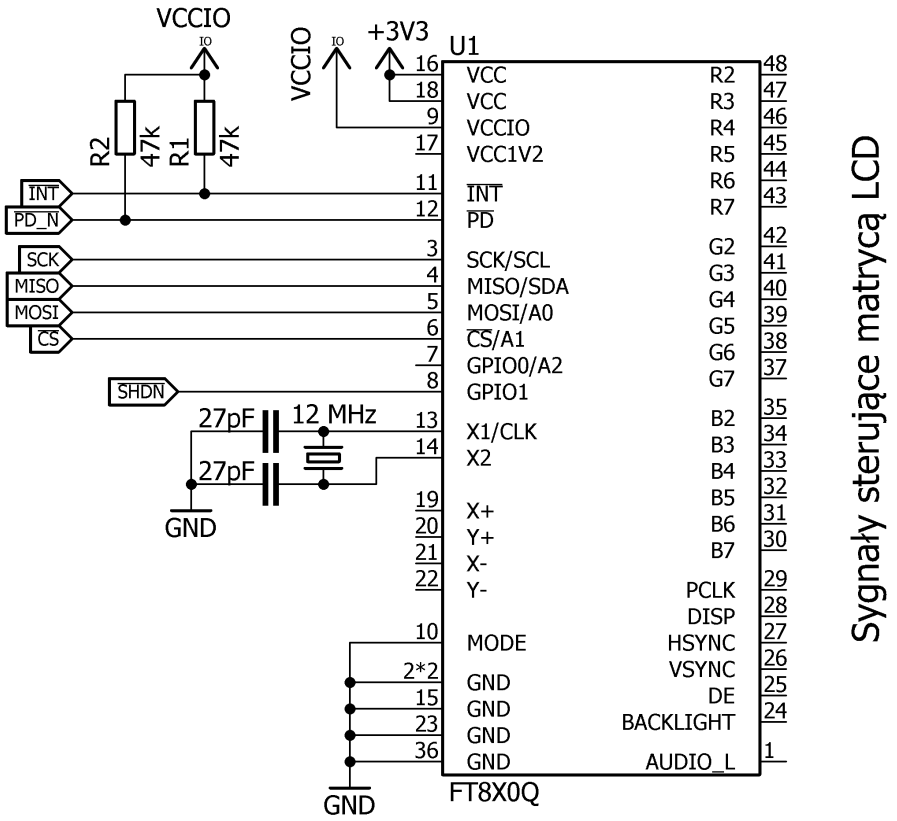
Wiele gotowych modułów ma stabilizator LDO, dzięki czemu mogą być zasilane napięciem 5 V. W takim przypadku moduł ma także translator napięć pomiędzy domenami 5 V i 3,3 V. Przed pierwszym podłączeniem modułu zawsze upewnij się, czy został on prawidłowo skonfigurowany. Tego typu moduły można zasilać również napięciem 3,3 V — wówczas musimy zewrzeć zworkę pomiędzy pinem wejściowym i pinem wyjściowym stabilizatora LDO. **Po ich zwarciu trzeba pamiętać, aby napięcie zasilające nie przekroczyło 3,63 V.**

Rysunek 13.2 przedstawia schemat przykładowego połączenia układu *FT80x* z mikrokontrolerem. Na schemacie nie pokazano pomocniczych elementów znajdujących się na gotowych modułach zawierających układy serii *FT8xx*. Na zamieszczonym schemacie *VCCIO* to napięcie zasilania mikrokontrolera. Jeśli jest ono takie samo jak napięcie zasilania akceleratora, to oba układy mogą być zasilane ze wspólnego źródła, a *VCC* można połączyć z *VCCIO*. W naszych przykładach zakładamy, że pokazane sygnały sterujące akceleratora połączone są z pinami IO mikrokontrolera XMEGA lub ARM tak, jak to zostało wskazane w tabeli 13.3.

Tabela 13.3. *Przyporządkowanie pinów portów IO do sygnałów sterujących akceleratorem graficznym*

Sygnał	XMEGA	ARM SAM D21
INT	*PORTE4*	*PA10*
PDN	*PORTE5*	*PA20*
MISO	*PORTE2*	*PA16*
MOSI	*PORTE3*	*PA18*
SCK	*PORTE1*	*PA19*
CS	*PORTE0*	*PA17*

Jeśli w posiadanym układzie wykonamy inne połączenia, należy to uwzględnić, zmieniając przyporządkowanie pinów w pliku *FT_SPI.h*.



Rysunek 13.2. *Schemat połączenia modułu zawierającego kontroler FT8xx z mikrokontrolerem. Zazwyczaj wszystkie pokazane elementy znajdują się na module, zatem musimy połączyć jedynie sygnały interfejsu SPI*

Trochę teorii związanej z tworzeniem obrazu

Projektanci *FT80x* wskrzesili pomysł na tworzenie grafiki sprzed 20 – 30 lat. Ale to, że pomysł jest stary, nie znaczy, że jest zły. Tworząc grafikę na LCD, myślimy „od końca", to znaczy mamy wizję, jak ma wyglądać finalny obraz, lecz tę wizję musimy przetłumaczyć na język zrozumiały dla mikrokontrolera. W efekcie obraz rozbijamy na elementy — powtarzające się bloki (cegiełki), prymitywy graficzne (teksty, linie, figury geometryczne, bitmapy) — a następnie za pomocą napisanych wcześniej funkcji elementy te rozbijamy na jeszcze mniejsze składowe, a mianowicie bajty danych, które zapisujemy w odpowiednich miejscach pamięci GRAM. I ten ostatni etap jest najbardziej czasochłonny, gdyż generuje sporo obliczeń i jeszcze większe ilości danych,

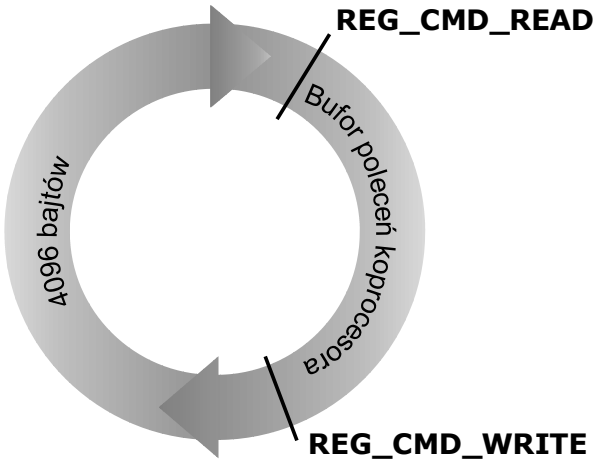
które musimy przetransferować pomiędzy MCU a GRAM, w dodatku często przez wąskie gardło, jakim jest interfejs SPI. A gdyby tak zrezygnować z tego ostatniego etapu? W końcu prościej wydać sterownikowi LCD polecenie typu „narysuj prostą od punktu A do punktu B" albo „w danym miejscu wyświetl napis" niż własnoręcznie (a dokładniej przy udziale MCU) generować potrzebne informacje dla sterownika. Z tego właśnie założenia wyszli twórcy układów *FT8xx*. Układy te w ogóle nie mają pamięci będącej buforem ramki, a obraz tworzony jest przez umieszczoną w pamięci kontrolera listę wyświetlanych obiektów (ang. *display list*). Dzięki temu zamiast pracowicie renderować obraz przez MCU, tak aby kontroler mógł go wyświetlić, przerzucamy całą pracę na kontroler, wysyłając do niego tylko informacje o obiektach, jakie mają zostać utworzone, i ich atrybutach (położeniu, przezroczystości, kolorach itd.). Ponieważ stworzenie takiej listy wyświetlanych obiektów jest procesem niezwykle prostym i nie obciąża on mikrokontrolera, praktycznie za pomocą dowolnego mikrokontrolera, nawet najprostszego, możemy tworzyć zaawansowaną grafikę, która do tej pory była dostępna wyłącznie w przypadku użycia potężnych mikrokontrolerów. Co więcej, lista instrukcji wykorzystywanych przez nasz układ jest bardzo prosta i jej opanowanie to dosłownie kilkanaście minut — aczkolwiek umiejętne łączenie poszczególnych obiektów wymaga pewnych ćwiczeń. No dobrze, ktoś przecież powie, że obraz nie składa się tylko z linii, kół, okręgów i tekstów. Trudno w ten sposób wyświetlić zdjęcie. I na to twórcy *FT80x* znaleźli rozwiązanie: jednym z wyświetlanych prymitywów graficznych są mapy bitowe, a dokładniej obrazy zapisane w różnych formatach, między innymi BMP czy JPEG. Wystarczy przesłać do kontrolera taki plik, a następnie na liście poleceń umieścić polecenie jego wyświetlenia w określonym miejscu i mamy problem z głowy.

Podobne pomysły na tworzenie grafiki były wykorzystywane w stacjach graficznych z lat 70. i 80. Pomysł nie jest zatem nowy, ale dzięki współczesnej elektronice jego realizacja jest znacznie lepsza, a co ważniejsze, dostępna dla elektronika hobbysty.

Mamy już załatwioną sprawę elektrycznych połączeń z MCU, pora na utworzenie obrazu. Jak już wiemy, kontrolery *FT8xx* nie mają tak zwanego bufora ramki — obszaru pamięci GRAM zawierającego tworzony obraz. Stąd też obraz tworzony jest na podstawie *Display List* (DL) — 8 kB obszaru RAM, w którym przechowywane są polecenia tworzące obraz. Każde polecenie kodowane jest za pomocą 32-bitowego pola, dlatego na naszej liście może znaleźć się maksymalnie 2048 poleceń tworzących obraz. Kontroler graficzny podczas wyświetlania każdej ramki obrazu przegląda listę i realizuje umieszczone na niej polecenia. Stąd jeśli chcemy zmienić zawartość ekranu, musimy zmienić polecenia umieszczone na DL. Możemy to zrobić poprzez bezpośredni zapis do pamięci przeznaczonej dla DL, której adres wskazuje zdefiniowany symbol o nazwie `RAM_DL`, lub poprzez koprocesor, który zapisze komendy GPU będące rezultatem wykonania tak zwanych poleceń koprocesora. Każda nowa DL rozpoczyna się od adresu `RAM_DL` i może mieć maksymalnie 8192 bajty.

Każdą DL kończy polecenie `DISPLAY`, które powoduje wygenerowanie i wyświetlenie obrazu definiowanego przez bieżącą DL. W kolejnych rozdziałach pokażę, jak tworzyć własne DL i jakie polecenia są obsługiwane. Ponieważ ręczne tworzenie DL jest raczej uciążliwe i podatne na błędy, firma FTDI udostępnia za darmo oprogramowanie ułatwiające tworzenie obrazu i generowanie odpowiadających mu DL, co zostanie pokazane w kolejnym rozdziale.

DL zawiera podstawowe polecenia dla akceleratora, bezpośrednio związane z tworzeniem obrazu. Ponieważ pewne elementy, takie jak napisy lub elementy interfejsu graficznego użytkownika (przyciski, pokrętła itd.), wymagają do ich utworzenia ciągu instrukcji w DL, twórcy układu *FT8xx* postanowili nam nieco ułatwić zadanie, wprowadzając tak zwany koprocesor graficzny. Koprocesor przyjmuje własne polecenia, które realizują wyświetlanie bardziej złożonych obiektów graficznych, a następnie dokonuje ich translacji na ciąg poleceń w DL. Dzięki temu, możemy wygodniej wykonywać pewne podstawowe operacje i operować na bardziej złożonych i zrozumiałych dla człowieka poleceniach koprocesora graficznego. Polecenia dla koprocesora graficznego umieszczane są w osobnej części pamięci RAM, o wielkości 4 kB, zbudowanej w ten sposób, że tworzy ona bufor pierścieniowy (rysunek 13.3). Ilość dostępnego miejsca w buforze można odczytać z rejestru `REG_CMDB_SPACE`. Po zapisaniu poleceń koprocesora do bufora są one wykonywane, czego efektem jest modyfikacja DL. Po ich wykonaniu koprocesor jest gotowy do przyjęcia kolejnych poleceń. Zinterpretowane polecenia koprocesora zamieniane są na polecenia DL, interpretowane przez GPU podczas tworzenia obrazu. Adres, pod który zapisywane są wyniki interpretacji poleceń przez koprocesor graficzny, określany jest wartością rejestru `REG_CMD_DL`. Dzięki temu możemy wskazać dowolne miejsce na DL, począwszy od którego zostaną dodane nowe polecenia tworzące obraz. Pocieszające jest to, że zwykle na tych rejestrach nie będziemy operować — robią to automatycznie funkcje biblioteki obsługi GPU, z których będziemy korzystać. Umożliwiają one dopisywanie nowych poleceń koprocesora, dbając o odpowiednią zawartość jego rejestrów adresowych.



Rysunek 13.3. *Pamięć poleceń koprocesora tworzy bufor pierścieniowy. Zapis polecenia dokonywany jest pod adres wskazywany przez REG_CMD_WRITE, z kolei koprocesor interpretuje polecenie, pobierając je spod adresu wskazywanego przez REG_CMD_READ. Normalnie oba rejestry wskazują na ten sam adres, co sygnalizuje, że w buforze nie ma nowych poleceń. Brak poleceń sygnalizowany jest także ustawieniem flagi przerwania INT_CMDEMPTY, dzięki czemu możemy użyć przerwań do uzupełniania listy poleceń koprocesora*

Wszystkie rejestry adresowe koprocesora zawierają zawsze wartości podzielne przez 4. Wynika to z tego, że wszystkie polecenia mają długość 4 bajtów.

Oprócz pamięci DL i pamięci dla poleceń koprocesora układy *FT8xx* dysponują pamięcią RAM ogólnego przeznaczenia. W układach serii *FT80x* mamy jej 256 kB, natomiast w układach serii *FT81x* aż 1024 kB. W pamięci RAM ogólnego przeznaczenia możemy umieścić próbki dźwiękowe, które będą odtwarzane przez podsystem audio kontrolera, a przede wszystkim dane graficzne, takie jak obrazy, które następnie będą wyświetlane na ekranie. Do danych graficznych umieszczonych w RAM odwołują się polecenia rysowania map bitowych lub odtwarzania materiału wideo znajdujące się na DL. W pamięci tej możemy także umieścić definicje nowych krojów czcionek.

Ponieważ najłatwiej będzie nam poznać polecenia koprocesora graficznego i polecenia kontrolera, korzystając z oprogramowania do projektowania grafiki na PC, w kolejnym rozdziale poznamy programy, za pomocą których będziemy mogli stworzyć nowocześnie wyglądający interfejs graficzny.

Wbudowany kontroler panelu dotykowego

Jak już wiemy, układy *FT8xx* mają także wbudowany kontroler panelu dotykowego. W zależności od wersji mamy do dyspozycji kontroler panelu rezystancyjnego lub kontroler panelu pojemnościowego obsługujący tak zwany *multitouch*. W przypadku panelu pojemnościowego układ współpracuje z poznanymi w poprzednim rozdziale kontrolerami paneli dotykowych z rodziny *FT5x06*. Komunikacja z nimi odbywa się automatycznie, a programista widzi przetworzone przez kontroler *FT8xx* dane o dotyku.

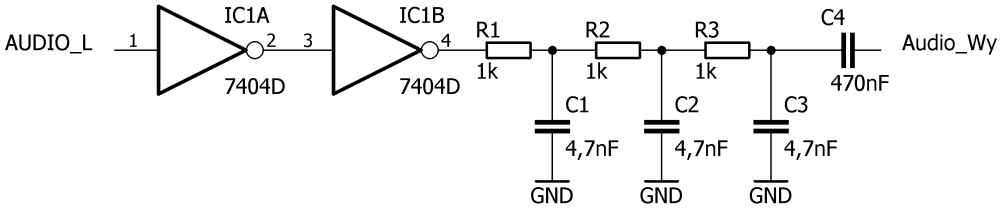
Skorzystanie z panelu dotykowego jest niezwykle łatwe — po jego uruchomieniu i ewentualnej kalibracji (która może odbywać się za pomocą wbudowanych w kontroler funkcji) odczytujemy po prostu pozycję punktu, w którym ekran został dotknięty. Co więcej, jeśli obiekty znajdujące się na DL odpowiednio oznaczymy, to nie będziemy musieli nawet odczytywać pozycji — od razu otrzymamy informację o obiekcie graficznym, którego dotknęliśmy. Układ ma możliwość identyfikacji do 255 różnych obiektów graficznych, co wystarczy dla obsługi niezwykle złożonych GUI. Oczywiście osoby lubiące wszystko robić samodzielnie mają możliwość odczytu surowych danych o położeniu. Jednak w praktyce nie ma to najmniejszego sensu i lepiej przerzucić całość pracy na kontroler.

Podsystem audio

Układ może generować także dźwięki monofoniczne — zarówno predefiniowane, zawarte w pamięci ROM układu, jak i odtwarzane z pamięci RAM, zakodowane w formacie µLAW, IMA-ADPCM lub PCM[4]. Nie są to może wybitne możliwości, lecz w większo-

[4] Więcej o sposobach kodowania i dekodowania dźwięku oraz o możliwości odtwarzania muzyki za pomocą mikrokontrolera dowiesz się z książki *AVR. Układy peryferyjne*.

ści przypadków zupełnie wystarczające. Dodatkowo mamy możliwość 256-stopniowej regulacji głośności dźwięku. Ponieważ wyjście audio ma małą obciążalność prądową, musimy do niego podłączyć bufor i wzmacniacz audio. Na wyjściu pojawia się przebieg PWM, dlatego będziemy potrzebowali filtra RC (rysunek 13.4).



Rysunek 13.4. *Wyjście AUDIO_L z układu FT8xx łączymy z buforem, na którego wyjściu umieszczamy filtr dolnoprzepustowy. Otrzymany sygnał możemy podłączyć do wzmacniacza*

Na większości gotowych modułów z układami *FT8xx* mamy już umieszczony tor analogowy, a często występuje także miniaturowy głośnik. Na płytce zwykle umieszczony jest wzmacniacz audio *TPA6205*, który daje opcjonalną możliwość włączenia lub wyłączenia toru audio. Sygnał wyłączenia audio (*AUDIO_SHDN*) najczęściej łączony jest z pinem *GPIO1* układu *FT8xx*, dzięki czemu możemy programowo włączać i wyłączać wzmacniacz. Ma to znaczenie w sytuacji, gdy nie korzystamy z możliwości generowania dźwięku — należy wyłączyć wtedy wzmacniacz (sygnał *AUDIO_SHDN* powinien mieć stan niski), w przeciwnym razie z wbudowanego w moduł głośniczka może wydobywać się cichy szum. W kolejnych rozdziałach pokażę, jak wykorzystać możliwości audio układu.

Sterownik kontrolera

Po wstępie na temat kontrolerów *FT8xx* przejdźmy do części programowej, czyli jak wyświetlić pierwszy obraz. Zanim nam się to uda, niestety czeka nas kolejna, spora porcja informacji, będziemy także musieli napisać kod sterownika umożliwiający przesłanie poleceń do akceleratora graficznego. Sam kod może wydawać się skomplikowany, ale dobra wiadomość jest taka, że piszemy go tylko raz i w większości przypadków już nigdy nie będziemy musieli do niego zaglądać. Do napisania sterownika wykorzystamy kody referencyjne udostępnione przez firmę FTDI.

Kody omówione w tym podrozdziale znajdują się w katalogu *Przyklady/FT800*. Kod, od którego rozpoczniemy zabawę, znajduje się w katalogu *FT800-startup*.

W projekcie *FT800-startup* wszystkie pliki związane z obsługą akceleratora znajdują się w katalogach *FT_Include*, gdzie znajdziemy pliki nagłówkowe zawierające definicje poleceń i funkcji, oraz *FT_src*, gdzie znajdziemy kody funkcji potrzebnych do wymiany danych między MCU a akceleratorem. Większość plików zawiera kod, który jest kompletnie niezależny od wykorzystywanej platformy sprzętowej. Jest to czysty kod w języku C, odpowiedzialny za przesyłane polecenia, realizujący kolejki zapisu i tym podobne funkcje. A skoro tak, to do tego typu kodu właściwie nigdy nie będziemy zaglądać (chyba że w celach edukacyjnych). Zależny od platformy sprzętowej, którą

wykorzystujemy, jest wyłącznie kod zaangażowany bezpośrednio w przeprowadzenie transmisji danych między MCU a akceleratorem. W naszym przykładzie stosujemy do transmisji interfejs SPI, dlatego w pliku *FT_SPI.h* znajdziemy kody funkcji realizujących transmisję danych. W pliku tym znajdziemy znane nam już z poprzednich rozdziałów funkcje odpowiedzialne za:

- konfigurację interfejsu SPI i pinów IO mikrokontrolera — `SPI_init()`;
- nadawanie i odbiór 8-bitowej danej — `uint8_t FT800_SPIRW(uint8_t ch)`;
- ustawienie zegara taktującego transmisję SPI; ponieważ przed inicjalizacją akceleratora zegar nie może pracować z częstotliwością wyższą niż 10 MHz, potrzebujemy dwóch funkcji: pierwsza, `FT800_SPICLK10M()`, ustawia taktowanie na taką właśnie częstotliwość, a druga, `FT800_SPICLKMAX()`, ustawia maksymalną dostępną w użytym mikrokontrolerze częstotliwość interfejsu, lecz nie wyższą niż 30 MHz, czyli maksymalną dopuszczaną przez akcelerator;
- zmianę stanu sygnałów *CS* i *PDN* odpowiedzialnych za wybór i uśpienie akceleratora — `FT800_CS(_Bool state)` i `FT800_PD(_Bool state)`.

Funkcji tych nie będziemy szerzej omawiać, gdyż w żaden sposób nie odbiegają one od znanych nam już przykładów z poprzednich rozdziałów.

Jeśli stosujesz mikrokontroler inny niż używane w tej książce, jedyne, co musisz zrobić, to dostosować kody wymienionych wyżej funkcji. Cała reszta kodu pozostaje bez zmian.

Ponieważ do połączenia z układem *FT8xx* możemy wykorzystać różne piny IO mikrokontrolera i różne interfejsy, na początku pliku *FT_SPI.h* zdefiniowanych zostało kilka stałych umożliwiających elastyczną konfigurację linii łączących mikrokontroler z akceleratorem bez głębszej ingerencji w strukturę pliku.

W powyższym przykładzie wykorzystujemy w przypadku mikrokontrolera XMEGA `USARTE0` należący do `PORTE` wraz z kilkoma innymi pinami sterującymi. Co ważne, w naszych przykładach przyjęliśmy, że wszystkie piny sterujące należą do tego samego portu. W przypadku mikrokontrolera ARM, tak jak w przykładach z części I, wykorzystany został układ `SERCOM0`.

Plikiem, którego zwykle nie będziemy edytować, jest *FT_DataTypes.h*. Jak sama jego nazwa wskazuje, jest to plik, w którym zdefiniowane są nazwy typów danych wykorzystywanych przez akcelerator oraz funkcje dostępu do pamięci FLASH. W różnych mikrokontrolerach dostęp do tej pamięci wymaga czasami wykorzystania specjalnych funkcji — głównie dotyczy to mikrokontrolerów rodziny AVR. Podstawowe typy danych zostały zaczerpnięte z nagłówka *stdint.h* i praktycznie nigdy nie zajdzie potrzeba zmiany tej części pliku. Potencjalnie możemy spotkać się z koniecznością edycji następującego fragmentu:

```
#define FT_PROGMEM PROGMEM
#define ft_pgm_read_byte_near pgm_read_byte_near
#define ft_pgm_read_byte pgm_read_byte
#define ft_pgm_read_word pgm_read_word
```

Definicje te są poprawne dla mikrokontrolerów z rodziny AVR, dla pozostałych powinny zostać puste, to znaczy powinny wyglądać w ten sposób:

```
#define FT_PROGMEM
#define ft_pgm_read_byte_near
#define ft_pgm_read_byte
#define ft_pgm_read_word
```

Ostatnią interesującą nas definicją w tym pliku jest definicja funkcji zwracającej losową liczbę nie większą niż x:

```
#define ft_random(x)      (random(x))
```

Zanim będziemy mogli skompilować pierwszy program wykorzystujący akcelerator, musimy jeszcze sprawdzić poprawność definicji znajdujących się w pliku *FT_Platform.h*. Jego zawartość jest właściwie samoobjaśniająca się, lecz spójrzmy, co ewentualnie w tym pliku musimy zmienić. Jego zadaniem jest dostarczenie pozostałym elementom biblioteki informacji dotyczących typu wykorzystywanego akceleratora (jaki chip jest używany), informacji o stosowanym interfejsie (my wspieramy wyłącznie SPI, z tym że dla układów *FT81x* może to być SPI, DSPI i QSPI), a także informacji, czy biblioteka ma wykorzystywać buforowanie poleceń. Ale po kolei. Zacznijmy od definicji umożliwiających wybór typu akceleratora i podłączonej do niego matrycy. Mamy tu kilka opcji:

- `VM800B43_50` lub `VM800B35` — wybór układu *FT800* z ekranem o przekątnej 5 cali lub 3,5 cala;
- `VM801B43_50` — wybór układu *FT801* z ekranem o przekątnej 5 cali.

Jeśli nie posiadamy żadnego z powyższych układów, to sami w kolejnej sekcji musimy zdefiniować pewne parametry:

- wybrać jeden z symboli (`FT_800_ENABLE`, `FT_801_ENABLE`, `FT_810_ENABLE`, `FT_811_ENABLE`, `FT_812_ENABLE` lub `FT_813_ENABLE`) określających typ używanego chipa;
- określić rozdzielczość użytego LCD (`DISPLAY_RESOLUTION_QVGA`, `DISPLAY_RESOLUTION_WQVGA`, `DISPLAY_RESOLUTION_WVGA` lub `DISPLAY_RESOLUTION_HVGA_PORTRAIT`);
- określić typ wykorzystywanego interfejsu (`ENABLE_SPI_SINGLE`, `ENABLE_SPI_DUAL` lub `ENABLE_SPI_QUAD`).

Stosowane biblioteki mogą wysyłać dane od razu do akceleratora graficznego lub korzystać z bufora w pamięci RAM, do którego dane są najpierw przesyłane, a dopiero stamtąd do akceleratora. Ta druga opcja daje pewne korzyści polegające na możliwości edycji danych w buforze czy wykorzystania do przesyłu danych układu DMA. Zazwyczaj nie są one jednak na tyle istotne, aby zrównoważyć podstawową wadę takiego rozwiązania — bufory wymagają rezerwacji aż 8 kB pamięci RAM na DL oraz 4 kB na listę poleceń dla koprocesora. W efekcie na samym początku mamy zajęte aż 12 kB pamięci, a zyski z tego płynące są raczej niewielkie. Jeśli jednak uznamy, że chcemy skorzystać z takiej opcji, to należy zdefiniować symbol o nazwie `BUFFER_OPTIMIZATION`. Normalnie jego definicja w pliku *FT_Platform.h* powinna być zakomentowana.

Dzięki temu możemy dostosować bibliotekę do własnych potrzeb i użytego układu. Dobra wiadomość jest taka, że to już wszystkie elementy, które wymagają naszej uwagi i ewentualnie zmian. Reszta plików biblioteki jest niezmienna i zasadniczo nie musimy do nich zaglądać. Warto jednak wiedzieć, co znajdziemy w poszczególnych plikach.

Plik *FT_Hal_Utils.h* zawiera kilka makrodefinicji umożliwiających między innymi konwersję koloru z formatu RGB z podanymi składowymi na liczbę.

Istotnym plikiem jest plik *FT_Gpu.h*, który zawiera definicje rejestrów i poleceń wykonywanych przez akcelerator graficzny. Nazwy rejestrów rozpoczynają się od prefiksu `REG_`, po którym znajduje się nazwa, dzięki czemu zamiast numerów możemy posługiwać się wygodnymi nazwami opisowymi. Z kolei nazwy poleceń zaczynają się od prefiksu `CMD_`. Trochę inaczej jest w przypadku poleceń znajdujących się na DL — wymagają one szeregu argumentów, które muszą być odpowiednio zakodowane w 32-bitowym ciągu zawierającym kod polecenia i argumenty. Aby to było możliwe, zostały one zdefiniowane jako makrodefinicje, na przykład:

```
#define VERTEX2F(x,y) ((1UL<<30)|(((x)&32767UL)<<15)|(((y)&32767UL)<<0))
```

Pokazana przykładowa makrodefinicja określa funkcję umożliwiającą wybór punktu o współrzędnych (*x*, *y*). Dodatkowo w tym nagłówku zdefiniowana jest struktura `FT_Gpu_Fonts`, określająca format definicji zestawu czcionek. Jakkolwiek definicje zawarte w tym pliku umożliwiają w pełni wykorzystanie możliwości akceleratorów serii *FT8xx*, to nie zawsze stosowanie ich jest wygodne. Dlatego w pliku *FT_CoPro_Cmds.h* umieszczone zostały prototypy funkcji umożliwiających skorzystanie z poleceń koprocesora realizujących bardziej złożone operacje, na przykład odpowiadające za rysowanie przycisków, pasków przewijania, odtwarzanie multimediów itd. Definicje zawartych tu funkcji znajdują się w pliku *FT_CoPro_Cmds.c* — pozwalają one wstawić wymagane polecenia i ich argumenty do kolejki poleceń koprocesora graficznego. Z funkcji znajdujących się w tych plikach będziemy bardzo często korzystać, warto więc się z nimi zaznajomić.

Ostatnie dwa pliki to *FT_Gpu_Hal.h* zawierający prototypy i *FT_Gpu_Hal.c* zawierający definicje podstawowych funkcji i struktur danych umożliwiających przesyłanie informacji między MCU a GPU. Tworzą one tak zwaną warstwę abstrakcji od sprzętu (ang. *Hardware Abstraction Layer*), czyli zestaw funkcji niezależnych od wykorzystywanej platformy sprzętowej, dzięki czemu bez względu na to, jaki stosujemy interfejs sprzętowy i mikrokontroler, reszta biblioteki posługuje się takim samym, ujednoliconym interfejsem programistycznym (API — ang. *Application Programming Interface*). Zawarte tu funkcje wykorzystują do komunikacji funkcje, które zdefiniowaliśmy w pliku *FT_SPI.h*, dodając do nich nowe funkcjonalności: możliwość transferu wielobajtowych ciągów danych z pamięci RAM lub FLASH, kolejkowanie i buforowanie poleceń oraz wysyłanie poleceń i danych do układu akceleratora. Ta część biblioteki dostarczona przez FTDI cechuje się największym przerostem kodu i stopniem „poplątania", przez co jest stosunkowo mało czytelna. Wykorzystywane są w niej pewne struktury danych, które zostały zdefiniowane na wyrost, prawdopodobnie na potrzeby przyszłych, złożonych systemów zawierających akceleratory graficzne, w tym systemów zawierających jednocześnie więcej niż jeden akcelerator. W praktyce z takich możliwości nigdy nie będziemy korzystać, a całą funkcjonalność zawartą w tych plikach można by znacznie uprościć, lecz nie ma to większego sensu. Ewentualne zyski w postaci zmniejszenia objętości generowanego kodu nie rekompensują czasu, jaki

poświęcimy na jej modyfikację. Ponieważ do tej części biblioteki raczej nigdy nie będziemy zaglądać, czytelność kodu też nie jest dla nas priorytetem.

Z grubsza poznaliśmy strukturę biblioteki obsługi układów z rodziny *FT8xx*. Jak widzimy, udostępnia ona jednolite API dla aplikacji wykorzystujących te akceleratory, unifikując je dla różnych modeli wchodzących w skład tej rodziny. Nie tylko ułatwia to pisanie własnych programów, ale także umożliwia łatwą współpracę z pokazanymi w kolejnych rozdziałach programami do projektowania ekranów graficznych i GUI. Programy te mają możliwość generowania kodu odpowiedzialnego za rysowanie zaprojektowanego ekranu — kod taki możemy bezpośrednio włączyć do programu wykorzystującego opisaną bibliotekę, co ułatwia i przyśpiesza pisanie oprogramowania wykorzystującego akceleratory z serii *FT8xx*.

Mniej więcej znamy bibliotekę i jej strukturę, musimy jeszcze poznać podstawowe funkcje w niej zawarte i skonfigurować akcelerator tak, aby mógł przyjmować nasze polecenia i, co ważniejsze, wyświetlić pierwszy obraz.

Pierwszy start

Pora więc na uruchomienie naszego sprzętu. W zależności od tego, jakiego używamy modułu, musimy zajrzeć do pliku *FT_Platform.h* i skonfigurować go zgodnie z posiadanym typem układu, rozdzielczością LCD i typem wykorzystywanego interfejsu.

> Jeśli korzystasz z gotowych modułów integrujących matrycę LCD z akceleratorem, pamiętaj, aby przed pierwszym podłączeniem dokładnie przeczytać instrukcję producenta, w szczególności jej fragmenty dotyczące ustawienia napięcia zasilania modułu. Nieprawidłowy wybór napięcia zasilającego może trwale uszkodzić moduł.

Aby rozpocząć normalną pracę z akceleratorem, musimy przejść kilka etapów konfiguracji:

- skonfigurować wykorzystywane piny IO mikrokontrolera oraz interfejs komunikacyjny MCU;
- zresetować i wybudzić akcelerator, opcjonalnie warto też zidentyfikować, z jakim mamy do czynienia typem akceleratora, oraz sprawdzić poprawność działania interfejsu wymiany danych;
- przesłać dane konfiguracyjne użytej matrycy LCD — jej rozdzielczość taktowanie, położenie impulsów synchronizacji, głębię kolorów;
- opcjonalnie skonfigurować część akceleratora odpowiedzialną za obsługę panelu dotykowego;
- skonfigurować lub wyłączyć obsługę audio — nie jest to krytyczny krok, ale brak konfiguracji lub niewłaściwa konfiguracja może objawiać się słyszalnym szumem z głośnika (o ile wykorzystywany moduł jest w niego wyposażony);
- skonfigurować zegar taktujący akcelerator i włączyć sterowanie matrycą LCD;
- załadować pierwszą DL, co spowoduje rozpoczęcie wyświetlania obrazu.

Musimy zatem wykonać kilka kroków, przy czym aby to było możliwe, potrzebujemy informacji dotyczących typu, rozdzielczości, taktowania i synchronizacji posiadanej matrycy LCD. Wartości te mogą różnić się pomiędzy matrycami i zawsze należy je sprawdzić w nocie producenta. Niektóre wartości można przyjąć z pewnym prawdopodobieństwem za standardowe i jeśli nie dysponujemy danymi dotyczącymi wykorzystywanej matrycy, możemy użyć tych wartości domyślnych. Zazwyczaj umożliwiają one poprawne działanie układu. Stąd też w naszym kodzie zostało zawartych kilka domyślnych konfiguracji dla matryc o różnej rozdzielczości. Zacznijmy więc konfigurację akceleratora.

Na początek pamiętajmy o przejrzeniu plików *FT_SPI.h*, *FT_Platform.h* i *FT_DataTypes.h* pod kątem używanej przez nas konfiguracji. Jeśli korzystasz z innego układu niż *FT800*/*FT801* lub matrycy innej niż matryca o rozdzielczości 640 na 480 punktów, zajdzie potrzeba dostosowania tych plików do posiadanej konfiguracji. W naszych przykładach przyjmujemy także, że do komunikacji wykorzystujemy interfejs SPI. Warto przejrzeć definicje używanych pinów w pliku *FT_SPI.h* i ewentualnie je dostosować do własnej konfiguracji sprzętowej.

Teraz nadeszła pora, aby zajrzeć do pliku *startup_FTEVE_HAL.c*. Zawiera on nasz pierwszy przykładowy program oraz funkcje odpowiedzialne za konfigurację akceleratora, a przy okazji na jego przykładzie prześledzimy wszystkie etapy konfiguracji akceleratora.

Zacznijmy od funkcji `main()`. W pierwszym etapie musimy zainicjalizować interfejs sprzętowy, za pomocą którego będziemy się komunikować z akceleratorem, oraz sam sterownik (`HAL`). W tym celu wywołamy funkcję biblioteczną `Ft_Gpu_Hal_Init`, która przyjmuje argument o typie `Ft_Gpu_HalInit_t` (nie będziemy się nim zbytnio interesować — w naszych przykładach niektóre możliwości konfiguracyjne nigdy nie będą wykorzystywane). Funkcja ta jest zdefiniowana w pliku *FT_Gpu_Hal.c* i… nic nie robi. Została wyłącznie ze względu na zachowanie kompatybilności z API sterownika, równie dobrze więc możemy ją usunąć. Jej wywołanie zostało dla utrzymania pewnej konwencji kodu. Funkcją, która odpowiada za inicjalizację sprzętu, jest `Ft_Gpu_Hal_Open`. Jej zadaniem jest otwarcie portu komunikacyjnego i skonfigurowanie wykorzystywanych pinów IO. Jako argument przyjmuje ona strukturę o typie `Ft_Gpu_Hal_Context_t`, przesadnie skomplikowaną; na nasze potrzeby wykorzystywać będziemy wyłącznie pole `host.hal_config.spi_clockrate_khz`, które zawiera informację na temat sugerowanej szybkości taktowania interfejsu SPI lub I2C. Pole to jest o tyle istotne, że jak pamiętamy, przed konfiguracją zegarów taktujących akcelerator taktowanie SPI nie może przekroczyć 10 MHz, a później możemy je zwiększyć do 30 MHz. W praktyce interesuje nas wyłącznie, czy wybrane taktowanie ma być niższe czy wyższe niż 10 MHz. Zmienna `host` i wskaźnik na nią `phost` są wykorzystywane przez inne funkcje API, stąd w naszym projekcie musi ona występować. Niemniej w przypadku, gdy korzystamy wyłącznie z jednego akceleratora, znowu jest to pewien przerost, lecz nie będziemy się tym przejmować. Wywołana funkcja `Ft_Gpu_Hal_Open` konfiguruje piny IO poprzez wywołanie funkcji inicjalizującej `SPI_init()`, zależnej od sprzętu, którą zdefiniowaliśmy w pliku *FT_SPI.h*. Oprócz tego ustawia ona zegar taktujący interfejs komunikacyjny i kilka dodatkowych pól zmiennej `host`. Od tego momentu możemy przesyłać informacje pomiędzy MCU a GPU, a interfejs został skonfigurowany tak, aby nie przekroczyć maksymalnej częstotliwości równej 10 MHz. Mogąc już wymieniać dane z GPU, możemy przystąpić do jego konfiguracji.

W tym celu wywołujemy funkcję `Ft_BootupConfig()` — chociaż nie zawiera ona zależnego od sprzętu kodu, z jakichś powodów twórcy biblioteki postanowili, żeby znalazła się ona poza nią, i pozostawili jej definicję użytkownikowi. Najpewniej jest to związane z tym, że zawiera ona pewne elementy wymagające uwagi programisty — między innymi elementy odpowiedzialne za konfigurację parametrów związanych z użytą matrycą LCD. Ale po kolei. Jak pamiętamy, pierwszym etapem konfiguracji GPU jest jego zresetowanie, co zapewnia przywrócenie nam ustawień domyślnych i wybudzenie układu z uśpienia. Osiągamy to, wywołując funkcję biblioteczną `Ft_Gpu_Hal_Powercycle`. Żongluje ona sygnałem *PD_N*, inicjalizując w ten sposób GPU. **Musimy pamiętać, że ze względu na specyfikację układu jej wykonanie może trwać 40 – 60 ms!** A więc relatywnie długo. Jeśli jest to dla nas nieakceptowalne, musimy napisać własną wersję tej funkcji, przeplatając inicjalizację GPU innym kodem. W praktyce miałoby to znaczenie w sytuacji, w której wykorzystujemy jakiś system operacyjny, na przykład RTOS. Kolejny fragment kodu sprawdza poprawność komunikacji:

```
chipid = Ft_Gpu_Hal_Rd8(phost, REG_ID);
while(chipid != 0x7C)
    {
        chipid = Ft_Gpu_Hal_Rd8(phost, REG_ID);
        Ft_Gpu_Hal_Sleep(100);
    }
```

Układy z rodziny *FT8xx* mają rejestr o nazwie `REG_ID`, którego odczyt niezależnie od posiadanego układu zwraca wartość *0x7C*. Opcjonalnie możemy jeszcze odczytać komórkę pamięci o adresie *0xC0001*. Jej zawartość pozwala zidentyfikować typ akceleratora graficznego. I tak wartość *0x10* to układ *FT810*, *0x12* to *FT812*, *0x11* to *FT811*, a *0x13* to *FT813*. Dla układów *FT800* i *FT801* nie można przeprowadzić identyfikacji na podstawie zawartości tej komórki pamięci, lecz można je odróżnić po zachowaniu innych rejestrów, między innymi rejestrów kontroli panelu rezystancyjnego i pojemnościowego.

Od tego momentu mamy już poprawnie zainicjalizowane podstawowe podsystemy GPU, czas więc na jego konfigurację. Zaczynamy od konfiguracji podłączonej matrycy LCD. Jest to jeden z najważniejszych etapów, a błędne parametry mogą spowodować nieprawidłowe wyświetlanie obrazu na LCD lub jego brak.

> Konfigurując tę część GPU, należy odnieść się do noty katalogowej matrycy LCD i odnaleźć parametry związane z jej rozdzielczością i położeniem impulsów synchronizacji. Konfigurację matrycy trzeba przeprowadzić zawsze przed wysłaniem polecenia włączenia obrazu. W przeciwnym razie możemy doprowadzić do jej uszkodzenia, a przynajmniej skracamy jej żywotność.

Zanim przejdziemy do konfiguracji, warto jeszcze raz wrócić do rozdziału 11., paragrafu „Sygnały sterujące matrycą”, w którym opisane są sygnały i parametry matryc LCD. Tu mamy identyczny problem: uniwersalny kontroler mogący współpracować z wieloma różnymi matrycami, co implikuje potrzebę konfiguracji pewnych parametrów. Cały blok konfigurujący tę część GPU wygląda następująco:

```
Ft_Gpu_Hal_Wr16(phost, REG_HCYCLE, FT_DispHCycle);
Ft_Gpu_Hal_Wr16(phost, REG_HOFFSET, FT_DispHOffset);
Ft_Gpu_Hal_Wr16(phost, REG_HSYNC0, FT_DispHSync0);
```

```
Ft_Gpu_Hal_Wr16(phost, REG_HSYNC1, FT_DispHSync1);
Ft_Gpu_Hal_Wr16(phost, REG_VCYCLE, FT_DispVCycle);
Ft_Gpu_Hal_Wr16(phost, REG_VOFFSET, FT_DispVOffset);
Ft_Gpu_Hal_Wr16(phost, REG_VSYNC0, FT_DispVSync0);
Ft_Gpu_Hal_Wr16(phost, REG_VSYNC1, FT_DispVSync1);
Ft_Gpu_Hal_Wr8(phost, REG_SWIZZLE, FT_DispSwizzle);
Ft_Gpu_Hal_Wr8(phost, REG_PCLK_POL, FT_DispPCLKPol);
Ft_Gpu_Hal_Wr16(phost, REG_HSIZE, FT_DispWidth);
Ft_Gpu_Hal_Wr16(phost, REG_VSIZE, FT_DispHeight);
Ft_Gpu_Hal_Wr16(phost, REG_CSPREAD, FT_DispCSpread);
Ft_Gpu_Hal_Wr16(phost, REG_DITHER, FT_DispDither);
```

Jak widzimy, do poszczególnych rejestrów wpisujemy parametry, które w naszej bibliotece są dostępne jako zestaw predefiniowanych stałych, zależnych od rozdzielczości wybranej matrycy. Nazwy poszczególnych rejestrów dokładnie odpowiadają parametrom matrycy, jakie znajdziemy w nocie, co więcej, w kodzie biblioteki mamy predefiniowanych kilka zestawów parametrów umożliwiających zazwyczaj poprawną pracę z różnymi matrycami — jedyne, co musimy zrobić, to określić ich rozdzielczość. Efektem powyższego kodu jest konfiguracja sygnałów sterujących matrycą oraz położenia impulsów synchronizacji.

Kolejnym elementem jest konfiguracja wyjść GPIO akceleratora:

```
Ft_Gpu_Hal_Wr8(phost, REG_GPIO_DIR,0x80);
Ft_Gpu_Hal_Wr8(phost, REG_GPIO,0x80);
```

Rejestr `REG_GPIO_DIR` określa, czy wybrany pin jest wejściem (stosowny bit ma wartość 0) lub wyjściem (odpowiadający bit ma wartość 1). Z kolei rejestr `REG_GPIO` określa stan wyjść, umożliwia także odczytanie stanu wejść. Warto pamiętać, że w rejestrze `REG_GPIO` tylko bity *0*, *1* i *7* określają wartości wyjść, pozostałe określają prąd sterujący pinami (tabela 13.4).

Tabela 13.4. *Znaczenie bitów rejestru sterującego GPIO*

	Bity [6:5]				Bit 4		Bity [3:2]			
Wartość	0b00	0b01	0b10	0b11	0b0	0b1	0b00	0b01	0b10	0b11
Prąd	4 mA	8 mA	12 mA	16 mA	4 mA	8 mA	4 mA	8 mA	12 mA	16 mA
Pin	*GPIO0* i *GPIO1*				Wszystkie pozostałe piny		*MISO*, *INT_N*			

Jak widzimy, możemy zmieniać prąd sterujący, dostosowując go do wymogów matrycy LCD oraz układów, które łączymy z akceleratorem. W praktyce zazwyczaj możemy wpisywać wartość 0 na tych pozycjach, co daje minimalny prąd na poziomie 4 mA. Warto pamiętać, że bit *7* tego rejestru steruje sygnałem matrycy o nazwie *DISP*. Stąd w czasie normalnej pracy powinien być wyjściem w stanie wysokim (bit *7* powinien mieć wartość 1). Nadanie mu wartości 0 powoduje wyłączenie matrycy LCD poprzez dezaktywację sygnału *DISP*. Z kolei bit *1* określa stan pinu *GPIO1*, zwykle połączonego z wejściem *SHDN* wzmacniacza audio, dlatego za pomocą tego pinu możemy go włączać lub wyłączać.

Kolejne dwie instrukcje ładują domyślną DL — rozkazy w niej zawarte powodują wyczyszczenie ekranu:

```
Ft_Gpu_Hal_WrMemFromFlash(phost, RAM_DL,(ft_uint8_t *) FT_DLCODE_BOOTUP,
➥sizeof(FT_DLCODE_BOOTUP));
Ft_Gpu_Hal_Wr8(phost, REG_DLSWAP,DLSWAP_FRAME);
```

Dzięki temu po włączeniu matrycy nie będziemy widzieć przypadkowych efektów związanych z losową wartością pamięci DL. Domyślna lista zdefiniowana jest w programie w tablicy `FT_DLCODE_BOOTUP`. Możemy ją dowolnie zmieniać.

Na koniec pozostaje nam włączenie matrycy LCD, dzięki czemu zobaczymy efekt działania wcześniej załadowanej DL:

```
Ft_Gpu_Hal_Wr8(phost, REG_PCLK,FT_DispPCLK);
```

Teraz możemy zwiększyć taktowanie interfejsu SPI, poprzez który wymieniamy dane z akceleratorem, i rozpocząć normalną pracę.

Tworzenie własnej listy

Skoro akcelerator jest skonfigurowany i rozpoczął wyświetlanie obrazu, czas poinstruować go, co ma wyświetlać. W tym celu musimy wysłać mu instrukcje służące do tworzenia poszczególnych elementów obrazu. Jak wiemy, znajdują się one w specjalnym obszarze pamięci akceleratora — liście wyświetlania. Polecenia znajdujące się na tej liście umożliwiają rysowanie podstawowych elementów graficznych, przez co utworzenie nawet relatywnie prostego obrazu wymaga całego ciągu poleceń. Możemy sobie nieco ułatwić pracę, korzystając z pomocy koprocesora graficznego. Jak już wiemy, przyjmuje on wszelkie polecenia, jakie mogą znaleźć się na DL, ale także polecenia bardziej złożone (na przykład wyświetlenie napisów), które zostaną przetłumaczone na polecenia dla DL. Dlatego szczególnie na początku łatwiej nam będzie się posługiwać koprocesorem. Koprocesor ma własną pamięć o pojemności 4 kB, która stanowi bufor dla jego poleceń. Koprocesor pobiera sobie z tego bufora polecenie, tłumaczy je i zapisuje wyniki na DL. Całość ułatwiają nam dostępne w naszej bibliotece funkcje. Pierwszym poleceniem, jakie musimy wydać, jest polecenie utworzenia nowej DL:

```
Ft_Gpu_CoCmd_Dlstart(phost);
```

Wysyła ono krótkie polecenie koprocesora, `CMD_DLSTART`, powodujące utworzenie nowej listy. Od tego momentu możemy wysyłać kolejne polecenia dla koprocesora. Zacznijmy od polecenia przywracającego domyślne parametry wyświetlania:

```
Ft_App_WrCoCmd_Buffer(phost,CLEAR(1, 1, 1));
```

Powinno ono rozpoczynać każdą DL, tak aby stan akceleratora odpowiadał wartościom domyślnym. Po nim możemy wysłać polecenie wypełnienia ekranu określonym kolorem:

```
Ft_App_WrCoCmd_Buffer(phost,COLOR_RGB(231, 239, 75));
```

Kolor kodowany jest w postaci trzech składowych, które za pomocą makra COLOR_RGB zamieniane są na element polecenia. Teraz możemy wyświetlić napis:

```
Ft_Gpu_CoCmd_Text(phost,16, 131, 28, 0, "Nacisnij kolejne punkty w celu kalibracji
↪panela");
```

Pierwsze dwa parametry to współrzędne *x* i *y* wyświetlanego tekstu, kolejny parametr określa użytą czcionkę z zakresu 0 – 31, następnie mamy parametry wyjustowania i wreszcie sam tekst. Na koniec możemy wywołać jeszcze jedno polecenie, umożliwiające kalibrację panelu dotykowego:

```
Ft_Gpu_CoCmd_Calibrate(phost, 0);
```

Jest to polecenie wbudowane w akcelerator, które pozwala skalibrować panel poprzez naciśnięcie go w kolejno pojawiających się trzech punktach. Tym poleceniem zajmiemy się jeszcze później. Każdą DL musi kończyć polecenie umożliwiające rozpoczęcie wyświetlania obrazu:

```
Ft_App_WrCoCmd_Buffer(phost, DISPLAY());
```

Mając już kompletną listę poleceń, należy przesłać polecenie dokonujące podmiany bieżącej DL na listę, którą przesłaliśmy:

```
Ft_Gpu_CoCmd_Swap(phost);
```

Na koniec jeszcze zastanówmy się, co robi polecenie Ft_App_WrCoCmd_Buffer i inne, które wykorzystaliśmy? Funkcje biblioteczne mogą bezpośrednio przesłać nowe komendy do koprocesora albo możemy skorzystać z opcji ich zbuforowania w pamięci RAM MCU, aby potem je przesłać naraz do akceleratora graficznego. Ta druga opcja wymaga oczywiście utworzenia odpowiednich buforów w pamięci RAM, co nie zawsze jest potrzebne. Jak pamiętamy, to, czy zostaną one utworzone, kontroluje stała BUFFER_OPTIMIZATION zdefiniowana w pliku *FT_Platform.h*. Jeśli korzystamy z bufora, musimy po zakończeniu tworzenia DL przesłać jego zawartość do akceleratora, co zapewnia polecenie:

```
Ft_App_Flush_Co_Buffer(phost);
```

Jeżeli nie korzystamy z bufora, wówczas polecenie to nic nie robi, nie ma jednak przeszkód, aby je wydać — dzięki temu nasz kod będzie działał poprawnie również w sytuacji, kiedy włączymy buforowanie. Ponieważ koprocesor graficzny potrzebuje chwilkę na przetworzenie naszych poleceń i utworzenie nowej DL, możemy wstrzymać wykonywanie programu do momentu zakończenia przetwarzania poleceń:

```
Ft_Gpu_Hal_WaitCmdfifo_empty(phost);
```

Takie wstrzymanie jest istotne, tylko jeśli chcielibyśmy od razu przesłać kolejne polecenia do akceleratora, w przeciwnym razie możemy swobodnie kontynuować wykonywanie programu.

Jak widzimy, tworzenie DL jest bajecznie proste, o ile oczywiście zna się polecenia akceleratora i koprocesora. Czas więc je poznać. Ponieważ jest ich niezwykle dużo (wystarczy przejrzeć dostępne pliki nagłówkowe, aby się przekonać, ile mamy ich do dyspozycji), nie będziemy ich po kolei szczegółowo opisywać. Z pewnością byłaby to

najnudniejsza część książki. Zamiast tego poznamy je w praktyce, za pomocą intuicyjnych programów, które umożliwią nam szybkie utworzenie DL — w tym celu posłużymy się programami narzędziowymi udostępnianymi za darmo przez FTDI, a opisanymi w kolejnym rozdziale.

Oczywiście pokazane proste operacje wykorzystujące koprocesor nie wyczerpują wszystkich możliwości wymiany danych z akceleratorem. W kolejnych rozdziałach omówię bardziej złożone funkcje i sposoby odwoływania się do pamięci akceleratora.

Skorowidz

F

G

I

K

L

M

N

O

P

R

S

Ś

T

U

W

Z

Naucz się obsługiwać grafikę na wyświetlaczach kolorowych LCD!

— Poznaj działanie kontrolerów kolorowych LCD
— Odkryj sposoby wykorzystania wyświetlaczy w swoich projektach
— Naucz się tworzyć grafikę na kolorowe LCD
— Dowiedz się, jak skutecznie optymalizować swoje programy

Mikrokontrolery zawojowały świat elektroniki użytkowej, dzięki czemu można je dziś znaleźć niemal w każdym zaawansowanym technicznie sprzęcie domowym czy rozrywkowym. Natomiast rosnąca dostępność i spadek cen kolorowych wyświetlaczy ciekłokrystalicznych sprawiły, że kolejne urządzenia zaczęto wyposażać w coraz lepsze ekrany. Poprawie ich jakości i rozdzielczości oraz wzrostowi wydajności układów sterujących towarzyszy zaś stałe ulepszanie interfejsów użytkownika, które są coraz wygodniejsze i atrakcyjne graficznie.

Jeśli dostrzegasz konieczność opracowywania lepszych interfejsów graficznych dla swoich projektów, chcesz pełnymi garściami korzystać z możliwości oferowanych przez nowoczesne mikrokontrolery oraz wyświetlacze lub po prostu interesuje Cię ten temat i pragniesz poszerzyć swoje umiejętności projektowania atrakcyjnych i użytecznych rozwiązań, sięgnij po książkę *Mikrokontrolery AVR i ARM. Sterowanie wyświetlaczami LCD*! Wprowadzi Cię ona w świat kontrolerów LCD i nauczy technik programistycznych, dzięki którym będziesz w stanie zaprojektować, opracować i zoptymalizować GUI w taki sposób, aby chciało się go używać i aby było to czystą przyjemnością!

— Wprowadzenie do środowiska AVR i ARM oraz konfiguracja warsztatu pracy
— Informacje na temat dostępnych na rynku wyświetlaczy LCD
— Działanie kontrolerów i konfiguracja odpowiednich interfejsów
— Funkcje wyświetlaczy i podstawowe operacje graficzne
— Formaty plików i konwersja danych graficznych
— Optymalizacja działania kodu odpowiedzialnego za grafikę
— Obsługa paneli dotykowych różnych typów
— Wykorzystanie możliwości akceleratorów graficznych

Atrakcyjny interfejs użytkownika w Twoim projekcie? **Nie ma problemu!**

Helion
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

ISBN 978-83-283-2846-4
9 788328 328464

Informatyka w najlepszym wydaniu
cena: 89,00 zł