Akcelerator do kolorowych wyświetlaczy graficznych z ILI9163C

Każdy, kto próbował używać kolorowego wyświetlacza graficznego z ArduinoUNO/Mega/Nano albo mikrokontrolerem AVR czy PIC wie, że szybkość wyświetlania obrazu pozostawia wiele do życzenia. Alternatywą są wyświetlacze z akceleratorem graficznym firmy FTDI albo NEXTION. Niestety wyświetlacze takie do tanich nie należą, a i filozofia obsługi jest inna niż wyświetlacza z klasycznym sterownikiem. Ponadto akceleratory FT8xx, ze względu na stosunkowo wysoką cenę, są stosowane w wyświetlaczach o większej rozdzielczości. Czy dla wyświetlacza o małej lub średniej rozdzielczości trzeba sięgać po wydajniejsze mikrokontrolery jak na przykład ARM? Na szczęście nie jest to już konieczne, ponieważ tani akcelerator przeznaczony dla mikrokontrolerów 8-bit zapewni płynną animację na wyświetlaczach o małej rozdzielczości.

W dużym skrócie, animacje na Arduino wyglądają tak: *https://youtu.be/yhmGWwZsM7w,* a powinny tak: *https://youtu.be/Smb9O3ywjIo.* Opisany akcelerator daje taką możliwość, a nawet dużo więcej: *https://youtu.be/HsY9-6lA6Cc.*

W EdW jest publikowany cykl artykułów dotyczący wyświetlaczy graficznych. Znalazło się tam słuszne stwierdzenie, że podłączenie kolorowego wyświetlacza do Arduino opartego na AVR nie ma sensu.

I tak było do dziś. Dzięki opisywanemu akceleratorowi graficznemu można przyspieszyć operacje na wyświetlaczu od 2 do 120 razy (średnio 10...20 razy). Bardzo istotną zaletą akceleratora jest to, że operacje graficzne przeprowadzane są na buforze, który później, jedną komendą, jest wysyłany w ciągu 30ms do wyświetlacza. Dzięki temu na ekranie nie widać przebiegu operacji graficznych, nawet gdy są wykonywane powoli. W czasie wysyłania danych do wyświetlacza mikrokontroler może wykonywać inne operacje, akcelerator pełni wtedy funkcję podobną do DMA. Aktualnie oprogramowanie akceleratora obsługuje kolorowy wyświetlacz 128×128 ze sterownikiem ILI9163C, ale można je dostosować do innych typów wyświetlaczy. W tej sprawie proszę o e-maile do redakcji EdW tak samo jak i w przypadku zainteresowania akceleratorem dla wyświetlaczy o średniej (320×240 – ILI9325/ILI9341) i większej (480×320 – ILI9486) rozdzielczości.

Opis układu

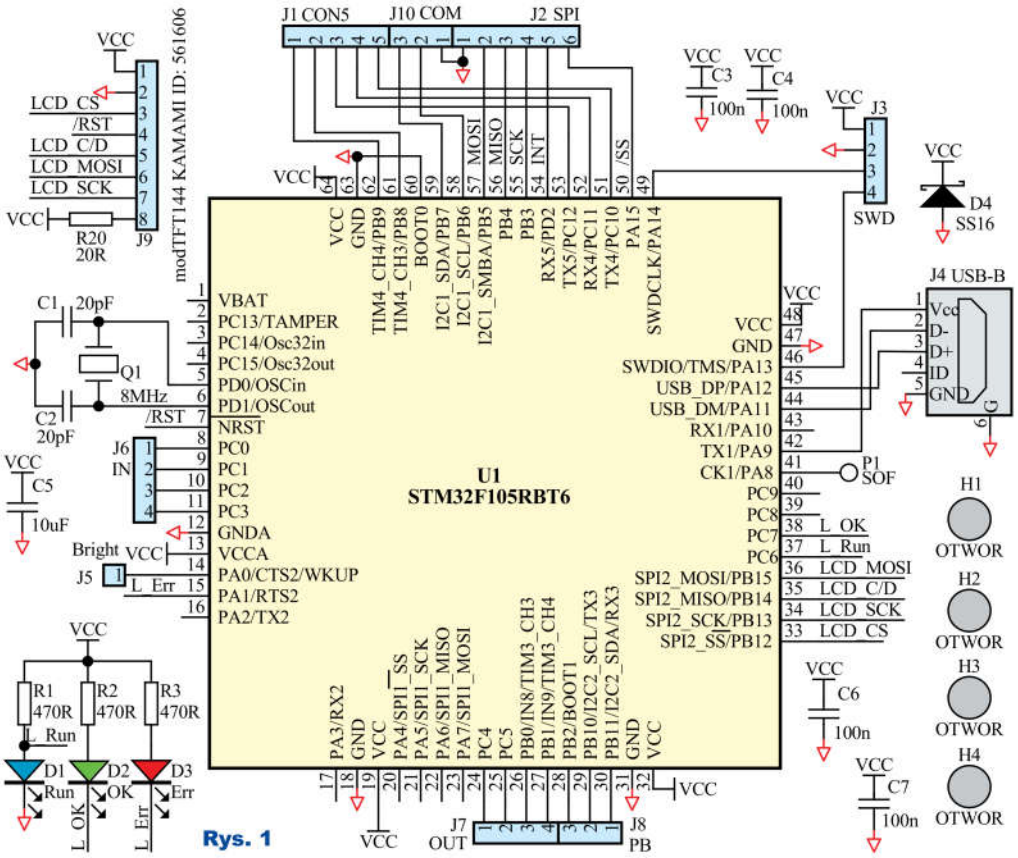
Schemat urządzenia pokazano na **rysunku 1**. Układ zasilany jest ze złącza J3. D4 zabezpiecza układ przed skutkami odwrotnej biegunowości zasilania. Komendy i dane graficzne dostarczone złączem J2 trafiają do akceleratora, którego funkcję pełni stosunkowo tani mikrokontroler U1 STM32F105RBT6 zawierający w swej strukturze, poza 128kB FLASH, 64kB RAM oraz między innymi DMA, które odbierają dane z Arduino oraz wysyłają do wyświetlacza podłączonego do J9. Diody LED sygnalizują stan pracy urządzenia. Kwarc Q1 oraz kondensatory C1 i C2 nie są używane w aktualnej wersji oprogramowania. J6 i J7 to wejścia i wyjścia cyfrowe do dowolnego wykorzystania. Wyprowadzenia te są dostępne za pośrednictwem komend akceleratora, podobnie jak sygnał PWM dostępny na J5. J1, J8, J10 tak jak i J4 służą do testowania oprogramowania i nie są potrzebne podczas normalnego użytkowania. Przyspieszenie i płynną animację uzyskano dzięki kilku zabiegom: m.in. dane o kolorach są 8- zamiast 16-bitowe, co dwukrotnie przyspiesza określanie stanu pikseli, jednak ogranicza paletę barw do 256 (paleta

Charakterystyka:
- Zasilanie 2,4...3,6V.
- Akceptacja napięć do 5V na liniach wejściowych.
- Przyspieszenie operacji graficznych od 2 do 120 razy; średnio 10...20 razy.
- Obsługa wyświetlaczy o małej (do 128×128) rozdzielczości.
- Sprzętowe operacje czyszczenia ekranu, wypełniania prostokątów, rysowania linii i czcionek.
- Brak ograniczenia liczby obiektów, jak w FT8xx (maksymalnie 2000).
- Wsparcie dla Sprite's, znanych z komputerów Amiga, C-64, Atari z przezroczystym tłem.
- Dowolna liczba Sprites i dowolne ich wymiary.
- Cztery wejścia cyfrowe.
- Pięć wyjść cyfrowych, w tym jedno PWM.

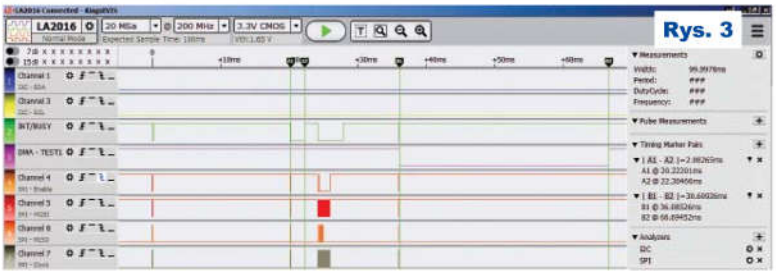
eWydanie dla: Sławomir Skrzyński (713)
Wydanie elektroniczne przeznaczone wyłącznie do użytku własnego bez prawa do rozpowszechniania.

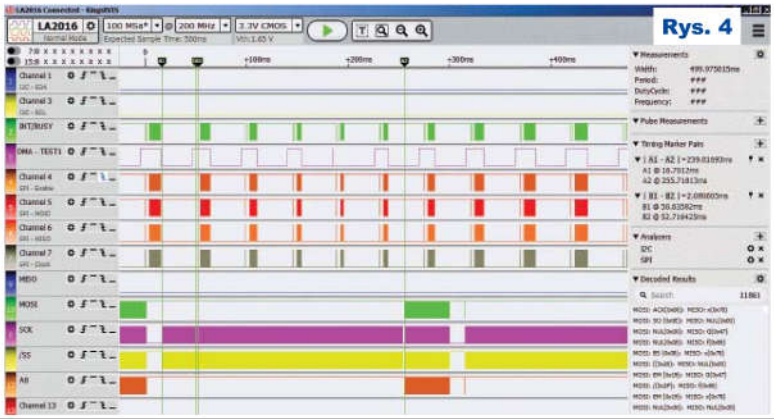
218 „bezpiecznych” barw w Internecie + 16 kolorów).

Akcelerator zawiera bufor na komendy/dane graficzne wielości 16386+512 bajtów. Pozwala to zapamiętać jednorazowo (pomiędzy aktywnym a nieaktywnym strobem /SS) dane o wszystkich pikselach ekranu o rozdzielczości 128×128 w 256 kolorach. Poza danymi można wysyłać komendy. Akcelerator interpretuje komendy i konwertuje dane kolorów z 8- na 16-bitowe (tabela barw znajduje się w pliku „paleta_barw.c”), wypełniając bufor przeznaczony dla wyświetlacza. Gdy dane ekranu są gotowe, można je wysłać do wyświetlacza. Transmisja 32kB danych (rozdzielczość 128×128 w 65 536 kolorach) trwa niecałe 31ms i jest realizowana przez DMA. **Rysunek 2** przedstawia dane z analizatora logicznego przyłączonego do akceleratora (***Uwaga! Oryginalne zrzuty ekranu o dużej rozdzielczości są dostępne w Elportalu***). Markery A1 i A2 obejmują transmisję 16 384 danych do bufora ekranu, natomiast B1, B2 transmisję danych 32 768 do LCD po transkodowaniu z 8 na 16-bit. **Rysunek 3** przedstawia przykładową sekwencję komend realizujących wypełnienie całego bufora kolorem (czyszczenie ekranu) oraz wysłania bufora do wyświetlacza. Markery A1 i A2 to wysłanie komendy wypełnienia (8 bajtów) oraz czekanie na gotowość BUSY/INT przed wysłaniem danych do wyświetlacza – komenda DISPLAY. Na **rysunku 4** przebiegi w kanałach 0, 2, 4...7 analizatora ukazują czas działania komendy czyszczenia ekranu wydanej do akcelerometru (markery A1 i A2), która trwa 2ms, w stosunku do tej samej operacji wykonanej przez Arduino UNO (AVR mega328), która trwa 239ms, co zaznaczono markerami B1 i B2 (kanały 8,10...12 analizatora). Ze względu na to, że porównanie zaawansowanych operacji graficznych nie ma sensu, na **rysunku 5** pokazane są czasy animacji sześcianu *(https://youtu.be/K2y4R1oa5pY)* wykonane przez UNO z akceleratorem i bez. Bez akceleratora jedna klatka animacji jest generowana przez 283ms (markery A1 i A2), przy czym sama animacja zajmuje 44ms (markery B1, B2), resztę czasu (239ms) zajmuje czyszczenie ekranu (markery A1, B1). Ta sama animacja, jeszcze bez sprzętowego rysowania linii przez akcelerator, zajmuje od 6,6ms (markery D1, D2) do 13ms (markety C1, C2) zależnie od długości linii. Po każdej klatce następuje pauza 31ms (poziom niski, kanał nr 0), w czasie której DMA wysyła dane do wyświetlacza. Wyraźnie widać, że akcelerator umożliwił wysłanie siedem razy więcej ramek w tym samym czasie i to w sytuacji, gdy nie wykorzystano sprzętowej możliwości rysowania linii. W sytuacji, gdy wykorzystuje się rysowanie sprzętowe, pojedyncza klatka animacji jest generowana w czasie poniżej



Rys. 1





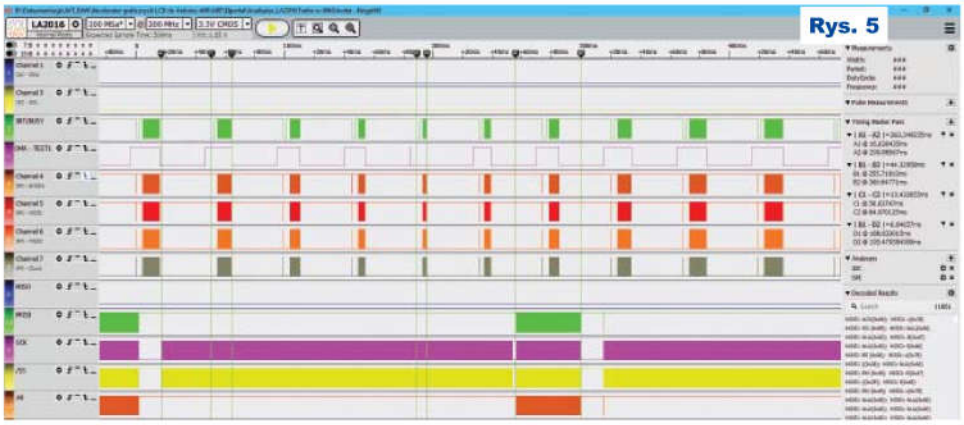
eWydanie dla: Sławomir Skrzyński (713)
Wydanie elektroniczne przeznaczone wyłącznie do użytku własnego bez prawa do rozpowszechniania.

6ms i to bez względu na długość linii, co widać na **rysunku 6**. W stosunku do 283ms **daje to ponad 47-krotne przyspieszenie!** Zaawansowana animacja *(https://youtu.be/HsY9-6lA6Cc)* generowana jest przez AVR mega328 w czasie 13ms! Czas ten oznaczony jest markerami (A1, A2) na **rysunku 7**. Po wygenerowaniu animacji CPU czeka na wykonanie operacji przez akcelerator (markery B1, B2) przez 50ms. Czas ten nie musi być tracony na czekanie na dezaktywację sygnału BUSY/INT. Wykorzystując mechanizm przerwań, CPU może realizować inne zadania. Maksymalna częstotliwość odświeżania animacji nie przekracza 15FPS, co wynika ze wzoru: 1/(0,013+0,05), gdzie 0,013 i 0,050 to czas generowania animacji (13ms) i wykonywania operacji przez akcelerator. Czas 50ms jest zbyt długi, aby nie było widać rysowania linii, a co za tym idzie migotania ekranu. Artefakty nie są widoczne tylko dlatego, że operacje graficzne przeprowadzane są na buforze, a później wysyłane komendą DISPLAY. W programie należy zadbać, aby komenda ta była wykonywana tylko raz po wygenerowaniu całej animacji. Przykłady znajdują się w katalogu „examples", w materiałach dodatkowych w Elportalu.

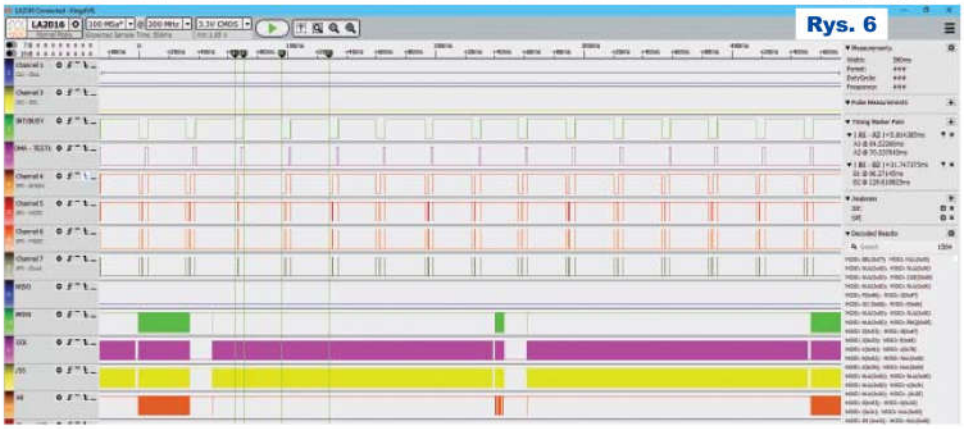
Dane z rejestratora znajdują się w materiałach dodatkowych na Elportalu. Po zainstalowaniu bezpłatnego programu można je dokładnie obejrzeć.

Realizacja programowa. Do komunikacji z wyświetlaczem można wykorzystać bibliotekę napisaną dla Arduino opartego na AVR (Uno/Mega/Nano i podobne). Biblioteka, wraz z przykładami, jest dostępna w materiałach dodatkowych na Elportalu. Na wstępie opiszę komendy akcelerometru, ale przedtem budowę pojedynczej ramki danych. Ramka danych rozpoczyna się od zmiany stanu linii /SS na niski. Pierwszy bajt danych zawiera następujące informacje:

Bit 7	Kierunek transmisji: 0-zapis, 1-odczyt (w akcelerometrze zawsze 0).
Bit 6	Typ następnej transmisji: 0-następna ramka dane, 1-komenda.
Bit 5	Rozmiar transmisji: 0-8 bit, 1-16 bit (w akcelerometrze zawsze 0).
Bity 4...0	Kod komendy



Rys. 5



Rys. 6

Rys. 7

Kolejne bajty (3 lub 7) mogą zawierać komendy lub dane. Komendy **zawsze** dopełniane są do czterech bajtów (mod. 4).

Ramkę kończy zmiana stanu linii /SS na wysoki. Przed wysłaniem kolejnej ramki trzeba czekać na nieaktywny sygnał BUSY/INT. W jednej ramce można wysłać wiele komend. Przykładowa komenda czyszczenia ekranu: 0x06, 0x00, 0x00, 0x00

(Komenda postawienia piksela o współrzędnych 2, 3 o kolorze numer 5:
0x09, 0x02, 0x03, 0x05

Ciąg komend czyszczenia ekranu i postawienia dwu punktów na współrzędnych 4,5 i 4,6 w kolorze numer 3:
0x46, 0x00, 0x00, 0x00
0x49, 0x04, 0x05, 0x03,
0x09, 0x04, 0x06, 0x03

Warto zwrócić uwagę, że gdy wysyłany jest ciąg komend, bit 6 komendy jest ustawiony. W **tabeli 1** przedstawiony jest skrócony spis komend. Szczegóły opisane są w pliku, dostępnym w Elportalu wśród materiałów dodatkowych do tego numeru.

Odczyt danych z akceleratora: Z akceleratora można odczytać infor-

Prenumerujesz Elektronikę dla Wszystkich?
Zaloguj się na swoje konto Prenumeratora (www.avt.pl/uzytkownik)
i pobierz ZA DARMO e-wydania kilkudziesięciu czasopism z serii #CzasNaCzytanie
Prenumeratę zamówisz na www.avt.pl/prenumerata

eWydanie dla: Sławomir Skrzyński (713)
Wydanie elektroniczne przeznaczone wyłącznie do użytku własnego bez prawa do rozpowszechniania.

mację o bieżącym adresie, z którego dane zostały wysłane do wyświetlacza. Można więc, w czasie wysyłania danych, operować na buforze ekranu pod warunkiem, że jest to obszar, który już został wysłany do wyświetlacza. Z akceleratora można też odczytać informacje o jego konfiguracji, wersji oprogramowania, itd. Pierwsze dwa bajty zawierają przypadkowe dane, a kolejne według **tabeli 2**.

Większość komend została zebrana w bibliotece „EdWgfx.cpp". Dzięki temu użycie akceleratora jest bardzo proste. Demonstrację funkcji można znaleźć w przykładach. Najwięcej efektów zawiera plik „cube.ino" w katalogu „examples/cube". Do podstawowej obsługi wyświetlacza wystarczy kilka funkcji:

void EdWgfx(uint8_t csPin, uint8_t busyPin); definiuje wyprowadzenia, które starują linią / SS akcelerometru oraz pin, do którego przyłączono sygnał „BUSY/INT".

void begin();
na razie nie realizuje żadnej funkcji, ale z uwagi na ewentualne przyszłe potrzeby dla porządku należy ją wywołać.

void fillScreen(uint8_t kolor);
wypełnia cały ekran (bufor) jedną barwą, inaczej, „czyści" ekran.

void drawPixel(uint8_t x, uint8_t y, uint8_t color);
umieszcza punkt o brawie „color" na ekranie w miejscu o współrzędnych x, y.

void display();
wysyła bufor z danymi ekranu do wyświetlacza. W pierwszej kolejności program będzie czekał, aż sygnał BUSY/INT przyjmie poziom nieaktywny. Gdy akcelerator będzie gotowy, zostanie wysłana komenda DISPLAY. Kolejne operacje na buforze ekranu można przeprowadzić nie wcześniej niż po 31ms od wykonania komendy display(), w przeciwnym wypadku mogą pojawić się zakłócenia na ekranie.

Powyższe komendy wystarczą do obsługi wyświetlacza, ale znaczne przyspieszenie uzyska się, używając sprzętowego rysowania linii, wypełniania obszarów, itd. Poniżej ważniejsze z komend:

void fillRect(uint8_t tx, uint8_t ty, uint8_t tw, uint8_t h, uint8_t color);
wypełnia prostokątny obszar kolorem.

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint8_t color);
rysuje linię pomiędzy dwoma punktami.

void drawSprite(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1, uint8_t * dat, uint16_t len);
rysuje „duszka". Kolor nr 0 jest przezroczysty. Dane „duszka" znajdują się w pamięci RAM.

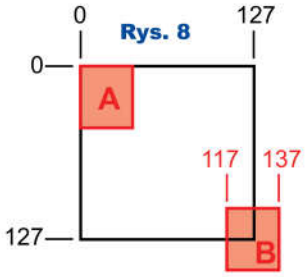
void drawSprite_P(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1, uint8_t * dat, uint16_t len);
Działa jak „drawSprite" z tym, że dane znajdują się w pamięci FLASH dla mikrokontrolerów do 128kB pamięci.

void drawSprite_PF(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1, uint8_t * dat, uint16_t len);
działa jak „ drawSprite_P" z tym, że pozwala zaadresować do 1MB pamięci.

ofs(uint8_t ofsX, uint8_t ofsY);
ofset pozwala na przesunięcie lewego górnego rogu ekranu do wartości ujemnych. Tę funkcję pozwolę sobie opisać bardziej szczegółowo. Na **rysunku 8** pokazano dwa obiekty o wielkości 20x20 punktów. Widoczna część ekranu (okno wyświetlania) jest zaznaczona niebieską ramką. Obiekt B nie jest widoczny w całości, ponieważ

Spis komend:	
NOP	Nic nie robi, używane do odczytu informacji z akceleratora lub wysyłania danych (pikseli)
CLS	Czyści ekran
DISPLAY	Wysyła dane z bufora do wyświetlacza
PIXEL	Stawia piksel w miejscu X, Y ekranu, w wybranym kolorze
CURSOR	Ustawia kursor
COLOR	Ustawia kolor pióra i tła. Zakres danych 0...255
OUT	Ustawia stan wyjść (bity 0...3). Zakres danych 0...15
WINDOW	Zdefiniowanie okna, w którym będą umieszczane dane. X0, Y0 - lewy górny róg okna, X1, Y1 – prawy dolny róg okna. Zakres wartości danych 0...127
SPRITE	Działa jak komenda WINDOW z tym, że dana o wartości 0 jest traktowana jako przezroczysta barwa. Zakres danych 0...127
OFS	Ofset współrzędnych x, y umożliwia „chowanie" obiektów na górze i z lewej strony ekranu. Wartość w zakresie 0...63
PWM	Ustawia wyjście PWM. Zakres danych 0..255
STRING	Umieszcza tekst w miejscu X, Y, napisany fontem (0...3) z atrybutami (0..2) w kolorze PEN. Gdy włączony atrybut TFT_MODE_TXT_BACK, czcionka zostanie umieszczona na tle o kolorze BACK. Łańcuch znaków musi być zakończony kodem 0, dane muszą być dopełnione do czterech bajtów
FILL	Wypełnia prostokąt kolorem. Lewy górny róg X, Y. Szerokość W, wysokość H
LINE_H	Rysuje linię poziomą
LINE_V	Rysuje linię pionową
LINE	Rysuje linię pomiędzy dwoma punktami

Tabela 1



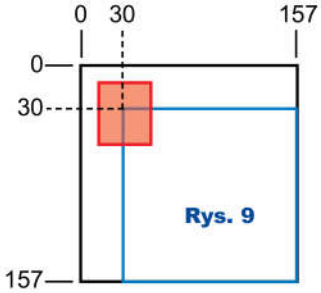
Rys. 8

Bajt	Komentarz
0, 1	tarszy/młodszy bajt słowa adresu bufora ekranu, który został wysłany do wyświetlacza. Wartość 0x8000 sygnalizuje koniec transmisji danych
2...7	ciąg znaków ASCII nazwy urządzenia
8...11	ciąg znaków ASCII wersji oprogramowania
12, 13	młodszy/ starszy bajt typu wyświetlacza (dla ILI9163C wartość 0x0001)
14, 15	młodszy/ starszy bajt konfiguracji wyświetlacza (nieużywany)
16, 17	młodszy/ starszy bajt rozdzielczości poziomej
18, 19	młodszy/ starszy bajt rozdzielczości pionowej
20, 21	młodszy/ starszy bajt startowej pozycji poziomej okna
22, 23	młodszy/ starszy bajt startowej pozycji pionowej okna
24, 25	młodszy/ starszy bajt końcowej pozycji poziomej okna
26, 27	młodszy/ starszy bajt końcowej pozycji pionowej okna
28, 29	młodszy/ starszy bajt poziomego położenia kursora
30, 31	młodszy/ starszy bajt pionowego położenia kursora
32, 33	młodszy/ starszy bajt koloru pióra
34, 35	młodszy/ starszy bajt koloru tła
36	wartość PWM
37	stan portu wyjściowego (bity 0...3) i wejściowego (bity 4...7)

Tabela 2

eWydanie dla: Sławomir Skrzyński (713)
Wydanie elektroniczne przeznaczone wyłącznie do użytku własnego bez prawa do rozpowszechniania.

jego lewy górny róg został umieszczony w punkcie o współrzędnych 117/117. Nie ma więc problemu, aby cały obiekt „schował się" w niewidocznej części ekranu, wystarczy umieścić go w punkcie 128. Dzięki temu łatwo można zrealizować funkcję „wychodzenia" obiektu zza ekranu. W przypadku obiektu A nie ma możliwości, aby schował się on z lewej strony ekranu lub na górze, ponieważ funkcje umieszczające punkty obiekty przyjmują tylko wartości dodatnie. Na **rysunku 9** przedstawiono sytuację po wywołaniu funkcji „ofs(30, 30);". W takim przypadku widoczna część ekranu ma współrzędne 30/30, a nie 0/0. Umieszczając obiekt w punkcie 10,10 (przypominam, że ma wymiary 20×20) spowodujemy, że nie będzie on widoczny. Gdy zostanie umieszczony w punkcie 20/20 widoczna będzie tylko jego dolna prawa ćwiartka. Ze względu na to, że maksymalna wartość przesunięcia wynosi 63 (–63) nie da się schować obiektów większych niż 63×63 punkty, ale warto pamiętać, że taki obiekt miałby szerokość i/lub wysokość połowy ekranu. Trzeba też pamiętać, że przesuwając punkt 0/0 w lewo i/lub w górę, o tyle samo zmniejszamy niewidoczną część ekranu z prawej/dolnej strony.



Rys. 9

void printString(char *txt);
umieszcza tekst na ekranie.
void setCursor(uint8_t x, uint8_t y);
umieszcza wirtualny kursor na ekranie, dzięki czemu można mieć wpływ na miejsce, w którym zostanie wyświetlony tekst funkcją „ printString". Ustawianie kursora dotyczy także pozostałych funkcji operujących na bloku danych; jak na przykład komenda WINDOWS.
void setTextColor(uint8_t color);
void setTextBack(uint8_t color);
komendy ustalają kolor pióra i tła tekstu. Kolor tła ma znaczenie tylko wtedy, gdy atrybut TFT_MODE_TXT_BACK jest włączony.
void setTextSize(uint8_t rozmiar);
wybór czcionki dla tekstu. Dostępne opcje:
0 – 5x7 proporcjonalna PL + dodatkowe symbole
1 – 16x16 proporcjonalna/ nieproporcjonalna (maszynowa), tylko podstawowe znaki ASCII, bez PL
2 – 32x32 proporcjonalna/ nieproporcjonalna (maszynowa), tylko podstawowe znaki ASCII, bez PL
3 – 10x7 proporcjonalna PL + dodatkowe symbole

Nic nie stoi na przeszkodzie, aby dowolne czcionki umieszczać, używając funkcji „drawPixel", których wzór znajduje się w pamięci mikrokontrolera.
void setTextAtrybut(uint8_t atrybuty);
ustawia atrybuty tekstu. Dostępne opcje:
- TFT_MODE_PROP – włącza czcionkę proporcjonalną
- TFT_MODE_TXT_BACK – włącza czcionkę z tłem ustalonym funkcją „setTextBack".

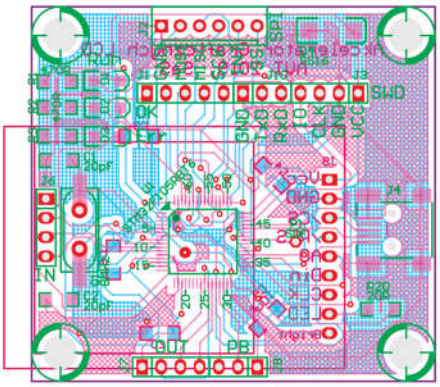
Atrybuty można sumować, na przykład „setTextAtrybut(TFT_MODE_PROP + TFT_MODE_TXT_BACK)" wybierze czcionkę proporcjonalną z tłem.

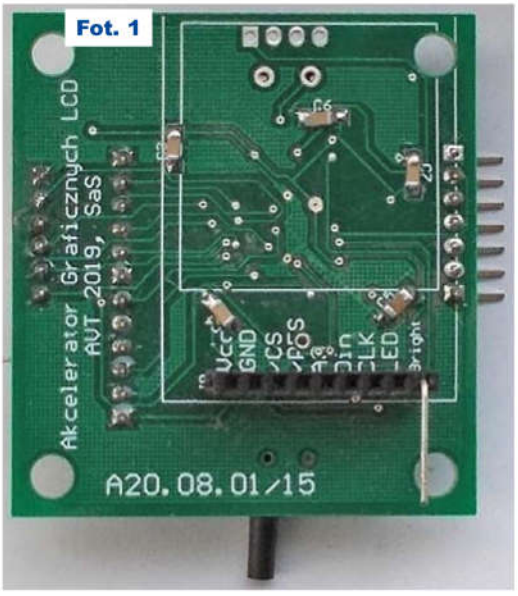
Na koniec kilka funkcji niezwiązanych bezpośrednio z operacjami na treści obrazu:
void setContinue(uint8_t cont);
standardowo po wykonaniu komendy program oczekuje na dezaktywację BUSY/INT. Funkcja „setContinue(true);" wyłączy to oczekiwanie i komendy będą wysyłane jedna za drugą (będą buforowane w akceleratorze). Uwaga! Po wykonaniu niektórych komend trzeba czekać na gotowość sygnału BUSY/INT (drawSprite, printString). W takiej sytuacji funkcja CONTINUE zostanie

Wykaz elementów

Rezystory 1206:	
R20	20Ω
R1 R2 R3	470Ω
Kondensatory ceramiczne:	
C1 C2	20pF, nie montować
C3 C4 C6 C7	100nF
C5	10uF
Półprzewodniki:	
U1	STM32F105RBT6
D1	Dioda LED Niebieska 1206
D2	Dioda LED Zielona 1206
D3	Dioda LED Czerwona 1206
D4	SS16
Pozostałe:	
Q1	Kwarc 8MHz, nie montować
	nie montować
J1 J8 J10 P1	Punkty kontrolne, nie montować
J2	NS25-W6P
J3 J6 J7	NS25-W4P
J4	MUSB-B5-S-RA-SMT/ AVT Kod: USB B MINI - nie montować
J5	Punkt lutowniczy
J9	Gniazdo goldpin 8x1
Wyświetlacz modTFT144 KAMAMI ID: 561606	

Rys. 10





Fot. 1

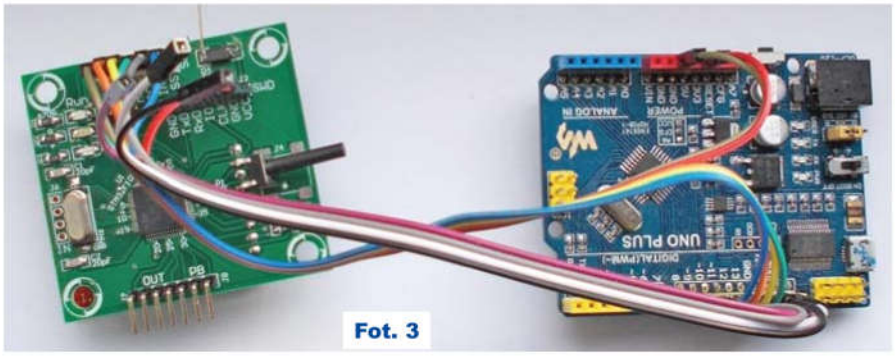
Fot. 2

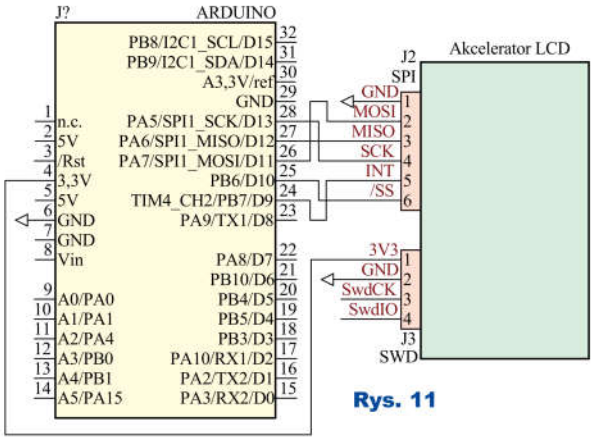
eWydanie dla: Sławomir Skrzyński (713)
Wydanie elektroniczne przeznaczone wyłącznie do użytku własnego bez prawa do rozpowszechniania.

wyłączona. Aby z niej korzystać, należy ją ponownie włączyć.
void setOUT(uint8_t r);
ustawia wyjścia (bity 0...3).
uint8_t readOUT();
odczytuje stan wyjść ustawionych funkcją „setOUT„. Wynik na bitach 0...3.
uint8_t readIN();
ddczytuje stan wejść. Wynik na bitach 0...3.
void setPWM(uint8_t r);
ustawia wypełnienie przebiegu PWM. Zakres 0...255.
uint8_t readPWM();
odczytuje wartość PWM ustawioną funkcją „setPWM„. Wynik na bitach 0...3.
uint16_t readAdrDMA();
zwraca adres w buforze, na którym operowało DMA. Zakres wartości 0...32768.

Pozostałe komendy są rzadziej używane. Osoby zainteresowane ich wykorzystaniem powinny zapoznać się z plikiem „EdWgfx.cpp”.

Efekt działania akceleratora można zobaczyć w materiałach dodatkowych na Elportalu oraz na kanale YouTube https://www.youtube.com/channel/UCQ3i4KfWue2P4r8PTMODyHw.


Fot. 3



Rys. 11

Montaż i uruchomienie

Układ można zmontować na płytce drukowanej, której projekt pokazany jest na **rysunku 10**.
Standardowo montujemy układ, zaczynając od elementów najmniejszych, a kończąc na największych.

Fotografia wstępna oraz **fotografie 1 i 2** pokazują model. Układ nie wymaga uruchomienia. Zmontowany prawidłowo ze sprawnych elementów powinien od razu pracować. Osoby niedoświadczone powinny poprosić kogoś o pomoc w zaprogramowaniu procesora. Biblioteki dla Arduino instaluje się w standardowy sposób. Schemat podłączenia akceleratora z ArduinoUNO pokazano na **rysunku 11.** Przy podłączeniu pomocna może być **fotografia 3**. Łączenie linii MISO nie jest konieczne, jeśli nie będą odczytywane informacje z akceleratora. Niestety, w ten sposób nie uzyska się dodatkowego pinu portu mikrokontrolera, bo w przeciwieństwie do STM32, w AVR nie można odłączyć nieużywanych wyprowadzeń układu peryferyjnego, w tym przypadku linii wejściowej MISO.

Epilog: Im dłużej pracuję z Arduino, tym więcej ograniczeń AVR i bibliotek Arduino zauważam. Rzeczy banalne dla ARM są trudne do zrobienia albo niemożliwe dla AVR. W przypadku wyświetlaczy graficznych zmuszony byłem sięgnąć do rozwiązań z lat osiemdziesiątych! Ówczesne mikroprocesory osiągały typowo ok. 1MIPS, maksymalnie 3...4MIPS (pomijam MC680x0 i inne procesory 16/32-bitowe), więc „dopalacze” były niezbędne, ale AVR ma 20 MIPS przy 20MHz, przynajmniej tak je reklamuje producent. W praktyce AVR osiąga ok. 15MIPS ze względu na to, że wiele rozkazów wykonuje się w dwóch cyklach zegarowych. Chociaż 15MIPS to nie 4, niemniej nadal w przypadku grafiki wymagane są „dopalacze” ☹.

W przypadku Xmega jest lepiej, bo mają DMA, ale cena nie zachęca. ARM będzie lepszy (duża ilość RAM, architektura 32-bit) i przeważnie tańszy.

Warto też zasygnalizować, że biblioteki dla Arduino, w przeważającej większości, są pisane niedbale. Nagminne jest nadużywanie liczb zmiennoprzecinkowych. Bardzo często używany jest typ „int” tam, gdzie wystarczyłby typ ośmiobitowy. Typ „int” może mieć sens dla CPU 32-bit, ponieważ wykonuje się szybciej niż 8-bit, ale w takiej sytuacji należy użyć „uint_fast8_t”, który będzie dobrany optymalnie do użytego CPU, ponadto programista analizujący kod wie, że ten fragment kodu jest optymalizowany pod kątem szybkości działania. Klasa „string” nadużywana w mikrokontrolerach z 2kB RAM może doprowadzić do zawieszania programu. Dynamiczna alokacja dużych obiektów w RAM, zwłaszcza gdy jest jej mało, też nie jest dobrym pomysłem. Wszystko to powoduje, że programy działają wolno i konieczna jest optymalizacja bibliotek, która nie zawsze jest łatwa, czasem po prostu lepiej napisać bibliotekę od nowa.

SaS
sas@elportal.pl

R E K L A M A


eWydanie dla: Sławomir Skrzyński (713)
Wydanie elektroniczne przeznaczone wyłącznie do użytku własnego bez prawa do rozpowszechniania.