



Tomasz Francuz

# AVR

## UKŁADY PERYFERYJNE

POZNAJ PRAKTYCZNE ZASTOSOWANIA URZĄDZEŃ PERYFERYJNYCH  
DLA MIKROKONTROLERA AVR!

- Dowiedz się, jak działają zewnętrzne pamięci półprzewodnikowe
- Naucz się korzystać z komparatorów oraz przetworników ADC i DAC
- Poznaj techniki wyświetlania obrazu i odtwarzania dźwięku

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Michał Mrowiec

Projekt okładki: Studio Gravite / Olsztyn  
Obarek, Pokoński, Pazdrijowski, Zaprucki

Fotografia na okładce została wykorzystana za zgodą Shutterstock.com

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto: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/avrukp>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:

<ftp://ftp.helion.pl/przyklady/avrukp.zip>

ISBN: 978-83-246-9225-5

Copyright © Helion 2014

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

<b>Wstęp</b> .....	<b>9</b>
Sprzęt .....	10
Przykłady .....	11
Skróty jednostek .....	12
<b>Rozdział 20. Zwalniamy, czyli kiedy opóźnienia są konieczne</b> .....	<b>13</b>
Opóźnienia i XMEGA .....	18
Wykorzystanie timerów do realizacji opóźnień .....	20
<b>Rozdział 21. Łączenie kodu C i asemblera</b> .....	<b>23</b>
ABI .....	25
Słowo kluczowe asm .....	27
Typy operandów .....	29
Dostęp do portów IO .....	30
Dostęp do danych wielobajtowych .....	31
Dostęp do wskaźników .....	32
Lista modyfikowanych rejestrów .....	32
Wielokrotne użycie wstawki asemblerowej .....	33
Pliki .S .....	34
Wykorzystanie rejestrów w asemblerze .....	34
Dyrektywy asemblera .....	40
Wywołanie funkcji języka C z asemblera .....	41
<b>Rozdział 22. Wyświetlacze LCD alfanumeryczne</b> .....	<b>43</b>
Obsługa wyświetlaczy alfanumerycznych .....	45
Funkcje biblioteczne .....	50
Definiowanie własnych znaków .....	56
Transakcyjna obsługa LCD .....	58
Optymalizacja .....	68
<b>Rozdział 23. Interfejs SPI i USART SPI</b> .....	<b>71</b>
Tryby pracy SPI .....	74
Konfiguracja interfejsu SPI .....	75
Konfiguracja USART w trybie SPI .....	76
Pamięci DataFLASH .....	77
Organizacja pamięci .....	77
Interfejs SPI pamięci .....	79
Dostęp do pamięci .....	81

Polecenia .....	81
Rejestr identyfikacyjny pamięci .....	87
Polecenia odczytu pamięci .....	88
Polecenia odczytu i zapisu bufora .....	89
Polecenia transferu pomiędzy pamięcią a buforem .....	91
Rejestry specjalne .....	94
Transakcyjny dostęp do SPI .....	95
Łączymy DMA ze SPI .....	96
Wywołania zwrotne (callbacks) .....	98
Transakcje .....	103
Zasilanie .....	109
<b>Rozdział 24. Budujemy system plików .....</b>	<b>111</b>
Najprostszy system plików .....	111
Inicjalizacja pamięci .....	119
System plików FAT/FAT32 .....	128
Obsługa FAT .....	131
Prototypy z diskio.h .....	136
Demonstracja .....	139
Biblioteka PetitFS .....	144
Konfiguracja PetitFS .....	149
<b>Rozdział 25. Jeszcze więcej pamięci, czyli karty SD, SDHC i spółka .....</b>	<b>151</b>
Trochę o budowie karty .....	152
Tryby pracy karty .....	153
Zasilanie karty .....	154
Komunikacja .....	156
Tryb pracy SPI .....	157
Format protokołu .....	157
Różnice pomiędzy kartami .....	160
Inicjalizacja karty .....	161
Rejestry specjalne karty .....	169
CID .....	169
CSD .....	172
Operacje zapisu i odczytu sektorów .....	173
Operacje odczytu .....	173
Operacje zapisu .....	176
Suma kontrolna .....	178
Program .....	178
Potencjalne problemy .....	179
FATFS na karcie .....	180
<b>Rozdział 26. Debugger .....</b>	<b>183</b>
Konfiguracja debugera .....	185
Konfiguracja projektu do debugowania .....	186
Debugger sprzętowy .....	187
Debugger programowy (symulator) .....	190
Plik stymulacji .....	191
Pułapki .....	197
Punkty śledzenia .....	203
Pułapki warunkowe .....	205
Podgląd pamięci .....	207
Podgląd stosu wywołań .....	209
Określenie czasu symulacji .....	209
Okno dezasemblera .....	210

Okno podglądu zmiennych .....	211
Łańcuchy formatujące .....	212
Makrodefinicja ASSERT .....	214
Przerwania w trakcie debugowania .....	218
_delay_xx i symulator .....	219
<b>Rozdział 27. Przetwornik analogowo-cyfrowy .....</b>	<b>221</b>
Przetwornik potokowy vs. cykliczny .....	222
Napięcie referencyjne .....	223
Konfiguracja pinu IO .....	225
Multiplexer wejściowy .....	226
Układ wzmacniania sygnału .....	227
Co to jest LSB? .....	228
Kalibracja ADC .....	229
Pomiar .....	231
Rozdzielczość przetwornika .....	231
Tryby pracy przetwornika .....	231
Wynik pomiaru .....	236
Kalibracja offsetu .....	237
Pomiar napięcia zasilania i temperatury mikrokontrolera .....	239
Redukcja poboru energii .....	242
Preskaler ADC .....	243
Wyzwalanie konwersji z wykorzystaniem systemu zdarzeń .....	245
Rejestr EVCTRL w XMEGA z ADC bez potoku .....	248
Rejestr porównania .....	249
Termometr LM35 .....	250
Budujemy termometr z alarmem .....	251
Tryb ciągłej konwersji .....	252
Przemiatanie wejść .....	253
Przerwania .....	254
Wykorzystanie DMA do transferu wyników .....	254
Nadpróbkowanie .....	258
Uśrednianie .....	259
Decymacja i interpolacja .....	259
Interpolacja i decymacja w XMEGA .....	260
Jak zwiększyć precyzję pomiarów? .....	260
Budujemy datalogger .....	261
Termistory jako mierniki temperatury .....	262
Program dataloggera .....	265
<b>Rozdział 28. Komparator analogowy .....</b>	<b>279</b>
Komparator — trochę teorii .....	279
Czas propagacji .....	281
Histereza .....	281
Komparatory analogowe XMEGA .....	282
Multiplexery wejścia .....	283
Komparator okienkowy .....	284
Przerwania .....	286
Uruchomienie komparatora .....	287
Rejestr stanu komparatora .....	287
Komparator jako oscylator .....	287
Termostat z wykorzystaniem komparatorów .....	290
Termistory .....	290

<b>Rozdział 29. DAC .....</b>	<b>293</b>
Buforowanie wyjścia .....	294
Napięcie referencyjne .....	295
Taktowanie .....	296
Zdarzenia .....	297
Wykorzystanie DMA .....	298
Próbkowanie 8-bitowe .....	300
Tryb dwukanałowy .....	301
Generowanie jednocześnie dwóch przebiegów .....	301
Wersja oszczędna .....	304
Inne sposoby wyzwalania konwersji .....	305
Tryb oszczędzania energii .....	306
Kalibracja DAC .....	306
<b>Rozdział 30. Monochromatyczne wyświetlacze graficzne .....</b>	<b>309</b>
Podłączenie LCD do mikrokontrolera .....	311
Budowa i funkcje kontrolera ST7565R .....	313
Funkcje specjalne kontrolera .....	319
Inwersja i testowanie obrazu .....	319
Obracanie obrazu .....	319
Regulacja kontrastu .....	321
Numer pierwszej wyświetlanej linii .....	321
Czcionki .....	322
Mała optymalizacja .....	330
Podwójne buforowanie .....	332
Adres początku wyświetlania obrazu .....	333
Menu .....	335
Menu oparte na piktogramach .....	342
<b>Rozdział 31. Pliki z danymi — jak je dodawać do projektu? .....</b>	<b>349</b>
Kompilacja plików binarnych .....	350
Łączenie plików obiektowych z projektem .....	352
Dostęp do danych binarnych .....	356
Klasyczny sposób dostępu do danych .....	358
Dostęp do danych z wykorzystaniem przestrzeni adresowych .....	359
<b>Rozdział 32. Magistrala pamięci zewnętrznej .....</b>	<b>361</b>
Podłączenie pamięci .....	362
Konfiguracja portów IO .....	362
Przyporządkowanie sygnałów interfejsu EBI do portów IO .....	363
Konfiguracja 4-portowa .....	364
Pamięć SRAM .....	366
Pamięć SRAM w trybie LPC .....	371
Pamięć SDRAM .....	372
Konfiguracja sygnału CS .....	377
Określenie adresu bazowego i wielkości pamięci .....	377
Układ sterowania sygnałem wyboru w trybie SRAM .....	379
Układ sterowania sygnałem wyboru w trybie SDRAM .....	379
Przykładowa konfiguracja pamięci SDRAM .....	381
Konfiguracja zegara .....	383
Dostęp do pamięci z poziomu języka C .....	384
Dostęp do pamięci poniżej granicy 64 kB .....	384
Dostęp do pamięci powyżej granicy 64 kB .....	386

<b>Rozdział 33. Generowanie obrazu wideo .....</b>	<b>391</b>
Generowanie obrazu wideo w standardzie VGA .....	393
Wtyczka VGA i konwersja sygnałów .....	396
Monochromatyczny tryb tekstowy VGA .....	397
Monochromatyczny tryb graficzny VGA .....	407
Generowanie sygnału composite .....	409
Standard PAL .....	410
Monochromatyczny tryb tekstowy .....	413
Monochromatyczny tryb graficzny .....	416
Tworzenie nakładek (OSD) .....	417
Czas na kolor .....	424
Kodowanie koloru .....	425
Konwerter cyfrowo-analogowy .....	426
Generator obrazu composite .....	426
Wykorzystanie EuroSCART .....	428
Kolorowy obraz na TV .....	430
<b>Rozdział 34. Niech zagra muzyka .....</b>	<b>439</b>
Formaty plików dźwiękowych .....	440
Częstotliwość próbkowania .....	441
Format pliku .....	442
Obróbka dźwięku .....	443
Program Audacity .....	443
Program SoX .....	445
Wzmacniacz .....	446
Odtwarzamy muzykę z wykorzystaniem DAC .....	447
Inny sposób na podwójne buforowanie .....	457
Generowanie dźwięku z wykorzystaniem PWM .....	461
PWM — trochę teorii .....	461
Filtrowanie sygnału PWM .....	468
Odtwarzamy dźwięk za pomocą 8-bitowego PWM .....	472
Rozszerzenie HiRes i PWM o większej rozdzielczości .....	477
Kompresja dźwięku .....	480
Próbkowanie nieliniowe .....	480
Kompresja ADPCM .....	482
Kompresja IMA ADPCM .....	482
Nagrywanie mowy z wykorzystaniem kompresji ADPCM .....	486
Algorytm ADPCM firmy Dialogic .....	492
<b>Rozdział 35. A może mp3? .....</b>	<b>495</b>
Koprocesor mp3 .....	496
Sposoby podłączenia do XMEGA .....	496
Dostęp do rejestrów układu .....	499
Format pliku wav .....	500
Rejestry GPIOR .....	502
Testy układu .....	502
Podstawowa komunikacja z koprocesorem .....	504
Rejestry układu VS1003B .....	508
Rejestr trybu pracy .....	508
Rejestr stanu układu .....	510
Rejestr kontroli basów .....	510
Rejestr SCI_CLOCKF .....	511
Rejestr czasu utworu .....	512
Rejestr formatu audio .....	512

Rejestry dostępu do pamięci RAM .....	513
Rejestr adresu wtyczki .....	513
Rejestr kontroli głośności .....	513
Odtwarzamy muzykę .....	513
Odtwarzamy muzykę z wykorzystaniem DMA .....	518
Magnetofon cyfrowy .....	528
VS100XX w roli magnetofonu .....	529
Własne wtyczki .....	535
DTMF jako przykład własnej wtyczki .....	535
Budujemy własną wtyczkę .....	541
Tworzenie tablicy z kodem wynikowym .....	543
Budowa wtyczki .....	544
Własna aplikacja, czyli dekodery DTMF .....	548
<b>Rozdział 36. Fusebity i lockbity .....</b>	<b>557</b>
Fusebity .....	558
Fusebit JTAGEN .....	558
Fusebit RSTDISBL .....	559
Fusebit BOOTRST .....	559
Fusebity SUT .....	559
Fusebit TOSCSEL .....	559
Układ detekcji awarii zasilania .....	559
Watchdog .....	560
Fusebit EESAVE .....	561
Lockbity .....	561
Sygnatura produkcyjna procesora .....	563
Numer serii .....	563
Numer wafra .....	563
Położenie na wafrze .....	564
Pozostałe bajty konfiguracyjne .....	564
Sygnatura użytkownika .....	565
Dostęp do danych z poziomu aplikacji użytkownika .....	568
Bajty kalibracyjne .....	569
Konfiguracja fuse- i lockbitów w AVR-libc .....	570
Lockbity w AVR-libc .....	570
Fusebity w AVR-libc .....	571
<b>Dodatek A Spis rozdziałów książki „AVR. Praktyczne projekty” .....</b>	<b>573</b>
<b>Skorowidz .....</b>	<b>575</b>



## Rozdział 22.

# Wyświetlacze LCD alfanumeryczne

### Z tego rozdziału dowiesz się...

- ♦ jak podłączyć popularne wyświetlacze alfanumeryczne do XMEGA;
- ♦ jak stworzyć bibliotekę obsługującą kontroler HD44780;
- ♦ jak przysyłać polecenia dla LCD w formie transakcji i jak łączyć dostęp do kontrolera z aplikacji i funkcji obsługi przerwań.

Ze względu na malejące ceny, zwiększającą się dostępność oraz niski pobór energii coraz większym zainteresowaniem cieszą się wyświetlacze LCD. W tym rozdziale zostaną omówione wyświetlacze alfanumeryczne — ze uwagi na ich prostotę są ciągle najczęściej stosowanymi typami wyświetlaczy. W dalszej części książki zostaną omówione monochromatyczne wyświetlacze graficzne — dają one o wiele większe możliwości, ale ich sterowanie jest nieco bardziej skomplikowane. Ze względu na konieczność przesyłania sporej liczby danych wymagają także szybszego interfejsu łączącego z mikrokontrolerem. Stąd też zanim przejdziemy do ich praktycznego wykorzystania, będziemy musieli bliżej zapoznać się z kilkoma układami peryferyjnymi XMEGA.

Wyświetlacze LCD alfanumeryczne zawierają zazwyczaj scalone kontrolery, stąd też procesor nie steruje bezpośrednio matrycą LCD, ale komunikuje się z wyspecjalizowanym sterownikiem, który wykonuje jego polecenia. Dlatego też wykorzystanie tego typu wyświetlaczy wymaga znajomości typu użytego w nich sterownika. Wyświetlacze LCD mogą być łączone z mikrokontrolerem za pomocą różnych magistral — najczęściej są to magistrale SPI lub równoległe (w wersji 4-, 8- i 16-bitowej). Same wyświetlacze ze względu na budowę i możliwości możemy podzielić na dwie grupy:

- ♦ Wyświetlacze alfanumeryczne — umożliwiają one wyświetlanie cyfr, liter i innych symboli. Jednak w tym typie nie mamy możliwości sterowania poszczególnymi pikselami.
- ♦ Wyświetlacze graficzne — w tym typie możemy sterować stanem każdego piksela.

Aby wyświetlana na LCD zawartość była widoczna, niezbędne jest podświetlenie. Kupując wyświetlacz, musimy zwrócić uwagę, czy jest on wyposażony w jakieś podświetlenie (można dostać wyświetlacze LCD bez podświetlenia). Oczywiście lepiej jest wybrać wyświetlacz, który ma wbudowane podświetlenie. Samo podświetlenie może być wykonane w różny sposób. W starszych modelach spotyka się podświetlenie oparte na lampie CCFL (ang. *Cold Cathode Fluorescent Lamp*). Tego typu wyświetlacze są zwykle większe, cięższe, ale przede wszystkim wymagają specjalnej przetwornicy do sterowania lampą. Jej zadaniem jest generowanie wysokiego napięcia niezbędnego do zaświecenia się lampy, a następnie utrzymywanie go w granicach zapewniających jej prawidłowe działanie. Wyświetlacze tego typu są więc kłopotliwe w użytkowaniu, musimy wydać czasami sporo pieniędzy na odpowiednią przetwornicę, a w dodatku trwałość takiej lampy jest krótsza niż diod LED. Występują także problemy ze sterowaniem jasnością podświetlenia. W nowoczesnych wyświetlaczach LCD stosuje się praktycznie wyłącznie podświetlenie za pomocą diod LED. Nie wymagają one żadnych wyrafinowanych przetwornic do zasilania, zwykle zadowolają się napięciem w granicach 3 – 12 V, mają też znacznie większą trwałość. Jasnością takiego podświetlenia z łatwością można sterować, wykorzystując technikę PWM. Podświetlenie oparte na diodach LED może mieć różny kolor — biały, żółty, zielony, niebieski, czerwony. Kolor możemy sobie więc dobrać wg własnych preferencji. Najnowsze wyświetlacze alfanumeryczne są wykonywane w technologii OLED — w tej technologii świecą poszczególne piksele, więc taki wyświetlacz nie wymaga podświetlenia. Niejako przy okazji zyskujemy znacznie lepsze czasy przełączania pikseli — dla wyświetlaczy LCD, szczególnie w niższych temperaturach, czas przełączania wynosi kilkadziesiąt do kilkuset milisekund, co powoduje pojawienie się wyraźnych „duchów” i psuje estetykę. Wyświetlacze OLED nie mają tej wady. Jednak zazwyczaj są droższe.



Wskazówka

Jeżeli możesz, staraj się wybierać wyświetlacze LCD z podświetleniem opartym na diodach LED lub wykonane w technologii OLED.

Kolejną istotną kwestią jest regulacja kontrastu. Zwykle graficzne kolorowe LCD z wbudowanymi sterownikami automatycznie regulują kontrast, dodatkowo sterownik ma zaimplementowane rejestry umożliwiające wyłącznie programową zmianę kontrastu. W przypadku prostszych sterowników, występujących najczęściej w wyświetlaczach monochromatycznych (zarówno alfanumerycznych, jak i graficznych), kontrast reguluje się, podając odpowiednie napięcie na jeden z pinów wyświetlacza. Jest to niezwykle istotne, gdyż bez właściwego napięcia regulującego kontrast na wyświetlaczu nic nie zobaczymy. Możemy w takiej sytuacji odnieść błędne wrażenie, że wyświetlacz nie działa. Z kolei jeśli kontrast będzie zbyt duży, wszystkie piksele będą włączone, co też może sugerować błędne działanie LCD.



Wskazówka

Jedną z częstszych przyczyn tego, że po prawidłowym podłączeniu wyświetlacza LCD nic na nim nie widać albo wszystkie piksele są włączone, jest nieprawidłowe napięcie na pinie regulującym kontrast.

## Obsługa wyświetlaczy alfanumerycznych

Wyświetlacze alfanumeryczne są zbudowane z matryc pikseli, zwykle o wielkości 5×8 pikseli, pogrupowanych razem. W tym typie wyświetlacza możemy sterować zawartością tak utworzonej matrycy znakowej, jednak nie ma możliwości sterowania poszczególnymi pikselami. Najczęściej spotykane wyświetlacze tego rodzaju mogą wyświetlać 1×8, 1×16, 2×16, 2×20, 4×16 znaków, gdzie pierwszy wymiar określa liczbę wyświetlanych wierszy, a drugi liczbę znaków w wierszu. Ponieważ nie sterujemy poszczególnymi pikselami, do kontrolera wysyłamy tylko kod znaku, który ma zostać wyświetlony na danej pozycji. Kontroler na podstawie kodu znaku generuje adres do specjalnego obszaru pamięci, tzw. tablicy znaków, w której znajduje się bajtowa reprezentacja poszczególnych pikseli tworzących znak. Dla matrycy 5×8 ma ona 40 bitów. Tablica ta jest umieszczona w pamięci ROM kontrolera LCD, nie możemy jej więc zmieniać. Zwykle jednak kontrolery udostępniają możliwość definicji kilku własnych znaków, dzięki czemu możemy zdefiniować np. znaki charakterystyczne dla języka polskiego. Takie podejście do generowania obrazu ma podstawową zaletę, jaką jest prostota i związane z tym niewielkie zapotrzebowanie na pamięć RAM. Do przechowania np. zawartości 8-znakowego wyświetlacza LCD wystarczy zaledwie 8 bajtów pamięci. Ma to szczególne znaczenie w prostych mikrokontrolerach, z niewielką ilością pamięci RAM. Najpopularniejszym kontrolerem stosowanym w wyświetlaczach alfanumerycznych jest *HD44780*; nawet jeśli w posiadanym wyświetlaczu jest inny kontroler, to mamy dużą szansę, że jest on z nim kompatybilny. Typowy rozkład sygnałów modułów opartych na tym kontrolerze pokazano w tabeli 22.1.

**Tabela 22.1.** Typowy układ wyprowadzeń modułów alfanumerycznych LCD opartych na kontrolerze *HD44780*

Pin	Nazwa	Funkcja
1	GND	Masa
2	Vcc	Zasilanie (typowo 5 V)
3	Vee	Regulacja kontrastu
4	RS	Zapis polecenia
5	RW	Strob odczyt/zapis
6	E	Wybór układu
7	D0	Bit danych 0
8	D1	Bit danych 1
9	D2	Bit danych 2
10	D3	Bit danych 3
11	D4	Bit danych 4
12	D5	Bit danych 5
13	D6	Bit danych 6
14	D7	Bit danych 7
15	A	Podświetlenie — anoda
16	K	Podświetlenie — katoda



Wskazówka

Przed podłączeniem modułu zawsze upewnij się, czy podany rozkład odpowiada użytemu modułowi. W szczególności sprawdź rozmieszczenie linii zasilających. Ich niewłaściwe podłączenie grozi uszkodzeniem modułu!

Zwykle piny 15 i 16 sterujące podświetleniem są umieszczone osobno lub też przed pinem 1. Stąd też zawsze należy je dokładnie zidentyfikować przed podłączeniem modułu. Do pinów tych należy podłączyć napięcie wykorzystane do podświetlenia modułu. Przyłączone napięcie zależy od typu zastosowanego podświetlenia — CCFL, folii elektroluminescencyjnej lub diod LED. W dalszej części będziemy zajmować się modulem z podświetleniem LED. W nocie katalogowej każdego modułu znajdują się szczegółowe informacje dotyczące zasilania podświetlenia. Zwykle w tym celu wymagane jest napięcie 5 V, a natężenie prądu podświetlenia wynosi 20 – 50 mA.

Wyświetlacze LCD alfanumeryczne są dostępne w handlu w wersjach dostosowanych do różnych napięć zasilających. Typowo jest to napięcie 5 V, co stwarza pewne problemy przy podłączeniu takiego LCD do procesorów rodziny XMEGA. Są one zasilane maksymalnym napięciem 3,6 V, nie ma więc możliwości, aby LCD i procesor działały w tych samych domenach napięciowych. W efekcie przy połączeniu obu układów musimy zastosować konwersję napięć. Ponieważ LCD alfanumeryczny z perspektywy procesora jest urządzeniem bardzo wolnym, całkowicie wystarczającym rozwiązaniem jest zastosowanie szeregowych rezystorów, które wraz z wbudowanymi w procesor diodami zabezpieczającymi będą stanowiły układ obniżający napięcie na magistrali komunikacyjnej z 5 V do bezpiecznego dla wejścia XMEGA napięcia. Z drugiej strony jedynek logicznych generowanych przez procesor zasilany napięciem 3,3 V jest poprawnie rozpoznawana przez LCD pracujący z napięciem 5 V. Jeśli procesor byłby zasilany bardzo niskim napięciem (rzędu 1,8 – 2,8 V), wymagany byłby bardziej skomplikowany konwerter, co zostało szerzej omówione w rozdziale 7. książki *AVR. Praktyczne projekty*.

Wyświetlacze LCD wymagają podświetlenia, bez tego wyświetlana zawartość jest praktycznie nieczytelna. W starszych konstrukcjach wykorzystywano podświetlenie w oparciu o lampy fluorescencyjne lub folie elektroluminescencyjne, jednak obecnie dominującym typem jest podświetlenie za pomocą diod LED.



Wskazówka

Ponieważ zasilamy diodę LED, wymagane jest dołączenie szeregowego rezystora. Czasami taki rezystor jest wbudowany w moduł LCD.

Podświetlenie tego typu zużywa sporą ilość energii. Innym rozwiązaniem jest kupno wyświetlacza wykonanego w technologii OLED (ang. *Organic Light-Emitting Diode*). Wyświetlacze tego typu cechują się zdecydowanie lepszym kontrastem, ale także większą szybkością przełączania, w związku z czym można za ich pomocą uzyskać o wiele lepsze efekty animacji, które są praktycznie nieosiągalne dla klasycznych wyświetlaczy LCD. Nie mają one także innej wady LCD — zależności szybkości przełączania segmentów od temperatury.

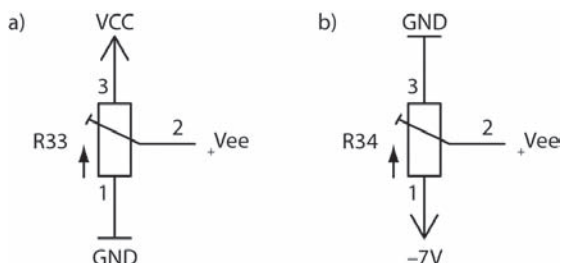
Wyświetlacze OLED nie mają wyprowadzonych pinów odpowiedzialnych za podświetlenie — po prostu go nie wymagają.

Kolejną istotną kwestią jest podłączenie właściwego napięcia regulującego kontrast wyświetlacza. Dla większości modułów LCD mieści się ono w przedziale  $GND - V_{cc}$ , a dla modułów o rozszerzonym zakresie temperatury w przedziale od  $-7$  do  $0\text{ V}$  — rysunek 22.1.

### Rysunek 22.1.

Podłączenie napięcia regulującego kontrast:

- a) moduły klasyczne,  
b) moduły o rozszerzonym zakresie temperatur



Napięcie ujemne można uzyskać z prostej pompy ładunkowej sterowanej przebiegiem PWM z procesora. Pobór prądu z tego wejścia jest znikomo mały. Alternatywnie można użyć układu *ICL7660*, który może generować napięcia ujemne.

Kolejną decyzją, jaką musimy podjąć, jest sposób podłączenia modułu LCD z mikrokontrolerem. Mamy do wyboru dwie opcje — albo podłączamy wszystkie 8 linii danych (D0 – D7), albo tylko 4 najstarsze (D4 – D7). Podłączając tylko 4 linie danych, zmniejszamy liczbę niezbędnych połączeń modułu z mikrokontrolerem oraz liczbę wykorzystanych linii I/O. Ze względu na znikomą ilość danych przesyłanych z/do modułu, wynikające z podłączenia tylko 4 linii danych spowolnienie transmisji jest bez znaczenia. **W trybie 4-bitowym najpierw transferowana jest starsza, a potem młodsza tetrada.** Niewykorzystane linie D0 – D3 można pozostawić niepodłączone; większość modułów wewnętrznie podciąga niepodłączone linie poprzez rezystor do  $V_{cc}$ .



Wskazówka

Ponieważ linie D0 – D7 są dwukierunkowe, musimy pamiętać, aby podczas operacji odczytu z LCD odpowiednie piny mikrokontrolera były ustawione jako wejścia. W przeciwnym przypadku dojdzie do konfliktu.

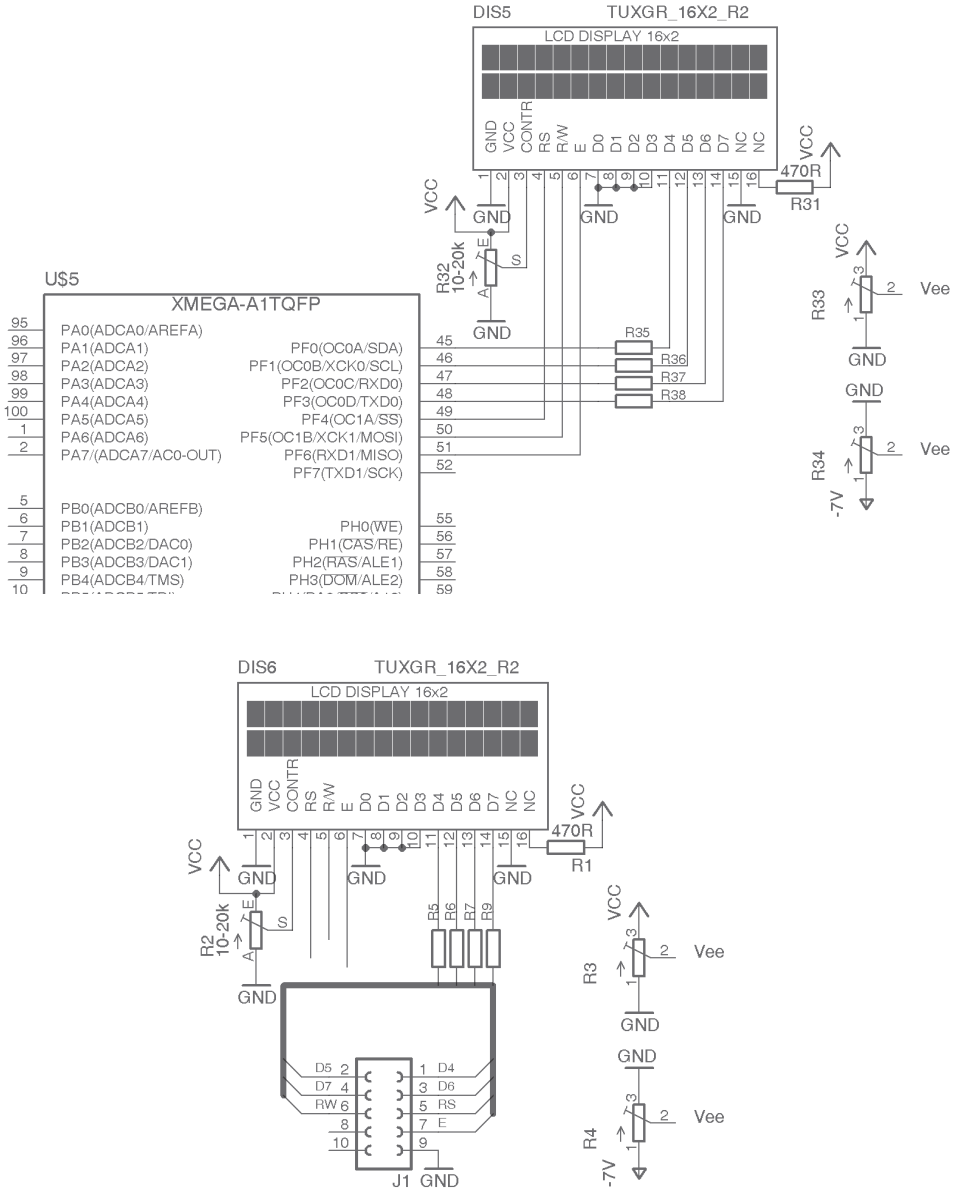
Kolejnym sygnałem, co do którego musimy podjąć decyzję, jest sygnał RW określający typ operacji (odczyt/zapis). Ponieważ zwykle zapisujemy dane do LCD, właściwie nie zachodzi konieczność ich odczytywania; możemy ten sygnał na stałe połączyć z masą, oszczędzając w ten sposób jedną linię I/O procesora. Takie rozwiązanie ma jednak poważną wadę — po jego zastosowaniu nie jest możliwe odczytywanie sygnału zajętości modułu (ang. *busy flag*). Powoduje to, że pomiędzy kolejnymi zapisami do modułu musimy stosować opóźnienia dobrane na najwolniejszą możliwą okoliczność. W efekcie cała transmisja jest wolniejsza, niż mogłaby być.



Wskazówka

Jeśli korzystamy z tych modułów, nie musimy pisać własnych procedur ich obsługi. Instalując środowisko WinAVR (lub nowy toolchain firmy Atmel), w katalogu `doc\avr-libc\examples\sdtiodemo` znajdziemy gotowe funkcje obsługi HD44780. Funkcje te nie działają z mikrokontrolerami XMEGA, stąd też w załączonych przykładach zostały one zmodyfikowane tak, aby poprawnie współpracować również z tymi mikrokontrolerami.

Możemy też skorzystać z licznych przykładów dostępnych w internecie. W dalszej części rozdziału zostaną wykorzystane funkcje z pakietu WinAVR napisane przez Joerga Wunscha. Pokazane przykłady zakładają, że wyświetlacz jest połączony z mikrokontrolerem tak jak na rysunku 22.2.



**Rysunek 22.2.** Połączenie LCD z mikrokontrolerem. Na górze podłączenie do mikrokontrolera XMEGA, na dole podłączenie do modułu Xplained. Na schematach pokazano także opcjonalne sposoby podłączenia pinu Vee odpowiedzialnego za regulację kontrastu LCD

Zanim coś wyświetlimy na LCD, musimy poznać, w jaki sposób są generowane znaki. Sterownik LCD zawiera w sobie, niezależnie od typu użytej matrycy, 80 bajtów pamięci DDRAM (ang. *Display Data RAM*). Jeśli zastosowana matryca wyświetla mniejszą liczbę znaków, to część komórek pamięci jest nieużywana. Możemy je wykorzystać do innych celów — np. jako zwykłą pamięć RAM. Możemy też użyć jej do przesuwania napisów, które nie mieszczą się na wyświetlaczu. Aby można było umieścić znak na właściwej pozycji, niezbędne jest poznanie organizacji pamięci kontrolera. W zależności od użytej matrycy pamięć ta jest zorganizowana następująco:

- ♦ Dla matryc  $16 \times 1$  — pamięć jest podzielona na dwa obszary, każdy zawierający po 8 znaków. Obszar pierwszy (od lewej) ma adresy  $0x00$  do  $0x07$ , a obszar drugi (prawy) adresy  $0x40$  do  $0x47$ . Stąd też gdy wyświetla się tekst przekraczający granicę obszarów, należy zadbać, aby poszczególne litery trafiły pod właściwe adresy.
- ♦ Dla matryc  $20 \times 1$  — adresy  $0x00$  do  $0x13$ .
- ♦ Dla matryc  $16 \times 2$  — adres pierwszej linii zaczyna się od  $0x00$  do  $0x0f$ , adres drugiej linii od  $0x40$  do  $0x4f$ .
- ♦ Dla matryc  $20 \times 2$  — adresy pierwszej linii to  $0x00$  do  $0x13$ , a drugiej  $0x40$  do  $0x53$ .
- ♦ Dla matryc  $40 \times 2$  — adresy pierwszej linii to  $0x00$  do  $0x27$ , drugiej linii  $0x40$  do  $0x67$ .
- ♦ Dla matryc  $20 \times 4$  — adresy pierwszej linii to  $0x00$  do  $0x13$ , drugiej  $0x40$  do  $0x53$ , trzeciej  $0x14$  do  $0x27$ , a czwartej  $0x54$  do  $0x67$ .

Jak widać, w niektórych typach wyświetlaczy po każdej linii zostaje od kilku do kilkunastu bajtów, których zawartość nie jest wyświetlana. Ma to pewną zaletę — możemy do pamięci modułu wpisywać teksty dłuższe, niż są wyświetlane na ekranie, co ułatwia przewijanie tekstu. Niestety ma to też pewną wadę — funkcje wyświetlające coś na ekranie muszą „wiedzieć”, jaką organizację ma matryca. W przeciwnym przypadku nie ma możliwości prawidłowego wyliczenia adresu komórki, do której ma odbyć się zapis.



W dalszej części rozdziału zostanie wykorzystany moduł LCD o organizacji  $16 \times 2$  (ale można też bez problemu wykorzystać dowolny inny).

Zanim będzie można coś wyświetlić na ekranie, moduł musi zostać zainicjalizowany. W trakcie inicjalizacji ustawiane są podstawowe parametry pracy sterownika LCD — organizacja matrycy (liczba linii), wielkość czcionki ( $5 \times 8$  lub  $8 \times 11$  pikseli). Wielkość czcionki zależy od typu podłączonej matrycy LCD. Dla konkretnego modułu LCD tylko jedna z tych wartości będzie poprawna. Oprócz inicjalizacji modułu należy także poprawnie ustawić stan pinów I/O wykorzystywanych do podłączenia modułu LCD.

Większość pracy związanej z inicjalizacją modułu LCD, odbieraniem i wysyłaniem do niego danych wykonuje biblioteka Joerga Wunscha. Dostarcza ona niskopoziomowe funkcje umożliwiające komunikację ze sterownikiem *hd44780* i innymi kompatybilnymi z nim sterownikami. Wystarczy, że do aplikacji, w której planujemy wykorzystać taki moduł, dołączymy plik *hd44780.c* oraz plik nagłówkowy *hd44780.h*.

Oba pliki „zakładają”, że istnieje plik nagłówkowy *defines.h*, zawierający podstawowe definicje związane ze sposobem podłączenia modułu LCD z mikrokontrolerem. Na początku należy określić, które linie I/O zostały wykorzystane do podłączenia modułu:

```
#define HD44780_RS A, 4
#define HD44780_RW A, 5
#define HD44780_E A, 6
#define HD44780_D4 A, 0
```

W powyższym przykładzie do kolejnych linii portu A (A4 – A6) zostały podłączone sygnały RS, RW i E modułu, natomiast do linii A0 – A3 linie D4 – D7 modułu LCD. Ważne jest, aby linie danych łączyć z pinami I/O w kolejności rosnącej. Biblioteka wykorzystuje do transmisji danych pomiędzy modułem a mikrokontrolerem tylko 4 linie danych — dzięki temu możemy zmniejszyć liczbę połączeń, a zwolnione linie I/O wykorzystać do innych celów. W pliku występuje jeszcze jedna definicja:

```
#define USE_BUSY_BIT 1
```

Określa ona, że biblioteka będzie używać flagi Busy modułu LCD — umożliwia to uzyskanie optymalnej szybkości współdziałania z modułem. Zakończenie realizacji bieżącej operacji moduł sygnalizuje zmianą stanu tej flagi, co umożliwia niezwłoczne wysłanie kolejnej komendy. Jeśli nie używamy flagi Busy, program musi generować maksymalne zalecane w nocie katalogowej opóźnienia, w efekcie cała transmisja zwalnia. Po zdefiniowaniu używanych pinów w pliku *defines.h* możemy rozpocząć korzystanie z biblioteki.



W pliku *hd44780.h* są udostępnione prototypy funkcji odpowiedzialnych za współpracę z modułem LCD.

## Funkcje biblioteczne

Podstawę modułu stanowią dwie funkcje umożliwiające odpowiednio wysłanie danych do modułu oraz ich odczytanie:

```
void hd44780_outbyte(uint8_t b, uint8_t rs);
uint8_t hd44780_inbyte(uint8_t rs);
```

Obie funkcje wysyłają/odczytują 8-bitową daną; w przypadku wykorzystania tylko 4-bitowej szyny danych wysyłanie i odczytywanie danych odbywa się w dwóch etapach. Dodatkowo zmienna *rs* określa tryb dostępu. W przypadku wysyłania danych do modułu *rs* = 0 powoduje, że dane zostaną wysłane do rejestru komend, a jeśli *rs* = 1, dane trafią do rejestru danych. Przy odczycie jest podobnie, z tym że dla *rs* = 0 jest odczytywana flaga Busy i rejestr AC (ang. *Address Counter*).

Kolejna funkcja odpowiada za oczekiwanie na gotowość modułu:

```
void hd44780_wait_ready(bool isLong);
```

W przypadku gdy w pliku *defines.h* symbol *USE\_BUSY\_FLAG* ma wartość 1, parametr *isLong* jest ignorowany, a optymalizator kompilatora usuwa go, gdyż nie jest on wykorzystywany przez funkcję. Kiedy wartość *USE\_BUSY\_FLAG* jest różna od 1, parametr



ten określa długość opóźnienia. Jeśli jest różna od 0, opóźnienie wynosi 1,52 ms, a jeżeli jest równa 0, opóźnienie wynosi 37  $\mu$ s. Ponieważ lepiej jest używać flagi Busy, możemy ten parametr ignorować.

Powyższe trzy funkcje wystarczą do komunikacji z modułem, dla wygody zdefiniowano jednak kolejne. Funkcje `hd44780_outcmd(n)` i `hd44780_incmd()` umożliwiają odpowiednio wysłanie polecenia do rejestru komend kontrolera lub odczytanie bitu Busy i licznika AC. Dwie kolejne — `hd44780_outdata(n)` oraz `hd44780_indata()` — umożliwiają zapis i odczytanie rejestru danych.

Oprócz funkcji odpowiedzialnych za komunikację z modułem LCD zdefiniowano także funkcję przeprowadzającą podstawową inicjalizację modułu, `hd44780_init(void)`, oraz funkcję `hd44780_powerdown(void)`, ustawiającą wszystkie linie użyte do komunikacji w stan 0, co umożliwia bezpieczne odłączenie zasilania od modułu LCD. **W przypadku pozostawienia którejś z wyżej wymienionych linii w stanie 1 przy odłączonym zasilaniu modułu stanowi ona jego zasilanie. W efekcie mimo wyłączenia moduł może pobierać znaczny prąd.**

Szerszego omówienia wymaga funkcja `hd44780_outcmd(cmd)`, która umożliwia wysłanie polecenia do rejestru poleceń sterownika LCD. Dla wygody zdefiniowane zostały różne komendy ułatwiające współpracę z modułem LCD. Komendy te są przekazywane jako argument wywołania funkcji `hd44780_outcmd, np.:`

```
hd44780_outcmd(HD44780_CLR);
```

powoduje wysłanie do sterownika polecenia `HD44780_CLR`, czyszczącego zawartość pamięci DDRAM kontrolera.

## Komendy

Funkcja `hd44780_outcmd` przyjmuje jako argument predefiniowane makrodefinicje, znacznie ułatwiające współpracę z kontrolerem modułu LCD. Poniżej zostaną one pokrótce omówione.

### HD44780\_SHIFT(shift, right)

Ta makrodefinicja powoduje przesunięcie pozycji wyświetlanego tekstu (`shift = 1`) o jedną pozycję w lewo (argument `right = 0`) lub w prawo (argument `right = 1`). W przypadku gdy `shift = 0`, przesuwany w lewo lub prawo jest tylko kursor. Wysyłając wielokrotnie to polecenie, uzyskujemy kolejne przesunięcia o jedną pozycję w lewo lub prawo. Polecenie to służy głównie do przesuwania tekstu. Jednak ze względu na dużą bezwładność tekstowych modułów LCD nie należy przesuwać liter częściej niż ok. raz na sekundę — częstsze przesuwanie daje bardzo nieprzyjemne efekty wizualne. Każdy bufor linii jest „zawijany”, to znaczy, że po ostatnim znaku z prawej jest wyświetlany pierwszy znak z lewej i odwrotnie. Należy pamiętać, że długość bufora jest stała dla danego modelu matrycy. Np. dla modułów  $16 \times 2$  bufor jednej linii wynosi dokładnie 40 bajtów, z czego 16 jest jednorazowo wyświetlanych. Poniższy kod cyklicznie przewija wyświetlany tekst w lewo:

```
lcd_puttext_P(PSTR("Test bufora i przewijania tekstu.\nTen tekst przewija sie w lewo"));
while(1)
{
```

```

    hd44780_outcmd(HD44780_SHIFT(1,0));
    _delay_ms(700);
}

```

### HD44780\_FNSET(if8bit, twoline, font5x10)

Polecenie to wybiera typ interfejsu — 8-bitowy, jeśli wartość parametru `if8bit` jest równa 1, lub 4-bitowy, jeśli jest równa 0. Typ interfejsu musi odpowiadać sposobowi połączenia modułu z mikrokontrolerem. Parametr `twoline` określa, czy podłączona matryca LCD ma jedną linię znaków, czy dwie. Matryce, które mają 4 linie tekstu, są traktowane przez kontroler jako matryce dwuliniowe — parametr `twoline` powinien mieć wartość 1. Jeśli dla matrycy zawierającej dwie lub więcej linii ustawimy go na 0, to wyświetlana będzie tylko jedna linia (w przypadku matryc dwuliniowych górna), a w przypadku matryc 4-liniowych będą wyświetlane dwie linie, rozdzielone linią pustą.

Ostatni parametr określa typ używanego generatora znaków. Powinien on odpowiadać użytej w module matrycy, zwykle więc jego wartość wynosi 0.

### HD44780\_DISPCTL(dispc, cursor, blink)

Ustawia różne parametry pracy matrycy. Jeśli parametr `dispc` ma wartość 1, to matryca jest włączona (wyświetla zawartość DDRAM), a jeśli 0, to matryca jest wyłączona, lecz stan pamięci DDRAM nie ulega zmianie. Np. kod:

```

while(1)
{
    hd44780_outcmd(HD44780_DISPCTL(0,1,1));
    _delay_ms(700);
    hd44780_outcmd(HD44780_DISPCTL(1,1,1));
    _delay_ms(700);
}

```

powoduje naprzemienne włączanie i wyłączanie matrycy, w efekcie wyświetlany tekst mruga.

Parametr `cursor` określa, czy sprzętowo wyświetlany kursor ma być widoczny (wartość parametru 1), czy niewidoczny (wartość parametru 0). Jeśli kursor jest widoczny, to dodatkowo możemy określić, czy ma migać (wartość parametru `blink` równa 1), czy nie (`blink` = 0). Jeśli kursor jest widoczny i nie miga, jest wyświetlany w postaci poziomej kreski. Wyświetlanie kursora może być użyteczne w sytuacji, kiedy użytkownik wprowadza jakieś dane. W pozostałych sytuacjach kursor możemy wyłączyć.

### HD44780\_ENTMODE(inc, shift)

Określa sposób wprowadzania danych do pamięci. Jeśli parametr `inc` ma wartość 1, to po każdej operacji zapisu/odczytu adres pamięci jest automatycznie zwiększany o jeden, co przyspiesza operację zapisu/odczytu kolejnych znaków. Jeśli wartość `inc` jest równa 0, to adres jest automatycznie zmniejszany o 1. Dodatkowo parametr `shift` określa, czy wyświetlany obraz ma być automatycznie przesuwany, tak aby wyświetlać ostatnio wprowadzony znak (`shift` = 1) lub go nie wyświetlać (`shift` = 0).

## HD44780\_CLR

Powoduje wyczyszczenie (wypełnienie znakami o kodzie 32) całej zawartości pamięci DDRAM. Jednocześnie kursor wraca do pozycji pierwszej (pierwsza linia, pierwszy znak).



Uwaga

Niewyświetlane bajty pamięci DDRAM także są kasowane. Jeśli są one używane do przechowywania zmiennych programu, to przed operacją kasowania należy je skopiować w bezpieczne miejsce. W przeciwnym przypadku zostaną utracone.

Wskaźnik adresu do pamięci DDRAM jest ustawiany na 0, zerowane jest także przesunięcie.

## HD44780\_HOME

Powoduje powrót kursora do pozycji początkowej — pierwsza linia, pierwszy znak. Wskaźnik adresu do pamięci DDRAM w efekcie zawiera 0, zerowane jest również przesunięcie wyświetlania znaków, w efekcie są one wyświetlane od pozycji 0.

## HD44780\_DDADDR(addr)

Ustawia następną adres zapisu/odczytu w pamięci DDRAM na `addr`. Tylko 7 bitów argumentu `addr` jest branych pod uwagę, w związku z czym można zaadresować maksymalnie 128 bajtów pamięci (z czego sterownik ma tylko 80, pozostałe lokacje są nieużywane).

## HD44780\_CGADDR(addr)

Ustawia następną operację odczytu/zapisu w pamięci CGRAM na `addr`. Pod uwagę branych jest tylko 6 bitów zmiennej `addr`, w efekcie można zaadresować maksymalnie 64 bajty pamięci CGRAM.

Przedstawione powyżej funkcje zapewniają jedynie podstawową komunikację z modulem LCD. Potrzebujemy jednak ciągle funkcji odpowiedzialnej za jego inicjalizację — funkcja biblioteczna przeprowadza ten proces tylko w podstawowym zakresie, musimy jeszcze zadbać o właściwe ustawienie parametrów związane z posiadaną matrycą. Napiszmy więc własne rozwinięcie funkcji inicjalizacyjnej:

```
void lcd_init()
{
    hd44780_init(); // Podstawowa inicjalizacja modułu
    hd44780_outcmd(HD44780_CLR); // Wyczyść pamięć DDRAM
    hd44780_wait_ready(1000);
    hd44780_outcmd(HD44780_ENTMODE(1, 0)); // Tryb autoinkrementacji AC
    hd44780_wait_ready(1000);
    hd44780_outcmd(HD44780_DISPCTL(1, 0, 0)); // Włącz wyświetlacz, wyłącz kursor
    hd44780_wait_ready(1000);
}
```

Zauważ, że po wysłaniu każdego polecenia do modułu jest wywoływana funkcja `hd44780_wait_ready(1000)` odpowiedzialna za oczekiwanie, aż realizacja wysłanego polecenia dobiegnie końca. **Jest to niezbędne, gdyż do czasu ukończenia bieżącej**

**operacji dalsze polecenia są ignorowane.** Jako argument podany jest parametr 1000 — jest on bez znaczenia, nasza biblioteka korzysta z flagi Busy i po prostu go ignoruje. Po wykonaniu powyższej funkcji moduł LCD jest poprawnie zainicjalizowany, a jego pamięć jest wyczyszczona (zawiera bajty 0x20) i w efekcie moduł nic nie wyświetla. Od tej chwili możemy wysyłać znaki do wyświetlenia.



Po wykonaniu inicjalizacji ekran może pozostać czysty lub wyświetlać całkowicie wypełnione kwadraty (wszystkie piksele włączone). Jeśli po inicjalizacji wszystkie piksele są włączone, należy nieco zmniejszyć kontrast.

Aby sobie ułatwić to zadanie, zdefiniujemy funkcję wysyłającą do modułu jeden znak:

```
void lcd_putchar(char c)
{
    static bool second_nl_seen;
    static uint8_t line=0;

    if ((second_nl_seen) && (c != '\n') && (line==0))
        // Odebrano pierwszy znak
        hd44780_wait_ready(40);
        hd44780_outcmd(HD44780_CLR);
        hd44780_wait_ready(1600);
        second_nl_seen=false;
    }
    if (c == '\n')
    {
        if (line==0)
        {
            line++;
            hd44780_outcmd(HD44780_DDADDR(64)); // Adres pierwszego znaku drugiej linii
            hd44780_wait_ready(1000);
        }
        else
        {
            second_nl_seen=true;
            line=0;
        }
    }
    else
    {
        hd44780_outdata(c);
        hd44780_wait_ready(40);
    }
}
```

Funkcja ta, ze względu na skomplikowane adresowanie pamięci DDRAM, jest zależna od konkretnego modelu modułu LCD. Będzie prawidłowo współpracować z modułami 2×16, 2×20 i 2×40 znaków. W przypadku innych modułów pozycje niektórych znaków mogą być nieprawidłowe.

Możemy już wyświetlać na LCD pojedyncze znaki, czas na wyświetlanie tekstu. Ponieważ teksty jako stałe dla zaoszczędzenia cennej pamięci RAM przechowuje się zazwyczaj w pamięci FLASH, zdefiniujemy funkcję odczytującą podany łańcuch znakowy z pamięci FLASH, a następnie umieszczającą go na LCD:

```
void lcd_puttext_P(prog_char *txt)
{
    char ch;
    while((ch=pgm_read_byte(txt)))
    {
        lcd_putchar(ch);
        txt++;
    }
}
```

Powyższa funkcja wysyła kolejne znaki odczytane z pamięci FLASH na LCD za pomocą funkcji `lcd_putchar`. Po napotkaniu znaku końca łańcucha (0, nul) przerywa swoje działanie.

Pozostaje nam przetestować działanie przedstawionych funkcji:

```
int main()
{
    lcd_init();
    lcd_puttext_P(PSTR("Test LCD.\nDruga linia"));
    while(1) {}
}
```

Na ekranie modułu LCD powinien wyświetlić się napis:

```
Test LCD.
Druga linia
```



Wskazówka

Jeśli na LCD nic nie widzimy, to w pierwszej kolejności powinniśmy sprawdzić kontrast, zmieniając napięcie na pinie Vee modułu LCD.

Warto zdefiniować jeszcze funkcję ustawiającą pozycję, od której będzie wyświetlany następny znak:

```
void lcd_goto(uint8_t x, uint8_t y)
{
    hd44780_outcmd(HD44780_DDADDR(0x40*y+x));
    hd44780_wait_ready(1000);
}
```

Funkcja ta przyjmuje dwa parametry: `x` i `y`, określające pozycję, na której zostanie wyświetlony następny znak wysłany z mikrokontrolera do modułu LCD. Dzięki temu z łatwością możemy uaktualniać zawartość LCD, bez konieczności ponownego przesyłania wszystkich danych. Funkcja ta jest zależna od typu zastosowanego modułu.

Ostatnią funkcją jest funkcja umożliwiająca czyszczenie zawartości pamięci modułu. Odpowiedzialne za to jest proste polecenie wysyłane do sterownika:

```
void lcd_cls()
{
    hd44780_outcmd(HD44780_CLR);
    hd44780_wait_ready(false);
}
```

Oprócz czyszczenia zawartości pamięci modułu powoduje ono ustawienie kursora w punkcie (0,0).

## Definiowanie własnych znaków

Układ HD44780 jest wyposażony w 64 bajty pamięci CGRAM (ang. *Character Generator RAM*), która może przechowywać mapy bitowe 8 znaków o matrycy 5×8 pikseli lub 4 znaków o matrycy 5×10 pikseli. Przykład definicji znaku na matrycy 5×8 pokazano na rysunku 22.3.

### Rysunek 22.3.

*Przykład definicji własnego znaku w pamięci CGRAM. Znak rysujemy na matrycy 5×8 bitów, pola zaciemnione odpowiadają wartości 1, pola białe wartości 0. Wartość wyliczoną dla każdej linii pikseli wpisujemy do pamięci CGRAM sterownika. Najstarsze 3 bity każdego bajta (pola szare) są bez znaczenia — wyświetlacz pomija je przy wyświetlaniu obrazu*

Adres		Wartość
0		00100 = 4
1		01110 = 14
2		11111 = 31
3		00100 = 4
4		00100 = 4
5		11111 = 31
6		01110 = 14
7		00100 = 4

Dostęp do tej pamięci jest możliwy dzięki wysłaniu polecenia ustawiającego adres zapisu/odczytu pamięci CGRAM. Po jego wysłaniu wszystkie operacje zapisu i odczytu dotyczą tej pamięci, a nie pamięci DDRAM. Aby wrócić do adresowania pamięci DDRAM, należy wybrać jej adres lub dokonać operacji wymagającej przesunięcia kursora.



Wskazówka

Pamięć CGRAM jest pamięcią ulotną, po ponownym włączeniu zasilania definicje znaków muszą być ponownie przesłane przez procesor.

Bity o wartości 1 w definicji znaku odpowiadają włączonym pikselom, bity o wartości 0 pikselom wyłączonym. Najstarsze 3 bity każdego bajta w pamięci CGRAM są bez znaczenia — te pozycje nie są wyświetlane przez matrycę. Definiując znak, należy także pamiętać, że ostatnia linia wyświetlana przez matrycę (w zależności od wielkości matrycy znaku linia 8 lub 10) powinna zawierać same zera — w tej linii będzie wyświetlany kursor.

Najczęściej matryce LCD podłączone do sterownika HD44780 mają organizację 5×8, czyli jeden znak jest definiowany przez 8 kolejnych bajtów określających stan poszczególnych linii tworzących znak. Stwórzmy funkcję, której zadaniem będzie załadowanie definicji znaków z pamięci FLASH mikrokontrolera do pamięci CGRAM:

```
void lcd_defchar_P(uint8_t charno, prog_uint8_t *chardef)
{
    hd44780_outcmd(HD44780_CGADDR(charno*8));
    hd44780_wait_ready(40);
    for(uint8_t c=0;c<8;c++)
    {
        hd44780_outdata(pgm_read_byte(chardef));
    }
}
```

```

    hd44780_wait_ready(40);
    chardef++;
}
}

```

Funkcja `LCD_defchar_P` jako argumenty przyjmuje numer definiowanego znaku (0 – 7) oraz wskaźnik do obszaru pamięci FLASH zawierającego definicję znaku (obszar ten zajmuje 8 następujących po sobie bajtów). Jak widzimy, funkcja najpierw ustawia właściwy adres, pod którym znajdzie się początek definicji znaku w pamięci CGRAM — ponieważ jeden znak jest opisywany za pomocą 8 bajtów, kolejne adresy znaków to *kod\_znaku* × 8. Następnie dane opisujące znak są ładowane z pamięci mikrokontrolera i zapisywane w pamięci CGRAM sterownika. Poniższy program ilustruje wykorzystanie funkcji `LCD_defchar_P`:

```

prog_uint8_t char1[8][8]={0,0,0,0,0,0,0,255}, {0,0,0,0,0,0,255,255},
↳{0,0,0,0,0,255,255,255}, {0,0,0,0,255,255,255,255}, {0,0,0,255,255,255,255,255},
↳{0,0,255,255,255,255,255,255}, {0,255,255,255,255,255,255,255},
↳{255,255,255,255,255,255,255,255}};

int main()
{
    for(uint8_t x=0;x<8;x++) lcd_defchar_P(x,&char1[x][0]); // Załaduj wzorce znaków
    lcd_init();
    for(uint8_t x=0;x<8;x++) lcd_putchar(x); // Wyświetl znaki o kodach 0 – 7
    while(1) {}
}

```

W pierwszej linii funkcji `main` następuje załadowanie nowych definicji znaków do pamięci CGRAM, następnie inicjalizowany jest wyświetlacz, po czym w pierwszej linii są wyświetlane znaki o kodach 0 – 7, a więc znaki, których definicja została przesłana z mikrokontrolera.

Zauważ, że pamięć CGRAM, podobnie jak DDRAM, możemy zapisywać i odczytywać przed inicjalizacją modułu LCD. Proces inicjalizacji jest potrzebny tylko do prawidłowego wyświetlania na ekranie LCD.

Ponieważ dane potrzebne do wyświetlenia znaku są na bieżąco pobierane z pamięci podczas odświeżania matrycy LCD, jakiegokolwiek zmiany w definicji znaku, kiedy jest on widoczny na LCD, pociągają za sobą natychmiastowe zmiany w wyświetlanym obrazie. Można tę właściwość sterownika wykorzystać do stworzenia prostych animacji:

```

void lcd_box(uint8_t y)
{
    hd44780_outcmd(HD44780_CGADDR(0)); // Zaczynamy od znaku o kodzie 0
    hd44780_wait_ready(40);
    for(uint8_t c=0;c<y;c++) // Stwórz linie z włączonymi pikselami
    {
        hd44780_outdata(0xFF);
        hd44780_wait_ready(40);
    }
    for(uint8_t c=y;c<8;c++) // Stwórz linie z wyłączonymi pikselami
    {
        hd44780_outdata(0x00);
        hd44780_wait_ready(40);
    }
}

```

```

}

int main()
{
    lcd_init();
    lcd_putchar(0);
    while(1)
    {
        for(uint8_t x=0;x<8;x++)
        {
            lcd_box(x);
            _delay_ms(100);
        }
        for(int8_t x=7;x>=0;x--)
        {
            lcd_box(x);
            _delay_ms(100);
        }
    }
}

```

Funkcja main wyświetla na LCD tylko jeden znak o kodzie 0, a następnie cyklicznie zmienia jego definicję w pamięci CGRAM. W efekcie postać znaku wyświetlana na ekranie ciągle się zmienia.

## Transakcyjna obsługa LCD

Pokazana do tej pory obsługa LCD alfanumerycznego miała pewne wady. Jak wiemy, operacje odwołujące się do kontrolera LCD są niezwykle wolne, cykl odczytu/zapisu dowolnego rejestru kontrolera trwa co najmniej 1600 ns, co dla MCU taktowanego zegarem 32 MHz przekłada się na czas wykonania około 52 instrukcji asemblera. Co gorsza, większość komend realizowanych przez kontroler jest wykonywana w czasie około 37  $\mu$ s, co przekłada się na tysiące zmarnowanych taktów MCU. Stąd też jakkolwiek zapis/odczyt LCD to z perspektywy programu straszne marnotrawstwo czasu.



Wskazówka

Problem ten dotyczy każdego urządzenia IO, które działa wolniej niż procesor. W takiej sytuacji warto napisać pewną warstwę pośrednią — sterownik — którego zadaniem będzie buforowanie żądań dostępu do urządzenia i rozstrzyganie konfliktów, dzięki czemu różne części programu będą „miały wrażenie”, że mają wyłączny dostęp do urządzenia.

Pokazane wcześniej funkcje obsługi LCD mają jeszcze jedną wadę — nie są współbieżne, czyli nie można ich wykonywać jednocześnie w kilku miejscach programu, w tym w szczególności w programie głównym i w funkcjach obsługi przerwań. W wielu przypadkach ta wada jest bez znaczenia, jednak czasami jest uciążliwa — dla przykładu gdy na wyświetlany tekst chcielibyśmy nałożyć inne informacje, np. o dacie lub godzinie. Oczywiście wyświetlanie można zorganizować tak, aby nie było potrzeby pisania współbieżnego kodu, np. poprzez zastosowanie bufora pamięci VRAM kontrolera LCD. Rodzi to jednak kolejne problemy i jest mało eleganckie.



Zastanówmy się, w jaki sposób, najlepiej jednocześnie, rozwiązać problem wolnego dostępu do urządzenia i współbieżności kodu. Rozwiązaniem może być wprowadzenie tzw. transakcji. **Transakcją nazywamy zbiór operacji tworzących pewną całość. To, co wyróżnia tak wykonywany zestaw operacji, to atomowość, spójność, izolacja i trwałość.** Pod tymi terminami kryje się istota wykonywanej operacji. Atomowość transakcji zapewnia, że jest ona wykonywana w sposób niepodzielny, co gwarantuje nam prawidłową współpracę z danym urządzeniem I/O, w tym przypadku kontrolerem LCD. Ważna dla nas jest także izolacja — dzięki temu inne operacje na LCD (inne transakcje) nie będą miały wpływu na wynik aktualnie wykonywanej transakcji — nie będą zakłócały jej wykonania, z drugiej strony nasza transakcja nie zakłóci wykonania innych. Trwałość z kolei zagwarantuje nam, że jeśli sterownik potwierdzi wykonanie transakcji, to znaczy, że efekt jej działania jest wyświetlony na LCD. Ta cecha nie jest nam aż tak potrzebna — jest ona, wydawałoby się, oczywista. Niemniej jest to pewna gwarancja, że cała operacja przebiegła poprawnie i w efekcie LCD znajduje się w określonym przez nas stanie. Tyle teorii, przejdźmy do praktyki.



Kody poniższego przykładu znajdują się w katalogu *Przykłady/LCD-alfa/LCD-transact*.

Aby zapewnić transakcyjny dostęp do LCD, musimy stworzyć pewne struktury danych, których zadaniem będzie przechowywanie informacji o operacjach składających się na transakcje i stanie ich wykonania. W tym celu posłużymy się wiadomościami z rozdziału 5. „Jak uporządkować chaos, czyli złożone typy danych i listy”. Do przechowywania oczekujących transakcji posłużymy się buforem pierścieniowym, zanim jednak go zaimplementujemy, zastanówmy się, jak powinna wyglądać struktura przechowująca informacje o transakcji. Struktura ta powinna zawierać co najmniej:

- ♦ pole danych (data) z informacjami, które mają być przesłane do kontrolera LCD,
- ♦ pole określające liczbę danych do wysłania (len),
- ♦ pole określające realizowane polecenie dla LCD (cmd) — pole to posłuży programowi sterownika do właściwej interpretacji danych znajdujących się w polu data,
- ♦ kilka pól kontrolnych przechowujących informacje potrzebne do realizacji samej transakcji. Jedno z nich — Ready — będzie przyjmować wartość true, jeśli dana transakcja została zrealizowana, lub false w przeciwnym przypadku. Znaczenie drugiego pola — SelfDel — może wydawać się nieco enigmatyczne; pole to określa, czy pamięć przydzielona dla transakcji ma być zwolniona w sterowniku LCD (wartość true), czy też zostanie zwolniona w programie (wartość false).

Oto jak może wyglądać struktura definiująca transakcję:

```
typedef struct
{
    volatile bool    Ready : 1; // true, jeśli transakcja została zrealizowana
    bool    SelfDel : 1;        // true, jeśli pamięć przydzielona transakcji ma zostać zwolniona
    LCD_Cmd    cmd : 2;        // Polecenie do zrealizowania
    uint8_t    len : 6;        // Długość polecenia
};
```

```

    unsigned    : 0;           // Wyrównanie do granicy bajta
    uint8_t data[];         // Dane transakcji
} LCD_trans;

```

Należy zwrócić uwagę na pole:

```
unsigned    : 0;
```

Nie definiuje ono żadnego pola struktury, ale jedynie informuje kompilator, że kolejne pole (w naszym przykładzie o nazwie `data`) ma być wyrównane do granicy bajta — dzięki temu dostęp do kolejnego pola będzie odbywał się szybciej. Z kolei samo pole `data` jest zadeklarowane jako tzw. **typ niekompletny**.



Typ niekompletny (ang. *incomplete type*) to typ, którego rozmiaru nie da się określić.

W pokazanym przykładzie rozmiaru pola `data` nie da się określić, gdyż został on zadeklarowany jako typ tablicowy, z tym że nie podano wielkości tablicy. Wielkości nie podano, gdyż na etapie deklaracji struktury nie wiemy, jakiej wielkości dane będą w niej przechowywane. Tego typu deklaracja ma bardzo poważne konsekwencje — skoro wielkości pola `data` nie da się określić, to jaki będzie wynik działania operatora `sizeof`  $\rightarrow$  `(LCD_trans)?` Kompilator stara się jakoś wybrnąć z tej sytuacji, w związku z czym zwróci wielkość struktury, ale taką, jakby pole `data` w ogóle nie istniało — jego rozmiar będzie wynosił 0. W ramach ćwiczenia zastanów się, ile bajtów pamięci będzie zajmowała struktura `LCD_trans`.

Skoro dysponujemy już strukturą przechowującą poszczególne transakcje, pora na stworzenie obiektu przechowującego kolekcję transakcji. Dodanie kolejnej struktury będzie polegało na dodaniu jej do takiego obiektu, z drugiej strony program sterownika LCD będzie mógł z tej kolekcji pobierać kolejne transakcje do realizacji. Opis ten przypomina opis działania bufora pierścieniowego — z jednym wyjątkiem — po zapelnieniu całego bufora nie będziemy usuwać niezrealizowanych transakcji, lecz zablokujemy dodawanie kolejnych. W tym celu posłużymy się definicją następną strukturą:

```

typedef struct
{
    LCD_trans *elements[LCD_MAXTRANS]; // Wskaźniki do transakcji
    uint8_t Beg;                       // Pierwszy element bufora
    uint8_t Count;                     // Liczba elementów w buforze
} CircBuffer;

```

Struktura ta zawiera deklarację tablicy wskaźników na transakcje (pole `elements`). Ponieważ jest to tablica, musimy znać liczbę jej elementów — jest ona zadeklarowana w stałej `LCD_MAXTRANS`:

```
#define LCD_MAXTRANS 10           // Maksymalna liczba pamiętanych transakcji
```

Do realizacji bufora są potrzebne jeszcze dwa pola: `Beg`, wskazujące na pierwszą transakcję w buforze, oraz pole `Count` określające liczbę transakcji znajdujących się w buforze. Jeśli `Count == 0`, to znaczy, że w buforze nie ma żadnych oczekujących transakcji, a jeśli `Count == LCD_MAXTRANS`, to znaczy, że bufor jest pełny i nie można

dodawać do niego kolejnych transakcji. Właściwość tę wykorzystują dwie zdefiniowane w pliku *LCD\_trans.c* funkcje:

```
static inline bool cbIsFull(CircBuffer *cb)
{
    return cb->Count == LCD_MAXTRANS;
}

static inline bool cbIsEmpty(CircBuffer *cb)
{
    return cb->Count == 0;
}
```

Jak pamiętamy, w przypadku funkcji modyfikatory `static inline` powodują, że dana funkcja nie będzie istnieć jako samodzielna, lecz w każdym miejscu, w którym jest wywoływana, jest wstawiany raczej jej kod, a nie wywołanie. Ma to znaczenie dla krótkich funkcji (a więc takich jak pokazane powyżej), dla których koszty wywołania są zazwyczaj większe niż koszty umieszczenia ciała funkcji w miejscu jej wywołania. Tego typu funkcje zachowują się w sposób zbliżony do makrodefinicji (z tym że respektują wszystkie reguły języka C, w szczególności kolejność operatorów czy kontrolę typów).

Takie funkcje jako argument pobierają adres bufora, na którym mają operować. W naszym przypadku bufor jest zdefiniowany następująco:

```
static CircBuffer LCD_TransBuffer;           // Bufor na transakcje
```

Należy zauważyć, że jest on zdefiniowany jako struktura globalna, z modyfikatorem `static`. Modyfikator ten ogranicza zasięg widoczności zmiennej `LCD_TransBuffer` do pliku, w którym zmienna ta została zdefiniowana (w tym przypadku modyfikator `static` jest niejako przeciwieństwem modyfikatora `extern`). Działanie to ma jedynie znaczenie kosmetyczne — uniemożliwia odwołanie się do bufora za pomocą funkcji, które nie zostały zdefiniowane w miejscu jego definicji. Oprócz funkcji umożliwiających określenie stanu bufora potrzebne są jeszcze funkcje umożliwiające dodanie i pobranie transakcji. Pierwszą funkcją będzie funkcja dodająca transakcję do bufora:

```
bool cbAdd(CircBuffer *cb, LCD_trans *elem)
{
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        if(cbIsFull(cb)) return false;           // Czy jest miejsce w kolejce?
        uint8 t end = (cb->Beg + cb->Count) % LCD_MAXTRANS;
        cb->elements[end] = elem;                // Dodaj transakcję
        ++cb->Count;                             // Liczba elementów w buforze
    }
    return true;                               // Wszystko OK
}
```

Funkcja ta dodaje transakcję (zmienna `elem`) do bufora kołowego. Zanim transakcja zostanie dodana, sprawdzane jest, czy w buforze jest miejsce — jeśli nie, funkcja zwróci `false`. Jeśli dodanie jest możliwe, wartość licznika określającego liczbę transakcji przechowywana w buforze (`Count`) zwiększa się o jeden. Warto zastanowić się też, dlaczego praktycznie całe ciało tej funkcji jest zamknięte w sekcji atomowej (`ATOMIC_BLOCK`). Jedną z cech naszego sterownika ma być możliwość współbieżnego dostępu

do LCD — to znaczy, że żądanie np. zapisu do LCD może wystąpić zarówno w głównej pętli programu, jak i asynchronicznie w przerwaniu. Stąd też wynika, że funkcja dodająca transakcję musi być napisana w sposób zapewniający jej poprawną pracę w sytuacji, gdy zostanie ona przerwana i ponownie wywołana przed zakończeniem działania jej poprzedniej instancji. Umieszczenie ciała tej funkcji w sekcji atomowej nam to zapewnia — dzięki temu bufor transakcji zawsze będzie spójny. Dla zaspokojenia ciekawości warto usunąć sekcję atomową i sprawdzić, jak wtedy będzie zachowywać się ta funkcja i jak wpłynie to na spójność przechowywanych danych.

Drugą niezbędną funkcją jest funkcja odczytująca bufor transakcji i zwracająca transakcję do zrealizowania (lub NULL, jeśli żadnej transakcji nie ma w buforze):

```

LCD_trans *cbRead(CircBuffer *cb)
{
    LCD_trans *elem;
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        if(cbIsEmpty(cb)) return NULL;           // Bufor pusty, nie można zwrócić elementu
        elem = cb->elements[cb->Beg];
        cb->Beg = (cb->Beg + 1) % LCD_MAXTRANS;
        -- cb->Count;                             // Zmniejszamy liczbę elementów pozostałych
                                                // w buforze
    }
    return elem;
}

```

Funkcja ta jest bardzo podobna do poprzedniej — najpierw sprawdza, czy bufor zawiera jakąś oczekującą transakcję (`cbIsEmpty(cb)`), jeśli tak, to zwraca adres opisujący ją struktury i zmniejsza liczbę transakcji oczekujących (pole `Count`). W ten sposób mamy zrealizowaną całą funkcjonalność związaną z obsługą bufora pierścieniowego. Pokazane funkcje są uniwersalne i z bardzo podobnych (lub wręcz tych samych) będziemy korzystać w kolejnych rozdziałach w kontekście np. obsługi transakcji związanych z interfejsami szeregowymi.

Pora przejść do samego sterownika LCD. Jego zadaniem będzie sprawdzenie, czy w buforze oczekują jakieś transakcje do realizacji, a jeśli tak, to ich wykonanie. Ponieważ dane do LCD mogą być wysyłane co określony, minimalny czas, znacznie dłuższy niż czas, jakiego na to zadanie potrzebuje mikrokontroler, do wysyłania kolejnych paczek danych posłużymy się timerem. Timer będzie generował przerwanie nadmiaru (`TCD0_OVF_vect`), które będzie wyznaczało tempo, w jakim kolejne porcje danych będą trafiały do LCD. Z noty katalogowej kontrolera wynika, że minimalny czas trwania wysokiego stanu sygnału *Enable* to 230 ns, a okres tego sygnału nie powinien być krótszy niż 500 ns. Ponieważ nasz dostęp do LCD odbywa się w sposób nieblokujący MCU, więc szybkość wysyłania danych do LCD jest bez znaczenia. Stąd też w pokazanym przykładzie przyjęto, że dane będą wysyłane co 4 μs, co określa stała:

```
#define LCD_ACCESTIME 0.000004
```

Ponieważ komunikacja z LCD odbywa się w sposób nieblokujący, nie zależy nam też na czasie (i tak w stosunku do czasu odświeżania samego LCD jest on bardzo krótki), w efekcie nie ma potrzeby odczytywania flagi busy kontrolera. Dzięki temu w części przypadków możemy w ogóle zrezygnować z sygnału wyboru R/W i wymusić jego stały

poziom logiczny, umożliwiający zapis do LCD. Dzięki temu można zaoszczędzić dodatkowy pin I<sup>0</sup><sup>1</sup>.

Cała logika naszego sterownika jest zawarta w funkcji obsługi przerwania przepelnienia timera (TCDO\_OVF\_vect). Ponieważ LCD jest połączony z MCU w trybie 4-bitowym (dla zaoszczędzenia liczby wykorzystanych pinów I<sup>0</sup>), kolejne 8-bitowe dane muszą być podzielone na dwie tetrazy i wysłane do LCD w kolejności starsza/młodsza tetra. Informacja o aktualnie wysyłanych danych jest umieszczona w zmiennej trans o typie:

```
static struct
{
    uint8_t pos      : 6;           // Pozycja w polu danych aktualnej transakcji
    uint8_t nibble  : 1;           // Która tetra jest wysyłana?
} seq;
```

Dlaczego tak? Dzięki wykorzystaniu struktury i pół bitowych można zaoszczędzić jeden bajt pamięci SRAM. Ze względu na wielkość pamięci obsługiwanej przez kontroler LCD możemy być pewni, że transakcje zawierające więcej niż 63 bajty są raczej nierealne (nie miałyby większego sensu). Stąd też do przechowywania informacji o aktualnie wysyłanym bajcie danych wystarczy 6 bitów — tyle samo zostało wykorzystanych do określenia długości pola danych transakcji w strukturze LCD\_trans. Zaoszczędzony bit można poświęcić na flagę nibble określającą, która tetra (młodsza/starsza) jest wysyłana. Bez pół bitowych flaga ta zajęłaby dodatkowy bajt pamięci.

Przejdźmy do implementacji sterownika. Pierwszą rzeczą, jaką należy sprawdzić, jest to, czy w buforze czeka jakaś transakcja do zrealizowania. Odpowiedzialny za to jest fragment kodu obsługi przerwania nadmiaru timera:

```
if(trans == NULL)           // Nic do zrobienia, sprawdźmy, czy jakaś transakcja oczekuje
{
    trans=cbRead(&LCD_TransBuffer); // Czy jest jakaś oczekująca transakcja?
    seq.pos=0;
    seq.nibble=0;
    if(trans == NULL) TCDO_CTRLA=TC_CLKSEL_OFF_gc; // Nie ma żadnych transakcji — wyłącz timer
}
```

W przypadku gdy żadna transakcja nie oczekuje na realizację, timer jest wyłączany — dzięki temu nie będą zgłaszane kolejne przerwania przepelnienia i MCU nie będzie tracił czasu na ich obsługę. Co prawda wynikający z tego zysk czasowy jest marginalny, lecz ma to istotniejszą konsekwencję — w przypadku usypiania MCU zbędne przerwanie timera nie będzie niepotrzebnie wybudzało procesora. W takiej sytuacji ponowne włączenie timera nastąpi dopiero w funkcji dodającej transakcję do kolejki — a więc w sytuacji, gdy mamy pewność, że istnieje oczekująca transakcja.

Druga część funkcji obsługi przerwania jest związana z realizacją samej transakcji. W pokazanym przykładzie zaimplementowano tylko jeden typ transakcji — żądanie zapisu łańcucha tekstowego pod wskazany adres VRAM (na określonej pozycji LCD). Jeśli zajdzie taka potrzeba, programista może zaimplementować inne polecenia i dodać

<sup>1</sup> Warto zauważyć, że w „klasycznej” obsłudze LCD też nie musimy czytać stanu flagi busy, zamiast tego wystarczy wprowadzać pętle opóźniające. Jednak ponieważ taka obsługa blokuje MCU, nie jest ona zalecana. Wady tej nie ma dostęp transakcyjny, co jest jego dodatkową zaletą.

kod odpowiedzialny za ich obsługę. Zauważmy, że każda transakcja musi zawierać adres początku pamięci wideo, od którego będzie zapisywany tekst. Dzieje się tak dlatego, że transakcja wykonana wcześniej może ustawić wskaźnik zapisu na dowolnej komórce pamięci. W efekcie tekst z kolejnej transakcji byłby wyświetlany od komórki, na której zakończył się zapis poprzedniej transakcji. Takie działanie oczywiście jest niedopuszczalne, stąd też każdy zapis do pamięci wideo poprzedza ustawienie rejestru indeksowego. Problem ten można rozwiązać także w inny sposób — poprzez wprowadzenie tzw. uchwytów (ang. *handlers*), które zawierałyby stan kontrolera LCD. Niemniej to rozwiązanie dodatkowo komplikowałoby nasz sterownik. Struktura danych (zawartych w polu `data`) ma więc następujący format:

- ◆ bajt nr 0 zawiera wartość pozycji, od której należy dokonywać zapisu pamięci wideo — jest to wartość przesyłana do rejestru określającego adres zapisu,
- ◆ kolejne bajty zawierają łańcuch znakowy, który należy umieścić w pamięci CGRAM.

Ponieważ bajt nr 0 zawsze zawiera wartość wpisywaną do rejestrów sterujących kontrolera, podczas jego wysyłania jest ustawiana linia RS sygnalizująca zapis do rejestru. Podczas wysyłania kolejnych bajtów linia ta jest zerowana, w związku z czym zapis odbywa się do pamięci CGRAM.

Kod obsługujący transakcję wygląda następująco:

```

if(trans)
{
    // Jest transakcja do zrealizowania
    uint8_t dat=trans->data[seq.pos];
    if(seq.nibble == 0) dat>>=4; // To trzeba zmienić, jeśli linie danych nie są połączone
                                // z pinami 0 – 3 portu IO
    hd44780_outnibble_nowait(dat & 0x0F, seq.pos != 0); // Zapisujemy rejestr sterujący lub dane
    ++seq.nibble;
    if(seq.nibble == 0) ++seq.pos; // Co drugą tetradę zwiększamy pozycję bufora
    if(seq.pos >= trans->len)
    {
        trans->Ready=true; // Koniec transakcji
        if(trans->SelfDel) free_re(trans); // Zwolnij pamięć transakcji, jeśli tak sobie życzył
                                        // programista
        trans=NULL; // Koniec transakcji
    }
    CLR(OUT, HD44780_E);
}

```

Ta część przzerwania wysyła kolejne tetrazy do kontrolera LCD. Warto zastanowić się nad działaniem sekwencji:

```
++seq.nibble;
```

Ponieważ pole `nibble` jest jednobitowe, operacja inkrementacji prowadzi do zmiany stanu tego pola na przeciwny (pole to ma wartość 0 lub 1). Kiedy zostaną wysłane wszystkie dane przeznaczone dla kontrolera, obsługa transakcji kończy się. Stan ten sygnalizuje nadanie wartości `true` polu `Ready` struktury `LCD_trans`. Dzięki temu program może sprawdzić, asynchronicznie w stosunku do przzerwania, kiedy transakcja uległa zakończeniu. Opcjonalnie jest wykonywana jeszcze jedna funkcja — jeśli pole `SelfDel` ma wartość `true`, to pamięć zarezerwowana na transakcję jest automatycznie

zwalniana. W tym celu wykorzystuje się funkcję `free_re`, której prototyp jest zadeklarowany w pliku `Alloc_safe.h`. Funkcja ta wywołuje funkcję biblioteczną `free`, która zwalnia pamięć przydzieloną dynamicznie na stercie. Ponieważ funkcja ta nie jest funkcją *reentrant*, należy stworzyć odpowiednią „protezę”, opakowując ją w blok atomowy, co zapewnia, że nie zostanie ona ponownie wywołana, zanim jej poprzednia instancja nie zakończy działania. Oczywiście lepszym rozwiązaniem byłoby napisanie własnych funkcji dynamicznej alokacji pamięci, które będą *reentrant*.



Wskazówka

Bardzo ważne jest, aby pole `SelfDel` miało wartość `true` wyłącznie dla transakcji, dla których pamięć została przydzielona dynamicznie funkcją `malloc` (lub jej pochodnymi). W pozostałych przypadkach pole to musi mieć wartość `false` — niestosowanie się do tej reguły spowoduje błąd menedżera pamięci i najprawdopodobniej nieprzewidywalne działanie programu.



Wskazówka

Drugą ważną zasadą jest, aby zmienna zawierająca dane transakcji nie była zmienną automatyczną — zmienne takie są automatycznie niszczone po opuszczeniu zawierającej je funkcji. W rezultacie obszar pamięci zawierający dane transakcji może zostać wykorzystany przez inne zmienne, co spowoduje uszkodzenie zawartych w nim danych.

Pamięć przydzielona transakcji musi być alokowana w sposób dynamiczny (funkcja `malloc_re`), ewentualnie może to być pamięć alokowana dla zmiennej globalnej lub statycznej — zmienne tego typu istnieją przez cały czas życia programu.

Warto też zauważyć, że instrukcje sterujące stanem linii *Enable* kontrolera LCD (`SET(OUT, HD44780_E)` i `CLR(OUT, HD44780_E)`) są rozdzielone wieloma innymi instrukcjami. Dzięki temu generowany impuls dodatni tego sygnału ma czas trwania > 230 ns wymagany w specyfikacji kontrolera. Nawet dla XMEGA taktowanej zegarem 32 MHz 230 ns przekłada się na około 7 instrukcji asemblera — czas, jaki z pewnością upłynie pomiędzy wykonaniem obu powyższych instrukcji sterujących.

Do wysyłania danych do kontrolera została wykorzystana nieco zmodyfikowana funkcja `hd44780_outnibble_nowait`, różniąca się od jej protoplasty `hd44780_outnibble` tylko tym, że nie czeka na sygnał braku zajętości mikrokontrolera (nie realizuje też opóźnień). Nie jest to potrzebne, gdyż kolejne tetrydy są wysyłane w odstępach czasowych określanych przez timer.

Mamy więc zapewnioną obsługę transakcji i możliwość transakcyjnego wyświetlania danych na LCD, musimy jeszcze mieć funkcję, która tworzy odpowiednie transakcje — jest odpowiednikiem funkcji bezpośrednio wysyłającej dane znakowe do LCD. Odpowiednikiem tym jest pokazana poniżej funkcja:

```
bool LCD_PutText_B(uint8_t x, uint8_t y, char *txt, LCD_trans *buf, bool autodel)
{
    buf->cmd=LCD_Text;
    buf->Ready=false;
    buf->SelfDel=autodel; // Czy zwolnić pamięć po zakończeniu transakcji?
    buf->data[0]=HD44780_DDADDR(x+y*0x40); // Ustaw adres w DDRAM
    strcpy((char*)&buf->data[1], txt); // Skopiuj dane tekstowe + NULL
}
```

```

buf->len=strlen(txt) + 1; // Długość tekstu + pozycji + NULL - 1
bool ret=cbAdd(&LCD_TransBuffer, buf);
if((ret==false) && (autodel)) free_re(buf); // Brak miejsca w kolejce
    else TCDO_CTRLA=TC_CLKSEL_DIV1_gc; // Preskaler 1 — odblokuj timer
return ret;
}

```

Funkcja ta buduje transakcję odpowiedzialną za wyświetlenie napisu txt na wskazanej pozycji x, y LCD. Wszystkie dane o transakcji zostaną umieszczone w buforze w pamięci, której adres jest określany wskaźnikiem buf. Pamięć w buforze musi być wystarczająco duża, aby pomieścić całą transakcję. Dodatkowy parametr autodel określa, czy pamięć ta ma być automatycznie zwalniana po obsłużeniu transakcji (true), czy nie (false). Jeśli transakcję udało się umieścić w kolejce, funkcja zwraca true. Warto zauważyć, że w takim przypadku funkcja uruchamia też timer (w naszym przypadku jest to timer TCDO) odpowiedzialny za generowanie przerwania. Funkcja obsługi przerwania wysyła dane do LCD.

Pewną odmianą powyższej funkcji jest funkcja:

```

bool LCD_PutText(uint8_t x, uint8_t y, char *txt)
{
    LCD_trans *trans=malloc_re(sizeof(LCD_trans) + strlen(txt) + 2);
    bool ret=LCD_PutText_B(x, y, txt, trans, true);
    return ret;
}

```

Wykorzystuje ona funkcję LCD\_PutText\_B, z tym że najpierw alokuje dynamicznie pamięć potrzebną na przechowanie tworzonej transakcji.

Mamy komplet potrzebnych funkcji, a więc sprawdźmy, jak całość sprawdza się w praktyce. W pliku *LCD-transact.c* znajduje się kod pokazujący, jak działają nasze transakcje. Aby to było możliwe, należy zainicjalizować kontroler LCD:

```
LCD_init();
```

Wykorzystujemy tu znaną już funkcję inicjalizującą — nie ma potrzeby jej zmieniać, gdyż operacja inicjalizacji LCD jest jednorazowa i nie trzeba wykonywać jej w postaci transakcji. W drugim kroku musimy zainicjalizować wykorzystywany timer i system przerwania:

```

LCD_Timer_init(&TCDO);
PMIC_CTRL1=PMIC_LOLVLEN_bm; // Odblokuj przerwanie niskiego poziomu
sei();

```

Ponieważ dostęp do LCD z pewnością nie jest czynnością krytyczną czasowo, przerwaniom timera nadano najniższy priorytet. Dzięki temu można innym, bardziej krytycznym przerwaniom nadać wyższe priorytety i uzyskać w ten sposób zmniejszenie czasu latencji obsługi przerwania. Sama inicjalizacja timera jest niezwykle prosta:

```

void LCD_Timer_init(TCO_t *tc)
{
    tc->INTCTRLA=TC_OVFINTLVL_LO_gc; // Odblokuj przerwanie nadmiaru timera i nadaj im niski
    // priorytet
    tc->PER=F_CPU*LCD_ACCESSTIME; // Przerwanie co 4 us (niezależnie od F_CPU)
}

```



Sprowadza się ona tylko do odblokowania możliwości zgłaszania przerwania nadmiaru oraz konfiguracji rejestru PER, tak aby przerwania były generowane co ok. 4  $\mu$ s — niezależnie od częstotliwości taktowania CPU. Możemy więc przystąpić do wyświetlenia pierwszego napisu:

```
LCD_PutText(0,0, "Numer:");
```

Powyższa funkcja ma jedną wadę — alokuje dynamicznie pamięć po każdym wywołaniu (pamięć ta jest automatycznie zwalniana po obsłużeniu transakcji), niemniej cały proces jest czasochłonny. Możemy tego uniknąć, alokując pamięć „ręcznie” i wielokrotnie wykorzystując zaalokowany blok pamięci. Ilustruje to poniższy przykład:

```
LCD_trans *buf=malloc_re(sizeof(LCD_trans) + 7);
```

Zmienna `buf` wskazuje na obszar pamięci, który może pomieścić łańcuch 5-znakowy. Tyle wystarczy na potrzeby tego przykładu — na LCD będą wyświetlane kolejne liczby w zakresie 0 – 65 535; najdłuższa ma 5 znaków, stąd też nasz bufor ma taką, a nie inną długość:

```
while(1)
{
    utoa(cnt++, bufstr, 10);
    while(strlen(bufstr)<6) strcat(bufstr, " ");
    LCD_PutText_B(6,0, bufstr, buf, false);
    while(!LCD_IsTransCompleted(buf));
}
```

Do konwersji liczby na łańcuch użyto funkcji bibliotecznej `utoa` (nagłówek `stdlib.h`). Ponieważ długość łańcucha po konwersji jest zmienna (co spowodowałoby pozostawianie na LCD „duchów” po poprzednich konwersjach), w kolejnej linii brakujące znaki uzupełniono spacjami. Następnie tworzona jest transakcja w obszarze pamięci przydzielonym zmiennej `buf`. Ostatnia pętla `while` jest odpowiedzialna za oczekiwanie na zakończenie realizacji transakcji. Po jej zakończeniu pętla jest wykonywana od początku.

Uważny czytelnik z pewnością zauważy, że pamięć przydzielana funkcją `malloc_re` nie jest nigdzie zwalniana — w normalnej sytuacji spowodowałoby to wyciek pamięci. W naszym przykładzie jednak program nigdy się nie kończy, nie ma więc potrzeby zwalniania pamięci wskazywanej przez zmienną `buf`.

Powyższy przykład pokazuje, jak uniknąć blokowania CPU na czas dostępu do wolno działającego urządzenia, jakim jest LCD alfanumeryczny. Jednak to nie wszystkie zalety wynikające z zastosowania transakcji. Drugą ważną zaletą naszego rozwiązania ilustruje przykład znajdujący się w katalogu *Przykłady/LCD-alfa/LCD-transact-int*. Poważnym ograniczeniem funkcji odpowiedzialnych za dostęp do LCD w sposób klasyczny była niemożność ich wywoływania z funkcji obsługi przerw — funkcje obsługi LCD nie były *reentrant*<sup>2</sup>. Nasz system oparty na transakcjach nie ma tych wad. Aby to zilustrować, został stworzony prosty przykład — tak jak poprzednio w pętli głównej programu są wyświetlane stale kolejne liczby, ale jednocześnie wyko-

<sup>2</sup> Poza tym blokowały CPU na długi czas, a jak pamiętamy, funkcje obsługi przerw powinny być wykonywane możliwie krótko.

rzystano drugi timer (TCD1), w którego funkcji obsługi przerwania przepełnienia jest wyświetlany licznik tych przerw:

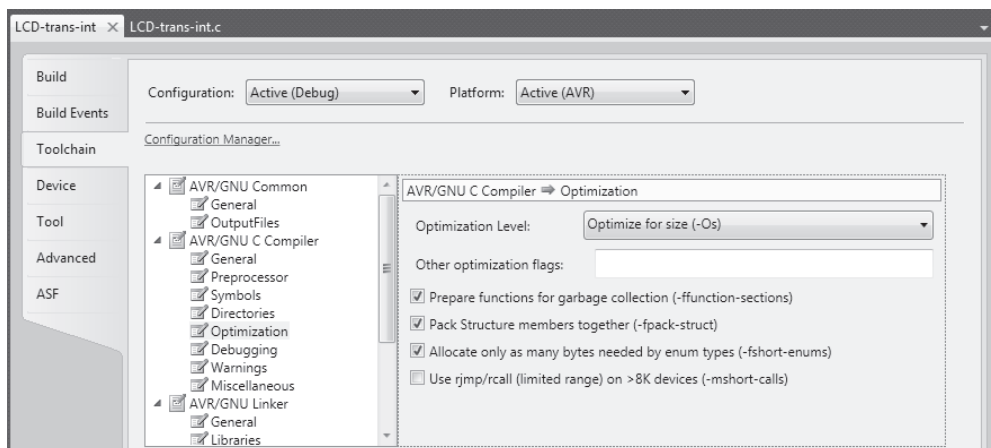
```
ISR(TCD1_OVF_vect)
{
    static char int_buf[sizeof(LCD_trans) + 7];
    static uint16_t cnt;
    char bufstr[6];

    utoa(cnt++, bufstr, 10);
    while(strlen(bufstr)<6) strcat(bufstr, " ");
    LCD_PutText_B(6,1, bufstr, (LCD_trans*)&int_buf, false);
}
```

Jak widzimy, w sposób asynchroniczny, z dwóch miejsc w programie możemy uzyskać dostęp do LCD, a każda funkcja „ma wrażenie”, że ma LCD na wyłączność. Przy okazji warto przeanalizować sposób, w jaki jest rezerwowana pamięć na transakcję w przerwaniu. Oczywiście można by wykorzystać dynamiczną alokację pamięci, lecz jak wspomniano, taka alokacja jest czasochłonna. Stąd też w pokazanym przykładzie wykorzystano tablicę statyczną `int_buf` o rozmiarze umożliwiającym przechowanie kompletnej transakcji. Dzięki temu, że zmienna `int_buf` jest zmienną statyczną, nie jest ona niszczone po zakończeniu funkcji obsługi przerwania. Gdyby pominąć słowo kluczowe `static`, zmienna ta byłaby niszczone, a na LCD najprawdopodobniej zaczęłyby się pojawiać przypadkowe treści. Ponieważ pamięć naszego bufora nie była alokowana dynamicznie, nie może ona zostać zwolniona w naszym sterowniku LCD, stąd też ostatni parametr wywołania funkcji `LCD_PutText_B` musi mieć wartość `false`.

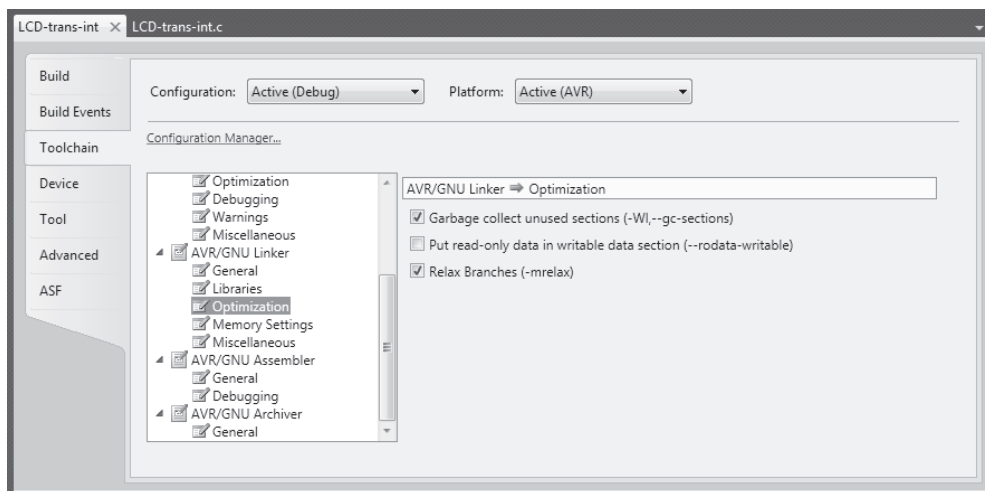
## Optymalizacja

Przy okazji tego i poprzedniego przykładu warto zastanowić się jeszcze nad jedną kwestią — optymalizacji. W podanych przykładach wykorzystano wcześniej omówione funkcje dostępu do wyświetlacza LCD znajdujące się w plikach `hd44780.c` i `hd44780.h`. Większość z zawartych tam funkcji nie jest wykorzystywana, jednak kompilator mimo wszystko umieszcza ich kod w pliku wynikowym — dzieje się tak, ponieważ nie może on „stwierdzić”, czy funkcje te nie są wykorzystywane w innych fragmentach programu. Pamiętajmy, że kompilator kompiluje poszczególne pliki źródłowe osobno i dana jego instancja nie „wie”, co znajduje się w innych plikach (innych jednostkach kompilacji). Taką całościową wiedzę ma dopiero konsolidator (linker), który łączy stworzone przez kompilator pliki obiektowe (o rozszerzeniu `.o`) w plik wynikowy. Linker jednak „nie wie”, która z funkcji programu została użyta, a która nie — informacje te ma wyłącznie kompilator. W ten sposób powstaje typowe błędne koło, w efekcie nasze nieużywane funkcje nie są usuwane (linker zostawia je „na wszelki wypadek”) i zajmują niepotrzebnie pamięć. Czy można temu jakoś zaradzić? Oczywiście tak. W tym celu trzeba połączyć siły kompilatora i linkera, co zapewniają odpowiednie opcje kompilacji. We właściwościach projektu (*Project/Properties* lub *Alt+F7*) wybieramy zakładkę *Toolchain*, a następnie *AVR/GNU C Compiler* i *Optimization*. Znajdziemy tam pole wyboru o nazwie *Prepare functions for garbage collection* (rysunek 22.4).



**Rysunek 22.4.** Aby dać linkerowi możliwość usuwania nieużywanych funkcji, należy skompilować program z wybraną opcją `-ffunction-sections`, która spowoduje, że każda funkcja programu znajdzie się we własnej sekcji

Zaznaczenie tej funkcji spowoduje umieszczenie każdej funkcji w osobnej sekcji — samo w sobie nie wpłynie to na skrócenie kodu wynikowego, ale jest warunkiem niezbędnym dla podjęcia stosownych działań przez linker. Aby wyeliminować nieużywane funkcje, należy zaznaczyć jeszcze w gałęzi *AVR/GNU Linker/Optimization* pole *Garbage collect unused sections* (rysunek 22.5).



**Rysunek 22.5.** Zaznaczenie opcji *Garbage collect unused sections* (`-Wl,--gc-sections`) spowoduje, że nieużywane funkcje zostaną usunięte z kodu wynikowego, w efekcie kod programu ulegnie skróceniu

Po zaznaczeniu tych opcji program zostanie skrócony o kilkaset bajtów. Oczywiście jeśli wszystkie funkcje byłyby wykorzystane, nie uzyskalibyśmy żadnego skrócenia kodu. Niemniej w sytuacji, gdy używa się bibliotek, zazwyczaj przynajmniej część funkcji nie jest wykorzystywana, warto więc skorzystać z tej możliwości optymalizacji.



Jeśli mamy do czynienia z prekompilowaną biblioteką (*lib*), to musi ona być skompilowana z zaznaczoną opcją *Prepare functions for garbage collection (-function-sections)*. Alternatywą jest umieszczenie każdej funkcji biblioteki w osobnym pliku (tak jak w przypadku bibliotek AVR-libc).

Na ekranie wyboru opcji optymalizacji linkera mamy jeszcze jedną przydatną opcję — *Relax Branches (-mrelax)*. Dla mikrokontrolerów, które mają ponad 8 kB pamięci FLASH, instrukcje skoków są generowane jako tzw. instrukcje długie — umożliwiają one skok do komórki pamięci FLASH o dowolnym adresie. Jednak ceną za to jest wydłużenie instrukcji — ich kod jest dłuższy, podobnie jak czas wykonania. Nie zawsze jednak takie długie instrukcje muszą być wykonywane — jeśli miejsce docelowe skoku leży blisko (w zasięgu instrukcji skoków względnych), to można w kodzie wynikowym zamienić instrukcje dłuższe na krótsze, co przynosi oszczędności w postaci skrócenia kodu wynikowego. Ponieważ dopiero linker „wie”, jak poszczególne fragmenty kodu są rozmieszczone w pamięci, takiej podmiany można dokonać na etapie linkowania programu, co zapewnia powyższa opcja. Stąd też warto to pole wyboru zaznaczyć.

# Skorowidz

## A

ABI, Application Binary Interface, 25  
AC, Address Counter, 50  
ADPCM, 482  
adres  
  bazowy, 377  
  bufora, 61  
  komórki, 84  
  pułapki, 202  
ALE, Address Latch Enable, 368  
alfanumeryczne wyświetlacze, 43  
algorytm ADPCM, 492  
algorytmy stratne, 495  
animacje, 57  
antialiasing, 325  
architektury procesorów XMEGA, 352  
assembler, 23  
  dyrektywy, 40  
  funkcje języka C, 41  
  obsługa przerwania, 39  
asynchroniczne wywołanie funkcji, 101  
Atmel Studio, 15

## B

bajty kalibracyjne, 569  
biblioteka  
  FATFS, 131, 139, 176, 180, 517  
  libcallback-lib.a, 99  
  PetitFS, 144, 149  
bit  
  ADC\_CH0START\_bm, 236  
  IMPMODE, 244  
bitrate, 442, 512  
blokowanie przerwania, 219  
błąd, 16, 19  
  kwantyzacji, 463  
  offsetu, 230  
  wzmocnienia ADC, 230  
BOD, 223  
bps, bits per second, 12

budowa  
  karty, 152  
  linii obrazu, 394  
  pamięci SDRAM, 374  
  pliku stymulacji, 192  
  ramki PAL, 412  
  sygnału PAL, 411  
  wtyczki, 544, 545  
bufor, 78  
  FIFO, 497  
  na transakcje, 61  
  transakcji, 62  
buforowanie wyjścia, 294

## C

CAS, Column Address Strobe, 375  
CCFL, 44  
charakterystyka  
  przetworników, 222  
  termistora, 264  
  trybów pracy, 232  
CID, Card Identification, 169  
CKE, Clock Enable, 375  
CLKIN, 421  
COG, Chip on Glass, 310  
CRC, 158, 178  
CRC16, 178  
CRC7, 164  
CS, Chip Select, 361, 375  
CSD, Card Specific Data, 172  
czas  
  dostępu do SDRAM, 424  
  ładowania kondensatora, 244  
  propagacji, 281  
  próbki, 260  
  symulacji, 209  
czcionki, 322  
częstotliwość  
  odchylenia pionowego, 428  
  próbki, 244, 441, 444, 530  
  przebiegu, 504

częstotliwość  
 PWM, 465  
 taktowania koprocatora, 549  
 taktowania procesora, 209  
 taktowania rdzenia, 18  
 czyszczenie pamięci, 318

## D

DAC, Digital to Analog Converter, 293, 447, 451  
 buforowanie wyjścia, 294  
 generowanie przebiegów, 301  
 jitter, 303  
 kalibracja, 306  
 napięcie referencyjne, 295  
 system zdarzeń, 297  
 taktowanie, 296  
 tryb dwukanałowy, 301  
 wykorzystanie DMA, 298  
 wyzwalanie konwersji, 305  
 datalogger, 261, 265, 275  
 DD, Device Density, 86  
 debugger, 183  
 programowy, 190  
 sprzętowy, 185, 187  
 decymacja, 259  
 DEBUG, 219  
 XMEGA, 260  
 definicja symboli globalnych, 36  
 definiowanie własnych znaków, 56  
 deklaracja zapowiadająca, 103, 337  
 dekodery DTMF, 548, 556  
 DMA, 254, 298, 402, 458, 518, 523  
 dostęp do  
 danych, 358, 359, 568  
 danych binarnych, 356, 360  
 danych wielobajtowych, 31  
 DataFLASH, 137  
 EEPROM, 128  
 FAT, 136  
 komórki, 375  
 LCD, 59, 62, 66, 68  
 operandu, 30  
 pamięci, 81, 384, 386, 387, 561  
 pamięci SDRAM, 424  
 pliku blokowy, 141  
 portów IO, 30  
 rejestrów IO procesora, 36  
 rejestrów układu, 499  
 SPI, 95  
 sygnatury użytkownika, 565, 567  
 wskaźników, 32  
 DQM, Data Mask, 375  
 DSP, Digital Signal Processor, 535

DTMF, 535, 553  
 dyktafon, 529  
 dynamiczna alokacja pamięci, 516  
 dyrektywa `.global`, 38  
 dyrektywy asemblera, 40  
 działanie  
 DMA, 255  
 komparatora, 280  
 timerów, 400  
 dźwięk, 443

## E

EBI, External Bus Interface, 361  
 EEPROM, 207  
 eliminacja jitteru, 24  
 emulacja RS232, 139

## F

FAT, File Allocation Table, 128, 131  
 FAT32, 128  
 FATFS, 180  
 filtr  
 dolnoprzepustowy, 491  
 drugiego rzędu, 470  
 pierwszego rzędu, 469  
 trzeciego rzędu, 471  
 filtrowanie sygnału PWM, 468  
 firmware, 565  
 FLASH, 207  
 format  
 odpowiedzi, 159  
 polecenia, 158  
 protokołu, 157  
 funkcja  
`_delay_ms`, 17, 219  
`c_zero`, 42  
`callback`, 106, 341  
`callback()`, 99  
`DataFLASH_CSEnable`, 100  
`DF_FinishBufWrite()`, 269  
`disk_initialize`, 136  
`disk_ioctl`, 181  
`disk_status`, 136  
`disk_write`, 138  
`DMA_SPI_init`, 104  
`DTMF_GetCode`, 553  
`free_re`, 65  
`GPIO0_init`, 507  
`hd44780_outcmd`, 51  
`hd44780_outnibble_nowait`, 65  
`LCD_defchar_P`, 57

Menu\_Click\_Func, 515  
 Menu\_Free, 516  
 Menu\_Recorder, 532  
 Menu\_Run, 527  
 MS\_CreateRegEntry, 117  
 MS\_GetDirPos, 125  
 MS\_GetRegItem, 115  
 MS\_WritePages, 116, 118  
 OSC\_wait\_for\_rdy, 473  
 RecMenu\_StartStop, 532  
 sprintf, 142  
 strtok, 124  
 sysclk\_init(), 200, 209  
 TMF\_memcpy\_PF, 454  
 udi\_cdc\_getc(), 123  
 udi\_cdc\_is\_rx\_ready(), 123  
 VS1003\_Reset, 506  
 funkcje  
   analizujące, 556  
   biblioteczne, 50  
   biblioteki FATFS, 143  
   języka C, 41  
   kontrolera ST7565R, 313  
   opóźniające, 17  
   opóźnień, 219  
   reentrant, 524  
   specjalne kontrolera, 319  
   sterujące, 314  
   zapisu, 133  
 fusebit, 557  
   BODACT, 560  
   BODLEVEL, 560  
   BOOTRST, 559  
   DVSDON, 560  
   EESAVE, 561  
   JTAGEN, 558  
   RSTDISBL, 559  
   SUT, 559  
   TOSCSEL, 559  
 fusebity w AVR-libc, 571

## G

generator obrazu composite, 426  
 generowanie  
   dźwięku, 461  
   kolorowego obrazu, 437  
   obrazu wideo, 391  
   przebiegów, 301  
   przerwań, 285  
   sygnału composite, 409  
 głębia koloru, 326  
 gniazdo  
   karty SD, 156  
   VGA, 396

## H

histereza, 281

## I

IDE, 11  
 IMA ADPCM, 483  
 informacje  
   o kolorze, 431  
   o obrazie, 434  
 inicjalizacja  
   karty, 161  
   karty SDHC, 167  
   karty SDSC, 167  
   karty SDXC, 167  
   licznika, 419  
   pamięci, 119  
   USART, 505  
 instrukcja  
   icall, 42  
   movw, 35  
   nop, 13, 24  
 interfejs  
   SCI, 498  
   SDI, 498  
   SPI, 71, 79  
   RS232, 120  
   UART, 123  
   USART, 71, 73  
 interpolacja, 259  
 interpolacja w XMEGA, 260  
 inwersja obrazu, 319

## J

jednostki, 12  
 jitter, 24, 303, 420

## K

kalibracja  
   ADC, 229  
   DAC, 306, 564  
   offsetu, 237  
 kanał DMA, 82, 522  
 karta, 154, 158  
   mikroSD, 152  
   SD, 151, 152  
   SDHC, 151, 152, 160  
   SDSC, 160  
   SDXC, 151, 160

- karty pamięci, 151, 152, 157
    - biblioteka FATFS, 180
    - budowa, 152
    - format polecenia, 158
    - inicjalizacja, 161, 167
    - komunikacja, 156
    - odczyt danych, 173
    - opcje konfiguracyjne, 181
    - rejestry specjalne, 169
    - tryby pracy, 153
    - zapis danych., 176
    - zasilanie, 154
  - kod
    - assemblerowy, 23, 24
    - C, 23
    - dostępu, 24
  - kodowanie
    - $\mu$ -law, 481
    - ADPCM, 483
    - Dialogic ADPCM, 494
    - informacji, 166, 426
    - koloru, 425
    - znaków, 536
  - kody
    - formatów audio, 501
    - zdarzeń, 545
  - kolejność bajtów, 533
  - kolor, 424, 430
  - kolorowy obraz, 437
  - komparator, 422
    - analogowy, 279, 280
    - analogowy XMEGA, 282
    - okienkowy, 284
  - kompilacja plików binarnych, 350
  - kompilator avr-gcc, 11, 359
  - kompresja
    - ADPCM, 482, 486
    - dźwięku, 480
    - IMA ADPCM, 482
  - komunikacja z koprocesorem, 504
  - konfiguracja
    - 4-portowa, 364
    - biblioteki, 133
    - debugera, 185
    - DMA, 459, 522
    - fusebitów, 570
    - kontrolera DMA, 257, 452
    - komparatorów, 285
    - lockbitów, 570
    - LPC, 371
    - pamięci SDRAM, 381
    - PetitFS, 149
    - pinu IO, 225
    - portów IO, 362
    - preskalera, 452
    - projektu, 186
    - przerwań, 219
    - pułapki, 203
    - rejestru EVCTRL, 248
    - rejestru PER, 67
    - SPI, 75
    - sygnału CS, 377
    - typu czcionki, 324
    - urządzeń IO, 189
    - USART, 76
    - zegara, 383
  - konsolidacja, 547
  - kontroler
    - DMA, , 254, 298, 402, 458, 518, 523
    - KS108, 310
    - ST7565R, 310–315, 321
  - konwersja
    - DAC, 305
    - sygnałów, 396
    - wyzwalana, 306
  - konwerter
    - cyfrowo-analogowy, 426
    - DAC, 427
  - koprocesor
    - mp3, 496
    - VS1003B, 10
  - korekcja fazy, 466
  - kwantyzacja, 463
- L**
- lampa CCFL, 44
  - LE, Latch Enable, 433
  - licznik, 419
    - cykli, 209
    - TCC1, 400
    - timera, 20
  - linie IO, 50
  - linie sygnałowe
    - MISO, 72, 164
    - MOSI, 72
    - SCK, 72
    - SS, 72
  - linker, 69
  - lista
    - mnemoników, 24
    - modyfikowanych rejestrów, 32
  - lockbity, 557, 561
  - lockbity w AVR-libc, 570
  - LPC, Low Pin Count, 371
  - LPCM, 443, 556
  - LSB, Least Significant Bit, 228



## Ł

łańcuchy formatujące, 212  
 łączenie plików, 352

## M

magistrala pamięci zewnętrznej, 361  
 magnetofon cyfrowy, 528  
 makro \_SFR\_IO\_ADDR(), 39  
 makrodefinicja  
   ASSERT, 214  
   HD44780\_CGADDR(), 53  
   HD44780\_CLR, 53  
   HD44780\_DDADDR(), 53  
   HD44780\_DISPCTL(), 52  
   HD44780\_ENTMODE(), 52  
   HD44780\_FNSET(), 52  
   HD44780\_HOME, 53  
   HD44780\_SHIFT(), 51  
   LOCKBITS, 571  
 mapowanie sygnałów, 366  
 mapy bitowe, 343  
 marker czasowy, 268  
 maski bitowe, 329  
 matryce, 49  
 menu, 335  
 menu piktogramowe, 342  
 mierniki temperatury, 262  
 mikrokontroler  
   ARM, 9  
   AVR, 9  
   PIC, 9  
   XMEGA128A3U, 10  
 MISO, 72  
 moc wydzielana, 265  
 model psychoakustyczny, 495  
 moduł  
   alfanumeryczny, 45  
   NVM, 568  
   Xmega eXploreGO, 10  
   Xplained ATXMEGA A3BU, 10  
   Xplained ATXMEGA-A1, 10  
   Xplained XMEGA128A1, 447  
   Xplained XMEGA256A3, 94  
   Xplained XMEGA256A3BU, 97, 143, 266  
 modyfikator, 30  
   const, 85  
   volatile, 28, 273  
 monitory CRT, 395  
 monochromatyczne wyświetlacze graficzne, 309  
 monochromatyczny  
   tryb graficzny, 416  
   tryb tekstowy, 413

MOSI, 72  
 mp3, 495  
 Msp/s, Megasamples per second, 12  
 multiplekser wejściowy, 226, 283  
 multipleksowanie adresów, 368, 369  
 muzyka, 439

## N

nadawanie kodów DTMF, 540  
 nadpróbkowanie, 258  
 nagłówek kontenera wav, 500  
 nagrywanie mowy, 486  
 najmniej znaczący bit, 228  
 napięcie  
   offsetu, 280  
   referencyjne, 223, 295  
   zasilania, 239

## O

obracanie obrazu, 319  
 obraz, 319  
   composite, 426  
   kolorowy, 430  
   video, 391  
 obróbka dźwięku, 443, 445  
 obsługa  
   FAT, 131, 143  
   klawiatury matrycowej, 24  
   LCD, 58, 63, 67  
   menu, 337  
   przerwania, 37, 38, 458  
   przerwania DMA, 525  
   przerwań w assemblerze, 39  
   przycisków, 272  
   wyświetlaczy alfanumerycznych, 45  
 odczyt  
   blokowy, 140  
   danych, 173, 358  
 odpowiedź, response, 159  
   R1, 159  
   R2, 159  
   R7, 160  
 odświeżanie pamięci, 376  
 odtwarzanie  
   dźwięku, 472  
   muzyki, 447, 513, 518, 521  
 offset, 238  
 offset ADC, 230  
 okno  
   Breakpoints, 202  
   Call stack, 209  
   dezasemblera, 210

okno  
 IO View, 207  
 podglądu zmiennych, 211  
 wyboru czcionek, 324  
 OLED, 46  
 opcje  
 kompilatora, 35  
 zapisu dźwięku, 445  
 operacje  
 na buforach, 91  
 odczytu, 173  
 zapisu, 176  
 zapisu strony, 92  
 operand, 29  
 operator  
 .ascii, 41  
 .asciz, 41  
 .byte, 41  
 .section, 41  
 opóźnienia, 13, 18  
 opóźnienie zmienne, 16  
 optymalizacja, 16, 68  
 optymalizator, 26  
 organizacja pamięci, 77  
 organizacja pamięci VRAM, 316  
 oscylator, 287  
 OSD, On-Screen Display, 417  
 ostrzeżenie, 16  
 oszczędzanie energii, 94

## P

pakiet WinAVR, 48  
 pamięć  
 CGRAM, 56, 57, 64  
 DataFLASH, 10, 77, 81, 113, 269  
 DDRAM, 49  
 EEPROM, 111  
 FLASH, 54, 124, 207  
 RAM, 49  
 SDRAM, 372  
 SRAM, 63, 124, 329, 366, 371  
 VRAM, 314, 317  
 PCLK, 421  
 PCM, 443  
 PetitFS, 144, 149  
 pętla  
 for, 13  
 while, 201  
 pierwsza linia obrazu, 321  
 piksel, 430  
 piktogram, 342  
 pin IO, 225

plik  
 Alloc\_safe.h, 65  
 DataFLASH.c, 83  
 DataFLASHIO.c, 136  
 defines.h, 50  
 delay.h, 14  
 diskio.h, 132, 136  
 ff.h, 132  
 ffconf.h, 132, 181  
 hd44780.h, 49  
 integer.h, 132  
 io.h, 30  
 LCD\_trans.c, 61  
 LCD-transact.c, 66  
 Makefile, 14  
 Menu.c, 19  
 nothing.o, 541  
 outfile.txt, 547, 551  
 pff.h, 149  
 SPI.h, 104  
 string.h, 113  
 stymulacji, 191, 192  
 pliki  
 .mp3, 495  
 .elf, 351  
 .fon, 323  
 .o, 24  
 .S, 24, 34  
 binarne, 350, 355  
 csv, 276  
 dźwiękowe, 440  
 obiektowe, 352  
 wav, 500  
 WMA, 512  
 z czcionkami, 327  
 z danymi, 349  
 podbijacz napięcia, 315  
 podgląd  
 pamięci, 207  
 stosu wywołań, 209  
 zmiennych, 211  
 znaków, 325  
 podłączenie  
 fototranzystora, 262  
 karty SD, 155  
 LCD z kontrolerem, 311  
 modułu, 50  
 pamięci, 362, 369, 370, 373  
 termistora, 262  
 układu LM35, 251  
 wyjść XMEGA, 398  
 podświetlenie  
 CCFL, 46  
 OLED, 46

- podwójne buforowanie, 255, 332, 457
- polecenia
  - kasowania strony, 81
  - konfiguracyjne pamięci, 82
  - kontrolera ST7565R, 315
  - odczytu bufora, 82, 89
  - odczytu pamięci, 81, 88
  - SDRAM, 376
  - transferu, 91
- polecenie
  - ACMD41, 168
  - cd, 181
  - CID, 178
  - CMD0, 165
  - CMD8, 167
  - CSD, 178
  - date, 276
  - dir, 127, 181
  - dump, 276
  - erase, 276
  - get, 276
  - HD44780\_CLR, 51
  - Init, 178
  - mkdir, 181
- połączenie
  - DMA ze SPI, 96
  - dwóch komparatorów, 285
  - koprocesora mp3, 497
  - LCD z mikrokontrolerem, 48
  - wyjścia XMEGA, 411
  - XMEGA z PC, 121
  - z telewizorem, 431
- pomiar
  - napięcia zasilania, 239
  - temperatury MCU, 241
- port IO, 30, 362
- pozycjonowanie grafik, 345
- półobraz, 410
- precyzja pomiarów, 260
- preskaler, 478
- preskaler ADC, 243
- program
  - Atmel Data Visualizer, 275
  - Audacity, 443
  - avr-objcopy, 350
  - dataloggera, 265
  - LCD Image Converter, 323
  - makelodingtable.exe, 543
  - Realterm, 127, 554
  - riffstrip, 483
  - SoX, 445
  - Tracepoint, 208
- programowanie sygnatury użytkownika, 566
- protokół master-slave, 157
- prototypy, 136
- próbki dźwiękowe, 447
- próbkowanie
  - 8-bitowe, 300
  - nieliniowe, 480
- prymitywy graficzne, 330
- przebieg PWM, 468
- przebiegi, 289, 303
- przechwytywanie danych, 276
- przemiatanie wejść, 253
- przerwanie, 218, 254, 286, 403
  - DMA, 522
  - komparatora, 286
- przestrzeń adresowa procesora, 378
- przesyłanie
  - danych, 498
  - polecień, 498
- przetaktowanie procesora, 436
- przetwornik
  - ADC, 221
  - analogowo-cyfrowy, 221
  - cykliczny, 222
  - potokowy, 222
- przyciski, 272
- pseudooperatory, 41
- pułapki, breakpoints, 197
- pułapki warunkowe, 205
- punkty śledzenia, 203
- PWM, 461, 462, 472

## R

- RAS, Row Address Strobe, 375
- redukcja poboru energii, 242
- regulacja kontrastu, 321
- rejestr, 32, 34
- rejestr AC, 50
  - adresu wtyczki, 513
  - BASEADDR, 377
  - CID, 169
  - CMP, 251
  - CSD, 172
  - czasu utworu, 512
  - DATA, 83
  - EVCTRL, 248
  - formatu audio, 512
  - identyfikacyjny pamięci, 87
  - INTFLAGS, 252
  - kontroli basów, 510
  - kontroli głośności, 513
  - MUXCTRL, 227
  - OCR, 168
  - PER, 67
  - porównania, 249

rejestr AC  
 SCI\_CLOCKF, 511  
 stanu komparatora, 287  
 stanu pamięci, 86  
 stanu układu, 510  
 trybu pracy, 508  
 WINCTRL, 285

rejestry  
 dostępu do pamięci RAM, 513  
 GPIOR, 502  
 IO procesora, 36  
 mikrokontrolera, 25  
 procesora, 35  
 specjalne, 94  
 specjalne karty, 169  
 układu VS1003B, 508

relacja rodzic – dziecko, 336

reset układu, 534

rozdzielczość  
 LCD, 329  
 przetwornika, 231  
 PWM, 463, 477

rozszerzenie HiRes, 477, 479

równanie Steinharta-Harta, 263

rysowanie map bitowych, 343

**S**

schemat układu dataloggera, 267

SCI, Serial Command Interface, 498

SCK, 72

SD, Secure Digital, 152

SDI, Serial Data Interface, 498

separator synchronizacji, 422

separowanie impulsów synchronizacji, 422

shieldy, 10

skaler, 422

skaler napięcia, 283

skrypt makefile, 353

słowo kluczowe  
 asm, 27  
 extern, 37  
 static, 68

specyfikatory lokalizacji zmiennej, 213

sps, samples per second, 12

SS, Slave Select, 72

standard PAL, 410

standard VGA, 393

sterowanie  
 dataloggerem, 275  
 sygnałem, 379

sterownik. 121  
 HD44780, 49, 56  
 LCD, 49, 62

stos wywołań, 209

struktura  
 FATFS, 139  
 Handle, 118  
 LCD\_trans, 60, 63  
 menu, 338  
 rejestru, 86  
 SimpleFSEntry, 113, 116  
 SimpleFSHandle, 113, 118, 140  
 submenu, 527  
 systemu plików, 112

suma  
 bitowa, 571  
 kontrolna, 178

sygnał  
 composite, 409  
 composite video output, 430  
 CS, 377  
 DREQ, 497  
 H-Sync, 396  
 synchronizacji, 423  
 V-Sync, 396  
 zegarowy CLK, 374

sygnały  
 interfejsu EBI, 363  
 magistrali EBI, 364  
 sterujące, 375

sygnatura  
 produkcyjna procesora, 563  
 użytkownika, 565

symbol  
 \_\_DELAY\_ROUND\_CLOSEST\_\_, 17  
 \_\_DELAY\_ROUND\_DOWN\_\_, 17  
 \_FS\_FAT12, 149  
 \_FS\_FAT32, 149  
 \_FS\_MINIMIZE, 133  
 \_FS\_READONLY, 133  
 \_FS\_RPATH, 135  
 \_FS\_TINY, 133  
 \_MAX\_SS, 135  
 \_MULTI\_PARTITION, 135  
 \_USE\_DIR, 149  
 \_USE\_FASTSEEK, 134  
 \_USE\_FORWARD, 134  
 \_USE\_LABEL, 134  
 \_USE\_LFN, 134  
 \_USE\_LSEEK, 149  
 \_USE\_MKFS, 134  
 \_USE\_READ, 149  
 \_USE\_STRFUNC, 134  
 \_USE\_WRITE, 149  
 \_VOLUMES, 136  
 F\_CPU, 14, 15, 18  
 increment, 36  
 USE\_BUSY\_FLAG, 50

symbole  
   globalne, 36  
   w asemblerze, 28  
   zewnętrzne, 39  
 symulacja, 198, 209  
 symulator, 190  
 synchronizacja  
   pamięci, 331  
   pionowa, 395  
   pozioma, 394  
 system  
   plików, 111, 128  
   przerwań, 404  
   zdarzeń, 245  
 szablon  
   font\_XMEGA.tmpl, 326  
   image\_XMEGA.tmpl, 343  
 szum, 463  
 szybkość przesyłania danych, 126

## Ś

śledzenie wiązki, 333  
 środowisko  
   graficzne, 11  
   programistyczne, 11  
   VSIIDE, 542  
   WinAVR, 47

## T

tabela FAT, 130  
 tablica  
   framebuf, 402, 408  
   plugintbl, 552, 555  
   tekst, 208  
   z kodem wynikowym, 543  
 taktowanie  
   DAC, 296  
   mikrokontrolera, 473  
 technologia OLED, 44  
 temperatura mikrokontrolera, 239, 241  
 termistor, 262, 290  
   NTC, 263  
   PTC, 263  
 termometr  
   LM35, 249, 250  
   z alarmem, 251  
 termostat, 290  
 test układu VS1003B, 503  
 testowanie obrazu, 319  
 timer, 16, 62  
   TCC0, 20

  TCC1, 399  
 transakcja, 59, 62, 65  
 transakcje, 17, 103–107  
 transakcyjna obsługa LCD, 58  
 transakcyjny dostęp do SPI, 95  
 transfer, 91  
   danych, 97  
   wyników, 254  
 transmisja, 72, 83  
 tryb  
   bez znaku, 234  
   ciągłej konwersji, 252  
   czterobitowy, 153  
   dwukanałowy, 301  
   graficzny, 416  
   graficzny VGA, 407  
   idle, 419  
   jednobitowy, 153  
   LPC, 371  
   monochromatyczny, 397, 407  
   oszczędzania energii, 94, 306  
   pracy multipleksera, 226  
   pracy SPI, 157  
   różnicowy, 232  
   single slope, 312  
   SPI, 153  
   tekstowy, 413  
   tekstowy VGA, 397  
   z pojedynczym wejściem, 234  
   ze znakiem, 235  
 tryby  
   generowania PWM, 467  
   pracy karty, 153  
   pracy przetwornika, 231  
   pracy SPI, 74  
   pracy układu, 508  
 twierdzenie Nyquista, 258  
 tworzenie  
   adresu, 84  
   nakładek, 417  
   obrazu, 335  
   tablicy, 543  
 typ  
   \_\_uint24, 119  
   karty, 160  
   niekompletny, 60  
   wyliczeniowy, 114  
 typy  
   operandów, 29  
   w AVR, 34

**U**

uchwyt, handle, 64, 113  
 układ  
   AD725, 427, 431  
   buforowy, 294  
   CS, 361  
   DAC, 284  
   dataloggera, 267  
   detekcji awarii zasilania, 559  
   generujący obraz, 393  
   HiRes, 477  
   ICL7660, 47  
   LM35, 250, 251  
   S&H, 293  
   sterowania sygnałem, 379  
   USART, 401  
   VS1003B, 496, 503, 508, 530, 538, 556  
   VS100XX, 529  
   wzmacniania sygnału, 227  
 układy peryferyjne, 9  
 uruchomienie komparatora, 287  
 urządzenie  
   master, 72  
   slave, 72  
 uśrednianie, 259  
 utrata danych, 141  
 użycie wstawki assemblerowej, 33

**V**

VGA, 393

**W**

wartości kalibracyjne  
   ADC, 564  
   oscylatorów, 564  
   temperatury, 564  
 warunkowa kompilacja, 36  
 WE, Write Enable, 375  
 wejście composite video input, 425  
 wektor przerwań, 38, 257  
 wideo, 391  
 wielkość pamięci, 86, 377  
 włączenie DAC, 299  
 wskaźnik na strukturę, 27  
 wskaźniki, 32  
 wstawki assemblerowe, 24, 33  
 wtyczka, 535, 541, 544  
   Sine/DTMF Generator, 537  
   VGA, 396  
 wybór częstotliwości PWM, 465, 474  
 wykorzystanie  
   DMA, 254, 518

EuroSCART, 428  
 PWM, 461  
   rejestrów, 34  
 wyliczanie predyktora, 489  
 wyświetlacze  
   alfanumeryczne, 43, 45  
   graficzne, 43, 309  
   LCD, 43  
   OLED, 46  
 wyświetlanie  
   literału, 213  
   map bitowych, 344  
   obrazu, 333, 346, 406  
   obrazu VGA, 393  
 wywołania zwrotne, callbacks, 98  
   asynchroniczne, 101  
   synchroniczne, 101  
 wyzwalanie konwersji, 245  
 wzmacniacz, 446  
 wzmacniacz operacyjny, 279  
 wzmacnianie sygnału, 227

**X**

XMEGA, 9, 18

**Z**

zapis  
   bufora, 269  
   danych., 176  
   dźwięku, 440, 556  
 zasilanie  
   karty, 154  
   pamięci, 109  
 zatrzask, 368  
 zatrzask 74HC573, 433  
 zatrzasknięcie adresu, 369  
 zdarzenia, 297  
 zdarzenie compare match, 399  
 zegar, 383, 420, 426  
 zintegrowane środowisko programistyczne, 11  
 złącze EuroSCART, 425, 428, 429  
 zmiana  
   głośności, 527  
   wielkości strony, 94  
 znaki, 56

**Ż**

źródło  
   napięcia odniesienia, 284  
   sygnału, 240

# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

**Mikrokontrolery bez urządzeń peryferyjnych** stanowią niszę rozwijaną głównie przez pasjonatów elektroniki, dla których programowanie jest celem samym w sobie. Praktyczne zastosowanie takich gadżetów jest możliwe dopiero po dołączeniu pamięci masowej, wyświetlacza alfanumerycznego i graficznego, przetworników analogowo-cyfrowych i cyfrowo-analogowych oraz czujników. Dzięki tym układom mikrokontrolery AVR komunikują się ze światem, zbierają informacje, magazynują dane i mogą służyć nam w codziennym życiu jako stacje pogodowe, odtwarzacze cyfrowe czy sterowniki ogrzewania.

**Jeśli masz już pewną wiedzę** na temat mikrokontrolerów AVR i chciałbyś ją wzbogacić o wiadomości dotyczące ciekawych zastosowań urządzeń peryferyjnych, sięgnij po jedyną w swoim rodzaju książkę AVR. *Układy peryferyjne*. Jest ona adresowana do czytelników, którzy chcą poszerzyć swoje praktyczne umiejętności programowania mikrokontrolerów AVR, w tym mikrokontrolerów z rodziny XMEGA. Stanowi naturalną kontynuację doskonałego podręcznika AVR. *Praktyczne projekty*, kierowanego do mniej zaawansowanych programistów. Książka omawia także specyficzne dla mikrokontrolerów AVR elementy języka C.

- Łączenie kodu C i asemblera oraz debugowanie programu
- Obsługa interfejsów dostępu do pamięci zewnętrznej
- Tworzenie i używanie systemu plików
- Korzystanie z przetworników ADC i DAC oraz komparatorów
- Obsługa wyświetlaczy alfanumerycznych i graficznych
- Przetwarzanie dźwięku i danych wideo

**Twórz praktyczne rozwiązania z mikrokontrolerami AVR i układami peryferyjnymi!**

Patronat medialny:

[mikrokontrolery.blogspot.com](http://mikrokontrolery.blogspot.com)

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 19426



Księgarnia internetowa:  
<http://helion.pl>



Zamówienia telefoniczne:  
**0 801 339900**



**0 601 339900**



**Helion**

Sprawdź najnowsze promocje:  
• <http://helion.pl/promocje>  
Książki najchętniej czytane:  
• <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
• <http://helion.pl/novosci>

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN: 978-83-246-9225-5



9 788324 692255

Cena: 99,00 zł

Informatyka w najlepszym wydaniu