

TOMASZ FRANCUZ

JĘZYK C

DLA MIKROKONTROLERÓW AVR
OD PODSTAW DO ZAAWANSOWANYCH APLIKACJI



Prezentujemy przebojowy duet — język C i mikrokontroler AVR!

Poznaj budowę i podstawy programowania mikrokontrolerów

Dowiedz się, jak do swoich celów wykorzystać język C

Naucz się rozwiązywać rzeczywiste problemy i tworzyć praktyczne rozwiązania



» Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

Język C dla mikrokontrolerów AVR. Od podstaw do zaawansowanych aplikacji

Autor: [Tomasz Francuz](#)

ISBN: 978-83-246-3064-6

Format: 158×235, stron: 568



Prezentujemy przebojowy duet – język C i mikrokontroler AVR!

- Poznaj budowę i podstawy programowania mikrokontrolerów
- Dowiedz się, jak do swoich celów wykorzystać język C
- Naucz się rozwiązywać rzeczywiste problemy i tworzyć praktyczne rozwiązania

Mikrokontrolery AVR firmy Atmel stanowią dynamicznie rozwijającą się rodzinę układów. Dzięki niskiej cenie, dużym możliwościom i dostępności darmowych narzędzi od lat niezmiennie cieszą się dużą popularnością wśród hobbystów i osób profesjonalnie zajmujących się programowaniem mikrokontrolerów.

Pewnym utrudnieniem dla polskich użytkowników AVR jest brak literatury na temat wykorzystania do ich programowania języków wysokiego poziomu, takich jak C. Niniejsza książka jest próbą wypełnienia tej luki. W sposób syntetyczny pokazuje różnice pomiędzy programowaniem w języku C komputerów klasy PC i mikrokontrolerów. Omawia programowanie peryferii dostępnych w mikrokontrolerach AVR w języku C, bibliotekę standardową oraz jej rozszerzenia znane jako AVR-libc. Dzięki temu nawet osoby w niewielkim stopniu znające podstawy języka C będą mogły bez problemów „przejąć się” na programowanie mikrokontrolerów AVR. Z drugiej strony książka opisuje zaawansowane techniki programowania, związane z obsługą bootloadera, zabezpieczaniem i szyfrowaniem kodu aplikacji oraz realizacją najpowszechniej stosowanych protokołów wymiany danych pomiędzy urządzeniami opartymi na mikrokontrolerach i komputerami PC. Porusza także tematy związane ze specyfiką pisania aplikacji na mikrokontrolery oraz wyszukiwaniem i usuwaniem błędów.

Podstawy programowania mikrokontrolerów AVR

- Warsztat pracy programisty AVR
- Wprowadzenie do języka C na AVR
- Budowa programu i jego części składowe
- Korzystanie z zasobów sprzętowych mikrokontrolera
- Używanie rejestrów i różnych rodzajów pamięci
- Zastosowania przetwornika ADC
- Obsługa wyświetlaczy LCD
- Korzystanie z interfejsów
- Zapewnianie bezpieczeństwa kodu

Programowanie mikrokontrolerów jeszcze nigdy nie było tak proste!

Spis treści

Wstęp	11
Kody przykładów	12
Schematy	12
Wymagane części	12
Rozdział 1. Instalacja środowiska i potrzebnych narzędzi	15
Instalacja WinAVR	16
Instalacja AVR Studio	17
Systemy GNU/Linux	18
AVR Studio	19
Pierwsza aplikacja	21
Dodawanie plików do projektu	25
Programy narzędziowe	27
Linker	27
Program avr-size	31
Program avr-nm	32
Program avr-objcopy	33
Program make	36
Pliki wynikowe	43
Biblioteki	46
Projekt biblioteki	47
Tworzenie biblioteki	48
Dołączanie biblioteki do programu	49
Funkcje „przestarzałe”	50
Nadpisywanie funkcji bibliotecznych	50
Usuwanie niepotrzebnych funkcji i danych	51
Rozdział 2. Programowanie mikrokontrolera	53
Podłączenie — uwagi ogólne	53
Problemy	55
Programatory ISP	55
Budowa programatora	56
Programator USBASP	59
Kilka procesorów w jednym układzie	59
Programatory JTAG	60
Programator JTAGICE	61
Programator JTAGICE mkII	62

Kilka procesorów w jednym układzie	62
AVR Dragon	63
Programatory HW i równoległe	63
Tryb TPI	64
Programowanie procesora w AVR Studio	64
Programowanie przy pomocy narzędzi dostarczonych przez firmę Atmel	65
Program AVRDUDE	67
Program PonyProg	70
Fusebity i lockbity w AVR-libc	70
Lockbity	71
Fusebity	71
Sygnatura	74
Lockbity w AVR-libc	74
Fusebity w AVR-libc	75
Rozdział 3. Podstawy języka C na AVR	77
Arytmetyka	77
Proste typy danych	77
Arytmetyka stałopozycyjna	81
Arytmetyka zmiennopozycyjna	87
Operacje bitowe	95
Reprezentacja binarna liczb	95
Operacja iloczynu bitowego	96
Operacja sumy bitowej	97
Operacja sumy wyłączającej	98
Operacja negacji bitowej	99
Operacje przesunięć bitowych	100
Zasięg zmiennych	100
Zmienne globalne	101
Zmienne lokalne	102
Modyfikator const	103
Wskaźniki	104
Tablice	109
Funkcje	112
Przekazywanie parametrów przez wartość i referencję	114
Wywołanie funkcji	114
Rekurencyjne wywołania funkcji	115
Słowa kluczowe	116
Operatory	116
Instrukcje sterujące	120
Preprocesor	123
Dyrektywa #include	124
Dyrektywy kompilacji warunkowej	124
Dyrektywa #define	126
Pliki nagłówkowe i źródłowe	127
Definicja a deklaracja	128
Słowo kluczowe static	129
Słowo kluczowe extern	130
Dyrektywa inline	132
Modyfikator register	136
Rozdział 4. Sekcje programu	141
Sekcje danych	142
Sekcja .text	142
Sekcja .data	142

Sekcja .bss	143
Sekcja .eeprom	143
Sekcje zawierające kod programu	144
Podsekcje .init[0-9]	144
Podsekcje .fini[0-9]	145
Sekcje specjalne	146
Sekcje tworzone przez programistę	146
Umieszczanie sekcji pod wskazanym adresem	147
Rozdział 5. Kontrola rdzenia i zarządzanie poborem energii	149
Źródła sygnału RESET	149
Power-on Reset	150
Zewnętrzny sygnał RESET	151
Brown-out Detector	151
Układ Watchdog	152
Zarządzanie poborem energii	156
Usypianie procesora	157
Wyłączanie układu BOD	157
Wyłączanie podsystemów procesora	158
Preskaler zegara	159
Inne sposoby minimalizowania poboru energii	160
Rozdział 6. Dynamiczna alokacja pamięci	163
Alokacja pamięci w bibliotece AVR-libc	164
Funkcja malloc	166
Funkcja calloc	166
Funkcja realloc	166
Funkcja free	168
Wycieki pamięci i błędne użycie pamięci alokowanej dynamicznie	169
Jak działa alokator	171
Wykrywanie kolizji sterty i stosu	172
Metoda I — własne funkcje alokujące pamięć	173
Metoda II — sprawdzanie ilości dostępnej pamięci	173
Metoda III — marker	173
Metoda IV — wzór w pamięci	173
Metoda V — wykorzystanie interfejsu JTAG	176
Rozdział 7. Wbudowana pamięć EEPROM	177
Zapobieganie uszkodzeniu zawartości pamięci EEPROM	178
Kontrola odczytu i zapisu do pamięci EEPROM	179
Odczyt zawartości komórki pamięci	180
Zapis do komórki pamięci	180
Dostęp do EEPROM z poziomu AVR-libc	181
Deklaracje danych w pamięci EEPROM	182
Funkcje realizujące dostęp do pamięci EEPROM	183
Inne funkcje operujące na EEPROM	185
Techniki wear leveling	186
Rozdział 8. Dostęp do pamięci FLASH	189
Typy danych związane z pamięcią FLASH	190
Odczyt danych z pamięci FLASH	191
Dostęp do pamięci FLASH >64 kB	192

Rozdział 9. Interfejs XMEM	193
Wykorzystanie zewnętrznej pamięci SRAM w programie	197
Konfiguracja I — w pamięci zewnętrznej jest tylko sekcja specjalna	198
Konfiguracja II — wszystkie sekcje w pamięci zewnętrznej, stos w pamięci wewnętrznej	199
Konfiguracja III — w pamięci zewnętrznej umieszczona jest tylko sarta	201
Konfiguracja IV — w pamięci zewnętrznej sarta i segment zdefiniowany przez programistę	202
Konfiguracja V — w pamięci zewnętrznej znajduje się stos	208
Pamięć ROM jako pamięć zewnętrzna	208
Rozdział 10. Dostęp do 16-bitowych rejestrów IO	211
Dostęp do 16-bitowego rejestru ADC	211
Dostęp do 16-bitowych rejestrów timerów	213
Rozdział 11. Opóźnienia	217
Rozdział 12. Dostęp do portów IO procesora	221
Konfiguracja pinu IO	221
Manipulacje stanem pinów IO	225
Zmiana stanu portu na przeciwny	225
Ustawianie linii IO	226
Zerowanie linii IO	226
Makrodefinicja _BV()	227
Użycie pól bitowych	227
Synchronizator	228
Przykłady praktyczne	230
Sterowanie wyświetlaczem 7-segmentowym	230
Podłączenie przycisków	232
Enkoder obrotowy	237
Klawiatura matrycowa	242
Rozdział 13. Rejestry IO ogólnego przeznaczenia	245
Wykorzystanie innych rejestrów jako GPIOR	246
Rozdział 14. Przerwania	249
Obsługa przerw	251
sei()/cli()	254
Atrybut naked i obsługa przerw w assemblerze	254
Modyfikator volatile	257
Atomowość dostępu do danych	263
Funkcje reentrant	266
Przykłady praktyczne	268
Wyświetlanie multipleksowane	268
Wyświetlanie multipleksowane z regulacją jasności wyświetlacza	272
Obsługa przycisków	276
Obsługa enkodera	279
Klawiatura matrycowa	280
Rozdział 15. Przetwornik analogowo-cyfrowy	283
Wybór napięcia referencyjnego	284
Multiplekser	285
Przetwornik ADC	285
Tryb pojedynczej konwersji	286
Tryb ciągłej konwersji	287
Wejścia pojedyncze i różnicowe	287

Wynik	288
Wyzwalacze	288
Blokowanie wejść cyfrowych	289
Przerwania ADC	289
Precyzyjne pomiary przy pomocy ADC	290
Nadpróbkowanie	291
Uśrednianie	292
Decymacja i interpolacja	292
Przykłady	292
Termometr analogowy LM35	293
Klawisze	295
Rozdział 16. Komparator analogowy	301
Funkcje dodatkowe	302
Blokowanie pinów	302
Wyzwalanie zdarzeń timera	302
Wybór wejścia komparatora	302
Wyzwalanie przetwornika ADC	303
Rozdział 17. Timery	305
Sygnał taktujący	306
Wewnętrzny sygnał taktujący	306
Zewnętrzny sygnał taktujący	308
Licznik	308
Układ porównywania danych	309
Wpływ na piny IO	309
Moduł przechwytywania zdarzeń zewnętrznych	310
Eliminacja szumów	311
Komparator jako wyzwalacz zdarzenia ICP	311
Tryby pracy timera	312
Tryb prosty	312
Tryb CTC	315
Tryby PWM	316
Układ ochronny	321
Modulator sygnału wyjściowego	322
Miernik częstotliwości i wypełnienia	323
Realizacja RTC przy pomocy timera	326
Realizacja sprzętowa	327
Realizacja programowa	328
Rozdział 18. Obsługa wyświetlaczy LCD	331
Obsługa wyświetlaczy alfanumerycznych	332
Funkcje biblioteczne	337
Definiowanie własnych znaków	342
Przykład — menu	345
Obsługa wyświetlaczy graficznych	354
Rozdział 19. Interfejs USART	367
Interfejsy szeregowo	367
Interfejs USART	368
Interfejs USART mikrokontrolera AVR	371
Przykłady	375
Połączenie mikrokontroler – komputer PC	375
RS485	383

Rozdział 20. Interfejs SPI	391
Inicjalizacja interfejsu	394
Ustawienie pinów IO	395
Zegar taktujący	396
Procesor w trybie Master SPI	396
Procesor w trybie slave SPI	397
Przykłady	397
Połączenie AVR-AVR	397
Połączenie AVR – rejestr szeregowy	403
Interfejs USART w trybie SPI	408
Taktowanie magistrali SPI	409
Tryb pracy SPI	409
Format ramki danych	409
Konfiguracja interfejsu	410
Rozdział 21. Interfejs TWI	413
Tryb multimaster	416
Inicjalizacja interfejsu	417
Procesor w trybie I2C master	417
Bity START i STOP	417
Podstawowe funkcje do współpracy z I2C	418
Współpraca z zewnętrzną pamięcią EEPROM	422
Współpraca z zewnętrzną pamięcią FRAM	427
Umieszczanie zmiennych w zewnętrznej pamięci EEPROM	427
Współpraca z zegarem RTC	431
Obsługa ekspandera IO PCF8574	436
Procesor w trybie I2C slave	437
Przykład	440
Rozdział 22. Interfejs USI	447
4-bitowy licznik i zegar	447
Przerwania USI	448
Zmiana pozycji pinów	449
Wykorzystanie interfejsu USI w trybie SPI	449
Tryb SPI master	451
Tryb SPI slave	452
Rozdział 23. Interfejs USB	453
Zasilanie	454
Sygnały danych	455
VID i PID	456
Interfejs USB realizowany przy pomocy konwertera	458
Interfejs USB realizowany programowo	459
Połączenie elektryczne	460
Dostęp na PC	460
Programowy interfejs USB na AVR	461
Sprzętowy interfejs USB	464
Rozdział 24. Interfejs 1-wire	465
Realizacja master 1-wire na AVR	469
Realizacja master 1-wire przy pomocy pinów IO	469
Realizacja master 1-wire przy pomocy interfejsu USART	472
Wysokopoziomowe funkcje obsługi 1-wire	477
Termometr cyfrowy DS1820	480

Rozdział 25. Bootloader	483
Pamięć NRWW i RWW	483
Bity konfiguracyjne bootloadera	485
Konfiguracja lockbitów z poziomu aplikacji	486
Programowanie pamięci FLASH	487
Wykorzystanie przerw w kodzie bootloadera	489
Usuwanie tablicy wektorów przerw	490
Skrócenie tablicy wektorów przerw	491
Start bootloadera	496
Wykorzystanie dodatkowego przycisku/zworki	496
Wykorzystanie markerów w pamięci EEPROM	497
Oczekiwanie na specjalny znak w wybranym kanale komunikacji	498
Start aplikacji	499
Współdzielenie kodu aplikacji i bootloadera	499
Wywoływanie funkcji bootloadera w procesorach ATmega256x	501
Wywoływanie funkcji obsługi przerw zawartych w kodzie bootloadera	505
Współdzielenie zmiennych pomiędzy aplikacją a bootloaderem	505
Mikrokontrolery AVR z wbudowanym bootloaderem	507
Rozdział 26. Kontrola integralności programu	509
Suma kontrolna	509
CRC	511
Automatyczne generowanie CRC	514
Rozdział 27. Bezpieczeństwo kodu	517
Metody łamania zabezpieczeń	517
Bezpieczne uaktualnianie aplikacji	518
Nota AVR231 — AES Bootloader	519
Ustawienie bitów konfiguracyjnych	524
Przygotowanie aplikacji	526
Wczytywanie uaktualnienia	527
Rozdział 28. Łączenie kodu w C i asemblerze	529
Słowo kluczowe asm	530
Typy operandów	531
Dostęp do portów IO	533
Dostęp do danych wielobajtowych	533
Dostęp do wskaźników	534
Lista modyfikowanych rejestrów	535
Wielokrotne użycie wstawki asemblerowej	535
Pliki .S	536
Wykorzystanie rejestrów w asemblerze	537
Przykłady	541
Rozdział 29. Optymalizacja i debugowanie programu	543
Optymalizacja programu	543
Opcje kompilatora związane z optymalizacją	545
Atrybuty optymalizacji	548
Debugowanie programu	551
Rozpoczęcie sesji debugera	553
Zaawansowane sterowanie przebiegiem wykonywanej aplikacji	556
Skorowidz	559

Rozdział 2.

Programowanie mikrokontrolera

Po wygenerowaniu plików wynikowych należy ich zawartość umieścić w pamięci mikrokontrolera. Dzięki temu po restarcie procesor będzie mógł rozpocząć wykonywanie programu. Procesory AVR dysponują możliwością programowania „w układzie” przy pomocy interfejsu ISP, część może być programowana poprzez interfejsy JTAG, debugWire, PDI, TPI, a w przypadku procesorów posiadających interfejs USB można także programować procesor poprzez wbudowany *bootloader*. W tym ostatnim przypadku nie da się jednak zmieniać konfiguracji *fusebitów*. Każda z metod programowania ma swoje zalety i wady.

Podłączenie — uwagi ogólne

Każdy programator łączy się z układem docelowym przy pomocy dedykowanych wyprowadzeń. Dla programatorów szeregowych jest to zwykle 4 – 5 wyprowadzeń, dla równoległych znacznie więcej. Wykorzystanie wyprowadzeń mikrokontrolera do programowania ogranicza możliwość ich wykorzystania do innych celów. Najlepiej, jeśli takie wyprowadzenia nie będą wykorzystywane do niczego innego — w układzie będą one podłączone wyłącznie do gniazda łączącego z programatorem. Jednak w układach posiadających niewielką liczbę wyprowadzeń nie zawsze jest to możliwe. Stąd też powinniśmy pamiętać, aby podłączone do tych wyprowadzeń urządzenia nie obciążały ich zbyt mocno (w trybie programowania będą one obciążały wyjście programatora). Z tego powodu nie zaleca się podłączać do nich np. diod LED, nie należy na tych liniach dodawać także kondensatorów, szczególnie o większych pojemnościach ($>1\text{ nF}$). Dodatkowo jeśli jakieś wyprowadzenie jest wejściem (czyli wyjściem układu programującego), nie należy łączyć do niego innych wyjść — w takiej sytuacji w trakcie programowania powstanie konflikt pomiędzy wyjściem programatora a wyjściem układu korzystającego z tego pinu.

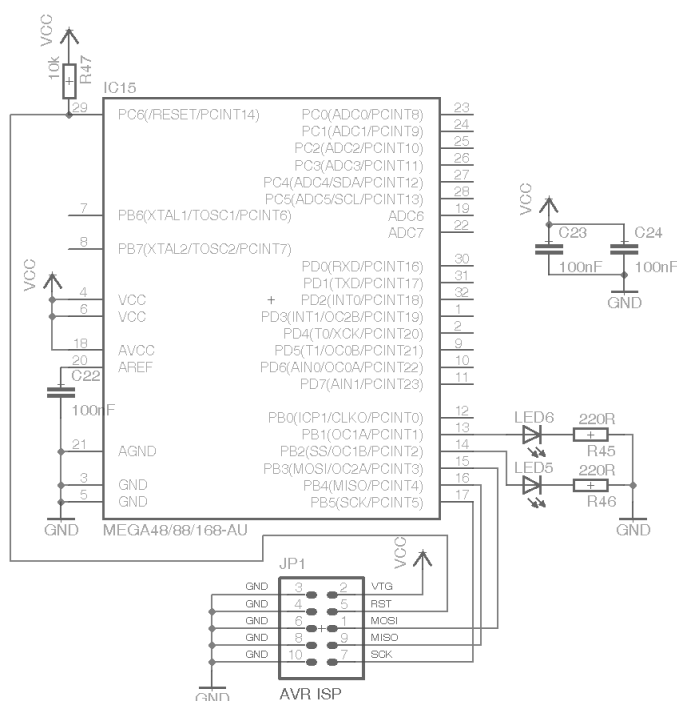


Pamiętajmy, że dla programatora nie ma znaczenia, jak w programie skonfigurowane są piny wykorzystywane do programowania.

Programator na czas programowania wprowadza procesor w stan *RESET*, co wiąże się z wprowadzeniem wszystkich pinów *IO* w stan wysokiej impedancji.

Szczególną uwagę należy zwrócić na podłączenie sygnału *RESET*. Aby wejść w tryb programowania, programator musi mieć możliwość wymuszenia na tej linii stanu niskiego (lub +12 V w przypadku programatorów wysokonapięciowych). W efekcie problem może wystąpić, jeśli w układzie używane są zewnętrzne układy generujące *RESET* lub monitorujące zasilanie. Przykładowy schemat podłączenia procesora do programatora ISP pokazano na rysunku 2.1.

Rysunek 2.1.
Przykład podłączenia
programatora ISP
do mikrokontrolera
ATMega88



W dalszych rozdziałach książki pokazane schematy, dla uproszczenia i większej przejrzystości, nie będą już zawierały elementów pokazanych na schemacie z rysunku 2.1, czyli gniazda ISP, kondensatorów odsprężających oraz połączeń z zasilaniem (Vcc i GND).

Aby pokazane w dalszej części układy działały poprawnie, należy zapewnić poprawne połączenie elementów pokazanych na rysunku 2.1. Szczególnie istotne jest podłączenie w każdym procesorze wszystkich występujących w nim wyprowadzeń zasilania (Vcc i GND). Przy braku podłączenia niektórych wyprowadzeń układ może działać

niestabilnie i stwarzać problemy. Drugim istotnym elementem są kondensatory odsprężające C23 i C24. Są to elementy, których zadaniem jest odsprężanie zasilania, a ich znaczenie rośnie wraz ze wzrostem stopnia skomplikowania układu.

Problemy

Najczęstsze problemy z zaprogramowaniem procesora:

1. W przypadku programowania w trybie ISP „zablokowanie” procesora, poprzez niewłaściwą konfigurację *fusebitów*.
2. Nieprawidłowa częstotliwość sygnału *SCK* (w przypadku programatorów ISP). Jeśli podejrzewamy taki problem, należy zmniejszyć szybkość programowania. W żadnym przypadku nie może ona przekroczyć $\frac{1}{4}$ częstotliwości taktowania procesora.
3. Zbyt długi kabel łączący programator z układem. Im dłuższy kabel, tym większe ryzyko niepoprawnej pracy układu. Zwykle problem ten objawia się niestabilną pracą programatora.
4. Błędne podłączenie sygnałów. Zawsze warto się upewnić, że wszystkie sygnały zostały prawidłowo połączone z odpowiednimi wyprowadzeniami procesora.
5. Pomyłkowe podłączenie programatora nie do wyprowadzeń związanych z ISP, lecz do wyprowadzeń związanych z interfejsem SPI (oznaczenia linii sygnałowych są podobne). Problem ten dotyczy głównie procesorów ATMega128.
6. Wybór niewłaściwego programatora lub niewłaściwego trybu programowania.
7. Zablokowanie wykorzystywanego interfejsu (dotyczy głównie próby programowania przy wyłączonym interfejsie JTAG lub próby programowania przy pomocy ISP, z włączonym interfejsem debugWire).

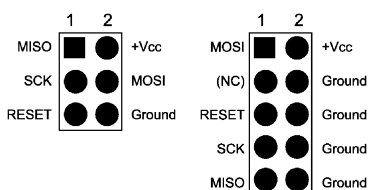
Programatory ISP

Prawie każdy procesor AVR dysponuje możliwością programowania przy pomocy interfejsu ISP (ang. *In-system Programming Interface*). Interfejs ten wykorzystuje do programowania piny *RESET*, *MISO*, *MOSI* oraz *SCK*. Zwykle piny te pokrywają się z analogicznymi wyprowadzeniami interfejsu SPI, lecz nie zawsze tak jest. Jednym z takich wyjątków jest procesor ATMega128. Stąd też zawsze należy dokładnie sprawdzić, jakie wyprowadzenia procesora wykorzystywane są do programowania przy pomocy interfejsu ISP. Informacje o wykorzystanych wyprowadzeniach znajdują się w nocie katalogowej procesora, w sekcji *Memory Programming/Serial Downloading*. Programatory ISP mają znormalizowany układ sygnałów wyprowadzonych na złącze programujące, pokazany na rysunku 2.2.

Istnieją dwa typy złącza ISP — jedno mniejsze, 6-pinowe, oraz większe 10-pinowe. Odstęp pomiędzy pinami wynosi 2,54 mm, chociaż w nowszych konstrukcjach Atmela

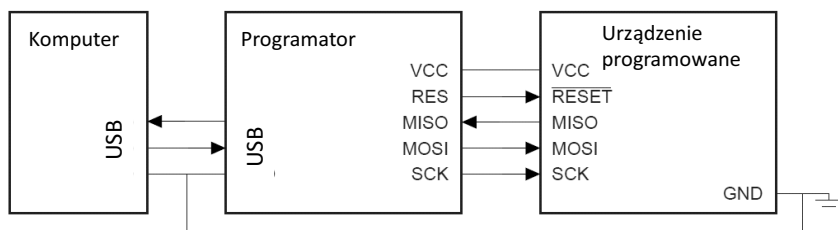
Rysunek 2.2.

Rozkład sygnałów standardowych programatorów. Kwadratem oznaczono wyprowadzenie o numerze 1



ISP Connectors: 6-pin & 10-pin

spotyka się rozstaw pinów 1,27 mm. Umożliwia to zmniejszenie rozmiarów złącza programującego na płytce. Pewnego omówienia wymaga przeznaczenie pinu oznaczonego jako +Vcc. Do tego pinu należy podłączyć napięcie zasilające układ. Napięcie z tego pinu wykorzystywane jest przez programator do zasilania buforów wyjściowych, dzięki czemu programator dostosowuje poziomy napięcie na pozostałych pinach programatora do napięć panujących w programowanym układzie. Część programatorów posiada także specjalną zwórkę przełączającą napięcie. W takiej sytuacji jedna z pozycji powoduje zasilanie programatora z programowanego układu, w drugiej pozycji to programator zasilą programowany układ. Programator łączy się z programowanym układem tak, jak pokazano na rysunku 2.3.



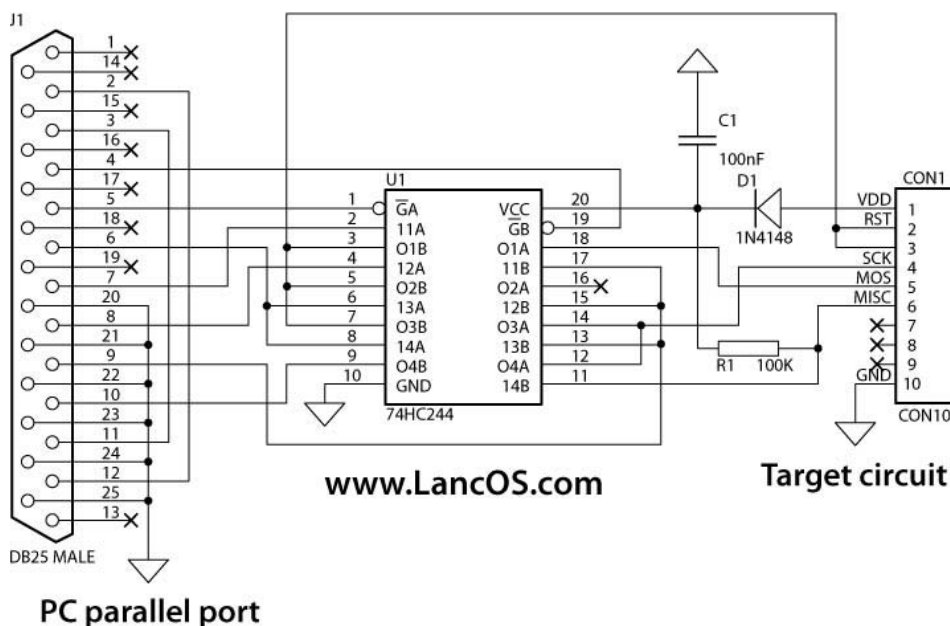
Rysunek 2.3. Połączenie programowanego układu z programatorem i komputerem PC. Masy wszystkich urządzeń muszą zostać połączone razem

Łącząc programator z komputerem i programowanym układem, należy zwracać uwagę na potencjał masy. W komputerach klasy PC, ze względu na budowę zasilacza, masa (obudowa komputera) przy braku zerowania ma potencjał ok. 115 V względem ziemi (wynika to z istnienia w zasilaczu układu filtrującego). W efekcie przy braku zerowania komputera lub niepoprawnym zerowaniu może dojść do uszkodzenia programatora lub programowanego układu. Aby uniknąć takich przykrych niespodzianek, można zaopatrzyć się w programator z optoizolowanymi wyjściami, lecz jest to dodatkowy, spory wydatek.

Budowa programatora

Programatory ISP są jednymi z najprostszych w budowie, w związku z tym każdy może poskładać sobie taki programator, dosłownie z niczego. W szczególnie dobrej sytuacji są tu użytkownicy posiadający komputery z wyprowadzonym portem równoległym. W takiej sytuacji programator może być zwykłą przejściówką pomiędzy portem komputera a gniazdem ISP. Układ taki jest niezwykle prosty, lecz niezalecany. Jakikolwiek błąd w połączeniach może bardzo łatwo doprowadzić do uszkodzenia portu równoległego,

dotatkowo jego niewielka wydajność prądowa powoduje znaczne ograniczenie maksymalnej długości przewodu łączącego komputer z programowanym układem (w praktyce do kilkunastu cm). Stąd znacznie lepszym rozwiązaniem jest to pokazane na rysunku 2.4.



Rysunek 2.4. Schemat prostego programatora ISP podłączanego do portu równoległego komputera

Jak widać, programator taki składa się z bufora 74XX244 (nie musi to być układ serii HC). Jego wprowadzenie umożliwia znaczne wydłużenie przewodu łączącego komputer z programatorem, nawet do 1 m i więcej (należy mieć na uwadze, że zbyt długi przewód łączący nie jest zalecany i w pewnych okolicznościach może prowadzić do problemów z programowaniem). Wprowadzenie tego układu chroni także port równoległy. W przypadku błędnego podłączenia zasilania np. do pinów programatora uszkodzeniu ulegnie tylko tani układ buforujący, ochraniając port równoległy. Wykonanie takiego programatora to koszt rzędu kilku złotych, lecz już za kilkanaście złotych można kupić programatory bardziej rozbudowane, których zaletami są:

- ♦ bezpośrednia współpraca z AVR Studio;
- ♦ możliwość programowania układów zasilanych innym napięciem niż 5 V;
- ♦ współpraca z portem USB mikrokontrolera.

Szczególnie ta ostatnia cecha jest pożądana. Porty USB, w przeciwieństwie do równoległych, występują praktycznie w każdym urządzeniu, lecz ich największą zaletą jest możliwość czerpania energii z takiego portu. Stwarza to możliwość nie tylko zasilania samego programatora, ale także zasilania programowanego układu (tu jednak trzeba mieć na uwadze ograniczoną do ok. 0,5 A wydajność prądową portu USB).

Programator AVRISP

Jest to prosty programator z możliwością podłączenia poprzez port szeregowy RS232 lub USB. Na rynku dostępne są liczne klony tego układu, w efekcie można go kupić już za kilkanaście złotych, co czyni go szczególnie interesującym dla amatora. W stosunku do prostych programatorów, posiadających tylko bufor, jego zaletą jest możliwość programowania układów zasilanych napięciem w granicach 2,7 – 5,5 V. Klasyczny programator AVRISP zasilany jest z urządzenia programowanego, lecz jego wersje na USB często posiadają zworki, umożliwiające wybór źródła zasilania. Przy jego pomocy można programować szeroką gamę modeli procesorów AVR. Wyjątkiem jest tu tylko rodzina AVR XMeta oraz AVR32, wymagające programatora AVRISP mkII. Programator ten posiada wbudowany procesor, którego firmware kontroluje proces programowania. Wraz z uaktualnianiem AVR Studio firmware ten też może zostać uaktualniony, w efekcie poszerza się lista obsługiwanych procesorów.

Programator ten jest rozwiązaniem tanim, lecz warto mieć na uwadze, że obecnie nie jest on już praktycznie rozwijany przez firmę Atmel. Stąd potencjalnie mogą być problemy z jego wykorzystaniem z najnowszymi modelami procesorów.

Programator AVRISP mkII

Rozwiązaniem dla bardziej zaawansowanych amatorów i osób, które półprofesjonalnie chcą się zajmować mikrokontrolerami AVR, jest programator AVRISP mkII. Również ten programator dostępny jest w postaci klonów, w efekcie można go kupić za cenę ok. 100 – 150 zł. Jest to programator aktywnie wspierany przez firmę Atmel, wspierający wszystkie rodziny procesorów AVR (łącznie z procesorami XMeta oraz AVR32). Wspiera także procesory ATTiny, nieposiadające interfejsu ISP, dzięki możliwości wykorzystania interfejsu TPI.

Programator ten może programować układy zasilane napięciem od 1,8 do 5,5 V, dodatkowo można regulować częstotliwość sygnału zegarowego taktującego transmisję w zakresie 50 Hz – 8 MHz. Ma to istotną zaletę w przypadku programowania układów niskonapięciowych, taktowanych z wolnych zegarów, np. kwarców zegarkowych o częstotliwości 32 768 Hz.



Wskazówka

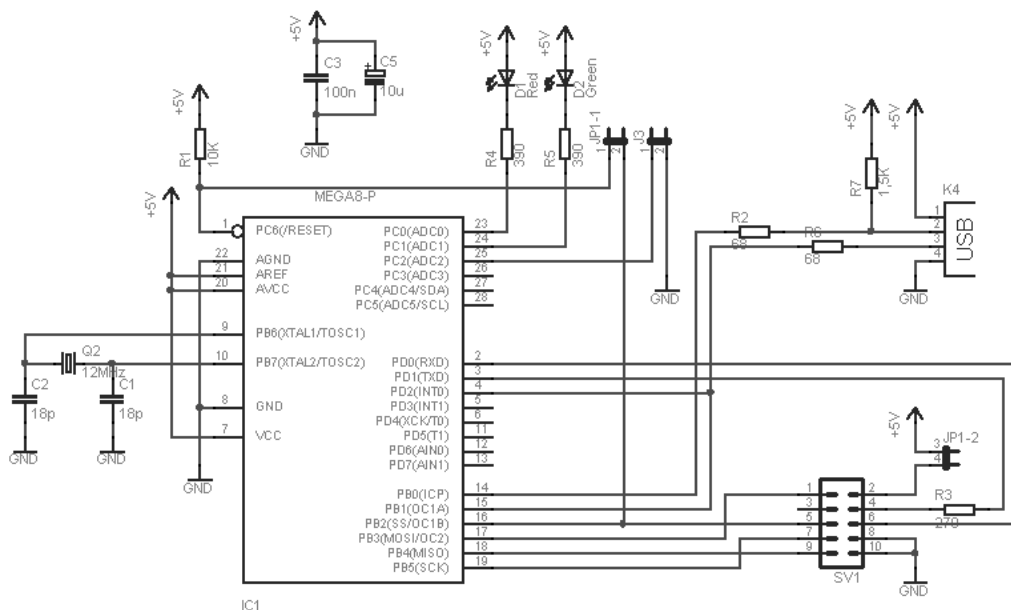
Maksymalna prędkość programowania wynika z ograniczeń interfejsów szeregowych — częstotliwość linii *SCK* nie może być większa niż czterokrotność częstotliwości taktującej rdzeń procesora.

Tak więc wykorzystanie programatora AVRISP mkII umożliwia programowanie procesorów taktowanych zegarem od 200 Hz wzwyż.

Programator ten współpracuje z interfejsem USB, posiada także wyjścia zabezpieczone przez zwarcie.

Programator USBASP

Programator ten jest niezwykle popularny ze względu na jego prostotę oraz niską cenę. Dodatkową zaletą tego programatora jest wsparcie ze strony WinAVR oraz bardzo popularnego programu AVRdude. Schemat tego programatora pokazano na rysunku 2.5.



Rysunek 2.5. Schemat programatora USBASP. Programator ten zawiera mikrokontroler sterujący procesem programowania układu docelowego. Dzięki temu programator ten jest niezależny od przebiegów czasowych generowanych przez komputer. Łączność z komputerem następuje poprzez złącze USB, stąd też programator czerpie zasilanie. Po zwarceniu zworki JP1-2 możliwe jest także zasilanie z portu USB układu programowanego

Programator ten umożliwia także programowanie procesorów taktowanych zegarami o niskiej częstotliwości. Przy pomocy zworki JP3 można przełączać częstotliwość linii SCK z 375 kHz na 8 kHz, co umożliwia programowanie układów taktowanych kwarcem zegarkowym o częstotliwości 32 768 Hz. Zwarcie zworki JP1-1 umożliwia zaprogramowanie lub uaktualnienie oprogramowania programatora poprzez jego złącze ISP.

Kilka procesorów w jednym układzie

Sporadycznie zdarza się, że na jednej płycie znajduje się więcej niż jeden procesor AVR i każdy powinien mieć zapewnioną możliwość programowania. Najchętniej w takiej sytuacji chcielibyśmy móc korzystać tylko z jednego gniazda programującego. Konfiguracja taka jest możliwa, musimy tylko pamiętać o spełnieniu pewnych dodatkowych założeń. W takiej sytuacji sygnały z programatora (RESET, MISO, MOSI) powinny być rozproszane do wszystkich mikrokontrolerów. Natomiast sygnał SCK musi pozostać

rozdzielony. Przy takiej konfiguracji wyboru programowanego mikrokontrolera dokonuje się poprzez wybranie mikrokontrolera, do którego doprowadzony zostanie sygnał *SCK*. Układ taki może działać, ponieważ aby procesor wszedł w tryb programowania (a co za tym idzie, linie *MOSI* i *MISO* stały się aktywne), musi być spełnionych kilka założeń. Po pierwsze, programator musi zapewnić aktywność sygnału *RESET*. Dzięki utrzymywaniu go w stanie aktywnym wyprowadzenia wszystkich mikrokontrolerów przechodzą w stan wysokiej impedancji. Dzięki temu nie zakłócają one transmisji. Uaktywnienie trybu programowania wymaga w takiej sytuacji doprowadzenia do wejścia *SCK* odpowiedniego przebiegu. Ponieważ przebieg taki zostanie doprowadzony wyłącznie do wybranego procesora, inne procesory pozostaną nieaktywne, z wyprowadzeniami w stanie wysokiej impedancji.

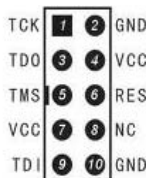
W przypadku gdy na płytce znajduje się jeden większy procesor i jeden lub więcej procesorów ze stosunkowo niewielką ilością pamięci FLASH, można rozważyć jeszcze jedną możliwość. Funkcję programatora może przejąć procesor „większy”, odpowiednio sterując wyprowadzeniami odpowiedzialnymi za programowanie innych procesorów. W takiej sytuacji ich przeprogramowanie wymaga wczytania do procesora kontrolującego pozostałe odpowiedniego programu oraz zawartości pamięci FLASH pozostałych procesorów. Rozwiązanie takie jest stosunkowo proste, lecz wymaga takiego podłączenia wszystkich mikrokontrolerów, aby ich wyprowadzenia programujące były dostępne dla procesora nadrzędnego.

Programatory JTAG

Programatory wykorzystujące interfejs JTAG są o wiele droższe, ale oprócz możliwości programowania przy ich pomocy procesora oferują także możliwość debugowania. Obecnie na rynku występują dwie wersje programatora JTAG dla mikrokontrolerów AVR — JTAGICE oraz JTAGICE II. Ten drugi cechuje się bardzo wysoką ceną (ok. 700 – 1200 zł), ale oferuje możliwość programowania wszystkich mikrokontrolerów AVR wyposażonych w interfejs JTAG. Za jego pomocą można także programować mikrokontrolery AVR32. Możliwości programatora JTAGICE są skromniejsze, ale za to jego cena jest niewiele wyższa niż programatora ISP. Programator ten łączy się z programowanym układem przy pomocy gniazda o innym rozkładzie sygnałów niż w przypadku programatora ISP — rysunek 2.6.

Rysunek 2.6.

Rozkład sygnałów
na złączu JTAG.
Pin 1 oznaczono
kwadratem



| Atmel Standard 10-pin JTAG layout

Interfejs JTAG wykorzystuje pięć sygnałów: *RESET*, *TCK*, *TMS*, *TDI* oraz *TDO*. Do pinu 7 (VCC) należy doprowadzić napięcie zasilające tylko w sytuacji, w której programator ma być zasilany z układu. Jeśli programator ma własne zasilanie, pin 7 można pozostawić

niepodłączony. Z kolei pin 4 dostarcza napięcia umożliwiającego programatorowi dostosowanie poziomu napięć na liniach *RESET*, *TCK*, *TMS*, *TDI* i *TDO* do napięć panujących w układzie. Na podstawie napięcia na tej linii programator wykrywa także podłączenie do układu programowanego. Wyprowadzenia oznaczone jako *NC* należy pozostawić niepodłączone.



Wskazówka

Aby móc korzystać z tego trybu, procesor musi obsługiwać interfejs JTAG, a *fusebit* JTAGEN musi być zaprogramowany (mieć wartość 0).

Programowanie przy użyciu interfejsu JTAG ma liczne zalety:

- ♦ Jest 3 – 4 razy szybsze w stosunku do programowania przy użyciu interfejsu ISP.
- ♦ Podobnie, znacznie szybsze jest także programowanie pamięci EEPROM.
- ♦ Umożliwia zmianę *fusebitów* określających źródło sygnału zegarowego, niezależnie od ich poprzednich wartości. Interfejs JTAG sam generuje zegar dla układu docelowego, stąd wybranie nawet błędnych wartości nie blokuje możliwości dalszego programowania (odmiennie niż w przypadku interfejsu ISP).
- ♦ Istnieje możliwość łączenia urządzeń w konfigurację *daisy-chain*, umożliwiającą programowanie wielu urządzeń przy pomocy jednego złącza JTAG.
- ♦ Istnieje możliwość programowania nie tylko mikrokontrolerów AVR, ale także innych układów kompatybilnych ze standardem JTAG (np. FPGA).



Wskazówka

Niezwykle istotną zaletą interfejsu JTAG jest możliwość debugowania przy jego pomocy programu w trakcie jego działania w docelowym układzie elektronicznym.

Możliwość taka jest wprost trudna do przecenienia, szerzej zostanie opisana w rozdziale 29.

Programator JTAGICE

Cena tego układu porównywalna jest z ceną dobrego programatora ISP. Jest to więc propozycja dla hobbystów zdecydowanie poważniej myślących o zajęciu się budowaniem układów w oparciu o mikrokontrolery AVR. Zastosowania tego programatora ogranicza stosunkowo niewielka liczba wspieranych układów [ATmega16(L), ATmega162(L), ATmega169(L or V), ATmega32(L), ATmega323(L), ATmega64(L), ATmega128(L)]. Lecz nawet pomimo tej wady warto rozważyć jego zakup, szczególnie jeśli jesteśmy w posiadaniu płytki rozwojowej zawierającej jeden z wyżej wymienionych procesorów. Programowanie przy jego pomocy jest nie tylko szybsze, lecz przede wszystkim udostępnia szerokie możliwości debugowania układu w systemie. Dzięki temu nawet jeśli pisany program będzie docelowo działał na innym typie procesora, łatwiej jest napisać aplikację na jednym ze wspieranych przez JTAGICE procesorów, a następnie ją tylko zmodyfikować dla potrzeb procesora docelowego. Użycie interfejsu JTAG umożliwia nie tylko debugowanie samego programu, ale także sprawdzenie stanu wszystkich bloków

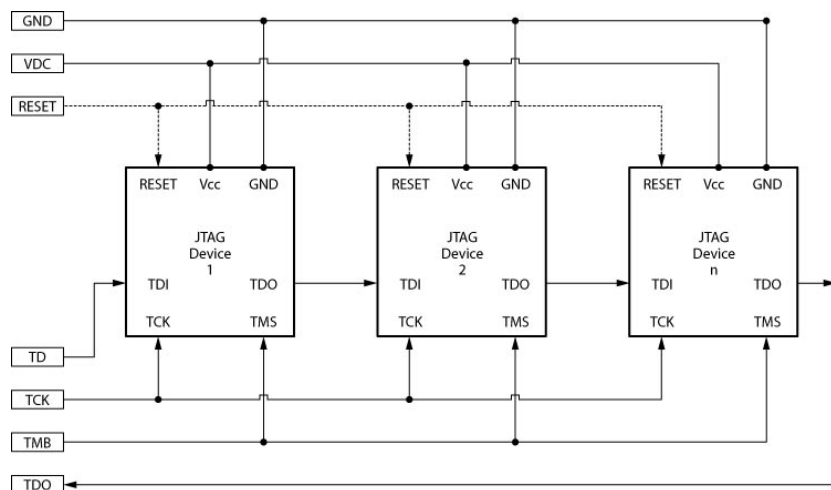
procesora, a także jego portów *I/O*. Oprócz możliwości sprawdzenia stanu można ich stan także modyfikować „w locie”. Ułatwia to testowanie poprawności połączeń elektrycznych na płycie i poprawności montażu.

Programator JTAGICE mkII

Programator JTAGICE mkII jest rozwinięciem układu JTAGICE. Umożliwia on programowanie wszystkich procesorów AVR wyposażonych w interfejs JTAG, w tym także procesorów z rodziny AVR32. Ze względu na cenę tego programatora (przekraczająca 1000 zł) jest to raczej propozycja dla osób chcących bardziej profesjonalnie zająć się programowaniem i budowaniem układów w oparciu o mikrokontrolery. Funkcjonalnie programator ten nie różni się od swojego poprzednika, udostępnia podobne możliwości. Oprócz programowania przez interfejs JTAG udostępnia także możliwość programowania z wykorzystaniem interfejsów PDI, debugWire, SPI oraz aWire. W efekcie za jego pomocą można zaprogramować praktycznie wszystkie procesory AVR.

Kilka procesorów w jednym układzie

Podobnie jak w przypadku ISP, także JTAG umożliwia wykorzystanie jednego złącza do programowania kilku układów. Funkcja taka jest wpisana w specyfikację protokołu JTAG, więc teoretycznie taka konfiguracja powinna być nawet łatwiejsza w realizacji. Tu, niestety, jak to zwykle bywa, napotykamy na problemy natury programowej. Większość dostępnego oprogramowania nie wspiera możliwości wybierania procesora w konfiguracji łańcuchowej JTAG (ang. *Daisy-chain JTAG mode*). Sytuacja ta stopniowo się zmienia i część oprogramowania dostarczonego przez firmę Atmel wspiera taką konfigurację dla programatora AVRICE mkII. Schemat podłączenia interfejsów JTAG w konfiguracji *daisy-chain* pokazano na rysunku 2.7. Linie *TDI* i *TDO* kolejnych procesorów są połączone szeregowo.



Rysunek 2.7. Połączenie kilku układów AVR, wykorzystujących jedno złącze JTAG

Inną możliwością jest rozwiązanie analogiczne do pokazanego przy okazji programowania ISP — połączenie równoległe odpowiednich linii JTAG, z wyjątkiem linii *SCK*. Wybór aktywnej linii *SCK* umożliwia wybór programowanego/debugowanego układu.



Korzystając z możliwości konfiguracji *daisy-chain*, należy mieć na uwadze jeszcze jeden problem — niektóre mikrokontrolery AVR mają błędną implementację obsługi JTAG, uniemożliwiającą zastosowanie konfiguracji *daisy-chain*. Stąd przed jej użyciem należy zawsze sprawdzić erratę do noty katalogowej procesora, zgodną z jego modelem oraz wersją układu.

AVR Dragon

Alternatywą dla wcześniej wymienionych programatorów, w tym dla drogiego JTAGICE mkII, jest układ AVR Dragon. W przeciwieństwie do wcześniejszych układów jest on sprzedawany bez obudowy, złącz i kabli. Potrzebne złącza należy wlutować samemu. Dzięki temu jego cena jest niezwykle atrakcyjna — można go kupić w cenie ok. 200 – 240 zł. **Niestety, brak wielu wbudowanych zabezpieczeń czyni go niezwykle podatnym na uszkodzenie.** Aby taką możliwość znacznie zmniejszyć, należy samemu dodać odpowiednie układy zabezpieczające — np. układy buforujące wyjścia programatora. Układ AVR Dragon umożliwia programowanie wszystkich mikrokontrolerów AVR dzięki wyposażeniu go w interfejsy HVPP, HVSP, ISP, JTAG, PDI. Umożliwia także debugowanie układu docelowego dzięki interfejsom JTAG i debugWire. Programowane układy mogą być zasilane napięciem z zakresu 1,8 – 5,5 V. Układ AVR Dragon może także dostarczać dla nich napięcia zasilającego o natężeniu maksymalnie 300 mA.

Programatory HW i równoległe

Programatory wysokonapięciowe (HW, ang. *High Voltage*) oraz równoległe są niezwykle rzadko wykorzystywane. Programator wysokonapięciowy wykorzystuje podobne sygnały co programator ISP, lecz podczas programowania na wejściu *RESET* procesora zamiast stanu niskiego doprowadzane jest napięcie +12V. Dzięki temu można programować procesory, w których przy pomocy *fusebitu* RSTDSBL wejście *RESET* zostało zablokowane. Nie wszystkie procesory dysponują możliwością programowania wysokonapięciowego. W tym trybie programowania procesor wymaga doprowadzenia sygnałów pokazanych w tabeli 2.1.

Tabela 2.1. Sygnały wykorzystywane do programowania w trybie wysokonapięciowym

Sygnał	Kierunek	Opis
SDI	Wejście	Wejście danych
SII	Wejście	Wejście instrukcji
SDO	Wyjście	Wyjście danych
SCI	Wejście	Wejście zegarowe

Programatory równoległe wykorzystywane są jeszcze rzadziej. Ich potencjalną zaletą jest większa szybkość działania, lecz do poprawnej pracy wymagają podłączenia kilkunastu różnych sygnałów. Zaletą tego typu programatorów jest możliwość programowania procesora zablokowanego w wyniku przeprogramowania *fusebitów* odpowiedzialnych za wybór zegara. Jest to możliwe, ponieważ w tym trybie programator generuje przebieg zegarowy taktujący procesor, który jest doprowadzony do wejścia *XTAL1*.

Tryb TPI

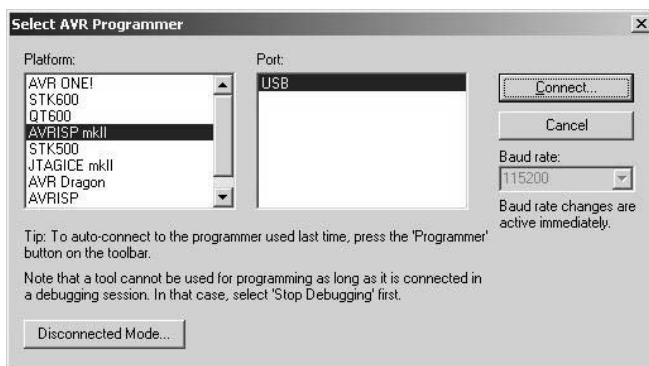
Jest to uproszczony interfejs umożliwiający programowanie najmniejszych procesorów Atmel z serii ATTiny. Używa on linii *RESET* oraz linii danych *TPIDATA* i zegara *TPICLK*. W przypadku kiedy pin *RESET* jest wykorzystywany jako zwykły pin *IO*, wejście w tryb TPI jest wymuszane poprzez podanie na ten pin napięcia +12 V. Protokół ten wspierany jest przez najnowsze programatory, m.in. AVRISP mkII, AVR Dragon.

Programowanie procesora w AVR Studio

Zdecydowanie najłatwiejszą opcją jest wykorzystanie do programowania zintegrowanego środowiska, jakim jest AVR Studio. Dzięki temu mamy możliwość, przy pomocy jednego programu, pisać program, kompilować go, debugować (za pomocą interfejsów sprzętowych lub wbudowanego w AVR Studio symulatora), a efekt finalny przy pomocy jednego przycisku wgrywać do pamięci procesora. Bezpośrednio AVR Studio wspiera narzędzia dostarczane przez firmę Atmel — programatory AVRISP, JTAGICE, Dragon. Po pewnych zabiegach można także korzystać z innych programatorów.

Rozpoczęcie procesu programowania w AVR Studio wymaga najpierw skonfigurowania interfejsu programatora — czyli wybrania z listy programatora, który posiadamy, oraz podania sposobu komunikacji z nim (rysunek 2.8). Opcję konfiguracji wybiera się z menu *Tools/Program AVR/Connect*.

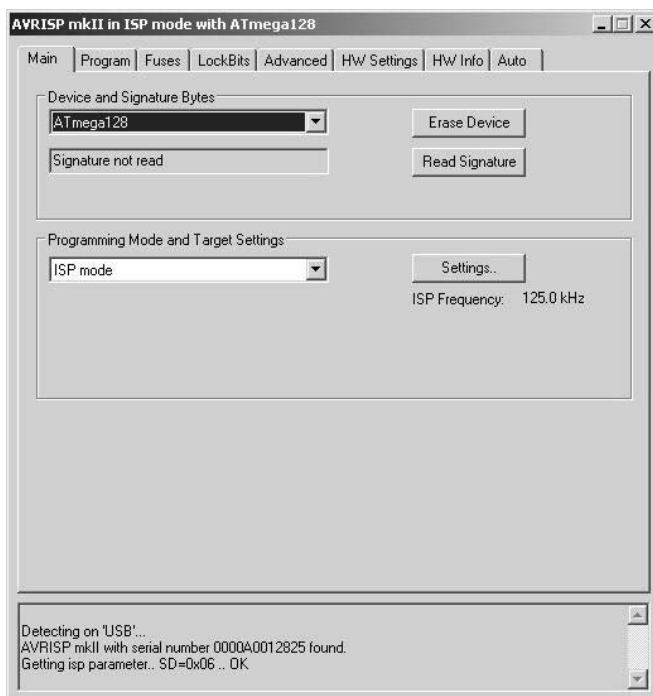
Rysunek 2.8.
Konfiguracja
programatora
w AVR Studio
— w powyższym
przykładzie został
wybrany programator
AVRISP mkII,
podłączony
przez port USB



Po udanej próbie nawiązania połączenia z programatorem wyświetlone zostanie kolejne okno, z opcjami, jakie możemy wybrać. Opcje niewspierane przez dany typ programatora nie będą dostępne (rysunek 2.9).

Rysunek 2.9.

*Opcje dostępne dla programatora AVRISP MkII. Ponieważ programator ten można łączyć z układem programowanym przy pomocy różnych interfejsów, właściwy tryb połączenia należy wybrać w menu *Programming Mode and Target Settings*. Dla tego programatora należy także określić częstotliwość zegara taktującego transmisję, pamiętając, że nie może ona być wyższa niż $\frac{1}{4}$ częstotliwości zegara taktującego rdzeń procesora*



W oknie konfiguracji można także ustawić konfigurację *fusebitów* oraz *lockbitów* (zakładki *Fuse* oraz *LockBits*) — rysunek 2.10.

W zakładce *Program* określa się ścieżki do plików zawierających program, który chcemy wczytać do mikrokontrolera (pliki muszą być w formacie *elf* lub *Intel HEX*) — rysunek 2.11.

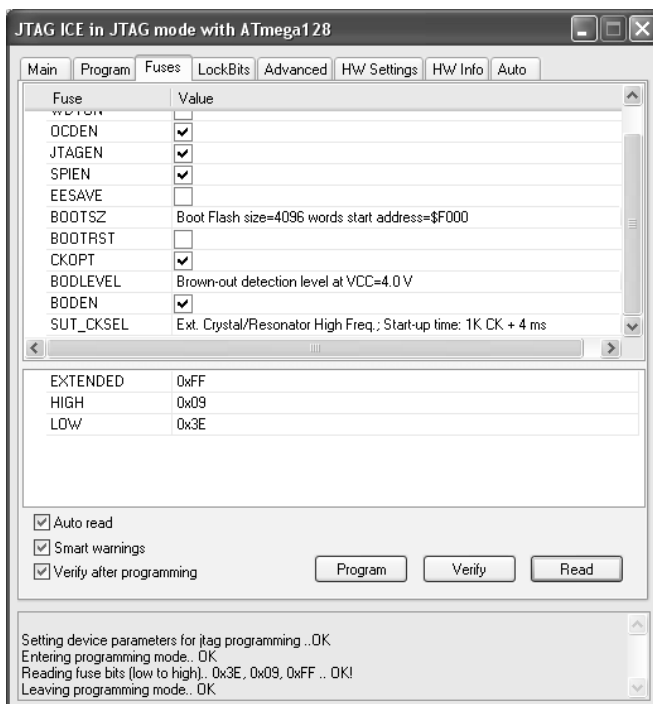
Programowanie przy pomocy narzędzi dostarczonych przez firmę Atmel

Firma Atmel wraz z AVR Studio dostarcza wielu różnych programów umożliwiających programowanie z linii poleceń przy pomocy programatorów kompatybilnych z protokołami firmy Atmel. Służą one generalnie do automatyzacji procesu programowania w przypadku programowania dłuższych serii procesorów. Wśród licznych programów na szczególną uwagę zasługuje program FLIP. Nie jest on dostarczany razem z AVR Studio, lecz wymaga osobnego pobrania ze strony www.atmel.com i instalacji. Wśród licznych

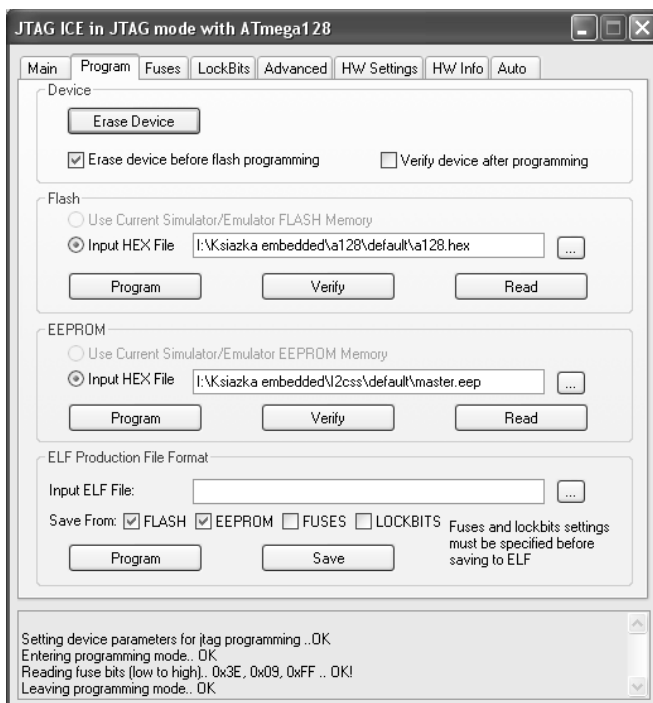
Rysunek 2.10.

Konfiguracja fusebitów w AVR Studio.

W przypadku złożonych operacji (np. wyboru zegara) zamiast wybierać konfigurację poszczególnych fusebitów, możemy posłużyć się rozwijalnymi listami z możliwymi do wybrania opcjami. W znacznym stopniu ogranicza to możliwość pomyłki. Wybraną konfigurację fusebitów wprowadza się do procesora po naciśnięciu przycisku Program

**Rysunek 2.11.**

W zakładce Program określa się ścieżki dostępu do plików wykorzystywanych w trakcie programowania. Najwygodniej jest użyć pliku w formacie elf, gdyż zawiera on wszystkie niezbędne do zaprogramowania procesora dane. Alternatywnie można podać ścieżki do plików hex i eep, zawierających wsad do zaprogramowania pamięci FLASH i EEPROM



jego możliwości jest także możliwość programowania urządzeń wyposażonych w *bootloader* kompatybilny ze specyfikacją Atmela dla urządzeń klasy DFU (ang. *Device Firmware Update*). Do tej klasy urządzeń zalicza się m.in. procesory AVR wyposażone w sprzętowy interfejs USB. Są one sprzedawane z firmowo wgranym *bootloaderem*, umożliwiającym wczytanie oprogramowania do pamięci FLASH i EEPROM mikrokontrolera.



Tryb DFU nie umożliwia zmiany *fusebitów*. W tym celu należy posłużyć się innym programatorem.

Aby uruchomić wbudowany w urządzenie *bootloader*, podczas wyprowadzenia urządzenia ze stanu *RESET* należy zewrzeć do masy pin HWB. Dzięki temu zamiast programu zostanie uruchomiony *bootloader* umożliwiający wczytanie nowego oprogramowania.

Po podłączeniu programowanego układu do komputera przy pomocy USB i uruchomieniu *bootloadera* przy pomocy pinu HWB urządzenie jest gotowe do programowania.



Aby klasa DFU była rozpoznawana przez komputer, należy zainstalować sterowniki DFU dostarczone przez firmę Atmel.

Po uruchomieniu programu FLIP wybieramy z menu *Device/Select*; w efekcie ukazuje się okno wyboru procesora (rysunek 2.12).

Rysunek 2.12.

Okno wyboru procesora. Będzie on programowany przy pomocy *bootloadera* w trybie DFU

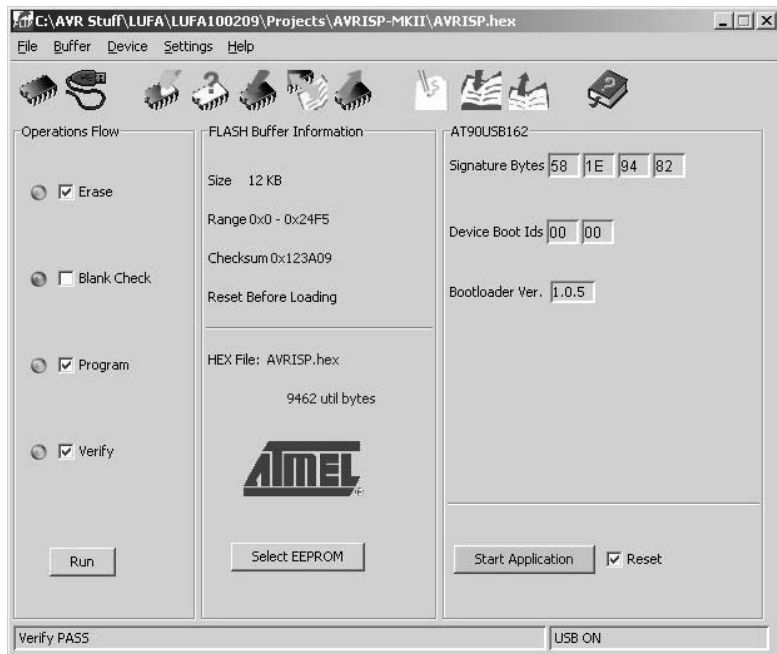


Po wyborze procesora klikamy na *Open*, co powoduje nawiązanie połączenia z programowanym układem. Następnie wczytujemy pliki do zaprogramowania (*File/Load HEX*), wybieramy opcje programowania i weryfikacji układu i klikamy na przycisk *Run*, co inicjuje proces uaktualniania oprogramowania (rysunek 2.13).

Program AVRDUDE

Jest to jeden z najpopularniejszych programów używanych do programowania mikrokontrolerów AVR. Jest on dostarczany wraz z pakietem WinAVR. Sam program AVRDUDE jest aplikacją uruchamianą z wiersza poleceń, parametry podaje się jako opcje wywołania.

Rysunek 2.13.
*Proces uaktualniania
oprogramowania
przy pomocy
programu FLIP*



Aby uczynić go nieco bardziej przyjaznym, w Internecie dostępnych jest wiele graficznych nakładek, umożliwiających uzyskanie tych samych efektów przy pomocy prostego interfejsu graficznego. Program AVRDUDE obsługuje następujące programatory:

- ◆ STK500, STK600,
- ◆ AVRISP i AVRISP mkII,
- ◆ AVRICE i AVRICE mkII,
- ◆ proste programatory podłączane do wyjścia równoległego i szeregowego komputera.

Program ten wspiera wszystkie protokoły transmisji używane przez firmę Atmel. Niestety, do programowania można posłużyć się wyłącznie plikami *IntelHEX*, gdyż nie wspiera on formatu *elf*. W efekcie musimy dysponować oddzielnymi plikami zawierającymi obrazy pamięci FLASH, EEPROM, a także wartościami numerycznymi *fuse- i lockbitów*. Stwarza to pewne dodatkowe możliwości pomyłki.

Program ten może pracować w dwóch trybach — terminalowym oraz wywoływany z wiersza poleceń. Poniżej krótko pokazane zostaną podstawowe opcje wywołania, umożliwiające zaprogramowanie przy jego pomocy mikrokontrolera.

Parametry wywołania:

- ◆ *-p procesor* — jest to obowiązkowy parametr wywołania programu. Określa on typ procesora podłączonego do programatora. Listę dostępnych typów można wyświetlić, wydając polecenie *avrdude -p ?*. W efekcie powinna wyświetlić się lista wspieranych typów procesorów.

- ♦ *-B okres* — parametr ten jest używany przy programowaniu za pomocą interfejsu JTAG w trybie ISP. Umożliwia on określenie prędkości programowania poprzez podanie okresu (w mikrosekundach) sygnału *SCK*. Np. `avrdude -B 1` powoduje, że linia *SCK* będzie taktowana sygnałem o częstotliwości 1 MHz.
- ♦ *-c programator* — określa typ programatora, który ma zostać użyty do programowania mikrokontrolera. Listę dostępnych typów można wyświetlić poleceniem `avrdude -c ?`. Na liście tej należy odnaleźć używany programator.



Wskazówka

Czasami dany programator wspiera różne protokoły programowania. W takiej sytuacji będzie występował na liście wiele razy z sufiksami określającymi wybrany tryb programowania.

- ♦ *-F* — powoduje, że program nie weryfikuje sygnatury układu z typem podanym jako parametr *-p*. W nielicznych sytuacjach umożliwia to obejście pewnych problemów związanych z uszkodzeniem sygnatury procesora, lecz normalnie opcja ta nie powinna być używana.
- ♦ *-n* — wykonuje wszystkie operacje, ale bez fizycznego zapisu do układu. Jest to przydatne do testowania różnych skryptów automatyzujących proces programowania.
- ♦ *-O* — przeprowadza kalibrację wewnętrznego generatora RC zgodnie z opisem z noty AVR053. Uzyskany w wyniku kalibracji bajt kalibracyjny jest zapisywany do komórki pamięci EEPROM o adresie 0, skąd może zostać odczytany przez program i użyty do kalibracji rejestru *OSCCAL* mikrokontrolera. Co prawda operacja ta nie poprawia stabilności wewnętrznego generatora RC, ale określa dokładniej jego częstotliwość.
- ♦ *-U obszar:typ:plik[:format]* — opcja ta przeprowadza operację na wskazanym obszarze (może to być operacja odczytu lub zapisu). Parametr *obszar* może być jednym z symboli: *eeeprom*, *flash*, *fuse*, *hfuse*, *lfuse*, *efuse*, *lock*. Określa on obszar podlegający danej operacji, zgodnie z nazwą podanych symboli. Parametr *typ* określa typ operacji: *r* — odczyt, *w* — zapis, *v* — weryfikacja, Parametr *plik* określa nazwę pliku, z którego będą odczytywane dane w przypadku operacji zapisu lub do którego będą zapisywane dane w przypadku operacji odczytu. Ostatni parametr, *format*, określa format pliku. Z licznych formatów istotne są *i* — określający, że plik jest w formacie *IntelHEX*, *i m* — określający, że parametr będzie wartością bezpośrednią, podaną w linii wywołania (najczęściej używane do programowania *fuse*- i *lockbitów*).

Wywołanie:

```
avrdude -p m88 -u -U flash:w:test.hex -U eeprom:w:test.eep -U efuse:w:0xff:m -U
hfuse:w:0x89:m -U lfuse:w:0x2e:m
```

spowoduje zaprogramowanie procesora ATmega88 plikami *test.hex* i *test.eep*, których zawartość zostanie umieszczona odpowiednio w pamięci FLASH i EEPROM mikrokontrolera. Dodatkowo wartość *fusebitów* zostanie ustawiona na 0x2E89FF. Z kolei wywołanie:

```
avrdude -c avrisp2 -P usb -p t26 -U flash:w:main.hex:i
```

powoduje zaprogramowanie procesora typu ATtiny26 plikiem *main.hex*, przy pomocy programatora AVRISP mkII, podłączonego przez port USB, w trybie ISP.

Program PonyProg

Program PonyProg jest bardzo prostym programem umożliwiającym programowanie różnych układów przy pomocy prostych interfejsów podłączanych do portu równoległego lub szeregowego komputera. Obsługuje on pliki *Intel Hex*. Program można pobrać ze strony <http://www.lancos.com/prog.html>. Oprócz programowania procesorów przy jego pomocy można także programować różnego typu pamięci szeregowo. Po skonfigurowaniu typu posiadanego programatora (opcja *Setup/Interface Setup*) należy wybrać typ programowanego układu. Następnie z menu *File* wybieramy *Open Program File* oraz *Open Data File* i wczytujemy uzyskane w trakcie kompilacji pliki z rozszerzeniem *hex* i *eep*. Ostatnią czynnością jest zaprogramowanie procesora poleceniem *Command/Write All*. PonyProg umożliwia także konfigurację *fuse-* i *lockbitów*. W tym względzie jego prostota prowadzi często do błędów. Do dyspozycji mamy tylko pojedyncze *fusebity*, których odpowiednią wartość należy ustalić po przejrzeniu sekcji noty katalogowej procesora poświęconej konfiguracji *fusebitów*.



Wskazówka

Należy pamiętać, że podobnie jak w przypadku innych programów, *fusebit* zaprogramowany oznacza *fusebit* o wartości 0.

Po skonfigurowaniu *fusebitów* wybieramy opcję *Write*, co powoduje ich zapisanie do procesora.

Fusebity i lockbity w AVR-libc

Biblioteka AVR-libc udostępnia wygodny sposób modyfikacji bitów konfiguracyjnych procesora. Ponieważ bity te nie mogą być zmieniane programowo, aby taka konfiguracja była możliwa, potrzebne jest specjalne oprogramowanie wspierające funkcje biblioteki AVR-libc. Zwykle wspierają taką możliwość programy, które jako źródło danych do programowania procesora wykorzystują pliki w formacie *elf*. W plikach w formacie *Intel HEX* nie ma możliwości umieszczenia informacji o konfiguracji *fuse-* i *lockbitów*; w efekcie programatory wykorzystujące ten format nie wspierają funkcji AVR-libc. W takiej sytuacji pozostaje ręczna konfiguracja tych bitów poprzez wybranie odpowiednich opcji programatora.



Wskazówka

Programując *lock-* i *fusebity*, należy pamiętać, że wartości jeden odpowiada *fusebit* niezaprogramowany, natomiast wartości 0 — zaprogramowany.

Aby *fusebity* i *lockbity* zostały poprawnie skonfigurowane i umieszczone w wynikowym pliku *elf*, należy wybrać właściwy typ procesora. Błędne ustawienie typu procesora może spowodować jego zablokowanie na skutek próby wpisania nieprawidłowej konfiguracji *fuse-* i *lockbitów*.

Lockbity

Lockbity zostały dokładnie omówione w rozdziale 25. Ich funkcją jest ochrona pamięci mikrokontrolera przed możliwością jej odczytania przy pomocy programatora. Dzięki temu umieszczony w pamięci FLASH program po zaprogramowaniu *lockbitów* nie daje się odczytać. Ich zaprogramowanie nie blokuje możliwości komunikacji z procesorem, przy próbie odczytu zwracane są wartości będące kolejnymi adresami komórek pamięci FLASH. Natomiast w żaden sposób nie da się odczytać ich zawartości, mimo że programator nie zasygnalizuje żadnego błędu. Raz zaprogramowane *lockbity* można skasować wyłącznie razem z kasowaniem pamięci FLASH i EEPROM poleceniem *Chip Erase*. W ten sposób odzyskujemy możliwość programowania i odczytywania zawartości pamięci FLASH procesora, lecz jednocześnie tracimy zawarty w niej poprzednio program. Zwykle właściwa konfiguracja *lockbitów* określana jest na końcowym etapie tworzenia urządzenia. Nie ma sensu ich używać w trakcie pisania aplikacji.

Fusebity

Wszystkie procesory AVR posiadają tzw. *fusebity*, umożliwiające określenie konfiguracji początkowej procesora po włączeniu zasilania. W zależności od modelu procesora dostępne *fusebity* mogą się nieznacznie różnić, oferując więcej lub mniej opcji konfiguracyjnych. Poniżej przedstawiona zostanie krótko charakterystyka poszczególnych *fusebitów*. Należy pamiętać, że nowa konfiguracja *fusebitów* zaczyna obowiązywać dopiero po wyjściu z trybu programowania procesora. Dzięki temu jeśli przypadkowo wprowadzono nieprawidłową konfigurację *fusebitów*, to można ją poprawić, o ile procesor nadal znajduje się w trybie programowania.

Fusebity BODLEVEL

Są one odpowiedzialne za konfigurację układu odpowiedzialnego za detekcję awarii zasilania. Jeśli napięcie zasilające procesor będzie poniżej progu wyznaczonego wartością *fusebitów* BODLEVEL, procesor utrzymywany będzie w stanie resetu do czasu, aż napięcie wróci do wartości prawidłowych. Domyślnie ich konfiguracja odpowiada zablokowanemu układowi detekcji awarii zasilania. **W gotowym układzie właściwie w każdej sytuacji należy włączyć ten układ ochronny.** Zapobiega to pracy procesora przy napięciach spoza specyfikacji, co może doprowadzić do nieprawidłowej pracy rdzenia procesora i układów peryferyjnych. Często spotykanym problemem przy wyłączonym układzie BOD jest uszkodzenie komórek pamięci EEPROM. Włączenie układu BOD praktycznie eliminuje ten problem. Układ BOD można wyłączać okresowo w systemach, w których pobór mocy jest szczególnie istotny. Jego wyłączenie nieznacznie zmniejsza pobór energii przez procesor. Szczegółowo zostało to omówione w rozdziale poświęconym trybom oszczędzania energii.

Fusebit WDTON

Jego zaprogramowanie powoduje włączenie układu *watchdog*. W takiej sytuacji układ ten nie może zostać wyłączony. O konsekwencjach włączenia układu *watchdog* szerzej napisano w rozdziale 5.

Fusebit EESAVE

Ma on znaczenie tylko podczas kasowania pamięci procesora przed programowaniem. Jego zaprogramowanie powoduje zachowanie zawartości pamięci EEPROM, zawartość pamięci FLASH nadal będzie mogła być normalnie kasowana. *Fusebit* ten ma zastosowanie w sytuacjach, w których wgrywana jest przy pomocy programatora nowa zawartość pamięci FLASH, a jednocześnie nowy program ma mieć dostęp do danych umieszczonych w pamięci EEPROM przez program poprzedni. Poza sytuacją programowania procesora przez programator, *fusebit* EESAVE nie ma znaczenia.

Fusebity BOOTSZ i BOOTRST

Ich znaczenie zostało szerzej omówione w rozdziale 25.

Fusebit JTAGEN

Fusebit ten umożliwia wyłączenie interfejsu JTAG. Domyślnie procesory sprzedawane są z zaprogramowanym *fusebitem* JTAGEN (o ile posiadają interfejs JTAG). **Odblokowanie układu JTAG powoduje przejście kontroli nad pinami I/O wspólnymi z tym interfejsem, co jest częstą przyczyną pomyłek.** Nad takimi pinami nie ma żadnej kontroli ze strony programu. Jeśli interfejs JTAG nie jest używany, można go wyłączyć, dzięki czemu wykorzystywane przez niego piny I/O zostaną zwolnione i będą mogły zostać wykorzystane w programie.

Fusebit SPIEN

Fusebit SPIEN odblokowuje interfejs ISP procesora. Domyślnie procesory są sprzedawane z zaprogramowanym *fusebitem* SPIEN, dzięki czemu można je programować przy pomocy programatorów szeregowych. Tego *fusebitu* nie można skasować w trybie programowania szeregowego — można to uczynić np. w trybie programowania poprzez interfejs JTAG lub w trybie wysokonapięciowym.

Fusebit CKDIV8

Określa on częstotliwość taktowania procesora. Domyślnie procesory sprzedawane są z zaprogramowanym *fusebitem* CKDIV8, w efekcie zegar taktujący jest dzielony przez 8, co prowadzi do częstych pomyłek — np. wyliczone pętle opóźniające są 8-krotnie dłuższe. Jego zaprogramowanie powoduje wpisanie po *resecie* do rejestru preskalera zegara (CLKPR) wartości odpowiadającej podziałowi przez 8. Zamiast kasować ten *fusebit*, można programowo zmienić wartość preskalera.



Przykładowe programy przedstawione w dalszej części książki zakładają, że *fusebit* CKDIV8 ma wartość 1 — ponieważ nie jest to domyślna wartość tego *fusebitu*, należy przed uruchomieniem aplikacji go przeprogramować.

Fusebity SUT

Określają one liczbę cykli zegara po włączeniu zasilania, po których procesor będzie wyprowadzony ze stanu reset. W większości przypadków ich konfiguracja nie ma znaczenia, tym bardziej że ich wartością domyślną jest najdłuższy możliwy czas wyjścia z resetu. W przypadku kiedy zasilanie procesora szybko ulega po włączeniu stabilizacji, czas ten można skrócić.

Fusebit CKOUT

Powoduje on wyprowadzenie na wyjście procesora *CKOUT* zbuforowanego zegara taktującego rdzeń. Dzięki temu inne układy mogą korzystać z zegara procesora, umożliwia to także synchroniczną pracę innych układów z procesorem. Domyślnie ten *fusebit* nie jest zaprogramowany, w efekcie wyjście *CKOUT* zachowuje się jak normalny pin portu *IO*.

Fusebity CKSEL

Określają one sposób generowania sygnału zegarowego dla procesora. Prawdopodobnie są to *fusebity* sprawiające najwięcej kłopotów, gdyż ich nieprawidłowe ustawienie może doprowadzić do zablokowania procesora (niemożność dalszego programowania w trybie ISP). Domyślną ich wartością jest wartość wybierająca jako źródło zegara wewnętrzny generator RC. W wielu przypadkach to domyślne ustawienie jest wystarczające i nie ma potrzeby go zmieniać. Potrzeba taka zachodzi, jeśli chcemy taktować procesor przy pomocy zewnętrznego układu generującego zegar lub przy pomocy kwarcu. **Konfiguracja tych bitów jest zależna od procesora i przed ich zmianą należy skonsultować się z notą katalogową używanego układu.**

Jeśli zdarzy się nam wybrać niewłaściwe źródło zegara, procesor można „uratować”, doprowadzając zewnętrzny przebieg zegarowy do wejścia *XTAL1*. Dzięki temu możliwe będzie nawiązanie komunikacji z programatorem i ponowne przeprogramowanie *fusebitów*.

Mikrokontrolery AVR mogą być taktowane z trzech źródeł zegara:

1. Zegara zewnętrznego doprowadzonego do wejścia *XTAL1*. Jest to rzadko wykorzystywana możliwość. W tym trybie potrzebny jest zewnętrzny generator zegara, którym może być np. inny procesor AVR. Umożliwia to synchronizację pracy obu kontrolerów.
2. Zegara wewnętrznego (układ generatora RC). W tym trybie (jest on trybem domyślnym) procesor jest taktowany z własnego generatora, w efekcie nie trzeba doprowadzać zewnętrznego przebiegu zegarowego. Wadą generatora wewnętrznego jest jego stosunkowo niewielka stabilność. W efekcie nie może on być wykorzystywany np. w programach, w których wymagana jest duża stabilność zegara.

3. Zewnętrzny rezonator kwarcowy. W tym celu do pinów *XTAL1* i *XTAL2* należy podłączyć rezonator kwarcowy o pożądanej częstotliwości. Tryb ten umożliwia taktowanie procesora ze stabilnego źródła zegara.

Fusebit RSTDISBL

Zwykle pin RESET procesora współdzieli wyprowadzenie w pinem IO. Aby móc wykorzystać pin RESET jako normalny pin IO, należy zaprogramować *fusebit* RSTDISBL. **Po jego zaprogramowaniu dalsze programowanie procesora w trybie szeregowym jest niemożliwe.** Procesor można nadal programować programatorem wysokonapięciowym lub równoległym.

Fusebit DWEN

Jego zaprogramowanie uruchamia możliwość wykorzystania interfejsu debugWire, przy pomocy którego można programować procesor oraz w ograniczonym stopniu debugować program. Tylko nieliczne procesory dysponują tą funkcją, w dodatku wymaga ona posiadania specjalnego programatora obsługującego debugWire.

Sygnatura

Wszystkie procesory AVR dysponują unikalną sygnaturą nadawaną im w czasie procesu produkcji. Sygnatura ta określa typ procesora i ilość pamięci. Dzięki temu programator, odczytując sygnaturę, może weryfikować, czy podłączony procesor odpowiada procesorowi wybranemu przez użytkownika. **Sygnatura nie zawiera unikalnego dla konkretnego chipu numeru seryjnego.**

Lockbity w AVR-libc

Definicje wspieranych przez dany model procesora *lockbitów* zawiera plik nagłówkowy *<avr/io.h>*. Po jego włączeniu można włączyć plik *<avr/io.h>* zawierający definicję różnych makrodefinicji związanych z konfiguracją *lockbitów* dla danego procesora. Po włączeniu do programu dostajemy do dyspozycji makrodefinicję LOCKBITS, której można przypisać pożądaną kombinację *lockbitów*. Domyślną wartością tego makra jest 0xFF; w efekcie wszystkie *lockbity* pozostają niezaprogramowane. Jeśli chcemy to zmienić, należy symbolowi LOCKBITS przypisać nową wartość, najlepiej posługując się predefiniowanymi w pliku *<avr/io.h>* symbolami. **Poszczególne symbole można łączyć ze sobą przy pomocy iloczynu bitowego.** Przykładowo:

```
LOCKBITS=(BLB1_MODE_3 & LB_MODE_3);
```

spowoduje wybranie trybu 3 dla kodu aplikacji oraz *bootloadera*. Więcej o trybach ochrony pamięci przy pomocy *lockbitów* znajdziesz w rozdziale 25.

Makrodefinicję `LOCKBITS` można zainicjować tylko raz, przyporządkowanie jej innej wartości w dalszych fragmentach programu nie odnosi żadnego skutku.

Wartość ustawionych *lockbitów* dla skompilowanego programu można odczytać z pliku *elf* przy pomocy polecenia:

```
avr-objdump -s -j .lock <plik ELF>
```

W efekcie dla wartości *lockbitów* określonej powyżej dla procesora ATmega88 powinniśmy uzyskać taki rezultat:

```
test.elf:      file format elf32-avr
```

```
Contents of section .lock:
830000 cc
```

Wartość *lockbitów* wynosi więc `0xCC`.

Fusebity w AVR-libc

Podobnie jak w przypadku *lockbitów*, *fusebity* określone w programie zostaną umieszczone w specjalnej sekcji pliku *elf*, skąd będą mogły zostać odczytane przez program sterujący programatorem i użyte do konfiguracji procesora. Plik `<avr/fuse.h>` zawiera definicje przydatne przy niskopoziomowym manipulowaniu *fusebitami*. Plik nagłówkowy `<avr/io.h>` zawiera definicje *fusebitów* używanych przez wybrany model procesora. W zależności od typu procesora *fusebity* mieszczą się w jednym, dwóch lub trzech bajtach. Ilość bajtów przeznaczonych na *fusebity* zwraca makro `FUSE_MEMORY_SIZE`. Definiowanie *fusebitów* następuje poprzez przypisanie wartości makru `FUSES`, np.:

```
FUSES =
{
    .low=LFUSE_DEFAULT,
    .high=(FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_EESAVE & FUSE_SPIEN),
    .extended=EFUSE_DEFAULT,
};
```

Jak widzimy, makro to ma trzy pola odpowiadające poszczególnym bajtom przechowywującym *fusebity*. Dla procesorów, w których makro `FUSE_MEMORY_SIZE` zwraca wartość jeden, dostępne jest tylko pole `.low`, dla procesorów, w których powyższe makro zwraca 2, dostępne są pola `.low` i `.high`, w pozostałych procesorach dostępne jest także pole `.extended`. Niestety, wykorzystując powyższe definicje, należy pamiętać, w którym bajcie jakie *fusebity* są przechowywane. Ich pomylenie spowoduje nieprawidłowe zaprogramowanie procesora, co może się nawet skończyć jego zablokowaniem. Podobnie jak w przypadku *lockbitów*, ważne jest tylko pierwsze przypisanie wartości do makra, kolejne są ignorowane. Stąd też najlepiej taką definicję umieścić raz, na początku programu.

Podobnie jak w przypadku *lockbitów*, wyliczone wartości *fusebitów* można odczytać z pliku *elf*:

```
avr-objdump -s -j .fuse test.elf
```

```
ADCNoiseReduction.elf:      file format elf32-avr
```

```
Contents of section .fuse:
```

```
820000 62d1f9                                b..
```

Wyświetlona wartość odpowiada poszczególnym bajtom przechowującym *fusebity*. Dla procesorów, w których mieszczą się one w mniejszej ilości bajtów, pokazanych zostanie ich mniej. **Co ważne, najmłodszy bajt wyświetlany jest z lewej, najstarszy z prawej strony.**

Skorowidz

2-wire Serial Interface, *Patrz*
interfejs TWI

A

ADC Noise Reduction Mode,
Patrz tryb redukcji szumów
adres, 104, 141, 142, 144, 146,
147, 148
akcelerometr, 391
algorytm
XMODEM, 512
arytmetyka
stałopozycyjna, 15, 81, 83, 85
zmiennopozycyjna, 81, 82, 87
assembler, 142, 252, 254, 529,
536, 538
Atmel, 15, 17, 55, 64, 82, 456,
464, 497, 507, 552
atomowość, *Patrz* dostęp
atomowy
atrybut
always_inline, 549
const, 549
deprecated, 50
flatten, 549
ISR_NAKED, 254
naked, 145, 550
noclone, 550
noinline, 550
nonnull, 550
noreturn, 551
optimize, 551
PROGMEM, 142, 349
pure, 549
SIGNAL, 251

used, 48
volatile, 212
warn_unused_result, 551
weak, 33, 50
AVR Studio, *Patrz* program
AVR Studio

B

biblioteka, 46, 48
AVR-libc, 15, 70, 74, 75,
137, 138, 141, 145, 147,
152, 157, 159, 181, 183,
190, 211, 217, 227, 249,
267, 483, 520
Joerga Wunscha, 336
kolejność przeszukiwania, 49
libc, 93
libm.a, 88
libprintf_float, 93
libprintf_min, 93
libusb, 460
LUFA, 464
zewnętrzna, 88
bity zabezpieczające BLB, 485,
486, 487
błąd e, 89
błąd reprezentacji, 89
bootloader, 53, 67, 146, 249,
464, 483, 487, 489, 496, 499,
504, 507, 509, 520
bootsektor, 484
breakpoint, *Patrz* pułapka
Brown-out Detector, 151
Brown-out Reset Circuit, *Patrz*
reset Power-on Reset
bufor, 393
74XX244, 57

cykliczny, 186
wejściowy, 160
wyjściowy, 56
bus keeper, 193

C

Camer Dean, 464
Cesko Igor, 459
char, *Patrz* typ danych znakowe
Clear Timer on Compare Match,
Patrz tryb CTC
Clock Prescale Register, *Patrz*
rejestr CLKPR
CodeBlocks, 18
Compare Match, 309, 314, 318
Complex Data Types, *Patrz* typ
danych złożony, *Patrz* typ
danych prosty
Controller Area Network, *Patrz*
interfejs CAN
Counter, *Patrz* licznik
CRC, 511, 514, 520, 521
Cyclic Redundancy Code, *Patrz*
CRC
cykl oczekiwania, 193, 194
czas martwy, *Patrz* generator
czasu martwego
częstotliwość, 22, 58, 59, 72,
149, 159, 178, 217, 218, 306,
316, 317, 318, 319, 323, 417
próbkiowania, 283
czujnik temperatury, 391

D

Daisy-chain JTAG mode, *Patrz* konfiguracja łańcuchowa

Dead Time Generator, *Patrz* generator czasu martwego

debugowanie, 60, 61, 63, 64, 74, 101, 141, 161, 164, 529, 544, 552, 553, 554, 556

decymacja, 292

definicja, 128, 130

 EEMEM, 182

 symbolu, *Patrz* symbol

 TW_STATUS_MASK, 416

deklaracja, 128, 130

Device Firmware Update, *Patrz* urządzenie DFU

Device Firmware Uploader, 507

DFU, 67

Digital To Analog Converter, *Patrz* przetwornik DAC

dioda LED, 53, 230, 231, 275, 332, 333

disasemblacja, 556

długość kodu, 31, 32

dostęp atomowy, 263, 264, 268

dyrektywa

 %elif, 40

 %else, 40

 %if, 40

 #define, 103, 104, 126

 #elif, 125

 #else, 125

 #endif, 125

 #if, 124

 #ifdef, 124, 125

 #ifndef, 124, 125

 #include, 124, 128

 defined, 125

 extern, 135

 inline, 132

 kompilacji warunkowej, 124

 warunkowa, 40

E

eavesdropping, 518

EEPROM Address Register, *Patrz* rejestr EEAR

EEPROM Control Register, *Patrz* rejestr EECR

EEPROM Data Register, *Patrz* rejestr EEDR

ekspander, 403, 404, 436

enkoder obrotowy, 230, 237, 242, 279

Executable and Linkable Format, *Patrz* plik elf

External Memory Interface, *Patrz* interfejs XMEM, *Patrz* interfejs XMEM

F

falsz, *Patrz* typ danych bool

Fault Protection Interrupt Enable, *Patrz* flaga FPIE

Fault Protection Interrupt Flag, *Patrz* flaga FPF

Fault Protection Unit, *Patrz* układ ochronny

Ferroelectric RAM, *Patrz* pamięć FRAM

flaga, 251, 254

 bitowa, 246

 Busy, 336

 FPF, 322

 FPIE, 322

 ICF, 311, 319

 OCF, 309, 319

 RXCIE, 375

 SPIF, 396

 TOV, 319

 TWIE, 417

 TXCIE, 375

 TXCO, 476

 UDRIE, 375

 USIOIE, 449

 USIOIF, 452

 USISIE, 449

 WCOL, 396

FLexible In-system Programmer, *Patrz* program FLIP

Frame, *Patrz* ramka

Free Running Mode, *Patrz* tryb ciągłej konwersji

FTDI, 456, 457

funkcja, 47, 108, 112, 126, 132, 535, 549, 550, 551

 _delay_loop_, 219

 _delay_ms, 217, 219

 _delay_us, 217, 219

 adres, 144

 asynchroniczna, 249

 atof, 91

 bus keeper, 194, 195

 calloc, 166

dtostre, 90

dtostrf, 91

EEPROM_busy_wait, 185

EEPROM_is_ready, 185

EEPROM_read_, 183, 184

EEPROM_update_, 183, 185

EEPROM_write_, 183, 184

fprintf, 95

free, 165, 168, 171

fscanf, 95

inline, 134, 135, 549

main, 144, 145

malloc, 165, 166, 506

MD5, 512

modulująca, 322

nie reentrant, 267

obsługa przerwań, 101, 251

opóźniająca, 217

parametry, 114

printf, 92, 95

prototyp, 113, 128, 130

przeciążanie, 113

Read_4kB, 196

realloc, 166

reentrant, 249, 250, 266

reentry, 170

rekurencyjna, 115, 165, 266

scanf, 95

SHA, 512

ShowOnLED, 271

snprintf, 95

sscanf, 94

static inline, 127

statyczna, 135

strsep, 107

strtod, 92

strtok_r, 107

switch, 144

void, 251

wdt_enable, 145

Write_4kB, 196

wywołanie, 114, 537

fusebit, 53, 55, 61, 64, 65, 70, 71, 73, 75, 146, 150, 524

BODLEVEL, 71, 151, 161

BOOTRST, 72

BOOTSZ, 72, 513

CKDIV8, 22, 72, 159

CKOUT, 73

DWEN, 74, 161

EESAVE, 72, 178

HWBE, 497

JTAGEN, 72

RSTDISBL, 74, 151

SPIEN, 72
 SUT, 73
 WDTON, 72, 152, 153, 161

G

General Purpose IO Registers,
Patrz rejestr IO ogólnego
 przeznaczenia
 generator
 czasu martwego, 320
 kwarcowy, 330
 wewnętrzny, 73, 307, 308
 zdarzeń, 311
 zewnętrzny, 73, 308
 GNU/Linux, 18

H

Heap, *Patrz* sverta
[http://dfu-programmer.
 ↳sourceforge.net/](http://dfu-programmer.sourceforge.net/), 497
[http://realterm.sourceforge.
 ↳net//](http://realterm.sourceforge.net//), 375
[http://sourceforge.net/projects/
 ↳libusb-win32](http://sourceforge.net/projects/libusb-win32), 461
<http://svn.savannah.nongnu.org>,
 495
<http://winavr.sourceforge.net/>, 16
<http://www.atmel.com/>, 17
[http://www.atmel.no/
 ↳beta_ware/](http://www.atmel.no/beta_ware/), 15
[http://www.atmel.com/dyn/
 ↳resources/prod_
 documents/doc7618.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc7618.pdf), 508
<http://www.atmel.com>, 65
[http://www.atmel.com/dyn/
 ↳products/tools_card.asp?
 tool_id=3886](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3886), 497
[http://www.avrfreaks.net/
 ↳index.php? module=
 ↳Freaks%20Files&func
 ↳=viewFile&id=3330
 ↳&showinfo=1](http://www.avrfreaks.net/index.php?module=Freaks%20Files&func=viewFile&id=3330&showinfo=1), 520
<http://www.cesko.host.sk>, 459
<http://www.codeblocks.org>, 18
[http://www.ftdichip.com/
 ↳FTDrivers.htm](http://www.ftdichip.com/FTDrivers.htm), 458
[http://www.lancos.com/
 ↳prog.html](http://www.lancos.com/prog.html), 70
[http://www.obdev.at/products/
 ↳usb/download.html](http://www.obdev.at/products/usb/download.html), 461
<http://www.usb.org>, 456

I

input capture, 311
 Input/Output Ports, *Patrz* port
 wejścia/wyjścia
 instrukcja
 asm, 529, 535, 536
 break, 122
 continue, 123
 for, 217
 goto, 123
 if, 120
 kolejność, 263
 MOVW, 537
 NOP, 229
 opóźniająca, 229
 pętla do..while, 122
 pętla for, 122
 pętla while, 121
 powrotu, 251
 RET, 251
 RETI, 251
 sei, 158
 sleep, 157
 SPM, 509
 switch, 120
 WDR, 152
 In-system Programming
 Interface, *Patrz* interfejs ISP
 interfejs, 417, 448
 1-wire, 465, 469, 470, 472,
 477, 481
 aWire, 62
 binary, 530
 CAN, 367
 debugWire, 53, 55, 62, 63,
 74, 552
 HVPP, 63
 HVSP, 63
 I2C, *Patrz* interfejs TWI
 ISP, 53, 55, 58, 63, 72
 JTAG, 53, 55, 60, 61, 62,
 63, 69, 72, 173, 176, 221,
 552, 556
 pamięci zewnętrznej, 221
 PDI, 53, 62, 63
 równoległy, 367, 453
 RS232, 368, 370, 375, 378,
 386, 456, 458, 498
 RS485, 367, 383, 384, 386
 SPI, 55, 62, 367, 371, 391,
 393, 397, 405, 449, 498
 inicjalizacja, 394
 nadajnik, 393
 odbiornik, 393

synchroniczny, 391, 413
 szeregowy, 354, 367, 413
 TPI, 53, 58
 TWI, 367, 391, 416, 437, 438
 inicjalizacja, 417
 UART, 221, 370, 371, 386
 odbiornik, 386
 prędkość, 372, 373
 USART, 246, 367, 368, 374,
 375, 378, 379, 408, 469,
 472, 473, 475, 477
 nadajnik, 373, 374, 408
 odbiornik, 373, 374, 408
 USB, 53, 58, 67, 367, 453, 454,
 455, 456, 460, 497, 498
 programowy, 461
 sprzętowy, 464, 507
 USI, 447, 449, 472
 XMEM, 141, 144, 193
 interpolacja, 292
 Interrupt Service Routine, *Patrz*
 procedura obsługi przerwań

K

klawiatura, 295
 klawiatura matrycowa, 229, 230,
 242, 244, 281
 kod
 assemblerowy, 141
 binarny, 238
 Graya, 238, 239
 źródłowy, 141
 komentarz, 37
 komparator analogowy, 160,
 301, 311, 321
 kompilacja, 23, 26, 47, 64, 128
 warunkowa, 125
 kompilator, 123, 132
 avr-gcc, 15, 18, 82, 88, 543
 gcc, 88, 142, 211, 520
 IAR, 520
 kondensator odprężający, 55
 konfiguracja
 daisy-chain, *Patrz*
 konfiguracja łańcuchowa
 łańcuchowa, 62, 63
 początkowa, 71
 kontroler
 CAN, 391
 HD44780, 333
 KS0108, 355
 kontynuacja linii, 37

konwersja, 283, 285, 287, 288,
289, 291, 295
konwerter, 458
koprocessor arytmetyczny, 87, 88

L

licznik, 305, 308, 309, 311, 315,
318, 319, 323, 447, 452
linia, *Patrz* sygnał
linia
adresowa, 195
kontrolna, 193
lista modyfikowanych
rejestrów, 535
literal memory, 535
lockbit, 65, 70, 71, 146, 485,
486, 496, 512, 524
konfiguracja, 74
LB, 525

M

magistrala, 250
1-Wire, 163
RS232, 369
SCL, 413
SDA, 413
makrodefinicja, 39, 126, 535, 536
_BV, 227
ATOMIC_BLOCK, 245,
264, 265
BADISR_vect, 253
boot_page_erase, 487
boot_page_erase_safe, 487
boot_page_write, 488
boot_page_write_safe, 488
boot_rww_busy, 488
boot_spm_busy, 488
boot_spm_busy_wait, 488
cli, 254, 264, 265
clock_prescale_, 159
clock_prescale_set, 159
EMPTY_INTERRUPT, 253
FUSE_MEMORY_SIZE, 75
GET_FAR_ADDRESS, 192
GetAddr, 349
ISR, 251
LOCKBITS, 74
NONATO_MIC_BLOCK, 266
pgm_read_, 191
power_, 159
PROGMEM, 190
PSTR, 94
sei, 251, 254, 264, 265
set_sleep_mode, 157
sleep_bod_disable, 157
sleep_cpu, 157
sleep_disable, 157
sleep_enable, 157
sleep_mode, 157
TW_STATUS, 438
wdt_disable, 155
wdt_enable, 155
wdt_reset, 155
marker czasowy, 310
maska bitowa, 227, 246
master-slave, 379, 391, 413,
415, 437, 438, 439, 449, 451,
452, 465, 466, 467, 469, 477
MCU Control Register, *Patrz*
rejestr kontrolny
menu
Build, 23
File, 25
Fuse, 65
LockBits, 65
Program, 65
Project, 26, 30, 200, 218, 490,
543, 545
Tools, 64, 524
View, 555
miernik częstotliwości, 323
miernik wypełnienia, 323
mikrokontroler, *Patrz* procesor
modulacja, 316, 322
modulator sygnału
wyjściowego, 322
moduł przechwytywania
zdarzeń zewnętrznych, 310, 312
modyfikator
const, 103, 189, 245
extern inline, 135
inline, 503
register, 136, 137
static, 267
static inline, 135, 536
volatile, 103, 138, 196, 217,
228, 229, 245, 257, 259,
262, 279, 530
modyfikatory makr, 40
multiplexer, 302
multiplexer analogowy, 283, 285
multipleksowanie, 269, 275, 405
Multi-processor Communication
Mode, *Patrz* tryb
wieloprocessorowy MPCM

N

nadpróbkowanie, 291
napiecie, 56, 57, 58, 60, 63, 150,
151, 178, 231, 301
referencyjne, 160, 161, 283,
284, 297
różnica, 283, 287
No Read While Write, *Patrz*
pamięć NRWW
numer
współdzielony ID, 457
seryjny, 74, 519

O

Objective Development, 457
opcja
fdata-sections, 548
ffunction-sections, 548
finline-limit=n, 547
flto, 548
fmerge-constants, 548
fno-inline, 547
fwhole-program, 548
mcall-prologues, 546
mint8, 546
mno-interrupts, 547
mtiny-stack, 547
open-drain, 413
operacja binarna, 95
iloczyn bitowy, 96, 98
negacja bitowa, 99
przesunięcie bitowe, 100
rotacja, 100
suma bitowa, 97
suma wyłączająca, 98
operand, 531, 533
sposób dostępu, 531, 532
typ, 531, 532
wejściowy, 535
wyjściowy, 535
operator, 116
&, 105
*&, 105
.ascii, 540
.asciz, 540
.byte, 540
.section, 540
arytmetyczny, 116, 117
bitowy, 116, 118
dereferencji, *Patrz* operator *&
hi8, 540
kolejność, 118, 119

lo8, 540
 logiczny, 116, 118
 pm, 540
 porównań, 116, 117
 relacji, 89
 sizeof, 111
 wyluskania, *Patrz* operator *&
 opóźnienie, 199, 217, 219, 235
 dostępu, 193
 optymalizacja, 545, 548, 554
 optymalizator, 543
 Output Compare Register, *Patrz*
 rejestr OCR
 Oversampling, *Patrz*
 nadpróbkowanie

P

pakiet danych, 456
 pamięć, 34, 53, 74, 249
 adres, 148
 adresowanie, 77
 dynamiczna alokacja, 163,
 164, 165, 166, 167, 169,
 171, 173
 EEPROM, 61, 67, 71, 72, 77,
 143, 151, 177, 178, 179,
 180, 181, 182, 183, 185,
 186, 250, 422, 458, 497,
 554
 FLASH, 24, 60, 67, 71, 72,
 77, 94, 142, 146, 177, 189,
 190, 191, 483, 484, 488,
 496, 509, 554
 FLASH >64 kB, 192
 fragmentacja, 163, 164, 171
 FRAM, 427
 kasowanie, 72, 180
 mikrokontrolera, *Patrz*
 pamięć FLASH
 modyfikowanie, 185
 NRWW, 483
 odczyt, 180, 183, 191, 192
 RAM, 142, 148, 197
 RWW, 483
 SRAM, 24, 77, 94, 149, 177,
 189, 193, 194, 195, 198, 554
 szeregową, 391
 wewnętrzna, 148, 195, 197,
 198, 201
 wyciek, 163, 165, 169
 zapis, 180, 184

zewnętrzna, 29, 147, 148,
 193, 194, 195, 197, 198,
 199, 201, 208, 422, 427, 458
 zwalnianie, 169
 pętla, *Patrz* instrukcja
 pętla synchronizacji fazowej, 307
 Phase Locked Loop, *Patrz* pętla
 synchronizacji fazowej
 PID, 456, 457, 463
 pin, *Patrz* sygnał
 platforma sprzętowa, 19
 plik
 .S, 529, 536
 /avr-libc/trunk/avr-libc/crt1/
 gcr1.S, 495
 <avr/EEPROM.h>, 181
 <avr/fuse.h>, 75
 <avr/io.h>, 74, 533, 538
 <avr/boot.h>, 483
 <avr/EEPROM.h>, 185
 <avr/interrupt.h>, 249
 <avr/io.h>, 221, 393
 <avr/pgmspace.h>, 94, 190
 <avr/power.h>, 156, 159
 <avr/sleep.h>, 156
 <avr/wdt.h>, 145, 152
 <math.h>, 89
 <stdfix.h>, 83
 <stdio.h>, 92
 <stdlib.h>, 90, 91, 164
 <stdlib_private.h>, 171
 <util/crc16.h>, 512
 <util/twi.h>, 416
 <util/delay.h>, 217, 218
 <util/delay_basic.h>, 219
 defines.h, 336
 dodawanie, 25
 elf, 43, 65, 68, 75, 146, 178
 hd44780.c, 336
 Intel HEX, 45, 65, 68, 70,
 178, 554
 macros.inc, 495
 Makefile, 26, 34, 36, 49, 93,
 208, 218
 nagłówkowy, 35, 47, 127, 134
 obiektowy, 33, 47, 128
 opisu sprzętu, 20
 rozszerzenie, 25
 sectionname.h, 495
 skryptu linkera, 204
 string.h, 107
 usbconfig-prototype.h, 461
 wynikowy, 146
 źródłowy, 128

podsekcja, *Patrz* sekcja
 pointer, *Patrz* typ danych
 wskaźnik
 pooling, 181, 268 *Patrz* technika
 częstego próbkowania
 port, *Patrz* sygnał
 IO, 221, 225, 533
 równoległy, 56, 57, 70
 RS232, 58
 szeregowy, 19, 58, 70, 447
 UART, 250
 USB, 57, 58
 wejścia/wyjścia, 221, *Patrz*
 port IO
 wyjściowy, 309
 potencjał masy, 56
 prawda, *Patrz* typ danych bool
 preprocesor, 123
 PRESENCE PULSE, 466, 476,
 478
 preskaler, 152, 153, 155, 159,
 286, 306, 307, 308, 318, 321,
 324, 328, 417
 procedura obsługi przerwań,
 249, 251, 252, 253, 254, 256,
 260, 262, 266, 445, 449
 procesor
 architektura, 28, 203
 AT90USB, 464
 ATmega, 61, 128, 145, 464
 ATmega128, 13, 176, 195,
 199, 203, 486, 520
 ATmega256, 502, 504
 ATmega64, 193
 ATmega8, 520
 ATmega8-32U2, 497
 ATmega88, 13, 69, 75, 250,
 252, 397
 ATTiny, 58, 64, 151, 447
 ATTINY, 268
 ATTiny26, 70
 ATTiny44, 13
 AVR architektura, 204
 AVR Xmega, 58
 AVR32, 58, 60, 62
 częstotliwość, 21
 peryferia, 20
 programowanie, 55
 rdzeń, 20, 159
 rejestr, 136, 144
 resetowanie, 149, 253
 sygnatura, 146
 typ, 19, 74

- procesor
 - uśpienie, 157, 160, 161, 326, 327, 328
 - Product ID, *Patrz* PID
 - program
 - alokator, 171
 - alokatora pamięci
 - dynamicznej, 164
 - ar, 28
 - AVR Simulator 2, 552
 - AVR Studio, 19, 64, 94
 - AVR Studio 5, 15
 - AVRDude, 59, 67, 68
 - avr-gcc, 536
 - avr-nm, 32
 - avr-objcopy, 33, 46
 - avr-objdump, 43
 - avr-size, 31
 - dfu-programmer, 507
 - FLIP, 65, 497, 507
 - linker, 27, 47, 93
 - make, 18, 27, 28, 30, 33, 34, 36, 46
 - Makefile, 495, 514
 - narzędziowy, 27
 - PonyProg, 70
 - Programmer's Notepad, 205
 - RealTerm, 375
 - srec_cat, 512
 - Typer Terminal, 375
 - WinAVR, 15, 59, 67, 82, 205
 - wine, 18
 - winetricks, 18
 - programator, 19
 - AVR Dragon, 63, 64
 - AVRICE, 68
 - AVRICE mkII, 62, 68
 - AVRISP, 58, 64, 68
 - AVRISP mkII, 58, 64, 68
 - HW, *Patrz* programator
 - wysokonapięciowy
 - ISP, 55, 56
 - JTAG, 60
 - JTAGICE, 61
 - JTAGICE mkII, 62
 - równoległy, 53, 64, 74
 - STK500, 68
 - STK600, 68
 - szeregowy, 53, 72, 74
 - USBASP, 59
 - wysokonapięciowy, 54, 63, 72, 74
 - prototyp funkcji, 47
 - przerwanie, 152, 153, 155, 156, 157, 158, 170, 176, 180, 196, 202, 219, 245, 249, 250, 251, 252, 254, 257, 264, 328, 371, 402, 410, 448, 449, 452, 489, 505
 - ADC, 289, 290
 - komparatora, 301
 - OCIE, 309
 - SPI, 393, 402, 406, 407, 408
 - TIMERN_COMPx_vect, 309
 - TIMn_CAPT_vec, 311
 - TIMn_CAPT_vect, 318
 - TIMn_COMP_vect, 313, 318
 - TIMn_OVF_vect, 312, 315, 318
 - TWI, 417, 444
 - USART, 408
 - USARTn_, 375
 - USI, 448
 - przetwornik
 - ADC, 127, 160, 285, 287, 303, 317, 391
 - analogowo-cyfrowy, *Patrz* przetwornik ADC
 - cyfrowo-analogowy, *Patrz* przetwornik DAC
 - DAC, 317, 318, 319
 - Pulse Width Modulation, *Patrz* tryb PWM
 - pułapka, 176, 556, 557, 558
- R**
- ramka, 371, 373, 374, 386, 498
 - rdzeń, 73
 - Read Only Memory, *Patrz* pamięć ROM
 - Read While Write, *Patrz* pamięć RWW
 - Real-Time Clock, *Patrz* układ zegarowy
 - Reduction Register, *Patrz* rejestr PRR
 - rejestr
 - AC, 337
 - ACSR, 302
 - ADC, 284, 287, 288
 - ADCH, 212
 - ADCSRA, 160, 287, 288, 291, 303
 - ADCSRB, 303
 - ADMUX, 160, 212, 285, 287
 - CLKPR, 159
 - DDR, 373
 - DDRN, 247
 - DDRx, 221, 222, 226, 229, 322
 - DIDR0, 289
 - DIDRx, 160, 302
 - DT, 321
 - EEAR, 177, 180
 - EECR, 177, 180, 181
 - EEDR, 177, 180
 - GPIOR, *Patrz* rejestr IO
 - ogólnego przeznaczenia
 - ICR, 311
 - ICRN, 213
 - indeksowy, 138
 - IO, 159, 245, 246, 538
 - IO ogólnego przeznaczenia, 245, 246
 - kontrolny MCUCR, 194
 - kontrolny XMCRA, 194
 - kontrolny XMCRB, 194, 197
 - liczników, 211
 - MCUSR, 154
 - OCR, 309, 318, 319
 - OCRn, 213
 - OCRx, 247
 - PIN, 310
 - PINx, 221, 223, 224, 228, 229, 259
 - PLLCSR, 307
 - PORT, 373
 - PORTn, 247
 - PORTx, 221, 222, 223, 226, 229, 322
 - preskalera, 306
 - procesora, 144, 554
 - PRR, 158
 - przesuwający, 403
 - przesuwny, 405
 - przetwornika ADC, 211
 - R0, 538
 - R1, 490, 538
 - R18-R27, 538
 - R25, 537
 - R2-R17, 538
 - SMCR, 157
 - SPCR, 394
 - SPDR, 396
 - SPSR, 394
 - SREG, 251
 - stosu, 144
 - szeregowy, 403, 405
 - TCCR, 309, 321
 - TCCRNb, 311
 - TCNT, 309, 319

TCNTn, 213
 TIFR, 308, 309, 311
 TWAMR, 437
 TWAR, 246, 437
 TWBR, 417, 438
 TWCR, 417
 TWSR, 416, 417, 438
 tymczasowy, 213, 535, 538
 UBRR, 371, 408
 UBRRH, 246
 UBRRL, 246
 UCSRA, 371, 374, 387
 UCSRB, 373
 UDR, 374
 USICR, 448, 450
 USIDR, 447, 451
 USISR, 450
 WDTCR, 153
 Y, 538
 zatraskowy, 194
 repeter, 414
 reset, 54, 61, 67, 71, 73
 Power-on Reset, 150
 zewnętrzny, 151
 RESET PULSE, 466, 476, 478
 rezonator kwarcowy, 22
 rezystor, 13, 231, 233, 239, 296
 podciągający, 413, 416, 465
 rozdzielczość, 319
 rozdzielczość pomiaru, 306

S

sekcja, 43, 189, 198, 199
 .bootloader, 146
 .bs, 141
 .bss, 143, 164, 197
 .data, 141, 142, 197
 .debug_, 141
 .eeprom, 141, 143
 .fini, 141, 145
 .fuse, 146
 .init, 141, 144, 194
 .jumpables, 144
 .lock, 146
 .noinit, 141, 143, 144, 154
 .progmem.gcc, 513
 .signature, 146
 .text, 141, 142, 144
 .trampolines, 144
 .vectors, 142
 adres, 142, 144, 146, 147
 danych, 141, 142

eeprom, 182
 krytyczna, 262
 specjalna, 146
 Serial Peripheral Interface,
 Patrz interfejs szeregowy
 signed char, *Patrz* typ danych
 znakowe
 Single Conversion Mode, *Patrz*
 tryb pojedynczej konwersji
 skrypt
 domyślny, 28
 linkera, 28
 Makefile, 27, 34, 37, 147
 Sleep Mode Control Register,
 Patrz rejestr SMCR
 słowo kluczowe, 116, 123
 __attribute_, 254
 asm, 529, 530
 const, 549
 extern, 130
 static, 104, 129
 volatile, *Patrz* modyfikator
 volatile
 stabilizator impulsowy, 317
 stabilizator liniowy, 317
 stała
 __malloc_margin, 165
 sterowanie kluczy mocy, 320
 sarta, 164, 165, 171, 172, 197,
 199, 202, 554
 fragmentacja, 166
 stos, 165, 172, 194, 196, 197,
 199, 208, 249, 490, 537,
 538, 554
 rejestr, 144
 suma kontrolna, 509
 sygnał
 +Vcc, 56
 aktywujący, *Patrz* sygnał
 wyzwalający
 analogowy, 160, 290
 asynchroniczny, 307
 CKOUT, 73
 cyfrowy, 321
 IO, 62, 72, 73, 74, 160
 MISO, 55, 59
 MOSI, 55, 59
 RESET, 54, 55, 59, 60, 63,
 64, 74, 149, 151, 152,
 153, 157
 SCK, 55, 58, 59, 63
 synchroniczny, 307
 taktujący, 306, 308
 TCK, 60

TDI, 60, 62
 TDO, 60, 62
 TMS, 60
 TPICLK, 64
 TPIDATA, 64
 wyzwalający, 321
 XTAL1, 64, 73
 sygnatura, 74
 symbol, 126
 __heap_end, 164
 __heap_start, 164
 __stack, 208
 symetryczny szyfr blokowy
 AES, 519
 symulator
 AVR Studio, 20
 programowy, 19
 synchronizator, 228
 szablon, 127
 szerokość impulsu, 316
 szum, 311
 szyfrowanie, 518, 519, 526

T

tablica, 109
 element, 110
 rozmiar, 109, 112
 skoków, 144
 typ, 109
 wektorów przerwań, 251,
 490, 492, 493, 494, 501,
 502, 513, 519, 540
 wielowymiarowa, 110
 technika częstego
 próbekowania, 230
 template, *Patrz* szablon
 termometr analogowy, 293
 termometr cyfrowy DS1820, 480
 timer, 219, 305, 308, 309, 312,
 321
 Timer/Counter Control
 Register, *Patrz* rejestr TCCR
 Timestamp, *Patrz* marker
 czasowy
 transceiver, 368
 transmisja
 asynchroniczna, 371
 synchroniczna, 368
 tryb
 asynchroniczny, 329, 330, 371
 ciągłej konwersji, 283, 287,
 288, 293
 CTC, 315

tryb
 emulacji SPI, 409, *Patrz* tryb MSPIM
 Fast PWM, 318
 FAST PWM, 309, 310
 full-duplex, 391
 głębokiego uśpienia, 328
 I2C, 417
 modulacji, 322
 MPCM, 386
 MSPIM, 408
 multimaster, 413, 416
 o wysokiej prędkości, 413
 pojedynczej konwersji, 283, 286, 287, 290
 power save, 328
 pracy timera, 312
 prosty, 312
 PWM, 310, 316, 317, 321
 PWM z korekcją fazy, 319
 PWM z korekcją fazy i częstotliwości, 319
 redukcji szumów, 290
 SLEEP_MODE_PWR_SAVE, 330
 synchroniczny, 371, 380
 szybki, 413
 wieloprocesorowy MPCM, 371
 Two Wire Serial Interface, *Patrz* interfejs TWI
 typ danych
 _Accum, 83, 84
 _Fract, 83
 _Sat, *Patrz* typ danych z saturacją
 binarne, 95
 bool, 78
 całkowite, 77, 96
 char, 537, 538
 double, 88, 90
 float, 88, 89, 90, 94
 int, 79, 80
 łańcuch znakowy, 86, 90, 92, 94, 103, 107
 porządkowy, 121
 prosty, 77
 stałopozycyjne, 83, 86, 96
 symbole, 85
 unsigned char, 538
 wielobajtowe, 533
 wskaźnik, 104, 107, 108, 109, 111
 wyliczeniowe, 80
 z saturacją, 84

złożony, 77
 zmiennopozycyjne, 77, 87, 93, 96, 118
 znakowe, 78
 typ rekordu, 45

U

układ
 analogowy, 316
 BOD, 71, 151, 157, 160, 161, 179
 cyfrowy, 316
 detekcji zboczy, 308
 ochronny, 321
 porównywania danych, 309
 redukcji zakłóceń, 321
 synchronizujący, 308
 układ nadzorujący zasilanie, *Patrz* układ BOD
 zegarowy, 326, 327, 328, 329
 Universal Serial Interface, *Patrz* interfejs USI
 Universal Synchronous and Asynchronous serial Receiver and Transmitter, *Patrz* interfejs USART
 unsigned char, *Patrz* typ danych znakowe
 urządzenie
 DFU, 67
 USART in Master SPI Mode, *Patrz* tryb MSPIM
 USB DFU Bootloader Datasheet, 508
 uśpienie, 290
 uśrednianie, 292

V

Vendor ID, *Patrz* VID
 VID, 456, 457, 463

W

Watchdog, 72, 144, 145, 152, 161
 reset, 156
 Watchdog Timer Control Register, *Patrz* rejestr WDTCR
 Wear Leveling, 182, 186, 497
 wektor
 obsługi przerwania, 252, 253
 przerwania, 142
 RESET, 496, 497, 499, 507

while, *Patrz* instrukcja pętla while
 WinAVR, 16
 wskaźnik, 105, 196, *Patrz* typ danych wskaźnik
 byteIO, 245
 dostęp, 534
 wordIO, 245
 współczynnik wypełnienia, 316, 317, 318, 319, 323
 wydajność prądowa, 57
 wyjście
 równoległe, 68
 szeregowo, 68
 wyświetlacz
 alfanumeryczny, 331, 332
 graficzny, 331, 354
 LCD, 127, 331
 LED, 405
 monochromatyczny, 99, 332
 wyświetlacz 7-segmentowy
 LED, 230, 268, 269, 270, 271, 272, 273, 275, 277, 279

Z

zabezpieczenie kodu, 517
 zakłócenia, 290
 zarządzanie energią, 156, 158, 159, 160, 290, 327, 328
 zatrząsk, 228
 zegar, 72, 73, 149, 152, 158, 159, 178, 199, 228, 286, 306, 311, 371, 394, 396, 456, 462
 RTC, 431, *Patrz* układ zegarowy
 SCL, 413
 zmienna, 136
 __malloc_heap_end, 164
 __malloc_heap_start, 164
 alokowana dynamicznie, 199, 201
 globalna, 100, 101, 131, 143, 196, 199, 259, 505, 554
 LDFLAGS, 29
 LIBDIRS, 29
 LIBS, 29
 lokalna, 100, 102, 196, 266, 505
 łańcuchowa, 103
 OBJECTS, 29
 statyczna, 102, 143, 199, 490
 współdzielona, 146

JĘZYK C

DLA MIKROKONTROLERÓW AVR

Mikrokontrolery AVR firmy Atmel stanowią dynamicznie rozwijającą się rodzinę układów. Dzięki niskiej cenie, dużym możliwościom i dostępności darmowych narzędzi od lat niezmiennie cieszą się dużą popularnością wśród hobbystów i osób profesjonalnie zajmujących się programowaniem mikrokontrolerów.

Pewnym utrudnieniem dla polskich użytkowników AVR jest brak literatury na temat wykorzystania do ich programowania języków wysokiego poziomu, takich jak C. Niniejsza książka jest próbą wypełnienia tej luki. W sposób syntetyczny pokazuje różnice pomiędzy programowaniem w języku C komputerów klasy PC i mikrokontrolerów. Omawia programowanie peryferii dostępnych w mikrokontrolerach AVR w języku C, bibliotekę standardową oraz jej rozszerzenia znane jako AVR-libc. Dzięki temu nawet osoby w niewielkim stopniu znające podstawy języka C będą mogły bez problemów „przebrać się” na programowanie mikrokontrolerów AVR. Z drugiej strony książka opisuje zaawansowane techniki programowania, związane z obsługą bootloadera, zabezpieczaniem i szyfrowaniem kodu aplikacji oraz realizacją najpowszechniej stosowanych protokołów wymiany danych pomiędzy urządzeniami opartymi na mikrokontrolerach i komputerami PC. Porusza także tematy związane ze specyfiką pisania aplikacji na mikrokontrolery oraz wyszukiwaniem i usuwaniem błędów.

- Podstawy programowania mikrokontrolerów AVR
- Warsztat pracy programisty AVR
- Wprowadzenie do języka C na AVR
- Budowa programu i jego części składowe
- Zastosowania przetwornika ADC
- Korzystanie z zasobów sprzętowych mikrokontrolera
- Używanie rejestrów i różnych rodzajów pamięci
- Obsługa wyświetlaczy LCD
- Korzystanie z interfejsów
- Zapewnianie bezpieczeństwa kodu

Programowanie mikrokontrolerów
jeszcze nigdy nie było tak proste!

W katalogu: 5500



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



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:
W <http://helion.pl> znajdziesz
Katalogi najlepszych ciekawych
W ofercie (druk, e-booki, audiobooki)
Zamów informacje o nowościach
W <http://helion.pl>

Helion Sp. z o.o.
ul. Rakocznego 10, 44-100 Gliwice
tel.: 32 230 96 63
e-mail: helion@helion.pl
<http://helion.pl>

helion.pl
e-booki
audiobooki

Cena: 89,00 zł

ISBN 978-83-246-3064-6



Informatyka w najlepszym wydaniu