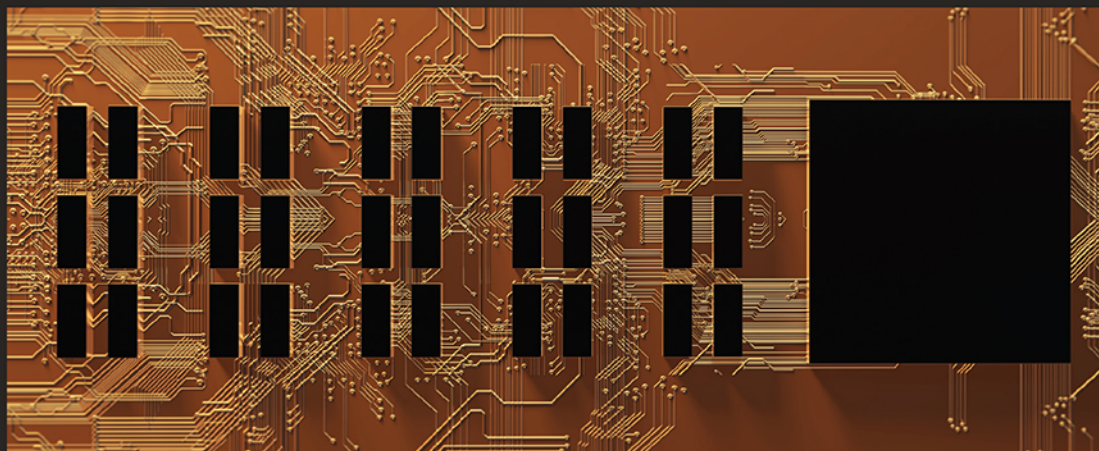




Wydajne systemy wbudowane w języku C

Niskopoziomowe programowanie
mikrokontrolerów ARM

ISRAEL GBATI

Tytuł oryginału: Bare-Metal Embedded C Programming: Develop high-performance embedded systems with C for Arm microcontrollers

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-3344-6

Copyright ©Packt Publishing 2025.

First published in the English language under the title 'Bare-Metal Embedded C Programming – (9781835460818)'

Polish edition copyright © 2026 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek 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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/wyslyc

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: *helion@helion.pl*

WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- **Lubię to!** » Nasza społeczność

Spis treści |

O autorze	14
O recenzencie	15
Przedmowa	16
Wstęp	17

ROZDZIAŁ 1

Przygotowywanie warsztatu pracy	23
Wymagania techniczne	23
Podstawowe narzędzia do tworzenia oprogramowania dla mikrokontrolerów	24
Instalacja STM32CubeIDE	25
Instalacja GNU Arm Embedded Toolchain	27
Instalacja OpenOCD	28
Płytki rozwojowa	32
Funkcje płytki rozwojowej	33
Przegląd płytki rozwojowej NUCLEO-F411	33
Karty katalogowe i instrukcje — odkrywanie szczegółów	34
Dokumentacja STMicroelectronics	34
Ogólny przewodnik użytkownika ARM	35
Pobieranie dokumentów	36
Praca w środowisku STM32CubeIDE	36
Ikony sterujące	40
Podsumowanie	40

ROZDZIAŁ 2

Konstruowanie rejestrów układów peryferyjnych na podstawie adresów pamięci	41
Wymagania techniczne	42
Różne metody tworzenia oprogramowania układowego	42
HAL	42
LL	43

Programowanie niskopoziomowe w C	44
Asembler	45
Komponenty na płycie rozwojowej	47
Lokalizowanie połączenia diody LED	47
Lokalizowanie połączenia przycisku użytkownika	49
Lokalizowanie goldpinów i złączy kompatybilnych z Arduino	49
Definiowanie i tworzenie rejestrów na podstawie dokumentacji	52
Lokalizowanie portu GPIO PORTA	52
Bramkowanie zegara	56
AHB1 ENR	58
Ustawianie i zerowanie bitów w rejestrach	59
Rejestr trybu portu GPIO (GPIOx_MODER)	61
Rejestr danych wyjściowych portu GPIO (GPIOx_ODR)	64
Manipulowanie rejestrami — od konfiguracji do uruchomienia pierwszego programu	65
Definicje rejestrów	66
Przyrostek UL	67
Funkcja main	69
Podsumowanie	72

ROZDZIAŁ 3

Proces budowania programów i eksplorowanie narzędzi GNU 74

Wymagania techniczne	74
Podstawy — proces budowania oprogramowania wbudowanego	75
Etap przetwarzania wstępnego	75
Etap kompilacji	76
Etap asemblacji	76
Etap konsolidacji	76
Etap lokalizacji	77
Przegląd narzędzi GNU dla systemów wbudowanych	77
arm-none-eabi-gcc	78
Często używane flagi kompilatora	78
Niektóre flagi specyficzne dla architektury	79
Inne polecenia w zestawie narzędzi GNU Arm Embedded Toolchain	81
Od IDE do wiersza poleceń — analiza procesu budowania programu	83
Proces budowania oprogramowania z perspektywy IDE	83
Kompilacja plików asemblera i C	84
Praca z narzędziami binarnymi GNU	85
Wgrywanie oprogramowania układowego do mikrokontrolera za pomocą OpenOCD	89
Podsumowanie	92

ROZDZIAŁ 4**Tworzenie skryptu linkera i pliku startowego 93**

Wymagania techniczne	93
Model pamięci STM32	93
Pamięć flash	94
SRAM	95
Pamięć układów peryferyjnych	96
Skrypt linkera	96
Proces konsolidacji	97
Kluczowe elementy skryptu linkera	98
Dyrektywy w skrypcie linkera	100
Stałe w skryptach linkera	108
Symbole w skryptach linkera	109
Pisanie skryptu linkera i pliku startowego	111
Regiony pamięci, do których wczytywane są poszczególne sekcje	111
Przerwania i tablica wektorów	112
Pisanie skryptu linkera	113
Pisanie pliku startowego	117
Testowanie skryptu linkera i pliku startowego	124
Podsumowanie	125

ROZDZIAŁ 5**System Make 127**

Wymagania techniczne	127
Wprowadzenie do systemów budowania oprogramowania	127
Make	129
Maven	129
System budowania Make	130
Podstawy systemu Make	130
Instalowanie i konfigurowanie systemu Make	132
Tworzenie plików Makefile dla projektów oprogramowania układowego	134
Testowanie pliku Makefile	136
Stosowanie zmiennych specjalnych i zdefiniowanych przez użytkownika	138
Podsumowanie	138

ROZDZIAŁ 6**Common Microcontroller Software Interface Standard (CMSIS) 140**

Wymagania techniczne	140
Definiowanie rejestrów układów peryferyjnych za pomocą struktur C	141
Uzyskiwanie adresu bazowego i przesunięć rejestrów	141
Implementowanie struktur układów peryferyjnych	143
Dostęp do rejestrów oparty na strukturach	145
Omówienie CMSIS	146
Czym jest CMSIS?	147
Kluczowe elementy CMSIS	147
Zasady kodowania CMSIS	148
Pliki CMSIS-Core	148
Przygotowywanie wymaganych plików CMSIS	150
Dołączanie odpowiednich plików nagłówkowych	150
Praca z plikami CMSIS	151
Podsumowanie	154

ROZDZIAŁ 7**Wejście-wyjście ogólnego przeznaczenia (GPIO) 156**

Wymagania techniczne	156
Układ peryferyjny GPIO	157
Rejestry GPIO w mikrokontrolerach STM32	158
Rejestr trybu GPIO (GPIOx_MODER)	158
Rejestr danych wyjściowych GPIO (GPIOx_ODR) oraz rejestr danych wejściowych GPIO (GPIOx_IDR)	159
Rejestr ustawiania/zerowania bitów GPIO (GPIOx_BSRR)	160
Rejestry alternatywnych funkcji GPIO (GPIOx_AFRL i GPIOx_AFRH)	161
Tworzenie sterowników wejścia i wyjścia	164
Sterownik wyjścia GPIO wykorzystujący rejestr BSRR	164
Sterownik wejścia GPIO	167
Podsumowanie	169

ROZDZIAŁ 8**Timer systemowy (SysTick) 171**

Wymagania techniczne	171
Wprowadzenie do timera SysTick	171
Przegląd timera SysTick	171
Rejestry timera SysTick	172
Tworzenie sterownika timera SysTick	175
Podsumowanie	178

ROZDZIAŁ 9

Timery ogólnego przeznaczenia (TIM)	179
Wymagania techniczne	179
Wprowadzenie do timerów i ich zastosowań	179
Typowe zastosowania timerów	180
Pomiar interwałów czasowych	180
Generowanie opóźnień	181
Wyzwalanie zdarzeń	181
Timery STM32	181
Wprowadzenie do timerów ogólnego przeznaczenia i timerów zaawansowanych	182
Jak działają timery w mikrokontrolerach STM32?	183
Tworzenie sterownika timera	186
Podsumowanie	188

ROZDZIAŁ 10

Protokół uniwersalnego asynchronicznego odbiornika-nadajnika (UART)	189
Wymagania techniczne	189
Wprowadzenie do protokołów komunikacyjnych	190
Czym są protokoły komunikacyjne?	190
Porównanie interfejsów UART, SPI i I2C	192
Typowe zastosowania protokołów UART, SPI i I2C	194
Przegląd protokołu UART	196
Czym jest UART?	197
Interfejs	197
Jak działa UART?	197
Układ peryferyjny UART w mikrokontrolerze STM32F4	200
Tworzenie sterownika UART	202
Podsumowanie	208

ROZDZIAŁ 11

Przetwornik analogowo-cyfrowy (ADC)	210
Wymagania techniczne	210
Podstawy konwersji analogowo-cyfrowej	211
Czym jest konwersja analogowo-cyfrowa?	211
Kluczowe parametry ADC — rozdzielczość, wielkość kroku kwantyzacji oraz napięcie referencyjne (VREF)	213

Układ peryferyjny ADC w mikrokontrolerze STM32F4	215
Kanały ADC	215
Kanały zwykłe i wstrzykiwane w przetwornikach ADC mikrokontrolerów STM32F411	217
Kluczowe rejestry i flagi ADC	218
1. rejestr sterujący ADC (ADC_CR1)	218
2. rejestr sterujący ADC (ADC_CR2)	219
Rejestr sekwencji zwykłej ADC (ADC_SQRx)	219
Rejestr danych ADC (ADC_DR)	219
Rejestr statusu ADC (ADC_SR)	220
Kluczowe flagi ADC	220
Tworzenie sterownika ADC	221
Identyfikowanie pinów GPIO na użytek ADC	221
Podsumowanie	228

ROZDZIAŁ 12

Serial Peripheral Interface (SPI)

Wymagania techniczne	229
Przegląd protokołu SPI	229
Czym jest SPI?	230
Kluczowe cechy protokołu SPI	230
Interfejs SPI	230
Jak działa SPI?	232
CPHA i CPOL	233
Tryby danych	233
Prędkość SPI	234
Układ peryferyjny SPI w mikrokontrolerze STM32F4	234
Kluczowe cechy	234
Kluczowe rejestry SPI	235
Tworzenie sterownika SPI	236
Zdefiniowane makra	239
Inicjalizacja pinów GPIO na użytek SPI	240
Konfiguracja SPI1	241
Wysyłanie danych przez SPI	242
Odbieranie danych przez SPI	242
Zarządzanie linią SS	242
Plik nagłówkowy	243
Akcelerometr ADXL345	243
Kluczowe pojęcia — statyczne przyspieszenie grawitacyjne, wykrywanie nachylenia oraz przyspieszenie dynamiczne	246
Tworzenie sterownika akcelerometru ADXL345	248
Podsumowanie	252

ROZDZIAŁ 13

Inter-Integrated Circuit (I2C)	254
Wymagania techniczne	254
Przegląd protokołu I2C	254
Czym jest I2C?	254
Układ peryferyjny I2C w mikrokontrolerach STM32F4	260
Kluczowe rejestry I2C	260
Tworzenie sterownika I2C	262
Podsumowanie	274

ROZDZIAŁ 14

Przerwania i zdarzenia zewnętrzne (EXTI)	275
Wymagania techniczne	275
Przerwania i ich rola w oprogramowaniu układowym	276
Czym są przerwania?	276
Jak działają przerwania?	276
Znaczenie przerw w oprogramowaniu układowym	277
Przerwania a wyjątki	277
Analiza porównawcza: rozwiązania oparte na przerwaniach a rozwiązania oparte na odpytywaniu	280
Kontroler EXTI w STM32	283
Kluczowe cechy EXTI	283
Odwzorowywanie zewnętrznych linii przerw/zdarzeń	284
Tworzenie sterownika EXTI	285
EXTI_IMR	285
EXTI_RTISR	285
EXTI_FTSR	285
Rejestr przerw oczekujących (EXTI_PR)	285
Sterownik EXTI	286
Podsumowanie	289

ROZDZIAŁ 15

Zegar czasu rzeczywistego (RTC)	290
Wymagania techniczne	290
Zegary RTC	290
Jak działają zegary RTC?	291
Typowe zastosowania zegarów RTC	291
Moduł RTC w mikrokontrolerach STM32	293
Główne cechy modułu RTC w STM32F4	293
Kluczowe elementy modułu RTC w STM32F4	294

Kluczowe rejestry RTC	299
Rejestr czasu RTC (RTC_TR)	299
Rejestr daty RTC (RTC_DR)	299
Rejestr sterujący RTC (RTC_CR)	300
Rejestr inicjalizacji i statusu RTC (RTC_ISR)	300
Rejestr preskalera RTC (RTC_PRER)	300
Rejestry alarmów RTC (RTC_ALRMAR i RTC_ALRMBR)	301
Rejestr timera wybudzania RTC (RTC_WUTR)	301
Tworzenie sterownika RTC	301
Plik implementacji RTC	302
Format BCD	310
Plik nagłówkowy	312
Główny plik	313
Podsumowanie	315

ROZDZIAŁ 16

Niezależny watchdog (IWDG)	317
Wymagania techniczne	317
Watchdogi	318
Czym są watchdogi?	318
Jak działa watchdog?	318
Typowe zastosowania	319
Typy watchdogów	320
IWDG w mikrokontrolerach STM32	322
Kluczowe cechy IWDG	323
Jak działa IWDG?	323
Rejestry IWDG	324
Tworzenie sterownika IWDG	325
Plik implementacji IWDG	325
Plik nagłówkowy	327
Główny plik	328
Testowanie projektu	330
Podsumowanie	330

ROZDZIAŁ 17

Bezpośredni dostęp do pamięci (DMA)	331
Wymagania techniczne	331
Bezpośredni dostęp do pamięci (DMA)	332
Jak działa DMA?	332
Kluczowe cechy	332
Typowe zastosowania	333

Moduły DMA mikrokontrolera STM32F4	335
Kluczowe cechy kontrolera DMA w mikrokontrolerze STM32F4	335
Tryby transferu	336
Tryby danych DMA	337
Schemat blokowy modułu DMA w mikrokontrolerze STM32F4	337
Kluczowe rejestry DMA w mikrokontrolerach STM32	339
Tworzenie sterownika DMA	341
Sterownik DMA dla przetwornika ADC	341
Sterownik DMA dla układu UART	344
Sterownik DMA do transferu danych między obszarami pamięci	352
Podsumowanie	356

ROZDZIAŁ 18

Zarządzanie energią i efektywność energetyczna

w systemach wbudowanych	357
Wymagania techniczne	358
Przegląd technik zarządzania energią	358
Dynamiczne skalowanie napięcia i częstotliwości (DVFS)	358
Bramkowanie zegara	359
Bramkowanie zasilania	359
Tryby niskiego poboru mocy	360
Studium przypadku 1. Energooszczędny smartwatch	360
Studium przypadku 2. Monitor środowiskowy zasilany energią słoneczną	361
Tryby niskiego poboru mocy w mikrokontrolerze STM32F4	362
Źródła i wyzwalacze wybudzania w STM32F4	364
Źródła wybudzania	364
Kwestie praktyczne	366
Tworzenie sterownika do wchodzenia w stan gotowości i wybudzania mikrokontrolera	367
Podsumowanie	371

Skorowidz	373
------------------------	------------

Protokół uniwersalnego asynchronicznego odbiornika-nadajnika (UART)

Rozdział

10

W tym rozdziale poznasz protokół **uniwersalnego asynchronicznego odbiornika-nadajnika** (ang. *Universal Asynchronous Receiver/Transmitter*, UART) — ważną metodę komunikacji szeroko stosowaną w systemach wbudowanych. UART umożliwia komunikację między mikrokontrolerami a różnymi urządzeniami peryferyjnymi, co czyni go niezbędnym elementem w rozwoju systemów wbudowanych.

Zacniemy od omówienia znaczenia protokołów komunikacyjnych w systemach wbudowanych i przedstawimy typowe przypadki użycia UART obok innych protokołów, takich jak SPI i I2C. Następnie zajmiemy się kompleksowym przeglądem protokołu UART, szczegółowo opisując jego zasadę działania i funkcje. Przeanalizujemy rejestry UART opisane w podręczniku referencyjnym STM32, aby zdobyć wiedzę niezbędną do tworzenia sterowników. Na koniec wykorzystamy tę wiedzę do napisania niskopoziomowego sterownika UART, aby zilustrować praktyczne aspekty inicjalizacji i przesyłania danych za pomocą tego interfejsu.

W tym rozdziale omówimy następujące główne tematy:

- wprowadzenie do protokołów komunikacyjnych,
- przegląd protokołu UART,
- układy peryferyjne UART w mikrokontrolerach STM32F4,
- tworzenie sterownika UART.

Po zakończeniu tego rozdziału będziesz dobrze rozumieć protokół UART i zdobędziesz umiejętności potrzebne do tworzenia niskopoziomowych sterowników do komunikacji UART.

Wymagania techniczne

Wszystkie przykłady kodu z tego rozdziału są dostępne w repozytorium na GitHubie pod adresem <https://github.com/PacktPublishing/Bare-Metal-Embedded-C-Programming>. Spolszczone przykłady można pobrać pod adresem <https://ftp.helion.pl/przyklady/wysyjc.zip>.

Wprowadzenie do protokołów komunikacyjnych

W świecie systemów wbudowanych protokoły komunikacyjne odgrywają ważną rolę, umożliwiając wymianę informacji między mikrokontrolerami a urządzeniami peryferyjnymi. Można je porównać do języków, którymi posługują się różne urządzenia — od smartfonów po inteligentne urządzenia domowe — aby rozumieć się nawzajem i wymieniać dane.

Zobaczmy zatem, czym są protokoły komunikacyjne, jak je klasyfikujemy, jakie mają cechy charakterystyczne i zalety, a następnie przeanalizujemy kilka typowych zastosowań, aby zobaczyć te protokoły w akcji.

Czym są protokoły komunikacyjne?

Protokoły komunikacyjne to zbiory reguł i konwencji umożliwiających urządzeniom elektronicznym wzajemną komunikację. Określają one sposób formatowania, przesyłania i odbierania danych, zapewniając dokładną i niezawodną wymianę informacji między urządzeniami. Bez tych protokołów próba komunikacji przypominałaby rozmowę z kimś mówiącym zupełnie innym językiem — byłaby chaotyczna i podatna na błędy.

W systemach wbudowanych protokoły te mają kluczowe znaczenie, ponieważ umożliwiają interakcję między mikrokontrolerami a urządzeniami peryferyjnymi, takimi jak czujniki, siłowniki, wyświetlacze czy inne mikrokontrolery. Niezależnie od tego, czy chodzi o przesłanie prostego odczytu temperatury z czujnika do mikrokontrolera, czy transmisję danych wideo z modułu kamery, to właśnie protokoły komunikacyjne sprawiają, że jest to możliwe.

Przyjrzyjmy się klasyfikacji protokołów komunikacyjnych, zaczynając od ogólnego podziału na komunikację **szeregową** i **równoległą**.

Komunikacja szeregową i równoległą

Zacznijmy od komunikacji szeregowej.

Komunikacja szeregową

Protokoły komunikacyjne tej kategorii można podzielić na asynchroniczne i synchroniczne:

- **Asynchroniczne.** Protokoły tego typu przesyłają dane bit po bicie bez sygnału zegarowego do synchronizacji nadajnika i odbiornika. Można to porównać do wysyłania listów pocztą bez ustalonego czasu doręczenia. Popularnym przykładem jest UART, który jest prosty i nadaje się do wielu zastosowań.
- **Synchroniczne.** W przeciwieństwie do komunikacji asynchronicznej ta forma komunikacji wykorzystuje sygnał zegarowy do koordynacji przesyłania bitów. To jak marsz w rytm bębna, który zapewnia, że wszyscy równo stawiają kroki. Przykładami są **Serial Peripheral Interface (SPI)** i **Inter-Integrated Circuit (I2C)**. Te protokoły zapewniają integralność danych i precyzyjne taktowanie, co czyni je odpowiednimi do bardziej złożonych zadań.

Komunikacja równoległa

Ten typ komunikacji polega na przesyłaniu wielu bitów jednocześnie przez wiele kanałów. Wyobraź sobie wysłanie całej floty samochodów zamiast jednego pojazdu — jest to szybsze, ale wymaga więcej pasów ruchu (w naszym przypadku pinów). Chociaż komunikacja równoległa jest szybsza, rzadziej stosuje się ją w systemach wbudowanych ze względu na większą liczbę potrzebnych pinów. Ponadto jest bardziej podatna na zakłócenia i problemy z integralnością sygnału, szczególnie na dłuższych dystansach.

Protokoły komunikacyjne możemy również klasyfikować ze względu na ich architekturę. W tym systemie klasyfikacji wyróżniamy komunikację punkt-punkt oraz komunikację wielourzędzeniową.

Komunikacja punkt-punkt a komunikacja wielourzędzeniowa

Przyjrzyjmy się różnicom.

Komunikacja punkt-punkt

Jest to bezpośrednia linia komunikacyjna między dwoma urządzeniami. Klasycznym przykładem jest interfejs **UART**, w którym dane przepływają bezpośrednio między mikrokontrolerem a urządzeniem peryferyjnym. Jest to proste, niezawodne i odpowiednie dla wielu systemów wbudowanych.

Komunikacja wielourzędzeniowa (magistrala)

W tym przypadku wiele urządzeń korzysta z tych samych linii komunikacyjnych. Może to być realizowane na dwa sposoby:

- **Wiele urządzeń nadrzędnych.** Wiele urządzeń może kontrolować magistralę komunikacyjną. Świetnym przykładem jest interfejs **I2C**, który pozwala na obecność wielu urządzeń nadrzędnych i podrzędnych na tej samej magistrali. Przypomina to grupę przyjaciół, którzy na zmianę zabierają głos w rozmowie.
- **Nadrzędny-podrzędny.** Komunikacją steruje jedno urządzenie nadrzędne, które kontroluje wymianę danych z wieloma urządzeniami podrzędnymi. W ten sposób działa **SPI** — jedno urządzenie nadrzędne komunikuje się z wieloma urządzeniami podrzędnymi za pomocą osobnych linii. **I2C** również może działać w tym trybie. Przypomina to nauczyciela (urządzenie nadrzędne) wywołującego po kolei uczniów do odpowiedzi.

Wreszcie protokoły komunikacyjne można klasyfikować ze względu na ich możliwości w zakresie przepływu danych.

Komunikacja pełnoduplexowa i półduplexowa

Przyjrzyjmy się różnicom między komunikacją pełnoduplexową i półduplexową:

- **Pełny duplex.** Umożliwia jednoczesną komunikację w obu kierunkach. Wyobraź sobie dwupasmową drogę, po której samochody mogą poruszać się w obu kierunkach w tym samym czasie. Interfejsy **UART** i **SPI** obsługują komunikację w trybie pełnoduplexowym, co umożliwia bardzo wydajną wymianę danych w czasie rzeczywistym.

- **Półdupleks.** W tym przypadku komunikacja może odbywać się w obu kierunkach, ale niejednocześnie — to jak jednokierunkowa droga, po której samochody jeżdżą na zmianę. Protokół **I2C** zazwyczaj działa w trybie półdupleksowym, co sprawdza się dobrze w jego zamierzonych zastosowaniach, ale może stanowić ograniczenie w scenariuszach wymagających szybkiej transmisji danych.

Porównajmy teraz trzy popularne protokoły komunikacyjne używane we współczesnych systemach wbudowanych.

Porównanie interfejsów UART, SPI i I2C

Zacznijmy od protokołu UART.

UART

Oto kluczowe cechy interfejsu UART:

- **Komunikacja asynchroniczna.** UART nie wymaga sygnału zegarowego. Zamiast tego wykorzystuje bity startu i stopu do synchronizacji transmisji danych.
- **Pełny duplex.** UART może jednocześnie wysyłać i odbierać dane, co przydaje się w wielu zastosowaniach wymagających komunikacji w czasie rzeczywistym.
- **Prostota i niski koszt.** Dzięki minimalnym wymaganiom sprzętowym UART jest łatwy w implementacji i ekonomiczny.

Oto niektóre z zalet interfejsu UART:

- **Łatwość użycia.** Konfiguracja komunikacji UART jest prosta, przez co jest on popularny wśród początkujących i w prostych aplikacjach.
- **Szerokie wsparcie.** UART jest powszechnie obsługiwany przez większość mikrokontrolerów i urządzeń peryferyjnych.
- **Niskie koszty dodatkowe.** Brak sygnału zegarowego oznacza, że używanych jest mniej pinów, co zmniejsza złożoność układu.

Ma on jednak również pewne wady:

- **Ograniczona szybkość.** UART jest zazwyczaj wolniejszy w porównaniu z SPI i I2C, więc nie nadaje się do szybkiej transmisji danych.
- **Ograniczony zasięg.** Podatność na zakłócenia przy dłuższych odległościach może ograniczać zasięg niezawodnej komunikacji.
- **Tylko połączenia punkt-punkt.** UART jest przeznaczony do bezpośredniej komunikacji między dwoma urządzeniami, co może być problemem, gdy potrzebna jest komunikacja między wieloma urządzeniami.

Przejdźmy teraz do omówienia interfejsu SPI.

SPI

Oto kluczowe cechy interfejsu SPI:

- **Komunikacja synchroniczna.** SPI wykorzystuje sygnał zegara wraz z liniami danych, co umożliwia synchroniczne przesyłanie informacji.
- **Pełny duplex.** Protokół umożliwia jednocześnie wysyłanie i odbieranie danych.
- **Architektura nadrzędny-podrzędny.** Jedno urządzenie nadrzędne (ang. *master*) kontroluje wiele urządzeń podrzędnych (ang. *slave*), przy czym każde z nich ma osobną linię.

Oto niektóre z jego zalet:

- **Duża szybkość.** SPI umożliwia szybki transfer danych, więc nadaje się idealnie do zastosowań wymagających szybkiej komunikacji.
- **Wszechstronność.** SPI pozwala na podłączenie wielu urządzeń o różnych konfiguracjach, zapewniając elastyczność w projektowaniu.

Jednak ma on również pewne wady:

- **Większa liczba pinów.** Każde urządzenie podrzędne wymaga osobnej linii wyboru, co może znacznie zwiększyć liczbę potrzebnych pinów.
- **Brak standardowego potwierdzenia.** W przeciwieństwie do I2C — SPI nie ma wbudowanego mechanizmu potwierdzania, co może utrudniać wykrywanie błędów.
- **Ograniczona możliwość pracy w trybie z wieloma urządzeniami nadrzędnymi.** SPI nie jest zaprojektowany do systemów z wieloma urządzeniami nadrzędnymi, co może być ograniczeniem w niektórych scenariuszach.

Ostatnim popularnym protokołem komunikacyjnym, który tu omówimy, jest I2C.

I2C

Oto kluczowe cechy interfejsu I2C:

- **Komunikacja synchroniczna.** I2C wykorzystuje sygnał zegarowy do synchronicznej transmisji danych.
- **Obsługa wielu urządzeń nadrzędnych.** Kilka urządzeń master może współdzielić tę samą magistralę, co jest przydatne w bardziej złożonych systemach.
- **Interfejs dwuprzewodowy.** I2C wymaga tylko dwóch linii (SDA i SCL), minimalizując liczbę potrzebnych pinów.

Oto niektóre z jego zalet:

- **Proste połączenia.** Dwuprzewodowy interfejs zmniejsza złożoność i liczbę wymaganych pinów.
- **Obsługa wielu urządzeń.** I2C łatwo łączy wiele urządzeń na tej samej magistrali, każde z unikatowym adresem.

- **Wbudowane adresowanie.** I2C ma wbudowany mechanizm adresowania, co ułatwia komunikację z wieloma urządzeniami.

Ma jednak również pewne wady:

- **Niższa szybkość.** I2C jest zazwyczaj wolniejszy niż SPI, co może być ograniczeniem w zastosowaniach wymagających dużej szybkości.
- **Złożony protokół.** Protokół jest bardziej skomplikowany niż UART i SPI, więc wymaga bardziej zaawansowanej obsługi przesyłania danych i adresowania.
- **Podatność na zakłócenia.** Podobnie jak UART, przy większych odległościach I2C może być podatny na zakłócenia, co potencjalnie wpływa na niezawodność komunikacji.

Wybór odpowiedniego protokołu komunikacyjnego zależy od konkretnych potrzeb Twojej aplikacji. Jeśli potrzebujesz prostej, nieskomplikowanej komunikacji i nie przeszkadza Ci mniejsza prędkość, UART jest świetnym wyborem. Do zastosowań wymagających dużej szybkości i komunikacji pełnodupleksowej idealnie nadaje się SPI, zwłaszcza jeśli dysponujesz większą liczbą wolnych pinów. Gdy potrzebujesz połączyć wiele urządzeń z użyciem jak najmniejszej liczby przewodów i masz złożoną konfigurację komunikacyjną, najlepszym rozwiązaniem będzie I2C. Aby lepiej zrozumieć, kiedy wybrać który protokół, przyjrzyjmy się kilku typowym zastosowaniom.

Typowe zastosowania protokołów UART, SPI i I2C

Podczas projektowania systemów wbudowanych wybór odpowiedniego protokołu komunikacyjnego ma kluczowe znaczenie dla zapewnienia wydajnej i niezawodnej wymiany danych. UART, SPI i I2C mają swoje unikatowe zalety, co sprawia, że nadają się do różnych zastosowań. Przyjrzyjmy się kilku praktycznym zastosowaniom i studiom przypadków użycia każdego z tych protokołów.

UART

Oto kilka typowych zastosowań protokołu UART:

- **Komunikacja szeregową z komputerami.** Interfejsu UART często używa się do komunikacji szeregową między mikrokontrolerem a komputerem, szczególnie w celu debugowania, aktualizacji oprogramowania i rejestrowania danych.
- **Moduły GPS.** Interfejsu UART można użyć do przesyłania danych o lokalizacji z modułu GPS do mikrokontrolera.
- **Moduły Bluetooth.** UART umożliwia bezprzewodową komunikację z urządzeniami przez Bluetooth.

Przykłady te reprezentują najpowszechniejsze zastosowania UART, ale protokół ten jest wszechstronny i może być używany w wielu innych scenariuszach wymagających prostej komunikacji szeregową.

Studium przypadku — integracja modułu GPS w autonomicznym dronie

Wyobraź sobie, że projektujesz autonomicznego drona, który wymaga precyzyjnej nawigacji do wykonywania zadań, takich jak pomiary i mapowanie terenu. Integracja modułu GPS z użyciem UART może dostarczyć danych o lokalizacji niezbędnych do nawigacji.

Konfiguracja. Podłącz pin nadawczy (TX) modułu GPS do pinu odbiorczego (RX) mikrokontrolera i odwrotnie. Ustaw prędkość transmisji tak, aby odpowiadała wyjściu modułu GPS.

Działanie. Moduł GPS nieustannie wysyła zdania NMEA (ciągi tekstowe) zawierające dane o położeniu. Mikrokontroler odczytuje te ciągi przez interfejs UART, analizuje je i wykorzystuje zawarte w nich informacje do precyzyjnego sterowania dronem.

Zalety. Prostota interfejsu UART i jego uniwersalna kompatybilność sprawiają, że integracja modułu GPS jest stosunkowo łatwa. Zapewnia to niezawodny i ciągły przepływ danych bez konieczności skomplikowanej konfiguracji.

Spójrzmy teraz na interfejs SPI.

SPI

Oto kilka typowych zastosowań protokołu SPI:

- **Szybki transfer danych.** Idealnie nadaje się do zastosowań, takich jak obsługa kart pamięci, **przetworników analogowo-cyfrowych (ADC)**, **przetworników cyfrowo-analogowych (DAC)** i wyświetlaczy.
- **Moduły wyświetlaczy.** Interfejsu SPI można używać do komunikacji z wyświetlaczami o wysokiej rozdzielczości wymagającymi szybkiego odświeżania.
- **Czujniki i siłowniki.** SPI radzi sobie z odczytem danych o wysokiej częstotliwości z różnych czujników.

Podobnie jak w przypadku UART, przykłady te pokazują typowe zastosowania SPI, ale wysoka szybkość transmisji sprawia, że protokół ten nadaje się do szerokiej gamy innych zastosowań wymagających szybkiego przesyłania danych.

Studium przypadku — rejestrowanie danych na karcie SD w urządzeniach przemysłowych

Rozważmy przemysłowy system monitorowania, który zapisuje dane z różnych czujników na karcie SD do celów długoterminowej analizy. SPI jest idealnym protokołem do szybkiego przesyłania takich danych.

Konfiguracja. Podłącz mikrokontroler do karty SD za pomocą pinów SPI (MISO, MOSI, SCLK i CS). Zainicjuj magistralę SPI i skonfiguruj kartę SD.

Działanie. Mikrokontroler zbiera dane z czujników (np. temperatury, ciśnienia i wibracji) i zapisuje je na karcie SD w czasie rzeczywistym.

Zalety. Szybka transmisja danych przez SPI zapewnia, że duże ilości informacji są rejestrowane szybko i efektywnie, co zapobiega utracie danych i gwarantuje dokładne monitorowanie.

Wykorzystanie SPI w tym scenariuszu pozwala systemowi przemysłowemu na precyzyjne rejestrowanie kluczowych parametrów, co ma kluczowe znaczenie dla konserwacji prewencyjnej i efektywności operacyjnej.

Na koniec omówimy I2C.

I2C

Rozważmy dwa typowe zastosowania magistrali I2C:

- **Integracja wielu czujników.** Polega na podłączeniu kilku czujników o różnych adresach do tej samej magistrali I2C.
- **Rozszerzanie układów peryferyjnych.** Polega na dodawaniu dodatkowych pinów GPIO do mikrokontrolera za pomocą ekspanderów I2C.

To tylko dwa przykłady zastosowań I2C. Zdolność tego interfejsu do obsługi wielu urządzeń na jednej magistrali sprawia, że jest on doskonałym wyborem w wielu innych scenariuszach, w których ważna jest skalowalność.

Studium przypadku — system monitorowania środowiskowego w inteligentnym rolnictwie

Załóżmy, że opracowujesz inteligentny system rolniczy wykorzystujący wiele czujników (temperatury, wilgotności powietrza i gleby) do optymalizacji warunków uprawy. I2C idealnie nadaje się do takiej integracji wielu czujników.

Konfiguracja. Podłącz wszystkie czujniki do magistrali I2C (linie SDA i SCL). Przypisz każdemu czujnikowi unikatowy adres.

Działanie. Mikrokontroler odpytuje kolejno każdy czujnik, zbiera dane i przetwarza je, aby dostarczyć informacje i sterować systemami nawadniania, wentylacji i oświetlenia.

Zalety. Możliwość obsługi wielu urządzeń na tej samej magistrali przy użyciu tylko dwóch linii upraszcza okablowanie, obniża koszty i oszczędza piny GPIO, co czyni I2C efektywnym rozwiązaniem dla złożonych sieci czujników.

W kolejnym podrozdziale skupimy się wyłącznie na protokole UART. Protokoły I2C i SPI omówimy w następnych rozdziałach.

Przegląd protokołu UART

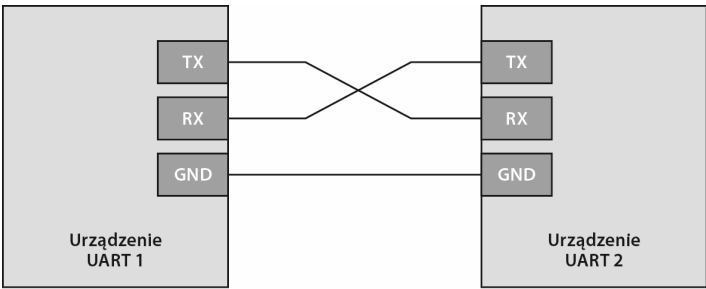
Jednym z najbardziej podstawowych i powszechnie stosowanych protokołów jest UART. Niezależnie od tego, czy debugujesz sprzęt, czy chcesz zapewnić komunikację między mikrokontrolerem a urządzeniami peryferyjnymi, zrozumienie działania UART ma kluczowe znaczenie. Zbadajmy dokładniej, jak działa ten protokół.

Czym jest UART?

UART to sprzętowy protokół komunikacyjny, który wykorzystuje asynchroniczną transmisję szeregową, umożliwiając regulację prędkości przesyłania danych. „Asynchroniczność” UART oznacza, że nie wymaga on sygnału zegarowego do synchronizacji przesyłania bitów między nadajnikiem a odbiornikiem. Zamiast tego oba urządzenia muszą uzgodnić konkretną prędkość w bodach (ang. *baud rate*), nazywaną też prędkością modulacji, która określa szybkość wymiany danych. Przyjrzyjmy się teraz bliżej temu interfejsowi.

Interfejs

Interfejs UART wykorzystuje do komunikacji dwa przewody: TX i RX. Aby nawiązać połączenie między dwoma urządzeniami, wystarczy podłączyć pin TX pierwszego urządzenia do pinu RX drugiego urządzenia, a pin RX pierwszego urządzenia do pinu TX drugiego urządzenia. Dodatkowo trzeba połączyć piny masy obu urządzeń, aby zapewnić wspólny punkt odniesienia elektrycznego. Połączenie między dwoma urządzeniami UART pokazano na rysunku 10.1.



Rysunek 10.1. Interfejs UART

Jak działa UART?

W transmisji UART dane są przesyłane w ramach zawierających **bit startu**, **bity danych**, opcjonalny **bit parzystości** oraz **bity stopu** (rysunek 10.2).

Bit startu	Ramka danych	Bity parzystości	Bity stopu
1 bit	5 do 9 bitów	1 bit	1 lub 2 bity

Rysunek 10.2. Pakiet danych UART

Oto szczegółowy opis tego procesu:

1. **Bit startu.** W stanie spoczynku linia transmisyjna jest utrzymywana w stanie wysokim. Aby rozpocząć przesyłanie danych, nadajnik UART obniża stan linii na jeden cykl zegara. Sygnalizuje to początek nowej ramki danych.

2. **Ramka danych.** Po bicie startu następuje ramka danych, która zwykle składa się z 5 do 9 bitów i jest wysyłana od **najmniej znaczącego bitu** (ang. *least significant bit*, LSB) do **najbardziej znaczącego bitu** (ang. *most significant bit*, MSB).
3. **Bit parzystości.** Jest to opcjonalny bit używany do wykrywania błędów. Wskazuje on, czy liczba ustawionych bitów (jedynek) w danych jest parzysta czy nieparzysta.
4. **Bity stopu.** Jeden lub dwa bity oznaczające koniec pakietu danych. Podczas transmisji bitów stopu linia jest utrzymywana w stanie wysokim.

Przyjrzyjmy się bliżej bitom startu, stopu i parzystości.

Bity startu, stopu i parzystości

Te bity stanowią podstawę protokołu UART, umożliwiając urządzeniom synchronizację i weryfikację integralności przesyłanych danych.

Bit startu

Bit startu to początkowy sygnał oznaczający rozpoczęcie ramki danych w komunikacji UART. Gdy urządzenie nadające jest bezczynne, linia danych utrzymywana jest na wysokim poziomie napięcia (stan logiczny 1). Aby zasygnalizować początek transmisji, nadajnik UART obniża napięcie na linii do poziomu niskiego (stan logiczny 0) na czas trwania 1 bitu. To przejście ze stanu wysokiego do niskiego informuje urządzenie odbierające o nowym nadchodzącym pakiecie danych, pozwalając mu zsynchronizować się i przygotować do odbioru.

Bit stopu

Po przesłaniu bitów danych i opcjonalnego bitu parzystości, bit stopu sygnalizuje koniec ramki danych. Nadajnik przywraca linię danych do wysokiego poziomu napięcia (stan logiczny 1) na czas trwania 1 bitu lub 2 bitów, w zależności od konfiguracji. Bit stopu zapewnia odbiornikowi czas na przetworzenie ostatniego bitu danych i przygotowanie się na kolejny bit startu. W istocie bit stopu działa jak bufor, zapewniając wyraźne rozgraniczenie między kolejnymi ramkami danych i pomagając w utrzymaniu synchronizacji między komunikującymi się urządzeniami.

Bit parzystości

Bit parzystości to opcjonalna funkcja używana do podstawowej kontroli błędów w komunikacji UART. Zapewnia prostą metodę wykrywania błędów, które mogły wystąpić podczas transmisji danych. Bit parzystości może być skonfigurowany jako parzysty lub nieparzysty:

- **Parzystość parzysta.** Bit parzystości jest ustawiany na 0, jeśli liczba jedynek w ramce danych jest parzysta, i na 1, jeśli liczba jedynek jest nieparzysta. Oznacza to, że łączna liczba jedynek (włącznie z bitem parzystości) jest zawsze parzysta.
- **Parzystość nieparzysta.** Bit parzystości jest ustawiany na 0, jeśli liczba jedynek w ramce danych jest nieparzysta, i na 1, jeśli liczba jedynek jest parzysta. Oznacza to, że całkowita liczba jedynek (włącznie z bitem parzystości) jest zawsze nieparzysta.

Gdy odbiornik otrzymuje ramkę danych, sprawdza bit parzystości w odniesieniu do odebranych bitów danych. Jeśli występuje niezgodność, oznacza to, że podczas transmisji wystąpił błąd. Chociaż parzystość nie koryguje błędów, pomaga w ich identyfikacji, umożliwiając w razie potrzeby ponowne przesłanie danych.

Bity startu, stopu i parzystości są kluczowymi elementami komunikacji UART. Każdy z nich odgrywa istotną rolę w zapewnieniu integralności danych i synchronizacji. Bit startu sygnalizuje początek transmisji, bit stopu oznacza jej koniec, a bit parzystości zapewnia podstawowy mechanizm kontroli błędów. Razem tworzą solidne ramy dla niezawodnej i efektywnej komunikacji szeregowej między urządzeniami.

Zanim zakończymy ten podrozdział, przyjrzymy się jednostce prędkości stosowanej w komunikacji UART.

Prędkość w bodach — szybkość komunikacji w systemach wbudowanych

W świecie systemów wbudowanych często spotyka się pojęcie **prędkości w bodach** (ang. *baud rate*). Niezależnie od tego, czy debugujesz mikrokontroler, konfigurujesz łącze komunikacji szeregowej czy pracujesz z różnymi urządzeniami peryferyjnymi, zrozumienie prędkości w bodach ma kluczowe znaczenie. Czym jednak właściwie jest prędkość w bodach i dlaczego jest tak ważna? Przyjrzymy się temu bliżej.

Czym jest prędkość w bodach?

Prędkość w bodach to zasadniczo szybkość, z jaką dane są przesyłane przez kanał komunikacyjny. Mierzy się ją w **bitach na sekundę (b/s)**. Można to porównać do ograniczenia prędkości na autostradzie: im wyższa prędkość modulacji, tym więcej danych może przepłynąć przez kanał komunikacyjny w danym czasie.

Na przykład, prędkość 9600 bodów oznacza, że w ciągu sekundy przesyłanych jest 9600 *bitów* danych. Innymi słowy, określa ona tempo, w jakim pakiety danych są wysyłane i odbierane.

Warto jednak rozróżnić prędkość w bodach od **prędkości w bitach** (ang. *bit rate*). Podczas gdy prędkość w bodach odnosi się do liczby zmian sygnału na sekundę, prędkość w bitach to liczba bitów przesyłanych na sekundę. W prostych systemach *każda zmiana sygnału może reprezentować jeden bit*, co sprawia, że prędkość w bodach i prędkość w bitach są takie same. W bardziej złożonych systemach każda zmiana sygnału może reprezentować wiele bitów, przez co prędkość w bitach jest wyższa od prędkości w bodach.

Dlaczego prędkość transmisji jest ważna?

Wyobraź sobie, że próbujesz rozmawiać z kimś, kto mówi w zupełnie innym tempie niż Ty. Byłoby to dezorientujące i nieefektywne, prawda? Ta sama zasada dotyczy urządzeń elektronicznych komunikujących się ze sobą. Zarówno urządzenie nadające, jak i odbierające muszą uzgodnić wspólną prędkość w bodach, aby poprawnie się komunikować. Jeśli tego nie zrobią, dane mogą zostać utracone lub zniekształcone, co prowadzi do błędów w komunikacji.

Aby komunikacja przebiegła pomyślnie, nadajnik i odbiornik muszą mieć ustawioną tę samą prędkość transmisji, co pozwala im na osiągnięcie synchronizacji. Jeśli jedno urządzenie jest ustawione na 9600 b/s, a drugie na 115 200 b/s, komunikacja się nie powiedzie, podobnie jak rozmowa nie udaje się, gdy jedna osoba mówi zbyt szybko lub zbyt wolno, by druga mogła ją zrozumieć.

Istnieją standardowe prędkości transmisji, które są powszechnie stosowane w komunikacji szeregowej. Oto kilka z nich:

- **300 bodów.** Bardzo niska prędkość, często używana do komunikacji na duże odległości, kiedy przepustowość jest ograniczona.
- **9600 bodów.** Powszechnie stosowana domyślna prędkość wielu urządzeń, w tym mikrokontrolerów.
- **19 200 bodów.** Większa prędkość, często używana w aplikacjach wymagających intensywniejszej wymiany danych.
- **115 200 bodów.** Komunikacja wysokiej prędkości, popularna w zastosowaniach wymagających szybkiego przesyłu danych.

Na tym zakończymy przegląd protokołu UART. W następnym punkcie omówimy układ peryferyjny UART w mikrokontrolerze STM32F4.

Układ peryferyjny UART w mikrokontrolerze STM32F4

Mikrokontrolery STM32 często zawierają kilka układów peryferyjnych UART, choć ich liczba zależy od konkretnego modelu. Mikrokontroler STM32F411 jest wyposażony w trzy układy peryferyjne UART:

- USART1,
- USART2,
- USART6.

USART a UART

W dokumentacji STM32 układ peryferyjny UART jest nazywany USART, co oznacza uniwersalny synchroniczny/asynchroniczny nadajnik-odbiornik. Nazwa ta odzwierciedla podwójną funkcjonalność tego układu:

Tryb asynchroniczny (UART). W tym trybie USART działa jako tradycyjny UART. Nadaje i odbiera dane bez sygnału zegarowego, co jest typowe dla standardowej komunikacji szeregowej.

Tryb synchroniczny (USART). W tym trybie USART może również działać z synchronicznym sygnałem zegarowym, umożliwiając komunikację z urządzeniami wymagającymi linii zegarowej oprócz linii danych.

Przeanalizujmy teraz kluczowe rejestry tego układu peryferyjnego, zaczynając od rejestru stanu USART.

Rejestr stanu USART (USART_SR)

USART_SR jest jednym z głównych rejestrów używanych do monitorowania stanu interfejsu UART. Dostarcza on informacji w czasie rzeczywistym o różnych flagach operacyjnych i błędach.

Przyjrzyjmy się kluczowym bitom tego rejestru:

- **Rejestr pustych danych nadawczych (TXE).** Ten bit jest ustawiany, gdy rejestr danych jest pusty i gotowy na przyjęcie nowych danych do wysłania. Oznacza to, że nadajnik może wysłać więcej danych.
- **Rejestr niepustych danych odbiorczych (RXNE).** Ten bit wskazuje, że rejestr danych zawiera dane, które nie zostały jeszcze odczytane. Sygnalizuje to, że są dostępne nowe dane do przetworzenia.
- **Transmisja zakończona (TC).** Ten bit jest ustawiany po zakończeniu ostatniej transmisji, włącznie ze wszystkimi bitami stopu. Oznacza to, że dane zostały w pełni wysłane.
- **Błąd przepełnienia (ORE).** Ten bit wskazuje, że dane zostały utracone, ponieważ rejestr danych nie został odczytany przed nadejściem nowych danych. Sygnalizuje to wystąpienie błędu.

Szczegółowe informacje na temat tego rejestru można znaleźć na stronie 547 podręcznika referencyjnego STM32F411 (*RM0383*). Kolejnym ważnym rejestrem jest **rejestr danych USART (USART_DR)**.

Rejestr danych USART (USART_DR)

Rejestr USART_DR służy zarówno do wysyłania, jak i odbierania danych. Odgrywa on rolę głównego interfejsu do wymiany informacji przez moduł UART.

Oto kluczowe funkcje tego rejestru:

- **Transmisja danych.** Zapisanie bajta do rejestru USART_DR powoduje wysłanie danych przez linię TX. Moduł UART automatycznie zajmuje się konwersją i szeregową transmisją danych.
- **Odbieranie danych.** Odczyt z rejestru USART_DR pobiera dane odebrane przez linię RX. Należy to robić niezwłocznie, aby uniknąć nadpisania danych.

Następnie mamy **rejestr prędkości transmisji USART (USART_BRR)**.

Rejestr prędkości transmisji USART (USART_BRR)

Rejestr USART_BRR służy do ustawiania prędkości transmisji (w bodach) w komunikacji UART, co ma kluczowe znaczenie dla synchronizacji przesyłania danych między urządzeniami.

Rejestr ten składa się z dwóch pól:

- **Mantysa.** Część całkowita dzielnika określającego prędkość transmisji.
- **Ułamek.** Część ułamkowa dzielnika, która pozwala na dokładne dostrojenie prędkości transmisji.

Ostatnim rejestrem, któremu się przyjrzymy, jest **1. rejestr sterujący USART** (USART_CR1).

1. rejestr sterujący USART 1 (USART_CR1)

Rejestr USART_CR1 to kompleksowy rejestr sterujący, który umożliwia konfigurację różnych funkcji interfejsu UART.

Przyjrzymy się kluczowym bitom tego rejestru:

- **Włączenie USART (UE).** Ten bit włącza lub wyłącza moduł UART. Trzeba go ustawić w celu aktywowania komunikacji UART.
- **Długość słowa (M).** Ten bit konfiguruje długość słowa, pozwalając wybrać 8-bitowe lub 9-bitowe ramki danych.
- **Włączenie kontroli parzystości (PCE).** Ten bit włącza sprawdzanie parzystości w celu wykrywania błędów.
- **Wybór parzystości (PS).** Ten bit wybiera parzystość parzystą lub nieparzystą.
- **Włączenie nadajnika (TE).** Ten bit włącza nadajnik, umożliwiając wysyłanie danych.
- **Włączenie odbiornika (RE).** Ten bit włącza odbiornik, umożliwiając odbieranie danych.

Teraz, kiedy znasz już te rejestry, jesteśmy gotowi do napisania sterownika UART. Zajmiemy się tym w następnym podrozdziale.

Tworzenie sterownika UART

W tym rozdziale wykorzystasz wszystko, czego nauczyłeś się o układzie peryferyjnym UART, aby napisać sterownik do transmisji danych za pośrednictwem układu USART2.

Zacznijmy od zidentyfikowania pinów GPIO podłączonych do układu USART2. W tym celu należy zajrzeć do tabeli na stronie 39 karty katalogowej *STM32F411RE*. Tabela ta zawiera listę wszystkich pinów GPIO mikrokontrolera wraz z ich opisami i dodatkowymi funkcjami. Jak pokazano na rysunku 10.3, część tej tabeli wskazuje, że pin PA1 ma funkcję alternatywną oznaczoną jako USART2_TX.

12	16	E5	25	K3	<u>PA2</u>	I/O	FT	-	TIM2_CH3, TIM5_CH3, TIM9_CH1, I2S2_CKIN, <u>USART2_TX</u> , EVENTOUT	ADC1_2
----	----	----	----	----	------------	-----	----	---	---	--------

Rysunek 10.3. Pin USART2_TX

Aby użyć pinu PA2 jako linii USART2_TX, musimy skonfigurować go do trybu funkcji alternatywnej w rejestrze GPIOA_MODER, a następnie określić numer funkcji alternatywnej USART2_TX w rejestrze GPIOA_AFR1. Mikrokontroler STM32F4 pozwala wybrać jedną spośród 16 różnych funkcji alternatywnych, ponumerowanych od AF00 do AF15. Funkcje

i ich numery są opisane w tabeli funkcji alternatywnych, którą można znaleźć na stronie 47 karty katalogowej. Jak pokazano na rysunku 10.4, zaczerpniętym z karty katalogowej, skonfigurowanie pinu PA2 jako AF07 ustawi go w tryb linii USART2_TX.

	AF00	AF01	AF02	AF03	AF04	AF05	AF06	AF07	AF08	AF09	AF10	AF11	AF12	AF13	AF14	AF15
Port	SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SP11/I2S1S PI2/ I2S2/SP13/ I2S3	SP12/I2S2/ SP13/ I2S3/SP14/ I2S4/SP15/ I2S5	SP13/I2S3/ USART1/ USART2	USART6	I2C2/ I2C3	OTG1_FS		SDIO			
PA0	-	TIM2_CH1/ TIM2_ETR	TIM5_CH1	-	-	-	-	USART2_ CTS	-	-	-	-	-	-	-	EVENT OUT
PA1	-	TIM2_CH2	TIM5_CH2	-	-	SP14_MOSI I2S4_SD	-	USART2_ RTS	-	-	-	-	-	-	-	EVENT OUT
PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	I2S2_CKIN	-	USART2_ TX	-	-	-	-	-	-	-	EVENT OUT

Rysunek 10.4. Funkcja alternatywna PA2

Teraz mamy wszystkie informacje potrzebne do napisania sterownika nadajnika UART2.

Utwórz kopię poprzedniego projektu i zmień jej nazwę na UART. Następnie utwórz nowy plik o nazwie *uart.c* w folderze *Src* oraz kolejny plik o nazwie *uart.h* w folderze *Inc*. W pliku *uart.c* wpisz następujący kod:

```
#include <stdint.h>
#include "uart.h"

#define GPIOAEN      (1U<<0)
#define UART2EN      (1U<<17)

#define DBG_UART_BAUDRATE      115200
#define SYS_FREQ                16000000
#define APB1_CLK                SYS_FREQ
#define CR1_TE                  (1U<<3)
#define CR1_UE                  (1U<<13)
#define SR_TXE                  (1U<<7)

static void uart_set_baudrate(uint32_t periph_clk,uint32_t baudrate);
static void uart_write(int ch);

int __io_putchar(int ch)
{
    uart_write(ch);
    return ch;
}

void uart_init(void)
{
    /* Włączanie dostępu do zegara dla GPIOA */
    RCC->AHB1ENR |= GPIOAEN;

    /* Ustawianie trybu PA2 na tryb funkcji alternatywnej */
    GPIOA->MODER &=~(1U<<4);
    GPIOA->MODER |= (1U<<5);

    /* Ustawianie typu funkcji alternatywnej na AF7(USART2_TX) */
    GPIOA->AFR[0] |= (1U<<8);
```

```

GPIOA->AFR[0] |= (1U<<9);
GPIOA->AFR[0] |= (1U<<10);
GPIOA->AFR[0] &= ~(1U<<11);

/* Włączanie dostępu do zegara dla UART2 */
RCC->APB1ENR |= UART2EN;

/* Konfigurowanie prędkości transmisji UART */
uart_set_baudrate(APB1_CLK,DBG_UART_BAUDRATE);

/* Konfigurowanie kierunku transmisji */
USART2->CR1 = CR1_TE;

/* Włączanie modułu UART */
USART2->CR1 |= CR1_UE;
}

static void uart_write(int ch)
{
    /* Upewniamy się, że rejestr transmisji jest pusty */
    while(!(USART2->SR & SR_TXE)){

        /* Zapisujemy dane do rejestru transmisji */
        USART2->DR =(ch & 0xFF);
    }

    static uint16_t compute_uart_bd(uint32_t periph_clk,uint32_t baudrate)
    {
        return((periph_clk + (baudrate/2U))/baudrate);
    }

    static void uart_set_baudrate(uint32_t periph_clk,uint32_t baudrate)
    {
        USART2->BRR = compute_uart_bd(periph_clk,baudrate);
    }
}

```

Przeanalizujmy ten kod.

Najpierw dołączamy pliki nagłówkowe i definiujemy makra:

```

#include <stdint.h>
#include "uart.h"

#define GPIOAEN (1U<<0)
#define UART2EN (1U<<17)

#define DBG_UART_BAUDRATE 115200
#define SYS_FREQ 16000000
#define APB1_CLK SYS_FREQ
#define CR1_TE (1U<<3)
#define CR1_UE (1U<<13)
#define SR_TXE (1U<<7)

```

Makr tych użyjemy do następujących celów:

- `GPIOAEN`. To makro włącza zegar dla GPIOA poprzez ustawienie bitu 0. W rejestrze `AHB1ENR`.
- `UART2EN`. To makro włącza zegar dla UART2 poprzez ustawienie bitu 17. W rejestrze `APB1ENR`.
- `DBG_UART_BAUDRATE`. To makro definiuje prędkość transmisji dla komunikacji UART, ustawiając ją na 115 200 bodów.
- `SYS_FREQ`. To makro definiuje częstotliwość zegara systemowego, 16 MHz, która jest domyślną częstotliwością mikrokontrolera STM32F411 na płycie rozwojowej NUCLEO.
- `APB1_CLK`. To makro ustawia częstotliwość zegara dla urządzeń peryferyjnych na magistrali APB1 na wartość równą częstotliwości systemowej (16 MHz).
- `CR1_TE`. To makro włącza nadajnik przez ustawienie bitu 3. w rejestrze `USART_CR1`.
- `CR1_UE`. To makro włącza moduł UART poprzez ustawienie bitu 13. w rejestrze `USART_CR1`.
- `SR_TXE`. To makro reprezentuje bit TXE w rejestrze `USART_SR`.

Następnie mamy funkcje pomocnicze do obliczania i ustawiania prędkości transmisji:

```
static uint16_t compute_uart_bd(uint32_t periph_clk, uint32_t baudrate)
{
    return ((periph_clk + (baudrate / 2U)) / baudrate);
}
```

Ta funkcja pomocnicza oblicza dzielnik dla prędkości transmisji. Wykorzystuje częstotliwość zegara układów peryferyjnych oraz żadaną prędkość transmisji do wyliczenia wartości, która zostanie wpisana do **rejestru prędkości transmisji (BRR)**.

```
static void uart_set_baudrate(uint32_t periph_clk, uint32_t baudrate)
{
    USART2->BRR = compute_uart_bd(periph_clk, baudrate);
}
```

Ta funkcja ustawia szybkość transmisji dla UART2 poprzez zapisanie obliczonego dzielnika do rejestru BRR. Przyjrzyjmy się teraz funkcji inicjalizacyjnej.

```
RCC->AHB1ENR |= GPIOAEN;
```

Powyższy wiersz włącza zegar dla GPIOA poprzez ustawienie odpowiedniego bitu w rejestrze włączania zegara dla urządzeń peryferyjnych `AHB1`.

```
GPIOA->MODER &= ~(1U << 4);
GPIOA->MODER |= (1U << 5);
```

Powyższe wiersze konfiguruje pin PA2 do pracy w trybie funkcji alternatywnej, co jest niezbędne do obsługi interfejsu UART.

```
GPIOA->AFR[0] |= (1U << 8);
GPIOA->AFR[0] |= (1U << 9);
GPIOA->AFR[0] |= (1U << 10);
GPIOA->AFR[0] &= ~(1U << 11);
```

Powyższe wiersze konfiguruje pin PA2 w tryb funkcji alternatywnej (AF7), który odpowiada funkcji UART2_TX.

```
RCC->APB1ENR |= UART2EN;
```

Powyższy wiersz włącza zegar dla UART2 poprzez ustawienie odpowiedniego bitu w rejestrze włączającym zegar dla urządzeń peryferyjnych na magistrali APB1.

```
uart_set_baudrate(APB1_CLK, DBG_UART_BAUDRATE);
```

To wywołanie funkcji ustawia prędkość transmisji dla UART2 za pomocą funkcji `uart_set_baudrate()`.

```
USART2->CR1 = CR1_TE;
```

Powyższy wiersz konfiguruje UART2 do transmisji poprzez ustawienie bitu włączającego nadajnik w rejestrze sterującym.

```
USART2->CR1 |= CR1_UE;
```

Powyższy wiersz włącza moduł UART2 poprzez ustawienie bitu włączenia UART w rejestrze sterującym.

Następnie mamy funkcję do zapisywania danych do UART:

```
static void uart_write(int ch)
{
    /* Upewniamy się, że rejestr transmisji jest pusty */
    while(!(USART2->SR & SR_TXE)) {}

    /* Zapisujemy dane do rejestru transmisji */
    USART2->DR = (ch & 0xFF);
}
```

Przeanalizujmy ją:

```
while (!(USART2->SR & SR_TXE)) {}
```

Ta pętla gwarantuje, że rejestr danych wyjściowych będzie pusty, zanim zapiszemy do niego dane.

```
USART2->DR = (ch & 0xFF);
```

Powyższy wiersz zapisuje znak do rejestru danych w celu transmisji.

Mamy też użyteczną funkcję, która pozwala przekierować wyjście funkcji `printf` do naszego nadajnika UART:

```
int __io_putchar(int ch)
{
    uart_write(ch);
    return ch;
}
```

Funkcja ta wywołuje `uart_write()` w celu wysłania znaku, a następnie zwraca ten sam znak.

Po wysłaniu znaku `__io_putchar` zwraca przekazany znak `ch`.

Zwracanie znaku jest standardową praktyką, pozwalającą funkcji zachować zgodność z typową sygnaturą funkcji `putchar`, która zwraca zapisany znak jako zmienną typu `int`.

Naszym kolejnym zadaniem jest uzupełnienie pliku *uart.h*. Oto kod:

```
#ifndef _UART_H
#define _UART_H
#include "stm32f4xx.h"
void uart_init(void);
#endif
```

Deklarujemy tu po prostu funkcję inicjalizacji UART zaimplementowaną w pliku *uart.c*, umożliwiając wywołanie jej z innych plików. Teraz możemy przetestować nasz sterownik w pliku *main.c*. Zaktualizuj plik *main.c* w następujący sposób:

```
#include <stdio.h>
#include "uart.h"
int main(void)
{
    /* Inicjalizacja UART */
    uart_init();
    while(1)
    {
        printf("STM32 - witamy...\r\n");
    }
}
```

Funkcja *main* po prostu inicjalizuje interfejs UART2, a następnie w nieskończonej pętli wypisuje zdanie STM32 - witamy...

Przetestujmy nasz projekt. W tym celu musisz zainstalować na komputerze program, który będzie wyświetlał dane odbierane przez port szeregowy komputera. W tym układzie nasza płytka rozwojowa działa jako nadajnik, a komputer jako odbiornik.

1. Zainstaluj program terminala szeregowego:

- Wybierz program terminala szeregowego odpowiedni dla Twojego systemu operacyjnego. Dostępne opcje to między innymi *Realterm*, *Tera Term*, *Hercules* i *Cool Term*.
- Jeśli korzystasz z systemu Windows, polecam *Realterm*. Możesz go pobrać ze strony SourceForge: <https://sourceforge.net/projects/realterm/>.
- Postępuj zgodnie ze wskazówkami kreatora, aby zainstalować program.

2. Przygotuj się do zidentyfikowania portu szeregowego płytki rozwojowej:

- I. Odłącz płytkę rozwojową od komputera.
- II. Otwórz program *Realterm* i przejdź do karty *Port*.
- III. Kliknij menu rozwijane *Port*; zobaczysz listę dostępnych portów. Ponieważ płytka rozwojowa jest obecnie odłączona, jej port nie pojawi się na liście. Zapamiętaj wyświetlone porty.

3. Zidentyfikuj port płytki rozwojowej:

- I. Zamknij *Realterm* i podłącz płytkę rozwojową do komputera.
- II. Ponownie otwórz *Realterm* i wróć do rozwijanego menu *Port*. Powinieneś teraz zobaczyć na liście nowy port, który odpowiada twojej płytce rozwojowej.
- III. Wybierz ten nowo dodany port.

4. Ustaw prędkość transmisji:

Wybierz z rozwijanego menu opcję *Baud* i wybierz wartość *115200*. Jest to prędkość transmisji, którą skonfigurowaliśmy w naszym sterowniku. Aby zmienić ustawienia, kliknij przycisk *Change*.

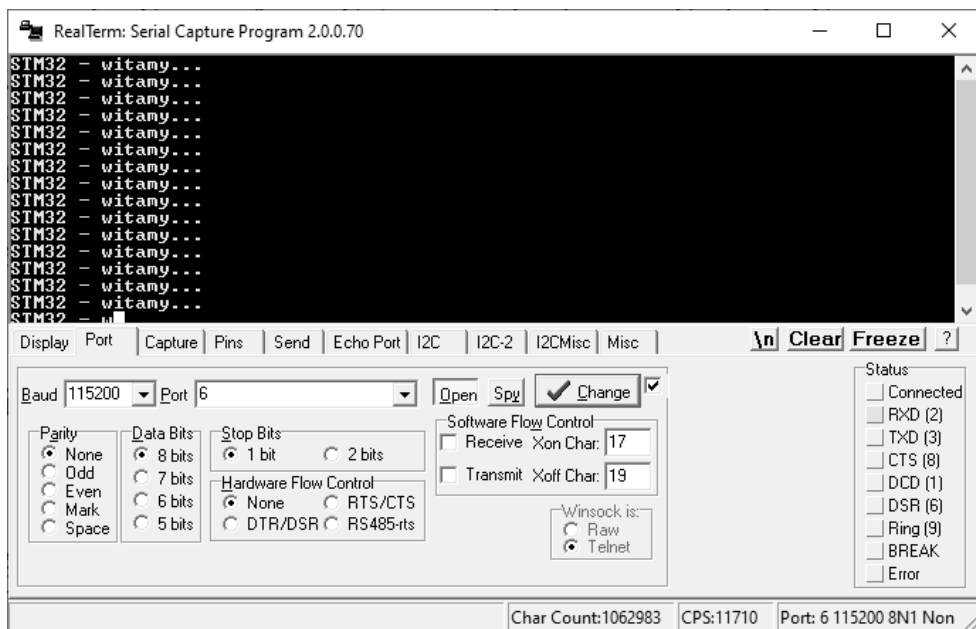
5. Zbuduj i uruchom projekt:

Wróć do środowiska programistycznego, zbuduj projekt i wgraj oprogramowanie na mikrokontroler.

6. Przetestuj konfigurację:

- Wróć do programu Realterm i kliknij przycisk *Open*, aby rozpocząć komunikację.
- W oknie terminala powinieneś zobaczyć ciągle powtarzany komunikat STM32 - witamy...

Ustawienia programu Realterm pokazano na rysunku 10.5.



Rysunek 10.5. Ustawienia programu Realterm

Podsumowanie

W tym rozdziale poznałeś protokół UART — prostą metodę komunikacji szeroko stosowaną w systemach wbudowanych. Rozpoczęliśmy od omówienia znaczenia protokołów komunikacyjnych w systemach wbudowanych, podkreślając, jak UART, wraz z SPI i I2C, umożliwiają komunikację między mikrokontrolerami a urządzeniami peryferyjnymi.

Następnie przedstawiliśmy szczegółowy przegląd protokołu UART, omawiając jego zasady działania, w tym sposób asynchronicznego przesyłania danych za pomocą bitów startu i stopu oraz rolę bitu parzystości w wykrywaniu błędów. Dowiedziałeś się również, jak konfiguruje się prędkość transmisji w bodach, aby zapewnić zsynchronizowany transfer danych między urządzeniami.

Później zagłęбилиśmy się w szczegóły dotyczące układu peryferyjnego UART w mikrokontrolerach STM32, analizując kluczowe rejestry, takie jak rejestr statusu (USART_SR), rejestr danych (USART_DR), rejestr prędkości transmisji (USART_BRR) i 1. rejestr sterujący (USART_CR1). Zrozumienie tych rejestrów jest niezbędne do prawidłowej konfiguracji UART w mikrokontrolerach STM32.

Na koniec wykorzystaliśmy teoretyczną wiedzę do utworzenia niskopoziomowego sterownika UART dla mikrokontrolera STM32F4. Obejmowało to: inicjalizację układu peryferyjnego UART, ustawienie prędkości transmisji i implementację funkcji do przesyłania danych. Dowiedziałeś się również, jak przekierować wyjście funkcji `printf` do UART, co umożliwia łatwe debugowanie i rejestrowanie danych przez terminal szeregowy.

W następnym rozdziale zajmiemy się **przetwornikiem analogowo-cyfrowym** (ang. *analog-to-digital converter*, ADC).

Skorowidz |

A

- ADC, Analog-to-Digital Converter, 54, 210
- ADC_CR1, 218
- ADC_CR2, 219
- ADC_DR, 219
- ADC_SQRx, 219
- ADC_SR, 220
- adres
 - GPIOA MODER, 62
 - GPIOA ODR, 64
 - LMA, 97, 105
 - początkowy GPIOA, 56
 - RCC_AHB1ENR, 59
 - VMA, 97, 105
- adresy bazowe
 - GPIO, 142
 - RCC, 57, 142
- ADXL345, 243
 - cechy akcelerometru, 244
 - funkcja pomiarowa, 244
 - podłączenie do płytki NUCLEO, 253
 - tworzenie sterownika, 248
 - zastosowania, 244
- AHB, Advanced High-Performance Bus, 54
- AHB1, Advanced High-Performance Bus 1, 58
- AHB1 ENR, 58
- akcelerometr ADXL345, 243
- alias, 68
- APB, Advanced Peripheral Bus, 54
- Arduino, 48
- aseblacja, 76
- assembler, 42

B

- bezpośredni dostęp do pamięci, DMA, 331, 332
- biblioteka
 - HAL, 42
 - niższego poziomu, LL, 42

- bit, 59
 - GPIOAEN, 59
 - parzystości, 198
 - startu, 198
 - stopu, 198
- bity
 - ustawianie, 60, 158, 160
 - zerowanie, 60, 158, 160
- bramkowanie
 - zasilania, 359
 - zegara, clock gating, 56, 359
- budowanie oprogramowania, 74
 - etapy procesu, 75–77
 - system Make, 127
 - w środowisku IDE, 83

C

- CMSIS, Common Microcontroller Software Interface Standard, 140, 146
 - integrowanie plików, 151
 - komponenty, 147
 - komunikacja z mikrokontrolerami, 147
 - programowanie procesorów, 140
 - zasady kodowania, 148
- CMSIS-Core, 147, 148
- CMSIS-Driver, 147
- CMSIS-DSP, 147
- CMSIS-NN, 147
- CMSIS-Pack, 148
- CMSIS-RTOS, 147
- CMSIS-SVD, 148
- CPU, Central Processing Unit, 53

D

- debuger GDB, 25, 91
- definiowanie rejestrów, 52, 66, 141
- deklaracja (*volatile unsigned int *), 67
- diody LED
 - lokalizowanie połączenia, 47

DMA, 331, 332
 cechy kontrolera, 332, 335
 działanie, 332
 rejestry, 339
 schemat blokowy modułu, 337
 testowanie projektu, 352, 356
 tryby danych, 337
 tryby transferu, 336
 tworzenie sterownika, 341
 w mikrokontrolerze STM32F4, 335
 zastosowania, 333
 DMA_SxCR, 339
 DMA_SxM0AR, 340
 DMA_SxM1AR, 340
 DMA_SxNDTR, 340
 DMA_SxPAR, 340
 dokumentacja, 52
 STMicroelectronics, 34
 dokumenty, 36
 dostęp do rejestrów, 145
 dupleks, 191
 DVFS, 358
 dynamiczne skalowanie napięcia i
 częstotliwości, DVFS, 358
 dyrektywa
 >region, 106
 ALIGN, 107
 AT, 108
 ENTRY, 102
 KEEP, 106
 MEMORY, 101
 PROVIDE, 107
 SECTIONS, 102

E

ENR, Enable Register, 44, 58
 EXTI, 275, 369
 kluczowe cechy, 283
 tworzenie sterownika, 285
 w STM32, 283
 EXTI_FTSR, 285
 EXTI_IMR, 285
 EXTI_PR, 285
 EXTI_RTSR, 285

F

flagi
 ADC, 220
 kompilatora, 78, 79
 specyficzne dla architektury, 79, 80

format
 BCD, 310
 ELF, 77
 funkcja main, 69
 funkcje alternatywne GPIO, 161, 163

G

gęstość pamięci, 94
 GNU Arm Embedded Toolchain, 25
 instalacja, 27
 GNU Compiler Collection, 25
 GNU Debugger, 25
 goldpiny, 49, 51
 GPIO, General Purpose Input/Output, 42, 141,
 156
 konfigurowanie pinu, 42, 44, 45
 konfigurowanie pinu ADC, 221
 rejestr BSRR, 164
 rejestry, 158
 rejestry funkcji alternatywnych, 161
 sterownik wejścia, 167
 sterowniki, 164
 GPIO PORTA, 52
 GPIOx_AFRH, 161
 GPIOx_AFRL, 161
 GPIOx_BSRR, 158, 160
 GPIOx_IDR, 158, 159
 GPIOx_LCKR, 158
 GPIOx_MODER, 61, 158
 GPIOx_ODR, 64, 158, 159
 GPIOx_OSPEEDR, 158
 GPIOx_OTYPER, 158
 GPIOx_PUPDR, 158

H

HAL, Hardware Abstraction Layer, 42

I

I2C, Inter-Integrated Circuit, 54, 254
 cechy interfejsu, 193, 255
 działanie magistrali, 256
 interfejs, 255
 kluczowe rejestry, 260
 tworzenie sterownika, 262
 zastosowania magistrali, 196
 I2C_CCR, 261
 I2C_CR1, 260
 I2C_CR2, 261
 I2C_DR, 262

I2C_TRISE, 262
IAR Embedded Workbench, 24
IDE, Integrated Development Environment, 23
ikona
 budowania, 71
 Build, 40
 Debug, 40
 New, 40
 uruchamiania, 71
instalacja
 GNU Arm Embedded Toolchain, 27
 OpenOCD, 28
 STM32CubeIDE, 25
 systemu Make, 132
instrukcja użytkownika NUCLEO-F411, 48
instrukcje, 34
interfejs komunikacji szeregowej
 I2C, 54, 193, 255
 SPI, 54, 193, 230
 UART, 53, 192, 197
IWDG, Independent Watchdog,
 Patrz także watchdog
 cechy układu, 323
 działanie modułu, 323
 rejstry, 324
 schemat blokowy, 323
 testowanie projektu, 330
 tworzenie sterownika, 325
 w mikrokontrolerach STM32, 322
IWDG_KR, 324
IWDG_PR, 324
IWDG_RLR, 325
IWDG_SR, 325

J

jednostka
 centralna, CPU, 53
 zarządzania pamięcią, MMU, 105
język C
 dostęp do rejestrów, 145
 sterowanie diodą LED, 69
 sterownik ADC, 221
 sterownik ADXL345, 248
 funkcja `adxl_init`, 250
 funkcja `adxl_read`, 250
 funkcja `adxl_write`, 250
 plik nagłówkowy, 248
 sterownik dla mikrokontrolera, 367
 sterownik DMA
 dla przetwornika ADC, 341
 dla układu UART, 344
 do transferu danych, 352

sterownik EXTI, 286
sterownik GPIO, 164
sterownik I2C, 262
 aktualizacja ASXL345, 270
 funkcja inicjalizacyjna, 262
 funkcja `main`, 272
 funkcja odczytu, 265
 funkcja zapisu, 269
 plik nagłówkowy, 270
sterownik IWDG, 325
 plik implementacji, 325
 plik `main.c`, 328
 plik nagłówkowy, 327
sterownik RTC, 301
 plik implementacji, 302
 plik `main.c`, 313
 plik nagłówkowy, 312
sterownik SPI, 236
 inicjalizacja pinów GPIO, 240
 konfiguracja SPI1, 241
 odbieranie danych, 242
 plik nagłówkowy, 243
 plik `spi.c`, 236
 wysyłanie danych, 242
 zarządzanie linią SS, 242
 zdefiniowane makra, 239
sterownik timera, 175, 186
sterownik UART, 202
struktura układów peryferyjnych, 143
wskaźnik, 67, 146

K

kanal ADC, 215
karta katalogowa, datasheet, 34
katalog
 bin, 85
 Debug, 87
Keil μ Vision, 24
kodowanie, 212
kompilacja, 76
 plików asemblera, 84
 plików C, 84
kompilator GCC, 25
komunikacja
 pełnodupleksowa, 191
 półdupleksowa, 191
 punkt-punkt, 191
 równoległa, 191
 szeregowa, 190
 wielourządzeniowa, 191

- konfigurowanie
 - bitów MODER, 62
 - pinu GPIO
 - z użyciem HAL, 42
 - z użyciem LL, 43
 - za pomocą asemblera, 45
 - za pomocą języka C, 44
 - systemu Make, 132
- konsolidacja, 76, 97
- kontroler
 - przerwań zewnętrznych, EXTI, 275, 283
 - zagnieżdżonych przerwań wektorowych, NVIC, 275, 278
- konwersja analogowo-cyfrowa, 211
 - kodowanie, 212
 - kwantyzacja, 211
 - próbkiowanie, 211

L

- licznik lokacji, 107
- liczniki, 54
- LL, Low Layer, 42
- LMA, 105
- lokalizator, 77, 98
- lokalizowanie
 - goldpinów, 49
 - połączenia diody LED, 47
 - połączenia przycisku użytkownika, 49
 - portu GPIO PORTA, 52
 - złączy kompatybilnych z Arduino, 49

Ł

- łańcuch narzędzi, toolchain
 - GNU Arm Embedded Toolchain, 24

M

- magistrala, 191
 - AHB1, 54, 58
 - AHB2, 54
 - APB1, 54
 - APB2, 54
 - portu GPIOA, 56
 - RCC, 57
- Make, 127, 129
- mapa pamięci
 - mikrokontrolera, 53
 - STM32F11, 94
 - układów peryferyjnych, 55

- maski bitowe, 68
- Maven, 129
- mikrokontroler
 - STM32F4
 - moduły DMA, 335
 - tryby niskiego poboru mocy, 362
 - tworzenie sterownika, 367
 - źródła wybudzania, 364–366
 - STM32F411, 34
 - kanały ADC, 215, 217
 - konfigurowanie projektu, 152
 - mapa pamięci, 53
 - układ peryferyjny ADC, 215
 - układ peryferyjny I2C, 260
 - układ peryferyjny SPI, 234
 - układ wyprowadzeń, 52
 - STM32F411RE, 33
- mikrokontrolery
 - porty, 157
 - rejestry GPIO, 158
 - STM32, 34
 - IWDG, 322
 - moduł EXTI, 283
 - moduł RTC, 293
 - timery, 183
 - układ peryferyjny UART, 200
 - WWDG, 322
 - układy peryferyjne, 53
- model pamięci STM32, 93
- modulacja szerokości impulsu, PWM, 180
- moduł DMA, 335
- multipleksowanie kanałów ADC, 216

N

- napiecie referencyjne, VREF, 213
- narzędzia GNU Arm Embedded Toolchain, 77, 81, 85
- niezależny watchdog, IWDG, 317
- Notepad++, 24
- nowy projekt, 36, 65
- NUCLEO-F411, 33

O

- ODR, Output Data Register, 63, 159
- ogólny przewódnik użytkownika ARM, 35
- OpenOCD
 - instalacja, 28
 - wgrywanie oprogramowania układowego, 89
 - wyświetlanie informacji, 89

operacja bitowa

AND, &, 60

NOT, ~, 60

OR, |, 60

P

pakiet

danych UART, 197

GNU Arm Embedded Toolchain, 77, 81, 85

I2C, 257

STM32CubeF4, 150

pamięć

flash, 94, 112

SRAM, 95, 112

układów peryferyjnych, 96

plik

Makefile, 130

testowanie, 136

tworzenie, 134

zmiennie specjalne, 138

obiekty

sekcje, 97, 104

relokowalny, 98

startowy

testowanie, 124

tworzenie, 117

wykonywalny, 98

pliki

CMSIS, 150, 151

CMSIS-Core, 148

nagłówkowe, 150

płytki rozwojowej, 32

funkcje, 33

NUCLEO-F411, 33

goldpiny, 50

komponenty, 47

podłączenie akcelerometru, 253, 273

systemy nazewnictwa pinów, 49

układ pinów, 51

złącza Arduino, 50

podłączanie

potencjometru, 227

akcelerometru, 253, 273

podręcznik

referencyjny, RM, 34, 58

użytkownika, UM, 35

polecenie

arm-none-eabi-gcc, 78, 80, 82

arm-none-eabi-nm, 81, 82

arm-none-eabi-objcopy, 82

arm-none-eabi-objdump, 81, 82

arm-none-eabi-readelf, 81, 82

arm-none-eabi-size, 81, 82, 88

cd, 87

monitor reset init, 91

porty

GPIO, 52, 157

rejestr MODER, 62

rejestr ODR, 64

zestawy rejestrów, 157

półdupleks, 192

prędkość komunikacji, 199

procedura obsługi przerwań, ISR, 111, 112, 275, 279

proces budowania oprogramowania

układowego, 75

aseblacja, 76

kompilacja, 76

konsolidacja, 76, 97

lokalizacja, 77, 96

przetwarzanie wstępne, 75

program

GDB, 90

Open On-Chip Debugger, 89

RealTerm, 208, 252

programowanie niskopoziomowe, 41

w C, 42

programy

nowy projekt, 36, 65

protokoły komunikacyjne, 190

protokół

I2C, 196, 254

SPI, 195, 229

UART, 189, 194

próbkowanie, 211

przerwania, 112, 276

a wyjątki, 277

działanie, 276

odzworowywanie linii

przerwań/zdarzeń, 284

programowe, 113

sprzętowe, 112

zastosowania, 280

zewnętrzne, 275, 369

znaczenie, 277

przesunięcie, offset, 141

rejestrów GPIO, 142

przetwarzanie wstępne, 75

przetwornik analogowo-cyfrowy, ADC, 54, 96, 159, 181, 210

flagi, 220

kanały wstrzykiwane, 217

kanały zwykłe, 217

konwersja analogowo-cyfrowa, 211

napięcie referencyjne VREF, 214

przetwornik analogowo-cyfrowy, ADC
 rejestry, 218
 rozdzielczość, 213
 tryby pracy, 216
 tworzenie sterownika, 221
 w mikrokontrolerze STM32F4, 215
 wielkość kroku kwantyzacji, 214
 przycisk użytkownika
 lokalizowanie połączenia, 49
 przyrostek UL, 67
 przyspieszenie
 dynamiczne, 247
 grawitacyjne, 246

R

RCC, Reset and Clock Control, 44, 58
 RCC AHB1ENR, 59
 rdzeń Cortex-M4, 36
 rejestry, 59
 adresu pamięci strumienia DMA, 340
 adresu układu peryferyjnego strumienia
 DMA, 340
 alarmów RTC, 301
 automatycznego przeładowania, 183
 blokowania, 158
 BSRR, 164
 czasu
 narastania I2C, 262
 RTC, 299
 danych
 ADC, 219
 I2C, 262
 SPI, 236
 USART, 201
 wejściowych, IDR, 158, 159
 wyjściowych, ODR, 63, 64, 158, 159
 daty RTC, 299
 definiowanie, 52, 66, 141
 funkcji alternatywnych, 161
 GPIO, 158
 inicjalizacji i statusu RTC, 300
 kluczowy IWDG, 324
 konfiguracji strumienia DMA, 339
 liczby danych strumienia DMA, 340
 MODER, 62
 podciągania/ściągnięcia, 158
 preskalera, 183
 IWDG, 324
 RTC, 300
 prędkości transmisji USART, 201
 przeładowania IWDG, 325
 przerwań oczekujących EXTI, 285
 sekwencji zwykłej ADC, 219

stanu USART, 201
 statusu, SR, 183
 ADC, 220
 IWDG, 325
 SPI, 236
 sterowania zegarem I2C, 261
 sterujący
 ADC, 218, 219
 I2C, 260, 261
 RTC, 300
 SPI, 235
 USART 1, 202
 szybkości wyjścia, 158
 timera, 183
 wybudzania RTC, 301
 trybu, 158
 portu GPIO, 61, 62
 tworzenie, 52
 typu wyjścia, 158
 układów peryferyjnych, 141
 ustawianie bitu, 60, 158, 160
 włączania, ENR, 44, 58
 zerowanie bitu, 60, 158, 160
 RM, Reference Manual, 34
 RTC, Real Time Clock, 290
 tworzenie sterownika, 301
 RTC_ALRMAR, 301
 RTC_ALRMBR, 301
 RTC_CR, 300
 RTC_DR, 299
 RTC_ISR, 300
 RTC_PRER, 300
 RTC_TR, 299
 RTC_WUTR, 301
 rzutowanie
 na wskaźnik, 67
 typu, 67

S

schemat blokowy DMA, 337
 IWDG, 323
 RTC, 294–298
 skrypt linkera, 77, 96
 dyrektywy, 100
 opcje i symbole, 100
 prefiksy i sufiksy, 109
 sekcje, 99, 100
 stałe, 108
 symbole, 109
 testowanie, 124
 tworzenie, 111, 113
 układ pamięci, 98

- słowo kluczowe
 - typedef, 143
 - volatile, 66, 68
 - SPI, Serial Peripheral Interface, 54, 229
 - cechy interfejsu, 193, 234
 - kluczowe rejestry, 235
 - testowanie sterownika, 243
 - tworzenie sterownika, 236
 - zastosowania protokołu, 195
 - SPI_CR1, 235
 - SPI_DR, 236
 - SPI_SR, 236
 - SR, status register, 183
 - SRAM, 95
 - sterowanie
 - procesami przemysłowymi, IPC, 181
 - resetem i zegarem, RCC, 44
 - sterownik
 - ADC, 221
 - akcelerometru ADXL345, 248
 - dla mikrokontrolera, 367
 - DMA, 341, 344, 352
 - EXTI, 285
 - GPIO, 164, 167
 - I2C, 262
 - IWDG, 325
 - RTC, 301
 - SPI, 236
 - timera, 186
 - timera SysTick, 175
 - wejścia i wyjścia, 164, 167
 - STM32, 93
 - pamięć flash, 94
 - pamięć SRAM, 95
 - STM32CubeIDE, 24
 - ikony sterujące, 40
 - instalacja, 25
 - karta
 - Includes, 152
 - Information Center, 37
 - Symbols, 153
 - okienko
 - Console, 84
 - Project Explorer, 37, 39
 - okno
 - Edit Configurations, 72
 - Setup STM32 project, 39
 - Target Selection, 38
 - proces budowania oprogramowania, 83
 - tworzenie projektu, 36
 - struktura układów peryferyjnych, 143
 - symbole linkera, 110
 - system
 - kontroli wersji, SCM, 129
 - Make, 129
 - instalowanie, 132
 - konfigurowanie, 132
 - plik Makefile, 130, 134
 - Maven, 129
 - operacyjny czas rzeczywistego, RTOS, 180
 - systemy budowania oprogramowania, 127
 - SysTick, 171
 - cechy timera, 172
 - rejestry timera, 172
 - tworzenie sterownika, 175
 - zastosowania, 172
- T**
- tabela
 - IVT, 279
 - wektorów przerwań, IVT, 275
 - tablica wektorów, 112
 - testowanie
 - pliku Makefile, 136
 - pliku startowego, 124
 - skryptu linkera, 124
 - TIM, 179
 - timer, 54
 - ogólnego przeznaczenia, TIM, 179
 - generowanie opóźnień, 181
 - pomiar interwałów czasowych, 180
 - wyzwalanie zdarzeń, 181
 - STM32, 181
 - cechy, 182
 - działanie, 183
 - tryby zliczania, 183
 - wstępne skalowanie, 184
 - systemowy SysTick, 171
 - tworzenie sterownika, 186
 - watchdog, WDT, 317
 - tryby niskiego poboru mocy, 360
 - tworzenie
 - kopii zapasowej projektu, 86
 - nowego projektu, 36, 65
 - oprogramowania układowego
 - assembler, 42
 - biblioteka niższego poziomu, LL, 42
 - programowanie niskopoziomowe w C, 42
 - warstwa abstrakcji sprzętowej, HAL, 42
 - pliku Makefile, 134
 - pliku startowego, 117

tworzenie

- rejestrów, 52
- skryptu linkera, 111, 113
- sterownika
 - ADC, 221
 - ADXL345, 248
 - dla mikrokontrolera, 367
 - DMA, 341, 344, 352
 - EXTI, 285
 - GPIO, 164
 - I2C, 262
 - IWDG, 325
 - RTC, 301
 - SPI, 236
 - timera, 175, 186
 - UART, 202
- sterowników wejścia i wyjścia, 164
- struktury języka C, 143

U

- UART, Universal Asynchronous Receiver/Transmitter, 53, 189, 197
 - cechy interfejsu, 192
 - działanie, 197
 - interfejs, 197
 - tworzenie sterownika, 202
 - zastosowania protokołu, 194
- UEV, update event, 183, 185
- układ peryferyjny
 - ADC, 215
 - GPIO, 157
 - I2C, 260
 - RCC, 57, 58
 - SPI, 234
 - UART, 200
- układy peryferyjne
 - adres bazowy, 54
 - definiowanie rejestrów, 141
 - implementowanie struktury, 143
 - interfejsy komunikacyjne, 53
 - mapa pamięci, 55
 - porty GPIO, 53
 - przetworniki analogowo-cyfrowe, 54
 - timery i liczniki, 54
- UM, User Manual, 35
- uniwersalny asynchroniczny odbiornik-nadajnik, UART, 197
- USART_BRR, 201
- USART_CR1, 202
- USART_DR, 201
- USART_SR, 201

V

- VMA, 105
- VREF, 214

W

- warstwa abstrakcji sprzętowej, HAL, 42
- watchdog, 318
 - działanie, 318
 - typy, 320
 - zastosowania, 319
- wejście-wyjście ogólnego przeznaczenia, *Patrz* GPIO
- wgrywanie oprogramowania układowego, 89
- wiersz poleceń, 83, 87, 88
 - działanie programu GDB, 90
 - działanie programu OpenOCD, 91
- wskaźnik do struktury, -, 146
- WWDG, Window Watchdog, 322
- wyjątki, 277
- wykrywanie nachylenia, 246
- wyłuskanie wskaźnika, 67
- wyzwalanie zdarzeń, 181

Z

- zarządzanie energią, 358
 - bramkowanie zasilania, 359
 - bramkowanie zegara, 359
 - dynamiczne skalowanie napięcia i częstotliwości, 358
 - tryby niskiego poboru mocy, 360, 362
- zdarzenia, 181
- zdarzenie aktualizacji, UEV, 183, 185
- zegar
 - bramkowanie, 56, 359
 - czasu rzeczywistego, RTC, 290
 - cechy modułu, 293
 - działanie, 291
 - kluczowe elementy modułu, 294
 - kluczowe rejestry, 299
 - schemat blokowy, 294–298
 - w mikrokontrolerach STM32, 293
 - zastosowania, 291
 - faza, 253
 - polaryzacja, 253
 - skalowanie, 184
- zintegrowane środowisko programistyczne, IDE
 - IAR Embedded Workbench, 24
 - Keil µVision, 24
 - STM32CubeIDE, 24
- złącza
 - kompatybilne z Arduino, 49, 50

PROGRAM PARTNERSKI

— GRUPY 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
Helion

Zrozum, jak naprawdę działa oprogramowanie sterujące współczesną elektroniką!

Systemy wbudowane sterują niemal każdym nowoczesnym urządzeniem, dlatego umiejętność tworzenia wydajnego i niezawodnego oprogramowania wbudowanego jest niezwykle cenna. Mówimy tu o sprzęcie AGD, systemach sterowania przemysłowego czy urządzeniach IoT. Potrzebują one oprogramowania, które bezpośrednio współpracuje ze sprzętem. Tworzenie takiego kodu nie jest jednak łatwe — wymaga zmiany podejścia i dyscypliny.

Ta książka pomoże Ci opanować sprzętowe zawiłości mikrokontrolerów. Dowiesz się, jak czytać podręczniki i dokumentację, co pozwoli Ci na dogłębne zrozumienie technologii. Nauczysz się od podstaw pisać zoptymalizowany kod i oswoisz się z manipulowaniem rejestrami. Zamiast podążać na skróty, zaczniesz świadomie pisać oprogramowanie układowe, co utoruje Ci drogę do zawodowego mistrzostwa. Po lekturze książki zdobędziesz umiejętności czytania kart katalogowych, wykonywania operacji na rejestrach i tworzenia zoptymalizowanego kodu. Uzyskasz też pewność siebie potrzebną do poruszania się po meandrach sprzętu i samodzielnego pisania zoptymalizowanego oprogramowania układowego. Dzięki temu staniesz się biegłym i samodzielnym programistą systemów wbudowanych.

Najciekawsze zagadnienia:

- zoptymalizowane oprogramowanie układowe
- manipulowanie rejestrami i pisanie optymalnego kodu
- złożone zagadnienia dotyczące sprzętu
- tworzenie niskopoziomowych sterowników układów peryferyjnych
- zapewnianie energooszczędności systemów wbudowanych

Israel Gbati jest wybitnym inżynierem oprogramowania wbudowanego, przedsiębiorcą i nagradzonym wynalazcą. Założył firmy EmbeddedExpertIO i BioStealthAI. Jest znany z tego, że chętnie dzieli się wiedzą z innymi specjalistami, a także ze zdolności przywódczych i z zaangażowania w rozwój innowacji technologicznych.

	KOD KORZYŚCI Sięgnij po więcej! ▶ 
 helion.pl  HELION S.A. ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	ISBN 978-83-289-3344-6  9 788328 933446
Cena: 99,00 zł	