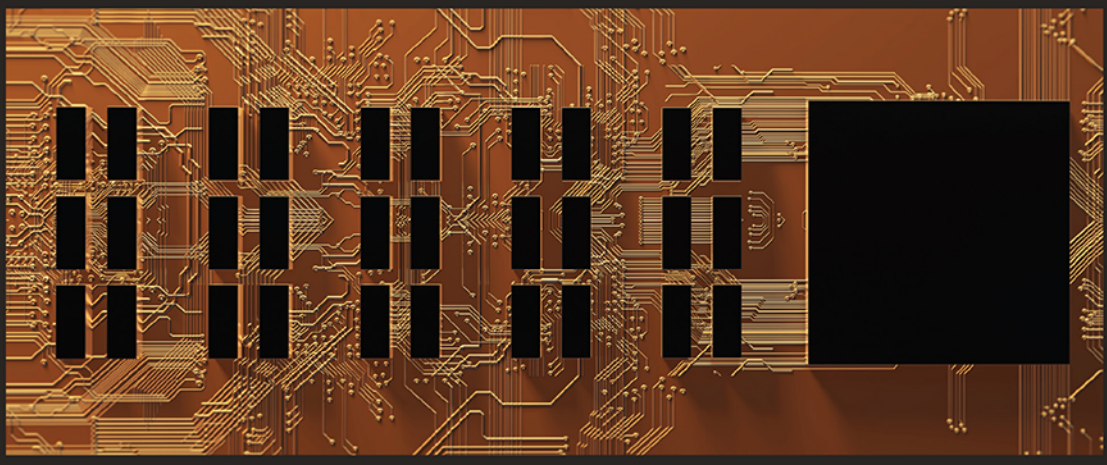
Helion

<packt>



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

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/wysyjc
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.

Spis treści

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.

Zaczniemy 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 przeanalizujmy 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 szeregowa i równoległa

Zacznijmy od komunikacji szeregowej.

Komunikacja szeregowa

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 wielourządzeniową.

Komunikacja punkt-punkt a komunikacja wielourządzeniowa

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 wielourządzeniowa (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 pełnodupleksowa i półdupleksowa

Przyjrzyjmy się różnicom między komunikacją pełnodupleksową i półdupleksową:

- **Pełny dupleks**. 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 pełnodupleksowym, 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 dupleks**. 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 dupleks**. Protokół umożliwia jednoczesne 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 szeregowa z komputerami**. Interfejsu UART często używa się do komunikacji szeregowej 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 szeregowej.

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

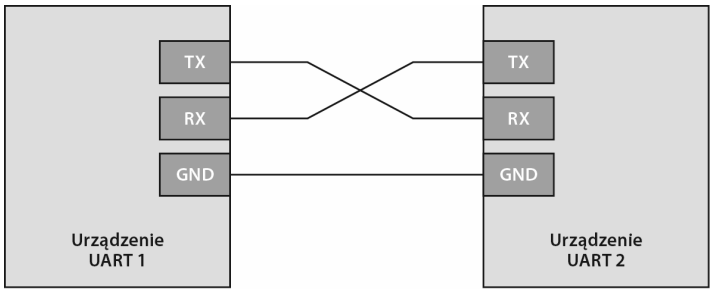
Jednym z najbardziej podstawowych i powszechnie stosowanych protokołów jest UART. Niezależnie od tego, czy debugujesz sprzęt, czy chcesz zapewnić komunikację między mikrokontrolerem a urządzeniami peryferyjnymi, zrozumienie działania UART ma kluczowe znaczenie. Zbadajmy dokładniej, jak działa ten protokół.

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 ramkach 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, przyjrzyjmy 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? Przyjrzyjmy 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.

Przyjrzyjmy 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 `UART2`. 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	PA2	I/O	FT	-	TIM2_CH3, TIM5_CH3, TIM9_CH1, I2S2_CKIN, USART2_TX, 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_AFRL`. 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.

Port	AF00	AF01	AF02	AF03	AF04	AF05	AF06	AF07	AF08	AF09	AF10	AF11	AF12	AF13	AF14	AF15
	SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/I2S1S PI2/ I2S2/SPI3/ I2S3	SPI2/I2S2/ SPI3/ I2S3/SPI4/ I2S4/SPI5/ I2S5	SPI3/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	-	-	SPI4_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(UART2_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 płytce 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 żądaną 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 konfigurują 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 konfigurują 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*.

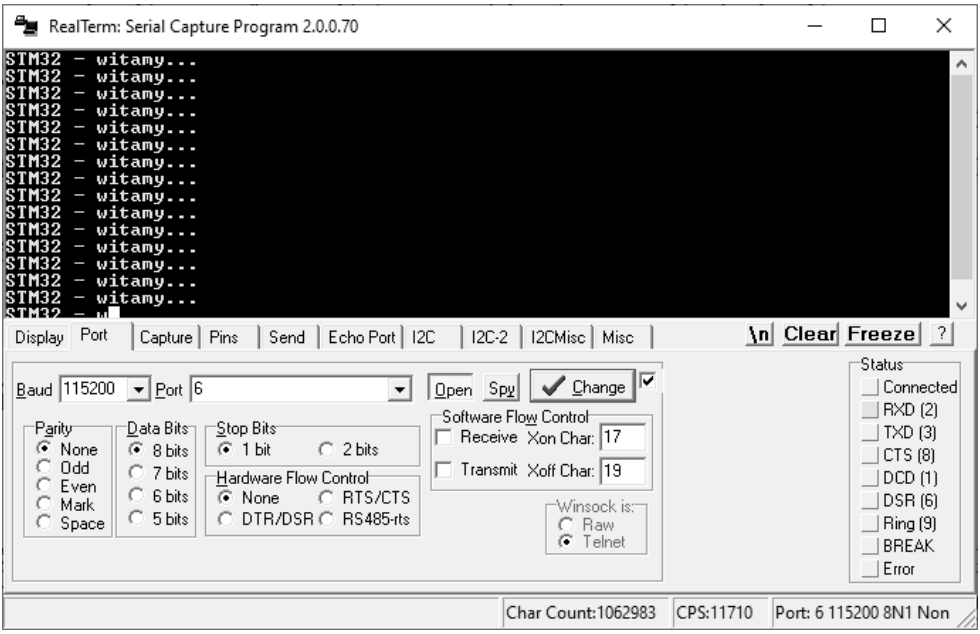
5. **Zbuduj i uruchom projekt**:

   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 zagłębiliśmy 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

J

K

R

S

T

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 nagradzanym 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.

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