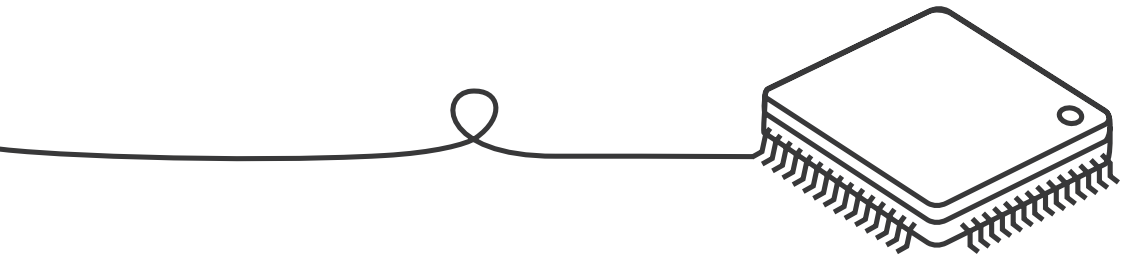
1

Wprowadzenie



1.1. Wstęp

We współczesnej technice powszechne jest wykorzystanie programowalnych układów cyfrowych. Często stanowią one jeden z elementów większego urządzenia bądź systemu i wówczas noszą nazwę „systemów wbudowanych". Zdarza się, że obecność takiego układu jest wręcz niezauważalna dla użytkownika i nie zdaje on sobie nawet sprawy, że gdzieś we wnętrzu danego urządzenia kryje się niewielki system komputerowy. Systemy wbudowane mogą być realizowane na różne sposoby, jednakże najczęściej wykorzystuje się w tym celu mikrokontrolery, czyli cyfrowe systemy mikroprocesorowe zrealizowane w postaci pojedynczego układu scalonego. Zaskoczeniem może być fakt, że są one obecnie najbardziej rozpowszechnioną formą systemów komputerowych. Najlepiej obrazują to dane dotyczące liczby sprzedanych sztuk różnych procesorów. Okazuje się, że rocznie sprzedaje się blisko 30 miliardów sztuk mikrokontrolerów [8] i jest to kilkanaście razy więcej niż procesorów przeznaczonych do „zwykłych" komputerów osobistych.

Pierwsze mikrokontrolery pojawiły się na rynku w połowie lat 70. XX w. Aż do końca lat 90. dominującą pozycję miały układy 8-bitowe, a układy 32-bitowe, przede wszystkim ze względu na wysokie ceny, były stosowane rzadko, tylko w najbardziej wymagających systemach. Ostatnie kilkanaście lat przyniosło jednak istotne zmiany. Sprzedaż układów 32-bitowych zaczęła dynamicznie rosnąć i obecnie stanowią one około 50% ogólnej liczby sprzedawanych mikrokontrolerów. Układy te zajęły więc czołową pozycję rynkową, a ich sprzedaż osiągnęła wartości (zarówno mierzone liczbą sztuk jak i obrotami) wyższe niż dotychczas dominujących układów 8-bitowych. Trend ten wynika przede wszystkim z rosnących potrzeb konsumentów i z wprowadzania przez producentów sprzętu elektronicznego coraz bardziej zaawansowanych urządzeń o coraz bogatszych funkcjach użytkowych. Również w zastosowaniach przemysłowych pojawiają się coraz bardziej zaawansowane systemy sterujące obiektami i procesami wymagające wydajnego przetwarzania sygnałów. Jednocześnie układy 32-bitowe pojawiają się także coraz szerzej w stosunkowo prostych systemach, gdzie zastępują układy 8- i 16-bitowe.

Znaczne zmiany zaszły także pod względem stosowanych architektur mikrokontrolerów. Dawniej każdy liczący się producent układów elektronicznych oferował procesory wykorzystujące własne konstrukcje rdzenia. Obecnie w ofercie większości wytwórców znaleźć można układy wyposażone w 32-bitowe rdzenie zaprojektowane przez firmę ARM. Niektórzy producenci w ogóle nie produkują procesorów 32-bitowych innych, niż z rdzeniem ARM.

Rosnąca popularność rodziny ARM i jej udział w rynku powoduje, że znajomość tej architektury jest niemal obowiązkowa wśród konstruktorów i programistów tworzących systemy wbudowane. Potwierdzeniem i zapowiedzią dalszego rozwoju trendów może być na przykład fakt, że już obecnie procesory z rdzeniami ARM mają ponad 1/3 udziału w globalnym rynku mikrokontrolerów, ponad 90% udziału w rynku mikrokontrolerów 32-bitowych i zdecydowanie dominujący udziału w rynku procesorów przeznaczonych dla telefonów komórkowych. Warto

więc poświęcić nieco czasu na zaznajomienie się z jedną z rodzin mikrokontrolerów wykorzystujących rdzeń ARM z aktualnie rozwijanej linii Cortex, jaką stanowią układy STM32 produkowane przez firmę STMicroelectronics.

Niniejsza książka przeznaczona jest przede wszystkim dla osób początkujących, które nie miały jeszcze styczności z mikrokontrolerami rodziny ARM, a zwłaszcza z układami z rdzeniem Cortex-M3. Zawiera ona 35 rozbudowanych ćwiczeń, które pozwalają poznać możliwości układów serii STM32F1xx. Ćwiczenia przygotowano w języku C z wykorzystaniem biblioteki STM32 Cube F1 HAL, przy czym zakłada się, że Czytelnik miał już styczność z językiem C i zna podstawowe pojęcia związane z programowaniem.

Podczas przygotowywania książki oraz kodu źródłowego poszczególnych ćwiczeń wykorzystano środowisko programistyczne *STM32 CubeIDE* w wersji 1.0.0 wraz ze zintegrowanym z nim programem *STM32 CubeMX* w wersji 5.2.0 i biblioteką *STM32 F1 HAL* w wersji 1.7.0.

Oprócz ćwiczeń, w książce zawarto także przydatne do ich wykonania opisy. Dotyczą one zastosowanych podukładów peryferyjnych takich jak m.in. porty wejścia/wyjścia, układy licznikowe, przetworniki A/C, interfejsy komunikacyjne oraz mechanizmów takich jak przerwania czy DMA. Ponadto, opisano i pokazano także sposób wykorzystania kilku ciekawych, zewnętrznych modułów rozszerzających (m.in. klawiatura, alfanumeryczne i graficzne wyświetlacze LCD, karty SD, akcelerometr, barometr, układ Bluetooth). Ta część może być interesująca także dla osób o nieco wyższym stopniu zaawansowania w programowaniu mikrokontrolerów.

Poszczególne ćwiczenia są pomyślane w ten sposób, że nakreślony jest zarys rozwiązania, przedstawione są jego kluczowe fragmenty oraz elementy nowo wprowadzane w stosunku do poprzednich ćwiczeń. Pozostała część większości ćwiczeń wymaga pewnego nakładu samodzielnej pracy, ułatwionej jednak licznymi podpowiedziami i wyjaśnieniami. Wykonanie wszystkich zadań z książki powinno zająć około 60…70 godzin. Sposób prowadzenia ćwiczeń pozwala zarówno na samodzielną naukę programowania jak i na wykorzystanie książki np. jako podstawy do prowadzenia kursów i zajęć laboratoryjnych w szkołach bądź na uczelniach.

Kod źródłowy ćwiczeń zawartych w książce przygotowano tylko na potrzeby edukacyjne i ma na celu możliwe proste i przystępne prezentowanie opisywanych mechanizmów i rozwiązań. Z tego powodu w wielu miejscach zastosowano pewne uproszczenia, np. pominięto lub zastosowano tylko podstawową kontrolę i obsługę potencjalnych błędów wykonania programu lub przypadków nietypowych.

1.1.1. Mikrokontrolery rodziny ARM

Rdzenie rodziny ARM są projektowane w firmie ARM Ltd. Jest to najbardziej znany i popularny wyrób tej firmy. Istotny jest fakt, iż firma ta nie zajmuje się produkcją, a jedynie opracowuje układy cyfrowe, w szczególności w postaci rozwiązań typu *IPcore* (*Intellectual Property Core*). Procesory wykorzystujące opracowane przez ARM rdzenie są natomiast produkowane przez większość najważniejszych producentów układów elektronicznych na świecie (m.in. NXP, Microchip, STMicroelectronics, Texas Instruments, Samsung). Warto zauważyć, że na końcowy produkt, czyli gotowy mikrokontroler, składa się nie tylko rdzeń, ale także wiele dodatkowych podukładów peryferyjnych, które dodawane są już przez poszczególnych producentów. Stąd bierze się bardzo duża różnorodność oferty mikrokontrolerów, mimo że podstawowym ich elementem są takie same rdzenie ARM.

Układy z rdzeniem ARM są, zależnie od wersji, 32- lub 64-bitowymi procesorami RISC. Cechują się stosunkowo prostą konstrukcją, w której wykorzystywana jest mniejsza (w porównaniu z rozwiązaniami konkurencyjnymi) liczba tranzystorów, co pozwala uzyskać niski pobór mocy i mały rozmiar procesora przy jednoczesnej dużej jego wydajności. Poszczególne wersje rdzeni różnią się przede wszystkim zbiorem obsługiwanych instrukcji i zastosowanymi technologiami, a co za tym idzie – możliwościami i wydajnością. Wyróżnić można wśród nich, m.in:

- podstawowy zestaw instrukcji ARM 32- lub 64-bitowych,
- instrukcje *Thumb* i *Thumb2* – specjalne zestawy instrukcji 16-bitowych,
- instrukcje NEON – instrukcje typu SIMD,
- instrukcje VFP – specjalne instrukcje do operacji na liczbach zmiennopozycyjnych,
- technologię *Jazelle* – sprzętową obsługę kodu bajtowego utworzonego w języku Java,
- instrukcje wspomagające przetwarzanie sygnałów (DSP),
- technologię *TrustZone* – wsparcie dla aplikacji wymagających wysokiego bezpieczeństwa (wykorzystywane np. w systemach bankowych),
- NVIC – wielopoziomowy kontroler przerwań.

Szukając informacji o procesorach ARM należy zwrócić uwagę na dwoistość stosowanych oznaczeń. Producenci procesorów najczęściej operują oznaczeniami wersji rdzenia (np. Cortex-M3, Cortex-A55 itp.), natomiast firma ARM często przywołuje oznaczenia wersji architektury rdzenia (np. ARMv7-M, ARMv8.2-A). Na **rysunku 1.1** przedstawiono zestawienie aktualnie oferowanych przez ARM wersji rdzenia.

Obecnie rozwijana rodzina Cortex składa się z czterech podrodzin:

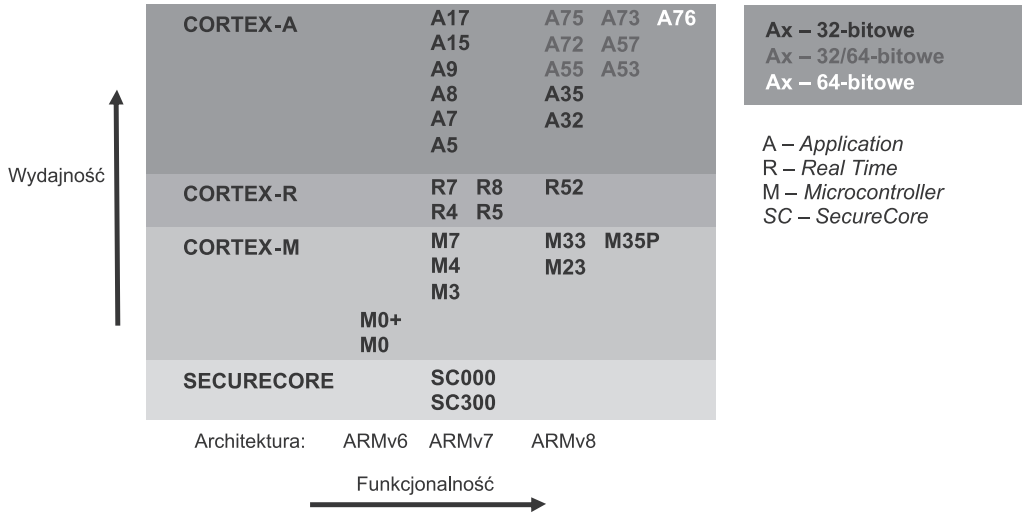
- **Cortex-Ax** – (*Application*) – rdzenie najbardziej zaawansowane, o najwyższej wydajności, przeznaczone do zastosowań, w których zwykle wykorzystuje się systemy operacyjne (np. Android, Linux, FreeRTOS, Windows Mobile); nowsze wersje tych rdzeni są układami o mieszanej architekturze 32- i 64-bitowej, a najnowsze – w pełni 64-bitowymi;

– **Cortex-Mx** – (*Microcontroller*) – rdzenie zaprojektowane z myślą o minimalizacji ceny przy zachowaniu dużej wydajności, przeznaczone zwłaszcza do zastosowań przemysłowych; cechuje je przede wszystkim brak obsługi zestawów A32 i A64, tj. podstawowych zestawów instrukcji ARM;

– **Cortex-Rx** – (*Real Time*) – rdzenie dedykowane dla systemów czasu rzeczywistego, w których krytycznym parametrem jest czas reakcji systemu, przeznaczone w szczególności do zastosowania w motoryzacji;

– **SCx** – (*Secure Core*) – rdzenie bazujące na Cortex-M dedykowane dla systemów wymagających podwyższonego bezpieczeństwa, takich jak np. karty chipowe, karty SIM, dokumenty biometryczne.

W przedstawionych wyżej oznaczeniach liczba w miejscu litery *x* oznacza wersję rdzenia.



Rys. 1.1. Aktualnie oferowane rdzenie ARM

Ponieważ niniejsza książka dotyczy układów z rdzeniem Cotex-M3 warto przyjrzeć się bliżej tej właśnie podrodzinie. Wśród rdzeni Cortex-M oferowane są aktualnie ich następujące wersje [3]:

– **Cortex-M0** – najmniejszy, najprostszy i najtańszy rdzeń o najniższej wydajności wykorzystujący architekturę ARMv6-M; charakteryzuje się bardzo niskim poborem energii (zwłaszcza w wersji M0+); przewidziany do zastosowań, gdzie istotna jest oszczędność energii, a także pomyślany jako bezpośrednia konkurencja dla mikrokontrolerów 8- i 16-bitowych; wydajność 2,33 CoreMark/MHz (M0) lub 2,46 (M0+) CoreMark/MHz,

– **Cortex-M1** – rdzeń przewidziany do implementacji w układach FPGA wykorzystujący architekturę ARMv6-M,

- **Cortex-M3** – rdzeń przewidziany do typowych zastosowań w systemach wbudowanych; wykorzystuje architekturę ARMv7-M; wydajność 3,34 CoreMark/MHz,
- **Cortex-M4** – rdzeń przewidziany do zastosowań w systemach wbudowanych, gdzie wymagane jest intensywniejsze przetwarzanie sygnałów (m.in. obsługuje dodatkowe instrukcje typowe dla zastosowań DSP); wykorzystuje architekturę ARMv7-M; w wersji Cortex-M4F wyposażony w jednostkę FPU; wydajność 3,42 CoreMark/MHz,
- **Cortex-M7** – rdzeń przewidziany do zastosowań w systemach wbudowanych wymagających wyższej wydajności; wydajność 5,01 CoreMark/MHz,
- **Cortex-M23** – nowocześniejszy rdzeń odpowiadający zakresem zastosowań rdzeniowi M3; wyposażony w technologie podnoszące bezpieczeństwo systemu i samego procesora; wykorzystuje architekturę ARMv8-M; wydajność 2,50 CoreMark/MHz,
- **Cortex-M33** – nowocześniejszy rdzeń odpowiadający zakresem zastosowań rdzeniowi M4; wyposażony w technologie podnoszące bezpieczeństwo systemu i samego procesora; wykorzystuje architekturę ARMv8-M; wydajność 3,86 CoreMark/MHz,
- **Cortex-M35P** – rdzeń oparty na M33 z dodanymi technologiami znacznie podnoszącymi bezpieczeństwo systemu.

Porównując wydajność poszczególnych rdzeni należy zwrócić uwagę, że podane wartości wskaźnika CoreMark odniesione są do jednego MHz częstotliwości taktowania rdzenia procesora. Oznacza to, że najwydajniejszy rdzeń M7, o wskaźniku 5,01 CM/MHz, przy skonfigurowaniu go do pracy z maksymalną częstotliwością (np. 400 MHz) jest w rzeczywistości ponad 25 razy wydajniejszy niż rdzeń M0, o wskaźniku 2,33 CM/MHz, również pracujący z najwyższą dostępną dla niego częstotliwością (np. 32 MHz).

1.2. Architektura rdzenia ARM Cortex-M3

1.2.1. Najważniejsze cechy architektury Cortex-M3

Rdzeń Cortex-M3 wykorzystuje architekturę ARMv7-M [1]. Pod względem organizacji pamięci jest to architektura harwardzka, tzn. pamięć zawierająca kod programu (Flash) i pamięć danych (SRAM) są rozdzielone i dostęp do nich odbywa się poprzez osobne magistrale. Niezależność obszarów pamięci daje m.in. możliwość równoległego wykonywania operacji na obu blokach pamięci, co pozwala zwiększyć wydajność systemu. W przypadku procesorów ARM istnieje jednak możliwość umieszczenia kodu programu w obszarze SRAM. Wadą takiego rozwiązania jest to, że procesor będzie wykonywał program nico wolniej gdyż jest on zoptymalizowany w taki sposób, by pobierał go z pamięci Flash, a nie SRAM. Należy też pamiętać, że po wyłączeniu zasilania, zawartość SRAM jest tracona. Ponadto pojemność pamięci SRAM jest kilkukrotnie mniejsza niż pamięci Flash, można więc w niej zmieścić tylko stosunkowo nieduże programy lub tylko ich fragmenty.

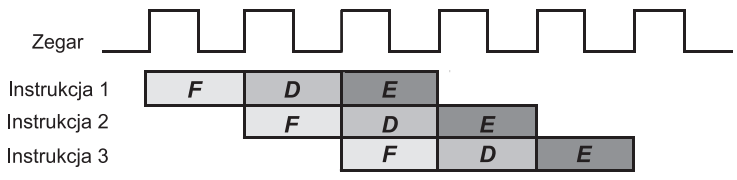
Jednakże, podczas dostępu do pamięci Flash występują opóźnienia (tzw. cykle oczekiwania) natomiast dostęp do SRAM jest natychmiastowy, co może być bardzo

ważne w aplikacjach wymagających determinizmu czasowego, np. rozwiązaniach *Real Time*. Wspomniane opóźnienie zależy od wybranej częstotliwości sygnału taktującego rdzeń procesora. Jeśli jest ona mniejsza niż 24 MHz – opóźnienie wynosi 0 cykli, jeśli jest pomiędzy 24 a 48 MHz – 1 cykl, a przy częstotliwości ponad 48 MHz – opóźnienie wynosi 2 cykle. Występowanie opóźnień wynika stąd, że maksymalna częstotliwość komunikacji z pamięcią Flash wynosi tylko 24 MHz. Dodatkową zaletą możliwości umieszczenia programu w pamięci danych jest także wydłużenie czasu życia pamięci Flash. Pamięć ta ma ograniczoną żywotność, która zwykle wynosi kilkanaście do kilkudziesięciu tysięcy cykli zapisu. Jeżeli program jest wielokrotnie zmieniany (np. w czasie jego testowania), można zaoszczędzić dość znaczną liczbę tych cykli.

Jak już wspomniano, procesory ARM są procesorami RISC. Architekturę tę nazywa się niekiedy architekturą *Load/Store*. Procesor nie wykonuje operacji wprost na komórkach pamięci i nie ma też możliwości przesłania danych bezpośrednio pomiędzy komórkami pamięci. Dane muszą zostać najpierw załadowane (*load*) do rejestrów procesora, a po operacji odesłane (*store*) do pamięci – stąd nazwa architektury. Pozornie może się wydawać, że generuje to więcej operacji, ale dzięki temu lista instrukcji procesora nie musi zawierać wielu złożonych, a jednocześnie rzadko wykorzystywanych poleceń. To z kolei pozwala na uproszczenie układu dekodera instrukcji co przekłada się na ich szybsze dekodowanie i wykonanie.

Lista instrukcji procesorów Cortex-M3 zawiera około 130 instrukcji z zestawów *Thumb* i *Thumb2*. Instrukcje te operują zarówno na danych 16-, jak i 32-bitowych, ale same kody instrukcji są 16-bitowe. Można dzięki temu zaoszczędzić pamięć, ponieważ kod programu jest w niej lepiej „upakowany". W skrajnych przypadkach można uzyskać nawet 40% mniejszą zajętość pamięci niż dla takiego samego programu, ale zapisanego z użyciem tylko instrukcji 32-bitowych. Dodatkowo, krótsze kody instrukcji przyczyniają się do ogólnej poprawy wydajności procesora. Warto zauważyć, że rdzenie Cortex-M nie obsługują „normalnych", podstawowych list instrukcji ARM 32-bitowych (A32), 64-bitowych (A64) ani ich rozszerzeń (np. NEON czy *Jazelle*).

Kolejną cechą wyróżniającą mikrokontrolery z rdzeniem Cortex-M, która zwiększa ich wydajność, jest wykorzystanie mechanizmu potokowego przetwarzania instrukcji. Polega on na tym, że każda instrukcja jest dzielona na 3 etapy: pobranie instrukcji (*Fetch*), jej zdekodowanie (*Decode*) i wykonanie (*Execute*). Gdy jedna instrukcja jest w fazie wykonania, następna może być już dekodowana, a jeszcze następna – pobierana. Przedstawiono to obrazowo na **rysunku 1.2**. Dodatkowo, procesor potrafi wykonywać instrukcje w sposób spekulatywny, tzn. zawiera on wbudowany układ przewidywania rozgałęzień i skoków w programie, dzięki któremu może wykonać „na zapas" instrukcje znajdujące się za rozgałęzieniem.



Rys. 1.2. Przetwarzanie potokowe

Jeszcze jedną przydatną i zwiększającą wydajność cechą rdzeni Cortex jest sprzętowe wykonywanie operacji dzielenia. Operacja ta jest stosunkowo złożona i dlatego wiele prostszych i tańszych układów mikroprocesorowych nie ma jej zaimplementowanej. Wykonanie dzielenia w takiej sytuacji wymaga napisania dość rozbudowanego algorytmu. W praktyce zwykle odpowiednie polecenia generowane są przez kompilator kodu źródłowego. Umieszczenie w procesorach ARM Cortex jednostki dzielącej zdecydowanie upraszcza wykonanie operacji dzielenia. Należy jednak pamiętać, że jest to tylko dzielnie liczb stałopozycyjnych, ponieważ procesor nie obsługuje zmiennopozycyjnych typów danych.

1.2.2. Rejestry i organizacja pamięci

Podczas wykonywania obliczeń procesor operuje na danych umieszczonych w rejestrach. Rdzeń Cortex-M3 jest wyposażony w szesnaście 32-bitowych rejestrów podstawowych oznaczonych od *R0* do *R15*. Rejestry od *R0* do *R12* są rejestrami ogólnego przeznaczenia (*General Purpose Register*). Spośród nich rejestry *R0...R7* dostępne są dla wszystkich instrukcji, pozostałe zaś, tylko dla niektórych. Kolejny rejestr, *R13*, jest tzw. wskaźnikiem stosu. W rzeczywistości są to dwa rejestry, z których w danym momencie widoczny jest tylko jeden, tj. *MSP* lub *PSP*. *MSP* (*Main Stack Pointer*) jest rejestrem domyślnym, używanym przez przerwania i jądro systemu operacyjnego. Z kolei *PSP* (*Process Stack Pointer*) jest używany przez program użytkownika, jeśli na mikrokontrolerze działa system operacyjny, a program użytkownika jest uruchamiany za jego pośrednictwem. Dzięki zastosowaniu dwóch rejestrów stosu możliwe jest tworzenie bezpieczniejszych systemów, ponieważ program użytkownika nie ma dostępu do stosu systemu operacyjnego, a więc nie może go uszkodzić w przypadku wystąpienia jakiegoś błędu. Rejestr *R14* (*Link Register*) zawiera tzw. adres powrotu. Dzięki niemu, procesor wie, w które miejsce wykonywanego programu należy powrócić po zakończeniu wykonywania podprocedury lub procedury obsługi przerwania. Ostatni z rejestrów podstawowych, *R15*, to licznik rozkazów (*Program Counter*) i zawiera adres aktualnie wykonywanej instrukcji.

Oprócz rejestrów podstawowych, rdzeń zawiera wiele rejestrów specjalnych, które służą do sterowania wykonaniem programu oraz pracą procesora. Wśród nich można np. wyróżnić grupę rejestrów *xPSR* (*Program Status Register*), odpowiedzialnych m.in. za przechowywanie wyniku operacji porównania i przeniesienia czy wykorzystywanych do obsługi instrukcji warunkowych i przerwań. Inną grupę rejestrów stanowią rejestry związane ze śledzeniem wykonywania programu, także w trybie pracy krokowej, które wykorzystywane są podczas debugowania.

Wszystkie rdzenie ARM Cortex-M mają tak samo zorganizowaną przestrzeń adresową. Łącznie może ona obejmować 4 GB pamięci, jest jednak podzielona na segmenty o z góry zdefiniowanym przeznaczeniu. Najważniejsze obszary to:

Code – obszar kodu programu (pamięć *Flash*),

SRAM – pamięć operacyjna,

Peripheral – porty i urządzenia peryferyjne,

External RAM – pamięć zewnętrzna,

External device – urządzenia zewnętrzne.

Rys. 1.3. Mapa pamięci rdzenia Cortex-M3

Podział i zakres adresów obejmujący poszczególne obszary pamięci (tzw. mapę pamięci) pokazano na **rysunku 1.3**. Możliwe jest inne skonfigurowanie przeznaczenia poszczególnych obszarów (np. umieszczenie danych w przestrzeni adresowej przeznaczonej dla kodu programu), ale są to rozwiązania stosowane przede wszystkim w przypadkach nietypowych i specjalnych.

Podczas przeglądania mapy pamięci, uwagę zwracają obszary *bit-band* i ich aliasy [2, 44]. Obszary te nazywane są także obszarami o dostępie atomowym lub bitowym. Zazwyczaj pamięć w systemach cyfrowych jest podzielona na komórki o pojemności 8 bitów. Każda z komórek ma określony adres, dzięki któremu można jednoznacznie określić, do której z nich chcemy się odwołać. Taka organizacja nie pozwala jednak uzyskać dostępu bezpośrednio do pojedynczych bitów słowa danych. By móc zmodyfikować wartość pojedynczego bitu, należy odczytać zawartość całej komórki pamięci, zmienić wartość interesującego nas bitu i ponownie

zapisać w pamięci całe słowo. Rozwiązaniem ułatwiającym operowanie na pojedynczych bitach są obszary *bit-band*, w których każdy bit ma swój własny adres. Jest to szczególnie ważne i przydatne w systemach zbudowanych z wykorzystaniem mikrokontrolerów, gdyż tego typu operacje są tam bardzo częste. Oprócz dostępu bitowego, do obszarów *bit-band* można się oczywiście odwołać także według normalnych zasad adresowania.

Wyjaśnienia wymaga jeszcze sposób adresowania bitów w obszarze o dostępie atomowym. W tym celu stosuje się następujący wzór:

adres_bitu = *poczatek_obszaru_bitband* + *przesuniecie_bajtu*×32 + *numer_bitu*× 4,

gdzie:

adres_bitu – adres, pod jakim należy zapisywać i spod którego można odczytywać stan bitu,

poczatek_obszaru_bitband – 0x20000000 dla obszaru *bit-band* w pamięci SRAM oraz 0x40000000 dla obszaru *bit-band* w segmencie urządzeń peryferyjnych,

przesuniecie_bajtu – przesuniecie (*offset*) w stosunku do początku regionu *bit-band*,

numer_bitu – pozycja bitu w słowie, do którego chcemy uzyskać dostęp.

Na przykład, jeśli chcemy ustawić bit 6 komórki o adresie 0x2000001 (*offset* = 1), to zgodnie z przedstawionym wzorem, powinniśmy dokonać zapisu nowej wartości bitu pod adresem 0x22000038:

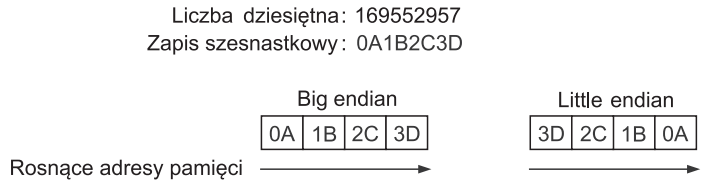
adres_bitu = 0x22000000 + 0x01 × 0x20 + 0x06 × 0x04 = 0x22000038.

W zrozumieniu powyższych zależności pomocny może być także **rysunek 1.4**.

Rys. 1.4. Sposób mapowania adresów z obszaru *bit-band*

Przeglądając mapę pamięci można zauważyć, że zawiera on nie tylko adresy typowych obszarów pamięci, ale również obszar adresów przeznaczony dla urządzeń peryferyjnych. W obszarze tym znajdują się rejestry danych i rejestry sterujące poszczególnych układów peryferyjnych mikrokontrolera. Na przykład w zakresie adresów od 0x40010800 do 0x40010BFF znajdują się rejestry związane z portem GPIOA, a w obszarze od 0x40005400 do 0x400057FF – rejestry układu pierwszego kontrolera interfejsu I^2C. Dzięki takiej organizacji pamięci, dostęp do tych rejestrów jest, z punktu widzenia programisty, taki sam jak dostęp do dowolnej komórki pamięci. Jest to więc bardzo wygodne rozwiązanie.

Mikrokontrolery ARM są układami 32-bitowymi. Oznacza to, że podstawową długością słowa, na jakiej operują, jest słowo 4-bajtowe. Potrafią one, oczywiście, operować również na słowach 1- i 2-bajtowych. Ponieważ pamięć jest podzielona na bloki 1-bajtowe, w przypadku, gdy słowo danych zajmuje więcej niż 1 bajt, powstaje problem, w jakiej kolejności zapisane są w pamięci kolejne jego bajty. Standardowym sposobem zapisu w procesorach ARM jest tzw. konwencja *little endian*, w której najmniej znaczący (najmłodszy) bajt jest zapisywany jako pierwszy, czyli pod najniższym adresem zajmowanego bloku pamięci. Ciekawą cechą procesorów ARM jest możliwość używania także konwencji *big endian*, w której jako pierwszy jest zapisywany bajt najbardziej znaczący. Różnice pomiędzy tymi sposobami zapisu pokazano także na **rysunku 1.5**. Wybór konwencji następuje w momencie startu lub zerowania procesora na podstawie wartości logicznej napięcia na wyprowadzeniu BIGEND. Zaletą stosowania konwencji *big endian* jest możliwość ułatwienia pisania i przyśpieszenia aplikacji, które służą np. do komunikacji przez sieć Ethernet, gdzie stosowany jest właśnie zapis *big endian*. Unika się wówczas konieczności nieustannych konwersji między oboma sposobami zapisu.



Rys. 1.5. Sposób zapisu danych w konwencjach *big endian* i *little endian*

1.2.3. Podstawowe elementy rdzenia

Oprócz jednostki arytmetyczno-logicznej będącej zasadniczym i najważniejszym elementem rdzenia, w jego skład wchodzą:

- interfejs pamięci,
- układ zarządzania zasilaniem i poborem energii,
- układy generowania sygnałów zegarowych dla poszczególnych podzespołów procesora,
- układ zarządzający magistralami systemowymi (*Bus Matrix*),
- kontroler przerwań (NVIC),

– układy śledzenia wykonania (debugowania) programu i pracy krokowej,
– opcjonalny układ ochrony pamięci,
– interfejs JTAG/SW, pozwalający m.in. programować procesor w docelowym systemie oraz śledzić działanie programu w trybie debugowania.

Inne układy peryferyjne, jakie można znaleźć w różnych mikrokontrolerach poszczególnych wytwórców, nie należą do rdzenia i nie są dziełem firmy ARM. Są one natomiast dodawane do głównego rdzenia przez producentów mikrokontrolerów.

1.3. Mikrokontrolery STM32F10x

STMicroelectronics jest jednym z największych na świecie producentów układów elektronicznych. Posiada fabryki i centra badawcze w Europie, Azji i Afryce. Jak przystało na tak znaczącego uczestnika rynku, ma w swojej ofercie mikrokontrolery 8- i 32-bitowe. Wśród nich najbogatszą grupę stanowią układy 32-bitowe wykorzystujące rdzenie ARM Cortex, których obecnie w ofercie jest ponad 800, w różnych wersjach. Układy te noszą oznaczenie STM32Fxxxxx, G, L i H gdzie xxxxx to oznaczenie konkretnego modelu z rodziny STM32. Litera F lub G oznacza układy podstawowej serii mikrokontrolerów (*Mainstream*), L – układy o obniżonym poborze energii (*ultra Low power*), a H – układy wysokiej wydajności (*High performance*). W ramach każdej z serii wydzielone są podgrupy oznaczane cyframi od 0 do 7. Poszczególne cyfry oznaczają wersję rdzenia Cortex użytą w danym mikrokontrolerze. Im wyższa cyfra, tym większa ogólna wydajność układu. Zestawienie dostępnych serii układów przedstawiono na **rysunku 1.6**. Na rysunku tym podano także maksymalne, możliwe do uzyskania częstotliwości sygnałów taktujących rdzenie poszczególnych mikrokontrolerów oraz wyniki testów CoreMark obrazujące ich wydajność.

W poszczególnych seriach oferowanych jest wiele różnych modeli mikrokontrolerów, np. na serię STM32F1 składają się układy F100, F101, F102, F103, F105 i F107. Z kolei każdy z układów oferowany jest w kilku różnych odmianach różniących się np. pojemnością dostępnej pamięci, wyposażeniem w układy peryferyjne i ich liczbą, rodzajem obudowy itp. Same tylko układy STM32F103 dostępne są w 30 różnych odmianach. Zestawienie najważniejszych, wybranych różnic między układami serii F1 przedstawiono w **tabeli 1.1**.

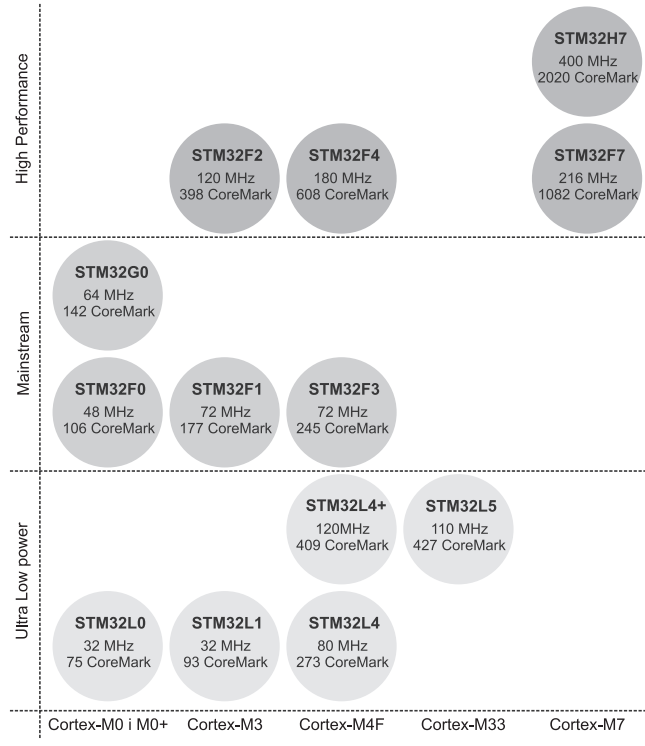
Tab. 1.1. Zestawienie wybranych różnic pomiędzy układami serii STM32F1

Seria STM32	F-CPU [Hz]	Flash [KB]	SRAM [KB]	TIM	USB	CAN	FSMC	SDIO	Eth
F100	24	16…512	4…32	6…11			X		
F101	36	16…1024	4…80	2…12			X		
F102	48	16…128	4…16	2…3	X				
F103	72	16…1024	4…96	3…14	X	X	X	X	
F105 i F107	72	64…256	64	7	X	X	X		X

gdzie:
F-CPU – maksymalna częstotliwość taktowania rdzenia mikrokontrolera,
Flash – rozmiar dostępnej pamięci Flash,
SRAM – rozmiar dostępnej pamięci operacyjnej SRAM,
TIM – liczba uniwersalnych układów licznikowych,
USB – obsługa interfejsu USB 2.0,
CAN – obsługa interfejsu CAN,
FSMC – (*Flexible Static Memory Controller*) kontroler zewnętrznych pamięci statycznych,
SD – obsługa interfejsu SDIO,
Eth – obsługa interfejsu Ethernet.

Ze względu na możliwości i wyposażenie układy STM32 pogrupowane są dodatkowo w pięciu liniach produktów:

– linia *Performance* to podstawowa linia układów,
– linia *Connectivity* to linia układów wyposażonych dodatkowo w interfejs Ethernet,
– linia *Access* to linia układów tańszych, ale zubożonych w porównaniu z linią podstawową przede wszystkim o układy interfejsów komunikacyjnych,
– linia *USB Access* to linia układów podobnych to tych z linii *Access*, ale z dodaną obsługą interfejsu USB,
– linia *Value* to linia układów najtańszych i najbardziej okrojonych pod względem wyposażenia.



Rys. 1.6. Zestawienie serii mikrokontrolerów oferowanych przez STMicroelectroncs

Dodatkowym kryterium podziału układów jest rozmiar pamięci Flash. Firma STM używa w dokumentacjach terminu „gęstość" (*density*) na określenie pojemności pamięci zawartej w układzie. Bierze się to stąd, że im więcej pamięci ma zawierać mikrokontroler, tym bardziej musi być ona upakowana w chipie, a więc musi mieć większą gęstość.

Ostatnim kryterium podziału jest rodzaj obudowy, w jakiej jest umieszczony mikrokontroler. Najmniejsze i najuboższe pod względem wyposażenia układy można znaleźć w małych obudowach o 36 wyprowadzeniach. Z kolei najbogatsze w wyposażenie mikrokontrolery, z wieloma układami peryferyjnymi, oferowane są w obudowach ze 100, a nawet 144 wyprowadzeniami.

Duża różnorodność oferowanych wersji pozwala konstruktorom wybrać mikrokontroler najlepiej dostosowany do ich wymagań pod względem wyposażenia w pamięć i układy peryferyjne, rozmiarów fizycznych obudowy i ceny. Dobór odpowiedniego, najlepiej dopasowanego do projektowanego systemu, a jednocześnie jak najtańszego układu jest często sporym wyzwaniem. Ponieważ jednak wszystkie układy wykorzystują rdzeń ARM Cortex, zachowana jest duża kompatybilność programowa i pełna kompatybilność fizyczna mikrokontrolerów montowanych w obudowach tego samego typu. Dzięki temu łatwo jest przenieść program napisany dla jednej wersji mikrokontrolera na inną.

W rozdziale 1 wykorzystano następujące materiały źródłowe: [1, 2, 3, 12, 43, 44, 45, 51, 54].