Interfejs SmartCard w mikrokontrolerach STM32

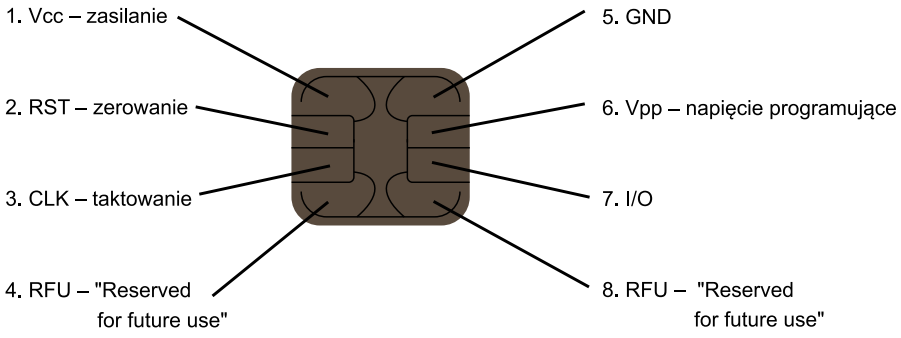
W świecie elektroniki istnieje wiele sposobów identyfikacji użytkownika. Obecnie bodaj najbardziej rozpowszechnionym w codziennym życiu (szczególnie w bankowości) jest standard wykorzystujący karty Smartcard. W artykule podajemy informacje, jak wykorzystać karty Smartcard w aplikacjach z mikrokontrolerami STM32.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 12040, pass: 15735862
- listingi

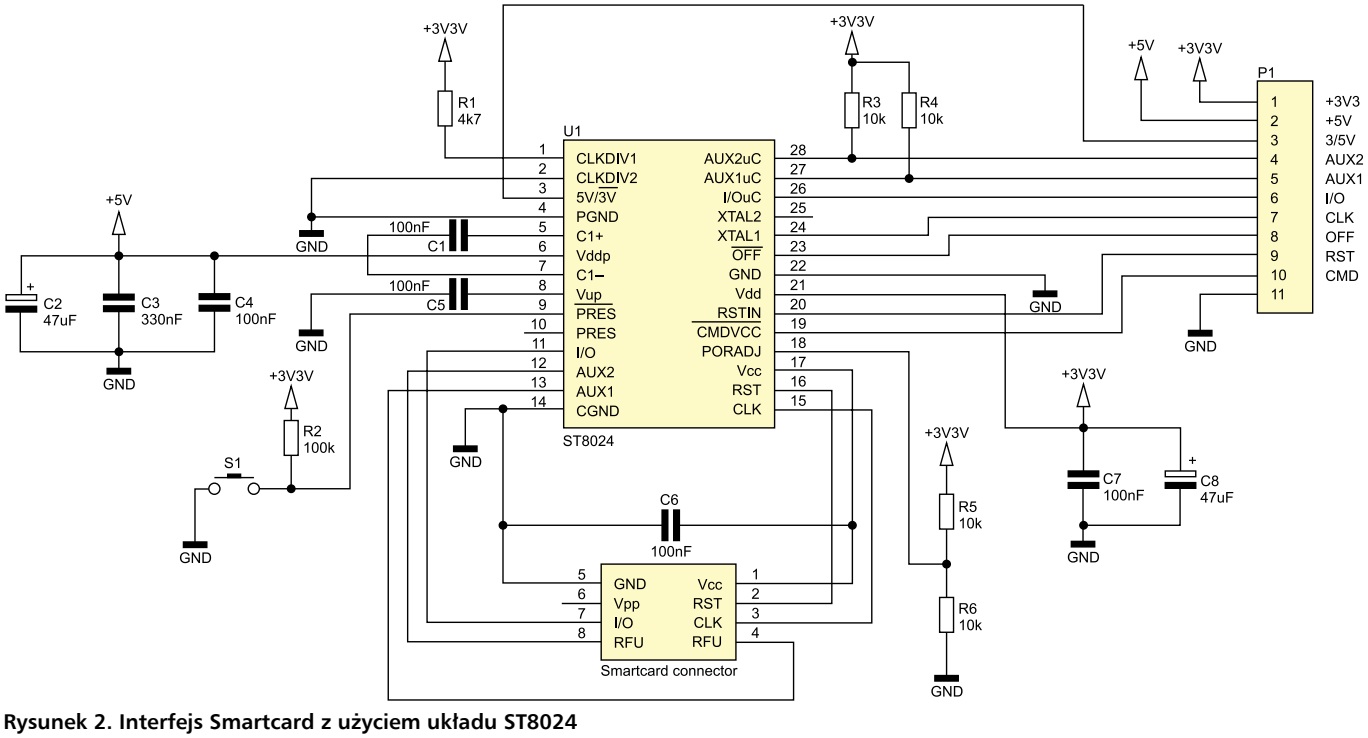
Elektroniczne karty płatnicze zupełnie zrewolucjonizowały codzienne życie ludzi. Dzięki nim nie ma potrzeby obracania gotówką, wystarczy jedynie kawałek plastiku z zaszytą weń elektroniką i już można dokonywać transakcji na całym świecie. Zastosowania kart identyfikacyjnych nie ogranicza się rzecz jasna tylko do dokonywania płatności. Współczesne karty znalazły zastosowanie w wielu obszarach, które wymagają zabezpieczeń lub autoryzacji, jak choćby karty umożliwiające odbiór płatnych pakietów telewizyjnych.

Pierwotnie autoryzacja kart elektronicznych odbywała dzięki informacjom zapisanym na pasku magnetycznym umieszczonym na karcie. Z czasem i postępującym rozwojem mikroelektroniki stało się możliwym technologicznie i, co niemniej istotne – opłacalnym, wmontowywanie w karty mikroprocesora. W ten sposób powstały wszechobecne dziś mikroprocesorowe karty typu Smartcard.

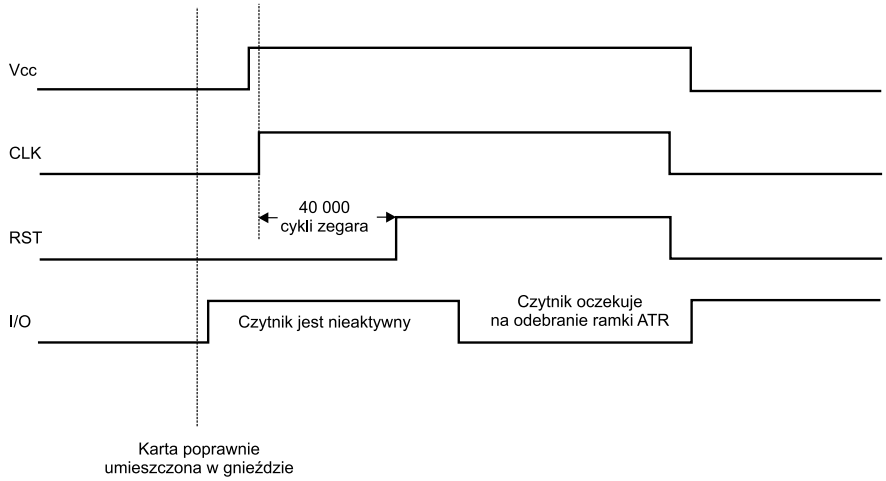
Mikrokontrolery STM32 mają wsparcie sprzętowe dla aplikacji pracujących z kartami identyfikacyjnymi. Wbudowany w te układy uniwersalny interfejs szeregowy USART może pełnić rolę kontrolera Smartcard. W programie przedstawionym pod koniec niniejszego artykułu wykorzystamy tą alternatywną funkcjonalność USART.



Rysunek 1. Styki karty identyfikacyjnej Smartcard



Rysunek 2. Interfejs Smartcard z użyciem układu ST8024



Rysunek 3. Sekwencja sygnałów na liniach interfejsu po umieszczeniu karty w gnieździe

Standard Smartcard (ISO 7816)

Wszystko, co jest związane z kartami identyfikacyjnymi Smartcard, zostało zdefiniowane w standardzie ISO 7816. Normę podzielono na części, a kolejne dodawano wraz z upływem czasu i powstającymi nowymi wymaganiami. Dotychczas powstało kilkanaście części standardu, lecz aby odczytywać i zapisywać dane do typowej karty identyfikacyjnej wystarczy tylko pięć pierwszych części:

- ISO 7816-1 – konstrukcja mechaniczna, parametry elektryczne oraz wytrzymałość na czynniki zewnętrzne;
- ISO 7816-2 – liczba, funkcje, położenie oraz wymiary pól kontaktowych;
- ISO 7816-3 – sygnały elektryczne oraz protokoły transmisji – warstwa łącza danych;
- ISO 7816-4 – specyfikuje komendy i metody dostępu;
- ISO 7816-5 – systemy numeracji oraz procedury nadawania identyfikatorów dla producentów.

Wygląd typowych styków karty identyfikacyjnej z mikroprocesorem zamieszczono na **rysunku 1**.

Podłączenie karty identyfikacyjnej do mikrokontrolera z rodziny STM32 wymaga zastosowania układu interfejsu analogowego. W tej roli dobrze spisuje się układ ST8024, a jego standardową aplikację przedstawiono na **rysunku 2**. Warto zauważyć, iż istnieje tylko jedna linia przeznaczona do wymiany danych, co sprawia, że możliwa jest tylko komunikacja w trybie half-dupleks.

Układ interfejsu zapewnia poprawne formowanie sygnałów oraz m. in. zabezpieczenie przed skutkami wyładowań elektrostatycznych. Ponadto ST8024 monitoruje stan mikroprzełącznika sygnalizującego umieszczenie karty w gnieździe. Na zamieszczonym schemacie zaznaczono, do których wyprowadzeń mikrokontrolera STM32 można podłączyć układ interfejsu. Taka konfiguracja jest stosowana do zrealizowania aplikacji przedstawionej w dalszej części artykułu.

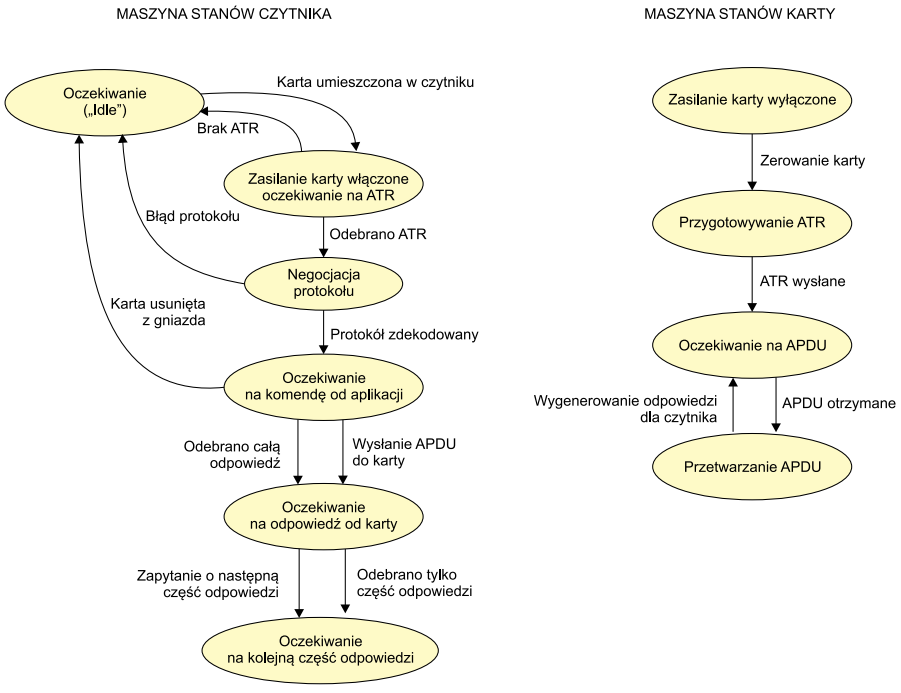
Charakterystyka gniazda Smartcard powoduje, że podczas umieszczania karty w gnieździe kontakty znajdujące się na karcie mogą w trakcie ruchu uzyskać połączenie elektryczne z niewłaściwymi kontaktami gniazda. Może to oczywiście grozić uszkodzeniem karty lub czytnika, dlatego zasilanie powinno być doprowadzone do karty dopiero po tym, jak zostanie ona poprawnie umieszczona w gnieździe. Oznacza to, że w stanie jałowym, to jest wtedy, kiedy gniazdo jest puste, oraz podczas wkładania karty na złączach gniazda nie może występować napięcie. W przeciwnym wypadku mikroprocesor umieszczony w karcie może ulec uszkodzeniu. Zachowanie się czytnika po umieszczeniu w nim karty definiuje trzecia część standardu: ISO 7816-3.

Wygląd przebiegów sygnałów inicjalizujących interfejs Smartcard zamieszczono wraz z komentarzem na **rysunku 3**. Gdy karta jest poprawnie umieszczona w czytniku to następuje włączenie napięcia zasilania oraz sygnału zegarowego. Taki stan musi trwać przynajmniej 40000 cykli zegarowych. Po ich upływie linia zerująca RST powinna zostać przełączona w stan wysoki. Od tej chwili karta identyfikacyjna ma 400 cykli na wygenerowanie tzw. odpowiedzi na sygnał zerowania ATR (*Answer to Reset*).

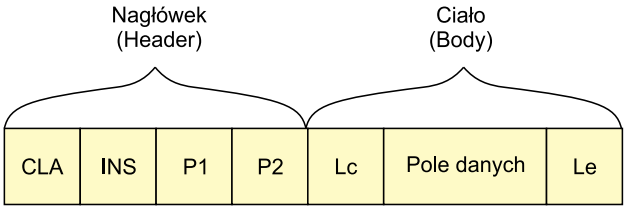
Cały proces inicjalizacji oraz wymiany danych opiera się o działanie dwóch maszyn stanów: jednej po stronie czytnika i drugiej po stronie karty identyfikacyjnej. Grafy stanów i przejść dla obydwu maszyn stanów przedstawia **rysunek 4**. Do konkretnego protokołu komunikacyjnego wrócimy jeszcze, teraz należy zrozumieć, czym jest tajemnicza odpowiedź na sygnał zerowania ATR oraz w jaki sposób przesyłane bajty są kodowane.

Standard nie określa, czy interpretacja logiczna potencjału linii danych ma być prosta, czy odwrotna. Oznacza to, że poziom wysoki na linii I/O może być zinterpretowany jako logiczne 0 lub logiczna 1. To, w jaki sposób przebiega interpretacja zależy od karty, a informację na ten temat zawiera w sobie pierwszy z bajtów sekwencji *Answer to Reset* – bajt TS. Różne możliwości interpretacji poziomów logicznych są też ściśle związane z kolejnością pojawiania się bitów na linii danych.

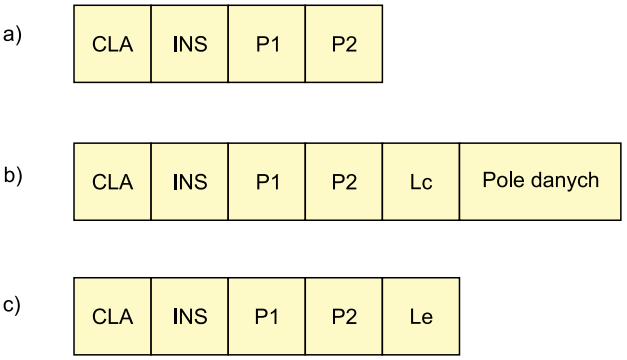
Jeśli jest wybrane kodowanie proste, czyli logiczna 1 odpowiada linii I/O w stanie wysokim, to wtedy pierwszy wysyłany będzie bit najmniej znaczący (LSB). W interpretacji odwrotnej, czyli takiej, w której potencjał wysoki będzie oznaczał logiczne 0, na linię danych wystawiany jest jako pierwszy bit MSB.



Rysunek 4. Maszyny stanów czytnika i karty identyfikacyjnej



Rysunek 5. Budowa pakietu komendy ADPU

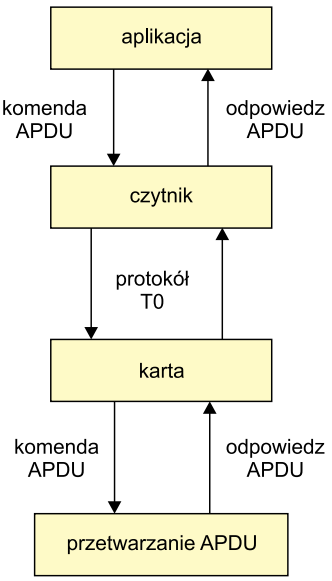


Rysunek 6. Różne postacie ciała komendy ADPU

Każdy bajt jest rozpoczynany przez znacznik START, a kończy się bitem parzystości, pozwalającym zweryfikować, czy odebrany bajt nie został przekłamany podczas transmisji. Zawartość odebranego bajta jest uznawana za poprawną wtedy, gdy wraz z bitem parzystości ilość logicznych jedynek będzie parzysta.

Ramka odpowiedzi na sygnał zerowania ATR może być zbudowana maksymalnie z 33 bajtów podzielonych na pięć grup:

– Znak inicjalizujący TS. Jak wyżej wspomniano, niesie ze sobą informację o sposobie kodowania bitów oraz kolejności ich wysyłania.
– Znacznik T0. Informuje, czy ramka *Answer to Reset* zawiera opcjonalne znaki, będące zaszłością historyczną.
– Bajty niosące informację o szczegółach implementacji interfejsu.



Rysunek 7. Elementy łańcucha komunikacji z kartą Smartcard

– Bajty o znaczeniu historycznym, mogą zawierać informacje dodatkowe, specyficzne dla danej aplikacji. Ich format nie jest definiowany w żaden sposób przez standard.
– Bajt kontrolny.

Protokół komunikacyjny

Dane przesyłane podczas komunikacji z kartą identyfikacyjną są dzielone w standardzie ISO 7816-4 na pakiety ADPU (*Application Protocol Data Units*). Pakiety ADPU podzielone są na dwie grupy: pakiety komend wysyłane do karty, oraz pakiety odpowiedzi wysyłane przez kartę do czytnika.

Pakiety komend wysyłane do karty składają się z nagłówka oraz pozostałej części nazwanej ciałem (*body*), patrz **rysunek 5**. Popularnym, a więc i często wykorzystywanym, jest protokół 'T0'. W tym protokole jednostki informacji nazwane zostały TDPU (*Transmission Protocol Data Units*). Pakiety ADPU są w tym przypadku identyczne w budowie jak TDPU, dzięki czemu warstwy nie są dodatkowo komplikowane i możemy omówić tylko pakiety ADPU, ponieważ w przykładowa aplikacja będzie operować właśnie na nich.

Nagłówek komendy ADPU zawiera cztery pola, które pozwalają określić między innymi, jaka komenda ma zostać wykonana. Poszczególne bajty nagłówka to:

– *CLA* – Bajt określający klasę komendy. Przykładowo wartość szesnastkowa 0y, gdzie y jest dowolne z przedziału 0 do F, oznacza, że przesyłana komenda będzie dotyczyć operacji na plikach lub będzie wykorzystywać system zabezpieczeń.
– *INS* – Klasyfikuje, jaka dokładnie komenda jest wysyłana. Jeśli klasą komendy są operacje na plikach, wówczas wartości bajta *INS* oznaczają akcję, jaka ma być podjęta w systemie plików
– *P1*, *P2* – Zawierają dodatkowe informacje potrzebne do wykonania instrukcji określonej przez pola *CLA* oraz *INS*. Jeśliby przytoczyć tutaj konkretny przykład w postaci funkcji wykonujących operacje na systemie plików karty, to pola *P1* oraz *P2* mogą definiować, w jaki sposób ma być uzyskany dostęp do pliku.

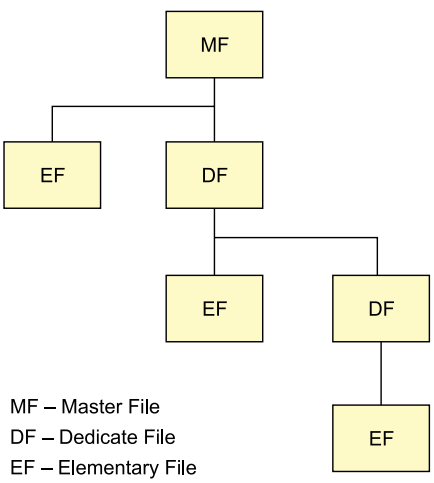
Ciało komendy APDU składa się z trzech pól *Lc*, *Data field*, *Le*. Pole *Lc* określa z ilu bajtów składa się pole danych (*Data field*), natomiast pole *Le* mówi o tym, ilu bajtów czytnik spodziewa się od karty identyfikacyjnej w odpowiedzi na wysłaną komendę. Wszystkie trzy pola są opcjonalne. Oznacza to, że jeśli komenda nie wymaga przesłania do karty dodatkowych danych oraz nie spodziewa się żadnej odpowiedzi, to cały pakiet ADPU będzie się składał jedynie z nagłówka. Druga możliwość wystąpi wtedy, kiedy dane są przesyłane do karty (pola *Lc* oraz *Data field* są zawarte w pakiecie), ale czytnik nie oczekuje odpowiedzi, czyli ciało komendy nie będzie zawierać pola *Le*. Ostatnia wariacja ma miejsce, gdy żadne dane nie są przesyłane do karty identyfikacyjnej wraz z komendą (brak pól *Lc* i *Data field*), ale oczekiwana jest odpowiedź ze strony karty, czyli ciało komendy składa się tylko z pola *Le*. Wszystkie trzy przypadki ilustruje **rysunek 6**.

Pakiet odpowiedzi na komendę również jest podzielony na dwie części, pierwsza to ciało (*body*), które jest w całości polem danych, natomiast na drugą (tzw. *Trailer*) składają się dwa pola *SW1* i *SW2*. Ostatnie pola mogą zawierać informację przykładowo o klasie błędu oraz jego konkretny kod.

Wiemy już, jak są zbudowane podstawowe jednostki wymiany informacji pomiędzy kartą identyfikacyjną, a czytnikiem, warto się teraz lepiej przyjrzeć, jak właściwie komunikacja z wykorzystaniem interfejsu Smartcard wygląda. Na **rysunku 7** przedstawiono diagram ilustrujący przepływ danych i poszczególne elementy interfejsu Smartcard. Aplikacja używa pakietów ADPU, które są przez warstwę łącza danych tłumaczone na pakiety TDPU. Te po stronie mikroprocesora karty identyfikacyjnej są z powrotem kodowane do postaci ADPU, by następnie mogły zostać wykonane żądane operacje w systemie plików karty.

System plików i biblioteka API

Podstawowy system plików kart identyfikacyjnych Smartcard składa się z tylko trzech typów plików:



Rysunek 8. Szkielet możliwego systemu plików karty identyfikacyjnej

– plik nadrzędny MF (*Master file*, *root*);
– pliki dedykowane DF (*Dedicated file*, katalogi);
– pliki elementarne EF (*Elementary file*).

Hierarchię plików przedstawiono na **rysunku 8**. W systemie istnieje tylko jeden katalog nadrzędny MF. Pliki dedykowane DF są analogią katalogów, a zatem mogą być puste, zawierać pliki elementarne EF lub inne katalogi. Każdy z plików, łącznie z plikiem nadrzędnym MF, posiada swój unikalny identyfikator dwubajtowy. Identyfikator pliku MF jest z góry ustalony, a jego wartość wynosi szesnastkowo 0x3F00. Dzięki nadanym identyfikatorom każdy plik elementarny EF w systemie ma swoją unikalną ścieżkę dostępu.

Właściwe dane są przechowywane w plikach elementarnych, które mogą należeć do jednego z poniższych czterech rodzajów:
– plik EF przeźroczysty (*Transparent file*);
– plik EF liniowy z ustaloną długością o organizacji w rekordy;
– plik EF liniowy ze zmienną długością o organizacji w rekordy;
– plik EF o organizacji kołowej w rekordy.

Należy się kilka słów wyjaśnienia, jak powyższe rodzaje plików są zbudowane i do czego służą. Najbardziej surową postacią danych jest format transparentny (przeźroczysty), który jest po prostu ciągiem bajtów, a więc jest to odpowiednik pliku binarnego. Dostęp do zawartości takiego pliku wymaga podania przesunięcia (offsetu) od początku pliku oraz liczby bajtów, które mają być zapisane lub odczytane. Nieco bardziej skomplikowane są pozostałe formaty. Dane w typach rekordowych są zorganizowane w logiczne grupy o ustalonej lub zmiennej długości. Takie skomplikowanie będzie w docelowej aplikacji rzutowało na jej złożoność. Przedstawiona dalej aplikacja wykorzystuje podstawowy, binarny typ plików.

Operacje na plikach mogą się odbywać albo w sposób niezabezpieczony, przy użyciu standardowych funkcji zapisu, odczytu itd., oraz z wykorzystaniem mechanizmów zabezpieczeń. System zabezpieczeń również może być dwojaki. Prostszy polega na włączeniu wymogu podawania numeru PIN (*Personal Identification Number*) w celu dokonania akcji na systemie plików karty. Zdecydowanie bardziej zaawansowany system zabezpieczeń może opierać się o wymianę kluczy autoryzacyjnych. W tym ostatnim przypadku autoryzacja może być przeprowadzana po stronie karty identyfikacyjnej lub po stronie układu czytnika.

Autoryzacja wewnętrzna (*Internal Authenticate*) działa następująco. Aplikacja zaszyta w mikrokontrolerze czytnika generuje losowy klucz, który jest następnie zakodowany według algorytmu szyfrującego i wysyłany do karty Smartcard. Ta dekoduje otrzymany ciąg bajtów korzystając z tajnego klucza, a następnie odsyła do czytnika wynik operacji. Aplikacja po stronie czytnika dokonuje porównania otrzymanych danych z pierwotnie wygenerowanymi losowymi wartościami. Jeśli operacja porównania zwróci wartość prawdziwą, to oznacza, że karta została zautentykowana poprawnie.

Tabela 1. Funkcje biblioteki firmy ST dla interfejsu Smartcard

Nazwa funkcji	Komentarz
SC_Handler	Służy do dwukierunkowej wymiany danych z kartą identyfikacyjną
SC_PowerCmd	Włącza lub wyłącza zasilanie karty
SC_Reset	Ustawia w stan wysoki bądź niski wyprowadzenie zerujące mikroprocesor karty
SC_ParityErrorHandler	Po wystąpieniu błędu parzystości pozwala na ponowne wysłanie bajtu do karty
SC_PTSConfig	Pozwala na skonfigurowanie prędkości wymiany danych

Autoryzacja zewnętrzna (*External Authenticate*) jest nijako odwrotna. Mikrokontroler czytnika otrzymuje losowe dane, szyfruje je i odsyła do karty identyfikacyjnej, by ta mogła je zdeszyfrować i porównać z losowymi danymi pierwotnymi. Ponieważ karta Smartcard musi zostać poproszona o podjęcie jakiejkolwiek akcji, to w pierwszej kolejności należy do niej wysłać prośbę o wygenerowania losowych danych.

Firma STMicroelectronics do mikrokontrolerów STM32 dostarcza bibliotekę funkcji API dla interfejsu Smartcard. Jest ona zawarta w archiwum biblioteki *Standard Peripherals Library*, dostępnej na stronie producenta. Na bibliotekę Smartcard składa się pięć funkcji skrótowo skomentowanych w **tabeli 1**.

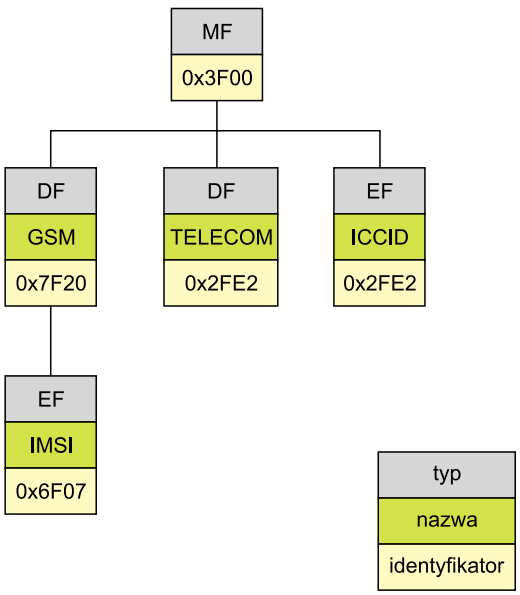
Narzędziem wykorzystywanym bezpośrednio podczas komunikacji z kartą identyfikacyjną jest funkcja SC_Handler(). Wszelka wymiana danych odbywa się za pomocą wywołań tej funkcji. W argumentach funkcja SC_Handler() wymaga między innymi podania wskaźnika na stan karty. Wyróżnia się sześć dozwolonych stanów karty:
– SC_POWER_OFF – Zasilanie nie jest doprowadzone do karty, ponadto wyłączony jest również cały interfejs Smartcard (włączając w to sygnał zegarowy dla karty).
– SC_POWER_ON – Zasilanie nie jest doprowadzone do karty, interfejs Smartcard jest jednak włączony, lecz sygnał zegarowy nadal nie jest doprowadzony.
– SC_RESET_LOW – W tym stanie powinno nastąpić otrzymanie odpowiedzi na sygnał zerowania ATR (*Answer to Reset*). Wyprowadzenie RST znajduje się w stanie niskim, sygnał zegarowy dla karty identyfikacyjnej jest włączony.
– SC_RESET_HIGH – Sekwencja ATR nadal nie została odebrana. Czytnik wymusza na wyprowadzeniu RST stan wysoki i utrzymuje go, aż nie otrzyma poprawnego ATR.
– SC_ACTIVE – Odebrano poprawą ramkę ART i następuje jej dekodowanie, dzięki czemu będzie można określić, jaki protokół komunikacyjny jest zastosowany.
– SC_ACTIVE_ON_T0 – Jeśli ustawiony jest ten stan, to znaczy, że użyto protokół komunikacyjny T0 i próba wymiany danych z kartą identyfikacyjną może zostać podjęta.

Sposób użycia SC_Handler() i pozostałych funkcji z biblioteki API przedstawia niżej omówiona aplikacja.

Aplikacja Smartcard

Zaproponowana aplikacja to zedytowana wersja przykładu dla interfejsu Smartcard zawartego w bibliotece API firmy ST. Wykorzystano kartę Smartcard dostarczaną wraz z zestawem STM3210B-EVAL, na której znajduje się system plików zgodny ze standardem GSM11.11. Filozofia używania kart identyfikacyjnych jest jednak na tyle elastyczna, że nie powinno być problemów podczas migracji aplikacji do współpracy z kartą z inną zawartością.

Standard GSM11.11 został opracowany przez ETSI (*European Telecommuni-*



Rysunek 9. Podstawowe drzewo plików w standardzie GSM11.11

cations Standards Institute) w połowie lat 90-tych na potrzeby telefonii komórkowej, a konkretnie dla kart SIM dla urządzeń mobilnych (*Subscriber Identity Module – Mobile Equipment*). Dla naszej aplikacji istotny jest jedynie mały wycinek z całego systemu plików GSM11.11. Ten niewielki fragment przedstawiono na **rysunku 9**. Na rysunku zamieszczono dwa pliki: ICCID oraz IMSI. Plik elementarny ICCID (*IC Card Idenfification*) zawiera unikalny numer identyfikacyjny karty, jego dwubajtowy identyfikator w systemie plików to 0x2FE2. Identyfikator drugiego pliku elementarnego – IMSI (*International Mobile Subscriber Identity*) – to 0x6F07. Poniżej przedstawiono sposób odczytu pliku ICCID.

Kod funkcji main() przedstawiono na **listingu 1**. Przed przystąpieniem do nawiązywania komunikacji z kartą identyfikacyjną należy odpowiednio skonfigurować system zegarowy mikrokontrolera STM32, timer systemowy SysTick oraz wyprowadzenie mikrokontrolera, do którego podłączony jest pin gniazda, detekujący obecność karty. W tej roli użyto wyprowadzenie PE14. Obecność karty w gnieździe ma generować przerwanie, a więc trzeba skonfigurować sterownik przerwań NVIC oraz sterownik przerwań zewnętrznych EXTI. Stosowny fragment programu, ciało funkcji SC_DetectPinConfig(), zamieszczono na **listingu 2**.

Na początku mikrokontroler pozostaje w pętli while() dopóty, dopóki w gnieździe nie zostanie umieszczona karta identyfikacyjna. Jeśli to nastąpi to wygenerowane zostanie przerwanie zewnętrzne od wspomnianego wyżej wyprowadzenia PE14. W konsekwencji mikrokontroler wywoła funkcję obsługi przerwania zewnętrznego EXTI15_10_IRQHandler() z pliku stm32f10x_it.c. Zamieszczono ją na **listingu 3**. Jest to standardowy kod obsługi przerwania dla układów z rodziny STM32, a zatem na początku sprawdzane jest, od której linii przerwanie pochodzi, a następnie mikrokontroler zeruje flagę przerwania i przystępuje do wykonywania właściwego kodu aplikacji. Po zmianie flagi obecności karty w gnieździe (CardInserted) na 1, uruchamiana jest procedura aktywacji karty i jej zerowania.

Po powrocie z ISR warunek wykonywania się pierwszej pętli while() nie jest już spełniony. Oznacza obecność karty w gnieździe, a więc aplikacja przystępuje do próby odebrania odpowiedzi na sygnał zerowania (ATR), ale najpierw za pomocą wywołania funkcji SC_PTSConfig() ustalana jest prędkość komunikacji.

Gdy ramka ATR jest już odebrana i zdekodowana, mikrokontroler rozpoczyna poruszanie się po systemie plików karty Smartcard. Najpierw sprawdzana jest obecność pliku głównego MF (*Master File*), a następnie wybierany jest plik ICCID. Jeśli te czynności zostaną zakończone powodzeniem, aplikacja przystępuje do odczytu zawartości pliku ICCID, jest to 10 bajtów, ponieważ taki jest właśnie jego rozmiar na użytej karcie identyfikacyjnej.

Jak widać, podstawowa obsługa kart Smartcard nie jest zbytnio skomplikowana. Taki stan rzeczy nie wynika oczywiście z banalności samego standardu, ponieważ z pewnością banalnym on nie jest. Dzięki sprzętowemu wsparciu ze strony układów STM32, oraz dzięki udostępnianiu przez firmę STMicroelectronics bibliotek, implementacja obsługi kart identyfikacyjnych we własnych aplikacjach nie powinna nastręczać większych problemów.

Krzysztof Paprocki
krzysztof.paprocki@gmail.com