Urządzenia Peryferyjne

Ćwiczenie 3 – Karty Mikroprocesorowe (GSM/SIM)

**Autorzy**

Dawid Waligórski (264015)

Adrian Kotula (263989)

**Data wykonania ćwiczenia**

25.01.2024

**Prowadzący**

Dr. Inż. Dariusz Caban

Spis treści

[1 Cele ćwiczenia 2](#_Toc157477116)

[2 Wstęp teoretyczny 2](#_Toc157477117)

[2.1 Budowa i działanie kart mikroprocesorowych 2](#_Toc157477118)

[2.2 Komunikacja z kartą mikroprocesorową (stykową) 3](#_Toc157477119)

[2.3 Zestaw komend używanych w ćwiczeniu 4](#_Toc157477120)

[3 Realizacja zadania laboratoryjnego 5](#_Toc157477121)

[3.1 Przetestowanie kart SIM oraz podejrzenie ich zawartości 5](#_Toc157477122)

[3.2 Implementacja aplikacji 5](#_Toc157477123)

[3.2.1 Opis działania aplikacji 5](#_Toc157477124)

[3.2.2 Wyświetlanie nazwy podłączonego czytnika i wybór czytnika 7](#_Toc157477125)

[3.2.3 Łączenie czytnika z kartą 7](#_Toc157477126)

[3.2.4 Wysyłanie komend do czytnika i odbiór odpowiedzi 8](#_Toc157477127)

[3.2.5 Odczytanie listy kontaktów 10](#_Toc157477128)

[3.2.6 Dekodowanie wiadomości ATR 10](#_Toc157477129)

[4 Protokół z laboratorium 10](#_Toc157477130)

[5 Podsumowanie 10](#_Toc157477131)

[6 Źródła 10](#_Toc157477132)

# Cele ćwiczenia

Celem ćwiczenia było zapoznanie studentów z działaniem kart mikroprocesorowych GSM/SIM a także sposobem ich obsługi przy pomocy komend APDU. W jego ramach mieli oni napisać aplikację pozwalającą na odczytywanie przy pomocy interfejsu PC/SC książki telefonicznej i listy SMS-ów zapisanych na dostępnych w laboratorium kartach.

# Wstęp teoretyczny

## Budowa i działanie kart mikroprocesorowych

Karty mikroprocesorowe, znane również jako karty SIM, są formą niewielkiego, mobilnego nośnika danych, który może być wykorzystywany w celu pełnienia różnego rodzaju usług (m.in. telefonicznych). Wyposażone są one w układ scalony, który pełni kluczową rolę w przetwarzaniu oraz przechowywaniu danych. Ów układ można podzielić zasadniczo na 3 główne komponenty:

* **Mikroprocesor** – odpowiadający za przetwarzania informacji przychodzących z zewnątrz jak i z wewnątrz karty.
* **Pamięć** – odpowiadającą za przechowywanie danych odpowiadających usłudze, której przypisana jest karta. Najczęściej pozwalającą na wielokrotny zapis z funkcją kasowania impulsem elektrycznym (*EEPROM, eng. electrically erasable programmable read only memory*).
* **Interfejs** – dla kart stykowych obecnego formie zbioru widocznych na awersie karty pozłacanych styków, służących do komunikacji z innymi urządzeniami. Wśród owych styków można wyróżnić m.in. styk informacyjny pozwalający na dwustronną (simpleksową) komunikację z urządzeniami zewnętrznymi, a także styk z napięciem odniesienia (*gnd*) oraz styk doprowadzający zewnętrzne napięcie programowania (*Vpp*), wymagane przy zapisie danych do karty.

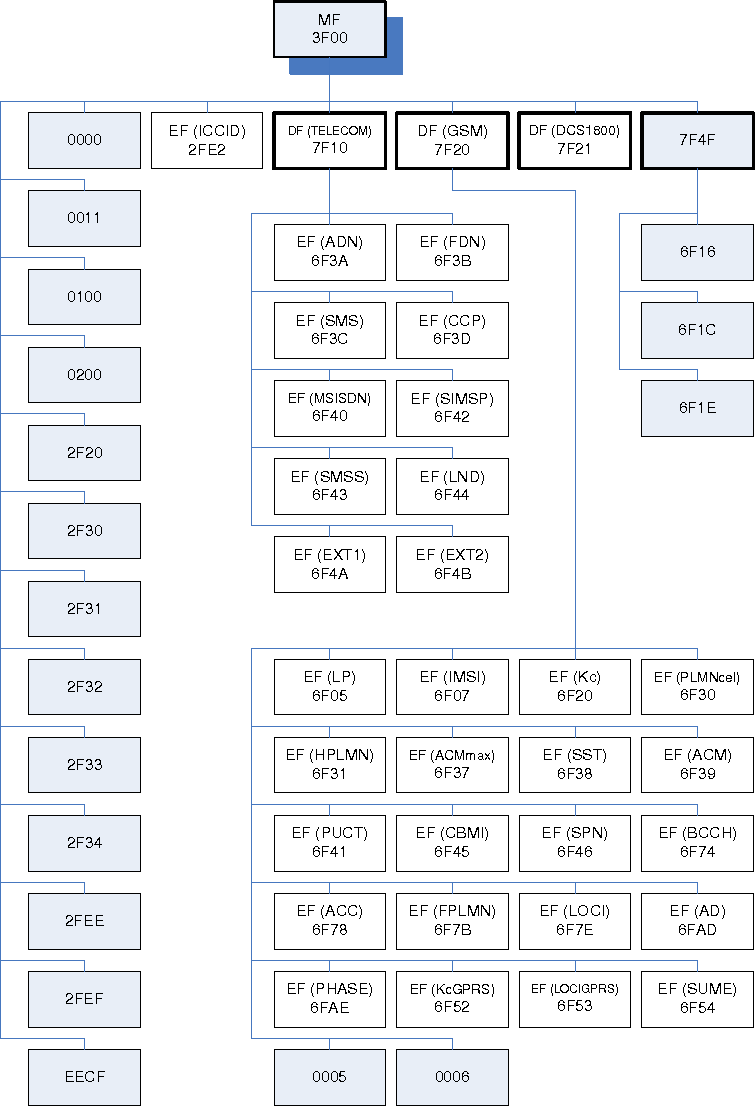
Karty SIM w odróżnieniu do kart magnetycznych oferują wyższy poziom bezpieczeństwa danych. Wynika to z faktu, iż mikroprocesor karty posiada możliwość kontrolowania dostępu do pewnych obszarów pamięci. Pierwszym z nich jest obszar swobodnego odczytu, który zawiera informacje powszechnie znane nt. karty i jej użytkownika. Drugim, obszar poufny, który dostępny jest dopiero po podaniu odpowiedniego kodu PIN (przy czym kilkukrotne podanie błędnego kodu może zostać wykryte i spowodować blokadę karty). W końcu ostatnim obszarem jest obszar roboczy karty, który zawiera dane ulegające ciągłej modyfikacji.

Ponadto karty mikroprocesorowe podzielić można ze względu na typ posiadanego przez nie interfejsu komunikacji z urządzeniami zewnętrznymi. W ten sposób wyróżnić można karty stykowe (zgodne normą *ISO-78156*) oraz karty bezstykowe (zgodne z normą *ISO-14443*), które pozwalają na komunikację przy pomocy fal elektromagnetycznych. W przypadku realizowanego zadania do dyspozycji były wyłącznie karty stykowe.

System plików na kartach SIM jest hierarchiczny (rys. 1). Podzielony może on zostać na trzy zasadnicze typy plików:

* **Master file (MF)** – katalog główny.
* **Dedicated file (DF)** – katalog karty, będący odpowiednikiem folderu.
* **Elementary file (EF)** – zbiór, będący odpowiednikiem pliku.

Każdy plik dowolnego typu zawarty na karcie przypisany ma ponadto 2-bajtowy adres (nazwa pliku), służący do jego jednoznacznej identyfikacji. Z zasady zapisywany jest on w postaci heksadecymalnej. W przypadku realizowanego zadania w centrum zainteresowania znalazły się pliki EF ADN (książka adresowa) oraz EF SMS (zawierający SMS-y) z DF TELECOM.



Rysunek : Schemat systemu plików karty mikroprocesorowej (GSM\SIM)

## Komunikacja z kartą mikroprocesorową (stykową)

Komunikacja ze stykową kartą mikroprocesorową może zostać przeprowadzona z użyciem specjalnego interfejsu PC/SC. Po włożeniu do niego karty wysyła on do niej sygnał *RESET*-u. Karta odpowiada na niego 32-znakową sekwencją *ATR* (*answer to reset*). Owa sekwencja przenosi dane niezbędne do nawiązania skutecznej komunikacji, na które składają się między innymi:

* Rodzaju karty
* Długości trwania jednego bitu, czyli *ETU* (*elementary time unit*)
* Używanym protokole komunikacyjnym (najczęściej *T0* lub *T1*)
* Częstotliwości sygnału zegarowego karty
* Napięciach, mocach i prądach niezbędnych do poprawnego programowania karty

Dalsza komunikacja między interfejsem a kartą również odbywa się w trybie simpleksowym, zgodnym z protokołem *APDU* (*Application Protocol Data Unit*). Polega ona na wysyłaniu odpowiednich komend przez interfejs stykowy oraz wysyłaniu przez kartę odpowiedzi na owe komendy.

Jednostka danych przenosząca komendę dla karty określana jest jako *Command PDU* (*CPDU*). Ogólna struktura tej jednostki przestawiona została w tabeli 1.

Tabela : Ogólna struktura CPDU

|  |  |
| --- | --- |
| CLA | Klasa karty reprezentowana przez wartość 1-bajtową. Wynosi 0xA0 dla kart rozważanych na laboratorium. |
| INS | Identyfikator instrukcji dla mikroprocesora karty reprezentowana przez wartość 1-bajtową. |
| P1 | Parametr 1 instrukcji, będący liczbą 1-bajtową. |
| P2 | Parametr 2 instrukcji, będący liczbą 1-bajtową. |
| Lc | Długość pola danych komendy (w bajtach). |
| DATA | Pole danych komendy. |
| Le | Spodziewana długość odpowiedzi (w bajtach). |

Jednostka danych przenosząca odpowiedź karty na komendę określana jest jako *Response PDU* (*RPDU*) Jej ogólna struktura przedstawiona została w tabeli 2. Zawiera ona w sobie 2-bajtową informację SWx, która pozwala ustalić status wykonania uprzednio wykonanej instrukcji. Przykładowymi statusami są:

* 0x9000 – komenda wykonała się poprawnie
* 0x61\*\* – komenda wykonała się poprawnie. Do odebrania pozostało \*\* bajtów odpowiedzi.
* 0x69\*\* – komenda jest niedozwolona. Przyczyna jest zakodowana przez liczbę \*\*.

Tabela : Ogólna struktura RPDU

|  |  |
| --- | --- |
| DATA | Dane odpowiedzi. |
| SW1 | Wyższy bajt słowa statusowego odpowiedzi. |
| SW2 | Niższy bajt słowa statusowego odpowiedzi. |

## Zestaw komend używanych w ćwiczeniu

W celu realizacji ćwiczenia przygotowano zestaw niezbędnych do odczytania EF ADN oraz EF SMS CPDU. Były to:

* SELECT FILE – komenda wybierająca wskazany plik z karty. Jej struktura zawarta została w tabeli 3.

Tabela : Struktura komendy SELECT FILE

|  |  |
| --- | --- |
| CLA | 0xA0 |
| INS | 0xA4 |
| P1 | 0x00 |
| P2 | 0x00 |
| Lc | 0x02 (długość adresu) |
| DATA | 2-bajtowy adres pliku |
| Le | - |

* GET RESPONSE – komenda pobierająca z karty odpowiedź na ostatnią komendę. Jej struktura została zawarta w tabeli 4.

Tabela : Struktura komendy GET RESPONSE

|  |  |
| --- | --- |
| CLA | 0xA0 |
| INS | 0xC0 |
| P1 | 0x00 |
| P2 | 0x00 |
| Lc | - |
| DATA | - |
| Le | Długość spodziewanej odpowiedzi (w bajtach) |

* READ RECORD – komenda odczytująca rekord z aktualnie wybranego EF. Jej struktura zawarta została w tabeli 5.

Tabela : Struktura komendy READ RECORD

|  |  |
| --- | --- |
| CLA | 0xA0 |
| INS | 0xB2 |
| P1 | Numer rekordu do odczytania |
| P2 | 0x04 (odczytanie wyłącznie rekordu z P1) |
| Lc | - |
| DATA | - |
| Le | Długość spodziewanej odpowiedzi (w bajtach) |

# Realizacja zadania laboratoryjnego

## Przetestowanie kart SIM oraz podejrzenie ich zawartości

W pierwszej części laboratorium zapoznano się z zestawem dostępnych kart SIM. Były to karty telefoniczne należące do operatorów: *Plus* oraz *T-Mobile*. W tym celu każdą z nich włożono do podłączonego do komputera interfejsu *OMNIKEY CardMan* oraz spróbowano odczytać ich zawartość przy pomocy programu *SIMCardManager*. Mimo licznych prób niemożliwym okazało się odczytanie zawartości karty *Plus*. Z powodzeniem odczytano jednak zawartość karty *T-Mobile* (rys. 2). Okazało się, iż zawiera ona wyłącznie dane w postaci książki adresowej. W związku z tym porzucono jeden z punktów ćwiczenia polegający na odczytaniu listy SMS-ów.

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

Rysunek : Wynik odczytania danych z karty T-Mobile widoczny w programie SIMCardManager

## Implementacja aplikacji

### Opis działania aplikacji

W drugiej części laboratorium studenci mieli zaimplementować aplikację na komputer PC, która umożliwić miała:

* Wyświetlenie nazwy podłączonego do komputera czytnika kart
* Wybór czytnika kart
* Wysyłanie komend (w formie HEX) do czytnika oraz wyświetlenie odpowiedzi karty (zarówno w formie HEX jak i znakowo)
* Odczytanie listy kontaktów (książkę telefoniczną)
* Odczytanie listy SMS-ów, wyświetlanie danych technicznych oraz ich treści

W ramach zajęć udało się zaimplementować jedynie część z wymienionych funkcjonalności (szczegóły w odpowiednich podsekcjach). Z powodów opisanych w sekcji *3.1* nie udało się odczytać listy SMS-ów. Niepowodzeniem zakończyły się także próby odczytania listy kontaktów zapisanej na karcie.

Stworzona aplikacja została wykonana za pomocą technologii *.NET* (*Windows Forms*). Umożliwiło to łatwe wyposażenie jej w graficzny interfejs użytkownika (rys. 3). Działanie poszczególnych elementów zawartych na interfejsie przedstawiono w tabeli 6. Przy implementacji aplikacji skorzystano również z biblioteki *pcsc-sharp* dla języka *C#* [2]. Pozwala ona na proste oprogramowywanie komunikacji komputera z kartą mikroprocesorową za pośrednictwem interfejsu PC/SC.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Rysunek : Wygląd graficznego interfejsu użytkownika zaimplementowanej aplikacji

Tabela : Opis działania poszczególnych elementów interfejsu

|  |  |
| --- | --- |
| **Element interfejsu** | **Funkcja** |
| Przycisk „Połącz z kartą” | Nawiązanie połączenia z kartą przy pomocy czytnika wybranego na liście „Czytnik”. |
| Przycisk „Wyślij” | Wysłanie do czytnika komendy CPDU. |
| Lista „Komenda” | Lista jednokrotnego wyboru, zawierająca możliwe do wysłania przy pomocy programu komendy CPDU. |
| Pole „Odpowiedź (hex)” | Przedstawienie odpowiedzi karty na ostatnią wysłaną komendę w postaci HEX. |
| Pole „Odpowiedź (ASCII)” | Przedstawienie odpowiedzi karty na ostatnią wysłaną komendę w postaci znakowej. |
| Lista „Czytnik” | Lista jednokrotnego wyboru spośród wszystkich dostępnych czytników PC/SC. |

### Wyświetlanie nazwy podłączonego czytnika i wybór czytnika

Wraz z inicjalizacją aplikacji dokonuje ona skanowania systemu w celu identyfikacji podłączonych do komputera czytników przy pomocy metody *getReaders* (listing 1), która uzupełnia listę czytników aplikacji. Dodatkowo, w przypadku odnalezienia co najmniej jednego czytnika PC/SC metoda automatycznie ustala domyślnie wybrany czytnik (pierwszy znaleziony czytnik). W przeciwnym razie, gdy nie wykryty zostanie żaden czytnik, zwróci ona odpowiedni status, który po odczytaniu spowoduje awaryjne zamknięcie aplikacji (przy jednoczesnym poinformowaniu użytkownika o powodzie zamknięcia).

Listing : Kod metody pozwalającej na uzyskanie informacji o czytnikach podłączonych do komputera i wybór czytnika

**bool** getReaders()

{

context.Establish(SCardScope.System);

**string**[] availableReaders = context.GetReaders();

**if**(availableReaders.Length == 0)

{

MessageBox.Show("Brak czytnika");

**return** **false**;

}

**foreach** (**var** availableReader **in** availableReaders)

{

readerList.Items.**Add**(availableReader);

}

readerList.SelectedIndex = 0;

**return** **true**;

}

### Łączenie czytnika z kartą

Łączenie czytnika z kartą inicjowane jest za pomocą przycisku „Połącz z kartą”. Jego wciśnięcie powoduje wywołanie metody *connectButton\_Click* (listing 2). Powoduje ona połączenie komputera z wybranym w GUI czytnikiem, a następnie próbę nawiązania połączenia między czytnikiem a kartą mikroprocesorową (przy użyciu protokołów *T0* lub *T1*). Później metoda weryfikuje powodzenia tej próby przy pomocy metody *checkCardError* (listing 3). W przypadku pomyślnej weryfikacji aktywowany jest dostęp do funkcji wysyłania komend oraz odbierana i wyświetlana w GUI jest sekwencja *ATR*.

Listing : Metoda pozwalająca na połączenie z komputera z kartą za pośrednictwem wybranego czytnika PC/SC

**private** **void** connectButton\_Click(**object** sender, EventArgs e)

{

reader = new SCardReader(context);

SCardError error = reader.Connect(

readerList.SelectedItem.ToString(),

SCardShareMode.Shared,

SCardProtocol.T0 | SCardProtocol.T1

);

**if** (!checkCardError(error))

**return**;

**if** ( reader.ActiveProtocol != SCardProtocol.T0 &&

reader.ActiveProtocol != SCardProtocol.T1)

{

MessageBox.Show(

"Aktywny protokół nie jest odbługiwany!");

sendCommandButton.Enabled = **false**;

}

**else**

{

MessageBox.Show("Połączono z kartą");

sendCommandButton.Enabled = **true**;

**byte**[] buffer;

reader.GetAttrib(

SCardAttribute.AnswerToResetString, **out** buffer

);

printToRespondeBox(buffer);

parseAtr(buffer);

}

}

Listing : Metoda sprawdzająca wystąpienie błędu w komunikacji z kartą

**bool** checkCardError(SCardError error)

{

**if**(error != SCardError.Success)

{

MessageBox.Show(

"Wystąpił błąd karty: " + error.ToString()

);

**return** **false**;

}

**return** **true**;

}

### Wysyłanie komend do czytnika i odbiór odpowiedzi

Wysłanie komendy do czytnika inicjowane jest za pomocą przycisku „Wyślij”. Jego wciśnięcie powoduje wywołanie metody *sendCommand* (listing 4). Powoduje ona pobranie aktualnie wybranej komendy z listy dostępnych komend widocznej w GUI. Następnie buduje ona odpowiednią, zgodną z treścią sekcji *2.3* *CPDU*. Owa *CPDU* zostaje później wysłana do karty, a poprawność transmisji zostaje zweryfikowana przy pomocy metody *checkCardError*. W przypadku pozytywnej weryfikacji wyświetlona na GUI zostaje również odebrana odpowiedź karty. Wykorzystywana jest przy tym metoda *printToRespondeBox* (listing 5).

Listing : Metoda wysyłająca wybraną komendę do czytnika i odbierająca odpowiedź na nią

**void** sendCommand()

{

**string** commandString = commandList.SelectedItem.ToString();

**byte**[] command = new **byte**[] { };

**byte**[] response = new **byte**[256];

**switch** (commandString)

{

**case** "select file telecom":

command = new **byte**[]

{ 0xA0, 0xA4, 0x00, 0x00, 0x02, 0x7F, 0x10 };

**break**;

**case** "select file adn":

command = new **byte**[]

{ 0xA0, 0xA4, 0x00, 0x00, 0x02, 0x6f, 0x3a };

**break**;

**case** "select file mf":

command = new **byte**[]

{ 0xA0, 0xA4, 0x00, 0x00, 0x02, 0x3f, 0x00 };

**break**;

**case** "read record":

command = new **byte**[]

{ 0xA0, 0xB2, 0x00, 0x04, 0x1C};

**break**;

**case** "get response":

command = new **byte**[]

{ 0xa0, 0xc0, 0x00, 0x00, 0xFF};

**break**;

**default**:

MessageBox.Show("Nieprawidłowa komenda");

**return**;

}

SCardError error = reader.Transmit(command, **ref** response);

**if** (!checkCardError(error))

**return**;

printToRespondeBox(response);

}

Listing : Metoda wyświetlająca odpowiedź karty

**void** printToRespondeBox(**byte**[] response)

{

**string** answerStrHex =

BitConverter.ToString(response).Replace("-", " ");

responseBoxHex.Text = answerStrHex;

**string** answerStrAscii = "";

**switch** (response[0])

{

**case** 0x9f:

answerStrAscii = "Komenda wykonała się poprawnie.

Pozostało " + response[1] + " bajtów odpowiedzi.";

**break**;

**default**:

answerStrAscii =

System.Text.Encoding.ASCII.GetString(response);

**break**;

}

responseBoxAscii.Text = answerStrAscii;

}

### Odczytanie listy kontaktów

W ramach laboratorium podjęto również próby odczytania listy kontaktów karty. Planowano w tym celu wykorzystać sekwencję instrukcji:

1. SELECT FILE TELECOM – przejście do DF TELECOM
2. SELECT FILE ADN – przejście do EF ADN
3. GET RESPONSE – odczytanie metadanych pliku (zawierających m.in. długość rekordu)
4. READ RECORD – odczytanie rekordu z EF ADN
5. GET RESPONSE – odczytanie zawartości odczytanego rekordu

Komendy wydawane miały być przy pomocy metody *sendCommand*, a ich wyniki wyświetlane na GUI przy pomocy metody *printToRespondeBox*. Z powodzeniem udało się wykonać komendę 1 (rys. 4) oraz 2 (rys. 5). Niestety mimo licznych prób i zmian w strukturze CPDU nie udało się wykonać komend 3-5 (rys. 6). Za każdym razem słowa statusowe odpowiadających im RPDU wskazywały na wystąpienie błędu w podanej długości spodziewanej odpowiedzi (0x6700).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

Rysunek : Wynik wysłania komendy SELECT FILE TELECOM

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

Rysunek : Wynik wysłania komendy SELECT FILE ADN

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

Rysunek : Wynik wysłania komendy READ RECORD

### Dekodowanie wiadomości ATR

W związku z nieudaną implementacją odczytu listy kontaktów postanowiono spróbować zdekodować zawartość sekwencji *ATR*. Dokonuje tego metoda *parseAtr* (listing 6). Ze względu na zakończenie się laboratorium udało się odkodować jedynie szczątkowe informacje z tejże sekwencji. Wyniki dekodowania można zobaczyć na rysunku 3.

Listing : Metoda dekodująca sekwencję ATR

**void** parseAtr(**byte**[] atr)

{

**string** atrStr = "";

**if** (atr[0] == 0x3b)

atrStr += "Direct Convention";

**else** **if** (atr[0] == 0x3f)

atrStr += "Inverse Convention";

atrStr += ", ";

atrStr += "Protokół T" + (atr[3] % 16).ToString() + ", ";

**int** historical\_bytes = atr[1] % 16;

atrStr += (atr[1]%16).ToString() + " znaków historycznych: ";

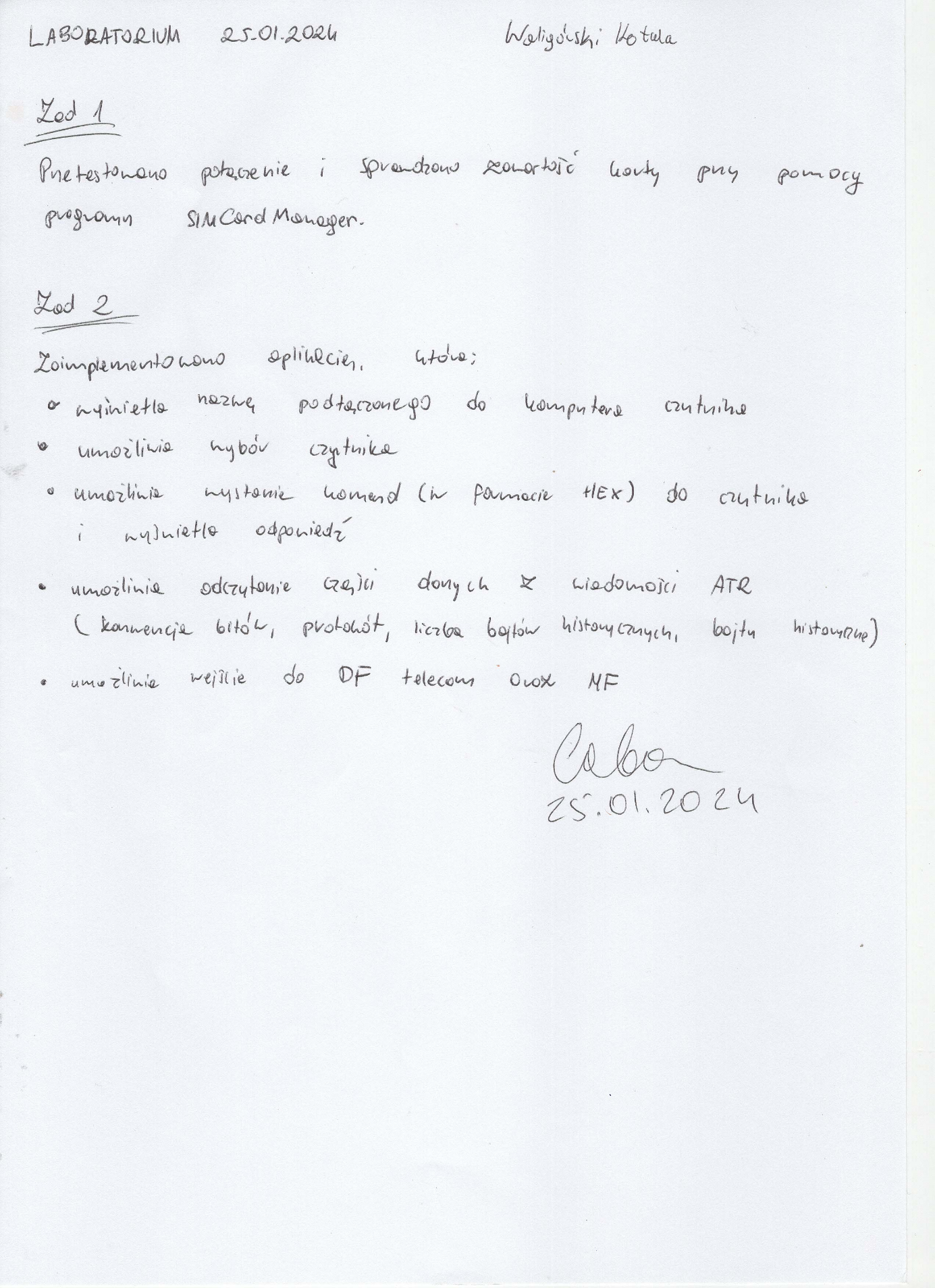
**for** (**int** i = 7; i < 7 + historical\_bytes; i++)

atrStr += atr[i] + " ";

responseBoxAscii.Text = atrStr;

}

# Protokół z laboratorium



# Podsumowanie

Niestety nie udało się zaimplementować wszystkich zakładanych funkcjonalności. Mimo wszystko ćwiczenie pozwoliło jednak na zapoznanie się z działanie kart mikroprocesorowych. Umożliwiło ono również na poznanie w praktyce sposobu komunikacji z tego typu urządzeniem przy pomocy protokołu APDU.

# Źródła

[1] Dokumentacja APDU dostępna pod adresem (stan na styczeń 2024): [ISO7816 part 4 section 6 with Basic Interindustry Commands (APDU level) (cardwerk.com)](https://cardwerk.com/smart-card-standard-iso7816-4-section-6-basic-interindustry-commands/)

[2] Repozytorium *GitHub* z kodem użytej biblioteki *pcsc-sharp* (stan na styczeń 2024): <https://github.com/danm-de/pcsc-sharp>