Doręczenie i odbiór SMS w trybie PDU (np.interpreter komend).

Opis interpretera poleceń
Odbiór wiadomości SMS
Wysłanie wiadomości SMS
Przykłady kodowania wiadomości SMS w trybie PDU

Opisywany przykład aplikacji, to prosty interpreter poleceń. W połączeniu z telefonem komórkowym (działanie aplikacji testowane było z telefonem GSM marki SIEMENS C35i) potrafi realizować proste polecenia wydawane ze pomocą SMS. Rezultaty wykonania poleceń oraz stan urządzenia raportowany jest również przez komunikaty SMS. Urządzenie zbudowane było w oparciu o płytkę prototypową z wbudowanym układem drivera interfejsu UART - MAX232 oraz z wykorzystaniem fabrycznego kabla połączeniowego dla telefonu komórkowego SIEMENS. Należy je traktować bardziej jako przykład wykonania aplikacji współpracującej z telefonem GSM aniżeli urządzenie gotowe do wykorzystania, chociaż z całą pewnością może posłużyć do jego budowy.

Opis interpretera poleceń.

Całość programu napisana została w języku C dla mikrokontrolera firmy ATMEL typu AT89S8252 taktowanego zegarem 7,3728MHz z dodatkową, zewnętrzną pamięcią RAM o rozmiarze 1kB. Użycie dodatkowego układu pamięci jest konieczne ze względu na konieczność buforowania odebranego komunikatu SMS. Policzmy:

- komunikat to maksymalnie 140 cyfr dwubajtowych w formie tekstu, to jest 280 bajtów koniecznych do zapamiętania łańcucha – części komunikatu o nazwie Dane Użytkownika,
- parametry SMS to w sumie 9 ważnych bajtów informacji,
- numery Centrum Usług i nadawcy wyrażone w formie łańcucha tekstowego to razem 24 bajty,
- reasumując: na zapamiętanie odebranego komunikatu SMS potrzebne jest 313 bajtów; niestety mikrokontroler AT89S8252 ma tylko 256 bajtów pamięci przeznaczonej na stos, zmienne itp.

Budując samodzielnie podobną aplikację można wykorzystać np. mikrokontroler T89C51-RD2. Posiada on wbudowane w strukturę 1280 bajtów pamięci RAM, którą można wykorzystać między innymi z przeznaczeniem na bufor odbioru komunikatów SMS.
Na listingu 1 umieściłem fragment programu wraz z definicjami odpowiednich stałych używanych przy kompozycji SMS oraz typów zmiennych. Stałe zawierają polecenia w formie łańcuchów tekstowych przesyłane do telefonu GSM. Odpowiednio są to:

- CPIN: wprowadzenie numeru PIN aparatu,
- CHOOSEMEM: fragment polecenia wybierającego domyślną lokalizację dla SMS,
- ECHOOFF: polecenie wyłączające echo wysłanej komendy,
- SETNOTICE: to ustawienie sposobu powiadamiania o komunikacie SMS,
- NOTICE: to komunikat, przy pomocy którego aplikacja powiadamiana jest o nowym, odebranym przez aparat GSM komunikacie SMS,
- SMSDEL: to polecenie usuwania wiadomości SMS z pamięci,
- SMSREAD: to polecenie odczytu odebranego SMS po powiadomieniu,
- SMSSEND: to polecenie wysłania wiadomości SMS bez pośrednictwa karty SIM.

W postaci stałych zdefiniowano również nagłówek wiadomości. Założono, że telefon pracuje w sieci Plus GSM, numer Centrum Usług i aparatu zawsze podawany jest w formacie międzynarodowym, okres ważności wiadomości (wysyłanej) wynosi 12 godzin.
Jako pierwszy zdefiniowany został bufor dla odebranej wiadomości SMS. Uwaga: jak wspomniano na początku, jego rozmiar przekracza rozmiar pamięci danych typowego układu 8951 / 8952. Następnie zadeklarowany został typ zmiennej służący do budowy wykazu poleceń realizowanych przez aplikację. Ten typ wykorzystywany jest do budowy tablicy zakwalifikowanej do obszaru *code*, ponieważ zawiera wartości stałe, nie ulegające zmianie w czasie wykonywania programu. Rozmiar tablicy – wykazu funkcji – nie jest ustalony. Jej koniec sygnalizuje znak o kodzie „0”.
Funkcja zawarta w definicji struktury musi zwracać jakąś wartość. Najłatwiej, gdy jest to wartość typu *char*, którą można później wykorzystać do sygnalizacji np. błędów realizacji poleceń, jednak może to być również inny typ zmiennych. Jest to wymóg konieczny dla późniejszej realizacji polecenia *return(wykaz[indeks].funkcja ())*, ponieważ polecenie *return* nie może zwracać wartości typu *void* oraz z powodów, o których będzie mowa dalej. Funkcja *command* powinna być tego samego typu, jak określono to w definicji struktury. Oczywiście powinna a nie musi. Jeśli typy będą różne, to nastąpi niejawna konwersja typu zwracanej wartości.

```
code char CPIN[] = "AT+CPIN=1643";          //wprowadzenie numeru PIN
code char CHOOSEMEM[] = "AT+CPMS=";         //wybór domyślnej lokalizacji pamięci
code char ECHOOFF[] = "ATE0";               //wyłączenie echa komendy AT
code char SETNOTICE[] = "AT+CNMI=1,1,0,2"; //ustawienie sposobu powiadamiania
code char NOTICE[] = "+CMTI:";              //powiadomienie o nowym SMS
code char SMSDEL[] = "AT+CMGD=";            //usunięcie SMS z pamięci
code char SMSREAD[] = "AT+CMGR=";           //odczyt SMS z pamięci
code char SMSSEND[] = "AT+CMGS=";           //wysłanie SMS

code char LOSCA = 0x07;                     //Długość numeru SCA+1
code char TOSCA = 0x91;                     //Typ numeracji SCA
code char SCA[] = "8406010013F0";           //Centrum Usług (tu dla sieci Plus GSM)
code char FO = 0x11;                        //Pierwszy oktet dla wysyłanego SMS
code char MR = 0x00;                        //Numer odniesienia dla komunikatu
code char LODA = 0x0B;                      //Liczba cyfr numeru telefonu ODBIORCA
code char TODA = 0x91;                      //Numeracja międzynarodowa dla CU i ODBIORCA
code char DA[] = "8406630364F1";            //Odbiorca wiadomości SMS
code char PID = 0x00;                       //Identyfikator protokołu (tekst)
code char DCS = 0x00;                       //Schemat kodowania
code char SCTS = 0x8F;                      //Okres ważności 12 godzin

idata char UDbuf[320];                      //Bufor dla komunikatu SMS
                                            //Definicje nagłówków funkcji programu
char in(idata char *bufor);
char out(idata char *bufor);
char status(idata char *bufor);
char on(idata char *bufor);
char off(idata char *bufor);
char help(idata char *bufor);


typedef struct                              //Definicja typu dla tablicy - wykazu poleceń
{
  char code *komenda;
  char (code *funkcja)(idata char*);
}komendy;

code komendy wykaz[] =                      //Wykaz komend i powiązanych z nimi funkcji
{
       "IN", in,
       "OUT", out,
       "STATUS", status,
       "ON", on,
       "OFF", off,
       "HELP", help,
       "?", help,
       "", NULL                             //Koniec wykazu
};
```

Listing 1. Fragment programu z definicjami zmiennych – wskaźników do funkcji.

Interpreter wykonuje następujące polecenia:

- IN <numer portu> np. IN 1 - odczyt portu o podanym numerze
- OUT <numer portu> <wartość> np. OUT 1 0x20 - zapisuje do portu o podanym numerze liczbę,
- STATUS – podaje informację o statusie WYŁĄCZONY/AKTYWNY,
- ON - załączenie operacji na portach tj. poleceń IN i OUT,
- OFF - wyłączenie operacji na portach tj. blokowanie funkcjonowania poleceń IN i OUT,
- HELP lub ? - informacja o realizowanych poleceniach.

Polecenia mogą być przesyłane przez dowolny telefon komórkowy oraz niektóre aplikacje wysyłające polecenia za pomocą bramki SMS z wykorzystaniem sieci Internet. Program skompilowany został z użyciem RC-51 firmy Raisonance.

Początek kodu źródłowego to właściwe dla RC-51 polecenie *#pragma DEFJ(TIM1_INIT=0xFE)* definiujące wartość zapisywaną do rejestru TH1 Timera 1 sterującego pracą UART. Dla rezonatora 7,3728MHz oraz podwójnej szybkości zegara sterującego pracą interfejsu szeregowego (SMOD = 1), zapis do TH1 wartości 0xFE wymusza transmisję szeregową asynchroniczną z prędkością 19200 bps. W następnej kolejności dołączane są definicje rejestrów mikrokontrolera, biblioteka funkcji wejścia – wyjścia oraz dla czytelności programu zdefiniowany zostaje typ WORD. W kolejnym kroku definiowane są nagłówki funkcji będących

odpowiednikami poleceń realizowanych przez interpreter. Zdefiniowanie ich w tym miejscu jest konieczne, ponieważ ze moment nazwy funkcji będą użyte do konstrukcji tablicy – wykazu poleceń.
Na listingu 2 umieściłem fragment programu rozpoznający polecenie oraz wywołujący odpowiadającą mu funkcję. Dwie zmienne *j* oraz *i* służą (odpowiednio) jako indeksy poszczególnych liter, przekazywanego jako argument funkcji, ciągu znaków oraz poszczególnych linii tablicy – wykazu poleceń. Funkcja XOR (^) służy do sprawdzenia warunku równości znaków a bitowe AND (&) maskuje bity odpowiadające małym literom alfabetu. Dzięki temu wielkość znaków (liter) odebranych z UART nie wpływa na interpretację polecenia.

```
//wyszukiwanie komend oraz wywołanie odpowiadających im funkcji
char command(char data *bufor)
{
	char i, j;                                  //256 komend o maks. długości 256 znaków

	for (i = 0;;)
		for (j = 0;; )
		{
			if(wykaz[i].komenda[j] != 0)  //jeśli komenda różna od znaku "pustego"
			{                                   //do porównania zamiana liter na duże
				if(((wykaz[i].komenda[j]^bufor[j]) & 0x5F) == 0)
					{
					j++;
					continue;      //następny znak
				}
				i++;
				break;                 //następna komenda
		}
		if( j == 0 )
		{                                       //brak komendy w wykazie
			printf("%s\n","BLAD: Nie rozpoznano komendy!");
			return(0);
		} else
		return (wykaz[i].funkcja(bufor+j));  //wykonanie funkcji spod wskazanego adresu
	}
```

Listing 2. Fragment programu odpowiedzialny za rozpoznawanie odbieranych poleceń.

W przypadku, gdy komenda nie zostanie odnaleziona w wykazie, wartość indeksu *j* będzie równa 0 i posłuży do wysłania komunikatu o błędzie. W innym przypadku zwracany jest wskaźnik do funkcji, którego przekształcenie do typu *char* owocuje wywołaniem funkcji.
Główna pętla programu zawiera tylko kilka poleceń: ustawia bit SMOD w rejestrze PCON mikrokontrolera, inicjuje telefon ustawiając sposób powiadamiania oraz wprowadzając do niego numer PIN, wywołuje funkcję *receivesms* odbierającą komunikat SMS po powiadomieniu a następnie interpretuje polecenie uruchamiając funkcję *command* jako parametr podając adres zdekodowanego ciągu łańcucha znaków właściwego odebranemu komunikatowi SMS.

Odbiór wiadomości SMS.

Do odbioru wiadomości SMS służy funkcja *receivesms(generic char *tekst).* Wyodrębnia ona fragment komunikatu o nazwie „Dane Użytkownika" i dokonuje jego konwersji na zwykły tekst. Pozostałe składniki komunikatu są odrzucane. Na listingu 3 umieszczono funkcję oraz pomocniczą dla niej *str2num*, zajmijmy się opisem sposobu ich działania.

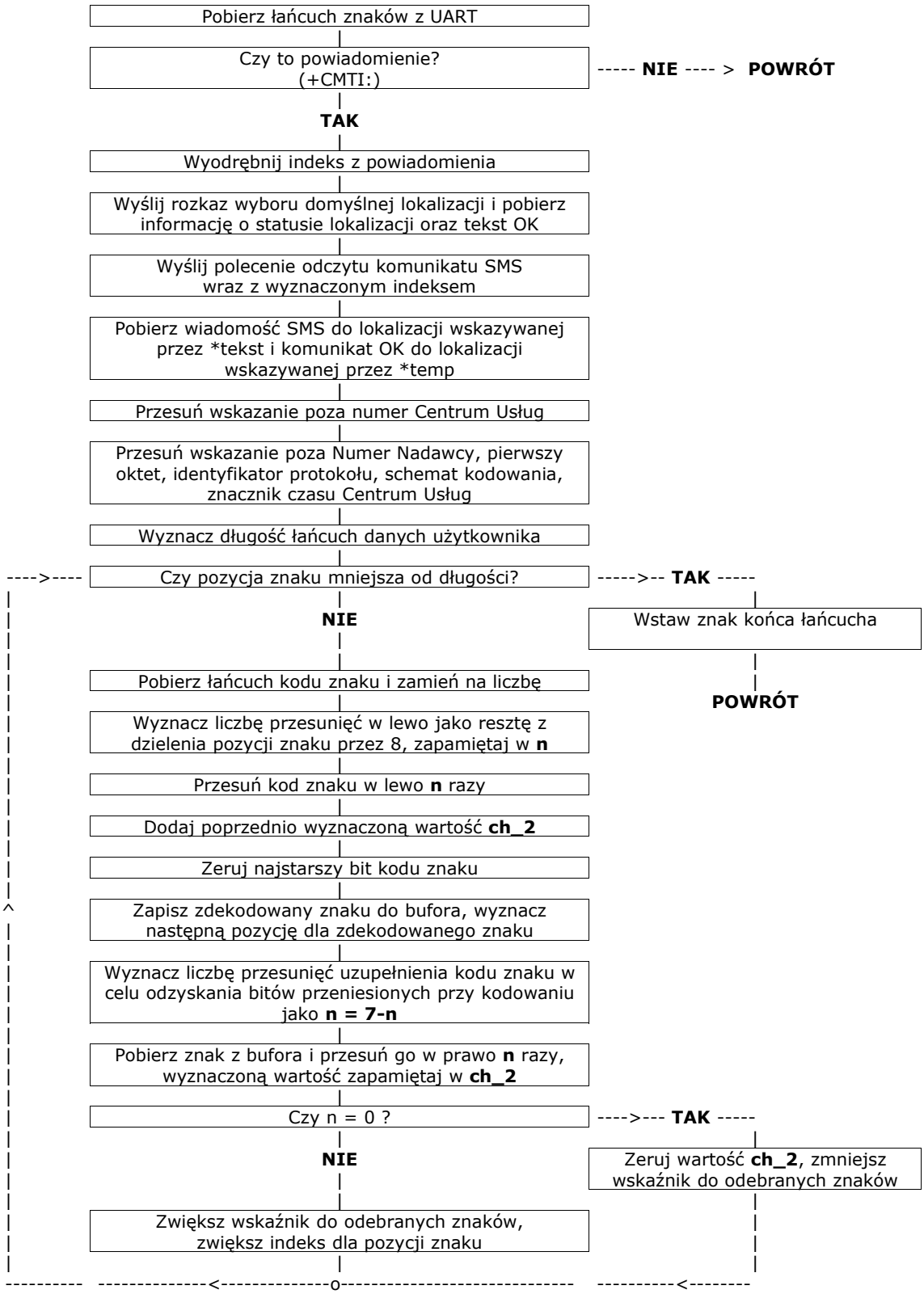
```
//zamiana dwubajtowej liczby – łańcucha znaków na typ unsigned int
WORD str2num(generic char *addr)
{
	data char temp[2];
	WORD num;

	temp[0] = *addr;
	temp[1] = *(addr+1);
	sscanf(temp, "%X", &num);
	return (num);
}

//odbiór wiadomości SMS
void receivesms(generic char *tekst)
{
	WORD i;
	char ch_1, ch_2, n, pos;
	data char temp[20];
```

```
        generic char *ptr;

        ptr = tekst;                        //bufor do zapamiętywania tekstu - rezultatu
        gets(tekst);                        //oczekiwanie na powiadomienie zgodne z NOTICE
        if (strncmp(tekst, &NOTICE, 6) == 0)
        {
                sscanf((tekst+12), "%i", &i);       //wyodrębnienie indeksu w pamięci
                printf("%s%c%c\n",CHOOSEMEM,*(tekst+8),*(tekst+9)); //wybór pamięci z SMS
                gets(temp);                 //pobranie informacji o statusie CPMS
                gets(temp);                 //pobranie OK
                printf("%s%i\n", SMSREAD, i);
                gets(tekst);                //pobranie SMS
                gets(temp);                 //pobranie OK
                printf("%s%i\n", SMSDEL, i); //usunięcie SMS z pamięci telefonu
                gets(temp);                 //pobranie OK

                i = str2num(tekst);         //przesunięcie wskaźnika poza numer centrum usług
                tekst += 2 * i + 4;         //i pierwszy oktet

                i = str2num(tekst);         //przesunięcie wskaźnika poza pola: typ n-ru
                tekst += 22 + i + i % 2;    //nadawcy, nr nadawcy, identyfikator protokołu,
                                            //schemat kodowania, znacznik czasu

                i = str2num(tekst);         //wyznaczenie długości łańcucha
                tekst += 2;                 //wskazanie do pierwszej pozycji tekstu SMS

                while (pos < i)             //dopóki nie przekroczono rozmiaru łańcucha
                {
                        ch_1 = str2num(tekst);      //zamiana łańcucha na kod znaku
                        n = pos % 8;        //obliczenie liczby przesunięć dla 1-go znaku
                        ch_1 <<= n;         //przesunięcie 1-go znaku w lewo n razy
                        ch_1 += ch_2;       //maskowanie najstarszego bitu oraz uwzględnienie
                        ch_1 &= 0x7F;       // bitów przeniesienia (przy kodowaniu)
                        *ptr = ch_1;        //zapis wyliczonego kodu znaku do bufora
                        ptr++;              //następna pozycja w buforze
                        n = 7 - n;          //wyliczenie ilości przesunięć następnego znaku
                        ch_2 = str2num(tekst) >> n;  //dla odzyskania bitów „utraconych” przy
                                            //kodowaniu
                        if (n == 0)         //cyklicznie - 8 znak brany jest w całości
                        {
                                ch_2 = 0;
                                tekst -= 2;
                        }
                        pos++;              //następna pozycja w dekodowanym ciągu
                        tekst += 2;
                }
                *ptr = 0;                   //zapis znaku końca łańcucha
        }
}
```

Listing 3. Funkcja odbierająca i dekodująca komunikat SMS.

Pomocnicza funkcja *str2num* dokonuje konwersji dwuznakowej liczby szesnastkowej wyrażonej w formie łańcucha znaków (np. A0, F1, FF itp.) na jej reprezentację liczbową typu unsigned int (WORD). Konwersja wykonywana jest za pomocą standardowej funkcji bibliotecznej języka C – *sscanf*.
Każdy ciąg znaków przesyłany przez telefon GSM (również terminal GSM) do komputera PC zakończony jest przez sekwencję znaków CR – LF (0x0D – 0x0A). Znajdujące się na początku *receivesms* wywołanie funkcji *gets* oczekuje na transmisję z UART zakończoną właśnie tymi znakami. Po ich odebraniu, łańcuch znaków znajdujący się w buforze transmisji porównywany jest z wzorcem dla powiadomienia. W przypadku zgodności podejmowana jest akcja mająca na celu odebranie i zdekodowanie komunikatu.
Jako pierwszy przy pomocy funkcji *sscanf* wyodrębniany jest numer – indeks, będący częścią powiadomienia a informujący o numerze odebranego komunikatu w preferowanej lokalizacji pamięci. Następnie, poprzez zwykłe wskazanie, pobierana jest nazwa tej lokalizacji (*(tekst+8) oraz *(tekst+9)) i przesyłana do telefonu. Później odbierana jest informacja o statusie wybranej lokalizacji, jednak jest ona ignorowana a jej odbiór przeprowadzany jest wyłącznie dla zapewnienia poprawnej komunikacji. Następne polecenia odczytują i usuwają komunikat SMS z pamięci aparatu. Teraz funkcja przeprowadza dekodowanie SMS umieszczonego w buforze w pamięci RAM, wskazywanego przez argument **tekst*.
Pierwszy oktet komunikatu zawiera liczbę oktetów numeru Centrum Usług. Bezpośrednio za nim umieszczony jest numer typ numeru Centrum a następnie tenże numer. Za nimi umieszczony jest pojedynczy bajt, zwany Pierwszym Oktetem. Tak więc pobranie pierwszego oktetu komunikatu i wyznaczenie jego wartości liczbowej,

pomnożenie jej przez 2 oraz dodanie do niej 4 wyznaczy pozycję następnego, zmiennego parametru tj. numeru Nadawcy wiadomości.
Numer Nadawcy prezentowany jest podobnie jak numer Centrum Usług. To znaczy rozpoczyna go oktet zawierający jego długość, później umieszczany jest oktet typu numeru nadawcy a następnie sam numer. Niestety nie można użyć tu tej samej metody wyznaczania następnej pozycji komunikatu, ponieważ tym razem liczba zawierająca długość numeru Nadawcy wyraża liczbę cyfr numeru a nie liczbę oktetów potrzebną na jego zapamiętanie... Dalej prezentowane są parametry o stałej długości takie jak: identyfikator protokołu, schemat kodowania, znacznik czasu. W sumie to 22 bajty + długość numeru nadawcy.
Po uwzględnieniu wyżej opisanych parametrów oraz zwiększeniu o tę wartość wskaźnika do bufora odebranego komunikatu, zaadresowany zostaje oktet zawierający długość komunikatu użytkownika. Funkcja wyznacza tę wartość i używa jej jako warunku – granicy dla pracy pętli pobierającej znaki z bufora.
W pierwszym kroku pobrany z bufora znak w formie tekstu zamieniany jest na odpowiadającą mu reprezentację liczbową. Następnie, na bazie numeru znaku wyznaczana jest liczba operacji przesunięcia w lewo i zapamiętywana w zmiennej o nazwie *n.* Dla przykładu jeśli rozpatrywany jest pierwszy znak z bufora, to n=0, dla drugiego n=1 i tak dalej. Ponieważ liczba przesunięć jest resztą z dzielenia pozycji znaku przez 8, to wartość n jest cykliczna i powtarza z okresem równym 8 znaków. Do wyznaczonego w ten sposób kodu znaku dodawana jest poprzednio wyznaczona wartość *ch_2* będąca kodem poprzednio pobranego znaku przesuniętego w prawo o liczbę pozycji 7-n. Operacja przesunięcia umożliwia wyodrębnienie „utraconych" w procesie kodowania bitów kodu znaku. Dla każdego znaku, którego pozycja jest wielokrotnością liczby 7, wartość *ch_2* jest równa 0 a wskaźnik znaku jest przesuwany na poprzednią pozycję. Jest to konieczne z tego powodu, że zgodnie z zasadą kodowania SMS, każdy co 8 znak jest tracony.
Zdekodowane znaki wstawiane są na początek bufora wskazywanego przez **tekst* a przechowującego komunikat SMS. Tak więc w rezultacie zdekodowania wzorzec odebranego komunikatu jest niszczony. Po zakończeniu dekodowania, na końcu łańcucha, wstawiany jest znak o kodzie 0, informujący pozostałe funkcje języka C, korzystające ze zdekodowanego ciągu znaków, o końcu tekstu.


Pobierz łańcuch znaków z UART
Czy to powiadomienie? (+CMTI:)
----- NIE ---- > POWRÓT
TAK
Wyodrębnij indeks z powiadomienia
Wyślij rozkaz wyboru domyślnej lokalizacji i pobierz informację o statusie lokalizacji oraz tekst OK
Wyślij polecenie odczytu komunikatu SMS wraz z wyznaczonym indeksem
Pobierz wiadomość SMS do lokalizacji wskazywanej przez *tekst i komunikat OK do lokalizacji wskazywanej przez *temp
Przesuń wskazanie poza numer Centrum Usług
Przesuń wskazanie poza Numer Nadawcy, pierwszy oktet, identyfikator protokołu, schemat kodowania, znacznik czasu Centrum Usług
Wyznacz długość łańcuch danych użytkownika
Czy pozycja znaku mniejsza od długości?
-----TAK -----
NIE
Wstaw znak końca łańcucha
POWRÓT
Pobierz łańcuch kodu znaku i zamień na liczbę
Wyznacz liczbę przesunięć w lewo jako resztę z dzielenia pozycji znaku przez 8, zapamiętaj w n
Przesuń kod znaku w lewo n razy
Dodaj poprzednio wyznaczoną wartość ch_2
Zeruj najstarszy bit kodu znaku
Zapisz zdekodowany znaku do bufora, wyznacz następną pozycję dla zdekodowanego znaku
Wyznacz liczbę przesunięć uzupełnienia kodu znaku w celu odzyskania bitów przeniesionych przy kodowaniu jako n = 7-n
Pobierz znak z bufora i przesuń go w prawo n razy, wyznaczoną wartość zapamiętaj w ch_2
Czy n = 0 ?
---TAK -----
NIE
Zeruj wartość ch_2, zmniejsz wskaźnik do odebranych znaków
Zwiększ wskaźnik do odebranych znaków, zwiększ indeks dla pozycji znaku

Wysłanie wiadomości SMS.

Listing 4 zawiera wydruk funkcji *sendsms(generic char *tekst)* wysyłającej dowolny komunikat SMS pod stały numer telefonu GSM. W tym przykładzie programowania, zarówno Nadawca jak i Odbiorca są abonentami sieci Plus GSM.

```
//wysłanie wiadomości SMS
void sendsms(generic char *tekst)
{
      char len, i, n, ch_1, ch_2;

      n = len = strlen(tekst);            //obliczenie długości łańcucha znaków
      i = n / 8;                          //licząc od "pierwszego oktetu"
      n = i * 7 + i % 8 + 13;
      printf("%s%d\n", SMSSEND, n);
      while(_getkey() != '>');            //oczekiwanie na znak "zachęty"
      printf("%02X%02X%s%02X%02X%02X%02X%s%02X%02X%02X%02X", \\
               LOSCA, TOSCA, SCA, FO, MR, LODA, TODA, DA, PID, DCS, SCTS, len);
      i = 0;
      while (i < len)
      {
            ch_1 = *(tekst + i);          //pobranie pierwszego znaku tekstu
            ch_2 = *(tekst + i + 1);      //pobranie następującego za nim znaku
            n = i % 8;                    //obliczenie ilości przesunięć
            ch_1 >>= n;                   //przesunięcie pierwszego znaku w prawo
            n = 7-n;                      //wyliczenie liczby przesunięć dla 2-go znaku
            ch_2 <<= n;                   //przesunięcie drugiego znaku w lewo
            ch_1 += ch_2;                 //wyliczenie kodu nowo powstałego znaku
            printf("%02X", ch_1);         //wysłania znaku szesnastkowego
            i++;                          //następna pozycja w łańcuchu
            if (n == 1) i++;              //dla każdego 8-go znaku przesunięcie o 1,
                                          //znak jest "gubiony"
      }
      putchar(0x1A);                      //wysłanie znaku końca EOT+CR+LF
      putchar(0x0D);
}
```

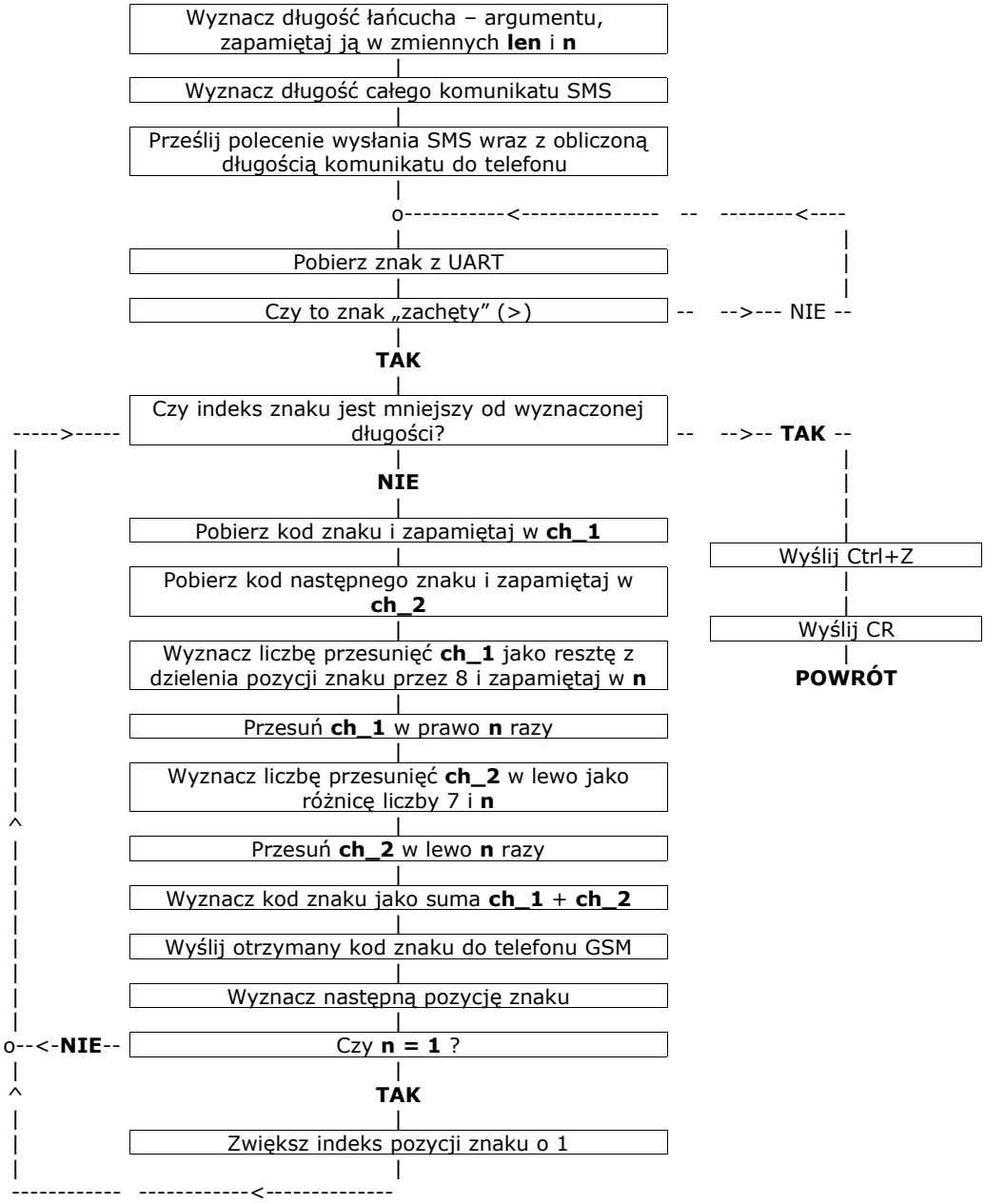
Listing 3. Funkcja kodująca i wysyłająca komunikat SMS.

Na początku wyznaczana jest długość łańcucha znaków podlegająca kodowaniu i zapamiętywana w zmiennych *n* i *len.* Ta pierwsza zmienna (n) wysyłana jest wraz poleceniem wysłania komunikatu SMS i w rezultacie, po kolejnych obliczeniach, będzie zawierać długość całego komunikatu. Pamiętajmy, że znaki o kodzie długości 8 bitów zamieniane zostają na 7 bitowe. Po przesłaniu polecenia wysyłki komunikatu SMS oraz wyznaczonej wartości n, funkcja oczekuje na znak „zachęty" po którym to może zostać wprowadzony komunikat SMS. Druga zmienna (len) używana jest jako wartość graniczna w pętli kodującej tekst oraz przekazywana jako długość pola Dane Użytkownika (UD).
Polecenie *printf* przesyła odpowiednio sformatowane, stałe składniki komunikatu:

- LOSCA: liczbę oktetów numeru Centrum Usług wraz z oktetem typu numeracji,
- TOSCA: typ numeru Centrum Usług; tu używana numeracja międzynarodowa, więc 91H
- SCA: numer Centrum Usług odpowiednio zakodowany, to jest:
  1) jeśli liczba cyfr jest nieparzysta, na końcu dodawana jest litera F,
  2) poszczególne znaki w obrębie jednego bajtu zamieniane są miejscami (połówki bajtu) dla przykładu: 48601000310 → 48601000310F → 8406010013F0
- FO: parametr Pierwszy Oktet, typowo ma wartość 11H dla wysyłanego komunikatu SMS
- MR: numer odniesienia dla komunikatu; komunikaty numerowane są automatycznie dla wieloczęściowych wiadomości SMS (MMS), należy wstawić wartość 0,
- LODA: długość pola Numeru Nadawcy,
- TODA: typ pola Numer Nadawcy,
- DA: pole Numer Nadawcy zakodowany identycznie jak numer Centrum Usług,
- PID: identyfikator protokołu, typowo 0 (SMS tekstowy),
- DCS: schemat kodowania, typowo 0 (alfabet domyślny, 7-bitowe znaki ASCII),
- SCTS: okres ważności komunikatu SMS,
- len: długość pola Dane Użytkownika.

W celu zakodowania zawartości pola Dane Użytkownika (treść komunikatu) funkcja pobiera kod znaku z łańcucha oraz następny w kolejności. Następnie wyznaczana jest liczba operacji przesunięć w prawo dla pierwszego znaku jako reszta z dzielenia numeru pozycji znaku przez 8: kod znaku pierwszego przesuwany jest w prawo o liczbę miejsc zgodną z wyznaczoną wartością. Teraz wyznaczana jest liczba przesunięć w lewo drugiego znaku jako różnica cyfry 7 i reszty z dzielenia numeru pozycji znaku przez 8: kod drugiego znaku

przesuwany jest w lewo o wyznaczoną liczbę miejsc. Na skutek zsumowania pierwszego i drugiego kodu uzyskuje się cyfrę właściwą dla zastosowanej metody kodowania. Cyfra ta, odpowiednio sformatowana, wysyłana jest przy pomocy instrukcji *printf* do dołączonego telefonu GSM.
Zgodnie z opisywaną wcześniej metodą kodowania, każdy co 8 znak jest „gubiony". Warunek *if* sprawdzający cykliczność liczby przesunięć, powoduje przemieszczenie indeksu na kolejny znak dla każdego co 8 w kolejności kodu znaku. Komunikat kończony jest przez wysłanie sekwencji Ctrl+Z – CR – LF. Komunikat wysyłany jest bezpośrednio po odebraniu tej sekwencji.



Przykłady kodowania wiadomości SMS w trybie PDU
1. Kodowanie wiadomości WITAJ! wysyłanej za pomocą aparatu pracującego w sieci Plus GSM na numer +48501102030, okres ważności 30 dni. Pierwsza wiadomość zostanie zakodowana w formie tabeli dla łatwiejszego opisu metody.

Opis parametru	Wartość (szesnastkowo)	Uwagi
Długość pola numeru Centrum Usług	**07**	Liczba oktetów numeru Centrum Usług z uwzględnieniem pola typu numeracji
Typ numeru Centrum Usług	**91**	Typ numeru Centrum Usług – dla numeracji międzynarodowej 91H
Numer Centrum Usług	**8406010013F0**	Numer Centrum Usług, tu podany dla operatora sieci Plus GSM (+48601000310)
Pierwszy oktet	**11**	Wartość 11H przy wywołaniu funkcji SMS-SUBMIT oraz relacyjny format okresu ważności SMS
Numer odniesienia	**00**	Pierwszy komunikat z wiadomości dzielonej lub pojedynczy komunikat
Długość numeru odbiorcy	**0B**	Liczba cyfr numeru odbiorcy
Typ numeru odbiorcy	**91**	Typ numeru odbiorcy – dla numeracji międzynarodowej 91H
Numer odbiorcy	**8405112030F0**	Numer odbiorcy zakodowany w identyczny sposób jak numer Centrum Usług
Identyfikator protokołu	**00**	Oznacza dostarczenie wiadomości jako normalnej wiadomości SMS
Schemat kodowania	**00**	Oznacza kodowanie domyślne w formacie 7 bitowych znaków ASCII
Okres ważności	**C4**	Okres ważności wiadomości SMS – 30 dni (format relatywny)
Długość pola danych użytkownika	**06**	Tekst użytkownika zawiera 5 oktetów (po konwersji septetów)
Dane użytkownika	**D72435A80C01**	Tekst komunikatu SMS: WITAJ!

Kompletny komunikat po zakodowaniu:
07918406010013F011000B918405112030F00000C405D72435A80C01

2. Kodowanie komunikatu WITAJ! wysyłanego na numer 605102030 z aparatu pracującego w sieci Plus GSM, okres ważności 12 godzin.

07 - długość pola Centrum Usług
91 - numeracja międzynarodowa
8406010013F0 - zakodowany numer Centrum Usług (+48601000310)
11 - pierwszy oktet komunikatu SMS
00 - numer odniesienia dla wiadomości SMS
09 - długość numeru odbiorcy SMS w cyfrach numeru
81 - typ numeru odbiorcy SMS
06152030F0 - zakodowany numer odbiorcy (605102030)
00 - identyfikator protokołu (zwykła wiadomość tekstowa SMS)
00 - schemat kodowania danych (domyślny, znaki 7-bitowe)
8F - okres ważności SMS 12 godzin
06 - liczba oktetów pola komunikatu
D72435A80C01 - zakodowany komunikat SMS

3. Kodowanie komunikatu WITAJ! wysyłanego na numer 605102030 z aparatu pracującego w sieci Plus GSM, okres ważności 12 godzin ale kodowanie 8 bitowe.

07 - długość pola Centrum Usług
91 - numeracja międzynarodowa
8406010013F0 - zakodowany numer Centrum Usług (+48601000310)
11 - pierwszy oktet komunikatu SMS
00 - numer odniesienia dla wiadomości SMS
09 - długość numeru odbiorcy SMS w cyfrach numeru
81 - typ numeru odbiorcy SMS
06152030F0 - zakodowany numer odbiorcy (605102030)

00 - identyfikator protokołu (zwykła wiadomość tekstowa SMS)
F6 - schemat kodowania danych - znaki 8 bitowe
8F - okres ważności SMS 12 godzin
06 - liczba oktetów pola komunikatu
574954414A21 - zakodowany komunikat SMS

4. Kodowanie komunikatu wieloczęściowego (3 wiadomości SMS) wysyłanego na numer 605102030 z aparatu pracującego w sieci Plus GSM, okres ważności 30 dni.

1-sza część komunikatu:
07 - długość pola Centrum Usług
91 - numeracja międzynarodowa
8406010013F0 - zakodowany numer Centrum Usług (+48601000310)
11 - pierwszy oktet komunikatu SMS
00 - numer odniesienia dla wiadomości SMS
09 - długość numeru odbiorcy SMS w cyfrach numeru
81 - typ numeru odbiorcy SMS
05112030F0 - zakodowany numer odbiorcy (605102030)
00 - identyfikator protokołu (zwykła wiadomość tekstowa SMS)
00 - schemat kodowania danych (domyślny, znaki 7-bitowe)
C4 - okres ważności SMS 30 dni
A0 - liczba oktetów pola komunikatu
D4B21B047F93E56F3D599F0EB341E232599F2E83E0F2B73807D297C5F2B03B1D06DDE7FAFC9CBE4E8FD1A0
FB585F9EBBD365F55C9F1FA34169B7F92D6F87C7EA34E80E82CBC36B7A7EAC77974170B93EBF6687C97990
3EBC7E93DFF7B03B3F4683EEE930F9DD7ECFC769D0B439752DD3EC7518047F93C3EEFC180DBA87E5E9B09B
FEBE83E06FFDFDCD4E8386

2-ga część komunikatu:
07
91
8406010013F0
11
00
09
81
05112030F0
00
00
C4
A0
FA3CBDCC76A7D7EF7B1AA47FCBD36537FD7D0F8F41F37419347E83C86FD03DDF0E9FC3EEF2F90D82CBF565
3D280C0FCBC374D071DA0499DFF276985E07DDD361F2BBFD9E8FD3A034885D86A7CB6A905EFED6D7DBE9F
218A40FCFC3E43C68FD26BFEF61773A0CBAA7C3E477FB3D1FA75DCEB7BC9D071DA74DD05B1ED683E6F0F7
FC2DCE83D66FF2FB1E76A7C3

3-cia część komunikatu:
07
91
8406010013F0
11
00
09
81
05112030F0
00
00
C4
32
20FA1B240ECBC9FA37E82D9EEBCB7277199406B9D3E539BD4CCF83E8F23AD95D06E9C3E730D99D2EBBD36
517

Uwaga: jak łatwo zauważyć, pole „numer odniesienia komunikatu” nie jest ustawiane, ponieważ oprogramowanie telefonu numeruje komunikaty samodzielnie.

Jacek Bogusz (jacek.bogusz@easy-soft.tsnet.pl)