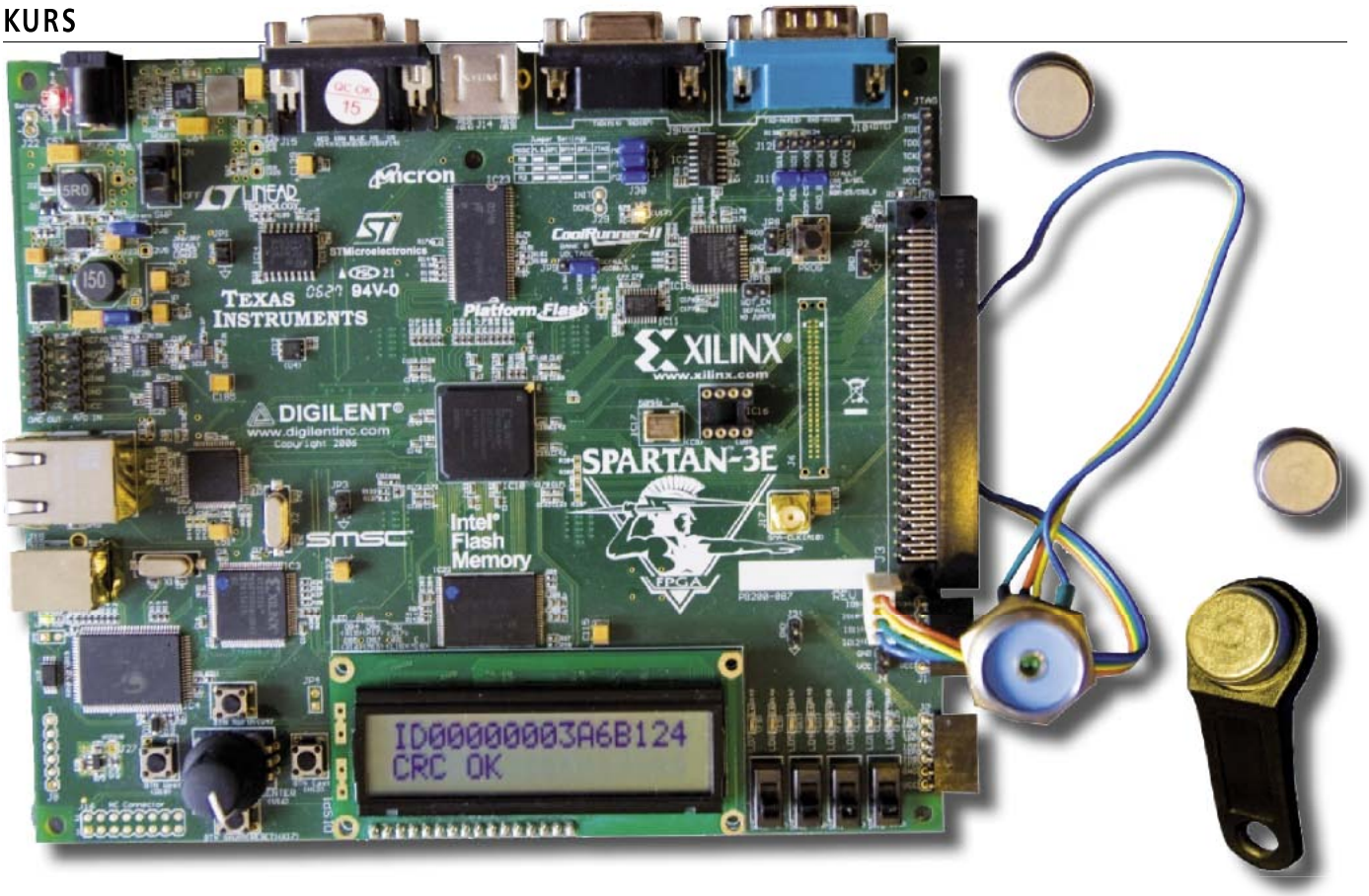


Język Verilog w przykładach (7)

Użycie procesora picoblaze do wysterowania wyświetlacza alfanumerycznego LCD

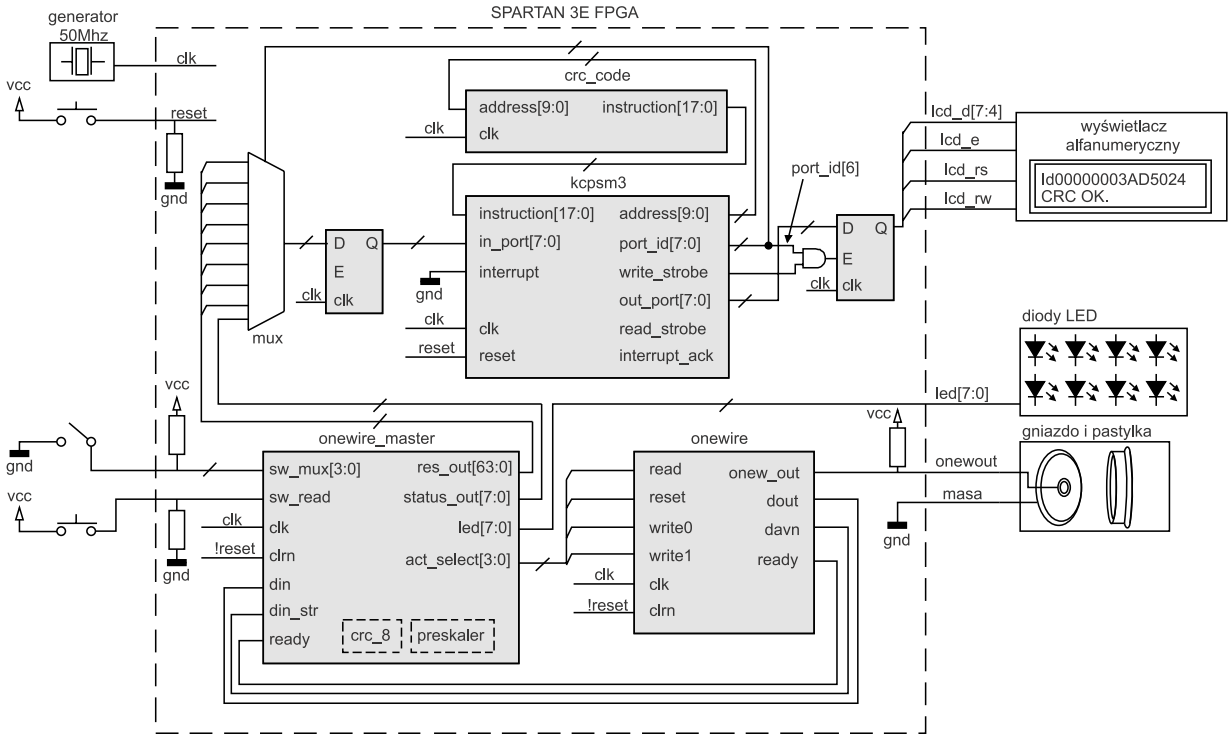
W artykule zaprezentowano sposób implementacji mikroprocesora picoblaze w układzie Spartan 3E. Jego użycie zilustrowano przykładem wysterowania wyświetlacza alfanumerycznego LCD wyświetlającego numer identyfikacyjny układu DS 1904, odczytany za pomocą interfejsu 1-Wire. Skupiono się przede wszystkim na specyfikacji użytego procesora, sposobie jego instancjonowania oraz wprowadzaniu pamięci z jego skompilowanym programem asemblerowym.

Dodatkowe materiały na CD i FTP:
host: ep.com.pl, user: 12235, pass: 60u61csy
- wszystkie części kursu w formacie PDF
- Kompletny projekt w Xilinx ISE oraz kod źródłowy programu dla procesora znajdują się na płycie CD dołączonej do tego numeru EP

Od redakcji:
Tym artykułem kończymy kurs języka Verilog w przykładach. Nie oznacza to, że nie będziemy publikowali artykułów z opisem projektów w tym języku, przeznaczonych do realizacji w układach programowalnych.

Projekt został zrealizowany w zestawie uruchomieniowym HW-SPAR3E-SK-UNI-G firmy Digilent, a więc na platformie Spartan 3E. Użyte pliki źródłowe procesora *picoblaze* są dostępne za darmo w Internecie, jednak istnieje kilka ich wersji. Wersja o nazwie KCPSM3 może być użyta w następujących układach FPGA: Spartan 3, Spartan 3E, Virtex II, Virtex II-Pro. Oczywiście istnieją wersje procesora także na inne układy, stąd wniosek, że kod *picoblaze'a* nie jest w pełni uniwersalny.

Zastosowany w projekcie mikroprocesor będzie odczytywał za pomocą interfejsu 1-Wire (z 64-bitowego rejestru) i wyświetlał kod ID układu na dwuwierszowym wyświetlaczu szesnastoznakowym LCD. Procesor będzie odczytywał w pętli zawartość tego re-



Rys. 7.1 Schemat blokowy układu z softprocesorem *picoblaze*

jestru i wyświetlał w pierwszym wierszu odczytany numer, a w drugim, w zależności od wskazań rejestru statusu, jeden z następujących komunikatów: CRC OK, CRC ERROR, NO PRESENCE.

Schemat blokowy opisywanego układu przedstawiono na **rys. 7.1**. Zamieszczone na nim bloki *onewire_master* i *onewire* zapewniają nadzór nad transmisję danych i realizację funkcji interfejsu 1-Wire.

Picoblaze jest bardzo prostym, 8-bitowym procesorem. W swojej strukturze ma szesnaście 8-bitowych rejestrów ogólnego przeznaczenia, dekoder instrukcji, kontroler przerwania, jednostkę arytmetyczno-logiczną oraz układy kontrolne. Brak tu akumulatora, ale każdy ze wspomnianych rejestrów może spełniać jego funkcję. Procesor wyposażony jest w wewnętrzną 64-bajtową pamięć danych. Jednostka ALU umożliwia przeprowadzenie podstawowych operacji logicznych, dodawania, odejmowania oraz porównania. Przestrzeń adresowa wejść/wyjść jest 8-bitowa. Procesor zajmuje około 5% zasobów układu Spartan XC3S200. Maksymalne częstotliwości sygnału zegarowego są zależne od typu układu i przykładowo wynoszą: 87,7 MHz w układzie Spartan 3 i 133,2 MHz w VirtexIIPro. Pamięć programu stanowi pojedynczy *blockRAM,* mogący pamiętać do 1024 instrukcji, z których każda jest wykonywana w dwóch cyklach zegarowych. Ułatwia to estymację czasu wykonywania poszczególnych funkcji. Lista rozkazów zawiera 57 instrukcji (**tab. 7.1**). Niestety nie zawiera operacji mnożenia i dzielenia arytmetycznego.

Tab. 7.1 Lista instrukcji procesora picoblaze

Kontrola programu	Arytmetyka	Logika	Przesunięcia

Skoki do etykiety
JUMP aaa
JUMP Z, aaa
JUMP NZ, aaa
JUMP C, aaa
JUMP NC, aaa
 	
Wywołanie podprogramu
CALL aaa
CALL Z, aaa
CALL NZ, aaa
CALL C, aaa
CALL NC, aaa
 	
Powrót z podprogramu
RETURN
RETURN Z
RETURN NZ
RETURN C
RETURN NC
*
	ADD sX, kk
	ADDCY sX, kk
	SUB sX,kk
	SUBCY sX,kk
	COMPARE sX, kk
	
	ADD sX, sY
	ADDCY sX, sY
	SUB sX, sY
	SUBCY sX, sY
	COMPARE sX, sY
		LOAD sX, kk
		AND sX, kk
		OR sX, kk
		XOR sX, kk
		TEST sX, kk
		
		LOAD sX, sY
		AND sX, sY
		OR sX, sY
		XOR sX, sY
		TEST sX, sY
			SR0 sX
			SR1 sX
			SRX sX
			SRA sX
			RR sX
			
			SL0 sX
			SL1 sX
			SLX sX
			SLA sX
			RL sX

	Przerwania	Zapis danych	Wejście/Wyjście
	RETURNI ENABLE RETURNI DISABLE ENABLE INTERRUPT DISABLE INTERRUPT	STORE sX, ss STORE sX, (sY) FETCH sX, ss FETCH sX, (sY)	INPUT sX, pp INPUT sX,(sY) OUTPUT sX, pp OUTPUT sX, (sY)

Legenda:
X, Y – definicja rejestrów ogólnego przeznaczenia s0...sF
kk – stała wartość w zakresie od 00 do FF
aaa – adres w zakresie od 000 do 3FF
pp – adres portu w zakresie od 00 do FF
ss – adres wewnętrznej pamięci w zakresie od 00 do 3F
* maksymalne zagnieżdżenie wynosi 31

W projektach, w których potrzebny jest dosyć złożony, ale niezbyt szybki układ sekwencyjny (mało inteligentny „kręcioł"), ten procesor sprawdza się znakomicie. Sterowanie alfanumerycznym wyświetlaczem LCD jest właśnie taką aplikacją. W praktyce ograniczenie długości programu do 1024 instrukcji nie jest poważnym problemem – zawsze istnieje możliwość rozdziału funkcji na więcej procesorów. Jeśli użyjemy np. dwóch procesorów, a kod dla obu z nich nie przekroczy w sumie rozmiaru jednego *blockRAMu*, można użyć tylko jednego bloku jako pamięci programu dla obu z nich.

Zajmijmy się implementacją procesora. Jest on opisany w kodzie źródłowym jego rdzenia *kcpsm3.v*. Do procesora musi być dołączona pamięć ROM z programem w postaci pliku otrzymanego po kompilacji programu asemblerowego działania procesora. Projektant musi również opisać porty wejścia i wyjścia. Bazą do ich zbudowania są sygnały we/wy procesora: *in_port[7:0], out_port[7:0], write_strobe, read_strobe, port_ID[7:0].* Port wejściowy realizuje się, opisując multiplekser, przy czym zalecane jest, aby pomiędzy multiplekserem a portem procesora znalazł się rejestr. Użycie sygnału strobującego nie

List 7.1 Opis portów mikroprocesora

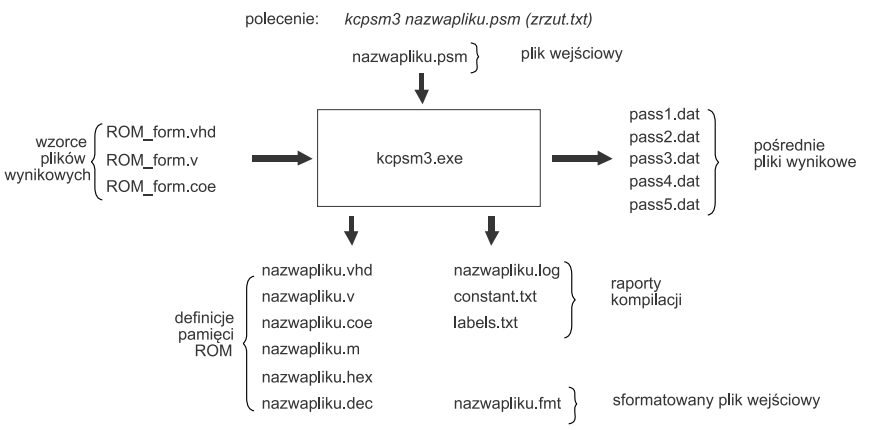
```
always@(posedge clk)
if (write_strobe)
  if (port_id[6]==1'b1) begin
    lcd_d <= out_port[7:4];
    lcd_rs<= out_port[2];
    lcd_e <= out_port[0];
  end

//input ports
always@(posedge clk)
  case(port_id[4:0])
  5'h01: in_port <= status;
  5'h02: in_port <= result[7:0];
  5'h03: in_port <= result[15:8];
  5'h04: in_port <= result[23:16];
  5'h05: in_port <= result[31:24];
  5'h06: in_port <= result[39:32];
  5'h07: in_port <= result[47:40];
  5'h08: in_port <= result[55:48];
  5'h09: in_port <= result[63:56];
  default: in_port <= 8'bX;
  endcase
```

List. 7.2 Inicjalizacja i główna pętla programu

```
      cold_start: CALL LCD_reset                    ;inicjalizacja LCD
                  ENABLE INTERRUPT

                  LOAD s5, 10             ;Ustaw kursor
                  CALL LCD_cursor
                  CALL disp_ep    ;Wyswietl „E.P. 1-wire"
                  LOAD s5, 20             ;ustaw kursor
                  CALL LCD_cursor
                  CALL disp_kasinski       ;Wyswietl „K.Kasinski 2008"

                  CALL delay_1s   ;Odczekaj 1s
                  CALL delay_1s
                  CALL delay_1s

 main_loop:       CALL delay_20ms ;Odczekaj 20ms
                  CALL delay_20ms
                  CALL delay_20ms
                  CALL delay_20ms
                  CALL delay_20ms

                  ;wyswietlanie gora
                  LOAD s5,10
                  CALL LCD_cursor
                  LOAD s5, character_I
                  CALL LCD_write_data
                  LOAD s5, character_D
                  CALL LCD_write_data
                  LOAD s5, 12
                  CALL LCD_cursor
                  INPUT s0,result7
                  CALL disp_hex_byte
                  INPUT s0,result6
                  CALL disp_hex_byte
                  INPUT s0,result5
                  CALL disp_hex_byte
                  INPUT s0,result4
                  CALL disp_hex_byte
                  INPUT s0,result3
                  CALL disp_hex_byte
                  INPUT s0,result2
                  CALL disp_hex_byte
                  INPUT s0,result1
                  CALL disp_hex_byte

    ;sprawdzenie rejestru statusu

                           LOAD s5,20
                           CALL LCD_cursor
                           CALL clear_row
                  INPUT s0,status_port
                  TEST s0,20
                  JUMP NC,crc_check
    presence_err: LOAD s5,20
                           CALL LCD_cursor
                           CALL disp_PRES_NO
                           JUMP continue
    crc_check:     TEST s0,40
                           JUMP NC,crc_ok
    crc_error:    LOAD s5,20
                           CALL LCD_cursor
                           CALL disp_CRC_ERR
                           JUMP continue
    crc_ok:       LOAD s5,20
                           CALL LCD_cursor
                           CALL disp_CRC_OK
                           JUMP continue
    continue:      JUMP main_loop
```

jest konieczne. Ma on zastosowanie w sytuacji, gdy moduł wystawiający dane wymaga informacji o „przeczytaniu danych" np. FIFO. Ponieważ dane, które chcemy odczytywać, mają 72 bity, to należało podzielić je na 9 grup 8-bitowych zajmujących adresy od 0x01 do 0x09. Port wyjściowy tworzy się na bazie zestawu rejestrów, których wejścia zezwolenia na zapis są sterowane na podstawie adresu (*port_ID*) podawanego przez procesor. W tym projekcie dekoder adresu został uproszczony – istotne jest (kontrolowane) ustawienie siódmego bitu adresu. Na **list. 7.1** zawarto fragment kodu przedstawiający opis obu portów. Procesor umożliwia obsługę przerwania. Ponieważ w tym przykładzie nie jest używane, należy dołączyć linię przerwania na stałe do masy.

Do pakietu KCPSM dołączony jest kompilator asemblera. Jest to niestety jedyny język programowania tego procesora. Kompilator stanowi plik wykonywalny KCPSM3.exe. Wymaga on trzech plików: *ROM_form.vhd, ROM_form.v oraz ROM_form.coe* (**rys. 7.2**). Są to wzorce bloków pamięci, wykorzystywane do wygenerowania plików źródłowych pamięci programu w *Verilogu* i *VHDL-u*. Plik kodu źródłowego powinien mieć rozszerzenie *psm*. Kompilator najłatwiej obsługuje się z poziomu okna poleceń. Poleceniem *kcpsm3 nazwapliku.psm* uruchamiamy kompilację. Po napotkaniu pierwszego błędu zostanie ona przerwana, a w oknie sygnalizowany będzie błąd. W trakcie kompilacji każda przeanalizowana linijka kodu jest wyświetlona. W ten sposób programista jest informowany, do której linii odnosi się komunikat o błędzie. W celu ułatwienia analizy wyników kompilacji można przy uruchomieniu kompilacji wskazać nazwę pliku tekstowego, do którego mają być zapisane wszystkie komunikaty wygenerowane przez kompilator. Po udanej kompilacji uzyskuje się pliki wynikowe noszące nazwę pliku wejściowego. Zbiory o rozszerzeniach *.vhd, v coe, hex, dec* i *m* zawierają definicje pamięci programu. Projektant wybiera plik odpowiadający jego projektowi i używa go jako źródłowego dla bloku pamięci. Użyjemy pliku z rozszerzeniem *v*, gdyż zawiera on opis w języku Verilog. Poza tym generowane są raporty asemblera (pliki: *.log, constant.txt, labels.txt*) oraz automatycznie sformatowana wersja wejściowego pliku asemblera (rozszerzenie *fmt*). Ponadto tworzonych jest pięć plików (pass1.dat, pass2.dat itd.) zawierających wyniki pośrednie, które mogą być przydatne w debugowaniu. Do projektu wybieramy wynikowy plik z rozszerzeniem *v* i instancjonu-



Rys. 7.2 Pliki wejściowe i wynikowe kompilatora kcpsm3

List. 7.3 Przykładowa funkcja wyświetlająca stały element tekstowy

```
disp_CRC_OK:  LOAD s5, character_C
              CALL LCD_write_data
              LOAD s5, character_R
              CALL LCD_write_data
              LOAD s5, character_C
              CALL LCD_write_data
              CALL disp_space
              LOAD s5, character_O
              CALL LCD_write_data
              LOAD s5, character_K
              CALL LCD_write_data
              CALL disp_space
              RETURN
```

List. 7.4 Funkcja odliczająca czas 1 µs dla zegara 50 MHz

```
delay_1us:       LOAD s0, 0B
czekaj_1us:      SUB s0, 01
JUMP NZ, czekaj_1us
RETURN
```

jemy jego zawartość jako pamięć programu procesora.

Xilinx na swoich stronach internetowych udostępnia wiele przykładowych kodów źródłowych wykorzystujących procesor *picoblaze* i bazujących na płytce uruchomieniowej użytej w projekcie opisanym w tym artykule (*http://www.xilinx.com/products/devkits/HW-SPAR3E-SK-US-G.htm*). Autor wykorzystał funkcje związane z obsługą wyświetlacza LCD. Najważniejsze z nich to: *LCD_reset, LCD_cursor, disp_hex_byte, LCD_write_data*. Przed jakąkolwiek akcją związaną z wyświetlaczem należy przeprowadzić jego inicjalizację. Funkcja *LCD_reset* wykonuje wymagane wpisy i czyści ekran. Następnie jest wyświetlamy ekran powitalny „E.P. 1 wire" „K. Kasinski 2008". Każde wyświetlenie danych składa się z ustawienia kursora w odpowiednim miejscu. Na przykład wpisanie wartości 20 do rejestru s5 i wywołanie funkcji *LCD_cursor* ustawi kursor na pierwszej pozycji drugiej linii.

Możemy już wyświetlać znaki. Dzięki utworzonym w programie stałym odpowiadającym wartościom odpowiednich znaków ASCII (np. CONSTANT *character_K, 4B*) tworzenie napisów jest intuicyjne. Do rejestru s5 wpisujemy poprzez nazwę stałej kod ASCII znaku, który chcemy wyświetlić i wywołujemy funkcję *LCD_write_data*. Aby uporządkować strukturę programu, utworzono funkcje wyświetlające stałe elementy (*disp_ep, disp_CRC_OK* itp. – **list. 7.3**).

Po wyświetleniu ekranu powitalnego program odczekuje 3 sekundy. Funkcja opóźnienia jest zrealizowana przez odliczanie do zera od odpowiedniej wartości (**list. 7.4**). Następnie program wchodzi w nieskończoną pętlę. Co 100 ms odświeża stan wyświetlacza. Wyświetlenie numeru seryjnego odbywa się w siedmiu cyklach. Najpierw, wykorzystując instrukcję *INPUT,* do rejestru s0 jest wczytywanych 8 odpowiednich bitów numeru seryjnego. Następnie dzięki funkcji *disp_hex_byte* ta 8-bitowa wartość jest wyświetlana w formie heksadecymalnej. Po wyświetleniu całego numeru seryjnego, do rejestru s0 ładowana jest wartość rejestru statusu. Po ustawieniu kursora w drugiej linii i wyczyszczeniu jej za pomocą instrukcji *TEST* sprawdzany jest stan szóstego bitu tego rejestru, informujący o braku impulsu obecności. Jeśli bit ten jest ustawiony, wywoływana jest funkcja wyświetlająca komunikat „NO PRESENCE". W przeciwnym wypadku sprawdzany jest dodatkowo stan siódmego bitu. Jeśli jest on ustawiony, to oznacza, że wystąpiła niezgodność kodów CRC – wyświetlany jest zatem komunikat „CRC ERROR". Jeśli wszystko jest w porządku, użytkownik odczyta komunikat „CRC OK", a program powróci do początku pętli. Jeśli wstąpił któryś z błędów, wyświetlany numer seryjny przyjmuje postać: „00000000DEADBEEF". Warto zapamiętać sobie tę wartość szesnastkową – często wykorzystuje się ją jako wskaźnik błędu. Powód jest prosty – często trudno jest zidentyfikować błędne dane, a reprezentowany w postaci szesnastkowej napis od razu rzuca się w oczy.

Krzysztof Kasiński
krzysztof.kasinski@o2.pl
http://home.agh.edu.pl/kasinski