POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRONIKI I TELEKOMUNIKACJI

Maciej Trawka

SPRZĘTOWA DEKOMPRESJA DANYCH TESTOWYCH NA BAZIE MIKROKONTROLERA AVR

INŻYNIERSKA PRACA DYPLOMOWA

Promotor
Prof. dr hab. inż. Jerzy Tyszer

Poznań 2012

Streszczenie

W pracy podjęto próbę wykonania i zbadania mikroprocesorowej implementacji układu dekompresującego dane, dostarczane przez tester do testowanego układu cyfrowego. Praca obejmuje projekt i wykonanie sprzętowego środowiska eksperymentalnego sterowanego kontrolerami z rdzeniem AVR, a także opracowanie efektywnych algorytmów programowej emulacji poszczególnych podukładów dekompresora trzeciej generacji i ich implementacja w pamięci kontrolerów środowiska testowego.

Abstract

This thesis covers a design of hardware environment for software implementations of third degree decompressor and proposes an algorithm for software emulation of the decompressor. The environment consists of AVR microcontrollers and PC software to control decompression experiments. I have attempted to demonstrate, that such a program implementation is relatively efficient.

Spis treści

1. Wstęp

1.1. Wprowadzenie

Od czasu skonstruowania pierwszego, działającego tranzystora ostrzowego minęło ponad pięćdziesiąt lat. Postępująca w szybkim tempie miniaturyzacja, oraz coraz tańsze metody wytwarzania elementów półprzewodnikowych, sprzyjają rosnącej złożoności współczesnych układów scalonych – analogowych i cyfrowych. Współczesne układy scalone to już nie pojedyncze bramki, czy proste bloki funkcjonalne, lecz skomplikowane, zawierające miliony bramek logicznych automaty, realizujące funkcje, których implementacja przy pomocy popularnych od lat układów serii 74 oraz 40 nie była i w dalszym ciągu nie jest możliwa, ze względu na ograniczoną przestrzeń, jaka może być spożytkowana na pojedynczy egzemplarz urządzenia. Ponieważ każdy producent układów scalonych chciałby obniżać koszt produkcji tak, by zysk osiągnięty na sprzedaży własnych wyrobów był jak największy, musi wziąć pod uwagę nie tylko koszt materiału i technologii wytwarzania, ale także koszt testowania wytworzonego układu.

Test produkcyjny współczesnych układów monolitycznych wymaga poświęcenia dodatkowego miejsca na podłożu, na którym napylane są kolejne warstwy tworzące elektrycznie funkcjonalne elementy, przeznaczonego na dodatkowe bufory, rejestry i sprzęgi komunikacyjne służące testerowi do uruchomienia procedury testowej gotowego wyrobu. Nie można także pominąć aspektu czasu – straty producenta są tym większe, im więcej czasu wytworzone przez niego elementy poddawane są procedurze testowania. Ponadto bardzo niekorzystna jest sytuacja, gdy testowany układ wymaga zbyt dużej liczby linii do współpracy z testerem. Jeżeli wziąć także pod uwagę fakt, że koszt testu może stanowić nawet 40% całkowitego kosztu produkcji łatwo zauważyć, że każda zmiana pozwalająca choć w niewielkim stopniu zniwelować jedno ze wspomnianych ograniczeń może przyczynić się do spadku kosztu produkcji układów scalonych (zwłaszcza tych o wielkiej skali integracji), co przełoży się na niższy koszt wyrobu końcowego.

1.2. Cel i zakres pracy

W niniejszej pracy podjęto próbę wykazania, że możliwe jest zaprojektowanie układu sprzętowej dekompresji danych testowych współpracującego z dowolnym układem zrealizowanym za pomocą powszechnie dostępnych elementów. Implementacja dekompresora

wewnątrz zaawansowanego mikrokontrolera lub układu FPGA (ang. *Field Programmable Gate Array*) nie jest kosztowna, a niesie ze sobą wiele zalet – zmniejszenie liczby wyprowadzeń potrzebnych do połączenia testera z badanym układem i znaczne ograniczenie wymaganej do zgromadzenia wszystkich wektorów testowych pamięci stanowiącej element składowy testera.

Przedmiotem pracy jest zatem:

- zaprojektowanie i wykonanie sprzętowego środowiska, umożliwiającego zaimplementowanie struktury dekompresora działającego zgodnie z założeniami technologii TestKompress [1]
- napisanie oprogramowania działającego na komputerze klasy PC, umożliwiającego nadzór i śledzenie pracy eksperymentalnego środowiska sprzętowego,
- opracowanie i prezentacja zebranych wyników badań i pomiarów oraz wynikających z nich konkluzji.

2. Kompresja danych testowych

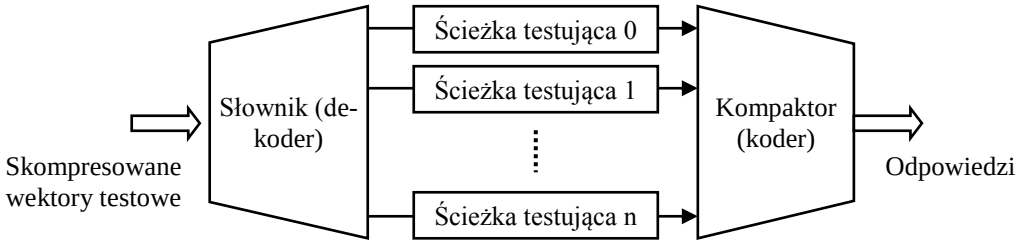
2.1. Metody kombinacyjne – pierwsza generacja

Najprostszą i historycznie pierwszą metodą kompresji jest kodowanie kombinacyjne, którego ideę przedstawia rysunek 2.1.. Wykorzystuje się tutaj kombinacyjny układ dekodujący słowa dostarczane poprzez kanały testera na dużo dłuższe ciągi, których każdy bit zostaje wsunięty do rejestru odpowiadającej jego położeniu w ciągu danych ścieżki testującej. Od konstrukcji dekodera zależy stopień kompresji i prawdopodobieństwo poprawnego zakodowania (skompresowania) wektorów testowych.

Najprostszy koder można zbudować łącząc równolegle wejścia kilku ścieżek testujących. Choć to proste rozwiązanie sprawdza się w wielu wypadkach, ma także wady. Najistotniejszą z nich jest fakt, że prawdopodobieństwo poprawnego zakodowania wymaganych wektorów testowych jest stosunkowo niskie, ponieważ zawartość rejestrów połączonych równolegle ścieżek testujących jest identyczna. Problem ten można wyeliminować, łącząc równoległe ścieżki testowe w szeregowy łańcuch, jednak pogarsza to współczynnik kompresji i znacznie wydłuża czas ładowania danych do tak utworzonych długich rejestrów przesuwających.

W celu zwiększenia prawdopodobieństwa zakodowania wektorów bez pogarszania współczynnika kompresji koder konstruuje się jako logikę ExOR o odpowiednio zaprojektowanej strukturze. W tym wypadku korelacja między ścieżkami testującymi jest mniejsza (choć nadal występuje – logika ExOR tworzy przesuwnik fazy), niż w przypadku równoległego połączenia ich wejść. W obydwu wspomnianych rozwiązaniach kodowanie wektorów polega na rozwiązaniu układu równań liniowych.

Kompresja kombinacyjna pozwala na zmniejszenie liczby kanałów testera, potrzebnych do przesłania wektorów testowych, w przybliżeniu dziesięciokrotnie – mówi się zatem o kompresji rzędu 10x.



Rys. 2.1. Idea kompresji kombinacyjnej

2.2. Metody sekwencyjne – druga generacja

Rozwinięciem kompresji kombinacyjnej jest poprzedzenie dekodera układem sekwencyjnym. Układ ten wprowadza dodatkowe zmienne stanu, umożliwiające zakodowanie wektorów za pomocą słów o mniejszej długości. W praktyce układem sekwencyjnym jest najczęściej rejestr liniowy LFSR (ang. *linear feedback shift register*). Wyjścia rejestru liniowego sterują wejściami kombinacyjnego dekodera, który z racji funkcji pełnionej w zmodyfikowanym układzie nazwany jest przesuwnikiem fazy.

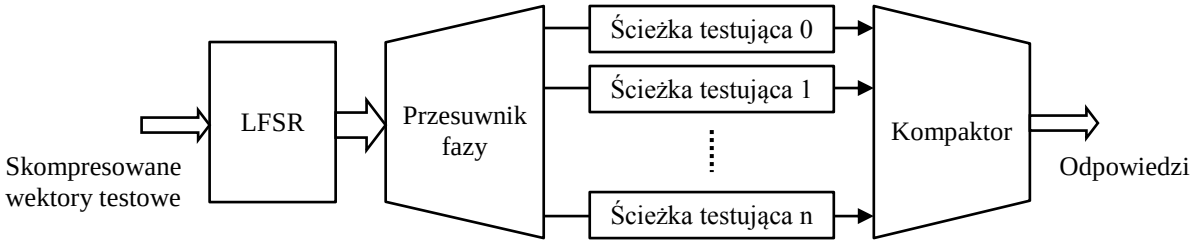
Ideę przedstawionego rozwiązania ilustruje rysunek 2.2. W początkowej fazie testu do rejestru liniowego wprowadzana jest, za pomocą kanałów testera, wartość początkowa (ang. *seed*). Po jej wprowadzeniu kanały testera nie są już wykorzystywane – przesuwająca się w takt kolejnych zboczy sygnału zegarowego zawartość rejestru LFSR, przeliczona w układzie przesuwnika fazy, jest wprowadzana do rejestrów ścieżek testujących.

Wprowadzony do dekompresora układ sekwencyjny zwiększa kompresję, dzięki zawartym w jego strukturze zmiennym stanu. Sumaryczny współczynnik kompresji całego układu (LFSR i przesuwnik fazy) to liczba rzędu 100 (mówi się zatem, że układ o opisanej architekturze cechuje się kompresją 100x). Wyższą skuteczność kodowania uzyskuje się zastępując rejestr liniowy generatorem pierścieniowym (ang. *ring generator*) [2].

Dalszy wzrost współczynnika kompresji osiągnięto wprowadzając do struktury rejestru liniowego (lub generatora pierścieniowego) dodatkowe zmienne – są nimi doprowadzone do wejść wybranych przerzutników kanały testera. Podczas wprowadzania danych do ścieżek testujących można teraz wpływać na stan rejestru LFSR poprzez zmianę stanu linii łączących tester z dekompresorem. Rozwiązanie to, znane jako wbudowany test deterministyczny (ang. *embedded deterministic test*), jest jedną z najczęściej stosowanych metod kompresji we współczesnych układach scalonych wielkiej skali integracji.

Kodowanie wektorów testowych, polegające – jak w przypadku kompresji kombinacyjnej – na rozwiązywaniu układu równań liniowych, wzbogacone zostało o dodatkową operację, zwaną sklejaniem kostek (ang. *cube merging*).. Polega ona na tym, że dla zakodowanego wektora testowego próbuje się znaleźć inny, którego wartości wyspecyfikowanych bitów nie są sprzeczne z wyspecyfikowanymi bitami wektora już zakodowanego. W przypadku znalezienia zgodnej pary nakłada się je (ang. *merging*) i dokonuje próby zakodowania tak złożonych wektorów. Jeśli próba zakończy się powodzeniem, znajduje się następny wektor, który nie będzie konfliktowy względem już sklejonych i podejmuje się próbę zakodowania nowego zbioru danych. Algorytm kodujący kończy się, gdy nie można

znaleźć kolejnego niekonfliktowego wektora, lub dodanie nowych danych uniemożliwi zakodowanie całości. Ponieważ w jednej sekwencji może być zakodowany więcej niż jeden wektor, współczynnik kompresji zostaje zwiększony.



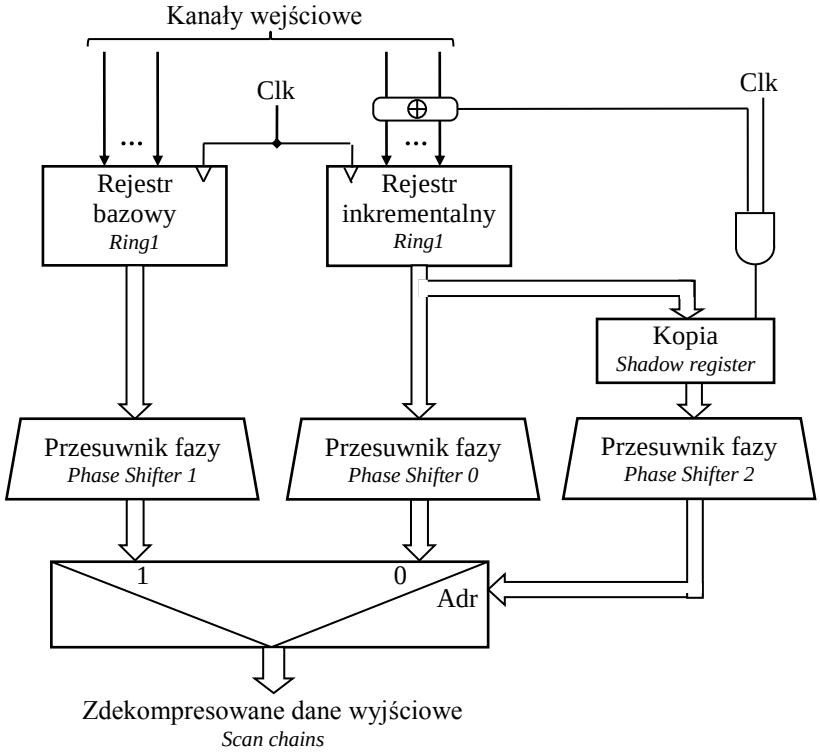
Rys. 2.2. Idea kompresji sekwencyjnej

.

2.3. Kompresja trzeciej generacji

Ponieważ dodawanie kolejnych szeregowych układów w torze danych nie pozwala zwiększyć współczynnika kompresji, skupiono się na rozszerzeniu koncepcji sklejania kostek. Rozszerzenie polega na umożliwieniu łączenia wektorów, których wyspecyfikowane bity pozostają w konflikcie. Łączenie to jest możliwe dzięki wzbogaceniu dekompresora o dwa dodatkowe rejestry: inkrementalny (ang. *incremental*) i kopiujący (ang. *shadow*), przechowujący kopię zawartości rejestru inkrementalnego przez określony czas. Rozwiązanie to zilustrowane jest na rysunku 2.3. Zawartość rejestru kopiującego ma decydujący wpływ na to, z którego źródła będą pobierane dane wsuwane do każdej ze ścieżek testujących – źródłem zasadniczym jest zawartość rejestru bazowego (ang. *parent*), źródłem dodatkowym (zastosowanym w celu eliminacji bitów pozostających w konflikcie) - zawartość rejestru inkrementalnego.

Stopień kompresji osiągnięty dzięki tak rozbudowanej konfiguracji układu dekompresującego może osiągać wartość rzędu 1000x. Jest to obecnie najbardziej efektywna kompresja danych testowych.



Rys. 2.3. Struktura dekompresora trzeciej generacji

3. Środowisko do implementacji dekompresora

3.1. Założenia dotyczące funkcjonalności

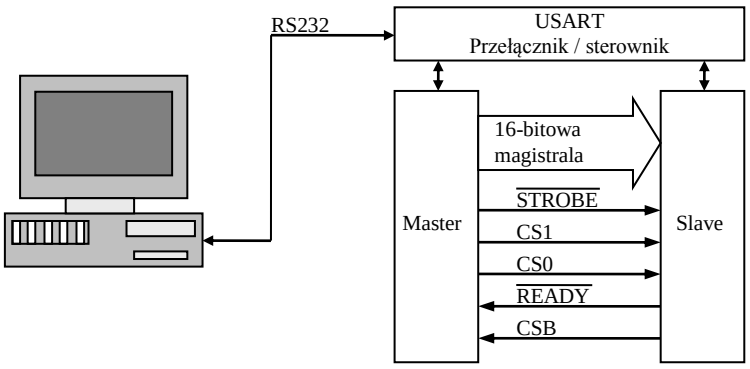
Zestaw umożliwiający implementacje dekompresora trzeciej generacji umożliwia dokładne testowanie i śledzenie każdego z wykorzystywanych przez dekompresor rejestrów oraz pozwala na dowolne wpływanie na ich zawartość. Umożliwia ponadto pomiar czasu dekompresji – parametr bardzo istotny, gdy struktura sprzętowa jest emulowane przez oprogramowanie. Czas dekompresji pozwoli ocenić efektywność i przydatność programowej implementacji.

Ważną kwestią jest zapewnienie niezależności środowiska testowego od innych układów, urządzeń, czy oprogramowania – zapewnia to względną obiektywność przeprowadzanych badań i eksperymentów.

3.2. Część sprzętowa

Na rysunku 3.1 pokazano blokową strukturę środowiska testowego. Składa się ono z dwóch mikrokontrolerów AVR Atmel Mega 32 [3], z których jeden – oznaczony jako „Master" – zapewnia bezpośrednią komunikację z komputerem oraz steruje drugim mikrokontrolerem, oznaczonym „Slave", pełniącym w układzie rolę dekompresora.

Kontrolery pracują z częstotliwością 16 MHz. Układy AVR wykonują jedną instrukcję w czasie jednego okresu sygnału taktującego. Sygnał zegarowy jest generowany przez wewnętrzny oscylator wbudowany w strukturę mikrokontrolerów, z zewnętrznym sprzężeniem w postaci rezonatora kwarcowego o względnie wysokiej stabilności, bez pojemności obciążających.



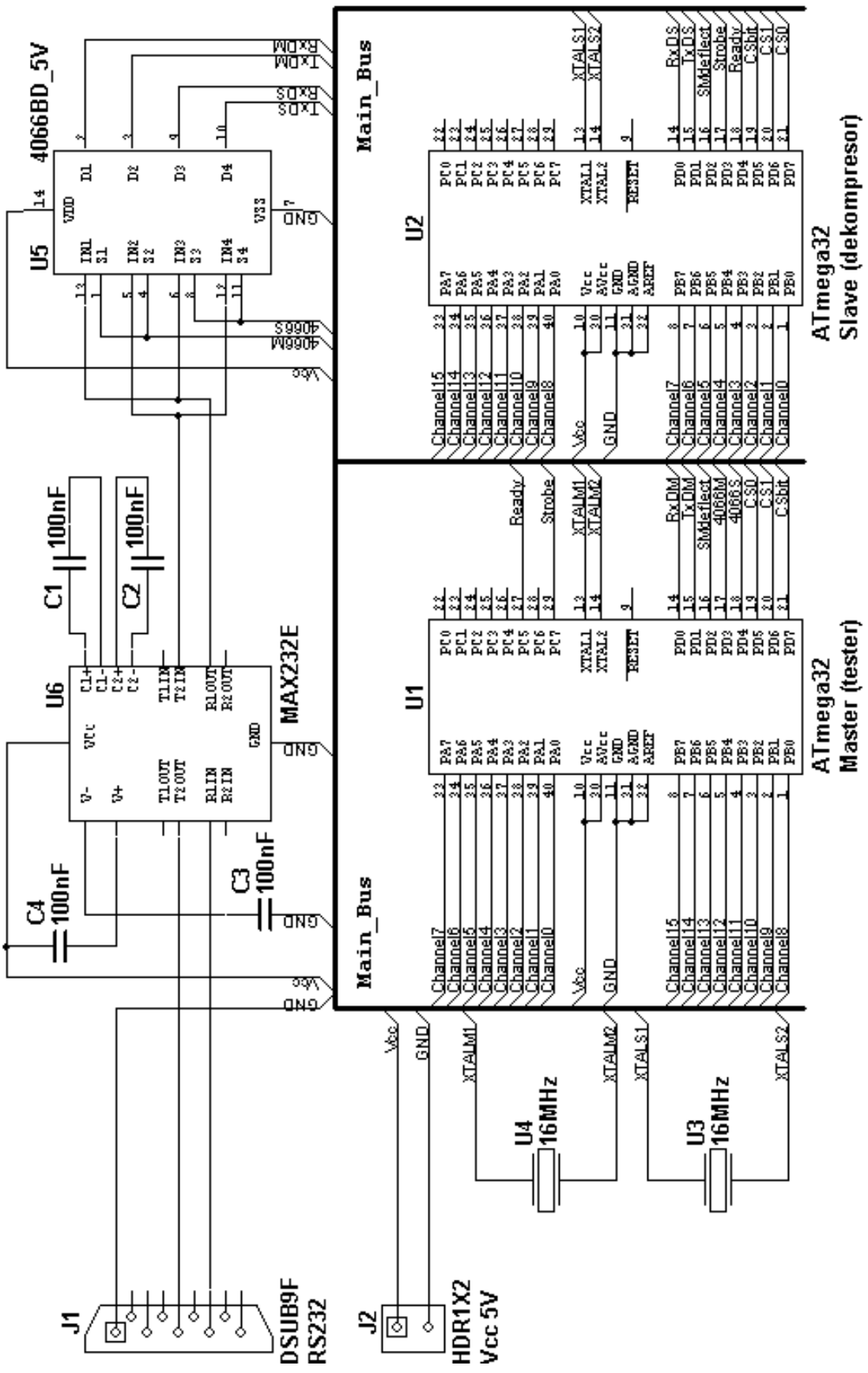
Rys. 3.1. Blokowa struktura środowiska testowego

Kontroler sterujący („Master”), za pomocą układu kluczującego, może połączyć komputer bezpośrednio z drugim – sterowanym kontrolerem dekompresującym, aby umożliwić szybki odczyt kontrolny (lub programowanie) zawartości rejestrów dekompresora. Obydwa mikrokontrolery połączone są między sobą 16-bitową magistralą danych oraz następującymi liniami sterującymi:

- STROBE – zbocze dodatnie podane przez kontroler sterujący wymusza przeprowadzenie pojedynczego cyklu dekompresji. Jest analogią do narastającego zbocza sygnału taktującego sprzętową strukturą dekompresora i układu sterującego testem,
- CS1, CS0 – wybór trybu pracy (dekompresja, dekompresja częściowa, programowanie),
- READY – zgłasza gotowość kontrolera sterowanego do wykonania kolejnego rozkazu,
- CSB – stan jednobitowej sumy kontrolnej zdekompresowanych danych w ostatniej szczelinie czasowej,

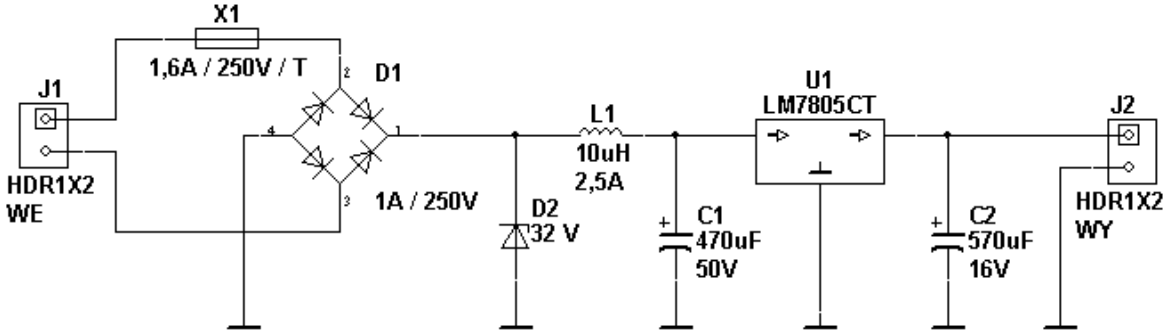
Oprócz wymienionych linii sterujących, związanych z logiką modułu badawczego, kontrolery łączą także inne linie, z których najważniejszą jest linia umożliwiająca mikrokontrolerowi sterowanemu wymuszenie na układzie sterującym komutację łącza RS232, którym środowisko testowe połączone jest z komputerem. Łącze to pracuje z szybkością 384 kbps, bez kontroli przepływu i parzystości, w trybie ośmiobitowym. Współczynnik wystąpień błędów na linii transmisyjnej jest pomijalnie mały i w żaden sposób nie wpłynie na wyniki badań i eksperymentów, dzięki odpowiednio zaprojektowanemu systemów komunikacji wymagającemu potwierdzenia odebrania rozkazu lub danych.

Blok zasilania części sprzętowej zrealizowany jest na bazie standardowego stabilizatora liniowego 7805 [4], z dodatkowo dołączonym szeregowym dławikiem oraz bezpiecznikiem topikowym, a także mostkiem prostowniczym z pojemnościowym filtrem na wejściu. Schemat zasilacza przedstawiony jest na rysunku 3.3. Umieszczony w stopniu wejściowym bezpiecznik topikowy X1 współpracuje z diodą Zenera D2 i stanowi ochronę stabilizatora U1 przed skutkami przepięcia. Gdy wartość napięcia wejściowego przekroczy 33V, wówczas przez diodę D2 przepłynie prąd o znacznej wartości, który powinien spowodować przepalenie drutu bezpiecznika. Pojemności kondensatorów C1 i C2 zostały dobrane według zależności: $C = k * I$, gdzie k to empiryczny współczynnik o wartości 1000μF/A, a I jest wartością maksymalną prądu pobieranego przez obciążenie [5].



Rys. 3.2. Zasadniczy schemat elektryczny środowiska testowego

Ponadto na płytce bazowej rozmieszczone są dodatkowe pojemności – dwa kondensatory tantalowe 10μF i cztery ceramiczne 100nF, których zadaniem jest eliminacja wysokoczęstotliwościowych składowych sieci energetycznej propagowanych przez stabilizator



Rys. 3.3. Schemat zasadniczy bloku zasilania.

oraz zapobieganie spadkom napięcia wywołanym przez przeciwsobne (ang. *push-pull*) stopnie pośrednie układów cyfrowych podczas przełączeń. Energia do zasilacza dostarczana jest za pośrednictwem wtyczkowego transformatora sieciowego 12V lub sieciowej przetwornicy impulsowej z izolacją galwaniczną, o wyjściowym napięciu z zakresu 7 – 24V.

Układ został wykonany na płytce uniwersalnej z metalizowanymi otworami tworzącymi raster zgodny z elementami w obudowach DIP. Fotografie gotowego układu przedstawiono w dodatku B.

3.3. Część programowa

Najważniejszą częścią zestawu badawczego jest jego wewnętrzne oprogramowanie sterujące, oraz oprogramowanie emulujące strukturę dekompresora. Program zapisany w pamięci mikrokontrolera sterującego („Master") zapewnia wsparcie dla szybkiej, bezbłędnej transmisji pomiędzy komputerem PC a dekompresorem. Pełni ono funkcje podobne do tych, które spełnia tester, komunikujący się z układem testowanym – ustawia na konfigurowalnej, równoległej magistrali dane podlegające dekompresji, steruje liniami sygnalizacyjnymi, oraz generuje sygnał podstawy czasu. W pamięci kontrolera „Slave" zaprogramowana została pełna struktura dekompresora wraz z interfejsem umożliwiającym śledzenie oraz ładowanie nowych zawartości poszczególnych generatorów pierścieniowych. Oprogramowanie wewnętrzne obydwu mikrokontrolerów zostało napisane w środowisku Bascom AVR firmy MCS Electronics [6] którego zaletą jest to, że zapewnia programiście dostęp do wewnętrznych rejestrów sprzętowych za pomocą instrukcji asemblera oraz róż-

nego rodzaju odwołani, a wygenerowany kod maszynowy zajmuje mniej miejsca w pamięci i jest wykonywany szybciej, niż kod wynikowy kompilatora języka C.

3.4. Interfejs komunikacyjny

Komunikacja ze środowiskiem testowym odbywa się poprzez wysyłanie dwubajtowych sekwencji i oczekiwaniu na potwierdzenie ich odbioru, poprzez łącze RS232, pracujące w konfiguracji *38400,8,n,1* (38,4 kbps, 8 bitowe słowa, brak bitu parzystości, jeden bit przerwy). Na sekwencje te składają się: bajt rozkazu i bajt danych (argument rozkazu). Argument musi zostać przesłany nawet wówczas, gdy nie jest to wymagane do wykonania instrukcji. Jedynym wyjątkiem jest rozkaz RESET, który służy do zerowania flagi oczekiwania na argument. Jego przesłanie gwarantuje, że kolejna przesłana instrukcja nie zostanie rozpoznana jako argument.

Każdy z kontrolerów posiada swój jednobajtowy identyfikator, którego odczyt za pomocą odpowiedniego rozkazu pozwala sprawdzić, który z kontrolerów jest aktualnie połączony z liniami transmisyjnymi RS232. Rozkazy sterujące zostały zebrane w tabelach 3.1 i 3.2.

Tabela 3.1. Zestawienie rozkazów rozpoznawanych przez kontroler sterujący (*Master*)

inst..	Rozkaz	Argument	Opis
255	INSTR_ARG_RESET	(brak)	Resetuje flagę oczekiwania na argument. Przesłanie tego rozkazu daje pewność, że kolejna wysłana instrukcja nie będzie błędnie rozpoznana jako argument.
0	GET_DEVICE_ID	(dowolny)	Powoduje odesłanie identyfikatora urządzenia (11)
1	GET_MODULE_ID	(dowolny)	Powoduje odesłanie identyfikatora modułu Master (12)
2	RS232_FORWARD	<bajt>	Powoduje przełączenie łącza RS232 do modułu Slave
3	RS232_TEST	<bajt>	Test łącza. Odsyła bajt zwiększony arytmetycznie o 1
4	GET_STATUS	(dowolny)	Odsyła bajt statusu. Bit0 = stan linii READY
5	SET_PX	<bajt>	Wystawia <bajt> na port PX
6	SET_PY	<bajt>	Wystawia <bajt> na port PY
7	SEND_TO_SLAVE	<bajt>	Ustawia bity kontrolne CS oraz generuje impuls strobujący zgodnie z argumentem

Tabela 3.2. Zestawienie rozkazów rozpoznawanych przez kontroler dekompresujący (*Slave*).

inst..	Rozkaz	Argument	Opis
255	INSTR_ARG_RESET	(brak)	Resetuje flagę oczekiwania na argument. Przesłanie tego rozkazu daje pewność, że kolejna wysłana instrukcja nie będzie błędnie rozpoznana jako argument.
0	GET_DEVICE_ID	(dowolny)	Powoduje odesłanie identyfikatora urządzenia (11)
1	GET_MODULE_ID	(dowolny)	Powoduje odesłanie identyfikatora modułu Slave (13)
2	RS232_FORWARD	<bajt>	Powoduje przełączenie łącza RS232 do modułu Slave
3	RS232_TEST	<bajt>	Test łącza. Odsyła bajt zwiększony arytmetycznie o 1
4	GET_PX	(dowolny)	Zwraca stan portu PX
5	GET_PY	(dowolny)	Zwraca stan portu PY
6	GET_RING0	(bajt)	Zwraca słowo o numerze (bajt) Ringa0
7	GET_RING1	(bajt)	Zwraca słowo o numerze (bajt) Ringa1
8	GET_INCR_PTRN	(bajt)	Zwraca słowo o numerze (bajt) rejestru inkrementalnego (Incremental Pattern)
9	RESET_REGISTERS	(dowolny)	Resetuje rejestry dekompresora
10	GET_SHADOW	(bajt)	Zwraca słowo o numerze (bajt) rejestru kopiującego (Shadow Register)
11	GET_CINFIG	(bajt)	Zwraca (bajt) konfiguracji.
12	GET_PARENT_PTRN	(bajt)	Zwraca słowo o numerze (bajt) rejestru bazowego (Parent Pattern)
13	GET_CONTROL_PTRN	(bajt)	Zwraca słowo o numerze (bajt) rejestru sterującego (Control Pattern)
14	GET_SCANS	(bajt)	Zwraca słowo o numerze (bajt) SCANS
15	GET_CHECKSUM	(dowolny)	Zwraca bajt sumy kontrolnej SCANS
16	INPUT_BYTE_RING0	(bajt)	Przesuwa Ring0 o bajt w lewo, potem najmłodszy bajt Ring0 zastępuje (bajt)'em.
17	INPUT_BYTE_RING1	(bajt)	Przesuwa Ring1 o bajt w lewo, potem najmłodszy bajt Ring1 zastępuje (bajt)'em.
18	INPUT_BYTE_SHADOW	(bajt)	Przesuwa Shadow o bajt w lewo, potem najmłodszy bajt Shadow zastępuje (bajt)'em.
19	INPUT_END	(dowolny)	Przeliczenie PS[0,1,2] i SCANS

Sterowanie transmisją odbywa się w trybie przerwaniowym. Po odebraniu bajtu danych przez kontroler, blok sterujący interfejsem UART ustawia flagę przerwania. W kolejnym cyklu maszynowym, jeśli układ przerwań jest aktywny, zawartość wskaźnika programu zostaje zapamiętana na stosie i zastąpiona przez wektor przerwania. Procedura realizowana po odebraniu danych z RS232 przedstawiona jest na rysunku 3.4. Na początku jest sprawdzana flaga oczekiwania na argument. Jeśli jest ustawiona, należy wykonać ostatnio przesłany rozkaz z nowo odebranym argumentem. W przeciwnym wypadku odebrany przez UART bajt należy uznać za rozkaz i zapisać go w buforze. Definicja procedury odpowiedzialnej za analizę i wykonanie przesłanego do kontrolera dekompresującego rozkazu przedstawiona jest na rysunku 3.5. Jest to prosta struktura *Select Case* – odpowiednik konstrukcji *switch* znanej z języka C/C++. Widoczne w niemal każdej linii instrukcje *Printbin* są wbudowanymi funkcjami języka Bascom Basic, służącymi do przesyłania podanego w ich argumencie bajtu (lub układu bajtów) poprzez moduł UART, który w tym układzie służy jako łącze RS232. Niektóre instrukcje wbudowane zostały jawnie zastąpione maszynowym kodem asemblera, w celu przyspieszenia ich działania – instrukcje te poprzedzone są znakiem wykrzyknienia lub dyrektywą *$asm*.

```
Rs232_int:
   If Rs232_recv_mode = 1 Then      'Jesli oczekiwanie na argument...
      Inputbin Rs232_arg            '- wczytaj argument,
      Reset Rs232_recv_mode         '- od teraz czekaj na nowy rozkaz
      Disable Urxc                  '- zablokuj przerwania od RS232
      Call Instr_anal               '- wykonaj rozkaz
      Enable Urxc                   '- odblokuj przerwania od RS232
   Else                          'Jeśli oczekiwanie na instrukcje...
      Inputbin Rs232_instr          '- pobierz rozkaz,
      Set Rs232_recv_mode           '- od teraz czekaj na argument
      If Rs232_instr = Instr_arg_reset Then '- ale jeśli rozkaz RESET,
         Reset Rs232_recv_mode      '- - to jednak czekaj na rozkaz.
      End If
   End If
   Return
```

Rys. 3.4. Podprogram obsługi przerwania od RS232

```
Sub Instr_anal()
   Select Case Rs232_instr
   Case Get_device_id : Printbin Device_id
   Case Get_module_id : Printbin Module_id
   Case Rs232_forward
      Set Rs232_forward_line
      Disable Int0
      Reset Rs232_forward_line
      Disable Urxc
      Delay
      Set Rs232_forward_line
      Config Rs232_forward_line = Input
      Bitwait Pind.2 , Set
      Enable Int0
   Case Rs232_test
      Incr Rs232_arg
      Printbin Rs232_arg
   Case Rs232_get_px : Printbin Px
   Case Rs232_get_py : Printbin Py
   Case Get_ring0 : Printbin Ring0
   Case Get_ring1 : Printbin Ring1
   Case Get_incr_ptrn :
      Longaux = Phase_shifter1(rs232_arg)
      Printbin Longaux
   Case Rs_reset_registers : Set Reset_regs
   Case Get_shadow : Printbin Shado
   Case Get_config :
      Aux = Lookup(rs232_arg , Config_data)
      Printbin Aux
   Case Get_parent:
      Longaux = Phase_shifter0(rs232_arg)
      Printbin Longaux
   Case Get_control:
      Longaux = Phase_shifter2(rs232_arg)
      Printbin Longaux
   Case Get_scans:
      Longaux = Scans(rs232_arg)
      Printbin Longaux
   Case Get_checksum : Printbin Scan_check_sum
   Case Input_ring0: Shift Ring0 , Left , 8
      Ring0lsb = Rs232_arg
   Case Input_ring1: Shift Ring1 , Left , 8
      Ring1lsb = Rs232_arg
   Case Input_shadow: Shift Shadow , Left , 8
      Shadowlsb = Rs232_arg
   Case Input_end : Disable Urxc
      Call Do_phase_shifter0
      Call Do_phase_shifter1
      Call Do_phase_shifter2
      For Aux = 1 To 4
         Not_phase_shifter2(aux) = Not Phase_shifter2(aux)
         Scans(aux) = Phase_shifter1(aux) And Not_phase_shifter2(aux)
         Scansaux(aux) = Phase_shifter0(aux) And Phase_shifter2(aux)
         Scans(aux) = Scans(aux) Or Scansaux(aux)
      Next Aux
      Call Xor_scans
      Enable Urxc
   Case Eeprom_write: Writeeeprom Rs232_arg , Eeprom_adres
   Case Eeprom_read: Readeeprom Rs232_arg , Eeprom_adres
      Printbin Rs232_arg
   Case Eeprom_msb:
      !push r16
      !in r16, {rs232_arg}
      !out {eeprom_adres+1}, r16
      !pop r16
   Case Eeprom_lsb:
      !push r16
      !in r16, {rs232_arg}
      !out {eeprom_adres}, r16
      !pop r16
   End Select
End Sub
```

Rys. 3.5. Definicja procedury wykonującej przesłany łączem RS232 rozkaz.

4. Emulacja dekompresora

4.1. Założenia emulacji programowej i sterowania

Zgodnie z rysunkiem 2.3 do budowy dekompresora trzeciej generacji potrzeba trzech zasadniczych bloków funkcjonalnych:

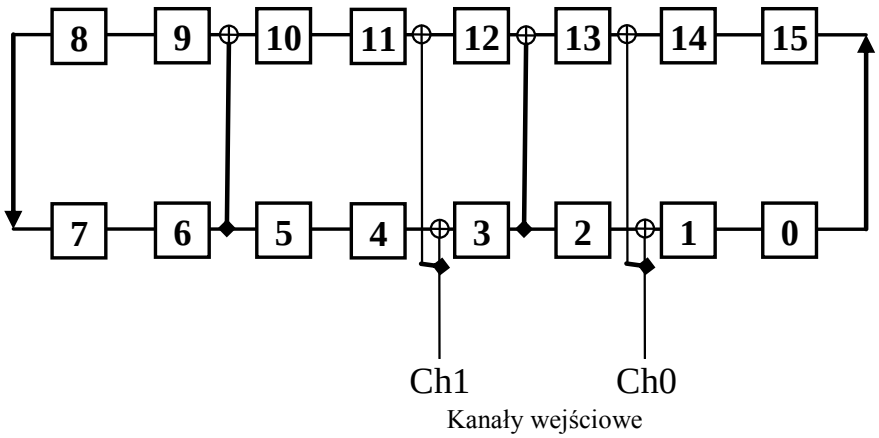
- pierścieniowych generatorów sekwencji pseudolosowej
- przesuwników faz (logiki ExOR),
- tablicy multiplekserów 2 x 1.

Każdy z tych elementów musi zostać opisany kodem maszynowym, wykonywanym przez arytmetyczno-logiczny mikrokontrolera. Warto w tym miejscu zauważyć, że procesor nie jest w stanie wykonać w ciągu jednego taktu zegara wszystkich operacji, które wykonywane są przez sprzętowy układ sekwencyjny w tym samym czasie. Wobec powyższego należy mówić nie o przesunięciu danych w rejestrach, lecz o przeliczaniu zawartości tychże.

Ponieważ na dekompresor składają się az trzy pary: generator pierścieniowy + przesuwnik fazy, wartości ich wyjść w poszczególnych taktach zegara (taktującego emulowany dekompresor) muszą zostać obliczone w odpowiedniej kolejności tak, by stan wyjść dekompresora był właścIwy.

4.2. Emulacja generatora pierścieniowego

Rysunek 4.1 przedstawia przykładową strukturę generatora pierścieniowego. Linie oznaczone Ch1 i Ch0 to kanały wejściowe dekompresora (ang. *Injectors*). Sygnały z tych wejść oraz sygnały pochodzące ze sprzężeń zwrotnych są sumowane (ExOR) z sygnałami wyjściowymi przerzutników, wyznaczając w ten sposób stan rejestru w kolejnej szczelinie czasowej (w kolejnym takcie zegara). Emulacja takiego rejestru polega na wyznaczeniu masek wejść sterujących oraz sprzężeń zwrotnych, przesunięciu cyklicznym zawartości rejestru oraz zsumowaniu nowej zawartości rejestru z maskami. Wspomniane maski to zmienne, których reprezentacja binarna jest równa długości rejestru. Zmienne te reprezentują stany wejść lub sprzężeń zwrotnych na wejściach poszczególnych przerzutników. w praktyce oznacza to, jedynki logiczne pojawią się w maskach tylko na tych pozycjach, które odpowiadają zakończeniom sprzężeń zwrotnych (lub do których dochodzą wejścia sterujące) przy jednoczesnej wartości „1” panującej na tych zakończeniach.



$$x^{16} + x^{10} + x^4 + 1$$

Rys. 4.1. Przykład generatora pierścieniowego

```
RingN = <last_state_of_Ring_N>

Restore (Ring_N);     // Ustaw wskaźnik danych na sekcję Ring_N

Inj_Mask = 0x0000;    // Maska wejść (injectors)
Fdb_Mask = 0x0000;    // Maska sprzężeń zwrotnych

Injectors = Read();   // Odczytaj liczbę wejść (injectors)

/************************************
Generacja maski wejść (injectors)
************************************/
For i = 1 To Injectors
{
    Source = Read();  // Odczytaj numer kanału wejściowego
    Target = Read();  // Odczytaj numer przerzutnika docelowego
    Inj_Mask.Target = Channels.Source; // X.Y = Y bit zmiennej X
}

/************************************
Generacja maski sprzężeń zwrotnych
************************************/
Feedbacks = Read();   // Odczytaj liczbę sprzężeń
For i = 1 To Feedbacks
{
    Source = Read();  // Odczytaj numer przerzutnika źródłowego
    Target = Read();  // Odczytaj numer przerzutnika docelowego
    Fdb_Mask.Target = RingN.Source; // X.Y = Y bit zmiennej X
}

RingN = RotateRight(RingN);       // Przesunięcie zawartości rejestru
RingN = RingN XOR Fdb_Mask XOR Inj_Mask; // dodanie wartości masek
```

Rys. 4.2. Pseudo-kod funkcji emulującej generator pierścieniowy

```
'--------- RING0 32b --------------------
'  Przelicza Ring0 Wymaga wypełnionej
'  zmiennej Channels as Integer.
'  Niszczy:
'  - (Aux, Aux2, Auxt(2)) as byte
'  - (longaux, longaux2) as long,
'----------------------------------------
Sub Shift_ring0()
    Longaux = &H00000000                    'Zerowanie maski stanu
    Longaux2 = &H00000000                   'Zeorwanie maski kanałów
    'Generowanie maski stanow
    Address = 2
    Readeeprom Auxt(1) , Address            'liczba XORow
    Incr Address
    Auxt(2) = 1
    For Auxt(2) = 1 To Auxt(1)
        Readeeprom Aux , Address            'cel
        Incr Address
        Readeeprom Aux2 , Address           'zrodlo
        Incr Address
        Longaux.aux2 = Ring0.aux
    Next Auxt(2)
    'Generowanie maski wejść
    Readeeprom Auxt(1) , Address            'liczba injectors
    Incr Address                            'liczba XORow
    Auxt(2) = 1
    For Auxt(2) = 1 To Auxt(1)
        Readeeprom Aux , Address            'cel
        Incr Address
        Readeeprom Aux2 , Address           'zrodlo (kanał)
        Incr Address
        Longaux2.aux2 = Channels.aux
    Next Auxt(2)
    '- - - - - - - - - -
    Rotate Ring0 , Right , 1          'Przesuniecie rejestru
    Ring0 = Ring0 Xor Longaux         'XOR z maską stanow
    Ring0 = Ring0 Xor Longaux2        'XOR z maską kanałów
End Sub
```

Rys. 4.3. Rzeczywisty kod funkcji przesuwającej rejestr generatora pierścieniowego

Opisane operacje ilustruje pseudo-kod pokazany na rysunku 4.2. W przykładzie tym zakłada się, że dane opisujące strukturę rejestru znajdują się w odpowiednim obszarze pamięci programu kontrolera w postaci bajtów, oznaczających (kolejno):

– liczbę wejść (injectors),
– zestaw par: „numer kanału źródłowego”, „numer przerzutnika docelowego”,
– liczbę sprzeżeń zwrotnych,
– zestaw par: „numer przerzutnika źródłowego”, „numer przerzutnika docelowego”.

Odczyt tych danych odbywa sie przy użyciu dwóch funkcji:

– restore(Value) – ustawiającej wewnętrzny wskaźnik na adres dany wartością parametru Value,

– Read() – zwracającej wartość bajtu pamięci programu o adresie danym wewnętrznym wskaźnikiem (ustawianym funkcją Restore) oraz inkrementującej ten wskaźnik.

Procesor wyznacza słowa masek kanałów wejściowych i sprzężeń zwrotnych. Po ich wyznaczeniu, zawartość rejestru zostaje przesunięta cyklicznie w lewo za pomocą funkcji RotateRight, a następnie sumowana (ExOR) z obiema maskami.

W układzie rzeczywistym struktura opisująca układ sprzężeń generatorów pierścieniowych została zapisana w pamięci EEPROM mikrokontrolera obsługującego dekompresję. Pamięć ta jest odrębna od stałej pamięci programu (Flash-ROM), a jej zawartość może być modyfikowana podczas normalnej pracy kontrolera. Dzięki temu można dokonywać zmian w strukturze generatorów pierścieniowych z poziomu oprogramowania obsługującego środowisko testowe, działające pod kontrolą komputera PC, bez ingerencji w kod źródłowy programu dekompresora. Rzeczywisty kod emulujący jeden z dwóch 32-bitowych generatorów pierścieniowych przedstawiono na rysunku 4.3.

4.3. Emulacja przesuwnika fazy

Przesuwnik fazy jest wielowyjściowym układem kombinacyjnym o specyficznej, prostej strukturze. Stan każdego z wyjść wyznacza jedna bramka ExOR na podstawie stanu *n* wejść. Założono, że*n* zawsze przyjmować będzie wartość 3, co ułatwi opis przesuwnika wewnątrz pamięci mikrokontrolera. W pamięci programu, poczynając od adresu wyznaczonego przez kompilator, umieszczonych jest *n* trzybajtowych słów. Poszczególne bajty każdego słowa oznaczają indeksy wejść przesuwnika fazy, które należy zsumować (ExOR), aby otrzymać stan odpowiedniego wyjścia. Pseudokod ilustrujący sposób korzystania z opisanej struktury danych pokazano na rysunku 4.4.

Z uwagi na ograniczenie dostępnej pamięci EEPROM, struktura przesuwników faz musi zostać na stałe zapisana w pamięci programu. Jej modyfikacja pociąga za sobą konieczność przeprogramowania zawartości pamięci Flash mikrokontrolera dekompresującego. Rzeczywisty kod funkcji emulującej przesuwnik fazy, na przykładzie rejestru bazowego (Parent Pattern) przedstawiony jest na rysunku 4.5.

```
Restore (Phase_Shifter_N);    // Ustaw wewnętrzny wskaźnik na adres
                              // struktury opisującej przesuwnik fazy

PS_Size = 127     //liczba wyjść przesuwnika fazy - 1

For Gate = 0 To PS_Size
{
      // Odczyt indeksów wejść
      Inp0 = Read();
      Inp1 = Read();
      Inp2 = Read();
      // Przeliczenie (ExOR)
      PS_M.Gate = Inpt0 XOR Inp1 XOR Inp2; // X.Y = Y bit zmiennej X
}
// Wynik operacji w PS_N
```

Rys. 4.4. Ilustracja sposobu emulacji przesuwnika fazy

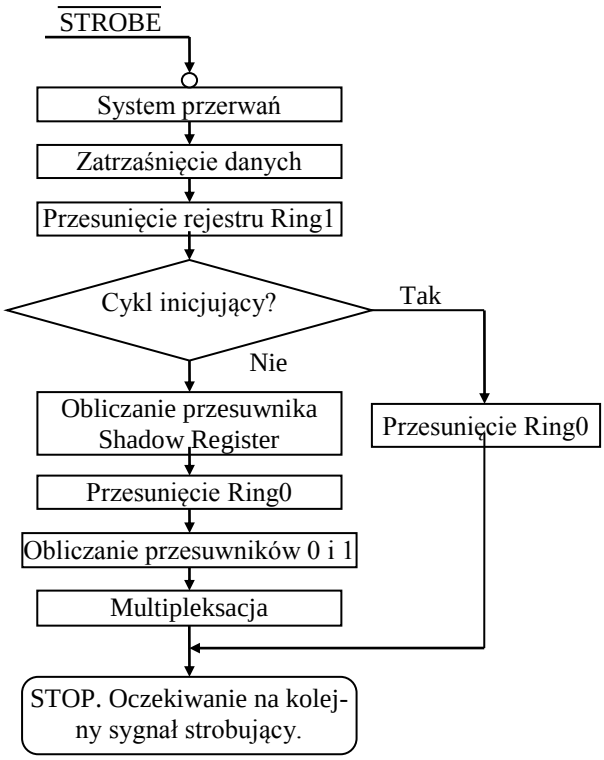
```
'--------- PARENT ----------------------
' Oblicza XORy bloku Phase Shifter 0
' Wynik w Phase_Shifter1(4) as long
' Niszczy Aux
'---------------------------------------
Sub Do_phase_shifter0()
   Restore Data_phase_shifter0
   B = 0
   For B = 0 To 31 'po bitach z tablicy Long phase shiftera
      For W = 1 To 4
         'odczytanie stanu wejsc 3-we bramki XOR
         Read Aux
         Ps_auxbit0 = Auxring0.aux
         Read Aux
         Ps_auxbit1 = Auxring0.aux
         Read Aux
         Ps_auxbit2 = Auxring0.aux
         'wyliczenie stanu wyjscia 3-we bramki XOR
         Ps_auxbit0 = Ps_auxbit0 Xor Ps_auxbit1
         Ps_auxbit0 = Ps_auxbit0 Xor Ps_auxbit2
         Shift Phase_shifter0(w) , Right , 1
         Phase_shifter0(w).31 = Ps_auxbit0
      Next W
   Next B
End Sub
```

Rys. 4.5. Rzeczywisty kod funkcji emulującej przesuwnik fazy

4.4. Algorytm dekompresji

W sprzętowym dekompresorze pojawienie się sygnału strobującego, którym może być na przykład takt zegara, powoduje niemal jednoczesne wykonanie przesunięć wszystkich rejestrów, a opóźnienie wprowadzane przez część kombinacyjną (na przykład przesuwniki faz) jest pomijalne. Gdy proces dekompresji wykonywany jest przez układ mikroprocesorowy, nie jest możliwe jednoczesne wykonanie wszystkich przesunięć, a wartości wyj-

ściowe układów kombinacyjnych muszą zostać obliczone, przy zachowaniu odpowiedniej kolejności, jako że wszystkie elementy dekompresora wywierają na siebie wzajemny wpływ



Rys. 4.6. Algorytm dekompresji

.

Algorytm dekompresji przedstawiono na rysunku 4.6. Kontroler sterujący oczekuje, aż linia READY kontrolowana przez dekompresor przyjmie stan aktywny, po czym ustawia na równoległej magistrali skompresowane słowo danych – odpowiada to ustawieniu stanu kanałów sterujących przez tester. Następnie na krótki moment aktywowana jest linia STROBE. Jej aktywacja polega na wyzerowaniu tej linii, co wykrywane jest przez kontroler „Slave" jako żądanie obsługi przerwania dla celów dekompresji. W tym momencie zostaje wywołana funkcja, której zadaniem jest obliczenie nowych wartości wszystkich rejestrów dekompresora. Obliczenia rozpoczynają się od przesunięcia głównego rejestru – Ring1 (zgodnie z rysunkiem 2.3). Jeśli trwa inicjacja dekompresora – a więc wprowadzane są wartości początkowe do rejestrów pierścieniowych – funkcja powoduje przeliczenie wartości Ring0 i udostępnia sygnał READY w oczekiwaniu na kolejny impuls strobujący. Jeśli jednak trwa proces dekompresji, przed przesunięciem Ring0 należy odczytać z buforu dane wejściowe dla tego rejestru i zapisać w buforze dane aktualnie wystawione na magistrali danych - czynność ta emuluje opóźnienie o jeden takt zegara, wnoszone przez układ

sterujący multipleksacją, wykonywaną po ustaleniu nowych wartości wszystkich przesuwników faz, występujących w układzie.

Wynikiem działania algorytmu dekompresji jest słowo danych, które zostałoby wprowadzone do ścieżek testujących wraz z pojawieniem się kolejnego impulsu strobującego wystawionego przez tester. Dodatkowo obliczany jest bit kontrolny (jako różnica symetryczna wszystkich bitów wyniku działania funkcji dekompresującej), który podlega kontroli przez nadzorujące proces oprogramowanie pracujące pod kontrolą komputera PC. Definicja funkcji realizującej algorytm dekompresji przedstawiony jest na rysunku 4.7.

```
'----------------------------------------
' Przelicza wszystkie rejestry.
'----------------------------------------
Sub Compute_data(byval Is_init As Byte)
    Dim My_aux1 As Byte
    Call Shift_ring1
    Call Do_ps1
    If Is_init = 0 Then 'Jesli nie trwa cykl inicjacyjny
       Call Shift_ring0
       Call Do_shadow
       Call Do_ps0
    End If
    My_aux1 = 1
    For My_aux1 = 1 To 4
       Not_ps2(my_aux1) = Not Ps2(my_aux1)
       Scans(my_aux1) = Ps0(my_aux1) And Not_ps2(my_aux1)
       Scansaux(my_aux1) = Ps1(my_aux1) And Ps2(my_aux1)
       Scans(my_aux1) = Scans(my_aux1) Or Scansaux(my_aux1)
    Next My_aux1
    Call Xor_scans       'Wystaw bit sumy kontrolnej
    Auxring0 = Ring0
    Auxring1 = Ring1
End Sub
```

Rys. 4.7. Definicja funkcji realizującej algorytm dekompresji

5. Oprogramowanie PC

5.1. Cechy i funkcjonalność

Oprogramowanie sterujące i kontrolujące proces dekompresji, działające pod kontrolą komputera PC, zostało napisane w środowisku Visual Studio 6 w języku Basic, jako aplikacja ze zintegrowanym graficznym interfejsem użytkownika. Zaletą języka Visual Basic 6 jest szybkość działania napisanej w nim aplikacji i jej niezależność od składników systemu operacyjnego, ponieważ do działania pliku wykonywalnego wymagana jest jedynie obecność (w systemie lub w katalogu programu) bibliotek dynamicznych pakietu *VBRunTme*.

W skład oprogramowania obsługującego dekompresor wchodzą następujące moduły:

- blok kontroli sprawności sprzętowego środowiska eksperymentalnego, w szczególności kontroli stanu połączeń między kontrolerami, poprawności komutacji łącza RS232 (kanału UART) i bezbłędności transmisji pomiędzy układem testowym a komputerem,
- blok sterujący procesem dekompresji,
- blok umożliwiający dokładną analizę wyników działania dekompresora, łącznie z wynikami pośrednimi, z możliwością bezpośredniego wprowadzania danych do rejestrów dekompresora.
- blok konwersji danych definiujących strukturę podukładów dekompresora z formatu wyjściowego narzędzi firmy Mentor Graphics do postaci wymaganej przez oprogramowanie wewnętrzne kontrolera dekompresującego,
- blok obsługi łącza i kontroli transmisji (kod modułu kontroli łącza umieszczony jest w dodatku C).

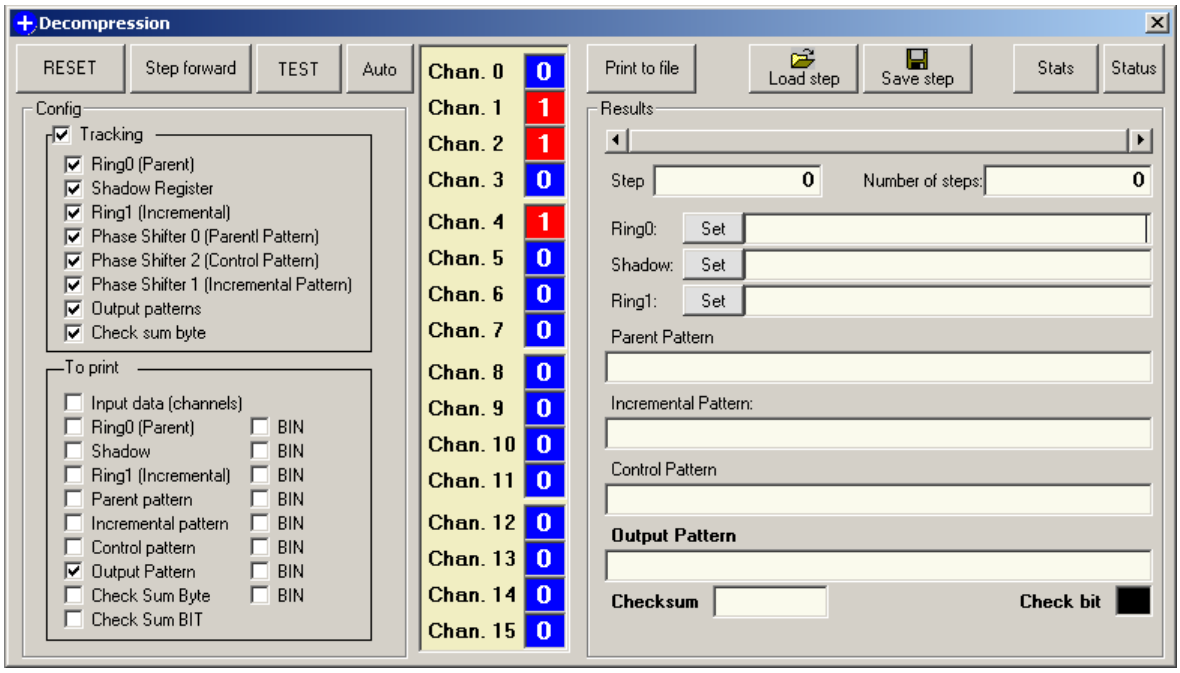
Testowanie środowiska testowego jest pierwszą czynnością wykonywaną po uruchomieniu programu komputerowego, ponieważ linie łączące obydwu kontrolery (sterujący i dekompresujący) są wykonane jako nietrwałe połączenia stykowe, cechujące się awaryjnością. Test polega na sprawdzeniu przewodzenia każdej linii z osobna przy przeciwsobnej pracy linii sąsiednich. W przypadku wykrycia awarii na wszystkich liniach sprawdzane jest automatycznie, czy wtyki przewodów połączeniowych nie zostały obsadzone odwrotnie, co ułatwia diagnozę problemu. Następnie testowane jest działanie komutacji łącza RS232 (UART) przy najwyższej możliwej szybkości, na wypadek uszkodzenia układu kluczującego.

Kontrola bezbłędności transmisji polega na przesłaniu losowo wybranej liczby będącej argumentem rozkazu „TEST”, przesłanego z komputera do kontrolera sterującego. Po odebraniu wartość argumentu jest inkrementowana i przesyłana do komputera jako potwierdzenie jej weryfikacja pozwala na zgrubną kontrolę poprawności transmisji.

5.2. Kontrola przepływu i śledzenie pracy dekompresora

Możliwe są trzy tryby dekompresji:

– Ręczny – manualne ustawianie stanu linii danych (kanałów testera),
– Automatyczny – samoczynne pobieranie danych do dekompresji z pliku i przeprowadzenie pełnej dekompresji,
– Automatyczny krokowy – pobieranie danych testowych z pliku, lecz każdy krok dekompresji odbywa się na polecenie użytkownika.

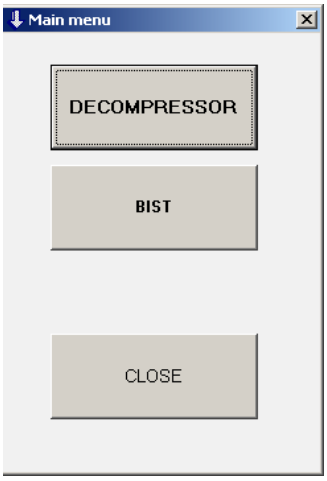


Rys. 5.1. Okno sterowania i kontroli pracy dekompresora podczas analizy danych

Niezależnie od wybranego trybu pracy możliwe jest rejestrowanie zawartości wybranych rejestrów i przeglądanie ich w wygodny sposób po zakończeniu (lub trakcie) działania dekompresora. Na rysunku 5.1 pokazano przykładową zawartość okna kontroli procesu dekompresji podczas analizy zebranych danych.

5.3. Obsługa oprogramowania PC

Po uruchomieniu programu obsługującego dekompresor na ekranie ukazuje się główne menu (jak na rysunku 5.2), w którym dostępne są dwie opcje: otwarcie okna obsługi procesu dekompresji, oraz BIST (ang. *Bult-In Self Test)* środowiska testowego. Po wybraniu opcji autotestu na ekranie ukaże się okno z wynikami (przykład pokazano na rysunku 5.3).



Rys. 5.2, Menu główne.

Rys. 5.3. Wyniki autotestu układu

Po uruchomieniu dekompresora wewnątrz pól reprezentujących wewnętrzne rejestry układu wyświetlone zostaną znajdujące się wewnątrz nich dane. W celu zainicjowania i wyzerowania dekompresora należy użyć przycisku „Reset".

Stan kanałów testera można zmieniać ręcznie, poprzez podwójne kliknięcie pól reprezentujących każdą z linii, w środkowej części okna. Stan tych pól zostanie natychmiast odwzorowany poprzez odpowiednie poziomy napięć na magistrali łączącej obydwa kontrolery w środowisku testowym. Kliknięcie przycisku „Step forward" spowoduje przesłanie rozkazu „DO_DECOMP" do kontrolera dekompresującego oraz kontrolne odczytanie stanu rejestrów wybranych w sekcji „Tracking". Wszystkie wartości są rejestrowane i zapamiętywane do czasu wyzerowania dekompresora (automatycznego lub ręcznego, przy użyciu przycisku „Reset"). Ponadto możliwe jest ręczne wpisanie wartości do wybranych rejestrów (przy użyciu odpowiednich przycisków „Set").

6. Rezultaty eksperymentu

6.1. Dane wejściowe

Za pomocą narzędzi do generacji testów firmy Mentor Graphics zostały wygenerowane pliki zawierające wszystkie niezbędne dane wejściowe oraz opisy struktur dekompresora, umożliwiające przeprowadzenie eksperymentów i badań weryfikujących działanie mikroprocesorowej dekompresji. Wygenerowane zostały dwa pliki:

– zawierający opis struktury, dane wejściowe i wartości wyspecyfikowanych bitów w ścieżkach testujących (kompleksowe dane),
– zawierający wyniki pośrednie – zawartość poszczególnych rejestrów wewnętrznych dekompresora w poszczególnych szczelinach czasowych (umożliwia wzrokową kontrolę poprawności pracy poszczególnych podukładów dekompresora).

Zawartość pierwszego z plików jest bardzo obszerna. Treść pliku rozpoczyna się od nagłówka, opisującego parametry fizyczne dekompresora:

```
Scan chains: 256        // liczba ścieżek testujących
Scan length: 99         // długość rejestrów ścieżek testujących
Ring size: 32           // rozmiar generatorów pierścieniowych
Input channels: 4       // liczba kanałów wejściowych
Init period: 8          // liczba cykli wstępnych
XOR inputs: 3           // liczba wejść bramek ExOR przesuwników fazy
```

W dalszej kolejności zdefiniowana jest struktura generatorów pierścieniowych:

```
Ring Taps: (tap(from, to))
18(7,25) 14(9,23) 9(11,20)
Injection sites: (channelNr, injectorNr)injectionSite
(0,0)1 (0,1)29 (1,0)5 (1,1)26 (2,0)9 (2,1)21 (3,0)13 (3,1)17
```

Kolejnym ważnym opisem w pliku konfiguracyjnym jest opis struktury przesuwników faz, podany jako jedna struktura (wejścia 0 – 15 oraz wyjścia 0 – 127 tworzą przesuwnik „Phase Shifter 0”, pozostałe – „phase shifter 1”):

```
Phase shifter:
[0]: 250 240 229 220 211 193 191 185 165 154 152 127 117 115 101 93 80 73
62 44 36 27 15 5
[1]: 248 234 226 216 208 195 190 170 169 159 144 127 123 113 104 88 75 65
61 51 39 20 17 7
[2]: 251 245 230 226 212 197 188 174 172 151 148 141 125 108 105 87 73 67
55 51 40 27 10 4
[3]: 236 229 217 202 197 189 183 170 153 142 129 122 106 99 90 81 63 54
42 31 23 14 6
```

```
[4]: 251 233 232 206 201 186 175 160 159 140 137 135 112 102 93 80 72 54
45 37 22 12 1
[5]: 252 250 241 209 196 183 179 166 162 145 137 119 114 102 85 83 65 59
46 45 24 19 7
[6]: 247 246 230 215 212 204 173 169 165 164 145 128 125 111 100 86 73 64
58 44 34 25 18 8
[7]: 236 233 232 210 201 187 186 165 164 140 131 119 113 100 86 80 69 62
50 39 23 11 3
[8]: 252 235 222 218 202 200 185 184 168 164 146 132 124 115 99 89 82 67
54 45 33 29 17 8
[9]: 254 242 231 221 203 193 191 190 167 150 147 130 120 112 98 94 91 81
61 52 40 21 19 3
[10]: 251 249 224 216 211 192 180 172 159 154 148 130 126 114 99 92 75 69
56 43 38 27 13 9
[11]: 244 241 227 219 204 200 199 190 162 150 140 136 120 110 96 91 76 71
57 51 37 26 16 8
[12]: 240 222 218 206 201 179 176 163 161 139 128 122 107 96 91 76 68 53
52 38 24 15 0
[13]: 255 253 234 226 221 210 200 184 182 157 150 146 133 126 109 104 103
74 72 56 44 33 25 16 2
[14]: 243 235 224 214 207 193 178 171 167 163 147 136 134 121 116 84 77
66 60 42 41 26 17 1
[15]: 245 238 225 217 204 197 182 178 156 149 138 137 123 109 97 94 78 66
62 48 37 20 19 2
[16]: 253 238 231 218 210 192 176 169 168 149 139 132 121 115 98 87 82 64
55 43 36 35 15 0
[17]: 243 242 223 213 212 211 181 175 167 166 144 128 119 110 103 87 83
70 59 46 31 29 12 5
[18]: 254 252 239 232 231 220 194 187 183 160 156 146 143 122 113 103 93
79 63 60 50 35 21 18 0
[19]: 248 247 233 215 205 196 174 172 154 151 142 129 125 108 97 92 89 84
83 82 40 23 18 6
[20]: 246 240 222 220 206 194 180 170 158 157 138 134 116 108 105 84 77
66 57 49 35 30 16 9
[21]: 250 236 223 214 209 195 177 175 153 151 139 134 118 112 97 88 78 70
56 43 34 28 14 9
[22]: 245 235 229 221 208 199 186 180 162 155 143 129 116 111 95 86 74 71
59 42 41 39 38 29
[23]: 249 234 228 219 207 196 185 184 157 149 141 130 124 106 95 85 79 68
57 47 32 25 13 4
[24]: 244 239 230 213 209 198 179 178 155 152 144 135 118 107 101 88 75
67 58 47 32 28 13 4
[25]: 249 239 227 217 203 191 182 174 171 161 138 136 121 106 96 85 74 63
53 48 32 22 10 7
[26]: 247 237 227 214 203 195 177 176 166 163 147 133 124 111 98 90 78 71
53 49 41 21 11 5
[27]: 255 248 241 228 225 208 194 181 173 156 155 143 127 123 110 95 90
77 65 60 48 36 22 10 1
[28]: 246 242 228 215 205 198 189 188 160 153 148 131 120 109 102 94 79
68 64 46 31 30 12 6
[29]: 255 244 237 223 219 207 199 189 188 187 158 141 135 117 114 100 89
81 69 61 47 33 30 28 24
[30]: 254 243 237 225 216 205 192 181 177 158 152 145 133 132 126 117 92
76 72 58 52 50 26 14 3
[31]: 253 238 224 213 202 198 173 171 168 161 142 131 118 107 105 104 101
70 55 49 34 20 11 2
```

Definicję przesuwnika fazy należy przetransformować ze struktury o konwencji (wejście: lista wyjść) do postaci (wyjście: lista wejść), zgodnej z algorytmem programowej emulacji

przesuwnika. Transformacji tej można dokonać przy pomocy sterującego oprogramowania PC, wybierając z menu „Code” pozycję „Conversion”, następnie „Phase shifter”. Wynikiem konwersji będzie opis zgodny z językiem Bascom, gotowy do wklejenia w odpowiednim miejscu kodu programu:

```
Data 6, 12, 17    'Output 0
Data 4, 18, 28    'Output 32
Data 10, 16, 30   'Output 64
Data 10, 14, 31   'Output 96
Data 3, 19, 22    'Output 1
Data 12, 25, 31   'Output 33
Data 0, 9, 14     'Output 65
Data 15, 27, 30   'Output 97
Data 9, 10, 23    'Output 2
Data 5, 11, 22    'Output 34
Data 18, 20, 27   'Output 66
Data 1, 2, 13     'Output 98
Data 7, 28, 31    'Output 3
Data 12, 14, 26   'Output 35
Data 1, 21, 26    'Output 67
Data 11, 25, 26   'Output 99
Data 8, 16, 30    'Output 4
Data 6, 7, 8      'Output 36
Data 5, 19, 23    'Output 68
Data 23, 27, 28   'Output 100
Data 13, 26, 30   'Output 5
Data 0, 6, 7      'Output 37
Data 2, 3, 15     'Output 69
Data 0, 3, 22     'Output 101
Data 14, 20, 21   'Output 6
Data 5, 17, 26    'Output 38
Data 24, 28, 31   'Output 70
Data 2, 6, 24     'Output 102
Data 4, 24, 29    'Output 7
Data 9, 14, 17    'Output 39
Data 11, 22, 29   'Output 71
Data 9, 16, 18    'Output 103
Data 11, 14, 25   'Output 8
Data 8, 16, 31    'Output 40
Data 8, 11, 13    'Output 72
Data 4, 7, 18     'Output 104
Data 4, 5, 15     'Output 9
Data 1, 6, 16     'Output 41
Data 4, 7, 12     'Output 73
Data 4, 7, 19     'Output 105
Data 15, 20, 25   'Output 10
Data 1, 3, 20     'Output 42
Data 3, 8, 31     'Output 74
Data 1, 13, 23    'Output 106
Data 12, 16, 21   'Output 11
Data 14, 25, 31   'Output 43
Data 9, 25, 26    'Output 75
Data 8, 14, 22    'Output 107
Data 4, 7, 11     'Output 12
Data 2, 10, 19    'Output 44
Data 6, 11, 15    'Output 76
Data 3, 7, 21     'Output 108
Data 2, 23, 29    'Output 13
Data 6, 27, 31    'Output 45
Data 19, 28, 30   'Output 77
Data 26, 29, 30   'Output 109
Data 3, 19, 31    'Output 14
Data 2, 19, 25    'Output 46
Data 4, 12, 20    'Output 78
Data 15, 16, 31   'Output 110
Data 18, 22, 27   'Output 15
Data 4, 17, 21    'Output 47
Data 14, 23, 29   'Output 79
Data 18, 24, 25   'Output 111
Data 1, 17, 24    'Output 16
Data 12, 16, 26   'Output 48
Data 1, 22, 27    'Output 80
Data 0, 12, 20    'Output 112
Data 5, 6, 30     'Output 17
Data 21, 26, 30   'Output 49
Data 5, 21, 24    'Output 81
Data 5, 11, 27    'Output 113
Data 8, 13, 18    'Output 18
Data 14, 15, 24   'Output 50
Data 7, 13, 16    'Output 82
Data 9, 17, 28    'Output 114
Data 9, 14, 26    'Output 19
Data 5, 12, 24    'Output 51
Data 0, 10, 17    'Output 83
Data 14, 17, 30   'Output 115
Data 2, 10, 28    'Output 20
Data 10, 20, 22   'Output 52
Data 2, 6, 17     'Output 84
Data 11, 24, 29   'Output 116
Data 15, 16, 23   'Output 21
Data 17, 27, 30   'Output 53
Data 17, 24, 31   'Output 85
Data 2, 15, 22    'Output 117
Data 9, 11, 13    'Output 22
Data 13, 15, 25   'Output 54
Data 14, 21, 26   'Output 86
Data 6, 20, 28    'Output 118
Data 2, 19, 21    'Output 23
Data 3, 5, 18     'Output 55
Data 6, 19, 28    'Output 87
Data 6, 19, 26    'Output 119
Data 0, 24, 30    'Output 24
Data 8, 13, 23    'Output 56
Data 1, 10, 30    'Output 88
Data 1, 19, 27    'Output 120
Data 3, 21, 28    'Output 25
Data 0, 8, 23     'Output 57
Data 3, 15, 25    'Output 89
Data 10, 23, 25   'Output 121
```

```
Data 0, 10, 19    'Output 26          Data 13, 20, 23   'Output 29
Data 4, 7, 22     'Output 58          Data 3, 28, 29    'Output 61
Data 8, 12, 16    'Output 90          Data 9, 13, 22    'Output 93
Data 0, 5, 21     'Output 122         Data 13, 16, 31   'Output 125
Data 22, 24, 27   'Output 27          Data 20, 29, 30   'Output 30
Data 7, 18, 29    'Output 59          Data 1, 9, 11     'Output 62
Data 11, 23, 29   'Output 91          Data 8, 12, 20    'Output 94
Data 2, 4, 10     'Output 123         Data 9, 18, 30    'Output 126
Data 15, 18, 27   'Output 28          Data 1, 4, 10     'Output 31
Data 2, 28, 29    'Output 60          Data 0, 9, 25     'Output 63
Data 0, 18, 20    'Output 92          Data 17, 21, 29   'Output 95
Data 5, 8, 18     'Output 124         Data 13, 27, 29   'Output 127
```

Funkcja transformująca oraz interfejs użytkownika konwertera przedstawione są w dodatku A

Po zdefiniowaniu budowy wszystkich elementów układu pojawiają się łańcuchy zawierające dane, które muszą zostać wprowadzone do dekompresora poprzez emulowane kanały testera, aby wewnątrz ścieżek testujących pojawiły się bity o zadanych wartościach na wyspecyfikowanych pozycjach. Przykładowe dane wyglądają następująco:

```
[0][-1][0]
00011100000100010000001010110011100110010111010010000010111101011101001100010011101010110101110011000011000
[0][-1][1]
00001101011100000000010000011101110011000100101100110111000010011000000010101010001001010001010010001111000
[0][-1][2]
00001010001100111010000010101110101010110011001011001010000110001000110110100011110100000101100100001101000
[0][-1][3]
00000011101111010101010111011000110101010010111000100001000000000001111000011011110011100101000000101001001

(0,84,82)(1,84,92)(1,93,74)(0,94,4)(0,94,55)….
```

Warto zauważyć, że po zapisie wszystkich danych wejściowych pojawia się opis wartości wyspecyfikowanych bitów, w konwencji (wartość, numer ścieżki testującej, pozycja). Taki zapis umożliwia łatwe przeprowadzenie kontroli poprawności dekompresji, choć wymaga odpowiedniego algorytmu parsującego odczytywane z rejestrów mikrokontrolera dane. Proces kontroli można jednak przyspieszyć, znając wartości zdekompresowanych danych ze sprzęgu łączącego dekompresor ze ścieżkami testującymi. Wartości te zostały zawarte w drugim pliku. Przykładowy jego fragment wygląda następująco:

```
end: Read test cubes
SEED: 0000 RING[0]: 00000000000000000000000000000000
SEED: 0000 RING[1]: 00000000000000000000000000000000
SEED: 0000 RING[2]: 00000000000000000000000000000000
```

```
SEED: 1000 RING[3]: 00000000000000000000000000000000
SEED: 1110 RING[4]: 10000000000000000000000000001000
SEED: 1100 RING[5]: 10001000100000000000100001011001
SEED: 0011 RING[6]: 10011001000000000001000011111011
SEED: 0101 RING[7]: 00110010100010001010100101110111
TIME_SLICE[0]:
01100000101011101110011010010000110111010011001011010010010100000010001100
11010010101110100100101000001111100000010011011011010110000000011110101110
01001001101000011011111010000010011011001010000000000111010101000100100100
00110000011010111110110001110001011111
TIME_SLICE[1]:
01100110000001001011011000001000110101011100011110101000000010010001101111
01100010011001010000100110000011010100100011000001110110000000111101011100
01001001101000011011111010000010011011001010000000000111010101000100100100
00110000011010111110110001110001011111
TIME_SLICE[2]:
11011110101011111100111010011001111000011111001001101000011111111100001010
01011110011111011100101010001110111010100101110010010000000000011110101110
01001001101000011011111010000010011011001010000000000111010101000100100100
00110000011010111110110001110001011111
TIME_SLICE[3]:
11001011010011100111011100100110010010110110101011000110100100000010111100
11110000001110010110110010000011100010100101111100001101111110100110000011
00010000100100110100001111001011110110010111010111100010000111001001001100
10010001101001010001011001111001010100
```

6.2. Sprawdzenie działania i pomiar czasu dekompresji

Dane wejściowe zostały przepisane do postaci zawierającej wyłącznie wartości bitów w formie ciągów znaków ASCII:

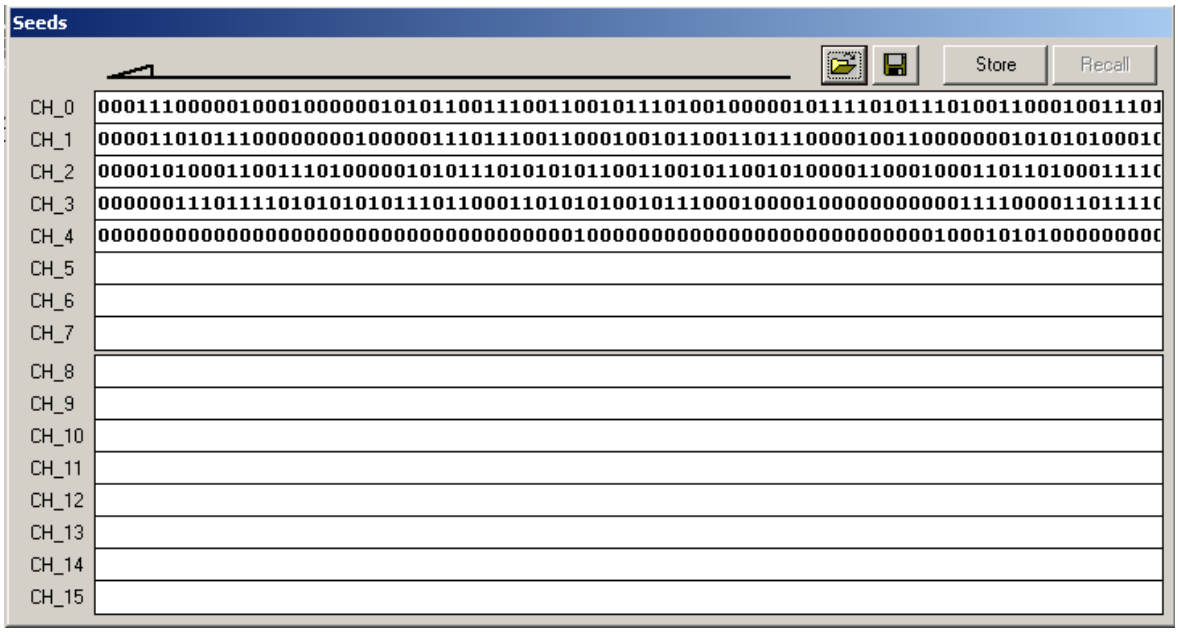
```
"00011100000100010000001010110011100110010111010010000010111101011101001100
0010011101010110101110011000011000"
"00001101011100000000100000111011100110001001011001101110000100110000000101
0101010001001010001010010001111000"
"00001010001100111010000010101110101010110011001011001010000110001000110101
10100011110100000101100100001101000"
"00000011101111010101010111011000110101010010111000100001000000000011110000
01101111001110010100000101001001"
"00000000000000000000000000000000000010000000000000000000000000001000101010
00000000000000000000000000000100000000000"
```

Warto zwrócić uwagę na fakt, że w ciągu odpowiadającemu kanałowi o indeksie 4 występuje 8 dodatkowych znaków. Wynika to z tego, że wprowadzanie danych do dekompresora poprzedzone jest ośmio taktowym cyklem inicjacyjnym (ang. *init period*). Podczas inicjacji pozostałe kanały pozostają wyzerowane.

Odpowiednio przygotowane dane zostają wczytane do programu sterującego, co pokazano na rysunku 6.1. Po zakończeniu procesu dekompresji dane z wybranych szczelin czasowych zostały porównane z ciągami wygenerowanymi przez narzędzie kompresujące.

Kontroli podlegały także wartości wszystkich wyspecyfikowanych bitów. Dane okazały się zgodne, stopa błędu wynosi więc 0%.

Podczas dekompresji zostały również zebrane dane pozwalające oszacować czas oraz liczbę instrukcji maszynowych, niezbędnych do wykonania wszystkich czynności emulujących działanie sprzętowego dekompresora w czasie jednego taktu sygnału zegarowego. Czas dekompresji (tj. w czasie jednego wirtualnego taktu zegara) zmierzony przez sterownik wynosi 16ms. Pomiar rozpoczynał się po zakończeniu przesłania rozkazu DO_DECOMP a kończył po uzyskaniu potwierdzenia zakończenia dekompresji. Potwierdzenie uzyskiwane jest poprzez wysłanie zapytania o bajt statusu, w celu oceny stanu linii READY dekompresora. Rozkaz ten wraz z odpowiedzą trwa 0,7ms (3 bajty, 3 bity przerwy, przepływność 38400 bps), a więc rzeczywisty czas dekompresji wynosi 15,3ms. Ponieważ mikrokontrolery z rdzeniem AVR wykonują jedną instrukcję w czasie jednego taktu sygnału zegarowego, a kontroler dekompresora taktowany jest sygnałem o częstotliwości 16MHz, szacunkowa liczba instrukcji w czasie 15,3ms wynosi ok. 244 tysięcy.



Rys. 6.1. Okno automatycznego sterowania stanami linii danych.

7. Uwagi końcowe

Celem eksperymentu było wykazanie, że dekompresor trzeciej generacji może zostać zaimplementowany w formie programu wykonywanego przez powszechnie dostępny, tani mikrokontroler. Cel ten został osiągnięty, choć zmierzone parametry czasowe mogą nie wydawać się zachęcające. Dekompresja przebiega relatywnie wolno – uzyskano ekwiwalent sprzętowego dekompresora taktowanego sygnałem o częstotliwości około 100Hz. Warto jednak zaznaczyć, że tak długi czas nie byłby zbyt wielkim narzutem, gdyby dekompresor zaimplementowano w formie programowej wprowadzanej do pamięci pracującego systemu mikroprocesorowego w celu okresowego testu jego działania. Potencjalnym zastosowaniem opracowanej implementacji może być programowa kontrola działania sterowników maszyn przemysłowych/ Program testujący mógłby zostać instalowany jako aktualizacja wewnętrznego oprogramowania. Mógłby funkcjonować także jako aplikacja na przenośnym nośniku, na przykład na karcie pamięci Flash.

Najważniejszym wnioskiem wyciągniętym z eksperymentu jest jednak spostrzeżenie, iż choć emulowana implementacja dekompresora danych testowych działała bez zarzutu, to z powodu liczby instrukcji procesora koniecznych do przetworzenia, aby wygenerować wyniki uzyskiwane w strukturze sprzętowej podczas jednego taktu zegara, nie można wykorzystać programowej implementacji do produkcyjnej kontroli sprawności wytworzonych układów. Programowa emulacja nie zastąpi dużo szybszej, choć wymagającej odpowiedniego projektowania, struktury sprzętowej.

8. Bibliografia

[1] J. Tyszer, M. Kassab, N. Mukherje J. Rajski, "Embedded deterministic test," *IEEE Transactions CAD*, vol. 23, 2004.

[2] J. Rajski, J. Tyszer G. Mrugalski, "Ring Generators: New devices for embedded test applications," *IEEE Transactions CAD*, vol. 23, 2004.

[3] Atmel. Atmega32. [Online]. http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf

[4] Fairchild. LM7805. [Online]. http://www.fairchildsemi.com/ds/LM/LM7805.pdf

[5] R. Pease, *Projektowanie układów analogowych*. Warszawa: BTC, 2005.

[6] MCS Electronics. Bascom. [Online]. http://www.mcselec.com/

[7] Wang Wu Wen, *VLSI test principles and architectures*. San Francisco: Ilsevier Inc., 2006.

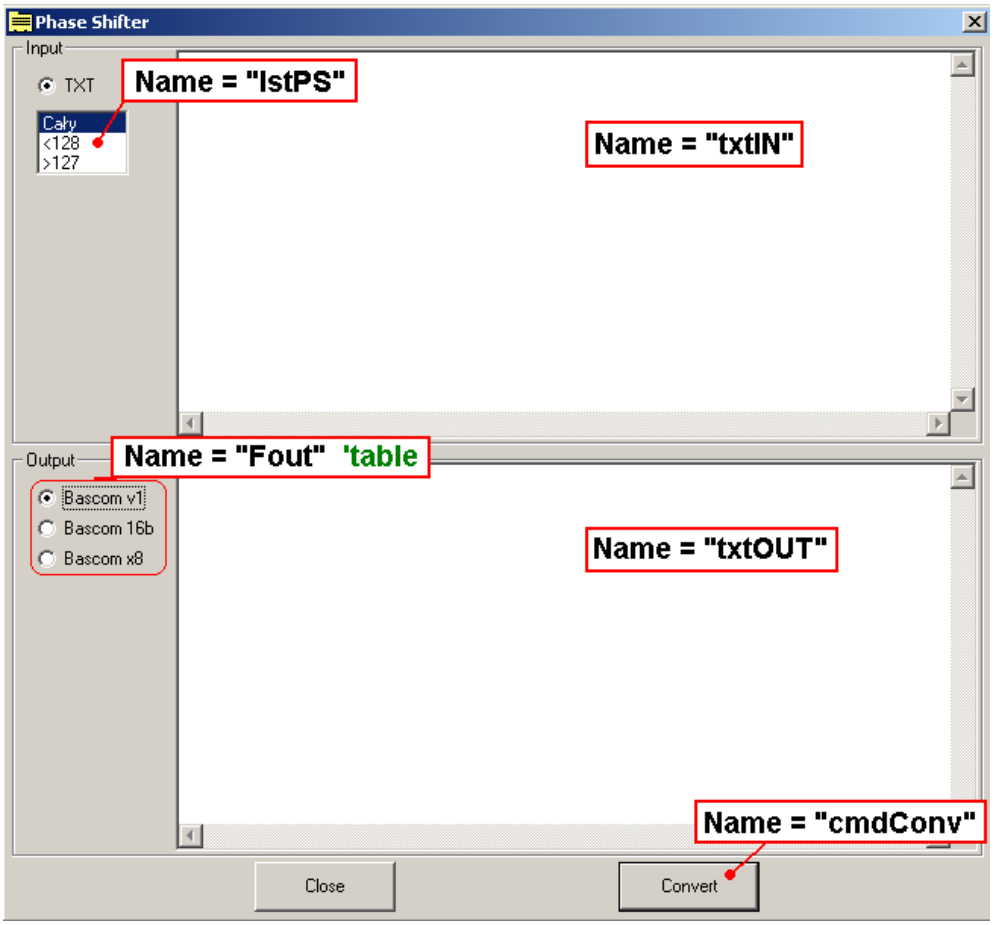
Zawartość płyty CD:

- Wersja elektroniczna niniejszej pracy (format PDF).
- Wszystkie wersje oprogramowania sterującego kontrolerami (kod źródłowy, HEX),
- Kod źródłowy, formatki oraz kontroli i moduły oprogramowania PC sterującego pracą dekompresora (zgodny z Microsoft Visual Studio 6.0),
- Plik wykonywalny programu PC oraz biblioteki VBRunTime niezbędne do jego uruchomienia pod kontrolą dowolnego systemu Windows (3.1. lub nowszego)

9. Dodatki i uzupełnienia

Dodatek A. Transformacja opisu przesuwnika fazy

Na rysunku 9.1 przedstawiono okno interfejsu użytkownika podprogramu, służącego do konwersji definicji przesuwnika fazy danego w pliku wyjściowym narzędzia kompresującego Mentor Graphics, do postaci zgodnej z algorytmem emulacji dekompresora, zaprogramowanym w języku Bascom Basic.



Rys. 9.1. Interfejs użytkownika podprogramu transformującego opis przesuwnika fazy

Treść programu w języku Visual Basic 6.0, skojarzonego z przedstawioną na rysunku 9.1 formatką jest następująca

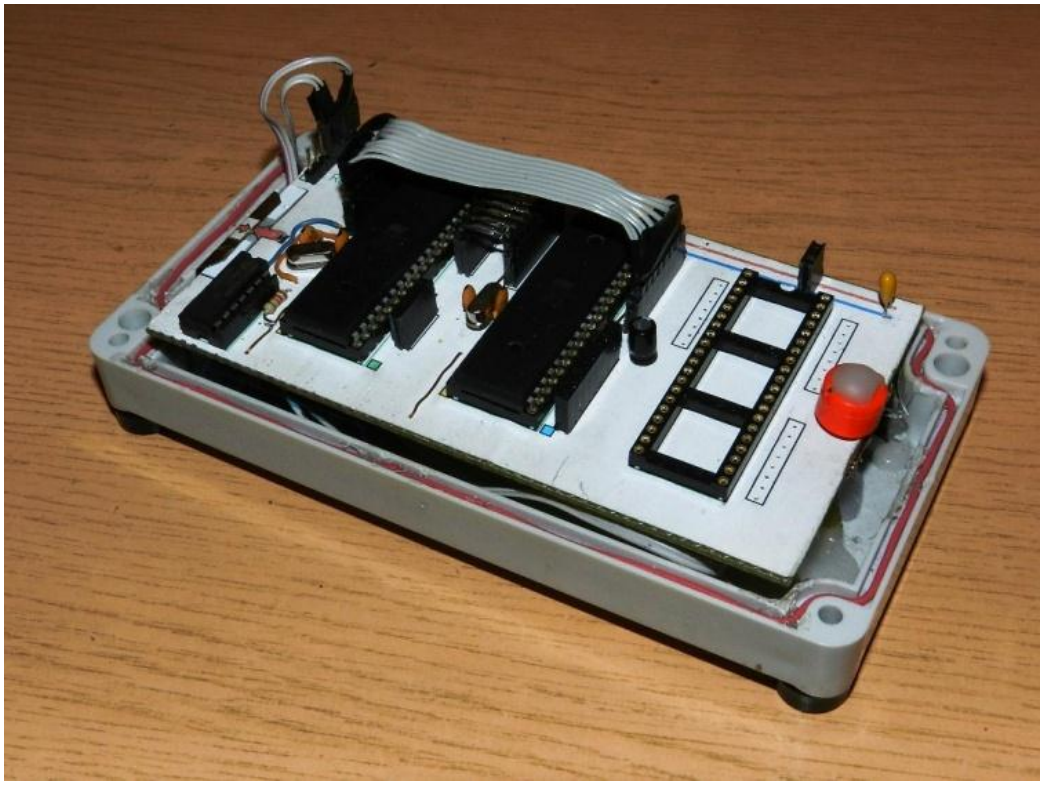
```
 'Globalne zmienne modułu klasy formatki konwertera
Dim Table(2048, 3) As Integer                  'Tablica PS (do konwersji)
Dim Size As Integer                             'Rozmiar PS (ilosc wyjsc)
Dim Inputs As Integer                           'Ilosc wejsc
```

```
Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdConv_Click()
    MakePSTable InFormat, txtIN
    txtOUT = PrintTable(OutFormat)
End Sub

'-----------------------------------------------
'Stworzenie tablicy Phase Shiftera
'-----------------------------------------------
Function MakePSTable(Format As Integer, Text As String)
Dim Temp, Aux As String
Dim I, IntAux, FindMax As Integer
Dim nInput, nOutput As Integer
Text = UCase(Text)
If Len(Text) > 16 And Left(Text, 14) = "PHASE SHIFTER:" Then
      Text = Right(Text, Len(Text) - 16)
End If
FindMax = 0
For I = 0 To 2048
    For II = 0 To 2
        Table(I, II) = "-1"
    Next II
Next I
On Error GoTo Error_
    Select Case Format                'Wybor formatu wejsciowego
    Case 0:                           'Format tekstowy (TXT)
      Do Until Len(Text) < 1
        Temp = GetLine(Text)
        If Len(Temp) + 2 < Len(Text) Then       'Wyciecie pierwszej linii
            Text = Right(Text, Len(Text) - Len(Temp) - 2)
        Else
            Text = ""
        End If
        Temp = Right(Temp, Len(Temp) - 1)       'Usun pierwszy znak "["
        nInput = 0
        Do Until Left(Temp, 1) = "]"
            nInput = (nInput * 10) + CInt(Left(Temp, 1))
            Temp = Right(Temp, Len(Temp) - 1)
        Loop
        Temp = Right(Temp, Len(Temp) - 3)
        Do Until Len(Temp) = 0
            nOutput = 0
            Do Until Left(Temp, 1) = " "
                nOutput = (nOutput * 10) + CInt(Left(Temp, 1))
                Temp = Right(Temp, Len(Temp) - 1)
            Loop
            If lstPS.ListIndex = 1 And nOutput > 127 Then: GoTo Pomin
            If lstPS.ListIndex = 2 And nOutput < 128 Then: GoTo Pomin
            If lstPS.ListIndex = 2 Then: nOutput = nOutput - 128
            If nOutput > FindMax Then: FindMax = nOutput
            I = 0
            Do While I < 2 And Table(nOutput, I) > -1
                I = I + 1
            Loop
            Table(nOutput, I) = nInput
Pomin:
            Temp = Right(Temp, Len(Temp) - 1)
        Loop
```

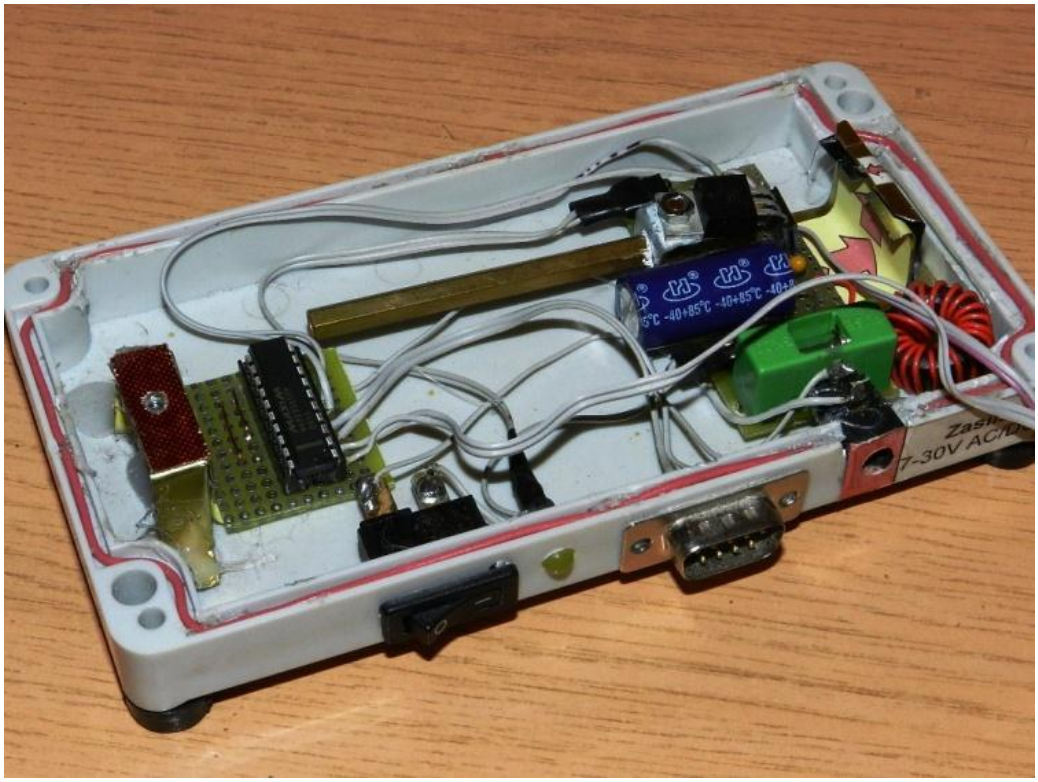
```
      Loop
      Size = FindMax + 1
    End Select
Exit Function
Error_:           'Gdy nie mozna wykonac tablicowania...
    MsgBox "Błąd formatu wejściowego!", vbCritical, "Błąd"
End Function

'-------------------------------------------------
'Drukowanie w txtOUT tablicy PS
'-------------------------------------------------
Function PrintTable(Format As Integer)
    On Error Resume Next
    Dim I, II, Wrd, Aux As Integer
    Dim strAux, txtAux As String
    txtAux = ""
    Select Case Format
    Case 0:                   'Bascom v1
        For I = 0 To (Size / 4 - 1)
          For Wrd = 0 To 3
            txtAux = txtAux & "Data "
            For II = 0 To 2
                If Table(Wrd * 32 + I, II) > -1 Then
                    txtAux = txtAux & Table(Wrd * 32 + I, II)
                Else
                    txtAux = txtAux & Table(Wrd * 32 + I, 0)
                End If
                If II < 2 Then
                    txtAux = txtAux & ", "
                Else
                    txtAux = txtAux & vbTab & "'Output " & _
                             Wrd * 32 + I & vbCrLf
                End If
            Next II
          Next Wrd
        Next I
    Case 1:                     'Bascom 16 bitów na wyjscie
        For I = 0 To (Size - 1) / 4
          For Wrd = 0 To 3
            txtAux = txtAux & "Data "
            Aux = 0
            For II = 0 To 2
                If Table(Wrd * 32 + I, II) > -1 Then
                    Aux = (Aux * (2 ^ 5)) + Table(Wrd * 32 + I, II)
                Else
                    Aux = (Aux * (2 ^ 5)) + Table(Wrd * 32 + I, 0)
                End If
            Next II
            strAux = CStr(Hex(Aux))
            Do While Len(strAux) < 4
                strAux = "0" & strAux
            Loop
            txtAux = txtAux & "&H" & strAux & "%"
            txtAux = txtAux & vbTab & "'Output " & Wrd * 32 + I & vbCrLf
          Next Wrd
        Next I
    Case 2:                      'Bascom - tablica(8)
        For I = 0 To (Size - 1) / 8
          For Wrd = 0 To 7
            txtAux = txtAux & "Data "
            For II = 0 To 2
```

```
                If Table(Wrd * 32 + I, II) > -1 Then
                    txtAux = txtAux & Table(Wrd * 32 + I, II)
                Else
                    txtAux = txtAux & Table(Wrd * 32 + I, 0)
                End If
                If II < 2 Then
                    txtAux = txtAux & ", "
                Else
                    txtAux = txtAux & vbTab & "'Output " & _
                             Wrd * 32 + I & vbCrLf
                End If
            Next II
          Next Wrd
        Next I
    End Select
    PrintTable = txtAux
End Function


'-------------------------------------------------
'Okreslenie formatu wejsciowego
'-------------------------------------------------
Function InFormat() As Integer
    InFormat = -1          'Blad - nie mozna okreslic formatu
    If Fin(0).Value = True Then: InFormat = 0   'Format tekstowy (TXT)
End Function

'-------------------------------------------------
'Okreslenie formatu wyjsciowego
'-------------------------------------------------
Function OutFormat() As Integer
    OutFormat = -1         'Blad - nie mozna okreslic formatu
    If Fout(0).Value = True Then: OutFormat = 0   'Format Bascom v1
    If Fout(1).Value = True Then: OutFormat = 1   'Format Bascom 16B
    If Fout(2).Value = True Then: OutFormat = 2   'Format Bascom x8
End Function

Private Sub Form_Load()
    lstPS.ListIndex = 0
End Sub
```

Dodatek B – fotografie części sprzętowej dekompresora



Rys. 9.2. Płyta bazowa z kontrolerami sterującym i dekompresującym, oraz komutator UART.



Rys. 9.3. Elementy zasilające i konwerter poziomów logicznych RS232 - TTL.

Dodatek C – obsługa transmisji od strony programu PC

Kod kontrolki realizującej sprzęg pomiędzy sprzętowym interfejsem RS232 komutera PC a oprogramowaniem sterującym:

```
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Dim busy As Boolean

Const NO_ACK = -1
Const NO_LINK = -2
Const LINK_ERROR = -3
Const DEVICE_ID = 11  'Identyfikator urządzenia
Const MASTER_ID = 12  'Identyfikator Master'a
Const SLAVE_ID = 13   'Identyfikator Slave'a
Const GET_DEVICE_ID = 0  'Rozkaz odeslania identyfikatora
Const GET_MODULE_ID = 1  'Rozkaz odeslania identyfikatora mikrokontrolera
Const RS232_FORWARD = 2  'Rozkaz Przekieruj RS232
Const RS232_TEST = 3     'Rozkaz testowy - odsyła argument zwiekszony + 1
Const RS232M_GET_STATUS = 4  'Dziala z modułem master;zwraca bajt statusu
Const RS232M_SET_PX = 5  'Rozkaz dla Master'a; Wystaw bajt na PX
Const RS232M_SET_PY = 6  'Rozkaz dla Master'a; Wystaw bajt na PY
Const RS232M_SEND_TO_SLAVE = 7
  'Rozkaz dla Master'a; Strobowanie - przerwanie transmisyjne do Slave'a
Const RS232S_GET_PX = 4  'Rozkaz dla Slave'a; zwroc stan PX
Const RS232S_GET_PY = 5  'Rozkaz dla Slave'a; zwroc stan PY
Const RS232S_GET_RING0 = 6  'Rozkaz dla Slave'a; zwroc stan Ring0
Const RS232S_GET_RING1 = 7  'Rozkaz dla Slave'a; zwroc stan Ring1
Const RS232S_GET_INCREMENTAL = 8
  'Rozkaz dla Slave'a; zwroc stan Incremental Pattern
Const RS232S_RESET_REGISTERS = 9  'Rozkaz dla Slave'a; reset dekompresora
Const RS232S_GET_SHADOW = 10    'Rozkaz dla Slave'a; Zwroc Shadow
Const RS232S_GET_CONFIG = 11    'Rozkaz dla Slave'a; Zwroc konfiguracje
Const RS232S_GET_PARENT = 12    'Rozkaz dla Slave'a; Zwroc Parent Pattern
Const RS232S_GET_CONTROL = 13   'Rozk. dla Slave'a; Zwroc Control Pattern
Const RS232S_GET_SCANS = 14     'Rozkaz dla Slave'a; Zwroc SCANS
Const RS232S_GET_CSUM = 15      'Rozkaz dla Slave'a; Zwroc CheckSum SCANS
Const RS232S_INPUT_RING0 = 16   'Rozkaz dla Slave'a; Wsun Ring0
Const RS232S_INPUT_RING1 = 17   'Rozkaz dla Slave'a; Wsun Ring1
Const RS232S_INPUT_SHADOW = 18  'Rozkaz dla Slave'a; Wsun Shadow
Const RS232S_INPUT_END = 19  'Rozk. dla Slave; Przelicz PS[0,1,2] i SCANS
Const RS232S_EEPRM_WRITE = 20
Const RS232S_EEPRM_READ = 21
Const RS232S_EEPRM_MSB = 22
Const RS232S_EEPRM_LSB = 23
Const INSTR_ARG_RESET = 255  'Rozkaz Reset flagi oczekiwania na argument

Dim CommNumber As Integer, PortOpen As Boolean

Private Sub UserControl_Initialize()
    CommNumber = 1
    PortOpen = False
End Sub

'-------------------------------------------------
' Szuka symulatora na portach od 1 do 16
'-------------------------------------------------
Function LinkToDevice() As Boolean
    CommNumber = 1
    busy = False
```

```
    Disconnect
    Dim Ack
NextTest:
        On Error GoTo NextFind   'Jesli nie mozna otworzyc portu...
        COM.CommPort = CommNumber
        If POpen = False Then: GoTo NextFind
        'Otwarto port. Sprawdz numer urzadzenia.
        Ack = COM.Input
        COM.Output = Chr(INSTR_ARG_RESET)
        COM.Output = Chr(GET_DEVICE_ID)
        COM.Output = Chr(GET_DEVICE_ID)
        PortOpen = True
        Ack = WaitAck(150)
        PortOpen = False
        If Ack <> DEVICE_ID Then
            COM.PortOpen = False
            GoTo NextFind
        End If
        LinkToDevice = True
        PortOpen = True
        SelectModule 0
        Exit Function
NextFind:
    CommNumber = CommNumber + 1
    If CommNumber < 17 Then: GoTo NextTest
    LinkToDevice = False
End Function

'-------------------------------------------------
' Oczekuje na odpowiedz (bajt) od urzadzenia
'-------------------------------------------------
Function WaitAck(Time_MS As Integer) As Integer
    Dim Aux As String
    If PortOpen = False Then
        WaitAck = NO_LINK
        Exit Function
    End If
    For I = 1 To Time_MS
        Sleep 1
        DoEvents
        If COM.InBufferCount > 0 Then
            Aux = COM.Input
            GoTo AreAck
        End If
    Next I
    WaitAck = NO_ACK
    Exit Function
AreAck:
    WaitAck = Asc(Aux)
End Function

'-------------------------------------------------
' Oczekuje na odpowiedz N-bajtowa od urzadzenia
'-------------------------------------------------
Function WaitAckN(Time_MS As Integer, Optional Size As Integer) As String
    If IsEmpty(Size) = True Or Size < 1 Then: Size = 1
    Dim Aux As String
    If PortOpen = False Then
        WaitAckN = "NO_LINK"
        Exit Function
    End If
```

```
    For I = 1 To Time_MS
        Sleep 1
        DoEvents
        If COM.InBufferCount >= Size Then
            Aux = COM.Input
            If Len(Aux) > Size Then: Aux = Right(Aux, Size)
            GoTo AreAck
        End If
    Next I
    WaitAckN = "NO_ACK"
    Exit Function
AreAck:
    WaitAckN = Aux
End Function


'-------------------------------------------------
' Otwiera (przechwytuje) port RS232
'-------------------------------------------------
Private Function POpen() As Boolean
POpen = False
PortOpen = False
On Error GoTo Blad
    COM.PortOpen = True
    POpen = True
    PortOpen = True
Blad:
End Function

'-------------------------------------------------
' Zwalnia port RS232
'-------------------------------------------------
Function Disconnect()
    On Error Resume Next
    SelectModule 0
    Sleep 100
    COM.PortOpen = False
    PortOpen = False
End Function

'-------------------------------------------------
' Wysyla rozkaz resetu flagi oczekiwania na argument
'-------------------------------------------------
Function InstrArgReset() As Boolean
    InstrArgReset = False
    If PortOpen = True Then
        COM.Output = Chr(INSTR_ARG_RESET)
        InstrArgReset = True
    End If
End Function

'-------------------------------------------------
' Wysyla instrukcje w postaci ROZKAZ, ARGUMENT
'-------------------------------------------------
Function SendToDevice(Instruction As Integer, Argument As Integer) As
Boolean
    SendToDevice = False
    If PortOpen = True Then
        COM.Output = Chr(Instruction)
        COM.Output = Chr(Argument)
        SendToDevice = True
```

```
    End If
End Function

'-------------------------------------------------
' Wysyla instrukcje w postaci ROZKAZ, ARGUMENT
' Zwraca odpowiedz od urzadzenia.
'-------------------------------------------------
Function SendAndRecv(Instruction As Integer, Argument As Integer, _
                             Miliseconds As Integer) As Integer
    Dim Old As String
    If PortOpen = True Then
        Old = COM.Input
        COM.Output = Chr(Instruction)
            If Not COM.Input = "" Then MsgBox "!!"
        COM.Output = Chr(Argument)
        SendAndRecv = WaitAck(Miliseconds)
    Else
        SendAndRecv = NO_LINK
    End If
End Function

'-------------------------------------------------
' Ustala, ktory z modulow aktualnie obsluguje RS232
' Zwraca:
' 0 - jesli Master,
' 1 - jesli Slave.
'-------------------------------------------------
Function WhoIsConnect() As Integer
    Dim Aux As Integer
    InstrArgReset
    Aux = SendAndRecv(GET_MODULE_ID, GET_MODULE_ID, 120)
    Select Case Aux
    Case MASTER_ID: WhoIsConnect = 0
    Case SLAVE_ID: WhoIsConnect = 1
    Case Else: WhoIsConnect = NO_ACK
    End Select
End Function

'-------------------------------------------------
' Przeprowadza jednorazowy test sprawnosci lacza
' dla podanego w argumencie bajtu.
'-------------------------------------------------
Function Test(ByVal Data As Integer) As Boolean
    Dim Aux As Integer
    Test = False
    Aux = SendAndRecv(RS232_TEST, Data, 150)
    Data = (Data + 1) Mod 256
    If Aux = Data Then: Test = True
End Function

'-------------------------------------------------
' Wymusza komutację lacza TS232 (w/g Module):
' 0 = do Mastter'a,
' 1 = do Slave'a.
'-------------------------------------------------
Function SelectModule(Module As Integer) As Boolean
Dim Aux
    If Module > 0 Then  'Wybierz Slave'a
        Select Case WhoIsConnect
        Case 0:
            InstrArgReset
```

```
            Sleep 1
            SendToDevice RS232_FORWARD, 0
            Sleep 1
            InstrArgReset
            Sleep 2
            InstrArgReset
            Aux = COM.Input
            If WhoIsConnect = 1 Then: SelectModule = True
        Case 1: SelectModule = True
        Case Else: SelectModule = False
        End Select
    Else                'Wybierz Mastera'a
        Select Case WhoIsConnect
        Case 0: SelectModule = True
        Case 1:
            InstrArgReset
            Sleep 1
            SendToDevice RS232_FORWARD, 0
            Sleep 1
            InstrArgReset
            Sleep 2
            InstrArgReset
            Aux = COM.Input
            If WhoIsConnect = 0 Then: SelectModule = True
        Case Else: SelectModule = False
        End Select
    End If
    Test 0
End Function

'-----------------------------------------------------
' Autotest. Ustawienie kolejnych bitow oznacza:
' 0 = blad polaczenia
' 1 = blad poprawnosci danych (Master)
' 2 = blad poprawnosci danych (Slave)
' 3 = blad komutacji M->S
' 4 = blad komutacji S->M
' 5 = blad portu X
' 6 = blad portu Y
'-----------------------------------------------------
Function BIST() As Integer
Dim EQ As Integer
EQ = 0
    If LinkToDevice = True Then
        SelectModule 0
        If Test(21) = False Then: EQ = EQ + 2
        If SelectModule(1) = False Then: EQ = EQ + 8
        If Test(21) = False Then: EQ = EQ + 4
        If SelectModule(0) = False Then: EQ = EQ + 16
        SetPX 0
        SetPY 0
        SelectModule 1
        If GetPX <> 0 Then: EQ = EQ + 32
        If GetPY <> 0 Then: EQ = EQ + 64
        SelectModule 0
    Else
        EQ = EQ + 1
    End If
BIST = EQ
End Function
```

```
'================ TYLKO MASTER ====================

'-------------------------------------------------
' Zwraca status modulu Master
'-------------------------------------------------
Function GetMasterStatus() As Integer
    If SelectModule(0) = True Then
        GetMasterStatus = SendAndRecv(RS232M_GET_STATUS, _
                                      RS232M_GET_STATUS, 100)
    Else
        GetMasterStatus = LINK_ERROR
    End If
End Function

'-------------------------------------------------
' Wystawia bajt na PX
'-------------------------------------------------
Function SetPX(DataByte As Integer) As Boolean
    SetPX = SendToDevice(RS232M_SET_PX, DataByte)
End Function

'-------------------------------------------------
' Wystawia bajt na PY
'-------------------------------------------------
Function SetPY(DataByte As Integer) As Boolean
    SetPY = SendToDevice(RS232M_SET_PY, DataByte)
End Function

'-------------------------------------------------
' Wysyła do Slave'a
'-------------------------------------------------
Function SendToSlave(Mode As Integer) As Boolean
    SendToSlave = SendToDevice(RS232M_SEND_TO_SLAVE, Mode)
End Function

'-------------------------------------------------
' czeka az Slave bbedzie gotowy
'-------------------------------------------------
Function WaitReady() As Boolean
    SelectModule 0
    Dim I As Long
    For I = 1 To 90000000
        If (GetMasterStatus Mod 2) = 0 Then: GoTo Dalej
    Next I
    WaitReady = False
Exit Function
Dalej:
    WaitReady = True
End Function

'-------------------------------------------------
' czeka az Slave bbedzie gotowy, zwraca status
'-------------------------------------------------
Function WaitReadyS() As Integer
Dim Aux As Integer
    SelectModule 0
    Dim I As Long
    For I = 1 To 90000000
        Aux = GetMasterStatus
        If (Aux Mod 2) = 0 Then: GoTo Dalej
```

```
    Next I
    WaitReadyS = -1
Exit Function
Dalej:
    WaitReadyS = Aux
End Function

'================ TYLKO SLAVE ====================

'-------------------------------------------------
' Zwraca stan PX
'-------------------------------------------------
Function GetPX() As Integer
    GetPX = SendAndRecv(RS232S_GET_PX, 0, 120)
End Function

'-------------------------------------------------
' Zwraca stan PY
'-------------------------------------------------
Function GetPY() As Integer
    GetPY = SendAndRecv(RS232S_GET_PY, 0, 120)
End Function

'-------------------------------------------------
' Zwraca stan Ring0
' Size = bajtów / slowo; Words = liczba slow
'-------------------------------------------------
Function GetRing0(Size As Integer, Optional Words As Integer) As String
Dim Word As Integer, Aux As String
Dim I As Integer, X As String, A As String
    If IsEmpty(Words) = True Or Words < 1 Then: Words = 1
    On Error Resume Next
  For Word = 1 To Words
    SendToDevice RS232S_GET_RING0, Word
    A = ""
    X = WaitAckN(200, Size)
    If X = "NO_ACK" Or X = "NO_ACK" Then
        GetRing0 = "NO_ACK"
    Else
        For I = Len(X) To 1 Step -1
            A = A & Mid(X, I, 1)
        Next I
        Aux = A & Aux
    End If
  Next Word
GetRing0 = Aux
End Function

'-------------------------------------------------
' Zwraca stan Ring1
' Size = bajtów / slowo; Words = liczba slow
'-------------------------------------------------
Function GetRing1(Size As Integer, Optional Words As Integer) As String
Dim Word As Integer, Aux As String
Dim I As Integer, X As String, A As String
    If IsEmpty(Words) = True Or Words < 1 Then: Words = 1
    On Error Resume Next
  For Word = 1 To Words
    SendToDevice RS232S_GET_RING1, Word
    A = ""
    X = WaitAckN(200, Size)
```

```
    If X = "NO_ACK" Or X = "NO_ACK" Then
        GetRing1 = "NO_ACK"
    Else
        For I = Len(X) To 1 Step -1
            A = A & Mid(X, I, 1)
        Next I
        Aux = A & Aux
    End If
  Next Word
GetRing1 = Aux
End Function

'-------------------------------------------------
' Reset dekompresora
'-------------------------------------------------
Function DekomprReset() As Boolean
    DekomprReset = SendToDevice(RS232S_RESET_REGISTERS, 0)
End Function

'-------------------------------------------------
' Zwraca stan PhaseShifter1 (Incremental Pattern)
' Size = bajtów / slowo; Words = liczba slow
'-------------------------------------------------
Function GetIncremental(Size As Integer, Words As Integer) As String
Dim Word As Integer, Aux As String
Dim I As Integer, X As String, A As String
On Error Resume Next
  For Word = 1 To Words
    SendToDevice RS232S_GET_INCREMENTAL, Word
    A = ""
    X = WaitAckN(200, Size)
    If X = "NO_ACK" Or X = "NO_ACK" Then
        GetIncremental = "NO_ACK"
        Exit Function
    Else
        For I = Len(X) To 1 Step -1
            A = A & Mid(X, I, 1)
        Next I
        Aux = A & Aux
    End If
  Next Word
GetIncremental = Aux
End Function

'-------------------------------------------------
' Zwraca stan Shadow
' Size = bajtów / slowo; Words = liczba slow
'-------------------------------------------------
Function GetShadow(Size As Integer, Optional Words As Integer) As String
Dim Word As Integer, Aux As String
Dim I As Integer, X As String, A As String
    If IsEmpty(Words) = True Or Words < 1 Then: Words = 1
    On Error Resume Next
  For Word = 1 To Words
    SendToDevice RS232S_GET_SHADOW, Word
    A = ""
    X = WaitAckN(200, Size)
    If X = "NO_ACK" Or X = "NO_ACK" Then
        GetShadow = "NO_ACK"
    Else
        For I = Len(X) To 1 Step -1
```

```
            A = A & Mid(X, I, 1)
        Next I
        Aux = A & Aux
'        Sleep 2
    End If
  Next Word
GetShadow = Aux
End Function

'-------------------------------------------------
' Zwraca bajt konfiguracji
'-------------------------------------------------
Function GetConfig(Word As Integer) As Integer
    GetConfig = SendAndRecv(RS232S_GET_CONFIG, Word, 120)
End Function

'-------------------------------------------------
' Zwraca stan Parent Pattern
' Size = bajtów / slowo; Words = liczba slow
'-------------------------------------------------
Function GetParent(Size As Integer, Words As Integer) As String
Dim Word As Integer, Aux As String
Dim I As Integer, X As String, A As String
On Error Resume Next
  For Word = 1 To Words
    SendToDevice RS232S_GET_PARENT, Word
    A = ""
    X = WaitAckN(200, Size)
    If X = "NO_ACK" Or X = "NO_ACK" Then
        GetParent = "NO_ACK"
        Exit Function
    Else
        For I = Len(X) To 1 Step -1
            A = A & Mid(X, I, 1)
        Next I
        Aux = A & Aux
'        Sleep 2
    End If
  Next Word
GetParent = Aux
End Function

'-------------------------------------------------
' Zwraca stan Control Pattern
' Size = bajtów / slowo; Words = liczba slow
'-------------------------------------------------
Function GetControl(Size As Integer, Words As Integer) As String
Dim Word As Integer, Aux As String
Dim I As Integer, X As String, A As String
On Error Resume Next
  For Word = 1 To Words
    A = COM.Input
    If Not A = "" Then MsgBox "!"
    SendToDevice RS232S_GET_CONTROL, Word
    A = ""
    X = WaitAckN(200, Size)
    If X = "NO_ACK" Or X = "NO_ACK" Then
        GetControl = "NO_ACK"
        Exit Function
    Else
        For I = Len(X) To 1 Step -1
```

```
            A = A & Mid(X, I, 1)
        Next I
        Aux = A & Aux
'        Sleep 2
    End If
  Next Word
GetControl = Aux
End Function

'-------------------------------------------------
' Zwraca stan SCANS
' Size = bajtów / slowo; Words = liczba slow
'-------------------------------------------------
Function GetSCANS(Size As Integer, Words As Integer) As String
Dim Word As Integer, Aux As String
Dim I As Integer, X As String, A As String
On Error Resume Next
  For Word = 1 To Words
    SendToDevice RS232S_GET_SCANS, Word
    A = ""
    X = WaitAckN(200, Size)
    If X = "NO_ACK" Or X = "NO_ACK" Then
        GetSCANS = "NO_ACK"
        Exit Function
    Else
        For I = Len(X) To 1 Step -1
            A = A & Mid(X, I, 1)
        Next I
        Aux = A & Aux
'        Sleep 2
    End If
  Next Word
GetSCANS = Aux
End Function

'-------------------------------------------------
' Zwraca stan sumy kontrolnej (bajt) SCANS
'-------------------------------------------------
Function GetCheckSum() As Integer
    GetCheckSum = SendAndRecv(RS232S_GET_CSUM, 0, 200)
End Function

'-------------------------------------------------
' Wsuń bajt do Ring0
'-------------------------------------------------
Function InputRing0(Data As Integer) As Boolean
    InputRing0 = SendToDevice(RS232S_INPUT_RING0, Data)
End Function

'-------------------------------------------------
' Wsuń bajt do Ring1
'-------------------------------------------------
Function InputRing1(Data As Integer) As Boolean
    InputRing1 = SendToDevice(RS232S_INPUT_RING1, Data)
End Function

'-------------------------------------------------
' Wsuń bajt do Shadow
'-------------------------------------------------
Function InputShadow(Data As Integer) As Boolean
    InputShadow = SendToDevice(RS232S_INPUT_SHADOW, Data)
```

```
End Function

'-------------------------------------------------
' Przelicz PS[0,1,2], SANS
'-------------------------------------------------
Function InputEnd() As Boolean
    InputEnd = SendToDevice(RS232S_INPUT_END, 0)
End Function

'-------------------------------------------------
' Wczytuje dane do Ring0
'-------------------------------------------------
Function WriteRing0(HexData As String) As Boolean
Dim I As Integer, Success As Boolean, DataByte As Integer
    Success = True
    If Len(HexData) Mod 2 = 1 Then: HexData = "0" & HexData
    For I = 1 To Len(HexData) Step 2
        DataByte = HexToByte(Mid(HexData, I, 2))
        Success = Success And InputRing0(DataByte)
    Next I
    WriteRing0 = Success
End Function

'-------------------------------------------------
' Wczytuje dane do Ring1
'-------------------------------------------------
Function WriteRing1(HexData As String) As Boolean
Dim I As Integer, Success As Boolean, DataByte As Integer
    Success = True
    If Len(HexData) Mod 2 = 1 Then: HexData = "0" & HexData
    For I = 1 To Len(HexData) Step 2
        DataByte = HexToByte(Mid(HexData, I, 2))
        Success = Success And InputRing1(DataByte)
    Next I
    WriteRing1 = Success
End Function

'-------------------------------------------------
' Wczytuje dane do Shadow
'-------------------------------------------------
Function WriteShadow(HexData As String) As Boolean
Dim I As Integer, Success As Boolean, DataByte As Integer
    Success = True
    If Len(HexData) Mod 2 = 1 Then: HexData = "0" & HexData
    For I = 1 To Len(HexData) Step 2
        DataByte = HexToByte(Mid(HexData, I, 2))
        Success = Success And InputShadow(DataByte)
    Next I
    WriteShadow = Success
End Function

Function ReadEerpom() As Integer
    Sleep 10
    ReadEerpom = SendAndRecv(RS232S_EEPRM_READ, 0, 200)
    Sleep 10
End Function

Function WriteEerpom(Data As Integer) As Boolean
    Sleep 10
    WriteEerpom = SendToDevice(RS232S_EEPRM_WRITE, Data)
    Sleep 10
```

```
End Function

Function EepromLsb(Data As Integer) As Boolean
    Sleep 10
    EepromLsb = SendToDevice(RS232S_EEPRM_LSB, Data)
    Sleep 10
End Function

Function EepromMsb(Data As Integer) As Boolean
    Sleep 10
    EepromMsb = SendToDevice(RS232S_EEPRM_MSB, Data)
    Sleep 10
End Function

Function EepromAddress(Data As Integer)
    Dim M As Integer, S As Integer
    S = Data Mod 256
    M = Data / 256
    EepromMsb M
    EepromLsb S
End Function
```