



**DOKUMENTACJA BIBLIOTEKI DLL DLA
ŚRODOWISKA MICROSOFT WINDOWS
MODUŁÓW CYFROWYCH
LC-055-PIO I LC-055-DCU**

WERSJA 1.1

Wersja: maj 1996

1. Informacje ogólne.....	3
2. Instalacja biblioteki.....	3
3. Opis biblioteki.....	3
3.1. Informacje wstępne.....	3
3.2. Standardy oznaczeń, numeracja, dane charakterystyczne.....	4
3.2.1. Uwagi dotyczące instalacji modułów.....	4
3.3. Komunikacja z biblioteką.....	5
4. Funkcje biblioteki.....	5
4.1. Inicjalizacja (MODULE_INIT_55).....	6
4.2. Informacja o konfiguracji ogólnej (GET_TOTAL_CONFIGURATION_55).....	6
4.3. Informacja o konfiguracji modułu (GET_MODULE_CONFIGURATION_55).....	7
4.4. Obsługa portów.....	9
4.4.1. Programowanie kierunku pracy (PORT_DIRECTION).....	9
4.4.2. Zapis do portu (PORT_WRITE).....	9
4.4.3. Odczyt z portu (PORT_READ).....	10
4.4.4. Zatrzaśnięcie danych (PORT_LATCH).....	10
4.4.5. Zerowanie portów (PORT_RESET).....	11
4.5. Obsługa układu czasowego 8253/8254.....	11
4.5.1. Programowanie i/lub ładowanie licznika (CTC_WRITE_55).....	11
4.5.2. Bramkowanie (CTC_GATE).....	12
4.5.3. Odczyt kanału CTC (CTC_READ_55).....	12
4.6. Obsługa przerw.....	13
4.6.1. Maskowanie przerw (INTERRUPT_MASK).....	13
4.6.2. Poziom sygnału przerywającego (INTERRUPT_LEVEL).....	14
4.6.3. Obsługa przerw (INTERRUPT_SERVICE_55).....	15
4.6.4. Kasowanie zgłoszenia przerwania (INTERRUPT_RESET).....	15
4.7. Odczyt statusu (READ_STATUS).....	16
4.17. Zakończenie pracy z driver'em (LEAVE_DRIVER_55).....	16
5. Zestawienie kodów zakończenia funkcji.....	17
6. Projektowanie programów użytkowych.....	18
6.1. Programowanie w języku C.....	18
6.2. Programowanie w języku Pascal.....	18

DODATEK A - struktury danych i stałe dla języka C

DODATEK B - program przykładowy w C

DODATEK C - struktury danych i stałe dla Pascala

DODATEK D - program przykładowy w Pascalu

DODATEK E - struktura pliku konfiguracyjnego AMBEX.INI

1. Informacje ogólne.

Przy tworzeniu oprogramowania modułów w środowisku MS WINDOWS przyjęto zasadę, że cała komunikacja z modulem prowadzona jest za pośrednictwem rezydującego w pamięci modułu w formie dynamicznie linkowanej biblioteki DLL (*Dynamic Link Library*) dostępnej dla aplikacji Windows poprzez wywołania eksportowalnych funkcji publicznych. Takie rozwiązanie jest praktycznie zawsze stosowane w środowisku Windows w przypadku programów obsługi urządzeń z uwagi na właściwości bibliotek DLL różniącą je od aplikacji: dostępność dla wszystkich modułów pracujących jako klient i możliwość pozostawiania w pamięci przez czas dostarczania obsługi aplikacjom.

Rozwiązanie to ma następujące zalety:

- biblioteka stanowi wirtualny pomost pomiędzy kartą a programem użytkownika, realizując w sposób przejrzysty zlecenia ze strony aplikacji klienckich; jest dostępna dla wszystkich aplikacji i system gwarantuje jej obecność w pamięci dopóki aktywne są programy z niej korzystające,
- użytkownik jest zwolniony ze znajomości szczegółów technicznych tak modułu jak i używanego komputera oraz tajników programowania systemowego w środowisku Windows,
- rozwiązanie to jest niezależne od używanej implementacji języka wyższego poziomu.

Program obsługi został napisany w standardzie 16-bitowych bibliotek DLL środowiska programowego MS WINDOWS (wersja 3.1. i wyższe). Główną przyczyną wyboru takiego rozwiązania jest możliwość rozszerzenia zastosowania kart pomiarowych serii LC na atrakcyjne pod względem zastosowań pomiarowym środowisko programowe MS Windows, w którym możliwa jest implementacja procesów wielozadaniowych oraz istnieje jednolity dostęp do całej pamięci systemu. Wymienione cechy jak i łatwość tworzenia przyjaznych użytkownikowi interfejsów graficznych stwarzają nowe możliwości realizacji programów pomiarowych z zaawansowanymi technikami obrazowania i obróbki wyników w porównaniu do jednozadaniowego i zorientowanego tekstowo środowiska MS DOS.

Driver'y (biblioteki DLL) modułów serii LC podporządkowane zostały standardowi przyjętemu przez firmę. W standardzie tym przyjęto, że w komputerze może być zainstalowanych kilka (do ośmiu) modułów danego rodzaju obsługiwanych przez jeden driver.

Opisywana biblioteka obsługuje do 8 modułów wejść/wyjść cyfrowych. Każdy z nich może być dowolnego typu: LC-055-PIO lub LC-055-DCU.

Interfejs komunikacyjny z biblioteką jest zaprojektowany z myślą o zachowaniu jak najdalej idącej kompatybilności z interfejsem dla driverów DOS modułów serii LC. Zdecydowana większość struktur danych poszczególnych funkcji biblioteki odpowiada analogicznym strukturom driverów DOS, natomiast w pozostałych wprowadzono niezbędne zmiany z uwagą na specyfikę nowego środowiska.

2. Instalacja biblioteki.

Instalacja polega na skopiowaniu z dyskietki do katalogu docelowego:

- zbioru LC055A.DLL (biblioteka DLL modułów wejść/wyjść cyfrowych),
- zbioru AMBEX.INI, który określa konfigurację zainstalowanych w komputerze modułów analogowych i cyfrowych serii LC.

Podane zbiory można skopiować do następujących katalogów na dysku:

- katalogu startowego WINDOWS, gdzie znajduje się plik win.com,
- katalogu systemowego WINDOWS, gdzie znajdują się zbiory driverów systemowych i fontów - standardowo podkatalog \SYSTEM,
- katalogu bieżącego aplikacji,
- dowolnego katalogu, znajdującego się na systemowej ścieżce przeszukiwań PATH wyszczególnionego w sekcji PATH pliku autoexec.bat.

Poszukiwanie obydwu zbiorów odbywa się w wymienionej powyżej kolejności katalogów. W przyszłości będzie dostępny program dokonujący automatycznej instalacji i konfiguracji modułu razem z uaktualnieniem pliku konfiguracyjnego.

Plik konfiguracyjny jest wspólny dla wszystkich rodzajów modułów analogowych i cyfrowych produkowanych przez firmę AMBEX. Określa on wszystkie parametry dla każdego z modułów niezbędne dla jego prawidłowej inicjacji, która ma miejsce w momencie wczytania biblioteki do pamięci. Poszczególne pozycje w pliku AMBEX.INI odpowiadają ustawieniom przełączników konfiguracyjnych (adres bazowy, nr przerwania, ...) danego modułu. Pełny opis struktury pliku konfiguracyjnego AMBEX.INI dla modułów cyfrowych zawarty jest w dodatku E.

3. Opis biblioteki.

3.1. Informacje wstępne.

Przed driver'em postawione zostały następujące zadania:

- pełne wykorzystanie możliwości sprzętowych oferowanych przez obsługiwane moduły,

- realizacja pewnych funkcji, dzięki którym możliwe jest pisanie uniwersalnych programów, niezależnych od instalacji konkretnego modułu,

- dostarczenie możliwości wielozadaniowej pracy w systemie polegającej zarówno na jednoczesnym dostępie do wielu zainstalowanych modułów przez różne aplikacje jak i obsłudze wielu modułów przez jedną aplikację; w czasie pracy istnieje możliwość dynamicznego przydziału i zwalniania wybranych modułów przez poszczególne aplikacje.

Driver rozpoczyna swoją pracę w momencie załadowania przez system do pamięci, co następuje wówczas, gdy jakaś aplikacja WINDOWS poprzez swój kod odwołuje się do funkcji drivera.

Wówczas to pobierane i analizowane są parametry instalacji podane w pliku konfiguracyjnym AMBEX.INI. Następnie wykonywane jest tzw. twarde zerowanie wszystkich zadeklarowanych modułów, w trakcie którego wykonywane jest wstępne programowanie m.in. ustawienie wszystkich portów na kierunek "wejście".

Driver jest do pewnego stopnia wielowejściowy. Oznacza to, że w trakcie wykonywania jednej z funkcji driver'a można wykonać inną. Sytuacja taka może mieć miejsce w przypadku procedur obsługi przzerwania sprzętowego modułu, w których wywoływane są funkcje drivera, a sterowanie przed przełączeniem się do procedury obsługi było w kodzie zawartym w bibliotece DLL. Oczywiście jakie funkcje mogą być wykonana a jakie nie w tym przypadku zależy od wzajemnych korelacji pomiędzy funkcjami np. w trakcie wykonywania funkcji odczytu portów wejściowych nie można wykonać funkcji zmiany kierunku portów cyfrowych karty, bo w oczywisty sposób mogło by to zaburzyć odczyt danych z portów. Natomiast bez żadnych ograniczeń w tym przypadku może być wykonana operacja nie odnosząca się do portów cyfrowych np. zaprogramowanie wybranych kanałów CTC. W przypadku zaistnienia wyżej opisanych konfliktów driver zwraca błąd: LC0_DEV_BUSY.

Po wczytaniu do pamięci driver jest dostępny dla wszystkich aplikacji i zostanie usunięty z pamięci wówczas, gdy ostatnia aplikacja kliencka zakończy pracę lub, w przypadku tzw. wiązania dynamicznego, wywoła funkcję API Windows FreeLibrary().

Driver'y modułów cyfrowych produkowanych przez firmę AMBEX zostały zaprojektowane w sposób jednolity. Dzięki temu, jeżeli tylko program nie korzysta jawnie z cech czy funkcji modułu specyficznych tylko dla niego, to może być bez zmiany wykorzystywany do współpracy z różnymi typami modułów. Oczywiście z powodu takich założeń pewne funkcje czy tryby pracy driver'a są dostępne dla jednych typów modułów, dla innych - nie.

3.2. Standardy oznaczeń, numeracja, dane charakterystyczne.

Przy pisaniu tak oprogramowania jak i niniejszej dokumentacji przyjęto następujące zasady:

- wszystkie nazwy pól rekordów, stałych itp. (z wyjątkiem nazw funkcji) opatrzone są przedrostkiem LC0_;
- nazwy występujące w dokumentacji są identyczne z nazwami występującymi w plikach źródłowych dla języków C, Pascal (z dokładnością do rozróżnienia małe / duże litery).

Wszystkie wejścia, wyjścia, kanały itp. numerowane są od 1. Wyjątkiem jest numeracja kanałów CTC (ze względu na odniesienie do dokumentacji układu 8253/8254).

Driver LC055A.DLL widziany jest w systemie jako moduł LC055A.

3.2.1. Uwagi dotyczące instalacji modułów.

Kodowanie numerów modułów oraz adresy bazowe modułów:

moduł	nazwa kodu	wartość	adres bazowy [Hex]
A	LC0_MODA	1	220
B	LC0_MODB	2	620
C	LC0_MODC	3	300
D	LC0_MODAL	4	700
E	LC0_MODE	5	definiowany
F	LC0_MODF	6	definiowany
G	LC0_MODG	7	definiowany
H	LC0_MODAL	8	definiowany

Driver może obsługiwać jednocześnie do 8 modułów zainstalowanych w komputerze. Adresy bazowe pierwszych czterech modułów muszą być zgodne z ich domyślnymi wartościami (patrz instrukcja obsługi modułu) i ustawieniami przełącznika adresu modułu na karcie. Dalsze 4 moduły wymagają właściwego ustawienia adresu bazowego karty z zakresu I/O 0-1FFF H (przełącznik DS1). Zajętość przestrzeni adresowej I/O dla modułów LC-055-PIO i LC-055-DCU wynosi 28 bajtów. Przestrzeń adresowa modułów dodatkowych nie może być zajęta przez inne urządzenia w systemie.

Adres bazowy każdego modułu jest podawany driver'owi w pliku konfiguracyjnym: parametr "Base_Addres".

W przypadku korzystania z przerw sprzętowych modułu możliwe jest ustawienie przerwania modułu dowolnego dozwolonego za pomocą przełączników konfiguracyjnych za wyjątkiem przerwania nr 2 (kaskada 8259). Przed instalacją i konfiguracją układu przerw należy sprawdzić, czy wykorzystywana linia przerwania nie jest zajęta przez inne urządzenia w systemie.

3.3. Komunikacja z biblioteką.

Do komunikacji z biblioteką służy grupa publicznych i eksportowalnych funkcji driver'a, które mogą być wywoływane przez inne aplikacje lub biblioteki DLL WINDOWS. Funkcje drivera są eksportowane zarówno przez nazwę jak i unikalny numer porządkowy. Numery porządkowe poszczególnych funkcji są równe co do wartości numerowi pola LC0_CODE rekordu opisu zlecenia (opis poniżej). Funkcje w module mogą być osiągalne zarówno przez nazwę jak, numer porządkowy jak i obie te metody. Istnieją trzy sposoby na wczytanie driver'a (biblioteki DLL), które zarazem określają sposób dostępu do funkcji publicznych biblioteki:

1. wczytanie domyślne - niejawne poprzez umieszczenie biblioteki importowalnej (.LIB) w linii poleceń programu linkującego dla wymaganego modułu; bibliotekę importowalną można uzyskać za pomocą odpowiednich programów narzędziowych (np. implib.exe) z pliku driver'a (.DLL); w celu wywołania żądanej funkcji driver'a należy posłużyć się nazwą z odpowiedniego pliku nagłówkowego (patrz dodatek A, C), który należy dołączyć do tworzonego programu,
 2. wczytanie jawne na żądanie poprzez zadeklarowanie ich w pliku definiującym moduł (.DEF) danej aplikacji: sekcja IMPORTS; można posłużyć się zarówno nawami funkcji jak i ich numerami porządkowymi; wywoływanie funkcji w programie odbywa się w sposób analogiczny jak przy wczytaniu domyślnym,
 3. wczytanie dynamiczne w module źródłowym poprzez wywołanie funkcji API Windows LoadLibrary("nazwa zbioru driver'a"); wskazania do funkcji driver'a można uzyskać za pomocą odpowiednich funkcji API np. GetProcAddress(); W momencie zakończenia programu należy jawnie zwolnić bibliotekę wywołując funkcję FreeLibrary().
- Podane sposoby, typowe dla języka C mają swoje odpowiedniki dla innych języków programowania.

Przesyłanie informacji pomiędzy programem użytkowym a driver'em odbywa się poprzez rekord opisu zlecenia i stanowi parametr wywołania funkcji. Rekord ten służy do przekazywania informacji zarówno do jak i od driver'a.

Rekord opisu zlecenia ma strukturę zależną od rodzaju zlecenia. Jedynie trzy pierwsze pola są niezmiennie i mają następujące znaczenie:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji
LC0_STATUS	1	kod zakończenia funkcji

LC0_CODE określa funkcję jaka ma być wykonana przez driver a zarazem sposób interpretacji ciągu bajtów znajdujących się w kolejnych polach rekordu.

LC0_STATUS informuje program wywołujący driver o poprawności wykonania funkcji:

LC0_STATUS = 0: wykonanie poprawne

LC0_STATUS < 0: wykonanie błędne

LC0_STATUS > 0: wykonanie poprawne z zastrzeżeniami (ostrzeżenia)

W rozdziale 5 podano tabele wszystkich kodów zwracanych przez LC0_STATUS.

4. Funkcje biblioteki.

W poniższych rozdziałach opisano wszystkie funkcje biblioteki. Każdy rozdział ma następującą strukturę:

- oryginalna nazwa eksportowanej funkcji z pliku definicji modułu driver'a (.DEF) oraz numer porządkowy funkcji;
- tabela zawierająca strukturę rekordu opisu zlecenia; w tabeli tej opisano każde pole rekordu w sposób następujący:
- nazwa pola; nazwa ta używana jest konsekwentnie w plikach źródłowych dotyczących języka C, Pascal (patrz rozdz. 6)
- rozmiar w bajtach; typ danej reprezentowanej przez to pole (np. czy jest to liczba ze znakiem czy bez) wynika ze znaczenia pola; w razie wątpliwości należy porównać z odpowiednim dla danego języka plikiem źródłowym deklarującym strukturę danych (dodatki A, C i E)
- znaczenie
- przeznaczenie funkcji
- szczegółowy opis parametrów funkcji (pół rekordu opisu zlecenia); ten punkt został zamieszczony tylko wtedy, gdy uznano, że znaczenie parametru podane w tabeli jest niewystarczająco oczywiste
- ostrzeżenia; lista ostrzeżeń zwracanych przez funkcję w parametrze LC0_STATUS; jeżeli punkt ten nie występuje to oznacza to, że dana funkcja nie zwraca żadnych ostrzeżeń
- błędy; lista błędów zwracanych przez funkcję w parametrze LC0_STATUS; jeżeli punkt ten nie występuje to oznacza to, że dana funkcja nie zwraca żadnych błędów.

4.1. Inicjalizacja (MODULE_INIT_55).

Nazwa funkcji: LC055_MODULEINIT; nr porządkowy: 16

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 16
LC0_STATUS	1	kod zakończenia funkcji
LC0_IMODULE	1	mapa modułów

Przeznaczenie:

Funkcja powoduje zainicjalizowanie pracy (zerowanie) wyspecyfikowanych modułów. W przypadku pomyślnej inicjacji funkcja przydziela wywołującej aplikacji dostęp do zainicjowanych modułów. Moduły podlegające inicjalizacji specyfikuje się na poszczególnych bitach parametru LC0_IMODULE:

```

b8                b1
H G F E D C B A
x x x x x x x x

```

x = 1 -inicjacja, zerowanie, rezerwacja modułu; x = 0 - brak akcji

UWAGA:

Aplikacja, która uzyskała dostęp do modułu (modułów), może bez żadnych ograniczeń z nich korzystać aż do momentu wywołania funkcji LEAVE_DRIVER, która kończy rezerwację modułów. Zwolnione moduły są odtąd dostępne dla innych programów. Wszystkie funkcje, które analizują parametr numer modułu (również w warunkach startu) po stwierdzeniu, że moduł jest zainicjowany przez inną aplikację, zwracają błąd LC0_REJECTED.

Ostrzeżenia (LC0_STATUS):

LC0_NON_EX_MOD - zażądano inicjalizacji nie istniejących modułów ale co najmniej 1 moduł został zainicjalizowany i zarezerwowany dla bieżącej aplikacji.

LC0_IS_INIT - zażądano inicjacji modułów, które są zainicjowane i zarezerwowane przez inną aplikację; żaden moduł nie został zarezerwowany dla bieżącej aplikacji.

Błędy (LC0_STATUS):

LC0_NO_MODULE - nie istnieje żaden z żądanych modułów.

4.2. Informacja o konfiguracji ogólnej (GET_TOTAL_CONFIGURATION_55).

Nazwa funkcji: LC055_GETTOTALCONFIGURATION; nr porządkowy: 17

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 17
LC0_STATUS	1	kod zakończenia funkcji = LC0_OK
parametry wyjściowe:		
LC0_TONF	1	konfiguracja modułów
LC0_TCTC	1	liczba dostępnych kanałów CTC
LC0_TPORT	1	liczba dostępnych portów

Przeznaczenie:

Funkcja zwraca informację o sumarycznej konfiguracji wszystkich modułów danego typu. Bajt konfiguracji modułów ma następujący format:

```

b8                b1
H G F E D C B A
x x x x x x x x

```

x: x = 1 - moduł zainstalowany, x = 0 - modułu nie ma

4.3. Informacja o konfiguracji modułu (GET_MODULE_CONFIGURATION_55).

Nazwa funkcji: LC055_GETMODULECONFIGURATION; nr porządkowy: 18

Rekord opisu zlecenia:

nazwa	rozmiar bajtach	w	znaczenie
LC0_CODE	1		kod funkcji = 18
LC0_STATUS	1		kod zakończenia funkcji
LC0_MMODULE			numer modułu
parametry wyjściowe:			
LC0_MTYPE	1		typ modułu
LC0_MBASE	2		adres bazowy modułu
LC0_MPORT	1		liczba dostępnych portów
LC0_MCTC	1		liczba kanałów CTC (3)
LC0_MCLOCK	12		tablica częstotliwości zegara CTC
LC0_MCTCGATE	1		bramki kanałów CTC
LC0_MINT	1		numer przerwania modułu (sprzętowy)
LC0_MINTSRC	2		źródła przerwania
LC0_MINTLEV	1		poziom przerywający
LC0_MINTMSK	1		maska przerwania

Przeznaczenie:

Funkcja zwraca informację o konfiguracji wyspecyfikowanego modułu. Informacje o konfiguracji sprzętowej modułu pochodzą z odpowiedniej sekcji pliku konfiguracyjnego AMBEX.INI (patrz dodatek E).

Znaczenie parametrów:

a) LC0_MTYPE: kod typu danego modułu:

kod	nazwa	typ modułu
1	LC_055_PIO	LC-055-PIO
2	LC_055_DCU	LC-055-DCU

b) LC0_MPORT:

Liczba dostępnych równoległych, 8-bitowych portów cyfrowych wejściowo-wyjściowych. Parametr dla modułu LC-055-PIO przybiera zawsze wartość 6, natomiast dla modułów LC-055-DCU 3 lub 6 - zależnie od zainstalowanych układów 8255.

c) LC0_MCLOCK:

Tablica (6 liczb całkowitych) określająca częstotliwość przebiegu zegarowego podawanego na poszczególne kanały układu CTC w (kHz). Dla modułu LC-055-PIO wszystkie elementy tablicy mają tę samą wartość (przebieg zegarowy jest wspólny dla wszystkich kanałów): 500, 2000 lub 4000 KHz. Dla modułu LC-055-DCU przebieg zegarowy jest ustawiany niezależnie dla każdego kanału: 500 lub 1000 kHz.

Wartość 0 oznacza przyłączenie odpowiedniego kanału CTC do zegara zewnętrznego.

Tablica przeznaczona jest perspektywnie dla 6 kanałów CTC.

d) LC0_MCTCGATE:

Maska aktywnych kanałów CTC. Znaczenie poszczególnych bitów:

```

b8          b1
- - - - - 3 2 1
- - - - - x x x

```

x: x = 0 -kanał aktywny, x = 1 -kanał nieaktywny (bramkowany)

e) LC0_MINT:

Parametr podaje nr sprzętowy przerwania związanego z danym modulem:

nr sprzętowy IRQ	nr programowy	uwagi
2	10	kaskada 8259
3	11	COM2
4	12	COM1
5	13	LPT2

f) LC0_MINTSRC:

Na poszczególnych bitach parametru LC0_MINTSRC zakodowane są źródła przerwania związane z poszczególnymi liniami przerywającymi modułu: moduł LC-055-DCU ma 4 linie przerywające, LC-055-PIO - 1 linię.

nr bitów	linia przerywająca	uwagi
1,2	-GATE IN ¹⁾ INT-1 ²⁾	-----
3,4	INT-2	tylko LC-055-DCU
5,6	INT-3	tylko LC-055-DCU
7,8	INT-4	tylko LC-055-DCU

1) dotyczy LC-055-PIO 2) dotyczy LC-055-DCU

Kody źródeł przerwania:

kod	nazwa	źródło przerwania	uwagi
0	LC0_NO_INT	brak przerwania	-----
1	LC0_INTSRC_EXT	sygnał zewnętrzny	-----
2	LC0_INTSRC_CTC	układ CTC	tylko LC-055-DCU

W module LC-055-DCU poszczególne kanały CTC są jednoznacznie przyporządkowane liniom przerywającym:

kanał CTC	nr linii przerywającej
0	INT 1, INT 2
1	INT 3
2	INT 4

f) LC0_MINTLEV:

Na poszczególnych bitach podany jest poziom sygnału przychodzącego do linii przerywającej, który powoduje przerwanie:

- moduł LC-055-PIO: przerwanie generowane zawsze niskim poziomem.

- moduł LC-055-DCU:

```

b8          b1
- - - - 4 3 2 1
- - - - x x x x

```

x: x=1 - poziom wysoki; x=0 - poziom niski

g) LC0_MINTMSK:

Na poszczególnych bitach podana jest maska przerwania:

moduł LC-055-PIO

```

b8          b1
- - - - - - 1
- - - - - - x

```

x: x=1 - zezwolenie na przerwanie; x=0 - zakaz przerwania

moduł LC-055-DCU

```

b8          b1
- - - - 4 3 2 1
- - - - x x x x

```

x: x=1 - zezwolenie na przerwanie od danej linii przerywającej; x=0 - zakaz przerwania przez daną linię przerywającą

Błędy:

LC0_NO_MODULE - nie ma takiego modułu

LC0_BAD_CONFIG - błąd w pliku konfiguracyjnym AMBEX.INI.

4.4. Obsługa portów.

4.4.1. Programowanie kierunku pracy (PORT_DIRECTION).

Nazwa funkcji: LC055_PORTDIRECTION; nr porządkowy: 19

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 19
LC0_STATUS	1	kod zakończenia funkcji
LC0_PMODULE	1	numer modułu
LC0_PDIR	2	zakodowane kierunki pracy portów

Przeznaczenie:

Funkcja służy do zaprogramowania kierunku pracy portów równoległych. Kierunek pracy portu kodowany jest na dwóch bitach:

nr bitu	nazwa portu	nr portu
1,2	A	1
3,4	B	2
5,6	C	3
7,8	D	4
9,10	E	5
11,12	F	6

Kierunek jest zakodowany na dwóch bitach:

nr bitu	wartość	nazwa	znaczenie
1	0	LC0_IN_DIRECTION	port wejściowy
	1	LC0_OUT_DIRECTION	port wyjściowy
2	0	-----	nie zmieniaj kierunku
	1	LC0_CHANGE_DIR	zmień kierunek

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany
 LC0_REJECTED - moduł zajęty przez inne zadanie
 LC0_DEV_BUSY - niemożliwy dostęp do portów

4.4.2. Zapis do portu (PORT_WRITE).

Nazwa funkcji: LC055_PORTWRITE; nr porządkowy: 20

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 20
LC0_STATUS	1	kod zakończenia funkcji
LC0_PWMODULE	1	numer modułu
LC0_PWPORT	1	numer portu
LC0_PWDATA	1	dana do wysłania

Przeznaczenie:

Bajt LC0_PWDATA jest wysyłany do danego portu.

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany
 LC0_REJECTED - moduł zajęty przez inne zadanie

LC0_DEV_BUSY - niemożliwy dostęp do portu
 LC0_NOT_OUTPORT - dany port nie jest zaprogramowany jako wyjściowy
 LC0_NONEX_DEV - port o danym numerze nie istnieje

4.4.3. Odczyt z portu (PORT_READ).

Nazwa funkcji: LC055_PORTREAD; nr porządkowy: 21

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 21
LC0_STATUS	1	kod zakończenia funkcji
LC0_PRMODULE	1	numer modułu
LC0_PRPORT	1	numer portu
LC0_PRLATCH	1	zatrzaśnięcie danych przed odczytem
parametry wyjściowe:		
LC0_PRDATA	1	odczytany bajt

Przeznaczenie:

Zadany port jest odczytywany, odczytany bajt umieszczany jest w polu LC0_PRDATA.

W przypadku modułu LC-055-PIO istotny jest parametr LC0_PRLATCH. Jeżeli ma on wartość 0 to odczyt przebiega standardowo. Natomiast jeżeli ma wartość 1 to na czas odczytu dane są zatrzaskiwane w rejestrze typu "latch". Zatrzaśnięcie to jest efektywne tylko wtedy gdy linia strobulująca -GATE_IN jest podłączona do "1" (patrz również instrukcja techniczna modułu i opis funkcji PORT_LATCH).

Ostrzeżenia:

LC0_NO_LATCH_WARN - dany moduł nie ma możliwości zatrzaskiwania danych - odczytano w sposób standardowy.

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany
 LC0_REJECTED - moduł zajęty przez inne zadanie
 LC0_DEV_BUSY - niemożliwy dostęp do portu
 LC0_NOT_INPORT - dany port nie jest zaprogramowany jako wyjściowy
 LC0_NONEX_DEV - port o danym numerze nie istnieje

4.4.4. Zatrzaśnięcie danych (PORT_LATCH).

Nazwa funkcji: LC055_PORTLATCH; nr porządkowy: 22

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 22
LC0_STATUS	1	kod zakończenia funkcji
LC0_PLMODULE	1	numer modułu
LC0_PLMODE	1	tryb pracy

Przeznaczenie:

Sterowanie zatrzaskiwaniem danych w rejestrze typu "latch". Zrealizowane tylko dla modułu LC-055-PIO.

Działanie funkcji określone jest precyzyjnie przez tryb pracy funkcji LC0_PLMODE i bieżący stan linii -GATE_IN modułu.

W przypadku pomyślnego zatrzaśnięcia danych w rejestrze wejściowym (także w funkcji PORT_READ) powrót do standardowego trybu odczytu danych nastąpi po wykonaniu funkcji z parametrem LC0_LATCH_OFF.

LC0_PLMODE	nazwa	linia -GATE_IN	znaczenie
1	LC0_LATCH_ON	1	zatrzaśnięcie danych w rejestrze
		0	zezwozenie na zatrzaśnięcie linią -GATE_IN

0	LC0_LATCH_OFF	1 0	zwolnienie rejestru do dalszej pracy zakaz zatraskiwania danych linią -GATE_IN
---	---------------	--------	---

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany
 LC0_REJECTED - moduł zajęty przez inne zadanie
 LC0_DEV_BUSY - niemożliwy dostęp do portu
 LC0_NOT_INPORT - dany port nie jest zaprogramowany jako wyjściowy
 LC0_NO_LATCH - dany moduł nie ma możliwości zatraskiwania danych

4.4.5. Zerowanie portów (PORT_RESET).

Nazwa funkcji: LC055_PORTRESET; nr porządkowy: 23

Rekord opisu zlecenia:

nazwa	rozmiar bajtach	w	znaczenie
LC0_CODE	1		kod funkcji = 23
LC0_STATUS	1		kod zakończenia funkcji
LC0_PRSMODULE	1		numer modułu

Przeznaczenie:

Wszystkie porty modułu ustawiane są jako wejściowe. W przypadku modułów LC-055-PIO wykonywane jest to programowo, w przypadku modułu LC-055-DCU wykorzystany jest odpowiedni sygnał sprzętowy.

W przypadku modułu LC-055-PIO funkcja dodatkowo zwalnia rejestry do dalszej pracy (odpowiednik funkcji PORT_LATCH z trybem pracy LC0_PLMODE =0).

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany
 LC0_REJECTED - moduł zajęty przez inne zadanie
 LC0_DEV_BUSY - niemożliwy dostęp do portu

4.5. Obsługa układu czasowego 8253/8254.

4.5.1. Programowanie i/lub ładowanie licznika (CTC_WRITE_55).

Nazwa funkcji: LC055_CTCWRITE; nr porządkowy: 24

Rekord opisu zlecenia:

nazwa	rozmiar bajtach	w	znaczenie
LC0_CODE	1		kod funkcji = 24
LC0_STATUS	1		kod zakończenia funkcji
LC0_CMODULE	1		numer modułu
LC0_CMODE	1		tryb pracy funkcji
LC0_CCTRL	1		wartość bajtu sterującego ¹⁾
LC0_CWDATA	2		wpisywana wartość licznika

¹⁾ wartość katalogowa bajtu sterującego układu 8253/8254 - patrz instrukcja techniczna modułu.

Przeznaczenie:

Przeznaczeniem funkcji jest zapisanie trybu pracy licznika układu 8253/8254 i/lub jego nowej wartości. Zależy to od ustawionych bitów trybu pracy LC0_CMODE.

nr bitu	nazwa	znaczenie
1	LC0_SET CTC MODE	zaprogramuj typ pracy kanału
2	LC0_SET COUNTER VALUE	załaduj nową wartość licznika
3...8	0	-----

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany
 LC0_REJECTED - moduł zajęty przez inne zadanie
 LC0_DEV_BUSY - niemożliwy dostęp do kanału CTC
 LC0_BAD_CTC_MODE - błędny parametr LC0_CCTRL (część dotycząca trybu pracy kanału CTC)
 LC0_CTC_NOT_PROGRAMMED - zlecono zapis wartości lecz nie zaprogramowano trybu pracy kanału
 LC0_NONEX_DEV - w parametrze LC0_CCTRL podano numer nieistniejącego kanału CTC.

4.5.2. Bramkowanie (CTC_GATE).

Nazwa funkcji: LC055_CTCGATE; nr porządkowy: 25

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 25
LC0_STATUS	1	kod zakończenia funkcji
LC0_CGMODULE	1	numer modułu
LC0_CGGATE	1	maska aktywnych kanałów

Przeznaczenie:

Zezwolenie na pracę lub zakazanie pracy poszczególnych kanałów CTC. Funkcja jest dostępna tylko dla modułu LC-055-DCU.

Znaczenie poszczególnych bitów parametru LC0_CGATE:

```

b8      b1
- - - - 3 2 1
- - - - - x x x
  
```

x: x = 0 -kanał aktywny, x = 1 -kanał nieaktywny (bramkowany)

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany
 LC0_REJECTED - moduł zajęty przez inne zadanie
 LC0_DEV_BUSY - niemożliwy dostęp do kanału CTC
 LC0_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułu LC-055-PIO).

4.5.3. Odczyt kanału CTC (CTC_READ_55).

Nazwa funkcji: LC055_CTCREAD; nr porządkowy: 26

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 26
LC0_STATUS	1	kod zakończenia funkcji
LC0_CRMODULE	1	numer modułu
LC0_CRCOUNT	1	numer kanału CTC (0, 1, 2)
parametry wyjściowe:		
LC0_CRDATA	2	odczytana wartość licznika

Przeznaczenie:

Funkcja odczytuje bieżącą wartość zadanego kanału CTC.

Błędy:

LC0_NO_MODULE - nie ma takiego modułu
 LC0_NOT_INIT - moduł nie jest zainicjowany

LC0_REJECTED - moduł zajęty przez inne zadanie
 LC0_DEV_BUSY - niemożliwy dostęp do kanału CTC
 LC0_NONEX_DEV- podano numer nieistniejącego kanału CTC

4.6. Obsługa przerwania.

Poniżej opisano zestaw funkcji driver'a umożliwiających wykorzystanie przerwania generowanych przez moduł cyfrowy. Na początek kilka ogólnych informacji opisujących ideologię obsługi przerwania przez driver.

Procedura obsługi przerwania składa się z dwóch części:

- procedura przyjęcia przerwania - dostarczana przez driver; jej zadaniem jest zapamiętanie rejestrów na stosie, wywołanie procedury wykonawczej klienta, powiadomienie kontrolera przerwania o obsłudze przerwania, ewentualne odblokowanie następnych przerwania, przywrócenie poprzedniego stosu i wartości rejestrów.

- procedura wykonawcza - dostarczana przez moduł klienta, w której wykonywana jest właściwa obsługa przerwania.

W driverze zastosowano 2 sposoby wywołania procedury wykonawczej:

- poprzez wstawienie do kolejki komunikatów komunikatu za pomocą funkcji *PostMessage()* lub *PostAppMessage()*, którego obsługa w module klienta stanowi właściwą procedurę wykonawczą,

- poprzez bezpośrednie wykonanie zarejestrowanej funkcji z modułu klienta.

W przypadku pierwszej metody nie ma żadnych dodatkowych ograniczeń co do operacji wykonywanych w procedurze obsługi komunikatu przerwania. Jest to zalecana metoda obsługi przerwania i należy ją preferować w stosunku do drugiej metody. W przypadku bezpośredniego wywołania funkcji klienta w procedurze obsługi przerwania muszą być spełnione następujące warunki:

1. procedura obsługi klienta powinna być procedurą daleką i być zadeklarowana jako jednoargumentowa z dalekim wskaźnikiem jako argumentem; przykładowy prototyp procedury jest następujący:

```
void far PASCAL MyISRProc(struct MyISRData far *ptr) ; język C
procedure MyISRProc(var ptr:MyISRData) ; język Pascal.
```

2. w procedurze nie można bezpośrednio odwoływać się do zmiennych globalnych (z domyślnego segmentu danych); procedura nie może modyfikować DS;

3. procedura powinna znajdować się w stałym i nieprzesuwalnym segmencie kodu (atrybut FIXED), wszystkie zmienne globalne modyfikowane przez procedurę muszą znajdować się w nieusuwalnych segmentach danych,

4. w czasie rejestracji obsługi (patrz opis funkcji INTERRUPT_SERVICE_55) należy ustawić parametr LC0_DATAPTR tak by był wskaźnikiem do bufora danych, do których będzie odwoływała się procedura wykonawcza; w momencie wywołania procedury wykonawczej z driver'a na stos przekazywany jest ten wskaźnik jako argument wywołania; dostęp do bufora danych w procedurze wykonawczej może być realizowany poprzez ten wskaźnik;

5. jedyne bezpieczne funkcje Windows API, które mogą być wykonane w procedurze wykonawczej to *PostMessage()* lub *PostAppMessage()*.

W procedurze wykonawczej można wywołać funkcję driver'a, przy czym powodzenie wykonania funkcji zależy od kontekstu: np. gdy obsługa przerwania wydarzyła się podczas wykonywania funkcji PORT_READ to wykonanie w obsłudze funkcji PORT_DIRECTION zwróci błąd LC0_DEV_BUSY, gdyż może to spowodować zaburzenie wykonania funkcji PORT_READ.

W celu odblokowania obsługi kolejnych przerwania modułu należy wywołać funkcję driver'a INTERRUPT_RESET, co może być zrealizowane w procedurze przyjęcia przerwania w driverze (patrz INTERRUPT_SERVICE_55) lub poprzez wywołanie funkcji driver'a INTERRUPT_RESET. Dla modułu LC-055-DCU w celu stwierdzenia przyczyny przerwania należy przed wykonaniem INTERRUPT_RESET wywołać funkcję READ_STATUS, która odczyta status modułu.

Jeżeli przerwanie przyjdzie przed zakończeniem obsługi poprzedniego to jest po prostu ignorowane.

Możliwe jest ustawienie dowolnego przerwania modułu dozwolonego za pomocą przełączników konfiguracyjnych za wyjątkiem przerwania nr 2 (kaskada 8259), które nie jest obsługiwane przez driver.

4.6.1. Maskowanie przerwania (INTERRUPT_MASK).

Nazwa funkcji: LC055_INTERRUPTMASK; nr porządkowy: 27

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 27
LC0_STATUS	1	kod zakończenia funkcji
LC0_IMMODULE	1	numer modułu

LC0_IMASK	1	maska przerw
-----------	---	--------------

Przeznaczenie:

Zezwolenie / zakaz przerwania przez poszczególne linie przerywajace.

Znaczenie poszczególnych bitów parametru LC0_IMASK:

moduł LC-055-PIO:

```
b8          b1
- - - - - 1
- - - - - x
```

x = 1 - zezwolenie na przerwanie; x = 0 - zakaz przerwania.

moduł LC-055-DCU:

```
b8          b1
- - - - 4 3 2 1
- - - - x x x x
```

x = 1 - zezwolenie na przerwanie od danej linii przerywajacej

x = 0 - zakaz przerywania przez daną linie przerywajacą

Błędy:

LC0_NO_MODULE - nie ma takiego modulu

LC0_NOT_INIT - moduł nie jest zainicjowany

LC0_REJECTED - moduł zajęty przez inne zadanie

LC0_DEV_BUSY - niemożliwy dostęp

4.6.2. Poziom sygnału przerywajacego (INTERRUPT_LEVEL).

Nazwa funkcji: LC055_INTERRUPTLEVEL; nr porzadkowy: 28

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 28
LC0_STATUS	1	kod zakonczenia funkcji
LC0_ILMODULE	1	numer modulu
LC0_ILEVEL	1	poziom sygnału przerywajacego na poszczególnych liniach

Przeznaczenie:

Zadanie poziomu sygnału przerywajacego dla poszczególnych linii przerywajacych. Tylko moduł LC-055-DCU.

Znaczenie poszczególnych bitów parametru LC0_ILEVEL:

```
b8          b1
- - - - 4 3 2 1
- - - - x x x x
```

x = 1 - poziom wysoki; x = 0 - poziom niski

Błędy:

LC0_NO_MODULE - nie ma takiego modulu

LC0_NOT_INIT - moduł nie jest zainicjowany

LC0_REJECTED - moduł zajęty przez inne zadanie

LC0_DEV_BUSY - niemożliwy dostęp

LC0_NOT_SUPPORTED - funkcja nie jest realizowana: moduł LC-055-PIO

4.6.3. Obsługa przerw (INTERRUPT_SERVICE_55).

Nazwa funkcji: LC055_INTERRUPTSERVICE; nr porzadkowy: 29

Rekord opisu zlecenia:

nazwa	rozmiar bajtach	w	znaczenie
LC0_CODE	1		kod funkcji = 29
LC0_STATUS	1		kod zakończenia funkcji
LC0_ISMODULE	1		numer modułu
LC0_IPROC	4		adres procedury wykonawczej offset-segment
LC0_DATAPTR	4		adres bufora parametrów procedury wykonawczej offset-segment
LC0_IMODE	2		tryb pracy funkcji
LC0_HWND	2		deskryptor okna dla komunikatu wstawianego przez <i>PostMessage</i>
LC0_HTASK	2		deskryptor zadania dla komunikatu wstawianego przez <i>PostAppMessage</i>
LC0_UMSG	2		kod wstawianego komunikatu
LC0_WPARAM	2		parametr WORD wstawianego komunikatu
LC0_LPARAM	4		parametr LONG wstawianego komunikatu

Przeznaczenie:

Funkcja służy do instalowania i wyinstalowania procedur obsługi przerwania przychodzącego z danego modułu.

Działanie funkcji zależy od poszczególnych bitów parametru LC0_IMODE:

- bit b1: 0 - wyinstalowanie obsługi przerwania i przywrócenie poprzedniej,
1 - zainstalowanie obsługi przerwania.
- bit b2: 0 - obsługa będzie polegała na wstawieniu komunikatu do kolejki komunikatów, zależnie od bitu b3,
1 - obsługa będzie polegała na wywołaniu funkcji klienta wskazywanej przez LC0_IPROC z parametrem wywołania LC0_DATAPTR.
- bit b3: 0 - komunikat będzie wstawiony porzez funkcję *PostMessage* z parametrami wywołania: LC0_HWND, LC0_UMSG, LC0_WPARAM, LC0_LPARAM,
1 - komunikat będzie wstawiony porzez funkcję *PostAppMessage* z parametrami wywołania: LC0_HTASK, LC0_UMSG, LC0_WPARAM, LC0_LPARAM,
- bit b4: 0 - procedura obsługi driver'a nie wykonuje funkcji INTERRUPT_RESET,
1 - procedura obsługi driver'a wykonuje funkcję INTERRUPT_RESET na zakończenie obsługi, odblokowując generację kolejnych przerw.

Procedura obsługi przerwania wyinstalowana jest również przez funkcję LEAVE_DRIVER_55.

Błędy:

LC0_NO_MODULE - nie ma takiego modułu

LC0_NOT_INIT - moduł nie jest zainicjowany

LC0_REJECTED - moduł zajęty przez inne zadanie

LC0_DEV_BUSY - niemożliwy dostęp

LC0_NO_IRQ - z danym modułem nie jest związane żadne przerwania

LC0_INTR_INST - procedura obsługi jest już zainstalowana dla danego modułu

LC0_IRQ_BUSY - dana linia przerwania jest aktualnie wykorzystywana przez inny moduł,

LC0_BAD_PROC - błędny adres procedury wykonawczej

4.6.4. Kasowanie zgłoszenia przerwania (INTERRUPT_RESET).

Nazwa funkcji: LC055_INTERRUPTRESET; nr porządkowy 30:

Rekord opisu zlecenia:

nazwa	rozmiar bajtach	w	znaczenie
LC0_CODE	1		kod funkcji = 30
LC0_STATUS	1		kod zakończenia funkcji
LC0_IRMODULE	1		numer modułu

Przeznaczenie:

Zgaszenie przerzutnika zgłoszenia przerwania co umożliwi generacją przerw sprzętowych modułu.

Błędy

LC0_NO_MODULE - nie ma takiego modułu

LC0_NOT_INIT - moduł nie jest zainicjowany

LC0_REJECTED - moduł zajęty przez inne zadanie

LC0_DEV_BUSY - niemożliwy dostęp

4.7. Odczyt statusu (READ_STATUS).

Nazwa funkcji: LC055_READSTATUS; nr porządkowy: 31

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 31
LC0_STATUS	1	kod zakończenia funkcji
LC0_SMODULE	1	numer modułu
parametry wyjściowe:		
LC0_SDATA	1	odczytany status
LC0_SRESRVD	1	pole zarezerwowane

Przeznaczenie:

Odczytanie statusu - stanu poszczególnych linii.

Znaczenie poszczególnych bitów parametru LC0_SDATA:

nr bitu	nazwa	znaczenie
1	LC0_INT1_INTERRUPTED	stan linii przerywającej INT 1
2	LC0_INT2_INTERRUPTED	stan linii przerywającej INT 2
3	LC0_INT3_INTERRUPTED	stan linii przerywającej INT 3
4	LC0_INT4_INTERRUPTED	stan linii przerywającej INT 4
5	LC0_COMMON_INTERRUPT	suma linii przerywających 1..4
6	LC0_CTC_OUT0	stan wyjścia kanału 0 CTC
7	LC0_CTC_OUT1	stan wyjścia kanału 1 CTC
8	LC0_CTC_OUT2	stan wyjścia kanału 2 CTC

Błędy:

LC0_NO_MODULE - nie ma takiego modułu

LC0_NOT_INIT - moduł nie jest zainicjowany

LC0_REJECTED - moduł zajęty przez inne zadanie

LC0_DEV_BUSY - niemożliwy dostęp

LC0_NOT_SUPPORTED - funkcja nie jest realizowana dla modułu LC-055-PIO.

4.17. Zakończenie pracy z driver'em (LEAVE_DRIVER_55).

Nazwa funkcji: LC055_LEAVEDRIVER; nr porządkowy: 32

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 32
LC0_STATUS	1	kod zakończenia funkcji = LC0_OK

Przeznaczenie:

Funkcja, którą należy wywołać w momencie zakończeniu pracy z driver'em. Powoduje "zapomnienie" przez driver wszystkiego co zostało mu przekazane w trakcie pracy programu. Ponadto wywołanie tej funkcji zwalnia

zainicjowane przez bieżącą aplikację moduły tak, że są one dostępne dla innych aplikacji. Funkcje LEAVE_DRIVER i MODULE_INIT sterują wielodostępem w korzystaniu z modułów za pośrednictwem driver'a. Należy bezwzględnie wykonać funkcję przed wyjściem z programu, gdyż w przeciwnym wypadku zainicjowane moduły przez bieżącą aplikację będą niedostępne dla innych programów aż do momentu zwolnienia driver'a.

Funkcja wyinstalowywuje procedurę obsługi przerwania z modułu zainstalowaną przez funkcję INTERRUPT_SERVICE_55 (patrz).

5. Zestawienie kodów zakończenia funkcji.

nazwa	kod	znaczenie
LC0_OK	0	poprawne zakończenie funkcji

Błędy (LC0_STATUS):

nazwa	kod	znaczenie
LC0_UNKN_FUNC	-1	nieznany kod funkcji
LC0_NO_MODULE	-2	nie istnieje żaden z żądanych modułów; nie ma takiego modułu
LC0_NOT_OUTPORT	-3	port nie jest zaprogramowany jako wyjściowy
LC0_NONEX_DEV	-4	nie istnieje urządzenie o tym numerze
LC0_NOT_INPORT	-5	port nie jest zaprogramowany jako wejściowy
LC0_BAD_COUNTER	-6	zły numer kanału CTC
LC0_NO_LATCH	-7	dany moduł nie ma możliwości zatraskiwania danych
LC0_DEV_BUSY	-11	urządzenie zajęte
LC0_BAD_PROC	-19	błędny adres procedury obsługi przerwania
LC0_NOT_SUPPORTED	-24	dla danego modułu funkcja nie jest realizowana
LC0_BAD_CTC_MODE	-25	błędny tryb pracy CTC
LC0_NO_PARAMS	-26	nie podano parametrów przetwarzania a/c, c/a
LC0_NO_IRQ	-29	z danym modułem nie jest związane żadne przerwanie lub procedura obsługi nie została zainstalowana
LC0_NOT_FULLY_SUP	-30	żądany tryb wykonania funkcji nie jest dla danego typu modułu realizowany lub jest w opracowaniu
LC0_INTR_INST	-33	procedura obsługi przerwania jest już zainstalowana
LC0_CTC_NOT_PROGRAMMED	-37	zlecono zapis wartości licznika lecz nie zaprogramowano trybu pracy kanału
LC0_REJECTED	-38	za dużo równoczesnych odwołań do driver'a; moduł zainicjowany przez inne zadanie
LC0_BAD_CONFIG	-39	błędny plik konfiguracyjny
LC0_NOT_INIT	-40	moduł nie zainicjowany
LC0_IRQ_BUSY	-41	inny moduł obsługuje dane przerwanie sprzętowe

Ostrzeżenia (LC0_STATUS):

nazwa	kod	znaczenie
LC0_NON_EX_MOD	1	zażądano inicjalizacji nie istniejących modułów ale co najmniej 1 moduł został zainicjalizowany
LC0_NO_LATCH_WARN	2	dany moduł nie ma możliwości zatraskiwania danych
LC0_IS_INIT	5	moduł zainicjowany

6. Projektowanie programów użytkowych.

W poniższym rozdziale zostanie omówiony sposób komunikacji programów użytkowych z driver'em. Na wstępie opisane zostaną ogólne zasady komunikacji a w dalszych podrozdziałach - komunikacja z driver'em z poziomu programów napisanych w C i Pascalu.

Wykonanie dowolnej z funkcji driver'a wymaga następujących czynności:

- wypełnienie odpowiedniego rekordu opisu zlecenia; odpowiednie struktury danych dostarczane są przez producenta w postaci zbiorów źródłowych

- wywołanie odpowiedniej funkcji driver'a: np (C, PASCAL): *NazwaFunkcji(daleki_wskaznik_do_rekordu_zlecenia)*

Dobrze napisany program powinien składać się z następujących części:

- część wstępna:

- rozpoznanie konfiguracji modułu (GET_TOTAL_CONFIGURATION_55 - ile i jakich modułów jest zainstalowanych, GET_MODULE_CONFIGURATION_5 - czy jest podłączony do któregoś z przerwań itp.; ten etap jest szczególnie ważny, gdy projektowany jest program uniwersalny, mający operować na kilku rodzajach modułów

- inicjalizacja modułu; przydziela bieżącej aplikacji (zadaniu) wyspecyfikowane moduły.

- część wykonawcza: tu powinny się znaleźć funkcje wykonujące właściwe operacje modułu; należy zwrócić uwagę na to, że każda funkcja może dostać błędne parametry i zasygnalizować to w odpowiedzi (LC0_STATUS); należy koniecznie sprawdzać tę odpowiedź, szczególnie w dwóch przypadkach: gdy program jest na etapie uruchamiania i gdy parametry funkcji są dostarczane interakcyjnie przez użytkownika,

- część końcowa:

- przed zakończeniem programu należy koniecznie wykonać funkcję LEAVE_DRIVER_55; jest to szczególnie ważne wtedy, gdy na komputerze wykonywanych jest kilka programów korzystających z tego samego driver'a.

6.1. Programowanie w języku C.

Z poziomu języka C komunikacja z driver'em odbywa się poprzez jawne wywołanie funkcji publicznych driver'a. Producent dostarcza dwa pliki źródłowe: plik AMBEX-55.H zawierający definicje wszystkich struktur danych, stałych deklaracje funkcji potrzebnych do współpracy z driver'em oraz plik WSMP055.C zawierający przykłady wykorzystania poszczególnych funkcji driver'a LC055ADLL.

W przypadku implementacji C++ Borland (Turbo C) należy deklarację wszystkich funkcji driver'a w pliku nagłówkowym otoczyć wskazaniem klasy "C" (extern "C").

W Dodatku A znajduje się wydruk pliku AMBEX-55.H, natomiast w Dodatku C - wydruki plików: WSMP055.C - program przykładowy C (API) z wczytaniem biblioteki poprzez plik definicji modułu; WSMP055.DEF - plik definicji modułu dla WSMP055.C.

6.2. Programowanie w języku Pascal.

Z poziomu języka Pascal komunikacja z driver'em odbywa się poprzez jawne wywołanie funkcji publicznych driver'a.

Producent dostarcza dwa pliki źródłowe: plik AMBEX-55.PAS zawierający definicje wszystkich struktur danych i stałych potrzebnych do współpracy z driver'em, który znajduje się w dodatku C; plik WSMP055.PAS - przykład wykorzystania poszczególnych funkcji driver'a; plik LC055U.PAS - plik źródłowy dla modułu LC055U.TPW, służącego do powiązania biblioteki LC055A.DLL z aplikacjami PASCAL-a.

DODATEK A

do dokumentacji biblioteki DLL

modułów cyfrowych

LC-055-PIO i LC-055-DCU

AMBEX-55.H

struktury danych i stałe dla języka C

```

/*****/
/*                                                                    */
/* Zestaw deklaracji struktur danych i definicji stałych do komunikacji */
/* z bibliotek DLL LC055A.DLL z poziomu języka C.                       */
/*                                                                    */
/*****/
/*****/
UWAGA ! Należy włączyć plik "windows.h" !.
/*****/

#define LC0_MAX_CTC          6      /* maksymalna liczba kanałów CTC      */
#define LC0_MAX_PORTS       6      /* maksymalna liczba portów          */
#define LC0_MAX_INT_LINES   4      /* maksymalna liczba linii           */
                                   /* przerywających                    */

/***** REKORDY OPISÓW ZLECEŃ FUNKCJI DRIVER'A *****/

/* Rekord opisu zlecenia funkcji MODULE_INIT_55 ===== */
struct lc0_init_55
{
    unsigned char    LC0_CODE;      /* numer funkcji (16)                */
    char             LC0_STATUS;    /* kod odpowiedzi driver'a           */
    unsigned char    LC0_IMODULE;   /* mapa modułów                      */
};

/* Rekord opisu zlecenia funkcji GET_TOTAL_CONFIGURATION_55 ===== */
struct lc0_total_55
{
    unsigned char    LC0_CODE;      /* numer funkcji (17)                */
    char             LC0_STATUS;    /* kod odpowiedzi driver'a           */
    unsigned char    LC0_TONF;     /* inf. o zainstalowanych modułach  */
    unsigned char    LC0_TCTC;     /* liczba dostępnych kanałów CTC     */
    unsigned char    LC0_TPORT;    /* liczba dostępnych portów         */
};

/* Rekord opisu zlecenia funkcji GET_MODULE_CONFIGURATION_55 ===== */
struct lc0_module_55
{
    unsigned char    LC0_CODE;      /* numer funkcji (18)                */
    char             LC0_STATUS;    /* kod odpowiedzi driver'a           */
    unsigned char    LC0_MMODULE;   /* numer modułu                      */
    unsigned char    LC0_MTYPE;     /* typ modułu                        */
    unsigned int     LC0_MBASE;     /* adres bazowy rejestrów modułu     */
    unsigned char    LC0_MPORT;    /* liczba dostępnych portów         */
    unsigned char    LC0_MCTC;     /* liczba dostępnych kanałów CTC     */
    unsigned int     LC0_MCLOCK[LC0_MAX_CTC]; /* tabl. częstotliwości          */
                                   /* zegara CTC w kHz                  */
    unsigned char    LC0_CTCGATE; /* bramki kanałów CTC                */
    unsigned char    LC0_MINT;     /* numer przerwania (programowy)     */
    unsigned int     LC0_MINTSRC;   /* określenie źródeł przerw         */
    unsigned char    LC0_MINTLEV;   /* poziom przerywający              */
    unsigned char    LC0_MINTMSK;   /* maski przerw                      */
};

/* Rekord opisu zlecenia funkcji PORT_DIRECTION ===== */
struct lc0_port_dir
{
    unsigned char    LC0_CODE;      /* numer funkcji (19)                */

```

```

char          LC0_STATUS;          /* kod odpowiedzi driver'a      */
unsigned char LC0_PMODULE;         /* numer modulu                  */
unsigned long LC0_PDIR;            /* kierunki pracy portow        */
};

/* Rekord opisu zlecenia funkcji PORT_WRITE ===== */
struct lc0_port_write
{
unsigned char LC0_CODE;           /* numer funkcji (20)           */
char          LC0_STATUS;         /* kod odpowiedzi driver'a      */
unsigned char LC0_PWMODULE;       /* numer modulu                  */
unsigned char LC0_PWPORT;         /* numer portu                   */
unsigned char LC0_PWDATA;         /* dana do wyslania             */
};

/* Rekord opisu zlecenia funkcji PORT_READ ===== */
struct lc0_port_read
{
unsigned char LC0_CODE;           /* numer funkcji (21)           */
char          LC0_STATUS;         /* kod odpowiedzi driver'a      */
unsigned char LC0_PRMODULE;       /* numer modulu                  */
unsigned char LC0_PRPORT;         /* numer portu                   */
unsigned char LC0_PRLATCH;        /* LC0_LATCH_ON/LC0_LATCH_OFF  */
unsigned char LC0_PRDATA;         /* dana odczytana               */
};

/* Rekord opisu zlecenia funkcji PORT_LATCH ===== */
struct lc0_port_latch
{
unsigned char LC0_CODE;           /* numer funkcji (22)           */
char          LC0_STATUS;         /* kod odpowiedzi driver'a      */
unsigned char LC0_PLMODULE;       /* numer modulu                  */
unsigned char LC0_PLMODE;         /* tryb pracy                    */
};

/* Rekord opisu zlecenia funkcji PORT_RESET ===== */
struct lc0_port_reset
{
unsigned char LC0_CODE;           /* numer funkcji (23)           */
char          LC0_STATUS;         /* kod odpowiedzi driver'a      */
unsigned char LC0_PRSMODULE;      /* numer modulu                  */
};

/* Rekord opisu zlecenia funkcji CTC_WRITE_55 ===== */
struct lc0_ctc_write_55
{
unsigned char LC0_CODE;           /* numer funkcji (24)           */
char          LC0_STATUS;         /* kod odpowiedzi driver'a      */
unsigned char LC0_CMODULE;        /* numer modulu                  */
unsigned char LC0_CMODE;          /* tryb pracy funkcji           */
unsigned char LC0_CCTRL;          /* bajt sterujacy                */
unsigned int  LC0_CWDATA;         /* wartosc licznika              */
};

/* Rekord opisu zlecenia funkcji CTC_GATE ===== */
struct lc0_ctc_gate
{
unsigned char LC0_CODE;           /* numer funkcji (25)           */
char          LC0_STATUS;         /* kod odpowiedzi driver'a      */
};

```

```

    unsigned char    LC0_CGMODULE;    /* numer modulu          */
    unsigned char    LC0_CGGATE;      /* maska aktywnych kanalow */
};

/* Rekord opisu zlecenia funkcji CTC_READ_55 ===== */
struct lc0_ctc_read_55
{
    unsigned char    LC0_CODE;        /* numer funkcji (26)     */
    char             LC0_STATUS;      /* kod odpowiedzi driver'a */
    unsigned char    LC0_CRMODULE;    /* numer modulu           */
    unsigned char    LC0_CRCOUNT;    /* numer kanalu           */
    unsigned int     LC0_CRDATA;      /* odczytana dana        */
};

/* Rekord opisu zlecenia funkcji INTERRUPT_MASK ===== */
struct lc0_int_mask
{
    unsigned char    LC0_CODE;        /* numer funkcji (27)     */
    char             LC0_STATUS;      /* kod odpowiedzi driver'a */
    unsigned char    LC0_IMMODULE;    /* numer modulu           */
    unsigned char    LC0_IMASK;      /* maska przerw          */
};

/* Rekord opisu zlecenia funkcji INTERRUPT_LEVEL ===== */
struct lc0_int_level
{
    unsigned char    LC0_CODE;        /* numer funkcji (28)     */
    char             LC0_STATUS;      /* kod odpowiedzi driver'a */
    unsigned char    LC0_ILMODULE;    /* numer modulu           */
    unsigned char    LC0_ILEVEL;      /* poziom przerywajacy na */
                                        /* poszczegolnych liniach */
};

/* Rekord opisu zlecenia funkcji INTERRUPT_SERVICE_55 ===== */
struct lc0_int_service
{
    unsigned char    LC0_CODE;        /* numer funkcji (29)     */
    char             LC0_STATUS;      /* kod odpowiedzi driver'a */
    unsigned char    LC0_ISMODULE;    /* numer modulu           */
    void FAR PASCAL (*LC0_IPROC)(void FAR *wsk); /* adres procedury obsługi */
    void FAR *LC0_DATAPTR;           /* wskaznik dla bufora danych dla funkcji */
    unsigned int LC0_IMODE;           /* tryb pracy funkcji      */
    HWND LC0_HWND;                   /* parametry dla funkcji PostMessage */
    HTASK LC0_HTASK;                 /* i PostAppMessage       */
    UINT LC0_UMSG;
    WPARAM LC0_WPARAM;
    LPARAM LC0_LPARAM;
};

/* opis trybu pracy LC0_IMODE */
#define LC0_IM_INST      0x0001      /* instalacja przerwania */
#define LC0_IM_UNINST   0            /* deinstalacja przerwania */
#define LC0_IM_CALL     0x0002      /* wywołanie funkcji klienta */
#define LC0_IM_POST     0            /* wysłanie komunikatu */
#define LC0_IM_PAMES    0x0004      /* ma być wywołana funkcja PostAppMessage(); */
#define LC0_IM_PMES     0            /* ma być wywołana funkcja PostMessage() */
#define LC0_IM_IREN     0x0008      /* wykonanie INTERRUPT_RESET przez ISR */
#define LC0_IM_IRDIS    0            /* INTERRUPT reset musi wywołać klient */

```

```

/* Rekord opisu zlecenia funkcji INTERRUPT_RESET ===== */
struct lc0_int_reset
{
    unsigned char    LC0_CODE;           /* numer funkcji (30)           */
    char            LC0_STATUS;         /* kod odpowiedzi driver'a      */
    unsigned char    LC0_IRMODULE;     /* numer modulu                 */
};

/* Rekord opisu zlecenia funkcji READ_STATUS ===== */
struct lc0_read_status
{
    unsigned char    LC0_CODE;           /* numer funkcji (31)           */
    char            LC0_STATUS;         /* kod odpowiedzi driver'a      */
    unsigned char    LC0_SMODULE;      /* numer modulu                 */
    unsigned char    LC0_SDATA;        /* odczytany status             */
    unsigned char    LC0_SRESRVD;      /* pole zarezerwowane          */
};

/* Rekord opisu zlecenia funkcji LEAVE_DRIVER_55 ===== */
struct lc0_leave_55
{
    unsigned char    LC0_CODE;           /* numer funkcji (32)           */
    char            LC0_STATUS;         /* kod odpowiedzi driver'a      */
};

/* ***** kody funkcji driver'a ***** */
#define MODULE_INIT_55          16      /* inicjalizacja modułow        */
#define GET_TOTAL_CONFIGURATION_55 17    /* odczyt konfig. modułow       */
#define GET_MODULE_CONFIGURATION_55 18   /* odczyt konfig. modulu        */
#define PORT_DIRECTION          19      /* ustawienie kierunku          */
/* pracy portow                */
#define PORT_WRITE              20      /* zapis do portu               */
#define PORT_READ               21      /* odczyt portu                 */
#define PORT_LATCH              22      /* zatrzasniecie danych         */
#define PORT_RESET              23      /* zerowanie ukladow 8255       */
/* (ustawienie wszystkich      */
/* portow w kierunku IN)       */
#define CTC_WRITE_55            24      /* zaprogramowanie i/lub        */
/* zapisanie wartosci do        */
/* kanalu CTC                   */
#define CTC_GATE                25      /* bramkowanie kanalu CTC       */
#define CTC_READ_55             26      /* odczyt kanalu CTC           */
#define INTERRUPT_MASK          27      /* maskowanie przerwan          */
#define INTERRUPT_LEVEL         28      /* ustawianie poziomu           */
/* przerywajacego              */
#define INTERRUPT_SERVICE_55    29      /* deklarowanie procedury       */
/* obsługi przerwania          */
#define INTERRUPT_RESET         30      /* gaszenie zgloszenia          */
/* przerwania                  */
#define READ_STATUS             31      /* odczyt statusu               */
#define LEAVE_DRIVER_55         32      /* zakonczenie pracy z          */
/* driver'em                    */

/* ***** kody odpowiedzi funkcji driver'a ***** */
#define LC0_OK                   0       /* OK                             */

/* ===== ostrzezenia ===== */

```

```

#define LC0_NON_EX_MOD      1      /* nie istniejacy(e) modul(y) */
#define LC0_NO_LATCH_WARN  2      /* modul nie ma mozliwosci */
/* zatrzaskiwania danych */
#define LC0_IS_INIT        5      /* zażądano inicjacji już zainicjowanego */

/* ===== bledy ===== */

#define LC0_UNKN_FUNC      -1      /* nieznany kod funkcji */
#define LC0_NO_MODULE      -2      /* brak modulu(ow) */
#define LC0_NOT_OUTPORT    -3      /* port nie jest wyjsciowy */
#define LC0_NONEX_DEV      -4      /* nie istnieje urzadzenie o tym */
/* numerze */
#define LC0_NOT_INPORT     -5      /* port nie jest wejsciowy */
#define LC0_BAD_COUNTER    -6      /* zly numer kanalu CTC */
#define LC0_NO_LATCH       -7      /* dany modul nie moze zatrzaskiwac */
#define LC0_DEV_BUSY       -11     /* urzadzenie jest zajete */
#define LC0_BAD_PROC       -19     /* bledny adres procedury obslugi */
/* przerwan */
#define LC0_NOT_SUPPORTED  -24     /* dla danego modulu funkcja nie */
/* jest realizowana */
#define LC0_BAD_CTC_MODE   -25     /* bledny tryb pracy CTC */
#define LC0_NO_IRQ         -29     /* z danum modulem nie jest */
/* zwiazane zadne przerwanie */
#define LC0_NOT_FULLY_SUP  -30     /* zadany tryb funkcji nie */
/* jest realizowany dla danego typu */
/* modulu lub funkcja w opracowaniu */
#define LC0_INTR_INST      -33     /* proba powtornej instalacji */
/* przerwania */
#define LC0_CTC_NOT_PROGRAMMED -37  /* zapis wartosci do */
/* niezaprogramowanego licznika */
#define LC0_REJECTED       -38     /* za duzo jednoczesnych odwo lan do */

#define LC0_BAD_CONFIG     -39     /* bład w pliku konfiguracyjnym *.ini */
#define LC0_NOT_INIT       -40     /* nie zainicjowany modul */
#define LC0_IRQ_BUSY       -41     /* inny modul obsluguje dane przerwanie sprzetowe */
/* driver'a */

/***** kody numerow modutow *****/
#define LC0_MODA  1      /* modul A */
#define LC0_MODB  2      /* modul B */
#define LC0_MODC  3      /* modul C */
#define LC0_MODD  4      /* modul D */
/* moduly dodatkowe */
#define LC0_MODE  5      /* modul E */
#define LC0_MODF  6      /* modul F */
#define LC0_MODG  7      /* modul G */
#define LC0_MODH  8      /* modul H */

/***** maski modutow w mapie *****/
#define LC0_MODAMAP1  /* modul A */
#define LC0_MODBMAP2  /* modul B */
#define LC0_MODCMAP4  /* modul C */
#define LC0_MODDMAP8  /* modul D */
/* moduly dodatkowe */
#define LC0_MODEMAP16  /* modul E */
#define LC0_MODFMAP32  /* modul F */
#define LC0_MODGMAP64  /* modul G */
#define LC0_MODHMAP128 /* modul H */

```

```
/****** kody typow modułow ******/
#define LC_055_PIO 1 /* LC-055-PIO */
#define LC_055_DCU 2 /* LC-055-DCU */

/****** kody zrodła przerwania ******/
#define LC0_NO_INT 0 /* brak przerwania */
#define LC0_INTSRC_EXT 1 /* sygnał zewnętrzny */
#define LC0_INTSRC_CTC 2 /* CTC */

/****** kody bramkowania CTC ******/
#define LC0_CTC_GATE_EXT 1 /* sygnał wewnętrzny */
#define LC0_CTC_GATE_EN 0 /* bramka "1" - zezwolenie na stałe */

/****** kody maski przerwań ******/
#define LC0_INT_ENABLE 1 /* zezwolenie na przerwanie */
#define LC0_INT_DISABLE 0 /* zakaz przerwania */

/****** kody kierunkow pracy portow ******/
/* bit kierunku : */
#define LC0_IN_DIRECTION 0 /* port wejsciowy */
#define LC0_OUT_DIRECTION 1 /* port wyjsciowy */
/* bit trybu : */
#define LC0_CHANGE_DIR 2 /* zmien kierunek pracy */

/****** PORT_LATCH/LC0_PLMODE, PORT_READ/LC0_PRLATCH ******/
#define LC0_LATCH_ON 1 /* zatrzasnij */
#define LC0_LATCH_OFF 0 /* zwolnij */

/****** kody trybu pracy funkcji CTC_MODE ******/
#define LC0_SET_CTC_MODE 1 /* zaprogramuj tryb pracy kanału */
#define LC0_SET_COUNTER_VALUE 2 /* załaduj nowa wartosc licznika */

/****** bity statusu (READ_STATUS) ******/
#define LC0_INT1_INTERRUPTED 1 /* linia INT 1 zglosila przerwanie */
#define LC0_INT2_INTERRUPTED 2 /* linia INT 2 zglosila przerwanie */
#define LC0_INT3_INTERRUPTED 4 /* linia INT 3 zglosila przerwanie */
#define LC0_INT4_INTERRUPTED 8 /* linia INT 4 zglosila przerwanie */
#define LC0_COMMON_INTERRUPT 16 /* jedna z linii zglosila przerwanie */
#define LC0_CTC_OUT0 32 /* stan linii OUT kanału 0 */
#define LC1_CTC_OUT1 64 /* stan linii OUT kanału 1 */
#define LC0_CTC_OUT2 128 /* stan linii OUT kanału 2 */

/****** wartosci fragmentow bajtu sterujacego 8253/8254 ******/
#define CTC_NB 0 /* kod naturalny binarny */
#define CTC_BCD 1 /* kod BCD */

#define CTC_MODE0 0 /* tryb 0 */
#define CTC_MODE1 2 /* tryb 1 */
#define CTC_MODE2 4 /* tryb 2 */
#define CTC_MODE3 6 /* tryb 3 */
#define CTC_MODE4 8 /* tryb 4 */
#define CTC_MODE5 10 /* tryb 5 */

#define CTC_LSB 0x10 /* tylko mlodszy bajt */
#define CTC_MSB 0x20 /* tylko starszy bajt */
#define CTC_BOTH 0x30 /* mlodszy - starszy */

#define CTC_COUNT0 0x00 /* licznik 0 */
```

```
#define CTC_COUNT1 0x40 /* licznik 1 */
#define CTC_COUNT2 0x80 /* licznik 2 */
```

```
/******
```

Funkcje publiczne biblioteki DLL

```
*****/
```

```
void FAR PASCAL _export LC055_ModuleInit(struct lc0_init_55 FAR *param);
void FAR PASCAL _export LC055_GetTotalConfiguration(struct lc0_total_55 FAR *param);
void FAR PASCAL _export LC055_GetModuleConfiguration(struct lc0_module_55 FAR *param);
void FAR PASCAL _export LC055_PortDirection(struct lc0_port_dir FAR *param);
void FAR PASCAL _export LC055_PortWrite(struct lc0_port_write FAR *param);
void FAR PASCAL _export LC055_PortRead(struct lc0_port_read FAR *param);
void FAR PASCAL _export LC055_PortLatch(struct lc0_port_latch FAR *param);
void FAR PASCAL _export LC055_PortReset(struct lc0_port_reset FAR *param);
void FAR PASCAL _export LC055_CTCWrite(struct lc0_ctc_write_55 FAR *param);
void FAR PASCAL _export LC055_CTCGate(struct lc0_ctc_gate FAR *param);
void FAR PASCAL _export LC055_CTCRead(struct lc0_ctc_read_55 FAR *param);
void FAR PASCAL _export LC055_InterruptMask(struct lc0_int_mask FAR *param);
void FAR PASCAL _export LC055_InterruptLevel(struct lc0_int_level FAR *param);
void FAR PASCAL _export LC055_InterruptService(struct lc0_int_service FAR *param);
void FAR PASCAL _export LC055_InterruptReset(struct lc0_int_reset FAR *param);
void FAR PASCAL _export LC055_ReadStatus(struct lc0_read_status FAR *param);
void FAR PASCAL _export LC055_LeaveDriver(struct lc0_leave_55 FAR *param);
```

DODATEK B

do dokumentacji biblioteki DLL

modułów cyfrowych

LC-055-PIO i LC-055-DCU

WSMP055.C

program przykładowy w C

```

/* WSMP055.C ****
Program przykładowy pokazujący sposób wykorzystania drivera modułów cyfrowych
serii LC-055-PIO i LC-055-DCU firmy AMBEX.
Sposób dołączenia biblioteki DLL : statycznie poprzez plik .DEF */
****
Program korzysta ze standardowych funkcji API Windows 3.1 .
W katalogu zawierającym standardowe nagłówki C muszą się znajdować pliki:
AMBEX-55.H - plik nagłówkowy drivera,
RESOURCE.H - plik nagłówkowy zasobów programu,
WSMP055.RC - plik zasobów ,
WSMP055.DEF - plik definicyjny modułu
Pliki :
- AMBEX.INI - plik konfiguracyjny modułów serii LC,
- LC055A.DLL - driver (biblioteka DLL) do kart LC-055-PIO i LC-055-DCU,
powinny znajdować się w jednym z katalogów ze standardowej ścieżki przeszukiwania Windows (według kolejności przeszukiwania) :
1. katalog systemowy WINDOWS,
2. katalog WINDOWS\SYSTEM,
3. bieżący katalog aplikacji,
4. katalog znajdujący się na ścieżce przeszukiwania PATH .
Opcje kompilatora (Borland C++)
- data alignment : byte,
- memory model : far,
- entry/exit code : windows smart callbacks explicit funstions exportable

UWAGA :
Procedura obsługi przerwania sprzętowego karty nie powinna bezpośrednio
odwoływać się do zmiennych globalnych i powinna znajdować się w segmencie
kodu oznaczonym jako FIXED . Wszystkie zmienne globalne modyfikowane przez tą
procedurę powinny znajdować się również w segmencie danych oznaczonym jako
FIXED . Jedynymi zalecanymi przez MICROSOFT funkcjami WINDOWS API, które
mogą być wywołane bezpiecznie przez procedurę obsługi przerwania są :
PostMessage i PostAppMessage .
Powyższe ograniczenia nie obowiązują, gdy do powiadomieniu o wystąpieniu
przerwania używa się funkcji obsługi komunikatu wysłanego przez driver .

*/
****
#include <windows.h> /* funkcje API Windows */
#include <stdio.h> /* wyświetlanie */
#include <string.h> /* funkcja strcat */
#include <stdlib.h> /* funkcja exit */
#include <dos.h>
#include "ambex-55.h" /* struktury danych, definicje stałych i funkcji */
/* do komunikacji z driver'em */
****
/* deklaracje procedur Windows */
****
#include "resource.h"
int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
long FAR PASCAL __export MainWndProc(HWND, UINT, WPARAM, LPARAM);
**** deklaracje procedur ****
void askdriver(void) ;
void displayconfiguration(HDC,RECT);
void PressOrClickAny(HDC,RECT);
int displaytext(HDC,RECT);
int displayinterrupts(HDC,RECT);

```

```

int displayresults(HDC,RECT);
void drivererror(char status);
void quit(void);
void drawscreen(void);
void GoToLine(RECT *, HDC);
/* struktura danych dla funkcji obsługi przerwania sprzętowego karty */
struct IRQRequest
{
    int counter; /* licznik przerwania */
    int type; /* typ : 1-komunikat 2-procedura obsługi */
    HWND hWnd; /* handler dla PostMessage */
    int modnum; /* nr-modułu */
};
/*procedura obsługi przerwania sprzętowego */
void FAR PASCAL newintPIODCU(struct IRQRequest FAR *);
/*procedura obsługi komunikatu wystąpienia przerwania sprzętowego */
void MesintPIODCU(void);
/**/
void readport(void);
void PortReadWrite(void);
void testDCUint(int);
void testPIOint(int);
void testinterrupts(int);
void writeportfail(void);
/* komunikat kolejnego kroku programu */
#define MY_MESSAGE WM_USER +1
/* komunikat powiadomienia o wystąpieniu przerwania sprzętowego */
#define MY_ISR_MESSAGE WM_USER +2

/***** definicje stałych *****/
#define TRUE 1
#define FALSE 0

/***** deklaracje zmiennych *****/
/* rekordy opisu poszczególnych zleceń */
struct lc0_init_55 s_init = {MODULE_INIT_55};
struct lc0_total_55 s_total = {GET_TOTAL_CONFIGURATION_55};
struct lc0_module_55 s_module = {GET_MODULE_CONFIGURATION_55};
struct lc0_port_dir s_port_dir = {PORT_DIRECTION};
struct lc0_port_write s_port_write = {PORT_WRITE};
struct lc0_port_read s_port_read = {PORT_READ};
struct lc0_ctc_write_55 s_ctc_write = {CTC_WRITE_55};
struct lc0_ctc_gate s_ctc_gate = {CTC_GATE};
struct lc0_int_mask s_int_mask = {INTERRUPT_MASK};
struct lc0_int_level s_int_level = {INTERRUPT_LEVEL};
struct lc0_int_service s_int_service = {INTERRUPT_SERVICE_55};
struct lc0_int_reset s_int_reset = {INTERRUPT_RESET};
struct lc0_leave_55 s_leave = {LEAVE_DRIVER_55};
int modulenum; /* numer badanego modulu */
int koniec = 0;
int sstat; /* slowo komunikacyjne dla funkcji obsługi */
int paintnum = 0; /* nr sekcji do wyświetlenia w komunikacie WM_PAINT*/
int paintdet = 0;
int enddet = 0;
unsigned char tstart;
struct IRQRequest ISRData; /* parametry funkcji obsługi przerwania sprzętowych karty */
HWND hWnd; /* uchwyt do głównego okna programu */
HANDLE hInst; /* deskryptor bieżącego zadania*/
HANDLE hDriverInst; /* deskryptor drivera */

```

```

unsigned char ReadVal[10];      /*tablica odczytanych danych z portu A */
int ReadCount=0;

                                /* ----- teksty błędów ----- */
char   *DriverErrors[] =
{
                                /* LC0_UNKN_FUNC          -1 */
"LC0_UNKN_FUNC: Nieznany kod funkcji",
                                /* LC0_NO_MODULE        -2 */
"LC0_NO_MODULE: Brak lub błędny numer modułu",
                                /* LC0_NOT_OUTPORT     -3 */
"LC0_NOT_OUTPORT: Port nie jest portem wyjściowym",
                                /* LC0_NONEX_DEV      -4 */
"LC0_NONEX_DEV: Błędny numer urządzenia",
                                /* LC0_NOT_INPORT     -5 */
"LC0_NOT_INPORT: Port nie jest portem wejściowym",
                                /* LC0_BAD_COUNTER   -6 */
"LC0_BAD_COUNTER: Zły numer kanału CTC",
                                /* LC0_NO_LATCH      -7 */
"LC0_NO_LATCH: Moduł nie może zatrzymać danych",
""
,
""
,
""
,
                                /* LC0_DEV_BUSY      -11 */
"LC0_DEV_BUSY: Urządzenie jest zajęte",
""
,
""
,
""
,
""
,
""
,
""
,
""
,
""
,
                                /* LC0_BAD_PROC      -19 */
"LC0_BAD_PROC: Błędny adres procedury obsługi przerwania",
""
,
""
,
""
,
""
,
""
,
                                /* LC0_NOT_SUPPORTED -24 */
"LC0_NOT_SUPPORTED: Funkcja nie jest realizowana",
                                /* LC0_BAD_CTC_MODE  -25 */
"LC0_BAD_CTC_MODE: Błędny tryb pracy CTC",
""
,
""
,
""
,
""
,
                                /* LC0_NO_IRQ        -29 */
"LC0_NO_IRQ: Z modułem nie jest związane żadne przerwanie",
                                /* LC0_NOT_FULLY_SUP -30 */
"LC0_NOT_FULLY_SUP: Funkcja w opracowaniu",
""
,
""
,
""
,
                                /* LC0_INTR_INST     -33 */
"LC0_INTR_INST: Próba powtórnej instalacji przerwania",
""
,
""
,
""
,
""
,
                                /* LC0_CTC_NOT_PROGRAMMED -37 */
"LC0_CTC_NOT_PROGRAMMED: Zapis licznika przy niezaprogramowanym trybie pracy",
                                /* LC0_REJECTED      -38 */

```

```

"LC0_REJECTED: Za duzo jednoczesnych odwozań do driver'a",
/* LC0_BAD_CONFIG -39 */
"LC0_BAD_CONFIG: Bład w pliku konfiguracyjnym AMBEX.INI",
/* LC0_NOT_INIT -40 */
"LC0_NOT_INIT: Moduł nie jest zainicjowany",
/* LC0_IRQ_BUSY -41 */
"LC0_IRQ_BUSY: Przerwanie sprzętowe modułu jest używane przez inny moduł",
};
char *DriverWarnings[] =
{
/* LC0_NON_EX_MOD 1 */
"LC0_NON_EX_MOD: Zażadano inicjalizacji nieistniejących modułów",
/* LC0_NO_LATCH_WARN 2 */
"LC0_NO_LATCH_WARN: Moduł nie może zatrząskiwac danych",
"",
"",
/* LC0_IS_INIT 5 */
"LC0_IS_INIT: Zażadano inicjacji zainicjowanego modułu",
};

```

```

/*****

```

Funkcja: WinMain(HANDLE, HANDLE, LPSTR, int)

Przeznaczenie: Inicjacja aplikacji, okna głównego, wczytanie drivera

Parametry wywołania :

hInstance	- deskryptor zadania
hPrevInstance	- deskryptor poprzedniego zadania
lpCmdLine	- string z parametrami wywołania
nCmdShow	- sposób wyświetlenia okna

```

*****/

```

```

int PASCAL WinMain
(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int nCmdShow)
{
MSG msg;
if (!hPrevInstance)
{
if (!InitApplication(hInstance)) /* rejestracja klasy okna */
return (FALSE);
}
if (!InitInstance(hInstance, nCmdShow)) /* aktywacja okna */
return (FALSE);
MessageBox(hWnd,"Zamknij by rozpocząć przykładową sekwencję pomiarowa","Informacja",MB_OK);
paintnum = 1;
PostMessage(hWnd,MY_MESSAGE,0,0);
/* pętla obsługi komunikatów */
while (GetMessage(&msg, NULL, NULL, NULL))
{
TranslateMessage(&msg);
DispatchMessage(&msg);
}
/* wykonanie funkcji LEAVE_DRIVER i zwolnienie drivera */
quit();
return (msg.wParam);
}

```

Funkcja: InitApplication(HANDLE)

Przeznaczenie: Inicjacja i rejestracja klasy okna głównego programu .

Rezultat: Status funkcji RegisterClass(); !=0 gdy O.K.

*****/

BOOL InitApplication(HANDLE hInstance)

```
{
    WNDCLASS wc;
    wc.style = NULL;
    wc.lpfnWndProc = MainWndProc;          /*procedura okienkowa */
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(hInstance,MAKEINTRESOURCE(IDC_AMBEX));
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = COLOR_WINDOW+1;
    wc.lpszMenuName = "MainMenu";
    wc.lpszClassName = "DemoDriverWClass";
    return (RegisterClass(&wc));
}
```

Funkcja: InitInstance(HANDLE, int)

Przeznaczenie: Stworzenie okna głównego programu

Rezultat: Status funkcji CreateWindow() ; != gdy O.K.

*****/

BOOL InitInstance(HANDLE hInstance,int nCmdShow)

```
{
    hInst = hInstance;
    /* stworzenie okna glownwgo */
    hWnd = CreateWindow(
        "DemoDriverWClass",
        "Przykład użycia biblioteki DLL Windows do kart cyfrowych LC-055",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance,
        NULL
    );
    if (!hWnd)
        return (FALSE);
    /*wyświetlenie okna */
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
}
```

```

ISRData.counter=0;
ISRData.hWnd=hWnd;
return (TRUE);
}

```

Funkcja: MainWndProc(HWND, UINT, WPARAM, LPARAM)

Przeznaczenie: Funkcja odpowiada okna na komunikaty .

Komunikaty:

```

WM_COMMAND- Menu okna
WM_DESTROY- Zniszczenie okna i następnie powrót do Windows
WM_PAINT - Odmalowywanie okna
WM_LBUTTONDOWN - Wciśnięcie lewego klawisza myszy
WM_KEYDOWN - Wciśnięcie klawisza
MY_MESSAGE - wywołanie kolejnej procedury programu
MY_ISR_MESSAGE - komunikat informujący o przerwaniu sprzętowym
WM_TIMER - komunikat timera 1s

```

```

long FAR PASCAL __export MainWndProc(HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam)
{
HDC dc ;
RECT rect ;
static int top=0;
PAINTSTRUCT ps ;
switch (message)
{
case WM_COMMAND:          /* komenda menu */
switch (wParam)
{
case CM_TEST :          /* powtórzenie pomiarów */
if(paintnum==0)
{
paintnum = 1;
drawscreen();
}
break;
case CM_EXIT:          /* wyjście z programu */
DestroyWindow(hWnd);
break;
default:
return (DefWindowProc(hWnd, message, wParam, lParam));
}
break;
case WM_PAINT :          /* wyświetlenie informacji */
dc = BeginPaint(hWnd,&ps);
GetClientRect(hWnd,&rect);
switch(paintnum)
{
case 1 :          /*konfiguracja*/
displayconfiguration(dc,rect);
break ;
case 2 :
case 3 :
case 4 :

```

```

case 5 :
rect.top=displaytext(dc,rect);
rect.left = 10 ;
/* wyświetlenie rezultatów pomiarów */
rect.top=displayresults(dc,rect);
/*wyświetlenie zawartości licznika przerwania */
rect.top=displayinterrupts(dc,rect);
top=rect.top;
if(enddet>0)
{
PressOrClickAny(dc,rect);
}
break ;
}
EndPoint(hwnd,&ps);
break ;
case WM_LBUTTONDOWN : /* kliknięcie myszą w oknie */
case WM_KEYDOWN : /* naciśnięcie klawisza */
enddet=0;
if(paintnum!=0)
{
if(paintnum==4 || paintnum==5) /* wyinstalowanie przerwania po zakończeniu testu*/
{
ISRData.counter=0;
ISRData.type=0;
/* zablokowanie kanałów CTC dla LC-055-DCU */
if(s_module.LC0_MTYPE == LC_055_DCU)
{
s_ctc_gate.LC0_CGGATE = 0x07;
/* CTC_GATE */
LC055_CTCGate((struct lc0_ctc_gate FAR *)&s_ctc_gate);
}
/* zablokowanie przerwania */
s_int_mask.LC0_IMASK = 0;
LC055_InterruptMask((struct lc0_int_mask FAR *)&s_int_mask);
if(s_int_service.LC0_STATUS == LC0_OK)
{
s_int_service.LC0_IMODE=0;
LC055_InterruptService((struct lc0_int_service FAR *)&s_int_service);
}
}
paintnum++;
if(paintnum==6)
{
paintnum = 0 ;
}
else
PostMessage(hwnd,MY_MESSAGE,0,0);
drawscreen();
}
break ;
case MY_MESSAGE :
enddet = 0 ;
switch(paintnum)
{
case 0 :
break ;
case 1 :
askdriver(); /*odpytanie driver'a o konfiguracje*/

```

```

        drawscreen(); /*wyswietlenie konfiguracji modulu i toru a/c poprzez funkcję displayconfiguration*/
        break ;
    case 2 :
        readport() ; /* odczyt portu cyfrowego */
        break ;
    case 3 :
        if(ReadCount!=11)
            KillTimer(hWnd,1);
        ReadCount=0;
        writeportfail(); /* zapis na port wejsciowy - blad */
        break ;
    case 4 :
        testinterrupts(4);/* test przerwan przychodzacych z modulu - funkcja */
        break ;
    case 5 :
        testinterrupts(5);/* test przerwan przychodzacych z modulu - komunikat */
        break ;
    }
    break ;
case WM_DESTROY: /* koniec programu */
    PostQuitMessage(0);
    break;
case WM_TIMER:
    PortReadWrite();
    break;

/* informacja o przerwaniu */
case MY_ISR_MESSAGE:
    if(wParam==1) /* bezpośrenie wysłanie komunikatu z driver'a */
    {
        /* modyfikujemy zmienne */
        ISRData.type=1;
        ISRData.counter++;
    }
    if(wParam==2) /* wysłanie komunikatu z procedury obsługi przerwania */
    {
        /* nie musimy ustawiać zmiennych bo zrobiła to funkcja obsługi przerwania
*/
    }

    /* odmalowanie okien */

    GetClientRect(hWnd,&rect);
    rect.top=top-20;
    rect.bottom=top;
    InvalidateRect(hWnd,&rect,TRUE);
    break;
default:
    return (DefWindowProc(hWnd, message, wParam, lParam));
}
return (NULL);
}

```

```

/*****

```

Funkcja quit()

Przeznaczenie : Wykonanie funkcji LEAVE_DRIVER

```

*****/

```

void quit()

```

{
    LC055_LeaveDriver((struct lc0_leave_55 FAR *)&s_leave);
}

```

```

}

/*****
GoToLine(*RECT) 0 przejście do następnej linii przy wyświetlaniu tekstu
*****/
void GoToLine(RECT *rect, HDC hdc)
{
    TEXTMETRIC tm ;
    GetTextMetrics(hdc,&tm);

    rect->top +=tm.tmHeight + tm.tmExternalLeading ;
    rect->bottom +=tm.tmHeight + tm.tmExternalLeading ;
}

/*****
/* Funkcja PressOrClickAny(HDC,RECT) */
/* Przeznaczenie: */
/* Wypisuje na ekranie informacje o końcu kolejnego kroku programu */
*****/
void PressOrClickAny(HDC dc, RECT rect)
{
    char line[100];
    GoToLine(&rect,dc);GoToLine(&rect,dc);
    sprintf(line,"Wciśnij dowolny klawisz lub kliknij by przejść dalej ... ");
    DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
}

/*****
Funkcja : drawscreen
Przeznaczenie : Wybranie odpowiedniej sekcji informacji do wyświetlenia
na ekranie w odpowiedzi na komunikat WM_PAINT
Parametr : przepisuje do zmiennej globalnej paintnum
*****/
void drawscreen(void)
{
    InvalidateRect(hWnd,NULL,TRUE);
    UpdateWindow(hWnd);
}

/*****
/* Przeznaczenie: */
/* Rozpoznanie konfiguracji badanego modulu. */
/* Sposob: */
/* Przez wykorzystanie funkcji driver'a GET_TOTAL_CONFIGURATION_55, */
/* GET_MODULE_CONFIGURATION_55. */
/* Uwagi: */
/* Funkcja nadaje wartosc zmiennej modulenum (numer badanego modulu) */
/* wypelnia struktury total, module, info i inicjalizuje zainstalowane */
/* moduly. Jezeli zainstalowanych jest wiecej niz jeden modulow to */
/* program bedzie analizowal tylko pierwszy z brzegu. */
*****/
void askdriver(void)
{
    /* GET_TOTAL_CONFIGURATION_55 */
    LC055_GetTotalConfiguration((struct lc0_total_55 FAR *)&s_total);
    /* inicjalizacja zainstalowanych modulow */
    s_init.LC0_IMODULE = s_total.LC0_TONF & 0xF;
    /* MODULE_INIT */

```

```

LC055_ModuleInit((struct lc0_init_55 FAR *)&s_init);
/* sprawdzenie, ktory modul jest */
/* zainstalowany: A,B,C,D,E,F,G czy H */
for(modulenum = 1; modulenum <= 8; modulenum++)
if(s_total.LC0_TONF & (1 << (modulenum - 1)))
break;
/* spytanie o konfiguracje modulu */
s_module.LC0_MMODULE = modulenum;
/* GET_MODULE_CONFIGURATION_55*/
LC055_GetModuleConfiguration((struct lc0_module_55 FAR *)&s_module);
/* pozostale zmienne struktury parametrów obsługi przerwania */
ISRData.modnum = modulenum;
}

/*****
/* Funkcja : displayconfiguration(HDC,RECT) */
/* Przeznaczenie: */
/* Wswietlenie konfiguracji badanego modulu i jego toru a/c. */
/* Sposob: */
/* Przez wykorzystanie informacji zawartych w strukturach module i info. */
*****/
void displayconfiguration(HDC dc,RECT rect)
{
int i;
char line[200],spar[100];
rect.left +=10 ;
DrawText(dc,"Konfiguracja modulu:",-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc); GoToLine(&rect,dc);
switch(s_module.LC0_MTYPE)
{
case LC_055_PIO :sprintf(spar,"LC-055-PIO");
break;
case LC_055_DCU :sprintf(spar,"LC-055-DCU");
break;
}
sprintf(line,"Modul %c : typ %s",'A'+modulenum-1,spar);
DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc);
sprintf(line,"Adres modulu: %04X (hex)", s_module.LC0_MBASE);
DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc);
sprintf(line,"Liczba portów we/wy: %d", s_module.LC0_MPORT);
DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc);
sprintf(line,"Liczba kanałów CTC: %d", s_module.LC0_MCTC);
DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc);
if(s_module.LC0_MCTC != 0)
{
if(s_module.LC0_MTYPE == LC_055_PIO)
{
if(s_module.LC0_MCLOCK[0] == 0)
sprintf(line,"Wszystkie kanały CTC podłączone do zegara zewnętrznego");
else
sprintf(line,"Częstotliwość zegara CTC: %d kHz",s_module.LC0_MCLOCK[0]);
DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc);
}
}
else

```

```

    {
    for(i = 0; i < s_module.LC0_MCTC; i++)
    {
        if(s_module.LC0_MCLOCK[i] == 0)
            sprintf(line,"Kanał %d CTC podłączony do zegara zewnętrznego", i);
        else
            sprintf(line,"Częstotliwość zegara kanału %d CTC: %d kHz",
                    i, s_module.LC0_MCLOCK[i]);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
    }
    }
    if(s_module.LC0_MINT == 0xFF)
        sprintf(line,"Moduł nie jest podłączony do żadnego przerwania");
    else
        sprintf(line,"Numer przerwania sprzętowego związanego z modulem: IRQ %d",
        ,s_module.LC0_MINT);

    DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
    GoToLine(&rect,dc);
    ReleaseDC(dc,hWnd);
}

/*****
/* Funkcja : displaytext(HDC,RECT,unsigned char) */
/* Przeznaczenie: */
/* Wyszwietlenie komunikatów tekstowych dla kolejnych kroków programu */
*****/
int displaytext(HDC dc,RECT rect)
{
    char line[200] ;
    int i;
    rect.left += 10;
    switch(paintnum)
    {
        case 2 : /* readport*/
            sprintf(line,"Przesłanie 10 bajtów danych z portu wejściowego na wyjściowy:");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            GoToLine(&rect,dc);
            sprintf(line,"Port A (1) - wejście ; Port B (2) - wyjście ; pozostałe bez zmian");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            break ;
        case 3 : /* writefail */
            sprintf(line,"Przesyłanie danych do portu wejściowego (błąd!);");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            GoToLine(&rect,dc);
            sprintf(line,"Port A (1) - wejście ; pozostałe bez zmian");
            break ;
        case 4 : /* testinterrupts - funkcja*/
            sprintf(line,"Testowanie przerwń przychodzących z modułu : obsługa bezpośrednia poprzez funkcję ");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            GoToLine(&rect,dc);
            sprintf(line,"Wciśnij dowolny klawisz lub kliknij by przerwać ...");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            break ;
        case 5 : /* testinterrupts - obsługa komunikatu*/
            sprintf(line,"Testowanie przerwń przychodzących z modułu : obsługa komunikatu wygenerowanego przez driver");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
    }
}

```

```

        GoToLine(&rect,dc);
        sprintf(line,"Wciśnij dowolny klawisz lub kliknij by przerwać ...");
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        break ;
    }
    GoToLine(&rect,dc);
    /* rozpoznanie źródeł przerwania na poszczególnych liniach dla modułu LC-055-DCU*/
    if(paintnum==4 || paintnum==5)
    if(s_module.LC0_MTYPE == LC_055_DCU)
        for(i = 0; i < LC0_MAX_INT_LINES; i++)
        {
            switch((s_module.LC0_MINTSRC >> (2 * i)) & 3)
            {
                case LC0_NO_INT:
                    sprintf(line,"Linia %d : niepodłączona", i + 1);
                    break;
                case LC0_INTSRC_EXT:
                    sprintf(line,"Linia %d : przerwania zewnętrzne", i + 1);
                    break;
                case LC0_INTSRC_CTC:
                    sprintf(line,"Linia %d : przerwania od CTC", i + 1);
                    break;
            }
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            GoToLine(&rect,dc);
        }
    return(rect.top);
}

/*****
/* Funkcja : displayinterrupts(HDC,RECT,unsigned char) */
/* Przeznaczenie: */
/* Wświetlenie licznika zgłoszeń przerwania sprzętowych */
*****/
int displayinterrupts(HDC dc,RECT rect)
{
    char line[200],spar[100] ;
    rect.left += 10;
    if(paintnum==4 || paintnum==5)
    {
        switch(ISRData.type)
        {
            case 0 :sprintf(spar,"Brak zgłoszeń");
                    break;
            case 1 :
                    sprintf(spar,"Obsługa komunikatu wysłanego przez driver");
                    break;
            case 2 :
                    sprintf(spar,"Obsługa bezpośrednia - funkcja ");
                    break;
        }
        sprintf(line,"Ilość zgłoszeń przerwania : %d ; typ obsługi : %s",ISRData.counter,spar);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
    }
    return(rect.top);
}

/*****

```

```

/* Funkcja : displayresults(HDC,RECT,unsigned char) */
/* Przeznaczenie: */
/* Wyszwietlenie danych odczytanych z portu A */
/*****
int displayresults(HDC dc,RECT rect)
{
char line[200],spar[100];
unsigned char maska[]={1,2,4,8,16,32,64,128};
int i,j;
rect.left += 10;
if(paintnum==2)
{
sprintf(line,"Port A - bit 8 ... 1" );
DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc);
for(i=0;i<ReadCount;i++)
{
if(i==10)
break;
for(j=0;j<8;j++)
if((ReadVal[i] & maska[j])==0)
spar[7-j]='0';
else
spar[7-j]='1';
spar[8]='\0';
sprintf(line," %d %s",i+1,spar);
DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
GoToLine(&rect,dc);
}
}
return(rect.top);
}

/*****
/* Przeznaczenie: */
/* Wypisanie komunikatow o bledzie / ostrzezeniu */
/* bledzie. */
/* Parametry: */
/* status - LC0_STATUS */
/*****
void drivererror(char status)
{
char sts[100];
if(status > 0)
sprintf(sts,"Ostrzezenie: %s",(LPSTR)DriverWarnings[status - 1]);
else
sprintf(sts,"Bład: %s", (LPSTR)DriverErrors[-status - 1]);
MessageBox(hWnd,sts,"Sygnał",MB_OK);
}

/*****
/* Przeznaczenie: */
/* Odczyt portu 1 (A) zaprogramowanego jako wejsciowy i wyslanie */
/* odczytanej wartosci na port 2 (B) zaprogramowany jako wyjsciowy. */
/*****
void readport(void)
{
s_port_dir.LC0_PMODULE = modulenum;
s_port_dir.LC0_PDIR = (LC0_IN_DIRECTION | LC0_CHANGE_DIR) |

```

```

        ((LC0_OUT_DIRECTION | LC0_CHANGE_DIR) << 2);
        /* PORT_DIRECTION */
LC055_PortDirection((struct lc0_port_dir FAR *)&s_port_dir);
if(s_port_dir.LC0_STATUS!=LC0_OK)
{
    enddet=1;
    drivererror(s_port_dir.LC0_STATUS);
    drawscreen();
    return;
}
    /* przygotowanie stałych pol      */
    /* rekordow opisow zleceń        */
s_port_read.LC0_PRMODULE = modulenum;
s_port_read.LC0_PRPORT = 1;
s_port_read.LC0_PRLATCH = LC0_LATCH_OFF;
s_port_write.LC0_PWMODULE = modulenum;
s_port_write.LC0_PWPORT = 2;
    /*ustawienie timera na 1s*/
SetTimer(hWnd,1,1000,0);
}

/*****
Procedura wywoływana przez timer co 1s : odczyt danych z portu A i zapis na
port B
*****/

*****/
void PortReadWrite()
{
    /* PORT_READ */
LC055_PortRead((struct lc0_port_read FAR *)&s_port_read);
if(s_port_read.LC0_STATUS!=LC0_OK)
{
    KillTimer(hWnd,1);
    ReadCount=0 ;//koniec
    enddet=1;
    drivererror(s_port_read.LC0_STATUS);
    drawscreen();
    return;
}
ReadVal[ReadCount]= s_port_read.LC0_PRDATA;
s_port_write.LC0_PWDATA = s_port_read.LC0_PRDATA;
    /* PORT_WRITE */
LC055_PortWrite((struct lc0_port_write FAR *)&s_port_write);
if(s_port_write.LC0_STATUS!=LC0_OK)
{
    KillTimer(hWnd,1);
    ReadCount=0 ;//koniec
    enddet=1;
    drivererror(s_port_write.LC0_STATUS);
    drawscreen();
    return;
}
ReadCount++;
/* gdy koniec */
if(ReadCount==11)
{
    KillTimer(hWnd,1);
    enddet=1;
}
}

```

```

drawscreen();
}

/*****
/* Przeznaczenie:
/* Zapis na port zaprogramowany jako wejsciowy - blad.
*****/

void writeportfail(void)
{
s_port_dir.LC0_PMODULE = modulenum;
/* port A - wejsciowy */
s_port_dir.LC0_PDIR = (LC0_IN_DIRECTION | LC0_CHANGE_DIR);
/* PORT_DIRECTION */
LC055_PortDirection((struct lc0_port_dir FAR *)&s_port_dir);
if(s_port_dir.LC0_STATUS!=LC0_OK)
{
enddet=1;
drivererror(s_port_dir.LC0_STATUS);
drawscreen();
return;
}
s_port_write.LC0_PWMODULE = modulenum;
s_port_write.LC0_PWPORT = 1;
s_port_write.LC0_PWDATA = 0;
/* PORT_WRITE */
LC055_PortWrite((struct lc0_port_write FAR *)&s_port_write);
if(s_port_write.LC0_STATUS!=LC0_OK)
{
enddet=1;
drivererror(s_port_write.LC0_STATUS);
drawscreen();
return;
}
}

/*****
/* Przeznaczenie:
/* Testowanie obsługi przerwanych przychodzących z modulu.
*****/

void testinterrupts(int nr)
{
if(s_module.LC0_MINT == 0xFF)
{
MessageBox(hWnd,"Z modulem nie jest związane żadne przerwanie!", "Info", MB_OK);
enddet=1;
drawscreen();
return;
}

/* w zależności od typu modulu procedury
/* testujące mają różną postać
*/
switch(s_module.LC0_MTYPE)
{
case LC_055_PIO:
testPIOint(nr);
break;
case LC_055_DCU:
testDCUint(nr);
break;
}
}

```

```

}

/*****
/* Przeznaczenie: */
/* Testowanie przerwan z modulu LC-055-PIO. */
/* Sposob: */
/* Przez wywołanie funkcji INTERRUPT_SERVICE_55 */
/* nr == 4 - ustawienie obsługi poprzez wywołanie funkcji */
/* nr == 5 - ----- rejestrację komunikatu */
/* Przerwanie musi być wygenerowane z zewnątrz */
*****/
void testPIOint(int nr)
{
    /* instalacja obsługi przerwania */
    s_int_service.LC0_ISMODULE = modulenum;
    if(nr==4) /*funkcja obsługi*/
    {
        /* rejestracja funkcji obsługi przerwania */
        s_int_service.LC0_IPROC = (void(pascal *))(void *)newintPIODCU;
        /* wskaźnik do struktury danych z parametrami dla funkcji obsługi przerwania */
        s_int_service.LC0_DATAPTR = (void far*)&ISRData;
        /* tryb pracy : instalacja funkcji klienta i wywołanie INTERRUPT_RESET*/
        s_int_service.LC0_IMODE=LC0_IM_INST | LC0_IM_CALL | LC0_IM_IREN ;
    }
    else /*komunikat driver'a*/
    {
        /* tryb pracy : instalacja komunikatu PostMessage, wywołanie INTERRUPT_RESET */
        s_int_service.LC0_IMODE=LC0_IM_INST | LC0_IM_POST | LC0_IM_PMES | LC0_IM_IREN ;
        /* parametry komunikatu wstawionego przez driver */
        s_int_service.LC0_HWND = hWnd;
        s_int_service.LC0_HTASK = GetCurrentTask();
        s_int_service.LC0_UMSG = MY_ISR_MESSAGE;
        s_int_service.LC0_WPARAM=1;
        s_int_service.LC0_LPARAM=0;
    }
    /* instalacja obsługi : INTERRUPT_SERVICE_55 */
    LC055_InterruptService((struct lc0_int_service FAR *)&s_int_service);
    if(s_int_service.LC0_STATUS!=LC0_OK)
    {
        enddet=1;
        drivererror(s_int_service.LC0_STATUS);
        drawscreen();
        return;
    }
    /* zezwolenie na przerwaniu : INTERRUPT_MASK*/
    s_int_mask.LC0_IMMODULE = modulenum;
    s_int_mask.LC0_IMASK = 1;
    LC055_InterruptMask((struct lc0_int_mask FAR *)&s_int_mask);
    if(s_int_mask.LC0_STATUS!=LC0_OK)
    {
        enddet=1;
        drivererror(s_int_mask.LC0_STATUS);
        drawscreen();
        return;
    }
    /* przerwania są odblokowane */
}

```

```

/*****
/* Przeznaczenie:
/* Testowanie przerwan z modulu LC-055-DCU.
/* Sposob:
/* Przez wywołanie funkcji INTERRUPT_SERVICE_55
/* nr == 4 - ustawienie obsługi poprzez wywołanie funkcji
/* nr == 5 - ----- rejestrację komunikatu
/* W przypadku linii przerywajacych polaczonych z CTC program odpowiednio
/* steruje CTC tak, aby osiagnac generowanie przerwan z czestotliwoscia
/* 20 Hz. W pozostalych przypadkach przerwanie musi byc oczywiscie
/* generowane z zewnatrz.
*****/
void testDCUint(int nr)
{
/* zaprogramowanie wszystkich kanalow CTC
/* na prace w trybie fali prostokatnej o
/* czestotliwosci 20 Hz
/* jezeli ktorakolwiek z linii
/* przerywajacych jest przylaczona do CTC
/* to przerwania beda generowane
/* samoczynnie
s_ctc_write.LC0_CMODULE = modulenum;
s_ctc_write.LC0_CMODE = LC0_SET_COUNTER_VALUE | LC0_SET CTC_MODE;
s_ctc_write.LC0_CCTRL = CTC_COUNT0 | CTC_NB | CTC_MODE3 | CTC_BOTH;
s_ctc_write.LC0_CWDATA = (unsigned int)((long)(s_module.LC0_MCLOCK[0]) * 50);
/* CTC_WRITE_55 */
LC055_CTCWrite((struct lc0_ctc_write_55 FAR *)&s_ctc_write);
s_ctc_write.LC0_CCTRL = CTC_COUNT1 | CTC_NB | CTC_MODE3 | CTC_BOTH;
s_ctc_write.LC0_CWDATA = (unsigned int)((long)(s_module.LC0_MCLOCK[1]) * 50);
/* CTC_WRITE_55 */
LC055_CTCWrite((struct lc0_ctc_write_55 FAR *)&s_ctc_write);
s_ctc_write.LC0_CCTRL = CTC_COUNT2 | CTC_NB | CTC_MODE3 | CTC_BOTH;
s_ctc_write.LC0_CWDATA = (unsigned int)((long)(s_module.LC0_MCLOCK[2]) * 50);
/* CTC_WRITE_55 */
LC055_CTCWrite((struct lc0_ctc_write_55 FAR *)&s_ctc_write);
if(s_ctc_write.LC0_STATUS!=LC0_OK)
{
enddet=1;
drivererror(s_ctc_write.LC0_STATUS);
drawscreen();
return;
}
/* zezwolenie na prace wszystkich kanalow CTC
s_ctc_gate.LC0_CGMODULE = modulenum;
s_ctc_gate.LC0_CGGATE = 0;
/* CTC_GATE */
LC055_CTCGate((struct lc0_ctc_gate FAR *)&s_ctc_gate);
if(s_ctc_gate.LC0_STATUS!=LC0_OK)
{
enddet=1;
drivererror(s_ctc_gate.LC0_STATUS);
drawscreen();
return;
}
}
/* instalacja obsługi przerwania */
s_int_service.LC0_ISMODULE = modulenum;
if(nr==4) //funkcja obsługi
{

```

```

        /* rejestracja funkcji obsługi przerwania */
s_int_service.LC0_IPROC = (void(pascal *))(void *)newintPIODCU;
        /* wskaźnik do struktury danych z parametrami dla funkcji obsługi przerwania */
s_int_service.LC0_DATAPTR = (void far*)(&ISRData);
        /* tryb pracy : instalacja funkcji klienta i wywołanie INTERRUPT_RESET*/
s_int_service.LC0_IMODE=LC0_IM_INST | LC0_IM_CALL | LC0_IM_IREN      ;
    }
else // komunikat driver'a
    {
        /* tryb pracy : instalacja komunikatu PostMessage, wywołanie INTERRUPT_RESET */
s_int_service.LC0_IMODE=LC0_IM_INST | LC0_IM_POST | LC0_IM_PMES | LC0_IM_IREN      ;
        /* parametry komunikatu wstawionego przez driver */
s_int_service.LC0_HWND = hWnd;
s_int_service.LC0_HTASK = GetCurrentTask();
s_int_service.LC0_UMSG = MY_ISR_MESSAGE;
s_int_service.LC0_WPARAM=1;
s_int_service.LC0_LPARAM=0;
    }

    /* instalacja obsługi : INTERRUPT_SERVICE_55 */
LC055_InterruptService((struct lc0_int_service FAR *)&s_int_service);
if(s_int_service.LC0_STATUS!=LC0_OK)
    {
        enddet=1;
        drivererror(s_int_service.LC0_STATUS);
        drawscreen();
        return;
    }

    /* zaprogramowanie aktywnych poziomów przerywających */
s_int_level.LC0_ILMODULE = modulenum;
s_int_level.LC0_ILEVEL = 0; /* wszystkie aktywne poziomy - 0 */
    /* INTERRUPT_LEVEL */
LC055_InterruptLevel((struct lc0_int_level FAR *)&s_int_level);
if(s_int_level.LC0_STATUS!=LC0_OK)
    {
        enddet=1;
        drivererror(s_int_level.LC0_STATUS);
        drawscreen();
        return;
    }

    /* zezwolenie na przerwanie : INTERRUPT_MASK*/
s_int_mask.LC0_IMMODULE = modulenum;
s_int_mask.LC0_IMASK = 0x0f; /* zezwolenie na przerwanie */
LC055_InterruptMask((struct lc0_int_mask FAR *)&s_int_mask);
if(s_int_mask.LC0_STATUS!=LC0_OK)
    {
        enddet=1;
        drivererror(s_int_mask.LC0_STATUS);
        drawscreen();
        return;
    }

    /* przerwania są odblokowane */
}

```

Procedura obsługi przerwania wołana bezpośrednio przez driver
 ptr - wskaźnik do struktury danych . Funkcja wywoływana jest przez driver
 w czasie obsługi przerwania sprzętowego : parametr ptr == LC0_DATAPTR podczas
 rejestracji funkcji (INTERRUPT_SERVICE) .

UWAGA :

Procedura obsługi przerwania sprzętowego karty nie powinna bezpośrednio odwoływać się do zmiennych globalnych i powinna znajdować się w segmencie kodu oznaczonym jako FIXED . Wszystkie zmienne globalne modyfikowane przez tą procedurę powinny znajdować się również w segmencie danych oznaczonym jako FIXED . Jedynymi zalecanymi przez MICROSOFT funkcjami WINDOWS API, które mogą być wywołane bezpiecznie przez procedurę obsługi przerwania są : PostMessage i PostAppMessage .

```

*****/
void FAR PASCAL newintPIODCU(struct IRQRequest FAR *ptr)
{
    struct lc0_int_reset SIntRes;
        /* licznik przerwania */
    ptr->counter++;
        /* typ wywołania : 2 == funkcja */
    ptr->type=2 ;
    /******/
    /* można tutaj również wywołać funkcję driver'a np. INERRUPT_RESET*/
    /******/
    SIntRes.LC0_IRMODULE=ptr->modnum;
    LC055_InterruptReset((struct lc0_int_reset FAR *)&SIntRes);

        /* odrysowanie okna z nowymi informacjami */
        /* bezpieczna funkcja API !!! WPARAM==2 -> tzn. funkcja*/
    PostMessage(ptr->hWnd,MY_ISR_MESSAGE,2,0);
}

/******/
Plik definicji modułu : WSMP055.DEF
*****/
NAME    WSmp055
DESCRIPTION 'Program przykładowy z użyciem DLL dla kart cyfrowych LC-055'
EXETYPE WINDOWS
STUB    'WINSTUB.EXE'
CODE PRELOAD FIXED NONDISCARDABLE
DATA PRELOAD FIXED NONDISCARDABLE MULTIPLE
HEAPSIZE 4096
STACKSIZE 8192
EXPORTS
    MAINWNDPROC
IMPORTS
    LC055_MODULEINIT=lc055a.16
    LC055_GETTOTALCONFIGURATION=lc055a.17
    LC055_GETMODULECONFIGURATION=lc055a.18
    LC055_PORTDIRECTION=lc055a.19
    LC055_PORTWRITE=lc055a.20
    LC055_PORTREAD=lc055a.21
    LC055_PORTLATCH=lc055a.22
    LC055_PORTRESET=lc055a.23
    LC055_CTCWRITE=lc055a.24
    LC055_CTCGATE=lc055a.25
    LC055_CTCREAD=lc055a.26
    LC055_INTERRUPTMASK=lc055a.27
    LC055_INTERRUPTLEVEL=lc055a.28
    LC055_INTERRUPTSERVICE=lc055a.29
    LC055_INTERRUPTRESET=lc055a.30
    LC055_READSTATUS=lc055a.31
    LC055_LEAVEDRIVER=lc055a.32

```


DODATEK C

do dokumentacji biblioteki DLL

modułów cyfrowych

LC-055-PIO i LC-055-DCU

AMBEX-55.PAS

struktury danych i stałe dla Pascala

```
(*****
*)
(* Zestaw deklaracji struktur danych i definicji stałych do komunikacji *)
(* z driver'em Windows LC055A.DLL z poziomu języka PASCAL. *)
*)
(*****)
```

```
const
LC0_MAX_CTC      = 6;      (* maksymalna liczba kanałów CTC      *)
LC0_MAX_PORTS   = 6;      (* maksymalna liczba portów          *)
LC0_MAX_INT_LINES = 4;    (* maksymalna liczba linii           *)
                      (* przerywających                    *)
```

```
(***** REKORDY OPISÓW ZLECEŃ FUNKCJI DRIVER'A *****)
```

```
(* Rekord opisu zlecenia funkcji MODULE_INIT_55 ===== *)
```

```
type
lc0_init_55 =
  record
    LC0_CODE:      byte;      (* numer funkcji (16)          *)
    LC0_STATUS:   shortint;  (* kod odpowiedzi driver'a     *)
    LC0_IMODULE:  byte;      (* mapa modułów                *)
  end;
```

```
(* Rekord opisu zlecenia funkcji GET_TOTAL_CONFIGURATION_55 ===== *)
```

```
type
lc0_total_55 =
  record
    LC0_CODE:      byte;      (* numer funkcji (17)          *)
    LC0_STATUS:   shortint;  (* kod odpowiedzi driver'a     *)
    LC0_TONF:     byte;      (* inf. o zainstalowanych modułach *)
    LC0_TCTC:     byte;      (* liczba dostępnych kanałów    *)
    LC0_TPORT:    byte;      (* liczba dostępnych portów     *)
  end;
```

```
(* Rekord opisu zlecenia funkcji GET_MODULE_CONFIGURATION_55 ===== *)
```

```
type
lc0_module_55 =
  record
    LC0_CODE:      byte;      (* numer funkcji (18)          *)
    LC0_STATUS:   shortint;  (* kod odpowiedzi driver'a     *)
    LC0_MMODULE:  byte;      (* numer modułu                *)
    LC0_MTYPE:    byte;      (* typ modułu                  *)
    LC0_MBASE:    word;      (* adres bazowy rejestrow modułu *)
    LC0_MPORT:    byte;      (* liczba dostępnych portów     *)
    LC0_MCTC:     byte;      (* liczba dostępnych kanałów    *)
                      (* tablica częstotliwości zegara *)
                      (* CTC w kHz                    *)
    LC0_MCLOCK:   array [1..LC0_MAX_CTC] of word;
    LC0_CTCGATE:  byte;      (* bramki kanałów CTC          *)
    LC0_MINT:     byte;      (* numer przerwania (programowy) *)
    LC0_MINTSRC:  word;      (* określenie źródeł przerw    *)
    LC0_MINTLEV:  byte;      (* poziom przerywający         *)
    LC0_MINTMSK:  byte;      (* maski przerw                *)
  end;
```

```
const
LC0_055_PIO     = 1;      (* kody typów modułów          *)
                      (* LC-055-PIO                  *)
```

```

LC_055_DCU    = 2;                (* LC-055-DCU *)

(* kody zrodla przerwania *)
LC0_NO_INT    = 0;                (* brak przerwania *)
LC0_INTSRC_EXT = 1;                (* sygnal zewnetrzny *)
LC0_INTSRC_CTC = 2;                (* CTC *)

(* Rekord opisu zlecenia funkcji PORT_DIRECTION ===== *)
type
lc0_port_dir =
  record
    LC0_CODE:    byte;            (* numer funkcji (19) *)
    LC0_STATUS:  shortint;        (* kod odpowiedzi driver'a *)
    LC0_PMODULE: byte;            (* numer modulu *)
    LC0_PDIR:    longint;         (* kierunki pracy portow *)
  end;

const
(* kody kierunkow pracy portow *)
LC0_IN_DIRECTION    = 0;          (* bit kierunku : *)
LC0_OUT_DIRECTION   = 1;          (* port wejsciu *)
LC0_CHANGE_DIR      = 2;          (* bit trybu : *)
(* zmien kierunek pracy *)

(* Rekord opisu zlecenia funkcji PORT_WRITE ===== *)
type
lc0_port_write =
  record
    LC0_CODE:    byte;            (* numer funkcji (20) *)
    LC0_STATUS:  shortint;        (* kod odpowiedzi driver'a *)
    LC0_PWMODULE: byte;            (* numer modulu *)
    LC0_PWPORT:  byte;            (* numer portu *)
    LC0_PWDATA:  byte;            (* dana do wyslania *)
  end;

(* Rekord opisu zlecenia funkcji PORT_READ ===== *)
type
lc0_port_read =
  record
    LC0_CODE:    byte;            (* numer funkcji (21) *)
    LC0_STATUS:  shortint;        (* kod odpowiedzi driver'a *)
    LC0_PMODULE: byte;            (* numer modulu *)
    LC0_PRPORT:  byte;            (* numer portu *)
    LC0_PRLATCH: byte;            (* LC0_LATCH_ON/LC0_LATCH_OFF *)
    LC0_PRDATA:  byte;            (* dana odczytana *)
  end;

(* Rekord opisu zlecenia funkcji PORT_LATCH ===== *)
type
lc0_port_latch =
  record
    LC0_CODE:    byte;            (* numer funkcji (22) *)
    LC0_STATUS:  shortint;        (* kod odpowiedzi driver'a *)
    LC0_PLMODULE: byte;            (* numer modulu *)
    LC0_PLMODE:  byte;            (* tryb pracy *)
  end;

const

```

```

(* PORT_LATCH/LC0_PLMODE, PORT_READ/LC0_PRLATCH *)
LC0_LATCH_ON      = 1;          (* zatrzasnij *)
LC0_LATCH_OFF     = 0;          (* zwolnij *)

```

```
(* Rekord opisu zlecenia funkcji PORT_RESET ===== *)
```

```

type
lc0_port_reset =
  record
    LC0_CODE:      byte;          (* numer funkcji (23) *)
    LC0_STATUS:    shortint;      (* kod odpowiedzi driver'a *)
    LC0_PRSMODULE: byte;          (* numer modulu *)
  end;

```

```
(* Rekord opisu zlecenia funkcji CTC_WRITE_55 ===== *)
```

```

type
lc0_ctc_write_55 =
  record
    LC0_CODE:      byte;          (* numer funkcji (24) *)
    LC0_STATUS:    shortint;      (* kod odpowiedzi driver'a *)
    LC0_CMODULE:   byte;          (* numer modulu *)
    LC0_CMODE:     byte;          (* tryb pracy funkcji *)
    LC0_CCTRL:     byte;          (* bajt sterujacy *)
    LC0_CWDATA:    word;          (* wartosc licznika *)
  end;

```

```
const
```

```

(* wartosci fragmentow bajtu sterujacego 8253/8254 *)
CTC_NB      = 0;          (* kod naturalny binarny *)
CTC_BCD     = 1;          (* kod BCD *)

```

```

CTC_MODE0   = 0;          (* tryb 0 *)
CTC_MODE1   = 2;          (* tryb 1 *)
CTC_MODE2   = 4;          (* tryb 2 *)
CTC_MODE3   = 6;          (* tryb 3 *)
CTC_MODE4   = 8;          (* tryb 4 *)
CTC_MODE5   = 10;         (* tryb 5 *)

```

```

CTC_LSB     = $10;        (* tylko mlodszy bajt *)
CTC_MSB     = $20;        (* tylko starszy bajt *)
CTC_BOTH    = $30;        (* mlodszy - starszy *)

```

```

CTC_COUNT0  = $00;        (* licznik 0 *)
CTC_COUNT1  = $40;        (* licznik 1 *)
CTC_COUNT2  = $80;        (* licznik 2 *)

```

```

(* kody trybu pracy funkcji CTC_WRITE_55 *)
LC0_SET_CTC_MODE      = 1;          (* zaprogramuj tryb pracy kanalu *)
LC0_SET_COUNTER_VALUE = 2;          (* zaladuj nowa wartosc licznika *)

```

```
(* Rekord opisu zlecenia funkcji CTC_GATE ===== *)
```

```

type
lc0_ctc_gate =
  record
    LC0_CODE:      byte;          (* numer funkcji (25) *)
    LC0_STATUS:    shortint;      (* kod odpowiedzi driver'a *)
    LC0_CGMODULE:  byte;          (* numer modulu *)
    LC0_CGGATE:    byte;          (* maska aktywnych kanalow *)
  end;

```

```

(* Rekord opisu zlecenia funkcji CTC_READ_55 ===== *)
type
lc0_ctc_read_55 =
  record
    LC0_CODE:   byte;           (* numer funkcji (26)           *)
    LC0_STATUS: shortint;      (* kod odpowiedzi driver'a     *)
    LC0_CRMODULE: byte;        (* numer modulu                 *)
    LC0_CRCOUNT: byte;         (* numer kanału                 *)
    LC0_CRDATA: word;          (* odczytana dana              *)
  end;

(* Rekord opisu zlecenia funkcji INTERRUPT_MASK ===== *)
type
lc0_int_mask =
  record
    LC0_CODE:   byte;           (* numer funkcji (27)           *)
    LC0_STATUS: shortint;      (* kod odpowiedzi driver'a     *)
    LC0_IMODULE: byte;         (* numer modulu                 *)
    LC0_IMASK:  byte;           (* maska przerw                 *)
  end;

(* Rekord opisu zlecenia funkcji INTERRUPT_LEVEL ===== *)
type
lc0_int_level =
  record
    LC0_CODE:   byte;           (* numer funkcji (28)           *)
    LC0_STATUS: shortint;      (* kod odpowiedzi driver'a     *)
    LC0_ILMODULE: byte;        (* numer modulu                 *)
    LC0_ILEVEL: byte;          (* poziom przerywajacy na      *)
    LC0_ILEVEL: byte;          (* poszczegolnych liniach     *)
  end;

(* Rekord opisu zlecenia funkcji INTERRUPT_SERVICE_55 ===== *)
type
lc0_int_service =
  record
    LC0_CODE:   byte;           (* numer funkcji (29)           *)
    LC0_STATUS: shortint;      (* kod odpowiedzi driver'a     *)
    LC0_ISMODULE: byte;        (* numer modulu                 *)
    LC0_IPROC:   pointer;       (* adres procedury obsługi     *)
    LC0_DATAPTR: pointer;       (* wskaznik do bufora parametrow procedury *)
    LC0_IMODE:   word;           (* tryb pracy funkcji           *)
    LC0_HWND:    HWND;          (* parametry dla funkcji        *)
    LC0_HTASK:   THandle;       (* PostMessage i PostAppMessage *)
    LC0_UMSG:    word;           (*                               *)
    LC0_WPARAM:  word;           (*                               *)
    LC0_LPARAM:  longint;        (*                               *)
  end;
  (* bity LC0_IMODE *)
const
LC0_IM_INST      = $1;          (* instalacja przerwania        *)
LC0_IM_UNINST   = 0;           (* deinstalacja przerwania     *)
LC0_IM_CALL     = $2;          (* wywołanie funkcji klienta    *)
LC0_IM_POST     = 0;           (* wysłanie komunikatu          *)
LC0_IM_PAMES    = $4;          (* ma być wywołana funkcja PostAppMessage(); *)
LC0_IM_PMES     = 0;           (* ma być wywołana funkcja PostMessage() *)
LC0_IM_IREN     = $8;          (* wykonanie INTERRUPT_RESET przez ISR *)
LC0_IM_IRDIS    = 0;          (* INTERRUPT_RESET musi wywołać klient *)

```

(* Rekord opisu zlecenia funkcji INTERRUPT_RESET ===== *)

```
type
lc0_int_reset =
  record
    LC0_CODE:    byte;          (* numer funkcji (30)          *)
    LC0_STATUS: shortint;      (* kod odpowiedzi driver'a     *)
    LC0_IRMODULE: byte;        (* numer modulu                 *)
  end;
```

(* Rekord opisu zlecenia funkcji READ_STATUS ===== *)

```
type
lc0_read_status =
  record
    LC0_CODE:    byte;          (* numer funkcji (31)          *)
    LC0_STATUS: shortint;      (* kod odpowiedzi driver'a     *)
    LC0_SMODULE: byte;        (* numer modulu                 *)
    LC0_SDATA:   byte;          (* odczytany status            *)
    LC0_SRESRVD: byte;        (* pole zarezerwowane          *)
  end;
```

const (* bity statusu (READ_STATUS) *)

```
LC0_INT1_INTERRUPTED = 1;      (* linia INT 1 zglosila przerwanie *)
LC0_INT2_INTERRUPTED = 2;      (* linia INT 2 zglosila przerwanie *)
LC0_INT3_INTERRUPTED = 4;      (* linia INT 3 zglosila przerwanie *)
LC0_INT4_INTERRUPTED = 8;      (* linia INT 4 zglosila przerwanie *)
LC0_COMMON_INTERRUPT = 16;     (*jedna z linii zglosila przerwanie*)
LC0_CTC_OUT0         = 32;     (* stan linii OUT kanalu 0        *)
LC1_CTC_OUT1         = 64;     (* stan linii OUT kanalu 1        *)
LC0_CTC_OUT2         = 128;    (* stan linii OUT kanalu 2        *)
```

(* Rekord opisu zlecenia funkcji LEAVE_DRIVER_55 ===== *)

```
type
lc0_leave_55 =
  record
    LC0_CODE:    byte;          (* numer funkcji (32)          *)
    LC0_STATUS: shortint;      (* kod odpowiedzi driver'a     *)
  end;
```

const

(* ***** kody funkcji driver'a ***** *)

```
MODULE_INIT_55          = 16;    (* inicjalizacja modulow        *)
GET_TOTAL_CONFIGURATION_55 = 17;  (* odczyt konfig. modulow       *)
GET_MODULE_CONFIGURATION_55 = 18;  (* odczyt konfig. modulu        *)
PORT_DIRECTION         = 19;    (* ustawienie kierunku          *)
                        (* pracy portow                 *)
PORT_WRITE             = 20;    (* zapis do portu               *)
PORT_READ              = 21;    (* odczyt portu                 *)
PORT_LATCH            = 22;    (* zatrzasniecie danych         *)
PORT_RESET            = 23;    (* zerowanie ukladow 8255       *)
                        (* (ustawienie wszystkich      *)
                        (* portow w kierunku IN)       *)
CTC_WRITE_55          = 24;    (* zaprogramowanie i/lub        *)
                        (* zapisanie wartosci do        *)
                        (* kanalu CTC                   *)
CTC_GATE              = 25;    (* bramkowanie kanalu CTC       *)
CTC_READ_55           = 26;    (* odczyt kanalu CTC           *)
```

```

INTERRUPT_MASK          = 27;          (* maskowanie przerw          *)
INTERRUPT_LEVEL         = 28;          (* ustawianie poziomu        *)
                          (* przerywajacego            *)
INTERRUPT_SERVICE_55    = 29;          (* deklarowanie procedury     *)
                          (* obsługi przerwania       *)
INTERRUPT_RESET         = 30;          (* gaszenie zgłoszenia        *)
                          (* przerwania                *)
READ_STATUS             = 31;          (* odczyt statusu             *)
LEAVE_DRIVER_55         = 32;          (* zakończenie pracy z      *)
                          (* driver'em                  *)

```

(***** kody odpowiedzi funkcji driver'a *****)

```

LC0_OK                   = 0;          (* OK                          *)
(* ===== ostrzezenia ===== *)

LC0_NON_EX_MOD           = 1;          (* nie istniejący(e) moduł(y) *)
LC0_NO_LATCH_WARN       = 2;          (* moduł nie ma możliwości   *)
                          (* zatrzymywania danych      *)
LC0_IS_INIT              = 5;          (* zadano inicjacji modułów  *)

```

(* ===== błędy ===== *)

```

LC0_UNKN_FUNC           = -1;          (* niezany kod funkcji        *)
LC0_NO_MODULE           = -2;          (* brak modułu(ów)            *)
LC0_NOT_OUTPORT         = -3;          (* port nie jest wyjściowy    *)
LC0_NONEX_DEV           = -4;          (* nie istnieje urządzenie o *)
                          (* numerze                    *)
LC0_NOT_INPORT          = -5;          (* port nie jest wejściowy    *)
LC0_BAD_COUNTER         = -6;          (* zły numer kanału CTC       *)
LC0_NO_LATCH            = -7;          (* dany moduł nie może zatrzym *)
LC0_DEV_BUSY            = -11;         (* urządzenie jest zajęte    *)
LC0_BAD_PROC            = -19;         (* błędny adres procedury obsługi *)
                          (* przerw                    *)
LC0_NOT_SUPPORTED       = -24;         (* dla danego modułu funkcja nie *)
                          (* jest realizowana           *)
LC0_BAD_CTC_MODE        = -25;         (* błędny tryb pracy CTC     *)
LC0_NO_IRQ              = -29;         (* z danym modulem nie jest  *)
                          (* związane żadne przerwanie *)
LC0_NOT_FULLY_SUP       = -30;         (* zadany tryb funkcji nie   *)
                          (* jest realizowany dla danego *)
                          (* typu modułu lub funkcja w *)
                          (* opracowaniu                *)
LC0_INTR_INST           = -33;         (* próba powtórnej instalacji *)
                          (* przerw                    *)
LC0_CTC_NOT_PROGRAMMED = -37;         (* zapis wartości do         *)
                          (* niezaprogramowanego licznika *)
LC0_REJECTED            = -38;         (* za dużo jednoczesnych odwo *)
                          (* łan do driver'a           *)
LC0_BAD_CONFIG          = -39;         (* błąd w pliku konfiguracyjnym *)
LC0_NOT_INIT            = -40;         (* nie zainicjowany moduł    *)
LC0_IRQ_BUSY            = -41;         (* inny moduł obsługuje dane *)
                          (* przerwanie sprzętowe driver'a *)

```

(***** kody numerów modułów *****)

```

LC0_MODA                = 1;          (* moduł A                    *)
LC0_MODB                = 2;          (* moduł B                    *)
LC0_MODC                = 3;          (* moduł C                    *)
LC0_MODD                = 4;          (* moduł D                    *)
(* moduły dodatkowe *)
LC0_MODE                = 5;          (* moduł E                    *)
LC0_MODF                = 6;          (* moduł F                    *)

```

```
LC0_MODG      = 7;          (* modul G          *)
LC0_MODH      = 8;          (* modul H          *)
```

```
(***** maski modulow w mapie *****)
```

```
LC0_MODAMAP   = 1;          (* modul A          *)
LC0_MODBMAP   = 2;          (* modul B          *)
LC0_MODCMAP   = 4;          (* modul C          *)
LC0_MODDMAP   = 8;          (* modul D          *)
(* moduly dodatkowe *)
LC0_MODEMAP   = 16;         (* modul E          *)
LC0_MODFMAP   = 32;         (* modul F          *)
LC0_MODGMAP   = 64;         (* modul G          *)
LC0_MODHMAP   = 128;        (* modul H          *)
```

```
(* definicje typow do struktur danych *)
```

```
type
```

```
Plc0_init_55      = ^lc0_init_55 ;
Plc0_total_55     = ^lc0_total_55 ;
Plc0_module_55    = ^lc0_module_55 ;
Plc0_port_dir     = ^lc0_port_dir ;
Plc0_port_write   = ^lc0_port_write ;
Plc0_port_read    = ^lc0_port_read ;
Plc0_port_latch   = ^lc0_port_latch ;
Plc0_port_reset   = ^lc0_port_reset ;
Plc0_ctc_write_55 = ^lc0_ctc_write_55 ;
Plc0_ctc_gate     = ^lc0_ctc_gate ;
Plc0_ctc_read_55  = ^lc0_ctc_read_55 ;
Plc0_int_mask     = ^lc0_int_mask ;
Plc0_int_level    = ^lc0_int_level ;
Plc0_int_service  = ^lc0_int_service ;
Plc0_int_reset    = ^lc0_int_reset ;
Plc0_read_status  = ^lc0_read_status ;
Plc0_leave_55     = ^lc0_leave_55 ;
```

```
(*****)
```

```
(* Funkcje biblioteki LC055A.DLL *)
```

```
*****)
```

```
(*
```

```
PROCEDURE LC055_ModuleInit(param : Plc0_init_55 )
PROCEDURE LC055_GetTotalConfiguration(param : Plc0_total_55 )
PROCEDURE LC055_GetModuleConfiguration(param : Plc0_module_55 )
PROCEDURE LC055_PortDirection(param : Plc0_port_dir )
PROCEDURE LC055_PortWrite(param : Plc0_port_write )
PROCEDURE LC055_PortRead(param : Plc0_port_read )
PROCEDURE LC055_PortLatch(param : Plc0_port_latch )
PROCEDURE LC055_PortReset(param : Plc0_port_reset )
PROCEDURE LC055_CTCWrite(param : Plc0_ctc_write_55 )
PROCEDURE LC055_CTCGate(param : Plc0_ctc_gate )
PROCEDURE LC055_CTCRead(param : Plc0_ctc_read_55 )
PROCEDURE LC055_InterruptMask(param : Plc0_int_mask )
PROCEDURE LC055_InterruptLevel(param : Plc0_int_level )
PROCEDURE LC055_InterruptService(param : Plc0_int_service )
PROCEDURE LC055_InterruptReset(param : Plc0_int_reset )
PROCEDURE LC055_ReadStatus(param : Plc0_read_status )
PROCEDURE LC055_LeaveDriver(param : Plc0_leave_55 )
```

```
*)
```

DODATEK D
do dokumentacji biblioteki DLL
modułów cyfrowych
LC-055-PIO i LC-055-DCU

WSMP055.PAS
program przykładowy w Pascalu

(* WSMP055.PAS *****)

Program przykładowy pokazujący sposób wykorzystania drivera modułów cyfrowych
serii LC-055-PIO i LC-055-DCU firmy AMBEX.

Sposób dołączenia biblioteki DLL : statycznie przez .TPW *)

(*****)

Program korzysta ze standardowych funkcji API Windows 3.1 .

W katalogu zawierającym standardowe nagłówki PAS muszą się znajdować pliki:

AMBEX-55.PAS - plik nagłówkowy drivera,

LC055U.TPW - modul biblioteki LC055A.DLL

WSMP055.RES - plik zasobów ,

Pliki :

- AMBEX.INI - plik konfiguracyjny modułów serii LC,

- LC055A.DLL - driver (biblioteka DLL) do kart LC-055-PIO i LC-055-DCU,

powinny znajdować się w jednym z katalogów ze standardowej ścieżki przeszukiwania Windows (według kolejności przeszukiwania) :

1. katalog systemowy WINDOWS,

2. katalog WINDOWS\SYSTEM,

3. bieżący katalog aplikacji,

4. katalog znajdujący się na ścieżce przeszukiwania PATH .

Opcje kompilatora (Borland Pascal)

- Force far calls : on,

- Word align data : off,

- smart callbacks : on,

- windows stack frame : on,

UWAGA :

Procedura obsługi przerwania sprzętowego karty nie powinna bezpośrednio odwoływać się do zmiennych globalnych i powinna znajdować się w segmencie kodu oznaczonym jako FIXED . Wszystkie zmienne globalne modyfikowane przez tą procedurę powinny znajdować się również w segmencie danych oznaczonym jako FIXED . Jedynymi zalecanymi przez MICROSOFT funkcjami WINDOWS API, które mogą być wywołane bezpiecznie przez procedurę obsługi przerwania są :

PostMessage i PostAppMessage .

Powyższe ograniczenia nie obowiązują, gdy do powiadomieniu o wystąpieniu przerwania używa się funkcji obsługi komunikatu wysłanego przez driver .

*)

(*****)

```
program wsmp055;
```

```
{ $R Wsmp055 }
```

```
uses Win31, WinProcs, WinTypes, Strings, lc055u;
```

```
{ $i ambex-55.pas }
```

```
type IRQRequest =
```

```
record
```

```
    counter: integer;      (* licznik przerwania      *)
```

```
    itype:    integer;    (* typ zgłoszenia        *)
```

```
    hWindow:  HWND;      (* descr. okna dla PostMessage *)
```

```
    modnum:   integer;   (* numer modułu          *)
```

```
end;
```

```
type PIRQRequest = ^IRQRequest;
```

```
var
```

```
    s_init:    lc0_init_55;
```

```
    s_total:   lc0_total_55;
```

```
    s_module:  lc0_module_55;
```

```
    s_port_dir: lc0_port_dir;
```

```
    s_port_write: lc0_port_write;
```

```
    s_port_read: lc0_port_read;
```

```

s_ctc_write: lc0_ctc_write_55;
s_ctc_gate: lc0_ctc_gate;
s_int_mask: lc0_int_mask;
s_int_level: lc0_int_level;
s_int_service: lc0_int_service;
s_int_reset: lc0_int_reset;
s_leave: lc0_leave_55;
modulenum: integer;          (* numer badanego modulu *)
paintnum: integer;
paintdet: integer;
enddet: integer;
vhWnd: HWnd;
koniec: integer;
HDLL: THandle;
HInst: THandle;
fproc: TFarProc;
ISRData: IRQRequest;        (* parametry funkcji obslugi przerwania *)
ReadVal: array[0..10] of byte;
ReadCount: integer;
top: integer;

const
  BUFLen = 320;              (* dlugosc bufora *)
  SAMPLES = 20;             (* liczba probek *)
  CHANNELS = 8;             (* liczba kanalow *)
  MY_MESSAGE = WM_USER + 1;
  (* komunikat powiadomienia o wystapieniu przerwania sprzetowego *)
  MY_ISR_MESSAGE = WM_USER + 2;

(*****
(* Przeznaczenie: *)
(* Inicjalizacja zmiennych. *)
(*****)
procedure initprogram;
begin
  s_init.LC0_CODE := MODULE_INIT_55;
  s_total.LC0_CODE := GET_TOTAL_CONFIGURATION_55;
  s_module.LC0_CODE := GET_MODULE_CONFIGURATION_55;
  s_port_dir.LC0_CODE := PORT_DIRECTION;
  s_port_write.LC0_CODE := PORT_WRITE;
  s_port_read.LC0_CODE := PORT_READ;
  s_ctc_write.LC0_CODE := CTC_WRITE_55;
  s_ctc_gate.LC0_CODE := CTC_GATE;
  s_int_mask.LC0_CODE := INTERRUPT_MASK;
  s_int_level.LC0_CODE := INTERRUPT_LEVEL;
  s_int_service.LC0_CODE := INTERRUPT_SERVICE_55;
  s_int_reset.LC0_CODE := INTERRUPT_RESET;
  s_leave.LC0_CODE := LEAVE_DRIVER_55;
  paintnum := 0;
  paintdet := 0;
  enddet := 0;
  koniec := 0;
  ReadCount := 0;
  top := 0;
end;

(*****
Funkcja quit()
Przeznaczenie : Wykonanie funkcji LEAVE_DRIVER_55

```

```

*****
PROCEDURE quit;
begin
  LC055_LeaveDriver(@s_leave);
end ;

(*****
(* Przeznaczenie: *)
(* Rozpoznanie konfiguracji badanego modulu. *)
(* Sposob: *)
(* Przez wykorzystanie funkcji driver'a GET_TOTAL_CONFIGURATION_55, *)
(* GET_MODULE_CONFIGURATION_55,MODULE_INIT_55 *)
(* Uwagi: *)
(* Funkcja nadaje wartosc zmiennej modulenum (numer badanego modulu) *)
(* wypelnia struktury total, module, info i inicjalizuje zainstalowane *)
(* moduly. *)
(*****
procedure askdriver;
label lab1;
begin
  LC055_GetTotalConfiguration(@s_total);
  (* GET_TOTAL_CONFIGURATION_55 *)
  (* inicjalizacja zainstalowanych modutow *)
  s_init.LC0_IMODULE := s_total.LC0_TONF and $F;
  LC055_ModuleInit(@s_init); (* MODULE_INIT_55 *)
  (* sprawdzenie, ktory modul jest *)
  (* zainstalowany: A, B, C czy .. H *)
  for modulenum := 1 to 8 do
    if ((s_total.LC0_TONF and (1 shl (modulenum - 1))) <> 0) then
      goto lab1;
lab1:
  (* spytanie o konfiguracje modulu *)
  s_module.LC0_MMODULE := modulenum;
  LC055_GetModuleConfiguration(@s_module);
  (* GET_MODULE_CONFIGURATION_55*)
  ISRData.modnum := modulenum;
  ISRData.counter :=0;
  ISRData.hWindow :=vhWnd;
end;

(*****
(* GoToLine(*RECT) 0 przejście do następnej linii przy wyświetlaniu tekstu *)
(*****
procedure GoToLine(var rect : TRECT ; dc :HDC) ;
var
  tm :TTEXTMETRIC ;
begin
  GetTextMetrics(dc,tm);
  rect.top := rect.top + tm.tmHeight + tm.tmExternalLeading ;
  rect.bottom := rect.bottom + tm.tmHeight + tm.tmExternalLeading ;
end ;

(*****
(*Funkcja : drawscreen *)
(* Przeznaczenie : Wybranie odpowiedniej sekcji informacji do wyświetlenia *)
(* na ekranie w odpowiedzi na komunikat WM_PAINT *)
(* Parametr : przepisuje do zmiennej globalnej paintnum *)
(*****

```

```

procedure drawscreen;
begin
  InvalidateRect(vhWnd,nil,TRUE);
  UpdateWindow(vhWnd);
end ;

(*****)
(* Funkcja PressOrClickAny(HDC,RECT) *)
(* Wypisuje na ekranie informacje o końcu kolejnego kroku programu *)
(*****)
procedure PressOrClickAny(dc : HDC ;rect : TRECT);
var
  line: array[0..100] of char ;
begin
  GoToLine(rect,dc);GoToLine(rect,dc);
  StrPCopy(line,'Wciśnij dowolny klawisz lub kliknij by przejść dalej ... ');
  DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
end;

(*****)
(* Przeznaczenie: *)
(* Zamiana cyfry szesnastkowej na jej reprezentacje znakowa. *)
(* Parametry: *)
(* dig - cyfra do zamiany *)
(* Wartosc: *)
(* Reprezentacja znakowa cyfry. *)
(*****)
function hexdigit(dig: word) : char;
begin
  if dig < 10 then hexdigit := chr(ord('0') + dig)
  else hexdigit := chr(ord('A') + dig - 10);
end;

(*****)
(* Przeznaczenie: *)
(* Wydrukowanie liczby calkowitej w postaci szesnastkowej. *)
(* Parametry: *)
(* val - wartosc do wydrukowania *)
(*****)
function writehex(val: word) :String ;
var s : string[4] ;
begin
  s:=hexdigit((val and $F000) shr 12)+hexdigit((val and $F00) shr 8)+
  hexdigit((val and $F0) shr 4)+hexdigit(val and $F);
  writehex :=s;
end;

(*****)
(* Przeznaczenie: *)
(* Wyszwietlenie konfiguracji badanego modulu *)
(* Sposob: *)
(* Przez wykorzystanie informacji zawartych w strukturach module i info. *)
(*****)
procedure displayconfiguration(dc:HDC ; var rect : TRECT);
var
  i: integer;
  line : Array[0..100]of char ;
  spar :string[255] ;
  sp1 : string[20];

```

```

begin
  (* konfiguracja modulu *)
  rect.left := rect.left + 10 ;
  DrawText(dc,'Konfiguracja modulu:',-1,rect,DT_LEFT or DT_SINGLELINE);
  GoToLine(rect,dc); GoToLine(rect,dc);
  case s_module.LC0_MTYPE of
    LC_055_PIO : spar :='LC-055-PIO';
    LC_055_DCU : spar :='LC-055-DCU';
  end;
  StrPCopy(line,'Moduł '+chr(ord('A')+modulenum-1) +' : ' +spar);
  DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
  GoToLine(rect,dc);
  StrPCopy(line,'Adres modułu: '+ writehex(s_module.LC0_MBASE)+ '(hex)');
  DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
  GoToLine(rect,dc);
  str(s_module.LC0_MPORT,spar) ;
  StrPCopy(line,'Liczba portów we/wy: ' + spar);
  DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
  GoToLine(rect,dc);
  str(s_module.LC0_MCTC,spar);
  StrPCopy(line,'Liczba kanałów CTC: ' + spar);
  DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
  GoToLine(rect,dc);
  if s_module.LC0_MCTC <>0 then
    begin
      if s_module.LC0_MTYPE = LC_055_PIO then
        begin
          if s_module.LC0_MCLOCK[1] = 0 then
            StrPCopy(line,'Wszystkie kanały CTC podłączone do zegara zewnętrznego')
          else
            begin
              str(s_module.LC0_MCLOCK[1],spar);
              StrPCopy(line,'Częstotliwość zegara CTC: '+spar+' kHz');
            end;
            DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
            GoToLine(rect,dc);
          end
        else
          begin
            for i:=1 to s_module.LC0_MCTC do
              begin
                if s_module.LC0_MCLOCK[i] = 0 then
                  StrPCopy(line,'Wszystkie kanał '+chr(ord('0')+i-1)+ ' podłączony do zegara zewnętrznego')
                else
                  begin
                    str(s_module.LC0_MCLOCK[i],spar);
                    StrPCopy(line,'Częstotliwość zegara kanału '+chr(ord('0')+i-1)+ ' ' + spar+' kHz');
                  end;
                  DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
                  GoToLine(rect,dc);
                end;
              end;
            end;
          end;
        if s_module.LC0_MINT = $FF then
          StrPCopy(line,'Moduł nie jest podłączony do żadnego przerwania')
        else
          begin
            str(s_module.LC0_MINT,spar);
            StrPCopy(line,'Numer przerwania sprzętowego związanego z modułem: IRQ '+spar);
          end;
        end;
      end;
    end;
  end;
end;

```

```

end;
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
GoToLine(rect,dc);
ReleaseDC(dc,vhWnd);
end;

(*****)
(* Funkcja : displaytext(HDC,RECT,unsigned char) *)
(* Przeznaczenie: *)
(* Wświetlenie komunikatów tekstowych *)
(*****)
function displaytext(dc : HDC ;rect : TRECT ):integer ;
var
line : array[0..200] of char;
spr : string[200];
i: integer;
begin
rect.left := rect.left + 10;
case paintnum of
2 :begin
(* readport *)
StrPCopy(line,'Przesłanie 10 bajtów danych z portu wejściowego na wyjściowy:');
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
GoToLine(rect,dc);
StrPCopy(line,'Port A (1) - wejście ; Port B (2) - wyjście ; pozostałe bez zmian');
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
end;
3 :begin
(* writefail *)
StrPCopy(line,'Przesyłanie danych do portu wejściowego (błąd!):');
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
GoToLine(rect,dc);
StrPCopy(line,'Port A (1) - wejście ; pozostałe bez zmian');
GoToLine(rect,dc);
end ;
4 :begin
(* testinterrupts-funkcja obsługi *)
StrPCopy(line,'Testowanie przerwanych przychodzących z modułu : obsługa bezpośrednia poprzez funkcję ');
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
GoToLine(rect,dc);
StrPCopy(line,'Wciśnij dowolny klawisz lub kliknij by przerwać ...');
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
end ;
5 :begin
(* testinterrupts - komunikat *)
StrPCopy(line,'Testowanie przerwanych przychodzących z modułu : obsługa komunikatu wygenerowanego przez driver');
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
GoToLine(rect,dc);
StrPCopy(line,'Wciśnij dowolny klawisz lub kliknij by przerwać ...');
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
end;
end ;
GoToLine(rect,dc);
(* rozpoznanie źródeł przerwanych na poszczególnych liniach dla modułu LC-055-DCU*)
if((paintnum=4) or (paintnum=5))then
begin
if s_module.LC0_MTYPE = LC_055_DCU then
for i:=1 to LC0_MAX_INT_LINES do
begin

```

```

case ((s_module.LC0_MINTSRC shr (2 * (i-1))) and 3) of
  LC0_NO_INT   :StrPCopy(line,'Linia '+ chr(ord('0')+i)+' : niepodłączona') ;
  LC0_INTSRC_EXT :StrPCopy(line,'Linia '+ chr(ord('0')+i)+' : przerwanie zewnętrzne') ;
  LC0_INTSRC_CTC :StrPCopy(line,'Linia '+ chr(ord('0')+i)+' : przerwanie od CTC') ;
end;
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
GoToLine(rect,dc);
end;
end;
displaytext := rect.top;
end ;

```

(*****)

Funkcja : displayinterrupts(HDC,RECT,unsigned char)

Przeznaczenie:

Wyswietlenie licznika zgłoszeń przerwania sprzętowych

(*****)

```
function displayinterrupts(dc : HDC;rect :TRECT):integer;
```

```
var
```

```
line : array[0..200] of char;
```

```
spar: string[100] ;
```

```
st1 : string[10];
```

```
begin
```

```
rect.left := rect.left + 10;
```

```
if((paintnum=4) or (paintnum=5)) then
```

```
begin
```

```
case ISRData.itype of
```

```
0 :spar:='Brak zgłoszeń';
```

```
1 :spar:='Obsługa komunikatu wysłanego przez driver';
```

```
2 :spar:='Obsługa bezpośrednia - funkcja ';
```

```
end;
```

```
str(ISRData.counter,st1);
```

```
StrPCopy(line,'Ilość zgłoszeń przerwania : '+st1 +' typ obsługi : '+spar);
```

```
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
```

```
GoToLine(rect,dc);
```

```
end;
```

```
displayinterrupts:=rect.top;
```

```
end ;
```

(*****)

Funkcja : displayresults(HDC,RECT,unsigned char)

Przeznaczenie:

Wyswietlenie danych odczytanych z portu A

(*****)

```
function displayresults(dc:HDC;rect:TRECT):integer;
```

```
var
```

```
line : array[0..200]of char;
```

```
spar: string[100] ;
```

```
sp1 : string[10];
```

```
(*unsigned char maska[]={1,2,4,8,16,32,64,128};*)
```

```
i,j: integer;
```

```
begin
```

```
rect.left :=rect.left + 10;
```

```
if paintnum=2 then
```

```
begin
```

```
StrPCopy(line,'Port A - bit 8 ... 1') ;
```

```
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
```

```
GoToLine(rect,dc);
```

```
for i:=0 to ReadCount-1 do
```

```

begin
if i<>10 then
begin
spar:="";
for j:=0 to 7 do
if ( ReadVal[i] and (1 shl j))<>0 then
spar:='1'+spar
else
spar:='0'+spar;
str(i+1,spl);
StrPCopy(line,' '+spl+' '+spar);
DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
GoToLine(rect,dc);
end;
end;
end;
displayresults:=rect.top;
end;

(*****
(* Przeznaczenie: *)
(* Wypisanie komunikatow o bledzie / ostrzezeniu / dodatkowej informacji *)
(* bledzie. *)
(* Parametry: *)
(* status - LC0_STATUS *)
(* err_stat - LC0_ERR_STAT *)
(*****)
procedure drivererror(status: shortint);
var
line : array [0..200] of char ;
spr : string[200];
begin
str(status,spr);
if status > 0 then
StrPCopy(line,'Ostrzezenie (kod): '+ spr)
else
StrPCopy(line,'Bład (kod): '+ spr);
MessageBox(vhWnd,line,'Sygnał',MB_OK);
end;

(*****
Przeznaczenie:
Odczyt portu 1 (A) zaprogramowanego jako wejsciowy i wyslanie
odczytanej wartosci na port 2 (B) zaprogramowany jako wyjsciowy.
(*****)
procedure readport;
begin
s_port_dir.LC0_PMODULE := modulenum;
s_port_dir.LC0_PDIR := (LC0_IN_DIRECTION or LC0_CHANGE_DIR) or
((LC0_OUT_DIRECTION or LC0_CHANGE_DIR) shl 2);
(* PORT_DIRECTION *)
LC055_PortDirection(@s_port_dir);
if s_port_dir.LC0_STATUS <> LC0_OK then
begin
enddet:=1;
drivererror(s_port_dir.LC0_STATUS);
drawscreen;
Exit;
end;

```

```

        (* przygotowanie stałych pol      *)
        (* rekordów opisów zleceń        *)
s_port_read.LC0_PMODULE := modulenum;
s_port_read.LC0_PRPORT := 1;
s_port_read.LC0_PRLATCH := LC0_LATCH_OFF;
s_port_write.LC0_PMODULE := modulenum;
s_port_write.LC0_PWPORT := 2;
        (*ustawienie timera na 1s*)
SetTimer(vhWnd,1,1000,nil);
end;

(*****
Procedura wywoływana przez timer co 1s : odczyt danych z portu A i zapis na
port B
*****)
procedure PortReadWrite;
begin
    (* PORT_READ *)
    LC055_PortRead(@s_port_read);
    if s_port_read.LC0_STATUS <> LC0_OK then
        begin
            KillTimer(vhWnd,1);
            ReadCount:=0;
            enddet:=1;
            drivererror(s_port_read.LC0_STATUS);
            drawscreen;
            Exit;
        end;
    ReadVal[ReadCount]:= s_port_read.LC0_PRDATA;
    s_port_write.LC0_PWDATA := s_port_read.LC0_PRDATA;
    (* PORT_WRITE *)
    LC055_PortWrite(@s_port_write);
    if s_port_write.LC0_STATUS <> LC0_OK then
        begin
            KillTimer(vhWnd,1);
            ReadCount:=0 ;
            enddet:=1;
            drivererror(s_port_write.LC0_STATUS);
            drawscreen;
            Exit;
        end;
    ReadCount:=ReadCount+1;
    (* gdy koniec *)
    if ReadCount = 11 then
        begin
            KillTimer(vhWnd,1);
            enddet:=1;
            end;
            drawscreen;
        end;
end;

(*****
Przeznaczenie:
Zapis na port zaprogramowany jako wejściowy - błąd.
*****)
procedure writeportfail;
begin
    s_port_dir.LC0_PMODULE := modulenum;
    (* port A - wejściowy *)

```

```

s_port_dir.LC0_PDIR := (LC0_IN_DIRECTION or LC0_CHANGE_DIR);
(* PORT_DIRECTION *)
LC055_PortDirection(@s_port_dir);
if s_port_dir.LC0_STATUS <> LC0_OK then
begin
enddet:=1;
drivererror(s_port_dir.LC0_STATUS);
drawscreen;
Exit;
end;
s_port_write.LC0_PWMODULE := modulenum;
s_port_write.LC0_PWPORT := 1;
s_port_write.LC0_PWDATA := 0;
(* PORT_WRITE *)
LC055_PortWrite(@s_port_write);
if s_port_write.LC0_STATUS <> LC0_OK then
begin
enddet:=1;
drivererror(s_port_write.LC0_STATUS);
drawscreen;
end;
end;

```

(*****)

Procedura obsługi przerwania wołana bezpośrednio przez driver
 ptr - wskaźnik do struktury danych . Funkcja wywoływana jest przez driver
 w czasie obsługi przerwania sprzętowego : parametr ptr = LC0_DATAPTR podczas
 rejestracji funkcji (INTERRUPT_SERVICE) .

UWAGA :

Procedura obsługi przerwania sprzętowego karty nie powinna bezpośrednio
 odwoływać się do zmiennych globalnych i powinna znajdować się w segmencie
 kodu oznaczonym jako FIXED . Wszystkie zmienne globalne modyfikowane przez tą
 procedurę powinny znajdować się również w segmencie danych oznaczonym jako
 FIXED . Jedynymi zalecanymi przez MICROSOFT funkcjami WINDOWS API, które
 mogą być wywołane bezpiecznie przez procedurę obsługi przerwania są :

PostMessage i PostAppMessage .

(*****)

```

procedure newintPIODCU(var ptr : IRQRequest );

```

```

var

```

```

  SIntRes :lc0_int_reset;

```

```

begin

```

```

    (* licznik przerwania *)

```

```

  ptr.counter :=ptr.counter+1;

```

```

    (* typ wywołania : 2 == funkcja *)

```

```

  ptr.itype:=2 ;

```

(*****)

można tutaj również wywołać funkcję driver'a np. INERRUPT_RESET

(*****)

```

  SIntRes.LC0_IRMODULE:=ptr.modnum;

```

```

  LC055_InterruptReset(@SIntRes);

```

```

    (* odrysowanie okna z nowymi informacjami *)

```

```

    (* bezpieczna funkcja API !!! WPARAM==2 -> tzn. funkcja*)

```

```

  PostMessage(ptr.hWindow,MY_ISR_MESSAGE,2,0);

```

```

end;

```

(*****)

Przeznaczenie:

Testowanie przerwania z modułu LC-055-PIO.

Sposob:

Przez wywołanie funkcji INTERRUPT_SERVICE_55
nr == 4 - ustawienie obsługi poprzez wywołanie funkcji
nr == 5 - ----- rejestrację komunikatu

Przerwanie musi być wygenerowane z zewnątrz

*****)

```
procedure testPIOint(nr:integer);
```

```
begin
```

```
    (* instalacja obsługi przerwania *)
```

```
s_int_service.LC0_ISMODULE := modulenum;
```

```
if nr= 4 then (*funkcja obsługi*)
```

```
begin
```

```
    (* rejestracja funkcji obsługi przerwania *)
```

```
s_int_service.LC0_IPROC := @newintPIODCU;
```

```
    (* wskaźnik do struktury danych z parametrami dla funkcji obsługi przerwania *)
```

```
s_int_service.LC0_DATAPTR := @ISRData;
```

```
    (* tryb pracy : instalacja funkcji klienta i wywołanie INTERRUPT_RESET*)
```

```
s_int_service.LC0_IMODE:=LC0_IM_INST or LC0_IM_CALL or LC0_IM_IREN ;
```

```
end
```

```
else (* komunikat driver'a *)
```

```
begin
```

```
    (* tryb pracy : instalacja komunikatu PostMessage, wywołanie INTERRUPT_RESET *)
```

```
s_int_service.LC0_IMODE:=LC0_IM_INST or LC0_IM_POST or LC0_IM_PME$ or LC0_IM_IREN ;
```

```
    (* parametry komunikatu wstawionego przez driver *)
```

```
s_int_service.LC0_HWND := vhWnd;
```

```
s_int_service.LC0_HTASK := GetCurrentTask;
```

```
s_int_service.LC0_UMSG := MY_ISR_MESSAGE;
```

```
s_int_service.LC0_WPARAM :=1;
```

```
s_int_service.LC0_LPARAM:=0;
```

```
end;
```

```
    (* instalacja obsługi : INTERRUPT_SERVICE_55 *)
```

```
LC055_InterruptService(@s_int_service);
```

```
if s_int_service.LC0_STATUS <>LC0_OK then
```

```
begin
```

```
    enddet:=1;
```

```
    drivererror(s_int_service.LC0_STATUS);
```

```
    drawscreen;
```

```
    Exit;
```

```
end;
```

```
    (* zezwolenie na przerwanie : INTERRUPT_MASK*)
```

```
s_int_mask.LC0_IMMODULE := modulenum;
```

```
s_int_mask.LC0_IMASK := 1;
```

```
LC055_InterruptMask(@s_int_mask);
```

```
if s_int_mask.LC0_STATUS<>LC0_OK then
```

```
begin
```

```
    enddet:=1;
```

```
    drivererror(s_int_mask.LC0_STATUS);
```

```
    drawscreen;
```

```
    Exit;
```

```
end;
```

```
    (* przerwania są odblokowane *)
```

```
end;
```

*****)

Przeznaczenie:

Testowanie przerw z modulu LC-055-DCU.

Sposob:

Przez wywołanie funkcji INTERRUPT_SERVICE_55

nr == 4 - ustawienie obsługi poprzez wywołanie funkcji

nr == 5 - ----- rejestrację komunikatu

W przypadku linii przerywających połączonych z CTC program odpowiednio steruje CTC tak, aby osiągnąć generowanie przerw z częstotliwością 20 Hz. W pozostałych przypadkach przerwanie musi być oczywiście generowane z zewnątrz.

*****)

```
procedure testDCU(nr:integer);
```

```
begin
```

```
(* zaprogramowanie wszystkich kanałów CTC
```

```
na pracę w trybie fali prostokątnej o
```

```
częstotliwości 20 Hz
```

```
jeżeli którakolwiek z linii
```

```
przerywających jest przyłączona do CTC
```

```
to przerwanie będą generowane
```

```
samoczynnie *)
```

```
s_ctc_write.LC0_CMODULE := modulenum;
```

```
s_ctc_write.LC0_CMODE := LC0_SET_COUNTER_VALUE or LC0_SET CTC_MODE;
```

```
s_ctc_write.LC0_CCTRL := CTC_COUNT0 or CTC_NB or CTC_MODE3 or CTC_BOTH;
```

```
s_ctc_write.LC0_CWDATA := s_module.LC0_MCLK[1] * 50;
```

```
(* CTC_WRITE_55 *)
```

```
LC055_CTCWrite(@s_ctc_write);
```

```
s_ctc_write.LC0_CCTRL := CTC_COUNT1 or CTC_NB or CTC_MODE3 or CTC_BOTH;
```

```
s_ctc_write.LC0_CWDATA := s_module.LC0_MCLK[2] * 50;
```

```
(* CTC_WRITE_55 *)
```

```
LC055_CTCWrite(@s_ctc_write);
```

```
s_ctc_write.LC0_CCTRL := CTC_COUNT2 or CTC_NB or CTC_MODE3 or CTC_BOTH;
```

```
s_ctc_write.LC0_CWDATA := s_module.LC0_MCLK[3] * 50;
```

```
(* CTC_WRITE_55 *)
```

```
LC055_CTCWrite(@s_ctc_write);
```

```
if s_ctc_write.LC0_STATUS <> LC0_OK then
```

```
begin
```

```
enddet:=1;
```

```
drivererror(s_ctc_write.LC0_STATUS);
```

```
drawscreen;
```

```
Exit;
```

```
end;
```

```
(* zezwolenie na pracę wszystkich *)
```

```
(* kanałów CTC *)
```

```
s_ctc_gate.LC0_CGMODULE := modulenum;
```

```
s_ctc_gate.LC0_CGATE := 0;
```

```
(* CTC_GATE *)
```

```
LC055_CTCGate(@s_ctc_gate);
```

```
if s_ctc_gate.LC0_STATUS <> LC0_OK then
```

```
begin
```

```
enddet:=1;
```

```
drivererror(s_ctc_gate.LC0_STATUS);
```

```
drawscreen;
```

```
Exit;
```

```
end;
```

```
s_int_service.LC0_ISMODULE := modulenum;
```

```
if nr= 4 then (*funkcja obsługi*)
```

```
begin
```

```
(* rejestracja funkcji obsługi przerwania *)
```

```
s_int_service.LC0_IPROC := @newintPIODCU;
```

```
(* wskaźnik do struktury danych z parametrami dla funkcji obsługi przerwania *)
```

```
s_int_service.LC0_DATAPTR := @ISRData;
```

```
(* tryb pracy : instalacja funkcji klienta i wywołanie INTERRUPT_RESET*)
```

```

s_int_service.LC0_IMODE:=LC0_IM_INST or LC0_IM_CALL or LC0_IM_IREN ;
end
else (* komunikat driver'a *)
begin
  (* tryb pracy : instalacja komunikatu PostMessage, wywołanie INTERRUPT_RESET *)
s_int_service.LC0_IMODE:=LC0_IM_INST or LC0_IM_POST or LC0_IM_PMES or LC0_IM_IREN ;
  (* parametry komunikatu wstawionego przez driver *)
s_int_service.LC0_HWND := vhWnd;
s_int_service.LC0_HTASK := GetCurrentTask;
s_int_service.LC0_UMSG := MY_ISR_MESSAGE;
s_int_service.LC0_WPARAM :=1;
s_int_service.LC0_LPARAM:=0;
end;
  (* instalacja obsługi : INTERRUPT_SERVICE_55 *)
LC055_InterruptService(@s_int_service);
if s_int_service.LC0_STATUS <>LC0_OK then
begin
  enddet:=1;
  drivererror(s_int_service.LC0_STATUS);
  drawscreen;
  Exit;
end;
  (* zaprogramowanie aktywnych poziomow przerywajacych *)
s_int_level.LC0_ILMODULE := modulenum;
s_int_level.LC0_ILEVEL := 0; (* wszystkie aktywne poziomy - 0 *)
  (* INTERRUPT_LEVEL *)
LC055_InterruptLevel(@s_int_level);
if s_int_level.LC0_STATUS <> LC0_OK then
begin
  enddet:=1;
  drivererror(s_int_level.LC0_STATUS);
  drawscreen;
  Exit;
end;
  (* zezwolenie na przerwanie : INTERRUPT_MASK*)
s_int_mask.LC0_IMMODULE := modulenum;
s_int_mask.LC0_IMASK := $0f; (* zezwolenie na przerwania *)
LC055_InterruptMask(@s_int_mask);
if s_int_mask.LC0_STATUS <> LC0_OK then
begin
  enddet:=1;
  drivererror(s_int_mask.LC0_STATUS);
  drawscreen;
  Exit;
end;
  (* przerwania są odblokowane *)
end;

(*****
Przeznaczenie:
  Testowanie obsługi przerwanych przychodzących z modulu.
*****
)
procedure testinterrupts(nr:integer);
begin
  if s_module.LC0_MINT = $FF then
  begin
    MessageBox(vhWnd,'Z modulem nie jest związane zadne przerwanie!','Info',MB_OK);
    enddet:=1;
    drawscreen;

```

```

Exit;
end;
      (* w zaleznosci od typu modulu procedury *)
      (* testujace maja rozna postac *)
case s_module.LC0_MTYPE of
  LC_055_PIO : testPIOint(nr);
  LC_055_DCU : testDCUint(nr);
end;
end;

```

(*****)

Funkcja: MainWndProc(HWND, UINT, WPARAM, LPARAM)

Przeznaczenie: Funkcja odpowiedzi okna na komunikaty .

Komunikaty:

```

WM_COMMAND- Menu okna
WM_DESTROY- Zniszczenie okna i nastepnie powrot do Windows
WM_PAINT - Odmalowywanie okna
WM_LBUTTONDOWN - Wcisniecie lewego klawisza myszy
WM_KEYDOWN - Wcisniecie klawisza
MY_MESSAGE - wywołanie kolejnej procedury programu
MY_ISR_MESSAGE - komunikat informujacy o przerwaniu sprzetowym karty
WM_TIMER - timer

```

(*****)

```

function MainWndProc(vhWnd : HWND ; message,wParam : Word ; lParam : Longint):Longint ; export;
var
dc : HDC;
rect : TRECT;
ps : TPAINTSTRUCT ;
begin
case message of
  WM_COMMAND:      (* komenda menu *)
    case wParam of
      101 :          (* pomiary *)
        begin
          if paintnum = 0 then
            begin
              paintnum := 1;
              askdriver;
              drawscreen;
            end;
          end ;
        102 :          (* koniec *)
        begin
          DestroyWindow(vhWnd);
        end;
      else
        MainWndProc:=DefWindowProc(vhWnd, message, wParam, lParam);
      end ;
  WM_PAINT :      (* wyswietlenie informacji *)
    begin
      dc := BeginPaint(vhWnd,ps);
      GetClientRect(vhWnd,rect);
      case paintnum of
        1 :          (* konfiguracja *)

```

```

    displayconfiguration(dc,rect);
2,3,4,5,6 :
begin
    rect.top :=displaytext(dc,rect);
    rect.left:=10;
                                (* wyświetlenie rezultatów pomiarów *)
    rect.top:=displayresults(dc,rect);
                                (*wyświetlenie zawartości licznika przerwań*)
    rect.top:=displayinterrupts(dc,rect);
    top:=rect.top;
    if enddet > 0 then
        PressOrClickAny(dc,rect);
    end ;
end;
EndPaint(vhWnd,ps);
end ;
WM_LBUTTONDOWN,WM_KEYDOWN :      (* goto next*)
begin
enddet :=0;
if paintnum <> 0 then
begin
if((paintnum=4) or (paintnum=5)) then (* wyinstalowanie przerwań po zakończeniu testu*)
begin
    ISRData.counter:=0;
    ISRData.itype:=0;
                                (* zablokowanie kanałów CTC dla LC-055-DCU *)
    if s_module.LC0_MTYPE = LC_055_DCU then
begin
        s_ctc_gate.LC0_CGGATE := $07;
                                (* CTC_GATE *)
        LC055_CTCGate(@s_ctc_gate);
end;
                                (* zablokowanie przerwań *)
        s_int_mask.LC0_IMASK := 0;
        LC055_InterruptMask(@s_int_mask);
        if s_int_service.LC0_STATUS = LC0_OK then
begin
            s_int_service.LC0_IMODE:=0;
            LC055_InterruptService(@s_int_service);
end;
end;
paintnum := paintnum +1 ;
if paintnum = 6 then
begin
    paintnum := 0 ;
end
else
    PostMessage(vhWnd,MY_MESSAGE,0,0);
drawscreen;
end ;
end ;
MY_MESSAGE : (* wykonanie kolejnego kroku algorytmu *)
begin
enddet := 0;
case paintnum of
1:begin
    askdriver;          (* odpytanie driver-a o konfiguracje *)
    drawscreen;        (* wyświetlenie konfiguracji modulu *)
end;

```

```

2:readport;      (* odczyt portu cyfrowego *)
3:
  begin
  if ReadCount <> 11 then
    KillTimer(vhWnd,1);
    ReadCount:=0;
    writeportfail; (* zapis na port wejściowy - blad *)
  end;
4:testinterrupts(4); (*test przerwan przychodzacych z modulu - funkcja*)
5:testinterrupts(5); (*test przerwan przychodzacych z modulu - komunikat *)
end ;
end ;
WM_DESTROY: (* koniec programu *)
begin
  PostQuitMessage(0);
end ;
WM_TIMER:      (* odczyt portów cyfrowych*)
begin
  PortReadWrite;
end;
MY_ISR_MESSAGE: (* informacja o przerwaniu *)
begin
  if wParam = 1 then (* bezpośrednio wysłanie komunikatu z driver'a *)
    begin
      (* modyfikujemy zmienne *)
      ISRData.itype:=1;
      ISRData.counter:=ISRData.counter+1;
    end;
  if wParam =2 then (* wysłanie komunikatu z procedury obsługi przerwania *)
    begin
      (* nie musimy ustawiać zmiennych bo zrobiła to funkcja obsługi przerwania *)
    end;
    (* odmalowanie okien *)
  GetClientRect(vhWnd,rect);
  rect.top:=top-20;
  rect.bottom:=top;
  InvalidateRect(vhWnd,@rect,TRUE);
end;
else
  MainWndProc := DefWindowProc(vhWnd, Message, wParam, lParam);
end ;
end ;

```

(*****)

Procedura WinMain
Przeznaczenie: Inicjacja aplikacji, okna głównego

(*****)

```

procedure WinMain;
var
  msg : TMsg;
  WndClas : TWndClass ;
  dresult : BOOL ;
begin
  if hPrevInst = 0 then
    begin
      hInst := HInstance ;

```

```
WndClas.Style := 0;
WndClas.lpfWndProc:= @MainWndProc;
WndClas.cbClsExtra := 0;
WndClas.cbWndExtra := 0;
WndClas.hInstance := HInstance;
WndClas.hIcon := LoadIcon(HInstance,'ICON_1');
WndClas.hCursor := LoadCursor(0, IDC_Arrow);
WndClas.hbrBackground := GetStockObject(White_Brush);
WndClas.lpszMenuName := 'MENU_1';
WndClas.lpszClassName := 'DemoDriverWClass';
if not RegisterClass(WndClas) then
  Halt;
end ;
vhWnd := CreateWindow('DemoDriverWClass','Przykład użycia biblioteki DLL Windows do kart cyfrowych LC-055',
  WS_OverLappedWindow,CW_UseDefault,CW_UseDefault, CW_UseDefault,CW_UseDefault, 0, 0, hInstance,
nil);
if vhWnd = 0 then
  Halt;
UpdateWindow(vhWnd);
ShowWindow(vhWnd,Sw_ShowNormal);
MessageBox(vhWnd,'Zamknij by rozpocząć przykładową sekwencję pomiarową .','Informacja',MB_OK);
paintnum := 1;
PostMessage(vhWnd,MY_MESSAGE,0,0);
while GetMessage(Msg, 0, 0, 0) do
begin
  TranslateMessage(msg);
  DispatchMessage(msg);
end;
quit ;
end ;

begin
  WinMain;
end.
```

DODATEK E

do dokumentacji biblioteki DLL

modułów cyfrowych

LC-055-PIO i LC-055-DCU

Plik konfiguracyjny AMBEX.INI

Opis pliku konfiguracyjnego 'ambex.ini'

1. Struktura pliku:

[nazwa karty 1]
 parametr globalny 1
 parametr globalny 2

...
 ...

[nazwa modułu 1]
 parametr 1
 parametr 2
 parametr 3

...
 parametr 2

[nazwa modułu 2]
 parametr 1
 parametr 2

...

[nazwa karty 2]

...
 ...

2. Parametry dla kart LC-055-PIO i LC-055-DCU.

Wartości poszczególnych parametrów muszą odpowiadać ustawieniom zwór konfiguracyjnych i jumperów modułu (patrz instrukcja obsługi).

Uwaga ! Ważna jest kolejność parametrów. Postawienie na początku linii znaku ';' spowoduje pominięcie jej podczas analizy parametrów.

[LC-055] - nazwa karty: karty LC-055-PIO lub LC-055-DCU

Path = 'x' - 'x' = ścieżka do pliku "ambex.ini"

Modules = 'x' - liczba modułów zainstalowanych; 'x' = 1..8

[MODULE 'x'] - 'x' = *A,B,C,D,E,F,G,H*: nazwa (numer) modułu;

Type = 'x' - 'x' = *PIO*: moduł LC-055-PIO; 'x' = *DCU*: moduł LC-055-DCU;

BaseAdres = 'x' - 'x' = adres bazowy modułu (szesnastkowo); moduły A...D muszą mieć adresy bazowe zgodne ze standardowymi ustawieniami (patrz instrukcja obsługi modułu);

Ports = 'x' - 'x' = ilość portów cyfrowych modułu; LC-055-PIO - 6; LC-055-DCU - 3 lub 6 w zależności od liczby zamontowanych układów 8255;

ClkFrequency = 'x' - 'x' = częstotliwość zegara modułu w MHz (*4* lub *8*);

- następna linia powinna być wyspecyfikowana tylko dla modułów LC-055-PIO;

PortInLatchEnable = **x1,x2,x3** - $x1 \dots x3 = YES$ lub **NO**; przyłączenie linii zewnętrznego ładowania rejestrów -GATE_IN oraz linii programowej do ładowania (zatrzaskiwania) par rejestrów; parametry $x1 \dots x3$ odpowiadają jumperom karty, odpowiednio JP 13 .. JP 15; **YES** = linia -GATE_IN przyłączona, **NO** - nieprzyłączona, brak możliwości zatrzaskiwania danych;

CTCClk = **x1,x2,x3** - określenie źródła przebiegu zegarowego kanałów 0,1,2 CTC: odpowiednio parametry $x1,x2,x3$;

parametry $x1,x2,x3$ mogą przybierać następujące wartości:

- moduł LC-055-PIO - przełączniki JP 10 .. JP 12 i JP 4 .. JP 9:

EXT - wejście przyłączone do zewnętrznej linii zegarowej CTC

2 - wejście przyłączone do zegara wewnętrznego, dzielnik zegara = 2 (4 lub 2 MHz)

4 - wejście przyłączone do zegara wewnętrznego, dzielnik zegara = 4 (2 lub 1 MHz)

16 - wejście przyłączone do zegara wewnętrznego, dzielnik zegara = 16 (0.5 lub 0.25 MHz)

- moduł LC-055-DCU - przełączniki JP 1.. JP 9:

EXT - wejście przyłączone do zewnętrznej linii zegarowej CTC

8 - wejście przyłączone do zegara wewnętrznego, dzielnik zegara = 8 (1 lub 0.5 MHz)

16 - wejście przyłączone do zegara wewnętrznego, dzielnik zegara = 16 (0.5 lub 0.25 MHz)

IRQNumber = 'x' - 'x' - numer linii sprzętowej przerwania, do której dołączony jest dany moduł: dla modułu LC-055-PIO: 'x' = **2, 3, 4, NO**; dla modułu LC-055-DCU: 'x' = **2, 3, 4, 5, NO**; **NO** - moduł nie jest przyłączony do żadnego przerwania;

Ostatni parametr określa przyłączenie źródeł przerwania do poszczególnych linii zgłoszenia przerwania:

- moduł LC-055-PIO - zworki JP 16, JP 17:

IRQSource = 'x' - 'x' = **EXT**: przyłączenie linii przerwania do złącza (ustawienie standardowe),
- 'x' = **NO**: linia odłączona - brak możliwości generowania przerwania;

- moduł LC-055-DCU - przełącznik DS3:

IRQSource = **x1,x2,x3,x4** - $x1 \dots x4$ określają konfigurację przerwania odpowiednio linii INT1..INT4;

NO - linia odłączona,

EXT - linia przyłączona bezpośrednio do odpowiedniej linii zewnętrznej;

CTC - linia podłączona z odpowiednim wyjściem układu CTC 8254;

3. Parametry są ładowane przy starcie biblioteki. Jest wtedy przeprowadzana kontrola składni i jeżeli wystąpi błąd składniowy to moduł nie jest inicjowany.

4. Parametry odpowiadające ustawieniom standardowym modułów:

4.1 Moduły LC-055-PIO:

[LC-055]

Path = C:\AMBEX

Modules = 1

[MODULE A]

Type = PIO

BaseAddress = 220

Ports = 6

ClkFrequency = 8

PortInLatchEnable = NO,NO,NO
CTCClk = 4,4,4
IRQNumber = 4
IRQSource = EXT

4.2 Moduły LC-055-DCU:

[LC-055]

Path = C:\AMBEX
Modules = 1

[MODULE A]

Type = DCU
BaseAddress = 220
Ports = 6
ClkFrequency = 8
CTCClk = 16,16,16
IRQNumber = 4
IRQSource = CTC,CTC,CTC,CTC