EGMONT INSTRUMENTS

Informacji na temat kart produkowanych dawniej przez Ambex udziela firma Egmont Instruments.

Pod adresem **http://www.ambex.com.pl** powstaje archiwum instrukcji i oprogramowania do kart komputerowych produkowanych dawniej przez Ambex, a obecnie jeszcze w części oferty przez Egmont Instruments. Archiwum to jest systematycznie uzupełniane. Jeśli poszukują Państwo informacji do starych lub aktualnych wyrobów, prosimy kierować się właśnie pod powyższy adres w Internecie. Jeśli nie znajdą tam Państwo potrzebnej informacji, prosimy o bezpośredni kontakt z nami.

Strony http://www.ambex.com.pl są prowadzone bezpośrednio przez firmę Egmont Instruments.

EGMONT INSTRUMENTS	**tel. (0-22) 823-30-17, 668-69-75**
02-304 Warszawa, Aleje Jerozolimskie 141/90	**fax (0-22) 659-26-11**
http://www.egmont.com.pl	**e-mail egmont@egmont.com.pl**
http://www.ambex.com.pl	**e-mail ambex@ambex.com.pl**

DOKUMENTACJA BIBLIOTEKI DLL DLA ŚRODOWISKA MICROSOFT WINDOWS MODUŁU KONTROLNO-POMIAROWEGO

LC-010-1612

WERSJA 1.1

Wydanie: marzec 1996

AMBEX POLAND Spółka z o.o.
00-350 Warszawa, ul. Topiel 6
dział handlowy i serwis
9:00-17:00

tel. (0-22) 635-87-24
tel. (0-22) 635-04-05
fax (0-22) 635-91-51
tlx 815262 ambex pl

1. Informacje ogólne.

Przy tworzeniu oprogramowania modułów w środowisku MS WINDOWS przyjęto zasadę, że cała komunikacja z modułem 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 pozostawania w pamięci przez czas dostarczania obsługi aplikacjom.

Takie rozwiązanie 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 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 czterech) modułów danego rodzaju obsługiwanych przez jeden driver. Ponieważ standard ten został zaprojektowany do różnych typów modułów, niektóre parametry wydawać się mogą nadmiarowe.

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 uwagę na specyfikę nowego środowiska.

2. Instalacja biblioteki.

Instalacja polega na skopiowaniu z dyskietki do katalogu docelowego:

- zbioru LC*.DLL (odpowiedni dla rodzaju komputera oraz modułu (np. dla LC-010-1612 - LC1016A.DLL),
- zbioru AMBEX.INI, który określa konfigurację zainstalowanych w komputerze modułów 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 - standartowo 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 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. Poniżej podany zostanie skrótowy opis wszystkich parametrów (pełny opis struktury pliku zawarty jest w dodatku E).

Parametry ogólne:

a) typ modułu: LC-010-1612, LC-011-1612,LC-015-1612, LC-020-0812, LC-020-3212, LC-030-1612;
b) liczba zainstalowanych modułów danego typu: od 1 do 4 [6)];
c) pełna nazwa ścieżki do katalogu, gdzie skopiowano zbiór LC*.DLL (łącznie z nazwą napędu dyskowego);
d) maksymalny czas obsługi przerwania, które może mieć miejsce w trakcie trwania długich pomiarów - podawany w mikrosekundach (dodatnia liczba mieszcząca się na 16 bitach) [1) 6)].

Parametry dotyczące jednego modułu [2)]:

e) nazwa modułu: od A do D (nazwa związana jest z adresem bazowym modułu);
f) wybór trybu pracy: z pamięcią rozszerzoną lub bez ;
g) rozmiar pamięci rozszerzonej w kilobajtach;
h) tryb pracy modułu: master lub slave (istotne przy współpracy kilku modułów) [6)];

i) częstotliwość zegara modułu: 4 lub 8 MHz [6];
j) zakres napięć dla wejść analogowych;
k) liczba kanałów wyjść analogowych [3];
l) zakres napięć dla wyjścia analogowego - kanał 1 [5];
m) zakres napięć dla wyjścia analogowego - kanał 2 [1) 5) 6)].
n) numery kanałów DMA przypisane torów wejść i wyjść analogowych,
o) numer przerwania sprzętowego przypisanego danemu modułowi,
p) wartości minimalnych okresów próbkowania dla toru wejść i wyjść analogowych,
r) typ przetwornika A/C i jego czas konwersji.

1) nie dotyczy pracy z modułem LC-010-1612.
2) przy instalacji kilku modułów jednego typu parametry dotyczące każdego z modułów różnią się między sobą np. moduły muszą różnić się nazwą.
3) moduł LC-010-1612: liczba kanałów zależna jest od tego czy na module zainstalowano multiplekser i wynosi 16 lub 1,
moduły LC-011-1612 i LC-030-1612: liczba kanałów jest stała i wynosi 16,
moduły LC-020-0812 i LC-020-3212: liczba kanałów jest zależna od liczby zainstalowanych wzmacniaczy Sample&Hold
4) dotyczy tylko modułu LC-010-1612.
5) nie dotyczy pracy z modułem LC-020-3212.
6) nie dotyczy pracy z modułem LC-030-1612.

Uwaga: sprawdzenie obecności modułu, podłączenie do przerwania sprzętowego oraz numer tego przerwania wykrywane są automatycznie przez driver w momencie ładowania do pamięci.

3. Opis biblioteki.

3.1. Informacje wstępne.

Przed driver'em postawione zostały następujące zadania:
- pełne wykorzystanie możliwości sprzętowych oferowanych przez obsługiwane moduły
- rozszerzenie w sposób programowy możliwości modułów o funkcje, które nie są lub nie mogą być realizowane sprzętowo; do funkcji takich należą:
- różnorodne warunki startu operacji wejścia / wyjścia (odczyt / zapis portów cyfrowych, przetwarzanie a/c i c/a); w grę wchodzi tu warunkowanie startu operacji sygnałami cyfrowymi (poziomem, zboczem, kombinacją sygnałów), oraz upływem czasu rzeczywistego (data i odcinek czasu);
- 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 pobiera i analizuje parametry instalacji podane w pliku konfiguracyjnym AMBEX.INI - dzięki tym informacjom możliwe jest pisanie programów niezwiązanych z konkretną instalacją modułu. Następnie wykonywane jest tzw. twarde zerowanie wszystkich zadeklarowanych modułów, w trakcie którego wykonywane jest wstępne programowanie. Następnie wykonywane są testy, których celem jest sprawdzenie czy rzeczywiście jest zainstalowany moduł, do jakich przerwań i kanałów DMA podłączony jest każdy z zainstalowanych modułów.

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 analogowych 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 rozróżnienie małe / duże litery).

Wszystkie driver'y widziane są w systemie WINDOWS jako moduły. Nazwy tych modułów są następujące:

typ modułu	nazwa driver'a	nazwa modułu
LC-010-1612	LC1016A.DLL	LC1016A
LC-011-1612	LC1116A.DLL	LC1116A
LC-012-1612	LC1216A.DLL	LC1216A
LC-015-1612	LC1516A.DLL	LC1516A
LC-020-0812	LC2008A.DLL	LC2008A
LC-020-3212	LC2032A.DLL	LC2032A
LC-030-1612	LC3016A.DLL	LC3016A

Kodowanie numerów modułów:

moduł	nazwa kodu	wartość
A	LC0_MODA	1
B	LC0_MODB	2
C	LC0_MODC	3
D	LC0_MODD	4

Kodowanie typów urządzeń w modułach:

urządzenie	nazwa kodu	wartość
porty cyfrowe wejściowe	LC0_DINPUT	1
porty cyfrowe wyjściowe	LC0_DOUTPUT	2
przetworniki a/c	LC0_AINPUT	3
przetworniki c/a	LC0_AOUTPUT	4
kanały CTC	LC0_CTC	5

Wszystkie wejścia, wyjścia, kanały itp. numerowane są od 1.
Długości buforów, pomiarów, marginesów itp. podawane są zawsze w próbkach.

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. GetProcAddres(); W momencie zakończenia programu należy jawnie zwolnić bibliotekę wywołując funkcję FreeLibrary().
Podane sposoby, typowe dla języka C moją 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 niezmienne i mają następujące znaczenie:

nazwa	rozmiar w bajtach	znaczenie

LC0_CODE	1	kod funkcji
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach

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)

LC0_ERR_STAT (jeżeli jest mniejszy od zera) niesie pewne dodatkowe informacje komentujące zwrócony w LC0_STATUS kod błędu. Dotyczy to dwóch sytuacji:
- błędnie podane parametry warunku startu / stopu (LC0_STATUS = LC0_ILL_START / LC0_ILL_STOP); LC0_ERR_STAT precyzuje co zostało podane błędnie
- funkcja przerwana wywołaniem funkcji BREAK (LC0_STATUS = LC0_BROKEN); LC0_ERR_STAT określa wtedy, czy funkcja została przerwana w trakcie oczekiwania na spełnienie warunku startu czy już w trakcie przetwarzania.

Istnieje jeden wyjątek od tej zasady: funkcja BREAK może zwrócić LC0_STATUS = 0 i LC0_ERR_STAT <> 0 (patrz opis funkcji, p.4.9).

W rozdziale 5 podano tabele wszystkich kodów zwracanych przez LC0_STATUS i LC0_ERR_STAT.

3.4. Typy warunków startu / zakończenia operacji.

W driverze zaimplementowano następujące startu operacji:
- start natychmiastowy: bez czekania na spełnienie jakichkolwiek warunków
- poziom sygnału cyfrowego: warunek jest spełniony, gdy sygnał cyfrowy ma zadaną wartość;
- zbocze sygnału cyfrowego: warunek jest spełniony, gdy sygnał cyfrowy zmieni swoją wartość w określony sposób;
- kombinacja sygnałów cyfrowych: warunek jest spełniony, gdy kombinacja sygnałów cyfrowych jest równa (różna) zadanej; w pierwszym przypadku wszystkie zadeklarowane sygnały muszą mieć zadany poziom, w drugim - wystarczy, że jeden z sygnałów ma poziom różny od zadanego;
- data: warunek jest spełniony, gdy bieżąca data zrówna się z zadaną;
- czas: warunek jest spełniony po upływie zadanego odcinka czasu;
- sygnał analogowy : warunek jest spełniony po osiągnięciu przez sygnał analogowy na wybranym wejściu określonej wartości.

Należy pamiętać o tym, że powyżej opisane warunki startu realizowane są programowo w związku z czym początek koniec pomiaru jest zawsze nieco opóźniony względem momentu spełnienia warunku. Opóźnienie to (rzędu mikrosekund) zależne jest od szybkości komputera.

Warunki startu związane z pomiarem czasu realizowane są w oparciu o czas systemowy. Z tego powodu czas podawany jest z dokładnością do sekundy.

Dla zakończenia operacji zdefiniowano następujące warunki:
- przetworzenie określonej liczby próbek;

Kodowanie typu warunku startu operacji:

kod typu /	parametry warunku				
znaczenie	bajt 1	bajt 2	bajt 3	bajt 4	bajt 5
0 (LC0_SIMMED natychmiast	---	---	---	---	---
1 (LC0_SHARD) od sygnału sprzętowego	---	---	---	---	---
2 (LC0_SLEVEL) 1) od poziomu sygnału cyfrowego	numer modułu	numer portu	numer wejścia	poziom	---
3 (LC0_SSLOPE) 2) od zbocza sygnału cyfrowego	numer modułu |	numer portu	numer wejścia	zbocze	---

4 (LC0_SDIG_EQ) 3) od kombinacji wejść cyfrowych warunek równości	numer modułu	numer portu	maska	wzorzec	---
5 (LC0_SDIG_NE) 3) od kombinacji wejść cyfrowych warunek nierówności	numer modułu	numer portu	maska	wzorzec	---
6 (LC0_STIME) po upłynięciu określonego czasu	odcinek czasu w sekundach				---
7 (LC0_SDATE) 4) o podanym czasie	sekunda	minuta	godzina	dzień miesiąca	---
8 (LC0_SANALOG) 5) od sygnału analogowego	numer modułu	nr przetwornika	nr kanału	próg wyzwolenia	

1) "Poziom" wskazuje oczekiwany stan wejścia cyfrowego (0/1). Wszystkie wartości różne od zera traktowane są jak "1".
2) Oczekiwane zbocze wejścia cyfrowego kodowane jest następująco:
0 - zbocze opadające, 1 - zbocze narastające.
3) Bajt maski wskazuje, które bity portu wejściowego brane są pod uwagę przy badaniu warunku: 1 wskazuje bit badany, 0 - ignorowany.
4) "Dzień miesiąca" dotyczy bieżącego miesiąca. Jeżeli numer dnia jest mniejszy niż bieżący - następnego miesiąca.
5) "nr kanału":
- bity b1..b6 określają numer monitorowanego kanału analogowego;
- bit b7 określa typ wyzwolenia: "0" - gdy sygnał osiągnie zadany poziom na zboczu narastającym lub opadającym, zależnie od bitu b8, "1" - gdy sygnał jest większy lub mniejszy od zadanego poziomu, zależnie od bitu b8.
- bit b8 określa kierunek zmian monitorowanego sygnału: "0" - w kierunku mniejszych wartości (zbocze opadające / mniejszy od zadanego poziomu); "1" - w kierunku większych wartości (zbocze narastające/większy od zadanego poziomu).
"próg wyzwolenia":
- bity b1..b12 określają zadany poziom wyzwolenia, wyrażony w kodzie przetwornika A/C;
- bity b13..b16 określają stopień redukcji zakłóceń (n=0..15) zgodnie z następującymi zasadami: podczas detekcji zbocza (b7=0), by nastąpiło wyzwolenie pomiaru musi nastąpić n pomiarów przed wyzwoleniem, potwierdzających zadany kierunek zmian sygnału; podczas detekcji poziomu (b7=1), by nastąpiło wyzwolenie pomiaru musi nastąpić n pomiarów przed wyzwoleniem, dla których spełniony jest warunek na osiągnięcie zadanego poziomu sygnału.

Kodowanie typu warunku stopu (zatrzymania) operacji (kody podane szesnastkowo):

kod typu (szesnastkowo)/	parametry warunku				
znaczenie	bajt 1	bajt 2	bajt 3	bajt 4	bajt 5
00 (LC0_ZSAMPLES) po zmierzeniu bloku danych	liczba próbek do zmierzenia				---

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 struktury danych (dodatki A, C i E)
- znaczenie
- przeznaczenie funkcji
- szczegółowy opis parametrów funkcji (pól 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
- dodatkowe informacje o błędach; lista dodatkowych informacji zwracanych przez funkcję w parametrze LC0_ERR_STAT; jeżeli punkt ten nie występuje to oznacza to, że dana funkcja nie zwraca żadnych dodatkowych informacji (LC0_ERR_STAT zawsze równe LC0_E_OK: 0).

4.1. Inicjalizacja (MODULE_INIT).

Nazwa funkcji: LC0_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_ERR_STAT	1	dodatkowe informacje o błędach
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. Dodatkową czynnością jest inicjacja dla każdego z modułów własnego bufora pamięci rozszerzonej (o ile był zadeklarowany w pliku AMBEX.INI). Moduły podlegające inicjalizacji specyfikuje się na poszczególnych bitach parametru LC0_IMODULE:

```
b8      b1
- - - - D C B A
0 0 0 0 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).

Nazwa funkcji: LC0_GETTOTALCONF; 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
LC0_ERR_STAT	1	dodatkowe informacje o błędach
parametry wyjściowe:		
LC0_TONF	1	konfiguracja modułów
LC0_TIAD	1	liczba dostępnych przetworników a/c
LC0_TIDA	1	liczba dostępnych przetworników c/a

LC0_TCTC	1	liczba dostępnych kanałów CTC
LC0_TIDI	1	liczba dostępnych portów we. cyfrowych
LC0_TIDO	1	liczba dostępnych portów wy. cyfrowych

Przeznaczenie:

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

```
b8      b1
D C B A D C B A
y y y y x x x x
```

x: x = 1 - moduł zainstalowany, x = 0 - modułu nie ma
y: y = 1 - moduł "master", y = 0 - moduł "slave" (dla modułów serii LC-010 , LC-011 , LC-020 wszystkie moduły są typu "master")

4.3. Informacja o konfiguracji modułu (GET_MODULE_INFORMATION).

Nazwa funkcji: LC0_GETMODULE; nr porządkowy: 18

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 18
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_MMODULE		numer modułu
parametry wyjściowe:		
LC0_MBASE1	2	adres bazowy modułu - pakiet 1
LC0_MBASE2[1)]	2	adres bazowy modułu - pakiet 2
LC0_MIAD	1	liczba dostępnych przetworników a/c
LC0_MIDA	1	liczba dostępnych przetworników c/a
LC0_MCTC	1	liczba dostępnych kanałów CTC
LC0_MIDI	1	liczba dostępnych portów we. cyfrowych
LC0_MIDO	1	liczba dostępnych portów wy. cyfrowych
LC0_MCLOCK	2	częstotliwość zegara modułu w kHz; częstotliwość próbkowania tworzona jest przez podział 1/4 częstotliwości zegara modułu
LC0_MINT	1	numer przerwania (programowy)
LC0_MMEMA	4	adres bufora pamięci rozszerzonej (offset-segment)
LC0_MMEML	4	wielkość bufora pamięci rozszerzonej w próbkach

1) dla modułów serii LC-010 , LC-011 i LC-020 - nieokreślone (moduły są jednopakietowe)

Przeznaczenie:

Funkcja zwraca informację o konfiguracji wyspecyfikowanego modułu.

Błędy (LC0_STATUS):

LC0_NO_MODULE - nie ma takiego modułu
LC0_BAD_CONFIG - błąd w pliku konfiguracyjnym AMBEX.INI.

4.4. Informacja o szczegółach technicznych (GET_INFO).

Nazwa funkcji: LC0_GETINFO; nr porządkowy: 19

4.4.1. Wejścia cyfrowe.

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 19
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_GTYPE	1	rodzaj urządzenia = 1
LC0_GMODULE	1	numer modułu
LC0_GNUM	1	numer portu wejściowego
parametry wyjściowe:		
LC0_GCHAN	1	liczba bitów portu

Przeznaczenie:
Funkcja zwraca informację o liczbie bitów badanego portu wejściowego.

Błędy (LC0_STATUS):
LC0_NONEX_DEV - port wejściowy o tym numerze nie istnieje
LC0_NO_MODULE - nie ma takiego modułu

4.4.2. Wyjścia cyfrowe.

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 19
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_GTYPE	1	rodzaj urządzenia = 2
LC0_GMODULE	1	numer modułu
LC0_GNUM	1	numer portu wyjściowego
parametry wyjściowe:		
LC0_GCHAN	1	liczba bitów portu

Przeznaczenie:
Funkcja zwraca informację o liczbie bitów badanego portu wyjściowego.

Błędy (LC0_STATUS):
LC0_NONEX_DEV - port wyjściowy o tym numerze nie istnieje
LC0_NO_MODULE - nie ma takiego modułu

4.4.3. Wejścia analogowe.

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 19
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_GTYPE	1	rodzaj urządzenia = 3
LC0_GMODULE	1	numer modułu
LC0_GNUM	1	numer przetwornika
parametry wyjściowe:		
LC0_GCHAN	1	liczba kanałów przetwornika
LC0_GRES	1	liczba bitów przetwornika
LC0_GTIME	2	czas konwersji przetwornika w ns

LC0_GMINV	1	dolna granica zakresu napięć w dziesiątych częściach wolta
LC0_GMAXV	1	górna granica zakresu napięć w dziesiątych częściach wolta
LC0_GDMA	1	1 - zainstalowany układ S&H
LC0_GMINP	64	tablica minimalnych okresów próbkowania

Przeznaczenie:
Funkcja zwraca informację o konfiguracji badanego toru analogowo-cyfrowego.
Tablica LC0_GMINP zawiera wartości minimalnych okresów próbkowania w 1, 2, 3,..., 32 kanałach.
Wypełnionych jest pierwszych n pozycji, gdzie n jest maksymalną liczbą kanałów modułu . Okresy podane są w dziesiątych częściach mikrosekundy i muszą być jawnie podane w pliku konfguracyjnym biblioteki (patrz dodatek E).

Błędy (LC0_STATUS):
LC0_NONEX_DEV - przetwornik a/c o tym numerze nie istnieje
LC0_NO_MODULE - nie ma takiego modułu

4.4.4. Wyjścia analogowe.

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 19
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_GTYPE	1	rodzaj urządzenia = 4
LC0_GMODULE	1	numer modułu
LC0_GNUM	1	numer przetwornika
parametry wyjściowe:		
LC0_GCHAN	1	liczba kanałów przetwornika
LC0_GRES	1	liczba bitów przetwornika
LC0_GTIME	2	czas konwersji przetwornika w ns
LC0_GMINV	1	dolna granica zakresu napięć w dziesiątych częściach wolta
LC0_GMAXV	1	górna granica zakresu napięć w dziesiątych częściach wolta
LC0_GDMA[1)]	1	numer kanału DMA

1) Dla modułu LC-010-1612 parametr nieokreślony (moduł nie współpracuje z układem DMA)

Przeznaczenie:
Funkcja zwraca informację o konfiguracji badanego toru cyfrowo-analogowego.

Błędy (LC0_STATUS):
LC0_BAD_DEV_TYP - brak przetworników c/a
LC0_NONEX_DEV - przetwornik c/a o tym numerze nie istnieje
LC0_NO_MODULE - nie ma takiego modułu

4.4.5. Kanały układów licznikowo-czasowych (CTC).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 19
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_GTYPE	1	rodzaj urządzenia = 5

LC0_GMODULE	1	numer modułu
LC0_GNUM	1	numer kanału

Przeznaczenie:
Funkcja informuje czy badany kanał CTC istnieje (przez kod odpowiedzi).

Błędy (LC0_STATUS):
LC0_BAD_DEV_TYP - brak dostępnych kanałów CTC
LC0_NONEX_DEV - kanał CTC o tym numerze nie istnieje
LC0_NO_MODULE - nie ma takiego modułu

4.5. Zadeklarowanie częstotliwości zegara magistrali (SET_CLOCK).

Nazwa funkcji: LC0_SETCLOCK; nr porządkowy: 20

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 20
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_LCLOCK	2	częstotliwość zegara w kHz

Przeznaczenie:
Funkcja nie jest realizowana dla opisanych modułów.

4.6. Ustawienie zakresu napięć (SET_VOLTAGE_RANGE).

Nazwa funkcji: LC0_SETVOLTAGE; nr porządkowy: 21

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 21
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_VTYPE	1	rodzaj urządzenia (3 lub 4)
LC0_VMODULE	1	numer modułu
LC0_VNUM	1	numer przetwornika
LC0_VMINV	1	dolna granica zakresu napięć w dziesiątych częściach wolta
LC0_VMAXV	1	górna granica zakresu napięć w dziesiątych częściach wolta

Przeznaczenie:
Funkcja nie jest realizowana dla opisanych modułów.

4.7. Zadeklarowanie maksymalnego czasu obsługi obcych przerwań (SET_TIME).

Nazwa funkcji: LC0_SETTIME; nr porządkowy: 22

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 22
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach

LC0_ETIME	2	maksymalny czas obsługi w dziesiątych częściach µs

Przeznaczenie:
Funkcja nie jest realizowana dla opisanych modułów.

4.8. Oczekiwanie na zakończenie operacji (WAIT_FOR_END).

Nazwa funkcji: LC0_WAIT; nr porządkowy: 23

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 23
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_WTYPE	1	rodzaj urządzenia (3 lub 4)
LC0_WMODULE	1	numer modułu
LC0_WNUM	1	numer przetwornika
LC0_WMODE	1	tryb pracy
parametry wyjściowe:		
LC0_WRMNUM	4	rzeczywista liczba zmierzonych / wysłanych próbek
LC0_WREMAR	2	rzeczywista długość marginesu końcowego

Przeznaczenie:
Funkcja nie jest realizowana dla opisanych modułów.

4.9. Przerwanie operacji (BREAK).

Nazwa funkcji: LC0_BREAK; nr porządkowy: 24

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 24
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_BMODE	1	tryb pracy
LC0_BPROC	4	adres procedury obsługi Ctrl-Break

Przeznaczenie:
Przerywanie pracy driver'a i modułu, instalowanie i wyinstalowywanie procedur obsługi przerwania generowanego przez klawisz Ctrl-Break.

Tryby pracy:

nazwa	wartość	znaczenie
LC0_BREAK_EXEC	0	przerwij pracę driver'a i modułu
LC0_BREAK_INST	1	zainstaluj procedurę obsługi przerwania Ctrl-Break
LC0_BREAK_UNINST	2	wyinstaluj procedurę obsługi przerwania Ctrl-Break

Przerwanie pracy - tryb LC0_BREAK_EXEC:

Działanie funkcji polega na przerwaniu wszystkich operacji wykonywanych we wszystkich modułach prawidłowo zainicjowanych przez bieżącą aplikację. Funkcja nie oddziaływuje na moduły zainicjowane przez inne aplikacje. W momencie wywołania tej funkcji driver i moduł mogą znajdować się w jednym z dwóch stanów:
1. Nie jest wykonywana żadna operacja - driver zwraca błąd LC0_STATUS = LC0_NO_OPER, LC0_ERR_STAT = LC0_E_OK.
2. Moduł wykonuje pod kontrolą driver'a operację synchroniczną. W tym przypadku funkcja przerywa pracę driver'a. Driver zwraca informację wyjściową w LC0_STATUS i LC0_ERR_STAT.

Zainstalowanie procedury obsługi - tryb LC0_BREAK_INST:

Funkcja instaluje obsługę Ctrl-Break przez przechwycenie sygnału klawiatury. Użytkownik może podać adres własnej procedury obsługi przerwania (LC0_BPROC, daleki adres offset-segment) lub zlecić obsługę standardową - przez procedurę wewnętrzną driver'a (LC0_BPROC = 0:0).

Wyinstalowanie procedury obsługi - tryb LC0_BREAK_UNINST:

Funkcja przywraca procedurę obsługi przerwania istniejącą przed pierwszym wywołaniem funkcji BREAK w trybie LC0_BREAK_INST. Istotne jest więc aby zadbać o wyinstalowanie obsługi Ctrl-Break przed zakończeniem programu.

Procedura obsługi wyinstalowywana jest również przez funkcję LEAVE_DRIVER (patrz).

Błędy (LC0_STATUS):
LC0_INTR_INST - procedura obsługi Ctrl-Break jest już zainstalowana (dla żądania zainstalowania procedury)
LC0_NO_OPER - z żadnym modułem nie jest związana żadna operacja w toku (dla trybu pracy LC0_BREAK_EXEC)
LC0_BAD_MODE - błędny tryb pracy

Dodatkowe informacje (LC0_ERR_STAT; tylko dla trybu pracy LC0_BREAK_EXEC):
LC0_E_BROKEN_RUN - funkcja wywołana w trakcie przetwarzania przy operacji asynchronicznej
LC0_E_BROKEN_WAIT- funkcja wywołana w trakcie oczekiwania na spełnienie warunku startu przy operacji asynchronicznej

4.10. Wejście cyfrowe (DIGITAL_INPUT).

Nazwa funkcji: LC0_DIGITALIN; nr porządkowy: 25

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 25
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_DMODULE	1	numer modułu
LC0_DNUM	1	numer portu
LC0_DSTST	1	typ warunku startu operacji
LC0_DVAL	1	odczytana wartość
LC0_DSTART	5	parametry warunku startu

Przeznaczenie:

Funkcja odczytuje stan wejść cyfrowych podanego portu . Warunkiem startu nie może być start od sygnału sprzętowego (LC0_SHARD) .

Błędy (LC0_STATUS):
LC0_ILL_START - błędne parametry trybu startu operacji
LC0_ILL_START_CODE - błędny tryb startu operacji
LC0_NONEX_DEV - port wejściowy o tym numerze nie istnieje
LC0_NO_MODULE - nie ma takiego modułu
LC0_NOT_INIT - moduł nie zainicjowany
LC0_REJECTED - moduł zajęty przez inne zadanie

Dodatkowe informacje o błędnych parametrach warunku startu operacji (LC0_ERR_STAT):
LC0_E_BAD_CHAN - numer nieistniejącego kanału
LC0_E_BAD_DATE - zła specyfikacja daty

LC0_E_BAD_TIME - zły odcinek czasu
LC0_E_NONEX_DEV - nie istnieje urządzenie o tym numerze
LC0_E_NO_MODULE - nie ma takiego modułu

4.11. Wyjście cyfrowe (DIGITAL_OUTPUT).

Nazwa funkcji: LC0_DIGITALOUT; nr porządkowy: 26

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 26
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_OMODULE	1	numer modułu
LC0_ONUM	1	numer portu
LC0_OSTST	1	typ warunku startu operacji
LC0_OVAL	1	zapisywana wartość
LC0_OSTART	5	parametry warunków startu

Przeznaczenie:
Wysłanie wartości na wyjścia cyfrowe podanego portu . Warunkiem startu nie może być start od sygnału sprzętowego (LC0_SHARD) .

Błędy (LC0_STATUS):
LC0_ILL_START - błędne parametry trybu startu operacji
LC0_ILL_START_CODE - błędny tryb startu operacji
LC0_NONEX_DEV - port wejściowy o tym numerze nie istnieje
LC0_NO_MODULE - nie ma takiego modułu
LC0_NOT_INIT - moduł nie zainicjowany
LC0_REJECTED - moduł zajęty przez inne zadanie

Dodatkowe informacje o błędnych parametrach warunku startu operacji (LC0_ERR_STAT):
LC0_E_BAD_CHAN - numer nieistniejącego kanału
LC0_E_BAD_DATE - zła specyfikacja daty
LC0_E_BAD_TIME - zły odcinek czasu
LC0_E_NONEX_DEV - nie istnieje urządzenie o tym numerze
LC0_E_NO_MODULE - nie ma takiego modułu

4.12. Zapis CTC (CTC_WRITE).

Nazwa funkcji: LC0_CTCWRITE; nr porządkowy: 27

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 27
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_CMODULE	1	numer modułu
LC0_CMODE	1	tryb pracy funkcji
LC0_CFUN	1	tryb pracy kanału CTC
LC0_CVAL	2	wpisywana wartość licznika

Przeznaczenie:
Przeznaczeniem funkcji jest zapisanie trybu pracy licznika układu 8254 i/lub jego nowej wartości a także uruchomienie licznika. 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	LC0_CTC_ENABLE	zezwól na pracę licznika

Pozostałe bity są ignorowane.

Błędy

LC0_NO_MODULE - dany moduł nie istnieje
LC0_BAD_CTC_MODE - błędny parametr LC0_CFUN (część dotycząca trybu pracy kanału CTC)
LC0_NONEX_DEV - w parametrze LC0_FCFUN podano numer nieistniejącego kanału CTC
LC0_CTC_NOT_PROGRAMMED - zlecono zapis wartości lecz nie zaprogramowano trybu pracy kanału

4.13. Odczyt CTC (CTC_READ).

Nazwa funkcji: LC0_CTCREAD; nr porządkowy 28:

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 28
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_UMODULE	1	numer modułu
LC0_UNUM	1	numer kanału CTC
parametry wyjściowe:		
LC0_UVAL	2	odczytana wartość licznika

Przeznaczenie:
Funkcja odczytuje bieżącą wartość zadanego kanału CTC.

Błędy

LC0_NO_MODULE - dany moduł nie istnieje
LC0_NONEX_DEV - w parametrze LC0_FCFUN podano numer nieistniejącego kanału CTC
LC0_CTC_NOT_PROGRAMMED - zlecono zapis wartości lecz nie zaprogramowano trybu pracy kanału

4.14. Transmisja danych (DATA_TRANSMIT).

Nazwa funkcji: LC0_DATATRANSMIT; nr porządkowy: 29

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 29
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_RMODULE	1	numer modułu
LC0_RMODE	1	tryb przesyłania
LC0_RADDR	4	adres bufora (offset-segment)
LC0_RLEN	4	długość bufora
LC0_RMEAS	4	numer próbki, od której należy zacząć
LC0_RNUM	4	liczba próbek do przesłania
LC0_RMEMA	4	adres bufora w pamięci rozszerzonej (offset-segment)
parametry wyjściowe:		
LC0_RRNUM	4	rzeczywista liczba przesłanych próbek

Przeznaczenie:
Funkcja przepisuje ciąg próbek między buforem w pamięci podstawowej a buforem w pamięci rozszerzonej.

Znaczenie poszczególnych parametrów:

a. LC0_RMODULE
Numer modułu do którego odnosi się transmisja.

b. LC0_RMODE

Tryb pracy funkcji:

nr bitu	wartość	nazwa	znaczenie
1	1	LC0_TO_EXT_DIR	do pamięci rozszerzonej
	0	LC0_FROM_EXT_DIR	z pamięci rozszerzonej
2..8	---	---	zarezerwowane; zawsze 0

c. LC0_RADDR
Adres bufora w podstawowej pamięci komputera.

d. LC0_RLEN
Długość bufora LC0_RADDR podana w próbkach. Ten parametr nie jest związany z liczbą próbek do przesłania.

e. LC0_RMEAS
Numer pierwszej próbki do przesłania. Przesłane zostaną próbki o numerach LC0_RMEAS, LC0_RMEAS + 1 itd. Należy pamiętać, że pierwsza próbka ma numer 1. Jeżeli mierzonych było np. 5 kanałów to 10 próbka w pierwszym kanale ma numer 51.

f. LC0_RNUM
Całkowita liczba próbek do przesłania.

g. LC0_RMEMA
Parametr określa daleki adres bufora w pamięci rozszerzonej. Adres ten nie może wychodzić poza obszar bufora zadeklarowanego przy instalacji driver'a.

h. LC0_RRNUM
Rzeczywista liczba przepisanych próbek. Parametr LC0_RRNUM przybiera wartość będącą minimum z LC0_RLEN, <n> i LC0_RNUM. <n> jest tu liczbą próbek zawartych między próbką LC0_RMEAS a końcem bufora w pamięci rozszerzonej.

Ostrzeżenia (LC0_STATUS):
LC0_OTHER_LEN - przepisano mniejszą liczbę próbek niż żądano

Błędy (LC0_STATUS):
LC0_NO_MODULE - nie ma takiego modułu
LC0_NOT_INIT - moduł nie zainicjowany
LC0_REJECTED - moduł zajęty przez inne zadanie
LC0_BAD_BUF_ADR - błędny adres bufora w pamięci podstawowej (odnoszący się do nieistniejącej pamięci)
LC0_BAD_BUF_LEN - błędna długość bufora w pamięci podstawowej (powodująca wyjście bufora poza pamięć itp.)
LC0_BAD_EXTMEM - błędny adres bufora w pamięci rozszerzonej
LC0_BAD_MNUM - błędny numer pierwszej próbki (np. powodujący wyjście poza jeden z buforów)
LC0_BAD_MODE - błędny tryb pracy
LC0_BROKEN - transmisja przerwana z powodu wykonania funkcji BREAK
LC0_NO_EXTMEM - brak pamięci rozszerzonej

4.15. Przetwarzanie analogowo-cyfrowe (ANALOG_INPUT).

Nazwa funkcji: LC0_ANALOGIN; nr porządkowy: 30

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 30
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach

LC0_AMODULE	1	numer modułu
LC0_ANUM	1	numer przetwornika
LC0_AMODE	2	tryb pracy funkcji
LC0_ASTST	1	typy warunków startu / stopu operacji
LC0_APER	4	okres próbkowania
LC0_APER2[1)]	2	wielokrotność okresu próbkowania dla kanałów dodatkowych
LC0_ACHAN	1	liczba kanałów / numer kanału
LC0_ACHAN2[1)]	1	liczba kanałów dodatkowych
LC0_AADDR	4	adres bufora (offset-segment) w pamięci podstawowej
LC0_ALEN	4	długość bufora w pamięci podstawowej
LC0_AMEMA	4	adres bufora w pamięci rozszerzonej (offset-segment)
LC0_ABMAR[1)]	2	długość marginesu początkowego
LC0_AEMAR[1)]	2	długość marginesu końcowego
LC0_AHAND[1)]	2	numer handlera pliku dyskowego
LC0_ASTART	5	warunki startu
LC0_ASTOP	5	warunki stopu
parametry wyjściowe:		
LC0_ARDIV1	2	rzeczywisty podzielnik zegara modułu
LC0_ARDIV2[2)]	2	- " -
LC0_ARMNUM	4	rzeczywista liczba zmierzonych próbek
LC0_ARBMAR[2)]	2	rzeczywista dł. marginesu początkowego
LC0_AREMAR[2)]	2	rzeczywista dł. marginesu końcowego
LC0_ARLEN	4	rzeczywista liczba przepisanych próbek
LC0_ARBUF[2)]	2	początek bufora cyklicznego

1) Wielkość bez znaczenia dla modułów serii LC-010 .
2) Wielkość nieokreślona dla modułów serii LC-010 .

Przeznaczenie:

Jest to podstawowa funkcja driver'a sterująca główną częścią modułu - torem przetwarzania analogowo - cyfrowego.

Generalnie pomiar (w trybie blokowym) można podzielić na następujące fazy :

1. Przygotowanie toru pomiarowego do pracy.
2. Oczekiwanie na spełnienie warunku startu przetwarzania.
3. Start właściwego pomiaru.
4. Oczekiwanie na spełnienie warunku stopu przetwarzania.
5. Zakończenie pomiaru.

Znaczenie poszczególnych parametrów:

a. LC0_AMODE

Parametr określa tryb pracy funkcji przy czym znaczenie mają kolejne bity parametru:

nazwa	nr bitu	uwagi
LC0_MOD_START	1	
LC0_MOD_NEW_PAR	2	
LC0_MOD_SYNCHR	3	1
LC0_MOD_INTR	4	0
LC0_MOD_INTR_TYPE	5	ignorowane
LC0_MOD_BLOCK	6	
LC0_MOD_CYCL	7	ignorowane
LC0_MOD_FILE	8	ignorowane
LC0_MOD_MEM_W	9	
LC0_MOD_EXT_CLK	10	
LC0_MOD_EXT_MEM	11	
LC0_MOD_PAGE	12	ignorowane
-------	13..16	zarezerwowane; zawsze 0

LC0_MOD_START:
- start pomiarów
  ustawienie tego bitu na 1 oznacza żądanie wykonania przetwarzania; wartość 0 powoduje jedynie analizę poprawności i zapamiętanie parametrów funkcji

LC0_MOD_NEW_PAR:
- ustawienie nowych parametrów
  1 oznacza, że parametry przetwarzania pobierane będą z rekordu opisu zlecenia; 0 oznacza, że parametry przetwarzania będą identyczne jak poprzednio - jeżeli do tej pory nie było wykonania funkcji ANALOG_INPUT lub ostatnie było niepoprawne to driver zgłosi błąd LC0_NO_PARAMS.
  Parametry: LC0_AMODULE ,LC0_ALEN, LC0_AADDR,LC0_AMEMA są pobierane zawsze z bieżącego rekordu.

LC0_MOD_SYNCHR
- rodzaj pracy: synchroniczna (1) / asynchroniczna (0):
- praca synchroniczna: driver zwraca sterowanie dopiero po całkowitym zakończeniu przetwarzania.
- praca asynchroniczna: driver zwraca sterowanie tak szybko jak tylko to jest możliwe ; wykorzystany jest wtedy tryb pracy z przerwaniem , które jest zgłaszane jest po zakończeniu każdego pojedynczego pomiaru (gotowość ADC) .

UWAGA: Praca asynchroniczna nie jest dostępna dla wersji 1.1 driver'a.

LC0_MOD_BLOCK:
- tryb przetwarzania: blokowy (1) / pojedynczy (0)
  tryb blokowy jest trybem podstawowym pracy modułu , w którym transmitujemy blok próbek do pamięci komputera (liczba próbek określona jest w parametrze LC0_ASTOP) , jest również możliwość ustawienia okresu próbkowania (parametr LC0_APER); przy przetwarzaniu w trybie pojedynczym mierzone jest tylko po jednej próbce z każdego zadeklarowanego kanału ; przy pierwszym wykonaniu funkcji w słowie trybu pracy (LC0_AMODE) powinien być ustawiony bit LC0_MOD_NEW_PAR (analizowane są wszystkie parametry funkcji); przy następnych wykonaniach funkcji bit ten może być (choć nie jest to konieczne) zgaszony;

UWAGA: należy zwrócić uwagę, że jeżeli chcemy wykonać pomiar bloku próbek, gdzie warunek startu odnosi się do całego bloku to drugi i kolejne pomiary należy wykonywać z warunkiem startu LC0_SIMMED.

LC0_MOD_MEM_W:
- przepisanie do pamięci (1) / bez przepisywania do pamięci (0) bit steruje działaniem funkcji po zakończeniu przetwarzania w trybie synchronicznym przy współpracy z pamięcią rozszerzoną; jeżeli bit jest równy 1 to po zakończeniu przetwarzania do pamięci podstawowej (pod adres LC0_AADDR) przepisywanych jest <n> pierwszych próbek gdzie <n> jest minimum z całkowitej liczby zmierzonych próbek (LC0_ARMNUM) i rozmiaru bufora (LC0_ALEN); całkowita liczba przepisanych próbek zwracana jest w parametrze LC0_ARLEN; jeżeli natomiast bit LC0_MOD_MEM_W jest równy 0 to po zakończeniu przetwarzania dane nie są przepisywane do pamięci (wówczas parametry LC0_AADDR i LC0_ALEN nie są analizowane); zarówno w jednym jak i drugim przypadku przepisanie danych do pamięci podstawowej jest możliwe poprzez wykonanie funkcji DATA_TRANSMIT; przy pracy z pamięcią podstawową (LC0_MOD_EXT_MEM = 0) bit LC0_MOD_MEM_W jest ignorowany.

LC0_MOD_EXT_CLK:
- praca z zegarem wewnętrznym (0) lub zewnętrznym (1) (linia SAMPLE_IN).

LC0_MOD_EXT_MEM:
- praca z pamięcią rozszerzoną (1) / pamięcią podstawową (0):
  przy pracy z pamięcią rozszerzoną zmierzone dane przesyłane są do bufora w pamięci rozszerzonej pod adres LC0_AMEMA; bufor ten musi się zawierać w buforze zadeklarowanym przy instalacji driver'a; zadeklarowany bufor można wykorzystywać jako kilka mniejszych ale należy pamiętać o tym, że driver nie kontroluje zachodzenia na siebie tak utworzonych buforów; dla pracy z pamięcią rozszerzoną adres (LC0_AADDR) i długość (LC0_ALEN) bufora w pamięci podstawowej są analizowane tylko wtedy gdy jednocześnie zapalony zostanie bit LC0_MOD_MEM_W (patrz wyżej); dla pracy z pamięcią podstawową adres bufora w pamięci rozszerzonej (LC0_AMEMA) nie jest analizowany;

b. LC0_ASTST

Parametr określa typy warunków startu i stopu (ten drugi tylko dla pracy blokowej) pomiaru. Jest on sumą odpowiednich kodów typów warunku startu i stopu (patrz p.3.4.).

c. LC0_ASTART, LC0_ASTOP

Parametry te określają szczegółowo warunki startu i stopu operacji.

Interpretacja ich zależna jest od zadanych typów warunków startu i stopu operacji (LC0_ASTST). Szczegółowy opis - patrz p."3.4.

d. LC0_APER

Okres próbkowania dla pracy blokowej z zegarem wewnętrznym (bit parametru LC0_AMODE LC0_MOD_BLOCK = 1 , LC0_MOD_EXT_CLK = 0). Okres ten podawany jest w dziesiątych częściach mikrosekundy (np. 10000 oznacza 1 ms). Okres ten nie może być mniejszy niż minimalny okres dla danej liczby kanałów. (O minimalnych okresach próbkowania można - należy - się dowiedzieć z programu za pomocą funkcji GET_INFO (parametr

LC0_GMINP, patrz opis funkcji).) Liczba kanałów określająca minimalny okres próbkowania wyznaczana jest następująco:
- dla pracy jednokanałowej (najstarszy bit parametru LC0_ACHAN równy 1, patrz niżej opis tego parametru) - oczywiście 1
- dla pracy wielokanałowej (najstarszy bit parametru LC0_ACHAN równy 0, patrz niżej opis tego parametru) - wartość LC0_ACHAN

Z parametrem tym związany jest parametr zwrotny driver'a LC0_ARDIV1 (patrz opis poniżej).

e. LC0_ACHAN

Parametr określający tryb pracy modułu: jednokanałowo / wielokanałowo oraz (odpowiednio): liczbę kanałów / numer kanału:

b8	tryb pracy	b1..b7
0	praca wielokanałowa	liczba kanałów (2..16)
1	praca jednokanałowa	numer kanału (1..16)

f. LC0_AADDR, LC0_ALEN

Adres i długość bufora w pamięci podstawowej. Parametry te są analizowane tylko wtedy, gdy bit LC0_MOD_EXT_MEM trybu pracy (LC0_AMODE) został ustawiony na 0 (praca z pamięcią podstawową) lub - w przeciwnym razie - gdy bit LC0_MOD_MEM_W = 1 i LC0_MOD_SYNCHR = 1(praca synchroniczna z pamięcią rozszerzoną z przepisaniem do pamięci podstawowej). Adres bufora jest adresem dalekim, tzn. podanym w postaci offset-segment, natomiast długość bufora podawana jest w próbkach (i jest to liczba długa tj. 32 bitowa). Długość bufora jest jednym z czynników określających liczbę próbek przepisywanych do pamięci podstawowej (patrz opis bitu LC0_MOD_MEM_W).

g. LC0_AMEMA

Parametr istotny tylko dla pracy z pamięcią rozszerzoną (LC0_MOD_EXT_MEM = 1). Oznacza daleki (offset:segment) adres bufora w pamięci rozszerzonej.

h. LC0_ARDIV1

Częstotliwość próbkowania tworzona jest przez podział (przez całkowitą wartość) 1/16 częstotliwości generatora w module. Parametr LC0_ARDIV1 przekazuje wartość podzielnika tej częstotliwości . Łącznie z częstotliwością zegara modułu (patrz funkcja GET_MODULE_INFORMATION) daje to informację o faktycznej częstotliwości próbkowania.

i. LC0_ARMNUM

Rzeczywista liczba zmierzonych próbek.

Błędy (LC0_STATUS):

LC0_BAD_BUF_ADR - błędny adres bufora (odnoszący się do nieistniejącej pamięci lub do pamięci rozszerzonej w przypadku transmisji programowej)
LC0_BAD_BUF_LEN - błędna długość bufora (powodująca wyjście bufora poza pamięć, przejście pomiędzy pamięcią podstawową a rozszerzoną, przekroczenie rozmiarów bufora w pamięci rozszerzonej itp.)
LC0_BAD_CHAN - numer nieistniejącego kanału
LC0_BAD_CHAN_N - zła liczba kanałów
LC0_BAD_EXTMEM - błędny adres bufora w pamięci rozszerzonej (dla pracy z pamięcią rozszerzoną, LC0_MOD_EXT_MEM = 1)
LC0_BAD_MARGIN - błędna długość marginesu początkowego (nie będąca wielokrotnością liczby kanałów)
LC0_BAD_MODE - błędny tryb pracy
LC0_BROKEN - operacja przerwana z powodu wykonania funkcji BREAK; LC0_ERR_STAT podaje, w jakim momencie operacja została przerwana
LC0_BAD_PER - za krótki okres próbkowania
LC0_DEV_BUSY - urządzenie zajęte; próba wykonania następnej funkcji ANALOG_INPUT przed zakończeniem poprzedniej
LC0_ILL_START - błędne parametry warunku startu
LC0_ILL_STOP - błędne parametry warunku stopu; w obu przypadkach sprecyzowanie błędu podane jest w LC0_ERR_STAT
LC0_ILL_START_CODE - nielegalny typ warunku startu
LC0_ILL_STOP_CODE - nielegalny typ warunku stopu;
LC0_INTR_NOT_INST - procedura obsługi przerwania nie jest zainstalowana (dla pracy z przerwaniem)
LC0_NONEX_DEV - nie istnieje przetwornik o tym numerze
LC0_NO_EXTMEM - brak pamięci rozszerzonej (dla pracy z pamięcią rozszerzoną)

LC0_NO_IRQ - z danym modułem nie jest związane żadne przerwanie (dla pracy z przerwaniem)
LC0_NO_MODULE - nie ma takiego modułu
LC0_NOT_INIT - moduł nie zainicjowany
LC0_REJECTED - moduł zajęty przez inne zadanie
LC0_NO_PARAMS - zgaszono bit LC0_MOD_NEW_PAR (LC0_AMODE) ale wcześniej nie ustawiono żadnych parametrów (nie było wykonania funkcji ANALOG_INPUT lub ostatnie wykonanie było niepoprawne)
LC0_NO_SECOND_FREQ - moduł nie może prowadzić przetwarzania z dwiema częstotliwościami (LC0_ACHAN2 <> 0)
LC0_OVERRUN - zakończono przetwarzanie z powodu błędu OVERRUN
LC0_TOO_LONG_MARG - suma marginesów dłuższa od bufora

Dodatkowe informacje o błędach (LC0_ERR_STAT):

LC0_E_BAD_CHAN - numer nieistniejącego kanału
LC0_E_BAD_DATE - zła specyfikacja daty
LC0_E_BAD_TIME - zły odcinek czasu
LC0_E_BROKEN_RUN - funkcja przerwana w trakcie przetwarzania
LC0_E_BROKEN_WAIT - funkcja przerwana w trakcie oczekiwania na spełnienie warunku startu
LC0_E_NONEX_DEV - nie istnieje urządzenie o tym numerze
LC0_E_NO_MODULE - nie ma takiego modułu

4.16. Przetwarzanie cyfrowo-analogowe (ANALOG_OUTPUT).

Nazwa funkcji: LC0_ANALOGOUT; nr porządkowy: 31

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 31
LC0_STATUS	1	kod zakończenia funkcji
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_NMODULE	1	numer modułu
LC0_NNUM	1	numer przetwornika
LC0_NMODE	2	tryb pracy funkcji
LC0_NSTST	1	typy warunków startu / stopu operacji
LC0_NCHAN	1	liczba kanałów / numer kanału
LC0_NPER[1)]	4	okres sterowania
LC0_NADDR	4	adres bufora
LC0_NLEN	4	długość bufora
LC0_NHAND[1)]	2	numer handlera pliku dyskowego
LC0_NSTART	5	warunki startu
LC0_NSTOP	5	warunki stopu
parametry wyjściowe		
LC0_NRMNUM	4	rzeczywista liczba wysłanych próbek

Przeznaczenie:

Funkcja służy wysyłania do danych na wyjście analogowe.

a. LC0_NMODE

Parametr określa tryb pracy funkcji przy czym znaczenie mają kolejne bity parametru:

nazwa	nr bitu	uwagi
LC0_MOD_START	1	
LC0_MOD_NEW_PAR	2	
LC0_MOD_SYNCHR	3	1
LC0_MOD_INTR	4	ignorowane
LC0_MOD_INTR_TYPE	5	ignorowane
LC0_MOD_BLOCK	6	
LC0_MOD_CYCL	7	ignorowane

LC0_MOD_FILE	8	zarezerwowane; zawsze 0
LC0_MOD_MEM_W	9	zarezerwowane; zawsze 0
LC0_MOD_EXT_CLK	10	ignorowane
LC0_MOD_EXT_MEM	11	
LC0_MOD_PAGE	12	ignorowane
-------	13..16	zarezerwowane; zawsze 0

LC0_MOD_START:
- start pomiarów
  ustawienie tego bitu na 1 oznacza żądanie wykonania przetwarzania; wartość 0 powoduje jedynie analizę poprawności i zapamiętanie parametrów funkcji

LC0_MOD_NEW_PAR:
- ustawienie nowych parametrów
  1 oznacza, że parametry przetwarzania pobierane będą z rekordu opisu zlecenia;
  0 oznacza, że parametry przetwarzania będą identyczne jak poprzednio - jeżeli do tej pory nie było wykonania funkcji ANALOG_OUTPUT lub ostatnie było niepoprawne to driver zgłosi błąd LC0_NO_PARAMS

Parametry: LC0_NLEN, LC0_NADDR,LC0_NMEMA są pobierane zawsze z bieżącego rekordu .

LC0_MOD_BLOCK:
- tryb przetwarzania: blokowy (1) / pojedynczy (0)
- tryb blokowy: tryb pracy modułu, w którym transmitowany jest cały blok danych na wyjście analogowe o rozmiarze określonym przez parametr LC0_NSTOP ;
- tryb pojedynczy: wysyłane jest tylko po jednej próbce do każdego zadeklarowanego kanału; przy pierwszym wykonaniu funkcji w słowie trybu pracy (LC0_NMODE) powinien być ustawiony bit LC0_MOD_NEW_PAR; nie jest analizowany warunek stopu przetwarzania (część LC0_NSTST oraz LC0_NSTOP) jako nie mający w tym kontekście sensu); przy następnych wykonaniach funkcji bit ten może być (choć nie jest to konieczne) zgaszony; wymagane jest zadeklarowanie adresu bufora z pamięci podstawowej;

UWAGA: należy zwrócić uwagę, że jeżeli chcemy wykonać wysłanie bloku próbek, gdzie warunek startu odnosi się do całego bloku to drugie i kolejne wysłanie należy wykonywać z warunkiem startu LC0_SIMMED

LC0_MOD_EXT_MEM:
- praca z pamięcią rozszerzoną (1) / pamięcią podstawową (0):
  przy pracy z pamięcią rozszerzoną wysyłane dane przesyłane są z bufora w pamięci rozszerzonej (adres LC0_AMEMA); bufor ten musi się zawierać w buforze zadeklarowanym przy instalacji driver'a; zadeklarowany bufor można wykorzystywać jako kilka mniejszych ale należy pamiętać o tym, że driver nie kontroluje zachodzenia na siebie tak utworzonych buforów; dla pracy z pamięcią podstawową adres bufora w pamięci rozszerzonej (LC0_AMEMA) nie jest analizowany;

b. LC0_NSTST

Parametr określa typy warunków startu i stopu (ten drugi tylko dla pracy blokowej) funkcji. Jest on sumą odpowiednich kodów typów warunku startu i stopu (patrz p.3.4.). Warunek startu nie może być typu LC0_SHARD.

c. LC0_NSTART, LC0_NSTOP

Parametry te określają szczegółowo warunki startu i stopu operacji.

Interpretacja ich zależna jest od zadanych typów warunków startu i stopu operacji (LC0_NSTST). Szczegółowy opis - patrz p. 3.4.

d. LC0_NCHAN

Parametr określający tryb pracy modułu: jednokanałowo / wielokanałowo oraz (odpowiednio): liczbę kanałów / numer kanału:

b8	tryb pracy	b1..b7
0	praca wielokanałowa	liczba kanałów (2)
1	praca jednokanałowa	numer kanału (1..n) gdzie n -liczba kanałów c/a w module (1)

f. LC0_NADDR, LC0_NLEN

LC0_NADDR oznacza adres w postaci offset-segment,
LC0_NLEN oznacza długość bufora w pamięci podstawowej.

Błędy (LC0_STATUS):

LC0_BAD_BUF_ADR - błędny adres bufora (odnoszący się do nieistniejącej pamięci, dla pracy z pamięcią podstawową)
LC0_BAD_BUF_LEN - błędna długość bufora (powodująca wyjście bufora poza pamięć, przejście pomiędzy pamięcią podstawową a rozszerzoną itp.; dla pracy z pamięcią podstawową)
LC0_BAD_CHAN - numer nieistniejącego kanału
LC0_BAD_CHAN_N - zła liczba kanałów
LC0_BAD_EXTMEM - błędny adres bufora w pamięci rozszerzonej (dla pracy z pamięcią rozszerzoną)
LC0_BAD_MODE - błędny tryb pracy
LC0_BAD_PER - za długi lub za krótki okres wysyłania
LC0_BROKEN - transmisja przerwana z powodu wykonania funkcji BREAK; LC0_ERR_STAT podaje, w jakim momencie operacja została przerwana
LC0_DEV_BUSY - urządzenie zajęte; próba wykonania następnej funkcji ANALOG_OUTPUT przed zakończeniem poprzedniej
LC0_ILL_START - błędne parametry sposobu startu
LC0_ILL_STOP - błędne parametry warunku stopu; w obu przypadkach sprecyzowanie błędu podane jest w LC0_ERR_STAT
LC0_ILL_START_CODE - nielegalny sposób startu
LC0_ILL_STOP_CODE - nielegalny sposób stopu
LC0_INTR_NOT_INST - procedura obsługi przerwania nie jest zainstalowana (dla pracy z przerwaniem)
LC0_NONEX_DEV - nie istnieje przetwornik o tym numerze
LC0_NO_EXTMEM - brak pamięci rozszerzonej (dla pracy z pamięcią rozszerzoną)
LC0_NO_IRQ - z danym modułem nie jest związane żadne przerwanie (dla pracy z przerwaniem)
LC0_NO_MODULE - nie ma takiego modułu
LC0_NOT_INIT - moduł nie zainicjowany
LC0_REJECTED - moduł zajęty przez inne zadanie
LC0_NO_PARAMS - zgaszono bit LC0_MOD_NEWPAR (LC0_NMODE) ale wcześniej nie ustawiono żadnych parametrów (nie było wykonania funkcji ANALOG_OUTPUT lub ostatnie wykonanie było niepoprawne)

Dodatkowe informacje o błędach (LC0_ERR_STAT):

LC0_E_BAD_CHAN - numer nieistniejącego kanału
LC0_E_BAD_DATE - zła specyfikacja daty
LC0_E_BAD_TIME - zły odcinek czasu
LC0_E_BROKEN_RUN - funkcja przerwana w trakcie przetwarzania
LC0_E_BROKEN_WAIT - funkcja przerwana w trakcie oczekiwania na spełnienie warunku startu
LC0_E_NONEX_DEV - nie istnieje urządzenie o tym numerze
LC0_E_NO_MODULE - nie ma takiego modułu

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

Nazwa funkcji: LC0_LEAVE; 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
LC0_ERR_STAT	1	dodatkowe informacje o błędach

Przeznaczenie:

Funkcja, którą należy wywołać w momencie zakończeniem 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 zeruje wszystkie zainstalowane moduły oraz wyinstalowywuje procedurę obsługi przerwania od Ctrl-Break zainstalowaną przez funkcję BREAK (patrz) a także procedurę obsługi przerwania z modułu zainstalowaną przez funkcję INTERRUPT_SERVICE (patrz).

4.18. Obsługa przerwań (INTERRUPT_SERVICE).

Nazwa funkcji: LC0_INTERRUPT; nr porządkowy: 33

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 33
LC0_STATUS	1	kod zakończenia funkcji = LC0_OK
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_SMODULE	1	numer modułu
LC0_SPROC	4	adres procedury użytkownika
LC0_SSTAT	4	adres słowa komunikacyjnego

Przeznaczenie:
Funkcja w przygotowaniu (opcja).

4.19. Zarządzanie buforem pamięci rozszerzonej (MEMORY_USE).

Nazwa funkcji: LC0_MEMORYUSE; nr porządkowy: 34

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 34
LC0_STATUS	1	kod zakończenia funkcji = LC0_OK
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_MMODULE	1	numer modułu
LC0_MMODE	1	obecnie 1
LC0_MNUM	4	żądana liczba próbek w nowym buforze
parametry wyjściowe		
LC0_MRNUM	4	rzeczywista wielkość bufora w próbkach
LC0_MRMEMA	4	adres bufora w pamięci rozszerzonej (offset:segment)

Przeznaczenie:
Funkcja pozwala zdefiniować nowy rozmiar bufora pamięci rozszerzonej dla danego modułu.
Znaczenie poszczególnych parametrów:

a. LC0_MMODULE
Numer modułu, dla którego przeprowadzana jest realokacja bufora

b. LC0_MNUM
Nowa wielkość żądanego bufora pamięci rozszerzonej w próbkach.

c. LC0_MRNUM
Rzeczywista wielkość otrzymanego bufora w próbkach

d. LC0_MRMEMA
daleki adres nowego bufora pamięci rozszerzonej.

Ostrzeżenia (LC0_STATUS)
LC0_OTHER_LEN - przydzielono mniejszą długość pamięci niż żądano,

Błędy (LC0_STATUS)
LC0_NO_MODULE - zły numer modułu
LC0_NOT_INIT - moduł nie zainicjowany
LC0_REJECTED- moduł zajęty przez inne zadanie
LC0_BAD_EXTMEM - błędny adres bufora w pamięci rozszerzonej
LC0_BAD_MNUM - zła liczba próbek
LC0_BAD_MODE - błędny tryb pracy
LC0_NO_EXTMEM - brak pamięci rozszerzonej dla danego modułu

4.20. Konfiguracja modułów i aktualizacja pliku AMBEX.INI (CONFIG).

Nazwa funkcji: LC0_CONFIG; nr porządkowy: 35

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 35
LC0_STATUS	1	kod zakończenia funkcji = LC0_OK
LC0_ERR_STAT	1	dodatkowe informacje o błędach
LC0_PATH	4	adres łańcucha tekstowego określającego ścieżkę dostępu do pliku AMBEX.INI

Przeznaczenie:
Funkcja w przygotowaniu (opcja).

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_BAD_DEV_TYP	-3	brak urządzeń danego typu
LC0_NONEX_DEV	-4	nie istnieje urządzenie o tym numerze
LC0_BAD_FREQ	-5	błędna częstotliwość
LC0_BAD_RANGE	-6	błędny zakres napięć
LC0_NO_OPER	-7	z wyspecyfikowanym urządzeniem nie jest związana żadna operacja w toku
LC0_BAD_MARGIN	-8	błędna długość marginesu początkowego (nie będąca wielokrotnością liczby kanałów
LC0_BAD_BUF_ADR	-9	błędny adres bufora (odnoszący się do nieistniejącej pamięci lub do pamięci rozszerzonej w przypadku transmisji programowej)
LC0_BAD_BUF_LEN	-10	błędna długość bufora (powodująca wyjście bufora poza pamięć, przejście pomiędzy pamięcią podstawową a rozszerzoną, przekroczenie rozmiarów bufora w pamięci rozszerzonej itp.)
LC0_DEV_BUSY	-11	urządzenie zajęte
LC0_BAD_PER	-12	za długi lub za krótki okres
LC0_BAD_CHAN_N	-13	zła liczba kanałów
LC0_BAD_CHAN	-14	numer nieistniejącego kanału

LC0_BROKEN	-15	przetwarzanie przerwane z powodu wykonania funkcji BREAK
LC0_INTR_NOT_INST	-16	procedura obsługi przerwania nie jest zainstalowana
LC0_ILL_START_CODE	-17	nielegalny typ warunku startu
LC0_ILL_STOP_CODE	-18	nielegalny typ warunku stopu
LC0_BAD_PROC	-19	błędny adres procedury obsługi przerwania lub słowa komunikacyjnego (spoza pamięci podstawowej)
LC0_TOO_LONG_MARG	-20	suma marginesów dłuższa od bufora
LC0_ILL_START	-21	błędne parametry warunku startu
LC0_ILL_STOP	-22	błędne parametry warunku stopu
LC0_BAD_MNUM	-23	błędny numer pierwszej próbki
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_OVERRUN	-27	zakończono przetwarzanie a/c z powodu błędu OVERRUN
LC0_NO_DMA	-28	z danym urządzeniem nie jest związany żaden kanał DMA
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_NO_EXTMEM	-31	brak pamięci rozszerzonej
LC0_NO_SEC_FREQ	-32	moduł nie może prowadzić przetwarzania z dwiema częstotliwościami
LC0_INTR_INST	-33	procedura obsługi przerwania jest już zainstalowana
LC0_BAD_PER2[1)]	-34	błędna wielokrotność okresu próbkowania (0 lub 1)
LC0_BAD_MODE	-35	błędny tryb pracy
LC0_BAD_EXTMEM	-36	błędny adres bufora w pamięci rozszerzonej
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_NO_DMA_TRANS	-41	brak możliwości blokowych transferów DMA

1) Nie występuje dla modułów serii LC-010 .

Dodatkowe informacje o błędach (LC0_ERR_STAT):

nazwa	kod	znaczenie
LC0_E_OK	0	brak dodatkowych informacji
błędy w warunkach startu/stopu		
LC0_E_NO_MODULE	-1	nie ma takiego modułu
LC0_E_NONEX_DEV	-2	nie istnieje urządzenie o tym numerze
LC0_E_BAD_CHAN	-3	numer nieistniejącego kanału
LC0_E_BAD_TIME	-4	zły odcinek czasu
LC0_E_BAD_DATE	-5	zła specyfikacja daty
LC0_E_BAD_THRE	-6	błędny próg wyzwalania analogowego
moment przerwania przez funkcję BREAK:		
LC0_E_BROKEN_WAIT	-7	funkcja przerwana w trakcie oczekiwania na spełnienie warunku startu
LC0_E_BROKEN_RUN	-8	funkcja przerwana w trakcie przetwarzania
LC0_E_BAD_LEN	-9	za duża liczba próbek
LC0_E_MOD_UNABLE	-10	moduł zajęty lub niezainicjowany

Ostrzeżenia (LC0_STATUS):

nazwa	kod	znaczenie
LC0_NON_EX_MOD	1	zażądano inicjalizacji nie istniejących modułów ale co najmniej 1 moduł został zainicjalizowany
LC0_OTHER_LEN	2	przepisano mniej próbek niż żądano
LC0_PREMATURE_END	3	przedwczesne zakończenie operacji z powodu przepełnienia / opróżnienia całego bufora
LC0_IN_PROGRESS	4	badana transmisja jeszcze trwa
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_wskaźnik_do_rekordu_zlecenia)***

Dobrze napisany program powinien składać się z następujących części:
- część wstępna:
- rozpoznanie konfiguracji modułu (GET_TOTAL_CONFIGURATION - ile i jakich modułów jest zainstalowanych, GET_MODULE_CONFIGURATION - czy moduł ma zainstalowany przetwornik c/a, czy jest podłączony do któregoś z przerwań itp., GET_INFO - jakie są minimalne okresy próbkowania, jakie są zakresy napięć przetworników a/c i c/a 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.
- instalacja procedury obsługi przerwania generowanego przez klawisz Ctrl-Break (jeżeli przewiduje się jego użycie)
- część wykonawcza: tu powinny się znaleźć funkcje wykonujące właściwe operacje modułu jak ANALOG_INPUT, DATA_TRANSMIT, ANALOG_OUTPUT, DIGITAL_INPUT, DIGITAL_OUTPUT itp.; należy zwrócić uwagę na dwie rzeczy:
- każda funkcja może być wywołana z błędnymi parametrami i zasygnalizować to w kodzie odpowiedzi (LC0_STATUS, LC0_ERR_STAT); należy koniecznie sprawdzać tę odpowiedź, szczególnie w dwóch sytuacjach: gdy program jest na etapie uruchamiania i gdy parametry funkcji są dostarczane interakcyjnie przez użytkownika
- moduł może ulec uszkodzeniu - wówczas niektóre funkcje nie mogą się zakończyć (przetwarzanie a/c, oczekiwanie na spełnienie warunków startu związanych z wejściami cyfrowymi czy sygnałem analogowym); poza tym funkcja może zostać wywołana z omyłkowo podanymi parametrami - należy - przewidując taką sytuację - albo umożliwić operatorowi przerwanie takiej funkcji przez naciśnięcie klawisza Ctrl-Break (i wykonanie w procedurze obsługi przerwania funkcji BREAK) albo samodzielnie odmierzać czas wykonania operacji i po przekroczeniu oszacowanego wcześniej limitu - automatycznie wykonywać funkcję BREAK; jest to istotne o tyle, że w przeciwnym razie operator zmuszony będzie powtórnie załadować system co może się wiązać ze stratą zmierzonych uprzednio - i być może unikatowych - danych
- część końcowa:
- przed zakończeniem programu należy koniecznie wykonać funkcję LEAVE_DRIVER; 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-LC.H zawierający definicje wszystkich struktur danych, stałych deklaracje funkcji potrzebnych do współpracy z driver'em oraz plik WSMP1016.C zawierający przykłady wykorzystania poszczególnych funkcji driver'a LC1016ADLL.
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-LC.H, natomiast w Dodatku C - wydruki plików: WSMP1016.C - program przykładowy C (API) z wczytaniem biblioteki poprzez plik definicji modułu;WSMP1016.DEF - plik definicji modułu dla WSMP1016.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-LC.PAS zawierający definicje wszystkich struktur danych i stałych potrzebnych do współpracy z driver'em, który znajduje się w dodatku C; plik WSMP1016.PAS - przykład wykorzystania poszczególnych funkcji driver'a; plik LC1016U.PAS - plik źródłowy dla modułu LC1016U.TPW, służącego do powiązania biblioteki LC1016A.DLL z aplikacjami PASCAL-a.

D O D A T E K A

d o d o k u m e n t a c j i b i b l i o t e k i D L L

m o d u ł u k o n t r o l n o - p o m i a r o w e g o

L C - 0 1 0 - 1 6 1 2

AMBEX-LC.H

struktury danych i stałe dla języka C

```
/*************************************************************************/
/*                                                                       */
/* definicje struktur danych i stalych dla wspolpracy z driver'ami modulow */
/* analogowych serii LC-... z poziomu jezyka C                           */
/*                                                                       */
/*************************************************************************/

/*************** REKORDY OPISOW ZLECEN FUNKCJI DRIVER'A ******************/

/************** struktury opisujace warunki startu i stopu ****************/
struct lc0_cond_level           /* warunek "poziom sygnalu cyfrowego"      */
        {
        unsigned char   mod_nr;         /* numer modulu                    */
        unsigned char   port_nr;        /* numer portu                     */
        unsigned char   inp_nr;         /* numer wejscia                   */
        unsigned char   value;          /* oczekiwana wartosc              */
        };

/*-----------------------------------------------------------------------*/
                                /* warunek "zbocze sygnalu cyfrowego"      */
struct lc0_cond_slope
        {
        unsigned char   mod_nr;         /* numer modulu                    */
        unsigned char   port_nr;        /* numer portu                     */
        unsigned char   inp_nr;         /* numer wejscia                   */
        unsigned char   slope;          /* rodzaj zbocza                   */
        };
                                /* kody rodzajow zbocza:                   */
#define LC0_SLOPE_COND_UP       1       /* narastajace                     */
#define LC0_SLOPE_COND_DOWN     0       /* opadajace                       */

/*-----------------------------------------------------------------------*/
struct lc0_cond_dig             /* warunek "kominacja sygnalow cyfrowych"  */
        {
        unsigned char   mod_nr;         /* numer modulu                    */
        unsigned char   port_nr;        /* numer portu                     */
        unsigned char   mask;           /* maska aktywnych wejsc           */
        unsigned char   pattern;        /* testowany wzorzec               */
        };


/*-----------------------------------------------------------------------*/
struct  lc0_cond_date           /* warunek "data"                          */
        {
        unsigned char   second;         /* sekunda                         */
        unsigned char   minute;         /* minuta                          */
        unsigned char   hour;           /* godzina                         */
        unsigned char   day;            /* dzien miesiaca                  */
        };

/*-----------------------------------------------------------------------*/
                                /* warunek "sygnal analogowy"              */
struct  lc0_cond_analog
        {
        unsigned char   mod_nr;         /* numer modulu                    */
        unsigned char   converter;      /* nr przetwornika                 */
        unsigned char   channel;        /* numer kanalu (b1..b7), kierunek */
                                        /*   przekroczenia (b8)            */
        int             level;          /* prog wyzwalania                 */
        };
                                /* kody kierunku przekroczenia progu:      */
#define LC0_ANALOG_COND_UP      0x80    /* w kierunku wiekszych wartosci   */
#define LC0_ANALOG_COND_DOWN    0       /* w kierunku mniejszych wartosci  */
#define LC0_ANALOG_COND_MASK    0x80    /* maska kodu kierunku             */

/*-----------------------------------------------------------------------*/
union lc0_start                 /* ========== warunki startu ============*/
        {
        struct  lc0_cond_level  level;          /* LC0_SLEVEL  */
        struct  lc0_cond_slope  slope;          /* LC0_SSLOPE  */
        struct  lc0_cond_dig    dig_eq;         /* LC0_SDIG_EQ */
        struct  lc0_cond_dig    dig_ne;         /* LC0_SDIG_NE */
        unsigned long           time;           /* LC0_STIME   */
        struct  lc0_cond_date   date;           /* LC0_SDATE   */
        struct  lc0_cond_analog analog;         /* LC0_SANALOG */
```

```
        };

union lc0_stop                  /* ========== warunki stopu ==============*/
        {
        unsigned long           samples;        /* LC0_ZSAMPLES */
        struct  lc0_cond_level  level;          /* LC0_ZLEVEL   */
        struct  lc0_cond_slope  slope;          /* LC0_ZSLOPE   */
        struct  lc0_cond_dig    dig_eq;         /* LC0_ZDIG_EQ  */
        struct  lc0_cond_dig    dig_ne;         /* LC0_ZDIG_NE  */
        unsigned long           time;           /* LC0_ZTIME    */
        struct  lc0_cond_date   date;           /* LC0_ZDATE    */
        struct  lc0_cond_analog analog;         /* LC0_ZANALOG  */
        };

/*=====================================================================*/

/* Rekord opisu zlecenia funkcji  MODULE_INIT ============================*/
struct lc0_init
        {
        unsigned char   LC0_CODE;       /* kod funkcji (16)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_IMODULE;    /* mapa modulow                    */
        };

/* Rekord opisu zlecenia funkcji  GET_TOTAL_CONFIGURATION ================*/
struct lc0_total
        {
        unsigned char   LC0_CODE;       /* kod funkcji (17)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_TONF;       /* konfiguracja modulow            */
        unsigned char   LC0_TIAD;       /* przetworniki a/c                */
        unsigned char   LC0_TIDA;       /* przetworniki c/a                */
        unsigned char   LC0_TCTC;       /* kanaly CTC                      */
        unsigned char   LC0_TIDI;       /* porty wejsc cyfrowych           */
        unsigned char   LC0_TIDO;       /* porty wyjsc cyfrowych           */
        };

/* Rekord opisu zlecenia funkcji   GET_MODULE_CONFIGURATION ==============*/
struct lc0_module
        {
        unsigned char   LC0_CODE;       /* kod funkcji (18)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_MMODULE;    /* numer modulu                    */
        unsigned int    LC0_MBASE1;     /* adres bazowy pakietu 1          */
        unsigned int    LC0_MBASE2;     /* adres bazowy pakietu 2          */
        unsigned char   LC0_MIAD;       /* przetworniki a/c                */
        unsigned char   LC0_MIDA;       /* przetworniki c/a                */
        unsigned char   LC0_MCTC;       /* kanaly CTC                      */
        unsigned char   LC0_MIDI;       /* porty wejsc cyfrowych           */
        unsigned char   LC0_MIDO;       /* porty wyjsc cyfrowych           */
        unsigned int    LC0_MCLOCK;     /* czestotliwosc zegara w kHz      */
        unsigned char   LC0_MINT;       /* numer przerwania (programowy)   */
       int far          *LC0_MMEMA ;   /* bufor w pamięci rozszerzonej    */
       unsigned long    LC0_MMEML; /*dlugosc bufora w pamieci rozszerzonej */
};

/* Rekord opisu zlecenia funkcji  GET_INFO ================================*/
struct lc0_info
        {
        unsigned char   LC0_CODE;       /* kod funkcji (19)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_GTYPE;      /* rodzaj urzadzenia               */
        unsigned char   LC0_GMODULE;    /* numer modulu                    */
        unsigned char   LC0_GNUM;       /* numer przetwornika/portu/       */
                                        /*                       ukladu CTC */
        unsigned char   LC0_GCHAN;      /* liczba kanalow                  */
        unsigned char   LC0_GRES;       /* liczba bitow przetwornika       */
        unsigned int    LC0_GTIME;      /* czas konwersji przetwornika w ns*/
        char            LC0_GMINV;      /* dolna granica zakresu napiec w  */
                                        /*   dziesiatych czesciach volta   */
        char            LC0_GMAXV;      /* gorna granica zakresu napiec w  */
```

```
                                        /*   dziesiatych czesciach volta   */
        unsigned char   LC0_GDMA;       /* numer kanalu DMA                */
        unsigned int    LC0_GMINP[32];  /* minimalne okresy probkowania    */
        };

/* Rekord opisu zlecenia funkcji  SET_CLOCK ================================*/
struct lc0_clock
        {
        unsigned char   LC0_CODE;       /* kod funkcji (20)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned int    LC0_LCLOCK;     /* czestotliowsc zegara w kHz      */
        };

/* Rekord opisu zlecenia funkcji  SET_VOLTAGE_RANGE ========================*/
struct lc0_volt
        {
        unsigned char   LC0_CODE;       /* kod funkcji (21)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_VTYPE;      /* rodzaj urzadzenia               */
        unsigned char   LC0_VMODULE;    /* numer modulu                    */
        unsigned char   LC0_VNUM;       /* numer przetwornika              */
        char            LC0_VMINV;      /* dolna granica zakresu napiec w  */
                                        /*   dziesiatych czesciach volta   */
        char            LC0_VMAXV;      /* gorna granica zakresu napiec w  */
                                        /*   dziesiatych czesciach volta   */
        };

/* Rekord opisu zlecenia funkcji  SET_TIME =================================*/
struct lc0_time
        {
        unsigned char   LC0_CODE;       /* kod funkcji (22)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned int    LC0_ETIME;      /* maksymalny czas obslugi         */
                                        /*   przerwania, ktore moze pojawic*/
                                        /*   sie w trakcie wykonywania     */
                                        /*   dlugiego pomiaru podawany w   */
                                        /*   mikrosekundach                */
        };

/* Rekord opisu zlecenia funkcji  WAIT_FOR_END ==============================*/
struct lc0_wait
        {
        unsigned char   LC0_CODE;       /* kod funkcji (23)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_WTYPE;      /* rodzaj urzadzenia               */
        unsigned char   LC0_WMODULE;    /* numer modulu                    */
        unsigned char   LC0_WNUM;       /* numer przetwornika              */
        unsigned char   LC0_WMODE;      /* tryb pracy                      */
        unsigned long   LC0_WRMNUM;     /* rzeczywista liczba probek       */
        unsigned int    LC0_WREMAR;     /* rzeczywista dlugosc marginesu   */
                                        /* koncowego                       */
        };
                                /* tryby pracy funkcji                     */
#define LC0_W_WAIT      0       /* oczekiwanie                             */
#define LC0_W_TEST      1       /* test konca                              */
#define LC0_W_FINISHED  2       /* powiadomienie o koncu                   */

/* Rekord opisu zlecenia funkcji  BREAK ====================================*/
struct lc0_break
        {
        unsigned char   LC0_CODE;       /* kod funkcji (24)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_BMODE;      /* tryb pracy                      */
        void (FAR *LC0_BPROC)(void);     /* adres procedury obslugi  */

        };
                                  /* bity trybu pracy funkcji              */
#define LC0_BREAK_EXEC          0 /* przerwanie                            */
#define LC0_BREAK_INST          1 /* instalacja Ctrl_Break...              */
#define LC0_BREAK_PREV          4 /* ...z wywolaniem poprzedniej obslugi   */
```

```
#define LC0_BREAK_UNINST        2 /* wyinstalowanie wszystkiego            */

/* Rekord opisu zlecenia funkcji  DIGITAL_INPUT ============================*/
struct lc0_digital_in
        {
        unsigned char   LC0_CODE;       /* kod funkcji (25)                  */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach    */
        unsigned char   LC0_DMODULE;    /* numer modulu                      */
        unsigned char   LC0_DNUM;       /* numer portu                       */
        unsigned char   LC0_DSTST;      /* typ warunku startu                */
        unsigned char   LC0_DVAL;       /* odczytana wartosc                 */
        union lc0_start LC0_DSTART;     /* parametry warunku startu          */
        };

/* Rekord opisu zlecenia funkcji  DIGITAL_OUTPUT ===========================*/
struct lc0_digital_out
        {
        unsigned char   LC0_CODE;       /* kod funkcji (26)                  */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach    */
        unsigned char   LC0_OMODULE;    /* numer modulu                      */
        unsigned char   LC0_ONUM;       /* numer portu                       */
        unsigned char   LC0_OSTST;      /* typ warunku startu                */
        unsigned char   LC0_OVAL;       /* wartosc do wyslania               */
        union lc0_start LC0_OSTART;     /* parametry warunku startu          */
        };

/* Rekord opisu zlecenia funkcji  CTC_WRITE ================================*/
struct lc0_ctc_write
        {
        unsigned char   LC0_CODE;       /* kod funkcji (27)                  */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach    */
        unsigned char   LC0_CMODULE;    /* numer modulu                      */
        unsigned char   LC0_CMODE;      /* tryb pracy funkcji                */
        unsigned char   LC0_CFUN;       /* tryb pracy kanalu                 */
        unsigned int    LC0_CVAL;       /* nowa wartosc licznika             */
        };
                                /* tryby pracy funkcji                       */
#define LC0_SET_CTC_MODE        1       /* zaprogramuj tryb pracy kanalu     */
#define LC0_SET_COUNTER_VALUE   2       /* zaladuj nowa wartosc licznika     */
#define LC0_CTC_ENABLE          4       /* zezwolenie CTC                    */
/* Rekord opisu zlecenia funkcji  CTC_READ =================================*/
struct lc0_ctc_read
        {
        unsigned char   LC0_CODE;       /* kod funkcji (28)                  */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach    */
        unsigned char   LC0_UMODULE;    /* numer modulu                      */
        unsigned char   LC0_UNUM;       /* numer kanalu                      */
        unsigned int    LC0_UVAL;       /* odczytana wartosc licznika        */
        };

/* Rekord opisu zlecenia funkcji  DATA_TRANSMIT ============================*/
struct lc0_transmit
        {
        unsigned char   LC0_CODE;       /* kod funkcji (29)                  */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach    */
       unsigned char    LC0_RMODULE ;   /* numer modułu */
        unsigned char   LC0_RMODE;      /* tryb przesylania                  */
        int far         *LC0_RADDR;     /* adres bufora w pam. podstawowej */
        unsigned long   LC0_RLEN;       /* dlugosc bufora w pam. podst.    */
        unsigned long   LC0_RMEAS;      /* numer pierwszej probki            */
        unsigned long   LC0_RNUM;       /* liczba probek do przeslania     */
       unsigned long   LC0_RRNUM;      /* rzeczywista liczba probek       */
        int far         *LC0_RMEMA;      /* adres bufora w pamieci          */
        };
                                /* tryby pracy funkcji                       */
#define LC0_TO_EXT_DIR          1       /* do pamieci rozszerzonej           */
#define LC0_FROM_EXT_DIR        0       /* z pamieci modulu / rozszerzonej */

/* Rekord opisu zlecenia funkcji ANALOG_INPUT ==============================*/
struct lc0_analog_in
```

```
        {
        unsigned char   LC0_CODE;       /* kod funkcji (30)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_AMODULE;    /* numer modulu                    */
        unsigned char   LC0_ANUM;       /* numer przetwornika              */
        unsigned int    LC0_AMODE;      /* tryb pracy funkcji              */
        unsigned char   LC0_ASTST;      /* typy warunkow startu i stopu    */
        unsigned long   LC0_APER;       /* okres probkowania               */
        unsigned int    LC0_APER2;      /* krotnosc okresu probkowania     */
        unsigned char   LC0_ACHAN;      /* liczba kanalow podstawowych     */
                                        /*   (b8 okresla czy praca         */
                                        /*   wielokanalowa (0) czy         */
                                        /*   jednokanalowa (1))            */
        unsigned char   LC0_ACHAN2;     /* liczba kanalow dodatkowych      */
        int far         *LC0_AADDR;     /* adres buforaw pam. podstawowej  */
        unsigned long   LC0_ALEN;       /* dlugosc bufora                  */
        int far         *LC0_AMEMA;      /* adres bufora w pam. ozszerzonej*/
       unsigned int    LC0_ABMAR;      /* dlugosc marginesu poczatkowego  */
        unsigned int    LC0_AEMAR;      /* dlugosc marginesu koncowego     */
        unsigned int    LC0_AHAND;      /* numer handler'a zbioru          */
        union lc0_start LC0_ASTART;     /* parametry warunku startu        */
        union lc0_stop  LC0_ASTOP;      /* parametry warunku stopu         */
        unsigned int    LC0_ARDIV1;     /* pierwszy podzielnik zegara      */
        unsigned int    LC0_ARDIV2;     /* drugi podzielnik zegara         */
        unsigned long   LC0_ARMNUM;     /* rzeczywista liczba probek       */
        unsigned int    LC0_ARBMAR;     /* rzeczywista dlugosc marg. pocz. */
        unsigned int    LC0_AREMAR;     /* rzeczywista dlugosc marg. konc. */
        unsigned long   LC0_ARLEN;      /* rzeczywista liczba przepisanych */
                                        /*   probek                        */
        unsigned int    LC0_ARBUF;      /* dla pracy z buforem cyklicznym: */
                                        /*   numer probki okreslajacej     */
                                        /*   poczatek bufora po zakonczeniu*/
                                        /*   pomiaru                       */
        };

/* Rekord opisu zlecenia funkcji  ANALOG_OUTPUT ============================*/
struct lc0_analog_out
        {
        unsigned char   LC0_CODE;       /* kod funkcji (31)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_NMODULE;    /* numer modulu                    */
        unsigned char   LC0_NNUM;       /* numer przetwornika              */
        unsigned int    LC0_NMODE;      /* tryb pracy funkcji              */
        unsigned char   LC0_NSTST;      /* typy warunkow startu i stopu    */
        unsigned char   LC0_NCHAN;      /* liczba kanalow (b8 okresla      */
                                        /*   czy praca wielokanalowa (0)   */
                                        /*   czy jednokanalowa (1))        */
        unsigned long   LC0_NPER;       /* okres probkowania */
        union   {
                int  far        *base_memory;
                int far          *extended_memory;
                }LC0_NADDR;             /* adres bufora (segment:offset dla*/
                                        /*   pamieci podstawowej, absolutny*/
                                        /*   dla pamieci rozszerzonej)     */
        unsigned long   LC0_NLEN;       /* dlugosc bufora w pam. podst.    */
        unsigned int    LC0_NHAND;      /* numer handler'a zbioru          */
        union lc0_start LC0_NSTART;     /* parametry warunku startu        */
        union lc0_stop  LC0_NSTOP;      /* parametry warunku stopu         */
        unsigned long   LC0_NRMNUM;     /* rzeczywista liczba wyslanych    */
                                        /*   probek                        */
        };

/* Rekord opisu zlecenia funkcji  LEAVE_DRIVER =============================*/
struct lc0_leave
        {
        unsigned char   LC0_CODE;       /* kod funkcji (32)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        };

/* Rekord opisu zlecenia funkcji  INTERRUPT_SERVICE ========================*/
struct lc0_interrupt
        {
```

```
        unsigned char   LC0_CODE;       /* kod funkcji (33)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        char            LC0_ERR_STAT;   /* dodatkowe informacje o bledach  */
        unsigned char   LC0_SMODULE;    /* numer modulu                    */
        void far (*LC0_SPROC)(void);    /* adres procedury obslugi         */
        int far         *LC0_SSTAT;     /* adres slowa komunikacyjnego     */
        };
/************* Rekord opisu zlecenia funkcji MEMORY_USE ******************/
struct lc0_memory
     {
       unsigned char LC0_CODE;          /* kod funkcji (34)              */
       char          LC0_STATUS;        /* kod odpowiedzi driver'a        */
       char          LC0_ERR_STAT;      /* dodatkowe informacje o błędach  */
       unsigned char LC0_MMODULE; /* nr modulu                          */
       unsigned char LC0_MMODE;   /* obecnie == 1                        */
       unsigned long LC0_MNUM;    /* zadana liczba próbek w nowym buforze */
       unsigned long LC0_MRNUM;   /* rzeczywista wielkosc bufora w probkach */
       int far *LC0_MRMEMA;             /* adres nowego bufora w pamieci  */
    };

/************* Rekord opisu zlecenia funkcji CONFIG         ***************/
struct lc0_config
    {
       unsigned char LC0_CODE;    /* kod funkcji (35)                */
       char          LC0_STATUS;  /* kod odpowiedzi driver'a         */
       char          LC0_ERR_STAT; /* dodatkowe informacje o bledach  */
       char          *LC0_INI_FILE;/* sciezka do pliku 'ambex.ini'    */
       };


#define LC0_IS_START    1       /* przerwanie wystapilo z powodu           */
                                /*   rozpoczecia pomiaru                   */
#define LC0_IS_ONE      2       /* przerwanie nastapilo z powodu zmierzenia*/
                                /*   kolejnej serii pomiarowej             */
#define LC0_IS_END_ADC  4       /* przerwanie nastapilo z powodu           */
                                /*   zakonczenia przetwarzania a/c         */
#define LC0_IS_END_DAC  8       /* przerwanie nastapilo z powodu           */
                                /*   zakonczenia przetwarzania c/a         */
#define LC0_IS_BROKEN   16      /* zakonczono przetwarzanie z powodu       */
                                /* wykonania funkcji BREAK                 */
#define LC0_IS_SAMPLE   32      /* przerwanie nastapilo z powodu zmierzenia*/
                                /*  kolejnej probki                        */

/* ***************** kody funkcji driver'a LC0_CODE ***********************/
/* ***************** indeksy funkcji w bibliotece DLL *********************/
#define MODULE_INIT                     16
#define GET_TOTAL_CONFIGURATION         17
#define GET_MODULE_CONFIGURATION        18
#define GET_INFO                        19
#define SET_CLOCK                       20
#define SET_VOLTAGE_RANGE               21
#define SET_TIME                        22
#define WAIT_FOR_END                    23
#define BREAK                           24
#define DIGITAL_INPUT                   25
#define DIGITAL_OUTPUT                  26
#define CTC_WRITE                       27
#define CTC_READ                        28
#define DATA_TRANSMIT                   29
#define ANALOG_INPUT                    30
#define ANALOG_OUTPUT                   31
#define LEAVE_DRIVER                    32
#define INTERRUPT_SERVICE               33
#define MEMORY_USE                      34
#define CONFIG                          35

/*************** numery przerwan obslugiwanych przez driver ***************/
#define LC010_16        0x99            /* LC-010-1612                     */
#define LC011_08        0x90            /* LC-011-0812                     */
#define LC011_16        0x91            /* LC-011-1612                     */
#define LC015_16        0x92            /* LC-015-1612                     */
#define LC020_08_0      0x93            /* LC-020-0812 v.0                 */
#define LC020_08_2      0x94            /* LC-020-0812 v.1 i v.2           */
#define LC020_32        0x95            /* LC-020-3212                     */
#define LC030_16        0x96            /* LC-030-1612                     */
```

```
#define LC060_06        0x97            /* LC-060-0612                      */

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

/* ======= ostrzezenia ======== */
#define LC0_NON_EX_MOD          1       /* nie istniejacy(e) modul(y)       */
#define LC0_OTHER_LEN           2       /* przepisano mniejsza liczbe       */
                                        /*   pomiarow                       */
#define LC0_PREMATURE_END       3       /* przedwczesne zakonczenie z       */
                                        /*   powodu przepelnienia bufora    */
#define LC0_IN_PROGRESS         4       /* badana transmisja jeszcze trwa   */
#define LC0_IS_INIT             5      /* zadano inicjacji zainicjowanego  */
                                       /* modulu                          */


/* ========== bledy ========== */
#define LC0_UNKN_FUNC           -1      /* nieznany kod funkcji             */
#define LC0_NO_MODULE           -2      /* brak modulu(ow)                  */
#define LC0_BAD_DEV_TYP         -3      /* bledny typ urzadzenie            */
#define LC0_NONEX_DEV           -4      /* nie istnieje urzadzenie o tym    */
                                        /*   numerze                        */
#define LC0_BAD_FREQ            -5      /* zla czestotliwosc zegara         */
#define LC0_BAD_RANGE           -6      /* zly zakres napiec                */
#define LC0_NO_OPER             -7      /* zadna operacja nie jest          */
                                        /*   wykonywana                     */
#define LC0_BAD_MARGIN          -8      /* bledna dlugosc marginesu         */
                                        /*   poczatkowego                   */
#define LC0_BAD_BUF_ADR         -9      /* bledny adres bufora              */
#define LC0_BAD_BUF_LEN         -10     /* bledna dlugosc bufora            */
#define LC0_DEV_BUSY            -11     /* urzadzenie jest zajete           */
#define LC0_BAD_PER             -12     /* zly okres probkowania            */
#define LC0_BAD_CHAN_N          -13     /* zla liczba kanalow               */
#define LC0_BAD_CHAN            -14     /* numer nie istniejacego kanalu    */
#define LC0_BROKEN              -15     /* przerwano funkcja BREAK          */
#define LC0_INTR_NOT_INST       -16     /* procedura obslugi przerwania nie*/
                                        /*   jest zainstalowana             */
#define LC0_ILL_START_CODE      -17     /* nielegalny typ warunku startu    */
#define LC0_ILL_STOP_CODE       -18     /* nielegalny typ warunku stopu     */
#define LC0_BAD_PROC            -19     /* bledny adres procedury obslugi   */
                                        /*   przerwania lub slowa           */
                                        /*   komunikacyjnego                */
#define LC0_TOO_LONG_MARG       -20     /* margines dluzszy od bufora       */
#define LC0_ILL_START           -21     /* bledne parametry warunku startu */
#define LC0_ILL_STOP            -22     /* bledne parametry warunku stopu   */
#define LC0_BAD_MNUM            -23     /* bledny numer pierwszej probki    */
#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_PARAMS           -26     /* nie podano parametrow            */
                                        /*   przetwarzania                  */
#define LC0_OVERRUN             -27     /* zakonczono przetwarzanie z       */
                                        /*   powodu bledu OVERRUN           */
#define LC0_NO_DMA              -28     /* z danym urzadzeniem nie jest     */
                                        /*   zwiazany zaden kanal DMA       */
#define LC0_NO_IRQ              -29     /* z danym modulem nie jest         */
                                        /*   zwiazane zadne przerwanie      */
#define LC0_NOT_FULLY_SUP       -30     /* zadany tryb wykonania funkcji    */
                                        /*   nie jest realizowany dla       */
                                        /*   danego typu modulu lub funkcja*/
                                        /*   w opracowaniu                  */
#define LC0_NO_EXTMEM           -31     /* brak pamieci roszerzonej         */
#define LC0_NO_SEC_FREQ         -32     /* modul nie moze wykonywac         */
                                        /*   pomiarow z podwojna            */
                                        /*   czestotliwoscia                */
#define LC0_INTR_INST           -33     /* procedura obslugi przerwania     */
                                        /*   juz zainstalowana              */
#define LC0_BAD_PER2            -34     /* bledna wielokrotnosc okresu      */
                                        /*   probkowania (0 lub 1)          */
#define LC0_BAD_MODE            -35     /* bledny tryb pracy                */
#define LC0_BAD_EXTMEM          -36     /* zly adres bufora w pamieci       */
                                        /*   rozszerzonej                   */
#define LC0_CTC_NOT_PROGRAMMED  -37     /* zapis licznika przy niezapro-    */
                                        /*   gramowanym trybie pracy        */
#define LC0_REJECTED            -38     /* za wiele rownoleglych wejsc do   */
```

```
#define LC0_BAD_CONFIG          -39    /*blad w pliku konfiguracyjnym *.ini*/
#define LC0_NOT_INIT            -40    /* modul nie zainicjowany   */
#define LC0_NO_DMA_TRANS        -41    /* brak mozliwości przeprowadzenia  */
                                        /* transferow DMA                */

/************************** dodatkowe informacje o bledach ****************/
#define LC0_E_OK                 0       /* brak dodatkowych informacji     */
#define LC0_E_NO_MODULE         -1       /* nie ma takiego modulu           */
#define LC0_E_NONEX_DEV         -2       /* nie istnieje urzadzenie o tym   */
                                         /*   numerze                       */
#define LC0_E_BAD_CHAN          -3       /* numer nieistniejacego kanalu    */
#define LC0_E_BAD_TIME          -4       /* zly odcinek czasu               */
#define LC0_E_BAD_DATE          -5       /* zla specyfikacja daty           */
#define LC0_E_BAD_THRE          -6       /* bledny prog wyzwalania          */
                                         /*   analogowego                   */
#define LC0_E_BROKEN_WAIT       -7       /* funkcja przerwana w trakcie     */
                                         /*   oczekiwania na spelnienie     */
                                         /*   warunku startu                */
#define LC0_E_BROKEN_RUN        -8       /* funkcja przerwana w trakcie     */
                                         /*   przetwarzania                 */
#define LC0_E_BAD_LEN           -9       /* zadeklarowano za duzo probek    */

#define LC0_E_MOD_UNABLE       -10   /* modul zajety lub nie zainicjowany */

/************************** kody warunkow startu ***************************/
#define LC0_SIMMED      0       /* natychmiastowy                          */
#define LC0_SHARD       1       /* od sygnalu sprzetowego                  */
#define LC0_SLEVEL      2       /* od poziomu sygnalu cyfrowego            */
#define LC0_SSLOPE      3       /* od zbocza sygnalu cyfrowego             */
#define LC0_SDIG_EQ     4       /* od kombinacji bitow - rowne             */
#define LC0_SDIG_NE     5       /* od kombinacji bitow - rozne             */
#define LC0_STIME       6       /* po uplynieciu okreslonego czasu         */
#define LC0_SDATE       7       /* o podanym czasie                        */
#define LC0_SANALOG     8       /* od sygnalu analogowego                  */

/************************** kody warunkow stopu ****************************/
#define LC0_ZSAMPLES    0x00    /* po zmierzeniu okreslonej liczby probek  */
#define LC0_ZBREAK      0x10    /* po wykonaniu funkcji BREAK              */
#define LC0_ZLEVEL      0x20    /* od poziomu sygnalu cyfrowego            */
#define LC0_ZSLOPE      0x30    /* od zbocza sygnalu cyfrowego             */
#define LC0_ZDIG_EQ     0x40    /* od kombinacji bitow - rowne             */
#define LC0_ZDIG_NE     0x50    /* od kombinacji bitow - rozne             */
#define LC0_ZTIME       0x60    /* po uplynieciu okreslonego czasu         */
#define LC0_ZDATE       0x70    /* o podanym czasie                        */
#define LC0_ZANALOG     0x80    /* od sygnalu analogowego                  */

/************************** kody numerow modulow ***************************/
#define LC0_MODA        1       /* modul A                                 */
#define LC0_MODB        2       /* modul B                                 */
#define LC0_MODC        3       /* modul C                                 */
#define LC0_MODD        4       /* modul D                                 */

/************************** maski modulow w mapie **************************/
#define LC0_MODAMAP     1       /* modul A                                 */
#define LC0_MODBMAP     2       /* modul B                                 */
#define LC0_MODCMAP     4       /* modul C                                 */
#define LC0_MODDMAP     8       /* modul D                                 */

/************************** kody typow urzadzen ****************************/
#define LC0_DINPUT      1       /* wejsciowy port cyfrowy                  */
#define LC0_DOUTPUT     2       /* wyjsciowy port cyfrowy                  */
#define LC0_AINPUT      3       /* przetwornik a/c                         */
#define LC0_AOUTPUT     4       /* przetwornik c/a                         */
#define LC0_CTC         5       /* kanal CTC                               */

/************ maski trybu pracy funkcji ANALOG_INPUT i ANALOG_OUTPUT ********/
#define LC0_MOD_START           1       /* start przetwarzania             */
#define LC0_MOD_NEW_PAR         2       /* nowe parametry                  */
#define LC0_MOD_SYNCHR          4       /* praca synchroniczna             */
#define LC0_MOD_ASYNCHR         0       /* praca asynchroniczna            */
#define LC0_MOD_INTR            8       /* praca bez przerwan              */
#define LC0_MOD_INTR_TYPE       16      /* bit trybu przerwan              */
#define LC0_MOD_END_INTR        0       /* przerwanie po koncu             */
                                        /*   przetwarzania                 */
#define LC0_MOD_ONE_INTR        16      /* przerwanie po kazdej probce     */
```

```
#define LC0_MOD_BLOCK           32      /* praca blokowa                    */
#define LC0_MOD_SINGLE          0       /* praca pojedyncza                 */
#define LC0_MOD_CYCL            64      /* bufor cykliczny                  */
#define LC0_MOD_FILE_W          128     /* zapis do pliku                   */
#define LC0_MOD_FILE_R          128     /* odczyt z pliku                   */
#define LC0_MOD_MEM_W           256     /* przepisanie do pamieci           */
#define LC0_MOD_EXT_CLK         512     /* zegar zewnetrzny                 */
#define LC0_MOD_EXT_MEM         1024    /* pamiec rozszerzona               */
#define LC0_MOD_RAM_SEK          2048    /* praca z sekwencja pomiarowa     */
#define LC0_MOD_PAGE            4096    /* blokada przy progr. strony       */

/************* wartosci fragmentow bajtu kontrolnego 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                                */
/*****************************************************************************/

/* MODULE_INIT ; nr 16 */
void FAR PASCAL  _export LC0_ModuleInit(struct lc0_init FAR *param);

/* GET_TOTAL_CONFIGURATION ; nr 17 */
          void FAR PASCAL  _export LC0_GetTotalConf(struct lc0_total FAR *param);

/* GET_MODULE_CONFIGURATION ; nr 18 */
void FAR PASCAL  _export LC0_GetModule(struct lc0_module  FAR *param)    ;

/* GET_INFO ; nr 19 */
void FAR PASCAL  _export LC0_GetInfo(struct lc0_info FAR *param)         ;

/* SET_CLOCK ; nr 20 */
void FAR PASCAL  _export LC0_SetClock(struct lc0_clock FAR *param) ;

/* SET_VOLTAGE_RANGE ; nr 21 */
void FAR PASCAL  _export LC0_SetVoltage(struct lc0_volt FAR *param)      ;

/* SET_TIME ; nr 22 */
void FAR PASCAL  _export LC0_SetTime(struct lc0_time FAR *param)   ;

/* WAIT_FOR_END ; nr 23 */
void FAR PASCAL  _export LC0_Wait(struct lc0_wait FAR *param)     ;

/* BREAK ; nr 24 */
void FAR PASCAL  _export LC0_Break(struct lc0_break FAR *param);

/* DIGITAL_INPUT ; nr 25 */
void FAR PASCAL  _export LC0_DigitalIn(struct lc0_digital_in FAR *param);

/* DIGITAL_OUTPUT ; nr 26 */
void FAR PASCAL  _export LC0_DigitalOut(struct lc0_digital_out FAR *param);

/* CTC_WRITE ; nr 27 */
void FAR PASCAL  _export LC0_CTCWrite(struct lc0_ctc_write FAR *param)    ;

/* CTC_READ ; nr 28 */
void FAR PASCAL  _export LC0_CTCRead(struct lc0_ctc_read FAR *param)     ;

/* DATA_TRANSMIT ; nr 29 */
```

```
void FAR PASCAL  _export LC0_DataTransmit(struct lc0_transmit FAR *param);

/* ANALOG_INPUT ; nr 30 */
void FAR PASCAL  _export LC0_AnalogIn(struct lc0_analog_in FAR *param);

/* ANALOG_OUTPUT ; nr 31 */
void FAR PASCAL  _export LC0_AnalogOut(struct lc0_analog_out FAR *param);

/* LEAVE_DRIVER ; nr 32 */
void FAR PASCAL  _export LC0_Leave(struct lc0_leave FAR *param)    ;

/* INTERRUPT_SERVICE ; nr 33 */
void FAR PASCAL  _export LC0_Interrupt(struct lc0_interrupt FAR *param)   ;

/* MEMORY_USE ; nr 34 */
void FAR PASCAL  _export LC0_MemoryUse(struct lc0_memory FAR *param);
/* CONFIG ; nr 35 */
void FAR PASCAL  _export LC0_Config(struct lc0_config FAR *param)  ;
```

D O D A T E K B

d o d o k u m e n t a c j i b i b l i o t e k i D L L

m o d u ł u k o n t r o l n o - p o m i a r o w e g o

L C - 0 1 0 - 1 6 1 2

WSMP1016.C
program przykładowy w C

```
/* WSMP1016.C ********************************************************************
        Program przykładowy pokazujący sposób wykorzystania drivera modułów
        serii LC   firmy AMBEX.
        Sposób dołaczenia biblioteki DLL : dynamicznie                              */
/*********************************************************************************
 Program korzysta ze standardowych funkcji API Windows 3.1 .
 W katalogu zawierajacym standardowe naglowki C muszą sie znajdowac pliki:
 AMBEX-LC.H  - plik nagłówkowy drivera ,
 RESOURCE.H  - plik nagłówkowy  zasobów programu,
 WSMP1016.RC - plik zasobów     ,
 WSMP1016.DEF - plik definicyjny modułu

 Pliki :
 - AMBEX.INI  - plik konfiguracyjny modułów serii LC ,
 - LC1016A.DLL - driver (biblioteka DLL) do kart LC-010-1612 ,
 powinny znajdować się w jednym z katalogów ze standardowej ścieżki przeszu-
 kiwań Windows (według kolejności przeszukiwania) :
 1. katalog systemowy WINDOWS ,
 2. katalog WINDOWS\SYSTEM ,
 3. bieżący katalog aplikacji ,
 4. katalog znajdujący się na ścieżce przeszukiwań PATH .

 Opcje kompilatora (Borland C++)
 - data alingment  : byte ,
 - memory model   : far ,
 - entry/exit code : windows smart callbacks .

 */
/*********************************************************************************/
#include <windows.h>            /* funkcje API Windows    */
#include <stdio.h>              /* wyswietlanie           */
#include <string.h>             /* funkcja strcat         */
#include <stdlib.h>             /* funkcja exit               */
#include <alloc.h>
#include <dos.h>                /* FP_SEG MK_FP */
#include "ambex-lc.h"           /* struktury danych , definicje stalych 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 ***********************************/
BOOL   loaddriver(void);
void   askdriver(void) ;
void   blocktransmission(void);
int    displaybuf(int *buf, int c, int s,HDC,RECT);
void   displayconfiguration(HDC,RECT);
void    PressOrClickAny(HDC,RECT);
int    displaytext(HDC,RECT,unsigned char);
void   drivererror(char status, char err_stat);
void   failblocktransmission(void);
void   installbreak(void);
void   interruptedbefore(void);
void   breakproces(void);
void   quit(void);
void   singletransmission(void);
```

```
void      transmission(unsigned char start);
void      singlewrite(void);
void FAR  my_break(void);
void drawscreen(void);
void GoToLine(RECT * , HDC ) ;


#define MY_MESSAGE WM_USER +1

/******************** definicje stalych ***********************************/
#define  TRUE             1
#define  FALSE            0
#define  BUFLEN           (8 * 20)          /* dlugosc bufora        */
#define  SAMPLES          20                /* liczba probek         */
#define  CHANNELS         8                 /* liczba kanalow        */

/******************** deklaracje zmiennych ********************************/

int      LCinterrupt;                       /* numer przerwania driver'a          */
                                            /* rekordy opisu poszczegolnych zlecen    */
struct   lc0_init                           s_init       = {MODULE_INIT};
struct   lc0_total                          s_total      = {GET_TOTAL_CONFIGURATION};
struct   lc0_module                         s_module     = {GET_MODULE_CONFIGURATION};
struct   lc0_info                           s_info       = {GET_INFO};
struct   lc0_break                          s_break      = {BREAK};
struct   lc0_analog_in                      s_analog_in  = {ANALOG_INPUT};
struct   lc0_analog_out                     s_analog_out = {ANALOG_OUTPUT};
struct   lc0_leave                          s_leave      = {LEAVE_DRIVER};
struct   lc0_interrupt                      s_interr  = {INTERRUPT_SERVICE};
int      buf[BUFLEN];                       /* bufor pomiarowy */
int      modulenum;                         /* numer badanego modulu              */
int      koniec = 0;
int      bufnum = 0;
int      huge *buf1;
int      huge *buf2;
int      sstat;                             /* slowo komunikacyjne dla funkcji obslugi */
int   paintnum = 0 ;                        /* nr sekcji do wyświetlenia w komunikacie WM_PAINT*/
int   paintdet = 0 ;
int   enddet      = 0 ;
unsigned char tstart ;
HWND     hWnd;                              /* uchwyt do głównego okna programu */
HANDLE   hInst;                             /* deskryptor bieżącego zadania*/
HANDLE   hDriverInst     ;                  /* desktyptor drivera                 */

/* deklaracje wskaźników do funkcji publicznych drivera                        */

void (FAR PASCAL _export *ModuleInit)(struct lc0_init FAR *param);
void (FAR PASCAL _export *GetTotalConfiguration)(struct lc0_total FAR *param);
void (FAR PASCAL _export *GetModuleConfiguration)(struct lc0_module FAR *param);
void (FAR PASCAL _export *GetInfo)(struct lc0_info FAR *param);
void (FAR PASCAL _export *Break)(struct lc0_break FAR *param);
void (FAR PASCAL _export *AnalogInput)(struct lc0_analog_in FAR *param);
void (FAR PASCAL _export *AnalogOutput)(struct lc0_analog_out FAR *param);
void (FAR PASCAL _export *Leave)(struct lc0_leave FAR *param);

                                            /* ------- teksty błędów              */
char     *DriverErrors[] =
{
                                                    /* LC0_UNKN_FUNC     -1 */
"LC0_UNKN_FUNC: Nieznany kod funkcji",
                                                    /* LC0_NO_MODULE     -2 */
```

```
"LC0_NO_MODULE: Błędny numer modułu",
                                        /* LC0_BAD_DEV_TYP  -3 */
"LC0_BAD_DEV_TYP: Brak urządzeń danego typu",
                                        /* LC0_NONEX_DEV    -4 */
"LC0_NONEX_DEV: Błędny numer urządzenia",
                                        /* LC0_BAD_FREQ     -5 */
"LC0_BAD_FREQ: Błędny okres",
                                        /* LC0_BAD_RANGE    -6 */
"LC0_BAD_RANGE: Błędny zakres napięc",
                                        /* LC0_NO_OPER      -7 */
"LC0_NO_OPER: Brak operacji w toku",
                                        /* LC0_BAD_MARGIN   -8 */
"LC0_BAD_MARGIN: Błędna długość marginesu początkowego",
                                        /* LC0_BAD_BUF_ADR  -9 */
"LC0_BAD_BUF_ADR: Błędny adres bufora",
                                        /* LC0_BAD_BUF_LEN  -10 */
"LC0_BAD_BUF_LEN: Błędna dlugość bufora",
                                        /* LC0_DEV_BUSY     -11 */
"LC0_DEV_BUSY: Urządzenie zajęte",
                                        /* LC0_BAD_PER      -12 */
"LC0_BAD_PER: Za krótki okres próbkowania",
                                        /* LC0_BAD_CHAN_N   -13 */
"LC0_BAD_CHAN_N: Błędna liczba kanałów",
                                        /* LC0_BAD_CHAN     -14 */
"LC0_BAD_CHAN: Błędny numer kanału",
                                        /* LC0_BROKEN       -15 */
"LC0_BROKEN: Przetwarzanie przerwane funkcją BREAK",
                                        /* LC0_INTR_NOT_INST  -16 */
"LC0_INTR_INST: Procedura obsługi przerwania nie jest zainstalowana",
                                        /* LC0_ILL_START_CODE -17 */
"LC0_ILL_START_CODE: Nielegalny sposob startu",
                                        /* LC0_ILL_STOP_CODE  -18 */
"LC0_ILL_STOP_CODE: Nielegalny sposob stopu",
                                        /* LC0_BAD_PROC     -19 */
"LC0_BAD_PROC: Błędny adres procedury obsługi przerwania",
                                        /* LC0_TOO_LONG_MARG  -20 */
"LC0_TOO_LONG_MARG: Za długi margines początkowy",
                                        /* LC0_ILL_START    -21 */
"LC0_ILL_START: Błędne parametry warunku startu",
                                        /* LC0_ILL_STOP     -22 */
"LC0_ILL_STOP: Błędne parametry warunku stopu",
                                        /* LC0_BAD_MNUM     -23 */
"LC0_BAD_MNUM: Błędny numer pierwszej probki",
                                        /* 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_PARAMS    -26 */
"LC0_NO_PARAMS: Nie podano parametrów przetwarzania",
                                        /* LC0_OVERRUN      -27 */
"LC0_OVERRUN: Błąd OVERRUN",
                                        /* LC0_NO_DMA       -28 */
"LC0_NO_DMA: Urządzenie nie jest podłączone do DMA",
                                        /* 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_NO_EXTMEM    -31 */
"LC0_NO_EXTMEM: Brak pamięci dodatkowej",
                                        /* LC0_NO_SEC_FREQ   -32 */
"LC0_NO_SEC_FREQ: Moduł ma tylko jedną czestotliwość",
```

```
                                        /* LC0_INTR_INST      -33 */
"LC0_INTR_INST: Procedura obsługi przerwania już zainstalowana",
                                        /* LC0_BAD_PER2       -34 */
"LC0_BAD_PER2: Błędna wielokrotność okresu próbkowania",
                                        /* LC0_BAD_MODE       -35 */
"LC0_BAD_MODE: Błędny tryb pracy",
                                        /* LC0_BAD_EXTMEM     -36 */
"LC0_BAD_EXTMEM: Błędny adres bufora w pamięci rozszerzonej",
                                        /* LC0_NOT_PROGRAMMED -37 */
"LC0_CTC_NOT_PROGRAMMED: Zapis licznika przy niezaprogramowanym trybie pracy",
                                        /* LC0_REJECTED       -38 */
"LC0_REJECTED: Za dużo jednoczesnych odwołań do driver'a",
                                        /* LC0_BAD_CONFIG     -39 */
"LC0_BAD_CONFIG: Błędny plik konfiguracyjny ",
                                        /* LC0_NOT_INIT       -40 */
"LC0_NOT_INIT: Moduł nie był zainicjowany" ,
                                        /* LC0_NO_DMA_TRANS   -41 */
"LC0_NO_DMA_TRANS: Nie jest możliwy blokowy transfer DMA" ,
};

                                /* ------ teksty ostrzezen       */
char    *DriverWarnings[] =
{
                                        /* LC0_NON_EX_MOD   1 */
"LC0_NON_EX_MOD: Zazadano inicjalizacji nieistniejacych modulow",
                                        /* LC0_OTHER_LEN    2 */
"LC0_OTHER_LEN: Przepisano mniej probek niz zazadano",
                                        /* LC0_PREMATURE_END  3 */
"LC0_PREMATURE_END: Zakonczenie operacji z powodu przepelnienia bufora",
                                        /* LC0_IN_PROGRESS  4 */
"LC0_IN_PROGRESS: Badana transmisja jeszcze trwa",
                                        /* LC0_IS_INIT      5 */
"LC0_IS_INIT: Moduł jest już zainicjowany ",

};

                                /* ------- teksty informacji dodatkowych  */
char    *DriverAdditionalErrors[] =
{
                                        /* LC0_E_NO_MODULE -1 */
"LC0_E_NO_MODULE: Nie ma takiego modulu",
                                        /* LC0_E_NONEX_DEV -2 */
"LC0_E_NONEX_DEV: Nie istnieje urzadzenie o tym numerze",
                                        /* LC0_E_BAD_CHAN  -3 */
"LC0_E_BAD_CHAN: Numer nieistniejacego kanalu",
                                        /* LC0_E_BAD_TIME  -4 */
"LC0_E_BAD_TIME: Zly odcinek czasu",
                                        /* LC0_E_BAD_DATE  -5 */
"LC0_E_BAD_DATE: Zla specyfikacja daty",
                                        /* LC0_E_BAD_THRE  -6 */
"LC0_E_BAD_THRE: Bledny prog wyzwalania analogowego",
                                        /* LC0_E_BROKEN_WAIT  -7 */
"LC0_E_BROKEN_WAIT: Przerwanie w trakcie oczekiwania na warunek startu",
                                        /* LC0_E_BROKEN_RUN   -8 */
"LC0_E_BROKEN_RUN: Przerwanie w trakcie przetwarzania",
                                        /* LC0_E_BAD_LEN    -9 */
"LC0_E_BAD_LEN: Zadeklarowano za duzo probek",
                                        /* LC0_E_MOD_UNABLE   -10 */
"LC0_E_MOD_UNABLE: Moduł zajęty lub nie zainicjowany ",
};
```

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

        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;
        BOOL dresult ;
        if (!hPrevInstance)
         {
          if (!InitApplication(hInstance))                      /* rejestracja klasy okna */
                return (FALSE);
         }
        if (!InitInstance(hInstance, nCmdShow))     /* aktywacja okna */
                return (FALSE);
        /* wczytanie drivera */
        dresult = loaddriver();
        if(dresult)
         {
          MessageBox(hWnd,"Zamknij by rozpocząć przykładową sesję pomiarową","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  */
        if(dresult)
         {
          quit();
          FreeLibrary(hDriverInst) ;
         }
        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 głownwgo */
        hWnd = CreateWindow(
                "DemoDriverWClass",
                "Przykład użycia biblioteki DLL Windows do kart AMBEX serii LC",
                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);
        return (TRUE);
}
/*************************************************************************

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

        Przeznaczenie:   Funkcja odpowiedzi 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

*************************************************************************/
long FAR PASCAL __export MainWndProc(HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam)
{
 HDC dc ;
 RECT rect ;
 PAINTSTRUCT ps ;
 switch (message)
        {
         case WM_COMMAND:                                        /* komenda menu */
          switch (wParam)
                {
                 case CM_POMIAR :       /* powtórzenie pomiarów */
                  if(paintnum==0)
                        {
                         paintnum = 1;
                         PostMessage(hWnd,MY_MESSAGE,0,0);
                        }
                  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   :
                 case 6   :
                  rect.top=displaytext(dc,rect,tstart);
                  rect.left = 10 ;
                  if(enddet>0)
                        {
                         if(enddet == 2)
                          {
                                rect.top = displaybuf(buf,
                                CHANNELS, SAMPLES,dc,rect);
                          }
                         PressOrClickAny(dc,rect);
                        }
                  break ;
                }
          EndPaint(hWnd,&ps);
          break ;
         case WM_LBUTTONDOWN :     /* kliknięcie myszą w oknie */
         case WM_KEYDOWN         :                  /* naciśnięcie klawisza */
          if(paintnum!=0)
                {
```

```
                paintnum++ ;
                if(paintnum==7)
                 {
                        paintnum = 0 ; /*koniec*/
                        drawscreen();
                 }
                else
                 PostMessage(hWnd,MY_MESSAGE,0,0);
               }
         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                */
                 installbreak();   /* zainstalowanie procedury obslugi   */
                                  /* przerwania generowanego przez Ctrl-Break*/
                 break ;
                case 2    :
                 blocktransmission();       /* wykonanie transmisji blokowej +*/
                                            /* wyswietlenie zmierzonych wartosci*/
                 break ;
                case 3    :
                 failblocktransmission();   /* wykonanie blednej                */
                                            /* transmisji blokowej   */
                 break ;
                case 4    :
                 interruptedbefore();
                                            /*wykonanie transmisji blokowej przerwanej*/
                                             /* przez operatora przed startem */
                 break ;
                case 5 :
                 singletransmission();      /* wykonanie pomiaru w */
                                            /* trybie pojedynczym  */
                 break ;
                case 6    :
                 singlewrite();             /* zapis wartosci na przetwornik CA  */
                 break ;
                }
         break ;
        case WM_DESTROY:              /* koniec programu */
         PostQuitMessage(0);
         break;
        default:
         return (DefWindowProc(hWnd, message, wParam, lParam));
        }
 return (NULL);
}

/*****************************************************************************
 Funkcja : loaddriver

 Przeznaczenie : Dynamiczne wczytanie biblioteki 'lc1116a.dll' , ustawienie
                 wskaźników do funkcji publicznych używanych w programie .

 Rezultat        : != 0 , gdy wszystko O.K
```

```
*************************************************************************/
BOOL loaddriver(void)
{
 FARPROC funptr ;
 hDriverInst = LoadLibrary("lc1016a.dll");   /* wczytanie do pamięci */
 /* ustawienie wskaźników do funkcji */
 if(hDriverInst > HINSTANCE_ERROR)
  {
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(MODULE_INIT));
        (FARPROC)ModuleInit = MakeProcInstance(funptr,hInst);
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(GET_TOTAL_CONFIGURATION));
        (FARPROC)GetTotalConfiguration = MakeProcInstance(funptr,hInst);
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(GET_MODULE_CONFIGURATION));
        (FARPROC)GetModuleConfiguration = MakeProcInstance(funptr,hInst);
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(GET_INFO));
        (FARPROC)GetInfo = MakeProcInstance(funptr,hInst);
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(BREAK));
        (FARPROC)Break = MakeProcInstance(funptr,hInst);
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(ANALOG_INPUT));
        (FARPROC)AnalogInput = MakeProcInstance(funptr,hInst);
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(ANALOG_OUTPUT));
        (FARPROC)AnalogOutput = MakeProcInstance(funptr,hInst);
        funptr =GetProcAddress(hDriverInst,MAKEINTRESOURCE(LEAVE_DRIVER));
        (FARPROC)Leave = MakeProcInstance(funptr,hInst);
        return(TRUE);
  }
 else
  {
        MessageBeep(-1);
        MessageBox(hWnd,"Błąd przy ładowaniu drivera do pamięci", "Informacja",MB_OK);
        DestroyWindow(hWnd);
        return(FALSE);
  }
}

/*************************************************************************
 Funkcja quit()

 Przeznaczenie : Wykonanie funkcji LEAVE_DRIVER

*************************************************************************/
void quit()
{
  (*Leave)((struct lc0_leave FAR * )&s_leave);
}

/*************************************************************************
Funkcja : askdriver
Przeznaczenie:
  Rozpoznanie konfiguracji badanego modulu.
 Sposob:
Przez wykorzystanie funkcji driver'a GET_TOTAL_CONFIGURATION,
  GET_MODULE_CONFIGURATION, GET_INFO.
 Uwagi:
  Funkcja nadaje wartosc zmiennej modulenum (numer badanego modulu)     wypelnia struktury total, module, info i
inicjalizuje zainstalowane       moduly.
*************************************************************************/
void askdriver(void)
{
        /* GET_TOTAL_CONFIGURATION */
```

```
        (*GetTotalConfiguration)((struct lc0_total FAR * )&s_total);
        /* inicjalizacja zainstalowanych modulow   */
        s_init.LC0_IMODULE = s_total.LC0_TONF & 0xF;
        /* MODULE_INIT     */
        (*ModuleInit)((struct lc0_init FAR * )&s_init);
                                        /* sprawdzenie, ktory modul jest   */
                                        /*   zainstalowany: A, B, C czy D  */
        for(modulenum = 1; modulenum <= 4; modulenum++)
                if(s_total.LC0_TONF & (1 << (modulenum - 1)))
                break;
                                        /* spytanie o konfiguracje modulu  */
        s_module.LC0_MMODULE = modulenum;
        /* GET_MODULE_CONFIGURATION*/
        (*GetModuleConfiguration)((struct lc0_module FAR * )&s_module);
                                        /* spytanie o konfiguracje toru a/c*/
        s_info.LC0_GMODULE = modulenum;
        s_info.LC0_GTYPE   = LC0_AINPUT;
        s_info.LC0_GNUM    = 1;
        /* GET_INFO        */
        (*GetInfo)((struct lc0_info FAR * )&s_info);
}

/*****************************************************************************
Funkcja : displayconfiguration(HDC,RECT)
 Przeznaczenie:
 Wyswietlenie 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[10];
        rect.left +=10 ;
        DrawText(dc,"Konfiguracja modułu:",-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc); GoToLine(&rect,dc);
        sprintf(line,"Adres modułu: %04X (hex)", s_module.LC0_MBASE1);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Liczba przetworników a/c: %d", s_module.LC0_MIAD);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Liczba przetworników c/a: %d", s_module.LC0_MIDA);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Liczba portów cyfrowych wejściowych: %d",
                s_module.LC0_MIDI);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Liczba portów cyfrowych wyjściowych: %d",
                s_module.LC0_MIDO);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Czestotliwość zegara modułu: %d MHz",
                                                s_module.LC0_MCLOCK / 1000);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
                                        /* konfiguracja toru a/c          */
        GoToLine(&rect,dc);GoToLine(&rect,dc);
        sprintf(line,"Konfiguracja toru a/c:");
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
```

```
        GoToLine(&rect,dc);GoToLine(&rect,dc);
        sprintf(line,"Liczba kanałów: %d", s_info.LC0_GCHAN);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Rozdzielczość: %d bitow", s_info.LC0_GRES);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Zakres napięć: %.1f..%.1f V",
                (float)s_info.LC0_GMINV / 10,
                (float)s_info.LC0_GMAXV / 10);
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        sprintf(line,"Karta posiada zainstalowany układ S&H");
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        sprintf(line,"Minimalne okresy próbkowania [us]:");
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        GoToLine(&rect,dc);
        rect.left +=5;
        sprintf(line,"");
        for(i = 0; i < s_info.LC0_GCHAN; i++)
         {
                if(i < s_info.LC0_GCHAN - 1)
                 sprintf(spar," %.1f ,",(float)s_info.LC0_GMINP[i] / 10);
                else
                 sprintf(spar," %.1f",(float)s_info.LC0_GMINP[i] / 10);
                strcat(line,spar);
                if(i == 7)
                 {
                  DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
                  sprintf(line,"");
                  GoToLine(&rect,dc);
                 }

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

/*****************************************************************************
 Funkcja : displaytext(HDC,RECT,unsigned char)
 Przeznaczenie:
  Wyswietlenie komunikatów tekstowych dla transmisji          A/C programu
  Korzysta ze zmiennej paintdet , która określa bieżącą fazę transmisji    blokowej oraz paintnum , która okkeśla rodzaj
  transmisji
*****************************************************************************/


int displaytext(HDC dc,RECT rect,unsigned char start)
{
 char line[200] ;
 rect.left += 10;
 switch(paintdet)
  {
    case 1:  /* blocktransmission */
        sprintf(line,"Poprawne przetwarzanie blokowe");
        DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
        break ;
    case 2:  /* failblocktransmission */
```

```
            sprintf(line,"Błędne przetwarzanie blokowe");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            break ;
      case 3:  /* interruptbefore */
            sprintf(line,"Przetwarzanie blokowe z oczekiwaniem 1000s .");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            GoToLine(&rect,dc);
            sprintf(line,"Oczekiwanie należy przerwać Ctrl-Break .");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            break ;
      case 4:   /* singletransmission */
            sprintf(line,"Przetwarzanie programowe - po 5 sekundach .");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            break ;
       case 5:   /* singlewrite */
            sprintf(line,"Wysłanie pojedynczej wartości na przetwornik CA");
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
            break ;
 }
 GoToLine(&rect,dc);
 if(paintdet < 4) /* transmisje blokowe : paintdet = 1..4*/
  {
            GoToLine(&rect,dc);
            sprintf
            (line,
            "Pomiar blokowy, %d kanałów, %d próbek, okres próbkowania %.1f us:",
            CHANNELS, SAMPLES, (float)s_analog_in.LC0_APER / 10
            );
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
  }
  switch(start)
            {
             case LC0_SIMMED:
              sprintf(line,"Warunek startu: natychmiast");
              break;
             case LC0_STIME:
              sprintf(line,"Warunek startu: upływ %lu sekund",
                        s_analog_in.LC0_ASTART.time);
              break;
            }
            GoToLine(&rect,dc);
            DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
 return(rect.top);
}

/*****************************************************************************
Funkcja my_break
Przeznaczenie:
Procedura obslugi przerwania pomiaru klawiszem Ctrl-Break.
*****************************************************************************
void FAR  my_break(void)
{
  /* sygnalizacja dźwiękowa */
  koniec = 1;
  MessageBeep(-1);
}

/*****************************************************************************
 Funkcja : installbreak
 Przeznaczenie:
 Zainstalowanie procedury obslugi przerwania generowanego przez
```

```
 Ctrl-Break.
 Sposob:
 Przez wywolanie funkcji BREAK.
*************************************************************************/
void installbreak(void)
{
        s_break.LC0_BMODE = LC0_BREAK_INST;
        s_break.LC0_BPROC = (void FAR *)&(my_break);  ;          /* podlozona zostanie     */
                                                                 /*   procedura użytkownika   */
        (*Break)((struct lc0_break FAR * )&s_break);
}

/*************************************************************************
 Funkcja : blocktransmission
 Przeznaczenie:
  Wykonanie poprawnej transmisji blokowej i wyswietlenie zmierzonych     wartosci.
 Sposob:
  Przez wykonanie funkcji transmission.
*************************************************************************/
void blocktransmission(void)
{
        tstart = LC0_SIMMED ;
        paintdet = 1 ;
        s_analog_in.LC0_APER = (long)s_info.LC0_GMINP[CHANNELS - 1]+200;
        drawscreen();
        transmission(LC0_SIMMED);
}

/*************************************************************************
 Funkcja : failblocktransmission()
 Przeznaczenie:
 Wykonanie blednej transmisji blokowej.
 Sposob:
 Przez wykonanie funkcji ANALOG_INPUT z okresem probkowania mniejszym   niz minimalny wskazany przez driver.
/*************************************************************************
void failblocktransmission(void)
{
        tstart = LC0_SIMMED ;
        paintdet = 2 ;
        s_analog_in.LC0_APER = (long)s_info.LC0_GMINP[CHANNELS - 1] - 1;
        drawscreen();
        transmission(LC0_SIMMED);
}

/*************************************************************************
 Funkcja : interruptbefore();
 Przeznaczenie:
Wykonanie poprawnej transmisji blokowej ale przerwanej przez operatora
  (Ctrl-Break) w trakcie czekania na spelnienie warunku startu.
 Sposob:
 Przez wykonanie funkcji ANALOG_INPUT z warunkiem startu LC0_STIME
  (start po okreslonym czasie) i parametrem tego warunku - 1000s.
*************************************************************************/
void interruptedbefore(void)
{
        tstart = LC0_STIME ;
        paintdet = 3 ;
        s_analog_in.LC0_APER = (long)s_info.LC0_GMINP[CHANNELS - 1];
        s_analog_in.LC0_ASTART.time = 1000;
        drawscreen();
        transmission(LC0_STIME);
```

```
}

/*************************************************************************
 Funkcja: transmission
 Przeznaczenie:
 Wykonanie transmisji blokowej.
  Sposob:
Przez wykonanie funkcji ANALOG_INPUT.
 Parametry:
  start - typ warunku startu
  Wartosc:
  Uwagi:
  Funkcja wywolujaca musi ustawic okres probkowania i parametry warunku   startu.
*************************************************************************/
void transmission(unsigned char start)
{
        s_analog_in.LC0_AMODULE = modulenum;
        s_analog_in.LC0_ANUM = 1;
        s_analog_in.LC0_AMODE = LC0_MOD_START |
                                        LC0_MOD_NEW_PAR |
                                        LC0_MOD_SYNCHR |
                                        LC0_MOD_BLOCK;
                        /* stop po zmierzeniu okreslonej liczby probek     */
        s_analog_in.LC0_ASTST = start + LC0_ZSAMPLES;
                                /* praca wielokanalowa, CHANNELS kanalow   */
        s_analog_in.LC0_ACHAN = CHANNELS;
        s_analog_in.LC0_AADDR = (int far *)buf;  /* konieczna konwersja na  */
                                                 /*   daleki adres          */
        s_analog_in.LC0_ALEN = BUFLEN;           /* dlugosc bufora                */
        s_analog_in.LC0_ABMAR = 0;
        s_analog_in.LC0_AEMAR = 0;      /* oba marginesy zerowe            */
                                        /* calkowita liczba probek         */
        s_analog_in.LC0_ASTOP.samples = SAMPLES * CHANNELS;
        /*ANALOG_INPUT    */
        (*AnalogInput)((struct lc0_analog_in FAR * )&s_analog_in);
        if(s_analog_in.LC0_STATUS != LC0_OK)
         {
                enddet = 1;
                drivererror(s_analog_in.LC0_STATUS, s_analog_in.LC0_ERR_STAT);
         }
        else
                enddet = 2;
        drawscreen();
}

/*************************************************************************
  Przeznaczenie:
 Wykonanie pomiaru bloku probek za pomoca tranmsisji programowej, przy
czym caly blok ma byc zmierzony po 10s od startu.
Sposob:
   Przez wykonanie funkcji ANALOG_INPUT.
*************************************************************************/
void singletransmission(void)
{
        int i ;
        tstart = LC0_STIME ;
        paintdet = 4 ;
        s_analog_in.LC0_AMODULE = modulenum;
        s_analog_in.LC0_ANUM = 1;
        s_analog_in.LC0_AMODE = LC0_MOD_START |
                                        LC0_MOD_NEW_PAR |
```

```
                                        LC0_MOD_SYNCHR;
                                    /* praca wielokanalowa, CHANNELS kanalow   */
        s_analog_in.LC0_ACHAN = CHANNELS;
        s_analog_in.LC0_ALEN = CHANNELS;     /* długosc bufora na jeden pomiar  */
        s_analog_in.LC0_ASTST = LC0_STIME+LC0_ZSAMPLES;
        s_analog_in.LC0_ASTART.time = 5;          /* start po 5 sekundach    */
        /* liczba probek do zmierzenia       */
        s_analog_in.LC0_ASTOP.samples =  CHANNELS;
        drawscreen();
        for(i=0;i<SAMPLES;i++)
         {
         s_analog_in.LC0_AADDR = (int far *)(buf+i*CHANNELS);
         /* ANALOG_INPUT    */
         (*AnalogInput)((struct lc0_analog_in FAR * )&s_analog_in);
         /* ustawienie trybu pomiarów natychmiastowych */
         if(i==0) s_analog_in.LC0_ASTST = LC0_SIMMED + LC0_ZSAMPLES;
         /* zgaszenie bitu analizy parametrow */
         if(i==1) s_analog_in.LC0_AMODE = LC0_MOD_START |LC0_MOD_SYNCHR;
         if(s_analog_in.LC0_STATUS != LC0_OK)
                 {
                 enddet = 1;
                 drivererror(s_analog_in.LC0_STATUS, s_analog_in.LC0_ERR_STAT);
                 break;
                 }
         else
                 enddet = 2;
         }
        drawscreen();
}

/*****************************************************************************
Przeznaczenie:
 Wyswietlenie bufora z danymi pomiarowymi.
 Parametry:
  buf - adres bufora
  c   - liczba kanalow
  s   - liczba probek na kanal
 dc - kontekt wyswietlania
 rect - okno do wyświetlania
*****************************************************************************/

int displaybuf(int *buf, int c, int s , HDC dc , RECT rect)
{
        int         i,j;
        char line[100],str[10];
        GoToLine(&rect,dc);
        for(i = 0; i < s; i++)
                 {
                 sprintf(line,"%2d:   ", i + 1);
                 for(j = 0; j < c; j++)
                  {
                          sprintf(str,"%04X    ", buf[i * c + j]);
                          strcat(line,str);
                  }
                  DrawText(dc,line,-1,&rect,DT_LEFT|DT_SINGLELINE);
                  GoToLine(&rect,dc);
                 }
         return(rect.top);
}

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

```
 Funkcja : singlewrite()
Przeznaczenie:
Wyslanie pojedynczej wartosci na przetwornik CA
Sposob:
Przez wykonanie funkcji ANALOG_OUTPUT.
*****************************************************************************/
void singlewrite(void)
{
        int     val[2];                 /* wartosc do wyslania          */
        tstart = LC0_SIMMED ;
        paintdet = 5 ;
        if(!s_module.LC0_MIDA)
        {
         MessageBox(hWnd,"Karta nie posiada zainstalowanych przetwornikow CA","Sygnał",MB_OK);
         drawscreen();
         return;
        }
        val[0] = 4095;
        s_analog_out.LC0_NMODULE = modulenum;
        s_analog_out.LC0_NNUM = 1;
        s_analog_out.LC0_NMODE = LC0_MOD_START |
                                 LC0_MOD_NEW_PAR |
                                 LC0_MOD_SYNCHR;
                                 /* praca jednokanalowa          */
        s_analog_out.LC0_NCHAN = 1 + 128;
        s_analog_out.LC0_NADDR.base_memory = (int far *)val; /* konieczna konwersja na */
                                             /*   daleki adres       */
        s_analog_out.LC0_NLEN = 1;           /* dlugosc bufora (tylko   */
                                             /*   na jedna wartosc)    */
        s_analog_out.LC0_NSTST = LC0_SIMMED + LC0_ZSAMPLES ;
        /* ilość próbek do wysłania */
        s_analog_out.LC0_NSTOP.samples = 1;
        drawscreen();
        /* ANALOG_OUTPUT    */
        (*AnalogOutput)((struct lc0_analog_out FAR * )&s_analog_out);
        if(s_analog_out.LC0_STATUS != LC0_OK)
         drivererror(s_analog_out.LC0_STATUS, s_analog_out.LC0_ERR_STAT);
        enddet = 1;
        drawscreen();
}

/*****************************************************************************/
 Przeznaczenie:
 Wypisanie komunikatow o bledzie / ostrzezeniu / dodatkowej informacji  bledzie.
 Parametry:
status   - LC0_STATUS
err_stat - LC0_ERR_STAT
*****************************************************************************/
void drivererror(char status, char err_stat)
{
 char sts[100];
        if(status > 0)
                sprintf(sts,"Ostrzeżenie: %s",
               (LPSTR)DriverWarnings[status - 1]);
        else
                sprintf(sts,"Błąd: %s", (LPSTR)DriverErrors[-status - 1]);
        MessageBox(hWnd,sts,"Sygnał",MB_OK);
        if(err_stat < 0)
         {
                sprintf(sts,"Informacje dodatkowe: %s",
               (LPSTR)DriverAdditionalErrors[-err_stat - 1]);
```

```
                MessageBox(hWnd,sts,"Sygnał",MB_OK);
        }
}

/*************************************************************************
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);
}

/*************************************************************************
 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 : 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);
 }
```

D O D A T E K C

d o d o k u m e n t a c j i b i b l i o t e k i D L L

m o d u ł u k o n t r o l n o - p o m i a r o w e g o

L C - 0 1 0 - 1 6 1 2

AMBEX-LC.PAS
struktury danych i stałe dla Pascala

```
(*************************************************************************)
(*                                                                       *)
(* definicje struktur danych i stalych dla wspolpracy z driver'ami modulow *)
(* analogowych serii LC-... z poziomu jezyka Pascal                      *)
(*                                                                       *)
(*************************************************************************)

(*************** REKORDY OPISOW ZLECEN FUNKCJI DRIVER'A ******************)

(************** struktury opisujace warunki startu i stopu ****************)
type
lc0_cond_level =                (* warunek "poziom sygnalu cyfrowego"      *)
        record
            mod_nr:     byte;           (* numer modulu                     *)
            port_nr:    byte;           (* numer portu                      *)
            inp_nr:     byte;           (* numer wejscia                    *)
            value:      byte;           (* oczekiwana wartosc               *)
        end;

(*-----------------------------------------------------------------------*)
type
lc0_cond_slope =                (* warunek "zbocze sygnalu cyfrowego"      *)
        record
            mod_nr:     byte;           (* numer modulu                     *)
            port_nr:    byte;           (* numer portu                      *)
            inp_nr:     byte;           (* numer wejscia                    *)
            slope:      byte;           (* rodzaj zbocza                    *)
        end;

const
                                (* kody rodzajow zbocza:                    *)
LC0_SLOPE_COND_UP       = 1;    (* narastajace                              *)
LC0_SLOPE_COND_DOWN     = 0;    (* opadajace                                *)

(*-----------------------------------------------------------------------*)
type
lc0_cond_dig =          (* warunek "kominacja sygnalow cyfrowych"          *)
        record
            mod_nr:     byte;           (* numer modulu                     *)
            port_nr:    byte;           (* numer portu                      *)
            mask:       byte;           (* maska aktywnych wejsc            *)
            pattern:    byte;           (* testowany wzorzec                *)
        end;


(*-----------------------------------------------------------------------*)
type
lc0_cond_date =         (* warunek "data"                                  *)
        record
            second:     byte;           (* sekunda                          *)
            minute:     byte;           (* minuta                           *)
            hour:       byte;           (* godzina                          *)
            day:        byte;           (* dzien miesiaca                   *)
        end;

(*-----------------------------------------------------------------------*)
type
lc0_cond_analog =               (* warunek "sygnal analogowy"              *)
        record
            mod_nr:     byte;           (* numer modulu                     *)
            converter:  byte;           (* nr przetwornika                  *)
            channel:    byte;           (* numer kanalu (b1..b7),           *)
                                        (*   kierunek przekroczenia (b8)    *)
            level:      integer;        (* prog wyzwalania                  *)
        end;

const
                        (* kody kierunku przekroczenia progu:              *)
LC0_ANALOG_COND_UP      = $80;  (* w kierunku wiekszych wartosci           *)
LC0_ANALOG_COND_DOWN    = 0;    (* w kierunku mniejszych wartosci          *)
LC0_ANALOG_COND_MASK    = $80;  (* maska kodu kierunku                     *)

(*-----------------------------------------------------------------------*)
type
lc0_start =                     (* ========== warunki startu ============*)
```

```
        record
            case integer of
                1: (level:      lc0_cond_level);        (* LC0_SLEVEL  *)
                2: (slope:      lc0_cond_slope);        (* LC0_SSLOPE  *)
                3: (dig_eq:     lc0_cond_dig);          (* LC0_SDIG_EQ *)
                4: (dig_ne:     lc0_cond_dig);          (* LC0_SDIG_NE *)
                5: (time:       longint);               (* LC0_STIME   *)
                6: (date:       lc0_cond_date);         (* LC0_SDATE   *)
                7: (analog:     lc0_cond_analog)        (* LC0_SANALOG *)
        end;

type
lc0_stop =                      (* ============ warunki stopu ==============*)
        record
            case integer of
                1: (samples:    longint);               (* LC0_ZSAMPLES *)
                2: (level:      lc0_cond_level);        (* LC0_ZLEVEL   *)
                3: (slope:      lc0_cond_slope);        (* LC0_ZSLOPE   *)
                4: (dig_eq:     lc0_cond_dig);          (* LC0_ZDIG_EQ  *)
                5: (dig_ne:     lc0_cond_dig);          (* LC0_ZDIG_NE  *)
                6: (time:       longint);               (* LC0_ZTIME    *)
                7: (date:       lc0_cond_date);         (* LC0_ZDATE    *)
                8: (analog:     lc0_cond_analog)        (* LC0_ZANALOG  *)
        end;

(*=========================================================================*)

(* Rekord opisu zlecenia funkcji  MODULE_INIT ============================*)
type
lc0_init =
        record
            LC0_CODE:           byte;   (* kod funkcji (16)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_IMODULE:        byte;   (* mapa modulow                     *)
        end;

(* Rekord opisu zlecenia funkcji  GET_TOTAL_CONFIGURATION ================*)
type
lc0_total =
        record
            LC0_CODE:           byte;   (* kod funkcji (17)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_TONF:           byte;   (* konfiguracja modulow             *)
            LC0_TIAD:           byte;   (* przetworniki a/c                 *)
            LC0_TIDA:           byte;   (* przetworniki c/a                 *)
            LC0_TCTC:           byte;   (* kanaly CTC                       *)
            LC0_TIDI:           byte;   (* porty wejsc cyfrowych            *)
            LC0_TIDO:           byte;   (* porty wyjsc cyfrowych            *)
        end;

(* Rekord opisu zlecenia funkcji   GET_MODULE_CONFIGURATION ==============*)
type
lc0_module =
        record
            LC0_CODE:           byte;   (* kod funkcji (18)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_MMODULE:        byte;   (* numer modulu                     *)
            LC0_MBASE1:         word;   (* adres bazowy pakietu 1           *)
            LC0_MBASE2:         word;   (* adres bazowy pakietu 2           *)
            LC0_MIAD:           byte;   (* przetworniki a/c                 *)
            LC0_MIDA:           byte;   (* przetworniki c/a                 *)
            LC0_MCTC:           byte;   (* kanaly CTC                       *)
            LC0_MIDI:           byte;   (* porty wejsc cyfrowych            *)
            LC0_MIDO:           byte;   (* porty wyjsc cyfrowych            *)
            LC0_MCLOCK:         word;   (* czestotliwosc zegara w kHz       *)
            LC0_MINT:           byte;   (* numer przerwania (programowy)    *)
             LC0_MEMA            pointer;(*adres buf. pamieci rozszerzonej *)
            LC0_MEML            longint;(*dlug. buf pamieci rozszerzonej   *)
        end;
```

```
(* Rekord opisu zlecenia funkcji  GET_INFO ================================*)
type
lc0_info =
        record
            LC0_CODE:           byte;   (* kod funkcji (19)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                        (* bledach                          *)
            LC0_GTYPE:          byte;   (* rodzaj urzadzenia                *)
            LC0_GMODULE:        byte;   (* numer modulu                     *)
            LC0_GNUM:           byte;   (* numer przetwornika/portu/        *)
                                        (*                       ukladu CTC *)
            LC0_GCHAN:          byte;   (* liczba kanalow                   *)
            LC0_GRES:           byte;   (* liczba bitow przetwornika        *)
            LC0_GTIME:          word;   (* czas konwersji przetwornika w ns*)
            LC0_GMINV:          shortint;       (* dolna granica zakresu   *)
                                        (*   napiec w dziesiatych  *)
                                        (*   czesciach volta   *)
            LC0_GMAXV:          shortint;       (* gorna granica zakresu   *)
                                        (*   napiec w dziesiatych  *)
                                        (*   czesciach volta   *)
            LC0_GDMA:           byte;   (* numer kanalu DMA                 *)
                                        (* minimalne okresy probkowania     *)
            LC0_GMINP:          array [1..32] of word;
        end;

(* Rekord opisu zlecenia funkcji  SET_CLOCK ================================*)
type
lc0_clock =
        record
            LC0_CODE:           byte;   (* kod funkcji (20)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                        (* bledach                          *)
            LC0_LCLOCK:         word;   (* czestotliowsc zegara w kHz       *)
        end;

(* Rekord opisu zlecenia funkcji  SET_VOLTAGE_RANGE ========================*)
type
lc0_volt =
        record
            LC0_CODE:           byte;   (* kod funkcji (21)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                        (* bledach                          *)
            LC0_VTYPE:          byte;   (* rodzaj urzadzenia                *)
            LC0_VMODULE:        byte;   (* numer modulu                     *)
            LC0_VNUM:           byte;   (* numer przetwornika               *)
            LC0_VMINV:          shortint;       (* dolna granica zakresu   *)
                                        (*   napiec w dziesiatych  *)
                                        (*   czesciach volta   *)
            LC0_VMAXV:          shortint;       (* gorna granica zakresu   *)
                                        (*   napiec w dziesiatych  *)
                                        (*   czesciach volta   *)
        end;

(* Rekord opisu zlecenia funkcji  SET_TIME =================================*)
type
lc0_time =
        record
            LC0_CODE:           byte;   (* kod funkcji (22)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                        (* bledach                          *)
            LC0_ETIME:          word;   (* maksymalny czas obslugi          *)
                                        (*   przerwania, ktore moze pojawic*)
                                        (*   sie w trakcie wykonywania     *)
                                        (*   dlugiego pomiaru podawany w   *)
                                        (*   mikrosekundach                *)
        end;

(* Rekord opisu zlecenia funkcji  WAIT_FOR_END =============================*)
type
lc0_wait_f_e =
```

```
        record
            LC0_CODE:           byte;   (* kod funkcji (23)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_WTYPE:          byte;   (* rodzaj urzadzenia               *)
            LC0_WMODULE:        byte;   (* numer modulu                    *)
            LC0_WNUM:           byte;   (* numer przetwornika              *)
            LC0_WMODE:          byte;   (* tryb pracy                      *)
            LC0_WRMNUM:         longint;(* rzeczywista liczba probek       *)
            LC0_WREMAR:         word;   (* rzeczywista dlugosc marginesu   *)
                                        (*   koncowego                     *)
        end;

const
                                (* tryby pracy funkcji                     *)
LC0_W_WAIT      = 0;            (* oczekiwanie                             *)
LC0_W_TEST      = 1;            (* test konca                              *)
LC0_W_FINISHED  = 2;            (* powiadomienie o koncu                   *)

(* Rekord opisu zlecenia funkcji  BREAK ===================================*)
type
lc0_sbreak =
        record
            LC0_CODE:           byte;   (* kod funkcji (24)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_BMODE:          byte;   (* tryb pracy                      *)
            LC0_BPROC:          pointer;(* adres procedury obslugi         *)

        end;

const
                                (* bity trybu pracy funkcji                *)
LC0_BREAK_EXEC          = 0;    (* przerwanie                              *)
LC0_BREAK_INST          = 1;    (* instalacja Ctrl_Break...                *)
LC0_BREAK_PREV          = 4;    (* ...z wywolaniem poprzedniej obslugi     *)
LC0_BREAK_UNINST        = 2;    (* wyinstalowanie wszystkiego              *)

(* Rekord opisu zlecenia funkcji  DIGITAL_INPUT ===========================*)
type
lc0_digital_in =
        record
            LC0_CODE:           byte;   (* kod funkcji (25)                 *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_DMODULE:        byte;   (* numer modulu                    *)
            LC0_DNUM:           byte;   (* numer portu                     *)
            LC0_DSTST:          byte;   (* typ warunku startu              *)
            LC0_DVAL:           byte;   (* odczytana wartosc               *)
            LC0_DSTART:         lc0_start;      (* parametry warunku startu*)
        end;

(* Rekord opisu zlecenia funkcji  DIGITAL_OUTPUT ==========================*)
type
lc0_digital_out =
        record
            LC0_CODE:           byte;   (* kod funkcji (26)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_OMODULE:        byte;   (* numer modulu                    *)
            LC0_ONUM:           byte;   (* numer portu                     *)
            LC0_OSTST:          byte;   (* typ warunku startu              *)
            LC0_OVAL:           byte;   (* wartosc do wyslania             *)
            LC0_OSTART:         lc0_start;      (* parametry warunku startu*)
        end;

(* Rekord opisu zlecenia funkcji  CTC_WRITE ===============================*)
type
lc0_ctc_write =
        record
            LC0_CODE:           byte;   (* kod funkcji (27)                *)
```

```
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_CMODULE:        byte;   (* numer modulu                    *)
            LC0_CMODE:          byte;   (* tryb pracy funkcji              *)
            LC0_CFUN:           byte;   (* tryb pracy kanalu               *)
            LC0_CVAL:           word;   (* nowa wartosc licznika           *)
        end;

const
                                (* tryby pracy funkcji                     *)
LC0_SET_CTC_MODE        = 1;    (* zaprogramuj tryb pracy kanalu           *)
LC0_SET_COUNTER_VALUE   = 2;    (* zaladuj nowa wartosc licznika           *)
LC0_CTC_ENABLE          = 4;    (* zezwolenie CTC                          *)

(* Rekord opisu zlecenia funkcji  CTC_READ ================================*)
type
lc0_ctc_read =
        record
            LC0_CODE:           byte;   (* kod funkcji (28)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_UMODULE:        byte;   (* numer modulu                    *)
            LC0_UNUM:           byte;   (* numer kanalu                    *)
            LC0_UVAL:           word;   (* odczytana wartosc licznika      *)
        end;

(* Rekord opisu zlecenia funkcji  DATA_TRANSMIT ===========================*)
type
lc0_transmit =
        record
            LC0_CODE:           byte;   (* kod funkcji (29)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
           LCO_RMODULE           byte;   (* numer modulu                    *)
            LC0_RMODE:          byte;   (* tryb przesylania                *)
            LC0_RADDR:          pointer;(* adres bufora w pam. podstawowej *)
            LC0_RLEN:           longint;(* dlugosc bufora w pam. podst.    *)
            LC0_RMEAS:          longint;(* numer pierwszej probki  *)
            LC0_RNUM:           longint;(* liczba probek do przeslania     *)
            LC0_RMEMA:          pointer;(* adres bufora w pamieci          *)
            LC0_RRNUM:          longint;(* rzeczywista liczba probek       *)

        end;

const
                                (* tryby pracy funkcji                     *)
LC0_TO_EXT_DIR          = 1;    (* do pamieci rozszerzonej                 *)
LC0_FROM_EXT_DIR        = 0;    (* z pamieci modulu / rozszerzonej         *)

(* Rekord opisu zlecenia funkcji ANALOG_INPUT =============================*)
type
lc0_analog_in =
        record
            LC0_CODE:           byte;   (* kod funkcji (30)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_AMODULE:        byte;   (* numer modulu                    *)
            LC0_ANUM:           byte;   (* numer przetwornika              *)
            LC0_AMODE:          word;   (* tryb pracy funkcji              *)
            LC0_ASTST:          byte;   (* typy warunkow startu i stopu    *)
            LC0_APER:           longint;(* okres probkowania               *)
            LC0_APER2:          word;   (* krotnosc okresu probkowania     *)
            LC0_ACHAN:          byte;   (* liczba kanalow podstawowych     *)
                                        (*   (b8 okresla czy praca         *)
                                        (*   wielokanalowa (0) czy         *)
                                        (*   jednokanalowa (1))            *)
            LC0_ACHAN2:         byte;   (* liczba kanalow dodatkowych      *)
            LC0_AADDR:          pointer;(* adres bufora w pam. podstawowej *)
            LC0_ALEN:           longint;(* dlugosc bufora                  *)
            LC0_AMEMA:          pointer;(* adres bufora w pamieci          *)
            LC0_ABMAR:          word;   (* dlugosc marginesu poczatkowego  *)
```

```
            LC0_AEMAR:          word;   (* dlugosc marginesu koncowego     *)
            LC0_AHAND:          word;   (* numer handler'a zbioru          *)
            LC0_ASTART:         lc0_start;      (* parametry warunku startu*)
            LC0_ASTOP:          lc0_stop;       (* parametry warunku stopu *)
            LC0_ARDIV1:         word;   (* pierwszy podzielnik zegara      *)
            LC0_ARDIV2:         word;   (* drugi podzielnik zegara         *)
            LC0_ARMNUM:         longint;(* rzeczywista liczba probek       *)
            LC0_ARBMAR:         word;   (* rzeczywista dlugosc marg. pocz. *)
            LC0_AREMAR:         word;   (* rzeczywista dlugosc marg. konc. *)
            LC0_ARLEN:          longint;(* rzeczywista liczba przepisanych *)
                                        (*   probek                        *)
            LC0_ARBUF:          word;   (* dla pracy z buforem cyklicznym: *)
                                        (*   numer probki okreslajacej     *)
                                        (*   poczatek bufora po zakonczeniu*)
                                        (*   pomiaru                       *)
        end;

(* Rekord opisu zlecenia funkcji  ANALOG_OUTPUT ============================*)
type
lc0_memory =
        record
            case integer of
                1: (base_memory:        pointer);
                2: (extended_memory:    pointer)
        end;

type
lc0_analog_out =
        record
            LC0_CODE:           byte;   (* kod funkcji (31)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_NMODULE:        byte;   (* numer modulu                    *)
            LC0_NNUM:           byte;   (* numer przetwornika              *)
            LC0_NMODE:          word;   (* tryb pracy funkcji              *)
            LC0_NSTST:          byte;   (* typy warunkow startu i stopu    *)
            LC0_NCHAN:          byte;   (* liczba kanalow (b8 okresla      *)
                                        (*   czy praca wielokanalowa (0)   *)
                                        (*   czy jednokanalowa (1))        *)
            LC0_NPER:           longint;(* okres probkowania               *)
            LC0_NADDR:          lc0_memory;     (* adres bufora            *)
                                        (*   (segment:offset               *)
            LC0_NLEN:           longint;(* dlugosc bufora w pam. podst.    *)
            LC0_NHAND:          word;   (* numer handler'a zbioru          *)
            LC0_NSTART:         lc0_start;      (* parametry warunku startu*)
            LC0_NSTOP:          lc0_stop;       (* parametry warunku stopu *)
            LC0_NRMNUM:         longint;(* rzeczywista liczba wyslanych    *)
                                        (*   probek                        *)
        end;

(* Rekord opisu zlecenia funkcji  LEAVE_DRIVER =============================*)
type
lc0_leave =
        record
            LC0_CODE:           byte;   (* kod funkcji (32)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
        end;

(* Rekord opisu zlecenia funkcji  INTERRUPT_SERVICE ========================*)
type
lc0_interrupt =
        record
            LC0_CODE:           byte;   (* kod funkcji (33)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
            LC0_SMODULE:        byte;   (* numer modulu                    *)
            LC0_SPROC:          pointer;(* adres procedury obslugi         *)
            LC0_SSTAT:          pointer;(* adres slowa komunikacyjnego     *)
        end;
(* Rekord opisu zlecenia funkcji MEMORY_USE                                 *)
type
```

```
lc0_memory =
            LC0_CODE:           byte;   (* kod funkcji (34)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
             LC0_MMODULE:         byte;            (* numer modulu             *)
             LC0_MMODE            byte;            (* obecnie 1                *)
             LC0_MNUM             longint          (* liczba probek w nowym    *)
                                                   (* buforze                  *)
             LC0_MRNUM            longint          (* rzeczywista wielkosc     *)
                                                   (* nowego bufora w probkach *)
             LC0_MRMEMA           pointer          (* adres nowego bufora      *)
          end;
(* Rekord opisu zlecenia funkcji  CONFIG ============================*)
type
lc0_sleave =
        record
            LC0_CODE:           byte;   (* kod funkcji (35)                *)
            LC0_STATUS:         shortint;       (* kod odpowiedzi driver'a *)
            LC0_ERR_STAT:       shortint;       (* dodatkowe informacje o  *)
                                                (* bledach                 *)
             LC0_PATH            pointer ;      (* adres stringu okreslajacego *)
                                                (* sciezka do pliku AMBEX.INI*)
        end;


LC0_IS_START    = 1;            (* przerwanie wystapilo z powodu           *)
                                (*   rozpoczecia pomiaru                   *)
LC0_IS_ONE      = 2;            (* przerwanie nastapilo z powodu zmierzenia*)
                                (*   kolejnej serii pomiarowej             *)
LC0_IS_END_ADC  = 4;            (* przerwanie nastapilo z powodu           *)
                                (*   zakonczenia przetwarzania a/c         *)
LC0_IS_END_DAC  = 8;            (* przerwanie nastapilo z powodu           *)
                                (*   zakonczenia przetwarzania c/a         *)
LC0_IS_BROKEN   = 16;           (* zakonczono przetwarzanie z powodu       *)
                                (* wykonania funkcji BREAK                 *)
LC0_IS_SAMPLE   = 32;           (* przerwanie wystapilo z powodu zmierzenia*)
                                (*   kolejnej probki                       *)

(****************** kody funkcji driver'a *********************************)
MODULE_INIT                     =  16;
GET_TOTAL_CONFIGURATION         =  17;
GET_MODULE_CONFIGURATION        =  18;
GET_INFO                        =  19;
SET_CLOCK                       =  20;
SET_VOLTAGE_RANGE               =  21;
SET_TIME                        =  22;
WAIT_FOR_END                    =  23;
BREAK                           =  24;
DIGITAL_INPUT                   =  25;
DIGITAL_OUTPUT                  =  26;
CTC_WRITE                       =  27;
CTC_READ                        =  28;
DATA_TRANSMIT                   =  29;
ANALOG_INPUT                    = 30;
ANALOG_OUTPUT                   = 31;
LEAVE_DRIVER                    = 32;
INTERRUPT_SERVICE               = 33;
MEMORY_USE                     = 34;
CONFIG                         = 35;

(*************** numery przerwan obslugiwanych przez driver ***************)
LC010_16        = $99;          (* LC-010-1612                             *)
LC011_08        = $90;          (* LC-011-0812                             *)
LC011_16        = $91;          (* LC-011-1612                             *)
LC015_16        = $92;          (* LC-015-1612                             *)
LC020_08_0      = $93;          (* LC-020-0812 v.0                         *)
LC020_08_2      = $94;          (* LC-020-0812 v.1 i v.2                   *)
LC020_32        = $95;          (* LC-020-3212                             *)
LC030_16        = $96;          (* LC-030-1612                             *)
LC060_06        = $97;          (* LC-060-0612                             *)

(****************** kody odpowiedzi funkcji driver'a **********************)
LC0_OK  = 0;
```

```
(* ======= ostrzezenia ======== *)
LC0_NON_EX_MOD          = 1;    (* nie istniejacy(e) modul(y)              *)
LC0_OTHER_LEN           = 2;    (* przepisano mniejsza liczbe pomiarow     *)
LC0_PREMATURE_END       = 3;    (* przedwczesne zakonczenie z powodu       *)
                                (*   przepelnienia bufora                  *)
LC0_IN_PROGRESS         = 4;    (* badana transmisja jeszcze trwa          *)
LC0_IS_INIT            =5;     (*modul zainicjowany                      *)

(* ========== bledy ========== *)
LC0_UNKN_FUNC           =  -1;  (* nieznany kod funkcji                    *)
LC0_NO_MODULE           =  -2;  (* brak modulu(ow)                         *)
LC0_BAD_DEV_TYP         =  -3;  (* bledny typ urzadzenie                   *)
LC0_NONEX_DEV           =  -4;  (* nie istnieje urzadzenie o tym numerze   *)
LC0_BAD_FREQ            =  -5;  (* zla czestotliwosc zegara                *)
LC0_BAD_RANGE           =  -6;  (* zly zakres napiec                       *)
LC0_NO_OPER             =  -7;  (* zadna operacja nie jest wykonywana      *)
LC0_BAD_MARGIN          =  -8;  (* bledna dlugosc marginesu poczatkowego   *)
LC0_BAD_BUF_ADR         =  -9;  (* bledny adres bufora                     *)
LC0_BAD_BUF_LEN         = -10;  (* bledna dlugosc bufora                   *)
LC0_DEV_BUSY            = -11;  (* urzadzenie jest zajete                  *)
LC0_BAD_PER             = -12;  (* zly okres probkowania                   *)
LC0_BAD_CHAN_N          = -13;  (* zla liczba kanalow                      *)
LC0_BAD_CHAN            = -14;  (* numer nie istniejacego kanalu           *)
LC0_BROKEN              = -15;  (* przerwano funkcja BREAK                 *)
LC0_INTR_NOT_INST       = -16;  (* procedura obslugi przerwania nie jest   *)
                                (*   zainstalowana                         *)
LC0_ILL_START_CODE      = -17;  (* nielegalny typ warunku startu           *)
LC0_ILL_STOP_CODE       = -18;  (* nielegalny typ warunku stopu            *)
LC0_BAD_PROC            = -19;  (* bledny adres procedury obslugi          *)
                                (*   przerwania lub slowa komunikacyjnego  *)
LC0_TOO_LONG_MARG       = -20;  (* margines dluzszy od bufora              *)
LC0_ILL_START           = -21;  (* bledne parametry warunku startu         *)
LC0_ILL_STOP            = -22;  (* bledne parametry warunku stopu          *)
LC0_BAD_MNUM            = -23;  (* bledny numer pierwszej probki           *)
LC0_NOT_SUPPORTED       = -24;  (* dla danego modulu funkcja nie jest      *)
                                (*   realizowana                           *)
LC0_BAD_CTC_MODE        = -25;  (* bledny tryb pracy CTC                   *)
LC0_NO_PARAMS           = -26;  (* nie podano parametrow przetwarzania     *)
LC0_OVERRUN             = -27;  (* zakonczono przetwarzanie z powodu bledu *)
                                (*   OVERRUN                               *)
LC0_NO_DMA              = -28;  (* z danym urzadzeniem nie jest zwiazany   *)
                                (*   zaden kanal DMA                       *)
LC0_NO_IRQ              = -29;  (* z danym modulem nie jest zwiazane zadne *)
                                (*   przerwanie                            *)
LC0_NOT_FULLY_SUP       = -30;  (* zadany tryb wykonania funkcji nie jest  *)
                                (*   realizowany dla danego typu modulu lub*)
                                (*   funkcja w opracowaniu                 *)
LC0_NO_EXTMEM           = -31;  (* brak pamieci roszerzonej                *)
LC0_NO_SEC_FREQ         = -32;  (* modul nie moze wykonywac pomiarow z     *)
                                (*   podwojna czestotliwoscia              *)
LC0_INTR_INST           = -33;  (* procedura obslugi przerwania juz        *)
                                (*   zainstalowana                         *)
LC0_BAD_PER2            = -34;  (* bledna wielokrotnosc okresu probkowania *)
                                (*   (0 lub 1)                             *)
LC0_BAD_MODE            = -35;  (* bledny tryb pracy                       *)
LC0_BAD_EXTMEM          = -36;  (* zly adres bufora w pamieci rozszerzonej *)
LC0_CTC_NOT_PROGRAMMED  = -37;  (* zapis licznika przy niezaprogramowanym  *)
                                (*   trybie pracy                          *)
LC0_REJECTED            = -38;  (* za wiele rownoleglych wejsc do driver'a *)
LC0_BAD_CONFIG          = -39    (* bledny plik konfiguracyjny             *)
LC0_NOT_INIT           = -40     (* modul nie zainicjowany                 *)
LC0_NO_DMA_TRANS        = -41    (* niemozliwy transfer DMA                *)

(************************* dodatkowe informacje o bledach ****************)
LC0_E_OK                = 0;    (* brak dodatkowych informacji             *)
LC0_E_NO_MODULE         = -1;   (* nie ma takiego modulu                   *)
LC0_E_NONEX_DEV         = -2;   (* nie istnieje urzadzenie o tym numerze   *)
LC0_E_BAD_CHAN          = -3;   (* numer nieistniejacego kanalu            *)
LC0_E_BAD_TIME          = -4;   (* zly odcinek czasu                       *)
LC0_E_BAD_DATE          = -5;   (* zla specyfikacja daty                   *)
LC0_E_BAD_THRE          = -6;   (* bledny prog wyzwalania analogowego      *)
LC0_E_BROKEN_WAIT       = -7;   (* funkcja przerwana w trakcie oczekiwania *)
                                (*   na spelnienie warunku startu          *)
LC0_E_BROKEN_RUN        = -8;   (* funkcja przerwana w trakcie             *)
                                (*   przetwarzania                         *)
```

```
LC0_E_BAD_LEN           = -9;   (* zadeklarowano za duzo probek do         *)
                                (*   zmierzenia                            *)
LC0_E_MOD_UNABLE        = -10    (* modul zajety lub nie zainicjowany    */

(************************* kody warunkow startu ***************************)
LC0_SIMMED      = 0;            (* natychmiastowy                         *)
LC0_SHARD       = 1;            (* od sygnalu sprzetowego                 *)
LC0_SLEVEL      = 2;            (* od poziomu sygnalu cyfrowego           *)
LC0_SSLOPE      = 3;            (* od zbocza sygnalu cyfrowego            *)
LC0_SDIG_EQ     = 4;            (* od kombinacji bitow - rowne            *)
LC0_SDIG_NE     = 5;            (* od kombinacji bitow - rozne            *)
LC0_STIME       = 6;            (* po uplynieciu okreslonego czasu        *)
LC0_SDATE       = 7;            (* o podanym czasie                       *)
LC0_SANALOG     = 8;            (* od sygnalu analogowego                 *)

(************************* kody warunkow stopu ****************************)
LC0_ZSAMPLES    = $00;          (* po zmierzeniu okreslonej liczby probek  *)
LC0_ZBREAK      = $10;          (* po wykonaniu funkcji BREAK              *)
LC0_ZLEVEL      = $20;          (* od poziomu sygnalu cyfrowego           *)
LC0_ZSLOPE      = $30;          (* od zbocza sygnalu cyfrowego            *)
LC0_ZDIG_EQ     = $40;          (* od kombinacji bitow - rowne            *)
LC0_ZDIG_NE     = $50;          (* od kombinacji bitow - rozne            *)
LC0_ZTIME       = $60;          (* po uplynieciu okreslonego czasu        *)
LC0_ZDATE       = $70;          (* o podanym czasie                       *)
LC0_ZANALOG     = $80;          (* od sygnalu analogowego                 *)

(************************* kody numerow modulow ***************************)
LC0_MODA        = 1;            (* modul A                                *)
LC0_MODB        = 2;            (* modul B                                *)
LC0_MODC        = 3;            (* modul C                                *)
LC0_MODD        = 4;            (* modul D                                *)

(************************* maski modulow w mapie **************************)
LC0_MODAMAP     = 1;            (* modul A                                *)
LC0_MODBMAP     = 2;            (* modul B                                *)
LC0_MODCMAP     = 4;            (* modul C                                *)
LC0_MODDMAP     = 8;            (* modul D                                *)

(************************* kody typow urzadzen ****************************)
LC0_DINPUT      = 1;            (* wejsciowy port cyfrowy                 *)
LC0_DOUTPUT     = 2;            (* wyjsciowy port cyfrowy                 *)
LC0_AINPUT      = 3;            (* przetwornik a/c                        *)
LC0_AOUTPUT     = 4;            (* przetwornik c/a                        *)
LC0_CTC         = 5;            (* kanal CTC                              *)

(*********** maski trybu pracy funkcji ANALOG_INPUT i ANALOG_OUTPUT ********)
LC0_MOD_START           =    1; (* start przetwarzania                    *)
LC0_MOD_NEW_PAR         =    2; (* nowe parametry                         *)
LC0_MOD_SYNCHR          =    4; (* praca synchroniczna                    *)
LC0_MOD_ASYNCHR         =    0; (* praca asynchroniczna                   *)
LC0_MOD_INTR            =    8; (* praca bez przerwan                     *)
LC0_MOD_INTR_TYPE       =   16; (* bit trybu przerwan                     *)
LC0_MOD_END_INTR        =    0; (* przerwanie po koncu przetwarzania      *)
LC0_MOD_ONE_INTR        =   16; (* przerwanie po kazdej probce            *)
LC0_MOD_BLOCK           =   32; (* praca blokowa                          *)
LC0_MOD_SINGLE          =    0; (* praca pojedyncza                       *)
LC0_MOD_CYCL            =   64; (* bufor cykliczny                        *)
LC0_MOD_FILE_W          =  128; (* zapis do pliku                         *)
LC0_MOD_FILE_R          =  128; (* odczyt z pliku                         *)
LC0_MOD_MEM_W           =  256; (* przepisanie do pamieci                 *)
LC0_MOD_EXT_CLK         =  512; (* zegar zewnetrzny                       *)
LC0_MOD_EXT_MEM         = 1024; (* pamiec rozszerzona                     *)
LC0_MOD_RAM_SEK         =  2048; (* praca z pamiecia sekwencji pomiarowej  *)
LC0_MOD_PAGE            = 4096; (* blokada pomiaru na czas progr. DMA      *)

(************* wartosci fragmentow bajtu kontrolnego 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                             *)

(* definicje typow wskaznikowych do struktur danych *)
type
 Plc0_init =    ^lc0_init ;
 Plc0_total=    ^lc0_total;
 Plc0_module=   ^lc0_module;
 Plc0_info=     ^lc0_info;
 Plc0_clock=    ^lc0_clock;
 Plc0_volt=     ^lc0_volt;
 Plc0_time=     ^lc0_time;
 Plc0_wait=     ^lc0_wait_f_e;
 Plc0_interrupt=^lc0_interrupt_s;
 Plc0_digital_in= ^lc0_digital_in;
 Plc0_digital_out= ^lc0_digital_out;
 Plc0_ctc_write=   ^lc0_ctc_write;
 Plc0_ctc_read=    ^lc0_ctc_read;
 Plc0_leave=       ^lc0_sleave;
 Plc0_config=      ^lc0_sconfig;
 Plc0_break=       ^lc0_sbreak;
 Plc0_transmit=    ^lc0_transmit;
 Plc0_memoryuse=   ^lc0_memory_u;
 Plc0_analog_in=   ^lc0_analog_in;
 Plc0_analog_out=  ^lc0_analog_out;

(*****************************************************************************)
(* Funkcje biblioteki LC1016A.DLL                                            *)
(*****************************************************************************)
(*
  PROCEDURE LC0_ModuleInit(param : Plc0_init);
  PROCEDURE LC0_GetTotalConf(param : Plc0_total);
  PROCEDURE LC0_GetModule(param : Plc0_module);
  PROCEDURE LC0_GetInfo(param : Plc0_info);
  PROCEDURE LC0_SetClock(param : Plc0_clock);
  PROCEDURE LC0_SetVoltage(param : Plc0_volt);
  PROCEDURE LC0_SetTime(param : Plc0_time);
  PROCEDURE LC0_Wait(param : Plc0_wait);
  PROCEDURE LC0_Interrupt(param : Plc0_interrupt);
  PROCEDURE LC0_DigitalIn(param : Plc0_digital_in);
  PROCEDURE LC0_DigitalOut(param : Plc0_digital_out);
  PROCEDURE LC0_CTCWrite(param : Plc0_ctc_write);
  PROCEDURE LC0_CTCRead(param : Plc0_ctc_read);
  PROCEDURE LC0_Leave(param : Plc0_leave);
  PROCEDURE LC0_Config(param : Plc0_config);
  PROCEDURE LC0_Break(param : Plc0_break);
  PROCEDURE LC0_DataTransmit(param : Plc0_transmit);
  PROCEDURE LC0_MemoryUse(param : Plc0_memoryuse);
  PROCEDURE LC0_AnalogIn(param : Plc0_analog_in);
  PROCEDURE LC0_AnalogOut(param : Plc0_analog_out);
 *)
```

D O D A T E K D

d o d o k u m e n t a c j i b i b l i o t e k i D L L

m o d u ł u k o n t r o l n o - p o m i a r o w e g o

L C - 0 1 0 - 1 6 1 2

WSMP1016.PAS
program przykładowy w Pascalu

```
(* WSMP1016.PAS ********************************************************************
        Program przykładowy pokazujący sposób wykorzystania drivera modułów
        serii LC   firmy AMBEX.
        Sposób dołaczenia biblioteki DLL : dynamicznie          *)
(*************************************************************************
  Program korzysta ze standardowych funkcji API Windows 3.1 .
  W katalogu zawierajacym standardowe naglowki PAS muszą sie znajdowac pliki:
  AMBEX-LC.PAS  - plik nagłówkowy drivera ,
  WSMP1016.RES - plik zasobów   ,

  Pliki :
  - AMBEX.INI  - plik konfiguracyjny modułów serii LC ,
  - LC1016A.DLL - driver (biblioteka DLL) do kart LC-010-1612 ,
  powinny znajdować się w jednym z katalogów ze standardowej ścieżki przeszu-
  kiwań Windows (według kolejności przeszukiwania) :
  1. katalog systemowy WINDOWS ,
  2. katalog WINDOWS\SYSTEM ,
  3. bieżący katalog aplikacji ,
  4. katalog znajdujący się na ścieżce przeszukiwań PATH .

  Opcje kompilatora (Borland Pascal)
  - Force far calls : on ,
  - Word align data : off ,
  - smart callbacks : on ,
  - windows stack frame : on ,

              *)
(*************************************************************************)

program wsmp1016;
{$R Wsmp1016}
uses Win31,WinProcs, WinTypes,Strings;
{$i ambex-lc.pas}
type
   buftype = array[1..320] of integer;                    (* bufor na probki        *)
   TModuleInit = PROCEDURE(param : Plc0_init);
   TGetTotalConfiguration=PROCEDURE( param : Plc0_total);
   TGetModuleConfiguration=PROCEDURE( param : Plc0_module);
   TGetInfo=PROCEDURE(param : Plc0_info);
   TBreak=PROCEDURE(param : Plc0_break);
   TAnalogInput=PROCEDURE(param : Plc0_analog_in);
   TAnalogOutput=PROCEDURE(param : Plc0_analog_out);
   TLeave=PROCEDURE(param : Plc0_leave);
var
  (* funkcje driver'a *)
   PModuleInit :TFarProc;
   PGetTotalConfiguration: TFarProc;
   PGetModuleConfiguration: TFarProc;
   PGetInfo:TFarProc;
   PBreak:TFarProc;
   PAnalogInput:TFarProc;
   PAnalogOutput:TFarProc;
   PLeave:TFarProc;
   buf: buftype;                          (* bufor na probki              *)
   s_init:      lc0_init;
   s_total:     lc0_total;
   s_module:    lc0_module;
   s_info:      lc0_info;
   s_break:     lc0_sbreak;
   s_analog_in: lc0_analog_in;
   s_analog_out:lc0_analog_out;
```

```
   s_leave:    lc0_sleave;
   modulenum: integer;                    (* numer badanego modulu        *)
   paintnum:  integer ;
   paintdet:  integer;
   enddet:    integer;
   tstart:    byte;
   vhWnd:     HWnd;
   sstat:     integer;
   koniec:    integer;
   HDLL :     THandle;
   HInst :    THandle ;
   fproc:     TFarProc;
   hDriverInst: THandle      ;  (* desktyptor drivera*)
const
   BUFLEN       = 320;                    (* dlugosc bufora        *)
   SAMPLES      = 20;                     (* liczba probek         *)
   CHANNELS     = 8;                      (* liczba kanalow        *)
   MY_MESSAGE  = WM_USER +1 ;

(*************************************************************************)
(* 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:                                                        *)
(*   Inicjalizacja zmiennych.                                            *)
(*************************************************************************)
procedure initprogram;
begin
   s_init.LC0_CODE              := MODULE_INIT;
   s_total.LC0_CODE             := GET_TOTAL_CONFIGURATION;
   s_module.LC0_CODE            := GET_MODULE_CONFIGURATION;
   s_info.LC0_CODE              := GET_INFO;
   s_break.LC0_CODE             := BREAK;
   s_analog_in.LC0_CODE         := ANALOG_INPUT;
   s_leave.LC0_CODE             := LEAVE_DRIVER;
   paintnum :=0;
   paintdet:=0;
   enddet:=0;
```

```
   koniec:=0;
end;

(*****************************************************************************
 Funkcja : loaddriver

 Przeznaczenie : Dynamiczne wczytanie biblioteki 'lc1116a.dll' , ustawienie
                 wskaźników do funkcji publicznych używanych w programie .

 Rezultat      : != 0 , gdy wszystko O.K


*****************************************************************************)
FUNCTION loaddriver:BOOL ;
var
 funptr :TFarProc ;
Begin
 hDriverInst := LoadLibrary('lc1016a.dll');    (* wczytanie do pamięci *)
 (* ustawienie wskaźników do funkcji *)
 if hDriverInst > HINSTANCE_ERROR then
  begin
        funptr := GetProcAddress(hDriverInst,MAKEINTRESOURCE(MODULE_INIT));
        PModuleInit:= MakeProcInstance(funptr,HInst);
        funptr :=GetProcAddress(hDriverInst,MAKEINTRESOURCE(GET_TOTAL_CONFIGURATION));
        PGetTotalConfiguration := MakeProcInstance(funptr,HInst);
        funptr :=GetProcAddress(hDriverInst,MAKEINTRESOURCE(GET_MODULE_CONFIGURATION));
        PGetModuleConfiguration := MakeProcInstance(funptr,HInst);
        funptr :=GetProcAddress(hDriverInst,MAKEINTRESOURCE(GET_INFO));
        PGetInfo := MakeProcInstance(funptr,HInst);
        funptr :=GetProcAddress(hDriverInst,MAKEINTRESOURCE(BREAK));
        PBreak := MakeProcInstance(funptr,HInst);
        funptr :=GetProcAddress(hDriverInst,MAKEINTRESOURCE(ANALOG_INPUT));
        PAnalogInput := MakeProcInstance(funptr,HInst);
        funptr :=GetProcAddress(hDriverInst,MAKEINTRESOURCE(ANALOG_OUTPUT));
        PAnalogOutput := MakeProcInstance(funptr,HInst);
        funptr :=GetProcAddress(hDriverInst,MAKEINTRESOURCE(LEAVE_DRIVER));
        PLeave := MakeProcInstance(funptr,HInst);
        loaddriver := TRUE;
  end
 else
  begin
        MessageBeep(0);
        MessageBox(vhWnd,'Błąd przy ładowaniu drivera do pamięci', 'Informacja',MB_OK);
        DestroyWindow(vhWnd);
        loaddriver :=FALSE;
  end ;
end ;

(*****************************************************************************
 Funkcja quit()

 Przeznaczenie : Wykonanie funkcji LEAVE_DRIVER

*****************************************************************************)
PROCEDURE quit;
begin
  TLeave(PLeave)(@s_leave);
  FreeLibrary(hDriverInst);
end ;
(*****************************************************************************)
(* Przeznaczenie:                                                            *)
```

```
(*   Rozpoznanie konfiguracji badanego modulu.                                  *)
(* Sposob:                                                                      *)
(*   Przez wykorzystanie funkcji driver'a GET_TOTAL_CONFIGURATION,              *)
(*   GET_MODULE_CONFIGURATION, GET_INFO.                                        *)
(* Uwagi:                                                                       *)
(*   Funkcja nadaje wartosc zmiennej modulenum (numer badanego modulu)          *)
(*   wypelnia struktury total, module, info i inicjalizuje zainstalowane        *)
(*    moduly.                                                                   *)
(******************************************************************************)
procedure askdriver;
label     lab1;
begin
   TGetTotalConfiguration(PGetTotalConfiguration)(@s_total);
         (* GET_TOTAL_CONFIGURATION *)
                                   (* inicjalizacja zainstalowanych modulow   *)
   s_init.LC0_IMODULE := s_total.LC0_TONF and $F;
   TModuleInit(PModuleInit)(@s_init);              (* MODULE_INIT     *)
                                          (* sprawdzenie, ktory modul jest   *)
                                          (* zainstalowany: A, B, C czy D    *)
   for modulenum := 1 to 4 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;
   TGetModuleConfiguration(PGetModuleConfiguration)(@s_module);
                              (* GET_MODULE_CONFIGURATION*)
                                          (* spytanie o konfiguracje toru a/c*)
   s_info.LC0_GMODULE := modulenum;
   s_info.LC0_GTYPE   := LC0_AINPUT;
   s_info.LC0_GNUM    := 1;
   TGetInfo(PGetInfo)(@s_info);            (* GET_INFO        *)
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 ;


(******************************************************************************)
(* Przeznaczenie:                                                             *)
(*   Wyswietlenie konfiguracji badanego modulu i jego toru a/c.               *)
(* 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 modułu:',-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc); GoToLine(rect,dc);
   StrPCopy(line,'Adres modułu: '+ writehex(s_module.LC0_MBASE1)+ '(hex)');
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   str(s_module.LC0_MIAD,spar) ;
   StrPCopy(line,'Liczba przetworników a/c: ' + spar);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   str(s_module.LC0_MIDA,spar);
   StrPCopy(line,'Liczba przetworników c/a: ' + spar);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   str(s_module.LC0_MIDI,spar);
   StrPCopy(line,'Liczba portów cyfrowych wejściowych: '+ spar);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   str(s_module.LC0_MIDO,spar);
   StrPCopy(line,'Liczba portów cyfrowych wyjściowych: ' +spar);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   str((longint(s_module.LC0_MCLOCK) / 1000):1:0,spar);
   StrPCopy(line,'Częstotliwość zegara modułu: ' + spar + ' MHz');
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);GoToLine(rect,dc);
   StrPCopy(line,'Konfiguracja toru a/c:');
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);GoToLine(rect,dc);
   str(s_info.LC0_GCHAN,spar);
   StrPCopy(line,'Liczba kanałów: ' + spar);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   str(s_info.LC0_GRES,spar);
   StrPCopy(line,'Rozdzielczość: '+ spar +' bitow');
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   str((s_info.LC0_GMINV / 10):3:1,spar);str((s_info.LC0_GMAXV / 10):3:1,sp1);
   StrPCopy(line,'Zakres napięć:' + spar+'..'+ sp1 +' V');
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   StrPcopy(line,'Karta posiada układ S&H ' );
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   StrPCopy(line,'Minimalne okresy próbkowania [us]:');
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
   rect.left := rect.left + 5;
   StrPCopy(line,'');
   spar := '';
   for i := 1 to s_info.LC0_GCHAN do
```

```
          begin
        str((s_info.LC0_GMINP[i] / 10):5:1,sp1);
        if i < s_info.LC0_GCHAN  then
         spar :=spar+ ' '+ sp1 +','
        else
         spar :=spar+ ' '+ sp1 ;
        if i =7 then
         begin
          StrPCopy(line,spar);
          DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
          StrPCopy(line,'');
          spar :='';
               GoToLine(rect,dc);
         end
            end;
   StrPCopy(line,spar);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   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);
   ReleaseDC(dc,vhWnd);
end;
(*************************************************************************)
(* Funkcja : displaytext(HDC,RECT,unsigned char)                         *)
(* Przeznaczenie:                                                        *)
(*   Wyswietlenie komunikatów tekstowych dla transmisji        A/C programu *)
(* Korzysta ze zmiennej paintdet , która określa bieżącą fazę transmisji *)
(* blokowej oraz paintnum , która określa rodzaj transmisji      .       *)
(*************************************************************************)
function displaytext(dc : HDC ;rect : TRECT;start : byte ):integer ;
 var
  line : array[0..200] of char;
  spr  : string[200];
  str1  : string[10];
 begin
  rect.left := rect.left + 10;
  case paintdet of
   1   :begin
       (* blocktransmission *)
       StrPCopy(line,'Poprawne przetwarzanie blokowe .');
       DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
       end;
   2   :begin
      (* failblocktransmission *)
       StrPCopy(line,'Błędne przetwarzanie blokowe.');
            DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
       end ;
   3   :begin
      (* interruptbefore *)
       StrPCopy(line,'Przetwarzanie blokowe z oczekiwaniem 1000s.');
            DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
            GoToLine(rect,dc);
            StrPCopy(line,'Oczekiwanie należy przerwać Ctrl-Break.');
            DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
       end ;
   4  :begin
      (* singletransmission *)
       StrPcopy(line,'Przetwarzanie programowe - po 5 sekundach .');
            DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
       end;
   5  :begin
```

```
      (* singlewrite *)
       StrPCopy(line,'Wysłanie pojedynczej wartości na przetwornik CA.');
            DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
       end;
  end ;
  GoToLine(rect,dc);
  if paintdet < 4 then (* transmisje blokowe : paintdet = 1..4*)
  begin
   GoToLine(rect,dc);
   spr := 'Pomiar blokowy ,  ';str(CHANNELS,str1);
   spr := spr + str1 + ' kanałów, '; str(SAMPLES,str1);
   spr := spr + str1 + ' próbek, ' ; str((s_analog_in.LC0_APER / 10):5:1,str1);
   spr := spr + ' okres próbkowania ' + str1 + ' us :'   ;
   StrPCopy(line,spr);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
  end ;
  case start of
    LC0_SIMMED :StrPCopy(line,'Warunek startu: natychmiast.');
    LC0_STIME  :begin
                str(s_analog_in.LC0_ASTART.time,str1);
                spr :='Warunek startu: upływ '+str1 + ' sekund.' ;
                StrPCopy(line,spr);
               end;
  end ;
  GoToLine(rect,dc);
  DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
  displaytext := rect.top;
 end ;

(*****************************************************************************
  Przeznaczenie:
   Wyswietlenie bufora z danymi pomiarowymi.
   Parametry:
   buf - adres bufora
   c   - liczba kanalow
   s   - liczba probek na kanal
   dc       - kontekt wyswietlania
  rect - okno do wyświetlania
*****************************************************************************)
function displaybuf(var buf:buftype ; c,s : integer ; dc : HDC ; rect : TRECT):integer ;
var
 i,j : integer ;
 line : array[0..200] of char ;
 spr  : string[200];
 spr1 : string[200];
begin
 GoToLine(rect,dc);
 for i:= 0 to s-1 do
  begin
   str(i+1,spr);
   spr :=spr + ':  ';
   for j:=0 to c-1 do
    begin
     spr1:= writehex(buf[i*c+j+1]);
     spr:= spr + spr1 +',  ';
    end;
   StrPCopy(line,spr);
   DrawText(dc,line,-1,rect,DT_LEFT or DT_SINGLELINE);
   GoToLine(rect,dc);
  end;
 displaybuf:=rect.top;
```

```
end;
(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*    Procedura obslugi przerwania pomiaru klawiszem Ctrl-Break.         *)
(*                                                                       *)
(* Sposob:                                                               *)
(*                                                                       *)
(*************************************************************************)
procedure my_break;
begin
   koniec :=1;
   MessageBeep(65535) ;
end;
(*************************************************************************)
(* Funkcja : installbreak                                                *)
(* Przeznaczenie:                                                        *)
(*   Zainstalowanie procedury obslugi przerwania generowanego przez      *)
(*   Ctrl-Break.                                                         *)
(* Sposob:                                                               *)
(*   Przez wywolanie funkcji BREAK.                                      *)
(*************************************************************************)
procedure installbreak;
begin
          s_break.LC0_BMODE := LC0_BREAK_INST;
          s_break.LC0_BPROC := @my_break;         (* podlozona zostanie      *)
                               (*   procedura użytkownika    *)
          TBreak(PBreak)(@s_break);                (* nil *)
end;
(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Wypisanie komunikatow o bledzie / ostrzezeniu / dodatkowej informacji *)
(*   bledzie.                                                            *)
(* Parametry:                                                            *)
(*   status   - LC0_STATUS                                               *)
(*   err_stat - LC0_ERR_STAT                                             *)
(*************************************************************************)
procedure drivererror(status, err_stat: 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,'Błąd (kod): '+ spr);
  MessageBox(vhWnd,line,'Sygnał',MB_OK);
  if err_stat < 0 then
   begin
    str(err_stat,spr);StrPCopy(line,'Informacje dodatkowe(kod): '+spr) ;
    MessageBox(vhWnd,line,'Sygnał',MB_OK);
   end;
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:                                                        *)
(*   Wykonanie transmisji blokowej.                                      *)
(* Sposob:                                                               *)
(*   Przez wykonanie funkcji ANALOG_INPUT.                               *)
(* Parametry:                                                            *)
(*   start - typ warunku startu                                          *)
(* Wartosc:                                                              *)
(* Uwagi:                                                                *)
(*   Funkcja wywolujaca musi ustawic okres probkowania i parametry warunku *)
(*   startu.                                                             *)
(*************************************************************************)
procedure transmission(start: byte);
var
   c: char;
begin
   s_analog_in.LC0_AMODULE := modulenum;
   s_analog_in.LC0_ANUM := 1;
   s_analog_in.LC0_AMODE := LC0_MOD_START or
                            LC0_MOD_NEW_PAR or
                            LC0_MOD_SYNCHR or
                            LC0_MOD_BLOCK;
                               (* stop po zmierzeniu okreslonej liczby    *)
                               (* probek                                  *)
   s_analog_in.LC0_ASTST := start + LC0_ZSAMPLES;
                               (* praca wielokanalowa, CHANNELS kanalow   *)
   s_analog_in.LC0_ACHAN := CHANNELS;
   s_analog_in.LC0_AADDR := @buf;
   s_analog_in.LC0_ALEN := BUFLEN;     (* dlugosc bufora                  *)
   s_analog_in.LC0_ABMAR := 0;
   s_analog_in.LC0_AEMAR := 0;         (* oba marginesy zerowe            *)
                                       (* calkowita liczba probek         *)
   s_analog_in.LC0_ASTOP.samples := SAMPLES * CHANNELS;
   TAnalogInput(PAnalogInput)(@s_analog_in);                    (* ANALOG_INPUT    *)
   if s_analog_in.LC0_STATUS <> LC0_OK then
         begin
            enddet :=1 ;
            drivererror(s_analog_in.LC0_STATUS, s_analog_in.LC0_ERR_STAT) ;
         end
   else
    enddet :=2 ;
   drawscreen;
end;
(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Wykonanie poprawnej transmisji blokowej i wyswietlenie zmierzonych  *)
(*   wartosci.                                                           *)
(* Sposob:                                                               *)
(*   Przez wykonanie funkcji transmission.                               *)
(*************************************************************************)
procedure blocktransmission;
begin
   tstart := LC0_SIMMED;
   paintdet := 1;
   s_analog_in.LC0_APER := s_info.LC0_GMINP[CHANNELS]+200;
   drawscreen;
```

```
   transmission(LC0_SIMMED);
end;
(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Wykonanie blednej transmisji blokowej.                              *)
(* Sposob:                                                               *)
(*   Przez wykonanie funkcji ANALOG_INPUT z okresem probkowania mniejszym*)
(*   niz minimalny wskazany przez driver.                                *)
(*************************************************************************)
procedure failblocktransmission;
begin
   tstart := LC0_SIMMED;
   paintdet := 2;
   s_analog_in.LC0_APER := s_info.LC0_GMINP[CHANNELS] - 1;
   drawscreen;
   transmission(LC0_SIMMED);
end;
(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Wykonanie poprawnej transmisji blokowej ale przerwanej przez operatora*)
(*   (Ctrl-Break) w trakcie czekania na spelnienie warunku startu.       *)
(* Sposob:                                                               *)
(*   Przez wykonanie funkcji ANALOG_INPUT z warunkiem startu LC0_STIME   *)
(*   (start po okreslonym czasie) i parametrem tego warunku - 1000s.     *)
(*************************************************************************)
procedure interruptedbefore;
begin
   tstart := LC0_STIME;
   paintdet := 3;
   s_analog_in.LC0_APER := s_info.LC0_GMINP[CHANNELS];
   s_analog_in.LC0_ASTART.time := 1000;
   drawscreen;
   transmission(LC0_STIME);
end;
(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Wykonanie pomiaru bloku probek za pomoca tranmsisji programowej, przy*)
(*   czym caly blok ma byc zmierzony po 5s od startu.                    *)
(* Sposob:                                                               *)
(*   Przez wykonanie funkcji ANALOG_INPUT.                               *)
(* Parametry:                                                            *)
(* Wartosc:                                                              *)
(* Uwagi:                                                                *)
(*************************************************************************)
procedure singletransmission;
var i : Integer ;
label 1;
begin
   tstart := LC0_STIME ;
   paintdet := 4 ;
   s_analog_in.LC0_AMODULE := modulenum;
   s_analog_in.LC0_ANUM := 1;
   s_analog_in.LC0_AMODE := LC0_MOD_START or
                            LC0_MOD_NEW_PAR or
                            LC0_MOD_SINGLE;
                                (* praca wielokanalowa, CHANNELS kanalow   *)
   s_analog_in.LC0_ACHAN := CHANNELS;
   s_analog_in.LC0_ALEN := CHANNELS;(* dlugosc bufora (tylko na jeden  *)
                                    (* pomiar)                  *)
   s_analog_in.LC0_ASTST := LC0_STIME+ LC0_ZSAMPLES;
   s_analog_in.LC0_ASTART.time := 5;      (* start po 5 sekundach          *)
```

```
   (* calkowita liczba probek        *)
   s_analog_in.LC0_ASTOP.samples := CHANNELS;
   drawscreen;
   for i :=1 to SAMPLES do
    Begin
     s_analog_in.LC0_AADDR := @buf[(i-1)*CHANNELS+1];
     TAnalogInput(PAnalogInput)(@s_analog_in);                    (* ANALOG_INPUT    *)
     (* ustawienie trybu pomiarów natychmiastowych *)
     if i = 1 then s_analog_in.LC0_ASTST := LC0_SIMMED+ LC0_ZSAMPLES;
      (* zgaszenie flagi nowych parametrów *)
     if i = 2 then s_analog_in.LC0_AMODE := LC0_MOD_START or LC0_MOD_SINGLE;
     if s_analog_in.LC0_STATUS <> LC0_OK then
           begin
              enddet :=1 ;
              drivererror(s_analog_in.LC0_STATUS, s_analog_in.LC0_ERR_STAT) ;
         goto 1
           end
     else
      enddet :=2 ;
    end ;
  1: drawscreen;
end;

(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Wyslanie pojedynczej wartosci na przetwornik CA.                    *)
(* Sposob:                                                               *)
(*   Przez wykonanie funkcji ANALOG_OUTPUT.                              *)
(* Parametry:                                                            *)
(* Wartosc:                                                              *)
(* Uwagi:                                                                *)
(*************************************************************************)
procedure singlewrite;
var
       val:  Integer;                    (* wartosc do wyslania          *)
begin
      tstart := LC0_SIMMED ;
           paintdet := 5 ;
           if(s_module.LC0_MIDA = 0) then
           begin
       MessageBox(vhWnd,'Karta nie posiada zainstalowanych przetwornikow CA','Sygnał',MB_OK);
            drawscreen;
            Exit;
           end;
           val := 4095;
           s_analog_out.LC0_NMODULE := modulenum;
           s_analog_out.LC0_NNUM := 1;
           s_analog_out.LC0_NMODE := LC0_MOD_START or
                                     LC0_MOD_NEW_PAR or
                                     LC0_MOD_SYNCHR;
                                          (* praca jednokanalowa     *)
           s_analog_out.LC0_NCHAN := $80 or $01;
           s_analog_out.LC0_NADDR.base_memory := @val;
                                     (*   daleki adres               *)
           s_analog_out.LC0_NLEN := 1;          (* dlugosc bufora (tylko   *)
                                     (*   na jedna wartosc)     *)
           s_analog_out.LC0_NSTST := LC0_SIMMED + LC0_ZSAMPLES;
                                          (* praca jednokanalowa     *)
           s_analog_out.LC0_NSTOP.samples := 1;
                                    (* ANALOG_OUTPUT   *)
      TAnalogOutput(PAnalogOutput)(@s_analog_out);
```

```
          if s_analog_in.LC0_STATUS <> LC0_OK then
           drivererror(s_analog_in.LC0_STATUS, s_analog_in.LC0_ERR_STAT) ;
      enddet :=1 ;
      drawscreen;
end;

(*************************************************************************

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

          Przeznaczenie:   Funkcja odpowiedzi 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

*************************************************************************)

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;
             PostMessage(vhWnd,MY_MESSAGE,0,0);
       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,tstart);
      rect.left:=10;
      if enddet > 0 then
       begin
        if enddet=2 then
```

```
          begin
           rect.top:=displaybuf(buf,CHANNELS, SAMPLES,dc,rect);
          end;
         PressOrClickAny(dc,rect);
        end;
      end ;
    end;
    EndPaint(vhWnd,ps);
   end ;
   WM_LBUTTONDOWN ,WM_KEYDOWN :  (*  goto next*)
   begin
    if paintnum <> 0 then
     begin
      paintnum := paintnum +1 ;
      if paintnum = 7 then
       begin
        paintnum := 0 ;
        drawscreen;
       end
      else
        PostMessage(vhWnd,MY_MESSAGE,0,0);
     end ;
   end ;
   MY_MESSAGE : (* wykonanie kolejnego kroku algorytmu *)
   begin
    enddet := 0;
    case paintnum of
      1:begin
        askdriver;         (* odpytanie driver-a o konfiguracje *)
        drawscreen;        (* wyswietlenie konfiguracji modulu *)
        installbreak;      (* zainstalowanie procedury obslugi CTRL_BRAEK  *)
        end;
      2:blocktransmission;    (* wykonanie transmisji blokowej *)
      3:failblocktransmission;(* wykonanie błędnej transmisji blokowej *)
      4:interruptedbefore;    (*wykonanie transmisji blokowej przerwanej*)
                        (* przez operatora przed startem *)
      5:singletransmission;   (* wykonanie pomiaru programowego *)
      6:singlewrite;          (* wysłanie pojedynczej wartości na DAC*)
    end ;
   end ;
   WM_DESTROY: (* koniec programu *)
    begin
     PostQuitMessage(0);
    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.lpfnWndProc:= @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 bibliotek DLL Windows do kart AMBEX LC',
          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);
  dresult := loaddriver ;         (* załadowanie driver'a *)
  if dresult = TRUE then
   begin
    MessageBox(vhWnd,'Zamknij by rozpocząć przykładową sesję pomiarową .','Informacja',MB_OK);
    paintnum := 1;
    PostMessage(vhWnd,MY_MESSAGE,0,0);
   end;
  while GetMessage(Msg, 0, 0, 0) do
  begin
    TranslateMessage(msg);
    DispatchMessage(msg);
  end;
  if dresult = TRUE then
   quit ;
end ;

begin
  WinMain;
end.
```

D O D A T E K E

d o d o k u m e n t a c j i b i b l i o t e k i D L L

m o d u ł u k o n t r o l n o - p o m i a r o w e g o

L C - 0 1 0 - 1 6 1 2

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 karty LC-010-1612
Uwaga! Ważna jest kolejność parametrów. Postawienie na początku linii znaku ';' spowoduje pominięcie jej podczas analizy parametrów.

[LC-010-1612]		- typ modułu
Path	= 'x'	- 'x' : ścieżka do pliku "ambex.ini"
INTTime	= 'x'	- 'x' : maksymalny czas obsługi przerwania w μs
Modules	= 'x'	- liczba modułów zainstalowanych; 'x' = 1..4
[MODULE 'x']		- 'x' : A,B,C,D - nazwa modułu
AD_Converter	= 'x'	- 'x' : typ przetwornika ADC dla danego modułu: AD574
AD_ConvTime	= 'x'	- 'x' : czas konwersji ADC podany w ns: typ 22000
AD_ConvTable	= x1,x2,...,xn	- tablica minimalnych czasów konwersji ADC w 1,2,...n kanałach, gdzie n - maksymalna liczba kanałów analogowych karty. Czasy powinny być podane w 0.1μs[1].
ExtMemBuffer	='x'	- 'x' : rozmiar bufora pamięci rozszerzonej w KB, gdy 0 - brak pamięci rozszerzonej
ClkFrequency	= 'x'	- 'x': zegar modułu w MHz (8)
IRQNumber	= 'x'	- 'x' : nr przerwania sprzętowy; 0 - brak przerwania
AD_Vmin	= 'x'	- minimalne napięcie zakresowe ADC podane w 0.1 V
AD_Vmax	= 'x'	- maksymalne napięcie zakresowe ADC podane w 0.1 V
DA_Channels	= 'x'	- 'x' : ilość przetworników DAC; 0 - brak

Ostatni parametr należy podać, gdy moduł posiada przetwornik C/A.
DA_ConvTable = x1 - wartość czasu konwersji przetworników DAC podana w 0.1μs.

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

1) Ze względu na specyfikę konstrukcji karty - pomiary wykonywane są w trybie programowym - maksymalna częstotliwość próbkowania dla danej liczby kanałów bardzo zależy od typu komputera oraz konfiguracji środowiska programowego. Tablicę minimalnych czasów konwersji należy wpisać jawnie w pliku konfiguracyjnym. Elementy tablicy konwersji należy dobrać tak, by nie wystąpiły zakłócenia w procesie akwizycji sygnału analogowego. Poniżej przedstawiono typowe wartości elementów tablicy konwersji dla następującej konfiguracji pomiarowej:

komputer z procesorem 80486DX2-80, MS Windows 3.1.

```
AD_Converter    = ADC574A
AD_ConvTime     = 22000
AD_ConvTable    = 400,720,940,1180,1400,1620,1860,2080,2330,2540,2860,3120,3380,3640,3900,4100
```