PRZEMYSŁOWY INSTYTUT AUTOMATYKI I POMIARÓW
MERA-PIAP
Al. Jerozolimskie 202 02-222 Warszawa Telefon 23-70-81

OŚRODEK AUTOMATYZACJI PROCESÓW PRODUKCJI

440 BE10

Główny wykonawca mgr inż. Andrzej Bratek

Wykonawcy dr inż. Marian Wrzesień

Konsultant

Nr zlecenia S 1329

"Opracowanie systemu zbierania danych analogowych z zastosowaniem komputera PC."

Zleceniodawca praca w ramach działalności statutowej PIAP

Pracę rozpoczęto dnia grudzień 92
Kierownik Ośrodka

dr inż. M.Wrzesień

zakończono dnia 31.03.93 r.
Zastępca Dyrektora d/s Badawczo-Rozwojowych

dr inż. J.Jabłkowski

Praca zawiera:

stron 10
rysunków
fotografii
tabel
tablic
załączników 1

Nr rejestr. 6952

Rozdzielnik - ilość egz:

Egz. 1 BOINTE
Egz. 2 OAP a/a
Egz. 3 OAP
Egz. 4
Egz. 5
Egz. 6

Analiza deskryptorowa

BADANIA

Metody pomiarowe

Analiza dokumentacyjna

W sprawozdaniu przedstawiono system zbierania danych analogowych z zastosowaniem komputera PC.

Tytuły poprzednich sprawozdań

UKD

PIAP 41/88 10000

2

1. Wstep

Praca objęta zleceniem S1329 ma na celu wsparcie umowy zewnętrznej nr 79/92/U z dnia 92.12.04 w ramach której opracowuje się system sterowania 16. piecami do wygrzewania tarcz ściernych w Zakładzie Produkcji Ściernic INCO VERITAS w Pruszkowie. Zakres niniejszej pracy dotyczy analizy możliwości zbudowania systemu zbierania danych analogowych i cyfrowych, oraz sterowania dwustawnego na bazie komputera PC. System taki może być stosowany wszędzie tam, gdzie są wymagane niewielkie dokładności przetwarzania (w opisanym systemie stosuje się przetwarzanie 12. bitowe), czy średnia niezawodność (niektóre sterowniki przemysłowe np. PEP MC charakteryzują się czasem pracy międzyawaryjnej rzędu kilku lat). W efekcie uzyskuje się relatywnie tani system, który funkcjonalnie spełnia wymagania ujmujące analizowanie badanego lub sterowanego obiektu, i kttórego sprawność jest porównywalna zesprawnością systemu wytwarzania (w przypadku INCO są to piece z elektrycznie sterowaną pracą grzałek.

2. Zestaw sprzetowy.

Zestaw sprzętowy został dobrany na podstawie następujących wymagań dotyczących sterowania każdego z 16 pieców obiektu:

- sterowanie 2 grzałkami - 2 sygnały dwustawne WY_D,
- sterowanie wentylatorem - 1 sygnał dwustawny WY_D,
- pomiar temperatury - 1 sygnał analogowy WE_AN,
- pomiar stanu pracy wentylatora - 1 sygnał dwustawny WE_D,

Ponadto wymagane są dwa sygnały dwustawne dla obsługi stanów alarmowych całego systemu, wskazywanych sygnałami dźwiękowym i świetlnym.

Łącznie, system wymaga sterowania 50 sygnałami dwustawnymi WY_D oraz pomiaru 16 sygnałów analogowych WE_AN oraz 16 sygnałów dwustawnych WE_D.

Komputer, na bazie którego zestawiono system ma cechy:

- 386SX/16 MHz,
- Monitor MONO, Hercules,

- 1 MB RAM,
- 40 MB HD,
- 1.44 MB FDD,

Komputer ten wyposażono dodatkowo w pakiety:

- Moduł kontrolno pomiarowy LC-010-1612,
- Moduł kontrolno pomiarowy LC-055-PIO,
- Moduł uniwersalny LC-UNI-XT.

Moduły te obsługują odpowiednio:

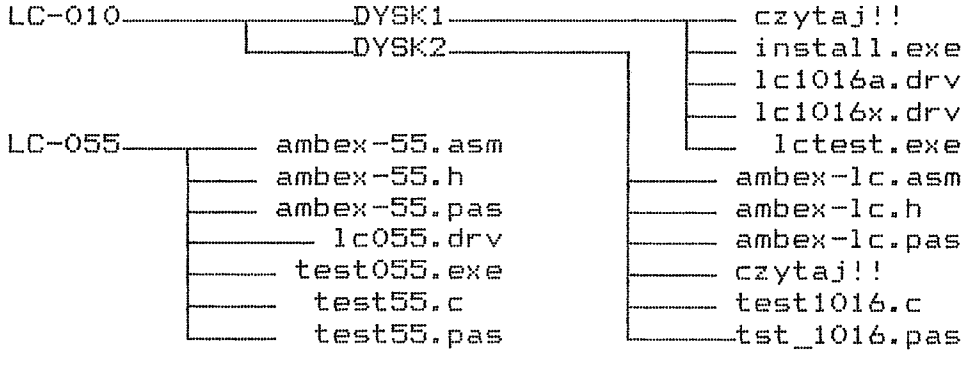
- LC-010-1612: 16 WE_AN oraz 16 WE_D,
- LC-055-PIO: 48 WY_D, oraz
- LC-UNI-XT: 2 WY_D.

Każdy z wymienionych modułów jest dostarczany wraz z instrukcją obsługi oraz, w przypadku modułów kontrolno-pomiarowych, dokumentacjami driverów. Ze względu na obszerność, dokumenty te nie są dołączone do niniejszego sprawozdania. Są one do wglądu u wykonawców niniejszego zlecenia oraz dostępne u producenta pakietów tj.:

AMBEX, Zakład Elektroniki i Pomiarów, Spółka Cywilna,

00-350 Warszawa, ul. Topiel 6

Ponadto, producent dostarcza dyskietkę z takimi plikami, jak:



3. Zasada współpracy z modułami.

Producent przyjął zasadę, że cała komunikacja z modułem prowadzona jest za pośrednictwem rezydentnego programu dostępnego dla programów użytkowych poprzez przerwania programowe. Takie rozwiązanie powoduje, że użytkownik jest zwolniony ze znajomości szczegółow technicznych tak modułu, jak i używanego komputera, oraz rozwiązanie to jest niezależne od używanej implementacji języka

wyższego poziomu. Program obsługi został napisany w standardzie driverów systemu MS-DOS (wersja 3.1 i wyższe), co ułatwia pracę z driverem przy stosowaniu komputera PC.

Funkcje drivera wywoływane są poprzez przerwania programowe. Przesyłanie informacji pomiędzy programem użytkowym, a driverem odbywa się poprzez rekord opisu zlecenia (w języku C odpowiada to strukturze), którego adres jest przekazywany procesorowi przed przerwaniem poprzez rejestry DX:DI.

Rekordy opisu zlecenia zostały zdefiniowane przez producenta i umieszczone w plikach: ambex-55.h i ambex-lc.h.

4. Konstrukcja funkcji obsługujących drivery.

Na podstawie opisu drivera oraz przy wykorzystaniu rekordów zleceń zamieszczonych w plikach nagłówkowych ambex-55.h i ambex-lc.h, opracowano sposób konstruowania funkcji obejmujących obsługę drivera. Zestaw możliwych funkcji pakietu LC-055-PIO, zdefiniowany poprzez kody funkcji drivera, i szczegółowo opisany w instrukcji obsługi, jest następujący:

```
/********** kody funkcji driver'a LC-055-PIO ***********/
#define MODULE_INIT_55                0
#define GET_TOTAL_CONFIGURATION_55    1
#define GET_MODULE_CONFIGURATION_55   2
#define PORT_DIRECTION                3
#define PORT_WRITE                    4
#define PORT_READ                     5
#define PORT_LATCH                    6
#define PORT_RESET                    7
#define CTC_WRITE_55                  8
#define CTC_GATE                      9
#define CTC_READ_55                  10
#define INTERRUPT_MASK               11
#define INTERRUPT_LEVEL              12
#define INTERRUPT_SERVICE_55         13
#define INTERRUPT_RESET              14
#define READ_STATUS                  15
#define LEAVE_DRIVER_55              16
```

natomiast dla pakietu LC-010-1612 funkcje te są następujące:

```
/************* kody funkcji driver'a LC-010-1612***************/
#define MODULE_INIT                   0
#define GET_TOTAL_CONFIGURATION       1
#define GET_MODULE_CONFIGURATION      2
#define GET_INFO                      3
#define SET_CLOCK                     4
#define SET_VOLTAGE_RANGE             5
```

```
#define SET_TIME                    6
#define WAIT_FOR_END                7
#define BREAK                       8
#define DIGITAL_INPUT               9
#define DIGITAL_OUTPUT             10
#define CTC_WRITE                  11
#define CTC_READ                   12
#define DATA_TRANSMIT              13
#define ANALOG_INPUT               14
#define ANALOG_OUTPUT              15
#define LEAVE_DRIVER               16
#define INTERRUPT_SERVICE          17
```

Na potrzeby realizacji sterowania w Zakładach INCO omówione we wstępie, w PIAP OAP opracowano niżej zdefiniowane funkcje. Przy definiowaniu ich korzystano z zalecanego przez producenta środowiska wyznaczonego przez kompilator języka C firmy Borland Inc.

4.1. Moduł LC-055-PIO

1. Funkcja inicjalizująca pakiet.

```
#include <dos.h>
#include "ambex-55.h"
#define LCO_MODA 1
char initialize55(void){
   union REGS wej;
   struct lc0_init_55 s_init;
   s_init.LCO_CODE    = MODULE_INIT_55;
   s_init.LCO_IMODULE = LCO_MODA;
   wej.x.dx = FP_SEG(&s_init);
   wej.x.di = FP_OFF(&s_init);
   int86(LC055, &wej, &wej);
   return(s_init.LCO_STATUS);
}
```

2. Funkcja sprawdzająca zainstalowanie drivera.

```
#include <io.h>
#define TRUE 1
#define FALSE 0
int driverinstalled(const char * driver_name){
  int hd=0;
  if((hd=open(driver_name,0))!=-1)
    close(hd);
  return(hd==-1?FALSE:TRUE);
}
```

3. Funkcja odczytująca informacje ogólne.

```
#include <dos.h>
#include "ambex-55.h"
char get_total_configuration(volatile struct lc0_total_55
                             *s_total){
   union REGS wej;
   s_total->LCO_CODE = GET_TOTAL_CONFIGURATION_55;
```

```
    wej.x.dx = FP_SEG(s_total);
    wej.x.di = FP_OFF(s_total);
    int86(LCO55, &wej, &wej);
    return(s_total->LCO_STATUS);
}
```

4. Funkcja odczytująca informacje o module.

```
#include <dos.h>
#include "ambex-55.h"
char get_module_configuration55(volatile struct lc0_module_55
                                *s_module){
    union REGS wej;
    s_module->LCO_CODE = GET_MODULE_CONFIGURATION_55;
    s_module->LCO_MMODULE = LCO_MODA;
    wej.x.dx = FP_SEG(s_module);
    wej.x.di = FP_OFF(s_module);
    int86(LCO55, &wej, &wej);
    return(s_module->LCO_STATUS);
}
```

5. Funkcja ustawiająca kierunek portów

```
#include <dos.h>
#include "ambex-55.h"
char set_direction55(void){
  union REGS wej;
  struct lc0_port_dir s_port_dir;
  s_port_dir.LC)_CODE   = PORT_DIRECTION;
  s_port_dir.LCO_PMODULE= LCO_MODA;
  s_port_dir.LCO_PDIR   = ((LCO_OUT_DIRECTION|LCO_CHANGE_DIR)<<0)|
                          ((LCO_OUT_DIRECTION|LCO_CHANGE_DIR)<<2)|
                          ((LCO_OUT_DIRECTION|LCO_CHANGE_DIR)<<4)|
                          ((LCO_OUT_DIRECTION|LCO_CHANGE_DIR)<<6)|
                          ((LCO_OUT_DIRECTION|LCO_CHANGE_DIR)<<8)|
                        ((LCO_OUT_DIRECTION|LCO_CHANGE_DIR)<<10|;
  wej.x.dx = FP_SEG(&s_port_dir);
  wej.x.di = FP_SEG(&s_port_dir);
  int86(LCO55, &wej, &wej);
  return(s_port_dir.LCO_STATUS);
}
```

6. Funkcja zapisująca dane cyfrowe.

```
#include <dos.h>
#include "ambex-55.h"
char writeport55(const unsigned char nr_portu, const unsigned char
                 data){
    union REGS wej;
    struct lc0_port_write s_port_write;
    s_port_write.LCO_CODE     = PORT_WRITE;
    s_port_write.LCO_PWMODULE = LCO_MODA;
    s_port_write.LCO_PWPORT   = nr_portu;
    s_port_write.LCO_PWDATA   = data;
    wej.x.dx = FP_SEG(&s_port_write);
    wej.x.di = FP_OFF(&s_port_write);
    int86(LCO55, &wej, &wej);
```

```
   return(s_port_write.LCO_STATUS);
}
```

7. Funkcja odczytująca dane cyfrowe.

```
#include <dos.h>
#include "ambex-55.h"
char readport55(const unsigned char nr_modulu, const unsigned char
                nr_portu, const unsigned char latch,
                volatile unsigned char *data){
   union REGS wej;
   struct lc0_port_read s_port_read;
   s_port_read.LCO_CODE     = PORT_READ;
   s_port_read.LCO_PRMODULE = nr_modulu;
   s_port_read.LCO_PRPORT   = nr_portu;
   s_port_read.LCO_PRLATCH  = latch;
   wej.x.dx = FP_SEG(&s_port_read);
   wej.x.di = FP_OFF(&s_port_read);
   int86(LC055, &wej, &wej);
   *data = s_port_read.LCO_PRDATA;
   return(s_port_read.LCO_STATUS);
}
```

4.2. Moduł LC-010-1612

1. Funkcja inicjalizująca pakiet.

```
#include <dos.h>
#define LC010_16 0x99  /* LC-011-1612 numer przerwania */
#defineMODULE_INIT 0  /* numer funkcji przerwania     */
#define LCO_MODAMAP 1
char initialize10(void){
   union REGS wej;
   struct lc0_init s_init;
   s_init.LCO_CODE    = MODULE_INIT;
   s_init.LCO_IMODULE = LCO_MODAMAP;
   wej.x.dx = FP_SEG(&s_init);
   wej.x.di = FP_OFF(&s_init);
   int86(LC010_16, &wej, &wej);
   return(s_init.LCO_STATUS);
}
```

2. Funkcja sprawdzająca zainstalowanie drivera.

```
#include <io.h>
#define TRUE 1
#define FALSE 0
int driverinstalled(const char * driver_name){
  int hd=0;
  if((hd=open(driver_name,0))!=-1)
    close(hd);
  return(hd==-1?FALSE:TRUE);
}
```

3. Funkcja odczytująca informacje ogólne.

```
#include <dos.h>
#define LC010_16 0x99   /* LC-011-1612 numer przerwania */
```

```
#defineGET_TOTAL_CONFIGURATION 1  /*numer funkcji przerwania */
char get_total_configuration10(volatile struct lc0_total
                                *s_total){
   union REGS wej;
   s_total->LCO_CODE = GET_TOTAL_CONFIGURATION;
   wej.x.dx = FP_SEG(s_total);
   wej.x.di = FP_OFF(s_total);
   int86(LCO10_16, &wej, &wej);
   return(s_total->LCO_STATUS);
}
```

4. Funkcja odczytująca informacje o module.

```
#include <dos.h>
#define LCO10_16 0x99          /* LC-011-1612 numer przerwania */
#define GET_MODULE_CONFIGURATION 2  /*numer funkcji przerwania */
char get_module_configuration10(volatile struct lc0_module
                                 *s_module){
   union REGS wej;
   s_module->LCO_CODE = GET_MODULE_CONFIGURATION;
   s_module->LCO_MMODULE = 1;
   wej.x.dx = FP_SEG(&s_module);
   wej.x.di = FP_OFF(s_module);
   int86(LCO10_16, &wej, &wej);
   return(s_module->LCO_STATUS);
}
```

5. Funkcja odczytująca informacje o submodułach modułu.

```
#include <dos.h>
#define LCO10_16 0x99    /* LC-011-1612 numer przerwania */
#defineGET_INFO 3        /* numer funkcji przerwania     */
#define LCO_MODA 1
#define LCO_NR_PORTU 1
char get_info10(const unsigned char typ, volatile struct lc0_info
                * s_info){
   union REGS wej;
   s_info->LCO_CODE    = GET_INFO;
   s_info->LCO_GTYPE   = typ;
   s_info->LCO_GNUM    = LCO_NR_PORTU;
   s_info->LCO_GMODULE = LCO_MODA;
   wej.x.dx = FP_SEG(&s_info);
   wej.x.di = FP_OFF(&s_info);
   int86(LCO10_16, &wej, &wej);
   return(s_info->LCO_STATUS);
}
```

6. Funkcja zapisująca dane cyfrowe.

```
#include <dos.h>
#define LCO10_16 0x99    /* LC-011-1612 numer przerwania */
#defineDIGITAL_OUTPUT 10          /*numer funkcji przerwania */
char writeport10d(const unsigned char nr_modulu, const unsigned
                  char nr_portu, const unsigned char typ_startu,
                  const unsigned char data){
   union REGS wej;
   struct lc0_digital_out s_port_write;
```

```
    s_port_write.LC0_CODE    = DIGITAL_OUTPUT;
    s_port_write.LC0_DMODULE = nr_modulu;
    s_port_write.LC0_DNUM    = nr_portu;
    s_port_write.LC0_DSTST   = typ_startu;
    s_port_write.LC0_DVAL    = data;
    wej.x.dx = FP_SEG(&s_port_write);
    wej.x.di = FP_OFF(&s_port_write);
    int86(LC010_16, &wej, &wej);
    return(s_port_write.LC0_STATUS);
}
```

7. Funkcja odczytująca dane cyfrowe.

```
#include <dos.h>
#define LC010_16 0x99            /* LC-011-1612 numer przerwania */
#define DIGITAL_INPUT 9           /*numer funkcji przerwania  */
#define LC0_MODA 1
#define LC0_NR_PORTU 1
#define LC0_SIMMED 0
char readport10d(const unsigned char data){
    union REGS wej;
    struct lc0_digital_in s_port_read;
    s_port_read.LC0_CODE    = DIGITAL_INPUT;
    s_port_read.LC0_DMODULE = LC0_MODA;
    s_port_read.LC0_DNUM    = LC0_NR_PORTU;
    s_port_read.LC0_DSTST   = LC0_SIMMED;
    s_port_read.LC0_DVAL    = data;
    wej.x.dx = FP_SEG(&s_port_read);
    wej.x.di = FP_OFF(&s_port_read);
    int86(LC010_16, &wej, &wej);
    return(s_port_read.LC0_STATUS);
}
```

8. Funkcja odczytująca dane analogowe.

```
#include <dos.h>
#include <conio.h>
#define LC010_16 0x99   /* LC-010-1612 numer przerwania */
#defineANALOG_INPUT 14    /*numer funkcji przerwania     */
#include "ambex-lc.h"
#define TEST
struct INFO{
  unsigned long int okres_probkowania;
  unsigned char liczba_kanalow;
};
extern struct INFO dane_pakietu;
char readport10a(void){
    const int liczba_probek = 20;
    unsigned long int y=0L;
    unsigned int tab[16*20];
    register int i=0, j=0;
    extern struct INFO dane_pakietu;
    unsigned long int dlugosc_bufora=
                    dane_pakietu.liczba_kanalow*liczba_probek;
    int buffer[16*20]={0};
    union REGS wej;
    struct lc0_analog_in s_analog_in;
    s_analog_in.LC0_CODE    = ANALOG_INPUT;
```

```
s_analog_in.LCO_AMODULE = 1;
s_analog_in.LCO_ANUM    = 1;
s_analog_in.LCO_AMODE   = LCO_MOD_START|LCO_MOD_NEW_PAR|
                          LCO_MOD_BLOCK;
s_analog_in.LCO_ASTST   = LCO_SIMMED+LCO_ZSAMPLES;
s_analog_in.LCO_APER    = dane_pakietu.okres_probkowania;
s_analog_in.LCO_ACHAN   = dane_pakietu.liczba_kanalow;
s_analog_in.LCO_AADDR   = (int far*)buffer;
s_analog_in.LCO_ALEN    = dlugosc_bufora;
s_analog_in.LCO_ASTOP.samples   =
                  dane_pakietu.liczba_kanalow*liczba_probek;
wej.x.dx = FP_SEG(&s_analog_in);
wej.x.di = FP_OFF(&s_analog_in);
int86(LCO10_16, &wej, &wej);
for(j=0;j<16;j++){
  y=0L;
  for(i=0;i<20;i++)
    y += buffer[16*i+j];
  tab[j] = (unsigned int)y/20;
}
```

4.3. Moduł LC-UNI XT.

Moduł LC-UNI-XT nie wymaga szczególnego oprogramowania, gdyż do jego portów można się dostać za pomocą znanych funkcji outportb() i inportb().

5. Podsumowanie

Wymienione funkcje zostały zaimplementowane i przetestowane na komputerze przeznaczonym do zainstalowania w fabryce tarcz ściernych.

Konstrukcja funkcji zapewnia modułowość programu. Pomimo zachowania definicji rekordów (struktur) zdefiniowanych przez producenta w zbiorach nagłówkowych, powyższe funkcje, jak i rekordy, są klasy auto. Propozycja producenta natomiast zachęcała do stosowania rekordów klasy extern, co niepotrzebnie zajmowałoby pamięć.

W przeciwieństwie do programów wzorcowych producenta, opisane funkcje są wywoływane z parametrami, które są umieszczane w rekordach przez funkcje, a nie, jak to proponowano, w programie głównym.

Obecnie funkcje te włączane są do programu sterującego procesem wygrzewania tarcz ściernych w ww fabryce.

Opracowanie przedstawione w niniejszym sprawozdaniu może zostać wykorzystane przez pracowników PIAP, przed którymi stoi zadanie zbudowania taniego systemu kontrolno-pomiarowego tak na potrzeby przemysłu, jak i przy pracach laboratoryjnych. Poniżej dołączono pełny program umożliwiający współpracę komputera z wyspecyfikowanymi wyżej pakietami.

Załącznik.

```
#include "ambex-55.h"
#include "obiekt.h"
#include "ambex-lc.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

#define TEST
typedef unsigned char uchar;
typedef unsigned long int ulint;

struct INFO{
  unsigned long int okres_probkowania;
  unsigned char liczba_kanalow;
};
char far inicjalizacja_do_pomiaru(void);
int far driverinstalled(const char* drive_name);
char far initialize55(void);
char far set_direction55(void);
char far initialize10(void);
char far get_info10(const unsigned char type,
                    volatile struct lc0_info *s_info);
char far writeport55(const uchar nr_portu, const uchar data);
char far write_all55u(void);
void far read_all55u(volatile unsigned char tablicad[2]);
char far readport10(volatile unsigned int tablica[16]);

PROCES proces[16];
struct INFO dane_pakietu;

int main(){
    unsigned char tablicad[2]={(char)0};
    unsigned int tablicaan[16]={0};
    char status=(char)0;
    clrscr();

    status = inicjalizacja_do_pomiaru();
    cprintf("status init=%d\r\n");
    getch();
    status = write_all55u();
    cprintf("status write=%d\r\n");
    getch();
    read_all55u(tablicad);
    cprintf("status readport10=%d\r\n",status);
    getch();
    status = readport10(tablicaan);
    return((int)status);
}

char far inicjalizacja_do_pomiaru(void){
     char status=(char)NULL;
     struct lc0_info s_info;
     if(!driverinstalled("LC1016^^")){
       cprintf("pakiet LC-010-1612 nie zainstalowany\r\n");
       getch();
       exit(1);
     }
     if(!driverinstalled("LC055^^^")){
```

```
        cprintf("pakiet LC-055 nie zainstalowany\r\n");
        getch();
        exit(1);
      }
      status  = initialize55();
      status |= set_direction55();
      status |= initialize10();
      status |= get_info10(LC0_AINPUT, &s_info);
      dane_pakietu.okres_probkowania=
                          s_info.LC0_GMINP[s_info.LC0_GCHAN-1];
      dane_pakietu.liczba_kanalow=s_info.LC0_GCHAN;
      return(status);
}
#define TRUE 1
#define FALSE 0

int driverinstalled(const char * driver_name){
  int hd=0;
  if((hd=open(driver_name,0))!=-1)
    close(hd);
  return(hd==-1?FALSE:TRUE);
}

char far initialize55(void){
   union REGS wej;
   struct lc0_init_55 s_init;
   s_init.LC0_CODE    = MODULE_INIT_55;
   s_init.LC0_IMODULE = LC0_MODA;
   wej.x.dx = FP_SEG(&s_init);
   wej.x.di = FP_OFF(&s_init);
   int86(LC055, &wej, &wej);
   return(s_init.LC0_STATUS);
}
char far set_direction55(void){
   union REGS wej;
   struct lc0_port_dir s_port_dir;
   s_port_dir.LC0_CODE    = PORT_DIRECTION;
   s_port_dir.LC0_PMODULE = LC0_MODA;
   s_port_dir.LC0_PDIR    =((LC0_OUT_DIRECTION | LC0_CHANGE_DIR)<<0)|
                           ((LC0_OUT_DIRECTION | LC0_CHANGE_DIR)<<2)|
                           ((LC0_OUT_DIRECTION | LC0_CHANGE_DIR)<<4)|
                           ((LC0_OUT_DIRECTION | LC0_CHANGE_DIR)<<6)|
                           ((LC0_OUT_DIRECTION | LC0_CHANGE_DIR)<<8)|
                           ((LC0_OUT_DIRECTION | LC0_CHANGE_DIR)<<10);
   wej.x.dx = FP_SEG(&s_port_dir);
   wej.x.di = FP_OFF(&s_port_dir);
   int86(LC055, &wej, &wej);
   return(s_port_dir.LC0_STATUS);
}
char far initialize10(void){
   union REGS wej;
   struct lc0_init s_init;
   s_init.LC0_CODE    = MODULE_INIT;
   s_init.LC0_IMODULE = LC0_MODAMAP;
   wej.x.dx = FP_SEG(&s_init);
   wej.x.di = FP_OFF(&s_init);
   int86(LC010_16, &wej, &wej);
   return(s_init.LC0_STATUS);
}
char far get_info10(const unsigned char typ,
```

```
                        volatile struct lc0_info * s_info){
    union REGS wej;
    s_info->LC0_CODE    = GET_INFO;
    s_info->LC0_GTYPE   = typ;
    s_info->LC0_GNUM    = 1;
    s_info->LC0_GMODULE = 1;
    wej.x.dx = FP_SEG(s_info);
    wej.x.di = FP_OFF(s_info);
    int86(LC010_16, &wej, &wej);
    return(s_info->LC0_STATUS);
}
char far write_al55u(void){
    register int i=0;
    const int ADRES_UNI_OUT=0X308;
    volatile uchar data1=0, data2=0, data3=0, data4=0,
                   data5=0, data6=0, data7=0, bit01=0, bit10=0;
    volatile char status =0;

    data1=(uchar)(
          ((proces[0].sterow&0x1) <<0) |
          ((proces[0].sterow&0x2) <<0) |
          ((proces[0].sterow&0x4) <<0) |
          ((proces[1].sterow&0x1) <<3) |
          ((proces[1].sterow&0x2) <<3) |
          ((proces[1].sterow&0x4) <<3) |
          ((proces[2].sterow&0x1) <<6) |
          ((proces[2].sterow&0x2) <<6) );
    data2=(uchar)(
          ((proces[2].sterow&0x4) >>2) |
          ((proces[3].sterow&0x1) <<1) |
          ((proces[3].sterow&0x2) <<1) |
          ((proces[3].sterow&0x4) <<1) |
          ((proces[4].sterow&0x1) <<4) |
          ((proces[4].sterow&0x2) <<4) |
          ((proces[4].sterow&0x4) <<4) |
          ((proces[5].sterow&0x1) <<7) );
    data3=(uchar)(
          ((proces[5].sterow&0x2) >>1) |
          ((proces[5].sterow&0x4) >>1) |
          ((proces[6].sterow&0x1) <<2) |
          ((proces[6].sterow&0x2) <<2) |
          ((proces[6].sterow&0x4) <<2) |
          ((proces[7].sterow&0x1) <<5) |
          ((proces[7].sterow&0x2) <<5) |
          ((proces[7].sterow&0x4) <<5) );
    data4=(uchar)(
          ((proces[8].sterow&0x1) <<0) |
          ((proces[8].sterow&0x2) <<0) |
          ((proces[8].sterow&0x4) <<0) |
          ((proces[9].sterow&0x1) <<3) |
          ((proces[9].sterow&0x2) <<3) |
          ((proces[9].sterow&0x4) <<3) |
          ((proces[10].sterow&0x1) <<6) |
          ((proces[10].sterow&0x2) <<6) );
    data5=(uchar)(
          ((proces[10].sterow&0x4) >>2) |
          ((proces[11].sterow&0x1) <<1) |
          ((proces[11].sterow&0x2) <<1) |
          ((proces[11].sterow&0x4) <<1) |
          ((proces[12].sterow&0x1) <<4) |
```

```
            ((proces[12].sterow&0x2) <<4) |
            ((proces[12].sterow&0x4) <<4) |
            ((proces[13].sterow&0x1) <<7) );
    data6=(uchar)(
            ((proces[13].sterow&0x2) >>1) |
            ((proces[13].sterow&0x4) >>1) |
            ((proces[14].sterow&0x1) <<2) |
            ((proces[14].sterow&0x2) <<2) |
            ((proces[14].sterow&0x4) <<2) |
            ((proces[15].sterow&0x1) <<5) |
            ((proces[15].sterow&0x2) <<5) |
            ((proces[15].sterow&0x4) <<5) );
    for(i=0;i<16;i++){
      bit01|=(uchar)((proces[i].sterow&0x8)>>3);
      bit10|=(uchar)((proces[i].sterow&0x10)>>3);
    }
    data7=(uchar)(bit01|bit10);

    status  = writeport55(LC0_PORTA, data1);
    status |= writeport55(LC0_PORTB, data2);
    status |= writeport55(LC0_PORTC, data3);
    status |= writeport55(LC0_PORTD, data4);
    status |= writeport55(LC0_PORTE, data5);
    status |= writeport55(LC0_PORTF, data6);
              outportb(ADRES_UNI_OUT, data7);
    return(status);
}

char far writeport55(const unsigned char nr_portu,
                     const unsigned char data){
   union REGS wej;
   struct lc0_port_write s_port_write;
   s_port_write.LC0_CODE     = PORT_WRITE;
   s_port_write.LC0_PWMODULE = LC0_MODA;
   s_port_write.LC0_PWPORT   = nr_portu;
   s_port_write.LC0_PWDATA   = data;
   wej.x.dx = FP_SEG(&s_port_write);
   wej.x.di = FP_OFF(&s_port_write);
   int86(LC055, &wej, &wej);
   return(s_port_write.LC0_STATUS);
}

void far read_all55u(volatile unsigned char tablica[2]){
    const int ADRES_UNI_IN_LOW=0X300;
    const int ADRES_UNI_IN_HIGH=0X304;

    tablica[0]=inportb(ADRES_UNI_IN_LOW);
    tablica[1]=inportb(ADRES_UNI_IN_HIGH);
}

char far readport10(volatile unsigned int tab[16]){
   char status=0;
   const int liczba_probek = 20;
   unsigned long int y=0L;
   register int i=0, j=0;
   extern struct INFO dane_pakietu;
   unsigned long int dlugosc_bufora=
                     dane_pakietu.liczba_kanalow*liczba_probek;
   int buffer[16*20]={0} ;
   union REGS wej;
```

```
    struct lc0_analog_in s_analog_in;
    s_analog_in.LCO_CODE    = ANALOG_INPUT;
    s_analog_in.LCO_AMODULE = 1;
    s_analog_in.LCO_ANUM    = 1;
    s_analog_in.LCO_AMODE   = LCO_MOD_START|LCO_MOD_NEW_PAR|
                              LCO_MOD_BLOCK;
    s_analog_in.LCO_ASTST   = LCO_SIMMED+LCO_ZSAMPLES;
    s_analog_in.LCO_APER    = dane_pakietu.okres_probkowania;
    s_analog_in.LCO_ACHAN   = dane_pakietu.liczba_kanalow;
    s_analog_in.LCO_AADDR   = (int far*)buffer;
    s_analog_in.LCO_ALEN    = dlugosc_bufora;
    s_analog_in.LCO_ASTOP.samples   =
                    dane_pakietu.liczba_kanalow*liczba_probek;
    wej.x.dx = FP_SEG(&s_analog_in);
    wej.x.di = FP_OFF(&s_analog_in);
    int86(LCO10_16, &wej, &wej);
    status=s_analog_in.LCO_STATUS;
    for(j=0;j<16;j++){
      y=0L;
      for(i=0;i<20;i++)
        y += buffer[16*i+j];
      tab[j] = (unsigned int)(0x800-(y/20));
    }
  #ifdef TEST
    clrscr();
    for(j=0;j<dane_pakietu.liczba_kanalow;j++)
      cprintf("%03X ",tab[j]);
    cprintf("\r\n");
    getch();
  #endif
  return(status);
}
```