ISO-DA16/DA8

DOS Software Manual

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assume no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only maybe registered trademarks of their respective companies.

License

The user can use, modify and backup this software **on a single**

machine. The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.

Table of Contents

I. Introduction

The ISODA?.lib is a collection of data acquisition subroutines for ISO-DA16/DA8. These subroutines are written with C language and perform a variety of data acquisition operations.

The subroutines in ISODA?.lib are easy understanding as its name standing for. It provides powerful, easy-to-use subroutine for developing your data acquisition application. To speed-up your developing process, some demonstration C source program are provided.

This driver can support 8 cards maximum in one PC based system.

Support 8 cards in one PC system maximum

I. Installation

It is recommended to install the ISO-DA16/DA8 DOS software to your hard disk to get the best performance. Before beginning, to make a backup copy of the ISO-DA16/DA8 DOS software. Store the original diskette in a safe place. The ISO-DA16/DA8 DOS disk includes the following files:

- \NAPDOS\ISO-LD*.* : for ISO-LDH, ISO-LDL.
- \NAPDOS\ISO-AD32*.* : for ISO-AD32H, ISO-AD32L.
- \NAPDOS\ISODA*.* : for ISO-DA16, ISO-DA8
- \README.DOC

  The contents of readme.doc is given as following:

The NAPDOS is designed for all ICP DAS data acquisition cards.
The NAPDOS can be divided into seven different disks as following and only
one disk is shipped with one card.

********************* DISK 1 (for A821) *********************
The NAPDOS:DISK 1 is for A-821PGH/PGL Multifunction board C langurage library.

Includes:
DIAG : A82XDIAG.EXE provides A-821PG function testing & calibration
DEMO : A-821PGH/PGL Demo program provides TC / MS-C / BC source code
& execution file
LIB : C Langurage Library

********************* DISK 2 (for A822) *********************
The NAPDOS:DISK 2 is for A-822PGH/PGL Multifunction board C langurage library.

Includes:
DIAG : A82XDIAG.EXE provides A-822PG function testing & calibration
DEMO : A-822PGH/PGL Demo program provides TC / MS-C / BC source code
& execution file
LIB : C Langurage Library

********** DISK 3 (for A823) **********

The NAPDOS:DISK 3 is for A-823PGH/PGL Multifunction board C langurage library.

Includes:

DIAG : A823DIAG.EXE provides A-823PG function testing & calibration

DEMO : A-823PGH/PGL Demo program provides TC / MS-C / BC source code & execution file

LIB : C Langurage Library

********** DISK 4 (for DIO&DA) **********

The NAPDOS:DISK 4 is for ICP DAS DIO&DA cards.

includes :

(1) A-626 : 6 Channel Analog Output Board

(2) A-628 : 8 Channel Analog Output Board

(3) DIO-24 : 24 Channel Digital I/O Board

(4) DIO-48 : 48 Channel Digital I/O Board

(5) DIO-64 : 64 Channel Digital I/O Board

(6) DIO-144 : 144 Channel Digital I/O Board

(7) P8R8DIO : 8 Channel Isolation Digital Input
8 Channel Relay Output Board

(8) P16R16DIO: 16 Channel Isolation Digital Input
16 Channel Relay Output board

(9) TMC-10 : 10 Channel Timer/Counter Board

(10) DB-889 : 16 Channel Multiplexer Board

Note :

File name.EXE Demo program Execution files

File name.PRJ Demo program Turbo C Projet file

File name.C Demo program Turbo C Source Code

File name.CPP Demo program C++ Source Code

File name.BAS Demo program Quick Basic Source code

********** DISK 5 (for MMICON) **********

The NAPDOS:DISK 5 is for MMI-CON.

********************** DISK 6 (for A-826) **********************

The NAPDOS:DISK 6 is for A-826PGH/PGL Multifunction board C langurage library.

Includes:

DIAG : A82XDIAG.EXE provides A-826PG function testing & calibration

DEMO : A-826PGH/PGL Demo program provides TC / MS-C / BC source code & execution file

LIB : C Langurage Library

********************** DISK 7 (for ISO series) **********************

The NAPDOS:DISK 7 is for ISO-AD32H/L, ISO-DA16/8, ISO-LDH/L.

If the program is not as same as the "User's Manual", this disk contains the correct program.

Therefore the label of companion floppy disk is “NAPDOS DISK 7”

The related software is given as following:

- \NAPDOS\ISODA\TC\LIB : for ISO-DA16/DA8, TC, library
- \NAPDOS\ISODA\TC\DIAG : for ISO-DA16/DA8, TC, diagnostic program
- **\NAPDOS\ISODA\TC\driver: for ISO-DA16/DA8, TC, driver source program**

A. Compiler & link using TC

- The including file is **ISODA.H**
- There are 5 different model library files : **ISODAS.LIB,…..,ISODAH.LIB**
- Support TC 2.x compiler
- Use text editor to create a project file include : program.c ISODA?.lib
- Use TC integrated environment to **select the correct compiler model**
- demo program with completely source in the companion floppy disk

A. Floppy Contents

The contents of company floppy disk is given as following:

\README.DOC symbol 224 \f "Wingdings" \s 12→❝ read me file

\NAPDOS\ISODA\TC\LIB*.lib symbol 224 \f "Wingdings" \s 12→❝ library for TC 2.X

\NAPDOS\ISODA\TC\DIAG*.lib symbol 224 \f "Wingdings" \s 12→❝ Diagnostic and demo program

The completely source listing of diagnostic program is given in TC format. This program is compiler in LARGE mode and link with ISODAL.lib in TC.

I. C Language Library

A. ISODA.H

```
#define NoError          0
#define SysNoOpen            1
#define ActiveError          2
#define TimeOut          3
#define SysOpenError     4
#define CheckError       5
#define BoardError       6
#define ChannelError     7
#define IrqError         8

extern short    ISODA_ShortSub2(short nA, short nB);
extern float    ISODA_FloatSub2(float fA, float fB);
extern WORD   ISODA_GetVersion(void);
extern WORD   ISODA_CheckBoard(WORD wBoard, WORD wBase, WORD wIrq);
extern WORD   ISODA_ActiveBoard(WORD wBoard);
extern WORD   ISODA_ReadActive(WORD *wActive, WORD *wBase, WORD
                  *wIrq);
extern WORD   ISODA_Do(WORD wDo);
extern WORD   ISODA_Di(WORD *wDi);
extern WORD   ISODA_ReadPowerOnValue(WORD wChannel, WORD *wValue);
extern WORD   ISODA_SetPowerOnValue(WORD wChannel, WORD wValue);
extern WORD   ISODA_ReadRom(WORD wAddress, WORD *wValue);
extern WORD   ISODA_WriteRom(WORD wAddress, WORD wValue);
extern WORD   ISODA_AnalogOutput(WORD wChannel, WORD wValue);
extern WORD   ISODA_SetTimer(float timeslice);
extern WORD   ISODA_ReadTimer(float *timeslice);
extern WORD   ISODA_StartTimer(void);
extern WORD   ISODA_StopTimer(void);
extern WORD   ISODA_DelayMs(WORD wMs);
```

A. TEST Function

short ISODA_ShortSub2(short nA, short nB);
float ISODA_FloatSub2(float fA, float fB);
WORD ISODA_GetVersion(void);

ISODA_ShortSub2

- **Description :**

Compute C=nA-nB in **short** format, **short=16 bits sign integer.** This function is provided for testing purpose.

- **Syntax :**

short ISODA_ShortSub2(short nA, short nB);

- **Input Parameter :**

nA : short integer
nB : short integer

- **Return Value :**

return=nA-nB symbol 224 \f "Wingdings" \s 12→“ short integer

ISODA_FloatSub2

- **Description :**

Compute C=A-B in **float** format, **float=32 bits floating pointer number.** This function is provided for testing purpose.

- **Syntax :**

float ISODA_FloatSub2(float fA, float fB);

- **Input Parameter :**

fA : floating point value
fB : floating point value

- **Return Value :**

return=fA-fB symbol 224 \f "Wingdings" \s 12→“ floating point value

ISODA_GetVersion

- **Description :**

  Read the software version
- **Syntax :**

  WORD ISODA_GetVersion(void) ;
- **Input Parameter :**

  void
- **Return Value :**

  return=0x100 symbol 224 \f "Wingdings" \s 12→❝ Version 1.0

A. ISODA_CheckBoard

- **Description :** This function will check the hardware board. If all checks are OK, this function will active this board. This software can support 8 cards at most in one PC system. **The program must call this function to check each board first, then call *ISODA_ActiveBoard* to select which board is active. All the other functions are referd to the active board only.** If only one board is used, the unique board will be active after this function is called.
- **Syntax :**

  WORD ISODA_CheckBoard(WORD wBoard, WORD wBase, WORD wIrq);
- **Input Parameter :**

  wBoard : board number, 0 to 7, support at most 8 cards in one PC system.

  wBase : board base address

  wIrq : board IRQ number (no support in this version software)
- **Return Value :**

  BoardError : validate board number from 0 to 7

  IrqError : invalidate IRQ number

  CheckError : check board error (maybe base address error)

  TimeOut : check board timeout (maybe base address or hardware error)

NoError : OK

- **Demo Program :** Refer to Chapter 4

A. ISODA_ActiveBoard

- **Description :** Active the specified board (defined by **wBoard**)
- **Syntax :** WORD ISODA_ActiveBoard(WORD wBoard);
- **Input Parameter :**

  wBoard : board number, 0 to 7, support at most 8 cards in one PC system.
- **Return Value :**

  BoardError : validate board number from 0 to 7

  ActiveError : this board must call **ISODA_CheckBoard** first

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_ReadActive

- **Description :** Read the information of current active board.
- **Syntax :** WORD ISODA_ActiveBoard(WORD *wBoard, WORD *wBase, WORD *wIrq);
- **Input Parameter :**

  *wBoard, *wBase, *wIrq : address of wBoard, wBase, wIrq
- **Return Value :**

  ActiveError : no board is active

  NoError : OK

- **Demo Program :** Refer to Chapter 4

A. ISODA_Di

- **Description :** Read the 16 channels of D/I data.
- **Syntax :** WORD ISODA_Di(WORD *wDi);
- **Input Parameter :**

  *wDi : address of wDi
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_Do

- **Description :** write data to the 16 channels of D/O.
- **Syntax :** WORD ISODA_Do(WORD wDo);
- **Input Parameter :**

  wDo : value send to D/O
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_ReadPowerOnValue

- **Description :** Read the 14-bit D/A power-on value
- **Syntax :** WORD ISODA_ReadPowerOnValue(WORD wChannel, WORD *wValue);
- **Input Parameter :**

  wChannel : D/A channel number, from 0 to 15

  *wValue : 14-bit D/A data
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_SetPowerOnValue

- **Description :** Set the power-on value of D/A.
- **Syntax :** WORD ISODA_SetPowerOnValue(WORD wChannel, WORD wValue);
- **Input Parameter :**

  wChannel : D/A channel number, from 0 to 15

  wValue : 14-bit data send to D/A
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

NoError : OK

- **Demo Program :** Refer to Chapter 4

A. ISODA_ReadRom

- **Description :** Read EEPROM data
- **Syntax :** WORD ISODA_ReadRom(WORD wAddress, WORD *wValue);
- **Input Parameter :**

  wAddress : EEPROM address, from 0 to 63

  *wValue : 16-bit data
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_WriteRom

- **Description :** Write data to EEPROM
- **Syntax :** WORD ISODA_WriteRom(WORD wAddress, WORD wValue);
- **Input Parameter :**

  wAddress : address of EEPROM, from 0 to 15

  wValue : 16-bit data
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK

- **Demo Program :** Refer to Chapter 4

A. ISODA_AnalogOutput

- **Description :** Write 14-bit data to D/A converter
- **Syntax :** WORD ISODA_AnalogOutput(WORD wChannel, WORD wValue);
- **Input Parameter :**

  wChannel : channel of D/A, from 0 to 15

  wValue : 14-bit data
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_SetTimer

- **Description :** Set the time constant of timer in ISO-DA16/DA8. This is a machine independent timer.
- **Syntax :** WORD ISODA_SetTimer(float fTimeSlice);
- **Input Parameter :**

  fTimeSlice : time constant, unit=ms
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK

- **Demo Program :** Refer to Chapter 4

A. ISODA_ReadTimer

- **Description :** Read back the time constant of timer in ISO-DA16/DA8.
- **Syntax :** WORD ISODA_ReadTimer(float *fTimeSlice);
- **Input Parameter :**

  *fTimeSlice : address of fTimeSlice, time constant, unit=ms
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_StartTimer

- **Description :** Start the timer in ISO-DA16/DA8.
- **Syntax :** WORD ISODA_StartTimer(void);
- **Input Parameter :**

  void
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK

- **Demo Program :** Refer to Chapter 4

A. ISODA_StopTimer

- **Description :** Stop the timer in ISO-DA16/DA8.
- **Syntax :** WORD ISODA_StopTimer(void);
- **Input Parameter :**

  void
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

  NoError : OK
- **Demo Program :** Refer to Chapter 4

A. ISODA_DelayMs

- **Description :** Machine independent timer function
- **Syntax :** WORD ISODA_DelayMs(WORD wMs);
- **Input Parameter :**

  wMs : number of ms to delay
- **Return Value :**

  ActiveError : this board must call **ISODA_CheckBoard** first

  CheckError : communication handshake error

  TimeOut : communication time out

NoError : OK

- **Demo Program :** Refer to Chapter 4

I. Diagnostic Program

The completely source of diagnostic program are given in company floppy disk.

\README.DOC symbol 224 \f "Wingdings" \s 12→❝ read me file

\NAPDOS\ISODA\TC\LIB*.lib symbol 224 \f "Wingdings" \s 12→❝ library for TC 2.X

\NAPDOS\ISODA\TC\DIAG*.* symbol 224 \f "Wingdings" \s 12→❝ Diagnostic program, refer to Sec. 4.1

A. Files

The completely source listing is given in **\NAPDOS\ISODA\TC\DIAG*.***

TEST.C symbol 224 \f "Wingdings" \s 12→❝ main program

TEST.PRJ symbol 224 \f "Wingdings" \s 12→❝ TC project file

TEST.EXE symbol 224 \f "Wingdings" \s 12→❝ DOS execution file

ISODAL.LIB symbol 224 \f "Wingdings" \s 12→❝ TC large mode library

ISODA.H symbol 224 \f "Wingdings" \s 12→❝ header file

ISODA.DAT symbol 224 \f "Wingdings" \s 12→❝ data file (stored the hardware configuration data)

UART.C symbol 224 \f "Wingdings" \s 12→❝ for ISODA calibration kits

USRT.H symbol 224 \f "Wingdings" \s 12→❝ for UART.C

The source program is too large to giving in this manual. Refer to the companion floppy disk “ NAPDOS DISK-7” for these files.

A. Functions of diagnostic program

1. ISODA.DAT

The hardware configuration setting is stored in this file. The TEST.EXE will automatically read this file and set the correct configuration setting based on this file. The contents of “ISODA.DAT” is giving as following:

220
15
16
3000

220 : hardware base address, default=220
15 : select IRQ channel 15, default=15, if no IRQ, set this value 15
16 : 16 for ISO-DA16 and 8 for ISO-DA8
3000 : wDelay1. This value is used to implement a machine dependent timer. This value is suitable for Pentium 120, the user can increase this value if the more powerful PC is used or decrease this value for the less powerful PC. This value is not expected to be very precise. The user can use function_0 of diagnostic program to test this value.

A. Fun_0 : Machine Dependent Timer Test

This function will perform the software library test & show the version number of software library. This function will test the wDelay1, machine dependent timer constant introduced in Sec. 4.2. The special mark, @, will appear in the screen every second. If the wDelay1 is OK, the user will find the @ appear in the screen every second. The default value of wDelay1 is 3000, this value is suitable for Pentium-120. If more powerful PC is used, the wDelay1 should be increased and decreased if less powerful PC is used. The

wDelay1 is defined in **ISODA.DAT**, the user can use TEXT EDITOR to change this value.

A. Fun_1 : D/I/O Test

This function will send data to D/O port of ISO-DA16/DA8 and read the D/I data. If the user connecting a 20-pin flat cable between CN1 and CN2, the D/O value and D/I value will be the same. This function will check these 2 value. This function will stop and beep if these 2 value are not equal. Therefore please connecting a 20-pin flat cable between CN1 and CN2 before test this function.

A. Fun_2 : Machine Independent Timer Test

This function will test the machine independent timer of ISO-DA16/DA8. It is very similar to function_0 except this function can executed under all PC and will get the same result. Therefore we call it “Machine Independent Timer”. This function will call **ISODA_DelayMs(1000)** to delay 1000 ms=1 second. Therefore the user will find the @ mark will appear in the screen every second until the user press any key to stop.

A. Fun_3 : Write To D/A EEPROM

This function will write the test data into the 64 word EEPROM on the ISO-DA16/DA8. The EEPROM is used to store the calibration data, therefore **the user should not write any data into EEPROM.**

NOTE : The calibration data is stored in the EEPROM, don’t write any data to EEPROM

A. Fun_4 : Read D/A EEPROM

This function will read out the 64 word data from ISO-DA16/DA8 on-board EEPROM.

A. Fun_5 : Set D/A Poweron Value

This function will set D/A power-on value of ISO_DA16/DA8. This function only write 14-bit binary data into EEPROM.

A. Fun_6 : Read D/A Poweron Value

This function will read out the D/A power-on 14-bit data of ISO-DA16/DA8 from EEPROM.

A. Fun_7/8/9/A : Calibration

Function_7 : D/A -10V calibration, save in wMinV[0..15]
Function_8 : D/A +10V calibration, save in wMaxV[0..15]
Function_9 : D/A 0mA calibration, save in wMinI[0..15]
Function_A : D/A 20mA calibration, save in wMaxI[0..15]

All these functions must used with “ISO-DA16/DA8 calibration kits”. These 4 functions will automatically find the calibration data then store these data into EEPROM.

A. Fun_B : Load Calibration Data from EEPROM

The calibration data are divided into 4 groups as following:

wMinV[0..15] : store the 14-bit data for -10V voltage output

wMaxV[0..15] : store the 14-bit data for +10V voltage output

wMinI[0..15] : store the 14-bit data for 0mA current output

wMaxI[0..15] : store the 14-bit data for 20mA current output

This function will load these data from EEPROM.

A. Fun_C : Read Voltage Output of All 16 Channels

This functions must used with “ISO-DA16/DA8 calibration kits”.

A. Fun_D : Read Current Output of All 16 Channels

This functions must used with “ISO-DA16/DA8 calibration kits”.

A. Fun_E : Send To D/A

This functions will send a 14-bit data to all D/A Converter. The D/A converter will output voltage and current at the same time, therefore this function will effect both the voltage and current output. This function will very useful for testing.

A. Fun_F : Voltage Output Test

Before execute this function, the wMinV[0..15] & wMaxV[0..15] must be correct. We call wMinV[0..15] & wMaxV[0..15] "Calibration data for voltage output". The user can execute function_7 & function_8 to get these data automatically if the "ISO-DA16/DA8 calibration kits" is present. **The user also can download these data from EEPROM by function-B.**

This function will control all 16 channels the D/A voltage output to the same value defined by user. For example, if the user key-in 5.432, this function will compute the 14-bit data based on wMinV[0..15] & wMaxV[0..12] and send this 14-bit data to D/A converter. Then user can use voltage meter to measure this value for test and find it is 5.432 volt. If the "ISO-DA16/DA8 calibration kits" is present, the user can use function-c to show voltage output value of all 16 channels.

A. Fun_G : Current Output Test

Before execute this function, the wMinI[0..15] & wMaxI[0..15] must be correct. We call wMinI[0..15] & wMaxI[0..15] "Calibration data for current output". The user can execute function_9 & function_a to get these data automatically if the "ISO-DA16/DA8 calibration kits" is present. **The user also can download these data from EEPROM by function-B.**

This function will control all 16 channels the D/A current output to the same value defined by user. For example, if the user key-in 5.432, this function will compute the 14-bit data based on wMinI[0..15] & wMaxI[0..12] and send this 14-bit data to D/A converter. Then user can use current meter to measure this value for test and will find it is 5.432mA. If the "ISO-DA16/DA8 calibration kits" is present, the user can use function-d to show current output value of all 16 channels.

A. Fun_H : Save EEPROM to File

This function will save the on-board EEPROM into file on disk. If the EEPROM is destroy by accident, this file can be used to down load EEPROM from file.

A. Fun_I : Load EEPROM from File

If the EEPROM is destroy by accident, this function can be used to down load

EEPROM from file. Therefore we recommend the user to save the EEPROM into file before using any ISO-DA16/DA8.

I. Driver Source program

The driver source is giving in the companion floppy disk as following:

- \NAPDOS\ISODA\TC\driver\isoda.c: driver source program
- \NAPDOS\ISODA\TC\driver\isoda.h: header file
- \NAPDOS\ISODA\TC\driver\isoda.prj: project file

Refer to “ISO-DA16/DA8 Hardware Manual” for I/O control register and command sets format.

A. Driver Source

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include "isoda.h"

#define MAX_ROM      64    /* Size of SEEPROM 93C46, 64*16 bit */

#define DIO_LSB      BASE
#define DIO_MSB      BASE+1
#define DO_ENABLE    BASE+2
#define DA_CTRL      BASE+3
#define STATUS       BASE+2

/* Status */
#define READ_READY       1     /* Return value 0001 */
#define WRITE_READY      2     /* Return value 0010 */
#define CON_STATUS       4     /* */
```

```
#define INT_PC              8     /* */
```

```
/* Command list */
#define DA_CMD_MASK    0x90
#define WRITE_DA_0   DA_CMD_MASK+0x00
#define WRITE_DA_1   DA_CMD_MASK+0x01
#define WRITE_DA_2   DA_CMD_MASK+0x02
#define WRITE_DA_3   DA_CMD_MASK+0x03
#define SET_PWR_ON  DA_CMD_MASK+0x04
#define READ_PWR_ON     DA_CMD_MASK+0x05

#define TIMER_CMD    DA_CMD_MASK+0x0B
#define READ_ROM     DA_CMD_MASK+0x0C
#define WRITE_ROM    DA_CMD_MASK+0x0D
#define FIRM_VER       DA_CMD_MASK+0x0E
#define IO_TEST    DA_CMD_MASK+0x0F

/* Receive code */
#define UNKNOW_CMD     '?'
#define CMD_OK               '!'
#define RET_VALUE          '>'

WORD BASE=0,IRQ=0,ACTIVE=8,OPEN=1;
WORD wBaseAddr[8],wIrqNum[8],wActive[8];

/* ---------------------------------------------------------------- */

short ISODA_ShortSub2(short nA, short nB)
{
return(nA-nB);
}

/* ---------------------------------------------------------------- */

float ISODA_FloatSub2(float fA, float fB)
{
return(fA-fB);
}
```

```
WORD ISODA_GetVersion(void)
{
return(0x0100);
}

/* ------------------------------------------------------ */
/*
  wIrq          --> IRQ channel number
  wBoard  = 0..7
*/

WORD  ISODA_CheckBoard(WORD wBoard, WORD wBase, WORD wIrq)
{
WORD i,j,wStatus,wRetVal;

if (wBoard>=8) return(BoardError);      /* wBoard = 0..7 */

if ((wIrq<3) && (wIrq!=0)) return(IrqError);
else if (wIrq==8) return(IrqError);
else if (wIrq==13) return(IrqError);
else if (wIrq>15) return(IrqError);

BASE=wBase;
i=0;
for (;;)
   {
   wStatus = inportb(STATUS) & 0x07;
   if (wStatus==READ_READY)
     {
     inportb(BASE+3);
     i++; if (i>100) return(TimeOut);
     }
   else break;
      }

for (i=0; i<256; i++)
```

```
    {
    if (send_command_2(IO_TEST, i) != NoError)return(TimeOut);
    if (receive_data(&j) != NoError) return(TimeOut);
    if (j != '>') return(CheckError);       /* first char return */
    if (receive_data(&j) != NoError) return(TimeOut);
    if (i != j) return(CheckError);
    }

inportb(BASE+4);          /* clear interrupt signal */
wRetVal=wait_int_ready();
if (wRetVal != NoError) return(CheckError);

wBaseAddr[wBoard]=wBase; BASE=wBase;
wIrqNum[wBoard]=wIrq;       IRQ=wIrq;
wActive[wBoard]=1;
ACTIVE=wBoard;
return(NoError);
}

/* ------------------------------------------------------ */

WORD  ISODA_ActiveBoard(WORD wBoard)
{
if (wBoard>=8) return(BoardError);      /* wBoard = 0..7 */
if (wActive[wBoard]!=1) return(ActiveError);

ACTIVE=wBoard;
BASE=wBaseAddr[wBoard];
IRQ=wIrqNum[wBoard];
return(NoError);
}

/* ------------------------------------------------------ */

WORD  ISODA_ReadActive(WORD *wBoard, WORD *wBase, WORD *wIrq)
{
```

```
if (ACTIVE>=8) return(ActiveError);

*wBoard=ACTIVE;
*wBase=BASE;
*wIrq=IRQ;
return(NoError);
}

WORD ISODA_Do(WORD wDo)
{
if (BASE==0) return(ActiveError);

outportb(BASE+2,0);  /* out any value to enable DO */
outportb(BASE, wDo&0xff);
wDo=wDo>>8;
outportb(BASE+1, wDo&0xff);
return(NoError);
}

/* ---------------------------------------------------------------- */

WORD ISODA_Di(WORD *wDi)
{
WORD h,l;

if (BASE==0) return(ActiveError);

l = inportb(BASE) & 0xff;
h = (inportb(BASE+1) & 0xff)<<8;
*wDi = l+h;
return(NoError);
}

/* ---------------------------------------------------------------- */

WORD send_command(WORD command)
```

```
{
if (wait_status(WRITE_READY)!=NoError) return(TimeOut);
outportb(DA_CTRL, command&0x0ff);
return(NoError);
}
```

```
WORD receive_data(WORD *data)
{
if (wait_status(READ_READY)!=NoError) return(TimeOut);
*data = inportb(DA_CTRL)&0x0ff;
return(NoError);
}

WORD wait_status(WORD in_status)
{
WORD status;
long t=0;
float f;

while (t<100000)
    {
    status = inportb(STATUS) & 0x07;
    if (status!=in_status) t++;
    else
      {
      inportb(STATUS); /* delay the settling time for */
      inportb(STATUS); /* embedded controller */
      return(NoError);
      }
    };

return(TimeOut);
}

WORD wait_int_ready(void)
{
WORD status;
long t=0;
float f;

while (t<100000)
    {
```

```
    status = inportb(STATUS) & 0x08;
    if (status!=0) t++;
    else return(NoError);
    };
return(TimeOut);
}

WORD send_command_4(WORD command, WORD arg1, WORD arg2, WORD arg3)
{
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, command&0x0ff);
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, arg1&0x0ff);
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, arg2&0x0ff);
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, arg3&0x0ff);
return(NoError);
}

WORD send_command_3(WORD command, WORD arg1, WORD arg2)
{
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, command&0x0ff);
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, arg1&0x0ff);
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, arg2&0x0ff);
return(NoError);
}

WORD send_command_2(WORD command, WORD arg)
{
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
outportb(DA_CTRL, command&0x0ff);
if (wait_status(WRITE_READY)==TimeOut) return(TimeOut);
```

```
outportb(DA_CTRL, arg&0x0ff);
return(NoError);
}
```

```
WORD ISODA_ReadPowerOnValue(WORD wChannel, WORD *wValue)
{
WORD rr, data;

if (BASE==0) return(ActiveError);
if (wChannel >= 16) return(ChannelError);

rr = send_command_2(READ_PWR_ON, wChannel);
if (rr != NoError) return(TimeOut);

rr = receive_data(&data);
if (rr != NoError) return(TimeOut);
if (data != RET_VALUE) return(CheckError);

rr = receive_data(&data);
if (rr != NoError) return(TimeOut);
*wValue = data;

rr = receive_data(&data);
if (rr != NoError) return(TimeOut);
*wValue = (*wValue)*256 + data;
return(NoError);
}

/* ---------------------------------------------------------------- */

WORD ISODA_SetPowerOnValue(WORD wChannel, WORD wValue)
{
WORD rr, data, msb, lsb;

if (BASE==0) return(ActiveError);
if (wChannel >= 16) return(ChannelError);

msb = wValue/256;
lsb = wValue%256;
```

```
rr = send_command_4(SET_PWR_ON, wChannel, msb&0x0ff, lsb&0x0ff);
if (rr != NoError) return(TimeOut);
rr = receive_data(&data);
if (rr != NoError) return(TimeOut);
if (data != CMD_OK) return(CheckError);

return(NoError);
}

/* ---------------------------------------------------------------- */

WORD ISODA_ReadRom(WORD wAddress, WORD *wValue)
{
int rr, data, msb,lsb;

if (BASE==0) return(ActiveError);
if (wAddress >= MAX_ROM) return(ChannelError);

rr = send_command_2(READ_ROM, wAddress);
if (rr != NoError) return(TimeOut);

rr = receive_data(&data);                    /* Get response */
if (rr != NoError) return(TimeOut);
if (data != RET_VALUE) return(CheckError);

rr = receive_data(&msb);                     /* Get MSB */
if( rr != NoError) return(TimeOut);

rr = receive_data(&lsb);                /* Get LSB */
if (rr != NoError) return(TimeOut);

*wValue=msb*256+lsb;
return(NoError);
}

/* ---------------------------------------------------------------- */
```

```
WORD ISODA_WriteRom(WORD wAddress, WORD wValue)
{
int rr, data,msb,lsb;
if (BASE==0) return(ActiveError);
if (wAddress >= MAX_ROM) return(CheckError);

lsb=wValue&0xff; wValue=wValue>>8;
msb=wValue&0xff;
rr = send_command_4(WRITE_ROM, wAddress, msb, lsb);
if (rr != NoError) return(TimeOut);

rr = receive_data(&data);
if (rr != NoError) return(TimeOut);
if (data != CMD_OK) return(CheckError);

return(NoError);
}

/* ---------------------------------------------------------------- */
/*
  Output to DA channel
  channel : 0 - 15
  value   : 0 - 16383 (0x3FFF)
*/
WORD ISODA_AnalogOutput(WORD wChannel, WORD wValue)
{
WORD arg1, arg2, command;

if (BASE==0) return(ActiveError);
if (wChannel >= 16) return(CheckError);

command = WRITE_DA_0 + wChannel/4;
arg1 =wChannel%4;
arg1 = arg1 * 64 + wValue/256;
arg2 = wValue%256;
```

```
if (send_command_3(command, arg1&0x0ff, arg2&0x0ff) != NoError)
  {
  receive_data(&arg1);
  return(TimeOut);
  }
```

```
else
  {
  receive_data(&arg1);
  if (arg1 == CMD_OK) return(NoError);
  else return(CheckError);
  }
}

/* ---------------------------------------------------------------- */
/*
  Time period in mSec
  Minimum    0.1 mSec
  Maximum 1000.0 mSec
*/
WORD ISODA_SetTimer(float timeslice)
{
WORD time_period;
char scale;
WORD rr, data;

if (BASE==0) return(ActiveError);

if (timeslice < 0.1)  return(CheckError);
else if (timeslice <= 12)
    {
    time_period = timeslice * 5000.0;
    scale = 0;
    }
else if (timeslice <= 25)
    {
    time_period = timeslice * 2500.0;
    scale = 1;
    }
else if (timeslice <= 50)
    {
    time_period = timeslice * 1250.0;
```

```
scale = 2;
}
```

```
else if (timeslice <= 100)
    {
    time_period = timeslice * 625.0;
    scale = 3;
    }
else return(CheckError);

time_period--;
rr = send_command_4(TIMER_CMD, scale, time_period/256, time_period%256);
if (rr != NoError) return(TimeOut);
rr = receive_data(&data);
if (rr != NoError) return(TimeOut);
if (data != CMD_OK) return(CheckError);
return(NoError);
}

/* ---------------------------------------------------------------------- */

WORD ISODA_ReadTimer(float *timeslice)
{
WORD rr, t1, t2, t3;

if (BASE==0) return(ActiveError);

rr = send_command_2(TIMER_CMD, 0x30);
if (rr != NoError) return(TimeOut);

rr = receive_data(&t1);
if (rr != NoError) return(TimeOut);
if (t1 != RET_VALUE) return(CheckError);
rr = receive_data(&t1);
if (rr != NoError) return(TimeOut);
rr = receive_data(&t2);
if(rr != NoError) return(TimeOut);
rr = receive_data(&t3);
if(rr != NoError) return(TimeOut);
```

```
*timeslice = t2*256.0 + t3 + 1;
```

```
if(t1==0) *timeslice = (*timeslice)/5000;
if(t1==1) *timeslice = (*timeslice)/2500;
if(t1==2) *timeslice = (*timeslice)/1250;
if(t1==3) *timeslice = (*timeslice)/625;
return(NoError);
}

/* ---------------------------------------------------------------- */

WORD ISODA_StartTimer(void)
{
WORD rr, data;

if (BASE==0) return(ActiveError);

rr = send_command_2(TIMER_CMD, 0x10);
if (rr != NoError) return(TimeOut);

rr = receive_data(&data);
if (rr != NoError) return(TimeOut);
if (data != CMD_OK) return(CheckError);
return(NoError);
}

/* ---------------------------------------------------------------- */

WORD ISODA_StopTimer(void)
{
WORD rr, data;

if (BASE==0) return(ActiveError);

rr = send_command_2(TIMER_CMD, 0x20);
if (rr != NoError) return(TimeOut);

rr = receive_data(&data);
```

```
if (rr != NoError) return(TimeOut);
if (data != CMD_OK) return(CheckError);
```

```
return(NoError);
}

/* ---------------------------------------------------------------- */

WORD ISODA_DelayMs(WORD wMs)
{
WORD i,wRetVal,wStatus;

if (BASE==0) return(ActiveError);

inportb(BASE+4);                /* clear interrupt signal */
wRetVal=wait_int_ready();
if (wRetVal != NoError) return(CheckError);

wRetVal=ISODA_SetTimer((float)1.0);       /* 1 ms */
if (wRetVal != NoError) return(TimeOut);
for (i=0; i<wMs; i++)
   {
   wRetVal=ISODA_StartTimer();
   if (wRetVal != NoError) return(TimeOut);

   for (;;)
      {
      wStatus = inportb(STATUS) & 0x08;
      if (wStatus != 0) break;
      }

   wRetVal=ISODA_StopTimer();
   if (wRetVal != NoError) return(TimeOut);

   inportb(BASE+4);                /* clear interrupt signal */
   wRetVal=wait_int_ready();
   if (wRetVal != NoError) return(CheckError);
   }
return(NoError);
```

```
}
```

I. Programming Notes

A. EEPROM Format

- total address : 0 to 63, 64 word, 16-bit per word
- address 0 to 15 : for power-on value
- address 16 to 31 : reserved for safe value
- address 32 to 47 : for voltage calibration data
- address 48 to 63 : for current calibration data

1. power-on value format : address_0 for D/A channel_0, address_1 for D/A channel_1, ….., address_15 for D/A channel_15. Because the D/A converter will output the voltage and current at the same time, therefore this value may be refer to voltage or current output. The ISO-DA16/DA8 equips with 16 channels D/A converter, every D/A converter output the voltage and current at the same time. So the user only can use only one of voltage or current output at the same channel. **The power-on value is refer to "D/A converter" not to "voltage or current".**
2. voltage calibration data format : address_32 for D/A channel_0, address_33 for D/A channel_1, ….., address_47 for D/A channel_15. **The low byte is the wMinV[0..15]** and **the high byte is equal to wMaxV[0..15]-BASE1. BASE1=wMin[0]+16050**
3. current calibration data format : address_48 for D/A channel_0, address_49 for D/A channel_1, ….., address_63 for D/A channel_15. **The low byte is equal to wMinI[0..15]-BASE2, BASE2=(wMaxV[0]-wMinV[0])/2-100** and **the high byte is equal to wMaxI[0..15]-BASE3. BASE3 = wMinI[0] + 7670**

voltage.low=wMinV[0..15]
voltage.high=wMaxV[0..15]-BASE1
current.low=wMinI[0..15]-BASE2
current.high=wMaxI[0..15]-BASE3

A. Calibration

ICP DAS use “ISO-DA16/DA8 calibration kits” to calibrate ISO-DA16/DA8. This kits make software calibration possible. There are five modules in this kits giving as following:

module 1 : I-7017, baud rate=9600, range code=08, address 01(for voltage 0 to 7)

module 2: I-7017, baud rate=9600, range code=08, address 02(for voltage 8 to 15)

module 3: I-7017, baud rate=9600, range code=09, address 03(for current 0 to 7)

module 4: I-7017, baud rate=9600, range code=09, address 02(for current 8 to 15)

module 5: I-7520, RS-232 to RS-485 converter

The user can perform calibration himself. The key point of calibration is to find

wMinV[0..15]
wMaxV[0..15]
wMinI[0..15]
wMaxI[0..15]

If you have voltage meter and current meter, you can perform calibration yourself.

The wMinV[0..15] calibration step are giving as following:

step 1 : use function-e to send out 14-bit data

step 2 : use voltage meter and find the maximum value that voltage < -10.000v

step3 : wMinV[0..15]=this value+1

The wMaxV[0..15] calibration step are giving as following:

step 1 : use function-e to send out 14-bit data

step 2 : use voltage meter and find the minimum value that voltage > 10.000v

step3 : wMaxV[0..15]=this value-1

The wMinI[0..15] calibration step are giving as following:

step 1 : use function-e to send out 14-bit data

step 2 : use current meter and find the maximum value that voltage < 0mA

step3 : wMinI[0..15]=this value+1

The wMaxI[0..15] calibration step are giving as following:

step 1 : use function-e to send out 14-bit data

step 2 : use voltage meter and find the minimum value that voltage > 20.000mA

step3 : wMaxI[0..15]=this value-1