ISO-AD32

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 may be 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

1. Introduction

The AD32?.lib is a collection of data acquisition subroutines for ISO-AD32H, ISO-AD32L. These subroutines are written with C language and perform a variety of data acquisition operations.

The subroutines in AD32?.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 programs are provided.

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

Support 8 cards in one PC maximum

2. Installation

It is recommended to install the ISO-AD32 application software to your hard disk to get the best performance. Before beginning, to make a backup copy of the ISO-AD32 application software. Store the original diskette in a safe place. The DOS software directory in ICPDAS CD of ISO-AD32 includes the following files:

- \ NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISODA*.* : for ISO-DA16/DA8
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32*.* : for ISO-AD32L/H
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_LD*.* : for ISO-LDL/H

The related software is given as following:

- \ NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DEMO
  : for ISO-AD32, TC, demo programs
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\LIB
  : for ISO-AD32, TC, library
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DRIVER
  : for ISO-AD32, library source code
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\MSC\DEMO
  : for ISO-AD32, MSC, demo programs
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\MSC\LIB
  : for ISO-AD32, MSC, library

2.1 Compiler & link using MSC

- The including file is ISO_AD32.H
- There are 5different mode library files: AD32S.lib, ……, AD32H.lib
- Support MSC 6.x compiler
- **SMALL** model compiler & link command: CL /AS program.c AD32S.lib
- **COMPACT** model compiler & link command: CL /AM program.c AD32M.lib
- **MEDIUM** model compiler & link command: CL /AC program.c AD32C.lib
- **LARGE** model compiler & link command: CL /AL program.c AD32L.lib
- **HUGE** model compiler & link command: CL /AH program.c AD32H.lib
- Demo programs with completely source in the companion diskette

2.2 Compiler & link using TC

- The including file is ISO_AD32.H
- There are 5different mode library files: AD32S.lib, ……, AD32H.lib
- Support TC2.x compiler
- Use text editor to create a project file include: program.c AD32?.lib
- Use TC integrated environment to **select the correct compiler model**
- Demo programs with completely source in the companion diskette

3. C Language Library

3.1 ISO_AD32.H

```
#define WORD int

#define NoError          0
#define SysNoOpen        1
#define ActiveError      2
#define TimeOut          3
#define SysOpenError     4
#define CheckError       5
#define BoardError       6
#define SingDiffError    7
#define IrqError         8
#define SampleRateError  9
#define HandshakeError   10


#define UNIPOLAR         0x40/* Unipolar */
#define BIPOLAR          0x00/* Bipolar */

                         /* ISO-AD32H     ISO-AD32L */
#define CONFIG_1         0x00    /* Gain = 1        1     */
#define CONFIG_2         0x10    /*        10       2     */
#define CONFIG_3         0x20    /*        100      4     */
#define CONFIG_4         0x30    /*        1000     8     */
#define CONFIG_5         0x80    /*        0.5      0.5   */
#define CONFIG_6         0x90    /*        5        1     */
#define CONFIG_7         0xA0    /*        50       2     */
#define CONFIG_8         0xB0    /*        500      4     */
```

```
#define SAMPLE_RATE_NO_INT      0x00
#define SAMPLE_RATE_WITH_INT    0x01
#define FAST_NO_INT             0x02
#define FAST_WITH_INT           0x03

extern short    AD32_ShortSub2(short nA, short nB);
extern float    AD32_FloatSub2(float fA, float fB);
extern WORD     AD32_GetVersion(void);
extern WORD     AD32_CheckBoard(WORD wBoard, WORD wBase, WORD wSingDiff,
                WORD wIrq);
extern WORD     AD32_ActiveBoard(WORD wBoard);
extern WORD     AD32_ReadActive(WORD *wActive, WORD *wBase, WORD *wIrq);
extern WORD     AD32_SetChannelConfigNoScan(WORD wChannel, WORD wMode,
                WORD wConfig, WORD wSettle);
extern WORD     AD32_ReadChannelConfigNoScan(WORD *wChannel, WORD *wMode,
                WORD *wConfig);
extern WORD     AD32_ReadSingDiff(WORD *wSingDiff);
extern WORD     AD32_ReadAdNoScan(WORD *wData);
extern WORD     AD32_StartAdsNoScan(WORD wMode, WORD wCount);
extern WORD     AD32_ReadSystemStatus(WORD *wStatus);
extern WORD     AD32_ReadFIFO(WORD *wData);
extern WORD     AD32_ClearFIFO(void);
extern WORD     AD32_SetSampleRate(float fSampleRate);
extern WORD     AD32_ReadSampleRate(float *fSampleRate);
extern WORD     AD32_ReadChannelScanStatus(WORD wScanNum, WORD *wChannel,
                WORD *wMode, WORD *wConfig);
extern WORD     AD32_SetChannelScanStatus(WORD wScanNum, WORD wChannel,
                WORD wMode, WORD wConfig);
extern WORD     AD32_StartChannelScanAds(WORD wMode, WORD wScanCount,
                WORD wAdCount);
```

3.2 Function Description

3.2.1 AD32_ShortSub2

- Description :
  Compute C=A-B in short format, short=16 bits sign integer.
  This function is provided for testing purpose.
- Syntax :
  short AD32_ShortSub2(short nA, short nB);
- Parameter :
  nA : [Input] short integer
  nB : [Input] short integer
- Return :
  return=nA-nB → short integer

3.2.2 AD32_FloatSub2

- Description :
  Compute A-B in float format, float=32 bits floating pointer number.
  This function is provided for testing purpose.
- Syntax :
  float AD32_FloatSub2(float fA, float fB);
- Parameter :
  fA : [Input] floating point value
  fB : [Input] floating point value
- Return :
  return=fA-fB → floating point value

3.2.3 AD32_GetVersion

- Description :
  Read the software version.
- Syntax :
  WORD AD32_GetVersion(void) ;
- Parameter :
  None
- Return :
  return=0x100 → Version 1.00

3.2.4 AD32_CheckBoard

- Description :
  This function is used to check the hardware board. If all checks are OK, this function actives this board. ISO-AD32 DOS driver supports 8 cards at most in one PC system. **The program must call this function to check each board first, then call *AD32_ActiveBoard( )* to select which board is active**. **All the other functions are referred to the active board only.** If only one board is used, the unique board will be active after this function is called.

- Syntax :
  WORD AD32_CheckBoard(WORD wBoard, WORD wBase, WORD wSingDiff,
  WORD wIrq);
- Parameter :
  wBoard : [Input] Board number, 0 to 7,
  support at most 8 cards in one PC system.
  The user must specify a number of the device.
  wBase : [Input] Board base-address, for example: 0x220.
  wSingDiff : [Input] 0 = single-ended type,
  Others = differential type
  wIrq : [Input] Board IRQ number, Reserved.
  The user must fill it with 0.
  (The IRQ does not supported in this software)
- Return :
  AD32_DriverNoOpen : Device-Driver no open
  AD32_BoardError : Validate board number from 0 to 7
  AD32_IrqError : Invalidate IRQ number
  AD32_CheckError : Check board error (maybe base address error)
  AD32_TimeOut : Check board timeout
  (maybe base address or hardware error)
  AD32_SiggDiffError : JP1 setting and checking inconsistent
  AD32_NoError : OK

3.2.5 AD32_ActiveBoard

- Description :
  Active the specific board (defined by wBoard). AD32_CheckBoard( ) must be called once before active this board.

- Syntax :
  WORD AD32_ActiveBoard(WORD wBoard);

- Parameter :
  wBoard : [Input] board number, 0 to 7,

- Return :
  BoardError : Validate board number from 0 to 7
  ActiveError : This board must call AD32_CheckBoard first
  NoError : OK

3.2.6 AD32_ReadActive

- Description :
  Read the information of the current active board.

- Syntax :
  WORD AD32_ActiveBoard(WORD *wBoard, WORD *wBase, WORD *wIrq);

- Parameter :
  wBoard, wBase, wIrq
  : [Output] address of wBoard, wBase and wIrq,

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  AD32_NoError : OK

3.2.7 AD32_SetChannelConfigNoScan

- Description :
  Set the configuration data. (No Channel-Scan mode.)

- Syntax :
  WORD AD32_SetChannelConfigNoScan(WORD wChannel,
  WORD wMode, WORD wConfig, WORD wSettle);

- Parameter :
  wChannel : [Input] Single-ended=0 to 31, Differential=0 to 15
  wMode : [Input] **UNIPOLAR or BIPOLAR (Refer to Sec.3.1 AD32.H)**
  wConfig : [Input] **AD32_CONFIG_1 to AD32_CONFIG_8**
  (Refer to Sec.3.1 AD32.H)
  wSettle : [Input] Settling time delay (in ms)
  (Please refer to Sec3.1.1 of Hardware Manual).

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  HandshakeError : Hardware handshake error
  NoError : OK

3.2.8 AD32_ReadChannelConfigNoScan

- Description :
  Read the active channel configuration data of current active board.
  (no channel scan mode)

- Syntax :
  WORD AD32_ReadChannelConfigNoScan(WORD *wChannel,
  WORD *wMode, WORD *wConfig);

- Parameter :
  wChannel, wMode, wConfig
  : [Output] The address of wChannel, wMOde and wConfig.
  (Refer to Hardware's Manual and AD32.h)

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  NoError : OK

3.2.9 AD32_ReadSingDiff

- Description :
  Read the setting of JP1

- Syntax :
  WORD AD32_ReadSignDiff(WORD *wSingDiff);

- Parameter :
  wSingDiff : [Output] The setting of wSingDiff,
  (0=single-ended type, 1=differential type)

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  NoError : OK

3.2.10 AD32_ReadAdNoScan

- Description :
  Perform one A/D operation and read this value. The active channel is defined in AD32_SetChannelConfigNoScan. (No channel scan mode)

- Syntax :
  WORD AD32_ReadAdNoScan(WORD *wData);

- Parameter :
  wData : [Output] 12 bits A/D data
  (min=0, max=4095, for both unipolar/bipolar)

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  NoError : OK

3.2.11 AD32_StartAdsNoScan

- Description :
  Perform multiple A/D operations. (No channel scan mode)

- Syntax :
  WORD AD32_StartAdsNoScan(WORD wMode, WORD wCount);

- Parameter :
  wMode : [Input] **SAMPLE_RATE_NO_INT to FAST_WITH_INT**, Refer to Sec3.18
  wCount : [Input] Number of A/D conversions to perform
  1 ➔ 1*256 conversions
  n ➔ n*256 conversions

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  NoError : OK

3.2.12 AD32_ReadSystemStatus

- Description :
  Read the system status

- Syntax :
  WORD AD32_ReadSystemStatus(WORD *wStatus);

- Parameter :
  wStatus : [Output] Return the value of System-Status.

Bit	Value	Status Description
Bit0	1	Command_FIFO not full
Bit1	1	Data_FIFO not empty
Bit2	1	Command_FIFO is empty
Bit3	1	Data_FIFO is half_full
Bit4	1	Data_FIFO is full
Bit5	1	Embedded Controller error

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  NoError : OK

3.2.13 AD32_ReadFIFO

- Description :
  Read the 8 bits FIFO data. (Call **AD32_ReadSystemStatus** first, if the FIFO data is not ready, this function will read the invalidate data)

- Syntax :
  WORD AD32_ReadFIFO(WORD *wData);

- Parameter :
  wData : [Output] address of wData
  which will stores the 8 bits data read from FIFO
  (12 bits data will need to read twice,
  LOW_BYTE first then HIGH_BYTE)

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  NoError : OK

3.2.14 AD32_ClearFIFO

- Description :
  Clear the on-board 1K FIFO. The AD32_CheckBoard(…) will clear FIFO for initial state. The other function will not clear FIFO.

- Syntax :
  WORD AD32_ClearFIFO(void);

- Parameter :
  Void

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  NoError : OK

3.2.15 AD32_SetSampleRate

- Description :
  Set the AD sampling rate.

- Syntax :
  WORD AD32_SetSampleRate(float fSampleRate);

- Parameter :
  fSampleRate : [Input] samples per second.
  1000.0 → 1K samples/second
  This value is validated between 100 and 100,000.

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  SampleRateError :
  NoError : OK

3.2.16 AD32_ReadSampleRate

- Description :
  Read the AD sampling rate.

- Syntax :
  WORD AD32_ReadSampleRate(float *fSampleRate);

- Parameter :
  fSampleRate : [Output] address of fSampleRate
  which will stores the value of Sampling-Rate.
  (1000.0 → 1K samples/second)

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  NoError : OK

3.2.17 AD32_ReadChannelScanStatus

- Description :
  Read the status of channel scan.

- Syntax :
  WORD AD32_ReadChannelScanStatus(WORD wScanNum,
  WORD *wChannel, WORD *wMode, WORD *wConfig);

- Parameter :
  wScanNum : [Input] the scan sequence number
  wChannel : [Output] Channel-Number corresponding to wScanNum.
  wMode : [Output] Unipolar /Bipolar mode corresponding to wScanNum
  Unipolar: 0x40 Bipolar: 0x0
  wConfig : [Output] the configure-code corresponding to wScanNum
  (Refer to Sec3.1)
- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  NoError : OK

3.2.18 AD32_SetChannelScanStatus

- Description :
  Set the status of channel scan.

- Syntax :
  WORD AD32_SetChannelScanStatus(WORD wScanNum,
  WORD wChannel, WORD wMode, WORD wConfig);

- Parameter :
  wScanNum : [Input] The scan sequence number
  wChannel : [Input] The channel number corresponding to wScanNum.
  wMode : [Input] The Uni/Bipolar mode corresponding to wScanNum
  Unipolar: 0x40 Bipolar: 0x0
  wConfig : [Input] The config code corresponding to wScanNum
  (Refer to Sec.3.1)

- Return :
  ActiveError : This board must call AD32_CheckBoard first
  TimeOut : Hardware time out
  NoError : OK

3.2.19 AD32_StartChannelScanAds

- Description :
  Start the channel scan operation.
  Note: wAdCount Max value=256 (1 byte)

- Syntax :
  WORD AD32_StartChannelScanAds(WORD wMode,
  WORD wScanCount, WORD wAdCount);

- Parameter :

wMode	: [Input]	AD32_SAMPLE_RATE_NO_INT to AD32_FAST_WITH_INT (Refer to Sec3.1 “AD32.h”)
wScanCount	: [Input]	Number of channels to be scanned. (Total will perform wScanCount * wAdCount data)
wAdCount	: [Input]	Number of A/D conversions to be performed. (Total will perform wScanCount * wAdCount data)

- Return :

ActiveError	: This board must call AD32_CheckBoard first
TimeOut	: Hardware time out
NoError	: OK

Note :

(1). **Sample_rate_speed** : The AD sampling rate is defined by AD32_SetSampleRate(…). The maximum sampling rate is about 50K

(2). **Interrupt** : The embedded controller will interrupt PC when this function is finished.

(3). **No-interrupt** : No interrupt signal when this function is finished.

(4). **Max-speed** : The A/D sampling rate is defined in maximum speed, about 200Ks/s.

It is recommended to use **SAMPLE_RATE_NO_INT** or **FAST_NO_INT**

4. Demo Program

The completely source of demo program is given in DOS software.

- \ NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DEMO
  : for ISO-AD32, TC, demo programs
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\LIB
  : for ISO-AD32, TC, library
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DRIVER
  : for ISO-AD32, library source code
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\MSC\DEMO
  : for ISO-AD32, MSC, demo programs
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\MSC\LIB
  : for ISO-AD32, MSC, library

4.1 Demo1

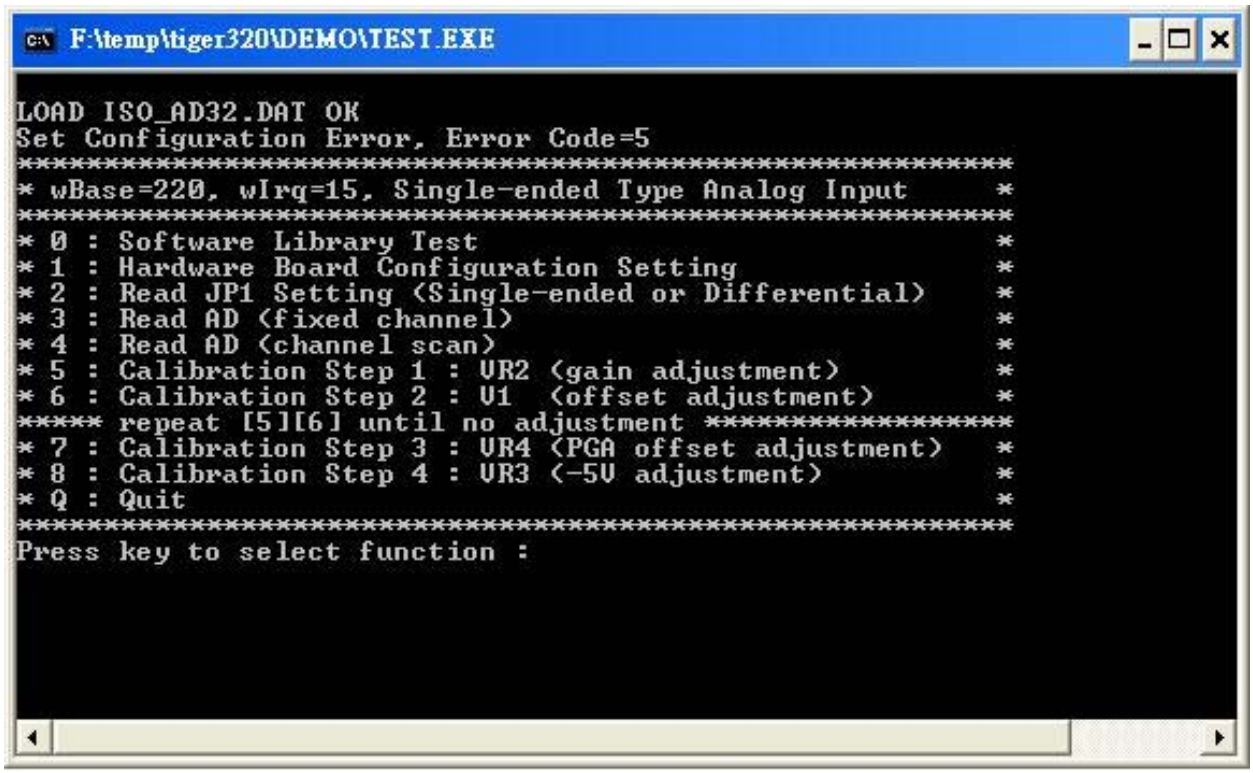
CD:\\NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DEMO

TEST.C	→ main program
TEST.PRJ	→ TC project file
TEST.EXE	→ DOS execution file
AD32L.LIB	→ TC large mode library
ISO_AD32.H	→ header file
ISO_AD32.DAT	→ data file (stored the hardware configuration data)

CD:\\ NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\MSC\DEMO

TEST.C	→ main program
C.BAT	→ MSC compiler and link batch file
TEST.EXE	→ DOS execution file
AD32L.LIB	→ TC large mode library
ISO_AD32.H	→ header file
ISO_AD32.DAT	→ data file (stored the hardware configuration data)

4.1.1 Function of demo program



Function 0 : test software library and read software version

Function 1 : if the hardware board not in the default setting, use this function to change those setting. All the setting will be stored in “ISO_AD32.DAT”. Also the TEST.EXE will automatic load the setting from the “ISO_AD_32.DAT”.

Function 2 : Read the setting of JP1. If the JP1 is in “single-ended setting” and hardware wire connection is “differential type”, the AD results will be incorrect. Therefore user must make sure that the ”hardware wire connection” and “JP1 setting” is consistent.

Function 3 : fixed-channel AD conversion function demo

Function 4 : channel-scan AD conversion function demo

Function 5,6,7,8 : calibration steps

4.1.2 Sources of demo program

```
#include    <stdio.h>
#include    <conio.h>
#include    <stdlib.h>
#include    <dos.h>

#include    "ISO_AD32.H"        // include this header file
#define WORD int

WORD wBase=0x220,wActive=0,wSingDiff=1,wIrq=15,wDelay1,wBuf[512];

/* ---------------------------------------------------------------- */
main()
{
char c;

load_configuration();
for(;;)
{
printf("\n*************************************************************");
show_configuration();
printf("\n*************************************************************");
printf("\n* 0 : Software Library Test                                 *");
printf("\n* 1 : Hardware Board Configuration Setting                  *");
printf("\n* 2 : Read JP1 Setting (Single-ended or Differential)       *");
printf("\n* 3 : Read AD (fixed channel)                               *");
printf("\n* 4 : Read AD (channel scan)                                *");
printf("\n* 5 : Calibration Step 1 : VR2 (gain adjustment)            *");
printf("\n* 6 : Calibration Step 2 : V1    (offset adjustment)        *");
printf("\n***** repeat [5][6] until no adjustment ***********************");
printf("\n* 7 : Calibration Step 3 : VR4 (PGA offset adjustment)      *");
printf("\n* 8 : Calibration Step 4 : VR3 (-5V adjustment)             *");
printf("\n* Q : Quit                                                  *");
printf("\n*************************************************************");
printf("\nPress key to select function : ");
```

Load the hardware configuration first. Perform AD32_CheckBoard(…) to detect hardware before any functions are called.

```
c=getch();
switch     (c)
       {
       case '0' : fun_0(); break;
       case '1' : fun_1(); break;
       case '2' : fun_2(); break;
       case '3' : fun_3(); break;
       case '4' : fun_4(); break;
       case '5' : fun_5(); break;
       case '6' : fun_6(); break;
       case '7' : fun_7(); break;
       case '8' : fun_8(); break;
       case 'q' :
       case 'Q' : goto ret_label;
       }
}
ret_label:
return;
}

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

set_pos(int x, int y)
{
union REGS regs;
regs.h.ah=2;
regs.h.dh=y;
regs.h.dl=x;
regs.h.bh=0;
int86(0x10,&regs,&regs);
}

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

clear_screen(void)
{
union REGS regs;
regs.h.ah=15;
```

```
int86(0x10,&regs,&regs);
regs.h.ah=0;
int86(0x10,&regs,&regs);

}

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

load_configuration()
{
FILE *stream;
int wRetVal;

wActive=0;
stream=fopen("ISO_AD32.DAT","r");
if (stream)
    {
    fscanf(stream,"%x",&wBase);
    fscanf(stream,"%d",&wIrq);
    fscanf(stream,"%d",&wSingDiff);
                        /* for fun_5 */
                        /* for fun_5 */
    fscanf(stream,"%d",&wDelay1);     /* for fun_5 */
                        /* for fun_5 */
                        /* for fun_5 */
    printf("\nLOAD ISO_AD32.DAT OK");

    wRetVal=AD32_CheckBoard(0,wBase,wSingDiff,wIrq);
    if (wRetVal==NoError)
        {
        wActive=1;
        printf("\nSet Configuration OK");
        }
    else printf("\nSet Configuration Error, Error Code=%d",wRetVal);

    fclose(stream);
    }
else printf("\nOPEN ISO_AD32.DAT ERROR");
```

Load the hardware configuration file "ISO_AD32.DAT"

Perform AD32_CheckBoard(…) to detect the hardware board.

```
}

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

show_configuration()
{
printf("\n");
printf("* wBase=%x",wBase);
printf(", wIrq=%d",wIrq);
if (wSingDiff == 0) printf(", Single-ended");
else printf(", Differential");
printf(" Type Analog Input          *");
}

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

save_configuration()
{
FILE *stream;

stream=fopen("ISO_AD32.DAT","w");
if (stream)
     {
     fprintf(stream,"%x\n",wBase);
     fprintf(stream,"%d\n",wIrq);
     fprintf(stream,"%d\n",wSingDiff);
     fprintf(stream,"%d",wDelay1);
     printf("\nSAVE ISO_AD32.DAT OK");
     fclose(stream);
     }
else printf("\nOPEN ISO_AD32.DAT ERROR");
}

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

fun_0()
{
int i;
```

```
float f;

i=AD32_ShortSub2(1,2);
printf("\nLibrary Test : AD32_ShortSub2(1,2)=%d",i);
if (i==-1) printf(" --> OK"); else printf(" --> Error");
f=AD32_FloatSub2(1.0,2.0);
printf("\nLibrary Test : AD32_FloatSub2(1.0,2.0)=%f",f);
if (f==-1.0) printf(" --> OK"); else printf(" --> Error");
i=AD32_GetVersion();
printf("\nSoftware Version=%x",i);
}

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

fun_1()
{
int i,j,k,l;

printf("\nwBASE      (ex. 220) = "); scanf("%x",&i);
printf("wIrq           (ex. 15) = "); scanf("%d",&j);
printf("wSingDiff (0=single-ended) = "); scanf("%d",&k);
l=AD32_CheckBoard(0,i,k,j);
if (l==0)
    {
    wBase=i; wIrq=j; wSingDiff=k;
    wActive=1;
    printf("\nSet Configuration OK");
    save_configuration();
    return;
    }
printf("\nSet Configuration Error, Error Code=%d",l);
wActive=0;
}

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

AD32_ShortSub2(…) & AD32_FloatSub2(…) test

Read software version number

Change hardware configuration

```
fun_2()
{
int i;

if (wActive==0)
    {
    i=AD32_CheckBoard(0,wBase,wSingDiff,wIrq);
    if (i!=0)
        {
        printf("\nHardware Board Configuration Error");
        return;
        }
    wActive=1;
    }

AD32_ReadSingDiff(&i);

printf("\nAD32_ReadSingDiff=%d",i);
if (i==0) printf(" --> Single-ended"); printf(" --> Differential");
}

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

fun_3()
{
int wBoard,wBase,wIrq,wRetVal,i,j,k,flag;
int wChannel,wMode,wConfig,wSettle,wData;
int wLow,wHigh,wMin,wMax;
float f;

wRetVal=AD32_ReadActive(&wBoard,&wBase,&wIrq);
if (wRetVal==NoError) printf("\nboard=%d, base=%x, irq=%d",wBoard,wBase,wIrq);
else
        {
        printf("\nAD32_ReadActive(...) Error");
        return;
        }
```

Read JP1 setting

Show current active board and its configuration.

```
wChannel=0; wMode=BIPOLAR; wConfig=CONFIG_1; wSettle=23;
wRetVal=AD32_SetChannelConfigNoScan(wChannel,wMode,wConfig,wSettle);
if (wRetVal!=NoError)
    {
    printf("\nAD32_SetChannelConfigNoScan(...) Error");
    return;
    }
```

Select the fixed-channel mode, channel_0 bipolar and +/- 5V range

```
wRetVal=AD32_ReadChannelConfigNoScan(&wChannel,&wMode,&wConfig);
if (wRetVal!=NoError)
    {
    printf("\nAD32_ReadChannelConfigNoScan(...) Error");
    return;
    }
printf("\nchannel=%d, mode=%d, config=%d",wChannel,wMode,wConfig);
```

Setting read back testing

```
wRetVal=AD32_ReadAdNoScan(&wData);
if (wRetVal!=NoError)
    {
    printf("\nAD32_ReadAdNoScan(...) Error");
    return;
    }
f=(float)(wData-2048)/2048.0*5.0;
printf("\ndata=%x, f=%f",wData,f);
```

Read one AD conversion data

Convert from 12 bits binary data to floating point value

```
wRetVal=AD32_StartAdsNoScan(FAST_NO_INT,1);
if (wRetVal!=NoError)
    {
    printf("\nAD32_StartAdsNoScan(...) Error");
    return;
    }
```

Perform 256*1 AD conversion in max. speed

```
j=0; flag=0;
for (k=0; k<10; k++)
for (i<0; i<32766; i++)
     {
     AD32_ReadSystemStatus(&wData);
     if ((wData&0x02)!=0)
```

This code is used to demo “only 256*1 data are putting in FIFO”. In real world application, the program can stop if 256 data are read (no need to loop for testing)

```
        {
        AD32_ReadFIFO(&wData);
        if (flag==0) {wLow=wData; flag=1;}
        else {wHigh=wData; flag=0; wBuf[j]=wLow+(wHigh<<8); j++;}
        }
    }

printf("\nTotal=%d bytes\n",j);
for (i=0; i<10; i++) printf("[%3x]",wBuf[i]);
wMin=0x7fff; wMax=0;
for (i=0; i<j; i++)
    {
    if (wBuf[i]>wMax) wMax=wBuf[i];
    if (wBuf[i]<wMin) wMin=wBuf[i];
    }
printf("\nwMin=%x, wMax=%x",wMin,wMax);

}

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

fun_4()
{
int wBoard,wBase,wIrq,wRetVal,i,j,k,flag;
int wChannel,wMode,wConfig,wSettle,wData;
int wLow,wHigh,wMin,wMax;
float f,fVal;

wRetVal=AD32_ReadActive(&wBoard,&wBase,&wIrq);
if (wRetVal==NoError) printf("\nboard=%d, base=%x, irq=%d",wBoard,wBase,wIrq);
else
    {
    printf("\nAD32_ReadActive(...) Error");
    return;
    }

wRetVal=AD32_SetSampleRate((float)1234.0);
if (wRetVal==NoError) printf("\nSetSampleRate 1234 s/s OK");
```

Read 8 bits FIFO. LOW_BYTE first then HIGH_BYTE

Find the min. and max. value of these 256 values

Set the expected sampling rate

```
else printf("\nSetSampleRate Error");

wRetVal=AD32_ReadSampleRate(&fVal);
if (wRetVal==NoError) printf("\nReadSampleRate=%.0f s/s",fVal);
else printf("\nReadSampleRate Error");

wRetVal=AD32_SetSampleRate((float)5678.0);
if (wRetVal==NoError) printf("\nSetSampleRate 5678 s/s OK");
else printf("\nSetSampleRate Error");

wRetVal=AD32_ReadSampleRate(&fVal);
if (wRetVal==NoError) printf("\nReadSampleRate=%.0f s/s",fVal);
else printf("\nReadSampleRate Error");

/* Scan sequence 3->2->1->0->3.... */
wRetVal=AD32_SetChannelScanStatus(0,3,0,CONFIG_1);
wRetVal += AD32_SetChannelScanStatus(1,2,0,CONFIG_2);
wRetVal += AD32_SetChannelScanStatus(2,1,0,CONFIG_3);
wRetVal += AD32_SetChannelScanStatus(3,0,0,CONFIG_4);
if (wRetVal==NoError) printf("\nSet Channel Scan TEST1 OK");
else printf("\nSet Channel Scan TEST1 Error");

/* Scan sequence 0->1->2->3->0.... */
wRetVal=AD32_SetChannelScanStatus(0,0,0,CONFIG_1);
wRetVal += AD32_SetChannelScanStatus(1,1,0,CONFIG_2);
wRetVal += AD32_SetChannelScanStatus(2,2,0,CONFIG_3);
wRetVal += AD32_SetChannelScanStatus(3,3,0,CONFIG_4);
if (wRetVal==NoError) printf("\nSet Channel Scan TEST2 OK");
else printf("\nSet Channel Scan TEST2 Error");

/* Conversion=FAST, scan=4 channel, total Ads=4*64=256 WORD=512 BYTE */
/* Scan sequence : 0->1->2->3->0...                  */
wRetVal=AD32_StartChannelScanAds(FAST_NO_INT,4,64);
if (wRetVal==NoError) printf("\nStart Channel Scan OK");
else printf("\nStart Channel Scan Error");
```

Read back the real sampling rate

Channel-scan setting testing

Channel-scan setting testing

Start channel-scan operation. Total will perform 4*64 = 256 data

```
j=0;
for (k=0; k<10; k++)
for (i=0; i<32760; i++)
      {
      AD32_ReadSystemStatus(&wData);
      if ((wData&0x02)!=0)
          {
          AD32_ReadFIFO(&wData);
          if (i<5) printf("[wdata=%x]",wData);
          j++;
          }
      }
printf("\nTotal data read from FIFO = %d bytes",j);
}

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

fun_5()
{
int wRetVal,i,j,k;
int wChannel,wMode,wConfig,wSettle,wData;
float f;

clear_screen();
set_pos(0,0);

wChannel=0; wMode=BIPOLAR; wConfig=CONFIG_1; wSettle=23;
wRetVal=AD32_SetChannelConfigNoScan(wChannel,wMode,wConfig,wSettle);
if (wRetVal!=NoError)
    {
    printf("\nAD32_SetChannelConfigNoScan(...) Error");
    return;
    }

printf("\nCalibration Step 1 = VR2 (gain adjustment)");
printf("\nApply stable 4.9988V to A/D channel_0");
printf("\nAdjust VR2 until DATA between 4094 and 4095");
```

This code is used to demo : “only 4*64 data are putting in FIFO”. In real world application, the program can stop if 256 data are read. (no need to loop for testing)

Select channel_0, +/- 5V range

```
for (;;)
      {
      for (i=0; i<wDelay1; i++)
      {
      wRetVal=AD32_ReadAdNoScan(&wData);
      if (wRetVal!=NoError)
           {
           printf("\nAD32_ReadAdNoScan(...) Error");
           return;
           }
      if (kbhit()!=0) {getch(); return;}
      }
      set_pos(0,4); printf("DATA=%d         ",wData);
      }
}

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

fun_6()
{
int wRetVal,i,j,k;
int wChannel,wMode,wConfig,wSettle,wData;
float f;

clear_screen();
set_pos(0,0);

wChannel=0; wMode=BIPOLAR; wConfig=CONFIG_1; wSettle=23;
wRetVal=AD32_SetChannelConfigNoScan(wChannel,wMode,wConfig,wSettle);
if (wRetVal!=NoError)
     {
     printf("\nAD32_SetChannelConfigNoScan(...) Error");
     return;
     }

printf("\nCalibration Step 2 = V1    (offset adjustment)");
printf("\nApply stable 0V to A/D channel_0");
printf("\nAdjust V1    until DATA between 2048 and 2049");
```

Read AD about one time per second. "wDalay1" is defined in the last value of "ISO_AD32.DAT"

Show the AD 12 bits for calibration

Refer to fun_5

```
for (;;)
      {
      for (i=0; i<wDelay1; i++)
      {
      wRetVal=AD32_ReadAdNoScan(&wData);
      if (wRetVal!=NoError)
           {
           printf("\nAD32_ReadAdNoScan(...) Error");
           return;
           }
      if (kbhit()!=0) {getch(); return;}
      }
      set_pos(0,4); printf("DATA=%d          ",wData);
      }
}

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

fun_7()
{
int wRetVal,i,j,k;
int wChannel,wMode,wConfig,wSettle,wData;
float f;

clear_screen();
set_pos(0,0);

wChannel=0; wMode=BIPOLAR; wConfig=CONFIG_1; wSettle=23;
wRetVal=AD32_SetChannelConfigNoScan(wChannel,wMode,wConfig,wSettle);
if (wRetVal!=NoError)
     {
     printf("\nAD32_SetChannelConfigNoScan(...) Error");
     return;
     }

printf("\nCalibration Step 3 = VR4 (PGA offset adjustment)");
printf("\nApply stable 0V to A/D channel_0");
```

Refer to fun_5

```
printf("\nAdjust VR4 until DATA between 2048 and 2049");

for (;;)
      {
      for (i=0; i<wDelay1; i++)
      {
      wRetVal=AD32_ReadAdNoScan(&wData);
      if (wRetVal!=NoError)
           {
           printf("\nAD32_ReadAdNoScan(...) Error");
           return;
           }
      if (kbhit()!=0) {getch(); return;}
      }
      set_pos(0,4); printf("DATA=%d         ",wData);
      }
}

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

fun_8()
{
int wRetVal,i,j,k;
int wChannel,wMode,wConfig,wSettle,wData;
float f;

clear_screen();
set_pos(0,0);

wChannel=0; wMode=BIPOLAR; wConfig=CONFIG_1; wSettle=23;
wRetVal=AD32_SetChannelConfigNoScan(wChannel,wMode,wConfig,wSettle);
if (wRetVal!=NoError)
     {
     printf("\nAD32_SetChannelConfigNoScan(...) Error");
     return;
     }

printf("\nCalibration Step 4 = VR3 (-5V adjustment)");
```

Refer to fun_5

```
printf("\nApply stable 0V to A/D channel_0");
printf("\nAdjust VR3 until DATA between 0 and 1");

for (;;)
      {
      for (i=0; i<wDelay1; i++)
      {
      wRetVal=AD32_ReadAdNoScan(&wData);
      if (wRetVal!=NoError)
           {
           printf("\nAD32_ReadAdNoScan(...) Error");
           return;
           }
      if (kbhit()!=0) {getch(); return;}
      }
      set_pos(0,4); printf("DATA=%d          ",wData);
      }
}
```

4.1.3 ISO_AD32.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 "ISO_AD32.DAT" is given as following:

```
220
15
0
2000
```

220 : hardware base address, default = 220, refer to "hardware manual" Sec 2.6 for setting.
15 : select IRQ channel 15, default = 15, if no IRQ, set this value 15.
0 : 0 = single-ended, others = differential.
2000: wDelay1. This value is used to delay the AD conversion of Calibration. It is suitable to perform one AD conversion per second in the calibration steps. 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.

4.1.4 Calibration

The calibration can be divided into 4 steps as following:

Step1 : AD gain adjustment → function 5 of TEST.EXE
Apply 9.988V to AD channel_0 and adjust VR2 until DATA between 4094 and 4095
Step2 : AD offset adjustment → function 6 of TEST.EXE
Apply 0V to AD channel_0 and adjust VR1 until DATA between 2048 and 2049

Repeat step1 and step2 until no adjustment is required

Step3 : PGA offset adjustment → function 7 of TEST.EXE
Apply 0V to AD channel_0 and adjust VR4 until DATA between 2048 and 2049
Step4 : -5V adjustment → function 8 of TEST.EXE
Apply 0V to AD channel_0 and adjust VR3 until DATA between 2048 and 2049

4.2 Demo2

The completely source listing is given in
CD : \\NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\WAVEDEMO*.*

TEST.C	→ main program
TEST.PRJ	→ TC project file
TEST.EXE	→ DOS execution file
AD32L.Lib	→ TC large mode library
ISO_AD32.H	→ header file
ISO_AD32.DAT	→ data file (stored the hardware configuration data)

The demo is designed to show the analog input wave form. The source code is very similar to Sec4.1 and given in the companion diskette. The function 3 is used to demo the 200Ks/s fixed-channel AD conversion. The function 4 is used to demo the 45Ks/s channel-scan AD conversion.

In the function 3 demo, the user can apply a 20K sine wave to AD channel_0 and the input wave form will be shown on the screen. Because the maximum sampling rate is about 200Ks/s, the analog input will be sampling about 10 points in one cycle. Therefore the analog input ware form can be reconstructed from the 10 points. If the frequency of analog input is over 20K, the sampling points will be less than 10, then the wave form shown on the screen will be distorted.

```
*********************************************************   *
* wBase = 220, wIrq =15, Single-ended Type Analog Input      *
*********************************************************   *
* 1 : Hardware Board Configuration Setting                   *
* 2 : Read JR1 Setting (Single-ended or Differential)        *
* 3 : Read AD (fixed channel) (channel 0) Waveform               *
* 4 : Read AD (channel scan) (0->1->2->3->0->1…)             *
* Q : Quit                                                   *
*********************************************************   *
Press key to select function :
```

TEST.EXE start menu.

5. Driver Source Program

The driver source is giving in the companion diskette as follows:

- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DRIVER\ISO_AD32.C driver source program
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DRIVER\ISO_AD32.H header file
- \NAPDOS\ISA\ISOAD32\DOS\ISOAD32\ISO_AD32\TC\DRIVER\ISO_AD32.PRJ project file

5.1 IO Base Register

	INP	OUTP
BASE	Read System Status Byte	Write to COMMAND_FIFO
BASE+4	Read DATA_FIFO	Clear DATA_FIFO & COMMAND_FIFO

Status Byte	Description
Bit_0	1 = COMMAND_FIFO not full
Bit_1	1 = DATA_FIFO not empty
Bit_2	1 = COMMAND_FIFO is empty
Bit_3	1 = DATA_FIFO is half full
Bit_4	1 = DATA_FIFO is full
Bit_5	1 = Embedded controller error

5.2 Command Sets

```
1.  #define CMD_READ_PATTERN          0x0A4   /* 1010-0100 */
2.  #define CMD_READ_CH_SCAN_DATA     0x0A5   /* 1010-0101 */
3.  #define CMD_READ_MUX_CONFIG       0x0A6   /* 1010-0110 */
4.  #define CMD_READ_SAMPLE_RATE      0x0A7   /* 1010-0110 */
5.  #define CMD_READ_CURRENT_AD       0x0A8   /* 1010-1000 */
6.  #define CMD_WRITE_MUX_CONFIG      0x0A9   /* 1010-1001 */
7.  #define CMD_WRITE_SAMPLE_RATE     0x0AA   /* 1010-1010 */
8.  #define CMD_AD_SAMPLE             0x0AB   /* 1010-1011 */
9.  #define CMD_WRITE_CH_SCAN_DATA    0x0AC   /* 1010-1100 */
10. #define CMD_CHANNEL_SCAN_START    0x0AD   /* 1010-1101 */
11. #define CMD_WRITE_TEST            0x0AE   /* 1010-1110 */
12. #define CMD_READ_TEST             0x0AF   /* 1010-1111 */
```

CMD_READ_PATTERN : Reserved
CMD_READ_CH_SCAN_DATA : read back channel scan data
CMD_READ_MUX_CONFIG : read back multiplex and configuration data
CMD_READ_SAMPLE_RATE : read back AD pacer sampling rate
CMD_READ_CURRENT_AD : read the current active channel AD data

CMD_WRITE_MUX_CONFIG : set multiplex and configuration data
CMD_WRITE_SAMPLE_RATE : set AD pacer sampling rate
CMD_AD_SAMPLE : set AD conversion by pacer trigger mode
CMD_WRITE_CH_SCAN_DATA : set channel scan data
CMD_CHANNEL_SCAN_START : start channel scan ADs
CMD_WRITE_TEST : Reserved
CMD_READ_TEST : Reserved

5.3 Driver Source

```
#include   <stdio.h>
#include   <conio.h>
#include   <stdlib.h>

#include   "ISO_AD32.H"
#define WORD int

#define BASE_CLOCK  20000.0   /* 20.0000 MHz */

#define READ_DATA   inportb(BASE+4)&0xff

#define CMD_READ_PATTERN          0x0A4     /* 1010-0100 */
#define CMD_READ_CH_SCAN_DATA     0x0A5     /* 1010-0101 */
#define CMD_READ_MUX_CONFIG       0x0A6     /* 1010-0110 */
#define CMD_READ_SAMPLE_RATE      0x0A7     /* 1010-0110 */
#define CMD_READ_CURRENT_AD       0x0A8     /* 1010-1000 */
#define CMD_WRITE_MUX_CONFIG      0x0A9     /* 1010-1001 */
#define CMD_WRITE_SAMPLE_RATE     0x0AA     /* 1010-1010 */
#define CMD_AD_SAMPLE             0x0AB     /* 1010-1011 */
#define CMD_WRITE_CH_SCAN_DATA    0x0AC     /* 1010-1100 */
#define CMD_CHANNEL_SCAN_START    0x0AD     /* 1010-1101 */
#define CMD_WRITE_TEST            0x0AE     /* 1010-1110 */
#define CMD_READ_TEST             0x0AF     /* 1010-1111 */

WORD wait_write_fifo(void);
WORD wait_read_fifo(void);
WORD send_cmd_2_arg(WORD out_1, WORD out_2, WORD out_3);
WORD send_cmd_3_arg(WORD out_1, WORD out_2, WORD out_3, WORD out_4);
WORD send_cmd_4_arg(WORD out_1, WORD out_2, WORD out_3, WORD out_4, WORD
out_5);

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

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

```
short AD32_ShortSub2(short nA, short nB)
{
return(nA-nB);
}

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

float AD32_FloatSub2(float fA, float fB)
{
return(fA-fB);
}

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

WORD AD32_GetVersion(void)
{
return(0x0100);
}

/* ------------------------------------------------- */
/* wSingDiff = 0          --> single-ended
                others --> differential

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

WORD    AD32_CheckBoard(WORD wBoard, WORD wBase, WORD wSingDiff, WORD
wIrq)
{
WORD in_1, in_2;

if (OPEN!=1) return(SysNoOpen);
if (wSingDiff!=0) wSingDiff=0x10;
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;

outportb(BASE+4,0);                /* clear FIFO */
outportb(BASE+4,0);
outportb(BASE+4,0);
outportb(BASE, CMD_READ_TEST); /* send command */
outportb(BASE,0x55);               /* send data=0x55 */

if (wait_read_fifo()==NoError)
    {
    if ((READ_DATA)!=0x55) return(CheckError);
    }
else return(TimeOut);

outportb(BASE, CMD_READ_TEST); /* send command */
outportb(BASE,0xaa);               /* send data=0xff */

if (wait_read_fifo()==NoError)
    {
    if ((READ_DATA)!=0xaa) return(CheckError);
    }
else return(TimeOut);

if (wait_write_fifo()==NoError) outportb(BASE, CMD_READ_MUX_CONFIG);
else return(TimeOut);

if (wait_read_fifo()==NoError)
    {
    in_1 = READ_DATA;
    if (wait_read_fifo()==NoError)
        {
        in_2 = READ_DATA;
        in_2 = in_1 & 0x10;
        }
```

CMD_READ_TEST
send 0x55 will return 0x55

CMD_READ_TEST
send 0xAA will return 0xAA

CMD_READ_MUX_CONFIG
read back 1 = AAAAGGGG
read back 2 = xxxDxxxA
D = 0 = JP1 select single-ended
D = 1 = JP1 select differential

```
      else return(TimeOut);
    }
    else return(TimeOut);

if (wSingDiff!=in_2) return(SingDiffError);


/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */
/* ---------- test IRQ is response or not ---------------- */


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

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

WORD    AD32_ActiveBoard(WORD wBoard)
{
if (OPEN!=1) return(SysNoOpen);
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    AD32_ReadActive(WORD *wBoard, WORD *wBase, WORD *wIrq)
{
if (OPEN!=1) return(SysNoOpen);
if (ACTIVE>=8) return(ActiveError);

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

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

WORD wait_write_fifo(void)
{
WORD t;

t = 0;
while (t < 30000)
      {
      if((inportb(BASE)&0x01)==0) t++;
      else return(NoError);
      };
return(TimeOut);
}

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

WORD wait_read_fifo(void)
{
WORD t;

t = 0;
```

Wait until COMMAND_FIFO ready, then send out COMMAND to embedded controller

Wait until DATA_FIFO ready, then read back DATA (written by embedded controller)

```
while (t < 30000)
        {
        if((inportb(BASE)&0x02)==0) t++;
        else return(NoError);
        };
return(TimeOut);
}

/* ------------------------------------------------------ */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */

WORD    AD32_SetChannelConfigNoScan(WORD channel, WORD mode, WORD config,
                        WORD wSettle)
{
WORD out_data_1, out_data_2;
WORD out_data_3, out_data_4;
WORD t;

if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

out_data_1 = channel / 16;
out_data_2 = channel%16 + mode + config;
t=wSettle*5-1;
out_data_3 = t/256;
out_data_4 = t%256;
return(send_cmd_4_arg(CMD_WRITE_MUX_CONFIG, out_data_1, out_data_2, out_data_3,
out_data_4));
}

/* ------------------------------------------------------ */
/* WRITE CMD to PIC --> need to handshake */

WORD send_cmd_2_arg(WORD out_1, WORD out_2, WORD out_3)
{
if (wait_write_fifo()==NoError)
```

CMD_READ_MUX_CONFIG
data1 = xxxxxxxA
data2 = GCGGAAAA
G = gain factory
C = single-ended/differential
A = channel address
data3 = settling time HIGH
data4 = settling time LOW

```
    {
    outportb(BASE, out_1);
    if (wait_write_fifo()==NoError)
        {
        outportb(BASE, out_2);
        if(wait_write_fifo()==NoError) outportb(BASE, out_3);
        else return(TimeOut);
        }
    else return(TimeOut);
    }
else return(TimeOut);

if (wait_read_fifo()==NoError)
    {
    out_2 = READ_DATA;
    if (out_2==out_1) return(NoError);
    else return(HandshakeError);
    }
else return(TimeOut);
}

/* ------------------------------------------------------ */
/* WRITE CMD to PIC --> need to handshake */

WORD send_cmd_3_arg(WORD out_1, WORD out_2, WORD out_3, WORD out_4)
{
if (wait_write_fifo()==NoError)
    {
    outportb(BASE, out_1);
    if (wait_write_fifo()==NoError)
        {
        outportb(BASE, out_2);
        if (wait_write_fifo()==NoError)
       {
       outportb(BASE, out_3);
       if (wait_write_fifo()==NoError) outportb(BASE, out_4);
       else return(TimeOut);
       }
```

```
          else return(TimeOut);
          }
     else return(TimeOut);
     }
else return(TimeOut);

if (wait_read_fifo()==NoError)
     {
     out_2 = READ_DATA;
     if (out_2==out_1) return(NoError);
     else return(HandshakeError);
     }
else return(TimeOut);
}

/* ------------------------------------------------------ */
/* WRITE CMD to PIC --> need to handshake */

WORD send_cmd_4_arg(WORD out_1, WORD out_2, WORD out_3, WORD out_4, WORD
out_5)
{
if (wait_write_fifo()==NoError)
     {
     outportb(BASE, out_1);
     if (wait_write_fifo()==NoError)
          {
          outportb(BASE, out_2);
          if (wait_write_fifo()==NoError)
        {
        outportb(BASE, out_3);
        if (wait_write_fifo()==NoError)
             {
             outportb(BASE, out_4);
             if (wait_write_fifo()==NoError) outportb(BASE, out_5);
             else return(TimeOut);
             }
        else return(TimeOut);
        }
```

```
        else return(TimeOut);
        }
    else return(TimeOut);
    }
else return(TimeOut);

if (wait_read_fifo()==NoError)
    {
    out_2 = READ_DATA;
    if (out_2==out_1) return(NoError);
    else return(HandshakeError);
    }
else return(TimeOut);
}
/* ------------------------------------------------------ */

WORD    AD32_ReadChannelConfigNoScan(WORD *channel, WORD *mode, WORD
*config)
{
WORD in_1, in_2;

if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

if (wait_write_fifo()==NoError) outportb(BASE, CMD_READ_MUX_CONFIG);
else return(TimeOut);

if (wait_read_fifo()==NoError)
    {
    in_1 = READ_DATA;
    if (wait_read_fifo()==NoError)
        {
        in_2 = READ_DATA;
        *config = in_2 & 0xB0;
        *mode = in_2 & 0x40;
        *channel = (in_1%2) * 16 + in_2%16;
```

CMD_READ_MUX_CONFIG
read back 1 = AAAAGGGG
read back 2 = xxxDxxxA
A = channel address
G = gain factory
D = 0 = JP1 select single-ended
D = 1 = JP1 select differential

```
        return(NoError);
        }
      else return(TimeOut);
    }
    else return(TimeOut);
}

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

WORD    AD32_ReadSingDiff(WORD *wSingDiff)
{
WORD in_1, in_2;

if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

if (wait_write_fifo()==NoError) outportb(BASE, CMD_READ_MUX_CONFIG);
else return(TimeOut);

if (wait_read_fifo()==NoError)
    {
    in_1 = READ_DATA;
    if (wait_read_fifo()==NoError)
        {
        in_2 = READ_DATA;
        *wSingDiff = (in_1 & 0x10)/16;
        return(NoError);
        }
      else return(TimeOut);
    }
    else return(TimeOut);
}

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

WORD    AD32_ReadAdNoScan(WORD *in)
{
WORD in_1, in_2;
```

CMD_READ_MUX_CONFIG
read back 1 = AAAAGGGG
read back 2 = xxxDxxxA
D = 0 = JP1 select single-ended
D = 1 = JP1 select differential

```
if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

if (wait_write_fifo()==NoError) outportb(BASE, CMD_READ_CURRENT_AD);
else return(TimeOut);
if (wait_read_fifo()==NoError)
    {
    in_1 = READ_DATA;
    if(wait_read_fifo()==NoError)
        {
        in_2 = READ_DATA;
        *in = in_2;
        *in = (*in)*256 + in_1;
        return(NoError);
        }
    else return(TimeOut);
    }
else return(TimeOut);
}

/* ------------------------------------------------------ */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */

WORD    AD32_StartAdsNoScan(WORD mode, WORD sample_count)
{
if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

return(send_cmd_2_arg(CMD_AD_SAMPLE, mode, sample_count));
}

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

WORD    AD32_ReadSystemStatus(WORD *wStatus)
{
```

CMD_READ_CURRENT_AD
read back 1 = LOW byte
read back 2 = High byte

CMD_AD_SAMPLE
data1 = mode
data2 = number of samples

```
if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

*wStatus=inportb(BASE)&0x07;
return(NoError);
}

/* ------------------------------------------------------ */
/* if FIFO empty --> read will return 0xff */
/* call read system status before read FIFO */

WORD    AD32_ReadFIFO(WORD *wData)
{
if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

*wData=READ_DATA;
return(NoError);
}

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

WORD    AD32_ClearFIFO(void)
{
if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);
outportb(BASE+4, 1);
}

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

WORD    AD32_SetSampleRate(float sample_rate)
{
WORD out_data_1, out_data_2;
float rate;
unsigned int sample_time;

if (OPEN!=1) return(SysNoOpen);
```

Read System Status Byte

Read DATA_FIFO

Clear DATA_FIFO &
COMMAND_FIFO

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

rate = (BASE_CLOCK / sample_rate) * 250.0;
sample_time = (unsigned int)(rate-1);

if (sample_time > 20)
    {
    out_data_1 = sample_time/256;
    out_data_2 = sample_time%256;
    return(send_cmd_2_arg(CMD_WRITE_SAMPLE_RATE, out_data_1, out_data_2));
    }
else return(SampleRateError);
}

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

WORD    AD32_ReadSampleRate(float *in)
{
WORD in_1, in_2;

if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

if (wait_write_fifo()==NoError) outportb(BASE, CMD_READ_SAMPLE_RATE);
else return(TimeOut);

if (wait_read_fifo()==NoError)
    {
    in_1 = READ_DATA;
    if (wait_read_fifo()==NoError)
        {
        in_2 = READ_DATA;
        *in = in_1;
/*
        *in = (*in)*256 + (float) in_2;
        *in = (BASE_CLOCK/(65554.0 - *in))/4;
        *in *= 1000.0;
        return(NoError);
```

CMD_WRITE_SAMPLE_RATE
data1 = HIGH byte
data2 = LOW byte

CMD_READ_SAMPLE_RATE
read back 1 = HIGH byte
read back 2 = LOW byte

```
*/
        *in = (*in)*256 + (float) in_2 + 1;
        *in = (BASE_CLOCK/(*in))*250.0;
        return(NoError);
        }
    else return(TimeOut);
}
else return(TimeOut);
}

/* ------------------------------------------------------ */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* ch_scan = 0,1,.... */

WORD    AD32_ReadChannelScanStatus(WORD ch_scan, WORD *channel, WORD *mode,
WORD *config)
{
WORD in_1, in_2;

if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

if (wait_write_fifo()==NoError)
    {
    outportb(BASE, CMD_READ_CH_SCAN_DATA);
    if(wait_write_fifo()==NoError) outportb(BASE, ch_scan);
    else return(TimeOut);
    }
else return(TimeOut);

if (wait_read_fifo()==NoError)
    {
    in_1 = READ_DATA;
    if (wait_read_fifo()==NoError)
        {
        in_2 = READ_DATA;
```

CMD_READ_CH_SCAN_DATA
data1 = xxxxxxxA
data2 = GCGGAAAA
C = single-ended/differential
G = Gain code

```
        *config = in_2 & 0xB0;
        *mode = in_2 & 0x40;
        *channel = (in_1%2) * 16 + in_2%16;
        return(NoError);
        }
    else return(TimeOut);
    }
else return(TimeOut);
}

/* ------------------------------------------------------ */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* ch_scan = 0,1,.... */

WORD    AD32_SetChannelScanStatus(WORD ch_scan, WORD channel, WORD mode,
WORD config)
{
WORD out_data_1, out_data_2;

if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

out_data_1 = channel / 16;
out_data_2 = channel%16 + mode + config;
return(send_cmd_3_arg(CMD_WRITE_CH_SCAN_DATA, ch_scan, out_data_1,
out_data_2));
}

/* ------------------------------------------------------ */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* WARNING : argument no check invalidate */
/* ch_scan = 1,.... */
/* ad_count no * 256 */
```

CMD_WRITE_CH_SCAN_DATA
data1 = mode
data2 = xxxxxxxA
data3 = GCGGAAAA
C = single-ended/differential
G = Gain code
A = channel address

```
WORD    AD32_StartChannelScanAds(WORD mode, WORD ch_count, WORD ad_count)
{
if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

return(send_cmd_3_arg(CMD_CHANNEL_SCAN_START, mode, ch_count, ad_count));
}
/* ------------------------------------------------- */
/*
char cBuf[80];

wRetVal=AD32_ReadPattern(cBuf);
if (wRetVal==NoError) sprintf(cShow,"17. Pattern=%s",cBuf);
else sprintf(cShow,"17. Read Pattern Error");
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
*/

WORD    AD32_ReadPattern(char in[])
{
WORD in_1, in_2;
int i;

if (OPEN!=1) return(SysNoOpen);
if (BASE==0) return(ActiveError);

if (wait_write_fifo()==NoError) outportb(BASE, CMD_READ_PATTERN);
else return(TimeOut);

for (i=0; i<36; i++)
      {
      if (wait_read_fifo()==NoError) in[i]=READ_DATA;
      else return(TimeOut);
      }
in[36]=0;
return(NoError);
}
```

CMD_CHANNEL_SCAN_START
data1 = mode
data2 = number of channels scan
data3 = numbers of samples

Read ICP DAS pattern string