Software

APAX-5522PE Embedded Linux

Software Development Manual

ADVANTECH

Enabling an Intelligent Planet

Copyright

Acknowledgements

Intel and Pentium are trademarks of Intel Corporation.

Microsoft Windows and MS-DOS are registered trademarks of Microsoft Corp.

All other product names or trademarks are properties of their respective owners.

Notes on the Manual

This is the Software Manual for the Advantech APAX-5570 product. This manual will help guide the end user through implementation and use of the software portion of this product.

What is covered in this manual:

This manual will give a general overview of the Windows XP Embedded operating system, most of the applications that are included with Windows XP Embedded as well as the applications added and/or created by Advantech Corporation in the Windows XP Embedded image. This manual will also cover installation and use of development and utility software that is needed. It will also reference optional software that can be used by the end user with the Windows XP Embedded Operating system.

What is not covered in this manual:

This manual will reference the hardware but does not contain hardware setup information, wiring information, electrical specifications or any detailed hardware information. Please refer to the hardware manual for this information.

Edition 1

October 2012

Contents

Chapter 1

Introduction

In order to save users development time, Advantech provides libraries and several examples that user can use it as reference to build their own application program in their development computer. These libraries and examples can be found in the APAX-5522 SDK or from Advantech website (in the download area under support page).

Advantech also provides some useful utility tools for user to control the APAX-5000 series I/O modules, other details information about utility tools please refer to the section 4 of APAX-5522 Embedded Linux user manual.

1.1 APAX-5522 Linux SDK overview

After decompress the SDK file, user can find the files for the tool chain, library and example code in the following paths. They are organized as shown below.

APAX-5522 SDK	
Tool Chain:	
arm-unknown-linux-gnueabi-4.2.2.tar.bz2	
Advantech Library:	
libadsdio.so.1.0.2	**APAX-5000 I/O modules Library**
libadsmod.so.1.1.4	**Advantech MODBUS Library**
libadvserial.so.1.0.0	**Advantech Serial Port Library**
Example:	
AdvAPAXIO:	
Include	
adsdio.h	Definitions and functions header for APAX-5000 I/O Library
apaxtype.h	Type definitions header for APAX Linux IO Driver and Library
Makefile	**Makefile for APAX-5000 I/O sample programs**
apax_5013_8ch_rtd.c	Sample program for APAX-5013 8-ch RTC module
apax_5017_12ch_ai.c	Sample program for APAX-5017 12-ch AI module
apax_5017H_12ch_ai.c	Sample program for APAX-5017H 12-ch High speed AI module
apax_5018_12ch_thermocouple.c	Sample program for APAX-5018 12-ch thermocouple module
apax_5028_8ch_ao.c	Sample program for APAX-5028 8-ch AO module
apax_5040_24ch_di.c	Sample program for APAX-5040 24-ch DI module
apax_5045_24ch_dio.c	Sample program for APAX-5045 24-ch DIO module
apax_5046_24ch_do.c	Sample program for APAX-5046 24-ch DO module

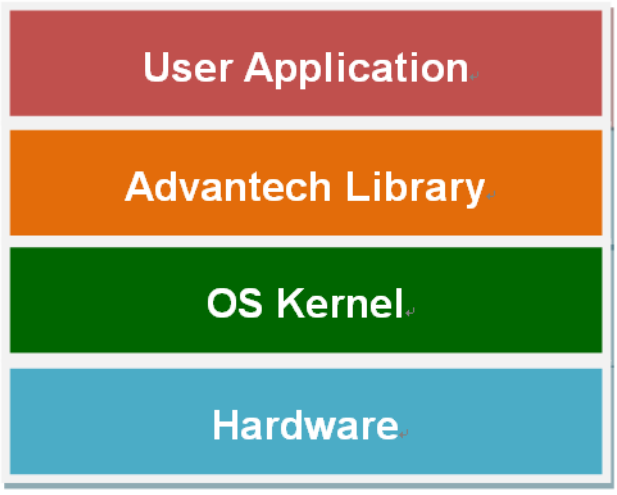
apax_5060_12ch_relay.c	Sample program for APAX-5060 12-ch Relay module
apax_5080_8ch_counter.c	Sample program for APAX-5080 8-ch Counter module
AdvModbus:	
Include	
ADSMOD.h	Definitions and functions header for Advantech MODBUS Library
adsdio.h	Definitions and functions header for APAX Linux IO Library
apaxtype.h	Type definitions header for APAX Linux IO Driver and Library
Makefile	**Makefile for Advantech MODBUS sample programs**
exModbusRtuClient.cpp	Advantech MODBUS RTU Client sample program
exModbusRtuServer.cpp	Advantech MODBUS RTU Server sample program
exModbusTcpClient.cpp	Advantech MODBUS TCP Client sample program
exModbusTcpServer.cpp	Advantech MODBUS TCP Server sample program
AdvSerialPort:	
include	
SerialPort.h	Definitions and functions header for Advantech Serial Port Library
crc.h	Definition CRC16 function header for Advantech Serial Port Library
Makefile	**Makefile for Advantech Serial Port sample program**
AdamIO_SerialTest.cpp	Sample program for serial port use ADAM IO demo
TimeStamp:	
include	
adsdio.h	Definitions and functions header for APAX-5000 I/O Library
apaxtype.h	Type definitions header for APAX Linux IO Driver and Library
Makefile	**Makefile for timestamp sample programs**
timeStampDI.c	Sample program for APAX-5040 DI module with timestamp feature
timeStampAI.c	Sample program for APAX-5017 AI module with timestamp feature
WDT (watch dog timer)	
Makefile	**Makefile for watch dog timer sample program**

testwdt.c	Watch dog timer sample program
Misc	
Query APAX-552X device IP address	
AdvGetDeviceIP-win32.exe	For Windows PC use
AdvGetDeviceIP-linux	For Linux PC use

Chapter 2

Software Architecture

The Advantech library provides higher layer API functions for user more easily developing, such as APAX-5000 I/O module, MODBUS and serial port applications. These higher layers API functions are divided into three libraries, APAX-5000 I/O module library, Advantech MODBUS library and Advantech serial port library. Information about these higher layer API functions will be introduced in later sections.



Chapter 3

Establish Develop Environment in Your PC

3.1 Install Tool Chain

Advantech provide a simple and fast cross-development environment for user developing their applications. You should have the Linux distribution pre-installed in your host computer, for example: Ubuntu Linux distribution. To install cross compile tool, please copy it to the "/ "folder, and then execute the following command to install it:

```
#  tar jxvf arm-unknown-linux-gnueabi-4.2.2.tar.bz2
```

After the installation, the development environment will be established.

You may execute "arm-unknown-linux-gnueabi-gcc -v" to check the development environment is successfully installed if you can see a similar message as below:

The cross tool chain is tested on following Linux distributions:

Fedora: Fedora10 Fedora11 Fedora12 Fedora13 Fedora14 Fedora15 Fedora16; **Ubuntu:** Ubuntu8.04 Ubuntu8.10 Ubuntu9.04 Ubuntu9.10 Ubuntu10.04Ubuntu10.10 Ubuntu11.04 Ubuntu11.10; **Debian:** Debian5.0 Debian6.0

3.2 Install Advantech Library

Before compiling the sample code or developing the applications, you may install the Advantech libraries. Please change your working directory to the Apax5522-SDK and do as following:

3.2.1 Install APAX-5000 I/O modules library

```
# cp Apax5522-SDK/AdvAPAXIO/lib/libadsdio.so.1.0.2 \
/usr/local/arm-unknown-linux-gnueabi/lib/.
# cd /usr/local/arm-unknown-linux-gnueabi/lib/
# ldconfig -n .
# ln -s libadsdio.so.1 libadsdio.so
```

After the installation, you can see three files such as libadsdio.so.1.0.2, libadsdio.so.1 and libadsdio.so in your system.

3.2.2 Install Advantech Modbus Library

```
# cp Apax5522-SDK/AdvModbus/lib/libadsmod.so.1.1.4 \
/usr/local/arm-unknown-linux-gnueabi/lib/.
# cd /usr/local/arm-unknown-linux-gnueabi/lib/
# ldconfig -n .
# ln -s libadsmod.so.1 libadsmod.so
```

After the installation, you can see three files such as libadsmod.so.1.1.4, libadsmod.so.1 and libadsmod.so in your system.

3.2.3 Install Advantech Serial Port Library

```
# cp Apax5522-SDK/AdvSerialPort/lib/ libadvserial.so.1.0.0 \
/usr/local/arm-unknown-linux-gnueabi/lib/.
# cd /usr/local/arm-unknown-linux-gnueabi/lib/
# ldconfig -n .
# ln -s libadvserial.so.1 libadvserial.so
```

After the installation, you can see three files such as libadvserial.so.1.0.0, libadvserial.so.1 and libadvserial.so in your system.

3.3 Quick start to develop application

3.3.1 Build Advantech Examples

It is easy to build sample code for Linux, For example, if user wants to build APAX-5000 I/O modules sample program, please change your working directory to the AdvAPAXIO then type "make all" command. The APAX-5000 I/O modules sample programs will be built. The sample programs will be deleting if you type "make clean" command.

- Build APAX-5000 I/O modules sample programs

```
# cd Apax5522-SDK/AdvAPAXIO/
# make all
# ls
apax_5013_8ch_rtd            apax_5013_8ch_rtd.c
apax_5017_12ch_ai            apax_5017_12ch_ai.c
apax_5017H_12ch_ai           apax_5017H_12ch_ai.c
apax_5018_12ch_thermocouple  apax_5018_12ch_thermocouple.c
apax_5028_8ch_ao             apax_5028_8ch_ao.c
apax_5040_24ch_di            apax_5040_24ch_di.c
apax_5045_24ch_dio           apax_5045_24ch_dio.c
apax_5046_24ch_do            apax_5046_24ch_do.c
apax_5060_12ch_relay         apax_5060_12ch_relay.c
apax_5080_8ch_counter        apax_5080_8ch_counter.c
…
```

- Build Advantech MODBUS sample programs

```
# cd Apax5522-SDK/AdvModbus/
# make all
# ls
exModbusRtuClient          exModbusRtuClient.cpp
exModbusRtuServer          exModbusRtuServer.cpp
exModbusTcpClient          exModbusTcpClient.cpp
exModbusTcpServer          exModbusTcpServer.cpp
…
```

- Build Advantech Serial port sample programs

```
# cd Apax5522-SDK/AdvSerialPort/
# make all
# ls
apax_SerialPort            apax_SerialPort.cpp
…
```

3.3.2 Develop Your Application

In this section, we will introduce how to develop a program for the APAX-5522 Embedded Linux. In general, program development involves the following steps.

1. Install the Tool Chain (GNU Cross Compiler & glib c). --- see Section 3.1
2. Install the Advantech Library. --- see Section 3.2
3. Write your program and include the Advantech library header files.
4. Write Makefile to manage your program.

Please note that include header path and the Advantech library that you are using.

5. Build and debug your program.
6. Upload your program and your library to the APAX-5522 Embedded Linux. --- see Appendix A

In addition, users also can refer to our example programs to quickly develop their own programs. For example, develop a program of APAX-5000 I/O modules.

1. Write your programs (for example, named my_test_app.c and my_test_app.h)
2. Put the source file "my_test_app.c" under the path "Apax5522-SDK/AdvA-PAXIO"
   Put the header file "my_test_app.h" under the path "Apax5522-SDK/AdvA-PAXIO/include"
3. Modify Makefile to manage your program. For example:

```
CC=arm-unknown-linux-gnueabi-gcc -fpack-struct
CFLAGS =-I./include -O2 -Wall -g
LDDIR=-L.
LDFLAGS=-ladsdio
BINS=apax_5040_24ch_di apax_5060_12ch_relay my_test_app
all: $(BINS)
$(BINS): % : %.c
        $(CC) $(CFLAGS) -o $@ $< $(LDDIR) $(LDFLAGS)
clean:
        rm -f *.o $(BINS) $(MBINS)
```

4. Type "make all" to build your program.
5. Upload **my_test_app** to the APAX-5522 Embedded Linux. You can refer to the section of Appendix A to upload your program to APAX-5522 Embedded Linux.

Chapter 4

Introduction
Advantech Library

This section includes important information for programmers. The Advantech library provides higher layer API functions for programmers easy to develop their own applications. The Advantech library is divided into APAX-5000 I/O module, MODBUS and serial port three libraries. The following sections will explain these API functions of the libraries.

4.1 APAX-5000 I/O modules library

The libadsdio.so.1.0.2 is a library file that is designed for APAX-5000 I/O module applications. Base on this library, user can develop their applications to achieve control of the APAX-5000 I/O modules purpose.

The API functions in the library are divided into the following categories:

- System functions
- Analog Input/output functions
- Digital Input/output functions
- Counter functions
- Timestamp functions are now only support at APAX-5017PE and APAX-5040PE modules.

4.1.1 System functions

4.1.1.1 ADAMDrvOpen

LONG ADAMDrvOpen(LONG* handle);
■ Purpose: This function opens a handle that operates the APAXIO driver. ■ Parameters: handle = driver handle ■ Return 1. ERR_SUCCESS, Driver initialization succeeded, the handle will be valid for function use until closed. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.1.2 ADAMDrvClose

LONG ADAMDrvClose(LONG* handle);
■ Purpose: This function closes the open handle of the APAXIO driver. ■ Parameters: handle = driver handle ■ Return 1. ERR_SUCCESS, Driver termination succeeded 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.1.3 SYS_SetInnerTimeout

LONG SYS_SetInnerTimeout(LONG handle, WORD i_wTimeout);
■ Purpose: Set the inner-timeout of the configuration functions that use internal communication channel. All functions with the exception of Get/Set values use the internal communication network. When using any of those functions, they must wait for completion before returning. This sets the timeout value for returning. ■ Parameters: handle = driver handle i_wTimeout = inner-timeout, in millisecond. Default is 50 milliseconds. ■ Return 1. ERR_SUCCESS, Set timer succeeded 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.1.4 SYS_GetModuleID

LONG SYS_GetModuleID (LONG handle, WORD i_wSlot, DWORD* o_dwModuleID);
■ Purpose: Get the module ID of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. o_dwModuleID = returned module ID ■ Return 1. ERR_SUCCESS, Module ID was found and returned 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.1.5 SYS_GetSlotInfo

LONG SYS_GetSlotInfo (LONG handle, WORD i_wSlot, struct SlotInfo*o_stSlotInfo);
■ Purpose: Get the module information of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. o_stSlotInfo = returned SlotInfo structure. ■ Return 1. ERR_SUCCESS, o_stSlotInfo contains slot information. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.1.6 SYS_GetVersion

LONG SYS_GetVersion(LONG handle, DWORD* o_dwVersion);
■ Purpose: Get the ADSDIO library version ■ Parameters: handle = driver handler o_dwVersion = ADSDIO library version number ■ Return ERR_SUCCESS

4.1.1.7 SYS_SetLocateModule

LONG SYS_SetLocateModule(LONG handle, WORD i_wSlot, WORD i_wTimes);
■ Purpose: Set the LED light flashing of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wTimes = the LED flashing time. ■ Return 1. ERR_SUCCESS, setting succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.1.8 SYS_SetRunLED

LONG SYS_SetRunLED(LONG handle, DWORD dwValue);
■ Purpose: Set the RUN LED light ON/OFF. ■ Parameters: handle = driver handler dwValue = 1 ON 0 OFF ■ Return 1. ERR_SUCCESS, setting succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.1.9 SYS_SetErrLED

LONG SYS_SetErrLED(LONG handle, DWORD dwValue);
■ Purpose: Set the ERR LED light ON/OFF. ■ Parameters: handle = driver handler dwValue = 1 ON 0 OFF ■ Return 1. ERR_SUCCESS, setting succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2 Analog Input/output functions

4.1.2.1 AIO_GetValue

LONG AIO_GetValue(LONG handle, WORD i_wSlot,WORD i_wChannel, WORD* o_wValue);
■ Purpose: Get a single analog input or output value from the indicated slot and channel. ■ Parameters: handle = driver handle i_wSlot = the slot ID which has a range of 0 to 31. i_wChannel = the channel ID which has a range of 0 to 31. o_wValue = the variable to hold the returned AIO value. ■ Return 1. ERR_SUCCESS, o_wValue contains AIO value. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.2 AIO_GetValues

LONG AIO_GetValues(LONG handle, WORD i_wSlot, WORD* o_wValues);
■ Purpose: Get the all analog input or output values of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. o_wValues = the variables array to hold the returned AIO values. The size of this array must be at least 32 WORD's. ■ Return 1. ERR_SUCCESS, o_wValue contains AIO values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.3 AIO_SetRanges

LONG AIO_SetRanges(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, WORD* i_wRanges);
■ Purpose: Set the channel ranges of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannelTotal = the channel total of the module in the indicated slot. i_wRanges=the ranges to be set. The size of this array must be i_wChannelTotal WORDs. ■ Return 1. ERR_SUCCESS, setting ranges succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.4 AIO_SetZeroCalibration

LONG AIO_SetZeroCalibration(LONG handle, WORD i_wSlot, WORD i_wChannel, WORD i_wType);
■ Purpose: Run the zero calibration of the indicated slot and channel. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannel = the channel ID which is ranged from 0 to 31. i_wType = the type value to be set. Currently, it is ignored. ■ Return 1. ERR_SUCCESS, setting zero calibration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.5 AIO_SetSpanCalibration

LONG AIO_SetSpanCalibration(LONG handle, WORD i_wSlot, WORD i_wChannel, WORD i_wType);
■ Purpose: Run the span calibration of the indicated slot and channel. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannel = the channel ID which is ranged from 0 to 31. i_wType = the type value to be set. Currently, it is ignored. ■ Return 1. ERR_SUCCESS, setting span calibration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.6 AIO_GetChannelStatus

LONG AIO_GetChannelStatus(LONG handle, WORD i_wSlot, BYTE* o_byStatus);
■ Purpose: Get all channels status of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. o_byStatus = the array to hold the returned channels status. The size of this array must be at least 32 BYTEs. ■ Return 1. ERR_SUCCESS, channel status succeeded. The value of o_byStatus indicates: 0: None 1: Normal 2: Over current 3: Under current 4: Burn out 5: Open loop 6: Not ready 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.7 AI_SetChannelMask

LONG AI_SetChannelMask(LONG handle, WORD i_wSlot, DWORD i_dwMask);
■ Purpose: Set enabled AI channel mask of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_dwMask = the enabled AI channel mask to be set. From LSB to MSB of the value indicate the slot-0 to slot-31 enabled mask. If the bit is 1, it means that the channel is enabled. ■ Return 1. ERR_SUCCESS, setting channel mask succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.8 AI_SetIntegrationTime

LONG AI_SetIntegrationTime(LONG handle, WORD i_wSlot, DWORD i_dwIntegration);
■ Purpose: Set AI integration time of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_dwIntegration = the AI integration time to be set. Two settings are available. 0 = 60Hz 1 = 50Hz ■ Return 1. ERR_SUCCESS, setting integration time succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.9 AI_SetAutoCalibration

LONG AI_SetAutoCalibration(LONG handle, WORD i_wSlot);
■ Purpose: Set to run the auto calibration of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. ■ Return 1. ERR_SUCCESS, setting auto calibration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.10 AI_SetChValOffset

LONG AI_SetChValOffset(LONG handle, WORD i_wSlot, WORD i_wChannel, DWORD i_dwOffset);
■ Purpose: Set the channel value offset of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannel = the channel ID which is ranged from 0 to 31 for normal value offset. Set this value to 0xFE indicates the value offset is for CJC offset. i_dwOffset = the offset value to be set. ■ Return 1. ERR_SUCCESS, setting value offset succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.11 AI_GetChValOffset

LONG AI_GetChValOffset(LONG handle, WORD i_wSlot, WORD i_wChannel, DWORD* o_dwOffset);
■ Purpose: Get the channel value offset of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannel = the channel ID which is ranged from 0 to 31 for normal value offset. Set this value to 0xFE indicates the value offset is for CJC offset. o_dwOffset = the variable to hold the offset value. ■ Return 1. ERR_SUCCESS, getting value offset succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.12 AI_GetCjcValue

LONG AI_GetCjcValue(LONG handle, WORD i_wSlot, DWORD* o_dwValue, BYTE* o_byStatus);
■ Purpose: Get the CJC value of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_dwValue = the variable to hold the CJC value. o_byStatus = the CJC setting status. ■ Return 1. ERR_SUCCESS, getting CJC value succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.13 AI_SetBurnoutFunEnable

LONG AI_SetBurnoutFunEnable(LONG handle, WORD i_wSlot, DWORD i_dwEnableMask);
■ Purpose: Set to enable/disable burnout function for each channel. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwEnableMask = the enable mask; each bit indicates one channel, bit = 1 enable burnout function for that channel ■ Return 1. ERR_SUCCESS, setting burnout function succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.14 AI_GetBurnoutFunEnable

LONG AI_GetBurnoutFunEnable(LONG handle, WORD i_wSlot, DWORD* o_dwEnableMask);
■ Purpose: Get the burnout function enable/disable for each channel. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_dwEnableMask = the variables to hold the enabled mask. ■ Return 1. ERR_SUCCESS, getting burnout function succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.15 AI_SetCjcInitValRecord

LONG AI_SetCjcInitValRecord(LONG handle, WORD i_wSlot);
■ Purpose: Set to record the CJC initial value. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. ■ Return 1. ERR_SUCCESS, setting CJC initial value succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.16 AI_SetThermoCalibration

LONG AI_SetThermoCalibration(LONG handle, WORD i_wSlot);
■ Purpose: Set to calibrate the thermocouple. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. ■ Return 1. ERR_SUCCESS, setting thermocouple calibration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.17 AI_SetBurnoutValue

LONG AI_SetBurnoutValue(LONG handle, WORD i_wSlot, DWORD i_dwValue);
■ Purpose: Set the burnout value. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwValue = the burnout value; when input value exceeds this value, the burnout status will on. ■ Return 1. ERR_SUCCESS, setting burnout value succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.18 AI_GetBurnoutValue

LONG AI_GetBurnoutValue(LONG handle, WORD i_wSlot, DWORD* o_dwValue);
■ Purpose: Get the burnout value. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_dwValue = the variables to hold the burnout value. ■ Return 1. ERR_SUCCESS, getting burnout value succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.19 AO_SetCalibrationMode

LONG AO_SetCalibrationMode(LONG handle, WORD i_wSlot);
■ Purpose: Set to switch to the AO calibration mode of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. ■ Return 1. ERR_SUCCESS, setting calibration mode succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.20 AO_GetStartupValues

LONG AO_GetStartupValues(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, WORD* o_wValues);
■ Purpose: Get the AO startup values of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannelTotal = the channel total of the module in the indicated slot. o_wValues = the variables array to hold the AO startup values. The size of this array must be at least 32 WORDs. ■ Return 1. ERR_SUCCESS, Getting values succeeded. o_wValues contains AO startup values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.21 AO_SetStartupValues

LONG AO_SetStartupValues(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, WORD* i_wValues);
■ Purpose: Set the AO startup values of the indicated slot. These values are stored in onboard flash and are initialized to the slot upon boot up of the hardware. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannelTotal = the channel total of the module in the indicated slot. i_wValues = the values array to be set. The size of this array must be i_wChannelTotal WORDs. ■ Return 1. ERR_SUCCESS, setting values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.22 AO_SetValue

LONG AO_SetValue(LONG handle, WORD i_wSlot, WORD i_wChannel, WORD i_wValue);
■ Purpose: Set a single AO value of the indicated slot and channel. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannel = the channel ID which is ranged from 0 to 31. i_wValue = the AO value to be set. ■ Return 1. ERR_SUCCESS, 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.23 AO_SetValues

LONG AO_SetValues(LONG handle, WORD i_wSlot, DWORD i_dwMask, WORD* i_wValues);
■ Purpose: Set multiple AO values of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_dwMask = the channels mask. From LSB to MSB of the value indicate the slot-0 to slot-31 mask. If the bit is 1, it means that the channel must change value. i_wValues = the AO values to be set. This is a pointer to an array of 32 words. ■ Return 1. ERR_SUCCESS, setting values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.24 AO_BufValues

LONG AO_BufValues(LONG handle, WORD i_wSlot, DWORD i_dwMask, WORD* i_wValues);
■ Purpose: Buffer the AO values of the indicated slot. This function is used along with OUT_FlushBufValues for a synchronized write Output. Once all slots are buffered, then OUT_FlushBufValues function triggers the synchronized buffer write of all masked slots. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_dwMask = the channels mask. From LSB to MSB of the value indicate the slot-0 to slot-31 mask. If the bit is 1, it means that the channel must buffer value. i_wValues = the AO values to be buffered. ■ Return 1. ERR_SUCCESS, buffering values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.25 AO_SetSaftyValues

LONG AO_SetSaftyValues(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, WORD* i_wValues);
■ Purpose: Set the AO safty values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannelTotal = the channel total of the module in the indicated slot. i_wValues = the AO safty values to be set. ■ Return 1. ERR_SUCCESS, setting values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.2.26 AO_GetSaftyValues

LONG AO_GetSaftyValues(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, WORD* o_wValues);
■ Purpose: Get the all AIO safty values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannelTotal = the channel total of the module in the indicated slot. o_wValues = the variables array to hold the AIO safty values. The size of this array must be at least 32 WORDs. ■ Return 1. ERR_SUCCESS, getting values succeeded. And the o_wValue contains AIO values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3 Digital Input/output functions

4.1.3.1 DIO_GetValue

LONG DIO_GetValue(LONG handle, WORD i_wSlot, WORD i_wChannel, BOOL* o_bValue);
■ Purpose: Get a single DIO value of the indicated slot and channel. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannel = the channel ID which is ranged from 0 to 31. o_bValue = the variable to hold the DIO value. ■ Return 1. ERR_SUCCESS, getting value succeeded o_wValue contains DIO value. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.2 DIO_GetValues

LONG DIO_GetValues(LONG handle, WORD i_wSlot, DWORD* o_dwHighValue, DWORD* o_dwLowValue);
■ Purpose: Get the all DIO values of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. o_dwHighValue = the variable to hold the returned DIO values from channel 32 to 63. The LSB indicates the channel-32. o_dwLowValue = the variable to hold the returned DIO values from channel 0 to 31. The LSB indicates the channel-0. ■ Return 1. ERR_SUCCESS, Get values succeeded. The o_dwHighValue and o_dwLowValue contain DIO values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.3 DIO_SetSaftyValues

LONG DIO_SetSaftyValues(LONG handle, WORD i_wSlot, DWORD i_dwHighValue, DWORD i_dwLowValue);
■ Purpose: Set the DO safty values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 31. i_dwHighValue = the DO safty values from channel 32 to 63 to be set. The LSB indicates the channel-32. i_dwLowValue = the DO safty values from channel 0 to 31 to be set. The LSB indicates the channel-0. ■ Return 1. ERR_SUCCESS, setting safty values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.4 DIO_GetSaftyValues

LONG DIO_GetSaftyValues(LONG handle, WORD i_wSlot, DWORD* o_dwHighValue, DWORD* o_dwLowValue);
■ Purpose: Get the all DIO safty values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 31. o_dwHighValue = the variable to hold the DIO safty values from channel 32 to 63. The LSB indicates the channel-32. o_dwLowValue = the variable to hold the DIO safty values from channel 0 to 31. The LSB indicates the channel-0. ■ Return 1. ERR_SUCCESS, getting values succeeded, and the o_dwHighValue and o_dwLowValue contain DIO values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.5 DI_GetFilters

LONG DI_GetFilters(LONG handle, WORD i_wSlot, DWORD* o_dwHighMask, DWORD* o_dwLowMask, DWORD* o_dwWidth);
■ Purpose: Get the DI filter mask and width of the indicated slot. All channels use the same filter. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. o_dwHighMask = RESERVED o_dwLowMask = If set to zero, filter is disabled. Non-zero indicates that filter is applied. o_dwWidth = the variable to hold the DI filter width. Filter is in .1msec units and value of filter width must be in multiples of 5. ■ Return 1. ERR_SUCCESS, get filters succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.6 DI_SetFilters

LONG DI_SetFilters(LONG handle, WORD i_wSlot, DWORD i_dwHighMask, DWORD i_diwLowMask, DWORD i_dwWidth);
■ Purpose: Set the DI filter mask and width of the indicated slot. Filter is amount of time needed to verify a change of state. This is to reduce noise. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_dwHighMask = RESERVED i_dwLowMaski = If set to zero, filter is disabled. Non-zero indicates that filter is applied. i_dwWidth = the variable to hold the DI filter width. Filter is in .1msec units and value of filter width must be in multiples of 5. ■ Return 1. ERR_SUCCESS, setting filter succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.7 DO_SetValue

LONG DO_SetValue(LONG handle, WORD i_wSlot, WORD i_wChannel, BOOL i_bValue);
■ Purpose: Set a single DO value of the indicated slot and channel. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_wChannel = the channel ID which is ranged from 0 to 31. i_bValue = the DO value to be set. ■ Return 1. ERR_SUCCESS, setting value succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.8 DO_SetValues

LONG DO_SetValues(LONG handle, WORD i_wSlot, DWORD i_dwHighValue, DWORD i_dwLowValue);
■ Purpose: Set all DO values of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_dwHighValue = the DI values from channel 32 to 63 to be set. The LSB indicates the channel-32. i_dwLowValue = the DI values from channel 0 to 31 to be set. The LSB indicates the channel-0. ■ Return 1. ERR_SUCCESS, setting values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.9 DO_BufValues

LONG DO_BufValues(LONG handle, WORD i_wSlot, DWORD i_dwHighValue, DWORD i_dwLowValue);
■ Purpose: Buffer the DO values of the indicated slot. ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. i_dwHighValue = the DI values from channel 32 to 63 to be buffered. The LSB indicates the channel-32. i_dwLowValue = the DI values from channel 0 to 31 to be buffered. The LSB indicates the channel-0. ■ Return 1. ERR_SUCCESS, buffering values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.10 OUT_FlushBufValues

LONG OUT_FlushBufValues(LONG handle, DWORD i_dwSlotMask);
■ Purpose: Flush the buffered values. This triggers all buffered values to write simultaneously. ■ Parameters: handle = driver handle i_dwSlotMask = the flush slot enable mask. The LSB indicates the slot-0. ■ Return 1. ERR_SUCCESS, flushing values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.11 OUT_SetSaftyEnable

LONG OUT_SetSaftyEnable(LONG handle, WORD i_wSlot, BOOL i_bEnable);
■ Purpose: Set the safty value enabled status of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_bEnable = the safty value enabled status. ■ Return 1. ERR_SUCCESS, setting enabled status succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.3.12 OUT_GetSaftyEnable

LONG OUT_GetSaftyEnable(LONG handle, WORD i_wSlot, BOOL* o_bEnable);
■ Purpose: Get the safty value enabled status of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_bEnable = the variable hold the safty value enabled status. ■ Return 1. ERR_SUCCESS, getting enabled status succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4 Counter functions

4.1.4.1 CNT_GetValue

LONG CNT_GetValue(LONG handle, WORD i_wSlot, WORD i_wChannel, DWORD* o_dwValue);
■ Purpose: Get the counter value of the indicated slot and channel. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannel = the channel ID which is ranged from 0 to 7. o_dwValue = the variable to hold the counter value. ■ Return 1. ERR_SUCCESS, getting value succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.2 CNT_GetValues

LONG CNT_GetValues(LONG handle, WORD i_wSlot, DWORD* o_dwValues);
■ Purpose: Get the all counter values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_dwValues = the variables array to hold the counter values. The size of this array must be at least 8 DWORDs. ■ Return 1. ERR_SUCCESS, getting value succeeded. The o_dwValues contains counter values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.3 CNT_ClearValues

LONG CNT_ClearValues(LONG handle, WORD i_wSlot, DWORD i_dwMask);
■ Purpose: Clear the masked counter values to startup values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwMask = the channels mask. From LSB to MSB of the value indicate the channel-0 to channel-31 mask. If the bit is 1, it means that the channel must set to startup value. ■ Return 1. ERR_SUCCESS, clearing value succeeded. The o_dwValues contains counter values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.4 CNT_SetRanges

LONG CNT_SetRanges(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, WORD* i_wRanges);
■ Purpose: Set the channel ranges of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannelTotal = the channel total of the module in the indicated slot. i_wRanges = the ranges to be set. The size of this array must be i_wChannelTotal WORDs. the range definition is as below: 0 : Bi-directory 1 : Up/Down 2 : Up 3 : Frequency 4 : A/B-1X 5 : A/B-2X 6 : A/B-4X ■ Return 1. ERR_SUCCESS, setting ranges succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.5 CNT_SetChannelMask

LONG CNT_SetChannelMask(LONG handle, WORD i_wSlot, DWORD i_dwMask);
■ Purpose: Set enabled counter channel mask of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwMask = the enabled counter channel mask to be set. From LSB to MSB of the value indicate the channel-0 to channel-31 enabled mask. If the bit is 1, it means that the channel started. ■ Return 1. ERR_SUCCESS, setting channel mask succeeded. The o_dwValues contains counter values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.6 CNT_GetFilter

LONG CNT_GetFilter(LONG handle, WORD i_wSlot, DWORD* o_dwWidth);
■ Purpose: Get the counter filter width of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_dwWidth = the variable to hold the counter digital filter width. ■ Return 1. ERR_SUCCESS, getting filter succeeded. The o_dwWidth contains the counter filter width. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.7 CNT_SetFilter

LONG CNT_SetFilter(LONG handle, WORD i_wSlot, DWORD i_dwWidth);
■ Purpose: Set the counter filter width of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwWidth = the counter filter width to be set. ■ Return 1. ERR_SUCCESS, getting filter succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.8 CNT_GetStartupValues

LONG CNT_GetStartupValues(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, DWORD* o_dwValues);
■ Purpose: Get the counter startup values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannelTotal = the channel total of the module in the indicated slot. The maximum value is 8. o_dwValues = the variables array to hold the counter startup values. The size of this array must be at least 8 DWORDs. ■ Return 1. ERR_SUCCESS, getting values succeeded. The o_dwValues contains counter startup values from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.9 CNT_SetStartupValues

LONG CNT_SetStartupValues(LONG handle, WORD i_wSlot, WORD i_wChannelTotal, DWORD* i_dwValues);
■ Purpose: Set the counter startup values of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannelTotal = the channel total of the module in the indicated slot. i_dwValues = the values array to be set. The size of this array must be i_wChannelTotal DWORDs. ■ Return 1. ERR_SUCCESS, setting values succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.10 CNT_ClearOverflows

LONG CNT_ClearOverflows(LONG handle, WORD i_wSlot, DWORD i_dwMask);
■ Purpose: Clear the counter overflow of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwMask = the counter channel mask to be set. From LSB to MSB of the value indicate the channel-0 to channel-31 mask. If the bit is 1, it means that the overflow of the channel must be cleared. ■ Return 1. ERR_SUCCESS, clearing overflows succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.11 CNT_GetAlarmFlags

LONG CNT_GetAlarmFlags(LONG handle, WORD i_wSlot, DWORD* o_dwFlags);
■ Purpose: Get the counter alarm of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_dwFlags = the variable to hold the counter alarm flags. ■ Return 1. ERR_SUCCESS, getting alarm flags succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.12 CNT_ClearAlarmFlags

LONG CNT_ClearAlarmFlags(LONG handle, WORD i_wSlot, DWORD i_dwMask);
■ Purpose: Clear the counter alarm of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwMask = the alarm flags mask. ■ Return 1. ERR_SUCCESS, clearing alarm flags succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.13 CNT_GetAlarmConfig

LONG CNT_GetAlarmConfig(LONG handle, WORD i_wSlot, WORD i_wAlarmIndex, BOOL* o_bEnable, BOOL* o_bAutoReload, BYTE* o_byType, BYTE* o_byMapChannel, DWORD* o_dwLimit, BYTE* o_byDoType, DWORD* o_dwDoPulseWidth);
■ Purpose: Get the counter alarm configuration of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wAlarmIndex = the alarm index o_bEnable = the variable to hold the alarm enabled status. o_bAutoReload = the variable to hold the alarm auto reload status. o_byType = the variable to hold the alarm type. 0 : Low alarm 1 : High alarm o_byMapChannel = the variable to hold the counter channel that the alarm mapped to. o_dwLimit = the variable to hold the counter limit which will fire the alarm. o_byDoType = the variable to hold the DO type. 0 : Low level 1 : High level 2 : Low pulse 3 : High pulse o_dwDoPulseWidth = the variable to hold the DO pulse width. ■ Return 1. ERR_SUCCESS, getting alarm configuration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.14 CNT_SetAlarmConfig

LONG CNT_SetAlarmConfig(LONG handle, WORD i_wSlot, WORD i_wAlarmIndex, BOOL i_bEnable, BOOL i_bAutoReload, BYTE i_byType, BYTE i_byMapChannel, DWORD i_dwLimit, BYTE i_byDoType, DWORD i_dwDoPulseWidth);
■ Purpose: Set the counter alarm configuration of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wAlarmIndex = the alarm index i_bEnable = the alarm enabled status to be set. i_bAutoReload = the alarm auto reload status to be set. i_byType = the alarm type to be set. 0 : Low alarm 1 : High alarm i_byMapChannel = the counter channel that the alarm mapped to. i_dwLimit = the counter limit which will fire the alarm. i_byDoType = the DO type to be set. 0 : Low level 1 : High level 2 : Low pulse 3 : High pulse i_dwDoPulseWidth = the DO pulse width to be set. ■ Return 1. ERR_SUCCESS, setting alarm configuration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.15 CNT_GetGateConfig

LONG CNT_GetGateConfig(LONG handle, WORD i_wSlot, WORD i_wChannel, BOOL* o_bEnable, BYTE* o_byTriggerMode, BYTE* o_byGateActiveType, BYTE* o_byMapGate);
■ Purpose: Get the counter gate configuration of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannel = the counter index o_bEnable = the variable to hold the gate enabled status. o_byTriggerMode = the variable to hold the trigger mode. 0 : Non re-trigger 1 : Re-trigger 2 : Edge start o_byGateActiveType = the variable to hold the gate active type. 0 : Low level 1 : Falling edge 2 : High level 3 : Rising edge o_byMapGate = the variable to hold the gate that the counter mapped to. ■ Return 1. ERR_SUCCESS, getting gate configuration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.16 CNT_SetGateConfig

LONG CNT_SetGateConfig(LONG handle, WORD i_wSlot, WORD i_wChannel, BOOL i_bEnable, BYTE i_byTriggerMode, BYTE i_byGateActiveType, BYTE i_byMapGate);
■ Purpose: Set the counter gate configuration of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannel = the counter index i_bEnable = the gate enabled status. i_byTriggerMode = the trigger mode. 0 : Non re-trigger 1 : Re-trigger 2 : Edge start i_byGateActiveType = the gate active type. 0 : Low level 1 : Falling edge 2 : High level 3 : Rising edge i_byMapGate = the gate that the counter mapped to. ■ Return 1. ERR_SUCCESS, setting gate configuration succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.17 CNT_GetCntTypeConfig

LONG CNT_GetCntTypeConfig(LONG handle, WORD i_wSlot, WORD i_wChannel, BOOL* o_bRepeat, BOOL* o_bReload);
■ Purpose: Get the counter counting type of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannel = the counter index o_bRepeat = the variable to hold the repeat enabled status. o_bReload = the variable to hold the reload to startup enabled status. ■ Return 1. ERR_SUCCESS, setting counting type succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.18 CNT_SetCntTypeConfig

LONG CNT_SetCntTypeConfig(LONG handle, WORD i_wSlot, WORD i_wChannel, BOOL i_bRepeat, BOOL i_bReload);
■ Purpose: Set the counter counting type of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_wChannel = the counter index i_bRepeat = the repeat enabled status. i_bReload = the reload to startup enabled status. ■ Return 1. ERR_SUCCESS, setting counting type succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.19 CNT_GetChannelStatus

LONG CNT_GetChannelStatus(LONG handle, WORD i_wSlot, BYTE* o_byStatus);
■ Purpose: Get all channels status of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. o_byStatus = the variables array to hold the channels status. The size of this array must be at least 8 BYTEs. ■ Return 1. ERR_SUCCESS, getting channel status succeeded. The value of o_byStatus indicates: 1: Normal 8: Over flow 9: Under flow 10 : Over and Under flow 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.4.20 CNT_SetFreqAcqTime

LONG CNT_SetFreqAcqTime(LONG handle, WORD i_wSlot, DWORD i_dwFreqAcqTime);
■ Purpose: Set the counter frequency acquisition time of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 15. i_dwFreqAcqTime = the frequency acquisition time ■ Return 2. ERR_SUCCESS, setting frequency acquisition time succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.5 Timestamp functions

4.1.5.1 SYS_ClearTSRingBuffer

LONG SYS_ClearTSRingBuffer(LONG handle, WORD i_wSlot);
■ Purpose: Clear the time stamp ring buffer of the indicated slot. 0xFF means clear all buffer ■ Parameters: handle = driver handle i_wSlot = the slot ID which is ranged from 0 to 31. ■ Return 1. ERR_SUCCESS, Clear the time stamp ring buffer succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.5.2 SYS_StartStopTimeStampCounter

LONG SYS_StartStopTimeStampCounter(LONG handle, DWORD i_dwValue);
■ Purpose: Start/Stop the time stamp counter. ■ Parameters: handle = driver handle i_dwValue = 1, start the time stamp counter value. = 0, stop the time stamp counter and clear the value to zero. ■ Return 1. ERR_SUCCESS, Start/Stop the time stamp counter succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.5.3 SYS_GetTimeStampCounter

LONG SYS_GetTimeStampCounter(LONG handle, DWORD* o_dwValue);
■ Purpose: Get the time stamp counter value. ■ Parameters: handle = driver handle o_dwValue = the variable to hold the time stamp counter value. Please note that every 500 micro seconds DSP counted once. Value 200 means 100 milliseconds. ■ Return 1. ERR_SUCCESS, Get the time stamp counter succeeded. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

4.1.5.4 AI_GetValuesWithTimeStamp

LONG AI_GetValuesWithTimeStamp(LONG handle, WORD i_wSlot, DWORD *o_dwTSData);
■ Purpose: Get the all AI values with time stamp of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 31. o_dwTSData = the variable to hold the AI values with time stamp information. ■ Return 1. ERR_SUCCESS, If getting values succeeded, function returns the amount of the data (Max is 32) and the o_dwTSData contain AI values and time stamp of the indicated channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

[Note]

The variable of o_dwTSData holds the AI value with time stamp information. Please refer to the following definition:

	Bit15 ~ Bit8	Bit7 ~ Bit0
Offset +0	AI_0 Value	
Offset +1	Channel index (0)	Reserved for 24 bit AI Value
Offset +2	AI_0 Timestamplow	
Offset +3	AI_0 Timestamphigh	

We for the stamp information define a struct array as below:

```
struct  AiSingleTSData
{
        WORD  wAiData;
        WORD  wChannelIdx;
        DWORD  dwTimeStamp;
};
```

- Each module of the ring buffer size is 256 bytes
- The total length of AI data is 8 bytes
- The data quantity of slot is 32. (256/8)
- The sample speed is 83 msec.
- Depth for APAX-5017PE = 83ms * 32 = 2.656 sec.

4.1.5.5 DI_GetValuesWithTimeStamp

LONG DI_GetValuesWithTimeStamp(LONG handle, WORD i_wSlot, DWORD *o_dwTSData);
■ Purpose: Get the all DI values with time stamp of the indicated slot. ■ Parameters: handle = driver handler i_wSlot = the slot ID which is ranged from 0 to 31. o_wTSData = the variable to hold the DI values with time stamp information. ■ Return 1. ERR_SUCCESS, If getting values succeeded, function returns the amount of the data (Max is 32) and the o_dwTSData contain DI values and time stamp from channel-0 to the last channel. 2. ERR_INTERNAL_FAILED, Call perror to get extended error information.

[Note]

Tthe variable of o_dwTSData hold the DI values with time stamp information. Please refer to the following definition:

	Bit15 ~ Bit0
Offset +0	DI Values 00~15
Offset +1	DI Values 16~31
Offset +2	Timestamplow
Offset +3	Timestamphigh

We for the stamp information define a struct array as below:

```
struct  Di32TSData
{
        DWORD  dwDi32Data;
        DWORD  dwTimeStamp;
};
```

- Each module of the ring buffer size is 256 bytes
- The total length of DI data is 8 bytes
- The data quantity of slot is 32. (256/8)
- The sample speed is 1 msec.
- Depth for APAX-5040PE = 1ms * 32 = 32 msec.

At later section will illustrate how to use these APIs to get the DI and AI values with time stamp information.

4.1.6 Error code definition

Error ID	Error Code
ERR_SUCCESS	0
ERR_MALLOC_FAILED	300
ERR_MAPADDR_FAILED	301
ERR_HANDLE_INVALID	302
ERR_MODULE_INVALID	303
ERR_SLOT_INVALID	304
ERR_CHANNEL_INVALID	305
ERR_FUNC_INVALID	306
ERR_INTRINIT_FAILED	307
ERR_FREQMEASU_FAILED	308
ERR_PARAM_INVALID	309
ERR_FIFO_NOTREADY	310
ERR_FIFO_FULL	311
ERR_FIFO_DATAFAILED	312
ERR_ACQSTOP_FAILED	313
ERR_DEVICE_NON	320
ERR_ACCESS_DENIED	321
ERR_LENGTH_INVALID	322
ERR_CONFIG_FAILED	323
ERR_DSPFLAG_INVALID	324
ERR_INTERNAL_FAILED	325
ERR_TIMEOUT	326
ERR_COMMAND_FAILED	390

4.2 APAX-5000 I/O Module Examples Demo

After decompress the SDK file, user can find the files for the example code under the path "Apax5522-SDK/AdvAPAXIO". In this section, we will show some examples, these examples were used APAX-5000 I/O modules, DIO module use APAX-5045, AI module use APAX-5017, AO module use APAX-5028 and Counter module use APAX-5080, so that users can more easily use these API functions described in the previous section.

4.2.1 DIO Module Example

The example of "apax_5045_24ch_dio.c" will illustrate how to control DI/DO with the APAX-5045 module. The APAX-5045 has 12 digital input channels and 12 digital output channels. The purpose of this example will get the DI channels status and set the DO channels ON or OFF. The mainly source code of this program is as follows:

```
int main(int argc, char* argv[])
{
        if(argc > 1)
                wSlotId= atoi(argv[1]);
if(ERR_SUCCESS != ADAMDrvOpen(&g_lHandle) )
                return ADV_FAIL;
        //Get multi DI channels
        if (ERR_SUCCESS == DIO_GetValues(g_lHandle, wSlotNum, &dwHighValue,
&dwLowValue) )
        {
                //show channel 0 ~ 11
                for (wChannel=0; wChannel<12; wChannel++){
                        if (dwLowValue & (0x0001 << wChannel) )
                                printf("Channel %d = True\n", wChannel); //LED light on
                        else
                                printf("Channel %d = False\n", wChannel); //LED light off
                }
        }
        else
                printf("Fail to get multi DI channels.\n");

        //Set single DO channel
        if (ERR_SUCCESS == DO_SetValue(g_lHandle, wSlotNum, 7, 1) )
                printf("Channel 7 = True\n");
        else
                printf("Fail to set DO value, channel 7\n");
        if (ERR_SUCCESS == DO_SetValue(g_lHandle, wSlotNum, 7, 0) )
                printf("Channel 7 = False\n");
        else
                printf("Fail to clear DO value, channel 7\n");
        //Set multi DO channels
        dwHighValue = 0x0;
        dwLowValue = 0x000000F0; //enable channel 4,5,6,7
        if (ERR_SUCCESS == DO_SetValues(g_lHandle, wSlotNum, dwHighValue, dwLowValue) )
                printf("Set multi DO channels successfully.\n");
        else
                printf("Fail to set multi DO channels.\n");

        dwLowValue = 0x0; //disable all channels
        if (ERR_SUCCESS == DO_SetValues(g_lHandle, wSlotNum, dwHighValue, dwLowValue) )
                printf("Clear multi DO channels successfully.\n");
        else
                printf("Fail to clear multi DO channels.\n");

        ADAMDrvClose(&g_lHandle);
        return 0;
}
```

In general, this program development involves the following steps.

1. Opens a handle that operates the APAXIO driver.
   ADAMDrvOpen (&g_lHandle)
2. Get the all DIO values of the indicated slot.
   DIO_GetValues (g_lHandle, wSlotNum, &dwHighValue, &dwLowValue)
3. Set a single DO value of the indicated slot and channel.
   DO_SetValue (g_lHandle, wSlotNum, 7, 0)
4. Set channel 4,5,6,7 DO values of the indicated slot.
   dwHighValue = 0x0;
   dwLowValue = 0x000000F0;
   DO_SetValues (g_lHandle, wSlotNum, dwHighValue, dwLowValue)
5. Closes the open handle of the APAXIO driver.
   ADAMDrvClose (&g_lHandle)

After you build this example, please upload the executable file "apax_5045_24ch_dio" to the APAX-5522 Embedded Linux. Then run and enter the module slot Id address, for example, my module is 3. The result after execution you can see a similar message as below:

```
# ./apax_5045_24ch_dio 3
Slot ID setting in this sample code is 3.
/*** ADSDIO library version is 101

/*** APAX 5045 module Sample code***/

*** Get multi DI channels  ***
Channel 0 = False
Channel 1 = False
Channel 2 = False
Channel 3 = False
Channel 4 = False
Channel 5 = False
Channel 6 = False
Channel 7 = False
Channel 8 = True
Channel 9 = True
Channel 10 = True
Channel 11 = True

*** Set single DO channel ***
Channel 7 = True
Channel 7 = False

*** Set multi DO channels ***
Set multi DO channels successfully.
Clear multi DO channels successfully.
/***        END         ***/

#
```

4.2.2 AI Module Example

The example of "apax_5017_12ch_ai.c" will illustrate how to control AI with the APAX-5017 module. The APAX-5017 has 12 analog input channels for voltage and current signal measurement. Users can configure each channel with different input type and measurement range. The purpose of this example will get the AI channels status, setting new range for each channel and get the AI channels value. The mainly source code of this program is as follows:

```
int main(int argc, char* argv[])
{
	struct SlotInfo MySlotInfo ={0};
	 if(argc > 1)
		wSlotId= atoi(argv[1]);
	if(ERR_SUCCESS != ADAMDrvOpen(&g_lHandle) )
		return ADV_FAIL;
	 // get the module information of the indicated slot.
	 SYS_GetSlotInfo(g_lHandle, wSlotNum, &MySlotInfo);
	//Get channel status
	if (ERR_SUCCESS == AIO_GetChannelStatus(g_lHandle, wSlotNum, byChannelStatus))
	{
	   for (i=0; i<12; i++){
			switch(byChannelStatus[i]){
				case 0: printf("\"None\"\n");
					break;
				case 1: printf("\"Normal\"\n");
					break;
				case 2: printf("\"Over current\"\n");
					break;
				case 3: printf("\"Under current\"\n");
					break;
				case 4: printf("\"Burn out\"\n");
					break;
				case 5: printf("\"Open loop\"\n");
					break;
				case 6: printf("\"Not ready\"\n");
					break;
				default: printf("\"Unknown\"\n");
			}
		}
	}
	else
		printf("Get Slot %d channel status error \n", wSlotNum);
	//Set inner timeout before config module
	wTimeout = 1000; //ms
	SYS_SetInnerTimeout(g_lHandle, wTimeout);
	//Set new range table for each channel
	wNewRangeTab[0] = ZERO_TO_20_MA;
	wNewRangeTab[1] = NEG_10_TO_10_V;
	wNewRangeTab[2] = _4_TO_20_MA;
	wNewRangeTab[3] = NEG_1_TO_1_V;
	wNewRangeTab[4] = NEG_150_TO_150_MV;
	wNewRangeTab[5] = NEG_5_TO_5_V;
	wNewRangeTab[6] = NEG_150_TO_150_MV;
	wNewRangeTab[7] = NEG_10_TO_10_V;
	wNewRangeTab[8] = NEG_150_TO_150_MV;
	wNewRangeTab[9] = NEG_5_TO_5_V;
	wNewRangeTab[10] = _4_TO_20_MA;
	wNewRangeTab[11] = NEG_10_TO_10_V;
	//Set range follow APAX 5017 range table setting
	AIO_SetRanges(g_lHandle, wSlotNum, 12, wNewRangeTab);
```

```
    //Set channel mask
    printf("\nSet channel mask.\n");
    dwMask = 0xFFFFFFFF; //enable all channel
    AI_SetChannelMask(g_lHandle, wSlotNum, dwMask);

    //Get single AI channel
    wValues[0] = 0x0;
    wChannel = 10;
    lResult = AIO_GetValue(g_lHandle, wSlotNum, wChannel, wValues + wChannel);
    if (ERR_SUCCESS == lResult)
        Scale_RawData(&MySlotInfo, wChannel, wValues);
    else
        printf("Channel %d get value error, error code = %d\n", wChannel, (WORD)lResult);

    //Get multi AI channels
    lResult = AIO_GetValues(g_lHandle, wSlotNum, wValues);
    if (ERR_SUCCESS == lResult){
        for (wChannel=0; wChannel<MySlotInfo.byHwIoTotal_0; wChannel++)
            Scale_RawData(&MySlotInfo, wChannel, wValues);
    }
    else
        printf("Get multi channels value error, error code = %d\n", (WORD)lResult);
    ADAMDrvClose(&g_lHandle);
    return 0;
}
```

In general, this program development involves the following steps.

1. Opens a handle that operates the APAXIO driver.
   ADAMDrvOpen (&g_lHandle)
2. Get the module information of the indicated slot.
   SYS_GetSlotInfo (g_lHandle, wSlotNum, &MySlotInfo);

```
// ============================================
// Slot information definition
// ============================================
struct SlotInfo
{
        BYTE  byUID; //Module slot address: 0x00 ~ 0xEF
        WORDwSeqNo; //Sequence Number
        WORDwPktVer; //Packet Version
        WORDwMsgType; //Message Type
        WORDwFwVerNo; //Firmware Version
        BYTE  byFwVerBld; //Firmware Build Version
        WORDwDevType; //Device Type: 1: DI, 2: DO, 3:AI, 4:AO, 5:Hybrid, 6:Counter
        DWORDdwModName; // Module Name: 0x50170000
        DWORDdwActPeriod; //Active message period
        BYTE  byCOS; //Change of State
        BYTE  byChTotal; //Total number of Channels
        WORDwHwVer; //Hardware Version: AD's firmware version for AI module
        BYTE  byEthSpeed; //0x00: 10/100M, 0x01: Giga, Default: 0x00
        BYTE  byMaxLoadShare; //0x05=> 5% of EthSpeed
//Default: (50/16)3.125%=>3%
        BYTE  byHwIoType_0; //Hardware I/O Type 0
//0:None
//1:DI, 2:DO ,3:AI, 4:AO, 5:Counter, 6:Motion, 7:Relay
        BYTE  byHwIoTotal_0; //Number of Hardware I/O Type 0 channels
        BYTE  byHwIoType_1;
        BYTE  byHwIoTotal_1;
        BYTE  byHwIoType_2;
        BYTE  byHwIoTotal_2;
        BYTE  byHwIoType_3;
        BYTE  byHwIoTotal_3;
        BYTE  byHwIoType_4;
        BYTE  byHwIoTotal_4;
        BYTE  byFunType_0; //Function Type 0
//0:None1:Burn-Out, 2:Digital Filter, 3:Sampling Rate
        DWORDdwFunMask_0;
        DWORDdwFunParam_0;
        BYTE  byFunType_1;
```

```
        DWORDdwFunMask_1;
        DWORDdwFunParam_1;
        BYTE  byFunType_2;
        DWORDdwFunMask_2;
        DWORDdwFunParam_2;
        BYTE  byFunType_3;
        DWORDdwFunMask_3;
        DWORDdwFunParam_3;
        BYTE  byFunType_4;
        DWORDdwFunMask_4;
        DWORDdwFunParam_4;
        WORDwHwIoType_0_Start; // Data register address range start
                               // 0x0000 means Reg. address starts with Reg. 0
        WORDwHwIoType_0_End;
        WORDwHwIoType_1_Start;
        WORDwHwIoType_1_End;
        WORDwHwIoType_2_Start;
        WORDwHwIoType_2_End;
        WORDwHwIoType_3_Start;
        WORDwHwIoType_3_End;
        WORDwHwIoType_4_Start;
        WORDwHwIoType_4_End;
        BYTE  wHwIoType_0_Resolution; //Resolution (Bits per register), 0x10: 16 bits
        BYTE  wHwIoType_1_Resolution;
        BYTE  wHwIoType_2_Resolution;
        BYTE  wHwIoType_3_Resolution;
        BYTE  wHwIoType_4_Resolution;
        BYTE  wHwIoType_0_TotalRange; //Number of supported I/O range
        BYTE  wHwIoType_1_TotalRange;
        BYTE  wHwIoType_2_TotalRange;
        BYTE  wHwIoType_3_TotalRange;
        BYTE  wHwIoType_4_TotalRange;
        WORDwHwIoType_0_Range[36]; // supported I/O range code
        WORDwHwIoType_1_Range[36];
        WORDwHwIoType_2_Range[36];
        WORDwHwIoType_3_Range[36];
        WORDwHwIoType_4_Range[36];
        WORDwChRange[32]; //current range code of the channel
        DWORDdwAlarmMask; //Alarm State Mask
        DWORDdwChMask;  //Channel Enabled Mask
        BYTE  bySizeOfReg; //Size of Register, default:16
        WORDwProtoVer; //Protocol version
        BYTE  byMacAddr[6]; //MAC address
        BYTE  byIpAddr[4]; //IP address
        WORDwSupportFwVer; //Supported Controller's Firmware Version
        BYTE  byExtUsed; //Extension
        WORDwExtLength; //Extension bytes
}__attribute__((packed));
```

3. Get the all AIO channel status of the indicated slot.
   AIO_GetChannelStatus (g_lHandle, wSlotNum, byChannelStatus)
   The value of byChannelStatus indicates:
   0: None
   1: Normal
   2: Over current
   3: Under current
   4: Burn out
   5: Open loop
   6: Not ready
4. Set inner timeout before configure module.
   SYS_SetInnerTimeout (g_lHandle, wTimeout);
   The default inner timeout value is 150msec.
5. Set the channel ranges of the indicated slot.
   AIO_SetRanges (g_lHandle, wSlotNum, TotalChannel, wNewRangeTab);

APAX 5017 Range Table	
Range Id	**Range code**
NEG_150_TO_150_MV (-/+150 mV)	0x0103
NEG_500_TO_500_MV (-/+500 mV)	0x0104
NEG_1_TO_1_V (-/+1 V)	0x0140
NEG_5_TO_5_V (-/+5 V)	0x0142
NEG_10_TO_10_V (-/+10 V)	0x0143
_4_TO_20_MA (4 ~ 20 mA)	0x0180
NEG_20_TO_20_MA (-/+20 mA)	0x0181
ZERO_TO_20_MA (0 ~ 20 mA)	0x0182

6. Set enabled/disabled AI channel mask of the indicated slot.
   dwMask = 0xFFFFFFFF; //enable all channel
   AI_SetChannelMask (g_lHandle, wSlotNum, dwMask)
7. Get a single analog input value of the indicated slot and channel.
   AIO_GetValue (g_lHandle, wSlotNum, wChannel, wValues)
8. Get the all analog input or output values of the indicated slot.
   AIO_GetValues (g_lHandle, wSlotNum, wValues)
9. Raw data scale and transfer.
   Scale_RawData (&MySlotInfo, wChannel, wValues)
10. Closes the open handle of the APAXIO driver.
    ADAMDrvClose (&g_lHandle)

After you build this example, please upload the executable file "apax_5017_12ch_ai" to the APAX-5522 Embedded Linux. Then run and enter the module slot Id address, for example, my module is 2. The result after execution you can see a similar message as below:

```
# ./apax_5017_12ch_ai 2
Slot ID setting in this sample code is 2.
/*** ADSDIO library version is 101
*** Get module ID and current range setting ***
APAX 5017 module
This module has 12 AI channels.
Channel 0 range is 0~20 mA.
Channel 1 range is +/- 10 V.
Channel 2 range is 4~20 mA.
Channel 3 range is +/- 1 V.
Channel 4 range is +/- 150 mV.
Channel 5 range is +/- 5 V.
Channel 6 range is +/- 150 mV.
Channel 7 range is +/- 10 V.
Channel 8 range is +/- 150 mV.
Channel 9 range is +/- 5 V.
Channel 10 range is 4~20 mA.
Channel 11 range is +/- 10 V.

*** Set inner timeout ***
Set inner timeout succeed, timeout = 1000 ms.

*** Set integration time ***
Set integration time succeed.

*** Get channel status ***
Slot 4 channel 0 status is "Normal"
Slot 4 channel 1 status is "Normal"
Slot 4 channel 2 status is "Burn out"
Slot 4 channel 3 status is "Normal"
Slot 4 channel 4 status is "Normal"
Slot 4 channel 5 status is "Normal"
Slot 4 channel 6 status is "Normal"
Slot 4 channel 7 status is "Normal"
Slot 4 channel 8 status is "Normal"
Slot 4 channel 9 status is "Normal"
Slot 4 channel 10 status is "Burn out"
Slot 4 channel 11 status is "Normal"
```

```
*** Set new range table ***
Channel total = 12.
APAX 5017 module
This module has 12 AI channels.
Channel 0 range is 0~20 mA.
Channel 1 range is +/- 10 V.
Channel 2 range is 4~20 mA.
Channel 3 range is +/- 1 V.
Channel 4 range is +/- 150 mV.
Channel 5 range is +/- 5 V.
Channel 6 range is +/- 150 mV.
Channel 7 range is +/- 10 V.
Channel 8 range is +/- 150 mV.
Channel 9 range is +/- 5 V.
Channel 10 range is 4~20 mA.
Channel 11 range is +/- 10 V.

Set channel mask.
Channel 0 = Enable.
Channel 1 = Enable.
Channel 2 = Enable.
Channel 3 = Enable.
Channel 4 = Enable.
Channel 5 = Enable.
Channel 6 = Enable.
Channel 7 = Enable.
Channel 8 = Enable.
Channel 9 = Enable.
Channel 10 = Enable.
Channel 11 = Enable.

*** Set inner timeout ***
Set inner timeout succeed, timeout = 50 ms.

Get single channel value.
Channel 10 raw data is 0xffff, scaled value is 20.0000 mA.

Get multi channels value.
Channel 0 raw data is 0xaadb, scaled value is 13.3483 mA.
Channel 1 raw data is 0x93cd, scaled value is 1.5471 V.
Channel 2 raw data is 0xffff, scaled value is 20.0000 mA.
Channel 3 raw data is 0x8000, scaled value is 0.0000 V.
Channel 4 raw data is 0x8003, scaled value is 0.0160 mV.
Channel 5 raw data is 0x8000, scaled value is 0.0001 V.
Channel 6 raw data is 0x8004, scaled value is 0.0206 mV.
Channel 7 raw data is 0x8000, scaled value is 0.0002 V.
Channel 8 raw data is 0x80f2, scaled value is 1.1101 mV.
Channel 9 raw data is 0x8000, scaled value is 0.0001 V.
Channel 10 raw data is 0xffff, scaled value is 20.0000 mA.
Channel 11 raw data is 0x8000, scaled value is 0.0002 V.
/***        END         ***/

#
```

4.2.3 AO module example

The example of "apax_5028_8ch_ao.c" will illustrate how to control AO with the APAX-5028 module. The APAX-5028 has 8 analog output channels used for voltage and current signal output. Users can configure each channel with different range type and output value. The purpose of this example will also show you how to process the channel calibration in each channel. The mainly source code of this program is as follows:

```
int main(int argc, char* argv[])
{
        if(argc > 1)
                wSlotId= atoi(argv[1]);

if(ERR_SUCCESS != ADAMDrvOpen(&g_lHandle) )
                return ADV_FAIL;

//Set inner timeout before config module
        wTimeout = 1000; //ms
        lResult = SYS_SetInnerTimeout(g_lHandle, wTimeout);
        if (ERR_SUCCESS == lResult)
                printf("Set inner timeout succeed, timeout = %d ms\n\n", wTimeout);
        else
                printf("Get inner timeout error, error code = %d\n\n", (WORD)lResult);

        //get module information
        SYS_GetSlotInfo(g_lHandle, wSlotNum, &MySlotInfo);

#ifdef ENABLE_CALIBRATION_PROCESS
        //take channel 0 as example
    //set channel Range, for example +/- 10V
        wNewRangeTab[0] = NEG_10_TO_10_V;
        lResult = AIO_SetRanges(g_lHandle, wSlotNum, MySlotInfo.byChTotal,  wNewRangeTab);
        if (ERR_SUCCESS == lResult)
                printf("Set new range table succeed.\n");
        else
                printf("Set new range table error, error code = %d\n", (WORD)lResult);
        //set calibration mode
        lResult = AO_SetCalibrationMode(g_lHandle, wSlotNum);
        if (ERR_SUCCESS == lResult)
                printf("Set calibration mode succeed.\n");
        else
                printf("Set calibration mode error, error code = %d\n", (WORD)lResult);
        //set AO module real output (Zero calibration)
        wChannel = 0;
        wValues[0] = 0x0; //set min
        lResult = AO_SetValue(g_lHandle, wSlotNum, wChannel, wValues[0]);
        if (ERR_SUCCESS == lResult)
                ; //use voltmeter to measure if output value equal to -10V or not
        else
                printf("wChannel %d set value error, error code = %d\n", wChannel, (WORD)lResult);
```

```
            // set Zero point
            lResult = AIO_SetZeroCalibration(g_lHandle, wSlotNum, wChannel, 0);
            if (ERR_SUCCESS == lResult)
                printf("Set channel %d Zero calibration succeed.\n", wChannel);
            else
                printf("Set channel %d Zero calibration error, err = %d\n", wChannel, (WORD)lRe-
sult);

//set AO module real output (Span calibration)
            wValues[0] = 0xFFFF; //Set max
            lResult = AO_SetValue(g_lHandle, wSlotNum, wChannel, wValues[0]);
            if (ERR_SUCCESS == lResult)
                ; //use voltmeter to measure if output value equal to 10V or not
            else
                printf("wChannel %d set value error, erro code = %d\n", wChannel, (WORD)lResult);
            // set Span point
            lResult = AIO_SetSpanCalibration(g_lHandle, wSlotNum, wChannel, 0);
            if (ERR_SUCCESS == lResult)
                printf("Set channel %d Span calibration succeed.\n\n", wChannel);
            else
                printf("Set channel %d Span calibration error, err = %d\n\n", wChannel, (WORD)lRe-
sult);
#endif
            //Get channel status
            if (ERR_SUCCESS == AIO_GetChannelStatus(g_lHandle, wSlotNum, byChannelStatus))
            {
               for (i=0; i<12; i++){
                        switch(byChannelStatus[i]){
                                case 0: printf("\"None\"\n");
                                        break;
                                case 1: printf("\"Normal\"\n");
                                        break;
                                case 2: printf("\"Over current\"\n");
                                        break;
                                case 3: printf("\"Under current\"\n");
                                        break;
                                case 4: printf("\"Burn out\"\n");
                                        break;
                                case 5: printf("\"Open loop\"\n");
                                        break;
                                case 6: printf("\"Not ready\"\n");
                                        break;
                                default: printf("\"Unknown\"\n");
                        }
}
            }
            else
                printf("Get Slot %d channel status error \n", wSlotNum);
```

```
        printf("\n*** Set new range table ***\n");
        wNewRangeTab[0] = NEG_10_TO_10_V;
        wNewRangeTab[1] = ZERO_TO_5_V;
        wNewRangeTab[2] = NEG_2P5_TO_2P5_V;
        wNewRangeTab[3] = NEG_2P5_TO_2P5_V;
        wNewRangeTab[4] = _4_TO_20_MA;
        wNewRangeTab[5] = NEG_5_TO_5_V;
        wNewRangeTab[6] = ZERO_TO_20_MA;
        wNewRangeTab[7] = NEG_2P5_TO_2P5_V;
        //Set range follow APAX 5028 range table setting
        AIO_SetRanges(g_lHandle, wSlotNum, MySlotInfo.byChTotal,  wNewRangeTab);

        //Set startup values
        wStartupVal[0] = 0x1000;wStartupVal[1] = 0x2000;wStartupVal[2] = 0x3000;
        wStartupVal[3] = 0x4000;wStartupVal[4] = 0x5000;wStartupVal[5] = 0x6000;
        wStartupVal[6] = 0x7000;wStartupVal[7] = 0x9000;
        AO_SetStartupValues(g_lHandle, wSlotNum, MySlotInfo.byChTotal, wStartupVal);

        //Get startup values
        lResult = AO_GetStartupValues(g_lHandle, wSlotNum, MySlotInfo.byChTotal, wStartupVal);
        if (ERR_SUCCESS == lResult){
                for (wChannel=0; wChannel<MySlotInfo.byHwIoTotal_0; wChannel++)
                        Scale_RawData(&MySlotInfo, wChannel, wStartupVal);
        }
        else
                printf("Get startup values error, error code =%d\n", (WORD)lResult);

        //set single channel value
        wValues[0] = 0x0800;
        wChannel = 0;
        AO_SetValue(g_lHandle, wSlotNum, wChannel, wValues[0]);
        //get single channel value
        wValues[0] = 0x0;
        AIO_GetValue(g_lHandle, wSlotNum, wChannel, wValues);
       Scale_RawData(&MySlotInfo, wChannel, wValues);

        //set multi channels value
        wOutput[0] = 0x0500;wOutput[1] = 0x0500;wOutput[2] = 0x0500;
        wOutput[3] = 0x0500;wOutput[4] = 0x0500;wOutput[5] = 0x0500;
        wOutput[6] = 0x0500;wOutput[7] = 0x0500;
        AO_SetValues(g_lHandle, wSlotNum, dwChannelMask, wOutput);
//get multi channels value
        lResult = AIO_GetValues(g_lHandle, wSlotNum, wValues);
        if (ERR_SUCCESS == lResult){
            for (wChannel=0; wChannel<MySlotInfo.byHwIoTotal_0; wChannel++)
                    Scale_RawData(&MySlotInfo, wChannel, wValues);
        }
        else
                printf("Get startup values error, error code =%d\n", (WORD)lResult);
```

```
            //Buffer and flush values
            wOutput[0] = 0x2500;wOutput[1] = 0x2500;wOutput[2] = 0x2500;
            wOutput[3] = 0x2500;wOutput[4] = 0x2500;wOutput[5] = 0x2500;
            wOutput[6] = 0x2500;wOutput[7] = 0x2500;

            lResult = AO_BufValues(g_lHandle, wSlotNum, dwChannelMask, wOutput);
            if (ERR_SUCCESS == lResult)
                OUT_FlushBufValues(g_lHandle, dwSlotMask);

            ADAMDrvClose(&g_lHandle);
            return 0;
}
```

In general, this program development involves the following steps.

1. Opens a handle that operates the APAXIO driver.
   ADAMDrvOpen (&g_lHandle)
2. Set inner timeout before configure module.
   SYS_SetInnerTimeout (g_lHandle, wTimeout);
   The default inner timeout value is 150msec.
3. Get the module information of the indicated slot.
   SYS_GetSlotInfo (g_lHandle, wSlotNum, &MySlotInfo);
   User can learn the module information like module Id, channel numbers, range table from SlotInfo data structure. (About SlotInfo data structure, please see section 4.2.2 for more detail)
4. User may enable the tag of ENABLE_CALIBRATION_PROCESS to do AO module calibration process. The AO module calibration has following steps to do, and only one channel can be calibrated at once.

1). Must set channel Range table among -10~10V, 5~5V, 0~20mA by calling AO_SetCalibrationMode() function

2). Call AO_SetCalibrationMode() function

3). Zero calibration: set minimum output by calling AO_SetValue() function and set zero point by calling AIO_SetZeroCalibration()

4). Span calibration: set maximum output by calling AO_SetValue() function and set span point by calling AIO_SetSpanCalibration()

5. Get the all AIO channel status of the indicated slot.
   AIO_GetChannelStatus (g_lHandle, wSlotNum, byChannelStatus)
   The value of byChannelStatus indicates:
   - 0: None
   - 1: Normal
   - 2: Over current
   - 3: Under current
   - 4: Burn out
   - 5: Open loop
   - 6: Not ready

6. Set the channel ranges of the indicated slot.
   AIO_SetRanges (g_lHandle, wSlotNum, TotalChannel, wNewRangeTab);

APAX 5028 Range Table	
Range Id	**Range code**
NEG_2P5_TO_2P5_V (-/+2.5 V)	0x0141
NEG_5_TO_5_V (-/+5 V)	0x0142
NEG_10_TO_10_V (-/+10 V)	0x0143
ZERO_TO_2P5_V (0~2.5 V)	0x0146
ZERO_TO_5_V (0~5 V)	0x0147
ZERO_TO_10_V (0 ~ 10 V)	0x0148
_4_TO_20_MA (4 ~ 20 mA)	0x0180
ZERO_TO_20_MA (0 ~ 20 mA)	0x0182

7. Set the AO startup values of the indicated slot.
   AO_SetStartupValues (g_lHandle, wSlotNum, MySlotInfo.byChTotal, wStartupVal)
8. Get the AO startup values of the indicated slot.
   AO_GetStartupValues (g_lHandle, wSlotNum, MySlotInfo.byChTotal, wStartupVal)
   // raw data scale and transfer.
   Scale_RawData (&MySlotInfo, wChannel, wValues)
9. Set a single analog output value of the indicated slot and channel.
   wValues[0] = 0x0800;
   wChannel = 0;
   AO_SetValue (g_lHandle, wSlotNum, wChannel, wValues[0])
10. Get a single analog output value of the indicated slot and channel.
    AIO_GetValue (g_lHandle, wSlotNum, wChannel, wValues)
    // raw data scale and transfer.
    Scale_RawData (&MySlotInfo, wChannel, wValues)
11. Set a multiple analog output values of the indicated slot.
    AO_SetValues (g_lHandle, wSlotNum, dwChannelMask, wOutput)
12. Get the all analog input or output values of the indicated slot.
    AIO_GetValues (g_lHandle, wSlotNum, wValues)
    // raw data scale and transfer.
    Scale_RawData (&MySlotInfo, wChannel, wValues)
13. Buffer the AO values of the indicated slot.
    AO_BufValues (g_lHandle, wSlotNum, dwChannelMask, wOutput)
    Once all slots are buffered, then OUT_FlushBufValues function triggers the synchronized buffer write of all masked slots.
14. Flush the buffered values.
    OUT_FlushBufValues (g_lHandle, dwSlotMask)
    This triggers all buffered values to write simultaneously.
15. Closes the open handle of the APAXIO driver.
    ADAMDrvClose (&g_lHandle)

After you build this example, please upload the executable file "apax_5028_8ch_ao" to the APAX-5522 Embedded Linux. Then run and enter the module slot Id address, for example, my module is 4. The result after execution you can see a similar message as below:

```
# ./apax_5028_8ch_ao 4
/*** ADSDIO library version is 101

Slot ID setting in this sample code is 4.
/*** APAX 5028 module Sample code***/

*** Set inner timeout before config module ***
Set inner timeout succeed, timeout = 1000 ms

APAX 5028 module
This module has 8 AO channels.
Channel 0 range is +/- 10 V.
Channel 1 range is 0~5 V.
Channel 2 range is +/- 2.5 V.
Channel 3 range is +/- 2.5 V.
Channel 4 range is 4~20 mA.
Channel 5 range is +/- 5 V.
Channel 6 range is 0~20 mA.
Channel 7 range is +/- 2.5 V.

*** Get module ID and current range setting ***
Set range succeed

***AO module calibration ***
Set new range table succeed.
Set calibration mode succeed.
Set channel 0 Zero calibration succeed.
Set channel 0 Span calibration succeed.

*** Get channel status ***
Slot 3 channel 0 status is "Normal"
Slot 3 channel 1 status is "Normal"
Slot 3 channel 2 status is "Normal"
Slot 3 channel 3 status is "Normal"
Slot 3 channel 4 status is "Normal"
Slot 3 channel 5 status is "Normal"
Slot 3 channel 6 status is "Normal"
Slot 3 channel 7 status is "Normal"

*** Set new range table ***
APAX 5028 module
This module has 8 AO channels.
Channel 0 range is +/- 10 V.
Channel 1 range is 0~5 V.
Channel 2 range is +/- 2.5 V.
Channel 3 range is +/- 2.5 V.
Channel 4 range is 4~20 mA.
Channel 5 range is +/- 5 V.
Channel 6 range is 0~20 mA.
Channel 7 range is +/- 2.5 V.
```

```
*** Set inner timeout back to default ***
Set inner timeout  succeed, timeout = 150 ms.

*** Get startup values ***
Set Startup Values succeed.
Channel 0 raw data is 0x0, scaled value is -10.0000 V.
Channel 1 raw data is 0x0, scaled value is 0.0000 V.
Channel 2 raw data is 0x0, scaled value is -2.5000 V.
Channel 3 raw data is 0x0, scaled value is -2.5000 V.
Channel 4 raw data is 0x0, scaled value is 4.0000 mA.
Channel 5 raw data is 0x0, scaled value is -5.0000 V.
Channel 6 raw data is 0x0, scaled value is 0.0000 mA.
Channel 7 raw data is 0x0, scaled value is -2.5000 V.

*** Set startup values ***
Channel 0 raw data is 0x1000, scaled value is -8.7500 V.
Channel 1 raw data is 0x2000, scaled value is 0.6250 V.
Channel 2 raw data is 0x3000, scaled value is -1.5625 V.
Channel 3 raw data is 0x4000, scaled value is -1.2500 V.
Channel 4 raw data is 0x5000, scaled value is 9.0001 mA.
Channel 5 raw data is 0x6000, scaled value is -1.2499 V.
Channel 6 raw data is 0x7000, scaled value is 8.7501 mA.
Channel 7 raw data is 0x9000, scaled value is 0.3125 V.

*** Set value (single channel)***
Channel 0 raw data is 0x800, scaled value is -9.3750 V.

*** Set values (multi channels)***

Set all channels output = 0x0500
Channel 0 raw data is 0x500, scaled value is -9.6094 V.
Channel 1 raw data is 0x500, scaled value is 0.0977 V.
Channel 2 raw data is 0x500, scaled value is -2.4023 V.
Channel 3 raw data is 0x500, scaled value is -2.4023 V.
Channel 4 raw data is 0x500, scaled value is 4.3125 mA.
Channel 5 raw data is 0x500, scaled value is -4.8047 V.
Channel 6 raw data is 0x500, scaled value is 0.3906 mA.
Channel 7 raw data is 0x500, scaled value is -2.4023 V.

*** Buffer and flush values ***

*** Set all channels output = 0x2500 ***
Channel 0 raw data is 0x500, scaled value is -9.6094 V.
Channel 1 raw data is 0x500, scaled value is 0.0977 V.
Channel 2 raw data is 0x500, scaled value is -2.4023 V.
Channel 3 raw data is 0x500, scaled value is -2.4023 V.
Channel 4 raw data is 0x500, scaled value is 4.3125 mA.
Channel 5 raw data is 0x500, scaled value is -4.8047 V.
Channel 6 raw data is 0x500, scaled value is 0.3906 mA.
Channel 7 raw data is 0x500, scaled value is -2.4023 V.
/***        END        ***/

#
```

4.2.4 COUNTER Module Example

The example of "apax_5080_8ch_counter.c" will illustrate how to control COUNTER with the APAX-5080 module. The APAX-5080 has 4 or 8 counter channels depends on operating modes. Users can configure each channel with different operating modes. The purpose of this example will get the COUNT channels value, setting new operating mode for each channel and set channel start/stop to count. The mainly source code of this program is as follows:

```
int main(int argc, char* argv[])
{
        if(argc > 1)
                wSlotId= atoi(argv[1]);

if(ERR_SUCCESS != ADAMDrvOpen(&g_lHandle) )
                return ADV_FAIL;

        //get module information
        SYS_GetSlotInfo(g_lHandle, wSlotNum, &MySlotInfo);

        //Get every channel current (counter) value
        if (ERR_SUCCESS == CNT_GetValues(g_lHandle, wSlotNum, dwValues) ){
                for (i=0; i<byCounterChNum; i++)
                        printf("Channel %d counter value = %u\n", i, (WORD)dwValues[i]);
        }
        else
                printf("Call CNT_GetValues() function error.\n");

        //Set new channel range mode
        //important!!!
        // Up/Down mode must set in a pair of channels
        wNewRangeTab[0] = UP_AND_DOWN;
        wNewRangeTab[1] = UP_AND_DOWN;
        wNewRangeTab[2] = UP;
        wNewRangeTab[3] = FREQUENCY;
        // AB phase mode must set in a pair of channels
        wNewRangeTab[4] = AB_1X;
        wNewRangeTab[5] = AB_1X;
        wNewRangeTab[6] = FREQUENCY;
        wNewRangeTab[7] = UP;
        CNT_SetRanges(g_lHandle, wSlotNum, byCounterChNum, wNewRangeTab);

        //Set startup value form 0
        CNT_SetStartupValues(g_lHandle, wSlotNum, byCounterChNum, dwValues);
        //Flash values after change range mode
CNT_ClearValues(g_lHandle, wSlotNum, 0xFFFFFFFF)

        //Set Channel start count
        CNT_SetChannelMask(g_lHandle, wSlotNum, MySlotInfo.dwChMask | 0x000000FF);
```

```
        //Get every channel current (counter) value
if (ERR_SUCCESS == CNT_GetValues(g_lHandle, wSlotNum, dwValues) ){
            for (i=0; i<byCounterChNum; i++)
                printf("Channel %d counter value = %u\n", i, (WORD)dwValues[i]);
        }
        else
                printf("Call CNT_GetValues() function error.\n");

        //Set Channel 2 stop count
        CNT_SetChannelMask(g_lHandle, wSlotNum, MySlotInfo.dwChMask & ~(0x4));//disable Ch
2

        ADAMDrvClose(&g_lHandle);
        return 0;
}
```

In general, this program development involves the following steps.

1. Opens a handle that operates the APAXIO driver.
   ADAMDrvOpen (&g_lHandle)
2. Get the module information of the indicated slot.
   SYS_GetSlotInfo (g_lHandle, wSlotNum, &MySlotInfo);
   User can learn the module information like module Id, channel numbers, operating mode and channel mask from SlotInfo data structure. (About SlotInfo data structure, please see section 4.2.2 for more detail)
3. Get every channel counter value.
   CNT_GetValues (g_lHandle, wSlotNum, dwValues)
4. Set the channel operating mode of the indicated slot.
   CNT_SetRanges (g_lHandle, wSlotNum, byCounterChNum, wNewRangeTab)

APAX 5080 Operating Table	
Range Id	**Range code**
PULSE_DIR(Pulse/Direction mode)	0x01C0
UP_AND_DOWN	0x01C1
UP	0x01C2
FREQUENCY	0x01C3
AB_1X (A/B 1X phase mode)	0x01C4
AB_2X (A/B 2X phase mode)	0x01C58
AB_4X (A/B 4X phase mode)	0x01C6

5. Set the counter startup values of the indicated slot.
   CNT_SetStartupValues (g_lHandle, wSlotNum, byCounterChNum, dwValues)
6. Clear the masked counter values to startup values of the indicated slot.
   CNT_ClearValues (g_lHandle, wSlotNum, 0xFFFFFFFF)
7. Set enabled counter channel mask of the indicated slot.
   CNT_SetChannelMask (g_lHandle, wSlotNum, dwChMask)
8. Closes the open handle of the APAXIO driver.
   ADAMDrvClose (&g_lHandle)

After you build this example, please upload the executable file "apax_5080_8ch_counter" to the APAX-5522 Embedded Linux. Then run and enter the module slot Id address, for example, my module is 0. The result after execution you can see a similar message as below:

```
# ./ apax_5080_8ch_counter 0
Slot ID setting in this sample code is 0.
/*** ADSDIO library version is 101


*** Get module ID and current range setting ***
APAX 5080 module
This module has 4 DI channels.
This module has 4 DO channels.
This module has 8 Counter channels.

Channel 0 is Up mode.
Channel 1 is Up and Down mode.
Channel 2 is Up and Down mode.
Channel 3 is Up mode.
Channel 4 is AB1X mode.
Channel 5 is AB1X mode.
Channel 6 is Frequency mode.
Channel 7 is Up mode.

Get every channel current (counter) value.
Channel 0 counter value = 230
Channel 1 counter value = 0
Channel 2 counter value = 0
Channel 3 counter value = 0
Channel 4 counter value = 0
Channel 5 counter value = 0
Channel 6 counter value = 0
Channel 7 counter value = 75

Set new channel range mode.

Show new channel range mode.
Channel 0 is Up and Down mode.
Channel 1 is Up and Down mode.
Channel 2 is Up mode.
Channel 3 is Frequency mode.
Channel 4 is AB1X mode.
Channel 5 is AB1X mode.
Channel 6 is Frequency mode.
Channel 7 is Up mode.
```

```
Set startup value form 0

Reset all counter

Set Channel start count

Start count....

Now get every channel current (counter) value again.
Channel 0 counter value = 0
Channel 1 counter value = 0
Channel 2 counter value = 62
Channel 3 counter value = 0
Channel 4 counter value = 0
Channel 5 counter value = 0
Channel 6 counter value = 0
Channel 7 counter value = 0

Set Channel 2 stop count
/***        END        ***/

#
```

4.3 Advantech MODBUS library

The libadsmod.so.1.1.4 is a MODBUS library file. It provides programming functions for developers implement MODBUS TCP client and server, MODBUS RTU master and slave applications. Base on this library, user can develop their applications to achieve remote control of the APAX-5000 I/O modules purpose.

The API functions in the library are divided into the following categories:

- MODBUS common functions
- MODBUS-TCP server functions
- MODBUS-TCP client functions
- MODBUS-RTU server functions
- MODBUS-RTU client functions
- MODBUS server internal data functions
- MODBUS server call back functions
- MODBUS client call back functions

4.3.1 MODBUS common functions

4.3.1.1 MOD_Initialize

LONG MOD_Initialize();
■ Purpose: Initialize the MODBUS library resource. ■ Parameters: None. ■ Return ERR_SUCCESS, initialized MODBUS library succeeded.

4.3.1.2 MOD_Terminate

LONG MOD_Terminate();
■ Purpose: Terminate the MODBUS library and release resource. Before calling this function, all MODBUS clients and servers have to be stopped. ■ Parameters: None. ■ Return ERR_SUCCESS, initialized MODBUS library succeeded.

4.3.1.3 MOD_GetVersion

LONG MOD_GetVersion();
■ Purpose: Get the version number of the MODBUS library. ■ Parameters: None. ■ Return The version number. In hex decimal format, for example 0x0114 means version 1.1.4

4.3.2 MODBUS-TCP server functions

4.3.2.1 MOD_StartTcpServer

LONG MOD_StartTcpServer();
■ Purpose: Start the MODBUS-TCP server. ■ Parameters: None. ■ Return ERR_SUCCESS, started MODBUS-TCP server succeeded. ERR_CREATESVR_FAILED, failed to start MODBUS-TCP server. Normally, this caused by the server port (502) has been used.

4.3.2.2 MOD_StopTcpServer

LONG MOD_StopTcpServer();
■ Purpose: Stop the MODBUS-TCP server. ■ Parameters: None. ■ Return ERR_SUCCESS, stopped MODBUS-TCP server succeeded.

4.3.2.3 MOD_SetTcpServerClientIpRestrict

LONG MOD_SetTcpServerClientIpRestrict(bool i_bRestrict);
■ Purpose: Enable/disable the restriction of remote client connection. If this function sets the restriction to true, then only those clients set by MOD_SetTcpServerClientIp will be acceptable by the server. ■ Parameters: i_bRestrict = The Boolean value indicates whether the IP restriction is applied. ■ Return ERR_SUCCESS, set the restriction succeeded.

4.3.2.4 MOD_GetTcpServerClientIp

LONG MOD_GetTcpServerClientIp(int i_iIndex, char *o_szIp);
■ Purpose: Get the acceptable client IP address with indicated index. ■ Parameters: i_iIndex = The index of the acceptable client. This value is ranged from 0 to 7. o_szIp = The IP address of the client in the indicated index. ■ Return ERR_SUCCESS, get the client IP succeeded. ERR_PARAMETER_INVALID, indicates the i_iIndex is out of range.

4.3.2.5 MOD_SetTcpServerClientIp

LONG MOD_SetTcpServerClientIp(int i_iIndex, char *i_szIp);
■ Purpose: Set the acceptable client IP address with indicated index. ■ Parameters: i_iIndex = The index of the acceptable client. This value is ranged from 0 to 7. i_szIp = The IP address of the client. ■ Return ERR_SUCCESS, set the client IP succeeded. ERR_PARAMETER_INVALID, indicates the i_iIndex is out of range or the IP is invalid.

4.3.2.6 MOD_GetTcpServerClientIdle

LONG MOD_GetTcpServerClientIdle(int *o_iIdleSec);
■ Purpose: Get the client transaction idle timeout. ■ Parameters: o_iIdleSec = The transaction idle timeout. ■ Return ERR_SUCCESS, get the transaction idle timeout succeeded.

4.3.2.7 MOD_SetTcpServerClientIdle

LONG MOD_SetTcpServerClientIdle(int *i_iIdleSec);
■ Purpose: Set the client transaction idle timeout. If a connected client has been idled for the setting time, the server will disconnect the client from the server. ■ Parameters: i_iIdleSec = The transaction idle timeout. The value is ranged from 5 to 600 (seconds). ■ Return ERR_SUCCESS, set the transaction idle timeout succeeded. ERR_PARAMETER_INVALID, indicates the i_iIdleSec is out of range.

4.3.3 MODBUS-TCP Client Functions

4.3.3.1 MOD_AddTcpClientConnect

MOD_AddTcpClientConnect(char *i_szServerIp, int i_iScanInterval, int i_iConnectTimeout, int i_iTransactTimeout, OnConnectTcpServerCompletedEvent connEvtHandle, OnDisconnectTcpServerCompletedEvent DisconnEvtHandle, void *i_Param, unsigned long *o_ulClientHandle);
■ Purpose: Add a MODBUS-TCP client into the polling list. This function has to be called before calling MOD_StartTcpClient. ■ Parameters: i_szServerIp = The remote server IP the client will connect to. i_iScanInterval = The scan interval for tag data reading. i_iConnectTimeout = The connection timeout. i_iTransactTimeout = The read/write timeout. connEvtHandle = The pointer of the call back function when connect to server completed. disconnEvtHandle = The pointer of the call back function when disconnect from server completed. i_Param = The pointer of user defined function or object. This pointer will be passed back when callback functions is called. o_ulClientHandle = The MODBUS-TCP client handle. ■ Return ERR_SUCCESS, add the MODBUS-TCP client succeeded. ERR_THREAD_RUNNING, indicates the client thread is running. ERR_MEMALLOC_FAILED, allocate memory failed.

4.3.3.2 MOD_AddTcpClientReadTag

LONG MOD_AddTcpClientReadTag(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned char i_byType, unsigned short i_iStartIndex, unsigned short i_iTotalPoint, OnModbusReadCompletedEvent evtHandle);
■ Purpose: Add a MODBUS-TCP client tag into the data polling list. This function has to be called after calling MOD_AddTcpClientConnect, and before calling MOD_StartTcpClient. ■ Parameters: i_ulClientHandle = The MODBUS-TCP client handle. i_byAddr = The slave (server) address, normally is set to 1. i_byType = The MODBUS reading type. The following values are possible for this member. MODBUS_READCOILSTATUS MODBUS_READINPUTSTATUS MODBUS_READHOLDREG MODBUS_READINPUTREG i_iStartIndex = The start address for reading. i_iTotalPoint = The total point for reading. evtHandle = The pointer of the call back function when reading data completed. ■ Return ERR_SUCCESS, add the MODBUS-TCP client succeeded. ERR_THREAD_RUNNING, indicates the client thread is running. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_PARAMETER_INVALID, indicates the i_byType is invalid. ERR_TCPCLIENT_INVALID, indicates i_ulClientHandle is invalid

4.3.3.3 MOD_StartTcpClient

LONG MOD_StartTcpClient();
■ Purpose: Start the MODBUS-TCP client. After calling with ERR_SUCCESS returned, the client thread will start polling all clients and their reading tags. ■ Parameters: None ■ Return ERR_SUCCESS, started MODBUS-TCP client succeeded. ERR_CREATECLN_FAILED, failed to create the MODBUS-TCP client.

4.3.3.4 MOD_StopTcpClient

LONG MOD_StopTcpClient();
■ Purpose: Stop the MODBUS-TCP client. After calling this function, the client thread will terminate and all polling will stop. ■ Parameters: None ■ Return ERR_SUCCESS, stopped MODBUS-TCP client succeeded.

4.3.3.5 MOD_ReleaseTcpClientResource

LONG MOD_ReleaseTcpClientResource();
■ Purpose: Release all the allocated memory for clients. This function has to be called after calling MOD_StopTcpClient. ■ Parameters: None ■ Return ERR_SUCCESS, released MODBUS-TCP client succeeded. ERR_THREAD_RUNNING, indicates the client thread is running.

4.3.3.6 MOD_ForceTcpClientSingleCoil

LONG MOD_ForceTcpClientSingleCoil(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iIndex, unsigned short i_iData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the single coil output. ■ Parameters: i_ulClientHandle = The MODBUS-TCP client handle. i_byAddr = The slave (server) address, normally is set to 1. i_iIndex = The coil address for writing. i_iData = The data to write. Set this value to 0 means the coil will be set to OFF, otherwise the coil will be set to ON. evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the single coil output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_SOCKET_CLOSED, indicates the client socket is closed. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.3.7 MOD_PresetTcpClientSingleReg

LONG MOD_PresetTcpClientSingleReg(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iIndex, unsigned short i_iData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the single holding register output. ■ Parameters: i_ulClientHandle = The MODBUS-TCP client handle. i_byAddr = The slave (server) address, normally is set to 1. i_iIndex = The holding register address for writing. i_iData = The data to write. evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the single holding register output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_SOCKET_CLOSED, indicates the client socket is closed. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.3.8 MOD_ForceTcpClientMultiCoils

LONG MOD_ForceTcpClientMultiCoils(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iStartIndex, unsigned short i_iTotalPoint, unsigned char i_byTotalByte, unsigned char *i_byData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the multiple coils output. ■ Parameters: i_ulClientHandle = The MODBUS-TCP client handle. i_byAddr = The slave (server) address, normally is set to 1. i_iStartIndex = The starting coil address for writing. i_iTotalPoint = The total of coil points to write. i_byTotalByte = The total byte length of data. i_byData = The byte data of coils to write. For example, if the i_iTotalPoint is 40, then the i_byTotalByte is 5 (8 coils form 1 byte). The buffer order is [Coil 0~7], [Coil 8~15]… evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the multiple coils output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_SOCKET_CLOSED, indicates the client socket is closed. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.3.9 MOD_ForceTcpClientMultiRegs

LONG MOD_ForceTcpClientMultiRegs(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iStartIndex, unsigned short i_iTotalPoint, unsigned char i_byTotalByte, unsigned char *i_byData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the multiple holding registers output. ■ Parameters: i_ulClientHandle = The MODBUS-TCP client handle. i_byAddr = The slave (server) address, normally is set to 1. i_iStartIndex = The starting holding register address for writing. i_iTotalPoint = The total of holding registers to write. i_byTotalByte = The total byte length of data. i_byData = The byte data of holding registers to write. For example, if the i_iTotalPoint is then the i_byTotalByte must be 80 (each register is 2 bytes long). The order of this buffer is as follow: [Reg-0 High byte], [Reg-0 Low byte], [Reg-1 High byte], [Reg-1 Low byte]… evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the multiple holding registers output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_SOCKET_CLOSED, indicates the client socket is closed. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.4 MODBUS-RTU Server Functions

4.3.4.1 MOD_StartRtuServer

LONG MOD_StartRtuServer(int i_iServerIndex, int i_iComIndex, unsigned char i_bySlaveAddr);
■ Purpose: Start the MODBUS-RTU server. ■ Parameters: i_iServerIndex = The server index. This library supports two MODBUS-RTU servers can be created at a single process. The range of this value is from 0 to 1. i_iComIndex = The index of the COM port. The range of this value is from 1 to 255. i_bySlaveAddr = The slave address of the server. The range of this value is from 1 to 247. disconnEvtHandle = The pointer of the call back function when disconnect from server completed. ■ Return ERR_SUCCESS, started MODBUS-RTU server succeeded. ERR_CREATESVR_FAILED, failed to start MODBUS-RTU server. ERR_PARAMETER_INVALID, indicates some of the parameters are out of range.

4.3.4.2 MOD_StopRtuServer

LONG MOD_StopRtuServer(int i_iServerIndex);
■ Purpose: Stop the MODBUS-RTU server. ■ Parameters: i_iServerIndex = The server index. The range of this value is from 0 to 1. ■ Return ERR_SUCCESS, stopped MODBUS-RTU server succeeded. ERR_PARAMETER_INVALID, indicates the i_iServerIndex is out of range.

4.3.4.3 MOD_SetRtuServerComm

LONG MOD_SetRtuServerComm(int i_iServerIndex, long i_lBaudrate, int i_iDataBits, int i_iParity, int i_iStop);

■ Purpose:

Set the MODBUS-RTU server communication parameters.

■ Parameters:

i_iServerIndex = The server index. The range of this value is from 0 to 1.

i_lBaudrate = Specifies the baud rate at which the MODBUS-RTU server operates.

This parameter can be one of the following values:

CBR_1200

CBR_2400

CBR_4800

CBR_9600

CBR_19200

CBR_38400

CBR_57600

CBR_115200

i_iDataBits = Specifies the number of bits in the bytes transmitted and received. The range of this value is from 5 to 8.

This parameter can be one of the following values:

DATA_5

DATA_6

DATA_7

DATA_8

i_iParity = Specifies the parity scheme to be used.

The following values are possible for this member:

NOPARITY

EVENPARITY

ODDPARITY

MARKPARITY

SPACEPARITY

i_iStop = Specifies the number of stop bits to be used.

The following values are possible for this member:

ONESTOPBIT

TWOSTOPBITS

■ Return

ERR_SUCCESS, set the MODBUS-RTU server communication parameters succeeded.

ERR_PARAMETER_INVALID, indicates the i_iServerIndex is out of range.

4.3.4.4 MOD_SetRtuServerTimeout

LONG MOD_SetRtuServerTimeout(int i_iServerIndex, int i_iReadTotalTimeout, int i_iWriteTotalTimeout);
■ Purpose: Set the timeout parameters for all read and write operations on the MODBUS-RTU server. ■ Parameters: i_iServerIndex = The server index. The range of this value is from 0 to 1. i_iReadTotalTimeout = in milliseconds, used to calculate the total timeout period for read operations. i_iWriteTotalTimeout = in milliseconds, used to calculate the total timeout period for write operations. ■ Return ERR_SUCCESS, set the timeout parameters succeeded. ERR_PARAMETER_INVALID, indicates the i_iServerIndex is out of range.

4.3.5 MODBUS-RTU Client Functions

4.3.5.1 MOD_AddRtuClientConnect

LONG MOD_AddRtuClientConnect(int i_iCom, unsigned long i_iBaudrate, unsigned char i_byDataBits, unsigned char i_byParity, unsigned char i_byStopBits, int i_iScanInterval, int i_iTransactTimeout, OnConnectRtuServerCompletedEvent connEvtHandle, OnDisconnectRtuServerCompletedEvent disconnevtHandle, void *i_Param, unsigned long *o_ulClientHandle);
■ Purpose: Add a MODBUS-RTU client into the polling list. This function has to be called before calling MOD_StartRtuClient. ■ Parameters: i_iCom = The COM port number the client will connect to. i_iBaudrate = The baud rate of the COM port. i_byDataBits = The data bits of the COM port. i_byParity = The parity of the COM port. i_byStopBits = The stop bits of the COM port i_iScanInterval = The scan interval for tag data reading. i_iTransactTimeout = The read/write timeout. connEvtHandle = The pointer of the call back function when open the COM port completed. disconnEvtHandle = The pointer of the call back function when close the COM port completed. i_Param = The pointer of user defined function or object. This pointer will be passed back when callback functions is called. o_ulClientHandle = The MODBUS-RTU client handle. ■ Return ERR_SUCCESS, add the MODBUS-RTU client succeeded. ERR_THREAD_RUNNING, indicates the client thread is running. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_RTUCOM_INUSED, indicates the COM port is used by other application.

4.3.5.2 MOD_AddRtuClientReadTag

LONG MOD_AddRtuClientReadTag(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned char i_byType, unsigned short i_iStartIndex, unsigned short i_iTotalPoint, OnModbusReadCompletedEvent evtHandle);
■ Purpose: Add a MODBUS-RTU client tag into the data polling list. This function has to be called after calling MOD_AddRtuClientConnect, and before calling MOD_StartRtuClient. ■ Parameters: i_ulClientHandle = The MODBUS-RTU client handle. i_byAddr = The slave (server) address. i_byType = The MODBUS reading type. The following values are possible for this member. MODBUS_READCOILSTATUS MODBUS_READINPUTSTATUS MODBUS_READHOLDREG MODBUS_READINPUTREG i_iStartIndex = The start address for reading. i_iTotalPoint = The total point for reading. evtHandle = The pointer of the call back function when reading data completed. ■ Return ERR_SUCCESS, add the MODBUS-RTU client succeeded. ERR_THREAD_RUNNING, indicates the client thread is running. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_PARAMETER_INVALID, indicates the i_byType is invalid. ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

4.3.5.3 MOD_StartRtuClient

LONG MOD_StartRtuClient();
■ Purpose: Start the MODBUS-RTU client. After calling with ERR_SUCCESS returned, the client thread will start polling all clients and their reading tags. ■ Parameters: None ■ Return ERR_SUCCESS, started MODBUS-RTU client succeeded. ERR_CREATECLN_FAILED, failed to create the MODBUS-RTU client.

4.3.5.4 MOD_StopRtuClient

LONG MOD_StopRtuClient();
■ Purpose: Stop the MODBUS-RTU client. After calling this function, the client thread will terminate and all polling will stop. ■ Parameters: None ■ Return ERR_SUCCESS, stopped MODBUS-RTU client succeeded.

4.3.5.5 MOD_ReleaseRtuClientResource

LONG MOD_ReleaseRtuClientResource();
■ Purpose: Release all the allocated memory for clients. This function has to be called after calling MOD_StopRtuClient. ■ Parameters: None ■ Return ERR_SUCCESS, released MODBUS-RTU client succeeded. ERR_THREAD_RUNNING, indicates the client thread is running.

4.3.5.6 MOD_ForceRtuClientSingleCoil

LONG MOD_ForceRtuClientSingleCoil(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iIndex, unsigned short i_iData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the single coil output. ■ Parameters: i_ulClientHandle = The MODBUS-RTU client handle. i_byAddr = The slave (server) address. i_iIndex = The coil address for writing. i_iData = The data to write. Set this value to 0 means the coil will be set to OFF, otherwise the coil will be set to ON. evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the single coil output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.5.7 MOD_PresetRtuClientSingleReg

LONG MOD_PresetRtuClientSingleReg(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iIndex, unsigned short i_iData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the single holding register output. ■ Parameters: i_ulClientHandle = The MODBUS-RTU client handle. i_byAddr = The slave (server) address. i_iIndex = The holding register address for writing. i_iData = The data to write. evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the single holding register output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.5.8 MOD_ForceRtuClientMultiCoils

LONG MOD_ForceRtuClientMultiCoils(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iStartIndex, unsigned short i_iTotalPoint, unsigned char i_byTotalByte, unsigned char *i_byData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the multiple coils output. ■ Parameters: i_ulClientHandle = The MODBUS-RTU client handle. i_byAddr = The slave (server) address. i_iStartIndex = The starting coil address for writing. i_iTotalPoint = The total of coil points to write. i_byTotalByte = The total byte length of data. i_byData = The byte data of coils to write. For example, if the i_iTotalPoint is 40, then the i_byTotalByte must be 5 (8 coils form 1 byte). The order of this buffer is as follow: [Coil 0~7], [Coil 8~15]… evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the multiple coils output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.5.9 MOD_ForceRtuClientMultiRegs

LONG MOD_ForceRtuClientMultiRegs(unsigned long i_ulClientHandle, unsigned char i_byAddr, unsigned short i_iStartIndex, unsigned short i_iTotalPoint, unsigned char i_byTotalByte, unsigned char *i_byData, OnModbusWriteCompletedEvent evtHandle);
■ Purpose: Set the multiple holding registers output. ■ Parameters: i_ulClientHandle = The MODBUS-RTU client handle. i_byAddr = The slave (server) address. i_iStartIndex = The starting holding register address for writing. i_iTotalPoint = The total of holding registers to write. i_byTotalByte = The total byte length of data. i_byData = The byte data of holding registers to write. For example, if the i_iTotalPoint is then the i_byTotalByte must be 80 (each register is 2 bytes long). The order of this buffer is as follow: [Reg-0 High byte], [Reg-0 Low byte], [Reg-1 High byte], [Reg-1 Low byte]… evtHandle = The pointer of the call back function when writing data completed. ■ Return ERR_SUCCESS, Set the multiple holding registers output succeeded. ERR_THREAD_STOP, indicates the client thread is stopped. ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid. ERR_MEMALLOC_FAILED, allocate memory failed. ERR_THREAD_BUSY, indicates the client is busy for previous writing.

4.3.6 MODBUS Server Internal Data Functions

4.3.6.1 . MOD_GetServerCoil

LONG MOD_GetServerCoil(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);
■ Purpose: Get the coil status data of the MODBUS server. In a process, the library has a single copy of coil status data that can be shared among threads. ■ Parameters: i_iStart = The start index of the coil status data. The range of this value is from 1 to 65535. i_iLen = The length of the coil status data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536. o_byBuf = The buffer pointer where to store the read coil status data. o_iRetLen = The length of the returned coil status data in bytes. ■ Return ERR_SUCCESS, get the coil status data succeeded. ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.3.6.2 MOD_GetServerInput

LONG MOD_GetServerInput(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);
■ Purpose: Get the input status data of the MODBUS server. In a process, the library has a single copy of input status data that can be shared among threads. ■ Parameters: i_iStart = The start index of the input status data. The range of this value is from 1 to 65535. i_iLen = The length of the input status data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536. o_byBuf = The buffer pointer where to store the read input status data. o_iRetLen = The length of the returned input status data in bytes. ■ Return ERR_SUCCESS, get the input status data succeeded. ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.3.6.3 MOD_GetServerInputReg

LONG MOD_GetServerInputReg(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);
■ Purpose: Get the input register data of the MODBUS server. In a process, the library has a single copy of input register data that can be shared among threads. ■ Parameters: i_iStart = The start index of the input register data. The range of this value is from 1 to 65535. i_iLen = The length of the input register data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536. o_byBuf = The buffer pointer where to store the read input register data. o_iRetLen = The length of the returned input register data in bytes. ■ Return ERR_SUCCESS, get the input status data succeeded. ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.3.6.4 MOD_GetServerHoldReg

LONG MOD_GetServerHoldReg(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);
■ Purpose: Get the holding register data of the MODBUS server. In a process, the library has a single copy of holding register data that can be shared among threads. ■ Parameters: i_iStart = The range of this value is from 1 to 65535. i_iLen = The length of the holding register data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536. o_byBuf = The buffer pointer where to store the read holding register data. o_iRetLen = The length of the returned holding register data in bytes. ■ Return ERR_SUCCESS, get the holding register data succeeded. ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.3.6.5 MOD_SetServerCoil

LONG MOD_SetServerCoil(int i_iStart, int i_iLen, unsigned char *i_byBuf);
■ Purpose: Set the coil status data of the MODBUS server. In a process, the library has a single copy of coil status data that can be shared among threads. ■ Parameters: i_iStart = The start index of the coil status data. The range of this value is from 1 to 65535. i_iLen = The length of the coil status data to be set. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536. i_byBuf = The buffer pointer where to hold the coil status data to be set. For example, if the i_iLen is 40, then the length of this buffer must be 5 bytes long. ■ Return ERR_SUCCESS, set the coil status data succeeded. ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the i_byBuf is NULL.

4.3.6.6 MOD_SetServerHoldReg

LONG MOD_SetServerHoldReg(int i_iStart, int i_iLen, unsigned char *i_byBuf);
■ Purpose: Set the holding register data of the MODBUS server. In a process, the library has a single copy of holding register data that can be shared among threads. ■ Parameters: i_iStart = The start index of the holding register data. The range of this value is from 1 to 65535. i_iLen = The length of the holding register data to be set. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536. i_byBuf = The buffer pointer where to hold the holding register data to be set. For example, if the i_iLen is 40, then the length of this buffer must be 80 bytes long (each register is 2 bytes long) The order of the bytes is as follow: [Reg-0 High byte], [Reg-0 Low byte], [Reg-1 High byte], [Reg-1 Low byte]… ■ Return ERR_SUCCESS, set the holding register data succeeded. ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the i_byBuf is NULL.

4.3.7 MODBUS Server Call Back Functions

4.3.7.1 OnCoilChangedEvent

typedef void (*OnCoilChangedEvent)(int i_iStart, int i_iLen);
■ Purpose: The prototype of call back function when the coil changed event occurred. ■ Parameters: i_iStart = The start index of the coil status data that has been changed. i_iLen = The length of the coil status data that has been changed. ■ Return None.

4.3.7.2 OnHoldRegChangedEvent

typedef void (*OnHoldRegChangedEvent)(int i_iStart, int i_iLen);
■ Purpose: The prototype of the call back function when the holding register changed event occurred. ■ Parameters: i_iStart = The start index of the holding register data that has been changed. i_iLen = The length of the holding register data that has been changed. ■ Return None.

4.3.7.3 OnRemoteTcpClientConnectEvent

typedef void (*OnRemoteTcpClientConnectEvent)(char *i_szIp);

- Purpose:

The prototype of the call back function when a remote client connects to the server event occurred.

- Parameters:

i_szIp = The IP address of the remote client.

- Return

None.

4.3.7.4 OnRemoteTcpClientDisconnectEvent

typedef void (*OnRemoteTcpClientDisconnectEvent)(char *i_szIp);

- Purpose:

The prototype of the call back function when a remote client disconnects from the server event occurred.

- Parameters:

i_szIp = The IP address of the remote client.

- Return

None.

4.3.7.5 MOD_SetServerCoilChangedEventHandler

LONG MOD_SetServerCoilChangedEventHandler(OnCoilChangedEvent i_evtHandle);

- Purpose:

Set the coil status data changed event handler of the MODBUS server. Any clients try to write the coil status data to the server will cause the call back function to be called.

- Parameters:

i_evtHandle = The pointer of the call back function.

- Return

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

4.3.7.6 MOD_SetServerHoldRegChangedEventHandler

LONG ADS_API MOD_SetServerHoldRegChangedEventHandler(OnHoldRegChangedEvent i_evtHandle);
■ Purpose: Set the holding register data changed event handler of the MODBUS server. Any clients try to write the holding register data to the server will cause the call back function to be called. ■ Parameters: i_evtHandle = The pointer of the call back function. ■ Return ERR_SUCCESS, get the holding register data succeeded. ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

4.3.7.7 MOD_SetTcpServerClientConnectEventHandler

LONG ADS_API MOD_SetTcpServerClientConnectEventHandler(OnRemoteTcpClientConnectEvent i_evtHandle);
■ Purpose: Set the MODBUS-TCP client connect to the server event handler. Any clients connect to the server will cause the call back function to be called. ■ Parameters: i_evtHandle = The pointer of the call back function. ■ Return ERR_SUCCESS, set client connect to server event handler succeeded. ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

4.3.7.8 MOD_SetTcpServerClientDisconnectEventHandler

LONG ADS_API MOD_SetTcpServerClientDisconnectEventHandler(OnRemoteTcpClientDisconnectEvent i_evtHandle);
■ Purpose: Set the MODBUS-TCP client disconnect from the server event handler. Any clients disconnect from the server will cause the call back function to be called. ■ Parameters: i_evtHandle = The pointer of the call back function. ■ Return ERR_SUCCESS, set client disconnect from server event handler succeeded. ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

4.3.8 MODBUS Client Call Back Functions

4.3.8.1 OnConnectTcpServerCompletedEvent

typedef void (*OnConnectTcpServerCompletedEvent)(long lResult, char *i_szIp);
■ Purpose: The prototype of call back function when the connection to the MODBUS-TCP server completed event occurred. ■ Parameters: lResult = The result of the connection. i_szIp = The IP of the MODBUS-TCP server the client connects to. ■ Return None.

4.3.8.2 OnDisconnectTcpServerCompletedEvent

typedef void (*OnDisconnectTcpServerCompletedEvent)(long lResult, char *i_szIp);
■ Purpose: The prototype of call back function when the connection to the MODBUS-TCP server closed event occurred. ■ Parameters: lResult = The result of the disconnection. i_szIp = The IP of the MODBUS-TCP server the client disconnect from. ■ Return None.

4.3.8.3 OnConnectRtuServerCompletedEvent

typedef void (*OnConnectRtuServerCompletedEvent)(long lResult, int i_iCom);
■ Purpose: The prototype of call back function when the open COM port completed event occurred. ■ Parameters: lResult = The result of the open process. i_iCom = The COM port the client opened. ■ Return None.

4.3.8.4 OnDisconnectRtuServerCompletedEvent

typedef void (*OnDisconnectRtuServerCompletedEvent)(long lResult, int i_iCom);
■ Purpose: The prototype of call back function when the close COM port completed event occurred. ■ Parameters: lResult = The result of the close process. i_iCom = The COM port the client closed. ■ Return None.

4.3.8.5 OnModbusReadCompletedEvent

typedef void (*OnModbusReadCompletedEvent)(long lResult, unsigned char *i_byData, int i_iLen);
■ Purpose: The prototype of call back function when the read data completed event occurred. ■ Parameters: lResult = The result of the reading. i_byData = The pointer of buffer that hold the read data. i_iLen = The length of the data read. ■ Return None.

4.3.8.6 OnModbusWriteCompletedEvent

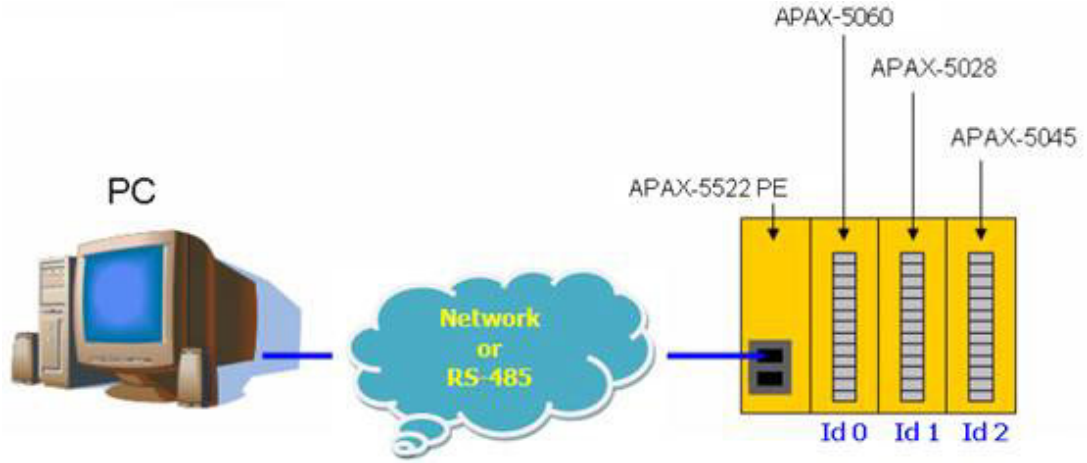
typedef void (*OnModbusWriteCompletedEvent)(long lResult);
■ Purpose: The prototype of call back function when the write data completed event occurred. ■ Parameters: lResult = The result of the writing. ■ Return None.

4.3.9 Error Code Definition

Error ID	Error code
MODERR_SUCCESS	0
MODERR_WSASTART_FAILED	801
MODERR_WSACLEAN_FAILED	802
MODERR_CREATESOCK_FAILED	803
MODERR_CREATESVR_FAILED	804
MODERR_PARAMETER_INVALID	805
MODERR_WAIT_TIMEOUT	806
MODERR_CONNECT_FAILED	807
MODERR_CLOSESOCKET_FAILED	808
MODERR_MEMALLOC_FAILED	809
MODERR_CREATECLN_FAILED	810
MODERR_TCPCLIENT_INVALID	811
MODERR_TCPSEND_FAILED	812
MODERR_TCPRECV_FAILED	813
MODERR_SOCKET_CLOSED	814
MODERR_THREAD_RUNNING	815
MODERR_THREAD_STOP	816
MODERR_THREAD_BUSY	817
MODERR_RTUCLIENT_INVALID	818
MODERR_RTUCOM_INUSED	819
MODERR_RTUCRC_INVALID	820
MODERR_SETPRIO_FAILED	821
MODERR_RTUSEND_FAILED	822
MODERR_RTURECV_FAILED	823
MODERR_MODEXCEPT_01 /* ILLEGAL FUNCTION */	901
MODERR_MODEXCEPT_02 /* ILLEGAL DATA ADDRESS */	902
MODERR_MODEXCEPT_03 /* ILLEGAL DATA VALUE */	903
MODERR_MODEXCEPT_04 /* SLAVE DEVICE FAILURE */	904
MODERR_MODEXCEPT_05 /* ACKNOWLEDGE */	905
MODERR_MODEXCEPT_06 /* SLAVE DEVICE BUSY */	906
MODERR_MODEXCEPT_07 /* NEGATIVE ACKNOWLEDGE */	907
MODERR_MODEXCEPT_08 /* MEMORY PARITY ERROR */	908
MODERR_MODEXCEPT_99 /* Invalid packet */	999

4.4 Advantech MODBUS Library Demo

After decompress the SDK file, user can find the files for the example code under the path "Apax5522-SDK/AdvModbus". The sample programs for the MODBUS library provide programming examples for MODBUS TCP client and server, MODBUS RTU client and server. The system hardware architecture for the demo can be shown as figure. One computer connected via Network or RS-485 interface to the APAX-5522 Linux and APAX-5000 I/O modules. This section will introduce major parts of sample programs using MODBUS functions to perform write or read action to the APAX-5522 Linux.



4.4.1 MODBUS Address Mapping

The most important configuration for APAX-5522 Linux is to define the MODBUS address mapping. After you have completed the address mapping, you can simply get data from or write data to the APAX-5522 Linux through the defined address.

Users can use MOD_SetServerCoil and MOD_SetServerHoldReg function to define your own MODBUS address mapping in your server program. The MODBUS address mapping for the demo can be shown as following. We define the length of 64 for each DIO modules (0x address) and the length of 32 for each AIO modules (4x address).

DIO module(0x address): MOD_SetServerCoil(SlotId*64, 64, byData);					
AIO module(4x address): MOD_SetServerHoldReg(SlotId*32, 32, byData);					
Module ID	Module Name	Address Type	Start Address	Length	Modbus Address
0	APAX-5060 (8ch Relay)	0x	1	64	00001 ~ 00064
1	APAX-5028 (8ch AO)	4x	33	32	40033 ~ 40064
2	APAX-5045 (12ch DI,12ch DO)	0x	129	64	00129 ~ 00192

You may refer to Appendix C: MODBUS mapping table example to know the results after the server configuration of your APAX-5000 I/O modules.

4.4.2 MODBUS TCP Server/client Example

The examples of "exModbusTcpServer.cpp" and "exModbusTcpClient.cpp" will illustrate how to control the APAX-5000 I/O modules via network by MODBUS TCP protocol. The mainly source code of these programs is as follows:

- MODBUS TCP server mainly source code

```
int main(int argc, char* argv[])
{
        char szIp[32] = "172.19.1.71"; //allows the client IP
        int iIdleSecs = 10; // client idle timeout 10 seconds
        MOD_Initialize();
        ADAMDrvOpen(&m_hdlAdsdio);
        for (int iSlot=0; iSlot<32; iSlot++)
                SYS_GetModuleID(m_hdlAdsdio, iSlot, &m_dwSlotID[iSlot]);
        // setup configuration before starting the server
        MOD_SetTcpServerClientIpRestrict(true);
        // assign the client IP that is legal to access the server (set to index 0)
        MOD_SetTcpServerClientIp(0, szIp);
        // set the client idle time limit, if the client become silent more than the time limit
        // the connection will be closed
        MOD_SetTcpServerClientIdle(iIdleSecs);
        // setup callback functions
  MOD_SetTcpServerClientConnectEventHandler(&RemoteTcpClientConnectEventHandler);
        MOD_SetTcpServerClientDisconnectEventHandler(&RemoteTcpClientDisconnectEventHandler);
        MOD_SetServerCoilChangedEventHandler(&ClientWriteCoilHandler);
        MOD_SetServerHoldRegChangedEventHandler(&ClientWriteHoldRegHandler);
        // start the MODBUS-TCP server
        MOD_StartTcpServer()
        while (kbhit() == 0)
        {
                UpdateSlot();
                usleep(1000000); // 1sec
        }
// stop the MODBUS-TCP server
        MOD_StopTcpServer();

        ADAMDrvClose(&m_hdlAdsdio);
        MOD_Terminate();
        return 0;
}
```

```
//Read and update module value to data buffer
void UpdateSlot()
{
        for (iSlot=0; iSlot<32; iSlot++){
// DIO module
        if (m_dwSlotID[iSlot] >= 0x50400000 && m_dwSlotID[iSlot] < 0x50700000){
        memset(byData, 0, 64);
        DIO_GetValues(m_hdlAdsdio, iSlot, (unsigned long*)&byData[4], (unsigned
long*)&byData[0]);
        MOD_SetServerCoil(iSlot*64, 64, byData);
        }
// AIO module
        else if (m_dwSlotID[iSlot] < 0x50300000){
        memset(byData, 0, 64);
        AIO_GetValues(m_hdlAdsdio, iSlot, (WORD*)byData);
        wTmp = (WORD*)byData;
        for (iWord = 0; iWord < 32; iWord++)
           wTmp[iWord] = BASE_htons(wTmp[iWord]);
        MOD_SetServerHoldReg(iSlot*32, 32, byData);
        }
// CNT module
        else if (m_dwSlotID[iSlot] >= 0x50800000){
        memset(byData, 0, 64);
CNT_GetValues(m_hdlAdsdio, iSlot, (DWORD*)byData);
        dwTmp = (DWORD*)byData;
        for (iDWord = 0; iDWord < 16; iDWord++)
                dwTmp[iDWord] = BASE_htonl(dwTmp[iDWord]);
        MOD_SetServerHoldReg(iSlot*32, 32, byData);
        }
}
//Callback functions from server if client has a write request.
void ClientWriteCoilHandler(int iStart, int iLen)
{
        int iRetLen;
        unsigned char byData[256] = {0};
        MOD_GetServerCoil(iStart, iLen, (unsigned char*)byData, &iRetLen);
        TOOLSetCoil(iStart, iLen, (unsigned char*)byData);
}
void ClientWriteHoldRegHandler(int iStart, int iLen)
{
        int iRetLen;
        unsigned char byData[256] = {0};
MOD_GetServerHoldReg(iStart, iLen, (unsigned char*)byData, &iRetLen);
        TOOLSetHoldReg(iStart, iLen, (unsigned char*)byData);
}
```

```
// ClientWriteCoilHandler
bool TOOLSetCoil(int iStart, int iLen, unsigned char *byBuf)
{
	int iIdx, iDataLen, iByteLen, iByteStart, iBitOffset, iTailBits, iRetLen;
	unsigned char byData[256] = {0};
	unsigned char byTmp;

	if (iStart < 0 || iStart >= 2048)
		return false;
	if (iStart + iLen > 2048)
		iLen = 2048 - iStart;

	iByteLen = (iLen - 1)/8 + 1;// the total byte length need
	iByteStart = iStart / 8;// the coil starting byte index
	iBitOffset = iStart % 8;// the first bit offset
	// move to temp buffer
	if (iBitOffset == 0){
	iDataLen = (iLen - 1)/8 + 1;// the total byte length that byData need
	iTailBits = (iLen - 1)%8 + 1;// the bits total of last byte
	for (iIdx = 0; iIdx < iDataLen; iIdx++)
		byData[iIdx] = *(byBuf + iIdx);
	}
	else	{
	iDataLen = (iLen + iBitOffset - 1)/8 + 1;// the total byte length that byData need
	iTailBits = (iLen + iBitOffset - 1)%8 + 1;// the bits total of last byte
	// first byte, keep the low bits from server coils, add the low bit of byBuf to the high of the
byData
	MOD_GetServerCoil(iByteStart * 8, 8, &byTmp, &iRetLen);
	byData[0] = (byTmp & m_LoOnMask[iBitOffset]) + (*byBuf << iBitOffset);
	for (iIdx = 1; iIdx < iDataLen; iIdx++)
	{
		byData[iIdx] = (*(byBuf + iIdx - 1) >> (8 - iBitOffset));
		if (iIdx < iByteLen) // any more byte can be used from the
			byData[iIdx] += (*(byBuf + iIdx) << iBitOffset);
	}
	}

	if (iTailBits < 8) { // mask the tail bits of the last byte with original coils
	MOD_GetServerCoil((iByteStart + iDataLen - 1) * 8, 8, &byTmp, &iRetLen);
	byData[iDataLen - 1] = (byTmp & m_HiOnMask[iTailBits])
+ (byData[iDataLen - 1] & m_LoOnMask[iTailBits]);
	}

	return APAXWriteCoil(iByteStart, iDataLen, byData);
}
```

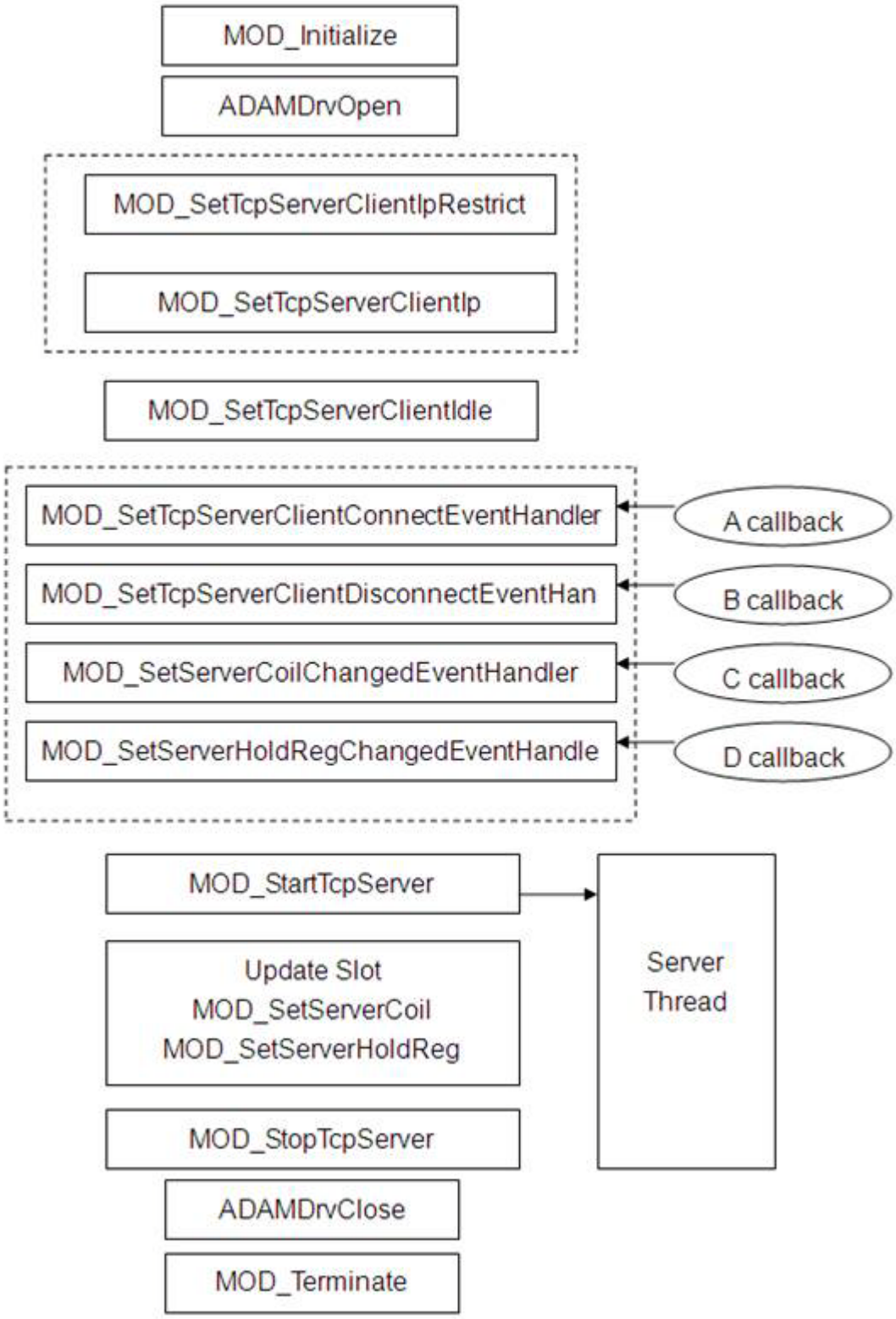
```
//perform writes action to APAX-5000 I/O modules.
bool APAXWriteCoil(int iStart, int iLen, unsigned char *byBuf)
{
	int iSlotStart, iSlotEnd, iIdx, iByteStart, iByteEnd, iByteIdx, iByteBase, iOffset, iRetLen;
	unsigned char byData[8], byTmp;
	DWORD dwSlotMask, dwHigh, dwLow;

	iSlotStart = iStart/8; // MAX 8 bytes (64 bits) for each slot
	iSlotEnd = (iStart + iLen - 1)/8;
	dwSlotMask = 0;
	iByteBase = 0;
	for (iIdx = iSlotStart; iIdx <= iSlotEnd; iIdx++)
	{
		if (iIdx == iSlotStart)
			iByteStart = iStart % 8;
		else
			iByteStart = 0;
		if (iIdx == iSlotEnd)
			iByteEnd = (iStart + iLen - 1) % 8;
		else
			iByteEnd = 7;
		if (m_dwSlotID[iIdx] == A5045_ID || // just list these few now, may add more later
			m_dwSlotID[iIdx] == A5046_ID ||
			m_dwSlotID[iIdx] == A5060_ID)
		{
			dwSlotMask |= (0x0001 << iIdx);
			// try to form data for current slot
			iOffset = 0;
			for (iByteIdx = 0; iByteIdx < 8; iByteIdx++)
			{
				if (iByteIdx < iByteStart || iByteIdx > iByteEnd)
				{
					MOD_GetServerCoil((iIdx * 8 + iByteIdx) * 8, 8, &byTmp,
&iRetLen);
					byData[iByteIdx] = byTmp;
				}
				else
				{
					byData[iByteIdx] = *(byBuf + iByteBase + iOffset);
					iOffset++;
				}
			}
			memcpy(&dwHigh, (byData + 4), sizeof(DWORD));
			memcpy(&dwLow, byData, sizeof(DWORD));
```

```
                        // start
                        if (m_dwSlotID[iIdx] == A5045_ID) // previous 12 bits are DI, so shift 12
                        {
                                dwLow = (dwLow >> 12) + (dwHigh << 20);
                                dwHigh = (dwHigh >> 12);
                        }
                        // end
                        if (MODERR_SUCCESS != DO_BufValues(m_hdlAdsdio, iIdx, dwHigh,
dwLow))
                                return false;
                }
                iByteBase += (iByteEnd - iByteStart + 1); // increase the byte base
        }
        if (MODERR_SUCCESS != OUT_FlushBufValues(m_hdlAdsdio, dwSlotMask))
                return false;

        return true;
}

//ClientWriteHoldRegHandler
bool TOOLSetHoldReg(int iStart, int iLen, unsigned char *byBuf)
{
        if (iStart < 0 && iStart >= 1024)
                return false;
        if (iStart + iLen > 1024)
                iLen = 1024 - iStart;
        return APAXWriteHoldReg(iStart, iLen, byBuf);
}

//perform writes action to APAX-5000 I/O modules.
bool APAXWriteHoldReg(int iStart, int iLen, unsigned char *byBuf)
{
        int iSlotStart, iSlotEnd, iIdx, iCh, iChStart, iChEnd, iByteBase, iMaxCh, iOffset;
        WORD wData[32];
        DWORD dwSlotMask, dwChMask;

        iSlotStart = iStart/32; // MAX 32 channel for each slot
        iSlotEnd = (iStart + iLen - 1)/32;
        dwSlotMask = 0;
        iByteBase = 0;
        for (iIdx = iSlotStart; iIdx <= iSlotEnd; iIdx++)
        {
                if (iIdx == iSlotStart)
                        iChStart = iStart % 32;
                else
```

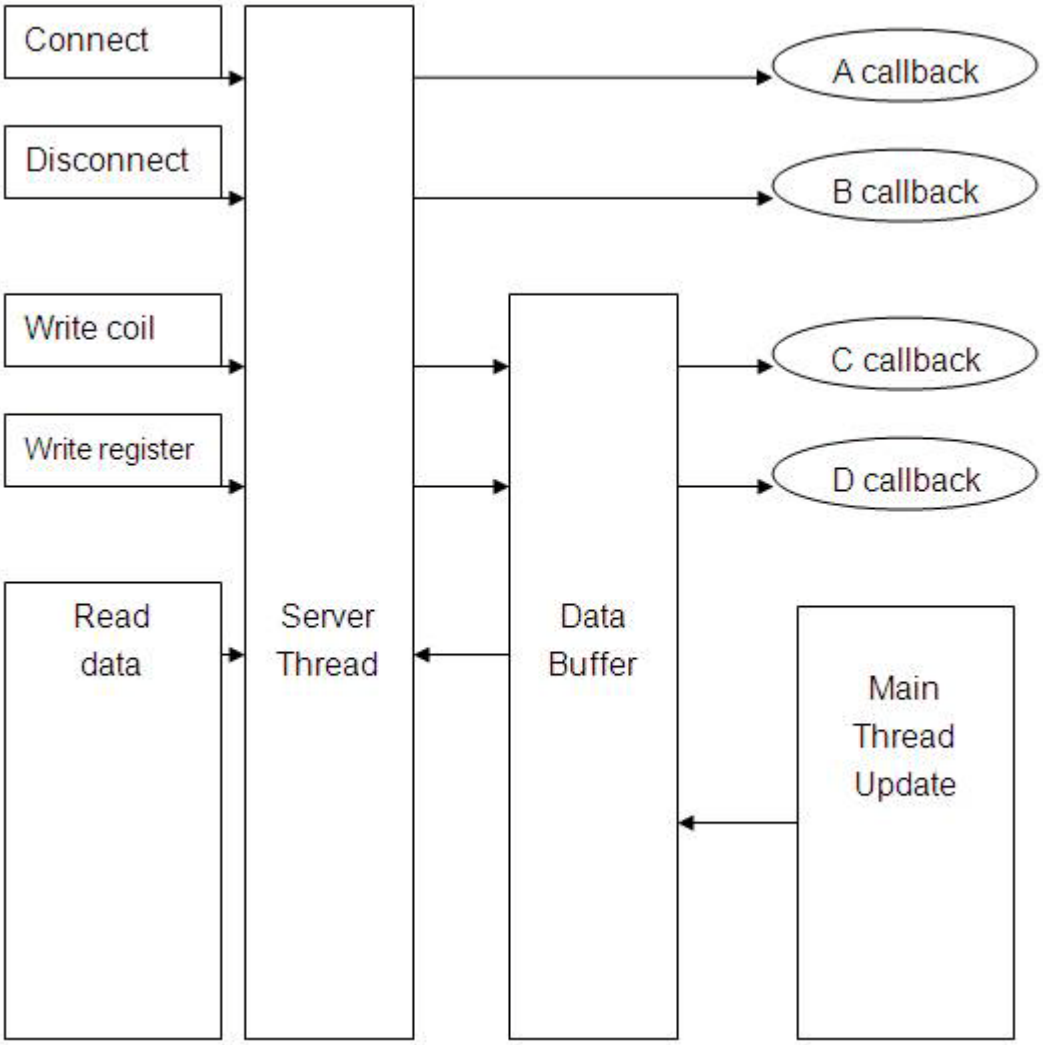
```
                        iChStart = 0;
                if (iIdx == iSlotEnd)
                        iChEnd = (iStart + iLen - 1) % 32;
                else
                        iChEnd = 31;
                if (m_dwSlotID[iIdx] == A5028_ID) // just list these few now, may add more later
                {
                        iMaxCh = 8; // APAX-5028 has 8 AO
                        dwSlotMask |= (0x0001 << iIdx);
                        // try to form data
                        memset(&wData, 0, sizeof(WORD)*32);
                        dwChMask = 0;
                        iOffset = 0;
                        for (iCh = 0; iCh < iMaxCh; iCh++)
                        {
                                if (iCh < iChStart || iCh > iChEnd)
                                        wData[iCh] = 0;
                                else
                                {
                                        dwChMask |= (0x0001 << iCh);
                                        wData[iCh] = *(byBuf + iByteBase + iOffset);
                                        wData[iCh] = (wData[iCh] << 8) + *(byBuf + iByteBase + iOffset
+ 1);
                                        iOffset += 2;
                                }
                        }
                        if (MODERR_SUCCESS !=
AO_BufValues(m_hdlAdsdio, iIdx, dwChMask, (WORD*)wData))
                                return false;
                }
                iByteBase += (iChEnd - iChStart + 1)*2; // increase the byte base, 1 ch = 2 bytes
        }
        if (MODERR_SUCCESS != OUT_FlushBufValues(m_hdlAdsdio, dwSlotMask))
                return false;
        return true;
}
```

In general, The MODBUS-TCP server calling and working flow can be shown as below:

- Calling flow



- Working flow


Connect
Disconnect
Write coil
Write register
Read data
Server Thread
Data Buffer
A callback
B callback
C callback
D callback
Main Thread Update

■ MODBUS TCP client mainly source code

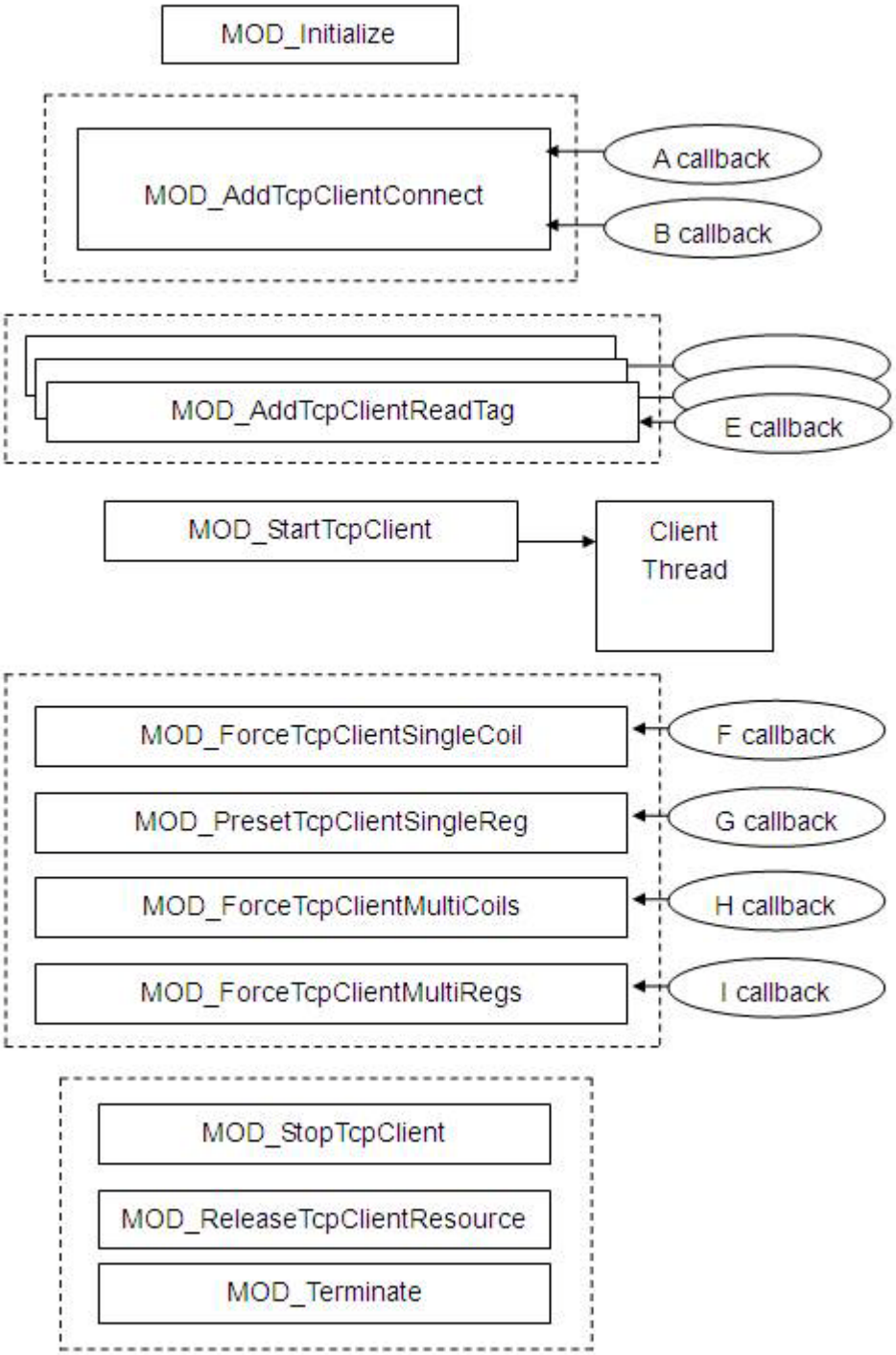
```
int main(int argc, char* argv[])
{
      unsigned long ulClientHandle;
      unsigned char byData[8] = {0};
      LONG lRet;

      MOD_Initialize();

     MOD_AddTcpClientConnect((char *)"172.19.1.71",  // remote server IP
                                    100                      // scan interval (ms)
                                    3000,                   // connection timeout (ms)
                                    100,                     // transaction timeout (ms)
                                     ConnectTcpServerCompletedEventHandler,   //callback function
                                     DisconnectTcpServerCompletedEventHandler, //callback function
                                       NULL,             // user defined
                                       &ulClientHandle);     // MODBUS-TCP client handle.
  // Slot Id 0: APAX-5060
     MOD_AddTcpClientReadTag(ulClientHandle,
                                       1,                       // The address normally set to 1.
                                      MODBUS_READCOILSTATUS, // MODBUS reading type.
                                       1,                       // start address for reading
                                       12,                      // total point for reading.
                                       ClientReadCoil_1_64_Handler); //callback function
   // Slot Id 1: APAX-5028
     MOD_AddTcpClientReadTag(ulClientHandle,
                                       1,
                                       MODBUS_READHOLDREG,
                                       33,                   // start address for reading
                                       8,
                                       ClientReadReg_33_64_Handler);
     // Slot Id 2: APAX-5045
     MOD_AddTcpClientReadTag(ulClientHandle,
                                       1,
                                       MODBUS_READCOILSTATUS,
                                       141,          // start address for reading, read DO only
                                       12,
                                       ClientReadCoil_129_192_Handler);
     //start the MODBUS-TCP client
     MOD_StartTcpClient();

     printf("MODBUS-TCP client started...\n");
            printf("Press any key to exit...\n");
```

```
        while (kbhit() == 0)
        {
                if( m_writeEnable == TRUE && m_isConnected == TRUE)
                {
                        m_writeEnable = FALSE;
                   // perform write action to the APAX-5060 12ch Relay
                        lRet = MOD_ForceTcpClientMultiCoils(ulClientHandle,
                                                                1,      // The address normally set to 1.
                                                                1,      // start address for writing
                                                                12,      // total point for writing.
                                                                2,      // total byte length of data
                                                                byData,  // The byte data of coils to write
                                                                 ModbusWriteCompletedEventHandler);    //callback
function
                        if (0 == lRet)
                     {
                                byData[0]++;
                                byData[1]++;
                                usleep(500000);
                        }
                    }
         }
         //stop the MODBUS-TCP client
         MOD_StopTcpClient();
         MOD_ReleaseTcpClientResource();
         MOD_Terminate();
         return 0;
}
void ConnectTcpServerCompletedEventHandler(long lResult, char *i_szIp, void *i_Param)
{
        if (lResult == MODERR_SUCCESS)
                m_isConnected = TRUE;
        else
                m_isConnected = FALSE;
}

void DisconnectTcpServerCompletedEventHandler(long lResult, char *i_szIp, void *i_Param)
{
        m_isConnected = FALSE;
}

void ModbusWriteCompletedEventHandler(long lResult, void *i_Param)
{
        m_writeEnable = TRUE;
}
```

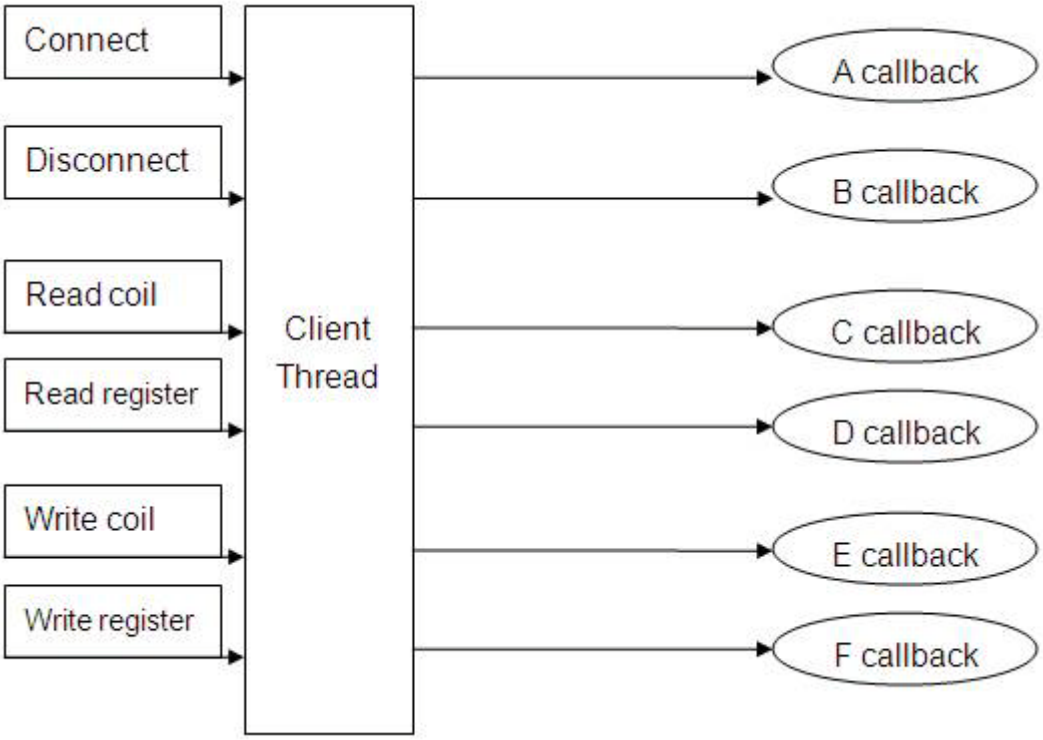
```
//Id 0: APAX-5060 (read 8ch Relay value)
// The order of this value is: [CH 0~7], [CH 8~15]…
void ClientReadCoil_1_64_Handler(long lResult, unsigned char *i_byData, int i_iLen, void *i_Param)
{
	if (lResult == MODERR_SUCCESS)
	{
		printf("ReadCoil[1~64] = ");
		for (int i=0; i<i_iLen; i++)
			printf("%02X ", i_byData[i]);
		printf("\n");
	}
}

//Id 1: APAX-5028 (read 8ch AO value)
// The order of this value is:
// [CH0 High byte], [CH0 Low byte], [CH1 High byte], [CH1 Low byte]…
void ClientReadReg_33_64_Handler(long lResult, unsigned char *i_byData, int i_iLen, void *i_Param)
{
	if (lResult == MODERR_SUCCESS)
	{
		printf("ReadReg[33~64] = ");
		for (int i=0; i<i_iLen; i++)
			printf("%02X ", i_byData[i]);
		printf("\n");
	}
}

//Id 2: APAX-5045 (read 12ch DO value only)
// The order of this value is: [CH 0~7], [CH 8~15]…
void ClientReadCoil_129_192_Handler(long lResult, unsigned char *i_byData, int i_iLen, void
*i_Param)
{
	if (lResult == MODERR_SUCCESS)
	{
		printf("ReadCoil[129~192] = ");
		for (int i=0; i<i_iLen; i++)
			printf("%02X ", i_byData[i]);
		printf("\n");
	}
}
```

In general, The MODBUS-TCP client calling and working flow can be shown as below:

- Calling flow



- Working flow



When you have built the MODBUS-TCP server and client example, please start the MODBUS-TCP server program on the APAX-5522 Embedded Linux and then execute the MODBUS-TCP client program on your computer or you may use other software which supports Modbus TCP client (like Modscan or HMI/SCADA software) to perform the read or write operation to the APAX-5522 Embedded Linux. For my example, you can see the message as following:

- The MODBUS-TCP server execution screen:

```
# ./exModbusTcpServer
ADAMDrvOpen: m_hdlAdsdio = 3
SYS_GetModuleID: [Slot 0]: 0x50600000
SYS_GetModuleID: [Slot 1]: 0x50280000
SYS_GetModuleID: [Slot 2]: 0x50450000
MODBUS-TCP server started...
Press any key to exit...          <-   waiting for client connections

Client connects from '172.19.1.71'  <-  client connected
```

- The MODBUS-TCP client execution screen:

```
MODBUS-TCP client started...
Press any key to exit...
Connect to '172.19.1.71' result = 0    <-  connect to server success
ReadCoil[1~64] = 00 00    <-  Id 0: APAX-5060 12ch DI value
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58    <-  Id 1: APAX-5028 8ch AO
value
ReadCoil[129~192] = F5 03    <-  Id 2: APAX-5045 12ch DO value
….
….
MOD_ForceTcpClientMultiCoils start (1) ? perform write action to Id 0: APAX-5060
Coil write result = 0
ReadCoil[1~64] = 00 00
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F5 03
…..
…..
MOD_ForceTcpClientMultiCoils start (2)
Coil write result = 0
ReadCoil[1~64] = 01 01    <-  Id 0: APAX-5060 12ch DI value has changed
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F5 03
ReadCoil[1~64] = 01 01
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F5 03
…..
…..
MOD_ForceTcpClientMultiCoils start (3)
Coil write result = 0
ReadCoil[1~64] = 02 02    <-  Id 0: APAX-5060 12ch DI value has changed
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F5 03
ReadCoil[1~64] = 02 02
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F5 03
…..
…..
Disconnect from '172.19.1.71' result = 0    <-  press any key to exit
Loop count = 6
Read failed = 0
Write failed = 0
Connect count = 0
Disconnect count = 1
```

4.4.3 MODBUS RTU Server/Client Example

The examples of "exModbusRtuServer.cpp" and "exModbusRtuClient.cpp" will illustrate how to control the APAX-5000 I/O modules via RS-485 by MODBUS RTU protocol. The mainly source code of these programs is as follows:

- MODBUS RTU server mainly source code

```
int main(int argc, char* argv[])
{
        int iServerIndex = 0;
        int iComPort = 1;
        int iSlaveAddr = 1;

        MOD_Initialize();

        ADAMDrvOpen(&m_hdlAdsdio);
        for (int iSlot=0; iSlot<32; iSlot++)
                SYS_GetModuleID(m_hdlAdsdio, iSlot, &m_dwSlotID[iSlot]);

        // setup configuration before starting the server
        MOD_SetRtuServerComm(iServerIndex, CBR_9600, DATA_8, NOPARITY, ONESTOPBIT);
        MOD_SetRtuServerTimeout(iServerIndex, 100, 100);
        MOD_SetServerCoilChangedEventHandler(&ClientWriteCoilHandler);
        MOD_SetServerHoldRegChangedEventHandler(&ClientWriteHoldRegHandler);

        // start the MODBUS-RTU server
        MOD_StartRtuServer(iServerIndex, iComPort, iSlaveAddr);

        printf("MODBUS-RTU server started...\n");
        printf("Press any key to exit...\n");

        while (kbhit() == 0)
        {
                UpdateSlot();
                usleep(1000000);
        }
        // stop the MODBUS-RTU server
        MOD_StopRtuServer(iServerIndex);
        ADAMDrvClose(&m_hdlAdsdio);
        MOD_Terminate();
        return 0;
}
```

```
//Read and update module value to data buffer
void UpdateSlot()
{
	for (iSlot=0; iSlot<32; iSlot++){
	// DIO module
	if (m_dwSlotID[iSlot] >= 0x50400000 && m_dwSlotID[iSlot] < 0x50700000){
	memset(byData, 0, 64);
	DIO_GetValues(m_hdlAdsdio, iSlot, (unsigned long*)&byData[4], (unsigned long*)&byData[0]);
	MOD_SetServerCoil(iSlot*64, 64, byData);
	}
	// AIO module
	else if (m_dwSlotID[iSlot] < 0x50300000){
	memset(byData, 0, 64);
	AIO_GetValues(m_hdlAdsdio, iSlot, (WORD*)byData);
	wTmp = (WORD*)byData;
	for (iWord = 0; iWord < 32; iWord++)
	   wTmp[iWord] = BASE_htons(wTmp[iWord]);
	MOD_SetServerHoldReg(iSlot*32, 32, byData);
	}
	// CNT module
	else if (m_dwSlotID[iSlot] >= 0x50800000){
	memset(byData, 0, 64);
	CNT_GetValues(m_hdlAdsdio, iSlot, (DWORD*)byData);
	dwTmp = (DWORD*)byData;
	for (iDWord = 0; iDWord < 16; iDWord++)
	      dwTmp[iDWord] = BASE_htonl(dwTmp[iDWord]);
	MOD_SetServerHoldReg(iSlot*32, 32, byData);
	}
}
//Callback functions from server if client has a write request.
void ClientWriteCoilHandler(int iStart, int iLen)
{
	int iRetLen;
	unsigned char byData[256] = {0};
	MOD_GetServerCoil(iStart, iLen, (unsigned char*)byData, &iRetLen);
	TOOLSetCoil(iStart, iLen, (unsigned char*)byData);
}
void ClientWriteHoldRegHandler(int iStart, int iLen)
{
	int iRetLen;
	unsigned char byData[256] = {0};
	MOD_GetServerHoldReg(iStart, iLen, (unsigned char*)byData, &iRetLen);
	TOOLSetHoldReg(iStart, iLen, (unsigned char*)byData);
}
```

```
// ClientWriteCoilHandler
bool TOOLSetCoil(int iStart, int iLen, unsigned char *byBuf)
{
	int iIdx, iDataLen, iByteLen, iByteStart, iBitOffset, iTailBits, iRetLen;
	unsigned char byData[256] = {0};
	unsigned char byTmp;

	if (iStart < 0 || iStart >= 2048)
		return false;
	if (iStart + iLen > 2048)
		iLen = 2048 - iStart;

	iByteLen = (iLen - 1)/8 + 1;// the total byte length need
	iByteStart = iStart / 8;// the coil starting byte index
	iBitOffset = iStart % 8;// the first bit offset
	// move to temp buffer
	if (iBitOffset == 0){
	iDataLen = (iLen - 1)/8 + 1;// the total byte length that byData need
	iTailBits = (iLen - 1)%8 + 1;// the bits total of last byte
	for (iIdx = 0; iIdx < iDataLen; iIdx++)
		byData[iIdx] = *(byBuf + iIdx);
	}
	else {
	iDataLen = (iLen + iBitOffset - 1)/8 + 1;// the total byte length that byData need
	iTailBits = (iLen + iBitOffset - 1)%8 + 1;// the bits total of last byte
	// first byte, keep the low bits from server coils, add the low bit of byBuf to the high of the byData
	MOD_GetServerCoil(iByteStart * 8, 8, &byTmp, &iRetLen);
	byData[0] = (byTmp & m_LoOnMask[iBitOffset]) + (*byBuf << iBitOffset);
	for (iIdx = 1; iIdx < iDataLen; iIdx++)
	{
		byData[iIdx] = (*(byBuf + iIdx - 1) >> (8 - iBitOffset));
		if (iIdx < iByteLen) // any more byte can be used from the
			byData[iIdx] += (*(byBuf + iIdx) << iBitOffset);
	}
	}

	if (iTailBits < 8) { // mask the tail bits of the last byte with original coils
	MOD_GetServerCoil((iByteStart + iDataLen - 1) * 8, 8, &byTmp, &iRetLen);
	byData[iDataLen - 1] = (byTmp & m_HiOnMask[iTailBits])
				+ (byData[iDataLen - 1] & m_LoOnMask[iTailBits]);
	}

	return APAXWriteCoil(iByteStart, iDataLen, byData);
}
```

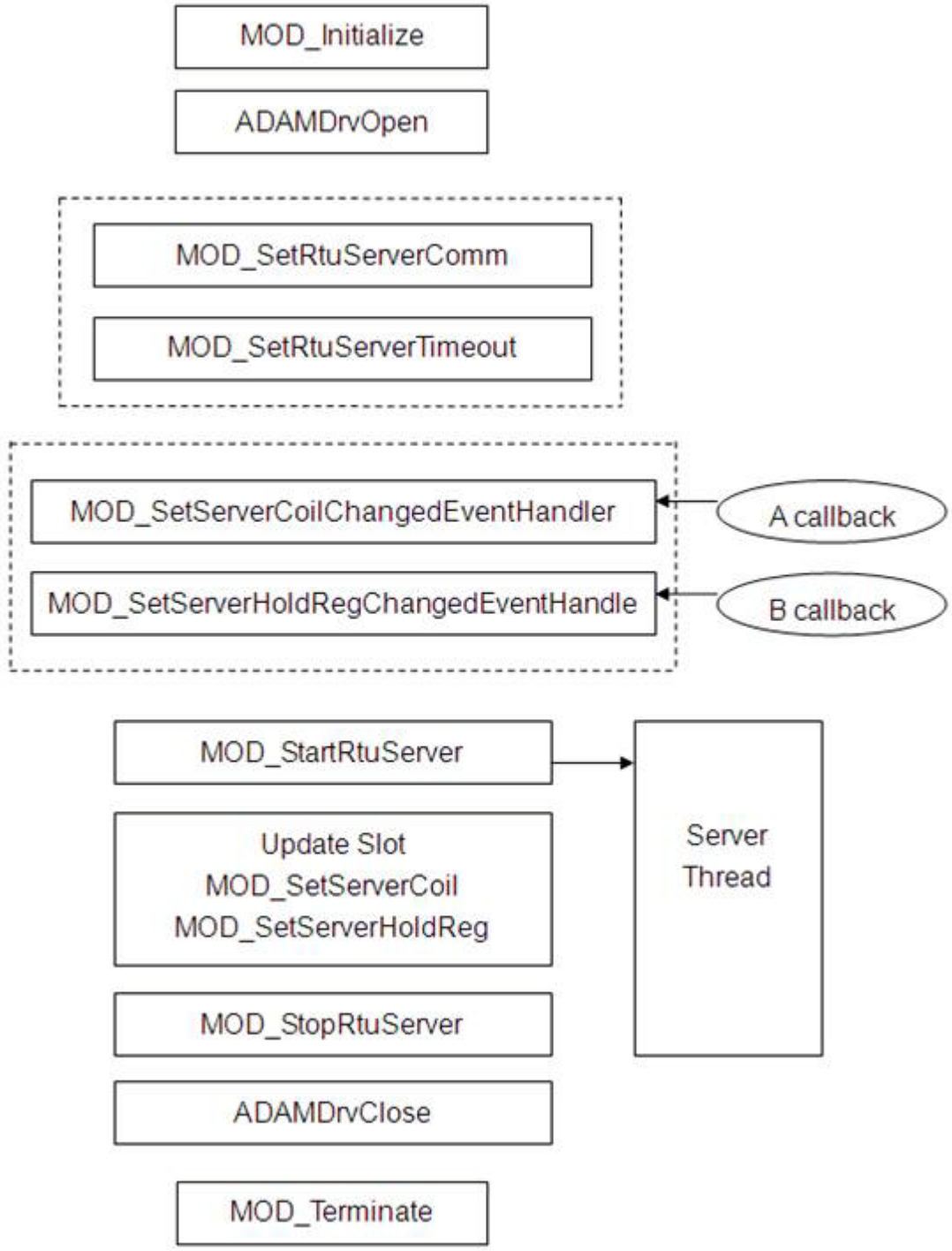
```
//perform writes action to APAX-5000 I/O modules.
bool APAXWriteCoil(int iStart, int iLen, unsigned char *byBuf)
{
	int iSlotStart, iSlotEnd, iIdx, iByteStart, iByteEnd, iByteIdx, iByteBase, iOffset, iRetLen;
	unsigned char byData[8], byTmp;
	DWORD dwSlotMask, dwHigh, dwLow;

	iSlotStart = iStart/8; // MAX 8 bytes (64 bits) for each slot
	iSlotEnd = (iStart + iLen - 1)/8;
	dwSlotMask = 0;
	iByteBase = 0;
	for (iIdx = iSlotStart; iIdx <= iSlotEnd; iIdx++)
	{
		if (iIdx == iSlotStart)
			iByteStart = iStart % 8;
		else
			iByteStart = 0;
		if (iIdx == iSlotEnd)
			iByteEnd = (iStart + iLen - 1) % 8;
		else
			iByteEnd = 7;
		if (m_dwSlotID[iIdx] == A5045_ID || // just list these few now, may add more later
			m_dwSlotID[iIdx] == A5046_ID ||
			m_dwSlotID[iIdx] == A5060_ID)
		{
			dwSlotMask |= (0x0001 << iIdx);
			// try to form data for current slot
			iOffset = 0;
			for (iByteIdx = 0; iByteIdx < 8; iByteIdx++)
			{
				if (iByteIdx < iByteStart || iByteIdx > iByteEnd)
				{
					MOD_GetServerCoil((iIdx * 8 + iByteIdx) * 8, 8, &byTmp, &iRetLen);
					byData[iByteIdx] = byTmp;
				}
				else
				{
					byData[iByteIdx] = *(byBuf + iByteBase + iOffset);
					iOffset++;
				}
			}
			memcpy(&dwHigh, (byData + 4), sizeof(DWORD));
				memcpy(&dwLow, byData, sizeof(DWORD));
```

```
                    // start
                    if (m_dwSlotID[iIdx] == A5045_ID) // previous 12 bits are DI, so shift 12
                    {
                        dwLow = (dwLow >> 12) + (dwHigh << 20);
                        dwHigh = (dwHigh >> 12);
                    }
                    // end
                    if (MODERR_SUCCESS != DO_BufValues(m_hdlAdsdio, iIdx, dwHigh, dwLow))
                        return false;
                }
                iByteBase += (iByteEnd - iByteStart + 1); // increase the byte base
        }
        if (MODERR_SUCCESS != OUT_FlushBufValues(m_hdlAdsdio, dwSlotMask))
            return false;

        return true;
}

//ClientWriteHoldRegHandler
bool TOOLSetHoldReg(int iStart, int iLen, unsigned char *byBuf)
{
        if (iStart < 0 && iStart >= 1024)
            return false;
        if (iStart + iLen > 1024)
            iLen = 1024 - iStart;
        return APAXWriteHoldReg(iStart, iLen, byBuf);
}

//perform writes action to APAX-5000 I/O modules.
bool APAXWriteHoldReg(int iStart, int iLen, unsigned char *byBuf)
{
        int iSlotStart, iSlotEnd, iIdx, iCh, iChStart, iChEnd, iByteBase, iMaxCh, iOffset;
        WORD wData[32];
        DWORD dwSlotMask, dwChMask;

        iSlotStart = iStart/32; // MAX 32 channel for each slot
        iSlotEnd = (iStart + iLen - 1)/32;
        dwSlotMask = 0;
        iByteBase = 0;
        for (iIdx = iSlotStart; iIdx <= iSlotEnd; iIdx++)
        {
            if (iIdx == iSlotStart)
                iChStart = iStart % 32;
                  else
```

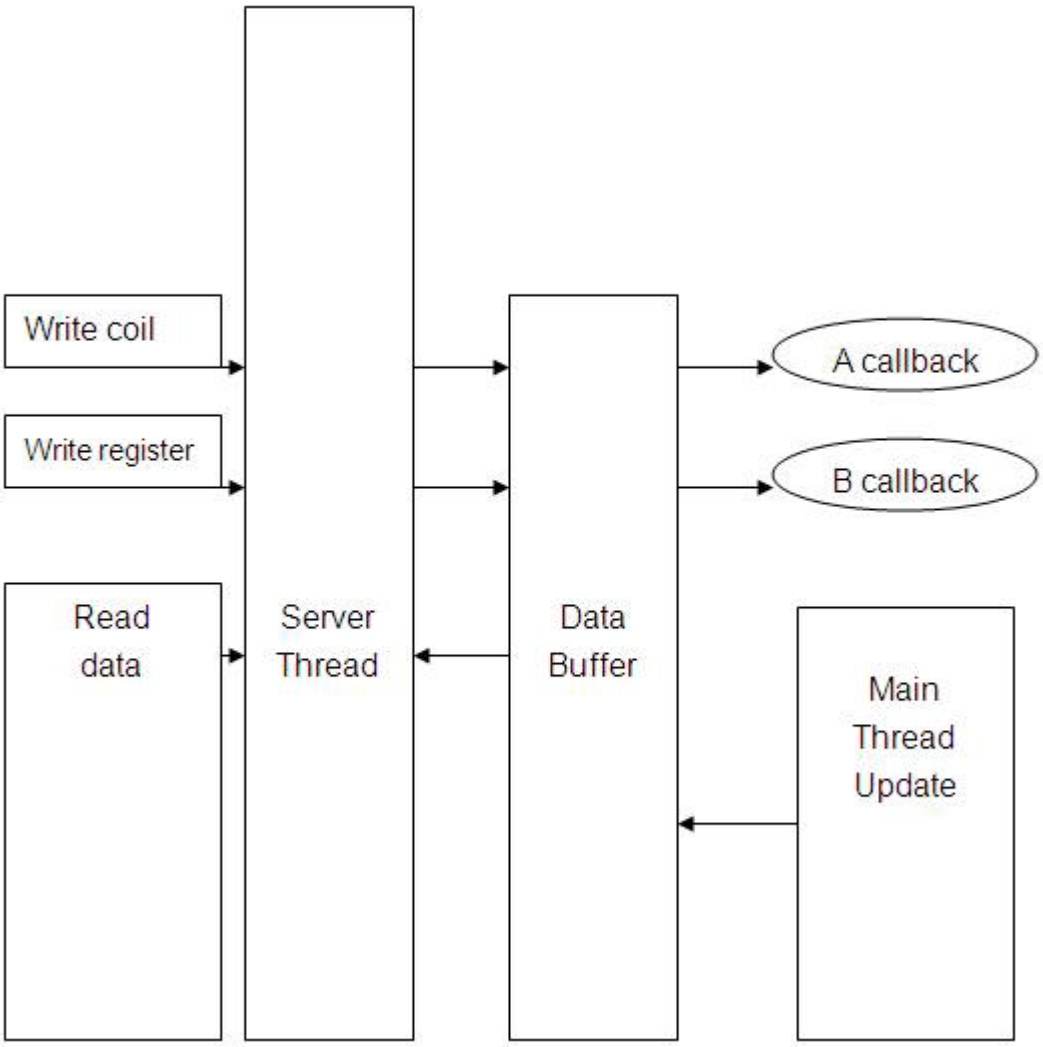
```
                iChStart = 0;
            if (iIdx == iSlotEnd)
                iChEnd = (iStart + iLen - 1) % 32;
            else
                iChEnd = 31;
            if (m_dwSlotID[iIdx] == A5028_ID) // just list these few now, may add more later
            {
                iMaxCh = 8; // APAX-5028 has 8 AO
                dwSlotMask |= (0x0001 << iIdx);
                // try to form data
                memset(&wData, 0, sizeof(WORD)*32);
                dwChMask = 0;
                iOffset = 0;
                for (iCh = 0; iCh < iMaxCh; iCh++)
                {
                    if (iCh < iChStart || iCh > iChEnd)
                        wData[iCh] = 0;
                    else
                    {
                        dwChMask |= (0x0001 << iCh);
                        wData[iCh] = *(byBuf + iByteBase + iOffset);
                        wData[iCh] = (wData[iCh] << 8) + *(byBuf + iByteBase + iOffset + 1);
                        iOffset += 2;
                    }
                }
                if (MODERR_SUCCESS !=
                    AO_BufValues(m_hdlAdsdio, iIdx, dwChMask, (WORD*)wData))
                    return false;
            }
            iByteBase += (iChEnd - iChStart + 1)*2; // increase the byte base, 1 ch = 2 bytes
        }
        if (MODERR_SUCCESS != OUT_FlushBufValues(m_hdlAdsdio, dwSlotMask))
            return false;
        return true;
}
```

In general, The MODBUS-RTU server calling and working flow can be shown as below:

- Calling flow



- Working flow



■ MODBUS RTU client mainly source code

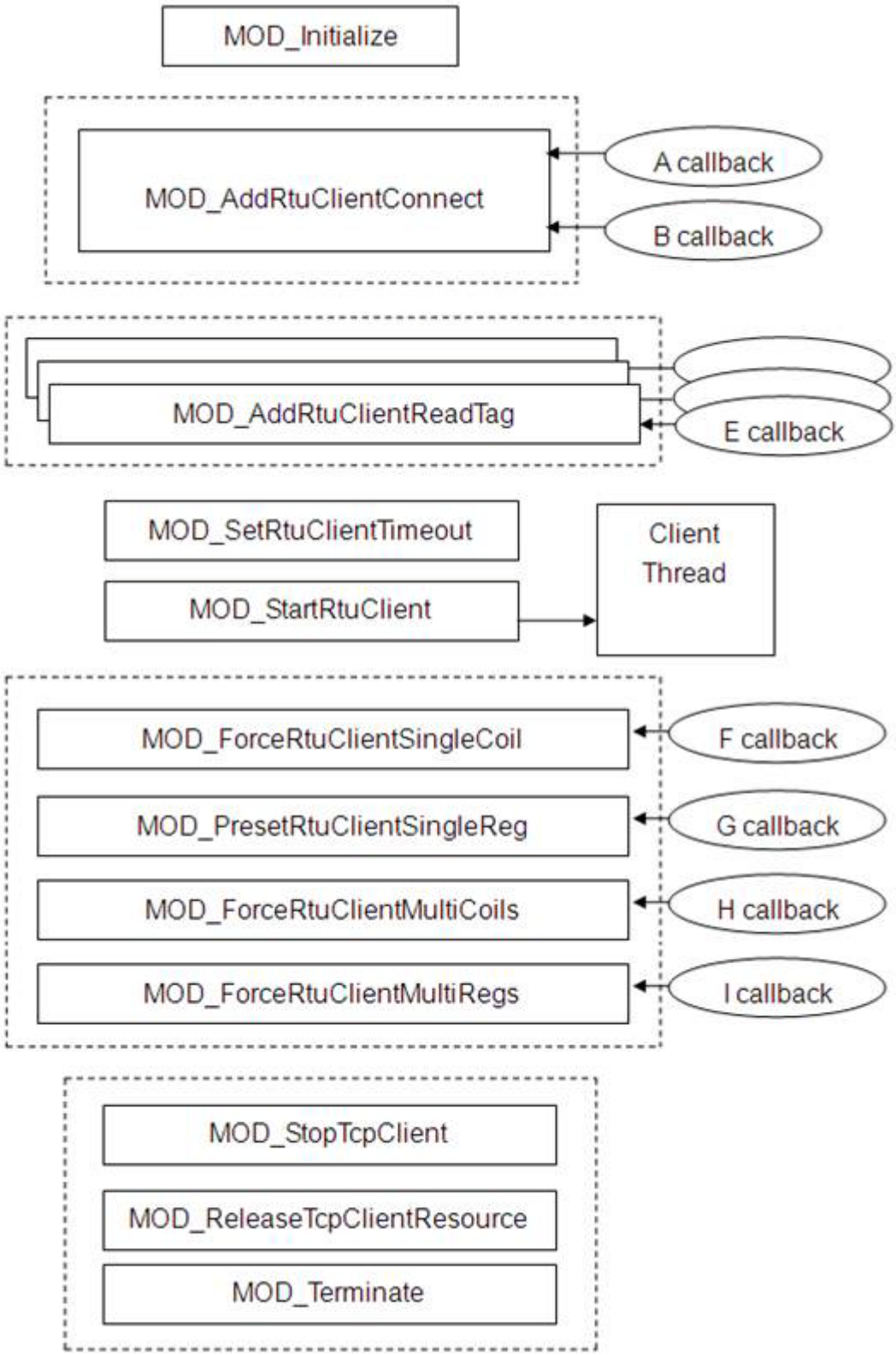
```
int main(int argc, char* argv[])
{
      unsigned long ulClientHandle;
      unsigned char byData[8] = {0};

      MOD_Initialize();
      MOD_AddRtuClientConnect(1,          //COM Id
                              CBR_9600,   //baudrate
                              DATA_8,     //data bits
                              NOPARITY,   //parity
                              ONESTOPBIT, //stop bit
                              500,        // scan interval (ms)
                              200,        // transaction timeout (ms)
                              ConnectRtuServerCompletedEventHandler, //callback function
                              DisconnectRtuServerCompletedEventHandler, //callback function
                              NULL,       // user defined
                              &ulClientHandle); // MODBUS-RTU client handle.
   // Slot Id 0: APAX-5060
      MOD_AddRtuClientReadTag(ulClientHandle,
                              1,                      // The server address.
                              MODBUS_READCOILSTATUS, // MODBUS reading type.
                              1,                      // start address for reading
                              12,                     // total point for reading.
                              ClientReadCoil_1_64_Handler); //callback function
    // Slot Id 1: APAX-5028
      MOD_ AddRtuClientReadTag(ulClientHandle,
                              1,
                              MODBUS_READHOLDREG,
                              33,                  // start address for reading
                              8,
                              ClientReadReg_33_64_Handler);
      // Slot Id 2: APAX-5045
      MOD_ AddRtuClientReadTag(ulClientHandle,
                              1,
                              MODBUS_READCOILSTATUS,
                              141,        // start address for reading, read DO only
                              12,
                              ClientReadCoil_129_192_Handler);

      //set COM port read and write timeout
      MOD_SetRtuClientTimeout(1000, 1000);
      //start the MODBUS-RTU client
      MOD_StartRtuClient();
      printf("MODBUS-RTU client started...\n");
            printf("Press any key to exit...\n");
```

```
        while (kbhit() == 0)
        {
                if( m_writeEnable == TRUE && m_isConnected == TRUE)
                {
                        m_writeEnable = FALSE;
                   // perform write action to the APAX-5060 12ch Relay
                        lRet = MOD_ForceRtuClientMultiCoils(ulClientHandle,
                                                                1,      // The server address.
                                                                1,      // start address for writing
                                                                12,      // total point for writing.
                                                                2,      // total byte length of data
                                                                byData,  // The byte data of coils to write
                                                                 ModbusWriteCompletedEventHandler);    //callback
function
                        if (0 == lRet)
                     {
                                byData[0]++;
                                byData[1]++;
                        }
                     }
         }
         //stop the MODBUS-RTU client
         MOD_StopRtuClient();
         MOD_ReleaseRtuClientResource();
         MOD_Terminate();
         return 0;
}

void ConnectRtuServerCompletedEventHandler(long lResult, char *i_szIp, void *i_Param)
{
        if (lResult == MODERR_SUCCESS)
                m_isConnected = TRUE;
        else
                m_isConnected = FALSE;
}

void DisconnectRtuServerCompletedEventHandler(long lResult, char *i_szIp, void *i_Param)
{
        m_isConnected = FALSE;
}


void ModbusWriteCompletedEventHandler(long lResult, void *i_Param)
{
        m_writeEnable = TRUE;
}
```

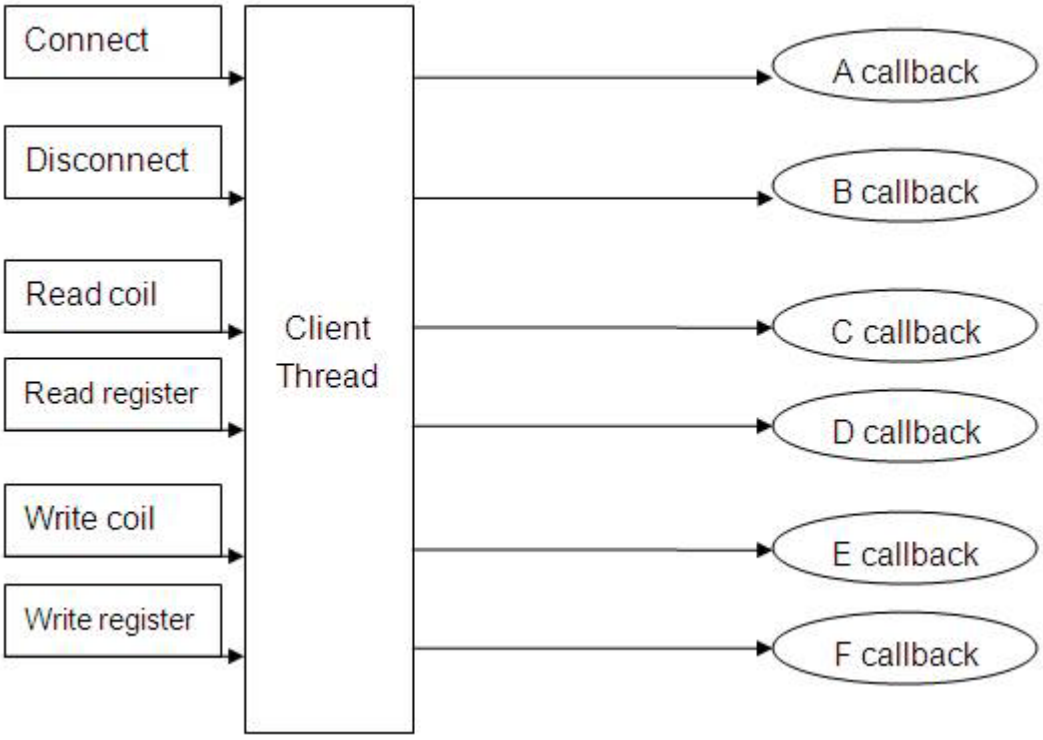
```
//Id 0: APAX-5060 (read 8ch Relay value)
// The order of this value is: [CH 0~7], [CH 8~15]…
void ClientReadCoil_1_64_Handler(long lResult, unsigned char *i_byData, int i_iLen, void *i_Param)
{
	if (lResult == MODERR_SUCCESS)
	{
		printf("ReadCoil[1~64] = ");
		for (int i=0; i<i_iLen; i++)
			printf("%02X ", i_byData[i]);
		printf("\n");
	}
}

//Id 1: APAX-5028 (read 8ch AO value)
// The order of this value is:
// [CH0 High byte], [CH0 Low byte], [CH1 High byte], [CH1 Low byte]…
void ClientReadReg_33_64_Handler(long lResult, unsigned char *i_byData, int i_iLen, void *i_Param)
{
	if (lResult == MODERR_SUCCESS)
	{
		printf("ReadReg[33~64] = ");
		for (int i=0; i<i_iLen; i++)
			printf("%02X ", i_byData[i]);
		printf("\n");
	}
}

//Id 2: APAX-5045 (read 12ch DO value only)
// The order of this value is: [CH 0~7], [CH 8~15]…
void ClientReadCoil_129_192_Handler(long lResult, unsigned char *i_byData, int i_iLen, void
*i_Param)
{
	if (lResult == MODERR_SUCCESS)
	{
		printf("ReadCoil[129~192] = ");
		for (int i=0; i<i_iLen; i++)
			printf("%02X ", i_byData[i]);
		printf("\n");
	}
}
```

In general, The MODBUS-RTU client calling and working flow can be shown as below:

- Calling flow



- Working flow



When you have built the MODBUS-RTU server and client example, please start the MODBUS-RTU server program on APAX-5522 Embedded Linux and then execute the MODBUS-RTU client program on your computer or you may use other software which supports Modbus RTU client (like Modscan or HMI/SCADA software) to perform the read or write operation to the APAX-5522 Embedded Linux. For my example, you can see the message as following:

- The MODBUS-RTU server execution screen:

```
# ./exModbusRtuServer

SYS_GetModuleID: [Slot 0]: 0x50600000
SYS_GetModuleID: [Slot 1]: 0x50280000
SYS_GetModuleID: [Slot 2]: 0x50450000
MODBUS-RTU server started...
Press any key to exit..
```

- The MODBUS-RTU client execution screen:

```
Connect to COM-1 result = 0    <-  connect to server success
MODBUS-RTU client started...
Press any key to exit...

ReadCoil[1~64] = 00 00    <-  Id 0: APAX-5060 12ch DI value
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58    <-  Id 1: APAX-5028 8ch AO value
ReadCoil[129~192] = F3 05    <- Id 2: APAX-5045 12ch DO value
MOD_ForceRtuClientMultiCoils start (1) = 0    <-  perform write action to Id 0: APAX-5060
Coil write result = 0
ReadCoil[1~64] = 00 00
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F3 05
MOD_ForceRtuClientMultiCoils start (2) = 0
Coil write result = 0
ReadCoil[1~64] = 01 01     <-  Id 0: APAX-5060 12ch DI value has changed
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F3 05
MOD_ForceRtuClientMultiCoils start (3) = 0
Coil write result = 0
ReadCoil[1~64] = 02 02
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F3 05
MOD_ForceRtuClientMultiCoils start (4) = 0
Coil write result = 0
ReadCoil[1~64] = 03 03
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F3 05
MOD_ForceRtuClientMultiCoils start (5) = 0
Coil write result = 0
ReadCoil[1~64] = 04 04
ReadReg[33~64] = 67 DB 8E EB 30 00 40 00 40 CB 8F C0 3E 80 1B 58
ReadCoil[129~192] = F3 05
Loop count = 6
Read failed = 0
Write failed = 0
Connect count = 0
Disconnect count = 0

#
```

4.5 Advantech serial port library

The libadvserial.so.1.0.0 is a library file. User can develop their serial applications base on this library. The API functions in the library are showing as below:

4.5.1 Advantech serial port functions

4.5.1.1 ComPort_OpenComPort

int ComPort_OpenComPort(int iComPort)
■ Purpose: This function opens a handle that operates the APAX serial port. ■ Parameters: iComPort = 0 means the device node is /dev/ttyAP0. 1 means the device node is /dev/ttyAP1. … ■ Return 1. ERR_NONE (0), open serial port succeeded, the handle will be return for function use until closed. 2. ERR_FAIL (-1), Call perror to get extended error information.

4.5.1.2 ComPort_CloseComPort

void ComPort_CloseComPort(int i_iPortHandle)
■ Purpose: This function closes the open handle of the APAX serial port. ■ Parameters: handle = driver handle ■ Return No return value.

4.5.1.3 ComPort_PurgeComm

void ComPort_PurgeComm(int i_iPortHandle, int flag)
■ Purpose: This function is used to discards data. ■ Parameters: i_iPortHandle = driver handle flag = 0 flushe data received but not read. 1 flushe data written but not transmitted. 2 flush both data received but not read, and data written but not transmitted. ■ Return No return value.

4.5.1.4 ComPort_GetComPortState

int ComPort_GetComPortState(int i_iPortHandle, struct ComPortState *o_strState)

- Purpose:

Gets the parameters associated with the serial port by port handle.

- Parameters:

i_iPortHandle = driver handle

o_strState = the variable to hold the returned serial port parameters.

```
struct ComPortState {
        int iBaudrate;
        int iParity;
        int iDatabits;
        int iStopbits;
};
```

- Return

1. ERR_NONE (0), gets serial port state succeeded.
2. ERR_FAIL (-1), Call perror to get extended error information.

4.5.1.5 ComPort_SetComPortState

int ComPort_SetComPortState(int i_iPortHandle, struct ComPortState *i_strState)

■ Purpose:

Sets the parameters associated with the serial port by port handle.

■ Parameters:

i_iPortHandle = driver handle

i_strState is referenced by =

```
struct ComPortState {
        int iBaudrate;
        int iParity;
        int iDatabits;
        int iStopbits;
};
```

Please fill out the information on the structure according to the following parameters

iBaudrate =

- CBR_1200
- CBR_2400
- CBR_4800
- CBR_9600
- CBR_19200
- CBR_38400
- CBR_57600
- CBR_115200

iParity =

- PARITY_NONE
- PARITY_EVEN
- PARITY_ODD
- PARITY_MARK
- PARITY_SPACE

iDatabits

- DATABITS_5
- DATABITS_6
- DATABITS_7
- DATABITS_8

iStopbits

- STOPBITS_1
- STOPBITS_2

■ Return

1. ERR_NONE (0), gets serial port state succeeded.
2. ERR_FAIL (-1), Call perror to get extended error information.

4.5.1.6 ComPort_Send

int ComPort_Send(int i_iPortHandle, int i_iSendTimeout, int i_iDataLen, unsigned char *i_byData)
■ Purpose: This function transmits the data from a port handle. ■ Parameters: i_iPortHandle = driver handle i_iSendTimeout = the transmission timeout, in millisecond. I_ iDataLen = The total byte length of data. i_byData = The byte data to transmit. ■ Return 1. On success, the number of bytes written is returned. 2. ERR_FAIL (-1), Call perror to get extended error information.

4.5.1.7 ComPort_Recv

int ComPort_Recv(int i_iPortHandle, int i_iRecvTimeout, int i_iDataLen, unsigned char *o_byData)
■ Purpose: This function receives the data from a port handle. ■ Parameters: i_iPortHandle = driver handle i_ iRecvTimeout = the receive timeout, in millisecond. I_ iDataLen = The total byte length of data to read. o_byData = The byte data received. ■ Return 1. On success, the number of bytes read is returned. 2. ERR_FAIL (-1), Call perror to get extended error information.

4.5.1.8 ComPort_SendByte

int ComPort_SendByte(int i_iPortHandle, int i_iSendTimeout, int i_iDataLen, unsigned char *i_byData)
■ Purpose: This function transmits the data from a port handle. ■ Parameters: i_iPortHandle = driver handle i_iSendTimeout = the transmission timeout, in millisecond. I_ iDataLen = The total byte length of data. i_byData = The byte data to transmit. ■ Return 1. On success, the number of bytes written is returned. 2. ERR_FAIL (-1), Call perror to get extended error information.

4.5.1.9 ComPort_RecvByte

int ComPort_RecvByte(int i_iPortHandle, int i_iRecvTimeout, unsigned char *o_byData)
■ Purpose: This function receives the data from a port handle. ■ Parameters: i_iPortHandle = driver handle i_ iRecvTimeout = the receive timeout, in millisecond. o_byData = The byte data received. ■ Return 1. On success, the number of bytes read is returned. 2. ERR_FAIL (-1), Call perror to get extended error information.

4.6 Advantech Serial Library Demo

After decompress the SDK file, user can find the files for the example code under the path "Apax5522-SDK/AdvSerialPort". In this section, we will use the Advantech ADAM-4000 I/O modules to do the demonstration. These modules support the ASCII command and MODBUS RTU protocol so that users can more easily use the library functions to develop their serial port application.

4.6.1 Advantech Serial Port Example

The example of "AdamIO_SerialTest.cpp" will illustrate how to read or write the value to the ADAM-4000 I/O module base on these serial port library APIs. The mainly source code of this program is as follows:

- The AdamIO_SerialTest.cpp mainly source code

```
int main(int argc, char* argv[])
{
	/*
	ComId = 1  (/dev/ttyAP0)
	Baudrate =
		CBR_1200
		CBR_2400
		CBR_4800
		CBR_9600
		CBR_19200
		CBR_38400
		CBR_57600
		CBR_115200
	Databits =
		DATABITS_5
		DATABITS_6
		DATABITS_7
		DATABITS_8
	Parity =
		PARITY_NONE : No parity
		PARITY_EVEN : Even
		PARITY_ODD: Odd
		PARITY_MARK : Mark
		PARITY_SPACE : Space
	Stopbits =
		STOPBITS_1: 1 stop bit
		STOPBITS_2: 2 stop bits
		*/
```

```
struct ComPortState comState;
// ADAM-4017, Address=1, 8 AI
int iSlotId1 = 1;
unsigned short iStartIndex1 = 1;
unsigned short iTotalPoint1 = 8;
unsigned char bySend1[16] = {0};
unsigned char RecvBuf1[64] = {0};
// ADAM-4069, Address=2, 8 DO Relay
int iSlotId2 = 2;
unsigned short iStartIndex2 = 17;
unsigned char bySend2[16] = {0};
unsigned short iData2 = 0xFF00;
unsigned char RecvBuf2[2] = {0};

memset(&comState,0,sizeof(struct ComPortState));
hdlCom = ComPort_OpenComPort(ComId);

printf("\n**** Advantech Serial Port Test Start ****\n");
ComPort_GetComPortState(hdlCom, &comState);
comState.iBaudrate = CBR_9600;
comState.iParity = PARITY_NONE;
comState.iDatabits = DATABITS_8;
comState.iStopbits = STOPBITS_1;
ComPort_SetComPortState(hdlCom, &comState);

////////////////////////////////////////////////////////////////////////
// Advantech MODBUS RTU command test //
////////////////////////////////////////////////////////////////////////
printf("\n--- Read ADAM 4017 8-ch AI (Raw Data) ---\n");
ConstructSendReadPacket(iSlotId1, MODBUS_READHOLDREG, iStartIndex1,
                        iTotalPoint1, (unsigned char*)bySend1, &iDataLen);
CRC16((unsigned char*)bySend1, iDataLen, (unsigned char*)bySend1+iDataLen);
iSendLen = iDataLen + 2; // add CRC 2 bytes length

ComPort_Send(hdlCom, iSendTimeout, iSendLen, (unsigned char*)bySend1);
ComPort_RecvByte(hdlCom, iRecvTimeout, (unsigned char*)RecvBuf1);
for (iCnt = 1; iCnt < 9; iCnt++)
{
    printf("ADAM 4017: [CH %d] = 0x%X\n",(iCnt-1),
        ((RecvBuf1[2*iCnt+1] << 8)|RecvBuf1[2*iCnt+2]));
}
```

```
printf("\n--- Set ADAM 4069 8-ch Relay ON ---\n");
for (iCnt = 0; iCnt < 8; iCnt++)
{
        ConstructSendWriteSinglePacket(iSlotId2, MODBUS_FORCESINGLECOIL,
                                iCnt+iStartIndex2, iData2, (unsigned char*)bySend2, &iDataLen);
        CRC16((unsigned char*)bySend2, iDataLen, (unsigned char*)bySend2+iDataLen);
        iSendLen = iDataLen + 2; // add CRC 2 bytes length
        ComPort_Send(hdlCom, iSendTimeout, iSendLen, (unsigned char*)bySend2);
        ComPort_Recv(hdlCom, iRecvTimeout, 8,(unsigned char*)RecvBuf2);
        ConstructSendReadPacket(iSlotId2, MODBUS_READCOILSTATUS,
                                        iCnt+iStartIndex2, 1, (unsigned char*)bySend2, &iDataLen);
        CRC16((unsigned char*)bySend2, iDataLen, (unsigned char*)bySend2+iDataLen);
        iSendLen = iDataLen + 2; // add CRC 2 bytes length
        ComPort_Send(hdlCom, iSendTimeout, iSendLen, (unsigned char*)bySend2);
        ComPort_RecvByte(hdlCom, iRecvTimeout, (unsigned char*)RecvBuf2);
        if(RecvBuf2[3] == 0)
                printf("[CH %d] OFF: %X \n",iCnt,RecvBuf2[3]);
        else
                printf("[CH %d] ON: %X \n",iCnt,RecvBuf2[3]);
        usleep(500000);
}

printf("\n--- Set ADAM 4069 8-ch Relay OF ---\n");
for (iCnt = 0; iCnt < 8; iCnt++)
{
        ConstructSendWriteSinglePacket(iSlotId2, MODBUS_FORCESINGLECOIL,
                                    iCnt+iStartIndex2, 0x0, (unsigned char*)bySend2, &iDataLen);
        CRC16((unsigned char*)bySend2, iDataLen, (unsigned char*)bySend2+iDataLen);
        iSendLen = iDataLen + 2; // add CRC 2 bytes length
        ComPort_Send(hdlCom, iSendTimeout, iSendLen, (unsigned char*)bySend2);
        ComPort_Recv(hdlCom, iRecvTimeout, 8, (unsigned char*)RecvBuf2);
        ConstructSendReadPacket(iSlotId2, MODBUS_READCOILSTATUS,
                                        iCnt+iStartIndex2, 1, (unsigned char*)bySend2, &iDataLen);
        CRC16((unsigned char*)bySend2, iDataLen, (unsigned char*)bySend2+iDataLen);
        iSendLen = iDataLen + 2; // add CRC 2 bytes length
        ComPort_Send(hdlCom, iSendTimeout, iSendLen, (unsigned char*)bySend2);
        ComPort_RecvByte(hdlCom, iRecvTimeout, (unsigned char*)RecvBuf2);
        if(RecvBuf2[3] == 0)
                printf("[CH %d] OFF: %X \n",iCnt,RecvBuf2[3]);
        else
                printf("[CH %d] ON: %X \n",iCnt,RecvBuf2[3]);

        usleep(500000);
        }
```

```
    ////////////////////////////////////////////////////////////////////
    // Advantech ASCII command test        //
    ////////////////////////////////////////////////////////////////////
    printf("\n--- Advantech ASCII command test ---\n");
    printf("[Read module ID & firmware version]\n");
    char ascii_cmd1[6] = "$07T\r";
    char ascii_cmd2[6] = "$07F\r";
    char ascii_buf[20] = {0};
    int cmdLen;
    int recvLen = 0;
    cmdLen = strlen(ascii_cmd1);
    ComPort_Send(hdlCom, iSendTimeout, cmdLen, (unsigned char*)ascii_cmd1);
    recvLen = ComPort_RecvByte(hdlCom, iRecvTimeout, (unsigned char*)ascii_buf);
    if(recvLen > 0)
        printf("Module ID: %s\n",(ascii_buf+3));

    memset(ascii_buf,0,sizeof(ascii_buf));

    cmdLen = strlen(ascii_cmd2);
    ComPort_Send(hdlCom, iSendTimeout, cmdLen, (unsigned char*)ascii_cmd2);
    recvLen = ComPort_RecvByte(hdlCom, iRecvTimeout, (unsigned char*)ascii_buf);
    if(recvLen > 0)
        printf("Firmware version: %s\n",(ascii_buf+3));

    ComPort_CloseComPort(hdlCom);
    printf("\n****      END    ****\n");
}

void ConstructSendReadPacket(unsigned char i_byAddr,
                             unsigned char i_byFun,
                             unsigned short i_iStartIndex,
                             unsigned short i_iTotalPoint,
                             unsigned char *o_byPacket,
                             int *o_iTotalByte)
{
    i_iStartIndex--;
    o_byPacket[0] = i_byAddr;
    o_byPacket[1] = i_byFun;
    o_byPacket[2] = (unsigned char)(i_iStartIndex >> 8);
    o_byPacket[3] = (unsigned char)(i_iStartIndex & 0x00FF);
    o_byPacket[4] = (unsigned char)(i_iTotalPoint >> 8);
    o_byPacket[5] = (unsigned char)(i_iTotalPoint & 0x00FF);
    *o_iTotalByte = 6;
}
```

```
void ConstructSendWriteSinglePacket(unsigned char i_byAddr,
                                    unsigned char i_byFun,
                                    unsigned short i_iStartIndex,
                                    unsigned short i_iData,
                                    unsigned char *o_byPacket,
                                    int *o_iTotalByte)
{
        i_iStartIndex--;
        o_byPacket[0] = i_byAddr;
        o_byPacket[1] = i_byFun;
        o_byPacket[2] = (unsigned char)(i_iStartIndex >> 8);
        o_byPacket[3] = (unsigned char)(i_iStartIndex & 0x00FF);
        o_byPacket[4] = (unsigned char)(i_iData >> 8);
        o_byPacket[5] = (unsigned char)(i_iData & 0x00FF);
        *o_iTotalByte = 6;
}
```

After you build this example, please upload the executable file "AdamIO_SerialTest" to the APAX-5522 Embedded Linux. The result after execution you can see a similar message as below:

```
# ./AdamIO_SerialTest
Open /dev/ttyAP0 ,Handle = 3

**** Advantech Serial Port Test Start ****

--- Read ADAM 4017 8-ch AI (Raw Data) ---
ADAM 4017: [CH 0] = 0x0
ADAM 4017: [CH 1] = 0x0
ADAM 4017: [CH 2] = 0x0
ADAM 4017: [CH 3] = 0x0
ADAM 4017: [CH 4] = 0x3BC8
ADAM 4017: [CH 5] = 0x0
ADAM 4017: [CH 6] = 0x0
ADAM 4017: [CH 7] = 0x0

--- Set ADAM 4069 8-ch Relay ON ---
[CH 0] ON: 1
[CH 1] ON: 1
[CH 2] ON: 1
[CH 3] ON: 1
[CH 4] ON: 1
[CH 5] ON: 1
[CH 6] ON: 1
[CH 7] ON: 1
```

```
--- Set ADAM 4069 8-ch Relay OF ---
[CH 0] OFF: 0
[CH 1] OFF: 0
[CH 2] OFF: 0
[CH 3] OFF: 0
[CH 4] OFF: 0
[CH 5] OFF: 0
[CH 6] OFF: 0
[CH 7] OFF: 0

--- Advantech ASCII command test ---
[Read module ID & firmware version]
Module ID: ADAM4051
Firmware version: A2.02

****      END    ****
#
```

Chapter 5

Timestamp Function Examples Demo

The APAX-5522 Linux supports the timestamp function. It increases the convenience of tracking events for good protection features when the source signal is unexpected. Additionally, the APAX-5000PE series modules are IEC 61850-3 compliant and can be used in power and energy applications e.g. smart substation. The timestamp with accuracy is 1 msec.

After decompress the SDK file, user can find the files for the example code under the path "Apax5522-SDK/Timestamp". In this section, we will illustrate how to use the timestamp APIs to get the DI and AI values with time stamp information.

5.1 Get AI Module Values With Timestamp

The APAX-5017PE module has 12 analog input channels. The purpose of this example will get the AI channels values with timestamp. The mainly source code of timeStampAI.c is as follows:

```
#include "adsdio.h"
struct AiSingleTSData m_AiTSData[32] = {{0}};
LONG g_lHandle = 0;
////////////////////////////////////////////////////////////////////////////
int OpenLib(void)
{
	DWORD dwVersion = 0;
	//Initialize the driver
	if(ERR_SUCCESS != ADAMDrvOpen(&g_lHandle) )
		return ADV_FAIL;
	if (ERR_SUCCESS == SYS_GetVersion(g_lHandle, &dwVersion) )
		printf(" ADSDIO library version is %lX\n\n", dwVersion);
	else
		return ADV_FAIL;
	return ADV_SUCCESS;
}

void CloseLib(void)
{
	//Terminate the driver
	if(0 != g_lHandle)
	{
		ADAMDrvClose(&g_lHandle);
		g_lHandle = 0;
	}
}
```

```
int main(int argc, char* argv[])
{
	WORD    wSlotNum = 0; //module id start from 0
	DWORD   dwTSCounter = 0;
	DWORD   dwEnable = 0;
	LONG    TotalData = 0;
	LONG    iCount = 0;
	time_t timep;
	struct tm *p;
	int     min = 1;
	int     i = 0;
	if(argc > 1)
		wSlotNum = atoi(argv[1]);

	printf("==== APAX-5017PE AI Timestamp example ====\n");
	if (ADV_SUCCESS == OpenLib() )
	{
	  / /Stop time stamp counter
	   dwEnable = 0;
	   SYS_StartStopTimeStampCounter(g_lHandle, dwEnable);
	      //clear all ring buffer
	      SYS_ClearTSRingBuffer(g_lHandle, 0xFF);
	      usleep(10000);
	      //Start time
	      timep = time(NULL);
	      p=localtime(&timep);
	      printf("Timestamp start from time:\n%s",asctime(p));
	      printf("=============================================\n");
	      //Start time stamp counter
	      dwEnable = 1;
	      SYS_StartStopTimeStampCounter(g_lHandle, dwEnable);
	      //Please note that every 500 microseconds DSP counted once.
	      //Get the AI data within 1 minute.
	      while(dwTSCounter < min * 120000)
	      {
	            SYS_GetTimeStampCounter(g_lHandle, &dwTSCounter);
	            iCount = AI_GetValuesWithTimeStamp(g_lHandle, wSlotNum,
	                                                    (DWORD *) m_AiTSData);
	            if((iCount > 0) && (iCount < 32))
	            {
	                  printf("Get %ld data\n",iCount);
	                  for(i = 0; i < iCount; i++)
	                  {
	                  printf("Single AI Data = 0x%X\n",m_AiTSData[i].wAiData);
	                  printf("Channel No. = %d\n",(m_AiTSData[i].wChannelIdx >> 8));
	               printf("(offset) = %ld ms\n",(m_AiTSData[i].dwTimeStamp/2));
	                  printf("----------------------------\n");
	                  TotalData++;
	                  }
	            }
	        }
```

```
                //Stop time stamp counter
                dwEnable = 0;
                SYS_StartStopTimeStampCounter(g_lHandle, dwEnable);
                printf("%ld data within 1 minute.\n",TotalData);
        }
        else
                printf("Open library fail.\n");
        CloseLib();
        return 0;
}
```

In general, this program development involves the following steps.

1. Opens a handle that operates the APAXIO driver.
   ADAMDrvOpen (&g_lHandle)
2. Clear timestamp ring buffer
   SYS_ClearTSRingBuffer(LONG handle, WORD i_wSlot)
3. Start timestamp counter
   SYS_StartStopTimeStampCounter(LONG handle, 1)
4. Get AI channel values with timestamp of the indicated slot.
   AI_GetValuesWithTimeStamp(LONG handle, WORD i_wSlot, DWORD *o_dwTSData);
   The variable of o_dwTSData holds the AI value with time stamp information.
   Please refer to the following definition:

	Bit15 ~ Bit8	Bit7 ~ Bit0
Offset +0	AI_0 Value	
Offset +1	Channel index (0)	Reserved for 24 bit AI Value
Offset +2	AI_0 Timestamplow	
Offset +3	AI_0 Timestamphigh	

   Total length of AI single channel is 4 WORD = 8 bytes
5. Stop timestamp counter
   SYS_StartStopTimeStampCounter(LONG handle, 0)
6. Closes the open handle of the APAXIO driver.
   ADAMDrvClose (&g_lHandle)

After you build this example, please upload the executable file "timestampAI" to the APAX-5522 Embedded Linux. Then run and enter the module slot Id address, for example, my module is 1. The result after execution you can see a similar message as below:

```
# ./timeStampAI 1
==== APAX-5017PE AI Timestamp example ====
ADSDIO library version is 102
Timestamp start from time:
Thu Jul 11 08:42:51 2012
=========================================
Get 1 data
Single AI Data = 0x813C
Channel No. = 4
(offset) = 62 ms
---------------------------
Get 1 data
Single AI Data = 0x80AB
Channel No. = 5
(offset) = 145 ms
---------------------------
Get 1 data
Single AI Data = 0x8067
Channel No. = 6
(offset) = 229 ms
---------------------------
Get 1 data
Single AI Data = 0x8040
Channel No. = 7
(offset) = 312 ms
---------------------------
Get 1 data
Single AI Data = 0x8029
Channel No. = 8
(offset) = 396 ms
---------------------------
Get 1 data
Single AI Data = 0x801E
Channel No. = 9
(offset) = 479 ms
---------------------------
Get 1 data
Single AI Data = 0x8017
Channel No. = 10
(offset) = 562 ms
---------------------------
Get 1 data
Single AI Data = 0x9465
Channel No. = 11
(offset) = 646 ms
---------------------------
```

```
Get 1 data
Single AI Data = 0x8D54
Channel No. = 0
(offset) = 729 ms
----------------------------
Get 1 data
Single AI Data = 0x8742
Channel No. = 1
(offset) = 813 ms
----------------------------
Get 1 data
Single AI Data = 0x8400
Channel No. = 2
(offset) = 896 ms
...
...
Get 1 data
Single AI Data = 0x9465
Channel No. = 11
(offset) = 59699 ms
----------------------------
Get 1 data
Single AI Data = 0x8D56
Channel No. = 0
(offset) = 59783 ms
----------------------------
Get 1 data
Single AI Data = 0x8744
Channel No. = 1
(offset) = 59866 ms
----------------------------
Get 1 data
Single AI Data = 0x8401
Channel No. = 2
(offset) = 59950 ms
----------------------------
719 data within 1 minute.
/***        END        ***/
```

5.2 Get DI module values with timestamp

The APAX-5040PE module has 24 digital input channels. The purpose of this example will get the DI channels values with timestamp. The mainly source code of timeStampDI.c is as follows:

```
#include "adsdio.h"
struct Di32TSData m_DiTSData[32] = {{0}}
LONG g_lHandle = 0;
////////////////////////////////////////////////////////////////////////
int OpenLib(void)
{
	DWORD dwVersion = 0;
	//Initialize the driver
	if(ERR_SUCCESS != ADAMDrvOpen(&g_lHandle) )
		return ADV_FAIL;
	if (ERR_SUCCESS == SYS_GetVersion(g_lHandle, &dwVersion) )
		printf(" ADSDIO library version is %lX\n\n", dwVersion);
	else
		return ADV_FAIL;
	return ADV_SUCCESS;
}
void CloseLib(void)
{
	//Terminate the driver
	if(0 != g_lHandle)
	{
		ADAMDrvClose(&g_lHandle);
		g_lHandle = 0;
	}
}
int main(int argc, char* argv[])
{
	WORD    wSlotNum = 0; //module id start from 0
	DWORD   dwTSCounter = 0;
	DWORD   dwEnable = 0;
	LONG    TotalData = 0;
	LONG    iCount = 0;
	time_t timep;
	struct tm *p;
	int     min = 1;
	int     i = 0;

	if(argc > 1)
			wSlotNum = atoi(argv[1]);
```

```
    printf("==== APAX-5040PE DI Timestamp example ====\n");
    if (ADV_SUCCESS == OpenLib() ){
      //Stop time stamp counter
      dwEnable = 0;
      SYS_StartStopTimeStampCounter(g_lHandle, dwEnable);
        //clear all ring buffer
        SYS_ClearTSRingBuffer(g_lHandle, 0xFF);
        usleep(10000);
        //Start time
        timep = time(NULL);
        p=localtime(&timep);
        printf("Timestamp start from time:\n%s",asctime(p));
        printf("=============================================\n");
        //Start time stamp counter
        dwEnable = 1;
        SYS_StartStopTimeStampCounter(g_lHandle, dwEnable);
        //Please note that every 500 microseconds DSP counted once.
        //Get the DI data within 1 minute.
        while(dwTSCounter < min * 120000) {
            SYS_GetTimeStampCounter(g_lHandle, &dwTSCounter);
            iCount = DI_GetValuesWithTimeStamp(g_lHandle, wSlotNum,
                                                (DWORD *) m_DiTSData);
            if((iCount > 0) && (iCount < 32))
            {
                printf("Get %ld data\n",iCount);
                for(i = 0; i < iCount; i++)
                {
                printf("DI 32 Data = 0x%lX\n",m_DiTSData[i].dwDi32Data);
                printf("(offset) = %ld ms\n",(m_DiTSData[i].dwTimeStamp/2));
                printf("----------------------------\n");
                TotalData++;
                }
            }
        }
        //Stop time stamp counter
        dwEnable = 0;
        SYS_StartStopTimeStampCounter(g_lHandle, dwEnable);
        printf("%ld data within 1 minute.\n",TotalData);
    }
    else
        printf("Open library fail.\n");
    CloseLib();
    return 0;
}
```

In general, this program development involves the following steps.

1. Opens a handle that operates the APAXIO driver.
   ADAMDrvOpen (&g_lHandle)
2. Clear timestamp ring buffer
   SYS_ClearTSRingBuffer(LONG handle, WORD i_wSlot)
3. Start timestamp counter
   SYS_StartStopTimeStampCounter(LONG handle, 1)
4. Get AI channel values with timestamp of the indicated slot.
   DI_GetValuesWithTimeStamp(LONG handle, WORD i_wSlot, DWORD *o_dwTSData)
   The variable of o_dwTSData holds the DI values with time stamp information.
   Please refer to the following definition:

	Bit15 ~ Bit0
Offset +0	DI Values 00~15
Offset +1	DI Values 16~31
Offset +2	Timestamplow
Offset +3	Timestamphigh

   Total length of all DI channel is 4 WORD = 8 bytes

   For example, if the dwDi32Data is 0x800003 that means channel 0,1 and 23 is high level and the rest of channel 2 to 22 is low level.
5. Stop timestamp counter
   SYS_StartStopTimeStampCounter(LONG handle, 0)
6. Closes the open handle of the APAXIO driver.
   ADAMDrvClose (&g_lHandle)

After you build this example, please upload the executable file "timestampDI" to the APAX-5522 Embedded Linux. Then run and enter the module slot Id address, for example, my module is 0. The result after execution you can see a similar message as below:

```
# ./timeStampDI 0
==== APAX-5040PE DI Timestamp example ====
ADSDIO library version is 102
Timestamp start from time:
Thu Jul 11 10:02:35 2012
==========================================
Get 1 data
DI 32 Data = 0x1
(offset) = 8163 ms
----------------------------
Get 1 data
DI 32 Data = 0x3
(offset) = 21499 ms
----------------------------
Get 1 data
DI 32 Data = 0x1
(offset) = 27211 ms
----------------------------
Get 1 data
DI 32 Data = 0x5
(offset) = 41323 ms
----------------------------
Get 1 data
DI 32 Data = 0x15
(offset) = 47603 ms
----------------------------
Get 1 data
DI 32 Data = 0x14
(offset) = 51851 ms
----------------------------
Get 1 data
DI 32 Data = 0x10
(offset) = 55396 ms
----------------------------
Get 1 data
DI 32 Data = 0x0
(offset) = 59355 ms
----------------------------
8 data within 1 minute.
/***        END        ***/
```

Chapter 6

WDT (Watch Dog Timer)

The WDT (Watch dog Timer) works like a watch dog function. The user can enable it or disable it. When the user enables WDT but the application does not acknowledge it, the system will reboot. The acknowledge time from 1 second to 255 seconds.

6.1 WDT Example

The user can find the example application and source code in /usr/adv/testwdt/ of APAX-5522. The mainly source code of this program is as follows:

- The WDT application example

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <linux/types.h>
#include <linux/watchdog.h>
#include <sys/time.h>
#include <sys/types.h>

#define WDIOC_SETTIMEOUT        _IOWR(WATCHDOG_IOCTL_BASE, 6, int)
#define WDIOC_GETTIMEOUT        _IOR(WATCHDOG_IOCTL_BASE, 7, int)
int fd;

/*
  This function simply sends an IOCTL to the driver, which in turn ticks the PC WDT to reset
    its internal timer so it doesn't trigger a computer reset.
*/
void keep_alive ( void )
{
   int dummy;
   ioctl( fd, WDIOC_KEEPALIVE, &dummy );
}

void usage ( char *filename )
{
   printf("Usage: #%s [OPTIONS]\n", filename);
   printf("OPTIONS:\n");
   printf(" -d: Disable WDT\n");
   printf(" -e: Enable WDT\n");
   printf(" -s SEC: Set timeout\n");
}
```

```
void disable ( void )
{
   int i_dis;
   write( fd, "V", 1 );
   i_dis = WDIOS_DISABLECARD;
   ioctl( fd, WDIOC_SETOPTIONS, &i_dis );
}

/*
  The main program. Run the program with "-d" to disable the card, or "-e" to enable the card.
*/
int main ( int argc, char *argv[] )
{
   int retval = 0;
   unsigned long timeout;
   fd_set rfds;
   struct timeval tv;
   char buf[ 256 ];

   if ( argc < 2 )
   {
      usage(argv[ 0 ]);
      exit(0);
   }

   fd = open( "/dev/watchdog", O_WRONLY );
   if ( fd == -1 )
   {
      printf( "WDT device not enabled.\n" );
      exit( -1 );
   }

   if ( !strncasecmp(argv[1], "-d", 2) )
   {
      disable();
   }
   else if ( !strncasecmp(argv[1], "-e", 2))
   {
      int i_en;
      i_en = WDIOS_ENABLECARD;
      retval = ioctl( fd, WDIOC_SETOPTIONS, &i_en );
      if ( retval != 0 )
      {
         disable();
         exit( -1 );
      }
```

```
    printf( "WDT enabled.\n" );
    ioctl( fd, WDIOC_GETTIMEOUT, &timeout );
    printf("WDT timeout has been set to %d seconds.\n", timeout );
    printf("You can type 'd' and press Enter to disable WDT.\n");
    while ( 1 )
    {
      //keep_alive();
      /* Watch stdin (fd 0) to see when it has input. */
      FD_ZERO(&rfds);
      FD_SET(0, &rfds);
      /* Wait up to one second */
      tv.tv_sec = 1;
      tv.tv_usec = 0;
      retval = select(1, &rfds, NULL, NULL, &tv);
      /* Don't rely on the value of tv now! */
      if (retval == -1)
        perror("select()");
      else if ( retval )
      {
        /* FD_ISSET(0, &rfds) will be true. */
        if ( fgets( buf, sizeof(buf), stdin ) == NULL )
          perror("fgets()");
        else if ( buf[ 0 ] == 'd' )
        {
          disable();
          break;
        }
        else
          printf("Invalid input.\n");
      }
    }
  }
  else if ( !strncasecmp(argv[ 1 ], "-s", 2))
  {
    if ( argc < 3 )
    {
      printf( "The input is invalid!\n");
      disable();
      exit( -1 );
    }
    timeout = atol( argv[ 2 ] );
    retval = ioctl( fd, WDIOC_SETTIMEOUT, &timeout );
    if ( retval != 0 )
    {
      disable();
      exit( -1 );
    }
```

```
    ioctl( fd, WDIOC_GETTIMEOUT, &timeout );
    printf( "WDT timeout has been set to %ld seconds.\n", timeout );
    printf( "After that, WDT will reset CPU.\n");
    printf( "You can use command: \"./testwdt -d\" to disable WDT.\n");
  }
  else
  {
    disable();
    usage(argv[ 0 ]);
  }
  exit ( 0 );
}
```

In general, this program development involves the following steps.

1. Opens a handle that operates the WDT driver.
   fd = open("/dev/watchdog", O_WRONLY);
2. Enable WDT
   i_en = WDIOS_ENABLECARD;
   ioctl(fd, WDIOC_SETOPTIONS, &i_dis);
3. Set WDT timeout
   ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
4. Disable WDT
   write(fd, "V", 1);
   i_dis = WDIOS_DISABLECARD;
   ioctl(fd, WDIOC_SETOPTIONS, &i_dis);

■ The result after execution you can see a similar message as below:

```
# ./testwdt
Usage: #./testwdt [OPTIONS]
OPTIONS:
 -d: Disable WDT
 -e: Enable WDT
 -s SEC: Set timeout

# ./testwdt -s 5
WDT timeout has been set to 5 seconds.
After that, WDT will reset CPU.
You can use command: "./testwdt -d" to disable WDT.
#
```

Appendix A

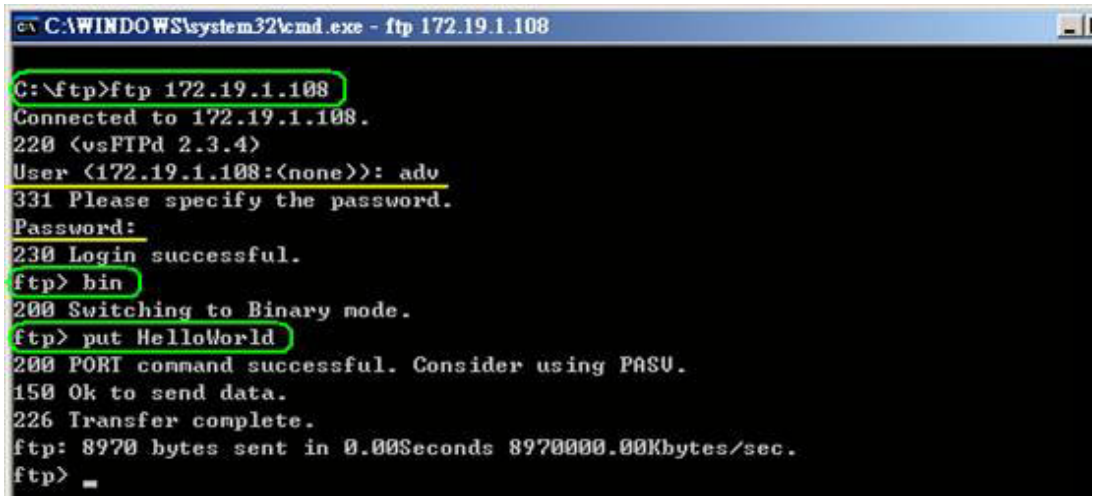
Upload and Update

A.1 Upload Your Program to APAX-5522 Linux

- Upload HelloWorld to APAX-5522 by using FTP

  In Windows, open the terminal and type command: "ftp 172.19.1.108

For example: IP address: 172.19.1.108 User name: adv Password: 1234

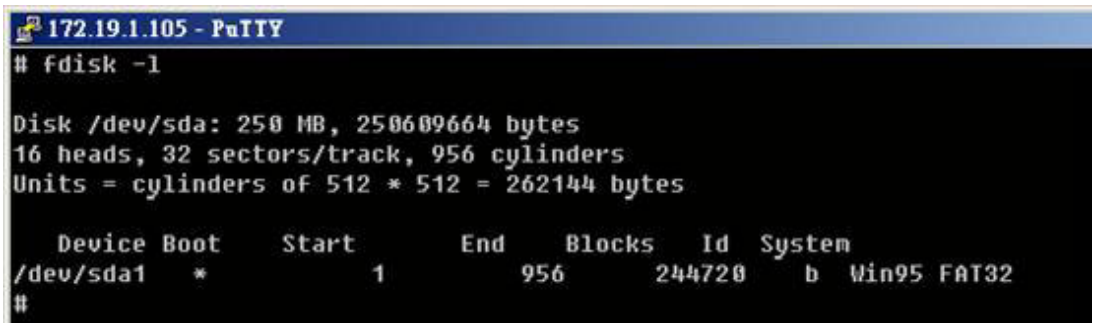


Basic FTP command:

■ !: exit FTP back to pc temporarily ■ bin: ransfer files in "Binary" mode ■ ascii: transfer files in "ASCII" mode ■ get: download file from APAX-5522 ■ put: upload file from PC to APAX-5522 ■ bye: exit FTP

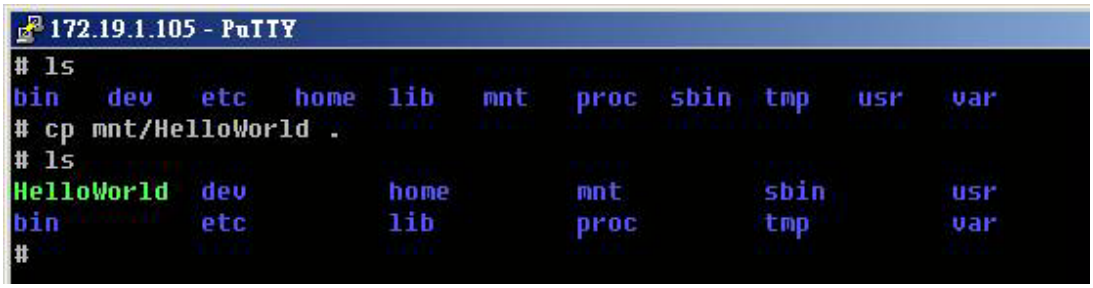
- Upload HelloWorld to APAX-5522 by using USB storage device.

1. Telnet and login to APAX-5522 device.
2. List the partition tables. This USB storage device node is "/dev/sda1".



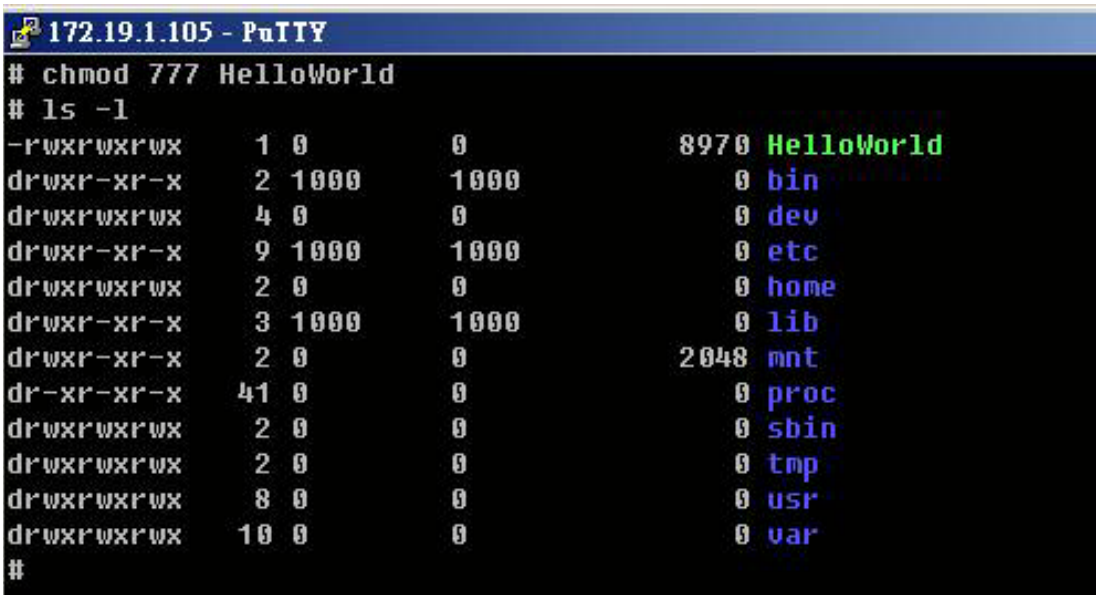
3. Mount USB storage device.

4. Copy HelloWorld to APAX-5522.



■ Execute "HelloWorld" on the APAX-5522

1. Change HelloWorld file mode bit as 755.

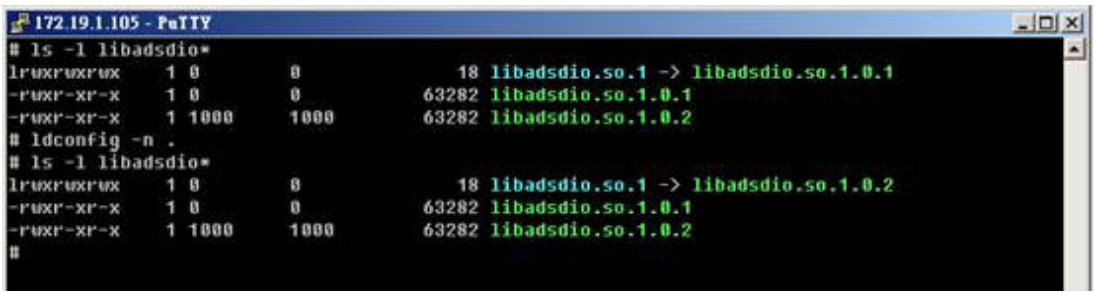


2. Execute HelloWorld program on APAX-5522

A.2 How to Update Advantech Library on the APAX-5522 Linux

For example: update APAX-5000 I/O modules library to version 1.0.2

1. According to the description of appendix A.1, please upload library libadsdio.so.1.0.2 to APAX-5522.
2. Change libadsdio.so.1.0.2 file mode bit as 755.
3. Move libadsdio.so.1.0.2 file to path "/lib/"
4. Type command "ldconfig" configure dynamic linker run-time bindings.



Appendix B

Analog I/O Board Range Settings

These ranges are provided for reference. Not all boards support all ranges. Please see hardware manual for valid ranges for a particular board.

	Setting Type	**Value (Hex)**
Millivolts DC (mV)	+/- 15mV +/- 50mV +/- 100mV +/- 150mV +/- 500mV 0~150mV 0~500mV	0x0100 0x0101 0x0102 0x0103 0x0104 0x0105 0x0106
Volts DC (V)	+/- 1V +/- 2.5V +/- 5V +/- 10V +/- 15V 0~1V 0~2.5V 0~5V 0~10V 0~15V	0x0140 0x0141 0x0142 0x0143 0x0144 0x0145 0x0146 0x0147 0x0148 0x0149
Milliamps (mA)	4~20mA +/-20mA 0~20mA	0x0180 0x0181 0x0182
Counter settings	Pulse/DIR Up/Down Up Frequency AB 1X AB 2X AB 4X	0x01C0 0x01C1 0x01C2 0x01C3 0x01C4 0x01C5 0x01C6
Pt-100 (3851)	Pt-100 (3851) -200~850 'C Pt-100 (3851) -120~130 'C Pt-100 (3851) -200~200 'C Pt-100 (3851) -100~100 'C Pt-100 (3851) -50~150 'C Pt-100 (3851) 0~100 'C Pt-100 (3851) 0~200 'C Pt-100 (3851) 0~400 'C Pt-100 (3851) 0~600 'C	0x0200 0x0201 0x0202 0x0203 0x0204 0x0205 0x0206 0x0207 0x0208
Pt-200 (3851)	Pt-200 (3851) -200~850 'C Pt-200 (3851) -120~130 'C	0x0220 0x0221
Pt-500 (3851)	Pt-500 (3851) -200~850 'C Pt-500 (3851) -120~130 'C	0x0240 0x0241
Pt-1000 (3851)	Pt-1000 (3851) -200~850 'C Pt-1000 (3851) -120~130 'C Pt-1000 (3851) -40~160 'C	0x0260 0x0261 0x0262

Pt-100 (3916)	Pt-100 (3916) -200~850 'C Pt-100 (3916) -120~130 'C Pt-100 (3916) -200~200 'C Pt-100 (3916) -100~100 'C Pt-100 (3916) -50~150 'C Pt-100 (3916) 0~100 'C Pt-100 (3916) 0~200 'C Pt-100 (3916) 0~400 'C Pt-100 (3916) 0~600 'C	0x0280 0x0281 0x0282 0x0283 0x0284 0x0285 0x0286 0x0287 0x0288
Pt-200 (3916)	Pt-200 (3916) -200~850 'C Pt-200 (3916) -120~130 'C	0x02A0 0x02A1
Pt-500 (3916)	Pt-500 (3916) -200~850 'C Pt-500 (3916) -120~130 'C	0x02C0 0x02C1
Pt-1000 (3916)	Pt-1000 (3916) -200~850 'C Pt-1000 (3916) -120~130 'C Pt-1000(3916)-40~160'C	0x02E0 0x02E1 0x02E2
Balco 500	Balcon(500) -30~120	0x0300
Ni 518	Ni(518) -80~100 'C Ni(518) 0~100 'C	0x0320 0x0321
Ni 508	Ni(508) 0~100 'C Ni(508) -50~200 'C	0x0340 0x0341
Thermistor 3K	Thermistor 3K 0~100 'C	0x0360
Thermistor 10K	Thermistor 10K 0~100 'C Thermistor 10K -50~100 'C	0x0380 0x0381
T/C TypeJ	T/C TypeJ 0~760 'C T/C TypeJ -200~1200 'C	0x0400 0x0401
T/C TypeK	T/C TypeK 0~1370 'C T/C TypeK -270~1372 'C	0x0420 0x0421
T/C TypeT	T/C TypeT -100~400 'C T/C TypeT -270~400 'C	0x0440 0x0441
T/C TypeE	T/C TypeE 0~1000 'C T/C TypeE -270~1000 'C	0x0460 0x0461
T/C TypeR	T/C TypeR 500~1750 'C T/C TypeR 0~1768 'C	0x0480 0x0481
T/C TypeS	T/C TypeS 500~1750 'C T/C TypeS 0~1768 'C	0x04A0 0x04A1
T/C TypeB	T/C TypeB 500~1800 'C T/C TypeB 300~1820 'C	0x04C0 0x04C1

Appendix C

MODBUS Address Mapping Example

C.1 Address (0x) with MOD_SetServerCoil (64)

Start Address	Length	Address (4x)	Description	Attribute
1	32	00001 ~ 00032	AI/O, counter channels' value (module with ID # 0)	Read/Write
33	32	00033 ~ 00064	AI/O, counter channels' value (module with ID # 1)	Read/Write
65	32	00065 ~ 00096	AI/O, counter channels' value (module with ID # 2)	Read/Write
97	32	00097 ~ 00128	AI/O, counter channels' value (module with ID # 3)	Read/Write
129	32	00129 ~ 00160	AI/O, counter channels' value (module with ID # 4)	Read/Write
161	32	00161 ~ 00192	AI/O, counter channels' value (module with ID # 5)	Read/Write
193	32	00193 ~ 00224	AI/O, counter channels' value (module with ID # 6)	Read/Write
225	32	00225 ~ 00256	AI/O, counter channels' value (module with ID # 7)	Read/Write
257	32	00257 ~ 00288	AI/O, counter channels' value (module with ID # 8)	Read/Write
289	32	00289 ~ 00320	AI/O, counter channels' value (module with ID # 9)	Read/Write
321	32	00321 ~ 00352	AI/O, counter channels' value (module with ID # 10)	Read/Write
353	32	00353 ~ 00384	AI/O, counter channels' value (module with ID # 11)	Read/Write
385	32	00385 ~ 00416	AI/O, counter channels' value (module with ID # 12)	Read/Write
417	32	00417 ~ 00448	AI/O, counter channels' value (module with ID # 13)	Read/Write
449	32	00449 ~ 00480	AI/O, counter channels' value (module with ID # 14)	Read/Write
481	32	00481 ~ 00512	AI/O, counter channels' value (module with ID # 15)	Read/Write
513	32	00513 ~ 00544	AI/O, counter channels' value (module with ID # 16)	Read/Write
545	32	00545 ~ 00576	AI/O, counter channels' value (module with ID # 17)	Read/Write
577	32	00577 ~ 00608	AI/O, counter channels' value (module with ID # 18)	Read/Write
609	32	00609 ~ 00600	AI/O, counter channels' value (module with ID # 19)	Read/Write
641	32	00641 ~ 00672	AI/O, counter channels' value (module with ID # 20)	Read/Write
673	32	00673 ~ 00704	AI/O, counter channels' value (module with ID # 21)	Read/Write
705	32	00705 ~ 00736	AI/O, counter channels' value (module with ID # 22)	Read/Write
737	32	00737 ~ 00768	AI/O, counter channels' value (module with ID # 23)	Read/Write
769	32	00769 ~ 00800	AI/O, counter channels' value (module with ID # 24)	Read/Write
801	32	00801 ~ 00832	AI/O, counter channels' value (module with ID # 25)	Read/Write
833	32	00833 ~ 00864	AI/O, counter channels' value (module with ID # 26)	Read/Write
865	32	00865 ~ 00896	AI/O, counter channels' value (module with ID # 27)	Read/Write
897	32	00897 ~ 00928	AI/O, counter channels' value (module with ID # 28)	Read/Write
929	32	00929 ~ 00960	AI/O, counter channels' value (module with ID # 29)	Read/Write
961	32	00961 ~ 00992	AI/O, counter channels' value (module with ID # 30)	Read/Write
993	32	00993 ~ 01024	AI/O, counter channels' value (module with ID # 31)	Read/Write

Example 1: Read channel 2 value from APAX-5040 DI module with ID number 2
ID number 2 means the Modbus address is between **00129 ~ 00192** For DI/O module, each channel occupy one 0x address (1-bit) Therefore, You can read channel 2 value from Modbus address **00131**
Example 2: Write *"True"* value to channel 5 of APAX-5046 DO module with ID number 7
ID number 7 means the Modbus address is between **00449 ~ 00512** For DI/O module, each channel occupy 1 0x address (1-bit) Therefore, You can write *"True"* value to Modbus address **00453** for channel 5

C.2 Address (4x) with MOD_SetServerHoldReg (32)

Start Address	Length	Address (4x)	Description	Attribute
1	32	00001 ~ 00032	AI/O, counter channels' value (module with ID # 0)	Read/Write
33	32	00033 ~ 00064	AI/O, counter channels' value (module with ID # 1)	Read/Write
65	32	00065 ~ 00096	AI/O, counter channels' value (module with ID # 2)	Read/Write
97	32	00097 ~ 00128	AI/O, counter channels' value (module with ID # 3)	Read/Write
129	32	00129 ~ 00160	AI/O, counter channels' value (module with ID # 4)	Read/Write
161	32	00161 ~ 00192	AI/O, counter channels' value (module with ID # 5)	Read/Write
193	32	00193 ~ 00224	AI/O, counter channels' value (module with ID # 6)	Read/Write
225	32	00225 ~ 00256	AI/O, counter channels' value (module with ID # 7)	Read/Write
257	32	00257 ~ 00288	AI/O, counter channels' value (module with ID # 8)	Read/Write
289	32	00289 ~ 00320	AI/O, counter channels' value (module with ID # 9)	Read/Write
321	32	00321 ~ 00352	AI/O, counter channels' value (module with ID # 10)	Read/Write
353	32	00353 ~ 00384	AI/O, counter channels' value (module with ID # 11)	Read/Write
385	32	00385 ~ 00416	AI/O, counter channels' value (module with ID # 12)	Read/Write
417	32	00417 ~ 00448	AI/O, counter channels' value (module with ID # 13)	Read/Write
449	32	00449 ~ 00480	AI/O, counter channels' value (module with ID # 14)	Read/Write
481	32	00481 ~ 00512	AI/O, counter channels' value (module with ID # 15)	Read/Write
513	32	00513 ~ 00544	AI/O, counter channels' value (module with ID # 16)	Read/Write
545	32	00545 ~ 00576	AI/O, counter channels' value (module with ID # 17)	Read/Write
577	32	00577 ~ 00608	AI/O, counter channels' value (module with ID # 18)	Read/Write
609	32	00609 ~ 00600	AI/O, counter channels' value (module with ID # 19)	Read/Write
641	32	00641 ~ 00672	AI/O, counter channels' value (module with ID # 20)	Read/Write
673	32	00673 ~ 00704	AI/O, counter channels' value (module with ID # 21)	Read/Write
705	32	00705 ~ 00736	AI/O, counter channels' value (module with ID # 22)	Read/Write
737	32	00737 ~ 00768	AI/O, counter channels' value (module with ID # 23)	Read/Write
769	32	00769 ~ 00800	AI/O, counter channels' value (module with ID # 24)	Read/Write
801	32	00801 ~ 00832	AI/O, counter channels' value (module with ID # 25)	Read/Write
833	32	00833 ~ 00864	AI/O, counter channels' value (module with ID # 26)	Read/Write
865	32	00865 ~ 00896	AI/O, counter channels' value (module with ID # 27)	Read/Write
897	32	00897 ~ 00928	AI/O, counter channels' value (module with ID # 28)	Read/Write
929	32	00929 ~ 00960	AI/O, counter channels' value (module with ID # 29)	Read/Write
961	32	00961 ~ 00992	AI/O, counter channels' value (module with ID # 30)	Read/Write
993	32	00993 ~ 01024	AI/O, counter channels' value (module with ID # 31)	Read/Write

Example 1: Read channel 4 value from APAX-5017 AI module with ID number 2
ID number 2 means the Modbus address (fixed mode) is between **65 ~ 96** For AI/O channel, each channel occupy one 4x address (2 byte) Therefore, You can read AI channel 4 value from Modbus address **69**
Example 2: Write analog value to channel 7 on APAX-5028 AO module with ID number 15
ID number 15 means the Modbus address (fixed mode) is between **481 ~ 512** For AI/O channel, each channel occupy one 4x address (2 byte) Therefore, You can write value to AO channel 4 value by Modbus address **488**
Example 3: Read channel 2 value from APAX-5080 Counter module with ID number 9
ID number 9 means the Modbus address (fixed mode) is between **289 ~ 320** For counter channel, each channel occupy two 4x address (4 byte) Therefore, You can read channel 2 value from Modbus address **293** and **294**

ADVANTECH

Enabling an Intelligent Planet

www.advantech.com

Please verify specifications before quoting. This guide is intended for reference purposes only.

All product specifications are subject to change without notice.