Software Guide

ICP DAS Modbus Server

(version 1.0) Version 1.0 lastupdate:2010-11-17

Warranty

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

Warning

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

Copyright

Trademark

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

License

The user can use, modify and backup this software **on a single machine.** The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.

Contents

Chapter 1 Introduction

This manual is written for modbus users of *ICPDAS LinPAC and LinCon series*.

This document will guide you:

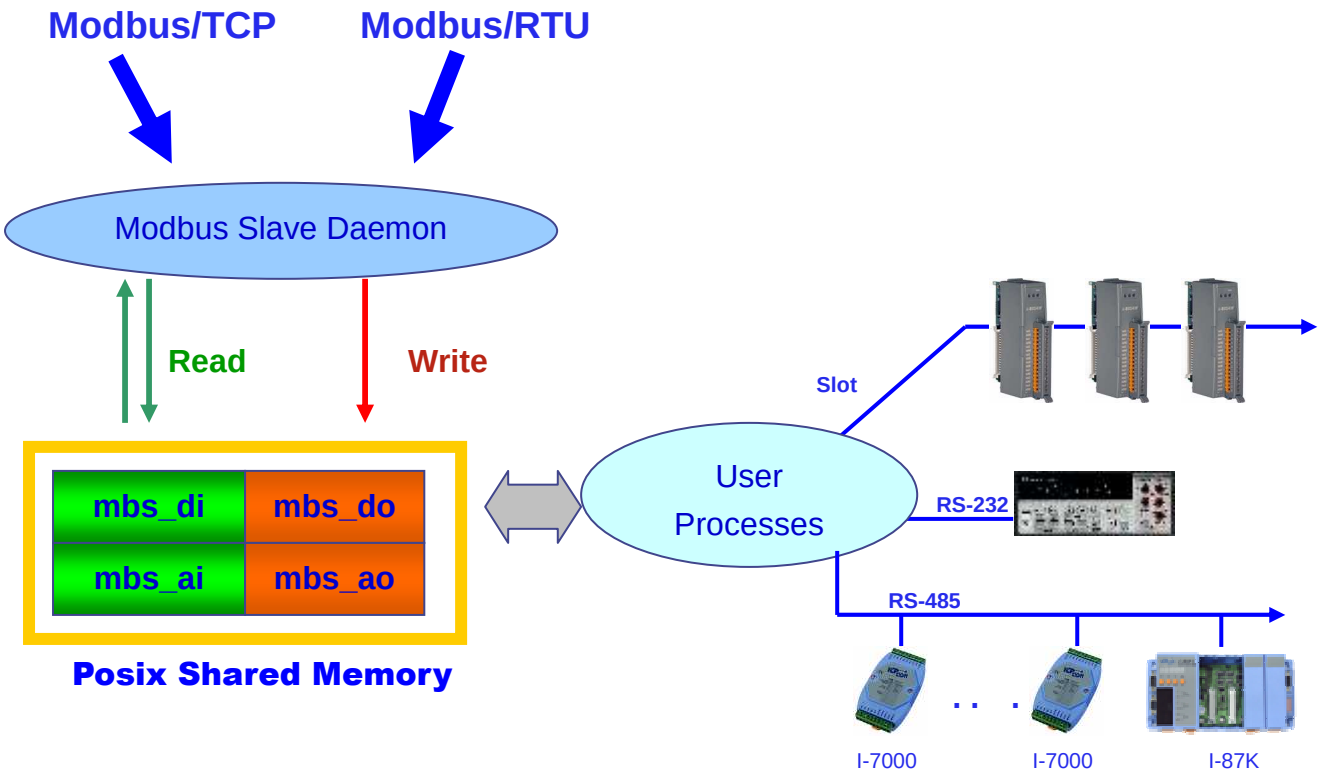
- How to setup Modbus Server on your devices.
- How to write program with shared memory

ICPDAS ® Modbus Server is a modbus slave server that provides shared memory to share data with other process.

Shared Memory is an efficient means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access. The use of shared memory in an application requires implementation of just a few interfaces bundled into the standard C language library.

ICPDAS ® Modbus Server provides POSIX shared memory mechanism. POSIX shared memory is a variation of mapped memory.

The following figure shows that Modbus Server communicating to other process via POSIX shared memory.



1.1.Install Modbus server on your LinPAC／LinCon

▶ Software Installation

If your LinPAC／LinCon is former released, you have to install the Modebus Server by manual.

Step 1.

ICPDAS® Modbus Server supports LinPAC 8x8x and 8x4x , you can download from the following links.

- LinPAC 8x8x
  ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x8x/addons/modbus/server
- LinPAC 8x4x
  ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/addons/modbus/server

Step 2.

Copy the “mbserver10” file to your LinPAC／LinCon, and change the file permission to “755” or above.

#chmod 755 ./mbserver10

Step 3.

Before executing LinPAC／LinCon Modbus Server, you have to check what’s the index number of your TTY devices.

#dmesg | grep ‘ttyS’

In general, LinCon support 2 serial ports,
ttyS0 ：COM2 RS-232 (Recommend!)
ttyS1 ：COM3 RS-485
and 3 serial ports are availabled on LinPAC.
ttyS0 ：COM1, RS232 (Reserved for console terminal)
ttyS1 ：COM3, RS232／RS-485
ttyS34 ：COM4, RS232 (Recommend!)

Step 4.

You can use following command to start Modbus Server.
./mbserver10
Modbus server provides the following arguments

-n	Set NetID , Default is 1
-p	Set Port Number for Modbus TCP,default is 502
-d	Enable Daemon mode, default is disable
-s	Set serial port device name for modubs rtu,default is /dev/ttyS34
-b	Set Baud rate ,default is 9600bps

-f	Set data format, default is 8N1

Example:

./mbserver10 –n 1 –s /dev/ttyS34 –b 115200 –f 8N1

Running automatically at boot time :

STEP 1 : Copy “mbserver” file to /etc/init.d of your LinPAC／LinCon

STEP 2 : Edit “mbserver” script file,

specify “mbserver10” directory and modify parameters to fit your system.

STEP 3 : Creat a symbolic link in /etc/rc2.d directory

ln –s /etc/init.d/mbserver /etc/rc2.d/S99MBServer

NOTE :

Modbus server must be executed after serial port initialization.

▶ Hardware Connection

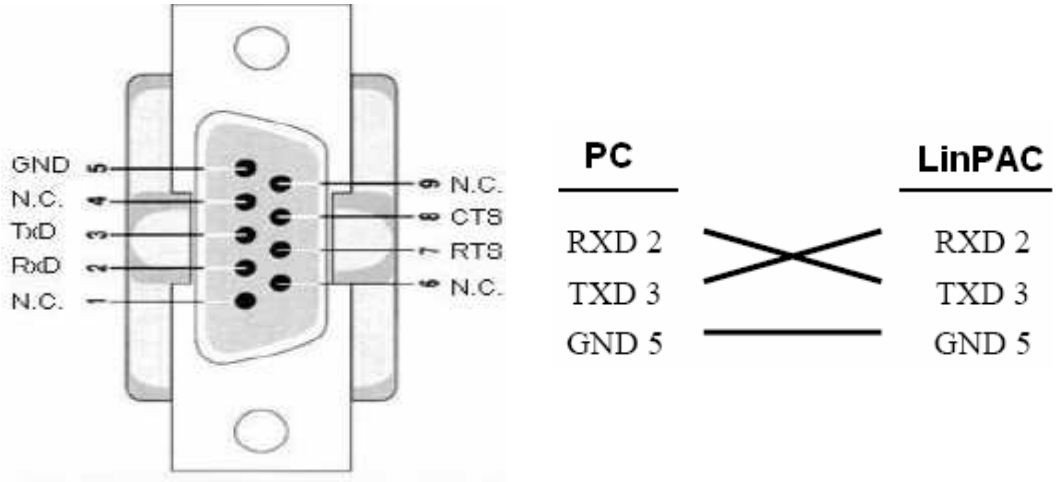
LinPAC／LinCon Modbus Server supports Modbus RTU and Modbus TCP protocol.

For TCP protocol

You can connect modbus client and server by using the ethernet cable with RJ-45 connector.

For RTU protocol

9 pins D-sub cable is necessary. The pin assignment is :



Ordering Information :

CA-0910F：9-pin Female-Female D-sub cable, 1M Cable

Chapter 2 Using shared memeoy in you program

2.1 Using Posix Shared memory

When you start the modbus server, it will allocate four shared memory on LinPAC / LinCon. Shared memory are located at `/dev/shm/mbs_ao`, `/dev/shm/mbs_ai` , `/dev/shm/mbs_do` and `/dev/shm/mbs_di` .

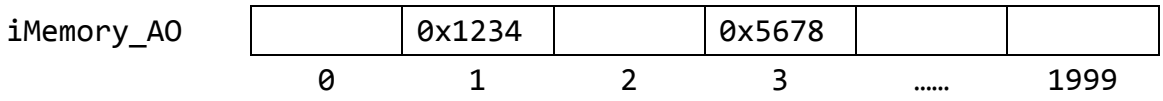
Shared memory	Size	Use
/dev/shm/mbs_ao	2000 * sizeof(int)	AO
/dev/shm/mbs_ai	2000 * sizeof(int)	AI
/dev/shm/mbs_do	2000 * sizeof(int)	DO
/dev/shm/mbs_di	2000 * sizeof(int)	DI

POSIX provides a standardized API for using shared memory. POSIX Shared Memory uses the function `shm_open` from `sys/mman.h`.
(http://www.opengroup.org/onlinepubs/009695399/basedefs/sys/mman.h.html)
If you never used POSIX shared memory, we also provide a simple MACRO `SHM_INIT` in `icpdas_mbs.h` to help you using shared memory.

Example：

```
SHM_INIT(AO,2000);
```

It will open shared memory `/dev/shm/mbs_ao` and mapping it to `*iMemory_AO` by default.Then you can access shared memory via array (Array's range is from 0~1999).



Example:

```
iMemory_AO[1]=0x01234;          //Set value 0x01234 to address 1 of iMemory_AO
printf("%04x\n",iMemory_AO[3]); //GET value from address 3 of iMemory_AO
```

2.2 Analog I/O DEMO

2.2.1. Set AO (I-8024W)

The following graph shows how to **SET AO** value between modbus server and **setao** program via shared memory .

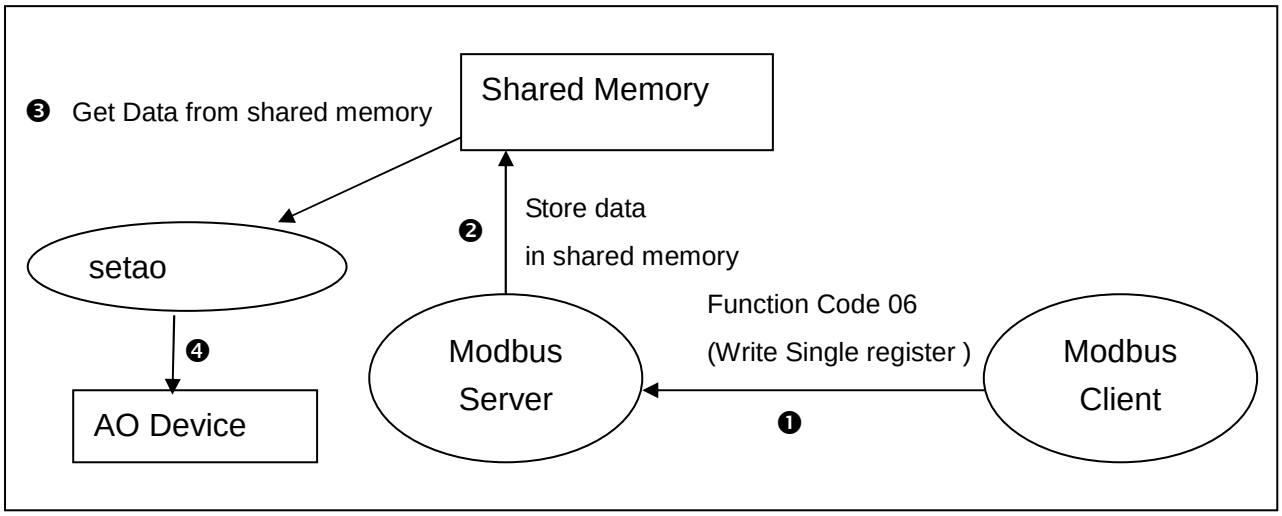


Figure 2.2.1

❶ modbus client send request to modbus server .
❷ modbus server put the AO value to shared memory.
❸ setao program fetch ao value from share memory
❹ setao program set ao value to AO device (I-8024W)

Example (setao.c)
Source code:

```
#include <stdio.h>
#include <stdlib.h>
#include "icpdas_mbs.h"
#include "msw.h"

int main( void ) {
    /* OPEN AO,AI,DO,DI Share Memory */
    SHM_INIT(AO,2000);
    int RetValue,slot,channel;
    float tmp=0.0;
    float f=0.0;
    unsigned long l;
    slot=2;
```

Include icpdas_mbs.h

Use SHM_INIT Marco to open shared memory

```
    channel=0;
    tmp=0;

    RetValue = Open_Slot(slot);
    if (RetValue >0) {
        printf("open Slot %d failed!\n",slot);
        return -1;
    }
    I8024_Initial(slot);
    while (1) {
        /* Float inverse (Big-Endian) , it depends on your SCADA */
        /*
            *((short *)&l+1)=iMemory_AI[0];
            *((short *)&l)=iMemory_AI[1];
        */
        /* Float (Little-Endian) */
        *((short *)&l)=iMemory_AO[0];
        *((short *)&l+1)=iMemory_AO[1];
        f = *((float *)&l);
        if (tmp!=f) {
            printf("set value %f\n",f);
            I8024_VoltageOut(slot,channel,f);
            tmp=f;
        }
        usleep(100);
    }
    Close_Slot(slot);
    return 0;
}
```

Analog value using floating point data type to represent. Floating point data size is 4 bytes. It uses two registers to store floating point data.

2.2.2. Get AI (I-8017HW)

The following graph shows the how to **Get AI** value between modbus server and **getai** program via shared memory .

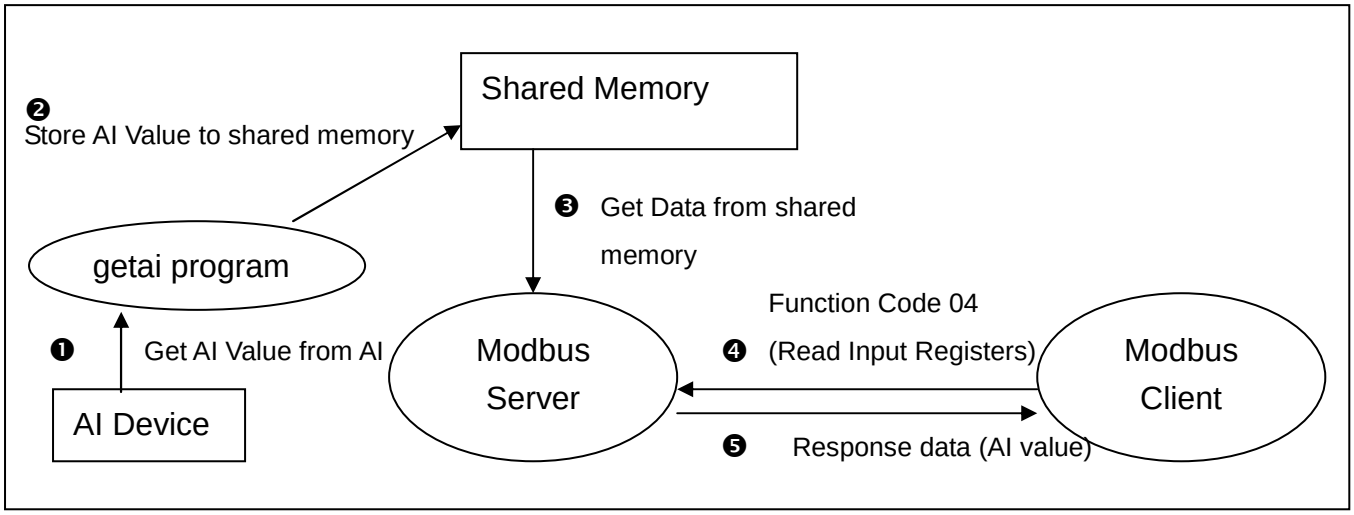


Figure 2.2.2

❶ getai program get ai value from AI Device (I-8017HW)
❷ getai program put AI value to share memory
❸ modbus server get AI value from shared memory
❹ modbus client send requeset to modbus server
❺ modbus server send response to modbus client

❶ to ❷ is Looping

Example (setao.c)
Source code:

```
#include <stdio.h>
#include "icpdas_mbs.h"
#include "msw.h"

int main(int argc, char **argv) {
    int RetValue,slot,channel,gain,mode;
    float f ;
    unsigned long l;
    slot=3;
    channel=0;
    gain=0;
    mode=0;
    /* OPEN Share Memory */
    SHM_INIT(AI,2000);
```

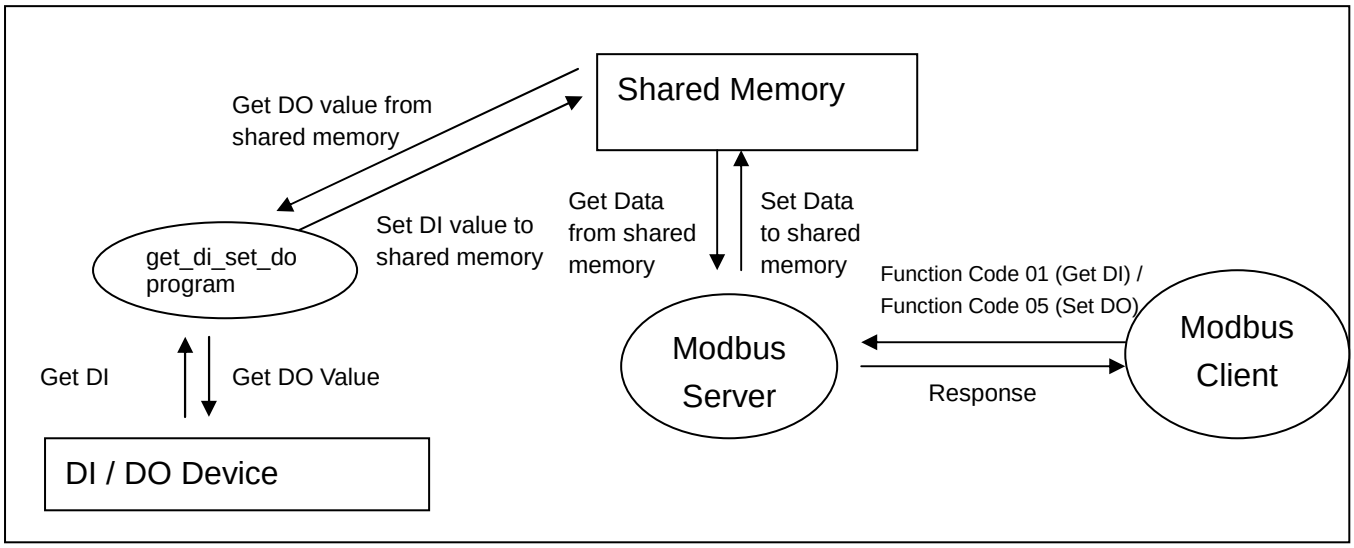
```
    /* open device file */
    RetValue = Open_Slot(slot);
    if (RetValue >0) {
       printf("open Slot %d failed!\n",slot);
       return FAILURE;
    }

    RetValue=I8017_Init(slot);
    I8017_SetChannelGainMode(slot,channel,gain,mode);
    while (1) {
       /* Get AI Value */
       f=I8017_GetCurAdChannel_Float_Cal(slot);

      l = *((unsigned long *)&f);
       /* Float inverse (Big-Endian) , depends on your SCADA */
       /* iMemory_AI[0]=*((short *)&l+1);
          iMemory_AI[1]=*((short *)&l);  */

       /* Float (Little-Endian) */
      iMemory_AI[0]=*((short *)&l);
      iMemory_AI[1]=*((short *)&l+1);
       printf("%f\n",f);
       usleep(1);
    }

    Close_Slot(slot);
    return 0;
}
```

Analog value using floating point data type to represent. Floating point data size is 4 bytes.
It use two register to store floating point data.

2.3 Digital I/O Demo

2.3.1 Set DO and Read DI (i-87054W)

The following graph shows the how to **Get AI** and **Set AO** value between modbus server and **get_di_set_do** program via shared memory .



Figrue 2.3.1

> i-87054w provides both DO and DI function

Example (get_di_set_do.c)

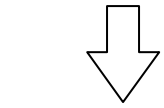
Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include "icpdas_mbs.h"
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

void change_bit(int *val, int num, char bitval) {
    *val=((*val & ~(1<<num)) | (bitval << num));
}
```

```
void set_di(char b) {
    unsigned char mask = 0x01;
    int i;
    for (i=0; i<8; ++i) {
        if ((b & mask) != 0) {
            iMemory_DI[i]=1;
        } else {
            iMemory_DI[i]=0;
        }
        mask <<= 1;
    }
}

int main() {
    int i, wRetVal;
    int tval=0;
    int tval2=0;

    SHM_INIT(DO,2000);
    SHM_INIT(DI,2000);
    //Check Open_Slot
    wRetVal = Open_Slot(0);
    if (wRetVal > 0) {
        printf("open Slot failed!\n");
        return (-1);
    }

    //Check Open_Com1
    wRetVal = Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }
```

EX:

DI value is **21(10101000)**



1	0	1	0	1	0	0	0
0	1	2	3	4	5	6	7

iMemory_DI

```
//Choose Slot3
ChangeToSlot(4);
while (1) {
    for (i=0;i<8;i++) {
        if (iMemory_DO[i]) {
            change_bit(&tval,i,1);
        } else {
            change_bit(&tval,i,0);
        }
    }

    if (tval != tval2 ) {
        //--- digital output ----  **(DigitalOut_87K()**)
        dwBuf[0] = 1;                // COM Port
        dwBuf[1] = 00;              // Address
        dwBuf[2] = 0x87054;         // ID
        dwBuf[3] = 0;               // CheckSum disable
        dwBuf[4] = 100;             // TimeOut , 100 msecond
        dwBuf[5] = tval;            // digital output
        dwBuf[6] = 0;               // string debug
        wRetVal = DigitalOut_87K(dwBuf, fBuf, szSend, szReceive);
// DO Output
        //  printf("DO Value= %u\n", dwBuf[5]);
        tval2=tval;
    }
    usleep(100);
    //--- digital Input ----   **(DigitalIn_87K()**)
    dwBuf[0] = 1;                 // COM Port
    dwBuf[1] = 00;               // Address
    dwBuf[2] = 0x87054;      // ID
    dwBuf[3] = 0;                 // CheckSum disable
    dwBuf[4] = 100;           // TimeOut , 100 msecond
    dwBuf[6] = 0;                 // string debug
    DigitalIn_87K(dwBuf, fBuf, szSend, szReceive);   // DI Input
    //get di value and put value to share memory
    set_di(dwBuf[5]);
    // printf("DI Value= %d\n",dwBuf[5]);
    usleep(100);
}
Close_Com(COM1);
```

Get DO value from shared memory(iMemory_DO[0]~iMemory[7]) and save to variable tval

Get DI value and Set its value to Shared Memory

```
    Close_SlotAll();

    return 0;
}
```

2.4 Download Demo Files

You can download example file from

- ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x8x/addons/modbus/demo
- ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/addons/modbus/demo