PIO-D48

Digital I/O Card

Linux Software Manual

Warranty

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

Warning

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

Copyright

Trademark

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

Tables of Content

1. Linux Software Installation

The PIO-D48 can be used in linux kernel 2.4.X to 4.15.X. For Linux O.S, the recommended installation and uninstall steps are given in Sec 1.1 ~ 1.2

1.1 Linux Driver Installing Procedure

Step 1: Copy the linux driver “ixpio.tar.gz” in the directory “NAPDOS\Linux” of the companion CD or download the latest driver from our website to the linux host.

Step 2: You must use the ‘**root**’ identity to compile and install PIO/PISO linux driver.

Step 3: Decompress the tarball “ixpio.tar.gz”.

Step 4: Type ‘**cd**' to the directory containing the package's source code, and ’type ‘**./configure**' to configure the package for your linux system.

Step 5: Type ‘**make**' to compile the package.

Step 6: You can type ‘**./ixpio.inst**' to install the PIO/PISO driver module and build the device file “ixpioX” in the device directory “/dev” automatically.

1.2 Linux Driver Uninstalling Procedure

Step 1: Type `cd' to the directory containing the package's source code.

Step 2: Type `./ixpio.remove' to remove the PIO/PISO driver module.

2. Static Library Function Description

The static library is the collection of function calls of the PIO-DIO cards for linux kernel 2.4.x and 2.6.x to 4.15.x system. The application structure is presented as following figure. The user application program developed by C (C++) language can call library “libpio.a” in user mode. And then static library will call the module ixpio to access the hardware system.

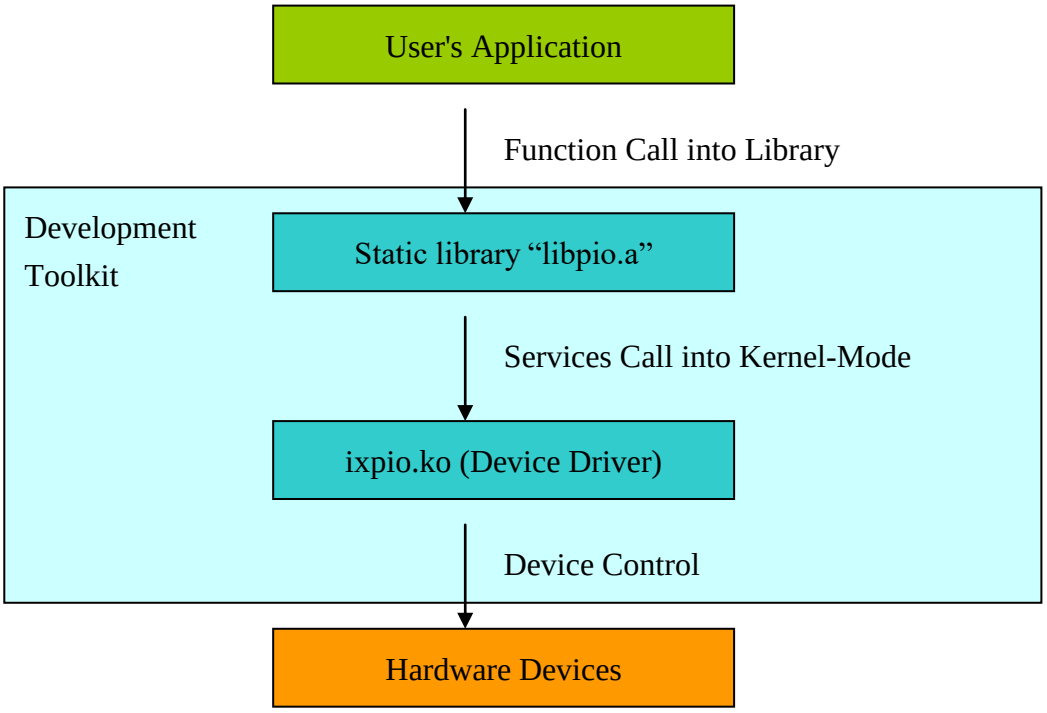

Figure 2.1

2.1 Table of ErrorCode and ErrorString

Table 2.1

Error Code	Error ID	Error String
0	PIODA_NOERROR	OK (No error !)
1	PIODA_MODULE_NAME_GET_ERROR	Module name can't get from file /proc/ixpio/ixpio
2	PIODA_DEVICE_DIO_INIT_ERROR	Configure port DI/O error
3	PIODA_ACTIVE_PORT_ERROR	Select I/O port error
4	PIODA_PORT_DEFINED_ERROR	Port number out of range
5	PIODA_DIGITAL_OUTPUT_ERROR	Digital output error
6	PIODA_DIGITAL_INPUT_ERROR	Digital input error
7	PIODA_INT_SOURCE_DEFINED_ERROR	Interrupt source defined error
8	PIODA_CONFIGURE_INTERRUPT_ERROR	Configure interrupt error
9	PIODA_ACTIVEMODE_DEFINED_ERROR	Defined Interrupt active mode error
10	PIODA_ADD_SIGNAL_ERROR	Add signal condition error
22	PIODA_LIBRARY_PARAMETER_ERROR	Set the number of parameters error

2.2 Function Descriptions

Table 2.2

Function Definition
char * PIODA_GetDriverVersion(void);
char * PIODA_GetLibraryVersion(void);
int PIODA_Open(char *dev_file);
WORD PIODA_Close(WORD fd);
WORD PIODA_DriverInit(WORD);
WORD PIODA_PortDirCfs(WORD, WORD, boolean);
WORD PIODA_Digital_Output(WORD, WORD, byte);
WORD PIODA_Digital_Input(WORD, WORD, WORD *);
WORD PIOD48_8254Clk(WORD, WORD);
WORD PIODA_8254CW(WORD, WORD, WORD, WORD, WORD);

WORD PIODA_8254C0(WORD, WORD, WORD);
WORD PIODA_8254C1(WORD, WORD, WORD);
WORD PIODA_8254C2(WORD, WORD, WORD);
WORD PIOD48_INT01_CONTROL(WORD , WORD, WORD);
WORD PIODA_IntInstall(WORD, HANDLE, WORD, WORD, WORD);
WORD PIODA_IntRemove(WORD, WORD);

2.3 Digital I/O FUNCTIONS

2.3.1 PIODA_GetDriverVersion

- **Description:**
  To show the version number of PIO/PISO linux driver.
- **Syntax:**
  char * PIODIO_GetDriverVersion(Void)
- **Parameter:**
  None
- **Return:**
  The version of PIO/PISO linux driver.

2.3.2 PIODA_GetLibraryVersion

- **Description:**
  To show the version number of PIO/PISO linux static library.
- **Syntax:**
  char * PIODIO_GetLibraryVersion(void)
- **Parameter:**
  None
- **Return:**
  PIO/PISO linux static library version.

2.3.3 PIODA_Open

- **Description:**
  To open device file.
- **Syntax:**
  int PIODIO_Open(char *dev_file)
- **Parameter:**
  dev_file : The path of device file
- **Return:**
  The file descriptor of device file. If the file descriptor < 0, it means that open device file failure.

2.3.4 PIODA_Close

- **Description :**
  To close device file.
- **Syntax :**
  Word PIODIO_Close(WORD fd)
- **Parameter :**
  fd : The file descriptor of device file that get from function PIODIO_Open
- **Return:**
  “PIODA_NOERROR”
  (Please refer to "Section 2.1 Error Code")

2.3.5 PIODA_DriverInit

- **Description :**
  To allocates the computer resource for the device. This function must be called once before applying other PIODA functions.
- **Syntax :**
  WORD PIODA_DriverInit(WORD fd)
- **Parameter :**
  fd : The file descriptor of device file that get from function PIODIO_Open
- **Return:**
  “PIODA_MODULE_NAME_GET_ERROR”
  “PIODA_NOERROR”
  (Please refer to "Section 2.1 Error Code")

2.3.6 PIODA_PortDirCfs

- **Description :**
  To change Digital I/O port. status(DI or DO)
- **Syntax :**
  WORD PIODA_PortDirCfs(WORD fd, WORD port, boolean io)
- **Parameter :**
  fd : The file descriptor of device file that get from function PIODIO_Open.
  port : The port number that want to change status(DI or DO)
  PIOD48_P0: port0
  PIOD48_P1: port1

PIOD48_P2: port2
PIOD48_P2_HB: port2 high nibble
PIOD48_P2_LB: port2 low nibble
PIOD48_P3: port3
PIOD48_P4: port4
PIOD48_P5: port5
PIOD48_P5_HB: port5 high nibble
PIOD48_P5_LB: port5 low nibble

io : The value 0 means digital output. The value 1 means digital input.

- **Return:**

"PIODA_DEVICE_DIO_INIT_ERROR"
"PIODA_NOERROR"
(Please refer to "Section 2.1 Error Code")

2.3.7 PIODA_Digital_Output

- **Description :**

This subroutine sends the 8 bits data to the specified I/O port.

- **Syntax :**

WORD PIODA_Digital_Output(WORD fd, WORD port, byte data);

- **Parameter :**

fd : The file descriptor of device file that get from function PIODIO_Open.

port : The output port number.
PIOD48_P0: port0
PIOD48_P1: port1
PIOD48_P2: port2
PIOD48_P2_HB: port2 high nibble
PIOD48_P2_LB: port2 low nibble
PIOD48_P3: port3
PIOD48_P4: port4
PIOD48_P5: port5
PIOD48_P5_HB: port5 high nibble
PIOD48_P5_LB: port5 low nibble

data : 8 bits data.

- **Return:**

"PIODA_DIGITAL_OUTPUT_ERROR"
"PIODA_NOERROR"

(Please refer to "Section 2.1 Error Code")

2.3.8 PIODA_Digital_Input

- **Description :**

  This subroutine reads the 8 bits data from the specified I/O port.

- **Syntax :**

  WORD PIODA_Digital_Input(WORD fd, WORD port, WORD *di_data);

- **Parameter :**

  fd : The file descriptor of device file that get from function PIODIO_Open.

  port : The input port number.

  PIOD48_P0: port0
  PIOD48_P1: port1
  PIOD48_P2: port2
  PIOD48_P2_HB: port2 high nibble
  PIOD48_P2_LB: port2 low nibble
  PIOD48_P3: port3
  PIOD48_P4: port4
  PIOD48_P5: port5
  PIOD48_P5_HB: port5 high nibble
  PIOD48_P5_LB: port5 low nibble

  di_data : A variable address used to storage the 8 bits input data.

- **Return:**

  “PIODA_DIGITAL_INPUT_ERROR“
  “PIODA_NOERROR”
  (Please refer to "Section 2.1 Error Code")

2.3.9 PIOD48_8254Clk

- **Description :**

  8254 Timer source CLK1 selection.

- **Syntax :**

  WORD PIOD48_8254Clk(WORD fd, WORD boolean);

- **Parameter :**

  fd : The file descriptor of device file that get from function PIODIO_Open.

  boolean :Select timer frequency
  0 : 2 MHz

1 : 32.768 KHz.

- **Return:**

  “PIODA_NOERROR”

  (Please refer to "Section 2.1 Error Code")

2.3.10 PIODA_8254CW

- **Description :**

  Set 8254 control word.

- **Syntax :**

  WORD PIODA_8254CW(WORD, WORD, WORD, WORD, WORD);

- **Parameter :**

  fd : The file descriptor of device file that get from function PIODIO_Open.

  bcd : Set binary or BCD count
  - 0: binary count
  - 1: BCD count

  mode : Select mode
  - 0: Interrupt on terminal count
  - 1: Programmable one-shot
  - 2: Rate generator
  - 3: Square-wave generator
  - 4: software triggered pulse
  - 5: Hardware triggered pulse

  r1 :Select counter read/write order
  - 0: Counter latch instruction
  - 1: Read/write low counter byte only
  - 2: Read/write high counter byte only
  - 3: Read/write low counter byte first, then high counter

  sc :Select counter
  - 0: Counter0
  - 1: Counter1
  - 2: Counter2
  - 3: Readback command

- **Return:**

  “PIODA_LIBRARY_PARAMETER_ERROR”

  “PIODA_NOERROR”

  (Please refer to "Section 2.1 Error Code")

2.3.11 PIODA_8254C0

- **Description :**
  Read/Write 8254 Counter0
- **Syntax :**
  WORD PIODA_8254C0(WORD fd, WORD hbyte, WORD lbyte);
- **Parameter :**
  fd : The file descriptor of device file that get from function PIODIO_Open.
  hbyte : Counter0 high byte
  lbyte : Counter0 low byte
- **Return:**
  "PIODA_NOERROR"
  (Please refer to "Section 2.1 Error Code")

2.3.12 PIODA_8254C1

- **Description :**
  Read/Write 8254 Counter1
- **Syntax :**
  WORD PIODA_8254C1(WORD fd, WORD hbyte, WORD lbyte);
- **Parameter :**
  fd : The file descriptor of device file that get from function PIODIO_Open.
  hbyte : Counter1 high byte
  lbyte : Counter1 low byte
- **Return:**
  "PIODA_NOERROR"
  (Please refer to "Section 2.1 Error Code")

2.3.13 PIODA_8254C2

- **Description :**
  Read/Write 8254 Counter2
- **Syntax :**
  WORD PIODA_8254C2(WORD fd, WORD hbyte, WORD lbyte);
- **Parameter :**
  fd : The file descriptor of device file that get from function

PIODIO_Open.

Hbyte : Counter1 high byte

lbyte : Counter1 low byte

- **Return:**

  “PIODA_NOERROR”

  (Please refer to "Section 2.1 Error Code")

2.3.14 PIOD48_INT01_CONTROL

- **Description :**

  INT_CHAN_0 and INT_CHAN_1 Interrupt source select

- **Syntax :**

  WORD PIOD48_INT01_CONTROL(WORD fd, WORD Int, WORD opt);

- **Parameter :**

  fd : The file descriptor of device file that get from function PIODIO_Open.

  Int : 0: INT_CHAN_0
  1: INT_CHAN_1

  opt :
  0: INT_CHAN_0=PC3 & !PC7 of port-2
  1: disable PC3 & !PC7 (of port-2) as interrupt source
  2: INT_CHAN_0=PC3 of port-2

- **Return:**

  “PIODA_LIBRARY_PARAMETER_ERROR”

  “PIODA_DIGITAL_OUTPUT_ERROR”

  “PIODA_NOERROR”

  (Please refer to "Section 2.1 Error Code")

2.3.15 PIODA_IntInstall

- **Description :**

  This subroutine installs the IRQ service routine.

- **Syntax :**

  WORD PIODA_IntInstall(WORD fd, HANDLE hisr, WORD signal, WORD int_source, WORD activemode);

- **Parameter :**

  fd :The file descriptor of device file that get from function PIODIO_Open.

  hisr : Address of a Event handle. The handle function will be

called when the interrupt happened.

signal : The number of signal is defined by user.

Int_source : Please refer to the following table 2.3.

Table 2.3

Card No.	Int_source	Description
PIO-D96	PIOD48_INT0	Enable P2C0
	PIOD48_INT1	Enable P2C1
	PIOD48_INT2	Enable P2C2
	PIOD48_INT3	Enable P2C3
	PIOD48_ALL_INT	Enable All Int Source

activemode : The value 0 means interrupt happened when signal is low.
The value 1 means interrupt happened when signal is high.
The value 2 means interrupt happened when signal is low or high.

- **Return:**

"PIODA_CONFIGURE_INTERRUPT_ERROR"
"PIODA_INT_SOURCE_DEFINED_ERROR"
"PIODA_ACTIVEMODE_DEFINED_ERROR"
"PIODA_ADD_SIGNAL_ERROR"
"PIODA_NOERROR"
(Please refer to "Section 2.1 Error Code")

2.3.16 PIODA_IntRemove

- **Description :**

This subroutine removes the IRQ service routine.

- **Syntax :**

WORD PIODA_IntRemove(WORD fd, WORD sig_id)

- **Parameter :**

fd : The file descriptor of device file that get from function PIODIO_Open.

sig_id : The number of signal is defined by user.

- **Return:**

"PIODA_CONFIGURE_INTERRUPT_ERROR",
"PIODA_NOERROR".
(Please refer to "Section 2.1 Error Code")

3. PIO-D48 Linux Demo

All of demo programs will not work normally if PIO/PISO linux driver would not be installed correctly. During the installation process of PIO/PISO linux driver, the install-scripts “ixpio.inst” will setup the correct kernel driver. After driver (version 0.23.0 or the later driver version) compiled and installation, the related demo programs, development library and declaration header files for different development environments are presented as follows.

Table 3.1

Driver Name	Directory Path	File Name	Description
ixpio	Include	piodio.h	PIO/PISO library header
	lib	libpio.a libpio_64.a	PIO/PISO static library
	examples/ piod48_pexd48	port.c	Digital input and output demo
		port_a.c	DI and DO demo with Library
		int.c	Interrupt demo
		Int_a.c	Interrupt demo with Library
		dio.c	Digital input and output demo
		counter.c	
		counter_a.c	

3.1 Demo code “port.c”

This demo program is used to output data from port1 (PB) and read data from port0 (PA).

In Figure 3.1, use PA_0 to test. When PA_0 read DO value, then present 0x1, else is 0.

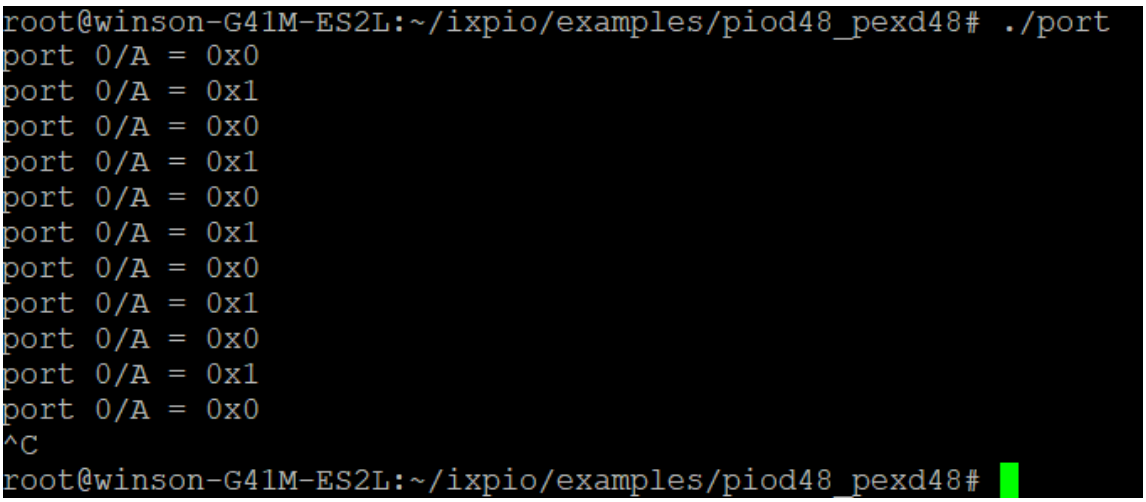

Figure 3.1

This demo end condition is PA_7 on, like Figure 3.2.

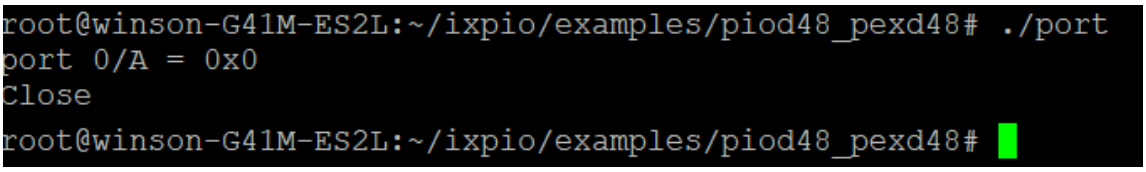

Figure 3.2

3.2 Demo code “port_a.c”

This demo is using the static library “libpio.a”. It is used to output digital from port 0 and read data from port 1.

In Figure 3.3 We connect PA_0 to PC_7. So, when PA_0 is output value, PC will get input value 0x80 (PC_7).

```
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48# ./porta
Set port0 as DO and port2 as DI
Enter to continue, Esc to exit.

Port2 input = 0x80. Port0 output = 0x1.

Port2 input = 0x0. Port0 output = 0x2.

Port2 input = 0x0. Port0 output = 0x4.

Port2 input = 0x0. Port0 output = 0x8.

Port2 input = 0x0. Port0 output = 0x10.

Port2 input = 0x0. Port0 output = 0x20.

Port2 input = 0x0. Port0 output = 0x40.

Port2 input = 0x0. Port0 output = 0x80.

Port2 input = 0x80. Port0 output = 0x1.

Port2 input = 0x0. Port0 output = 0x2.
```

Figure 3.3

3.3 Demo code “int.c”

This demo program uses INT_CHAN0 as interrupt source. The interrupt will be triggered according to the argument “sig.bedge”(if value = 1, then the interrupt triggered at high signal and low signal) or “sig.edge”(if value = 1, then interrupt triggered at high signal. Otherwise, if “sig.edge” = 0, the interrupt triggered at low signal).

```
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48# ./int
press <enter> to exit

Got single 34 for 3237 times

End of program.
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48#
```

Figure 3.4

3.4 Demo code “int_a.c”

This demo program coded by using the static library “libpio.a” to enable INT_CHAN0, as interrupt source. The interrupt will be triggered by external trigger PC_3.

```
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48# ./inta
 Enable INT_CHAN0, external trigger by port2 PC3, signaling for both edge.
press <enter> to exit

Got single 34 for 2249 times

End of program.
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48#
```

Figure 3.5

3.5 Demo code “dio.c”

This demo program configure port and test DO、DI function.

```
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48# ./dio
a. Program port0 to port5 as DI or DO port. (Default setting is DI port.)
b. Digital Output
c. Digital Input
p. Show function menu.
q. Quit this demo.
```

Figure 3.6

3.6 Demo code “counter.c”

The demo “counter.c” can be used to configure the counter and startup timer interrupt

```
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48# ./counter
time beat 1, press <enter> to exit
INT_CHAN_3 is 1 HZ
Press <enter> to exit
time beat 2, press <enter> to exit
time beat 3, press <enter> to exit
time beat 4, press <enter> to exit
time beat 5, press <enter> to exit
time beat 6, press <enter> to exit

End of program.
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48#
```

3.7 Demo code “counter_a.c”

This demo program coded by using the static library “libpio.a” to configure the counter and startup timer interrupt

```
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48# ./countera
INT_CHAN_3 is 1 HZ
press <enter> to exit

time beat 1, press <enter> to exit
time beat 2, press <enter> to exit
time beat 3, press <enter> to exit
time beat 4, press <enter> to exit
time beat 5, press <enter> to exit


End of program.
root@winson-G41M-ES2L:~/ixpio/examples/piod48_pexd48#
```