

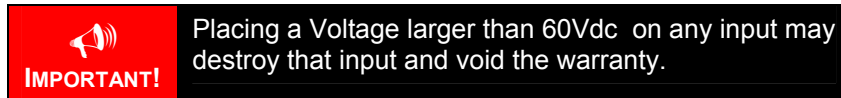
Table of Contents

General Information	3
I/O	3
Figure 1 – Output Configuration	4
Electrical Description	5
Digital Inputs	5
Digital Outputs	5
Physical Description	6
Dimensions	6
Figure 2 - RP0000xxxxN Version 1.00	7
Figure 3 - RP000000xxN Version 1.00	8
Pinout	9
Warranty	9
Installation	9
Software	9
Functions in RP2005.DLL	10
RP_ListDIO	10
RP_OpenDIO	11
RP_CloseDIO	11
RP_ReadPort	12
RP_ReadAll	13
RP_WritePort	14
RP_WriteAll	16
RP_GetVer	17

The RP0000xxxxN has 8/16/24/32 buffered Digital Inputs and 8/16/24/32 buffered Digital Outputs. The RP000000xxN has 16/32/48/64 buffered Digital Outputs. The unit is controlled through a USB connection. The board is bus powered and will draw a maximum of 125 ma from the USB Bus. While not required, it is recommended that the RP0000xxxxN be connected directly to a computer's USB port or to a port of a powered USB hub. Reading and writing the DIO is done through a DLL (dynamic link library). Therefore most popular programming languages (C++builder, VisualC, Visual Basic, NI's Measurement Studio, etc.) will be able to access the routines needed to control the DIO board. The board can also be accessed from an action step in NI's TestStand using the DLL Flexible Prototype Adapter.

I/O

The digital inputs are optically isolated transistors. Each input has a 7.5K 1 Watt current limiting resistor in series with the optoisolator. The inputs can sense voltages from 3 to 60 VDC. Higher voltages can be sensed by adding another resistor in series with the input. Each input has a clearly marked LED located on the board to indicate when a voltage is sensed.



See figures 2-6 for the pinouts for these models.

The output ports are MOSFETS capable of switching voltages between 6 and 48 VDC. Each output is configured as a current sink to ground. The outputs are able to switch up to 2 Amps at 48 VDC. Each output port is configured as below. Please note that each output port must have between 6 and 48 VDC on pin 1 of the connector in order to work properly. Each output has a clearly marked LED located on the board to indicate when that output is sinking current.

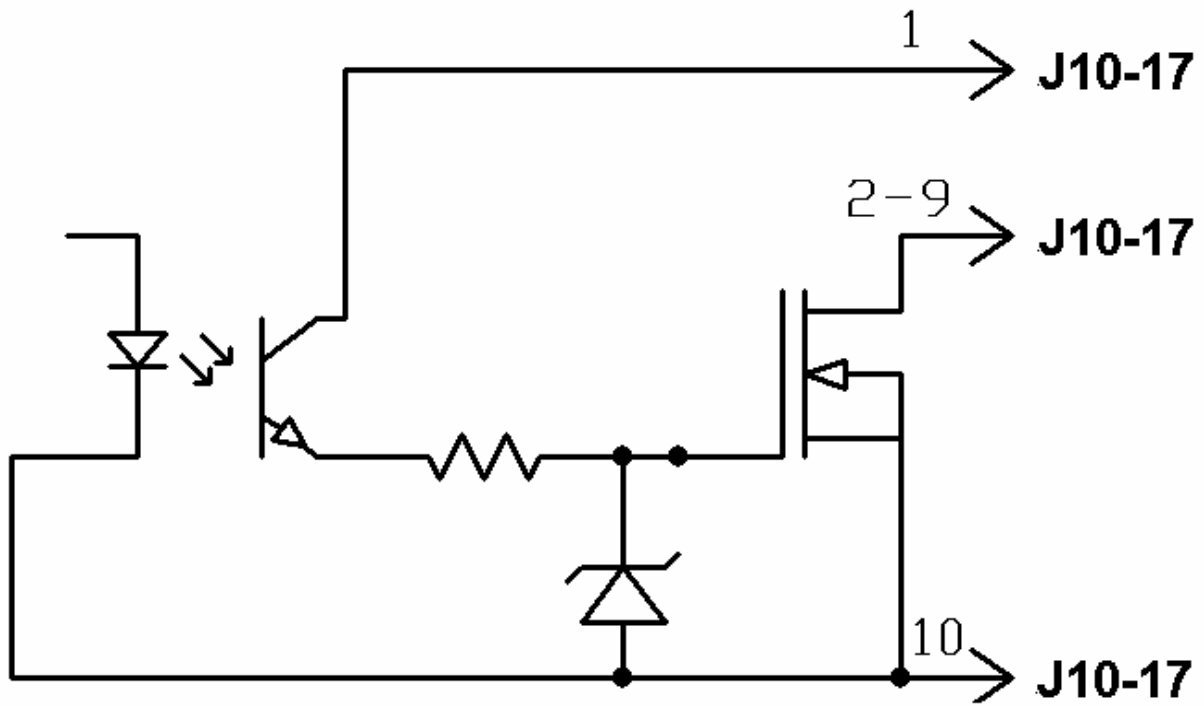


Figure 1 – Output Configuration

IMPORTANT! Placing a load larger than 48Vdc at over 2 Amps on an output may destroy that output and void the warranty.

Digital Inputs

Absolute Maximum Ratings		
Parameter	Value	Units
Total Device Power Dissipation @ 25C	480	mW
Forward Current (DC)	50	mA
Peak Forward Current	1	A
LED Power Dissipation @ 25C	80	mW

Electrical Characteristics				
Parameter	Min	Typ	Max	Units
Input Forward Voltage (Forward Current = 5 mA)		1.1	1.4	V
Terminal Capacitance (V = 0V, f = 1.0Mhz)		15		pF
Input Output Isolation Voltage			2500	V (rms)
Rise Time		200		uS
Fall Time		200		uS

Digital Outputs

Electrical Characteristics				
Parameter	Min	Typ	Max	Units
Drain Source Voltage - Vds			50	V
Continuous Drain Current - Id			2	Amps
Drain Source On State Resistance - Rds		.09	.12	Ohm
Turn On Time		25	40	nS
Turn Off Time		25	40	nS

Dimensions

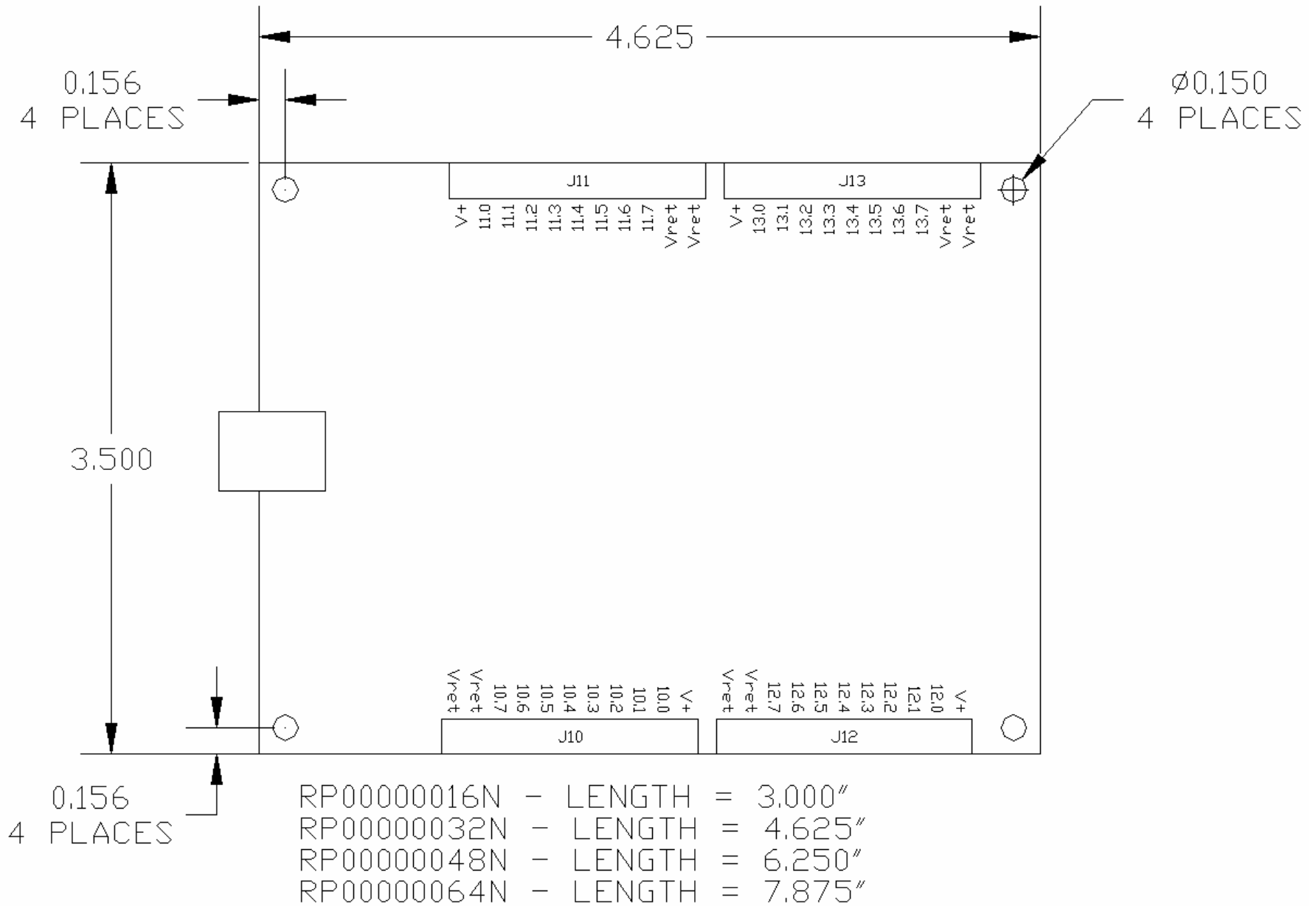


Figure 3 - RP000000xxN Version 1.00

Pinout

The DIO connectors for the RP0000xxxxN are screw clamp terminal blocks. The input connectors are labeled J0 – J4. The output connectors are labeled J10-J17. Figures 2-3 shows the pinouts for each connector.

Warranty

The RP0000xxxxN is warranted for 1 year. If within the first year of ownership the RP0000xxxxN fails while being used within the specifications of the board it will be replaced with a new one. The user will be responsible for shipping the old board back to BCS. If it is determined that the board has been misused in any way the warranty will be void.

Installation

Install the RP0000xxxxN as follows:

1. Plug the RP0000xxxxN into a computer's USB Port or a powered USB Hub. The operating system will acknowledge new hardware.
2. When prompted, browse the New Hardware Wizard to the subdirectory **USB_Driver** on the CDROM.
3. Select the file **FTD2xx.inf**. The operating system will then load the necessary files for the RP0000xxxxN to work on the computer. The system will acknowledge the installation of the new hardware.
4. To finish installation, browse to the **Root** directory on the CDROM. Run the program called **Setup.exe**. This will install the code example, documentation and an executable for testing the functionality of the RP0000xxxxN.

Software

There are 3 files included to assist in using RP0000xxxxN. They are RP2005.dll, FTD2xx.dll and RP2005.lib.

FTD2xx.dll – This file is used by RP2005.dll and needs to reside on the machine in order to communicate with the digital IO. The file will be placed in the System32 subdirectory by the setup program located on the CDROM.

RP2005.dll – This file contains all the functions needed to use the RP0000xxxxN. A brief description of each function can be found below. Sample code is also supplied and can be found at <Install Dir>\BCS\RP2005\Software\Sample Code\. The default for <Install Dir> is C:\Program Files\. This file can be included in a software project or it can be used directly by a test executive such as National Instruments TestStand. The file will be placed in the System32 subdirectory by the setup program located on the CDROM.

RP2005.lib – This file is used by the software project that will be created to use the RP0000xxxxN. RP2005.dll was written using Visual C/C++ therefore the supplied import library file will work with Microsoft programming products. Other development environments like National Instruments Measurement Studio or Boland C++Builder will not be able to use this import library. Both products have an easy to use utility for creating a

compatible import library. The file should be placed in the software project's folder. Include this file in the project. Sample code has been supplied to show how to use the functions in RP2005.dll. The file will be placed in <Install Dir>\BCS\RP2005\Software\Sample Code\. The default for <Install Dir> is C:\Program Files\.

Functions in RP2005.DLL

The following functions are available in RP2005.DLL.

RP_ListDIO

Get information concerning the devices currently connected. This function returns the number of devices connected, and each device's serial number and product description.

unsigned long **RP_ListDIO** (int * *iNumBrds*, char * *SN*, char * *Desc*)

Parameters

<i>iNumDev</i>	The number of RP2005 devices currently attached to USB
<i>SN</i>	Comma delimited string containing the serial number of each RP2005 device currently attached to USB
<i>Desc</i>	Comma delimited string containing the device description of each RP2005 device currently attached to USB

Return Value

0 if successful, otherwise the return value is an error code.

Remarks

This function is used to return each device's serial number. The serial number is then used by **RP_Open** to obtain a handle for subsequent reading and writing of DIO.

Examples

Sample code shows how to get the number of devices, each serial number and each description.

```
unsigned long ulErrCode;
int iNumDevs;
char SN[256];
char Desc[256];

ulErrCode = RP_ListDIO( &iNumDevs, SN, Desc );
if (ulErrCode == 0)
{
    // Do something
}
else
{
    // Handle error
}
```

RP_OpenDIO

Open the device and return a handle which will be used for subsequent reading and writing of DIO.

unsigned long **RP_OpenDIO** (char * *SN*, unsigned long **hDIO*)

Parameters

<i>SN</i>	The serial number for the device.
<i>hDIO</i>	Pointer to a variable of type long where the handle will be stored. This handle must be used to access the device.

Return Value

0 if successful, otherwise the return value is an error code.

Remarks

Example

This sample shows how to open a device.

```
unsigned long hDIO; // handle of an open device
unsigned long ulErrCode;
char SN[256];
char Desc[256];

ulErrCode = RP_ListDIO( &numDevs, SN, Desc );
if (( ulErrCode == 0 ) && ( numDevs == 1 ))
{
    ulErrCode = RP_OpenDIO( SN, & hDIO );
    if ( ulErrCode == 0 )
    {
        // Do something
    }
    else
    {
        // Handle error
    }
}
else
{
    // Handle error
}
```

RP_CloseDIO

Close an open device.

unsigned long **RP_CloseDIO**(unsigned long *hDIO*)

Parameters

<i>hDIO</i>	Handle of the device to close.
-------------	--------------------------------

Return Value

0 if successful, otherwise the return value is an error code.

Example

This sample shows how to close a device.

```
unsigned long hDIO;
unsigned long ulErrCode;
char SN[256];
char Desc[256];

ulErrCode = RP_ListDIO( &numDevs, SN, Desc );
if ( ( ulErrCode == 0 ) && ( numDevs == 1 ) )
{
    ulErrCode = RP_OpenDIO( SN, &hDIO );
    if ( ulErrCode == 0 )
    {
        // Do something
        ulErrCode = RP_CloseDIO( hDIO );
    }
    else
    {
        // Handle error
    }
}
else
{
    // Handle error
}
```

RP_ReadPort

Read data from the device.

```
unsigned long RP_ReadPort( unsigned long hDIO, unsigned char ucPort, unsigned char * ucPVal,
                           char * ErrMsg )
```

Parameters

<i>hDIO</i>	Handle of the device to read.
<i>ucPort</i>	The number of the port to be read.
<i>ucPVal</i>	Pointer to a variable of type unsigned char which receives the value of the port.
<i>ErrMsg</i>	String containing any error messages.

Return Value

0 if successful, otherwise the return value is an error code.

Remarks

The function does not return until the requested port has been read or read timeout occurs. The read timeout is set to 1 second.

The parameter *ucPort* represents either input port 0-3 or output port 0-3. A value of 0 will access input port 0, a value of 1 will access input port 1, a value of 2 will access input port 2, a value of 3 will access input port 3, a value of 16 (0x10) will access output port 0, a value of 17 (0x11) will access output port 1, a value of 18 (0x12) will access output port 2, a value of 19 (0x13) will access output port 3. Any other value will return an error.

The parameter *ucPVal* represents the value of the requested port. A value of 0 (00000000 binary) means all 8 bits are active. A value of 255 (11111111 binary) means all 8 bits are off.

Example

This sample shows how to read bit 6 of input port 0.

```
unsigned long hDIO; // handle of an open device
unsigned long ulErrCode;
unsigned char ucPort = 0;
unsigned char ucPVal, ucBit6;
char ErrMsg[256];

    ulErrCode = RP_ReadPort(hDIO, ucPort, &ucPVal, ErrMsg);
if (ulErrCode == 0)
{
    ucBit6 = ucPVal & 0x40; // 0100 0000
    // Do something
}
else
{
    // Handle error
}
```

RP_ReadAll

Read all 8 ports of data from the device.

NOTE : This function will only work with boards that have a firmware version of 2.00 or higher.

```
unsigned long RP_ReadAll( unsigned long hDIO, unsigned long * uLIP, unsigned long * uIHP,
                          char * ErrMsg )
```

Parameters

<i>hDIO</i>	Handle of the device to read.
<i>uLIP</i>	Pointer to a variable of type unsigned long which receives the value of four of the ports.
<i>uIHP</i>	Pointer to a variable of type unsigned long which receives the value of four of the ports.
<i>ErrMsg</i>	String containing any error messages.

Return Value

0 if successful, otherwise the return value is an error code.

Remarks

The function does not return until the requested port has been read or read timeout occurs. The read timeout is set to 1 second.

The parameter *uILP* stores the port data as follows :

Bits	RP0000xxxxN	RP000000xN
24 - 31	Input Port 0	Output Port 0
16 - 23	Input Port 1	Output Port 1
8 - 15	Input Port 2	Output Port 2
0 - 7	Input Port 3	Output Port 3

The parameter *uIHP* stores the port data as follows :

Bits	RP0000xxxxN	RP000000xN
24 - 31	Output Port 0	Output Port 4
16 - 23	Output Port 1	Output Port 5
8 - 15	Output Port 2	Output Port 6
0 - 7	Output Port 3	Output Port 7

For any port a value of 0 (00000000 binary) means all 8 bits are active. A value of 255 (11111111 binary) means all 8 bits are off.

Example

This sample shows how to read bit 6 of input port 0.

```
unsigned long hDIO; // handle of an open device
unsigned long ulErrCode;
unsigned long ucLPort;
unsigned long ucHPort;
unsigned char ucBit6;
char ErrMsg[256];

ulErrCode = RP_ReadAll(hDIO, &ucLPort, &ucHPort, ErrMsg);
if (ulErrCode == 0)
{
    ucBit6 = ((ucLPort & 0x40000000) > 0) ? 1 : 0;
    // Do something
}
else
{
    // Handle error
}
```

RP_WritePort

Set bits for an output port.

unsigned long **RP_WritePort** (unsigned long *hDIO*, unsigned char *ucPort*, unsigned char *ucPVal*, char * *ErrMsg*)

Parameters

<i>hDIO</i>	Handle of the device to write.
<i>ucPort</i>	The number of the output port to be written.
<i>ucPVal</i>	The value to write to the output port.
<i>ErrMsg</i>	String containing any error messages.

Return Value

0 if successful, otherwise the return value is an error code.

Remarks

The function does not return until the requested port has been written or write timeout occurs. The write timeout is set to 1 second.

The parameter *ucPort* represents either output port 0, 1, 2 or 3. A value of 16 (0x10) will access port 0, a value of 17 (0x11) will access port 1, a value of 18 (0x12) will access port 2 and a value of 19 (0x13) will access port 3. Any other value will return an error.

The parameter *ucPVal* represents the value that the port will be set to. A value of 0 (00000000 binary) means all 8 bits are on (the mosfets are sinking current). A value of 255 (11111111 binary) means all 8 bits are off (the mosfets are not sinking current).

Example

This sample shows how to set bit 3 of output port 0.

```
unsigned long hDIO; // handle of an open device
unsigned long ulErrCode;
unsigned char ucPort = 0x10; // Output port 0
unsigned char ucPVal;
char ErrMsg[256];

    ulErrCode = RP_ReadPort(hDIO, ucPort, &ucPVal, ErrMsg);
if (ulErrCode == 0)
{
    ucPVal &= 0xf7; // 1111 0111
    ulErrCode = RP_WritePort(hDIO, ucPort, ucPVal, ErrMsg);
    if (ulErrCode == 0)
    {
        ulErrCode = RP_ReadPort(hDIO, ucPort, &ucPVal, ErrMsg);
    }
    else
    {
        // Handle error
    }
}
else
```

```
{
  // Handle error
}
```

RP_WriteAll

Set bits for all of the output ports.

NOTE : This function will only work with boards that have a firmware version of 2.00 or higher.

unsigned long **RP_WriteAll** (unsigned long *hDIO*, , unsigned long *uILP*, unsigned long *uIHP*, char * *ErrMsg*)

Parameters

<i>hDIO</i>	Handle of the device to write.
<i>uILP</i>	The value to write to four of the output ports.
<i>uIHP</i>	The value to write to four of the output ports.
<i>ErrMsg</i>	String containing any error messages.

Return Value

0 if successful, otherwise the return value is an error code.

Remarks

The function does not return until the requested port has been written or write timeout occurs. The write timeout is set to 1 second.

The parameter *uILP* stores the port data as follows :

Bits	RP0000xxxxN	RP000000xN
24 - 31	Output Port 0	Output Port 0
16 - 23	Output Port 1	Output Port 1
8 - 15	Output Port 2	Output Port 2
0 - 7	Output Port 3	Output Port 3

The parameter *uIHP* stores the port data as follows :

Bits	RP0000xxxxN	RP000000xN
24 - 31	Not Used	Output Port 4
16 - 23	Not Used	Output Port 5
8 - 15	Not Used	Output Port 6
0 - 7	Not Used	Output Port 7

For any port a value of 0 (00000000 binary) means all 8 bits are on (the mosfets are sinking current). A value of 255 (11111111 binary) means all 8 bits are off (the mosfets are not sinking current).

Example

This sample shows how to set bit 3 of output port 0.

```
unsigned long hDIO; // handle of an open device
unsigned long ulErrCode;
unsigned long ucLPort;
unsigned long ucHPort;
char ErrMsg[256];

ulErrCode = RP_ReadAll(hDIO, &ucLPort, &ucHPort, ErrMsg);
if (ulErrCode == 0)
{
    ucLPort &= 0xf7ffffff;
    ulErrCode = RP_WriteAll(hDIO, ucLPort, ucHPort, ErrMsg);
    if (ulErrCode == 0)
    {
        ulErrCode = RP_ReadAll(hDIO, &ucLPort, &ucHPort, ErrMsg);
    }
    else
    {
        // Handle error
    }
}
else
{
    // Handle error
}
```

RP_GetVer

Read software version from the device.

unsigned long **RP_GetVer** (unsigned long *hDIO*, char **SWVer*, char **ErrMsg*)

Parameters

<i>hDIO</i>	Handle of the device to write.
<i>SWVer</i>	String containing the software version of the device.
<i>ErrMsg</i>	String containing any error messages.

Return Value

0 if successful, otherwise the return value is an error code.

Remarks

The function does not return until the requested information has been returned or read timeout occurs. The read timeout is set to 1 second.

Examples

This sample shows how to read the software version.

```
unsigned long hDIO; // handle of an open device
unsigned long ulErrCode;
char ErrMsg[256];

ulErrCode = RP_GetVer( hDIO, SWVer, ErrMsg );
if (ulErrCode == 0)
{
    // Do something
}
```

```
else
{
  // Handle error
}
```



www.bcsideas.com

General Inquiries
info@bcsideas.com

Sales Information
sales@bcsideas.com

Product Support or Recommendations
support@bcsideas.com