I-8084W User Manual

Version 2.0, Jan 2014

Written by Hans Chen
Edited by Anna Huang
Revised by Jose Dai

Warranty

All products manufactured by ICP DAS are under warranty regarding defective materials for a period of one year, beginning from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for any damage resulting from 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

Trademarks

Names used in this manual are for identification purposes only and may be registered trademarks of their respective companies.

Contact Us

If you have any encounter any difficulties or questions, feel free to contact us at service@icpdas.com or service.icpdas@gmail.com. We will provide a response within two working days.

Table of Contents

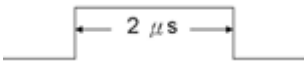
1. Introduction to the I-8084W

The I-8084W-G is a 32-bit (hardware) high-speed Counter/Frequency/Encoder IO module that provides “Up Counter”, "Frequency”, “Up/Down Counter”, “Dir/Pulse Counter” and “A/B Phase Counter” modes.

The I-8084W-G can be operated in a variety of mode combinations for all 8 channels, meaning that it some channels can be used as an “Up Counter” and others as a “Frequency” or “A/B Phase” module.

The module is suitable for counter measurement, frequency measurement and encoder motion control applications.

1.1. Specifications

Digital Input	
Mode	4-channel Up/Down Counter (Up/Down)
	4-channel Dir/Pulse Counter (Bi-direction)
	4-channel Quadrant Counting
	8-channel Up Counter
	8-channel Frequency
	Programmable Digital Noise Filter: 1 to 32737 µs
Isolated Input Level	Logic Level 0: +1 V Max.
	Logic Level 1: +4.5 to +30 V
TTL Input Level	Logic Level 0: 0 to +0.8 V
	Logic Level 1: 2 to +5 V
Minimum Pulse Width	2µs 
Input Frequency	**1 Hz ~ (typically) 250 kHz** for both counter and frequency mode, where 250 kHz is calculated as follows: supposed that the duty cycle = 50%, by referring to the Minimum Pulse Duration of the High Level, the pulse period will be 2 µs x 2 = 4 µs, which is 250 kHz as a maximum. **Maximum Frequency**: Refer to the Minimum Pulse Duration of the High Level, the maximum frequency is highly affected by the duty cycle **Frequency Accuracy** = ±0.4%
EEPROM	128 KB
Isolated Voltage	1000 Vrms
ESD Protection	2 kV (Contact for each Channel)

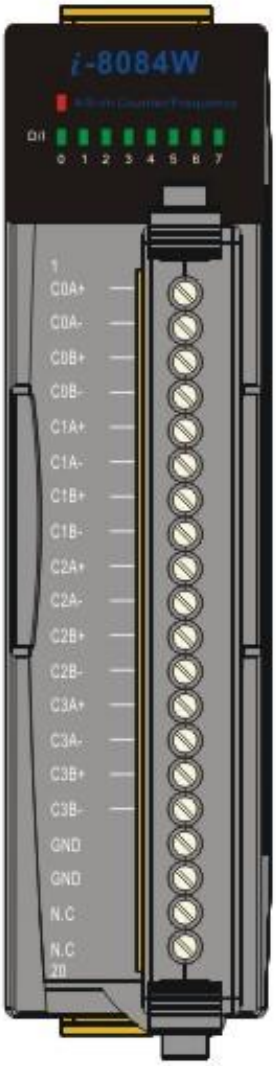
LED Display
1 LED as Power Indicator
8 LEDs as Digital Input Indicators

Power	
Power Consumption	1 W

Environment	
Operating Temperature	-25 to +75°C
Storage Temperature	-30 to +85°C
Humidity	5 to 95% RH, Non-condensing

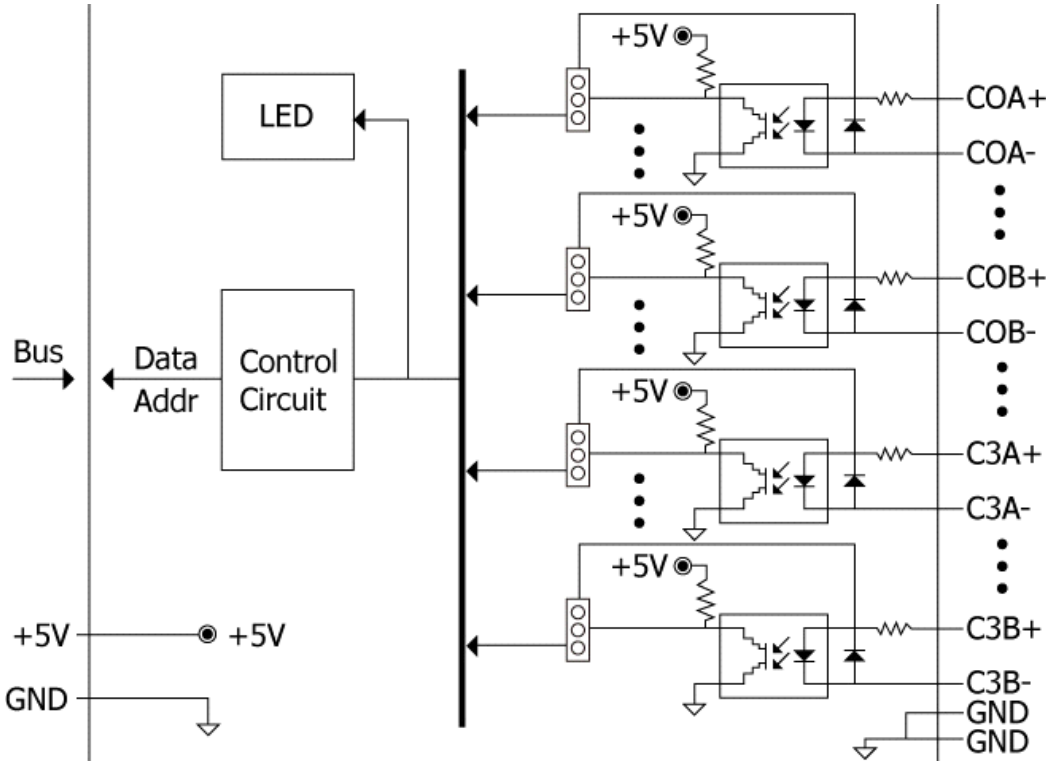
Dimensions
30 mm x 102 mm x 115 mm (W x L x H)

1.2. Pin Assignments

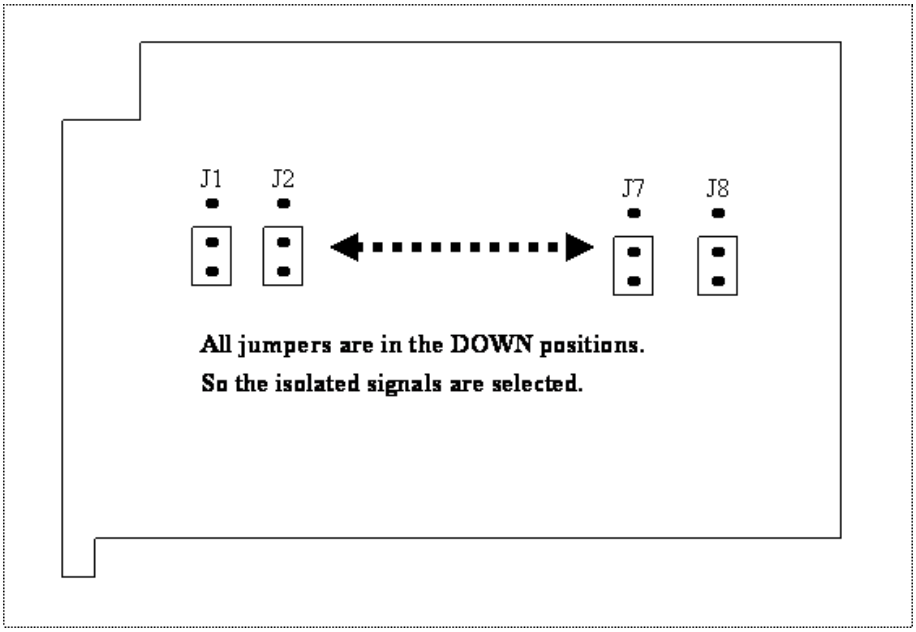


Terminal No.		Pin Assignment
	01	C0A+
	02	C0A-
	03	C0B+
	04	C0B-
	05	C1A+
	06	C1A-
	07	C1B+
	08	C1B-
	09	C2A+
	10	C2A-
	11	C2B+
	12	C2B-
	13	C3A+
	14	C3A-
	15	C3B+
	16	C3B-
	17	GND
	18	GND
	19	N.C
	20	N.C

1.3. I/O Structure



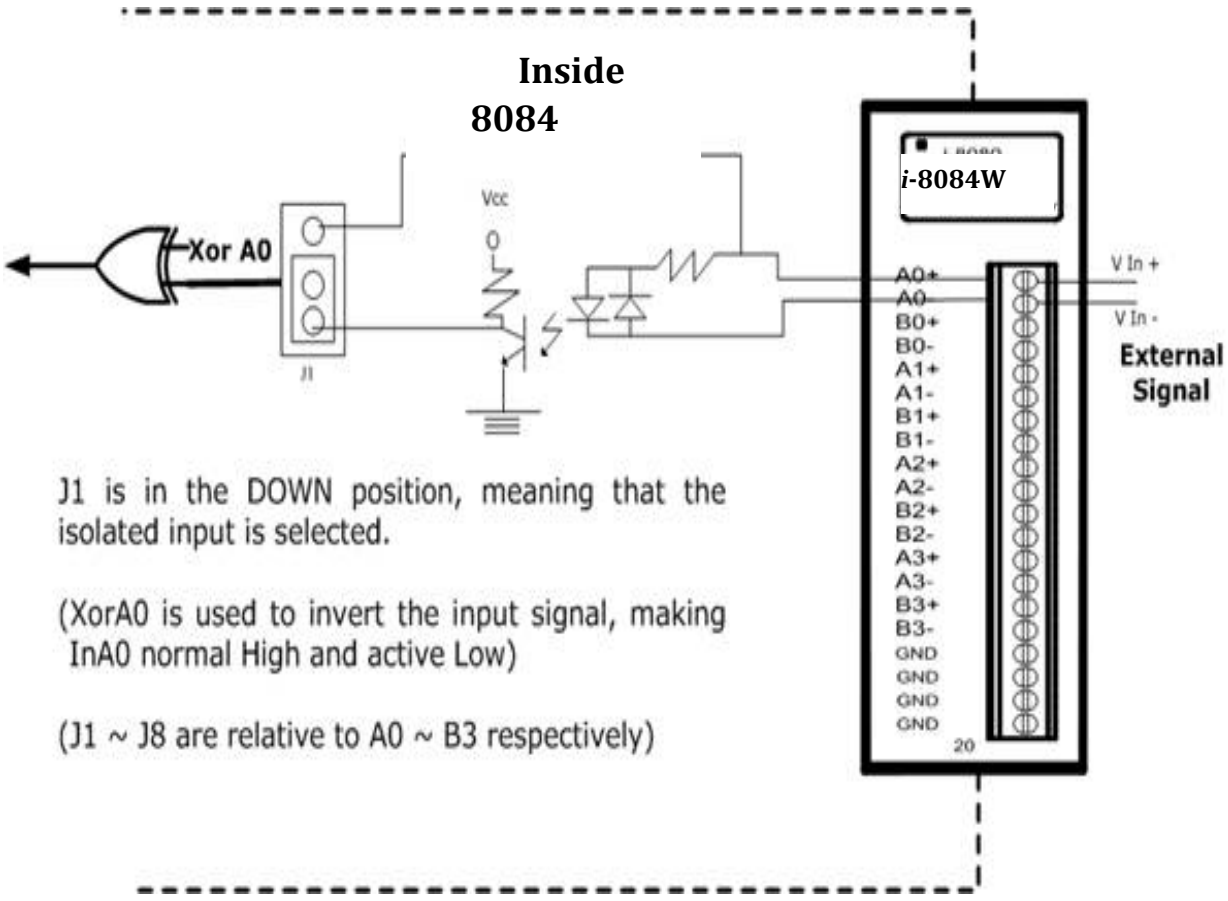
The I-8084W provides two input signal modes, isolation and TTL (Non-isolation).
The default values are all based on isolation mode and the jumper settings are as follows:



The location of the Jumpers on the I-8084W board

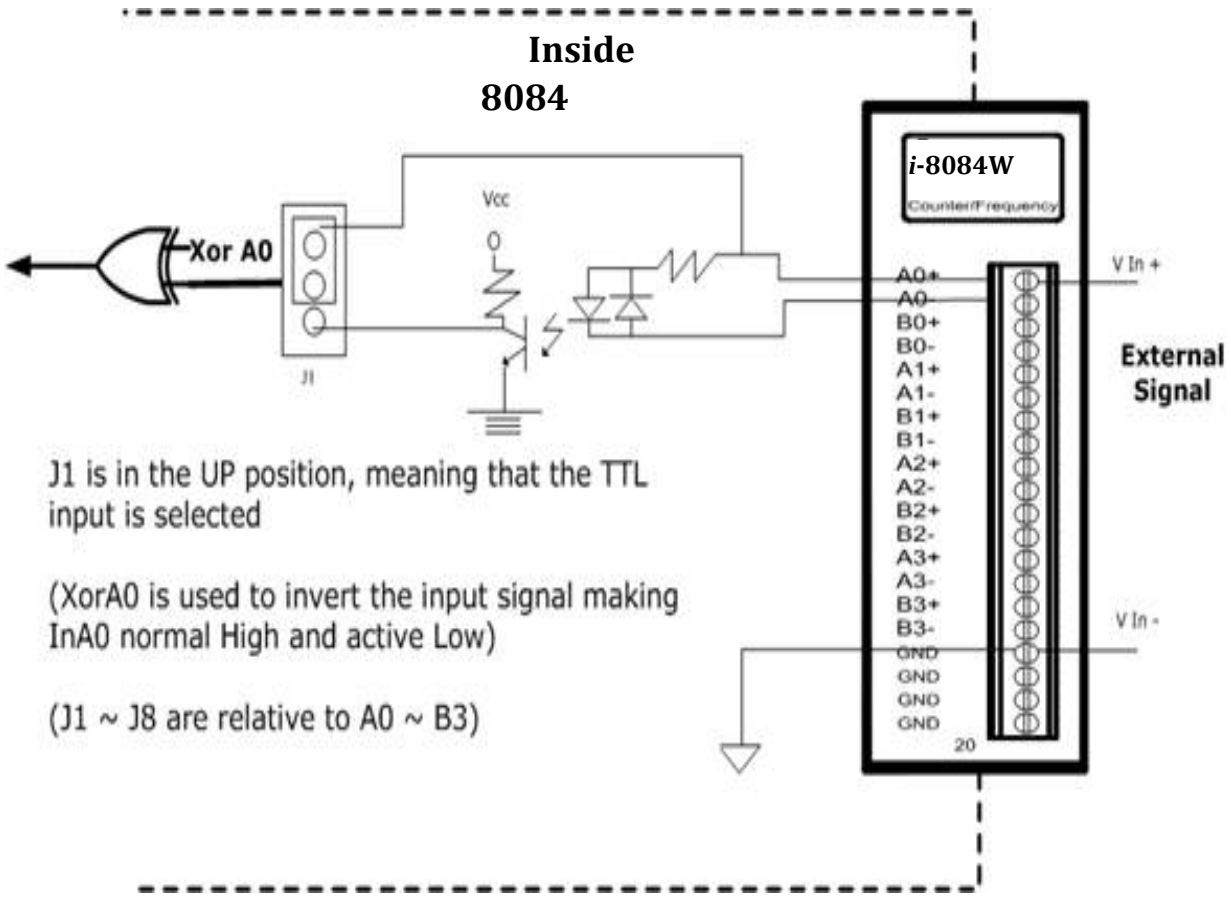
Isolated

Input:



TTL

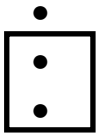
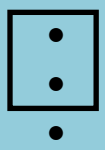
Input:


Inside
8084
i-8084W
Counter/Frequency
Xor A0
Vcc
J1
A0+
A0-
B0+
B0-
A1+
A1-
B1+
B1-
A2+
A2-
B2+
B2-
A3+
A3-
B3+
B3-
GND
GND
GND
GND
20
V In +
V In -
External
Signal
J1 is in the UP position, meaning that the TTL input is selected
(XorA0 is used to invert the input signal making InA0 normal High and active Low)
(J1 ~ J8 are relative to A0 ~ B3)

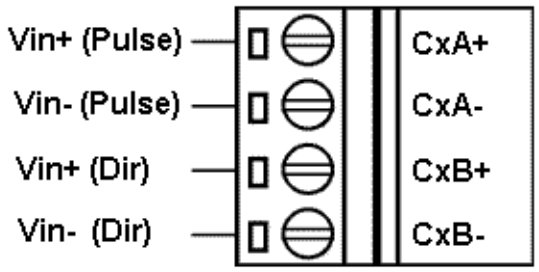
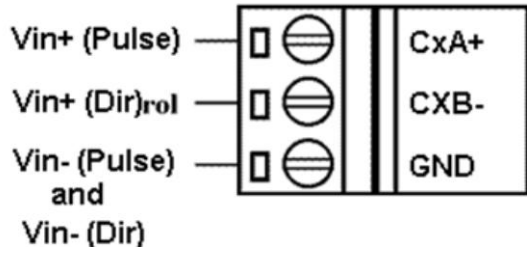
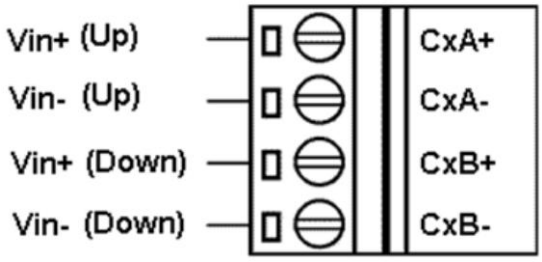
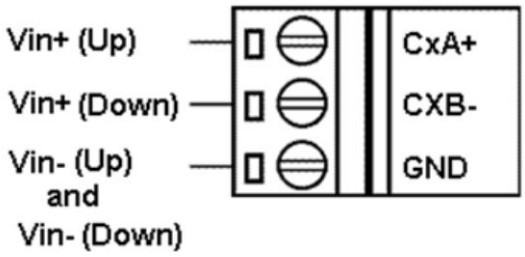
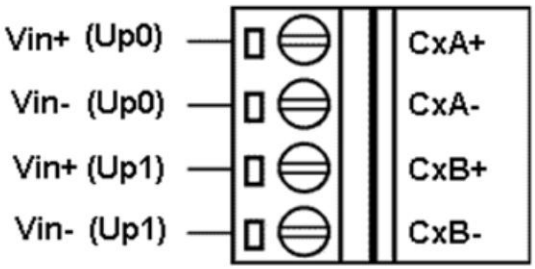
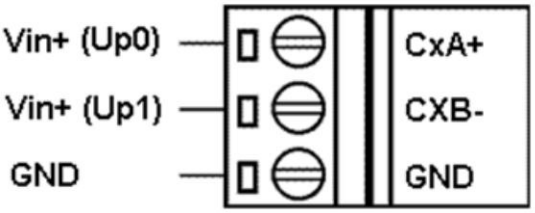
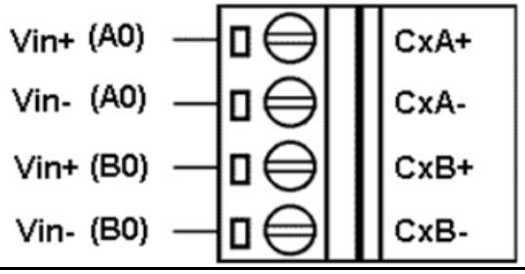
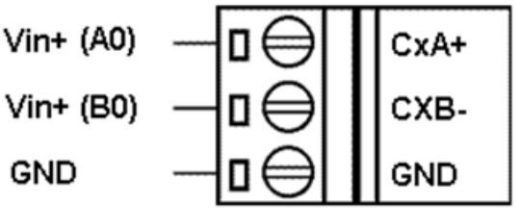
Isolated or TTL input is selected by using jumpers JP1 to JP8, as indicated below:

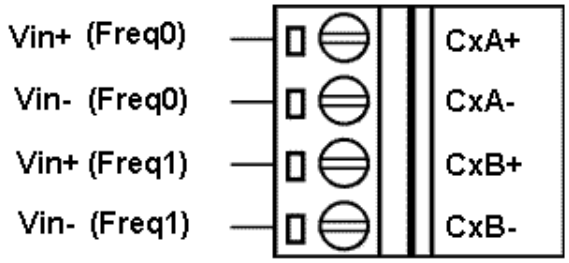
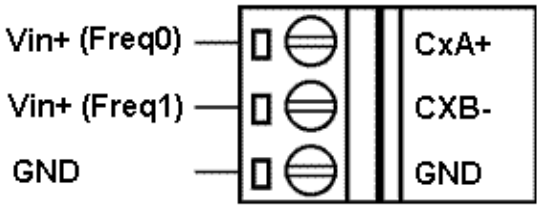
J1	**Select A0**
J2	Select B0
J3	Select A1
J4	Select B1
J5	Select A2
J6	Select B2
J7	Select A3
J8	Select B3

Selecting isolated or TTL input:

J1to J8	**J1to J8**
	
Isolated Input (Default Setting)	TTL Input

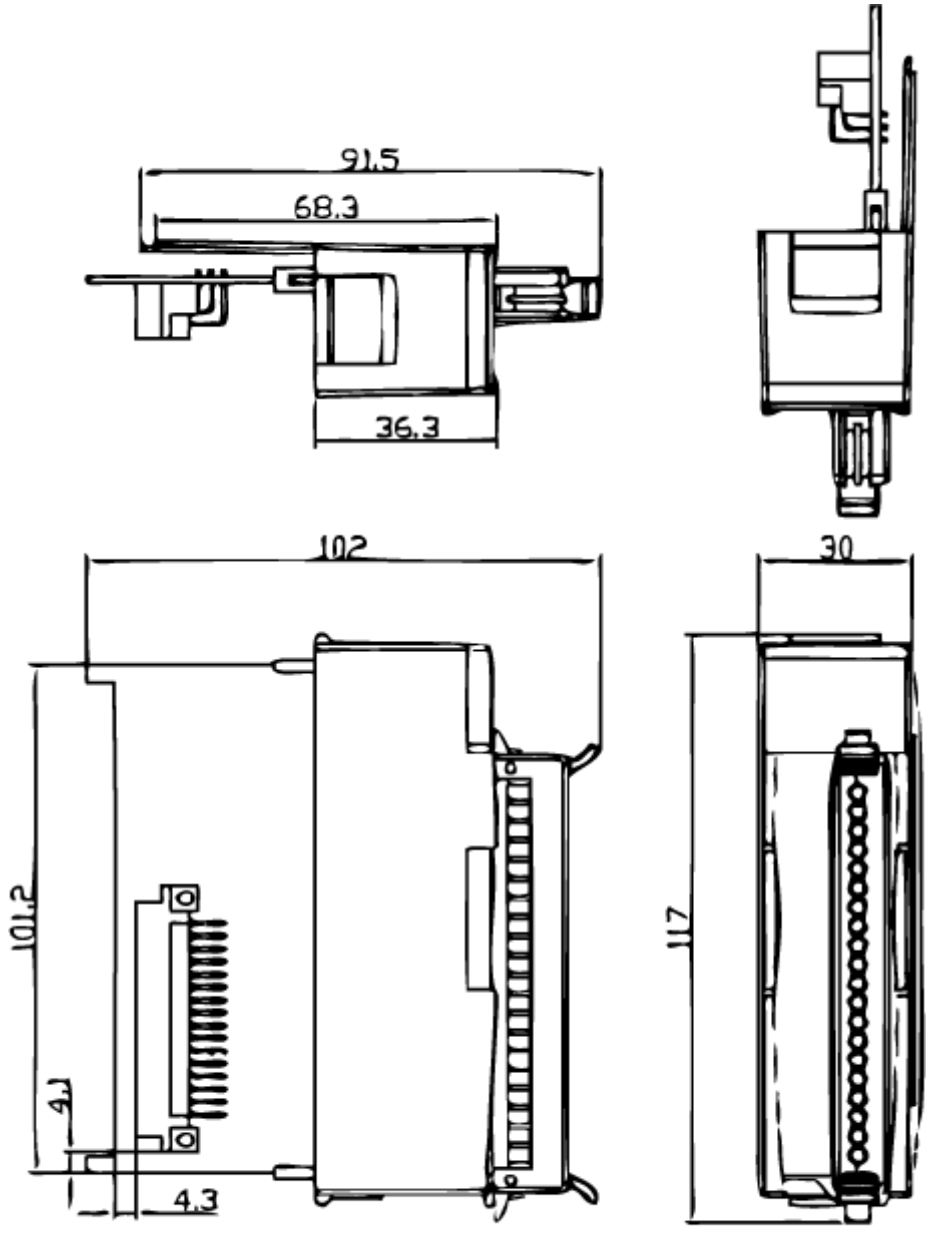
1.4. Wiring Connections

Counter Type		
Mode	Isolation	Non-isolation
Dir/Pulse		
Up/Down		
Up		
Quadrant		

Frequency Type		
Mode	Isolation	Non-isolation
Frequency		

1.5. Dimensions

Units: mm



2. Hardware Operation Principle

2.1. Input Signal Model

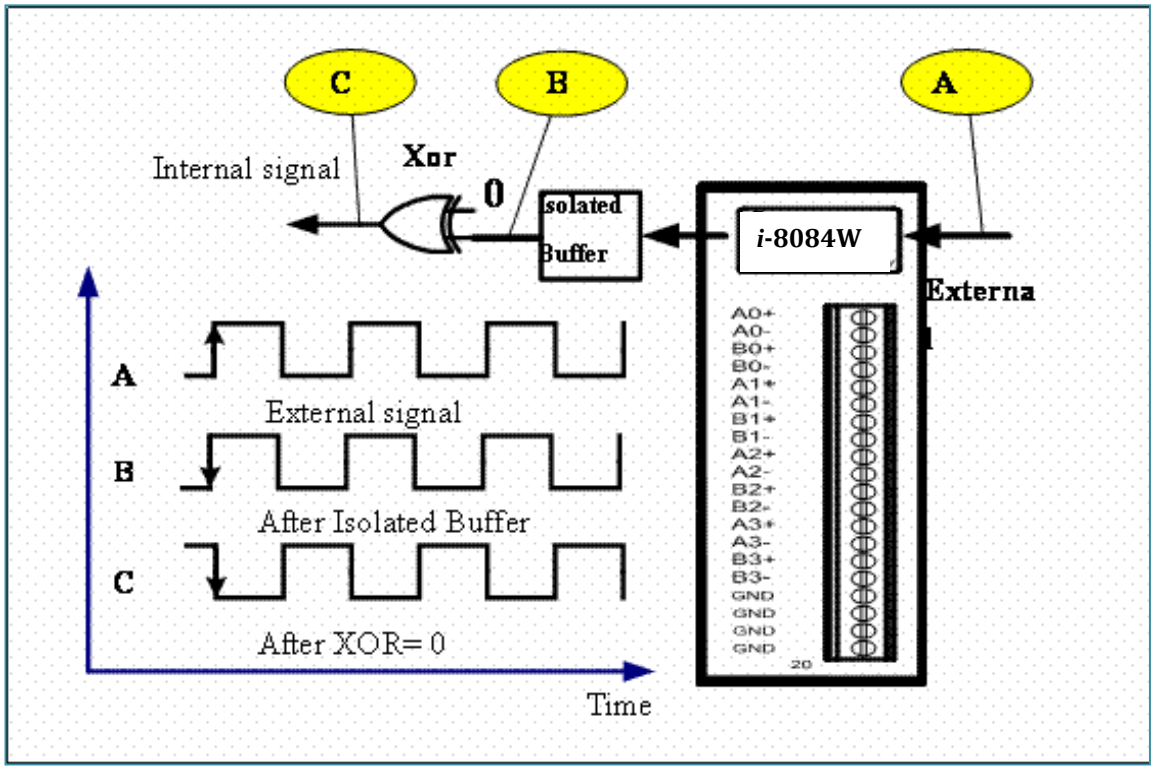
1. Isolated Input (XOR=0)

The operational logic applied on I-8084W modules is a falling edge trigger.
(Normal High and Active Low)

The external signal is input into an I-8084W module through the isolation mechanism, with the signal being reversed from the external signal.

This internal signal is the suggested waveform, as the XOR operation (XOR=0) doesn't need to be executed.

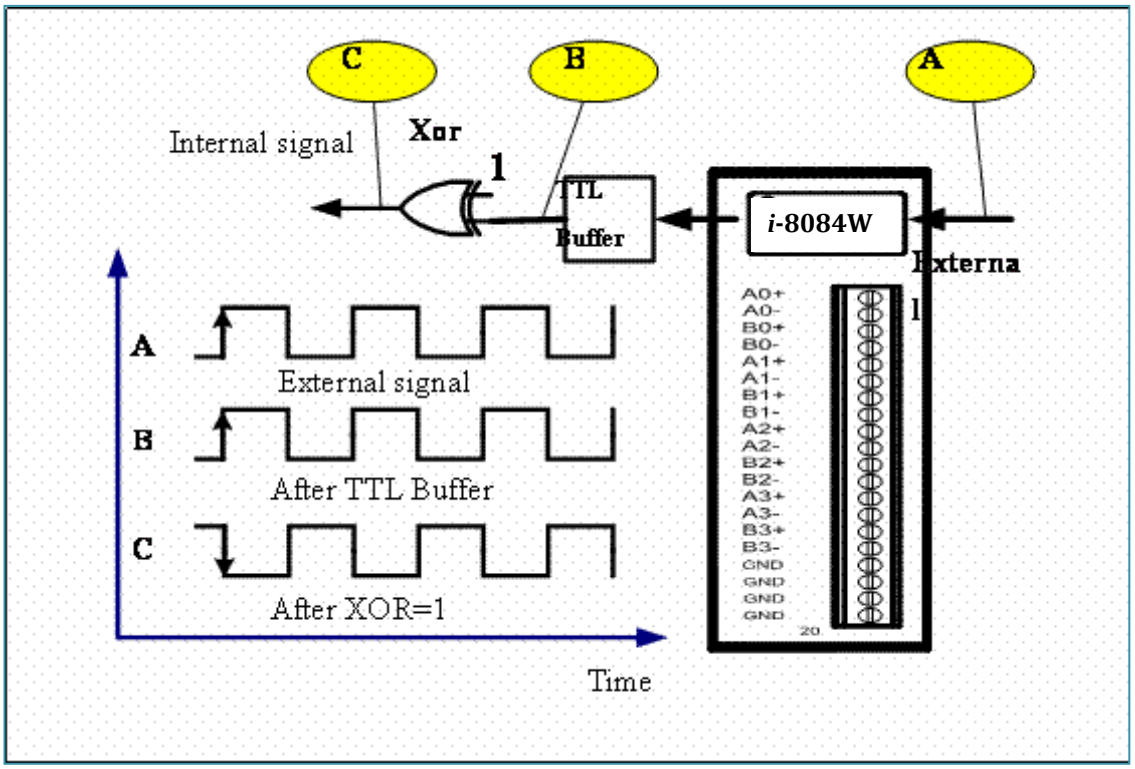
The solution is shown below.



2. TTL Input (XOR=1)

When an external TTL signal is input into an I-8084W module through the TTL mechanism, the signal will be the same as the external signal. This internal signal isn't the recommended waveform as the exclusive OR (XOR=1) operation must be executed to invert the waveform.

The solution is shown below.



3. Always XOR=0

Regardless of whether the input signal is TTL or isolated, XOR is always set to 0, and the maximum count error can only be 1. XOR=0 can be used for all cases if a 1-count error is acceptable.

Notes:

- When XOR = 0 and the I-8084W module is in its OPEN status (i.e., no signals are present on the input terminal), regardless of whether TTL or Isolated mode is selected, the signal at point C will always be 1. Similarly, if XOR=1 and the status is OPEN, then the signal at the point C will always be 0.

- If the input signal is a pulse rather than a 50/50 duty cycle square waveform, then the 1-count error will not occur as the pulse width is shorter.

2.2. Digital Low Pass Filter

The I-8084W includes three independent 2nd-order digital noise filters that can be used to remove noise, and are implemented as follows:

Channel	Low Pass Filter
0 (A0)	Low Pass Filter 0
1 (B0)	
2 (A1)	Low Pass Filter 1
3 (B1)	
4 (A2)	Low Pass Filter 2
5 (B2)	
6 (A3)	
7 (B3)	

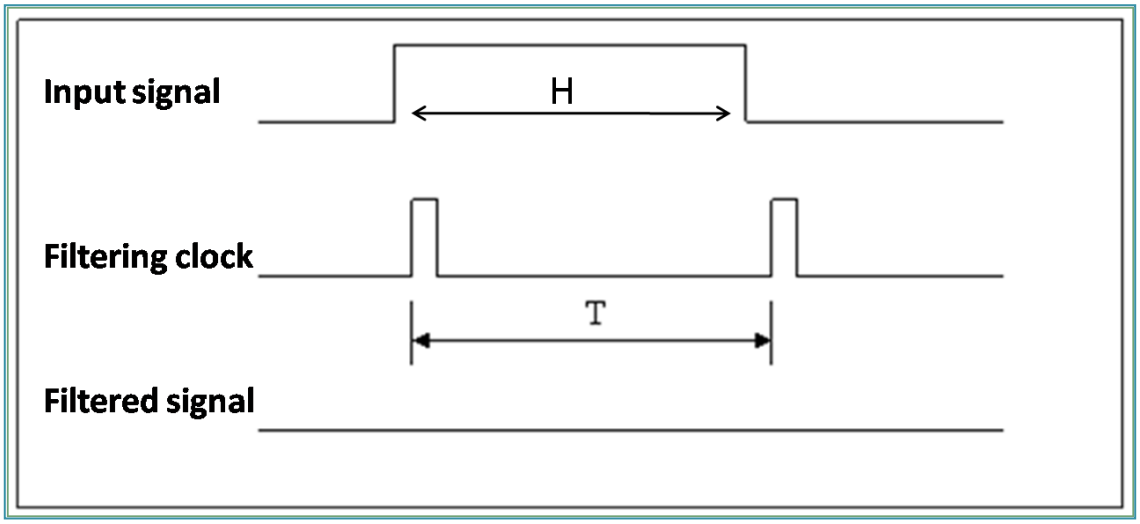
- The Low Pass Filter can be set to either enabled or disabled. The width of the Low Pass Filter is programmable and can be set within a range from 1 μs to 32767 μs.
- The Low Pass Filter can be applied to all working modes, both counter and frequency.
- All three Low Pass Filters are disabled by default when the module is shipped. ICP DAS provides API functions that can be used within your custom program to change the settings of the Low Pass Filters.

Refer to the following table for details of how to set the Low Pass Filter:

H = the **HIGH** width of the input signal **T** = the period of the filtering clock	
Case	**Filtering status**
H > 2T	The signal will be **PASSED**
T □ H □ 2T	The signal may be **FILTERED** or **PASSED**
H < T	The signal will be **FILTERED**

An example is illustrated in the following figure:

In this example, H < T and the high width of the input signal < the period of the filtering clock, and so the signal will be filtered.



Supposed T = 1 ms, that is the filtering clock has a frequency of 1 kHz. Now, if the duty cycle of the input signal is 50%, that is the high width is equal to the low width, then:

H < T → H < 1 ms
→ input signal period < 2 ms (duty cycle = 50%)
→ input frequency > 500 Hz,

Consequently, the input signal will be filtered.

Now, if the duty cycle of the input signal is 25%, that is the low width is three times that olf the high width, then:

H < T → H < 1 ms
→ Input signal period < 4 ms (duty cycle = 25%)
→ Input frequency > 250 Hz,

Consequently, the input signal will be filtered.

Similarly, the maximum period of the filtering clock can be calculated to allow the input signal to be passed using the formula H > 2T.

2.3. Operation Mode

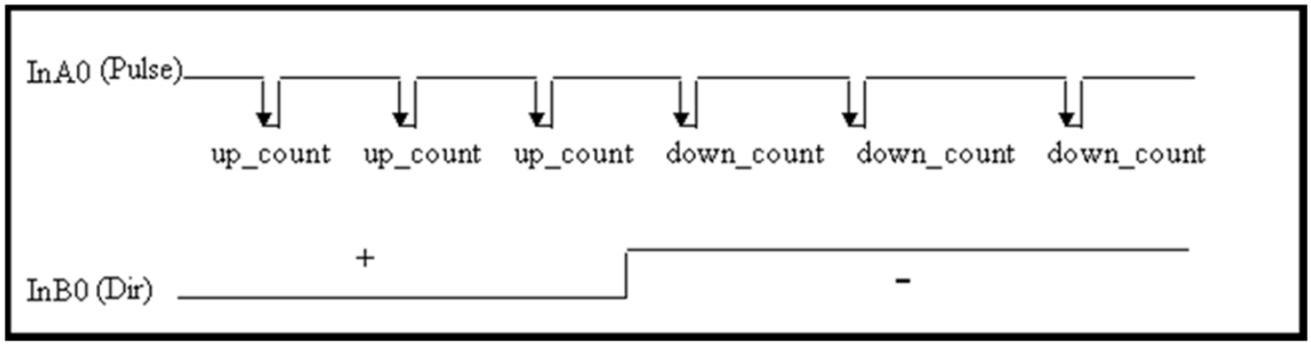
Operation Mode	Description	Number of counter and frequency sets
0	Dir/Pulse Counting Mode	4 sets
1	Up/Down Counting Mode	4 sets
2	Frequency Mode	8 sets
3	Up Counting Mode	8 sets
4	Quadrant Counting Mode	4 sets

The mapping table for the input channels and the working modes are indicated below:

Channel	Mode 0	Mode 1	Mode 2	Mode 3	Mode 4
0 (A0)	Pulse 0	Up 0	Freq 0	Up 0	A0
1 (B0)	Dir 0	Down 0	Freq 1	Up 1	B0
2 (A1)	Pulse 2	Up 2	Freq 2	Up 2	A1
3 (B1)	Dir 2	Down 2	Freq 3	Up 3	B1
4 (A2)	Pulse 4	Up 4	Freq 4	Up 4	A2
5 (B2)	Dir 4	Down 4	Freq 5	Up 5	B2
6 (A3)	Pulse 6	Up 6	Freq 6	Up 6	A3
7 (B3)	Dir 6	Down 6	Freq 7	Up 7	B3

2.3.1. Mode 0: Dir/Pulse Counting

The counter operation for mode 0 (Dir/Pulse mode) is as follows:

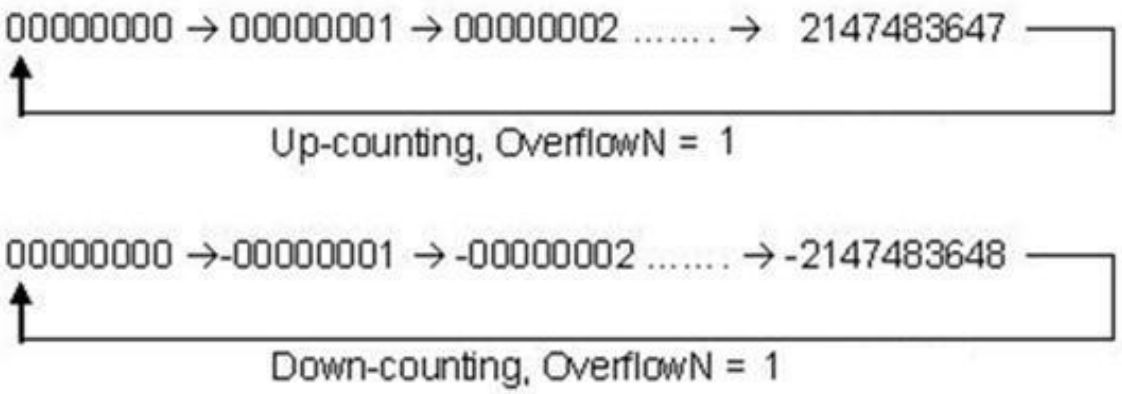


When InB0 is used as Dir:

- If InB0 is High, counter_0 will be increased by one for every falling edge of InA0.
- If InB0 is Low, counter_0 will be decreased by one for every falling edge of InA0.

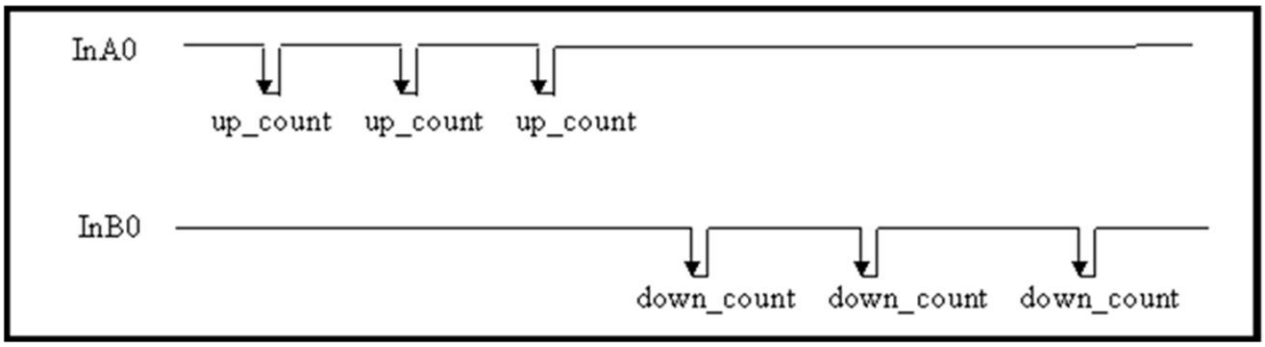
CountN	the current counter value for channel N, 32 bits wide, from -2147483648 to 2147483647
OverflowN	0 = no overflow 1 = overflow

This gives the following:



2.3.2. Mode 1: Up/Down Counting

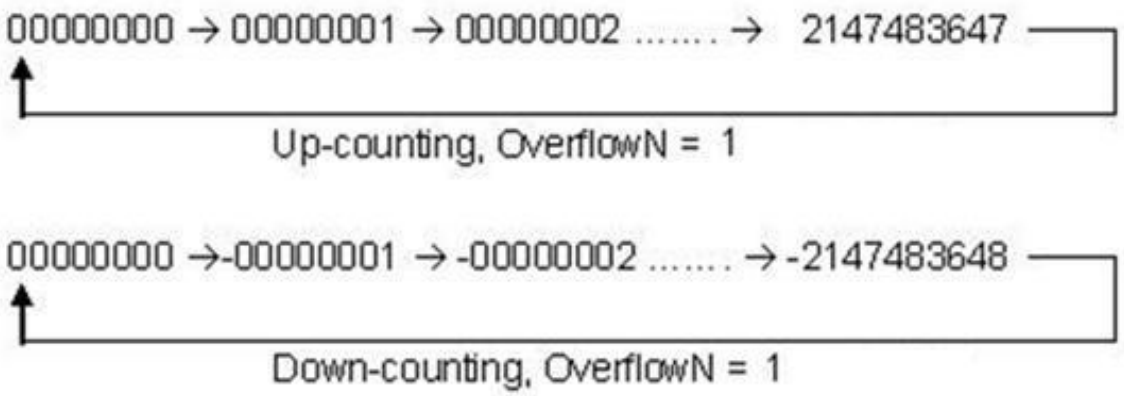
The counter operation for mode 1 (Up/Down mode) is as follows:



When InA0 is used as an UP_clock and InB0 is used as a DOWN_clock, counter_0 will be increased by one for every falling edge of InA0 and decreased by one for every falling edge of InB0.

CountN	the current counter value for channel N, 32 bits wide, from -2147483648 to 2147483647
OverflowN	0 = no overflow 1 = overflow

This gives the following:



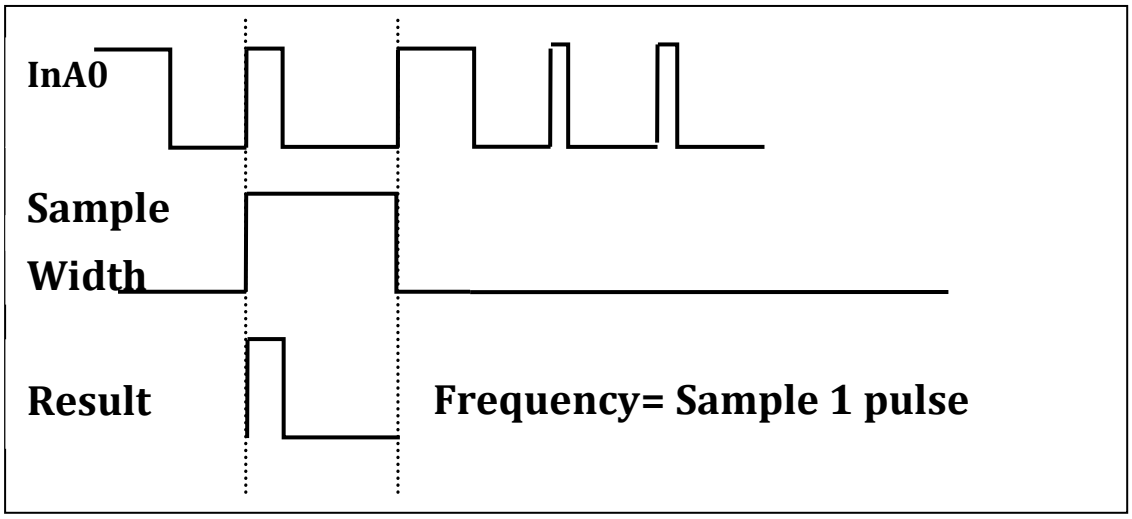
2.3.3. Mode 2: Frequency Mode

The operation for frequency mode is as follows:

We calculate frequency of a certain channel by how many counts in a period of time.

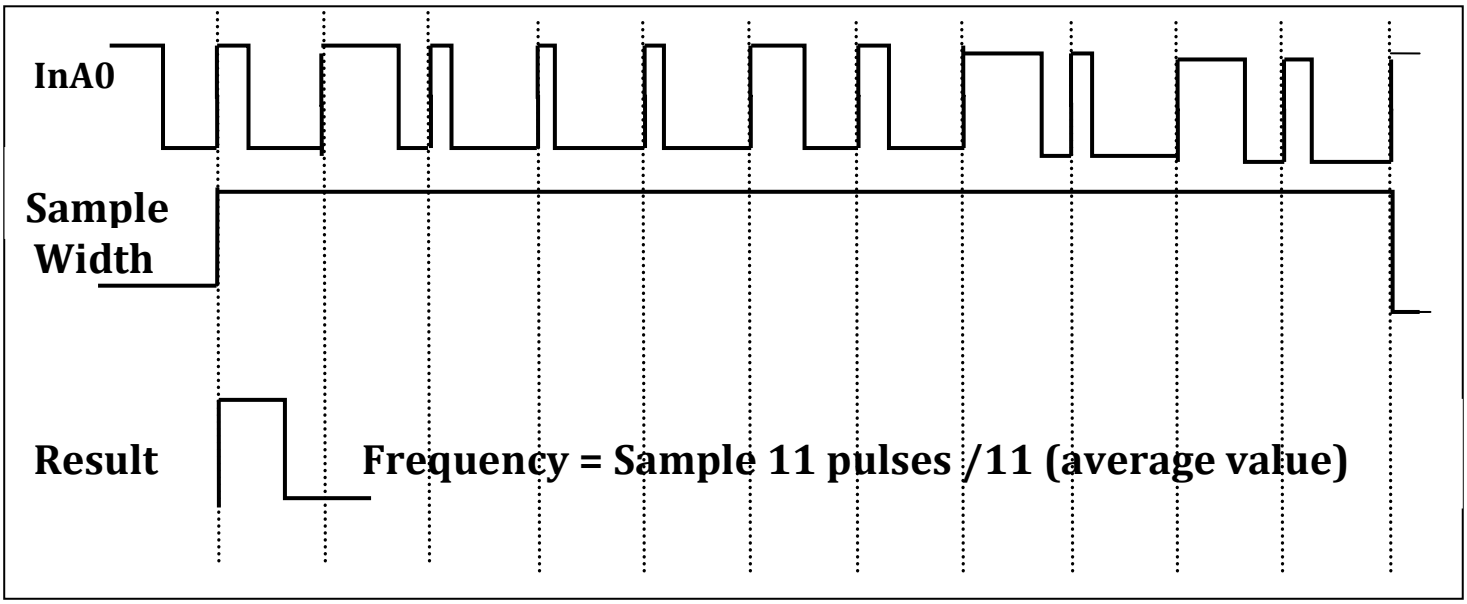
There are two modes: Normal Mode and High Speed Mode for measurement frequency.

Normal Mode:



Normal Mode will read 1 pulse, this pulse will be calculated frequency value.

High Speed Mode:



High Speed Mode will read 11 pulses to calculate average value of those 11 pluses.

High Speed Mode will be more accuracy than Normal mode for measurement Frequency. We suggest using High Speed Mode if measurement frequency is more than 10k Hz. About frequency accuracy, please refer Appendix B.

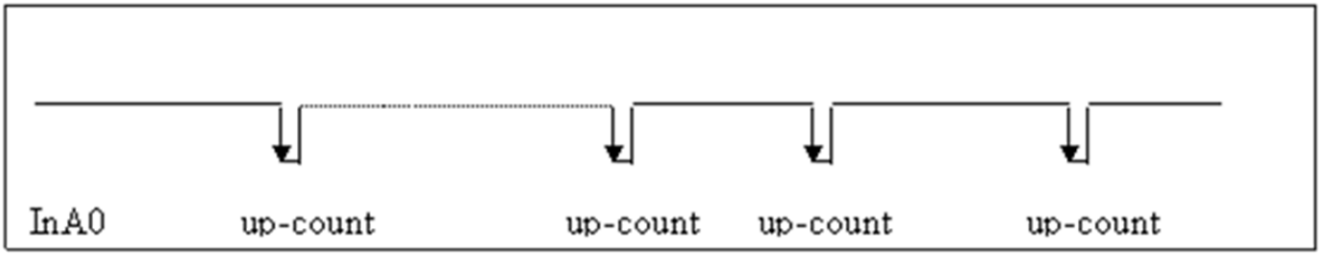
Set timeout value for Frequency Mode:

High Speed Mode will need more time than Normal Mode for measurement Frequency. The necessary times between Normal Mode and High Speed Mode for measurement Frequency are not the same. A case of low frequency such as 10 Hz or so, there should be a larger period of time to have enough counts to calculate frequency. We provide an API to do this: i8084W_SetFreqTimeoutValue to set timeout value.

For example: To measure 1k Hz frequency. In normal mode, I-8084W only needs 1 ms to update frequency value. In High Speed mode, it will measure 11 times (necessary 1ms*11= 11ms) and calculate frequency value. Then the frequency value will be update.
If I-8084W reads frequency faster than necessary value, it will keep frequency as last measurement. (About necessary time for I-8084W frequency mode, please refer Appendix B).

2.3.4. Mode 3: Up Counting

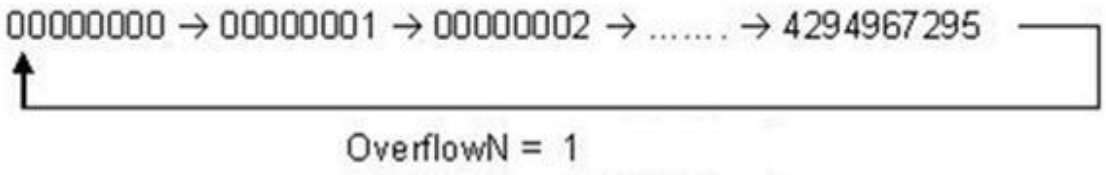
The counter operation for mode 3 is as follows:



Counter_0 will be incremented by one for every falling edge of InA0

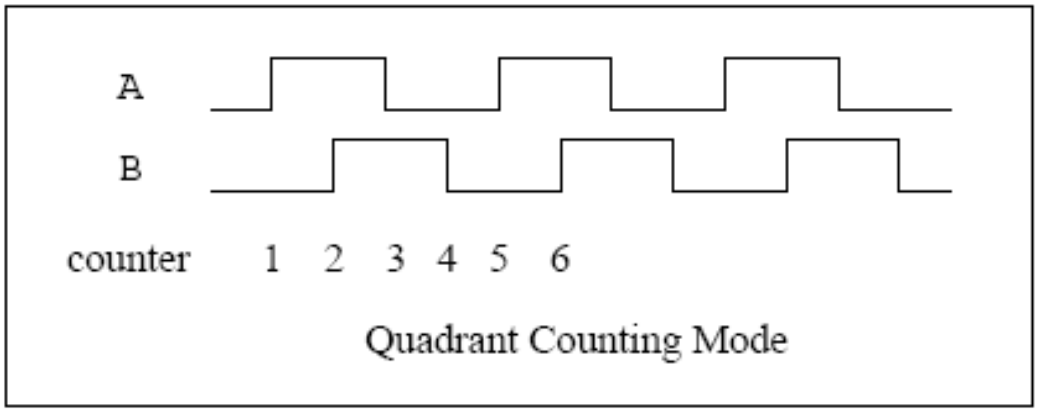
CountN = the current counter value for channel N, 32 bits wide, from 0 to 4294967295
OverflowN 0 = no overflow
1 = overflow

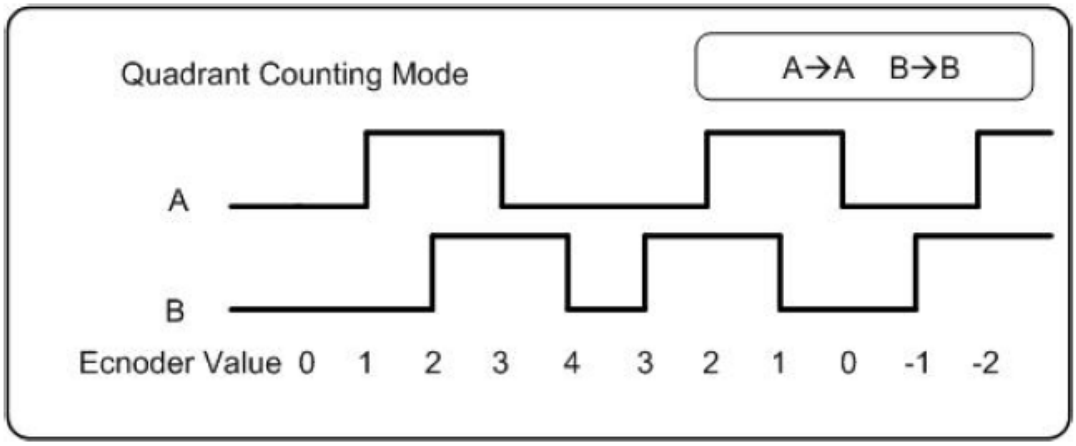
The counter operation is as follows:



2.3.5. Mode 4: Quadrant Counting

The counter operation for mode 4 is as follows:





When InA0 is used as an A signal and InB0 is used as a B signal:

- Counter_0 will be increased by one when the InA0 phase leads by 90 degrees to InB0.
- Counter_0 will be decreased by one when the InA0 phase lags by 90 degrees to InB0.

2.3.6. Location of the Demo Programs

ICP DAS provides a range of demo programs for different platforms that can be used to verify the functions of the I-8084W. The source code contained in these programs can also be reused in your own custom programs if needed. The following is a list of the locations where both the demo programs and associated libraries can be found on either the ICP DAS web site or the enclosed CD.

Platform	Location
For the I-8000 series on the Web	
Library	ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/8000/841x881x/demo/lib/
Demo	ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/8000/841x881x/demo/io_in_slot/
For the I-8000 series on the CD	
Library	CD:\Napdos\8000\841x881x\demo\Lib
Demo	CD:\Napdos\8000\841x881x\demo\IO_in_Slot
For the iPAC-8000 series on Web	
Library	ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/ipac8000/demo/basic/ip-84x1_ip-88x1/lib/
Demo	ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/ipac8000/demo/basic/ip-84x1_ip-88x1/io_in_slot/
For the iPAC-8000 series on the CD	
Library	CD:\Napdos\iPAC8000\Demo\Basic\iP-84x1_iP-88x1\Lib
Demo	CD:\Napdos\iPAC8000\Demo\Basic\iP-84x1_iP-88x1\IO_in_Slot
For the Windows CE5 platform on the Web	
Library	ftp://ftp.icpdas.com/pub/cd/winpac/napdos/wp-8x4x_ce50/sdk/io_modules/
Demo	ftp://ftp.icpdas.com/pub/cd/winpac/napdos/wp-8x4x_ce50/demo/winpac/evc/

pac_io/local/ (eVC demo)

ftp://ftp.icpdas.com/pub/cd/winpac/napdos/wp-8x4x_ce50/demo/winpac/dotnet/c%23.net/pac_io/local/ (C# demo)

Platform	Location
For the Windows CE5 platform on the CD	
Library	CD:\napdos\wp-8x4x_ce50\sdk\IO_Modules
Demo (eVC & C#)	CD:\napdos\wp-8x4x_ce50\Demo\WinPAC\eVC\PAC_IO\Local CD:\napdos\wp-8x4x_ce50\Demo\WinPAC\DOTNET\C#.NET\PAC_IO\Local
For the Windows CE6 platform on the Web	
XP-8000-CE6	ftp://ftp.icpdas.com/pub/cd/xp-8000-ce6/sdk/special_io/ ftp://ftp.icpdas.com/pub/cd/xp-8000-ce6/demo/xpac/vc2008/io/local/ ftp://ftp.icpdas.com/pub/cd/xp-8000-ce6/demo/xpac/c%23/io/local/
XP-8000-Atom-CE6	ftp://ftp.icpdas.com/pub/cd/xpac-atom-ce6/sdk/special_io/ ftp://ftp.icpdas.com/pub/cd/xpac-atom-ce6/demo/xpac/vc2008/io/local/ ftp://ftp.icpdas.com/pub/cd/xpac-atom-ce6/demo/xpac/c%23/io/local/
For the Windows CE6 platform on the CD	
XP-8000-CE6	CD:\SDK\Special_IO CD:\Demo\XPAC\VC2008\IO\Local CD:\Demo\XPAC\C#\IO\Local
XP-8000-Atom-CE6	CD:\SDK\Special_IO CD:\Demo\XPAC\VC2008\IO\Local CD:\Demo\XPAC\C#\IO\Local
For the Windows Embedded Standard on Web	
XP-8000	ftp://ftp.icpdas.com/pub/cd/xp-8000/sdk/io/ ftp://ftp.icpdas.com/pub/cd/xp-8000/demo/specialized_io/
XP-8000-	ftp://ftp.icpdas.com/pub/cd/xpac-atom/sdk/io/

Atom	ftp://ftp.icpdas.com/pub/cd/xpac-atom/demo/specialized_io/
For the Windows Embedded Standard on the CD	
XP-8000	CD:\SDK\IO CD:\Demo\Specialized_IO
XP-8000-Atom	CD:\SDK\IO CD:\Demo\Specialized_IO

3. API

ICP DAS provides APIs, libraries and demo programs, including the source code, that allow the integration of the I-8084W into the platforms indicated in the table below.

The APIs and programming procedures are similar on both the MiniOS7 and the Windows platforms, with the only difference being the **prefix characters** added to the name of the function in the library (APIs). For functions applicable to the MiniOS7 and Linux platforms, the prefix "8084W_" is added to the function name, and the prefix "pac_i8084W_" is added to functions applicable to the Windows platform.

In this document, the function relevant to the MiniOS7 platform is used in the examples and as the title of the section for each function.

The following table gives an overview of the relationship between the different platforms and the product series, together with the respective prefix used for the function name.

Platform	Product	API Prefix
Windows CE5 Windows CE6	WP-8000 series WP-2000 series XP-8000-CE6 series	"pac_i8084W_ "+ function name
Windows Embedded Standard (WES)	XP-8000 series	"pac_i8014W_ "+ function name
MiniOS7	I-8000 series iPAC-8000 series VP-2000 series	"i8084W_" + function name
Linux	LinPAC-8000 series	"i8084W_" + function name

Function list

The following is a list of the functions provided in the 8084W.lib for all platforms.

Function	Description

i8084W_GetLibVersion	This function is used to read the version information for the currently installed library
i8084W_GetLibDate	This function is used to read the build date information for the currently installed library.
i8084W_GetFirmwareVersion	This function is used to read the firmware version information for the module in the specified slot.
i8084W_InitDriver	This function is used to initialize the 8084W Library. Note that this function MUST be used before using any other function in the library,
i8084W_SetChannelMode	This function is used to set the operation mode for a specified channel.
i8084W_ReadChannelMode	This function is used to read the operation mode for a specified channel.
i8084W_ReadCntABPhase	This function is used to read the counter value for a specified channel in Quadrant Counting mode
i8084W_ReadCntPulseDir	This function is used to read the counter value for a specified channel in Dir/Pulse counting mode.
i8084W_ReadCntUpDown	This function is used to read the counter value for a specified channel in Up/Down counting mode.
i8084W_ReadCntUp	This function is used to read the counter value for a specified channel in Up counting mode
i8084W_ReadFreqInFloat	This function is used to read the frequency of the specified channel in Frequency mode.
i8084W_ClrCnt	This function is used to clear the counter value for a specified channel regardless of the operation mode.
i8084W_ClrAllCnt	This function is used to clear the counter value for all channels regardless of the operation mode.
i8084W_RecoverDefaultSetting	This function is used to restore the default settings of the I-8084W module installed in a specified slot.
i8084W_ReadXorRegister	This function is used to read the status of the XOR register for a specified channel.

Function	Description
i8084W_SetXorRegister	This function is used to set the status of the XOR register for a specified channel.
i8084W_ReadLowPassFilterUs	This function is used to read the width of the low pass filter in microseconds.
i8084W_SetLowPassFilterUs	This function is used to set the width of the low pass filter in microseconds.
i8084W_ReadLowPassFilterStatus	This function is used to read the status of the low pass filter for a specified channel.
i8084W_SetLowPassFilterStatus	This function is used to set the status of the low pass filter for a specified channel.
i8084W_ReadFreqMode	This function is used to read the frequency mode for a specified channel.
i8084W_SetFreqMode	This function is used to set the frequency mode for a specified channel.
i8084W_ReadFreqTimeoutValue	This function is used to read the timeout value for reading the frequency.
i8084W_SetFreqTimeoutValue	This function is used to set the timeout value for reading the frequency.
i8084W_ReadDIXor	This function is used to read the Digital Input value after the XOR operation has been performed.
i8084W_ReadDIXorLPF	This function is used to read the Digital Input value after the both the XOR and the low pass filter operations have been performed.

3.1 i8084W_GetLibVersion

This function is used to read the version information for the library installed on the I-8084 module.

Prototype

For MiniOS7

short i8084W_GetLibVersion();

For Windows (CE and WES)

Short pac_i8084W_GetLibVersion();

Parameters

This function does not require any parameters

Return Values

The version information for the Library file
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int version;
version = i8084W_GetLibVersion( );
```

[C++]Windows

```
int version;
version = pac_i8084W_GetLibVersion( );
```

[C#]

```
string version;
version = pac8084W.GetLibraryVersion( );
```

3.2 i8084W_GetLibDate

This function is used to read the build date for the Library currently installed on the I-8084 module.

Prototype

For MiniOS7

void i8084W_GetLibDate(char* LibDate);

For Windows (CE and WES)

void pac_i8084W_GetLibDate(char* LibDate);

Parameters

*LibDate: [out] The build date for the I-8084W library

Return Values

This function does not return a value
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++]MiniOS7

```
char libdate[32];
i8084W_GetLibDate(libdate);
```

[C++] Windows

```
char libdate[32];
pac_i8084W_GetLibDate(libdate);
```

[C#]

```
string libdate;
```

```
libdate = pac8084W.LibDate();
```

3.3 i8084W_GetFirmwareVersion

This function is used to read the firmware (LPGA) version information for the I-8017 module inserted in the specified slot.

Prototype

For MiniOS7

short i8084W_GetFirmwareVersion(int slot, short* firmwareVersion);

For Windows (CE and WES)

short pac_i8084W_GetFirmwareVersion(int slot, short* firmwareVersion);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

**firmwareVersion: [out]*

The firmware version information for the I-8084W LPGA code.

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
short firmwareVersion;
i8084W_GetFirmwareVersion(slot, &firmwareVersion);
```

[C++] Windows

```
int slot = 1;
short firmwareVersion;
pac_i8084W_GetFirmwareVersion(slot, &firmwareVersion);
```

[C#]

```
int slot = 1;
string firmwareVersion;
firmwareVersion = pac8084W.FirmwareVersion(slot);
```

3.4 i8084W_InitDriver

This function is used to initialize the Library for the 8084W module and **MUST** be used before any other library function is used.

Prototype

For MiniOS7

short i8084W_InitDriver(int slot);

For Windows (CE and WES)

short pac_i8084W_InitDriver(int slot);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

Return Values

0: The module inserted in the slot is an I-8084W.
-1: There is no I-8084W module inserted in this slot.
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Note: Before executing any functions on the I-8084W module, the InitDriver function needs to be called once for each I-8084W. If there are two or more I-8084W modules inserted in the controller, this function needs to be called individually for each I-8084W module by specifying the slot number where the I-8084W is inserted.

Examples

[C++]MiniOS7

```
int slot = 1;
i8084W_InitDriver(slot);
```

[C++]Windows

```
int slot = 1;
pac_i8084W_InitDriver(slot);
```

[C#]

```
int slot = 1;
pac8084W.Init(slot);
```

3.5 i8084W_SetChannelMode

This function is used to set the channel operation mode for the specific channel of the specified I-8084W module.

Prototype

For MiniOS7

int i8084W_SetChannelMode(int slot, int channel, int mode);

For Windows (CE and WES)

int pac_i8084W_SetChannelMode(int slot, int channel, int mode);

Parameters

slot:[in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel from which the I-8084W reads input. Valid range = 0 to 7.

Note: Configuring the operation mode for one of the channels in any of the four sets causes the other channel in the same set to also be configured to the same mode. Four sets can be used in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode and the operation mode does not need to be specified for both channels in these modes.

Set	Channels	Remarks
1	0, 1	The mapping between sets and channels in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode. Configuring the operation mode for one of the channels in a set also configures the other channel to the same mode.
2	2, 3	
3	4, 5	
4	6, 7	

mode: [in]

Specifies the operation mode for the specified channel.

Operation Mode	Description	Number of Counter and Frequency sets
00	Dir/Pulse Counting mode	4 sets
01	Up/Down Counting mode	4 sets
02	Frequency mode	8 sets
03	Up Counting mode	8 sets
04	Quadrant Counting mode	4 sets

Return Values

This function does not return a value

Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 3;    // Up counting mode
i8084W_InitDriver(slot);
i8084W_SetChannelMode(slot, channel, mode);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 3;    // Up counting mode
pac_i8084W_InitDriver(slot);
pac_i8084W_SetChannelMode(slot, channel, mode);
```

[C#]

```
using pac8084WNet;
…
int slot = 1;
```

```
int channel = 0;
Int16 mode = 3; // Up counting mode
pac8084W.Init(slot);
pac8084W.SetChannelMode(slot, channel,mode);
```

3.6 i8084W_ReadChannelMode

This function is used to read the channel operation mode currently in use by the specific channel of the specified I-8084W module.

Prototype

For MiniOS7

int i8084W_ReadChannelMode(int slot, int channel, int * mode);

For Windows (CE and WES)

int pac_i8084W_ReadChannelMode(int slot, int channel, int* mode);

Parameters

slot:[in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

Note: Configuring the operation mode for one of the channels in any of the four sets causes the other channel in the same set to also be configured to the same mode. Four sets can be used in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode, and the operation mode does not need to be specified for both channels in these modes.

Set	Channels	Remarks
1	0, 1	The mapping between sets and channels in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode. If one of the channels in a set is configured to a specific operation mode, then the other channel in the set will also be configured to the same mode, so there is no need to read both channels.
2	2, 3	
3	4, 5	
4	6, 7	

**mode: [out]*

A pointer to a value representing the operation mode. Valid range = 0 to 4.

Operation Mode	Description	Number of Counter and Frequency sets
00	Dir/Pulse Counting mode	4 sets
01	Up/Down Counting mode	4 sets
02	Frequency mode	8 sets
03	Up Counting mode	8 sets
04	Quadrant Counting mode	4 sets

Return Values

This function does not return a value

Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode;
i8084W_ReadChannelMode(slot, channel, &mode);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode;
pac_i8084W_ReadChannelMode(slot, channel, &mode);
```

[C#]

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
Int16 mode = 0;
```

```
pac8084W.Init (slot);
pac8084W.ReadChannelMode(slot, channel, ref mode);
```

3.7 i8084W_ReadCntABPhase

This function is used to read the counter value for a specific channel of the specified I-8084W module in Quadrant Counting mode.

Prototype

For MiniOS7

int i8084W_ReadCntABPhase(int slot, int channel, long * Cnt32U, int * overflow);

For Windows (CE and WES)

int pac_i8084W_ReadCntABPhase(int slot, int channel, long * Cnt32U, int * overflow);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7

Note: Configuring the operation mode for one of the channels in any of the four sets causes the other channel in the same set to also be configured to the same mode. Four sets can be used in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode, and the operation mode does not need to be specified for both channels in these modes.

Set	Channels	Remarks
1	0, 1	The mapping between sets and channels in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode. If one of the channels in a set is configured to a specific operation mode, then the other channel in the set will also be configured to the same mode, so there is no need to read both channels.
2	2, 3	
3	4, 5	
4	6, 7	

**Cnt32U: [out]*

A pointer to a value representing the counter value.
Bit31 = 0: Up Counter (count > 0)
Bit31 = 1: Down Counter (count < 0)

**overflow: [out]*

A pointer to a value representing the status of the overflow.
0: No overflow has occurred
1: An overflow has occurred

Return Values

This function does not return a value
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 4;    // Quadrant Counting mode (AB Phase)
long Cnt32U = 0;
int overflow = 0;
i8084W_InitDriver(slot);
i8084W_SetChannelMode(slot, channel, mode);
i8084W_ReadCntABPhase(slot, channel, &Cnt32U, &overflow);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 4;    // Quadrant Counting mode (AB Phase)
long Cnt32U = 0;
int overflow = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetChannelMode(slot, channel, mode);
```

```
pac_i8084W_ReadCntABPhase(slot, channel, &Cnt32U, &overflow);
```

[C#]

```
using pac8084WNet;
...
int slot = 1;
int channel = 0;
Int32 count = 0;
Int16 overflow = 0;
Int16 mode = 4; // Quadrant counting mode (AB Phase)
pac8084W.Init(slot);
pac8084W.SetChannelMode(slot, channel, mode);
pac8084W.ReadABPhase(slot, channel, ref count, ref overflow);
```

3.8 i8084W_ReadCntPulseDir

This function is used to read the counter value for a specific channel of the specified I-8084W module in Dir/Pulse counting mode.

Prototype

For MiniOS7

int i8084W_ReadCntPulseDir(int slot, int channel, long *Cnt32U, int *overflow);

For Windows (CE and WES)

int pac_i8084W_ReadCntPulseDir(int slot, int channel, long *Cnt32U, int *overflow);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

Note: Configuring the operation mode for one of the channels in any of the four sets causes the other channel in the same set to also be configured to the same mode. Four sets can be used in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode and the operation mode does not need to be specified for both channels in these modes.

Set	Channels	Remarks
1	0, 1	The mapping between sets and channels in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode. If one of the channels in a set is configured to a specific operation mode, then the other channel in the set will also be configured to the same mode, so there is no need to read both channels.
2	2, 3	
3	4, 5	
4	6, 7	

**Cnt32U: [out]*

A pointer to a value representing the counter value.
Bit31 = 0: Up Counter (count > 0)
Bit31 = 1: Down Counter (count < 0)

**overflow: [out]*

A pointer to a value representing the status of the overflow.
0: No overflow has occurred
1: An overflow has occurred

Return Values

This function does not return a value
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 0;    // Dir/Pulse counting mode
long Cnt32U = 0;
int overflow = 0;
i8084W_InitDriver(slot);
i8084W_SetChannelMode(slot, channel, mode);
i8084W_ReadCntPulseDir(slot, channel, &Cnt32U, &overflow);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 0;    // Dir/Pulse counting mode
long Cnt32U = 0;
int overflow = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetChannelMode(slot, channel, mode);
```

```
pac_i8084W_ReadCntPulseDir (slot, channel, &Cnt32U, &overflow);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int channel = 0;
Int32 count = 0;
Int16 overflow = 0;
Int16 mode = 0; // Dir/Pulse counting mode
pac8084W.Init(slot);
pac8084W.SetChannelMode(slot, channel, mode);
pac8084W.ReadPulseDir(slot, channel, ref count, ref overflow);
```

3.9 i8084W_ReadCntUpDown

This function is used to read the counter value for a specific channel of the specified I-8084W module in Up/Down (CW/CCW) counting mode.

Prototype

For MiniOS7

int pac_i8084W_ReadCntUpDown(int slot, int channel, long *Cnt32U, int *overflow);

For Windows (CE and WES)

int pac_i8084W_ReadCntUpDown(int slot, int channel, long *Cnt32U, int *overflow);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

Note: Configuring the operation mode for one of the channels in any of the four sets causes the other channel in the same set to also be configured to the same mode. Four sets can be used in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode, and the operation mode does not need to be specified for both channels in these modes.

Set	Channels	Remarks
1	0, 1	The mapping between sets and channels in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode. If one of the channels in a set is configured to a specific operation mode, then the other channel in the set will also be configured to the same mode, so there is no need to read both channels..
2	2, 3	
3	4, 5	
4	6, 7	

**Cnt32U: [out]*

A pointer to a value representing the counter value.
Bit31 = 0: Up Counter (count > 0)
Bit31 = 1: Down Counter (count < 0)

**overflow: [out]*

A pointer to a value representing the status of the overflow.
0: No overflow has occurred
1: An overflow has occurred

Return Values

This function does not return a value
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 1;    // Up/Down counting mode (CW/CCW)
long Cnt32U = 0;
int overflow = 0;
i8084W_InitDriver(slot);
i8084W_SetChannelMode(slot, channel, mode);
i8084W_ReadCntUpDown(slot, channel, &Cnt32U, &overflow);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 1;    // Up/Down counting mode (CW/CCW)
long Cnt32U = 0;
int overflow = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetChannelMode(slot, channel, mode);
```

```
pac_i8084W_ReadCntUpDown(slot, channel, &Cnt32U, &overflow);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int channel = 0;
Int32 count = 0;
Int16 overflow = 0;
Int16 mode = 1; // Up/Down counting mode (CW/CCW)
pac8084W.Init(slot);
pac8084W.SetChannelMode(slot, channel, mode);
pac8084W.ReadUpDown(slot, channel, ref count, ref overflow);
```

3.10 i8084W_ReadCntUp

This function is used to read the counter value for a specific channel of the specified I-8084W module in Up counting mode.

Prototype

For MiniOS7

int i8084W_ReadCntUp(int slot, int channel, unsigned long *Cnt32U, unsigned int *overflow);

For Windows (CE and WES)

int pac_i8084W_ReadCntUp(int slot, int channel, unsigned long *Cnt32U, unsigned int *overflow);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

**Cnt32U: [out]*

A pointer to a value representing the counter value.
Bit31 = 0: Up Counter (count > 0)
Bit31 = 1: Down Counter (count < 0)

**overflow: [out]*

A pointer to a value representing the status of the overflow .
0: No overflow has occurred
1: An overflow has occurred

Return Values

This function does not return a value

Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 3;    // Up counting mode
unsigned long Cnt32U = 0;
unsigned int overflow = 0;
i8084W_InitDriver(slot);
i8084W_SetChannelMode(slot, channel, mode);
i8084W_ReadCntUp(slot, channel, &Cnt32U, &overflow);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 3;    // Up counting mode
unsigned long Cnt32U = 0;
unsigned int overflow = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetChannelMode(slot, channel, mode);
pac_i8084W_ReadCntUp(slot, channel, &Cnt32U, &overflow);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
UInt32 Cnt32U = 0;
UInt16 overflow = 0;
Int16 mode = 3; //Up counting mode
```

```
pac8084W.Init(slot);
pac8084W.SetChannelMode(slot, channel, mode);
pac8084W.ReadUpCnt(slot, channel, ref Cnt32U,ref overflow);
```

3.11 i8084W_ReadFreq

This function is used to read the frequency of the signal for a specific channel of the specified I-8084W module in Frequency mode.

Prototype

For MiniOS7

int i8084W_ReadFreq(int slot, int channel, unsigned long * Freq);

For Windows (CE and WES)

int pac_i8084W_ReadFreq(int slot, int channel, unsigned long * Freq);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

**Freq: [out]*

A pointer to a value representing the signal frequency (in Hz)

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 2;    // Frequency mode
```

```
unsigned long Freq = 0;
i8084W_InitDriver(slot);
i8084W_SetChannelMode(slot, channel, mode);
i8084W_ReadFreq(slot, channel, &Freq);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 2;    // Frequency mode
unsigned long Freq = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetChannelMode(slot, channel, mode);
pac_i8084W_ReadFreq(slot, channel, &Freq);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
UInt32 Freq = 0;
Int16 mode = 2; // Frequency mode
pac8084W.Init(slot);
pac8084W.SetChannelMode(slot, channel, mode);
pac8084W.ReadFreq(slot, channel, ref Freq);
```

3.12 i8084W_ReadFreqInFloat

This function is used to read the frequency of the signal for a specific channel of the specified I-8084W module in Frequency mode. The frequency value is read in floating-point format. To achieve an accuracy of ±0.4% for a low frequency signal, such as 10 Hz, use the i8084W_ReadFreqInFloat function instead of i8084W_ReadFreq.

Prototype

For MiniOS7

int i8084W_ReadFreqInFloat(int slot, int channel, float * Freq);

For Windows (CE and WES)

int pac_i8084W_ReadFreqInFloat(int slot, int channel, float * Freq);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

**Freq: [out]*

A pointer to a value representing the signal frequency (in Hz).

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 2;    // Frequency mode
float Freq = 0;
i8084W_InitDriver(slot);
i8084W_SetChannelMode(slot, channel, mode);
i8084W_ReadFreqInFloat(slot, channel, &Freq);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 2;    // Frequency mode
float Freq = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetChannelMode(slot, channel, mode);
pac_i8084W_ReadFreqInFloat(slot, channel, &Freq);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
float freq = 0;
Int16 mode = 2; // Frequency mode
pac8084W.Init(slot);
pac8084W.SetChannelMode(slot, channel, mode);
pac8084W.ReadFreqFloat(slot, channel, ref freq);
```

3.13 i8084W_ClrCnt

This function is used to clear the counter value for a specific channel of the specified I-8084W module, regardless of the operation mode. Unless there is a particular reason not to do so, be sure to also clear the counter value for the other channel in the same set in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode.

Prototype

For MiniOS7

int i8084W_ClrCnt(int slot, int channel);

For Windows (CE and WES)

int pac_i8084W_ClrCnt(int slot, int channel);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be cleared. Valid range = 0 to 7.

Set	Channels	Remarks
1	0, 1	The mapping between sets and channels in Dir/Pulse Counting mode, Up/Down Counting mode, and Quadrant Counting mode.
2	2, 3	
3	4, 5	
4	6, 7	

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
i8084W_ClrCnt(slot, channel);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
pac_i8084W_ClrCnt(slot, channel);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
pac8084W.ClrCnt(slot, channel);
```

3.14 i8084W_ClrAllCnt

This function is used to clear the counter value for all channels of the specified I-8084W module, regardless of the operation mode.

Prototype

For MiniOS7

int i8084W_ClrAllCnt(int slot);

For Windows (CE and WES)

int pac_i8084W_ClrAllCnt(int slot);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
i8084W_ClrAllCnt(slot);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
```

```
pac_i8084W_ClrAllCnt(slot);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int channel = 0;
pac8084W.ClrAllCnt(slot);
```

3.15 i8084W_RecoverDefaultSetting

This function is used to restore the default settings for a specified I-8084W module.
The default settings are as follows:

- XOR Register = 0 (active high. The I-8084W counts when signal switches from Low to High)
- Operation Mode = 3 (Up Counting mode)
- Frequency Mode = 0 (Normal mode)
- Frequency Timeout Value = 1000 ms
- Low Pass Filter Status = 0 (Disabled)
- Low Pass Filter Signal Width = 1 us

Prototype

For MiniOS7

void i8084W_RecoverDefaultSetting(int slot);

For Windows (CE and WES)

void pac_i8084W_RecoverDefaultSetting(int slot);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

Return Values

This function does not return a value
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
i8084W_InitDriver(slot);
i8084W_RecoverDefaultSetting(slot);
```

[C++] Windows

```
int slot = 1;
pac_i8084W_InitDriver(slot);
pac_i8084W_RecoverDefaultSetting(slot);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
pac8084W.Init (slot);
pac8084W.RecoverDefaultSetting(slot);
```

3.16 i8084W_ReadXorRegister

This function is used to read the status of the XOR register for a specific channel of the specified I-8084W module. The XOR register determines whether or not the inputs need to be inverted. If XOR = 0, the inputs will not be inverted. If XOR = 1, the inputs will inverted.

Prototype

For MiniOS7

int i8084W_ReadXorRegister(int slot, int channel, int * XorReg);

For Windows (CE and WES)

int pac_i8084W_ReadXorRegister(int slot, int channel, int * XorReg);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

**XorReg: [out]*

A pointer to a value representing the status of the XOR register (0 or 1).
XorReg = 0: the counter is increased when the signal changes from Low to High.
XorReg = 1: the counter is increased when the signal changes from High to Low.

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int XorReg = 0;
i8084W_InitDriver(slot);
i8084W_ReadXorRegister(slot, channel, &XorReg);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int XorReg = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_ReadXorRegister(slot, channel, &XorReg);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
Int16 XorReg = 0;

pac8084W.Init(slot);
pac8084W.ReadXor(slot, channel, ref XorReg);
```

3.17 i8084W_SetXorRegister

This function is used to set the status of the XOR register for a specific channel of the specified I-8084W module. The XOR register determines whether or not the inputs need to be inverted. If XOR = 0, the inputs will not be inverted.
If XOR = 1, the inputs will be inverted.

Prototype

For MiniOS7

int i8084W_SetXorRegister(int slot, int channel, int XorReg);

For Windows (CE and WES)

int pac_i8084W_SetXorRegister(int slot, int channel, int XorReg);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

XorReg: [in]

Specifies the status of the XOR register (0 or 1).
XorReg = 0: the counter is increased when the signal changes from Low to High.
XorReg = 1: the counter is increased when the signal changes from High to Low.

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int XorReg = 0;
i8084W_InitDriver(slot);
i8084W_SetXorRegister(slot, channel, XorReg);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int XorReg = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetXorRegister(slot, channel, XorReg);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
Int16 XorReg = 1;
pac8084W.Init(slot);
pac8084W.SetXor(slot, channel, XorReg);
```

3.18 i8084W_ReadLowPassFilterUs

This function is used to read the width of the low pass filter for a specific channel of the specified I-8084W module in microseconds. As a maximum of three low pass filters can be set on each module, and the same filter can be shared by more than one channel, setting a low pass filter in one channel may cause the filters for other channel(s) to be changed.

Prototype

For MiniOS7

int i8084W_ReadLowPassFilterUs(int slot, int channel, unsigned int *Us);

For Windows (CE and WES)

int pac_i8084W_ReadLowPassFilterUs(int slot, int channel, unsigned int *Us);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

As shown in the following table, a filter can be shared by more than one channel. For example, channels 0 and 1 share the same Low Pass Filter. Therefore, changing the filter settings for channel 0 will also change the filter settings for channel 1.

Channel	Low Pass Filter
0 (A0)	Low Pass Filter 0
1 (B0)	
2 (A1)	Low Pass Filter 1
3 (B1)	
4 (A2)	Low Pass Filter 2
5 (B2)	
6 (A3)	

7 (B3)	

**Us: [out]*

A pointer to a value representing the width of the low pass filter in microseconds (1 to 32767).

H = the **HIGH** width of the input signal **T** = the period for the filtering clock	
Condition	**Filtering Status**
H > 2T	The signal is **PASSED**.
T ≤ H ≤ 2T	The signal may be **FILTERED** or **PASSED**.
H < T	The signal is **FILTERED**.

For example, set T = 1000 us and assume that the duty cycle of the input signal is 50% (high width = low width). Therefore, the minimum H = 2 * 1000 us, and the period for the input signal is 2 * H = 4000 us, meaning that the maximum value that will be allowed to pass by the low pass filter is 250 Hz.

Return Values

This function does not return a value

Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
unsigned int Us = 1000;
i8084W_InitDriver(slot);
i8084W_ReadLowPassFilterUs(slot, channel, &Us);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
unsigned int Us = 1000;
pac_i8084W_InitDriver(slot);
pac_i8084W_ReadLowPassFilterUs(slot, channel, &Us);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
UInt us = 1000;
pac8084W.Init (slot);
pac8084W.ReadLowPassFilter_Us(slot, channel, ref us);
```

3.19 i8084W_SetLowPassFilterUs

This function is used to set the width of the low pass filter for a specific channel of the specified I-8084W module in microseconds. As a maximum of three low pass filters can be set on each I-8084W module, setting a low pass filter in one channel may cause the filters for other channel(s) to be changed. This function works on **both** counting modes and the frequency mode.

Prototype

For MiniOS7

int i8084W_SetLowPassFilterUs(int slot, int channel, unsigned int Us);

For Windows (CE and WES)

int pac_i8084W_SetLowPassFilterUs(int slot, int channel, unsigned int Us);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

As shown in the following table, a filter can be shared by more than one channel. For example, channels 0 and 1 share the same Low Pass Filter. Therefore, changing the filter settings for channel 0 will also change the filter settings for channel 1.

Channel	Low Pass Filter
0 (A0)	Low Pass Filter 0
1 (B0)	
2 (A1)	Low Pass Filter 1
3 (B1)	
4 (A2)	Low Pass Filter 2

5 (B2)
6 (A3)
7 (B3)

Us: [in]

Specifies the width of the low pass filter in microseconds (1 to 32767).

H = the **HIGH** width of the input signal **T** = the period for the filtering clock	
Condition	**Filtering Status**
H > 2T	The signal is **PASSED**.
T ≤ H ≤ 2T	The signal may be **FILTERED** or **PASSED**.
H < T	The signal is **FILTERED**.

For example, set T = 1000 us and assume that the duty cycle of the input signal is 50% (high width = low width). Therefore,the minimum H = 2 * 1000 us, and the period for the input signal is 2 * H = 4000 us, meaning that the maximum value that will be allowed to pass by the low pass filter is 250 Hz.

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
unsigned int Us = 1000;
i8084W_InitDriver(slot);
i8084W_SetLowPassFilterUs(slot, channel, Us);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
unsigned int Us = 1000;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetLowPassFilterUs(slot, channel, Us);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int channel = 0;
UInt us = 1000;
Int16 status =1; // Enable Low Pass Filter
pac8084W.Init (slot);
pac8084W.SetLowPassFilter_Status(slot, channel, status);
pac8084W.SetLowPassFilter_Us(slot, channel, us);
```

3.20 i8084W_ReadLowPassFilterStatus

This function is used to read the status of the low pass filter for a specific channel of the specified I-8084W module.

Prototype

For MiniOS7

void i8084W_ReadLowPassFilterStatus(int slot, int channel, int * status);

For Windows (CE and WES)

void pac_i8084W_ReadLowPassFilterStatus(int slot, int channel, int * status);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

**status: [out]*

A pointer to a value representing the status of the low pass filter (0 or 1).
0: Disabled
1: Enabled

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int status = 0; // Disable Low Pass Filter
i8084W_InitDriver(slot);
i8084W_ReadLowPassFilterStatus(slot, channel, &status);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int status = 0; // Disable Low Pass Filter
pac_i8084W_InitDriver(slot);
pac_i8084W_ReadLowPassFilterStatus(slot, channel, &status);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int channel = 0;
Int16 status = 0; // Disable Low Pass Filter

pac8084W.Init(slot);
pac8084W.ReadLowPassFilter_Status(slot, channel, ref status);
```

3.21 i8084W_SetLowPassFilterStatus

This function is used to set the status of the low pass filter for a specific channel of the specified I-8084W module.

Prototype

For MiniOS7

void i8084W_SetLowPassFilterStatus(int slot, int channel, int status);

For Windows (CE and WES)

void pac_i8084W_SetLowPassFilterStatus(int slot, int channel, int status);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

status: [in]

Specifies the status to be set for the low pass filter (0 or 1).
0: Disabled
1: Enabled

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int status = 0; // Disable Low Pass Filter

i8084W_InitDriver(slot);
i8084W_SetLowPassFilterStatus(slot, channel, status);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int status = 0; // Disable Low Pass Filter

pac_i8084W_InitDriver(slot);
pac_i8084W_SetLowPassFilterStatus(slot, channel, status);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
Int16 status = 0; // Disable Low Pass Filter
pac8084W.Init(slot);
pac8084W.SetLowPassFilter_Status(slot, channel, status);
```

3.22 i8084W_ReadFreqMode

This function is used to read the frequency mode currently used by a specific channel of the specified I-8084W module.

Prototype

For MiniOS7

void i8084W_ReadFreqMode(int slot, int channel, int * mode);

For Windows (CE and WES)

void pac_i8084W_ReadFreqMode(int slot, int channel, int * mode);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

**mode: [out]*

A pointer to a value representing the currently used frequency mode (0 or 1).
0: Normal mode
1: High Speed mode (Average mode)

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 0;
i8084W_InitDriver(slot);
i8084W_ReadFreqMode(slot, channel, &mode);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_ReadFreqMode(slot, channel, &mode);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
Int16 mode =0;

pac8084W.Init(slot);
pac8084W.ReadFreqMode(slot, channel, ref mode);
```

3.23 i8084W_SetFreqMode

This function is used to set the frequency mode to be used by a specific channel of the specified I-8084W module. Set the frequency mode to High Speed mode when measuring high frequencies such as 10 kHz.

Prototype

For MiniOS7

void i8084W_SetFreqMode(int slot, int channel, int mode);

For Windows (CE and WES)

void pac_i8084W_SetFreqMode(int slot, int channel, int mode);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

mode: [in]

Specifies the frequency mode to be used (0 or 1).
0: Normal mode. The I-8084W will only measure the frequency once.
1: High Speed mode (Average mode). The I-8084W will measure the frequency 11 times and then calculate the average value.

For more details regarding frequency accuracy, refer to Appendix B.

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
int mode = 0; // Normal mode
i8084W_InitDriver(slot);
i8084W_SetFreqMode(slot, channel, mode);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
int mode = 0; // Normal mode
pac_i8084W_InitDriver(slot);
pac_i8084W_SetFreqMode(slot, channel, mode);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int channel = 0;
Int16 freqMode=0; // Normal mode
pac8084W.Init(slot);
pac8084W.SetFreqMode(slot, channel, mode);
```

3.24 i8084W_ReadFreqTimeoutValue

This function is used to read the timeout value used when reading the frequency for a specific channel of the specified I-8084W module. Frequency measurement is achieved by calculating the values measured within a certain period i.e., the timeout value. When reading low frequencies, such as 1 Hz, the timeout value may need to be increased in order to ensure sufficient measurements are performed to calculate the frequency.

Prototype1

For MiniOS7

unsigned short i8084W_ReadFreqTimeoutValue(int slot, int channel);

For Windows (CE and WES)

unsigned short pac_i8084W_ReadFreqTimeoutValue(int slot, int channel);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

Return Values

The frequency timeout value in milliseconds.
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
unsigned short timeout = 0;
```

```
i8084W_InitDriver(slot);
timeout = i8084W_ReadFreqTimeoutValue(slot, channel);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
unsigned short timeout = 0;
pac_i8084W_InitDriver(slot);
timeout = pac_i8084W_ReadFreqTimeoutValue(slot, channel);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
UInt16 timeout = 0;

pac8084W.Init(slot);
timeout = pac8084W.ReadFreqTimeoutValue(slot, channel);
```

3.25 i8084W_SetFreqTimeoutValue

This function is used to set the timeout value used when reading the frequency for a specific channel of the specified I-8084W module. Frequency measurement is achieved by calculating the values measured within a certain period i.e., the timeout value. When reading low frequencies such as those around 1 Hz, the timeout value may need to be increased in order to ensure sufficient measurements are performed to calculate the frequency.

Prototype

For MiniOS7

void i8084W_SetFreqTimeoutValue(int slot, int channel, unsigned short timeout);

For Windows (CE and WES)

Void pac_i8084W_SetFreqTimeoutValue(int slot, int channel, unsigned short timeout);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

channel: [in]

Specifies the channel to be read. Valid range = 0 to 7.

timeout: [in]

Specifies the frequency timeout value in milliseconds.

For more details regarding the necessary timeout values for frequency mode, refer to Appendix B.

Return Values

This function does not return a value
Refer to Appendix A: “Error Code Definitions” for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int channel = 0;
unsigned short timeout = 1000;
i8084W_InitDriver(slot);
i8084W_SetFreqTimeoutValue(slot, channel, timeout);
```

[C++] Windows

```
int slot = 1;
int channel = 0;
unsigned short timeout = 1000;
pac_i8084W_InitDriver(slot);
pac_i8084W_SetFreqTimeoutValue(slot, channel, timeout);
```

[C#] Windows

```
using pac8084WNet;
…
int slot = 1;
int channel = 0;
ushort timeout = 0;

pac8084W.Init(slot);
pac8084W.SetFreqTimeoutValue(slot, channel, timeout);
```

3.26 i8084W_ReadDIXor

This function is used to read the Digital Input value after the XOR operations have been applied on the specified I-8084W module.

Prototype

For MiniOS7

int i8084W_ReadDIXor(int slot, int * DI);

For Windows (CE and WES)

int pac_i8084W_ReadDIXor(int slot, int * DI);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

**DI: [out]*

A pointer to a value representing the DI value. Valid range = 0 to 255.
The DI value is an integer whose hexadecimal form represents the ON/OFF status of 8 channels. For example, if channels 0 and 1 are ON and all other channels are OFF, the DI value will be 0x03.

Return Values

This function does not return a value
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int DI = 0;
i8084W_InitDriver(slot);
```

```
i8084W_ReadDIXor(slot, &DI);
```

[C++] Windows

```
int slot = 1;
int DI = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_ReadDIXor(slot, &DI);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int DI = 0;

pac8084W.Init(slot);
pac8084W.ReadDI_Xor(slot, ref DI);
```

3.27 i8084W_ReadDIXorLPF

This function is used to read the Digital Input value after both the XOR and Low Pass Filter operations have been applied on the specified I-8084W module.

Prototype

For MiniOS7

int i8084W_ReadDIXorLPF(int slot, int * DI);

For Windows (CE and WES)

int pac_i8084W_ReadDIXorLPF(int slot, int * DI);

Parameters

slot: [in]

Specifies the slot where the I-8084W module is inserted. Valid range = 0 to 7.

**DI: [out]*

A pointer to a value representing the DI value. Valid range = 0 to 255.
The DI value is an integer whose hexadecimal form represents ON/OFF status of 8 channels. For example, if channels 0 and 1 are ON and all other channels are OFF, the DI value will be 0x03.

Return Values

This function does not return a value
Refer to Appendix A: "Error Code Definitions" for details of other returned values.

Examples

[C++] MiniOS7

```
int slot = 1;
int DI = 0;
i8084W_InitDriver(slot);
```

```
i8084W_ReadDIXorLPF(slot, &DI);
```

[C++] Windows

```
int slot = 1;
int DI = 0;
pac_i8084W_InitDriver(slot);
pac_i8084W_ReadDIXorLPF(slot, &DI);
```

[C#] Windows

```
using pac8084WNet;
...
int slot = 1;
int DI = 0;

pac8084W.Init(slot);
pac8084W.ReadDI_XorLPF(slot, ref DI);
```

Appendix A: Error Code Definitions

Programming or operation errors usually occur when illegal data exists in a message or there was a problem communicating with a device, resulting in an exception response from either the master or the slave device, depending on the type of error. When a slave detects one of these errors, it sends a response message to the master consisting of the slave address, function code, error code and error check fields. The exception response codes and their definitions are listed below.

Error Code	Definition	Description
0	No Error	The command was successful.
-1	ID_ERROR	There was an error accessing the module ID. Verify the ID and try again
-2	SLOT_OUT_RANGE	The Slot index value is out of range. The valid range is 0 to 7.
-3	CHANNEL_OUT_RANGE	The Channel index value is out of range. The valid range is 0 to 15.
-4	SELECT_CHANNEL_ERROR	An error occurred when accessing the channel. The valid range is 0 to 7.
-5	ADDRESS_ERROR	The Address is out of range. The valid range is 0 to 63.
-6	MODE_ERROR	The Mode value is out of range. The valid range is 0 to 4.
-10	DATA_ERROR	A data error has occurred or the parameters are incorrect
-15	Timeout	The timeout for reading the frequency has expired.

Appendix B: Accuracy value and necessary update time for frequency mode

The following is an overview of the required update time and the relevant accuracy values when performing frequency measurement in both Normal mode and High-speed mode.

Normal Mode: (Sample pulse: 1)

Frequency (Hz)	Update time (ms)	measurement max value(Hz)	measurement min value (Hz)	Inaccuracy for max value (%)	Inaccuracy for min value (%)
10	100.00	10.001	9.998	0.01	0.02
900	1.11	900.195	899.758	0.02	0.03
4000	0.25	4000.960	3998.880	0.02	0.03
10000	0.10	10003.001	9997.001	0.03	0.03
39000	0.03	39016.777	38971.160	0.04	0.07
40000	0.03	40016.008	39984.008	0.04	0.04
45000	0.02	45024.762	44964.027	0.06	0.08
80000	0.01	80064.055	79936.055	0.08	0.08
129000	0.01	129032.258	128865.977	0.03	0.10
130000	0.01	130208.336	129870.133	0.16	0.10
200000	0.01	200400.797	199600.797	0.20	0.20
250000	0.00	250626.562	249376.562	0.25	0.25
300000	0.00	300300.312	299401.188	0.10	0.20

Inaccuracy value = (Frequency-measurement value)/ Frequency*100

Note: The frequency measurements indicated in this table were produced using an Agilent 33220A Function Generator.

High Speed Mode: (Sample pulse: 11)

Frequency (Hz)	Update time (ms)	measurement max value(Hz)	measurement min value (Hz)	Inaccuracy for max value (%)	Inaccuracy for min value (%)
10	1100.00	10.000	10.000	0.00	0.00
900	12.22	899.996	899.957	0.00	0.00
4000	2.75	4000.000	3999.782	0.00	0.01
10000	1.10	10000.000	9999.454	0.00	0.01
39000	0.28	39001.559	38996.031	0.00	0.01
40000	0.28	40001.453	39997.090	0.00	0.01
45000	0.24	45000.816	44995.297	0.00	0.01
80000	0.14	80000.000	79994.180	0.00	0.01
129000	0.09	129017.125	128986.817	0.01	0.01
130000	0.08	130008.273	129977.547	0.01	0.02
200000	0.06	200036.375	199963.641	0.02	0.02
250000	0.04	250056.828	249943.188	0.02	0.02
300000	0.04	300054.564	299890.938	0.02	0.04

Inaccuracy value = (Frequency-measurement value)/ Frequency*100

Note: The frequency measurements indicated in this table were produced using an Agilent 33220A Function Generator.

Appendix C: Revision Information

Version	Date	Comments
V2.0.0		Added API functions
V1.0.0		First Release