PMDK

Basic Functions Reference Manual

Language	English
Version	V1.2
Date	2012/12/01

ICP DAS CO., LTD.

泓格科技股份有限公司

Important Notices

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, not 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.

Index

1 Introduction

The PMDK (Professional Motion Development Kit) is designed for professional motion development. It integrates a high-speed floating DSP (TI C672x), FPGA (Field Programmable Gate Array), I/O buffering circuitries, and application software samples. It has plenty of different I/O interfaces. There are six channels of pulse inputs and outputs, six channels of analog inputs and outputs, and many digital inputs and outputs on this card for users to realize their own applications. PMDK can be used on any IPC with a PCI bus. Together with the DN-20M extension board which supports one 2-wired FRnet port this card can control up to 128 DI and 128 DO remotely.

For motion control, ICP DAS can provide a variety of motion control functions, such as multi-axis linear and circular interpolations with acceleration/deceleration processing. Various synchronous actions are possible through programming. Users can refer to sample programs and design their own motion functions; these functions can be appended to the old motion command set. DSP program is developed based on a real-time kernel (DSP/BIOS); therefore, motion status, FRnet I/O, and other I/O can still be monitored while driving. As a result of the low CPU loading on IPC, one or more motion cards can be used on a single IPC. ICP DAS will provide more and more functions and examples to reduce the programming load of users.

When PMDK is used for signal processing, ICP DAS provide samples to show how to do FFT, FIR, and IIR. Users can also refer to the resources provided by TI. ICP DAS would like to make the PMDK a highly cost-effective solution for users to develop applications for motion control, process control, I/O logical control, digital processing, and other domains.

Main Features

- DSP-based control card with PCI interface
- Capable for 6-axis motion control
- Maximum pulse output frequency: 4 Mpps
- Maximum Encoder input frequency: 12 Mpps
- High-speed position latching and comparing functions
- Home, positive and negative limit sensors for each axis manual-pulse-generator interface
- Expandable remote I/O: 128 DI & 128 DO via two-wired FRnet interface

Application

- Advanced multi-axis motion control development
- Signal processing
- I/O logical control
- Training and education for multiple purposes
- High speed encoder interface for laser encoders

- Coordinate measurement machine
- Robotics

Optional Accessories (not included in the PMDK package):

DN-8368GB	Three axis daughter board for general purpose amplifier
DN-8368MB	Three axis daughter board for **Mitsubishi J2S amplifier**.
DN-20M	Daughter board for FRnet, manual pulse generator and general purpose digital input
CA-MINI68-15	Mini SCSI II 68-pin to SCSI II 68-pin & 68-pin Male connector cable 1.5M
CA-SCSI20-M1	SCSI II 20-pin & 20-pin Male connector cable 1 meter , for – DN-20M board – Mitsubishi J2S amplifier
CA-SCSI20-M3	SCSI II 20-pin & 20-pin Male connector cable 3 meter, for – DN-20M board – Mitsubishi J2S amplifier
CA-SCSI20-M5	SCSI II 20-pin & 20-pin Male connector cable 5 meter, for – DN-20M board – Mitsubishi J2S amplifier
CA-2P4C-0100	The cable for FRnet DIO modules; Length:100m

1.1 Features

- 6-axis DSP based Voltage/Pulse command motion card
- 32-Bits/33MHz universal PCI bus
- 6-axis full closed loop with P/FF control in position loop and PI/FF in velocity loop (optional)
- 250 us servo update rate (configurable)
- Input/Output signals for each axis
    - 1 channel 16-bits +/-10V analog output
    - 1 channel 16-bits +/-10V analog input
    - 1 channel high speed pulse output
    - 32-bits high speed quadrature encoder (A, B, Z) interface
    - Positive and negative limit switch input
    - Home switch input
    - Slow down switch input
    - Servo alarm input
    - Servo in-position input
    - Servo ready input
    - High speed position latch input
    - Servo on output
    - Error counter clear output
    - Servo alarm reset output
    - High speed auto incremental compare output
- Dedicated 1 channel emergency stop input
- Support full function manual pulse generator (with 1 A/B phase input, 6-axis inputs, 3 magnification inputs, 2 jog inputs and 1 EMG inputs)
- Additional 1 channel general purpose input and 3 channel general purpose outputs
- Programmable digital filter for all input signals
- 1 port FRnet real-time/high-speed serial interface for I/O expansion (128 in/128 out)
- 5V TTL UART interface or inter-board synchronization interface (optional)

2 PMDK Specifications

2.1 Technical Specifications

The following describe PMDK specification together with terminal boards DN-8368GB and DN-8368MB.

2.1.1 General Specification:

- On board high performance floating-point DSP with 200MHz / 1600MIPS / 1200MFLOPS
- On board 128KB nonvolatile FRAM
- I/O connector: Two 68-pins VHDCI female connectors and one 20-pins SCSI-II female connector
- Operating temperature: 0 to 60 ℃
- Storage temperature: -20 to 80 ℃
- Operating humidity: 10% to 85%, non-condensing
- Storage humidity: 5% to 95%, non-condensing

2.1.2 Analog Output

- Number of Channels: 6
- Simultaneous update of all axes
- On board auto calibration
- Resolution: 16 bits
- Output Voltage Range: +/-10V
- Output Current: Single channel +/-40mA max.
- Settling Time: 2.5us typ. (to +/-0.1%, -10V to +10V step)
- Slew Rate: 10V/us typ.
- Power On State: 0V steady-state
- Power On Glitch: None
- Differential Non-Linearity (DNL): +/-1LSB max.

2.1.3 Analog Input

- Number of Channels: 6
- On board auto calibration
- Maximum Sampling Rate: All channels 250kS/s
- Resolution: 16 bits, No missing codes
- Input Voltage Range: +/-10V
- Over-voltage Protection: 70Vpp

2.1.4 Pulse Output

- Number of Channels: 6
- Isolation Voltage: 3kVrms
- Max. Output Pulse Rate: 4Mhz
- Programmable OUT/DIR, CW/CCW and EA/EB output modes
- Programmable high pulse width or 50% duty cycle
- Selectable line drive or open collector output

2.1.5 Encoder Interface

- Number of Channels: 6
- Isolation Voltage: 3kVrms
- Max. Count Rate: 12Mhz
- Programmable OUT/DIR, CW/CCW and EA/EB input modes
- Min. turn on voltage: 3V max.

2.1.6 Manual Pulse Generator Interface

- Number of Channels: 1
- Isolation Voltage: 3kVrms
- Max. Count Rate: 40Kpps
- Programmable OUT/DIR, CW/CCW and EA/EB input modes
- Selectable 0 to 30 VDC input or 0 to 5 VDC input

2.1.7 Digital Input

- Number of Channels
    - 42-CH dedicated inputs for 6-axis LMT+/LMT-/HOME/SLD/ALARM/INP/ RDY
    - 1-CH dedicated input for emergency stop
    - 1-CH general purpose digital input
    - 6-CH inputs for axis-selections in MPG mode
    - 3-CH inputs for magnification selection in MPG mode
    - 2-CH inputs for jog operation in MPG mode
    - 1-CH input for optional emergency stop in MPG mode
- Isolation Voltage: 3kVrms
- Input Voltage: 0 to 30 VDC
    - Logic Low: 3V max.
    - Logic High: 10V min.
- Input Impedance: 4.7kΩ, 0.25W
- Input Turn-On Delay: 4 us type.
- Input Turn-Off Delay: 75 us type.

2.1.8 High Speed Input

- Number of Channels: 6-CH inputs for 6-axis LTC
- Isolation Voltage: 3kVrms
- Input Voltage:
    - 0 to 30 VDC

        - Logic Low: 3V max.
        - Logic High: 10V min.
    - 0 to 5 VDC
        - Logic Low: 1.5V max.
        - Logic High: 3V min.
- Input Impedance
    - 0 to 30 VDC: 4.7kΩ, 0.25W
    - 0 to 5 VDC: 220Ω, 0.125W
- Input Turn-On Delay:
    - 0 to 30 VDC: 50 ns type
    - 0 to 5 VDC: 50 ns type
- Input Turn-Off Delay:
    - 0 to 30 VDC: 300 ns type
    - 0 to 5 VDC: 150 ns type

2.1.9 Digital Output

- Number of Channels
    - 18-CH open collector outputs for 6-axis SRV_ON/ERC/ALM_RST
    - 3-CH general purpose open collector digital outputs
- Isolation Voltage: 3kVrms
- Output Current: 20mA max.
- Output Turn-On Delay: 7 us typ. @ ILOAD=5mA
- Output Turn-Off Delay: 200 us typ. @ ILOAD=5mA

2.1.10 High Speed Output

- Number of Channels: 6-CH open collector outputs for 6-axis CMP (with 390R pull-up to 5V)
- Isolation Voltage: 3kVrms
- Output Turn-On Delay: 40 ns type
- Output Turn-Off Delay: 60 ns type
- Maximum compare trigger output rate:
    - 2000Hz in buffer mode (Arbitrary Spacing)
    - 4MHz in auto increment mode (Constant Spacing)
- Compare trigger output pulse width: Programmable (160ns to 2.6ms)

2.1.11 FRnet Interface

- 2-Wire (wire saving serial interface)
- Max. Expandable I/O: 128 DI and 128 DO
- Scan Cycle Time: 0.72 ms
- Max. Distance: 100 M

2.2 Pin Definition

The PMDK card has got the following ports (Figure 1):

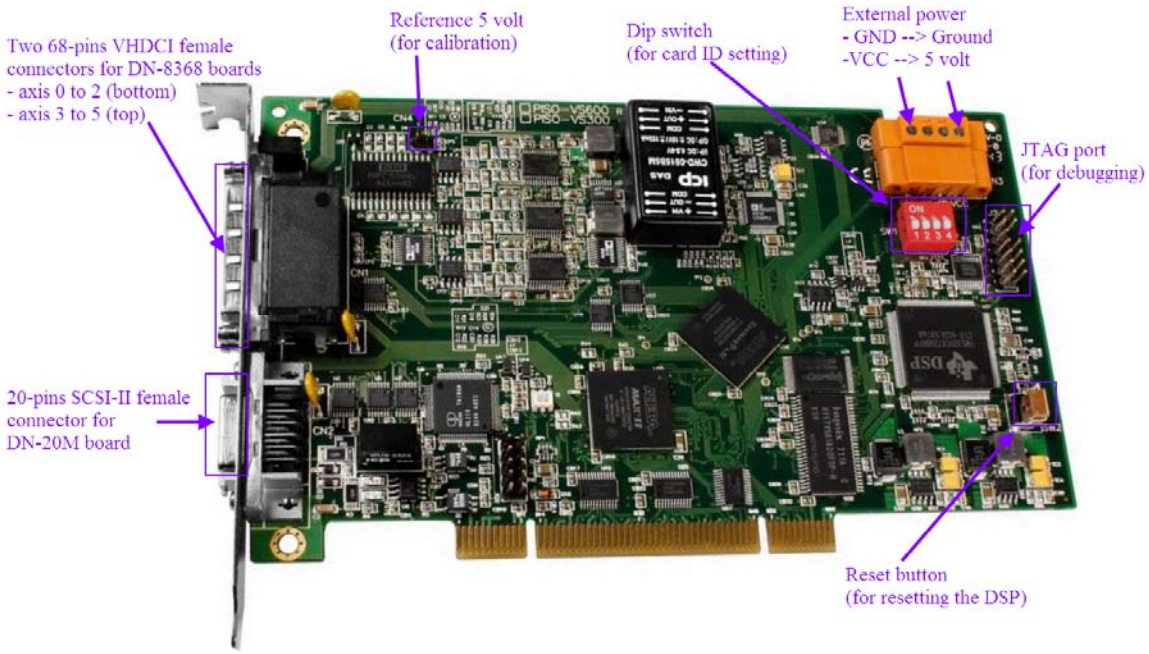


Figure 1: Port definitions of the PMDK card

1. Two 68-pin VHDCI female connectors for two DN-3868 boards

No.	Name	I/O	Function Axis	No.	Name	I/O	Function Axis
1	AOUT0	O	Analog Output	**35**	AIN0	I	Analog Input
2	AOUT1	O	Analog Output	**36**	AIN1	I	Analog Input
3	AOUT2	O	Analog Output	**37**	AIN2	I	Analog Input
4	AGND	-	Analog Ground	**38**	AGND	-	Analog Ground
5	DGND	-	Digital Ground	**39**	ERC0	O	Error Counter Clear
6	LTC0	I	Position Latch	**40**	SVON0	O	Servo On
7	EA0	I	Encoder A-Phase	**41**	RDY0	I	Servo Ready
8	EB0	I	Encoder B-Phase	**42**	INP0	I	Servo In-Position
9	EZ0	I	Encoder Z-Phase	**43**	ALM0	I	Servo Alarm
10	CW0	O	Clockwise pulse	**44**	SLD0	I	Slow Down
11	CCW0	O	Counter-Clockwise pulse	**45**	ORG0	I	Origin Signal
12	CMP0	O	Compare Trigger	**46**	MEL0	I	Minus End Limit

13	EMG	I	Emergency Stop	**47**	PEL0	I	Positive End Limit
14	ALMRST0	O	Servo Alarm Reset	**48**	DGND	-	Digital Ground
15	DGND	-	Digital Ground	**49**	ERC1	O	Error Counter Clear
16	LTC1	I	Position Latch	**50**	SVON1	O	Servo On
17	EA1	I	Encoder A-Phase	**51**	RDY1	I	Servo Ready
18	EB1	I	Encoder B-Phase	**52**	INP1	I	Servo In-Position
19	EZ1	I	Encoder Z-Phase	**53**	ALM1	I	Servo Alarm
20	CW1	O	Clockwise pulse	**54**	SLD1	I	Slow Down
21	CCW1	O	Counter-Clockwise pulse	**55**	ORG1	I	Origin Signal
22	CMP1	O	Compare Trigger	**56**	MEL1	I	Minus End Limit
23	GDO1	O	Generic Digital Output	**57**	PEL1	I	Positive End Limit
24	ALMRST1	O	Servo Alarm Reset	**58**	DGND	-	Digital Ground
25	DGND	-	Digital Ground	**59**	ERC2	O	Error Counter Clear
26	LTC2	I	Position Latch	**60**	SVON2	O	Servo On
27	EA2	I	Encoder A-Phase	**61**	RDY2	I	Servo Ready
28	EB2	I	Encoder B-Phase	**62**	INP2	I	Servo In-Position
29	EZ2	I	Encoder Z-Phase	**63**	ALM2	I	Servo Alarm
30	CW2	O	Clockwise pulse	**64**	SLD2	I	Slow Down
31	CCW2	O	Counter-Clockwise pulse	**65**	ORG2	I	Origin Signal
32	CMP2	O	Compare Trigger	**66**	MEL2	I	Minus End Limit
33	DGND	-	Digital Ground	**67**	PEL2	I	Positive End Limit
34	ALMRST2	O	Servo Alarm Reset	**68**	VCC	-	5V Digital Power from Bus

No.	Name	I/O	Function Axis	No.	Name	I/O	Function Axis
1	AOUT3	O	Analog Output	**35**	AIN3	I	Analog Input
2	AOUT4	O	Analog Output	**36**	AIN4	I	Analog Input
3	AOUT5	O	Analog Output	**37**	AIN5	I	Analog Input
4	AGND	-	Analog Ground	**38**	AGND	-	Analog Ground

5	DGND	-	Digital Ground	**39**	ERC3	O	Error Counter Clear
6	LTC3	I	Position Latch	**40**	SVON3	O	Servo On
7	EA3	I	Encoder A-Phase	**41**	RDY3	I	Servo Ready
8	EB3	I	Encoder B-Phase	**42**	INP3	I	Servo In-Position
9	EZ3	I	Encoder Z-Phase	**43**	ALM3	I	Servo Alarm
10	CW3	O	Clockwise pulse	**44**	SLD3	I	Slow Down
11	CCW3	O	Counter-Clockwise pulse	**45**	ORG3	I	Origin Signal
12	CMP3	O	Compare Trigger	**46**	MEL3	I	Minus End Limit
13	GDI11	I	Generic Digital Input	**47**	PEL3	I	Positive End Limit
14	ALMRST3	O	Servo Alarm Reset	**48**	DGND	-	Digital Ground
15	DGND	-	Digital Ground	**49**	ERC4	O	Error Counter Clear
16	LTC4	I	Position Latch	**50**	SVON4	O	Servo On
17	EA4	I	Encoder A-Phase	**51**	RDY4	I	Servo Ready
18	EB4	I	Encoder B-Phase	**52**	INP4	I	Servo In-Position
19	EZ4	I	Encoder Z-Phase	**53**	ALM4	I	Servo Alarm
20	CW4	O	Clockwise pulse	**54**	SLD4	I	Slow Down
21	CCW4	O	Counter-Clockwise pulse	**55**	ORG4	I	Origin Signal
22	CMP4	O	Compare Trigger	**56**	MEL4	I	Minus End Limit
23	GDO2	O	Generic Digital Output	**57**	PEL4	I	Positive End Limit
24	ALMRST4	O	Servo Alarm Reset	**58**	DGND	-	Digital Ground
25	DGND	-	Digital Ground	**59**	ERC5	O	Error Counter Clear
26	LTC5	I	Position Latch	**60**	SVON5	O	Servo On
27	EA5	I	Encoder A-Phase	**61**	RDY5	I	Servo Ready
28	EB5	I	Encoder B-Phase	**62**	INP5	I	Servo In-Position
29	EZ5	I	Encoder Z-Phase	**63**	ALM5	I	Servo Alarm
30	CW5	O	Clockwise pulse	**64**	SLD5	I	Slow Down

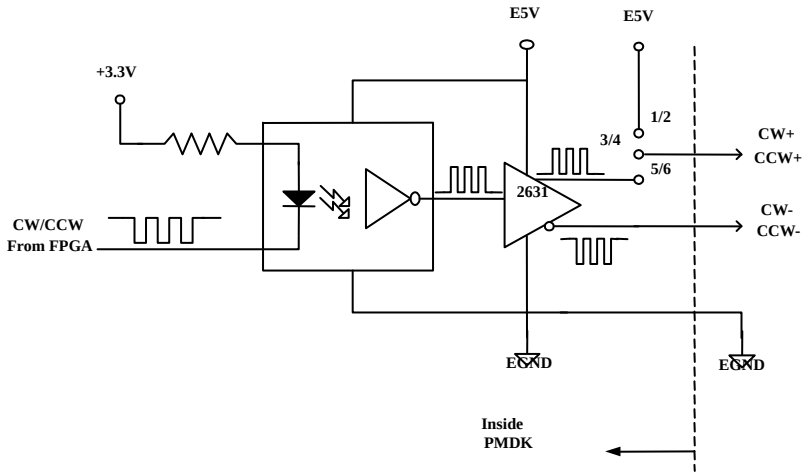
31	CCW5	O	Counter-Clockwise pulse	**65**	ORG5	I	Origin Signal
32	CMP5	O	Compare Trigger	**66**	MEL5	I	Minus End Limit
33	DGND	-	Digital Ground	**67**	PEL5	I	Positive End Limit
34	ALMRST5	O	Servo Alarm Reset	**68**	VCC	-	5V Digital Power from Bus

2. 20-pin SCSI-II female connector for the DN-20M board

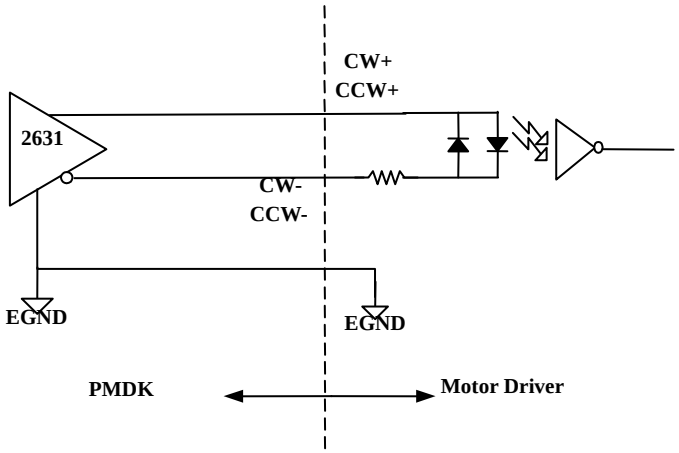
No.	Name	I/O	Function	No.	Name	I/O	Function
1	FR_A	O	FRnet A Phase	**11**	DGND	-	Digital Ground
2	FR_B	O	FRnet B Phase	**12**	MPG_EMG	I	MPG Emergency Stop
3	FR_GND	-	GND of FRnet Signal	**13**	GDI10 / MPG_Axis5	I	Generic Digital Input or MPG Axis Selection
4	GDI9 / MPG_Axis4	I	Generic Digital Input or MPG Axis Selection	**14**	GDI8 / MPG_Axis3	I	Generic Digital Input or MPG Axis Selection
5	GDI7 / MPG_Axis2	I	Generic Digital Input or MPG Axis Selection	**15**	GDI6 / MPG_Axis1	I	Generic Digital Input or MPG Axis Selection
6	GDI5 / MPG_Axis0	I	Generic Digital Input or MPG Axis Selection	**16**	GDI4 / MPG_Gain2	I	Generic Digital Input or MPG Magnification Selection
7	GDI3 / MPG_Gain1	I	Generic Digital Input or MPG Magnification Selection	**17**	GDI2 / MPG_Gain0	I	Generic Digital Input or MPG Magnification Selection
8	GDI1	I	Generic Digital Input	**18**	GDI0	I	Generic Digital Input
9	MPG_B	I	Manual Pulse Generator B Phase	**19**	MPG_A	I	Manual Pulse Generator A Phase
10	GDO0	O	Generic Digital Output	**20**	VCC	-	5V Digital Power from Bus

2.3 Signal Connection

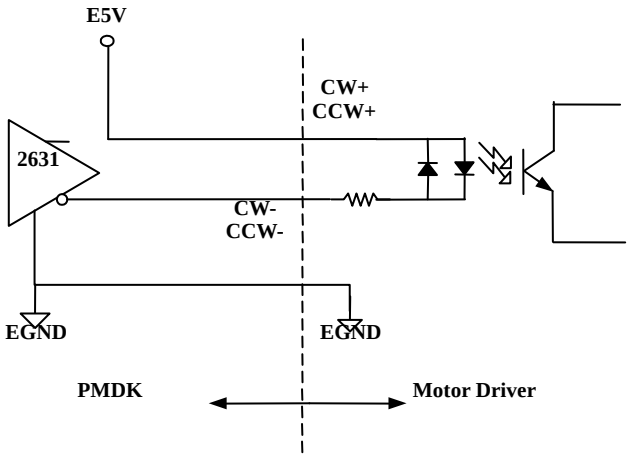
2.3.1 Pulse Output Signal



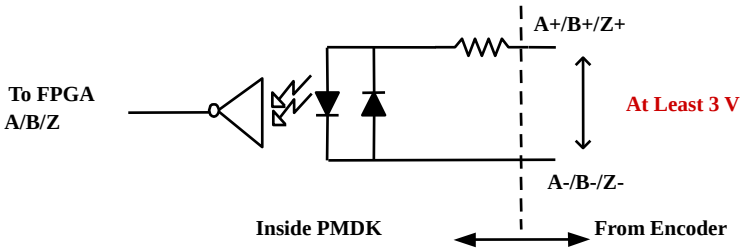
- Differential Type (Set Jumper as 3-5, 4-6)



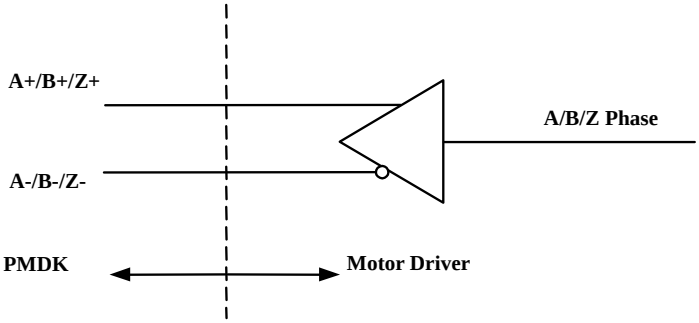
- Open Collector Type (Set Jumper as 1-3, 2-4)



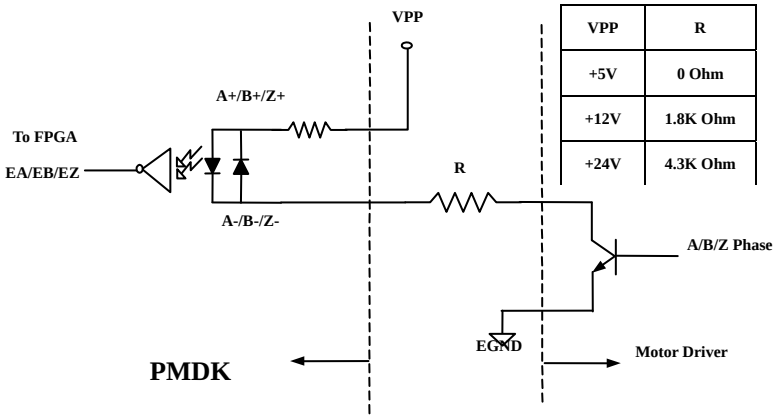
2.3.2 Encoder and MPG Input Signal



- Connect to a Differential type Encoder or MPG

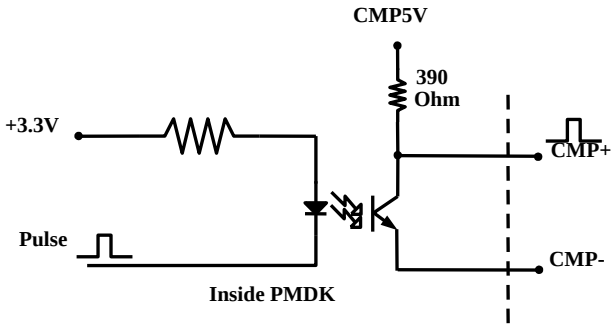


- Connect to a Open Collector type Encoder or MPG



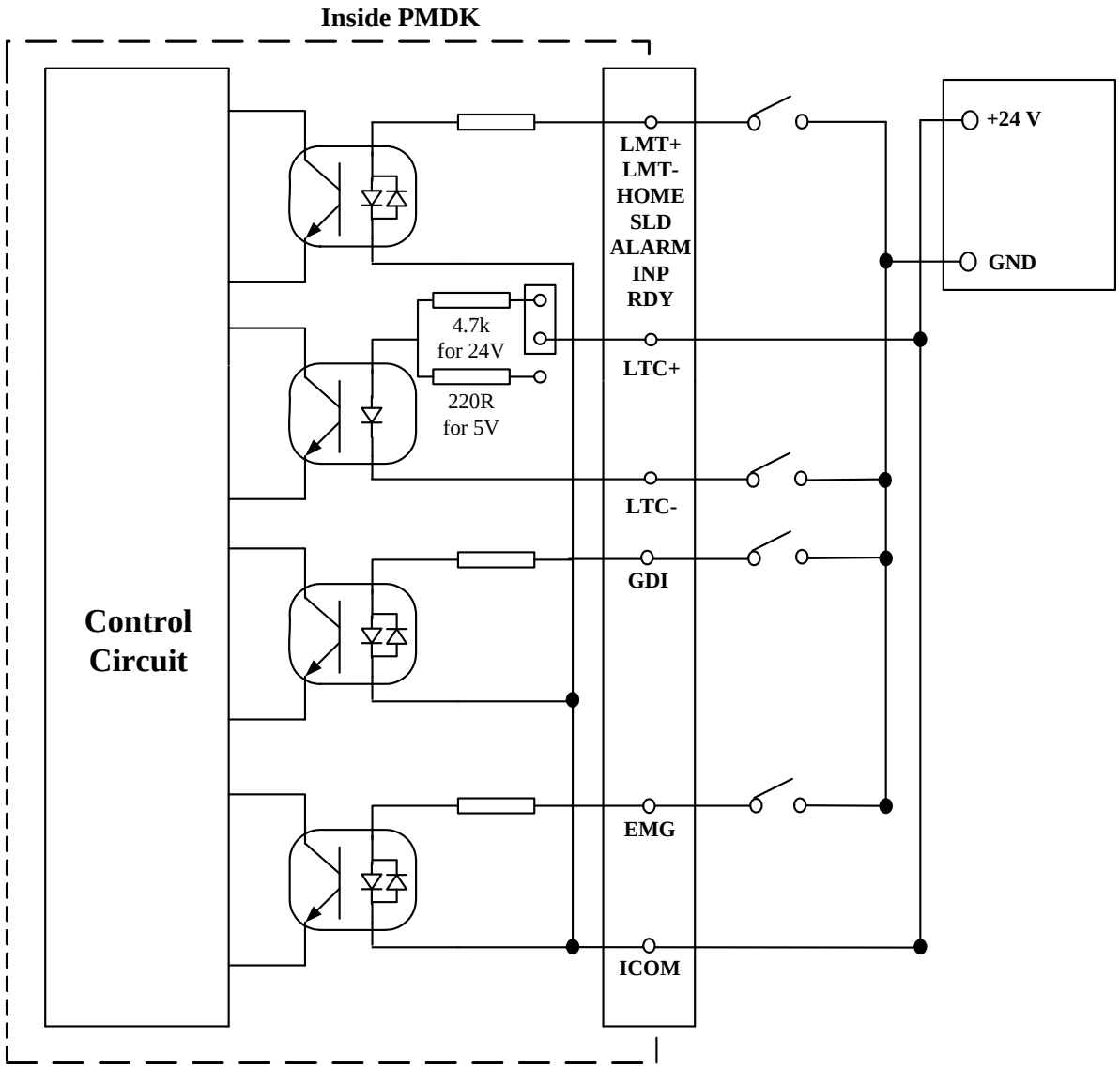
VPP	R
+5V	0 Ohm
+12V	1.8K Ohm
+24V	4.3K Ohm

2.3.3 Compare Output

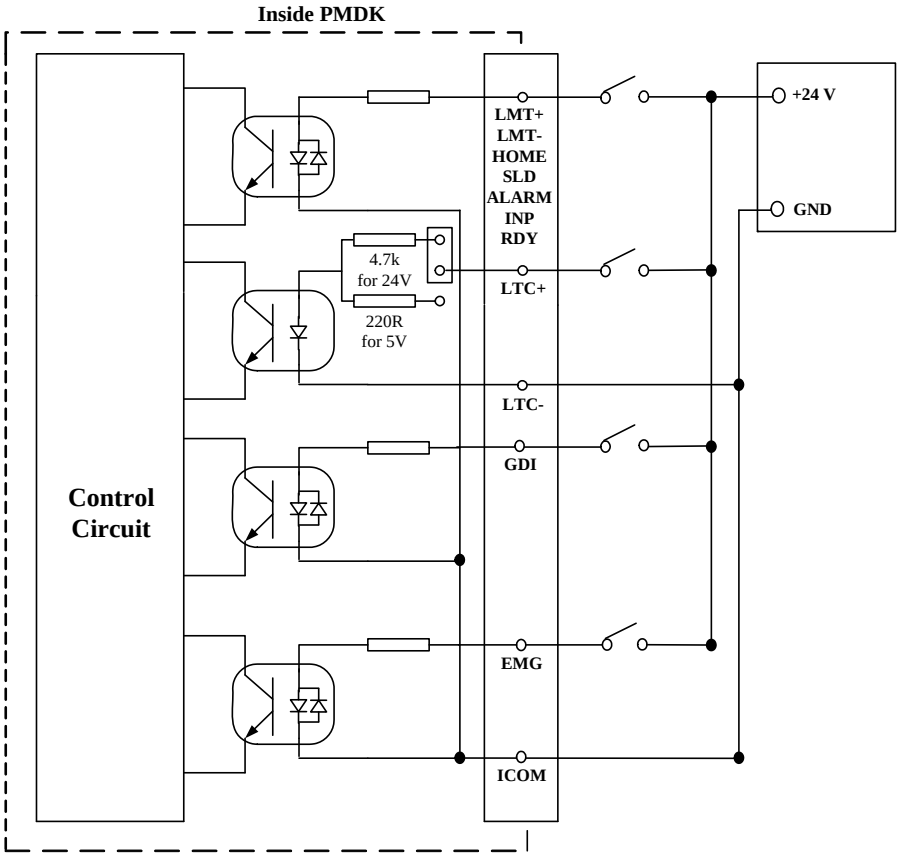


2.3.4 Digital Input

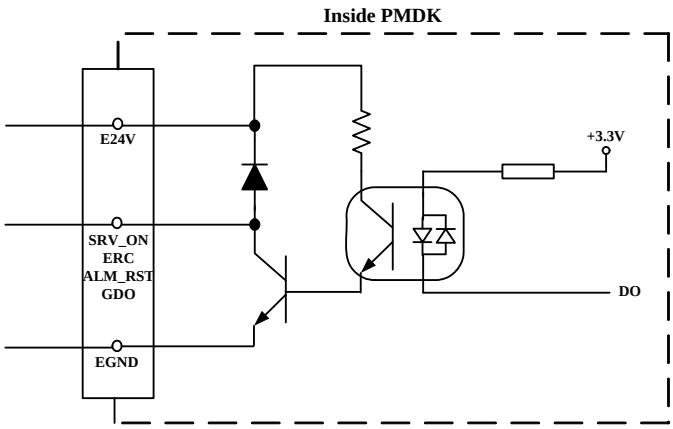
- NPN Input for NPN Sinking Type Sensor



- PNP Input for PNP Sourcing Type Sensor



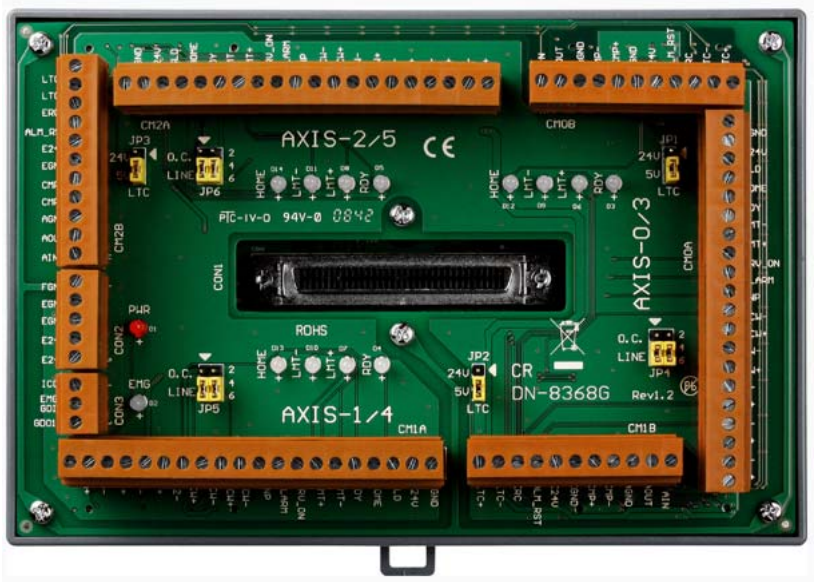
2.3.5 Digital Output

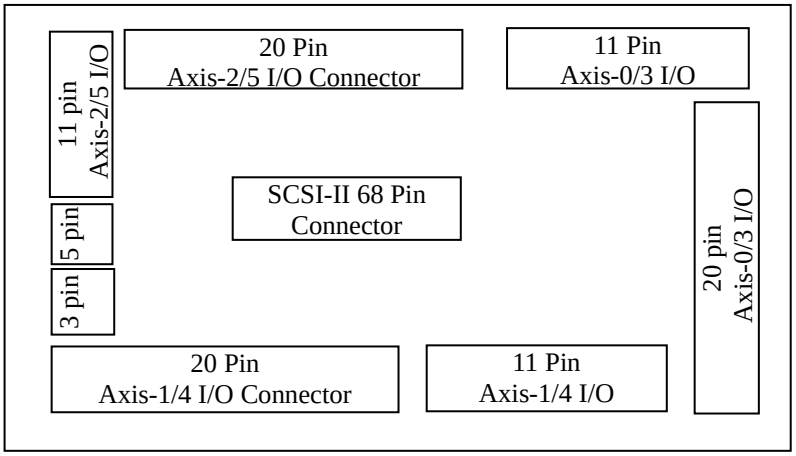


2.4 Terminal Boards

2.4.1 DN-8368GB, DN-8368MB Board

3 Axes termination board with one 68-pin SCSI-II connector





- 20 pin connector:

No.	Name	No.	Name
1	A+	**11**	INP
2	A-	**12**	ALARM
3	B+	**13**	SRV_ON
4	B-	**14**	LMT+
5	Z+	**15**	LMT-
6	Z-	**16**	RDY
7	CW+	**17**	HOME
8	CW-	**18**	SLD
9	CCW+	**19**	E24V
10	CCW-	**20**	EGND

- 11 pin connector:

No.	Name	No.	Name
1	LTC+	7	CMP+
2	LTC-	8	CMP-
3	ERC	9	AGND
4	ALM_RST	10	AOUT
5	E24V	11	AIN
6	EGND		

- 5 pin connector:

No.	Name
1	FGND
2	EGND
3	EGND
4	E24V
5	E24V

- 3 pin connector:

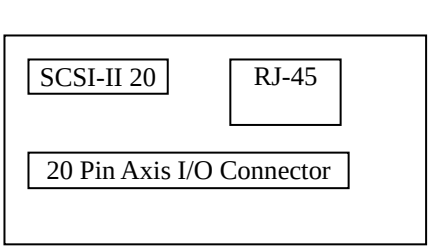
No.	Name
1	ICOM
2	EMG/GDI11
3	GDO1/2

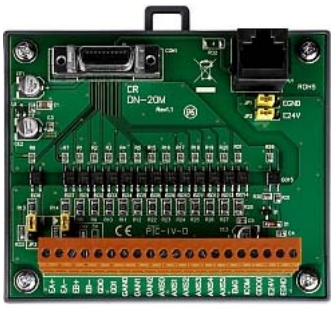
2.4.2 DN-20M Board

Termination board with one 20-pin SCSI-II connector for MPG and FRnet

- 20 pin connector:

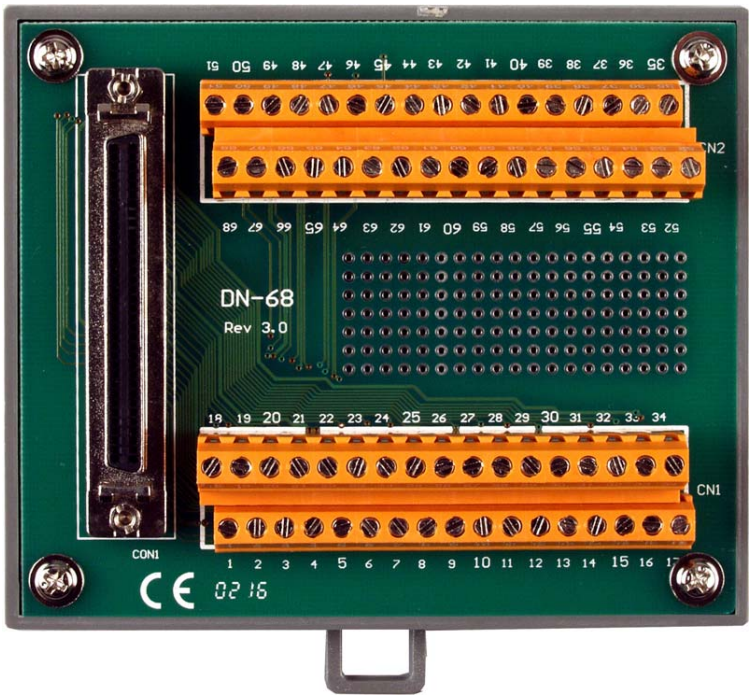
No.	Name	No.	Name
1	EA+	11	AXIS1 / GDI6
2	EA-	12	AXIS2 / GDI7
3	EB+	13	AXIS3 / GDI8
4	EB-	14	AXIS4 / GDI9
5	GDI0	15	AXIS5 / GDI10
6	GDI1	16	EMG
7	GAIN0 / GDI2	17	ICOM
8	GAIN1 / GDI3	18	GDO0
9	GAIN2 / GDI4	19	E24V
10	AXIS0 / GDI5	20	EGND





2.4.3 DN-68 Board

General purpose termination board with one 68-pin SCSI-II connector (DN-68)



2.4.4 Accessory

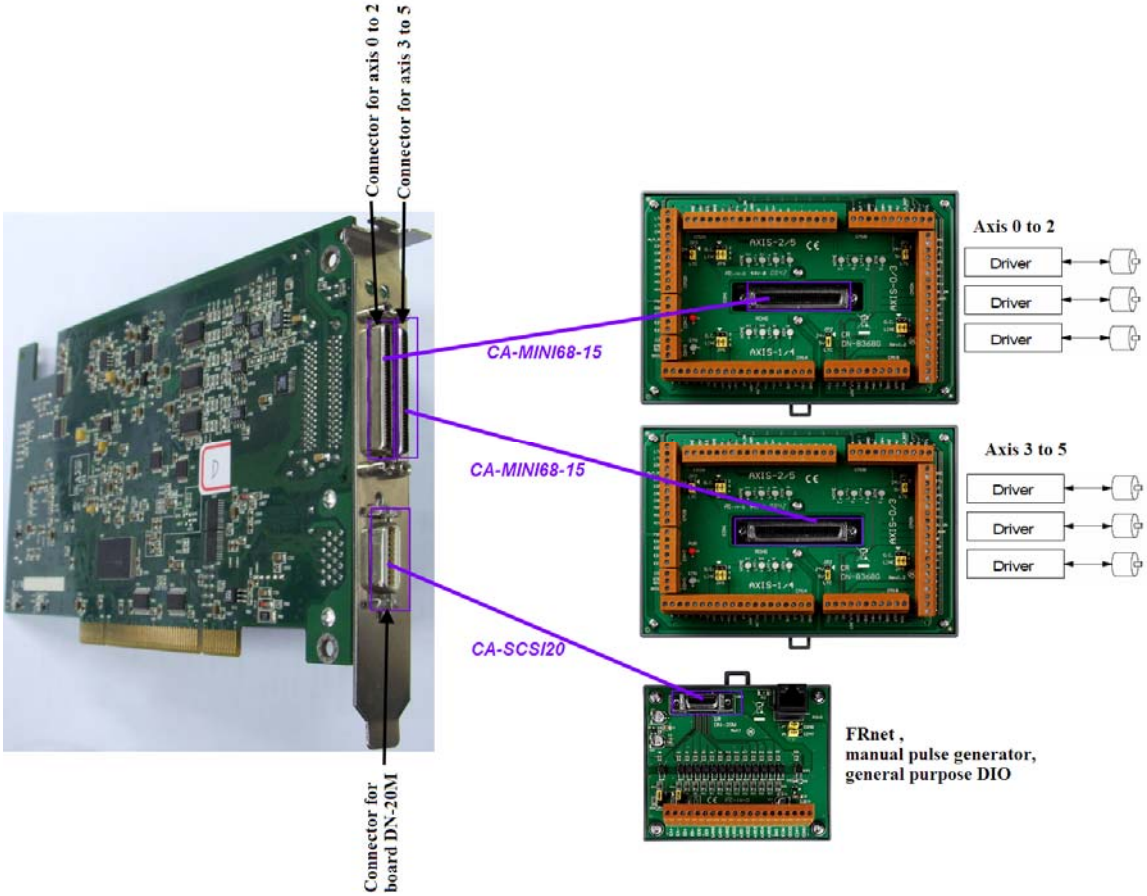
2.4.4.1 VHDCI 0.8mm 68M to SCSI-II MD68M Cable

2.4.4.2 SCSI-II 20M to SCSI-II 20M Cable

2.5 Hardware Installation

- **Prepare controller**
  1. Choose a personal PC with empty PCI slot.
  2. Turn power off

- **Motion card plug-in and wiring**
  1. Plug the PMDK card into an empty PCI slot of PC.
  2. Assign each PMDK card a unique ID by setting its DIP switch
  3. Connect the PMDK card to two three axis terminal boards (***DN-8368***) by using the ***CA-MINI68-15*** cable.
  4. Connect the daughter board (***DN-20M***) to the PMDK card by using the ***CA-SCSI20*** cable**.**
  5. Connect the axis terminal boards to the amplifier driver



3 Function Library

3.1 DSP Initialization Function

```
void PMDK_Init ();
```

Description Initialize the PMDK card. Call this function only once. Most of the basic functions provided by ICPDAS are functional only after initialization. This function will do following actions:

1. Reset and enable the cache memory on DSP.
2. Set the timing of DSP and its peripherals.
3. Reset FPGA.
4. Set the FPGA and FRnet as the sources of interrupts.
5. Set the DSP_OK bit as '1'. Host computer can read this bit through PCI bus. The register is loacated at address BAR0+0x04, and the bit 7 is the DSP_OK bit.

Parameters None

Return Value None

Example

```
PMDK_Init();
```

PMDK_Init();

See Also

3.2 Memory Access Functions

3.2.1 Set_Mem

```
void Set_Mem (U32 addr, U16 data);
```

Description Write a 16-bit data to the assigned address.

Parameters

addr
is an absolute address. Since each address can only save a byte (8-bit), the *addr* must be an even number.

data
is the 16-bit data to be written.

Return Value None

Example

```
// Write 100 to the starting address of DPRAM

Set_Mem(DPRAM_ADDR + 0, 100);
```

See Also

Get_Mem()

3.2.2 Get_Mem

```
U16 Get_Mem (U32 addr);
```

Description Read a 16-bit data from the assigned address.

Parameters

addr

is an absolute address. Since each address can only save a byte (8-bit), the *addr* must be an even number.

Return Value The16-bit data located at the assigned and its next address.

Example

```
U16 data;
data = Get_Mem(DPRAM_ADDR + 6);
//  data is composed by two bytes located at
(DPRAM+6) //and (DPRAM+7)
```

See Also

Set_Mem()

3.2.3 Flash_Erase

```
I16 Flash_Erase (U8 Region);
```

Description Erase the data located in the assigned region.

Parameters

Region

is the assigned region to be erased. The definitions of regions are in following table.

Macro	*Value*	*Description*
FLASH_ERASE_CHIP	0	Erase all FLASH data Range: 0x00000~0xFFFFF
FLASH_ERASE_CODE	1	Erase DSP code region Range: 0x00000~0x9FFFF
FLASH_ERASE_RSV	2	Erase Reserved region Range: 0xA0000~0xAFFFF
FLASH_ERASE_CAL	3	Erase AD/DA calibration data region Range: 0xB0000~0xBFFFF
FLASH_ERASE_FPGA	4	Erase FPGA Configuration data region Range: 0xC0000~0xFFFFF

Return Value

ERR_ADDR_OUT_RANGE
ERR_HW_ID_ERROR
ERR_TIMEOUT_ERROR
ERR_NO_ERROR

Example

```
// Erase FPGA Configuration Data
Flash_Erase(FLASH_ERASE_FPGA);
```

See Also

Flash_Program()

3.2.4 Flash_Program

I16 Flash_Program (U32 *Source_addr*, **U32** *Flash_ofst*, **U32** *Num_byte*);

Description Write data to the Flash.

Parameters

Source_addr
is the starting address of the data source. The addresses located in the Flash or FRAM are not allowed to be used.

Flash_ofst
is the starting offset address of Flash to be written

Num_byte
is the length of data in bytes

Return Value
ERR_ADDR_OUT_RANGE
ERR_TIMEOUT_ERROR
ERR_NO_ERROR

Example

```
// Erase FPGA Configuration Data region in Flash
Flash_Erase(FLASH_ERASE_FPGA);

// Write new FPGA Configuration Data to Flash
Flash_Program(SDRAM_ADDR+FPGA_DATA_SDRAM_OFST,
                  FPGA_DATA_FLASH_OFST,
                  FPGA_DATA_FLASH_SIZE);
```

See Also

Flash_Program()

3.2.5 Get_Flash

```
U16 Get_Flash (U32 Offset);
```

Description Read a 16-bit data from the offset address of the Flash.

Parameters

Offset

Offset is the assigned offset address. The range of this offset value is 0 ~ 0xFFFFF. The total size is 1MB, or 512K *16-bit. Since DSP only can save one byte (8-bit data) for each address value, the offset value must be even number for correct operation.

Return Value The read out 16-bit data.

Example

```
U16 data;
data = Get_Flash(6);
// data is the 16-bit value read at offset = 6
```

See Also

3.2.6 Set_FRAM

```
void Set_FRAM (U32 Offset, U16 data);
```

Description Write a 16-bit data to the assigned offset address on FRAM.

Parameters

Offset

Offset is the relative address to the starting address of the FRAM. The range of offset value is 0 ~ 0x3FFFF. The total size is 256KB, or 128K *16-bit. Since DSP only can save one byte (8-bit data) for each address value, the offset value must be even number for correct operaton.

data

Data to write to the FRAM
Note: only the low byte will be saved in FRAM.

Return Value None

Example

```
// write a value 100 to the FRAM at offset = 0.
Set_FRAM(0, 100);
```

See Also

Get_FRAM()

3.2.7 Get_FRAM

```
U16 Get_FRAM (U32 Offset);
```

Description Read a 16-bit data from the specified offset address on FRAM.

Parameters

Offset

Offset is the relative address to the starting address of FRAM. The range of offset value is 0 ~ 0x3FFFF. The total size is 256KB, or 128K *16-bit. Since DSP only can save one byte (8-bit data) for each address value, the offset value must be a even number for correct operation.

Return Value The read out 16-bit data.
Note: only the low byte is valid.

Example

```
U16 data;
data = Get_FRAM(4) & 0xff;
// data is low byte read at offset = 4
```

See Also

Set_FRAM()

3.3 FRnet Functions

The DN-20M daughter board provides one FRnet master port connection. The master can control up to 128 digital output and 128 digital inputs. ICPDAS provides two types of FRnet slaves: digital input device with 16 channels and digital output devices with 16 channels. Up to16 ICPDAS FRnet slaves can be connected to the master port: a maximum of 8 digital input FRnet slaves and maximum of 8 digital output FRnet slaves. For each master port the first 8 slave addresses (00 to 07) are reserved for digital output modules and the remaining 8 addresses (08 to 15) are reserved for digital input modules.
The data update time for FRnet bus running at 1Mbps communication speed is 0.72ms; and for 250 kbps bus speed is 2.88 ms. However, lower bus speed can provides longer communication distance between master and slave modules.

3.3.1 Read_FRnet

```
I16 Read_FRnet (U8 Ch, U16 *Data);
```

Description

Read the data of digital inputs of FRnet. Each group has 16 bit data (16 DI). There are 8 digital input groups in this system; therefore, totally 128 DI can be defined in this interface.

Parameters

Ch

CH is the group number. Available *Ch* values for input are 8~15. Number 0~7 are reserved for digital outputs (DO).

Data

Data is a 16-bit DI value.

Return Value

ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
I16 error;
U16 data;
error = Read_FRnet(8, &data);
// data is the 16-bit FRnet DI data of group 8
```

See Also

Write_FRnet()

3.3.2 Write_FRnet

```
I16 Write_FRnet (U8 Ch, U16 Data);
```

Description

Write 16-bit FRnet DO data to the specified group (*Ch*). Each group can has 16-bit data (16 DO). There are 8 digital output groups in this system; therefore, totally 128 DO can be defined in this interface.

Parameters

Ch is the group number. Available *Ch* values for output are 0~7. Number 8~15 are reserved for digital inputs (DI).

Data is the DO value to the assigned group *Ch*.

Return Value

ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set all DO in group 0 to 1
Write_FRnet (0, 0xFFFF);
```

See Also

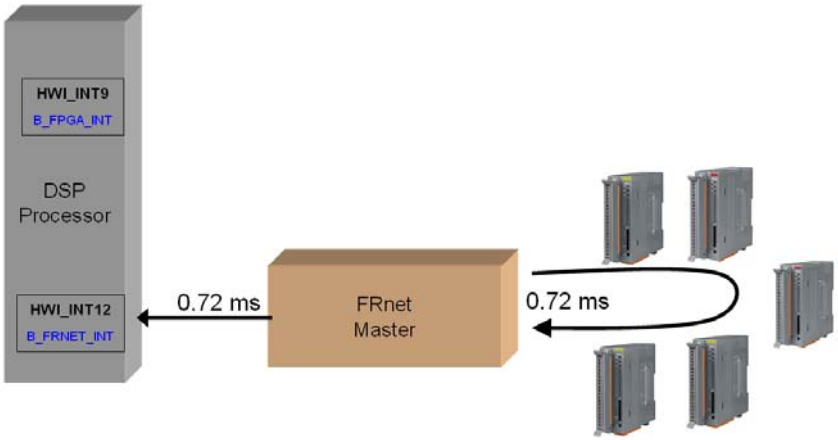
Read_FRnet()

3.3.3 Enable_FRnet_Scan

```
void Enable_FRnet_Scan ();
```

Description

Enable FRnet to update I/O data every 0.72 ms (for 1Mbps communication speed). The FRnet master chip starts reading DI and writing DO data to its slaves every 0.72 ms and send an interrupt signal to the DSP at hardware port 12 every 0.72ms. To enable the DSP interrupt for FRnet see function **`Set_DSP_Int_Factor`.**



Parameters

None

Return Value

None

Example

```
// set communication speed:1 Mbps
Set_FRnet_HighSpeed(1);
Enable_FRnet_Scan();
```

See Also

Disable_FRnet_Scan()

3.3.4 Disable_FRnet_Scan

```
void Disable_FRnet_Scan (void);
```

Description Disable the I/O data updating and sending interrupts to the DSP chip.

Parameters None

Return Value None

Example

```
Disable_FRnet_Scan();
```

See Also

Enable_FRnet_Scan()

3.3.5 Get_FRnetStatus

```
U8 Get_FRnetStatus (void);
```

Description Read the communication status of DI module on FRnet bus.

Parameters None

Return Value Each FRnet bus can control 8 input groups. Each group has 16-bit DI data. The range of DI module numbers are 8 ~15. Following table uses Group_8 ~ Group_15 to represent each DI group's status.

Bit	7	6	5	4	3	2	1	0
Function	Group_15	Group_14	Group_13	Group_12	Group_11	Group_10	Group_9	Group_8

If Group_n (n = 8 ~15) is '0', the corresponding group is not existing on the bus.
If Group_n (n = 8 ~15) is '1', the corresponding group is on the bus. It means the hardware module which contains this group number is existed and the communication is fine.
The host can use this status to judge whether the communication with the slave modules has been established. For example, a broken line can be detected if there is a DI module on FRnet bus while the corresponding status bit shows '0'.

Example

```
// Get the status data
U16 data;
data = Get_FRnetStatus();
```

See Also

Read_FRnet(), Write_FRnet()

3.3.6 Set_FRnet_HighSpeed

```
void Set_FRnet_HighSpeed(U8 Value);
```

Description Set the communication speed of FRnet bus. The default speed is 1 Mbps.

Parameters

Value 1: set the communication speed of FRnet bus to be 1 Mbps; or
0: for FRnet bus speed to be 250 Kbps.
The default speed is 1 Mbps.

The FRnet interrupt frequency is related to the speed setting. The interrupt will be generated according to communication speed
250 Kbps: every 2.88 ms an interrupt
1 Mbps: every 0.72 ms an interrupt

Return Value None

Example

```
// Set the FRnet communication speed to be 1 Mbps
Set_FRnet_HighSpeed(1);
```

See Also

Get_FRnet_HighSpeed ()

3.3.7 Get_FRnet_HighSpeed

```
U8 Get_FRnet_HighSpeed(void);
```

Description Get the communication speed setting of FRnet bus.

Parameters

Return Value 1: the FRnet bus speed is 1Mbps;
0: the FRnet bus speed is 250kbps.

Example

```
// Get the FRnet communication speed
U8 HighSpeedStatus;
HighSpeedStatus = Get_FRnet_HighSpeed();
```

See Also

Set_FRnet_HighSpeed ()

3.4 FPGA Initialization Functions

3.4.1 FPGA_Init

```
I16 FPGA_Init (U8 FPGA_Src, U16 Servo_Prd);
```

Description Initialize the FPGA. It will load the FPGA configuration data to FPGA for execution; set both servo periodic update time and DDA cycle time (or DDA length); and load the calibration data of peripherals.

Parameters

FPGA_Src

assigns the source of FPGA codes. The source can locate either on SDRAM or FLASH. Please refer to definitions in following table.

Macro	*Value*	*Description*
FPGA_SRC_SDRAM	0	PMDK will load the configuration data for FPGA from SDRAM. The data start from offset address 0x00F00000 (FPGA_DATA_SDRAM_OFST) on SDRAM. If user's project includes file "LoadFPGAData.C", the data in fpgaPMDK.H will be loaded to SRAM from this offset address first. Then load these data to FPGA.
FPGA_SRC_FLASH	1	PMDK will load the configuration data for FPGA from FLASH. DSP will copy FLASH data, starting from offset 0xC0000 to 0xFFFFF, onto the SDRAM (starting from offset 0x00F00000, FPGA_DATA_SDRAM_OFST) first. Then load these data to FPGA.

Servo_Prd

value must be in the range from 3 to 32767. The default value is 6250. The definition of servo update time is:
Servo_Update_Time = (*Servo_Prd* * 40) ns
Therefore, the default value will be
6250 * 40 ns = 250000ns = 250 us

Return Value

ERR_ADDR_OUT_RANGE
ERR_FPGA_DL_FAILED
ERR_VALUE_OUT_RANGE

ERR_NO_ERROR

Example

```
// Load the FPGA Configuration Data from FLASH
// Servo update time is 6250 * 40ns = 250us
I16   error;
error = FPGA_Init(FPGA_SRC_FLASH,
DEFAULT_DDA_LENGTH);
```

See Also

Set_Servo_Period(),
Set_DDA_Length(),
Write_Cal_Data()

3.4.2 Get_FPGA_Version

```
void Get_FPGA_Version (U16 *Value);
```

Description Get the version information of FPGA code.

Parameters

Value is the version information where low-byte is the FPGA Version, and high-byte is the PCB Version.

Bit	7	6	5	4	3	2	1	0
Function	FPGA Version							
Bit	15	14	13	12	11	10	9	8
Function	PCB Version							

Return Value None

Example

See Also

3.4.3 Set_Servo_Period

```
I16 Set_Servo_Period (U16 Value);
```

Description Set ther servo update period. User can set the servo update period for PMDK when **FPGA_Init()** is called. It is better not to change it afterward.

Parameters

Value must be in the range from 3 to 32767. The default value is 6250. The definition of servo update time is:
Servo_Update_Time = (*Servo_Prd* * 40) ns
Therefore, the default value will be
6250 * 40 ns = 250000ns = 250 us

Return Value ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// change the servo update period to 500 us
I16   error;
error = Set_Servo_Period(DEFAULT_DDA_LENGTH*2);
```

See Also

Get_Servo_Period(), FPGA_Init(), Set_DDA_Length()

3.4.4 Get_Servo_Period

```
void Get_Servo_Period (U16 *Value);
```

Description Get the servo update period setting.

Parameters

Value must be in the range of 3~ 32767. Please refer to the explanations of function **Set_Servo_Period()**.

Return Value None

Example

```
// Get the setting of servo period parameter
U16  period;
Get_Servo_Period(&period);
```

See Also

Get_Servo_Period()

3.5 General-Purpose DIO and MPG signals

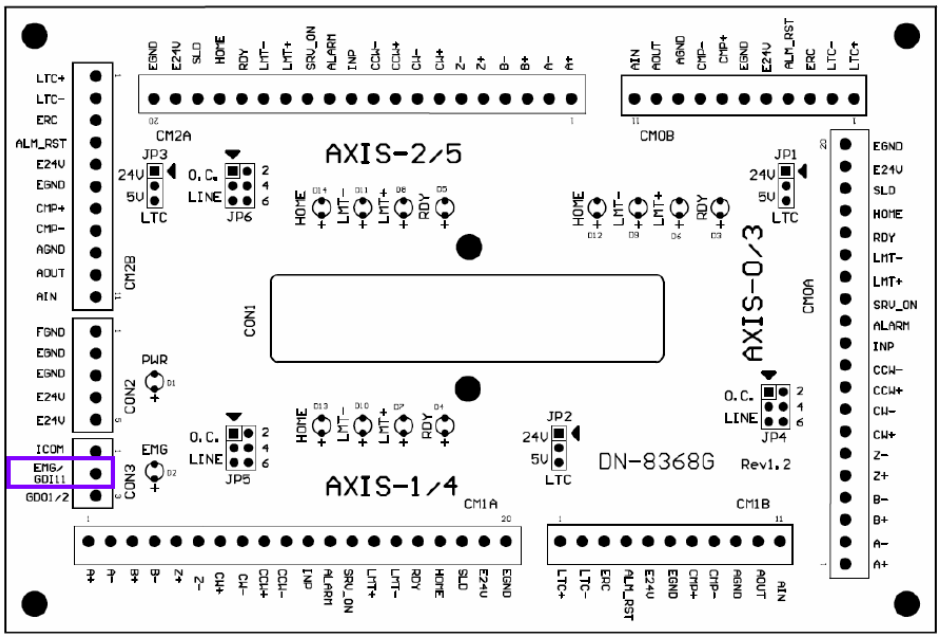


Figure 2: Emergency stop (EMG) and GDI11 channels are on the DN-8368 board

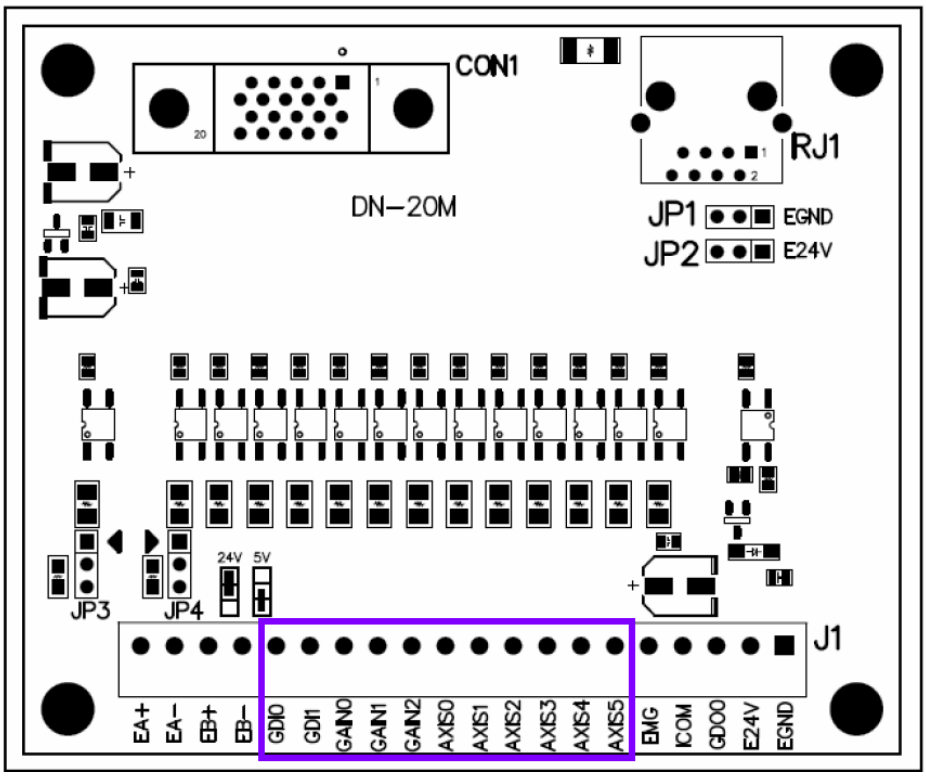


Figure 3: GDI0 to GDI10 pins are on the DN-20M board

3.5.1 Get_DI

```
void Get_DI(U16 *Value);
```

Description Get the values of general-purpose DI.

Parameters

Value is the value of DI. In the following table, GDI0 ~ GDI11 are general-purpose DI. But this function also provide the values of EMG (emergency stop signal) and MPG_EMG (the EMG signal on MPG).

Bit	7	6	5	4	3	2	1	0
Signal	GDI7	GDI6	GDI5	GDI4	GDI3	GDI2	GDI1	GDI0
Bit	15	14	13	12	11	10	9	8
Signal	EMG	MPG_EMG	-	-	GDI11	GDI10	GDI9	GDI8

Many hand-held MPG got more than pulse input interface; they also have gain switches and axis-selected switches. The daughter board of this card gives the pin definitions for wiring and programming in the following table.

Bit	7	6	5	4	3	2	1	0
Signal	**AXIS2**	**AXIS1**	**AXIS0**	**GAIN2**	**GAIN1**	**GAIN0**	**GDI1**	**GDI0**
Bit	15	14	13	12	11	10	9	8
Signal	**EMG**	**MPG_EMG**	-	-	EMG/GDI11	**AXIS5**	**AXIS4**	**AXIS3**

Note: These definitions are only for hardware labeling. Users still have to implement their own MPG functions by programming.

Users can refer to the following constant definitions in their programs.

Macro	***Value***	***Description***	***DN-20M daughter board pin***
B_GDI0_SIG	0x0001	DI channel 0	GDI0
B_GDI1_SIG	0x0002	DI channel 1	GDI1
B_GDI2_SIG	0x0004	DI channel 2	GAIN0
B_GDI3_SIG	0x0008	DI channel 3	GAIN1
B_GDI4_SIG	0x0010	DI channel 4	GAIN2
B_GDI5_SIG	0x0020	DI channel 5	AXIS0
B_GDI6_SIG	0x0040	DI channel 6	AXIS1
B_GDI7_SIG	0x0080	DI channel 7	AXIS2
B_GDI8_SIG	0x0100	DI channel 8	AXIS3

B_GDI9_SIG	0x0200	DI channel 9	AXIS4
B_GDI10_SIG	0x0400	DI channel 10	AXIS5
B_GDI11_SIG	0x0800	DI channel 11	EMG/GDI11 (on the DN-8368 board for axis 3 to 5)
B_MPG_EMG_SIG	0x4000	Emergency Stop from MPG (MPG_EMG)	EMG
B_EMG_SIG	0x8000	Emergency Stop (EMG)	EMG/GDI11 (on the DN-8368 board for axis 0 to 2)

Return Value None

Example

```
U16 di;
Get_DI(&di);

If( (di & B_EMG_SIG) == B_EMG_SIG)
{
   // if EMG is '1' then …
}
```

See Also

Set_DI_Pol(), Get_DI_Pol()

3.5.2 Set_DI_Pol

```
void Set_DI_Pol(U16 Value);
```

Description

Set the polarity of DI to be active-low or active-high. These polartity definitions are referred by functions **Set_DA_Clear_Ctl()** and **Set_DDA_Clear_Ctl()**. If these DI's interrupt functions are enabled, the polarity settings will also affect their interrupt behaviors by reversing the rising-edge and falling-edge triggers.

Parameters

Value In the following table, GDI0_POL ~ GDI10_POL are used for general-purpose DI. But this function also provide the polarity setting values of EMG (emergency stop signal) and MPG_EMG (the EMG signal on MPG). The default polarity value for each signal is '1'.

Bit	7	6	5	4	3	2	1	0
Signal	GDI7_POL	GDI6_POL	GDI5_POL	GDI4_POL	GDI3_POL	GDI2_POL	GDI1_POL	GDI0_POL
Bit	15	14	13	12	11	10	9	8
Signal	EMG_POL	MPG_EMG_POL	-	-	-	GDI10_POL	GDI9_POL	GDI8_POL

EMG_POL :
'1': EMG is active High and rising edge trigger
'0': EMG is active Low and falling edge trigger
MPG_EMG_POL :
'1': MPG_EMG is active High and rising edge trigger
'0': MPG_EMG is active Low and falling edge trigger
GDIx_POL :
'1': GDIx is rising edge trigger
'0': GDIx is falling edge trigger

Return Value None

Example

```
// Set all DI polarities to be active_high
Set_DI_Pol(0xCFFF);
```

See Also

Get_DI_Pol(), Set_DA_Clear_Ctl(), Set_DDA_Clear_Ctl(), Set_FPGA_Int_Factor()

3.5.3 Get_DI_Pol

```
void Get_DI_Pol(U16 *Value);
```

Description Get the polarity settings for DI. It will affect functions **Set_DA_Clear_Ctl()** and **Set_DDA_Clear_Ctl()**. It will also affect the interrupt behaviors.

Parameters

Value contains the polarity settings of all DI.
Please refer to function **Set_DI_Pol()** for more information.

Return Value None

Example

```
// Read all DI polarities
U16 Value;
Get_DI_POL(&Value);
```

See Also

Set_DI_POL()

3.5.4 Set_DO

```
void Set_DO (U16 Value);
```

Description

Set the output values of the general purpose DO signals. The PMDK card supports three digital outputs:

- GDO0 on the DN-20M board
- GDO1 on the DN-8368 board for axis 0 to 2
- GDO2 on the DN-8368 board for axis 3 to 5

Parameters

Value

contains the definition of each DO signal. The bit definition can be 1 (OFF) or 0 (ON). Three DOs are defined in the following table.

Bit	7	6	5	4	3	2	1	0
Signal	-	-	-	-	-	**GDO2**	**GDO1**	**GDO0**
Bit	15	14	13	12	11	10	9	8
Signal	-	-	-	-	-	-	-	-

GDO2 : Write '0' to turn on the output transistor of GDO2.
GDO1 : Write '0' to turn on the output transistor of GDO1.
GDO0 : Write '0' to turn on the output transistor of GDO0.

Constant definitions in the following table can be used in user's programs.

Macro	***Value***	***Description***
B_GDO0_SIG	0x0001	Refer to DO channel 0 (DN-20M board)
B_GDO1_SIG	0x0002	Refer to DO channel 1 (DN-8368 board for axis 0~2)
B_GDO2_SIG	0x0004	Refer to DO channel 2 (DN-8368 board for axis 3~5)

Return Value

None

Example

```
Set_DO( (B_GDO0_SIG | B_GDO1_SIG) ^ 0xFFFF);
// Set GDO0 and GDO1 to 0
// It will cause the GDO0 and GDO1 to turn on.
```

See Also

Get_DO()

3.5.5 Get_DO

```
void Get_DO (U16 *Value);
```

Description Get the outputs of DOs.

Parameters

Value contains the value of DO.
Please refer to function **Set_DO()** for more information.

Return Value None

Example

```
U16 do;
Get_DO(&do);
```

See Also

Set_DO()

3.5.6 Set_DI_Filter

```
I16 Set_DI_Filter (U8 Ch, U8 Value);
```

Description Set the parameter for the low-pass filter of the assigned general-purpose DI.

Parameters

Ch is defined as in the following table. The definitions of following constants can be used in user's programs.

Macro Name	*Ch*	*Description*
GDI0_FLT_CH	0	Refer to general-purpose DI0
GDI1_FLT_CH	1	Refer to general-purpose DI1
GDI2_FLT_CH	2	Refer to general-purpose DI2
GDI3_FLT_CH	3	Refer to general-purpose DI3
GDI4_FLT_CH	4	Refer to general-purpose DI4
GDI5_FLT_CH	5	Refer to general-purpose DI5
GDI6_FLT_CH	6	Refer to general-purpose DI6
GDI7_FLT_CH	7	Refer to general-purpose DI7
GDI8_FLT_CH	8	Refer to general-purpose DI8
GDI9_FLT_CH	9	Refer to general-purpose DI9
GDI10_FLT_CH	10	Refer to general-purpose DI10
GDI11_FLT_CH	11	Refer to general-purpose DI11
MPG_B_FLT_CH	12	Refer to the MPG's pulse input phase B (MPG_B)
MPG_A_FLT_CH	13	Refer to MPG's pulse input phase A (MPG_A)
MPG_EMG_FLT_CH	14	Refer to MPG's EMG (MPG_EMG)
EMG_FLT_CH	15	Refer to card's EMG

Value for each channel is default to be 0 (disable the low-pass filter function). There are several different low-pass filters for different DI channels. Their definitions are as follow:

For MPG_EMG, MPG_A, MPG_B and EMG

Value	Min. Steady Input Width (us)	Input Signal Delay Time (us)
0	disable	disable
1	5.12	3.84
2	40.96	30.72
3	327.68	245.76

For DI0, DI1 ~ DI11

Value	Min. Steady Input Width (us)	Input Signal Delay Time (us)
0	disable	disable
1	5.12	3.84

Return Value

ERR_ADDR_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the parameter of digital filter of DI7 to
be // 1 (or ON).
I16   error;
error = Set_DI_Filter(7, 1);
```

See Also

Get_DI_Filter(), Set_Axis_IO_Filter()

3.5.7 Get_DI_Filter

```
I16 Get_DI_Filter (U8 Ch, U8 *Value);
```

Description

Get the general-purpose DI filter setting.

Parameters

Ch and *Value* are defined according to the parameter definitions of function **Set_DI_Filter()**.

Return Value

ERR_ADDR_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the value setting of digital filter of DI7.
U8 value;
Get_DIO_Filter(7, &value);
```

See Also

Set_DI_Filter()

3.6 Axis-specific DIO functions

3.6.1 Set_Axis_IO

```
I16 Set_Axis_IO (U8 Axis, U16 Value);
```

Description Set the axis-specific **digital output (DO) value**.

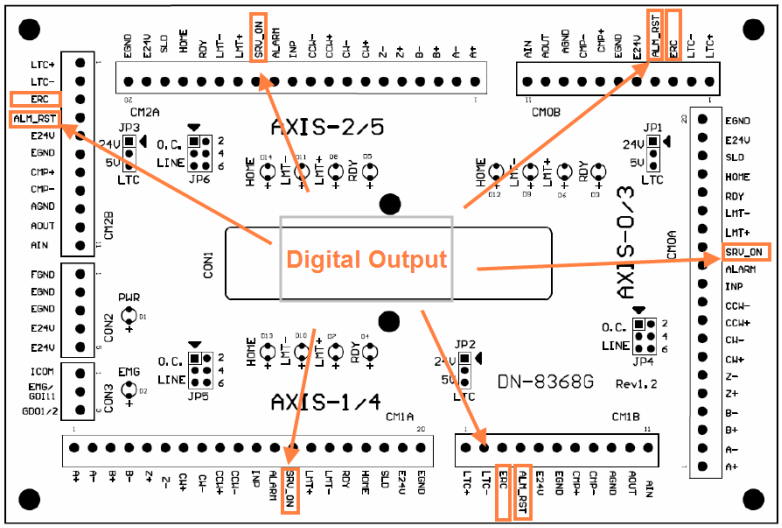


Figure 4: The function refers to the DO of the two DN-8368 boards

Parameters

Axis is the assigned axis; range is 0 ~ 5.

Value contains the DOs defined in the following table. The default bit value for each DO is '1' (OFF).

Bit	7	6	5	4	3	2	1	0
Function	-	-	-	-	-	-	-	-

Bit	15	14	13	12	11	10	9	8
Function	-	ALMRST	ERC	SVON	-	-	-	-

ALMRST : Write '0' to turn on the output transistor of ***alarm reset*** signal (ALMRST).

ERC : Write '0' to turn on the output transistor of error ***counter clear signal (ERC)***.

SVON : Write '0' to turn on the output transistor of ***servo on*** signal (SVON).

Constant definitions in the following table can be used in user's programs.

Macro	*Value*	*Description*	*Pin definition of DN-8368 board*
B_SVON_SIG	0x1000	Refer to servo on signal	SRV_ON
B_ERC_SIG	0x2000	Refer to error counter clear signal	ERC
B_ALMRST_SIG	0x4000	Refer to alarm reset signal	ALARM

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Let SVON = 0 (active)
I16   error, state;
Get_Axis_IO(0, &state);
error = Set_Axis_IO(0, (~B_SVON_SIG) & state);
```

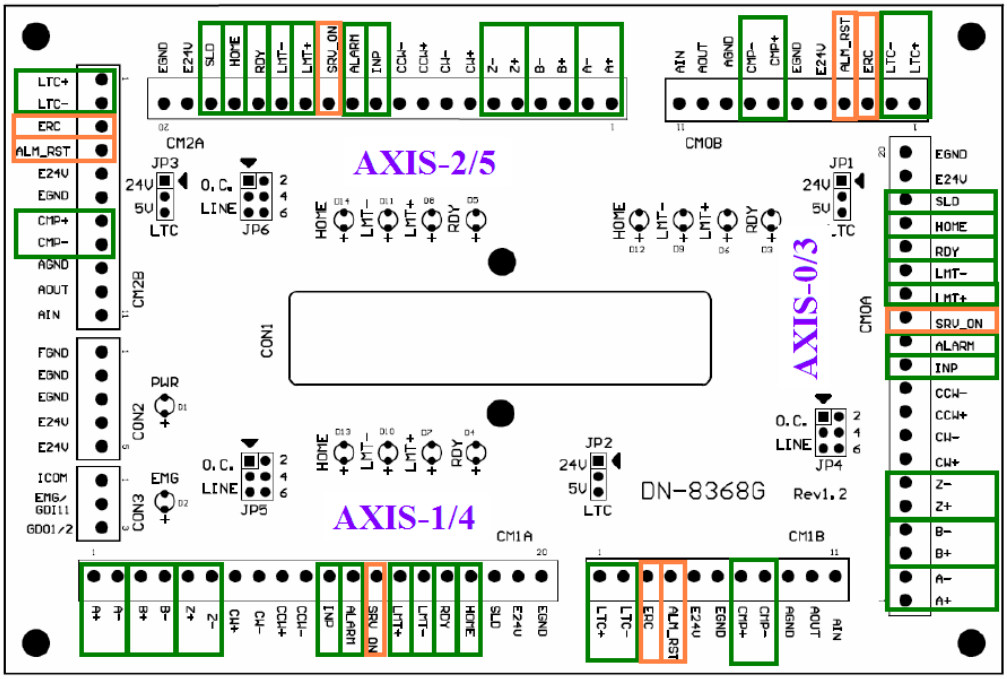
See Also

Get_Axis_IO(), Set_Axis_IO_POL(), Get_Axis_IO_POL()

3.6.2 Get_Axis_IO

```
I16 Get_Axis_IO (U8 Axis, U16 *Value);
```

Description Get the axis-specific **DI/DO state**. Reads digital input and outputs.



----Digital Input
----Digital Output

Parameters

Axis is the assigned axis; range is 0 ~ 5.

Value contains the states of axis-specific DIs/DOs defined in the following table.

Bit	7	6	5	4	3	2	1	0
Function	LTC	RDY	INP	ALM	SLD	ORG	MEL	PEL
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Function	CMP	ALMRST	ERC	SVON	EA	EB	EZ	IDX

Constant definitions in the following table can be used in user's programs.

Macro	*Value*	*Pin definition of DN-8368 board*	*Description*
B_PEL_SIG	0x0001	LMT+	Refer to Positive End Limit signal
B_MEL_SIG	0x0002	LMT-	Refer to Minus End Limit signal
B_ORG_SIG	0x0004	HOME	Refer to Original (Home) signal
B_SLD_SIG	0x0008	SLD	Refer to Slow Down signal
B_ALM_SIG	0x0010	ALARM	Refer to Alarm signal
B_INP_SIG	0x0020	INP	Refer to In Position signal
B_RDY_SIG	0x0040	RDY	Refer to Ready signal
B_LTC_SIG	0x0080	LTC+/LTC-	Refer to Latch signal
B_IDX_SIG*	0x0100		Refer to Index signal *
B_EZ_SIG	0x0200	Z+/Z-	Refer to Encoder Z signal
B_EB_SIG	0x0400	B+/B-	Refer to Encoder B signal
B_EA_SIG	0x0800	A+/A-	Refer to Encoder A signal
B_SVON_SIG	0x1000	SRV_ON	Refer to servo on signal
B_ERC_SIG	0x2000	ERC	Refer to error counter clear signal
B_ALMRST_SIG	0x4000	ALM_RST	Refer to alarm reset signal
B_CMP_SIG	0x8000	CMP+/CMP-	Refer to compare signal

***Note**: Please refer to **Set_Index_Mode()** for the definition of Index.

SVON, ERC, ALMRST and CMP are all DO signals; others are DI signals.
ALM, INP, and RDY are signals coming from drives.
PEL, MEL, ORG, and SLD are sensor signals. These four sensors are commonly installed on machines and are used for safety design and homing purposes.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U16 IOstatus;
I16   error;
error = Get_Axis_IO(0, &IOstatus);
```

See Also

Set_Axis_IO(), Set_Axis_IO_Pol(), Get_Axis_IO_Pol()

3.6.3 Set_Axis_IO_Pol

```
I16 Set_Axis_IO_Pol (U8 Axis, U16 Value);
```

Description

Set the polarities of axis-specific I/O to be **active-low** or **active-high**. These polartity definitions are referred by functions **Set_DA_Clear_Ctl()** and **Set_DDA_Clear_Ctl()**. If these DI's interrupt functions are enabled, the polarity settings will also affect their interrupt behaviors by reversing the rising-edge and falling-edge triggers.

Parameters

Axis is the assigned axis; range is 0 ~ 5.

Value contains the polarity definition for each DI. The default definition for each bit is '1'.

Bit	7	6	5	4	3	2	1	0
Function	LTC_POL	-	-	-	SLD_POL	ORG_POL	MEL_POL	PEL_POL
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Function	CMP_POL	-	-	-	-	-	EZ_POL	IDX_POL

CMP_POL : '1': CMP is rising edge trigger
'0': CMP is falling edge trigger
EZ_POL : '1': EZ is rising edge trigger
'0': EZ is falling edge trigger
IDX_POL : '1': Index is rising edge trigger
'0': Index is falling edge trigger
LTC_POL : '1': LTC is rising edge trigger
'0': LTC is falling edge trigger
SLD_POL : '1': LTC is rising edge trigger
'0': LTC is falling edge trigger
ORG_POL : '1': LTC is rising edge trigger
'0': LTC is falling edge trigger
MEL_POL : '1': MEL is active High and rising edge trigger
'0': MEL is active Low and falling edge trigger
PEL_POL : '1': PEL is active High and rising edge trigger
'0': PEL is active Low and falling edge trigger

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the DI polarity of AXIS0.
// SLD, ORG, MEL, and PEL are active low.

I16   error;
error = Set_Axis_IO_Pol(0, 0x8380);
```

See Also

Set_Axis_IO(), Get_Axis_IO(), Get_Axis_IO_Pol()

3.6.4 Get_Axis_IO_Pol

```
I16 Get_Axis_IO_Pol (U8 Axis, U16 *Value);
```

Description Get the polarity definitions of axis-specific DIs. These polartity definitions will be referred by functions **Set_DA_Clear_Ctl()** and **Set_DDA_Clear_Ctl()**. If these DI's interrupt functions are enabled, the polarity settings will also affect their interrupt behaviors by reversing the rising-edge and falling-edge triggers.

Parameters

Axis is the assigned axis; range is 0 ~ 5.

Value contains the polarity definitions of axis-specific DI. Please refer to function **Set_Axis_IO_Pol()** for more information.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Read the I/O polarity setting of AXIS0

U16 IO_polarity;
I16   error;
error = Get_Axis_IO_Pol(0, &IO_polarity);
```

See Also

Set_Axis_IO(), Get_Axis_IO(), Set_Axis_IO_Pol()

3.6.5 Set_Axis_IO_Filter

```
I16 Set_Axis_IO_Filter (U8 Axis, U8 Ch, U8 Value);
```

Description Configure the parameter of digital filter for the assigned axis-specific DI.

Parameters

Axis is the assigned axis; range is 0 ~ 5.

Ch is defined according to following table.

Ch	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Function	LTC	RDY	INP	ALM	SLD	ORG	MEL	PEL
Ch	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Function	-	-	-	-	-	EA	EB	EZ

Following pre-defined constants can be used in user's programs.

Macro Name	***Ch***	***Pin definition***	***Description***
PEL_FLT_CH	0	LMT+	Refer to the positive end limit DI (PEL or OT+) signal
MEL_FLT_CH	1	LMT-	Refer to the negative end limit DI (MEL or OT-) signal
ORG_FLT_CH	2	HOME	Refer to the origin DI (ORG or Home) signal
SLD_FLT_CH	3	SLD	Refer to the slow down DI (SLD) signal
ALM_FLT_CH	4	ALARM	Refer to the alarm DI (ALM) signal
INP_FLT_CH	5	INP	Refer to the in-position DI (INP) signal
RDY_FLT_CH	6	RDY	Refer to the ready DI (RDY) signal
LTC_FLT_CH	7	LTC+/LTC-	Refer to the latch DI (LTC) signal
EZ_FLT_CH	8	Z+/Z-	Refer to the encoder Z-phase (EZ) signal
EB_FLT_CH	9	B+/B-	Refer to the encoder B-phase (EB) signal
EA_FLT_CH	10	A+/A-	Refer to the encoder A-phase (EA) signal

Value Default setting for each channel is 0 (disable the low-pass filter function). There are several different low-pass filters for different DI channels. Their definitions are as follow:

For EA, EB, EZ, and LTC

Value	**Min. Steady Input Width (us)**	**Input Signal Delay Time (us)**

0	disable	disable
1	0.16	0.12
2	0.64	0.48
3	2.56	1.92

For PEL, MEL, ORG, SLD and INP

Value	Min. Steady Input Width (us)	Input Signal Delay Time (us)
0	disable	disable
1	5.12	3.84
2	40.96	30.72
3	327.68	245.76

For ALM and RDY

Value	Min. Steady Input Width (us)	Input Signal Delay Time (us)
0	disable	disable
1	5.12	3.84

Return Value

ERR_AXIS_OUT_RANGE
ERR_ADDR_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the parameter of digital filter of AXIS0's
LTC to 1.
I16   error;
error = Set_Axis_IO_Filter(0, 7, 1);
```

See Also

Get_Axis_IO_Filter()

3.6.6 Get_Axis_IO_Filter

```
I16 Get_Axis_IO_Filter (U8 Axis, U8 Ch, U8 *Value);
```

Description Get the axis-specific DI filter setting.

Parameters

Axis is the assigned axis; range is 0 ~ 5.

Ch and *Value*
are defined according to the parameter definitions of function **Set_Axis_IO_Filter()**.

Return Value

ERR_AXIS_OUT_RANGE
ERR_ADDR_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the parameter of digital filter of AXIS0's
LTC.
I16   error;
U8 parameter;
error = Get_Axis_IO_Filter(0, 7, &parameter);
```

See Also

Set_Axis_IO_Filter()

3.7 DA Functions

The following functions only refer to the 6 analog output ports (A**OUT**) of the six axis. The board DN-8368G in Figure 2has got 3 analog output ports. Two boards are needed for six axis motion control.

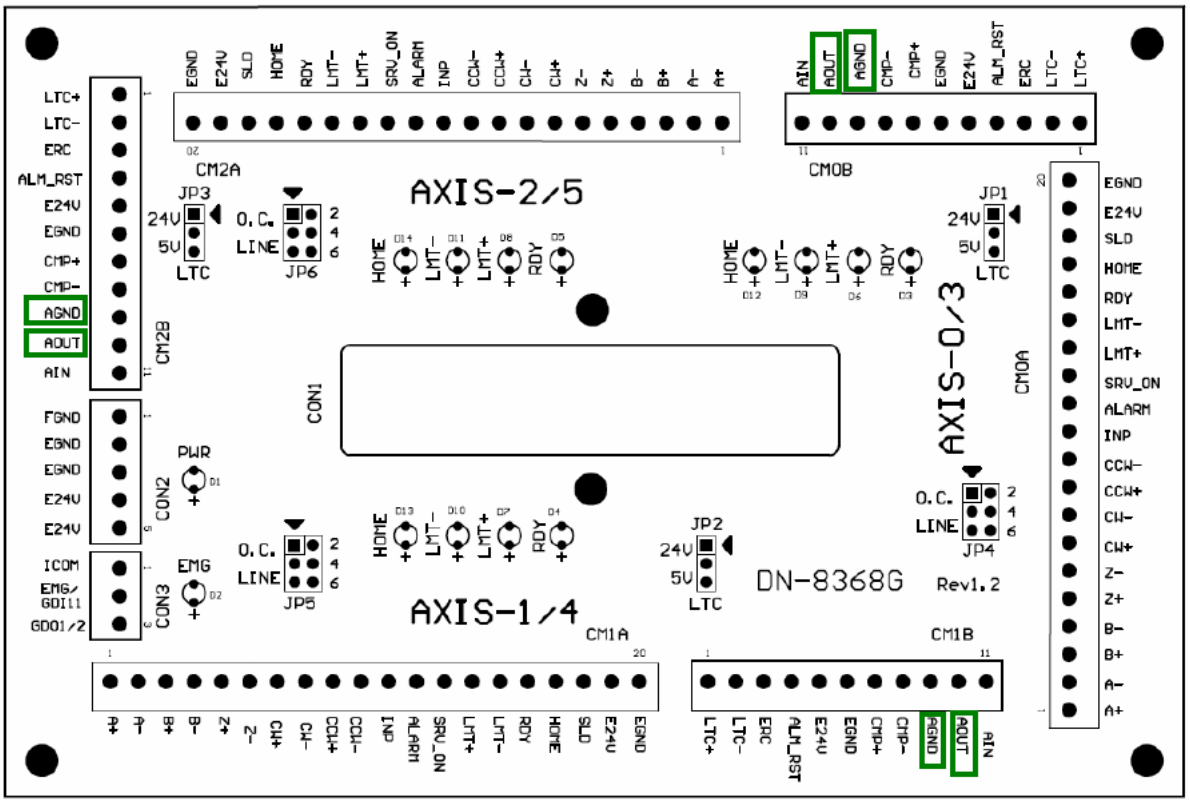


Figure 5: DN-8368G daughter board

3.7.1 DA_Update

```
I16 DA_Update (void);
```

Description All the DA channels update their values.

Parameters None

Return Value
ERR_DA_AUTO_UPDATE
ERR_DA_BUSY
ERR_NO_ERROR

Example

```
I16   error;
U8  Busy;
F32 fTest[]={2.5, 5, -2.5, 0, 0, 0};

for(i=0; i<6; i++)
{
  Set_DA_Value(i, fTest[i]);
  //assign a value for each DA channel
}
do { Get_DA_Busy(&Busy); } while (Busy);
error = DA_Update();
```

See Also

Get_DA_Busy(), DA_Auto_Update()

3.7.2 Get_DA_Busy

```
void Get_DA_Busy (U8 *Value);
```

Description Check whether the DA is busy or not. DA_Busy must be 0 before calling the **DA_Update()**.

Parameters

Value Value is 1 (DA is busy) or 0 (DA is ready for updating).

Return Value None

Example

```
I16   error;
U8  Busy;
F32 fTest[]={2.5, 5, -2.5, 0, 0, 0};

for(i=0; i<6; i++)
{
  Set_DA_Value(i, fTest[i]);
  //assign a value for each DA channel
}
do { Get_DA_Busy(&Busy); } while (Busy);
error = DA_Update();
```

See Also

DA_Update(), Set_DA_Value()

3.7.3 DA_Auto_Update

```
void DA_Auto_Update (U8 Enable);
```

Description Set the DA automatic-update function is enabled or disabled.

Parameters

Enable

- 1 - for automatic-updating values for DA channels. All the DA channels will be updated at the beginning of every servo update time.
- 0 - the DA will be updated only after calling the function **DA_Update()**.

Return Value None

Example

```
DA_Auto_Update(1);
```

See Also

DA_Update()

3.7.4 Clear_DA

```
I16 Clear_DA (U8 Ch);
```

Description Clear the DA output to 0V.

Parameters

Ch specifies the channel to be cleared. It is in the range of 0~5.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
I16   error;
error = Clear_DA(0);
```

See Also

Set_DA_Clear_Ctl()

3.7.5 Set_DA_Clear_Ctl

```
I16 Set_DA_Clear_Ctl (U8 Ch, U8 DA_Clr_Src, U8 Enable);
```

Description The DA of the assigned axis will be cleared when the EMG is tripped or that axis's PEL or MEL is tripped.

Parameters

Ch specifies the channel to be cleared. It is in the range of 0~5.

DA_Clr_Src

specifies the trigger source.

Macro	*Value*	*Description*
CLR_SRC_EMG	0	Refer to the Emergency Stop DI (EMG) signal
CLR_SRC_PEL	1	Refer to the positive end limit DI (PEL) signal
CLR_SRC_MEL	2	Refer to the negative end limit DI (MEL) signal

Enable

- 1 - for enabling the clear function;
- 0 – for disabling the clear function.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
//DA0 will be cleared if EMG is tripped.

I16   error;
error = Set_DA_Clear_Ctl(0, CLR_SRC_EMG, 1);
```

See Also

Get_DA_Clear_Ctl(), Set_DI_Pol(), Set_Axis_IO_Pol()

3.7.6 Get_DA_Clear_Ctl

```
I16 Get_DA_Clear_Ctl (U8 Ch, U8 DA_Clr_Src, U8
*Enable);
```

Description Check whether the DA of a certain axis will be cleared if the EMG or the axis's PEL and MEL are tripped.

Parameters

Ch specifies the channel to be cleared. It is in the range of 0~5.

DA_Clr_Src
specifies the trigger source.

Enable contains the information whether clearing the DA is enabled or not. For more informations, please refer to the descriptions of function **Set_DA_Clear_Ctl()**.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
// E_axis contains the setting information.

U8    E_axis;
I16   error;
error = Get_DA_Clear_Ctl (0, CLR_SRC_EMG,
&E_axis);
```

See Also

Set_DA_Clear_Ctl(), Set_DI_Pol(), Set_Axis_IO_Pol()

3.7.7 Set_DA_Data

```
I16 Set_DA_Data (U8 Ch, U16 Value);
```

Description Set the 16-bit value for a DAC channel. For directly setting floating point values use the function **Set_DA_Value()** instead.

Parameters

Ch specifies the DA channel number. It is in the range of 0~5.

Value is a 16-bit integer that maps to a **voltage range -10V ~ +10V**. The relationship between these two values can be defined in the following table.

16-bit Integer	0x0000	0x2000	0x4000	0x6000	0x8000	0xa000	0xc000	0xe000	0xffff
Voltage (V)	10	7.5	5	2.5	0	-2.5	-2.5	-7.5	-10

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
//set DA0 to output 5V

I16   error;
error = Set_DA_Data(0, 0x4000);
```

See Also

Get_DA_Data(), Set_DA_Value(), Get_DA_Value()

3.7.8 Get_DA_Data

```
I16 Get_DA_Data (U8 Ch, U16 *Value);
```

Description Get the setting value of a DA channel in a format of 16-bit integer. It is recommended to use **Get_DA_Value()** to read a floating value rather than an integer value from this function.

Parameters

Ch specifies the DA channel number. It is in the range of 0~5.

Value is a 16-bit integer that maps to a voltage range -10V ~ +10V. Please refer to **Set_DA_Data()** for more information.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
//Get DA0 value in a 16-bit integer format

U16   DA_value;
I16   error;
error = Get_DA_Data(0, &DA_value);
```

See Also

Set_DA_Data(), Set_DA_Value(), Get_DA_Value()

3.7.9 Set_DA_Value

```
I16 Set_DA_Value (U8 Ch, F32 Value);
```

Description Set the DA output by assigning a 32-bit floating point value. The range is -10.0 ~ +10.0 that maps to -10.0V ~ +10.0V.

Parameters

Ch specifies the DA channel number. It is in the range of 0~5.

Value is a 32-bit floating point value which must be in the range of -10.0 ~ +10.0. If the assigned value is out of this range, the output will still be clamped between -10.0V or +10.0V.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
// set DA0 output be 5V

I16   error;
error = Set_DA_Value(0, 5.0);
```

See Also

Get_DA_Value(), Get_DA_Data(), Set_DA_Data()

3.7.10 Get_DA_Value

```
I16 Get_DA_Value (U8 Ch, F32 *Value);
```

Description Get the DA output setting with a 32-bit floating point format.

Parameters

Ch specifies the DA channel number. It is in the range of 0~5.

Value is a 32-bit floating point value which must be in the range of -10.0 ~ +10.0. Its unit is V (voltage).

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
//Get DA0 output value as a 32-bit floating point
format

F32   DA_value;
I16   error;
error = Get_DA_Value(0, &DA_value);
```

See Also

Set_DA_Data(), Set_DA_Value(), Get_DA_Value()

3.8 AD Functions

Since there is only one AD chip shared by all AD channels on the PMDK, all AD channels are multiplexed.
The following functions only refer to the 6 analog input ports (AIN) of the six axes.
The board DN-8368G in Figure 6 has got 3 analog input ports. Two boards are needed for the six axis.

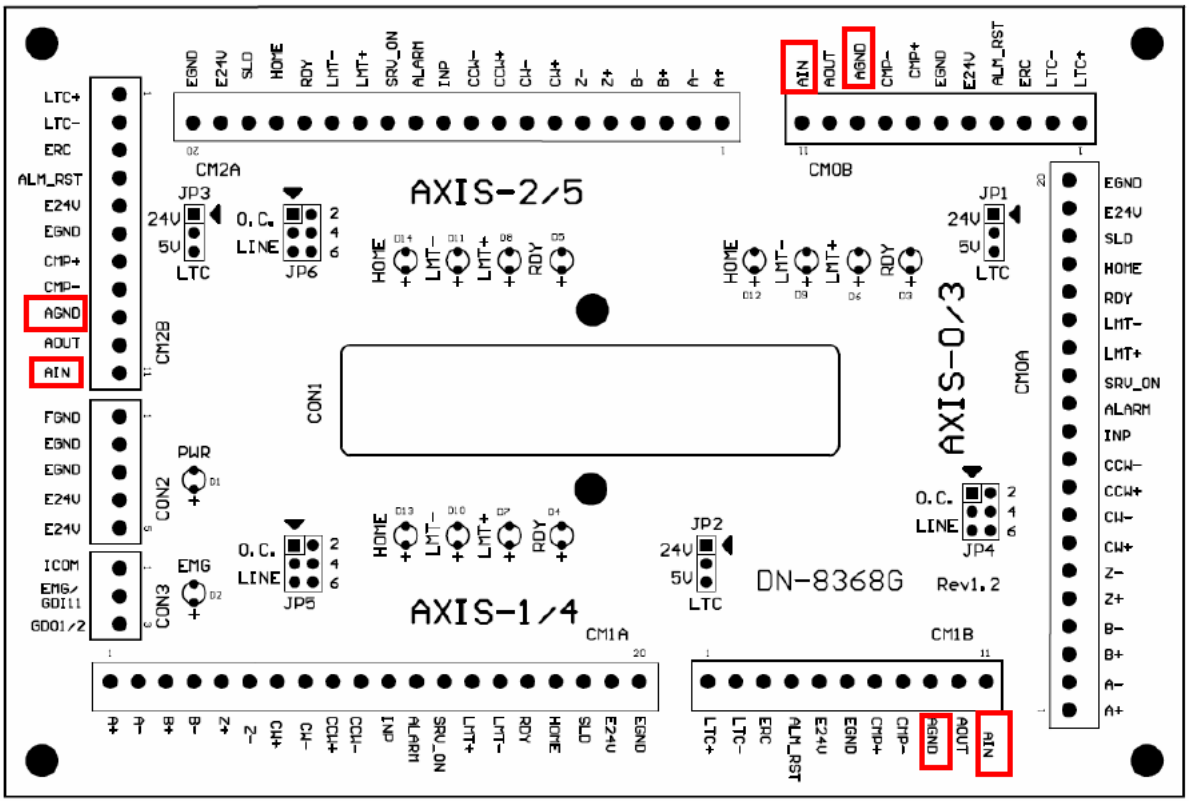


Figure 6 Analog input of the DN-8368G daughterboard

3.8.1 Set_AD_Start

```
void Set_AD_Start (U8 Enable);
```

Description Enable/Disable the conversion function of AD. When this function is enabled, the hardware will convert the AD values for the specified channels at the speed of **250 kHz**. The specified channels are defined as from Channel 0 to the channel assigned by function **Set_AD_Last()**.

Parameters

Enable

- 1 → AD will start the conversion.
- 0 →AD will stop the conversion. (Default value)

Return Value None

Example

```
//Start AD conversion

Set_AD_Start(1);
```

See Also

Get_AD_Start(), Set_AD_Last()

3.8.2 Get_AD_Start

```
void Get_AD_Start (U8 *Enable);
```

Description Get the setting of enabling AD conversion.

Parameters

Enable

- 1 → AD conversion is enabled.
- 0 → AD conversion is disabled.

Return Value None

Example

```
// Get the current AD conversion setting

U8 AD_enable;
Get_AD_Start(&AD_enable);
```

See Also

Set_AD_Start()

3.8.3 Set_AD_Last

```
I16 Set_AD_Last (U16 Value);
```

Description Set the last channel number for AD conversion. For example, if value is 5, AD0 ~ AD5 will do conversion after function **Set_AD_Start** (1) is called.

Parameters

Value is the last channel number for AD conversion. It must in the range of 0 ~ 6; default is 6.

Return Value ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
//Set AD5 as the last channel for AD conversion.

I16   error;
error = Set_AD_Last(5);
```

See Also

Get_AD_Last(), Set_AD_Start()

3.8.4 Get_AD_Last

```
void Get_AD_Last (U16 *Value);
```

Description Get the setting of the last AD conversion channel number.

Parameters

Value is the last AD channel number. Please refer to function Set_AD_Last() for more information.

Return Value None

Example

```
U16 AD_last;
Get_AD_Last(&AD_last);
```

See Also

Set_AD_Last(), Set_AD_Start()

3.8.5 Set_AD6_Src

```
I16 Set_AD6_Src (U8 AD6_Src);
```

Description

Set the input source for AD6. User can set different input source for AD6 to calibrate DA. If analog ground or reference 5V is set as the input source, user can use the information to calibrate the AD.

Parameters

AD6_Src

is the input source of AD6. It must in the range of 1 ~ 8. The default value is 1. The definitions are in the following table.

Macro	*Value*	*Description*
AD6_SRC_AGND	1	Connect to Analog Ground
AD6_SRC_DA0	2	Connect to DA0
AD6_SRC_DA1	3	Connect to DA1
AD6_SRC_DA2	4	Connect to DA2
AD6_SRC_DA3	5	Connect to DA3
AD6_SRC_DA4	6	Connect to DA 4
AD6_SRC_DA5	7	Connect to DA 5
AD6_SRC_REF5V	8	Connect to Ref_5V*

*Note：Ref_5V is the reference 5V for the other circuits on this card.

Return Value

ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// AD6 is connected to DA0. So the AD6 can read
// the DA0 value; and user can use this value to
// calibrate DA0.

I16   error;
error = Set_AD6_Src(AD6_SRC_DA0);
```

See Also

Get_AD6_Src()

3.8.6 Get_AD6_Src

```
void Get_AD6_Src (U8 *AD6_Src);
```

Description Get the input source setting of AD6.

Parameters

AD6_Src

contains the source setting of AD6. Please refer to function Get_AD6_Src() for more information.

Return Value None

Example

```
U8 soruce;
Get_AD6_Src(&source);
```

See Also

Set_AD6_Src()

3.8.7 Get_AD_Data

```
I16 Get_AD_Data (U8 Ch, U16 *Value);
```

Description

Get a 16-bit integer value from the assigned AD channel. Since the conversion to a voltage value is complicated, the function, **Get_AD_Value()** is recommended instead.

Parameters

Ch is the assigned AD channel. It must be in the range of 0 ~ 6.

Value

is a 16-bit integer that represents a voltage in the range of -10V ~ +10V. The conversion table is defined as the following table.

16-bit Value	0x0000	0x2000	0x4000	0x6000	0x7fff	0x8000	0xa000	0xc000	0xe000	0xffff
voltage (V)	-0.0	-2.5	-5.0	-7.5	-10.0	10.0	7.5	5.0	2.5	+0.0

Return Value

ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
//AD_data is a 16-bit integer read from AD0

U16 AD_data;
Get_AD_Data(0, &AD_data);
```

See Also

Get_AD_Value()

3.8.8 Get_AD_Value

```
I16 Get_AD_Value (U8 Ch, F32 *Value);
```

Description Get a 32-bit floating point value from the assigned AD channel. This value represents the voltage read from the AD. It is used to replace function **Get_AD_Data()**.

Parameters

Ch is the assigned AD channel. It must be in the range of 0 ~ 6.

Value is a 32-bit floating point value that represents a voltage in the range of -10V ~ +10V. The unit is voltage. Please refer to **Get_AD_Data()** for more information.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
//Read the AD0 voltage value

F32   AD_value;
I16   error;
error = Get_AD_Value(0, &AD_value);
```

See Also

Get_AD_Data()

3.9 Calibration Functions

3.9.1 Write_Cal_Data

```
I16 Write_Cal_Data (U8 Ch, U8 Value);
```

Description Write a value to a calibration register.

Parameters

Ch is a register for calibration. Different values represent different calibration registers. They are defined in the following table.

Macro	*Value*	*Description*
DA0_SPAN_CAL_CH	1	Refer to Span register of DA0
DA1_SPAN_CAL_CH	2	Refer to Span register of DA1
DA2_SPAN_CAL_CH	3	Refer to Span register of DA2
DA3_SPAN_CAL_CH	4	Refer to Span register of DA3
DA4_SPAN_CAL_CH	5	Refer to Span register of DA4
DA5_SPAN_CAL_CH	6	Refer to Span register of DA5
AD_OFST_CAL_CH	7	Refer to Offset register of AD converter
AD_SPAN_CAL_CH	8	Refer to Span register of AD converter
DA0_OFST_CAL_CH	11	Refer to Offset register of DA0
DA1_OFST_CAL_CH	12	Refer to Offset register of DA1
DA2_OFST_CAL_CH	13	Refer to Offset register of DA2
DA3_OFST_CAL_CH	14	Refer to Offset register of DA3
DA4_OFST_CAL_CH	15	Refer to Offset register of DA4
DA5_OFST_CAL_CH	16	Refer to Offset register of DA5
R5V_OFST_CAL_CH	17	Refer to Offset register of Ref_5V

Value is the calibration value. It is in the range of 0 ~ 255.

Return Value ERR_TIMEOUT_ERROR
ERR_NO_ERROR

Example

```
//write 128 to the offset register of AD converter

I16   error;
error = Write_Cal_Data(AD_OFST_CAL_CH, 128);
```

See Also

Read_Cal_Data(), Save_All_Cal_Data()

3.9.2 Read_Cal_Data

```
I16 Read_Cal_Data(U8 Ch, U8 *Value);
```

Description Read out the value from a calibration register.

Parameters

Ch represents a calibration register. Please refer to function **Write_Cal_Data()** for more information.

Value is the readback value from a calibration register. It must be in the range of 0 ~ 255.

Return Value ERR_NO_ERROR
ERR_ADDR_OUT_RANGE

Example

```
//read the Offset calibration register of AD
// converter

U8    Cal_value;
I16   error;
error = Read_Cal_Data(AD_OFST_CAL_CH, &Cal_value);
```

See Also

Write_Cal_Data(), Save_All_Cal_Data()

3.9.3 Save_All_Cal_Data

```
I16 Save_All_Cal_Data();
```

Description Save all calibration values to Flash. The address for the calibration data starts from offset 0x000B0000 (CAL_DATA_FLASH_OFST). When the power of the control card turns on, all the DA span calibration registers are loaded to make sure the DA outputs are 0 voltages. The other calibration values will be loaded when **FPGA_Init()** is called.

Parameters None

Return Value

ERR_ADDR_OUT_RANGE
ERR_HW_ID_ERROR
ERR_TIMEOUT_ERROR
ERR_NO_ERROR

Example

```
I16   error;
error = Save_All_Cal_Data();
```

See Also

Write_Cal_Data(), Read_Cal_Data()

3.10 DDA Functions

DDA is essentially an algorithm for digital integration and generates a pulse train varying in frequency. The DDA will generate pulse rate corresponding to the desired feed rate. Your DSP program has to calculate the pulse numbers, and then allocates these numbers into the DDA pulse buffer. The DDA interpolator will generate equal-spaced pulse train corresponding to pulse number in the DDA pulse buffer. The number in the DDA buffer determines how many pulses will be generated during a DDA cycle. The pulse train frequency therefore depends on the pulse number and is constant in a DDA period. The DDA buffer automatically will be emptied after all the pulses have been generated. The duration of the DDA cycle can be set by API. In one DDA cycle the pulse number can be set from 0 to 8191.

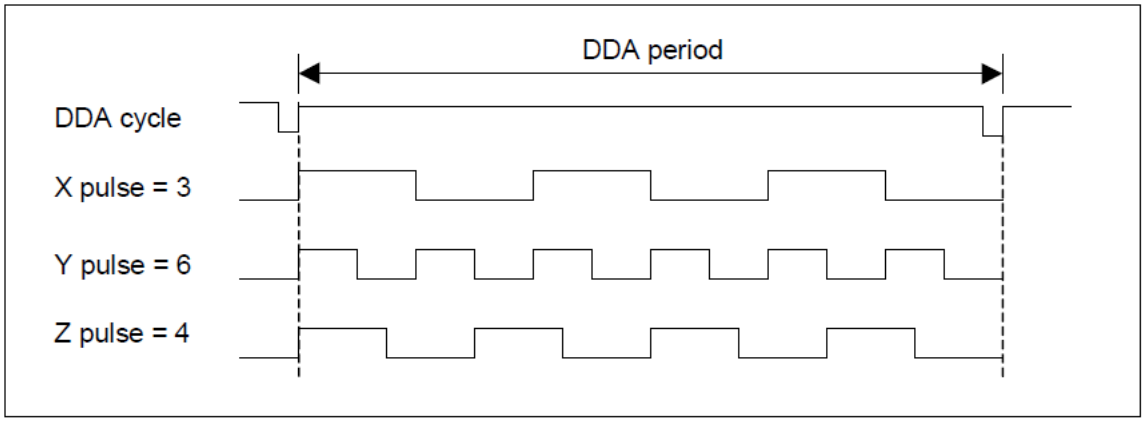


Figure 7: DDA mechanism

3.10.1 Set_DDA_GEn

```
void Set_DDA_GEn (U8 Enable);
```

Description Set the global control of DDA to be enabled or disabled. The DDA means the pulse out interfaces. If this global control is disabled, no pulse can send out from this card. If the global control is enabled, the pulse output of each axis still can be controlled by individual control setting.

Parameters

Enable is **0** for “disable”. It is the default setting.
Set *Enable* to be **1** for enabling the global control of pulse output interface.

Return Value None

Example

```
//Enable the DDA output
Set_DDA_GEn(1);
```

See Also

Get_DDA_GEn()

3.10.2 Get_DDA_GEn

```
void Get_DDA_GEn (U8 *Enable);
```

Description Get the global control setting of DDA.

Parameters

Enable is the global control of DDA interface. Value ***0*** indicates “disable”; and ***1*** indicates “enable”.

Return Value None

Example

```
U8 enable;
Get_DDA_Gen (&enable);
```

See Also

Set_DDA_GEn()

3.10.3 Get_DDA_Empty

```
I16 Get_DDA_Empty (U8 Axis, U8 *Value);
```

Description

Each axis's DDA (pulse output interface) has a **buffer**. Only when the buffer is empty, a new pulse command can be sent. This function gets the status of the buffer. However, an empty buffer does not mean there is no pulse out at that moment; it just indicates that a new command can be issued when buffer is empty. Please refer to **Get_DDA_Busy()** for more information.

Parameters

Axis is the assigned axis. It must be in the range of 0 ~ 5.

Value

- ***0***: DDA buffer is not empty
- ***1***: DDA buffer is empty
- If the buffer is empty a new position command can be issued. Writing new data to the DDA will cause malfunction if the DDA buffer is not empty. This must be avoided.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U8    empty;
I16   error;

// wait until the DDA buffer of axis 0 is empty
do{
      error = Get_DDA_Empty(0, &empty);
    } while (empty == 0);

//send 100 pulses to DDA of axis 0
error = Set_DDA_Data (0, 100);
```

See Also

Get_DDA_Busy()

3.10.4 Get_DDA_Busy

```
I16 Get_DDA_Busy (U8 Axis, U8 *Value);
```

Description Get the status of DDA output to see if there are pulses still outputting.

Parameters

Axis is the assigned axis. It must be in the range of 0 ~ 5.

Value

- ***0***, if no pulses are outputted.
- ***1***, if pulses are still being outputted from the assigned axis interface.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U8    busy;
I16   error;

// wait until axis0 does not send pulse any more
do{
     error = Get_DDA_Busy(0, &busy);
   } while (busy == 1);
```

See Also

Get_DDA_Empty()

3.10.5 Set_DDA_En

```
I16 Set_DDA_En (U8 Axis, U8 Enable);
```

Description Set the DDA output of an axis is enabled or not. Note: The DDA can send pulses out only when glob control from **Set_DDA_GEn()** is enabled and and this function is enabled, too.

Parameters

Axis is the assigned axis. It must be in the range of 0 ~ 5.

Enable

- ***0,*** if pulse is not allowed to send out. This is the default setting.
- **1**, DDA is allowed to send pulses out.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
//enable the axes 0 and 1 to send pulses out

I16   error;
error = Set_DDA_En(0, 1);
error = Set_DDA_En(1, 1);

Set_DDA_GEn(1);
```

See Also

Get_DDA_En(), Set_DDA_GEn();

3.10.6 Get_DDA_En

```
I16 Get_DDA_En (U8 Axis, U8 *Enable);
```

Description Get the DDA output control status of a axis.

Parameters

Axis is the assigned axis. It must be in the range of 0 ~ 5.

Enable

- 0, if the current DDA pulse output is disabled.
- 1, when DDA pulse output is enabled.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U8 enable;
I16   error;
error = Get_DDA_En(0, &enable);
```

See Also

Set_DDA_En(), Set_DDA_GEn();

3.10.7 Set_DDA_Clear_Ctl

```
I16 Set_DDA_Clear_Ctl (U8 Axis, U8 DDA_Clr_Src,
                       U8 Enable);
```

Description The DDA of the assigned axis will be cleared when EMG is tripped or that axis's PEL or MEL is tripped.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

DDA_Clr_Src

is defined as in following table.

Macro	*Value*	*Description*
CLR_SRC_EMG	0	Refer to the emergy stop (EMG)
CLR_SRC_PEL	1	Refer to the positive end limit (PEL)
CLR_SRC_MEL	2	Refer to the negative end limit (MEL)

Enable

- 1 – clears the DDA when the *DDA_Clr_Src* signal is triggered
- 0 – disables DDA clearing when the *DDA_Clr_Src* is triggered.

Return Value

ERR_AXIS_OUT_RANGE
ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
//Clear the DDA output of axis 0 when EMG is
triggered
I16   error;
error = Set_DDA_Clear_Ctl(0, CLR_SRC_EMG, 1);
```

See Also

Get_DDA_Clear_Ctl(), Set_DI_Pol(), Set_Axis_IO_Pol()

3.10.8 Get_DDA_Clear_Ctl

```
I16 Get_DDA_Clear_Ctl (U8 Axis, U8 DDA_Clr_Src,
                       U8 *Enable);
```

Description Get the DDA setting of a certain axis whether it will be cleared if either the EMG, the axis's PEL or MEL is tripped.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

DDA_Clr_Src
is defined as in following table.

Macro	*Value*	*Description*
CLR_SRC_EMG	0	Refer to the emergy stop (EMG)
CLR_SRC_PEL	1	Refer to the positive end limit (PEL)
CLR_SRC_MEL	2	Refer to the negative end limit (MEL)

Enable Enable displays whether the DDA will be cleard when the signal defined by *DDA_Clr_Src* is triggered. Please refer to function **Set_DDA_Clear_Ctl()** for more information.

Return Value

ERR_AXIS_OUT_RANGE
ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the setting of axis 0 to see whether the
// EMG will clear its DDA
U8 E_axis;
I16   error;
error = Get_DDA_Clear_Ctl (0, CLR_SRC_EMG,
&E_axis);
```

See Also

Set_DDA_Clear_Ctl(), Set_DI_Pol(), Set_Axis_IO_Pol()

3.10.9 Set_DDA_Length

```
I16 Set_DDA_Length (U8 Axis, U16 Value);
```

Description

Set the **DDA cycle time**. It is better to set the same update time for both servo and DDA. When the function **FPGA_Init()** is called for initializing the control card, the same value is set for both servo and DDA. It is preferred not to change this value later on.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Value decides the period of DDA. The range is 3 ~ 32767. The time unit is 40 ns. Therefore, the cycle time is:

DDA cycle time = *Value* * 40 ns

Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// set DDA cycle time of axis 0 to be 6250 * 40 ns
= 250 us

I16   error;
error = Set_DDA_Length(0, DEFAULT_DDA_LENGTH);
```

See Also

Get_DDA_Length(), FPGA_Init(), Set_Servo_Period()

3.10.10 Get_DDA_Length

```
I16 Get_DDA_Length (U8 Axis, U16 *Value);
```

Description Get the DDA cycle setting.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Value is the period of DDA. The range is 3 ~ 32767. The time unit is 40 ns. Therefore, the cycle time is:

DDA cycle time = (**Value*) * 40 ns

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U16   dda_cycle;
I16   error;
error = Get_DDA_Length(0, &dda_cycle);
//The DDA cycle time of axis 0 is (dda_cycle * 40)
ns
```

See Also

Get_DDA_Length(), FPGA_Init(), Set_Servo_Period()

3.10.11 Set_DDA_Inv

```
I16 Set_DDA_Inv (U8 Axis, U8 OUT_Inv, U8 DIR_Inv);
```

Description

Motor drives designed by various companies may have different definitions for their own pulse input interface. This card provides this function to let the pulse output (DDA) match these different definitions. Either OUT/DIR or CW/CCW pulse interface can be adjusted according to the motor drive.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

OUT_Inv
is 1, then OUT (or CW) signal will be inversed.

DIR_Inv
is 1, then DIR (or CCW) signal will be inversed.

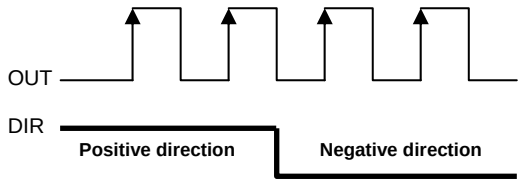
Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

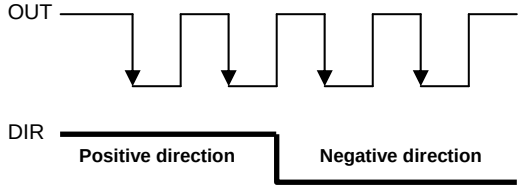
Example

```
Example 1:

//Set the DDA0 to OUT/DIR mode
I16   error;
error = Set_DDA_Pulse_Mode(0, 1);
```




```
//Inverse the OUT (Pulse) signal of DDA0
(DDA of axis 0)
error = Set_DDA_Inv(0, 1, 0);
```



See Also

Get_DDA_Inv()

3.10.12 Get_DDA_Inv

```
I16 Get_DDA_Inv (U8 Axis, U8 *OUT_Inv, U8 *DIR_Inv);
```

Description Get the setting of DDA inversion status.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

OUT_Inv
is 1, then OUT (or CW) signal is inversed.

DIR_Inv
is 1, then DIR (or CCW) signal is inversed.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the setting of DDA inversion status

U8    out_inv, dir_inv;
I16   error;
eror = Get_DDA_Inv(0, &out_inv, &dir_inv);
```

See Also

Set_DDA_Inv()

3.10.13 Set_DDA_Swap

```
I16 Set_DDA_Swap (U8 Axis, U8 Enable);
```

Description Enable/Disable the swapping of two DDA interface signals (OUT/DIR or CW/CCW). It allows users to solve some wiring problems.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Enable

- 1 - OUT (or CW) and DIR (or CCW) signals will swap.
- 0 - default value

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
//swap the DIR and OUT of DDA0
I16   error;
error = Set_DDA_Swap(0, 1);
```

See Also

Get_DDA_Swap()

3.10.14 Get_DDA_Swap

```
I16 Get_DDA_Swap (U8 Axis, U8 *Enable);
```

Description Get the DDA's swap setting of OUT (or CW) and DIR (or CCW) signals.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Enable is1, then the current swap setting is true.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U8    swap;
I16   error;
error = Set_DDA_Swap(0, &swap);
```

See Also

Get_DDA_Swap()

3.10.15 Set_DDA_Pulse_Mode

```
I16 Set_DDA_Pulse_Mode (U8 Axis, U8 Mode);
```

Description Set the DDA pulse mode of a certain axis to be OUT/DIR, CW/CCW, or AB-phase to match a motor drive.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Mode can be any mode in following table. Its default mode is 0.

Macro	*Value*	*Description*
PLS_MD_OUTDIR	0	OUT/DIR mode
PLS_MD_CWCCW	1	CW/CCW mode
PLS_MD_1xAB	2	A/B phases mode with a multiplier being 1.

Return Value ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
//Set the DDA0 to be CW/CCW mode.
I16   error;
error = Set_DDA_Pulse_Mode(0, 1);
```

See Also

Get_DDA_Pulse_Mode ()

3.10.16 Get_DDA_Pulse_Mode

```
I16 Get_DDA_Pulse_Mode (U8 Axis, U8 *Mode);
```

Description Get the DDA pulse mode setting of a specified axis.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Mode is the current setting for the assigned DDA interface. Please refer to function **Set_DDA_Pulse_Mode()** for more information.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U8 mode;
I16   error;
error = Get_DDA_Pulse_Mode(0, &mode);
```

See Also

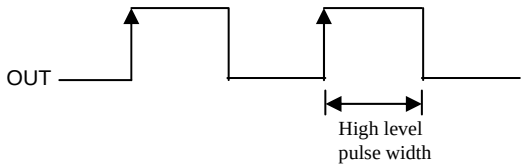
Set_DDA_Pulse_Mode ()

3.10.17 Set_DDA_Width

```
I16 Set_DDA_Width (U8 Axis, U16 Value);
```

Description

Set the high level pulse width for the DDA pulse mode. It affects the duty cycle of the pulse output.



Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Value must be 0 or 3 ~ 2047.
If *Value* = 0, the pulse will have 50% duty cycle.
If *Value is between* 3 ~ 2047, the definition will be

high level time = *Value* * 40 ns

Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
//Output pulses of DDA0 have 50% duty cycle
I16   error;
error = Set_DDA_Width (0, 0);
```

See Also

Get_DDA_Width(), Set_DDA_Length(), Set_DDA_Data()

3.10.18 Get_DDA_Width

```
I16 Get_DDA_Width (U8 Axis, U16 *Value);
```

Description Get the high level time setting of the assigned DDA.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Value The Value must be 0 or 3 ~ 2047. Please refer to function **Set_DDA_Width()** for more information.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
//Get the high level time setting of DDA0
U16   high_ext;
I16   error;
error = Get_DDA_Width (0, &high_ext);
```

See Also

Set_DDA_Width()

3.10.19 Set_DDA_Data

```
I16 Set_DDA_Data (U8 Axis, I16 Value);
```

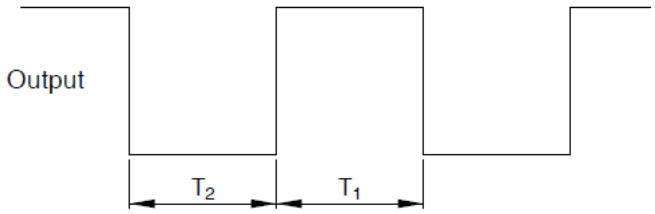
Description Set the pulse number of the assigned DDA to output in a cycle time.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Value The number of pulses to output in a cycle time. The valid range is 0 ~ +/-8191. The waveform of pulse is defined by three functions: **Set_DDA_Data()**, **Set_DDA_Width() and Set_DDA_Length()**. Therefore, the setting of the *Value* should be defined as follows to let the system work normally.
If DDA_Width is 0, (the waveform is 50% duty cycle T_1=T_2):

DDA_Data <= DDA_Length / 4



If DDA_Width is not 0, (the waveform has a fixed high pulse width T_1) :

DDA_Data <= DDA_Length / (DDA_Width + 3)

Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
I16   error;

// set cycle time of DDA0 = 6250 * 40 ns = 250 us:
error = Set_DDA_Length(0, DEFAULT_DDA_LENGTH);

// set the waveform of DDA0 to be 50 % duty cycle:
error = Set_DDA_Width (0, 0);

// DDA0 has to send 1000 pulses within the DDA
time // (250 us):
error = Set_DDA_Data (0, 1000);
```

See Also

Get_DDA_Data(), Set_DDA_Width(), Set_DDA_Length()

3.10.20 Get_DDA_Data

```
I16 Get_DDA_Data (U8 Axis, I16 *Value);
```

Description Get the pulse number that should be sent out during a DDA cycle time.

Parameters

Axis indicates the DDA channel; it must be in the range of 0 ~ 5.

Value The value represents the number of pulses to be outputted in a cycle time. Valid range 0 ~ +/-8191.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the pulse number that should be sent out
during
// a DDA cycle time.

I16   dda_data;
I16   error;
error = Get_DDA_Data (0, &dda_data);
```

See Also

Set_DDA_Data(), Set_DDA_Width(), Set_DDA_Length()

3.11 Encoder Functions

3.11.1 Clear_ENC_Cntr

```
I16 Clear_ENC_Cntr (U8 Axis);
```

Description Clear the counter value of the assigned encoder to zero.

Parameters

Axis -axis number. Its value must between 0 ~ 5.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
//Clear the counter value of encoder 0
I16   error;
error = Clear_ENC_Cntr(0);
```

See Also

Get_ENC_Cntr(), Set_ENC_Cntr()

3.11.2 Set_ENC_Inv

```
I16 Set_ENC_Inv (U8 Axis, U8 EA_Inv,
                 U8 EB_Inv, U8 EZ_Inv);
```

Description

This function can used for inversing the phases of encoder related signals. When the direction of encoder feedback is differrent from the direction of the motor, this function provides a very convient way to change the feedback direction without re-wiring the encoder interface.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

EA_Inv

- 1→ EA signal is inversed.
- 0→ default value.

EB_Inv

- 1→ EB signal is inversed.
- 0→ default value .

EZ_Inv

- 1→ EZ signal is inversed.
- 0→default value.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
//Inverse the EA of encoder 0
I16   error;
error = Set_ENC_Inv(0, 1, 0, 0);
```

See Also

Get_ENC_Inv()

3.11.3 Get_ENC_Inv

```
I16 Get_ENC_Inv (U8 Axis, U8 *EA_Inv,
                 U8 *EB_Inv, U8 *EZ_Inv);
```

Description Get the inversion setting of the specified encoder.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.
EA_Inv, EB_Inv and *EZ_Inv* record the settings of EA, EB and EZ, respectively. Please refer to function **Set_ENC_Inv()** for more information.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// A, B, Z stores the inversion setting of EA, EB
and EZ

U8    A, B, Z;
I16   error;
error = Get_ENC_Inv(0, &A, &B, &Z);
```

See Also

Set_ENC_Inv()

3.11.4 Set_ENC_Pulse_Mode

```
I16 Set_ENC_Pulse_Mode (U8 Axis, U8 Mode);
```

Description Set the encoder pulse mode of the assigned axis.

Parameters

Axis -axis number. Its value must between 0 ~ 5.

Mode can be defined according to the following table.

Macro	*Value*	*Description*
PLS_MD_OUTDIR	0	OUT/DIR mode
PLS_MD_CWCCW	1	CW/CCW mode
PLS_MD_1xAB	2	A/B phase mode, multiplier is 1
PLS_MD_2xAB	3	A/B phase mode, multiplier is 2
PLS_MD_4xAB	4	A/B phase mode, multiplier is 4

Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the encoder pulse mode to be AB phase mode
with the multiplier 4
I16   error;
error = Set_ENC_Pulse_Mode(0, PLS_MD_4xAB);
```

See Also

Get_ENC_Pulse_Mode()

3.11.5 Get_ENC_Pulse_Mode

```
I16 Get_ENC_Pulse_Mode (U8 Axis, U8 *Mode);
```

Description Get the encoder pulse mode of the assigned axis.

Parameters

Axis - axis number. Its value must between 0 ~ 5.

Mode is the mode setting. Please refer to function **Get_ENC_Pulse_Mode ()** for more information.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the encoder pulse mode

U8    mode;
I16   error;
error = Get_ENC_Pulse_Mode(0, &mode);
```

See Also

Get_ENC_Pulse_Mode()

3.11.6 Set_ENC_Cntr

```
I16 Set_ENC_Cntr (U8 Axis, I32 Value);
```

Description Set the encoder value of the assigned axis.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Value is the requested value after setting. It can be any signed 32-bit integer.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
//  Set the value of encoder 0 to be 0
I16   error;
error = Set_ENC_Cntr(0, 0);
//Similar to calling the Clear_ENC_Cntr(0)
// function
```

See Also

Get_ENC_Cntr(), Clear_ENC_Cntr()

3.11.7 Get_ENC_Cntr

```
I16 Get_ENC_Cntr (U8 Axis, I32 *Value);
```

Description Get the encoder value of the assigned axis.

Parameters

Axis - axis number. Its value must between 0 ~ 5.

Value - value read from the specified encoder.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Read the counter value of encoder 0

I32   value;
I16   error;
error = Get_ENC_Cntr(0, &value);
```

See Also

Set_ENC_Cntr()

3.11.8 Set_ENC_Vring

```
I16 Set_ENC_Vring (U8 Axis, U32 Value);
```

Description Set the encoder counter to be a specified ring counter with an assigned maximum value.

Parameters

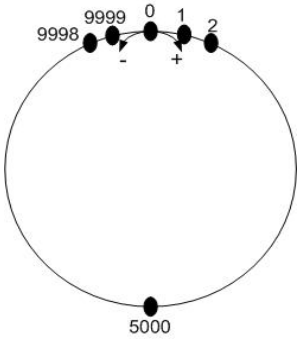
Axis -axis number. Its value must between 0 ~ 5.

Value is a positive 31-bit integer (0 ~ 0x7FFFFFFF). Its default value is 0x7FFFFFFF.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the encoder 0 to be a ring counter with a
// maximum value 9999
I16   error;
error = Set_ENC_Vring(0, 9999);
```



For example:
Design a ring counter with a resolution 10000 pulses per revolution. Then, the maximum value of this ring counter should be 9999. Each time when the encoder counts up to 9999, its next count should return to 0.

See Also

Get_ENC_Vring()

3.11.9 Get_ENC_Vring

```
I16 Get_ENC_Vring (U8 Axis, U32 *Value);
```

Description Get the ring counter setting of an assigned axis.

Parameters

Axis -axis number. Its value must between 0 ~ 5.

Value is a positive 31-bit integer (0 ~ 0x7FFFFFFF). Please refer to function **Set_ENC_Vring()** for more information.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U32   ring;
I16   error;
error = Get_ENC_Vring(0, &ring);
```

See Also

Set_ENC_Vring()

3.11.10 Set_Index_Mode

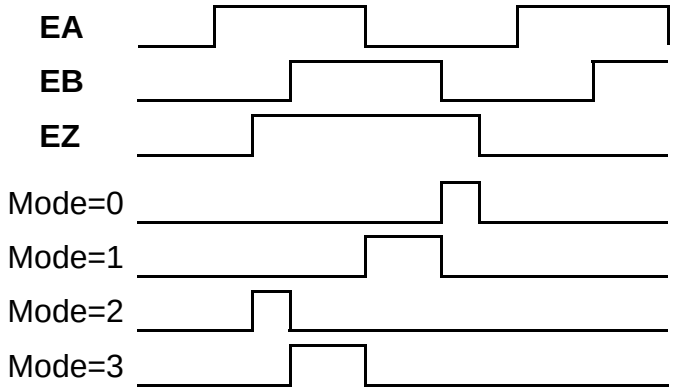
```
I16 Set_Index_Mode (U8 Axis, U8 Mode);
```

Description Set the mode of the Index signal. The Index signal is derived from **EZ** signal, and is used for providing more accurate reference position.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Mode selects the definition for **Index**. Please refer to the following signal graphs to choose suitable mode value for applications. Default Mode is 0. **EA** and **EB** are signals of the encoder interface.



Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
I16   error;
error = Set_Index_Mode(0, 1);
```

See Also

Get_Index_Mode, Get_Axis_IO(), Set_Axis_IO_Pol(), Set_Axis_IO_INT_En()

3.11.11 Get_Index_Mode

```
I16 Get_Index_Mode (U8 Axis, U8 *Mode);
```

Description Get the current mode setting of Index signal. The **Index** signal is derived from **EZ** signal, and is used for providing more accurate reference position.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Mode is the current setting of **Index**. Please refer to function **Set_Index_Mode()** for more information.

Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
U8    mode
I16   error;
error = Get_Index_Mode(0, &mode);
```

See Also

Set_Index_Mode

3.12 Manual Pulse Generator Functions

3.12.1 Set_MPG_EMG_En

```
void Set_MPG_EMG_En (U8 Enable);
```

Description Enable or disable the **EMG** of **MPG** (i.e. **MPG_EMG**) for being used as an **EMG** (emergency stop) signal to clear **DAC** and **DDA**.

Parameters

Enable

- 0: disable **MPG_ENG** to have the control over **DAC** and **DDA**. It is the default setting.
- 1: enable the clear action on **DAC** and **DDA**.

Return Value None

Example

```
// MPG_EMG can clear DAC and DDA
Set_MPG_EMG_En(1);
```

See Also

Get_MPG_EMG_En(), Set_DA_Clear_Ctl(), Set_DDA_Clear_Ctl()

3.12.2 Get_MPG_EMG_En

```
void Get_MPG_EMG_En (U8 *Enable);
```

Description Check whether **EMG** of **MPG** (i.e. **MPG_EMG**) is used as an **EMG** (emergency stop) signal to clear **DAC** and **DDA**.

Parameters

Enable

- 0: clear **DAC** and **DDA** function are disabled.
- 1: clear function is enabled.

Return Value None

Example

```
// get the clear function of MPG_EMG is enabled.

U8 enable;
Get_MPG_EMG_En(&enable);
```

See Also

Set_MPG_EMG_En(), Set_DA_Clear_Ctl(), Set_DDA_Clear_Ctl()

3.12.3 Set_MPG_Inv

```
I16 Set_MPG_Inv (U8 EA_Inv, U8 EB_Inv);
```

Description

Set the pulse input signals of **MPG** to be inversed or not. When the pulse input direction is different from the definition of desired motion, this function can provide a convient way to change the direction definiton without re-wiring.

Parameters

EA_Inv

– 1 for inversing the **EA** signal; default value is 0.

EB_Inv

– 1 for inversing the **EB** signal; default value is 0.

Return Value

ERR_NO_ERROR

Example

```
// reverse the EB signal of MPG
I16   error;
error = Set_MPG_Inv(0, 1);
```

See Also

Get_MPG_Inv()

3.12.4 Get_MPG_Inv

```
void Get_MPG_Inv (U8 *EA_Inv, U8 *EB_Inv);
```

Description Get the invrsion settings of **EA** and **EB** signals of **MPG**.

Parameters *EA_Inv* and *EB_Inv* record the invsersion settings of **EA** and **EB** signals of **MPG**, respectively. Please refer to function **Set_MPG_Inv()** for more information.

Return Value None

Example

```
// A, B are the inverse settings of EA and EB
// signals of MPG

U8 A, B;
Get_MPG_Inv(&A, &B);
```

See Also

Set_MPG_Inv()

3.12.5 Set_MPG_Pulse_Mode

```
I16 Set_MPG_Pulse_Mode (U8 Mode);
```

Description Set the pulse input mode of **MPG**.

Parameters

Mode The mode definition is as follows:

Macro	*Value*	*Description*
PLS_MD_OUTDIR	0	OUT/DIR mode
PLS_MD_CWCCW	1	CW/CCW mode
PLS_MD_1xAB	2	A/B phase mode with multiplier 1
PLS_MD_2xAB	3	A/B phase mode with multiplier 2
PLS_MD_4xAB	4	A/B phase mode with multiplier 4

Return Value ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// set the MPG pulse input mode to be AB phase
// with multiplier 4
I16   error;
error = Set_MPG_Pulse_Mode(PLS_MD_4xAB);
```

See Also

Get_MPG_Pulse_Mode()

3.12.6 Get_MPG_Pulse_Mode

```
void Get_MPG_Pulse_Mode (U8 *Mode);
```

Description Get the input pulse mode setting of **MPG**.

Parameters Please refer to function **Set_MPG_Pulse_Mode()** for more information.

Return Value None

Example

```
// Get the input pulse mode setting of MPG

U8 mode;
Get_MPG_Pulse_Mode(&mode);
```

See Also

Get_ENC_Pulse_Mode()

3.12.7 Get_MPG_Cntr

```
void Get_MPG_Cntr (I8 *Value);
```

Description Get the pulse counter value of **MPG**. When **MPG** pulse counter is used for control axis motion, this value will be read as the command to move a specified axis. This counter can only be read inside the servo ISR (interrupt service routine). After the ISR is finished, this counter will be automatically cleared.

Parameters

Value is the **MPG** pulse counter value.

Return Value None

Example

```
// inside an ISR
I8 cmd;
Get_MPG_Cntr(&cmd);
```

See Also

3.13 Compare Functions

3.13.1 Set_Direct_CMP

```
I16 Set_Direct_CMP (U8 Axis, U8 Enable);
```

Description

The compare CMP+/CMP- pins are used for following two purposes:

- Position comparison output
  - If the compare mode is enabled (`Set_CMP_En(x, 1)`) a pulse will be outputted when the encoder position of the specified axis reaches a set position. In the compare mode this function **Set_Direct_CMP()** is disabled and can not set the output of **CMP**. Please refer to function **Set_CMP_En()** for more information.

- General purpose DO
  - If the CMP output is set to general purpose mode (`Set_CMP_En(x, 0)`**)** the state of the digital output can be directly controlled by function **Set_Direct_CMP()**.

Parameters

Axis -axis number. Its value must between 0 ~ 5.

Enable

- 0 : turn on the output transistor of CMP signal;
- 1 : turn off the output transistor of CMP signal; this is the default value.

Return Value

ERR_AXIS_OUT_RANGE
ERR_CMP_INUSE
ERR_NO_ERROR

Example

```
I16   error;

// disable position comparison of AXIS0
error = Set_CMP_En(0, 0);

// turn off the output transistor of CMP signal
error = Set_Direct_CMP(0, 1);
```

See Also

Get_Direct_CMP(), Set_CMP_En()

3.13.2 Get_Direct_CMP

```
I16 Get_Direct_CMP (U8 Axis, U8 *Enable);
```

Description Get the setting of Direct_CMP value. Please refer to function **Set_Direct_CMP()** for more information.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Enable is the setting value of **CMP**. Please refer to function **Set_Direct_CMP()** for more information.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
I16 value;
error = Get_Diret_CMP(0, &value);
```

See Also

Set_Direct_CMP()

3.13.3 Set_CMP_En

```
I16 Set_CMP_En (U8 Axis, U8 Enable);
```

Description

This function can set **CMP** to be a specific DO for the position comparison or just a general-purpose DO.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Enable

- 1: the **CMP** pins will become a specific DO for the position comparison. Each time the position comparison is true, **CMP** will output a pulse.
- 0: the **CMP** pins will become a general-purpose DO. The output has to be set by the **Set_Direct_CMP()** function.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Enable CMP for position comparison for axis 0
I16   error;
error = Set_CMP_En(0, 1);
```

See Also

Get_CMP_En(), Set_Direct_CMP()

3.13.4 Get_CMP_En

```
I16 Get_CMP_En (U8 Axis, U8 *Enable);
```

Description Get the setting of **CMP** to see whether it is used for position comparion or not.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Enable

- 1: **CMP** is in a specific DO mode for the position comparison.
- 0: **CMP** is in a general-purpose DI.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the setting of CMP0 to see whether it is
// used for position comparion or not.
U8    CMP_en;
I16   error;
error = Get_CMP_En(0, &CMP_en);
```

See Also

Set_CMP_En(), Set_Direct_CMP()

3.13.5 Set_CMP_Pol

```
I16 Set_CMP_Pol (U8 Axis, U8 CMP_Pol);
```

Description Set the polarity of **CMP** output to be a high or a low pulse.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.
Each time the position comparison is true, **CMP** will output a pulse. According to the value of *CMP_Pol*, this pulse can be a high or a low pulse.

CMP_Pol

- 1 : **CMP** output a high pulse.
- 0 : **CMP** output a low pulse.
  The pulse width is set by function **Set_CMP_Width()**.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// set CMP of axis 0 to output a high pulse when
// comparison result is true
error = Set_CMP_Pol(0, 1);
```

```
I16     error;
```

See Also

Set_CMP_Width(), Get_CMP_Pol()

3.13.6 Get_CMP_Pol

```
I16 Get_CMP_Pol (U8 Axis, U8 *CMP_Pol);
```

Description Get the polarity setting of **CMP**. This setting determines whether **CMP** will output a high or a low pulse.

Parameters

Axis - axis number. Its value must between 0 ~ 5.
Each time the position comparison is true, **CMP** will output a pulse. According to the value of *CMP_Pol* , this pulse can be a high or a low pulse.

CMP_Pol

- 1: **CMP** will output a high pulse.
- 0: **CMP** will output a low pulse.
  The pulse width is set by function **Set_CMP_Width()**.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
//Get the polarity setting of CMP of axis 0 (CMP0)

U8    CMP_pol;
I16   error;
error = Get_CMP_Pol(0, &CMP_pol);
```

See Also

Set_CMP_Width(), Set_CMP_Pol()

3.13.7 Set_CMP_Inc_En

```
I16 Set_CMP_Inc_En (U8 Axis, U8 Enable);
```

Description

This function enables the automatic shifting of the target position at fixed increments in positive or negative direction. The automatic increasing function is very suitable for high-speed and fixed-increment application. The increment is set by function **Set_CMP_Inc()**.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Enable

- 1: The comparison register will be increased automatically with a value specified by **Set_CMP_Inc().**
- 0: The automatic increasing function is disabled.This is the default value.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
I16   error;
// the target position for comparison will
automatically increase 100 pulses each time the
comparison condition is met.
error = Set_CMP_Inc(0, 100);

// enable automatically increase for axis 0
error = Set_CMP_Inc_En(0, 1);
```

See Also

Get_CMP_Inc_En(), Set_CMP_Inc()

3.13.8 Get_CMP_Inc_En

```
I16 Get_CMP_Inc_En (U8 Axis, U8 *Enable);
```

Description

Get the automatic-increasing setting of **CMP**. The automatic increasing function is very suitable for high-speed and fixed-increment application. Please refer to function **Set_CMP_Inc_En()** for more information.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Enable

- 1: The comparison register will be increased automatically with a value specified by **Set_CMP_Inc().**
- 0: The automatic increasing function is disabled.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Check if target position for comparison will
automatically increase or not.
U8    value
I16   error;
error = Get_CMP_Inc_En(0, &value);
```

See Also

Set_CMP_Inc_En()

3.13.9 Set_CMP_Width

```
I16 Set_CMP_Width (U8 Axis, U8 Value);
```

Description Set the output pulse width of **CMP**.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Value

Value must in the range of 0 ~ 15. The default value is 0. The width definition is as follows:

- If *Value* = 0 ➔ Do not adjust the width; output the true width.
- If *Value* got other value, its width is:

$2^{(Value+1)} * 40 \sim 2^{(Value+2)} * 40$ ns

Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
//Set the CMP pulse width to be (160ns ~ 320ns)
I16   error;
error = Set_CMP_Width(0, 1);
```

See Also

Get_CMP_Width()

3.13.10 Get_CMP_Width

```
I16 Get_CMP_Width (U8 Axis, U8 *Value);
```

Description Get the output pulse width setting of **CMP**.

Parameters

Axis -axis number. Its value must between 0 ~ 5.

Value

must be in the range of 0 ~ 15. Please refer to function **Set_CMP_Width()** for more information.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the pulse width setting of CMP0
U8    width;
I16   error;
error = Get_CMP_Width(0, &width);
```

See Also

Set_CMP_Width()

3.13.11 Set_CMP_Data

```
I16 Set_CMP_Data (U8 Axis, I32 Value);
```

Description Set the value of the assigned comparison register. Each axis has a comparison register; and this register can store a signed 32-bit integer.

Parameters

Axis -axis number. Its value must between 0 ~ 5.

Value is a signed 32-bit integer. It is used for comparison with the encoder value of the same axis. If the **Set_CMP_En()** is executed for enabling the comparing operation, the **CMP** will send pulse out when the comparison result is true.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the comparison register to be 20000
I16   error;
error = Set_CMP_Data(0, 20000);
```

See Also

Get_CMP_Data()

3.13.12 Get_CMP_Data

```
I16 Get_CMP_Data (U8 Axis, I32 *Value);
```

Description Get the value of the assigned comparison register. It is a signed 32-bit integer.

Parameters

Axis - axis number. Its value must between 0 ~ 5.

Value is the value stored in the comparison register.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
I32   cmp_data;
I16   error;
error = Get_CMP_Data(0, &cmp_data);
```

See Also

Set_CMP_Data()

3.13.13 Set_CMP_Inc

```
I16 Set_CMP_Inc (U8 Axis, I16 Value);
```

Description Set the increment for automatic comparison. This value will take effect only after function **Set_CMP_Inc_En()** sets automatic increasing function to be enabled. The automatic increasing function is very suitable for high-speed and fixed-increment application.

Parameters

Axis - axis number. Its value must between 0 ~ 5.

Value must be a signed 16-bit integer number. When the comparison and automatic-increasing function of an axis is enabled by using **Set_CMP_En()** and **Set_CMP_Inc_En()**, each time the encoder value of that axis is the same with the comparison register, the **CMP** will send out a pulse and then the *Value* will be added to the comparison register for next comparison.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Each time the comparison result is true, the
// comparison register will increase 100
// automatically.
I16   error;
error = Set_CMP_Inc(0, 100);
error = Set_CMP_Inc_En(0, 1);
```

See Also

Get_CMP_Inc(), Set_CMP_Inc_En()

3.13.14 Get_CMP_Inc

```
I16 Get_CMP_Inc (U8 Axis, I16 *Value);
```

Description Get the increment setting of automatic position comparison.

Parameters

Axis - axis number. Its value must between 0 ~ 5.

Value is the increment for each comparison. Please refer to function **Set_CMP_Inc()** for more information.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Read the increment value for comparison
I16   value
I16   error;
error = Get_CMP_Inc(0, &value);
```

See Also

Set_CMP_Inc()

3.13.15 Set_CMP_Out_Src

```
I16 Set_CMP_Out_Src (U8 Axis, U8 Value);
```

Description

Set the output of the assigned axis's physical CMP pin to be driven by an internal CMP signal. This function makes it possible to control several axes' CMP output pins by one specified internal comparison result. It is a special function; if not necessary, do not use this function.

Parameters

Axis indicates the axis number of a CMP pin. On the daughter board each CMP pin is assigned to a different axis. Its value must between 0 ~ 5.

Value is internal CMP number; its value is between 0 ~ 5. The default setting is equal to *Axis* to the axis number. Therefore, the internal CMP0 drives the output of physical CMP0 pin; the internal CMP01 drives the output of physical CMP1 pin; and so on.

Return Value

ERR_AXIS_OUT_RANGE
ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the internal CMP1 to drive the physical CMP
// pin of axis 0 (CMP0).

I16   error;
error = Set_CMP_Out_Src(0, 1)

// If the comparison result of axis 1 is true, it
// will cause CMP0 pin to output a pulse.
```

See Also

Get_CMP_Out_Src()

3.13.16 Get_CMP_Out_Src

```
I16 Get_CMP_Out_Src (U8 Axis, U8 *Value);
```

Description Returns for the requested axis CMP pin the designated internal compare action which output a pulse signal. 6 internal CMP actions are available. Please refer to function **Set_CMP_Out_Src()** for more information.

Parameters

Axis - axis number. Its value must between 0 ~ 5.

Value is the number of internal compare signal (CMPx).

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the reference source of CMP pin of axis 0.
U8 value;
I16   error;
error = Get_CMP_Out_Src(0, &value)
```

See Also

Set_CMP_Out_Src()

3.14 Latch Functions

3.14.1 Set_LTC_En

```
I16 Set_LTC_En (U8 Axis, U8 Enable);
```

Description Enable/disable the LATCH function of a specified axis. When the LATCH (**LTC)** is triggered externally with a digital input the current encoder position is captured and stored in LATCH register. Latch gets the encoder position immediately when it is being triggered. If it is disabled, the LTC becomes a normal DI without latch function. Please refer to function **Get_Axis_IO ()** for more information about the axis-specific DI.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Enable

- 0: Disable the LATCH function. Default is 0.
- 1: Enable the LATCH function.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Enable the LATCH function of AXIS0

I16   error;
error = Set_LTC_En(0, 1);
```

See Also

Get_LTC_En(), Get_Axis_IO ()

3.14.2 Get_LTC_En

```
I16 Get_LTC_En (U8 Axis, U8 *Enable);
```

Description Get the setting of LATCH function of a specific axis.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Enable

- 0: LATCH function is disabled.
- 1: LATCH function is enabled.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the setting of LATCH function of AXIS0
U8    enable;
I16   error;
error = Get_LTC_En(0, &enable);
```

See Also

Set_LTC_En()

3.14.3 Set_LTC_Pol

```
I16 Set_LTC_Pol (U8 Axis, U8 LTC_Pol);
```

Description Set the polarity of LTC input logic to be active-high or active-low.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

LTC_Pol

- 0: low-active (Default)
- 1: high-active

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the LTC0 to be high-active
I16   error;
error = Set_LTC_Pol(0, 1);
```

See Also

Get_LTC_Pol()

3.14.4 Get_LTC_Pol

```
I16 Get_LTC_Pol (U8 Axis, U8 *LTC_Pol);
```

Description Get the polarity setting of the assigned LTC.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

LTC_Pol

- 0: low-active
- 1: high-active

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Get the polarity setting of LTC0
U8    polarity;
I16   error;
error = Set_LTC_Pol(0, &polarity);
```

See Also

Set_LTC_Pol()

3.14.5 Get_LTC_Data

```
I16 Get_LTC_Data (U8 Axis, I32 *Value);
```

Description Get the encoder position from the assigned LATCH register. LATCH function has to be enabled by function **Set_LTC_En().**

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Value is the encoder value when LTC is triggered. It is a signed 32-bit integer.

Return Value ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
// Read the encoder position of the axis 0 (AXIS0)
U32   latch_data;
I16   error;
error = Get_LTC_Data(0, &latch_data);
```

See Also

Set_LTC_En(), Set_LTC_Pol()

3.14.6 Get_LTC_Error

```
I16 Get_LTC_Error (U8 Axis, U8 *Value);
```

Description

If a LATCH register does two latching operation without a read operation in between, the LTC_Error register would be set to 1 to indicate this error. Each time DSP reads the LATCH register, the LTC_Error register will be cleared to 0. Thus, the LTC_Error value allows user to check whether an latch value has been lost.

Parameters

Axis is the axis number. Its value must between 0 ~ 5.

Value

is defined as:

- 1: overrun error
- 0: latched value is valid.

Return Value

ERR_AXIS_OUT_RANGE
ERR_NO_ERROR

Example

```
U8    latch_error;
Get_LTC_Data (0, &latch_error);
```

See Also

Set_LTC_En(), Set_LTC_Pol(), Get_LTC_Data()

3.15 UART Functions

The UART is not a standard interface of this card. It needs special cable to do the communication. There are no buffer for transmitting and receiving. The related functions are very basic and are used just for testing. If users want to use this interface for their applications, they can build their functions based on these basic functions.

3.15.1 Set_UART_Baud

```
I16 Set_UART_Baud (U8 Baud);
```

Description Set the baud rate of UART. The protocol is E81 (Even parity, 8 data bits, 1 stop bit).

Parameters

Baud This value represents the baud rate of UART. Its range is 0 ~ 3; and its default value is 0. Its definition is as follows:

Macro	*Value*	*Description*
BAUD_4800	0	baud rate is 4800
BAUD_9600	1	baud rate is 9600 (Recommended*)
BAUD_19200	2	baud rate is19200
BAUD_38400	3	baud rate is 38400

*Note: Since the clock source of the card can not provide exact baud rate for each setting, it is better to use a baud rate of 9600 or 4800 for large data transfer. A little delay between two data is recommended.

Return Value ERR_VALUE_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the baud rate to 9600

Set_UART_Baud (BAUD_9600);
```

See Also

Get_UART_Baud()

3.15.2 Get_UART_Baud

```
void Get_UART_Baud (U8 *Baud);
```

Description Get the baud rate setting of UART.

Parameters

Baud is the current baud rate setting. Please refer to function **Set_UART_Baud()** for more information.

Return Value None

Example

```
U8    data;
Get_UART_Baud(&data);
```

See Also

Set_UART_Baud()

3.15.3 Get_UART_Sts

```
void Get_UART_Sts (U8 *Value);
```

Description Get the status of UART.

Parameters

Value contains the status value. The bits in the status value are defined in the following table.

Bit	7	6	5	4	3	2	1	0
Function	-	-	-	TxRdy	Overrun	FrmErr	PrtErr	RxRdy

Following constants are already defined in an include file. Users can use them in their program:

Macro	*Value*	*Description*
B_RX_RDY	0x01	If this bit is 1, there is a data in RX register.
B_PRT_ERR	0x02	If this bit is 1, there is a parity error when receiving.
B_FRM_ERR	0x04	If this bit is 1, there is a frame error when receiving.
B_OVERRUN	0x08	If this bit is 1, there is a data overrun error. It means new data comes in the RX register before the old data is read.
B_TX_RDY	0x10	If the bit is 1, new data is allowed to write to the TX register.

Return Value None

Example

```
U8    status, data;
do{
   Get_UART_Sts(&status);
  } while( (status & B_RX_RDY) != B_RX_RDY);

// wait until the RX data gets a data
if(   ( (status & B_PRT_ERR)  != B_PRT_ERR )
    &&( (status & B_FRM_ERR)  != B_FRM_ERR)   )

// make sure there is no error
  {
   // get the reveived data
   Get_UART_Data(&data);


   // wait TX register is ready
   do{
      Get_UART_Sts(&status);
```

```
     } while( (status & B_TX_RDY) != B_TX_RDY);

   // write data to TX register
   Set_UART_Data(data);
  }

// code lines listed above use do~while to check
// status; therefore, it will block other thread
// and is not suitable to run in a timer ISR.
// Similar functions can run in a timer ISR if the
// do~while structure is rewritten by if
// statements
```

See Also

Set_UART_Baud()

3.15.4 Set_UART_Data

```
void Set_UART_Data (U8 Value);
```

Description Write data to TX register of UART.

Parameters *Value* is an 8-bit data.

Return Value None

Example

```
U8    status, data = 1;

Get_UART_Sts(&status);
if ( status & B_TX_RDY)  Set_UART_Data(data);
// if TX register is ready for accepting new data,
//a data is sent to the TX register.
```

See Also

Get_UART_Data(), Get_UART_Sts()

3.15.5 Get_UART_Data

```
void Get_UART_Data (U8 *Value);
```

Description Get the data in RX register.

Parameters *Value* is the data stored in RX register.

Return Value None

Example

```
U8    status, data;

Get_UART_Sts(&status);
if (status & B_RX_RDY))  Get_UART_Data(&data);
// if data in RX register is ready, get the data
```

See Also

Set_UART_Data(), Get_UART_Sts()

3.16 DPRAM Functions

DPRAM (dual-port RAM) can be accessed by both the PC and the DSP. Therefore, it can be used for data transfer and communication between PC and DSP. The size provided by PMDK is 0x800 bytes (2048 bytes) or 0x400 words (1024 words). In general, the access is by word address. There are two special word addresses, **0x7FC** and **0x7FE**, that can generate interrupts. **0x7FC** is used by DSP when DSP needs to generate an interrupt to PC side. **0x7FE** is used by PC when PC needs to generate an interrupt to DSP side. The value written to the special address can be used as an event number for the interrupted side. Suitable action can be taken according to different event number.

3.16.1 Set_DPRAM_Int_Code

```
void Set_DPRAM_Int_Code (U16 Value);
```

Description Write a 16-bit data to DPRAM (DPRAM_ADDR, 0x90001800) offset **0x07FC** (DPR_L2H_INT_OFST).
This action will also send an interrupt to Host via the PCI bus.

Parameters

Value data to write to the DPRAM at offset x07FC. This can be a special message that DSP wants Host to know.

Return Value None

Example

```
// Generate an interrupt to host with a message
(0x01)

Set_DPRAM_Int_Code (0x01);
```

See Also

Get_DPRAM_Int_Code()

3.16.2 Get_DPRAM_Int_Code

```
void Get_DPRAM_Int_Code (U16 *Value);
```

Description DSP gets the message from Host. When Host writes data at the address of DPRAM (DPRAM_ADDR, 0x90001800) with offset **0x07FE** (DPR_H2L_INT_OFST), DPRAM will generate an interrupt to DSP if the related interrupt is enabled. When DSP read this address to get data, DPL_INT flag of the MAIN_INT_CH will be cleared at the same time.

Parameters

Value is the data that DSP reads from DPRAM with offset 0x07FE.

Return Value None

Example

```
// Clear the interrupt flag of DPL_INT
U16   Value;
Get_DPRAM_Int_Code(&Value);
```

See Also

Set_FPGA_Int_Factor(), Clr_FPGA_Int_Flag()

3.17 Miscellaneous Functions

3.17.1 Sleep_us

```
void Sleep_us (U32 time_us);
```

Description Program will delay the requested time; then continue to run.

Parameters

time_us is the requested delay time. The time unit is microseconds. Note: this delay is just an approximation and may easily be affected by interrupts.

Return Value None

Example

```
// Delay for 500 microseconds
Sleep_us(500);
```

See Also

3.17.2 Set_LED

```
void Set_LED (U8 On_nOff);
```

Description There are 2 LEDs on this control card. The LED on the right-hand side is controlled by FRnet. The other LED can be controlled by this function.

Parameters

On_nOff

is defined as

1: led is on;
0: led is off.

Return Value None

Example

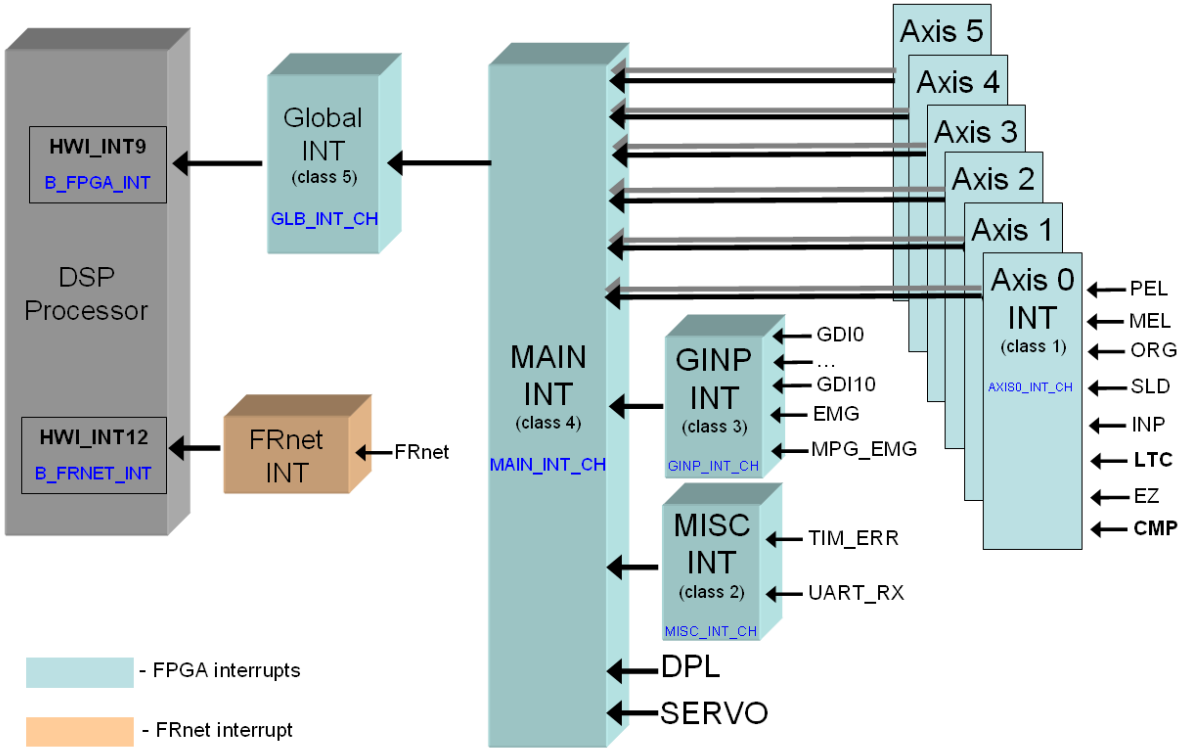
```
// Switch LED on
Set_LED (1);
```

See Also

4 Interrupt Functions

On the DSP two external interrupts are available for use: one for the FPGA and one for FRnet. The FPGA has more than 70 interrupt sources and the FRnet only one. The FPGA interrupt sources consist of

- axis interrupts: 9 interrupts for each axis,
- 11 general purpose interrupts,
- 1 emergency interrupt
- 1 manual pulse generator interrupt
- 1 UART interrupt and 1 timing error interrupt
- 1 Dual port RAM interrupt
- 1 SERVO or timer interrupt



4.1 DSP Interrupt Functions

4.1.1 Set_DSP_Int_Factor

```
void Set_DSP_Int_Factor (U8 Value);
```

Description Enables or disables the two interrupts available for the DSP, namely the FPGA and FRnet interrupts.

Parameters

Value

– contains the settings of interrupt factors. The two interrupt factors are set according to the following table.

Bit	*Signal*	*Function description*	*Enable/Disable*
0	FPGA_INT	FPGA interrupt	1 / 0
1	FRnet_INT	FRnet interrupt	1 / 0

Following macro definitions can be used in programming.

Macro	*Value*	*Description*
B_FPGA_INT	0x01	Enable FPGA interrupt (Please refer to FPGA interrupt functions.)
B_FRNET_INT	0x02	Enable FRnet interrupt (If the FRnet scan is enabled the interrupt is triggered every 0.72 ms. →see Enable_FRnet_Scan)

Return Value None

Example

```
// Enable FRnet and FPGA interrupts

//Method 1:
Set_DSP_Int_Factor(B_FRNET_INT | B_FPGA_INT);

//Methode 2:
Set_DSP_Int_Factor(0x03);
```

See Also

Get_DSP_Int_Factor()

4.1.2 Get_DSP_Int_Factor

```
U8 Get_DSP_Int_Factor (void);
```

Description Get the current interrupt factors settings of the DSP. Each bit represents an interrupt setting.

Parameters None

Return Value The returned value contains the current settings. Please refer to **Set_DSP_Int_Factor()** for more detailed explanations.

Example

```
// check if the FPGA interrupt of the DSP is
enabled:

if( (Get_DSP_Int_Factor() & B_FPGA_INT) ==
B_FPGA_INT)
{
   …
}
```

See Also

Set_DSP_Int_Factor()

4.2 FPGA-Controlled Interrupt Functions

4.2.1 Set_FPGA_Int_Factor

```
I16 Set_FPGA_Int_Factor (U8 Ch, U16 Value);
```

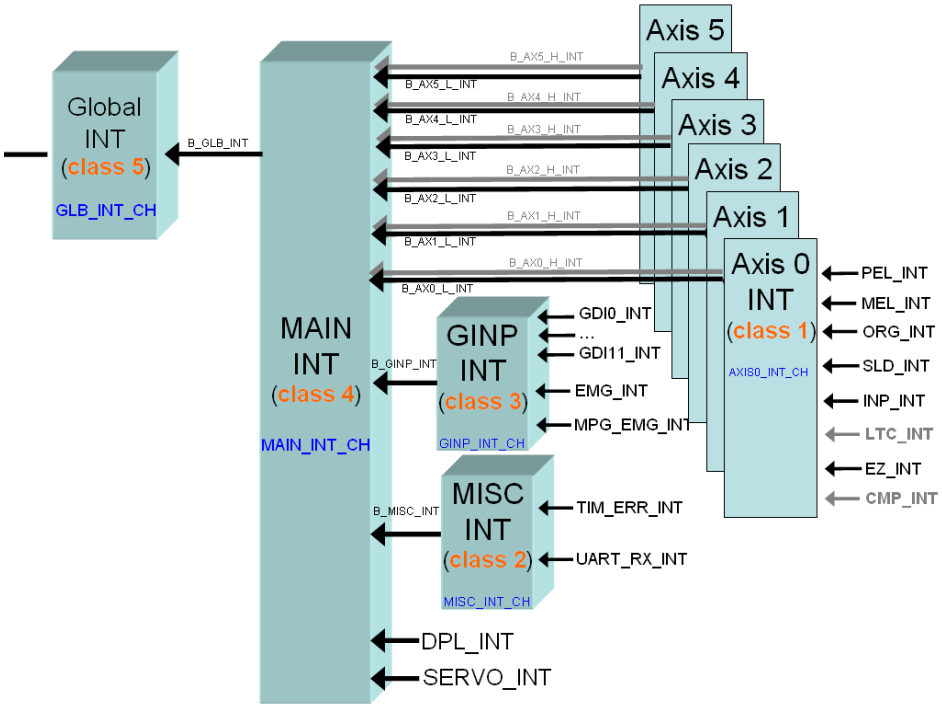
Description Enable/disable the FPGA-controlled interrupt factors.

Parameters

Ch

is the assigned channel or interrupt level. The interrupt level defines the source of the interrupt and is often referred to as the interrupt source. The interrupt level is NOT related to the interrupt priority. The FPGA has got 5 levels:

- Global interrupt level (class 5)
- Main interrupt level (class 4)
- General purpose interrupt level (class 3)
- Miscellaneous interrupt level (class 2)
- Axes interrupt level (class 1).



Value

This parameter is a 16-bit field (WORD). Each interrupt level has its *Value* definitions.

Return Value

ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

See 4.2.5

See Also

Get_FPGA_Int_Factor(), Get_FPGA_Int_Flag(),
Clear_FPGA_Int_Flag(), Get_Axis_IO_Pol(), Get_DI_Pol()

The following section describes the different mask levels with their *Value* definitions.

4.2.1.1 Axes interrupt level (class 1)

The axis interrupt level refers to the external interrupts of the DN-8368GB and DN-8368MB daughter boards.

Ch specifies one of the six axes interrupt levels (0 ~ 5). The interrupts for each axis has to be set individually.

Axis interrupt level	***Ch***
AXIS0_INT_CH	0
AXIS1_INT_CH	1
AXIS2_INT_CH	2
AXIS3_INT_CH	3
AXIS4_INT_CH	4
AXIS5_INT_CH	5

Value is a 16-bit field. The bits in *Value* are defined in following table. (1: enable; 0: disable)

Bit	7	6	5	4	3	2	1	0
Signal	LTC_INT	-	INP_INT	-	SLD_INT	ORG_INT	MEL_INT	PEL_INT
Bit	15	14	13	12	11	10	9	8
Signal	CMP_INT	-	-	-	-	-	EZ_INT	IDX_INT

Users can use the following constants for enabling the interrupt factors:

Macro	*Value*	*Description*	*Interrupt Pin*
B_PEL_INT	0x0001	Positive end limit (PEL)	LMT+
B_MEL_INT	0x0002	Negative end limit (MEL)	LMT-
B_ORG_INT	0x0004	Origin or Home (ORG)	HOME
B_SLD_INT	0x0008	Slow Down (SLD)	SLD
	0x0010		
B_INP_INT	0x0020	In Position (INP)	INP
	0x0040		
B_LTC_INT	0x0080	Latch (LTC)	LTC+/LTC-
B_IDX_INT	0x0100	Index (IDX)*	
B_EZ_INT	0x0200	Encoder Z (EZ)	Z-/Z+
B_CMP_INT	0x8000	Compare (CMP) The interrupt is generated by the FPGA when encoder position reaches a set position. In addition signal is outputted at pin CMP+/CMP-	-

*Note: Please refer to function **Set_Index_Mode()** for details.

Note: If the upper level interrupt control from the GLB_INT of GLB_INT_CH or the AXx_H_INT/ AXx_L_INT (x can be 0~5) of MAIN_INT_CH is not enabled but its corresponding interrupt factor is enabled, the related interrupt flag will be set to 1 when that signal is tripped; however, this trigger will not interrupt the DSP. In this way, the change of state is recorded without interrupting the DSP. User can devise a polling method to process this kind of events.
Please refer to function **Set_Axis_IO_Pol()** for more information.

4.2.1.2 Miscellaneous interrupt level (class 2)

Ch specifies UART and timing error as the interrupt source.

Miscellaneous interrupt level	*Ch*
MISC_INT_CH	250

Value is a 16-bit field. The bits in the value for enabling /disabling the two interrupt factors are defined in following table.
(1: enable; 0: disable)

Bit	7	6	5	4	3	2	1	0
Signal	-	-	-	-	-	UART_RX_INT	-	-
Bit	15	14	13	12	11	10	9	8
Signal	TIM_ERR_INT	-	-	-	-	-	-	-

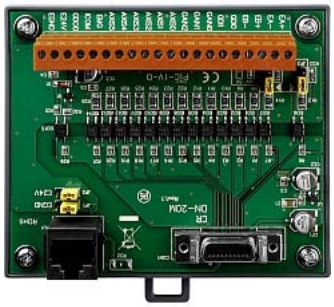
Following constant definitions can be used for the masking process.

Macro	*Value*	*Description*
B_UART_RX_INT	0x0004	UART RX Ready
B_TIM_ERR_INT	0x8000	Timing Error*

*Note: Timing Error means when a new servo interrupt (bit 14 of MAIN_INT_CH) happens before the pervious servo interrupt is cleared.

Note: Even if the upper three interrupt levels (class 5, class 4 and class 3) are disabled, the event of Timing Error will be logged in the flag register without interrupting the DSP.

4.2.1.3 General purpose interrupt level (class 3)



Ch specifies the external interrupts of the DN-20M daughter board (two external interrupts are on the daughterboard DN-8368) .

General Purpose interrupt level	*Ch*
GINP_INT_CH	253

Value is a 16-bit field (WORD). The bits in the value for enabling /disabling the 13 interrupt factors are defined in following table. (1: enable; 0: disable)

Bit	7	6	5	4	3	2	1	0
Signal	GDI7_INT	GDI6_INT	GDI5_INT	GDI4_INT	GDI3_INT	GDI2_INT	GDI1_INT	GDI0_INT
Bit	15	14	13	12	11	10	9	8
Signal	EMG_INT	MPG_EMG_INT	-	-	GDI11_INT	GDI10_INT	GDI9_INT	GDI8_INT

Following constant definitions can be used for enabling/ disabling interrupt factors and to mask the interrupt.

Macro	*Value*	*Description*	*DN-20M daughter Board port*
B_GDI0_INT	0x0001	DI channel 0	GDI0
B_GDI1_INT	0x0002	DI channel 1	GDI1
B_GDI2_INT	0x0004	DI channel 2	GAIN0
B_GDI3_INT	0x0008	DI channel 3	GAIN1
B_GDI4_INT	0x0010	DI channel 4	GAIN2
B_GDI5_INT	0x0020	DI channel 5	AXIS0
B_GDI6_INT	0x0040	DI channel 6	AXIS1
B_GDI7_INT	0x0080	DI channel 7	AXIS2
B_GDI8_INT	0x0100	DI channel 8	AXIS3
B_GDI9_INT	0x0200	DI channel 9	AXIS4
B_GDI10_INT	0x0400	DI channel 10	AXIS5
B_GDI11_INT	0x0800	DI channel 11	EMG/GDI11 (on the DN-8368 board for axis 3 to 5)
B_MPG_EMG_INT	0x4000	Emergency Stop from MPG (MPG_EMG)	EMG
B_EMG_INT	0x8000	Emergency Stop (EMG)	EMG/GDI11 (on the DN-8368 board for axis 0 to 2)

Note: Even though the upper two interrupt levels (class 5 and class 4) are disabled, the interrupt signal of the DI channels will be logged in the flag register without interrupting the DSP. User can devise a polling method to process DGI events.
Please refer to function **Set_DI_Pol()** for more information.

4.2.1.4 Main interrupt level (class 4)

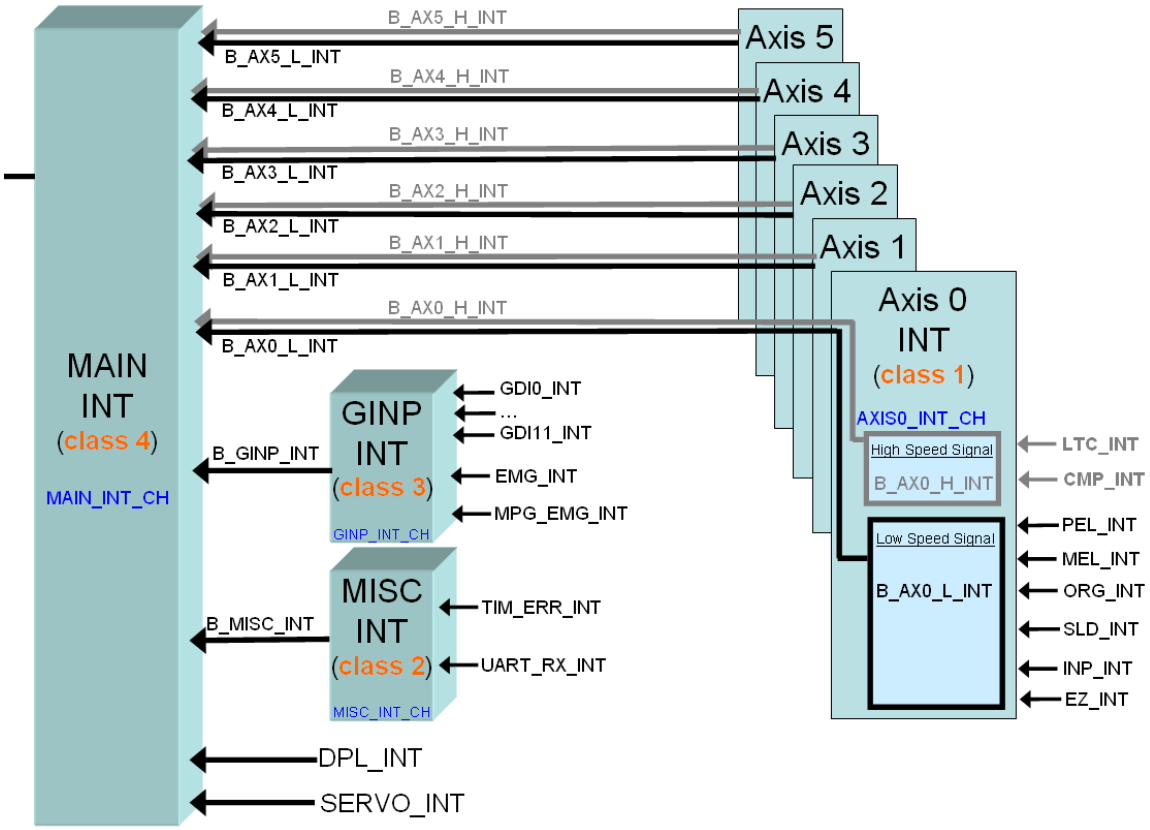
Ch specifies the higher level control for the different interrupt event groups.

Main interrupt level	***Ch***
MAIN_INT_CH	254

Value is a 16-bit field (WORD). The bits for enabling /disabling the 16 interrupt factors are defined in following table. (1: enable; 0: disable)

Bit	7	6	5	4	3	2	1	0
Signal	DPL_INT	GINP_INT	AX5_L_INT	AX4_L_INT	AX3_L_INT	AX2_L_INT	AX1_L_INT	AX0_L_INT
Bit	15	14	13	12	11	10	9	8
Signal	MISC_INT	SERVO_INT	AX5_H_INT	AX4_H_INT	AX3_H_INT	AX2_H_INT	AX1_H_INT	AX0_H_INT

Following constant definitions can be used in program to do masking process.



Macro	*Value*	*Description*
B_AX0_L_INT	0x0001	Interrupt from the low speed signal group of AXIS0 (AXIS0_INT_CH) *
B_AX1_L_INT	0x0002	Interrupt from the low speed signal group of AXIS1 (AXIS1_INT_CH) *
B_AX2_L_INT	0x0004	Interrupt from the low speed signal group of AXIS2 (AXIS2_INT_CH) *
B_AX3_L_INT	0x0008	Interrupt from the low speed signal group of AXIS3 (AXIS3_INT_CH) *
B_AX4_L_INT	0x0010	Interrupt from the low speed signal group of AXIS4 (AXIS4_INT_CH) *
B_AX5_L_INT	0x0020	Interrupt from the low speed signal group of AXIS5 (AXIS5_INT_CH) *
B_GINP_INT	0x0040	Interrupt from GDI channel (GINP_INT_CH)
B_DPL_INT	0x0080	Interrupt from DPRAM
B_AX0_H_INT	0x0100	Interrupt from the high speed signal group of AXIS0 (AXIS0_INT_CH) *

B_AX1_H_INT	0x0200	Interrupt from the high speed signal group of AXIS1 (AXIS1_INT_CH) *
B_AX2_H_INT	0x0400	Interrupt from the high speed signal group of AXIS2 (AXIS2_INT_CH) *
B_AX3_H_INT	0x0800	Interrupt from the high speed signal group of AXIS3 (AXIS3_INT_CH) *
B_AX4_H_INT	0x1000	Interrupt from the high speed signal group of AXIS4 (AXIS4_INT_CH) *
B_AX5_H_INT	0x2000	Interrupt from the high speed signal group of AXIS5 (AXIS5_INT_CH) *
B_SERVO_INT	0x4000	Interrupt from servo update timer
B_MISC_INT	0x8000	Interrupt from miscellaneous interrupt group (MISC_INT_CH)

*Note: The **low speed signal group** is composed of PEL, MEL, ORG, SLD, INP, IDX and EZ.
The **high speed signal group** is composed of CMP and LTC.

4.2.1.5 Global interrupt level (class 5)

Ch specifies the highest level for interrupt control (or global interrupt control).

Global interrupt level	*Ch*
GLB_INT_CH	255

Value is a 16-bit integer. Only bit 0 has effect.
(1: enable; 0: disable)

Following constant definitions can be used in program.

Macro	*Value*	*Description*
B_GLB_INT	0x0001	Global interrupt control

4.2.2 Get_FPGA_Int_Factor

```
I16 Get_FPGA_Int_Factor (U8 Ch, U16 *Value);
```

Description Get the interrupt factor settings.

Parameters

Ch

is the assigned channel value (interrupt level). There are 5 classes of interrupts. Please refert to function **Set_FPGA_Int_Factor()** for more information.

Value is a 16-bit field. The bits display the interrupt factor settings.
(1: enable; 0: disable)

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the interrupt factor setting of AXIS 0


U16   Value;
I16   error;
error = Get_FPGA_Int_Factor(AXIS0_INT_CH, &Value);
```

See Also

Set_FPGA_Int_Factor()

4.2.3 Get_FPGA_Int_Flag

```
I16 Get_FPGA_Int_Flag (U8 Ch, U16 *Value);
```

Description Get the FPGA-control interrupt flag. It is used to determine the type of interrupt that occurred in the FPGA.

Parameters

Ch is the assigned channel (interrupt level). There are 5 classes of interrupts. Please refert to function **Set_FPGA_Int_Factor()** for more information.
Note: Class 5 (GLB_INT_CH) does not support this function.

Value is a 16-bit field. The field displays the kind of interrupt occured.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
// Set the interrupt flag of AXIS 0


U16   Value;
I16   error;
error = Get_FPGA_Int_Flag(AXIS0_INT_CH, &Value);
```

See Also

Clr_FPGA_Int_Flag()

4.2.4 Clr_FPGA_Int_Flag

```
I16 Clr_FPGA_Int_Flag (U8 Ch, U16 Value);
```

Description Clear an assigned FPGA-controlled interrupt flag.

Parameters

Ch is the assigned channel, There are 5 classes of interrupts.

Value is a 16-bit field. This field is for clearing the FPGA interrupt flag. Write 1 to the bit representing the FPGA-controlled interrupt to clear the interrupt flag.
Please refer to function **Set_FPGA_Int_Factor()** for more information about *Ch* and *Value*.

Note:
Class 5 (GLB_INT_CH) does not support this function.
Class 2 (MISC_INT_CH) and Class 4(MAIN_INT_CH) have following restrictions:

- When *Ch* is MISC_INT_CH, TIM_ERR_INT interrupt can not be cleared; only reset FPGA can clear this interrupt.

- When *Ch* is MAIN_INT_CH,
  1. in order to clear the group interrupt of AXx_L_INT / GINP_INT / AXx_H_INT / MISC_INT, where x is 0~5, all interrupt factors belonging to that group must be cleared.
  2. in order to clear SERVO_INT, the *Value* has to be 0x4000 (B_SERVO_INT) for this function call.
  3. in order to clear DPL_INT, just call function **Get_DPRAM_Int_Code(U16 ****Value***)** or read the DPRAM (DPRAM_ADDR, 0x90001800) with offset 0x07FE (DPR_H2L_INT_OFST); that is, read memory from the address 0x90001FFE.

Return Value ERR_ADDR_OUT_RANGE
ERR_NO_ERROR

Example

```
// clear the interrupt flag of servo timer
// interrupt
I16   error;
```

```
error = Clr_FPGA_Int_Flag (MAIN_INT_CH,
                           B_SERVO_INT);

// clear the interrupt flag of latch LTC and
// compare CMP interrupt of axis 1
error = Clr_FPGA_Int_Flag (AXIS1_INT_CH,
                           B_LTC_INT|B_CMP_INT);

// clear the interrupt flag of general digital
// input GDI0, GDI1, GDI2:
error = Clr_FPGA_Int_Flag (GINP_INT_CH,
                           B_GDI0_INT|
                           B_GDI1_INT |
                           B_GDI2_INT);
```

See Also

Get_FPGA_Int_Flag(), Get_DPRAM_Int_Code()

4.2.5 FPGA Interrupt Examples

4.2.5.1 Example 1: Enabling HOME interrupt

This example demonstrates how to enable the HOME (ORG) interrupt of axis 3 (Figure 8).

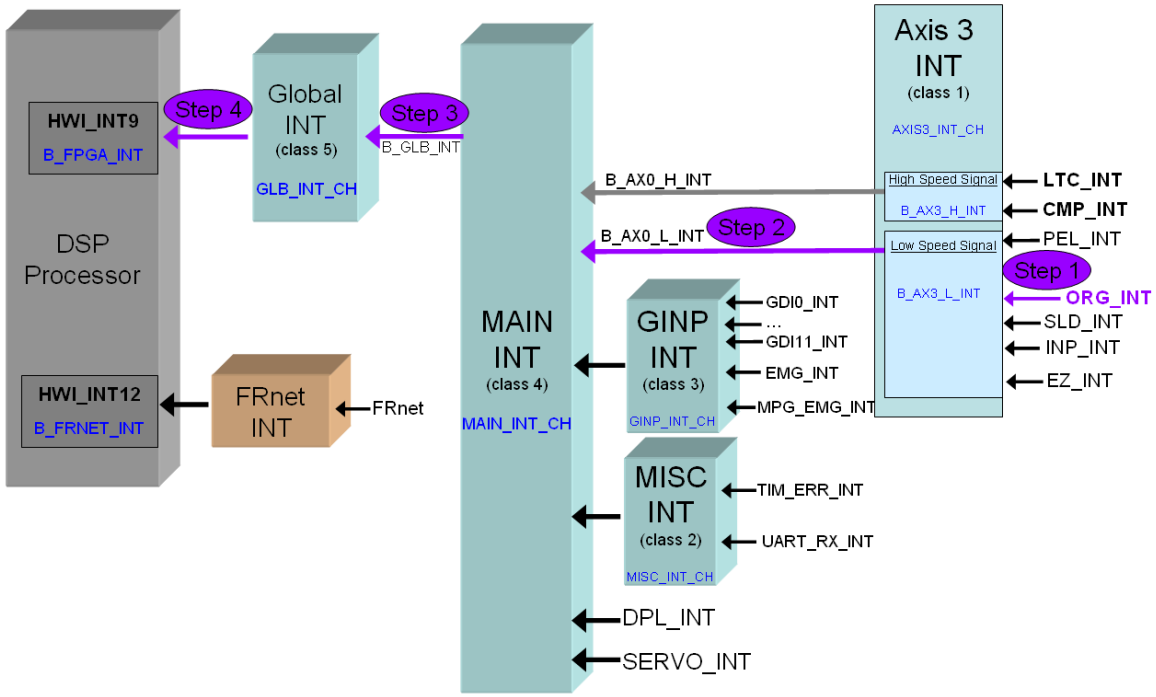


Figure 8: Interrupt levels

Procedure to enable the interrupt channels for the HOME input signal of axis 3:

Step 1: Enable the external ORG interrupt channel of axis 3:

```
Set_FPGA_Int_Factor( AXIS3_INT_CH, B_ORG_INT);
```

Step 2: Enable the axis 3 interrupt channel. As the ORG interrupt belongs to the low speed signal group the low speed interrupt channel (AX3_L_INT) of axis 3 is enabled.

```
Set_FPGA_Int_Factor(MAIN_INT_CH, B_AX3_L_INT);
```

Step 3: Enable the global FPGA interrupt channel

```
Set_FPGA_Int_Factor(GLB_INT_CH, B_GLB_INT);
```

Step 4: Enable the hardware interrupt channel 9 of the DSP chip

```
Set_DSP_Int_Factor(B_FPGA_INT);
```

Every HOME input signal results in an interrupt. All axis interrupts are managed by the FPGA chip and triggers a hardware interrupt of the DSP at channel 9. The hardware interrupt at channel 9 of the DSP chip calls the interrupt service routine FPGA_ISR(). In the ISR the type of FPGA interrupt has to be determined.

The following code shows how to detect a HOME interrupt of axis 3 in the ISR FPGA_ISR().

```
void FPGA_ISR()
{
     U16 usAxis3_Int_Flg;

      // STEP 1: Get the interrupt flag of axis 3:
     Get_FPGA_Int_Flag(AXIS3_INT_CH, & usAxis3_Int_Flg);


      if(usAxis3_Int_Flg)
     {

           // STEP 2: Check for HOME interrupt of axis 3
           if( (usAxis3_Int_Flg & B_ORG_INT) == B_ORG_INT)
           {

                 // STEP 3:  HOME interrupt has occurred.
                 //          Process the HOME interrupt.

           }

           // STEP 4: Clear the FPGA interrupt flag to enable
            //         the HOME interrupt of axis 3 again.
            Clr_FPGA_Int_Flag(MAIN_INT_CH, B_AX3_L_INT);
      }

}
```

4.2.5.2 Example 2: Enable several axis interrupts

This example describes how to enable the following interrupts for axis 0 (Figure 9):

- Positive end limit (PEL)
- Negative end limit (MEL)
- Origin or Home (ORG)
- Latch (LTC)
- Compare (CMP)

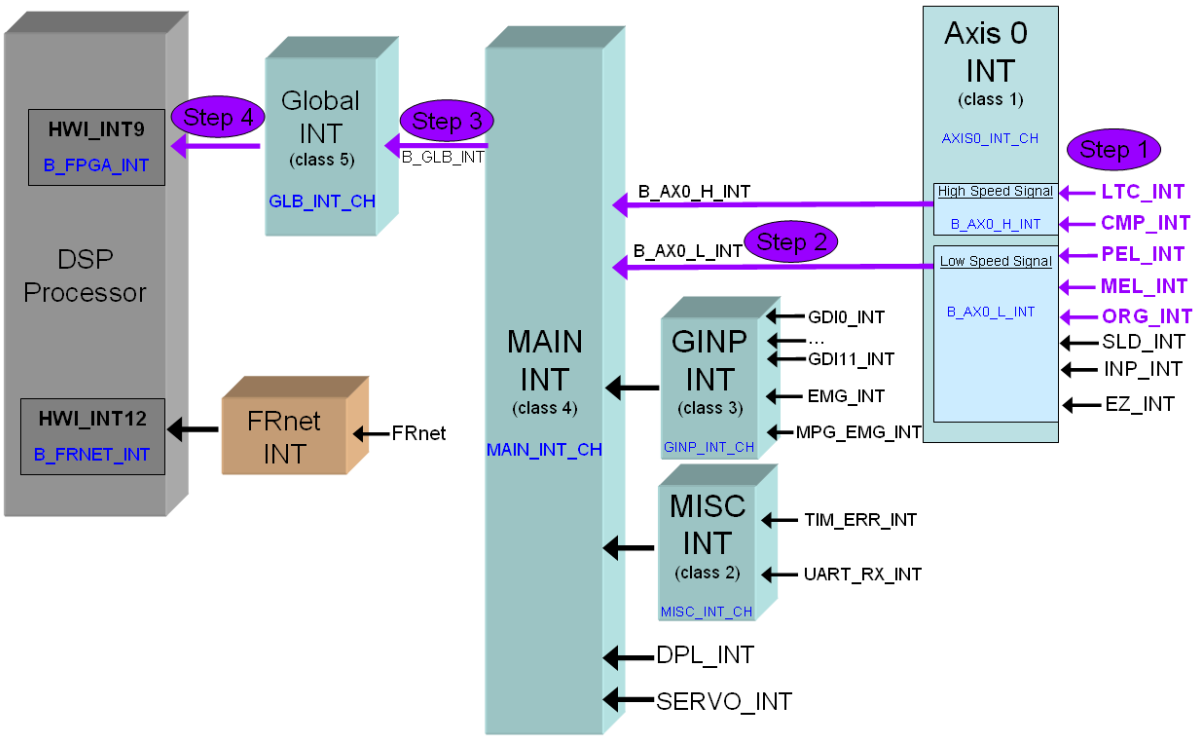


Figure 9: Interrupt channels

Procedure to enable the selected interrupt channels of axis 0:

Step 1: Enable the five interrupt channels of axis 0:

Macro	*Value*	*Description*
B_PEL_INT	**0x0001**	**Positive end limit (PEL)**
B_MEL_INT	**0x0002**	**Negative end limit (MEL)**
B_ORG_INT	**0x0004**	**Origin or Home (ORG)**
B_SLD_INT	0x0008	Slow Down (SLD)
B_INP_INT	0x0020	In Position (INP)
B_LTC_INT	**0x0080**	**Latch (LTC)**
B_IDX_INT	0x0100	Index (IDX)*

B_EZ_INT	0x0200	Encoder Z (EZ)
B_CMP_INT	**0x8000**	**Compare (CMP)**

```
Set_FPGA_Int_Factor( AXIS0_INT_CH,
                     B_PEL_INT |
                     B_MEL_INT |
                     B_ORG_INT |
                     B_LTC_INT |
                     B_CMP_INT );
```

Step 2: Enable the axis 0 interrupt channel. Both, the low and high speed interrupt channel of the axis have to be enabled as PEL, MEL and ORG use the low speed and LTC and CMP the high speed channel.

```
Set_FPGA_Int_Factor(MAIN_INT_CH,
                     B_AX0_L_INT | B_AX0_H_INT );
```

Step 3: Enable the global FPGA interrupt channel.

```
Set_FPGA_Int_Factor(GLB_INT_CH, B_GLB_INT);
```

Step 4: Enable the hardware interrupt channel 9 of the DSP chip.

```
Set_DSP_Int_Factor(B_FPGA_INT);
```

The following code shows how to identify the individual interrupt in case of an FPGA interrupt.

```
void FPGA_ISR()
{
     U16 usAxis0_Int_Flg;

      // STEP 1: Get the interrupt flag of axis 0:
     Get_FPGA_Int_Flag(AXIS0_INT_CH, & usAxis0_Int_Flg);


      if(usAxis0_Int_Flg)
         {
           // STEP 2: Determine the interrupt factor

            // Positive end limit (PEL):
           if( (usAxis0_Int_Flg & B_PEL_INT) == B_PEL_INT){…}

           // Negative end limit (PEL):
           if( (usAxis0_Int_Flg & B_MEL_INT) == B_MEL_INT){…}

           // HOME (ORG):
           if( (usAxis0_Int_Flg & B_ORG_INT) == B_ORG_INT)    {…}

            // Latch (LTC):
```

```
            if( (usAxis0_Int_Flg & B_LTC_INT) == B_LTC_INT)    {…}

             // Compare (ORG):
            if( (usAxis0_Int_Flg & B_CMP_INT) == B_CMP_INT)    {…}



            // STEP 3: Clear the FPGA interrupt flag to enable
             //         the axis 0 to receive the next interrupt.
             Clr_FPGA_Int_Flag(MAIN_INT_CH,
                               B_AX0_L_INT | B_AX0_H_INT );
      }
```

4.2.5.3 Example 3: SERVO (Timer) interrupt

The servo timer of the FPGA sends an interrupt signal to the hardware interrupt channel 9 of the DSP chip at fixed time intervals (Figure 10).

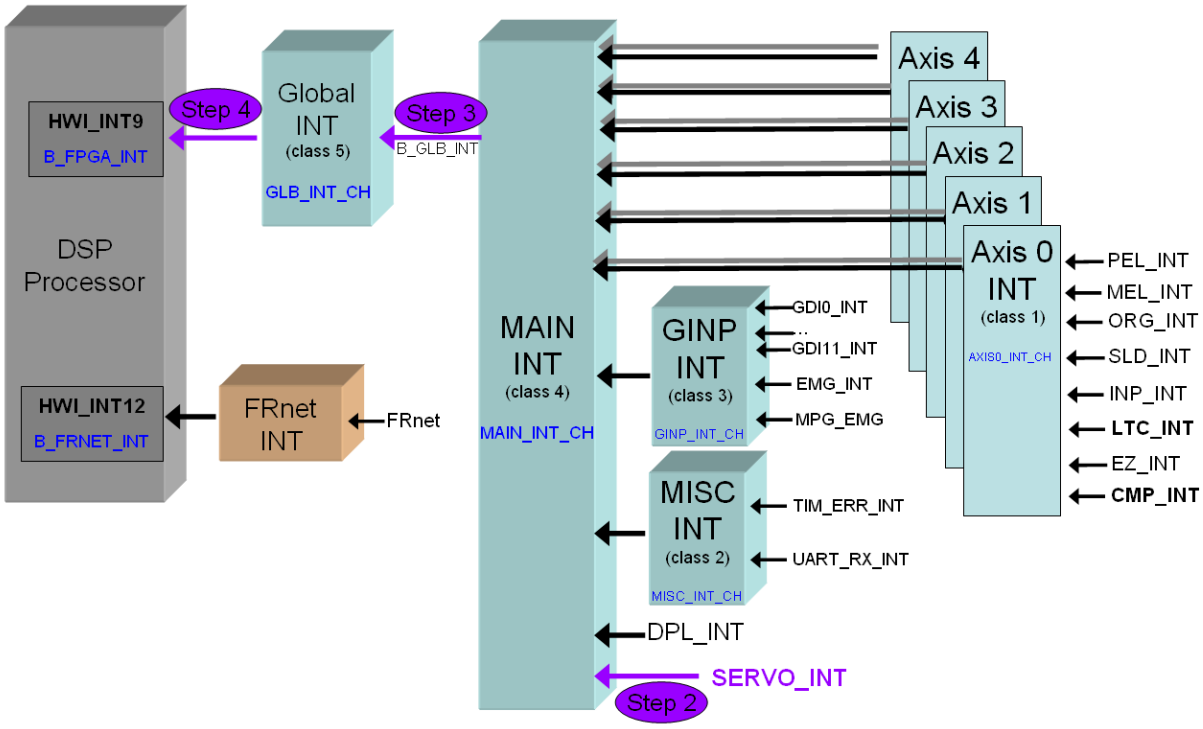


Figure 10: Enabling SERVO interrupt

Procedure to enable the SERVO interrupt channel:

Step 1: In the main() initialize the FPGA and set the SERVO interrupt time interval:

```
FPGA_Init(FPGA_SRC_FLASH, DEFAULT_DDA_LENGTH);
```

DEFAULT_DDA_LENGTH is defined as 6250.
The interrupt time interval is therefore
6250 * 40 ns = 250000ns = 250 us

Step 2: Enable the SERVO interrupt channel:

```
Set_FPGA_Int_Factor(MAIN_INT_CH, B_SERVO_INT);
```

Step 3: Enable the global FPGA interrupt channel

```
Set_FPGA_Int_Factor(GLB_INT_CH, B_GLB_INT);
```

Step 4: Enable the hardware interrupt channel 9 of the DSP chip

```
Set_DSP_Int_Factor(B_FPGA_INT);
```

The following code shows how to detect a SERVO interrupt in the ISR FPGA_ISR().

```
void FPGA_ISR()
{
     U16 usServo_Int_Flg;

      // STEP 1: Get the interrupt flag:
     Get_FPGA_Int_Flag(MAIN_INT_CH, &usServo_Int_Flg);


      if(usServo_Int_Flg)
     {

           // STEP 2: Check for SERVO interrupt
           if( (usServo_Int_Flg & B_SERVO_INT) == B_SERVO_INT)

           {

                 // STEP 3:  Process the SERVO interrupt.

           }

           // STEP 4: Clear the FPGA interrupt flag to enable
            //         the SERVO channel for the next interrupt.
            Clr_FPGA_Int_Flag(MAIN_INT_CH, B_SERVO_INT);
      }

}
```

4.2.5.4 Example 4: Enable emergency (EMG) interrupt

The emergency interrupt is an external interrupt. The channel EMG/GDI11 is on the DN-8368 board for axis 3 to 5).

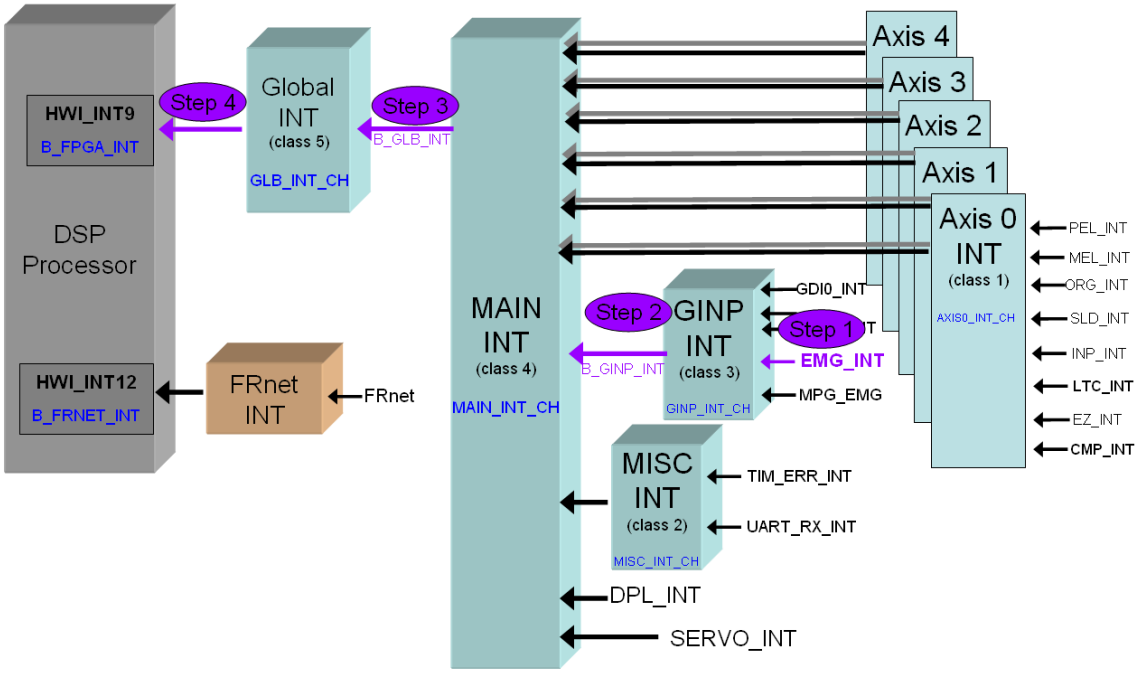


Figure 11: Enabling emergency interrupt

Procedure to enable the emergency interrupt channel:

Step 1: Enable the external emergency interrupt channel:

```
Set_FPGA_Int_Factor(GINP_INT_CH, B_EMG_INT);
```

Step 2: Enable the general purpose interrupt level (class 3):

```
Set_FPGA_Int_Factor(MAIN_INT_CH, B_GINP_INT);
```

Step 3: Enable the main interrupt level (class 4):

```
Set_FPGA_Int_Factor(GLB_INT_CH, B_GLB_INT);
```

Step 4: Enable the hardware interrupt channel 9 of the DSP chip

```
Set_DSP_Int_Factor(B_FPGA_INT);
```

The following code shows how to detect a emergency interrupt in the ISR FPGA_ISR().

```
void FPGA_ISR()
{
     U16 usEMG_Int_Flg;

      // STEP 1: Get the interrupt flag:
     Get_FPGA_Int_Flag(GINP_INT_CH, & usEMG_Int_Flg);


      if(usEMG_Int_Flg)
     {

           // STEP 2: Check for EMG interrupt
           if( (usEMG_Int_Flg & B_EMG_INT) == B_EMG_INT)
           {

                 // STEP 3:  Process the SERVO interrupt.

           }

           // STEP 4: Clear the FPGA interrupt flag to enable
            //         the EMG channel for the next interrupt.
            Clr_FPGA_Int_Flag(GINP_INT, B_EMG_INT);
      }

}
```

5 Appendix

5.1 Definitions of Constants

5.1.1 Definitions of Data Types

```
typedef     char            I8  ;
typedef     short           I16 ;
typedef     int             I32 ;
typedef     long            I40 ;
typedef     unsigned char   U8  ;
typedef     unsigned short  U16 ;
typedef     unsigned int    U32 ;
typedef     unsigned long   U40 ;
typedef     float           F32 ;
typedef     double          F64 ;
```

5.1.2 Constants in ICPDAS_PMDK.h

Macro	*Value*	*Description*
SDRAM_ADDR	0x80000000	Start address of the SDRAM Range: 0x80000000~0x80FFFFFF
FLASH_ADDR	0x90000000	Start address of a FLASH page. The Page number is defined by GPIO and has the range from 0 to 0xFF. Range of each page: 0x90000000~0x90000FFF
FPGA_ADDR	0x90001000	Start address of the FPGA. Range: 0x90001000~0x900013FF
CPLD_ADDR	0x90001400	Start address of the CPLD. Range: 0x90001400~0x900017FF
DPRAM_ADDR	0x90001800	Start address of the DPRAM Range: 0x90001800~0x90001FFF
FRAM_ADDR	0x90002000	Start address of an FRAM page. The page is defined by the GPIO and has the range from 0 to 0x1F. Range of each page: 0x90002000~0x90003FFF

Macro	*Value*	*Description*
SDRAM_SIZE	0x01000000	Size of SDRAM
FLASH_SIZE	0x00100000	Total size of all FLASH memories. There are 256 pages of Flash defined by GPIO. Numbers are from 0 to 0xFF.
FPGA_SIZE	0x0400	Size of FPGA
CPLD_SIZE	0x0040	Size of CPLD
DPRAM_SIZE	0x0800	Size of DPRAM
FRAM_SIZE	0x00040000	Total size of all FRAM memories. There are 32 pages of FRAM defined by GPIO. Numbers are from 0 to 0x1F.

Definitions of Constants

```
#define     MATH_PI             3.141592653589793

#define     MY_TRUE             1
#define     MY_FALSE            0
#define     MY_MINUS            (-1)

#define     MAX_AXIS_NO         6
#define     MAX_DA_CHANNEL      MAX_AXIS_NO
#define     MAX_AD_CHANNEL      MAX_AXIS_NO+1
#define     MAX_DDA_CHANNEL     MAX_AXIS_NO
#define     MAX_ENC_CHANNEL     MAX_AXIS_NO
```

5.1.3 Definition of Return Code

```
#define  ERR_NO_ERROR              0
#define  ERR_TIMEOUT_ERROR         (-1)
#define  ERR_HW_ID_ERROR           (-2)
#define  ERR_AXIS_OUT_RANGE        (-3)
#define  ERR_ADDR_OUT_RANGE        (-4)
#define  ERR_VALUE_OUT_RANGE       (-5)
#define  ERR_FPGA_DL_FAILED        (-6)
#define  ERR_DA_AUTO_UPDATE        (-101)
#define  ERR_DA_BUSY               (-102)
#define  ERR_CMP_INUSE             (-201)
#define  ERR_NOT_IMPLEMENT         (-32768)
```

5.1.4 Constants in PMDKdsp.h

Following definitions are used for the "Region" argument of the function *Flash_Erase()*

Macro	*Value*	*Description*
FLASH_ERASE_CHIP	0	Clear all FLASH memories Range: 0x00000~0xFFFFF
FLASH_ERASE_CODE	1	Clear DSP code region Range: 0x00000~0x9FFFF
FLASH_ERASE_RSV	2	Clear Reserved region Range: 0xA0000~0xAFFFF
FLASH_ERASE_CAL	3	Clear AD/DA calibration data region Range: 0xB0000~0xBFFFF
FLASH_ERASE_FPGA	4	Clear FPGA Configuration Data region Range: 0xC0000~0xFFFFF

Offset address values for the Flash configuration

Macro	*Value*	*Description*
DSP_CODE_FLASH_OFST	0x00000000	Offset address of DSP code
RSV_DATA_FLASH_OFST	0x000A0000	Not configured memory
CAL_DATA_FLASH_OFST	0x000B0000	Offset address of AD/DA calibration data
FPGA_DATA_FLASH_OFST	0x000C0000	Offset address of FPGA Configuration Data

Memory sizes on Flash

Macro	*Value*	*Description*
DSP_CODE_FLASH_SIZE	0x000A0000	Memory size of the DSP code
RSV_DATA_FLASH_SIZE	0x00010000	Size of the not configured memory
CAL_DATA_FLASH_SIZE	0x00010000	Memory size of the AD/DA calibration data
FPGA_DATA_FLASH_SIZE	0x00040000	Memory size of the FPGA Configuration Data

Bits within *Get/Set_DSP_Int_Factor()* and *Get/Set_DSP_Int_Flag()*

```
#define B_FRNET_INT          (0x02)
#define B_FPGA_INT           (0x01)
```

5.1.5 Constants in PMDKfpga.h

Default DDA cycle time = 6250*40ns = 0.25ms

```
#define DEFAULT_DDA_CYCLE_TIME      0.00025
#define DEFAULT_DDA_LENGTH          6250
```

Offset address in SDRAM reserved for FPGA download data

Macro	*Value*	*Description*
FPGA_DATA_SDRAM_OFST	0x00F00000	It is the starting address of FPGA configuration data. The size is 0x00040000 bytes. If a DSP project includes the LoadFPGAData.C, data in file fpgaPMDK.H will be loaded on this memory for configuration the FPGA..

Offset address in DPRAM reserved for interrupt notification

Macro	*Value*	*Description*
DPR_L2H_INT_OFST	0x07FC	When DSP writes a data to this offset address on DPRAM, DPRAM will generate an interrupt signal through the PCI bus to the host (PC).
DPR_H2L_INT_OFST	0x07FE	When the host PC writes a data to this offset address on DPRAM, the DPRAM will generate an interrupt signal to DSP. The interrupt flag will be cleared when DSP read this memory address.

Values for the "FPGA_Src" parameter in *FPGA_Init()*:

Macro	*Value*	*Description*
FPGA_SRC_SDRAM	0	Use data on SDRAM to configure FPGA.
FPGA_SRC_FLASH	1	Use data on FLASH to configure FPGA.

Values for the for "Ch" parameter in *Set/Get_FPGA_Int_Factor() and Get/Clr_FPGA_Int_Flag()* functions:

```
#define AXIS0_INT_CH            (0)
#define AXIS1_INT_CH            (1)
#define AXIS2_INT_CH            (2)
#define AXIS3_INT_CH            (3)
#define AXIS4_INT_CH            (4)
#define AXIS5_INT_CH            (5)
#define MISC_INT_CH             (250)
#define GINP_INT_CH             (253)
#define MAIN_INT_CH             (254)
#define GLB_INT_CH              (255)
```

Bits within GLB_INT_CH

```
#define B_GLB_INT           (0x0001)
```

Bits within MAIN_INT_CH

```
 #define B_MISC_INT             (0x8000)
 #define B_SERVO_INT            (0x4000)
 #define B_AX5_H_INT            (0x2000)
 #define B_AX4_H_INT            (0x1000)
 #define B_AX3_H_INT            (0x0800)
 #define B_AX2_H_INT            (0x0400)
 #define B_AX1_H_INT            (0x0200)
 #define B_AX0_H_INT            (0x0100)
 #define B_DPL_INT              (0x0080)
 #define B_GINP_INT             (0x0040)
 #define B_AX5_L_INT            (0x0020)
 #define B_AX4_L_INT            (0x0010)
 #define B_AX3_L_INT            (0x0008)
 #define B_AX2_L_INT            (0x0004)
 #define B_AX1_L_INT            (0x0002)
 #define B_AX0_L_INT            (0x0001)
```

Bits within GINP_INT_CH

```
 #define B_EMG_INT              (0x8000)
 #define B_MPG_EMG_INT          (0x4000)
//  #define B_GDI11_INT         (0x0800)
 #define B_GDI10_INT            (0x0400)
 #define B_GDI9_INT             (0x0200)
 #define B_GDI8_INT             (0x0100)
 #define B_GDI7_INT             (0x0080)
 #define B_GDI6_INT             (0x0040)
 #define B_GDI5_INT             (0x0020)
 #define B_GDI4_INT             (0x0010)
```

#define B_GDI3_INT (0x0008)
#define B_GDI2_INT (0x0004)
#define B_GDI1_INT (0x0002)
#define B_GDI0_INT (0x0001)

Bits within MISC_INT_CH
#define B_TIM_ERR_INT (0x8000)
#define B_UART_RX_INT (0x0004)

Bits within AXISx_INT_CH
#define B_CMP_INT (0x8000)
#define B_EZ_INT (0x0200)
#define B_IDX_INT (0x0100)
#define B_LTC_INT (0x0080)
#define B_RDY_INT (0x0040)
#define B_INP_INT (0x0020)
#define B_ALM_INT (0x0010)
#define B_SLD_INT (0x0008)
#define B_ORG_INT (0x0004)
#define B_MEL_INT (0x0002)
#define B_PEL_INT (0x0001)

Values for the for "AD6_Src" parameters in *Set_AD6_Src()* function
#define AD6_SRC_AGND (1)
#define AD6_SRC_DA0 (2)
#define AD6_SRC_DA1 (3)
#define AD6_SRC_DA2 (4)
#define AD6_SRC_DA3 (5)
#define AD6_SRC_DA4 (6)
#define AD6_SRC_DA5 (7)
#define AD6_SRC_REF5V (8)

Values for the for "Mode" parameter in the *Set_DDA_Pulse_Mode()*, *Set_ENC_Pulse_Mode()* and *Set_MPG_Pulse_Mode()* functions

Macro	*Value*	*Description*
PLS_MD_OUTDIR	0	Use OUT/DIR mode
PLS_MD_CWCCW	1	Use CW/CCW mode
PLS_MD_1xAB	2	Use A/B phase mode, X1
PLS_MD_2xAB	3	Use A/B phase mode, X2
PLS_MD_4xAB	4	Use A/B phase mode, X4

Values for the for "Ch" parameter in the *Write_Cal_Data()* and *Read_Cal_Data()* functions

```
#define DA0_SPAN_CAL_CH         (1)
#define DA1_SPAN_CAL_CH         (2)
#define DA2_SPAN_CAL_CH         (3)
#define DA3_SPAN_CAL_CH         (4)
#define DA4_SPAN_CAL_CH         (5)
#define DA5_SPAN_CAL_CH         (6)
#define AD_OFST_CAL_CH          (7)
#define AD_SPAN_CAL_CH          (8)
#define DA0_OFST_CAL_CH         (11)
#define DA1_OFST_CAL_CH         (12)
#define DA2_OFST_CAL_CH         (13)
#define DA3_OFST_CAL_CH         (14)
#define DA4_OFST_CAL_CH         (15)
#define DA5_OFST_CAL_CH         (16)
#define R5V_OFST_CAL_CH         (17)
```

Values for the for "DA_Clr_Src" parameter in the *Set/Get_DA_Clear_Ctl()* and "DDA_Clr_Src" parameter in the *Set/Get_DDA_Clear_Ctl()* functions

```
#define CLR_SRC_EMG             (0)
#define CLR_SRC_PEL             (1)
#define CLR_SRC_MEL             (2)
```

Bits within “Value” of *Set_Axis_IO () and Get_Axis_IO ()*

```
#define B_CMP_SIG               (0x8000)
#define B_ALMRST_SIG            (0x4000)
#define B_ERC_SIG               (0x2000)
#define B_SVON_SIG              (0x1000)
#define B_EA_SIG                (0x0800)
#define B_EB_SIG                (0x0400)
#define B_EZ_SIG                (0x0200)
#define B_IDX_SIG               (0x0100)
#define B_LTC_SIG               (0x0080)
#define B_RDY_SIG               (0x0040)
#define B_INP_SIG               (0x0020)
#define B_ALM_SIG               (0x0010)
#define B_SLD_SIG               (0x0008)
#define B_ORG_SIG               (0x0004)
#define B_MEL_SIG               (0x0002)
#define B_PEL_SIG               (0x0001)
```

Bits within “Value” of *Get_DI()*

```
#define B_EMG_SIG               (0x8000)
```

```
#define B_MPG_EMG_SIG          (0x4000)
#define B_GDI11_SIG            (0x0800)
#define B_GDI10_SIG            (0x0400)
#define B_GDI9_SIG             (0x0200)
#define B_GDI8_SIG             (0x0100)
#define B_GDI7_SIG             (0x0080)
#define B_GDI6_SIG             (0x0040)
#define B_GDI5_SIG             (0x0020)
#define B_GDI4_SIG             (0x0010)
#define B_GDI3_SIG             (0x0008)
#define B_GDI2_SIG             (0x0004)
#define B_GDI1_SIG             (0x0002)
#define B_GDI0_SIG             (0x0001)
```

Bits within "Value" of *Set_DO() and Get_DO()*

```
#define B_GDO2_SIG             (0x0004)
#define B_GDO1_SIG             (0x0002)
#define B_GDO0_SIG             (0x0001)
```

Values for the for "Ch" parameter in the *Set/Get_Axio_IO_Filter()* functions:

```
#define EA_FLT_CH              (10)
#define EB_FLT_CH              (9)
#define EZ_FLT_CH              (8)
#define LTC_FLT_CH             (7)
#define RDY_FLT_CH             (6)
#define INP_FLT_CH             (5)
#define ALM_FLT_CH             (4)
#define SLD_FLT_CH             (3)
#define ORG_FLT_CH             (2)
#define MEL_FLT_CH             (1)
#define PEL_FLT_CH             (0)
```

Values for the for "Baud" parameter in the *Set/Get_UART_Baud()* functions:

```
#define BAUD_38400             (3)
#define BAUD_19200             (2)
#define BAUD_9600              (1)
#define BAUD_4800              (0)
```

Bits within UART_Sts

```
#define B_TX_RDY               (0x10)
#define B_OVERRUN              (0x08)
#define B_FRM_ERR              (0x04)
#define B_PRT_ERR              (0x02)
#define B_RX_RDY               (0x01)
```

Values for the for "Ch" parameter in the *Set/Get_DI_Filter()* functions:

#define EMG_FLT_CH	(15)
#define MPG_EMG_FLT_CH	(14)
#define MPG_A_FLT_CH	(13)
#define MPG_B_FLT_CH	(12)
#define GDI11_FLT_CH	(11)
#define GDI10_FLT_CH	(10)
#define GDI9_FLT_CH	(9)
#define GDI8_FLT_CH	(8)
#define GDI7_FLT_CH	(7)
#define GDI6_FLT_CH	(6)
#define GDI5_FLT_CH	(5)
#define GDI4_FLT_CH	(4)
#define GDI3_FLT_CH	(3)
#define GDI2_FLT_CH	(2)
#define GDI1_FLT_CH	(1)
#define GDI0_FLT_CH	(0)