PCI-8158

고밀도 & 진보된
8 축 서보 / 스테퍼
모션 제어 카드

매뉴얼 수정 : 2.00
수정 일자 : 2006 년 8 월 5 일
2012 년 1 월 31 일자 번역
부품 번호 : 50-11139-2B00

Recycled Paper

Advance Technologies; Automate the World.

ADLINK 의 서비스가 필요하신 경우

고객 만족은 ADLINK Technology Inc 의 최우선 사항입니다 . 귀하가 어떠한 서비스나 도움이 필요하신 경우 언제든지 우리를 찾아 주십시오 .

ADLINK TECHNOLOGY INC.

웹사이트 : http://www.adlinktech.com

판매 & 서비스 : Service@adlinktech.com

TEL: + 886-2-82265877

FAX: + 886-2-82265717

주소 : 9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan

신속하고 정확한 서비스를 위해 이 서비스 폼을 이메일 또는 팩스로 보내주시기 바랍니다 .

회사 정보	
회사 / 단체	
담당자	
E-mail 주소	
주소	
국가	
TEL	FAX:
웹사이트	
제품 정보	
제품 모델	
환경	OS: M/B: CPU: Chipset: BIOS:

문제에 대한 자세한 설명을 써 주시기 바랍니다 :

의도적인 공백 페이지

목 차

List of Tables

List of Figures

1 소 개

PCI-8158 는 PCI 방식의 진보적인 모션 제어 카드로서 8 축 모션 제어가 가능하며 스테퍼와 서보모터를 가동하기 위해 최대 6.55MHz 의 고주파 펄스를 출력합니다. 모션 제어기로서 8 축 선형, 원호 보간과 연속 속도를 위한 연속 보간을 제공하며 한 축에 대해서는 움직이는 동안 위치와 속도를 바꿀 수 있습니다.

또한, 하나의 시스템 내에서 여러 장의 카드를 사용할 수 있습니다. 모든 8 개 축의 증가형 인코더 인터페이스는 부정확한 기계적 전달에 의해 발생한 위치 에러값을 조정하는 기능을 제공합니다.

PCI-8158 는 이전과 다른 새로운 설계로 캐리어 보드는 8 축 펄스열 출력 제어 채널을 내장합니다. 다른 기능으로는 고속의 트리거 또는 분산형 I/O 제어와 사용자는 편의에 따라 도터보드를 추가할 수 있습니다. 또한, 보드는 위치 비교 함수를 가지고 있습니다. 보드는 비교 출력 기능을 가지고 모션 제어기는 라인 스캔 적용을 위하여 고속 트리거 펄스를 출력할 수 있어야 하며 고 해상도의 이미지를 얻을 수 있어야 합니다 . 이러한 경우 사용자는 PCI-8158 의 기능을 확장하는 DB-8150 를 선택할 수 있습니다. 또한, 모션 제어기로서 설계뿐 아니라 센서와 엑츄에이터 또한 자동화의 중요한 요소입니다 . 일반적으로 I/O 는 제어장치 내에 센서와 엑츄에이터의 통합을 필요로 하는데 ADLINK 는 이러한 장치들을 연결하는 또 다른 방법을 제공합니다 (분산형 I/O). 도터보드는 분산형 I/O 와 PCI-8158 을 효과적으로 연결하는 데 사용 될 수 있습니다. 이러한 구성은 배선과 물리적 제어기의 크기에 관한 수고를 덜해주며 또한 비용 효율적입니다.

그림 1-1 은 PCI-8158 의 기능적인 블록 다이어그램을 보여줍니다. 모션제어 기능은 S- 커브의 가속과 감속, 두 축 사이에서의 선형, 원호 보간법 또한 연속적 모션 포지셔닝, 13 가지 홈 리턴 모드를 포함합니다. 이 모든 기능과 복잡한 계산은 ASIC 에 의해 내부적으로 실행되어 CPU 를 내부적 충돌로부터 보호합니다.

또한, PCI-8158 은 세 가지 사용자 친화적 기능을 제공합니다.

1. 카드 인덱스 설정 :

   PCI-8158 는 DIP 스위치 설정으로 카드 인덱스를 지정할 수 있습니다. 값은 0 부터 15 까지 이며 전체 제어 시스템이 매우 큰 경우 제조 업체들이 카드 인덱스를 인식하는 데 있어 유용합니다.

2. 비상 입력

비상 입력 핀은 사용자가 어떠한 긴급 상황이 발생하였을 경우 유선 비상 버튼을 통해 펄스 출력의 전송을 중지할 수 있습니다 .

3. 소프트웨어의 보안 보호

보안 보호 설계로서 사용자는 16 비트 변수를 EEPROM 에 설정할 수 있습니다 . 귀하의 인터페이스 프로그램은 소프트웨어와 하드웨어의 도용을 방지하여 보안을 강화하는데 EEPROM 을 사용할 수 있습니다 .

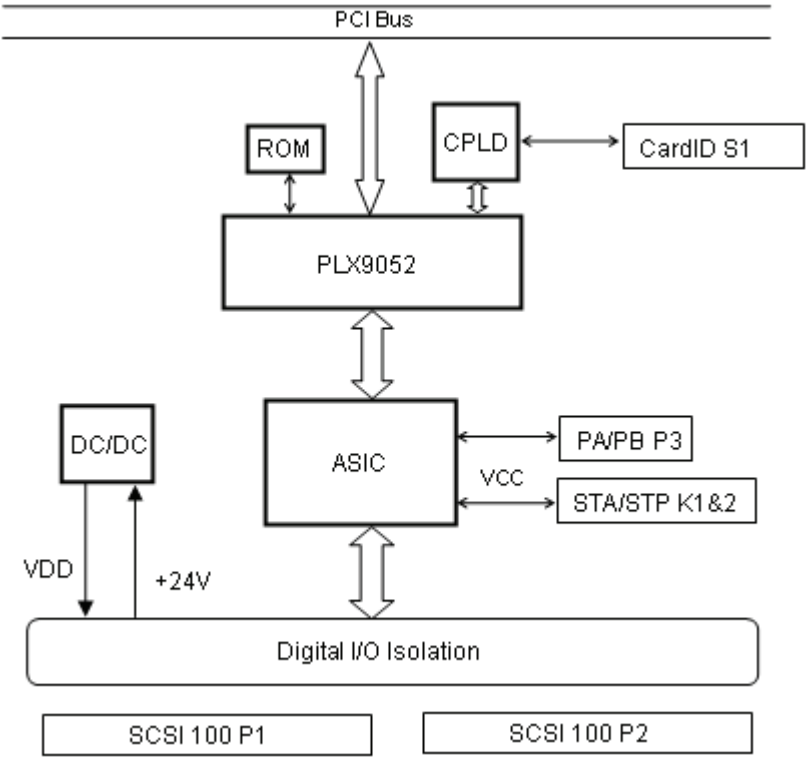


Figure 1-1: PCI-8158 의 블록 다이어그램

MotionCreatorPro 는 윈도우 기반의 응용 소프트웨어 패키지이며 프로젝트의 디자인 단계 중 , 모션 제어 시스템의 결함을 없애기 위하여 PCI-8158 과 함께 포함되었습니다 . PCI-8158 의 모든 축과 입 / 출력 신호 상태를 화면에 나타냅니다 .

또한 , 윈도우 프로그래밍을 위한 C++ 와 베이직 라이브러리가 포함되어 있으며 함수의 운영을 위한 샘플 프로그램이 제공됩니다 .

그림 1-2 는 응용 프로그램을 개발하는 과정에서 추천하는 프로세스의 순서도를 설명합니다 . 각 단계의 세부사항에 대해서는 각 장을 참고 하십시오 .

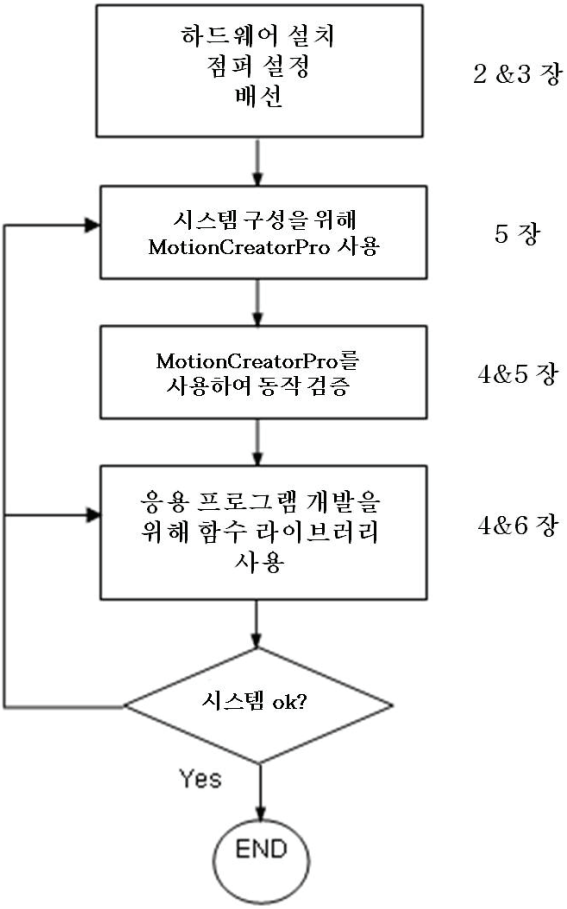


Figure 1-2: 응용 프로그램 개발을 위한 흐름도

1.1 특 징

다음 목록들은 PCI-8158 모션 제어 시스템의 주요한 특징들을 요약합니다 .

- 31 비트 PCI bus Plug and Play (보편적)
- 서보모터 또는 스테핑 제어를 위한 8 채널의 펄스열
- 최대 6.55 MPPS 주파수 출력
- 펄스 출력 옵션 : OUT/DIR, CW/CCW
- 프로그램상의 모든 모드에서 가속 / 감속 시간 설정 가능
- 모든 모드에 대해서 Trapezoidal 과 S- 커브 속도 프로파일을 제공
- 2-4 축 선형 보간
- 2 축 원형 보간
- 모션에 따른 윤곽에 대한 연속 보간
- 동작 중 속도 및 위치 변경 (Change position and speed on the fly)
- 자동 검색과 함께 13 가지의 귀환 모드 제공
- 하드웨어적인 오류 보정 및 진동 감소
- 각 축에 대한 2 가지 소프트웨어
- 증가형 인코더 피드백에 대한 28 비트 up/down 카운터
- 모든 축에 홈 스위치 , 인덱스 신호 (EZ), 긍정적 , 부정적인 end-limit 스위치
- 8 축 고속 위치 latch 입력
- 8 축 위치 비교 출력과 트리거 출력 (고속 트리거 출력을 위해서는 사용자는 DB-8150 을 추가 구매 해야함)
- 모든 디지털 입 / 출력 신호에 대해서 2500Vrms 까지 절연
- 프로그래밍이 가능한 인터럽트 소스제공
- 동시 시작 / 정지
- 수동 펄스 입력 인터페이스
- 카드 인덱스 선택 가능
- EERPOM 의 보안 보호
- 유선 전용 긴급 입력 핀

- 한 시스템 내에서 최대 12 개의 PCI-8158 카드 지원
- 소형 PCB 설계
- 마이크로소프트 윈도우 기반의 응용 소프트웨어와 MotionCreatorPro 포함
- PCI-8158 라이브러리와 윈도우즈 2000/XP 유틸리티 지원 .

1.2 상세 요약

- 사용 가능한 모터 :
  - 스테핑 모터
  - 펄스열에 의해 동작하는 AC 또는 DC 서보 모터
- 성능 :
  - 제어 가능한 축수 : 8 축
  - 최대 출력 펄스 : 6.55MPPS, 선형 , trapezoidal 또는 S-커브 속도 프로파일
  - 내부 클록 : 19.66 MHz
  - 28 비트 up/down 카운터 범위 : 0~268,435,455 또는 -134,217,728 ~ + 134,217,727
  - 위치 펄스 설정 범위 (28 비트): -134,217,728 ~ + 134,217,728
  - 펄스 레이트 설정 범위 : (펄스 비 = 1:65535):

    0.1PPS ~ 6553.5 PPS (승수 = 0.1)

    1PPS ~ 65535PPS (승수 = 1)

100PPS~6553500PPS(승수 = 100)

- ▶ 입 / 출력 신호 :
  - ▷ 각 축에 대한 입 / 출력
  - ▷ 모든 입 / 출력 신호는 2500Vrms 까지 지원이 되는 절연 체로 구성
  - ▷ 명령 펄스 출력 핀 : OUT, DIR
  - ▷ 증가형 인코더 신호 입력 핀 : EA, EB
  - ▷ 인코더 인덱스 신호 입력 핀 : EZ
  - ▷ 물리적 limit/home 신호 입력 핀 : ±EL, ORG
  - ▷ 복합 핀 : DI / LTC(Latch) / SD(Slow-down) / PCS(위치 변경 신호) / CLR(클리어) / EMG(긴급입력)
  - ▷ 서보모터 인터페이스 입 / 출력 핀 : INP, ALM, ERC
  - ▷ 범용 디지털 출력 핀 : SVON, DO
  - ▷ 범용 디지털 입력 핀 : RDY, GDI
  - ▷ 펄스 신호 입력 핀 : PA, PB (절연)
  - ▷ 동시 시작 / 멈춤 신호 : STA, STP
- ▶ 기본 사양
  - ▷ 커넥터 : 68 핀 SCSI 타입 커넥터
  - ▷ 동작 온도 : 0°C – 50°C
  - ▷ 저장 온도 : -20°C – 80°C
  - ▷ 습도 : 5 – 85% (일반적인 상태)
- ▶ 전력 소비
  - ▷ 슬롯 전원 공급 (입력): + 5V DC ±5%, 최대 900mA
  - ▷ 외부 전원 공급 (입력): + 24V DC ±5%, 최대 500mA
  - ▷ 외부 전원 공급 (출력): + 5V DC ±5%, 최대 500mA
- ▶ PCI-8158 치수 (PCB 크기): 185mm(L) X 100mm(W)

1.3 소프트웨어 지원

1.3.1 프로그램 라이브러리

PCI-8158 은 윈도우즈 2000/XP DLL 을 제공하며 이 함수 라이브러리는 보드와 함께 제공됩니다

1.3.2 MotionCreatorPro

윈도우 기반의 유틸리티로서 카드 , 모터 , 시스템을 설정하는데 이용됩니다 . 또한 , 하드웨어와 소프트웨어 문제 결함을 없애는 것을 도울 수 있고 I/O 논리 파라미터의 세팅을 사용자 프로그램에서 불러와 사용할 수 있습니다 . 또한 , 이 제품은 카드와 함께 제공됩니다 .

자세한 내용은 5 장을 참고하시기 바랍니다 .

1.4 사용 가능한 터미널 보드

ADLINK 가 제공하는 서보 & 스테퍼는 사용자의 보다 쉬운 연결을 위해 터미널 보드를 사용합니다 . ADLINK 는 서보 & 스테퍼의 쉬운 연결을 위한 터미널 보드를 제공하며 스테퍼의 경우 Pin-to-Pin 터미널 보드인 DIN-100S 를 제공하고 서보 사용자의 경우 ADLINK 는 DIN- 814M, DIN-814M-J3A, DIN-814Y 과 DIN-814P-A4 를 제안합니다 . 적합한 서보는 다음과 같습니다 .

Mitsubishi J2 Super	DIN-814M
Mitsubishi J3A	DIN-814M-J3A
Yaskawa Sigma II	DIN-814Y
Panasonic MINAS A4	DIN-814P-A4

Table 1-1: 사용 가능한 터미널 보드

2 PCI-8158 설치

이 장에서는 PCI-8158 을 설치하는 방법을 설명하고 있습니다 . 그 순서는 다음과 같습니다 .

- 내용물 확인 (2.1 장)
- PCB 확인 (2.2 장)
- 하드웨어 설치 (2.3 장)
- 소프트웨어 드라이버 설치 (2.4 장)
- 입 / 출력 신호결선 (3 장) 과 동작 원리의 이해 (4 장)
- 커넥터 핀 할당과 결선의 이해 (나머지 장)

2.1 내용물 확인

본 사용자 가이드를 포함하여 다음의 아이템을 포함하는 패키지 :

- PCI-8158: 진보된 8 축 서보 / 스테핑 제어 카드
- ADLINK All-in-one CD (제품드라이버 CD)

터미널 보드는 선택 사항이며 PCI-8158 패키지에 포함되지 않습니다 .

만약 이 아이템 중 하나라도 없거나 손상된 경우에는 귀하께서 제품을 구매하셨던 판매자에게 문의하시기 바랍니다. 배송자료와 향후 제품을 보관하거나 선적하는데 사용될 상자는 보관해 주십시오 .

2.2 PCI-8158 외형도

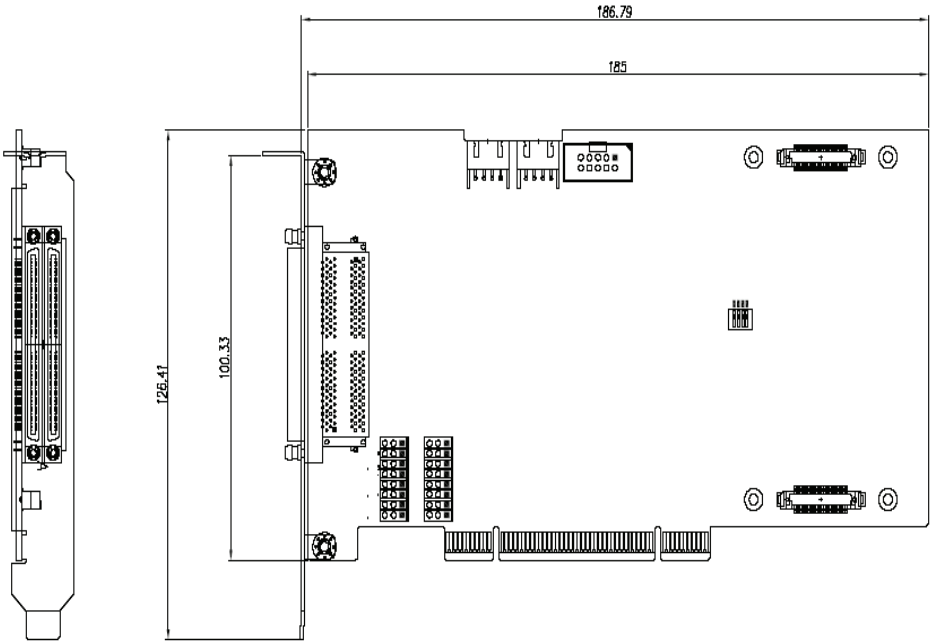


Figure 2-1: PCI-8158 의 PCB Layout

- P1 / P2: 입 / 출력 신호 커넥터 (100- 핀)
- K1 / K2: 동시 시작 / 정지 커넥터
- P3: 동기 구동
- S1: DIP 카드 인덱스 선택 (0-15)
- J1-J16: 펄스 출력 형태를 선택하기 위한 점퍼 (라인 드라이버 / 오픈 컬렉터)

2.3 PCI-8158 하드웨어 설치

2.3.1 하드웨어 구성

PCI-8158 은 Plug & Play 를 완벽하게 지원합니다 . 그러므로 메모리 할당 (입 / 출력 포트 설정) 과 PCI 카드의 IRQ 채널은 시스

템 BIOS 에 의해 자동 할당 됩니다 . 주소 할당은 시스템 내 모든 PCI 카드에 대하여 보드마다 지정됩니다 .

2.3.2 PCI 슬롯 선택

귀하의 PC는 PCI 와 ISA 슬롯을 모두 가지고 있을 것입니다. PA/AT 슬롯에 PCI 카드를 강제로 설치하지 마십시오 . PCI-8158 은 PCI 슬롯 어디에 설치하셔도 무관합니다 .

2.3.3 설치 순서

1. 이 사용자 매뉴얼을 읽고 점퍼를 적용환경에 맞게 설정 합니다 .
2. 컴퓨터와 기타 모든 장치 (프린터 , 모뎀 , 모니터 등) 를 끄고 컴퓨터의 커버를 여십시오 .
3. 32- 비트 PCI 확장 슬롯을 선택하십시오 . PCI 슬롯은 ISA 또는 EISA 슬롯보다 짧고 흰색 또는 아이보리 색을 가지고 있습니다 .
4. PCI-8158 작업을 하기 전에 컴퓨터의 금속 케이스를 접촉하여 귀하의 몸의 정전기를 제거하시기 바랍니다 . 카드의 양 끝을 잡고 부품을 만지지 마십시오 .
5. 보드를 귀하가 선택한 PCI 슬롯에 꼽으십시오 .
6. 카드의 안정을 위해 슬롯 고정 나사로 카드를 배면 패널에 고정합니다 .

2.3.4 문제 해결

만약 시스템이 부팅되지 않거나 오작동 현상이 발생한다면 대부분 원인은 인터럽트 충돌 때문입니다 (부적절한 ISA 선정). 문제가 발생할 경우 사용하고 있는 시스템에 관련된 BIOS 문서를 찾아보십시오 .

제어판의 장치관리자에 카드가 없다면 BIOS 의 PCI 설정을 체크하시고 또는 다른 PCI 슬롯에 설치해 보십시오 .

2.4 소프트웨어 드라이버 설치

1. ADLINK All-In-One CD 를 드라이버에 삽입한 후 Driver Installation -> Motion Control -> PCI-8158 을 선택하십시오
2. 인스톨 화면의 설치 순서에 따라 설치하십시오 .
3. 설치가 끝난 후 컴퓨터를 재부팅 하십시오 .

참고 : 최신 소프트웨어가 필요하시다면 ADLINK 웹 사이트에서 다운로드 하실 수 있습니다 .

2.5 P1/P2 핀 할당 : 메인 커넥터

P1/P2 는 모션 제어 입 / 출력 신호를 위한 메인 커넥터입니다 .

No.	명칭	I/O	기 능	No.	명칭	I/O	기 능
1	VDD	O	+5V 전원 공급 출력	51	VDD	O	+5V 전원 공급 출력
2	EXGND	-	Ext. 전원 접지	52	EXGND	-	Ext. 전원 접지
3	OUT0+	O	펄스 신호 (+)	53	OUT2+	O	펄스 신호 (+)
4	OUT0-	O	펄스 신호 (-)	54	OUT2-	O	펄스 신호 (-)
5	DIR0+	O	Dir. 신호 (+)	55	DIR2+	O	Dir. 신호 (+)
6	DIR0-	O	Dir. 신호 (-)	56	DIR2-	O	Dir. 신호 (-)
7	SVON0	O	서보 On/Off	57	SVON2	O	서보 On/Off
8	ERC0	O	Dev. ctr, clr. 신호	58	ERC2	O	Dev. ctr, clr. 신호
9	ALM0	I	경보 신호	59	ALM2	I	경보 신호
10	INP0	I	In-position 신호	60	INP2	I	In-position 신호
11	RDY0	I	다용도 입력 신호	61	RDY2	I	다용도 입력 신호
12	EXGND		Ext. 전원 접지	62	EXGND		Ext. 전원 접지
13	EA0+	I	인코더 A- 위상 (+)	63	EA2+	I	인코더 A- 위상 (+)
14	EA0-	I	인코더 A- 위상 (-)	64	EA2-	I	인코더 A- 위상 (-)
15	EB0+	I	인코더 B- 위상 (+)	65	EB2+	I	인코더 B- 위상 (+)
16	EB0-	I	인코더 B- 위상 (-)	66	EB2-	I	인코더 B- 위상 (-)
17	EZ0+	I	인코더 Z- 위상 (+)	67	EZ2+	I	인코더 Z- 위상 (+)
18	EZ0-	I	인코더 Z- 위상 (-)	68	EZ2-	I	인코더 Z- 위상 (-)
19	VDD	O	+5V 전원 공급 출력	69	VDD	O	+5V 전원 공급 출력
20	EXGND	-	Ext. 전원 접지	70	EXGND	-	Ext. 전원 접지
21	OUT1+	O	펄스 신호 (+)	71	OUT3+	O	펄스 신호 (+)
22	OUT1-	O	펄스 신호 (-)	72	OUT3-	O	펄스 신호 (-)
23	DIR1+	O	Dir. 신호 (+)	73	DIR3+	O	Dir. 신호 (+)
24	DIR1-	O	Dir. 신호 (-)	74	DIR3-	O	Dir. 신호 (-)
25	SVON1	O	서보 On/Off	75	SVON3	O	서보 On/Off
26	ERC1	O	Dev. ctr, clr. 신호	76	ERC3	O	Dev. ctr, clr. 신호
27	ALM1	I	경보 신호	77	ALM3	I	경보 신호
28	INP1	I	In-position 신호	78	INP3	I	In-position 신호
29	RDY1	I	다목적 입력 신호	79	RDY3	I	다목적 입력 신호
30	EXGND		Ext. 전원 접지	80	EXGND		Ext. 전원 접지
31	EA1+	I	인코더 A- 위상 (+)	81	EA3+	I	인코더 A- 위상 (+)
32	EA1-	I	인코더 A- 위상 (-)	82	EA3-	I	인코더 A- 위상 (-)
33	EB1+	I	인코더 B- 위상 (+)	83	EB3+	I	인코더 B- 위상 (+)
34	EB1-	I	인코더 B- 위상 (-)	84	EB3-	I	인코더 B- 위상 (-)

Table 2-1: P1/P2 핀 할당

No.	명칭	I/O	기 능	No.	명칭	I/O	기 능
35	EZ1+	I	인코더 Z- 위상 (+)	85	EZ3+	I	인코더 Z- 위상 (+)
36	EZ1-	I	인코더 Z- 위상 (-)	86	EZ3-	I	인코더 Z- 위상 (-)
37	PEL0	I	End limit 신호 (+)	87	PEL2	I	End limit 신호 (+)
38	MEL0	I	End limit 신호 (-)	88	MEL2	I	End limit 신호 (-)
39	GDI0	I	DI/LTC/PCS/SD/CLR0	89	GDI2	I	DI/LTC/PCS/SD/CLR2
40	DO0	O	다용도 출력 0	90	DO2	O	다용도 출력 2
41	ORG0	I	원점 신호	91	ORG2	I	원점 신호
42	EXGND		Ext. 전원 접지	92	EXGND		Ext. 전원 접지
43	PEL1	I	End limit 신호 (+)	93	PEL3	I	End limit 신호 (+)
44	MEL1	I	End limit 신호 (-)	94	MEL3	I	End limit 신호 (-)
45	GDI1	I	DI/LTC/PCS/SD/CLR1/ EMG	95	GDI3	I	DI/LTC/PCS/SD/CLR3
46	DO1	O	다용도 출력 1	96	DO3	O	다용도 출력 3
47	ORG1	I	원점 신호	97	ORG3	I	원점 신호
48	EXGND	-	Ext. 전원 접지	98	EXGND	-	Ext. 전원 접지
49	EXGND	-	Ext. 전원 접지	99	E_24V	-	Isolation 전원 입력 , + 24V
50	EXGND	-	Ext. 전원 접지	100	E_24V	-	Isolation 전원 입력 , + 24V

Table 2-1: P1/P2 핀 할당

▶ P1은 0축부터 3축까지 제어하고 P2는 4축부터 7축까지 제어합니다 .

2.6 K1/K2 핀 할당 : 동시 시작 / 정지

K1 과 K2 는 여러 축과 여러 카드에 대한 동시 시작 / 정지 신호를 위한 부분입니다 .

No.	명칭	기 능
1	+ 5V	PCI 버스 전원 출력 (VCC)
2	STA	동시 시작 신호 입 / 출력
3	STP	동시 정지 신호 입 / 출력
4	GND	PCI 버스 전원 접지

Table 2-2: K1/K2 핀 할당

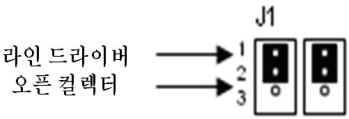
참고 : + 5V 와 GND 핀은 PCI 버스 전원에 의해 제공됩니다 .

2.7 J1-J16 펄스 출력 점퍼 설정

J1-J16 은 펄스 출력 신호의 신호 형태를 결정하기 위해 사용됩니다 (DIR, OUT). 출력 신호 형태는 차동 라인 드라이버 또는 오픈 컬렉터 출력으로 선택 가능합니다. 자세한 점퍼 설정은 3.1 을 참고하십시오. 기본적인 설정은 차동 라인 드라이버 모드입니다. 맵핑테이블은 다음과 같습니다.

JP1 & JP2	축 0	JP9 & JP10	축 4
JP3 & JP4	축 1	JP11 & JP12	축 5
JP5 & JP6	축 2	JP13 & JP14	축 6
JP7 & JP8	축 3	JP15 & JP16	축 7

Table 2-3: J1-J16 점퍼 설정



2.8 S1 카드 인덱스 스위치 설정

S1 스위치는 카드 인덱스를 설정하는데 사용됩니다. 예를 들어 첫 번째 스위치가 ON 이고 다른 나머지가 모두 OFF 이면 카드의 인덱스는 1 입니다. 이 값은 0 부터 15 까지이며 자세한 내용은 다음의 표를 참고하십시오.

카드 ID	스위치 설정 (ON=1)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table 2-4: S1 스위치 설정

2.9 P3 수동 펄스

P3 의 신호는 수동 펄스 입력을 위한 부분입니다 .

No.	명칭	기능 (축)
1	VDD	절연된 전원 +5V
2	PA+	펄스 A+ 위상 신호 입력
3	PA-	펄스 A- 위상 신호 입력
4	PB+	펄스 B+ 위상 신호 입력
5	PB-	펄스 B- 위상 신호 입력
6	EXGND	외부 접지
7	N/A	사용할 수 없음
8	N/A	사용할 수 없음
9	N/A	사용할 수 없음

Table 2-5: P3 수동 펄스

참고 : +5V 와 GND 핀은 PCI- 버스 전원에 의해 직접적으로 제공 됩니다 . 그러므로 이 신호는 분리되지 않습니다 .

의도적인 공백 페이지

3 PCI-8158 신호 연결

이 장에서는 모든 입/출력 신호 연결에 대하여 설명합니다. PCI-8158 과 모터 드라이버를 연결하기 전에 이 장의 내용을 참고하시기 바랍니다.

이 장은 아래의 내용을 포함합니다.

3.1 펄스 출력 신호 OUT, DIR

PCI-8158 에는 8 축 펄스 출력 신호가 있습니다. 모든 각 축에 대해 두 쌍의 다른OUT 와 DIR 신호는 방향을 지시하고 펄스 열을 전송하기 위해 사용됩니다 . OUT 와 DIR 신호는 CW 와 CCW 신호로 프로그램될 수 있습니다. 4.1.1 에서는 보다 자세한 OUT 와 DIR 신호의 논리적 특성에 대해 다루고 있습니다 . 여기에서는 OUT 와 DIR 신호의 전기적 특성에 대해 자세히 설명합니다 . 예를 들어 OUT0 신호는 OUT0+ 와 OUT0- 로 구성되어 있습니다 . 다음의 표는 P1 의 모든 펄스 출력 신호를 보여줍니다 .

P1 핀번호	신호 명칭	설 명	축 #
3	OUT0+	펄스 신호 (+)	0
4	OUT0-	펄스 신호 (-)	0
5	DIR0+	방향 신호 (+)	0
6	DIR0-	방향 신호 (-)	0
21	OUT1+	펄스 신호 (+)	1
22	OUT1-	펄스 신호 (-)	1
23	DIR1+	방향 신호 (+)	1
24	DIR1-	방향 신호 (-)	1
53	OUT2+	펄스 신호 (+)	2
54	OUT2-	펄스 신호 (-)	2
55	DIR2+	방향 신호 (+)	2
56	DIR2-	방향 신호 (-)	2
71	OUT3+	펄스 신호 (+)	3
72	OUT3-	펄스 신호 (-)	3
73	DIR3+	방향 신호 (+)	3
74	DIR3-	방향 신호 (-)	3

Table 3-1: 펄스 출력 신호 OUT (P1)

P2 핀번호	신호 명칭	설 명	축 #
3	OUT4+	펄스 신호 (+)	4
4	OUT4-	펄스 신호 (-)	4
5	DIR4+	방향 신호 (+)	4
6	DIR4-	방향 신호 (-)	4
21	OUT5+	펄스 신호 (+)	5
22	OUT5-	펄스 신호 (-)	5
23	DIR5+	방향 신호 (+)	5
24	DIR5-	방향 신호 (-)	5
53	OUT6+	펄스 신호 (+)	6
54	OUT6-	펄스 신호 (-)	6
55	DIR6+	방향 신호 (+)	6
56	DIR6-	방향 신호 (-)	6
71	OUT7+	펄스 신호 (+)	7
72	OUT7-	펄스 신호 (-)	7
73	DIR7+	방향 신호 (+)	7
74	DIR7-	방향 신호 (-)	7

Table 3-2: 펄스 출력 신호 OUT (P2)

OUT 또는 DIR 신호의 출력은 차동 라인 드라이버 또는 오픈 컬렉터 출력으로 점퍼 설정에 의해 구성될 수 있습니다. 사용자는 출력 모드를 점퍼 J1 부터 J16 의 1-2 또는 2, 3 에 의해 설정할 수 있습니다.

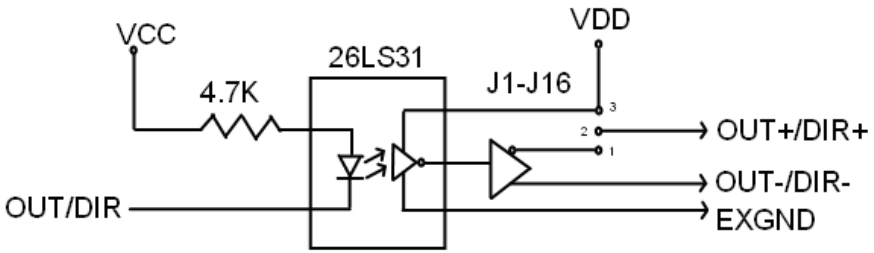
출력 신호	차동 라인 드라이버 출력에 대한 이하의 1,2 간 close breaks:	오픈 컬렉터 드라이버 출력에 대한 이하의 2,3 간 close breaks:
OUT0+	J1	J1
DIR0+	J9	J9
OUT1+	J2	J2
DIR1+	J10	J10
OUT2+	J3	J3
DIR2+	J11	J11
OUT3+	J4	J4
DIR3+	J12	J12

출력 신호	차동 라인 드라이버 출력에 대한 이하의 1,2 간 close breaks:	오픈 컬렉터 드라이버 출력에 대한 이하의 2,3 간 close breaks:
OUT4+	J5	J5
DIR4+	J13	J13
OUT5+	J6	J6
DIR5+	J14	J14
OUT6+	J7	J7
DIR6+	J15	J15
OUT7+	J8	J8
DIR7+	J16	J16

Table 3-3: 출력 신호

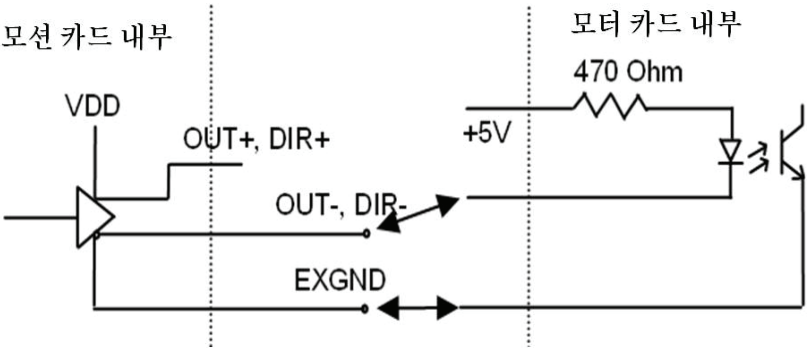
OUT 와 DIR 의 기본 설정은 차동 라인 드라이버로 설정되어 있습니다 .

다음의 결선도는 2 개의 축의 OUT 와 DIR 신호에 대한 그림입니다 .



참고 : 만약 펄스 출력이 오픈 컬렉터 출력 모드로 설정되어 있다면 OUT- 와 DIR- 은 OUT 와 DIR 신호를 전송하기 위하여 사용됩니다 . OUT- 와 DIR- 핀의 전류싱크가 20mA 를 초과하지 않도록 주의하십시오 . 기본 설정은 1-2 쇼트입니다 .

사용 권장 : 2-3 점퍼를 쇼트 시키고 OUT-/DIR- 를 드라이버의 470 옴 펄스 입력 인터페이스 COM 에 연결하십시오 . 다음의 그림을 보고 OUT-/DIR- 를 드라이버의 OUT/DIR 에 연결되도록 선택하십시오



주의 : 싱크를 20mA 또는 26LS31 을 초과하지 않도록 주의하십시오 . 이는 제품에 손상을 줄 수 있습니다 !

3.2 인코더 피드백 신호 EA, EB 와 EZ

인코더 피드백 신호는 EA, EB 와 EZ 를 포함합니다 . 모든 축은 위상 -A, 위상 -B 와 인덱스 (EZ) 입력의 세 가지 차동 회로에 대한 6 핀으로 되어 있습니다 . EA 와 EB 는 위치 계산을 위해 사용되고 EZ 는 영점 지표로 사용됩니다 . 아래의 표는 관련된 신호 명칭 , 핀 번호 , 축 번호를 보여줍니다 .

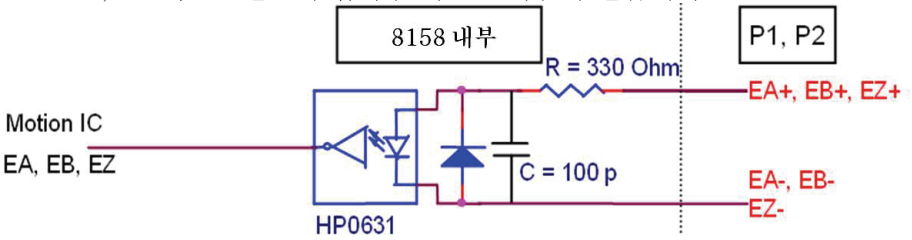
P1 핀번호	신호 명칭	축 #	P1 핀번호	신호 명칭	축 #
13	EA0+	0	14	EA0-	0
15	EB0+	0	16	EB0-	0
31	EA1+	1	32	EA1-	1
33	EB1+	1	34	EB1-	1
63	EA2+	2	64	EA2-	2
65	EB2+	2	66	EB2-	2
81	EA3+	3	82	EA3-	3
83	EB3+	3	84	EB3-	3

P2 핀번호	신호 명칭	축 #	P2 핀번호	신호 명칭	축 #
13	EA4+	4	14	EA4-	4
15	EB4+	4	16	EB4-	4
31	EA5+	5	32	EA5-	5
33	EB5+	5	34	EB5-	5
63	EA6+	6	64	EA6-	6
65	EB6+	6	66	EB6-	6
81	EA7+	7	82	EA7-	7
83	EB7+	7	84	EB7-	7

P1 핀번호	신호 명칭	축 #	P1 핀번호	신호 명칭	축 #
17	EZ0+	0	18	EZ0-	0
35	EZ1+	1	36	EZ1-	1
67	EZ2+	2	68	EZ2-	2
85	EZ3+	3	86	EZ3-	3

P2 핀번호	신호 명칭	축 #	P2 핀번호	신호 명칭	축 #
17	EZ4+	4	18	EZ4-	4
35	EZ5+	5	36	EZ5-	5
67	EZ6+	6	68	EZ6-	6
85	EZ7+	7	86	EZ7-	7

EA, EB 와 EZ 신호의 입력부 회로는 다음과 같습니다 .

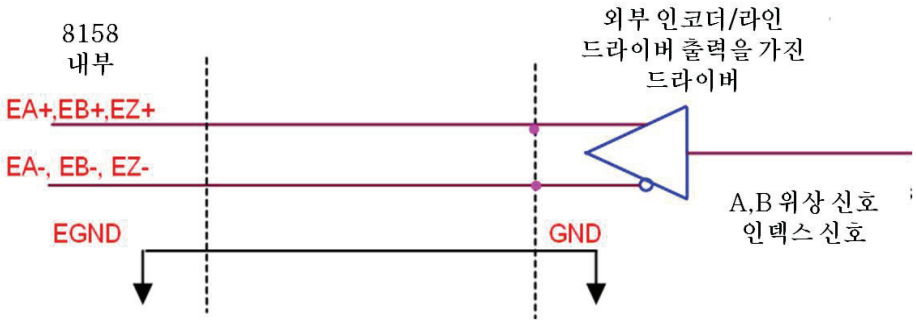


모든 인코더 입력 신호 (EA+ , EA-), (EB+ , EB-) 와 (EZ+ , EZ-) 의 각 차동회로를 지나는 전압은 최소 3.5V 가 되어야 합니다 . 따라서 인코더 피드백 또는 모터 드라이버 피드백이 연결되었을 때 그 구동 용량에 의해 주의하셔야 합니다 . 차동회로는 EA, EB 와 EZ 를 디지털 신호로 변환된 후 모션 제어 ASIC 에 공급됩니다 .

외부 회로에서 입력 신호를 연결하는 방법의 예는 다음과 같습니다 . 입력 회로는 (1) 차동 라인 드라이버 또는 (2) 오픈 컬렉터 출력을 가지는 모터 드라이버 또는 인코더에 연결할 수 있습니다 .

3.2.1 라인 드라이버 출력 연결

PCI-8158 인코더 입력을 구동하기 위해 드라이버 출력은 반드시 적어도 8mA 의 구동 용량을 가져야 하며 적어도 차동 회로를 지나는 전압이 3.5V 가 되어야 합니다 . 또한 , 양 끝은 반드시 묶여 있어야 합니다 . 최대 주파수는 4Mhz 또는 배선의 길이나 신호 상태에 따라 그 이상이 될 수 있습니다 .

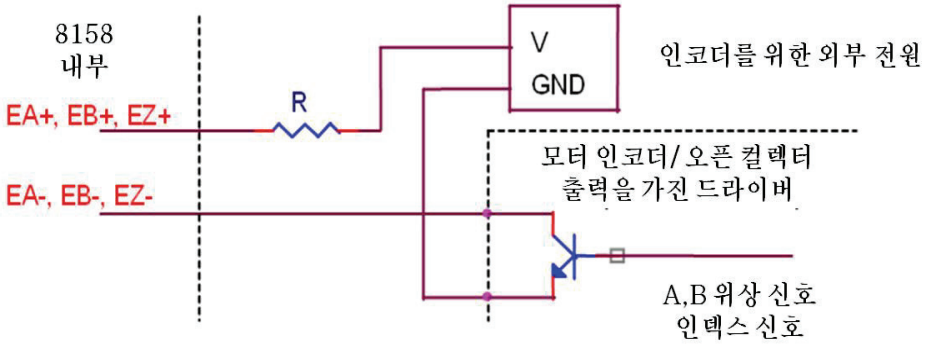


3.2.2 오픈 컬렉터 출력 연결

오픈 컬렉터 출력 연결을 위해서는 외부 전력 공급이 필요합니다 . 일부 모터 드라이버는 전원을 제공합니다 . PCI-8158, 인코더와 외부 전력 공급 사이의 연결은 다음의 그림에서 보여주고 있습니다 . 외부 전류 limit 저항 R 은 PCI-8158 의 입력 회로를 보호하기 위해 필요합니다 . 다음의 표 목록은 인코더 전원 공급에 따른 저항 값을 제안합니다 .

인코더 전원 (V)	외부 레지스터 R
+ 5V	0W(없음)
+ 12V	1.5kW
+ 24V	3.0kW

I_f = 8mA



더 많은 인코더 피드백 신호에 대한 운용 정보는 4.4 를 참고 하십시오 .

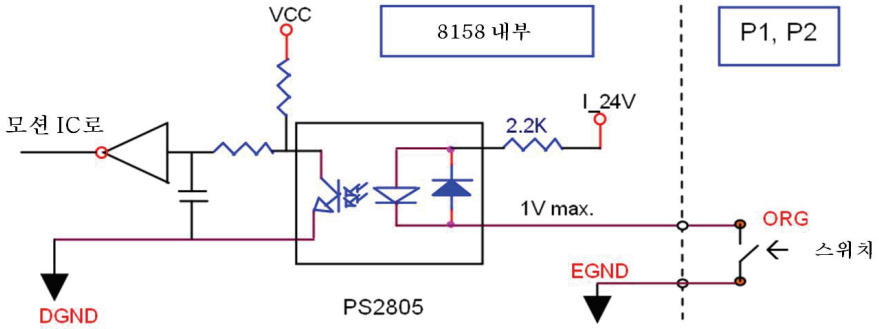
3.3 원점 신호 ORG

원점 신호 (ORG0-ORG7) 는 기구의 원점에 대한 입력신호로써 사용됩니다. 다음의 표는 신호 명칭, 핀 번호, 축 번호를 나타냅니다.

P1 핀번호	신호 명칭	축 #
41	ORG0	0
47	ORG1	1
91	ORG2	2
97	ORG3	3

P2 핀번호	신호 명칭	축 #
41	ORG4	4
47	ORG5	5
91	ORG6	6
97	ORG7	7

ORG 신호의 입력 회로는 다음과 같습니다 . 일반적으로 하나의 limit 스위치는 하나의 축의 원점을 표시하기 위해 사용됩니다 . limit 스위치의 세부사항으로는 + 24V 최소 6mA의 용량을 가져야 하며 내부 필터 회로는 오작동에 의한 높은 주파수를 필터링하기 위해 사용됩니다 .



모션 제어기가 " 홈 리턴 " 모드에서 동작할 때 ORG 신호는 제어 출력 신호 (OUT, DIR) 를 정지하기 위해 사용됩니다 . ORG 신호의 운용에 대해서는 4.3.3 을 참고하십시오 .

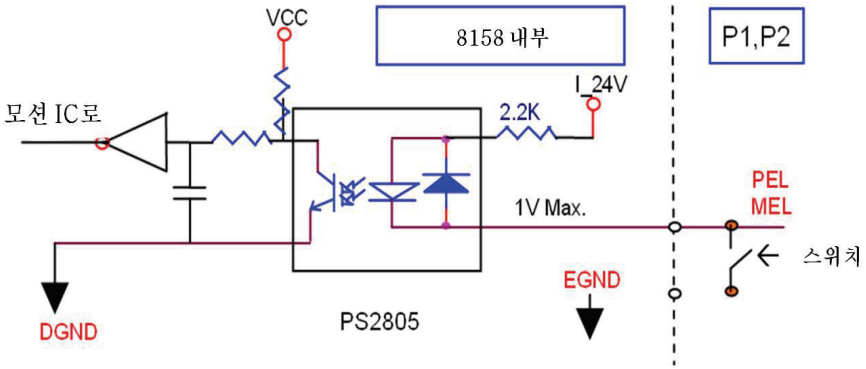
3.4 End-Limit 신호 PEL 과 MEL

본 제품은 한 축에 대해 PEL 과 MEL End-Limit 신호가 있습니다 . PEL 은 양 (+) 의 방향에서의 End-Limit 을 나타내고 MEL 은 음 (-) 의 방향에서의 End-Limit 신호를 나타냅니다 . 다음의 표는 신호 명칭 , 핀 번호 , 축 번호를 나타냅니다 .

P1 핀번호	신호 명칭	축 #	P1 핀번호	신호 명칭	축 #
37	PEL0	0	38	MEL0	0
43	PEL1	1	44	MEL1	1
87	PEL2	2	88	MEL2	2
93	PEL3	3	94	MEL3	3

P2 핀번호	신호 명칭	축 #	P2 핀번호	신호 명칭	축 #
37	PEL4	4	38	MEL4	4
43	PEL5	5	44	MEL5	5
87	PEL6	6	88	MEL6	6
93	PEL7	7	94	MEL7	7

회로도는 아래와 같습니다 . 외부 limit 스위치는 반드시 + 24V 최소 8mA 의 접촉 용량을 가져야 합니다 . A- 타입 ' (normal open) 접촉 또는 B- 타입 ' (normal closed) 접촉 스위치를 사용 할 수 있습니다 . 외부 limit 신호의 자동 논리를 설정하려면 8158 limit_ 논리 기능 설정의 설명을 참조하십시오 .



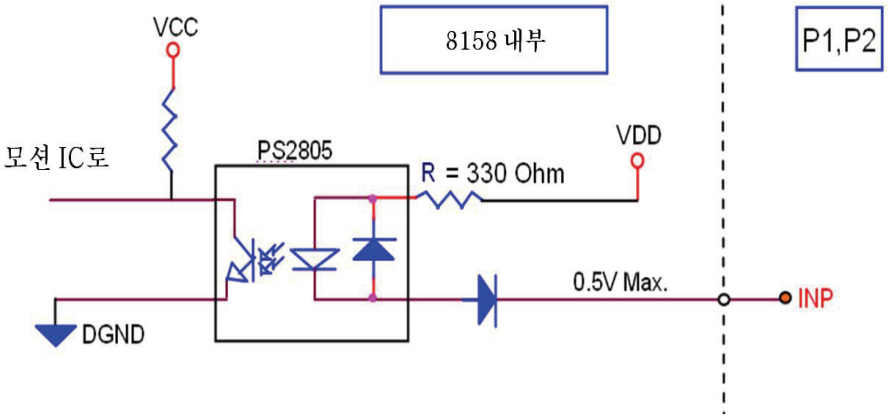
3.5 In-position 신호 INP

서보 모터 드라이버에서의 In-position 신호 INP 는 편차 에러를 의미합니다 . 만약 편차 에러가 없을 경우 서보 위치 에러는 영 (Zero) 을 의미합니다 . 신호 명칭 , 핀 번호 , 축 번호는 아래 표와 같습니다 .

P1 핀번호	신호 명칭	축 #
10	INP0	0
28	INP1	1
60	INP2	2
78	INP3	3

P2 핀번호	신호 명칭	축 #
10	INP4	4
28	INP5	5
60	INP6	6
78	INP7	7

다음 그림은 INP 신호의 입력 회로를 보여줍니다 .



In-position 신호는 일반적으로 오픈 컬렉터 출력 신호를 제공하는 서보 모터 드라이버로 인가됩니다 . 외부 회로는 INP 신호를 구동하기 위해 적어도 8mA 의 전류가 제공되어야 합니다 .

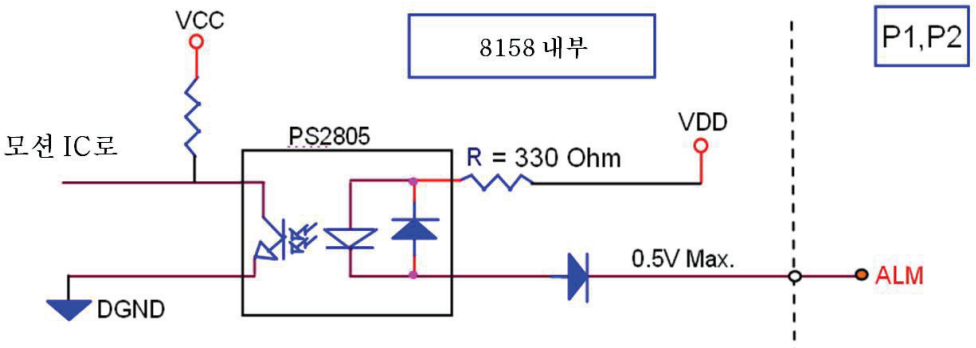
3.6 경보 신호 ALM

경보신호 ALM 은 통상 서보 드라이버로부터의 경보 신호를 나타내기 위해 사용 됩니다. 신호 명칭, 핀 번호, 축 번호는 다음 표와 같습니다.

P1 핀번호	신호 명칭	축 #
9	ALM0	0
27	ALM1	1
59	ALM2	2
77	ALM3	3

P2 핀번호	신호 명칭	축 #
9	ALM4	4
27	ALM5	5
59	ALM6	6
77	ALM7	7

입력 경보 회로는 다음과 같습니다. ALM 신호는 통상 오픈 컬렉터 출력 신호를 제공하는 서보 모터 드라이버로 인가됩니다. 외부 회로는 ALM 신호를 구동하기 위해 반드시 적어도 8mA 를 제공해 주어야 합니다.



3.7 편차 카운터 클리어 신호 ERC

편차 카운터 클리어 신호 (ERC) 는 다음 4 가지 상황에 따라 활성화됩니다 .

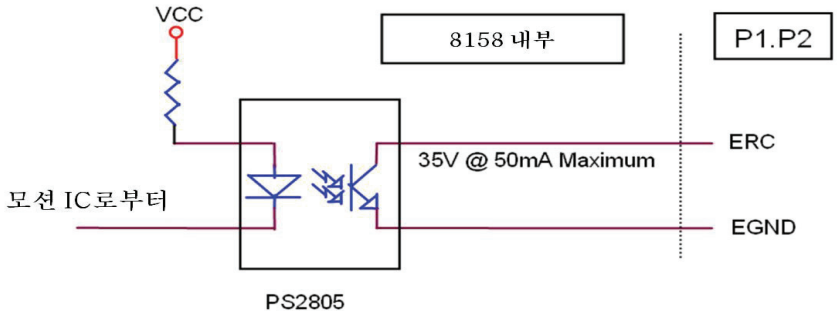
1. 홈 리턴 완료
2. End-limit 스위치 활성화
3. 경보 신호가 OUT 과 DIR 신호를 정지
4. 소프트웨어 (사용자) 로 부터 긴급 정지 명령 활성화

신호 명칭 , 핀 번호 , 축 번호는 다음 표와 같습니다 .

P1 핀번호	신호 명칭	축 #
8	ERC0	0
26	ERC1	1
58	ERC2	2
76	ERC3	3

P2 핀번호	신호 명칭	축 #
8	ERC4	4
26	ERC5	5
58	ERC6	6
76	ERC7	7

ERC 신호는 서보 모터 드라이버의 편차 카운터를 명확히 하는데 사용됩니다 . ERC 출력 회로는 50mA 구동 용량에서 최대 35V 의 외부전원을 사용하는 오픈 컬렉터입니다 .



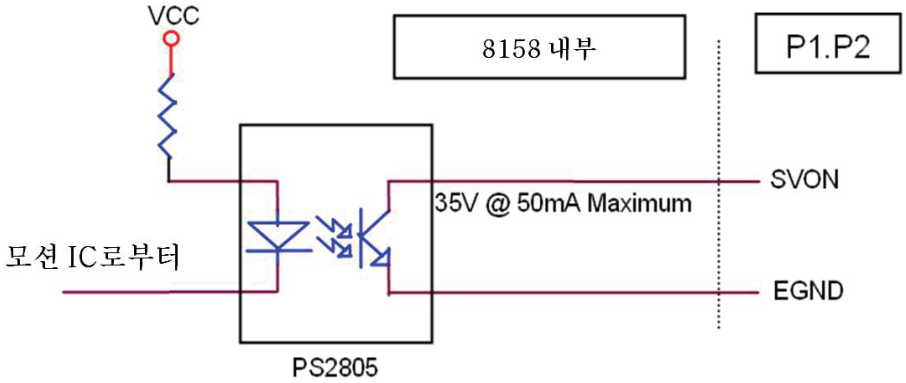
3.8 범용 신호 SVON

SVON 신호는 서보모터 -on 제어 또는 범용 출력 신호로 사용됩니다. 신호 명칭, 핀 번호, 축 번호는 다음 표와 같습니다.

P1 핀번호	신호 명칭	축 #
7	SVON0	0
25	SVON1	1
57	SVON2	2
75	SVON3	3

P2 핀번호	신호 명칭	축 #
7	SVON4	4
25	SVON5	5
57	SVON6	6
75	SVON7	7

SVON 신호를 위한 외부 회로는 다음 그림과 같습니다.



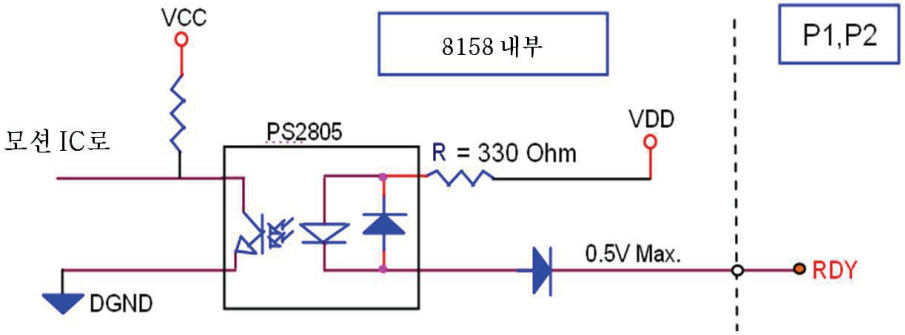
3.9 범용 신호 RDY

RDY 신호는 모터 드라이버 준비 입력 또는 범용 입력 신호로 사용됩니다. 신호 명칭, 핀 번호, 축 번호는 다음 표와 같습니다

P1 핀번호	신호 명칭	축 #
11	RDY0	0
29	RDY1	1
61	RDY2	2
79	RDY3	3

P2 핀번호	신호 명칭	축 #
11	RDY4	4
29	RDY5	5
61	RDY6	6
79	RDY7	7

RDY 신호의 입력 회로는 다음 그림과 같습니다.



3.10 다기능 출력 핀 : DO/CMP

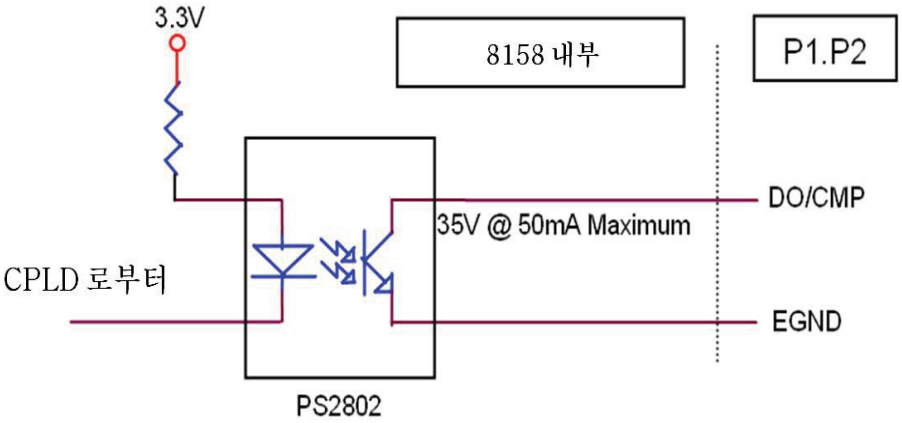
PCI-8158 은 8 개의 축에 반응하는 DO/CMP0 부터 DO/CMP7 의 8 개 다기능 출력 채널을 제공합니다 . 각 출력 핀은 디지털 출력 (DO) 또는 비교 출력 (CMP) 으로 설정할 수 있습니다 . 비교 출력 핀으로 설정한 경우 핀은 사용자가 설정한 선설정 값과 인코더 카운터가 일치할 때 펄스 신호를 발생합니다 .

다기능 채널들은 P1 와 P2 에 위치하며 신호 명칭 , 핀 번호 , 축 번호는 다음 표와 같습니다 .

P1 핀번호	신호 명칭	축 #
40	DO/CMP0	0
46	DO/CMP1	1
90	DO/CMP2	2
96	DO/CMP3	3

P2 핀번호	신호 명칭	축 #
40	DO/CMP4	4
46	DO/CMP5	5
90	DO/CMP6	6
96	DO/CMP7	7

첫 2 개 축의 CMP 회로도 :



3.11 다기능 입력 핀 : DI/LTC/SD/PCS/CLR/EMG

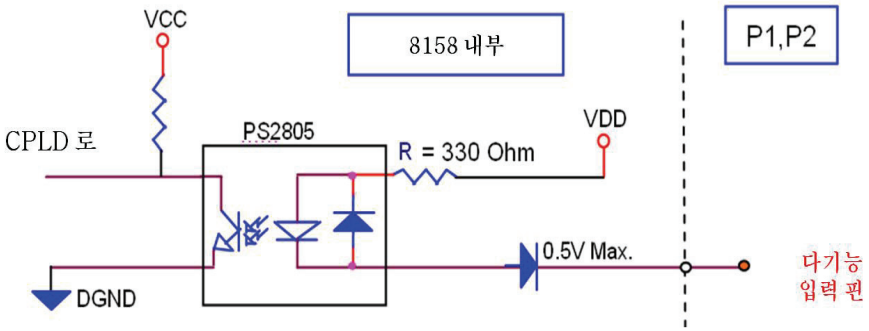
PCI-8158 은 8 개의 다기능 입력 핀을 제공합니다 . 각 8 개의 핀은 DI(디지털 입력) 또는 LTC (Latch), SD (Slow down), PCS(수동 목표 위치), CLR (카운터 클리어), EMG(긴급) 로 설정이 가능합니다 . 핀 기능의 선택은 6.12 를 참조하십시오 .

다기능 입력 핀은 P1 와 P2 에 위치하며 신호 명칭 , 핀 번호 , 축 번호는 다음 표와 같습니다 .

P1 핀번호	신호 명칭	축 #
39	DI/LTC/SD/PCS/CLR/EMG_0	0
45	DI/LTC/SD/PCS/CLR/EMG_1	1
89	DI/LTC/SD/PCS/CLR/EMG_2	2
95	DI/LTC/SD/PCS/CLR/EMG_3	3

P2 핀번호	신호 명칭	축 #
39	DI/LTC/SD/PCS/CLR/EMG_4	4
45	DI/LTC/SD/PCS/CLR/EMG_5	5
89	DI/LTC/SD/PCS/CLR/EMG_6	6
95	DI/LTC/SD/PCS/CLR/EMG_7	7

다음은 다기능 입력 핀 회로도입니다 .



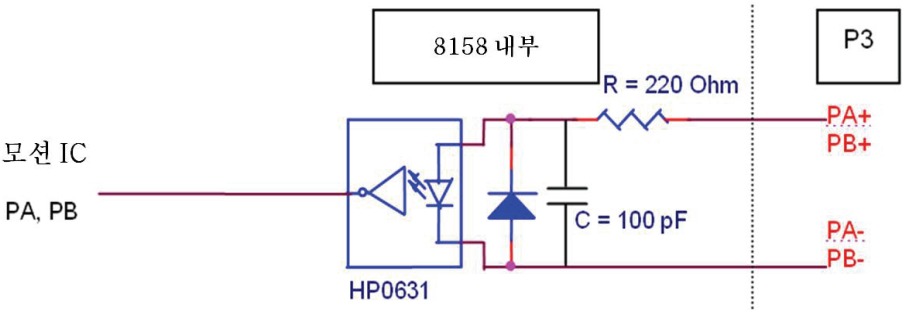
3.12 펄스 입력 신호 PA, PB (PCI-8158)

PCI-8158 은 아래 표에서 보여주는 PN1 의 핀을 통해서 차동 펄스 입력 신호를 얻을 수 있습니다. 여기서 펄스 입력은 주로 인코더에 의해 작동됩니다. A-B 상황 신호는 모터를 안내할 위치 정보를 발생합니다.

P3 핀번호	신호 명칭	축 #	P3 핀번호	신호 명칭	축 #
2	PA+	0-7	3	PA-	0-7
4	PB+	0-7	5	PB-	0-7

펄스 신호는 축 0 부터 축 7 에 대해 사용되며 사용자는 각 축의 펄스에 _8158_disable_pulser_ 입력 기능을 활성화 또는 비활성화 할 것 인지를 결정할 수 있습니다.

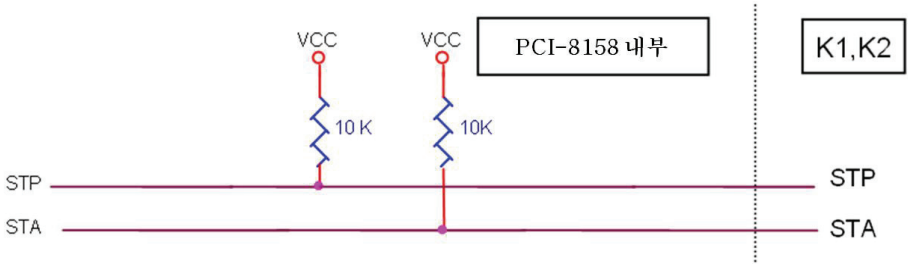
다음 회로도는 차동 펄스 입력 핀을 보여줍니다.



3.13 동시 시작 / 정지 신호 STA 와 STP

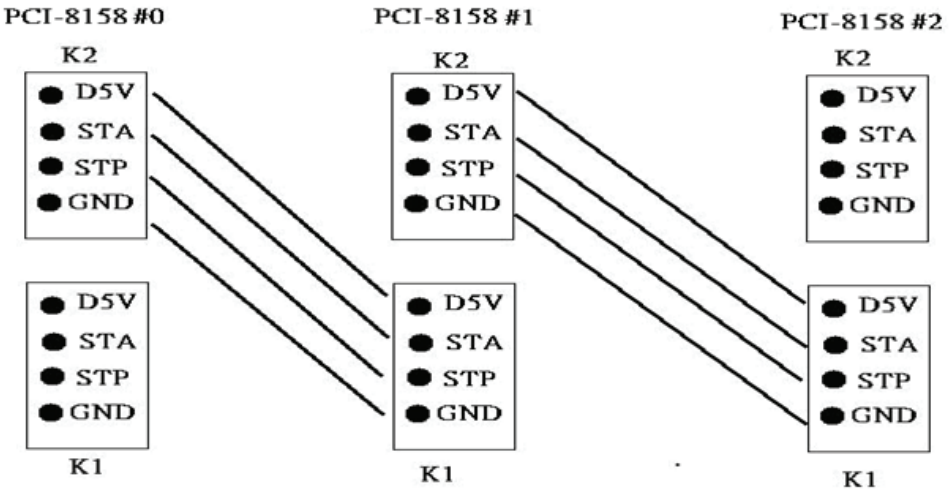
PCI-8158 은 다 축 모션에 동시 시작 / 정지를 제어할 수 있는 STA, STP 신호를 제공합니다 . STA 와 STP 는 CN4 에 위치합니다 .

다음 그림은 보드 상의 회로를 보여줍니다 . 4 개의 축의 STA 와 STP 신호는 서로 형태에 맞게 연결되어 있습니다 .

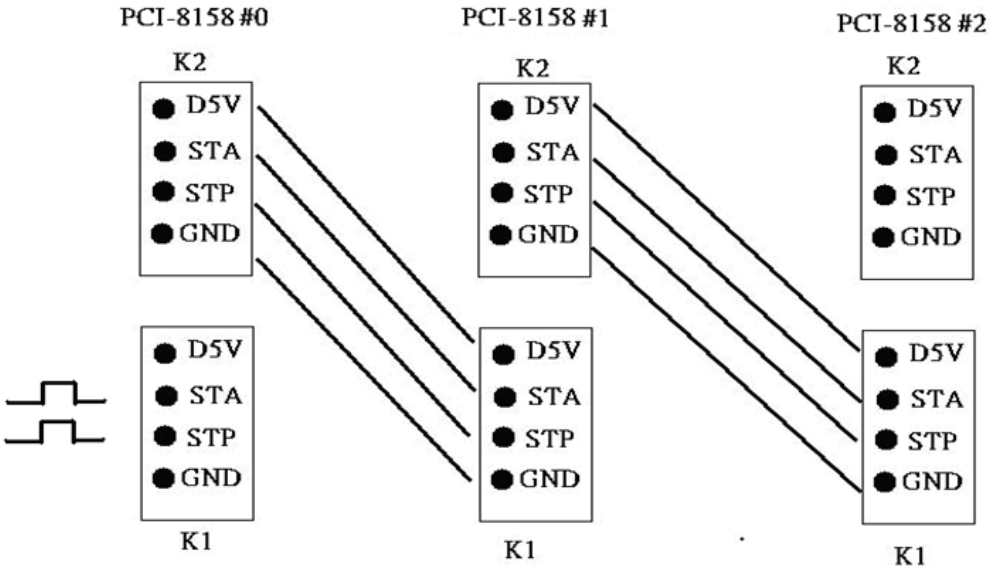


STP 와 STA 신호는 입력과 출력 신호가 있으며 동시 시작 / 정지 동작을 위해서는 소프트웨어와 외부 제어가 둘 다 필요합니다 . 소프트웨어 제어와 함께 PCI-8158 의 어느 한 곳에서 신호가 발생될 것 이며 사용자는 또한 동시 시작 / 정지를 위한 STP/STA 신호의 작동을 위하여 외부 오픈 컬렉터 또는 스위치를 사용할 수 있습니다 .

만약 두 개 이상의 PCI-8158 카드가 있는 경우 연결할 카드의 K1 커넥터에 K2 커넥터를 연결합니다. K1 와 K2 커넥터는 같은 PCI-8158 에 내부적으로 연결 됩니다 .



귀하는 또한 외부의 시작 / 정지 신호를 이용하여 crosscard 동시 모터 운영을 하실 수 있습니다 . 첫 PCI-8158 카드 K1 커넥터 상의 STA/STR 의 핀에 외부의 시작 / 정지 신호를 연결하십시오 .


PCI-8158 #0
K2
D5V
STA
STP
GND
D5V
STA
STP
GND
K1
PCI-8158 #1
K2
D5V
STA
STP
GND
D5V
STA
STP
GND
K1
PCI-8158 #2
K2
D5V
STA
STP
GND
D5V
STA
STP
GND
K1

의도적인 공백 페이지

4 동작 원리

이 장에서는 모션 제어 카드의 동작에 대하여 자세히 설명하고 있습니다. 내용은 다음과 같습니다.

Section 4.1: 모션 제어 분류
Section 4.2: 모션 제어 모드
Section 4.3: 모션 드라이버 인터페이스
Section 4.4: 기계적 스위치 인터페이스
Section 4.5: 카운터
Section 4.6: 콤퍼레이터
Section 4.7: 다른 모션 기능들
Section 4.8: 인터럽트 제어
Section 4.9: 여러 장의 카드 동작

4.1 모션 제어기의 분류

서보 / 스테퍼 드라이버를 처음 시작할 때 모터 제어는 모터 제어와 모션 제어의 두 가지로 분류됩니다. 모터 제어는 PWM, 전력, Closed loop, Hall 센서, 벡터 스페이스 등에 관련이 있고 모션 제어는 속도 프로파일 생성, 궤도 수행, 다 축 동기화 및 좌표와 관련이 있습니다

4.1.1 전압 타입의 모션 제어 인터페이스

모션과 모터 제어 사이의 인터페이스는 빠르게 변화하고 있습니다. 초기에 전압 신호는 모터 제어기에 대한 명령 신호로써 사용되었습니다. 신호의 진폭은 얼마나 빠르게 모터를 돌릴 수 있는 지를 의미하며 전압 신호의 속도 변화는 얼마나 빠르게 모터가 가속할 수 있는지를 의미합니다. 모터 드라이버에 명령신호로 전압 신호를 사용하는 경우 이는 “아날로그” 타입의 모션 제어기로 불립니다. 이는 모션제어기의 아날로그 회로로 통합하기 용이하지만 가금씩의 소음은 이 타입의 모션 제어의 큰 문제점입니다. 게다가 귀하가 모터의 위치 제어를 하기 원하신다면 아날로그 타입의 모션 제어기는 반드시 위치 정보의 피드백 신호를 가져야 하며 이를 가능하게 하기 위해서는 Closed loop 제어 알고리즘을 사용하셔야 합니다. 이는 모션 제어의 복잡성을 증가시킬 수 있습니다.

4.1.2 펄스 타입의 모션 제어 인터페이스

두 번째 모션, 모터 제어 인터페이스 타입은 펄스열 타입입니다. 디지털 세상의 추세에 따라 펄스 열 타입은 새로운 컨셉의 모션 제어를 대표합니다. 펄스의 수는 모터 회전의 스텝 수를 보여주며 펄스의 주파수는 얼마나 빨리 모터를 가동할지를 보여줍니다. 또한, 주파수 지속 시간의 변화는 모터의 가속률을 보여줍니다. 이 인터페이스의 특징 때문에 사용자는 서보 또는 스테퍼 모터의 위치 제어 응용프로그램을 아날로그 타입에서 보다 훨씬 쉽게 사용하실 수 있습니다 . 이는 모션과 모션 제어가 이 방법을 통해 쉽게 분류될 수 있음을 의미합니다.

두 인터페이스는 Gain 튜닝을 지원해야 합니다. 아날로그 위치 제어기의 경우 제어루프가 내장되어 있어야 하며 사용자는 반드시 제어기로부터 Gain 튜닝을 해주어야 합니다. 펄스 타입의 위치 제어기의 경우 제어루프는 모터 드라이버의 외부에 위치하며 사용자는 드라이버에 Gain 튜닝을 해주어야 합니다.

한 축 이상의 동작을 위해서는 모션 제어는 모터 제어보다 중요합니다. 산업용 응용에서 신뢰성은 아주 중요한 요소입니다. 모터 드라이버 생산자는 좋은 성능의 제품을 만들고 모션 제어기 생산자는 강력하고 다양한 모션 소프트웨어를 만듭니다. 이 두 가지 제품이 합쳐질 때 본 제품은 완벽해질 수 있습니다.

4.1.3 네트워크 타입의 모션 제어 인터페이스

네트워크 모션 제어기는 최근에 소개되었습니다 . 모터 드라이버와 모션 제어기 사이의 명령은 더 이상 아날로그 또는 펄스 신호가 아니며 위치 정보와 모터 정보를 포함하는 네트워크 패킷을 사용하는 이 타입의 제어기는 디지털화와 패킷화 되므로 훨씬 더 신뢰할 수 있습니다. 모션 제어기는 반드시 실시간 제어되어야 하므로 네트워크는 반드시 1ms 이하 사이클 타임의 실시간 처리 용량을 가져야 합니다. 미츠비시의 SSC-NET 네트워크는 이 속도를 만족하는 네트워크 중 하나입니다.

4.1.4 소프트웨어 실시간 모션 제어 커널

모션 제어 커널은 DSP 기반, ASIC 기반, 소프트웨어 실시간 기반의 세 가지 방법을 사용합니다.

모션 제어 시스템은 확실한 실시간 제어 사이클과 연산을 필요로 하며 이러한 사이클을 이용하여 제어기는 제어 데이터를 제공해야 합니다. 만약 모터가 부드럽게 작동하지 않을 경우 일반적으로 PC

의 연산 능력을 사용하여 문제를 해결합니다 . 또한 , 간단한 피드백 카운터 카드 , 전압 출력 또는 펄스 출력 카드를 사용하기도 합니다 . 이 방법은 저가이지만 소프트웨어의 개발이 필요하며 실시간 성능을 개발하기 위해서는 실시간 소프트웨어를 시스템에 사용해야 합니다 . 이는 시스템을 더욱 복잡하게 하지만 전문적인 모션 설계자를 위해서는 높은 유연성을 가진 방법이며 대부분의 경우 NC 장비에 사용되고 있습니다 .

4.1.5 DSP 기반의 모션 제어 커널

DSP 기반의 모션 제어 커널은 컴퓨터 상의 실시간 소프트웨어 문제들을 해결할 수 있습니다 . DSP 는 마이크로 프로세서이며 모든 모션 제어 연산은 이것을 통해서 할 수 있습니다 . 또한 , DSP 는 관련된 모든 과정의 자체 OS 를 가지고 있기 때문에 실시간 소프트웨어 문제가 존재하지 않습니다 . 또한 , 다른 입력이나 윈도우 기반의 컴퓨터와 같은 상황정보 스위칭 문제로부터의 문제점이 없습니다 . 비록 DSP 가 실시간 요구에 적합한 완벽한 성능을 가졌다고 하여도 연산 속도는 오늘날의 컴퓨터와 같이 빠르지 않습니다 . 문제점은 DSP 기반 제어기의 생산자와 사용자 간의 소프트웨어 인터페이스가 사용하기 쉽지 않다는 것입니다 . 몇몇 제어기 공급자는 사용자의 학습을 위한 어셈블리 언어와 같은 프로그램을 제공하고 몇몇 제어기 공급자는 간단한 참고 문서만을 제공합니다 . 이런 두 가지 방법 모두 사용하기에는 쉽지 않으며 일반적으로 DSP 기반의 제어기는 장비 제조업자가 쉽게 적용하기 위하여 소프트웨어 커널보다 더 나은 방법을 제공해야 합니다 .

4.1.6 ASIC 기반의 모션 제어 커널

ASIC 기반의 모션 제어 커널은 DSP 커널과는 조금 다릅니다 . 이는 모든 모션 기능을 ASIC 를 통해서 수행하므로 실시간 제어에 전혀 문제가 없습니다 . 사용자 또는 제어기 공급자는 ASIC 가 요구하는 몇 가지 파라미터의 설정을 필요로 하는데 이는 수행하기 쉽습니다 . 이러한 종류의 모션 제어는 모든 시스템의 통합에 발생하는 문제를 4 가지로 나뉘는데 이는 모터 드라이버의 성능 , ASIC 출력 프로파일 , 공급자의 ASIC 에 대한 소프트웨어파라미터 , 공급자의 소프트웨어에 대한 사용자의 명령입니다 . 이는 모션 제어기가 장치 간 좀더 부드럽게 연동 되는 것을 돕습니다 .

4.1.7 모든 모션 제어 타입의 비교표

	소프트웨어	ASIC	DSP
가 격	* 적 당	저 렴	고 가
기 능	최 상	낮 음	보 통
유 지	어려움	쉬 움	보 통

* 실시간 OS 포함

	아날로그	펄스	네트워크
가 격	고 가	저 렴	** 보 통
신호 품질 (거리에 따라 다름)	보 통	좋 음	최 상
유 지	어려움	보 통	쉬 움

** DSP 또는 소프트웨어 실시간 OS 필요함

4.1.8 PCI-8158 의 모션 제어기 타입

PCI-8158 은 ASIC 기반의 펄스 타입 모션 제어기입니다. 이 제어기는 모션 ASCI, PCI 카드 , 소프트웨어 모션 라이브러리의 세 가지 부분으로 이루어져 있습니다. 사용자는 Windows 2000/XP, 리눅스 , RTX 드라이버 하에서 드라이버에서 제공하는 소프트웨어 모션 라이브러리를 통해 모션 ASCI에 쉽게 접근할 수 있습니다. 이 제품의 소프트웨어 모션 라이브러리는 모터를 제어하기 위한 one-stop 함수를 제공하며 모든 속도 파라미터의 연산은 이 제품의 라이브러리를 통해 행해집니다.

예를 들어 만약 귀하가 한 축에 대한 지점 간 모션을 trapezoidal 속도 프로파일을 이용하여 수행할 경우 단지 하나의 함수의 가속 시간, 속도, 목표 위치만 정하면 동작이 가능합니다. 이후 모터는 프로파일에 따라 동작하게 됩니다. 이러한 작업은 CPU 의 자원을 사용하지 않는데 그 이유는 모든 제어 사이클 펄스는 ASIC에 의해 수행되기 때문입니다. 목표 위치의 정확성은 모션 제어기의 명령값이 아닌 closed loop 제어 수행능력과 모터 드라이버의 기계적 부분에 의존합니다. 그 이유는 모션 제어기는 단지 요구되는 속도 프로파일을 통해 정확한 펄스를 출력하는 부분에만 책임이 있기 때문입니다. 이는 프로그래머, 기술자, 엔지니어들이 문제의 원인을 찾아내는 것을 훨씬 용이하게 해줍니다.

4.2 모션 제어 모드

모션 제어는 양 (+) 또는 음 (-) 의 움직임뿐 아니라 , 다른 축과 동기 조건 , 궤도 경로 , 지정된 속도 프로파일에 따라 모터가 구동할 수 있도록 돕습니다 . 이하의 내용은 모션 제어기가 수행할 수 있는 모션 제어 모드에 대해 자세히 설명하고 있습니다 .

4.2.1 좌표 시스템

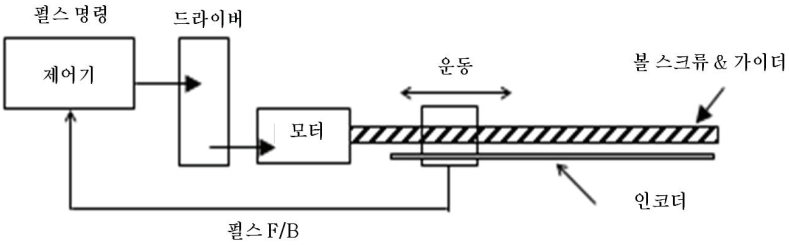
데카르트 좌표 시스템이 사용되며 펄스는 길이의 단위입니다 . 물리적 길이는 기계적 부분과 모터의 분해능에 따라 다릅니다 . 예를 들어 , 만약 모터가 피치가 10mm 인 스크류 볼에 설치되고 모터 한 바퀴당 필요한 펄스가 10,000 펄스일 때 한 펄스의 물리적 단위는 10mm/10,000p=1 마이크로미터입니다 .

만약 모션 제어기를 15mm 이동하고 싶다면 명령 펄스를 15,000 으로 설정 하시면 됩니다 . 만약 15.0001mm 를 움직이고 싶다면 어떻게 해야 할까요? 모션 제어기는 1 펄스보다 작은 잔여 값은 기억해 두었다가 다음 명령에 포함 시킵니다 .

모션 제어기는 증가형 펄스를 모터 드라이버로 보냅니다 . 이것은 우리가 단지 상대적 명령을 모터 드라이버로 보낼 수 있다는 것을 뜻하지만 , 이 문제는 초기에 현재 위치와 목표 위치 사이의 차이를 계산함으로써 해결되었습니다 . 예를 들어 만약 현재 위치가 1000 이고 9000 으로 모터를 이동하고 싶다면 사용자는 절대 명령을 사용하여 목표 위치를 9000 으로 설정 하면 됩니다 . 모션 제어기 내부에는 먼저 현재 위치 1000 을 얻은 후 목표 위치인 9000 과의 차이점을 얻어낼 수 있습니다 . 이것은 + 8000 이며 모션 제어기는 8000 펄스를 목표 위치인 9000까지의 이동 명령으로 모터 드라이브에 전송하게 됩니다 .

때때로 귀하는 기계의 위치 값을 확인하기 위하여 선형 스케일 또는 외부의 인코더를 설치할 수 있습니다 . 하지만 어떻게 이 좌표 시스템을 설정할 수 있을까요? 만약 외부 인코더의 분해능이 1mm 당 10,000 펄스이면 모터는 1mm 이동을 위해 1000 펄스를 전송할 것입니다 . 이것은 1mm 이동을 위해서 1000 펄스를 모터 드라이버로 보냈고 10,000 펄스의 인코더 피드백 값을 받는다는 뜻입

니다 . 만약 우리가 절대 명령으로 모터를 10,000 펄스 위치로 이동시키고 싶다면 얼마의 펄스를 모터드라이버로 보내야 할까요 ? 답은 (10000 – 3500) / (10,000 / 1,000)=650 펄스 입니다 . 모션 제어기는 만약 귀하가 이미 “move ratio” 로 설정하신 경우 자동으로 연산할 것이며 ‘’move ratio” 는 피드백 분해능 / 명령 분해능과 동일 합니다 .

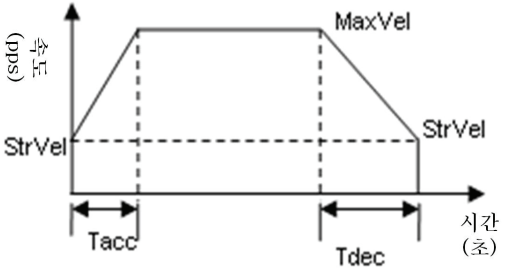


4.2.2 절대 , 상대 위치 이동

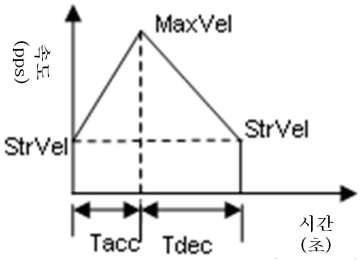
본 제품은 좌표 시스템에서 목표 위치로 이동하기 위해 절대 이동과 상대 이동의 두 가지 명령이 존재합니다 . 절대이동 명령은 사용자가 모션 제어기에 위치 값을 주면 모션 제어기는 현재의 위치부터 목표위치까지 이동하게 됩니다 . 상대이동 명령은 모션 제어기에 거리 값을 주면 모션 제어기는 현재의 위치에서 거리 값에 의하여 모터를 움직이게 됩니다 . 이동 중에 귀하는 속도 프로파일을 설정할 수 있는데 이는 귀하가 얼마나 빠르게 어떠한 속도로 위치에 도착할 것인가를 결정할 수 있습니다 .

4.2.3 Trapezoidal 속도 프로파일

Trapezoidal 은 가속 / 감속 구간이 1 차 성분 선형 속도 프로파일을 따르는 것을 말합니다 . 이 프로파일 차트는 다음과 같습니다 .



속도 프로파일 구간은 이 모션의 거리를 나타내며 때때로 요구되는 거리가 주어진 속도 파라미터 구간보다 작을 경우 삼각형으로 보이기도 합니다 . 이러한 상황이 발생할 때 모션 제어기는 최대 속도보다 작아지게 되지만 가속율은 요구거리를 만족시키기 위해 유지하게 됩니다 . 다음 차트는 이와 같은 상황을 보여 줍니다 .

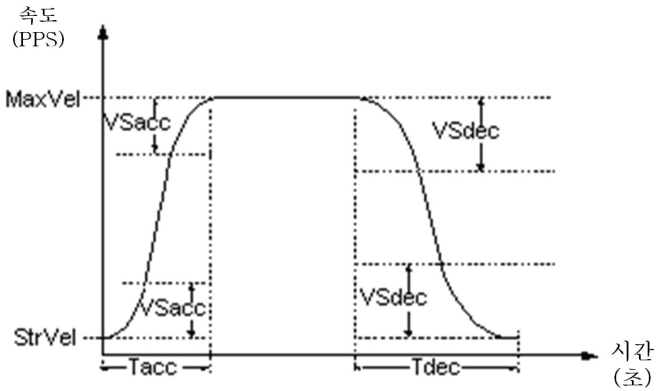


이러한 종류의 속도 프로 파일은 한 축 또는 다 축의 선형 보간이나 2 개의 축의 원형 보간 모드에서 속도 모드 , 위치 모드에 적용이 가능합니다 .

4.2.4 S- 커브와 Bell- 커브 속도 프로파일

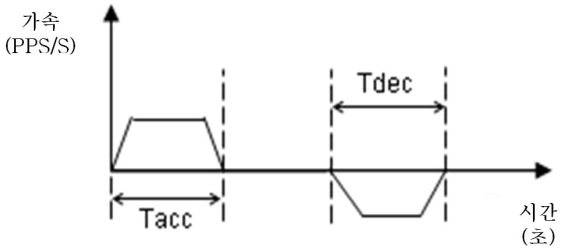
S- 커브는 가속 / 감속 구간이 2 차 성분 커브를 따르는 것입니다 . 이것은 모터의 시작과 정지 동작의 시작 시 진동을 감소시킬 수 있습니다 . 모션 중 가속 / 감속의 속도 증가를 위해 S- 커브 구간에 선형 구간이 필요하고 이를 “Bell” 커브라 부릅니다 . 이는 S- 커브의 상위 구간과 하위 구간 사이에 선형 커브를 추가하게 되고 이러한 형상은 가속의 속도를 증가시키며 가속으로 인한 진동을 감소시킵니다 .

다음 그림은 Bell 커브 형상의 파라미터를 보여줍니다 .



- Tacc: 초당 가속 시간
- Tdec: 초당 감속 시간
- StrVel: 시작속도 (PPS 단위)
- MaxVel: 최대속도 (PPS 단위)
- VSacc: 커브의 가속 S 커브 부분 (PPS 단위)
- VSdec: 커브의 감속 S 커브 부분 (PPS 단위)

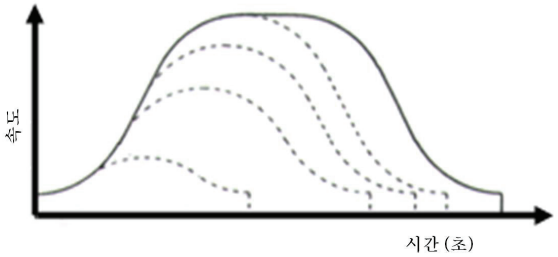
만약 VSacc 또는 VSdec 값이 0 일 경우 가속 또는 감속이 어떤 선형 구간 없이 순수 S- 커브라는 것을 의미합니다 . Bell 커브의 가속 차트는 다음과 같습니다 .



S- 커브 프로파일 모션 함수는 항상 " 부드러운 모션 " 을 하도록 설계되었습니다 . 만약 최종 위치를 가지는 가속 파라미터에 대한 시간이 최대 속도로 축에 도달하는 것을 허용하지 않을 경우 . 즉 이동거리가 너무 작아서 최고 속도에 도달하지 않는 경우에는 최대 속도는 자동으로 낮아지게 되고 이는 다음의 그림과 같습니다 .

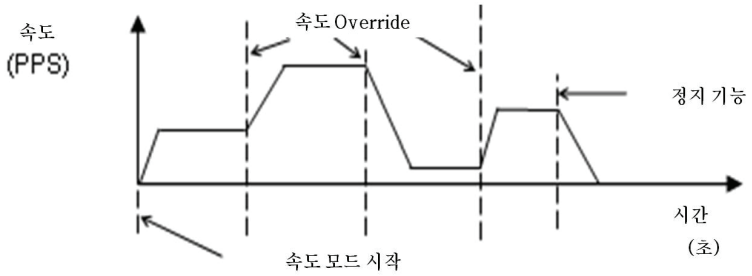
이 규칙은 MaxVel 과 Tacc, Tdec, VSacc, VSdec 의 값을 자동적으로 낮추며 StrVel, 가속 , 변경되지 않은 jerk 의 값을 유지합니다 . 이는 또한 Trapezoidal 프로파일 모션에 적용 가능 합니다 .

이러한 속도 프로파일은 한 개의 축 또는 다축의 선형 보간과 두 개의 축 간 원형 보간 모드의 위치모드 , 속도 모드에 적용할 수 있습니다 .



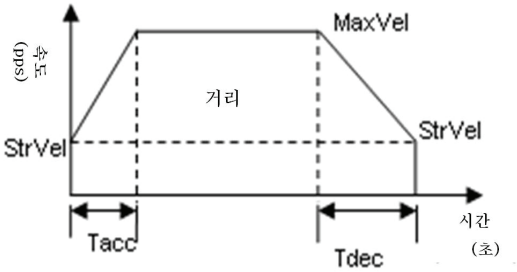
4.2.5 속도 모드

속도 모드는 정지 명령이 내려질 때까지 계속된 출력을 하는 것을 의미합니다 . 모터는 목표 위치 또는 요구되는 거리 값이 없이 다른 이유에 의해 정지될 때까지 동작하게 됩니다 . 출력 펄스는 시작 속도부터 정해진 최대 속도까지 가속하게 되며 이는 선형 또는 S- 커브 가속 형태를 지닐 수 있습니다 . 펄스 출력 비율은 다른 속도 명령이 설정되거나 정지 명령이 내려질 때까지 최대 속도를 유지합니다 . 속도는 새로운 속도 설정에 의해 치환될 수 있으며 새로운 속도는 원래의 진행 속도에 반대되는 속도가 될 수 없습니다 . 이러한 종류의 모션 속도 프로파일은 다음과 같습니다 .



4.2.6 한 축의 위치 모드

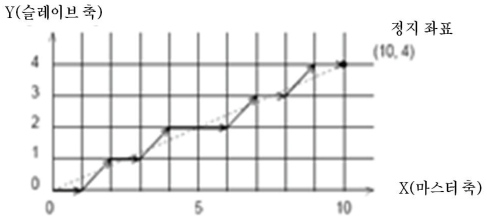
위치 모드는 모션 제어기가 요구된 위치 또는 거리 값과 같은 지정된 펄스 수를 출력하는 것을 의미합니다. 거리나 위치의 단위는 모션 제어기 내부에서 지정되며 거리의 최소 단위는 1 펄스입니다. PCI-8158에서는 사용자가 물리적 거리를 펄스로 변환하기 위한 실수 사용이 가능한 함수를 제공하며 소프트웨어는 1 펄스보다 작은 거리 값을 레지스터에 보관하였다가 다음 모션 동작에 적용합니다. 펄스 카운터를 통한 위치이동을 제외하고 본 모션 제어기는 세 가지 타입의 속도 프로파일을 위치이동을 위해 제공하는데 이는 1 차 성분 trapezoidal, 2 차 성분 S- 커브 그리고 혼합된 Bell 커브입니다. 사용자는 각 함수를 사용 가능하며 다음의 그림은 거리 값과 속도 프로파일의 관계를 보여줍니다.



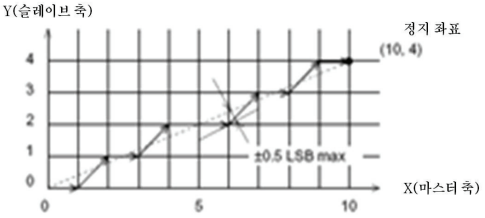
거리는 이 프로파일의 V-t 다이어그램 구간입니다.

4.2.7 두 축의 선형 보간 위치 모드

“다축간 보간”은 축들이 동시에 모션을 시작하고 지정된 위치에 도달하는 것을 말하며 선형은 모든 축의 속도비율이 상수 값을 가지는 것을 의미합니다. 모션을 (0,0) 부터 (10,4) 까지 이동시킨다고 가정할 때 선형 보간의 결과는 다음과 같습니다. X 축 또는 Y 축부터의 펄스 출력은 완벽한 선형 라인에 따라 1/2 의 펄스 차가 남아있습니다. 선형 보간의 정확도는 다음 그림에서 보실 수 있습니다.



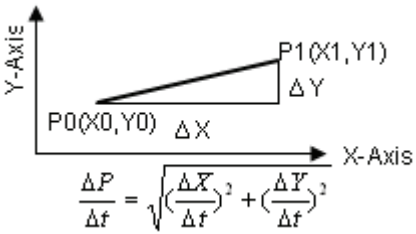
X 또는 Y 축으로부터의 펄스 출력은 완벽한 선형 보간에 따라 1/2 펄스차가 남아있게 되며 선형 보간의 정확도는 다음 그림을 통해 확인하실 수 있습니다.



보간 그룹의 정지를 위해서는 단지 첫 축의 그룹에 대한 정지 함수를 호출하면 됩니다.

밑의 그림과 같이 8 축 선형 보간은 XY 위치가 P0 부터 P1 으로 이동 하는 것을 의미합니다. 두 개의 축은 동시에 시작 / 정지 하며 경로는 직선을 나타냅니다.

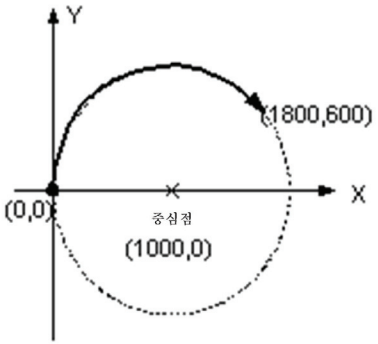
X 축과 Y 축의 속도 비율은 (ΔX: ΔY) 이며 각각의 벡터 속도는 :



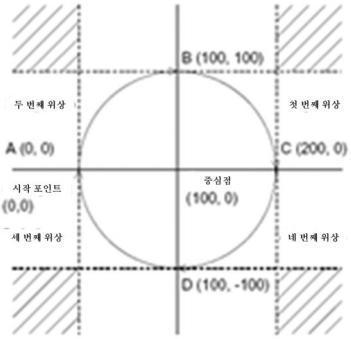
8 축 선형 보간 함수를 호출할 경우 벡터 속도는 시작 속도 , StrVel, 최대 속도 , MaxVel 의 정의를 필요로 합니다 .

4.2.8 두 축간 원형 보간 모드

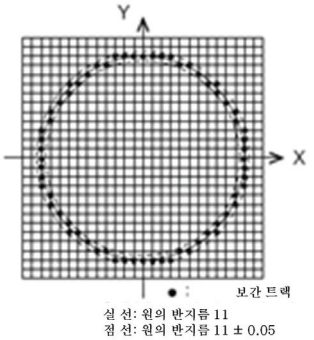
원형 보간은 XY 축이 모두 동시에 특정한 지점 , (0,0) 에서 출발하여 정지지점 (1800,600) 에서 정지하는 것을 의미합니다 . 두 지점 사이의 경로는 arc 이며 MaxVel 은 접선속도입니다 . 만약 arc 의 정지지점이 적절한 위치가 아닌 경우 원형 이동은 정지하지 않을 것입니다 .



모션 제어기는 사용자가 원하는 지점이 arc 경로가 아닐지라도 작동하겠지만 , 만약 정지 지점이 아래 그림의 검은색 부분이 아닌 경우 원형이동은 정지하지 않을 것입니다 .



원형 보간의 명령 정확도는 다음과 같으며 정확도 범위는 반지름 ±1/2 펄스입니다.

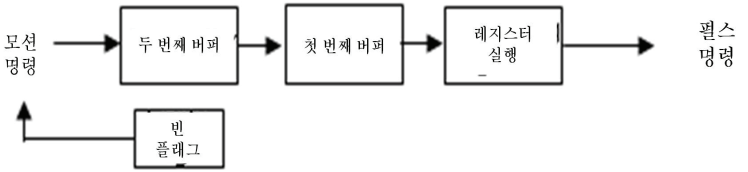


4.2.9 연속적 모션

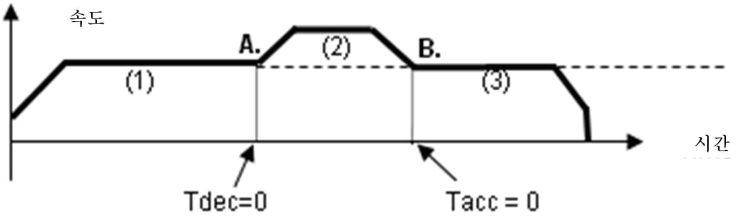
연속적 모션은 일련의 모션 명령 또는 계속해서 동작 가능한 위치를 의미합니다. 귀하는 이전의 동작 이후 간단하게 새로운 명령을 설정할 수 있습니다. 본 모션 제어기는 이미 설정된 내부의 세 가지 커맨드 버퍼를 동해 이를 가능하게 만들었습니다.

첫 번째 명령을 실행할 때 귀하는 두 번째 명령을 첫 번째 버퍼에, 세 번째 명령을 두 번째 버퍼에 설정할 수 있습니다. 첫 번째 명령이 완료되면 모션 제어기는 두 번째 명령을 실행 레지스터에 입력할 것이고 세 번째 명령을 첫 번째 버퍼에 입력할 것입니다. 이 때 두 번째 버퍼는 비어있게 되는데 사용자는 네 번째 명령을 두 번째 버퍼에 설정할 수 있습니다. 통상적으로 만약 사용자가 실행

레지스터가 완료 되기 전 두 번째 버퍼에 새로운 명령을 설정할 수 있는 충분한 시간이 있다면 모션은 정지하지 않고 영원히 동작할 것 입니다. 다음의 그림은 계속되는 모션의 구조를 보여줍니다.



위치 명령 이외에 속도 명령은 속도의 계속되는 프로파일의 구동을 위해 정확히 설정되어야 합니다. 이하의 예시는 계속되는 모션의 세 가지 모션 명령이 존재하고 두 번째 명령은 다른 명령보다 빠른 속도를 가지고 있습니다. 이러한 세 가지 모션 함수 간의 속도의 상호 연결은 다음 그림과 같이 설정됩니다.



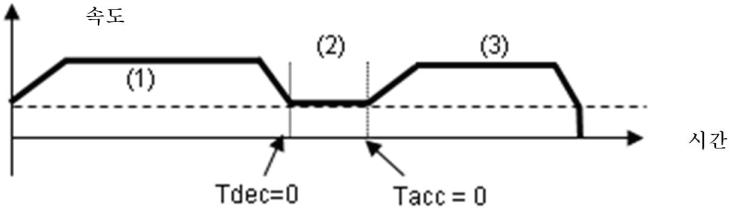
첫 번째 명령의 Tdec = 0

두 번째 명령의 StrVel = 첫 번째 명령의 MaxVel

세 번째 명령의 Tacc = 0

세 번째 명령의 MaxVel = 두 번째 명령의 StrVel

만약 세 번째 명령의 속도 값이 다른 명령보다 작다면 다음 그림과 같이 설정될 것입니다 .



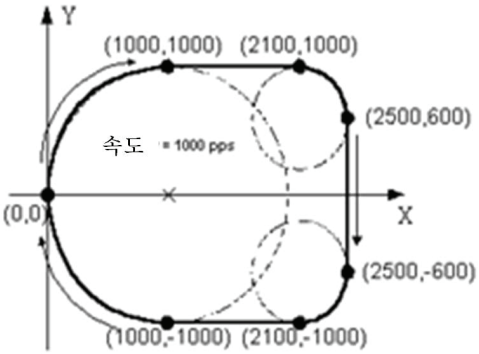
두 번째 명령의 Tacc = 0

두 번째 명령의 Tdec = 0

두 번째 명령의 MaxVel = 첫 번째 명령의 StrVel

두 번째 명령의 MaxVel = 두 번째 명령의 StrVel

이것은 8 축의 계속되는 원호 보간에 대해서도 같은 컨셉을 가집니다 . 귀하는 두 가지 명령의 속도 설정 간 속도를 일치하도록 설정할 수 있습니다 .



INP 검사가 활성화된 경우 모션은 버퍼의 각 명령 간 약간의 지연시간을 가지게 됩니다 . INP 검사 활성화는 요구되는 지점에 도달하게 하지만 각 명령 간의 smoothing 을 감소시킵니다 . 지연시간을 원치 않거나 부드러운 모션을 원하는 경우 INP 검사 기능을 끄십시오 .

4.2.10 홈 리턴 모드

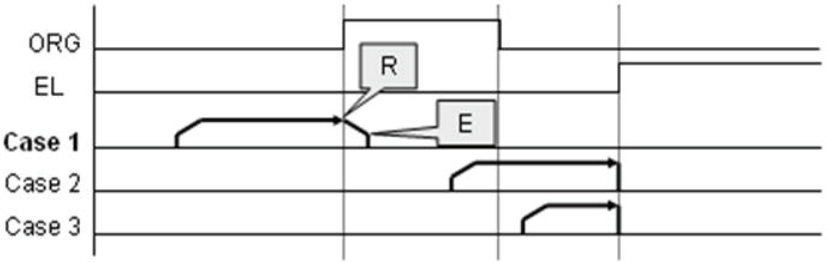
홈 리턴 모드는 좌표에서 0 점의 위치를 찾는 것을 의미합니다. 때때로 사용자는 ORG, EZ 또는 EL 을 좌표상의 0 점 위치로 사용합니다. 시스템의 전원 공급 동안 프로그램은 기계의 0 점을 찾아야 하며 본 모션 제어기는 이를 위한 홈 리턴 모드를 제공합니다.

본 제품은 많은 홈 리턴 모드와 각 모드의 내용은 많은 제어 상황을 포함하고 있습니다. 이러한 모든 상황은 ASIC 를 통해 수행되며 CPU 의 로딩이나 다른 소프트웨어는 필요치 않습니다. 예를 들어 상승 엣지 시 ORG 입력의 경우 홈 리턴 이 완료된 후 목표 카운터는 홈 모드의 요구되는 조건에서 0 으로 리셋될 것입니다. 때때로 카운터가 리셋된 이후에도 기계에는 여전히 출력 펄스가 나타날 수 있습니다. 모터가 정지하였을 때 카운터는 0 점이 아니겠지만 홈 리턴 과정은 이미 완료되었습니다. 이때 귀하가 보는 카운터 값은 기계의 0 점에서부터의 참조 위치입니다.

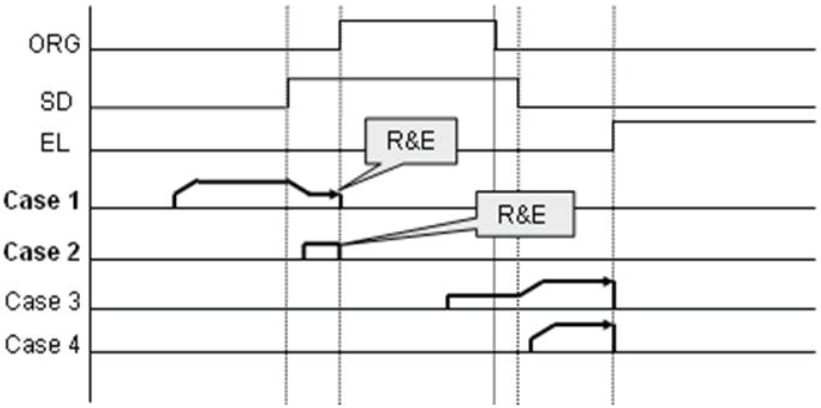
이하의 그림은 여러 가지 홈 리턴 모드를 보여줍니다. R 은 명령과 위치 카운터의 카운터 리셋을 의미하고 E 는 ERC 신호 출력을 의미합니다.

홈 모드=0 (ORG 켜진 후 리셋 카운터)

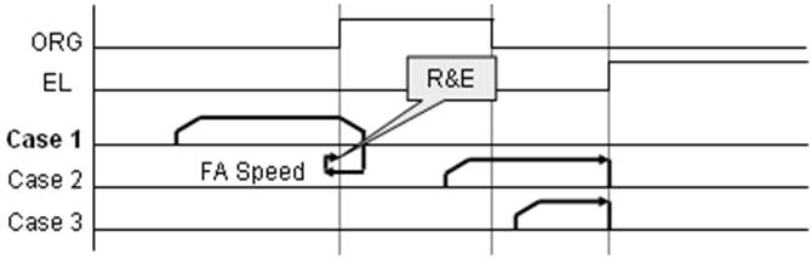
- SD 설치 되지 않은 경우



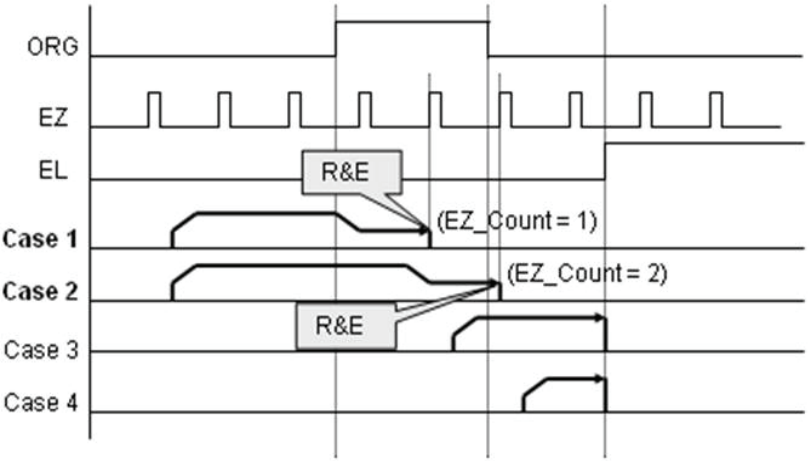
- SD 설치 되었고 latch 되지 않은 경우



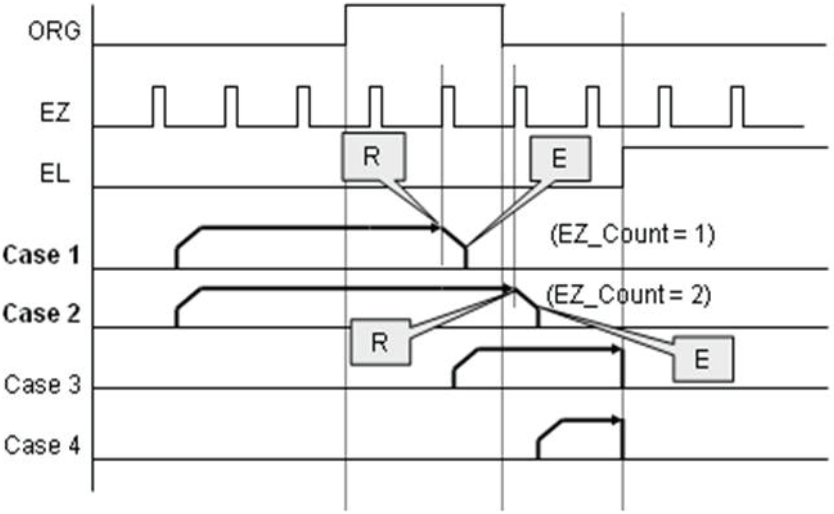
홈 모드=1 (두번 ORG ON 후 리셋 카운터)



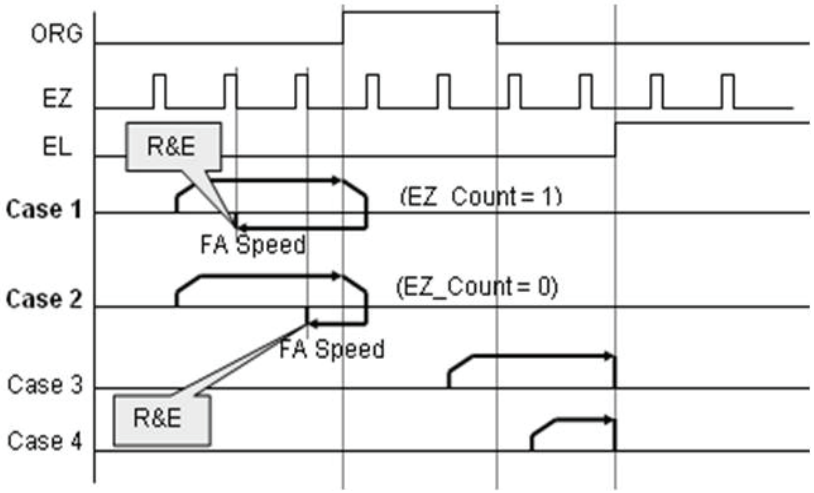
홈 모드=2 (ORG ON 후 EZ 수 카운트를 위한 감속 및 리셋 카운트)



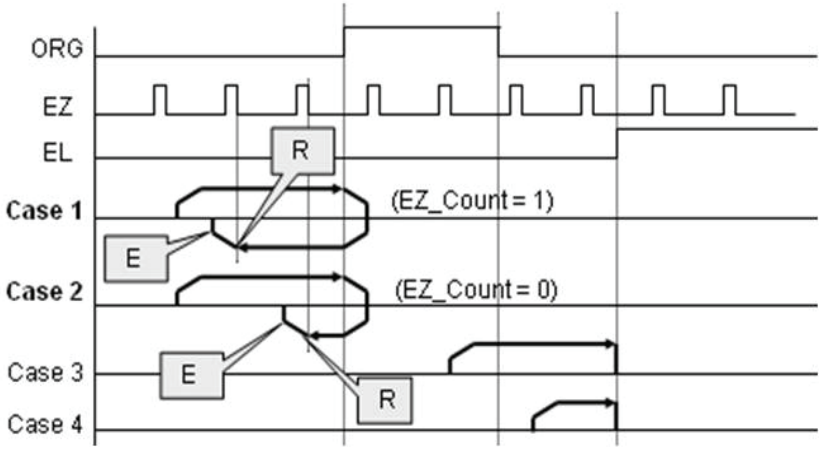
홈 모드=3 (ORG ON 후 EZ 수 카운트 및 리셋 카운터)



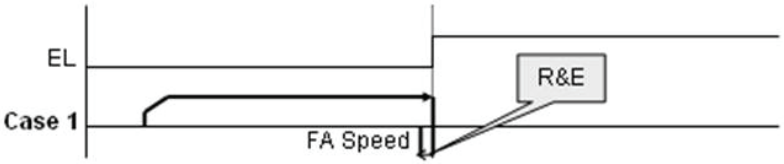
홈 모드=4 (ORG ON 후 EZ 수 카운트를 위한 역방향 전환 및 리셋 카운터)



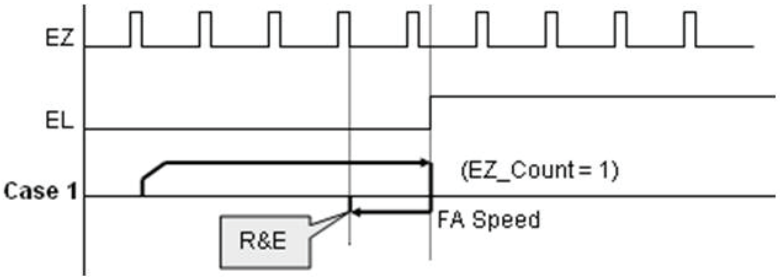
홈 모드=5 (ORG ON 후 EZ 수 카운트를 위한 역방향 및 리셋 카운터, FA 속도 사용 하지 않음)



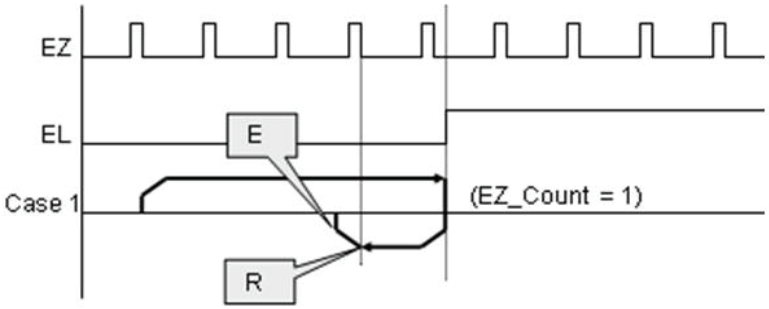
홈 모드=6 (EL ON 후 EL을 떠나기 위한 역방향 전환 및 리셋 카운터)



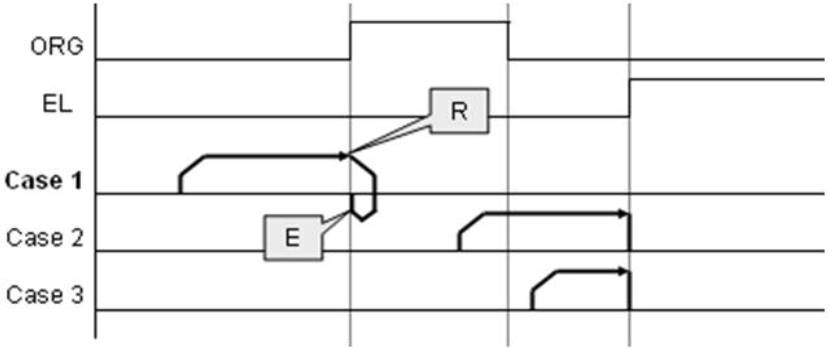
홈 모드=7 (EL ON 후 EL 카운트를 위한 역방향 전환 및 리셋 카운터)



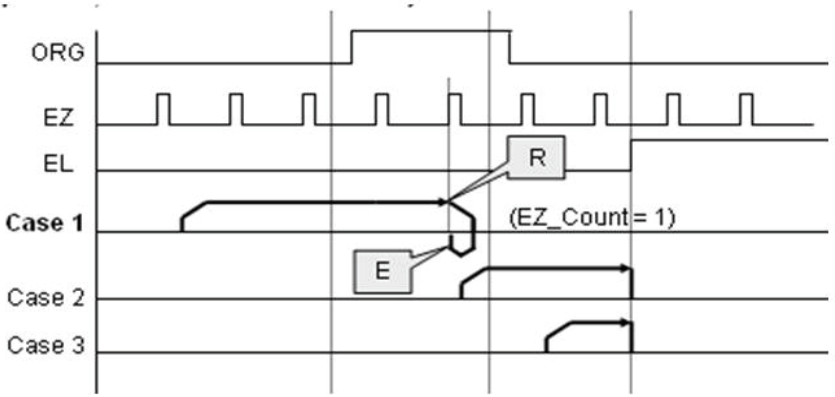
홈 모드=8 (EL ON 후 EL 카운트를 위한 역방향 전환 및 리셋 카운터, FA 속도 사용하지 않음)



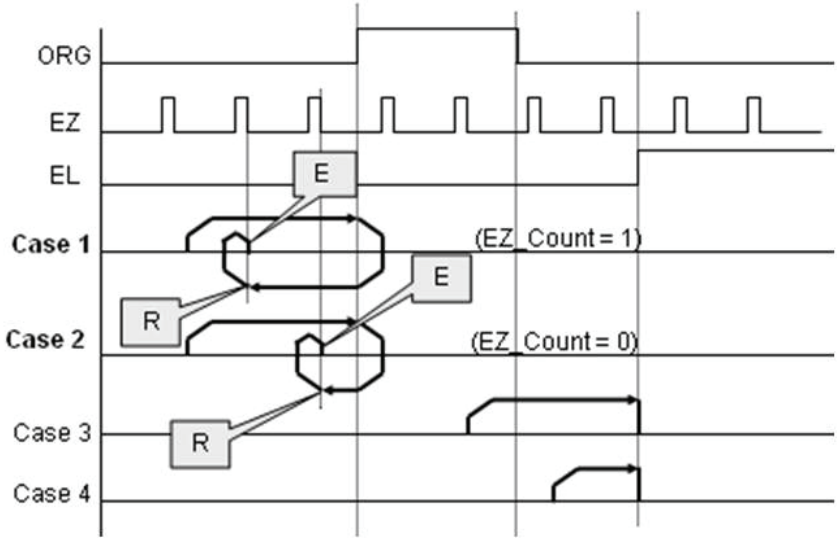
홈 모드=9 (ORG ON 후 영점을 향한 역방향 전환, 모드 0로 부터의 확장)



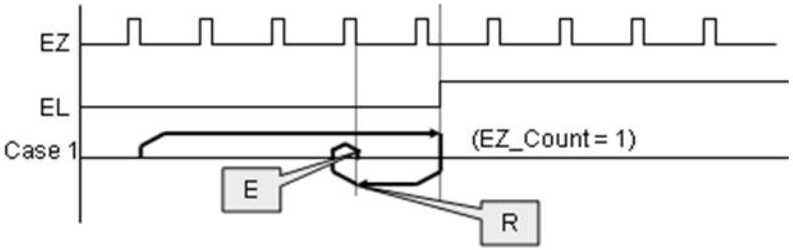
홈 모드=10 (ORG ON 후 EZ 카운트 및 리셋 카운터, 모드 2로 부터의 확장)



홈 모드=11 (ORG ON 후 EZ 카운트를 위한 역방향 전환 및 영점을 향한 역방향 전환, 모드 5로 부터의 확장)

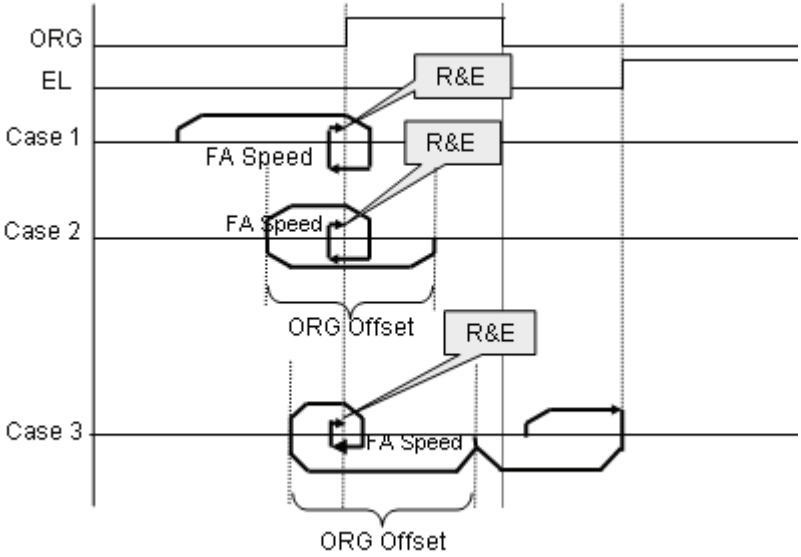


홈 모드=12 (EL ON 후 EZ 카운트를 위한 역방향 전환 및 영점을 향한 역방향 전환, 모드 8로 부터의 확장)



4.2.11 홈 검색 기능

이 기능은 축의 위치와 관계없이 이전에 언급된 보통의 홈 리턴 모드에서 자동검색 기능을 추가하기 위하여 사용됩니다. 다음의 그림은 홈 검색 기능을 통한 홈 모드 2 의 예를 보여줍니다. ORG 옵셋은 0 일 수 없으며 추천 값은 ORG 구간 길이의 두 배입니다.

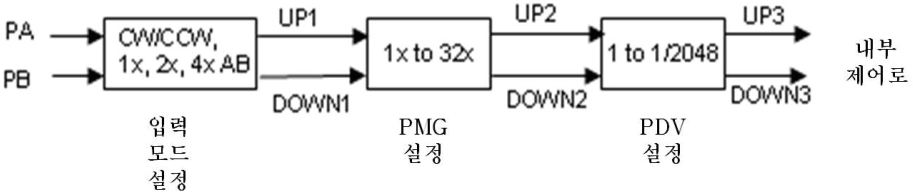


4.2.12 수동 펄스 기능

수동 펄스는 펄스 열을 수동으로 발생시키는 장치입니다. 발생된 펄스는 모션 제어기로 전송되고 펄스 출력 핀으로 재전송 됩니다. 입력 펄스는 전송되기 전 곱해지거나 나누어질 수 있습니다.

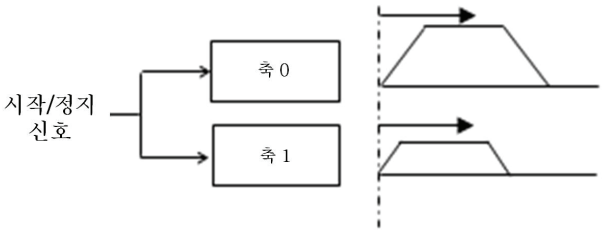
모션 제어기는 수동 펄스 장치로부터 CW/CCW 와 AB 위상의 두 가지 종류 펄스열을 받습니다. 만약 AB 위상 입력 모드가 선택된 경우 승수는 1, 2 또는 4 의 추가적인 선택요소를 가지게 됩니다.

다음 그림은 펄스 비율을 나타냅니다.



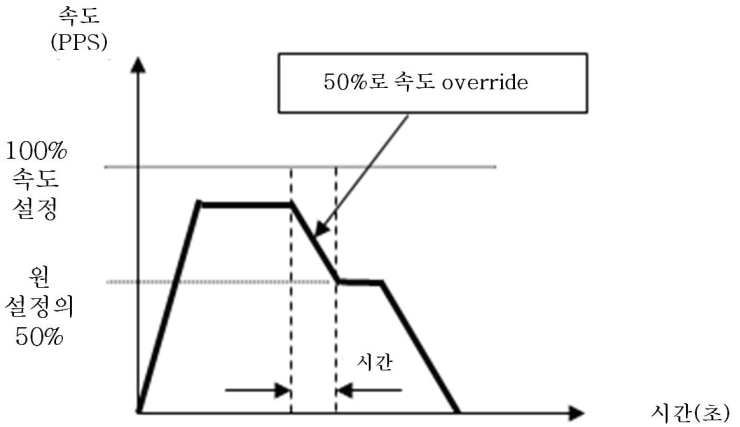
4.2.13 동시 시작 기능

동시 모션은 하나의 축 이상이 외부 또는 내부의 신호인 동시 신호로 시작될 수 있음을 의미합니다. 외부 신호를 위해 사용자는 반드시 모든 축에 대해 외부의 시작 / 정지 명령을 대기하게 하는 동작인 동작 파라미터를 먼저 설정해야 합니다. 내부 신호에 대해서는 소프트웨어 시작 함수가 시작 명령이 됩니다. 한번 이것이 호출되면 동기 모드에서 대기하고 있는 모든 축이 동시에 시작하게 됩니다.



4.2.14 속도 Override 기능

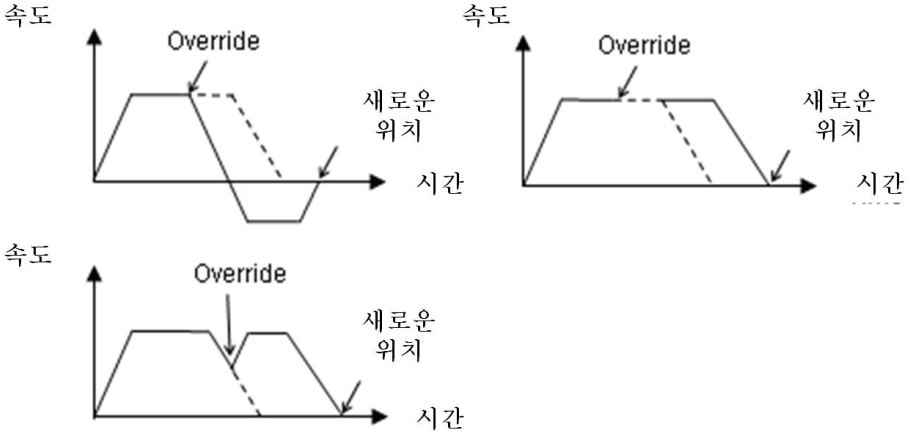
속도 override 는 사용자가 모션의 운영 동안 명령의 속도를 바꿀 수 있음을 의미합니다. 변환 파라미터는 원래 정의된 속도의 비율이며 사용자는 속도 값을 100% 로 정의함으로써 모션이 동작하는 동안 원 속도의 비율로 속도를 변환할 수 있습니다. 만약 사용자가 속도 값을 100% 로 정의하지 않았을 경우 , 속도 기본값의 100% 는 가장 최근 모션 명령의 최대 속도로 설정됩니다. 이 기능은 어떠한 모션 기능에도 적용될 수 있는데 만약 동작중인 모션이 S- 커브 또는 Bell 커브인 경우 속도 override 는 순수 S- 커브가 될 것이며 동작 중인 모션이 t- 커브 인 경우에는 속도 override 은 t- 커브가 됩니다.



4.2.15 위치 Override 기능

위치 override 는 사용자가 위치 명령을 내릴 때나 운영 중 위치 명령의 목표 위치의 변경을 원하는 경우를 의미합니다. 만약 새로운 목표 위치가 override 명령이 내려질 때의 위치보다 뒤에 위치할 경우 모터는 감속 할 것이며 새로운 위치 목표를 향해 반대로 움직이게 될 것입니다. 만약 새로운 목표 위치가 현재 위치로부터 멀리 떨어진 같은 방향의 위치일 경우 모터는 이 명령의 속도를 유지하며 새로운 목표 위치를 향해 동작할 것입니다. 만약 override 명령을 내릴 때 현재 모션이 감속 중이고 목표 위치가 현재 위치로부터 멀리 떨어진 같은 방향에 위치할 경우 모터는 원 속도를 향해 가속하며 새로운 목표 위치를 향해 동작할 것입니다. 운영 예시는 아래 보이는 그림과 같고 만약 새로운 목표 위치의 상대적인 펄스

가 원래의 감속하는 펄스보다 작을 경우 이 기능은 제대로 작동하지 않습니다.



4.3 모터 드라이버 인터페이스

본 제품은 각자 고유의 기능을 가지며 모터 드라이버에 직접 연결이 가능한 여러 가지 고유의 I/O 를 제공합니다. 모터 드라이버는 외부의 모션 제어기 사용을 위한 많은 종류의 입 / 출력 핀을 가지는데 우리는 이것을 두 가지 그룹으로 분류했습니다. 펄스의 입 / 출력 신호는 펄스 명령, 인코더 인터페이스를 포함하고 디지털 입 / 출력 신호는 서보 ON, 경보, INP, 서보 준비, 경보 재설정, 긴급 정지 입력을 포함합니다. 다음 내용에서는 이러한 입 / 출력 핀에 대해서 자세히 설명합니다.

4.3.1 펄스 명령 출력 인터페이스

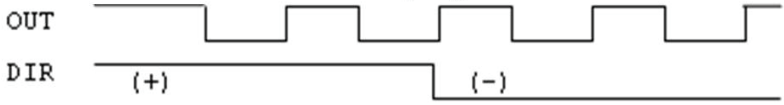
모션 제어기는 모터 드라이버를 통해 서보/ 스테퍼 모터를 제어하기 위하여 펄스 명령을 사용합니다. 드라이버를 펄스열을 위치 명령으로 인식하는 위치모드로 설정하십시오 . 펄스 명령은 커넥터 상의 OUT, DIR 핀의 두 가지 신호 쌍으로 구성되어 있습니다. 각 신호는 서로 다른 출력을 위해 두 개의 핀을 가지고 있습니다. 본 제품은 펄스 출력 명령을 위한 단일 펄스 출력 모드 (OUT/DIR) 과 이중 펄스 출력 모드 (CW/CCW 타입 펄스 출력) 의 두 개의 신호 모드를 제공합니다. 이 모드는 반드시 모터 드라이버의 모드와 같아야 하며 OUT 와 DIR 핀의 모드 대 신호 타입은 다음의 표와 같습니다.

단일 펄스 출력 (OUT/DIR 모드)

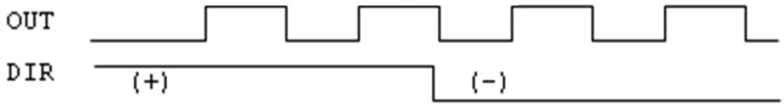
모드	OUT 핀의 출력	DIR 핀의 출력
이중 펄스 출력 (CW/CCW)	양의 방향 (또는 CW) 의 펄스 신호	음의 방향 (또는 CCW) 의 펄스 신호
단일 펄스 출력 (OUT/DIR)	펄스 신호	방향 신호 (level)

이 모드에서 OUT 핀은 명령 펄스 열의 출력을 보여주고 펄스열의 수는 펄스의 이동거리를 나타내며 펄스열의 주파수는 초당 움직이는 속도를 보여줍니다 . DIR 신호는 양의 방향 또는 음의 방향의 명령 방향을 나타냅니다 . 펄스열의 극성을 설정할 수도 있으며 아래의 그림은 출력 파형을 보여줍니다 .

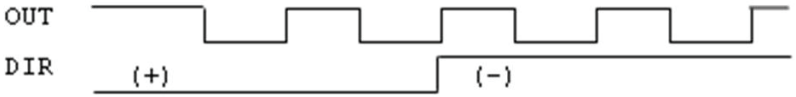
펄스 모드 = 0: (OUT 핀 normally high)



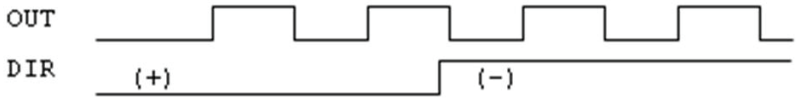
펄스 모드 = 1: (OUT 핀 normally low)



펄스 모드 = 2: (OUT 핀 normally high)



펄스 모드 = 3: (OUT 핀 normally low)

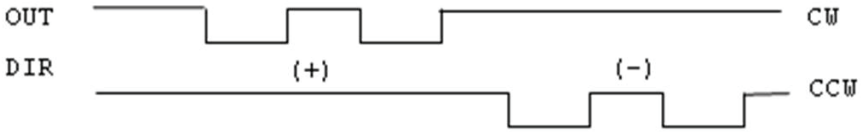


이중 펄스 출력 모드 (CW/CCW 모드)

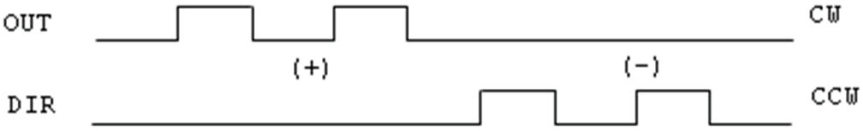
이 모드에서 OUT, DIR 핀의 파형은 CW (시계방향) 과 CCW (시계 반대방향) 각각의 펄스 출력을 나타냅니다 . 펄스열의 수는 펄스의 이동거리를 나타내고 펄스열의 주파수는 초당 움직이는 속도

를 보여줍니다 . CW 핀으로부터의 펄스 출력은 모터의 양의 방향 움직임을 만들고 , CCW 핀으로부터의 펄스 출력은 음의 방향으로의 이동을 만듭니다 . 아래의 그림은 양의 방향 명령과 음의 방향 명령 출력의 파형을 보여줍니다 .

펄스 모드 = 4: (펄스 normally high)



펄스 모드 = 5: (펄스 normally low)



명령 펄스는 28 비트 카운터로 카운팅되며 명령 카운터는 제어기로부터 출력되는 총 펄스 출력량의 값을 저장할 수 있습니다 .

4.3.2 펄스 피드백 입력 인터페이스

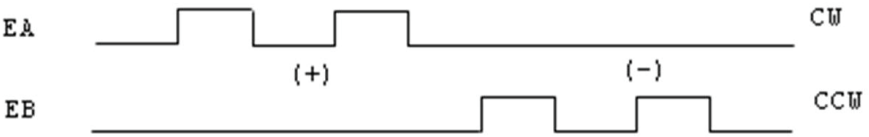
본 모션 제어기는 펄스 피드백 카운팅을 위해 각 축에 하나의 28 비트 up/down 카운터를 제공합니다 . 이 카운터는 위치 카운터라 불리며 차동 신호 입력을 위해 양 , 음의 핀을 가진 커넥터 상의 EA, EB 신호로부터의 펄스를 카운팅합니다 . 여기에는 이중 펄스 입력 (CW/CCW 모드) 와 AB 위상 입력의 두 가지 펄스 타입이 존재합니다 . AB 위상 입력은 1, 2, 4 의 승수이며 4 AB 위상 모드가 증분 인코더 입력에서 가장 일반적으로 사용됩니다 .

예를 들어 만약 로터리 인코더가 회전당 2000 펄스를 가진다면 위치 카운터로부터 판독되는 카운터 값은 4 AB 위상를 통해 4 배인 회전당 8000 펄스가 될 것입니다 .

만약 귀하가 모션 제어기를 위한 인코더를 사용하지 않는 경우 , 이 카운터를 위한 피드백 소스는 반드시 펄스 명령 출력으로 설정되어야 하며 카운터 값은 항상 0 이 되어야 합니다 . 만약 펄스 명령 출력으로 설정된 경우 사용자는 펄스 명령 출력 카운터로부터 위치 카운터 값을 얻을 수 있는데 그 이유는 피드백 펄스가 명령 출력 펄스로부터 내부 계산되기 때문입니다 .

다음 그림은 두 가지 타입의 펄스 피드백 신호를 보여줍니다 .

음, 양 펄스 입력 모드 (CW/CCW 모드)



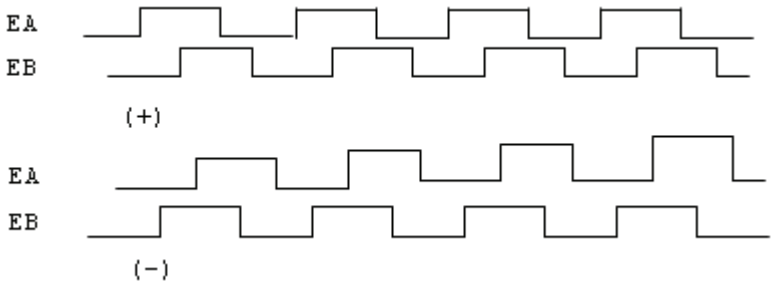
이 모드의 펄스 패턴은 입력 핀인 EA 와 EB 를 제외한 펄스 명령 출력 부분의 이중 펄스 출력 모드와 같습니다 .

이 모드에서 EA 핀으로부터의 펄스는 카운터가 count up 하는 원인인 반면 EB 핀으로부터의 펄스는 카운터가 count down 하는 원인이 됩니다 .

90° 위상 차 신호 입력 모드 (AB 위상 모드)

이 모드에서 EA 신호는 EB 신호에 비해 90° 위상이 선행하거나 후행하며 두 신호간 위상 차를 나타내는 'Lead' 와 'Lag' 는 모터의 방향 전환에 의하여 나타납니다 . Up/down 카운터의 EA 신호의 위상이 EB 신호의 위상보다 선행할 때 카운트를 하게 됩니다 .

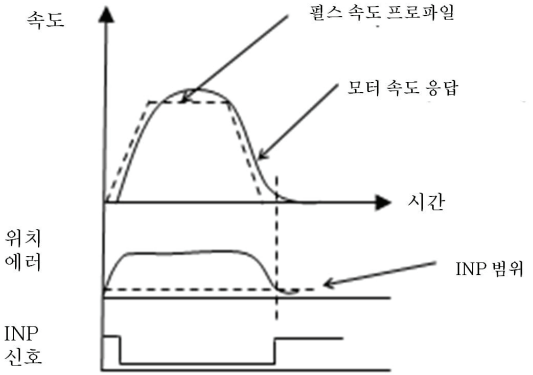
다음 그림은 파형을 보여주고 있습니다 .



인덱스 입력 (EZ) 신호는 선형 또는 로터리 인코더가 영점을 나타내는 것으로 간주하는데 메커니즘의 기계적 영점 위치를 정의하기 위해 사용될 수 있습니다 . 논리 신호는 반드시 정확한 결과를 얻을 수 있도록 정확히 설정하셔야 합니다 .

4.3.3 In position 신호

In-position 신호는 모터 드라이버로부터의 출력 신호이며 이것은 모션 제어기에 모터가 미리 지정된 에러와 함께 위치에 도달했음을 말해줍니다 . 미리 지정된 에러 값은 in-position 값이며 대부분의 모터 드라이버는 이를 INP 값이라 부릅니다 . 모션 제어기가 위치 명령을 발생한 후 , 모션 busy 상태는 INP 신호가 켜져 있는 동안 오차가 보정된 상태를 유지합니다 . 사용자는 모션 busy flag 를 체크하기 위한 INP 의 사용을 중단할 수 있는데 이 경우 모션 busy 는 펄스 명령이 모두 전송되었을 때 FALSE 가 될 수 있습니다 .



4.3.4 서보 경보 신호

경보 신호는 모터 드라이버부터의 출력 신호로서 이것은 모션 제어기에 서보모터 또는 드라이버 내부의 어떤 에러를 가지고 있음을 보고합니다. 한번 모션 제어기가 이 신호를 받으면 펄스 명령은 전송을 중단할 것이며 ALM 신호의 위상이 작동하게 됩니다. 알람의 원인은 서보모드의 과속, 과전류, 과부하와 같은 것이며 모터 드라이버의 매뉴얼을 자세히 확인하시기 바랍니다.

경보 신호의 논리는 반드시 정확히 설정되어야 합니다. 만약 경보 논리의 설정이 모터 드라이버의 설정과 동일하지 않은 경우 ALM 위상은 항상 켜져 있는 상태가 될 것이며 펄스 명령은 절대 출력되지 않을 것입니다.

4.3.5 에러 클리어 신호

ERC 신호는 모션 제어기로부터의 출력이며 이것은 에러 카운터를 클리어하기 위해 모터 드라이버에 보고합니다. 에러 카운터는 명령 펄스와 피드백 펄스 간 차이로부터 카운트 되며 피드백 위치는 항상 명령 위치에 시간적 지연을 가지고 있습니다. 그 결과 펄스는 매 순간 두 위치 간의 차이를 가지게 되며 그 차이는 에러 카운터에서 볼 수 있습니다. 모터 드라이버는 기초 제어 인덱스로 에러 카운터를 사용하며 큰 에러 카운터 값은 보다 빠른 모터 속도 명령이 설정될 것을 의미합니다. 만약 에러 카운터 값이 제로 (0)인 경우 제로 모터 속도 명령이 설정되어야 함을 의미합니다.

다음의 4 가지 상황에서 자동적으로 ERC 신호는 에러 카운터를 동시에 클리어하기 위해 모션 제어기로 출력됩니다.

1. 홈 리턴이 완료되었을 때
2. End-Limit 스위치가 활성화되었을 때
3. 경보 신호가 활성화되었을 때
4. 긴급 정지 명령이 활성화되었을 때

4.3.6 서보 ON/OFF 스위치

서보 ON/OFF 스위치는 모션 제어기의 일반적 디지털 출력 신호입니다. 이것은 커넥터 상의 SVON 핀과 같이 정의되며 모터 드라이버의 제어 상태를 변환하기 위해 사용됩니다. 전원이 켜지고 나면 드라이버의 제어 루프가 활성화되었기 때문에 모터는 가동될 것입니다. 축이 수직으로 설치되었으며 서보 신호가 꺼졌을 경우 축의 상태는 제어되지 않을 것이며 떨어질 수 있으므로 주의하십

시오 . 서보 경보와 긴급 신호가 켜지는 상황도 발생할 수 있습니다 .

4.3.7 서보 준비 신호

서보 준비 신호는 모션 제어기의 목적과는 연관성이 없는 모션 제어기의 일반적인 디지털 입력 신호입니다 . 사용자는 모터 드라이버가 준비 상황일 때 점검을 위해 이 신호를 모터 드라이버의 RDY 신호에 연결할 수 있습니다 . 이 신호는 예를 들어 모터 드라이버의 전원이 입력되었는지를 확인하는데 사용할 수 있습니다 . 또한 , 사용자는 이 핀을 일반적인 입력처럼 다른 목적을 위해 연결할 수 있으며 이는 모션 제어기에 어떠한 영향도 끼치지 않습니다 .

4.3.8 서보 경보 리셋 스위치

서보 드라이버는 서보 드라이버 내부에 어떠한 문제가 생겼을 경우 경보 신호를 발생시킵니다 . 경보 상황은 서보 모터의 과전류 , 과속 , 과부하 등의 상황을 포함하며 전원 리셋은 알람 상황을 정지할 수 있지만 , 사용자는 대부분 서보 모터가 운용 중일 때 전원 차단을 원치 않기 때문에 본 기계는 사용자의 알람 상황을 리셋하기 위한 서보 드라이버의 핀이 있습니다 . 본 모션 제어기는 각 축을 위한 하나의 일반적인 출력 핀을 제공하는데 사용자는 이 핀을 서보 알람 상황의 리셋을 위해 사용할 수 있습니다 .

4.4 기계적 스위치 인터페이스

본 회사는 원점 스위치 (ORG), 플러스와 마이너스의 End-Limit 스위치 (±EL), slow down 스위치 (SD), 포지셔닝 시작 스위치 (PCS), 카운터 latch 스위치 (CLR) 와 같은 기계적 스위치를 위한 전용 입력 핀을 제공하며 이러한 스위치들은 아주 작은 ASIC 클럭 사이클을 소비하는 빠른 응답 시간을 가집니다 . 이러한 신호들을 사용할 시 실시간 처리 문제가 존재하지 않으며 모든 기능은 모션 ASCI 에 의해 수행되므로 소프트웨어는 단지 결과를 확인하는 작업만 필요합니다 .

4.4.1 원점 또는 홈 신호

본 제어기는 각 축에 대해 하나의 원점 또는 홈 신호를 제공합니다 . 이 신호는 축의 영점을 정의하기 위해 사용되며 귀환 동작을 하기 전에 신호의 논리는 반드시 제대로 설정되어야 합니다 . 귀환 모드의 자세한 설명은 귀환 모드 부분을 참고하시기 바랍니다 .

4.4.2 End-Limit 스위치 신호

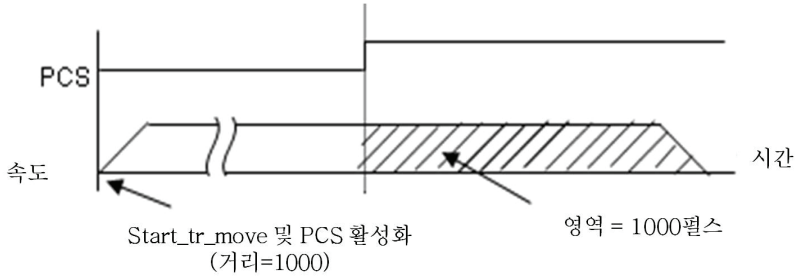
End-Limit 스위치는 일반적으로 축의 양 끝에 설치되며 축의 양의 방향에는 +EL 을 음의 방향에는 -EL 을 설치합니다 . 이 두 신호는 안전을 위함이며 만약 두 신호가 반대로 설치되었을 경우에는 End-Limit 는 제대로 동작하지 않을것 입니다 . 모터가 동작 중일 시 End-Limit 신호를 건드리면 모션 제어기는 ERC 신호 출력과 펄스의 출력을 멈출 것입니다 . 이것은 오작동 시의 기계 충돌을 방지할 수 있습니다 .

4.4.3 Slow down 스위치

Slow Down 은 활성화되었을 때 명령 펄스를 시작 속도로 감속시키는데 사용되며 매우 빠른 속도로 기구의 끝 부분을 향해 움직일 시 발생하는 충돌을 방지하는데 사용됩니다 . SD 신호는 양의 방향과 음의 방향 모두 사용할 수 있습니다 .

4.4.4 포지셔닝 시작 스위치

포지셔닝 시작 스위치는 켜졌을 경우 정해진 위치로 이동하는데 사용되며 기능은 다음과 같습니다 .



4.4.5 카운터 클리어 스위치

카운터 클리어 스위치는 모션 제어기의 카운터를 리셋 하기 위한 입력 신호로서 사용자가 외부의 명령에 따라 카운터의 리셋이 필요한 경우 이 핀을 사용하여 제어할 수 있습니다.

4.4.6 카운터 Latch 스위치

카운터 Latch 스위치는 입력이 활성화되었을 때 카운터 값을 레지스터로 유지하도록 하는 입력 신호이며 사용자가 입력 활성화 시간의 카운터 값을 알고 싶을 때 이 핀을 연결함으로서 알 수 있습니다.

4.4.7 긴급 정지 입력

본 모션 제어기는 긴급 상황을 위한 글로벌 디지털 입력을 제공합니다. 한번 입력이 가동되면 본 모션 제어기는 기계의 손상을 막기 위해 축의 모든 모션을 즉시 정지할 것이며 일반적으로 사용자는 긴급 정지 버튼을 기계 상의 입력으로 연결할 수 있습니다. 본 회사는 안전을 위해 평상시 닫힘 상태로 할 것을 제안합니다.

4.5 카운터

본 기계에는 모션 제어기의 각 축에 4 개의 카운터가 존재합니다. 다음 장에서는 이에 대해 자세히 설명합니다.

- 명령 위치 카운터 : 출력 펄스의 수를 카운터
- 피드백 위치 카운터 : 입력 펄스의 수를 카운터
- 위치 에러 카운터 : 명령과 피드백 펄스 수 간의 에러를 카운터
- 범용 목적 카운터: 소스는 명령위치, 피드백 위치, 수동 펄스 또는 ASIC 클럭의 절반으로 구성.
- 목표 위치 리코더 : 최근 모션 명령의 목표 위치 값을 유지

4.5.1 명령 위치 카운터

명령 위치 카운터는 28 비트의 2 진법 up/down 카운터입니다. 입력 소스는 모션 제어기부터의 출력 펄스이며 현재의 명령 위치의 정보를 제공할 것입니다. 이것은 모션 시스템의 문제를 수정하는데 유용합니다.

본 모션 시스템은 오픈 루프 타입입니다. 모터 드라이버는 모션 제어기로부터 펄스를 전송받고 드라이버는 모터를 움직입니다. 드라이버가 움직이지 않을경우 명령 카운터를 확인하고 새로운 값이 있는지 확인하십시오. 새로운 값이 있는 경우 펄스가 문제점을 전송하였으며 모터 드라이버에 문제가 있다는 의미이므로 모터 드라이버의 펄스 수신 카운터를 확인하시기 바랍니다.

명령 카운터의 단위는 펄스이며 카운터 값은 카운터 클리어 신호 또는 홈 기능의 활성화를 통해 리셋할 수 있습니다. 사용자는 또한 소프트웨어 명령 카운터 설정 기능으로 이를 리셋 할 수 있습니다.

4.5.2 피드백 위치 카운터

피드백 위치 카운터는 28 비트의 2 진법 up/down 카운터입니다. 입력 소스는 EA/EB 핀으로부터의 입력 펄스이며 모터의 인코더 출력으로부터의 모터 위치를 카운트합니다. 카운터는 외부 인코더 입력이 없을 경우 옵션에 대한 명령 위치의 소스로부터 설정될 것입니다.

명령 출력 펄스와 피드백 입력 펄스는 항상 같은 미니미터의 비율을 가지지 않으며 사용자는 두 펄스가 1:1 이 아닌 경우 반드시 이를 설정하셔야 합니다.

본 모션 제어기는 closed loop 타입이 아니므로 피드백 위치 카운터는 모션이 동작한 후에 단순한 참고용으로 쓰입니다. closed loop 위치는 서보 모터 드라이버에 의해 작동하며 서보 드라이버가 잘 조율 되었거나 기계적 부분이 잘 조합 된 경우에는 모션 명령 후 전체 위치 에러가 수용 가능한 범위 내에서 완료될 것입니다.

4.5.3 명령 및 피드백 에러 카운터

명령과 피드백 에러 카운터는 명령 위치와 피드백 위치 간의 에러를 계산하는데 사용됩니다. 값은 명령 위치에서 피드백 위치를 뺀 값입니다.

만약 명령과 피드백 간의 비율이 1:1 이 아닌 경우 에러 카운트는 아무런 의미가 없습니다 .

이 카운터는 16 비트의 2 진법 up/down 카운터입니다 .

4.5.4 범용 카운터

일반적인 범용 카운터의 소스는 다음과 같습니다 .

1. 명령 위치 출력 - 명령 위치 카운터와 같음
2. 피드백 위치 입력 - 피드백 위치 카운터와 같음
3. 수동 펄스 입력 - 기본 설정
4. 클럭 틱 - 대략 9.8MHz 타이머에서의 카운터

4.5.5 목표 위치 리코더

목표 위치 리코더는 목표 위치 정보를 제공하는데 사용되며 연속적 모션에서 사용되는데 그 이유는 모션 제어기는 현재 명령에 연관된 펄스를 계산하여 그 결과를 Pre-register 로 전송하기 위해 이전의 모션 명령의 목표 위치와 현재 모션 명령의 목표 위치를 알아야 하기 때문입니다 . 연속적 모션 이전에 목표 위치와 현재 명령 위치가 같은지 확인하여 주시고 특별히 홈 기능과 정지 기능 이후에 반드시 확인하여 주십시오 .

4.6 콤퍼레이터 (Comparators)

본 기계는 각 축에 5 가지 카운터 콤퍼레이터를 갖추고 있습니다. 각 콤퍼레이터는 아래와 같은 기능이 있습니다.

1. 명령 카운터를 위한 Positive soft end-limit 콤퍼레이터
2. 명령 카운터를 위한 Negative soft end-limit 콤퍼레이터
3. 명령, 피드백 에러 카운터 콤퍼레이터
4. 모든 카운터를 위한 일반적 콤퍼레이터
5. 모든 명령, 피드백 카운터를 위한 트리거 콤퍼레이터

4.6.1 Soft end-limit 콤퍼레이터

본 기계는 각 축의 end-limit 기능을 위한 두 가지 콤퍼레이터를 갖추고 있으며 이는 soft end-limit 콤퍼레이터라 불립니다. 이 중 하나는 + 또는 Positive end-limit 를 위한 것이고 다른 하나는 - 또는 Negative end-limit 를 위한 것입니다. 이 end-limit 는 초과 이동 시 기계의 충돌을 방지하기 위하여 사용되고 real end-limit 스위치 대신 soft end-limit 를 사용할 수 있습니다. 이 두 가지 콤퍼레이터는 단지 명령 위치 카운터를 비교하기 위한 것이며 한번 명령 위치가 positive 또는 negative 의 콤퍼레이터 내부에서 설정된 한계를 초과할 시 이는 end-limit 스위치를 누른 것과 마찬가지로 작동이 정지됩니다.

4.6.2 명령 및 피드백 에러 카운터 콤퍼레이터

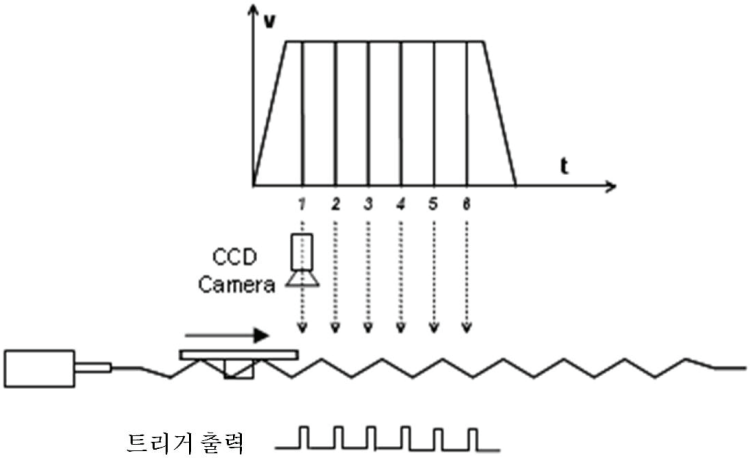
콤퍼레이터는 오직 명령, 피드백 카운터 에러를 위한 것이며 사용자는 에러의 값이 너무 큰 경우 이를 확인하기 위하여 사용할 수 있습니다. 이 콤퍼레이터는 조건이 맞을 시 동작의 설정이 가능하며 인터럽트 발생, 즉시 정지, 정지를 위한 감속이 이 동작에 포함됩니다.

4.6.3 일반 콤퍼레이터

일반 콤퍼레이터는 사용자가 비교할 소스를 선택할 수 있으며 이는 명령, 피드백 위치 카운터, 에러 카운터 또는 일반적 카운터로부터 선택할 수 있습니다. 비교방법은 방향의 유무와 함께 동등 또는 보다 큰, 보다 작은 으로 선택 가능하며 조건이 충족되었을 시의 동작으로 인터럽트 발생, 정지 모션 등을 선택할 수 있습니다.

4.6.4 트리거 콤퍼레이터

트리거 콤퍼레이터는 일반적 콤퍼레이터와 유사합니다 . 이는 추가 기능을 가지고 있는데 조건이 충족되었을 때 트리거 펄스를 발생시킵니다 . 조건이 충족될 경우 커넥터 상의 CMP 핀은 사진을 찍을 때의 카메라와 같은 특정한 목적의 펄스를 출력하게 됩니다 . 모든 축이 이 기능을 가지고 있는 것은 아니며 축의 CMP 핀의 존재에 따라 다릅니다 . 아래의 그림은 트리깅의 응용을 보여줍니다 ,



이 응용 중에서 테이블은 모션 명령에 따라 제어되고 CCD 카메라는 CMP 핀에 의해 제어됩니다 . 비교위치에 도달 했을 경우 펄스는 출력되고 이미지는 캡쳐 될 것이며 이는 즉시 이미지 캡쳐 입니다 . 사용자가 모션의 경로 중 더 많은 이미지를 얻기를 원하는 경우 이전의 이미지가 캡쳐 된 바로 직후에 새로운 비교 지점을 설정할 수 있습니다 . 이 방법을 통해 뛰어난 연속 이미지 캡쳐링 기술을 완성할 수 있었습니다 .

4.7 다른 모션 기능들

본사는 모션 제어기 상의 backlash 보정 , slip 보정 , 진동 억제 (vibration restriction), 속도 프로파일 계산 등의 다양한 다른 기능 들을 제공합니다 . 다음의 내용은 이를 자세히 설명합니다 .

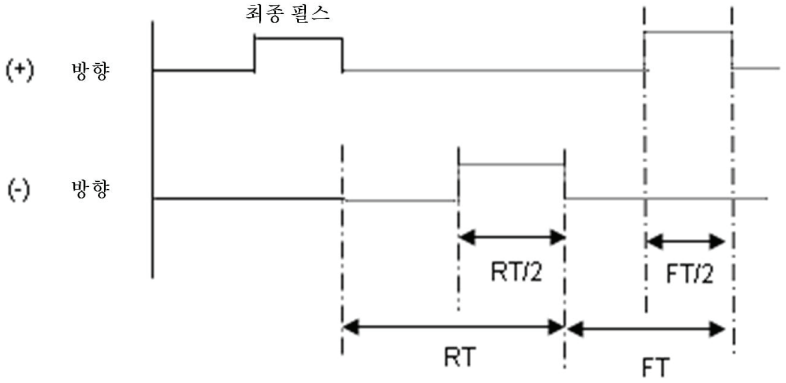
4.7.1 Backlash 와 slip 보정

본 모션 제어기는 backlash 와 slip 보정 기능을 가지고 있습니다 . 이 기능들은 FA 속도 내 명령 펄스의 수를 출력합니다 . Back-

lash 보정은 동작 중 방향 전환 시 작동하고 slip 보정 기능은 모션 명령 전에 작동합니다. 방향에 관계없이 보정 펄스의 양은 함수 라이브러리를 통해 설정됩니다.

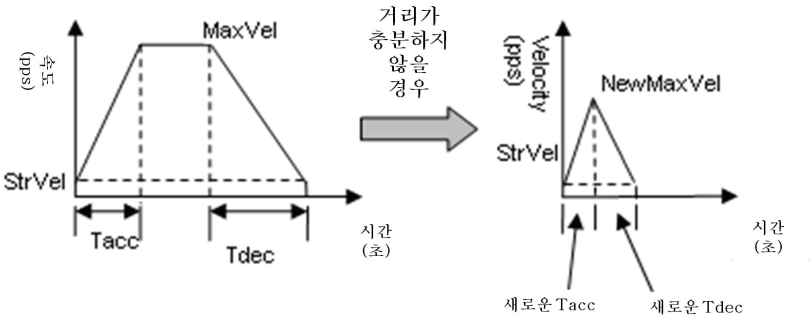
4.7.2 진동 억제 (Vibration restriction) 기능

모션 제어기의 진동 억제 (vibration restriction) 방법은 양의 방향 펄스가 모션 명령을 마친 후 바로 반대 방향의 펄스를 추가하는 것입니다. 두 가짜 펄스의 타이밍은 다음과 같습니다. (RT 는 역행 타임, FT 는 전진 타임)



4.7.3 속도 프로파일 연산 기능

본 모션 기능은 사용자로부터의 몇 가지 속도 파라미터가 필요합니다. 몇몇 파라미터들은 속도 프로파일과 충돌을 일으킵니다. 예를 들어 사용자가 아주 빠른 속도를 입력하고 아주 짧은 거리를 모션 기능에 입력한다면 속도 프로파일은 존재하지 않을 것입니다. 이러한 상황에서는 모션 라이브러리는 가속과 감속율을 유지할 것이며 사용자가 자동적으로 실현 가능한 속도 프로파일로 수정할 때까지 최대 속도보다 낮은 속도를 유지할 것입니다. 다음 그림은 이 개념을 보여줍니다.



본 모션 라이브러리는 사용자로부터 명령의 실제 속도 프로필을 알아내기 위한 함수열을 내장하고 있습니다.

4.8 인터럽트 제어

본 모션 제어기는 주 PC 에 인터럽트 신호를 발생시킬 수 있습니다. 이것은 event-driven 소프트웨어 응용에 굉장히 유용하며 사용자는 _8158_int_control() 기능을 사용하여 인터럽트 서비스를 활성화 또는 비활성화 할 수 있습니다.

PCI-8158 에는 모션 인터럽트 소스, 에러 인터럽트 소스, GPIO 인터럽트 소스의 3 가지 종류의 인터럽트 소스가 있으며 모션과 GPIO 인터럽트 소스는 마스크가 가능하지만 에러 인터럽트 소스는 불가능합니다. 모션 인터럽트 소스는 _8158_set_motion_int_factor() 에 의해 마스크가 가능하며 마스크 비트는 다음의 표와 같습니다.

모션 인터럽트 소스 비트 설정

비트	설 명
0	정상적 정지
1	버퍼 시작에서의 다음 명령
2	명령 pre- 레지스터 2 비어있음 , 새로운 명령 입력 허용
3	0
4	가속 시작
5	가속 정지
6	감속 시작
7	감속 정지
8	+ Soft limit 또는 콤퍼레이터 1 활성화
9	-Soft limit 또는 콤퍼레이터 2 활성화
10	에러 콤퍼레이터 또는 콤퍼레이터 3 활성화
11	일반적 콤퍼레이터 또는 콤퍼레이터 4 활성화
12	트리거 콤퍼레이터 또는 콤퍼레이터 활성화
13	CLR 입력에 의한 카운터 재설정
14	LTC 입력에 의한 카운터 Latch
15	ORG 입력에 의한 카운터 Latch
16	SD 입력 활성화
17	0
18	0
19	CSTA 입력 또는 _8158_start_move_all() 활성화
20-31	0

Table 4-1: 모션 인터럽트 소스 비트 설정

에러 인터럽트 소스는 마스크 할 수 없지만 타임아웃된 상황이 아니라면 상황의 에러 번호는 _8158_wait_error_interrupt() 의 리턴 코드에서 얻을 수 있습니다 .

에러 인터럽트 리턴 코드

값	설 명
0	+ Soft Limit 활성화 , 축 정지
1	-Soft Limit 활성화 , 축 정지
2	콤퍼레이터 3 활성화 , 축 정지
3	일반적 콤퍼레이터 또는 콤퍼레이터 4 활성화 , 축 정지
4	트리거 콤퍼레이터 또는 콤퍼레이터 4 활성화 , 축 정지
5	+ End Limit 활성화 , 축 정지
6	-End Limit 활성화 , 축 정지
7	ALM 활성화 , 축 정지
8	CSTP 활성화 또는 _8158_stop_move_all 활성화 , 축 정지
9	CEMG 활성화 , 축 정지
10	SD 입력 활성화 , 축 정지를 위한 감속
11	0
12	보간 동작 에러와 정지
13	다른 축의 에러에 의한 정지로 인해 축 정지
14	펄스 입력 버퍼 오버플로와 정지
15	보간 카운터 오버플로
16	인코더 입력 신호 에러 , 축 정지하지 않음
17	펄스 입력 신호 에러 , 축 정지하지 않음
11-31	0

Table 4-2: 에러 인터럽트 리턴 코드

GPIO 인터럽트 소스는 마스크 가능하며 마스크 비트 표는 아래와 같습니다 .

GPIO 인터럽트 소스 비트 설정 (1= 활성화 ,0= 비활성화)

비트	설 명
0	DI0 하강 엣지
1	DI1 하강 엣지
2	DI2 하강 엣지
3	DI3 하강 엣지
4	DI0 상승 엣지
5	DI1 상승 엣지
6	DI2 상승 엣지
7	DI3 상승 엣지
8	핀 23 입력 인터럽트
9	핀 57 입력 인터럽트
10	핀 23/57 인터럽트 모드 선택 (0= 하강 , 1= 상승)
11-14	0
15	GPIO 인터럽트 스위치 (항상 =1)

Table 4-3: GPIO 인터럽트 소스 비트 설정

다음은 인터럽트를 사용하기 위한 과정을 보여줍니다 .

1. _8158_int_control(CARD_ID, 활성화 =1/ 비활성화 =0) 사용
2. 이벤트 또는 GPIO 인터럽트를 위한 인터럽트 소스 설정
3. _8158_set_motion_int_facor(AXIS0, 0x01); // 축 0 정상적 정지
4. _8158_set_gpio_int_factor(CARD0, 0x01); // DI0 하강 엣지
5. _8158_wait_motion_interrupt(AXIS0, 0x01, 1000) // 정상적 정지 인터럽트를 위해 1000ms 대기
6. _8158_wait_gpio_interrupt(CARD0, 0x01, 1000) // D10 하강 엣지 인터럽트를 위해 1000ms 대기
7. I16 ErrNo=_8158_wait_error_interrupt(AXIS0, 2000); // 에러 인터럽트를 위해 2000ms 대기

4.9 여러 장의 카드 동작

PCI-8158 모션제어기는 한 시스템에 한 장 이상의 제어카드를 장착할 수 있습니다. 또한, 모션 제어기는 plug-and-play 특성을 가지고 있어 시스템 부팅 시에 기본 주소와 IRQ 설정이 PCI BIOS 에 자동으로 할당되므로 사용자는 리소스 관련 설정을 할 필요가 없습니다.

시스템에 여러 장의 카드를 사용할 시 카드의 번호는 반드시 기억해야 하며 카드의 번호는 보드의 카드 ID 스위치 설정으로 정해집니다. 축의 번호는 카드의 ID 에 의해 정해지게 되며 예를 들어 3 장의 모션 제어 카드를 PCI 슬롯에 장착하면 해당 카드 ID 가 설정됩니다. 각 카드의 축 번호는 다음과 같습니다.

축 / No. / 카드 ID	X	Y	Z	U	A	B	C	D
0	0	1	2	3	4	5	6	7
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31

만약 여러 장의 카드를 똑같은 카드 ID 로 설정한다면 본 제품은 정상적으로 작동 되지 않습니다.

5 MotionCreatorPro

하드웨어 설치 후 (2, 3 장), 모든 카드와 시스템이 올바르게 설정되고 작동하는지 확인해야 합니다 . 이 장에서는 제어 시스템을 구성하고 PCI-8158 의 카드가 정확히 동작하는지를 확인하기 위한 지침을 제공합니다 . MotionCreatorPro 소프트웨어는 PCI-8158 카드를 사용하는데 있어 간단히 설정 , 구성 , 테스트 , 문제 해결을 하는 것을 도와줍니다 .

MotionCreatorPro 는 오직 윈도우즈 2000/XP 에서의 1024x768 이상의 화면 해상도에서만 사용이 가능하며 1024x768 의 화면 해상도를 추천합니다 . 도스 환경에서는 사용하실 수 없습니다 .

5.1 MotionCreatorPro 의 실행

윈도우즈 2000/XP 에 PCI-8158 의 소프트웨어 드라이버 설치가 완료된 후에 MotionCreatorPro 은 < 선택된 경로 >₩PCI-Motion₩ MotionCreatorPro 에 위치 하게 됩니다 . 실행하시기 위해서는 실행 파일을 더블클릭하시거나 시작 > 프로그램 파일 > PCI Motion>₩PCI-Motion₩MotionCreatorPro 를 사용하실 수 있습니다 .

5.2 MotionCreatorPro 에 관하여

MotionCreatorPro 를 실행하기 전에 몇 가지 알아두셔야 할 사항이 있습니다 .

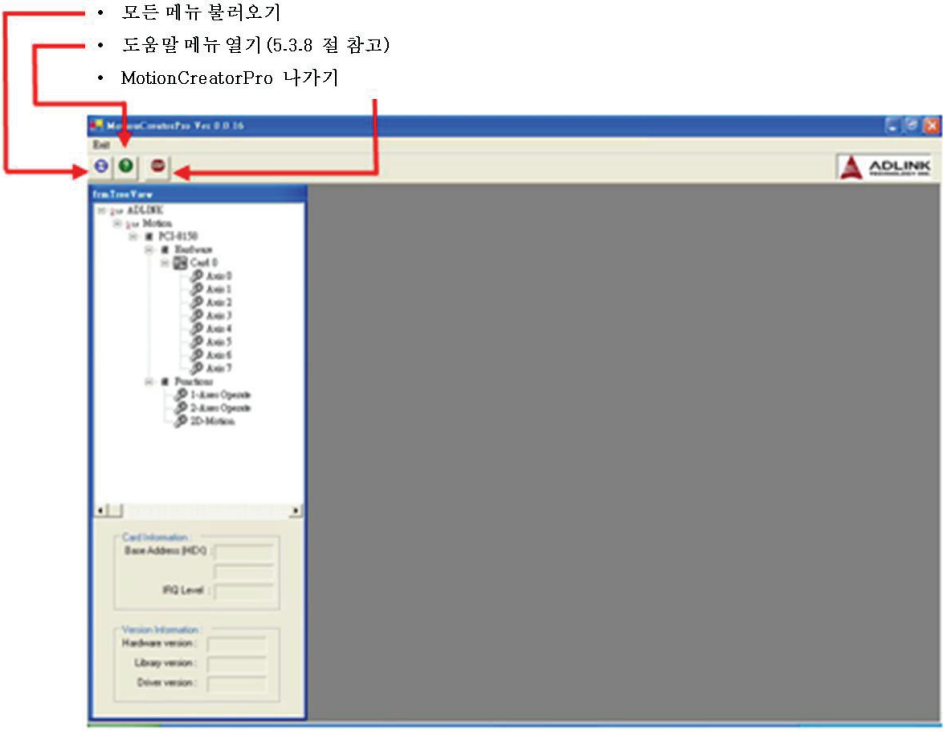
1. MotionCreatorPro는 VB.NET 2003에서 만들어졌으며 윈도우즈 2000/XP 에서 1024x768 이상의 화면 해상도에서만 작동가능 합니다 . 도스에서는 사용할 수 없습니다 .
2. MotionCreatorPro 는 사용자가 설정한 값을 저장할 수 있고 저장된 설정 값은 다음에 MotionCreatorPro 가 실행 될때 자동으로 불러오게 됩니다 . 8158.ini 과 8158MC.ini 파일은 윈도우 루트 폴더에 저장됩니다 .
3. 한 시스템과 동일한 구성을 다른 시스템에 적용하기를 원하실 경우에는 윈도우 루트 폴더에 있는 8158.ini 과 8158MC.ini 파일을 복사하여 넣으시면 됩니다 .
4. 만약 여러 장의 PCI-8158 카드를 동일한 MotionCreatorPro 로 저장된 설정 값으로 사용하실 경우 사용자가 개발한 프로그램에서 DLL 함수 _8158_config_from_file() 를 호출하여 사용하시면 되고 이는 도스 환경에서도 가능합니다 .

5.3 MotionCreatorPro 형태 소개

5.3.1 주메뉴

주메뉴는 MotionCreatorPro 를 처음 실행했을 때 나타나며 다음과 같습니다 .

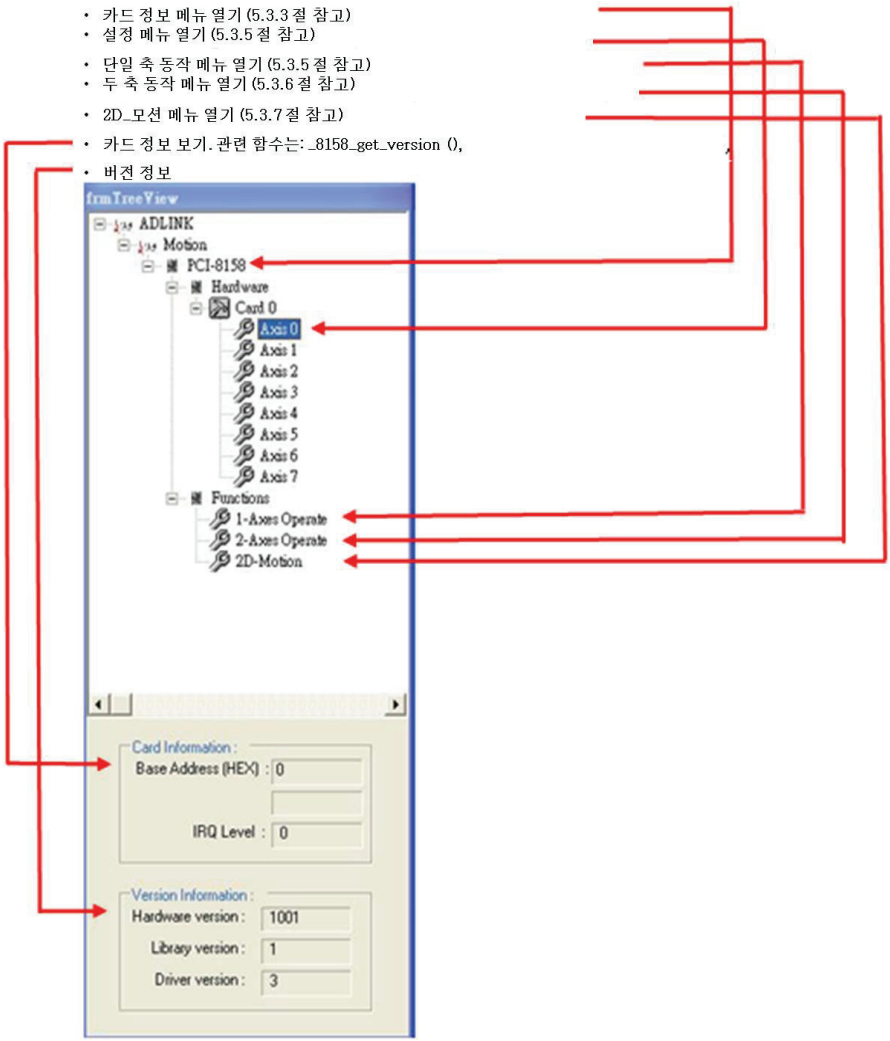
- 모든 메뉴 불러오기
- 도움말 메뉴 열기 (5.3.8 절 참고)
- MotionCreatorPro 나가기



5.3.2 선택 메뉴

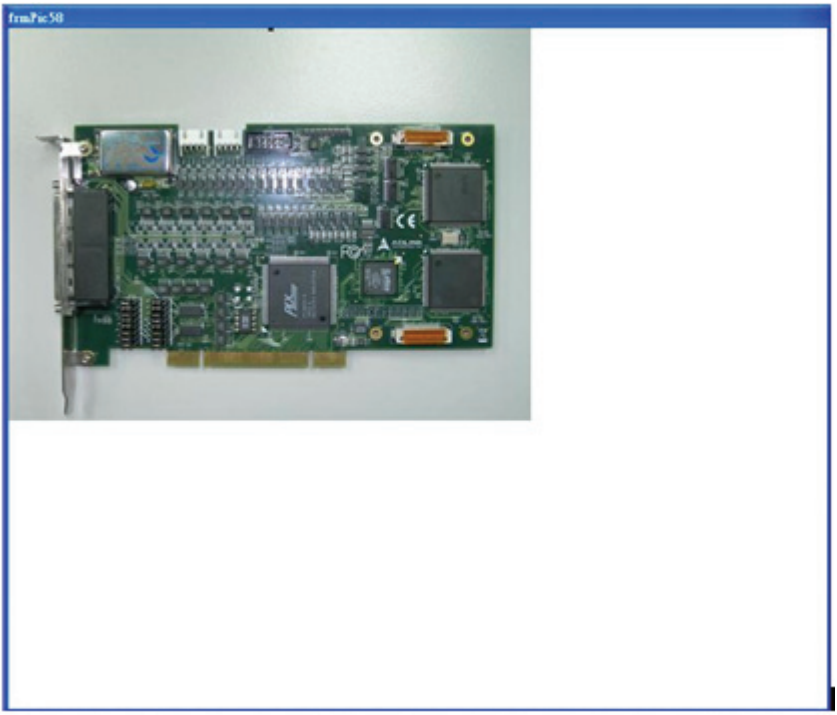
선택 메뉴는 MotionCreatorPro 실행 후 나타나고 다음과 같습니다.

- 동작 카드 및 축 선택
- 카드 정보 메뉴 열기 (5.3.3 절 참고)
- 설정 메뉴 열기 (5.3.5 절 참고)
- 단일 축 동작 메뉴 열기 (5.3.5 절 참고)
- 두 축 동작 메뉴 열기 (5.3.6 절 참고)
- 2D_모션 메뉴 열기 (5.3.7 절 참고)
- 카드 정보 보기. 관련 함수는: _8158_get_version (),
- 버전 정보



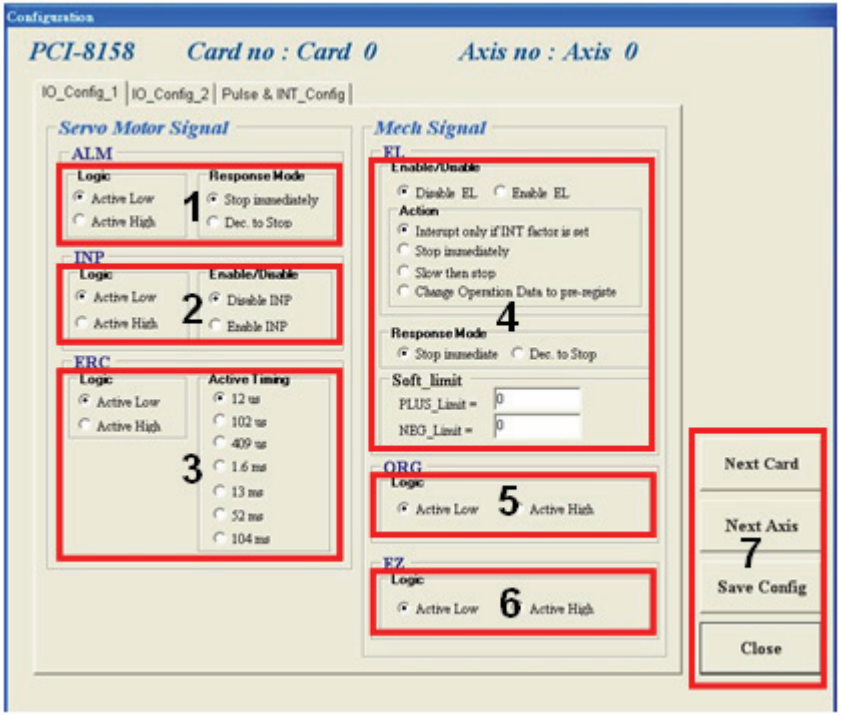
5.3.3 카드 정보 메뉴

이 메뉴는 카드에 대한 정보를 보여줍니다.



5.3.4 설정 메뉴

이 메뉴에서는 ALM, INP, ERC, EL, ORG, EZ 과 같은 설정을 할 수 있습니다 .

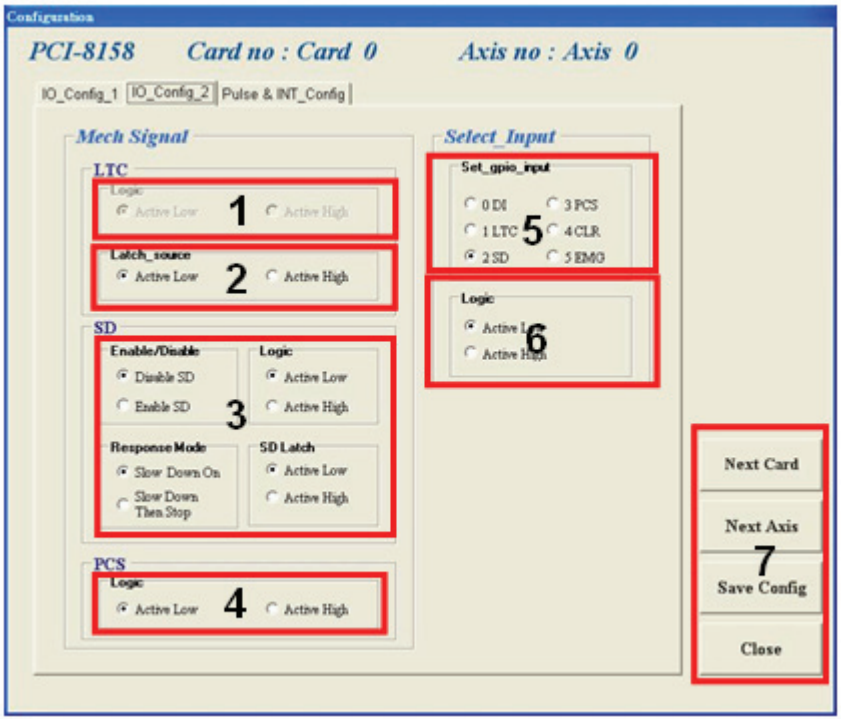


1. **ALM 논리와 응답모드** : ALM 신호의 논리 및 응답모드 선택 . 관련된 함수의 호출은 _8158_set_alm() 입니다 .
2. **INP 논리 및 활성화/비활성화 선택** : INP 신호의 논리 및 활성화 / 비활성화 선택 . 관련된 함수의 호출은 _8158_set_inp() 입니다 .
3. **ERC 논리와 활성화 타이밍** : ERC 신호의 논리 및 활성화 타이밍 선택 . 관련된 함수의 호출은 _8158_set_erc() 입니다 .
4. **EL 응답 모드** : EL 신호의 응답 모드 선택 . 관련된 함수의 호출은 _8158_set_limit_logic () 입니다 .
5. **ORG 논리** : ORG 신호의 논리 선택 . 관련된 함수의 호출

은 _8158_set_home_config() 입니다 .

6. **EZ 논리** : EZ 신호의 논리 선택 . 관련된 함수의 호출은 _8158_set_home_config() 입니다 .
7. **버튼** :
   - ▷ Next Card: 동작 카드 변경 .
   - ▷ Next Axis: 동작 축 변경 .
   - ▷ Save Config: 8158.ini과 8158MC.ini에 현재의 설정 값 저장 .
   - ▷ Close: 메뉴 닫기 .

이 메뉴에서는 LTC, SD, PCS, Select_Input 을 설정할 수 있습니다 .



1. **LTC 논리 :** LTC 신호의 논리 선택 . 관련된 함수의 호출은 _8158_set_ltc_logic() 입니다 .
2. **LTC latch_ 소스** : latch 소스 신호의 논리 선택 . 관련된 함수의 호출은 _8158_set_latch_source () 입니다 .
3. **SD 설정** : SD 신호 설정 . 관련된 함수의 호출은 _8158_set_sd() 입니다 .
4. **PCS 논리** : SelectNo 신호의 논리 선택 . 관련된 함수의 호출은 _8158_set_pcs_logic() 입니다 .
5. **gpio 입력 설정** : gpio 입력의 설정 선택 . 관련된 함수의 호출은 _8158_set_gpio_input_function 입니다 .
6. **Gpio 논리** : gpio 의 논리 선택 . 관련된 함수의 호출은 _8158_set_gpio_input_function 입니다 .

7. **버튼** :

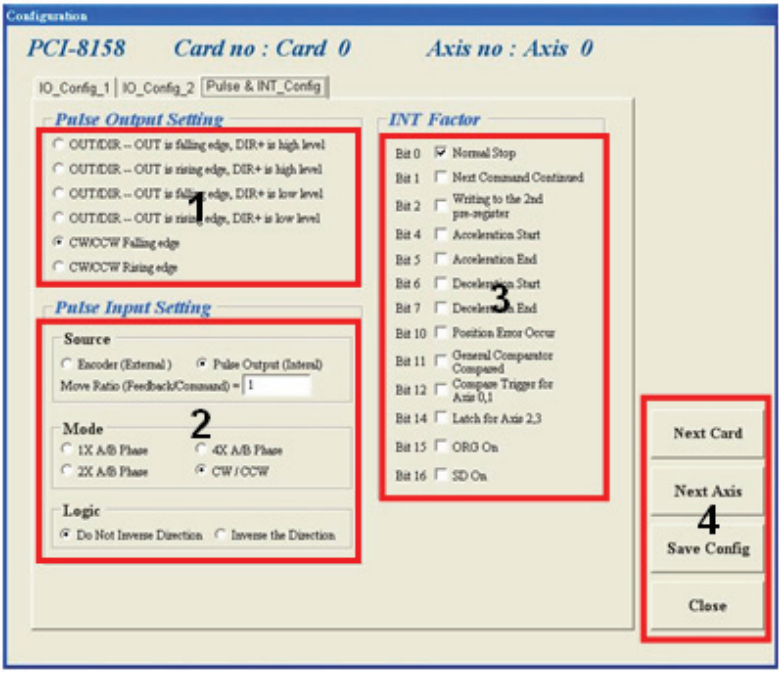
   ▷ **Next Card**: 동작 카드 변경 .

   ▷ **Next Axis**: 동작 축 변경 .

   ▷ **Save Config**: 8158.ini과 8158MC.ini에 현재의 설정 값 저장 .

   ▷ **Close**: 메뉴 닫기 .

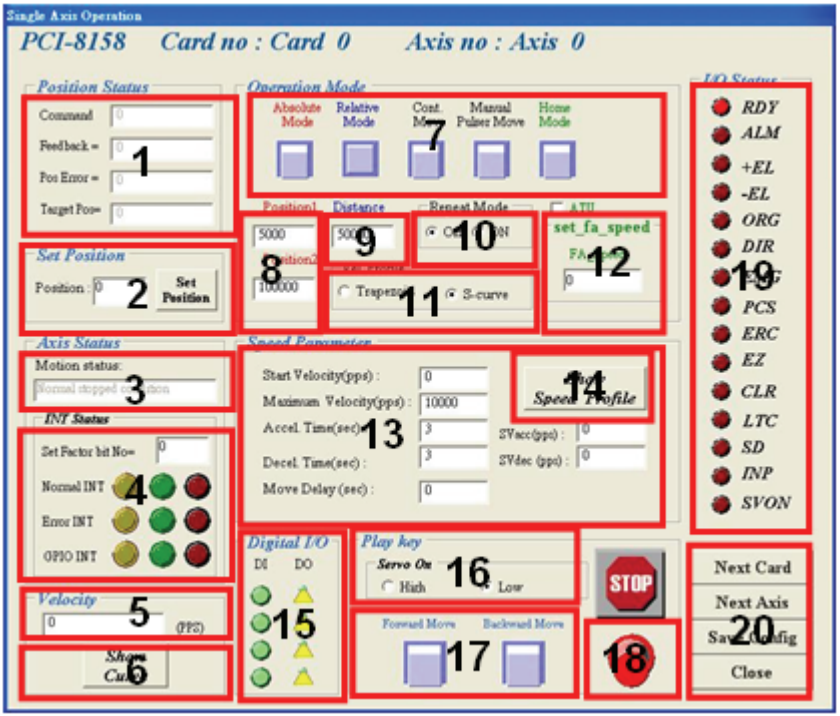
이 메뉴에서는 펄스 입 / 출력의 설정과 동작 비율 , INT factor 를 설정할 수 있습니다 .



1. **펄스 출력 모드** : 펄스 신호 (OUT/ DIR) 의 출력 모드 선택 . 관련된 함수의 호출은 _8158_set_pls_outmode() 입니다 .
2. **펄스 입력** : 펄스 입력 신호 (EA/EB) 의 구성 설정 . 관련된 함수의 호출은 _8158_set_pls_iptmode(), _8158_set_feedback_src() 입니다 .
3. **INT Factor**: INT 활동을 시작하는 factor 선택 . 관련된 함수의 호출은 _8158_set_int_factor() 입니다 .
4. **버튼** :
   - ▷ **Next Card**: 동작 카드 변경 .
   - ▷ **Next Axis**: 동작 축 변경 .
   - ▷ **Save Config**: 8158.ini 과 8158MC.ini 에 현재의 설정 값 저장 .
   - ▷ **Close**: 메뉴 닫기 .

5.3.5 단일 축 동작 메뉴

이 메뉴에서는 속도 모드 모션 , 상대 / 절대 모션 선설정 , 수동 펄스 동작 , 홈 리턴 모드를 포함하는 축의 설정을 변경 할 수 있습니다 .

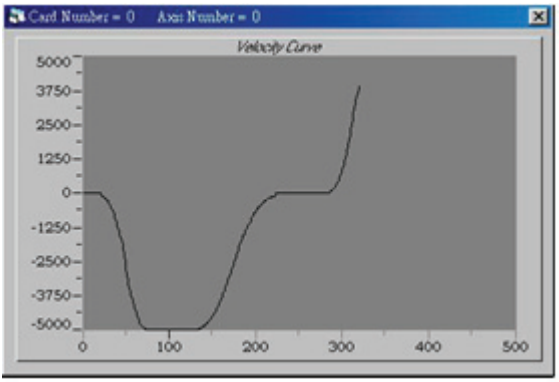


1. **위치** :
   - ▷ **Command**: 명령 카운터의 값을 보여줍니다 . 관련된 함수는 _8158_get_command() 입니다 .
   - ▷ **Feedback**: 피드백 위치 카운터의 값을 보여줍니다. 관련된 함수는 _8158_get_position() 입니다 .
   - ▷ **Pos Error**: 위치 에러 카운터의 값을 보여줍니다 . 관련된 함수는 _8158_get_error_counter() 입니다 .
   - ▷ **Target Pos**: 목표 위치 리코더의 값을 보여줍니다 . 관련된 함수는 _8158_get_target_pos() 입니다 .
2. **위치 리셋** : 이 버튼을 클릭하면 모든 포지셔닝 카운터는

지정된 값으로 설정됩니다 . 관련된 함수는 :

```
_8158_set_position()
_8158_set_command()
_8158_reset_error_counter()
_8158_reset_target_pos()
```

3. **모션 상태** : _8158_motion_done function 의 반환 값을 보여줍니다 . 관련된 함수는 _8158_motion_done() 입니다 .
4. **INT 상태** :
   - ▷ **int_factor bit no**: int_factor 비트 설정 .
   - ▷ **Normal INT**: Normal INT 위상을 보여줍니다 . 관련된 함수는 _8158_wait_motion_interrupt () 입니다 .
   - ▷ **Error INT**: Error INT 위상을 보여줍니다 . 관련된 함수는 _8158_wait_error_interrupt () 입니다 .
   - ▷ **GPIO INT**: GPIO INT 위상을 보여줍니다 . 관련된 함수는 _8158_wait_gpio_interrupt () 입니다 .
5. **속도**: PPS 단위의 속도 절대값을 보여줍니다 . 관련된 함수는 _8158_get_current_speed() 입니다 .
6. **속도 커브 보기 버튼** : 이 버튼을 클릭하면 윈도우는 속도 : 시간 커브를 보여줍니다 . 이 커브에는 매 100ms 마다 새로운 속도 데이터 지점이 추가될 것입니다 . 닫기를 원하실 때는 같은 버튼을 다시 클릭 하시면 되고 데이터를 지우기 원하시는 경우 커브를 클릭 하십시오 .



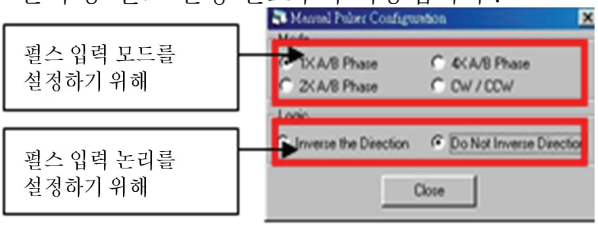
7. **동작 모드** : 동작 모드 선택 .

   ▷ **Absolute Mode**: “위치1” 과 “위치2” 는 모션의 절대 목표 위치로 사용될 것입니다 . 관련된 함수들은 _8158_start_ta_move(), _8158_start_sa_move() 입니다 .

   ▷ **Relative Mode**: “Distance” 는 모션의 상대적인 변위로 사용됩니다 . 관련된 함수는 _8158_start_tr_move(), _8158_start_sr_move() 입니다 .

   ▷ **Cont. Move**: 속도 모션 모드 . 관련된 함수는 _8158_tv_move(), _8158_start_sv_move() 입니다 .

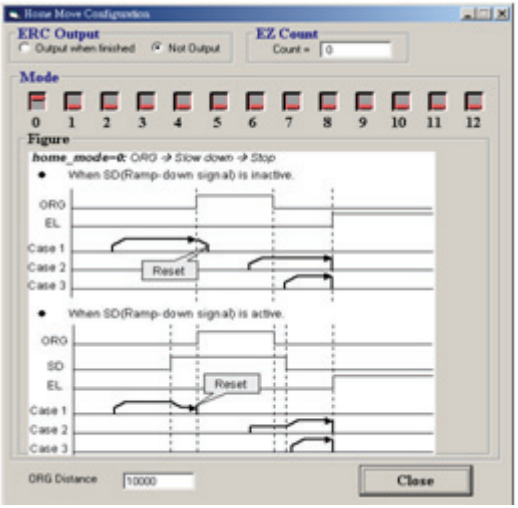
   ▷ **Manual Pulse Move**: 수동 펄스 모션 . 이 버튼을 클릭하면 수동 펄스 설정 윈도우가 작동됩니다 .



   ▷ **Home Mode**: 홈 리턴 모드. 이 버튼을 클릭하면 홈 동작 설정 윈도우가 작동됩니다 . 관련된 함수는 _8158_set_home_config() 입니다 . 만약 체크박스

"ATU" 가 체크된 경우에는 모션 시작 시 자동 homing 이 실행됩니다 .

- ▷ **ERC Output**: 선택 시 홈 동작이 완료될 때 ERC 신호가 전송될 것입니다 .
- ▷ **EZ Count**: EZ 카운트 번호 설정. 이는 지정된 홈 리턴 모드에 영향을 줍니다 .
- ▷ **Mode**: 홈 리턴 모드 설정. 13가지의 모드가 사용 가능합니다 .
- ▷ **Home Mode figure**: 다음 그림은 각 홈 모드의 동작을 설명합니다 .
- ▷ **Close**: 클릭 시 메뉴를 닫습니다 .



8. **위치** : " 절대 모드 " 의 절대 위치 설정 . 이는 오직 " 절대 모드 " 를 선택 했을 때만 유효합니다 .
9. **거리** : " 상대 모드 " 의 상대 거리 설정 . 이는 오직 " 상대 모드 " 를 선택 했을 때만 유효합니다 .
10. **반복 모드** : "On" 이 선택된 경우 모션은 반복 모드가 됩니다 . (전진 <-> 후진 또는 위치 1<-> 위치 2). 이는 오

직 “ 상대 모드 ” 또는 “ 절대 모드 ” 를 선택 했을 때만 유효합니다 .

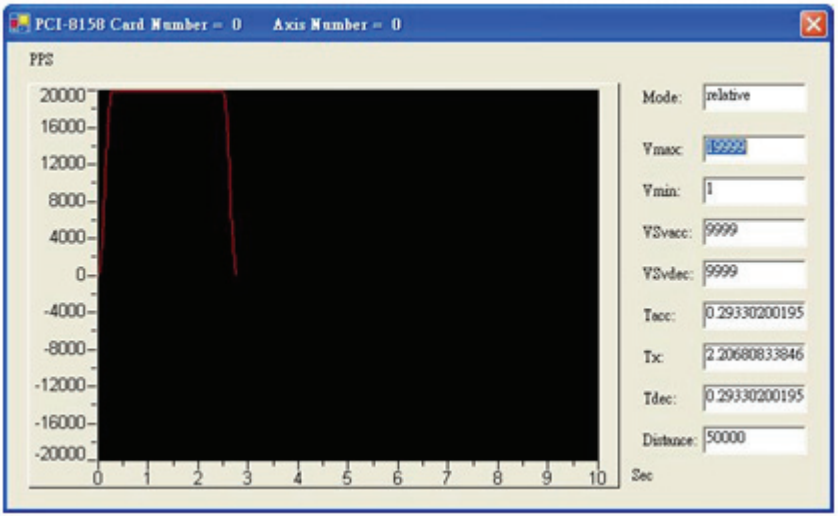
11. **속도 프로파일** : 속도 프로파일 선택 . Trapezoidal 과 S- 커브 “ 절대 모드 ”, “ 상대 모드 ”, “ 일정 운동 ” 에서 사용 가능 .

12. **FA 속도 /ATU**: FA 속도의 구성 설정 . 관련된 함수의 호출은 _8158_set_fa_speed() 입니다 . 만약 체크박스 “ATU” 가 체크된 경우 , 모션이 시작할 때 자동 homing 이 실행 됩니다 .

13. **모션 파라미터** : 단일 축 모션의 파라미터 설정 . 이 파라미터는 “ 수동 펄스 운동 ” 이 선택 되었다면 속도와 이동 거리는 펄스 입력에 의해 결정되므로 아무런 효과가 없습니다 .

    - ▷ **Start Velocity**: PPS 단위의 모션 시작 속도 설정 . “ 절대 모드 ” 또는 “ 상대 모드 ” 모드에서만 이 값은 유효합니다 . 예를 들어 , -100.0 은 100.0 와 동일하며 “ 일정 운동 ” 에서는 값과 사인 모두 유효합니다 . -100.0 은 100.0 의 역 방향을 의미합니다 .
    - ▷ **Maximum Velocity**: PPS 단위의 모션 최대 속도 설정 . 오직 “ 절대 모드 ” 또는 “ 상대 모드 ” 에서 이 값은 유효합니다 . 예를 들어 -5000.0 은 5000.0 와 같으며 “ 일정 운동 ” 에서는 값과 사인 모두 유효합니다 . -5000.0 는 5000.0 의 역 방향을 의미합니다 .
    - ▷ **Accel. Time**: 초 단위의 가속 시간 설정 .
    - ▷ **Decel. Time**: 초 단위의 감속 시간 설정 .
    - ▷ **SVacc**: PPS 단위의 가속 중 S- 커브 범위 설정 .
    - ▷ **SVdec**: PPS 단위의 감속 중 S- 커브 범위 설정 .
    - ▷ **Move Delay**: 이 설정은 반복 모드가 “ON” 으로 설정 되었을 때만 유효합니다 . 이는 PCI-8158 이 다음 모션을 계속 하기 전에 지정 시간만큼 지연시킵니다 .

14. **속도 _ 프로파일** : 이 버튼을 클릭 시 사용자는 속도 프로파일을 볼 수 있습니다 .



15. **디지털 I/O**: 디지털 입 / 출력 설정과 보기 . 관련된 함수는 _8158_get_gpio_output(),_8158_get_gpio_input(), _8158_set_gpio_output() 입니다 .

16. **서보 On**: SVON 신호 출력 위상 설정 . 관련된 함수는 _8158_set_servo().

17. **플레이 키** :

왼쪽 플레이 버튼 : 이 버튼 클릭 시 PCI-8158 은 이전의 설정에 따라 .

- ▷ “ 절대 모드 ” 모드에서 이는 축을 위치 1 로 이동시킵니다 .
- ▷ “ 상대 모드 ” 모드에서 이는 축을 앞으로 이동시킵니다 .
- ▷ “일정 운동” 에서 이는 축이 속도 설정에 따라 이동을 시작하는 원인이 됩니다 .
- ▷ “수동 펄스 운동” 에서 이는 축을 펄스 방향으로 이동시키며 한계 속도는 “ 최대 속도 ” 에서 설정된 값입니다 .

오른쪽 플레이 버튼 : 이 버튼을 클릭 시 이는 PCI-8158 의 이전의 설정에 따라 outlet 펄스를 시작합니다 .

- ▷ “ 절대 모드 ” 모드에서 이는 축을 위치로 이동시킵니다 .
- ▷ “ 상대 모드 ” 모드에서 이는 축을 뒤로 이동시킵니다 .
- ▷ “일정 운동”모드에서 이는 축이 속도 설정에 따라 이동을 시작하는 원인이지만 이는 반대 방향입니다 .
- ▷ “ 수동 펄스 운동 ” 이는 축을 펄스 방향으로 이동시키며 한계 속도는 “ 최대 속도 ” 에서 설정된 값입니다 .

18.**정지 버튼** : 이 버튼을 클릭하면 PCI-8158 은 감속과 정지를 시작합니다 . 감속시간은 “Decel. Time.” 으로 정의되며 관련 함수는 _8158_sd_stop() 입니다 .

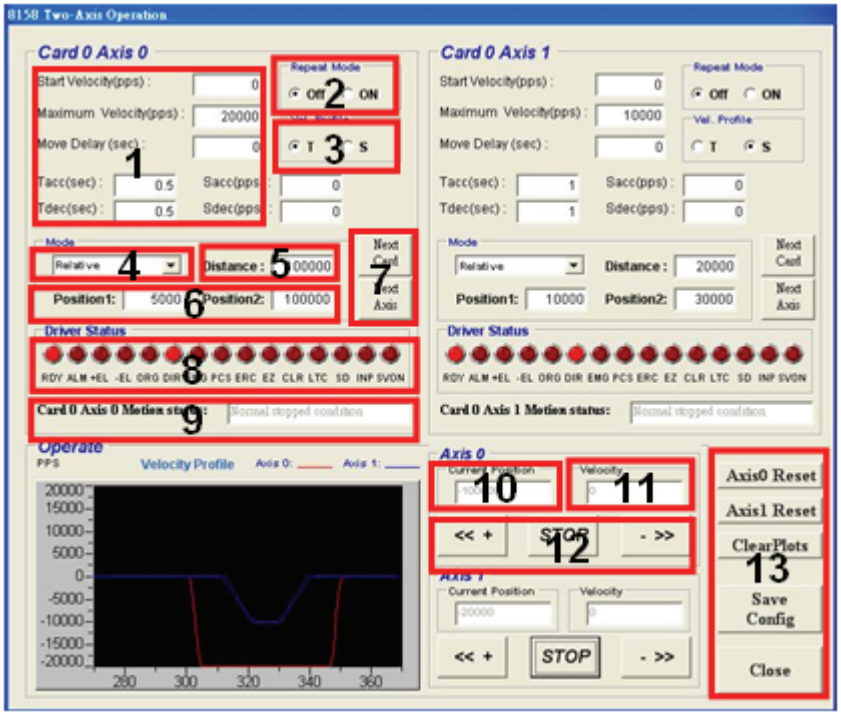
19.**I/O 상태** : 모션 입 / 출력의 위상 . Light-On 은 활성화 , Light-Off 는 비 활성화를 의미합니다 . 관련 함수는 _8158_get_io_status() 입니다 .

20.**버튼** :

- ▷ Next Card: 동작 카드 변경 .
- ▷ Next Axis: 동작 축 변경 .
- ▷ Save Config: 8158.ini과 8158MC.ini에 현재의 설정 값 저장 .
- ▷ Close: 메뉴 닫기 .

5.3.6 두 축 동작 메뉴

이 메뉴에서 사용자는 두 선택된 축의 속도 모드 모션 , 선설정 상대 / 절대 모션을 설정할 수 있습니다 .



1. **모션 파라미터** : 단일 축의 파라미터 설정 . “ 수동 펄스 운동 ” 이 선택된 경우 속도와 이동 거리는 펄스 입력을 통해 결정되므로 이 파라미터는 무효합니다 .
   - ▷ Start Velocity: PPS 단위의 모션의 시작 속도 설정 . 오직 “ 절대 모드 ” 또는 “ 상대 모드 ” 에서만 값은 유효합니다 . 예를 들어 -100.0 는 100.0 와 동일합니다 .
   - ▷ Maximum Velocity: PPS 단위의 모션 최대 속도 설정 . 오직 “ 절대 모드 ” 또는 “ 상대 모드 ” 에서만 값은 유효하며 예를 들어 -5000.0 은 5000.0 와 동일합니다 .
   - ▷ Tacc: 초 단위의 가속 시간 설정 .
   - ▷ Tdec: 초 단위의 감속 시간 설정 .

- ▷ Sacc: PPS 단위의 가속 중 S- 커브범위 설정 .
- ▷ Sdec: PPS 단위의 감속 중 S- 커브범위 설정 .

2. **반복 모드** : “On” 이 선택된 경우 모션은 반복 모드가 됩니다 (전진 <-> 후진 또는 위치 1<-> 위치 2). 이는 오직 “ 상대 모드 ” 또는 “ 절대 모드 ” 가 선택된 경우 유효합니다 .
3. **속도 프로파일** : 속도 프로파일 선택 . Trapezoidal 과 S- 커브는 “ 절대 모드 ” “ 상대 모드 ”, “ 일정 운동 ” 에 사용 가능합니다 .
4. **동작 모드** : 동작 모드 선택 .
   - ▷ Absolute Mode: “위치1”와 “위치2” 은 모션의 절대 목표 위치로 사용되며 관련 함수는 _8158_start_ta_move(), _8158_start_sa_move() 입니다 .
   - ▷ Relative Mode: “거리” 는 모션의 상대 변위로 사용되며 관련 함수는 _8158_start_tr_move(), _8158_start_sr_move() 입니다 .
5. **거리** : “ 상대 모드 ” 의 상대 거리 설정 . 이는 오직 “ 상대 모드 ” 가 선택되었을 경우에만 유효합니다 .
6. **위치** : “ 절대 모드 ” 의 절대 위치 설정 . 이는 오직 “ 절대 모드 ” 가 선택되었을 경우에만 유효합니다 .
7. **버튼** :
   - ▷ Next Card: 동작 카드 변경 .
   - ▷ Next Axis: 동작 축 변경 .
8. **I/O 상태** : 모션 입 / 출력의 위상 설정 . 불이 켜진 것은 활성화 , 꺼진 것은 비활성화를 의미 . 관련 함수는 _8158_get_io_status() 입니다 .
9. **모션 상태** : _8158_motion_done 함수의 리턴 값 보여줌 . 관련 함수는 _8158_motion_done() 입니다 .
10. **현재 위치** :
    - ▷ Command: 명령 카운터의 값을 보여줌 . 관련 함수는 _8158_get_position() 입니다 .

11.**속도** : PPS 단위의 속도 절대 값 . 관련 함수는 _8158_get_current_speed() 입니다 .

12.**플레이 키** :

왼쪽 플레이 버튼 : 이 버튼 클릭 시 PCI-8158 은 이전의 설정에 따라 outlet 펄스를 시작합니다 .

- ▷ “절대 모드”,모드에서 이는 축을 위치1로 이동시킵니다.
- ▷ “ 상대 모드 ”, 모드에서 이는 축을 전진시킵니다 .

오른쪽 플레이 버튼 : 이 버튼 클릭 시 PCI-8158 은 이전의 설정에 따라 outlet 펄스를 시작합니다 .

- ▷ “Absolute Mode” 모드에서 이는 축을 위치 2 로 이동시킵니다 .
- ▷ “Relative Mode” 모드에서 이는 축을 후진시킵니다 .

Stop Button: 이 버튼 클릭 시 PCI-8158 은 감속과 정지를 시작합니다 . 감속 시간은 “Decel. Time.” 에 정의되고 관련 함수는 _8158_sd_stop() 입니다 .

13.**버튼** :

- ▷ Axis0 Reset: 이 버튼 클릭 시 모든 선택된 축의 포지셔닝 카운터는 영으로 설정됩니다 . 관련 함수는 다음과 같습니다 .

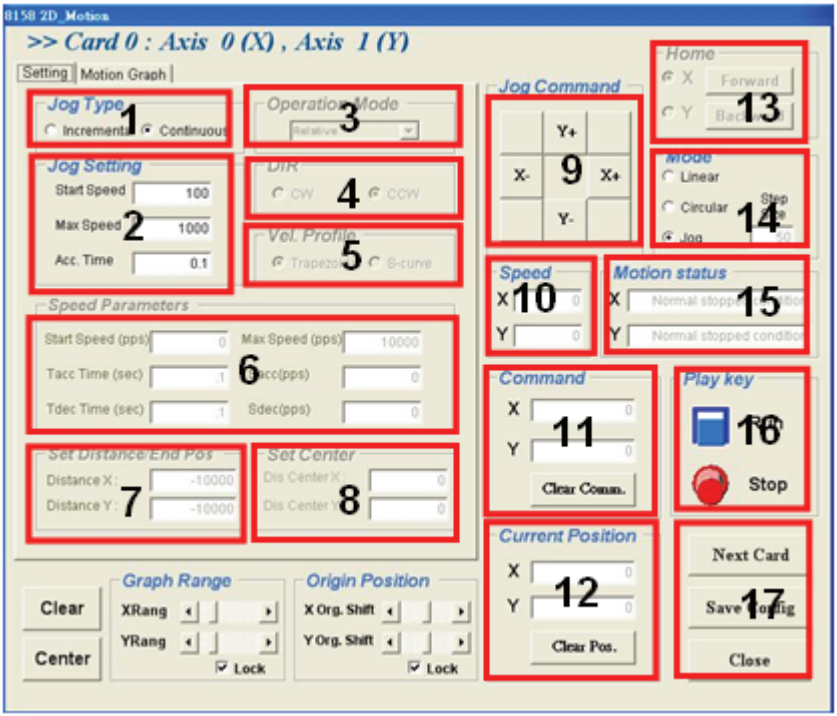
```
_8158_set_position()
_8158_set_command()
_8158_reset_error_counter()
_8158_reset_target_pos()
```

- ▷ Axis1 Reset: 이 버튼 클릭 시 모든 선택된 축의 포지셔닝 카운터는 영으로 설정됩니다 .
- ▷ ClearPlots: 모션 그래프를 제거합니다 .
- ▷ Save Config: 8158.ini과 8158MC.ini에 현재의 설정 값 저장 .
- ▷ Close: 메뉴 닫기 .

5.3.7 2D_모션 메뉴

2-D 버튼을 동작 윈도우에서 클릭하면 아래의 윈도우를 볼 수 있습니다. 이는 2-D 모션 테스트를 위한 메뉴이며 다음의 내용을 포함하고 있습니다.

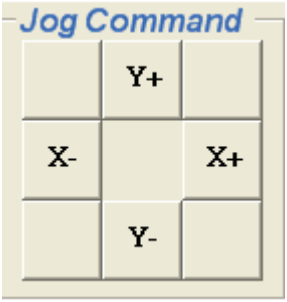
- ▶ 선형 보간
- ▶ 원형 보간
- ▶ 증가형 Jog
- ▶ 연속적 Jog
- ▶ 기타 제어 객체



1. **Jog 타입** :

▷ **연속적 Jog**: 연속적 Jog는 사용자가 한 방향 버튼을 누를 때 축이 속도를 증가하며 연속적으로 동작하는 것을 의미합니다. 사용자가 버튼을 길게 누른다면 동작은 더욱

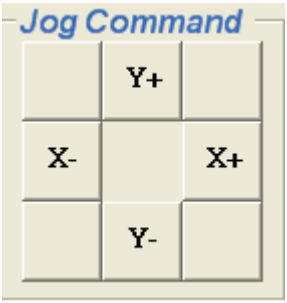
빨라질 것이며 버튼을 누르지 않는다면 축은 즉시 동작을 멈출 것 입니다 .



▷ **증가형 Jog**: 증가형 jog는 사용자가 한 방향 버튼을 누를 때 축은 증가량 설정에 따라 이동할 것입니다 .

2. **Jog 설정** : 단일 축 모션에 대한 파라미터 설정 . “Jog 모드 ” 가 선택된 경우 속도와 이동 거리는 펄스 입력에 따라 결정되므로 이 파라미터는 무효합니다 .
   - ▷ Start Velocity: PPS 단위의 모션 시작 속도 설정 .
   - ▷ Maximum Velocity: PPS 단위의 모션 최대 속도 설정 .
   - ▷ Tacc: 초 단위의 가속 시간 설정 .
3. **동작 모드** : 동작 모드 설정 .
   - ▷ Absolute Mode: “위치”는 “선형 보간 모드”가 선택되었을 경우 모션의 절대 목표 위치와 같이 사용됩니다 . “ABS 정지 위치 ” 와 “ABS 중심 ” 은 “ 원형 보간 모드 ” 가 선택되었을 경우 모션의 절대 목표 위치로 사용되며 관련 함수는 _8158_start_ta_move(), _8158_start_sa_move() 입니다 .
   - ▷ Relative Mode: “ 거리 ” 는 “ 선형 보간 모드 ” 가 선택 되었을 경우 모션의 절대 목표 위치로 사용되며 “Dis 정지 위치 ” 와 “Dis 중심 ” 는 “ 원형 보간 모드 ” 가 선택되었을 경우 모션의 절대 목표 위치로 사용됩니다 . 관련 함수는 _8158_start_tr_move(), _8158_start_sr_move() 입니다 .

4. **DIR**: arc 의 지정된 방향 , CW/CCW, 이는 오직 “ 원형 보간 모드 ” 가 선택되었을 경우 유효합니다 .
5. **속도 프로파일** : 속도 프로파일 선택 . Trapezoidal 와 S-커브 는 “ 선형 보간 모드 ” 와 “ 원형 보간 모드 ” 에 사용 가능 합니다 .
6. **속도 파라미터** : 단일 축의 파라미터 설정 . 이 파라미터는 만약 “ 선형 보간 모드 ” 또는 “ 원형 보간 모드 ” 가 선택된 경우 속도와 이동 거리는 펄스 입력에 의해 결정되므로 무효합니다 .
   - ▷ **Start Velocity**: PPS 단위의 모션 시작 속도 설정 .
   - ▷ **Maximum Velocity**: PPS 단위의 모션 최대 속도 설정 .
   - ▷ **Accel. Time**: 초 단위의 가속 시간 설정 .
   - ▷ **Decel. Time**: 초 단위의 감속 시간 설정 .
   - ▷ **SVacc**: PPS 단위의 가속 중 S- 커브 범위 설정 .
   - ▷ **SVdec**: PPS 단위의 감속 중 S- 커브 범위 설정 .
7. **거리 / 정지 위치 설정** : “ 원형 보간 모드 ” 의 절대 목표 위치 또는 상대 거리 설정 . “ 원형 보간 모드 ” 의 arc 정지 위치 설정 . 이는 “ 선형 보간 모드 ” 와 “ 원형 보간 모드 ” 에 사용 가능 합니다 .
8. **중심점 설정** : “ 원형 보간 모드 ” 의 중앙 위치 설정 . 이는 오직 “ 원형 보간 모드 ” 가 선택되었을 경우 유효합니다 .
9. **Jog 명령** : 이동을 위한 한 방향 버튼 누름 .



10. **속도** : PPS 단위의 속도 절대 값 . 관련된 함수는 _8158_get_current_speed() 입니다 .

11.**보간 명령** :

▷ 명령 : 명령 카운터의 값을 보여줌 . 관련된 함수는 _8158_get_command() 입니다 .

12.**현재 위치** :

▷ 피드백 : 피드백 위치 카운터의 값을 보여줌 . 관련된 함수는 _8158_get_position() 입니다 .

13.**홈 모드** : 홈 리턴 모드 . 이 버튼 클릭 시 홈 이동 설정 윈도우를 볼 수 있습니다 . 관련된 함수는 _8158_set_home_config() 이며 본 기계에는 두 개의 홈 리턴 버튼이 이 윈도우의 좌측 하단에 위치하는데 이는 사용자가 원점으로 돌아가기를 원하는 경우 유용합니다 .

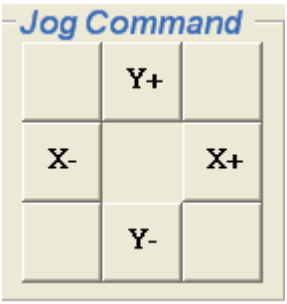
14.**모드** :

▷ **선형 보간** : " 모션 파라미터 설정 프레임 " 에서의 모션 파라미터의 정확한 설정이 완료된 후에 사용자는 이 프레임의 목적지를 입력 할 수 있습니다 . 그 후 Run 버튼을 클릭 하시면 선형 보간이 시작됩니다 .

▷ **원형 보간** : 원형 보간 모드의 설정은 " 모션 파라미터 설정 프레임 " 에 arc 각도 , 분할 축 , 최적 옵션의 세 가지 추가적인 파라미터를 내장합니다 . 설정에 대한 자세한 내용은 6.7, 6.8 장을 참고 하십시오 .

설정이 완료 된 후 " 플레이 키 프레임 " 에 arc 의 중심점과 각도를 입력 할 수 있습니다 . Run 버튼을 클릭 하시면 원형 보간을 시작할 것입니다 .

▷ Jog Type: 연속적 Jog



연속적 jog 는 사용자가 한 방향 버튼을 클릭할 때 축은 속도를

증가하며 연속적으로 움직이는 것을 의미합니다 . 사용자가 길게 버튼을 누를경우 이는 더욱 빠르게 동작 할 것이며 버튼을 누르지 않는다면 축은 즉시 정지 할 것입니다 .

▷ Incremental Jog: 증가형 jog 는 사용자가 한 방향 버튼을 클릭하였을 경우 축이 증가량 설정에 따라 움직이는 것을 의미합니다 .

15.**모션 상태** : _8158_motion_done 기능의 반환값을 보여줌 . 관련 함수는 _8158_motion_done() 입니다 .

16.**플레이 키** :

▶ Play button: 이 버튼 클릭시 PCI-8158 은 이전의 설정에 따라 outlet 펄스를 시작할 것 입니다 .

▷ “ 선형 모드 ,” 에서 이는 축을 거리로 이동시키며 관련 함수는 _8158_start_tr_move_xy, _8158_start_sr_move_xy 입니다 .

▷ “원형 보간,”에서 이는 축을 거리(위치/거리(펄스)로 이동시키며 관련 함수는 _8158_start_tr_arc_xy, _8158_start_sr_arc_xy 입니다 .

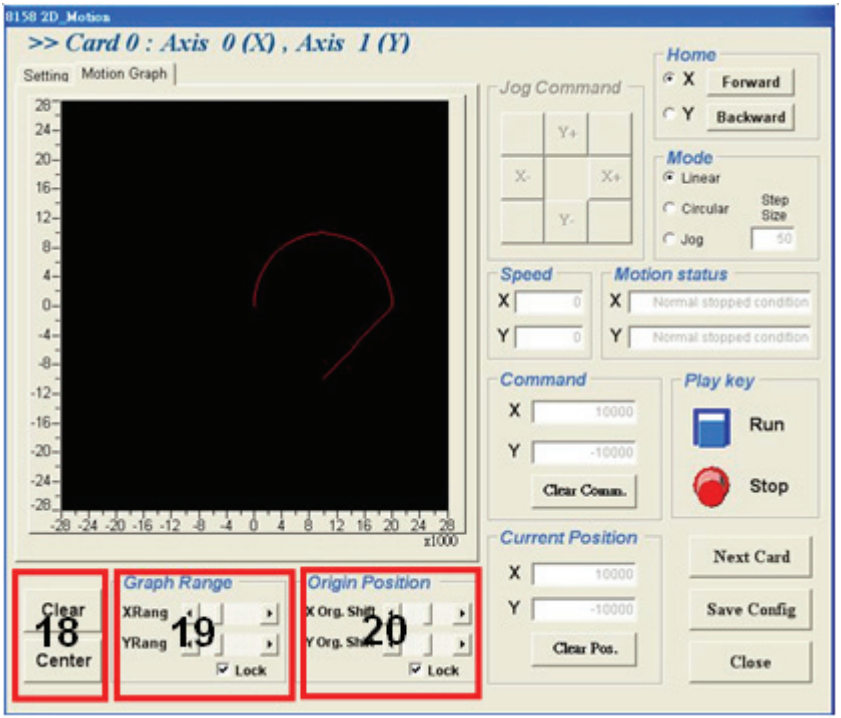
▶ Stop Button: 이 버튼 클릭시 PCI-8158 은 감속 및 정지할 것입니다 . 감속 시간은 “Decel. Time.” 에 정의되며 관련 함수는 _8158_sd_stop() 입니다 .

17.**버튼** :

▷ Next Card: 동작 카드 변경 .

▷ Save Config: 8158.ini과 8158MC.ini에 현재의 설정 값 저장 .

▷ Close: 메뉴 닫기 .



18.**그래프 범위 프레임** :

▷ Clear: 모션 그래프 제거 .

▷ Center: 중앙에 모션 그래프 표시 .

19.**그래프 범위** : X 또는 Y 축의 표시 범위 제어 .

20.**원점 위치**: 표시 위치를 사용자에게 보여주기 위해 제공.

5.3.8 도움말 메뉴

이 메뉴에서 사용자는 마우스 오른쪽 클릭으로 도움말을 볼 수 있습니다 .

의도적인 공백 페이지

6 함수 라이브러리

이 장에서는 PCI-8158 카드에서 지원하는 소프트웨어에 대해 설명합니다 . 사용자는 이 함수를 이용하여 C, C++ 또는 Visual Basic 을 이용하여 프로그램을 개발할 수 있습니다 . 만약 Delphi 환경에서 프로그래밍을 하실 경우 사용자는 헤더파일과 pci_8158.h 의 수동 변경이 필요합니다 .

6.1 함수 리스트

시스템 & 초기화 부분 6.3 절

함수 명칭	설 명
_8158_initial	카드 초기화
_8158_close	카드 종료
_8158_get_version	하드웨어 , 소프트웨어 버전 확인
_8158_set_security_key	보안 암호 설정
_8158_check_security_key	보안 암호 확인
_8158_reset_security_key	보안 암호를 기본 값으로 설정
_8158_config_from_file	파일의 PCI-8158 설정 값을 가져옴

펄스 입 / 출력 설정 6.4 절

함수 명칭	설 명
_8158_set_pls_outmode	펄스 명령 출력 모드 설정
_8158_set_pls_iptmode	인코더 입력 모드 설정
_8158_set_feedback_src	인코더 입력 소스 설정

속도 모드 모션 6.5

함수 명칭	설 명
_8158_tv_move	Trapezoidal 프로파일에서의 일정속도 축 가속
_8158_sv_move	S- 커브 프로파일에서의 일정속도 축 가속
_8158_sd_stop	정지를 위한 감속
_8158_emg_stop	즉시 정지
_8158_get_current_speed	현재 속도 가져옴 (펄스 / 초)
_8158_speed_override	즉시 속도 변경
_8158_set_max_override_speed	최대 override 속도 설정

단일 축 위치 모드 6.6 절

함수 명칭	설 명
_8158_start_tr_move	trapezoidal 프로파일에서의 상대 이동 시작
_8158_start_ta_move	trapezoidal 프로파일에서의 절대 이동 시작
_8158_start_sr_move	S- 커브 프로파일에서의 상대 이동 시작
_8158_start_sa_move	S- 커브 프로파일에서의 절대 이동 시작
_8158_set_move_ratio	명령 펄스와 피드백 펄스의 비율 설정
_8158_position_override	즉시 위치 변경

선형 보간 모션 6.7 절

함수 명칭	설 명
_8158_start_tr_move_xy	trapezoidal 프로파일에서의 X & Y에 대한 상대적 2 축 선형 보간 시작
_8158_start_ta_move_xy	trapezoidal 프로파일에서의 X & Y에 대한 절대적 2 축 선형 보간 시작
_8158_start_sr_move_xy	S- 커브 프로파일에서의 X & Y 에 대한 상대적 2 축 선형 보간 시작
_8158_start_sa_move_xy	S- 커브 프로파일에서의 X & Y 에 대한 절대적 2 축 선형 보간 시작
_8158_start_tr_move_zu	trapezoidal 프로파일에서의 Z & U에 대한 상대적 2 축 선형 보간 시작
_8158_start_ta_move_zu	trapezoidal 프로파일에서의 Z & U에 대한 절대적 2 축 선형 보간 시작
_8158_start_sr_move_zu	S- 커브 프로파일에서의 Z & U 에 대한 상대적 2 축 선형 보간 시작
_8158_start_sa_move_zu	S- 커브 프로파일에서의 Z & U 에 대한 절대적 2 축 선형 보간 시작
_8158_start_tr_move_ab	trapezoidal 프로파일에서의 A & B 에 대한 상대적 2 축 선형 보간 시작
_8158_start_ta_move_ab	trapezoidal 프로파일에서의 A & B 에 대한 절대적 2 축 선형 보간 시작
_8158_start_sr_move_ab	S- 커브 프로파일에서의 A & B 에 대한 상대적 2 축 선형 보간 시작
_8158_start_sa_move_ab	S- 커브 프로파일에서의 A & B 에 대한 절대적 2 축 선형 보간 시작

함수 명칭	설 명
_8158_start_tr_move_cd	trapezoidal 프로파일에서의 C & D 에 대한 상대적 2 축 선형 보간 시작
_8158_start_ta_move_cd	trapezoidal 프로파일에서의 C & D 에 대한 절대적 2 축 선형 보간 시작
_8158_start_sr_move_cd	S- 커브 프로파일에서의 C & D 에 대한 상대적 2 축 선형 보간 시작
_8158_start_sa_move_cd	S- 커브 프로파일에서의 C & D 에 대한 절대적 2 축 선형 보간 시작
_8158_start_tr_line2	trapezoidal 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 상대적 2 축 선형 보간 시작
_8158_start_ta_line2	trapezoidal 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 절대적 2 축 선형 보간 시작
_8158_start_sr_line2	S- 커브 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 상대적 2 축 선형 보간 시작
_8158_start_sa_line2	S- 커브 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 절대적 2 축 선형 보간 시작
_8158_start_tr_line3	trapezoidal 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 상대적 3 축 선형 보간 시작
_8158_start_ta_line3	trapezoidal 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 절대적 3 축 선형 보간 시작
_8158_start_sr_line3	S- 커브 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 상대적 3 축 선형 보간 시작
_8158_start_sa_line3	S- 커브 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 절대적 3 축 선형 보간 시작
_8158_start_tr_line4	trapezoidal 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 상대적 4 축 선형 보간 시작
_8158_start_ta_line4	trapezoidal 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 절대적 4 축 선형 보간 시작
_8158_start_sr_line4	S- 커브 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 상대적 4 축 선형 보간 시작
_8158_start_sa_line4	S- 커브 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 절대적 4 축 선형 보간 시작

원형 보간 모션 6.8 절

함수 명칭	설 명
_8158_start_tr_arc_xy	X & Y 축에 대한 상대적 t- 커브 원형 보간 시작
_8158_start_ta_arc_xy	X & Y 축에 대한 절대적 t- 커브 원형 보간 시작
_8158_start_sr_arc_xy	X & Y 축에 대한 상대적 s- 커브 원형 보간 시작
_8158_start_sa_arc_xy	X & Y 축에 대한 절대적 s- 커브 원형 보간 시작
_8158_start_tr_arc_zu	Z & U 축에 대한 상대적 t- 커브 원형 보간 시작
_8158_start_ta_arc_zu	Z & U 축에 대한 절대적 t- 커브 원형 보간 시작
_8158_start_sr_arc_zu	Z & U 축에 대한 상대적 s- 커브 원형 보간 시작
_8158_start_sa_arc_zu	Z & U 축에 대한 절대적 s- 커브 원형 보간 시작
_8158_start_tr_arc_ab	A & B 축에 대한 상대적 t- 커브 원형 보간 시작
_8158_start_ta_arc_ab	A & B 축에 대한 절대적 t- 커브 원형 보간 시작
_8158_start_sr_arc_ab	A & B 축에 대한 상대적 s- 커브 원형 보간 시작
_8158_start_sa_arc_ab	A & B 축에 대한 절대적 s- 커브 원형 보간 시작
_8158_start_tr_arc_cd	C & D 축에 대한 상대적 t- 커브 원형 보간 시작
_8158_start_ta_arc_cd	C & D 축에 대한 절대적 t- 커브 원형 보간 시작
_8158_start_sr_arc_cd	C & D 축에 대한 상대적 s- 커브 원형 보간 시작
_8158_start_sa_arc_cd	C & D 축에 대한 절대적 s- 커브 원형 보간 시작
_8158_start_tr_arc2	4 개 축의 어떠한 2 축에 대한 상대적 t- 커브 원형 보간 시작
_8158_start_ta_arc2	4 개 축의 어떠한 2 축에 대한 절대적 t- 커브 원형 보간 시작
_8158_start_sr_arc2	4 개 축의 어떠한 2 축에 대한 상대적 s- 커브 원형 보간 시작
_8158_start_sa_arc2	4 개 축의 어떠한 2 축에 대한 절대적 s- 커브 원형 보간 시작

홈 리턴 모드 6.9 절

함수 명칭	설 명
_8158_set_home_config	홈 / 인덱스 논리 구성 설정
_8158_home_move	홈 리턴 동작 시작
_8158_home_search	자동 홈 검색 수행

수동 펄스 모션 6.10 절

함수 명칭	설 명
_8158_set_pulser_iptmode	펄스 입력 모드 설정
_8158_disable_pulser_input	펄스 입력 비활성화
_8158_pulser_vmove	펄스 v 동작 시작
_8158_pulser_pmove	펄스 p 동작 시작
_8158_set_pulser_ratio	실제 펄스 출력 속도를 위한 수동 펄스 비율 설정

모션 상태 선택 6.11

함수 명칭	설 명
_8158_motion_done	모션 위상 귀환

모션 입 / 출력 인터페이스 6.12 절

함수 명칭	설 명
_8158_set_servo	SVON 신호의 On-Off 상태 설정
_8158_set_pcs_logic	PCS(위치 변경 신호) 신호의 논리 설정
_8158_set_pcs	위치 override 를 위한 PCS 비활성화
_8158_set_clr_mode	CLR 신호의 모드 설정
_8158_set_inp	INP 신호의 논리와 동작 모드 설정
_8158_set_alm	ALM 신호의 논리와 동작 모드 설정
_8158_set_erc	ERC 신호의 논리와 타이밍 설정
_8158_set_erc_out	ERC 신호 출력
_8158_clr_erc	ERC 신호 제거
_8158_set_sd	SD 신호의 논리와 동작 모드 설정
_8158_enable_sd	SD 신호 비활성화
_8158_set_limit_logic	EL 신호의 논리 설정
_8158_set_limit_mode	EL 동작 모드 설정
_8158_get_io_status	PCI-8158 의 모든 모션 입 / 출력 위상 불러옴

인터럽트 제어 6.13 절

함수 명칭	설 명
_8158_int_control	INT 서비스 활성화 / 비활성화
_8158_wait_error_interrupt	에러 관련 인터럽트를 대기
_8158_wait_motion_interrupt	모션 관련 인터럽트를 대기
_8158_set_motion_int_factor	모션 관련 인터럽트의 요소를 설정

위치 제어와 카운터 6.14 절

함수 명칭	설 명
_8158_get_position	피드백 위치 카운터의 값을 불러옴
_8158_set_position	피드백 위치 카운터 설정
_8158_get_command	명령 위치 카운터의 값을 불러옴
_8158_set_command	명령 위치 카운터 설정
_8158_get_error_counter	위치 에러 카운터의 값을 불러옴
_8158_reset_error_counter	위치 에러 카운터 리셋
_8158_get_general_counter	일반적 카운터의 값을 불러옴
_8158_set_general_counter	일반적 카운터 설정
_8158_get_target_pos	목표 위치 리코더의 값을 불러옴
_8158_reset_target_pos	목표 위치 리코더 리셋
_8158_get_res_distance	모션에서 축적된 남은 펄스를 불러옴
_8158_set_res_distance	남아있는 펄스 리코더 설정

위치 비교와 Latch 6.15 절

함수 명칭	설 명
_8158_set_trigger_logic	CMP 신호 논리 설정
_8158_set_error_comparator	에러 콤퍼레이터 설정
_8158_set_general_comparator	일반적 콤퍼레이터 설정

함수 명칭	설 명
_8158_set_trigger_comparator	트리거 콤퍼레이터 설정
_8158_set_latch_source	카운터를 위한 latch 타이밍 설정
_8158_set_ltc_logic	LTC 신호의 논리 설정
_8158_get_latch_data	latch 데이터 불러옴

연속적 모션 6.16 절

함수 명칭	설 명
_8158_set_continuous_move	절대 모션을 위한 연속적 모션 활성화
_8158_check_continuous_buffer	버퍼가 비어있는지 확인
_8158_dwell_move	dwell 이동 설정

다축의 동시 동작 6.17 절

함수 명칭	설 명
_8158_set_tr_move_all	다축 동시 동작 설치
_8158_set_ta_move_all	다축 동시 동작 설치
_8158_set_sr_move_all	다축 동시 동작 설치
_8158_set_sa_move_all	다축 동시 동작 설치
_8158_start_move_all	다축 trapezoidal 프로파일 모션 시작
_8158_stop_move_all	다축 모션 동시 정지

범용 입 / 출력 6.18 절

함수 명칭	설 명
_8158_set_gpio_output	디지털 출력 설정
_8158_get_gpio_output	디지털 출력 불러옴
_8158_get_gpio_input	디지털 입력 불러옴
_8158_set_gpio_input_function	모든 디지털 입력에 신호 타입 설정

Soft Limit 6.19

함수 명칭	설 명
_8158_disable_soft_limit	soft limit 기능 비활성화
_8158_enable_soft_limit	soft limit 기능 활성화
_8158_set_soft_limit	soft limits 설정

Backlash 보정 / 진동 억제 6.20

함수 명칭	설 명
_8158_backlash_comp	보정을 위한 backlash 조정 펄스 설정
_8158_suppress_vibration	진동 억제를 위한 유휴 펄스 카운트 설정
_8158_set_fa_speed	홈 모드를 위한 FA 속도 설정

속도 프로파일 연산 6.21

함수 명칭	설 명
_8158_get_tr_move_profile	상대적인 trapezoidal 속도 프로파일을 불러옴
_8158_get_ta_move_profile	절대적인 trapezoidal 속도 프로파일을 불러옴
_8158_get_sr_move_profile	상대적인 S- 커브 속도 프로파일을 불러옴
_8158_get_sa_move_profile	절대적인 S- 커브 속도 프로파일을 불러옴

6.2 C/C++ 프로그래밍 라이브러리

이 장에서는 pci_8158.h 에 있는 함수의 프로토타입과 데이터 타입에 대한 자세한 설명을 다룹니다. 본사는 사용자가 응용 프로그램에서 이하의 데이터를 사용할 것을 제안하며 다음의 표는 데이터 타입의 명칭과 범위를 보여줍니다

타입 명칭	설 명	범 위
U8	8- 비트 ASCII character	0 부터 255
I16	16- 비트 signed integer	-32768 부터 32767
U16	16- 비트 unsigned integer	0 부터 65535
I32	32- 비트 signed long integer	-2147483648 부터 2147483647
U32	32- 비트 unsigned long integer	0 부터 4294967295
F32	32- 비트 단정도 부동 소수점	-3.402823E38 부터 3.402823E38
F64	64- 비트 2 배 정밀도 부동 소수점	-1.797683134862315E308 부터 1.797683134862315E309
Boolean	Boolean 논리 값	TRUE, FALSE

PCI-8158 에서 사용하는 함수의 이름은 함수의 실제 의미를 나타내는데 표현하는 규칙은 다음과 같습니다 :

“C” 언어 환경에서 :

_{하드웨어 모델}_{동작 명칭}. 예를 들어, **_8158_initial().**

C 라이브러리와 VB 라이브러리의 차이를 인식하기 위해 “B” 가 앞에 붙어있는 함수는 VB 라이브러리를 가리킵니다. 예를 들어, **B_8158_initial().**

6.3 시스템 & 초기화

@ 명칭

_8158_initial – 카드 초기화

_8158_close – 카드 정지

_8158_get_version – 하드웨어 , 소프트웨어 버전 정보 점검

_8158_set_security_key – 보안 암호 설정

_8158_check_security_key – 보안 암호 확인

_8158_reset_security_key – 보안 암호 기본값으로 설정

_8158_config_from_file Config – 파일로부터 PCI-8158 설정

@ 설명

_8158_initial:

이 함수는 8158 카드를 초기화 하거나 하드웨어 자원을 할당하는데 사용됩니다. 모든 8158 카드는 응용 프로그램에서 다른 함수를 호출하기 전에 반드시 이 기능을 통해 초기화되어야 합니다. 매개 변수 "Manual_ID" 를 설정하면 사용자는 카드 ID 의 할당을 자동 혹은 수동으로 할 것인지를 선택할 수 있습니다.

_8158_close:

이 함수는 8158 카드를 정지하고 사용자의 응용 프로그램 마지막에 호출되어야 할 자원을 방출하는데 사용됩니다.

_8158_get_version:

사용자는 펌웨어, 드라이버, DLL 버전의 정보를 불러올 수 있습니다.

_8158_set_security_key:

이 함수는 PCI 카드의 보안 코드를 설정하는데 사용됩니다.

또한 : _8158_check_security_key,
_8158_reset_security_key

_8158_check_security_key:

이 함수는 "_8158_set_security_key" 에서 설정된 사용자의 보안 코드를 확인하는데 사용됩니다 .

또한 : _8158_set_security_key, _8158_reset_security_key

_8158_reset_security_key:

이 함수를 통해 사용자는 PCI 카드의 보안 코드를 기본값으로 재설정 할 수 있습니다 . 보안 코드의 기본값은 0 입니다 .

또한 : _8158_check_security_key, _8158_set_security_key

_8158_config_from_file:

이 기능은 지정된 파일에 따라 PCI-8158 의 설정을 불러오는 데사용됩니다 . MotionCreatorPro 를 이용하여 사용자는 8158 의 구성이 정확한지 점검 할 수 있으며 설정의 저장 후 파일은 사용자의 시스템 디렉토리 8158.ini 에 남게 됩니다 .

이 함수가 실행되면 시스템 내의 모든 8158 카드는 8158.ini 에 기록된 파라미터에 따라 이하의 함수들로 설정될 것입니다 .

```
_8158_set_limit_logic
_8158_set_pcs_logic
_8158_set_ltc_logic
_8158_set_inp
_8158_set_erc
_8158_set_alm
_8158_set_pls_iptmode
_8158_set_pls_outmode
_8158_set_move_ratio
_8158_set_latch_source
_8158_set_feedback_src
_8158_set_home_config
_8158_set_soft_limit
_8158_set_fa_speed
_8158_set_sd
```

@ 문법

C/C++(Windows 2000/XP)

```
I16 _8158_initial(I16 *CardID_InBit, I16
     Manual_ID);
I16 _8158_close(void);
I16 _8158_get_version(I16 card_id, I16
     *firmware_ver, I32 *driver_ver, I32
     *dll_ver);
I16 _8158_set_security_key(I16 card_id, I16
     old_secu_code, I16 new_secu_code);
I16 _8158_check_security_key(I16 card_id, I16
     secu_code);
I16 _8158_reset_security_key(I16 card_id);
I16 _8158_config_from_file();
```

Visual Basic 6(Windows 2000/XP)

```
B_8158_initial(CardID_InBit As Integer, ByVal
     Manual_ID As Integer) As Integer
B_8158_close() As Integer
B_8158_get_version(ByVal card_id As Integer,
     firmware_ver As Integer, driver_ver As Long,
     dll_ver As Long) As Integer
B_8158_set_security_key(ByVal card_id As Integer,
     ByVal old_secu_code As Integer, ByVal
     new_secu_code As Integer) As Integer
B_8158_check_security_key(ByVal card_id As
     Integer, ByVal secu_code As Integer)As
     Integer
B_8158_reset_security_key(ByVal card_id As
     Integer);
B_8158_config_from_file() As Integer
```

@ 인수

CardID_InBit:

Manual_ID: 카드 ID 를 결정하는 보드 상의 DIP 스위치(SW1) 비활성화

값의 의미:

카드 ID 는 다음에 의해 결정됩니다 :

0: PCI 슬롯의 순서 .

1: 보드 상 DIP 스위치 (SW1).

card_id: PCI-8158 카드 인덱스 지정 . 카드 ID 는 DIP 스위치 (SW1) 또는 슬롯의 순서에 의해 정해집니다. _8158_initial()를 참고하십시오 .

firmware_ver: 현재의 펌웨어 버전 .

driver_ver: 장치 드라이버 버전 .

dll_ver: 현재의 DLL 라이브러리 버전 .

old_secu_code: 이전 보안 코드 .

new_secu_code: 새로운 보안 코드 .

secu_code: 보안 코드 .

6.4 펄스 입/출력 구성

@ 명칭

_8158_set_pls_iptmode – 피드백 펄스 입력 구성.

_8158_set_pls_outmode – 펄스 명령 출력 구성.

_8158_set_feedback_src – 외부 피드백 입력 펄스 활성화/비활성화.

@ 설 명

_8158_set_pls_iptmode:

외부 피드백 입력 모드 설정. 본 기계는 피드백 입력 펄스를 위한 4 가지 모드를 제공합니다. _8158_set_feedback_src() 함수의 Src 파라미터가 활성화 된 경우에만 이 함수는 유효합니다.

_8158_set_pls_outmode:

명령 펄스의 출력 모드 설정. 본 기계는 명령 펄스 출력을 위한 6 가지 모드를 제공합니다.

_8158_set_feedback_src:

시스템에서 외부 인코더 피드백이 가능할 때 활성화를 위해 이 함수의 Src 파라미터를 설정할 수 있습니다. 그 후 내부 28 비트 up/down 카운터는 _8158_set_pls_iptmode() 함수의 설정에 따라 카운트 할 것이며 명령 펄스의 출력을 카운트할 것입니다.

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_set_pls_iptmode(I16 AxisNo, I16
     pls_iptmode, I16 pls_logic);
I16 _8158_set_pls_outmode(I16 AxisNo, I16
     pls_outmode);
I16 _8158_set_feedback_src(I16 AxisNo, I16 Src);
```

Visual Basic6 (Windows 2000/XP)

```
B_8158_set_pls_iptmode(ByVal AxisNo As Integer,
     ByVal pls_iptmode As Integer, ByVal
     pls_logic As Integer) As Integer
B_8158_set_pls_outmode(ByVal AxisNo As Integer,
     ByVal pls_outmode As Integer) As Integer
B_8158_set_feedback_src(ByVal AxisNo As Integer,
     ByVal Src As Integer) As Integer
```

@ 인수

AxisNo: 축 번호는 펄스의 입 / 출력을 구성하도록 지정됩니다 .

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

pls_iptmode: 인코더 피드백 펄스 입력 모드의 설정

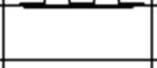
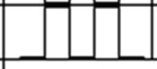
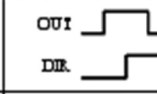
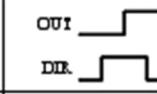
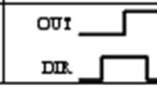
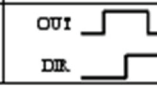
값	의미
0	1X A/B
1	2X A/B
2	4X A/B
3	CW/CCW

pls_logic: 인코더 피드백 펄스의 논리

값	의미
0	역 방향 아님
1	역 방향

pls_outmode: 명령 펄스 출력 모드의 설정 .

값 의미

값	타입	양의(Positive) 방향	음의(Negative) 방향
0	OUT/DIR		
1	OUT/DIR		
2	OUT/DIR		
3	OUT/DIR		
4	CW / CCW		
5	CW / CCW		
6	AB		
7	AB		

Src: 카운터 소스

값	의미
0	외부 피드백
1	명령 펄스

6.5 속도 모드 모션

@ 명칭

_8158_tv_move – trapezoidal 프로파일에서 일정한 속도로 축 가속

_8158_sv_move – S- 커브 프로파일에서 일정한 속도로 축 가속

_8158_emg_stop – 즉시 정지

_8158_sd_stop – 정지를 위한 감속

_8158_get_current_speed – 현재 속도 불러옴

_8158_speed_override – 현재 속도 즉시 변경

_8158_set_max_override_speed – 최대 override 속도 설정

@ 설명

_8158_tv_move:

이 함수는 trapezoidal 프로파일에서 정해진 일정한 속도로 축을 가속 시킵니다. 축은 속도가 변경되거나 정지 명령이 내려질 때까지 일정한 속도로 이동하게 되며 방향은 속도 파라미터의 사인으로 결정됩니다.

_8158_sv_move:

이 함수는 S- 커브 프로파일에서 정해진 일정한 속도로 축을 가속 시킵니다. 축은 속도가 변경되거나 정지 명령이 내려질 때까지 일정한 속도로 이동하게 되며 방향은 속도 파라미터의 사인으로 결정됩니다.

_8158_emg_stop:

이 함수는 축을 즉시 정지시키는데 사용됩니다. 이 함수는 또한 선설정 이동 (trapezoidal, S- 커브), 수동 이동 , 또는 홈 리턴 기능이 수행될 때 유용합니다.

_8158_sd_stop:

trapezoidal 또는 S- 프로파일에서 축을 감속시키는데 사용됩니다. 이 함수는 또한 선설정 이동 (trapezoidal, S- 커브), 수동 이동 , 또는 홈 리턴 기능이 수행될 때 유용합니다. 속도 프로파일은 원 모션 프로파일에 의해 결정됩니다.

_8158_get_current_speed:

이 함수는 현재 펄스 출력 속도 (펄스 / 초) 를 읽는 데 사용하며 언제 어느 동작에서나 사용 가능합니다 .

_8158_speed_override:

이 함수는 예를 들어 "_8158_tv_move" 과 같은 모션 동작 중 속도를 즉시 바꾸는 데 사용합니다. 자세한 내용은 4.2.14를 참고 하십시오 .

또한 : _8158_set_max_override_speed

8158_set_max_override_speed:

이 함수는 속도 override 동작을 하기 전에 최대 override 속도 (100% 속도)를 설정하는데 사용됩니다. 자세한 내용은 4.2.14를 참고 하십시오 .

또한 : _8158_speed_override

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_tv_move(I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc);
I16 _8158_sv_move(I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc, F64 SVacc);
I16 _8158_emg_stop(I16 AxisNo);
I16 _8158_sd_stop(I16 AxisNo, F64 Tdec);
I16 _8158_get_current_speed(I16 AxisNo, F64
     *speed)
I16 _8158_set_max_override_speed(I16 AxisNo, F64
     OvrdSpeed, I16 Enable);
```

Visual Basic6 (Windows 2000/XP)

```
B_8158_tv_move(ByVal AxisNo As Integer, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal Tacc As Double) As Integer
B_8158_sv_move(ByVal AxisNo As Integer, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal Tacc As Double, ByVal SVacc As Double)
     As Integer
B_8158_emg_stop(ByVal AxisNo As Integer) As
     Integer
```

```
B_8158_sd_stop(ByVal AxisNo As Integer, ByVal
     Tdec As Double) As Integer
B_8158_get_current_speed(ByVal AxisNo As Integer,
     ByRef Speed As Double) As Integer
B_8158_set_max_override_speed(ByVal AxisNo As
     Integer, ByVal OvrdSpeed As Double, ByVal
     Enable As Integer) As Integer
```

@ 인수

AxisNo: 축 번호는 이동 또는 정지를 위해 지정됩니다 .

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

StrVel: 초당 펄스 단위의 시작 속도

MaxVel: 초당 펄스 단위의 최대 속도

Tacc: 초 단위의 지정된 가속 시간

SVacc: S- 커브 가속 수행 시 특정한 속도 간격 .

참고 : SVacc = 0, 은 순수 S- 커브

Tdec: 초 단위의 지정된 감속 시간

*** 속도** : 현재 속도를 얻기 위한 변수 (펄스 / 초).

NewVelPercent: 최대 override 속도의 비율 (100% 속도)

Time: override 속도를 위한 현재 속도의 지속시간 . 단위 : 초

OvrdSpeed: 최대 override 속도 (펄스 / 초)

Enable: 0: 비활성화 , 1: override 속도 동작의 활성화

6.6 단일 축 위치 모드

@ 명칭

_8158_start_tr_move - 상대적인 trapezoidal 프로파일 이동 시작

_8158_start_ta_move - 절대적인 trapezoidal 프로파일 이동 시작

_8158_start_sr_move - 상대적인 S- 커브 프로파일 이동 시작

_8158_start_sa_move - 절대적인 S- 커브 프로파일 이동 시작

_8158_set_move_ratio - 명령 펄스와 피드백 펄스의 비율 설정

_8158_position_override - 위치 즉시 변경

@ 설명

일반 :

위치 또는 거리 파라미터의 사인은 이동 방향을 결정합니다 . 만약 이동거리가 특정한 속도에 도달하기 너무 짧을 경우 제어기는 자동적으로 MaxVel 보다 낮게 동작하며 Tacc, Tdec, VSacc 와 VSdec 또한 짧아지는 반면 dV/dt(가속 / 감속) 과 d(dV/dt)/dt (jerk) 는 변하지 않습니다 .

_8158_start_tr_move:

이 함수는 축을 시작 속도 (StrVel) 에서 부터 가속시키고 일정한 속도 (MaxVel) 로 회전시키며 trapezoidal 프로파일의 상대적 거리에서 정지를 위한 감속을 합니다 . 가속 (Tacc) 과 감속 (Tdec) 시간은 각각 특정하며 이는 프로그램을 모션이 완료될 때까지 대기하게 하지는 않지만 , 프로그램으로 즉시 복귀를 위한 제어는 가능합니다 .

_8158_start_ta_move:

이 함수는 축을 시작 속도 (StrVel) 에서 가속시키며 일정한 속도 (MaxVel) 로 회전시키고 trapezoidal 프로파일의 절대적 위치에서 정지를 위한 감속을 시킵니다 . 가속 (Tacc) 과 감속

(Tdec) 시간은 각각 특정하며 이 명령은 프로그램을 모션이 완료 될 때까지 대기하게 하지는 않지만, 프로그램으로 즉시 복귀를 위한 제어는 가능합니다.

_8158_start_sr_move:

이 함수는 축을 시작 속도 (StrVel) 에서 부터 가속시키고 일정한 속도(MaxVel)로 회전시키며 S-커브 프로파일의 상대적 거리에서 정지를 위한 감속을 시킵니다. 가속 (Tacc) 과 감속 (Tdec) 시간은 각각 특정하며 이는 프로그램을 모션이 완료될 때까지 대기하게 하지는 않지만, 프로그램으로 즉시 복귀를 위한 제어는 가능합니다.

_8158_start_sa_move:

이 함수는 축을 시작 속도 (StrVel) 에서 가속시키고 일정한 속도로 회전시키며 S- 커브 프로파일의 특정한 절대적 위치에서 정지를 위한 감속을 시킵니다. 가속과 감속 시간은 각각 특정하며 이 명령은 프로그램이 모션의 완료를 대기하도록 하지는 않지만, 프로그램으로의 즉시 복귀 제어는 가능합니다.

_8158_set_move_ratio:

이 함수는 특정한 축을 위한 척도 인자를 설정합니다. 일반적으로 기계적 분해능이 다를 경우 축은 오직 척도 인자들을 필요로 합니다. 예를 들어 피드백 센서의 분해능이 명령 펄스 분해능의 두 배일 경우 "move_ratio" 파라미터는 2로 설정합니다.

_8158_position_override:

이 함수는 목표 위치를 즉시 바꾸는 데 사용하며 몇 가지의 한계점이 있습니다. 사용하기 전에 4.2.15 를 참고 하십시오.

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_start_tr_move(I16 AxisNo, F64 Dist, F64
     StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8158_start_ta_move(I16 AxisNo, F64 Pos, F64
     StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8158_start_sr_move(I16 AxisNo, F64 Dist, F64
     StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64
     SVacc, F64 SVdec);
```

```
I16 _8158_start_sa_move(I16 AxisNo, F64 Pos, F64
     StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64
     SVacc, F64 SVdec);
I16 _8158_set_move_ratio(I16 AxisNo, F64
     move_ratio);
I16 _8158_position_override(I16 AxisNo, F64
     NewPos);
```

Visual Basic6 (Windows 2000/XP)

```
B_8158_start_tr_move(ByVal AxisNo As Integer,
     ByVal Dist As Double, ByVal StrVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double, ByVal Tdec As Double) As Integer
B_8158_start_ta_move(ByVal AxisNo As Integer,
     ByVal Pos As Double, ByVal StrVel As Double,
     ByVal MaxVel As Double, ByVal Tacc As
     Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_move(ByVal AxisNo As Integer,
     ByVal Dist As Double, ByVal StrVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double, ByVal Tdec As Double, ByVal SVacc
     As Double, ByVal SVdec As Double) As Integer
B_8158_start_sa_move(ByVal AxisNo As Integer,
     ByVal Pos As Double, ByVal StrVel As Double,
     ByVal MaxVel As Double, ByVal Tacc As
     Double, ByVal Tdec As Double, ByVal SVacc As
     Double, ByVal SVdec As Double) As Integer
B_8158_set_move_ratio(ByVal AxisNo As Integer,
     ByVal move_ratio As Double) As Integer
B_8158_position_override(ByVal AxisNo As Integer,
     ByVal NewPos As Double) As Integer
```

@ 인수

AxisNo: 축 번호는 이동 또는 위치 변경을 위해 지정되었습니다.

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Dist: 이동을 위한 지정된 상대거리 (단위 : 펄스)

Pos: 이동을 위한 지정된 절대위치 (단위 : 펄스)

StrVel: PPS 단위의 속도 프로파일의 시작 속도

MaxVel: PPS 단위의 최대 속도

Tacc: 초 단위의 지정된 가속 시간

Tdec: 초 단위의 지정된 감속 시간

SVacc: S- 커브 가속이 수행될 때의 지정된 속도 간격

Note: SVacc = 0, 은 순수 S- 커브 . 더 자세한 정보는 4.2.4 를 참고 하십시오 .

SVdec: S- 커브 감속이 수행될 때의 지정된 속도 간격 .

Note: SVdec = 0, 은 순수 S- 커브 . 더 자세한 정보는 4.2.4 를 참고 하십시오 .

Move_ratio: (피드백 분해능)/(명령 분해능) 의 비율 , 이는 0 이 될 수 없음

NewPos: 이동을 위하여 새로 지정된 절대 위치

6.7 선형 보간 모션

@ 명칭

_8158_start_tr_move_xy - trapezoidal 프로파일에서의 X & Y 에 대한 상대적 2 축 선형 보간 시작

_8158_start_ta_move_xy - trapezoidal 프로파일에서의 X & Y 에 대한 절대적 2 축 선형 보간 시작

_8158_start_sr_move_xy - S-커브 프로파일에서의 X & Y 에 대한 상대적 2 축 선형 보간 시작

_8158_start_sa_move_xy - S-커브 프로파일에서의 X & Y 에 대한 절대적 2 축 선형 보간 시작

_8158_start_tr_move_zu - trapezoidal 프로파일에서의 Z & U 에 대한 상대적 2 축 선형 보간 시작

_8158_start_ta_move_zu - trapezoidal 프로파일에서의 Z & U 에 대한 절대적 2 축 선형 보간 시작

_8158_start_sr_move_zu - S-커브 프로파일에서의 Z & U 에 대한 상대적 2 축 선형 보간 시작

_8158_start_sa_move_zu - S-커브 프로파일에서의 Z & U 에 대한 절대적 2 축 선형 보간 시작

_8158_start_tr_move_ab - trapezoidal 프로파일에서의 A & B 에 대한 상대적 2 축 선형 보간 시작

_8158_start_ta_move_ab - trapezoidal 프로파일에서의 A & B 에 대한 절대적 2 축 선형 보간 시작

_8158_start_sr_move_ab - S- 커브 프로파일에서의 A & B 에 대한 상대적 2 축 선형 보간 시작

_8158_start_sa_move_ab - S-커브 프로파일에서의 A & B 에 대한 절대적 2 축 선형 보간 시작

_8158_start_tr_move_cd - trapezoidal 프로파일에서의 C & D 에 대한 상대적 2 축 선형 보간 시작

_8158_start_ta_move_cd - trapezoidal 프로파일에서의 C & D 에 대한 절대적 2 축 선형 보간 시작

_8158_start_sr_move_cd – S-커브 프로파일에서의 C & D에 대한 상대적 2 축 선형 보간 시작

_8158_start_sa_move_cd – S-커브 프로파일에서의 C & D에 대한 절대적 2 축 선형 보간 시작

_8158_start_tr_line2 – trapezoidal 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 상대적 2 축 선형 보간 시작

_8158_start_ta_line2 – trapezoidal 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 절대적 2 축 선형 보간 시작

_8158_start_sr_line2 – S-커브 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 상대적 2 축 선형 보간 시작

_8158_start_sa_line2 – S-커브 프로파일에서의 4 개의 축의 어떠한 2 개의 축에 대한 절대적 2 축 선형 보간 시작

_8158_start_tr_line3 – trapezoidal 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 상대적 3 축 선형 보간 시작

_8158_start_ta_line3 – trapezoidal 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 절대적 3 축 선형 보간 시작

_8158_start_sr_line3 – S-커브 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 상대적 3 축 선형 보간 시작

_8158_start_sa_line3 – S-커브 프로파일에서의 4 개의 축의 어떠한 3 개의 축에 대한 절대적 3 축 선형 보간 시작

_8158_start_tr_line4 – trapezoidal 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 상대적 4 축 선형 보간 시작

_8158_start_ta_line4 – trapezoidal 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 절대적 4 축 선형 보간 시작

_8158_start_sr_line4 – S-커브 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 상대적 4 축 선형 보간 시작

_8158_start_sa_line4 – S-커브 프로파일에서의 4 개의 축의 어떠한 4 개의 축에 대한 절대적 4 축 선형 보간 시작

@ 설명

이 함수는 다른 프로파일에서의 선형 보간 모션을 수행합니다. 함수들의 자세한 비교는 다음 표에서 설명하고 있습니다.

함수	총 축	속도 프로파일	상대 / 절대	목표 / 축
_8158_start_tr_move_xy	2	T	상대	축 0 & 1
_8158_start_ta_move_xy	2	T	절대	축 0 & 1
_8158_start_sr_move_xy	2	S	상대	축 0 & 1
_8158_start_sa_move_xy	2	S	절대	축 0 & 1
_8158_start_tr_move_zu	2	T	상대	축 2 & 3
_8158_start_ta_move_zu	2	T	절대	축 2 & 3
_8158_start_sr_move_zu	2	S	상대	축 2 & 3
_8158_start_sa_move_zu	2	S	절대	축 2 & 3
_8158_start_tr_move_ab	2	T	상대	축 4& 5
_8158_start_ta_move_ab	2	T	절대	축 4 & 5
_8158_start_sr_move_ab	2	S	상대	축 4 & 5
_8158_start_sa_move_ab	2	S	절대	축 4 & 5
_8158_start_tr_move_cd	2	T	상대	축 6 & 7
_8158_start_ta_move_cd	2	T	절대	축 6 & 7
_8158_start_sr_move_cd	2	S	상대	축 6 & 7
_8158_start_sa_move_cd	2	S	절대	축 6 & 7

함수	총 축	속도 프로파일	상대 / 절대	목표 / 축
_8158_start_tr_line2	2	T	상대	4 개 축의 어떤 2 축
_8158_start_ta_line2	2	T	절대	4 개 축의 어떤 2 축
_8158_start_sr_line2	2	S	상대	4 개 축의 어떤 2 축
_8158_start_sa_line 2	2	S	절대	4 개 축의 어떤 2 축

참고 : 선형 보간의 목표가 되는 두 축은 카드상 , 앞의 0-3 축 또는 뒤의 4-7 축 가운데 2 축이며 두 그룹 간 교차할 수 없습니다 .

함수	총 축	속도 프로파일	상대 / 절대	목표 / 축
_8158_start_tr_line3	3	T	상대	4 개 축의 어떤 3 축
_8158_start_ta_line3	3	T	절대	4 개 축의 어떤 3 축
_8158_start_sr_line3	3	S	상대	4 개 축의 어떤 3 축 s
_8158_start_sa_line3	3	S	절대	4 개 축의 어떤 3 축

참고 : 선형 보간의 목표가 되는 세 축은 카드상 앞의 0-3 축 또는 뒤의 4-7 축 가운데 3 축이며 두 그룹 간 교차할 수 없습니다 .

함수	총 축	속도 프로파일	상대 / 절대	목표 / 축
_8158_start_tr_line4	4	T	상대	4 개 축의 어떤 4 축
_8158_start_ta_line4	4	T	절대	4 개 축의 어떤 4 축
_8158_start_sr_line4	4	S	상대	4 개 축의 어떤 4 축
_8158_start_sa_line4	4	S	절대	4 개 축의 어떤 4 축

참고 : 선형 보간의 목표가 되는 네 축은 카드상 앞의 0-3 축 또는 뒤의 4-7 축 가운데 4 축이며 두 그룹 간 교차할 수 없습니다 .

속도 프로파일 :

T: trapezoidal 프로파일

S: s 커브 프로파일

상대 / 절대 :

상대 : 상대 거리

절대 : 절대 위치

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_start_tr_move_xy(I16 Card_id, F64
      DistX, F64 DistY, F64 StrVel, F64 MaxVel,
      F64 Tacc, F64 Tdec);
I16 _8158_start_ta_move_xy(I16 Card_id, F64 PosX,
      F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
      F64 Tdec);
I16 _8158_start_sr_move_xy(I16 Card_id, F64
      DistX, F64 DistY, F64 StrVel, F64 MaxVel,
      F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_move_xy(I16 Card_id, F64 PosX,
      F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
      F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_tr_move_zu(I16 Card_id, F64
      DistX, F64 DistY, F64 StrVel, F64 MaxVel,
      F64 Tacc, F64 Tdec);
```

```
I16 _8158_start_ta_move_zu(I16 Card_id, F64 PosX,
     F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec);
I16 _8158_start_sr_move_zu(I16 Card_id, F64
     DistX, F64 DistY, F64 StrVel, F64 MaxVel,
     F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_move_zu(I16 Card_id, F64 PosX,
     F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_tr_move_ab(I16 Card_id, F64
     DistX, F64 DistY, F64 StrVel, F64 MaxVel,
     F64 Tacc, F64 Tdec);
I16 _8158_start_ta_move_ab(I16 Card_id, F64 PosX,
     F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec);
I16 _8158_start_sr_move_ab(I16 Card_id, F64
     DistX, F64 DistY, F64 StrVel, F64 MaxVel,
     F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_move_ab(I16 Card_id, F64 PosX,
     F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_tr_move_cd(I16 Card_id, F64
     DistX, F64 DistY, F64 StrVel, F64 MaxVel,
     F64 Tacc, F64 Tdec);
I16 _8158_start_ta_move_cd(I16 Card_id, F64 PosX,
     F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec);
I16 _8158_start_sr_move_cd(I16 Card_id, F64
     DistX, F64 DistY, F64 StrVel, F64 MaxVel,
     F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_move_cd(I16 Card_id, F64 PosX,
     F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_tr_line2(I16 *AxisArray, F64
     *DistArray, F64 StrVel, F64 MaxVel, F64
     Tacc, F64 Tdec);
I16 _8158_start_ta_line2(I16 *AxisArray, F64
     *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec);
I16 _8158_start_sr_line2(I16 *AxisArray, F64
     *DistArray, F64 StrVel, F64 MaxVel, F64
     Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
```

```
I16 _8158_start_sa_line2(I16 *AxisArray, F64
     *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_tr_line3(I16 *AxisArray, F64
     *DistArray, F64 StrVel, F64 MaxVel, F64
     Tacc, F64 Tdec);
I16 _8158_start_ta_line3(I16 *AxisArray, F64
     *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec);
I16 _8158_start_sr_line3(I16 *AxisArray, F64
     *DistArray, F64 StrVel, F64 MaxVel, F64
     Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_line3(I16 *AxisArray, F64
     *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_tr_line4(I16 *AxisArray, F64
     *DistArray, F64 StrVel, F64 MaxVel, F64
     Tacc, F64 Tdec);
I16 _8158_start_ta_line4(I16 *AxisArray, F64
     *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec);
I16 _8158_start_sr_line4(I16 *AxisArray, F64
     *DistArray, F64 StrVel, F64 MaxVel, F64
     Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_line4(I16 *AxisArray, F64
     *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc,
     F64 Tdec, F64 SVacc, F64 SVdec);
```

Visual Basic6 (Windows 2000/XP)

```
B_8158_start_tr_move_xy(ByVal Card_id As Integer,
     ByVal DistX As Double, ByVal DistY As
     Double, ByVal StrVel As Double, ByVal MaxVel
     As Double, ByVal Tacc As Double, ByVal Tdec
     As Double) As Integer
B_8158_start_ta_move_xy(ByVal Card_id As Integer,
     ByVal PosX As Double, ByVal PosY As Double,
     ByVal StrVel As Double, ByVal MaxVel As
     Double, ByVal Tacc As Double, ByVal Tdec As
     Double) As Integer
B_8158_start_sr_move_xy(ByVal Card_id As Integer,
     ByVal DistX As Double, ByVal DistY As
     Double, ByVal StrVel As Double, ByVal MaxVel
     As Double, ByVal Tacc As Double, ByVal Tdec
```

```
        As Double, ByVal SVacc As Double, ByVal
        SVdec As Double) As Integer
B_8158_start_sa_move_xy(ByVal Card_id As Integer,
        ByVal PosX As Double, ByVal PosY As Double,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal SVacc As Double, ByVal SVdec
        As Double) As Integer
B_8158_start_tr_move_zu(ByVal Card_id As Integer,
        ByVal DistX As Double, ByVal DistY As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal Tacc As Double, ByVal Tdec
        As Double);
B_8158_start_ta_move_zu(ByVal Card_id As Integer,
        ByVal PosX As Double, ByVal PosY As Double,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double) As Integer
B_8158_start_sr_move_zu(ByVal Card_id As Integer,
        ByVal DistX As Double, ByVal DistY As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal Tacc As Double, ByVal Tdec
        As Double, ByVal SVacc As Double, ByVal
        SVdec As Double) As Integer
B_8158_start_sa_move_zu(ByVal Card_id As Integer,
        ByVal PosX As Double, ByVal PosY As Double,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal SVacc As Double, ByVal SVdec
        As Double) As Integer
B_8158_start_tr_move_ab(ByVal Card_id As Integer,
        ByVal DistX As Double, ByVal DistY As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal Tacc As Double, ByVal Tdec
        As Double);
B_8158_start_ta_move_ab(ByVal Card_id As Integer,
        ByVal PosX As Double, ByVal PosY As Double,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double) As Integer
B_8158_start_sr_move_ab(ByVal Card_id As Integer,
        ByVal DistX As Double, ByVal DistY As
        Double, ByVal StrVel As Double, ByVal MaxVel
```

```
        As Double, ByVal Tacc As Double, ByVal Tdec
        As Double, ByVal SVacc As Double, ByVal
        SVdec As Double) As Integer
B_8158_start_sa_move_ab(ByVal Card_id As Integer,
        ByVal PosX As Double, ByVal PosY As Double,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal SVacc As Double, ByVal SVdec
        As Double) As Integer
B_8158_start_tr_move_cd(ByVal Card_id As Integer,
        ByVal DistX As Double, ByVal DistY As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal Tacc As Double, ByVal Tdec
        As Double);
B_8158_start_ta_move_cd(ByVal Card_id As Integer,
        ByVal PosX As Double, ByVal PosY As Double,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double) As Integer
B_8158_start_sr_move_cd(ByVal Card_id As Integer,
        ByVal DistX As Double, ByVal DistY As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal Tacc As Double, ByVal Tdec
        As Double, ByVal SVacc As Double, ByVal
        SVdec As Double) As Integer
B_8158_start_sa_move_cd(ByVal Card_id As Integer,
        ByVal PosX As Double, ByVal PosY As Double,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal SVacc As Double, ByVal SVdec
        As Double) As Integer
B_8158_start_tr_line2(AxisArray() As Integer,
        DistArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_ta_line2(AxisArray() As Integer,
        PosArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_line2((AxisArray() As Integer,
        DistArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
```

```
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_sa_line2(AxisArray() As Integer,
        PosArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_tr_line3(AxisArray() As Integer,
        DistArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_ta_line3(AxisArray() As Integer,
        PosArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_line3((AxisArray() As Integer,
        DistArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_sa_line3(AxisArray() As Integer,
        PosArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_tr_line4(AxisArray() As Integer,
        DistArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_ta_line4(AxisArray() As Integer,
        PosArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_line4((AxisArray() As Integer,
        DistArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_sa_line4(AxisArray() As Integer,
        PosArray() As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
```

@ 인수

card_id: PCI-8158 카드 인덱스를 지정. 카드 ID는 DIP 스위치 (SW1) 또는 슬롯의 순서에 의해 결정됩니다. _8158_initial()를 참고 하십시오.

AxisNo: 축 번호는 이동 또는 위치 변경을 위해 지정되었습니다.

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

DistX: 이동을 위해 지정된 축 0 의 상대 거리 (단위 : 펄스).

DistY: 이동을 위해 지정된 축 1 의 상대 거리 (단위 : 펄스).

PosX: 이동을 위해 지정된 축 0 의 절대 위치 (단위 : 펄스).

PosY: 이동을 위해 지정된 축 1 의 절대 위치 (단위 : 펄스).

StrVel: PPS 단위의 속도 프로파일 내 시작 속도

MaxVel: PPS 단위의 최대 속도.

Tacc: 초 단위의 지정된 가속 시간.

Tdec: 초 단위의 지정된 감속 시간.

SVacc: S- 커브 가속이 수행될 시의 지정된 속도 간격.

참고 : SVacc = 0, 은 순수 S- 커브. 자세한 사항은 4.2.4 를 참고 하십시오

SVdec: S- 커브 감속이 수행될 시의 지정된 속도 간격.

참고 : SVdec = 0, 은 순수 S- 커브. 자세한 사항은 4.2.4 를 참고 하십시오

***AxisArray:** 보간 수행을 위한 축 번호의 배열 .

예시 : I16 AxisArray[2] = {0, 3}; // 축 0, & 축 3 (정확함)

I16 AxisArray[3] = {0,2, 3}; // 축 0, 2 & 3 (정확함)

I16 AxisArray[2] = {1, 6}; // 축 1, & 축 6 (부정확함)

***DistArray:** 선형 보간에 대한 상대적 거리의 배열 .

예시 : I16 AxisArray[2] = {0, 3}; // 축 0, & 축 3

F64 DistArray[2] = {1000.0, 2000.0} // 축 0 & 3 대해

***PosArray:** 선형 보간에 대한 절대적 위치의 배열 .

예시 : I16 AxisArray[3] = {0,2, 3}; // 축 0, 2 & 3

F64 PosArray[3] = {200.0, 300.0, 400.0} // 축 0, 2 & 3 에 대한 절대 위치

6.8 원형 보간 모션

@ 명칭

_8158_start_tr_arc_xy – X & Y 축에 대한 상대적 t- 커브 원형 보간 시작

_8158_start_ta_arc_xy – X & Y 축에 대한 절대적 t- 커브 원형 보간 시작

_8158_start_sr_arc_xy – X & Y 축에 대한 상대적 s- 커브 원형 보간 시작

_8158_start_sa_arc_xy – X & Y 축에 대한 절대적 s- 커브 원형 보간 시작

_8158_start_tr_arc_zu – Z & U 축에 대한 상대적 t- 커브 원형 보간 시작

_8158_start_ta_arc_zu – Z & U 축에 대한 절대적 t- 커브 원형 보간 시작

_8158_start_sr_arc_zu – Z & U 축에 대한 상대적 s- 커브 원형 보간 시작

_8158_start_sa_arc_zu – Z & U 축에 대한 절대적 s- 커브 원형 보간 시작

_8158_start_tr_arc_ab – A & B 축에 대한 상대적 t- 커브 원형 보간 시작

_8158_start_ta_arc_ab – A & B 축에 대한 절대적 t- 커브 원형 보간 시작

_8158_start_sr_arc_b – A & B 축에 대한 상대적 s- 커브 원형 보간 시작

_8158_start_sa_arc_ab –A & B 축에 대한 절대적 s- 커브 원형 보간 시작

_8158_start_tr_arc_cd – C & D 축에 대한 상대적 t- 커브 원형 보간 시작

_8158_start_ta_arc_cd – C & D 축에 대한 절대적 t- 커브 원형 보간 시작

_8158_start_sr_arc_cd – C & D축에 대한 상대적 s-커브 원형 보간 시작

_8158_start_sa_arc_cd – C & D축에 대한 절대적 s-커브 원형 보간 시작

_8158_start_tr_arc2 – 4 개 축의 어떠한 2 축에 대한 상대적 t- 커브 원형 보간 시작

_8158_start_ta_arc2 – 4 개 축의 어떠한 2 축에 대한 절대적 t- 커브 원형 보간 시작

_8158_start_sr_arc2 – 4 개 축의 어떠한 2 축에 대한 상대적 s- 커브 원형 보간 시작

_8158_start_sa_arc2 – 4 개 축의 어떠한 2 축에 대한 절대적 s- 커브 원형 보간 시작

@ 설명

이 함수는 다른 프로파일에서의 원형 보간 모션을 수행합니다. 함수들의 자세한 비교는 다음 표에서 설명하고 있습니다.

함수	총 축	속도 프로파일	상대 / 절대	목표 축
_8158_start_tr_arc_xy	2	trapezoidal	상대	축 0 & 1
_8158_start_ta_arc_xy	2	trapezoidal	절대	축 0 & 1
_8158_start_sr_arc_xy	2	S- 커브	상대	축 0 & 1
_8158_start_sa_arc_xy	2	S- 커브	절대	축 0 & 1
_8158_start_tr_arc_zu	2	trapezoidal	상대	축 2 & 3
_8158_start_ta_arc_zu	2	trapezoidal	절대	축 2 & 3
_8158_start_sr_arc_zu	2	S- 커브	상대	축 2 & 3
_8158_start_sa_arc_zu	2	S- 커브	절대	축 2 & 3
_8158_start_tr_arc_ab	2	trapezoidal	상대	축 4 & 5
_8158_start_ta_arc_ab	2	trapezoidal	절대	축 4 & 5
_8158_start_sr_arc_ab	2	S- 커브	상대	축 4 & 5
_8158_start_sa_arc_ab	2	S- 커브	절대	축 4 & 5
_8158_start_tr_arc_cd	2	trapezoidal	상대	축 6 & 7
_8158_start_ta_arc_cd	2	trapezoidal	절대	축 6 & 7
_8158_start_sr_arc_cd	2	S- 커브	상대	축 6 & 7
_8158_start_sa_arc_cd	2	S- 커브	절대	축 6 & 7

함수	총 축	속도 프로파일	상대 / 절대	목표 축
_8158_start_tr_arc2	2	trapezoidal	상대	4 개 축의 어떤 2 축
_8158_start_ta_arc2	2	trapezoidal	절대	4 개 축의 어떤 2 축
_8158_start_sr_arc2	2	S- 커브	상대	4 개 축의 어떤 2 축
_8158_start_sa_arc2	2	S- 커브	절대	4 개 축의 어떤 2 축

참고 : 선형 보간의 목표가 되는 2 개의 축은 앞의 4 축 (0-3) 또는 뒤의 4 축 (4-7) 중의 2 축이며 두 그룹 간 교차할 수 없습니다 .

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_start_tr_arc_xy(I16 card_id, F64
     OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
     OffsetEy, I16 CW_CCW, F64 StrVel,F64
     MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_ta_arc_xy(I16 card_id, F64 Cx,
     F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
     StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_sr_arc_xy(I16 card_id, F64
     OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
     OffsetEy, I16 CW_CCW, F64 StrVel,F64
     MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64
     SVdec);
I16 _8158_start_sa_arc_xy(I16 card_id, F64 Cx,
     F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
     StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64
     SVacc,F64 SVdec);
I16 _8158_start_tr_arc_zu(I16 card_id, F64
     OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
     OffsetEy, I16 CW_CCW, F64 StrVel,F64
     MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_ta_arc_zu(I16 card_id, F64 Cx,
     F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
     StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_sr_arc_zu(I16 card_id, F64
     OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
```

```
        OffsetEy, I16 CW_CCW, F64 StrVel,F64
        MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64
        SVdec);
I16 _8158_start_sa_arc_zu(I16 card_id, F64 Cx,
        F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
        StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64
        SVacc,F64 SVdec);
I16 _8158_start_tr_arc_ab(I16 card_id, F64
        OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
        OffsetEy, I16 CW_CCW, F64 StrVel,F64
        MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_ta_arc_ab(I16 card_id, F64 Cx,
        F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
        StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_sr_arc_ab(I16 card_id, F64
        OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
        OffsetEy, I16 CW_CCW, F64 StrVel,F64
        MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64
        SVdec);
I16 _8158_start_sa_arc_ab(I16 card_id, F64 Cx,
        F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
        StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64
        SVacc,F64 SVdec);
I16 _8158_start_tr_arc_cd(I16 card_id, F64
        OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
        OffsetEy, I16 CW_CCW, F64 StrVel,F64
        MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_ta_arc_cd(I16 card_id, F64 Cx,
        F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
        StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_sr_arc_cd(I16 card_id, F64
        OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
        OffsetEy, I16 CW_CCW, F64 StrVel,F64
        MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64
        SVdec);
I16 _8158_start_sa_arc_cd(I16 card_id, F64 Cx,
        F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64
        StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64
        SVacc,F64 SVdec);
I16 _8158_start_tr_arc2(I16 *AxisArray, F64
        *OffsetCenter, F64 *OffsetEnd, I16 CW_CCW,
        F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);
```

```
I16 _8158_start_ta_arc2(I16 *AxisArray, F64
     *CenterPos, F64 *EndPos, I16 CW_CCW, F64
     StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);
I16 _8158_start_sr_arc2(I16 *AxisArray, F64
     *OffsetCenter, F64 *OffsetEnd, I16 CW_CCW,
     F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec, F64
     SVacc,F64 SVdec);
I16 _8158_start_sa_arc2(I16 *AxisArray, F64
     *CenterPos, F64 *EndPos, I16 CW_CCW, F64
     StrVel,F64 MaxVel, F64 Tacc, F64 Tdec, F64
     SVacc, F64 SVdec);
```

Visual Basic6 (Windows 2000/XP)

```
B_8158_start_tr_arc_xy( ByVal card_id As Integer,
     ByVal OffsetCx As Double, ByVal OffsetCy As
     Double, ByVal OffsetEx As Double, ByVal
     OffsetEy As Double, ByVal CW_CCW As Integer,
     ByVal StrVel As Double, ByVal MaxVel As
     Double, ByVal Tacc As Double, ByVal Tdec As
     Double);
B_8158_start_ta_arc_xy(ByVal card_id As Integer,
     ByVal Cx As Double, ByVal Cy As Double,
     ByVal Ex As Double, ByVal Ey As Double,
     ByVal CW_CCW As Integer, ByVal StrVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_arc_xy(ByVal card_id As Integer,
     ByVal OffsetCx As Double, ByVal OffsetCy As
     Double, ByVal OffsetEx As Double, ByVal
     OffsetEy As Double, ByVal CW_CCW As Integer,
     ByVal StrVel As Double, ByVal MaxVel As
     Double, ByVal Tacc As Double, ByVal Tdec As
     Double, ByVal Svacc As Double, ByVal Svdec
     As Double) As Integer
B_8158_start_sa_arc_xy(ByVal card_id As Integer,
     ByVal Cx As Double, ByVal Cy As Double,
     ByVal Ex As Double, ByVal Ey As Double,
     ByVal CW_CCW As Integer, ByVal StrVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double, ByVal Tdec As Double, ByVal Svacc
     As Double, ByVal Svdec As Double) As Integer
B_8158_start_tr_arc_zu( ByVal card_id As Integer,
     ByVal OffsetCx As Double, ByVal OffsetCy As
```

```
        Double, ByVal OffsetEx As Double, ByVal
        OffsetEy As Double, ByVal CW_CCW As Integer,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double);
B_8158_start_ta_arc_zu(ByVal card_id As Integer,
        ByVal Cx As Double, ByVal Cy As Double,
        ByVal Ex As Double, ByVal Ey As Double,
        ByVal CW_CCW As Integer, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_arc_zu(ByVal card_id As Integer,
        ByVal OffsetCx As Double, ByVal OffsetCy As
        Double, ByVal OffsetEx As Double, ByVal
        OffsetEy As Double, ByVal CW_CCW As Integer,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal Svacc As Double, ByVal Svdec
        As Double) As Integer
B_8158_start_sa_arc_zu(ByVal card_id As Integer,
        ByVal Cx As Double, ByVal Cy As Double,
        ByVal Ex As Double, ByVal Ey As Double,
        ByVal CW_CCW As Integer, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_tr_arc_ab( ByVal card_id As Integer,
        ByVal OffsetCx As Double, ByVal OffsetCy As
        Double, ByVal OffsetEx As Double, ByVal
        OffsetEy As Double, ByVal CW_CCW As Integer,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double);
B_8158_start_ta_arc_ab(ByVal card_id As Integer,
        ByVal Cx As Double, ByVal Cy As Double,
        ByVal Ex As Double, ByVal Ey As Double,
        ByVal CW_CCW As Integer, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_arc_ab(ByVal card_id As Integer,
        ByVal OffsetCx As Double, ByVal OffsetCy As
        Double, ByVal OffsetEx As Double, ByVal
        OffsetEy As Double, ByVal CW_CCW As Integer,
```

```
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal Svacc As Double, ByVal Svdec
        As Double) As Integer
B_8158_start_sa_arc_ab(ByVal card_id As Integer,
        ByVal Cx As Double, ByVal Cy As Double,
        ByVal Ex As Double, ByVal Ey As Double,
        ByVal CW_CCW As Integer, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_tr_arc_cd( ByVal card_id As Integer,
        ByVal OffsetCx As Double, ByVal OffsetCy As
        Double, ByVal OffsetEx As Double, ByVal
        OffsetEy As Double, ByVal CW_CCW As Integer,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double);
B_8158_start_ta_arc_cd(ByVal card_id As Integer,
        ByVal Cx As Double, ByVal Cy As Double,
        ByVal Ex As Double, ByVal Ey As Double,
        ByVal CW_CCW As Integer, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_arc_cd(ByVal card_id As Integer,
        ByVal OffsetCx As Double, ByVal OffsetCy As
        Double, ByVal OffsetEx As Double, ByVal
        OffsetEy As Double, ByVal CW_CCW As Integer,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal Svacc As Double, ByVal Svdec
        As Double) As Integer
B_8158_start_sa_arc_cd(ByVal card_id As Integer,
        ByVal Cx As Double, ByVal Cy As Double,
        ByVal Ex As Double, ByVal Ey As Double,
        ByVal CW_CCW As Integer, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
B_8158_start_tr_arc2(AxisArray() As Integer,
        OffsetCenter() As Double, OffsetEnd() As
        Double, Byval CW_CCW As Integer, ByVal
        StrVel As Double , ByVal MaxVel As Double,
```

```
        ByVal Tacc As Double, ByVal Tdec As Double)
        As Integer
B_8158_start_ta_arc2(AxisArray() As Integer,
        CenterPos() As Double, EndPos() As Double,
        Byval CW_CCW As Integer, ByVal StrVel As
        Double , ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_arc2(AxisArray() As Integer,
        OffsetCenter() As Double, OffsetEnd() As
        Double, Byval CW_CCW As Integer, ByVal
        StrVel As Double , ByVal MaxVel As Double,
        ByVal Tacc As Double, ByVal Tdec As Double,
        ByVal Svacc As Double, ByVal Svdec As
        Double) As Integer
B_8158_start_sa_arc2(AxisArray() As Integer,
        CenterPos() As Double, EndPos() As Double,
        Byval CW_CCW As Integer, ByVal StrVel As
        Double , ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Svacc
        As Double, ByVal Svdec As Double) As Integer
```

@ 인수

card_id: PCI-8158 카드 인덱스를 지정. 카드 ID는 DIP 스위치 (SW1) 또는 슬롯의 순서에 의해 결정됩니다. _8158_initial()를 참고 하십시오.

AxisNo: 축 번호는 이동 또는 위치 변경을 위해 지정되었습니다.

카드 id	물리적축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

OffsetCx: X- 축 (목표축 중의 첫 번째 축) 을 중심점으로 옵셋

OffsetCy: Y- 축 (목표축 중의 두 번째 축) 을 중심점으로 옵셋

OffsetEx: X- 축 (목표축 중의 첫 번째 축) 을 arc 의 정지점으로 옵셋

OffsetEy: Y- 축을 arc 의 정지점으로 옵셋

Cx: arc 중심점의 X- 축 (목표축 중의 첫 번째 축) 절대 위치

Cy: arc 중심점의 Y- 축 (목표축 중의 두 번째 축) 절대 위치

Ex: arc 정지점의 X- 축 (목표축 중의 첫 번째 축) 절대 위치

Ey: arc 정지점의 Y- 축 (목표축 중의 두 번째 축) 절대 위치

CW_CCW: 지정된 arc 방향

값	의미
0	시계방향 (cw)
1	시계 반대방향 (ccw)

StrVel: PPS 단위의 속도 프로파일 내 시작 속도 .

MaxVel: PPS 단위의 최대 속도 .

Tacc: 초 단위의 지정된 가속 시간 .

Tdec: 초 단위의 지정된 감속 시간 .

SVacc: S- 커브 가속 수행 시 지정된 속도 간격 .

참고 : SVacc = 0, 은 순수 S- 커브 자세한 내용은 4.2.4 를 참고하십시오

SVdec: S- 커브 감속 수행 시 지정된 속도 간격 .

참고 : SVdec = 0, 은 순수 S- 커브 자세한 내용은 4.2.4 를 참고하십시오

***AxisArray:** 보간 수행을 위한 축 번호의 배열 .

예시 : I16 AxisArray[2] = {0, 3}; // 축 0, & 축 3 (정확함)

I16 AxisArray[2] = {1, 6}; // 축 1, & 축 6 (부정확함)

***OffsetCenter:** 옵셋을 중심에 배열 (시작위치에 따라)

예시 : F64 OffsetCenter[2] = {2000.0, 0.0}; // 첫번째 & 두 번째 축에 대한 시작위치 (초기지점) 에서 옵셋

***OffsetEnd:** 옵셋을 arc 의 정지점에 배열 (시작 위치에 따라)

예시 : F64 OffsetEnd[2] = {4000.0, 0.0}; // 첫번째 & 두번째 축에 대한 시작위치 (초기지점) 에서 옵셋

***CenterPos:** arc 절대위치의 중심점에 배열

예시 : F64 CenterPos[2] = {2000.0, 0.0}; // 첫번째 & 두번째 축에 대한 절대적 중심점 위치

***EndPos:** arc 절대위치의 정지점에 배열

예시 : F64 EndPos[2] = {4000.0, 0.0}; // 첫번째 & 두번째 축에 대한 절대적 정지점 위치

6.9 홈 리턴 모드

@ 명칭

_8158_set_home_config – 홈 / 인덱스 논리 구성 설정

_8158_home_move – 홈 리턴 동작 시작

_8158_home_search – 자동 홈 검색 수행

@ 설명

_8158_set_home_config:

홈 리턴 모드 설정 , 원점 (ORG) 과 인덱스 신호 (EZ) 논리 , EZ 카운트, home_move() 함수에 대한 ERC 출력 옵션을 포함합니다 . 자세한 홈 모드 제어의 설정에 자세한 내용은 4.2.10 을 참고하십시오 .

_8158_home_move:

이 함수는 _8164_set_home_config() 함수의 설정에 따라 축이 홈 리턴 동작을 수행하도록 합니다 . 움직임의 방향은 속도 파라미터의 사인 (MaxVel) 에 의해 결정됩니다 . 이 함수는 홈 모드의 설정에 따라 정지가 결정되기 때문에 사용자는 초기 이동 방향의 선택에 반드시 주의하셔야 합니다 . 사용자는 또한 limit 스위치를 누르거나 축의 정지에 영향을 미칠 수 있는 다른 상황에 대해 주의하셔야 합니다 . 좀 더 자세한 내용은 4.2.10 을 참고하십시오 .

_8158_home_search:

이 함수는 _8164_set_home_config() 함수의 설정에 따라 축이 홈 검색 동작을 수행하도록 합니다 . 움직임의 방향은 속도 파라미터의 사인 (MaxVel) 에 의해 결정됩니다 . 이 함수는 홈 모드의 설정에 따라 정지가 결정되기 때문에 사용자는 초기 이동 방향의 선택에 반드시 주의하셔야 합니다 . 사용자는 또한 limit 스위치를 누르거나 축의 정지에 영향을 미칠 수 있는 다른 상황에 대해 주의하셔야 합니다 . 좀 더 자세한 내용은 4.2.11 을 참고하십시오 .

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_set_home_config(I16 AxisNo, I16
     home_mode, I16 org_logic, I16 ez_logic, I16
     ez_count, I16 erc_out);
I16 _8158_home_move(I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc);
I16 _8158_home_search(I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc, F64 ORGOffset);
```

Visual Basic (Windows 2000/XP)

```
B_8158_set_home_config(ByVal AxisNo As Integer,
     ByVal home_mode As Integer, ByVal org_logic
     As Integer, ByVal ez_logic As Integer, ByVal
     ez_count As Integer, ByVal erc_out As
     Integer) As Integer
B_8158_home_move(ByVal AxisNo As Integer, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal Tacc As Double) As Integer
B_8158_home_search(ByVal AxisNo As Integer, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal Tacc As Double, ByVal ORGOffset As
     Double) As Integer
```

@ 인수

AxisNo: 축 번호는 이동 또는 위치 변경을 위해 지정되었습니다.

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

home_mode: 홈 리턴을 위한 정지모드, 값은 0 부터 12 까지 이며 자세한 내용은 4.2.10 을 참고 하십시오.

org_logic: ORG 에 대한 동작 논리 구성

값	의미
0	액티브 LO
1	액티브 HI

ez_logic: EZ 에 대한 동작 논리 구성

값	의미
0	액티브 LO
1	액티브 HI

ez_count: 0-15 (4.2.10 을 참고하십시오)

erc_out: ERC 출력 옵션 설정 .

값	의미
0	ERC 출력 안함
1	홈 이동 완료 후 ERC 신호 출력

StrVel: 속도 프로파일 내 시작 속도 . (단위 : 펄스 / 초)

MaxVel: 최대 속도 . (단위 : 펄스 / 초)

Tacc: 지정된 가속 시간 (단위 : 초)

ORGOffset: 홈 검색이 ORG 에 닿을 경우 빠져나가는 펄스의 양 (단위 : 펄스)

6.10 수동 펄스 모션

@ 명칭

_8158_disable_pulser_input – 펄스 입력 비활성화

_8158_pulser_pmove – 펄스 p 동작 시작

_8158_pulser_vmove – 펄스 v 동작 시작

_8158_set_pulser_ratio – 실제 펄스 출력 속도를 위한 수동 펄스 비율 설정

_8158_set_pulser_iptmode – 펄스의 입력 신호 모드 설정

@ 설명

_8158_disable_pulser_input

이 함수는 펄스 입력의 활성화 또는 비활성화를 설정합니다 .

_8158_pulser_pmove

이 명령은 축이 수동 펄스 입력에 따라 이동을 시작하게 합니다 . 축은 한 펄스 수신 시 _8158_disable_pulser_input 함수가 펄스를 비활성화 시키거나 출력 펄스의 수가 거리에 이를 때까지 하나의 펄스를 출력합니다 .

_8158_pulser_vmove

이 명령은 축이 수동 펄스 입력에 따라 이동을 시작하게 합니다 . 축은 _8158_disable_pulser_input 가 펄스를 비활성화 할 때 까지 하나의 수동 펄스를 받을때 하나의 펄스를 출력하게 됩니다 .

_8158_set_pulser_ratio

실제 펄스 출력 속도를 위한 수동 펄스 비율을 설정합니다 . 수동 펄스 출력 속도를 위한 공식은 :

출력 펄스 카운트 = 입력 펄스 카운트 × 4 (MultiF + 1) ×(DivF + 1) / 2048

The DivF = 0~2047 나눔인자

The MultiF= 0~31 곱셈인자

_8158_set_pulser_iptmode

이 함수는 수동 펄스의 입력모드를 설정하는데 사용합니다.

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_disable_pulser_input(I16 AxisNo, U16
     Disable );
I16 _8158_pulser_pmove(I16 AxisNo, F64 Dist, F64
     SpeedLimit);
I16 _8158_pulser_vmove(I16 AxisNo, F64
     SpeedLimit);
I16 _8158_set_pulser_ratio(I16 AxisNo, I16 DivF,
     I16 MultiF);
I16 _8158_set_pulser_iptmode(I16 AxisNo, I16
     InputMode, I16 Inverse);
```

Visual Basic (Windows 2000/XP)

```
B_8158_disable_pulser_input(ByVal AxisNo As
     Integer, ByVal Disable As Integer) As
     Integer
B_8158_pulser_pmove(ByVal AxisNo As Integer,
     ByVal Dist As Double, ByVal SpeedLimit As
     Double) As Integer
B_8158_pulser_vmove(ByVal AxisNo As Integer,
     ByVal SpeedLimit As Double) As Integer
B_8158_set_pulser_ratio(ByVal AxisNo As Integer,
     ByVal DivF As Integer, ByVal MultiF As
     Integer) As Integer
B_8158_set_pulser_iptmode(ByVal AxisNo As
     Integer, ByVal InputMode As Integer, ByVal
     Inverse As Integer) As Integer
```

@ 인수

AxisNo: 축 번호는 이동 또는 위치 변경을 위해 지정되었습니다.

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Disable: 펄스 입력 비활성화

Disable = 1, 펄스 비활성화

Disable = 0, 펄스 활성화

Dist: 이동을 위한 지정된 상대 거리 (단위 : 펄스)

예를 들어 제한 속도가 100pps 로 설정되었을 경우 입력 펄스 신호 속도가 100pps 보다 높더라도 축은 최고 100pps 로 움직이게 됩니다.

DivF: 나눔인자 (0-2047)

MultiF: 곱셈인자 (0-31)

InputMode: PA 와 PB 핀에서의 수동 펄스 입력 모드 설정

값	의미
0	1X AB 위상 타입 펄스 입력
1	2X AB 위상 타입 펄스 입력
2	4X AB 위상 타입 펄스 입력
3	CW/CCW 타입 펄스 입력

Inverse: 펄스 방향의 반대 방향으로 이동

값	의미
0	정방향
1	역방향으로 이동

6.11 모션 상태

@ 명칭

_8102_motion_done – 모션 동작 완료 시 원 상태로 복귀

@ 설명

_8102_motion_done:

8102 의 원 모션 상태로 복귀합니다 . 복귀 코드는 다음과 같습니다 .

0	정상적인 정지 상태
1	DR 대기 중
2	CSTA 입력 대기 중
3	외부 동기 신호 대기 중
4	다른 축의 정지 대기 중
5	ERC 타이머의 동작완료 대기 중
6	방향 전환 타이머의 동작완료 대기 중
7	backlash 보정 중
8	PA/PB 대기
9	FA 속도
10	FL 속도
11	가속 중
12	FH 속도
13	감속 중
14	INP 대기
15	기타 (시작 제어 중)
16	SALM
17	SPEL
18	SMEL
19	SEMG
20	SSTP
21	SERC

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8102_motion_done(I16 AxisNo)
```

Visual Basic (Windows 2000/XP)

```
B_8102_motion_done(ByVal AxisNo As Integer) As
      Integer
```

@ 인수

AxisNo: 축 번호는 이동 또는 위치 변경을 위해 지정되었습니다.

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

6.12 모션 입/출력 인터페이스

@ 명칭

_8158_set_servo – SVON 신호의 On-Off 상태 설정

_8158_set_pcs_logic – PCS 신호의 논리 설정

_8158_set_pcs – 위치 override 를 위한 PCS 활성화

_8158_set_clr_mode – CLR 신호의 모드 설정

_8158_set_inp – INP 신호의 논리와 동작모드 설정

_8158_set_alm – ALM 신호의 논리와 동작모드 설정

_8158_set_erc – ERC 신호의 논리와 타이밍 설정

_8158_set_erc_out – ERC 신호 출력

_8158_clr_erc – ERC 신호 제거

_8158_set_sd – SD 신호의 논리와 동작모드 설정

_8158_enable_sd –SD 신호 활성화

_8158_set_limit_logic – PEL/MEL 신호의 논리 설정

_8158_set_limit_mode – PEL/MEL 동작모드 설정

_8158_get_io_status –PCI-8158 의 모든 모션 입/출력 위상 불러옴

@ 설명

_8158_set_servo:

이 함수에서 사용자는 SVON 신호의 On-Off 상태를 설정할 수 있습니다. 기본값은 1(OFF) 이며 이는 SVON 이 GND 에 개방되어 있음을 나타냅니다.

_8158_set_pcs_logic:

PCS 신호 입력의 활성 논리를 설정합니다.

_8158_set_pcs:

입력 신호 PCS 가 켜졌을 경우 위치 override 를 활성화합니다 . PCS 터미널 상태는 “_8158_get_io_status” 함수로 모니터 할 수 있습니다 .

_8158_set_clr_mode

CLR 이 입력된 신호는 지정된 카운터 (명령 , 위치 , 에러 , 범용 카운터) 를 리셋 할 수 있습니다 . 리셋 동작은 이 함수로 설정할 수 있으며 4 가지 타입의 리셋 동작 모드가 존재합니다 . 자세한 내용은 요지 부분을 참고하십시오 .

_8158_set_inp:

서보 드라이버에서의 In-Position 신호 입력의 활성 논리를 설정합니다 . 사용자는 이 함수를 활성화할 것인지 선택할 수 있으며 기본적으로 비활성화 되어 있습니다 .

_8158_set_alm:

서보 드라이버에서의 ALARM 신호 입력의 활성 논리를 설정합니다 . ALARM 신호가 활성화될 시의 두 가지 반응 모드를 선택할 수 있습니다 .

_8158_set_erc:

사용자는 이 함수로 ERC 의 시간과 논리를 설정할 수 있습니다 . 또한 , ERC 신호의 펄스 폭을 설정 할 수 있습니다 .

_8158_set_erc_out:

이 함수는 ERC 신호를 수동으로 출력합니다 .

_8158_clr_erc:

이 함수는 ERC 신호 출력이 특정 레벨 타입 출력으로 정해진 경우 출력을 리셋하는데 사용됩니다 .

_8158_set_sd:

기계 시스템에서의 SD 신호 입력 동작 모드 , latch 제어 , 활성 논리 설정 . 사용자는 _8158_enable_sd 함수로 활성화할 것인지 선택할 수 있으며 기본적으로 비활성화 되어 있습니다 .

_8158_enable_sd:

SD 신호 입력 활성화 . 기본적으로는 비활성화 되어 있습니다 .

_8158_set_limit_logic:

EL 논리 , normal open 또는 normal closed 설정 .

_8158_set_limit_mode:

EL 신호의 반응 모드 설정 .

_8158_get_io_status:

모든 축에 대한 입 / 출력 위상 불러옴 . 각 비트의 정의는 다음과 같습니다 .

비트	명칭	설명
0	RDY	RDY 핀 입력
1	ALM	경보 신호
2	+EL	양의 Limit 스위치
3	-EL	음의 Limit 스위치
4	ORG	원점 스위치
5	DIR	DIR 출력
6	EMG	EMG 위상
7	PCS	PCS 신호 입력
8	ERC	ERC 핀 출력
9	EZ	인덱스 신호
10	CLR	클리어 신호
11	LTC	Latch 신호 입력
12	SD	Slow Down 신호 입력
13	INP	In-Position 신호 입력
14	SVON	서보 -ON 출력 위상

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_set_servo(I16 AxisNo, I16 on_off);
I16 _8158_set_pcs_logic(I16 AxisNo, I16
     pcs_logic);
I16 _8158_set_pcs(I16 AxisNo, I16 enable);
I16 _8158_set_clr_mode(I16 AxisNo, I16 clr_mode,
     I16 targetCounterInBit);
```

```
I16 _8158_set_inp(I16 AxisNo, I16 inp_enable, I16
     inp_logic);
I16 _8158_set_alm(I16 AxisNo, I16 alm_logic, I16
     alm_mode);
I16 _8158_set_erc(I16 AxisNo, I16 erc_logic, I16
     erc_pulse_width, I16 erc_mode);
I16 _8158_set_erc_out(I16 AxisNo);
I16 _8158_clr_erc(I16 AxisNo);
I16 _8158_set_sd(I16 AxisNo, I16 sd_logic, I16
     sd_latch, I16 sd_mode);
I16 _8158_enable_sd(I16 AxisNo, I16 enable);
I16 _8158_set_limit_logic(I16 AxisNo, U16 Logic
     );
I16 _8158_set_limit_mode(I16 AxisNo, I16
     limit_mode);
I16 _8158_get_io_status(I16 AxisNo, U16 *io_sts);
```

Visual Basic (Windows 2000/XP)

```
B_8158_set_servo(ByVal AxisNo As Integer, ByVal
     on_off As Integer) As Integer
B_8158_set_pcs_logic(ByVal AxisNo As Integer,
     ByVal pcs_logic As Integer) As Integer
B_8158_set_pcs(ByVal AxisNo As Integer, ByVal
     enable As Integer)As Integer
B_8158_set_clr_mode(ByVal AxisNo As Integer,
     ByVal clr_mode As Integer, ByBal
     targetCounterInBit as Integer) As Integer
B_8158_set_inp(ByVal AxisNo As Integer, ByVal
     inp_enable As Integer, ByVal inp_logic As
     Integer) As Integer
B_8158_set_alm(ByVal AxisNo As Integer, ByVal
     alm_logic As Integer, ByVal alm_mode As
     Integer) As Integer
B_8158_set_erc(ByVal AxisNo As Integer, ByVal
     erc_logic As Integer, ByVal erc_pulse_width
     As Integer, ByVal erc_mode As Integer) As
     Integer
B_8158_set_erc_out(ByVal AxisNo As Integer) As
     Integer
B_8158_clr_erc(ByVal AxisNo As Integer) As
     Integer
B_8158_set_sd(ByVal AxisNo As Integer, ByVal
     sd_logic As Integer, ByVal sd_latch As
```

```
        Integer, ByVal sd_mode As Integer) As
        Integer
B_8158_enable_sd(ByVal AxisNo As Integer, ByVal
        Enable As Integer) As Integer
B_8158_set_limit_logic(ByVal AxisNo As Integer,
        ByVal Logic As Integer) As Integer
B_8158_set_limit_mode(ByVal AxisNo As Integer,
        ByVal limit_mode As Integer) As Integer
I16 _8158_get_io_status(ByVal AxisNo As Integer,
        io_sts As Integer) As Integer
```

@ 요지

AxisNo: 목표 축의 축 번호.

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

on_off: SVON 신호의 ON-OFF 상태

값	의미
0	ON
1	OFF

pcs_logic: PCS 신호 입력 논리

값	의미
0	부정 논리
1	긍정 논리

enable: 활성화 또는 비활성화

값	의미
0	비활성화
1	활성화

clr_mode: CLR 입력 클리어 모드 지정

clr_mode = 0 , 하강 엣지에서 클리어 (기본값)

clr_mode = 1 , 상승 엣지에서 클리어

clr_mode = 2 , LOW 레벨에서 클리어

clr_mode = 3 , HIGH 레벨에서 클리어

targetCounterInBit: 비트의 클리어 목표 카운터 활성화 / 비활성화

값	의미
비트	설명
0	CLR 입력 켜졌을 경우 명령 카운터 리셋
1	CLR 입력 켜졌을 경우 위치 카운터 리셋
2	CLR 입력 켜졌을 경우 에러 카운터 리셋
3	CLR 입력 켜졌을 경우 범용 카운터 리셋

inp_enable: INP 함수 활성화 / 비활성화

inp_enable = 0, 비활성화 (기본값)

inp_enable = 1, 활성화

inp_logic: INP 신호에 대한 활성 논리 설정

값	의미
0	부정 논리
1	긍정 논리

alm_logic: ALARM 신호에 대한 활성 논리 설정

값	의미
0	부정 논리
1	긍정 논리

alm_mode: ALARM 신호 받을 시 반응 모드

값	의미
0	모터 즉시 정지 (기본값)
1	정지를 위한 모터 감속

erc_logic: ERC 신호의 활성 논리 설정

값	의미
0	부정 논리
1	긍정 논리

erc_pulse_width: ERC 신호의 펄스 폭 설정

값	의미
0	12 ms
1	102 ms
2	409 ms
3	1.6 ms
4	13 ms
5	52 ms
6	104 ms
7	레벨 출력

erc_mode:

값	의미
0	비활성화
1	ERCEL, ALM 또는 EMG 입력으로 인해 정지하였을 경우 ERC 출력
2	홈 리턴 완료하였을 시 ERC 출력
3	위의 두 상황 모두 ERC 출력

sd_logic:

값	의미
0	부정 논리
1	긍정 논리

sd_latch: SD 신호에 대한 latch 제어 설정

값	의미
0	latch 안 함
1	latch

sd_mode: SD 신호의 반응 모드 설정

값	의미
0	단지 slow down
1	slow down 후 정지

enable: 고속 피드를 위한 ramping-down 지점 설정

Logic: PEL/MEL 논리 설정 .

값	의미
0	Normal low(normal open)
1	Normal high(normal close)

limit_mode:

값	의미
0	즉시 정지
1	감속 후 정지

***io_sts:** 입 / 출력 위상 . 6.12 의 함수 설명을 참고하십시오 .

6.13 인터럽트 제어

@ 명칭

_8158_int_control – INT 서비스 활성화 / 비활성화

_8158_set_motion_int_factor – 모션 관련 인터럽트의 요소를 설정

_8158_wait_error_interrupt – 에러 관련 인터럽트를 대기

_8158_wait_motion_interrupt – 모션 관련 인터럽트를 대기

@ 설명

_8158_int_control:

이 함수는 주 PC로 윈도우 인터럽트 이벤트를 활성화하는데 사용됩니다 .

_8158_set_motion_int_factor:

이 함수는 사용자가 이벤트 int 를 시작하는 모션관련 요소를 선택하는 것을 가능하게 합니다 . _8158_int_control() 에 의해 인터럽트 서비스가 활성화된 경우 에러는 절대 마스크 할 수 없으며 인터럽트 함수가 활성화된 이후에 사용자는 _8158_wait_motion_interrupt()를 이벤트를 기다리는 데 사용할 수 있습니다 .

_8158_wait_error_interrupt:

사용자가 _8158_int_control()로 인터럽트 함수를 활성화한 경우 이 함수를 에러 인터럽트를 대기하도록 하는 데 사용할 수 있습니다 . 자세한 동작 이론은 4.8 절을 참고 하십시오 .

_8158_wait_motion_interrupt:

사용자가 _8158_int_control()로 인터럽트 함수를 활성화했거나 _8158_set_motion_int_factor() 로 인터럽트 요소를 설정한 경우 사용자는 이 함수를 지정된 인터럽트를 대기하게 하는 데 사용할 수 있습니다 . 이 함수가 동작 중 일 경우 이벤트가 트리거되거나 함수의 시간이 초과할 때 까지 공정은 절대 멈추지 않을 것입니다 .

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_int_control(I16 card_id, I16 intFlag);
I16 _8158_set_motion_int_factor(I16 AxisNo, U32
    int_factor );
I16 _8158_wait_error_interrupt(I16 AxisNo, I32
    TimeOut_ms );
I16 _8158_wait_motion_interrupt(I16 AxisNo, I16
    IntFactorBitNo, I32 TimeOut_ms );
```

Visual Basic (Windows 2000/XP)

```
B_8158_int_control(ByVal card_id As Integer,
    ByVal intFlag As Integer) As Integer
B_8158_wait_error_interrupt(ByVal AxisNo As
    Integer, ByVal TimeOut_ms As Long) As
    Integer
B_8158_wait_motion_interrupt(ByVal AxisNo As
    Integer, ByVal IntFactorBitNo As Integer,
    ByVal TimeOut_ms As Long) As Integer
B_8158_set_motion_int_factor(ByVal AxisNo As
    Integer, ByVal int_factor As Long) As
    Integer
```

@ 인수

card_id: 목표 PCI-8158 카드의 인덱스를 지정. 카드 ID는 DIP 스위치 (SW1) 또는 슬롯의 순서로 정해집니다. _8158_initial() 을 참고 하십시오.

intFlag: 인터럽트 함수 활성화 / 비활성화

값	의미
0	비활성화
1	활성화

AxisNo: 목표 축의 축 번호 .

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

int_factor: 인터럽트 요소

모션 INT 요소

값	의미 (0: 비활성화 , 1: 활성화)
비트	설명
0	정상적 정지
1	버퍼의 다음 명령 시작
2	명령 pre-register 2 비어있음 , 다음 명령 입력 허용
3	(보류) (항상 0 으로 설정)
4	가속 시작
5	가속 정지
6	감속 시작
7	감속 정지
8	+ Soft limit 또는 콤퍼레이터 1 켜짐
9	-Soft limit 또는 콤퍼레이터 2 켜짐
10	에러 콤퍼레이터 또는 콤퍼레이터 3 켜짐
11	일반적 콤퍼레이터 또는 콤퍼레이터 4 켜짐
12	트리거 콤퍼레이터 또는 콤퍼레이터 5 켜짐
13	CLR 입력에 의해 카운터 리셋
14	LTC 입력에 의해 카운터 latch
15	ORG 입력에 의해 카운터 latch
16	SD 입력 켜짐
17	(보류) (항상 0 으로 설정)
18	CSTA 입력 또는 _8158_start_move_all() 켜짐
19~31	정의되지 않음 (항상 0 으로 설정)

TimeOut_ms: 밀리초 단위의 time-out 간격 지정. 만약 TimeOut_ms 가 영 (0) 이라면 함수는 지정된 대상의 상태를 테스트하고 즉시 복귀할 것이며 TimeOut_ms 가 -1 이라면 함수의 time-out 간격은 절대 흐르지 않습니다 (무한)

IntFactorBitNo: INT 요소의 비트 번호 지정.

예를 들어 . IntFactorBitNo = 4, 는 " 가속 시작 " 인터럽트의 요소를 대기 중인 것을 의미합니다 .

6.14 위치 제어와 카운터

@ 명칭

_8158_get_position – 피드백 위치 카운터의 값을 불러옴

_8158_set_position – 피드백 위치 카운터 설정

_8158_get_command – 명령 위치 카운터의 값을 불러옴

_8158_set_command – 명령 위치 카운터 설정

_8158_get_error_counter – 위치 에러 카운터의 값을 불러옴

_8158_reset_error_counter – 위치 에러 카운터 리셋

_8158_get_general_counter – 일반적 카운터의 값을 불러옴

_8158_set_general_counter – 일반적 카운터 설정

_8158_get_target_pos – 목표 위치 리코더의 값을 불러옴

_8158_reset_target_pos – 목표 위치 리코더 리셋

_8158_get_res_distance – 모션에서 축적된 남은 펄스를 불러옴

_8158_set_res_distance – 남아있는 펄스 리코더 설정

@ 설 명

_8158_get_position:

이 함수는 피드백 위치 카운터 값을 읽기 위해 사용됩니다. 이 값은 이미 _8158_set_move_ratio() 에 의해 설정된 이동 비율을 통해 처리되었으며 이동 비율이 0.5 일 경우 위치값은 두 배가 됩니다. 피드백 카운터의 소스는 외부의 EA/EB 또는 PCI-8158 내부 펄스로 출력되는 _8158_set_feedback_src() 함수로 선택 가능합니다.

_8158_set_position:

이 함수는 피드백 위치 카운터를 지정된 값으로 변경하는데 사용됩니다. 값은 이동 비율에 의해 처리되도록 설정될 것이며 이동비율이 0.5 일 경우 설정값은 주어진 값의 두 배가 될것 입니다.

_8158_get_command:

이 함수는 명령 위치 카운터의 값을 읽는데 사용되며 명령 위치 카운터의 소스는 PCI-8158 의 펄스 출력입니다 .

_8158_set_command:

이 함수는 명령 위치 카운터의 값을 변경하는데 사용됩니다 .

_8158_get_error_counter:

이 함수는 위치 에러 카운터의 값을 읽는데 사용됩니다 .

_8158_reset_error_counter:

이 함수는 위치 에러 카운터를 클리어하는데 사용됩니다 .

_8158_get_general_counter:

이 함수는 일반적 카운터의 값을 읽는데 사용됩니다 .

_8158_set_general_counter:

이 함수는 카운팅 소스를 설정하고 일반적 카운터의 값을 변경하는데 사용됩니다 . (펄스 입력은 소스의 기본 값입니다 .)

_8158_get_target_pos:

이 함수는 목표 위치 리코더의 값을 읽는데 사용되며 목표 위치 리코더는 PCI-8158 소프트웨어 드라이버에 의해 유지됩니다 . 이는 현재의 운영 중인 모션에 대한 정착 위치를 기록합니다 .

_8158_reset_target_pos:

이 함수는 목표 위치 리코더에 대한 새로운 값을 설정하는데 사용됩니다. 이 함수의 호출은 홈 리턴이 완료되었거나 새로운 피드백 카운터 값이 _8158_set_position() 함수에 의해 설정되었을 때 필요합니다 .

_8158_get_res_distance:

이 함수는 잔여 거리 리코더의 값을 읽는데 사용되며 목표 위치 리코더는 PCI-8158 소프트웨어 드라이버에 의해 유지됩니다. 이는 현재의 운영 중인 모션에 대한 정착 위치를 기록합니다.

_8158_set_res_distance:

이 함수는 잔여 거리 카운터의 값을 변경하는데 사용됩니다.

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_get_position(I16 AxisNo, F64 *Pos);
I16 _8158_set_position(I16 AxisNo, F64 Pos);
I16 _8158_get_command(I16 AxisNo, I32 *Command);
I16 _8158_set_command(I16 AxisNo, I32 Command);
I16 _8158_get_error_counter(I16 AxisNo, I16
     *error);
I16 _8158_reset_error_counter(I16 AxisNo);
I16 _8158_get_general_counter(I16 AxisNo, F64
     *CntValue);
I16 _8158_set_general_counter(I16 AxisNo, I16
     CntSrc, F64 CntValue);
I16 _8158_get_target_pos(I16 AxisNo, F64 *T_pos);
I16 _8158_reset_target_pos(I16 AxisNo, F64
     T_pos);
I16 _8158_get_res_distance(I16 AxisNo, F64
     *Res_Distance);
I16 _8158_set_res_distance(I16 AxisNo, F64
     Res_Distance);
```

Visual Basic (Windows 2000/XP)

```
B_8158_get_position(ByVal AxisNo As Integer, Pos
     As Double) As Integer
B_8158_set_position(ByVal AxisNo As Integer,
     ByVal Pos As Double) As Integer
B_8158_get_command(ByVal AxisNo As Integer, Cmd
     As Long) As Integer
B_8158_set_command(ByVal AxisNo As Integer, ByVal
     Cmd As Long) As Integer
B_8158_get_error_counter(ByVal AxisNo As Integer,
     ByRef error As Integer) As Integer
```

```
B_8158_reset_error_counter(ByVal AxisNo As
      Integer) As Integer
B_8158_set_general_counter(ByVal AxisNo As
      Integer, ByVal CntSrc As Integer, ByVal
      CntValue As Double) As Integer
B_8158_get_general_counter(ByVal AxisNo As
      Integer, ByRef Pos As Double) As Integer
B_8158_reset_target_pos(ByVal AxisNo As Integer,
      ByVal Pos As Double) As Integer
B_8158_get_target_pos(ByVal AxisNo As Integer,
      ByRef Pos As Double) As Integer
B_8158_set_res_distance(ByVal AxisNo As Integer,
      ByVal Res_Distance As Double) As Integer
B_8158_get_res_distance(ByVal AxisNo As Integer,
      ByRef Res_Distance As Double) As Integer
```

@ 인수

AxisNo: 목표 축의 축 번호 .

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Pos, *Pos: 피드백 위치 카운터 값 , (_8158_get/set_position)

범위 : -134217728-134217727

Cmd, *Cmd: 명령 위치 카운터 값 ,

범위 : -134217728-134217727

***error:** 위치 에러 카운터 값 ,

범위 : -32768-32767

CntSrc: 일반적 카운터 소스

값	의미
0	명령 펄스
1	EA/EB
2	펄스 입력
3	시스템 클럭 ÷2

CntValue, *CntValue: 카운터 값

TargetPos, *TargetPos: 목표 위치 리코더 값,

범위 : -134217728-134217727

ResDistance, *ResDistance: 잔여 거리

6.15 위치 비교 및 latch

@ 명칭

_8158_set_trigger_logic – CMP 신호 논리 설정

_8158_set_trigger_comparator – 트리거 콤퍼레이터 설정

_8158_set_error_comparator – 에러 콤퍼레이터 설정

_8158_set_general_comparator – 일반적 콤퍼레이터 설정

_8158_set_latch_source – 카운터를 위한 latch 타이밍 설정

_8158_set_ltc_logic – LTC 신호의 논리 설정

_8158_get_latch_data – latch 데이터 불러옴

@ 설명

_8158_set_trigger_logic:

이 함수는 단일 CMP 의 논리를 설정하는데 사용됩니다 .

_8158_set_error_comparator:

이 함수는 비교방법과 에러 콤퍼레이터의 값을 설정하는데 사용됩니다 . 위치 에러 카운터의 값이 비교 값에 도달했을 경우 8158 은 주 PC 로 인터럽트를 발생합니다 . 6.14 의 “ 인터럽트 제어 ” 부분을 참고하십시오 .

_8158_set_general_comparator:

이 함수는 비교 소스 카운터 , 비교 방법 , 일반적 콤퍼레이터의 값을 설정하는데 사용됩니다 . 비교 조건이 충족되었을 경우 4 가지 반응 중 하나가 수행될 것입니다 . 자세한 설정은 인수 설명을 참고하십시오 .

_8158_set_trigger_comparator:

이 함수는 비교 소스 카운터 , 비교 방법 , 트리거 콤퍼레이터의 값을 설정하는데 사용됩니다 . 비교 소스 카운터의 값이 비교값에 도달하였을 경우 8158 은 CMP 를 통해 펄스를 발생하고 인터럽트 (event_int_status, bit 12) 또한 주 PC 로 전송될 것입니다 .

_8158_set_latch_source:

본 기계는 4 가지의 latch 트리거링 소스가 있으며 이 함수를 사용하여 사용자는 카운터의 데이터로 latch 하기 위한 이벤트 소스를 선택할 수 있습니다 .

_8158_set_ltc_logic:

이 함수는 latch 입력의 논리를 설정하는데 사용됩니다 .

_8158_get_latch_data:

latch 신호가 도달한 후에 이 함수는 카운터의 latch 된 값을 읽는데 사용됩니다 .

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_set_trigger_logic(I16 AxisNo, I16
     Logic);
I16 _8158_set_error_comparator(I16 AxisNo, I16
     CmpMethod, I16 CmpAction, I32 Data);
I16 _8158_set_general_comparator(I16 AxisNo, I16
     CmpSrc, I16 CmpMethod, I16 CmpAction, I32
     Data);
I16 _8158_set_trigger_comparator(I16 AxisNo, I16
     CmpSrc, I16 CmpMethod, I32 Data);
I16 _8158_set_latch_source(I16 AxisNo, I16
     LtcSrc);
I16 _8158_set_ltc_logic(I16 AxisNo, I16
     LtcLogic);
I16 _8158_get_latch_data(I16 AxisNo, I16
     CounterNo, F64 *Pos);
```

Visual Basic (Windows 2000/XP)

```
B_8158_set_trigger_logic(ByVal AxisNo As Integer,
     ByVal Logic As Integer) As Integer
B_8158_set_error_comparator(ByVal AxisNo As
     Integer, ByVal CmpMethod As Integer, ByVal
     CmpAction As Integer, ByVal Data As Long) As
     Integer
B_8158_set_general_comparator(ByVal AxisNo As
     Integer, ByVal CmpSrc As Integer, ByVal
     CmpMethod As Integer, ByVal CmpAction As
     Integer, ByVal Data As Long) As Integer
```

```
B_8158_set_trigger_comparator(ByVal AxisNo As
      Integer, ByVal CmpSrc As Integer, ByVal
      CmpMethod As Integer, ByVal Data As Long) As
      Integer
B_8158_set_latch_source(ByVal AxisNo As Integer,
      ByVal LtcSrc As Integer) As Integer
B_8158_set_ltc_logic(ByVal AxisNo As Integer,
      ByVal StcLogic As Integer) As Integer
B_8158_get_latch_data(ByVal AxisNo As Integer,
      ByVal CounterNo As Integer, Pos As Double)
      As Integer
```

@ 인수

AxisNo: 목표 축의 축 번호.

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Logic: 트리거 비교 논리

값	의미
0	부정 논리
1	긍정 논리

CmpSrc: 비교 소스 카운터

값	의미
0	명령 카운터
1	피드백 카운터
2	에러 카운터
3	일반적 카운터

CmpMethod: 비교 방법

값	의미
0	비교 안함 (비활성화)
1	데이터 = 소스 카운터 (독립적인 방향)
2	데이터 = 소스 카운터 (오직 Count up)
3	데이터 = 소스 카운터 (오직 Count down)
4	데이터 > 소스 카운터
5	데이터 < 소스 카운터

Data: 비교값 (위치)

CmpAction:

값	의미
0	동작안함
1	즉시 정지
2	감속 후 정지

ltc_src:

값	의미
0	LTC 핀 입력
1	ORG 핀 입력
2	일반적 콤퍼레이터 조건 충족
3	트리거 콤퍼레이터 조건 충족

ltc_logic: LTC 신호 동작 엣지

값	의미
0	부정 논리
1	긍정 논리

CounterNo: latch 할 카운터 지정

값	의미
0	명령 카운터
1	피드백 카운터
2	에러 카운터
3	일반적 카운터

***Pos:** Latch 데이터 (위치)

6.16 연속적 모션

@ 명칭

_8158_set_continuous_move – 절대 모션을 위한 연속적 모션 활성화

_8158_check_continuous_buffer – 버퍼가 비어있는지 확인

_8158_dwell_move – dwell 동작 설정

@ 설명

_8158_set_continuous_move:

이 함수는 연속적 모션 명령의 일련의 과정 전후에 필요합니다.

_8158_check_continuous_buffer:

이 함수는 명령 pre-register(버퍼) 가 비어있는지 확인하는데 사용됩니다 . 명령 pre-register(버퍼) 가 비어있다면 사용자는 다음 모션 명령을 입력할 수 있습니다 . 그렇지 않으면 새로운 명령은 두 번째 명령 pre-register 의 이전 명령에 덮어 쓰여지게 될 것입니다 . 만약 복귀 코드가 1 이라면 버퍼는 가득 차 있는 것이고 복귀 코드가 0 이라면 버퍼는 가득 차 있지 않은 것입니다 .

_8158_dwell_move:

이 함수는 dwell 동작을 시작하는데 사용되고 이는 동작이 지정된 시간에 대하여 실제 동작을 하지 않는 것을 의미합니다 .

예시 :

```
_8158_set_continuous_move( 2, 1 ); // 연속 운동 시작
_8158_start_tr_move( 2, 20000.0, 10.0, 10000.0,
     0.1, 0.1);
_8158_dwell_move( 2, 2000); //2 초 간 dwell 동작 시작
_8158_start_sr_move( 2, 20000.0, 10.0, 10000.0,
     0.1, 0.1, 0, 0 );
_8158_set_continuous_move( 2, 0 ); // 연속 운동 정지
```

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_set_continuous_move(I16 AxisNo, I16
     Enable);
I16 _8158_check_continuous_buffer(I16 AxisNo);
I16 _8158_dwell_move(I16 AxisNo, F64 miniSecond);
```

Visual Basic (Windows 2000/XP)

```
B_8158_set_continuous_move(ByVal AxisNo As
     Integer, ByVal Enable As Integer) As Integer
B_8158_check_continuous_buffer(ByVal AxisNo As
     Integer) As Integer
B_8158_dwell_move(ByVal AxisNo As Integer, ByVal
     miniSecond As Double) As Integer
```

@ 인수

AxisNo: 목표 축의 축 번호 .

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Enable: 연속 모션 스위치 논리

값	의미
0	연속 모션 전 과정 완료 (비활성화)
1	연속 모션 전 과정 시작 (활성화)

millisecond: 밀리초 (ms) 단위의 dwell 동작 시간 .

6.17 다축의 동시 동작

@ 명칭

_8158_set_tr_move_all – 다축 동시 동작 설치

_8158_set_ta_move_all – 다축 동시 동작 설치

_8158_set_sr_move_all – 다축 동시 동작 설치

_8158_set_sa_move_all – 다축 동시 동작 설치

_8158_start_move_all – 다축 trapezoidal 프로파일 모션 시작

_8158_stop_move_all – 다축 모션 동시 정지

@ 설명

이 함수는 다른 카드라 할지라도 여러 축을 동시에 동작하는 것과 관련된 함수입니다. 동시 다축 동작은 지정된 축이 동시에 시작하거나 정지하는 것을 의미하며 축은 파라미터 “AxisArray,” 에 의해 지정되어 움직이고 축의 수는 _8158_set_tr_move_all() 의 파라미터 “TotalAxes” 에 의해 정의됩니다.

_8158_set_xx_move_all() 가 정확히 설치된 경우, 함수 _8158_start_move_all()은 모든 지정된 축을 trapezoidal 상대 운동 시키며 _8158_stop_move_all() 으로 정지시킬 수 있습니다. 두 함수는 모든 지정된 축을 동시에 시작/정지할 수 있습니다. 두 함수가 필요한 경우 2.6 절에 따라 연결이 필요합니다.

다음의 코드는 이 함수들을 어떻게 사용할 것인지에 관한 데모이며 이 코드는 각각 축 0 와 축 1 을 거리 80000.0 와 120000.0 로 이동시킵니다. 만약 속도와 가속도를 거리 비율에 비례하여 선택하였을 경우 축은 동시에 정지점에 도착하게 됩니다.

[예시]

```
I16 axes[2] = {0, 1};
F64 dist[2] = {80000.0, 120000.0},
F64 str_vel[2] = {0.0, 0.0},
F64 max_vel[2] = {4000.0, 6000.0},
F64 Tacc[2] = {0.1, 0.6},
F64 Tdec[2] = {0.1, 0.6};
```

```
_8158_set_tr_move_all(2, axes, dist, str_vel,
     max_vel, Tacc, Tdec);
_8158_start_move_all(axes[0]);
```

@ 문법

C/C++(Windows 2000/XP)

```
I16 _8158_set_tr_move_all(I16 TotalAxes, I16
     *AxisArray, F64 *DistA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA);
I16 _8158_set_ta_move_all(I16 TotalAx, I16
     *AxisArray, F64 *PosA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA);
I16 _8158_set_sr_move_all(I16 TotalAx, I16
     *AxisArray, F64 *DistA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA, F64
     *SVaccA, F64 *SVdecA);
I16 _8158_set_sa_move_all(I16 TotalAx, I16
     *AxisArray, F64 *PosA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA, F64
     *SVaccA, F64 *SVdecA);
I16 _8158_start_move_all(I16 FirstAxisNo);
I16 _8158_stop_move_all(I16 FirstAxisNo);
```

Visual Basic (Windows 2000/XP)

```
B_8158_set_tr_move_all(ByVal TotalAxes As
     Integer, ByRef AxisArray As Integer, ByRef
     DistA As Double, ByRef StrVelA As Double,
     ByRef MaxVelA As Double, ByRef TaccA As
     Double, ByRef TdecA As Double) As Integer
B_8158_set_sa_move_all(ByVal TotalAxes As
     Integer, ByRef AxisArray As Integer, ByRef
     PosA As Double, ByRef StrVelA As Double,
     ByRef MaxVelA As Double, ByRef TaccA As
     Double, ByRef TdecA As Double, ByRef SVaccA
     As Double, ByRef SVdecA As Double) As
     Integer
B_8158_set_ta_move_all(ByVal TotalAxes As
     Integer, ByRef AxisArray As Integer, ByRef
     PosA As Double, ByRef StrVelA As Double,
     ByRef MaxVelA As Double, ByRef TaccA As
     Double, ByRef TdecA As Double) As Integer
```

```
B_8158_set_sr_move_all(ByVal TotalAxes As
      Integer, ByRef AxisArray As Integer, ByRef
      DistA As Double, ByRef StrVelA As Double,
      ByRef MaxVelA As Double, ByRef TaccA As
      Double, ByRef TdecA As Double, ByRef SVaccA
      As Double, ByRef SVdecA As Double) As
      Integer
B_8158_start_move_all(ByVal FirstAxisNo As
      Integer) As Integer
B_8158_stop_move_all(ByVal FirstAxisNo As
      Integer) As Integer
```

@ 인수

TotalAxes: 동시 모션에 대한 축의 수

***AxisArray:** 이동을 위해 지정된 축의 번호 배열

***DistA:** 펄스 단위의 지정된 거리 배열

***StrVelA:** PPS 단위의 시작 속도 배열

***MaxVelA:** PPS 단위의 최대 속도 배열

***TaccA:** 초 단위의 가속 시간 배열

***TdecA:** 초 단위의 감속 시간 배열

***PosA:** 펄스 단위의 지정된 위치 배열

***SvaccA:** S-커브 가속 수행 시 지정된 속도 간격 배열.

***SvdecA:** S-커브 감속 수행 시 지정된 속도 간격 배열.

FirstAxisNo: AxisArray 의 첫 번째 요소.

6.18 범용 DIO

@ 명칭

_8158_set_gpio_output – 디지털 출력 설정

_8158_get_gpio_output – 디지털 출력 불러옴

_8158_get_gpio_input – 디지털 입력 불러옴

_8158_set_gpio_input_function – 모든 디지털 입력에 신호 타입 설정

@ 설명

_8158_set_gpio_output:

PCI-8158 은 8 개의 디지털 출력 채널을 가지고 있습니다. 이 함수로 사용자는 디지털 출력을 제어할 수 있습니다.

_8158_get_gpio_output:

이 함수는 디지털 출력 상태를 불러올 때 사용됩니다.

_8158_get_gpio_input:

PCI-8158 은 8 개의 디지털 입력 채널을 가지고 있습니다. 이 함수로 사용자는 입력 상태를 불러올 수 있습니다.

_8158_set_gpio_input_function:

PCI-8158 은 8 개의 디지털 입력 채널을 가지고 있습니다. 이 함수로 사용자는 어떠한 지정된 DI 채널로 여러 입력 신호 중 하나를 설정할 수 있으며 입력 신호는 LTCn, SDn, PCSn, CLRn, EMG(지표어와 축 인덱스) 를 포함합니다.

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_set_gpio_output(I16 card_id, I16
     DoValue);
I16 _8158_get_gpio_output(I16 card_id, I16 *
     DoValue);
I16 _8158_get_gpio_input(I16 card_id, I16 *
     DiValue);
```

```
I16 _8158_set_gpio_input_function(I16 card_id,
    I16 Channel, I16Select, I16 Logic);
```

Visual Basic (Windows 2000/XP)

```
B_8158_set_gpio_output(ByVal card_id As Integer,
    ByVal DoValue As Integer) As Integer
B_8158_get_gpio_output(ByVal card_id As Integer,
    DoValue As Integer) As Integer
B_8158_get_gpio_input(ByVal card_id As Integer,
    DiValue As Integer) As Integer
B_8158_set_gpio_input_function(ByVal card_id As
    Integer, ByVal Channel As Integer, ByVal
    Select As Integer, ByVal Logic As Integer)As
    Integer
```

@ 인수

card_id: PCI-8158 카드 인덱스 지정 . 카드 ID 는 DIP 스위치 (SW1) 또는 슬롯의 순서로 결정됩니다 . _8158_initial() 을 참고하십시오 .

DoValue, *DoValue: 디지털 출력 값 . 비트 0-7: D_out0-7.

***DiValue:** 디지털 입력 값 , 비트 0-7: D_in0-7

Channel: 디지털 채널 DI0 - DI7

Select: 신호 타입 선택

값	의미
0	일반적 DI (기본값)
1	LTC (active low)
2	SD (active low)
3	PCS (active low)
4	CLR (active low)
5	EMG (active low)

Logic: 입력 신호 논리

값	의미
0	역방향 아님 (기본값)
1	역방향

6.19 Soft Limit

@ 명칭

_8158_disable_soft_limit – soft limit 기능 비활성화

_8158_enable_soft_limit – soft limit 기능 활성화

_8158_set_soft_limit – soft limit 설정

@ 설명

_8158_disable_soft_limit:

이 함수는 soft limit 기능을 비활성화하는데 사용됩니다 .

_8158_enable_soft_limit:

이 함수는 soft limit 기능을 활성화하는 데 사용됩니다 . 활성화 된 후 soft limit 동작은 물리적 한계와 정확히 같아질 것입니다 .

_8158_set_soft_limit:

이 함수는 soft limit 값을 설정하는데 사용됩니다 .

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_disable_soft_limit(I16 AxisNo);
I16 _8158_enable_soft_limit(I16 AxisNo, I16
     Action);
I16 _8158_set_soft_limit(I16 AxisNo, I32
     PlusLimit, I32 MinusLimit);
```

Visual Basic (Windows 2000/XP)

```
B_8158_disable_soft_limit(ByVal AxisNo As
     Integer) As Integer
B_8158_enable_soft_limit(ByVal AxisNo As Integer,
     ByVal Action As Integer) As Integer
B_8158_set_soft_limit(ByVal AxisNo As Integer,
     ByVal PlusLimit As Long, ByVal MinusLimit As
     Long) As Integer
```

@ 인수

AxisNo: 목표 축의 축 번호 .

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Action: soft limit 의 반응 방법

값	의미
0	오직 INT
1	즉시 정지
2	감속 후 정지

PlusLimit: Soft limit 값 , 양의 방향

MinusLimit: Soft limit 값 , 음의 방향

6.20 Backlash 보정 / 진동 억제

@ 명칭

_8158_backlash_comp – 보정을 위한 backlash 조정 펄스 설정

_8158_suppress_vibration – 진동 억제를 위한 유휴 펄스 카운트 설정

_8158_set_fa_speed – FA 속도 설정

@ 설명

_8158_backlash_comp:

방향이 변할 때 마다 8158 은 명령이 전송되기 전 backlash 조정 펄스를 출력합니다. 이 함수는 조정 펄스의 수를 설정하는 데 사용됩니다.

_8158_suppress_vibration:

이 함수는 명령 운동의 완료 직후 양의 방향에 대한 단일 펄스와 음의 방향에 대한 단일 펄스의 출력으로 인한 기계 시스템의 진동 억제를 위하여 사용됩니다.

_8158_set_fa_speed:

이 함수는 backlash 보정 또는 slip 보정에 대한 낮은 속도를 지정하는 데 사용됩니다. 또한, 홈 리턴 동작에 대한 반대의 낮은 속도에도 사용됩니다.

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_backlash_comp(I16 AxisNo, I16
     CompPulse, I16 Mode);
I16 _8158_suppress_vibration(I16 AxisNo, U16
     ReverseTime,
U16 ForwardTime);
I16 _8158_set_fa_speed(I16 AxisNo, F64 FA_Speed);
```

Visual Basic (Windows 2000/XP)

```
B_8158_backlash_comps (ByVal AxisNo As Integer,
     ByVal CompPulse As Integer, ByVal Mode As
     Integer) As Integer
```

```
B_8158_suppress_vibration(ByVal AxisNo As
      Integer, ByVal ReverseTime As Integer, ByVal
      ForwardTime As Integer) As Integer
B_8158_set_fa_speed(ByVal AxisNo As Integer,
      ByVal FA_Speed As Double) As Integer
```

@ 인수

AxisNo: 목표축의 축 번호 .

카드 id	물리적 축	축번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

CompPulse: 보정 펄스의 지정된 번호 , 12 비트

Mode:

값	의미
0	꺼짐
1	backlash 보정 활성화
2	Slip 보정

ReverseTime: 지정된 후진 시간 , 0 – 65535, 단위 1.6 us

ForwardTime: 지정된 전진 시간 , 0 – 65535, 단위 1.6 us

FA_Speed: fa 속도 (단위 : 펄스 / 초)

6.21 속도 프로파일 연산

@ 명칭

_8158_get_tr_move_profile – 상대적인 trapezoidal 속도 프로파일을 불러옴

_8158_get_ta_move_profile – 절대적인 trapezoidal 속도 프로파일을 불러옴

_8158_get_sr_move_profile – 상대적인 S- 커브 속도 프로파일을 불러옴

_8158_get_sa_move_profile – 절대적인 S- 커브 속도 프로파일을 불러옴

@ 설명

_8158_get_tr_move_profile:

이 함수는 상대적인 trapezoidal 속도 프로파일을 불러오는데 사용됩니다. 이 함수로 사용자는 동작 전 실제 속도 프로파일을 얻을 수 있습니다.

_8158_get_ta_move_profile:

이 함수는 절대적인 trapezoidal 속도 프로파일을 불러옵니다. 이 함수를 통해 사용자는 동작 전 실제 속도 프로파일을 얻을 수 있습니다.

_8158_get_sr_move_profile:

이 함수는 상대적인 S- 커브 속도 프로파일을 불러오는데 사용됩니다. 이 함수로 사용자는 동작 전 실제 속도 프로파일을 얻을 수 있습니다.

_8158_get_sa_move_profile:

이 함수는 절대적인 S- 커브 속도 프로파일을 불러오는데 사용됩니다. 이 함수로 사용자는 동작 전 실제 속도 프로파일을 얻을 수 있습니다.

@ 문법

C/C++ (Windows 2000/XP)

```
I16 _8158_get_tr_move_profile(I16 AxisNo, F64
      Dist, F64 StrVel, F64 MaxVel, F64 Tacc, F64
      Tdec, F64 *pStrVel, F64 *pMaxVel, F64
      *pTacc, F64 *pTdec, F64 *pTconst );
I16 _8158_get_ta_move_profile(I16 AxisNo, F64
      Pos, F64 StrVel, F64 MaxVel, F64 Tacc, F64
      Tdec, F64 *pStrVel, F64 *pMaxVel, F64
      *pTacc, F64 *pTdec, F64 *pTconst );
I16 _8158_get_sr_move_profile(I16 AxisNo, F64
      Dist, F64 StrVel, F64 MaxVel, F64 Tacc, F64
      Tdec, F64 SVacc, F64 SVdec,F64 *pStrVel, F64
      *pMaxVel, F64 *pTacc, F64 *pTdec, F64
      *pSVacc, F64 *pSVdec, F64 *pTconst);
I16 _8158_get_sa_move_profile(I16 AxisNo, F64
      Pos, F64 StrVel, F64 MaxVel, F64 Tacc, F64
      Tdec, F64 SVacc, F64 SVdec,F64 *pStrVel, F64
      *pMaxVel, F64 *pTacc, F64 *pTdec, F64
      *pSVacc, F64 *pSVdec, F64 *pTconst);
```

Visual Basic (Windows 2000/XP)

```
B_8158_get_tr_move_profile(ByVal AxisNo As
      Integer, ByVal Dist As Double, ByVal StrVel
      As Double, ByVal MaxVel As Double, ByVal
      Tacc As Double, ByVal Tdec As Double, ByRef
      pStrVel As Double, ByRef pMaxVel As Double,
      ByRef pTacc As Double, ByRef pTdec As
      Double, ByRef pTconst As Double) As Integer
B_8158_get_ta_move_profile(ByVal AxisNo As
      Integer, ByVal Pos As Double, ByVal StrVel
      As Double, ByVal MaxVel As Double, ByVal
      Tacc As Double, ByVal Tdec As Double, ByRef
      pStrVel As Double, ByRef pMaxVel As Double,
      ByRef pTacc As Double, ByRef pTdec As
      Double, ByRef pTconst As Double) As Integer
B_8158_get_sr_move_profile(ByVal AxisNo As
      Integer, ByVal Dist As Double, ByVal StrVel
      As Double, ByVal MaxVel As Double, ByVal
      Tacc As Double, ByVal Tdec As Double, ByVal
      SVacc As Double, ByVal SVdec As Double,
      ByRef pStrVel As Double, ByRef pMaxVel As
```

```
        Double, ByRef pTacc As Double, ByRef pTdec
        As Double, ByRef pSVacc As Double, ByRef
        pSVdec As Double, ByRef pTconst As Double)
        As Integer
    B_8158_get_sa_move_profile(ByVal AxisNo As
        Integer, ByVal Pos As Double, ByVal StrVel
        As Double, ByVal MaxVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double, ByVal
        SVacc As Double, ByVal SVdec As Double,
        ByRef pStrVel As Double, ByRef pMaxVel As
        Double, ByRef pTacc As Double, ByRef pTdec
        As Double, ByRef pSVacc As Double, ByRef
        pSVdec As Double, ByRef pTconst As Double)
        As Integer
```

@ 인수

AxisNo: 목표 축의 축 번호 .

카드 id	물리적 축	축 번호
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Dist: 지정된 상대 거리 (단위 : 펄스)

Pos: 지정된 절대 거리 (단위 : 펄스)

StrVel: 시작 속도 (단위 : 펄스 / 초)

MaxVel: 최대 속도 (단위 : 펄스 / 초)

Tacc: 가속 시간 (단위 : 초)

Tdec: 감속 시간 (단위 : 초)

SVacc: 가속 시 S- 커브 범위 (단위 : 펄스 / 초)

참고 : SVacc = 0, 은 순수 S- 커브 . 자세한 내용은 4.2.4 를 참고 하십시오 .

SVdec: 감속 시 S- 커브 범위 (단위 : 펄스 / 초)

참고 : SVdec = 0, 은 순수 S- 커브 . 자세한 내용은 4.2.4 를 참고 하십시오

***pStrVel:** 연산에 의한 시작 속도

***pMaxVel:** 연산에 의한 최대 속도

***pTacc:** 연산에 의한 가속 시간

***pTdec:** 연산에 의한 감속 시간

***pSVacc:** 연산에 의한 가속 시 S- 커브 범위

***pSVdec:** 연산에 의한 감속 시 S- 커브 범위

***pTconst:** 일정 속도 시간 (최대 속도)

6.22 복귀 코드

복귀 에러 코드는 “8158_err.h” 에 정의 되어 있으며 의미는 다음의 표와 같습니다 .

코드	의미
0	에러 없음
-10000	에러 카드 번호
-10001	에러 동작 시스템 버전
-10002	에러 카드의 ID 충돌
-10300	에러 다른 프로세스 존재
-10301	에러 카드 찾을 수 없음
-10302	에러 드라이버 열기 실패
-10303	에러 ID 맵핑 실패
-10304	에러 트리거 채널
-10305	에러 트리거 타입
-10306	에러 이벤트 이미 활성화됨
-10307	에러 이벤트 아직 활성화 되지 않음
-10308	에러 보드 상 FIFO 꽉참
-10309	에러 알 수 없는 명령 타입
-10310	에러 알 수 없는 칩 타입
-10311	에러 초기 카드 아님
-10312	에러 위치 범위 벗어남
-10313	에러 모션 바쁨
-10314	에러 속도 에러
-10315	에러 slow down 지점
-10316	에러 축 범위 에러
-10317	에러 비교 파라미터 에러
-10318	에러 비교 방법 에러
-10319	에러 축 이미 정지
-10320	에러 축 INT 대기 실패
-10321	에러 사용자 코드 입력 실패
-10322	에러 배열 크기 초과
-10323	에러 요소 번호
-10324	에러 활성화 범위
-10325	에러 자동 가속 시간
-10326	에러 dwell 시간
-10327	에러 dwell 거리
-10328	에러 새로운 위치
-10329	에러 모션 동작 중 아님

코드	의미
-10330	에러 속도 변경 시간
-10331	에러 목표 속도
-10332	에러 속도 비율
-10333	에러 위치 뒤로 변경
-10334	에러 카운터 수
-10335	에러 gpio 입력 함수 파라미터
-10336	에러 채널 수
-10337	에러 ERC 모드
-10338	에러 보안 코드

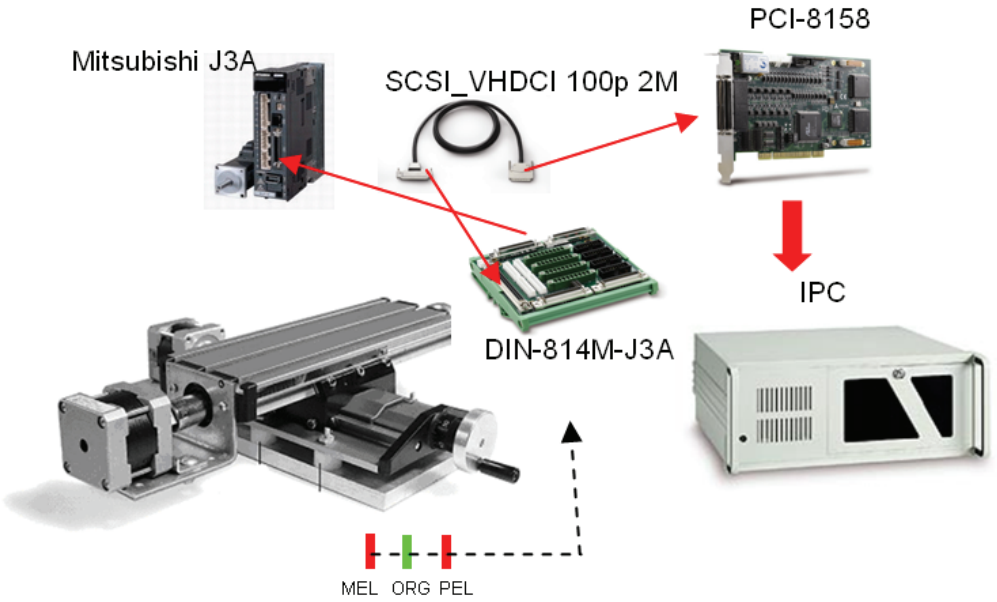
의도적인 공백 페이지

7 결선 예시

이 장은 PCI-8158, 서보 드라이버와 스테핑 드라이버 사이의 결선 예를 보여줍니다.

7.1 일반적인 결선에 대한 설명

PCI-8158 과 펄스 입력, 서보 드라이버 또는 스테핑 드라이버 간의 연결은 메인 연결입니다. 다음의 그림은 PCI-8158 과 DIN-814M-J3A 을 어떻게 결합하는지 보여줍니다.



7.2 터미널 보드 사용자 가이드

개별 터미널 보드의 사용자 가이드를 참고 하십시오. 지원되는 터미널 보드는 다음과 같습니다.

Mitsubishi J2 Super	DIN-814M
Mitsubishi J3A	DIN-814M-J3A
Yaskawa Sigma II	DIN-814Y
Panasonic MINAS A4	DIN-814P-A4

의도적인 공백 페이지

보증 제도

ADLINK 를 선택해 주셔서 감사합니다 . 사용자의 권리를 이해하고 모든 사후 서비스를 누리기 위하여 다음의 내용을 주의 깊게 읽어 주시기 바랍니다 .

1. ADLINK 의 제품을 사용하시기 전에 반드시 사용자 매뉴얼을 읽어 보시고 지침을 정확히 따라주십시오 . 수리를 위하여 손상된 제품을 보내실 때에는 RMA 신청서를 함께 보내주시기 바라며 신청서는 다음의 웹 사이트에서 다운받으실 수 있습니다 : http://rma.adlinktech.com/policy/.
2. 모든 ADLINK 제품은 2 년간의 보증수리 서비스가 제공되며 중국에서 구매하신 제품에 한해 1 년간의 보증기간이 제공됩니다 .
   - 보증기간은 ADLINK 의 공장에서 출고되는 날부터 시작됩니다 .
   - ADLINK 가 제조하지 않은 주변 장치 및 다른 공급 업체 제품은 원 제조업자의 보증이 적용됩니다 .
   - 저장 장치를 포함하는 제품들 (하드 드라이브 , 플래시 카드 등) 의 경우 수리하시기 위해 제품을 보내기 전에 사용자의 데이터를 백업하여 주십시오 . ADLINK 는 데이터의 어떠한 손실에 대해서도 책임을 지지 않습니다 .
   - 우리의 시스템과 함께 올바른 라이센스의 소프트웨어를 사용하는지 확인하여 주십시오 . ADLINK 는 불법 복제 소프트웨어의 사용을 용납하지 않으며 그러한 소프트웨어를 사용하여 서비스하지 않습니다 . ADLINK 는 사용자가 설치한 비면허 소프트웨어와 함께 선적된 제품에 대해 법적 책임을 지지 않습니다 .
   - 일반 수리의 경우 주변 장치를 포함하지 마십시오 . 주변 장치의 포함이 필요하신 경우 RMA 신청 & 확인 양식에 사용자가 보내는 아이템을 확실히 명시하여 주십시오 . ADLINK 는 RMA 신청 & 확인 양식에 명시되지 않은 아이템에 대한 책임을 지지 않습니다 .

3. ADLINK 의 수리 서비스에서 이하의 상황에서는 본사의 보증이 적용되지 않습니다 :
   - ▶ 사용자 설명서를 따르지 않아 발생하는 손상 .
   - ▶ 제품 운송 중 사용자의 부품에 대한 부주의로 인하여 발생하는 손상 .
   - ▶ 화재, 지진, 홍수, 번개, 환경 오염 등의 천재지변 및/또는 잘못된 전압 변압기의 사용으로 인해 발생하는 손상 .
   - ▶ 적합하지 않은 보관 환경으로 인하여 발생하는 손상 (i.e. 고온 , 높은 습도 , 휘발성 화학물질).
   - ▶ 구매자 / 사용자에 의한 배터리 교체 동안 또는 후에 배터리 유체의 누출로 인해 발생하는 손상 .
   - ▶ 공인되지 않은 ADLINK 기술자에 의해 부적절한 수리로 발생하는 손상 .
   - ▶ 제품의 일련번호가 변경 및 / 또는 손상되었을 경우 본사의 서비스를 이용할 수 없습니다 .
   - ▶ 이 증서는 양도하거나 연장할 수 없습니다 .
   - ▶ 다른 카테고리는 보증에 의해 보호되지 않습니다 .
4. 소비자는 손상된 제품을 본사 또는 영업부로 배송하기 위하여 드는 선적비용을 부담할 책임이 있습니다 .
5. 제품 수리의 품질과 속도를 보장하기 위해 RMA 신청서 양식을 본사의 웹사이트에서 다운받아 주시기 바랍니다 : http://rma.adlinktech.com/policy. RMA 양식을 첨부한 손상제품은 우선순위를 받을 수 있습니다 .

언제든지 문의사항이 있으시면 저희 FAE 직원에게 이메일을 보내주시기 바랍니다 :
service@adlinktech.com.