

# 이지컨트롤러 메뉴얼(R7)

---

*eventbus\_net\_0.60*  
*eventbus\_fw\_0.60*

이지컨트롤러는 여러 개의 작은 모듈을 통신버스로 연결하여 기능을 확장할 수 있으며, C 언어로 스캔방식 뿐만 아니라 이벤트방식으로 펌웨어를 프로그래밍할 수 있도록 EventBus\_fw 프레임워크를 제공하며, 컴퓨터와 모듈을 연결하면 PC 기반 제어 및 모니터링 시스템을 단시간에 저렴하게 구성할 수 있도록 EventBus\_net 라이브러리를 제공하므로 경제성과 생산성을 동시에 갖춘 경쟁력 있는 PLC 대체용 자동화 제어기로 사용할 수 있도록 개발되었습니다.



## 저작권 관련규정

이지컨트롤에서 제공되는 모든 소프트웨어에 대한 저작권은 이지컨트롤社에 귀속되며, 소스코드의 형태로 제공되는 모든 소프트웨어는 제품을 구매한 사용자에게 복사, 수정등 자유로운 사용이 허용됩니다. 그러나, 오브젝트코드의 형태로 제공되는 라이브러리에 대하여는 이지컨트롤社에서 판매되는(또는 라이센스 받은) 제품에 내장될 경우 이용 및 배포가 자유롭게 허용되지만, **이지컨트롤社의 제품이 아닌 경우 이를 다운로드 등의 형태로 내장하여 사용하거나 배포하는 것이 허용되지 않습니다.** 사용을 원하는 경우 별도의 협의가 필요합니다.

## 반품 및 환불규정

제품 구매후 포장을 제거하지 않은 경우 1 주일내로 반품하실 수 있습니다. 단, **제품의 포장을 제거한 경우 제품 특성상 환불할 수 없습니다.** 제조사의 명백한 책임으로 볼 수 있는 제품의 이상이 발견되면 반품, 환불, 교환할 수 있습니다. 그러나, 사용자의 실수 또는 잘못된 사용방법으로 인한 파손의 경우 반품, 환불, 교환이 되지 않으므로 주의하여 사용해 주십시오. 본 제품은 저가형으로 제작되어 수리에 필요한 비용이 제품 구매 비용과 크게 차이가 나지 않으므로 특별한 예외의 경우를 제외하면 수리는 하지 않음을 양해해 주십시오.

**제공되는 소프트웨어는 무상으로 제공되고 있으며, 이로인해 발생하는 어떠한 손해에 대하여도 이지컨트롤은 배상등의 책임을 지지 않습니다.** 충분한 테스트를 거쳐 출시해 주십시오. 단, 동작오류, 기능부족, 기타 불편함에 대하여는 최선을 다해 신속히 지원해 드리고 있습니다.

## 고객지원

이지컨트롤은 고객의 원활한 제품개발을 위하여 최선을 다해 기술지원을 제공하려 노력하고 있습니다. 어떠한 불편에 대하여도 업무시간의 제한없이 통화할 수 있으며, 필요에 따라서는 원격접속으로 교육, 기술지원이 가능합니다. 단, 책임이 명백히 이지컨트롤에 있다고 판단되는 경우나 대량 구매처에 대한 기술지원등과 같은 특별한 경우를 제외하고는 **오프라인상의 기술지원은 제공하지 않습니다.** 온라인으로 모든 업무를 처리하는 것이 저희 이지컨트롤의 기본 입장임을 양해 바랍니다. 항상 최선의 파트너가 되기 위해 노력하겠습니다.

### 이지컨트롤 대표

제품판매 : <http://www.easycontrol.co.kr>

자료제공 : <http://cafe.naver.com/easycontrol>

연락처 : [infoic@naver.com](mailto:infoic@naver.com), 010-2502-8502

# 목 차

<b>0. 리비전 히스토리 .....</b>	<b>10</b>
<b>1. 기본개요 .....</b>	<b>12</b>
1.1. 하드웨어 구성 .....	12
1.2. 소프트웨어 구성 .....	13
1.2.1. 스캔 프로그램 .....	14
1.2.2. 이벤트 프로그램 .....	15
1.2.3. 입력이미지 동기화(SyncIn) .....	18
1.2.4. 출력이미지 동기화(SyncOut) .....	19
<b>2. Quick Start .....</b>	<b>22</b>
2.1. 시스템 구성 .....	22
2.2. 다른 제어시스템과의 비교 .....	26
2.3. PC에서 모듈 제어하기 .....	29
2.3.1. 개발환경 설치하기 .....	29
2.3.2. PC에서 모듈 제어하기 .....	29
2.3.3. 콘솔 응용프로그램 .....	30
2.3.4. 윈도우 폼 응용프로그램 .....	36
2.4. 펌웨어로 모듈 제어하기 .....	43
2.4.1. 개발환경 설치하기 .....	43
2.4.2. 펌웨어 프로그래밍 .....	43
2.5. eventbus OS의 다양한 사용법 .....	44
2.5.1. 스캔(Scan) 방식으로 프로그램 하기 .....	48
2.5.2. 이벤트(Event) 방식으로 프로그램 하기 .....	50
2.5.3. 스위치로 릴레이 온오프 시키기 .....	52
<b>3. eventbus-fw .....</b>	<b>54</b>
3.1. 사용자 이벤트 .....	54
3.1.1. 예제 프로그램 : 입력접점의 에지검출 .....	58
3.2. 시간지연 이벤트 .....	59
3.2.1. 온딜레이 타이머(OnDelay timer) .....	59
3.2.2. 온딜레이된 원샷 타이머(OnDelayed OneShot timer) .....	60
3.3. 통신 프로토콜 .....	61
3.3.1. 바이너리 프로토콜 .....	61
3.3.2. DEL 시퀀스 .....	63
3.3.3. 이벤트의 종류 .....	64

3.4.	시스템 시계, <code>clock</code>	66
3.5.	유용한 테크닉	68
3.5.1.	이벤트 소스 판별	69
3.5.2.	상승(하강) 에지 검출	70
3.6.	EEPROM 사용자 영역	71
3.6.1.	RCS 적용예제	71
<b>4.</b>	<b>AT명령</b>	<b>73</b>
4.1.	AT명령 규칙	74
4.2.	동기화 명령	75
4.3.	문자열 표현	76
4.4.	각 모듈별 사용예	76
4.4.1.	AIM 모듈	76
4.4.1.	AOM 모듈	77
4.4.2.	DIE 모듈	77
4.4.3.	DIM 모듈	78
4.4.4.	DOE 모듈	79
4.4.5.	DIO 모듈	80
4.4.6.	DOM 모듈	81
4.4.7.	ENV 모듈	82
4.4.8.	MOC 모듈	83
4.4.9.	MSB 모듈	84
4.4.10.	SEG 모듈	85
4.4.11.	SMD 모듈	85
4.4.12.	TCM 모듈	87
4.4.13.	LAN, LMO, USB 모듈	88
4.4.14.	USD 모듈	88
<b>5.</b>	<b>eventbus_net</b>	<b>90</b>
5.1.	컴퓨터와 연결하는 방법	90
5.1.1.	ColinkEx를 사용하는 방법	90
5.1.2.	UUB를 사용하는 방법	91
5.1.3.	UUM 모듈을 사용하는 방법	91
5.1.4.	USB 모듈을 사용하는 방법	92
5.1.5.	U2M 모듈을 사용하는 방법	92
5.1.6.	LAN 모듈을 사용하는 방법	93
5.2.	<code>ezconfig.exe</code> 사용가이드	93
<b>6.</b>	<b>모듈의 종류</b>	<b>100</b>

6.1.	AIH, Analog Input High-resolution Module .....	102
6.1.1.	입력전압 교정법 .....	105
6.2.	AIM, Analog Input Module .....	106
6.2.1.	입력전압 교정법 .....	109
6.2.2.	입력전압 범위를 넓히는 방법 .....	110
6.3.	AOM, Analog Output Module .....	112
6.3.1.	출력전압 교정법 .....	116
6.3.2.	최대 출력전압의 변경 .....	117
6.4.	DIO, Digital Input/Output Module .....	119
6.5.	DOM, Digital Output Module .....	123
6.6.	DIM, Digital Input Module .....	127
6.6.1.	싱크(Sink)타입 스위치 입력 .....	130
6.6.2.	트랜지스터 출력타입의 센서를 연결하는 방법 .....	131
6.6.3.	근접센서(스위치) 연결방법 .....	133
6.7.	DIE, Digital Input Expanded Module .....	135
6.7.1.	싱크(Sink)타입 스위치 입력 .....	137
6.8.	DOE, Digital Output Expanded Module .....	141
6.9.	SMD, Step Motor Driver Module .....	145
6.10.	USB, USB Communication Module .....	152
6.11.	ENV, Environmental Sensing Module .....	156
6.12.	SEG, 16- Segment Display Module .....	160
6.13.	MSB, Multiport Serial Bridge .....	164
6.14.	MOC, Motion Control Module .....	169
6.15.	USD, Unipolar Step-motor Driver .....	177
6.16.	RCS, RC-Servo Motor Controller .....	187
6.17.	TCM, Thermocouple Measurement Module .....	191
6.18.	LAN, TCP/IP Communication Module .....	195
6.19.	LMO, Modbus Tcp/ip 인터페이스 모듈 .....	201
6.20.	UPS, Uninterrupted Power Supply Module .....	220
6.20.1.	만충전압, 방전종지전압 설정방법 .....	221
6.21.	PSM, Power Supply Module .....	223
6.22.	ColinkEx, SWD JTAG Module .....	223
6.23.	EXP, Expansion Module .....	224
6.24.	BPU, Bus Pull-Up Module .....	224
6.25.	UUB, USB-UART Bridge (단종예정) .....	225
6.26.	UUM, UART-USB Bridge Module .....	225
6.27.	U2M, UART-RS232 Bridge Module .....	226

6.28.	DIN레일용 부품	226
6.29.	PWR 전원연결 모듈	227
<b>7.</b>	<b>모듈의 커넥터 사양</b>	<b>228</b>
7.1.	커넥터 사양	228
7.2.	커넥터의 핀번호	229
7.3.	버스커넥터	229
<b>8.</b>	<b>PLC 프로그래밍</b>	<b>232</b>
8.1.	PLC 래더다이어그램	232
8.1.1.	PLC의 기본구조	232
8.1.2.	Ladder 기본개요	232
8.1.3.	연산자 종류	233
8.2.	ezplc에 의한 래더다이어그램 구현	235
8.2.1.	Edge, 에지검출	236
8.2.2.	Hold, 자기유지회로	237
8.2.3.	Interlock, 인터록회로	238
8.2.4.	OnDelay, OffDelay, OneShot 타이머	238
8.2.5.	UpCount, DownCount, UpDownCount 카운터	240
8.3.	평선블록의 사용예	244
8.4.	시퀀스 프로그램 주의사항	249
8.5.	래더프로그래밍 예제	250
8.5.1.	삼로 스위치	250
8.5.2.	다이나믹 플립플롭 회로	252
8.5.3.	기동모터 개수제어	254
8.5.4.	평션	257
<b>9.</b>	<b>유한상태도(FSM)</b>	<b>258</b>
9.1.	예제: 상태도 구현	258
9.2.	예제: 상태도 확장(1)	260
9.3.	예제: 상태도 확장(2)	262
9.4.	예제: 상태도 확장(3)	264
<b>10.</b>	<b>eventbus_fw_0.60 레퍼런스 가이드</b>	<b>266</b>
10.1.	기본 데이터형	266

10.2.	CLOCK 타입 .....	266
10.3.	EEdge 열거형.....	266
10.4.	ELogic 열거형.....	266
10.5.	EActiveLogic 열거형 .....	267
10.6.	SMD_CMD, SMD_DIR, SMD_STEP, MOC_CMD, MOC_DIR, USD_CMD, USD_DIR, USD_STEP 열거형 .....	267
10.7.	Ascii 열거형.....	268
10.8.	TCType 열거형.....	268
10.9.	ESender 열거형.....	268
10.10.	EEventTag 열거형 .....	270
10.11.	MType 열거형.....	276
10.12.	EventHead 구조체 .....	276
10.13.	Event 구조체.....	276
10.14.	CyclicTimeoutEvent .....	277
10.15.	VaaEvent, VbbEvent, VdcEvent .....	277
10.16.	DelayedEvent .....	277
10.17.	MsbTransmitEvent, MsbReceivedEvent .....	278
10.18.	XxxInEvent : 입력이미지 .....	278
10.19.	XxxOutEvent : 출력이미지 .....	278
10.20.	AIM, AOM, DIE, DIM, DIO... : 모듈 이미지 .....	278
10.21.	EventQ_Put() 함수 .....	279
10.22.	EventQ_Get() 함수 .....	279
10.23.	Event_Sync() 함수 .....	279
10.24.	Event_Is() 함수 .....	279
10.25.	Event_SetDefault() 함수 .....	280
10.26.	CreateEvent(tag) 매크로함수 .....	280
10.27.	UserEvent(event_var, tag) 매크로함수 .....	280
10.28.	SendEvent(), SendDelayedEvent() 함수 .....	280
10.29.	ez_malloc, ez_free 함수 .....	281

10.30.	<code>clock</code> 전역변수 .....	281
10.31.	<code>EventBus()</code> 함수 .....	281
10.32.	<code>CRC()</code> 함수 .....	281
10.33.	<code>O CreateXXX</code> 이미지 생성함수 .....	281
10.34.	<code>WriteROM, ReadROM</code> 매크로 .....	282
10.35.	<code>UART_SetBaudrate()</code> 함수 .....	282
10.36.	<code>printc(), printf()</code> 함수 .....	282
10.37.	<code>seg_sprintf ()</code> 함수 .....	283
10.38.	<code>Hw_SyncToOut()</code> 함수 .....	283
10.39.	<code>Wait_10us</code> 시간지연 함수 .....	283
10.40.	<code>DISABLE_IRQ, ENABLE_IRQ</code> 인터럽트 함수 .....	283
10.41.	<code>EventProgram</code> 매크로 .....	283
10.42.	<code>ScanProgram</code> 매크로 .....	284
10.43.	<code>End</code> 매크로.....	284
10.44.	<code>IF()</code> 매크로함수 .....	284
10.45.	<code>Pressed, Released, Toggled, Toggle</code> 매크로함수 .....	284
<b>11.</b>	<b><code>eventbus_net_0.60</code> 레퍼런스 가이드 .....</b>	<b>286</b>
11.1.	<code>모듈이미지</code> 클래스 .....	286
11.1.1.	<code>CModule</code> 클래스 .....	286
11.1.2.	<code>CAIH</code> 클래스 .....	287
11.1.3.	<code>CAIM</code> 클래스 .....	288
11.1.4.	<code>CAOM</code> 클래스 .....	288
11.1.5.	<code>CDIE</code> 클래스 .....	289
11.1.6.	<code>CDIM</code> 클래스 .....	289
11.1.7.	<code>CDIO</code> 클래스 .....	290
11.1.8.	<code>CDOE</code> 클래스 .....	290
11.1.9.	<code>CDOM</code> 클래스 .....	291
11.1.10.	<code>CENV</code> 클래스 .....	291
11.1.11.	<code>CLAN</code> 클래스 .....	292
11.1.12.	<code>CMOC</code> 클래스 .....	292
11.1.13.	<code>CMSB</code> 클래스 .....	293
11.1.14.	<code>CRCS</code> 클래스 .....	293
11.1.15.	<code>CSEG</code> 클래스 .....	294

11.1.16.	CSMD 클래스 .....	294
11.1.17.	CTCM 클래스 .....	294
11.1.18.	CUSB 클래스 .....	295
11.1.19.	CUSD 클래스 .....	295
<b>11.2.</b>	<b>Serial 통신클래스 .....</b>	<b>297</b>
<b>11.3.</b>	<b>이벤트 클래스 .....</b>	<b>297</b>
11.3.1.	MsbTransmitEvent 이벤트 .....	297
<b>11.4.</b>	<b>EventBus\CDevice.cs 데이터 클래스 .....</b>	<b>298</b>
11.4.1.	CSwitch 클래스 .....	298
11.4.2.	CRelay 클래스 .....	298
11.4.3.	CVoltInput 클래스 .....	299
11.4.4.	CVoltOutput 클래스 .....	299
11.4.5.	CCurrentOutput 클래스 .....	299
11.4.6.	CNumericInput 클래스 .....	300
11.4.7.	CNumericOutput 클래스 .....	300
11.4.8.	CFlag 클래스 .....	301
11.4.9.	CCommand 클래스 .....	301
11.4.10.	CDirection 클래스 .....	302
11.4.11.	CStringOutput 클래스 .....	302
11.4.12.	CStepper 클래스 .....	303
11.4.13.	CMsbPacket 클래스 .....	304
<b>11.5.</b>	<b>EventBus\Event.cs 이벤트 아규먼트 클래스 .....</b>	<b>305</b>
11.5.1.	CUserEventArgs 클래스 .....	305
11.5.2.	CVoltageEventArgs 클래스 .....	305
11.5.3.	CCurrentEventArgs 클래스 .....	305
11.5.4.	CLogicEventArgs 클래스 .....	306
11.5.5.	CNumericEventArgs 클래스 .....	306
11.5.6.	CCommandEventArgs 클래스 .....	306
11.5.7.	CDirectionEventArgs 클래스 .....	306
11.5.8.	CStepperEventArgs 클래스 .....	307
11.5.9.	CStringEventArgs 클래스 .....	307
11.5.10.	CAdcEventArgs 클래스 .....	308
11.5.11.	CUsdStepEventArgs 클래스 .....	308
11.5.12.	CMsbReceivedEventArgs 클래스 .....	308

## 0. 리비전 히스토리

---

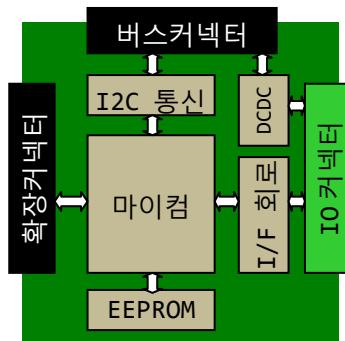
- Rev 0.10 (2011.11.15)
  - 최초발행
- Rev 0.10a (2012.1.4)
  - 이벤트버스 커넥터 사양 추가
- Rev 0.20 (2012.3.6)
  - EventBus\_C\_v0.20 업그레이드
  - EventBus\_dotnet\_v0.21 업그레이드
  - EventBus\_java\_v0.2 추가
- Rev 0.30 (2012.4.5)
  - EventBus\_C\_v0.30 업그레이드
  - EventBus\_dotnet\_v0.30 업그레이드
  - EventBus\_java\_v0.30 업그레이드
- V0.40(R3) (2012.7.25)
  - eventbus\_c\_v0.40 내용반영
- V0.40(R4) (2012.8.3)
  - 통신프로토콜, 시스템시계 clock 추가
  - 각 모듈의 핀맵, 이미지 구조체 설명추가
  - PLC 프로그래밍 추가
- V0.40(R5) (2012.8.30)
  - UPS 전압설정방법 추가
  - 각 모듈의 인터페이스 방법 추가
- V0.50(R6) (2012.12.1)
  - 스캔방식 프로그래밍 스타일 추가
  - 신제품 추가 (UUM,U2M,MSB,USD,DIE,DOE)
  - ezconfig.exe 설정항목에서 지연시간(Td) 제거
  - eventbus\_dotnet 성능향상
- V0.60(R7)
  - eventbus\_fw\_0.60, eventbus\_net\_0.60 Upgrade
  - CoIDE 1.7.4 적용
  - AT 명령 추가
    - ezconfig.exe에 명령터미널 추가제공
  - EventHead 의 from 0| crc 로 변경
    - eventbus\_fw 와 eventbus\_net 은 crc 바이트로 이벤트의 손상여부 검사
  - EEPROM에 사용자 데이터 저장
  - LMO, AIH, RCS, PWR 모듈 추가

- **ezconfig.exe Upgrade**
    - 한모듈만 PC에 연결되면 모든 모듈을 설정할 수 있도록 개선
    - USB, RS232 외에 LAN 연결 추가
  - **eventbus.h**에 정의된 각 모듈의 이미지 구조체의 명칭이 변경됨(\*\*)
  - SyncOut() 사용없이도 출력이 자동동기화
  - SoftReset 이벤트 추가로 프로그램에서 특정모듈을 초기화(리셋)
  - UART, I2C, Ethernet 등 송수신 호출함수를 드라이버 경유하도록 변경
    - 사용자가 송수신 데이터를 가로채기 할 수 있도록 노출
  - UART(PC 연결, UUM, U2M, UUB) 디폴트 통신속도를 115,200bps로 변경
  - 참조모듈의 생성위치 변경
    - 이전버전에서 EventBus() 호출되었으나 v0.60 부터 호출전으로 변경(\*\*)
  - **ezplc.h Upgrade**
    - 함수명 변경, Osc() 함수추가
    - FSM(유한상태도) 방식의 시퀀스제어 방법론 추가
  - LAN 모듈의 8개 소켓 각각에 독립된 수신버퍼를 두어서 안정성 향상
  - SEG 모듈 사용법 변경(이미지 및 예제 확인요)
  - ENV 모듈 사용법 변경(박수소리 0인 경우 이벤트 발생하지 않도록 변경)
  - 하드웨어 락키추가
    - 기존버전(v0.50 이하) 사용자는 각 모듈마다 락키를 받아야 합니다.
    - 펌웨어 다운로드만으로는 동작하지 않습니다.(네이버카페 공지참고)
    - 새로 구매한 모듈에는 락키가 들어있으므로 그대로 사용하면 됩니다.
- (\*\*) 기존 사용자 프로그램 업그레이드시 필수 확인 필요

## 1. 기본개요

이지컨트롤에서는 제어시스템을 구성하기 위한 전문지식이 부족하거나, PLC 를 대체하여 비용을 절감하기를 원하는 기업을 위하여 쉽게 시스템 중심의 시퀀스제어 및 임베디드 시스템을 개발할 수 있도록 이지컨트롤러를 개발 공급하고 있습니다.

### 1.1. 하드웨어 구성



이지컨트롤러는 위 그림과 같은 기본구조를 가지는 기본 사이즈 W50 X L55 X H10~20MM 크기의 소형 제어모듈입니다. 각각의 세부 구성요소는 아래와 같습니다.

#### ■ I/O 커넥터

스위치, 램프, 모터 등의 외부 장치를 연결하는 커넥터입니다. DINKLE 사의 ECH350R 플러그&헤더 타입 터미널 블럭을 사용하여 컴팩트하며, 스크류방식이므로 납땜없이 쉬운 배선을 할 수 있습니다.

#### ■ 버스커넥터

각각의 모듈들은 독립적으로 동작할 수도 있지만, 버스커넥터를 통하여 IDC 커넥터와 플랫케이블로 연결되어 더욱 복잡한 기능의 제어를 수행할 수 있습니다. 커넥터에는 I2C 통신선, VAA(로직용 전원, 5V), VBB(I/O 용 전원, 24V), 그라운드, 시스템리셋 신호선이 할당되어 있습니다. 2MM 피치의 20 핀 (또는 특정 모듈의 경우 10 핀) 박스헤더커넥터를 사용합니다.

#### ■ 확장커넥터

マイコン의 내장 I/O 장치 중에서 모듈의 동작에 사용된 것을 제외한 나머지를 사용자가 확장하여 사용할 수 있도록 외부로 제공하는 커넥터로, 전원출력, UART, JTAG SWD, GPIO, SPI 등이 연결되어 있습니다. 2MM 피치의 24 핀 박스헤더커넥터를 사용합니다.

#### ■ 마이컴

모든 기능모듈은 32 비트 ARM CORTEX-M0 계열의 저전력 LPC1114 마이컴을 내장하고 있습니다. 사용자는 마이컴의 Flash 에 eventbus\_fw 프레임워크를 이용하여 이벤트 또는 스캔방식으로 사용자 프로그램을 쉽고 빠르게 작성할 수 있습니다.

#### ■ I2C 통신

이지컨트롤러는 버스커넥터에 있는 SDA, SCL 두선의 I2C 통신라인으로 각 모듈을 연결합니다. FM+(Fast Mode Plus) 모드를 사용하여 1Mbps 의 고속통신으로 각 모듈을 연결하여 모듈간에 이벤트를 교환하며 상호동작을 합니다.

#### ■ I/F 회로

I/O 커넥터에 연결된 외부장치를 구동하기 위한 회로입니다. 예를 들면, 스위치 입력회로, 릴레이 구동회로, 모터 구동회로, 센서 인터페이스 회로 등이 있습니다.

#### ■ DC/DC 컨버터

버스커넥터로 부터 공급받은 VBB(24V)를 적절한 전압으로 다운시켜(사용자가 모듈에 있는 가변저항을 회전시켜서) 사용자가 외부 장치의 동작을 위해 사용할 수 있도록 I/O 커넥터를 통하여 외부로 제공하는 가변전원입니다. 가변범위는 약 1.5V ~ 20V 입니다. 간이용 전원이므로 출력전류는 그다지 크지 않습니다.

#### ■ EEPROM

ID(통신주소)와 같은 설정값을 저장할 수 있는 저장공간을 제공합니다. 프로그램의 개발 및 다운로드가 끝난 후에도 언제든지 ColinkEx, UUM, U2M 으로 EEPROM 에 저장된 파라미터를 변경하여 프로그램의 동작을 제어할 수 있습니다. 함께 제공되는 ezconfig.exe 프로그램으로 저장된 설정내용을 변경할 수 있습니다.

## 1.2. 소프트웨어 구성

이지컨트롤러는 사용자가 C 언어를 이용하여 완전히 처음부터 펌웨어를 작성할 수도 있습니다만, 개발에 많은 시간과 노력이 필요합니다. 그래서 초급 사용자도 빠르고 안정적으로 펌웨어를 프로그램할 수 있도록 OS 인 eventbus 를 제공합니다.

eventbus 의 프로그래밍 모델은 크게 **스캔 프로그램과 이벤트 프로그램**이 있습니다. 스캔프로그램은 일반적인 임베디드 프로그램이나 PLC 의 래더프로그램과 같이 특정 프로그램을 무한반복 실행하는 방식이며, 이벤트 프로그램은 윈도우의 폼 프로그램에서처럼 마우스가 클릭되는 것과 같은 특정 이벤트(사건)에 대하여 이벤트핸들러라는 처리 프로그램을 작성하는 방식으로 프로그램의 모듈화, 가독성, 문서화 등의 많은 장점이 있는 프로그래밍 방식입니다.

eventbus 는 스캔방식과 이벤트방식을 혼용하여 펌웨어를 개발 할 수 있으므로 쉽고 빠르며 이해하기 좋은 프로그래밍을 지원합니다.

**이벤트방식 :** 이벤트(사건)발생시 처리할 내용을 기술할 때 적합

**스캔방식 :** 시간흐름에 따른 순서적(시퀀스) 처리를 기술할 때 적합

**\*핵심\*** 아래 프로그램은 eventbus 를 이용하여 이지컨트롤러에 작성한 예제입니다. EventProgram; 은 이벤트 프로그램의 시작을 나타내고, ScanProgram; 은 스캔프로그램의 시작을 나타내며, End; 는 프로그램의 종료를 나타냅니다.

이 세가지 기본 골격은 사용하지 않더라도 반드시 프로그램에 순서대로 있어야 합니다. 예를 들어 스캔프로그램을 작성하지 않더라도 ScanProgram;은 반드시 이벤트 프로그램의 끝부분에 있어야 합니다. (이 3 개의 매크로가 어떻게 정의되어 있는지 살펴보면 이해에 큰 도움이 됩니다)

```
int main()
{
    // 참조모듈 생성
    EventBus();

    // 변수선언, 초기화 코드
    EventProgram;
    // 이벤트핸들러

    ScanProgram;
    // 스캔프로그램 (무한반복)

    End;
}
```

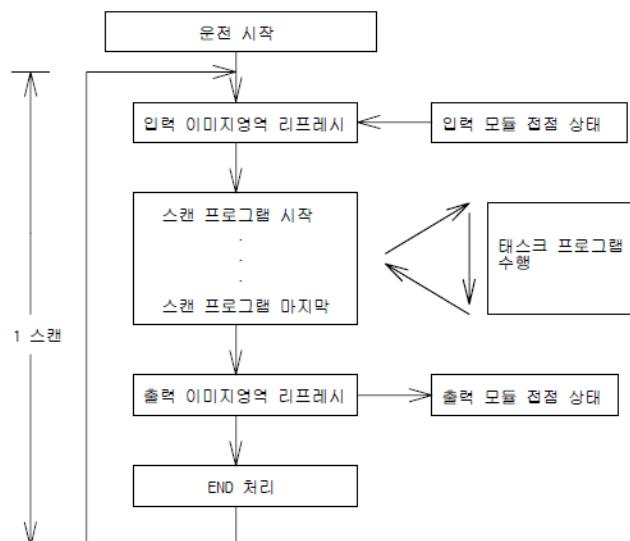
**\*\*주의\*\***

참조모듈의 선언은 v0.60 부터는 EventBus( ); 앞에 있어야 합니다. 이전 버전과 위치가 다르므로 이전 버전으로 작성된 프로그램의 업그레이드시에 주의가 필요합니다.

### 1.2.1. 스캔 프로그램

스캔프로그램은 PLC 에서 사용되는 프로그래밍 방식으로, 말 그대로 처음부터 순서대로 프로그램을 스캔하면서 무한 반복으로 실행하는 단순한 방식입니다.

일반적으로 시퀀스제어용 프로그램을 작성할 때 사용되는 방식으로 C 언어의 문법만 알면 바로 프로그램을 작성할 수 있으며, 하드웨어나 OS 등과 같은 보조수단에 대한 별도의 학습이 필요없으므로 빠르고 간편하게 프로그램을 개발할 때 많이 사용됩니다.



위 그림은 PLC에서 스캔방식 프로그램을 실행하는 방법을 도식화한 것입니다. 연결된 입력을 읽어 메모리의 이미지영역에 저장후, 스캔프로그램을 순서대로 실행하는데, 이때 계산된 출력은 즉시 출력하지 않고 메모리의 출력 이미지영역에 저장하였다가, 스캔프로그램의 실행이 끝나면 일괄적으로 출력합니다. PLC의 래더프로그램은 이렇게 스캔프로그램을 무한반복하는 구조를 가지고 있습니다.

아래 예제처럼 eventbus에서는 스캔프로그램을 ScanProgram; 매크로 아래에서 구현합니다.

```
int main()
{
    EventBus();

    EventProgram;

    ScanProgram;
    Eval_DioIn(FALSE);           // 입력검사를 무한반복하는 간단한 스캔 프로그램

    End;
}
```

위의 경우 이벤트 프로그램은 없으며, 스캔프로그램에서는 Eval\_DioIn(FALSE) 함수를 반복적으로 호출하게 됩니다.(이전버전의 Eval\_DioInChanged()함수가 함수명만 변경되었습니다). 사실 이 함수는 DIO 모듈에서 입력접점(sw0~sw5)의 상태를 읽어서 접점에 변화가 있을 때만 이벤트를 발생하는 함수입니다. 발생한 이벤트는 통신으로 연결된 다른 모듈(컴퓨터 포함)에 전달되어 접점의 상태를 공유할 수 있도록 하는 중요한 기능을 합니다.

\*\* V0.50 이전에는 CyclicTimeout 이벤트핸들러에서만 이 함수를 호출할 수 있었지만 스캔프로그램에서 사용할 수 있으므로 더욱 고속실행이 가능합니다.

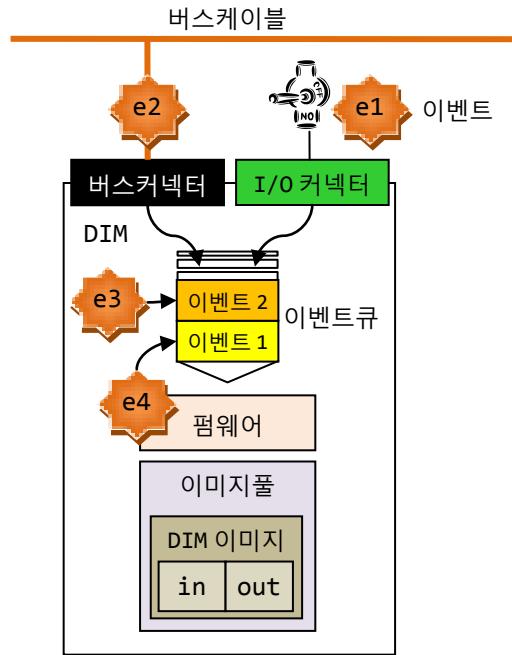
### 1.2.2. 이벤트 프로그램

“**이벤트(Event)**”란 사건을 뜻하는데, 스위치가 눌러지는 것과 같은 사건이 발생하는 것을 의미합니다. 그리고, “**버스(Bus)**”란 서로 데이터를 공유할 수 있도록 연결된 통로를 의미하는데, 이지컨트롤러 모듈들을 연결하는 버스케이블에는 I2C 통신선과 전원선이 있어서 데이터와 전원을 서로 공유하는 구조(버스)를 가지고 있습니다.

즉, “**이벤트버스**”란 이벤트가 발생하면 버스에 연결된 컨트롤러와 컴퓨터가 발생한 이벤트(입출력 정보를 저장한 데이터)를 쉽게 공유할 수 있도록 고안된 하드웨어 또는 소프트웨어 시스템을 말합니다.

이렇게 말하면 뭐 특별하고 새로운 방식인것 같지만, 과거 MS-DOS 시절에는 일을 처리하는 순서대로 프로그램을 작성했었는데(일종의 스캔프로그램) 매우 불편했습니다. 하지만 윈도우즈가 GUI(그래픽 유저 인터페이스)를 채택하면서 이전 방식으로 프로그램하는 것이 불편하여 채택한 방식이 바로 이벤트방식입니다.

“eventbus”는 이 두가지 프로그램 개발방식을 모두 사용할 수 있으며, 특히 이벤트는 윈도우즈와 매우 유사한 방식으로 동작합니다. 하지만, 강력한 성능의 CPU를 가진 컴퓨터에서 동작하는 윈도우를 그대로 임베디드 시스템에 가져와 사용할 수는 없습니다. 그래서 eventbus는 컴퓨터에 비해 성능이 약한 MCU를 가진 제어시스템에 적합하게 동작방식을 재구성하였습니다.

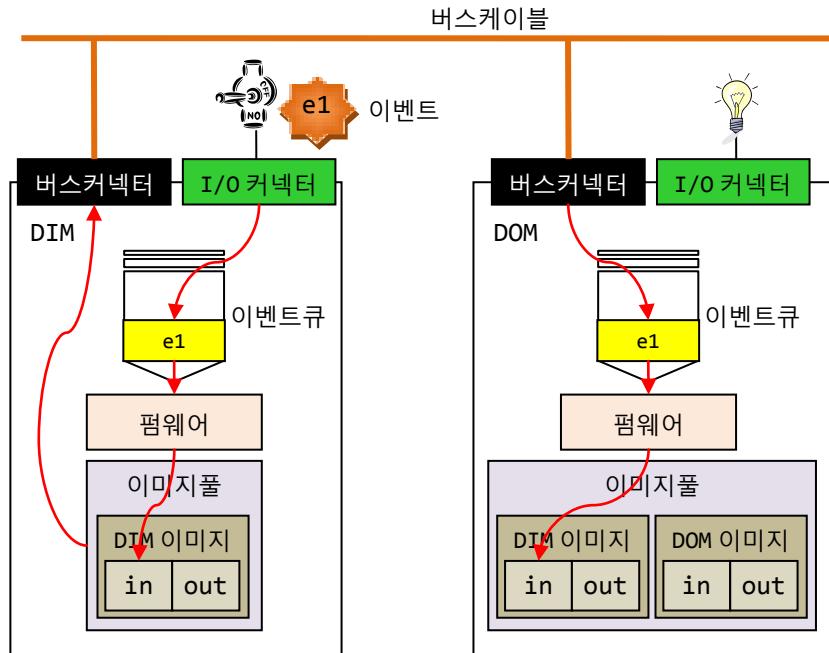


위 그림은 이자컨트롤러에서 이벤트를 처리하는 과정을 보여줍니다. 그림처럼, 이벤트(사건)는 다양한 곳에서 발생하지만 발생한 이벤트는 모두 이벤트큐에 저장됩니다.

- 외부장치 (e1)  
스위치처럼 I/O 커넥터에 연결된 외부장치에 의해 발생되는 이벤트.
- 연결된 모듈 (e2)  
버스케이블의 통신선을 통하여 다른 컨트롤러로 부터 수신한 이벤트.
- 내부 리소스 (e3)  
마이컴의 타이머(시계)와 같은 컨트롤러 내부의 리소스에서 발생한 이벤트.
- 펌웨어 프로그램 (e4)  
펌웨어 프로그램(eventbus 또는 사용자 프로그램)에서 발생시킨 이벤트.

윈도우즈의 이벤트처리과정과 비교하면 매우 유사함을 알 수 있습니다.

- 윈도우즈의 경우  
마우스클릭(이벤트발생) → 메시지큐에 저장 → 메시지루프에서 이벤트핸들러 실행
- 이벤트버스의 경우  
스위치 ON(이벤트발생) → 이벤트큐에 저장 → 이벤트루프에서 이벤트핸들러 실행



위 그림은 스위치를 동작시킬 때 발생하는 이벤트(e1)가 어떻게 처리되는지를 보여줍니다.

**\*핵심\*** 스위치를 동작하면, DimIn 이벤트(e1)가 발생하고 메모리의 “이벤트큐”에 저장됩니다. 이벤트큐는 이벤트를 저장된 순서대로 이벤트핸들러(이벤트가 발생할 때 처리할 내용을 기술한 사용자 프로그램)에게 건네 줍니다. 스위치 또는 릴레이처럼 입출력에 관련된 이벤트는 특별히 이미지풀에 별도로 저장됩니다. 필요한 경우, 발생한 이벤트는(위의 경우 DIM에서 발생) 버스케이블을 통하여 연결된 다른 컨트롤러(위의 경우 DOM)에게도 이벤트발생 사실을 알리기 위하여 통신으로 이벤트를 전송합니다.

이벤트의 동작은 윈도우와 유사하지만, 이벤트의 후처리 과정에는 두 가지 큰 차이가 있습니다.

- 입출력 이벤트는 이미지에 저장해 둔다.

윈도우즈의 경우 사용을 마친 이벤트는 버리지만, eventbus에서는 입출력관련 이벤트 (XxxInEvent, XxxOutEvent)는 이미지풀에 저장하여 향후 다시 사용할 수 있게 한다.

- 연결된 다른 컨트롤러에게 전송한다.

대부분의 이벤트는 버스케이블의 I2C 통신으로 연결된 다른 컨트롤러에게 보내지고, 그 컨트롤러의 이벤트큐에 들어갑니다.

위에서 설명한 이벤트의 처리과정은 이벤트방식으로 프로그램을 작성하는데 있어서 매우 중요하므로 반드시 숙지해 주십시오. 그리고 용어 몇가지를 정리하면,

- 이미지 (image)

모듈의 입출력 데이터를 저장한 변수. 이미지에는 입력이미지(in)와 출력이미지(out)가 있어서, 입력이미지에는 입력이벤트(XxxInEvent)가 저장되고, 출력이미지에는 출력이벤트(XxxOutEvent)가 저장됩니다.

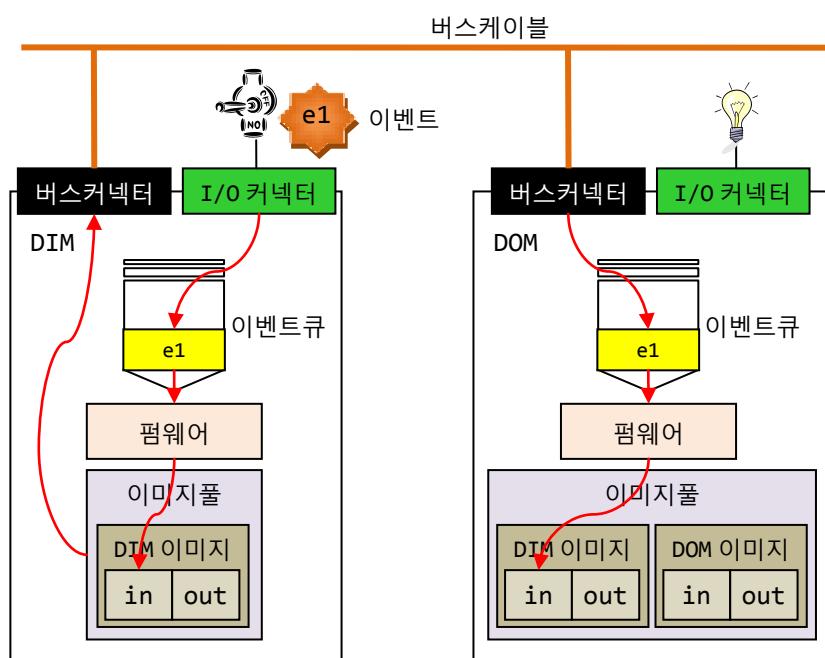
### ■ 이미지풀 (Image Pool)

버스에 연결된 각 모듈의 이미지를 저장하는 곳. 이미지풀에는 자신의 이미지 뿐만 아니라 다른 모듈의 이미지도 저장하도록 프로그램 할 수 있습니다. (그림에서 DOM 이미지풀은 자신의 이미지 뿐만 아니라 DIM 의 이미지도 저장하고 있어서 언제든지 DIM 의 입출력을 참조할 수 있다. `CreateDIM()`함수 참고)

### ■ 이미지 동기화 (Synchronize)

각 모듈에 저장된 이미지의 내용을 서로 동일하게 만드는 과정. 위 그림에서 DIM 모듈은 이미지풀에 자신의 이미지(DIM 이미지)만 저장하면 되지만, DOM 모듈은 램프를 동작시키기 위하여 DIM 모듈의 스위치 상태를 알아야 한다고 가정하면, 이미지풀에 자신의 이미지(DOM 이미지)뿐만 아니라 DIM 모듈의 이미지도 함께 저장해 두면 편리할 것입니다. 이때, DIM 에 저장된 이미지와 DOM 에 저장된 이미지는 서로 동일한 값을 가져야 합니다. 이렇게 이미지들이 서로 동일한 값을 가지도록 만드는 것을 동기화(Sync)라 합니다. 동기화는 `eventbus`의 가장 중요한 기능중 하나인데, 완전히 자동으로 수행됩니다.

#### 1.2.3. 입력이미지 동기화(SyncIn)



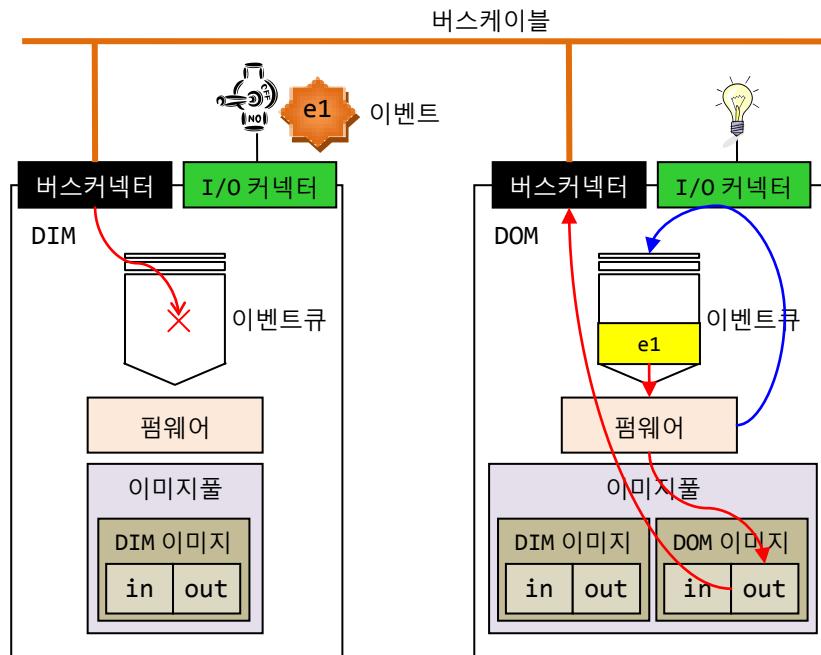
두 컨트롤러 DIM, DOM 이 그림처럼 버스케이블(플랫케이블)로 연결되고, DIM 에는 스위치가 연결되고 DOM 의 릴레이에는 전구가 연결되어 있습니다. 여기서 DIM 의 스위치를 ON/OFF 하면 DOM 의 릴레이가 ON/OFF 되어 램프를 제어하는 펌웨어를 DOM 의 마이컴에 작성하려 합니다.

\*참고\* 이지컨트롤러는 `eventbus OS` 가 내장되어 출고되므로, 사용자가 별도의 펌웨어를 작성하지 않아도 이미지 동기화는 자동으로 실행됩니다. (`DIM` 은 `dim_def.c`, `DOM` 은 `dom_def.c` 참고)

동작과정을 순서대로 살펴보면, (빨간 화살표 참고)

**\*핵심\*** 스위치 ON  $\Rightarrow$  DimIn 이벤트 발생  $\Rightarrow$  이벤트큐에 저장  $\Rightarrow$  펌웨어(작성하지 않았으므로 지나감)  $\Rightarrow$  입력이미지(DIM->in)에 저장  $\Rightarrow$  통신버스로 DOM에 전달  $\Rightarrow$  DOM의 이벤트큐에 저장  $\Rightarrow$  펌웨어(여기서 릴레이를 동작시는 이벤트핸들러를 작성)  $\Rightarrow$  입력이미지(DIM->in)에 저장(입력이미지 동기화)

#### 1.2.4. 출력이미지 동기화(SyncOut)



앞에서 DOM의 릴레이를 동작시키는 이벤트핸들러를 DOM의 마이컴에 작성한다고 하였습니다. 구체적인 프로그램과 그 동작순서를 살펴봅니다. 우선 DOM 모듈에 작성하는 펌웨어는 DIM의 입력(스위치 상태)을 알아야 하므로, 이미지풀에 그림처럼 “**DIM 이미지**”를 생성합니다.

```
DIM* dim = CreateDIM(3);
```

DIM 모듈의 이미지인 `dim`을 생성하였습니다. 여기서, `ID_DIM3`(반드시 `ezconfig.h`의 EID에 선언해야 함)은 DIM의 ID(통신주소)입니다. 버스케이블로 연결된 모든 컨트롤러는 다른 컨트롤러와 구별되는 유일한 ID를 가져야 합니다. 그리고, 가능하면 ID는 1부터 순서대로 사용하는 것이 성능에 좋습니다. `CreateDIM` 함수는 DIM 이미지를 만들어 이미지풀에 저장하고 이미지의 주소를 반환하는 함수입니다. 이제 이미지에 접근하려면 변수명 `dim`을 사용하면 됩니다. (이미지의 명칭은 컨트롤러의 용도에 맞게 적절히 작성하면 됩니다) 여기서, `dim`은 포인터형이므로 `dim->in` 또는 `dim->out` 처럼 “`->`” 연산자로 액세스 합니다.

```
int main()
{
    DIM* dim = CreateDIM(ID_DIM3);

    EventBus();

    EventProgram;
}
```

```

IF (dim, DimIn) {
    this->out->rly0 = pe->sw0;
}

ScanProgram;
End;
}

```

위 프로그램은 DOM 에 내장된 릴레이를 동작시키는 이벤트핸들러의 예제입니다 (DOM 의 마이콤에 다운로드 함). **IF** 는 이벤트핸들러를 작성하기 위한 매크로함수이며 형식은 아래와 같습니다.

```

IF (이미지 명칭, 이벤트 태그) {
    // 이벤트핸들러 프로그램
}

```

즉, **IF (dim, DimIn) { ... }**은 “dim에서 발생한 DimIn 이벤트를 처리하는 이벤트핸들러”라는 의미입니다 (닫는 괄호가 2개임에 주의).

this->out->rly0 는 this(프로그램을 작성하고 있는 모듈, 여기서는 DOM) 모듈의 출력이미지(out)에 내장된 0 번 릴레이(rly0)를 말합니다. 자세한 구조를 알고 싶다면, eventbus.h 의 DomOutEvent 구조체를 참고하십시오.

pe->sw0 은 발생한 이벤트(pe, pointer to event 의 약어, dim 모듈에서 발생한 DimInEvent)에 저장된 sw0 스위치값을 말합니다. 자세한 구조를 알고 싶다면, eventbus.h 의 DimInEvent 구조체를 참고하십시오.

결국, this->out->rly0 = pe->sw0;은 dim 모듈에서 발생한 이벤트(pe)에서 스위치(sw0)의 최근 상태를 읽고, DOM 모듈(this)의 출력이미지(out)의 릴레이(rly0)를 스위치와 동일하게 동작하게 하는 것입니다.

이전버전(V0.50)에서는 릴레이를 동작시키려면 SyncOut() 함수를 호출해야 했지만, V0.60 부터는 이미지를 변경하면 자동으로 출력됩니다.

이제 this->out->rly0 = pe->sw0;이 실제 릴레이를 동작시키는 과정을 순서대로 살펴봅니다.

\*핵심\* this->out->rly0 변경 ⇒ DomOut 이벤트가 발생 ⇒ 이벤트큐에 저장(파란 화살표) ⇒ 펌웨어(DomOut 이벤트핸들러를 작성하지 않았으므로 아무런 동작도 하지 않음) ⇒ eventbus 가 DomOut 이벤트를 출력이미지(out)에 저장후 드라이버(dom\_drv.c)의 HW\_SyncToOut() 함수를 자동호출(여기서 릴레이가 동작한다) ⇒ 통신버스로 DIM 에 이벤트 전달 ⇒ DIM 에는 DOM 이미지가 없으므로 동기화 하지 않고 이벤트를 버림

■ 의문 1 : 릴레이 움직이는게 왜 이리 복잡한가?

this->out->rly0 에 ON 또는 OFF 를 저장하면 릴레이가 동작합니다. 이렇게

사용자의 편의를 위하여 `eventbus` 는 내부적으로 복잡하게 동작합니다. 하드웨어에 대한 자세한 사항은 몰라도 되므로 사용법은 매우 간단해 집니다.

■ 의문 2 : 그래서 얻는게 뭐지?

수 많은 장점이 있습니다. 프로그래밍이 쉬워지고, 버스케이블만 연결하면 모든 컨트롤러가 마치 하나의 보드처럼 동작하므로 원하는 기능을 가진 모듈을 연결하면 시스템을 완성할 수 있고, 컴퓨터와 연결도 쉬워집니다. 큰 프로그램을 작은 태스크(작업단위)로 나눌 수 있습니다. 문서화가 쉬워집니다. 가독성이 좋아집니다. 별 다른 프로그램 작성 없이도 컨트롤러(컴퓨터 포함)간 통신이 자동으로 됩니다.

■ 의문 3 : 무엇부터 하면 되지?

카페([cafe.naver.com/easycontrol](http://cafe.naver.com/easycontrol))에서 최상의 개발환경인 이클립스를 이용한 개발환경을 무료로 설치하고 따라 해 보실 수 있습니다.

## 2. Quick Start

이지컨트롤러의 빠른 사용법을 익히기 위하여 스위치 상태를 읽어 릴레이를 온, 오프 시키는 간단한 펌웨어를 작성합니다. 그리고, 컨트롤러를 컴퓨터에 연결하여 동작상태를 모니터에 표시하고, PC에서 릴레이를 온/오프 시킬 수 있도록 제어하는 간단한 제어시스템을 구축하는 예를 살펴봅니다.

### 2.1. 시스템 구성

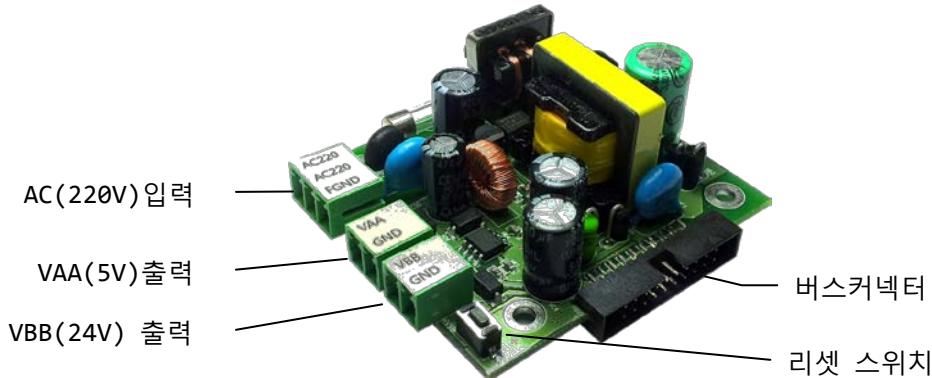
이지컨트롤러의 특징을 한마디로 말하면 **C 언어를 사용하지만 이벤트방식과 스캔방식으로 쉽게 프로그램할 수 있도록 eventbus라는 OS를 제공한다는 것입니다.** 따라서, PLC(프로그래머블로직 컨트롤러)처럼 간편함과 생산성을 그대로 살리면서도 낮은 가격에 컴팩트하게 구현할 수 있습니다.



기본적인 개발시스템의 구성입니다. 왼쪽의 모듈은 CoLinkEx 다운로더/디버거입니다. 이 모듈은 USB를 통하여 PC에 연결되고, 프로그램을 다운로드 및 디버깅할 수 있는 SWD(Serial Wired Debugger)입니다. 일종의 JTAG 디버거 인데 연결되는 선이 작아서 소형시스템의 개발에 적합합니다. (그런데, 선이 왜 이리 많나? 사실 대부분은 사용되지 않습니다)



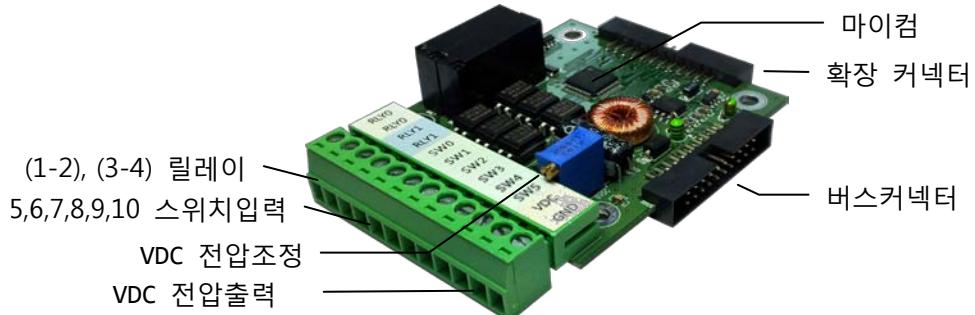
다음으로 우측에는 3 개의 모듈이 PCB 지지대를 사용해서 연결되어 있으며, 사진에는 일부 가렸지만, 뒤쪽에 플랫케이블(버스케이블-4S)로 모듈이 연결된 것을 볼 수 있습니다. 이 버스케이블에는 각 모듈을 연결하는 I2C 통신선과 시스템 리셋, VAA(5V), VBB(24V), GND 선들로 구성되어 있습니다. 가장 아래 모듈은 220VAC 입력에서 VAA, VBB 전원을 만들어 전체 모듈에 공급하는 PSM(Power Supply Module) 모듈입니다. 가운데는 스위치입력과 릴레이출력이 내장된 DIO(Digital Input Output) 모듈이고, 가장 위에는 컴퓨터의 USB에 연결하기 위한 USB 통신모듈입니다.



220VAC 입력에서 직류전원 VAA, VBB를 생성하여 버스커넥터를 통하여 연결된 다른 모듈에 전원을 공급하는 역할을 합니다. 리셋 스위치는 버스케이블에 연결된 모든 모듈의 마이컴을 리셋(초기화)시킵니다. 좌측은 AC 전원 입력부이며 휴즈와 라인필터 등이 보입니다.

\* 오픈프레임 구조이므로 감전되지 않도록 매우 주의하여 주십시오!

PSM 모듈 상단에는 DIO(Digital Input Output) 모듈이 있습니다. 이 모듈은 6 개의 온오프 접점입력과 2 개의 릴레이 출력으로 구성되어 있습니다.



펌웨어 개발에 사용할 모듈입니다. 2 개의 릴레이 접점출력과 6 개의 비절연형 스위치 접점 입력, 그리고 VDC 가변전압 출력이 있습니다. VDC 전압은 필요에 따라 가변저항으로 조정할(출고때 12V에 설정) 수 있습니다. 버스 커넥터는 20 핀을 사용하지만 센서모듈과 같은 소형모듈에서는 10 핀을 사용하기도 합니다.

マイ컴은 NXP 사의 32 비트 CORTEX-M0 저전력 마이컴인 LPC1114 입니다. 32KB 플래시와 8KB SRAM 을 가지며 48MHz 의 고속동작을 합니다. 마이컴의 IO 포트중 일부는 컨트롤러의 기본동작에 사용되지만 여분의 핀은 확장 커넥터로 연결되어 있으므로 필요에 따라 사용자가 사용할 수도

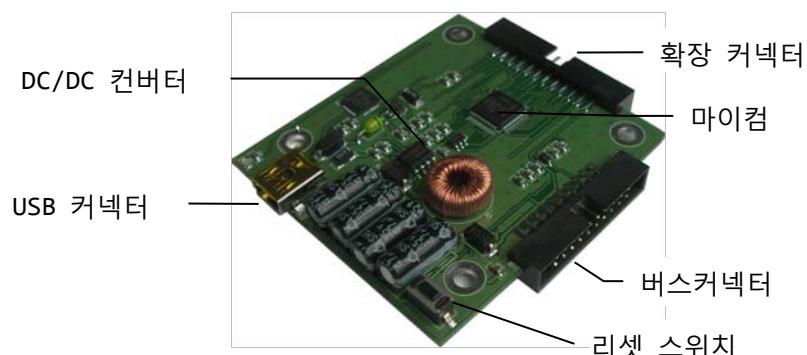
있습니다. 이외에도 전원, UART, SWD 도 확장커넥터로 연결되어 있어서 사용자가 필요에 따라 회로를 확장할 수 있도록 제공됩니다.

사용자는 필요에 따라 이 확장커넥터를 통하여 사용자가 제작한 회로를 연결할 수 있습니다 (단, 사용자가 제작한 회로와 관련 프로그램은 기술지원 대상이 아닙니다). 이런 용도로 확장용 모듈인 EXP 모듈이 별도로 제공하고 있습니다.

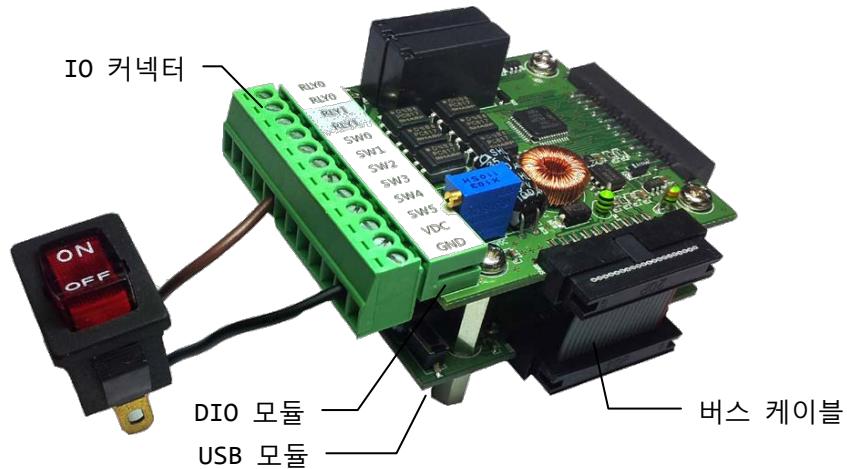
CoLinkEx, PSM, DIO 3 개의 모듈만 있으면 지금 우리가 제어하고자 하는 시스템을 구성할 수 있습니다. 즉, DIO 에 연결된 스위치의 입력을 읽고, 스위치 상태에 따라서 릴레이를 구동하는 시스템을 구성할 수 있습니다. 또한, USB, UUB, UUM, U2M, LAN 등의 모듈을 사용하면 PC 에 연결해서 화면에 DIO 의 스위치 상태를 표시하거나 마우스로 릴레이를 동작시킬 수 있습니다. (개발시에는 CoLinkEX 에 내장된 COM 포트를 사용해도 동일하게 PC 와 통신을 연결해서 테스트할 수 있습니다.)

이러한 시스템은 매우 획기적이고 지금까지 볼 수 없었던 저희만의 독자적인 시스템이기 때문에 언뜻 이해가 되지 않을 수 있습니다. PC 에서 저희가 제공하는 `eventbus_net_0.60` 라이브러리(`eventbus.dll`)를 이용하여 닷넷언어인 C#.net, C++.net, VB.net 등으로 프로그램을 작성해서 릴레이를 켜고, 스위치가 눌러졌는지 확인하고, 아날로그 입력전압을 읽어서 그래프를 그리고, 스텝모터를 회전시키는 작업을 단지 몇 라인의 프로그램으로 완성도 높게 구현할 수 있습니다.

여기서 컴퓨터와 연결하기 위한 용도로 사용되는 전용의 모듈이 아래 그림의 USB 모듈입니다.



그림의 “USB 커넥터”를 컴퓨터와 연결합니다. 확장커넥터, 마이컴, 버스커넥터는 다른 모듈과 동일하므로 설명을 생략합니다. 그리고 DC/DC 컨버터가 내장되어 USB 에서 공급되는 5V 전원에서 VAA(5V), VBB(24V) 전원을 만들어 버스케이블에 공급합니다. 그러므로 큰 전류를 필요로 하지 않는다면 PSM 모듈이 없어도 USB 모듈만으로도 전원공급이 가능합니다만, 당연히 전원공급 기능이 있는 USB HOST(예, 컴퓨터)에 연결되어 있어야 합니다. 이 경우 구성은 아래와 같이 PSM 모듈을 제거하고 간략화 할 수 있습니다.



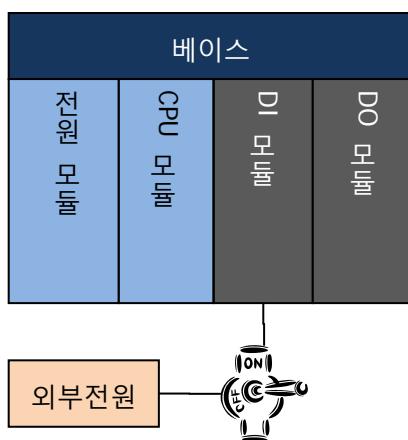
또 다른 구성으로는 아래 그림처럼 DIO + UUM 으로 구성해도 마찬가지 입니다. UUM 에도 DC/DC 컨버터가 내장되어 있으므로 USB 에서 공급되는 5V 전원에서 VAA(5V), VBB(24V) 동작전원을 만들어 버스케이블에 공급합니다.



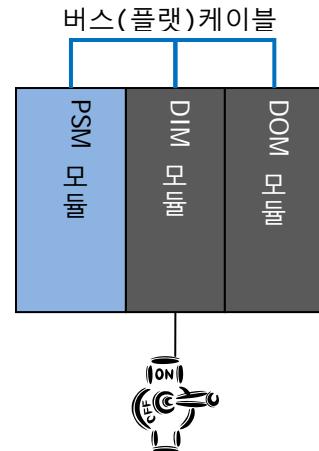
USB 와 UUM 의 차이점은 USB 에는 마이컴이 있으므로 독자적으로 프로그램을 저장할 수 있지만, UUM 에는 마이컴이 없으므로 DIO, DOM 등과 같이 마이컴이 있는 기능모듈과 함께 사용해야 합니다. (그러므로, UUM, U2M 은 마이컴이 있는 모듈과 확장커넥터를 연결해야 합니다)

## 2.2. 다른 제어시스템과의 비교

이지컨트롤러의 각 모듈은 비록 저가의 간단한 구성이지만, 버스케이블로 연결되면 상당히 복잡한 자동화 시스템을 구현할 수 있습니다. 그러면, PLC를 사용하는 경우와 시스템 구성을 비교해 보겠습니다.



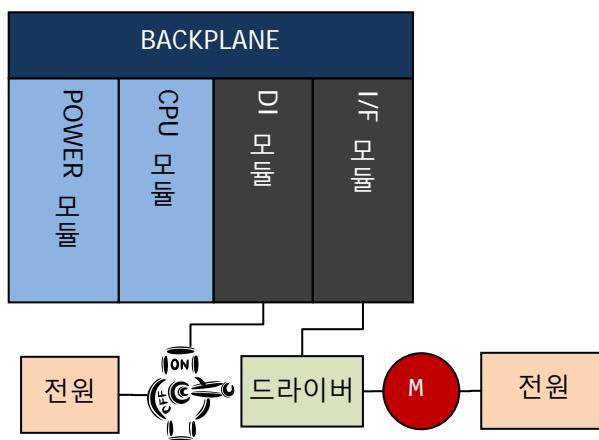
(a) PLC 구성의 예



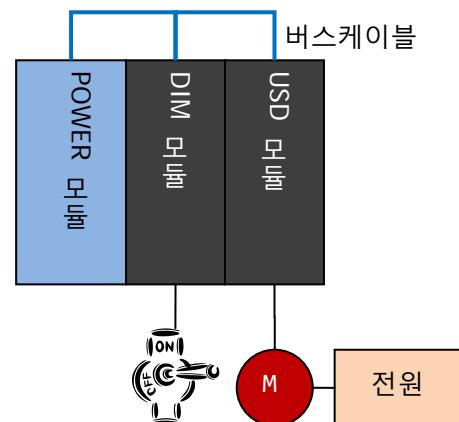
(b) 이지컨트롤러 구성의 예

그림에서 보듯이 PLC는 기본적으로 베이스, POWER 모듈, CPU 모듈이 있어야 하고, 추가적으로 구현하고자 하는 기능에 따라서 DI, DO 모듈 등이 필요합니다. 또한 PLC는 신호처리용 시스템이므로 외부와 절연된 구성을 가집니다. 그 결과 안정성은 높아지지만 별도의 드라이버를 필요로 하는 등 구성이 복잡해지므로 고가의 시스템이 됩니다.

반면, 이지컨트롤러는 각 모듈에 마이컴이 내장되어 있으며 베이스를 저가의 플랫케이블로 대체하여 비용이 저렴할 뿐만 아니라, 모듈이 서로 떨어진 곳에 위치할 때도 편리하게 사용할 수 있습니다. 통계에 따르면 PLC 응용의 80%는 단순 시퀀스제어에 사용되고 있습니다. 이런 경우라면 굳이 고가의 PLC를 사용하지 않아도 되는 경우가 대부분입니다. 이지컨트롤러가 경쟁력있는 대안이 될 수 있을 것입니다.



(a) PLC 구성의 예



(b) 이지컨트롤러 구성의 예

그림은 조금 더 복잡한 구성으로 스텝모터 구동시스템을 구성한 예 입니다. 그림의 (a)처럼 PLC 는 신호처리 시스템이므로 모터를 직접구동할 수 없으므로 외부에 모터드라이버를 별도로 부착하여야 합니다. 이는 추가적인 비용뿐만 아니라 모듈간의 인터페이스를 위한 별도의 통신프로토콜에 대한 학습을 필요로 하여 단시간내에 쉽게 시스템을 구성하기에 어려움이 있습니다. 반면 그림 (b)처럼 이지컨트롤러를 사용한 경우는 USD 에 모터드라이버가 내장되어 있으므로 추가적인 외부 드라이버가 불필요하여 가격이 매우 저렴할 뿐만아니라 별도의 통신 프로토콜에 대한 학습도 필요하지 않아서 매우 단시간에 시스템을 완성할 수 있습니다. 또한 V0.60 부터 제공되는 텍스트(AT) 명령을 사용하면 프로토콜을 몰라도 영어로 쉽게 동작시킬 수 있습니다.

또한 프로그램의 측면에서 살펴보면, PLC 를 사용하기 위해서는 래더 디어그램과 같은 학교에서 배운 적이 없는 새로운 언어를 학습해야 합니다. 그런데, 이런 언어는 배울 곳도 마땅치 않습니다. 인터넷에서 동영상 강좌도 구하기 어렵습니다. 또한 구현에 있어서 많은 제약이 따릅니다. 왜냐하면, 원래 래더 디어그램은 릴레이 제어반 기술자들이 쉽게 PLC 로 옮겨 타도록 하기 위하여 쉬운 사용방법에 초점을 맞추다 보니 기능이 제한될 수 밖에 없습니다.

그래서 이지컨트롤러는 C 언어를 사용합니다. 비슷한 몇몇 컨트롤러 처럼 변형된 C 언어를 사용하지도 않습니다. 그런 언어는 문제가 발생하면 해결하기 매우 어려워지고 회사마다 사용법이 달라져서 경력에도 도움이 되지 않습니다. C 언어는 학교에서 가장 많이 강의하고 있을 뿐만 아니라, 가장 성능이 우수하고, 자료도 풍부합니다. 아직 배우지 않았다 하더라도 인터넷에 동영상 강좌가 매우 많아서 언제든지 학습할 수 있습니다. 그리고 C 언어를 사용하면 매우 다양한 프로그램을 작성할 수 있습니다. 흔히들 “C 언어로 할 수 없는 것은 컴퓨터로 할 수 없는 것이다”라고 합니다.

하지만, 단점이 한가지 있습니다. C 언어는 포인터를 자주 사용합니다. 특히 포인터는 초보 프로그래머들이 가장 어려워하는 부분이기도 하고 에러를 가장 많이 일으키는 부분이기도 합니다.

그러나, 크게 걱정할 필요는 없습니다. 다음절에서 살펴보겠지만, 이지컨트롤러는 이미 상당부분의 펌웨어가 라이브러리로 작성되어 제공되므로, 여러분이 매우 특별한 프로젝트를 하는 것이 아니라면(이런 프로젝트를 기획중이라면 이미 C 언어에 경험이 많겠지만), 특별히 포인터와 같은 C 언어의 어려운 기능들을 사용할 경우는 많지 않을 것입니다.

특히, 이지컨트롤러는 윈도우 프로그래밍에서 사용하는 이벤트방식 프로그래밍을 사용하고 있습니다. 물론 윈도우만큼 복잡하고 다양한 기능을 제공하지는 않지만 제어용 시스템에서는 그런 복잡한 기능은 사실 필요하지도 않습니다. 그러나 이지 컨트롤러가 제공하는 이벤트 방식은 소형 제어시스템에 최적화된 매우 편리한 프로그래밍 환경을 제공하므로 이를 이용하면 제어용 프로그래밍이 매우 간략하게 되어 통신프로토콜과 포인터에 익숙치 않은 C 언어의 초급사용자라 할지라도 쉽게 프로그램을 작성할 수 있습니다.

만일, PLC 에 익숙한 사용자라면 eventbus 에서 제공하는 ezplc.h 를 사용하면 PLC 방식의 스캔프로그램으로 작성할 수도 있습니다.

뿐만 아니라, PC 환경에서 사용할 수 있는 단넷용 라이브러리인 `eventbus.dll` 을 소스코드와 함께 제공하므로 PC 에 연결해서 이지컨트롤러를 제어하는 어플리케이션 프로그램을 작성하는 것도 매우 쉽습니다. 또한 V0.60 부터 제공되는 AT 명령(영단어로 된 명령)을 사용하면 프로토콜을 몰라도 아래처럼 하이퍼터미널 또는 제공되는 `ezconfig.exe` 프로그램 또는 C, JAVA 등의 언어에서도 쉽게 동작시킬 수 있습니다.

```
4 set rly0 on      // ID=4 인 DIO 모듈의 rly0(0 번 릴레이)를 온 시킨다
: 4 ok            // 정상적으로 수행하였음을 알린다
4 get sw0          // ID=4 인 DIO 모듈의 sw0(0 번 스위치)를 읽어온다
: 4 sw0 1          // 스위치가 1(온)임을 알린다
```

여러분 회사의 생산라인 일부를 저가로 신속하게 자동화하거나, 또는 테스트지그와 같은 제품을 만들어 납품한다고 생각해 보십시오. 또는 출업작품을 만들어 전시한다고 생각할 수도 있고, 아니면 긴급히 제품의 프로토타입을 만들어야 하는 경우 PLC 와 이지컨트롤러를 비교해 보십시오. 쉽게 선택할 수 있을 것입니다. PLC 로 라인을 자동화 할수는 있지만, 전용기에 PLC 를 넣는다면 이미 그 제품은 경쟁력을 상실하게 됩니다.

	PLC	싱글보드	이지컨트롤러
구성비용	X	◎	○
개발언어	LD(래더)	C / C++	C / eventbus
확장성	◎	X	◎
학습비용	○	X	○
사이즈	X	X	◎
유연성	X	○	◎
생산성	◎	X	◎
안정성	◎	X	○



PLC, 이지컨트롤러 사이즈 비교

## 2.3. PC에서 모듈 제어하기

---

PC에서 모듈을 제어하기 위해서는 우선 개발환경을 설치해야 합니다. 이지컨트롤러는 마이크로소프트의 닷넷개발환경을 사용하고 있습니다. 여러분도 비주얼스튜디오를 설치하고 C#.NET, C++.NET, VB.NET과 같은 닷넷언어를 사용하면, 저희가 제공하는 `eventbus.dll` 라이브러리를 사용할 수 있고, 매우 쉽게 프로그램을 작성할 수 있습니다.

### 2.3.1. 개발환경 설치하기

펌웨어를 개발하기 위해서는 아래의 몇가지 프로그램을 설치해야 합니다.

- **비주얼 스튜디오**

마이크로소프트의 통합개발환경(IDE)입니다. 무료로 사용할 수 있는 Express 버전을 제공하므로 이를 다운로드 받아서 설치하면 됩니다.

- **CP2102 드라이버**

UUM, USB, CoLinkEx를 컴퓨터에 연결할 때 필요한 USB통신드라이버 프로그램입니다. (네이버카페에서 다운로드 받을 수 있습니다)

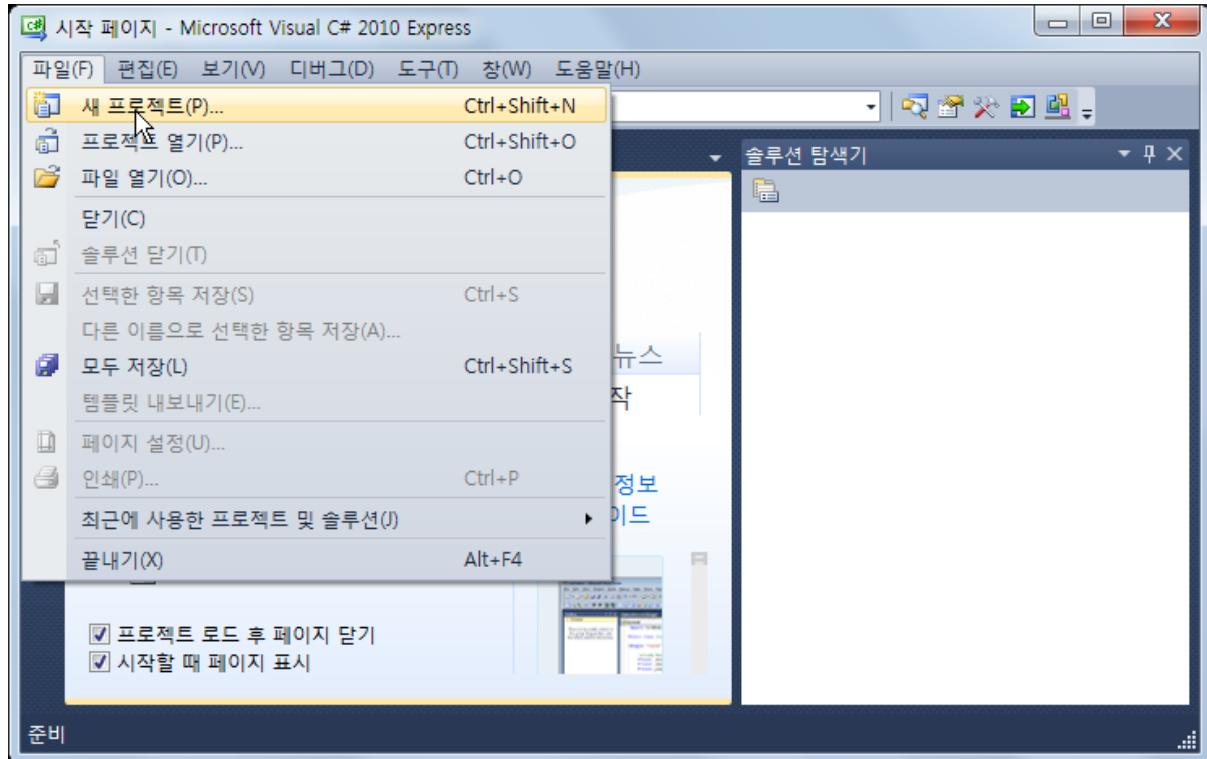
자세한 설치과정은 “[프로그래밍 실습](#)” 메뉴얼을 참고해 주십시오.

### 2.3.2. PC에서 모듈 제어하기

LAN, USB, U2M, UUM(UUB 대체품) 또는 CoLinkEX 디버거를 이용하면 모듈을 컴퓨터에 연결시켜 컴퓨터에서 모니터링 또는 제어할 수 있습니다. (ColinkEx는 개발용 툴이므로 현장에 설치하여 사용할 때는 LAN, USB, U2M, UUM 모듈이 적절합니다)

PC에서 제어프로그램을 쉽게 작성할 수 있도록 `eventbus.dll` 라이브러리를 제공하고 있습니다. `eventbus.dll`을 이용하면 닷넷언어인 C#.NET, C++.NET, VB.NET을 사용해서 프로그램을 쉽게 작성할 수 있으며, 그외의 언어인 경우는 프로토콜(프로토콜 참고) 또는 AT 명령을 이용하여 프로그램을 작성할 수 있습니다.

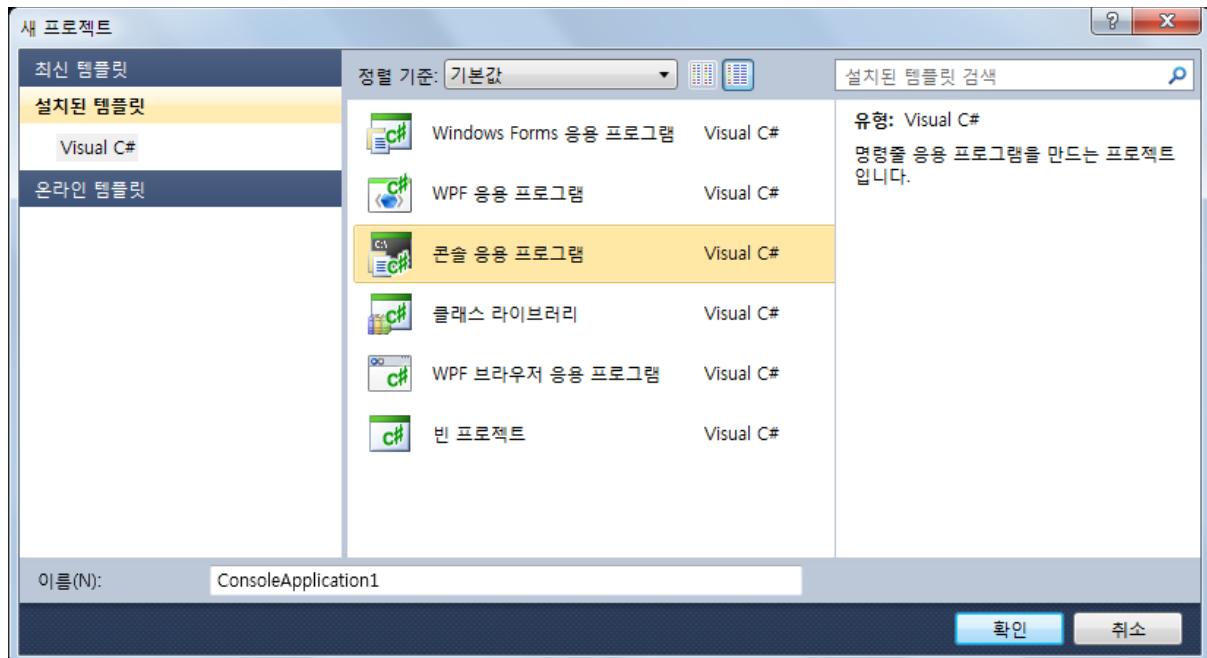
`eventbus_net_0.60.zip`에 포함된 모든 프로그램은 비주얼스튜디오 2008 버전으로 제공됩니다. 그러나, 상위버전에서도 사용할 수 있습니다. 상위버전인 비주얼스튜디오 2010에서 솔루션을 열면 간단한 변환과정을 거쳐 2010 버전으로 변환되므로 사용에 별다른 어려움은 없습니다. 아래에서는 Visual C# 2010 Express 버전을 사용하여 테스트를 진행 하겠습니다.



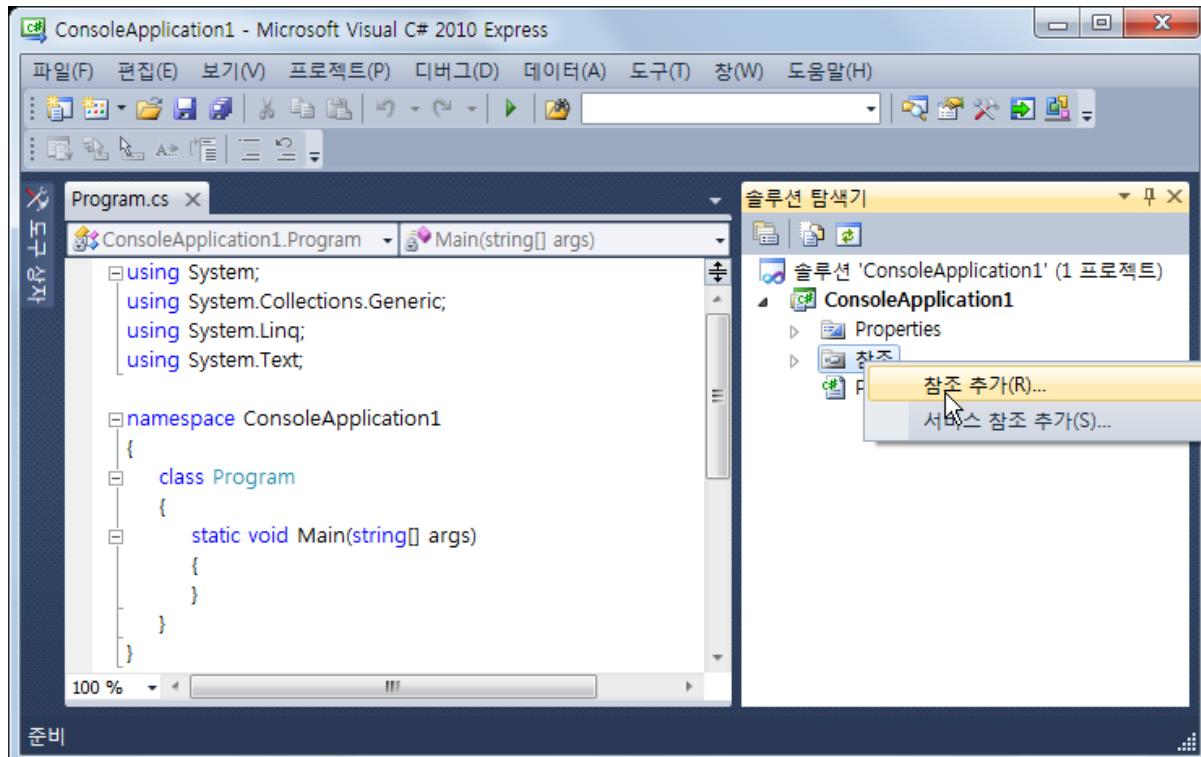
위 그림처럼 “파일/새프로젝트”를 선택합니다.

콘솔프로그램과 윈도우 폼 프로그램 두 가지를 살펴봅니다. 콘솔프로그램은 `eventbus_net`의 동작방식을 이해하기에 적합하지만, 실용적인 프로그램은 “Windows Forms 응용프로그램”이나 “WPF 응용 프로그램”을 사용하는 것이 일반적입니다.

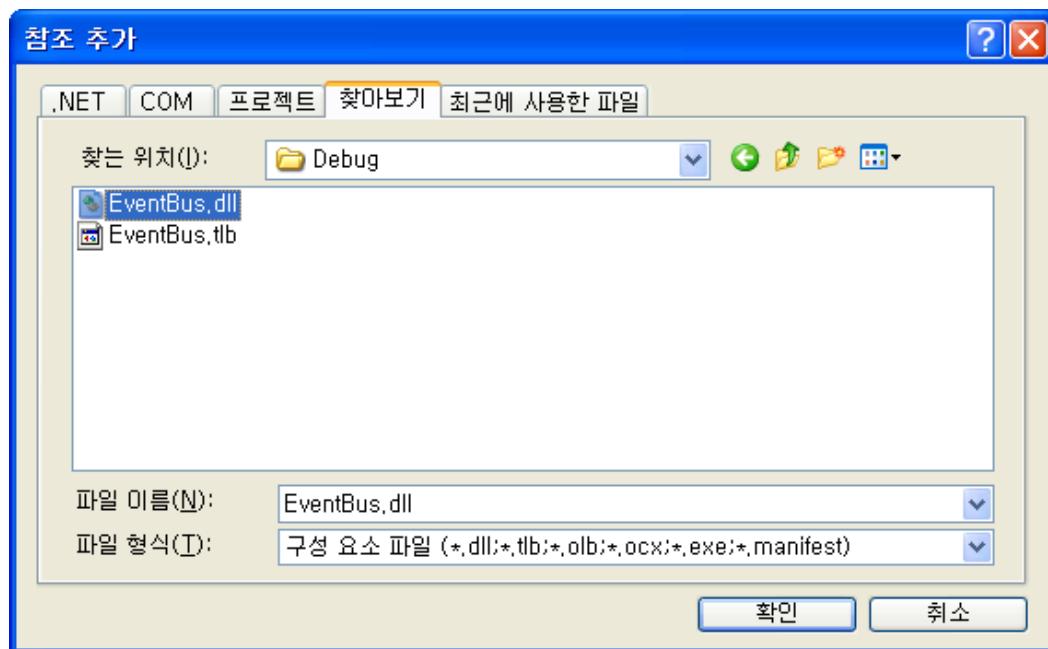
### 2.3.3. 콘솔 응용프로그램



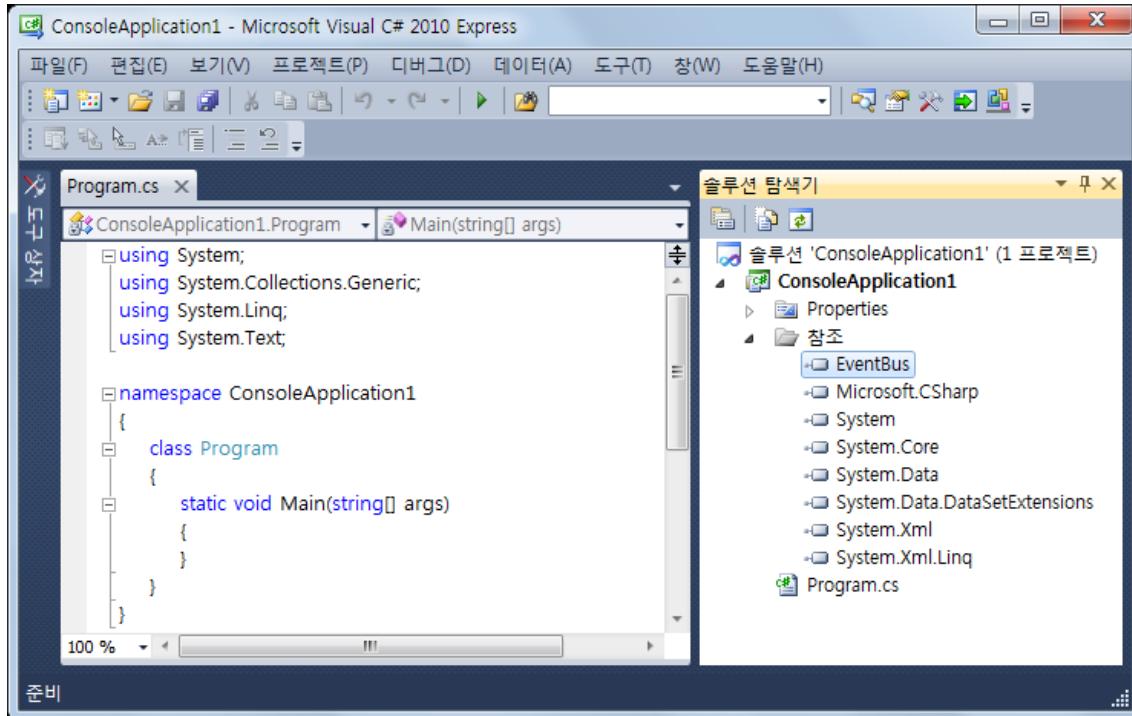
[콘솔 응용 프로그램]을 선택 후 [확인] 버튼을 눌러 새 프로젝트를 생성합니다.



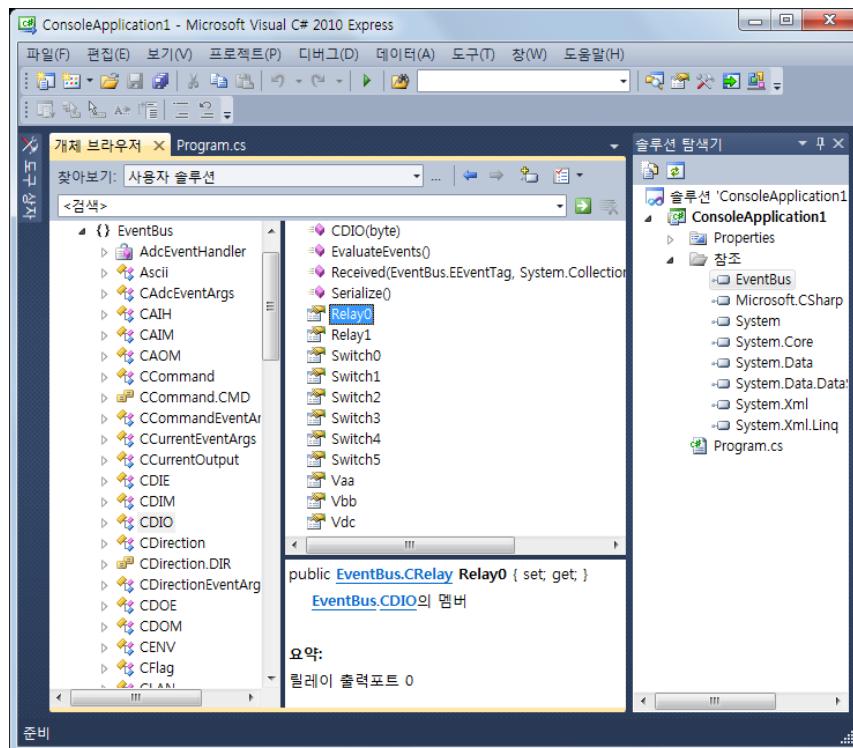
솔루션 탐색기의 [참조]에서 우측버튼을 누르고 [참조추가]를 선택하면 아래와 같이 라이브러리를 추가할 수 있는 대화창이 열립니다.



제공된 `eventbus_net_0.60\eventbus\eventbus\bin\Debug\EventBus.dll` 을 선택하면 참조(References)에 추가됩니다. (미리 `eventbus_net_0.60.zip` 을 다운받아서 Projects 폴더에 저장해야 합니다. 윈 7 의 경우는 `C:\Users\(\사용자명)\Documents\Visual Studio 2010\Projects` 에 압축을 풀고, 윈 XP 의 경우는 `내문서\Visual Studio 2010\Projects` 에 압축을 풀면 됩니다)



그림은 `EventBus.dll` 이 참조에 추가된 모습입니다. `EventBus.dll` 이 추가되면 이제 `EventBus.dll` 라이브러리에서 제공하는 클래스를 사용할 수 있습니다. 여기에는 연결된 여러 종류의 컨트롤러 모듈과 연동하여 작업하기 위한 모든 지원내용이 포함되어 있으므로 매우 편리하게 제어 프로그램을 작성할 수 있도록 도와줍니다.

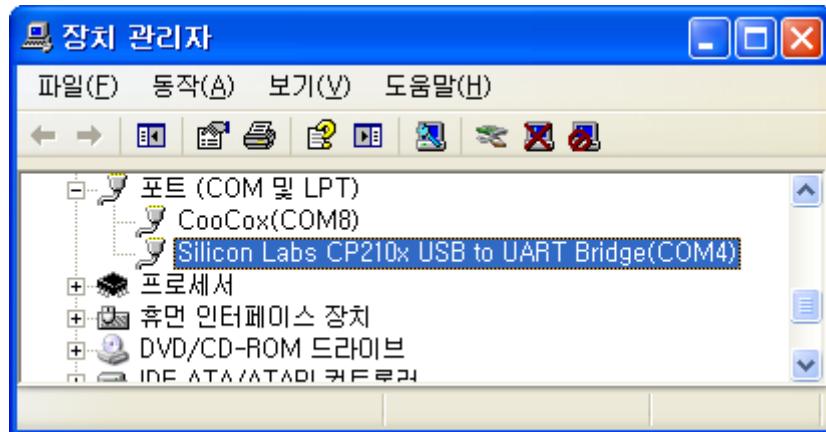


참조의 `EventBus` 를 더블클릭해 보십시오. 아래 그림처럼 사용할 수 있는 기능들을 확인할 수 있습니다.



그림은 우리가 프로그램 작성에 사용할 모듈의 구성입니다. PSM, DIO 모듈이 버스케이블로 연결되어 있고, PCB 서포터로 체결되었습니다. 스위치가 DIO 의 SW0 과 VDC 사이에 연결되어 있으며, ColinkEx 가 DIO 의 확장케이블에 연결되어 있습니다.

검은색 USB 케이블은 SWD(Serial Wired Debug) 기능으로 모듈의 펌웨어를 수정하거나 디버깅할 때 필요합니다. 컴퓨터와 연결하여 응용프로그램을 작성할 때는 연결하지 않아도 됩니다. 흰색 USB 케이블은 이번 예제처럼 컴퓨터와 연결하여 응용프로그램을 작성할 때 필요한데, 컴퓨터에 가상 COM 포트를 생성해 줍니다. 앞에서 설명했듯이 반드시 드라이버를 설치하셔야 합니다. 정상적으로 연결되면, 내컴퓨터/속성에서 하드웨어 탭에 있는 장치관리자에서 그림처럼 연결된 포트를 확인할 수 있습니다.



USB, U2M, UUM(UUB 대체품)을 연결해도 마찬가지로 연결된 포트를 확인할 수 있습니다. 그림에서는 COM4에 연결되었지만 컴퓨터마다 차이가 있으므로 반드시 확인해 주십시오.

```

1:  using System;
2:  using EventBus;
3:  namespace ConsoleApplication1
4:  {
5:      class Program
6:      {
7:          static Serial com = new Serial();
8:          static CDIO dio = new CDIO(4);      // ID = 4
9:
10:         static void Main(string[] args) {
11:             com.Connect("COM4", 115200);
12:             if (!com.Connected) {
13:                 Console.WriteLine("Connection failed");
14:             }
15:             else {
16:                 Console.WriteLine("Connected to COM4");
17:             }
18:         }
19:     }
20: }
```

```

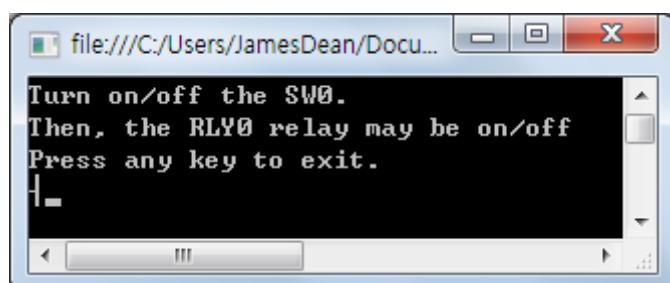
12:             Console.WriteLine("USB Connection failed!");
13:         }
14:     else {
15:         dio.sw0.OnChanged += new LogicEventHandler(sw0_OnChanged);
16:
17:         Console.WriteLine("Turn on/off the SW0.");
18:         Console.WriteLine("Then, the RLY0 relay may be on/off");
19:     }
20:
21:     Console.WriteLine("Press any key to exit.");
22:     Console.ReadKey();
23: }

24: static void sw0_OnChanged(object sender, CLogicEventArgs args) {
25:     dio.rly0.State = args.State;
26:     com.Synchronize(dio);
27: }
28: } // class
29: } // namespace

```

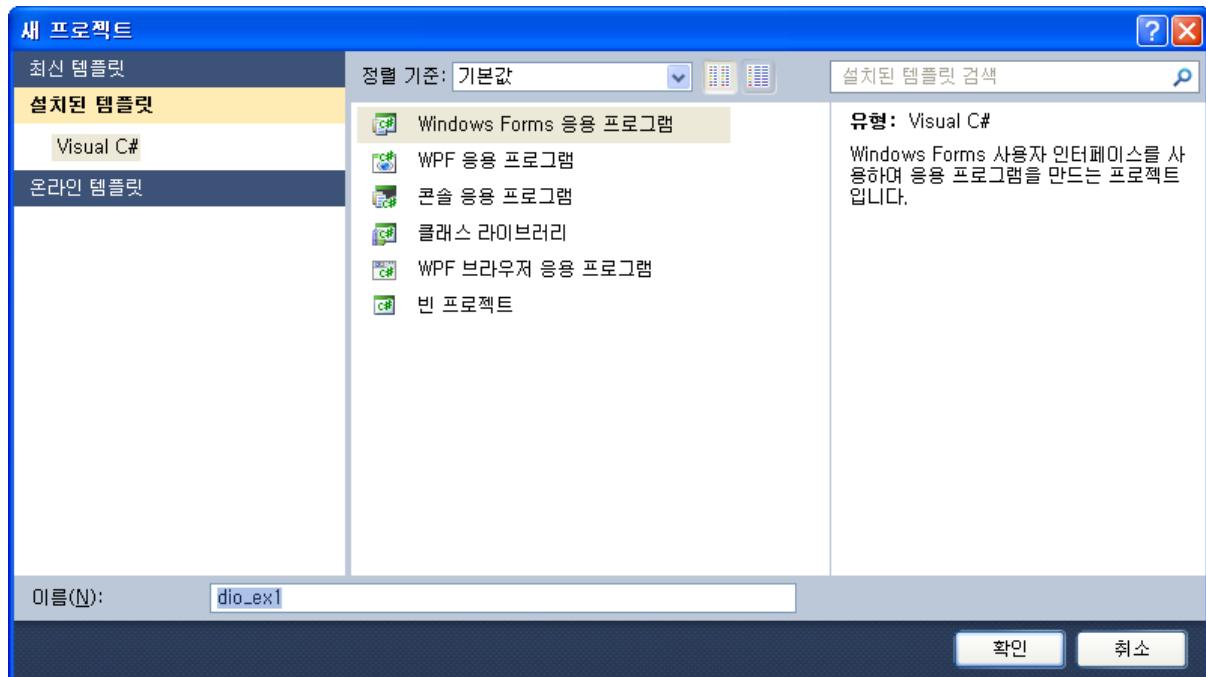
Program.cs에 입력할 소스코드입니다.

- 7: 연결된 ColinkEx(또는 UUM,U2M도 동일)의 시리얼포트를 사용하기 위해 com을 생성합니다.
- 8: DIO 모듈을 엑세스할 수 있도록 이미지를 생성합니다.(ID는 4 번을 사용했습니다)
- 10: USB,UUM,CoLinkEX의 경우 CP2102 USB-to-UART 브릿지 칩을 사용하므로 CP2102 칩에 대한 USB 드라이버가 설치되어 있어야 합니다. 0.60 부터 디폴트 통신속도는 115,200bps입니다. 다른 통신속도를 사용하려면 ezconfig.exe 프로그램으로 통신속도를 설정해 주십시오(V0.60 부터는 UART 통신속도를 ezconfig.exe에서 설정할 수 있습니다). 그러나, U2M의 경우 하드웨어의 제한으로 최고속도는 115,200bps입니다. com.Connect(포트명,통신속도) 메서드를 사용하여 모듈에 접속합니다.
- 11:~13: 통신이 연결되지 않은 경우 콘솔에 메시지를 표시합니다.
- 15: DIO에 연결된 스위치의 상태에 변경이 일어난 경우에 eventbus\_net은 OnChanged 이벤트를 발생합니다. 이때 실행할 이벤트핸들러(sw0\_OnChanged)를 등록합니다.
- 22: 키보드에서 어떤 키가 눌러질 때까지 대기하고, 눌러지면 프로그램을 종료합니다.
- 24:~27: DIO의 sw0의 OnChanged 이벤트에 대한 핸들러입니다. 이전에 펌웨어에서 작성한 DioIn 이벤트핸들러와 비교하면 매우 유사합니다. args.State에는 변경된 스위치 상태가 전네지므로, 스위치 상태를 dio 이미지의 rly0에 넣으면 스위치 상태에 따라서 릴레이 상태가 바뀌게 됩니다. 실제 릴레이가 동작하려면 DioOut 이벤트를 DIO에 전송해야 하는데, 20: com.Synchronize(dio)가 그 역할을 합니다. com에게 dio의 출력이미지를 전송하도록 요청합니다.(DioOut 이벤트가 DIO 모듈에게 전송됨)

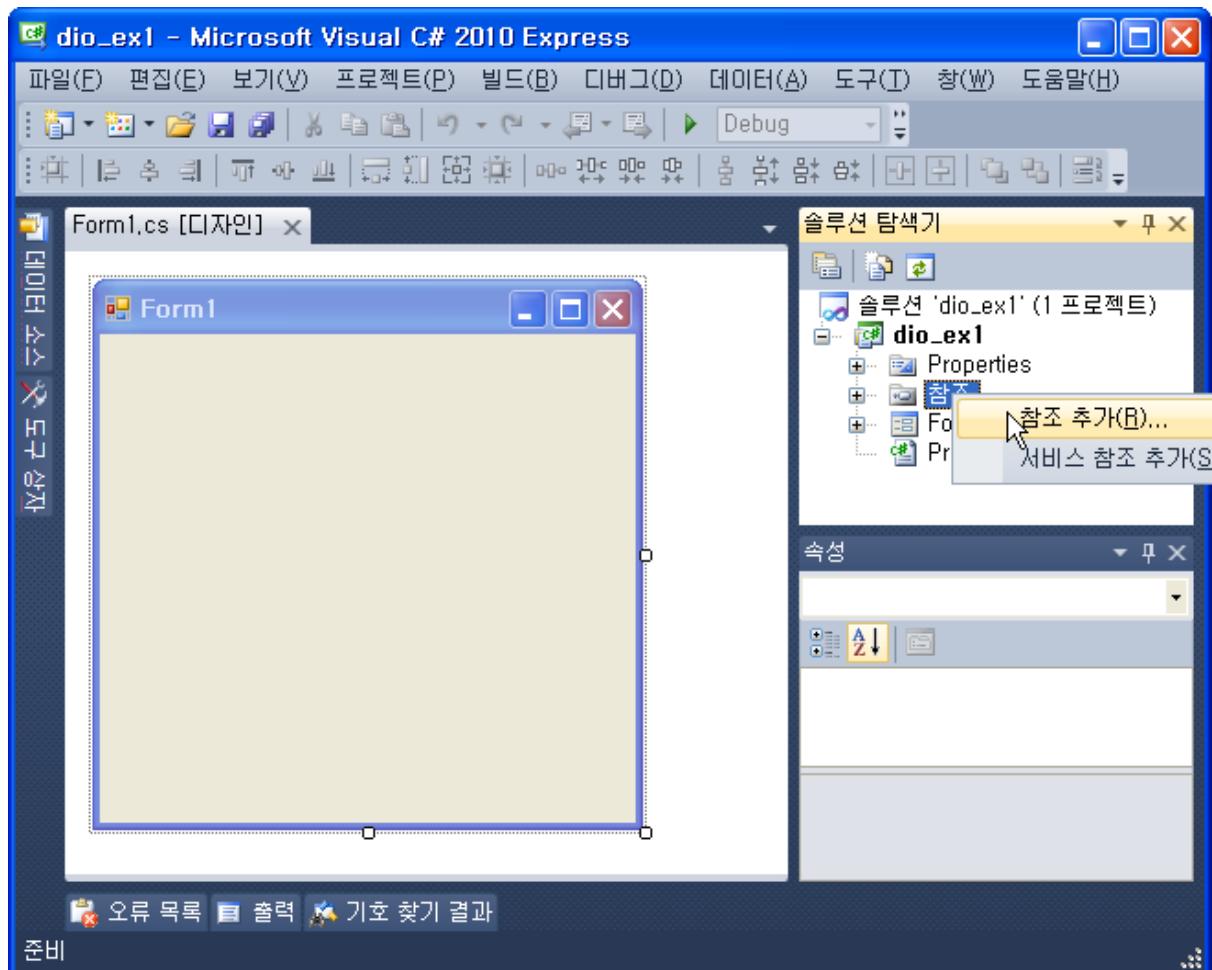


별다른 메시지 없이 위 그림처럼 표시되면 시리얼통신이 제대로 연결된 것입니다. 이제 DIO 모듈의 SW0에 연결된 스위치를 동작시키면 릴레이가 스위치에 따라서 동작하는 것을 볼 수 있습니다.

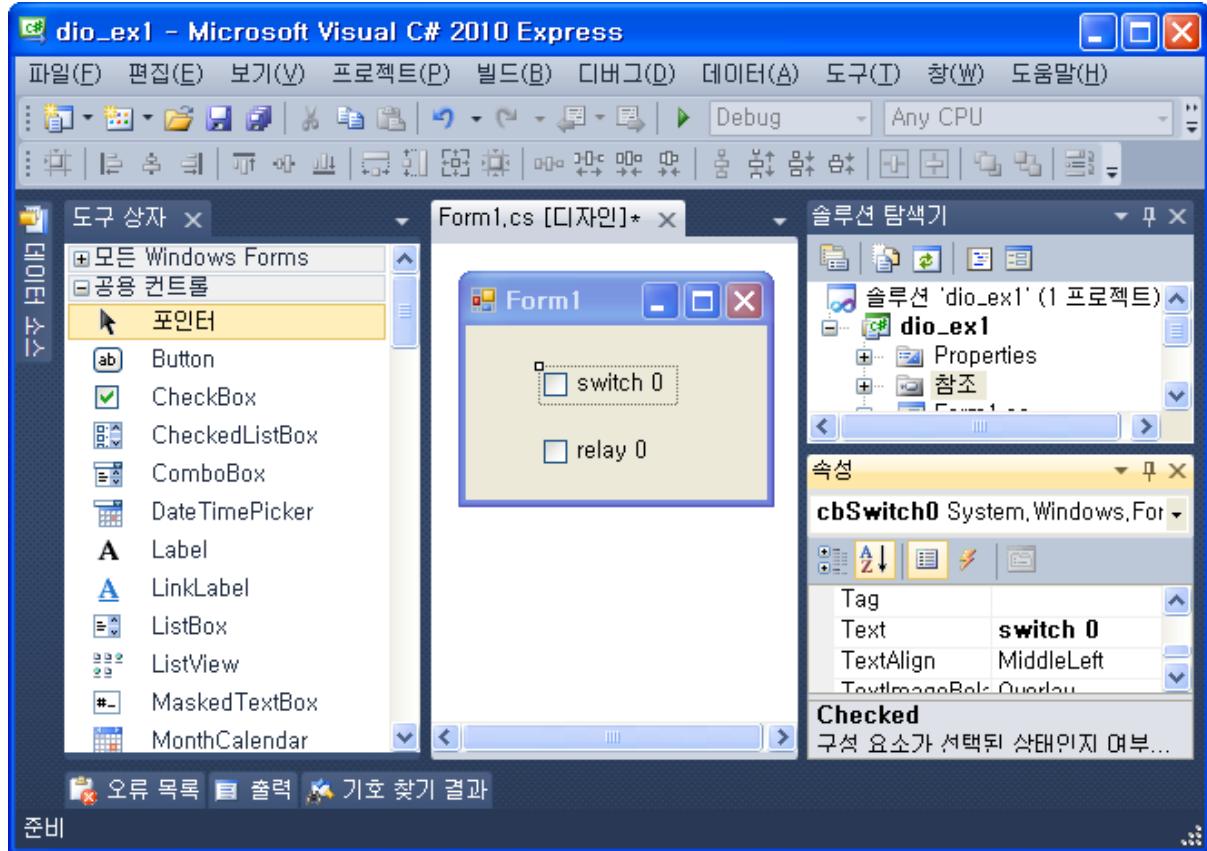
### 2.3.4. 윈도우 폼 응용프로그램



[Windows Forms 응용프로그램]을 선택하고, 프로젝트 이름은 “dio\_ex1”으로 입력합니다.



이벤트버스를 사용하기 위해서 앞의 예제처럼 EventBus.dll 을 참조에 추가합니다.



솔루션 탐색기에서 **Form1.cs** 를 더블클릭하여 그래픽 디자인을 열고 도구상자에서 체크박스를 가져와서 그림과 같이 디자인 합니다. 이때 속성은 다음 표를 참조하여 입력하시기 바랍니다.

컨트롤	속성	값
<input type="checkbox"/> switch 0	Name	cbSwitch0
<input type="checkbox"/> relay 0	Name	cbRelay0
	Text	switch 0
	Text	relay 0

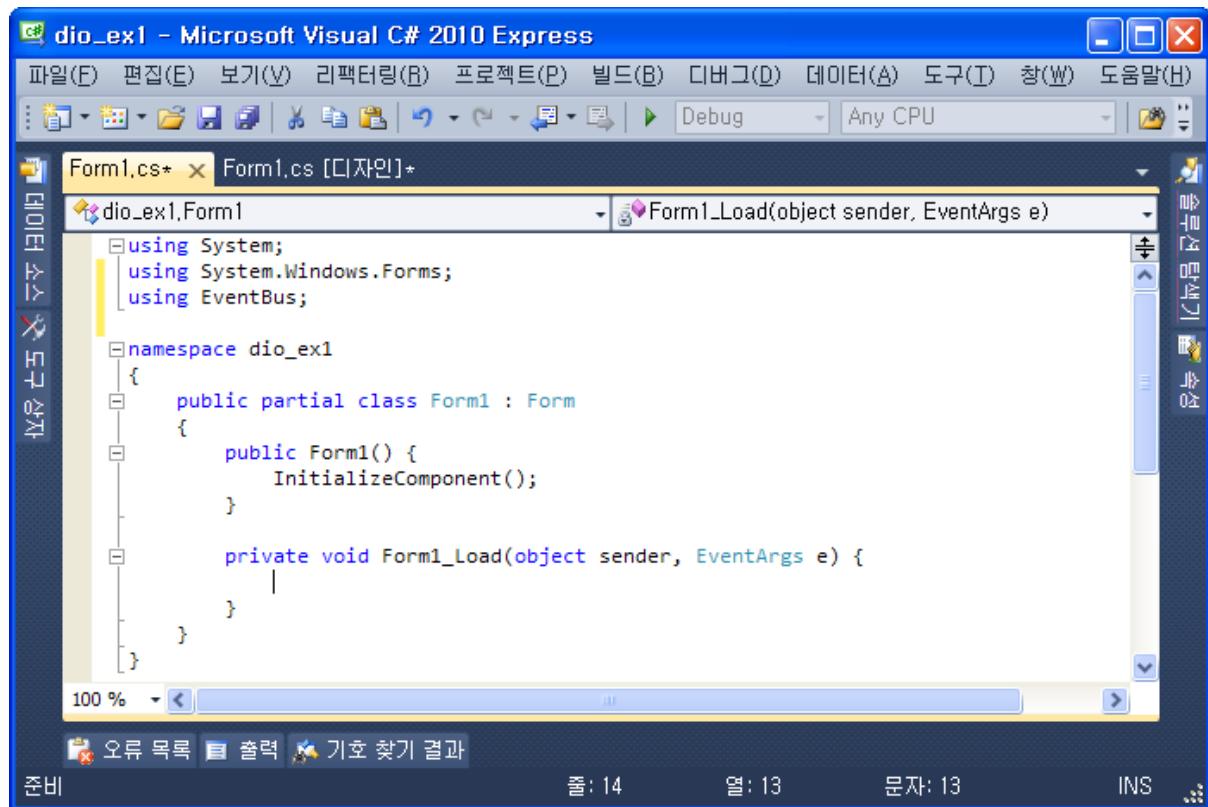
구현하고자 하는 기능은 다음과 같습니다.

- 실행하면 DIO 모듈과 통신을 연결한다.
- DIO 에 연결된 0 번 스위치를 읽어서 cbSwitch0 체크박스에 체크로 표시한다.
- cbRelay0 체크박스를 클릭하면 DIO 의 0 번 릴레이가 ON/OFF 된다.

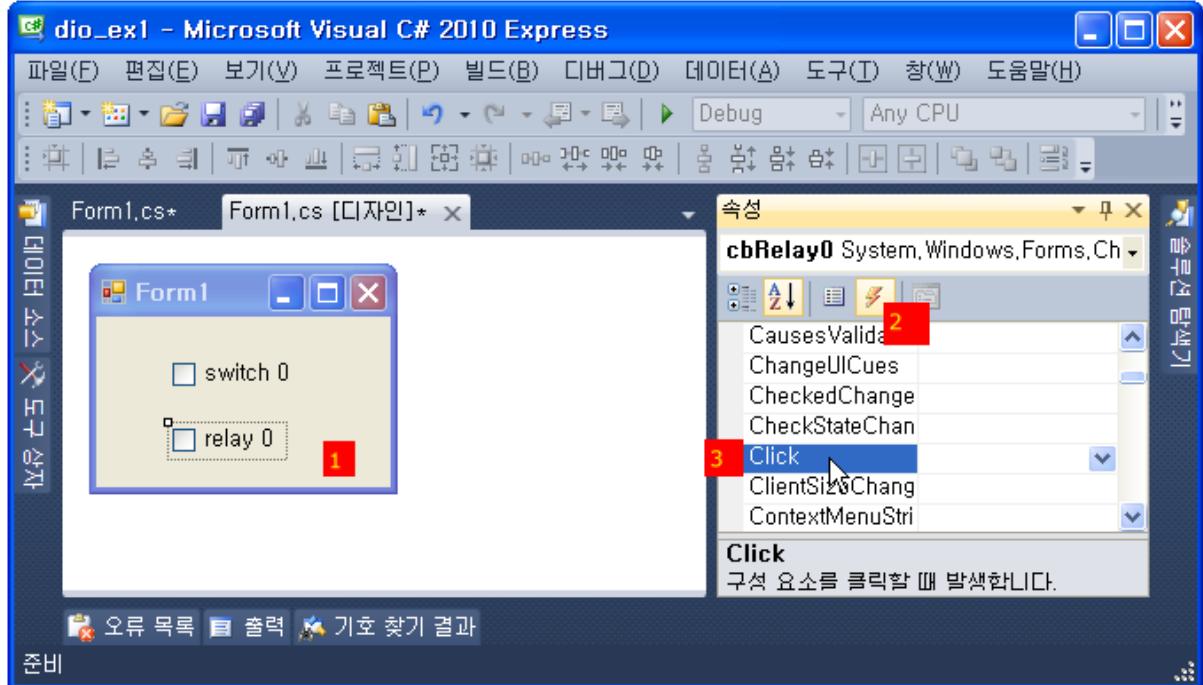
이 정도 프로그램은 eventbus\_net 을 사용하지 않더라도 그리 어렵지는 않지만, 그래도 금방 쉽게 뚝딱하고 만들기에는 꽤 부담스러울 거라고 생각됩니다. 하드웨어 모듈과 통신프로토콜 규약을 정하고, 통신프로그램을 작성하는 등의 까다로운 작업을 직접해야만 합니다. 그러나, EventBus.dll 라이브러리를 이용하면 매우 쉽게 구현할 수 있습니다.

단, 여기에서 C# 프로그램의 문법부터 모두 설명할 수는 없으므로 언어의 사용법은 알고 있다고 가정하겠습니다. 다른 단넷언어를 사용해도 좋습니다. C#언어의 가장 큰 장점은 높은 생산성입니다. 사실 이용자의 입장에서는 세련된 프로그램도 좋지만, 빨리 개발해서 빨리 사용할 수 있는 것이 매우 중요합니다.

이제 윈도우 프로그래밍의 특징인 이벤트 프로그래밍을 시작하겠습니다. 그래픽디자인에서 Form1 폼을 더블클릭하면 다음과 같이 Form1\_Load 이벤트핸들러를 작성할 수 있는 코드창이 열리고, Form1\_Load 이벤트핸들러의 뼈대가 자동으로 생성됩니다. 이런 방식으로 이벤트핸들러를 추가해 가면 됩니다. 이외에도 여러가지 방법이 있으므로 자세한 사용법은 관련도서를 참고하십시오.



참조에 추가한 EventBus.dll 을 사용하려면, 그림의 셋째 줄처럼 using EventBus;를 추가합니다. 그리고, 방금 자동추가된 Form1\_Load 이벤트핸들러를 작성합니다. (코드는 다음 페이지 참고)



cbRelay0 을 클릭할 때 릴레이를 토글하도록 Click 이벤트핸들러를 추가하겠습니다. 그림처럼, 디자이너에서 relay0 체크박스를 선택후, 우측 [속성]에서 번개표시 [이벤트] 아이콘을 선택하고, [Click]을 더블클릭하면, 앞의 Form1\_Load 이벤트핸들러에서처럼 cbRelay0\_Click 이벤트핸들러 코드의 빠대가 자동으로 생성됩니다.

마찬가지 방식으로 다음페이지의 소스코드를 참고하여 Form1\_FormClosing 이벤트핸들러도 추가하십시오.

앞에서 Form1\_Load 이벤트핸들러를 작성할 때 처럼 디자이너에서 콤포넌트를 더블클릭하면 대표 이벤트핸들러가 자동으로 작성되지만, cbRelay0 처럼 [속성]을 이용하면 원하는 이벤트핸들러를 선택적으로 작성할 수 있습니다. 자세한 사용법은 관련도서를 참고하십시오.

관련도서 몇가지를 추천하면,

- 뇌를 자극하는 C#4.0 프로그래밍  
기초적 내용으로 쉽고 간략하게 설명되어 있다.
- 닷넷 프로그래밍 정복  
내용이 방대하고 충실해서 옆에 두고 참고하기에 좋다.
- 소설같은 C#  
책보다 제공되는 동영상 강좌가 압권이고, 짧은 시간에 듣기에 좋다.
- Head First C# (번역판)  
삽화가 많아 이해에 도움이 되지만, 약간은 산만한 느낌도 있다.

완성된 Form1.cs 코드는 아래와 같습니다.

```
using System;
using System.Windows.Forms;
using EventBus;

namespace dio_ex1
{
    public partial class Form1 : Form
    {
        Serial com = new Serial();
        CDIO dio = new CDIO(4);

        public Form1()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false;
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            com.Connect("COM4", 115200);
            if (!com.Connected)
            {
                MessageBox.Show("USB connection failed!");
                Close();
            }
        }

        dio.sw0.OnChanged += new LogicEventHandler(sw0_OnChanged);
    }

    void sw0_OnChanged(object sender, CLogicEventArgs args)
    {
        cbSwitch0.Checked = args.State;
    }

    private void cbRelay0_Click(object sender, EventArgs e)
    {
        dio.rly0.State = cbRelay0.Checked;
        com.Synchronize(dio);
    }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        com.Disconnect();
    }
}
```

위 코드를 하나씩 살펴보겠습니다.

- **using EventBus;**  
참조에 추가한 EventBus.dll 라이브러리를 사용하겠다는 선언입니다. 이 선언이 있어야 이벤트버스의 클래스를 사용할 수 있습니다.
- **Serial com = new Serial();**  
EventBus.dll에 선언된 Serial 클래스를 이용해서 모듈과 통신하기 위해서 com 인스턴스를 생성합니다.
- **CDIO dio = new CDIO(4);**  
DIO 모듈의 이미지를 생성합니다. 모듈의 이미지 클래스는 Class의 머리글자인 ‘C’로 시작합니다. 그리고, 4는 DIO 모듈의 ID(통신주소)입니다. 모듈에서도 ezconfig.exe를 사용해서 ID를 4로 설정해 주어야 합니다.

- `CheckForIllegalCrossThreadCalls = false;`  
메뉴의 “디버그::디버깅하지 않고 시작(Ctrl+F5)”으로 실행하면 문제되지 않지만, “디버그::디버깅 시작(F5)”으로 실행하면 “크로스 쓰레드”라는 예외가 발생합니다. 어떤 경우에도 예외가 발생하지 않게 하려고 넣은 코드입니다 (중요한 내용은 아니니 그냥 입력하시면 됩니다).
- `com.Connect("COM4", 115200)`  
시리얼통신을 연결합니다. 연결된 시리얼포트 이름은 컴퓨터마다 다를 수 있으므로 미리 확인하십시오. CP2102 를 사용하는 USB, UUM, UUB 또는 CoLinkEX 를 이용하여 컴퓨터에 연결한 경우, 아래처럼 연결된 시리얼포트를 자동검색할 수도 있습니다. 이경우 System.Management 참조가 추가되어야 합니다.

```
string FindPortName()
{
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher("select * from Win32_SerialPort");

    foreach (ManagementObject serial in searcher.Get())
    {
        if (serial["PNPDeviceID"].ToString().Contains("5142308698"))
        {
            return serial["DeviceID"].ToString();
        }
    }
    return null;
}
```

- `dio.sw0.OnChanged += new LogicEventHandler(sw0_OnChanged);`
- ```
void sw0_OnChanged(object sender, CLogicEventArgs args) {
    cbSwitch0.Checked = args.State;
}
```
- 이벤트핸들러를 추가하는 핵심코드입니다. 이 코드를 이해하려면 C#의 대리자(델리게이트,Delegate)와 이벤트(Event)에 대하여 이해해야 합니다. 대부분은 C#문법에 대한 내용이므로 관련도서를 참고하십시오. 간략히 살펴보면, dio 모듈의 sw0 의 OnChanged 이벤트가 발생하면(스위치를 누르거나 놓으면 발생함), sw0\_OnChanged 이벤트핸들러(메서드)를 실행하도록 등록하는 것입니다. 즉, sw0 이 동작하면 sw0\_onChanged 이벤트핸들러가 실행됩니다. 이때, 바뀐 스위치 값은 args 인자를 통해서 전달됩니다. 결국, 체크박스의 체크상태가 모듈에 연결된 스위치 상태에 따라 바뀌게 됩니다.

메서드를 정의하고 등록하는 절차를 C#2.0 에서 추가된 익명(Anonymous)메서드나 C#3.0 에서 추가된 람다(Lambda)식을 이용하면 더욱 간략하게 기술할 수 있습니다.

```
// C#2.0 익명메서드를 사용한 이벤트핸들러
dio.sw0.OnChanged += new LogicEventHandler(
    delegate(object sender, CLogicEventArgs args) {
        cbSwitch0.Checked = args.State;
    }
);

// C#3.0 람다식을 사용한 이벤트핸들러
dio.sw0.OnChanged += (sender, args) => cbSwitch0.Checked=args.State;
```

- ```
private void cbRelay0_Click(object sender, EventArgs e) {
    dio.rly0.State = cbRelay0.Checked;
    com.Synchronize(dio);
}
```

cbRelay0(  relay 0 )를 클릭하면 실행되는 이벤트핸들러입니다. 체크박스의 체크상태에 따라 dio 이미지의 rly0의 상태를 변경하고, 변경된 출력이미지(dio)를 시리얼통신(com)을 통하여 모듈에게 전송해서(Synchronize) 실제 릴레이가 동작하도록 합니다.

\*주의\* 모듈의 출력을 나타내는 컴포넌트(릴레이 체크박스 처럼)를 사용하는 경우 CheckedChanged 또는 CheckStateChanged 처럼 값 자체의 변경을 검사하는 이벤트를 사용하는 것보다 Click 이벤트와 같은 실제 사용자의 조작을 검사하는 이벤트가 좋습니다. 값의 변경을 검사하는 이벤트는 프로그램이나 이벤트에 의해서 값이 변경되었을 때도 DioOut 이벤트를 발생하게 되어 부수효과(Side Effect, 릴레이가 깜박이는 등의 기대하지 않은 동작)를 일으킬 수 있으므로 주의하여 사용해야 합니다. 그러나 조작여부를 검사하면 그런 상황은 발생하지 않습니다.
- ```
private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
    com.Disconnect();
}
```

폼이 닫힐때 발생하는 FormClosing 이벤트의 핸들러입니다. 폼이 닫힐때 시리얼 통신의 연결을 해제하고 있습니다. 연결을 해제하지 않으면 USB 모듈은 PC 가 연결되어 있다고 생각하고 계속 이벤트를 전송하므로 약간의 성능저하가 발생합니다.



위의 C# 코드가 입문자에게는 복잡해 보일 수도 있지만, 기존에 프로그램을 작성해 보신분은 위의 코드가 얼마나 간결한지 쉽게 공감하실 것으로 생각됩니다.

좀더 복잡한 예제를 살펴보려면, 메뉴에서 파일/열기/프로젝트/솔루션을 선택하고, 다운로드한 eventbus\_net\_0.60 폴더에 있는 sample 폴더에 있는 sample.sln 솔루션파일을 열면 다양한 프로젝트를 볼 수 있습니다. 특히 sample 폴더의 각 프로젝트는 매우 중요한 내용을 담고 있으므로 꼭 살펴보시기 바랍니다.

**\*\*주의\*\*** 각 솔루션에는 여러 프로젝트가 있으며 실행할 프로젝트를 선택하려면, 솔루션팀색기에서 실행하려는 프로젝트에서 마우스우측버튼을 누른 후 “시작프로젝트로 설정(A)”을 선택하면, 프로젝트가 굵은 글씨로 바뀌면서 시작프로젝트로 설정됩니다.

이제까지 컨트롤러를 프로그램하는 몇가지 방법에 대한 간단한 예제를 살펴보았습니다. 이벤트버스의 기능은 이보다 훨씬 다양하므로 좀더 고급프로그램을 작성하려면 이어지는 다음장을 꼭 숙지해 주십시오.

## 2.4. 펌웨어로 모듈 제어하기

컴퓨터에 연결하지 않고 동작하기 위해서는 모듈의 마이컴에 펌웨어를 다운로드해야 합니다. 펌웨어 프로그램을 작성하기 위하여 우선 펌웨어 개발환경을 설치해야 합니다. 이지컨트롤러는 CooCox.org 그룹의 이클립스 기반 오픈소스 무료 개발환경인 CoIDE를 사용합니다.

### 2.4.1. 개발환경 설치하기

펌웨어를 개발하기 위해서는 아래의 몇가지 프로그램을 설치해야 합니다.

- CoIDE  
통합개발환경, 에디터, 디버거, 컴파일러에 대한 유저인터페이스
- GCC 컴파일러  
오픈소스 C-컴파일러
- CoLinkEx USB 드라이버  
빌드된 프로그램을 마이컴에 다운로드 및 디버깅 하기위한 툴인 CoLinkEx 의 USB 통신드라이버 프로그램
- CP2102 드라이버  
모듈을 컴퓨터에 연결할 때 사용되는 UUM, USB, CoLinkEx 의 USB 통신드라이버 프로그램

자세한 설치과정은 “[이벤트버스 프로그래밍 실습](#)” 메뉴얼을 참고해 주십시오.

### 2.4.2. 펌웨어 프로그래밍

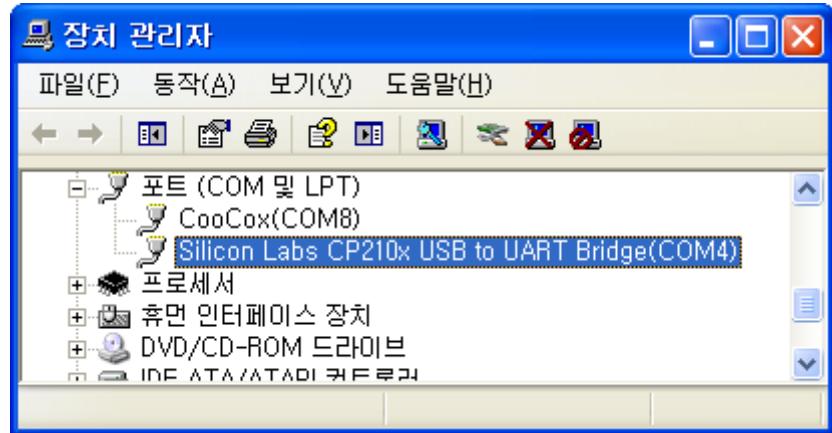
모든 설치를 마쳤으면, 이번 절에서는 PC 와 연결하지 않고, DIO 모듈만으로 릴레이를 동작시키는 펌웨어 프로그램을 작성해 보겠습니다.

실습을 위한 하드웨어 시스템을 구성하겠습니다. 아래 그림은 우리가 프로그램에 사용할 모듈의 구성입니다. PSM, DIO 모듈이 버스케이블로 연결되어 있고, PCB 서포터로 체결되었습니다. 스위치가 DIO 의 SW0 과 VDC 사이에 연결되어 있으며, ColinkEx 가 DIO 의 확장케이블에 연결되어 있습니다. 앞장에서 사용한 시스템과 동일합니다.



검은색 USB 케이블은 SWD(Serial Wired Debug) 기능으로 모듈의 펌웨어를 수정하거나 디버깅할 때 필요합니다. 컴퓨터와 연결하여 응용프로그램을 작성할 때는 연결하지 않아도 됩니다.

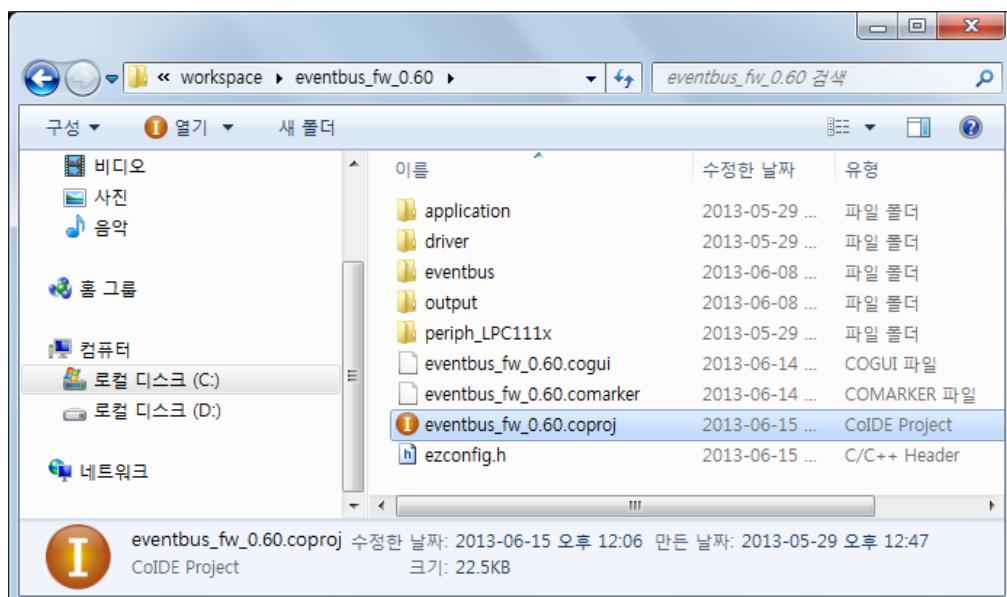
흰색 USB 케이블은 이번 예제처럼 컴퓨터와 연결하여 응용프로그램을 작성할 때 필요합니다, 컴퓨터에 가상 COM 포트를 생성해 줍니다. 앞에서 설명했듯이 반드시 드라이버를 설치하셔야 합니다. 정상적으로 연결되면, 내컴퓨터/속성에서 하드웨어 탭에 있는 장치관리자에서 그림처럼 연결된 포트를 확인할 수 있습니다.



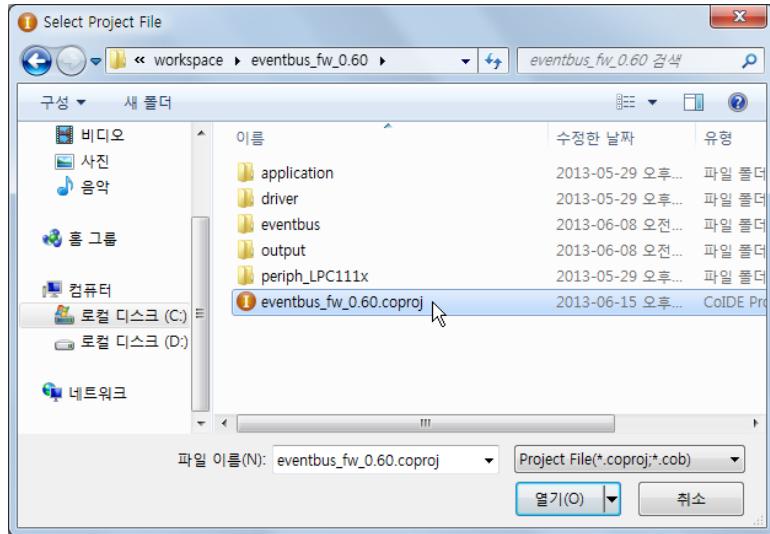
USB, U2M, UUM(UUB 대체품)을 연결해도 마찬가지로 연결된 포트를 확인할 수 있습니다. 그림에서는 COM4에 연결되었지만 컴퓨터마다 차이가 있으므로 반드시 확인해 주십시오.

## 2.5. eventbus OS 의 다양한 사용법

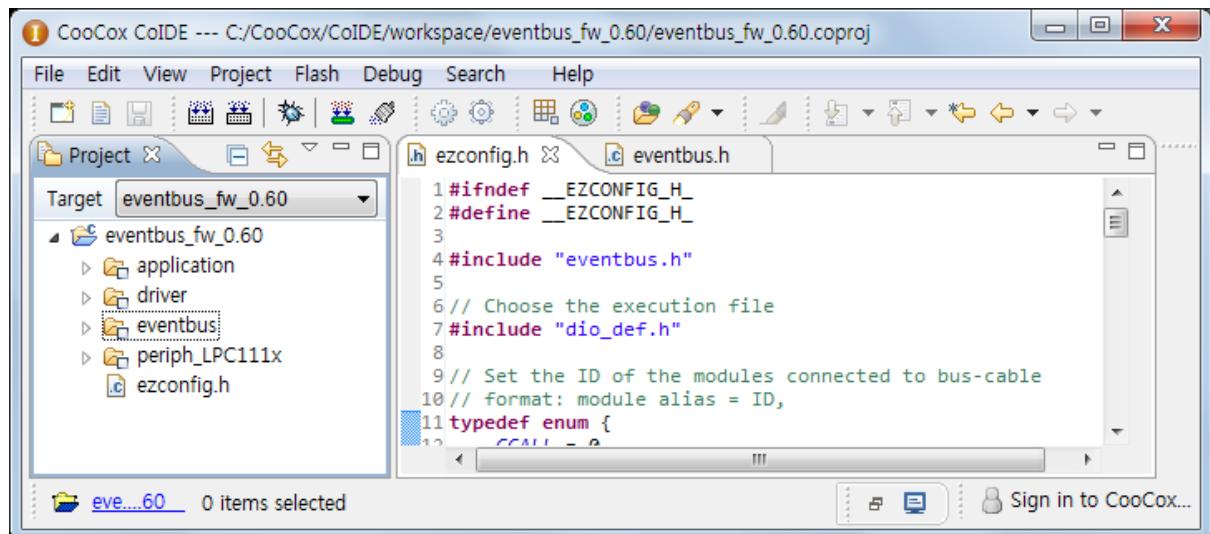
이지컨트롤러는 대상을 제어시스템으로 국한시키고, 이벤트 방식을 도입하여 문제를 단순화 시켰습니다. 이제부터 위의 프로그램을 이지컨트롤러에서 제공하는 이벤트버스를 사용해서 작성해 보겠습니다. 빠른 내용파악을 원하면, 네이버카페 ([cafe.naver.com/easycontrol](http://cafe.naver.com/easycontrol))에서 제공되는 동영상 강좌를 참고해 주십시오.



네이버카페([cafe.naver.com/easycontrol](http://cafe.naver.com/easycontrol))에서 `eventbus_fw_0.60.zip` 을 다운로드 후 `C:\CooCox\CoIDE\workspace\eventbus_fw_0.60`에 그림처럼 압축파일을 풀고 저장합니다.



CoIDE 를 실행하고 Project 메뉴에서 Open Project 를 선택하고 `eventbus_fw_0.60.copproj` 프로젝트파일을 엽니다.



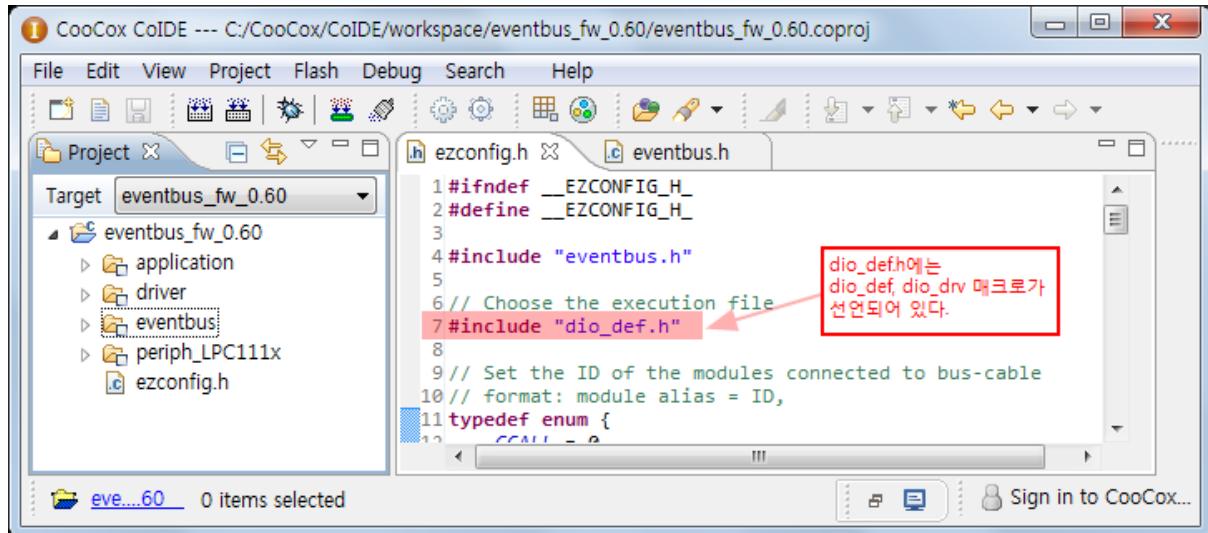
여기에는 개발에 필요한 모든 것이 미리 설정되어 있으므로 별도로 프로젝트를 생성하는 등의 작업이 필요치 않습니다.

왼쪽에 있는 Project 창의 주요 항목을 살펴보면,

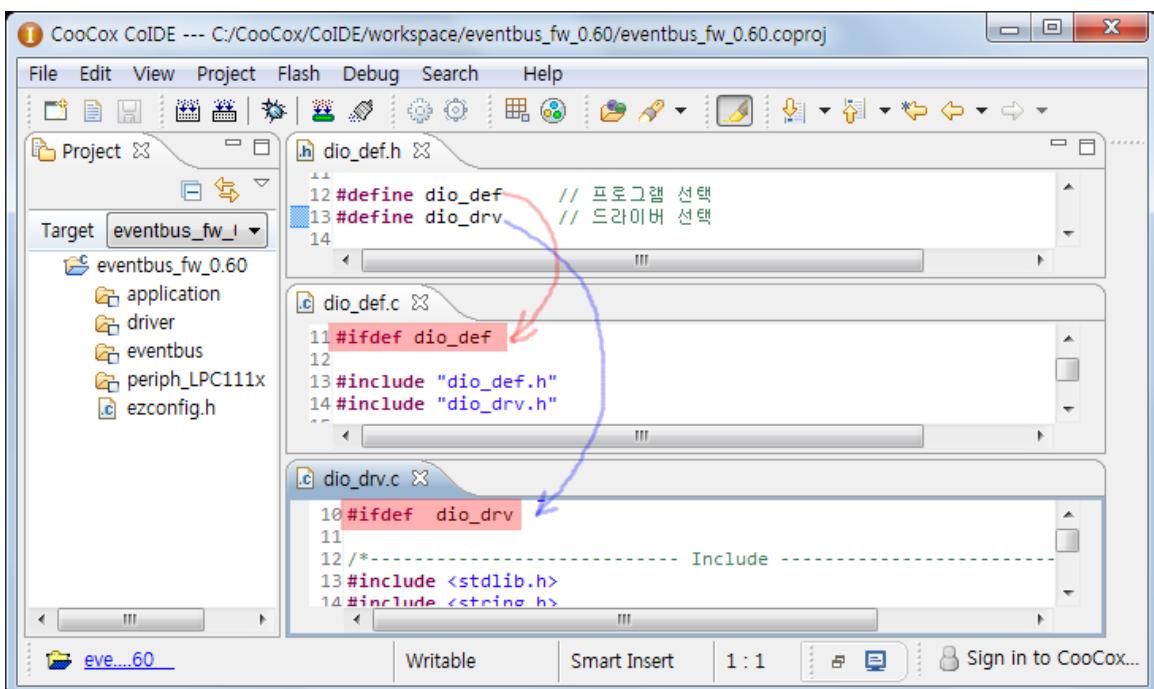
- application : eventbus\_fw 를 이용하여 작성된 예제 프로그램
  - default : 출고시 모듈에 내장된 기본 프로그램의 소스파일들
  - exam : eventbus\_fw 의 주요기능을 보여주는 예제 프로그램
- driver : 각 모듈의 하드웨어 드라이버 프로그램
- eventbus: eventbus\_fw 의 OS 헤더파일
- periph\_LPC111x : LPC111x 하드웨어 라이브러리

■ ezconfig.h: 프로젝트의 최상위 설정파일(빌드할 펌웨어를 선택할 수 있음)

default 폴더에는 많은 프로그램이 있습니다. 이중에서 dio\_def.c, dio\_def.h 는 DIO 모듈이 출고될 때 기본으로 모듈에 다운로드된 프로그램의 소스입니다. 앞의 예제처럼 DIO 의 릴레이를 동작시키는 프로그램을 스캔방식과 이벤트방식으로 이 프로그램에 추가하는 과정을 통하여 두 방식의 동작을 이해하도록 합니다.

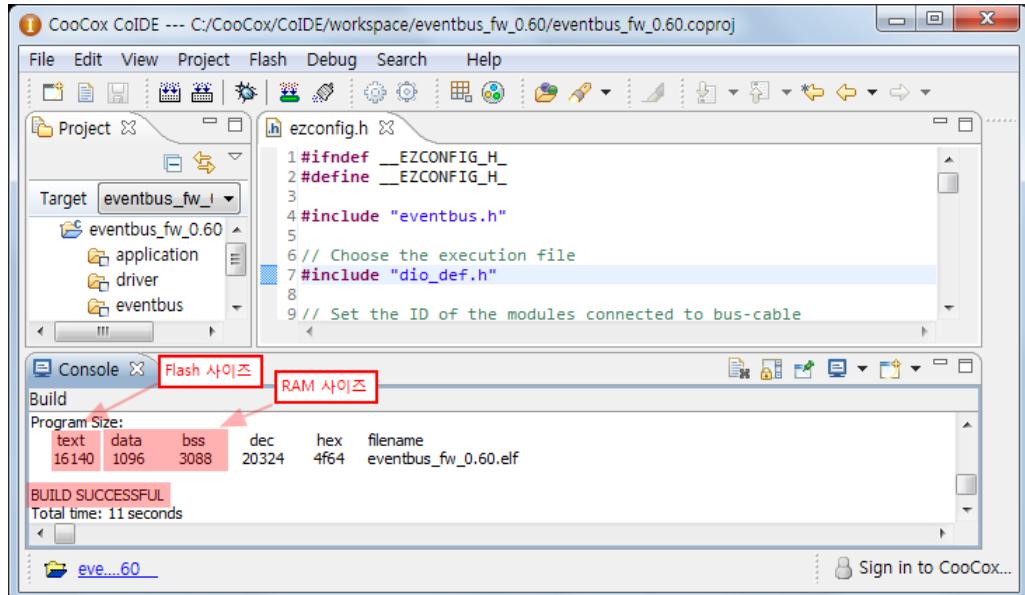


application 폴더에는 많은 응용프로그램 파일이, driver 폴더에는 많은 드라이버 프로그램 파일이 있습니다. 이중에서 우리가 프로젝트를 빌드할 때 사용할 **응용프로그램**과 **드라이버**를 선택해야 합니다. 프로젝트창에서 ezconfig.h 파일을 열고, 7 번줄에서 빌드할 펌웨어를 선택하면 됩니다.

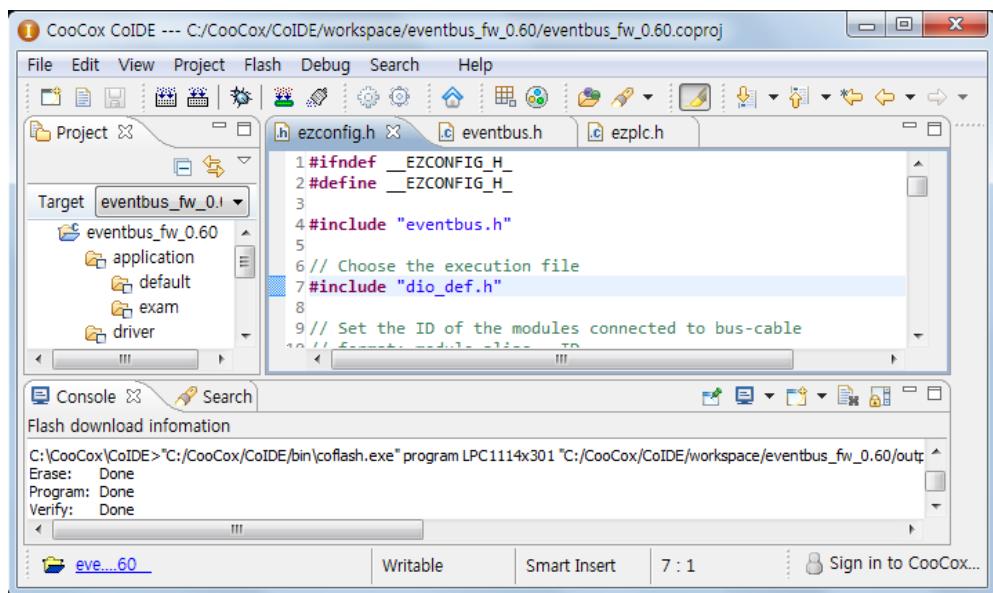


위 그림처럼 dio\_def.h 에는 두개의 매크로 dio\_def, dio\_drv 가 선언되어 있으므로, 이 매크로가 선언되었을 때만 빌드되도록 프로그램되어 있는 dio\_def.c 와 dio\_drv.c 가 빌드에 포함됩니다. (선택되지 않은 파일을 열어보면 프로그램이 회색으로 표시되어 선택되지 않은 것을 표시합니다. 매크로명 dio\_def, dio\_drv 는 전체프로젝트에서 유일한 이름이어야 하므로 파일명 dio\_def.c, dio\_drv.c 와 동일하게 사용하는 것이 중복방지에 좋습니다.)

메뉴에서 [Project] - [Build]를 선택하거나, 단축키 F7 을 누르거나 또는 툴바에서  를 선택하여 프로젝트를 빌드합니다.



그림처럼 “BUILD SUCCESSFUL” 메시지가 나오면 성공적으로 빌드 된 것입니다. 마이컴의 플래시 메모리에 저장될 사이즈와 사용되는 램 메모리의 사이즈도 표시됩니다.



그림처럼 툴바에서 “Download Code To Flash” 아이콘을 눌러 프로그램을 마이컴에 다운로드하면 펌웨어 프로그래밍 작업이 완료됩니다.

### 2.5.1. 스캔(Scan) 방식으로 프로그램 하기

1 초마다 릴레이가 온오프를 반복하는 프로그램을 ezplc.h 라이브러리를 이용하여 래더프로그램(스캔방식)으로 작성합니다. default 폴더에서 dio\_def.c 를 더블클릭으로 열고, 아래와 같이 프로그램을 작성합니다. 새로 입력할 부분은 회색으로 음영처리된 부분입니다. 나머지 부분은 원래 있는 그대로입니다.

```

1. #include "ezconfig.h"
2. #include "ezplc.h"
3. #ifdef dio_def

4. #include "dio_def.h"
5. #include "dio_drv.h"

6. DIO* this;

7. LEdge edge;

8. #define RLY0 (this->out->rly0)

9. int main()
10. {
11.     EventBus();

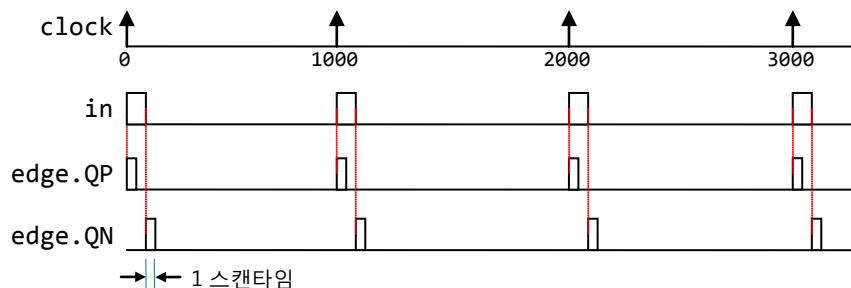
12.     EventProgram;

13.     ScanProgram;
14.     ELogic in = (clock % 1000 == 0) ? ON : OFF;
15.     Edge(&edge, in);
16.     if (edge.QP) RLY0 = !RLY0;

17.     End;
18. }
19. #endif

```

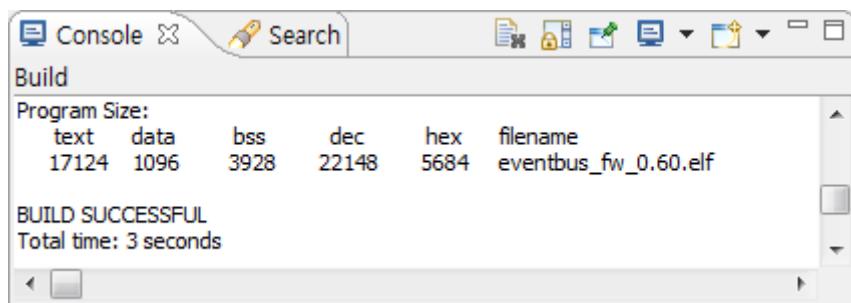
6:this 는 현재 프로그램중인 모듈입니다 7:에지검출을 위한 구조체변수를 정의합니다(PLC 프로그래밍 참고). 11:EventBus()는 이벤트버스를 초기화합니다. 13:스캔방식에서는 프로그램을 ScanProgram; 아래에 작성해야 합니다. 14:이벤트버스에서 정의된 시스템 전역 시간변수인 clock 은 전원투입후 계속 증가하는 밀리초(ms) 단위의 시계입니다. (clock % 1000)의 계산결과는 항상 0~999 이므로 변수 in 은 clock 0이 0, 1000, 2000...에서 ON 이 됩니다.



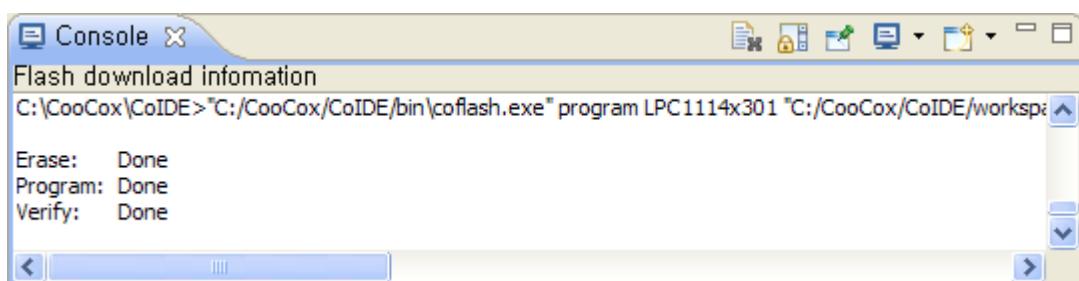
Edge(LEdge 타입 변수의 주소, 입력) 함수는 입력에서 상승/하강에지를 검출하여 LEdge 타입의 변수에 저장합니다(상승에지는 QP, 하강에지는 QN에 저장). 이 경우는 변수 in이 상승에지 일 때와 하강에지 일 때를 그림처럼 QP, QN에 저장하게 됩니다.

그런 후에, 16:에서 QP가 ON이 될 때 this->out->rly0을 토글(반전)하게 되므로 전원투입->0ms, 999ms->1000ms, 1999ms->2000ms...처럼 변화하는 순간마다 1초 단위로 온/오프를 반복하게 되는 것입니다.

일반적으로 펌웨어를 프로그램하려면 수백페이지의 마이컴 데이터시트를 이해해야 하고, 하드웨어도 이해하고 있어야 합니다. eventbus\_fw 를 이용하는 것이 얼마나 편리한지를 이해할 수 있습니다. 무엇보다도 하드웨어, 마이컴 등에 대한 이해가 없이도 프로그램이 가능하다는 것은 매우 큰 장점입니다.



[빌드하기] 메뉴의 Project/Build 또는 F7 키로 프로그램을 빌드하면 그림처럼 Console 창에 결과가 표시됩니다. “BUILD SUCCESSFUL” 메시지가 표시되면 성공적으로 빌드된 것입니다. text 는 사용된 Flash 메모리 용량입니다. LPC1114 는 32KByte 의 Flash 를 사용할 수 있으므로 text 가 32768 보다 작아야 합니다. 또한 SRAM 용량은 8KByte 이므로 data+bss 가 8192 보다 작아야 합니다. 일반적인 프로그램은 이 용량을 초과하지 않지만, 만일 프로그램이 매우 큰 경우에는 프로그램을 다른 모듈에 나누어서 작성해도 됩니다. 예를 들어, DIO 와 DOM 모듈을 사용한다면 두 모듈에 나누어 작성할 수 있으므로 문제되지 않습니다.



[다운로드하기] 메뉴의 Flash/Program Download 또는 으로 프로그램을 컨트롤러에 다운로드 하십시오. Console 창에 그림처럼 표시되면 정상적으로 다운로드된 것입니다. 만일 ColinkEx 의 USB Driver 가 설치되지 않았거나, USB 케이블에 이상이 있으면, “ColinkEx not found, make sure adapter is connected.” 모듈에 전원이 공급되지 않거나 이상이 있으면 “Target Chip not found”라는 에러메시지가 출력됩니다.

[결과확인] 위 과정을 정상적으로 마쳤다면, 릴레이가 ON/OFF 하는 소리를 들을 수 있습니다.

## 2.5.2. 이벤트(Event) 방식으로 프로그램 하기

동일한 프로그램을 이벤트방식으로 작성해 보겠습니다.

```

1: #include "ezconfig.h"
2: #include "ezplc.h"

3: #ifdef dio_def
4: #include "dio_def.h"
5: #include "dio_drv.h"

6: DIO* this;

7: int main()
8: {
9:     EventBus();

10:    EventProgram;
11:    IF (this, CyclicTimeout) {
12:        if (pe->ms % 1000 == 0) {
13:            Toggle(this->out, rly0);
14:        }
15:    }

16:    ScanProgram;
17:    End;
18: }
19: #endif

```

이벤트 프로그램이란, 어떤 이벤트(사건)가 발생하면 그 이벤트에 맞는 처리내용을 **IF (이미지, 이벤트태그) { 처리내용 } }** 형식의 이벤트핸들러에 작성하는 것입니다. 이벤트핸들러는 10:EventProgram; 아래에 작성해야 합니다. 11:this 모듈(이 경우 DIO)에서 발생한 CyclicTimeout 이벤트를 처리하는 이벤트핸들러입니다. CyclicTimeOut 이벤트는 (ezconfig.exe 로) 설정한 시간간격(Tp)을 주기로 반복적으로 발생하는 이벤트입니다. 12:이벤트핸들러에는 발생한 이벤트에 대한 부가정보가 pe 를 통해서 전달됩니다. CyclicTimeout 이벤트의 경우 이벤트가 발생한 시간이 변수 ms 에 전달되므로 12:pe->ms 처럼 읽을 수 있습니다. 이 시간을 1000 으로 나눈 나머지가 0 이 되는 것은 1000ms, 2000ms, 3000ms...입니다. 즉, 1 초마다 반복적으로 13:이 실행됩니다. 13:에서는 rly0 을 반전시키고 있습니다.

아직 익숙하지 않아서 어렵게 느껴질 수 있습니다. 그러나 윈도우의 이벤트 프로그래밍을 해보신 분은 매우 유사하다고 느꼈을 것입니다.

이렇게 이벤트버스를 이용해서 이벤트방식으로 펌웨어를 개발하는 이점을 살펴보면,

- 첫째, 하드웨어에 관련된 함수나 주소, 매크로 등이 보이지 않습니다. 실제로 이런 함수가 수백개가 넘습니다. 모두 기억하기도 힘이 듭니다.  
SYSCON\_AHBPeriphClockCmd(), GPIO\_SetDir(), GPIO\_SetBits(),  
GPIO\_ResetBits(), SYSCON\_AHBPeriph\_GPIO, PORT1, GPIO\_Pin\_8 ...

- 둘째, 임베디드 프로그램에서 매우 중요한 시간의 처리가 매우 쉽습니다. “C 라이브러리로 만드는 프로젝트”에서는 간단히 `delay()` 함수를 사용했지만 지연시간이 얼마가 될지는 알 수 없습니다. 정확히 시간을 알려면 타이머를 사용해야 하지만 이 경우에도 부가적인 문제들이 있어 코드는 현재보다 매우 복잡해 집니다. 그러나, 이벤트버스를 사용한 예에서는 시계변수인 `clock` 을 참조하면 정확한 시간간격으로 릴레이를 동작시킬 수 있습니다. 뒤에서 설명할 V0.40 부터 추가된 `DelayedEvent` 를 사용하면 “10 초후에 1 초동안 릴레이를 ON 시켜라”와 같은 복잡한 시간관리도 쉽게 구현할 수 있습니다.
- 셋째, 각각의 이벤트 핸들러는 기능적으로 독립적입니다. 그러므로 구현할 기능이 증가할 경우에도 프로그램의 복잡도는 크게 증가하지 않습니다. 반면 이벤트버스를 사용하지 않을 경우에는 기능이 증가하면 기능간의 상호영향(Side Effect)으로 코드의 복잡도는 기하급수적으로 증가하게 됩니다. 물론 이것은 RTOS 를 사용해도 어느정도는 개선할 수 있지만 이벤트핸들러 만큼 작은 단위로 분할하는 것은 어려우며, RTOS 를 새로 배우기도 쉽지않다는 문제점도 있습니다.
- 넷째, 하드웨어 관련 코드가 줄어들고, 기능간의 상호영향(부수효과)이 제거되므로 유지관리가 쉬워집니다. 향후 기능추가나 수정이 필요할 때 마이컴의 메뉴얼을 다시 펼쳐보지 않아도 쉽게 기능을 구현할 수 있습니다.
- 다섯째, 생산성이 매우 높아집니다. 자체 테스트에 따르면 펌웨어 만으로 개발한 경우에 비하여 수배~수십배 이상의 생산성을 얻을 수 있었습니다. 그러므로 프로토타이핑으로 사용하면 매우 적합합니다.
- 여섯째, 이벤트마다 별도의 이벤트핸들러를 구현하게 되므로 자연스럽게 문서화가 됩니다. 그래서, 개발 담당자가 바뀌었을 때에도 쉽게 업무를 인수인계할 수 있어 제품개발의 연속성을 유지할 수 있습니다.
- 여덟번째, PLC 래더프로그램에 익숙한 분들도 쉽게 사용할 수 있도록 스캔프로그램에서 사용할 수 있는 `ezplc.h` 라이브러리를 제공하고 있습니다. 이 라이브러리를 이용하면 래더프로그램과 유사하게 프로그램할 수 있습니다.
- 일곱번째, 몇가지 사항만 유의하여 개발하면, 개발된 모듈들을 저렴한 하나의 양산보드로 제작할 수 있으며, 프로그램은 변경없이 그대로 사용할 수 있으므로 개발속도가 매우 빨라지고 리스크가 제거됩니다. (카페 공지사항 참조, 별도문의)

### 2.5.3. 스위치로 릴레이 온오프 시키기

이번에는 조금 더 복잡한 예제를 작성해 봅니다. DIO 모듈에 연결된 스위치 상태를 읽고, 스위치 상태에 따라 릴레이를 온오프 시켜 봅니다.

```

1: int main()
2: {
3:     EventBus();
4:     EventProgram;
5:     IF (this, DioIn) {
6:         this->out->rly0 = pe->sw0;
7:     }
8:     ScanProgram;
9:     Eval_DioIn(FALSE);
10:    End;
11: }
```

위의 프로그램은 dio\_def.c에서 main() 함수만 나타냈습니다. 스캔프로그램과 이벤트핸들러가 모두 포함되어 있는데, 이런 경우는 이벤트핸들러가 우선적으로 실행됩니다. 즉, **이벤트가 있으면 이벤트핸들러를 우선적으로 실행하고, 이벤트가 없을 때는 스캔프로그램을 반복적으로 실행하게 됩니다.** 이벤트는 빈번하게 발생하지 않는 것이 일반적이므로 평상시에는 스캔프로그램이 실행되다가 이벤트가 발생하면 이벤트핸들러를 실행한다고 생각하면 됩니다.

9:Eval\_DioIn() 함수는 드라이버(dio\_drv.c)에서 선언되어 있는데, DIO 모듈의 입력(스위치)을 검사하고 입력에 변동이 발생하면 DioIn 이벤트를 발생시킵니다. 즉, 스위치를 동작하면 DioIn 이벤트가 이벤트큐에 삽입됩니다. 이벤트가 없으면 스캔프로그램이 반복실행되므로, 평소에는 Eval\_DioIn(FALSE);가 매우 빠르게 반복되고, 스위치의 변화를 즉시 감지합니다. (채터링이 있는 스위치의 경우 여러 번 누른 것으로 인식될 수 있으므로 주의가 필요합니다. 이경우 채터링을 제거하기 위한 디바운스(debounce)프로그램이 필요합니다.)

Eval\_DioIn(FALSE)에서 인자 FALSE는 입력에 변동이 있을 경우에만 이벤트를 발생하도록 합니다. TRUE 이면 스위치상태에 변화가 없어도 무조건 이벤트를 발생합니다.(스캔프로그램에서 사용할 때는 TRUE로 하는 것은 좋지 않습니다. 너무 많은 이벤트는 통신을 마비시킵니다)

5:DioIn 이벤트를 처리하는 이벤트핸들러입니다. 9:에서 발생한 DioIn 이벤트를 처리합니다. DioIn 이벤트핸들러에 전달되는 이벤트(pe)의 변수형은 DioInEvent 구조체(태그명 DioIn 에 Event 를 붙인다. eventbus.h 참고)입니다. 6:pe 에는 이벤트가 발생한 시점의 입출력데이터가 저장되므로, pe->sw0 은 스위치 sw0 의 최근 상태가 됩니다(반면, 이미지에 저장된 this->in->sw0 은 이벤트가 발생하기 이전의 상태입니다). 스위치 상태를 출력 rly0 에 저장하였으므로, 스위치에 따라 릴레이의 상태가 변하게 됩니다.

이제 F7 키를 눌러 빌드하고, 메뉴 [Flash] - [Program Download]로 프로그램을 DIO에 다운로드 하십시오. 동작을 확인하기 위하여 스위치를 5 번핀(SW0)과 11 번핀(VDC) 사이에

연결후 ON/OFF 시키면 릴레이 동작 소리를 들을 수 있습니다. 스위치 또는 센서를 연결하는 방법은 “6.5 DIM, Digital Input Module”을 참고하십시오.

스위치에 의해서 릴레이를 온/오프 시킬 수 있는 제어시스템이 완성되었습니다. 어려웠습니까? 이벤트의 동작방식을 완전히 이해하지 못하면 어려울 수 있지만, 하드웨어에 대한 내용이 코드에 없으며, 단순히 대입문과 함수호출문 밖에 없었습니다. 문법이 매우 단순하므로 조금만 연습하면 금방 익숙해 집니다.

### 3. eventbus\_fw

---

C#에서 eventbus\_net에서 제공하는 EventBus.dll을 이용하여 이벤트스타일로 프로그램하는 것이 얼마나 편리한지 느낄 수 있었습니다. 만일 이벤트스타일이 아니라면 프로그램은 언제 입력이 들어올지 몰라서 입력을 검사하면서 무한대기상태에 있어야 할 것입니다.

그러나, eventbus\_fw를 이용하면 마치 C#에서 이벤트핸들러 스타일로 프로그램을 작성할 때처럼 펌웨어를 이벤트 방식으로 프로그램할 수 있으므로 많은 장점들이 있습니다.

- 확장성 : 프로그램의 복잡도 증가없이 모듈 연결만으로 기능/성능을 확장할 수 있다.
- 문서화 : 이벤트 단위로 프로그램이 분리되므로 자연스럽게 문서화 된다.

이 장에서는 eventbus\_fw에서 기본적으로 제공되는 이벤트만으로 복잡한 프로그램을 개발할 때 직면하는 어려움을 해결하는데 도움이 되는 추가기능에 대하여 살펴보겠습니다.

#### 3.1. 사용자 이벤트

---

eventbus\_fw 프레임워크에서 기본적으로 발생되고 처리되는 이벤트(CyclicTimeout, XxxIn, XxxOut 등등)를 이용하면 쉽게 프로그램을 작성할 수 있습니다. 그러나, 경우에 따라서는 복잡한 기능을 좀더 효율적이고 쉽게 프로그래밍하려면 사용자가 필요한 이벤트를 직접 만들어 사용하는 것이 더 효과적인 경우가 많습니다.

```
1. IF (this, DioIn) {
2.   if (Pressed(this->in,sw0)) {
3.     UserDefinedEvent* ude = CreateEvent(UserDefined); // 사용자이벤트
4.     ude->ms = clock;
5.     SendEvent(ude);
6.   }
7. }
```

1:DioIn 이벤트는 “eventbus.h”에 태그와 이벤트의 구조체를 미리 선언해 두었으므로, 별도의 선언을 하지 않고 사용할 수 있었지만, 사용자가 직접 만드는 이벤트(위에서 UserDefined, UserDefinedEvent 처럼)는 **이벤트태그**와 **이벤트구조체**를 사용자가 직접 “ezconfig.h”에 선언해야 합니다. (ezconfig.h에 이미 선언된 UserDefined, EdgeChanged, SetVref, SetRelay 이벤트는 예제 프로그램에서 사용하기 위한 것이므로 이벤트버스의 동작에 반드시 필요한 것은 아닙니다. exam 폴더의 관련 예제를 살펴보십시오. 삭제하여도 문제되지 않습니다).

**이벤트 = 태그 + 구조체**

**예) UserDefined 이벤트 = UserDefined(태그) + UserDefinedEvent(구조체)**

사용자가 자신만의 새로운 이벤트를 만들고 사용하려면 태그와 구조체 두 가지를 선언해야 합니다. 이때 구조체는 반드시 태그명 뒤에 “Event”를 붙여서 사용해야 합니다.

```

16 // 사용자이벤트 태그
17 typedef enum {
18     UserDefined = StartOfUserEventTag, // used in application
19     EdgeChanged,
20     SetVref,
21     SetRelay,
22 } USER_EVENT_TAG;
23

```

사용자의 태그명은 그림처럼 ezconfig.h 에 있는 USER\_EVENT\_TAG 열거형에 선언하며, StartOfUserEventTag 부터 시작해야 합니다.

```

24 //=====
25 // 사용자이벤트
26 #pragma pack(push, 1)
27 typedef TimeEvent UserDefinedEvent; // UserDefined 이벤트
28
29 typedef struct {
30     EventHead e;
31     uint8 edge;
32 } EdgeChangedEvent;
33
34 typedef struct { // SetVref 이벤트
35     EventHead e;
36     float vref;
37     float hyst;
38 } SetVrefEvent;
39
40 typedef Event SetRelayEvent; // SetRelay 이벤트용
41 #pragma pack(pop)
42

```

29~32: 구조체의 선언은 그림처럼 26:과 41:의 `#pragma` 사이에 와야 하며, 30:구조체의 첫 항목은 `EventHead e;`로 시작해야 하고, 31:이어서 이벤트(pe)로 전달하고자 하는 데이터가 옵니다. 32:구조체의 이름은 “태그명 + Event” (태그명에 “Event”를 붙여서)로 해야 합니다. 40:만일, 이벤트로 전달할 데이터가 특별히 없다면, 기본제공되는 `Event` 구조체를 재정의해서 사용해도 되고, 27:동일한 이벤트가 이미 선언되어 있으면 그것을 재정의해도 됩니다. (다시 한번 정리하면 아래와 같습니다)

- 전달할 데이터가 특별히 없다면, 기본형인 `Event` 구조체를 재정의해서 사용하면 된다.

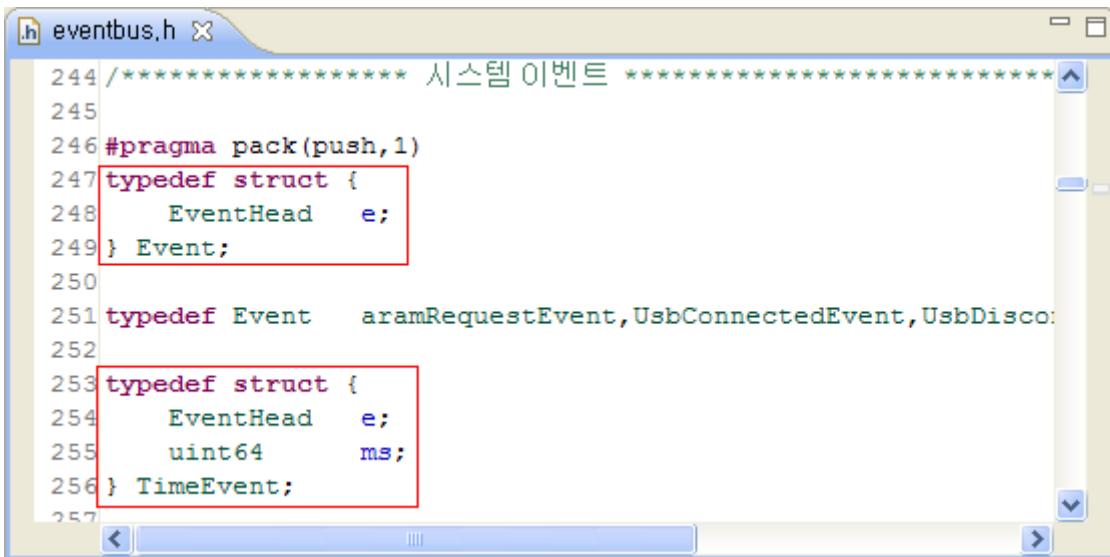
```
typedef Event UserDefinedEvent; // 이벤트는 태그명+”Event”
```

- 다른 이벤트와 같다면 그것을 재정의해서 사용해도 된다.

```
typedef TimeEvent UserDefinedEvent; // 이벤트는 태그명+”Event”
```

- 독자적인 이벤트 구조체를 선언하려면,

```
typedef struct {
    EventHead e;           // 첫항목은 EventHead
    float      vref;        // 이벤트로 전달할 데이터
} UserDefinedEvent;      // 이벤트는 태그명+”Event”
```



사실, Event 와 TimeEvent 는 그림처럼 eventbus.h 에 미리 선언되어 있습니다. Event 는 전달할 데이터가 없으므로 필수항목인 EventHead 만 가지는 텅빈 이벤트이며, TimeEvent 는 밀리초 단위의 시간값을 전송하기 위한 이벤트입니다.

이제 선언된 이벤트를 생성하고 이벤트큐에 넣는 과정을 살펴봅니다.

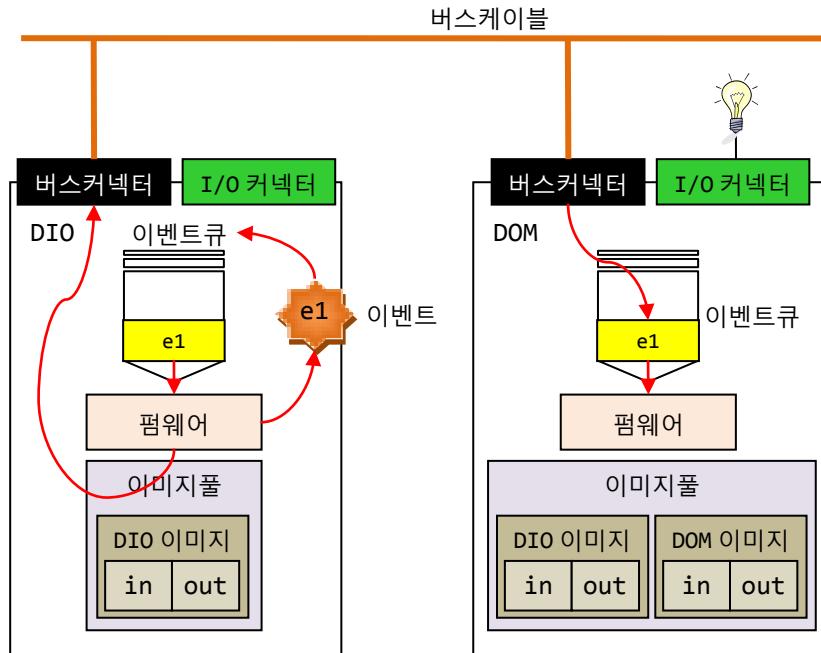
(application\exam\create\_event.c 참고)

```
IF (this, DioIn) {
    if (Pressed(this->in, sw0)) {                                // sw0이 눌러지면
        UserDefinedEvent* ude = CreateEvent(UserDefined); // 사용자이벤트 생성
        ude->ms = clock;                                     // 이벤트데이터 설정
        SendEvent(ude);                                       // 이벤트 발생(큐에 저장)
    }
}
```

ezconfig.h 에 보면 아래와 같이 UserDefinedEvent 가 선언되어 있습니다.

```
typedef TimeEvent UserDefinedEvent;          // UserDefined 이벤트
```

UserDefinedEvent 는 TimeEvent 를 재정의 했으며, TimeEvent 는 eventbus.h 에서 시간을 나타내는 ms 변수를 가지도록 선언되어 있습니다. 생성된 이벤트에 현재시간을 저장하는 전역 시계변수인 clock 의 시간을 저장하고, SendEvent(ude); 함수로 이벤트를 발생시킵니다(이벤트큐에 삽입).



펌웨어에서 만들어지고, `SendEvent()` 함수로 이벤트큐에 삽입된, `UserDefinedEvent`(그림의 `e1`)는 그림처럼, 먼저 펌웨어의 이벤트핸들러

```
IF (this, UserDefined) { ... }
```

에게 넘겨집니다. 그리고, 이미지에 저장할 필요는 없으므로(이미지에는 `DioIn`, `DioOut` 이벤트만 저장됨) I2C 통신으로 버스케이블에 연결된 다른 모듈들에게 전송되고 DIO를 참조하는 모듈(그림에서는 DOM)의 이벤트큐에 들어갑니다(사실, 버스케이블에 연결된 모든 모듈이 이벤트를 받지만, 이미지풀에 DIO의 이미지를 생성한 모듈만 이벤트를 정상적으로 수신하고 이벤트큐에 수신한 이벤트를 저장할 수 있습니다). 그리고 DOM 모듈의 펌웨어에 구현된 이벤트핸들러가 있다면,

```
IF (dio, UserDefined) { ... }
```

에게 넘겨져 사용된 후 삭제됩니다(사용자 이벤트는 여기서도 DOM 이미지에 저장되지 않음).

DIO 모듈에 구현된 `UserDefined` 이벤트핸들러에서 이벤트에 첨부된 시간데이터 `pe->ms`를 이용할 수 있습니다.

```
IF (this, UserDefined) {
    printf("switch pressed at %d[ms]\r\n", (int)(pe->ms));
}
```

이것은 스위치가 눌러진 시간(`pe->ms`)을 컴퓨터의 하이퍼터미널에 표시(UART로 전송)합니다.

### 3.1.1. 예제 프로그램 : 입력접점의 에지검출

사용자 정의 이벤트(User Defined Event)의 사용을 보여주기 위한 간단한 예제를 작성해 봅니다. 스위치를 동작하면 DioIn 이벤트가 발생합니다. 그러면, DioIn 이벤트핸들러에서 스위치의 상태를 검사해서 EdgeChanged 사용자정의 이벤트를 발생하는 예제를 살펴봅니다.

```

1:      #include "ezconfig.h"
2:      #include "ezplc.h"
3:      #ifdef user_event1

4:      #include "user_event1.h"
5:      #include "dio_drv.h"

6:      DIO* this;

7:      int main()
8:      {
9:          EventBus();

10:         EventProgram;
11:         IF (this, DioIn) {
12:             if (this->in->sw0 != pe->sw0) { // if (Toggled(this->in, sw0)) 와 동일
13:                 EdgeChangedEvent* ece = CreateEvent(EdgeChanged);
14:                 ece->edge = pe->sw0 ? RISE : FALL;
15:                 SendEvent(ece);
16:             }
17:         }

18:         IF(this, EdgeChanged) {
19:             this->out->rly0 = (pe->edge == RISE) ? ON : OFF;
20:         }

21:         ScanProgram;
22:         Eval_DioIn(FALSE);           // 입력검사

23:         End;
24:     }

25: #endif

```

11:DioIn 이벤트핸들러에서 12:sw0에 변화가 있는지 검사하고 13:EdgeChanged 사용자이벤트를 만들어서 14:이벤트의 edge 변수에 sw0의 에지상태를 저장하고(12:에서 스위치가 조작되었음을 확인했으므로 스위치의 현재상태가 ON이라는 것은 상승에지를 의미한다) 15:이벤트를 발생합니다(이벤트큐에 삽입).

발생한 EdgeChanged 이벤트는 18:의 EdgeChanged 이벤트핸들러에서 처리됩니다. 19:이벤트의 edge 값에 따라 릴레이 rly0을 동작시킵니다.

## 3.2. 시간지연 이벤트

발생한 이벤트가 이벤트큐에 들어가면, 이벤트큐에 들어온 순서대로 이벤트핸들러에게 건네집니다. 그러나, 시간지연 이벤트인 `Delayed` 이벤트를 사용하면, 즉시 이벤트핸들러에 건네지지 않고 설정된 시간 이후에 건네지도록 지연시간을 설정할 수 있습니다.

이 기능을 사용하면 시간에 관련된 많은 기능을 간편하게 구현할 수 있습니다.

```
typedef struct {
    EventHead e;
    uint64 delayTime;
    Event* delayedEvent;
} DelayedEvent;
```

`Delayed` 이벤트의 구조체에는 이벤트핸들러에게 건네줄 시간을 설정하는 `delayTime`과 실제로 발생시킬 `delayedEvent` 가 있습니다. 즉, `delayTime` 만큼 시간이 지난 후에 `delayedEvent` 를 이벤트핸들러에게 건네줍니다. 시간지연 이벤트를 발생하기 위하여 `SendDelayedEvent` 함수를 사용합니다.

```
void SendDelayedEvent(uint32 delayTime, void* delayedEvent);
```

여기서, `delayTime`은 지연시킬 시간간격(ms)

`delayedEvent`는 지연시킬 이벤트

### 3.2.1. 온딜레이 타이머(OnDelay timer)

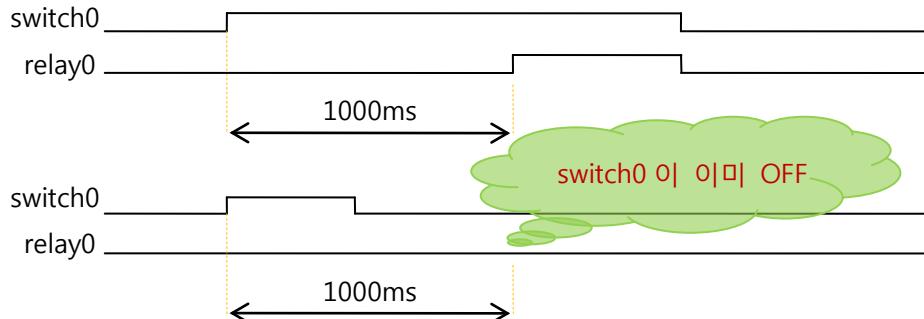
사용자이벤트와 시간지연 이벤트를 함께 사용해서 `OnDelay` 타이머를 구현한 예입니다. 스위치가 상승에지이면 1초간 지연후 릴레이를 ON시키고, 하강에지에서는 즉시 OFF시키는 것이 `OnDelay` 타이머입니다.

```
IF (this, DioIn) {
    if (Pressed(this->in, sw0)) // 상승에지에서는 시간지연이벤트를 발생하고
        SendDelayedEvent(1000, CreateEvent(SetRelay));
    else if (Released(this->in, sw0)) // 하강에지에서는 즉시 OFF시킨다
        this->out->rly0 = OFF;
}

IF (this, SetRelay) {
    if (this->in->sw0 == ON) // 스위치가 ON일때만 릴레이를 ON시킨다.
        this->out->rly0 = ON;
}
```

`DioIn` 이벤트핸들러에서는 `sw0` 의 상승에지에서 1초후 `SetRelay` 이벤트를 발생하는 지연이벤트를 발생하고, `sw0`의 하강에지에서는 릴레이를 즉시 OFF시킵니다.

`SetRelay` 이벤트핸들러는 스위치가 여전히 ON 상태를 유지하고 있으면 릴레이를 ON 시키지만, `sw0`이 이미 OFF 상태로 되었으면 릴레이를 ON 시키지 않도록 동작합니다.



결과적으로, 그림처럼 스위치를 누르면 1 초 후 릴레이가 ON 되지만, 스위치를 놓으면 즉시 OFF 되는 온딜레이타이며 동작을 합니다.

### 3.2.2. 온딜레이된 원샷 타이머(OnDelayed OneShot timer)

시간지연이벤트의 사용예로 스위치가 ON 되면 1 초가 지난 후에 릴레이를 ON(온딜레이)시킨 후, 1초동안 ON 상태를 유지시키는(원샷) 예제입니다. (application\exam\delayed\_event.c 참고)

```
IF (this, DioIn) {
    if (Pressed(this->in, sw0)) { // 상승에지(양변환) 검출되면
        DioOutEvent* onEvent = CreateEvent(DioOut);
        onEvent->rly0 = ON;
        SendDelayedEvent(1000, onEvent); // 1초후 릴레이를 ON시킨다.

        DioOutEvent* offEvent = CreateEvent(DioOut);
        offEvent->rly0 = OFF;
        SendDelayedEvent(2000, offEvent); // 2초후 릴레이를 OFF시킨다.
    }
}
```

DIO 모듈의 sw0이 눌러지는 상승에지(Pressed)에서 릴레이를 동작시키는 DioOut 이벤트를 2개(onEvent, offEvent) 생성후 시간지연이벤트를 만들어서 이벤트큐에 넣었습니다. onEvent는 1초후 rly0을 ON 시키고, offEvent는 2초후 rly0을 OFF 시킵니다.

참고로, eventbus.h에 정의된 매크로함수 UserEvent(event\_var, tag)를 이용하면, DioOutEvent\* onEvent = CreateEvent(DioOut)을 간단히 UserEvent(onEvent,DioOut)로 바꿀수 있어 편리합니다.



### 3.3. 통신 프로토콜

이지컨트롤러는 `eventbus_net` 라이브러리를 기본으로 제공하고 있어 C#.NET, C++.NET, VB.NET 같은 닷넷언어를 사용하면 매우 쉽게 PC 용 프로그램을 개발할 수 있으므로, 가능하면 `eventbus_net` 라이브러리를 사용하기를 권장합니다. 그러나, 아래와 같은 경우 불가피하게 닷넷언어를 사용할 수 없습니다.

- 리눅스처럼 닷넷을 지원하지 않는 OS를 사용하는 경우.
- 이미 개발된 프로그램을 사용하여야 하는 경우.
- 빠른 속도가 필요해서 네이티브 언어를 사용해야 하는 경우.
- 사용중인 언어를 꼭 사용해야 하는 경우.
- 다른 컨트롤러(자체모델, PLC, LabView)에 연결하여 사용하려는 경우.

이외에도 다양한 이유가 있을 수 있을 것입니다. 이처럼 닷넷라이브러리를 사용할 수 없는 경우에는 컨트롤러와 직접 통신할 수 있도록 프로토콜을 구현해야 합니다.

이지컨트롤러의 기본적인 통신방식은 이벤트를 사용하며 모듈과 모듈, 컴퓨터와 모듈사이에 이벤트를 주고 받으면서 서로 유기적으로 동작합니다. 그러므로 사용자가 제작하였거나, 아니면 타사로 부터 구매한 컨트롤러라 하더라도 이러한 이벤트를 주고 받을 수 있도록 프로그램 하면 이지컨트롤러와 연결하여 사용할 수 있습니다.

비록 바이너리 프로토콜이 효율과 성능에서는 앞서지만, 만일 프로토콜의 구현에 어려움이 있다면 4 장에서 설명하는 AT 명령을 사용하는 것도 좋습니다.

#### 3.3.1. 바이너리 프로토콜

이벤트를 통신으로 주고 받기 위해서는 이벤트의 구조와 동작방식을 이해해야 합니다. 여기서는 이벤트의 구조에 대하여 살펴봅니다. 이벤트는 구조체 데이터 타입으로 되어 있으며, `eventbus.h` 파일에 선언되어 있습니다.

```
typedef struct {
    __POOL;                      // 시스템변수
    ESender    sender;           // 시스템변수
    ID         id;               // 여기서부터 이벤트헤드
    uint8      size;
    uint8      crc;
    EEventTag  tag;
} EventHead;
```

모든 이벤트는 `EventHead` 구조체로 시작합니다. 처음 두개의 시스템변수 선언은 이벤트버스의 동작에 필요한 변수이므로 프로토콜과 상관없습니다. 나머지 4 개의 변수는 이벤트 전송시 반드시 첫부분에 전송되어야 하는 4 바이트의 데이터입니다.

```
typedef struct {
    EventHead  e;
```

```

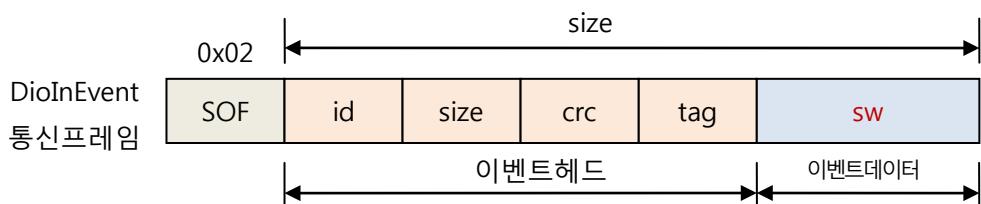
union {                                // 여기서부터 이벤트데이터
    uint16 sw;
    struct {
        ELogic sw0 :1;
        ELogic sw1 :1;
        ELogic sw2 :1;
        ELogic sw3 :1;
        ELogic sw4 :1;
        ELogic sw5 :1;
    };
};
} DioInEvent;

typedef struct {
    EventHead e;
    union {                                // 여기서부터 이벤트데이터
        uint16 rly;
        struct {
            ELogic rly0 :1;
            ELogic rly1 :1;
        };
    };
} DioOutEvent;

```

DIO 모듈의 입력이미지와 출력이미지입니다. 복잡해 보이지만, 입력이미지 DioInEvent 는 2 바이트 크기의 sw이고, 출력이미지 DioOutEvent 도 2 바이트 크기의 rly입니다.

DioInEvent 와 DioOutEvent 구조체는 모두 EventHead 를 가지고 있으므로 이벤트이며, 그러므로 이벤트큐에 넣고 전송될 수 있습니다. PC 또는 다른 컨트롤러와 이벤트를 주고 받을 때는 이 구조체를 1Byte 씩 통신으로 전송하면 되는데, EventHead 앞부분의 \_\_POOL, sender 는 시스템관리용 변수이므로 전송할 필요가 없으며 id 부터 전송합니다. 그리고, 수신측에서 프레임의 시작을 구분할 수 있도록 구분자 SOF(0x02)를 앞에 붙입니다.



#### ■ id : 이벤트가 속하는 모듈의 ID

(예) DIO(ID=4 라 가정)에서 자신의 입력이미지(DioInEvent)를 전송하는 경우, 입력이미지가 DIO 의 것이므로 id=4 이다.

(예) DIO(ID=4)에서 DOM 의 릴레이를 동작시키기 위해서 DOM(ID=5)의 출력이미지 (DomOutEvent)를 전송한다면, 출력이미지는 DOM 의 것이므로 id=5 이다.

(예) PC(ID=120)에서 DIO(ID=4)의 출력이미지 DioOutEvent 를 전송하는 경우, 이벤트는 DIO 의 것이므로 id=4 이다.

■ **size : SOF를 제외한 데이터의 바이트수**

DEL 시퀀스를 적용하기 이전의 원본 데이터의 길이(바이트수)입니다. DioInEvent 의 경우 6 이 됩니다.(아래의 “DEL 시퀀스” 참고)

■ **crc : 체크섬**

crc 바이트를 제외한 나머지(id 뒷부분 8)를 서로 XOR 한 값이며, 수신측에서도 crc 가 동일할때만 이벤트 프레임을 수신합니다.

■ **tag : 이벤트의 태그명, “9.11 EEventTag 열거형” 참고**

위의 경우, 이벤트 DioInEvent 의 태그는 DioIn, 이벤트 DioOutEvent 의 태그는 DioOut 입니다.

이벤트를 함수와 비교하면 쉽게 이해할 수 있습니다. tag 는 함수명에 해당하고, 이벤트데이터 sw 는 함수인자에 해당합니다. 위의 DioInEvent 를 전송한다는 것은 void DioIn (uint8 sw); 함수를 호출하는 것과 유사하다고 생각하면 이해하기 쉽습니다.

**\*중요\*** 이벤트는 구조체이며, 구조체의 이름은 태그이름 뒤에 “Event”를 붙여 사용하도록 약속한다. 예를 들면, “DioIn” 태그에 대한 이벤트는 “DioInEvent”가 된다.

(예제) 만일 여러분의 PCB 보드에서 DIO(ID=0x04)의 rly0=ON, rly1=ON 으로 동작시키려면 아래처럼 출력이벤트(DioOutEvent)를 전송하면 됩니다.

|             |            |              |             |             |               |
|-------------|------------|--------------|-------------|-------------|---------------|
| SOF<br>0x02 | id<br>0x04 | size<br>0x06 | crc<br>0x20 | tag<br>0x21 | rly<br>0x0003 |
|             |            |              |             |             | 0x03 0x00     |

tag(0x21)은 DioOut 태그입니다. rly 는 uint16 이므로 2 바이트입니다. LSB 2 비트만 사용하므로 0x0003 을 전송하면 되지만, LPC1114 는 리틀엔디안을 사용하므로 하위바이트(0x03)부터 전송해야 합니다.(인텔 CPU 를 사용하는 PC 도 리틀엔디안입니다)

### 3.3.2. DEL 시퀀스

그런데, 한가지 문제가 있습니다. 프레임의 시작을 알리기 위하여 SOF(0x02)를 사용하고 있으므로 이벤트헤더와 이벤트데이터에 SOF 가 나오면 문제가 됩니다. 어느것이 이벤트의 시작인지 구별되지 않습니다. 그래서 DEL 시퀀스를 사용합니다. 이는 이스케이프시퀀스와 비슷합니다. 전송할 문자가 SOF(0x02)와 같으면 MSB 를 ‘1’로 세트해서 0x82 로 바꾸고, 앞에 DEL(0x7F)을 추가하여 {0x7F, 0x82} 두 바이트를 전송합니다. 그렇다면, DEL(0x7F)이 나오면 뒤따르는 바이트를 다른 값으로 해석해 버릴테니, DEL 문자를 전송할 방법이 없어집니다. 그래서 DEL(0x7F)도 동일하게 DEL(0x7F)의 MSB 를 ‘1’로 세트한 0xFF 로 바꾸고, 앞에 DEL(0x7F)을 추가하여 {0x7F, 0xFF} 두 바이트를 전송합니다.

수신측에서는 SOF 가 수신되면 프레임의 시작으로 간주하고 이후부터 수신을 시작하면 되고, 수신중 DEL 문자를 만나면 그다음 바이트의 MSB 를 ‘0’으로 리셋한 값을 실제값으로 변환해서 사용하면 됩니다.

이렇게 DEL 시퀀스를 사용하는 이유는 실수로 또는 다른 장치로 부터 전송된 데이터가 이벤트로 오인식되는 경우를 배제시키기 위해서입니다.

현재 DEL 시퀀스를 사용하는 문자는 SOF, DEL 두 개 밖에 없으므로 쉽게 생각해서,

- 송신측에서는  
0x02 는 {0x7F, 0x82} 로 바꾸고, 0x7F 는 {0x7F, 0xFF}로 바꾸어서 전송한다.
- 수신측에서는  
{0x7F, 0x82}는 0x02로 바꾸고, {0x7F, 0xFF}는 0x7F로 바꾸어서 수신한다.

(예제) 만일 여러분의 PCB 보드에서 DIO(ID=0x04)의 rly0=OFF, rly1=ON 으로 동작시키려면 아래처럼 출력이벤트(DioOutEvent)를 전송하면 됩니다.

|             |            |              |             |             |               |
|-------------|------------|--------------|-------------|-------------|---------------|
| SOF<br>0x02 | id<br>0x04 | size<br>0x06 | crc<br>0x21 | tag<br>0x21 | rly<br>0x0002 |
|             |            |              |             |             | 0x02 0x00     |

tag(0x21)은 DioOut 태그입니다. rly 는 uint16 이므로 2 바이트입니다. LSB 2 비트만 사용하므로 0x0002 을 전송하면 되지만, LPC1114 는 리틀엔디안을 사용하므로 하위바이트(0x02)부터 전송해야 합니다.

그런데 rly 의 첫바이트가 0x02 이므로 DEL 시퀀스를 적용해서 {0x7F, 0x82}처럼 두바이트를 전송해야 하므로 아래와 같이 전송해야 합니다.

|             |            |              |             |             |                |
|-------------|------------|--------------|-------------|-------------|----------------|
| SOF<br>0x02 | id<br>0x04 | size<br>0x06 | crc<br>0x21 | tag<br>0x21 | rly<br>0x0002  |
|             |            |              |             |             | 0x7F 0x82 0x00 |

### 3.3.3. 이벤트의 종류

그러면, 언제 어떤 이벤트가 사용되는가?를 알면 위의 규칙대로 이벤트를 전송하면 됩니다. 사용자 이벤트의 경우 사용자가 필요에 의해 만들었으므로 언제 전송해야 할지 사용자가 잘 알고 있겠지만, 시스템에서 미리 정해둔 이벤트의 경우 언제 어떤 이벤트를 사용해야 하는지 알아야 합니다. (“9.11 EEventTag 열거형”에 설명된 이벤트의 기능을 참고하십시오)

- 지역(로컬)이벤트  
다른 모듈로 전송되지 않고 모듈 내에서만 사용되는 이벤트. 전송되지 않습니다.  
CyclicTimeout, Delayed
- XxxIn (0x20) 이벤트  
Xxx 는 모듈명을 의미합니다. (DIO 의 경우 DioIn 처럼)  
입력에 변동이 있을 때 다른 모듈에게 알리기 위하여 전송합니다.  
PC 나 다른 컨트롤러에서 이 이벤트를 수신하면 모듈의 입력상태를 알 수 있습니다.
- XxxOut (0x21) 이벤트  
출력에 변동이 있음을 알리기 위해 전송합니다. 그런데 한가지 생각할 점이 있습니다.  
XxxIn 의 경우 이벤트를 전송하는 모듈은 자신의 입력에 변동이 발생했음을 타모듈에

알리는 것이 목적입니다. 즉, DIO 모듈의 스위치입력에 변동이 있으면 DioIn 이벤트를 전송하게 됩니다.

하지만, DioOut 의 경우 DIO 모듈의 릴레이를 동작시킬 목적으로 누군가(DIO 자신 또는 다른 모듈에서) DIO 모듈의 출력이미지를 전송할 수 있다는 점에 주의해야 합니다. 예를들어, DIO의 릴레이를 ON 시키기 위하여 DioOutEvent 를 PC 가 전송하는 경우입니다. 이경우 이벤트를 수신한 DIO 는 이벤트데이터(rlly)를 출력이미지에 저장 후, 드라이버의 HW\_SyncToOut(Event\* pe) 함수를 호출해서 릴레이를 동작시킵니다. 즉, XxxOut 이벤트는 출력을 변경할 목적으로 전송할 수 있습니다.

#### ■ **UsbConnected(0x25), UsbDisconnected(0x26)** 이벤트

모듈의 UART 에 연결해서 이벤트를 주고 받으려면, 먼저 이벤트버스에 접속해야 하고, 경우에 따라서는 접속을 해제해야 합니다. 예를들어 컴퓨터가 DIO(ID=4)에게 통신으로 연결하려면 아래와 같이 이벤트를 보냅니다.

|             |                 |              |             |                           |
|-------------|-----------------|--------------|-------------|---------------------------|
| SOF<br>0x02 | id<br>DIO(0x04) | size<br>0x04 | crc<br>0x25 | tag<br>UsbConnected(0x25) |
|-------------|-----------------|--------------|-------------|---------------------------|

#### ■ **SyncRequested(0x27)** 이벤트

입출력 이미지의 전송을 요청합니다. 모듈은 SyncRequested 이벤트를 수신하면 자신의 입력이미지(XxxInEvent)와 출력이미지(XxxOutEvent)를 전송하고 드라이버의 IfSyncRequested() 함수를 호출합니다. 일반적으로 이 함수에는 VaaEvent, VbbEvent, VdcEvent 를 전송하도록 프로그램되어 있습니다. (드라이버의 IfSyncRequested() 함수 참고).

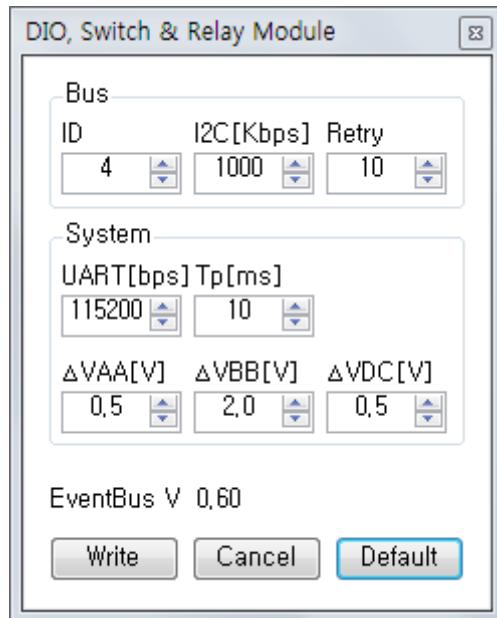
#### ■ 사용자 이벤트

사용자가 자신이 제작한 컨트롤러(또는 PC)와 이지컨트롤러 사이에 특정목적의 사용자 이벤트를 만들어 사용하는 것도 가능합니다.(앞절의 사용자이벤트 참고)

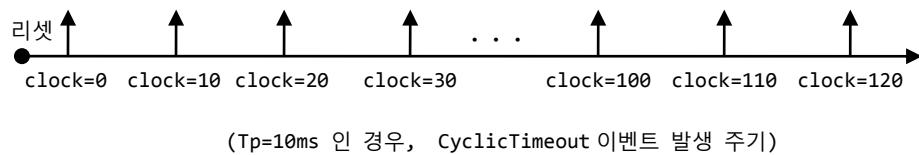
### 3.4. 시스템 시계, clock

```
extern CLOCK      clock;           // 시스템 타이머(밀리초 단위)
```

eventbus\_fw 는 전역변수 clock 으로 시간을 관리합니다(eventbus.h 참고). 전원이 투입되면 clock 에는 0ms 부터 시작하여 시간이 매 밀리초마다 증가합니다.



그림처럼 ezconfig.exe 를 실행하면 DIO 모듈의 경우 위 그림처럼 설정창이 나타납니다. 여기에는 clock 에 대한 설정값으로 반복주기(Tp)가 있습니다. Tp 는 반복적으로 CyclicTimeout 이벤트를 발생하는 시간간격입니다.



이지컨트롤러는 기본값으로 Tp=10ms 를 사용하고 있습니다. 물론 Tp 를 1ms 로 하면 시간관리가 쉽겠지만, 너무 빈번한 이벤트 발생은 바람직하지 않으므로 10ms 이상이 적절합니다.

```
typedef struct {
    EventHead e;
    uint64    ms;
} TimeEvent;

typedef TimeEvent CyclicTimeoutEvent;
```

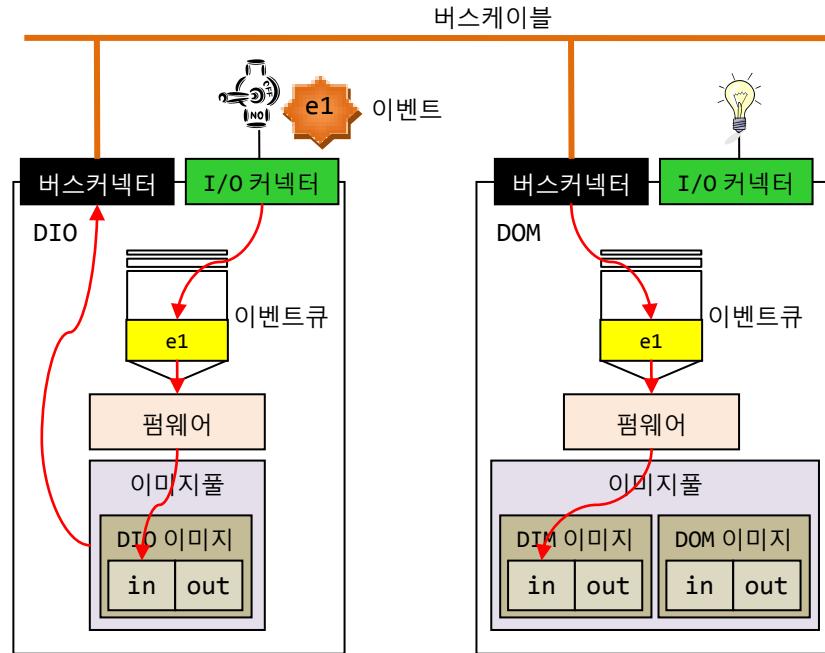
CyclicTimeout 이벤트는 TimeEvent 를 재정의하고 있는데, ms 에는 이벤트가 발생한 시간이 저장되어 있습니다.

\*주의\* Tp=10ms 로 설정된 상태에서 아래의 이벤트핸들러는 틀리게 동작합니다.

```
IF (this, CyclicTimeout) {  
    if (pe->ms % 105 == 0) {           // 105ms마다  
        // 실행할 코드  
    }  
}
```

프로그램의 본래 의도는 105ms 마다(105, 210, 315, 420...) 코드를 실행하는 것이지만, 위 그림에서 보듯이 이벤트가 발생한 시간(pe->ms)은 10ms 단위로 증가하므로 210ms, 420ms 에서만 코드가 실행되고 105ms, 315ms 에서는 실행되지 않습니다. 만일 정확히 105ms 마다 실행해야 한다면 ezconfig.exe에서 Tp=5ms로 설정해야 하지만, 그렇게 정확한 시간을 필요로하는 경우는 많지 않으므로 Tp의 배수인 100ms 또는 110ms 마다 실행하도록 수정하는 것이 좋습니다.

### 3.5. 유용한 테크닉



그림에서 DIO 에 연결된 스위치를 ON 하면 DioIn 이벤트가 생성되고 (dio\_drv.c 의 Eval\_DioIn 함수 참고), 이벤트큐에 들어갑니다. 그후 차례가 되면 이벤트루프가 이벤트(pe)를 꺼내어 펌웨어에 있는 이벤트핸들러에 넘겨줍니다.

여기에서 꼭 기억해야 할점은 이벤트핸들러에 도착한 이벤트(pe)는 새로 발생한 이벤트이며, pe->sw0 은 최근의 스위치 상태(ON)를 기억하고 있습니다. 그러나, DIO 이미지에는 아직 저장하지 않았으므로 this->in->sw0 에는 이전상태(OFF)가 저장되어 있습니다.

즉, 결론적으로 이벤트핸들러의 입장에서는 이미지(this)에 있는 값은 이전 값이고, 이벤트(pe)에 있는 값은 새로운 값이 됩니다. 이점을 이용하면 몇가지 유용한 이용방법이 가능합니다.

### 3.5.1. 이벤트 소스 판별

입력이미지에 변동이 발생하면 `Eval_DioIn()` 함수에서 이를 검출하여 DioIn 이벤트를 발생시킵니다. DIO 모듈의 입력은 6 개의 스위치(`sw0 ~ sw5`)로 구성되는데, 이중 한개의 스위치라도 상태가 바뀌면 이벤트는 발생합니다.

| 이전상태<br>this->in | sw0 | sw1 | sw2 | sw3 | sw4 | sw5 |
|------------------|-----|-----|-----|-----|-----|-----|
|                  | OFF | OFF | OFF | OFF | OFF | OFF |
| ↓                |     |     |     |     |     |     |
| 현재상태<br>pe       | sw0 | sw1 | sw2 | sw3 | sw4 | sw5 |
|                  | ON  | OFF | OFF | OFF | OFF | OFF |

그림처럼 `sw0` 이 `ON` 되면 이벤트가 발생하는데, 어느 스위치가 동작했는지 알려면 이벤트핸들러에서는 이전상태와 현상을 서로 비교해야 동작한 스위치를 알 수 있습니다.

```
IF (this, DioIn) {
    if (this->in->sw0 != pe->sw0) {
        // sw0이 동작한 경우의 처리
    }
}
```

여기서 `this->in->sw0` 은 이전상태를, `pe->sw0` 은 현상을 나타내므로 `sw0` 이 동작하였음을 알 수 있습니다.

\*Tip\* 매크로함수를 사용하면 가독성을 향상시킬 수 있습니다. `Toggled()` 매크로는 `eventbus.h`에 이미 선언되어 있으므로 그대로 사용하기만 하면 됩니다.

```
#define Toggled(image, io) (image->io != pe->io)

IF (this, DioIn) {
    if (Toggled(this->in, sw0)) {
        // sw0이 동작한 경우의 처리
    }
}
```

### 3.5.2. 상승(하강) 에지 검출

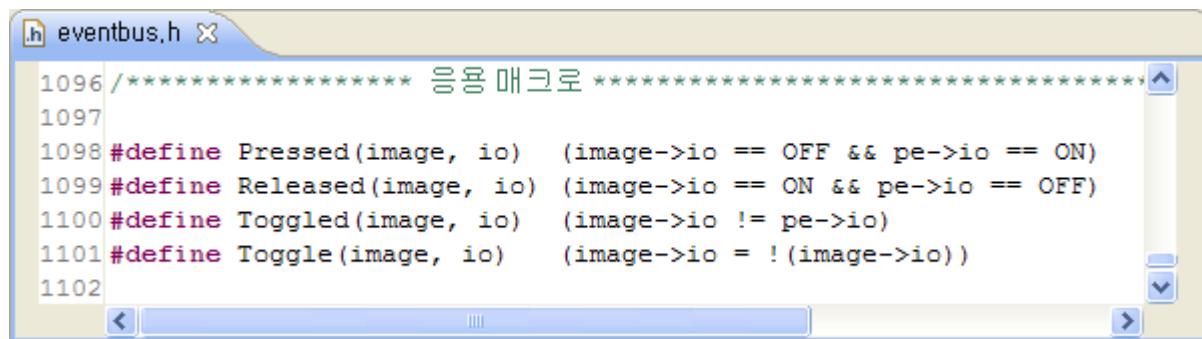
경우에 따라서는 스위치의 상태보다 에지에 대한 정보가 더욱 중요한 경우가 많습니다. 위의 예를 응용하면, 스위치가 눌러(상승에지, Pressed)졌는지, 아니면 릴리즈(하강에지, Released)되었는지 판별할 수 있습니다. 상승에지란 이전상태가 OFF이고 현재상태가 ON인 경우이므로 다음과 같이 간단히 판단할 수 있습니다.

```
IF (this, DioIn) {
    if (this->in->sw0 == OFF && pe->sw0 == ON) {
        // sw0이 눌러진(상승) 경우의 처리
    }
}
```

또는 간단히,

```
IF (this, DioIn) {
    if (!this->in->sw0 && pe->sw0) {           // ON은 true, OFF는 false
        // sw0이 눌러진(상승) 경우의 처리
    }
}
```

마찬가지로, eventbus.h에 선언된 매크로를 이용하면 코드의 가독성을 높일 수 있습니다.



```
IF (this, DioIn) {
    if (Pressed(this->in, sw0)) {
        // sw0이 눌러진(상승에지) 경우의 처리
    }

    if (Released(this->in, sw0)) {
        // sw0이 놓아진(상승에지) 경우의 처리
    }

    if (Toggled(this->in, sw0)) {
        // sw0이 조작된(상승 또는 하강에지) 경우의 처리
    }

    Toggle(this->out, rly0) // this->out->rly0을 반전시킴
}
```

### 3.6. EEPROM 사용자 영역

eventbus\_fw\_0.60 부터 사용자 데이터를 EEPROM 에 영구저장할 수 있습니다. 예를 들어, 사용시간을 누적으로 기록하거나, 전원을 내렸을 때도 지워지지 않는 설정값을 저장하는 등의 프로그램에 사용할 수 있습니다.

```
#ifndef __DIO_DEF_H_
#define __DIO_DEF_H_

#define dio_def      // application selection
#define dio_drv     // driver selection

struct {
    // the variables appended here is
    // stored to eeprom with write_rom();
    // and restored from eeprom with read_rom();
    // for example,
    // uint8 mydata;
    // struct {
    //     int     var1;
    //     uint16  var2;
    // } var;
    uint32 my_var;
} rom;

#endif
```

위 코드는 dio\_def.h 에 정의된 rom 변수입니다. 여기에 사용자가 저장하고자 하는 변수를 선언한 후, dio\_def.c 의 main()에서 WriteROM;으로 EEPROM 에 쓸 수 있습니다.

그리고, EEPROM 에 저장된 값을 읽어서 변수 rom 에 저장하는 매크로는 ReadROM; 입니다. 그러나, 전원이 입력되면 초기화 과정에서 ReadROM;이 실행되므로 사용자는 EEPROM 을 읽지 않아도 자동으로 변수 rom 에 저장된 값이 EEPROM 에서 읽혀지므로 사용자는 이전에 저장한 값을 rom.my\_var 처럼 사용하기만 하면됩니다.

(주의) EEPROM 의 전체용량은 128 바이트입니다. 그리고, 여기에는 각모듈의 설정값인 XxxParamEvent(각 모듈의 이미지 구조체 참고, eventbus.h)도 저장되므로 XxxParamEvent 를 제외한 나머지 공간만 사용할 수 있습니다. 예를들어, DioParamEvent 의 경우 32 바이트이므로 DIO 모듈의 경우 사용자용 공간은 128-32 바이트보다 작아야 하므로 주의가 필요합니다.

#### 3.6.1. RCS 적용예제

RCS 모듈에서 모터의 마지막 위치를 기억해서 전원이 켜졌을 때 이전 위치로 초기화하는 예제를 살펴보겠습니다. 우선, rcs\_def.h 에 EEPROM 에 저장할 변수(pwm0)를 아래처럼 선언합니다.

```
#ifndef __RCS_DEF_H_
#define __RCS_DEF_H_

#define rcs_def      // application selection
#define rcs_drv     // driver selection
```

```

struct {
    uint16 pwm0;      // EEPROM에 저장할 모터0의 위치값
} rom;
#endif

```

그런 후, rcs\_def.c 에 아래처럼 코드를 추가합니다.

```

int main()
{
    EventBus();

    this->out->pwm[0] = rom.pwm0;      // EEPROM에 저장된 위치를 출력이미지에 복사하고
    SyncOut(this);                      // RcsOut 이벤트를 발생해서 출력을 갱신한다.

    EventProgram;
    IF(this, RcsOut) {                // 출력이미지에 변경이 있으면
        rom.pwm0 = pe->pwm[0];        // 모터 위치를 저장하고
        WriteROM;                     // EEPROM에 기록한다.
    }
    ScanProgram;

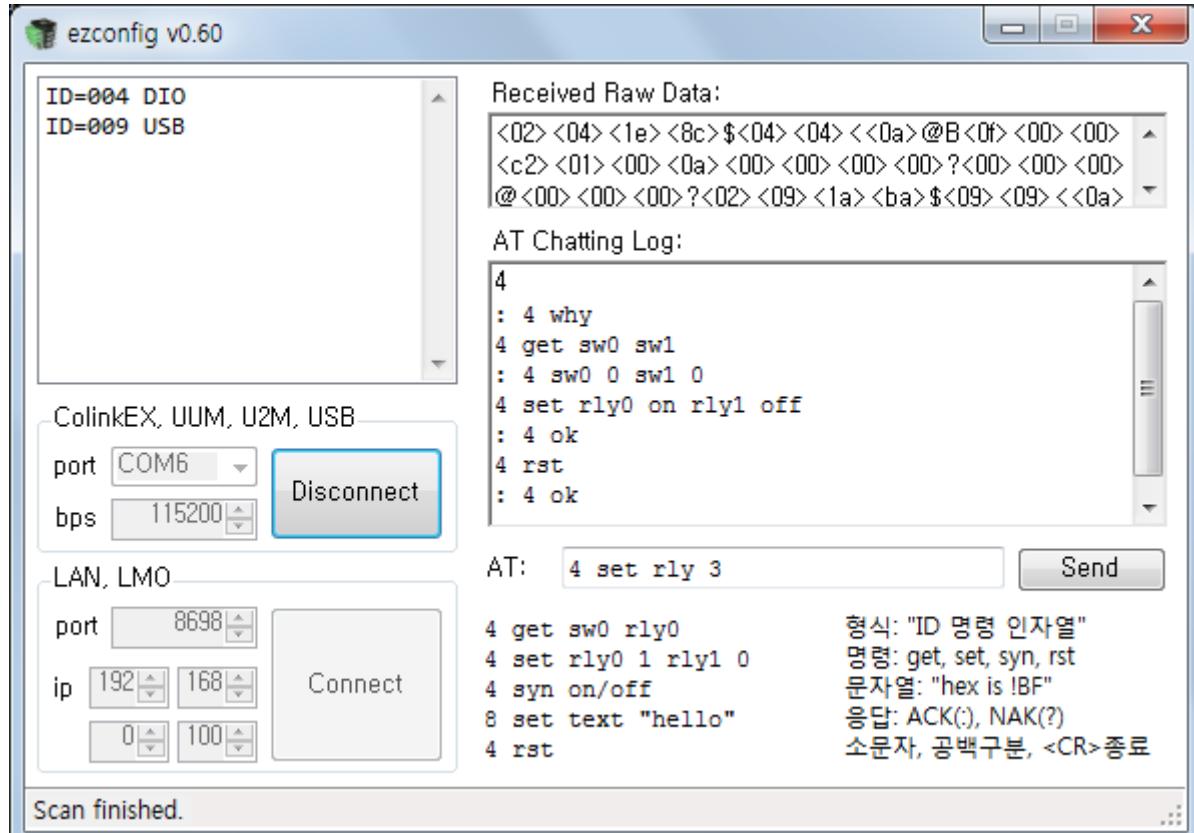
    End;
}

```

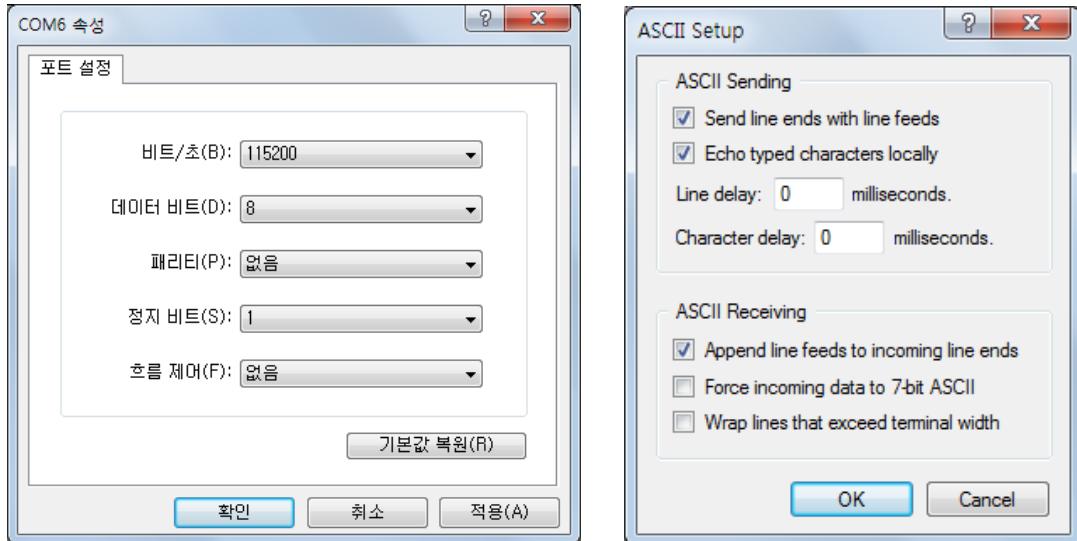
이제 전원을 껐다 켜면 최종 위치로 복원됩니다.

## 4. AT 명령

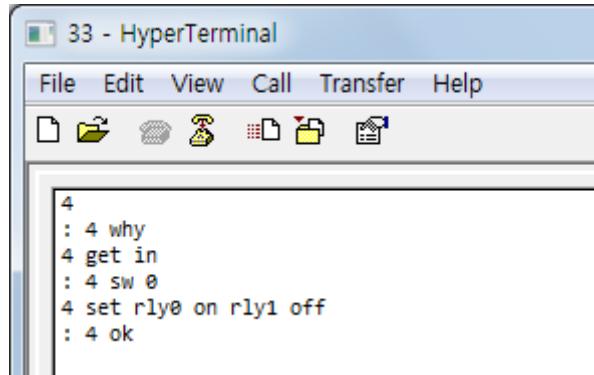
eventbus\_fw\_0.60 버전부터 텍스트로 모듈을 제어할 수 있는 AT 명령이 추가되어 처음 시작하는 분들도 매우 쉽고 간편하게 동작시킬 수 있으며, 현장에서 설치후 배선의 정상유무 장치의 동작여부등을 쉽게 검사하는데 사용할 수 있습니다.



`ezconfig.exe`에 추가된 AT 명령 전용의 채팅프로그램으로 편리하게 사용할 수 있습니다. UART 또는 LAN 모듈을 통해서 연결할 수 있으며, 상단에는 수신한 데이터를 그대로 표시하고, 하단에는 AT 명령만 표시합니다. 수신데이터에서 <0d>는 비문자데이터이므로 16 진 Ascii 로 표시하고 있습니다.



물론 위 그림처럼 윈도 XP 사용시는 하이퍼터미널을 사용해도 됩니다. 단, AT 명령은 ID로 시작하고, <CR> 문자로 끝나는 규칙을 사용하므로, 하이퍼터미널에서 <CR>을 전송하도록 설정해야 합니다. 속성에서 [ASCII Setup]을 선택하고 우측 그림처럼 체크해 주신후 사용하면 됩니다.



그림은 하이퍼터미널에서 제어하는 예입니다. 통신연결후 ID=4 가 연결되었는지 확인후 전압과 스위치를 읽었으며 릴레이를 온오프 시켰습니다.

`ezconfig.exe` 의 소스코드가 공개되어 있으므로 참고하시면 프로그램에서 AT 명령으로 제어하는 방법을 더욱 상세히 이해할 수 있습니다. 또한 `eventbus_fw_0.60` 의 드라이버 소스코드에 AT 명령을 구현하는 `AT_GetCommand()`, `AT_SetCommand()` 함수가 있으므로 이를 참고해서 필요한 형태로 추가 및 변경해서 사용할 수도 있습니다.

#### 4.1. AT 명령 규칙

AT 명령의 규칙은 매우 간단합니다. 명령을 내리는 방법과 응답을 읽는 방법만 알면 됩니다.

$$\text{명령라인} = \text{ID} + \text{명령어} + \text{속성}$$

한줄의 명령은 모듈의 ID로 시작해서 소문자로 된 명령어와 속성이 공백문자로 분리됩니다. 그리고 명령라인은 100자 이내의 길이를 사용해야 합니다.

■ ID

모듈의 주소, ezconfig.exe에서 설정한다.

■ 명령어

**get** : 읽기. 속성에 읽을 이미지를 나열한다. 예) 4 get sw0 sw1

**set** : 쓰기. 속성에 쓸 이미지와 값을 나열한다. 예) 4 set rly0 on rly1 off

**syn** : 동기화. 속성은 on/off 가 온다. 예) 4 syn on

**rst** : 모듈을 리셋(초기화)시킨다. 예) 4 rst

**id** : 별다른 명령없이 id 만 전송하면 해당 ID 의 모듈이 연결되어있는지 확인하는 명령이 된다. 연결되었으면 “: 4”처럼 그렇지 않으면 응답이 없다.

(단, on 은 1 을 off 는 0 을 대신해서 사용될 수 있다)

■ 속성

eventbus.h에 있는 모듈의 이미지 구조체의 멤버변수의 이름이다.

예) DIO의 경우 sw, sw0, sw1, sw2, sw3, sw4, sw5, rly, rly0, rly1

추가적으로 전체입력은 in, 전체출력은 out 이라는 속성명을 가진다.

예) 4 get in      4 get out

응답 메시지의 구성도 매우 간단합니다.

**응답메시지 = :/? + ID + 처리결과**

■ :/? (처리상태)

‘:’은 정상적으로 명령이 처리되었음을 알리며, 이경우 처리결과가 뒤따라 온다.

예) 4 get sw0   ->   : 4 sw0 0   (sw0이 off임을 알린다)

‘?’은 명령을 이해하지 못했음을 알리며, 이해하지 못한 명령을 보여줍니다.

예) 4 get sw6   ->   ? 4 sw6   (DIO는 sw0~sw5 가 있으며 sw6은 없다)

■ ID

명령을 처리한 모듈의 주소. 명령의 ID 와 동일하므로 여러 모듈에 명령을 내린 경우 이 주소로 구분할 수 있다.

■ 처리결과

별다른 응답이 없는 명령의 경우 ok 로 응답한다. 예) : 4 ok

이미지를 읽는 경우 이미지와 값을 함께 응답한다. 예) : 4 sw0 0 sw1 0

## 4.2. 동기화 명령

---

명령어 syn 은 동기화 명령으로 명령을 보내지 않아도 값이 변경될 때(내부적으로 이벤트가 발생할 때)마다 응답메시지를 자동으로 전송합니다.

■ 4 syn on            // 동기화 명령

: 4 ok                // 동기화 명령이 제대로 시작됨

: 4 sw 1             // 여기서 sw0 스위치를 온시켰다.

```
: 4 sw 0          // 여기서 sw0 스위치를 오프시켰다.  
4 syn off       // 동기화 종료명령
```

### 4.3. 문자열 표현

---

AT 명령에서 문자열 데이터를 사용할 경우도 있습니다. SEG 모듈에 문자열을 표시하거나, MSB 의 시리얼포트에 문자열을 출력하는 경우 등입니다.

1 바이트 16 진수 값 = !00 ~ !ff

문자열뿐 아니라 비문자데이터도 전송할 수 있습니다. !(느낌표)를 사용하면 1 바이트의 16 진수 값을 전송할 수 있습니다. 반드시 두자리를 사용해야 합니다. !0 또는 !0FF 처럼 사용해서는 않습니다. 단, 문자열을 표현하기 위해서 두개의 문자 ‘”’와 ‘!’를 사용하게 되므로 이 두문자는 문자열에 나타나서는 않됩니다.

- !(0x21), “(0x22)  
두문자는 위처럼 특수용도로 사용되므로 문자열에 올 수 없습니다. 두 문자를 문자열에 사용하려면 ‘!’은 !21로, “”은 !22로 입력해야 합니다.
- “abc def!00!21!22”  
!21(‘!’), !22(“”)이므로 이 문자열은 아래처럼 인식됩니다.

|     |     |     |     |     |     |     |      |             |            |
|-----|-----|-----|-----|-----|-----|-----|------|-------------|------------|
| ‘a’ | ‘b’ | ‘c’ | ‘ ’ | ‘d’ | ‘e’ | ‘f’ | 0x00 | 0x21<br>‘!’ | 0x22<br>“” |
|-----|-----|-----|-----|-----|-----|-----|------|-------------|------------|

- 8 set text “hello” -> SEG 에 “hello”를 표시한다.  
8 set text “hell!ef -> SEG 에 “hello.”를 표시한다.  
소문자 ‘o’는 0x6f 인데 최상위비트(MSB)를 ‘1’로 세트하면 세그먼트의 소수점이 ON 되므로 0xef 는 ‘o.’을 나타냅다 (SEG 참고).

### 4.4. 각 모듈별 사용예

---

#### 4.4.1. AIM 모듈

```
// eventbus.h 참고
typedef struct {
    EventHead e;
    float      vin[4];
} AIM_IN, AimInEvent;

typedef Event AIM_OUT, AimOutEvent;
```

AIM 모듈의 입출력이미지는 위와 같습니다. 출력은 없으며, 전압입력 vin 이 4 개 있습니다. 다음은 사용예입니다.

```
1 get in
: 1 vin0 0.048 vin1 0.146 vin2 0.107 vin3 0.097

1 get vin0
: 1 vin0 0.048
```

```

1 get vin0 vin1 vin2 vin3
: 1 vin0 0.048 vin1 0.146 vin2 0.107 vin3 0.097

1 syn on
: 2 ok

: 1 vin2 0.097
: 1 vin0 0.048
: 1 vin0 0.009 vin2 0.146

1 syn off
: 2 ok

```

#### 4.4.1. AOM 모듈

```

// eventbus.h 참고
typedef Event AOM_IN, AomInEvent;

typedef struct {
    EventHead e;
    float vout[2];
    float iout[2];
} AOM_OUT, AomOutEvent;

```

AOM 모듈의 입출력이미지는 위와 같습니다. 입력은 없으며, 출력 vout,iout 이 각각 2 개씩 있습니다. 다음은 사용예입니다.

```

2 get out
: 2 vout0 0.000 vout1 0.000 iout0 0.000 iout1 0.000

2 get vout0 vout1 iout0 iout1
: 2 vout0 0.000 vout1 0.000 iout0 0.000 iout1 0.000

2 get vout0
: 2 vout0 0.000

2 set vout0 10 iout0 20
: 2 ok

2 syn on
: 2 ok

: 2 vout0 5.000 iout0 10.000      <- 다른 모듈에서 AOM의 vout0,iout0을 변경시켰다.

2 syn off
: 2 ok

```

#### 4.4.2. DIE 모듈

```

// eventbus.h 참고
typedef struct {
    EventHead e;
    union {
        uint16 sw;
        struct {
            ELogic sw0 :1;
            ELogic sw1 :1;
            ELogic sw2 :1;
            ELogic sw3 :1;
        }
    }
}
```

```

    ELogic sw4      :1;
    ELogic sw5      :1;
    ELogic sw6      :1;
    ELogic sw7      :1;
    ELogic sw8      :1;
    ELogic sw9      :1;
    ELogic sw10     :1;
    ELogic sw11     :1;
    ELogic sw12     :1;
    ELogic sw13     :1;
    ELogic sw14     :1;
    ELogic sw15     :1;
}
};

} DIE_IN, DieInEvent;

typedef Event DIE_OUT, DieOutEvent;

```

DIE 모듈의 입출력이미지는 위와 같습니다. 출력은 없으며, 입력 sw0...sw15 와 모든 스위치를 한번에 읽을 수 있도록 sw 가 있습니다. 다음은 사용예입니다.

```

13 get in
: 13 sw 3

13 get sw
: 13 sw 3

13 get sw0 sw1 sw2 sw3
: 13 sw0 1 sw1 1 sw2 0 sw3 0

13 syn on
: 13 ok

: 13 sw 8          <- SW3에 연결된 스위치를 ON시킴
: 13 sw 0          <- SW3에 연결된 스위치를 OFF시킴

13 syn off
: 13 ok

```

#### 4.4.3. DIM 모듈

```

// eventbus.h 참고
typedef struct {
    EventHead e;
    union {
        uint16 sw;
        struct {
            ELogic sw0 :1;
            ELogic sw1 :1;
            ELogic sw2 :1;
            ELogic sw3 :1;
            ELogic sw4 :1;
            ELogic sw5 :1;
            ELogic sw6 :1;
            ELogic sw7 :1;
            ELogic sw8 :1;
            ELogic sw9 :1;
        };
    };
}
```

```

    };
} DIM_IN, DimInEvent;

typedef Event DIM_OUT, DimOutEvent;

```

DIM 모듈의 입출력이미지는 위와 같습니다. 출력은 없으며, 입력 sw0...sw9 와 모든 스위치를 한번에 읽을 수 있도록 sw 가 있습니다. 다음은 사용예입니다.

```

3 get in
: 3 sw 1

3 get sw
: 3 sw 1

3 get sw0 sw1 sw2 sw3
: 3 sw0 1 sw1 0 sw2 0 sw3 0

3 syn on
: 3 ok

: 3 sw 8          <- SW3에 연결된 스위치를 ON시킴
: 3 sw 0          <- SW3에 연결된 스위치를 OFF시킴

3 syn off
: 3 ok

```

#### 4.4.4. DOE 모듈

```

// eventbus.h 참고
typedef Event DOE_IN,DoeInEvent;

typedef struct {
    EventHead e;
    union {
        uint16 rly;
        struct {
            ELogic rly0      :1;
            ELogic rly1      :1;
            ELogic rly2      :1;
            ELogic rly3      :1;
            ELogic rly4      :1;
            ELogic rly5      :1;
            ELogic rly6      :1;
            ELogic rly7      :1;
            ELogic rly8      :1;
            ELogic rly9      :1;
            ELogic rly10     :1;
            ELogic rly11     :1;
            ELogic rly12     :1;
            ELogic rly13     :1;
            ELogic rly14     :1;
            ELogic rly15     :1;
        };
    };
} DOE_OUT,DoeOutEvent;

```

DOE 모듈의 입출력이미지는 위와 같습니다. 입력은 없으며, 출력 rly0...rly15 와 모든 출력을 한번에 쓸 수 있도록 rly 가 있습니다. 다음은 사용예입니다.

```

14 get out
: 14 rly 0

14 get rly
: 14 rly 0

14 get rly0 rly1
: 14 rly0 0 rly1 0

14 set rly0 on rly1 on
: 14 ok

14 set rly 0           <- 모든 릴레이를 OFF 한다.
: 14 ok

14 syn on
: 14 ok

: 14 rly 49152        <- 다른 모듈이 DOE의 rly14,rly15를 ON시켰다.

14 syn off
: 14 ok

```

#### 4.4.5. DIO 모듈

```

// eventbus.h 참고
typedef struct {
    EventHead e;
    union {
        uint16 sw;
        struct {
            ELogic sw0 :1;
            ELogic sw1 :1;
            ELogic sw2 :1;
            ELogic sw3 :1;
            ELogic sw4 :1;
            ELogic sw5 :1;
        };
    };
} DIO_IN, DioInEvent;

typedef struct {
    EventHead e;
    union {
        uint16 rly;
        struct {
            ELogic rly0   :1;
            ELogic rly1   :1;
        };
    };
} DIO_OUT, DioOutEvent;

```

DIM 모듈의 입출력이미지는 위와 같습니다. 입력 sw0...sw5 와 모든 스위치를 한번에 읽을 수 있도록 sw 가 있으며, 출력 rly0,rly1 과 모든 릴레이를 한번에 읽고쓸수 있도록 rly 가 있습니다. 다음은 사용예입니다.

```

4 get in
: 4 sw 0

4 get sw
: 4 sw 0

4 get sw0 sw1
: 4 sw0 0 sw1 0

4 get out
: 4 rly 0

4 get rly
: 4 rly 0

4 get rly0 rly1
: 4 rly0 0 rly1 0

4 set rly 3
: 4 ok

4 set rly0 off rly1 on
: 4 ok

4 syn on
: 4 ok

: 4 sw 1      <- SW0에 연결된 스위치를 2회 ON/OFF 시킴
: 4 sw 0
: 4 sw 1
: 4 sw 0

4 syn off
: 4 ok

```

#### 4.4.6. DOM 모듈

```

// eventbus.h 참고
typedef Event DOM_IN,DomInEvent;

typedef struct {
    EventHead e;
    union {
        uint16 rly;
        struct {
            ELogic rly0 :1;
            ELogic rly1 :1;
            ELogic rly2 :1;
            ELogic rly3 :1;
            ELogic rly4 :1;
            ELogic rly5 :1;
        };
    };
}DOM_OUT,DomOutEvent;

```

DOM 모듈의 입출력이미지는 위와 같습니다. 입력은 없으며, 출력 rly0...rly5 와 모든 출력을 한번에 쓸 수 있도록 rly 가 있습니다. 다음은 사용예입니다.

```
5 get out
: 5 rly 0

5 get rly
: 5 rly 0

5 get rly0 rly1
: 5 rly0 0 rly1 0

5 set rly0 on rly1 on
: 5 ok

5 set rly 0           <- 모든 릴레이를 OFF 한다.
: 5 ok

5 syn on
: 5 ok
: 5 rly 1           <- 다른 모듈이 DOM의 rly0을 ON시켰다.
5 syn off
: 5 ok
```

#### 4.4.7. ENV 모듈

```
// eventbus.h 참고
typedef struct {
    EventHead e;
    int16 temp; // temperature
    uint16 uv; // ultra violet
    uint32 lux; // brightness
    uint16 hand; // obation
} ENV_IN,EnvInEvent;

typedef Event ENV_OUT,EnvOutEvent;
```

ENV 모듈의 입출력이미지는 위와 같습니다. 출력은 없으며, 입력 temp,uv,lux,hand 가 있습니다. 다음은 사용예입니다.

```
6 get in
: 6 temp 30 uv 7632 lux 47 hand 0

6 get lux uv
: 6 lux 47 uv 7432

6 syn on
: 6 ok

: 6 hand 1
: 6 hand 2
: 6 temp 31
: 6 lux 57
: 6 uv 8032
: 6 hand 2
: 6 lux 52

6 syn off
: 6 ok
```

#### 4.4.8. MOC 모듈

```
// eventbus.h 참고
typedef struct {
    EventHead e;
    union {
        struct {
            ELogic pos      :1;
            ELogic neg      :1;
            ELogic nnp      :1;
            ELogic jcw      :1;
            ELogic jccw     :1;
            ELogic jspd     :1;
            Bool   busy     :1;
        };
        uint16 status;
    };
    uint16 jvol;
    int32 loc;
} MOC_IN,MocInEvent;

typedef struct {
    EventHead e;
    union {
        struct {
            MOC_CMD cmd      :4;
            MOC_DIR dir      :1;
        };
        uint16 config;
    };
    uint16 ipps;
    uint16 accel;
    uint32 pps;
    uint32 move;
} MOC_OUT,MocOutEvent;
```

MOC 모듈의 입출력이미지는 위와 같습니다. 다음은 사용예입니다.

```
10 get in                      <- 모든 입력
: 10 status 0 jvol 0 loc -2147483648    <- 원점(NEG) 설정되지 않으면 음의 최대값

10 get status                   <- 전체 상태값 읽기
: 10 status 0

10 get pos neg nnp jcw jccw jspd busy    <- 항목별로 상태값 읽기
: 10 pos 0 neg 0 nnp 0 jcw 0 jccw 0 jspd 0 busy 0

10 get jvol loc
: 10 jvol 0 loc -2147483648

10 get out                      <- 모든 출력
: 10 config 0 ipps 100 accel 20 pps 20000 move 0    <- 초기값

10 get config                   <- 전체 설정값 읽기
: 10 config 0
```

```

10 get cmd dir           <- 항목별로 설정값 읽기
: 10 cmd free dir cw

10 get ipps accel pps move
: 10 ipps 100 accel 20 pps 20000 move 0

10 set cmd run dir cw ipps 100 accel 10 pps 10000   <- CW방향으로 회전시작
: 10 ok

10 set cmd stop          <- 정지
: 10 ok

10 set cmd run dir ccw      <- 나머지 설정은 동일하게 CCW방향으로 회전
: 10 ok

10 set cmd move move 30000    <- RUN으로 회전중 MOVE 이동명령. 약 3초후 정지
: 10 ok

```

#### 4.4.9. MSB 모듈

// eventbus.h 참고

```

typedef Event MSB_IN, MsbInEvent;
typedef Event MSB_OUT, MsbOutEvent;

MSB 모듈의 입출력이미지는 기본 이미지를 사용하므로 액세스할 항목이 없습니다.  

시리얼포트의 출력(전송)은 내부적으로 이미지를 사용하지 않고 사용자 이벤트를 사용합니다.  

시리얼포트의 수신은 syn 명령을 사용합니다. 다음은 사용예입니다.

```

```

15 con on
: 15 ok

15 syn on
: 15 ok

15 set tx0 "loopback test from ch0 to ch2"
: 15 ok

: 15 rx2 "loopback test from ch0 to ch2"

15 set tx2 "12345678901234567890123456789012345678901234567890"   <- 60글자
: 15 ok

: 15 rx0 "12345678901234567890123456789012345678901234567890"   <- 50글자(패킷최대)
: 15 rx0 "1234567890"   <- 10글자

15 set tx0 "hello !21"      <- 실제 포트에는 "!21"이 아니라 0x21이 전송된다.
: 15 ok

: 15 rx2 "hello !21"    <- 수신부는 0x21(문자가 아닌 값)을 !nn(16진수)으로 바꾸어 PC로 보낸다.

15 set tx0 "!01!02!03!04!05!06!07!08!09!10!01!02!03!04!05!06!07!08!09!10!01!02!03!04!05"
: 15 ok
    <- 실제 포트에는 0x01,0x02,...처럼 16진수 값으로 전송되므로 총 25바이트가 전송된다.

: 15 rx2 "!01!02!03!04!05!06!07!08!09!10!01!02!03!04!05!06!07!08!09!10!01!02!03!04!05"
    <- 채널2에서 수신된 결과는 16진수 25바이트 이지만 !nn(16진수)으로 바꾸어 수신된다.

```

```
15 set tx0 "01!02!03!04!05!06!07!08!09!10!01!02!03!04!05!06!07!08!09!10!
? 15 too long line
<- AT명령은 한번에 100자를 넘을 수 없다.
```

#### 4.4.10. SEG 모듈

```
// eventbus.h 참고
typedef struct {
    EventHead e;
    uint16 cds;           // illumination on CDS cell
} SEG_IN, SegInEvent;

typedef struct {
    EventHead e;
    char text[SEG_CHARS]; // SEG_CHARS는 8
} SEG_OUT, SegOutEvent;
```

SEG 모듈의 입출력이미지는 위와 같습니다. 입력은 주변의 밝기를 나타내는 cds 가 있고, 출력은 세그먼트에 표시할 text 가 있습니다. 다음은 사용예입니다.

```
7 get in
: 7 cds 91

7 get cds
: 7 cds 91

7 set text "hello"
: 7 ok

7 get text
: 7 "hello"

7 get out
: 7 "hello"

7 set text "hell!ef"      <- 0xEF는 'o'의 MSB를 '1'로 세트한 값이므로 "hello."으로 표시
: 7 ok

7 get text
: 7 "hell!EF"
```

#### 4.4.11. SMD 모듈

```
// eventbus.h 참고
typedef struct {
    EventHead e;
    float vm;
    int16 heat;
    union {
        struct {
            Bool fan :1;
            Bool busy0 :1;
            Bool busy1 :1;
        };
        uint16 status;
    };
} SMD_IN, SmdInEvent;
```

```

typedef struct {
    union {
        struct {
            SMD_CMD     cmd      :4;
            SMD_STEP    step     :2;
            SMD_DIR     dir      :1;
            Bool        run      :1;
        };
        uint16 config;
    };
    uint16 ipps;
    uint16 accel;
    uint32 pps;
    uint32 move;
} SMD_SETTING;

typedef struct {
    EventHead   e;
    SMD_SETTING m0;
    SMD_SETTING m1;
    uint16      amp;
} SMD_OUT,SmdOutEvent;

```

SMD 모듈의 입출력이미지는 위와 같습니다. 다음은 사용예입니다.

```

8 get in                         <- 모든 입력
: 8 vm 24.0 heat 24 status 0      <- 모터전원, 방열판 온도, 전체 상태값 읽기

8 get fan busy0 busy1      <- 항목별로 상태값 읽기
: 8 fan 0 busy0 0 busy1 0

8 get out
: 8 amp 10

8 get amp
: 8 amp 10

8 get m0
: 8 config0 0 ipps0 100 accel0 20 pps0 20000 move0 0

8 get m1
: 8 config1 0 ipps1 100 accel1 20 pps1 20000 move1 0

8 get cmd0 step0 dir0 run0
: 8 cmd0 free step0 16 dir0 cw run0 0

8 get cmd1 step1 dir1 run1
: 8 cmd1 free step1 16 dir1 cw run1 0

8 get ipps0 accel0 pps0 move0
: 8 ipps0 100 accel0 20 pps0 20000 move0 0

8 get ipps1 accel1 pps1 move1
: 8 ipps1 100 accel1 20 pps1 20000 move1 0

8 syn on
: 8 ok

8 set run0 on cmd0 run step0 16 dir0 cw ipps0 100 accel0 10 pps0 10000 amp 20
: 8 ok

```

```

: 8 amp 20      <- 전류 2A
: 8 config0 131 ipps0 100 pps0 10000 accel0 10 move0 0
    <- 131은 이진수로 100000110이므로, 1(run=on) 0(dir=CW) 00(step=16분주) 0011(cmd=RUN)
: 8 status 2
    <- busy0=on, 즉 모터0이 회전중임

8 set run0 on cmd0 stop
: 8 ok

: 8 config0 130 ipps0 100 pps0 10000 accel0 10 move0 0
    <- 130은 이진수로 10000010이므로, 1(run=on) 0(dir=CW) 00(step=16분주) 0010(cmd=STOP)
: 8 status 0
    <- busy0=off, 즉 모터0이 정지함

8 set run0 off run1 off amp 20      <- 전류 2A 설정
: 8 ok

8 set cmd0 run step0 16 dir0 cw ipps0 100 accel0 10 pps0 10000 <- 모터0 설정
: 8 ok

8 set cmd1 run step1 16 dir1 cw ipps1 100 accel1 10 pps1 10000 <- 모터1 설정
: 8 ok

8 get busy0 busy1          <- 현재 모터상태: m0,m1 모두 정지
: 8 busy0 0 busy1 0

8 set run0 on run1 on      <- m0,m1 회전개시
: 8 ok

8 get busy0 busy1          <- 현재 모터상태: m0,m1 모두 회전
: 8 busy0 1 busy1 1

8 get run0 run1           <- run0, run1이 모두 ON이므로
: 8 run0 1 run1 1

8 set cmd0 free cmd1 free      <- 바로 명령만 free로 바꿔주어도
: 8 ok

8 get busy0 busy1          <- 모터는 정지한다.
: 8 busy0 0 busy1 0

8 syn off
: 8 ok

```

#### 4.4.12. TCM 모듈

```

// eventbus.h 참고
typedef struct {
    EventHead e;
    float tc[NUM_TC]; // NUM_T = 6
} TCM_IN, TcmInEvent;

typedef Event TCM_OUT, TcmOutEvent;

```

TCM 모듈의 입출력이미지는 위와 같습니다. 출력은 없으며, 온도입력 tc 가 6 개 있습니다. 다음은 사용예입니다.

```

11 get in
: 11 tc0 26.0 tc1 -89.5 tc2 -89.5 tc3 -89.5 tc4 -89.5 tc5 -89.5

11 get tc0 tc1
: 11 tc0 26.0 tc1 -89.5

```

#### 4.4.13. LAN, LMO, USB 모듈

```

// eventbus.h 참고
typedef Event LAN_IN,LanInEvent;
typedef Event LAN_OUT,LanOutEvent;

typedef Event LMO_IN,LmoInEvent;
typedef Event LMO_OUT,LmoOutEvent;

typedef Event USB_IN,UsbInEvent;
typedef Event USB_OUT,UsbOutEvent;

```

LAN, LMO, USB 모듈은 입출력이미지에 데이터가 없으므로 AT 명령을 사용하지 않습니다.

#### 4.4.14. USD 모듈

```

// eventbus.h 참고
typedef struct {
    EventHead e;
    union {
        struct {
            ELogic pos      :1;
            ELogic neg      :1;
            ELogic nnp      :1;
            ELogic jcw      :1;
            ELogic jccw     :1;
            ELogic jspd     :1;
            Bool   busy     :1;
            Bool   fan      :1;
            ELogic sw0      :1;
            ELogic sw1      :1;
        };
        uint16 status;
    };
    uint16 jvol;
    int32 loc;
    float vm;
    int16 heat;
} USD_IN,UsdInEvent;

typedef struct {
    EventHead e;
    union {
        struct {
            USD_CMD cmd     :4;
            USD_STEP step   :3;
            USD_DIR dir    :1;
        };
        uint16 config;
    };
    uint16 ipps;
}

```

```
    uint16 accel;
    uint32 pps;
    uint32 move;
} USD_OUT,UsdOutEvent;
```

USD 모듈의 입출력이미지는 위와 같습니다. 다음은 사용예입니다.

```
16 get in
: 16 status 0 jvol 0 loc -2147483648 vm 24.0 heat 26

16 get pos neg nnp jcw jccw jspd busy fan sw0 sw1
: 16 pos 0 neg 0 nnp 0 jcw 0 jccw 0 jspd 0 busy 0 fan 0 sw0 0 sw1 0

16 get jvol loc vm heat
: 16 jvol 0 loc -2147483648 vm 24.0 heat 26

16 get out
: 16 config 96 ipps 100 pps 20000 accel 10 move 100000

16 get cmd step dir
: 16 cmd free step 16 dir cw

16 get ipps accel pps move
: 16 ipps 100 accel 10 pps 20000 move 100000

16 set cmd run step 16 dir ccw ipps 100 accel 10 pps 10000
: 16 ok

16 get busy
: 16 busy 1
```

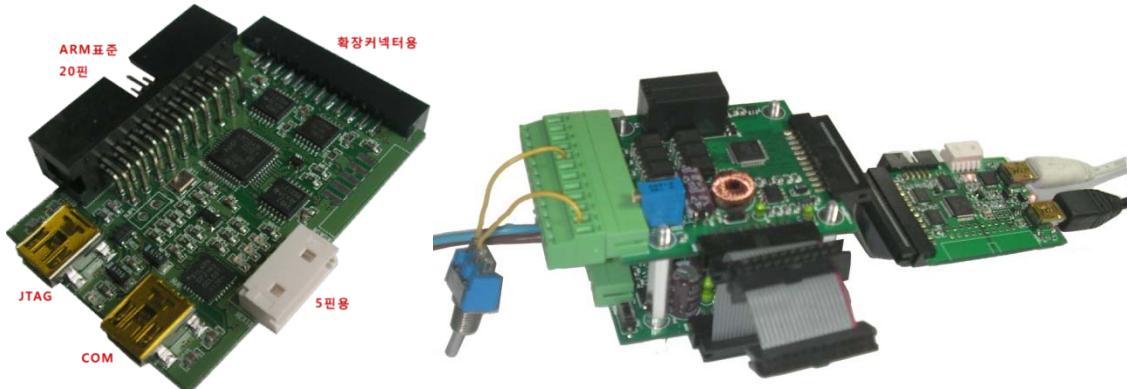
## 5. eventbus\_net

eventbus\_net 을 이용하면 PC 응용프로그램을 매우 쉽게 작성할 수 있습니다. LAN, USB, UUM, U2M 또는 ColinkEx 를 사용하여 PC 와 이지컨트롤러 모듈을 연결하면 eventbus\_net 라이브러리가 컴퓨터와 모듈 사이의 통신을 비롯한 제반사항을 모두 처리해 주므로, 사용자는 이벤트핸들러만 작성하면 됩니다.

### 5.1. 컴퓨터와 연결하는 방법

이지컨트롤러를 컴퓨터와 연결하는 다양한 방법을 제공하므로, 각각의 장단점을 파악하여 응용에 따라 적절한 방법을 선택하십시오. LAN 을 제외한 모듈은 마이컴의 UART 통신포트를 이용합니다. U2M 은 RS232 통신을 사용하며 115,200bps 까지 지원하고, USB 에 연결되는 모듈(UUM, USB, ColinkEx)은 CP2102 칩을 사용하며 최대 921,600bps 까지 지원합니다.(그러나, 이벤트버스의 기본설정은 115,200bps 를 사용합니다)

#### 5.1.1. ColinkEx 를 사용하는 방법



ColinkEx 는 SWD 방식의 JTAG 디버거와 CP2102 를 이용한 버추얼 시리얼 COM 포트를 동시에 제공하는 펌웨어 개발용 다운로더, 디버거, 시리얼통신의 세가지 복합기능을 갖춘 개발툴입니다.

그림의 COM 커넥터를 PC 의 USB 에 연결하고, 24 핀의 확장커넥터 또는 5 핀의 몰렉스 커넥터를 모듈의 확장커넥터에 연결하면 마이컴의 UART 가 PC 에 가상 시리얼포트로 인식됩니다.

이 방법은 개발단계에서 사용하기에 적합한 방법입니다. 디버거와 시리얼통신이 한번에 해결되므로 번거롭게 커넥터를 바꿔 꽂을 필요가 없어서 매우 편리합니다. **펌웨어를 개발하기 위하여는 필수적으로 필요한 제품입니다.**

### 5.1.2. UUB 를 사용하는 방법



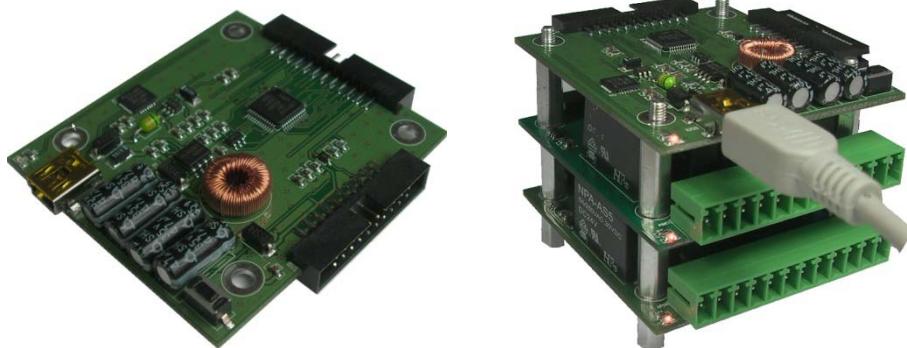
UUB 는 시리얼통신용 브릿지 칩인 CP2102 회로만 넣었으므로 가격이 저렴하고 설치가 간편합니다. 그러므로 개발시 또는 개발후 상시로 연결해 두어야 할 때도 사용할 수 있는 가장 일반적인 연결수단입니다. 하지만, USB 통신케이블에서 공급되는 5V 전원으로 모듈에 동작전원을 공급하는 기능이 없어서 PSM 과 같은 별도의 전원모듈이 있어야 하는 불편함이 있습니다. 이를 보강하기 위하여 UUM 이 새로 출시되었으므로, 향후 개발은 UUM 을 사용해 주십시오. UUB 는 향후 단종예정입니다.

### 5.1.3. UUM 모듈을 사용하는 방법



UUB 모듈의 불편함을 해소하기 위해 새로 개발된 모듈입니다. 컴퓨터의 USB 통신단자에 연결하여 가상 시리얼포트로 사용할 수 있을 뿐만아니라, USB 에서 공급되는 +5V 전원에서 이지컨트롤러의 동작에 필요한 VAA, VBB 전원을 생성하여 연결된 모듈에 공급할 수 있도록 DCDC 컨버터회로가 내장되어 있으므로, 그리 크지 않은 전류를 소모하는 경우에는 PSM 등의 별도전원이 필요치 않은 장점이 있습니다. 또한 터미널블럭으로 VAA, VBB 전원을 출력합니다.

#### 5.1.4. USB 모듈을 사용하는 방법



USB 모듈은 CP2102, DCDC 컨버터외에도 자체 마이컴을 가지고 있습니다. 그러므로 컴퓨터 또는 호스트 USB 를 가지는 임베디드컴퓨터에 연결되는 경우 PSM 과 같은 별도의 전원 없이도 USB 전원으로 연결된 모듈에 VBB(24V)전원을 공급할 수 있습니다.

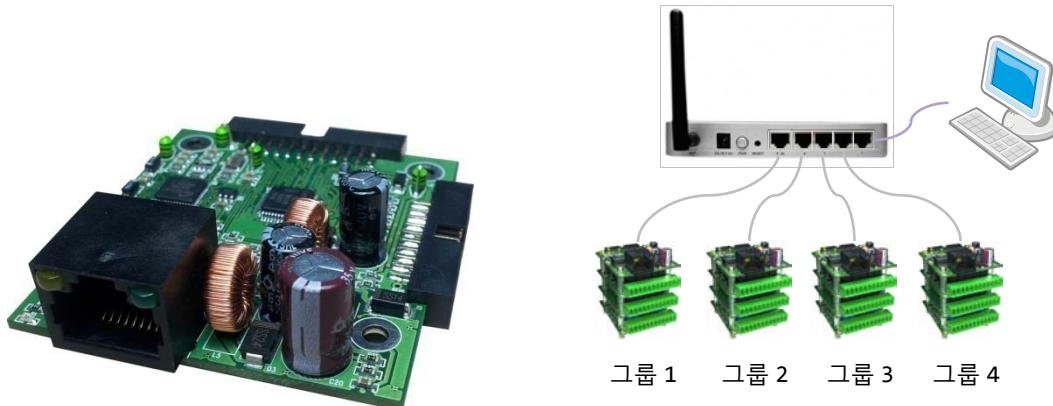
그리고, USB 모듈의 가장 중요한 용도는 자체 마이컴을 내장하고 있으므로, SMD, MOC, USD, SEG 처럼 마이컴에 부하가 심한 모듈의 경우 사용자 프로그램을 추가하면 성능저하의 원인이 되므로, 이경우 USB 모듈을 사용하여 사용자 프로그램을 저장/실행하는데 사용하면 좋습니다.

#### 5.1.5. U2M 모듈을 사용하는 방법



컴퓨터, 임베디드 보드 등에 연결된 RS232 통신포트에 연결하여 이지컨트롤러를 동작시키기 위하여 U2M 을 사용할 수 있습니다. 그림처럼 터미널블럭으로 +24V 를 입력하면 내장된 스텝다운 DCDC 컨버터가 VAA(5V), VBB(24V)를 버스에 연결된 모듈에 공급하므로 PSM 과 같은 별도의 전원 없이도 사용할 수 있습니다. 라인에 24V 전원이 있는 경우 편리하게 사용할 수 있습니다.

### 5.1.6. LAN 모듈을 사용하는 방법



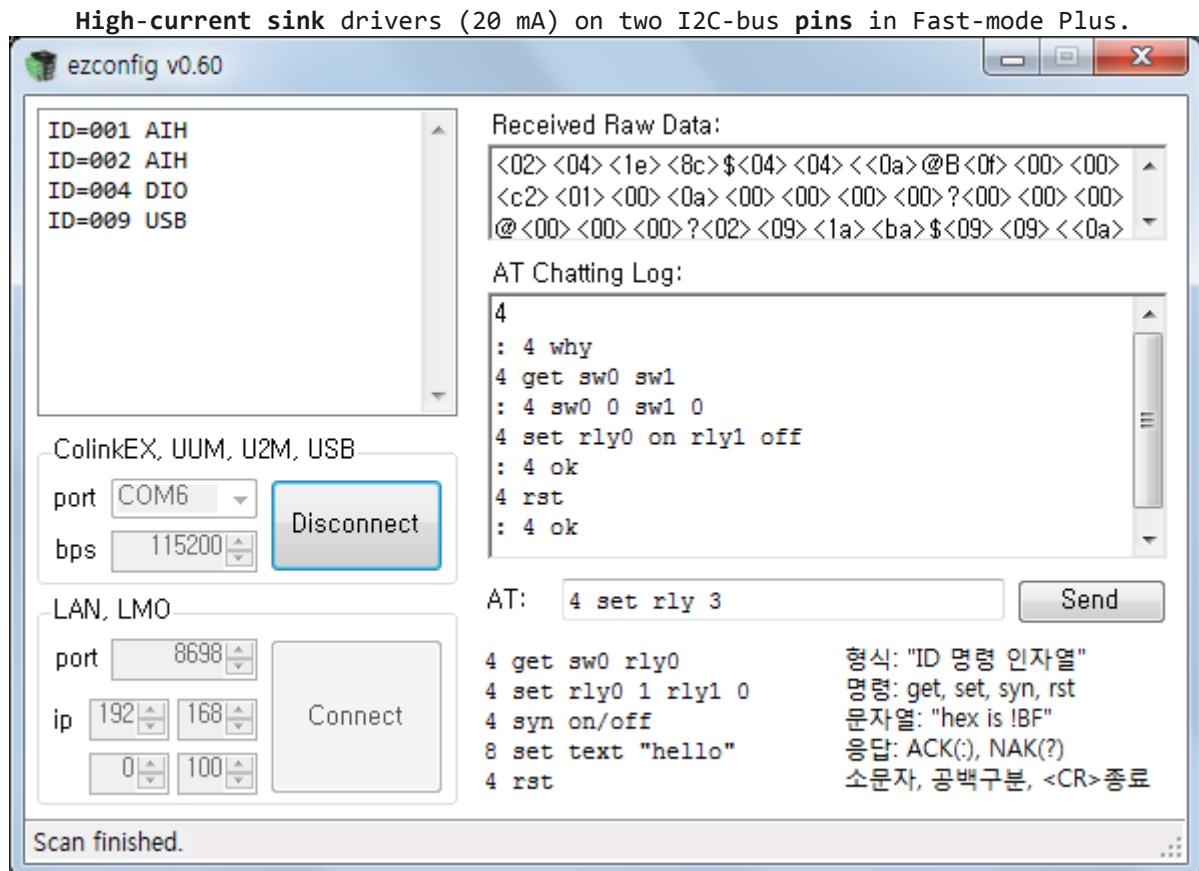
LAN 모듈은 공유기(스위칭허브)에 연결하여 로컬네트워크를 구성할 수 있는 서버모듈입니다. 그러므로, 연결되는 PC 또는 임베디드 컴퓨터는 클라이언트로 동작해서 LAN 모듈에 연결된 다른 모듈들과 이벤트를 교환할 수 있습니다.

LAN 모듈은 TCP/IP 소켓을 8 개까지 제공하므로 자유로운 통신연결이 가능합니다. LAN 모듈은 각 그룹의 모듈의 ID 를 분리시킵니다. 즉, 그룹 1 에 속한 모듈과 그룹 2 에 속한 모듈은 서로 중복된 ID 를 사용할 수 있습니다. 결과적으로 LAN 모듈을 이용하면 연결할 수 있는 모듈 개수가 자유로워집니다.

추가로 LAN 모듈은 **POE** 를 지원하여 원격 전원공급이 가능합니다(POE 지원 스위칭허브 사용시). 그러므로, 전원연결이 어렵거나 많은 모듈을 연결하여야 할 때 사용하면 편리합니다.

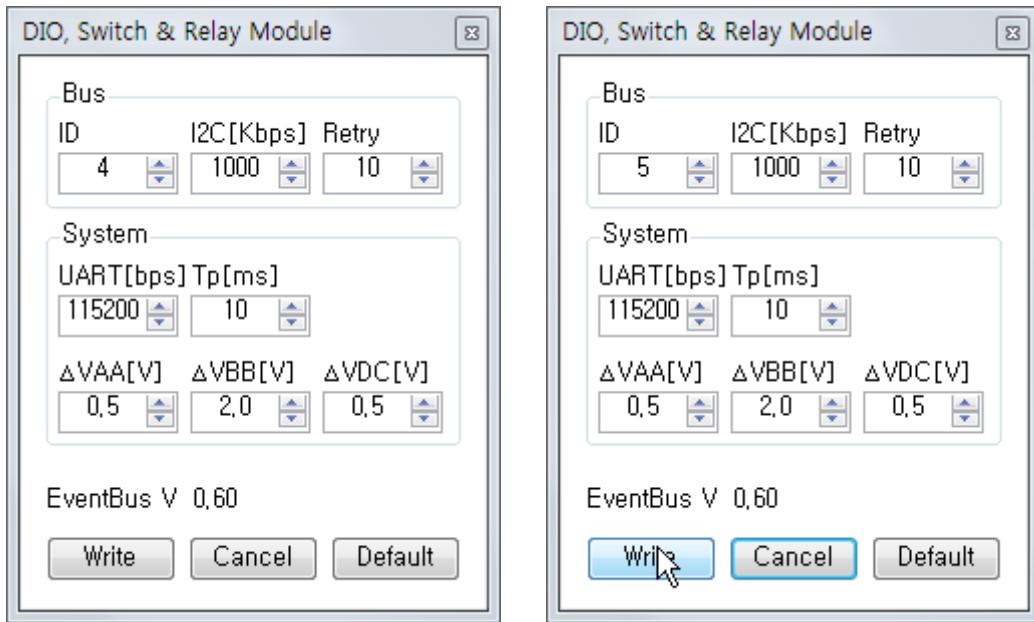
## 5.2. ezconfig.exe 사용가이드

ColinkEX, UUM, U2M, USB, UUB 는 컴퓨터에 연결하면 가상 COM 포트를 생성합니다. 이 가상 COM 포트가 연결되면 ezconfig.exe 를 실행해서 연결된 모듈들의 eeprom 설정을 변경할 수 있습니다. v0.60 이전의 버전에서는 설정하고자 하는 모듈마다 24 핀 확장커넥터에 ColinkEX, UUM, U2M, UUB(USB 제외)를 바꿔가면서 연결해서 하나씩 설정했지만, v0.60 부터는 어느 모듈이라도 한 모듈에만 연결되면 버스케이블로 연결된 모든 모듈들의 설정을 변경할 수 있으므로 매우 편리하게 설정할 수 있습니다.

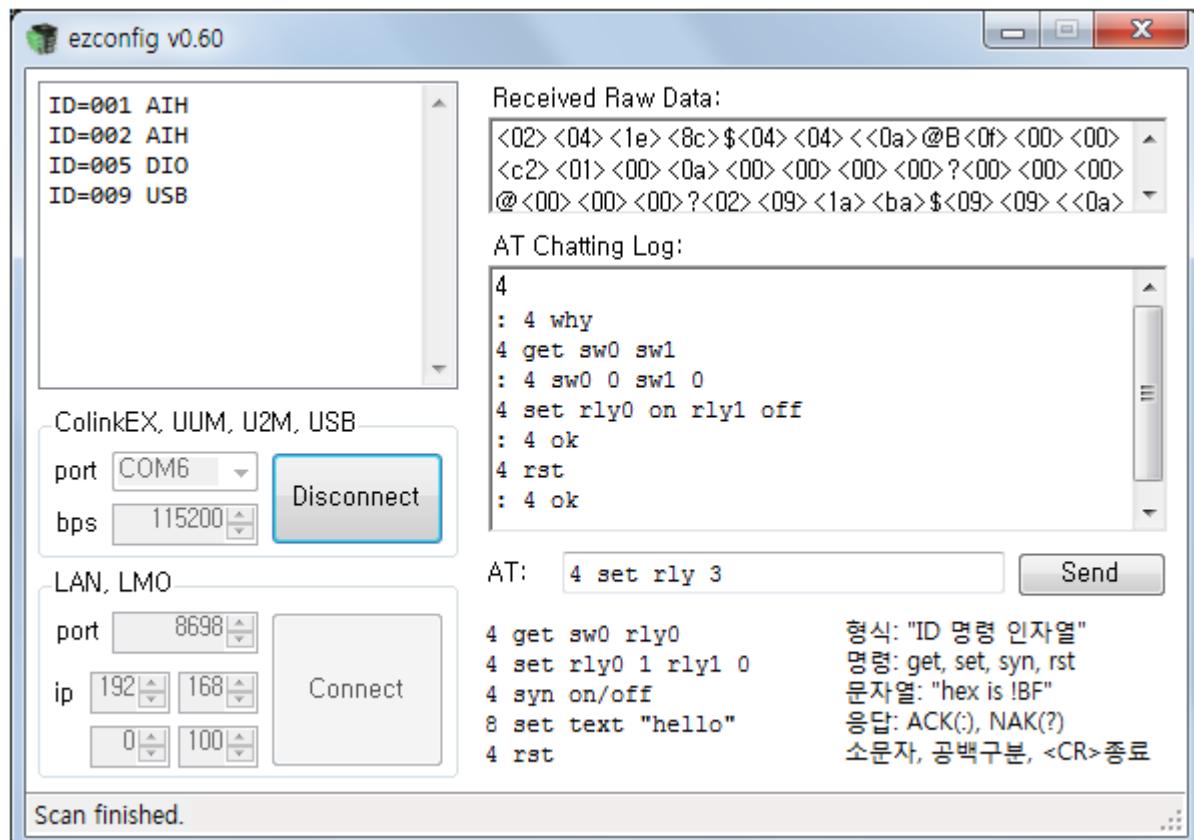


연결된 통신포트를 선택하고 통신속도를 설정(V0.60 부터 115,200bps 가 기본속도)하고 Connect 버튼을 클릭하면 그림처럼 모듈의 ID 를 1 번부터 스캔합니다. 이 경우 AIH(ID=1), AIH(ID=2), DIO(ID=4), USB(ID=9) 네개의 모듈이 버스케이블에 연결되어 있고, USB 모듈에는 USB-MINI 케이블이 컴퓨터와 연결되어 있습니다.

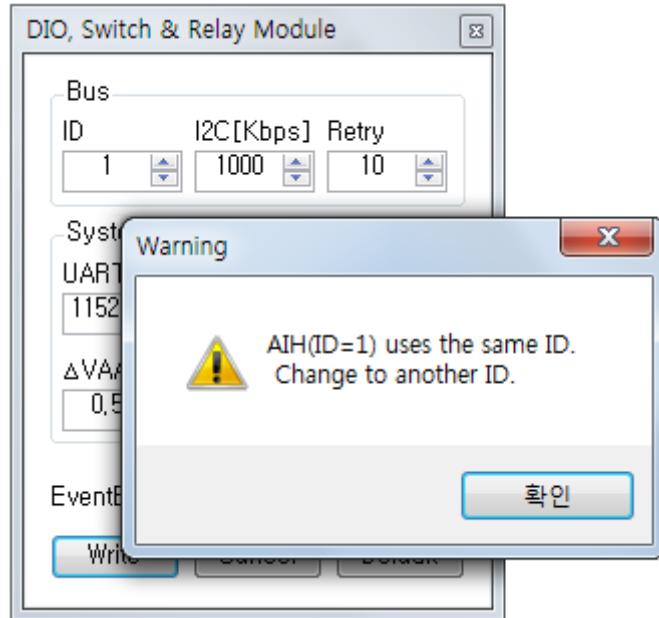
이제 DIO 모듈의 설정을 변경해 보겠습니다. 그림에서 “ID=004 DIO” 항목을 더블클릭합니다. (아래의 상태바에 Scanning...으로 표시됩니다. 스캔이 진행중에도 모듈을 더블클릭해도 됩니다. 스캔이 종료되기를 기다릴 필요가 없습니다.)



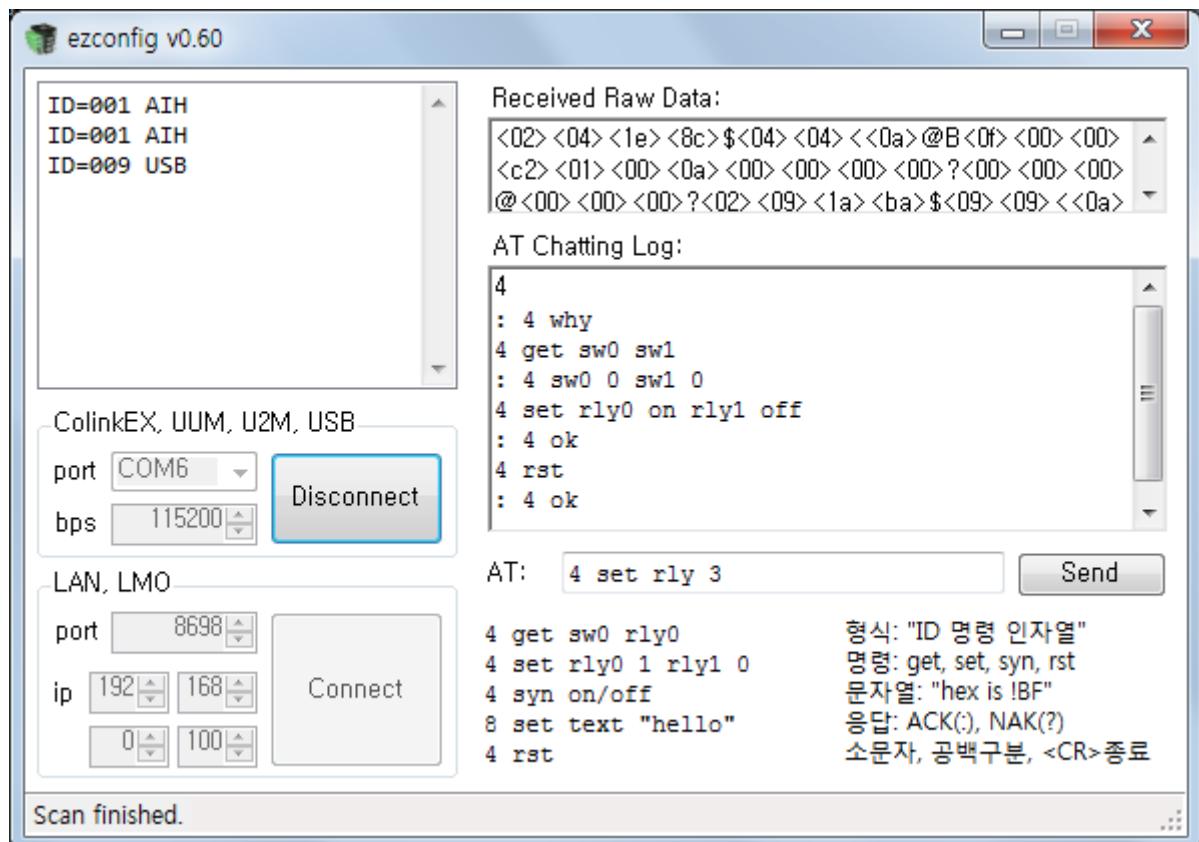
왼쪽 그림처럼 DIO 모듈의 설정창이 열립니다. 변경하려는 항목을 변경후 Write 버튼을 클릭하면 설정이 변경됩니다. ID 를 5 번으로 변경해 보겠습니다. 아래 그림처럼 DIO 의 ID 가 5로 변경된 것을 확인할 수 있습니다.



이번에는 DIO(ID=5)를 이미 사용중인 ID 인 1 번으로 변경해 봅니다.



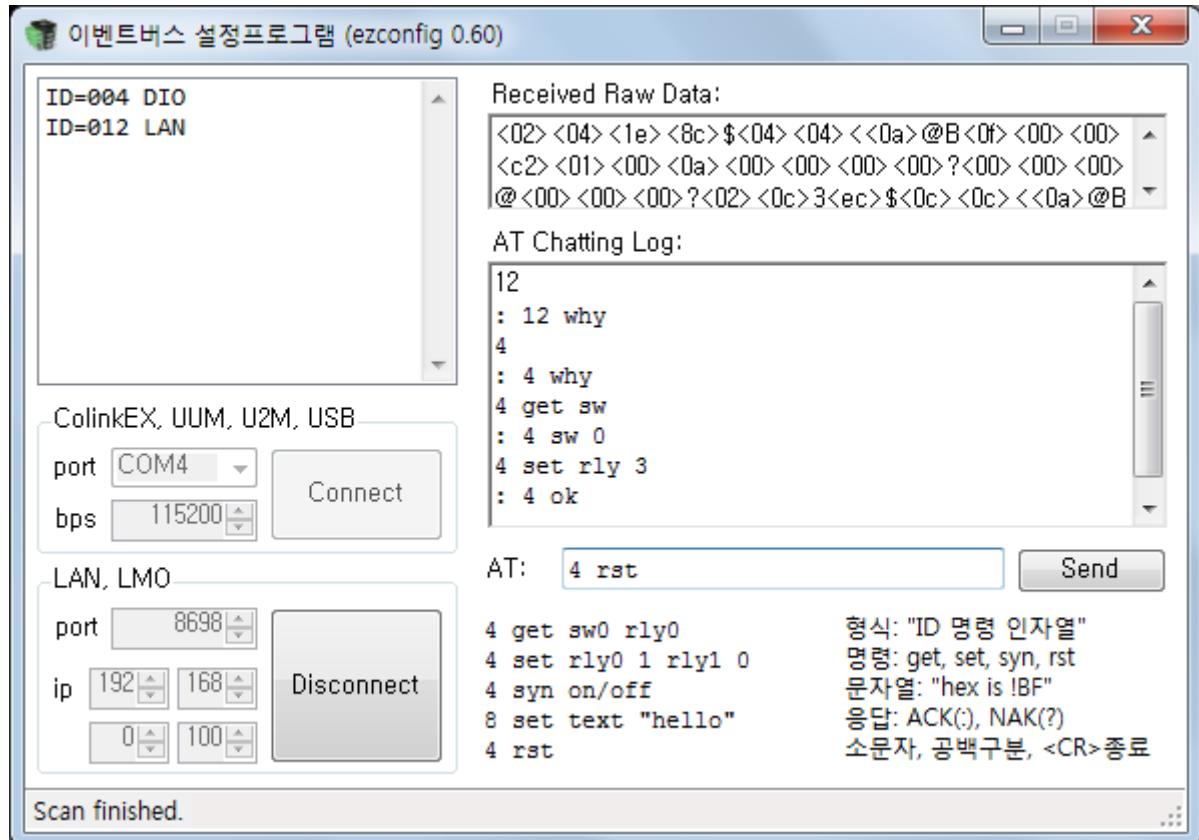
그림처럼 1 번은 이미 사용중인 ID임을 알려줍니다.



### \*\*주의\*\* 동일 ID 를 연결하지 말 것

ezconfig.exe 는 ID 로 연결된 모듈을 구분합니다. 그러므로 동일한 ID 가 있으면 정상적으로 구별할 수 없습니다. 만일, 처음 2 개의 동일한 모듈을 구입했다면, 그림처럼 ID 가 동일하게 설정되어 있습니다. 이 상태에서 어느 한쪽 모듈의 ID 를 변경하면 다른 쪽 ID 도 동일하게 변경되므로 이 상태로는 변경할 수 없습니다. 이럴 때는 한 개의 모듈만 연결한 후 ID 를

사용하지 않는 다른 ID로 바꾼 후 나머지 모듈을 연결하면 됩니다(모듈 연결시는 반드시 전원을 끈 후 연결하십시오). 위의 경우 AIH 모듈 한개만 연결하고 2 번으로 변경후 다시 연결하면 됩니다.



이번에는 LAN 모듈이 연결된 경우에 ezconfig.exe 를 사용해서 설정을 변경하는 방법입니다. LAN 모듈은 출고시 기본으로 IP 주소가 192.168.0.100 으로 설정되어 있습니다.

주소를 알기위해서는 여러분의 공유기에 접속하면 주소체계를 확인할 수 있습니다. 공유기의 주소는 “시작메뉴>보조프로그램>명령프롬프트”를 실행하고 ipconfig 를 입력하면 기본 게이트웨이 주소를 알 수 있습니다.

```
C:\Users\JamesDean>ipconfig

Windows IP 구성

무선 LAN 어댑터 무선 네트워크 연결:
  미디어 상태 . . . . . : 미디어 연결 끊김
  연결별 DNS 접미사 . . . . : local

이더넷 어댑터 로컬 영역 연결:
  연결별 DNS 접미사 . . . . : local
  링크-로컬 IPv6 주소 . . . . : fe80::c7e0:4ff%14
  IPv4 주소 . . . . . : 192.168.0.18
  서브넷 마스크 . . . . . : 255.255.255.0
  기본 게이트웨이 . . . . . : 192.168.0.1
```

기본게이트웨이 주소는 192.168.0.1 입니다.



게이트웨이 주소를 검색기 주소창에 입력하면 공유기의 관리창이 열립니다. 여기서 관리도구를 선택하면 네트워크에 할당된 주소체계를 확인할 수 있습니다.

**\*\*주의\*\*** 만일 주소가 192.168.0 으로 시작하지 않는다면 LAN 으로 접속할 수 없습니다. 이경우는 LAN 의 IP 를 변경해야 하므로 UART 로 접속하는 ColinkEx, UUM, U2M 으로 연결하고 ezconfig.exe 를 실행한 후 우선 LAN의 IP 주소부터 바꾸어야 합니다.

예를 들어, 여러분의 공유기의 게이트웨이 주소가 192.168.10.1 이라면 ezconfig.exe 에서 LAN 모듈의 Gateway 주소를 192.168.10.1 로 변경하고, ip 주소도 192.168.10.xxx 처럼 변경하면 됩니다.

주의할 점은 ip 주소가 자동할당(DHCP)되는 주소영역이면 다른 인터넷기기가 사용해 버릴 수도 있으므로 자동할당 되지 않도록 설정하면 좋습니다. 이것은 위의 관리도구에서 “동적 IP 주소 범위”(제조사마다 명칭이 다를 수도 있음)를 설정하면 됩니다. 예를 들어 범위를 192.168.10.2 ~ 192.168.10.100 으로 한다면, 100 번까지는 자동으로 할당되므로 주소가 변동될 수 있지만, 101~255 번 까지의 주소는 자동할당 되지 않으므로 항상 LAN 모듈에 할당되어 안정하게 사용할 수 있습니다.

## 6. 모듈의 종류

이지컨트롤러의 설계컨셉은 다음과 같습니다.

- 케이블만으로 기능을 쉽게 확장할 수 있어야 한다.
- 하드웨어 및 C 언어 초급자도 제어시스템을 구현할 수 있어야 한다.
- PC 프로그래머도 하드웨어를 제어할 수 있어야 한다.
- 가격이 저렴해야 한다.
- 번거로운 추가 외부장치를 최소화 한다.
- 컴팩트해서 설치가 쉬워야 한다.
- 개발 후 양산보드로 제작할 수 있어야 한다.

이지컨트롤러는 현재 20 여종의 다양한 제품이 제공되고 있으며, 향후 지속적으로 추가할 예정입니다. 현재 제공되는 모듈은 다음과 같습니다.

| 모듈       | 주 기능                                   | 추가 기능                               |
|----------|----------------------------------------|-------------------------------------|
| AIM      | 4 채널 아날로그 입력                           | VDC 전원출력                            |
| AOM      | 2 채널 전압출력, 2 채널 전압/전류출력                | VDC 전원출력, 4~20mA 전류출력               |
| DIE      | 16 채널 DC, AC 전압입력                      | 무극성, AC 입력                          |
| DIM      | 10 채널 스위치 접점입력                         | VDC 전원출력                            |
| DIO      | 6 채널 스위치 접점입력, 2 채널 릴레이 출력             | VDC 전원출력                            |
| DOE      | 16 채널 릴레이 출력                           |                                     |
| DOM      | 6 채널 릴레이 출력                            |                                     |
| LAN      | TCP/IO 소켓통신 서버                         | POE 원격 전원공급, 소켓 8 개 지원              |
| LMO      | MODBUS TCP/IP 모듈                       | LAN 모듈과 동일 하드웨어 사용                  |
| MOC      | 1 축 모션제어(위치제어) 펄스출력                    | 조그, 리imits위치 입력                     |
| MSB      | RS232 시리얼 멀티포트                         | 4 포트 입출력, 최대 120Kbps                |
| PSM      | 상용전원 시스템 전원공급(5V, 24V)                 | 220VAC 입력, 리셋 버튼                    |
| SMD      | 2 채널 스텝모터 구동, 16 분주 미세각 구동             | 가감속 프로파일 내장, 온도센서, 자동 속도제어 냉각팬, 방열판 |
| TCM      | 6 채널 써모커플 온도측정                         | ICE 센서내장, K,T,E 타입 지원               |
| ENV      | 환경센서 (온도, 조도, 소리, 인체감지)                |                                     |
| SEG      | 16 세그먼트 영문, 숫자, 특수문자 표시                | 주변밝기 인식용 CDS                        |
| ColinkEx | Serial Wired Debugger (Serial JTAG)    | USB-UART 가상 COM 포트제공                |
| UPS      | 배터리에 의한 전원공급(5V, 24V)                  | 충전회로, 과방전 방지회로                      |
| USB      | PC 연결, 펌웨어 저장공간 제공                     | VAA,VBB 전원공급                        |
| UUM      | UART-USB 브릿지                           | 전원회로 내장, USB 전원사용                   |
| U2M      | UART-RS232 브릿지                         | 전원회로 내장, 24V 입력                     |
| USD      | Unipolar Step-Motor Driving Unit (1 축) | 조그, 리밋센서, 사용자용 스위치입력                |
| BPU      | 버스 풀업, 통신거리 확장, 버스폭 변환                 | 고속, 단거리 통신용                         |

|     |                           |                       |
|-----|---------------------------|-----------------------|
| EXP | 모듈의 I/O 확장 및 사용자회로 추가용    | 8, 16 핀 SOIC 부품용 패드   |
| AIH | 4 채널 24 비트 아날로그입력, 전류출력   | 정밀전압, 저항, RTD 측정용     |
| RCS | 8 채널 RC 서보모터용 제어기         | 1us 단위의 정밀 PWM 펄스출력   |
| LMO | HMI 등에 연결할 수 있는 모드버스 통신모듈 | HMI, MMI 에 슬레이브로 연결가능 |
| PWR | 5V, 24V 전원을 연결하는 전원커넥터 모듈 | 저가                    |

## 6.1. AIH, Analog Input High-resolution Module

4 채널의 아날로그 전압을 차동입력으로 매우 정밀하게 읽을 수 있으며, 저항형 센서의 측정에 사용할 수 있는 2 채널의 전류를 출력할 수 있습니다. 각 채널은 차동으로 설계되어 있으므로 신호원과 모듈사이의 라인드롭이 있는 경우에도 정밀하게 입력을 읽을 수 있습니다. 최대 입력전압은 10VDC(분해능 1[ $\mu$ V], 24 비트 ADC)까지 입력받을 수 있으며 역전압은 (-)부호로 읽힙니다(주의, 음전압이 아님). 채널당 1.25~80SPS 까지의 고속 샘플링을 지원합니다. 내부 ADC 는 출고시 교정되어 있으므로 구입 즉시 사용할 수 있으며, 원활 경우 사용자가 직접 교정할 수 있습니다. 각각의 입력은 외부에 저항을 추가하면 더 높은 전압을 입력받을 수 있습니다.

그리고, 2 채널의 가변전류 출력을 제공합니다. 단, 채널별로 전류를 별도로 설정할 수는 없으며, 동일한 전류가 출력됩니다. 전류출력을 사용하면 RTD, 저항측정 등의 다양한 응용측정에 사용할 수 있습니다.

IO 커넥터 핀사양

| 핀번호 | 명 칭   | 기 능        | 기 타                                             |
|-----|-------|------------|-------------------------------------------------|
| 1   | VIN0+ | 0~10V 전압입력 | 분해능 약 1[ $\mu$ V].                              |
| 2   | VIN0- |            | 출고시 교정되어 있음.                                    |
| 3   | VIN1+ | 0~10V 전압입력 | 역전압 측정.                                         |
| 4   | VIN1- |            | 외부저항 추가시 입력범위 확대가능.                             |
| 5   | VIN2+ | 0~10V 전압입력 |                                                 |
| 6   | VIN2- |            |                                                 |
| 7   | VIN3+ | 0~10V 전압입력 |                                                 |
| 8   | VIN3- |            |                                                 |
| 9   | IOUT  | 전류출력       | 0, 50, 100, 250, 500, 750, 1000, 1500[ $\mu$ A] |
| 10  | IOUT  |            |                                                 |
| 11  | GND   |            |                                                 |
| 12  | GND   |            |                                                 |

확장커넥터 핀사양

| 핀  | 핀 명칭                       | 핀 기능                                                      |
|----|----------------------------|-----------------------------------------------------------|
| 1  | P2.0/SSEL1                 | GPIO / SPI1 Slave Select                                  |
| 2  | P2.1/SCK1                  | GPIO / SPI1 Serial Clock                                  |
| 3  | P2.2/MISO1                 | GPIO / SPI1 Master Input Slave Output                     |
| 4  | P2.3/MOSI1                 | GPIO / SPI1 Master Output Slave Input                     |
| 5  | P2.4                       | GPIO                                                      |
| 6  | P2.5                       | GPIO                                                      |
| 7  | P2.6                       | GPIO                                                      |
| 8  | P2.7                       | GPIO                                                      |
| 9  | P2.8                       | GPIO                                                      |
| 10 | P2.9                       | GPIO                                                      |
| 11 | P2.10                      | GPIO                                                      |
| 12 | P2.11                      | GPIO                                                      |
| 13 | P1.1/AD2/CT32B1_MAT0       | GPIO/ADC2 / Match output 0 for 32-bit timer 1             |
| 14 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1             |
| 15 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1           |
| 16 | VAA                        | +5V Ouptpu                                                |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC 4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.       |

|    |                         |                                                      |
|----|-------------------------|------------------------------------------------------|
| 19 | SWCLK/P0.10/CT16B0_MAT2 | SWD clock / GPIO / Match output 2 for 16-bit timer 0 |
| 20 | RXD/P1.6/CT32B0_MAT0    | UART RX / GPIO / Match output 0 for 32-bit timer 0   |
| 21 | VCC                     | VCC(3.3VDC)로 직용 전원 출력                                |
| 22 | GND                     |                                                      |
| 23 | VBB                     | VBB(24VDC) 전원 출력                                     |
| 24 | GND                     |                                                      |

```
typedef struct {
    MType        mtype;
    ID          id;
    AihInEvent*  in;
    AihOutEvent* out;
} AIH;
```

AIH 모듈의 이미지는 위와 같은 AIH 구조체로 되어 있습니다(eventbus.h 참조). mtype은 모듈의 타입으로 Aih(0x12) 타입이 됩니다. id는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력이미지는 AihInEvent 구조체에, 출력이미지는 AihOutEvent 구조체에 저장됩니다.

```
typedef struct {
    EventHead  e;
    float      vin[4];
} AihInEvent;
```

AIH 모듈은 4개의 아날로그 전압입력이 있습니다.

예) 0번 채널의 전압을 읽으려면,  
`if (this->in->vin[0] > 2.5) ...`

```
typedef enum {UA0,UA50,UA100,UA250,UA500,UA750,UA1000,UA1500} AIH_UA;
typedef struct {
    EventHead  e;
    AIH_UA   uamp;
} AihOutEvent;
```

AIH 모듈은 전류출력을 위한 uamp가 있습니다. 여기에 설정한 전류는 IOUT0,IOUT1에 출력됩니다.

```
typedef struct {
    __BASE_PARAM;
    float      dvin[4];
    float      slope[4];
    float      intercept[4];
    uint8     vinMax[4];
    uint8     sps;    // 0(5sps, 1.25cycle) ~ 6(320sps, 80cycle)
} AihParamEvent, AihParamChangedEvent, AihParamNotifiedEvent;
```

이 구조체는 EEPROM에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezConfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(aih\_drv.c)에 정의된 param 변수를 사용해서 param.dVin[0]처럼 읽을 수 있습니다.

- `__BASE_PARAM`  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고

■ dVin

입력전압에 변동이 있으면 이벤트버스를 통하여 다른 모듈이나 컴퓨터로 싱크하게 되는데, 이때 전압이 변동하였는지를 판단하는 기준으로 dVin 보다 크게 변하면 변동한 것으로 간주하고 AimIn 이벤트를 발생합니다.

■ slope, intercept

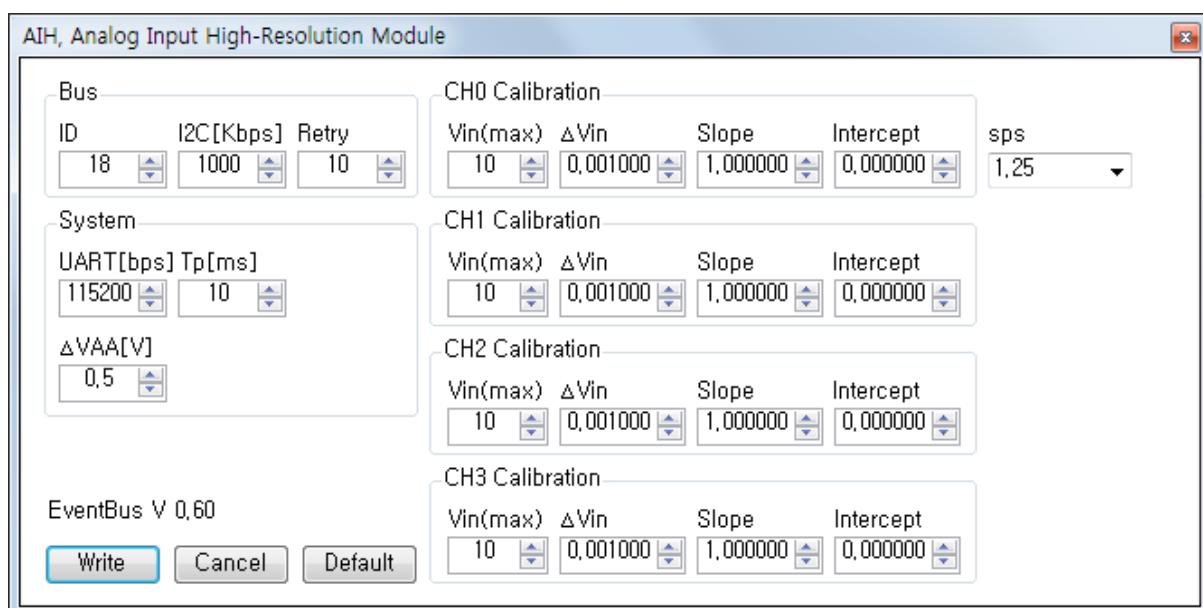
입력전압 교정에 사용되는 파라미터입니다.

■ vinMax

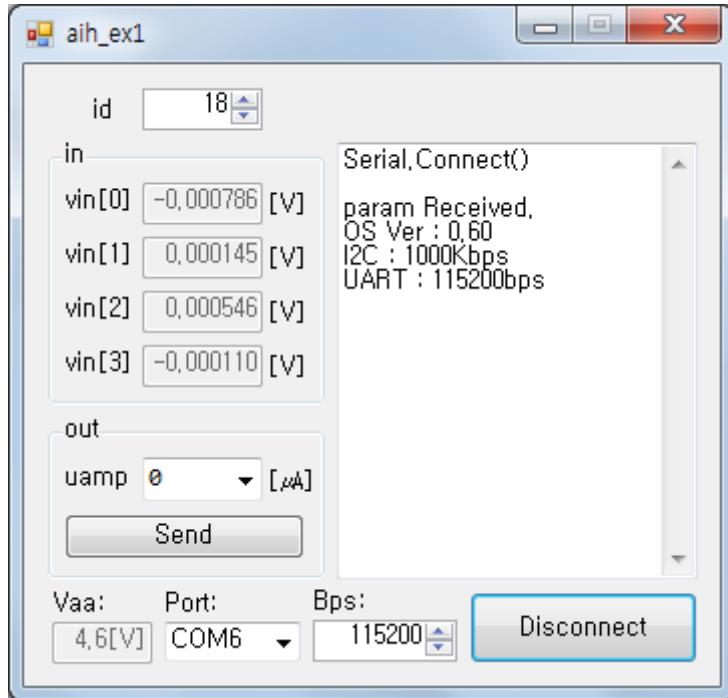
최대입력전압의 범위를 설정합니다. 외부저항을 추가해서 입력전압 범위를 확장한 경우 이 값을 적절히 설정해야 합니다.

■ sps

ADC의 샘플링 속도를 설정합니다. 0은 5sps(4채널이므로 채널당 1.25sps)를 6은 320sps(채널당 80sps)로 동작합니다. 저속으로 설정하면 정밀도가 향상되며, 고속으로 읽으면 정밀도는 떨어지게 됩니다. 가능하면 저속으로 사용하는 것이 좋습니다.



`ezconfig.exe`의 설정창입니다. AIH 모듈을 ColinkEx 또는 UUM,U2M으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 AihParamEvent와 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM에 영구 저장됩니다. V0.60부터 컴퓨터와 연결속도를 `ezconfig.exe`에서 설정할 수 있으며, 기본속도는 115,200bps입니다.



`eventbus_net_0.60\example\aih_ex1\bin\Debug\aih_ex1.exe` 예제프로그램을 실행하면 그림처럼 AIH 를 테스트할 수 있습니다. 연결된 Port 를 선택하고, Bps 는 기본적으로 설정된 115,200bps 를 사용합니다. V0.50 이후부터 Vaa 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(aih\_def.c 참고). 입력전압이 ezconfig.exe 에서 설정한 dVin 보다 크게 변하면 전압표시가 갱신됩니다.

### 6.1.1. 입력전압 교정법

출고시 입력전압은 교정되어 있지만, 필요에 따라서는 사용자가 교정할 수 있습니다. 교정은 ezconfig.exe 를 사용하여 AIH 모듈에 접속하여 Vin(max), Slope, Intercept 값을 바꾸면 됩니다. Slope, Intercept 를 쉽게 계산할 수 있도록 엑셀파일이 제공되므로 카페에서 다운로드해서 사용하십시오. (카페의 다운로드에서 “AIM 교정” 엑셀파일 참고).

## 6.2. AIM, Analog Input Module



4 채널의 아날로그 전압을 입력받을 수 있습니다. 각 채널은 차동으로 설계되어 있으므로 한쪽 입력이 반드시 그라운드일 필요는 없으며 입력전압보다 큰 공통모드전압을 허용하므로 입력이 모듈과 떨어져 있어도 정밀하게 입력을 읽을 수 있습니다. 출고시 최대 입력전압은 10VDC(분해능 10mV)까지 입력받을 수 있으며, 내부 ADC 는 출고시 교정되어 있으므로 구입 즉시 사용할 수 있으며, 원활 경우 사용자가 직접 교정할 수 있습니다. 각각의 입력은 외부에 저항을 추가하면 더 높은 전압을 입력받을 수 있습니다.

추가적으로 VDC 가변전원을 출력으로 제공하는데, 이는 버스의 VBB(24V) 전원에서 DCDC 변환기를 통하여 출력되는 전원이므로 큰 용량의 전원을 공급할 수는 없지만, 외부에 장착되는 센서 또는 장치등에 전원을 공급하기 위해 사용될 수 있습니다. 더우기 그림에서 보듯이 가변저항으로 출력전압을 조정할 수 있으므로 매우 다양한 기기에 응용할 수 있습니다.

또한, 4 개의 LED 는 입력전압의 크기에 비례하는 밝기로 켜지므로, 현재 입력되는 전압의 크기를 쉽게 모니터링 할 수 있습니다.

IO 커넥터 핀사양

| 핀번호 | 명칭    | 기능         | 기타                                               |
|-----|-------|------------|--------------------------------------------------|
| 1   | VIN0+ | 0~10V 전압입력 | 분해능 10mV.                                        |
| 2   | VIN0- |            | 출고시 교정되어 있음.                                     |
| 3   | VIN1+ | 0~10V 전압입력 | 외부저항 추가시 입력범위 확대 가능.                             |
| 4   | VIN1- |            |                                                  |
| 5   | VDC   | 외부장치용 전압출력 | 11,12 번 핀과 연결되어 있음.                              |
| 6   | GND   |            |                                                  |
| 7   | VIN2+ | 0~10V 전압입력 | 분해능 10mV.                                        |
| 8   | VIN2- |            | 출고시 교정되어 있음.                                     |
| 9   | VIN3+ | 0~10V 전압입력 | 외부저항 추가시 입력범위 확대 가능.                             |
| 10  | VIN3- |            |                                                  |
| 11  | VDC   | 외부장치용 전압출력 | 전류: 3V(350mA), 5V(320mA), 12V(240mA), 18V(120mA) |
| 12  | GND   |            |                                                  |

확장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5 | P2.4       | GPIO                                  |
| 6 | P2.5       | GPIO                                  |
| 7 | P2.6       | GPIO                                  |
| 8 | P2.7       | GPIO                                  |
| 9 | P2.8       | GPIO                                  |

|    |                            |                                                           |
|----|----------------------------|-----------------------------------------------------------|
| 10 | P2.9                       | GPIO                                                      |
| 11 | P2.10                      | GPIO                                                      |
| 12 | P2.11                      | GPIO                                                      |
| 13 | P1.8/CT16B1_CAP0           | GPIO / Capture input 0 for 16-bit timer 1                 |
| 14 | P1.9/CT16B1_MAT0           | GPIO / Match output 0 for 16-bit timer 1                  |
| 15 | P3.4                       | GPIO                                                      |
| 16 | P3.5                       | GPIO                                                      |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC 4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.       |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0      |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0        |
| 21 | VCC                        | VCC(3.3VDC)로직용 전원출력                                       |
| 22 | GND                        |                                                           |
| 23 | VBB                        | VBB(24VDC)전원출력                                            |
| 24 | GND                        |                                                           |

```
typedef struct {
    MType      mtype;
    ID         id;
    AimInEvent* in;
    AimOutEvent* out;
} AIM;
```

AIM 모듈의 이미지는 위와 같은 AIM 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Aim(0x01) 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력이미지는 AimInEvent 구조체에, 출력이미지는 AimOutEvent 구조체에 저장됩니다. (V0.60 부터 AimInChangedEvent,AimOutChangedEvent 가 간략히 AimInEvent,AimOutEvent 로 명칭변경됨)

```
typedef struct {
    EventHead e;
    float     vin[4];
} AimInEvent;
```

AIM 모듈은 4 개의 아날로그 전압입력이 있습니다.

예) 0 번 채널의 전압을 읽으려면,  
`if (this->in->vin[0] > 2.5) ...`

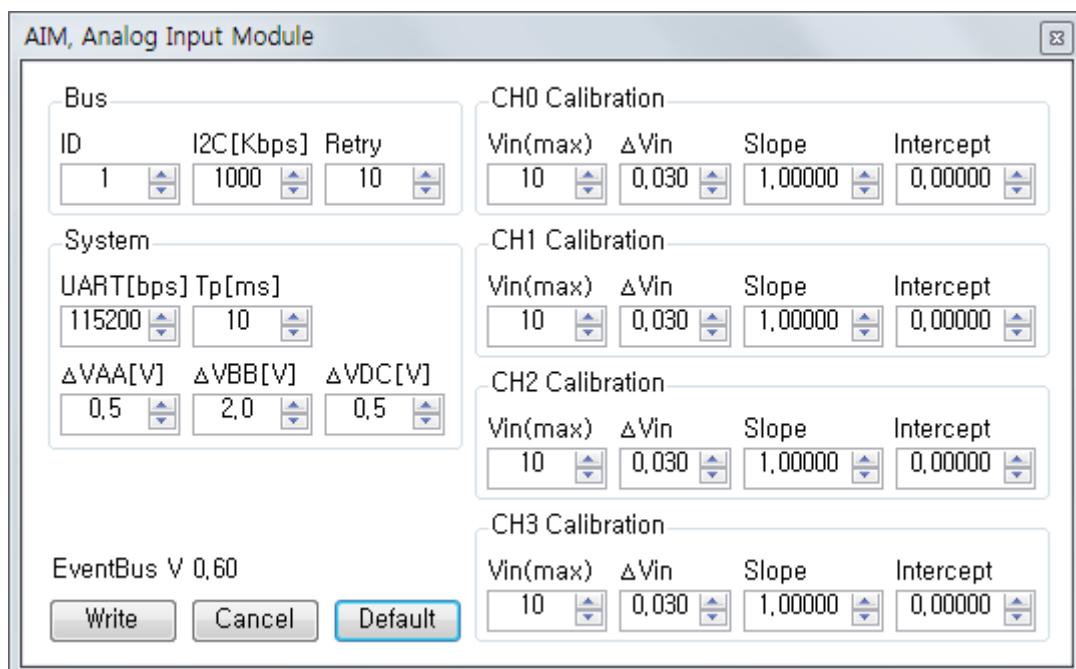
```
typedef Event AimOutEvent;
```

AIM 에는 별다른 출력이 없으므로 기본 이벤트구조체 Event 를 재정의해서 사용합니다.

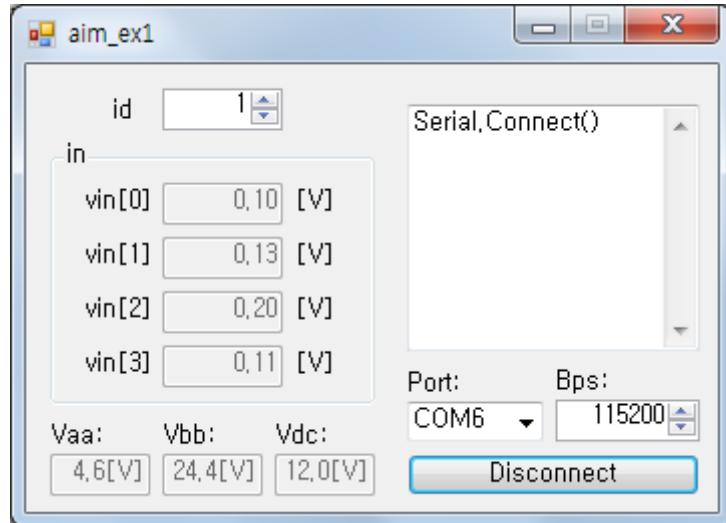
```
typedef struct {
    __BASE_PARAM;
    float      dVBB;
    float      dVDC;
    float      dVin[4];
    float      slope[4];
    float      intercept[4];
    uint8     vinMax[4];
} AimParamEvent,AimParamChangedEvent,AimParamNotifiedEvent;
```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezConfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(aim\_drv.c)에 정의된 param 변수를 사용해서 param.dVBB 와 같이 읽을 수 있습니다.

- \_\_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- dVBB, dVDC  
\_\_BASE\_PARAM 의 dVAA 참고.
- dVin  
입력전압에 변동이 있으면 이벤트버스를 통하여 다른 모듈이나 컴퓨터로 싱크하게 되는데, 이때 전압이 변동하였는지를 판단하는 기준으로 dVin 보다 크게 변하면 변동한 것으로 간주하고 AimIn 이벤트를 발생합니다.
- slope, intercept  
입력전압 교정에 사용되는 파라미터입니다.
- vinMax  
최대입력전압의 범위를 설정합니다. 외부저항을 추가해서 입력전압 범위를 확장한 경우 이 값을 적절히 설정해야 합니다.



ezconfig.exe 의 설정창입니다. AIM 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 AimParamEvent 와 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 ezconfig.exe 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.



`eventbus_net_0.60\example\aim_ex1\bin\Debug\aim_ex1.exe` 예제프로그램을 실행하면 그림처럼 AIM 을 테스트할 수 있습니다. 연결된 Port 를 선택하고, Bps 는 기본적으로 설정된 115,200bps 를 사용합니다. V0.50 이후부터 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(aim\_def.c 참고). 입력전압이 ezconfig.exe 에서 설정한  $\Delta$  Vin 보다 크게 변하면 Analog Input 의 전압표시가 갱신됩니다.

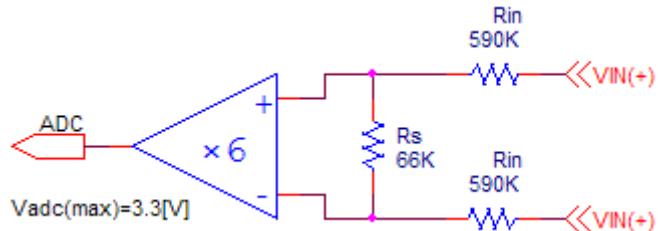
### 6.2.1. 입력전압 교정법

출고시 입력전압은 교정되어 있지만, 필요에 따라서는 사용자가 교정할 수 있습니다. 교정은 ezconfig.exe 를 사용하여 AIM 모듈에 접속하여  $Vin(max)$ , Slope, Intercept 값을 바꾸면 됩니다. Slope, Intercept 를 쉽게 계산할 수 있도록 아래와 같은 엑셀파일이 제공되므로 카페에서 다운로드해서 사용하십시오. (카페의 다운로드에서 “AIM 교정” 엑셀파일 참고).

|    | A                                                                           | B       | C       | D        | E                | F       | G         | H |
|----|-----------------------------------------------------------------------------|---------|---------|----------|------------------|---------|-----------|---|
| 1  |                                                                             |         |         |          |                  |         |           |   |
| 2  |                                                                             |         |         |          | [V0.50] AIM 교정방법 |         |           |   |
| 3  |                                                                             |         |         |          |                  |         |           |   |
| 4  |                                                                             |         |         |          |                  |         |           |   |
| 5  |                                                                             | DVM1[V] | DVM2[V] | AIM1[V]  | AIM2[V]          | Slope   | Intercept |   |
| 6  | CH0                                                                         | 25.000  | 1.000   | 24.60000 | 1.05000          | 1.01911 | -0.07006  |   |
| 7  | CH1                                                                         | 9.996   | 0.251   | 9.75000  | 0.38000          | 1.04006 | -0.14462  |   |
| 8  | CH2                                                                         | 9.996   | 0.251   | 9.83000  | 0.35000          | 1.02800 | -0.10920  |   |
| 9  | CH3                                                                         | 9.996   | 0.251   | 9.86000  | 0.36000          | 1.02583 | -0.11870  |   |
| 10 |                                                                             |         |         |          |                  |         |           |   |
| 11 | 1. 2개의 기준전압(높은전압, 낮은전압)을 준비한다.                                              |         |         |          |                  |         |           |   |
| 12 | * AOM의 출력전압을 사용해도 된다.                                                       |         |         |          |                  |         |           |   |
| 13 | * 최대입력의 90%와 10% 정도의 전압이 좋다.                                                |         |         |          |                  |         |           |   |
| 14 | 2. ezconfig.exe에서 교정할 채널의 $Vin(max)$ =최대입력전압, Slope=1, Intercept=0 으로 설정한다. |         |         |          |                  |         |           |   |
| 15 | * 전채널을 교정한다면 “Default”, “Write” 버튼으로 초기화 한다.                                |         |         |          |                  |         |           |   |
| 16 | 3. aim_ex1 예제프로그램을 실행한다.                                                    |         |         |          |                  |         |           |   |
| 17 | * 입력전압을 DVM으로 읽어서 위 표의 DVM1, DVM2에 기록한다.                                    |         |         |          |                  |         |           |   |
| 18 | * 프로그램에서 읽은 값은 위 표의 AIM1, AIM2에 기록한다.                                       |         |         |          |                  |         |           |   |
|    | 4. Slope, Intercept를 ezconfig.exe에 써 넣고 [Write] 버튼으로 저장한다.                  |         |         |          |                  |         |           |   |

### 6.2.2. 입력전압 범위를 넓히는 방법

AIM 모듈의 VIN+와 VIN- 핀 사이에 입력할 수 있는 전압의 범위는 0~10[V]가 최대입니다. 그 이상의 전압은 내부 ADC에서 인식하지 못하고 포화되어 최대값으로 표시됩니다. 그러나, 응용에 따라서는 더 높은 전압을 읽어야 할 경우가 있습니다. 이때 외부에 2개의 저항을 사용하면 간단히 입력전압의 범위가 넓어집니다.



그림은 AIM의 입력을 간략화한 회로입니다. VIN+와 VIN-의 입력전압을 저항으로 낮추어 ADC로 보냅니다.

그런데, ADC는 최대 3.3[V]까지만 인식하므로, 입력전압이 이 값을 넘어서면 포화되어 최대값으로 표시되는 것입니다.

위의 회로를 식으로 나타내면,

$$(식 1) \dots VIN * 6 * Rs / (Rs + 2 Rin) = 3.3V$$

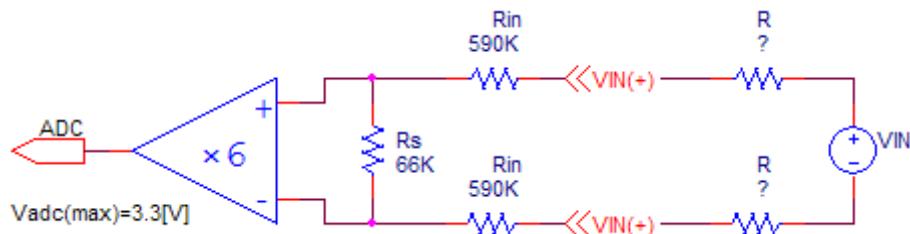
(식 1)을 정리하여 최대 입력전압의 범위를 계산하면,

$$(식 2) \dots VIN = 3.3 * (Rs + 2 Rin) / (6 * Rs)$$

(식 2)에 값을 대입하면,

$$VIN = 3.3 * (66K + 2 * 590K) / (6 * 66K) = 10.38[V]$$

이 전압은 희망하는 최대입력전압인 10[V]보다 조금 큰 값어야 합니다.



위 그림은 IO 커넥터의 입력단에 두개의 동일 저항(R)을 직렬로 연결하여 입력전압을 확대하는 방법입니다. 그러면, AIM의 본래 입력회로에서  $Rin \rightarrow (Rin + R)$ 로 바뀐 것과 동일하므로,

앞의 (식 1)에서  $Rin \rightarrow (Rin + R)$ 로 바꾸면 식도 간단히 유추됩니다.

$$(식 5) \dots VIN * 6 * Rs / (Rs + 2 (Rin + R)) = 3.3V$$

(식 5)를 정리하면,

(식 6)...  $R = 60 * VIN - 623 [k\Omega]$

예를들어, VIN에 최대 25[V]를 입력하고자 하면, 저항  $R = 60 * 25 - 623 = 877[k\Omega]$  이므로, 입력에 조금 큰 표준저항 910[kΩ] 2개를 직렬로 연결하면 됩니다.

외부저항을 사용하면 다시 사용자 교정을 해야 하므로, 외부에 연결한 저항 R의 정밀도는 그리 중요치 않습니다. 오히려 온도계수(TCR)가 낮은 것이 더 좋습니다. 일반적으로 1%정도의 저항이면 무난합니다.

### 6.3. AOM, Analog Output Module



4 채널의 아날로그 전압 및 전류를 출력할 수 있습니다. 채널 0/1 은 전압출력용이며 출력전압의 최대치를 설정할 수 있도록 포텐시미터를 내장하고 있으며, 출력단에 전류버퍼가 있으므로 수십 mA 이상의 큰 전류를 흘릴 수 있습니다.

그리고 채널 2/3 은 전압/전류를 동시에 출력합니다. 전압출력 단자는 최대 3.3V 의

전압이 출력됩니다만, OP-AMP 출력버퍼이므로 외부에서 수 mA 이상의 과도한 전류를 흘릴 경우 출력이 약간 변동될 수 있습니다. 전류출력은 최대 20mA 의 전류를 출력합니다. 기본적으로 0~20mA 의 전류를 출력하지만, 사용자 설정에 따라 원하는 범위의 전류를 출력하도록 설정할 수 있으므로, 산업용으로 많이 사용되는 4~20mA 의 전류를 출력하도록 설정할 수 있습니다. 전류출력은 노이즈에 강하므로 산업용으로 적합합니다. 신호를 수신하는 외부 디바이스는 적절한 저항을 추가하여 저항양단의 전압을 사용하면 됩니다.

전류출력 채널의 경우 최대 출력전압에 제한(**최대전압 4.0V**)이 있으므로 이를 고려하여 적당한 저항을 사용하십시오. 예를 들어 저항으로 200Ω 을 사용하면 출력전압은  $0.8V(4mA * 200\Omega)$  ~  $4V(20mA * 200\Omega)$ 가 됩니다. 250Ω 을 사용하면 출력전압은  $1.0V(4mA * 250\Omega)$  ~  $5V(20mA * 250\Omega)$ 가 되어야 하지만, 최대전압이 4.0V 이므로 1.0~4.0V 로 출력이 잘리게되므로 주의해 주십시오.

4~20mA 의 출력과 같이 ezconfig.exe 에서 설정을 바꾸면 원하는 전류를 출력할 수 있습니다. 단, 전류 출력의 경우 외부에 부하저항이 없으면 출력전압이 보장되지 않으므로 외부기기를 연결하지 않고 측정하면 약 4.5V 이상의 전압이 측정될 수도 있으므로 반드시 부하를 연결후 측정하십시오.

추가적으로 VDC 가변전원을 출력으로 제공하는데, 이는 버스의 VBB 전원에서 DCDC 변환기를 통하여 출력되는 전원이므로 큰 용량의 전원을 공급할 수는 없지만, 외부에 장착되는 센서 또는 장치등에 전원을 공급할 수 있습니다. 또한 그림처럼 가변저항으로 출력전압을 조정할 수 있으므로 매우 다양한 기기에 응용할 수 있습니다.

IO 커넥터 핀사양

| 핀 | 명칭    | 기능           | 기타                                                                                        |
|---|-------|--------------|-------------------------------------------------------------------------------------------|
| 1 | VOUT0 | 전압출력, 0~10V  | 사용자 교정시 최대 (VDC-3.0)[V] 까지 출력가능. 출력회로는 VDC 를 사용하므로 최대 출력 전압보다 VDC 를 3.0[V] 이상 더 높게 설정해야함. |
| 2 | GND   | 신호 그라운드      |                                                                                           |
| 3 | VOUT1 | 전압출력, 0~10V  |                                                                                           |
| 4 | GND   | 신호 그라운드      |                                                                                           |
| 5 | VOUT2 | 전압출력, 0~3.3V | 출고시 전류출력을 기준으로 교정되어 있으며                                                                   |

|    |       |                |                                                                             |
|----|-------|----------------|-----------------------------------------------------------------------------|
| 6  | IOUT2 | 전류 출력, 4~20mA  | 전압 출력은 교정되어 있지 않음.<br>0~20mA, 4~20mA 등으로 설정 가능.<br>전압 출력으로 교정도 가능(전압, 전류선택) |
| 7  | GND   | 신호 그라운드        |                                                                             |
| 8  | VOUT3 | 전압 출력, 3.3V 이하 |                                                                             |
| 9  | IOUT3 | 전류 출력, 4~20mA  |                                                                             |
| 10 | GND   | 신호 그라운드        |                                                                             |
| 11 | VDC   | 외부장치용 공급전압     | 전류: 3V(350mA), 5V(320mA),<br>12V(240mA), 18V(120mA)                         |
| 12 | GND   | 신호 그라운드        |                                                                             |

확장커넥터 핀사양

| 핀  | 핀 명칭                       | 핀 기능                                                     |
|----|----------------------------|----------------------------------------------------------|
| 1  | P2.0/SSEL1                 | GPIO / SPI1 Slave Select                                 |
| 2  | P2.1/SCK1                  | GPIO / SPI1 Serial Clock                                 |
| 3  | P2.2/MISO1                 | GPIO / SPI1 Master Input Slave Output                    |
| 4  | P2.3/MOSI1                 | GPIO / SPI1 Master Output Slave Input                    |
| 5  | P2.4                       | GPIO                                                     |
| 6  | P2.5                       | GPIO                                                     |
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 14 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1          |
| 15 | P1.10/AD6/CT16B1_MAT1      | GPIO / ADC6 / Match output 1 for 16-bit timer 1          |
| 16 | P1.11/AD7                  | GPIO / ADC7                                              |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로 직용 전원 출력                                    |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC) 전원 출력                                         |
| 24 | GND                        |                                                          |

```

typedef struct {
    MType          mtype;
    ID             id;
    AomInEvent*   in;
    AomOutEvent*  out;
} AOM;

```

AOM 모듈의 이미지는 위와 같은 AOM 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Aom 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력 이미지는 AomInEvent 구조체에, 출력 이미지는 AomOutEvent 구조체에 저장됩니다. (V0.6 부터 AomInChangedEvent,AomOutChangedEvent 가 간략히 AomInEvent,AomOutEvent 로 명칭변경됨)

```
|| typedef Event AomInEvent;
```

AOM 모듈은 출력모듈로 입력이 없으므로 기본이미지를 사용합니다.

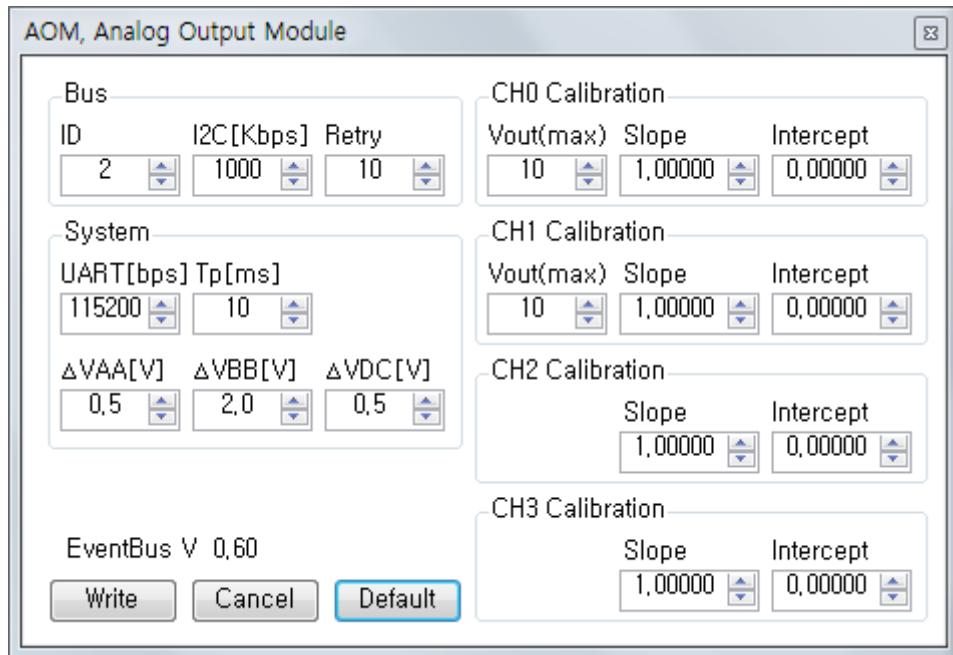
```
|| typedef struct {  
||   EventHead e;  
||   float      vout[2];  
||   float      iout[2];  
|| } AomOutEvent;
```

AOM 모듈의 각 출력은 12 비트 DAC 를 사용하므로 0~4095( $=2^{12}$ )까지의 값을 가질 수 있지만, 약간의 여유를 두어 4000 을 최대 분해능으로 사용하고 있습니다. 즉, 전압출력의 경우 10[V]를 출력하면 4000 이, 전류출력의 경우 20[mA]를 출력하면 4000 이 DAC 에 출력됩니다. 그리고 전압, 전류의 분해능은 각각  $10V/4000=2.5mV$ ,  $20mA/4000=5uA$  가 됩니다.

```
|| typedef struct {  
||   __BASE_PARAM;  
||   float      dVBB;  
||   float      dVDC;  
||   float      slope[4];  
||   float      intercept[4];  
||   uint8     voutMax[2];  
|| } AomParamEvent, AomParamChangedEvent, AomParamNotifiedEvent;
```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezConfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(aom\_drv.c)에 정의된 param 변수를 사용해서 param.dVBB 와 같이 읽습니다.

- \_\_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- dVBB, dVDC  
\_\_BASE\_PARAM 의 dVAA 참고.
- slope, intercept  
출력전압 교정에 사용되는 파라미터입니다.
- voutMax  
채널 0/1 의 최대출력전압의 범위를 설정합니다. 예를들어, 채널 0 으로 15V 를 출력하려면 Vout(max)를 15V 로 설정후 aom\_ex1.exe 를 이용하여 실제로 15V 를 출력한 후 가변저항(GAIN0)으로 15V 가 출력되도록 조정하면 됩니다.(출력전압 교정법 참고)



AOM 모듈을 ColinkEx 또는 UUM 으로 연결 후 ezconfig.exe 의 “Connect” 버튼으로 접속하면 이 설정창이 나타납니다. 각 항목이 위의 AOM\_PARAM 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 ezconfig.exe 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.

### 6.3.1. 출력전압 교정법

출고시 출력전압은 교정되어 있지만, 필요에 따라서는 사용자가 교정할 수 있습니다. 교정은 ezconfig.exe 를 사용하여 AOM 모듈에 접속한 후 Vout(max), Slope, Intercept 값을 바꾸면 됩니다. Slope, Intercept 를 쉽게 계산할 수 있도록 아래와 같은 엑셀파일이 제공되므로 카페에서 다운로드해서 사용하십시오. (카페의 다운로드에서 “AOM 교정” 엑셀파일 참고).

|    | A | B | C | D | E | F | G | H | I | J |
|----|---|---|---|---|---|---|---|---|---|---|
| 1  |   |   |   |   |   |   |   |   |   |   |
| 2  |   |   |   |   |   |   |   |   |   |   |
| 3  |   |   |   |   |   |   |   |   |   |   |
| 4  |   |   |   |   |   |   |   |   |   |   |
| 5  |   |   |   |   |   |   |   |   |   |   |
| 6  |   |   |   |   |   |   |   |   |   |   |
| 7  |   |   |   |   |   |   |   |   |   |   |
| 8  |   |   |   |   |   |   |   |   |   |   |
| 9  |   |   |   |   |   |   |   |   |   |   |
| 10 |   |   |   |   |   |   |   |   |   |   |
| 11 |   |   |   |   |   |   |   |   |   |   |
| 12 |   |   |   |   |   |   |   |   |   |   |
| 13 |   |   |   |   |   |   |   |   |   |   |
| 14 |   |   |   |   |   |   |   |   |   |   |
| 15 |   |   |   |   |   |   |   |   |   |   |
| 16 |   |   |   |   |   |   |   |   |   |   |
| 17 |   |   |   |   |   |   |   |   |   |   |
| 18 |   |   |   |   |   |   |   |   |   |   |
| 19 |   |   |   |   |   |   |   |   |   |   |
| 20 |   |   |   |   |   |   |   |   |   |   |
| 21 |   |   |   |   |   |   |   |   |   |   |
| 22 |   |   |   |   |   |   |   |   |   |   |
| 23 |   |   |   |   |   |   |   |   |   |   |

[V0.50] AOM 교정방법

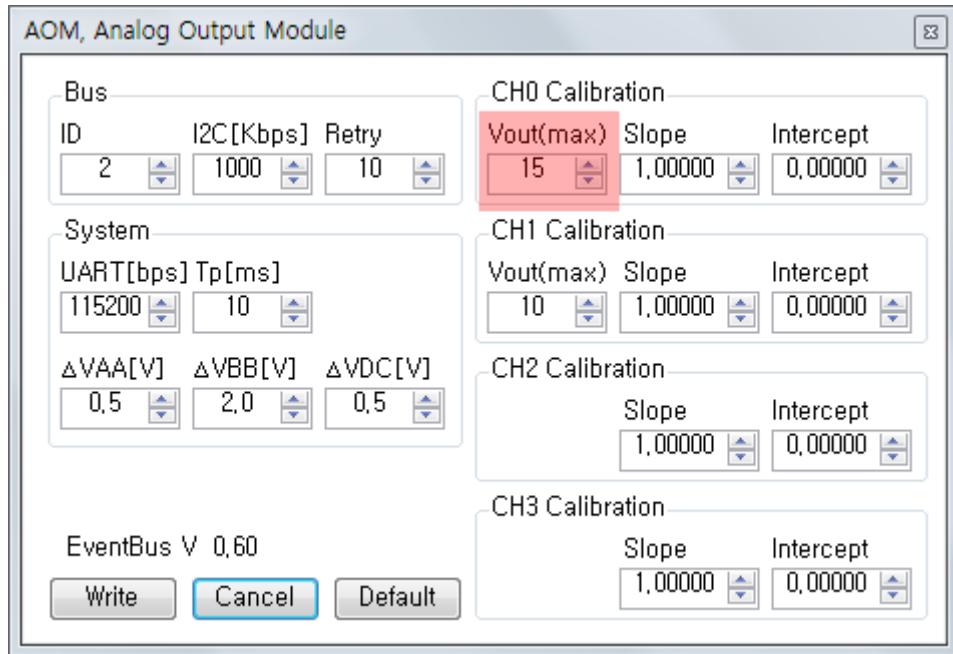
|         | AOM1[V] | AOM2[V] | DVM1[V]  | DVM2[V] | Slope   | Intercept |
|---------|---------|---------|----------|---------|---------|-----------|
| CH0 [V] | 10.000  | 0.500   | 10.03200 | 0.28030 | 0.97419 | 0.22693   |
| CH1 [V] | 10.000  | 0.500   | 10.08400 | 0.27990 | 0.96898 | 0.22878   |
| CH2 [V] | 2.000   | 0.050   | 2.22180  | 0.07106 | 0.90666 | -0.01443  |
| CH3 [V] | 2.000   | 0.050   | 2.22050  | 0.05898 | 0.90214 | -0.00321  |

0. CH0,1은 전압출력이므로 그대로 측정하면 되지만,  
 \* CH2,3은 전류출력이므로 정밀저항을 연결후 전압을 측정해야 한다.  
 \* 정밀저항이 없을 경우 저항값을 측정하여 사용한다.  
 \* VDC=13V에 맞춘다

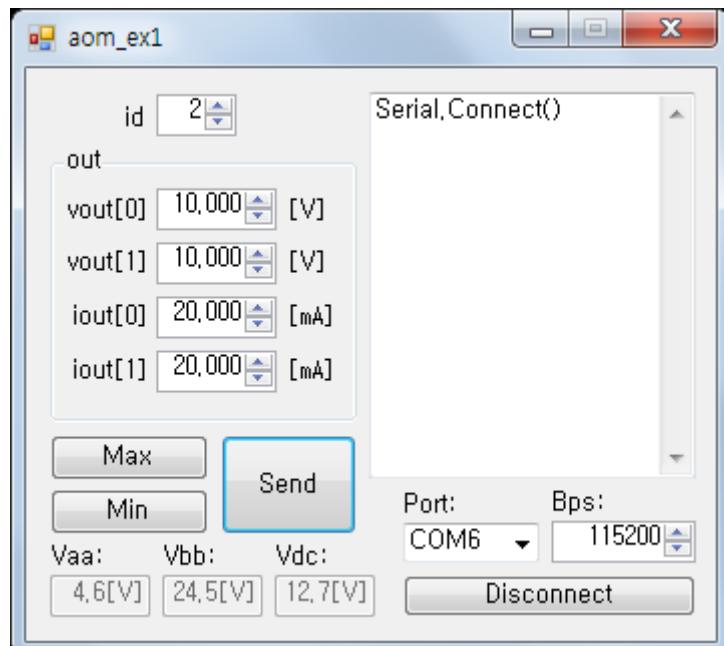
- ezconfig.exe에서 [Default], [Write] 버튼으로 초기치로 설정한다.  
 \* vout(max)=10, slope=1, Intercept=0 으로 초기화 된다.
- aom\_ex1에서 [Max]+[Send] 버튼으로 CH0,1=10V, CH2,3=20mA(2V출력)를 출력한다.
- 가변저항(GAIN0,1)으로 CH0,1의 DVM전압을 대략 10V에 맞춘다.
- CH0-3의 전압을 DVM으로 측정하고 DVM1에 기록한다.
- [Min]+[Send] 버튼으로 CH0,1=0.5V, CH2,3=0.5mA(0.05V출력)를 출력한다.
- CH0-3의 전압을 DVM으로 측정하고 DVM2에 기록한다.
- slope, intercept를 ezconfig.exe에 입력후 [Write] 버튼으로 저장한다.  
 \* 만일, 채널0의 최대출력이 20V라면, 채널0의 Vout(max)를 20으로 수정하고 저장한 후  
 \* aom\_ex1에서 20V를 출력후 실제출력이 20V가 되도록 DVM으로 측정하면서 가변저항으로 조정한다.

### 6.3.2. 최대 출력전압의 변경

채널 0/1 의 최대 출력전압은 기본적으로 10V 입니다. 교정을 마친후 가변저항(GAIN0, GAIN1)으로 출력전압의 범위를 변경할 수도 있습니다. 예를 들어, 출력전압을 0~15V 범위로 변경하는 과정을 살펴보겠습니다.



우선 그림처럼 ezconfig.exe 를 실행후 Vout(max)를 15V 로 설정하고, VDC 전압은 가변저항을 시계방향으로 돌려서 최대전압보다 약 3V 높은 18V로 조정합니다.



그리고, aom\_ex1.exe 예제프로그램을 실행하고 최대값 15V 를 출력후 가변저항(GAIN0)으로 실제로 15V 가 출력되도록 조정하면 됩니다.

마찬가지로 출력전압을 낮출수도 있습니다. 방법은 동일합니다. 예를들어, 출력전압을 0~5V 범위로 줄이려면, `aom_ex1.exe` 예제프로그램을 실행하고 최대전압인 5V를 출력후 가변저항으로 5V가 출력되도록 조정하면 됩니다. 단, 이경우에는 VDC를 8V로 낮출 필요는 없습니다. 그대로 13V로 두어도 무방합니다.

`eventbus_net_0.60\example\aom_ex1\bin\Debug\aom_ex1.exe` 예제프로그램을 실행하면 AOM을 테스트할 수 있습니다. 연결된 COM 포트를 선택하고, 통신속도는 기본적으로 설정된 115,200bps를 사용합니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(`aom_def.c` 참고). 출력할 전압 또는 전류를 입력후 [Send]버튼을 클릭하면 설정한 전압이 출력됩니다. 출력의 크기에 따라 PCB 아래에 있는 LED의 밝기가 변합니다.

채널 0/1의 최대 출력전압은 20V까지 설정할 수 있습니다. 이경우 VDC 전압을 23V가 되도록 가변저항으로 조정해야 합니다(출고시 13V로 설정되어 있습니다).

## 6.4. DIO, Digital Input/Output Module



스위치 접점, 온오프 센서, 디지털 전압 등의 온오프 형태의 디지털 신호를 입력받아 시스템을 제어하기 위한 6 채널의 접점 입력과 램프, 솔레노이드, 전등 등의 온오프 출력장치를 구동하기 위한 2 채널의 릴레이 접점출력으로 구성되어 있습니다.

추가적으로 VDC 가변전원을 출력으로 제공하는데, 이는 버스의 VBB 전원에서 DCDC 변환기를 통하여 출력되는 전원이므로 큰 용량의 전원을 공급할 수는

없지만, 외부에 장착되는 센서 또는 장치등에 전원을 공급할 수 있습니다. 이 전원을 이용하면 매우 편리한데, 일례로 PLC 에서는 이런 용도로 별도의 전원을 외부에 설치해야 합니다. 하지만 이지컨트롤러에서는 적절한 용량의 전원을 외부로 제공하고 있어 별도의 외부 전원이 필요하지 않습니다. 더우기 그림에서 보듯이 가변저항으로 출력전압을 조정할 수 있으므로 매우 다양한 기기에 응용할 수 있습니다.

스위치 입력회로는 싱크(SINK)타입으로 외부에서 핀으로 전류를 공급하는 방식(DIM 모듈 참고)이므로 PNP 타입의 센서가 적합합니다. 릴레이 출력은 DOM 모듈을 참고하십시오.

커넥터 판사양

| 핀번호 | 명 칭  | 기 능        | 기 태                                                                       |
|-----|------|------------|---------------------------------------------------------------------------|
| 1   | RLY0 | 릴레이 접점 출력. | 최대 접점내압: 110VDC, 250VAC<br>최대 접점전류: 5A                                    |
| 2   | RLY0 |            |                                                                           |
| 3   | RLY1 | 릴레이 접점 출력. |                                                                           |
| 4   | RLY1 |            |                                                                           |
| 5   | SW0  | 스위치 접점 입력. | 동작 전압 : 8V 이상<br>최대 내압 : ±50V                                             |
| 6   | SW1  | 스위치 접점 입력. |                                                                           |
| 7   | SW2  | 스위치 접점 입력. |                                                                           |
| 8   | SW3  | 스위치 접점 입력. |                                                                           |
| 9   | SW4  | 스위치 접점 입력. |                                                                           |
| 10  | SW5  | 스위치 접점 입력. |                                                                           |
| 11  | VDC  | 외부장치용 공급전압 | 가변저항 조정: 18VDC MAX<br>전류: 3V(350mA), 5V(320mA),<br>12V(240mA), 18V(120mA) |
| 12  | GND  | 신호 그라운드    |                                                                           |

확장커넥터 판사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5 | P2.4       | GPIO                                  |
| 6 | P2.5       | GPIO                                  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 14 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1          |
| 15 | P1.10/AD6/CT16B1_MAT1      | GPIO / ADC6 / Match output 1 for 16-bit timer 1          |
| 16 | P1.11/AD7                  | GPIO / ADC7                                              |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | 3.3V로 직용 전원 출력                                           |
| 22 | GND                        |                                                          |
| 23 | VBB                        | 24V 전원 출력                                                |
| 24 | GND                        |                                                          |

```
typedef struct {
    MType          mtype;
    ID             id;
    DioInEvent*   in;
    DioOutEvent*  out;
} DIO;
```

DIO 모듈의 이미지는 위와 같은 DIO 구조체로 되어 있습니다(eventbus.h 참조). mtype은 모듈의 타입으로 Dio 타입이 됩니다. id는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 스위치 입력은 DioInEvent 구조체에, 릴레이 출력은 DioOutEvent 구조체에 저장됩니다. (V0.6부터 DioInChangedEvent,DioOutChangedEvent 가 간략히 DioInEvent,DioOutEvent로 명칭변경됨)

```
typedef struct {
    EventHead e;
    union {
        uint16 sw;
        struct {
            ELogic sw0 :1;
            ELogic sw1 :1;
            ELogic sw2 :1;
            ELogic sw3 :1;
            ELogic sw4 :1;
            ELogic sw5 :1;
        };
    };
} DioInEvent;
```

DIO 모듈의 각 스위치 입력 상태는 1비트에 저장되며 sw0 ~ sw5의 6개로 구성됩니다. 6개의 입력을 동시에 다루고자 할 때는 1바이트 크기의 변수 sw를 사용합니다.

예) 스위치 0번이 눌러졌는지 검사하려면,  
`if (dio->in->sw0 == ON) ...`

예) 현재 전체 스위치의 상태를 저장하려면,  
`uint16 sw = dio->in->sw;`

```
typedef struct {
    EventHead e;
    union {
        uint16 rly;
        struct {
            ELogic rly0 :1;
            ELogic rly1 :1;
        };
    };
} DioOutEvent;
```

DIO 모듈의 각 릴레이 출력 상태는 1 비트에 저장되며 rly0 ~ rly1 의 2 개로 구성됩니다.  
2 개의 출력을 동시에 다루고자 할 때는 1 바이트 크기의 변수 rly 를 사용하며, 최하위 2 비트만  
사용됩니다.

예) 스위치 0 번에 연동하여 릴레이 1 번을 동작 시키려면,

`dio->out->rly1 = dio->in->sw0;`

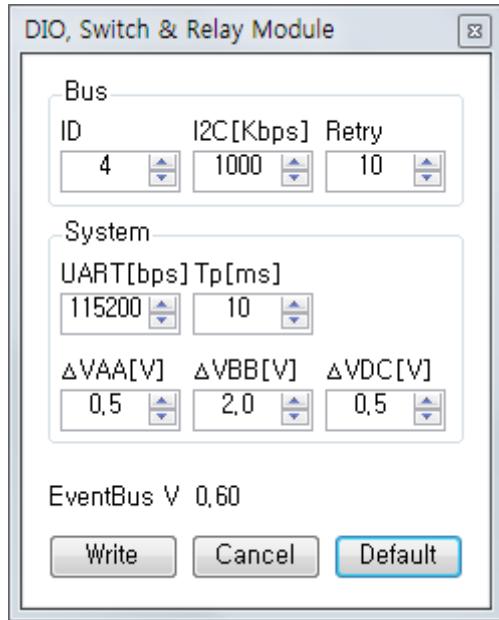
예) 릴레이를 모두 온 시키려면,

`dio->out->rly = 3;`

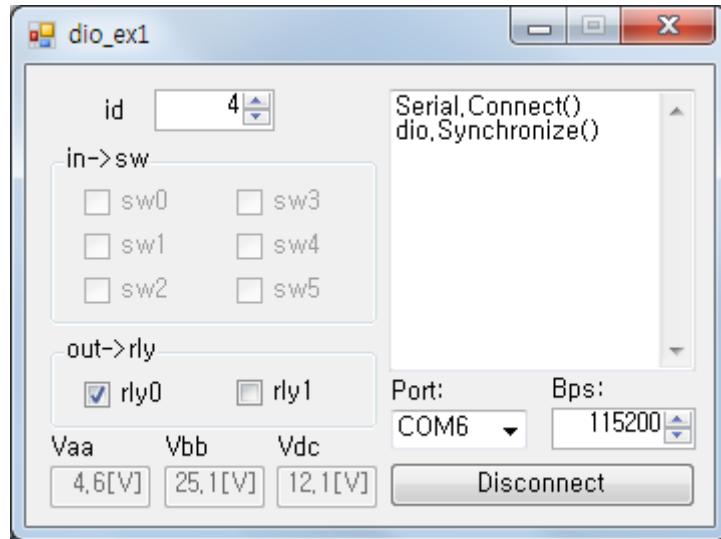
```
typedef struct {
    __BASE_PARAM;
    float dVBB;
    float dVDC;
} DioParamEvent,DioParamChangedEvent,DioParamNotifiedEvent;
```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezConfig.exe  
프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면  
드라이버(dio\_drv.c)에 정의된 param 변수를 사용해서 param.dVBB 와 같이 읽습니다.

- `__BASE_PARAM`  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- `dVBB, dVDC`  
`__BASE_PARAM`의 `dVAA` 참고.

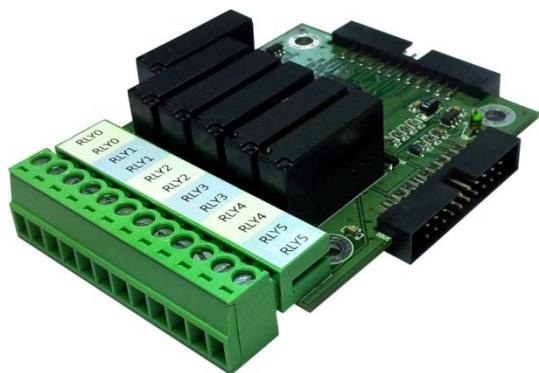


`ezconfig.exe` 의 설정창입니다. DIO 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 DIO\_PARAM 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 `ezconfig.exe`에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.



`eventbus_net_0.60\example\dio_ex1\bin\Debug\dio_ex1.exe` 예제프로그램을 실행하면 그림처럼 DIO 를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. rly0,1 체크박스를 클릭하면 릴레이의 동작음을 들을 수 있으며, 스위치를 동작하면 sw 체크박스에 표시됩니다. V0.60 부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(`dio_def.c` 참고)

## 6.5. DOM, Digital Output Module



램프, 솔레노이드, 전등 등과 같은 온오프 출력장치를 구동하기 위한 6 채널의 릴레이 접점출력으로 구성되어 있습니다. 릴레이는 최대 5A 까지 전류를 흘릴 수 있으며 DC110V, AC250V 의 전압을 스위칭 할 수 있으므로 전등이나 AC 모터의 온오프제어에도 사용할 수 있습니다. 각각의 릴레이는 NO 접점이며, 릴레이간에 서로 연결되지 않은 별도의 접점을 제공하므로 사용이 편리합니다.

### Contact Data

|                                   |                          |                                                                      |
|-----------------------------------|--------------------------|----------------------------------------------------------------------|
| Contact Arrangement               | 1A(SPSTNO)               |                                                                      |
| Contact Material                  | Silver Alloy (Gold clad) |                                                                      |
| Contact Rating (resistive)        | 3A,5A/30VDC,250VAC;      |                                                                      |
| Max. Switching Power              | 150W                     | 1250VAC<br>min Load:0.1mA/0.1VDC (reference value)                   |
| Max. Switching Voltage            | 110VDC                   | 250VAC<br>Max.Switching Current:5A                                   |
| Contact Resistance & Voltage drop | <50mΩ (at 1A/6V)         | Item 4.12 of IEC 61810-7                                             |
| Operational life                  | Electrical               | 1×10 <sup>5</sup> × 10 <sup>4</sup> (5A)<br>Item 4.30 of IEC 61810-7 |
|                                   | Mechanical               | 2×10 <sup>7</sup><br>Item 4.31 of IEC 61810-7                        |

### IO 커넥터 핀사양

| 핀번호 | 명 칭  | 기 능        | 기 타                     |
|-----|------|------------|-------------------------|
| 1   | RLY0 | 릴레이 접점 출력. | 최대 접점내압: 110VDC, 250VAC |
| 2   | RLY0 |            | 최대 접점전류: 5A             |
| 3   | RLY1 | 릴레이 접점 출력. |                         |
| 4   | RLY1 |            |                         |
| 5   | RLY2 | 릴레이 접점 출력. |                         |
| 6   | RLY2 |            |                         |
| 7   | RLY3 | 릴레이 접점 출력. |                         |
| 8   | RLY3 |            |                         |
| 9   | RLY4 | 릴레이 접점 출력. |                         |
| 10  | RLY4 |            |                         |
| 11  | RLY5 | 릴레이 접점 출력. |                         |
| 12  | RLY5 |            |                         |

### 확장커넥터 핀사양

| 핀  | 핀 명칭       | 핀 기능                                  |
|----|------------|---------------------------------------|
| 1  | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2  | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3  | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4  | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5  | P2.4       | GPIO                                  |
| 6  | P2.5       | GPIO                                  |
| 7  | P2.6       | GPIO                                  |
| 8  | P2.7       | GPIO                                  |
| 9  | P2.8       | GPIO                                  |
| 10 | P2.9       | GPIO                                  |
| 11 | P2.10      | GPIO                                  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 14 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1          |
| 15 | P1.10/AD6/CT16B1_MAT1      | GPIO / ADC6 / Match output 1 for 16-bit timer 1          |
| 16 | P1.11/AD7                  | GPIO / ADC7                                              |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | 3.3V로직용 전원 출력                                            |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24V) 전원 출력                                           |
| 24 | GND                        |                                                          |

```
typedef struct {
    MType          mtype;
    ID             id;
    DomINEven*    in;
    DomOutEvent*  out;
} DOM;
```

DOM 모듈의 이미지는 위와 같은 DOM 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Dom 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 그리고, 릴레이 출력은 DomOutEvent 구조체에 저장됩니다. (V0.60 부터 DomInChangedEvent,DomOutChangedEvent 가 간략히 DomInEvent,DomOutEvent 로 명칭변경됨)

```
typedef Event DomInEvent;
```

DOM 모듈은 출력모듈이며 입력이 없으므로 기본이미지를 재정의하여 사용합니다.

```
typedef struct {
    EventHead  e;
    union {
        uint16 rly;
        struct {
            ELogic rly0  :1;
            ELogic rly1  :1;
            ELogic rly2  :1;
            ELogic rly3  :1;
            ELogic rly4  :1;
            ELogic rly5  :1;
        };
    };
} DomOutEvent;
```

DOM 모듈의 각 릴레이 출력 상태는 1 비트에 저장되며 rly0 ~ rly5 의 6 개로 구성됩니다. 6 개의 출력을 동시에 다루고자 할 때는 1 바이트 크기의 변수 rly 를 사용합니다.

예) 1 번 릴레이를 ON 시키려면,

```
dom->out->rly1 = ON;
```

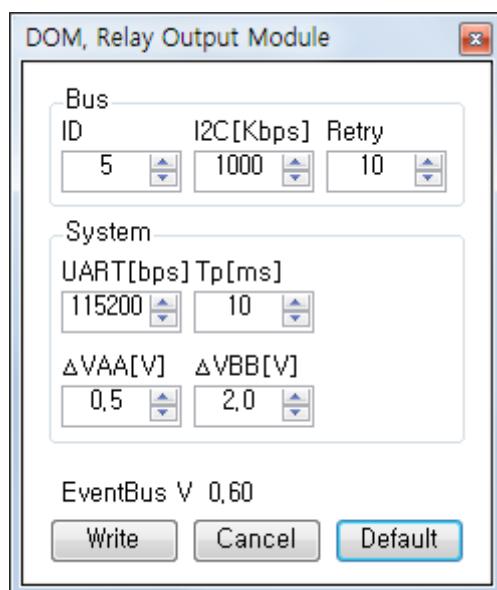
예) 전체릴레이를 모두 ON 시키려면,

```
dom->out->rly = 0x3F;
```

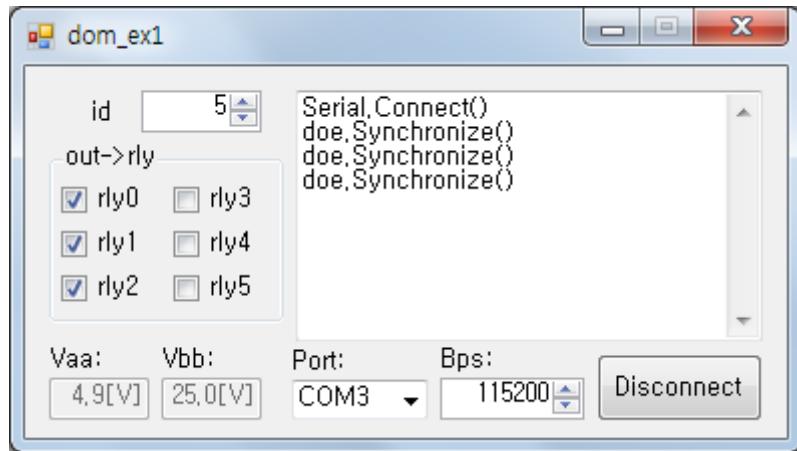
```
typedef struct {
    __BASE_PARAM;
    float      dVBB;
} DomParamEvent,DomParamChangedEvent,DomParamNotifiedEvent;
```

이 구조체는 EEPROM에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezConfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(dom\_drv.c)에 정의된 param 변수를 사용해서 param.dVBB와 같이 읽습니다.

- \_\_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- dVBB  
\_\_BASE\_PARAM의 dVAA 참고.

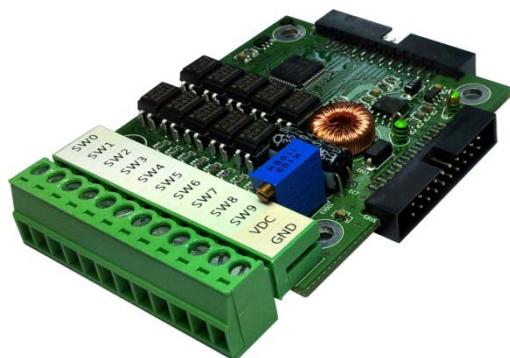


ezconfig.exe의 설정창입니다. DOM 모듈을 ColinkEx 또는 UUM으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 DOM\_PARAM과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM에 영구 저장됩니다. V0.60부터 컴퓨터와 연결속도를 ezconfig.exe에서 설정할 수 있으며, 기본속도는 115,200bps입니다.



eventbus\_net\_0.60\example\dom\_ex1\bin\Debug\dom\_ex1.exe 예제프로그램을 실행하면 그림처럼 DOM 을 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. rly0 ~ rly5 체크박스를 클릭하면 릴레이의 동작음을 들을 수 있습니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(dom\_def.c 참고)

## 6.6. DIM, Digital Input Module



스위치 접점, 장애물 검출센서 등의 온오프 센서, 온오프 전압입력과 같은 디지털 입력 등의 온오프 형태의 디지털 신호를 입력받아 시스템을 제어하기 위한 10 채널의 접점 입력을 제공합니다. DIM은 비절연 싱크타입 접점입니다. 절연타입 접점은 D1E를 사용해 주십시오.

추가적으로 VDC 가변전원을 출력으로 제공하는데, 이는 버스의 VBB 전원에서 DCDC 변환기를 통하여

출력되는 전원이므로 큰 용량의 전원을 공급할 수는 없지만, 외부에 장착되는 센서 또는 장치등에 전원을 공급할 수 있습니다. 더욱이 그림에서 보듯이 가변저항으로 출력전압을 조정할 수 있으므로 매우 다양한 기기에 응용할 수 있습니다.

IO 커넥터 핀사양

| 핀  | 명 칭 | 기 능        | 기 타                                              |
|----|-----|------------|--------------------------------------------------|
| 1  | SW0 | 스위치 접점 입력. |                                                  |
| 2  | SW1 | 스위치 접점 입력. |                                                  |
| 3  | SW2 | 스위치 접점 입력. |                                                  |
| 4  | SW3 | 스위치 접점 입력. |                                                  |
| 5  | SW4 | 스위치 접점 입력. |                                                  |
| 6  | SW5 | 스위치 접점 입력. |                                                  |
| 7  | SW6 | 스위치 접점 입력. |                                                  |
| 8  | SW7 | 스위치 접점 입력. |                                                  |
| 9  | SW8 | 스위치 접점 입력. |                                                  |
| 10 | SW9 | 스위치 접점 입력. |                                                  |
| 11 | VDC | 외부장치용 공급전압 | 전류: 3V(350mA), 5V(320mA), 12V(240mA), 18V(120mA) |
| 12 | GND |            |                                                  |

확장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5 | P2.4       | GPIO                                  |
| 6 | P2.5       | GPIO                                  |
| 7 | P2.6       | GPIO                                  |
| 8 | P2.7       | GPIO                                  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 14 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1          |
| 15 | P1.5/CT32B0_CAP0           | GPIO / Capture input 0 for 32-bit timer 0                |
| 16 | P0.3                       | GPIO                                                     |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로직용 전원 출력                                     |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC) 전원 출력                                         |
| 24 | GND                        |                                                          |

```
typedef struct {
    MType          mtype;
    ID             id;
    DimInEvent*   in;
    DimOutEvent*  out;
} DIM;
```

DIM 모듈의 이미지는 위와 같은 DIM 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Dim 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 스위치 입력은 DimInEvent 구조체에 저장되며, 입력모듈이므로 출력은 없으며 DimOutEvent 구조체는 내용이 없습니다. (v0.60 부터 DimInChangedEvent, DimOutChangedEvent 가 간략히 DimInEvent, DimOutEvent 로 명칭변경됨)

```
typedef struct {
    EventHead e;
    union {
        uint16 sw;
        struct {
            ELogic sw0 :1;
            ELogic sw1 :1;
            ELogic sw2 :1;
            ELogic sw3 :1;
            ELogic sw4 :1;
            ELogic sw5 :1;
            ELogic sw6 :1;
            ELogic sw7 :1;
            ELogic sw8 :1;
            ELogic sw9 :1;
        };
    };
}
```

```

    };
};

} DimInEvent;

```

DIM 모듈의 각 스위치 입력 상태는 1 비트에 저장되며 sw0 ~ sw9 의 10 개로 구성됩니다. 10 개의 입력을 동시에 다루고자 할 때는 2 바이트 크기의 변수 sw를 사용합니다.

예) 스위치 0 번이 눌러졌는지 검사하려면,  
`if (dim->in->sw0 == ON) ...`

예) 현재 전체 스위치의 상태를 저장하려면,  
`uint16 temp = dim->in->sw;`

```

typedef Event DimOutEvent;

```

DIM 모듈은 입력모듈로 출력이 없으므로 기본이미지를 사용합니다.

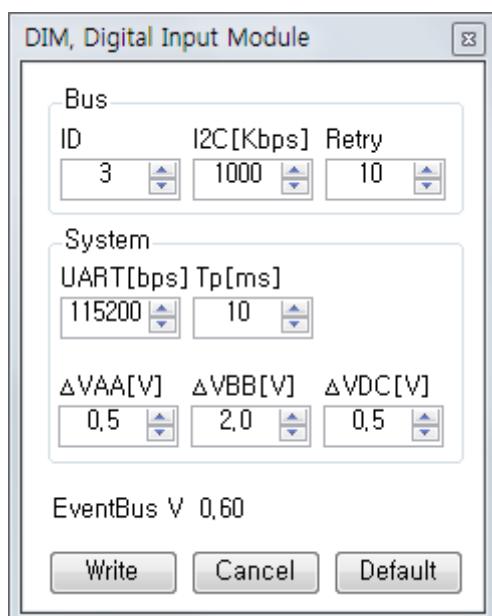
```

typedef struct {
    __BASE_PARAM;
    float      dVBB;
    float      dVDC;
} DimParamEvent, DimParamChangedEvent, DimParamNotifiedEvent;

```

이 구조체는 EEPROM에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezConfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(dim\_drv.c)에 정의된 param 변수를 사용해서 param.dVBB와 같이 읽습니다.

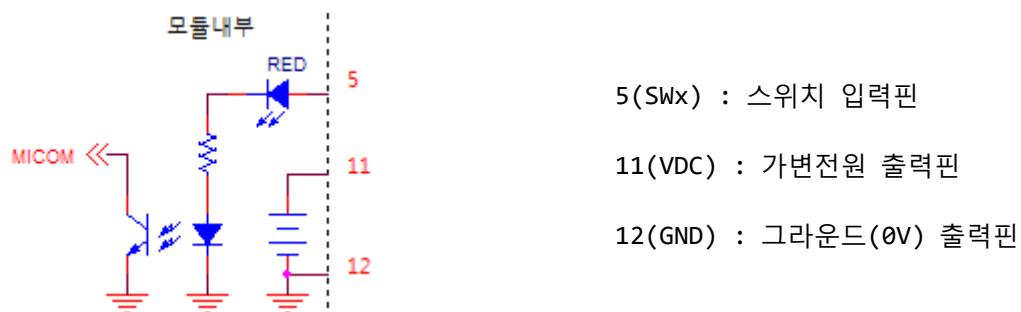
- \_\_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- dVBB, dVDC  
\_\_BASE\_PARAM 의 dVAA 참고.



ezconfig.exe 의 설정창입니다. DIM 모듈을 ColinkEx 또는 UUM 으로 PC 에 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 DIM\_PARAM 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 ezconfig.exe 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.

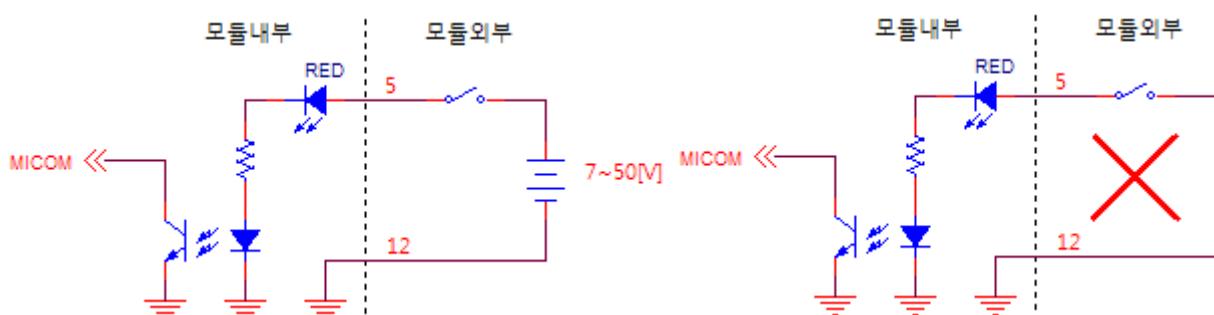
### 6.6.1. 싱크(Sink)타입 스위치 입력

DIM, DIO 의 스위치 입력회로는 싱크(SINK) 타입으로 외부에서 핀으로 전류를 공급할 때 스위치가 온으로 인식되는 방식입니다. 입력회로는 아래와 같습니다.

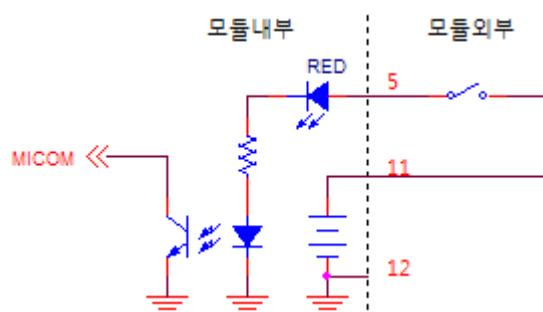


5 번핀으로 8V 이상의 전압이 입력되면 포토커플러가 동작하고 마이컴은 스위치의 동작을 인식할 수 있습니다.

\*주의\* 비절연타입이므로 스위치와 같은 접점신호만 연결하십시오. 센서등을 연결할 때는 출력되는 VDC 전원이나 VAA, VBB 전원을 사용해 주십시오. 외부전원이 연결되면 전위차에 의한 오동작 및 파손이 될 수 있으므로, 이 경우는 외부에 포토커플러, 릴레이 등의 절연회로를 추가하거나 절연타입인 DIe 를 사용해 주십시오.

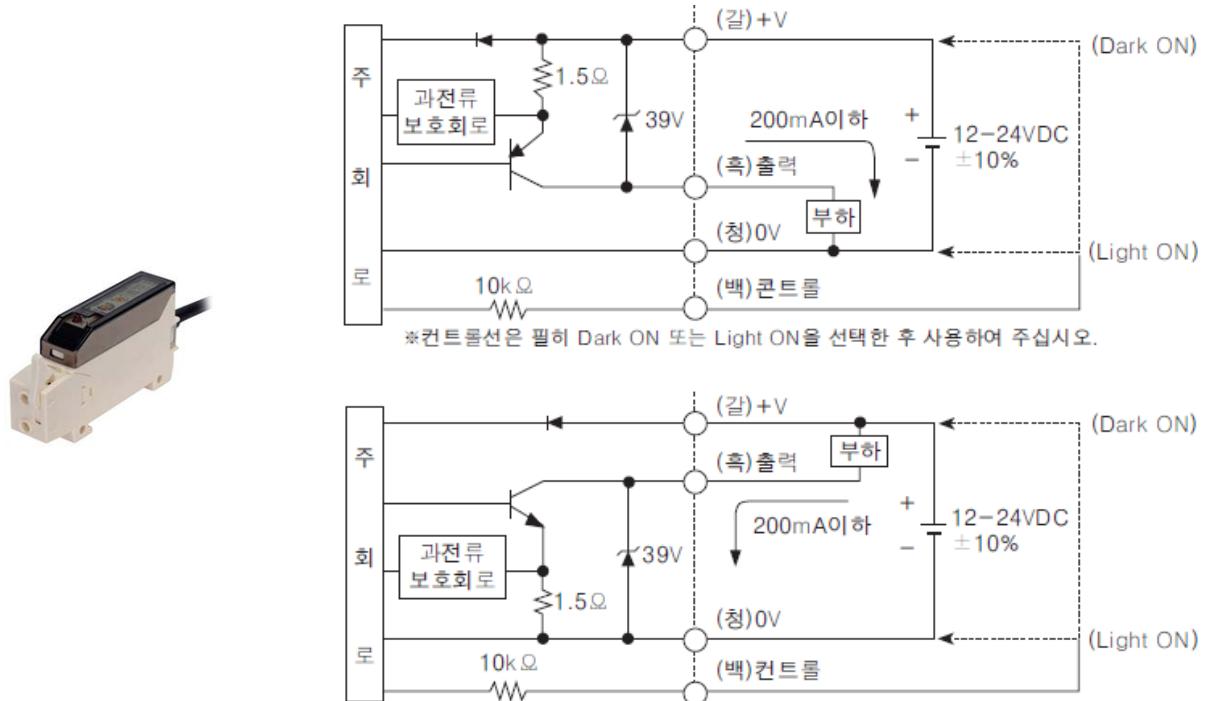


그림처럼, 스위치만 연결해서는 동작하지 않으며, 외부전원을 직렬로 연결하여야 합니다.



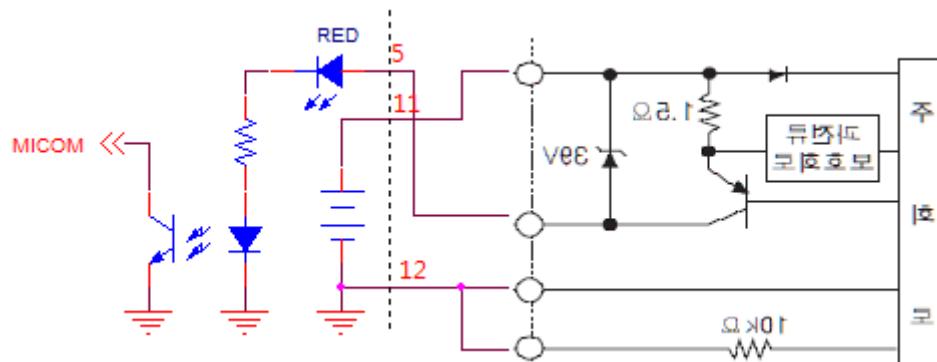
하지만, 외부에 별도로 전원을 설치하는 것은 경우에 따라서는 매우 불편하므로, 이지컨트롤러의 VDC 가변전원을 이용하여 그림처럼 스위치만 연결할 수 있습니다.

### 6.6.2. 트랜지스터 출력타입의 센서를 연결하는 방법

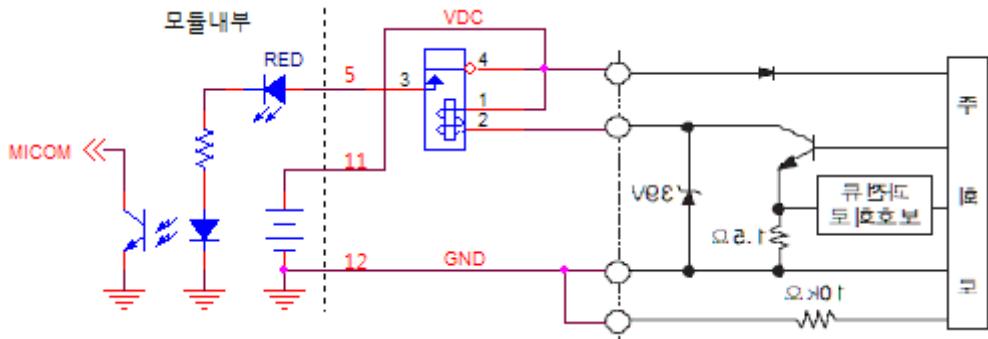


그림은 오토닉스의 광화이버센서입니다. 그림처럼 TR 출력은 NPN, PNP 출력의 두가지 타입이 있습니다. 요즘은 두가지 모두 내장하고 선택할 수 있도록 하는 편리한 타입도 있습니다. 갈색과 청색은 전원선이고, 흑색이 출력선입니다. 백색은 동작방식을 선택하는 입력으로 청색선에 연결하면 Light ON(빛을 받으면 동작)으로 동작하고 갈색선에 연결하면 Dart ON(어두울때 동작)으로 동작합니다.

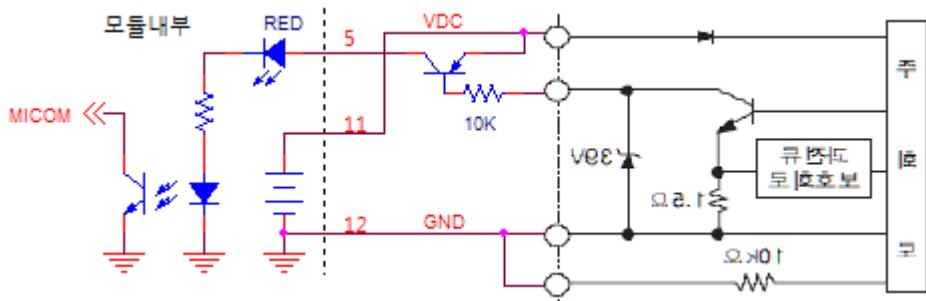
문제는 IO 커넥터의 스위치입력핀에 연결하는 흑색선입니다. 앞에서 살펴본 것처럼, DIO, DIM 의 스위치 입력회로는 싱크(SINK) 타입이므로 센서는 전류를 출력하는 소스(SOURCE) 타입이어야 합니다. 그러므로, 위의 두가지 타입중 PNP 타입 출력이 적합합니다.



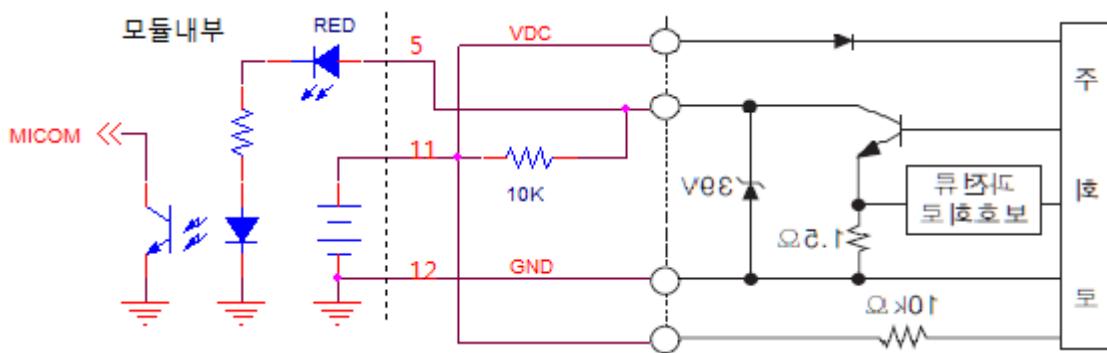
그림은 광화이버센서를 연결한 예입니다. PNP 출력의 경우 바로 연결할 수 있으므로 편리하지만, NPN 출력의 경우는 바로 연결할 수 없으므로 아래와 같은 추가회로가 필요합니다.



소형릴레이를 사용한 경우입니다. VDC=12V에 설정하고, 12V용 릴레이를 사용합니다. 주의할 점은 VDC의 전류용량이 그리 크지 않으므로 큰 전류를 소모하는 릴레이를 사용하지 말아주십시오(표 참고).

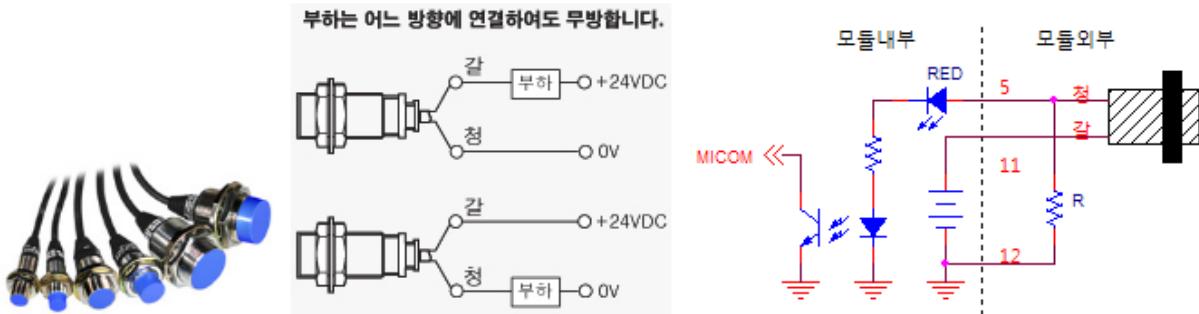


PNP 트랜지스터를 사용한 경우입니다. 범용의 T092 타입 트랜지스터와 리드저항 한개를 사용하여 연결하였습니다.



그림은 저항 한 개로 연결한 경우입니다. 단, 이경우는 센서값이 반대로 읽히게 되므로, 프로그램에서 읽은 값을 반전하여 사용해도 되지만, 위의 센서의 경우 백색(컨트롤)선이 있으므로 이를 VDC에 연결해서 Dark ON으로 동작시키면 프로그램에서는 반대로 읽히므로 Light ON이 되어 정상적으로 동작합니다.

### 6.6.3. 근접센서(스위치) 연결방법



그림은 오토닉스의 근접센서(스위치)입니다. 근접스위치는 내부에 스위치가 있다고 생각해도 무방합니다. 즉, 물체가 접근하면 두선에 연결된 스위치가 동작하므로, DIM, DIO 모듈의 접점입력 단자에 연결하려면, 갈색선(V+)을 VDC(11 번핀)에 연결하고, 청색선(V-)은 IO 커넥터의 입력으로 연결하면 됩니다.

하지만, 근접스위치는 특성상 스위치가 동작하지 않아도 흘러야 하는 최소전류(누설전류)가 있습니다. 누설전류가 입력단자(그림에서 5 번핀)로 흘러들면 입력회로가 오동작 할 수 있습니다. 그래서 외부에 블리더저항(R)을 연결해서 누설전류가 블리더저항으로 흐르도록 합니다.

단, 블리더저항에 누설전류가 흘러서 저항양단의 전압이 입력최소전압인 7[V]를 넘어서면 오인식 하므로,

$$R(\text{블리더저항}) < (7[V] \div \text{누설전류})$$

누설전류가 0.7mA 인 경우 저항은 10kΩ 이하가 되므로, 안전하게 5.1kΩ 을 사용합니다. 또한, 근접센서는 온 되어도 양단에 전압이 완전히 0 이 되지 않습니다. 당연히 동작에 필요한 최소전압은 있어야 겠지요. 이것이 잔류전압입니다. 보통 데이터시트에는 3.5[V] 정도로 나옵니다.

즉, VDC=12V 이고, 센서가 온되면, IO 커넥터로 입력되는 전압은

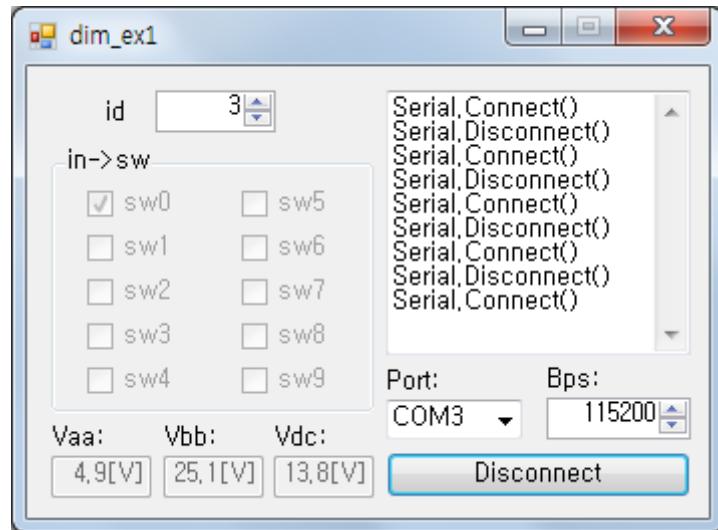
$$(VDC - \text{잔류전압}) = (12 - 3.5) = 8.5[V] \text{ 이므로,}$$

스위치를 인식하기 위한 최소전압인 7[V]보다 커지므로 정상적으로 센서가 인식 됩니다.

만일, 전압이 7[V]보다 작다면, 가변저항을 조정하여(시계방향이 전압증가) 전압을 높여야 합니다.

$$VDC > 7[V] + \text{잔류전압}$$

이지만, 충분히 높은 전압을 사용하는 것이 안정적입니다.



eventbus\_net\_0.60\example\dim\_ex1\bin\Debug\dim\_ex1.exe 예제프로그램을 실행하면 그림처럼 DIM을 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps를 사용합니다. 스위치를 동작하면 sw 체크박스에 표시됩니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(dim\_def.c 참고)

## 6.7. DIE, Digital Input Expanded Module



스위치 접점, 장애물  
검출센서와 같은 온오프 센서,  
디지털 전압입력 등과 같은  
온오프 형태의 입력으로  
제어시스템을 구성할 수 있도록  
16 채널의 접점 입력을  
제공합니다. DIO, DIM 과 달리  
DIE 는 내부동작전원과  
외부배선이 완전히 절연된

절연방식 접점을 사용하므로 공급전원의 전위차가 있는 경우에도 안정하게 동작하며, 양전압, 음전압 또는 AC 전압(아파 표 참고)의 입력도 가능하므로 매우 유연한 결선방법을 제공합니다.

추가적으로 2 핀 커넥터는 VBB 전원을 출력으로 제공하는데, 이는 버스의 VBB 전원이 그대로 노출되는 것이므로 스위치 또는 센서와 같이 절연이 필요치 않은 간단한 용도로 사용하기 위함이며, 완전한 절연이 필요한 입력에는 사용하지 않아야 합니다.

I01 커넥터 핀사양

| I01 커넥터 |     |      | I02 커넥터 |      |      | VBB 커넥터 |     |                      |
|---------|-----|------|---------|------|------|---------|-----|----------------------|
| 핀       | 명칭  | 기능   | 핀       | 명칭   | 기능   | 핀       | 명칭  | 기능                   |
| 1       | SW0 | 접점입력 | 1       | SW8  | 접점입력 | 1       | VBB | 24V 출력               |
| 2       | SW1 | 접점입력 | 2       | SW9  | 접점입력 | 2       | GND |                      |
| 3       | C01 | 공통접점 | 3       | C89  | 공통접점 |         |     | 입력전압                 |
| 4       | SW2 | 접점입력 | 4       | SW10 | 접점입력 |         |     | - 외부저항 없이 사용시        |
| 5       | SW3 | 접점입력 | 5       | SW11 | 접점입력 |         |     | 직류: ±3 ~ ±50VDC      |
| 6       | C23 | 공통접점 | 6       | C01  | 공통접점 |         |     | 교류: 3 ~ 50VAC        |
| 7       | SW4 | 접점입력 | 7       | SW12 | 접점입력 |         |     | - 외부저항 200 kΩ/2W 사용시 |
| 8       | SW5 | 접점입력 | 8       | SW13 | 접점입력 |         |     | 직류: ±50 ~ ±500VDC    |
| 9       | C45 | 공통접점 | 9       | C123 | 공통접점 |         |     | 교류: 50 ~ 500VAC      |
| 10      | SW6 | 접점입력 | 10      | SW14 | 접점입력 |         |     |                      |
| 11      | SW7 | 접점입력 | 11      | SW15 | 접점입력 |         |     |                      |
| 12      | C67 | 공통접점 | 12      | C145 | 공통접점 |         |     |                      |

확장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5 | P2.4       | GPIO                                  |
| 6 | P2.5       | GPIO                                  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | N.C                        |                                                          |
| 14 | N.C                        |                                                          |
| 15 | N.C                        |                                                          |
| 16 | N.C                        |                                                          |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로직용 전원출력                                      |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC)전원출력                                           |
| 24 | GND                        |                                                          |

```
typedef struct {
    MType          mtype;
    ID             id;
    DieInEvent*   in;
    DieOutEvent*  out;
} DIE;
```

DIE 모듈의 이미지는 위와 같은 DIE 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 탑입으로 Die 탑입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 접점입력은 DieInEvent 구조체에 저장됩니다. (V0.60 부터 DieInChangedEvent, DieOutChangedEvent 가 간략히 DieInEvent, DieOutEvent 로 명칭변경됨)

```
typedef struct {
    EventHead e;
    union {
        uint16 sw;
        struct {
            ELogic sw0      :1;
            ELogic sw1      :1;
            ELogic sw2      :1;
            ELogic sw3      :1;
            ELogic sw4      :1;
            ELogic sw5      :1;
            ELogic sw6      :1;
            ELogic sw7      :1;
            ELogic sw8      :1;
            ELogic sw9      :1;
            ELogic sw10     :1;
            ELogic sw11     :1;
            ELogic sw12     :1;
            ELogic sw13     :1;
            ELogic sw14     :1;
            ELogic sw15     :1;
        } sw;
    } sw;
}
```

```

    };
};

} DieInEvent;

```

DIE 모듈의 각 스위치 입력 상태는 1 비트에 저장되며 sw0 ~ sw15 의 16 개로 구성됩니다. 16 개의 입력을 동시에 다루고자 할 때는 2 바이트 크기의 변수 sw를 사용합니다.

예) 스위치 0 번이 눌러졌는지 검사하려면,  
`if (die->in->sw0 == ON) ...`

예) 현재 전체 스위치의 상태를 저장하려면,  
`uint16 temp = die->in->sw;`

```

typedef Event DieOutEvent;
typedef BaseParam DieParamEvent,DieParamChangedEvent,DieParamNotifiedEvent;

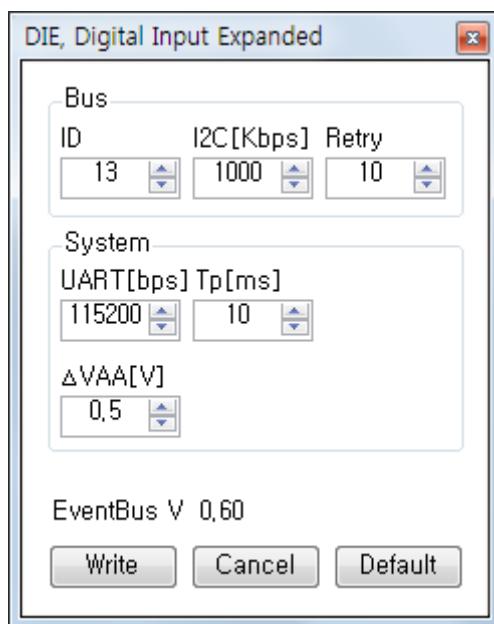
```

DIE 모듈은 입력모듈로써 출력이 없으므로 기본이미지를 그대로 재정의해서 사용합니다.

`DieParamEvent` 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. 기본 파라미터와 동일하므로 `BaseParam` 을 재정의하여 사용하며, `ezConfig.exe` 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(`die_drv.c`)에 정의된 `param` 변수를 사용해서 `param.dVAA` 와 같이 읽습니다.

#### ■ \_\_BASE\_PARAM

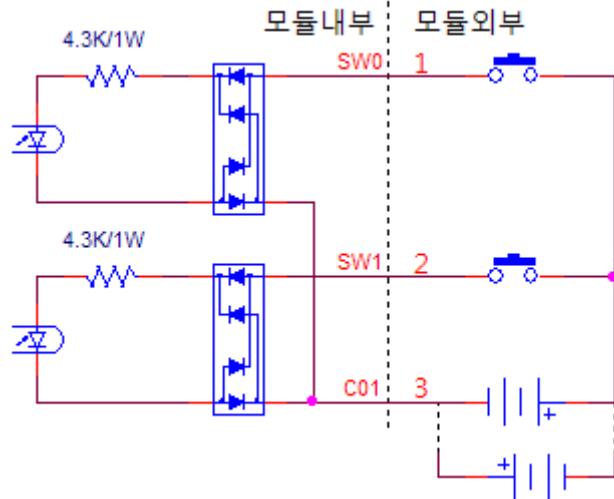
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고



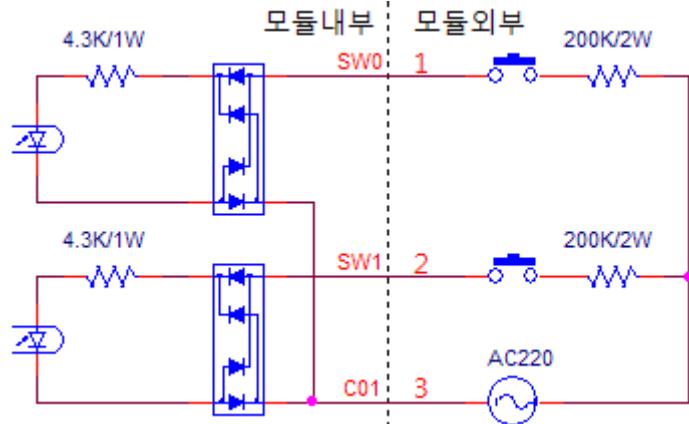
`ezconfig.exe` 의 설정창입니다. DIE 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 `DIE_PARAM` 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 `ezconfig.exe` 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.

### 6.7.1. 싱크(Sink)타입 스위치 입력

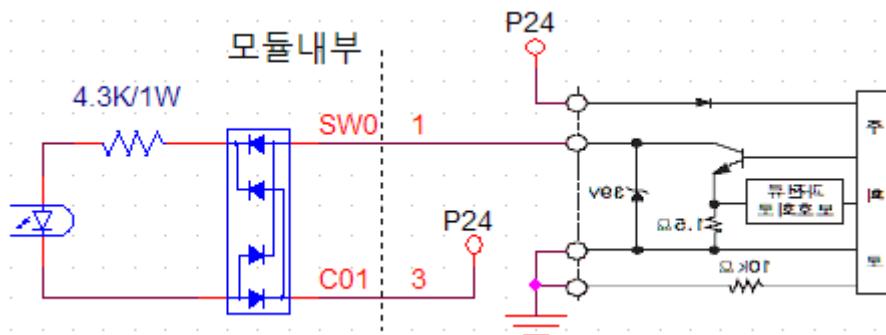
DIM, DIO 의 스위치 입력회로는 싱크(SINK) 타입이므로 핀에 입력전류를 공급할 때(또는 8V 이상의 전압이 공급될 때) ON 으로 인식하는 방식입니다. 반면, DI/E 입력회로는 그림처럼 입력단에 다이오드브릿지가 있으므로  $\pm 3 \sim \pm 50\text{VDC}$  의 양방향 전압뿐만 아니라  $\pm 3 \sim \pm 50\text{VAC}$ (그 이상의 전압인 경우 외부저항 사용) 전압도 인식할 수 있습니다. 또한 내부 동작전원과 완전히 분리되어 있어 외부에 별도의 절연이 필요치 않아 산업용으로 안정되게 사용할 수 있습니다.



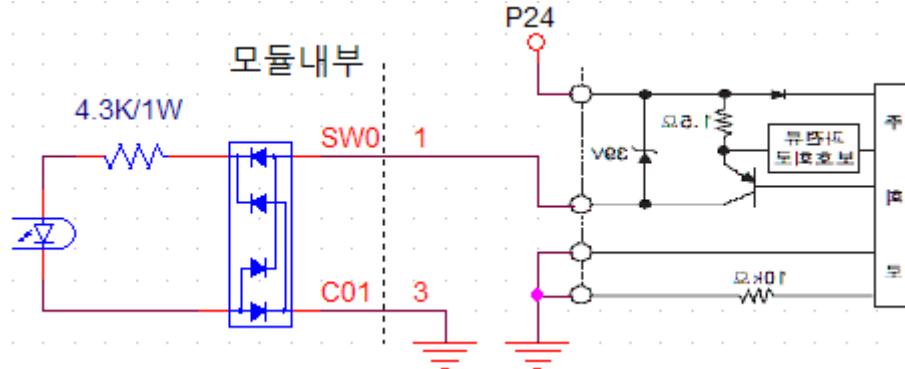
만일, 더 넓은 범위의 전압으로 접점을 구동하기 위하여는 외부에  $200\text{k}\Omega/2\text{W}$  저항을 사용하면,  $\pm 50 \sim \pm 500\text{VDC}$  의 양방향 전압뿐만 아니라  $50 \sim 500\text{VAC}$  전압도 인식할 수 있습니다.



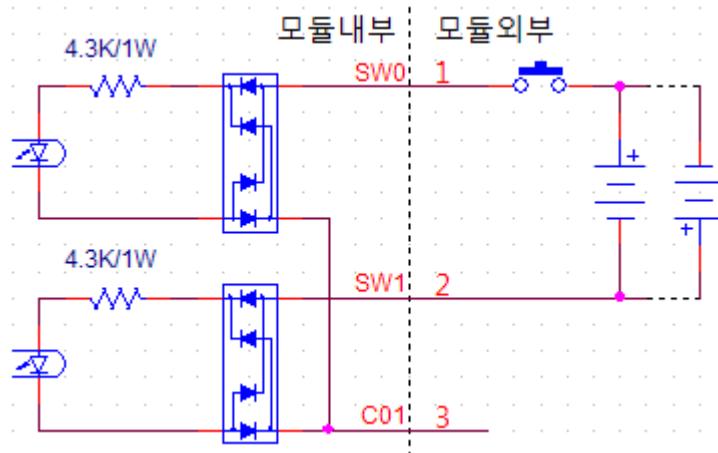
DI/E 입력회로는 매우 유연성있게 설계되어 거의 대부분의 경우 외부에 추가부품이 없어도 손쉽게 연결할 수 있습니다.



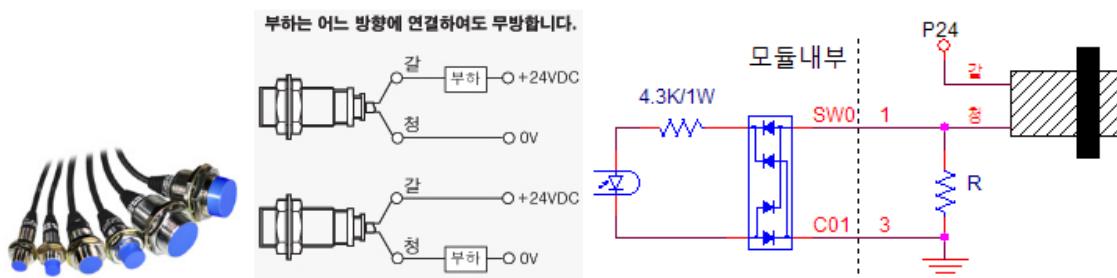
그림은 NPN 출력의 광센서를 연결한 예입니다. 그림처럼 공통단자(C01)는 전원에 연결하는 것이 좀 더 편리한 경우가 많습니다.



그림은 PNP 출력의 광센서를 연결한 예입니다.



그림처럼 스위치를 연결하면, 두 개의 입력이 동시에 온오프됩니다. 더블체크가 가능합니다.



그림은 오토닉스의 근접센서(스위치)입니다. 근접스위치는 내부에 스위치가 있다고 생각해도 무방합니다. 즉, 물체가 접근하면 두선에 연결된 스위치가 동작하므로, D1E에 연결하려면, 갈색선(V+)에 +24V를 연결하고, 청색선(V-)은 D1E 입력에 연결합니다.

하지만, 근접스위치는 특성상 스위치가 동작하지 않아도 흘려야 하는 최소전류(누설전류)가 있습니다. 누설전류가 입력단자(그림의 1 번핀)로 흘러들면 입력회로가 오동작 합니다. 그래서 외부에 블리더저항(R)을 연결해서 누설전류가 블리더저항으로 흐르도록 합니다.

단, 블리더저항에 누설전류가 흘러서 저항양단의 전압이 입력최소전압인 3[V]를 넘어서면 오인식 하므로,

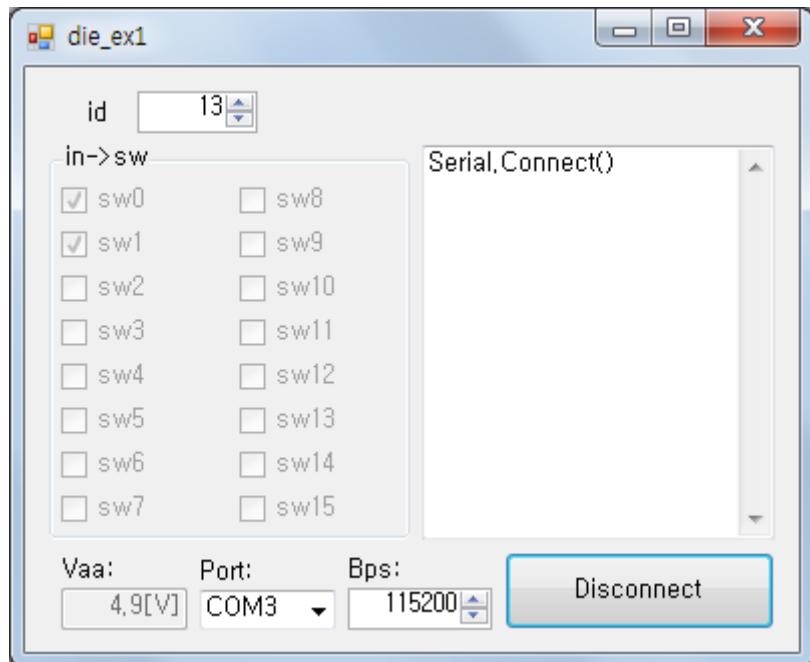
$$R(\text{블리더저항}) < (3[V] \div \text{누설전류})$$

누설전류가 0.7mA 인 경우 저항은 4.2KΩ 이하가 되므로, 안전하게 3KΩ 을 사용합니다. 또한, 근접센서는 온 되어도 양단에 전압이 완전히 0 이 되지 않습니다. 당연히 동작에 필요한 최소전압은 있어야 겠지요. 이것이 잔류전압입니다. 보통 데이터시트에는 3.5[V] 정도로 나옵니다.

즉, 전원이 24V 이므로, 센서가 온되면, IO 커넥터로 입력되는 전압은 잔류전압을 제외하면,

$$(24V - \text{잔류전압}) = (24V - 3.5V) = 21.5[V] \text{ 이므로,}$$

스위치를 인식하기 위한 최소전압인 3[V]보다 크므로 정상적으로 센서가 인식 됩니다.



`eventbus_dotnet_0.60\example\die_ex1\bin\Debug\die_ex1.exe` 예제프로그램을 실행하면 그림처럼 DIE 를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. 스위치를 동작하면 접점상태가 체크박스에 표시됩니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(`die_def.c` 참고)

## 6.8. DOE, Digital Output Expanded Module



경우에는 외부에 추가적인 대형릴레이를 설치하면 됩니다.

램프, 솔레노이드, 전등 등과 같은 온오프 출력장치를 구동하기 위한 16 채널의 릴레이 접점출력으로 구성되어 있습니다. 릴레이는 최대 5A 까지 전류를 스위칭 할 수 있으며 DC60V, AC240V 의 전압을 스위칭 할 수 있으므로 전등이나 AC 모터의 온오프제어에도 사용할 수 있습니다. 이보다 더 큰 용량의 부하를 제어할

접두사양

- Contact resistance : Max. 100mΩ at 6VDC 1A
  - Contact ratings : 3A 120VAC/24VDC
  - Switching Voltage : 240VAC/60VDC
  - Switching current : 5A

## IO 커넥터 핀사양

| IO1 커넥터 |     |          | IO2 커넥터 |      |             |
|---------|-----|----------|---------|------|-------------|
| 핀       | 명칭  | 기능       | 핀       | 명칭   | 기능          |
| 1       | RY0 | 접점출력     | 1       | RY8  | 접점출력        |
| 2       | RY1 | 접점출력     | 2       | RY9  | 접점출력        |
| 3       | C01 | 0,1 공통접점 | 3       | C89  | 8,9 공통접점    |
| 4       | RY2 | 접점출력     | 4       | RY10 | 접점출력        |
| 5       | RY3 | 접점출력     | 5       | RY11 | 접점출력        |
| 6       | C23 | 2,3 공통접점 | 6       | C01  | ,10,11 공통접점 |
| 7       | RY4 | 접점출력     | 7       | RY12 | 접점출력        |
| 8       | RY5 | 접점출력     | 8       | RY13 | 접점출력        |
| 9       | C45 | 4,5 공통접점 | 9       | C123 | 12,13 공통접점  |
| 10      | RY6 | 접점출력     | 10      | RY14 | 접점출력        |
| 11      | RY7 | 접점출력     | 11      | RY15 | 접점출력        |
| 12      | C67 | 6,7 공통접점 | 12      | C145 | 14,15 공통접점  |

화장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 5  | P2.4                       | GPIO                                                     |
| 6  | P2.5                       | GPIO                                                     |
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | N.C                        |                                                          |
| 14 | N.C                        |                                                          |
| 15 | N.C                        |                                                          |
| 16 | N.C                        |                                                          |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로 적용 전원 출력                                    |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC) 전원 출력                                         |
| 24 | GND                        |                                                          |

```
typedef struct {
    MType          mtype;
    ID             id;
    DoeInEvent*   in;
    DoeOutEvent*  out;
} DOE;
```

DOE 모듈의 이미지는 위와 같은 DOE 구조체로 되어 있습니다(eventbus.h 참조). mtype은 모듈의 타입으로 Doe 타입이 됩니다. id는 모듈의 주소이며 유일한 값을 가져야 합니다. 그리고, 릴레이 출력상태는 DoeOutEvent 구조체에 저장됩니다. (V0.60 부터 DoeInChangedEvent, DoeOutChangedEvent 가 간략히 DoeInEvent, DoeOutEvent로 명칭변경됨)

```
typedef Event DoeInEvent;
```

DOE 모듈은 출력모듈로 입력이 없으므로 기본이미지를 재정의해서 사용합니다.

```
typedef struct {
    EventHead e;
    union {
        uint16 rly;
        struct {
            ELogic rly0 :1;
```

```

    ELogic rly1 :1;
    ELogic rly2 :1;
    ELogic rly3 :1;
    ELogic rly4 :1;
    ELogic rly5 :1;
    ELogic rly6 :1;
    ELogic rly7 :1;
    ELogic rly8 :1;
    ELogic rly9 :1;
    ELogic rly10 :1;
    ELogic rly11 :1;
    ELogic rly12 :1;
    ELogic rly13 :1;
    ELogic rly14 :1;
    ELogic rly15 :1;
}
};

} DoeOutEvent;

```

DOE 모듈의 각 릴레이 출력 상태는 1 비트에 저장되며 rly0 ~ rly15 의 16 개로 구성됩니다.  
16 개의 출력을 동시에 다루고자 할 때는 변수 rly 를 사용합니다.

예) 1 번 릴레이를 ON 시키려면,

```
dom->out->rly1 = ON;
```

예) 전체릴레이를 모두 ON 시키려면,

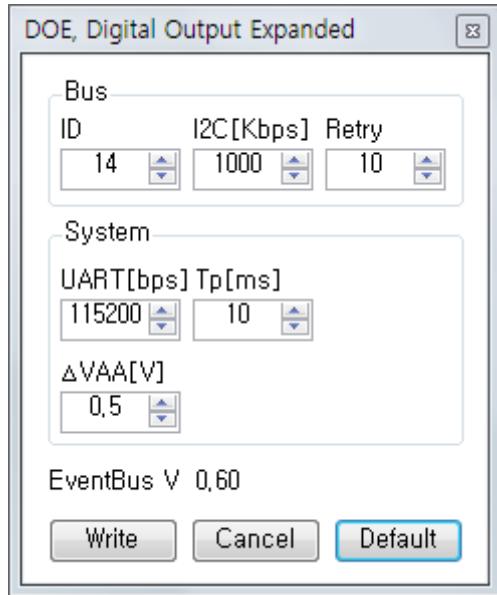
```
dom->out->rly = 0xFFFF;
```

```
|| typedef BaseParam DoeParamEvent,DoeParamChangedEvent,DoeParamNotifiedEvent;
```

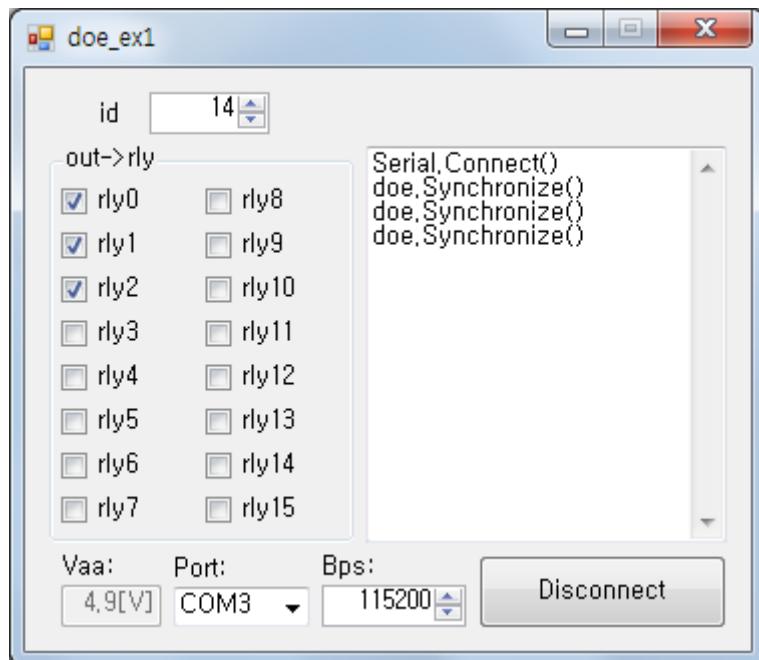
이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe  
프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면  
드라이버(doe\_drv.c)에 정의된 param 변수를 사용해서 param.dVAA 와 같이 읽습니다.

■ \_\_BASE\_PARAM

“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고

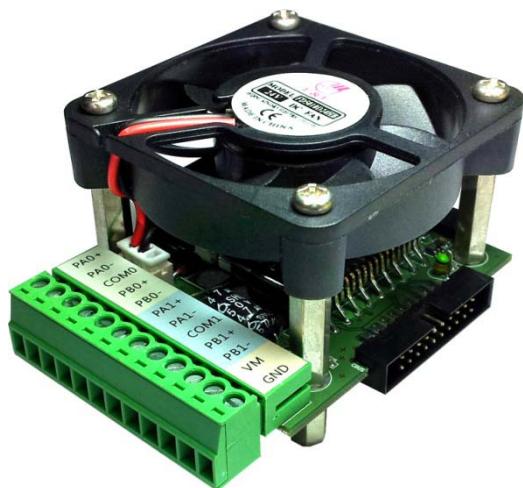


`ezconfig.exe` 의 설정창입니다. DOE 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 `DOE_PARAM` 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 `ezconfig.exe`에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.



`eventbus_net_0.60\example\doe_ex1\bin\Debug\doe_ex1.exe` 예제프로그램을 실행하면 그림처럼 DOE 를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. `rly0,1` 체크박스를 클릭하면 릴레이의 동작음을 들을 수 있습니다. V0.50 이후부터는 `Vaa`, `Vbb`, `Vdc` 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(`doe_def.c` 참고)

## 6.9. SMD, Step Motor Driver Module



두 개의 SLA7062M 스텝모터 전용 제어칩을 사용하여 16, 8, 4, 2 스텝으로 마이크로스테핑 동작이 가능하여 미소각으로 회전을 제어할 수 있으므로 정밀 액츄에이터의 용도로 사용할 수 있습니다. 또한 최대 3A 까지 모터전류를 훌릴 수 있으며, 모터전류를 프로그램으로 설정할 수 있으므로 목적에 맞는 최적의 전류를 런타임에 설정할 수 있습니다. 또한 온도센서, 방열판, 쿨링팬이 일체화되어 있으며, 쿨링팬 속도 제어회로가 기본 내장되어 있으므로 모터구동시 발생하는 열을 방열시키기 위해 필수적으로

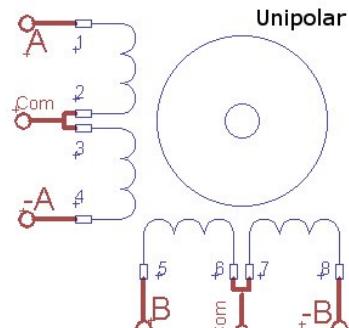
갖추어야하는 방열을 위한 별도의 추가없이 바로 시스템을 구성할 수 있습니다.

타사 제품처럼 별도의 방열을 위한 추가작업이 필요치 않으며, 제품의 높이도 타사제품의 절반이하로 매우 컴팩트하므로 제품에 내장시키기에 적합합니다.

모터 구동용 전원은 IO 커넥터로 외부에서 공급하여야 하는데, 저전류 구동의 경우는 PSM 모듈의 VBB 커넥터에서 공급받을 수 있으나, 공급전류에 제한이 있으므로 모터전류가 1[A] 이상인 경우 외부전원을 사용하여야 합니다. 전원은 모터정격전압의 4 배이상이 적합하며, 24V 가 일반적(모터정격은 6V 이하)이지만 경우에 따라서는 12V 이상의 전원으로도 동작시킬 수 있습니다. 그러나 최적의 고속 고토크 동작을 위하여는 24V 이상의 전원이 적절합니다.

IO 커넥터 핀사양

| 핀번호 | 명칭   | 기능            | 기타        |
|-----|------|---------------|-----------|
| 1   | PA0+ | A 상           | 모터 0/1 채널 |
| 2   | PA0- | A/ 상          |           |
| 3   | COM0 | COMMON        |           |
| 4   | PB0+ | B 상           |           |
| 5   | PB0- | B/ 상          |           |
| 6   | PA1+ | A 상           |           |
| 7   | PA1- | A/ 상          |           |
| 8   | COM1 | COMMON        |           |
| 9   | PB1+ | B 상           |           |
| 10  | PB1- | B/ 상          |           |
| 11  | VM   | 모터전원(외부에서 공급) |           |
| 12  | GND  | 모터전원 그라운드     |           |



24V (최대 46VDC)

확장커넥터 핀사양

| 핀 | 핀 명칭 | 핀 기능 |
|---|------|------|
|   |      |      |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 1  | P2.0/SSEL1                 | GPIO / SPI1 Slave Select                                 |
| 2  | P2.1/SCK1                  | GPIO / SPI1 Serial Clock                                 |
| 3  | P2.2/MISO1                 | GPIO / SPI1 Master Input Slave Output                    |
| 4  | P2.3/MOSI1                 | GPIO / SPI1 Master Output Slave Input                    |
| 5  | P2.4                       | GPIO                                                     |
| 6  | P2.5                       | GPIO                                                     |
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 13 | N.C                        | Not Connected                                            |
| 14 | N.C                        | Not Connected                                            |
| 15 | N.C                        | Not Connected                                            |
| 16 | N.C                        | Not Connected                                            |
| 12 | N.C                        | Not Connected                                            |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | 3.3V로직용 전원 출력                                            |
| 22 | GND                        |                                                          |
| 23 | VBB                        | 24V 전원 출력                                                |
| 24 | GND                        |                                                          |

```

typedef struct {
    MType      mtype;
    ID         id;
    SmdInEvent* in;
    SmdOutEvent* out;
} SMD;

```

SMD 모듈의 이미지는 위와 같은 SMD 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Smd 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력이미지는 SmdInEvent 구조체에, 출력이미지는 SmdOutEvent 구조체에 저장됩니다. (V0.60 부터 SmdInChangedEvent, SmdOutChangedEvent 가 간략히 SmdInEvent, SmdOutEvent 로 명칭변경됨)

```

typedef struct {
    EventHead e;
    float     vm;           // Step Motor Driving voltage

```

```

int16      heat;      // Thermistor 측정온도
union {
    struct {
        Bool     fan       :1;      // 냉각팬 고장상태, LSB
        Bool     busy0     :1;      // 모터0 동작상태
        Bool     busy1     :1;      // 모터1 동작상태
    };
    uint16  status;
};
} SmdInEvent;

```

SMD 모듈의 입력으로는 IO 커넥터 11,12 번 핀으로 공급되는 모터구동용 외부전원의 전압인 `vm` 과 쿨링팬 속도제어용 온도센서의 온도인 `heat` 와 동작상태를 나타내는 상태플래그인 `status` 가 있습니다. `fan` 은 모터구동 칩이 설정온도(`ezconfig.exe`로 설정한 `upperTemp` 온도)를 넘을 때 참이 되며, `busy0`, `busy1` 은 각각의 모터가 이전의 명령을 수행중인지 종료했는지를 알려줍니다. `busy` 신호를 이용하면 이전 명령과 중복되지 않게 순차적으로 구동명령을 보낼 수 있습니다.

- 예) 모터 1이 정지상태인지 확인하려면,  
`if (smd->in->busy1 == OFF) ...`
- 예) 모터구동 칩의 온도가 60 도를 넘었는지 검사하려면,  
`if (smd->in->heat > 60) ...`

```

typedef struct {
    union {
        struct {
            SMD_CMD    cmd      :4;      // LSB
            SMD_STEP   step     :2;
            SMD_DIR    dir      :1;
            Bool       run      :1;
        };
        uint16  config;
    };
    uint16  ipps;      // initial PPS
    uint16  accel;     // accel,decel speed. [pulse/ms]
    uint32  pps;       // steady state speed [pulse/second]
    uint32  move;      // total pulse of rotation.
} SMD_SETTING;

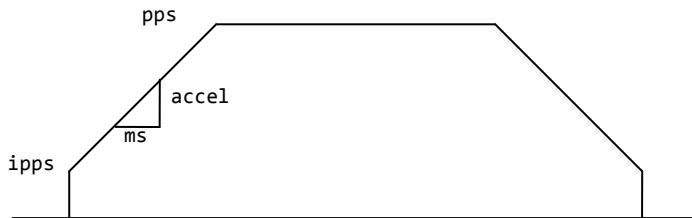
typedef struct {
    EventHead    e;
    SMD_SETTING m0;
    SMD_SETTING m1;
    uint16       amp;     // 0(free mode), amp[A] = 모터전류 * 10
} SmdOutEvent;

```

SMD 모듈의 출력이미지에는 각 모터에 대한 명령과 모터전류를 설정하기 위한 `amp` 가 있습니다. `amp` 는 0.1[A] 단위이므로 헐리고자 하는 모터전류에 10 을 곱한 값을 넣어야 합니다. 예로 1[A]의 전류를 헐리려면 `smd->out->amp = 10;` 으로 해야 합니다.

\*주의\* 단 SMD 모듈은 m0, m1 두개의 모터를 구동할 수 있지만, 전류 설정은 공통입니다. 모터에 전류를 차단하려면 m0, m1 에 동시에 FREE 명령을 보내야 합니다. 한쪽 모터에만 FREE 명령을 보내면 그 모터는 자동으로 HOLD 명령으로 대체됩니다.

다음으로 각 모터의 명령은 SMD\_SETTING 구조체로 되어있습니다. SMD 모듈은 사다리꼴 가감속 프로파일을 지원합니다. ipps 는 초기속도(0 보다 커야함. 보통 100~1000 을 사용), accel 는 밀리초(ms)당 가감속할 펄스개수이며, pps 는 정속구간의 회전속도를 나타내며, move 는 MOVE 명령에서의 회전할 펄스의 총개수입니다.



- ipps = 1000, accel = 10, pps = 10,000, move = 200,000 인 경우

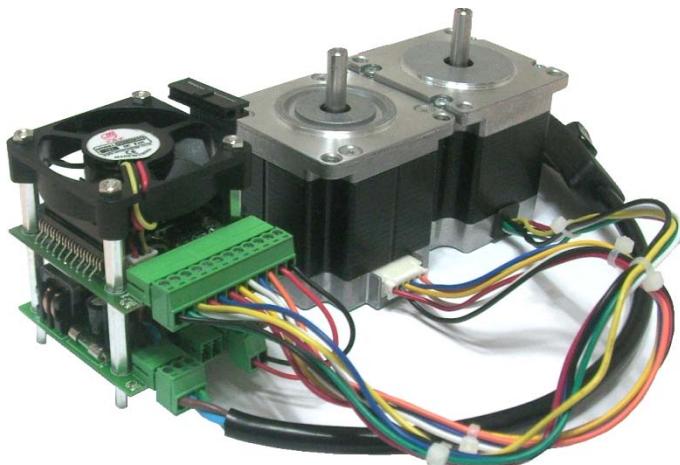
$$\text{가감속구간}(\Delta\text{pps}) = \text{pps} - \text{ipps} = 9000[\text{pps}]$$

$$\text{가감속시간}(\text{Taccel}) = \Delta\text{pps} / \text{accel} = 900[\text{ms}]$$

$$\text{가감속구간 펄스수}(\text{pacc}) = \text{Taccel} * \Delta\text{pps} / 2 = 4050[\text{pulse}]$$

$$\text{정속구간펄스수}(\text{pulse}) = \text{move} - 2*\text{pacc} = (200000 - 2*4050) = 191,900[\text{pulse}]$$

$$\text{정속구간 소요시간} = \text{pulse} / \text{pps} = 191900 / 10000 = 19.19[\text{s}]$$



```

typedef enum {FREE, HOLD, STOP, RUN, MOVE} SMD_CMD, MOC_CMD, USD_CMD;
typedef enum {CW, CCW} SMD_DIR, MOC_DIR, USD_DIR;
typedef enum {SIXTEENTH, QUARTER, EIGHTH, HALF} SMD_STEP;

```

dir 은 시계방향과 반시계방향의 회전방향을 설정하고, step 은 마이크로 스텝핑 구동방법을 설정합니다. 스텝당 1.8 도 회전하는 일반적인 모터 기준으로 SIXTEENTH(16 분주, 펄스당 0.1125 도 회전, 1 회전에 3200 펄스가 필요)는 부드럽고 미세한 회전에 적합하고, HALF(펄스당 0.9 도 회전, 1 회전에 400 펄스가 필요)는 고속회전에 적합합니다. cmd 는 모터의 동작방법을

설정하는데, FREE 는 모터에 전류가 흐르지 않는 상태로 모터샤프트는 자유자재로 움직입니다. HOLD 는 전류를 흘리면서 정지한 상태인데, 외부에서 힘을 가해도 정지상태를 유지합니다. RUN 은 가속후 정속도로 계속회전하므로 정지하기 위하여는 FREE, HOLD 로 즉시정지 하거나, STOP 명령으로 감속정지할 수 있습니다. MOVE 는 일정 펄스만큼만 회전(이동)할 때 필요합니다.

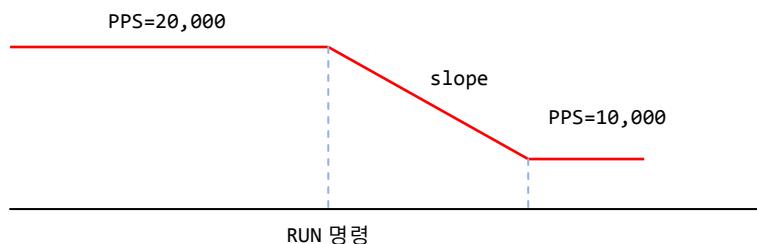
마지막으로 run 은 명령이 유효한지를 나타냅니다. 모든 설정을 마쳤더라도 run 0| FALSE 이면 이 명령은 실행되지 않습니다.

- 모터 0 은 현상태를 유지하고, 모터 1 에만 명령을 보내는 경우,

```
smd->out->m0.run = FALSE;
smd->out->m1.run = TRUE;
```

- 모터전류를 차단하려면(FREE)

```
smd->out->m0.run = smd->out->m1.run = TRUE;
smd->out->m0.cmd = smd->out->m1.cmd = FREE;
```



SMD 는 부드러운 회전을 위하여 자동 가감속의 연속동작을 지원합니다. 즉, 그림처럼 20Kpps 로 회전중 10Kpps 의 RUN 명령을 받으면 설정한 slope 에 따라 부드럽게 가감속함으로써 충격없이 부드러운 회전을 할 수 있습니다.

```
typedef struct {
    __BASE_PARAM;
    float      dVM;           // 모터구동전압 변동검출 기준치
    uint8      dTEMP;          // 온도변동 검출 기준치
    uint8      upperTemp;       // 팬제어 목표온도, 팬듀티 100%
    uint8      lowerTemp;       // 팬제어 시작온도, 팬듀티 50%
} SmdParamEvent,SmdParamChangedEvent,SmdParamNotifiedEvent;
```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(smd\_drv.c)에 정의된 param 변수를 사용해서 param.dVM 와 같이 읽습니다.

- \_\_BASE\_PARAM

“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고

- dVM

외부에서 공급하는 모터구동용 전원의 전압변동을 검사하는 판단 기준.

- dTEMP

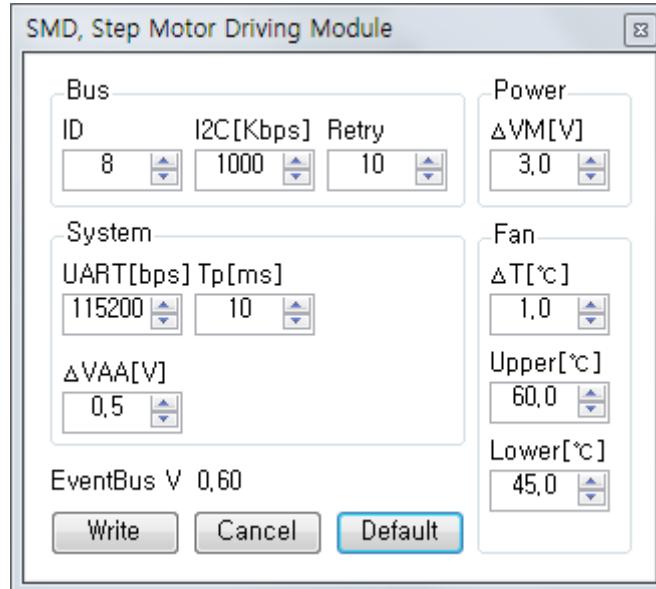
모터구동칩 온도의 변동을 검사하는 판단기준.

■ `upperTemp, lowerTemp`

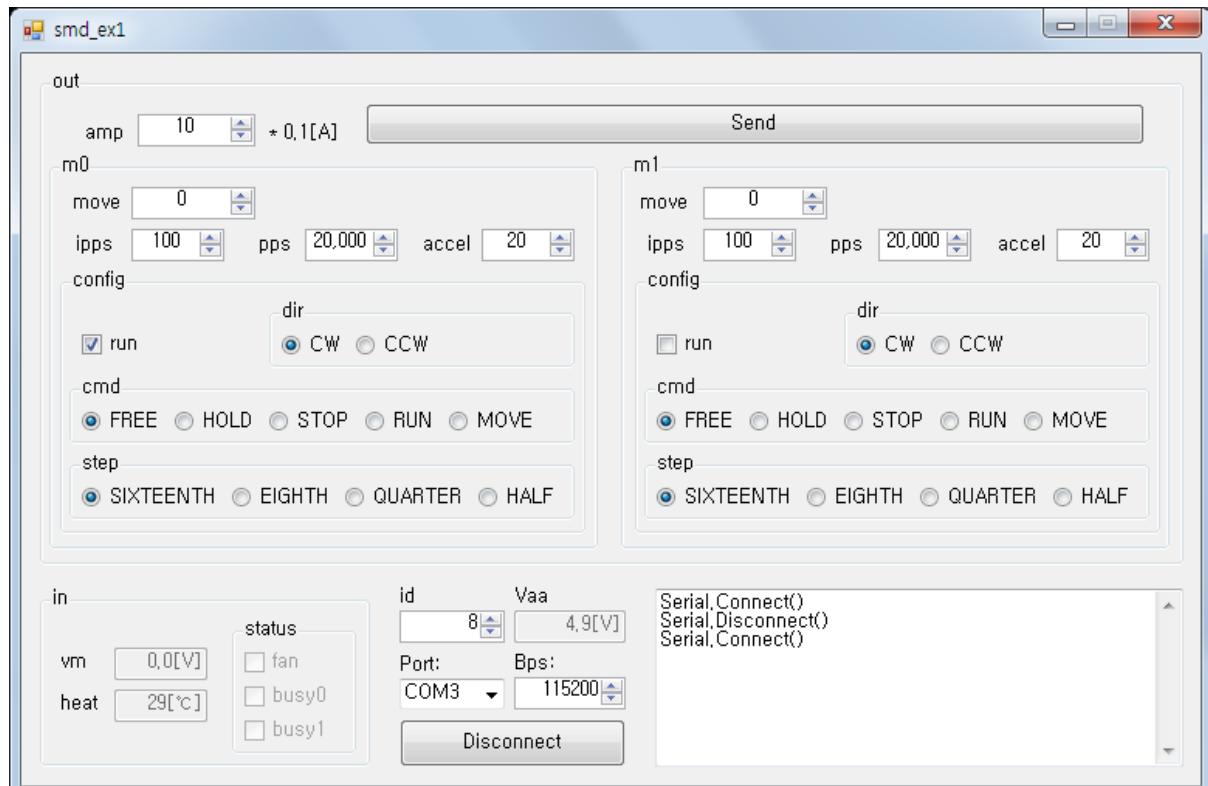
쿨링팬 제어를 위한 온도의 상하한치.

`lowerTemp`에서 회전을 시작하고, `upperTemp`에서 최대속도로 쿨링팬이 동작.

온도가 `upperTemp`를 넘으면 `status`의 `fan` 플래그가 ON 되고 모터전류는 차단된다.



`ezconfig.exe`의 설정창입니다. SMD 모듈을 ColinkEx 또는 UUM으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 SMD\_PARAM과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM에 영구 저장됩니다. V0.60부터 컴퓨터와 연결속도를 `ezconfig.exe`에서 설정할 수 있으며, 기본속도는 115,200bps입니다.



eventbus\_dotnet\_0.60\example\smd\_ex1\bin\Debug\smd\_ex1.exe 예제프로그램을 실행하면 그림처럼 SMD를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps를 사용합니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(smd\_def.c 참고)

## 6.10. USB, USB Communication Module



PC 의 USB 포트에 연결하여 이벤트버스에 연결된 다른 모듈들을 제어할 수 있습니다. 이 경우 PC 도 마치 하나의 모듈처럼 동작하게 되는데 PC 용으로 제공되는 `eventbus_net` 라이브러리를 이용하면 이벤트버스에 연결된 모든 모듈들의 동작이나 데이터를 모니터링하거나 제어할 수 있습니다.

이런 기능을 이용하여 PC 용 시스템 운영 프로그램을 개발하면 하드웨어에 대한 매우 기초적인 지식만으로도 운용프로그램을 작성할 수 있으므로 놀라운 생산성을 제공합니다.

추가적인 기능으로 PC 의 USB 전원을 VAA, VBB 전원으로 변환하여 이벤트버스로 제공할 수 있어 PC 가 켜져있는 환경이라면 PSM 모듈과 같은 별도의 전원이 없어도 동작할 수 있습니다.

물론, 이벤트버스의 전원 시스템은 **병렬동작**이 가능하므로 PSM 을 동시에 사용하면 더 큰 전류를 사용할 수 있습니다.

확장커넥터 핀사양

| 핀  | 핀 명칭                       | 핀 기능                                                     |
|----|----------------------------|----------------------------------------------------------|
| 1  | P2.0/SSEL1                 | GPIO / SPI1 Slave Select                                 |
| 2  | P2.1/SCK1                  | GPIO / SPI1 Serial Clock                                 |
| 3  | P2.2/MISO1                 | GPIO / SPI1 Master Input Slave Output                    |
| 4  | P2.3/MOSI1                 | GPIO / SPI1 Master Output Slave Input                    |
| 5  | P2.4                       | GPIO                                                     |
| 6  | P2.5                       | GPIO                                                     |
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.1/AD2/CT32B1_MAT0       | GPIO/ADC2 / Match output 0 for 32-bit timer 1            |
| 14 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 15 | P1.10/AD6/CT16B1_MAT1      | GPIO / ADC6 / Match output 1 for 16-bit timer 1          |
| 16 | P1.11/AD7                  | GPIO / ADC7                                              |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | 3.3V 로직용 전원 출력                                           |
| 22 | GND                        |                                                          |
| 23 | VBB                        | 25V 전원 출력                                                |
| 24 | GND                        |                                                          |

```
|| typedef struct {
```

```

    MType      mtype;
    ID         id;
    UsbInEvent*   in;
    UsbOutEvent*  out;
} USB;

```

USB 모듈의 이미지는 위와 같은 USB 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Usb 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. (V0.60 부터 UsbInChangedEvent, UsbOutChangedEvent 가 간략히 UsbInEvent, UsbOutEvent 로 명칭변경됨)

```

typedef Event UsbInEvent;
typedef Event UsbOutEvent;

```

USB 모듈은 입출력이 없으므로 기본이미지를 사용합니다.

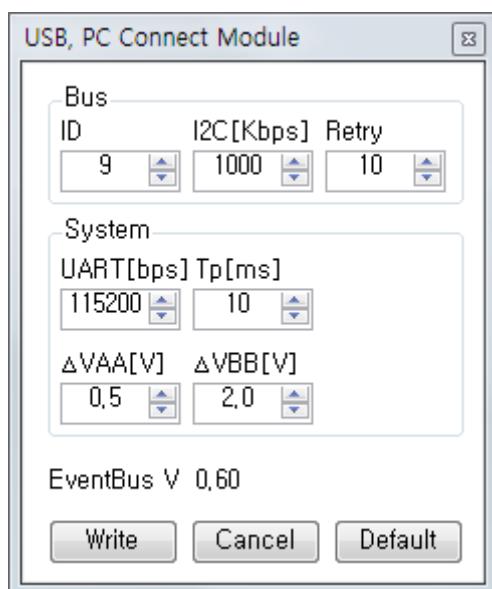
```

typedef struct {
    __BASE_PARAM;
    float      dVBB;
} UsbParamEvent,UsbParamChangedEvent,UsbParamNotifiedEvent;

```

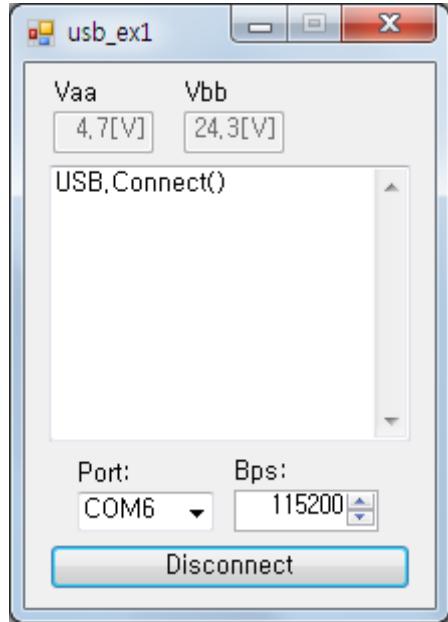
이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어에서 이 값을 참조하려면 드라이버(usb\_drv.c)에 정의된 param 변수를 사용해서 param.dVBB 와 같이 읽습니다.

- \_\_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- dVBB  
\_\_BASE\_PARAM 의 dVAA 참고.

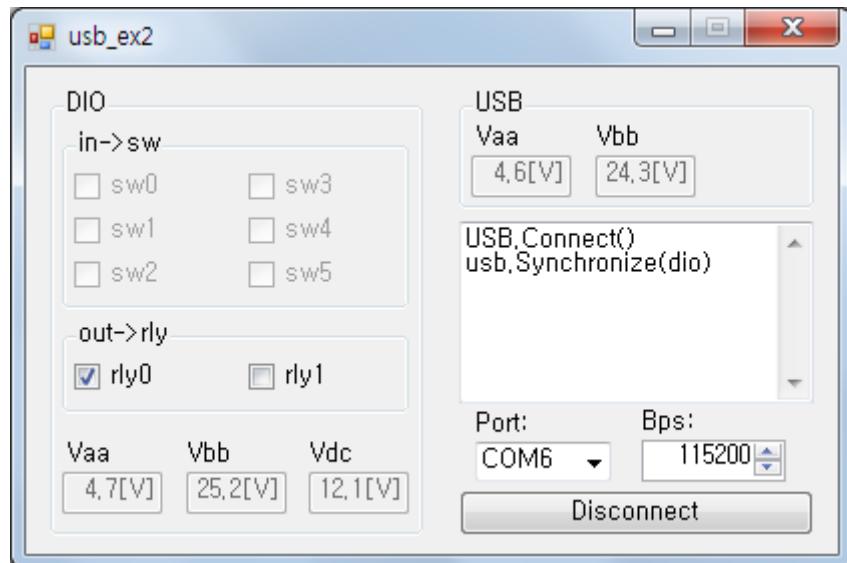


ezconfig.exe 의 설정창입니다. USB 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 UsbParamEvent 과 동일한 것을 볼 수

있습니다. 이 값들은 EEPROM에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 ezconfig.exe에서 설정할 수 있으며, 기본속도는 115,200bps입니다.



eventbus\_dotnet\_0.60\example\usb\_ex1\bin\Debug\usb\_ex1.exe 예제프로그램을 실행하면 그림처럼 USB를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps를 사용합니다. USB는 자체에 입출력을 가지고 있지 않으므로 전압외에는 달리 표시할 것이 없습니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(usb\_def.c 참고)



eventbus\_dotnet\_0.60\example\usb\_ex2\bin\Debug\usb\_ex2.exe 예제프로그램을 실행하면 그림처럼 USB를 통해서 연결된 DIO를 제어할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps를 사용합니다. USB에 컴퓨터를

연결하고 컴퓨터에서 DIO 의 입출력을 읽고 쓸수 있습니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(usb\_def.c 참고)

## 6.11. ENV, Environmental Sensing Module



ENV 는 환경센서입니다. 주변환경에서 얻을 수 있는 정보를 일괄적으로 센싱하기 위하여 4 개의 센서를 내장하고 있습니다. 첫째 온도센서는  $-20 \sim +100^{\circ}\text{C}$  까지 검출할 수 있습니다. 둘째 적외선센서는 사람이나 동물의 움직임을 검출하여 외부기기를 자동으로 동작하도록 프로그램 할 수 있습니다. 셋째 조도센서는 주위의 밝기를 검출하여 전등을 자동으로 점멸할 수 있습니다. 넷째 박수센서가 있습니다. 박수센서는 박수소리와 같은 충격있는 소리를 검출하여 횟수를 카운트합니다. 일반적인 음악소리 또는 주변의 자동차소리와 같은 강한 충격이 없는 소음은 카운트하지 않습니다. 이를 이용하면 박수소리로 원거리에서 전등을 온오프하는 등의 재미있는 기능을 구현할 수 있습니다.

사용예로 학생이 불을 켜둔채로 책상에서 자고 있는지를 검출하여 깨우는 경우를 생각해 보면, 학생이 불을 켜둔 상태에서 일정시간 움직이지 않는다면 자고있다고 판단하고 사이렌을 울리는 등의 잠을 깨울 수 있는 알람을 실행하고, 알람을 정지시키기 위해서 학생은 박수를 쳐서 깨어있음을 알리면 알람이 정지되도록 프로그램 할 수 있습니다. 그래서 DIO 모듈을 통하여 부저를 울리고 선풍기도 돌리고 전등도 점멸시켜 단계적으로 수험생의 주위를 환기시킬 수도 있습니다. 또한 온도센서로 감지된 온도를 이용하여 선풍기를 회전하는 것과 같이 재미있는 동작을 구현할 수 있습니다. 또한 박수소리의 횟수에 따라 동작하도록 리모콘기능을 만들수 있습니다.

확장커넥터 EXT5

| 핀번호 | 핀 명칭                           | 핀 기능                                                                                   |
|-----|--------------------------------|----------------------------------------------------------------------------------------|
| 1   | SWDIO<br>/P1.3/AD4/CT32B1_MAT2 | Serial wire debug IO / GPIO / AD Converter Input 4 / Match output 2 for 32-bit timer 1 |
| 2   | SWCLK/<br>P0.10/CT16B0_MAT2    | Serial wire clock / GPIO / Match output 2 for 16-bit timer 0                           |
| 3   | TXD<br>/P1.7/CT32B0_MAT1       | Transmitter output for UART / GPIO / Match output 1 for 32-bit timer 0.                |
| 4   | RXD<br>/P1.6/CT32B0_MAT0       | Receiver input for UART / GPIO / Match output 0 for 32-bit timer 0                     |
| 5   | GND                            |                                                                                        |

```

typedef struct {
    MType          mtype;
    ID             id;
    EnvInEvent*   in;
    EnvOutEvent*  out;
} ENV;

```

ENV 모듈의 이미지는 위와 같은 ENV 구조체로 되어 있습니다(eventbus.h 참조). `mtype` 은 모듈의 타입으로 Env 타입이 됩니다. `id` 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 센서입력은 `EnvInEvent` 구조체에 저장되며, 입력모듈이므로 출력은 없으며 `EnvOutEvent` 구조체는 내용이 없습니다. (V0.60 부터 `EnvInChangedEventArgs`, `EnvOutChangedEventArgs` 가 간략히 `EnvInEvent`, `EnvOutEvent` 로 명칭변경됨)

```
||| EventHead e;
int16    temp;      // signed -128 ~ +127°C
uint16   uv;        // infrared
uint32   lux;       // Lux
uint16   hand;      // Binary Code
} EnvInEvent;
```

ENV 모듈의 입력이미지는 4 개의 센서값이 있습니다. `temp` 는 온도센서로 검출한 영하에서 영상의 온도값을 가지며, 인체감지센서의 값은 `uv` 에 저장됩니다. `uv` 값은 V0.50 이전에는 변화치를 검출하였으나 V0.50 부터는 센서의 원래값을 그대로 출력합니다. 이값은 평상시 대략 7500 정도이지만 인체감지시 값이 변화(증가 또는 감소)합니다. `lux` 에는 주위의 밝기인 조도가 저장되고, `hand` 에는 박수소리와 같은 충격음파의 개수를 카운트한 값이 저장됩니다. `hand` 에는 연속된 박수소리의 횟수가 저장되는데, 일정시간(ezconfig.exe 에서 설정한) 동안 무음인 경우 0부터 다시 카운트하게 됩니다(V0.60 부터는 0 은 출력되지 않습니다).

- 온도를 검사하려면,  
`if (env->in->temp > 30) ...`
- 연속 박수회수에 따라 전등을 ON/OFF 하려면,  
`switch (env->in->hand) {
 case 2: dio->out->rly0 = ON; break;
 case 3: dio->out->rly0 = OFF; break;
}`

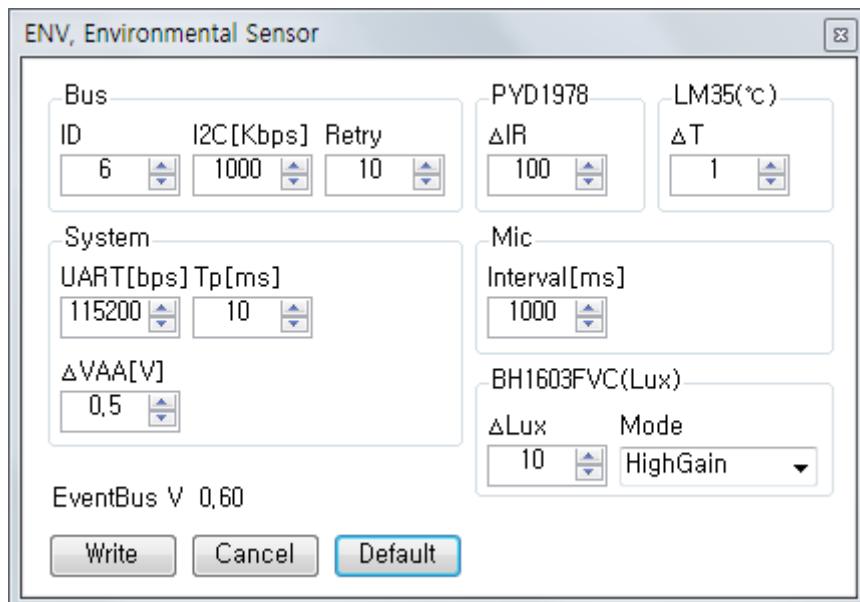
```
||| typedef Event EnvOutEvent;
```

ENV 모듈은 입력모듈로 출력이 없으므로 기본이미지를 사용합니다.

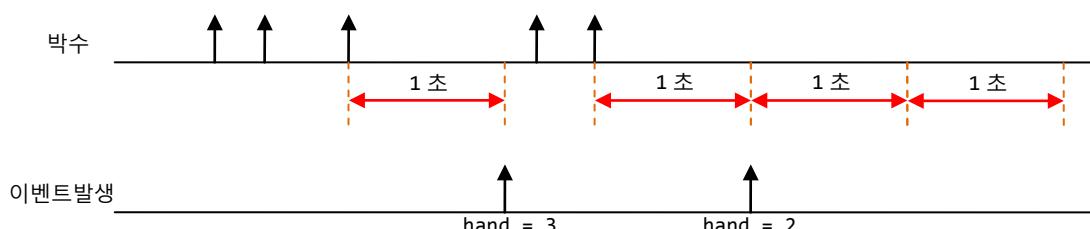
```
||| typedef struct {
    __BASE_PARAM;
    uint8    dTEMP;
    uint8    dUV;
    uint8    dLUX;
    uint8    lux_mode;
    uint16   mic_interval;
} ENV EnvParamEvent, EnvParamChangedEvent, EnvParamNotifiedEvent;
```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(env\_drv.c)에 정의된 `param` 변수를 사용해서 `param.dTEMP` 와 같이 읽습니다.

- \_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- dTEMP  
온도의 변동 여부를 판단할 기준이 되는 값.
- dUV  
인체감지 센서의 변동여부를 판단할 기준이 되는 값.
- dLUX  
조도센서의 변동여부를 판단할 기준이 되는 값.
- lux\_mode  
BH1603VC 조도센서는 증폭도를 조절하는 모드가 있다. High Gain 모드는 최대 1,000Lux, Middle Gain 모드는 최대 10,000Lux, Low Gain 모드는 최대 100,000Lux 까지 검출할 수 있다.
- mic\_interval  
박수소리 카운터를 리셋하는 시간. 마지막 박수소리 이후 설정시간이 경과하면 카운터는 0 이 되어 처음부터 다시 카운트한다.



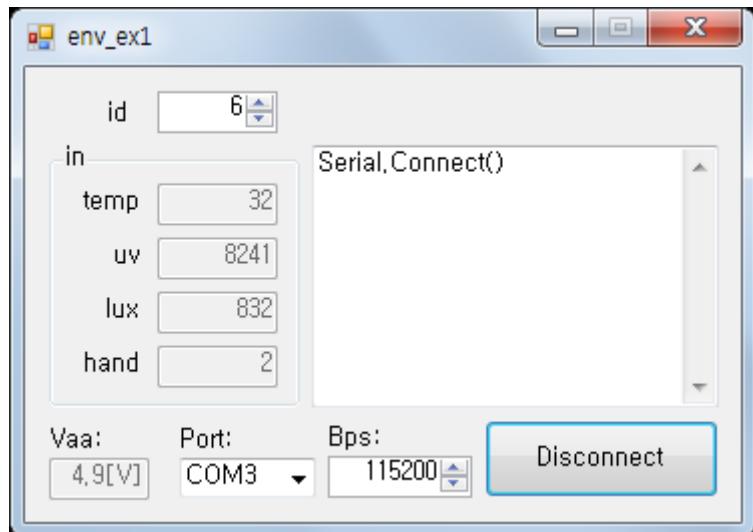
ezconfig.exe 의 설정창입니다. ENV 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 EnvParamEvent 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 ezconfig.exe 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.



マイク에서 감지한 박수소리의 횟수를 변수 hand에 저장하고 이벤트를 발생합니다. 이때, mic\_interval에서 설정한 간격(위의 경우 1 초)동안 소리를 감지하지 못하면 박수가 끝난 것으로 간주하고 카운트한 횟수를 hand에 저장하고 이벤트를 발생합니다.

그림에서 박수를 3회 친 후 1초동안 박수를 감지하지 못하였으므로 hand=3으로 이벤트를 발생하였으며, 그 다음 이벤트는 마찬가지로 hand=2로 이벤트를 발생합니다. 이전버전에서는 박수소리를 감지하지 못한 채 또 1초가 흐르면 hand=0으로 이벤트가 발생하였으나, V0.60에서는 hand=0은 이벤트를 발생하지 않도록 변경되었습니다.

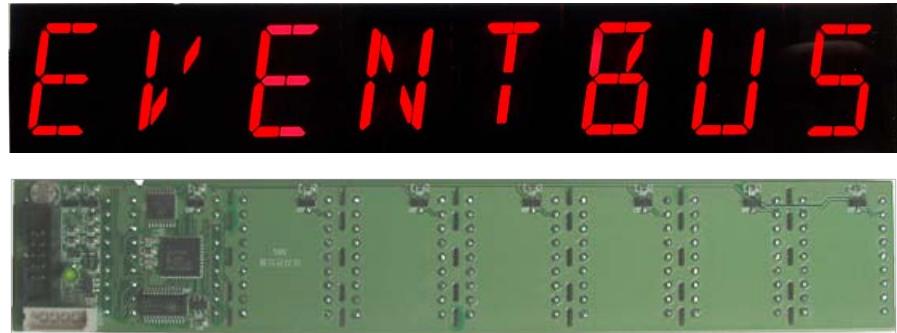
이런 방법으로 박수소리로 숫자를 입력할 수 있습니다.



`eventbus_net_0.60\example\env_ex1\bin\Debug\env_ex1.exe` 예제프로그램을 실행하면 그림처럼 ENV를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps를 사용합니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다.

## 6.12. SEG, 16- Segment Display Module

SEG 모듈은 8 자리의 영문자, 숫자, 특수문자 등을 표시할 수 있으며, 자릿수를 확장하려면 여러개의 SEG 모듈을 서로 연결하면 되고, 자릿수를 줄이려면 각 자리의 사이에 있는 홈에서 자르면 됩니다. 표시되는 문자를 왼쪽으로 스크롤 하는 기능도 내장되어 있습니다.



광학필터 부착후 이미지

SEG 모듈은 매우 편리하게 사용될 수 있습니다. 예를 들면, 생산라인의 생산량의 표시, 특정라인의 동작상태 표시, 가격표시등 매우 다양한 분야에 활용될 수 있습니다.

또한 여려개를 가로로 연결하면 긴 문장을 표시할 수도 있으며, 세로로 연결하여 여러 줄의 문장을 표시할 수도 있습니다.

CDS 센서도 내장하여 주변 밝기에 따라 적절히 동작하도록 쉽게 구현할 수 있습니다.

**\*주의\*** 마이컴이 LPC1114FHN33/20x(SRAM 4K)이므로 유저 펌웨어를 추가할 경우 램사용이 4K를 넘지 않도록 유의할 것. 가능하면 유저 펌웨어는 다른 모듈에서 작성할 것.

확장커넥터 EXT5

| 핀번호 | 핀 명칭                           | 핀 기능                                                                                   |
|-----|--------------------------------|----------------------------------------------------------------------------------------|
| 1   | SWDIO<br>/P1.3/AD4/CT32B1_MAT2 | Serial wire debug IO / GPIO / AD Converter Input 4 / Match output 2 for 32-bit timer 1 |
| 2   | SWCLK/<br>P0.10/CT16B0_MAT2    | Serial wire clock / GPIO / Match output 2 for 16-bit timer 0                           |
| 3   | TXD<br>/P1.7/CT32B0_MAT1       | Transmitter output for UART / GPIO / Match output 1 for 32-bit timer 0.                |
| 4   | RXD<br>/P1.6/CT32B0_MAT0       | Receiver input for UART / GPIO / Match output 0 for 32-bit timer 0                     |
| 5   | GND                            |                                                                                        |

```
typedef struct {
    MType          mtype;
    ID             id;
    SegInEvent*   in;
    SegOutEvent*  out;
```

```
    } } SEG;
```

SEG 모듈의 이미지는 위와 같이 구조체로 되어 있습니다(`eventbus.h` 참조). `mtype`은 모듈의 타입으로 `Seg` 타입이 됩니다. `id`는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력은 `SegInEvent` 구조체에, 출력은 `SegOutEvent` 구조체에 저장됩니다. (v0.60 부터 `SegInChangedEvent`, `SegOutChangedEvent` 가 간략히 `SegInEvent`, `SegOutEvent`로 명칭변경됨)

```
typedef struct {
    EventHead e;
    uint16    cds;
} SegInEvent;
```

SEG 모듈의 입력에는 밝기를 검출하기 위한 CDS 셀이 있어서 주변의 밝기를 `cds`에 저장합니다. 이 값은 오차가 매우 크기 때문에 대략적인 밝기 정도만 짐작하는 용도로 사용합니다. 좀 더 정확한 조도가 필요하면 ENV 모듈의 조도센서 값을 이용해야 합니다.

```
typedef struct {
    EventHead e;
    char      text[SEG_CHARS]; // 최대 8문자까지
} SegOutEvent;
```

출력이미지에는 위에서처럼 출력할 8 문자만으로 구성되어 있습니다. 이전 버전처럼 8 문자 이상의 긴 문자열은 표시할 수 없으며 문자스크롤 기능도 더 이상 지원하지 않습니다. 이런 기능은 8 문자의 내용을 반복적으로 변경하는 방법을 사용하면 됩니다. 그리고, 0x80(MSB가 ‘1’) 보다 큰 아스키코드는 표시할 수 없습니다. 그대신 MSB가 ‘1’인 문자는 세그먼트의 소수점(점문자)을 ON 시킵니다. 예를들어, 0x30(0b00110000)은 문자 ‘0’인데, MSB를 ON 시킨 0xb0(0b10110000)은 ‘0.’으로 표시됩니다.

```
void seg_sprintf(char* text, const char *pFormat, ...)
```

그리고, SEG에 문자를 표시하는 용도로 `seg_sprintf()`함수가 제공됩니다. 사용법은 `sprintf()`함수와 동일하지만, 8 문자보다 큰 경우 8 문자로 제한시키고, 8 문자보다 작은 경우 공백을 추가하는 기능이 있습니다.

- 세그먼트에 “hello”를 표시하려면,  
`seg_sprintf(this->out->text, “hello”);`
- CDS 가 변할 때 세그먼트에 “CDS:25”처럼 표시하려면,  
`IF(this, SegIn) {  
 seg_sprintf(this->out->text, "CDS:%d", pe->cds);  
}}`
- “CDS.25”처럼 ‘S’에 점을 찍으려면(점은 별도의 문자로 표시되는 것이 아니라 세그먼트의 DP를 사용해서 ‘S.’으로 표시하므로 총 5 글자)  
`IF(this, SegIn) {  
 seg_sprintf(this->out->text, "CDS:%d", pe->cds);  
 this->out->text[2] |= 0x80; // ‘S’의 MSB를 ‘1’로 세트한다.  
}`

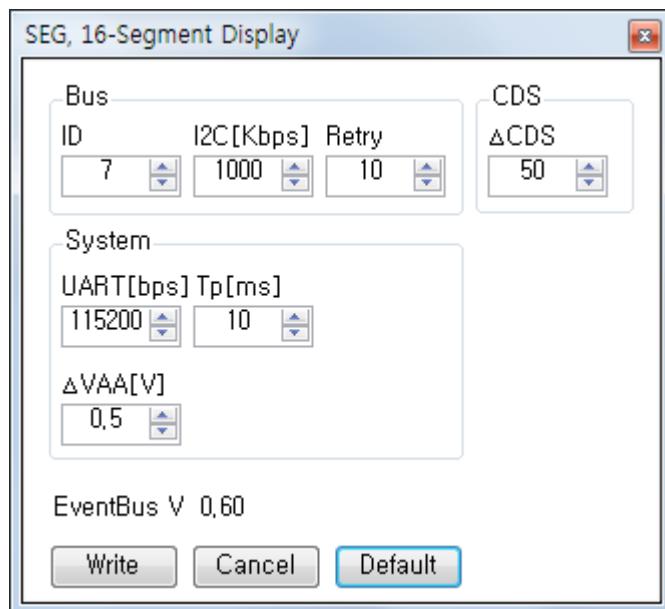
```

typedef struct {
    __BASE_PARAM;
    uint8      dCDS;
} SegParamEvent,SegParamChangedEvent,SegParamNotifiedEvent;

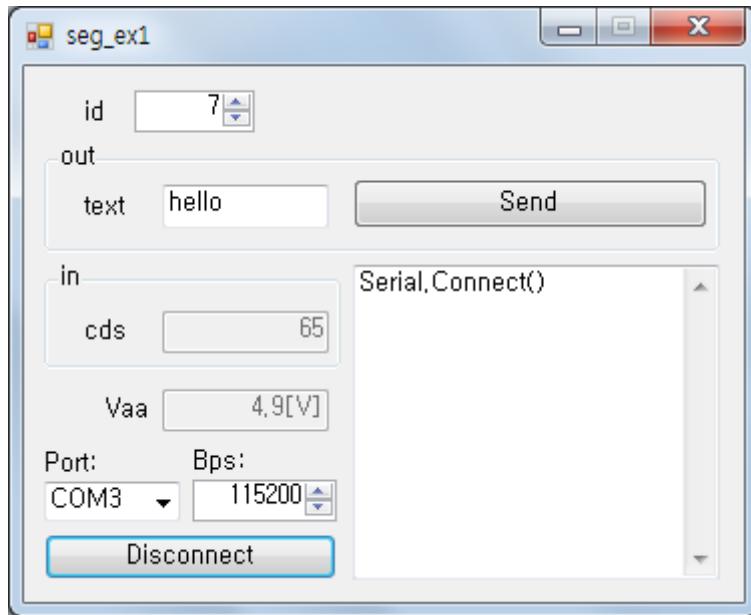
```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(seg\_drv.c)에 정의된 param 변수를 사용해서 param.dCDS 와 같이 읽습니다.

- \_\_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- dCDS  
CDS 로 검출한 조도값의 변동여부를 검사하는 기준값입니다.



ezconfig.exe 의 설정창입니다. SEG 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 SegParamEvent 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 ezconfig.exe 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.



eventbus\_dotnet\_0.60\example\seg\_ex1\bin\Debug\seg\_ex1.exe 예제프로그램을 실행하면 그림처럼 SEG를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps를 사용합니다. 세그먼트에 표시할 텍스트를 입력후 [Send]버튼을 클릭하면 표시문자를 변경할 수 있습니다. V0.50 이후부터는 Vaa 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(seg\_def.c 참고)

### 6.13. MSB, Multiport Serial Bridge



RS232 통신으로 외부장치 또는 센서와 데이터를 교환하는데 사용할 수 있는 통신모듈입니다. 예를 들면, RS232 를 지원하는 계측기에 연결하여 계측기를 제어하거나, RS232 를 사용하는 센서에 연결하여 이지컨트롤러(또는 PC)에서 센서의 데이터를 읽는 것과 같은 경우 유용하게 사용할 수 있습니다.

MSB 는 4 개의 RS232 포트를 제공합니다. 유의하실 점은 각 포트는 컴퓨터에서 가상 COM 포트 (COM5 처럼)로 잡히는 것이 아닙니다(포트가 많아지면 데이터를 놓치거나 불안정해 질 수 있고 프로그램도 복잡해 집니다). RS232 포트로 수신한 데이터는 컴퓨터(또는 다른 모듈)에 이벤트를 발생시키므로 이벤트핸들러를 작성해서 데이터를 수신하면 됩니다. 그러므로, 별도로 COM 포트를 설정하는 등의 작업이 필요하지 않습니다. 그리고, RS232 포트로 데이터를 송신하려면 단지 이벤트를 보내기만 하면 됩니다.(eventbus\_net\_0.60 의 msb\_ex1 프로젝트 참고)

커넥터 핀사양

| 핀번호 | 명칭  | 기능       | 기타         |
|-----|-----|----------|------------|
| 1   | TX0 | Transmit | RS232 포트 0 |
| 2   | RX0 | Receive  |            |
| 3   | GND | Ground   |            |
| 4   | TX1 | Transmit | RS232 포트 1 |
| 5   | RX1 | Receive  |            |
| 6   | GND | Ground   |            |
| 7   | TX2 | Transmit | RS232 포트 2 |
| 8   | RX2 | Receive  |            |
| 9   | GND | Ground   |            |
| 10  | TX3 | Transmit | RS232 포트 3 |
| 11  | RX3 | Receive  |            |
| 12  | GND | Ground   |            |

확장커넥터 핀사양

| 핀  | 핀 명칭                 | 핀 기능                                           |
|----|----------------------|------------------------------------------------|
| 1  | P2.0/SSEL1           | GPIO / SPI1 Slave Select                       |
| 2  | P2.1/SCK1            | GPIO / SPI1 Serial Clock                       |
| 3  | P2.2/MISO1           | GPIO / SPI1 Master Input Slave Output          |
| 4  | P2.3/MOSI1           | GPIO / SPI1 Master Output Slave Input          |
| 5  | P2.4                 | GPIO                                           |
| 6  | P2.5                 | GPIO                                           |
| 7  | P2.6                 | GPIO                                           |
| 8  | P2.7                 | GPIO                                           |
| 9  | P2.8                 | GPIO                                           |
| 10 | P2.9                 | GPIO                                           |
| 11 | P2.10                | GPIO                                           |
| 12 | P2.11                | GPIO                                           |
| 13 | P1.0/AD1/CT32B1_CAP0 | GPIO/ADC1 / Capture input 0 for 32-bit timer 1 |
| 14 | P1.1/AD2/CT32B1_MAT0 | GPIO/ADC2 / Match output 0 for 32-bit timer 1  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 15 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 16 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1          |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | 3.3V로직용 전원 출력                                            |
| 22 | GND                        |                                                          |
| 23 | VBB                        | 24V 전원 출력                                                |
| 24 | GND                        |                                                          |

```

typedef struct {
    MType      mtype;
    ID         id;
    MsbInEvent* in;
    MsbOutEvent* out;
} MSB;

```

MSB 모듈의 이미지는 위와 같은 MSB 구조체로 되어 있습니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Msb 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 입출력 이미지는 MsbInEvent, MsbOutEvent 구조체에 각각 저장되지만 (v0.60 부터 MsbInChangedEvent, MsbOutChangedEvent 가 간략히 MsbInEvent, MsbOutEvent로 명칭변경됨) MSB 모듈은 예외적으로 데이터의 송수신에 MsbInEvent, MsbOutEvent 를 사용하지 않으므로 아래처럼 기본이미지를 재정의해서 사용하고 있습니다.

```

typedef Event     MsbInEvent;
typedef Event     MsbOutEvent;

```

MSB 모듈은 입출력이 없으므로 기본이미지를 재정의하여 사용합니다. 실제 데이터의 송수신에는 사용자이벤트 MsbTransmitEvent, MsbReceivedEvent 를 사용합니다.

```

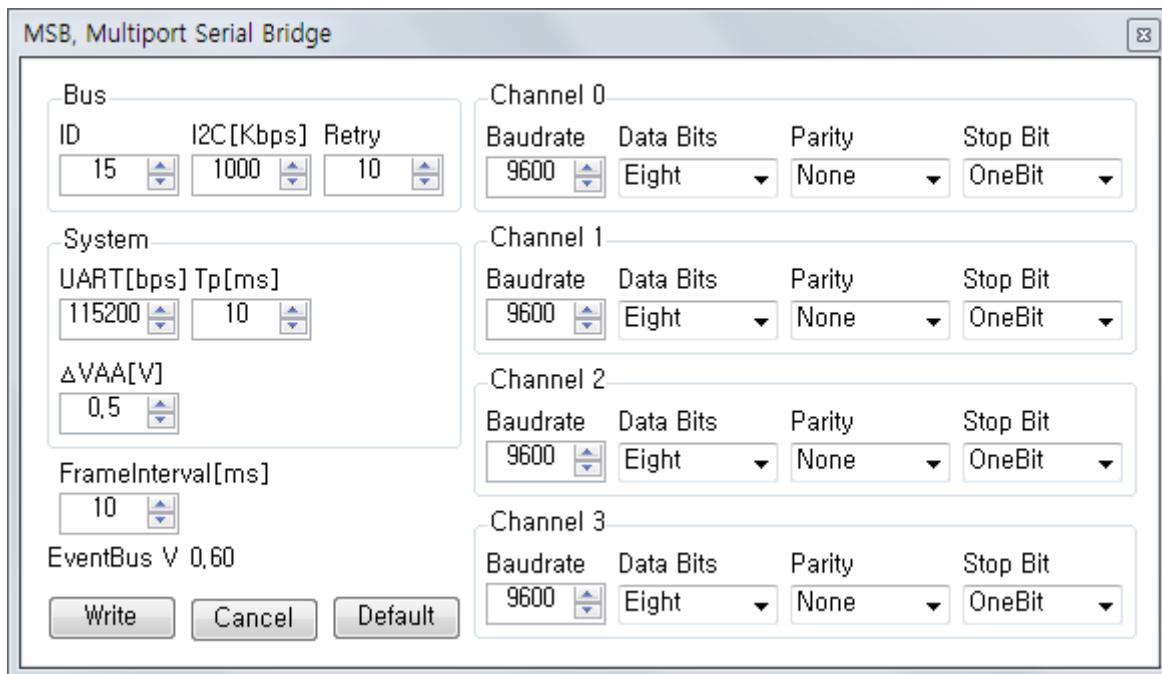
typedef struct {
    __BASE_PARAM;
    uint32 bps[4];
    uint8 dataBits[4];
    uint8 parity[4];
    uint8 stopBit[4];
    uint8 frameInterval; // frame-to-frame interval time to distinguish
} MsbParamEvent, MsbParamChangedEvent, MsbParamNotifiedEvent;

```

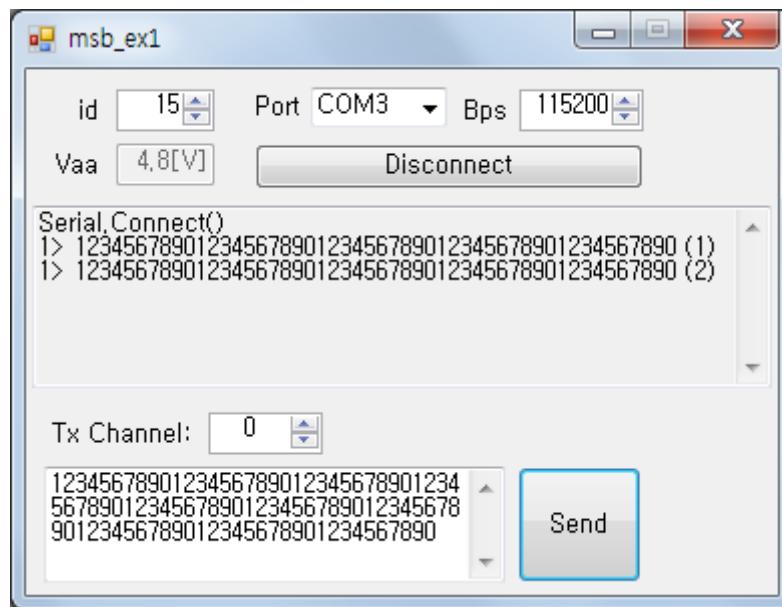
이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(msb\_drv.c)에 정의된 param 변수를 사용해서 param.bps[0]와 같이 읽습니다.

- \_\_BASE\_PARAM  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- bps  
RS232 통신속도, 최대 115,200bps.

- **dataBits**  
0(6 비트), 1(7 비트), 2(8 비트)
- **parity**  
0(None), 1(홀수 패리티), 2(짝수 패리티)
- **stopBit**  
0(1 비트), 1(2 비트)
- **frameInterval**  
RS232 채널에서 수신할 때 프레임과 다음 프레임을 구분하는데 사용되는 시간으로  
**frameInterval** 이 경과하면 프레임이 종료된 것으로 간주하고 **MsbReceivedEvent** 를  
발생시킨다.



`ezconfig.exe` 의 설정창입니다. MSB 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 `MsbParamEvent` 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 `ezconfig.exe` 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.



그림은 eventbus\_net에서 제공되는 기본 테스트용 프로그램입니다. Connect 버튼을 클릭하면 UUM(또는 ColinkEx)를 통해 PC와 연결됩니다. 아래의 “Tx Channel”에서 송신할 포트를 0으로 선택하였으며, 통신선 배선은 채널 0과 1을 루프백으로 연결하였습니다.(TX0-RX1, RX0-TX1). 아래 텍스트박스에 송신데이터 100글자를 입력한 후 Send 버튼으로 전송하면 포트 1에서 수신한 데이터가 표시됩니다. “1>”은 수신포트이며 뒤이어 수신한 데이터가 표시됩니다. 끝의 “(1), (2)”는 수신한 데이터가 50바이트를 넘을 경우 분할하여 전송하는데 수신받은 패킷에 번호를 붙인 것입니다. 즉, 100글자를 50글자씩 나누어서 2번 전송하고 있습니다.

MSB 모듈은 송수신에 입출력이미지를 사용하지 않습니다. 즉, 다른 모듈처럼 MsbIn, MsbOut 이벤트를 사용하지 않고 사용자정의 이벤트(User Defined Event)를 사용합니다. 태그명은 MsbTransmit(0x43), MsbReceived(0x44)이고, 이벤트는 eventbus.h에 아래와 같이 선언되어 있습니다.

```
#define MSB_PACKET_SIZE 50

typedef struct {
    EventHead e;
    uint8      chan;
    uint8      len;
    uint8      packet[MSB_PACKET_SIZE];
} MsbTransmitEvent, MsbReceivedEvent;
```

위의 이벤트를 사용해서 데이터를 송신하는 예제가 `application\exam\msb_loopback.c`에 있습니다. 펌웨어에서 데이터를 전송하려면 다음과 같이 이벤트를 발생하면 됩니다.

```
UserEvent(mte, MsbTransmit); // 이벤트 생성  
mte->chan = 0; // 송신할 포트  
mte->len = 16; // 데이터 길이  
memcpy(mte->packet, "abcdefghijklmnop", 16); // 송신데이터 저장  
SendEvent(mte); // 이벤트 전송
```

아래 코드는 MSB 가 시리얼포트에서 데이터를 수신할 때 발생하는 `MsbReceived` 이벤트를 처리하는 이벤트핸들러입니다. 수신 데이터(`pe->packet`)를 컴퓨터의 하이퍼터미널로 전송하고 있습니다.

```
IF (this, MsbReceived) {      // 패킷수신 이벤트핸들러
    int i;

    printf("COM%d (%d): ", pe->chan, pe->len); // 하이퍼터미널에 포트, 길이 출력
    for (i = 0; i < pe->len; ++i) {                // 하이퍼터미널에 데이터 출력
        printc(pe->packet[i]);
    }
}
```

물론, `MsbReceived` 이벤트는 PC 에게도 전송되므로 아래처럼 적절한 이벤트핸들러를 등록한 후 수신데이터를 처리하도록 프로그램을 작성하면 됩니다.

```
private void Form1_Load(object sender, EventArgs e)
{
    // 이벤트핸들러 등록
    msb.Packet.OnReceived += new MsbReceivedEventHandler(msb_OnPacketReceived);
}
```

## 6.14. MOC, Motion Control Module



모션제어(위치제어)용 펄스출력 모듈입니다. 최대 펄스출력 100KPPS 의 고속동작과 차동출력으로 모터드라이버에 직접연결하여 사용할 수 있으며, 정방향과 역방향에 대한 리미트 스위치(센서) 접점입력 포트가 있어서 범위밖으로 이탈하지 못하게 회전을 제한하며, 고속동작시 빠른 원점찾기를 위하여 NNP(원점부근위치) 입력 기능을 가지고 있습니다(옵션). 자유자재로 조그할 수 있도록 조그방향 선택스위치와 조그속도선택 스위치가 있어 2 단으로 조그속도를 조절할 수 있으며, 가변저항이나 포텐시미터 또는 전압입력에 의한 무단 속도조절기능이 있습니다. 또한 사다리꼴 가감속 프로파일을 내장하여 부드러운 가감속 구동이 가능하며, FREE, STOP, RUN, MOVE 동작을 지원하므로 간단한 명령으로 자유로운 회전을 구현할 수 있습니다. 갑작스런 속도가변에도 부드럽게 가감속할 수 있는 스무드 자동변속기능이 있습니다.

IO 커넥터 핀사양

| 핀  | 명 칭  | 기 능                    | 기 타                                                            |
|----|------|------------------------|----------------------------------------------------------------|
| 1  | DIR+ | 회전방향                   | CW 일 때 전압 : DIR+ > DIR- (약 4V 차동출력)                            |
| 2  | DIR- |                        | CCW 일 때 전압 : DIR+ < DIR-                                       |
| 3  | PLS+ | 펄스출력                   | HIGH 일 때 전압 : DIR+ > DIR- (약 4V 차동출력)                          |
| 4  | PLS- |                        | LOW 일 때 전압 : DIR+ < DIR-                                       |
| 5  | POS  | CW 방향 리미트 입력           | POS~GND 사이에 스위치 또는 NPN 타입의 센서사용                                |
| 6  | NEG  | CCW 방향 리미트 입력          | NEG~GND 사이에 스위치 또는 NPN 타입의 센서사용                                |
| 7  | NNP  | 사용자입력용.<br>(옵션) NNP 기능 | NNP~GND 사이에 스위치 또는 NPN 타입의 센서사용<br>NNP(Near Negative Position) |
| 8  | JCW  | CW 조그스위치 입력            | 3-Position Toggle 스위치 (CW, OFF, CCW)                           |
| 9  | JCCW | CCW 조그스위치 입력           | JCW/JCCW~GND 사이에 스위치 연결                                        |
| 10 | JREF | 볼륨용 기준전압 출력            |                                                                |
| 11 | JSPD | 조그 속도 입력               |                                                                |
| 12 | GND  | 신호 그라운드                |                                                                |



확장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5 | P2.4       | GPIO                                  |
| 6 | P2.5       | GPIO                                  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.1/AD2/CT32B1_MAT0       | GPIO/ADC2 / Match output 0 for 32-bit timer 1            |
| 14 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 15 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1          |
| 16 | VAA                        | VAA(5VDC) 버스전원                                           |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로직용 전원 출력                                     |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC) 전원 출력                                         |
| 24 | GND                        |                                                          |

```
typedef struct {
    MType          mtype;
    ID             id;
    MocInEvent*   in;
    MocOutEvent*  out;
} MOC;
```

MOC 모듈의 이미지입니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Moc 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력이미지는 MocInEvent 구조체에, 출력이미지는 MocOutEvent 구조체에 저장됩니다. (V0.60 부터 MocInChangedEvent, MocOutChangedEvent 가 간략히 MocInEvent, MocOutEvent 로 명칭변경됨)

```
typedef struct {
    EventHead  e;
    union {
        struct {
            ELogic pos      :1;    // positive 스위치, LSB
            ELogic neg      :1;    // negative 스위치
            ELogic nnp      :1;    // near negative position 스위치(옵션)
            ELogic jcw      :1;    // jog cw 스위치, LSB
            ELogic jccw     :1;    // jog ccw 스위치
            ELogic jspd     :1;    // jog speed 스위치
            Bool   busy     :1;    // busy flag
        };
    };
};
```

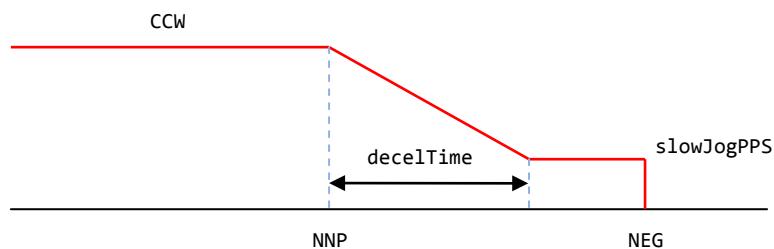
```

    uint16 status;
};

uint16 jvol;           // 조그 속도 볼륨의 A/D 변환값 (0~255)
int32 loc;             // NEG센서로부터 이동한 펄스수(절대위치)
} MocInEvent;

```

`pos`, `neg` 는 소스스타입 접점으로 GND 와의 사이에 스위치를 설치합니다. 이 입력은 모터가 CW, CCW 방향으로 회전하여 이동범위를 벗어나는 것을 방지해주는 모터 안전동작범위 제한용입니다.



`NNP` 는 `NEG` 위치(보통 CCW 방향의 끝위치) 근처에 설치하여 급속이동을 구현하기 위하여 사용됩니다. 즉, 처음에는 `NEG` 의 위치를 알지 못하므로 빠른 속도로 CCW 로 회전하면 자칫 `NEG` 센서를 지나칠 수 있으므로 원점(NEG)을 찾기 위하여 느린 속도로 CCW 로 회전할 필요가 있습니다. 그래서 고속으로 CCW 방향으로 회전하는 중에 `NNP` 를 만나면 감속을 시작하고 `NEG` 에서 정지하는 방법을 사용하면 고속으로 원점으로 이동할 수 있습니다. (출고시 기본설정은 `NNP` 기능이 꺼져있으며, 사용자 입력용으로 사용할 수 있습니다. `NNP` 기능을 켜기 위해서는 `moc_def.c` 에서 `#define USE_NNP` 의 주석을 해제하고 빌드후 모듈에 펌웨어를 다시 다운로드 해야 합니다, ColinkEx 필요)

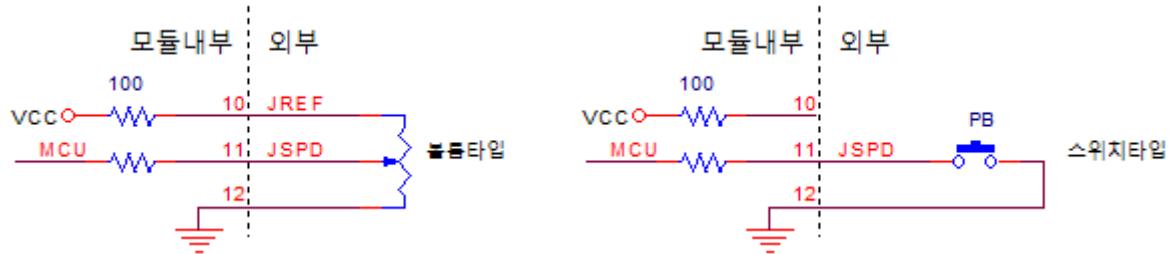
`jcw`, `jccw` 는 조그의 회전방향을 결정하기 위하여 사용됩니다. `jcw` 에 연결된 스위치는 정방향을, `jccw`에 연결된 스위치는 역방향을 선택합니다.



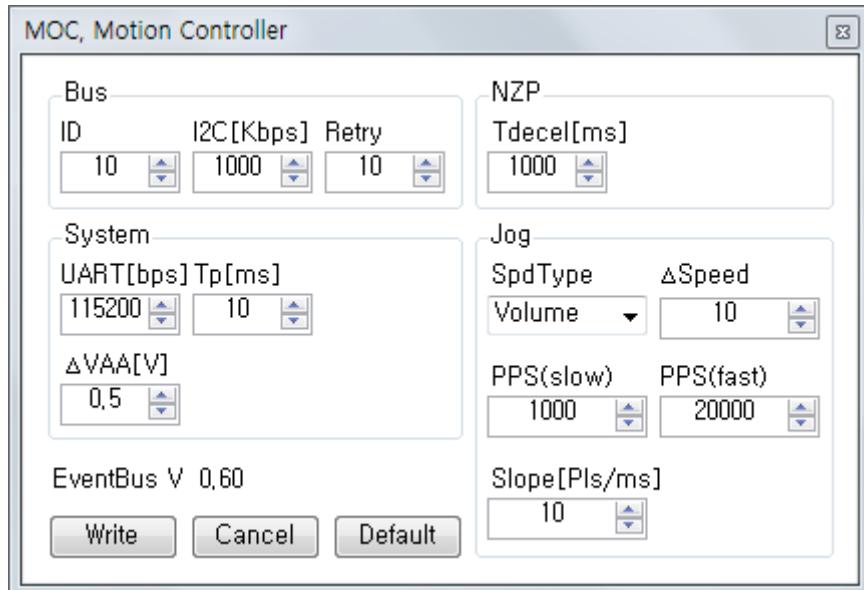
`POS(5), NEG(6), NNP(7), JCW(8), JCCW(9)`의 내부회로는 위와 같으므로, 외부에 스위치를 바로 연결할 수 있습니다. 내부회로가 DIO, DIM 과 다르므로 주의해 주십시오.



리미스위치를 사용하면 스위치의 ON-OFF 사이에 히스테리시스가 있어 오차가 발생할 수 있으므로, 비접촉식 센서를 사용하는 것이 좋으며, 이 경우 그림처럼 NPN 출력이 적절합니다.



MOC 는 스위치 방식과 볼륨방식(약 1K~10K 정도가 적당)의 조그를 모두 지원합니다. 왼쪽 그림은 볼륨타입으로 무단변속하는 경우이고, 오른쪽은 스위치타입으로 일정속도로 온오프할 때 사용됩니다.



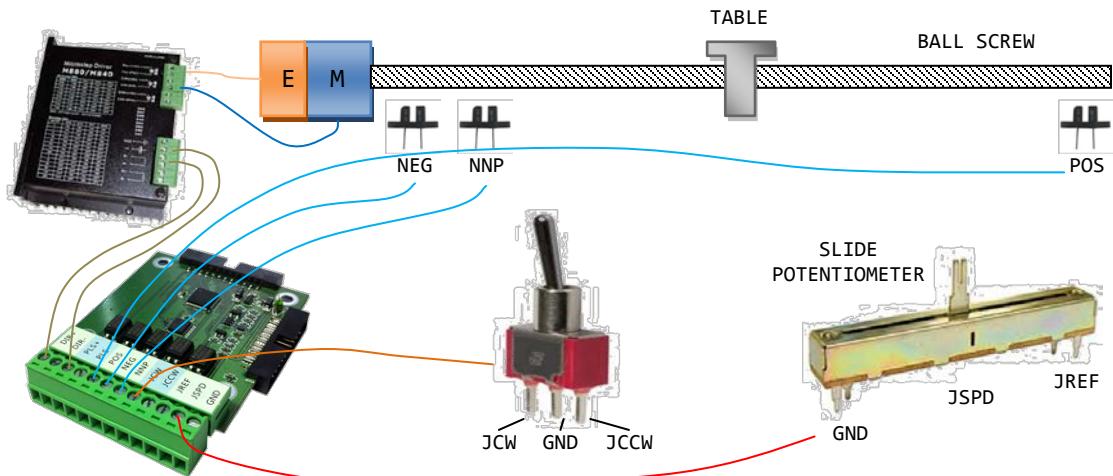
볼륨타입 또는 전압입력으로 조그를 사용하는 경우, 그림처럼 ezconfig.exe에서 SpdType 을 Volume 으로 선택하면 됩니다. 볼륨의 전압을 A/D 변환한 값(0~255)이 그림의  $\Delta$  Speed 보다 크게 변하면, 그 값에 따라 PPS(slow)~PPS(fast) 사이의 적절한 속도로 펄스가 출력됩니다.

스위치타입의 경우 2 단으로 속도를 변경할 수 있으며, 스위치가 오프이면 PPS(slow), 온이면 PPS(fast)의 속도로 회전합니다.

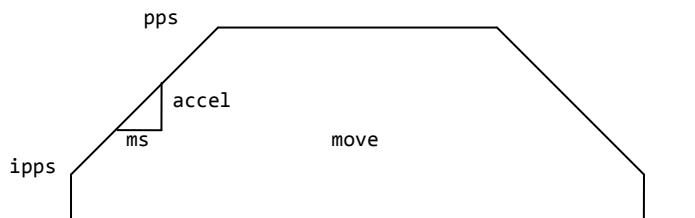
| JCW(8) | JCCW(9) | JSPD(11) | 조그속도(PPS)      |
|--------|---------|----------|----------------|
| OFF    | OFF     | 상관없이     | OFF            |
| ON     | OFF     | OFF      | CW, PPS(slow)  |
| ON     | ON      | ON       | CW, PPS(fast)  |
| OFF    | ON      | OFF      | CCW, PPS(slow) |
| OFF    | ON      | ON       | CCW, PPS(fast) |

busy 는 현재 펄스가 출력중 인지를 알려줍니다.

`loc` 는 절대위치를 알려줍니다. 처음 전원투입후 음의 값(원점을 초기화 하지 못했음을 알리는)을 가지고 있으며, NEG 스위치가 한번 ON 되면 `loc` 는 0 이 됩니다(즉, 영점이 잡힙니다). 이후 CW 펄스는 이값을 증가시키고, CCW 펄스는 이값을 감소시킵니다. 주의할 점은 `loc` 값이 실시간으로 변경되지는 않으므로 이 값으로 위치제어를 하는 것은 적절치 않습니다.



(서보모터 드라이버와 연결하여 1 축 위치제어를 구현한 예)



(MOC 의 가감속 속도프로파일)

```

typedef enum {FREE, HOLD, STOP, RUN, MOVE} SMD_CMD, MOC_CMD, USD_CMD;
typedef enum {CW, CCW} SMD_DIR, MOC_DIR, USD_DIR;

typedef struct {
    EventHead e;
    union {
        struct {
            MOC_CMD cmd      :4;
            MOC_DIR dir      :1;
        };
        uint16 config;
    };
    uint16 ipps;           // initial PPS
    uint16 accel;          // accel,decel speed. [pulse/ms]
    uint32 pps;            // steady state speed [pulse/second]
    uint32 move;           // total pulse to move.
} MocOutEvent;

```

MOC 모듈은 사다리꼴 가감속 프로파일을 지원합니다. `ipps` 는 초기속도, `accel` 은 밀리초(`ms`)당 가감속할 펄스개수이며, `pps` 는 정속구간의 회전속도를 나타내며, `move` 는 MOVE 명령에서의 이동(회전)할 펄스의 총개수입니다.

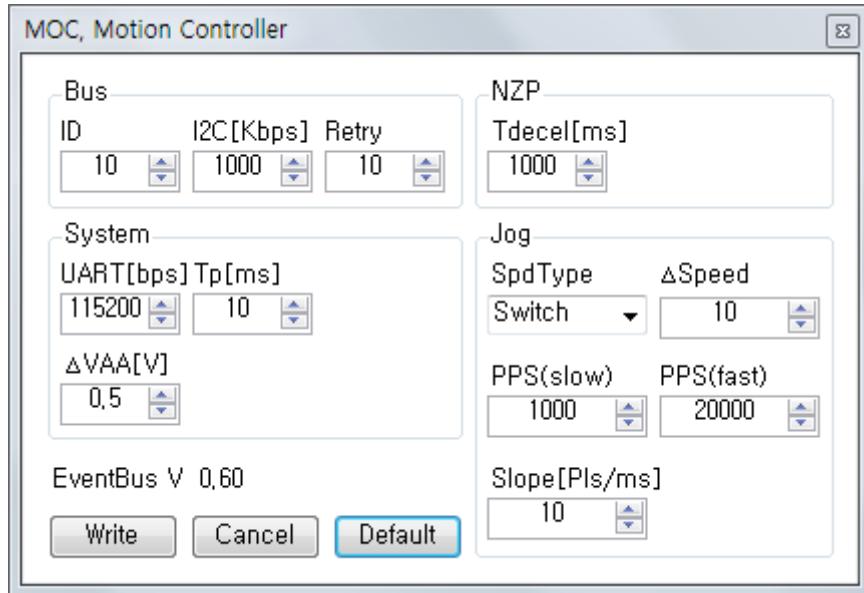
`cmd` 는 펠스 출력 방식을 제어하는 명령입니다. `FREE` 와 `HOLD` 는 펠스 출력을 정지시키고, `STOP` 은 감속정지하며, `RUN` 은 가속후 연속회전 하도록 연속펄스(pps)를 출력하는 명령이며, `MOVE` 는 설정된 펠스개수(`move`)만큼 가감속프로파일을 따라 이동(회전)하도록 펠스를 출력합니다. `dir` 은 회전방향을 지시하는 것으로 `CW` 는 시계방향으로, `CCW` 는 반시계방향으로 회전합니다.

```
typedef struct {
    __BASE_PARAM;

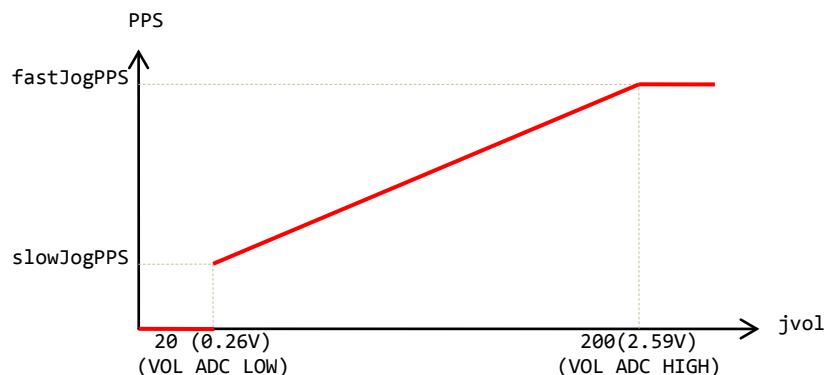
    uint32 slowJogPPS;           // jogSpdType=Volumn(2) 일 경우 최저 회전속도
    uint32 fastJogPPS;          // jogSpdType=Volumn(2) 일 경우 최고 회전속도
    uint16 jogAccel;            // 조그동작에 적용할 가감속
    uint8 dJVol;                // 볼륨인 경우 변동검출 기준치
    uint8 jogType;              // NoConnect(0), Switch(1), Volume(2)
    uint16 decelTime;           // NNP 검출시 감속시간[밀리초]
} MocParamEvent,MocParamChangedEvent,MocParamNotifiedEvent;
```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. `ezconfig.exe` 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(`moc_drv.c`)에 정의된 `param` 변수를 사용해서 `param.slowJogPPS` 와 같이 읽습니다.

- `__BASE_PARAM`  
“10.22 XxxParamChangedEventArgs : EEPROM 설정 데이터” 참고
- `slowJogPPS, fastJogPPS`  
조그의 최저 PPS 와 최대 PPS 를 설정합니다. JSPD 단자(11 번핀)에 볼륨타입 포텐쇼미터를 연결하면 포텐쇼미터의 위치에 비례하여 최저 PPS 에서 최대 PPS 까지 속도가 가변됩니다. 만일 JSPD 에 스위치를 연결하면 OFF 일때 최저 PPS, ON 일때 최대 PPS 로 회전하는 2 단속도의 조그가 가능합니다.
- `jogAccel`  
조그 속도의 변경시(조그스위치 또는 볼륨의 조작으로) 사용할 가감속 기울기.  
[펄스/ms]
- `dJVol`  
볼륨타입 조그 사용시 A/D 변환결과는 8 비트(0~255)를 사용합니다. 볼륨값에 변동이 있는지 판단하는 기준이 되는 값입니다. `dJVol` 보다 큰 볼륨값의 변동은 조그속도를 변경시킵니다.
- `jogType`  
조그를 사용하지 않으면 0, 스위치타입 조그는 1, 볼륨타입 조그는 2 입니다.
- `decelTime`  
CCW 회전중 NNP 를 만나면 안전을 위하여 자동감속을 시작하는데, 최저속도인 `slowJogPPS` 까지 감속하는데 걸리는 시간을 설정합니다.

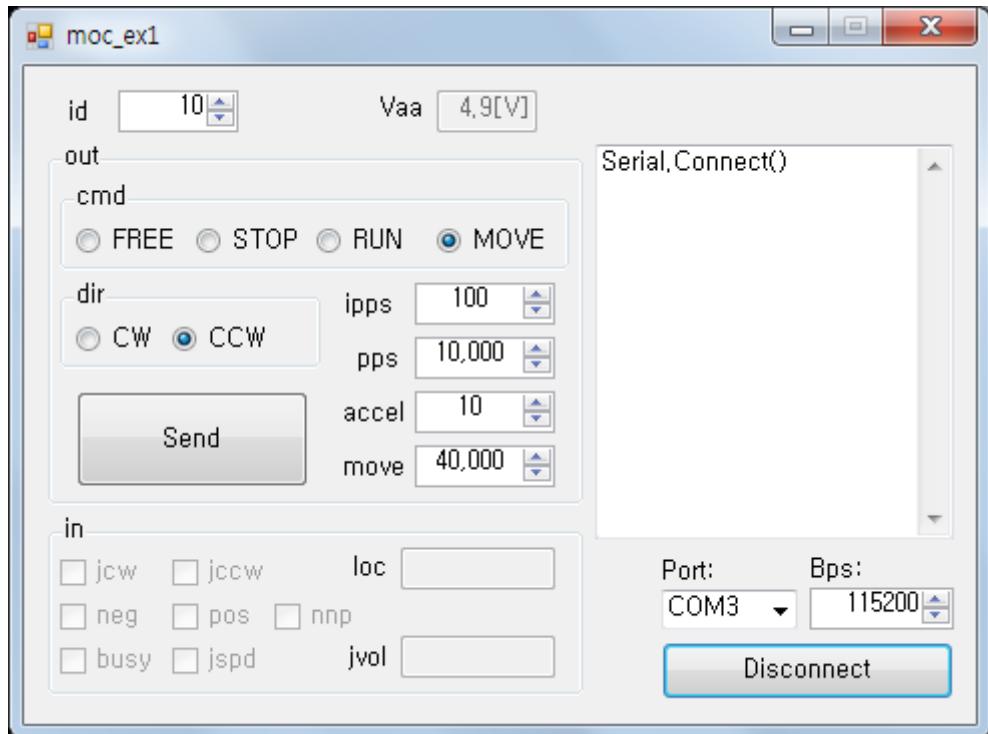


`ezconfig.exe` 의 설정창입니다. MOC 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 `MocParamEvent` 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 `ezconfig.exe` 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.



그림은 볼륨타입 조그를 사용할 때 볼륨의 A/D 변환값(0~255)과 조그 PPS 와의 관계를 나타냅니다. 변환값(jvol)이 20 보다 작으면 정지합니다 (`moc_def.c` 에서 변경가능). 그러므로, JCW/JCCW 에 연결된 스위치로 방향을 설정 한 후 볼륨만으로 정지부터 회전속도 조정까지 모든 작업을 볼륨만으로 할 수 있습니다.

\***용용\*** 푸시버튼을 한번 누를때마다 일정거리(펄스) 이동하는 조그동작을 구현하고 싶다면, NNP 에 푸시버튼을 연결하고(NNP 의 디폴트는 사용자 입력용이므로), MocIn 이벤트핸들러에서 MOVE 명령으로 원하는 펄스수(move)만큼 이동하도록 프로그램 하면 됩니다. NNP 기능을 활성화시키기 위하여는 `moc_def.c` 에서 매크로 설정을 활성화시킨 후 프로그램을 마이컴에 다시 다운로드해야 합니다.



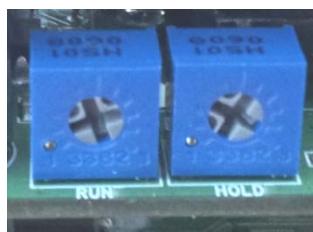
eventbus\_net\_0.60\example\moc\_ex1\bin\Debug\moc\_ex1.exe 예제프로그램을 실행하면 그림처럼 MOC 를 테스트할 수 있습니다. 연결된 시리얼포트를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. 위 그림처럼 설정 후 [Send] 버튼을 클릭하면 모터를 구동하기 위한 필스가 출력됩니다. v0.50 이후부터는 Vaa 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(moc\_def.c 참고)

## 6.15. USD, Unipolar Step-motor Driver



모터 제어용으로는 SMD, MOC, USD 의 3 가지 제품이 있습니다. 각 제품의 특징을 구분하여 사용하시기 바랍니다. 모터를 구동하기 위하여 구동펄스를 생성하는 컨트롤러와 모터에 전류를 흘리는 드라이버가 필요합니다. SMD 는 2 개의 유니폴러 스텝모터를 구동할 수 있는 컨트롤러와 드라이버가 모두 내장된 제품입니다. 반면, 조그기능이나 스위치 입력기능이 없어서 개루프 회전을 제어하기에는 적합하지만, 본격적인 위치제어에는 부족한 점이 있습니다. 그리고, MOC 는 모터제어용 펄스를 생성하는 컨트롤러입니다. 조그기능과 스위치 입력기능이 있어서 위치제어에 편리하게 사용할 수 있습니다.

그러나 외부에 모터드라이버를 별도로 연결하여야 합니다. 예를 들어, 서보모터를 구동하려면 MOC 에서 출력되는 펄스신호를 구입한 서보모터 드라이버에 연결하여야 합니다. 반면, USD 는 MOC 와 유니폴러 스텝모터용 드라이버를 결합한 제품입니다. 1 축 유니폴러 스텝모터의 위치제어를 위한 모든 기능이 내장되어 있습니다. 사다리꼴 가감속 펄스생성, 스위치 및 볼륨형 조그기능, 회전범위제한을 위한 리미트 센서입력, 사용자용 스위치 입력(스타트/스톱 스위치등을 연결하여 프로그램 할 수 있음)의 기능을 갖추고 있어서 위치제어에 유용하게 사용할 수 있습니다.



모터전류는 우측에 있는 2 개의 가변저항으로 RUN, HOLD 전류를 설정할 수 있습니다. 최대전류는 약 2.8A이며 가운데 눈금에 맞추면 약 1.4A 의 전류가 흐릅니다(대략 눈금당 0.28A). HOLD 전류는 모터 정지시 부하의 무게 또는 관성에 의한 자유회전을 방지하기 위한 용도이며 기구물의 구조상 자유회전하지 않는 경우는 필요치 않으므로 완전히 왼쪽으로 돌려서 훌딩전류가 흐르지 않도록 하는 것이 발열이 줄어서 좋습니다.

그림에서 보듯이 USD 는 2 장의 PCB 로 구성되어 있습니다. 아래의 PCB 는 컨트롤러 역할을 하는 MOC 모듈인데, 하드웨어는 MOC 와 완전히 동일하고 내장된 펌웨어 프로그램이 다릅니다(eventbus\_fw 참고). 자세한 동작은 MOC 의 사용법을 참고해 주십시오. 위에는 유니폴러 스텝모터를 구동하는 드라이버용 PCB 가 있습니다. 방열판과 온도에 따라 자동으로 속도제어가 되는 쿨링팬이 장착되어 있습니다.

하단 PCB(MOC) 커넥터 핀사양

| 핀  | 명칭   | 기능                     | 기타                                                              |
|----|------|------------------------|-----------------------------------------------------------------|
| 1  | DIR+ | 회전방향                   | CW 일 때 전압 : DIR+ > DIR- (전압차 약 4V)                              |
| 2  | DIR- |                        | CCW 일 때 전압 : DIR+ < DIR-                                        |
| 3  | PLS+ | 펄스출력                   | HIGH 일 때 전압 : DIR+ > DIR- (전압차 약 4V)                            |
| 4  | PLS- |                        | LOW 일 때 전압 : DIR+ < DIR-                                        |
| 5  | POS  | CW 방향 리미트 입력           | POS~GND 사이에 스위치 또는 NPN 출력의 센서사용                                 |
| 6  | NEG  | CCW 방향 리미트 입력          | NEG~GND 사이에 스위치 또는 NPN 출력의 센서사용                                 |
| 7  | NNP  | 사용자입력용.<br>(옵션) NNP 기능 | NNP~GND 사이에 스위치 또는 NPN 출력의 센서사용<br>NNP(Near Negative Position)  |
| 8  | JCW  | CW 조그스위치 입력            | 3-Position Toggle 스위치 (CW, OFF, CCW)<br>JCW/JCCW~GND 사이에 스위치 연결 |
| 9  | JCCW | CCW 조그스위치 입력           |                                                                 |
| 10 | JREF | 볼륨용 기준전압 출력            |                                                                 |
| 11 | JSPD | 조그 속도 입력               |                                                                 |
| 12 | GND  | 신호 그라운드                |                                                                 |

상단 PCB 커넥터 핀사양

| 핀  | 명칭  | 기능             | 기타                 |
|----|-----|----------------|--------------------|
| 1  | DIR | 회전방향           |                    |
| 2  | PLS | 회전펄스           |                    |
| 3  | SW0 | 사용자 스위치 입력     | 연결 및 사용법은 DIE 참고   |
| 4  | SW1 | 사용자 스위치 입력     |                    |
| 5  | C01 | 사용자 스위치 공통     |                    |
| 6  | PA+ | A 상            |                    |
| 7  | PA- | A/ 상           |                    |
| 8  | COM | COMMON         |                    |
| 9  | PB+ | B 상            |                    |
| 10 | PB- | B/ 상           |                    |
| 11 | VM  | 모터전원 (외부에서 공급) | 24V (최대 46VDC, 3A) |
| 12 | GND | 모터전원 그라운드      |                    |

```

typedef struct {
    MType      mtype;
    ID         id;
    UsdInEvent* in;
    UsdOutEvent* out;
} USD;

```

USD 모듈의 이미지입니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Usd 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력이미지는 UsdInEvent 구조체에,

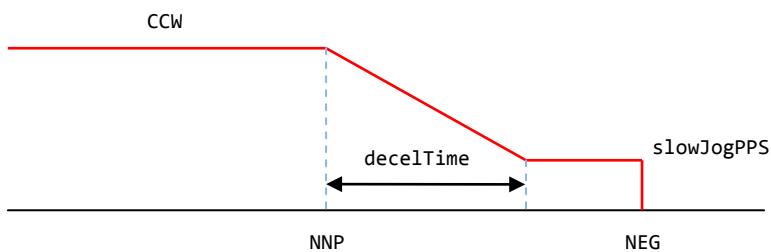
출력이미지는 `UsdOutEvent` 구조체에 저장됩니다. (V0.60 부터 `UsdInChangedEvent`, `UsdOutChangedEvent` 가 간략히 `UsdInEvent`, `UsdOutEvent`로 명칭변경됨)

```

typedef struct {
    EventHead e;
    union {
        struct {
            ELogic pos      :1;      // positive 스위치, LSB
            ELogic neg      :1;      // negative 스위치
            ELogic nnp      :1;      // near negative position 스위치
            ELogic jcw      :1;      // jog cw 스위치, LSB
            ELogic jccw     :1;      // jog ccw 스위치
            ELogic jspd     :1;      // jog speed 스위치
            Bool busy       :1;      // busy flag
            Bool fan        :1;      // cooling fan fault
            ELogic sw0       :1;      // switch for user
            ELogic sw1       :1;      // switch for user
        };
        uint16 status;
    };
    uint16 jvol;           // 조그 속도 볼륨의 A/D 변환값 (0~255)
    int32 loc;             // 이동한 펄스수(절대위치)
    float vm;              // Step Motor Driving voltage
    int16 heat;            // Thermistor
} UsdInEvent;

```

입력이미지 `UsdInEvent` 는 `fan`, `sw0`, `vm`, `heat` 를 제외하면 `MocInEvent` 와 동일합니다. `pos`, `neg` 는 소스타입 접점으로 GND 와의 사이에 스위치를 설치합니다. 이 입력은 모터가 CW, CCW 방향으로 회전하여 이동범위를 벗어나는 것을 막아주는 모터 안전동작범위 제한용입니다.



`NNP` 는 `NEG` 위치(보통 CCW 방향의 끝위치) 근처에 설치하여 급속이동을 구현하기 위하여 사용됩니다. 즉, 처음에는 `NEG` 의 위치를 알지 못하므로 빠른 속도로 CCW 로 회전하면 자칫 `NEG` 센서를 지나칠 수 있으므로 원점(`NEG`)을 찾기 위하여 느린 속도로 CCW 로 회전할 필요가 있습니다. 그래서 고속으로 CCW 방향으로 회전하는 중에 `NNP` 를 만나면 감속을 시작하고 `NEG` 에서 정지하는 방법을 사용하면 고속으로 원점으로 이동할 수 있습니다. (출고시 기본설정은 `NNP` 기능이 꺼져있으며, 사용자 입력용으로 사용할 수 있습니다. `NNP` 기능을 켜기 위해서는 `usd_def.c` 에서 `#define USE_NNP` 의 주석을 해제하고 빌드후 모듈에 펌웨어를 다시 다운로드 해야 합니다)

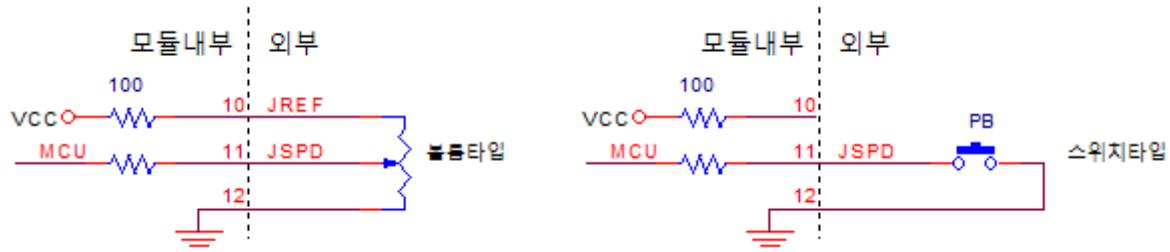
jcw, jccw 는 조그의 회전방향을 결정하기 위하여 사용됩니다. jcw 에 연결된 스위치는 정방향을, jccw에 연결된 스위치는 역방향을 선택합니다.



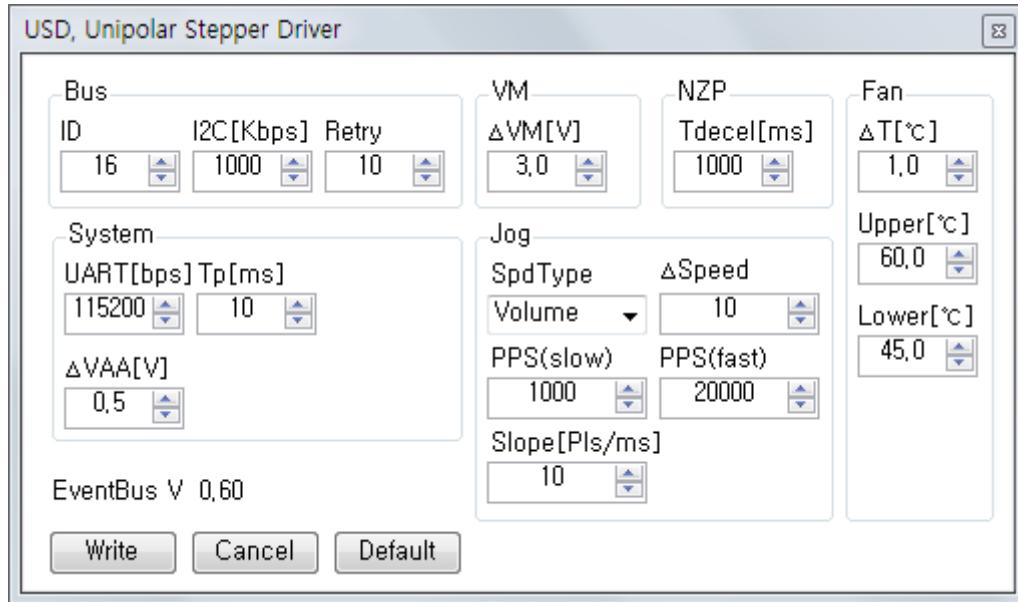
POS(5), NEG(6), NNP(7), JCW(8), JCCW(9)의 내부회로는 위와 같으므로, 외부에 스위치를 바로 연결할 수 있습니다. 내부회로가 DIO, DIM과 반대이므로 주의해 주십시오.



리미트스위치를 사용하면 스위치의 ON-OFF 사이에 히스테리시스가 있어 오차가 발생할 수 있으므로, 비접촉식 센서를 사용하는 것이 좋으며, 이 경우 그림처럼 NPN 타입이 적절합니다.



USD 는 스위치 방식과 볼륨방식(약 1K~10K 정도가 적당)의 조그를 모두 지원합니다. 왼쪽 그림은 볼륨타입으로 무단변속하는 경우이고, 오른쪽은 스위치타입으로 일정속도로 온오프할 때 사용됩니다.

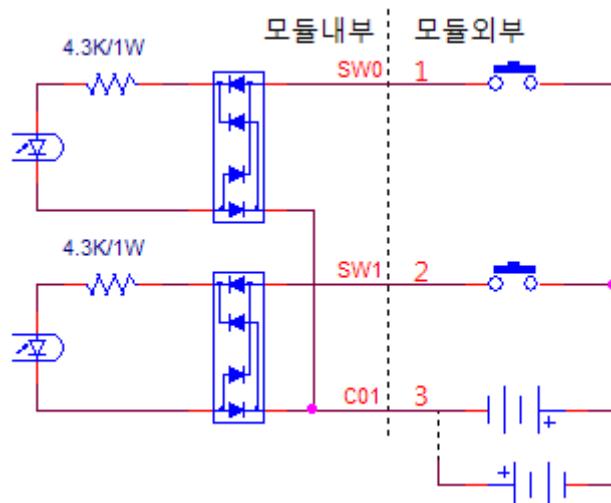


볼륨타입 또는 전압입력으로 조그를 사용하는 경우, 그림처럼 ezconfig.exe에서 SpdType 을 Volume 으로 선택하면 됩니다. 볼륨의 전압을 A/D 변환한 값(0~255)이 그림의  $\Delta$  Speed 보다 크게 변하면, 그 값에 따라 PPS(slow)~PPS(fast) 사이의 적절한 속도로 펄스가 출력됩니다.

스위치타입의 경우 2 단으로 속도를 변경할 수 있으며, 스위치가 오프이면 PPS(slow), 온이면 PPS(fast)의 속도로 회전합니다.

| JCW(8) | JCCW(9) | JSPD(11) | 조그속도(PPS)     |
|--------|---------|----------|---------------|
| OFF    | OFF     | 상관없이     | OFF           |
| ON     | OFF     | OFF      | CW-PPS(slow)  |
|        |         | ON       | CW-PPS(fast)  |
| OFF    | ON      | OFF      | CCW-PPS(slow) |
|        |         | ON       | CCW-PPS(fast) |

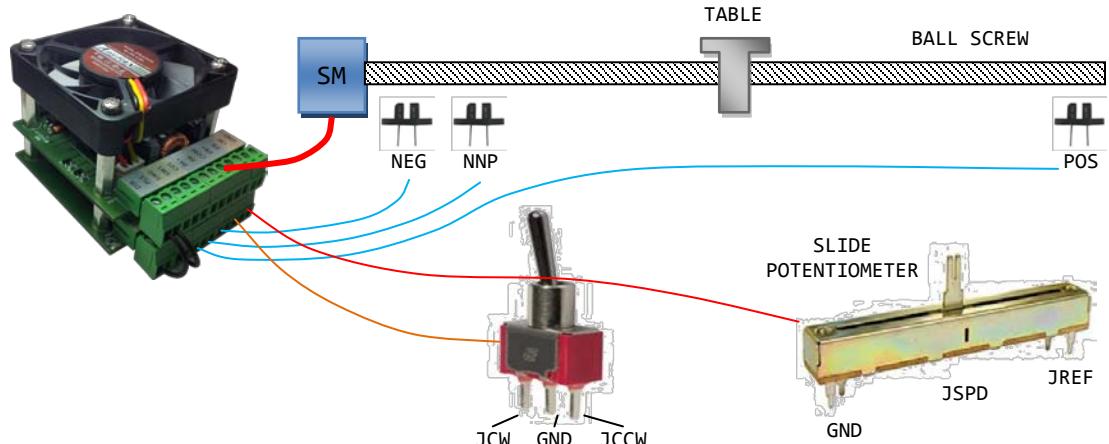
busy 는 현재 모터가 회전중인지를 알려주고, fan 은 모터드라이버 칩(SLA7078M)의 과열여부를 알려주며, 과열시 자동으로 모터전류는 차단되어 파손을 보호합니다.



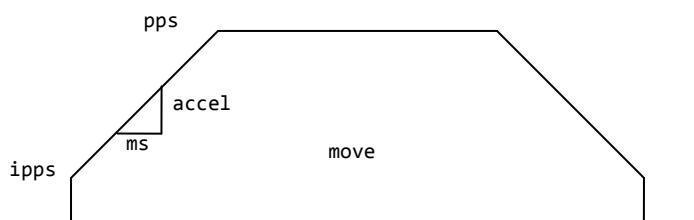
`sw0`, `sw1` 은 사용자용으로 제공되는 스위치의 상태값이 저장됩니다. 그림처럼 입력회로는 D1E 와 동일하므로 사용법은 D1E 를 참고해 주십시오.

`loc` 는 절대위치를 알려줍니다. 처음 전원투입후 음의 값(원점을 초기화 하지 못했음을 알리는)을 가지고 있으며, NEG 스위치가 한번 ON 되면 `loc` 는 0 이 됩니다(즉, 영점이 잡힙니다). 이후 CW 펄스는 이값을 증가시키고, CCW 펄스는 이값을 감소시킵니다. 주의할 점은 `loc` 값이 실시간으로 변경되지는 않으므로 이 값으로 위치제어를 하는 것은 적절치 않습니다.

`vm`, `heat` 는 각각 `VM` 으로 공급되는 모터구동용 입력전원의 전압과 구동칩(SLA7078M)의 온도를 알려줍니다.



(스텝모터로 1 축 위치제어를 구현한 예)



(USD 의 가감속 속도프로파일)

```

typedef enum {FREE, HOLD, STOP, RUN, MOVE} SMD_CMD, MOC_CMD, USD_CMD;
typedef enum {CW, CCW} SMD_DIR, MOC_DIR, USD_DIR;
typedef enum {
    USD_FULL=1, USD_HALF=3, USD_QUARTER=4, USD_EIGHTH=5, USD_SIXTEENTH=6
} USD_STEP;

```

```

typedef struct {
    EventHead      e;
    union {
        struct {
            USD_CMD    cmd     :4;
            USD_STEP   step    :3;
            USD_DIR    dir     :1;
        };
        uint16 config;
    };
}

```

```

    };
    uint16 ipps;      // initial PPS
    uint16 accel;     // accel,decel speed. [pulse/ms]
    uint32 pps;       // steady state speed [pulse/second]
    uint32 move;      // total pulse of rotation.
} UsdOutEvent;

```

USD 모듈은 사다리꼴 가감속 프로파일을 지원합니다. `ipps` 는 초기속도, `accel` 은 밀리초(ms)당 가감속할 펄스개수이며, `pps` 는 정속구간의 회전속도를 나타내며, `move` 는 MOVE 명령에서의 이동(회전)할 펄스의 총개수입니다.

`cmd` 는 모터의 회전동작을 제어하는 명령입니다. `FREE` 는 모터전류를 차단하여 정지시키고, `HOLD` 는 모터전류를 훌리면서 정지시켜 홀딩토크를 발생합니다. `STOP` 은 감속정지후 HOLD 상태로 되며, `RUN` 은 가속후 연속회전(`pps` 의 속도로)시키는 명령이며, `MOVE` 는 설정된 펄스개수(`move`)만큼 가감속프로파일을 따라 이동(회전)합니다.

`step` 은 마이크로스테핑 미세회전을 제어하는 명령으로, `USD_FULL` 은 한 스텝의 회전을 위해서 한 개의 펄스가 필요하므로 고속회전에 적합하며, `USD_SIXTEENTH` 는 16 개의 펄스가 한 스텝만큼 회전시키므로, 스텝당 1.8 도 회전하는 일반적인 유니폴러모터의 경우 `USD_SIXTEENTH` 를 선택하면 한개의 펄스는 0.1125 도 회전하게 됩니다.

`dir` 은 회전방향을 지시하는 것으로 `CW` 는 시계방향으로, `CCW` 는 반시계방향으로 회전합니다.

```

typedef struct {
    __BASE_PARAM;

    uint32 slowJogPPS;           // jogSpdType=Volumn(2) 일 경우 최저 회전속도
    uint32 fastJogPPS;          // jogSpdType=Volumn(2) 일 경우 최고 회전속도
    uint16 jogAccel;             // 조그동작에 적용할 가감속
    uint8 dJVol;                 // 볼륨인 경우 변동검출 기준치
    uint8 jogType;               // NoConnect(0), Switch(1), Volume(2)
    uint16 decelTime;            // NNP 검출시 감속시간[밀리초]
    float dVM;                  // 모터구동전압 변동검출 기준치
    uint8 dHeat;                 // 온도변동 검출 기준치
    uint8 upperHeat;             // 팬제어 목표온도, 팬듀티 100%
    uint8 lowerHeat;             // 팬제어 시작온도, 팬듀티 50%
} UsdParamEvent,UsdParamChangedEvent,UsdParamNotifiedEvent;

```

이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. `ezConfig.exe` 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(`usd_drv.c`)에 정의된 `param` 변수를 사용해서 `param.slowJogPPS` 와 같이 읽습니다.

- `__BASE_PARAM`  
“10.22 XxxParamChangedEventArgs : EEPROM 설정 데이터” 참고
- `slowJogPPS, fastJogPPS`  
조그의 최저 PPS 와 최대 PPS 를 설정합니다. JSPI 단자(11 번핀)에 볼륨타입

포텐시미터를 연결하면 포텐시미터의 위치에 비례하여 최저 PPS에서 최대 PPS 까지 속도가 가변됩니다. 만일 JSPD에 스위치를 연결하면 OFF 일때 최저 PPS, ON 일때 최대 PPS로 회전하는 2 단속도의 조그가 가능합니다.

#### ■ jogAccel

조그 속도의 변경시(조그스위치 또는 볼륨의 조작으로) 사용할 가감속 기율기.  
[펄스/ms]

#### ■ dJVol

볼륨타입 조그 사용시 A/D 변환결과는 8비트(0~255)를 사용합니다. 볼륨값에 변동이 있는지 판단하는 기준이 되는 값입니다. dJVol 보다 큰 볼륨값의 변동은 조그속도를 변경시킵니다.

#### ■ jogType

조그를 사용하지 않으면 0, 스위치타입 조그는 1, 볼륨타입 조그는 2 입니다.

#### ■ decelTime

CCW 회전중 NNP 를 만나면 안전을 위하여 자동감속을 시작하는데, 최저속도인 slowJogPPS 까지 감속하는데 걸리는 시간을 설정합니다.

#### ■ dVM

외부에서 공급하는 모터구동용 전원의 전압변동을 검사하는 판단 기준.

#### ■ dHeat

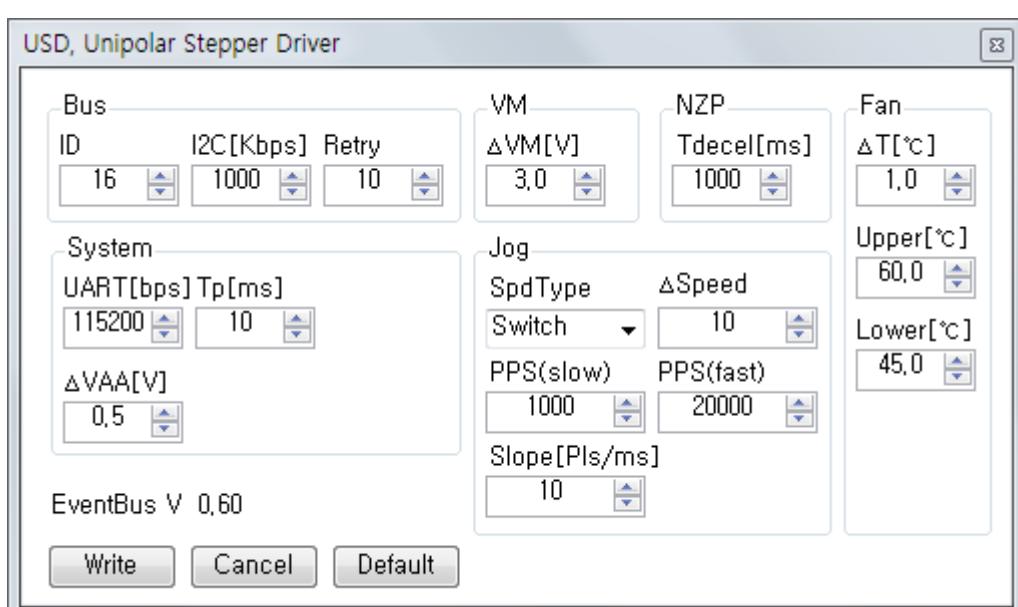
모터구동칩 온도의 변동을 검사하는 판단기준.

#### ■ upperHeat, lowerHeat

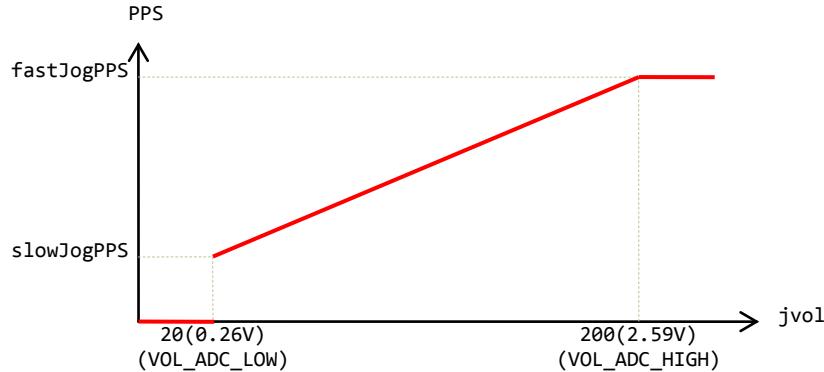
쿨링팬 제어를 위한 온도의 상하한치.

lowerHeat에서 회전을 시작하고, upperHeat에서 최대속도로 쿨링팬이 동작.

온도가 upperHeat 를 넘으면 UsdInEvent 의 변수 fan 이 세트되고 모터전류는 차단된다.

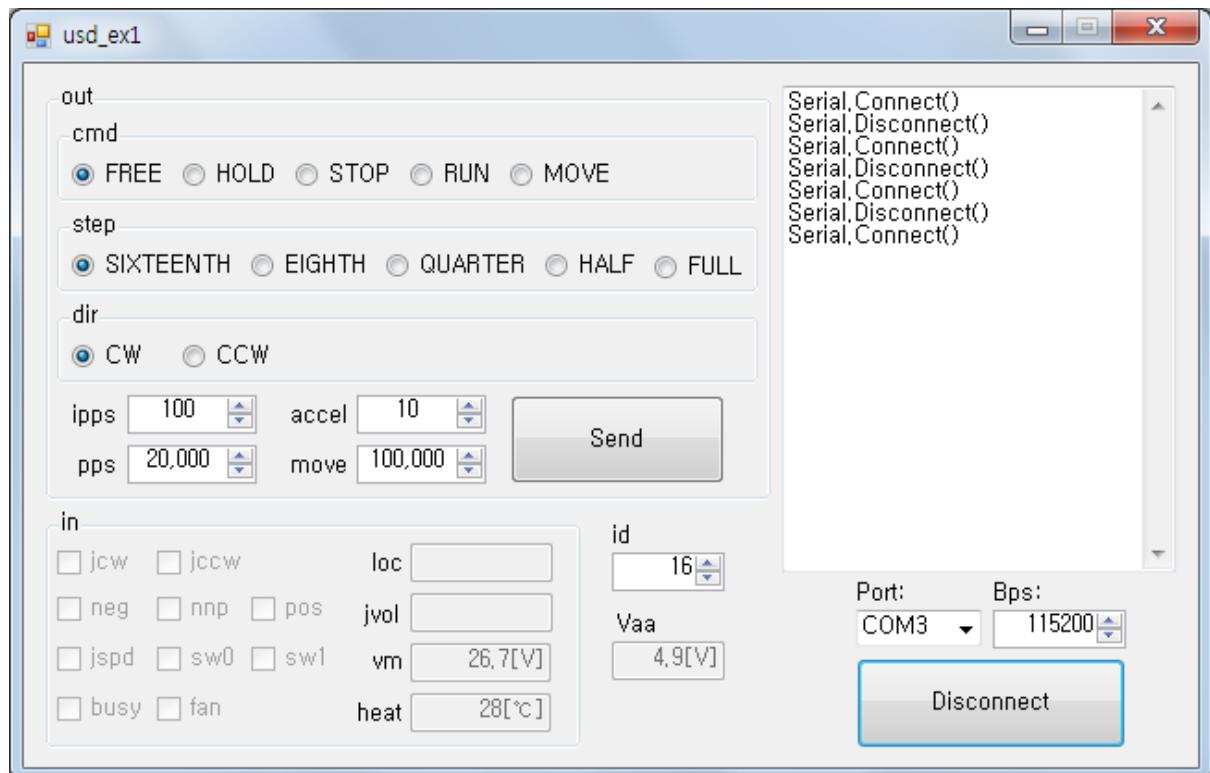


`ezconfig.exe`의 설정창입니다. USD 모듈을 ColinkEx 또는 UUM으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 `UsdParamEvent`과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM에 영구 저장됩니다. V0.60부터 컴퓨터와 연결속도를 `ezconfig.exe`에서 설정할 수 있으며, 기본속도는 115,200bps입니다.



그림은 볼륨타입 조그를 사용할 때 볼륨의 A/D 변환값(0~255)과 조그 PPS 와의 관계를 나타냅니다. 변환값(jvol)이 20 보다 작으면 정지합니다 (`usd_def.c`에서 변경가능). 그러므로, JCW/JCCW에 연결된 스위치로 방향을 설정 한 후 볼륨만으로 정지부터 회전속도조정까지 모든 작업을 할 수 있습니다.

\***응용**\* 푸시버튼을 한번 누를때마다 일정거리(펄스) 이동하는 조그동작을 구현하고 싶다면, NNP, SW0, SW1 중 하나에 푸시버튼을 연결하고, UsdIn 이벤트핸들러에서 MOVE 명령으로 원하는 펄스만큼 이동하도록 프로그램 하면 됩니다. (NNP는 사용자입력 처리가 디폴트이며, NNP 동작이 옵션입니다)

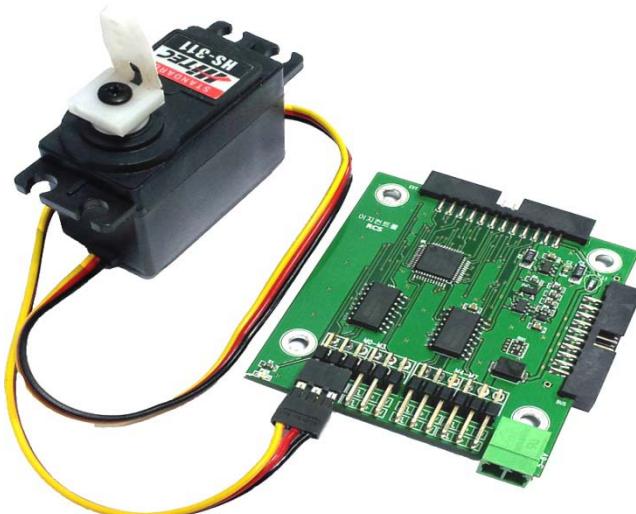


`eventbus_dotnet_0.60\example\usd_ex1\bin\Debug\usd_ex1.exe` 예제프로그램을 실행하면 그림처럼 USD 를 테스트할 수 있습니다. 연결된 시리얼포트를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. 위 그림처럼 설정 후 [Send] 버튼을 클릭하면 연결된 모터가 회전합니다. v0.50 이후부터는 Vaa 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(`usd_def.c` 참고)

## 6.16. RCS, RC-Servo Motor Controller



정해진 위치로 회전하므로 완전히



DC 모터를 사용하다보면 속도제어 위치제어에 어려움을 겪습니다. 그래서 제어용으로 스텝모터를 많이 사용합니다. 그러나 스텝모터도 원점을 잡기위해 센서가 필요한데, 이때 편리하게 사용할 수 있는 것이 RC 용으로 많이 사용되고 있는 RC 서보모터입니다. 특히 요즘 소형 관절형 로봇이 인기를 끌면서 많이 이용되고 있습니다. RC 서보모터는 일정한 펄스폭의 신호를 모터에 가하면 항상 정해진 위치로 회전하므로 완전히 오픈루프제어가 가능해서 전용기나 자동화 기기의 제작에 응용하면 시스템을 획기적으로 단순화시켜줍니다. RCS 모듈은 최대 8 개의 RC 서보모터를 연결해서 회전시킬 수 있도록 설계되었습니다. 2 핀의 I/O 커넥터에 모터동작 전원(보통 5~6V 사용)을 공급할 수 있습니다. I/O 커넥터에 모터동작 전원을 연결하지 않으면 버스커넥터에서 공급받는 VAA 를 사용하도록 구성되어 있지만, 이 경우 2A 이하의 전류만 사용할 수 있으며, 시스템에 노이즈를 발생할 수 있으므로 테스트용이나 간단한 동작에만 사용하는 것이 좋습니다.

모터연결 (핀헤더)커넥터 핀사양

| 핀 | 명 칭 | 기 능          | 기 타                                   |
|---|-----|--------------|---------------------------------------|
| 1 | M0  | pwm0, V+, 0V | 3 핀이 1 쌍으로 모터에 연결됨.<br>PCB 상의 실크를 참고. |
| 2 | M1  | pwm1, V+, 0V |                                       |
| 3 | M2  | pwm2, V+, 0V |                                       |
| 4 | M3  | pwm3, V+, 0V |                                       |
| 5 | M4  | pwm4, V+, 0V |                                       |
| 6 | M5  | pwm5, V+, 0V |                                       |
| 7 | M6  | pwm6, V+, 0V |                                       |
| 8 | M7  | pwm7, V+, 0V |                                       |

모터전원 입력용 터미널블럭 핀사양

| 핀 | 명 칭 | 기 능     | 기 타                |
|---|-----|---------|--------------------|
| 1 | VM  | 모터구동전원  | 통상 5~6V 사용(모터사양참고) |
| 2 | GND | 0V 전원입력 | 최대허용 전압은 18V 임     |

확장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                     |
|---|------------|--------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 2  | P2.1/SCK1                  | GPIO / SPI1 Serial Clock                                 |
| 3  | P2.2/MISO1                 | GPIO / SPI1 Master Input Slave Output                    |
| 4  | P2.3/MOSI1                 | GPIO / SPI1 Master Output Slave Input                    |
| 5  | P2.4                       | GPIO                                                     |
| 6  | P2.5                       | GPIO                                                     |
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P3.0                       | GPIO                                                     |
| 14 | P3.1                       | GPIO                                                     |
| 15 | P3.2                       | GPIO                                                     |
| 16 | P3.3                       | GPIO                                                     |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로 사용 전원 출력                                    |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC) 전원 출력                                         |
| 24 | GND                        |                                                          |

```

typedef struct {
    MType      mtype;
    ID         id;
    RcsInEvent* in;
    RcsOutEvent* out;
} RCS;

```

RCS 모듈의 이미지입니다(eventbus.h 참조). mtype은 모듈의 타입으로 Rcs 타입이 됩니다. id는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 스위치 입력은 RcsInEvent 구조체에 저장되며, 출력은 RcsOutEvent 구조체에 저장됩니다.

```
typedef Event RcsInEvent;
```

RCS 모듈은 출력전용의 모듈이므로 입력은 기본이미지 Event를 재정의하여 사용합니다.

```

typedef struct {
    EventHead e;
    uint16     pwm[8];
} RcsOutEvent;

```

RCS 모듈에 연결된 서보모터의 PWM 출력의 펄스의 HIGH 신호 폭을 us(마이크로초) 단위로 설정하는 배열입니다.

예) 채널 0에 하이텍의 HS-311 서보모터를 연결하고 HIGH 신호폭을 1500us로 설정하면 모터는 중립위치로 회전합니다. 이 경우 아래처럼 프로그램합니다.

```
rcs->out->pwm[0] = 1500;
```

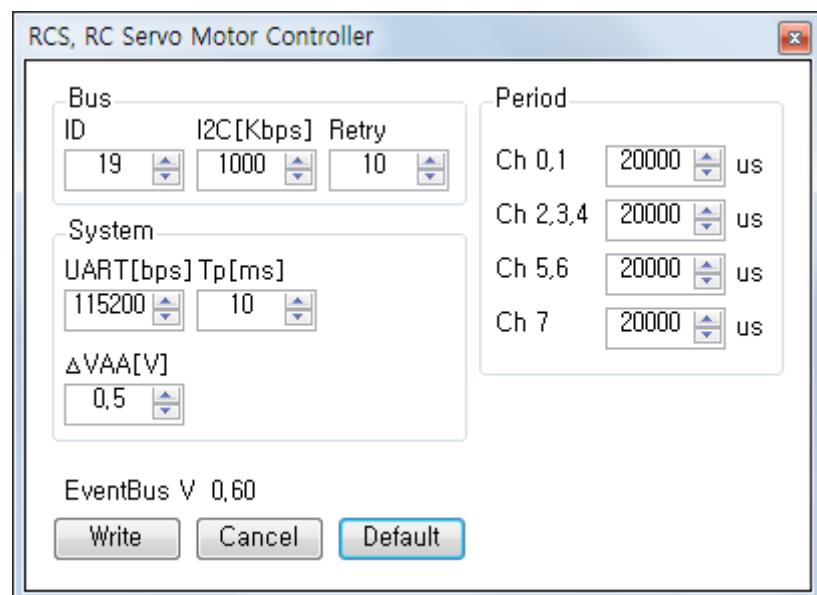
```

typedef struct {
    __BASE_PARAM;
    uint16 period[4]; // cycle time in us
} RcsParamEvent, RcsParamChangedEvent, RcsParamNotifiedEvent;

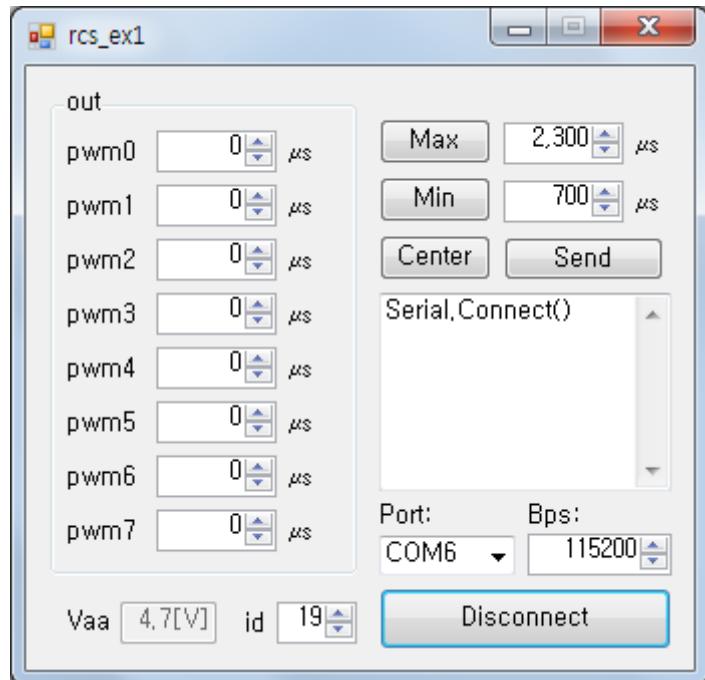
```

이 구조체는 EEPROM에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(rcs\_drv.c)에 정의된 param 변수를 사용해서 param.period[0]와 같이 읽습니다.

- **\_\_BASE\_PARAM**  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- **period**  
각 채널의 pwm 출력신호의 주기를 us(마이크로초) 단위로 설정합니다. 4 그룹으로 구성되고 그룹별로 주기를 변경할 수 있습니다.



ezconfig.exe의 설정창입니다. RCS 모듈을 ColinkEx, U2M 또는 UUM으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 `RcsParamEvent`과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM에 영구 저장됩니다. V0.60부터 컴퓨터와 연결속도를 ezconfig.exe에서 설정할 수 있으며, 기본속도는 115,200bps입니다.



eventbus\_net\_0.60\example\rcs\_ex1\bin\Debug\rcs\_ex1.exe 예제프로그램을 실행하면 그림처럼 RCS 를 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. V0.50 이후부터는 Vaa 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(rcs\_def.c 참고)

## 6.17. TCM, Thermocouple Measurement Module



6 채널의 써모커플 온도측정을 지원하는 모듈입니다. 16 비트 A/D 변환기를 사용하여 기본 0.1 도의 분해능을 구현하였으며, 콜드정션 온도센서 내장으로 절대온도를 측정할 수 있습니다. K, T, E 타입 써모커플을 지원하며(단, 전채널 모두 동일타입으로 설정), -4mV ~ 23mV 의 측정범위로 K 타입(약 -100~550 도), T 타입(약 -120~400 도), E 타입(약 -70~320 도)의 측정범위를 가집니다 (기타 온도범위는 주문시 별도문의). 상용전원 노이즈를 제거하는 50/60Hz 노이즈 필터의 내장으로 더욱 정밀한 온도측정이 가능합니다.

**\*주의\*** 연결되는 써모커플의 정션은 채널간 절연 후 사용하십시오. 그렇지 않으면 채널간 전기적 간섭으로 정확한 온도를 측정할 수 없습니다. 또한 정션은 외부 샤시(프레임)와도 절연하여야 합니다. 즉, 써모커플의 정션은 전기적으로 완전히 절연(코팅)시킨 후 사용하십시오. 그리고 커넥터와 플러그의 온도가 평형상태일 때 정확한 온도측정이 가능하므로 설치후 약 10 여분의 시간이 지난후 측정하는 것이 좋습니다. 플러그를 손으로 잡으면 손의 온도가 전도되어 온도가 변하므로 일정시간 후 측정하는 것이 좋습니다.

IO 커넥터 핀사양

| 핀  | 명칭   | 기능        | 기타 |
|----|------|-----------|----|
| 1  | TC0+ | 써모커플 채널 0 |    |
| 2  | TC0- |           |    |
| 3  | TC1+ | 써모커플 채널 1 |    |
| 4  | TC1- |           |    |
| 5  | TC2+ | 써모커플 채널 2 |    |
| 6  | TC2- |           |    |
| 7  | TC3+ | 써모커플 채널 3 |    |
| 8  | TC3- |           |    |
| 9  | TC4+ | 써모커플 채널 4 |    |
| 10 | TC4- |           |    |
| 11 | TC5+ | 써모커플 채널 5 |    |
| 12 | TC5- |           |    |

확장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5 | P2.4       | GPIO                                  |
| 6 | P2.5       | GPIO                                  |
| 7 | P2.6       | GPIO                                  |
| 8 | P2.7       | GPIO                                  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.1/AD2/CT32B1_MAT0       | GPIO/ADC2 / Match output 0 for 32-bit timer 1            |
| 14 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 15 | P1.4/AD5/CT32B1_MAT3       | GPIO / ADC5 / Match output 3 for 32-bit timer 1          |
| 16 | VAA                        | VAA(5VDC) 버스전원                                           |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로직용 전원 출력                                     |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC) 전원 출력                                         |
| 24 | GND                        |                                                          |

```
typedef struct {
    MType          mtype;
    ID             id;
    TcmInEvent*   in;
    TcmOutEvent*  out;
} TCM;
```

TCM 모듈의 이미지입니다(eventbus.h 참조). mtype 은 모듈의 타입으로 Tcm 타입이 됩니다. id 는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 스위치 입력은 TcmInEvent 구조체에 저장되며, 출력은 TcmOutEvent 구조체에 저장됩니다. (V0.60 부터 TcmInChangedEvent, TcmOutChangedEvent 가 간략히 TcmInEvent, TcmOutEvent 로 명칭변경됨)

```
typedef struct {
    EventHead  e;
    float      tc[NUM_TC];      // 측정결과[°C]
} TcmInEvent;
```

TCM 모듈이 측정한 온도는 배열 tc 에 저장됩니다.

예) 채널 0 의 온도를 읽는다.

```
float temperature = tcm->in->tc[0];
```

```
typedef Event     TcmOutEvent;
```

TCM 모듈은 입력전용의 모듈이므로 출력이 없어 기본이미지 Event 를 재정의하여 사용합니다.

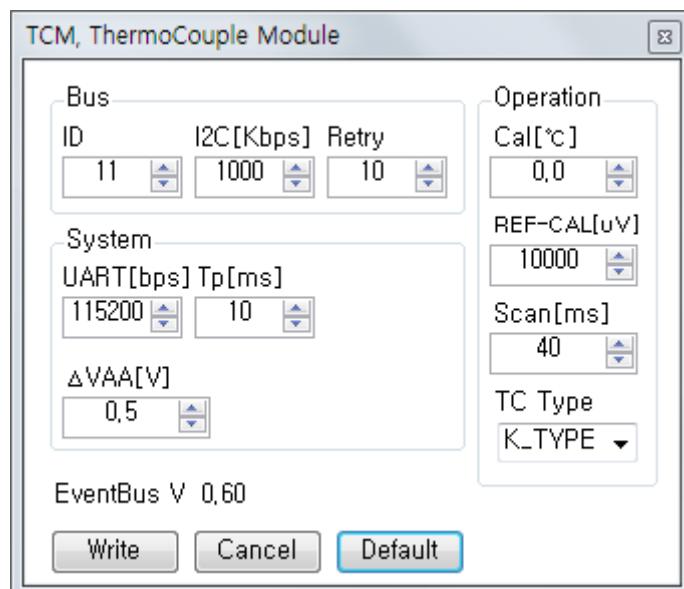
```

typedef struct {
    __BASE_PARAM;
    float calTemp;          // 온도보정[°C]
    uint32 scanTime;        // ADC scanning time [ms]
    uint16 calVolt;         // 전압보정[uV]
    TCType tcType;          // 열전대 타입
} TCM_PARAM, TcmParamChangedEvent, TcmParamNotifiedEvent;

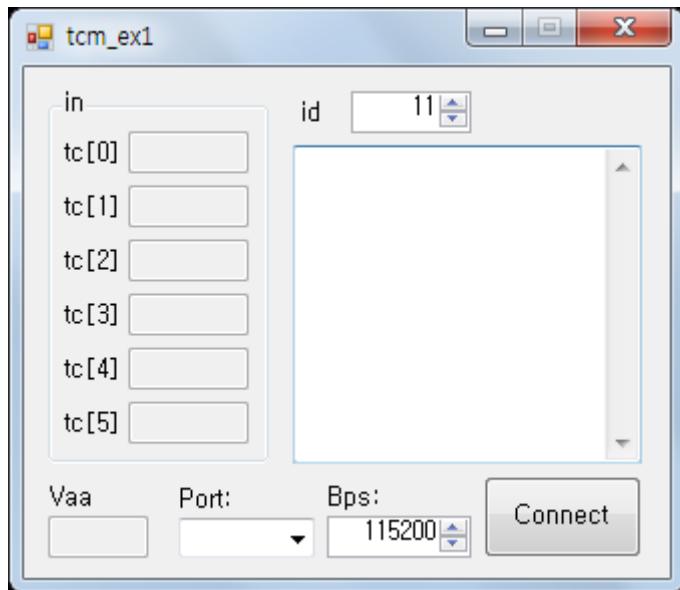
```

이 구조체는 EEPROM에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(tcm\_drv.c)에 정의된 param 변수를 사용해서 param.calTemp와 같이 읽습니다.

- **\_\_BASE\_PARAM**  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- **calTemp**  
온도보정값입니다. 측정결과에 이 보정치를 더한 값이 실제 출력이 됩니다.
- **scanTime**  
각 채널의 써모커플을 측정하는 시간입니다. 최소 40ms 이상입니다. 측정은 총 8 개입력(6 채널과 보정용 2 채널)이므로 최소시간은 320ms가 되어 초당 3 번까지 측정이 가능합니다.
- **calVolt**  
자동보정용 전압값입니다. 모듈의 우측에 두개의 측정용 TP인 CAL, REF가 있습니다. 이 두점 사이의 전압을 마이크로볼트 단위로 측정한 값입니다. 그러므로 매우 정밀한 계측기를 사용하여 측정하여야 합니다. (5-1/2 Digit 이상)
- **tcType**  
써모커플의 종류를 설정한 값입니다. K\_TYPE(0), T-TYPE(1), E\_TYPE(2)입니다. eventbus.h TCType 열거형 참고.



`ezconfig.exe` 의 설정창입니다. TCM 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 `UsdParamEvent` 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 `ezconfig.exe` 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.  $Ca1[\mu V]$  항목은 기준전압 측정용 테스트포인트인 CAL~REF 간의 전압입니다. 이 전압을 설정후 아래의 테스트용 프로그램 `tcm_ex1.exe` 를 이용하여 온도를 측정하십시오. 이때 온도오차가 발생한다면  $Ca1[^\circ C]$  항목에 온도차를 입력하면 온도오프셋이 조정됩니다. `Scan[ms]`는 한개 채널을 읽는데 소요되는 시간으로 40ms 가 최소시간입니다. 6 개 채널과 자동교정용 2 개채널이 있으므로  $40ms * 8 = 320ms$  마다 전체널 온도가 측정완료됩니다. 즉, 초당 3 회이상의 측정은 불가합니다.



`eventbus_net_0.60\example\tcm_ex1\bin\Debug\tcm_ex1.exe` 예제프로그램을 실행하면 그림처럼 TCM 을 테스트할 수 있습니다. 연결된 시리얼포트(Port)를 선택하고, 통신속도(Bps)는 기본적으로 설정된 115,200bps 를 사용합니다. 써모커플이 연결되지 않은 채널은 측정가능한 최소온도를 표시합니다. V0.50 이후부터는 Vaa 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(`tcm_def.c` 참고)

## 6.18. LAN, TCP/IP Communication Module



랜 네트워크를 구성하기 위한 이더넷 서버모듈로 최대 8 개의 TCP/IP 소켓을 지원하므로 공유기 또는 스위칭 허브에 연결하면 PC 와 같은 여러 대의 클라이언트에서 동시에 다수의 이지컨트롤러 네트워크에 액세스할 수 있습니다.

또한 POE(Power Over Ethernet) 기능을 지원하므로 POE 를 지원하는 스위칭허브를 사용하면 이지컨트롤러 네트워크에 별도의 전원공급 없이도 CAT5 랜선을 통하여 전원공급이 가능하므로 상용전원의 공급이 어려운 구석진 장소나 실외에도 쉽게 설치할 수 있으며, 스위칭 허브의 확장으로 장거리까지 전송이 가능합니다. 또한 중앙집중식 전원관리가 가능하므로 전원공급라인의 안정성을 구현할 수 있습니다.



버스케이블을 사용한 모듈의 확장은 I2C 통신의 물리적 제약으로 통신거리 및 연결모듈의 개수에 제약이 있으나, LAN 모듈을 사용하면 이러한 제약이 사라져 대규모 시스템을 구성할 수 있습니다. 그림은 POE 를 지원하는 4 채널 스위칭허브에 LAN+DIO 로 구성된 이지컨트롤러 네트워크를 연결한 경우입니다. 이런 방버브로 스위칭허브(또는 공유기)에 연결된 각각의 모듈들은 서로 별개의 ID 를 사용하므로 모듈의 확장에 사실상 제약이 없어 대규모 제어 및 감시를 위한 시스템을 구성할 수 있습니다.

이벤트버스의 전원 시스템은 **병렬동작**이 가능하므로 PSM 을 동시에 사용하면 더 큰 전류를 사용할 수 있습니다. (주의) 이경우, PSM 의 전원표시 LED 가 소등될 수 있으나 정상적 동작입니다.

확장커넥터 핀사양

| 핀 | 핀 명칭       | 핀 기능                                  |
|---|------------|---------------------------------------|
| 1 | P2.0/SSEL1 | GPIO / SPI1 Slave Select              |
| 2 | P2.1/SCK1  | GPIO / SPI1 Serial Clock              |
| 3 | P2.2/MISO1 | GPIO / SPI1 Master Input Slave Output |
| 4 | P2.3/MOSI1 | GPIO / SPI1 Master Output Slave Input |
| 5 | P2.4       | GPIO                                  |

|    |                            |                                                          |
|----|----------------------------|----------------------------------------------------------|
| 6  | P2.5                       | GPIO                                                     |
| 7  | P2.6                       | GPIO                                                     |
| 8  | P2.7                       | GPIO                                                     |
| 9  | P2.8                       | GPIO                                                     |
| 10 | P2.9                       | GPIO                                                     |
| 11 | P2.10                      | GPIO                                                     |
| 12 | P2.11                      | GPIO                                                     |
| 13 | P1.1/AD2/CT32B1_MAT0       | GPIO/ADC2 / Match output 0 for 32-bit timer 1            |
| 14 | P1.2/AD3/CT32B1_MAT1       | GPIO/ADC3 / Match output 1 for 32-bit timer 1            |
| 15 | P1.10/AD6/CT16B1_MAT1      | GPIO / ADC6 / Match output 1 for 16-bit timer 1          |
| 16 | P1.11/AD7                  | GPIO / ADC7                                              |
| 17 | SWDIO/P1.3/AD4/CT32B1_MAT2 | SWD IO / GPIO / ADC4 / Match output 2 for 32-bit timer 1 |
| 18 | TXD/P1.7/CT32B0_MAT1       | UART TX / GPIO / Match output 1 for 32-bit timer 0.      |
| 19 | SWCLK/P0.10/CT16B0_MAT2    | SWD clock / GPIO / Match output 2 for 16-bit timer 0     |
| 20 | RXD/P1.6/CT32B0_MAT0       | UART RX / GPIO / Match output 0 for 32-bit timer 0       |
| 21 | VCC                        | VCC(3.3VDC)로 사용 전원 출력                                    |
| 22 | GND                        |                                                          |
| 23 | VBB                        | VBB(24VDC) 전원 출력                                         |
| 24 | GND                        |                                                          |

```

typedef struct {
    MType      mtype;
    ID         id;
    LanInEvent* in;
    LanOutEvent* out;
} LAN;

```

LAN 모듈의 이미지입니다(eventbus.h 참조). mtype은 모듈의 타입으로 Lan 타입이 됩니다. id는 모듈의 주소이며 유일한 값을 가져야 합니다. 모듈의 입력이미지는 LanInEvent 구조체에, 출력이미지는 LanOutEvent 구조체에 저장됩니다. (V0.6부터 LanInChangedEvent, LanOutChangedEvent가 간략히 LanInEvent, LanOutEvent로 명칭변경됨)

```

typedef Event LanInEvent;
typedef Event LanOutEvent;

```

LAN 모듈은 입출력이 없는 통신전용 모듈이므로 기본이미지 Event를 재정의 합니다

```

typedef struct {
    __BASE_PARAM;
    float      dVBB;
    uint16     retryTime;    // retransmission time, 1 per 100us
    uint8      retryCount;   // retransmission count
    uint16     port;
}

```

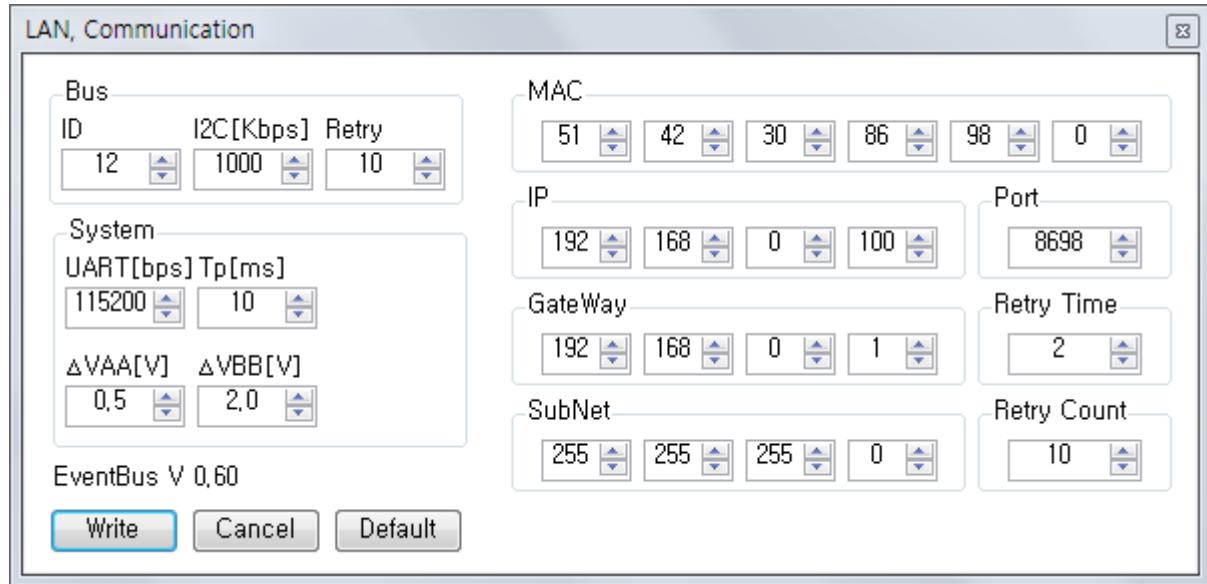
```

    uint64      mac;
    uint32      ip;
    uint32      subnet;
    uint32      gateway;
} LanParamEvent, LanParamChangedEvent, LanParamNotifiedEvent;

```

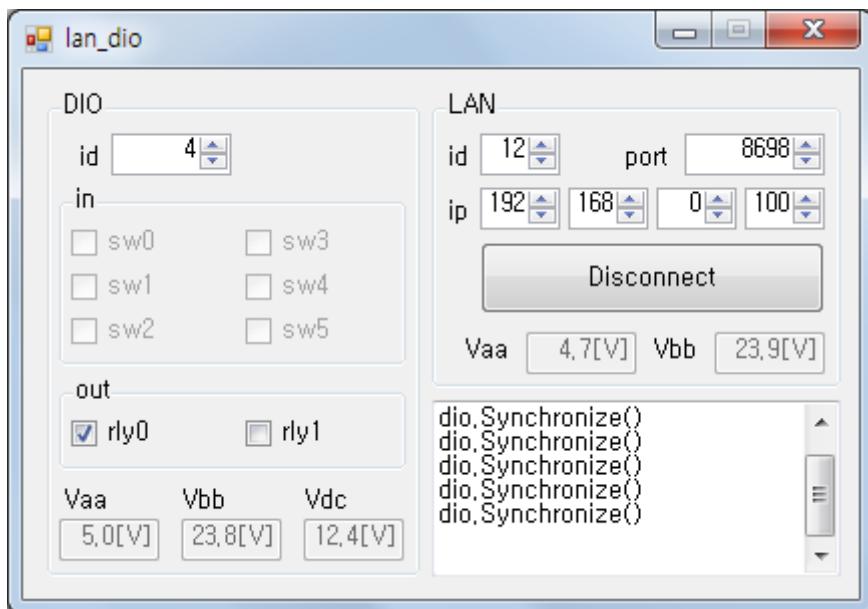
이 구조체는 EEPROM 에 저장되는 비휘발성 설정데이터의 관리에 사용됩니다. ezconfig.exe 프로그램으로 설정값을 수정할 수 있습니다. 펌웨어 작성에서 이 데이터를 참조하려면 드라이버(lan\_drv.c)에 정의된 param 변수를 사용해서 param.dVBB 와 같이 읽습니다.

- **\_BASE\_PARAM**  
“10.22 XxxParamChangedEvent : EEPROM 설정 데이터” 참고
- **dVBB**  
**\_BASE\_PARAM** 의 dVAA 참고.
- **retryTime, retryCount**  
전송 실패시 재전송을 시도할 시간간격 및 재시도횟수.
- **ip, port**  
LAN 모듈은 임베디드 TCP/IP 서버로 동작하며 소켓통신을 사용합니다. ip 는 모듈의 주소이며, port 는 소켓의 포트번호입니다. ip 는 공유기 또는 허브에 할당된 주소를 확인후 사용하지 않는 주소를 할당하십시오. 예를 들어 컴퓨터의 ip 가 192.168.0.2 라면 앞의 3 자리인 192.168.0 까지는 동일하며 마지막 주소를 100 처럼 사용하고 있지 않은 주소를 할당합니다. 공유기에서 높은 주소(예 100 번지)를 허용하지 않는 경우도 있으니 반드시 게이트웨이의 설정을 확인하십시오. 소켓은 총 8 개를 사용할수 있으며 포트번호는 port ~ (port+7)까지 8 개 입니다.
- **mac**  
모든 네트워크 장치에 대해 유일한 Hardware address(Ethernet MAC address)값을 Ethernet MAC layer 에서 사용하도록 정해져 있습니다. 그렇지 않으면 주소충돌로 제대로 동작하지 않을 수 있습니다. 이 MAC address 의 할당은 IEEE 에서 관리하고 있으며, IEEE 로부터 할당 받은 MAC address 를 부여해야 합니다. 그러나, LAN 모듈을 인터넷에 연결하지 않을 경우 MAC 주소는 임의로 정해서 사용해도 됩니다. 만일 인터넷에 연결하려면 사용하지 않는 랜카드와 같은 인터넷장비에 할당된 MAC 주소를 사용하면 됩니다.
- **subnet**  
서브넷 마스크 입니다. 255.255.255.0 과 같이 네트워크를 구분하는 마스크입니다.
- **gateway**  
게이트웨이 주소입니다. 공유기를 사용하는 경우 192.168.0.1 에 할당된 경우가 많지만, 공유기마다 다를 수 있으므로 공유기의 메뉴얼을 참고하십시오.



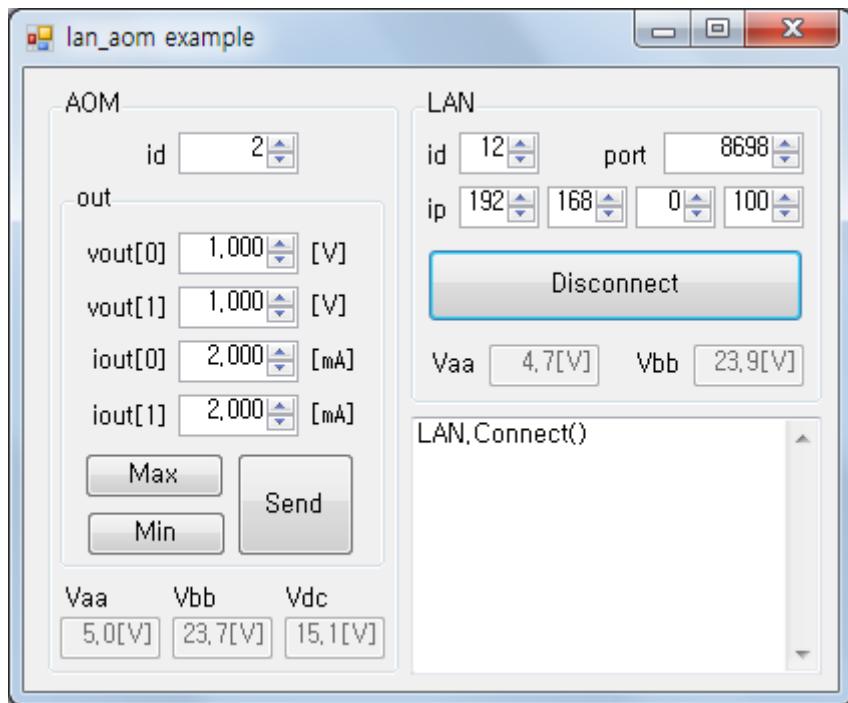
ezconfig.exe 의 설정창입니다. LAN 모듈을 ColinkEx 또는 UUM 으로 연결 후 “Connect” 버튼으로 접속하면 이 설정창이 뜹니다. 각 항목이 위의 LanParamEvent 과 동일한 것을 볼 수 있습니다. 이 값들은 EEPROM 에 영구 저장됩니다. V0.60 부터 컴퓨터와 연결속도를 ezconfig.exe 에서 설정할 수 있으며, 기본속도는 115,200bps 입니다.

위의 경우 IP 주소는 192.168.0.100, 포트는 8698 번을 사용하도록 설정하였습니다.



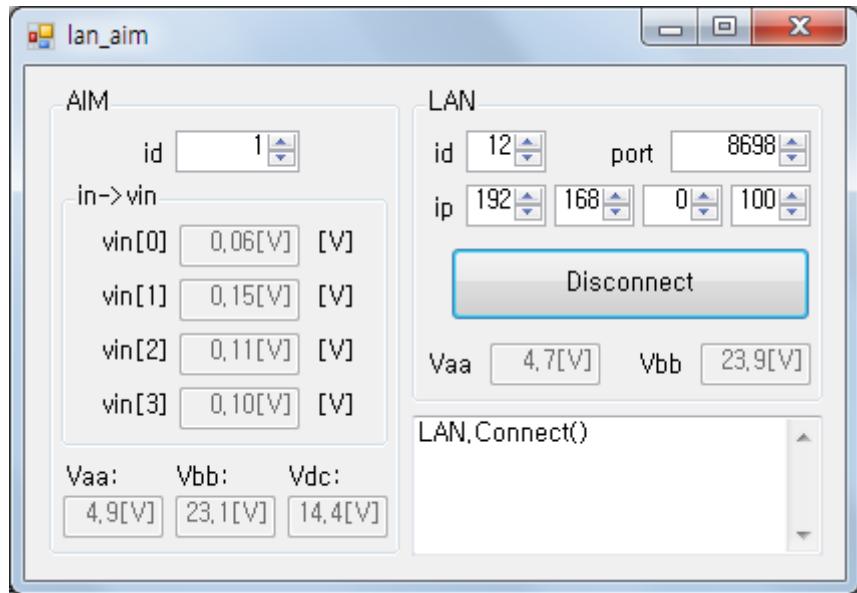
eventbus\_net\_0.60\example\lan\_ex1\bin\Debug\lan\_ex1.exe 예제프로그램을 실행하면 그림처럼 LAN 과 DIO 를 연결하여 사용하는 경우를 테스트할 수 있습니다. Connect 버튼으로 TCP/IP 소켓으로 연결하면 LAN, DIO 의 전압이 표시됩니다. 그리고 스위치와 릴레이를 동작하면 결과를 확인할 수 있습니다. V0.50 이후부터는 Vaa, Vbb, Vdc 전압표시는 처음 연결할 때 한번만 값을 읽어 표시하고 이후 값이 변하여도 갱신하지 않도록 펌웨어가 프로그램되어 있습니다(lan\_def.c 참고)

위 그림의 경우  $V_{bb}$  가 24V 에 못미칩니다. 그래서  $V_{dc}$  도 12V 에 못미칩니다. 이렇게 전압이 낮으면 릴레이가 정상적으로 동작하지 않습니다. 전원을 보강해 주어야 합니다. (위의 경우는 USB 모듈의 전원을 LAN 모듈에 공급하도록 시스템을 구성해서 전원이 부족해 졌습니다. LAN 모듈은 더 많은 전류를 소모하기 때문입니다.)



`eventbus_net_0.60\example\lan_aom\bin\Debug\lan_aom.exe` 예제프로그램을 실행하면 그림처럼 LAN 과 AOM 을 연결하여 사용하는 경우를 테스트할 수 있습니다. ip 와 port 를 `ezconfig.exe`에서 설정한 값과 동일하게 맞춘 후, Connect 버튼으로 통신에 연결하면 LAN, AOM 의 전압이 표시됩니다. 그리고 출력할 전압 및 전류를 설정후 [Send] 버튼을 클릭하면 결과를 확인할 수 있습니다.

위 그림의 경우  $V_{bb}$  가 24V 에 못미칩니다. 그래서  $V_{dc}$  도 13V 에 못미칩니다. 이렇게 동작전압이 낮으면 정상적으로 전압이 출력되지 않습니다. 전원을 보강해 주어야 합니다. (위의 경우는 USB 모듈의 전원을 LAN 모듈에 공급하도록 연결해서 전원이 부족해 졌습니다. LAN 모듈은 더 많은 전류를 소모하기 때문입니다.)

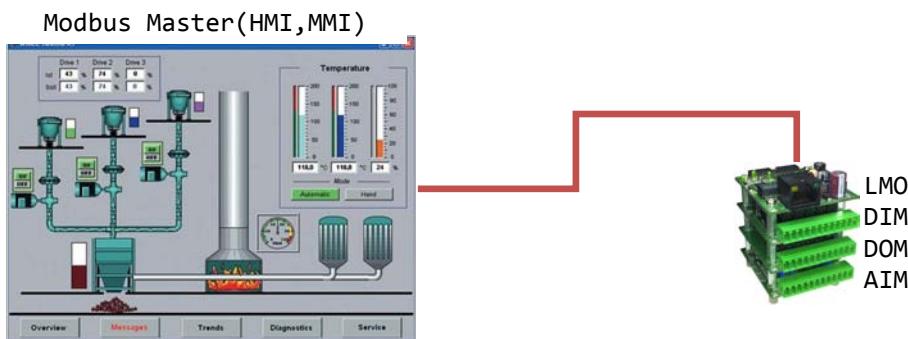


eventbus\_net\_0.60\example\lan\_aom\bin\Debug\lan\_aim.exe 예제프로그램을 실행하면 그림처럼 LAN 과 AIM 을 연결하여 사용하는 경우를 테스트할 수 있습니다. ip 와 port 를 ezconfig.exe 에서 설정한 값과 동일하게 맞춘 후, Connect 버튼으로 통신에 연결하면 LAN, AIM 의 전압이 표시됩니다.

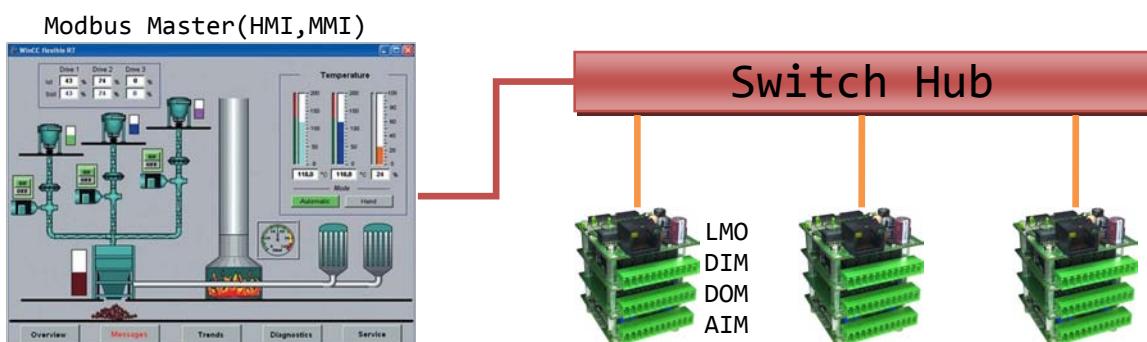
## 6.19. LMO, Modbus Tcp/ip 인터페이스 모듈

**<알림>** LMO 는 HMI 사용자를 위해 개발된 제품이지만 이지컨트롤러의 이벤트버스방식에 대한 이해가 없다면 사용이 불편할 수 밖에 없습니다. 특히, 사용할 출력모듈들을 LMO 모듈에 참조모듈로 등록해야하므로, ColinkEx 를 이용해서 펌웨어에 참조모듈을 생성 후 다운로드 하여야 합니다. 그러므로, 양쪽 모두를 원활히 사용할 수 있는 사용자께서 적용하시면 좋습니다.

LMO 는 LAN 모듈을 사용하여 모드버스에 연결할 수 있도록 펌웨어만 변경한 모듈입니다. 그러므로 LAN 모듈을 보유한 사용자는 펌웨어만 새로 다운로드하고 ezconfig.exe 로 설정하면 LMO 로 사용할 수 있습니다. LMO 를 사용하면 HMI(MMI)와 같은 Modbus Master 에 이지컨트롤러를 연결할 수 있게 되므로 수 많은 모드버스 장치들과 함께 사용할 수 있게됩니다.



MODBUS 지원 HMI 에 1:1 로 연결한 구성



MODBUS 지원 HMI 에 1:N 으로 연결한 구성

위에서 처럼 DIM,DOM,AIM 3 개의 모듈을 사용하려면 application\default\lmo\_def.c 에 아래 프로그램처럼 3 개의 모듈을 등록해야 합니다.

```
int main()
{
    // create modules
    // **important** LMO must create modules connected to bus
    DIM* dim = CreateDIM(ID_DIM1);
    DOM* dom = CreateDOM(ID_DOM1);
    AIM* aim = CreateAIM(ID_AIM1);
```

```

// start eventbus
EventBus();

// variable definition, starting code

EventProgram;
IF (this, ModbusFrameReceived) {
    ProcessModbusFrame(pe);
}

ScanProgram;
Aux_ReceiveEvent();

End;
}

```

위 프로그램을 빌드 후 LMO 모듈에 다운로드해야 3 개의 모듈을 엑세스할 수 있습니다.

아래는 모드버스 TCP/IP 에서 사용하는 통신패킷의 구성입니다.

| TID<br>(Transaction ID) | PID<br>(Protocol ID) | Length | UID<br>(Unit ID) | FC    | Data |
|-------------------------|----------------------|--------|------------------|-------|------|
| 2byte                   | 2byte                | 2byte  | 2byte            | 1byte | 가변길이 |

TID : 패킷을 전송할 때마다 증가하는 패킷 번호

PID : 프로토콜 종류. 모드버스는 0x0000

Length : 뒤 따르는 바이트 개수. UID,FC,Data 의 길이의 합

UID : 모듈의 ID(ezconfig.exe 에서 설정한 ID)

FC : 모드버스 평선클. 아래 도표 참고

Data : 각 평선클마다 약속된 파라메타 데이터

(표) LMO 지원 모드버스 평션

| 모드버스 평션     |      | 읽기         | 쓰기                         |
|-------------|------|------------|----------------------------|
| 입력<br>(스위치) | 비트단위 | FC2(multi) |                            |
|             | 워드단위 | FC4(multi) |                            |
| 출력<br>(릴레이) | 비트단위 | FC1(multi) | FC5(single)<br>FC15(multi) |
|             | 워드단위 | FC3(multi) | FC6(single)<br>FC16(multi) |

표는 모드버스에서 사용하는 평션(FC, Function Code)중에서 LMO 가 지원하는 평션을 나열한 것입니다. 이 8 개의 평션을 사용하면 연결된 이지컨트롤러 모듈의 입출력을 읽고 쓸 수 있습니다. 예를 들어, FC1 은 릴레이와 같은 출력코일을 비트단위로 여러비트(multi) 읽을 수 있는 명령입니다. FC5 는 단지 하나의 비트(single)에만 쓸 수 있는 명령입니다. 여러비트에 쓰려면 FC15 를 사용하면 됩니다. FC4 는 스위치와 같은 입력을 워드단위(16 비트)로 읽을 수 있습니다.

## 각 평션의 패킷 구조(ModBus TCP/IP 표준)

### ■ FC2 (Read multi-bit switchs)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 비트수   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0D  | 02 | 00 00 | 00 10 |

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 바이트수 | 데이터   |
|-------|-------|--------|-----|----|------|-------|
| xx xx | 00 00 | 00 05  | 0D  | 02 | 02   | AB CD |

바이트수: 뒤따르는 바이트 데이터의 개수

\* 모든 주소는 0 번부터 시작

### ■ FC4 (Read multi-word switchs)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 포트수   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0D  | 04 | 00 00 | 00 01 |

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 바이트수 | 데이터   |
|-------|-------|--------|-----|----|------|-------|
| xx xx | 00 00 | 00 05  | 0D  | 04 | 02   | AB CD |

### ■ FC1 (Read multi-bit coils)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 비트수   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0E  | 01 | 00 00 | 00 10 |

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 바이트수 | 데이터   |
|-------|-------|--------|-----|----|------|-------|
| xx xx | 00 00 | 00 05  | 0E  | 01 | 02   | FF FF |

■ FC5 (Write single-bit coil)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 데이터   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0E  | 05 | 00 02 | FF 00 |

\* 데이터: FF00(ON), 0000(OFF)

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 시작주소  | 데이터   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0E  | 05 | 00 02 | FF 00 |

\* 응답은 요청과 동일

■ FC15 (Write multi-bit coils)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 비트수   | 바이트수 | 데이터 |
|-------|-------|--------|-----|----|-------|-------|------|-----|
| xx xx | 00 00 | 00 08  | 0E  | 0F | 00 08 | 00 04 | 01   | 0F  |

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 시작주소  | 비트수   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0E  | 0F | 00 08 | 00 04 |

■ FC3 (Read multi-word coils)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 유디수   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 01  | 03 | 00 00 | 00 02 |

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 바이트수 | 데이터         |
|-------|-------|--------|-----|----|------|-------------|
| xx xx | 00 00 | 00 07  | 01  | 03 | 04   | 89 AB CD EF |

바이트수: 뒤따르는 바이트 데이터의 개수

■ FC6 (Write single-word coils)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 데이터   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0E  | 06 | 00 00 | FF FF |

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 시작주소  | 데이터   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 0E  | 06 | 00 00 | FF FF |

\* 응답은 요청과 동일

■ FC16 (Write multi-word coils)

요청프레임 : Master→Slave

| TID   | PID   | Length | UID | FC | 시작주소  | 워드수   | 바이트수 | 데이터         |
|-------|-------|--------|-----|----|-------|-------|------|-------------|
| xx xx | 00 00 | 00 0B  | 02  | 10 | 00 02 | 00 02 | 04   | 89 AB CD EF |

응답프레임 : Slave→Master

| TID   | PID   | Length | UID | FC | 시작주소  | 워드수   |
|-------|-------|--------|-----|----|-------|-------|
| xx xx | 00 00 | 00 06  | 02  | 10 | 00 02 | 00 02 |

위의 8 가지 평션을 이용하면 메모리의 특정값을 읽을 수 있습니다. 이지컨트롤러의 입력이미지는 입력평션으로, 출력이미지는 출력평션으로 읽고 쓸 수 있습니다. 특히 출력평션을 사용해서 출력이미지를 변경시키면 LMO 는 XxxOut 이벤트를 발생시켜서 변경된 값을 실제 하드웨어로 출력합니다. 예를들어, LMO+DIO 로 구성된 시스템에서 DIO 의 rly0 을 변경시키면 DioOut 이벤트가 발생해서 rly0 이 ON/OFF 됩니다. 그리고, DIO 의 스위치입력(sw)과 같은

입력이미지의 경우 자동으로 이벤트가 HMI에게 전송되지는 않습니다. LMO는 HMI의 평션을 수신한 경우에만 응답하게 됩니다.

## 엔디안

“모드버스는 빅엔디안을 사용하고, 아지컨트롤러는 리틀엔디안을 사용합니다” 그러므로 서로 데이터를 저장하는 순서가 달라서 그대로는 문제가 발생합니다.

엔디안(Endian)이란 정수형, 실수형처럼 여러바이트에 걸쳐 데이터가 저장될 때 “큰주소에 숫자의 끝자리를 저장하면 빅엔디안(Big-Endian)”, “작은주소에 숫자의 끝자리를 저장하면 리틀엔디안(Little-Endian)”이라 합니다.

| address       | 00         | 01 | 02     | 03 | ... |
|---------------|------------|----|--------|----|-----|
| Big-Endian    | 12         | 34 | AB     | CD | ... |
| Little-Endian | 34         | 12 | CD     | AB | ... |
| int16         | 0x1234     |    | 0xABCD |    |     |
| Big-Endian    | 12         | 34 | 56     | 78 | ... |
| Little-Endian | 78         | 56 | 34     | 12 | ... |
| int32         | 0x12345678 |    |        |    |     |

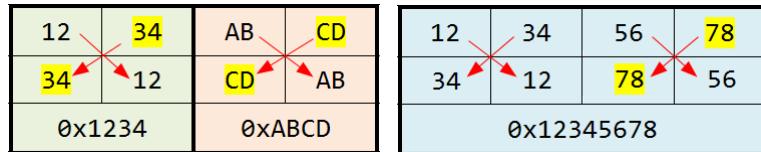
워드값(int16) 0x1234 와 0xABCD 를 저장할 때의 메모리 순서를 주의해서 봐 주십시오. 각각의 끝자리인 34 와 CD 가 오는 위치에 따라서 Big/Little 이 결정됩니다. 또한, 정수값(int32) 0x12345678에서도 끝자리인 78 이 오는 위치에 따라서 엔디안이 결정됩니다.

| address       | 00         | 01 | 02     | 03 | ... |
|---------------|------------|----|--------|----|-----|
| Big-Endian    | 12         | 34 | AB     | CD | ... |
| Little-Endian | 34         | 12 | CD     | AB | ... |
| int16         | 0x1234     |    | 0xABCD |    |     |
| Big-Endian    | 12         | 34 | 56     | 78 | ... |
| Little-Endian | 78         | 56 | 34     | 12 | ... |
| int32         | 0x12345678 |    |        |    |     |

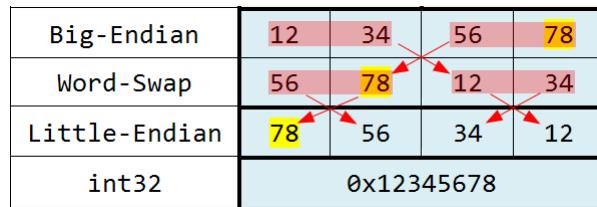
앞의 그림을 다른 관점으로 보면, address 가 오른쪽으로 증가할 때, Little-Endian(녹색화살표)을 제외한 모든 경우가 주소의 증가방향과 동일합니다(붉은화살표). Little-Endian 만 반대방향입니다. 그렇다고 Little-Endian 이 불편하다는 것은 아닙니다. 나름의 장점이 많기에 인텔, ARM을 비롯한 오늘날의 많은 CPU가 이 방식을 채택하고 있습니다. 어쨌든 결과적으로 Big-Endian 은 실제 숫자값과 저장 순서가 동일하므로 눈으로 읽기에는 쉽습니다.

모드버스는 Byte 또는 Word(2Byte) 단위의 데이터만 전송할 수 있습니다. Byte 데이터는 하나의 저장소를 가지므로 저장순서가 필요없으므로 엔디안과 무관하지만, Word 의 경우

2Byte로 구성되므로 Endian에 따라 저장순서가 달라집니다. 더구나 4Byte의 정수 또는 실수는 아예 전송규약이 없습니다. 그래서 2개의 Word로 나누어서 전송합니다. 이 각각의 경우에 대하여 모드버스와 엔디안을 맞추어 주어야 합니다.(하지만 간단합니다)



왼쪽 그림처럼 LMO는 모드버스 마스터로 부터 Word 데이터를 전송받으면 2Byte를 뒤집어서 저장하고(Word Swap), 또 마스터에게 전송할 때도 뒤집어서 전송하므로 사용자가 별도의 작업을 하지 않아도 자동으로 엔디안이 일치합니다. 그러나 오른쪽 그림처럼 4Byte의 정수 또는 실수의 경우는 2Byte씩 나누어 전송하므로 결과적으로 0x56781234가 되어 영뚱한 값으로 되어 버립니다.



해결책은 마스터에서 4Byte 데이터를 보낼 때만 상위/하위 Word를 서로 바꾸어서 전송하면 됩니다(워드스왑이라 함). 이제 결과는 Little-Endian으로 읽으면 0x12345678이 되었습니다. 마찬가지로 이지컨트롤러의 4Byte 데이터를 마스터가 읽는 경우에도 읽은 2개의 Word를 서로 Swap하면 됩니다. Word-Swap은 작화프로그램에서 처리할 수 있으므로 해당 HMI(MMI) 메이커의 메뉴얼을 참고해 주십시오.

## 이지컨트롤러의 이미지 구조

이제 이지컨트롤러의 각 모듈의 이미지가 메모리에서 어떤 순서로 저장되어 있는지를 알면 위의 모드버스 평션으로 읽고 쓸 수 있습니다. 이미지의 구조는 `eventbus.h`에 구조체로 정의되어 있으므로 이를 참고하면 되지만, 알기 쉽게 그림으로 나타내 보도록 하겠습니다.

**주의** Word 단위의 주소로 나타내므로 1Word는 2Byte를 의미합니다.

### ■ AIM 이미지

\* 입력이미지

`vin0, vin1, vin2, vin3`은 각각 4바이트 float값(2Word)임

| 변수명       | vin[0]<br>(float) | vin[1]<br>(float) | vin[2]<br>(float) | vin[3]<br>(float) |
|-----------|-------------------|-------------------|-------------------|-------------------|
| Word Addr | 00 01             | 02 03             | 04 05             | 06 07             |

\* 출력이미지 : 없음

## ■ AOM 이미지

- \* 입력이미지 : 없음
  - \* 출력이미지

| 변수명       | vout[0]<br>(float) | vout[1]<br>(float) | iout[0]<br>(float) | iout[1]<br>(float) |
|-----------|--------------------|--------------------|--------------------|--------------------|
| Word Addr | 00 01              | 02 03              | 04 05              | 06 07              |

## ■ DIE 이미지

- ## \* 입력이미지

- \* 출력이미지 : 없음

## ■ DIM 이미지

- ## \* 입력이미지

- \* 출력이미지 : 없음

#### ■ DIO 이미지

- ## \* 입력이미지

- \* 출력이미지

## ■ DOE 이미지

- \* 입력이미지 : 없음
  - \* 출력이미지

|       |       |       |       |       |       |      |      |      |      |      |      |      |      |      |      |             |
|-------|-------|-------|-------|-------|-------|------|------|------|------|------|------|------|------|------|------|-------------|
| rly15 | rly14 | rly13 | rly12 | rly11 | rly10 | rly9 | rly8 | rly7 | rly6 | rly5 | rly4 | rly3 | rly2 | rly1 | rly0 | 로<br>사<br>포 |
|-------|-------|-------|-------|-------|-------|------|------|------|------|------|------|------|------|------|------|-------------|

## ■ DOM 이미지

- \* 입력이미지 : 없음
  - \* 출력이미지

## ■ ENV 이미지

- ## \* 입력이미지

| Word | Addr            | 00               | 01             | 02              | 03 | 04 |
|------|-----------------|------------------|----------------|-----------------|----|----|
| 수표   | lux<br>(uint32) | hand<br>(uint16) | uv<br>(uint16) | temp<br>(int16) |    |    |

- \* 출력이미지 : 없음

## ■ MOC 이미지

- ## \* 입력이미지

| loc<br>(int32) | jvol<br>(uint16) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
|----------------|------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|--|----|----|----|
| Bit Addr       | 0                | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  | 01 | 02 | 03 |
| Word Addr      | 00               |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| 문자<br>포<br>함수  |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| busy           |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| jspd           |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| jccw           |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| jcw            |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| nnp            |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| neg            |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |
| pos            |                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |  |    |    |    |

- \* 출력 이미지

## ■ SMD 이미지

- \* 입력이미지

|      |      |    |    |    |    |
|------|------|----|----|----|----|
| Word | Addr | 00 | 01 | 02 | 03 |
|------|------|----|----|----|----|

## \* 출력이미지

## ■ TCM 이미지

## \* 입력이미지

| 변수명       | tc[0]<br>(float) | tc[1]<br>(float) | tc[2]<br>(float) | tc[3]<br>(float) | tc[4]<br>(float) | tc[5]<br>(float) |
|-----------|------------------|------------------|------------------|------------------|------------------|------------------|
| Word Addr | 00 01            | 02 03            | 04 05            | 06 07            | 08 09            | 10 11            |

\* 출력이미지 : 없음

## ■ USD 이미지

### \* 입력이미지

### \* 출력이미지

#### ■ 지원되지 않는 모듈

\* SEG, MSB

: 버스에 연결해서 사용할 수는 있지만, 모드버스로 엑세스할 수는 없습니다.

\* USB, LAN, LMO

: 버스에 연결해서 사용할 수는 있지만, 모드버스로 액세스할 이미지가 없습니다.

### 사용 예 1: DIO 입출력

이번에는 <http://www.modbustools.com> 의 Modbus Poll 시뮬레이션 프로그램으로 동작을 테스트하는 방법을 설명합니다. 비록 10 분의 시간제한이 있지만, 간단한 동작테스트에 매우 유용합니다. 우선 DIO의 스위치입력을 읽고, 릴레이를 동작시켜 보겠습니다.

우선 DIO 모듈을 LMO에 등록하겠습니다.

```
int main()
{
    // create modules
    // **important** LMO must create modules connected to bus
    DIO* dio = CreateDIO(ID_DIO1);

    // start eventbus
    EventBus();

    // variable definition, starting code

    EventProgram;
    IF (this, ModbusFrameReceived) {
        ProcessModbusFrame(pe);
    }

    ScanProgram;
    Aux_ReceiveEvent();

    End;
}
```

위에서처럼 DIO를 등록하는 코드를 추가한 후 프로그램을 빌드하고 LMO에 다운로드합니다. 위에서 ID\_DIO1(여기서는 4)은 ezconfig.h에 정의된 DIO 모듈의 ID입니다. 당연히 DIO 모듈에도 ezconfig.exe를 사용해서 ID를 4로 설정해야 합니다.

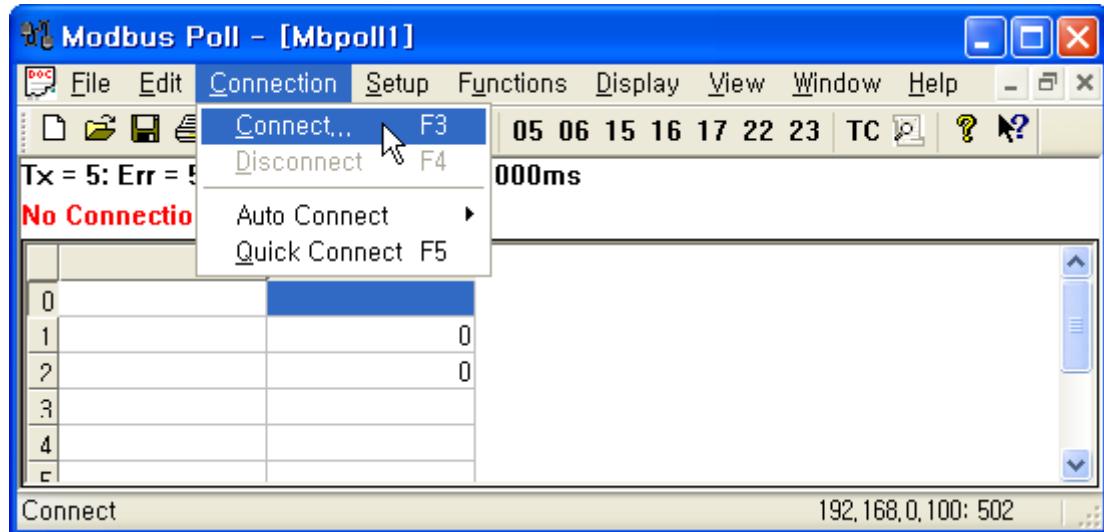
#### ■ DIO 입력이미지

| 변수명       | S0 | S1 | S2 | S3 | S4 | S5 |   |   |   |   |    |    |    |    |    |    |
|-----------|----|----|----|----|----|----|---|---|---|---|----|----|----|----|----|----|
| Bit Addr  | 0  | 1  | 2  | 3  | 4  | 5  | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Word Addr | 00 |    |    |    |    |    |   |   |   |   |    |    |    |    |    |    |

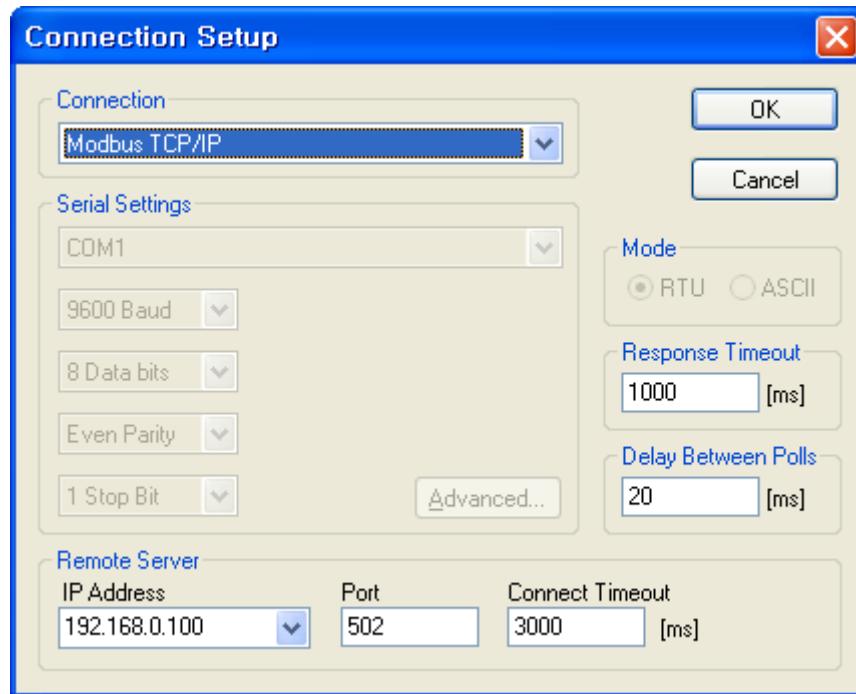
DIO에는 6개의 스위치입력과 2개의 릴레이 출력이 있습니다. 스위치입력은 두가지 방법으로 읽을 수 있습니다. FC2(Read Multi-Bit Switches) 또는 FC4(Read Multi-Word Switches).

위의 이미지를 보면 Bit 단위로 보면 시작주소는 0번이고 입력의 개수는 6개입니다. 비트 6~15는 읽거나 써도 오류는 발생하지 않지만 사용되지 않는 공간이므로 사용하지 마십시오. 만일 이 범위를 벗어난 비트(예, 비트 16)를 읽으면 LMO는 익셉션(오류) 프레임을 응답합니다.

그리고, Word 단위로 보면 시작주소는 00번이고 입력의 워드개수는 1개입니다. 만일 2개의 워드를 읽으려 시도한다면 LMO는 익셉션(오류) 프레임을 응답합니다.

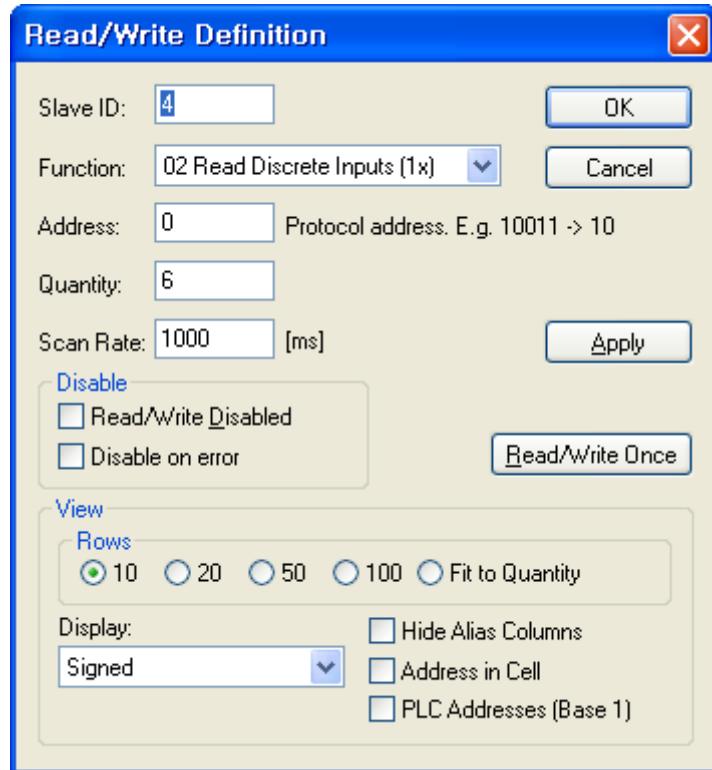


LMO에 랜선을 연결하고, 메뉴에서 Connect(F3)를 선택합니다. 등록화면이 나오면 [Register later]를 선택합니다.

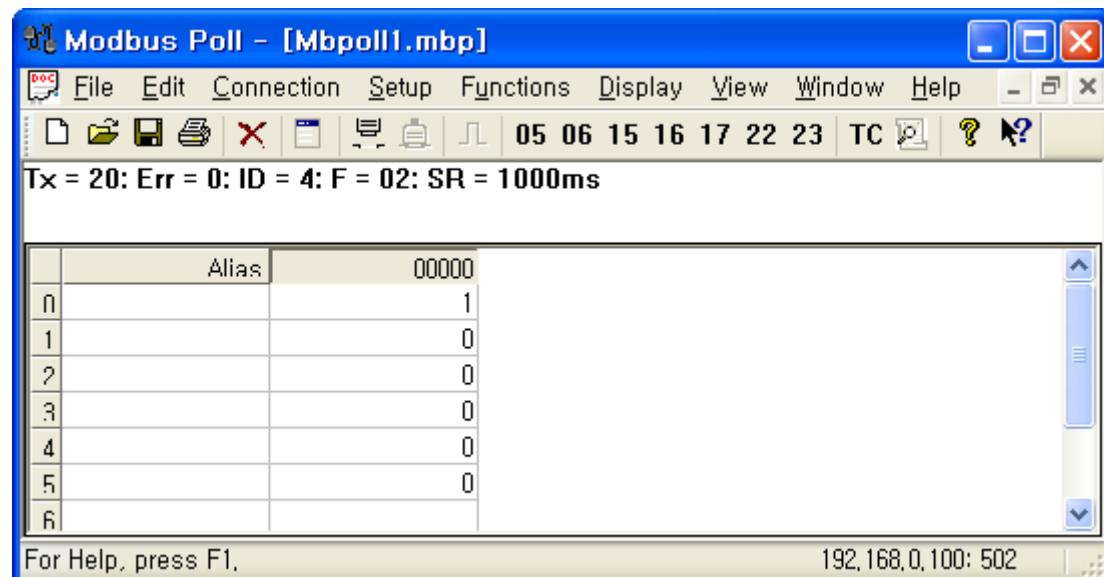


통신이 연결되면 툴바는 처럼 Quick Connect 아이콘은 비활성으로 되고 Disconnect 버튼이 활성화 됩니다. 툴바에서 Read/Write Definition 을 선택합니다.

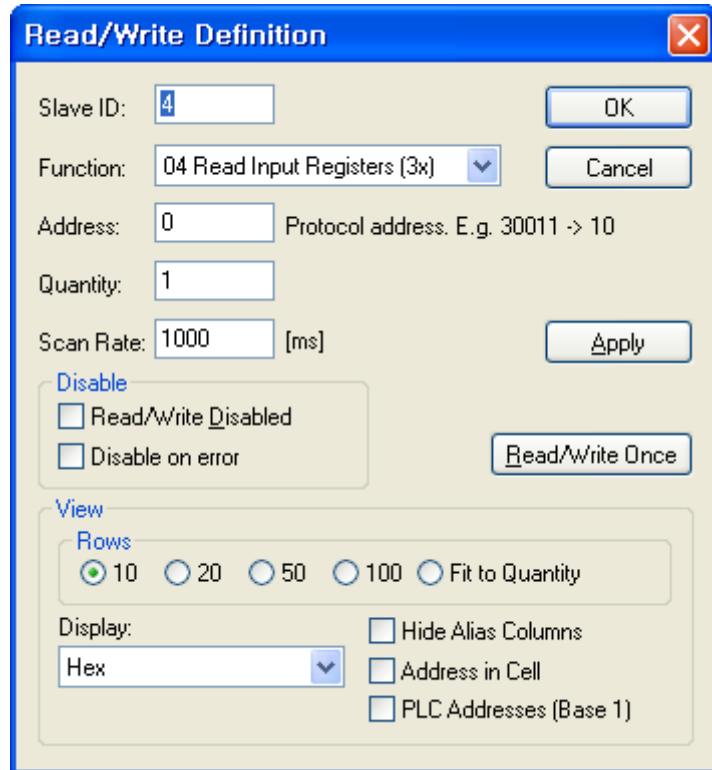
통신이 제대로 연결되면 “No Connection” 메시지가 사라집니다.



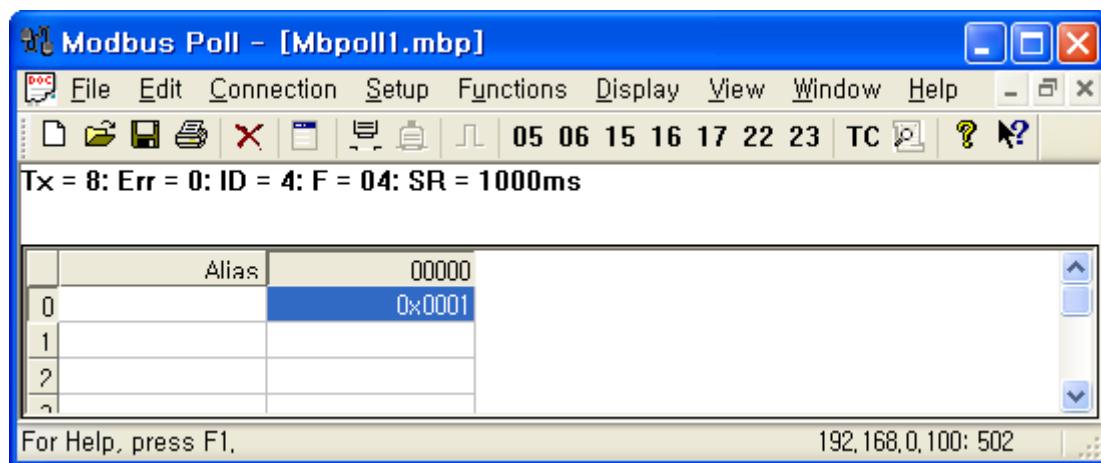
Slave ID 는 DIO 의 ID(4)를 입력하고, 평선클 2 를 선택해서 비트단위로 읽겠습니다.  
주소는 0 번비트에서부터 6 개비트를 읽도록 입력하고, 읽는 주기는 1000[ms]로 합니다.



스위치를 조작하면 화면처럼 ON 상태인 스위치는 1로 표시됩니다.



 툴바에서 **Read/Write Definition** 을 다시 선택하고, 이번에는 워드단위로 읽겠습니다. 평선클을 04 로 선택하고 시작주소는 0, 워드개수는 1 개로 선택합니다. 그리고 **Display** 메뉴에서 Hex 를 선택하여 16 진수로 보겠습니다.

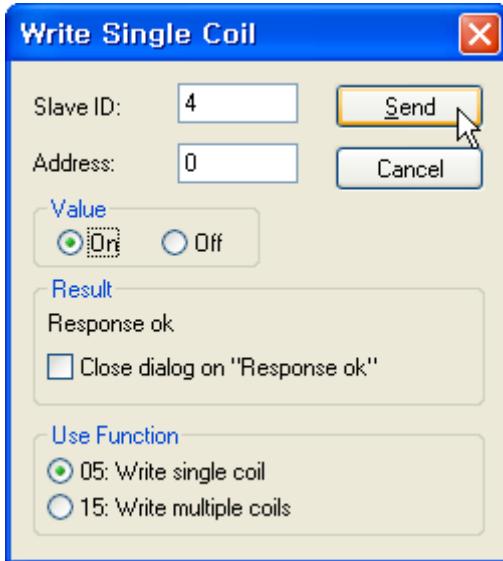


0 번스위치만 ON 되었으므로 그림처럼 0x0001 로 표시됩니다. 만일 6 개의 스위치가 모두 ON 이라면 0x003F 로 표시됩니다.

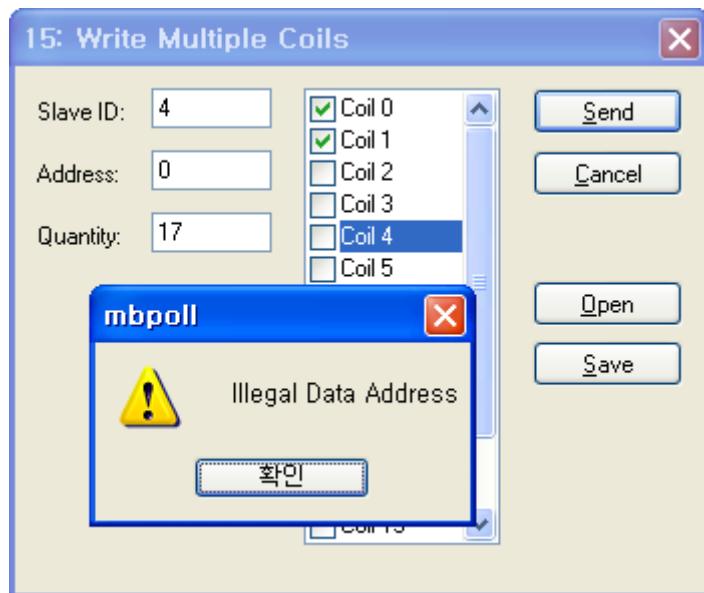
#### ■ DIO 출력이미지

|           |    |    |   |   |   |   |   |   |   |   |    |    |    |
|-----------|----|----|---|---|---|---|---|---|---|---|----|----|----|
| 변수명       | Y0 | Y1 |   |   |   |   |   |   |   |   |    |    |    |
| Bit Addr  | 0  | 1  | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Word Addr | 00 |    |   |   |   |   |   |   |   |   | 13 | 14 | 15 |

릴레이 출력코일은 FC01(Read Multi-Bit Coils), FC03(Read Multi-Word Coils)으로 현재 상태를 읽을 수 있습니다. 앞의 스위치 입력을 읽는 것과 동일한 방법으로 하면 됩니다. 릴레이 출력코일에 값을 쓰는 것은 FC5(Write Single-Bit Coil), FC15(Write Multi-Bit Coils), FC6(Write Single-Word Coil), FC16(Write Multi-Word Coils)을 사용할 수 있습니다.



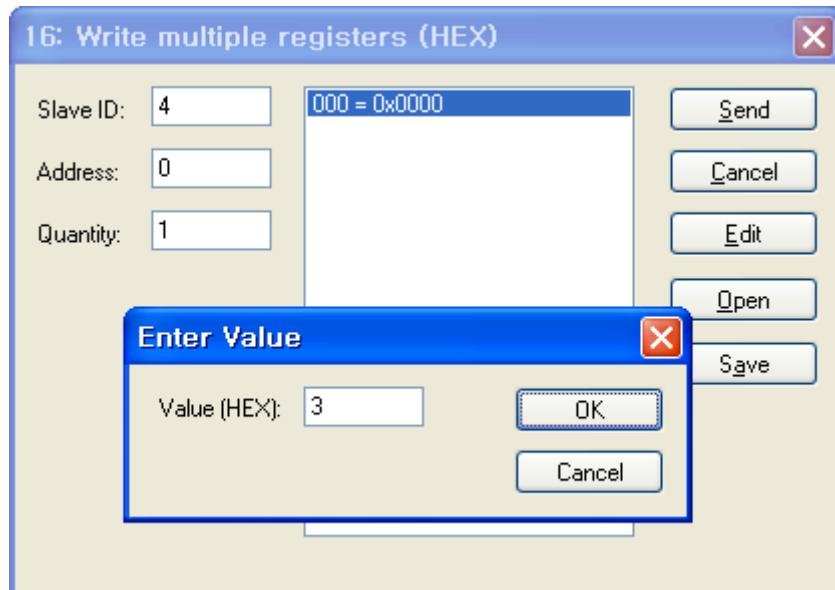
Functions 메뉴 또는 툴바에서 “05”를 선택합니다. ID 는 DIO 의 ID(4)를 주소는 rly0 을 온오프시키기 위해서는 0 을, rly1 을 온오프시키기 위해서는 1 을 입력합니다. Value 를 선택하고 Send 버튼을 클릭하면 해당하는 릴레이가 온오프됩니다.



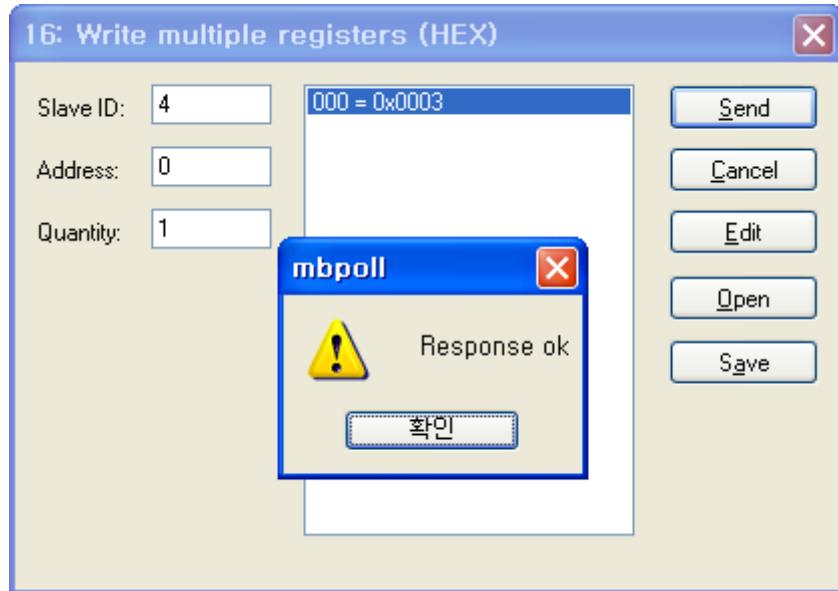
여러개의 릴레이(코일)를 동시에 온오프하려면 FC15 를 사용합니다. 그림처럼 릴레이개수를 17 로 하면 DIO 출력이미지의 범위(비록 릴레이는 2 개이지만, uint16 타입이므로 16 비트임에 주의)를 벗어나므로 익셉션(오류) 프레임이 전송되어 그림처럼 “Illegal Data Address” 메시지가 표시됩니다.



이번에는 워드단위로 값을 써서 릴레이를 동작시키기 위하여, Functions 메뉴 또는 툴바에서 “06”을 선택합니다. ID 는 DIO 의 ID(4)를 워드주소는 0 을, 그리고 Value 에는 relay0 과 relay1 을 모두 온시키기 위하여 3 을 입력합니다. Send 버튼을 클릭하면 릴레이가 온오프됩니다.



FC16 은 여려개의 워드데이터를 동시에 쓸 때 사용합니다. 표시되는 형식은 “주소 = 워드값”입니다. 이 항목을 더블클릭하면 Enter Value 창이 나타납니다. 두 릴레이를 모두 온시키기 위하여 3 을 입력합니다.



프레임이 제대로 LMO에게 전송되면 릴레이가 온되고, 응답프레임이 돌아옵니다. Modus Poll 프로그램은 응답프레임을 받으면 “Response ok” 메시지창을 보여줍니다.

## 사용예 2: USD 모터제어

USD의 출력이미지에 값을 써서 모터를 회전하는 예제를 살펴봅니다.

\* 출력이미지

| 변수명       | cmd                                   | step | dir |  |  |  |  |  |  |  |  |  |  |                   | move<br>(uint32) |
|-----------|---------------------------------------|------|-----|--|--|--|--|--|--|--|--|--|--|-------------------|------------------|
| Bit Addr  | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 |      |     |  |  |  |  |  |  |  |  |  |  | pps<br>(uint32)   |                  |
| Word Addr | 00                                    |      |     |  |  |  |  |  |  |  |  |  |  | accel<br>(uint16) |                  |

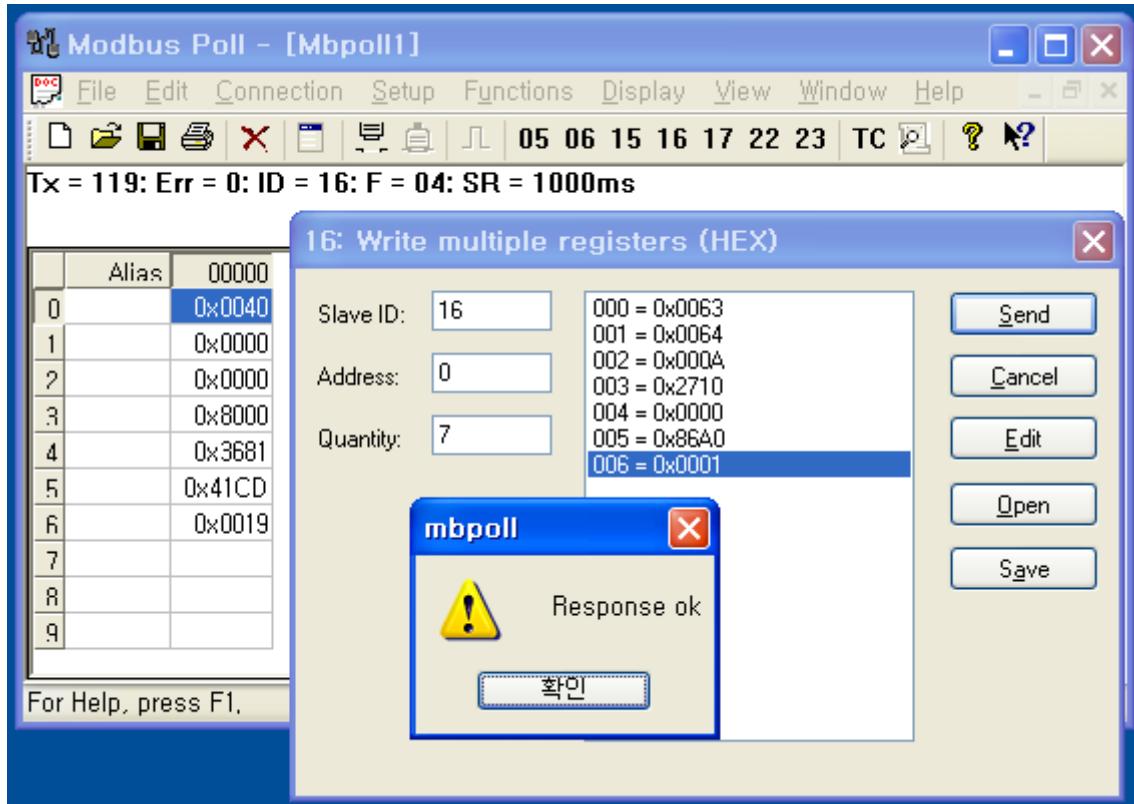
각 변수에 쓸 적절한 값을 살펴보면,

eventbus.h의 매크로 선언을 참조하면,

```
typedef enum {FREE, HOLD, STOP, RUN, MOVE} SMD_CMD, MOC_CMD, USD_CMD;
typedef enum {CW, CCW} SMD_DIR, MOC_DIR, USD_DIR;
typedef enum {USD_FULL=1, USD_HALF=3, USD_QUARTER=4, USD_EIGHTH=5,
USD_SIXTEENTH=6} USD_STEP;
```

위의 출력이미지 UsdOutEvent에서 config는 하위 8비트만 사용하는데, cmd=3(RUN), step=6(USD\_SIXTEENTH), dir=0(CW) 이므로 word00은 이진법으로 표시하면 “000000001100011”(0x0063)이 됩니다. ipps=100(0x0064), accel=10(0x000A), pps=10,000(0x00002710), move=100,000(0x000186A0)이 됩니다.

USD의 출력이미지에 쓸 값은 시작주소=00, 워드수=7, 워드값={0063, 0064, 000A, 2710, 0000, 86A0, 0001} 가 됩니다. 여기서 주의할 점은 ipps, move는 2워드 길이이므로 워드스왑해야 합니다.



툴바에서 Read/Write Definition 을 선택하고 ID=16, 평선=04, Address=0, Quantity=7 을 선택하면 위 그림의 배경처럼 USD의 입력이미지가 표시됩니다.

이어서 Functions 메뉴 또는 툴바에서 평선코드 16 을 선택하고 그림처럼 워드값을 입력한 후 Send 버튼을 클릭합니다. 모터가 회전하기 시작하고, 배경의 입력이미지 값이 변하는 것을 볼 수 있습니다.

#### \* USD 입력이미지

| 변수명       | pos | neg | nnp | jcw | jccw | jspd | busy | fan | sw0 | sw1 | jvol | loc | vm | heat |    |    |    |    |    |    |    |
|-----------|-----|-----|-----|-----|------|------|------|-----|-----|-----|------|-----|----|------|----|----|----|----|----|----|----|
| Bit Addr  | 0   | 1   | 2   | 3   | 4    | 5    | 6    | 7   | 8   | 9   | 10   | 11  | 12 | 13   | 14 | 15 |    |    |    |    |    |
| Word Addr | 00  |     |     |     |      |      |      |     |     |     |      |     |    |      |    | 01 | 02 | 03 | 04 | 05 | 06 |

배경의 워드 0=0x0040 은 busy=1 을 의미하므로 현재 모터가 회전중임을 알 수 있습니다. 워드 1=jvol=0x0000 인데, 이것은 현재 ezconfig.exe 에서 조그세팅이 스위치타입으로 선택되었으므로 볼륨을 읽지 않기 때문에 조그볼륨값은 0 으로 표시됩니다. 워드 2~3=0x80000000 (워드스왑해서 읽어야 함)은 NEG 리미트스위치에 도달하지 않아서 아직 초기화 되지 않았으므로 비초기화를 나타내는 음의 최대값을 나타냅니다(타입이 int32 임에 유의). 워드 4~5=0x41CD3681 은 25.6516 의 float 타입 표현으로 모터전원(VM)이 약 25V 임을 나타냅니다. 마지막 워드 6=0x0019 는 모터드라이버의 칩온도가 19 도임을 나타냅니다.

#### \*\* 사용상 주의할 점

LMO 는 출력이미지에 값을 쓰는 프레임을 받으면 내부적으로 해당 모듈에게 XxxOut 이벤트를 보냅니다. 즉, 해당 모듈의 출력이 갱신됩니다.

예를들어, 위의 예에서처럼 USD 의 출력이미지에 값을 쓰는 프레임을 보내면 LMO 는 UsdOut 이벤트를 USD 에게 보내어 출력동작이 실행되도록 합니다. 이것은 출력이미지 전체를 쓸 때 뿐만아니라, 단 한개의 워드만 쓸 때도 동일하므로 주의해서 사용해야 합니다.

예를 들면,

- 모터회전을 시작한다.

시작주소=00, 워드수=7, 워드값={0063, 0064, 000A, 2710, 0000, 86A0, 0001}

- 모터회전을 정지시킨다.

시작주소=00, 워드수=1, 워드값=0060 (나머지 설정값은 이전값이 사용된다)

- 다시 회전시킨다.

시작주소=00, 워드수=1, 워드값=0063

- 회전속도를 20KPPS 로 바꾼다

시작주소=03, 워드수=2, 워드값={4E20, 0000}

그러므로, USD 의 경우, 출력이미지의 각 파라미터를 모두 쓴 후 마지막에 워드 0 의 cmd 를 3(RUN)으로 세팅해서 모터의 회전을 개시하는 방법도 사용할 수 있습니다.

## 6.20. UPS, Uninterrupted Power Supply Module



UPS 무정전 전원모듈은 불안정한 전원환경에서 배터리에 의하여 연결된 모듈에 일정시간 전원공급을 할 수 있으므로 안정된 동작을 보장합니다. 뿐만아니라 충전회로와 방전 최저전압 설정회로가 내장되어 있어서 배터리의 과충전과방전을 방지할 수 있으므로 배터리를 장시간 안전하게 사용할 수 있습니다(출고시 12V 연축전지에 맞게 만충전압 13.3[V], 방전종지전압 10.8[V]로 설정되어 있음). 사용할 수 있는 배터리는 12V 연축전지, 3~55(9~21V) 리튬/폴리머 이차전지입니다.

또한 유연한 연결방법을 제공하여 결선방법에 따라서 다양한 방법으로 응용할 수 있습니다.

- 무정전 전원기능

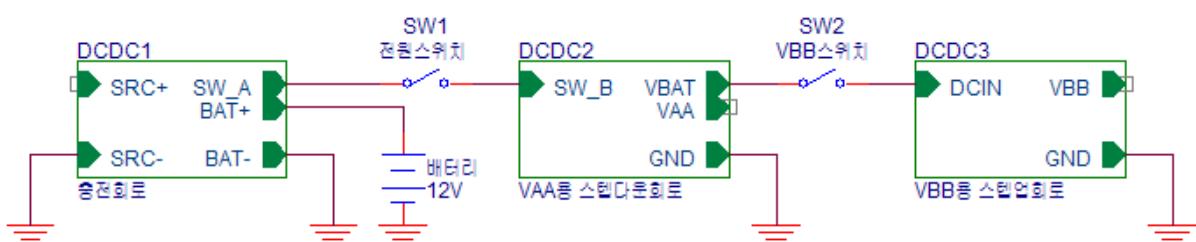
어댑터와 배터리를 연결하면 정전시에도 안정된 전원을 공급한다.

- 이동형 설비제작

배터리 연결만으로 모듈에 전원을 공급하여 이동중에도 사용할 수 있고, 차량에도 설치할 수 있다.

- 태양광, 풍력 사용

태양전지와 같은 에너지 수집장치와 배터리를 사용해서 모듈을 동작할 수 있다.



그림은 UPS 모듈의 내부회로 구성도입니다. 녹색화살표는 IO 커넥터의 핀입니다. SRC+/-에 PSM 모듈의 VBB 출력이나 DC 전원 또는 태양전지 등이 연결되면 DCDC1(스텝다운 충전회로)에서 BAT+/-에 연결된 배터리를 충전합니다. 충전전압은 모듈의 왼쪽에 있는 포텐ショ미터(VHB)로 설정할 수 있습니다.

내부적으로 SW\_A 와 BAT+는 연결되어 있어서 SW\_A/SW\_B 에 전원스위치 SW1 를 연결하면 DCDC2 에 전원을 공급/차단할 수 있습니다. DCDC2 는 입력단에 과방전(저전압) 방지회로가 있어서 배터리의 과방전을 방지합니다. VBAT 단자로는 배터리 전원이 출력되고, 스텝다운회로에서 VAA(5V)를 생성해서 VAA 핀으로 출력합니다. VBB 출력이 필요치 않은 경우에는 SW2 를 OPEN 시키면 되고, 상시 출력하려면 쇼트시키면 되고, SW2 를 연결하여 온오프 할 수도 있습니다. 과방전 방지회로를 거쳐 VBAT 로 출력된 전원이 DCIN 으로 입력되면 DCDC3(스텝업회로)에서 24V 의 VBB 전압으로 승압시켜 출력합니다.

충전회로에서 배터리에 충전하는 충전전류를 조정하기 위하여는 UPS 모듈의 왼쪽상단의 R1~R4를 이용합니다. 저항을 한개씩 제거할 때마다 충전전류는 약 25% 감소합니다.

**\*주의\*** SRC+/-에 공급되는 충전용전원은 배터리의 최대전압(만충전압)보다 4V 이상 높은 전원을 사용해야 합니다. 연축전지의 경우 13.32[V]에 설정되어 있으므로, 18[V]이상의 전원을 사용하여야 합니다.

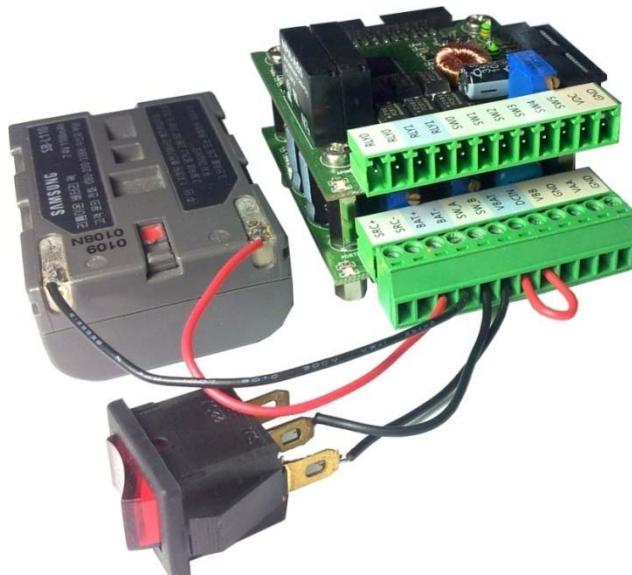
**\*주의\*** 만충전 전압은 왼쪽 포텐시미터(VH)로 설정하고, 과방전 방지용 최저전압은 오른쪽 포텐시미터(VL)로 설정합니다.

**\*주의\*** 두개의 버스커넥터(BUS20, BUS10)가 있습니다. BUS10에는 SEG, ENV와 같은 10핀 버스커넥터를 사용하는 모듈(SEG, ENV와 같은)을 연결할 수 있습니다. (확장커넥터가 아닙니다)

### 6.20.1. 만충전압, 방전종지전압 설정방법

연축전지의 경우 만충전압 13.32[V], 방전종지전압 10.8[V]가 적절합니다. UPS를 이 전압에 맞도록 설정하는 방법을 살펴봅니다.

- 설정작업 전 준비사항
  - VBAT, DCIN을 전선으로 쇼트시킨다.
  - SW\_A, SW\_B에 토클스위치를 연결하거나, 전선으로 쇼트시킨다.
  - 약간의 부하전류를 흘리기 위하여 VAA, GND 사이에 100옴정도의 저항을 연결한다.
  - 입력(SRC+/-)에 13.32V보다 4V 이상 높은 전원을 연결한다.(극성주의)
- 방전종지전압 설정
  - VH를 돌려 SW\_B의 전압을 10.8[V]에 맞춘다.
  - VL을 CCW로 돌려 LED를 ON 시킨후, 다시 서서히 CW로 돌려서 LED를 끈다.
- 만충전압 설정
  - VH를 CW로 돌려 SW\_B의 전압을 13.32[V]에 맞춘다.



## IO 커넥터 핀사양

| 핀  | 명칭   | 기능               | 기타                               |
|----|------|------------------|----------------------------------|
| 1  | SRC+ | 충전전원 (+) 입력      | 최대 35V 를 초과하지 말것 ( <b>극성주의</b> ) |
| 2  | SRC- | 충전전원 (-) 입력      | 내부에서 GND 와 연결됨                   |
| 3  | BAT+ | 배터리 (+) 입력       | 8~24V 범위의 배터리를 사용할것              |
| 4  | BAT- | 배터리 (-) 입력       | 내부에서 GND 와 연결됨                   |
| 5  | SW_A | 전원스위치 연결         | 내부에서 BAT+와 연결됨                   |
| 6  | SW_B | 전원스위치 연결         |                                  |
| 7  | VBAT | 배터리 출력           | 저전압 차단회로를 거쳐 나오는 출력              |
| 8  | DCIN | VBB 용 DC/DC 입력단자 | 통상 VBAT 에 연결함                    |
| 9  | VBB  | 24V IO 용 전원출력    |                                  |
| 10 | GND  |                  |                                  |
| 11 | VAA  | 5V 로직용 전원출력      |                                  |
| 12 | GND  |                  |                                  |

## 전기적 특성

|                             |                                                                    |                                  |
|-----------------------------|--------------------------------------------------------------------|----------------------------------|
| 출력전류<br>VAA(5V)<br>VBB(24V) | 12V 배터리, SRC=24V                                                   | 0.5[A] 이상<br>0.25[A] 이상          |
| 소모전류<br>SRC+<br>BAT+        | 무부하, SRC=24V, 12V 배터리<br>VAA, VBB 사용시<br>VAA, VBB 사용시<br>VAA 만 사용시 | 8[mA] 이하<br>7[mA] 이하<br>4[mA] 이하 |
| 충전전류                        | SRC=24V, 12V 배터리                                                   |                                  |

## 6.21. PSM, Power Supply Module



이벤트버스를 통하여 연결되는 모든 이지컨트롤러 모듈들에게 전원을 공급하는 가장 기본적인 모듈입니다. 220VAC 를 입력으로 공급받아서 5.0V 의 VAA 전원과 24V 의 VBB 전원을 생성하고 이를 이벤트버스를 통해서 연결된 모듈에 공급합니다.

추가적인 기능으로 VAA, VBB 전원을 외부에서 사용할 수 있도록 별도의 2 핀 커넥터를 제공하고 있습니다. 그리고 리셋 푸쉬버튼을 제공하므로

이벤트버스에 연결된 전체 모듈을 동시에 리셋시킬 때 사용할 수 있습니다.

이벤트버스의 전원 시스템은 **병렬동작**이 가능하므로 여러 대의 PSM 을 동시에 사용하면 더 큰 전류를 사용할 수 있습니다.

## 6.22. ColinkEx, SWD JTAG Module



SWD(Serial Wired Debugger) 입니다. 펌웨어 프로그램을 개발하면 각 모듈에 프로그램을 다운로드해야 합니다. 이때 사용되는 것이 ColinkEx 입니다. 프로그램을 다운로드 하는 이외에도 프로그램이 비정상적으로 동작할 때 이를 디버깅하기 위한 디버거로서의 역할도 합니다. 즉, 기존의 JTAG 이 시리얼로 만들어져 있어서 연결되는 선의 가닥수가 적다고 생각하면 됩니다.

ColinkEx 는 모듈의 확장커넥터에 연결해서 사용할 수 있도록 만들어져 있어서 24 핀 핀헤더 커넥터를 그대로 사용합니다. 또한 여기에는 추가적으로 UART 에 연결하여 PC 에 연결할 수 있도록 CP2102 칩을 이용하여 USB-UART 브릿지가 추가되어 있습니다. 그러므로 별도의 장비가 없어도 PC 의 하이퍼터미널에 접속하여 디버깅이 가능합니다. (CP2102 드라이버 설치 필요, 개발자료 게시판 참고)

결국, 컴퓨터의 USB 포트에 두개의 커넥터로 연결되고, 24 핀 확장커넥터 또는 5 핀 몰렉스를 타겟모듈에 연결하면 프로그램 다운로드, JTAG 디버깅, 가상 COM 포트(USB), 하이퍼터미널을 모두 동시에 사용할 수 있게 됩니다.

흰색의 몰렉스 53015-5 핀 커넥터는 50X55MM 의 표준 모듈이 아닌 소형모듈에서 사용할 수 있도록 만들어진 디버거 커넥터입니다. ENV, SEG 모듈은 이 커넥터를 사용합니다.

나머지 두개의 커넥터는 편의를 위해 표준 JTAG 과 타사의 JTAG 에 맞추어져 있습니다. 핀 규격을 확인하여 사용하시기 바랍니다. (이부분은 별도의 기술지원은 하지 않으며, 향후 삭제 예정입니다.)

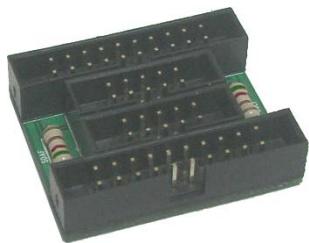
## 6.23. EXP, Expansion Module



확장커넥터에 연결해서 사용자의 회로를 추가할 수 있도록 지원하는 모듈입니다. 이 모듈을 사용하면 사용자는 다른 모듈의 확장커넥터에 연결하여 그 모듈의 마이컴에서 제공하는 여유 IO 핀에 연결되는 사용자의 회로를 구성할 수 있습니다. 이렇게 연결된 IO는 사용자이벤트(User Defined Event) 기능을 이용하여 다른 모듈이나 컴퓨터와 이벤트를 공유하거나 메시지루프에서 이벤트핸들러를 작성하여 처리할 수 있습니다.

전면에는 12 핀의 IO 커넥터도 제공되므로 외부에 센서를 연결하거나 출력을 연결할 수 있으므로 학생들의 전자회로 및 임베디드 실습용으로 최적의 환경을 제공합니다.

## 6.24. BPU, Bus Pull-Up Module



이벤트버스는 모듈간의 통신으로 I2C 통신을 사용합니다. I2C 통신은 멀티마스터, 멀티드롭 통신을 지원하므로 연결된 모듈들이 이벤트가 발생할 때마다 언제든 이벤트를 연결된 모듈에 전송할 수 있으므로 이벤트버스에 매우 적합한 통신입니다. 또한 외부에 별도의 트랜시버 칩이 필요치 않아 가격적으로도 매우 장점이 클 뿐만 아니라, 장거리 통신은 어렵지만 어느 정도의 근거리 통신은 가능하므로 실제 시스템을 구성해 보면 매우 큰 장점임을 알 수 있습니다.

하지만 I2C 통신은 오픈 콜렉터 방식을 사용하므로 외부에 풀업저항이 필요합니다. 기본적으로 모든 모듈은 내부에  $10\text{ k}\Omega$  저항이 풀업되어 있으므로 별도로 외부에 BPU 모듈을 사용하지 않아도 됩니다. 하지만 좀더 **안정되고 고속으로** 데이터를 송수신하여 이벤트버스의 성능을 올리고, **통신거리를 확장하기** 위하여는 외부에 추가적인 풀업저항이 필요합니다.

추가적으로 BPU에는 2 개의 20 핀커넥터와 2 개의 10 핀 커넥터가 있으므로 20 핀버스와 10 핀버스를 상호연결하거나, 짧은 케이블을 연결 할 때도 사용할 수 있습니다.

**\*주의\*** BPU 에 내장된 풀업저항은 매우 신중하게 결정하여야 합니다. 버스의 최대 풀업전류는  $20\text{ mA}$  입니다. 이를 초과하면 버스에 연결된 모든 모듈에 치명적 손상을 가져올 수 있습니다. 다음의 계산법을 반드시 숙지 후 사용하시기 바랍니다.

- 통신을 사용하는 모듈은 각각  $0.33\text{ mA}$  의 풀업전류를 험린다.  
BPU 저항으로 흐르는 전류와, 모듈의 풀업전류의 합이  $20\text{ mA}$  를 넘지 않아야 한다.
- (예 1) BPU 없이 사용시, 버스에 연결할 수 있는 모듈의 최대 개수는 몇개인가.  
 $\text{최대개수} = \text{최대전류} / \text{모듈 개당 전류} = 20\text{ mA} / 0.33\text{ mA} = 60 \text{ 개}$

단, 전류규격상 60 개이지만, 실제로 통신이 가능한 개수는 아님.  
즉, 구성에 따라 연결할 수 있는 모듈개수는 제한될 수 있다.

- (예 2) BPU 에는 기본적으로 330 옴 저항이 VAA(5V)에 연결되어 있다. BPU 를 연결할 때 모듈은 몇개까지 연결할 수 있는가?  
BPU 의 풀업전류 =  $5V / 330\Omega = 15.2mA$  (옴의 법칙, 전류 = 전압 / 저항)  
그러므로 ( $20mA - 15.2mA$ )의 전류가 여유분이 되므로,  
최대개수 =  $(20mA - 15.2mA) / 0.33mA = 14$  개
- (예 3) 20 개의 모듈을 연결하려면 BPU 의 저항을 얼마로 변경해야 하는가?  
모듈이 흘리는 전류의 합은  $20 \text{ 개} * 0.33mA = 6.6mA$  이므로,  
옴의 법칙에서, BPU 저항 =  $5V / (20mA - 6.6mA) = 373$  옴  
그러므로 BPU 는 이보다 큰 390 옴 이상의 저항을 사용하는 것이 최적이 됩니다.

## 6.25. UUB, USB-UART Bridge (단종예정)



(본 제품은 향후 단종예정이므로 UUM 사용을 권장합니다) USB-MINI 커넥터에 연결하여 확장커넥터의 UART 에 접속하는 모듈입니다. SILAB 사의 CP2102 브리지 칩을 사용하여 매우 소형으로 설계되었습니다. 4 핀의 2MM 피치 몰렉스 커넥터를 사용합니다. 핀배열은 VBUS, TX, RX, GND 이므로 타사의 UART 에 연결할 때도 사용할 수 있습니다.

모듈의 확장커넥터에 바로 연결할 수 있도록 보조커넥터와 케이블을 함께 제공합니다.

- 컴퓨터와 모듈을 연결할 때  
: PC --- UUB --- 24 핀 (ENV,SEG 의 경우 5 핀) 확장커넥터

## 6.26. UUM, UART-USB Bridge Module



UUB 의 단점을 보완한 제품으로 컴퓨터의 USB 에 연결하여 이지컨트롤러모듈을 제어할 수 있습니다. SILAB 사의 CP2102 칩을 사용하므로 컴퓨터에 USB 드라이버 프로그램(카페참조)을 설치후 사용하십시오. 또한, 스텝업 DCDC 컨버터를 내장하여 USB 의 5V 전원에서 VBB 전원(24V)을 생성하고 이벤트버스에 공급하므로 그리 큰 전류를 필요로하지 않는 경우라면 PSM 과 같은 별도의 전원이 없어도 됩니다. 물론, 이벤트버스의 전원 시스템은

**병렬동작**이 가능하도록 설계되었으므로 PSM 을 동시에 사용하면 더 큰 전류를 사용할 수 있습니다. VAA(5V), VBB(24V) 전원을 외부에서 사용할 수 있도록 두개의 2 핀 IO 커넥터가 있고, SEG, ENV 에 연결할 수 있도록 5 핀의 몰렉스 커넥터도 준비되어 있습니다.

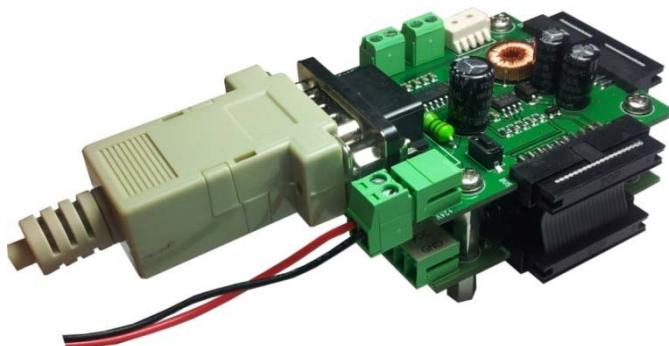


DIN 레일을 사용할 경우 UUB 는 기구적 간섭을 일으키지만, UUM 은 간섭이 없어 장착이 용이합니다.

### 6.27. U2M, UART-RS232 Bridge Module



임베디드 컴퓨터처럼 호스트기능의 USB 통신을 사용할 수 없거나, 또는 타사의 컨트롤러와 연결하려는 경우 RS232 를 사용하는 것이 유용합니다. U2M 은 최대 115,200bps 의 통신속도를 지원합니다. 또한, 스텝다운 DCDC 컨버터를 내장하고 있어서 전면에 있는 2 핀 터미널블럭에 아래 그림처럼 외부에서 24V 를 입력하면 내부의 DC/DC 에서 VAA(5V)를 생성하여 이벤트버스에 공급하며, 생성된 5V 와 24V 를 좌측의 4 핀 터미널블럭으로 출력하여 사용자용으로 사용할 수 있도록 했습니다. 또한 SEG, ENV 의 5 핀 확장커넥터에 연결할 수 있도록 몰렉스커넥터도 제공합니다.



\*주의\* U2M 은 PC, 임베디드 보드등과 연결하여 이벤트를 송수신하여 모듈을 제어하기 위한 용도입니다. 사용자용 데이터를 RS232 통신으로 송수신하는 용도로 사용할 수 없습니다. 이 경우 MSB 를 사용하십시오.

### 6.28. DIN 레일용 부품



산업용 표준 딘레일과 장착용 레일 및 PCB 를 제공하여 산업현장에 설치가 용이합니다. 두세개의 피스로 딘레일을 벽에 고정하면 장착을 위한 모든 준비가

완료되므로 빠르게 현장에 설치할 수 있습니다.

### 6.29. PWR 전원연결 모듈

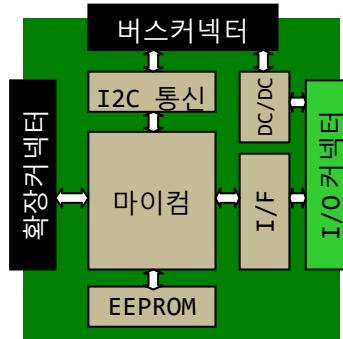
시스템에 전원을 연결할 때 배선을 쉽게 할 수 있도록 도와 줍니다. 이지컨트롤러의 버스케이블에는 VAA(5V, 필수), VBB(24V, 모듈에 따라서)가 사용됩니다. 만일 5V and/or 24V 전원이 있다면 별도의 전원을 사용하지 않고 PWR을 사용하면 아래 그림처럼 쉽게 버스에 전원을 연결할 수 있습니다.



## 7. 모듈의 커넥터 사양

이지컨트롤러 모듈은 통일된 커넥터 구성을 사용하고 있습니다. 모듈간의 원활한 연결을 위하여 사용되는 커넥터의 사양을 정확히 파악한 후 사용해 주십시오.

### 7.1. 커넥터 사양



우측의 버스커넥터는 **20 핀 박스헤더(2MM 피치)**이며, 후면의 확장커넥터는 **24 핀 박스헤더(2MM 피치)**입니다. 주의 할 점은 핀의 피치가 2MM 이므로 플랫케이블도 2MM 피치용(1MM 간격)을 사용하고, 연결되는 IDC 커넥터도 2MM 용을 사용하여야 합니다. 이것은 모듈을 컴팩트하게 만들고, 배선이 차지하는 면적을 줄이기 위함입니다.

또한, 각 모듈은 전면에 외부장치와 연결할 수 있는 IO 커넥터(DINKLE 터미널블럭)를 가지고 있습니다. 이 커넥터를 통하여 외부의 센서, 부하 등이 연결되므로 정확한 핀의 사양을 알 필요가 있습니다.

| 모듈                                                                                       | 전면 커넥터                                 | 메이팅 플러그                            |
|------------------------------------------------------------------------------------------|----------------------------------------|------------------------------------|
| AIH, AIM, AOM, DIE, DIO, DOE, DOM, DIM, SMD, MOC, TCM, UPS, EXP, MSB, MOC, USD, TCM, UPS | ECH350R-12                             | EC350V, EC350R                     |
| PSM                                                                                      | ECH350R-3, ECH350R-2                   | EC350V, EC350R                     |
| USB                                                                                      | 5 핀 미니 USB                             |                                    |
| LAN, LMO                                                                                 | RJ45                                   |                                    |
| ColinkEx                                                                                 | 5 핀 미니 USB                             |                                    |
| BPU                                                                                      | 박스헤더 10 핀(2MM 피치)<br>박스헤더 20 핀(2MM 피치) | IDC1016-10(2MM)<br>IDC1016-20(2MM) |

가장 많이 사용되는 커넥터는 DINKLE 사의 터미널블럭으로 외부 배선용으로는 아래 그림과 같은 V 타입이나 R 타입을 사용하면 됩니다.

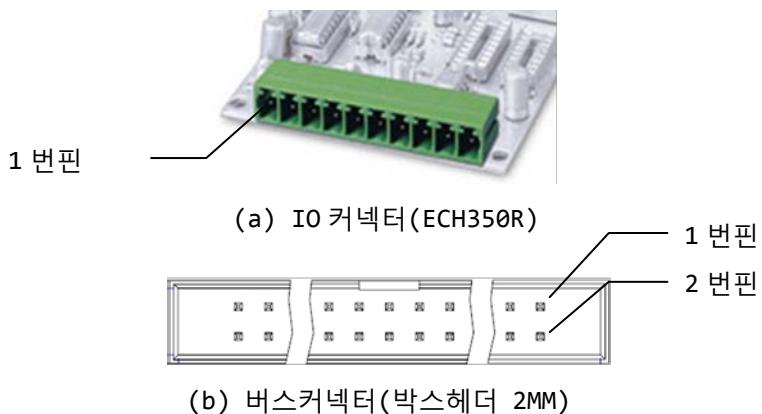


왼쪽의 V 타입은 모듈의 아랫면에 있는 표시용 LED를 가리게 됩니다. 또한 모듈을 보드서포터로 스택하면 높이가 조금 높아지는 단점이 있지만 선이 앞으로 나오므로 배선작업이 편리합니다. 반면, 오른쪽의 R 타입은 LED도 잘 보이고 스택시에 높이도 낮아집니다만 배선이 위에서 삽입되므로 조금 불편할 수도 있습니다. 적절하게 선택하십시오. (V 타입 권장)

또한 메이팅커넥터는 반드시 한 개를 사용할 필요는 없습니다. 예를 들어, AIM의 경우 12 핀인데, 사용자는 VDC 용 2 핀, 채널 1 용으로 2 핀 등과 같이 여러개의 작은 핀수를 가지는 커넥터로 나누어 사용하는 것도 가능합니다. 디버깅을 위하여는 이렇게 나누어 사용하는 방법이 편리합니다.

## 7.2. 커넥터의 핀번호

각 모듈의 앞면에 있는 IO 커넥터는 왼쪽이 1 번이고, 우측의 버스커넥터와 후면의 확장커넥터는 우측상단이 1 번핀입니다(커넥터 표면의 화살표 확인).



## 7.3. 버스커넥터

버스커넥터는 버스케이블을 사용해서 각 모듈을 이벤트버스로 연결하는 커넥터입니다. 여러 개의 이지컨트롤러 모듈을 연결하여 기능을 확장할 수 있도록 2MM 핀피치의 박스헤더 커넥터를 사용합니다. 버스케이블로 각 모듈을 연결하면 I2C 통신 뿐만 아니라 동작전원도 공급할 수 있도록 VAA, VBB 전원공급 라인도 포함하고 있습니다. 시스템 리셋 신호선도 있어서 GND와 스위치로 연결하면 연결된 모든 모듈의 MICOM을 동시에 리셋할 수 있습니다.

SDA, SCL 은 I2C 통신을 위한 선으로 풀업신호입니다. 각각의 모듈에는 10K 옴의 저항이 내장되어 있어서 0.33[mA]의 풀업전류를 흘리고 있습니다. 주의할 점은 SDA,SCL 에 연결된 저항을 통하여 최대 20[mA]를 넘는 전류가 흐르도록 풀업저항을 연결해서는 않습니다. **이경우 사용자 실수로 간주되며 A/S 의 대상이 아니므로 주의해 주십시오.**

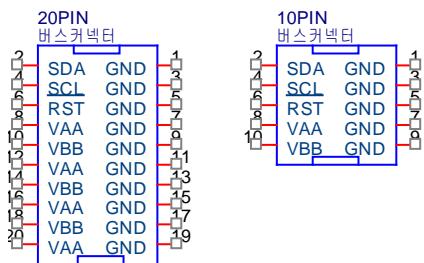
그리고, 이지컨트롤러는 HOT-SWAP 을 지원하지 않습니다. **이것은 전원이 공급된 상태에서 버스케이블을 탈부착해서는 않된다는 의미입니다.** 모듈이 파손될 수 있습니다.

반면, 확장커넥터에 ColinkEx 디버거를 연결하거나 뺄때는 모듈에 전원이 켜져 있어도 문제를 일으키지 않습니다. 이것은 매우 편리합니다. 예를들어, DIO,AIM 두개의 모듈이 연결되어 있을 때 전원이 켜진 상태에서 ColinkEx 를 DIO 모듈에 꽂아서 다운로드후, 다시 AIM 모듈에 꽂아서 다운로드할 수 있으므로 전원을 켜고 끄지 않아도 되므로 매우 편리합니다.

아래 표에서 리셋단자는 외부에 스위치를 그라운드에 연결하여 시스템전체를 리셋하는 용도로 사용할 수 있습니다. 전원은 기본적으로 VAA(5V, 로직전원)는 반드시 입력되어야 하며, VBB(24V, IO 구동용 전원)는 연결하지 않으면 릴레이이나 쿨링팬등의 24V로 구동되는 기능이 동작하지 않으며, 가변전원 VDC 가 출력되지 않습니다만, 기본적 동작에는 문제를 일으키지 않습니다.

| 20 핀 버스커넥터 핀맵            |        |              |               |
|--------------------------|--------|--------------|---------------|
| 핀번호                      | 명 칭    | 기 능          | 기 타           |
| 2                        | SDA    | I2C 데이터신호    |               |
| 4                        | SCL    | I2C 동기클럭     |               |
| 6                        | RESET/ | 시스템 리셋       | Active LOW    |
| 8,12,16,20               | VAA    | 로직전원, 5[V]   | 마이컴, IC       |
| 10,14,18                 | VBB    | IO 전원, 24[V] | VDC, 릴레이, 쿨링팬 |
| 1,3,5,7,9,11,13,15,17,19 | GND    | 시스템 그라운드     |               |

| 10 핀 버스커넥터 핀맵 |        |              |               |
|---------------|--------|--------------|---------------|
| 핀번호           | 명 칭    | 기 능          | 기 타           |
| 2             | SDA    | I2C 데이터신호    |               |
| 4             | SCL    | I2C 동기클럭     |               |
| 6             | RESET/ | 시스템 리셋       | Active LOW    |
| 8             | VAA    | 로직전원, 5[V]   | 마이컴, IC       |
| 10            | VBB    | IO 전원, 24[V] | VDC, 릴레이, 쿨링팬 |
| 1,3,5,7,9     | GND    | 시스템 그라운드     |               |



버스커넥터는 10 핀, 20 핀 두 종류가 있습니다. 위 그림처럼 10 번 핀 까지는 배치가 동일하므로 20 핀의 플랫케이블의 절반을 잘라서 10 핀 IDC 커넥터를 연결하면 10 핀 커넥터에 바로 연결할 수 있습니다. 또는 BPU(Bus Pull-Up) 모듈을 사용해도 됩니다.

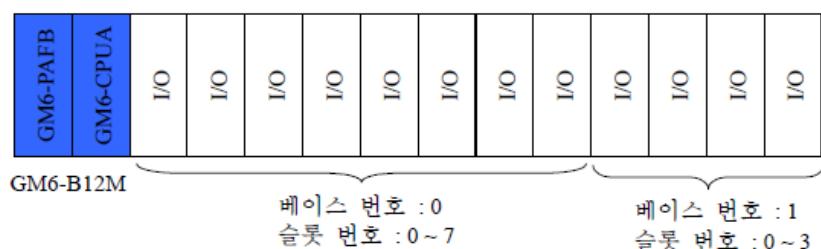
## 8. PLC 프로그래밍

산업현장에서 시퀀스제어에 가장 많이 사용되는 제어기는 아마도 PLC 일 것입니다. PLC 는 래더다이어그램이라는 그래픽적 프로그래밍 수단을 갖추어서 그 많은 제약에도 불구하고 다양한 분야에서 애용되고 있습니다. 여기서는 PLC 에 대한 기본적 내용을 살펴보고, ezplc.h 에서 제공하는 라이브러리를 이용하여 어떻게 래더다이어그램을 이지컨트롤러에서 사용할 수 있는지를 알아봅니다. PLC 에 대한 내용인 8.1 장은 건너뛰고 바로 “8.2ezplc 에 의한 래더다이어그램 구현”를 읽어도 좋습니다.

## 8.1. PLC 래더다이어그램

### 8.1.1. PLC의 기본구조

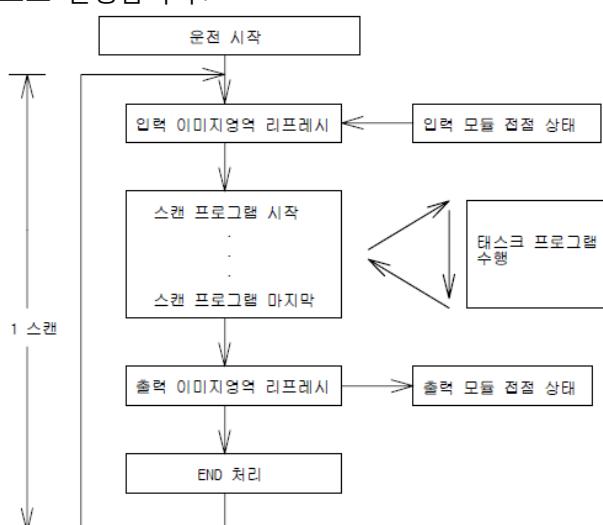
PLC 는 모듈을 장착할 수 있는 슬롯(Slot)을 여러 개 가지고 있는 베이스(Base)에 필요한 기능을 가지는 모듈을 끼워서 사용할 수 있도록 만들어져 있습니다.



각 목들의 입출력을 주소를 가지는데, 이 주소를 사용하여 프로그램을 작성할 수 있습니다.

### 8.1.2. Ladder 기본개요

PLC 에서 가장 흔히 사용되는 프로그래밍 수단이 래더ダイ어그램(Ladder Diagram, LD)입니다. LD 는 그림처럼 스캔방식(Scan)으로 동작하는데, 입력을 읽어오는 입력리플레시(Refresh), 프로그램을 순차적으로 실행하는 실행과정, 연산결과를 출력하는 출력리플레시 과정을 반복적으로 실행합니다.



### 8.1.3. 연산자 종류

LD 프로그램에서 사용되는 연산자는 크게 시퀀스연산자, 평선, 평선블록이 있습니다.

#### \*\* 시퀀스 연산자

##### ■ 입력접점

| 접점  | 이름         | 기능                                           |
|-----|------------|----------------------------------------------|
| I   | 평상시 열린 접점  | 해당 메모리의 논리값을 표시                              |
| I/I | 평상시 닫힌 접점  | 해당 메모리의 논리값을 반전하여 표시                         |
| I/P | 양 변환 겸출 접점 | 해당 메모리 논리값이 OFF → ON으로 변화하는 순간 1 스캔 시간 동안 ON |
| I/N | 음 변환 겸출 접점 | 해당 메모리 논리값이 ON → OFF로 변화하는 순간 1 스캔 시간 동안 ON  |

##### ■ 출력코일

| 코일   | 이름         | 기능                                      |
|------|------------|-----------------------------------------|
| (-)  | 출력 코일      | 접점 연산 결과를 지정된 접점으로 출력                   |
| (/)  | 반전 코일      | 접점 연산 결과를 반전하여 지정된 접점으로 출력              |
| -(P) | 양 변환 겸출 코일 | 연산 결과가 OFF → ON으로 변화하는 순간 1 스캔 시간 동안 ON |
| -(N) | 음 변환 겸출 코일 | 연산 결과가 ON → OFF로 변화하는 순간 1 스캔 시간 동안     |
| -(S) | 셋 코일       | 연산 결과가 1 번 ON 되었다가 OFF 되어도 출력값은 ON 유지   |
| -(R) | 리셋 코일      | 셋 되었던 접점을 리셋 시킴                         |

##### ■ 기타 시퀀스 연산자

| 연산자   | 이름         | 기능                         |
|-------|------------|----------------------------|
| <SC>  | 서브루틴 콜     | 메인 프로그램 연산 도중 서브루틴 프로그램 호출 |
| <RET> | 리턴(Return) | 서브루틴 연산 완료 후 메인 프로그램으로 복귀  |
| >>    | 점프(Jump)   | 레이블 위치로 연산 이동              |

#### \*\* 평선

평선은 한번의 스캔에서 즉시 연산결과를 계산할 수 있는 연산입니다. 예를 들어 크기비교를 하는 평선의 경우 즉시 비교가 가능하므로 즉시 출력을 계산할 수 있습니다. 종류는 전송, 형변환, 비교, 산술연산, 논리연산, 비트연산용이 있습니다. 개수가 매우 많아 몇가지만 살펴봅니다.

### ■ 전송평션

| 평 션      | 기 능                        |
|----------|----------------------------|
| MOVE     | 입력 변수를 출력 변수로 복사           |
| ARY_MOVE | 입력 배열 변수의 원소를 출력 배열 변수로 복사 |

### ■ 비교평션

| 평션 이름 | 기 능(단, n은 8까지 가능)                                                          |
|-------|----------------------------------------------------------------------------|
| GT    | '크다' 비교<br>(IN1 > IN2) And (IN2 > IN3) And ... And (INn-1 > INn) → OUT     |
| >     | (IN1 > IN2) And (IN2 > IN3) And ... And (INn-1 > INn) → OUT                |
| GE    | '크거나 같다' 비교<br>(IN1 ≥ IN2) And (IN2 ≥ IN3) And ... And (INn-1 ≥ INn) → OUT |
| ≥     | (IN1 ≥ IN2) And (IN2 ≥ IN3) And ... And (INn-1 ≥ INn) → OUT                |
| EQ    | '같다' 비교<br>(IN1 = IN2) And (IN2 = IN3) And ... And (INn-1 = INn) → OUT     |
| =     | (IN1 = IN2) And (IN2 = IN3) And ... And (INn-1 = INn) → OUT                |
| LE    | '작거나 같다' 비교<br>(IN1 ≤ IN2) And (IN2 ≤ IN3) And ... And (INn-1 ≤ INn) → OUT |
| ≤     | (IN1 ≤ IN2) And (IN2 ≤ IN3) And ... And (INn-1 ≤ INn) → OUT                |
| LT    | '작다' 비교<br>(IN1 < IN2) And (IN2 < IN3) And ... And (INn-1 < INn) → OUT     |
| <     | (IN1 < IN2) And (IN2 < IN3) And ... And (INn-1 < INn) → OUT                |
| NE    | '같지 않다' 비교<br>(IN1 ≠ IN2) And (IN2 ≠ IN3) And ... And (INn-1 ≠ INn) → OUT  |
| ≠     | (IN1 ≠ IN2) And (IN2 ≠ IN3) And ... And (INn-1 ≠ INn) → OUT                |

### \*\* 평션블록

평션블록은 한번의 스캔에서 출력력을 계산할 수 없으며, 여러번의 스캔을 반복하면서 결과를 계산하는 연산입니다. 그러므로, 기능이 복잡하고 혼돈되는 연산입니다. 그러나, 시퀀스 프로그램에서 없어서는 않을 매우 중요한 기능이므로 반드시 사용할 수 있도록 숙지해야 합니다.

### ■ 카운터 및 타이머

| 평션 블록 | 기 능                             | 비고             |
|-------|---------------------------------|----------------|
| CTD   | 다운 카운터 (현재값 : -32768 ~ 32767)   |                |
| CTU   | 업 카운터 (현재값 : -32768 ~ 32767)    |                |
| CTUD  | 업 다운 카운터 (현재값 : -32768 ~ 32767) |                |
| TOF   | OFF Delay 타이머                   |                |
| TON   | ON Delay 타이머                    |                |
| TP    | 펄스 타이머                          | 타임형<br>설정값(T#) |

## 8.2. ezplc 에 의한 래더다이어그램 구현

이지컨트롤러에서 LD(래더)프로그램의 작성률 지원하기 위하여 아래와 같은 eventbus\_fw\_0.60\eventbus\include\ezplc.h 를 제공합니다.

예제파일(application\exam\plc\_ex1.c)을 참고해 주십시오. 각 구조체는 Ladder Diagram 을 의미하는 접두어 ‘L’로 시작합니다.

(주의) system var 로 주석이 달린 변수는 사용자용이 아니므로 여기에는 표시하지 않았습니다. 그리고 v0.60 에서 명칭이 변경되었으므로 다시 한번 확인바랍니다.

```

typedef struct {
    ELogic     QP;           // positive output
    ELogic     QN;           // negative output
} LEdge;

typedef struct {
    ELogic     Q;            // Timer Output
    uint32    ET;            // Elapsed Time
} LTimer;

typedef struct {
    ELogic     Q;            // Counter Output
    int32      CV;           // Current Counter Value
} LCounter;

typedef struct {
    ELogic     QU;           // Up Counter Output
    ELogic     QD;           // Down Counter Output
    int32      CV;           // Current Counter Value
} LUUpDown;

typedef struct {
    ELogic     Q;            // Oscillating Clock Output
    uint32    N;             // Count of the fired pulse
} LOsc;

void Edge(LEdge* edge, ELogic in);
void OnDelay(LTimer* tmr, ELogic in, uint32 pt);
void OffDelay(LTimer* tmr, ELogic in, uint32 pt);
void OneShot(LTimer* tmr, ELogic in, uint32 pt);
void UpCount(LCounter* cnt, ELogic cu, ELogic rst,int32 pv);
void DownCount(LCounter* cnt, ELogic cd, ELogic ld,int32 pv);
void UpDownCount(LUpDown* cnt,
                  ELogic cu, ELogic cd, ELogic rst, ELogic ld, int32 pv);
void Osc(LOsc* osc, ELogic in, uint32 pt);

// 자기유지회로
extern ELogic Hold (ELogic set, ELogic rst, ELogic coil);

// 인터록 회로
extern ELogic Interlock (ELogic in, ELogic with);

```

그런데, 앞의 PLC에서 보았던 연산자의 대부분이 보이지 않습니다. 단지 평선클록 연산자인 타이머, 카운터, 에지검출용 평선클록만 있으며, 자기유지회로를 구현하는 Hold 와 Interlock 평션과 v0.60 부터 osc 평션이 추가되었습니다.

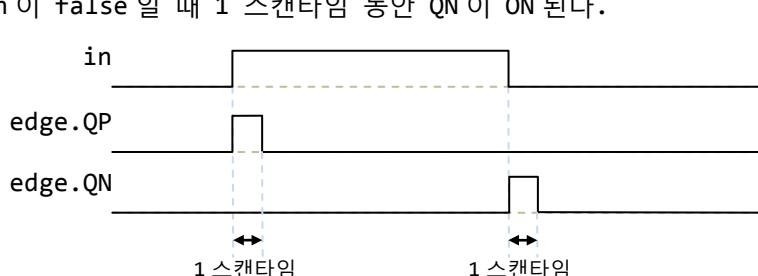
이유는 여기에 구현되지 않은 나머지 연산자는 모두 C 언어에서 기본적으로 제공하는 기능이므로 별도로 제공하지 않습니다. 예를 들어, MOVE 연산자는 대입연산자로 구현되며 논리연산자는 C 언어의 논리연산자를 사용하면 됩니다. 기타 형변환 연산자는 사용자가 필요에 따라 함수로 작성할 수 있으므로 별도로 제공되지 않습니다. C 언어에서 제공하는 기능이 LD 보다 훨씬 강력한데 굳이 별도로 제공할 이유가 없기 때문입니다.

### 8.2.1. Edge, 에지검출

입력접점, 출력접점, 논리식의 결과의 상승 또는 하강에지를 검출하여 1 스캔타임 동안 결과를 출력합니다. PLC 와 달리 논리식으로 표현될 수 있는 것은 무엇이든 입력으로 사용할 수 있습니다. 아래의 사용예에서 볼 수 있듯이 PLC에서 제공하는 연산자와는 기능 및 표현력의 측면에서 비교되지 않을 정도로 강력합니다.

```
typedef struct {
    ELogic      QP;           // positive output
    ELogic      QN;           // negative output
} LEdge;
```

- **void Edge (LEdge\* edge, ELogic in);**  
edge : 에지검출용 구조체  
in : 검출대상 입력신호 (true, false로 해석될 수 있는 값)
- 동작  
in이 true일 때 1 스캔타임 동안 QP 가 ON 되고,  
in이 false일 때 1 스캔타임 동안 QN이 ON 된다.



- 사용예  

```
LEdge    edge1, edge2, edge3, edge4;
Edge(&edge1, this->in->sw0);                      // 스위치의 에지
Edge(&edge2, this->out->rly0);                     // 릴레이의 에지
Edge(&edge3, this->in->sw0 && this->in->rly0);   // 논리식 사용
Edge(&edge4, this->in->sw0 && (count > 3));     // 논리, 관계식 사용
```
- 매크로의 사용으로 간략화 할 수 있다.  

```
#define SW0 (this->in->sw0)
#define RY0 (this->out->rly0)
```

```

LEdge edge1, edge2, edge3, edge4;
FindEdge (&edge1, SW0);
FindEdge (&edge2, RY0);
FindEdge (&edge3, SW0 && RY0);
FindEdge (&edge4, SW0 && (count > 3));

```

### 8.2.2. Hold, 자기유지회로

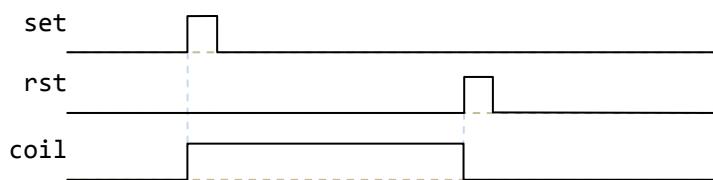
자기유지회로는 일반적으로 많이 사용되므로 별도로 제공합니다. 입력 IN 이 한번 ON 되면 COIL 이 ON 되고, 이후 IN 이 OFF 되어도 계속 ON 상태를 지속합니다. RST 를 ON 하면 COIL 도 리셋됩니다.



- **ELogic Hold (ELogic set, ELogic rst, ELogic coil);**  
 set : 세트 입력 (true, false 값)  
 rst : 리셋 입력 (true, false 값)  
 coil : 자기유지할 출력점점(coil)  
 반환값 : 자기유지된 출력점점(coil)의 상태값

- 동작

set 0| true 일 때 coil 0| ON 되고, rst 0| true 일 때 coil 0| OFF 된다.



- 사용예



**RY0 = Hold (PB0, PB1, RY0);**

푸쉬버튼 PB0 을 누르면 릴레이 RY0 이 ON 되고,

푸쉬버튼 PB1 을 누르면 릴레이 RY0 이 OFF 된다.



**RY0 = Hold (PB0, PB1 || PB2, RY0);**

푸쉬버튼 PB0 을 누르면 릴레이 RY0 이 ON 되고,

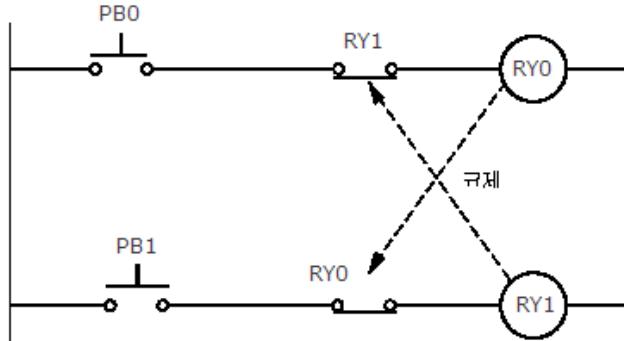
푸쉬버튼 PB1 또는 PB2 를 누르면 릴레이 RY0 이 OFF 된다.

**RY0 = Hold (count == 3, count > 8, RY0);**

count 가 3 이 되면 RY0 이 ON 되고,  
count 가 8 보다 커지면 RY0 이 OFF 된다.

### 8.2.3. Interlock, 인터록회로

인터록회로는 상대가 동작중일 때는 동작을 금지시키는 상호동작방지 회로입니다. 이것은 상대편이 사용중일 때는 사용할 수 없도록 하여 자원을 분배할 때 사용됩니다. 예를 들어, 컴프레셔의 압력이 낮을 때 공압튜이 동작하지 않도록 하는 것과 같습니다.



그림에서 처럼 RY1 이 동작중이라면 PB0 을 눌러도 RY0 은 동작하지 않는 것이 인터록입니다.

- `ELogic Interlock (ELogic set, ELogic lock);`  
 set : 출력을 세트시키는 입력신호, 그림의 PB0 에 해당  
 lock : 인터록하는 대상, 그림의 RY1 에 해당  
 반환값 : 인터록된 결과 상태값, 그림의 RY0 에 해당
- 사용 예  
`RY0 = Interlock (PB0, RY1);`  
 PB0 에 따라 RY0 이 동작하지만, RY1 이 ON 이면 RY0 은 무조건 OFF 된다.  
  
`RY0 = Interlock (count > 3, volt < 24);`  
 count 가 3 보다 크면 RY0 은 ON 되지만, 전압이 24V 보다 낮으면 RY0 은 OFF 된다.  
  
`RY0 = Interlock(PB0, RY1);`  
`RY1 = Interlock(PB1, RY0);`  
 위 그림을 구현한 코드이다.

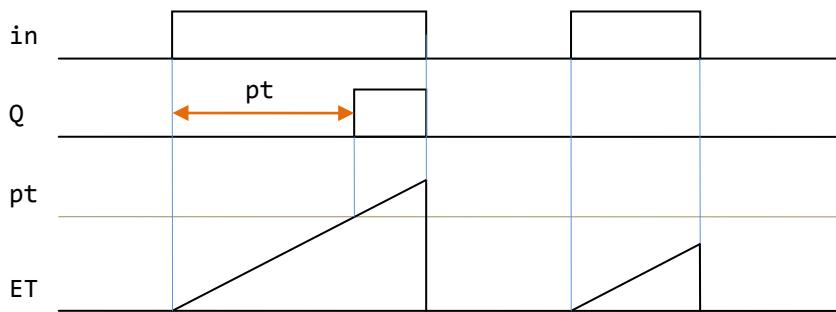
### 8.2.4. OnDelay, OffDelay, OneShot 타이머

타이머는 평션블록 연산자이며, 여러 스캔타임에 걸쳐 연산이 이루어지는 특징이 있습니다. 타이머는 시간을 측정/지연하는데 응용되며 구조체 `LTimer` 가 사용됩니다.

```
typedef struct {
    ELogic      Q;          // Timer Output
    uint32      ET;         // Elapsed Time
} LTimer;
```

- **온 딜레이 타이머(OnDelay Timer)**  
`void OnDelay (LTimer* tmr, ELogic in, uint32 pt);`

| ON-Delay Timer                                                                                                                                            | 설명                                                                                                           |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| <pre>     graph LR       in[IN] --&gt; tmr[tmr]       pt[pt] --&gt; tmr       tmr -- Q --&gt; outQ[OUT Q]       tmr -- ET --&gt; outET[OUT ET]     </pre> | <p>in : 타이머 기동조건<br/>   pt : Preset Time, 지연시간(ms)<br/>   Q : 타이머 접점출력<br/>   ET : Elapsed Time, 경과시간 출력</p> |



in의 상승에지(false→true, OFF→ON)에서 ET(경과시간)가 증가하기 시작하고, pt(설정시간)에 도달하면 Q(출력)가 ON됩니다. pt보다 입력펄스의 폭이 더 긴 경우에만 출력이 발생합니다.  
 (예제 프로그램)

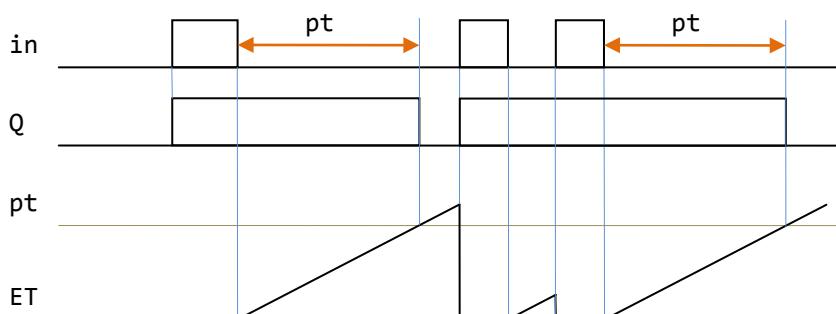
```

LTimer tmr;
OnDelay(&tmr, PB0, 1000);           // PB0 누르면 1초 후
RY0 = tmr.Q;                      // 릴레이 RY0이 ON된다.
printf("%d,%d\r\n",tmr.Q, tmr.ET); // PC에 결과표시
  
```

### ■ 오프 딜레이 타이머(OffDelay Timer)

```
void OffDelay (LTimer* tmr, ELogic in, uint32 pt);
```

| OFF-Delay Timer                                                                                                                                           | 설명                                                                                                           |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| <pre>     graph LR       in[IN] --&gt; tmr[tmr]       pt[pt] --&gt; tmr       tmr -- Q --&gt; outQ[OUT Q]       tmr -- ET --&gt; outET[OUT ET]     </pre> | <p>in : 타이머 기동조건<br/>   pt : Preset Time, 지연시간(ms)<br/>   Q : 타이머 접점출력<br/>   ET : Elapsed Time, 경과시간 출력</p> |



in의 상승에지(false→true, OFF→ON)에서 Q가 ON되고, 하강에지에서 타이머가 동작해서 설정시간(pt)이 되면 Q가 OFF됩니다. 설정된 지연시간(pt) 이전에 다시 in이 ON되면 타이머는 리셋되어 다시 in의 하강에지를 기다립니다.

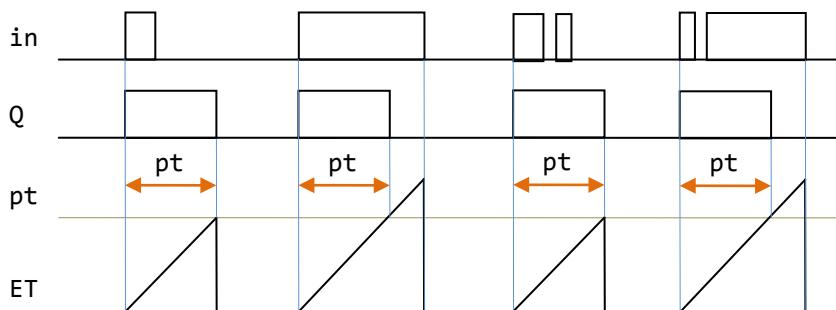
(예제 프로그램)

```
LTimer tmr;
OffDelay(&tmr, PB0, 1000);           // PB0 누르면 RY0이 ON 되고
RY0 = tmr.Q;                      // PB0 놓으면 1초후 RY0이 OFF된다.
printf("%d,%d\r\n", tmr.Q, tmr.ET); // PC에 결과표시
```

### ■ 원샷(펄스) 타이머(OneShot)

```
void OneShot (LTimer* tmr, ELogic in, uint32 pt);
```

| Pulse Timer | 설명                                                                                           |
|-------------|----------------------------------------------------------------------------------------------|
|             | in : 타이머 기동조건<br>pt : Preset Time, ON 출력시간(ms)<br>Q : 타이머 접점출력<br>ET : Elapsed Time, 경과시간 출력 |



in의 상승에지(false→true, OFF→ON)에서 Q가 ON되고 타이머가 동작합니다. 설정시간(pt)이 되면 Q가 OFF되어 일정폭(pt)의 펄스를 출력합니다. (주의)Non-Trigerable 타입이므로 펄스가 출력중(Q=ON)일 때 입력되는 in 펄스는 무시됩니다.

원샷타이머는 일정폭의 동작펄스를 만들 때 사용합니다. 예를 들어, 스위치를 한번 누르면 모터를 10초동안 회전시키고자 할 때, 모터의 동작접점을 만드는 용도와 같은 것입니다.

(예제 프로그램)

```
LTimer tmr;
OneShot(&tmr, PB0, 1000);           // PB0 누르면 RY0이 ON 되고
RY0 = tmr.Q;                      // 1초후 RY0이 OFF된다.
printf("%d,%d\r\n", tmr.Q, tmr.ET); // PC에 결과표시
```

### 8.2.5. UpCount, DownCount, UpDownCount 카운터

카운터는 입력이 발생한 횟수를 세는 연산자이며, 타이머와 마찬가지로 여러 스캔타임에 걸쳐 연산이 이루어지는 특징이 있습니다. 업카운터와 다운카운터는 LCounter 구조체를 사용하고, 업다운카운터는 LUpDown 구조체를 사용합니다.

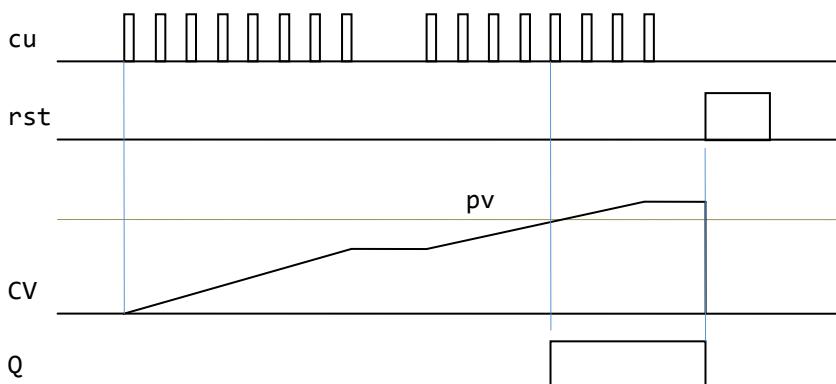
```
typedef struct {
    ELogic      Q;           // Counter Output
    int32       CV;          // Current Counter Value
} LCounter;

typedef struct {
    ELogic      QU;          // Up Counter Output
    ELogic      QD;          // Down Counter Output
    int32       CV;          // Current Counter Value
} LUpDown;
```

### ■ 업 카운터(UpCount)

```
void UpCount (LCounter* cnt, ELogic cu, ELogic rst, int32 pv);
```

| Up Counter | 설 명                                                                                                  |
|------------|------------------------------------------------------------------------------------------------------|
|            | cu : 카운트업 펄스입력<br>pv : Preset Value, 설정개수<br>rst : 리셋 입력<br>Q : 카운터 접점출력<br>CV : Current Value, 현재개수 |



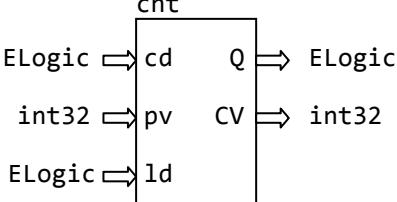
cu에 입력되는 펄스의 개수를 카운트하여 pv(설정개수)에 도달하면 출력 Q가 ON 됩니다.  
리셋 R은 현재 카운터 값과 출력을 클리어시킵니다.

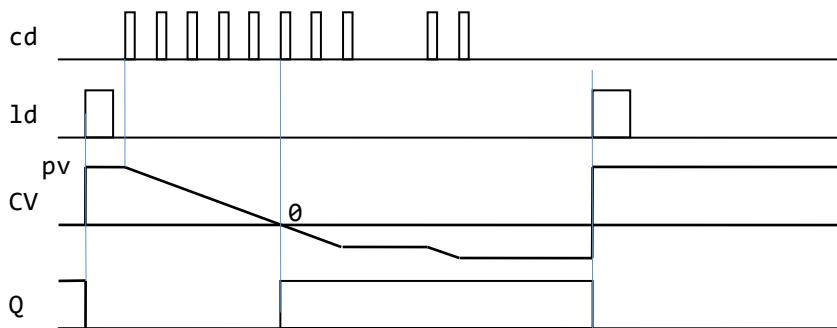
(예제 프로그램)

```
LCounter  cnt;
UpCounter(&cnt, PB0, PB1, 3);    // PB0을 3 개 카운트한다. PB1 은 리셋
RY0 = cnt.Q;                    // 3 개이면 RY0 을 ON 시킨다.
printf("%d\r\n", cnt.CV);       // 카운트한 현재개수를 PC 에 출력한다.
```

### ■ 다운 카운터(DownCount)

```
void DownCount (LCounter* cnt, ELogic cd, ELogic ld, int32 pv);
```

| Down Counter                                                                      | 설명                                                                                                   |
|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
|  | cd : 카운트다운 펄스입력<br>pv : Preset Value, 설정개수<br>ld : 로드 입력<br>Q : 카운터 접점출력<br>CV : Current Value, 현재개수 |



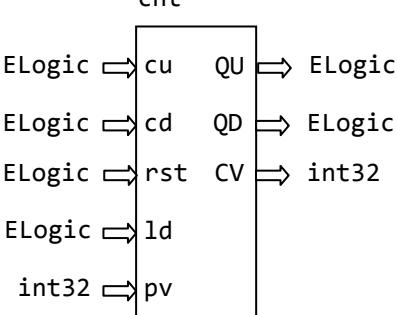
ld는 pv(설정개수)를 cv(현재개수)에 복사합니다. 입력펄스 cd는 cv를 감소시키고, cv가 0 이하일 때 출력 Q 가 ON 됩니다. 초기에는 카운트 값이 0 이므로 출력 Q 도 ON 되는 점에 주의하십시오.

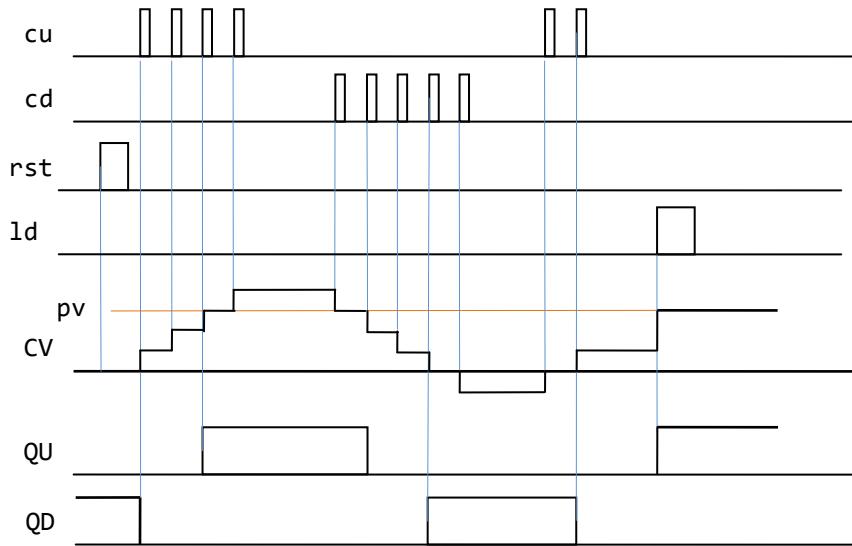
(예제 프로그램)

```
LCounter cnt;
DownCount(&cnt, PB0, PB1, 3); // PB0 을 3 개 카운트한다. PB1 은 로드
RY0 = cnt.Q; // 3 개이면 RY0 을 ON 시킨다.
printf("%d\r\n", cnt.CV); // 카운트한 현재개수를 PC 에 출력한다.
```

### ■ 업다운 카운터(UpDownCount)

```
void UpDownCount (LUpDown* cnt,
                  ELogic cu, ELogic cd, ELogic rst, ELogic ld, int32 pv);
```

| Up Down Count                                                                       | 설명                                                                                                                                                         |
|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | cu : 카운트업 펄스입력<br>cd : 카운트다운 펄스입력<br>rst : 리셋 입력<br>ld : 로드 입력<br>pv : Preset Value, 설정개수<br>QU : 업카운터 접점출력<br>QD : 다운카운터 접점출력<br>CV : Current Value, 현재개수 |



cu 는 업카운트 펄스, cd 는 다운카운트 펄스로 현재값 cv 를 증가 또는 감소시킵니다. 리셋 rst 은 cv 를 0 으로 클리어 하고, 로드 ld 는 cv 를 설정값 pv 로 초기화합니다.

cv 가 0 이하일 때 QD 가 ON 되고, cv 가 pv 이상일 때 QU 가 ON 됩니다. 초기에는 cv 가 0 이므로 QD 가 ON 되는 점에 주의하십시오.

입력신호의 우선순위는  $\text{rst} > \text{ld} > \text{cu}, \text{cd}$  이고, 우선순위가 높은 입력이 처리될 때는 다른 입력은 무시됩니다. 즉, rst, ld 가 동시입력되면 rst 동작만 실행되어  $\text{cv}=0$  이 됩니다. 또한, rst 또는 ld 가 입력될 때 cu, cd 에 펄스가 입력되어도 카운트되지 않습니다. 단, cu, cv 가 동시에 상승에지이면 카운트는 증가도 감소도 하지 않습니다.

(예제 프로그램)

```
LUpDown cnt;
UpDownCount(&cnt, PB0, PB1, PB2, PB3, 3);           // 3 개 카운트한다.
RY0 = cnt.QU;   // 3 개이상이면 RY0 을 ON 시킨다.
RY1 = cnt.QD;   // 0 개이하면 RY1 을 ON 시킨다.
printf("%d\r\n", cnt.CV);                            // 카운트한 현재개수를 PC 에 출력한다.
```

### ■ 오실레이터(Oscillator, 발진기)

```
void Osc (LOsc* osc, ELogic in, uint32 pt);
```

| Oscillator                                                                                                                             | 설 명                                                                                       |
|----------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <pre>     graph LR       in[ELogic] --&gt; osc[osc]       osc -- Q --&gt; outQ[ELogic]       osc -- N --&gt; outN[uint32]       </pre> | <p>in : 발진기 기동조건<br/> pt : Preset Time, 발진주기(ms)<br/> Q : 발진기 접점출력<br/> N : 출력펄스 누적횟수</p> |



in이 ON인 동안 발진기가 동작합니다. 발진주기(pt)마다 출력 Q는 1 스캔타임동안 ON인 펄스를 출력합니다. 원샷타이머는 일정폭의 동작펄스를 만들 때 사용합니다. 예를 들어, 비상스위치(EMG)를 누르면 경광등(RY0)을 1 초간격으로 켜고 끄는 동작을 반복시키려면 아래와 같이 하면됩니다.

(예제 프로그램)

```
LTimer tmr;
LOsc osc;
Osc(&osc, EMG, 2000); // 2초 주기의 펄스를 만들고,
OneShot(&tmr, osc.Q, 1000); // 펄스마다 1초간 ON 시킨 후,
RY0 = tmr.Q; // 원샷타이머의 출력으로 경광등을 동작시킨다.
```

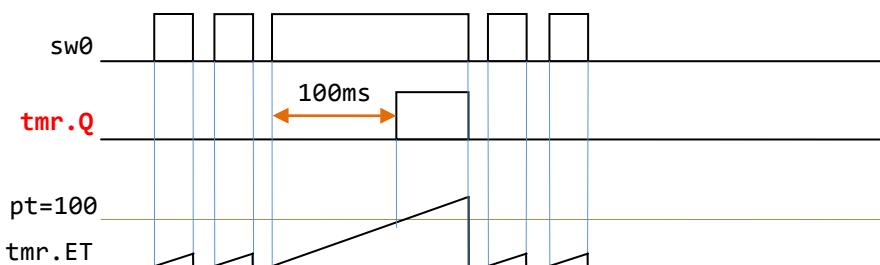
### 8.3. 평션블록의 사용예

Edge(), OnDelay(), OffDelay(), OneShot(), UpCount(), DownCount(), UpDownCount(), Osc()를 이용한 여러가지 사용예를 살펴보겠습니다.

#### ■ 디바운스(1)

: 스위치의 바운스를 제거하여 오동작을 방지하는 프로그램

```
OnDelay(&tmr, this->in->sw0, 100); // 100ms 미만의 'H' 바운스는 제거된다.
```

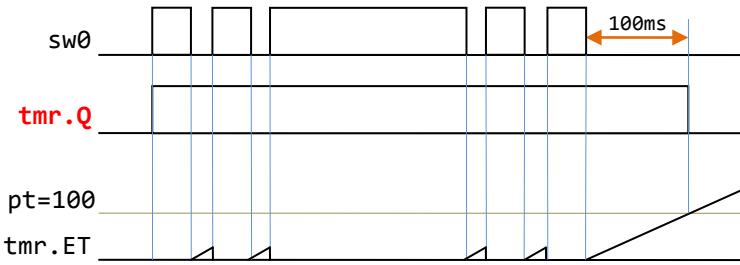


위 타임챠트에서 보듯이 sw0에는 바운스가 있어서 오동작의 원인이 될 수 있지만, tmr.Q는 바운스가 제거된 깨끗한 파형이 되므로, sw0을 사용할 위치에 대신 tmr.Q를 사용하면 됩니다. 단, sw0 보다 약간의 지연이 있음에 유의해야 합니다.

#### ■ 디바운스(2)

: 지연없는 즉각적인 입력이 필요한 경우는 OffDelay() 타이머를 사용합니다.

```
OffDelay(&tmr, this->in->sw0, 100); // 100ms 미만의 'L' 바운스는 제거된다.
```

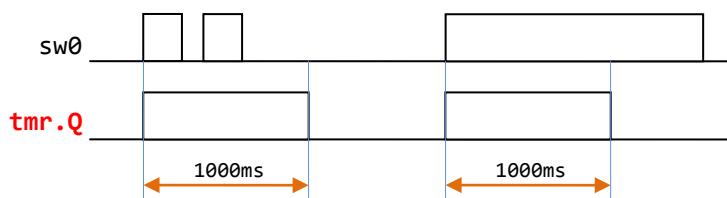


위 타임챠트에서 보듯이 **sw0**에는 바운스가 있어서 오동작의 원인이 될 수 있지만, **tmr.Q**는 바운스가 제거된 깨끗한 파형이 되므로, **sw0**을 사용할 위치에 대신 **tmr.Q**를 사용하면 됩니다. 단, `OnDelay()` 타이머를 사용한 디바운스에 비해 자연시간은 없지만, 노이즈에 취약하므로 용도에 맞게 선택할 필요가 있습니다.

### ■ 원샷 (일정시간 동작)

: 스위치를 누른 시간이 길고 짧음에 상관없이 일정시간 동작시키는 프로그램

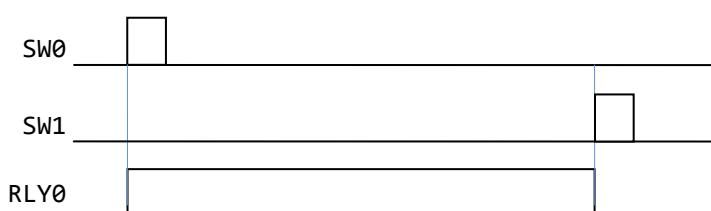
```
OneShot(&tmr, this->in->sw0, 1000); // tmr.Q는 1000ms 펄스폭이 된다.
```



### ■ 스타트-스톱 동작

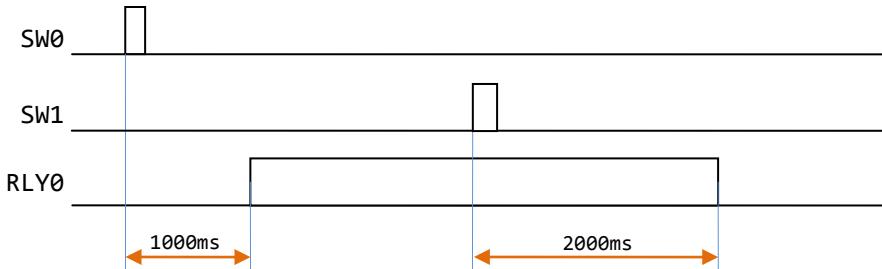
: SW0은 출력을 ON 시키고, SW1은 출력을 OFF 시키는 프로그램

```
RLY0 = Hold(SW0, SW1, RLY0); // SW0(세트), SW1(리셋)
```



### ■ 지연된 스타트-스톱 동작

: SW0은 1초후에 출력을 ON시키고, SW1은 2초후 출력을 OFF시키는 프로그램



```
#define SW0      (this->in->sw0)
#define SW1      (this->in->sw1)
#define RLY0     (this->out->rlly0)

LTimer tmr0,tmr1;
ELogic coil0,coil1;

int main()
{
    EventBus();
    EventProgram;
    ScanProgram;
    // 세트 1초 딜레이
    if (SW0) { coil0 = ON; }
    OnDelay(&tmr0, coil0, 1000);

    // 리세트 2초 딜레이
    if (SW1) { coil1 = ON; }
    OnDelay(&tmr1, coil1, 2000);

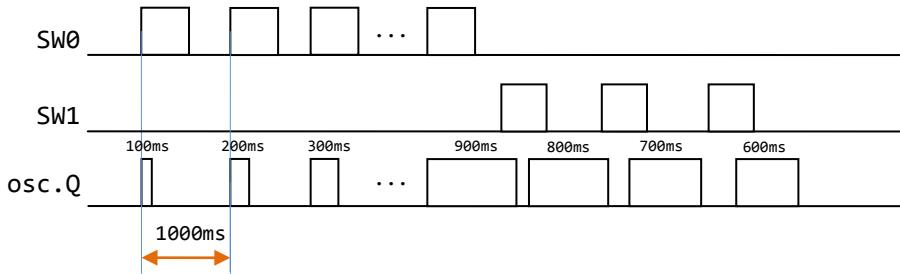
    RLY0 = Hold(tmr0.Q, tmr1.Q, RLY0);

    // 출력이 OFF될 때 변수를 초기화시켜서 다음 동작에 대비함
    if (tmr1.Q) {coil0 = coil1 = OFF; }

    // 입력 리플레이
    Eval_DioIn(FALSE);
    End;
}
```

### ■ PWM 출력

: 스위치를 누르면 펄스폭이 조절되는 PWM 신호를 만든다.



타임챠트에서 보듯이 SW0 을 누르면 펄스폭이 증가하고, SW1 을 누르면 펄스폭이 감소합니다. 단, 주의할 점은 펄스폭이 100%인 펄스는 출력할 수 없습니다. 이유는 OneShot()는 Non-Trigerable 타입이므로 펄스가 출력중인동안 다시 입력이 들어와도 무시하기 때문입니다.

```
#define SW0    (this->in->sw0)
#define SW1    (this->in->sw1)
#define RLY0   (this->out->rly0)

Losc    osc;
LTimer  tmr;

#define period 1000      // 펄스 주기
int width;                // 펄스 'H'폭

int main()
{
    EventBus();

    EventProgram;
    IF (this, DioIn) {
        if (Pressed(this->in, sw0)) {           // 펄스폭 증가
            width += 100;
            if (width > period) width = period;
        } else if (Pressed(this->in, sw1)) {       // 펄스폭 감소
            width -= 100;
            if (width < 0) width = 0;
        }
    }

    ScanProgram;
    Osc(&osc, ON, period);                  // 1000ms마다 1스캔타임 펄스를 생성
    OneShot(&tmr, osc.Q, width);           // 펄스폭을 width만큼 늘린다.
    RLY0 = tmr.Q;                          // PWM 출력으로 릴레이를 동작시킨다.

    Eval_DioIn(FALSE);                   // 입력 리플레시
    End;
}
```

#### ■ 버튼 연속입력 카운트

: 버튼을 수백번 누르는 것은 불편하므로, 버튼을 누르고 있으면 연속으로 버튼을 누른

것처럼 펄스를 발생시킵니다. 시간이 지남에 따라서 펄스의 주기가 짧아지도록 기능을 추가해 보았습니다.



타임챠트에서 보듯이 SW0 을 누르면 1 스캔타임의 짧은 펄스가 발생하고, SW0 을 계속 누르고 있으면 연속으로 펄스가 발생하는 프로그램입니다. 최소주기를 100ms 로 설정해서 계속 누르더라도 더이상 빨라지지 않도록 했습니다. 이것은 사용자로부터 갯수를 입력받는 것처럼 여러번 버튼을 눌러야 하는 경우에 불편함을 없애기 위해서 사용할 수 있습니다.

---

```

#define SW0      (this->in->sw0)
#define SW1      (this->in->sw1)
#define RLY0     (this->out->rly0)

Losc    osc;

#define PERIOD      1000
#define PERIOD_MIN   100
uint32 period;

int main()
{
    EventBus();

    EventProgram;

    ScanProgram;
    if (osc.N)
        period = PERIOD / (osc.N + 1) + PERIOD_MIN; // 주기를 짧게 변경
    else
        period = PERIOD;           // 주기 초기화

    Osc(&osc, SW0, period);    // SW0이 ON인 동안 주기에 맞춰 펄스발생

    Eval_DioIn(FALSE);         // 입력 리플레시
    End;
}

```

---

## 8.4. 시퀀스 프로그램 주의사항

래더 프로그램을 처음 접하면 가장 어려워하는 부분이 C, JAVA 와 같은 기존의 절차적 프로그래밍과 사고방식이 다르다는 점입니다. 즉, 절차적 언어와 래더는 순서대로 실행된다는 점에서는 비슷합니다. 그러나 래더는 스캔방식으로 전체프로그램이 무한히 반복되는 구조를 가지고 있습니다. 그래서 어느 한 부분에서 프로그램의 흐름을 잡고(기다리도록)있도록 프로그램하는 것은 좋지 않습니다. **프로그램은 멈추거나 기다림 없이 계속 반복되어야 합니다.**

| 순서도 | 절차 프로그램                                                                                                                                     | ScanProgram                                                  |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
|     | <pre>if (SW0 == ON)     RLY0 = OFF; else     RLY0 = ON;</pre> <p>또는</p> <pre>RLY0 = SW0 ? OFF : ON;</pre> <p>또는</p> <pre>RLY0 = !SW0;</pre> | 절차 프로그램과 동일                                                  |
|     | <pre>while (!SW0);     RLY0 = ON;</pre>                                                                                                     | <b>while 문에서 멈춘다.</b> <pre>if(SW0) {     RLY0 = SW0; }</pre> |

마지막 순서도를 눈여겨 보아야 합니다. SW0 이 OFF 이면 더 이상 프로그램이 진행하지 못하고 ON 이 될 때까지 멈추게 됩니다. 시퀀스 프로그램에서는 프로그램의 반복실행이 멈추는 것은 좋지 않습니다. 이렇게 제어가 멈추면 다른 스위치나 릴레이등의 입출력을 처리할 수 없으므로 심각한 문제를 야기합니다. 심지어 SW0 이 고장나면 다른 기능도 모두 멈추어 버리게 되므로 위험합니다.

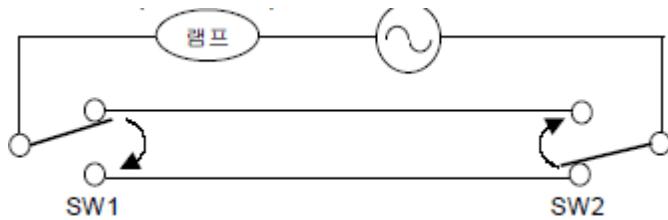
이런 경우는 오른쪽 코드에서처럼 SW0 이 OFF 라면 일단은 다음번 반복때 처리하도록 남겨두고 프로그램은 계속 진행시키는 방법으로 바꾸어야 합니다.

## 8.5. 래더프로그래밍 예제

위에서 살펴본 연산자를 이용하여 래더 다이어그램을 작성하고 이를 이벤트버스를 이용하여 다시 구현하며 비교하겠습니다.

### 8.5.1. 삼로 스위치

긴 복도나 계단의 양단에 있는 스위치를 사용해서 전등을 켜고 끌 수 있도록 만들어진 회로입니다.



계단을 오르기 전에 1 층 스위치 SW1 을 ON 하면 램프가 켜집니다. 2 층에 올라와서 스위치 SW2 를 OFF 하면 램프를 끌 수 있습니다.



위의 LD(래더다이어그램)를 이벤트버스로 구현하면 아래와 같습니다. 이벤트버스에서 LD 프로그램 하려면 “ezplc.h” 헤더를 인크루드해야 합니다. 그리고 입출력을 네임드변수처럼 미리 #define 을 사용하여 정의해 두면 프로그램의 가독성이 향상되어 편리합니다.

```
#include "ezconfig.h"
#include "ezplc.h"
#define plc_ex1

#include "plc_ex1.h"
#include "dio_drv.h"

DIO* this;

#define SW0    (this->in->switch0)
#define SW1    (this->in->switch1)
#define RY0    (this->out->relay0)

int main()
{
    EventBus();
    EventProgram;
    ScanProgram;
    RY0 = (SW0 && !SW1) || (!SW0 && SW1);

    Eval_DioIn(FALSE); // 입력 리프레시
    End;
}
```

**#endif**

이벤트버스 V0.50 부터 이벤트프로그램 외에도 스캔프로그램이 지원됩니다. 스캔프로그램은 무한반복으로 실행되므로 PLC의 래더다이어그램을 구현하기에 적절합니다.

물론, 이벤트프로그램도 함께 사용할 수 있으므로 이를 적절히 섞어서 사용하면 매우 편리하게 프로그램을 작성할 수 있습니다.

주의할 점은 마지막 부분에서 사용된 모든 모듈의 입력이미지를 리플레시하는 코드를 넣어주어야 합니다. (출력이미지는 V0.60부터는 자동으로 리플레시됩니다)

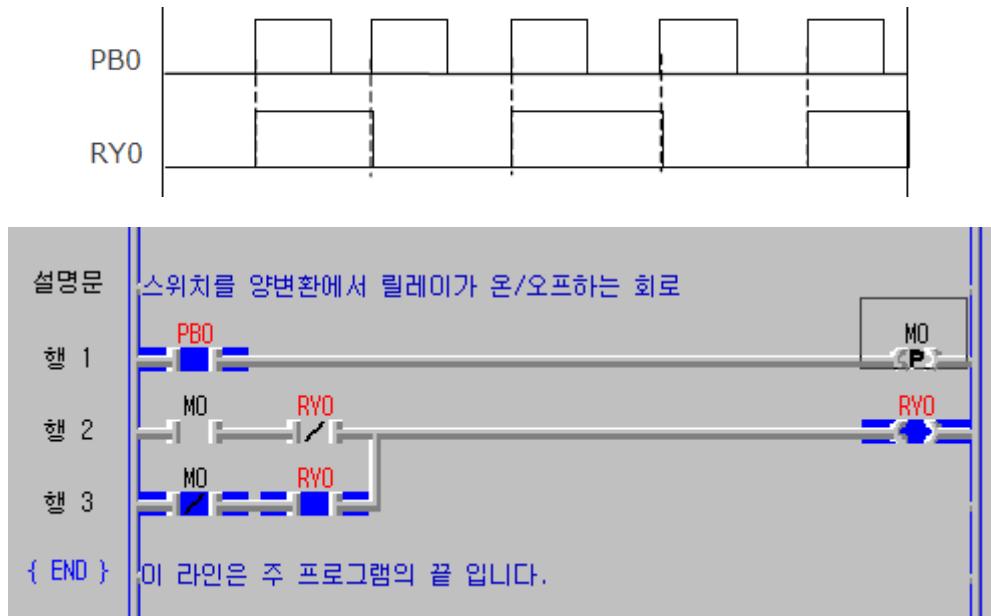
위의 래더다이어그램과 프로그램은 모두 논리식을 사용하므로 프로그램이 단순해지는 장점은 있지만, 그 의미를 파악하기가 어려운 단점이 있습니다. 그러나, C 언어를 사용하는 이벤트버스에서는 시스템의 동작을 분석하여 프로그램으로 옮길 수 있으므로 의미를 파악하기 쉽도록 프로그램을 수정할 수 있습니다.

위의 삼로스위치는 SW1과 SW2의 상태가 서로 같으면 램프가 점등하고, 서로 다르면 소등됩니다(반대로 생각해도 마찬가지입니다). 이 성질을 이용하면 다음과 같이 프로그램 할 수 있습니다. (편의상 main 함수만 다시 씁니다)

```
int main()
{
    EventBus();
    EventProgram;
    ScanProgram;
    RY0 = (SW0 == SW1);
    Eval_DioIn(FALSE);      // 입력 리프레시
    End;
}
```

### 8.5.2. 다이나믹 플립플롭 회로

스위치(PB0)의 상승에지에서 출력(RY0)을 토글하는 회로입니다.



위의 LD(래더다이어그램)를 이벤트버스로 구현하면 아래와 같습니다. 이벤트버스에서 LD 프로그램 하려면 “ezplc.h” 헤더를 인크루드해야 합니다. 그리고 입출력을 네임드변수처럼 미리 #define 을 사용하여 정의해 두면 프로그램의 가독성이 향상되어 편리합니다.

```

DIO* this;

#define SW0      (this->in->switch0)
#define RY0      (this->out->relay0)

LEdge edge;                                // 에지검출용 변수

int main()
{
    EventBus();
    EventProgram;
    ScanProgram;
    Edge(&edge, SW0);
    RY0 = (edge.QP && !RY0) || (!edge.QP && RY0);
    Eval_DioIn (FALSE);                      // 입력 리프레시
    End;
}
#endif

```

여기서는 LEdge 타입과 Edge()함수를 사용하여 양변환검출을 구현하였습니다 (ezplc.h 참조). Edge()함수는 입력접점에서 상승 및 하강시점을 찾아서 1 스캔타임 동안만 QP, QN에 양변환에지, 음변환에지를 출력합니다.

이번에도 마찬가지로 시스템의 동작을 분석해 보면, 결국 이 시스템은 입력(PB0)의 상승에지에서 출력을 토글하는 것이므로 아래처럼 프로그램 할 수 있습니다.

```
// Dynamic Flip-Flop
Edge(&edge, SW0);
if (edge.QP) RY0 = !RY0;
```

스캔프로그램은 많은 장점에도 불구하고 모든 코드를 반복실행하므로 프로그램이 커지면 속도가 느려질 수 있다는 단점이 있습니다. 반면, 이벤트프로그램은 이벤트가 발생한 시점에서 한번만 실행되므로 실행속도 측면에서 유리합니다.

```
int main()
{
    EventBus();

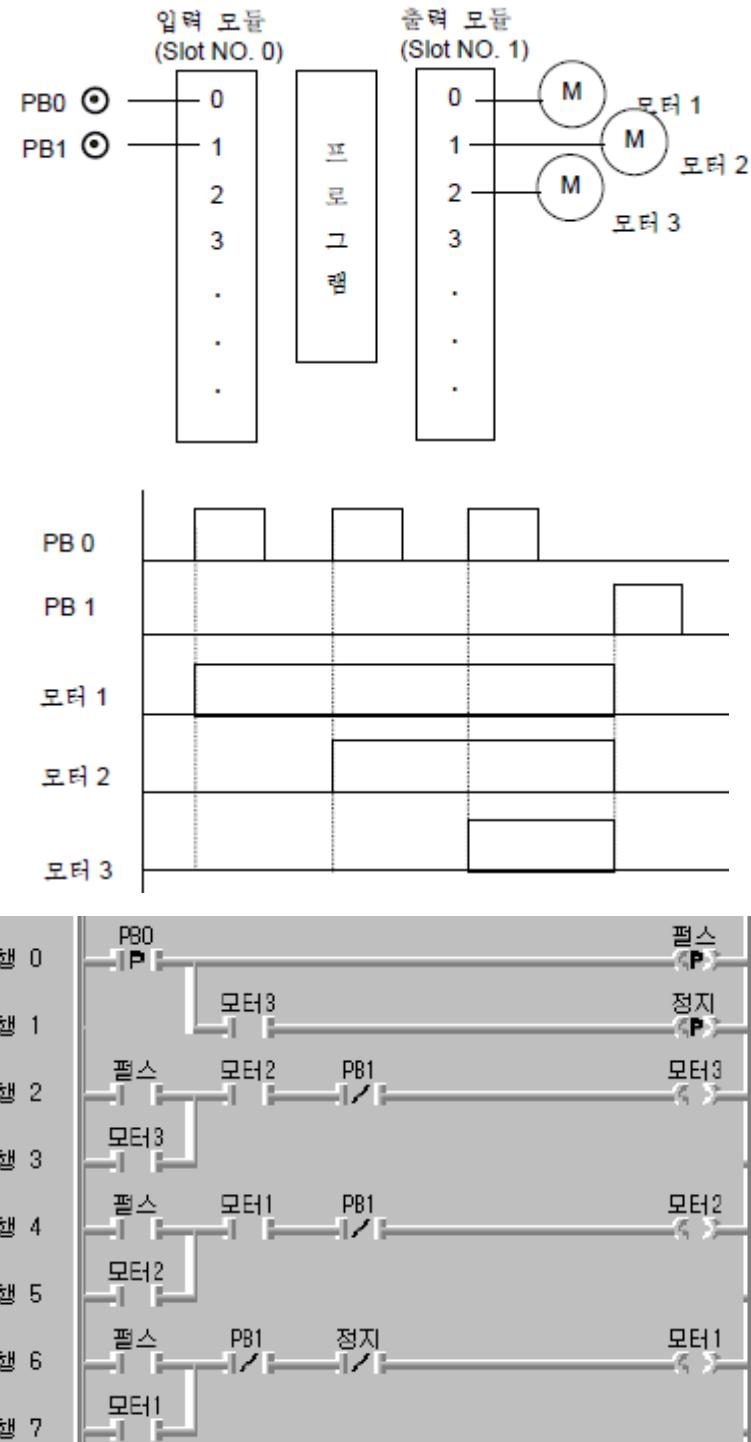
    EventProgram;
    IF (this, DioIn) {
        if (Pressed(this->in, sw0)) RY0 = !RY0;
    }

    ScanProgram;
    Eval_DioIn (FALSE); // 입력 리프레시

    End;
}
```

### 8.5.3. 기동모터 개수제어

PB0 을 한번 누르면 모터 1 이 회전하고, 한번 더 누르면 모터 2 도 회전하고, 또 한번 더 누르면 모터 3 도 모두함께 회전하며, 한번 더 누르면 모두 정지합니다. PB1 은 리셋스위치로 언제든지 전체 모터를 정지시킬 수 있습니다.



모터가 3 개이므로 3 개의 릴레이가 필요하지만, DIO 모듈에는 릴레이가 2 개 뿐이므로 DOM 모듈을 추가하겠습니다.

```

#include "ezconfig.h"
#include "ezplc.h"
#ifndef plc_ex1

#include "plc_ex1.h"
#include "dio_drv.h"

DIO* this;
DOM* dom;

#define PB0    (this->in->switch0)
#define PB1    (this->in->switch1)
#define M1     (dom->out->relay0)
#define M2     (dom->out->relay1)
#define M3     (dom->out->relay2)

LEdge ePB0, eStop;           // 에지검출용 변수

int main()
{
    dom = CreateDOM(ID_DOM1);      // 참조모듈 생성

    EventBus();

    EventProgram;

    ScanProgram;
    Edge(&ePB0, PB0);
    Edge(&eStop, PB0 && M3);

    ELogic pulse = ePB0.QP;
    ELogic stop = eStop.QP;

    M3 = (pulse || M3) && M2 && !PB1;
    M2 = (pulse || M2) && M1 && !PB1;
    M1 = (pulse || M1) && !PB1 && !stop;

    Eval_DioIn (FALSE);          // 입력 리프레시
    End;
}

```

위의 LD 를 그대로 프로그램으로 옮겼습니다. 하지만, 이해하기가 어렵습니다. 즉, 논리식으로 프로그램하면 프로그램을 이해하기가 어려운 단점이 있습니다. 이것을 C 언어 문법으로 풀어보겠습니다. (나머지는 동일하므로 main 함수부분만 기술합니다.)

```

LEdge ePB0;           // 에지용 변수

int main()
{
    dom = CreateDOM(ID_DOM1);      // 참조모듈 생성

    EventBus();

    EventProgram;

```

```

ScanProgram;
FindEdge(&ePB0, PB0);

if (PB1 == ON) {                                // PB1은
    M1 = M2 = M3 = OFF;                         // 모두 OFF시킨다
}
else if (ePB0.QP){                            // PB1=OFF 이고, PB0이 눌러지면
    if (M3)                                     // 현재 M3=ON 이면
        M1 = M2 = M3 = OFF;                     // 모두 OFF시키고
    else if (M2)                               // 현재 M2=ON 이면
        M3 = ON;                             // M3를 ON 하고
    else if (M1)                               // 현재 M1=ON 이면
        M2 = ON;                             // M2를 ON 하고
    else                                      // 현재 모두 OFF이면
        M1 = ON;                           // M1을 ON 한다.
}

Eval_DioIn(FALSE);                            // 입력 리프레시
End;
}

```

훨씬 이해가 쉬워졌습니다. 또한 컴파일러가 최적화하기 쉬우므로 생성되는 코드사이즈도 더욱 작아지고 실행속도도 빨라집니다. 그러므로 PLC 처럼 LD 만으로 프로그램하면 가독성이 떨어져 향후 유지보수도 어려워지므로 C 언어의 문법과 이벤트버스의 이벤트핸들러를 섞어서 프로그램하면 유지보수도 쉬울뿐 아니라 성능도 향상됩니다.

#### 그렇다면, 왜 PLC는 래더프로그램을 사용할까?

초기 PLC는 릴레이 제어반을 대체하기 위하여 고안되었습니다. 그러나, 릴레이 제어반 기술자들이 쉽게 사용할 수 있어야 하는데, 그들이 프로그래밍 언어를 능숙하게 사용하지 못하는 경우가 많았으므로 쉽게 사용할 수 있도록 하기 위하여 그래픽 심볼 위주의 프로그래밍 방식이 고안되었으며, 이것이 LADDER DIAGRAM입니다. 초기에는 릴레이 제어반을 대체하기만 하면 되었으므로 고기능이 필요치 않았고, 그래서 성능만 손해보면 되었습니다. 그러나, 현재의 PLC는 고기능, 고성능화 되고 있으며, 기술자들도 프로그래밍 언어를 어느정도 사용할 수 있게 되었으므로 사실 LADDER DIAGRAM 방식의 프로그래밍은 손실이 큰 방식이라 할 수 있습니다.

마지막으로 업카운터(UpCounter)를 사용해서 구현해 보겠습니다.

```

LCounter cnt;                                // 카운터용 변수

int main()
{
    dom = CreateDOM(ID_DOM1);                // 참조모듈 생성
}

```

```

EventBus();

EventProgram;

ScanProgram;
UpCounter(&cnt, PB0, PB1 || cnt.Q, 4); // 업카운터
M1 = (cnt.CV >= 1) ? ON : OFF;
M2 = (cnt.CV >= 2) ? ON : OFF;
M3 = (cnt.CV >= 3) ? ON : OFF;

Eval_DioIn(FALSE); // 입력 리프레시
End;
}

```

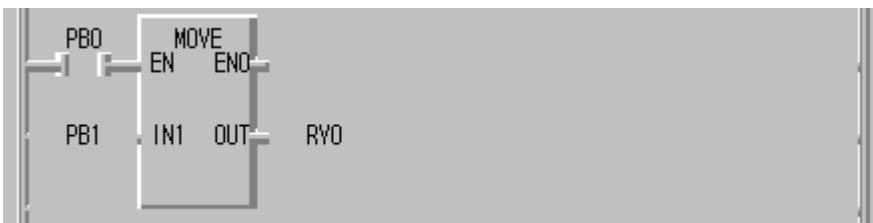
업카운터의 리셋(rst)조건이 (PB1 || cnt.Q)이므로, 리셋스위치 PB1 이 눌러지거나 업카운터의 출력(cnt.Q)이 ON 되면, 카운터가 리셋되고 cnt.CV 가 0 이 되므로 M1 = M2 = M3 = OFF; 가 됩니다.

PB0 을 한번 누르면 M1, 두번 누르면 M2, 세번 누르면 M3 가 ON 되고, 4 번 누르면 카운터의 출력(cnt.Q)이 ON 되어 카운터가 리셋되므로 M1, M2, M3 는 모두 OFF 됩니다.

어떤 방법이 가장 편리한가요? **가장 중요한 것은 “가독성”입니다.** 프로그램을 읽고 이해하기 쉬워야 합니다. 몇달 후에 다시 프로그램을 읽더라도 쉽게 이해하려면 가독성이 가장 중요합니다.

#### 8.5.4. 평선

앞에서 언급했듯이 이벤트버스에서는 평선은 별도로 제공하지 않습니다. C 언어 자체가 평선보다 훨씬 뛰어난 기능을 하기 때문입니다. 몇가지 평선을 LD 와 C 언어로 구현하여 비교해 보겠습니다.



```
if (PB0) RY0 = PB1;
```



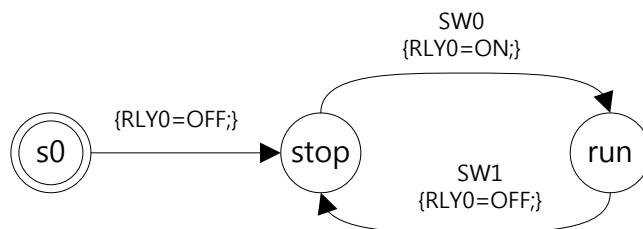
```
if (PB0) RY0 = (IN1 > IN2 && IN2 > IN3) ? ON : OFF;
```

## 9. 유한상태도(FSM)

유한상태도(Finite State Machine)는 공학에서 매우 중요한 개념입니다. FSM으로 프로그램을 작성하는 방법을 적용하여 버그없는 프로그램, 가독성이 뛰어난 프로그램을 작성하는 방법에 대하여 살펴봅니다. 유한상태도의 가장 큰 장점중의 하나는 프로그램을 상태도라는 디어그램으로 표현할 수 있고, 역으로 디어그램만 그리면 즉시 프로그램으로 옮길 수 있다는 점입니다. 그러므로, 코드를 보면서 버그를 찾는 것이 아니라 디어그램을 보면서 버그를 찾을 수 있고, 시간이 지나도 코드를 보는 것이 아니라 디어그램만 보면 프로그램의 동작을 빠르게 이해할 수 있다는 점입니다. 그러므로, 알고리즘 작성, 유지보수, 문서화 등에 매우 유리합니다.

FSM의 관점에서 프로그램은 여러개의 **상태(state)**가 있으며, 이러한 상태는 **입력조건**에 따라서 현재상태에서 다른 상태로 **전이(transition)**하는 과정으로 생각합니다. 프로그램을 상태도로 그리는 방법에는 밀리(Mealy)모델과 무어(Moore)모델이라는 두 가지 방법이 있는데, 밀리모델을 사용하는 것이 편리한 점이 많으므로 이 방법에 대하여 설명하겠습니다.

### 9.1. 예제: 상태도 구현



위의 상태도에서 원은 **상태**를 화살표는 **전이**를 화살표에 붙은 텍스트는 **입력조건**을 표시합니다. 입력조건에서 중괄호 {...}의 내용은 입력조건이 만족할 때 실행할 내용입니다. 입력조건에서 조건과 실행할 내용은 선택적입니다. 심지어 입력조건 전체가 없을 수도 있습니다.

이중원으로 그려진 *s0*는 “시작상태”입니다. 어떤 프로그램이든 시작은 반드시 있습니다. 그냥 개념상 있는 겁니다. 별 의미는 없습니다.

위의 상태도를 설명하면, 시작상태에서 조건없이 즉시 RLY0=OFF 시킨 후 stop 상태로 전이합니다. stop 상태에서 SW0이 눌러지면 RLY0=ON 시킨 후 run 상태가 되고, run 상태에서 SW1이 눌러지면 RLY0=OFF 시킨 후 stop 상태가 됩니다. 즉 SW0은 스타트스위치이고 SW1은 스톱스위치에 해당합니다.

그런데, run 상태에서 SW0을 누르면 어떻게 될까요? 위 상태도에서 run 상태에서 SW0이 눌러지는 입력조건은 없으므로 아무런 동작도 하지 않습니다.

그런데, 중요한 점은 이렇게 상태도만 완성하면 프로그램은 거의 끝난 것이나 다름없습니다. ScanProgram; 아래에 switch 문을 사용해서 아래처럼 기계적으로 상태도를 프로그램 코드로 변경만 해주면 됩니다.

```
#include "ezconfig.h"
```

```

#include "ezplc.h"
#ifndef state_diagram1

#include "state_diagram1.h"
#include "dio_drv.h"

DIO* this;

#define SW0    (this->in->sw0)
#define SW1    (this->in->sw1)
#define RLY0   (this->out->rly0)

enum {s0,stop,run} state = s0;

int main()
{
    EventBus();

    EventProgram;

    ScanProgram;
    switch (state) {
        case s0:
            RLY0 = OFF;
            state = stop;
            break;

        case stop:
            if (SW0) {
                RLY0 = ON;
                state = run;
            }
            break;

        case run:
            if (SW1) {
                RLY0 = OFF;
                state = stop;
            }
            break;
    }

    // refresh in-image
    Eval_DioIn(FALSE);
    End;
}

#endif

```

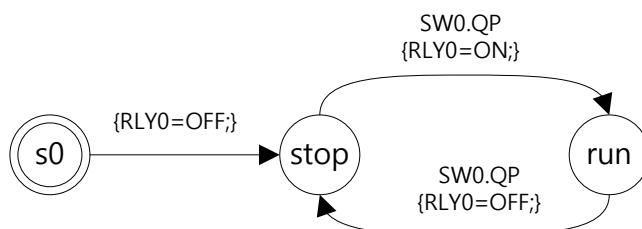
위의 프로그램은 별다른 설명을 하지 않아도 상태도와 비교해 보면 쉽게 이해할 수 있습니다. 상태도에서 사용된 3 가지 상태를 enum 형으로 정의하고, 상태를 저장할 변수 state 를 생성하고 초기상태를 s0 으로 초기화하였습니다. 그런 후 ScanProgram;에서 switch 문으로 어느 상태인지를 검사후 각 상태에서 입력될 수 있는 입력조건을 if 문으로 기술하면 됩니다.

이방식의 가장 큰 장점은 상태도 그림만 보면 프로그램의 동작을 이해할 수 있다는 것입니다. 즉, 기능이 추가되어도 프로그램을 수정하기 전에 상태도를 수정하면 버그없이 프로그램을 빠르게 수정할 수 있게됩니다.

사실, 이정도 프로그램이야 상태도를 사용하지 않아도 간단히 프로그램할 수 있습니다. 하지만, 알고리즘이 복잡해지면 유지보수, 버그, 문서화의 측면에서 상태도를 사용하는 것이 훨씬 유리합니다.

## 9.2. 예제: 상태도 확장(1)

상태도만으로 전체 프로그램을 작성하는 것보다, 상태도는 신호의 흐름을 처리하고 실제동작은 별도로 프로그램하는 것이 좋습니다. 이렇게 상태도를 확장하는 프로그래밍 방법을 살펴봅니다.



위의 상태도는 SW0 스위치 한개를 사용해서 한번 누르면 run 상태로, 다시 한번 누르면 stop 상태가 됩니다. 입력조건으로 스위치의 값(SW0)을 사용한다면 run 상태가 되는 즉시 stop 상태가 되고 다시 run 상태가 되고... 스위치를 놓을 때까지 계속 반복됩니다. 이 경우 스위치의 상승에지(SW0.QP, 간략표기로 사용했습니다)를 사용하는 것이 좋습니다. 또한 스위치의 바운싱 때문에 오동작이 발생할 수 있으므로 이를 방지하려면 디바운싱을 위한 온딜레이 타이머를 사용해서 해결할 수 있습니다.

```

#include "ezconfig.h"
#include "ezplc.h"
#define state_diagram2

#include "state_diagram2.h"
#include "dio_drv.h"

DIO* this;

#define SW0      (this->in->sw0)
#define RLY0     (this->out->rly0)

enum {s0,stop,run} state = s0;

LEdge eSW0;
LTimer tSW0;

int main()
{
    EventBus();
    EventProgram;
  
```

```

ScanProgram;
// 상태도 구현
switch (state) {
    case s0:
        RLY0 = OFF;
        state = stop;
        break;

    case stop:
        if (eSW0.QP) {
            RLY0 = ON;
            state = run;
        }
        break;

    case run:
        if (eSW0.QP) {
            RLY0 = OFF;
            state = stop;
        }
        break;
}

// 상태도 보조를 위한 코드.
// 평션블록은 상태도구현을 위한 switch문 외부에 별도로 구현한다.
OnDelay(&tSW0, SW0, 100);      // 디바운스를 위한 타이머
Edge(&eSW0, tSW0.Q);          // SW0 에지검출

// 입력 리플레시
Eval_DioIn(FALSE);
End;
}

#endif

```

에지검출을 위한 Edge(), 온딜레이를 위한 OnDelay()함수를 상태도 구현을 위한 switch 문에 넣으면 코드가 복잡해질 뿐만 아니라, 여러가지 문제가 발생할 수 있습니다. 예를들어, Edge(), OnDelay()와 같은 plc.h 에 선언된 함수들은 반복적으로 실행되어야 타이머가 계속 시간을 감시할 수 있어 정상적으로 동작하는데, 상태도에서는 현재상태에 있는 코드만이 반복실행되므로 문제가 발생할 수 있습니다.

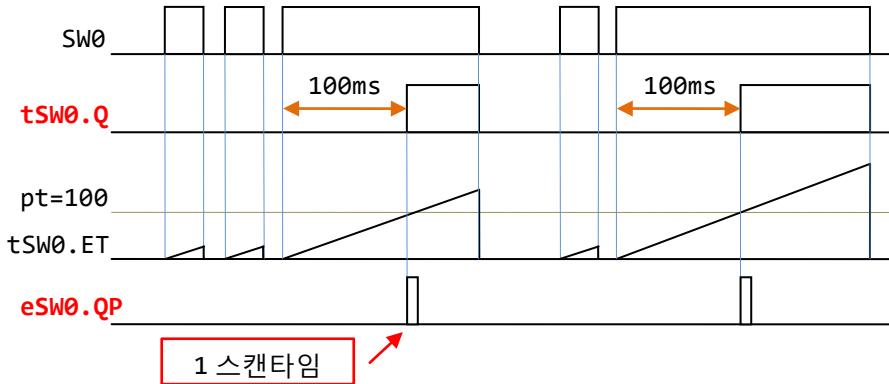
그러므로, 상태도 구현코드에서는 타이머, 카운터, 발진기와 같이 여러 스캔에 걸쳐서 실행되어야 하는 함수(평션블록이라 부름)는 사용하지 않는 것이 좋습니다. 평션블록은 상태도 외부에 별도로 구현해야 합니다.

평션블록 코드를 이해하기 쉽게 타임챠트로 그려보면 아래와 같습니다.

```

OnDelay(&tSW0, SW0, 100);      // 디바운스를 위한 타이머
Edge(&eSW0, tSW0.Q);          // SW0 에지검출

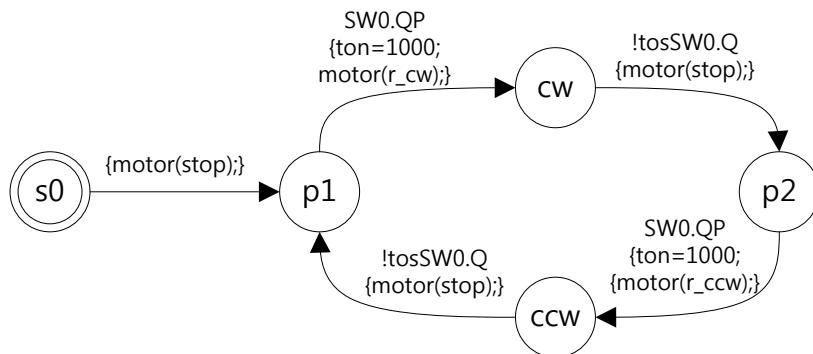
```



100ms 이하의 짧은 펄스는 OnDelay() 타이머의 출력(Q)을 활성화시킬 수 없고, 100ms 보다 긴 펄스만 디바운싱된 출력(tSW0.Q)을 발생합니다. 이 출력을 Edge()함수로 1 스캔타임동안 ON 되는 신호(eSW0.QP)를 만들어내고, 이 신호를 상태도 구현코드에서 사용하면 됩니다.

### 9.3. 예제: 상태도 확장(2)

상태도를 확장하는 두번째 예제는 액션(코일출력)이 종료되기를 기다려야 하는 경우입니다.



앞의 예제에서는

```

case stop:
  if (eSW0.QP) {
    RLY0 = ON;
    state = run;
  }
  break;
  
```

처럼 릴레이를 온시킨후 바로 다음상태로 천이하면 되는 예제였습니다. 만일, 릴레이가 오프되는 것을 확인 후 다음상태로 천이해야 한다면 어떻게 처리할 수 있을까요?

스위치를 누르면 모터가 2 위치(p1,p2)를 왔다 갔다 회전하는 예제입니다. 상태도에서 전원이 들어오면 모터를 정지시키고 p1 상태가 됩니다. p1에서 스위치를 누르면 1 초동안 모터를 정방향으로 회전합니다. cw 상태에서는 모터가 회전을 마쳤는지 확인후 p2로 천이합니다.

마찬가지로 p2에서도 스위치를 누르면 1 초동안 모터를 역방향으로 회전합니다. ccw 상태에서는 모터가 회전을 마쳤는지 확인후 p1으로 천이합니다.

이제 코드로 옮기면 아래와 같습니다. (application\exam\state\_diagram3.c 참고)

```

#include "ezconfig.h"
#include "ezplc.h"
#ifndef state_diagram3

#include "state_diagram3.h"
#include "dio_drv.h"

DIO* this;

#define SW0    (this->in->sw0)
#define RLY0   (this->out->rlly0)
#define RLY1   (this->out->rlly1)

LTimer todSW0,tosSW0; // timer on-delay for SW0, timer one-shot for SW0
LEdge eSW0;

uint32 ton;

enum {s0,p1,p2,cw,ccw} state = s0;

typedef enum {stop,r_cw,r_ccw} CMD;

void motor(CMD cmd)
{
    switch (cmd) {
    case stop: RLY0 = OFF; RLY1 = OFF; break;
    case r_cw: RLY0 = ON; RLY1 = OFF; break;
    case r_ccw: RLY0 = OFF; RLY1 = ON; break;
    }
}

int main()
{
    EventBus();

    EventProgram;

    ScanProgram;
    // 상태도 구현
    switch (state) {
        case s0:
            motor(stop);
            state = p1;
            break;

        case p1:
            if (eSW0.QP) {          // 스위치가 눌러지면
                ton = 1000;         // 1초 동안
                motor(r_cw);       // 모터를 정방향으로 회전시킨다.
                state = cw;
            }
            break;

        case cw:
            if (!tosSW0.Q) {      // 1초가 지나면

```

```

        motor(stop);           // 모터를 정지한다.
        state = p2;
    }
break;

case p2:
    if (eSW0.QP) {           // 스위치가 눌러지면
        ton = 1000;             // 1초 동안
        motor(r_ccw);          // 모터를 역방향으로 회전시킨다.
        state = ccw;
    }
break;

case ccw:
    if (!tosSW0.Q) {         // 1초가 지나면
        motor(stop);           // 모터를 정지시킨다.
        state = p1;
    }
break;
}

// 상태도 보조를 위한 코드.
// 평션블록은 상태도구현을 위한 switch문 외부에 별도로 구현한다.

OnDelay(&todSW0, SW0, 100);      // SW0 바운스제거
Edge(&eSW0, todSW0.Q);          // SW0 에지검출

OneShot(&tosSW0, ton>0, ton);     // ton 시간동안 ON되는 원샷타이머
if (!tosSW0.Q) { ton = 0; }       // 타이머 입력 초기화

// 입력이미지 리플레시
Eval_DioIn(FALSE);
End;
}
#endif

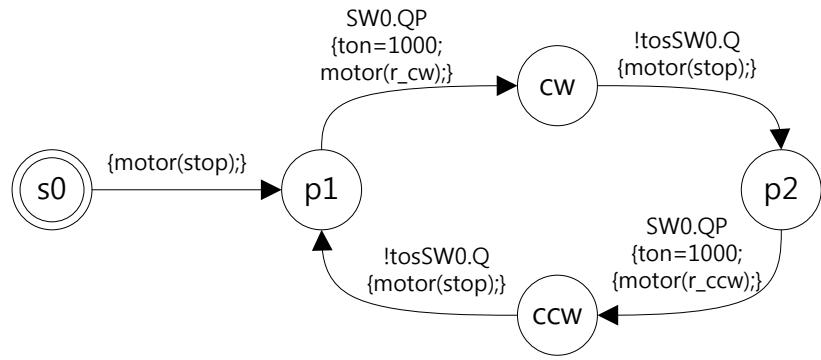
```

만 아래의 OneShot() 타이머 아래에 타이머의 입력인 ton 을 0 으로 초기화 하는 코드를 관심있게 보시기 바랍니다. 타이머가 동작을 마치면 동작조건을 해제해 주어야 합니다. 그래야 다음에 동작조건(ton>0)이 만족되면 다시 동작을 하게 됩니다.(OneShot 타이머에 대한 이전장의 설명을 다시 한번 확인하십시오)

#### 9.4. 예제: 상태도 확장(3)

---

상태도를 확장하는 세번째 예제는 액션(코일출력)이 종료되기를 기다려야 하는 경우입니다.



## 10. eventbus\_fw\_0.60 레퍼런스 가이드

`eventbus.h` 헤더파일은 사용자가 사용할 수 있는 유용한 여러 기능을 제공한다. 여기서 제공되는 기능을 잘 이해하면 이벤트버스를 좀 더 다양한 방법으로 응용할 수 있다.

### 10.1. 기본 데이터형

펌웨어에 가장 많이 사용되는 기본 데이터형

| 데이터타입               | 의미                              | 바이트 수 | 표현 숫자 범위                                                |
|---------------------|---------------------------------|-------|---------------------------------------------------------|
| <code>uint8</code>  | <code>unsigned char</code>      | 1     | 0 ~ 255                                                 |
| <code>int8</code>   | <code>signed char</code>        | 1     | -128 ~ 127                                              |
| <code>uint16</code> | <code>unsigned short</code>     | 2     | 0 ~ 65,535                                              |
| <code>int16</code>  | <code>signed short</code>       | 2     | -32,768 ~ 32,767                                        |
| <code>uint32</code> | <code>unsigned long</code>      | 4     | 0 ~ 4,294,967,295                                       |
| <code>int32</code>  | <code>signed long</code>        | 4     | -2,147,483,648 ~ 2,147,483,647                          |
| <code>uint64</code> | <code>unsigned long long</code> | 8     | 0 ~ 18,446,744,073,709,551,615                          |
| <code>int64</code>  | <code>signed long long</code>   | 8     | -9,223,372,036,854,775,806 ~ +9,223,372,036,854,775,807 |
| <code>float</code>  | <code>float</code>              | 4     | $8.4 \times 10^{-37} \sim 3.4 \times 10^{38}$           |

### 10.2. CLOCK 타입

```
| typedef uint64 CLOCK;
```

시스템 내부의 밀리초(ms) 단위의 시계에 해당하는 `clock`을 위한 데이터타입이다.

### 10.3. EEdge 열거형

```
| typedef enum {FALL, RISE, BOTH, UNDEF} EEdge;
```

스위치, 릴레이, 상태플래그 등과 같은 온오프 타입의 값이 변하는 순간을 표현하는 상수명칭이다.

- FALL : ON에서 OFF로 변하는 하강에지
- RISE : OFF에서 ON으로 변하는 상승에지
- BOTH : 하강 또는 상승에지
- UNDEF : 결정되지 않은 에지

### 10.4. ELogic 열거형

```
| typedef enum {LOW = 0, HIGH = 1, OFF = LOW, ON = HIGH} ELogic;
```

스위치, 릴레이, 상태플래그 등과 같은 온오프 타입의 값을 표현하는 상수명칭이다.

- LOW = OFF = FALSE = 0  
HIGH = ON = TRUE = 1
- 아래는 모두 동일한 표현입니다.  
`if (this->in->sw0 == ON) ...`

```
if (this->in->sw0 == HIGH) ...
if (this->in->sw0) ...
```

## 10.5. EActiveLogic 열거형

```
typedef enum {ACTIVE_LOW = LOW, ACTIVE_HIGH = !ACTIVE_LOW} EActiveLogic;
```

스위치, 센서, 릴레이 등과 같은 온오프 타입의 경우 마이컴 포트로 입출력되는 값과 실제 상태가 반대인 경우가 있다. 예를 들면, DIM 모듈의 경우 IO 단자에 7[V]이상의 전압이 연결되면 스위치가 ON으로 인식되지만, 하드웨어에서 반전되어 마이컴에는 LOW인 신호가 입력된다. 즉, 액티브 상태에서 마이컴으로 LOW가 입력되는 경우를 ACTIVE\_LOW라고 하고 HIGH가 입력되는 경우를 ACTIVE\_HIGH라고 한다. 입력이 반대인 경우(ACTIVE\_LOW) 혼돈스러우므로 이를 조정해서 입력값과 읽은값이 동일하게 인식하도록 설정할 때 사용하는 상수명칭이다.

- eventbus\_fw\_0.60\eventbus\include\ezgpio.h에는 **GPIO\_ReadInputPins()**가 있다. port, pins로 지정된 입력포트를 읽어서 logic으로 읽은 값을 조정한다.  
`uint32 GPIO_ReadInputPins (uint8 port, uint32 pins, EActiveLogic logic);`
- eventbus\_fw\_0.60\driver\dim\_drv.c 드라이버의 **Eval\_DimIn()**에는 IO 접점입력을 읽기 위하여 위의 **GPIO\_ReadInputPins()**를 사용한다. IO\_DIN\_L로 선언된 마이컴 포트에서 값을 읽어서 ACTIVE\_LOW로직이므로 반전시킨 후 din5\_0에 저장한다. 결국, 스위치가 눌러진(또는, 7V 이상의 전압이 가해진) 입력은 해당비트가 '1'이 된다.  
`uint16 din5_0 = (uint16)GPIO_ReadInputPins(IO_DIN_L, ACTIVE_LOW);`

## 10.6. SMD\_CMD, SMD\_DIR, SMD\_STEP, MOC\_CMD, MOC\_DIR, USD\_CMD, USD\_DIR, USD\_STEP 열거형

```
typedef enum {FREE, HOLD, STOP, RUN, MOVE} SMD_CMD, MOC_CMD, USD_CMD;
typedef enum {CW, CCW} SMD_DIR, MOC_DIR, USD_DIR;
typedef enum {SIXTEENTH, QUARTER, EIGHTH, HALF} SMD_STEP;
typedef enum {USD_FULL=1, USD_HALF=3, USD_QUARTER=4, USD_EIGHTH=5,
USD_SIXTEENTH=6} USD_STEP;
```

SMD, MOC, USD 모듈에서 사용되는 동작명령, 회전방향, 마이크로스텝 설정을 위한 열거형이다.

- FREE : 모터회전을 정지(SMD, USD의 경우 모터전류차단)하는 명령
- HOLD : 모터회전을 정지(SMD, USD의 경우 모터 홀딩전류 유지)하는 명령
- STOP : 모터회전 감속후 정지명령
- RUN : 모터회전 가속명령
- MOVE : 일정거리(펄스) 이동 명령

- CW : 정방향 회전, (MOC 의 경우 DIR 출력이 HIGH)  
CCW : 역방향 회전, (MOC 의 경우 DIR 출력이 LOW)
- SIXTEENTH, USD\_SIXTEENTH : SMD, USD 모듈에서 16 분주 회전  
QUARTER, USD\_QUARTER : SMD, USD 모듈에서 4 분주 회전  
EIGHTH, USD\_EIGHTH : SMD, USD 모듈에서 8 분주 회전  
HALF, USD\_HALF : SMD, USD 모듈에서 2 분주 회전  
USD\_FULL : USD 모듈에서 1 분주 회전

## 10.7. Ascii 열거형

```
typedef enum {Asc_SOF = 0x02, Asc_DEL = 0x7f, Asc_SPACE = 0x20, Asc_CR = 0x0D,
Asc_BS = 0x08} Ascii;
```

ASCII 문자 열거형이다.

## 10.8. TCType 열거형

```
typedef enum {K_TYPE, T_TYPE, E_TYPE } TCType;
```

TCM 모듈에서 써모커플 타입을 의미하는 열거형이다.

## 10.9. ESender 열거형

```
typedef enum {
    UART,
    I2C,
    HW,
    APPL,
    NOSYNC,
    ENET,
    END_OF_SENDER,
} ESender;
```

이벤트가 발생한 곳을 가리키는 열거형이다. UART, I2C 는 통신으로 수신한 이벤트, HW 는 타이머나 IO 포트와 같은 하드웨어 장치에서 발생한 이벤트, APPL 은 프로그램에서 사용자가 발생시킨 이벤트, NOSYNC 는 이미지를 연결된 다른 모듈이나 PC 로 전송하는 것을 막아서 이미지 동기화를 하지 않겠다는 사실을 알리는 속성이다.

아래 코드는 DioIn 이벤트핸들러에서 이벤트의 sender 속성을 NOSYNC 로 변경하였으므로, evetnbus 는 DioIn 이벤트를 다른 모듈에게 전송하지 않으며, 단지 모듈 내부에서만 사용된다. 이 방법을 사용하면 빈번하게 발생하는 이벤트가 통신에 부하를 주는 것을 막을 수 있다. 즉, 빈번하게 DioIn 이벤트가 발생하는 경우, 조건에 맞는 경우에만 이벤트가 전송되도록 할 수 있다.

\*주의\* DioIn 이벤트를 수신하지 못한 모듈은 입력이미지가 동기화되지 않으므로 입력이미지가 실제 입력과 다른 값을 가지게 되고, DioOut 이벤트를 수신하지 못한 모듈은 출력이미지가

동기화 되지 않으므로 출력이미지가 실제 출력과 다른 값을 가지게 되므로 NOSYNC 는 매우 주의하여 사용하여야 한다.

```
Event(this, DioIn) {
    if (this->in->sw0 == pe->sw0) {      // switch0에 변화가 없으면,
        pe->e.sender = NOSYNC;            // 이벤트를 타모듈에게 전송하지 않음
    }
}
```

이 방법은 매우 유용하게 사용될 수 있다. 스위치가 3 회 눌러질 때마다 릴레이를 반전시키는 프로그램을 예로 들어보면, 매번 스위치가 ON 될 때와 OFF 될 때 마다 DioIn 이벤트가 발생하고, 버스케이블을 통하여 연결된 모듈들에게 이벤트가 전송된다. 이것은 통신버스에 부하를 가중시킨다. 관심있는 것은 3 회 눌러진 시점이므로 아래와 같이 pe->e.sender = NOSYNC; 을 사용하면 이벤트는 전송되지 않으며 다른 동작은 모두 그대로이다. 스위치가 3 회 온/오프될 때 이벤트는 6 회 발생되고 전송되지만, 이 방법을 사용하면 이벤트가 전송되지 않아서 통신부하는 감소하고 실행속도는 빨라지므로 성능이 향상된다.

```
// application\exam\dio_counter.c 참고
int main()
{
    EventBus();

    int count = 0;

    EventProgram;
    IF (this, DioIn) {
        if (Pressed(this->in, sw0)) {          // 상승에지 검출
            count++;                          // 카운터 증가

            if (count == 3) {                  // 스위치가 3회 눌러지면
                count = 0;                    // 카운터 초기화
                this->out->rly0 = !this->out->rly0; // 릴레이반전
            }

            pe->e.sender = NOSYNC;           // 이미지동기화 않음
        }
    }

    ScanProgram;
    Eval_DioIn (FALSE);                   // 스위치입력 검사

    End;
}
```

그리고, 주의할 점은 입력이미지 동기화가 이루어지지 않으므로, 만일 SMD 모듈이 이 DIO 모듈의 입출력이미지를 생성하고 참조한다면, 입력이미지는 서로 동기화 되지 않으므로, SMD 에 있는 DIO 의 입력이미지(dio->in)는 신뢰할 수 없다는 점을 명심해야 한다.

즉, DIO 모듈에서 DioIn 이벤트를 전송하지 않는다면, DIO 모듈을 참조하는 다른 모듈은 DIO 의 입력이미지(dio->in)를 사용하지 않는 것이 안전하다. 이것은 출력이미지의 경우도 마찬가지이다. 그러므로, DioIn/DioOut 이벤트의 전송을 막는 NOSYNC 는 매우 주의하여 사용하는 것이 좋으며, 이 경우 다른 모듈과의 데이터 교환은 사용자이벤트를 사용하는 것이 안전하다.

DioOut 이벤트의 경우에도 NOSYNC 를 사용할 수 있으며, 이 경우에도 이벤트를 전송하지 않으므로 동기화를 위한 이벤트 전송은 하지 않는다.

### 10.10. EEventTag 열거형

```
typedef enum {
    AnyTag,

    // Local Events
    // these events are not transmitted on the serial bus
    Obsolete,           // CharReceived is obsoleted,
    CyclicTimeout,
    Delayed,

    // Global Events
    // these events are transmitted on the serial bus (to end)
    StartOfSystemEventTag = 0x20,

    InChanged = StartOfSystemEventTag,
    AimIn = InChanged,
    AomIn = InChanged,
    DimIn = InChanged,
    DioIn = InChanged,
    DomIn = InChanged,
    EnvIn = InChanged,
    SegIn = InChanged,
    SmdIn = InChanged,
    UsbIn = InChanged,
    MocIn = InChanged,
    TcmIn = InChanged,
    LanIn = InChanged,
    DieIn = InChanged,
    DoeIn = InChanged,
    MsbIn = InChanged,
    UsdIn = InChanged,
    LmoIn = InChanged,
    AihIn = InChanged,
    RcsIn = InChanged,
    CanIn = InChanged,

    OutChanged,
    AimOut = OutChanged,
    AomOut = OutChanged,
    DimOut = OutChanged,
    DioOut = OutChanged,
    DomOut = OutChanged,
    EnvOut = OutChanged,
    SegOut = OutChanged,
```

```
SmdOut = OutChanged,  
UsbOut = OutChanged,  
MocOut = OutChanged,  
TcmOut = OutChanged,  
LanOut = OutChanged,  
DieOut = OutChanged,  
DoeOut = OutChanged,  
MsbOut = OutChanged,  
UsdOut = OutChanged,  
LmoOut = OutChanged,  
AihOut = OutChanged,  
RcsOut = OutChanged,  
CanOut = OutChanged,  
  
ParamRequested,  
  
ParamChanged,  
AimParamChanged = ParamChanged,  
AomParamChanged = ParamChanged,  
DimParamChanged = ParamChanged,  
DioParamChanged = ParamChanged,  
DomParamChanged = ParamChanged,  
EnvParamChanged = ParamChanged,  
SegParamChanged = ParamChanged,  
SmdParamChanged = ParamChanged,  
UsbParamChanged = ParamChanged,  
MocParamChanged = ParamChanged,  
TcmParamChanged = ParamChanged,  
LanParamChanged = ParamChanged,  
DieParamChanged = ParamChanged,  
DoeParamChanged = ParamChanged,  
MsbParamChanged = ParamChanged,  
UsdParamChanged = ParamChanged,  
LmoParamChanged = ParamChanged,  
AihParamChanged = ParamChanged,  
RcsParamChanged = ParamChanged,  
CanParamChanged = ParamChanged,  
  
ParamNotified,  
AimParamNotified = ParamNotified,  
AomParamNotified = ParamNotified,  
DimParamNotified = ParamNotified,  
DioParamNotified = ParamNotified,  
DomParamNotified = ParamNotified,  
EnvParamNotified = ParamNotified,  
SegParamNotified = ParamNotified,  
SmdParamNotified = ParamNotified,  
UsbParamNotified = ParamNotified,  
MocParamNotified = ParamNotified,  
TcmParamNotified = ParamNotified,  
LanParamNotified = ParamNotified,  
DieParamNotified = ParamNotified,  
DoeParamNotified = ParamNotified,  
MsbParamNotified = ParamNotified,  
UsdParamNotified = ParamNotified,  
LmoParamNotified = ParamNotified,  
AihParamNotified = ParamNotified,
```

```

RcsParamNotified = ParamNotified,
CanParamNotified = ParamNotified,

UsbConnected,
UsbDisconnected,
SyncRequested,
AtCommand,
AtResponse,
SoftReset,

// User Responsible Events (to end)
// if need, user have to transmit this events on the serial bus

// Event made by system
StartOfDiagEventTag = 0x40,
Vaa = StartOfDiagEventTag,           // VaaChanged
Vbb,                                // VbbChanged
Vdc,                                // VdcChanged

// MSB UserTag
MsbTransmit,                         // Request to transmit a Packet
MsbReceived,                          // A Packet is received from communication
reserved1,
reserved2,

// Modbus UserTag
ModbusFrameReceived,
reserved3,
reserved4,

// the first value of user defined event (see ezconfig.h)
StartOfUserEventTag = 0x60,
} EEventTag;

```

Eventbus\_fw 에서 기본적으로 제공하는 시스템이벤트는 그리 많지 않으며, 또한, 대부분의 이벤트는 자동으로 처리되므로 사용자가 별도의 이벤트핸들러를 작성할 필요가 없다. 사용자는 단지 관심있는 이벤트에 대한 핸들러만 작성하면 된다.

이벤트를 분류하면,

- 지역이벤트, Local Event  
모듈내부에서만 사용할 수 있으며, 통신버스로 다른 모듈에게는 전달되지 않는 이벤트.
- 전역이벤트, Global Event  
모듈내부에서 뿐만 아니라, 통신버스로 다른모듈에게도 전송되는 이벤트.
- 사용자 위임 이벤트, User Responsible Event  
Eventbus\_fw 프레임워크에서는 별도의 처리는 하지 않으며, 사용자가 필요에 따라 생성해서 사용할 수 있는 이벤트.
  - StartOfDiagEventTag = 0x40 부터는 시스템에서 제공하는 이벤트.
  - StartOfUserEventTag = 0x60 부터는 사용자가 만들어 쓰는 유저이벤트.

**\*중요\*** 이벤트는 구조체 데이터타입이며, 타입명칭은 태그명칭 뒤에 “Event”를 붙여 사용하도록 약속한다. 예를 들면, DioIn 태그에 대한 이벤트는 “DioInEvent”가 된다. 만일 사용자가 새로운 이벤트를 만들어 사용하려 한다면, 태그와 이벤트를 정의해야 한다. 태그(일례로 SetVref)는 ezconfig.h 의 USER\_EVENT\_TAG 열거형에 추가하고, 이벤트는 그 아래에 #pragma pack(push,1) ~ #pragma pack(pop); 사이에 추가하면 된다. 이때 이벤트 명칭은 태그에 “Event”를 붙여서 SetVrefEvent 로 해야한다.

- 이벤트에 사용할 데이터가 없으면, 기본타입인 Event 타입을 재정의해야 한다.

```
typedef Event SetRelayEvent; // SetRelay 태그의 이벤트
```

- 이벤트에 사용할 데이터가 있으면, 첫 항목에 EventHead 를 반드시 추가해야 한다.

```
typedef struct { // SetVref 이벤트
    EventHead e;
    float     vref;
    float     hyst;
} SetVrefEvent;
```

```
typedef struct {
    __POOL;
    ESender   sender;
    ID        id;
    uint8     size;
    uinst8    crc;
    EEventTag tag;
} EventHead;
```

EventHead 에는 이벤트 관리용 데이터가 있는데, 이벤트를 발생시킨 곳을 알리는 sender, 어느 모듈의 이벤트인지 나타내는 id, 이벤트의 길이 size, 이벤트의 체크섬인 crc, 이벤트의 종류 tag 로 구성된다.

이벤트를 함수와 비교하면 쉽게 이해할 수 있다. tag 는 함수명에 해당하고, 이벤트 데이터 vref, hyst 는 함수인자에 해당한다. 위의 SetVrefEvent 를 전송한다는 것은 SetVref(float vref, float hyst); 함수를 호출하는 것과 유사하다고 이해해도 좋다.

각각의 태그와 해당하는 이벤트의 구조체를 살펴보면 다음과 같다.

|                                                                                        |                                                                                                                    |
|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| 태그 : <i>CyclicTimeout</i><br>일정시간 간격으로 반복 발생하며,<br>발생주기는 ezconfig.exe 에서 설정한 Tp<br>이다. | <pre>typedef struct {     EventHead e;     uint64    ms; } TimeEvent;  typedef TimeEvent CyclicTimeoutEvent;</pre> |
|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|

|                                      |                                                                                                                   |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| 태그 : <i>Delayed</i><br>시간지연 이벤트용 태그. | <pre>typedef struct {     EventHead e;     uint64    delayTime;     Event*    delayedEvent; } DelayedEvent;</pre> |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------|

태그 : *MsbTransmit, MsbReceived*  
MSB 모듈의 송신 및 수신 이벤트.

```
typedef struct {
    EventHead e;
    uint8 chan;
    uint8 len;
    uint8 packet[MSB_PACKET_SIZE];
} MsbTransmitEvent, MsbReceivedEvent;
```

태그 : *XxxIn*  
*AimIn = InChanged,*  
*AomIn = InChanged,*  
*DimIn = InChanged,*  
*DioIn = InChanged,*  
*DomIn = InChanged,*  
*EnvIn = InChanged,*  
*SegIn = InChanged,*  
*SmdIn = InChanged,*  
*UsbIn = InChanged,*  
*MocIn = InChanged,*  
*TcmIn = InChanged,*  
*LanIn = InChanged,*  
*DieIn = InChanged,*  
*DoeIn = InChanged,*  
*MsbIn = InChanged,*  
*UsdIn = InChanged,*  
*LmoIn = InChanged,*  
*AihIn = InChanged,*  
*RcsIn = InChanged,*  
*CanIn = InChanged,*

모듈의 입력에 변동이 있을 때 발생.

DIO 모듈의 경우 :

```
typedef struct {
    EventHead e;
    union {
        uint8 sw;
        struct {
            ELogic sw0 :1;
            ELogic sw1 :1;
            ELogic sw2 :1;
            ELogic sw3 :1;
            ELogic sw4 :1;
            ELogic sw5 :1;
        };
    };
} DioInEvent;
```

태그 : *XxxOut*  
*AimOut = OutChanged,*  
*AomOut = OutChanged,*  
*DimOut = OutChanged,*  
*DioOut = OutChanged,*  
*DomOut = OutChanged,*  
*EnvOut = OutChanged,*  
*SegOut = OutChanged,*  
*SmdOut = OutChanged,*  
*UsbOut = OutChanged,*  
*MocOut = OutChanged,*  
*TcmOut = OutChanged,*  
*LanOut = OutChanged,*  
*DieOut = OutChanged,*  
*DoeOut = OutChanged,*  
*MsbOut = OutChanged,*  
*UsdOut = OutChanged,*  
*LmoOut = OutChanged,*  
*AihOut = OutChanged,*  
*RcsOut = OutChanged,*  
*CanOut = OutChanged,*

모듈의 출력을 변경하기 위하여 발생.

DIO 모듈의 경우 :

```
typedef struct {
    EventHead e;
    union {
        uint8 rly;
        struct {
            ELogic rly0 :1;
            ELogic rly1 :1;
        };
    };
} DioOutEvent;
```

태그 : *UsbConnected*

PC 에서 UART 에 연결을 요청하는 이벤트.  
연결되면 PC 와 모듈은 서로 이벤트를  
송수신할 수 있다.

```
typedef Event UsbConnectedEvent;
```

태그 : *UsbDisconnected*

UART 연결해제를 요청하는 이벤트. 해제되면  
PC 와 모듈은 서로 이벤트를 송수신할 수  
없다.

```
typedef Event UsbDisconnectedEvent;
```

태그 : *SyncRequested*

모듈에게 모든 이미지를 전송할 것을 요청함.  
모듈은 입력이미지, 출력이미지, 버스전압  
등의 데이터를 강제로 전송함.

```
typedef Event SyncRequestedEvent;
```

태그 : *AtCommand, AtResponse*  
AT 명령을 송수신할 때 사용되는  
이벤트.

```
typedef struct {
    EventHead e;
    ComSrc src;
    char atCmd[AT_CMD_MAX_LENGTH+1];
} AtCommandEvent, AtResponseEvent;
```

태그 : *SoftReset*

버스에 연결된 특정 모듈을 초기화하는  
용도로 사용되는 이벤트

```
typedef Event SoftResetEvent;
```

태그: *Vaa, Vbb, Vdc*

VAA, VBB, VDC 전압이 변동하면  
발생하는 이벤트. V0.50 부터  
옵션으로 바뀜.

```
typedef struct {
    EventHead e;
    float volt;
} VoltEvent;

typedef VoltEvent VaaEvent, VbbEvent, VdcEvent;
```

태그: *MsbTransmit, MsbReceived*  
MSB 에서 패킷전송을 요청하거나,  
패킷수신을 알리는 이벤트

```
typedef struct {
    EventHead e;
    uint8 chan;
    uint8 len;
    uint8 packet[MSB_PACKET_SIZE];
} MsbTransmitEvent, MsbReceivedEvent;
```

### 10.11. MType 열거형

```
typedef enum {
    Aim = 1,      // Analog Input Module
    Aom,          // Analog Output Module
    Dim,          // Digital Input Module
    Dio,          // Digital Input Output Module
    Dom,          // Digital Output Module
    Env,          // ENVIRONMENT Sensor Module
    Seg,          // Segment-16 Module
    Smd,          // Step Motor Driver Module
    Usb,          // USB to PC link Module
    Moc,          // Motion Control Module
    Tcm,          // Thermocouple Module
    Lan,          // Ethernet Module
    Die,          // Digital Input Expanded
    Doe,          // Digital Output Expanded
    Msb,          // Multiport Serial Bridge
    Usd,          // Unipolar Stepper Driver
    Lmo,          // LAN MODBUS bridge module
    Aih,          // Analog Input High-resolution
    Rcs,          // RC Servo motor controller
    Can,          // CAN Communication module
    Pc = 120,
} MType;
```

모듈의 종류를 나타내는 상수 열거값이다. 통신주소인 ID 와는 다른 의미인데, 예를 들어, 두개의 DIO 모듈을 사용한다면, ID 는 서로달라야 하므로 한개는 4 번 다른 한개는 5 번을 사용해야 하지만, 두 모듈은 모두 DIO 모듈이므로 MType 은 Dio(4 번)으로 동일하다.

### 10.12. EventHead 구조체

```
typedef struct {
    __POOL;
    ESender    sender;
    ID         id;
    uint8      size;        // from id to end
    uint8      crc;
    EEventTag  tag;
} EventHead;
```

이벤트를 선언할 때 사용되는 기본 데이터를 모아둔 구조체. 모든 이벤트는 반드시 EventHead 구조체를 가장 앞에 두어야 한다.

### 10.13. Event 구조체

```
typedef struct {
    EventHead  e;
} Event;
```

전달 인자가 없는 텅빈 이벤트이다. 함수에 비유하면 `tag` 가 함수명이고, 함수의 전달인자는 없는(`void`) 경우에 해당한다. 달리 전달인자가 없는 모든 이벤트들은 `Event` 를 재정의하여 사용하면 된다.

#### 10.14. CyclicTimeoutEvent

```
typedef struct {
    EventHead e;
    uint64 ms;
} TimeEvent;

typedef TimeEvent CyclicTimeoutEvent;
```

`ezconfig.exe` 에서 설정한 `Tp` 시간마다 (디폴트 10ms) 발생하는 이벤트. `ms` 는 이벤트 발생시간.

#### 10.15. VaaEvent, VbbEvent, VdcEvent

```
typedef struct {
    EventHead e;
    float volt;
} VoltEvent;

typedef VoltEvent VaaEvent, VbbEvent, VdcEvent;
```

버스전압에 변동이 있을 때 발생하는 이벤트(드라이버 파일 참고).

#### 10.16. DelayedEvent

```
typedef struct {
    EventHead e;
    uint64 delayTime;
    Event* delayedEvent;
} DelayedEvent;
```

시간지연 이벤트. `Application\delayed_event.c` 참고. 첨부된 이벤트 `delayedEvent` 를 지연시간 `delayTime` 만큼 지난후에 실행시키도록 시간을 지연시킨다.

```
UserEvent(e,DioOut);
e->rly0 = ON;
SendDelayedEvent(1000,e);
```

위 코드는 `DioOut` 이벤트를 만들고, `relay0` 을 `ON` 시킨후 1000ms 후에 실행되도록 시간지연 이벤트를 발생(이벤트큐에 삽입) 시킨다.

\*중요\* 시간지연 이벤트는 지연시간을 이용하는 매우 유익한 방법이다. 스위치가 `ON` 되면 10초후에 1초동안 릴레이를 `ON` 시키도록 하려면 시간지연 이벤트를 사용하면 매우 쉽게 구현할 수 있다.

### 10.17. MsbTransmitEvent, MsbReceivedEvent

```
typedef struct {
    EventHead e;
    uint8     chan;
    uint8     len;
    uint8     packet[MSB_PACKET_SIZE];
} MsbTransmitEvent, MsbReceivedEvent;
```

MSB 모듈에서 패킷을 전송하거나 패킷수신을 알리기 위하여 사용되는 이벤트의 구조체. 길이가 MSB\_PACKET\_SIZE(24 바이트) 보다 더 긴 경우 24 바이트씩 나누어서 여러 패킷으로 전송된다.

```
UserEvent(pte,MsbTransmit);           // 이벤트 생성
pte->chan = 0;                        // 송신할 포트
pte->len = 16;                         // 데이터 길이
memcpy(pte->packet, "abcdefghijklmnp", 16); // 송신데이터 저장
SendEvent(pte);                       // 이벤트 전송
```

### 10.18. XxxInEvent : 입력이미지

```
typedef struct {
    EventHead e;
    float     vin[4];
} AimInEvent;
```

Eventbus\_fw 는 모듈의 입력을 XxxIn 구조체에 저장한다(Xxx 는 모듈명). 그러므로 입력을 직접 읽을 필요없이 이 구조체에서 값을 읽으면 간단히 하드웨어의 입력을 읽을 수 있다. 위의 코드는 AIM 모듈의 입력이미지이며, EventHead 가 있으므로 입력이미지가 바로 이벤트임을 알 수 있다. 그리고, 입력된 전압값을 vin[0]~vin[3]에 저장한다.

### 10.19. XxxOutEvent : 출력이미지

```
typedef struct {
    EventHead e;
    float     vout[2];
    float     iout[2];
} AomOutEvent;
```

Eventbus\_fw 는 모듈의 출력을 XxxOut 구조체에 저장한다(Xxx 는 모듈명). 그러므로 출력중인 값을 알고 싶을 때 이 구조체에서 값을 읽으면 된다. 위의 코드는 AOM 모듈의 출력이미지이며, EventHead 가 있으므로 출력이미지가 바로 이벤트임을 알 수 있다. 그리고 출력중인 전압은 vout[0],[1]에 전류는 iout[0],[1]에 저장된다.

모듈의 출력을 변경하려면, XxxOut 이벤트에 출력할 값을 설정후 이벤트를 발생시키면(이벤트큐에 넣으면) 된다.

### 10.20. AIM, AOM, DIE, DIM, DIO... : 모듈 이미지

```
typedef struct {
    MType     mtype;
    ID        id;
    DIO_IN*   in;
    DIO_OUT*  out;
} DIO;
```

모듈의 모든 데이터를 모은 모듈이미지 구조체이다. 어느 모듈을 참조하기 위하여는 그 모듈의 이미지를 생성해야 한다. 예를 들어, 현재 DIO 모듈에서 프로그램을 작성하고 있으며, 프로그램에서 DIM 모듈을 참조하여야 한다면, 아래와 같이 DIM 모듈의 이미지를 생성하여야 한다. 생성함수는 CreateXXX(ID) 이다.

```
DIM* dim = CreateDIM(ID_DIM1); // 참조하는 모듈은 반드시 생성해야 한다.
```

### 10.21. EventQ\_Put() 함수

```
extern Event* EventQ_Put(void* pe);
```

이벤트를 이벤트큐에 삽입한다. 삽입된 이벤트는 향후 이벤트핸들러 메시지루프(EventProgram ~ End)에서 각각의 이벤트핸들러에 건네진다. SendEvent(void\* event); 역시 이벤트큐에 삽입하지만 event 가 NULL 인지 검사하므로 더욱 안정성이 좋다.

```
DioInEvent* dio_in = CreateEvent(DioIn); // 이벤트 생성
dio_in->sw = 0b000111; // 이벤트 전달인자 설정
EventQ_Put(dio_in); // SendEvent(dio_in); 이벤트큐에 삽입
```

### 10.22. EventQ\_Get() 함수

```
extern Event* EventQ_Get();
```

이벤트큐에서 다음 실행할 이벤트를 가져온다. 이벤트가 없으면 NULL 을 리턴한다. 다음은 이벤트큐에서 이벤트를 읽어오는 예이다.

```
pe = EventQ_Get(); // dio_ex1.c 참고
```

### 10.23. Event\_Sync() 함수

```
extern void Event_Sync(Event* pe);
```

이벤트핸들러에서 사용을 마친 이벤트의 후처리를 담당한다. 매크로 ScanProgram 에서 자동으로 호출된다.

### 10.24. Event\_Is() 함수

```
extern Bool Event_Is(Event* pe, void* pimg, EEventTag tag);
```

이벤트 pe 가 원하는 모듈(pimg)에 대한 이벤트(tag)인지 비교한다. 일치하면 TRUE 를 리턴한다. IF(module, event\_tag) 매크로 참조. 아래 코드는 이벤트(pe)가 this 모듈에 대한 *CyclicTimeout* 이벤트 인지 비교한다.

```
IF (this, CyclicTimeout) ...
```

## 10.25. Event\_SetDefault() 함수

```
extern void* Event_SetDefault(void* pe, EEventTag tag, ID id, int size);
```

이벤트 pe 를 tag, id, size 로 초기화 한다. 초기화된 이벤트를 리턴한다.

## 10.26. CreateEvent(tag) 매크로함수

```
#define CreateEvent(tag) \
    (tag##Event*)Event_SetDefault(ez_malloc_(sizeof(tag##Event)), tag, \
my_param->e.id, sizeof(tag##Event) - INDEX_OF_ID)
```

새로운 이벤트를 생성하는 매크로함수이다. 태그명칭(tag)만으로 기본값으로 초기화된 이벤트를 메모리풀에 생성하고 그 주소를 리턴한다. 이벤트의 EventHead 에 sender 는 “APPL”로 id 는 자신의 ID 로(param.id) 초기화 한다.

```
|| UserDefinedEvent* ude = CreateEvent(UserDefined); // 사용자 이벤트 생성
|| ude->ms = clock; // 이벤트 멤버 설정
|| SendEvent(ude); // 이벤트 전송
```

## 10.27. UserEvent(event\_var, tag) 매크로함수

```
#define UserEvent(event_var, tag) \
    tag##Event* event_var = CreateEvent(tag)
```

“10.26 CreateEvent(tag) 매크로함수” 를 이용해서 새로운 이벤트변수를 생성하는 매크로함수이다. 이벤트변수 명칭(event\_var)과 태그명칭(tag)만으로 기본값으로 초기화된 이벤트를 메모리풀에 생성하고 그 주소를 event\_var 에 저장한다. 이벤트의 EventHead 에 sender 는 “APPL”로 id, from 은 자신의 ID 로(param.id) 초기화 한다.

```
|| UserEvent(ude, UserDefined); // 사용자 이벤트 생성
|| ude->ms = clock; // 이벤트 멤버 설정
|| SendEvent(ude); // 이벤트 전송
```

## 10.28. SendEvent(), SendDelayedEvent() 함수

```
extern void SendEvent(void* event);
extern void SendDelayedEvent(uint32 delayTime, void* delayedEvent);
```

이벤트를 이벤트큐에 넣는다. SendDelayedEvent 는 시간지연 이벤트를 이벤트큐에 넣는다(DelayedEvent 참고)

### 10.29. ez\_malloc, ez\_free 함수

```
extern void ez_free(void* pe);
extern void* ez_malloc_(int size);
#define ez_malloc(data_type) (data_type*)ez_malloc_(sizeof(data_type))
```

메모리 할당 함수이다. eventbus\_fw 는 이벤트를 Heap 이 아닌 메모리풀에 저장한다. ez\_malloc(data\_type)은 data\_type 을 저장할 수 있는 메모리공간을 풀에 할당후 할당된 메모리의 시작주소를 리턴한다. 사용을 마친 메모리는 반드시 ez\_free(pe)를 사용하여 반환하여야 한다. 그렇지 않으면 메모리 부족으로 실행이 정지되므로 메모리 할당 및 해제는 매우 신중하여야 한다. 풀의 용량을 늘리고자 하면 드라이버 헤더파일에 선언된 아래 매크로를 변경하면 된다. (이벤트의 생성은 CreateEvent(), UserEvent()를 사용하는 것이 안전하다)

```
#define SIZE_OF_SMALL_POOL    (32)
#define SIZE_OF_MEDIUM_POOL   (16)
#define SIZE_OF_BIG_POOL      (8)
```

### 10.30. clock 전역변수

```
extern CLOCK clock; // 시스템 타이머(밀리초 단위)
```

밀리초(ms) 단위의 시스템의 기준 시계이다. 마이컴 리셋후 0ms 에서부터 증가하기 시작한다. 이 시간을 사용하면 시간간격을 측정할 수 있다. 그러나 대부분의 모듈은 마이컴 내부의 RC 발진기(최대 1% 오차)를 사용하므로 정밀한 시간측정에는 적합하지 않다.

### 10.31. EventBus() 함수

```
extern void EventBus();
```

이벤트버스 초기화 함수. 펌웨어 작성시 main() 함수의 앞부분에 위치해야 한다.

### 10.32. CRC() 함수

```
extern uint8 CRC(Event* pe);
```

이벤트의 CRC 를 계산하는 함수.

### 10.33. 이 CreateXXX 이미지 생성함수

```
extern AIH* CreateAIH(ID id);
extern AIM* CreateAIM(ID id);
extern AOM* CreateAOM(ID id);
extern DIE* CreateDIE(ID id);
extern DIM* CreateDIM(ID id);
extern DIO* CreateDIO(ID id);
extern DOE* CreateDOE(ID id);
extern DOM* CreateDOM(ID id);
```

```
extern ENV* CreateENV(ID id);
extern MOC* CreateMOC(ID id);
extern MSB* CreateMSB(ID id);
extern RCS* CreateRCS(ID id);
extern SEG* CreateSEG(ID id);
extern SMD* CreateSMD(ID id);
extern TCM* CreateTCM(ID id);
extern USB* CreateUSB(ID id);
extern LAN* CreateLAN(ID id);
extern LMO* CreateLMO(ID id);
extern USD* CreateUSD(ID id);
```

모듈의 이미지를 생성하고 주소를 리턴한다. 이벤트버스 케이블에 연결된 다른 모듈을 참조하려면 반드시 그 모듈의 이미지를 생성해야 한다.

```
DIM* dim = CreateDIM(ID_DIM); // 참조하는 모듈은 반드시 생성해야 한다.
```

#### 10.34. WriteROM, ReadROM 매크로

```
#define WriteROM      Write_ROM(&rom, sizeof(rom))
#define ReadROM       Read_ROM(&rom, sizeof(rom))
```

사용자 데이터를 EEPROM 에 저장하기 위해서 구조체 변수 rom 이 xxx\_def.h 에 정의되어 있다. 이 데이터를 EEPROM 에 쓰거나 읽기 위해 선언된 매크로이다. EEPROM 에 저장하려는 변수를 rom 구조체에 선언한 후 이 매크로를 호출하면 된다.

```
struct {
    int32 myvar;
} rom;

rom.myvar = 123;
WriteROM;
```

#### 10.35. UART\_SetBaudrate() 함수

```
extern void UART_SetBaudrate(uint32 bps);
```

UART 통신속도는 기본 115,200bps 로 설정되어 있으나 필요에 따라 이함수를 사용하면 프로그램에서 통신속도를 변경할 수 있다. 굳이 프로그램에서 설정해야 하는 경우가 아니라면 ezconfig.exe 에서 변경하는 것이 좋다.

#### 10.36. printc(), printf() 함수

```
extern void printc(char ch);
extern signed int printf(const char *pFormat, ...);
```

printc 함수는 문자 ch 를 UART 로 송신하고, printf 함수는 포맷 문자열 pFormat 으로 지정한 포맷으로 문자열을 UART 로 전송한다. printf 함수는 임베디드용으로 간략화되었으므로 실수형 %f 포맷은 지원되지 않는다. 이경우 1000 을 곱한 정수값으로 출력하는 등의 방법을 사용하여야 한다.

```
printf("Switch: 0x%X", in->switch5_0);
```

```
printf("Volt[mv]: %d", (int)(1000 * in->vin0));
```

### 10.37. seg\_sprintf () 함수

```
extern void seg_sprintf(char* text, const char *pFormat, ...);
```

SEG 문자표시모듈에 포맷된 문자열을 표시하는 함수이다.

```
seg_sprintf(seg->out->text, "val=%d", val);
```

### 10.38. HW\_SyncToOut() 함수

```
extern void HW_SyncToOut (Event* pe);
```

XxxOutEvent 이벤트를 수신할 때 호출되는 함수. 이 함수는 각 모듈의 드라이버 파일(예, dio\_drv.c 참고)에서 구현되어 있다. 사용자는 이 함수를 수정하여 출력방법을 변경할 수 있다.

### 10.39. Wait\_10us 시간지연 함수

```
extern void Wait_10us(uint32 n);
```

for 문을 사용한 정밀하지 않은 10 마이크로초 단위의 시간지연 함수이다. 지연시간은 10\*n 마이크로초이다.

### 10.40. DISABLE\_IRQ, ENABLE\_IRQ 인터럽트 함수

```
#define DISABLE_IRQ \
    do { \
        __disable_irq(); \
        n_irq++; \
    } while(0)

#define ENABLE_IRQ \
    do { \
        n_irq--; \
        if (n_irq == 0) \
            __enable_irq(); \
    } while(0)
```

DISABLE\_IRQ 는 인터럽트를 금지시키고, ENABLE\_IRQ 는 인터럽트를 다시 허용하므로, 두함수는 항상 쌍으로 사용되어야 한다.

### 10.41. EventProgram 매크로

```
#define EventProgram \
    Event* pe_; \
    EVENT_HANDLER: \
    pe_ = EventQ_Get(); \
    SaveOutImage(); \
    if (pe_ == NULL) goto SCAN_PROGRAM
```

이벤트큐에서 이벤트를 읽어서 포인터 pe\_에 넣고 이벤트핸들러에 건네준다. 만일 처리할 이벤트가 없다면 ScanProgram이 실행된다(IF 매크로함수 참고).

#### 10.42. ScanProgram 매크로

```
#define ScanProgram \
    Scan2SyncOut(); \
    Event_Sync(pe_); \
    goto EVENT_HANDLER; \
SCAN_PROGRAM:
```

이벤트 핸들러에서 사용을 마친 이벤트는 ScanProgram 매크로에 전달되고, Event\_Sync(pe\_) 함수에서 정해진 후처리 동작을 수행한다.

#### 10.43. End 매크로

```
#define End \
    Scan2SyncOut(); \
    goto EVENT_HANDLER
```

ScanProgram의 실행을 마치면 다시 EventProgram으로 점프하고 처리할 이벤트가 있는지 검사한다.

#### 10.44. IF() 매크로함수

```
#define IF(module, event_tag) \
    else if (Event_Is(pe_, module, event_tag)) { \
        event_tag##Event* pe __attribute__ ((unused)) = (event_tag##Event*)pe_;
```

이벤트핸들러를 정의하는 매크로함수이다. Event\_Is() 함수를 호출해서 이벤트(pe\_)가 event\_tag와 일치하는지 비교한다.

#### 10.45. Pressed, Released, Toggled, Toggle 매크로함수

```
#define Pressed(image, io)      (image->io == OFF && pe->io == ON) \
#define Released(image, io)     (image->io == ON && pe->io == OFF) \
#define Toggled(image, io)      (image->io != pe->io) \
#define Toggle(image, io)       (image->io = !(image->io))
```

스위치와 같은 ELogic 타입 접점(스위치, 릴레이등의)의 상승 및 하강에지를 검출하기 위한 응용매크로이다. 매크로 내용을 충분히 이해 후 사용해야 한다.

예를들어, Pressed(this->in, sw0)의 경우 매크로를 확장하면 아래와 같다.

```
(this->in->sw0 == OFF && pe->sw0 == ON)
```

여기서, this->in->sw0은 이미지에 저장된 스위치 상태로써 이벤트가 발생하기 이전의 상태이고, pe->sw0은 이벤트가 발생한 현재 스위치의 상태이므로, 이전에 OFF 상태에서 ON 상태로 바뀐 것을 의미하고, 상승에지가 되는 것입니다.

Toggled 는 ELogic 타입 접점이 토글(조작)되었는지 검사하고, Toggle 는 접점 상태를 토글(반전)시킬 때 사용합니다.

```
IF (this, DioIn) {
    if (Pressed(this->in, sw0)) {
        // sw0이 눌러진(상승에지) 경우의 처리
    }

    if (Released(this->in, sw0)) {
        // sw0이 놓아진(상승에지) 경우의 처리
    }

    if (Toggled(this->in, sw0)) {
        // sw0이 조작된(상승 또는 하강에지) 경우의 처리
    }

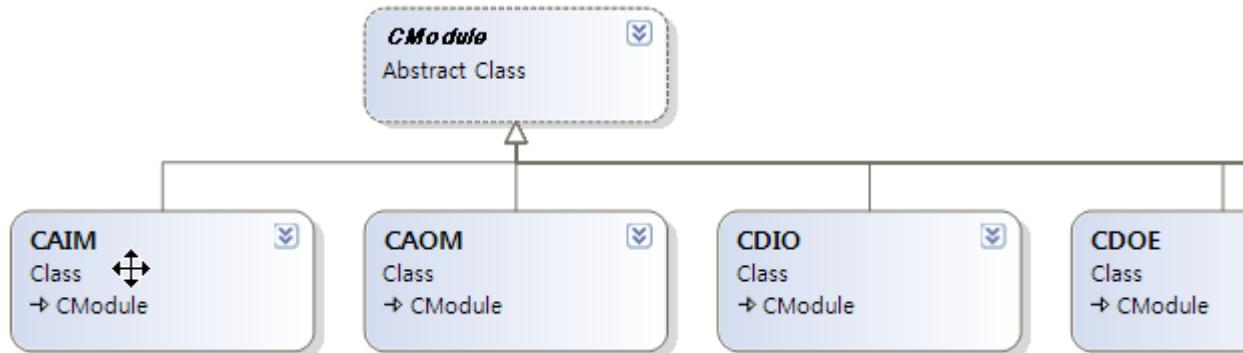
    Toggle(this->out, rly0)) // this->out->rly0을 반전시킴
}
```

## 11. eventbus\_net\_0.60 레퍼런스 가이드

eventbus\_net 라이브러리를 이용하면 닷넷에서 C#, C++, VB 으로 프로그램을 쉽게 개발할 수 있습니다. 지면상 중요 내용만 기술하였으므로 세부 내용은 소스코드를 참고하시기 바랍니다.

### 11.1. 모듈이미지 클래스

모듈의 이미지를 나타내는 클래스. eventbus\_fw 의 eventbus.h 에 정의된 각 모듈의 이미지 구조체와 비교해 보십시오. 구성이 유사하므로 쉽게 이해할 수 있습니다.



이지컨트롤러 모듈의 입출력 데이터는 앞서 보았듯이 이미지의 형태로 저장되고 교환됩니다. 예를 들어, AIM 모듈의 이미지는 eventbus\_fw 에서는 AIM 구조체로 eventbus\_net 에서는 CAIM 클래스로 선언되어 있습니다. 그림은 eventbus\_net 에서의 이미지에 대한 클래스다이어그램입니다. 모든 이미지는 CModule 클래스로부터 상속되었습니다. 전체코드를 모두 옮기면 지면이 복잡해 지므로 중요한 코드를 중심으로 나열하겠습니다.

#### 11.1.1. CModule 클래스

모든 이미지의 부모클래스.

```

public abstract class CModule
{
    public byte Id { get; set; }
    public event UserEventHandler OnUserEvent;
    public CModule(byte id);
}
  
```

- > Id : 모듈의 ID(통신주소, 다른 모듈과 중복되지 않는 유일한 값이어야 함)
- > OnUserEvent : 사용자정의 이벤트를 수신할 때 실행 하는 이벤트

```

public delegate void UserEventHandler(object sender,CEventArgs args);
  
```

- > 사용자정의 이벤트를 처리하는 이벤트핸들러의 원형(프로토타입)
- > sender : 이벤트를 발생한 모듈의 이미지
- > args : 발생한 사용자정의 이벤트의 데이터

```

public class CEventArgs : EventArgs
{
    public CEventArgs(byte tag, List<byte> serialized);
}
  
```

```

    public DateTime EventTime { get; set; }
    public List<byte> Serialized { get; set; }
    public byte Tag { get; set; }
}

```

- > EventTime: 이벤트 발생(통신으로 이벤트를 수신한)시간
- > Serialized : 사용자정의 이벤트의 데이터
- > Tag : 사용자정의 이벤트 태그(명)

```

typedef struct {
    EventHead e;
    uint64 ms;
} UserDefinedEvent;

```

예를 들어, `eventbus_fw`에서 사용자정의 이벤트 `MyEvent`를 만들 때, 항상 `EventHead`로 시작해야 합니다. 여기서 `EventHead`를 제외한 나머지 부분인 `ms`가 실제 사용자의 데이터입니다. 이 데이터를 바이트로 직렬화한 것이 `args.Serialized`, `EventHead`의 `tag`가 `args.Tag`로 이벤트핸들러에 넘어옵니다.

이제 사용자정의 이벤트(`UserDefinedEvent`)가 수신되면 이를 처리할 이벤트핸들러(`UserEventHandler`)를 작성하는 예를 살펴봅니다. 이벤트핸들러 메소드를 별도로 작성후 이벤트에 등록해도 되지만, 여기서는 익명메소드(delegate를 이용한)로 바로 작성하였습니다. 사용자정의 이벤트의 태그명은 `UserDefined`이며, 데이터는 64비트의 시간(`eventbus_fw`의 `clock`)인 경우입니다. 사용자 이벤트의 경우 사용자가 어떤 데이터를 전송할 지 `eventbus_net`은 알수 없으므로 수신한 `args.Serialized(List<byte> 타입)`를 수동으로 직접 `BitConverter`클래스를 이용하여 원래의 타입으로 복원시켜야 합니다.

```

dio.OnUserEvent += new UserEventHandler(
    delegate(object src, CUserEventArgs args) {
        switch ((EUserEventTag)args.Tag) {
            case EUserEventTag.UserDefined:
                UInt64 time = BitConverter.ToInt64(args.Serialized.ToArray(), 0);
                my_func(time);
                break;
            default:
                break;
        }
    }
);

```

### 11.1.2. CAIH 클래스

AIH 모듈의 이미지 클래스.

```

public class CAIH : CModule, IDisposable
{
    public AihParam param;

    public CVoltInput[] vin;

    public CVoltInput vin0 { get; }
    public CVoltInput vin1 { get; }
    public CVoltInput vin2 { get; }
    public CVoltInput vin3 { get; }
}

```

```

    public CNumericOutput uamp { get; set; }

    public CVoltInput Vaa { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CAIH(byte id);
    public void Dispose();
}

```

### 11.1.3. CAIM 클래스

AIM 모듈의 이미지 클래스.

```

public class CAIM : CModule, IDisposable
{
    public AimParam param;

    public CVoltInput[] vin;

    public CVoltInput vin0 { get; }
    public CVoltInput vin1 { get; }
    public CVoltInput vin2 { get; }
    public CVoltInput vin3 { get; }

    public CVoltInput Vaa { get; set; }
    public CVoltInput Vbb { get; set; }
    public CVoltInput Vdc { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CAIM(byte id);
    public void Dispose();
}

```

### 11.1.4. CAOM 클래스

AOM 모듈의 이미지 클래스.

```

public class CAOM : CModule, IDisposable
{
    public AomParam param;

    public CVoltOutput[] vout;
    public CCurrentOutput[] iout;

    public CVoltOutput vout0 { get; set; }
    public CVoltOutput vout1 { get; set; }
    public CCurrentOutput iout0 { get; set; }
    public CCurrentOutput iout1 { get; set; }

    public CVoltInput Vaa { get; set; }
    public CVoltInput Vbb { get; set; }
    public CVoltInput Vdc { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;
}

```

```

    public CAOM(byte id);
    public void Dispose();
}

```

### 11.1.5. CDIE 클래스

DIE 모듈의 이미지 클래스.

```

public class CDIE : CModule, IDisposable
{
    public DieParam param;

    public CSwitch[] sw;

    public CSwitch sw0 { get; set; }
    public CSwitch sw1 { get; set; }
    public CSwitch sw2 { get; set; }
    public CSwitch sw3 { get; set; }
    public CSwitch sw4 { get; set; }
    public CSwitch sw5 { get; set; }
    public CSwitch sw6 { get; set; }
    public CSwitch sw7 { get; set; }
    public CSwitch sw8 { get; set; }
    public CSwitch sw9 { get; set; }
    public CSwitch sw10 { get; set; }
    public CSwitch sw11 { get; set; }
    public CSwitch sw12 { get; set; }
    public CSwitch sw13 { get; set; }
    public CSwitch sw14 { get; set; }
    public CSwitch sw15 { get; set; }

    public CVoltInput Vaa { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CDIE(byte id);
    public void Dispose();
}

```

### 11.1.6. CDIM 클래스

DIM 모듈의 이미지 클래스.

```

public class CDIM : CModule, IDisposable
{
    public DimParam param;

    public CSwitch[] sw;

    public CSwitch sw0 { get; set; }
    public CSwitch sw1 { get; set; }
    public CSwitch sw2 { get; set; }
    public CSwitch sw3 { get; set; }
    public CSwitch sw4 { get; set; }
    public CSwitch sw5 { get; set; }
    public CSwitch sw6 { get; set; }
    public CSwitch sw7 { get; set; }

```

```

public CSwitch sw8 { get; set; }
public CSwitch sw9 { get; set; }

public CVoltInput Vaa { get; set; }
public CVoltInput Vbb { get; set; }
public CVoltInput Vdc { get; set; }

public event OnParamReceivedEventHandler onParamReceived;

public CDIM(byte id);
public void Dispose();
}

```

### 11.1.7. CDIO 클래스

eventbus\_fw 의 eventbus.h 에 정의된 DIO 구조체와 비교해 주십시오.

```

public class CDIO : CModule, IDisposable
{
    public DioParam param;

    public CRelay[] rly;

    public CRelay rly0 { get; set; }
    public CRelay rly1 { get; set; }

    public CSwitch[] sw;

    public CSwitch sw0 { get; set; }
    public CSwitch sw1 { get; set; }
    public CSwitch sw2 { get; set; }
    public CSwitch sw3 { get; set; }
    public CSwitch sw4 { get; set; }
    public CSwitch sw5 { get; set; }

    public CVoltInput Vaa { get; set; }
    public CVoltInput Vbb { get; set; }
    public CVoltInput Vdc { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CDIO(byte id);
    public void Dispose();
}

```

### 11.1.8. CDOE 클래스

DOE 모듈의 이미지 클래스.

```

public class CDOE : CModule, IDisposable
{
    public DoeParam param;

    public CRelay[] rly;

    public CRelay rly0 { get; set; }
    public CRelay rly1 { get; set; }

```

```

public CRelay rly2 { get; set; }
public CRelay rly3 { get; set; }
public CRelay rly4 { get; set; }
public CRelay rly5 { get; set; }
public CRelay rly6 { get; set; }
public CRelay rly7 { get; set; }
public CRelay rly8 { get; set; }
public CRelay rly9 { get; set; }
public CRelay rly10 { get; set; }
public CRelay rly11 { get; set; }
public CRelay rly12 { get; set; }
public CRelay rly13 { get; set; }
public CRelay rly14 { get; set; }
public CRelay rly15 { get; set; }

public CVoltInput Vaa { get; set; }

public event OnParamReceivedEventHandler onParamReceived;

public CDOM(byte id);
public void Dispose();
}

```

### 11.1.9. CDOM 클래스

DOM 모듈의 이미지 클래스.

```

public class CDOM : CModule, IDisposable
{
    public DomParam param;

    public CRelay[] rly;

    public CRelay rly0 { get; set; }
    public CRelay rly1 { get; set; }
    public CRelay rly2 { get; set; }
    public CRelay rly3 { get; set; }
    public CRelay rly4 { get; set; }
    public CRelay rly5 { get; set; }

    public CVoltInput Vaa { get; set; }
    public CVoltInput Vbb { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CDOM(byte id);
    public void Dispose();
}

```

### 11.1.10. CENV 클래스

ENV 모듈의 이미지 클래스.

```

public class CENV : CModule, IDisposable
{
    public EnvParam param;
}

```

```

public CNumericInput lux { get; set; }
public CNumericInput uv { get; set; }
public CNumericInput hand { get; set; }
public CNumericInput temp { get; set; }

public CVoltInput Vaa { get; set; }

public event OnParamReceivedEventHandler onParamReceived;

public CENV(byte id);
public void Dispose();
}

```

### 11.1.11. CLAN 클래스

LAN 모듈의 이미지 클래스.

```

public class CLAN : CModule, IDisposable
{
    public LanParam param;

    public CVoltInput Vaa { get; set; }
    public CVoltInput Vbb { get; set; }

    public event ReceivedEventHandler OnAtResponseReceived;
    public event CharReceivedEventHandler OnCharReceived;
    public event LanConnectedEventHandler OnConnected;
    public event DisconnectedEventHandler OnDisconnected;
    public event OnParamReceivedEventHandler onParamReceived;

    public event OnParamReceivedEventHandler onParamReceived;

    public bool Connected { get; }

    public CLAN(byte id);
    public bool Connect(string ip, int port);
    public void Disconnect();
    public void Dispose();
    public void RequestParam(CModule module);
    public void RequestSync(CModule module);
    public void Synchronize(CModule module);
    public bool Transmit(List<byte> serialized);
    public bool Transmit(string packet);
}

```

### 11.1.12. CMOC 클래스

MOC 모듈의 이미지 클래스.

```

public class CMOC : CModule, IDisposable
{
    public MocParam param;

    public CNumericOutput accel { get; set; }
    public CFlag busy { get; set; }
    public CCommand cmd { get; set; }
    public CDirection dir { get; set; }
}

```

```

public CNumericOutput ipps { get; set; }
public CSwitch jccw { get; set; }
public CSwitch jcw { get; set; }
public CSwitch jspd { get; set; }
public CNumericInput jvol { get; set; }
public CNumericInput loc { get; set; }
public CNumericOutput move { get; set; }
public CSwitch neg { get; set; }
public CSwitch nnp { get; set; }
public CSwitch pos { get; set; }
public CNumericOutput pps { get; set; }

public CVoltInput Vaa { get; set; }

public event OnParamReceivedEventHandler onParamReceived;

public CMOC(byte id);
public void Dispose();
}

```

#### 11.1.13. CMSB 클래스

MSB 모듈의 이미지 클래스.

```

public class CMSB : CModule, IDisposable
{
    public MsbParam param;

    public CMsbPacket Packet { get; set; }
    public CVoltInput Vaa { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CMSB(byte id);
    public void Dispose();
}

```

#### 11.1.14. CRCS 클래스

RCS 모듈의 이미지 클래스.

```

public class CRCS : CModule, IDisposable
{
    public RcsParam param;

    public CNumericOutput[] pwm;

    public CNumericOutput pwm0 { get; }
    public CNumericOutput pwm1 { get; }
    public CNumericOutput pwm2 { get; }
    public CNumericOutput pwm3 { get; }
    public CNumericOutput pwm4 { get; }
    public CNumericOutput pwm5 { get; }
    public CNumericOutput pwm6 { get; }
    public CNumericOutput pwm7 { get; }

    public CVoltInput Vaa { get; set; }
}

```

```

    public event OnParamReceivedEventHandler onParamReceived;

    public CRCS(byte id);
    public void Dispose();
}

```

### 11.1.15. CSEG 클래스

SEG 모듈의 이미지 클래스.

```

public class CSEG : CModule, IDisposable
{
    public SegParam param;

    public CNumericInput cds { get; set; }
    public CStringOutput Text { get; set; }

    public CVoltInput Vaa { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CSEG(byte id);
    public void Dispose();
}

```

### 11.1.16. CSMD 클래스

SMD 모듈의 이미지 클래스.

```

public class CSMD : CModule, IDisposable
{
    public SmdParam param;

    public CCurrrentOutput amp { get; set; }
    public CFlag busy0 { get; set; }
    public CFlag busy1 { get; set; }
    public CFlag fan { get; set; }
    public CNumericInput heat { get; set; }
    public CStepper m0 { get; set; }
    public CStepper m1 { get; set; }
    public CVoltInput vm { get; set; }

    public CVoltInput Vaa { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CSMD(byte id);
    public void Dispose();
}

```

### 11.1.17. CTCM 클래스

TCM 모듈의 이미지 클래스.

```

public class CTCM : CModule, IDisposable
{

```

```

public TcmParam param;

public CNumericInput[] tc;

public CNumericInput tc0 { get; set; }
public CNumericInput tc1 { get; set; }
public CNumericInput tc2 { get; set; }
public CNumericInput tc3 { get; set; }
public CNumericInput tc4 { get; set; }
public CNumericInput tc5 { get; set; }

public CVoltInput Vaa { get; set; }

public event OnParamReceivedEventHandler onParamReceived;

public CTCM(byte id);
public void Dispose();
}

```

### 11.1.18. CUSB 클래스

USB 모듈의 이미지 클래스.

```

public class CUSB : CModule, IDisposable
{
    public UsbParam param;

    public Serial COM { get; set; }
    public bool Connected { get; }

    public CVoltInput Vaa { get; set; }
    public CVoltInput Vbb { get; set; }

    public event OnParamReceivedEventHandler onParamReceived;

    public CUSB(byte id);
    public void Dispose();
    public bool Connect(string portname, int baudrate);
    public void Disconnect();
    public void RequestParam(CModule module);
    public void RequestSync(CModule module);
    public void Synchronize(CModule module);
}

```

### 11.1.19. CUSD 클래스

USD 모듈의 이미지 클래스.

```

public class CUSD : CModule, IDisposable
{
    public UsdParam param;

    public CNumericOutput accel { get; set; }
    public CFlag busy { get; set; }
    public CCommand cmd { get; set; }
    public CDirection dir { get; set; }
    public CFlag fan { get; set; }
}

```

```
public CNumericInput heat { get; set; }
public CNumericOutput ipps { get; set; }
public CSwitch jccw { get; set; }
public CSwitch jcw { get; set; }
public CSwitch jspd { get; set; }
public CNumericInput jvol { get; set; }
public CNumericInput loc { get; set; }
public CNumericOutput move { get; set; }
public CSwitch neg { get; set; }
public CSwitch nnp { get; set; }
public CSwitch pos { get; set; }
public CNumericOutput pps { get; set; }
public CUsdStep step { get; set; }
public CSwitch sw0 { get; set; }
public CSwitch sw1 { get; set; }
public CVoltInput vm { get; set; }

public CVoltInput Vaa { get; set; }

public event OnParamReceivedEventHandler onParamReceived;

public CUSD(byte id);
public void Dispose();
}
```

## 11.2. Serial 통신클래스

---

시리얼 COM 포트를 통한 통신에 사용되는 클래스.

```
public class Serial
{
    public bool Connected { get; }
    public SerialPort Port { get; set; }
    public string PortName { get; set; }

    public event ReceivedEventHandler OnAtResponseReceived;
    public event CharReceivedEventHandler OnCharReceived;
    public event ComConnectedEventHandler OnConnected;
    public event CrcMismatchEventHandler OnCrcMismatch;
    public event DisconnectedEventHandler OnDisconnected;

    public Serial();
    public bool Connect(string portname, int baudrate);
    public void Disconnect();
    public void RequestParam(CModule module);
    public void RequestSync(CModule module);
    public void Synchronize(CModule module);
    public bool Transmit(List<byte> serialized);
    public bool Transmit(string packet);
}
```

## 11.3. 이벤트 클래스

---

### 11.3.1. MsbTransmitEvent 이벤트

MSB 의 시리얼 채널에 데이터 송신을 요청하는 이벤트.

```
public class CMsbTransmitEvent
{
    public CMsbTransmitEvent(byte chan, List<byte> packet);

    public byte Chan { get; set; }
    public List<byte> Packet { get; set; }
}
```

## 11.4. EventBus\DDevice.cs 데이터 클래스

---

eventbus\_fw에서 사용된 스위치, 릴레이, 전압 등의 입출력데이터를 나타내는 클래스.

### 11.4.1. Cswitch 클래스

DIM,DIO,MOC에서 사용된 스위치 입력 클래스.

```
public class Cswitch
{
    public Cswitch(bool initialState);

    public DateTime EventTime { get; set; }
    public bool State { get; set; }

    public event LogicEventHandler OnChanged;
    public event LogicEventHandler OnFalling;
    public event LogicEventHandler OnRising;
}
```

- 속성

EventTime: 이벤트 발생(통신으로 이벤트를 수신한)시간  
State : 스위치 상태, true(스위치 ON), false(스위치 OFF)

- 이벤트

OnChanged : 스위치의 상태가 변할 때 발생  
OnRising : 스위치가 ON 될 때 발생  
OnFalling : 스위치가 OFF 될 때 발생

### 11.4.2. CRelay 클래스

DIO,DOM에서 사용된 릴레이 출력 클래스.

```
public class CRelay
{
    public CRelay(bool initialState);

    public DateTime EventTime { get; set; }
    public bool State { get; set; }

    public event LogicEventHandler OnChanged;
    public event LogicEventHandler OnFalling;
    public event LogicEventHandler OnRising;
}
```

- 속성

EventTime: 이벤트 발생(통신으로 이벤트를 수신한)시간  
State : 릴레이 상태, true(릴레이 ON), false(릴레이 OFF)

- 이벤트

OnChanged : 릴레이의 상태가 변할 때 발생

OnRising : 릴레이가 ON 될 때 발생  
 OnFalling : 릴레이가 OFF 될 때 발생

#### 11.4.3. CVoltInput 클래스

AIM에서 사용된 입력전압 클래스.

```
public class CVoltInput
{
    public CVoltInput(decimal initialVolt);

    public DateTime EventTime { get; set; }
    public decimal Volt { get; set; }

    public event VoltageEventHandler OnChanged;
}
```

- 속성
  - EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
  - Volt : 입력받은 전압
- 이벤트
  - OnChanged : 입력전압이 문턱치 이상 변할 때 발생

#### 11.4.4. CVoltOutput 클래스

AOM에서 사용된 전압 출력 클래스.

```
public class CVoltOutput
{
    public CVoltOutput(decimal initialVolt);

    public DateTime EventTime { get; set; }
    public decimal Volt { get; set; }

    public event VoltageEventHandler OnChanged;
}
```

- 속성
  - EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
  - Volt : 출력할 전압
- 이벤트
  - OnChanged : 출력할 전압값이 변할 때 발생

#### 11.4.5. CCurrentOutput 클래스

AOM에서 사용된 전류 출력 클래스.

```
public class CCurrentOutput
{
```

```

public CCurrentOutput(decimal initialMilliAmp);

public DateTime EventTime { get; set; }
public decimal mA { get; set; }

public event CurrentEventHandler OnChanged;
}

```

■ 속성

EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간  
MilliAmp : 출력할 전류, mA 단위

■ 이벤트

OnChanged : 출력할 전류값이 변할 때 발생

#### 11.4.6. CNumericInput 클래스

ENV에서 사용된 숫자 입력 클래스.

```

public class CNumericInput
{
    public CNumericInput(decimal initialValue);

    public DateTime EventTime { get; set; }
    public decimal Value { get; set; }

    public event NumericEventHandler OnChanged;
}

```

■ 속성

EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간  
Value : 입력숫자 데이터

■ 이벤트

OnChanged : 입력숫자 값이 변할 때 발생

#### 11.4.7. CNumericOutput 클래스

MOC에서 사용된 숫자 출력 클래스.

```

public class CNumericOutput
{
    public CNumericOutput(decimal initialValue);

    public DateTime EventTime { get; set; }
    public decimal Value { get; set; }

    public event NumericEventHandler OnChanged;
}

```

■ 속성

`EventTime` : 이벤트 발생(통신으로 이벤트를 수신한)시간  
`Value` : 출력숫자 데이터

■ 이벤트

`OnChanged` : 출력할 숫자값이 변할 때 발생

#### 11.4.8. CFlag 클래스

MOC, SMD에서 사용된 상태플래그에 대한 클래스.

```
public class CFlag
{
    public CFlag(bool initialState);

    public DateTime EventTime { get; set; }
    public bool State { get; set; }

    public event LogicEventHandler OnChanged;
    public event LogicEventHandler OnSet;
    public event LogicEventHandler OnClear;
}
```

■ 속성

`EventTime` : 이벤트 발생(통신으로 이벤트를 수신한)시간  
`State` : 플래그 상태 데이터

■ 이벤트

`OnChanged` : 플래그 상태가 변할 때 발생  
`OnSet` : 플래그 상태가 세트될 때 발생  
`OnClear` : 플래그 상태가 리세트될 때 발생

#### 11.4.9. CCommand 클래스

MOC, SMD에서 사용된 모터 동작명령 데이터 클래스.

```
public class CCommand
{
    public CCommand(CCommand.CMD command);

    public DateTime EventTime { get; set; }
    public CCommand.CMD Value { get; set; }

    public event CommandEventHandler OnChanged;

    public enum CMD
    {
        FREE = 0,
        HOLD = 1,
        STOP = 2,
        RUN = 3,
        MOVE = 4,
    }
}
```

```
| } }
```

■ 속성

EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간

Value : 동작명령 데이터

■ 이벤트

OnChanged : 명령이 바뀔 때 발생

#### 11.4.10. CDirection 클래스

SMD, MOC에서 사용된 회전방향 데이터 클래스.

```
public class CDirection
{
    public CDirection(CDirection.DIR direction);

    public DateTime EventTime { get; set; }
    public CDirection.DIR Value { get; set; }

    public event DirectionEventHandler OnChanged;

    public enum DIR
    {
        CW = 0,
        CCW = 1,
    }
}
```

■ 속성

EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간

Value : 회전방향 데이터

■ 이벤트

OnChanged : 회전방향이 바뀔 때 발생

#### 11.4.11. CStringOutput 클래스

ENV에서 사용된 표시문자 데이터 클래스.

```
public class CStringOutput
{
    public CStringOutput(string initialText);

    public DateTime EventTime { get; set; }
    public string Body { get; set; }

    public event StringEventHandler OnChanged;
}
```

■ 속성

EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간  
Body : 표시할 문장(스트링)

■ 이벤트

OnChanged : 표시할 문장이 변할 때 발생

#### 11.4.12. CStepper 클래스

SMD에서 사용된 스텝모터 동작데이터 클래스.

```
public class CStepper
{
    public CStepper(uint pps, uint pulse);

    public DateTime EventTime { get; set; }
    public bool run { get; set; }
    public CStepper.CMD cmd { get; set; }
    public CStepper.DIR dir { get; set; }
    public CStepper.STEP step { get; set; }
    public ushort ippss { get; set; }
    public uint pps { get; set; }
    public ushort accel { get; set; }
    public uint move { get; set; }

    public event StepperEventHandler OnChanged;

    public enum CMD
    {
        FREE = 0,
        HOLD = 1,
        STOP = 2,
        RUN = 3,
        MOVE = 4,
    }

    public enum DIR
    {
        CW = 0,
        CCW = 1,
    }

    public enum STEP
    {
        SIXTEENTH = 0,
        QUARTER = 1,
        EIGHTH = 2,
        HALF = 3,
    }
}
```

■ 속성

EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간  
run : 명령의 유효성

cmd : 명령의 종류  
 dir : 회전방향  
 step : 마이크로스테핑 구동 선택  
 ipp : 초기 회전속도  
 pps : 정속 회전속도  
 accel : 가감속 기울기, 펄스/밀리초  
 move : 이동거리(MOVE 명령에서)

- 이벤트

OnChanged : 스텝모터 구동명령이 바뀔 때 발생

#### 11.4.13. CMsbPacket 클래스

MSB에서 사용된 패킷데이터 클래스.

```

public class CMsbPacket
{
  public CMsbPacket(byte channel, List<byte> packet);

  public DateTime EventTime { get; set; }
  public byte Channel { get; set; }
  public List<byte> Packet { get; set; }

  public event MsbReceivedEventHandler OnReceived;
}
  
```

- 속성

EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간

Channel : 패킷을 수신한 채널

Packet : 채널에서 수신한 시리얼 데이터

- 이벤트

OnReceived : 패킷을 수신할 때 실행할 이벤트

## 11.5. EventBus\Event.cs 이벤트 아규먼트 클래스

---

이벤트핸들러에 건네는 아규먼트.

사용예제)

```
dio.Switch2.OnChanged += new LogicEventHandler( //이벤트에 핸들러를 추가한다.
    delegate(object src, CLogicEventArgs args) { //핸들러의 두번째 인자는 아규먼트
        Switch2_CheckBox.Checked = args.State; //아규먼트에서 값을 읽을 수 있다.
    }
);
```

### 11.5.1. CUserEventArgs 클래스

사용자정의 이벤트의 경우 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CUserEventArgs : EventArgs
{
    public CUserEventArgs(byte tag, List<byte> serialized);

    public DateTime EventTime { get; set; }
    public List<byte> Serialized { get; set; }
    public byte Tag { get; set; }
}

> EventTime: 이벤트 발생(통신으로 이벤트를 수신한)시간
> Serialized : 사용자정의 이벤트 데이터(구조체)
> Tag : 사용자정의 이벤트 태그(명)
```

### 11.5.2. CVoltageEventArgs 클래스

VoltageEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CVoltageEventArgs : EventArgs
{
    public CVoltageEventArgs(decimal volt);

    public DateTime EventTime { get; set; }
    public decimal Volt { get; set; }
}

> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> Volt : 이벤트를 발생시킨 전압
```

### 11.5.3. CCurrentEventArgs 클래스

CurrentEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CCurrentEventArgs : EventArgs
{
    public CCurrentEventArgs(decimal milliAmp);

    public DateTime EventTime { get; set; }
    public decimal MilliAmp { get; set; }
```

```
| }
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> MilliAmp : 이벤트를 발생시킨 전류
```

#### 11.5.4. CLogicEventArgs 클래스

LogicEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CLogicEventArgs : EventArgs
{
    public CLogicEventArgs(bool state);

    public DateTime EventTime { get; set; }
    public bool State { get; set; }
}
```

```
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> State : 이벤트를 발생시킨 로직 상태값
```

#### 11.5.5. CNumericEventArgs 클래스

NumericEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CNumericEventArgs : EventArgs
{
    public CNumericEventArgs(decimal value);

    public DateTime EventTime { get; set; }
    public decimal Value { get; set; }
}
```

```
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> Value : 이벤트를 발생시킨 숫자
```

#### 11.5.6. CCommandEventEventArgs 클래스

CommandEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CCommandEventEventArgs : EventArgs
{
    public CCommandEventEventArgs(CCommand.CMD cmd);

    public DateTime EventTime { get; set; }
    public CCommand.CMD cmd { get; set; }
}
```

```
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> cmd : MOC, SMD 의 동작명령
```

#### 11.5.7. CDirectionEventArgs 클래스

DirectionEventArgs 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CDirectionEventArgs : EventArgs
{
    public CDirectionEventArgs(CDirection.DIR dir);

    public DateTime EventTime { get; set; }
    public CDirection.DIR dir { get; set; }
}

> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> dir: MOC, SMD 의 회전방향
```

### 11.5.8. CStepperArgs 클래스

StepperEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CStepperArgs : EventArgs
{
    public CStepperArgs(CStepper.CMD cmd, CStepper.STEP step, CStepper.DIR dir,
                        bool run, ushort ipps, ushort accel, uint pps, uint move);

    public DateTime EventTime { get; set; }
    public bool run { get; set; }
    public CStepper.CMD cmd { get; set; }
    public CStepper.DIR dir { get; set; }
    public CStepper.STEP step { get; set; }
    public ushort ipps { get; set; }
    public uint pps { get; set; }
    public ushort accel { get; set; }
    public uint move { get; set; }
}

> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> run : 명령의 유효성
> cmd : 명령의 종류
> dir : 회전방향
> step : 마이크로스테핑 구동 선택
> ipps : 초기 회전속도
> pps : 정속 회전속도
> accel : 가감속 기울기, 펄스/밀리초
> move : 이동거리(MOVE 명령에서)
```

### 11.5.9. CStringEventArgs 클래스

StringEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```
public class CStringEventArgs : EventArgs
{
    public CStringEventArgs(string text);
```

```

    public DateTime EventTime { get; set; }
    public string Text { get; set; }
}
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> Text : 이벤트를 발생시킨 문자열

```

#### 11.5.10. CAdcEventArgs 클래스

AdcEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```

public class CAdcEventArgs : EventArgs
{
    public CAdcEventArgs(UInt32 adc);

    public DateTime EventTime { get; set; }
    public UInt32 Adc { get; set; }
}
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> Adc : 이벤트를 발생시킨 A/D 변환값

```

#### 11.5.11. CUsdStepEventArgs 클래스

UsdStepEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```

public class CUsdStepEventArgs : EventArgs
{
    public CUsdStepEventArgs(CUsdStep.STEP step);

    public DateTime EventTime { get; set; }
    public CUsdStep.STEP step { get; set; }
}
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> Step : 이벤트를 발생시킨 USD 모듈의 step 값

```

#### 11.5.12. CMsbReceivedEventArgs 클래스

MsbReceivedEventHandler 델리게이트 타입의 이벤트핸들러에 건네주는 아규먼트입니다.

```

public class CMsbReceivedEventArgs : EventArgs
{
    public CMsbReceivedEventArgs(byte chan, List<byte> packet);

    public DateTime EventTime { get; set; }
    public byte Chan { get; set; }
    public List<byte> Packet { get; set; }
}
> EventTime : 이벤트 발생(통신으로 이벤트를 수신한)시간
> Chan : 패킷을 수신한 시리얼 채널 번호
> Packet : 수신한 시리얼 데이터 패킷

```