



User's Manual 2005-06-29

Closer to Real, ROBOTIS

# Dynamixel AX-12



## 목 차

<b>1. 요약</b>	
1-1. AX-12의 개요와 특징	Page 2
1-2. 주요 사양 요약	Page 3
<b>2. Dynamixel 기동</b>	
2-1. 기계부 조립방법	Page 4
2-2. Connector 조립	Page 5
2-3. Dynamixel의 배선연결	Page 6
<b>3. Communication Protocol</b>	
3-1. Communication 개요	Page 9
3-2. Instruction Packet	Page 10
3-3. Status Packet	Page 10
3-4. Control Table	Page 12
<b>4. Instruction Set과 그 사용 예</b>	
4-1. WRITE DATA	Page 19
4-2. READ DATA	Page 20
4-3. REG WRITE와 ACTION	Page 20
4-4. PING	Page 21
4-5. RESET	Page 22
4-6. SYNC WRITE	Page 23
<b>5. Example</b>	Page 24
<b>Appendix</b>	Page 30

## 1. Dynamixel AX-12

### 1-1. AX-12의 개요와 특징

**Dynamixel AX-12**

Dynamixel은 감속기, Driver, Control Unit 및 Network기능까지 일체형으로 구성되어 있는 Module형 Smart Actuator이다. Dynamixel은 Compact size임에도 불구하고 큰 Torque를 낼 수 있고, 강한 외력에 견딜 수 있는 특수한 재질로 만들어졌다. 또한 내부온도 변화나 공급전압의 변화등 내부 상황을 스스로 인식하고 처리할 수 있는 기능을 갖고 있다. Dynamixel AX-12는 다음과 같은 장점이 있다.

**정밀 제어**

1024단계의 resolution으로 position과 speed를 제어 할 수 있다.

**Compliance Driving**

위치 제어에 있어서 탄력의 정도를 설정할 수 있다.

**Feedback**

위치각, 현재 속도는 물론, 구동중인 Load의 크기까지 Feedback해 줄 수 있다.

**Alarm System**

내부 온도, Torque, 공급 전압 등이 사용자가 지정한 동작 범위를 벗어났을 때 이를 알려줄 뿐 아니라(Alarming) 스스로 대처할 수 있는 기능(Torque Off)이 있다.

**Communication**

Daisy chain으로 연결되어 배선이 간단하며 통신속도는 1M BPS까지 지원된다.

**Distributed Control**

한번의 명령 Packet을 전달하여 속도, 위치, Compliance, Torque를 동시에 설정할 수 있기 때문에 메인 프로세서는 매우 적은 Resource로 여러 개의 Dynamixel을 Control할 수 있다.

**Engineering Plastic**

Main Body에 Engineering Plastic이 사용되어 큰 Torque에도 견딜 수 있다.

**Axle Bearing**

기어의 최종 축에 Bearing을 사용하였으므로 축이 강한 외력을 받을 경우도 효율이 감소하지 않는다.

**Status LED**

ERROR 상황을 LED를 통하여 사용자에게 알려준다.

**Frames**

Hinge와 side mount frame을 기본으로 제공한다.

## 1-2. 주요 사양 요약

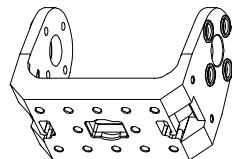
AX-12	
Weight (g)	55
Gear Reduction Ratio	1/254
Input Voltage (V)	at 7V at 10V
Final Max Holding Torque(kgf.cm)	12 16.5
Sec/60degree	0.269 0.196

Resolution	0.35°
Operating Angle	300° , Endless Turn
Voltage	7V~10V (Recommended voltage: 9.6V)
Max. Current	900mA
Operate Temperature	-5°C ~ +85°C
Command Signal	Digital Packet
Protocol Type	Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical)	TTL Level Multi Drop (daisy chain type Connector)
ID	254 ID (0~253)
Communication Speed	7343bps ~ 1 Mbps
Feedback	Position, Temperature, Load, Input Voltage, etc.
Material	Engineering Plastic

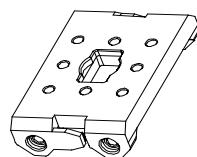
## 2. Dynamixel의 설치

### 2-1. 기계부 조립 방법

기본 제공 Frames AX-12와 함께 기본으로 제공되는 Frame은 다음과 같다.



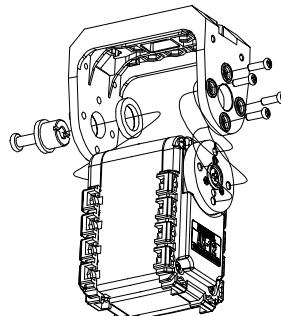
OF-12SH



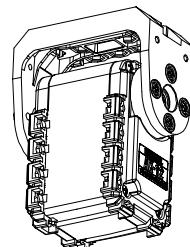
OF-12S

OF-12SH의 적용

OF-12SH는 다음과 같은 방법으로 적용된다.



조립 방법

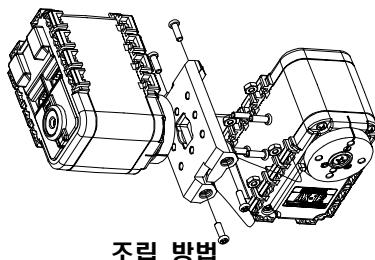


조립된 모습

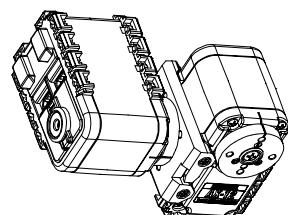
OF-12S의 적용

OF-12S는 다음과 같은 방법으로 적용된다. OF-12S는 AX-12의 3면(좌,우,아래면)에 결합될 수 있다.

Horn2Body

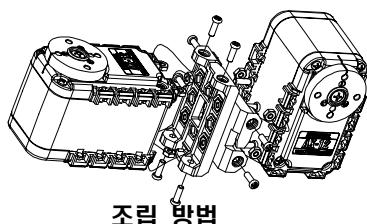


조립 방법

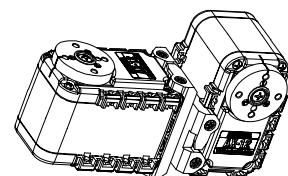


조립된 모습

Body2Body



조립 방법



조립된 모습

## 2-2. Connector 조립

아래 그림 순서대로 Connector를 조립한다. Wire Former를 사용해서 터미널을 선에 압착시킨다. Wire Former를 사용할 수 없을 때는 터미널과 선을 납땜하여 동작 중에 선이 빠지지 않도록 한다.

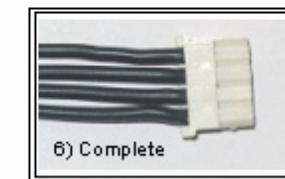
1) Stripping



2) Inserting



3) Forming

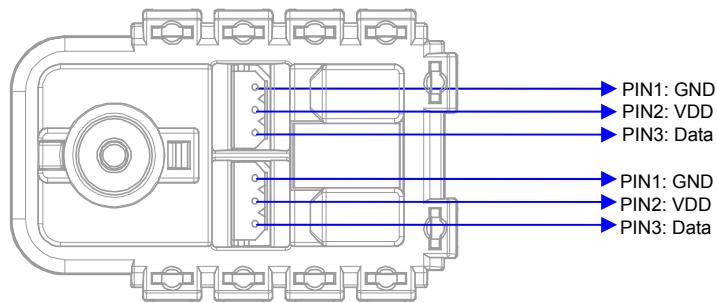


5) Assembling

6) Complete

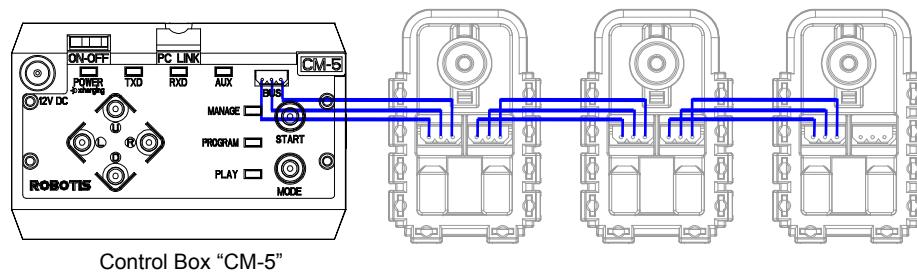
## 2-3. Dynamixel의 배선 연결

**Pin Assignment** Connector의 Pin 배열은 다음과 같으며 두개의 Connector는 Dynamixel 내부에서 Pin2Pin으로 연결되어 있으므로 하나의 커넥터에만 연결해서 AX-12를 구동할 수 있다.



### Wiring

아래 그림과 같이 Pin2Pin으로 연결한다. 이러한 방식으로 연결하여 여러 개의 AX-12들을 하나의 BUS상에서 제어할 수 있다.

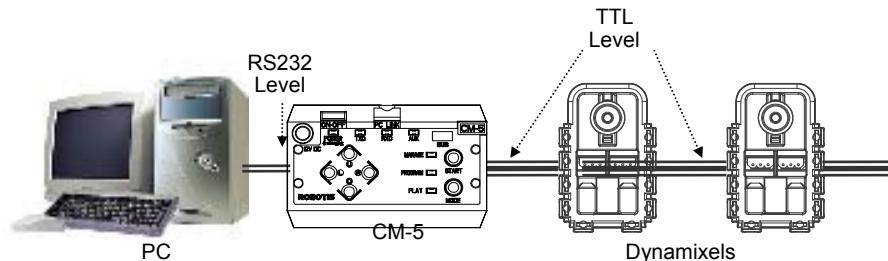


### Main Controller

Dynamixel을 구동할 Main Controller는 TTL level의 Half Duplex UART를 지원해야 한다. Main Controller는 직접 자작해도 좋으나 Dynamixel 전용 Controller인 CM-5를 사용하는 것을 추천한다.

### PC LINK

CM-5를 경유하여 PC로 Dynamixel을 제어할 수 있다.



bioloid

CM-5와 AX-12를 연결하는 것만으로도 로봇을 구성할 수 있다. CM-5와 AX-12를 기반으로 하여 구성된 로봇으로서 Bioloid라는 Edutainment 만능 로봇 KIT가 있다.

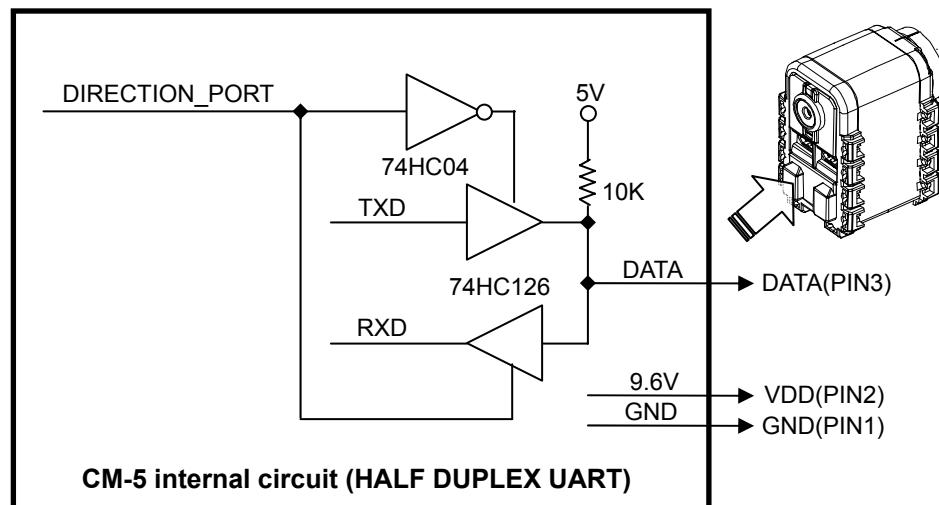


Bioloid로 만든 로봇 예

Bioloid의 보다 자세한 내용은 해당 Manual을 참조하기 바란다.

#### UART와의 연결

Dynamixel AX-12를 제어하기 위해서는 Main Controller UART의 신호를 Half duplex type으로 변환시켜 줘야 한다. 다음은 그 권장 회로도이다.



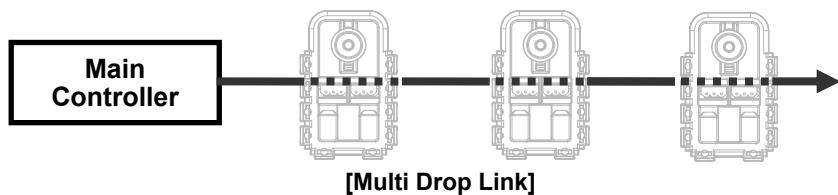
전원은 Main Controller의 Molex3P Connector의 Pin1, Pin2를 통하여 Dynamixel로 공급된다. (위의 회로는 Half duplex방식을 설명하기 위해 도시한 것이며, CM-5에는 이미 위의 회로가 내장되어 있으므로 단지 Dynamixel을 연결만 하면 된다)

위의 회로도에서 TTL Level의 TxD와 RxD는 DIRECTION\_PORT의 Level에 따라 다음과 같이 Data 신호의 방향이 결정된다.

- DIRECTION\_PORT의 Level이 High인 경우 : TxD의 신호가 Data로 출력
- DIRECTION\_PORT의 Level이 Low인 경우 : Data의 신호가 RxD로 입력

**Half Duplex UART**

Half Duplex UART을 사용함으로써 하나의 Node에 여러 개의 Dynamixel을 연결하는 Multi-Drop Link방식이 가능하다. 때문에 Dynamixel을 제어 할 때는 여러 군데에서 동시에 Data를 송신하지 않도록 Protocol이 운영되어야 한다.

**주의**

배선시엔 Pin 배열이 틀리지 않도록 각별히 주의한다. 최초로 전원을 인가할 때 전류 소비량을 확인한다. Standby상태에서 한 개의 Dynamixel의 소비 전류는 50mA이하이다.

**연결 확인**

배선을 통하여 Dynamixel에 전원이 올바르게 공급되었다면 Dynamixel의 LED가 두 번 깜박인다.

**점검**

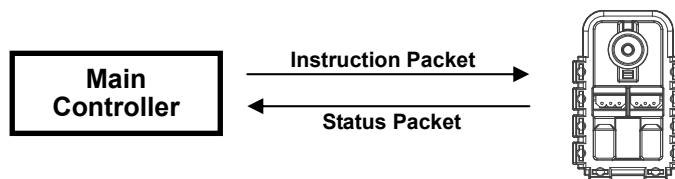
위의 과정을 성공하지 못했을 경우 Connector의 Pin 배열을 다시 확인한다. 전원 공급 장치의 전압 및 전류 허용량을 확인한다.

### 3. Communication Protocol

#### 3-1. Communication의 개요

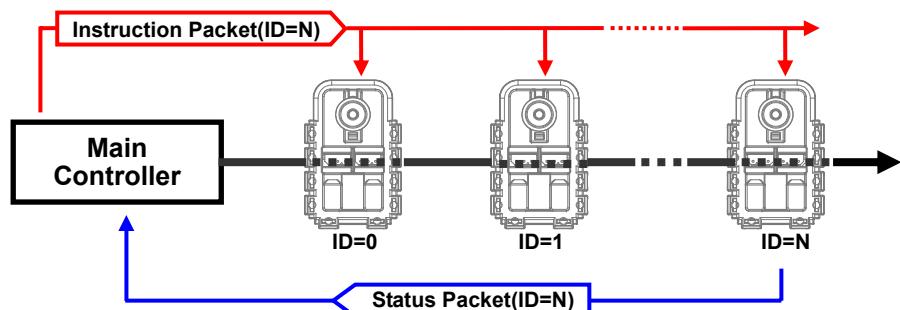
##### Packet

Main Controller와 Dynamixel은 packet을 주고 받으며 통신한다. packet의 종류로는 Main Controller에서 Dynamixel로 전송되는 Instruction Packet과 Dynamixel에서 Main Controller로 전송되는 Status Packet이 있다.



##### Communication

아래의 그림과 같이 연결된 시스템에서, Main Controller가 ID=N으로 설정된 Instruction Packet을 전송할 경우 여러 개의 Dynamixel중 ID가 N인 Dynamixel만이 Status Packet을 return하고 그 Instruction을 수행한다.



##### Unique ID

여러 개의 Dynamixel이 동시에 Packet을 선로로 전송하면 Packet충돌이 일어나서 통신에 문제를 일으킨다. 그러므로 Network Node안에 ID가 같은 Dynamixel이 존재하지 않도록 ID설정을 해야 한다.

##### Protocol

Dynamixel은 8 bit, 1 Stop bit, None Parity의 Asynchronous Serial 통신을 한다.

### 3-2. Instruction Packet

Instruction Packet은 Main Controller가 Dynamixel에게 동작을 지시하는 Packet이다. Instruction Packet의 구조는 다음과 같다.

Instruction Packet **OXFF OXFF ID LENGTH INSTRUCTION PARAMETER1 …PARAMETER N CHECK SUM**

Packet을 이루는 각 byte들의 의미는 다음과 같다.

**OXFF OXFF** 맨 앞에 위치한 두개의 OXFF는 Packet의 시작을 알리는 신호이다.

**ID** Dynamixel의 ID이다. Dynamixel의 ID는 0X00 ~ 0XFD까지 254개가 가능하다.

**Broadcasting ID** ID OXFE는 연결되어 있는 Dynamixel 전체를 지정하는 Broadcast ID이다. ID를 OXFE로 설정한 Packet은 연결된 모든 Dynamixel에게 유효하다. 그러므로 Broadcasting으로 전달된 Packet의 경우는 Status Packet이 return되지 않는다.

**LENGTH** Packet의 길이로서, 그 값은 “Parameter 개수(N) + 2” 이다.

**INSTRUCTION** Dynamixel에게 수행하라고 지시하는 명령.

**PARAMETER0…N** Instruction외에 추가 정보가 더 필요할 경우 사용된다.

**CHECK SUM** Check Sum의 계산 방법은 다음과 같다.

Check Sum = ~(ID + Length + Instruction + Parameter1 + … Parameter N)  
계산된 값이 255보다 클 경우 결과값의 하위 byte가 Checksum이다.  
~은 Not Bit 연산자이다.

### 3-3. Status Packet(Return Packet)

Status Packet은 Dynamixel이 Instruction Packet을 전송 받은 후 그 응답으로 Main Controller에게 return하는 Packet이다. Status Packet의 구조는 다음과 같다.

**OXFF OXFF ID LENGTH ERROR PARAMETER1 PARAMETER2 …PARAMETER N CHECK SUM**

Packet을 이루는 각 byte들의 의미는 다음과 같다.

**0xFF 0xFF**

맨 앞에 위치한 두개의 0xFF는 Packet의 시작을 알리는 신호이다.

**ID**

Packet을 Return하는 Dynamixel의 ID이다.

**LENGTH**

Status Packet의 길이로서, 그 값은 “Parameter 개수(N) + 2” 이다.

**ERROR**

그 다음에 위치한 **ERROR**는 Dynamixel이 동작 중에 발생된 Error 상태를 나타내며, 각 Bit별 의미는 다음 표와 같다.

Bit	명칭	내용
Bit 7	0	-
Bit 6	Instruction Error	정의되지 않은 Instruction이 전송된 경우. 또는 reg_write 명령없이 action명령이 전달된 경우 1로 설정됨
Bit 5	Overload Error	지정된 최대 Torque로 현재의 하중을 제어할 수 없을 때 1로 설정됨
Bit 4	Checksum Error	전송된 Instruction Packet의 Checksum이 맞지 않을 때 1로 설정됨
Bit 3	Range Error	사용범위를 벗어난 명령일 경우 1로 설정됨.
Bit 2	Overheating Error	Dynamixel 내부 온도가 Control Table에 설정된 동작 온도 범위를 벗어났을 때 1로 설정됨
Bit 1	Angle Limit Error	Goal Position이 CW Angle Limit ~ CCW Angle Limit 범위 밖의 값으로 Writing 되었을 때 1로 설정됨
Bit 0	Input Voltage Error	인가된 전압이 Control Table에 설정된 동작 전압 범위를 벗어났을 경우 1로 설정됨

**PARAMETER0…N**

추가 정보가 필요할 경우 사용된다.

**CHECK SUM**

Check Sum의 계산 방법은 다음과 같다.

$$\text{Check Sum} = \sim(\text{ID} + \text{Length} + \text{Instruction} + \text{Parameter1} + \dots + \text{Parameter N})$$

계산된 값이 255보다 클 경우 결과값의 하위 byte가 Checksum이다.

~은 Not Bit 연산자이다.

### 3-4. Control Table

Address	Item	Access	Initial Value
0(OX00)	Model Number(L)	RD	12(0x0C)
1(OX01)	Model Number(H)	RD	0(0x00)
2(OX02)	Version of Firmware	RD	?
3(OX03)	ID	RD,WR	1(0x01)
4(OX04)	Baud Rate	RD,WR	34(0x22)
5(OX05)	Return Delay Time	RD,WR	250(0xFA)
6(OX06)	CW Angle Limit(L)	RD,WR	0(0x00)
7(OX07)	CW Angle Limit(H)	RD,WR	0(0x00)
8(OX08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
9(OX09)	CCW Angle Limit(H)	RD,WR	3(0x03)
10(Ox0A)	(Reserved)	-	0(0x00)
11(OX0B)	the Highest Limit Temperature	RD,WR	85(0x55)
12(OX0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
13(OX0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
14(OX0E)	Max Torque(L)	RD,WR	255(0xFF)
15(OXF)	Max Torque(H)	RD,WR	3(0x03)
16(OX10)	Status Return Level	RD,WR	2(0x02)
17(OX11)	Alarm LED	RD,WR	4(0x04)
18(OX12)	Alarm Shutdown	RD,WR	4(0x04)
19(OX13)	(Reserved)	RD,WR	0(0x00)
20(OX14)	Down Calibration(L)	RD	?
21(OX15)	Down Calibration(H)	RD	?
22(OX16)	Up Calibration(L)	RD	?
23(OX17)	Up Calibration(H)	RD	?
24(OX18)	Torque Enable	RD,WR	0(0x00)
25(OX19)	LED	RD,WR	0(0x00)
26(OX1A)	CW Compliance Margin	RD,WR	0(0x00)
27(OX1B)	CCW Compliance Margin	RD,WR	0(0x00)
28(OX1C)	CW Compliance Slope	RD,WR	32(0x20)
29(OX1D)	CCW Compliance Slope	RD,WR	32(0x20)
30(OX1E)	Goal Position(L)	RD,WR	[Addr36]value
31(OX1F)	Goal Position(H)	RD,WR	[Addr37]value
32(OX20)	Moving Speed(L)	RD,WR	0
33(OX21)	Moving Speed(H)	RD,WR	0
34(OX22)	Torque Limit(L)	RD,WR	[Addr14] value
35(OX23)	Torque Limit(H)	RD,WR	[Addr15] value
36(OX24)	Present Position(L)	RD	?
37(OX25)	Present Position(H)	RD	?
38(OX26)	Present Speed(L)	RD	?
39(OX27)	Present Speed(H)	RD	?
40(OX28)	Present Load(L)	RD	?
41(OX29)	Present Load(H)	RD	?
42(OX2A)	Present Voltage	RD	?
43(OX2B)	Present Temperature	RD	?
44(OX2C)	Registered Instruction	RD,WR	0(0x00)
45(OX2D)	(Reserved)	-	0(0x00)
46(Ox2E)	Moving	RD	0(0x00)
47(Ox2F)	Lock	RD,WR	0(0x00)
48[0x30]	Punch(L)	RD,WR	32(0x20)
49[0x31]	Punch(H)	RD,WR	0(0x00)

**Control Table**

Control Table은 Dynamixel의 상태와 구동에 관한 Data로 구성되어 있다. Control Table의 값을 Writing함으로써 Dynamixel을 구동시키고, Control Table의 값을 Reading하여 Dynamixel의 상태를 파악할 수 있다.

**RAM and EEPROM**

RAM Area의 Data는 전원이 인가될 때마다 다시 초기값으로 설정된다. 그러나 EEPROM Area Data의 경우 값을 설정하면 전원이 Off되어도 그 값이 보존된다.

**Initial Value**

Control Table의 우측에 표시된 Initial Value는 EEPROM Area Data인 경우 Factory Default Value이고, RAM Area Data인 경우는 전원이 인가되었을 때 갖는 초기 값을 의미한다.

다음은 Control Table의 각 Address에 지정된 Data의 의미를 설명하였다. 이 내용은 Instruction만큼 의미 있는 내용이므로 사용자가 꼭 숙지하여야 한다.

**Address 0x00,0x01**

**Model Number.** AX-12 경우 값은 0X000C(12)이다.

**Address 0x02**

**Firmware Version.**

**Address 0x03**

**ID.** Dynamixel을 식별하기 위한 고유 번호이다. Link된 Dynamixel들은 서로 다른 ID가 할당되어야 한다.

**Address 0x04**

**Baud Rate.** 통신 Speed를 결정한다. 산출 공식은 다음과 같다.

$$\text{Speed(BPS)} = 2000000 / (\text{Address}4 + 1)$$

주요 Baud Rate별 Data Value

Address4	설정 BPS	목표 BPS	오차
1	1000000.0	1000000.0	0.000%
3	500000.0	500000.0	0.000%
4	400000.0	400000.0	0.000%
7	250000.0	250000.0	0.000%
9	200000.0	200000.0	0.000%
16	117647.1	115200.0	-2.124%
34	57142.9	57600.0	0.794%
103	19230.8	19200.0	-0.160%
207	9615.4	9600.0	-0.160%

**참고**

UART는 Baud Rate오차가 3%이내이면 통신에 지장이 없다.

**Address 0x05**

**Return Delay Time.** Instruction Packet 전송 후 Status Packet이 return되기까지 걸리는 지연 시간. 2uSec \* Address5값 만큼 지연된다.

**Address 0x06,0x07,0x08,0x09**

**Operating Angle Limit.** Dynamixel이 동작이 허용되는 Angle 구간을 설정한다. Goal Position은 CW Angle Limit <= Goal Position <= CCW Angle Limit의 범위 내에서 사용되어야 하며, 범위를 벗어날 경우 Angle Limit Error가 발생한다.

**Address 0x0B**

**the Highest Limit Temperature.** Dynamixel의 동작 제한 온도의 상한선. Dynamixel의 내부 온도가 이 값을 넘으면 Status Packet의 **ERROR**의 Over Heating Error Bit(Bit2)가 '1'로 return되고, Address 17, 18에 설정된 대로 Alarm이 실행된다. 값은 실제 섭씨 온도와 일치한다.

**Address 0x0C,0x0D**

**the Lowest (Highest) Limit Voltage.** Dynamixel의 동작 전압 범위의 상한선과 하한선을 지정하는 Data이다. Present Voltage(Address 42)가 이 범위를 벗어날 경우 Status Packet의 **ERROR**의 Voltage Range Error Bit(Bit0)가 '1'로 return되고 Address 17, 18에서 설정된 대로 Alarm이 실행된다. 실제 전압의 10배를 값으로 한다. 예를 들어 Address 12의 값이 80일 경우 동작 하한선 전압을 8V로 설정한 것이다.

**Address 0x0E,0x0F, 0x22,0x23**

**Max Torque.** Dynamixel의 최대 Torque 출력값을 설정한다. 이 값을 '0'으로 설정할 경우 Torque가 없는 Free Run 상태가 된다. Max Torque(Torque Limit)는 ROM(Address 0x0E, 0x0F)과 RAM(Address 0x22, 0x23) 두 곳에 할당이 되어 있는데, P 전원이 On될 때 EEPROM의 값이 RAM으로 복사된다. Dynamixel의 Torque는 RAM에 위치한 값(Address 0x22, 0x23)에 의해 제한된다.

**Address 0X10**

**Status Return Level.** Instruction Packet이 전송된 후 Dynamixel이 Status Packet을 Return해 줄지 여부를 결정한다.

Address16	Status Packet의 Return
0	모든 Instruction에 대해 Return하지 않음
1	READ_DATA 명령에 대해서만 Return함
2	모든 Instruction에 대해 Return함

Broadcast ID(0xFE)의 instruction packet의 경우는 Address 0x10의 값에 상관없이 Status Packet이 return되지 않는다.

**Address 0X11**

**Alarm LED.** Error가 발생했을 때, 해당 Bit가 1로 설정되어 있으면 LED가 깜빡인다.

Bit	기 능
Bit 7	0
Bit 6	1로 설정해 놓으면 Instruction Error발생시 LED가 깜빡임
Bit 5	1로 설정해 놓으면 Overload Error발생시 LED가 깜빡임
Bit 4	1로 설정해 놓으면 Checksum Error발생시 LED가 깜빡임
Bit 3	1로 설정해 놓으면 Range Error발생시 LED가 깜빡임
Bit 2	1로 설정해 놓으면 Overheating Error발생시 LED가 깜빡임
Bit 1	1로 설정해 놓으면 Angle Limit Error발생시 LED가 깜빡임
Bit 0	1로 설정해 놓으면 Input Voltage Error발생시 LED가 깜빡임

각 Bit의 기능은 ‘OR’ 의 논리로 작동된다. 즉, 0X05로 설정되었을 경우 Input Voltage Error가 발생해도 LED는 깜빡이고, Overheating Error가 발생해도 LED가 깜빡인다. Error가 발생했다가 정상 상황으로 복귀하면 2초 후에 LED는 깜빡임을 멈춘다.

**Address 0X12**

**Alarm Shutdown.** Error가 발생했을 때 해당 Bit가 1로 설정되어 있을 경우 Dynamixel은 Torque off된다.

Bit	기 능
Bit 7	0
Bit 6	1로 설정해 놓으면 Instruction Error발생시 Torque Off
Bit 5	1로 설정해 놓으면 Overload Error발생시 Torque Off
Bit 4	1로 설정해 놓으면 Checksum Error발생시 Torque Off
Bit 3	1로 설정해 놓으면 Range Error발생시 Torque Off
Bit 2	1로 설정해 놓으면 Overheating Error발생시 Torque Off
Bit 1	1로 설정해 놓으면 Angle Limit Error발생시 Torque Off
Bit 0	1로 설정해 놓으면 Input Voltage Error발생시 Torque Off

각 Bit의 기능은 Alarm LED와 마찬가지로 ‘OR’ 의 논리로 작동된다. 그러나 Alarm LED와는 달리 Error가 발생했다가 정상 상황으로 복귀해도 Torque OFF상태는 계속 유지된다. Shutdown상태에서 벗어나려면 Torque Enable(Address0X18)를 1로 재설정해야 한다.

**Address 0x14~0x17**

**Calibration.** Potentio Meter의 제품간 편차를 보상하기 위한 Data이다. 사용자가 변경할 수 없는 영역이다.

이어질 Address 0x18부터는 RAM 영역이다.

**Address 0x18**

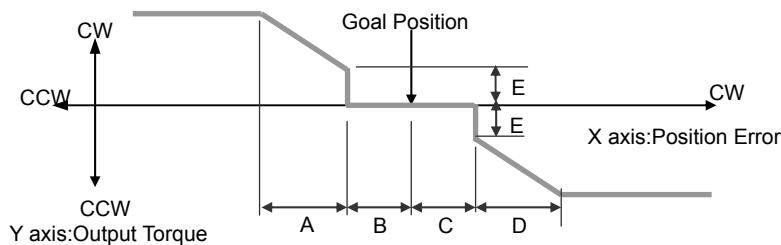
**Torque Enable.** Digital Mode에서 Dynamixel에 전원을 인가하면 Torque가 발생하지 않는 Free Run상태이다. Address 0x18에 1을 설정하면 Torque Enable상태로 된다.

**Address 0x19**

**LED.** 1로 설정되어 있으면 LED가 켜지고 0으로 설정되어 있으면 LED가 꺼진다.

**Address 0x1A~0x1D**

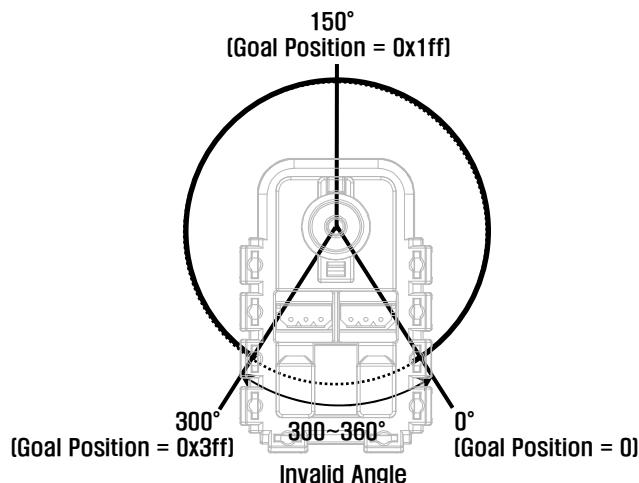
**Compliance Margin과 Slope.** Dynamixel에서는 Margin과 Slope를 설정하여 Compliance를 조절한다. Compliance를 잘 활용하면 충격 흡수를 하는 효과를 낼 수 있다. 다음의 Position Error에 따른 출력 곡선에서 A,B,C,D의 길이가 Compliance값이다.



- A : CCW Compliance Slope(Address0x1D)
- B : CCW Compliance Margin(Address0x1B)
- C : CW Compliance Margin(Address0x1A)
- D : CW Compliance Slope (Address0x1C)
- E : Punch(Address0x30,31)

**Address 0x1E,0x1F**

**Goal Position.** Dynamixel이 이동하고자 하는 위치. 값 0x3ff로 설정하면  $300^\circ$ 로 이동한다.



**Address 0x20,0x21** **Moving Speed.** Goal Position으로 이동하는 속도. 최대값인 0x3ff로 설정되면 전압공급이 충분할 경우 114RPM의 속도로 움직인다.(속도가 1로 Setting될 경우가 최저속이며, 0으로 Setting되어 있을 경우는 현재 인가 전압상에서 낼 수 있는 최대 속도로 움직임, 즉 속도제어를 하지 않음)

**Address 0x24,0x25** **Present Position.** Dynamixel의 현재 위치.

**Address 0x26,0x27** **Present Speed.** Dynamixel의 현재 속도 Data.

**Address 0x28,0x29** **Present Load.** Dynamixel의 현재 구동하는 Load의 크기. Bit10은 Load가 걸려있는 방향이다.

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Load Direction										Load Value

Load Direction = 0 : CCW Load, Load Direction = 1: CW Load

**Address 0x2A** **Present Voltage.** Dynamixel에 현재 인가되고 있는 전압. 이 값은 실제 전압의 10배이다. 즉 10V일 경우 100(0x64)이 읽혀진다.

**Address 0x2B** **Present Temperature.** Dynamixel 내부의 섭씨 온도.

**Address 0x2C** **Registered Instruction.** REG\_WRITE명령에 의해 명령이 등록되어 있을 때 1로 설정되고, Action명령에 의해 등록된 명령이 수행 완료된 후에는 0으로 변한다.

**Address 0x2E** **Moving.** Dynamixel이 자신의 동력에 의한 Moving상태일 때 1로 Setting된다.

**Address 0x2F** **Lock.** 1로 Setting되면 Address 0X18~ Address0x23 의 값만 Writing 할 수 있고 나머지 영역은 Writing이 금지 된다. 한번 Lock되면 Power Off로만 Unlock할 수 있다.

**Address 0x30,0x31** **Punch.** 구동시에 모터에 공급되는 최소 전류량. 초기값은 0x20이며 최고 0x3ff까지 설정할 수 있다.

**Endless Turn** CW Angle Limit과 CCW Angle Limit이 모두 0으로 설정되면 Goal Speed를 설정하여 Endless Turn을 구현할 수 있다. Endless Turn은 바퀴 구동 등에 사용된다.

**Goal Speed Setting**

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Turn Direction										Speed Value

Turn Direction = 0 : CCW Direction Turn, Load Direction = 1: CW Direction Turn

#### Range

각 Data들은 유효한 범위가 정해져 있다. 이를 벗어난 Write명령이 전송될 경우 Error가 return된다. 아래 표에 사용자가 Write할 수 있는 Data의 길이, 그리고 범위를 정리하였다. 16bit Data는 (L)과 (H), 두 byte로 표시된다. 이 두 byte는 하나의 Instruction Packet으로 동시에 Write되어야 한다.

Write Address	Writing Item	Length (bytes)	Min	Max
3(0X03)	ID	1	0	253(0xfd)
4(0X04)	Baud Rate	1	0	254(0xfe)
5(0X05)	Return Delay Time	1	0	254(0xfe)
6(0X06)	CW Angle Limit	2	0	1023(0x3ff)
8(0X08)	CCW Angle Limit	2	0	1023(0x3ff)
11(0X0B)	the Highest Limit Temperature	1	0	150(0x96)
12(0X0C)	the Lowest Limit Voltage	1	50(0x32)	250(0xfa)
13(0X0D)	the Highest Limit Voltage	1	50(0x32)	250(0xfa)
14(0X0E)	Max Torque	2	0	1023(0x3ff)
16(0X10)	Status Return Level	1	0	2
17(0X11)	Alarm LED	1	0	127(0x7f)
18(0X12)	Alarm Shutdown	1	0	127(0x7f)
19(0X13)	(Reserved)	1	0	1
24(0X18)	Torque Enable	1	0	1
25(0X19)	LED	1	0	1
26(0X1A)	CW Compliance Margin	1	0	254(0xfe)
27(0X1B)	CCW Compliance Margin	1	0	254(0xfe)
28(0X1C)	CW Compliance Slope	1	1	254(0xfe)
29(0X1D)	CCW Compliance Slope	1	1	254(0xfe)
30(0X1E)	Goal Position	2	0	1023(0x3ff)
32(0X20)	Moving Speed	2	0	1023(0x3ff)
34(0X22)	Torque Limit	2	0	1023(0x3ff)
44(0X2C)	Registered Instruction	1	0	1
47(0X2F)	Lock	1	1	1
48(0X30)	Punch	2	0	1023(0x3ff)

[Control Table Data Range and Length for Writing]

## 4. Instruction Set과 그 사용 예

다음과 같은 종류의 Instruction이 있다.

Instruction	Function	Value	Number of Parameter
PING	수행 내용 없음. Dynamixel이 Status Packet을 return받고자 할 경우 사용.	0x01	0
READ DATA	Control Table의 값을 읽는 명령.	0x02	2
WRITE DATA	Control Table에 값을 쓰는 명령.	0x03	2 ~
REG WRITE	WRTE_DATA와 내용은 유사하나, 대기상태로 있다가 ACTION명령이 도착하면 Write된다.	0x04	2 ~
ACTION	REG_WRITE으로 등록된 동작을 시작하라는 명령	0x05	0
RESET	Dynamixel내의 Control Table값을 Factory Default Value로 바꾼다.	0x06	0
SYNC WRITE	한번에 여러 개의 Dynamixel들을 동시에 제어하고자 할 때 사용되는 명령	0x83	4~

### 4-1. WRITE\_DATA

Function	Dynamixel 내부 Control Table에 Data를 쓰는 명령
Length	N+3 (Writing Data가 N개일 경우)
Instruction	0X03
Parameter1	Data를 쓰고자 하는 곳의 시작 Address
Parameter2	쓰고자 하는 첫번째 Data
Parameter3	쓰고자 하는 두번째 Data
Parameter N+1	쓰고자 하는 N번째 Data

#### **Example 1** 연결된 Dynamixel의 ID를 1로 설정하고자 하는 경우

Control Table의 Address 30의 1을 Writing한다. ID는 Broadcasting ID(0xFE)로 전송하기로 한다.



Broadcast ID(0xFE)로 전송되었으므로 Status Packet은 return되지 않는다.

## 4-2. READ\_DATA

Function	Dynamixel 내부 Control Table의 Data를 읽는 명령
Length	0x04
Instruction	0x02
Parameter1	Read하고자 하는 data의 시작 Address
Parameter2	Read하고자 하는 data의 길이

### Example 2

#### ID가 1인 Dynamixel의 현재 내부 온도를 읽고자 하는 경우

Control Table의 Address 0x2B값에서 1byte를 읽는다.

Instruction Packet : 0xFF 0xFF 0x01 0x04 0x02 0x2B 0x01 0xCC



이에 대하여 return되는 Status Packet은 다음과 같다.

Status Packet : 0xFF 0xFF 0x01 0x03 0x00 0x20 0xDB



읽혀진 Data 값은 0x20이다. 현재 Dynamixel의 내부 온도는 약 32°C (0x20)이다.

## 4-3. REG\_WRITE과 ACTION

### 4-3-1. REG\_WRITE

Function	REG_WRITE명령은 WRITE_DATA명령과 기능은 유사하나, 명령이 수행되는 시점이 다
----------	---

르다. Instruction Packet이 도착하면 그 값이 Buffer에 저장되어 있고 Write 동작은 대기 상태로 남아 있다. 이때, Registered Instruction(Address 0x2C)이 1로 설정된다. 이후에 Action Instruction Packet이 도착하면 바로소 등록되어 있던 Write 명령이 실행된다.

Length	N+3 (Writing Data가 N개일 경우)
Instruction	0X04
Parameter1	Data를 쓰고자 하는 곳의 시작 Address
Parameter2	쓰고자 하는 첫번째 Data
Parameter3	쓰고자 하는 두번째 Data
Parameter N+1	쓰고자 하는 N번째 Data

#### 4-3-2. ACTION

Function	REG_WRITE로 등록된 WRITE 작업을 수행하라는 명령
Length	0X02
Instruction	0X05
Parameter	NONE

ACTION 명령은 여러 개의 Dynamixel을 동시에 정확히 움직여야 하는 경우 유용하다. 여러 개의 구동장치를 통신에 의해 제어할 때, 맨 처음 명령을 전달 받는 구동장치 와 맨 마지막에 명령을 전달 받는 구동장치는 구동 시점에 약간의 시간 차이가 발생한다. Dynamixel에서는 ACTION Instruction을 사용하여 이러한 문제를 해결하였다.

Broadcasting	두개 이상의 Dynamixel에 ACTION 명령을 전송할 경우 Broadcast ID(0xFE)를 사용하여야 하는데, 이 때 Packet은 Return되지 않는 점에 유의한다.
--------------	---

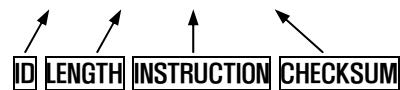
#### 4-4. PING

Function	아무 것도 지시하지 않는다. 단지 Status Packet을 받고자 할 때나 특정 ID를 갖는 Dynamixel의 존재를 확인하기 위해 사용된다.
Length	0X02
Instruction	0X01
Parameter	NONE

**Example 3**

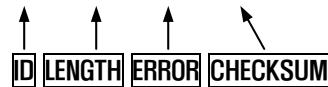
ID가 1인 Dynamixel의 Status Packet을 얻고 싶을 때

Instruction Packet : 0xFF 0xFF 0X01 0X02 0X01 0XFB



이에 대하여 return되는 Status Packet은 다음과 같다.

Status Packet : 0xFF 0xFF 0X01 0X02 0X00 0XFC



Broadcasting ID가 지정되거나 Status Return Level(Address16)이 0이더라도, PING Instruction에 대해서는 무조건 Status Packet을 return한다.

#### 4-5. RESET

Function

Dynamixel을 Factory Default 상태로 Control Table을 되돌려 놓는다.

Length

0X02

Instruction

0X06

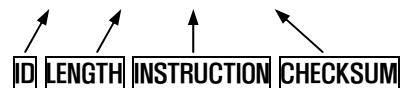
Parameter

NONE

**Example 4**

ID가 0인 Dynamixel을 RESET하고자 할 경우

Instruction Packet : 0xFF 0xFF 0X00 0X02 0X06 0XF7



이에 대하여 return되는 Status Packet은 다음과 같다.

Status Packet : 0xFF 0xFF 0X00 0X02 0X00 0XFD



RESET명령 수행 이후엔 Dynamixel의 ID가 1로 바뀌어 있음을 유의한다.

## 4-6. SYNC WRITE

**Function** 한번의 Instruction Packet 전송으로 여러 개의 Dynamixel들을 동시에 제어하고자 할 때 사용되는 명령어이다. Sync Write 명령을 사용하면 여러 개의 명령을 한번에 전달 하므로 다수의 Dynamixel을 제어할 때 통신 시간을 줄여든다. 단, 각 Dynamixel들에 writing하고자 하는 Control Table의 Address와 Length가 모두 동일해야 SYNC WRITE 명령을 사용할 수 있다. 또한 ID는 Broadcasting ID로 전송되어야 한다.

<b>ID</b>	0XFE
<b>Length</b>	(L+1) X N + 4 (L:Dynamixel별 Data Length, N:Dynamixel의 개수)
<b>Instruction</b>	0X83
<b>Parameter1</b>	Data를 쓰고자 하는 곳의 시작 Address
<b>Parameter2</b>	쓰고자 하는 Data의 길이 (L)
<b>Parameter3</b>	첫번째 Dynamixel의 ID
<b>Parameter4</b>	첫번째 Dynamixel의 첫번째 Data
<b>Parameter5</b>	첫번째 Dynamixel의 두번째 Data
...	
<b>Parameter L+3</b>	첫번째 Dynamixel의 L번째 Data
<b>Parameter L+4</b>	두번째 Dynamixel의 ID
<b>Parameter L+5</b>	두번째 Dynamixel의 첫번째 Data
<b>Parameter L+6</b>	두번째 Dynamixel의 두번째 Data
...	
<b>Parameter 2L+4</b>	두번째 Dynamixel의 L번째 Data
....	

{ 첫번째 Dynamixel에 대한 Data }

{ 두번째 Dynamixel에 대한 Data }

### Example 5

#### 4개의 Dynamixel에 대하여 각각 다음과 같은 위치와 속도를 설정하고자 하는 경우

ID 0인 Dynamixel : 0x010위치로 속도 0x150으로 이동

ID 1인 Dynamixel : 0x220위치로 속도 0x360으로 이동

ID 2인 Dynamixel : 0x030위치로 속도 0x170으로 이동

ID 3인 Dynamixel : 0x220위치로 속도 0x380으로 이동

Instruction Packet : 0XFF 0XFF 0XFE 0X18 0X83 0X1E 0X04 0X00 0X10 0X00 0X50  
0X01 0X01 0X20 0X02 0X60 0X03 0X02 0X30 0X00 0X70 0X01 0X03 0X20 0X02  
0X80 0X03 0X12

Broadcasting ID로 전송하므로 Status Packet은 return되지 않는다.

## 5. Example

Dynamixel은 Reset상태인 ID = 1, Baudrate = 57142BPS이라고 가정하고 예제를 설명한다.

**Example 6**

ID가 1인 Dynamixel의 Model Number와 Firmware Version을 읽는다.

**Instruction Packet**

Instruction = READ\_DATA, Address = 0x00, Length = 0x03

**Communication**

->[Dynamixel]:FF FF 01 04 02 00 03 F5 (LEN:008)  
<-[Dynamixel]:FF FF 01 05 00 74 00 08 7D (LEN:009)

**Status Packet Result**

Model Number = 116(0x74)(DX-116인 경우임) Firmware Version = 0x08

**Example 7**

ID가 1인 Dynamixel의 ID를 0으로 변경한다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x03, DATA = 0x00

**Communication**

->[Dynamixel]:FF FF 01 04 03 03 00 F4 (LEN:008)  
<-[Dynamixel]:FF FF 01 02 00 FC (LEN:006)

**Status Packet Result**

NO ERROR

**Example 8**

Dynamixel의 Baud Rate를 1M bps로 변경한다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x04, DATA = 0x01

**Communication**

->[Dynamixel]:FF FF 00 04 03 04 01 F3 (LEN:008)  
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result**

NO ERROR

**Example 9**

ID가 0인 Dynamixel의 Return Delay Time을 4uSec로 재설정한다.

Return Delay Time Value 10l 2uSec에 해당한다.

**Instruction Packet** Instruction = WRITE\_DATA, Address = 0x05, DATA = 0x02

**Communication**  
 ->[Dynamixel]:FF FF 00 04 03 05 02 F1 (LEN:008)  
 <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result** NO ERROR

Return Delay Time은 Main Controller의 허용범위 내에서 최소값으로 설정하는 것이 좋다.

**Example 10** ID가 0인 Dynamixel의 동작각을 0~150°로 제한한다.

CCW Angle Limit가 0x3ff일 경우 300°이므로 150°에 해당하는 값은 0x1ff이다.

**Instruction Packet** Instruction = WRITE\_DATA, Address = 0x08, DATA = 0xff, 0x01

**Communication**  
 ->[Dynamixel]:FF FF 00 05 03 08 FF\_01 EF (LEN:009)  
 <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result** NO ERROR

**Example 11** ID가 0인 Dynamixel의 동작온도 상한선을 80°로 재설정한다.

**Instruction Packet** Instruction = WRITE\_DATA, Address = 0x0B, DATA = 0x50

**Communication**  
 ->[Dynamixel]:FF FF 00 04 03 0B 50 9D (LEN:008)  
 <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result** NO ERROR

**Example 12** ID가 0인 Dynamixel의 동작전압을 10V ~ 17V로 설정한다.

10V는 100(0x64), 17V는 170(0xAA)의 값으로 표시된다.

**Instruction Packet** Instruction = WRITE\_DATA, Address = 0x0C, DATA = 0x64, 0xAA

**Communication**  
 ->[Dynamixel]:FF FF 00 05 03 0C 64 AA DD (LEN:009)  
 <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result** NO ERROR

**Example 13**

ID가 0인 Dynamixel이 항상 Torque를 최대값의 50%만 발휘하도록 한다.  
ROM영역에 위치한 MAX Torque의 값을 최대값인 0x3ff의 50%인 0x1ff로 설정한다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x0E, DATA = 0xff, 0x01

**Communication**

->[Dynamixel]:FF FF 00 05 03 0E FF 01 E9 (LEN:009)  
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result**

NO ERROR

전원을 Off한 후 다시 공급해야 Max Torque값을 조정한 효과를 확인할 수 있다.

**Example 14**

ID가 0인 Dynamixel이 항상 Status Packet을 Return하지 않도록 한다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x10, DATA = 0x00

**Communication**

->[Dynamixel]:FF FF 00 04 03 10 00 E8 (LEN:008)  
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result**

NO ERROR

Status Packet은 다음번 Instruction부터 Return되지 않는다.

**Example 15**

동작 온도가 설정된 한계 온도보다 높은 값을 가질 경우 LED를 깜빡이고, Shutdown (Torque off)되도록 Alarm을 설정한다.  
Overheating Error는 Bit 20이므로 Alarm값을 0x04로 설정한다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x11, DATA = 0x04, 0x04

**Communication**

->[Dynamixel]:FF FF 00 05 03 11 04 04 DE (LEN:009)  
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result**

NO ERROR

**Example 16**

ID가 0인 Dynamixel의 LED를 켜고, Torque를 Enable시킨다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x18, DATA = 0x01, 0x01

**Communication**

->[Dynamixel]:FF FF 00 05 03 18 01 01 DD (LEN:009)

<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result**

NO ERROR

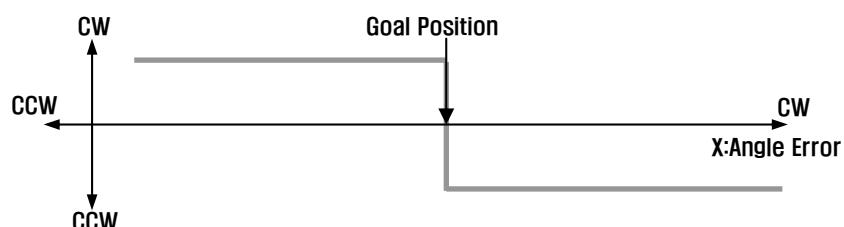
손으로 Dynamixel의 축을 만져보면 Torque Enable상태를 확인할 수 있다.

**Example 17**

ID가 0인 Dynamixel을 Compliance Margin = 1, Compliance Slope를 0x40으로 설정한다.

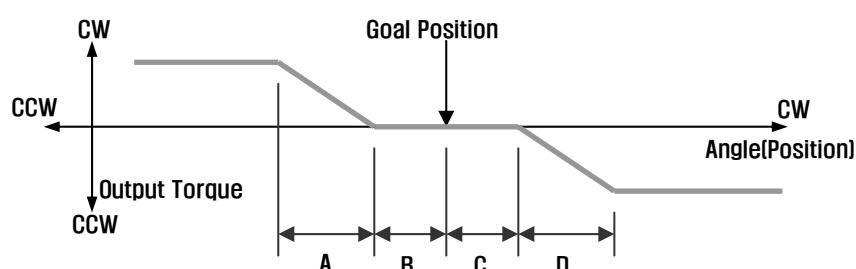
**Compliance**

Angle Error와 Torque Output은 다음과 같은 Graph로 나타낼 수 있다.



위치가 Goal Position으로부터 CW방향으로 조금만 벗어나도 CCW방향으로 큰 Torque를 발생시켜 Goal Position으로 위치를 수렴시키고 있다. 그러나 관성이 고려되어야 하므로 실제 제어방식은 이와 다소 차이가 있다.

위의 예제에서 제시한 조건을 이러한 방식의 Graph로 표시해 보면 다음과 같다.



A : CCW Compliance Slope(Address 0x1D) = 0x40(약 18.8° )

B : CCW Compliance Margin(Address 0x1B) = 0x01 (약 0.29° )

C : CW Compliance Margin(Address 0x01A) = 0x01[약 0.29° ]

D : CW Compliance Slope(Address 0x1C) = 0x40 [약 18.8° ]

**Instruction Packet** Instruction = WRITE\_DATA, Address = 0x1A, DATA = 0x01, 0x01, 0x40, 0x40

**Communication** →[Dynamixel]:FF FF 00 07 03 1A 01 01 40 40 59 (LEN:011)

←[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result** NO ERROR

Compliance Slope는  $2^n$ [n은 정수]를 경계로 효과가 변화한다. 즉 Compliance값 0x11~0x20은 그 효과가 동일하다.

#### Example 18

ID가 0인 Dynamixel을 57RPM의 속도로 Position 180°에 위치시킨다.

Address 0x1E(Goal Position) = 0x200, Address 0x20(Moving Speed) = 0x200으로 설정한다.

**Instruction Packet** Instruction = WRITE\_DATA, Address = 0x1E, DATA = 0x00, 0x02, 0x00, 0x02

**Communication** →[Dynamixel]:FF FF 00 07 03 1E 00 02 00 02 D3 (LEN:011)

←[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result** NO ERROR

#### Example 19

ID가 0인 Dynamixel은 Position 0°에, ID가 1인 Dynamixel은 Position 300°에, 위치시킨다. 단 두 Dynamixel은 똑 같은 시점에 움직이기 시작하도록 한다.

WRITE\_DATA 명령을 사용하면 두 Dynamixel을 똑 같은 시점에 출발시킬 수 없다. 그러므로 REG\_WRITE와 ACTION을 사용한다.

**Instruction Packet** ID=0, Instruction = REG\_WRITE, Address = 0x1E, DATA = 0x00, 0x00

ID=1, Instruction = REG\_WRITE, Address = 0x1E, DATA = 0xff, 0x03

ID=0xfe(Broadcasting ID), Instruction = ACTION,

**Communication** →[Dynamixel]:FF FF 00 05 04 1E 00 00 D8 (LEN:009)

←[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

→[Dynamixel]:FF FF 01 05 04 1E FF 03 D5 (LEN:009)

←[Dynamixel]:FF FF 01 02 00 FC (LEN:006)

→[Dynamixel]:FF FF FE 02 05 FA (LEN:006)

←[Dynamixel]: //No return packet against broadcasting ID

**Status Packet Result** NO ERROR

**Example 20**

ID가 0인 Dynamixel을 Address0x18 ~ Address0x23의 값 이외에는 변경할 수 없도록 한다.

Address 0x2F(Lock)을 1로 설정한다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x2F, DATA = 0x01

**Communication**

->[Dynamixel]:FF FF 00 04 03 2F 01 C8 (LEN:008)

<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result**

NO ERROR

Lock이 되면 전원을 재거해야만 unlock할 수 있다.

Lock된 상태에서 다른 데이터를 Access하면 Error가 Return된다.

->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)

<-[Dynamixel]:FF FF 00 02 08 F5 (LEN:006)

Range Error

**Example 21**

ID가 0인 Dynamixel의 최소 출력값(Punch)을 0x40으로 한다.

**Instruction Packet**

Instruction = WRITE\_DATA, Address = 0x30, DATA = 0x40, 0x00

**Communication**

->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)

<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

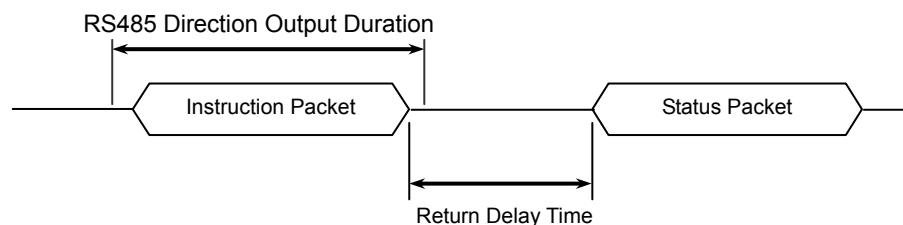
**Status Packet Result**

NO ERROR

## Appendix

### Half duplex UART

Half duplex UART란 동시에 TxD, RxD가 동시에 수행될 수 없는 Serial 통신방식을 의미한다. 보통 하나의 BUS에 여러 개의 통신 장치를 연결할 경우에 사용하는 방식이다. 여러 개의 장치가 동일한 BUS에 연결되어 있으므로 하나의 장치가 송신하는 동안 그 외의 다른 모든 장치들은 입력상태이어야 한다. Dynamixel을 제어하는 Main Controller는 통신방향을 입력으로 설정하고 있다가 Instruction Packet을 전송하는 동안만 통신 방향을 출력으로 설정한다.



### Return Delay Time

Dynamixel이 Instruction Packet을 받은 후 Status Packet을 return하는데 걸리는 시간을 의미한다. Default Value는 160uSec이며, Control Table Address 5를 설정하여 Return Delay Time을 변경할 수 있다. Main Controller는 Instruction Packet 전송 후 Return Delay time 구간 안에서 Direction Port를 입력 상태로 전환해야 한다.

### Tx,Rx Direction

Half Duplex UART에서는 송신이 끝나는 Timing을 잘 맞춰서 Direction을 수신Mode로 바꾸어야 주어야 한다. CPU에서는 일반적으로 UART\_STATUS를 표시해주는 REGISTER내에 다음과 같은 의미의 BIT가 있다.

**TXD\_BUFFER\_READY\_BIT** : Transmission DATA를 Buffer에 적재할 수 있는 상태임을 뜻한다. 주의할 점은 이 상태는 SERIAL TX BUFFER가 비어 있다는 의미이지, 이전에 전송한 데이터가 모두 CPU 밖으로 배출된 상태를 의미하는 것은 아니다.

**TXD\_SHIFT\_REGISTER\_EMPTY\_BIT** : Transmission Data가 모두 CPU밖으로 배출되었을 때 SET된다.

**TXD\_BUFFER\_READY\_BIT**의 경우는 Serial 통신에서 한 Byte를 송신할 때 사용되며 그에는 다음과 같다.

```
TxDByte(byte bData)
{
    while(!TXD_BUFFER_READY_BIT); //wait until data can be loaded.
    SerialTxDBuffer = bData;      //data load to TxD buffer
}
```

Direction을 전환하는 시점에서는 TXD\_SHIFT\_REGISTER\_EMPTY\_BIT를 확인해야 한다.

다음은 Instruction packet을 전송하는 예제 프로그램이다.

```

LINE 1 DIRECTION_PORT = TX_DIRECTION;
LINE 2 TxDByte(0xff);
LINE 3 TxDByte(0xff);
LINE 4 TxDByte(bID);
LINE 5 TxDByte(bLength);
LINE 6 TxDByte(bInstruction);
LINE 7 TxDByte(Parameter0); TxDByte(Parameter1); ...
LINE 8 DisableInterrupt(); // Interrupt should be disable
LINE 9 TxDByte(Checksum); //last TxD
LINE 10 while(!TXD_SHIFT_REGISTER_EMPTY_BIT); //Wait till last data bit has been sent
LINE 11 DIRECTION_PORT = RX_DIRECTION; //Direction change to RXD
LINE 12 EnableInterrupt(); // enable interrupt again

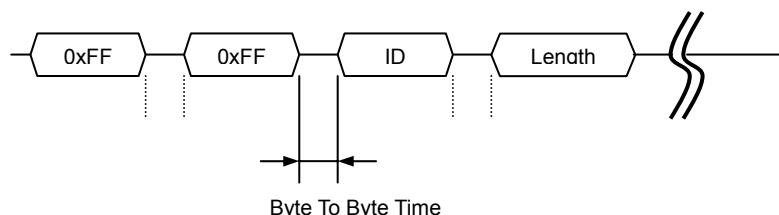
```

주의할 부분은 LINE 8부터 LINE12이다..

LINE 8이 필요한 이유는 그 시점에서 Interrupt가 발생하여 Return Delay Time 보다 긴 시간동안 Interrupt routine이 수행될 경우 Status Packet의 앞부분이 손상되기 때문이다.

#### Byte to Byte Time

Instruction Packet을 전송할 때 Byte와 Byte사이의 Delay time을 의미하는데, 이 시간이 100mSec가 넘을 경우 Dynamixel은 전송 장애가 발생한 것으로 간주하고, 다시 Packet의 header[0xff 0xff]를 기다린다.



다음은 Atmega128로 Dynamixel을 Access하는 Example.c의 source이다.

**C Language Example : Dynamixel access with Atmega128**

```

/*
 * The Example of Dynamixel Evaluation with Atmega128
 * Date : 2005.5.11
 * Author : BS KIM
 */

/*
 * included files
 */
#define ENABLE_BIT_DEFINITIONS
#include <iio.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#define cbi(REG8,BITNUM) REG8 &= ~_BV(BITNUM)
#define sbi(REG8,BITNUM) REG8 |= _BV(BITNUM)

typedef unsigned char byte;
typedef unsigned int word;
#define ON 1
#define OFF 0
#define _ON 0
#define _OFF 1

//--- Control Table Address ---
//EEPROM AREA
#define P_MODEL_NUMBER_L 0
#define P_MODEL_NUMBER_H 1
#define P_VERSION 2
#define P_ID 3
#define P_BAUD_RATE 4
#define P_RETURN_DELAY_TIME 5
#define P_CW_ANGLE_LIMIT_L 6
#define P_CW_ANGLE_LIMIT_H 7
#define P_CCW_ANGLE_LIMIT_L 8
#define P_CCW_ANGLE_LIMIT_H 9
#define P_SYSTEM_DATA2 10
#define P_LIMIT_TEMPERATURE 11
#define P_DOWN_LIMIT_VOLTAGE 12
#define P_UP_LIMIT_VOLTAGE 13
#define P_MAX_TORQUE_L 14
#define P_MAX_TORQUE_H 15
#define P_RETURN_LEVEL 16
#define P_ALARM_LED 17
#define P_ALARM_SHUTDOWN 18
#define P_OPERATING_MODE 19
#define P_DOWN_CALIBRATION_L 20
#define P_DOWN_CALIBRATION_H 21
#define P_UP_CALIBRATION_L 22
#define P_UP_CALIBRATION_H 23

#define P_TORQUE_ENABLE (24)
#define P_LED (25)
#define P_CW_COMPLIANCE_MARGIN (26)
#define P_CCW_COMPLIANCE_MARGIN (27)
#define P_CW_COMPLIANCE_SLOPE (28)
#define P_CCW_COMPLIANCE_SLOPE (29)
#define P_GOAL_POSITION_L (30)
#define P_GOAL_POSITION_H (31)
#define P_GOAL_SPEED_L (32)
#define P_GOAL_SPEED_H (33)
#define P_TORQUE_LIMIT_L (34)
#define P_TORQUE_LIMIT_H (35)
#define P_PRESENT_POSITION_L (36)
#define P_PRESENT_POSITION_H (37)
#define P_PRESENT_SPEED_L (38)
#define P_PRESENT_SPEED_H (39)
#define P_PRESENT_LOAD_L (40)
#define P_PRESENT_LOAD_H (41)
#define P_PRESENT_VOLTAGE (42)

#define P_PRESENT_TEMPERATURE (43)
#define P_REGISTERED_INSTRUCTION (44)
#define P_PAUSE_TIME (45)
#define P_MOVING (46)
#define P_LOCK (47)
#define P_PUNCH_L (48)
#define P_PUNCH_H (49)

//--- Instruction ---
#define INST_PING 0x01
#define INST_READ 0x02
#define INST_WRITE 0x03
#define INST_REG_WRITE 0x04
#define INST_ACTION 0x05
#define INST_RESET 0x06
#define INST_DIGITAL_RESET 0x07
#define INST_SYSTEM_READ 0x0C
#define INST_SYSTEM_WRITE 0x0D
#define INST_SYNC_WRITE 0x83
#define INST_SYNC_REG_WRITE 0x84

#define CLEAR_BUFFER gbRxBufferReadPointer = gbRxBufferWritePointer
#define DEFAULT_RETURN_PACKET_SIZE 6
#define BROADCASTING_ID 0xfe

#define Tx8D Tx81
#define Rx8D Rx81

//Hardware Dependent Item
#define DEFAULT_BAUD_RATE 34 //57600bps at 16MHz

////// For CM-5
#define RS485_TXD PORTE &= ~_BV(PE3),PORTE |= _BV(PE2)
//_485_DIRECTION = 1
#define RS485_RXD PORTE &= ~_BV(PE2),PORTE |= _BV(PE3)
//PORT_485_DIRECTION = 0
/*
////// For CM-2
#define RS485_TXD PORTE |= _BV(PE2); //_485_DIRECTION = 1
#define RS485_RXD PORTE &= ~_BV(PE2); //PORT_485_DIRECTION = 0
*/
//##define TXDO_FINISH UCSROA,6 //This bit is for checking TxD Buffer
in CPU is empty or not.
//##define TXD1_FINISH UCSR1A,6

#define SET_TxD0_FINISH sbi(UCSROA,6)
#define RESET_TxD0_FINISH cbi(UCSROA,6)
#define CHECK_TxD0_FINISH bit_is_set(UCSROA,6)
#define SET_TxD1_FINISH sbi(UCSR1A,6)
#define RESET_TxD1_FINISH cbi(UCSR1A,6)
#define CHECK_TxD1_FINISH bit_is_set(UCSR1A,6)

#define RX_INTERRUPT 0x01
#define TX_INTERRUPT 0x02
#define OVERFLOW_INTERRUPT 0x01
#define SERIAL_PORT0 0
#define SERIAL_PORT1 1
#define BIT_RS485_DIRECTION0 0x08 //Port E
#define BIT_RS485_DIRECTION1 0x04 //Port E

#define BIT_ZIGBEE_RESET PD4 //out : default 1 //PORTD
#define BIT_ENABLE_RXD_LINK_PC PD5 //out : default 1
#define BIT_ENABLE_RXD_LINK_ZIGBEE PD6 //out : default 0
#define BIT_LINK_PLUGIN PD7 //in, no pull up

void TxD81(byte bTxData);
void TxD80(byte bTxData);
void TxDString(byte *bData);
void TxDHex(byte bSendData);
void TxD32Dec(long lLong);
byte Rx8D1(void);
void MilliSec(word wDelayTime);
void PortInitialize(void);
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt);
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength);

```

```

byte RxPacket(byte bRxLength);
void PrintBuffer(byte *pbPrintBuffer, byte bLength);

// --- Global Variable Number ---
volatile byte gbpRxInterruptBuffer[256];
byte gbpParameter[128];
byte gbpRxBufferReadPointer;
byte gbpRxBuffer[128];
byte gbpTxBuffer[128];
volatile byte gbpRxBufferWritePointer;

int main(void)
{
    byte bCount, bID, bTxPacketLength, bRxPacketLength;

    PortInitialize(); //Port In/Out Direction Definition
    RS485_RXD; //Set RS485 Direction to Input State.

    SerialInitialize(SERIAL_PORT0, DEFAULT_BAUD_RATE,
                    RX_INTERRUPT); //RS485 Initializing(RxInterrupt)
    SerialInitialize(SERIAL_PORT1, DEFAULT_BAUD_RATE, 0); //RS232
        Initializing(None Interrupt)

    gbpRxBufferReadPointer = gbpRxBufferWritePointer = 0; //RS485
        RxBuffer Clearing.

    sei(); //Enable Interrupt -- Compiler Function
    TxString("¥r¥n [The Example of Dynamixel Evaluation with
        ATmega128_GCC-AVR]");

//Dynamixel Communication Function Execution Step.
// Step 1. Parameter Setting (gbpParameter[]). In case of no parameter
// instruction(Ex. INST_PING), this step is not
// needed.
// Step 2. TxPacket(ID, INSTRUCTION, LengthOfParameter); --Total
// TxPacket Length is returned
// Step 3. RxPacket(ExpectedReturnPacketLength); -- Real RxPacket
// Length is returned
// Step 4 PrintBuffer(BufferStartPointer, LengthForPrinting);

    bID = 1;
    TxString("¥r¥n Example 1. Scanning Dynamixels(0~9). -- Any Key to
        Continue."); Rx8();
    for(bCount = 0; bCount < 0x0A; bCount++)
    {
        bTxPacketLength = TxPacket(bCount, INST_PING, 0);
        bRxPacketLength = RxPacket(255);
        TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
        TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);
        if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE)
        {
            TxString(" Found!! ID:"); Tx8Hex(bCount);
            bID = bCount;
        }
    }

    TxString("¥r¥n Example 2. Read Firmware Version. -- Any Key to
        Continue."); Rx8();
    gbpParameter[0] = P_VERSION; //Address of Firmware Version
    gbpParameter[1] = 1; //Read Length
    bTxPacketLength = TxPacket(bID, INST_READ, 2);
    bRxPacketLength =
        RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter
            [1]);
    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);
    if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
    {
        TxString("¥r¥n Return Error : "); Tx8Hex(gbpRxBuffer[4]);
        TxString("¥r¥n Firmware Version : "); Tx8Hex(gbpRxBuffer[5]);
    }

    TxString("¥r¥n Example 3. LED ON -- Any Key to Continue.");
    Rx8();
    gbpParameter[0] = P_LED; //Address of LED
    gbpParameter[1] = 1; //Writing Data
    bTxPacketLength = TxPacket(bID, INST_WRITE, 2);

```

```

    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

    TxString("¥r¥n Example 4. LED OFF -- Any Key to Continue."); Rx8();
    gbpParameter[0] = P_LED; //Address of LED
    gbpParameter[1] = 0; //Writing Data
    bTxPacketLength = TxPacket(bID, INST_WRITE, 2);
    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

    TxString("¥r¥n Example 5. Read Control Table. -- Any Key to
        Continue."); Rx8();
    gbpParameter[0] = 0; //Reading Address
    gbpParameter[1] = 49; //Read Length
    bTxPacketLength = TxPacket(bID, INST_READ, 2);
    bRxPacketLength =
        RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter
            [1]);

    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);
    if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
    {
        TxString("¥r¥n");
        for(bCount = 0; bCount < 49; bCount++)
        {
            Tx8(' '); Tx8Hex(bCount); TxString(":");
            Tx8Hex(gbpRxBuffer[bCount+5]); Tx8(' ');
        }
    }

    TxString("¥r¥n Example 6. Go 0x200 with Speed 0x100 -- Any Key to
        Continue."); Rx8();
    gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
    gbpParameter[1] = 0x00; //Writing Data P_GOAL_POSITION_L
    gbpParameter[2] = 0x02; //Writing Data P_GOAL_POSITION_H
    gbpParameter[3] = 0x00; //Writing Data P_GOAL_SPEED_L
    gbpParameter[4] = 0x01; //Writing Data P_GOAL_SPEED_H
    bTxPacketLength = TxPacket(bID, INST_WRITE, 5);
    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

    TxString("¥r¥n Example 7. Go 0x00 with Speed 0x40 -- Any Key to
        Continue."); Rx8();
    gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
    gbpParameter[1] = 0x00; //Writing Data P_GOAL_POSITION_L
    gbpParameter[2] = 0x00; //Writing Data P_GOAL_POSITION_H
    gbpParameter[3] = 0x40; //Writing Data P_GOAL_SPEED_L
    gbpParameter[4] = 0x00; //Writing Data P_GOAL_SPEED_H
    bTxPacketLength = TxPacket(bID, INST_WRITE, 5);
    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

    TxString("¥r¥n Example 8. Go 0x3ff with Speed 0x3ff -- Any Key to
        Continue."); Rx8();
    gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
    gbpParameter[1] = 0xff; //Writing Data P_GOAL_POSITION_L
    gbpParameter[2] = 0x03; //Writing Data P_GOAL_POSITION_H
    gbpParameter[3] = 0xff; //Writing Data P_GOAL_SPEED_L
    gbpParameter[4] = 0x03; //Writing Data P_GOAL_SPEED_H
    bTxPacketLength = TxPacket(bID, INST_WRITE, 5);
    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

    TxString("¥r¥n Example 9. Torque Off -- Any Key to Continue.");
    Rx8();
    gbpParameter[0] = P_TORQUE_ENABLE; //Address of LED
    gbpParameter[1] = 0; //Writing Data
    bTxPacketLength = TxPacket(bID, INST_WRITE, 2);
    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
    TxString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);

```

```

TxDString("Yr\n RxD:");
PrintBuffer(gbpRxBuffer, bRxPacketLength);

TxDString("Yr\n End. Push reset button for repeat");
while(1);
}

void PortInitialize(void)
{
    DDRA = DDRB = DDRC = DDRD = DDRE = DDRF = 0; //Set all port to
                                                input direction first.
    PORTB = PORTC = PORTD = PORTE = PORTF = PORTG = 0x00; //PortData
                                                initialize to 0
    cbi(SFIOR, 2); //All Port Pull Up ready
    DDRE |= (BIT_RS485_DIRECTION0|BIT_RS485_DIRECTION1); //set output
                                                the bit RS485direction

    DDRD |=
        (BIT_ZIGBEE_RESET|BIT_ENABLE_RXD_LINK_PC|BIT_ENA
        BLE_RXD_LINK_ZIGBEE);

    PORTD &= ~_BV(BIT_LINK_PLUGIN); // no pull up
    PORTD |= _BV(BIT_ZIGBEE_RESET);
    PORTD |= _BV(BIT_ENABLE_RXD_LINK_PC);
    PORTD |= _BV(BIT_ENABLE_RXD_LINK_ZIGBEE);
}

/*
TxPacket() send data to RS485.
TxPacket() needs 3 parameter; ID of Dynamixel, Instruction byte,
Length of parameters.
TxPacket() return length of Return packet from Dynamixel.
*/
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength)
{
    byte bCount, bCheckSum, bPacketLength;

    gbpTxBuffer[0] = 0xff;
    gbpTxBuffer[1] = 0xff;
    gbpTxBuffer[2] = bID;
    gbpTxBuffer[3] = bParameterLength+2; //Length(Parameter, Instruction, Checksum)
    gbpTxBuffer[4] = bInstruction;
    for(bCount = 0; bCount < bParameterLength; bCount++)
    {
        gbpTxBuffer[bCount+5] = gbpParameter[bCount];
    }
    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;
    for(bCount = 2; bCount < bPacketLength-1; bCount++) //except
                                                0xff, checksum
    {
        bCheckSum += gbpTxBuffer[bCount];
    }
    gbpTxBuffer[bCount] = ~bCheckSum; //Writing Checksum with Bit
                                                Inversion

    RS485_TXD:
    for(bCount = 0; bCount < bPacketLength; bCount++)
    {
        sbi(UCSROA, 6); //SET_TXDO_FINISH;
        TxD80(gbpTxBuffer[bCount]);
    }
    while(!CHECK_TXDO_FINISH); //Wait until TXD Shift register empty
    RS485_RXD;
    return(bPacketLength);
}

/*
RxPacket() read data from buffer.
RxPacket() need a Parameter: Total length of Return Packet.
RxPacket() return Length of Return Packet.
*/
byte RxPacket(byte bRxPacketLength)
{
#define RX_TIMEOUT_COUNT2 3000L
#define RX_TIMEOUT_COUNT1 (RX_TIMEOUT_COUNT2*10L)
}

```

```

unsigned long ulCounter;
byte bCount, bLength, bChecksum;
byte bTimeout;

bTimeout = 0;
for(bCount = 0; bCount < bRxPacketLength; bCount++)
{
    ulCounter = 0;
    while(gbRxBufferReadPointer == gbRxBufferWritePointer)
    {
        if(ulCounter++ > RX_TIMEOUT_COUNT1)
        {
            bTimeout = 1;
            break;
        }
    }
    if(bTimeout) break;
    gbpRxBuffer[bCount] =
        gbpRxInterruptBuffer[gbRxBufferReadPointer++];
}

bLength = bCount;
bChecksum = 0;

if(gbpTxBuffer[2] != BROADCASTING_ID)
{
    if(bTimeout && bRxPacketLength != 255)
    {
        TxDString("Yr\n [Error:RxD Timeout]");
        CLEAR_BUFFER;
    }
}

if(bLength > 3) //checking is available.
{
    if(gbpRxBuffer[0] != 0xff || gbpRxBuffer[1] != 0xff )
    {
        TxDString("Yr\n [Error:Wrong Header]");
        CLEAR_BUFFER;
        return 0;
    }
    if(gbpRxBuffer[2] != gbpTxBuffer[2] )
    {
        TxDString("Yr\n [Error:TxID != RxID]");
        CLEAR_BUFFER;
        return 0;
    }
    if(gbpRxBuffer[3] != bLength-4)
    {
        TxDString("Yr\n [Error:Wrong Length]");
        CLEAR_BUFFER;
        return 0;
    }
    for(bCount = 2; bCount < bLength; bCount++) bChecksum +=
        gbpRxBuffer[bCount];
    if(bChecksum != 0xff)
    {
        TxDString("Yr\n [Error:Wrong CheckSum]");
        CLEAR_BUFFER;
        return 0;
    }
}
return bLength;
}

/*
PrintBuffer() print data in Hex code.
PrintBuffer() needs two parameter; name of Pointer(gbpTxBuffer,
                                                gbpRxBuffer)
*/
void PrintBuffer(byte *bpPrintBuffer, byte bLength)
{
    byte bCount;
    for(bCount = 0; bCount < bLength; bCount++)
    {
        TxD8Hex(bpPrintBuffer[bCount]);
        TxD8(' ');
    }
}

```

```

        }
        TxDString(("(LEN:") ;TxD8Hex(bLength) ;TxD8(')');
    }

/*
Print value of Baud Rate.
*/
void PrintBaudrate(void)
{
    TxDString("¥r¥n
RS232:");TxD32Dec((1600000L/8L)/((long)UBRR1L+1
L)); TxDString(" BPS.");
    TxDString(" RS485:");TxD32Dec((1600000L/8L)/((long)UBRR0L+1L));
    TxDString(" BPS");
}

/*Hardware Dependent Item*/
#define TXD1_READY           bit_is_set(UCSR1A, 5)
                           // (UCSR1A_Bit5)
#define TXD1_DATA            (UDR1)
#define RXD1_READY           bit_is_set(UCSR1A, 7)
#define RXD1_DATA            (UDR1)

#define TXD0_READY           bit_is_set(UCSROA, 5)
#define TXD0_DATA            (UDRO)
#define RXD0_READY           bit_is_set(UCSROA, 7)
#define RXD0_DATA            (UDRO)

/*
SerialInitialize() set Serial Port to initial state.
Vide Mega128 Data sheet about Setting bit of register.
SerialInitialize() needs port, Baud rate, Interrupt value.

*/
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt)
{
    if(bPort == SERIAL_PORT0)
    {
        UBRROH = 0; UBRROL = bBaudrate;
        UCSROA = 0x02; UCSROB = 0x18;
        if(bInterrupt&RX_INTERRUPT) sbi(UCSROB, 7); // RxD interrupt enable
        UCSROC = 0x06; UDRO = 0xFF;
        sbi(UCSROA, 6);//SET_TXD0_FINISH; // Note. set 1, then 0 is read
    }
    else if(bPort == SERIAL_PORT1)
    {
        UBRR1H = 0; UBRR1L = bBaudrate;
        UCSR1A = 0x02; UCSR1B = 0x18;
        if(bInterrupt&RX_INTERRUPT) sbi(UCSR1B, 7); // RxD interrupt enable
        UCSR1C = 0x06; UDR1 = 0xFF;
        sbi(UCSR1A, 6);//SET_RXD1_FINISH; // Note. set 1, then 0 is read
    }
}

/*
TxD8Hex() print data seperately.
ex> 0x1a -> '1' 'a'.
*/
void TxD8Hex(byte bSentData)
{
    byte bTmp;

    bTmp =((byte)(bSentData>>4)&0x0f) + (byte)'0';
    if(bTmp > '9') bTmp += 7;
    TxD8(bTmp);
    bTmp = (byte)(bSentData & 0x0f) + (byte)'0';
    if(bTmp > '9') bTmp += 7;
    TxD8(bTmp);
}

/*
TxD80() send data to USART 0.
*/
void TxD80(byte bTxData)
{
    while(!TXD0_READY);

    TXD0_DATA = bTxData;
}

/*
TXD81() send data to USART 1.
*/
void TxD81(byte bTxData)
{
    while(!TXD1_READY);
    TXD1_DATA = bTxData;
}

/*
TXD32Dec() change data to decimal number system
*/
void TxD32Dec(long lLong)
{
    byte bCount, bPrinted;
    long lTmp, lDigit;
    bPrinted = 0;
    if(lLong < 0)
    {
        lLong = -lLong;
        TxD8('-');
    }
    lDigit = 100000000L;
    for(bCount = 0; bCount < 9; bCount++)
    {
        lTmp = (byte)(lLong/lDigit);
        if(lTmp)
        {
            TxD8(((byte)lTmp)+'0');
            bPrinted = 1;
        }
        else if(bPrinted) TxD8(((byte)lTmp)+'0');
        lLong -= ((long)lTmp)*lDigit;
        lDigit = lDigit/10;
    }
    lTmp = (byte)(lLong/lDigit);
    /*if(lTmp)*/ TxD8(((byte)lTmp)+'0');

}

/*
TxDString() prints data in ASCII code.
*/
void TxDString(byte *bData)
{
    while(*bData)
    {
        TxD8(*bData++);
    }
}

/*
RxD81() read data from UART1.
RxD81() return Read data.
*/
byte RxD81(void)
{
    while(!RXD1_READY);
    return(RXD1_DATA);
}

/*
SIGNAL() UART0 Rx Interrupt - write data to buffer
*/
SIGNAL (SIG_UART0_RECV)
{
    gpbRxBuffer[gbRxBufferWritePointer++] = RXD0_DATA;
}

```

**Connector**

Company Name : Molex

Pin Number: 4

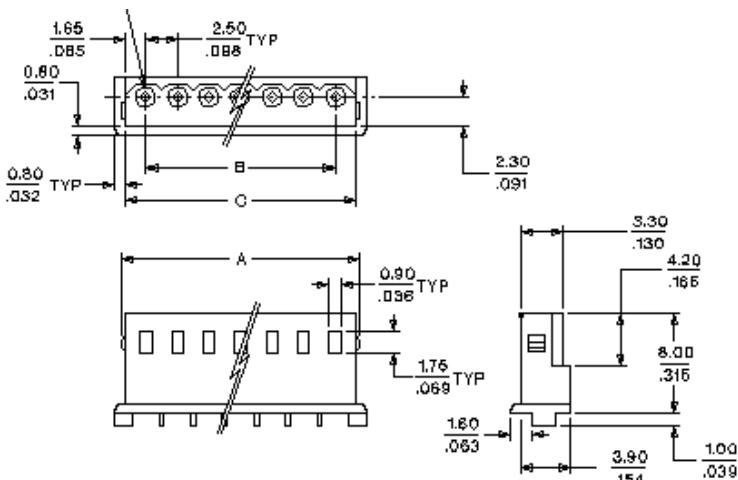
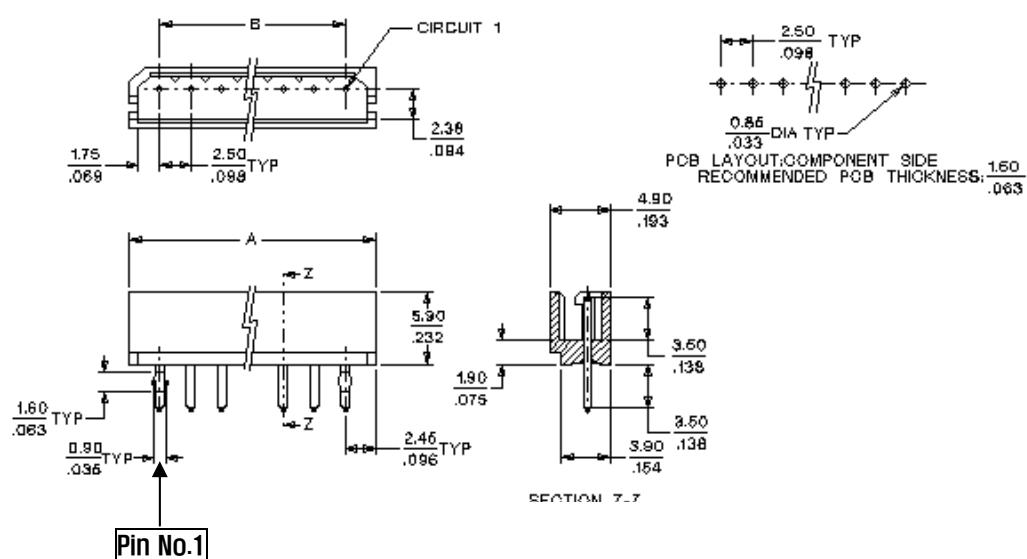
Model Number

	Molex Part Number	Old Part Number
Male	22-03-5045	5267-03
Female	50-37-5043	5264-03

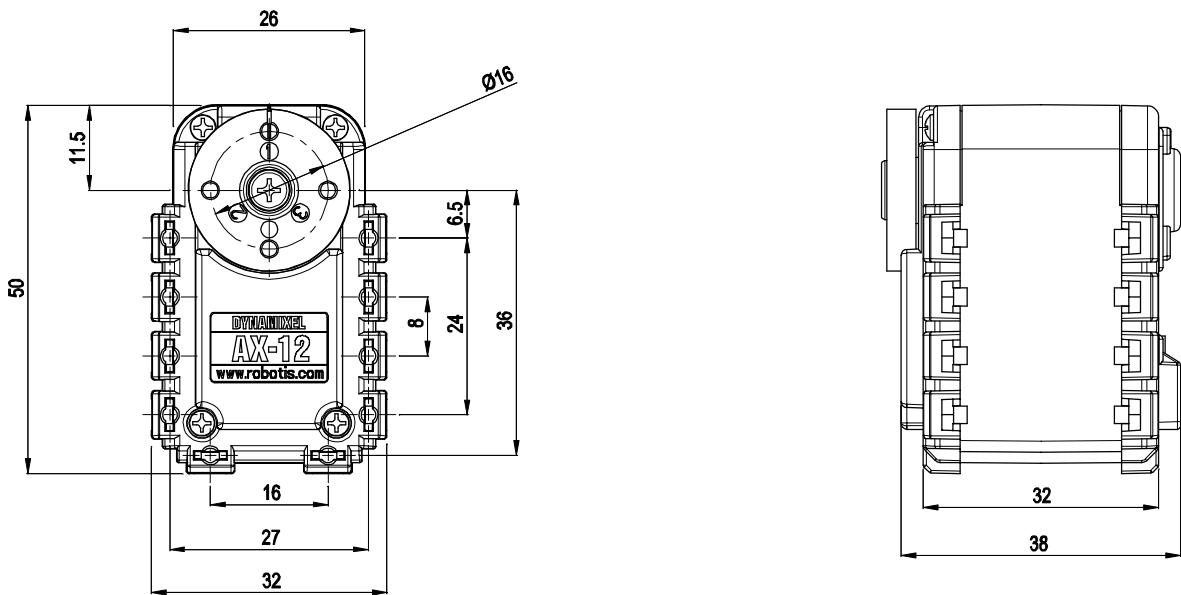
Temperature range : -40°C to +105°C

Contact Insertion Force-max : 14.7N (3.30 lb)

Contact Retention Force-min : 14.7N (3.30 lb)

[www.molex.com](http://www.molex.com) or [www.molex.co.jp](http://www.molex.co.jp) for more detail information**Female Connector****Male Connector**

## Dimension



## CM-5

AX-12 전용 콘트롤 박스. 30개의 AX-12 제어 가능

6개의 Push button(5개 : 선택용, 1개 reset용)

Wireless 장치를 추가 장착할 수 있다.

충전지 수납공간(AA X 8)이 있으며 충전기능이 있다.(외부 SMPS 연결시)

