1. Dynamixel AX－12

1.1 AX－12 概览和特性

Dynamixel 系列机器人舵机是一种智能化、模块化动力装置，由齿轮减速箱、一个精确的直流电机以及具备通讯功能的控制芯片打包而成。能产生大扭矩，材料坚固，保证承受极大外力必需的强度和韧性。工作时可反馈内部状况，例如内部温度或输入电压。

- **精确控制**
  位置和速度可设为 1024 等分控制。
- **柔性驱动**
  控制舵机位移过程中，柔性转动的角度可调。
- **反馈**
  反馈值包括角位移、角速度和负载扭矩。
- **报警系统**
  当舵机内部参数（例如内部电压、扭矩和电压等）偏离工作值过大时，系统将警告用户，也可以自动处理该问题（例如撤销扭矩）
- **通讯**
  支持通讯速度最高达 1Mbps。
- **分布式控制**
  位置、速度、伺服性和扭矩可以通过一个指令包设定，这样主控制器即使资源很少，也可以控制很多舵机。
- **工程塑料**
  壳体由高质量工程塑料制成，可承受大扭矩负载。
- **LED 指示灯**
  显示用户是否操作错误。

1.2 主要参数

	AX-12	
重量（g）	55	
减速比	1/254	
输入电压	7V	10V
最大扭矩（kgf • cm）	12	16.5
转速（秒/60°）	0.269	0.196

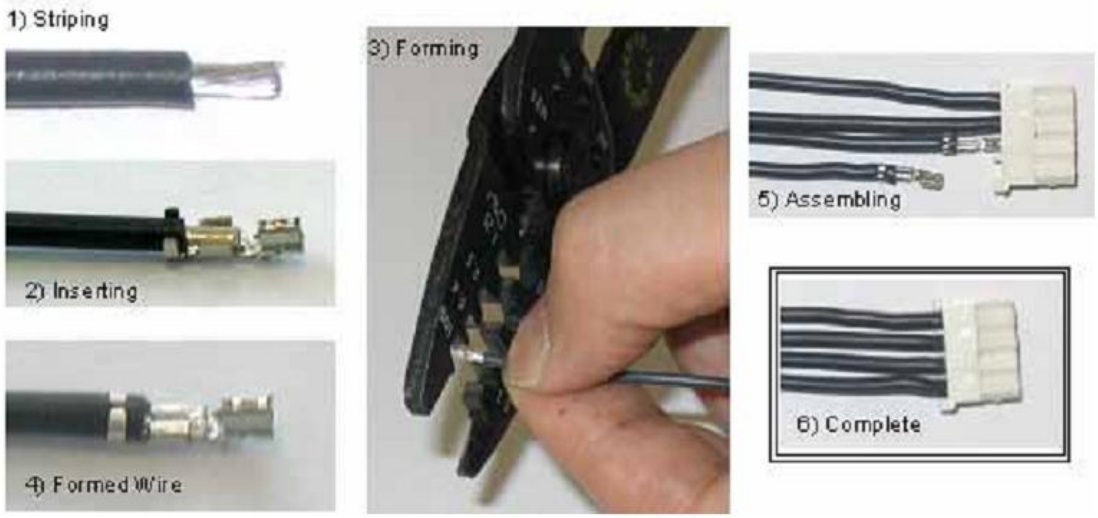
- ◆ 通讯　　半双工异步串口通讯（8bit、1 中断位，无奇偶校验）
- ◆ 最小角度　　0.35°
- ◆ 波特率　　7343 bps ~ 1M bps
- ◆ ID 扩展　　0 ~ 253
- ◆ 工作电压　　7VDC ~ 10VDC（推荐 9.6V）
- ◆ 工作温度　　-5℃ ~ +85℃
- ◆ 最大电流　　900mA
- ◆ 指令包　　数字信号
- ◆ 位移角度　　0° ~ 300°，无限旋转
- ◆ 物理连接　　TTL 多通道（带菊花链传输线）
- ◆ 反馈　　位置、温度、负载和输入电压等

◆ 外壳材料 工程塑料

2. Dynamixel 舵机操作

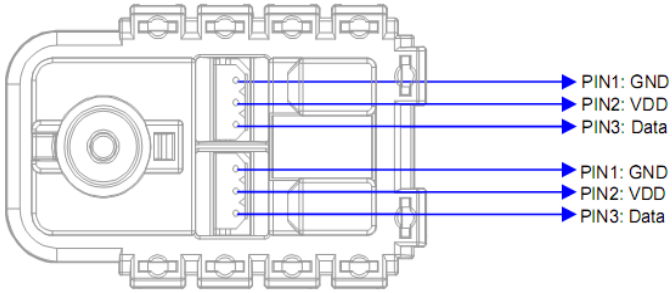
2.1 传输线组装

按下图所示组装传输线。使用合适的夹线钳。如果手边没有夹线钳，可焊接终端与通讯线，确保舵机工作时传输线不松开。

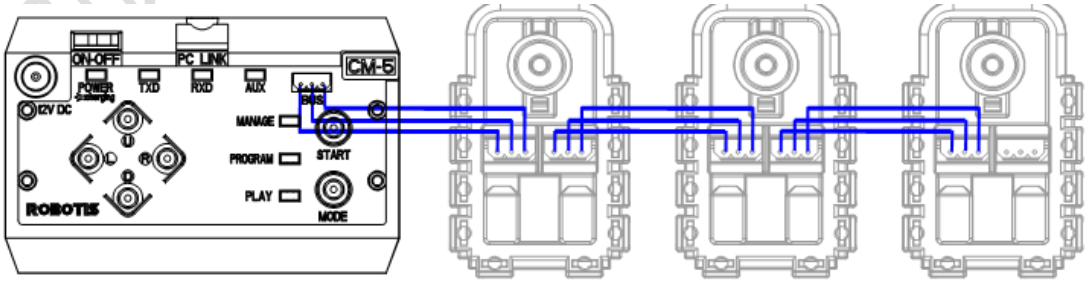


2.2 Dynamixel 通讯线

- **针脚排布** 传输线针脚排布如图。舵机有两个插座，只要连接其中一个，舵机就可以工作。



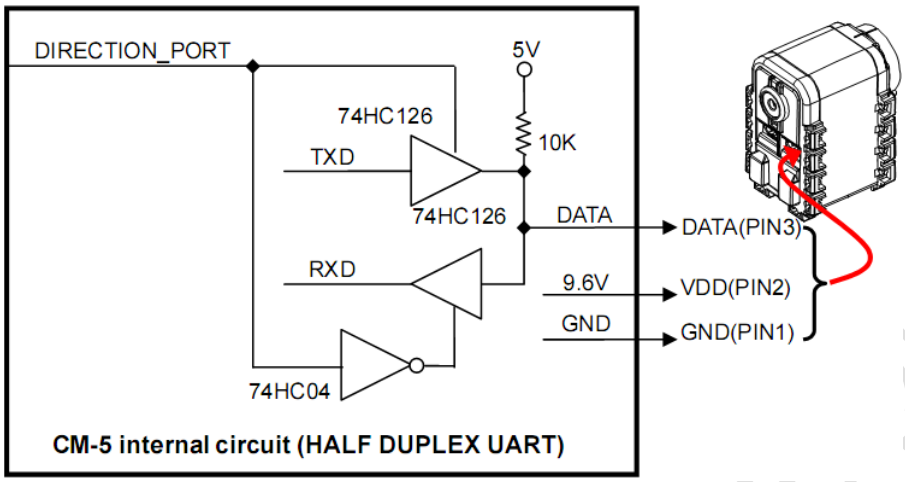
- **连接** 如图，舵机连接是针脚对针脚的，这样，只需一条总线通道就可以连接多个舵机。



控制器“CM-5”

- **主控制器** 为了控制舵机，主控制器必须支持 TTL 半双工 UART 协议。这里推荐使用 CM-5 控制器，也可以使用其它合适的控制器。

- **计算机连接** 计算机通过 RS232 与控制器连接，再通过控制器控制舵机。
- **与 UART 连接** 要控制 Dynamixel 舵机，主控制器需要转换 UART 信号为半双工信号。下图为推荐电路图。



舵机供电来自主控制器针脚 1 和针脚 2。（上图只用于说明半双工 UART 的使用。CM-5 控制器已内嵌上述电路，因此 Dynamixel 舵机可直接与之连接。）

TTL 发射和接收数据信号的方向取决于下面的“方向端口”信号水平：

方向端口高电平：发送数据信号

方向端口低电平：接收数据信号

- **半双工 UART** 半双工 UART 使多个舵机通过多通道方式连接至单个节点成为可能。

注意：连接 Dynamixel 舵机时确保针脚排布正确。接通电源后检查电流分配是否合理。单个舵机待机状态下的电流不高于 50mA。

- **连接状态确认** 当 Dynamixel 舵机接电后，LED 闪两次，表示确认连接。
- **检测** 如果上述操作未成功，检查针脚排布和电源的电压/电流限制。

3. 通讯协议

3.1 通讯方式概述

- **信号包** 主控制器给 Dynamixel 舵机发送“指令包”，舵机反馈“状态包”。
- **通讯** 主控制器向 ID 为 N 的舵机发送指令包，只有该舵机会反馈状态包并执行指令。
- **ID 唯一** 如果舵机 ID 相同，同时发送的多个指令包会相互冲突，造成通讯问题。因此，在同一网络中，不要设置相同的 ID。

3.2 指令包

指令包结构如下：

OXFF	0XFF	ID	LENGTH	INSTRUCTION	PARAMETER1	…	PARAMETER N	CHECK SUM

0XFF 0XFF	2 个 0xFF 表示开始传送指令包。
ID	每个舵机对应唯一 ID。可连接 254 个 ID，范围从 0X00 至 0XFD。 **ID** 0XFE 是广播 ID，表示所有连接舵机。向这个 ID 发送的信号包将传递至同一网络的所有 ID。因此，向广播 ID 发送的指令包不会收到任何状态包。
LENGTH	指令包的长度，该值表示为“参数个数（N）+2”。
INSTRUCTION	要求舵机执行的指令。
PARAMETER0…N	除指令本身之外，如果还有附加信息需要传送，用该值表示。
CHECK SUM	“校验码”的计算方法如下： CheckSum = ~（ID + Length + Instruction + Parameter1 + … + Parameter N） 如果计算结果大于 255，则后面字节的值设为校验码的值。 “~”表示非逻辑运算。

3.3 状态包（反馈包）

状态包是舵机接收主控制器的指令包后传回的响应包。状态包的结构如下：

OXFF	0XFF	ID	LENGTH	ERROR	PARAMETER1	PARAMETER2	…	PARAMETER N	CHECK SUM

0XFF 0XFF	2 个 0xFF 表示开始传送反馈包。
ID	每个舵机对应唯一 ID。初始值设为 1。
LENGTH	指令包的长度，该值表示为“参数个数（N）+2”。

ERROR 该字节每个字位定义如下：

Bit	名称	说明
Bit 7	0	
Bit 6	指令错误	如果控制器发送的是未定义指令或无 Reg_Write 指令的行为指令，该字节设为 1
Bit 5	过载错误	如果规定的最大扭矩无法带动负载，该字节设为 1
Bit 4	校验码错误	如果指令包的校验码不正确，该字节设为 1
Bit 3	范围错误	如果指令超出规定范围，该字节设为 1
Bit 2	过热错误	如果舵机内部温度高于控制表规定的工作温度，该字节设为 1
Bit 1	角度限制错误	舵机目标位置限于 CW 角和 CCW 角之间，如果指令包给定的目标位置超出这个范围，该字节设为 1
Bit 0	输入电压错误	如果舵机电压超出控制表规定的工作电压范围，该字节设为 1

PARAMETER0...N 有附加信息需要传送，用该值表示。

CHECK SUM “校验码”的计算方法同上。

3.4 控制表

- **EEPROM 区**

地址	条目	途径	初始值
0（0X00）	模型序号（低字节）	读	12（0x0C）
1（0X01）	模型序号（高字节）	读	0（0x00）
2（0X02）	固件版本	读	?
3（0X03）	ID	读、写	1（0x01）
4（0X04）	波特率	读、写	1（0x01）
5（0X05）	反馈延时	读、写	250(0xFA)
6（0X06）	最小角度（低）	读、写	0（0x00）
7（0X07）	最小角度（高）	读、写	0（0x00）
8（0X08）	最大角度（低）	读、写	255（oxFF）
9（0X09）	最大角度（高）	读、写	3（0x03）
10（0X0A）	（保留）		0（0x00）
11（0X0B）	温度上限	读、写	85（0x55）
12（0X0C）	电压下限	读、写	60（0x3C）
13（0X0D）	电压上限	读、写	190（0xBE）
14（0X0E）	最大扭矩（低）	读、写	255（0xFF）
15（0X0F）	最大扭矩（高）	读、写	3（0x03）
16（0X10）	状态反馈程度	读、写	2（0x02）
17（0X11）	LED 警报	读、写	4（0x04）
18（0X12）	撤消扭矩警报	读、写	4（0x04）
19（0X13）	（保留）		0（0x00）
20（0X14）	向下校正（低）	读	?
21（0X15）	向下校正（高）	读	?
22（0X16）	向上校正（低）	读	?

23（0X17）	向上校正（高）	读	?

- RAM 区

地址	条目	途径	初始值
24（0X18）	激活扭矩	读、写	0（0x00）
25（0X19）	LED	读、写	0（0x00）
26（0X1A）	顺时针柔性边距	读、写	0（0x00）
27（0X1B）	逆时针柔性边距	读、写	0（0x00）
28（0X1C）	顺时针柔性斜率	读、写	32（0x20）
29（0X1D）	逆时针伺服斜率	读、写	32（0x20）
30（0X1E）	目标位置（低）	读、写	地址 36 的值
31（0X1F）	目标位置（高）	读、写	地址 37 的值
32（0X20）	运动速度（低）	读、写	0
33（0X21）	运动速度（高）	读、写	0
34（0X22）	扭矩限制（低）	读、写	地址 14 的值
35（0X23）	扭矩限制（高）	读、写	地址 15 的值
36（0X24）	当前位置（低）	读	?
37（0X25）	当前位置（高）	读	?
38（0X26）	当前速度（低）	读	?
39（0X27）	当前速度（高）	读	?
40（0X28）	当前负载（低）	读	?
41（0X29）	当前负载（高）	读	?
42（0X2A）	当前电压	读	?
43（0X2B）	当前温度	读	?
44（0X2C）	寄存器指令	读、写	0（0x00）
45（0X2D）	（保留）		0（0x00）
46（0X2E）	运动中	读	0（0x00）
47（0X2F）	锁定	读、写	0（0x00）
48（0X30）	撞击（低）	读、写	32（0x20）
49（0X31）	撞击（高）	读、写	0（0x00）

- 控制表　　控制表包含舵机状态和操作信息。操作 Dynamixel 舵机是通过向控制表写数值实现的，检测状态通过从控制表读数值实现。
- 初始值　　EEPROM 的初始值指的是产品出厂值，RAM 的初始值指的是接电后的原始值。

下面说明每个地址存储的数据的意义。

- 地址 0x00、0x01　模型序号，对 AX-12 来说，这个值是 0x000C（12）。
- 地址 0x02　　固件程序版本
- 地址 0x03　　分配给每个舵机的唯一 ID 序号。同一网络中的舵机序号不能相同。
- 地址 0x04　　波特率，确定通讯速度。计算公式如下：

速度（BPS）= 2 000 000 / (地址 4 + 1)

地址 4 初始值	Hex	设置 BPS	实际 BPS	误差
1	0x01	1000 000	1000 000	0.000%

3	0x03	500 000	500 000	0.000%
4	0x04	400 000	400 000	0.000%
7	0x07	250 000	250 000	0.000%
9	0x09	200 000	200 000	0.000%
16	0x10	117 647.1	115200.0	-2.124%
34	0x22	57142.9	57600.0	0.794%
103	0x67	19230.8	19200.0	-0.160%
207	0xCF	9615.4	9600.0	-0.160%

注意：UART 通讯范围内的波特率最大误差率为 3%。

- 地址 0x05　　表示指令包发送后舵机传回状态包所需时间。延时的计算是：2(μs)× 地址 5 的值。
- 地址 0x06、07、08、09　设置 Dynamixel 舵机操作角度范围。目标位置的范围是：顺时针角度限制《=目标位置《=逆时针角度范围。
- 地址 0x0B　　显示舵机操作温度的上限。如果舵机内部温度超过这个值，状态包第 2 位——过热错误位——赋值 1，地址 17 和 18 将设置警报。
- 地址 0x0C、0x0D　显示最低（最高）电压。如果当前电压（地址 42）超出指定范围，指令包 0 位——电压范围错误位——赋值 1，地址 17 和 18 将设置警报。该地址的值是实际电压值的 10 倍。例如，如果地址 12 的值是 80，那么电压下限设为 8V。
- 地址 0x0E、0F、22、23　显示舵机的最大扭矩输出。当该值设为 0 时，舵机进入自由运行模式。最大扭矩的设定存在于两个区内：EEPROM（地址 0x0E、0x0F）和 RAM 区（0x22、0x23）。接电后，EEPROM 中的最大扭矩值复制到 RAM 区中。舵机的扭矩受到 RAM 区（0x22、0x23）的值的限制。
- 地址 0x10　　确定舵机接收到指令包后是否返回状态包。

地址 0x10	返回状态包
0	不响应任何指令包
1	只响应 Read_Data 指令
2	响应所有指令

注意：如果指令使用了“广播 ID”（0xFE），那么无论地址 0x10 是什么值，舵机都不会返回状态包。

- 地址 0x11　　如果响应的位设为 1，出现错误时，LED 发光。

Bit	功能
7	0
6	如设为 1，出现指令错误时，LED 发光
5	如设为 1，出现过载错误时，LED 发光
4	如设为 1，出现校验码错误时，LED 发光
3	如设为 1，出现运动边界错误时，LED 发光
2	如设为 1，出现过热错误时，LED 发光
1	如设为 1，出现角度限制错误时，LED 发光
0	如设为 1，出现输入电压错误时，LED 发光

该功能是按照各位上的值的“或”逻辑运算实现的。例如，该地址的值为 0x05，也就是 00000101，则出现输入电压错误**或者**过热错误时，LED 发光。当舵机从错误状态返回正常状态时，LED 发光 2 秒后

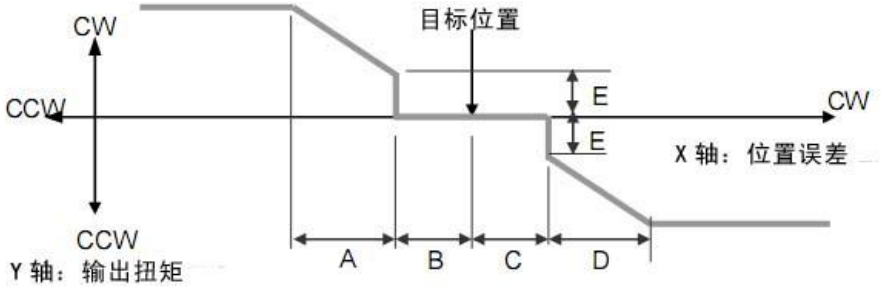
熄灭。

- 地址 0x12　　如果相应位设为 1，出现错误时，舵机撤消扭矩。

Bit	功能
7	0
6	如设为 1，出现指令错误时，撤消扭矩
5	如设为 1，出现过载错误时，撤消扭矩
4	如设为 1，出现校验码错误时，撤消扭矩
3	如设为 1，出现运动边界错误时，撤消扭矩
2	如设为 1，出现过热错误时，撤消扭矩
1	如设为 1，出现角度限制错误时，撤消扭矩
0	如设为 1，出现输入电压错误时，撤消扭矩

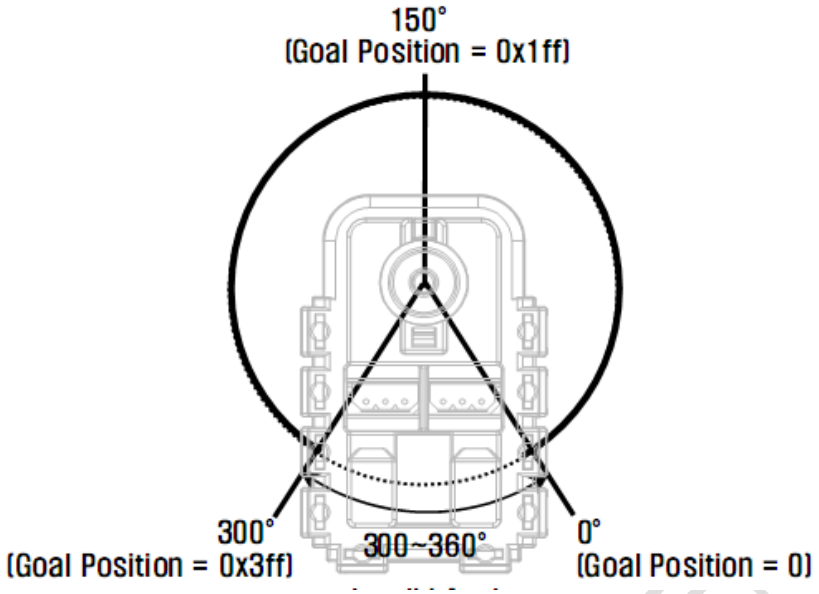
该功能是按照各位上的值的“或”逻辑运算实现的。不过，与 LED 警报不同的是，返回正常状态后，仍保持无扭矩状态。要恢复扭矩，需将“激活扭矩”（地址 0x18）设为 1。

- 地址 0x14~0x17　　补充舵机的电位计之间的差值。用户无权修改该值。下面（地址 0x18）存储在 RAM 区。
- 地址 0x18　　刚开始打开电源，Dynamixel 舵机首先进入“无扭矩运行”状态。设置地址 0x18 的值为 1，激活扭矩。
- 地址 0x19　　该地址设置为 1，LED 打开，设置为 0，LED 关闭。
- 地址 0x1A~ox1D　　该地址表示柔性边距和斜率。舵机的柔性通过柔性边距和斜率表示，这个特性可用于减小输出轴受到的冲击。下图显示了如何用位置误差和输出扭矩表示每一段柔性值。



A：CCW 柔性斜线（地址 0x1D）
B：CCW 柔性边距（地址 0x1B）
C：CW 柔性边距边距（地址 0x1A）
D：CW 柔性斜线（地址 0x1C）
E：撞击（地址 0x30、0x31）

- 地址 0x1E、0x1F　　设置舵机运动必需的角度位置。将该值设为 0x3ff，可以使舵机运动到 300°的目标位置，如图。其中 300°~360°为无效角度。



- 地址 0x20，0x21　　设置舵机运动到目标位置的角速度。该值最大值可为 0x3ff，表示角速度 114 RPM。最小角速度设为 1。如设为 0，舵机将以电压能提供的最大速度运动，也就是说，没有速度控制。
- 地址 0x24、0x25　　舵机当前角度位置。
- 地址 0x26、0x27　　舵机当前角速度。
- 地址 0x28、0x29　　舵机当前负载。

Bit	15-11	10	9	8	7	6	5	4	3	2	1	0
值	0	负载方向	负载值									

负载方向=0：逆时针负载；负载方向=1：顺时针方向

- 地址 0x2A　　舵机当前电压。该值是实际电压的 10 倍。
- 地址 0x2B　　舵机当前温度，用摄氏度表示。
- 地址 0x2C　　当指令由 REG_WRITE 命令分配时，设为 1。当 ACTION 命令完成被分配的指令时，设为 0。
- 地址 0x2E　　舵机靠自身电力运动时，设为 1。
- 地址 0x2F　　如果设为 1，只有地址 0x18 至 0x23 可以写入信息，其它区域不可以。一旦锁定，只有关闭电源，才会解锁。
- 地址 0x30、0x31　　表示舵机运转过程中使用的最小电流。初始值设为 0x20，最大值为 0x3ff。
- 无限旋转　　如果“顺时针角度限制”和“逆时针角度限制”的值设为 0，那么设置好速度后，舵机执行“无限旋转模式”。这个特性可用于连续旋转的轮子。旋转速度设置如下：

Bit	15-11	10	9	8	7	6	5	4	3	2	1	0
值	0	旋转方向	速度值									

旋转方向=0：逆时针方向旋转；旋转方向=1：顺时针方向旋转

- 各地址取值范围　　下表列出了每个地址的数据范围。16 位数据寄存器包含低字节和高字节。同一指令包内必须同时给两个字节赋值。

可写地址	写入项目	长度（字节）	最小值	最大值
3（0X03）	ID	1	0	253（0xfd）
4（0X04）	波特率	1	0	254（0xfe）
5（0X05）	反馈延时	1	0	254（0xfe）
6（0X06）	顺时针角度限制	2	0	1023（0x3ff）
8（0X08）	逆时针角度限制	2	0	1023（0x3ff）
11（0X0B）	温度上限	1	0	150（0x96）
12（0X0C）	电压下限	1	50（0x32）	250（0xfa）
13（0X0D）	电压上限	1	50（0x32）	250（0xfa）
14（0X0E）	最大扭矩	2	0	1023（0x3ff）
16（0X10）	状态反馈程度	1	0	2
17（0X11）	LED 警报	1	0	127（0x7f）
18（0X12）	撤消扭矩警报	1	0	127（0x7f）
19（0X13）	（保留）	1	0	1
24（0X18）	激活扭矩	1	0	1
25（0X19）	LED	1	0	1
26（0X1A）	顺时针柔性边距	1	0	254（0xfe）
27（0X1B）	逆时针柔性边距	1	0	254（0xfe）
28（0X1C）	顺时针柔性斜率	1	0	254（0xfe）
29（0X1D）	逆时针伺服斜率	1	0	254（0xfe）
30（0X1E）	目标位置	2	0	1023（0x3ff）
32（0X20）	运动速度	2	0	1023（0x3ff）
34（0X22）	扭矩限制	2	0	1023（0x3ff）
44（0X2C）	寄存器指令	1	0	1
47（0X2F）	锁定	1	1	1
48（0X30）	撞击	2	0	1023（0x3ff）

4. 指令设置和范例

指令	功能	值	参数量
Ping	无行为。用于得到状态包。	0x01	0
Read Data	读取控制表中数据。	0x02	2
Write Data	向控制表写数据。	0x03	2—
Reg Write	与上一指令类似。不过，接收到 Action 指令之前保持待机模式。	0x04	2—
Action	触发由 Reg Write 指令指定的行为。	0x05	0
Reset	将舵机控制表的值改为出厂默认值。	0x06	0
Sync Write	用于同时控制多个舵机	0x83	4—

4-1 Write_Data

功能　　向舵机控制表写数据。
长度　　N+2（N 是要写的数据的数量，最小数为 0）。
指令　　0x03
参数 1　　写入数据的存储单元的起始地址。
参数 2　　第 1 个要写入的数据
参数 3　　第 2 个要写入的数据
参数 N+1　　第 N 个要写入的数据

例 1　　设置被连接舵机的 ID 为 1。
向控制表的地址 3 写入 1。该 ID 用“广播 ID”（0xFE）指令传送。
指令包：0xFF 0xFF 0xFE 0x04 0x03 0x03 0x01 0xF6
0xFE：ID；　0x04：长度；　第 1 个 0x03：指令；
0x03 0x01：参数；　0xF6：校验码
因为 ID 是通过“广播 ID”指令传送，所以不返回状态包。

4-2 READ_Data

功能　　从舵机控制表中读取数据。
长度　　0X04
指令　　0X02
参数 1　　要读取的数据单元的起始地址。
参数 2　　要读取的数据长度

例 2　　读取 ID 为 1 的舵机的内部温度值。
从控制表中地址 0x2B 的单元读取 1 字节信息。
指令包：0XFF 0XFF 0X01 0X04 0X02 0X2B 0X01 0XCC
0x01：ID；　0x04：长度；　0x02：指令；
0x2B 0x01：参数；　0xCC：校验码
返回的状态包如下：
状态包：0XFF 0XFF 0X01 0X03 0X00 0X20 0XDB
0x01：ID；　0x03：长度；　0x00：误差；

0x20：参数； 0xDB：校验码
要读的数据是 0x20。因此舵机内部当前温度约为 32℃（0x20）。

4-3 REG_WRITE:ACTION

4-3-1 REG_WRITE

功能 REG_WRITE 功能类似 WRITE_DATA，但执行时机不同。当收到指令包时，其中的值存储在缓冲区，WRITE 指令处于待机状态。此时，Registered 指令寄存器（地址 0x2C）设为 1。收到 Action 指令包后，寄存的 Write 指令才最终执行。

长度 N+3（N 是要写入的数据个数）

指令 0x04

参数 1 要写入的数据起始存储地址

参数 2 第 1 个要写入的数据

参数 3 第 2 个要写入的数据

参数 N+1 第 N 个要写入的数据

4-3-2 ACTION

功能 执行 REG_WRITE 指令寄存的行为。

长度 0x02

指令 0x05

参数 无

ACTION 指令用于多个舵机需要同步转动时。当多个舵机同时转动时，第 1 台舵机和最后的舵机接收同一指令时可能出现微小的时间偏差，ACTION 指令可以解决该问题。

广播 广播 ID（0xFE）用于向 2 个以上舵机传送 ACTION 指令时。注意，这个操作不会传回反馈包。

4-4 PING

功能 不执行任何操作。用于询问状态包或者检查指定 ID 舵机是否存在。

长度 0x02

指令 0x01

参数 无

例 3 获取 ID 为 1 的舵机状态包。

指令包：0XFF 0XFF 0X01 0X02 0X01 0XFB

0x01：ID； 0x02：长度； 0x01：指令；
0xFB：校验码

状态包：0XFF 0XFF 0X01 0X02 0X00 0XFC

0x01：ID； 0x02：长度； 0x00：误差；
0xFC：校验码

不论是否使用广播 ID，还是状态返回水平（地址 16）为 0，Ping 指令总是会返回状态包。

4-5 RESET

功能 将舵机的控制表值改为出厂默认值。

长度 0x02

指令 0x06
参数 无

例 4 重设 ID 为 0 的舵机。

指令包：0XFF 0XFF 0X00 0X02 0X06 0XF7
0x00：ID； 0x02：长度； 0x06：指令；
0xF7：校验码

状态包：0XFF 0XFF 0X00 0X02 0X00 0XFD
0x00：ID； 0x02：长度； 0x00：误差；
0xFD：校验码

注意，执行重设指令后该舵机 ID 改为 1。

4-6 SYNC WRITE

功能 用于同时控制多个舵机。执行 Sync Write 指令后，多条指令可通过一条指令传送，通讯时间缩短了。但是，该指令只能在要写入的控制表值长度和地址相同时使用。这里同样要用到广播 ID。

ID 0xFE
长度 （L+1）*N+4（L：每个舵机接收的数据长度，N：舵机数量）
指令 0x83
参数 1 要写入的数据起始地址
参数 2 要写入的数据长度（L）
参数 3 第 1 个舵机的 ID
参数 4 第 1 个舵机的第 1 个数据
参数 5 第 1 个舵机的第 2 个数据
参数 L+3 第 1 个舵机的第 L 个数据
…… 第 1 个舵机的数据
参数 L+4 第 2 个舵机的 ID
参数 L+5 第 2 个舵机的第 1 个数据
参数 L+6 第 2 个舵机的第 2 个数据
…… 第 2 个舵机的数据
参数 2L+4 第 2 个舵机的第 L 个数据

例 5 对 4 个舵机设置以下位置和速度。

ID0 的舵机：位置 0X010，速度 0X150
ID1 的舵机：位置 0X220，速度 0X360
ID2 的舵机：位置 0X030，速度 0X170
ID3 的舵机：位置 0X220，速度 0X380

指令包：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

由于使用了广播 ID，无状态包反馈。

5. 范例

在下面例子中，假定 ID 为 1 的舵机处于重置状态，波特率为 57142 bps。

例 6 读取舵机 ID1 的型号和固化文件版本。

指令包 指令=READ_DATA，地址=0x00，长度=0x03

通讯 ->[Dynamixel]: FF FF 01 04 02 00 03 05(LEN:008)

<-[Dynamixel]: FF FF 01 05 00 74 00 08 7D(LEN:009)

状态包结果 型号=116（0x74）（指代 DX-116），固化文件版本=0x08

例 7 改变舵机 ID1 为 ID0。

指令包 指令=WRITE_DATA，地址=0x03，长度=0x00

通讯 ->[Dynamixel]: FF FF 01 04 03 03 00 F4(LEN:008)

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

状态包结果 无误差

例 8 修改舵机波特率至 1M bps。

指令包 指令=WRITE_DATA，地址=0x04，长度=0x01

通讯 ->[Dynamixel]: FF FF 00 04 03 04 01 F3(LEN:008)

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

状态包结果 无误差

例 9 重置 ID0 的舵机反馈延迟时间为 4μs。

ID1 的反馈延迟时间为 2μs。

指令包 指令=WRITE_DATA，地址=0x05，长度=0x02

通讯 ->[Dynamixel]: FF FF 00 04 03 05 02 F1(LEN:008)

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

状态包结果 无误差

建议设置反馈延迟时间为主控制器允许的最小值。

例 10 限制 ID0 的舵机的操作角度为 0°—150°。

由于逆时针角度边界 0x3ff 对应 300°，角度 150° 由 0x1ff 表示。

指令包 指令=WRITE_DATA，地址=0x08，数据=0xff，0x01

通讯 ->[Dynamixel]: FF FF 00 05 03 08 FF 01 EF(LEN:009)

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

状态包结果 无误差

例 11 设置 ID0 的舵机工作电压为 10V—17V。

10V 由 100（0x64）表示，17V 由 170（0xAA）表示。

指令包 指令=WRITE_DATA，地址=0x0C，数据=0x64，0xAA

通讯 ->[Dynamixel]: FF FF 00 05 03 0C 64 AA DD(LEN:009)

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

状态包结果 无误差

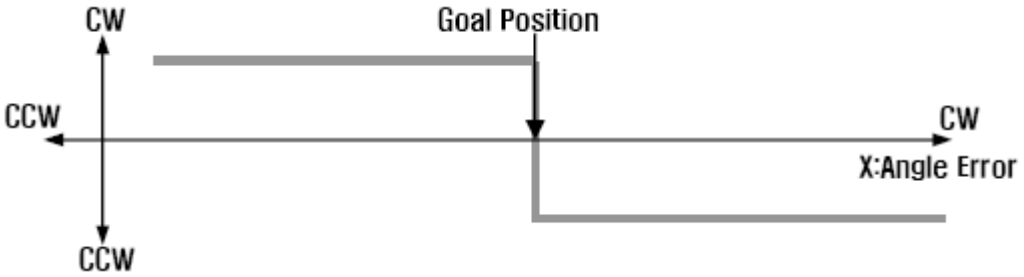
例 12 设置 ID0 的舵机最大扭矩为其最大可能值的 50%。
设置 ROM 中的最大扭矩值 0x1ff，是最大可能值 0x3ff 的 50%。
指令包 指令=WRITE_DATA，地址=0x0E，数据=0xff，0x01
通讯 ->[Dynamixel]: FF FF 00 05 03 0E FF 01 E9(LEN:009)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)
状态包结果 无误差
要验证经过改动后的最大扭矩值，需要重新开启电源。

例 13 设置 ID0 的舵机从不返回状态包。
指令包 指令=WRITE_DATA，地址=0x10，数据=0x00
通讯 ->[Dynamixel]: FF FF 00 04 03 10 00 E8 (LEN:008)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)
状态包结果 无误差
无状态包返回，直接进入下一条指令。

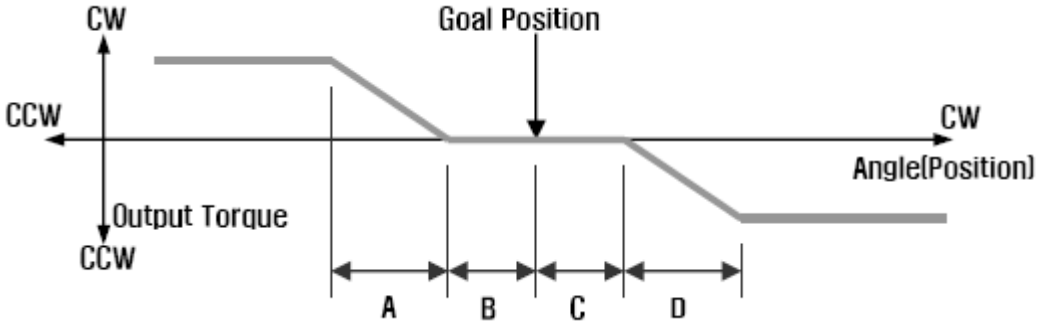
例 14 当工作温度超过限制时，让 LED 发光报警，舵机扭矩撤消。
由于过热错误是 Bit 2，故设定警报值为 0x04。
指令包 指令=WRITE_DATA，地址=0x11，数据=0x04，0x04
通讯 ->[Dynamixel]: FF FF 00 05 03 11 04 04 DE (LEN:009)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)
状态包结果 无误差

例 15 打开 LED，激活 ID0 的舵机的扭矩。
指令包 指令=WRITE_DATA，地址=0x18，数据=0x01，0x01
通讯 ->[Dynamixel]: FF FF 00 05 03 18 01 01 DD (LEN:009)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)
状态包结果 无误差
可通过用手扳动舵机输出轴来检验舵机是否激活。

例 16 设置 ID0 舵机的柔性边界为 1，柔性斜率为 0x40。
柔性 角度误差和扭矩输出可用下图表示。



即使顺时针方向舵机位置稍稍偏离了目标位置，在逆时针方向会出现反向扭矩补偿这个偏离。但由于必须考虑惯性，实际的补偿有所不同。考虑到这一点，实际情形如下图所示：



A：逆时针柔性斜率（地址 0x1D）= 0x40（约 18.8°）
B：逆时针柔性边界（地址 0x1B）= 0x01（约 0.29°）
C：顺时针柔性边界（地址 0x1A）= 0x01（约 0.29°）
D：顺时针柔性斜率（地址 0x1C）= 0x40（约 18.8°）

指令包 指令=WRITE_DATA，地址=0x1A，数据=0x01,0x01,0x40,0x40
通讯 ->[Dynamixel]: FF FF 00 07 03 1A 01 01 40 40 59 (LEN:011)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)
状态包结果 无误差
任何位于 0x11 和 0x20 之间的柔性斜率效用相同。

例 17 以 57 RPM 的角速度将 ID0 的舵机输出位置设置为 180°。
给地址 0x1E（目标位置）赋值 0x200，地址 0x20（运动速度）赋值 0x200。
指令包 指令=WRITE_DATA，地址=0x1E，数据=0x00,0x02,0x00,0x02
通讯 ->[Dynamixel]: FF FF 00 07 03 1E 00 02 00 02 D3 (LEN:011)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)
状态包结果 无误差

例 18 设置 ID0 的舵机输出位置为 0°，ID1 的输出位置为 300°，并同时启动两个舵机。
如果用 WRITE_DATA，两个舵机的运动无法同步，所以要用 REG_WRITE 和 ACTION 代替。
指令包 ID=0，指令=REG_WRITE，地址=0x1E，数据=0x00，0x00
ID=0，指令=REG_WRITE，地址=0x1E，数据=0xff，0x03
ID=0xfe（广播 ID），指令=ACTION
通讯 ->[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]: //广播 ID 无反馈
状态包结果 无误差

例 19 锁定 ID0 的舵机除 0x18—0x23 以外的所有地址。
给地址 0x2F（锁定）赋值 1。
指令包 指令=WRITE_DATA，地址=0x2F，数据=0x01

通讯 ->[Dynamixel]: FF FF 00 04 03 2F 01 C8 (LEN:008)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)

状态包结果 无误差

一旦锁定，唯一解锁的方法就是关闭电源。
任何存取被锁定数据的尝试都会显示错误。
->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 08 F5 (LEN:006)

值域错误

例 20 设置 ID0 的舵机的最小功率为 0x40。

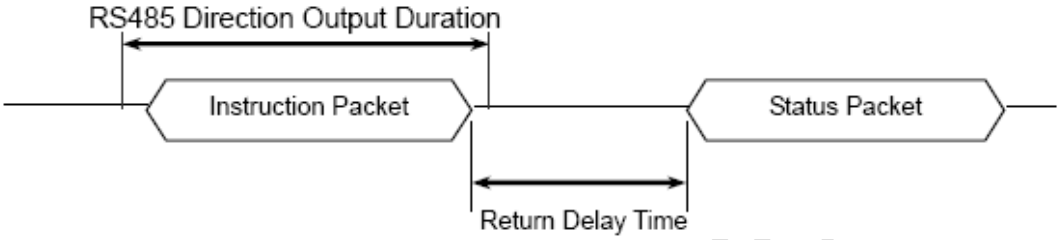
指令包 指令=WRITE_DATA，地址=0x30，数据=0x40，0x00

通讯 ->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]: FF FF 00 02 00 FD(LEN:006)

状态包结果 无误差

附件

半双工 UART 半双工 UART 是一种串口通讯协议，其中规定不能同时执行 TxD 和 RxD。这种协议通常用于需要将众多设备连接到单一总线上时。由于不只一个设备连接到同一总线，因此当某个设备发送数据时，其它设备都要处于输入模式。主控制器将通讯方向设置为输入模式，只有当它传送指令包时，才改为输出模式。



反馈延迟时间 指的是舵机收到指令包后反馈状态包所需的时间。默认值为 160 μs，可通过控制表上地址 5 修改。主控制器发出指令包后，在反馈延迟时间段需要改动方向端口至输入模式。

Tx，Rx 方向 在半双工 UART 中，发送结束时间对于改变方向端口至接收模式很重要。注册表内显示 UART_STATUS 的位定义如下：

TXD_BUFFER_READY_BIT：显示传送数据可以装入缓存区中。注意这仅仅意味着 SERIAL_TX_BUFFER 为空，不一定说明所有之前传送的数据已离开 CPU。

TXD_SHIFT_REGISTER_EMPTY_BIT：当所有传送数据完成传送并离开 CPU 时设置该值。

The TXD_BUFFER_READY_BIT 用于通过串口传送单个字节时。举例如下：

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

改变传送方向时，必须校验 TXD_SHIFT_REGISTER_EMPTY_BIT。

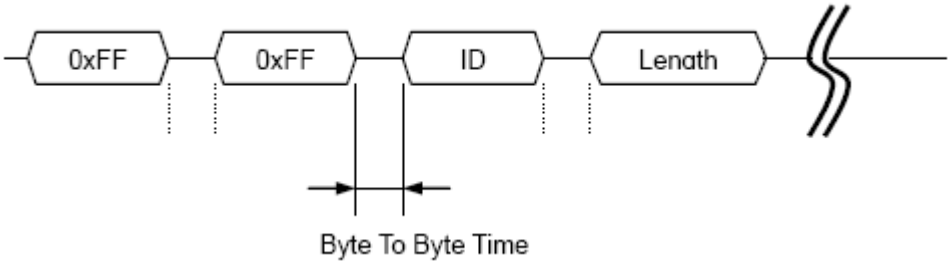
下面是发送指令包的程序范例：

```
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
LINE10 while(!TXD_SHIFT_REGISTER_EMPTY_BIT); //Wait till last data bit has been sent
LINE11 DIRECTION_PORT = RX_DIRECTION; //Direction change to RXD
LINE12 EnableInterrupt(); // enable interrupt again

请注意第 8 行到 12 行的重要语句。由于中断可能造成延迟比反馈延迟时间长，并导致状态包传送错误，所以第 8 行是必需的。

字节到字节时间 指的是发送指令包时字节间的延迟时间。如果该时间大于 100ms，那么舵机将认为出现了通讯问题，等待下一个指令包的头代码（oxff oxff）。



下面是一个应用于 Atmega 128 的范例程序源代码。

```
C Language Example : Dinamixel access with Atmega128
/*
* The Example of Dynamixel Evaluation with Atmega128
* Date : 2005.5.11
* Author : BS KIM
*/
/*
* included files
*/
#define ENABLE_BIT_DEFINITIONS
//#include <io.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_MODOEL_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 TxD8 TxD81
#define RxD8 RxD81
//Hardware Dependent Item
#define DEFAULT_BAUD_RATE 34 //57600bps at 16MHz
////// For CM-5
#define RS485_TXD PORTE &= ~_BV(PE3),PORTE |= _BV(PE2)
//PORT_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 TXD0_FINISH UCSR0A,6 //This bit is for checking TxD Buffer
in CPU is empty or not.
```

```
//#define TXD1_FINISH UCSR1A,6
#define SET_TxD0_FINISH sbi(UCSR0A,6)
#define RESET_TXD0_FINISH cbi(UCSR0A,6)
#define CHECK_TXD0_FINISH bit_is_set(UCSR0A,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 bTxdData);
void TxD80(byte bTxdData);
void TxDString(byte *bData);
void TxD8Hex(byte bSentData);
void TxD32Dec(long lLong);
byte RxD81(void);
void MiliSec(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 *bpPrintBuffer, byte bLength);
// --- Gloval Variable Number ---
volatile byte gbpRxInterruptBuffer[256];
byte gbpParameter[128];
byte gbRxBufferReadPointer;
byte gbpRxBuffer[128];
byte gbpTxBuffer[128];
volatile byte gbRxBufferWritePointer;
int main(void)
{
byte bCount,bID, bTxPacketLength,bRxPacketLength;
PortInitialize(); //Port In/Out Direction Definition
RS485_RXD; //Set RS485 Direction to Input State.
SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//RS485
Initializing(RxInterrupt)
```

```
SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,0); //RS232
Initializing(None Interrupt)
gbRxBufferReadPointer = gbRxBufferWritePointer = 0; //RS485
RxBuffer Clearing.
sei(); //Enable Interrupt -- Compiler Function
TxDString("¥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;
TxDString("¥r¥n¥n Example 1. Scanning Dynamixels(0~9). -- Any Key to
Continue."); RxD8();
for(bCount = 0; bCount < 0x0A; bCount++)
{
bTxPacketLength = TxPacket(bCount,INST_PING,0);
bRxPacketLength = RxPacket(255);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString(", RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE)
{
TxDString(" Found!! ID:");TxD8Hex(bCount);
bID = bCount;
}
}
TxDString("¥r¥n¥n Example 2. Read Firmware Version. -- Any Key to
Continue."); RxD8();
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]);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
{
TxDString("¥r¥n Return Error : ");TxD8Hex(gbpRxBuffer[4]);
```

```
TxDString("¥r¥n Firmware Version : ");TxD8Hex(gbpRxBuffer[5]);
}
TxDString("¥r¥n¥n Example 3. LED ON -- Any Key to Continue.");
RxD8();
gbpParameter[0] = P_LED; //Address of LED
gbpParameter[1] = 1; //Writing Data
bTxPacketLength = TxPacket(bID,INST_WRITE,2);
bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
TxDString("¥r¥n¥n Example 4. LED OFF -- Any Key to Continue.");
RxD8();
gbpParameter[0] = P_LED; //Address of LED
gbpParameter[1] = 0; //Writing Data
bTxPacketLength = TxPacket(bID,INST_WRITE,2);
bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
TxDString("¥r¥n¥n Example 5. Read Control Table. -- Any Key to
Continue."); RxD8();
gbpParameter[0] = 0; //Reading Address
gbpParameter[1] = 49; //Read Length
bTxPacketLength = TxPacket(bID,INST_READ,2);
bRxPacketLength =
RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter
[1]);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
{
TxDString("¥r¥n");
for(bCount = 0; bCount < 49; bCount++)
{
TxD8('[');TxD8Hex(bCount);TxDString("]:");
TxD8Hex(gbpRxBuffer[bCount+5]);TxD8(' ');
}
}
TxDString("¥r¥n¥n Example 6. Go 0x200 with Speed 0x100 -- Any Key to
Continue."); RxD8();
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);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
TxDString("¥r¥n¥n Example 7. Go 0x00 with Speed 0x40 -- Any Key to
Continue."); RxD8();
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);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
TxDString("¥r¥n¥n Example 8. Go 0x3ff with Speed 0x3ff -- Any Key to
Continue."); RxD8();
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);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
TxDString("¥r¥n¥n Example 9. Torque Off -- Any Key to Continue.");
RxD8();
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);
TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
TxDString("¥r¥n¥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(Paramter,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(UCSR0A,6);//SET_TXD0_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("¥r¥n [Error:RxD Timeout]");
CLEAR_BUFFER;
}
if(bLength > 3) //checking is available.
{
if(gbpRxBuffer[0] != 0xff || gbpRxBuffer[1] != 0xff )
```

```
{
TxDString("¥r¥n [Error:Wrong Header]");
CLEAR_BUFFER;
return 0;
}
if(gbpRxBuffer[2] != gbpTxBuffer[2] )
{
TxDString("¥r¥n [Error:TxID != RxID]");
CLEAR_BUFFER;
return 0;
}
if(gbpRxBuffer[3] != bLength-4)
{
TxDString("¥r¥n [Error:Wrong Length]");
CLEAR_BUFFER;
return 0;
}
for(bCount = 2; bCount < bLength; bCount++) bChecksum +=
gbpRxBuffer[bCount];
if(bChecksum != 0xff)
{
TxDString("¥r¥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. /*
*/ TXD81() send data to USART 1.
void PrintBaudrate(void) */
{ void TxD81(byte bTxdData)
TxDString("¥r¥n
RS232:");TxD32Dec((16000000L/8L)/((long)UBRR1L+1
L) ); TxDString(" BPS,");
{
while(!TXD1_READY);
TXD1_DATA = bTxdData;
TxDString(" RS485:");TxD32Dec((16000000L/8L)/((long)UBRR0L+1L) );
TxDString(" BPS");
}
} /*
TXD32Dex() change data to decimal number system
*/
/*Hardware Dependent Item*/ void TxD32Dec(long lLong)
#define TXD1_READY bit_is_set(UCSR1A,5)
//(UCSR1A_Bit5)
{
byte bCount, bPrinted;
#define TXD1_DATA (UDR1) long lTmp,lDigit;
#define RXD1_READY bit_is_set(UCSR1A,7) bPrinted = 0;
#define RXD1_DATA (UDR1) if(lLong < 0)
{
#define TXD0_READY bit_is_set(UCSR0A,5) lLong = -lLong;
#define TXD0_DATA (UDR0) TxD8('-');
#define RXD0_READY bit_is_set(UCSR0A,7) }
#define RXD0_DATA (UDR0) lDigit = 1000000000L;
for(bCount = 0; bCount < 9; bCount++)
/* {
SerialInitialize() set Serial Port to initial state. lTmp = (byte)(lLong/lDigit);
Vide Mega128 Data sheet about Setting bit of register. if(lTmp)
SerialInitialize() needs port, Baud rate, Interrupt value. {
TxD8(((byte)lTmp)+'0');
*/ bPrinted = 1;
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt) }
{ else if(bPrinted) TxD8(((byte)lTmp)+'0');
if(bPort == SERIAL_PORT0) lLong -= ((long)lTmp)*lDigit;
{ lDigit = lDigit/10;
UBRR0H = 0; UBRR0L = bBaudrate; }
UCSR0A = 0x02; UCSR0B = 0x18; lTmp = (byte)(lLong/lDigit);
if(bInterrupt&RX_INTERRUPT) sbi(UCSR0B,7); // RxD interrupt enable /*if(lTmp)*/
TxD8(((byte)lTmp)+'0');
```

```
UCSR0C = 0x06; UDR0 = 0xFF; }
sbi(UCSR0A,6);//SET_TXD0_FINISH; // Note. set 1, then 0 is read
} /*
else if(bPort == SERIAL_PORT1) TxDString() prints data in ACSII code.
{ */
UBRR1H = 0; UBRR1L = bBaudrate; void TxDString(byte *bData)
UCSR1A = 0x02; UCSR1B = 0x18; {
if(bInterrupt&RX_INTERRUPT) sbi(UCSR1B,7); // RxD interrupt enable while(*bData)
UCSR1C = 0x06; UDR1 = 0xFF; {
sbi(UCSR1A,6);//SET_TXD1_FINISH; // Note. set 1, then 0 is read TxD8(*bData++);
} }
} }
/* /*
TxD8Hex() print data seperatly. RxD81() read data from UART1.
ex> 0x1a -> '1' 'a'. RxD81() return Read data.
*/ */
void TxD8Hex(byte bSentData) byte RxD81(void)
{ {
byte bTmp; while(!RXD1_READY);
return(RXD1_DATA);
bTmp =((byte)(bSentData>>4)&0x0f) + (byte)'0'; }
if(bTmp > '9') bTmp += 7;
TxD8(bTmp); /*
bTmp =(byte)(bSentData & 0x0f) + (byte)'0'; SIGNAL() UART0 Rx Interrupt - write data to buffer
if(bTmp > '9') bTmp += 7; */
TxD8(bTmp); SIGNAL (SIG_UART0_RECV)
} {
gbpRxInterruptBuffer[(gbRxBufferWritePointer++)] = RXD0_DATA;
/* }
TxD80() send data to USART 0.
*/
void TxD80(byte bTxdData)
{
while(!TXD0_READY);
TXD0_DATA = bTxdData;
```

CM-5 AX-12专用控制器，可控制30个AX-12舵机。
6个按钮，其中5个用于选择功能，1个用于重启。