机器人舵机 CDS55xx 用户手册

ProMotion CDS55xx

Manual

文档概述：CDS55xx 用户手册				
文档版本号	更新时间	修订人	审定人	备注
1.0	2010-3-22	何裕德		
1.1	2010-3-27	计海锋		加入第五章内容
1.2	2010-4-22	何裕德		调整内存控制表，列出型号间差异

感谢您阅读本文档。本文档当前为试用版本。由于时间、水平有限，错漏在所难免，请您不吝指正。

对本文的意见、建议和疑问都可以到“北京博创论坛”发帖讨论，网址是：http://robot.up-tech.com/bbs/index.asp?boardid=1，我们将及时为您解答。

您也可以通过电话与北京博创取得联系：

技术服务电话：86-10-82114870/4887/4890

技术服务邮箱：robot_service@up-tech.com

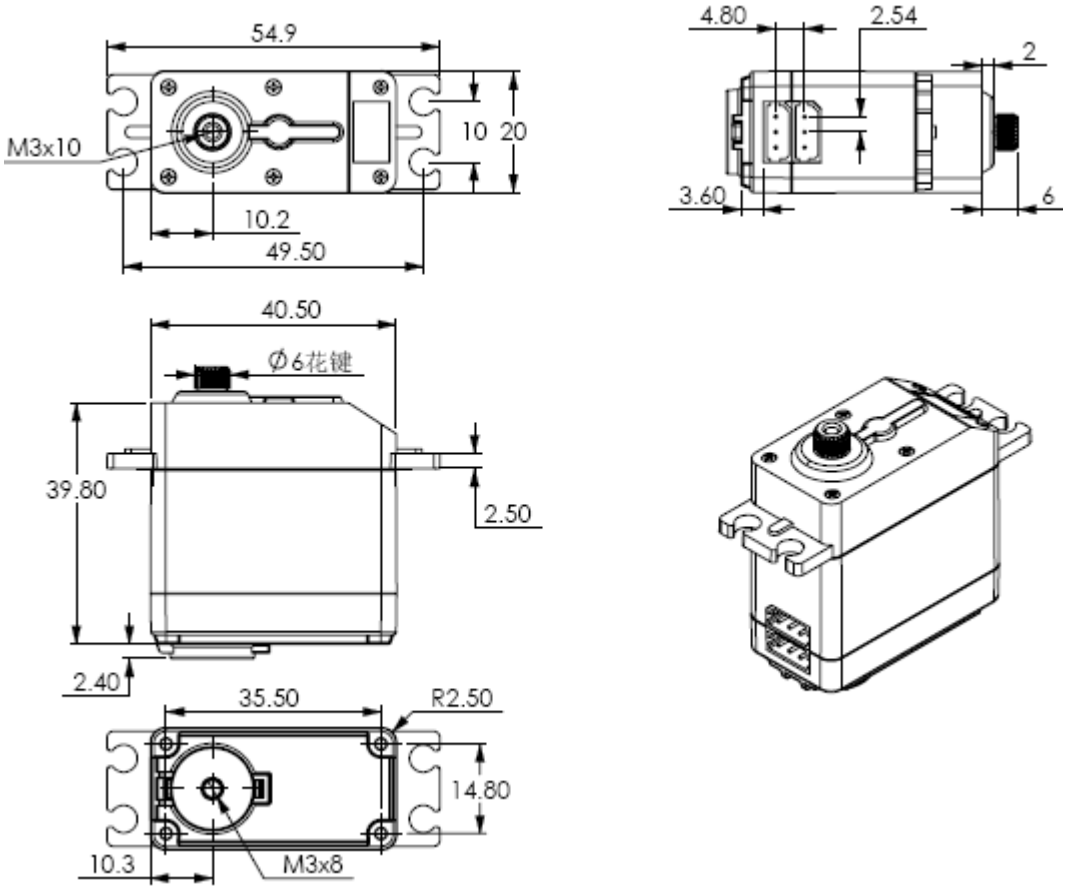
第一章 部件概述

1.1 产品特性

proMOTION CDS 系列机器人舵机属于一种集电机、伺服驱动、总线式通讯接口为一体的集成伺服单元，主要用于微型机器人的关节、轮子、履带驱动，也可用于其他简单位置控制场合。CDS 55XX 的特点如下所示：

- 大扭矩：16Kgf.cm
- 高转速：最高 0.16s/60°输出转速
- 宽电压范围供电
  CDS5500:DC 6.6V～10V
  CDS5516:DC 6.0V～16V
- 0.32°位置分辨率
- 双端安装方式，适合安装在机器人关节
- 高精度全金属齿轮组，双滚珠轴承
- 连接处 O 型环密封，防尘防溅水
- 位置伺服控制模式下转动范围 0-300°
- 可设置为电机模式整周旋转，开环调速
- 总线连接，理论可串联 254 个单元
- 高达 1M 通讯波特率
- 0.25KHz 的伺服更新率
- 兼容 Robotis Dynamixel 通讯协议
- 具备位置、温度、速度、电压反馈

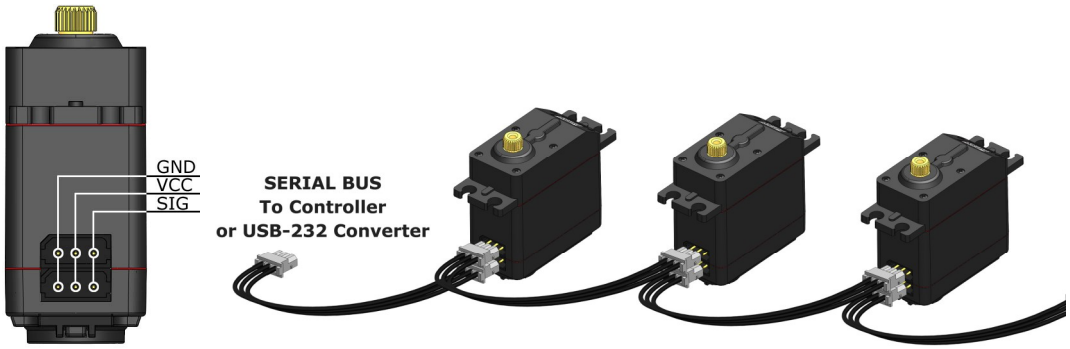
1.2 结构尺寸



1.3 电气连接

1.3.1 引脚定义

proMOTION CDS 系列机器人舵机电气接口如下图，两组引脚定义一致的接线端子可将舵机逐个串联起来。

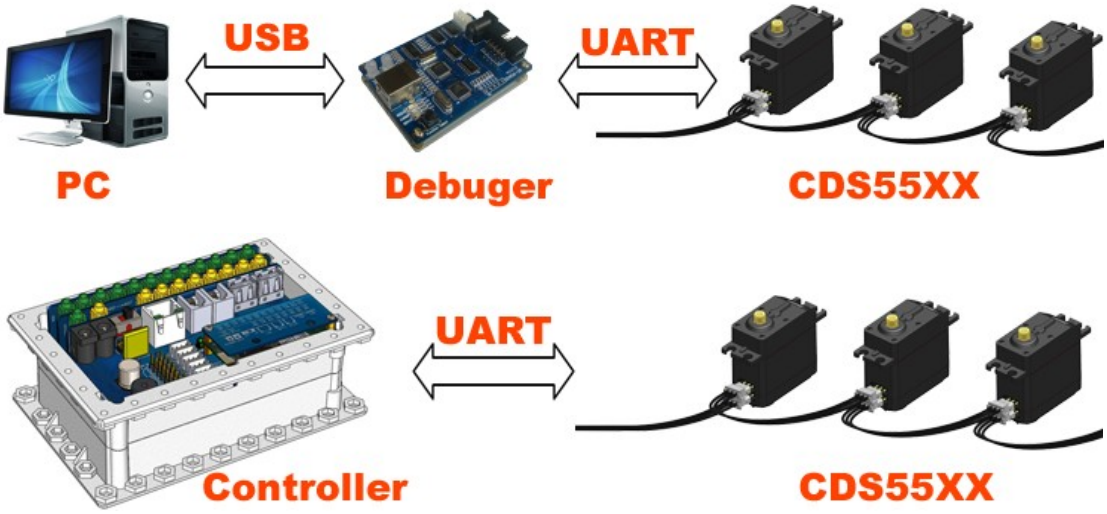


1.3.2 舵机通讯方式

CDS55xx 采用异步串行总线通讯方式，理论多至 254 个机器人舵机可以通过总线组成链型，通过 UART 异步串行接口统一控制。每个舵机可以设定不同的节点地址，多个舵机可以统一运动也可以单个独立控制。

CDS55xx 的通讯指令集开放 通过异步串行接口与用户的上位机(控制器或 PC 机)通讯，您可对其进行参数设置、功能控制。通过异步串行接口发送指令，CDS55xx 可以设置为电机控制模式或位置控制模式。 在电机控制模式下，CDS55xx 可以作为直流减速电机使用，速度可调；在位置控制模式下，CDS55xx 拥有 0-300°的转动范围，在此范围内具备精确位置控制性能，速度可调。

只要符合协议的半双工 UART 异步串行接口都可以和 CDS55xx 进行通讯 ,对 CDS55xx 进行各种控制。主要有以下两种形式：



方式 1：通过调试器控制 CDS55xx

PC 机会将调试器识别为串口设备，上位机软件通过串口发出符合协议格式的数据包，经调试器转发给 CDS55xx。CDS55xx 会执行数据包的指令，并且返回应答数据包。

RobotServoTerminal 是博创推荐调试软件，您也可根据本手册提供的协议设计专用的 PC 端软件。

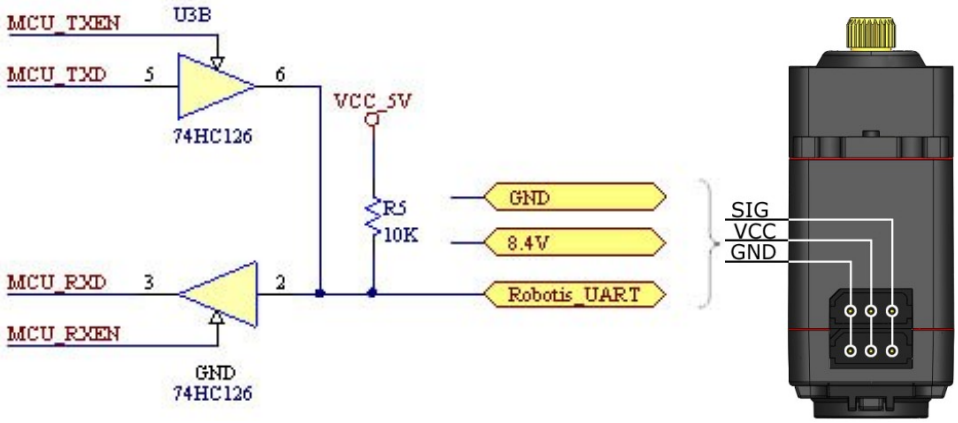
方式 2：通过专用控制器控制 CDS55xx

方式 1 可以快捷地调试 CDS 系列机器人舵机、修改各种性能与功能参数。但是，这种方式离不开 PC 机，不能搭建独立的机器人构型。您可以设计专用的控制器，通过控制器的 UART 端口控制舵机，1.3.3 小节给出了控制器 UART 接口原理图，第五章给出了控制器的最小系统设计说明，您可以依此设计出专用控制器。

1.3.3 UART 接口原理图

CDS 系列机器人舵机用程序代码对 UART 异步串行接口进行时序控制，实现半双工异步串行总线通讯，通讯速度可高达 1Mbps，且接口简单、协议精简。

在您自行设计的控制器中，用于和 CDS55xx 通讯的 UART 接口必须如下图所示进行处理。



第二章 通讯协议

2.1 通信协议概要

控制器和舵机之间采用问答方式通信，控制器发出指令包，舵机返回应答包。

一个网络中允许有多个舵机，所以每个舵机都分配有一个 ID 号。控制器发出的控制指令中包含 ID 信息，只有匹配上 ID 号的舵机才能完整接收这条指令，并返回应答信息。

通信方式为串行异步方式，一帧数据分为 1 位起始位，8 位数据位和 1 位停止位，无奇偶校验位，共 10 位。

2.2 指令包

指令包格式：

字头	ID号	数据长度	指令	参数	校验和
0XFF 0XFF	ID	Length	Instruction	Parameter1 ...Parameter N	Check Sum

字头：连续收到两个 0XFF ,表示有数据包到达。

ID: 每个舵机都有一个 ID 号。ID 号范围 0～253,转换为十六进制 0X00～0XFD。

广播 ID: ID 号 254 为广播 ID,若控制器发出的 ID 号为 254(0XFE)，所有的舵机均接收指令，但都不返回应答信息。

数据长度：等于待发送的参数长度 N 加上 2，即“N+2”。

参数：除指令外需要补充的控制信息。

校验和：校验和 Check Sum，计算方法如下：

Check Sum = ~ (ID + Length + Instruction + Parameter1 + ... Parameter N)

若括号内的计算和超出 255,则取最低的一个字节，“~”表示取反。

2.3 应答包

应答包是 CDS55xx 对控制器的应答，应答包格式如下：

字头	ID	数据长度	当前状态	参数	校验和
0XFF 0XFF	ID	Length	ERROR	Parameter1 ...Parameter N	Check Sum

返回的应答包包含舵机的当前状态 ERROR，若舵机当前工作状态不正常，会通过这个字节反映出来，每一位的代表的信息如下：

BIT	名称	详细
BIT7	0	---
BIT6	指令错误	如果收到一个未定义的指令或收到ACTION前未收到REG WRITE指令置1
BIT5	过载	位置模式运行时输出扭矩小于负载置1
BIT4	校验和错	校验和错误置1
BIT3	指令超范围	指令超过指定范围置1
BIT2	过热	温度超过指定范围置1
BIT1	角度超范围	角度超过设定范围置1
BIT0	过压欠压	电压超过指定范围置1

若 ERROR 为 0，则舵机无报错信息。

若指令是读指令 READ DATA，则 Parameter1 ...Parameter N 是读取的信息。
其他数据的含义和指令包一样。

2.4 指令类型

可用指令如下：

指令	功能	值	参数长度
PING（查询）	查询工作状态	0x01	0
READ DATA（读）	查询控制表里的字符	0x02	2
WRITE DATA（写）	往控制表里写入字符	0x03	不小于2
REG WRITE（异步写）	类似于WRITE DATA，但是控制字符写入后并不马上动作，直到ACTION指令到达	0x04	不小于2
ACTION（执行异步写）	触发REG WRITE写入的动作	0x05	0
RESET（复位）	把控制表复位为出厂值	0x06	0
SYNC WRITE（同步写）	用于同时控制多个CDS55xx	0x83	不小于4

2.4.1 写指令 WRITE DATA

功能 写数据到 CDS55xx 的控制表
长度 N+3 (N 为写入数据的长度)
指令 0X03
参数 1 数据写入段的首地址
参数 2 写入的第一个数据
参数 3 第二个数据
参数 N+1 第 N 个数据

例 1 把一个任意编号的 CDS55xx 的 ID 设置为 1。
在控制表里保存 ID 号的地址为 3（控制表见后文），所以在地址 3 处写入 1 即可. 发送指令包的 ID 使用广播 ID (0xFE)。
指令帧： 0XFF 0XFF 0XFE 0X04 0X03 0X03 0X01 0XF6

字头		ID	有效数据长度	指令	参数		校验和
0XFF	0XFF	0XFE	0X04	0X03	0X03	0X01	0XF6

因为采用广播 ID 发送指令，所以不会有数据返回
校验和的计算方法如下：
Check Sum = ~ (ID + Length + Instruction + Parameter1 + ... Parameter N)
若计算的和超出 255,则取最低的一个字节，“~”表示取反。

2.4.2 读指令 READ DATA

功能 从CDS55xx的控制表里读出数据
长度 0X04
指令 0X02
参数1 数据读出段的首地址
参数2 读取数据的长度

例 2 读取 ID 为 1 的 CDS55xx 的内部温度。
在控制表里从地址 0X2B 处读取一个字节。
指令帧：0XFF 0XFF 0X01 0X04 0X02 0X2B 0X01 0XCC

字头		ID	有效数据长度	指令	参数		校验和
0XFF	0XFF	0X01	0X04	0X02	0X2B	0X01	0XCC

返回的数据帧： 0XFF 0XFF 0X01 0X03 0X00 0X20 0XDB

字头		ID	有效数据长度	工作状态	参数	校验和
0XFF	0XFF	0X01	0X03	0X00	0X20	0XDB

读出的数据是 0x20，说明当前的温度约是 32℃（0x20）。

2.4.3 异步写指令 REG WRITE

REG WRITE 指令相似于 WRITE DATA，只是执行的时间不同。当收到 REG WRITE 指令帧时，把收到的数据储存在缓冲区备用，并把 Registered Instruction 寄存器（Address 0x2c）置 1。当收到 ACTION 指令后，储存的指令最终被执行。

长度 N+3 (N 为要写入数据的个数)
指令 0X04
参数 1 数据要写入区的首地址
参数 2 要写入的第一个数据
参数 3 要写入的第二个数据
参数 N+1 要写入的第 N 个数据

2.4.4 执行异步写指令 ACTION

功能 触发 REG WRITE 指令
长度 0X02
指令 0X05
参数 无

ACTION 指令在同时控制多个 CDS55xx 时非常有用。

在控制多个独立的 CDS55xx 时 ,使用 ACTION 指令可以使第一个和最后一个 CDS55xx 同时执行各自的动作，中间无延时。

对多个 CDS55xx 发送 ACTION 指令时，要用到广播 ID（0xFE）,因此，发送此指令不会有数据帧返回。

2.4.5 查询状态指令 PING

功能 读取 CDS55xx 的工作状态
长度 0X02
指令 0X01
参数 无

例 3 读取 ID 号为 1 的 CDS55xx 的工作状态

指令帧：0XFF 0XFF 0X01 0X02 0X01 0XFB`

字头		ID	有效数据长度	指令	校验和
0XFF	0XFF	0X01	0X02	0X01	0XFB

返回的数据帧： 0XFF 0XFF 0X01 0X02 0X00 0XFC

字头		ID	有效数据长度	工作状态	校验和
0XFF	0XFF	0X01	0X02	0X00	0XFC

不管是广播 ID 还是 Return Level (Address 16)等于 0，只要发送 PINE 指令且校验和正确，都会返回舵机工作状态。

2.4.6 复位指令 RESET

功能 把控制表里的数据复位为出厂值
长度 0X02
指令 0X06
参数 无

例 4 复位 CDS55xx，ID 号为 0。

指令帧：0XFF 0XFF 0X00 0X02 0X06 0XF7`

字头	ID	有效数据长度	指令	校验和
0XFF 0XFF	0X00	0X02	0X06	0XF7

返回的数据帧：0XFF 0XFF 0X00 0X02 0X00 0XFD

字头	ID	有效数据长度	工作状态	校验和
0XFF 0XFF	0X00	0X02	0X00	0XFD

2.4.7 同步写指令 SYNC WRITE

功能 用于同时控制多个 CDS55xx。
ID 0XFE
长度 (L + 1) * N + 4 (L: 发给每个 CDS55xx 的数据长度, N: CDS55xx 的个数)
指令 0X83
参数 1 写入数据的首地址
参数 2 写入的数据的长度(L)
参数 3 第一个 CDS55xx 的 ID 号
参数 4 写入第一个 CDS55xx 的第一个数据
参数 5 写入第一个 CDS55xx 的第二个数据
…
参数 L+3 写入第一个 CDS55xx 的第 L 个数据
参数 L+4 第二个 CDS55xx 的 ID 号
参数 L+5 写入第二个 CDS55xx 的第一个数据
参数 L+6 写入第二个 CDS55xx 的第二个数据
…
参数 2L+4 写入第二个 CDS55xx 的第 L 个数据
….

不同于 REG WRITE+ACTION 指令的是实时性比它更高，一条 SYNC WRITE 指令可一次修改多个 CDS55xx 的控制表内容，而 REG WRITE+ACTION 指令是分步做到的。 尽管如此，使用 SYNC WRITE 指令时，写入的数据长度和保存数据的首地址必须相同即必须执行相同的动作。

例 5 对 4 个 CDS55xx 写入如下的位置和速度

ID0: 位置：0X010；速度:0X150
ID1: 位置：0X220；速度:0X360
ID2: 位置：0X030；速度:0X170
ID3: 位置：0X220；速度:0X380

指令帧：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

字头	ID	有效数据长度	指令	参数	校验和
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

因为采用广播 ID 发送指令，所以不会有数据返回。

第三章 内存控制表

3.1 内存控制表

机器人舵机本身的信息和控制参数形成了一张表，保存在其控制芯片的 RAM 和 EEPROM 区域。我们通过实时修改表里的内容，可以达到实时控制舵机的目的。这张表称为内存控制表，内容如下：

地址	命令项	读写	初始值	存储区域
0（0x00）	--	--	--	EEPROM
1（0x01）	--	--	--	
2（0x02）	软件版本	读	--	
3（0x03）	ID	读/写	1（0x01）	
4（0x04）	波特率	读/写	1（0x01）	
5（0x05）	返回延迟时间	读/写	0（0x00）	
6（0x06）	顺时针角度限制（L）	读/写	0（0x00）	
7（0x07）	顺时针角度限制（H）	读/写	0（0x00）	
8（0x08）	逆时针角度限制（L）	读/写	255（0xFF）	
9（0x09）	逆时针角度限制（H）	读/写	3（0x03）	
10（0x0A）	--			
11（0x0B）	最高温度上限	读/写	80（0x50）	
12（0x0C）	最低输入电压	读/写	?	
13（0x0D）	最高输入电压	读/写	?	
14（0x0E）	最大扭矩（L）	读/写	255（0xFF）	
15（0x0F）	最大扭矩（H）	读/写	3（0x03）	
16（0x10）	应答状态级别	读/写	2（0x02）	
17（0x11）	LED闪烁	读/写	37（0x25）	
18（0x12）	卸载条件	读/写	4（0x04）	
19（0x13）	--	--	--	
20（0x14）	电位器校正	--	--	
21（0x15）	电位器校正	--	--	
22（0x16）	电位器校正	--	--	
23（0x17）	电位器校正	--	--	
24（0x18）	扭矩开关	读/写	0（0x00）	RAM
25（0x19）	LED开关	读/写	0（0x00）	
26（0x1A）	顺时针不灵敏区	读/写	2（0x02）	
27（0x1B）	逆时针不灵敏区	读/写	2（0x02）	
28（0x1C）	顺时针比例系数	读/写	32（0x20）	
29（0x1D）	逆时针比例系数	读/写	32（0x20）	
30（0x1E）	目标位置（L）	读/写	[Addr36]value	
31（0x1F）	目标位置（H）	读/写	[Addr37]value	

32（0x20）	运行速度（L）	读/写	0	
33（0x21）	运行速度（H）	读/写	0	
34（0x22）	加速度	读/写	32	
35（0x23）	减速度	读/写	32	
36（0x24）	当前位置（L）	读	?	
37（0x25）	当前位置（H）	读	?	
38（0x26）	当前速度（L）	读	?	
39（0x27）	当前速度（H）	读	?	
40（0x28）	当前负载	读	?	
41（0x29）	当前负载	读	?	
42（0x2A）	当前电压	读	?	
43（0x2B）	当前温度	读	?	
44（0x2C）	REG WRITE标志	读	0（0x00）	
45（0x2D）	--		0（0x00）	
46（0x2E）	运行中标志	读	0（0x00）	
47（0x2F）	锁标志	读/写	0（0x00）	
48（0x30）	最小PWM(L)	读/写	90（0x5A）	
49（0x31）	最小PWM(H)	读/写	00（0x00）	

若控制参数有“L”、“H”之分的命令，其范围为 0x00—0x3ff；参数只占一个字节的命令，其范围为 0x00—0xff。

保存在 RAM 里的参数掉电后不会保存，保存在 EEPROM 里的参数，掉电后可以保存。“--”表示不可修改参数。

详细描述如下：

0x03：

保存舵机的 ID 号。

0x04:

保存波特率计算参数。

波特率的计算公式： Speed(BPS) = 2000000/(Address4+1)。

Address4 默认为 1，表示的波特率为 1M，可按此公式把波特率修改为用户需要的其他波特率，但掉电后只有 1M，500K，250K，115200，57600，19200 会保存。其他的波特率会被恢复为 1M.

公式中 Address4 即为 0x04 地址保存的数据。波特率和相应的计算参数的对照如下表：

Address4	Hex	实际波特率	目标波特率	误差
1	0x01	1000000.0	1000000.0	0.000%
3	0x03	500000.0	500000.0	0.000%
7	0x07	250000.0	250000.0	0.000%
16	0x10	117647.1	115200.0	-2.124%
34	0x22	57142.9	57600.0	0.794%
103	0x67	19230.8	19200.0	-0.160%

0x05:

设置返回延迟时间，即当舵机收到一条需要应答的指令后，延迟多长时间应答可由您设置。时间范围：参数（0~255）*2US，若参数 250，即 500us 后应答；但参数为 0，表示以最短的时间应答，由于 CDS55XX 需要约 8us 的最小反应时间，所以实际最小应答时间为 8us。

0x06～0x09:

设置舵机可运行的角度范围，顺时针角度限制 =< 目标角度值≤<= 逆时针角度限制值。

注意，顺时针角度限制值必须小于逆时针角度限制值。若目标角度值超过范围，则等于限制值，且舵机状态位的角度超范围标志会置 1。

0x0B

最高工作温度，定为 80 度，用户修改无效。

0x0E～0x0F:

设置舵机的最大输出力矩。0X03FF 对应 CDS55XX 的最大输出能力。

0x10:

应答级别，设置舵机接收到数据后是否返回数据。

地址16	返回应答包
0	对所有的指令都不返回
1	只对读指令返回
2	对所有指令返回

0x11:

设置 LED 闪烁报警条件。

BIT	功能
BIT7	0
BIT6	如果设置为1，则发生指令错误时闪烁
BIT5	如果设置为1，则发生过载时闪烁
BIT4	如果设置为1，则发生校验和错误时闪烁
BIT3	如果设置为1，则发生指令超范围时闪烁
BIT2	如果设置为1，则发生过热时闪烁
BIT1	如果设置为1，则发生超过角度时闪烁
BIT0	如果设置为1，则发生超过电压范围时闪烁

以上若同时发生，遵行逻辑或的原则。

0x12:

设置卸载条件。

BIT	功能
BIT7	0
BIT6	如果设置为1，则发生指令错误时卸载
BIT5	--
BIT4	如果设置为1，则发生校验和错误时卸载
BIT3	如果设置为1，则发生指令超范围时卸载
BIT2	如果设置为1，则发生过热时卸载
BIT1	如果设置为1，则发生超过角度时卸载
BIT0	如果设置为1，则发生超过电压范围时卸载

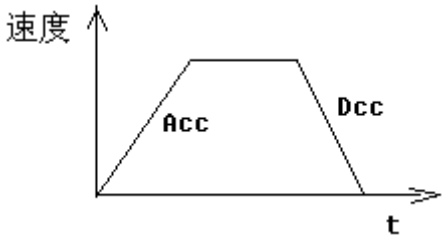
以上若同时发生，遵行逻辑或的原则。

BIT5 过载标志无效，当 CDS55XX 过载后，扭力会自动降低为不会烧毁的安全值，而不会完全卸载。

0x18:
力矩输出开关，“1”开，“0”关。
0x19:
LED 开关，“1”开，“0”关。
0x1A～0x1B:
位置闭环的死区大小。
0x1C～0x1D:
位置闭环的 P 参数，影响位置环的调节快慢。用户一般不需要修改此参数，控制运动平滑度的是 ACC，DCC。即加速度和减速度。由 0x22～0x23 设置。
0x1E～0x1F:
命令舵机运行至的位置 ,范围 0x0000—0x03ff,0x0000 对应 0 度 ,0x03ff 对应 300 度，偏差±2%。
0x20～0x21:
设置舵机运行至目标位置的速度。设置为 1023 时，对应与 CDS55xx 的最大速度 62RPM。

位置模式时最小速度为 1，最大速度 1023，速度 0 等价于 1023。

0x22～0x23:
舵机运行的加速度和减速度(ACC,Dcc).范围 0~255。



0x24～0x2E:
舵机的运行状态，如当前位置、速度等，只能读不能写。
0x2C：
若有 REG WRITE 指令等待执行 ,则显示为 1 ,当 REG WRITE 指令执行完毕后显示为 0。
0x2E：
若舵机正在运行中显示为 1，反之显示为 0。
0x2F：
锁功能位。若该位设置为 1，则只有 0x18～0x23 可以被修改，其他位置被锁住不能修改。一旦锁功能生效，必须掉电后该功能方才失效。
0x30～0x31:
PWM 占空比的最小输出值。

3.2 电机调速模式

CDS 系列机器人舵机可以切换为电机调速模式 ,可用于轮子、履带等周转的执行机构上。

把顺时针角度限制和逆时针角度限制(0x06～0x09)都设置为 0 ,再给一个速度(**0x20～0x21**) ，舵机就以电机调速模式转动起来。速度有大小和方向的控制方式，如下表所示：

BIT	11~15	10	9	8	7	6	5	4	3	2	1	0
VALUE	0	0/1	SPEED VALUE									

地址 **0x20～0x21：**BIT10 是方向位，为 0 逆时针转动，为 1 顺时针转动。BIT0～BIT9 为大小。

注意：电机模式时，加减速度相等，由 0x22 设置。

第四章 示例

- **例 1： 把 ID 号为 1 的舵机改为 0**

指令 = WRITE DATA；地址 = 0X03; 数据 = 0x00
指令包：FF FF 01 04 03 03 00 F4
应答包：FF FF 01 02 00 FC
状态：无错误

注意：除非同时将多个舵机设置成相同的 ID 地址，否则，舵机需要逐个设置 ID 号。指令包是指主控制器或调试器发送给 CDS55xx 的控制数据，应答包是指 CDS55xx 返回的信息。

- **例 2：把 0 号舵机运行的角度范围限制在 0～150°之间**

指令 = WRITE DATA；地址 = 0X08; 数据 = 0XFF, 0X01
指令包：FF FF 00 05 03 08 FF 01 EF
应答包：FF FF 00 02 00 FD
状态：无错误

- **例 3：把 0 号舵机工作温度的上限设置到 80 摄氏度**

指令 = WRITE DATA；地址 = 0X0B; 数据 = 0X50
指令包：FF FF 00 04 03 0B 50 9D
应答包：FF FF 00 02 00 FD
状态：无错误

- **例 4：把 0 号舵机允许的工作电压范围设为 6V～9V**

6V 用 60（0X3C）表示，9V 用 90（0X5A）表示
指令 = WRITE DATA；地址 = 0X0C; 数据 = 0X3C, 0X5A
指令包：FF FF 00 05 03 0C 3C 5A 55
应答包：FF FF 00 02 00 FD
状态：无错误

- **例 5：把 0 号舵机的输出力矩限制为最大值的一半**

输出最大时对应 0x03ff，所以输出一半对应 0x01ff
指令 = WRITE DATA；地址 = 0X0E; 数据 = 0XFF, 0X01
指令包：FF FF 00 05 03 0E FF 01 E9
应答包：FF FF 00 02 00 FD
状态：无错误

- **例 6：设置 0 号舵机对所有的输入都不返回数据**

指令 = WRITE DATA；地址 = 0X10; 数据 = 0X00
指令包：FF FF 00 04 03 10 00 E8
应答包：FF FF 00 02 00 FD
状态：无错误

- **例 7：让 0 号舵机卸载**

指令 = WRITE DATA；地址 = 0X18; 数据 = 0X00
指令包：FF FF 00 04 03 18 00 E0
应答包：FF FF 00 02 00 FD
状态：无错误

- **例 8：让 0 号舵机以中速运动至 150°的位置**

全速时对应 0x03ff ,则中速可设置为 0x01ff ,300°对应 0x03ff ,所以 150°对应 0x01ff
指令 = WRITE DATA；地址 = 0X1E; 数据 = 0x00, 0x02, 0x00, 0x02
指令包：FF FF 00 07 03 1E 00 02 00 02 D3
应答包：FF FF 00 02 00 FD
状态：无错误

- **例 9：让 2 号舵机运动至 0°位置,1 号舵机运行至 300°位置。**

注意：要求它们同时开始运行。

运用 REG_WRITE + ACTION 指令可以实现它们同时动作
ID=2；指令 = REG_WRITE；地址 = 0X1E; 数据 = 0x00, 0x00
ID=1；指令 = REG_WRITE；地址 = 0X1E; 数据 = 0xFF, 0x03
ID=0XFE; 指令 = ACTION

指令包：FF FF 02 05 04 1E 00 00 D6
应答包：FF FF 02 02 00 FB
指令包：FF FF 01 05 04 1E FF 03 D5
应答包：FF FF 01 02 00 FC
指令包：FF FF FE 02 05 FA
使用广播地址，无应答包
状态：无错误

- **例 10：设置 0 号舵机内存控制表里，只能对 0x18～0x23 地址段进行修改**

即在地址 0x2f 上写入 1
指令 = WRITE DATA；地址 = 0X2F; 数据 = 0x03
指令包：FF FF 00 04 03 2F 01 C8
应答包：FF FF 00 02 00 FD
状态：无错误

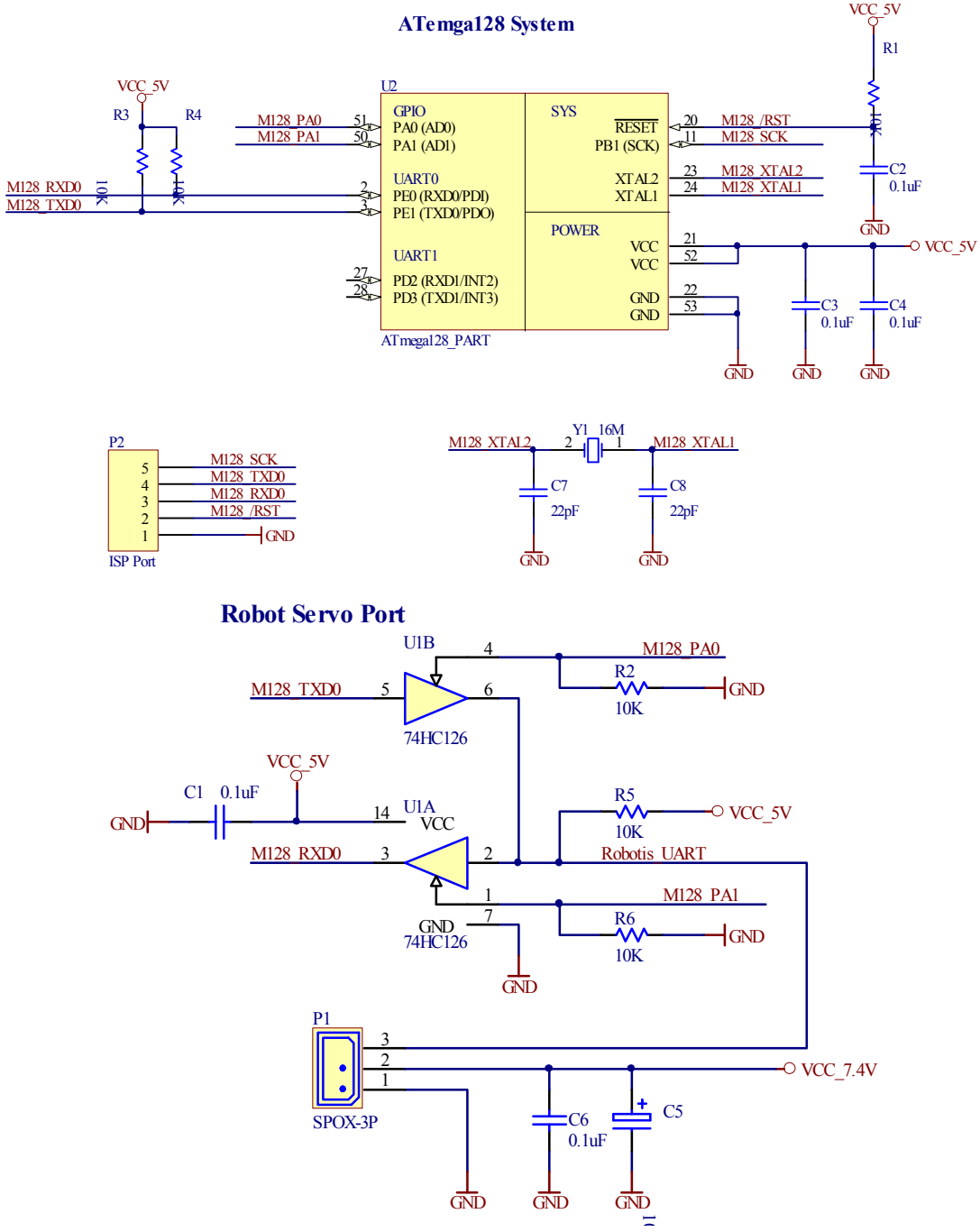
- **例 11： 将 0 号舵机位置闭环时运行的加速度设为 4，减速度设为 6**

加、减速度的最大值为 255
指令 = WRITE DATA；地址 = 0X30；数据 = 0X04, 0X06
指令包：FF FF 00 05 03 30 04 06 BD
应答包：FF FF 00 02 00 FD
状态：无错误

第五章 基于AVR单片机的舵机控制器开发

5.1 原理图

在实际使用 CDS55xx 舵机时，您可能需要自己开发控制器来控制舵机。下图是一个比较简单的舵机控制卡原理图，您可以参考此图设计自己的控制卡。



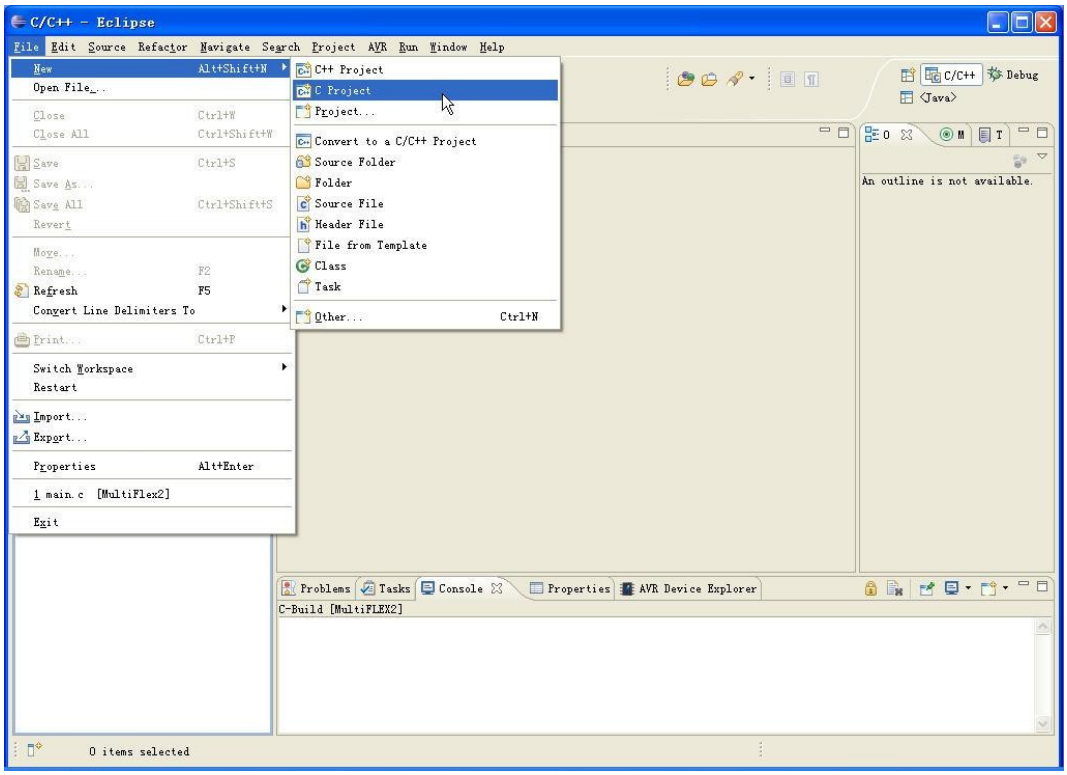
5.2 控制卡程序开发

。

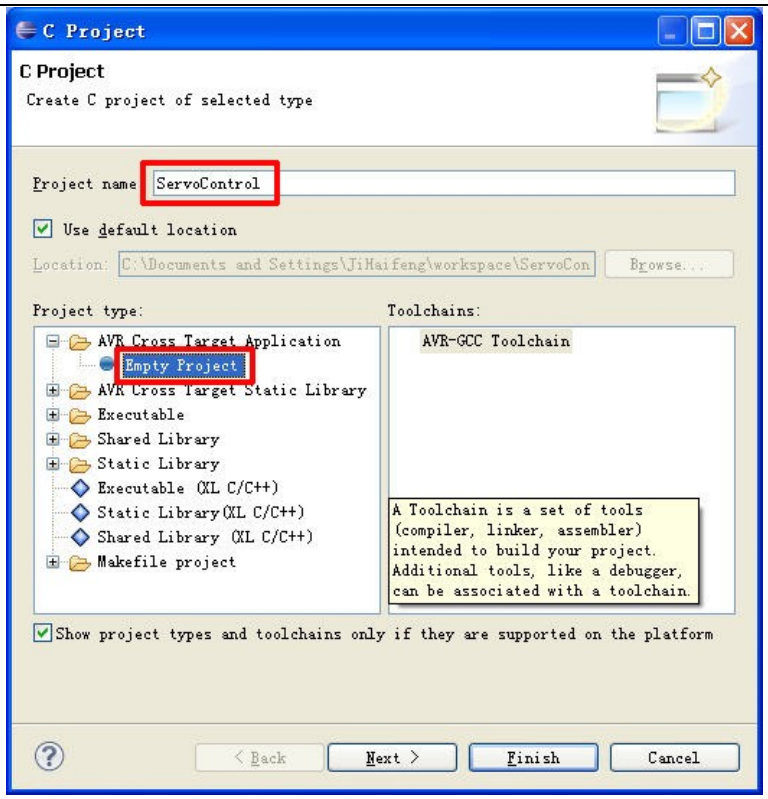
本文使用 Eclipse 作为开发环境，您如果不熟悉 Eclipse 开发 AVR 程序，请参考文档《EclipseForAVR 程序开发》。该程序最初的目标是让一个舵机在两个位置之间摆动，接下来通过添加代码，可以实现一个通过两个红外传感器来进行避障小车的控制逻辑。下面介绍开发过程。

5.21 创建工程

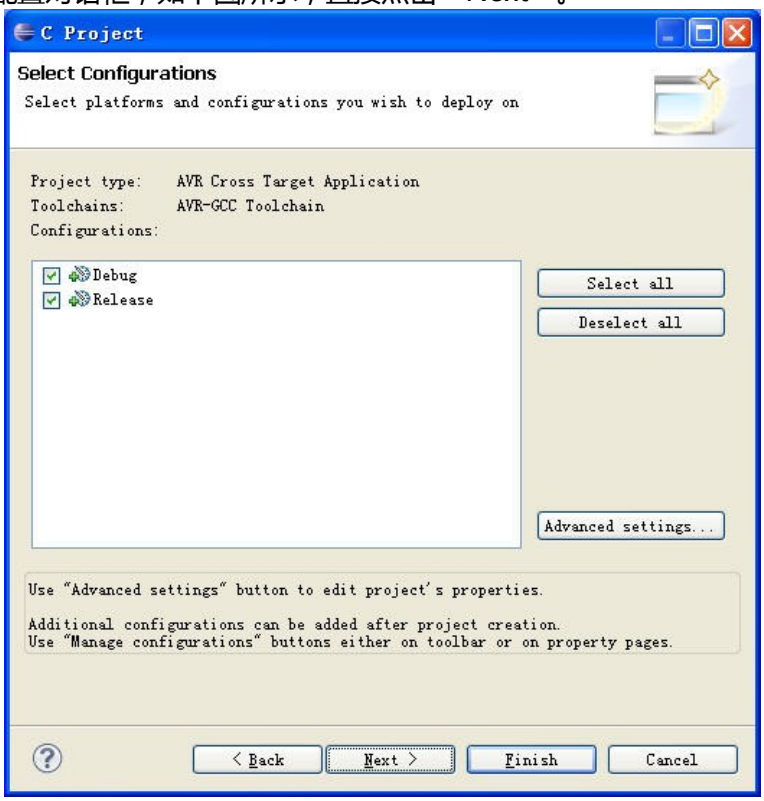
运行 Eclipse，选择“File”->”New”->”C Project”菜单项，新建工程，如下图所示。



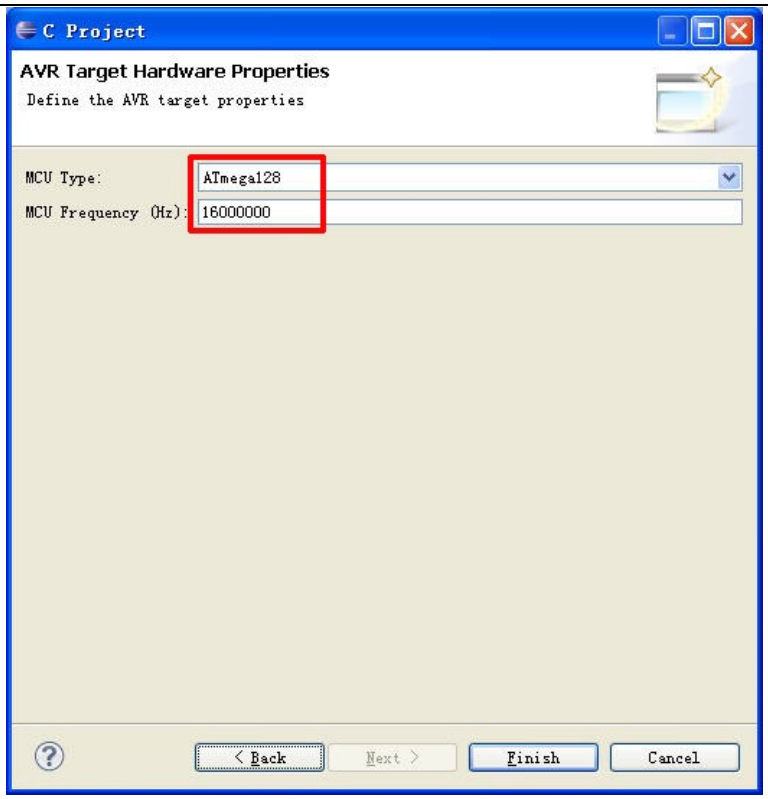
在弹出的对话框中，“Project Name”处输入“ServoControl”，在“Project Type”列表中展开“AVR Cross Target Application”项，选择“Empty Project”，如下图所示，点击“Next”。



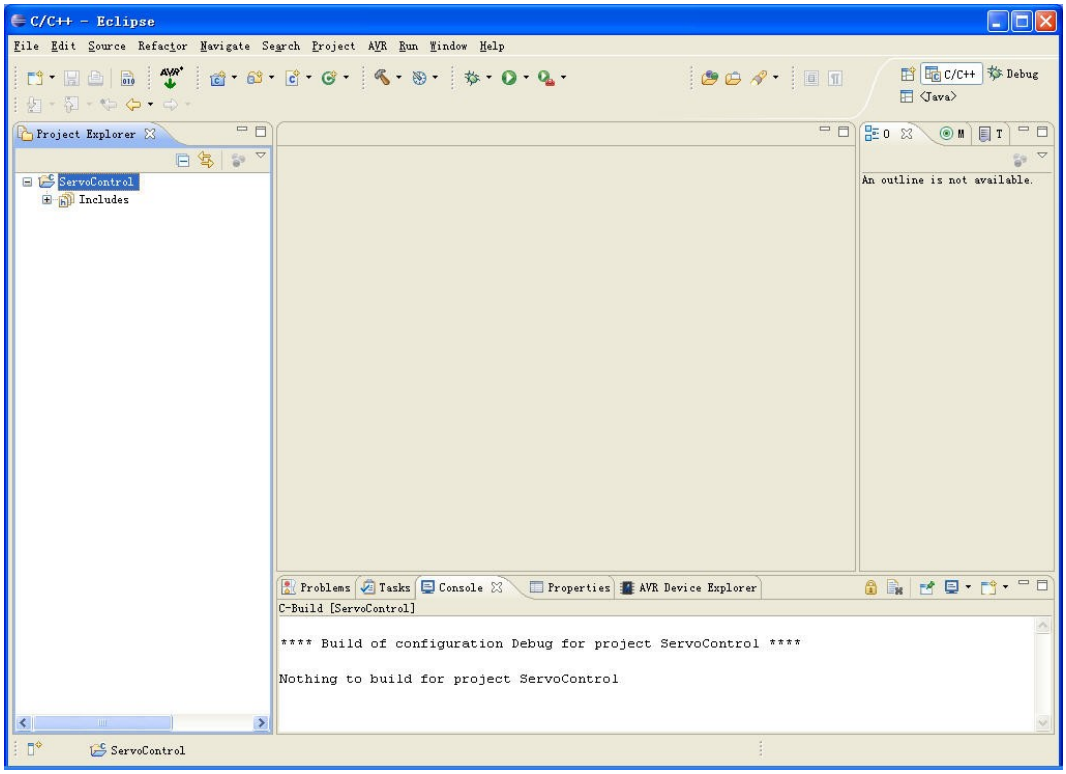
弹出选择配置对话框，如下图所示，直接点击“Next”。



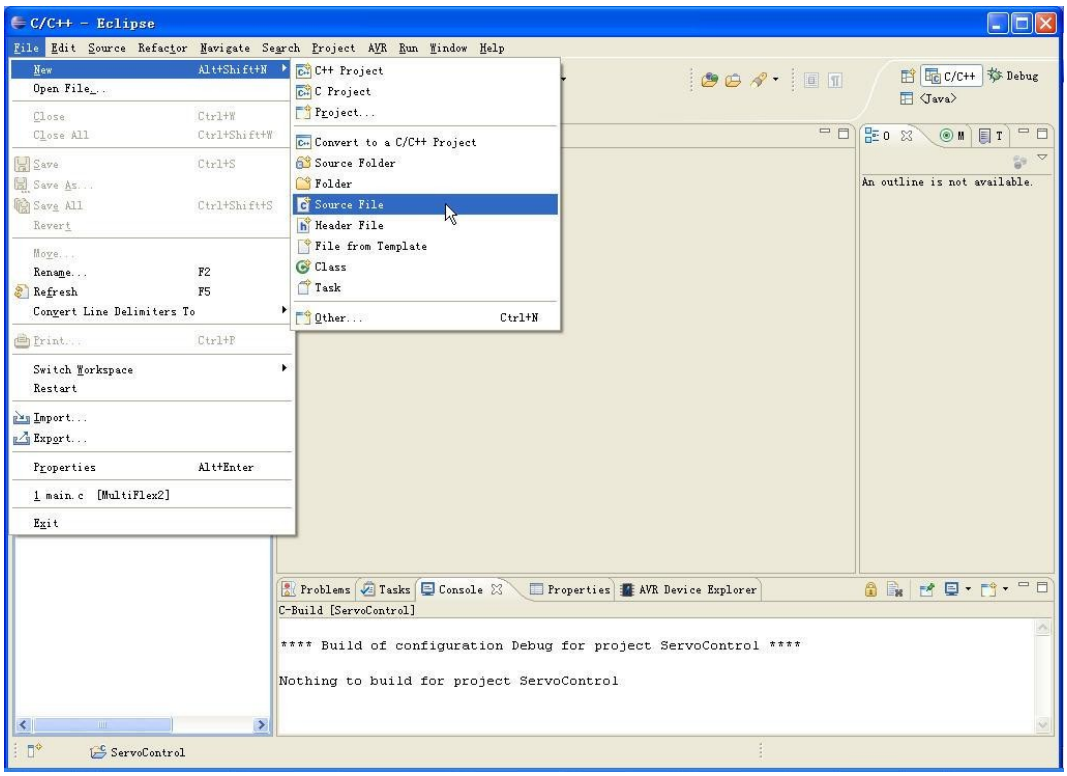
弹出目标设备属性设置对话框，在“MCU Type”下拉列表中选择“ATmega128”，“MCU Frequency”中输入 “16000000”，如下图所示。点击“Finish”完成工程创建。



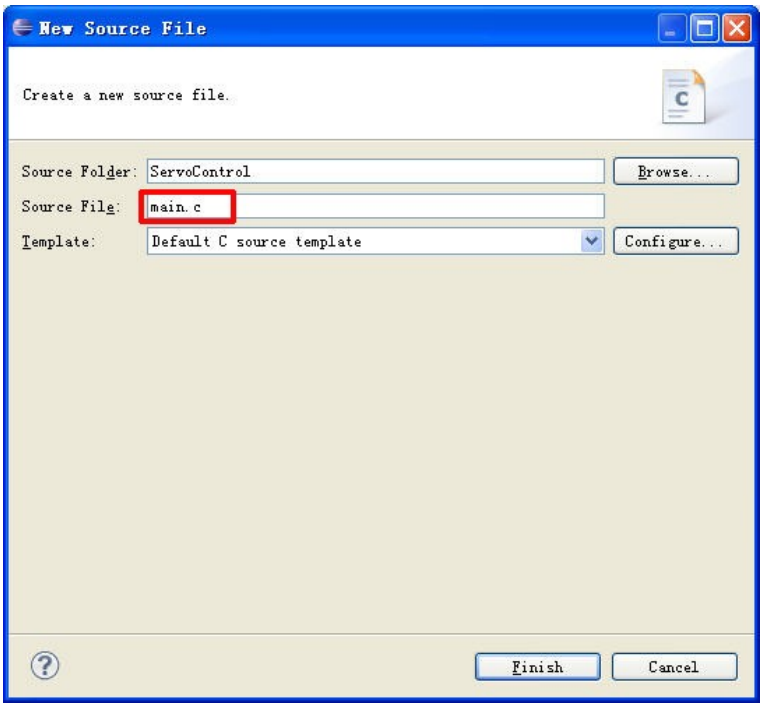
返回 Eclipse 主窗口，在左侧“Project Explorer”子窗口中，出现“ServoControl”工程，如下图所示：



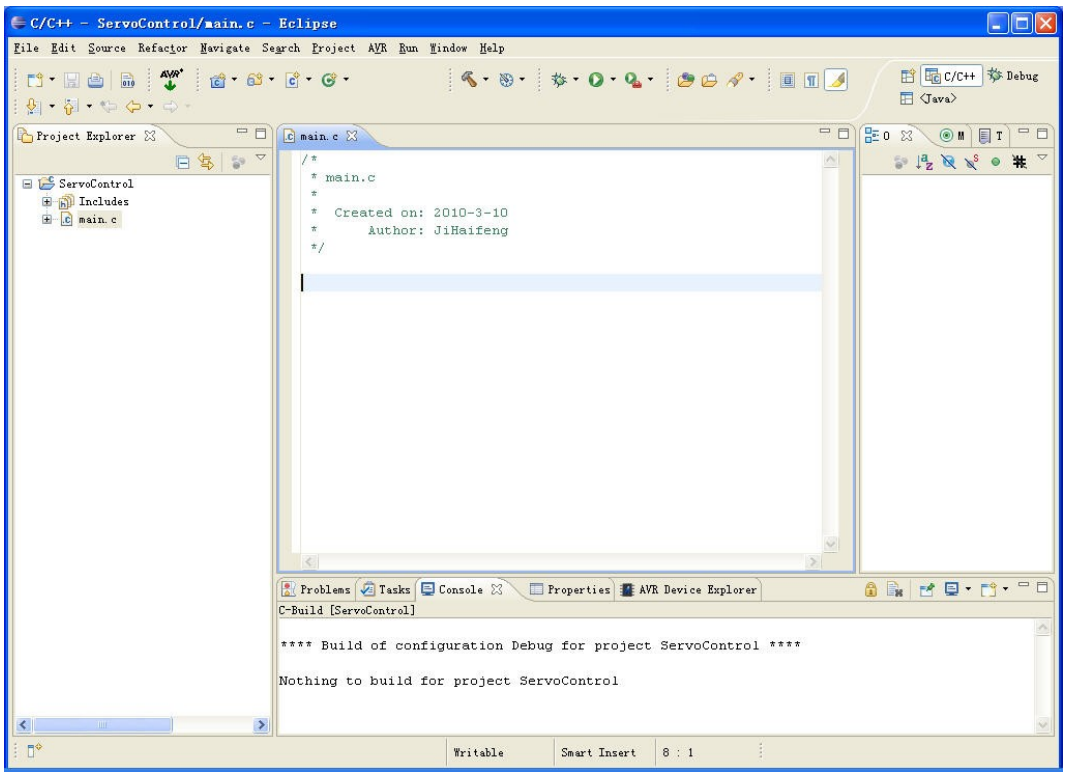
选择“File”->”New”->”Source File”菜单项，新建源代码文件，如下图所示：



弹出设置对话框，在“Source File”处输入“main.c”，如下图所示。点击“Finish”完成新建。



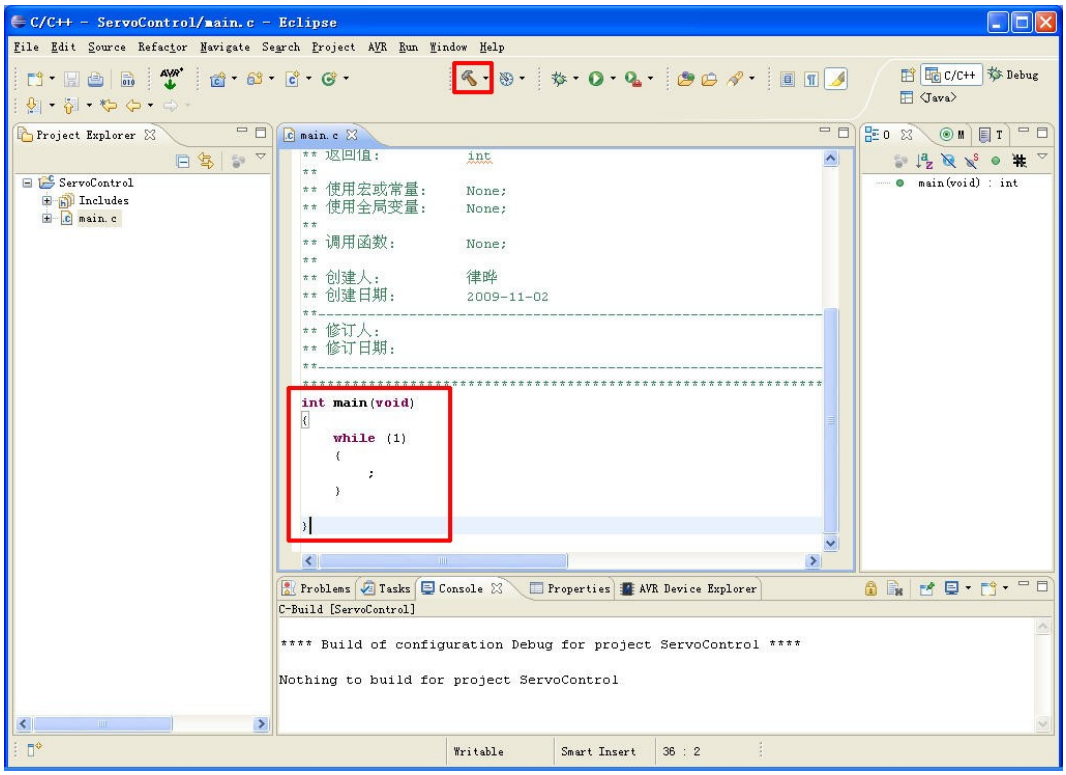
返回 Eclipse 主窗口，main.c 文件自动添加到“ServoControl”工程并处于可编辑状态，如下图所示：



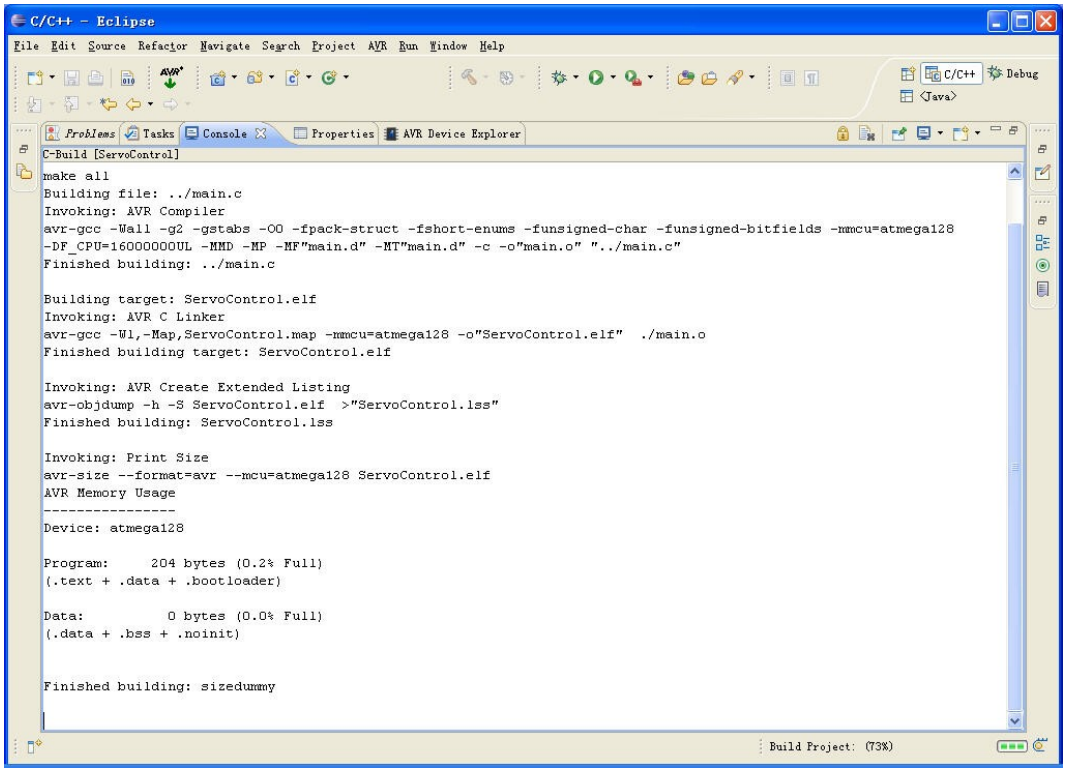
在 main.c 窗口中加入主函数，如下图所示。然后点击  开始编译程序。主函数代码如下：

```
/*********************************Copyright*************************************
**                          Beijing UPTECH Robotics Co.Ltd.
**                               yanfabu@126.com
**                            http://robot.up-tech.com
**
**-------------------------------------------------File Info-------------------------------------------------
** 文件名称(File Name):                  main.c
** 最后修订日期(Date Last Modified):     2010-3-25
** 最后版本(Last Version):               1.0
** 描述(Description):
**
**-----------------------------------------------------------------------------------------------------------
** 创建人(Author):                       robot
** 创建日期(Date Created):               2010-3-25
** 版本(Version):                        1.0
** 描述(Description):
**
***********************************************************************************/
int main(void)
{
```

```
    while (1)
    {
        ;
    }
}
```



编译信息出现在“Console”子窗口，可以双击“Console”标签最大化窗口查看，如下图所示。可以看出编译正确。



再次双击“Console”标签，回到之前状态。

5.22 添加初始化函数

控制卡和舵机时通过 UART0 来通讯的，所以程序运行后首先需要进行 UART0 的初始化。另外，实现避障小车控制逻辑时，需要连接两个 IO 量传感器来判断障碍，所以也需要进行 IO 的初始化。

这里添加 InitGpio 函数来实现 IO 初始化，代码如下：

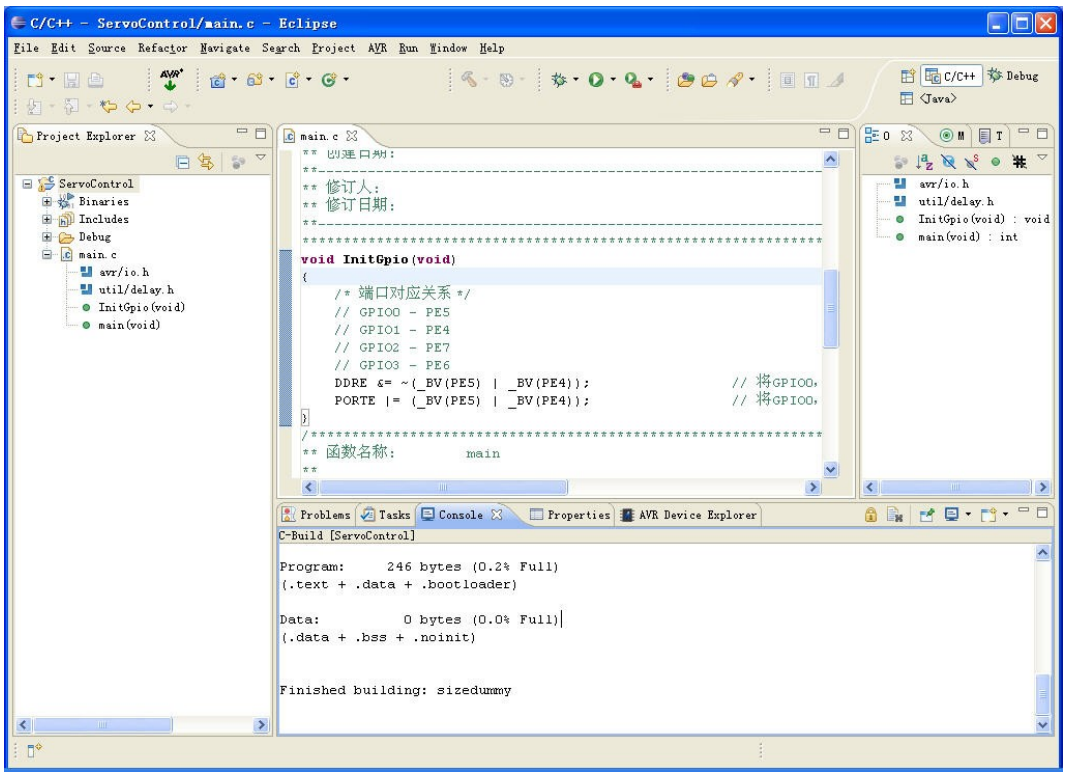
```
/******************************** Function Info *************************
**  函数名称(Function Name):              InitGpio
**
**  函数描述(Description):                通用 IO 初始化函数
                                          (Init general digital I/O port)
**
**
**  输入变量(Input Variable):                 void
**  返回值(Return Value):                     void
**
**  使用宏或常量(Macro or constant used):None
**  使用全局变量(Global variable used):   None
**
**  调用函数(Other function called):      None
**
**  创建人(Author):                           robot
**  创建日期(Date created):               2010-3-25
**
```

```
** 修订人(Revised):
** 修订日期(Date Revised):
*********************************************************************************/
void InitGpio(void)
{
    /*端口映射表（Port map） */
    // GPIO0 - PE5
    // GPIO1 - PE4
    // GPIO2 - PE7
    // GPIO3 - PE6

    // 将 GPIO0，GPIO1 的方向设置为输入
    // (Set general digital I/O port 0 and port 1 to input mode)
    DDRE &= ~(_BV(PE5) | _BV(PE4));

    // 将 GPIO0，GPIO1 的状态设置为上拉
    // (Set Set general digital I/O port 0 and pull-up state)
    PORTE |= (_BV(PE5) | _BV(PE4));
}
```

将上面代码复制粘贴到 main.c 文件中、main 函数之前，并在 main.c 文件开始出加入需要包含的头文件：

```
#include <avr/io.h>
#include <util/delay.h>
```

点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。

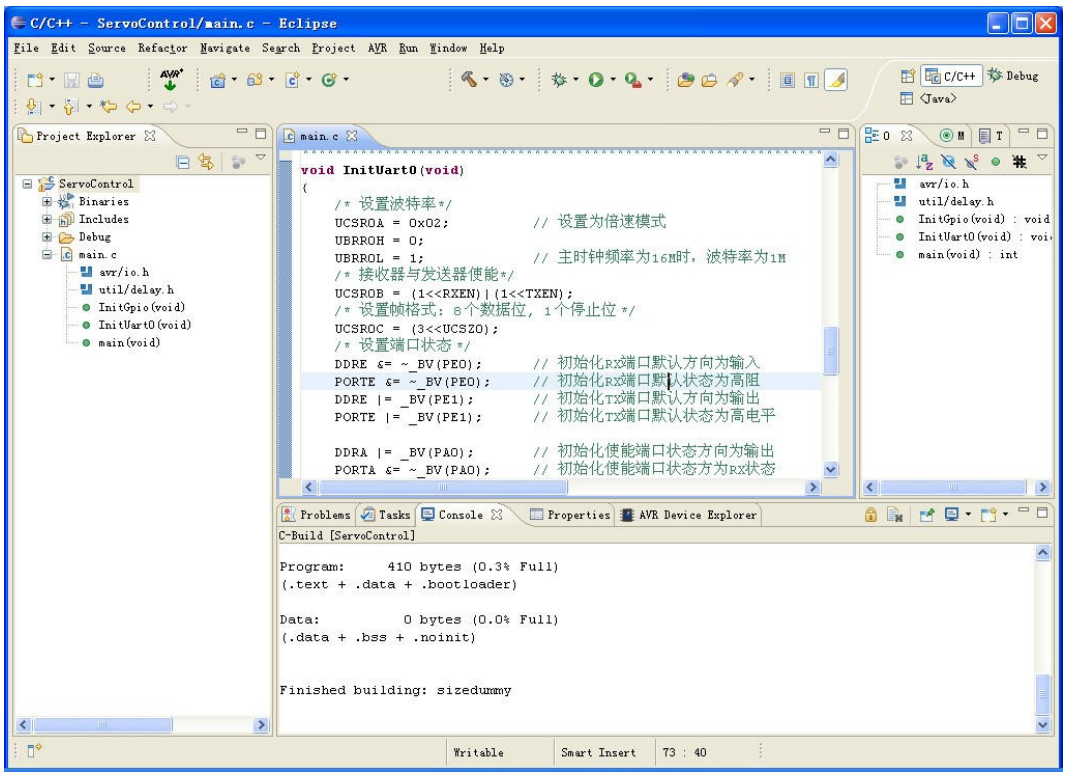


在 main.c 中添加函数 InitUart0 来实现 UART0 的初始化，代码如下：

```
/*********************************Function Info **********************
**  函数名称(Function Name):                    InitUart0
**
**  函数描述(Description):                      uart0 初始化函数(Init UART0)
**
**
**  输入变量(Input Variable):                   void
**  返回值(Return Value):                       void
**
**  使用宏或常量(Macro or constant used):None
**  使用全局变量(Global variable used):     None
**
**  调用函数(Other function called):            None
**
**  创建人(Author):                             robot
**  创建日期(Date created):                     2010-3-25
**
**  修订人(Revised):
**  修订日期(Date Revised):
*************************************************************************/
void InitUart0(void)
{
    /*设置波特率（Set baud rate）*/
```

```
    UCSR0A = 0x02;                  // 设置为倍速模式（Set to double velocity mode）
    UBRR0H = 0;

    // 主时钟频率为 16M 时，波特率为 1M
    // (Set main clock to 16M, baud rate to 1M)
    UBRR0L = 1;

    // 接收器与发送器使能
    // (Enable UART0's receiver and transmitter)
    UCSR0B = (1<<RXEN)|(1<<TXEN);

    // 设置帧格式: 8 个数据位, 1 个停止位
// (Set data frame:8 data bits,1 stop bit)
    UCSR0C = (3<<UCSZ0);
    /* 设置端口状态(Set port state) */
    DDRE &= ~_BV(PE0);      // 初始化 RX 端口默认方向为输入
                            // (Set default direction of RX to input mode)
    PORTE &= ~_BV(PE0);     // 初始化 RX 端口默认状态为高阻
                            // (Set default state of RX to tri-state)
    DDRE |= _BV(PE1); // 初始化 TX 端口默认方向为输出
                            // (Set default direction of TX to output mode)
    PORTE |= _BV(PE1);      // 初始化 TX 端口默认状态为高电平
                            // (Set default state of RX to high state)
    DDRA |= _BV(PA0); // 初始化使能端口状态方向为输出
                            // (Enable direction of Port A as output)
    PORTA &= ~_BV(PA0);     // 初始化使能端口状态为 RX 状态
                            // (Set Port A state as same to RX)
    DDRA |= _BV(PA1); // 初始化使能端口状态方向为输出
                            // (Set Port A state as same to RX)
    PORTA |= _BV(PA1);      // 初始化使能端口状态方为 RX 状态
                            // (Set Port A state as same to RX)
}
```

将上面代码复制粘贴到 main.c 文件，点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。



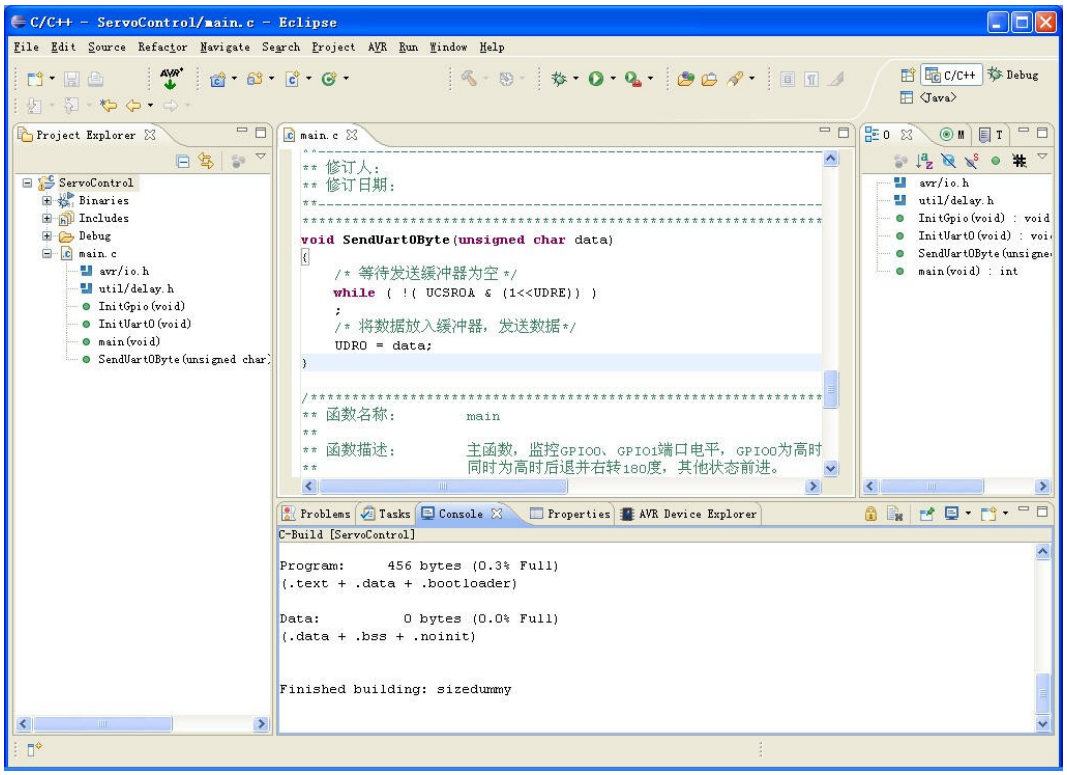
5.23 添加舵机控制函数

实现舵机的控制需要通过 UART0 发送控制命令给舵机。控制命令一般是一帧数据，实际发送时，UART0 是按字节发送的。为了方便调用，这里首先在 main.c 中加入一个函数 SendUart0Byte，该函数实现通过 UART0 发送一个字节数据的功能。代码如下：

```
/*********************************Function Info **********************
** 函数名称(Function Name):                SendUart0Byte
**
** 函数描述(Description):                  串口发送数据函数(Send one byte through UART0)
**
**
** 输入变量(Input Variable):               unsigned char data
** 返回值(Return Value):                   void
**
** 使用宏或常量(Macro or constant used):   None
** 使用全局变量(Global variable used):     None
**
** 调用函数(Other function called):        None
**
** 创建人(Author):                         robot
** 创建日期(Date created):                 2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
```

```
*********************************************************************/
void SendUart0Byte(unsigned char data)
{
    // 等待发送缓冲器为空
    // waiting for finishing sending all datas in the Transmit Buffer
    while ( !( UCSR0A & (1<<UDRE)) );
    /* 将数据放入缓冲器，发送数据*/
    UDR0 = data;
}
```

将上面代码复制粘贴到 main.c 文件，点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。



CDS55xx 系列舵机可以设置为两种模式：舵机模式和电机模式。舵机出厂时默认是舵机模式。舵机的模式设置通过设置舵机的最大最小限位参数来实现。这里添加一个函数 SetServoLimit 来设置舵机模式，当程序中需要改变模式实际操作时可以通过调用该函数来实现，代码如下：

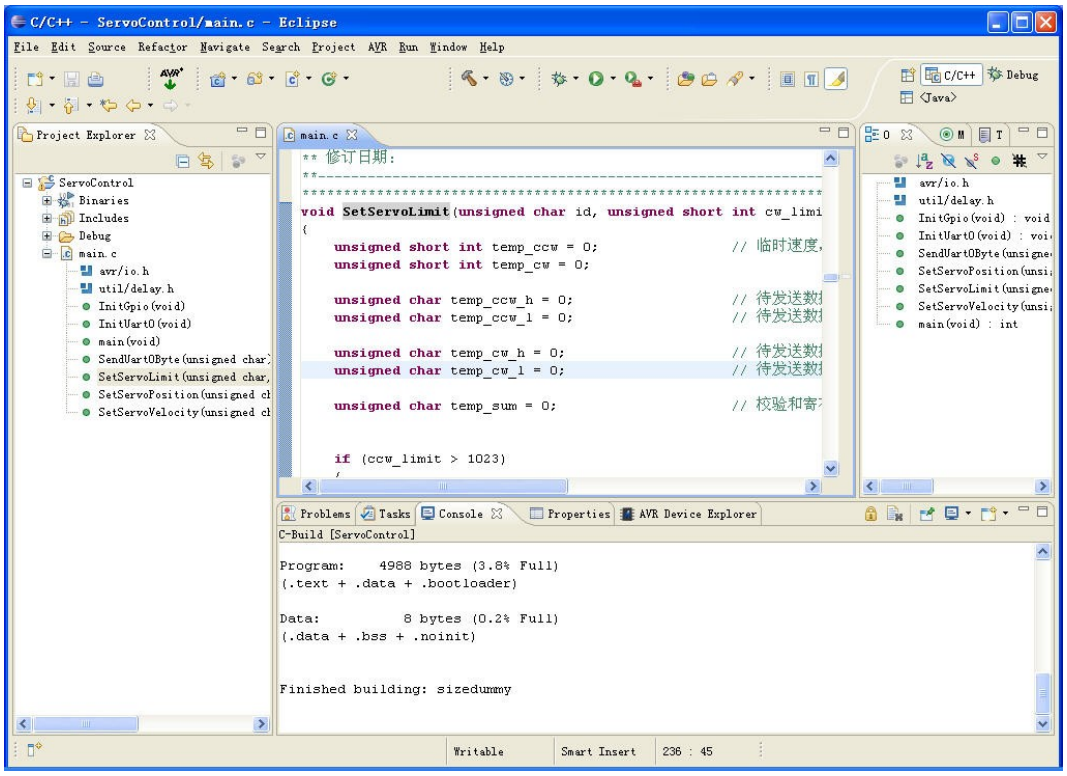
```
/**********************************Function Info ***********************
** 函数名称(Function Name):                SetServoLimit
**
** 函数描述(Description):          设置舵机位置限制，同时也用于切换舵机工作模式
                                   (Set servo position limitation and change servo mode)
**
**
```

```
** 输入变量(Input Variable):                unsigned char id; unsigned short int
cw_limit;signed short int ccw_limit;
** 返回值(Return Value):                    void
**
** 使用宏或常量(Macro or constant used):    None
** 使用全局变量(Global variable used):      None
**
** 调用函数(Other function called):         None
**
** 创建人(Author):                          robot
** 创建日期(Date created):                  2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
********************************************************************************/
void SetServoLimit(unsigned char id, unsigned short int cw_limit, unsigned short int ccw_limit)
{
    unsigned short int temp_ccw = 0;        // 临时速度，用于进行方向判别
                                            // (temp velocity to judge the direction)
    unsigned short int temp_cw = 0;

    unsigned char temp_ccw_h = 0;           // 待发送数据 h 位
                                            // (h bits to be sended)
    unsigned char temp_ccw_l = 0;           // 待发送数据 l 位
                                            // (l bits to be sended)
    unsigned char temp_cw_h = 0;
    unsigned char temp_cw_l = 0;

    unsigned char temp_sum = 0;             // 校验和寄存变量
                                            // (temp variable to save checksum)
    if (ccw_limit > 1023)
    {
        temp_ccw = 1023;                    // 限制速度值在可用范围内
                                            // (limit the velocity to 0-1023)
    }
    else
    {
        temp_ccw = ccw_limit;
    }

    if (cw_limit > 1023)
    {
        temp_cw = 1023;
    }
```

```
else
{
    temp_cw = cw_limit;
}

temp_ccw_h = (unsigned char)(temp_ccw >> 8);
temp_ccw_l = (unsigned char)temp_ccw;          // 将 16bit 数据拆为 2 个 8bit 数据
                                               // (split 16 bits to 2 bytes)
temp_cw_h = (unsigned char)(temp_cw >> 8);
temp_cw_l = (unsigned char)temp_cw;            // 将 16bit 数据拆为 2 个 8bit 数据
                                               // (split 16 bits to 2 bytes)
PORTA &= ~_BV(PA1);
PORTA |= _BV(PA0);                             // 使总线处于主机发送状态
                                               // (Set the bus to host transmit state)
UCSR0A |= (1<<TXC0);                           // 清除 UART0 写完成标志
                                               // (Clear written flag of UART0)
SendUart0Byte(0xFF);                           // 发送启动符号 0xFF
                                               // (Send the start byte 0xff)
SendUart0Byte(0xFF);                           // 发送启动符号 0xFF
                                               // (Send the start byte 0xff)
SendUart0Byte(id);                     // 发送 id
                                       // (Send the servo's ID)
SendUart0Byte(7);                      // 发送数据长度为参数长度+2，参数长度为 3
                                       // (Send the length of frame)
SendUart0Byte(0x03);                   // 命令数据为"WRITE DATA"
                                       // (Send command "WRITE DATA")
SendUart0Byte(0x06);                   // 舵机控制寄存器首地址
                                       // (Send the start address of control rigister)
SendUart0Byte(temp_cw_l);              // 发送顺时针位置限制低位
                                       // (Send the low byte of clockwise position limit)
SendUart0Byte(temp_cw_h);              // 发送顺时针位置限制高位
                                       //(Send the high byte of clockwise position limit)
SendUart0Byte(temp_ccw_l);     // 发送逆时针位置限制低位
                               // (Send the low byte of counterclockwise position limit)

SendUart0Byte(temp_ccw_h);     // 发送逆时针位置限制高位
                               // (Send the low byte of counterclockwise position limit)
temp_sum = id + 7 + 0x03 + 0x06 + temp_cw_l + temp_cw_h + temp_ccw_l + temp_ccw_h;
temp_sum = ~temp_sum;          // 计算校验和
                               // (Calculate the checksum)
SendUart0Byte(temp_sum);       // 发送校验和
                               // (Send checksum)

while ( !( UCSR0A & (1<<TXC0)) )  // 等待发送完成
```

```
    {                                   // (Waiting for finishing sending)
        ;
    }

    PORTA |= _BV(PA1);
    PORTA &= ~_BV(PA0);                 // 使总线处于主机接收状态
                                        // (Set the UART bus to host receiving state)
    _delay_ms(2);       //送完成后，总线会被从机占用，反馈应答数据，所以进行延时
                        // (The bus will be overrode by slave after finishing sending
                        // to receive the answer, so here delays 2 ms.)
}
```

将上面代码复制粘贴到 main.c 文件，点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。



接下来在 main.c 中添加舵机位置控制函数 SetServoPosition ,该函数实现通过 UART0 发送舵机位置控制命令的功能。代码如下：

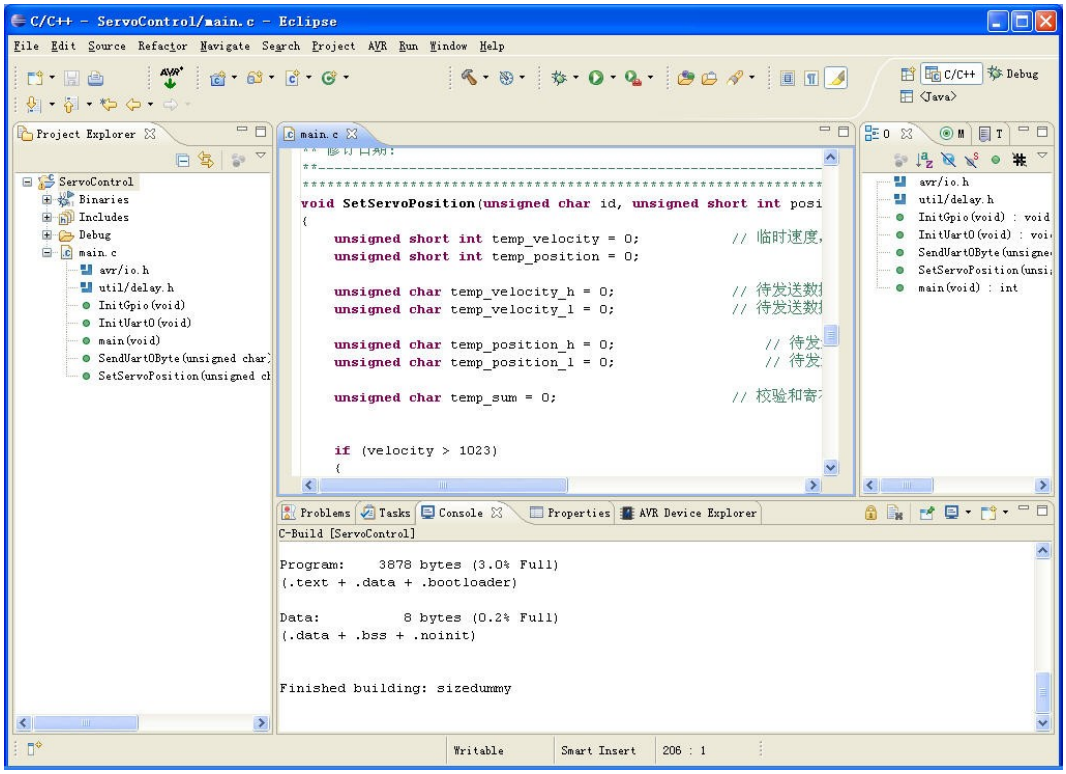
```
/**********************************Function Info **********************
**  函数名称(Function Name):                SetServoPosition
**
**  函数描述(Description):                  设置舵机位置(Set servo postion)
**
**
**  输入变量(Input Variable):               unsigned char id; unsigned short int position;
```

```
signed short int velocity;
** 返回值(Return Value):                    void
**
** 使用宏或常量(Macro or constant used):     None
** 使用全局变量(Global variable used):       None
**
** 调用函数(Other function called):          None
**
** 创建人(Author):                           robot
** 创建日期(Date created):                   2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
********************************************************************************/
void SetServoPosition(unsigned char id, unsigned short int position, unsigned short int velocity)
{
    unsigned short int temp_velocity = 0;       // 临时速度，用于进行方向判别
                                                // (temp velocity to judge the direction)
    unsigned short int temp_position = 0;

    unsigned char temp_velocity_h = 0;          // 待发送数据 h 位
                                                // (h bits to be sended)
    unsigned char temp_velocity_l = 0;          // 待发送数据 l 位
                                                // (l bits to be sended)
    unsigned char temp_position_h = 0;
    unsigned char temp_position_l = 0;

    unsigned char temp_sum = 0;                 // 校验和寄存变量
                                                // (temp variable to save checksum)

    if (velocity > 1023)
    {
        temp_velocity = 1023;                   // 限制速度值在可用范围内
                                                // (limit the velocity to 0-1023)
    }
    else
    {
        temp_velocity = velocity;
    }

    if (position > 1023)
    {
        temp_position = 1023;
```

```
    }
    else
    {
        temp_position = position;
    }

    temp_velocity_h = (unsigned char)(temp_velocity >> 8);
    // 将 16bit 数据拆为 2 个 8bit 数据
    // (split 16 bits to 2 bytes)
    temp_velocity_l = (unsigned char)temp_velocity;

    temp_position_h = (unsigned char)(temp_position >> 8);
    // 将 16bit 数据拆为 2 个 8bit 数据
    // (split 16 bits to 2 bytes)
    temp_position_l = (unsigned char)temp_position;

    PORTA &= ~_BV(PA1);
    PORTA |= _BV(PA0);                  // 使总线处于主机发送状态
                                        // (Set the bus to host transmit state)
    UCSR0A |= (1<<TXC0);                // 清除 UART0 写完成标志
                                        // (Clear written flag of UART0)
    SendUart0Byte(0xFF);                // 发送启动符号 0xFF
                                        // (Send the start byte 0xff)
    SendUart0Byte(0xFF);

    SendUart0Byte(id);                  // 发送 id
                                        // (Send the servo's ID)
    SendUart0Byte(7);                   // 发送数据长度为参数长度+2，参数长度为 3
                                        // (Send the length of frame)
    SendUart0Byte(0x03);                // 命令数据为"WRITE DATA"
                                        // (Send command "WRITE DATA")
    SendUart0Byte(0x1E);                // 舵机控制寄存器首地址
                                        // (Send the start address of control register)
    SendUart0Byte(temp_position_l);     // 发送速度数据低位
                                        // (Send the low byte of velocity)
    SendUart0Byte(temp_position_h);     // 发送速度数据高位
                                        // (Send the high byte of velocity)
    SendUart0Byte(temp_velocity_l);     //发送位置低字节
                                        // (Send the low byte of position)
    SendUart0Byte(temp_velocity_h);     // 发送位置高字节
                                        // (Send the high byte of position)
    temp_sum = id + 7 + 0x03 + 0x1E + temp_position_l + temp_position_h + temp_velocity_l +
temp_velocity_h;
    temp_sum = ~temp_sum;               // 计算校验和
```

```
                                        // (calculate the checksum)
    SendUart0Byte(temp_sum);            // 发送校验和 (Send the checksum)

    while ( !( UCSR0A & (1<<TXC0)) )    // 等待发送完成
    {                                       // (Waiting for finishing sending)
        ;
    }

    PORTA |= _BV(PA1);
    PORTA &= ~_BV(PA0);                 // 使总线处于主机接收状态
                                        // (Set the UART bus to host receiving state)
    _delay_ms(2); // 发送完成后，总线会被从机占用，反馈应答数据，所以进行延时
                  // (The bus will be overrode by slave after finishing sending
                  // to receive the answer, so here delays 2 ms.)

}
```

将上面代码复制粘贴到 main.c 文件，点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。



对于电机模式的舵机，控制时只需要设置舵机速度。这里添加函数 SetServoVelocity，实现通过 UART0 发送速度控制命令的功能。代码如下：

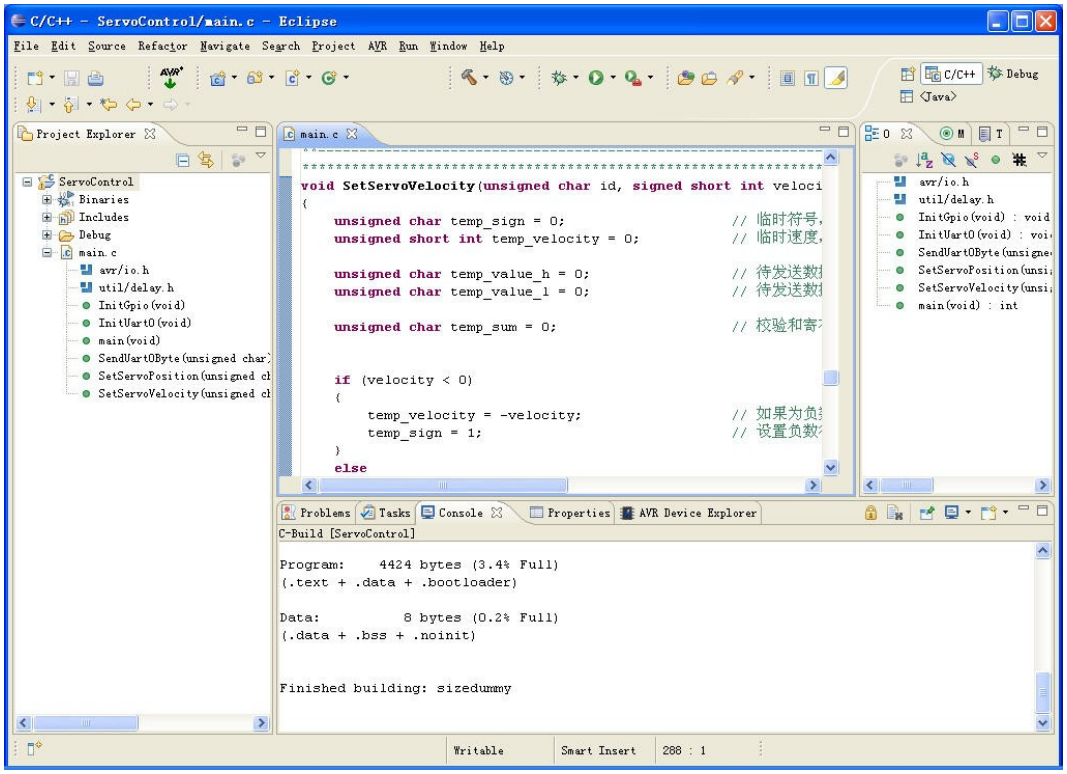
```
/**********************************Function Info **********************
** 函数名称(Function Name):                SetServoVelocity
```

```
**
** 函数描述(Description):              设置舵机速度(Set servo velocity)
**
**
** 输入变量(Input Variable):           unsigned char id; signed short int velocity;
** 返回值(Return Value):               void
**
** 使用宏或常量(Macro or constant used):  None
** 使用全局变量(Global variable used):    None
**
** 调用函数(Other function called):       None
**
** 创建人(Author):                     robot
** 创建日期(Date created):             2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
********************************************************************/
void SetServoVelocity(unsigned char id, signed short int velocity)
{
    unsigned char temp_sign = 0;                // 临时符号，用于进行方向判别
                                                // (temp variable to judge the direction)
    unsigned short int temp_velocity = 0;       // 临时速度，用于进行方向判别
                                                // (temp velocity to judge the direction)
    unsigned char temp_value_h = 0;             // 待发送数据 h 位
                                                // (h bits to send)
    unsigned char temp_value_l = 0;             // 待发送数据 l 位
                                                // (l bits to send)
    unsigned char temp_sum = 0;                 // 校验和寄存变量
                                                // (temp variable to save checksum)

    if (velocity < 0)
    {
        temp_velocity = -velocity;              // 如果为负数，则取绝对值
                                                // (if negative, get the absolute value)
        temp_sign = 1;                          // 设置负数符号标志
                                                // (Set negative flag)
    }
    else
    {
        temp_velocity = velocity;
        temp_sign = 0;                          // 设置正数符号标志
                                                // (Set positive flag)
    }
```

```
if (temp_velocity > 1023)
{
      temp_velocity = 1023;                    // 限制速度值在可用范围内
                                               // (Limit the velocity to 0-1023)
}
// 设置 bit10 为方向位，这时 temp_velocity 就是待发送的速度数据了
// (Set bit0 as direction bit, then temp_velocity is the data to be sended)
temp_velocity |= (temp_sign << 10);

temp_value_h = (unsigned char)(temp_velocity >> 8);
// 将 16bit 数据拆为 2 个 8bit 数据
// (Split the 16 bits to 2 bytes)
temp_value_l = (unsigned char)temp_velocity;

PORTA &= ~_BV(PA1);
PORTA |= _BV(PA0);                   // 使总线处于主机发送状态
                                     // (Set the bus to host transmit state)
UCSR0A |= (1<<TXC0);                 // 清除 UART0 写完成标志
                                     // (Clear written flag of UART0)
SendUart0Byte(0xFF);                 // 发送启动符号 0xFF
                                     // (Send the start byte 0xFF)
SendUart0Byte(0xFF);                 // 发送启动符号 0xFF
                                     // (Send the start byte 0xFF)

SendUart0Byte(id);                   // 发送 id
                                     // (Send the servo's ID)
SendUart0Byte(5);                    // 发送数据长度为参数长度+2，参数长度为 3
                                     // (Send the length of frame)
SendUart0Byte(0x03);                 // 命令数据为"WRITE DATA"
                                     // (Send command "WRITE DATA")
SendUart0Byte(0x20);                 // 舵机控制寄存器首地址
                                     // (Send the start address of control register)
SendUart0Byte(temp_value_l);         // 发送速度数据低位
                                     // (Send the low byte of velocity)
SendUart0Byte(temp_value_h);         // 发送速度数据高位
                                     // (Send the high byte of velocity)
temp_sum = id + 5 + 0x03 + 0x20 + temp_value_l + temp_value_h;
temp_sum = ~temp_sum;                // 计算校验和
                                     // (Calculate the checksum)
SendUart0Byte(temp_sum);             // 发送校验和
                                     // (Send the checksum)
while ( !( UCSR0A & (1<<TXC0)) )     // 等待发送完成
{                                    // (Waiting for finishing sending)
```

```
        ;
    }

    PORTA |= _BV(PA1);
    PORTA &= ~_BV(PA0);          // 使总线处于主机接收状态
                                 // (Set the UART bus to host receiving state)
    _delay_ms(2); // 发送完成后，总线会被从机占用，反馈应答数据，所以进行延时
                  //the bus will be overrode by slave after finishing sending
                  // to receive the answer, so here delays 2 ms.
}
```

将上面代码复制粘贴到 main.c 文件，点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。



5.24 实现控制逻辑

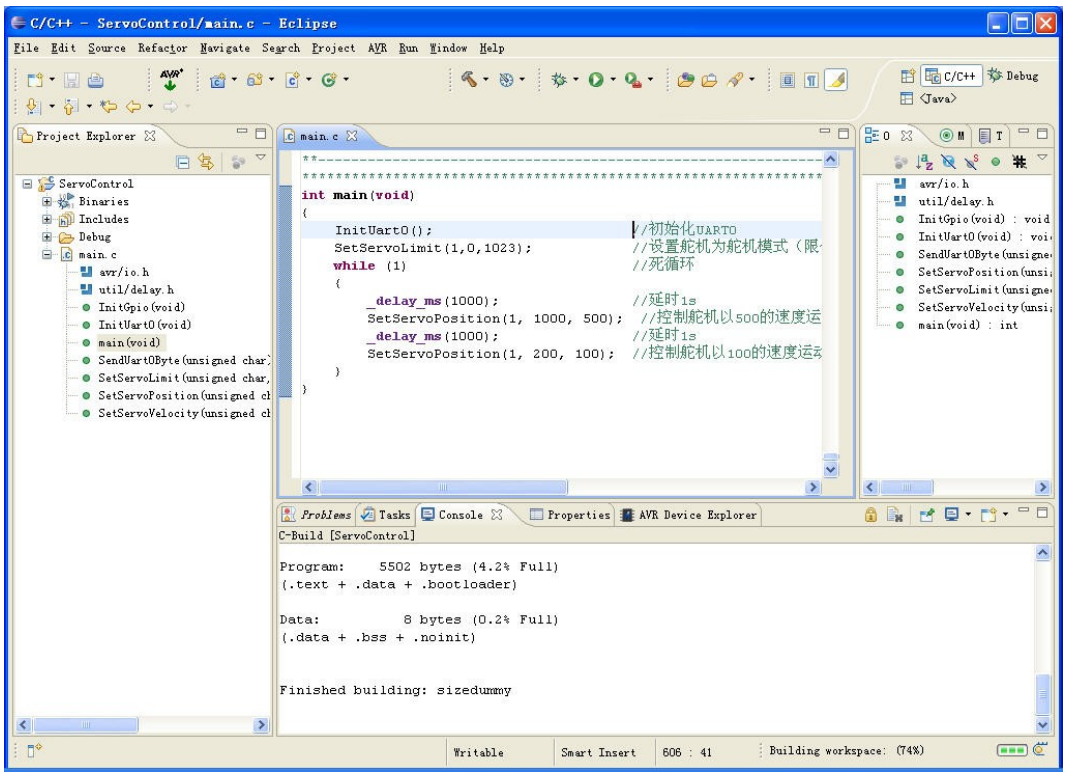
- **舵机摆动逻辑的实现**

有了控制舵机位置和速度的函数，就可以来完成控制一个舵机在两个位置之间摆动的逻辑。这里假设要控制的舵机 ID 是 1，修改 main 函数后代码如下：

```
/**********************************Function Info **********************
** 函数名称(Function Name):                    main
**
** 函数描述(Description):                 控制舵机摆动(Wave the Servo of ID 1)
```

```
**
**
** 输入变量(Input Variable):                    void
** 返回值(Return Value):                        int
**
** 使用宏或常量(Macro or constant used):        None
** 使用全局变量(Global variable used):          None
**
** 调用函数(Other function called):             None
**
** 创建人(Author):                              robot
** 创建日期(Date created):                      2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
*****************************************************************************
int main(void)
{
    InitUart0();                          //初始化 UART0
                                          // (Init UART0)
    SetServoLimit(1,0,1023);              //设置舵机为舵机模式（限位 0-1023）
                                          // (Set servo of ID 1 to servo mode)
    while (1)                             //死循环 (Infinite loop)
    {
        _delay_ms(1000);                  //延时 1s
                                          //(Delay 1 s)
        SetServoPosition(1, 1000, 500);   //控制舵机以 500 的速度运动到 1000 的位置
                                          // (Control the servo of ID 1 to position 1000
                                          //with the velocity of 500)
        _delay_ms(1000);                  //延时 1s
                                          //(Delay 1 s)
        SetServoPosition(1, 200, 100);    //控制舵机以 100 的速度运动到 200 的位置
                                          // (Control the servo of ID 1 to position 200
                                          //with thevelocity of 100)
    }
}
```

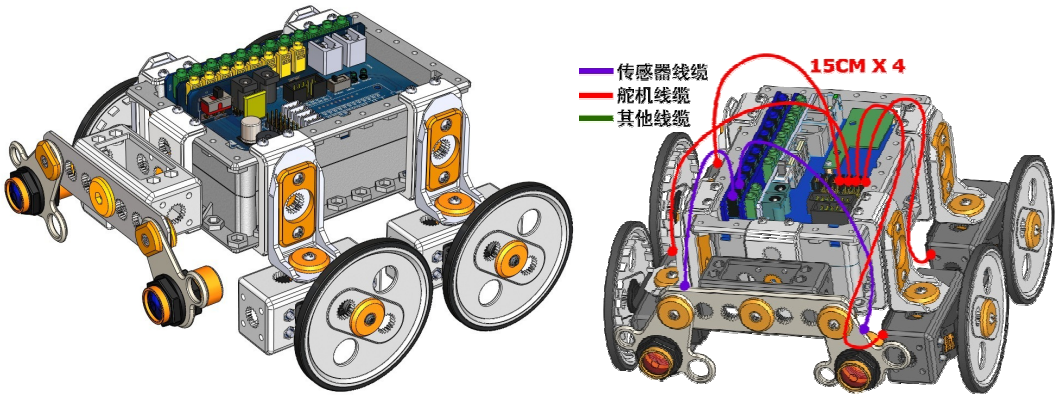
将上面代码复制粘贴到 main.c 文件，点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。正确编译后，即可用 ISP 或者 IAP 的方式下载程序到控制卡中。详细方法请参考《EclipseForAVR 程序开发》。



如果要实现避障小车的控制逻辑，应该怎么实现呢？

- **避障小车逻辑的实现**

为了调试程序，用户可以参考下图搭建一个避障小车。图中小车右侧的舵机 ID 为 1（前）和 2（后），小车左侧的舵机 ID 为 3（前）和 4（后）。右侧的红外传感器连接到控制器的 IO0，左侧的红外传感器连接到 IO1。



要实现避障，就需要查询传感器来判断是否遇到障碍，所以需要添加代码来实现传感器数据的读取。

根据避障的逻辑，小车碰到障碍时需要左转，或者右转，或者倒退，没有障碍时需要前进。为了使主逻辑清楚，可以添加左转、右转、前进、倒退四个函数以方便调用。

读取传感器数据的函数 GetGpio 代码如下：

```
/**********************************Function Info **********************
**  函数名称(Function Name):                    GetGpio
**
**  函数描述(Description):                  读取 GPIO 值(Get general digital I/O port value)
**
```

```
**
** 输入变量(Input Variable):                unsigned char* val
** 返回值(Return Value):                    void
**
** 使用宏或常量(Macro or constant used):    None
** 使用全局变量(Global variable used):      None
**
** 调用函数(Other function called):         None
**
** 创建人(Author):                          robot
** 创建日期(Date created):                  2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
********************************************************************************/
void GetGpio(unsigned char* val)
{
    unsigned char temp_input_value = 0;

    temp_input_value = 0;

    if (PINE & _BV(PE5))                // 读取 PINE 指定位(Get value of PE5)
    {
        temp_input_value |= _BV(0); // _BV 是一个用于移位的宏，_BV(0)表示(1 << 0)
                                    // (_BV is a macro to shift, _BV(0) means (1<<0)
    }
    else
    {
        temp_input_value &= ~_BV(0);
    }

    if (PINE & _BV(PE4))                // 读取 PINE 指定位(Get value of PE5)
    {
        temp_input_value |= _BV(1);
    }
    else
    {
        temp_input_value &= ~_BV(1);
    }

    *val = temp_input_value;
}
```

左转函数 TernLeft 的代码如下：

```
/*********************************Function Info **********************
```

```
** 函数名称(Function Name):                    TurnLeft
**
** 函数描述(Description):                  左转( Turnleft)
**
**
** 输入变量(Input Variable):                  unsigned short int time,
** 返回值(Return Value):                      void
**
** 使用宏或常量(Macro or constant used):      None
** 使用全局变量(Global variable used):        None
**
** 调用函数(Other function called):           None
**
** 创建人(Author):                            robot
** 创建日期(Date created):                    2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
********************************************************************/
void TurnLeft(unsigned short int time)
{
    SetServoVelocity(1, 500);          //控制 ID 为 1 的舵机以 500 的速度正转
                                       // (Control the servo of ID 1 to rotate
                                       //with the velocity of 500)
    SetServoVelocity(2, 500);          //控制 ID 为 2 的舵机以 500 的速度正转
                                       // (Control the servo of ID 2 to rotate
                                       //with the velocity of 500)
    SetServoVelocity(3, 500);          //控制 ID 为 3 的舵机以 500 的速度正转
                                       // (Control the servo of ID 3 to rotate
                                       //with the velocity of 500)
    SetServoVelocity(4, 500);          //控制 ID 为 4 的舵机以 500 的速度正转
                                       // (Control the servo of ID 4 to rotate
                                       //with the velocity of 500)
    _delay_ms(time);                   //延时(Delay time ms)
}
```

右转函数 TurnRight 的代码如下：

```
/*********************************Function Info **********************
** 函数名称(Function Name):                    TurnRight
**
** 函数描述(Description):                  右转(TurnRight)
**
**
** 输入变量(Input Variable):                  unsigned short int time,
** 返回值(Return Value):                      void
```

```
**
** 使用宏或常量(Macro or constant used):      None
** 使用全局变量(Global variable used):        None
**
** 调用函数(Other function called):           None
**
** 创建人(Author):                            robot
** 创建日期(Date created):                    2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
*********************************************************************/
void TurnRight(unsigned short int time)
{
    SetServoVelocity(1, -500);          //控制 ID 为 1 的舵机以 500 的速度反转
                                        // (Control the servo of ID 1 to rotate
                                        //with the velocity of -500)
    SetServoVelocity(2, -500);          //控制 ID 为 2 的舵机以 500 的速度反转
                                        // (Control the servo of ID 2 to rotate
                                        //with the velocity of -500)
    SetServoVelocity(3, -500);          //控制 ID 为 3 的舵机以 500 的速度反转
                                        // (Control the servo of ID 3 to rotate
                                        //with the velocity of -500)
    SetServoVelocity(4, -500);          //控制 ID 为 4 的舵机以 500 的速度反转
                                        // (Control the servo of ID 4 to rotate
                                        //with the velocity of -500)
    _delay_ms(time);                    //延时(Delay time ms)
}
```

前进函数 Forward 的代码如下：

```
/*********************************Function Info **********************
** 函数名称(Function Name):                   Forward
**
** 函数描述(Description):                     前进(Forward)
**
**
** 输入变量(Input Variable):                  unsigned short int time,
** 返回值(Return Value):                      void
**
** 使用宏或常量(Macro or constant used):      None
** 使用全局变量(Global variable used):        None
**
** 调用函数(Other function called):           None
**
** 创建人(Author):                            robot
```

```
** 创建日期(Date created):                    2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
*********************************************************************************/
void Forward(void)
{
    SetServoVelocity(1, 500);          //控制 ID 为 1 的舵机以 500 的速度正转
                                       // (Control the servo of ID 1 to rotate
                                       //with the velocity of 500)
    SetServoVelocity(2, 500);          //控制 ID 为 2 的舵机以 500 的速度正转
                                       // (Control the servo of ID 2 to rotate
                                       //with the velocity of 500)
    SetServoVelocity(3, -500);         //控制 ID 为 3 的舵机以 500 的速度反转
                                       // (Control the servo of ID 3 to rotate
                                       //with the velocity of -500)
    SetServoVelocity(4, -500);         //控制 ID 为 4 的舵机以 500 的速度反转
                                       // (Control the servo of ID 4 to rotate
                                       //with the velocity of -500)
}
```

倒退函数 Back 的代码如下：

```
/*********************************Function Info ************************
** 函数名称(Function Name):                  Back
**
** 函数描述(Description):                  后退(Back)
**
**
** 输入变量(Input Variable):                 unsigned short int time,
** 返回值(Return Value):                     void
**
** 使用宏或常量(Macro or constant used):     None
** 使用全局变量(Global variable used):       None
**
** 调用函数(Other function called):          None
**
** 创建人(Author):                           robot
** 创建日期(Date created):                   2010-3-25
**
** 修订人(Revised):
** 修订日期(Date Revised):
*********************************************************************************/
void Back(void)
{
    SetServoVelocity(1, -500);         //控制 ID 为 1 的舵机以 500 的速度反转
```

```
                                    // (Control the servo of ID 1 to rotate
                                    //with the velocity of -500)
    SetServoVelocity(2, -500);      //控制 ID 为 2 的舵机以 500 的速度反转
                                    // (Control the servo of ID 2 to rotate
                                    //with the velocity of -500)
    SetServoVelocity(3, 500);       //控制 ID 为 3 的舵机以 500 的速度正转
                                    // (Control the servo of ID 3 to rotate
                                    //with the velocity of 500)
    SetServoVelocity(4, 500);       //控制 ID 为 4 的舵机以 500 的速度正转
                                    // (Control the servo of ID 4 to rotate
                                    //with the velocity of 500)
}
```

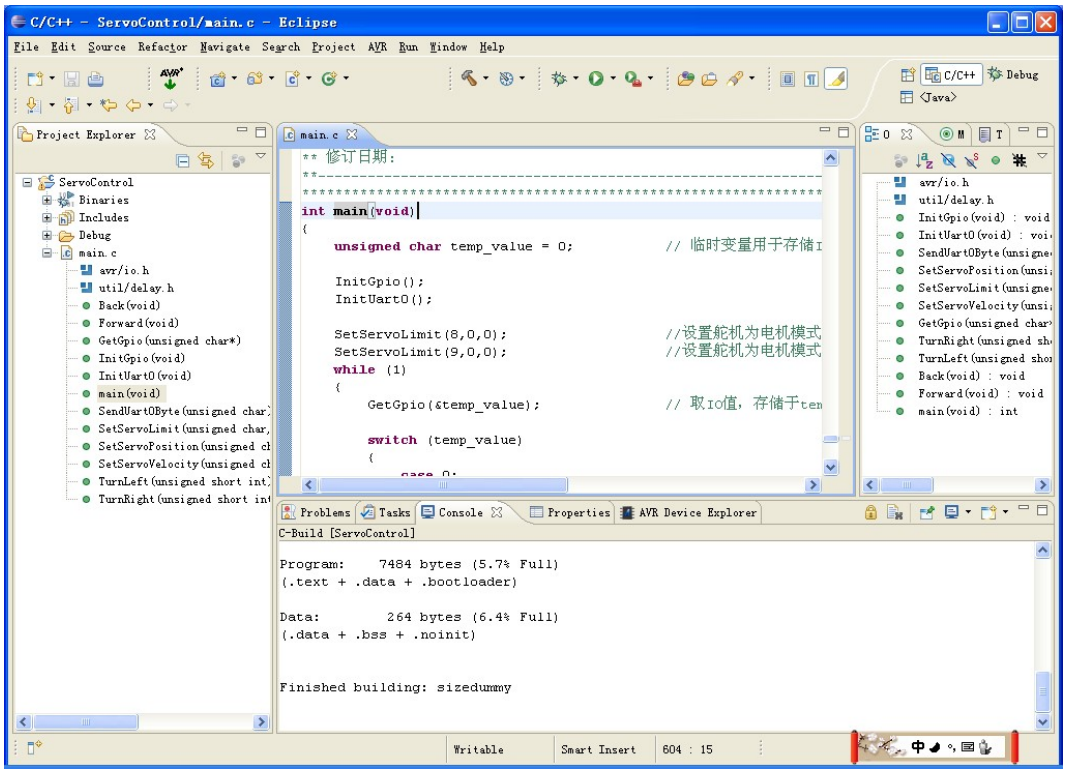
将 GetGpio，TurnLeft，TurnRight，Forward 和 Back 函数添加到 main.c 中 main 函数前面。修改 main 函数后代码如下：

```
/*********************************Function Info ***********************
**  函数名称(Function Name):                main
**
**  函数描述(Description):                  小车避障( Logic to control the robot escape
from hit the obstacle)
**
**
**  输入变量(Input Variable):               void
**  返回值(Return Value):                   int
**
**  使用宏或常量(Macro or constant used):   None
**  使用全局变量(Global variable used):     None
**
**  调用函数(Other function called):        None
**
**  创建人(Author):                         robot
**  创建日期(Date created):                 2010-3-25
**
**  修订人(Revised):
**  修订日期(Date Revised):
*****************************************************************************/
int main(void)
{
    unsigned char temp_value = 0;       //  临时变量用于存储 IO 数据
                                        // (temp variable to save I/O data)
    InitGpio();
    InitUart0();
    SetServoLimit(1,0,0);               //设置 1 号舵机为电机模式
                                        // (Set the servo of ID 1 to motor mode)
    SetServoLimit(2,0,0);               //设置 2 号舵机为电机模式
```

```
                                    // (Set the servo of ID 2 to motor mode)
SetServoLimit(3,0,0);               //设置 3 号舵机为电机模式
                                    // (Set the servo of ID 3 to motor mode)
SetServoLimit(4,0,0);               //设置 4 号舵机为电机模式
                                    // (Set the servo of ID 4 to motor mode)

while (1)
{
    GetGpio(&temp_value);           // 取 IO 值，存储于 temp_value 的 bit0 和 bit1 中
                                    // (Get the I/O value, save in the
                                    //bit0 and bit1 of temp_value )
    switch (temp_value)
    {
        case 0:
        {
            Back();                 // 后退  (Back)
            _delay_ms(2000);        // 后退 2s (Delay 2 s)
            TurnRight(2000);        // 转弯 2s (Turn right and delay 2 s)
            break;
        }
        case 1:
        {
            Back();
            _delay_ms(1000);
            TurnRight (1000);       // 转弯 1s (Turn right and delay 1 s)
            break;
        }
        case 2:
        {
            Back();
            _delay_ms(1000);
            TurnLeft(1000);
            break;
        }

        default:
        {
            Forward();                  // 其他情况前进
                                        // (Go forward in other conditions)
            _delay_ms(200);
            break;
        }
    }
}
```

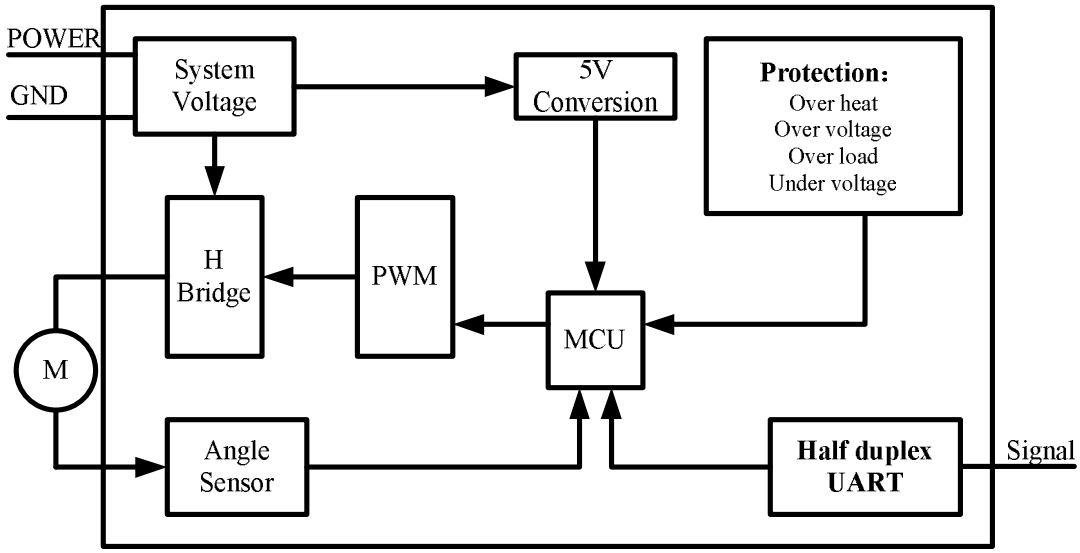
```
}
```

将上面代码复制粘贴到 main.c 文件，点击  保存 main.c，然后点击  重新编译程序，在“Console“子窗口可以看到编译结果，如下图所示。正确编译后，即可用 ISP 或者 IAP 的方式下载程序到控制卡中。详细方法请参考《EclipseForAVR 程序开发》。



附录

A. CDS55xx 电气原理框图



B. CDS55xx 系列舵机之间的区别

型号	CDS5500	CDS5516	
启动时间（秒）	2	0.6	
工作电压(V)	6.6~10	6.0~16	
LED	不可见	可见	