make a figure

Robotics and Electronics

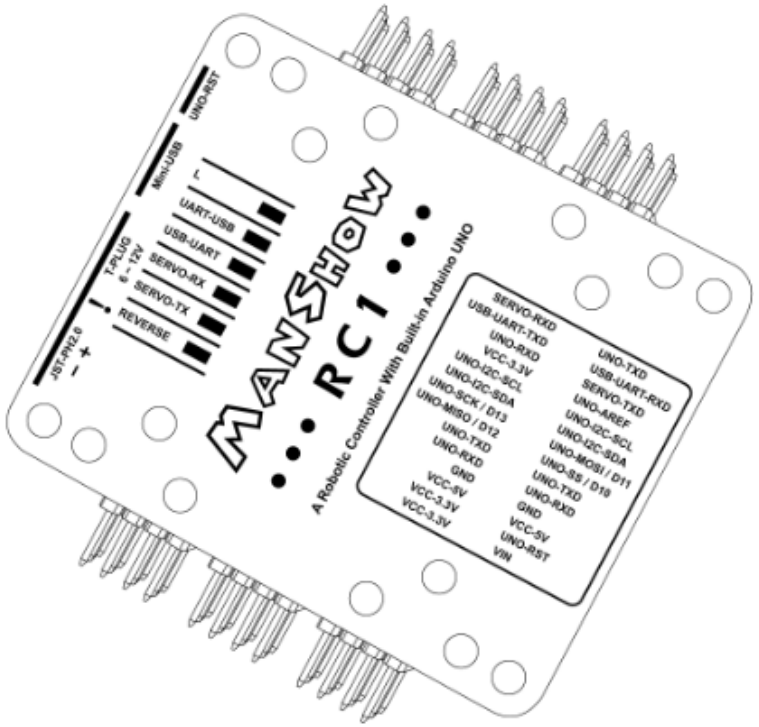
ManShow – RC1

简介

ManShow-RC1 是一款结构高度集成，专为人形、仿生及机械手等机器人量身定制的控制器（当然亦可用于其它机器人系统），集成 Arduino UNO R3 及 24 通道的 SoftServo 舵机控制系统。丰富的 Arduino 资源、标准化的传感器接口及灵活高效稳定的 PWM 类型舵机控制，极大程度地加速了机器人原型系统的开发。

规格

电压范围　：DC6 ~ 12V (电源输入：Mini-T 插头或 JST-PH2.0-2P)
尺寸　　：58.2mm * 62.2mm * 23.5mm(长*宽*高)
(注：电路板的尺寸及安装孔遵循 **Gicren Robotics and Electronics** 的规范，但 **ManShow-RC1** 属于 **Gicren** 产品系中的衍生物，没有完全遵循其标准产品的规范。）



内部通讯

Arduino UNO R3 与 24 通道的 **SoftServo** 舵机控制系统间可通过 I2C(硬件固定)或 UART(可选择)两种接口进行通讯，这两种通讯接口共享同一用户寄存器区。其中 I2C 接口的通讯速度为 100KHz；UART 接口的波特率范围为 1200 ~ 115200bps，并支持波特率自适应与非自适应两种模式。通讯协议请参考：
GI2C_Vxx 及 **G485_Vxx**。

特点

– – – 关于舵机控制 – – –

- 所有的 PWM 信号同时输出，实际分辨率可达 1μs。
- 极低的 PWM 信号耦合性，如同所有的 PWM 信号均由单独的硬件电路输出。
- 集成软启动功能，众所周知，PWM 类型的舵机其初始位置往往是未知的。传统的启动模式：将使舵机以全速运行到目标位置，在很多机器人应用中是非常忌讳的。软启动模式：在初始位置未知的情况下可将舵机缓速运行到目标位置，但在短时间内需要较大的电流(当需要控制多个舵机时，可采用分时启动的方式缓解)。
- PWM 信号周期、PWM 有效宽度最小值和 PWM 有效宽度最大值均可被灵活设置，兼容所有 PWM 类型的舵机。
- 基于细分算法，最大细分值可达 250。每个通道的位置和速度均可单独控制，极为利于机器人动态控制（速度取决于细分值，请见下文）。
- 所有的 PWM 信号，在任意时刻，均可通过将当量位置值设置为 251，以停止相应通道的信号输出，极为利于舵机的保护。
- 集成高效、稳定的 UART 接口，支持波特率自动检测，当波特率不高于 28800bps 时为不丢帧检测，否则将丢失首帧指令包，此外，可以通过发送 4 个 **“0xff”** 以避免首帧丢失的现象。
- 集成高效、稳定的 I2C 接口。

– – – 关于 Arduino UNO R3 – – –

- 绝大多数的数字口和模拟口均通过 JST-PH2.0-3P 接线端子引出（数字口：10 路　/　模拟口：4 路），以方便传感器的连接。
- JST-PH2.0-3P 接线端子中的电源可配置（数字口：5V 或 Vin-外部电源输入　/　模拟口：3.3V 或 5V）。
- UNO 剩余的引脚通过 2*14Pin，间距为 2.54mm 的排针引出，下文简称为 **“扩展口”**。可扩展 UART、I2C 及 SPI 三种通讯接口，可灵活切换 USB<->UART<->UNO、USB<->UART<->SoftServo 或 UNO<->UART<->SoftServo，亦可灵活地将 UNO 的 ADC 参考电压切换到 3.3V，等等。
- 关于 Arduino UNO R3 的详细资料，请从 Arduino 官网获取 **www.arduino.cc**

用户寄存器定义

以下为 24 通道 SoftServo 舵机控制系统的寄存器分配表：

地址	名称	位置	权限	初始值	描述
1(0x01)	ProtocolVer	ROM	读	0x__	协议版本 (UART / I2C)
2(0x02)	DeviceMainClass	ROM	读	0x01	设备主类 (0x01 : 主控制器)
3(0x03)	DeviceSubClass	ROM	读	0x01	设备子类 (0x01 : ManShow-RC 系列)
4(0x04)	HardwareVer	ROM	读	0x1_	硬件版本 (0x1_ :)
5(0x05)	SoftwareVer	ROM	读	0x1_	软件版本 (0x1_ :)
6(0x06)	Command	RAM	写	0x00	命令字
7(0x07)	ID	ROM	读/写	0x01	设备地址 (1~126, 0 :广播地址)
8(0x08)	PWM_Cycle	ROM	读/写	0x14	PWM 信号的周期(5~250, x1ms)
9(0x09)	PWM_MinL	ROM	读/写	0xf4	PWM 有效宽度最小值低字节
10(0x0a)	PWM_MinH	ROM	读/写	0x01	PWM 有效宽度最小值高字节
11(0x0b)	PWM_MaxL	ROM	读/写	0xc4	PWM 有效宽度最大值低字节
12(0x0c)	PWM_MaxH	ROM	读/写	0x09	PWM 有效宽度最大值高字节
13(0x0d)	Position1	RAM	读/写	0xfb	通道 1 当量位置
14(0x0e)	Position2	RAM	读/写	0xfb	通道 2 当量位置
15(0x0f)	Position3	RAM	读/写	0xfb	通道 3 当量位置
16(0x10)	Position4	RAM	读/写	0xfb	通道 4 当量位置
17(0x11)	Position5	RAM	读/写	0xfb	通道 5 当量位置
18(0x12)	Position6	RAM	读/写	0xfb	通道 6 当量位置
19(0x13)	Position7	RAM	读/写	0xfb	通道 7 当量位置
20(0x14)	Position8	RAM	读/写	0xfb	通道 8 当量位置

21(0x15)	Position9	RAM	读/写	0xfb	通道 9 当量位置
22(0x16)	Position10	RAM	读/写	0xfb	通道 10 当量位置
23(0x17)	Position11	RAM	读/写	0xfb	通道 11 当量位置
24(0x18)	Position12	RAM	读/写	0xfb	通道 12 当量位置
25(0x19)	Position13	RAM	读/写	0xfb	通道 13 当量位置
26(0x1a)	Position14	RAM	读/写	0xfb	通道 14 当量位置
27(0x1b)	Position15	RAM	读/写	0xfb	通道 15 当量位置
28(0x1c)	Position16	RAM	读/写	0xfb	通道 16 当量位置
29(0x1d)	Position17	RAM	读/写	0xfb	通道 17 当量位置
30(0x1e)	Position18	RAM	读/写	0xfb	通道 18 当量位置
31(0x1f)	Position19	RAM	读/写	0xfb	通道 19 当量位置
32(0x20)	Position20	RAM	读/写	0xfb	通道 20 当量位置
33(0x21)	Position21	RAM	读/写	0xfb	通道 21 当量位置
34(0x22)	Position22	RAM	读/写	0xfb	通道 22 当量位置
35(0x23)	Position23	RAM	读/写	0xfb	通道 23 当量位置
36(0x24)	Position24	RAM	读/写	0xfb	通道 24 当量位置
37(0x25)	Subdivision1	RAM	读/写	0x01	通道 1 当量速度
38(0x26)	Subdivision2	RAM	读/写	0x01	通道 2 当量速度
39(0x27)	Subdivision3	RAM	读/写	0x01	通道 3 当量速度
40(0x28)	Subdivision4	RAM	读/写	0x01	通道 4 当量速度
41(0x29)	Subdivision5	RAM	读/写	0x01	通道 5 当量速度
42(0x2a)	Subdivision6	RAM	读/写	0x01	通道 6 当量速度
43(0x2b)	Subdivision7	RAM	读/写	0x01	通道 7 当量速度
44(0x2c)	Subdivision8	RAM	读/写	0x01	通道 8 当量速度
45(0x2d)	Subdivision9	RAM	读/写	0x01	通道 9 当量速度

46(0x2e)	Subdivision10	RAM	读/写	0x01	通道 10 当量速度
47(0x2f)	Subdivision11	RAM	读/写	0x01	通道 11 当量速度
48(0x30)	Subdivision12	RAM	读/写	0x01	通道 12 当量速度
49(0x31)	Subdivision13	RAM	读/写	0x01	通道 13 当量速度
50(0x32)	Subdivision14	RAM	读/写	0x01	通道 14 当量速度
51(0x33)	Subdivision15	RAM	读/写	0x01	通道 15 当量速度
52(0x34)	Subdivision16	RAM	读/写	0x01	通道 16 当量速度
53(0x35)	Subdivision17	RAM	读/写	0x01	通道 17 当量速度
54(0x36)	Subdivision18	RAM	读/写	0x01	通道 18 当量速度
55(0x37)	Subdivision19	RAM	读/写	0x01	通道 19 当量速度
56(0x38)	Subdivision20	RAM	读/写	0x01	通道 20 当量速度
57(0x39)	Subdivision21	RAM	读/写	0x01	通道 21 当量速度
58(0x3a)	Subdivision22	RAM	读/写	0x01	通道 22 当量速度
59(0x3b)	Subdivision23	RAM	读/写	0x01	通道 23 当量速度
60(0x3c)	Subdivision24	RAM	读/写	0x01	通道 24 当量速度
61(0x3d)	CurrentVoltage	RAM	读	0x__	当前供电电压*8

命令字定义

命令字起什么作用？首先，100 个用户寄存器或许不能迎刃所有的功能需求，保留这样一个字节，以备不时之需，理论上可以通过改变该字节的值扩展出额外 255 种类型的操作。其次，有些时候为便于记忆，希望对一个命令字节赋予一些特殊的值以实现一些特殊的操作（注：使用 0xf0 和 0xf1 命令字时，软件版本需高于 V12）。

- 0xbf ：使能软启动模式。
- 0xbc ：使能传统启动模式，此为设备上电后的默认模式。
- 0xcc ：同步用于产生 PWM 信号的定时器。

- 0xf0 ：当断开 PWM 信号后再次输出信号时，以上一次断开时的当量位置值为参考点，此为设备上电后的默认模式。
- 0xf1 ：当断开 PWM 信号后再次输出信号时，以中位为参考点。

技术问答

- **如何评价一个 PWM 类型的舵机控制器？**
  1. PWM 类型的舵机，其 PWM 有效宽度并非完全一致，考虑到兼容性问题，PWM 有效宽度必须可设定。
  2. 对于标准的 PWM 类型舵机，PWM 信号周期通常建议为 20ms，但对于数字舵机，可允许更高频率的信号输入，当应用系统要求舵机具有更高的响应速度时，可缩短 PWM 信号的周期。因此，PWM 信号的周期可人为设定，也是很有必要的。
  3. 有些时候我们希望所有的 PWM 信号同时输出，并且信号间的耦合性控制在肉眼难以察觉的程度。
  4. 在一些动态的应用系统中，尤其是人形机器人，我们希望在任意时刻每个通道的位置和速度均可单独控制，因为机器人的动态平衡是依靠惯性力实现的。
  5. 绝大多数的舵机控制器均将舵机的动作代码固化在 EEPROM 或 Flash ROM 中，或许这样操作起来很简单。我们认为，一个机器人系统，需要结合搭载的传感器与智能控制算法，动态生成舵机的动作代码。此外，关于通讯接口的简易性、实时性和稳定性也是非常的关键。
  6. 为了更好地保护舵机，必须能够在任意时刻停止 PWM 信号的输出。在使用过程中经常会发现一个现象：舵机毁于冲击力，原则上，当存在异常的冲击力时，应当停止信号的输出。例如：机器人处于跌落状态(可由加速度传感器检测)，倘若提前停止 PWM 信号的输出，使舵机进入惰行或制动状态，便可避免刚性冲击。

- **PWM 有效宽度、当量位置与实际位置之间是何种映射关系？**

倘若将 PWM 有效宽度直接作为指令包的参数，每个位置需要由 2 个字节来描述，这极大地增长了数据包的长度。折衷考虑位置控制的分辨率与通讯的实时性，我们认为以下陈述的是一种较为合理的方法。此外，可将当量位置值设置为 251，以停止 PWM 信号的输出，并且强烈建议使用周期信号锁存的 PWM 类型舵机(非单信号锁存)。

将 PWM 有效宽度的变化量进行 250 等分，例如：PWM 有效宽度范围为 500μs ~ 2500μs，相应的角度范围为 0°~180°，可得映

射关系如下所示(“X”的范围为 0~250)：

PWM 有效宽度（μs)	当量位置	实际位置（°)
500	0	0
2500	250	180
(2500-500)*X/250+500	X	180*X/250
低电平	251	惰行、制动或锁存(因舵机而异)

- **何谓细分？**

细分为当量速度，简而言之，速度的控制是通过细分实现的，此外，细分值与速度成反比关系。例如：PWM 有效宽度范围为 500μs~2500μs，细分值为 240，起始当量位置值为 30，目标当量位置值为 60，可得：脉冲宽度的变化量为 (2500-500)*(60-30)/250=240μs，这个变化量将在 240 个 PWM 信号周期内完成。

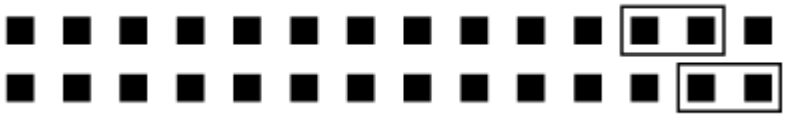
- **批量传输与控制传输有何不同？**

二者在 RS485 通讯协议中有简单提及(**ManShow-RC1** 无内置的 RS485 接口，但其 UART 接口的通讯协议与 **Gicren** 标准产品系中的 RS485 通讯协议一致)，在舵机控制中，有时我们侧重于脉宽的精度，有时我们却侧重于通讯的稳定性(不能忽略数据流中任何一帧有效的指令包)，这显然是矛盾的，但可以通过提高微控制器的处理速度与优化代码的方式来减小这种矛盾带来的影响。考虑到电平的兼容性与电路的简洁性，我们毅然选择工作电压为 5V 的微控制器，通过优化代码的方式来减小上文提及的矛盾所带来的影响，当然，若采用 32 位微控制器(大多数都具有 DMA 功能，UART 接口的数据接收可采用 DMA 方式)，可完全避免上文提及的矛盾，但工作电压大多为 3.3V 或更低。由于 I2C 接口的时钟具有可延展性，批量传输与控制传输仅针对 UART 接口而言，于是 **ManShow-RC1** 将 I2C 接口作为 Arduino UNO 与 SoftServo 的默认通讯接口，并在印刷电路板上连接在一起，而 SoftServo 的 UART 接口作为 USB<->UART<->SoftServo 直接通讯之用（此为主要用途，通过客户端调试软件调试动作组，应确保 Arduino UNO 没有通过 I2C 接口发送舵机控制指令，否则需要将扩展口中的 UNO-RST 接到 GND）以及 UNO 与 SoftServo 间的备用通讯接口。此外，应注意一点，在批量传输模式下，波特率必须高于 10000bps。或许您仍存在困惑，即在批量传输模式下如何确保指令包已被正确接收？一个指令包重复发送几次是非常有效的解决方法(建议连续发送三次)。

应用实例

- **（单舵机位置及速度的控制）通过一个电位器实现舵机的速度控制、两个按钮实现两个给定位置的切换**

步骤 1：　　确保 USB<->UART<->UNO 这一通讯链路正常连接，对应的扩展口连接关系如下图所示（ADC 参考电压为 5V）：



（USB-UART-TXD 与 UNO-RXD 连接，USB-UART-RXD 与 UNO-TXD 连接。）

步骤 2：　　将舵机连接到通道 1，电位器连接到模拟口 A0，按钮 A 连接到数字口 D2，按钮 B 连接到数字口 D3。

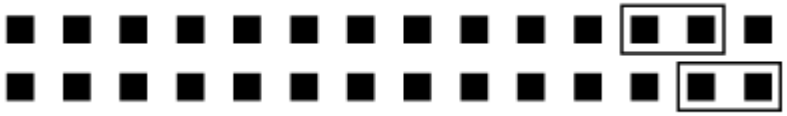
Arduino 代码(**SingleServoControl**)：

```
#include "GI2C_V11.h"
#include <Wire.h>

#define ButtonA           2
#define ButtonB           3
#define PositionA         50
#define PositionB         200
#define Potentiometer     A0

unsigned char Buf[61+1];
GI2CV11 ManShow_RC1(Buf,sizeof(Buf));

void setup()
{
   pinMode(ButtonA,INPUT_PULLUP);
   pinMode(ButtonB,INPUT_PULLUP);
}
```

```
void loop()
{
   ManShow_RC1.Read(1,13,48);
   while(1)
   {
      Buf[37]=(unsigned long int)250*analogRead(Potentiometer)/1024;
      if(Buf[37]==0)
      {
         Buf[37]=1;
      }
      if((digitalRead(ButtonA)==LOW) && (digitalRead(ButtonB)==HIGH))
      {
         Buf[13]=PositionA;
         ManShow_RC1.Write(1,13,48);
      }
      else if((digitalRead(ButtonA)==HIGH) && (digitalRead(ButtonB)==LOW))
      {
         Buf[13]=PositionB;
         ManShow_RC1.Write(1,13,48);
      }
      delay(20);
   }
}
```

- **（多舵机位置及速度的控制）同时控制 24 个通道的舵机，以不同的速度反复运动**

步骤 1：　　确保 USB<->UART<->UNO 这一通讯链路正常连接，对应的扩展口连接关系如下图所示：



（USB-UART-TXD 与 UNO-RXD 连接，USB-UART-RXD 与 UNO-TXD 连接。）

步骤 2：　将舵机分别连接到通道 1 至通道 24（当然可根据实际情况，连接几个通道以作测试）。

Arduino 代码(**MultiServoControl**)：

```
#include "GI2C_V11.h"
#include <Wire.h>

unsigned char Buf[61+1];
GI2CV11 ManShow_RC1(Buf,sizeof(Buf));

void setup()
{

}

void loop()
{
   unsigned char i;
   unsigned long int j;
   ManShow_RC1.Read(1,13,48);
   for(i=13;i<37;i++)
      Buf[i]=50;
   for(i=37;i<61;i++)
      Buf[i]=(i-36)*10;
```

```
  while(1)
  {
    for(j=0;j<2677114440;j++)
    {
      for(i=13;i<37;i++)
      {
        if((j+1)%(i-12)==0)
        {
          if(Buf[i]==50)
            Buf[i]=200;
          else if(Buf[i]==200)
            Buf[i]=50;
        }
      }
      delay(200);
      ManShow_RC1.Write(1,13,48);
    }
  }
}
```

上述三个实例简单介绍了如何结合 Arduino UNO 对舵机位置及速度进行控制，通过实例证明了舵机控制的实时性及各通道的独立性。由于采用寄存器操作的方式，因此只需实时设置相应通道的位置及速度寄存器即可。接下来，结合丰富的 Arduino 资源，开始您的机器人创作之旅吧！