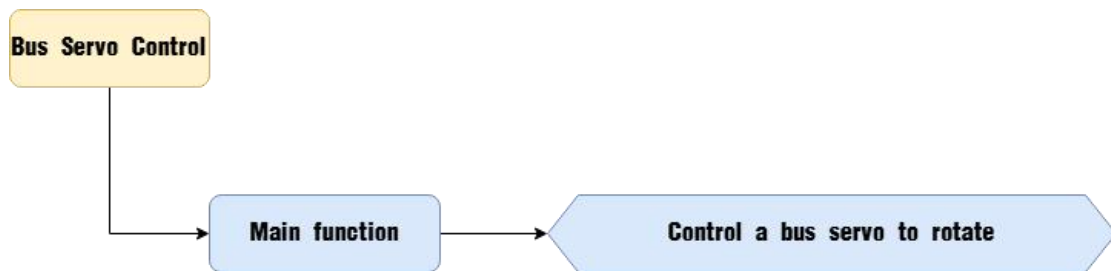


01 Serial Bus Servo Instructions

1. Project Overview

In this lesson, we will control the bus servo to rotate continuously in a loop.

2. Program Procedure



3. Module Introduction

A bus servo, also known as a serial bus smart servo, can be considered an advanced version of a digital servo. It typically uses a single-wire UART (serial) communication interface, meaning that both data transmission and reception are handled through a single wire.

In a standard UART setup, two wires are used: one for transmitting (TX) and one for receiving (RX), allowing full-duplex communication. However, in a single-wire UART configuration, a single line alternates between sending and receiving data. When not transmitting, the line remains in receive mode. During transmission, the receive function is temporarily disabled.

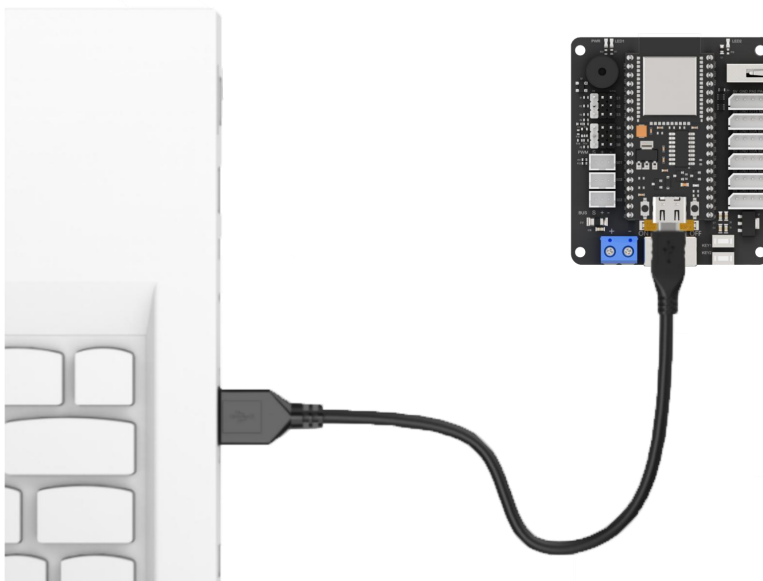
To achieve this on the servo controller, a Double Pole Double Throw (DPDT) switch is used to convert the standard two-wire UART into a single-wire UART. The TX and RX lines are connected to the communication line through a logic circuit.

- When idle (not transmitting), the controller outputs a low signal to the switch's enable pin. This connects RX to the communication line and disconnects TX.
- When transmitting data, the controller outputs a high signal to the enable pin. The logic circuit then connects TX to the communication line and disconnects RX.

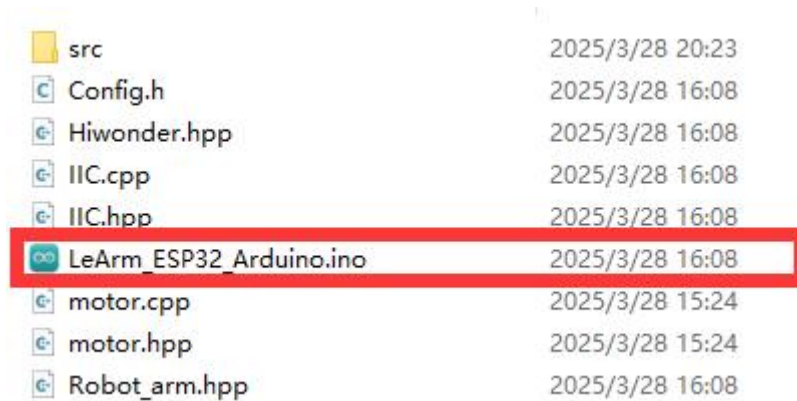
Note: The bus servo communication protocol used in this example is compatible with all bus servo models from our company. For details on the protocol format, please refer to the document titled "**03 Bus Servo Communication Protocol**" located in the same folder as this manual.

4. Program Download

- 1) Connect the main control board to the computer using a USB cable.



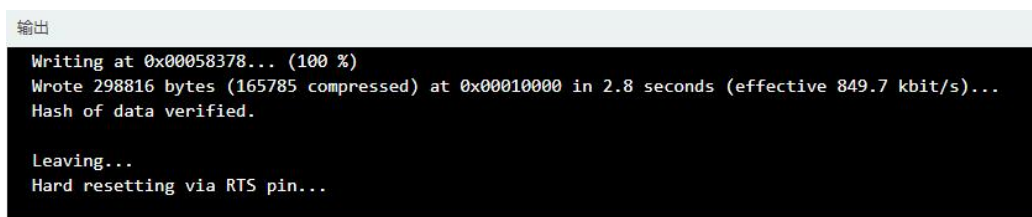
- 2) Locate the corresponding example project file in the Arduino project folder provided in the same directory as the documentation.



- 3) Open the project and select the appropriate board model, as shown in the image below.



- 4) First, click "Verify" (Compile), then click "Upload". Once the upload is complete, the output window at the bottom of the Arduino software should display a message similar to the one below, indicating that the program has been successfully uploaded.



5. Program Outcome

After the robotic arm is powered on, serial bus servo 1 will rotate.

6. Program Analysis

6.1 main.c File (Business Layer)

- 1) **Include** the config.h and Hiwonder.hpp header files. The Hiwonder.hpp file contains the definition of the bus servo object.

```
#include "Config.h"
#include "Hiwonder.hpp"
```

- 2) **Create** a bus servo object for subsequent control operations.

```
BusServo_t busservo_obj;
```

- 3) In the setup function:

- Delay for 1000 ms (1 second),
- Then cut off the power to the Bluetooth module,
- Create a software serial port named Serial1 and open it with a baud rate of 115200,
- Initialize the bus servo object and bind its communication to Serial1,
- Finally, open the default serial port and set its baud rate to 9600.

- 4) In the loop function (main loop):

① Call the set_angle method of the bus servo object to rotate PWM channel 1 to 0° over 1000 ms. At this point, the servo rotates to 0°, closing the robotic gripper.

② After a delay of 1100 ms (1.1 seconds), call the read_angle method to get the current angle of Servo 1 and print it via the serial port.

③ After another delay of 500 ms (0.5 seconds), call the set_angle method again to rotate PWM channel 1 to 180° over 2000 ms. This opens the

robotic gripper.

④ After a delay of 2100 ms (2.1 seconds), call the `read_angle` method again to get the current angle of Servo 1 and print it via the serial port.

⑤ Finally, delay another 500 ms (0.5 seconds) before repeating the loop.

```
void setup() {  
  delay(1000);  
  pinMode(IO_BLE_CTL, OUTPUT);  
  digitalWrite(IO_BLE_CTL, LOW); // Set the Bluetooth control pin to low to cut off the Bluetooth module power (设置蓝牙控制引脚为低电平时, 断开蓝牙模块电源)  
  
  Serial1.begin(115200, SERIAL_8N1, BUS_RX, BUS_TX);  
  busservo_obj.init(&Serial1);  
  
  Serial.begin(9600);  
  delay(2000);  
}
```

5) In the main loop (loop function):

① First, call the `set_angle` method of the bus servo object to rotate Servo 1 (PWM channel 1) to 0° over 1000 ms. At this point, the servo moves to 0°, causing the robotic gripper to close.

② After a delay of 1100 ms (1.1 seconds), call the `read_angle` method to read the current angle of Servo 1, and print the value through the serial port.

③ After another delay of 500 ms (0.5 seconds), call the `set_angle` method again to rotate Servo 1 to 180° over 2000 ms. This causes the servo to move to 180°, opening the robotic gripper.

④ After a delay of 2100 ms (2.1 seconds), call the `read_angle` method again to get the current angle of Servo 1, and print it via the serial port.

⑤ Finally, wait for another 500 ms (0.5 seconds) before starting the next loop cycle.

```
void loop() {  
    busservo_obj.set_angle(1, 0, 1000);  
    delay(1100);  
    Serial.println(busservo_obj.read_angle(1));  
    delay(500);  
    busservo_obj.set_angle(1, 180, 2000);  
    delay(2100);  
    Serial.println(busservo_obj.read_angle(1));  
    delay(500);  
}
```

6.2 SerialServo.cpp File (Low-Level)

6.2.1 BusServo_t::set_angle Method

① The set_angle method is used to control the servo's movement. It takes three parameters:

- (1) Servo ID,
- (2) Target position (angle),
- (3) Movement duration.

② Inside the method, a buffer buf is first created to hold the serial data to be sent. Then, the target angle (angle) is mapped to a corresponding position value (position), which is then limited to a valid range.

③ After that, the buffer is filled sequentially with the appropriate data values. Finally, the write method is called to send the data over the serial port.

```
void BusServo_t::set_angle(uint16_t servo_ID, uint32_t angle, uint16_t duration)
{
    uint8_t buf[10];
    uint16_t position = angle > 180 ? 1000 : (angle*1000/180);
    if(position < 0)
        position = 0;
    if(position > 1000)
        position = 1000;
    buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
    buf[2] = servo_ID;
    buf[3] = 7;
    buf[4] = LOBOT_SERVO_MOVE_TIME_WRITE;
    buf[5] = GET_LOW_BYTE(position);
    buf[6] = GET_HIGH_BYTE(position);
    buf[7] = GET_LOW_BYTE(duration);
    buf[8] = GET_HIGH_BYTE(duration);
    buf[9] = CheckSum(buf);
    SerialX->write(buf, 10);
}
```

Note: This function only prepares the command frame before data transmission. It does not handle the actual sending or receiving of data.
