

# Hiwonder

## EPS32 Development V1.0

---



### 1. Getting Started

#### 1.1 Wiring Instruction

This section employs an ESP32 microcontroller and ESP32 extension board for development, powered by a 12V 5A adapter. The bus servo is connected to the servo interface on the ESP32 extension board on the RRC controller, and the ESP32 microcontroller is connected to the computer using a data cable.



## 1.2 Environment Configuration

Install Python editor on PC. The software package is stored in “**2. Software -> 3. ESP32 Software**”. For the detailed operations of Python editor, please refer to the relevant tutorials.

## 2. Development Case - Python

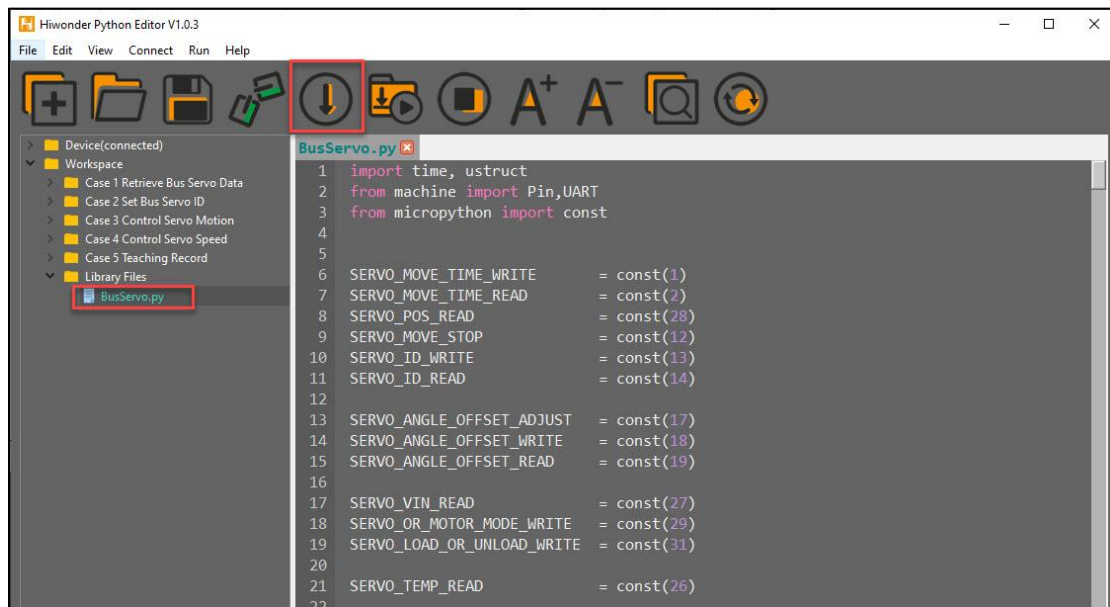
### 2.1 Case 1 Read Bus Servo Information

In this case, the servo ID, position, temperature and other information will be displayed in the terminal monitor.

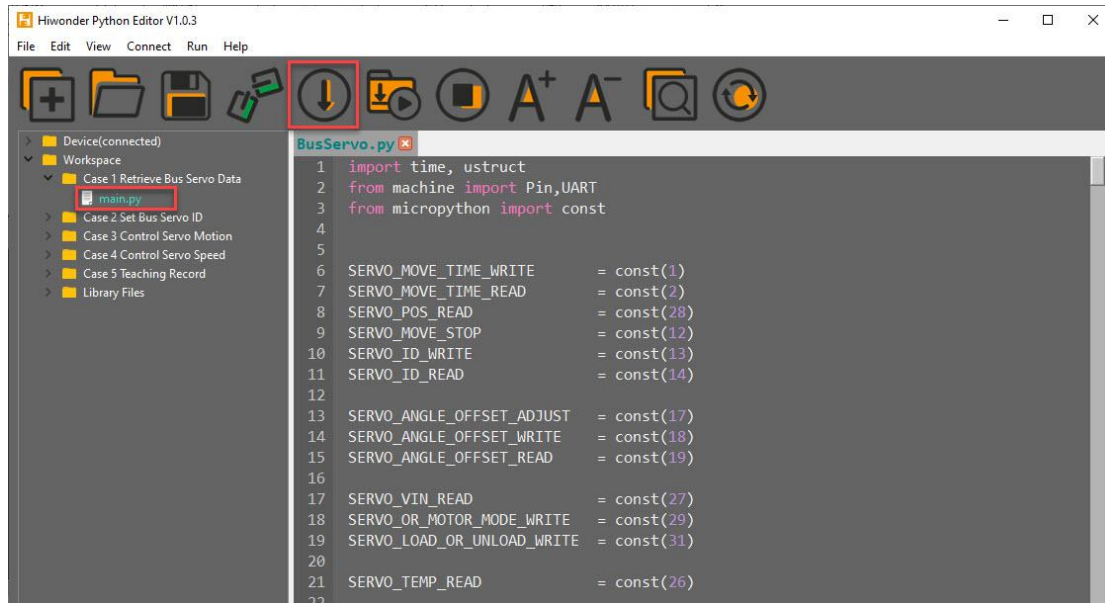
```
ID: 1
Position: 1053
Vin: 12.551
Offset: -16
Temp: 34
```

#### 2.1.1 Run Program

- 1) Double click “**BuServo.py**” to open the file, then click “**Download**” to download the library files into ESP32 microcontroller.



- 2) Download the “main.py” library file into ESP32 microcontroller.



3) Wait for the file download to complete, and click  to run the program.

```
>>>
Downloading.....
BusServo.py Download ok !
>>>
```



## 2.1.1 Run Program

After the program runs, the servo status information will be printed in the terminal.

```
ID: 1
Position: 870
Vin: 12.531
Offset: -16
Temp: 34
```

**id:** Servo ID. In this example, it is 1.

**Position:** The current position of the servo. In this example, it is 870.

**dev:** Servo deviation. In this example, it is -.

**Temp:** The current temperature of the servo. In this example, it is 34°C.

**Vin:** The current voltage value of the servo. In this example, it is 12.531V.

## 2.1.3 Case Program Analysis

- Import the Necessary Function Package

```
1 import time
2 from BusServo import BusServo
```

Import “**time**” module to execute time operations.

Import “**BusServo**” function package, which mainly encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- **Initialize bus\_servo Function**

```
6 bus_servo = BusServo()
```

- **Obtain and Print Servo Status**

```
8 =if __name__ == '__main__':
9
10     ID =bus_servo.get_ID(254)
11     .....
12     print('ID:', bus_servo.get_ID(254))
13
14     print('Position:', bus_servo.get_position(ID))
15
16     print('Vin:', bus_servo.get_vin(ID)/1000)
17
18     print('Offset:', bus_servo.get_offset(ID))
19
20     print('Temp:', bus_servo.get_temp(ID))
21
```

By calling various functions from the BusServo package, you can obtain various status information of the servo. This status information includes the servo ID, position, voltage, deviation, and current temperature.

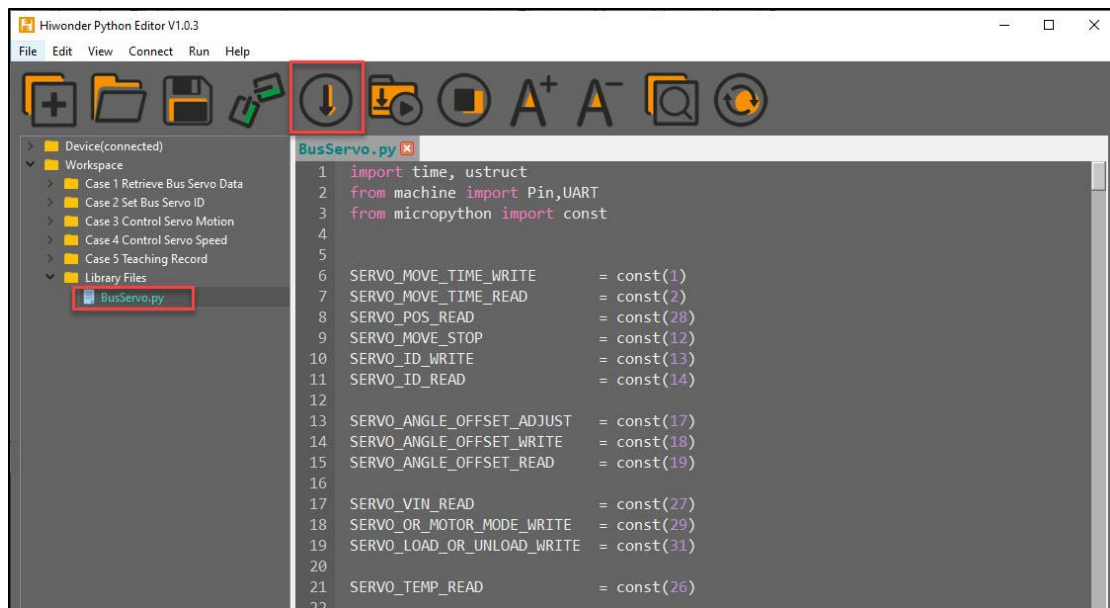
## 2.2 Case 2 Set Servo ID

In this case, adjust the servo ID and display the new ID of the bus servo in the terminal window.

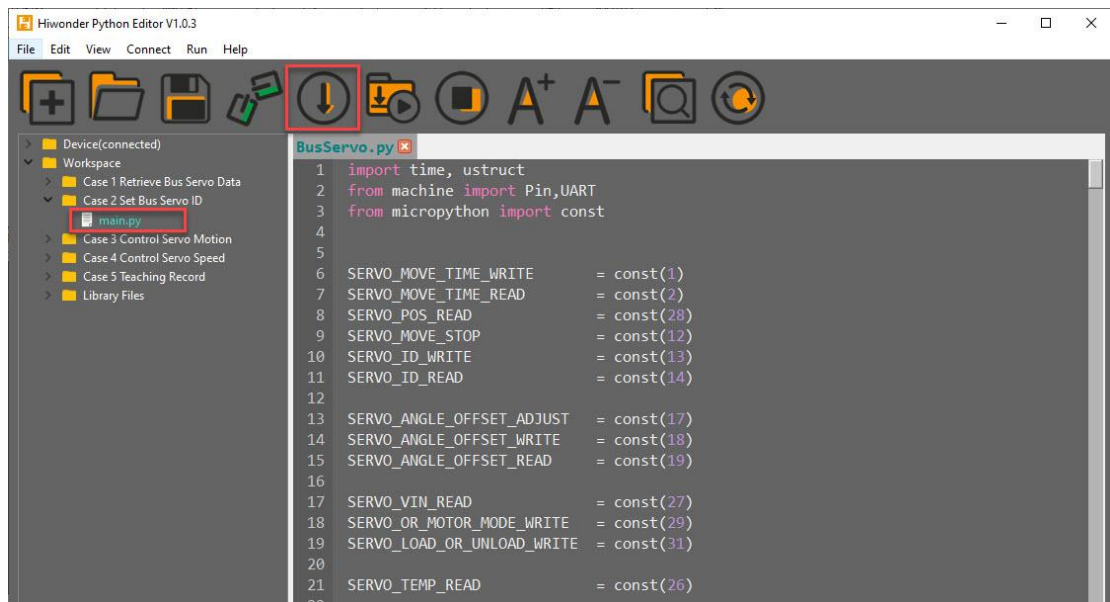
```
old ID: 1
new ID: 2
```

### 2.2.1 Run Program

- 1) Double click “**BuServo.py**” to open the file, then click “**Download**” to download the library files into ESP32 microcontroller.



2) Download the “main.py” library file into ESP32 microcontroller.



3) Wait for the file download to complete, and click  to run the program.

```
>>>
Downloading.....
BusServo.py Download ok !
>>>
```



## 2.2.2 Performance

After executing the program, the terminal window will display both the old and



new IDs of the servo.

```
old ID: 1  
new ID: 2
```

**old ID:** Indicates the old servo ID. In this example, it is 1.

**new ID:** Indicates the new servo ID. In this example, it is 2.

### 2.2.3 Case Program Analysis

- **Import the Necessary Function Package**

```
1 import time  
2 from BusServo import BusServo
```

Import “**time**” module to execute time operations.

Import “**BusServo**” function package, which mainly encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- **Initialize bus\_servo Function**

```
6 bus_servo = BusServo()
```

- **Obtain and Print Servo ID**

```
10 oldID =bus_servo.get_ID(254)  
11  
12 print('old ID:', bus_servo.get_ID(254))
```

By calling the `bus_servo.get_ID()` function, the ID value of the servo connected to the bus servo debug board is read. Here, the parameter value is 254, which represents the broadcast ID in the bus servo communication protocol and can be used to read the ID value of a servo with an unknown ID. The old servo ID value is then printed.

- **Set and Print New Servo ID**

```
14 newID = 2  
15  
16 bus_servo.set_ID(oldID, newID)  
17  
18 print('new ID:', newID)  
19
```

By calling the `bus_servo.set_ID()` function, the ID value of the servo connected to the bus servo debug board is changed to the value of “newID”. The new

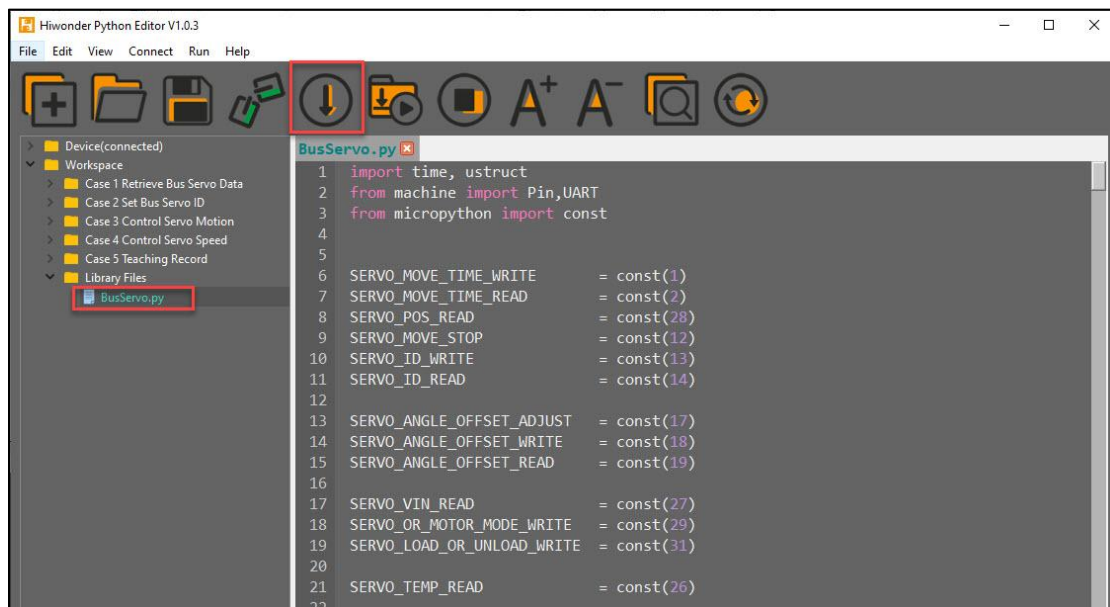
servo ID value is then printed out.

## 2.3 Case 3 Control Servo to Rotate

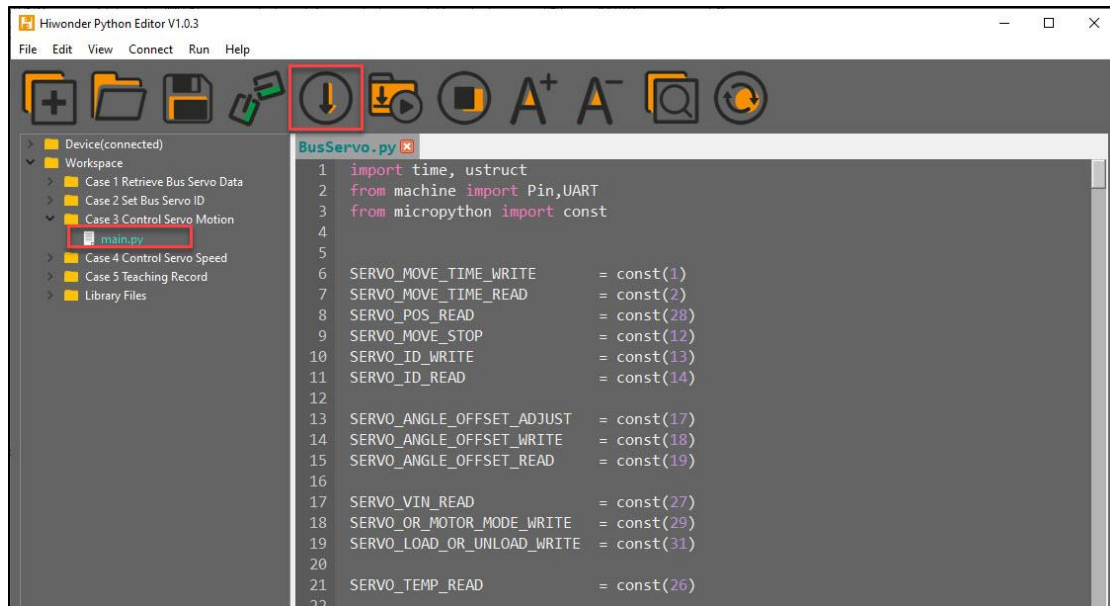
In this example, an ESP32 microcontroller controls a servo to rotate through positions 500, 1000, 0, and back to 500 with a 1-second interval between each movement.

### 2.3.1 Run Program

- 1) Double click **"BuServo.py"** to open the file, then click **"Download"** to download the library files into ESP32 microcontroller.



- 2) Download the **"main.py"** library file into ESP32 microcontroller.



3) Wait for the file download to complete, and click  to run the program.

```

>>>
Downloading.....
BusServo.py Download ok !
>>>
  
```



## 2.3.2 Performance

After the program runs, the servo rotates through positions 500, 1000, 0, and 500 with a 1-second interval between each movement.

## 2.3.3 Case Program Analysis

### ● Import the Necessary Function Package

```

1  import time
2  from BusServo import BusServo
  
```

Import “**time**” module to execute time operations.

Import “**BusServo**” function package, which mainly encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

### ● Initialize bus\_servo Function



```
6 bus_servo = BusServo()
```

- **Obtain Servo ID**

```
9 ID =bus_servo.get_ID(254)
10
```

By calling the **bus\_servo.get\_ID()** function with the parameter value set to 254, which represents the broadcast ID in the bus servo communication protocol, you can read the ID value of the servo connected to the bus servo debug board. This is useful for retrieving the ID of a servo when its specific ID is unknown.

- **Control Servo to Rotate**

```
bus_servo.run(ID, 500, 1000) # Set the servo to rotate to the position with pulse width 500 in 1000ms
time.sleep_ms(1000)          # delay for 1000ms

bus_servo.run(ID, 1000, 1000) # Set the servo to rotate to the position with pulse width 1000 in 1000ms
time.sleep_ms(1000)

bus_servo.run(ID, 0, 2000) # Set the servo to rotate to the position with pulse width 0 in 2000ms
time.sleep_ms(2000)

bus_servo.run(ID, 500, 1000) # Set the servo to rotate to the position with pulse width 500 in 1000ms
time.sleep_ms(1000)
```

By calling the **bus\_servo.run()** function, control of the servo rotation is achieved. The above process is as follow: the servo rotates to position 0 over 1 second, delays for 1 second, then rotates to position 1000 over 1 second, delays for 1 second, rotates to position 0 over 1 second, delays for 1 second, and finally rotates to position 500 over 1 second.

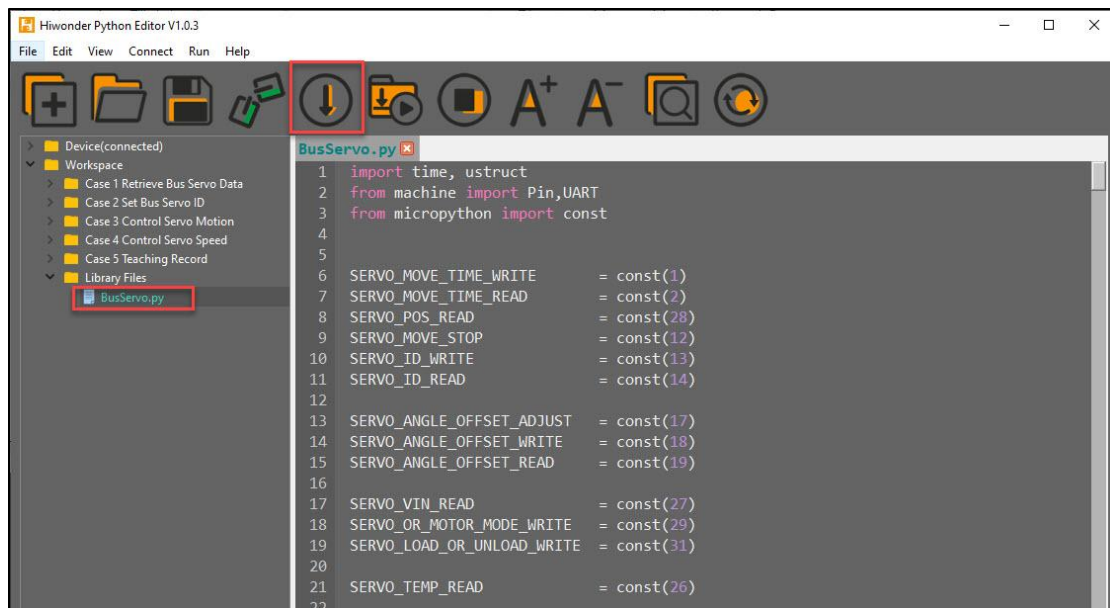
The servo's rotation range is from 0 to 1000, corresponding to angles from 0° to 240° .

## 2.4 Case 4 Adjust Servo Speed

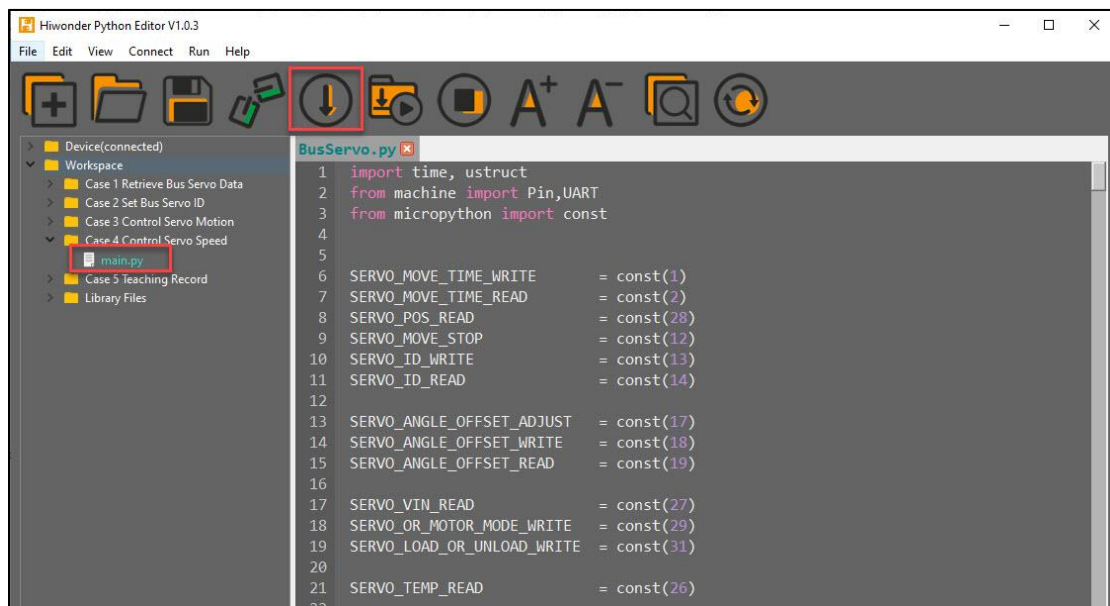
### 2.4.1 Run Program

In this case, an ESP32 microcontroller controls a servo to rotate at different speeds.

Double click “**BuServo.py**” to open the file, then click “**Download**” to download the library files into ESP32 microcontroller.



Download the “main.py” library file into ESP32 microcontroller.



Wait for the file download to complete, and click  to run the program.

```

>>>
Downloading.....
BusServo.py Download ok !
>>>
    
```



## 2.4.2 Performance

After the program runs, the servo performs as follows:

The servo starts from position 500 and rotates according to the following duration:

- It takes 0.5 seconds to move to position 1000.
- It takes 1.5 seconds to move to position 500 again.
- It takes 2.5 seconds to move to position 0.
- It takes 3.5 seconds to move back to position 500.

### 2.4.3 Case Program Analysis

- **Import the Necessary Function Package**

```
1 import time
2 from BusServo import BusServo
```

Import “**time**” module to execute time operations.

Import “**BusServo**” function package, which mainly encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- **Initialize bus\_servo Function**

```
6 bus_servo = BusServo()
```

- **Obtain Servo ID**

```
9 ID =bus_servo.get_ID(254)
10
```

By calling the **bus\_servo.get\_ID()** function with the parameter value set to 254, which represents the broadcast ID in the bus servo communication protocol, you can read the ID value of the servo connected to the bus servo debug board. This is useful for retrieving the ID of a servo when its specific ID is unknown.

- **Control Servo to Rotate**

```
bus_servo.run(ID, 500, 500) # Set the servo to rotate to the position with pulse width 500 in 500ms
time.sleep_ms(1000)        # delay for 1000ms

bus_servo.run(ID, 1000, 500) # Set the servo to rotate to the position with pulse width 1000 in 500ms
time.sleep_ms(1000)

bus_servo.run(ID, 500, 1500) # Set the servo to rotate to the position with pulse width 500 in 1500ms
time.sleep_ms(2000)

bus_servo.run(ID, 0, 2500) # Set the servo to rotate to the position with pulse width 0 in 2500ms
time.sleep_ms(3000)

bus_servo.run(ID, 500, 3500) # Set the servo to rotate to the position with pulse width 500 in 3500ms
time.sleep_ms(4000)
```

By controlling the servo's runtime, the speed of the servo can be adjusted.

Using the **bus\_servo.run()** function, the following process is implemented:

The servo rotates to position 500 over 0.5 seconds.

After a 1-second delay, the servo rotates to position 1000 over 0.5 seconds.

Following a 2-second delay, the servo rotates to position 500 over 1.5 seconds.

After a 3-second delay, the servo rotates to position 0 over 2.5 seconds.

Finally, after a 4-second delay, the servo rotates to position 500 over 3.5 seconds.

The servo's rotation range is from 0 to 1000, corresponding to angles from 0° to 240° .

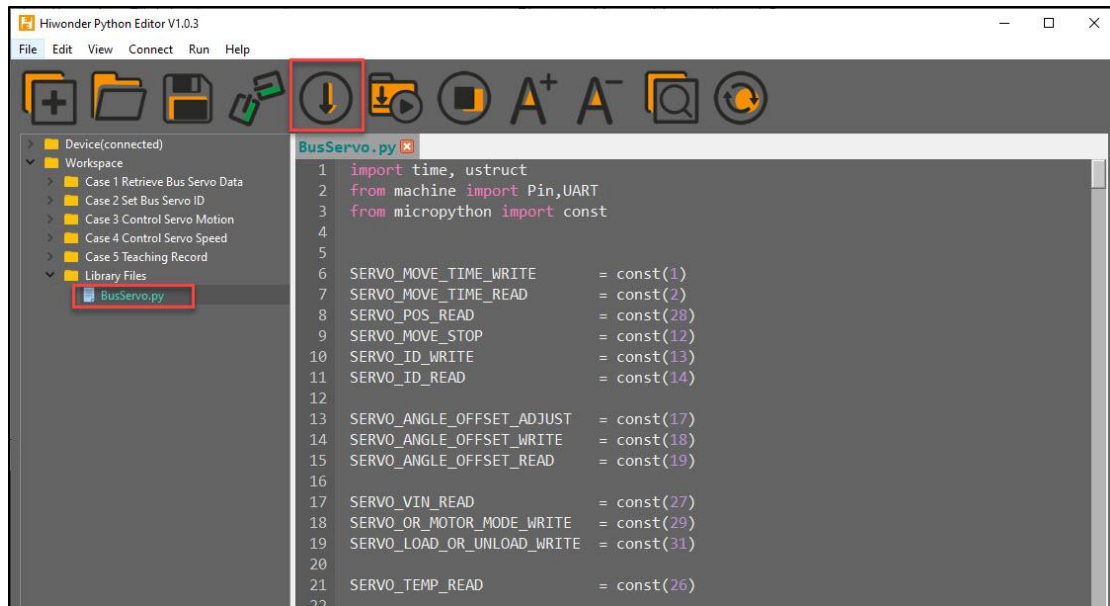
## 2.5 Case 5 Teaching Record Operations

In this example, an ESP32 microcontroller controls a servo by storing positions to make the servo rotate to specified angles.

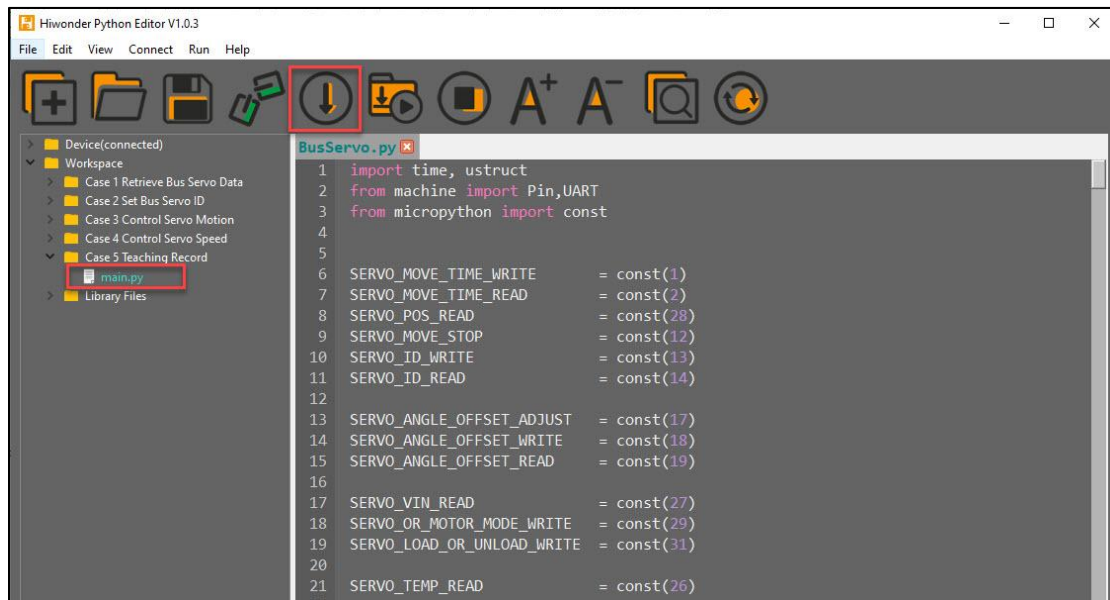
### 2.5.1 Run Program

In this example, an ESP32 microcontroller controls a servo motor to rotate at different speeds.

- 1) Double click **"BuServo.py"** to open the file, then click **"Download"** to download the library files into ESP32 microcontroller.



2) Download the “**main.py**” library file into ESP32 microcontroller.



3) Wait for the file download to complete, and click  to run the program.

```

>>>
Downloading.....
BusServo.py Download ok !
>>>
    
```





## 2.5.2 Performance

After the program runs, the terminal will print the servo's ID number. The servo will return to the 500 pulse width position. Once the "Start turning the servo" message is printed, the servo will begin powering down. You can then manually move the servo arm. Afterward, the servo will record the current position, return to the 500 pulse width position, and then move back to the position where it was manually adjusted.

## 2.5.3 Case Program Analysis

- **Import the Necessary Function Package**

```
1 import time
2 from BusServo import BusServo
```

Import **"time"** module to execute time operations.

Import **"BusServo"** function package, which mainly encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- **Initialize bus\_servo Function**

```
6 bus_servo = BusServo()
```

- **Obtain Servo ID**

```
9 ID = bus_servo.get_ID(254)
10
```

By calling the **bus\_servo.get\_ID()** function with the parameter value set to 254, which represents the broadcast ID in the bus servo communication protocol, you can read the ID value of the servo connected to the bus servo debug board. This is useful for retrieving the ID of a servo when its specific ID is unknown.

- **Control Servo Back to Central Position**

```
12 bus_servo.run(ID, 500, 1000)
13 time.sleep_ms(1000)
```

By calling the **bus\_servo.run()** function, control the servo to rotate and return to the 500 pulse width position. Use the **time.sleep\_ms()** function to perform a

1-second delay.

- **Print Prompt Message and Power Down the Servo**

```
print('Start turning the servo')  
  
bus_servo.unload(ID)  
time.sleep_ms(3000) # delay for 3000ms
```

Use print to print the prompt message "**Start turning the servo**".

Call `bus_servo.unload()` to power down the servo. Use `time.sleep_ms()` to introduce a 3-second delay, allowing us to manually move the servo arm.

- **Record Servo Position**

```
19 pos=bus_servo.get_position(ID)  
20 time.sleep_ms(2000)
```

Record the servo position by calling the `bus_servo.get_position()` function and assign it to "pos". Use the `time.sleep_ms()` function to introduce a 2-second delay.

- **Go Back to Central Postion**

```
bus_servo.run(ID, 500, 2000) # Set the servo to rotate to the position with pulse width 500 in 2000ms  
time.sleep_ms(3000)
```

Control the servo to rotate back to the 500 pulse width position by calling the `bus_servo.run()` function. Use the `time.sleep_ms()` function to introduce a 3-second delay.

- **Reset Servo**

```
bus_servo.run(ID, pos, 2000) # Set the servo to rotate to the position with previous pulse width in 2000ms  
time.sleep_ms(2000)
```

Use the `bus_servo.run()` function to control the servo to rotate to the recorded position "pos". Use the `time.sleep_ms()` function to introduce a 2-second delay.

## 3. Development Case - Arduino


### 3.1 Case 1 Read Bus Servo Information

In this case, the servo ID, position, temperature and other information will be displayed in the terminal monitor.

```
ID: 1
Position: 1053
Vin: 12.551
Offset: -16
Temp: 34
```

#### 3.1.1 Run Program

Open the program file “**BusServo\_status.ino**” stored in “**4.ESP32 Program/Arduino/ Case 1 Read Bus Servo Information/ BusServo\_status**”

Connect ESP32 microcontroller to the PC, and then click  to download the program.

(Note: If you encounter upload failure prompts, try disconnecting the servo debugging board from the ESP32 before uploading. After uploading is complete, reconnect the ESP32.)

#### 3.1.2 Performance

After the program runs, the various servo information will be printed in the terminal.

```
start...
ID:1
Position: 872
Vin: 12.51 V
Temp: 35
Dev:124
```

**id:** Servo ID. Example: 1

**Position:** Current position of the servo. Example: 872

**Vin:** Current voltage value of the servo. Example: 12.51V

**Temp:** Current temperature of the servo. Example: 35°C

**Dev:** Servo deviation. Example: 124

### 3.1.3 Case Program Analysis

- **Import the Necessary Function Package**

```
#include "LobotSerialServoControl.h" // import library file
```

Include the "**LobotSerialServoControl.h**" library, which primarily encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- **Initialize bus\_servo Function**

```
#define SERVO_SERIAL_RX    35
#define SERVO_SERIAL_TX    12
#define receiveEnablePin   13
#define transmitEnablePin  14
```

**SERVO\_SERIAL\_RX:** This macro is defined as the integer 35, representing the pin number or port number used for receiving (RX) data in the serial communication interface.

**SERVO\_SERIAL\_TX:** This macro is defined as the integer 12, representing the pin number or port number used for transmitting (TX) data in the serial communication interface.

**receiveEnablePin:** This macro is defined as the integer 13, representing a receive enable pin. In serial communication, the enable pin is typically used to control data reception operations.

**transmitEnablePin:** This macro is defined as the integer 14, representing a transmit enable pin. In serial communication, the enable pin is typically used to control data transmission operations.

```
HardwareSerial HardwareSerial(2);
LobotSerialServoControl BusServo(HardwareSerial, receiveEnablePin, transmitEnablePin);
```

**HardwareSerial HardwareSerial(2)** creates an object named

**HardwareSerial** and initializes it with 2 as the parameter. This indicates the

use of the second hardware serial communication port. Hardware serial communication ports are used for serial communication with external devices.

**LobotSerialServoControl BusServo** initializes an object named **BusServo** using a **HardwareSerial** object (**HardwareSerial** initialized earlier with port 2), **receiveEnablePin**, and **transmitEnablePin** as parameters. The **BusServo** object is used for controlling and managing serial servos.

- **Initialization Settings**

```
void setup() {  
  // put your setup code here, to run once:  
  Serial.begin(115200);           // set baud rate of serial port  
  Serial.println("start...");     // serial port prints "start..."  
  BusServo.OnInit();              // initialize bus servo library  
  HardwareSerial.begin(115200, SERIAL_8N1, SERVO_SERIAL_RX, SERVO_SERIAL_TX);  
  delay(500);                     // delay for 500ms  
}
```

**Serial.begin(115200):** this initializes the default serial port and sets the baud rate to 115200.

**Serial.println("start..."):** This uses the default serial port to print a text message, "start...".

**BusServo.OnInit() :** This calls the **OnInit** method of the **BusServo** object, initializing the bus servo library.

**HardwareSerial.begin(115200,SERIAL\_8N1,SERVO\_SERIAL\_RX,SERVO\_SERIAL\_TX):** This initializes a serial communication object named **HardwareSerial** with a baud rate of 115200 bps, 8 data bits, no parity bit (N), and 1 stop bit (1).

- **Read and Print the Relevant Information**



```
void loop() {  
  
    Serial.print("ID: ");  
    Serial.println(BusServo.LobotSerialServoReadID(0xFE)); // obtain servo ID and print via serial port  
    delay(500); // delay  
  
    int ID =1;  
    Serial.print("Position: ");  
    Serial.println(BusServo.LobotSerialServoReadPosition(ID)); // obtain servo position and print via serial port  
    delay(500); // delay  
  
    Serial.print("Vin: ");  
    Serial.print(BusServo.LobotSerialServoReadVin(ID)/1000.0); // obtain servo voltage and print via serial port  
    Serial.println(" V");  
    delay(900); // delay  
  
    Serial.print("Temp: ");  
    Serial.println(BusServo.LobotSerialServoReadTemp(ID)); // obtain servo temperature and print via serial port  
    delay(500); // delay  
  
    Serial.print("Dev: ");  
    Serial.println(BusServo.LobotSerialServoReadDev(ID)); // obtain servo deviation and print via serial port  
    delay(1000); // delay  
  
}
```

To retrieve various status information of the servo using the functions mentioned earlier, including servo ID, position, voltage, deviation, and current temperature, and then print them out, you would typically do something like this in code.

## 3.2 Case 2 Set Servo ID


In this case, adjust the servo ID and display the new ID of the bus servo in the terminal window.

```
oldID: 1  
newID: 2
```

### 3.2.1 Run Program

Open the program file “**BusServo\_setID.ino**” stored in “**4. ESP32 Program/**

## Arduino/ Case 3 Set Servo ID/ BusServo\_setID”

Connect ESP32 microcontroller to the PC, and then click  to download the program.

(Note: If you encounter upload failure prompts, try disconnecting the servo debugging board from the ESP32 before uploading. After uploading is complete, reconnect the ESP32.)

### 3.2.2 Performance

After the program runs, the new and old IDs of the servo will be printed in the terminal window.

```
oldID: 1
newID: 2
```

Old ID: The servo ID before modification. In this example, it is 1.

New ID: The servo ID after modification. In this example, it is 2.

### 3.2.3 Case Program Analysis

- Import the Necessary Function Package

```
#include "LobotSerialServoControl.h" // import library file
```

Include the "**LobotSerialServoControl.h**" library, which primarily encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- Initialize bus\_servo Function

```
#define SERVO_SERIAL_RX    35
#define SERVO_SERIAL_TX    12
#define receiveEnablePin   13
#define transmitEnablePin  14
```

**SERVO\_SERIAL\_RX:** This macro is defined as the integer 35, representing the pin number or port number used for receiving (RX) data in the serial communication interface.

**SERVO\_SERIAL\_TX:** This macro is defined as the integer 12, representing the pin number or port number used for transmitting (TX) data in the serial communication interface.

**receiveEnablePin:** This macro is defined as the integer 13, representing a receive enable pin. In serial communication, the enable pin is typically used to control data reception operations.

**transmitEnablePin:** This macro is defined as the integer 14, representing a transmit enable pin. In serial communication, the enable pin is typically used to control data transmission operations.

```
HardwareSerial HardwareSerial(2);
LobotSerialServoControl BusServo(HardwareSerial, receiveEnablePin, transmitEnablePin);
```

**HardwareSerial HardwareSerial(2)** creates an object named **HardwareSerial** and initializes it with 2 as the parameter. This indicates the use of the second hardware serial communication port. Hardware serial communication ports are used for serial communication with external devices. **LobotSerialServoControl BusServo** initializes an object named **BusServo** using a **HardwareSerial** object (**HardwareSerial** initialized earlier with port 2), **receiveEnablePin**, and **transmitEnablePin** as parameters. The **BusServo** object is used for controlling and managing serial servos.

- **Initialization Settings**

```
void setup() {
    // put your setup code here, to run once:
    Serial.begin(115200);          // set baud rate of serial port
    Serial.println("start...");    // serial port prints "start..."
    BusServo.OnInit();             // initialize bus servo library
    HardwareSerial.begin(115200, SERIAL_8N1, SERVO_SERIAL_RX, SERVO_SERIAL_TX);
    delay(500);                   // delay for 500ms
}
```

**Serial.begin(115200):** this initializes the default serial port and sets the baud rate to 115200.

**Serial.println("start..."):** This uses the default serial port to print a text message, "start...".

**BusServo.OnInit() :** This calls the OnInit method of the BusServo object, initializing the bus servo library.

**HardwareSerial.begin(115200,SERIAL\_8N1,SERVO\_SERIAL\_RX,SERVO\_SERIAL\_TX):** This initializes a serial communication object named HardwareSerial with a baud rate of 115200 bps, 8 data bits, no parity bit (N), and 1 stop bit (1).

## ● Obtain and Print Servo ID

```
Serial.print("oldID: ");
Serial.println(BusServo.LobotSerialServoReadID(0xFE)); // obtain servo ID and print via serial port
delay(1000); // delay

uint8_t oldID =BusServo.LobotSerialServoReadID(0xFE);
delay(1000); // delay
```

By calling the **BusServo.LobotSerialServoReadID()** function, read the ID value of the servo connected to the bus servo debugging board. Here, the parameter value is 0xFE, which translates to 254 in decimal and represents a broadcast ID in the bus servo communication protocol, used to read the ID value of servos with unknown IDs. Print the old servo ID value.

## ● Set and Print Servo New ID

```
uint8_t newID =2;
BusServo.LobotSerialServoSetID(oldID, newID);
delay(1000); // delay

Serial.print("newID: ");
Serial.println(String(newID)); // obtain servo position and print via serial port
delay(500); // delay
```


By calling **BusServo.LobotSerialServoSetID()** function, change the ID value of the servo connected to the bus servo debugging board to the value of "newID". Print the new servo ID value.

### 3.3 Case 3 Control Servo to Rotate

In this case, the ESP32 microcontroller controls the servo to rotate between positions 500, 1000, 0, and 500 with 1-second intervals.

#### 3.3.1 Run Program

Open the program file "**BusServo\_turn.ino**" stored in "**4. ESP32 Program/Arduino/ Case 3 Control Servo to Rotate/ BusServo\_turn**"

Connect ESP32 microcontroller to the PC, and then click  to download the program.

(Note: If you encounter upload failure prompts, try disconnecting the servo debugging board from the ESP32 before uploading. After uploading is complete, reconnect the ESP32.)

#### 3.3.2 Performance

After the program runs, servo will rotate between positions 500, 1000, 0, and 500 with 1-second intervals.

#### 3.3.3 Case Program Analysis

- **Import the Necessary Function Package**

```
#include "LobotSerialServoControl.h" // import library file
```

Include the "**LobotSerialServoControl.h**" library, which primarily



encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- **Initialize bus\_servo Function**

```
#define SERVO_SERIAL_RX    35
#define SERVO_SERIAL_TX    12
#define receiveEnablePin  13
#define transmitEnablePin 14
```

**SERVO\_SERIAL\_RX:** This macro is defined as the integer 35, representing the pin number or port number used for receiving (RX) data in the serial communication interface.

**SERVO\_SERIAL\_TX:** This macro is defined as the integer 12, representing the pin number or port number used for transmitting (TX) data in the serial communication interface.

**receiveEnablePin:** This macro is defined as the integer 13, representing a receive enable pin. In serial communication, the enable pin is typically used to control data reception operations.

**transmitEnablePin:** This macro is defined as the integer 14, representing a transmit enable pin. In serial communication, the enable pin is typically used to control data transmission operations.

```
HardwareSerial HardwareSerial(2);
LobotSerialServoControl BusServo(HardwareSerial, receiveEnablePin, transmitEnablePin);
```

**HardwareSerial HardwareSerial(2)** creates an object named **HardwareSerial** and initializes it with 2 as the parameter. This indicates the use of the second hardware serial communication port. Hardware serial communication ports are used for serial communication with external devices. **LobotSerialServoControl BusServo** initializes an object named **BusServo** using a **HardwareSerial** object (**HardwareSerial** initialized earlier with port 2), **receiveEnablePin**, and **transmitEnablePin** as parameters. The **BusServo** object is used for controlling and managing serial servos.

- **Initialization Settings**

```
void setup() {  
  // put your setup code here, to run once:  
  Serial.begin(115200);      // set baud rate of serial port  
  Serial.println("start..."); // serial port prints "start..."  
  BusServo.OnInit();         // initialize bus servo library  
  HardwareSerial.begin(115200, SERIAL_8N1, SERVO_SERIAL_RX, SERVO_SERIAL_TX);  
  delay(500);                // delay for 500ms  
}
```

**Serial.begin(115200):** this initializes the default serial port and sets the baud rate to 115200.

**Serial.println("start..."):** This uses the default serial port to print a text message, "start...".

**BusServo.OnInit() :** This calls the OnInit method of the BusServo object, initializing the bus servo library.

**HardwareSerial.begin(115200,SERIAL\_8N1,SERVO\_SERIAL\_RX,SERVO\_SERIAL\_TX):** This initializes a serial communication object named HardwareSerial with a baud rate of 115200 bps, 8 data bits, no parity bit (N), and 1 stop bit (1).

## ● Control Servo to Rotate

```
BusServo.LobotSerialServoMove(1,500,1000); // Set servo 1 to rotate to the position with pulse width 500 in 1000ms  
delay(2000); // delay for 2000ms  
  
BusServo.LobotSerialServoMove(1,1000,1000); // Set servo 1 to rotate to the position with pulse width 1000 in 1000ms  
delay(2000); // delay for 2000ms  
  
BusServo.LobotSerialServoMove(1,0,1000); // Set servo 1 to rotate to the position with pulse width 0 in 1000ms  
delay(2000); // delay for 2000ms  
  
BusServo.LobotSerialServoMove(1,500,1000); // Set servo 1 to rotate to the position with pulse width 500 in 1000ms  
delay(2000); // delay for 2000ms
```


By calling the **bus\_servo.run()** function, control the servo's rotation. The above process mainly achieves the following: the servo rotates to position 0 over 1 second, waits for 1 second, then rotates to position 1000 over 1 second, waits for 1 second, rotates back to position 0 over 1 second, waits for 1 second, and finally rotates to position 500 over 1 second.

The servo's rotation range is from 0 to 1000, corresponding to an angle range of 0° to 240° .

## 3.4 Case 4 Adjust Servo Speed

### 3.4.1 Run Program

Open the program file “**BusServo\_speed.ino**” stored in “**4. ESP32 Program/ Arduino/ Case 4 Adjust Servo Speed/ BusServo\_speed**”

Connect ESP32 microcontroller to the PC, and then click  to download the program.

(Note: If you encounter upload failure prompts, try disconnecting the servo debugging board from the ESP32 before uploading. After uploading is complete, reconnect the ESP32.)

### 3.4.2 Performance

After the program runs, the servo behaves as follows:

- ① The servo starts from position 500.
- ② It rotates to position 1000 over 0.5 seconds.
- ③ It rotates to position 500 over 1.5 seconds.
- ④ It rotates to position 0 over 2.5 seconds.
- ⑤ It rotates to position 500 over 3.5 seconds.

### 3.4.3 Case Program Analysis

- **Import the Necessary Function Package**

```
#include "LobotSerialServoControl.h" // import library file
```

Include the “**LobotSerialServoControl.h**” library, which primarily encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

- **Initialize bus\_servo Function**

```
#define SERVO_SERIAL_RX    35
#define SERVO_SERIAL_TX    12
#define receiveEnablePin   13
#define transmitEnablePin  14
```

**SERVO\_SERIAL\_RX**: This macro is defined as the integer 35, representing

the pin number or port number used for receiving (RX) data in the serial communication interface.

**SERVO\_SERIAL\_TX:** This macro is defined as the integer 12, representing the pin number or port number used for transmitting (TX) data in the serial communication interface.

**receiveEnablePin:** This macro is defined as the integer 13, representing a receive enable pin. In serial communication, the enable pin is typically used to control data reception operations.

**transmitEnablePin:** This macro is defined as the integer 14, representing a transmit enable pin. In serial communication, the enable pin is typically used to control data transmission operations.

```
HardwareSerial HardwareSerial(2);  
LobotSerialServoControl BusServo(HardwareSerial, receiveEnablePin, transmitEnablePin);
```

**HardwareSerial HardwareSerial(2)** creates an object named **HardwareSerial** and initializes it with 2 as the parameter. This indicates the use of the second hardware serial communication port. Hardware serial communication ports are used for serial communication with external devices. **LobotSerialServoControl BusServo** initializes an object named **BusServo** using a **HardwareSerial** object (**HardwareSerial** initialized earlier with port 2), **receiveEnablePin**, and **transmitEnablePin** as parameters. The **BusServo** object is used for controlling and managing serial servos.

### ● Initialization Settings

```
void setup() {  
    // put your setup code here, to run once:  
    Serial.begin(115200);           // set baud rate of serial port  
    Serial.println("start...");    // serial port prints "start..."  
    BusServo.OnInit();              // initialize bus servo library  
    HardwareSerial.begin(115200, SERIAL_8N1, SERVO_SERIAL_RX, SERVO_SERIAL_TX);  
    delay(500);                    // delay for 500ms  
}
```

**Serial.begin(115200):** this initializes the default serial port and sets the baud rate to 115200.

**Serial.println("start..."):** This uses the default serial port to print a text message, "start...".

**BusServo.OnInit() :** This calls the OnInit method of the BusServo object, initializing the bus servo library.

**HardwareSerial.begin(115200,SERIAL\_8N1,SERVO\_SERIAL\_RX,SERVO\_SERIAL\_TX):** This initializes a serial communication object named HardwareSerial with a baud rate of 115200 bps, 8 data bits, no parity bit (N), and 1 stop bit (1).

### ● Control Servo to Rotate

```
BusServo.LobotSerialServoMove(1,500,500); // Set servo 1 to rotate to the position with pulse width 500 in 500ms
delay(2000); // delay for 2000ms

BusServo.LobotSerialServoMove(1,1000,500); // Set servo 1 to rotate to the position with pulse width 1000 in 500ms
delay(1500); // delay for 1500ms

BusServo.LobotSerialServoMove(1,500,1500); // Set servo 1 to rotate to the position with pulse width 500 in 1500ms
delay(2000); // delay for 2000ms

BusServo.LobotSerialServoMove(1,0,2500); // Set servo 1 to rotate to the position with pulse width 0 in 2500ms
delay(3000); // delay for 3000ms

BusServo.LobotSerialServoMove(1,500,3500); // Set servo 1 to rotate to the position with pulse width 500 in 3500ms
delay(4000); // delay for 4000ms
```

Control the servo's speed by adjusting its running time. Use the BusServo.LobotSerialServoMove() function to control the servo's rotation. The above process primarily achieves the following: rotate to position 500 over 0.5 seconds, wait for 1 second, rotate to position 1000 over 0.5 seconds, wait for 2 seconds, rotate to position 500 over 1.5 seconds, wait for 3 seconds, rotate to position 0 over 2.5 seconds, wait for 4 seconds, and finally rotate to position 500 over 3.5 seconds.

The servo's rotation range is from 0 to 1000, corresponding to an angle range of 0° to 240° .


## 3.5 Case 5 Teaching Record Operations

In this example, an ESP32 microcontroller controls a servo by storing positions to make the servo rotate to specified angles.



## 3.5.1 Run Program

Open the program file “**BusServo\_record.ino**” stored in “**4. ESP32 Program/ Arduino/ Case 4 Adjust Servo Speed/ BusServo\_record**”

Connect ESP32 microcontroller to the PC, and then click  to download the program.

(Note: If you encounter upload failure prompts, try disconnecting the servo debugging board from the ESP32 before uploading. After uploading is complete, reconnect the ESP32.)

## 3.5.2 Performance

After the program runs, the terminal will print the servo's ID number. The servo will return to the 500 pulse width position. Once the "Start turning the servo" message is printed, the servo will begin powering down. You can then manually move the servo arm. Afterward, the servo will record the current position, return to the 500 pulse width position, and then move back to the position where it was manually adjusted.

## 3.5.3 Case Program Analysis

### ● Import the Necessary Function Package

```
#include "LobotSerialServoControl.h" // import library file
```

Include the "**LobotSerialServoControl.h**" library, which primarily encapsulates various functional modules for bus servo communication. We can use the variables and functions defined in it to control the servo.

### ● Initialize bus\_servo Function

```
#define SERVO_SERIAL_RX    35
#define SERVO_SERIAL_TX    12
#define receiveEnablePin  13
#define transmitEnablePin 14
```

**SERVO\_SERIAL\_RX**: This macro is defined as the integer 35, representing the pin number or port number used for receiving (RX) data in the serial communication interface.

**SERVO\_SERIAL\_TX:** This macro is defined as the integer 12, representing the pin number or port number used for transmitting (TX) data in the serial communication interface.

**receiveEnablePin:** This macro is defined as the integer 13, representing a receive enable pin. In serial communication, the enable pin is typically used to control data reception operations.

**transmitEnablePin:** This macro is defined as the integer 14, representing a transmit enable pin. In serial communication, the enable pin is typically used to control data transmission operations.

```
HardwareSerial HardwareSerial(2);  
LobotSerialServoControl BusServo(HardwareSerial, receiveEnablePin, transmitEnablePin);
```

**HardwareSerial HardwareSerial(2)** creates an object named **HardwareSerial** and initializes it with 2 as the parameter. This indicates the use of the second hardware serial communication port. Hardware serial communication ports are used for serial communication with external devices. **LobotSerialServoControl BusServo** initializes an object named **BusServo** using a **HardwareSerial** object (**HardwareSerial** initialized earlier with port 2), **receiveEnablePin**, and **transmitEnablePin** as parameters. The **BusServo** object is used for controlling and managing serial servos.

### ● Initialization Settings

```
void setup() {  
    // put your setup code here, to run once:  
    Serial.begin(115200);           // set baud rate of serial port  
    Serial.println("start...");    // serial port prints "start..."  
    BusServo.OnInit();              // initialize bus servo library  
    HardwareSerial.begin(115200, SERIAL_8N1, SERVO_SERIAL_RX, SERVO_SERIAL_TX);  
    delay(500);                    // delay for 500ms  
}
```

**Serial.begin(115200):** this initializes the default serial port and sets the baud rate to 115200.

**Serial.println("start..."):** This uses the default serial port to print a text message, "start..."

**BusServo.OnInit()** : This calls the OnInit method of the BusServo object, initializing the bus servo library.

**HardwareSerial.begin(115200,SERIAL\_8N1,SERVO\_SERIAL\_RX,SERVO\_SERIAL\_TX):** This initializes a serial communication object named HardwareSerial with a baud rate of 115200 bps, 8 data bits, no parity bit (N), and 1 stop bit (1).

## ● Obtain Servo ID

```
Serial.print("ID: ");  
Serial.println(BusServo.LobotSerialServoReadID(0xFE)); // obtain servo ID and print via serial port  
delay(500); // delay  
  
uint8_t ID =BusServo.LobotSerialServoReadID(0xFE);
```

By calling the **BusServo.LobotSerialServoReadID()** function, you can read the ID value of the servo connected to the bus servo debugging board. Here, the parameter value is 0xFE, which translates to 254 in decimal. In the bus servo communication protocol, 254 is known as the broadcast ID, and it is used to read the ID value of servos whose ID is unknown.

## ● Control Servo Back to Central Position

```
BusServo.LobotSerialServoMove(ID,500,1000); // Set the servo to rotate to the position with pulse width 500 in 1000ms  
delay(1000); // delay for 1000ms
```

By calling the **BusServo.LobotSerialServoMove()** function, control the servo to rotate and return to the 500 pulse width position. Use the **time.sleep\_ms()** function to perform a 1-second delay.

## ● Print Prompt Message and Power Down the Servo

```
Serial.println("Start turning the servo");  
BusServo.LobotSerialServoUnload(ID);//set the servo to power off for 3s  
delay(3000); // delay for 3000ms
```

Use print to print the prompt message **"Start turning the servo"**.

Call the **BusServo.LobotSerialServoUnload()** function to power down the servo. Use **time.sleep\_ms()** to introduce a 3-second delay, allowing us to manually move the servo arm.

## ● Record Servo Position

```
int16_t position=BusServo.LobotSerialServoReadPosition(ID);// obtain the current pulse width position of  
delay(2000); // delay for 2000ms
```

Record the servo position by calling the

**BusServo.LobotSerialServoReadPosition()** function and assign it to "pos".

Use the **time.sleep\_ms()** function to introduce a 2-second delay.

## ● Reset Servo

```
BusServo.LobotSerialServoMove(ID,position,1000); // Set the servo to rotate to the position with  
delay(1000); // delay for 1000ms
```

Use the **BusServo.LobotSerialServoMove()** function to control the servo to rotate to the recorded position "**position**". Use the **time.sleep\_ms()** function to introduce a 2-second delay.