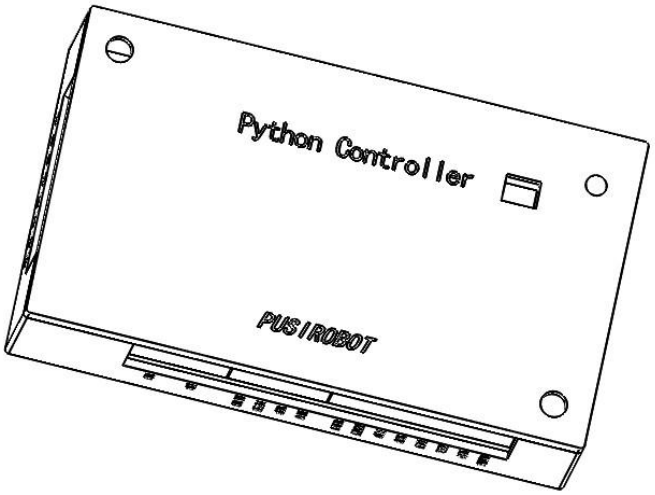
PUSIROBOT

CQPUSI ROBOT CONTROL SYSTEM

User Manual

Python Controller Series

Programmable controller



1. Version Control

1） Update Records

Date	Author	Version	Remark
2024-9-30	Tony	V1.0.1	Initial

Catalog

1 Introduction

1.1 Disclaimer

The using method of the device and other content in the description of this manual is only used to provide convenience for you. To ensure the application conforms to the technical specifications is the responsibility of your own. CQPUSI does not make any form of statement or guarantee to the information, which include but not limited to usage, quality, performance, merchantability or applicability of specific purpose. CQPUSI is not responsible for these information and the consequences result caused by such information. If the CQPUSI device is used for life support and/or life safety applications, all risks are borne by the buyer. The buyer agrees to protect the CQPUSI from legal liability and compensation for any injury, claim, lawsuit or loss caused by the application.

2 Overview

2.1 General Description

The Python Controller is a programmable controller that integrates communication interfaces such as CANOpen, RS485, and SPI, along with analog input, isolated PWM output, and motor control interfaces. It can connect to various devices and achieve complex control logic through MicroPython programming.

2.2 Features

- ✓ 5V Type-C power supply
- ✓ Wide voltage supply range of 9-36V
- ✓ Programmable control via MicroPython
- ✓ One CAN channel
- ✓ Two RS485 channels
- ✓ Control channel for two DC motors or one stepper motor
- ✓ One analog input interface (4-20mA or 0-10V optional)
- ✓ One SPI interface
- ✓ Two GPIO interfaces
- ✓ Two isolated PWM output interfaces

3 Connector Description

3.1 Terminal port location

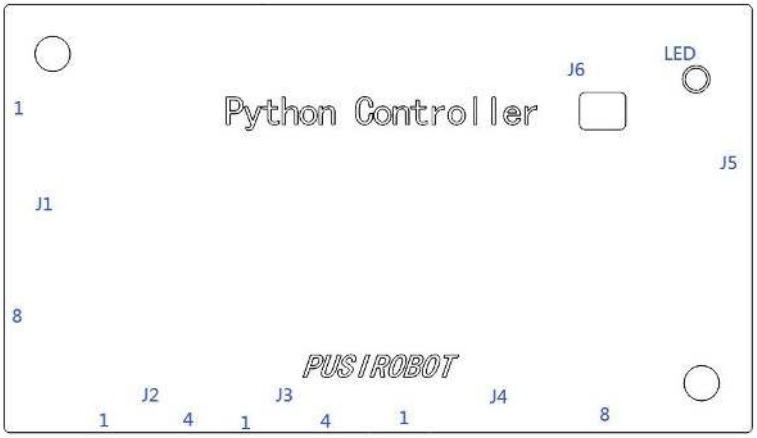


Figure 1

3.2 Connection J1

Pin No:	1	2	3	4	5	6	7	8
Designator	24+	GND	TX1	RX1	CANL	CANH	GPIO-1	GPIO-2

Description:

24+: Positive terminal for the controller's wide voltage power supply, 9-36V;

GND: Negative terminal for the controller's wide voltage power supply;

TX1: RS485 bus transmission signal 1, interface withstand voltage 5V;

RX1: RS485 bus receiving signal 1, interface withstand voltage 5V;

CANL: Connects to the CAN bus, interface withstand voltage 5V;

CANH: Connects to the CAN bus, interface withstand voltage 5V;

GPIO-1: General IO port 1, can be used for input or output, high level is 3.3V, interface withstand voltage 3.3V;

GPIO-2: General IO port 2, can be used for input or output, high level is 3.3V, interface withstand voltage 3.3V.

3.3 Motor connection J2

Pin No:	1	2	3	4
Designator	M1+	M1-	M2+	M2-

Description:

M1+: solenoid valve 1+ (DC brush motor 1+);

M1-: solenoid valve 1- (DC brush motor 1-);

M2+: solenoid valve 2+ (DC brush motor 2+);

M2-: solenoid valve 2- (DC brush motor 2-).

3.4 SPI connection J3

Pin No:	1	2	3	4
Designator	MISO	MOSI	SCK	CSN

Description:

MISO: Pin for the master device to send data, connected to the receiver of the slave device, interface withstand voltage 3.3V;

MOSI: Pin for the master device to receive data, connected to the transmitter of the slave device, interface withstand voltage 3.3V;

SCK: Clock signal, interface withstand voltage 3.3V;

CSN: Pin used to select the slave device, interface withstand voltage 3.3V.

3.5 Connection J4

Pin	1	2	3	4	5	6	7	8
Designator	PWM1	PWM2	AIN-	AIN+	RX2	TX2	GND	VDD5

Signal Description:

PWM1: PWM output 1;

PWM2: PWM output 2;

AIN-: Negative terminal for 4-20mA analog input;

AIN+: Positive terminal for 4-20mA / 0-10V analog input;

RX2: RS485 bus receiving signal 2, interface withstand voltage 5V;

TX2: RS485 bus transmission signal 2, interface withstand voltage 5V;

GND: Digital ground for the controller;

VDD5: 5V output from the controller.

3.6 Type-C connection J5

Connect to a computer via the Type-C interface for programming.

3.7 Reset button J6

Hold the button while connecting the device to the computer; the device will be recognized as a storage device.

4 Programming environment

4.1 Programming language

MicroPython is a complete software implementation of the Python 3 programming language, written in C and optimized to run on microcontrollers. It is a full Python compiler and runtime system that operates on microcontroller hardware. It provides an interactive prompt (REPL) for immediate command execution. In addition to the selected core Python libraries, MicroPython includes modules for accessing low-level hardware.

MicroPython strives to be as compatible as possible with standard Python (CPython), so if you know Python, you already know MicroPython. The more you understand MicroPython, the better you will perform in Python.

The examples presented here primarily rely on the 'machine' library in MicroPython, so for specific interpretations beyond the examples shown, please refer to the official documentation for function descriptions.

4.2 Programming software

MicroPython IDEs include Thonny, VSCode, and PyCharm, which can be downloaded and installed from their official websites. Thonny is specifically designed for MicroPython and can be used immediately after installation, while VSCode and PyCharm require additional plugins to provide MicroPython support.

4.3 Device connection

4.3.1 Connect PC

Connect the device to the computer using a Type-C data cable, then check the ports in Device Manager (see Figure 2).

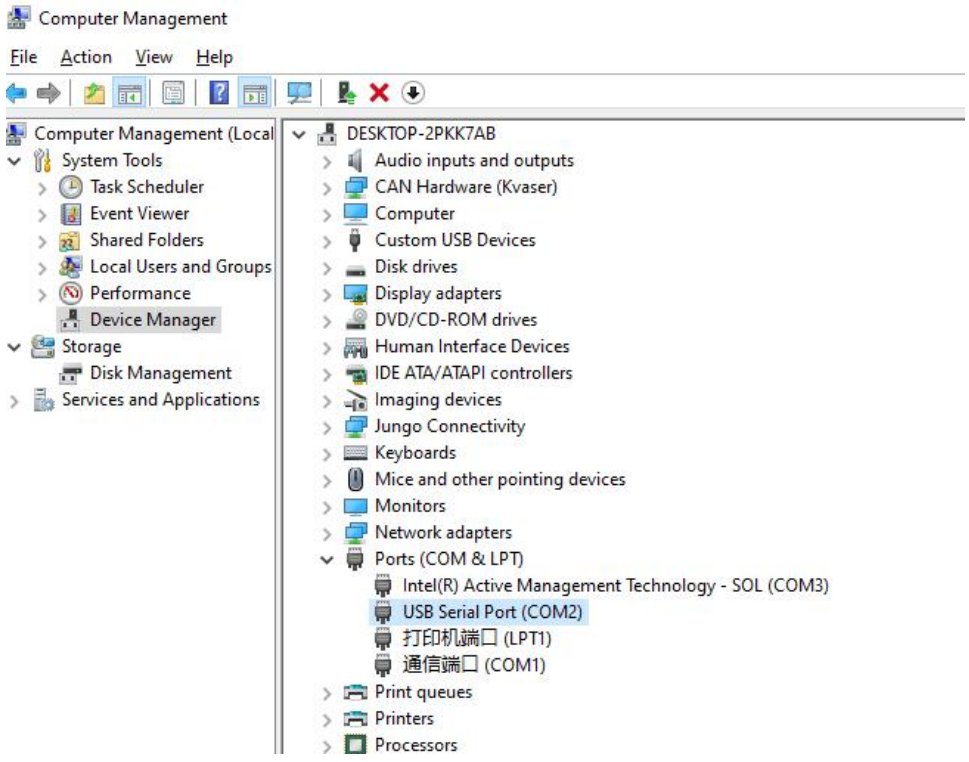


Figure 2

4.3.2 Thonny connection

Once the device is connected to the computer, open Thonny, click on the bottom right corner of the software, select the corresponding port for the device, and click to connect to COM18 (see Figure 3).

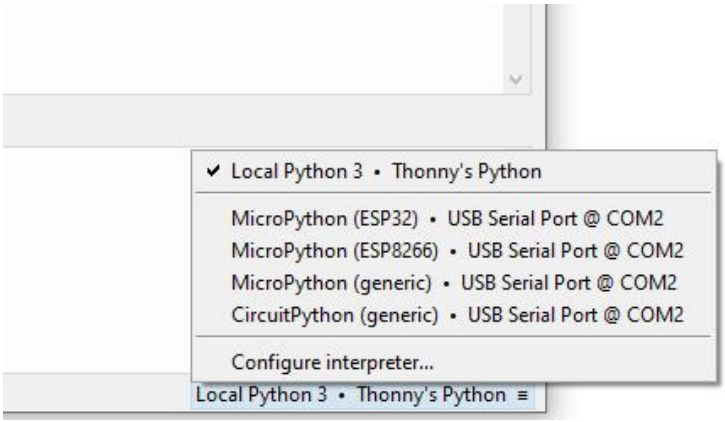


Figure 3

4.3.3 VSCode connection

Open the PyMakr plugin page, and in the DEVICES window, click to connect the device on the corresponding port: COM18 (see Figure 4).

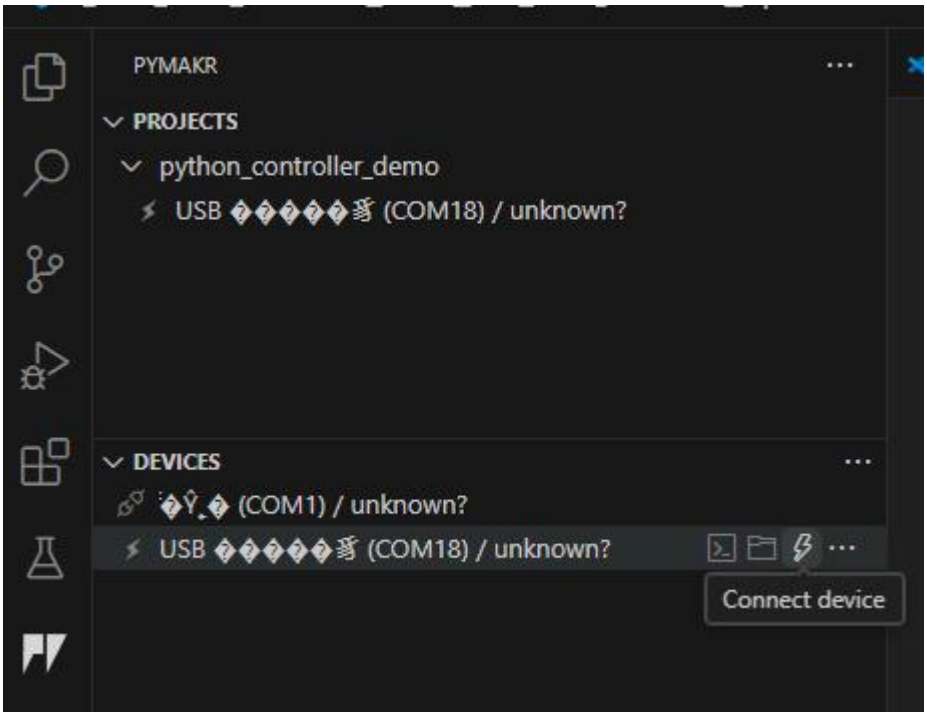


Figure 4

4.3.4 Pycharm connection

Install the MicroPython plugin and configure MicroPython settings (see Figure 5). PyCharm will automatically recognize the device and connect to it.

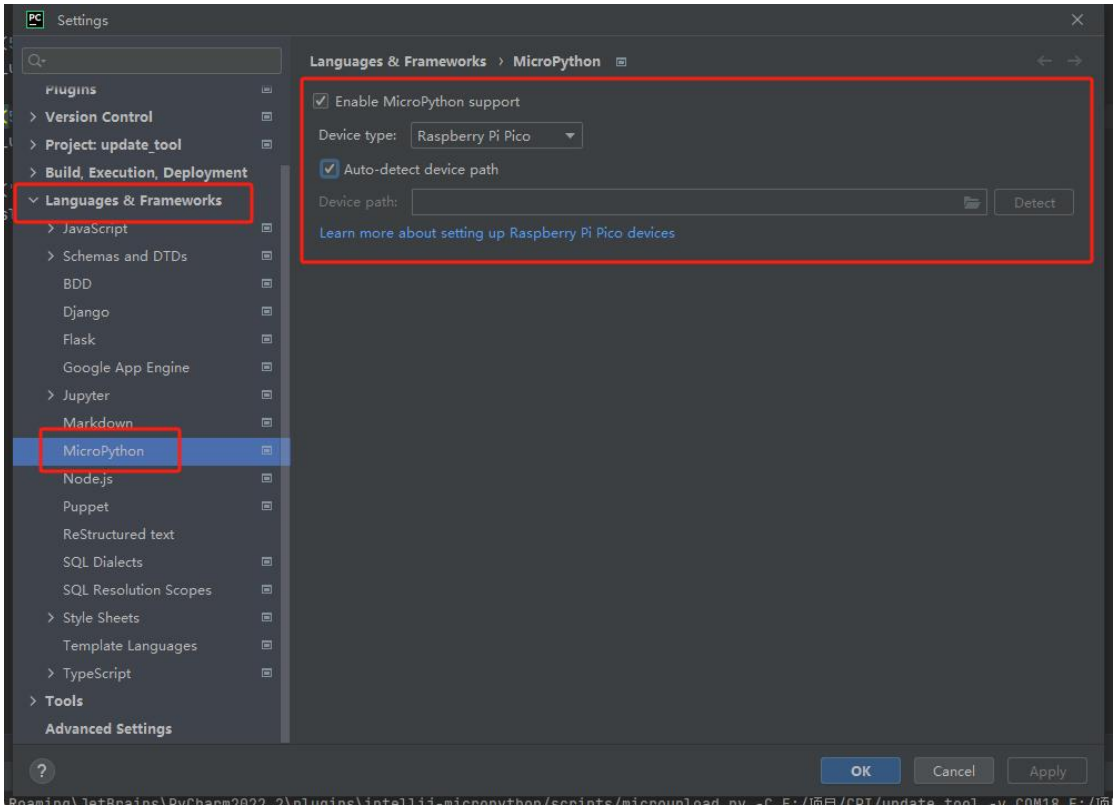


Figure 5

5 CANopen communication

The Python Controller has one CAN channel and includes a built-in CAN programming library, which makes it easy to handle CAN communication.

5.1 CANopen overview

CAL provides all the network management services and message transport protocols, but does not define the content of the objects or the types of objects being communicated (it only defines how, not what). This is the entry point for CANopen. CANopen is developed on the basis of CAL, utilizing a subset of CAL communication and service protocol, providing an implementation solution for distributed control systems. CANopen allows for functional expansion of nodes while ensuring interoperability of network nodes, whether simple or complex.

The core concept of CANopen is the Object Dictionary (OD), which is also used as a device description format in other fieldbus systems (such as Profibus and Interbus-S). CANopen communication can access all parameters of the drive through the Object Dictionary (OD). Note: The Object Dictionary is not part of CAL, but is implemented in CANopen. The Object Dictionary supported by the PMC006CX is shown in Appendix 1.

The CANopen communication model defines several types of messages (communication objects):

Abbreviation	Full Name	Description
SDO	Service Data Object	Used for non-time-critical data, such as parameters.
PDO	Process Data Object	Used for transmitting time-critical process data (e.g., set values, control words, status information).
SYNC	Synchronization Message	Used for synchronizing CAN nodes.
EMCY	Emergency Message	Used for transmitting emergency events from the drive.
NMT	Network Management	Used for managing the CANopen network.
Heartbeat	Error Control Protocol	Used to monitor the life status of all nodes.

5.2 CAN Frame Structure

CAN transmits data between the host (controller) and bus nodes through data frames. The table below shows the structure of the data frame.

Frame Header	Arbitration Field		Control Field	Data Field	Checksum Field	Acknowledgment Field	Frame Trailer
	COB-ID (Communication Object Identifier)	RTR (Remote Request)					
1 bit	11 or 29 bits	1 bit	6 bit	0~8bytes	16 bit	2bit	7bit

5.3 CAN connection

Using a CAN bus connection allows for a maximum transmission distance of 5000 meters. The diagram provides a network solution for connecting multiple CAN devices via the CAN bus, supporting up to 127 nodes.

Note: It is recommended to use shielded twisted pair cables specifically for CAN bus, with a 120-ohm terminating resistor connected at both ends of the twisted pair.

5.4 Programming

The Python Controller includes a built-in CAN programming library, pyccan, which makes it easy to implement CAN communication programming.

5.4.1 CAN Initialization

def __init__(self, bitrate)

bitrate is the baud rate, measured in kbps.

Example:

```
import pyccan

# Initialize CAN and set baud rate, in kbps
can = pyccan.pyccan(500)
```

5.4.2 CAN Data Transmission

```
def send(self, node_id, data_count, data) -> None
```

node_id is the node ID, data_count is the data length, and data is the actual data to be sent.

Example:

```
def send_data():
    data = [0x40, 0x0C, 0x60, 0x00, 0x01, 0x02, 0x03, 0x04]
    can.send(0x0603, 8, data)
```

```
def check_message(self) -> bool, int
```

This method checks whether any CAN data has been received, returning whether data exists and the data buffer channel.

Example:

```
def check_message(self):
    # Data reception monitoring: 0: Message in RXB0, 1: Message in RXB1, 2: No message
    self.cs.value(0)
    self.spi.write(bytearray([0xB0]))
    status = self.spi.read(1)[0]
    self.cs.value(1)
    if (status & (1 << 6)) != 0:
        return True, 0
    if (status & (1 << 7)) != 0:
        return True, 1
    return False, None
```

5.4.3 CAN Data Reception

```
def receive(self, channel) -> str, list[str]
```

channel is the data buffer channel, with valid parameters being 0 or 1. The return parameters are str for the frame ID and list[str] for CAN data.

Example:

```
def receive(self, channel):
data_cmd = 0xFF
meta_cmd = 0xFF
if channel == 0:
```

```
        data_cmd = 0x92
        meta_cmd = 0x90
    elif channel == 1:
        data_cmd = 0x96
        meta_cmd = 0x94

    self.cs.value(0)
    self.spi.write(bytearray([meta_cmd]))
    meta_data = self.spi.read(5)
    self.cs.value(1)
    frame_id = 0x0000
    frame_id &= ~(0xFF << 3)
    frame_id |= (meta_data[0] & 0xFF) << 3
    frame_id &= ~0x07
    frame_id |= ((meta_data[1] >> 5) & 0x07 & 0x07)

    data_count = int(meta_data[4])
    self.cs.value(0)
    self.spi.write(bytearray([data_cmd]))
    data = self.spi.read(data_count)
    self.cs.value(1)
    result = [0x00] * data_count
    for i in range(data_count):
        result[i] = hex(data[i])
    return hex(frame_id), result
```

5.4.4 CAN Receive Interrupt

```
def receive_interrupt_init(self) -> None
```

This method enables the CAN receive interrupt.

Example:

```
def receive_by_interrupt(pin):
    # After triggering the interrupt, first check if the RX channel has values, read until complete, then exit the interrupt.
    while 1:
    recive_message, channel = can.check_message()
    if recive_message:
        frame_id, received_msg = can.receive(channel)
        print("frame_id: ", frame_id, " Received message:", received_msg)
    else:
        break

can.recive_interrupt_init()
```

```
rx_interrupt_pin = machine.Pin(22, machine.Pin.IN)
rx_interrupt_pin.irq(trigger=machine.Pin.IRQ_FALLING, handler=recive_by_interrupt)
```

6 RS485 communication

The Python Controller provides two RS485 channels, and RS485 communication can be implemented through MicroPython programming. Initially, you can monitor the command sending and response situation using a serial assistant.

```
from machine import UART, Pin
import time

# RS485-2
# Configure UART1
uart = UART(1, baudrate=9600, tx=Pin(4), rx=Pin(5))    # Using UART1, TX pin is GPIO 4, RX pin is GPIO 5
de_re = Pin(3, Pin.OUT)

# Switch to transmit mode
def enable_transmit():
    de_re.value(1)    # High level to enable transmit mode

# Switch to receive mode
def enable_receive():
    de_re.value(0)    # Low level to enable receive mode

# Send RS-485 message
def send_rs485_message(message):
    enable_transmit()    # Enable transmit mode
    time.sleep(0.01)    # Wait briefly to ensure mode switch
    uart.write(message)    # Send message via UART
    time.sleep(0.01)    # Wait for message to complete sending
    enable_receive()    # Switch back to receive mode after sending
    count = 0
    while not uart.any():
        if count > 20:
            break
        time.sleep(0.001)
        count += 1
    return uart.read()

# Main loop, periodically send messages
while True:
    res = send_rs485_message('AABBCC\n')    # Send message
```

```
print("Message sent! Received: ", res)
time.sleep(1)    # Send once every second
```

7 SPI communication

7.1 Overview

SPI (Serial Peripheral Interface) is a full-duplex synchronous serial bus developed by Motorola for communication between microcontroller units (MCUs) and peripheral devices. It is primarily used to interface with EEPROMs, Flash memory, real-time clocks (RTCs), analog-to-digital converters (ADCs), network controllers, MCUs, digital signal processors (DSPs), and digital signal decoders. The SPI system can directly interface with a variety of standard peripheral devices from different manufacturers and generally uses four lines: serial clock line (SCK), master input/slave output data line (MISO), master output/slave input data line (MOSI), and active-low slave select line (SSEL).

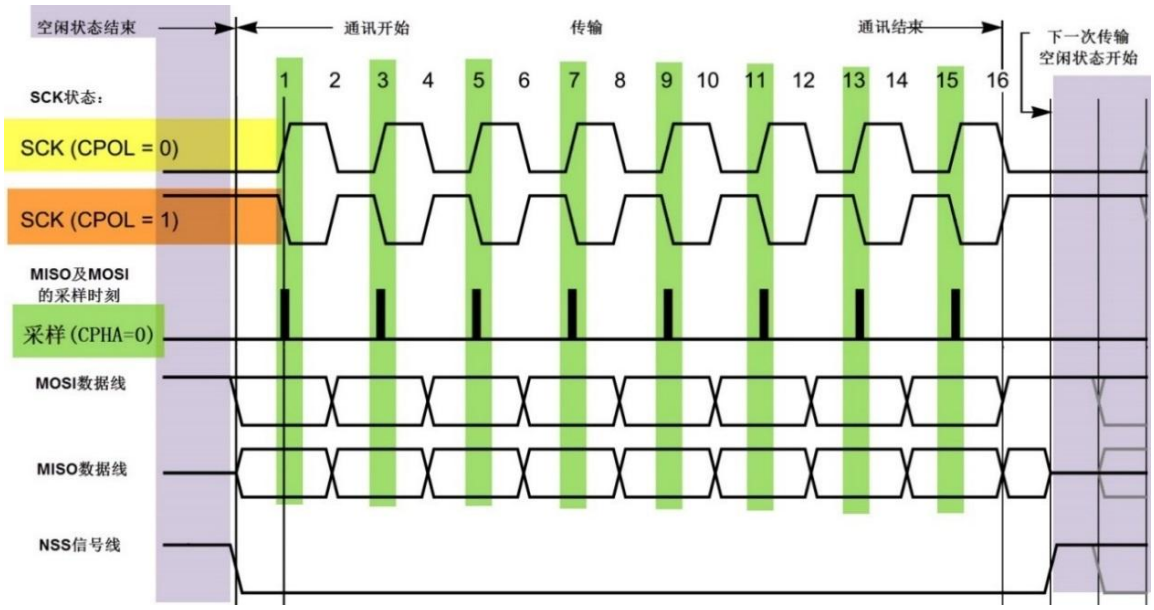
7.2 SPI Communication Lines

- SCK (Serial Clock): Provides the clock signal, controlled by the master. Data transmission relies on the SCK clock signal.
- MOSI (Master Out Slave In): Master output, slave input. Used for sending data from the master device to the slave device.
- MISO (Master In Slave Out): Slave output, master input. Used for sending data from the slave device to the master device.
- SS (Slave Select): Chip select signal used to select the slave device. Each slave device has an independent SS line, which is pulled low to select the corresponding slave device for communication.

7.3 Communication Process

This describes the communication timing for the master. NSS, SCK, and MOSI signals are all generated by the master, while the MISO signal is generated by the slave.

The high bit is sent first, followed by the low bit. After sending one byte, the next byte can be sent without requiring an acknowledgment. The NSS signal is used to select the slave device as a start/stop signal.



7.4 Programming

The Python Controller provides an SPI communication interface, which can be implemented through MicroPython programming.

Example:

```
from machine import Pin, SPI
import time

# Configure SPI pins
# SPI settings
spi = SPI(0, baudrate=1000000, polarity=0, phase=0, sck=Pin(18), mosi=Pin(19), miso=Pin(16))

# CS pin settings
cs = Pin(17, Pin.OUT)

# Pull CS low to start communication
def select():
    cs.value(0)

# Pull CS high to end communication
def deselect():
    cs.value(1)

# Send command and read response
def send_command(cmd):
    select()
```

```
        tx_buffer = bytearray([cmd])
        rx_buffer = bytearray(1)    # Receive buffer length is 1 byte
        spi.write_readinto(tx_buffer, rx_buffer)
        deselect()
        return rx_buffer

# Test communication by sending different commands and receiving responses
while True:
        command = 0x03    # Send command 0x03
        response = send_command(command)
        print(f"Sent command 0x{command:02X}, Received: 0x{response[0]:02X}")

        command = 0x29    # Send command 0x29
        response = send_command(command)
        print(f"Sent command 0x{command:02X}, Received: 0x{response[0]:02X}")

        command = 0xFF    # Send command 0xFF
        response = send_command(command)
        print(f"Sent command 0x{command:02X}, Received: 0x{response[0]:02X}")

        time.sleep(5)
```

8 Motor drive

8.1 Overview

The Python Controller provides control channels for two DC motors or solenoids, or one stepper motor. You can adjust the output levels of the two ports through code to change the direction, and adjust the output frequency to modify the running speed of the DC motors or the power supply ratio for the solenoids.

8.2 Programming

Realize the motor motion control by MicroPython programming.

Example:

```
def set_direction(motor_number, direction):
        if motor_number == 1:
                if direction == 1:
                        in2 = Pin(7, Pin.OUT)
                        in2.value(0)
                        vref_pwm = PWM(Pin(8, Pin.OUT))
                        vref_pwm.freq(2000)
                        vref_pwm.duty_u16(65535)
```

```
                in1_pwm = PWM(Pin(6, Pin.OUT))
                in1_pwm.freq(5000)
                return in1_pwm
            elif direction == 0:
                in1 = Pin(6, Pin.OUT)
                in1.value(0)
                vref_pwm = PWM(Pin(8, Pin.OUT))
                vref_pwm.freq(2000)
                vref_pwm.duty_u16(65535)
                in2_pwm = PWM(Pin(7, Pin.OUT))
                in2_pwm.freq(5000)
                return in2_pwm
duty_cycle = int(50 * 65535 / 100)
# Begins to specify the speed step
def start_motor(pwm, speed):
    pwm.duty_u16(int(speed * 65535 / 100))

try:
    pwm = set_direction(1, 1)
    start_motor(pwm, 40)
    time.sleep(5)
    start_motor(pwm, 0)
    time.sleep(0.1)
    pwm = set_direction(1, 0)
    start_motor(pwm, 100)
    time.sleep(5)
    start_motor(pwm, 0)
except KeyboardInterrupt:
    print("stop")
    start_motor(pwm, 0)
```

9 Analog Input

The Python Controller provides an analog input interface with an input range of 4-20mA / 0-10V. You can read the input values using MicroPython programming. In practical applications, this value is typically used in calculations to facilitate back-and-forth operations between two points or to set corresponding speed values.

Wiring: Connect the positive and negative outputs to AIN+ and AIN-, respectively.

Example:

```
from machine import ADC, Pin
import time

# Initialize ADC object, connected to GPIO 28 (corresponding to ADC0 channel)
adc_pin = ADC(Pin(28))
```

```
# Read analog value and convert to voltage
conversion_factor = 10 / 65535    # Reference voltage is 0-10V, ADC has 16-bit resolution

while True:
    # Read ADC value (range 0 to 65535)
    adc_value = adc_pin.read_u16()

    # Convert ADC value to voltage
    voltage = adc_value * conversion_factor

    # Output ADC value and voltage; actual voltage should be calculated based on real pressure and PICO receiving voltage relationship
    print("ADC Value: ", adc_value, " Voltage: ", voltage, "V")

    time.sleep(1)
```

10 PWM output

10.1 Overview

The Python Controller provides two PWM output interfaces. Connect the PWM+ of the external device to the PWM1 or PWM2 pin on J4, and connect PWM- to the GND pin on J4. You can then achieve the desired PWM output control through MicroPython programming.

10.2 Programming

The Python Controller offers PWM output interfaces that allow for specifying frequency and duty cycle through MicroPython programming.

Example:

```
import machine
from machine import Pin
import time

pwm1 = machine.PWM(machine.Pin(20))
pwm2 = machine.PWM(machine.Pin(21))

pwm1.freq(5000)
pwm1.duty_u16(30000)

pwm2.freq(5000)
pwm2.duty_u16(30000)

while 1:
```

```
print("----")
time.sleep(3)
```

11 GPIO

The Python Controller provides two GPIO interfaces. You can control them as input or output interfaces through MicroPython programming. As an input, the voltage range is 0-3.3V; as an output, it can output either 3.3V or 0V. An output cannot be connected to GND or another output.

Example:

```
from machine import ADC, Pin
import time

# Configure GPIO pin 1 as input
pin1 = Pin(26, Pin.IN)
# Read and print the value of pin 26
print(pin1.value())
# Configure GPIO pin 2 as output
pin2 = Pin(27, Pin.OUT, Pin.PULL_UP)
# Set GPIO pin 2 output to low
pin2.value(0)
# Set GPIO pin 2 output to high
pin2.value(1)
```

12 Electrical Characteristics

Parameter	Condition	Min	Typical	Max	Unit
Input voltage	25℃	9	24	36	V
Temperature	24V input voltage	-20		85	℃
IO max current	Sourcing/sinking	2	8	16	mA
Output current per phase	25℃	0	2	3	A

13 Dimensions (Unit: mm)