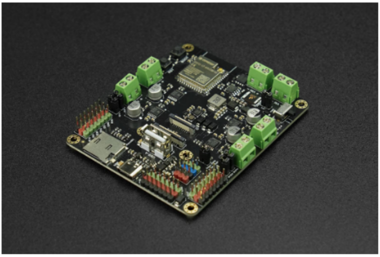
search
DFRobot Products (DFR) Index
Sensor (SEN) Index
Telecom Module (TEL) Index
Motor Driver (DRI) Index
Fitting (FIT) Index
Robot (ROB) Index
micro:bit (MBT) Index
Kit (KIT) Index
Toy (TOY) Index
Servos (SER) Index
What is XX series
How to select series
How it works series
PinPong Library

Introduction
Features
Specification
Board Overview
Pin Diagram
Sample code
Drive car practice
FAQ
More Documents

SKU:DFR0994



Introduction

The Romeo ESP32-S3 development board is a high-performance board meticulously designed for robotic projects. Embracing the ESP32-S3-WROOM-1U-N16R8 module, this board bestows formidable computational prowess and wireless communication capabilities upon robotic ventures. Notably, the Romeo ESP32-S3 is also equipped with a 4-channel 2.5A DC motor driver, eliminating the need for an additional motor driver board and enabling effortless robotic project creation. Additionally, this development board features a high-quality OV2640 camera, boasting a resolution of 2 million pixels and a 68° field of view. With support for resolutions of up to 1600*1200, it furnishes the robotic endeavor with a potent visual input device. Within the confines of the ESP32-S3-WROOM-1U-N16R8 module resides 16MB Flash and 8MB PSRAM, enabling ample storage for code and data. The integrated ESP32-S3 chip empowers the robot with formidable neural network computation and signal processing capabilities, elevating its cognitive faculties. The Romeo ESP32-S3 supports dual-mode communication, encompassing Wi-Fi and Bluetooth 5 (LE), thereby facilitating wireless robot control and image transmission functionalities. This board can be programmed through diverse environments, including Arduino IDE, ESP-IDF, and MicroPython, rendering hardware manipulation effortless with both C language and Python.

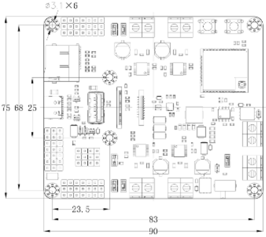
** Note: Before burning code, press and hold BOOT, click the reset button, and release BOOT to enter download mode. **

Features

- A four-channel 2.5A high-current H-bridge motor driver
- Supporting both PH/EN control mode and PWM control mode
- Multiple 5V power outputs, facilitating servo motor control
- An autonomous camera power circuit
- On-board GDI display, DVP camera, USB, and MicroSD card interfaces
- Featuring the on-board ESP32-S3-1U-N16R8 (16MB FLASH / 8MB PSRAM)
- Enabling dual-mode communication through Wi-Fi and Bluetooth 5 (LE).

Specification

Basic Parameters



- Type-C Input Voltage: 5V
- Operating Voltage: 3.3V
- VIN Input Voltage: 7-24V
- VM Input Voltage: 5-24V
- Maximum Output Current of 5V Power Supply: 2A
- Motor Drive Capacity: 2.5A
- Operating Temperature: 0 to 60℃
- Dimension: 25.4x60mm/2.95x3.54"

Hardware Information

- Processor: Xtensa® dual-core 32-bit LX7 microprocessor
- Main Frequency: 240 MHz
- SRAM: 512KB
- ROM: 384KB
- Flash: 16MB
- PSRAM: 8MB
- RTC SRAM: 16KB
- USB: USB 2.0 OTG full-speed

WIFI

- WIFI Protocol: IEEE 802.11b/g/n
- Bandwidth: Support 20 MHz and 40 MHz at 2.4 GHz band
- WIFI Mode: Station, SoftAP, SoftAP+Station combined mode
- WIFI Frequency: 2.4GHz
- Frame Aggregation: TX/RX A-MPDU, TX/RX A-MSDU

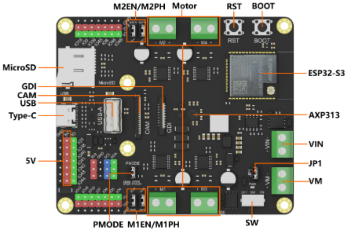
蓝牙

- Bluetooth Protocol: Bluetooth 5, Bluetooth mesh
- Bluetooth Frequency: 125 Kbps, 500 Kbps, 1 Mbps, 2 Mbps

Ports

- Digital I/O x27
- LED PWM Controller 8 Channels
- SPI x2
- UART x3
- I2C x2
- I2S x2
- IR Transceiver: transmit channel x5, receive channel x5
- 2×12-bit SAR ADC, 20 Channels
- DMA Controller: transmit channel x5, receive channel x5

Board Overview



- **Type-C**： Type-C USB port
- **5V**： 5V power output
- **VIN**： 7-24V, MCU system power input
- **VM**： 5-24V, Motor power input
- **JP1**： Power switch jumper cap, controller and motor share power when short-circuited
- **SW**： Power control switch
- **PMODE**: Motor drive mode Jumper cap, PH/EN control mode when short-circuited, PWM control mode when disconnected
- **Motor**: Dc motor interface
- **ESP32-S3 module**： ESP32-S3 module launched by Espressif
- **M1EN/M1PH**： M1 motor drive jumper
- **M2EN/M2PH**： M2 motor drive jumper
- **RST**： reset button
- **BOOT**： BOOT button
- **AXP313A**： Power management chip
- **GPI**:GDI display interface
- **CAM**:DVP camera interface
- **USB**： USB interface, can be connected to USB peripherals
- **MicroSD**： TF card slot

Pin Diagram

GPIO Allocation Table

Num	Pin	GDI	CAM	Motor	Other
1	GND				
2	3V3				
3	EN				
4	IO4		D5		
5	IO5		PCLK		
6	IO6		VSYNC		
7	IO7	FCS	D6		
8	IO15	MOSI			
9	IO16	MISI			
10	IO17	SCLK			
11	IO18	LCD_CS			
12	IO8		D7		
13	IO19				USB D-
14	IO20				USB D+
15	IO3	LCD_DC			
16	IO46		D8		
17	IO9			M3_EN/IN1	

18	IO10			M3_PH/IN2	
19	IO11			M4_PH/IN2	
20	IO12	TCS		M1_EN/IN1	
21	IO13	INT		M1_PH/IN2	
22	IO14	BUSY		M2_EN/IN1	
23	IO21	LCD_BL		M2_PH/IN2	
24	IO47			M4_EN/IN1	
25	IO48		D9		
26	IO45		XMCLK		
27	IO0	SDCS			Boot Key
28	IO35				
29	IO36				
30	IO37				
31	IO38	LCD_RST			
32	IO39		D2		
33	IO40		D3		
34	IO41		D4		
35	IO42		HREF		
36	IO44				RX
37	IO43				TX
38	IO2	SCL	SCL		SCL
39	IO1	SDA	SDA		SDA
40	GND				

Motor Pin

PINS	Romeo ESP32-S3 PINS
M1_EN/IN1	12
M1_PH/IN2	13
M2_EN/IN1	14
M2_PH/IN2	21
M3_EN/IN1	9
M3_PH/IN2	10
M4_EN/IN1	47
M4_PH/IN2	11

GDI Display Interface

This interface is a DFRbot dedicated GDI display interface for connecting a screen using a 18pin-FPC wire.

The pin list for using GDI camera interface is shown below:

FPC PINS	Romeo ESP32-S3 PINS	Description
VCC	3V3	3.3V
LCD_BL	21	Backlight
GND	GND	GND
SCLK	17/SCK	SPI clock
MOSI	15/MOSI	Host output, slave input
MISO	16/MISO	Host input, slave output
LCD_DC	3	Data/command
LCD_RST	38	Reset
LCD_CS	18	TFT Chip Select
SD_CS	0	SD card chip select

FCS	7	Font library chip select
TCS	12	Touch chip select
SCL	2	I2C clock
SDA	1	I2C data
INT	13	INT
BUSY	14	Tearproof pins
X1	NC	custom pin 1
X2	NC	custom pin 2

When using FPC to connect the screen, please configure the corresponding pin numbers according to the GDL demo. Normally, only three pins need to be configured on different main controllers.

Displays that support GDI:

- 1.54" 240x240 IPS wide viewing angle TFT display
- 1.8" 128x160 IPS TFT LCD Display
- 2.0" 320x240 IPS wide viewing angle TFT display
- 2.8" 320x240 IPS TFT resistive touch display
- 3.5" 480x320 IPS TFT capacitive touch display
- 1.51" OLED Transparent Display with Converter

CAM接口

The CAM interface is compatible with both OV2640 and OV7725 camera. Enable AXP313A power output when using a camera. Click to download AXP313A library(ESP-IDF and MicroPython drivers are included).

The pin list for using DVP camera interface is shown below.

CAM PINS	FireBeetle ESP32-S3 PINS	Description
NC	NC	NC
AGND	/	Analog GND
SDA	1/SDA	I2C data
AVDD	/	AXP313A Controllable Power
SCL	2/SCL	I2C Clock
RST	/	Pulled up to DOVDD
VSYNC	6/A2	Frame sync signal
PWDN	/	Pulled down
HREF	42	Horizontal sync signal
DVDD	/	AXP313A Controllable Power
DOVDD	/	AXP313A Controllable Power
D9	48	DATA 9
XMCLK	45	Clock signal
D8	46	DATA 8
DGND	GND	Digital GND

D7	8/A3	DATA 7
PCLK	5/A1	Pixel Clock signal
D6	7/D5	DATA 6
D2	39	DATA 2
D5	4/A0	DATA 5
D3	40	DATA 3
D4	41	DATA 4
NC	NC	NC
NC	NC	NC

Pin connection

Connect the pin 1 of the camera to the white dot position

Sample code

See the FireBeetle ESP32-S3 tutorial for the basic tutorial.

EN/PH mode drives the motor

*PH/EN Control mode Truth Table *

EN	PH	OUT1	OUT2	说明
0	X	L	L	Braking (low side slow attenuation)
1	0	L	H	retreat (OUT2 → OUT1)
1	1	H	L	advance (OUT1 → OUT2)

Drive dc motor

```
/**
 *@brief motorSpeeddetectionbyPH_EN.ino
 *@ Motor driver in PH/EN mode
 */
#include "driver/mcpwm.h"
#include "soc/mcpwm_struct.h"
#include "soc/mcpwm_reg.h"

//IO12->M1_IN1
//IO13->M1_IN2
#define MOTOR_STEP_PIN 12
#define MOTOR_DIRECTION_PIN 13

//Initialize the pins needed to generate the PWM signal
void mcpwm_init(void)
{
/**
 * @brief This function initializes each gpio signal for MCPWM
 *        @note
 *        This function initializes one gpio at a time.
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param io_signal set MCPWM signals, each MCPWM unit has 6 c
 *                  'X' is timer_num(0-2)
 * @param gpio_num set this to configure gpio for MCPWM, if yo
 *
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0A,MOTOR_STEP_PIN);//The b
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0B,MOTOR_DIRECTION_PIN);
```

```
  //Configure mcpwm information
  mcpwm_config_t pwm_config;
  pwm_config.frequency = 1000;/*!<Set frequency of MCPWM in Hz
  pwm_config.cmpr_a = 0;/*!<Set % duty cycle for operator a(MC
  pwm_config.cmpr_b = 0;/*!<Set % duty cycle for operator b(MC
  pwm_config.counter_mode/*!<Set  type of MCPWM counter*/ = MC
  pwm_config.duty_mode/*!<Set type of duty cycle*/ = MCPWM_DUT

/**
 * @brief Initialize MCPWM parameters
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param mcpwm_conf configure structure mcpwm_config_t
 *
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_init(MCPWM_UNIT_0,MCPWM_TIMER_0,&pwm_config);//Initial
}

void advance(/*range: 0~100*/uint8_t speed)
{

/**
 * @brief Use this function to set MCPWM signal high
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param gen set the operator(MCPWMXA/MCPWMXB), 'x' is timer
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B

/**
 * @brief Set duty either active high or active low(out of pha
 *        @note
 *        Call this function every time after mcpwm_set_signal
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param gen set the generator(MCPWMXA/MCPWMXB), 'x' is opera
 * @param duty_type set active low or active high duty type
 *
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,M

/**
 * @brief Set duty cycle of each operator(MCPWMXA/MCPWMXB)
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param gen set the generator(MCPWMXA/MCPWMXB), 'X' is opera
 * @param duty set duty cycle in %(i.e for 62.3% duty cycle, d
 *
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,speed)
}

void retreat(/*range: 0~100*/uint8_t speed)
{
  //Give MOTOR_STEP_PIN a continuous low level
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B)

  //The MOTOR_DIRECTION_PIN pin outputs a PWM wave with a duty
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,M
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,speed)
```

```
}

void breake(void)
{
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A)
}



void setup() {
  mcpwm_init();
}

void loop() {
  advance(70);
  delay(2000);
  retreat(30);
  delay(2000);
  breake();
  delay(2000);
}
```

Drive DC motor with encoder

The motor rotation speed is controlled by PID

```
/**
 *@brief motorSpeeddetectionbyPH_ENwithencoder.ino
 *@ Motor driver in PWM mode (with encoder)
 */
#include "driver/mcpwm.h"
#include "soc/mcpwm_struct.h"
#include "soc/mcpwm_reg.h"
#include <PID_v1.h>

//IO12->M1_IN1
//IO13->M1_IN2
#define MOTOR_STEP_PIN 12
#define MOTOR_DIRECTION_PIN 13

const byte encoder0pinA = 3;//A pin -> the interrupt pin 3
const byte encoder0pinB = 4;//B pin -> the digital pin 4
byte encoder0PinALast;
double duration,abs_duration;//the number of the pulses
boolean Direction;//the rotation direction
boolean result;
double val_output;//Used to provide PWM power values to the mo
double Setpoint;
double Kp=0.6, Ki=5, Kd=0;
PID myPID(&abs_duration, &val_output, &Setpoint, Kp, Ki, Kd, D

//Initialize the pins needed to generate the PWM signal
void mcpwm_init(void)
{
/**
 * @brief This function initializes each gpio signal for MCPWM
 *        @note
 *        This function initializes one gpio at a time.
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param io_signal set MCPWM signals, each MCPWM unit has 6 o
 *                 'X' is timer_num(0-2)
 * @param gpio_num set this to configure gpio for MCPWM, if yo
 *
 * @return
 *      - ESP_OK Success
 *      - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0A,MOTOR_STEP_PIN);//The b
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0B,MOTOR_DIRECTION_PIN);

  //Configure mcpwm information
  mcpwm_config_t pwm_config;
  pwm_config.frequency = 1000;/*!<Set frequency of MCPWM in Hz
```

```
  pwm_config.cmpr_a = 0;/*!<Set % duty cycle for operator a(MC
  pwm_config.cmpr_b = 0;/*!<Set % duty cycle for operator b(MC
  pwm_config.counter_mode/*!<Set  type of MCPWM counter*/ = MC
  pwm_config.duty_mode/*!<Set type of duty cycle*/ = MCPWM_DUT

/**
 * @brief Initialize MCPWM parameters
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param mcpwm_conf configure structure mcpwm_config_t
 *
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_init(MCPWM_UNIT_0,MCPWM_TIMER_0,&pwm_config);//Initial
}

void advance(/*range: 0~100*/uint8_t speed)
{
/**
 * @brief Use this function to set MCPWM signal high
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param gen set the operator(MCPWMXA/MCPWMXB), 'x' is timer
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B

/**
 * @brief Set duty either active high or active low(out of pha
 *        @note
 *        Call this function every time after mcpwm_set_signal
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param gen set the generator(MCPWMXA/MCPWMXB), 'x' is opera
 * @param duty_type set active low or active high duty type
 *
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,M

/**
 * @brief Set duty cycle of each operator(MCPWMXA/MCPWMXB)
 *
 * @param mcpwm_num set MCPWM unit(0-1)
 * @param timer_num set timer number(0-2) of MCPWM, each MCPWM
 * @param gen set the generator(MCPWMXA/MCPWMXB), 'X' is opera
 * @param duty set duty cycle in %(i.e for 62.3% duty cycle, d
 *
 * @return
 *     - ESP_OK Success
 *     - ESP_ERR_INVALID_ARG Parameter error
 */
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,speed)
}

void retreat(/*range: 0~100*/uint8_t speed)
{
  //Give MOTOR_STEP_PIN a continuous low level
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B)

  //The MOTOR_DIRECTION_PIN pin outputs a PWM wave with a duty
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,M
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,speed)
}

void breake(void)
{
```

```
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A)
//  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_
}

void wheelSpeed()
{
  int Lstate = digitalRead(encoder0pinA);
  if((encoder0PinALast == LOW) && Lstate==HIGH)
  {
    int val = digitalRead(encoder0pinB);
    if(val == LOW && Direction)
    {
      Direction = false; //Reverse
    }
    else if(val == HIGH && !Direction)
    {
      Direction = true;  //Forward
    }
  }
  encoder0PinALast = Lstate;

  if(!Direction)
    duration++;
  else
    duration--;
}

void EncoderInit()
{
  Direction = true;//default -> Forward
  pinMode(encoder0pinB,INPUT);
  attachInterrupt(digitalPinToInterrupt(encoder0pinA), wheelSp
}

void setup() {
  Serial.begin(9600);//Initialize the serial port
  mcpwm_init();
  advance(70);
  Setpoint =80;  //Set the output value of PID
  myPID.SetMode(AUTOMATIC);//Set PID to automatic mode
  myPID.SetSampleTime(100);//Set the PID sampling frequency to
  EncoderInit();//Initialize the module
}

void loop() {
  abs_duration=abs(duration);
  result=myPID.Compute();//The return value of PID conversion
  if(result)
  {
    Serial.print("Pluse: ");
    Serial.println(duration);
    duration = 0; //Count clear and wait for the next count
  }
}
```

PWM mode drives the motor

Truth table of PWM control mode

IN1	IN2	OUT1	OUT2	explain
0	0	Hi-Z	Hi-Z	Taxiing (H-bridge high impedance)
0	1	L	H	retreat (OUT2 → OUT1)
1	0	H	L	advance (OUT1 → OUT2)
1	1	L	L	Braking (low side slow attenuation)

Drive dc motor

```
/**
 *@brief motorSpeeddetectionbyPWM.ino
 *@ Motor driver in PWM mode
 */
#include "driver/mcpwm.h"
#include "soc/mcpwm_struct.h"
#include "soc/mcpwm_reg.h"

//IO12->M1_IN1
//IO13->M1_IN2
#define MOTOR_PWM_IN1 12
#define MOTOR_PWM_IN2 13

//Initialize the pins needed to generate the PWM signal
void mcpwm_init(void)
{
  // Binding needs to output the PWM pin to the PWM channel
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0A,MOTOR_PWM_IN1);
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0B,MOTOR_PWM_IN2);

  //Configure mcpwm information
  mcpwm_config_t pwm_config;
  pwm_config.frequency = 1000;/*!<Set frequency of MCPWM in Hz
  pwm_config.cmpr_a = 0;/*!<Set % duty cycle for operator a(MC
  pwm_config.cmpr_b = 0;/*!<Set % duty cycle for operator b(MC
  pwm_config.counter_mode/*!<Set  type of MCPWM counter*/ = MC
  pwm_config.duty_mode/*!<Set type of duty cycle*/ = MCPWM_DUT

  //Initialize a unit of mcpwm and bind the clock
  mcpwm_init(MCPWM_UNIT_0,MCPWM_TIMER_0,&pwm_config);
}

void advance(/*range: 0~100*/uint8_t speed)
{
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,M
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B)
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,speed)
}

void retreat(/*range: 0~100*/uint8_t speed)
{
  //Give MOTOR_DIRECTION_PIN a continuous low level
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A)

  /*
   * Call function mcpwm_set_duty_type() every time after mcpw
   * or mcpwm_set_signal_low() to resume with previously set d
   */
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B,M

  //The MOTOR_STEP_PIN pin outputs a PWM wave with a duty cycl
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B,speed)
}

void stop(void)
{
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A)
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B)
}

void breake(void)
{
  mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A
  mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B
}

void setup() {
  mcpwm_init();
}

void loop() {
  advance(30);
  delay(2000);
  retreat(60);
  delay(2000);
  stop();
```

```
  delay(2000);
  advance(30);
  delay(2000);
```

Drive DC motor with encoder

The motor rotation speed is controlled by PID

```
#define MOTOR_PWM_IN2 13

const byte encoder0pinA = 3;//A pin -> the interrupt pin 3
const byte encoder0pinB = 4;//B pin -> the digital pin 4
byte encoder0PinALast;
double duration,abs_duration;//the number of the pulses
boolean Direction;//the rotation direction
boolean result;
double val_output;//Used to provide PWM power values to the m
double Setpoint;
double Kp=0.6, Ki=5, Kd=0;
PID myPID(&abs_duration, &val_output, &Setpoint, Kp, Ki, Kd, D

//Initialize the pins needed to generate the PWM signal
void mcpwm_init(void)
{
  //The binding needs to output the PWM pin to the PWM channel
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0A,MOTOR_PWM_IN1);
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0B,MOTOR_PWM_IN2);

  //Configure mcpwm information
  mcpwm_config_t pwm_config;
  pwm_config.frequency = 1000;/*!<Set frequency of MCPWM in Hz
  pwm_config.cmpr_a = 0;/*!<Set % duty cycle for operator a(MC
  pwm_config.cmpr_b = 0;/*!<Set % duty cycle for operator b(MC
  pwm_config.counter_mode/*!<Set  type of MCPWM counter*/ = MC
  pwm_config.duty_mode/*!<Set type of duty cycle*/ = MCPWM_DUT

  // Initializes a unit of mcpwm and binds the clock
  mcpwm_init(MCPWM_UNIT_0,MCPWM_TIMER_0,&pwm_config);
}


void advance(/*range: 0~100*/uint8_t speed)
{
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,M
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B)
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,speed)
}

void retreat(/*range: 0~100*/uint8_t speed)
{
  //Give MOTOR_DIRECTION_PIN a continuous low level
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A)

  /*
   * Call function mcpwm_set_duty_type() every time after mcpw
   * or mcpwm_set_signal_low() to resume with previously set d
   */
```

```
  mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B,M

  //The MOTOR_STEP_PIN pin outputs a PWM wave with a duty cycl
  mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B,speed)
}

void stop(void)
{
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A)
  mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B)
}

void breake(void)
{
  mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A
  mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_B
}

void wheelSpeed()
{
  int Lstate = digitalRead(encoder0pinA);
  if((encoder0PinALast == LOW) && Lstate==HIGH)
  {
    int val = digitalRead(encoder0pinB);
    if(val == LOW && Direction)
    {
      Direction = false; //Reverse
    }
    else if(val == HIGH && !Direction)
    {
      Direction = true;  //Forward
    }
  }
  encoder0PinALast = Lstate;

  if(!Direction)
    duration++;
  else
    duration--;
}

void EncoderInit()
{
  Direction = true;//default -> Forward
  pinMode(encoder0pinB,INPUT);
  attachInterrupt(digitalPinToInterrupt(encoder0pinA), wheelSp
}

void setup() {
  Serial.begin(115200);//Initialize the serial port
  mcpwm_init();
  advance(70);
  Setpoint =80;  //Set the output value of PID
  myPID.SetMode(AUTOMATIC);//Set PID to automatic mode
  myPID.SetSampleTime(100);//Set the PID sampling frequency to
  EncoderInit();//Initialize the module
}

void loop() {
  abs_duration=abs(duration);
  result=myPID.Compute();//The return value of PID conversion
  if(result)
  {
    Serial.print("Pluse: ");
    Serial.println(duration);
    duration = 0; //Count clear and wait for the next count
  }
```

Drive car practice

Connect the hardware according to the wiring diagram, burn the code, connect "ESP32 S3 Romeo", enter 192.168.4.1 in the browser to drive the car and view the camera data

Please download and install before use

- AXP313A library
- ESPAsyncWebServer library
- AsyncTCP library

```
DFRobot_AXP313A axp;

#define M1_EN 12
#define M1_PN 13
#define M2_EN 14
#define M2_PN 21
#define M3_EN 9
#define M3_PN 10
#define M4_EN 47
#define M4_PN 11


#define UP 1
#define DOWN 2
#define LEFT 3
#define RIGHT 4
#define STOP 0

#define FORWARD 1
#define BACKWARD -1

//Camera related constants
#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     45
#define SIOD_GPIO_NUM     1
#define SIOC_GPIO_NUM     2

#define Y9_GPIO_NUM       48
#define Y8_GPIO_NUM       46
#define Y7_GPIO_NUM       8
#define Y6_GPIO_NUM       7
#define Y5_GPIO_NUM       4
#define Y4_GPIO_NUM       41
#define Y3_GPIO_NUM       40
#define Y2_GPIO_NUM       39
#define VSYNC_GPIO_NUM    6
#define HREF_GPIO_NUM     42
#define PCLK_GPIO_NUM     5

const char* ssid     = "ESP32 S3 Romeo";
const char* password = "12345678";

AsyncWebServer server(80);
AsyncWebSocket wsCamera("/Camera");
AsyncWebSocket wsCarInput("/CarInput");
uint32_t cameraClientId = 0;

uint8_t Speed = 50;

const char* htmlHomePage PROGMEM = R"HTMLHOMEPAGE(
<!DOCTYPE html>
<html>
  <head>
  <meta name="viewport" content="width=device-width, initial-s
    <style>
```

```
  .arrows {
    font-size:30px;
    color:red;
  }
  td.button {
    background-color:black;
    border-radius:25%;
    box-shadow: 5px 5px #888888;
  }
  td.button:active {
    transform: translate(5px,5px);
    box-shadow: none;
  }

  .noselect {
    -webkit-touch-callout: none; /* iOS Safari */
      -webkit-user-select: none; /* Safari */
       -khtml-user-select: none; /* Konqueror HTML */
         -moz-user-select: none; /* Firefox */
          -ms-user-select: none; /* Internet Explorer/Edge *
              user-select: none; /* Non-prefixed version, cu
                                    supported by Chrome and
  }

  .slidecontainer {
    width: 100%;
  }

  .slider {
    -webkit-appearance: none;
    width: 100%;
    height: 15px;
    border-radius: 5px;
    background: #d3d3d3;
    outline: none;
    opacity: 0.7;
    -webkit-transition: .2s;
    transition: opacity .2s;
  }

  .slider:hover {
    opacity: 1;
  }

  .slider::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 25px;
    height: 25px;
    border-radius: 50%;
    background: red;
    cursor: pointer;
  }

  .slider::-moz-range-thumb {
    width: 25px;
    height: 25px;
    border-radius: 50%;
    background: red;
    cursor: pointer;
  }

  </style>

</head>
<body class="noselect" align="center" style="background-colo
  <table id="mainTable" style="width:400px;margin:auto;table
    <tr>
      <img id="cameraImage" src="" style="width:400px;height
    </tr>
    <tr>
      <td></td>
      <td class="button" ontouchstart='sendButtonInput("Move
      <td></td>
    </tr>
    <tr>
```

```
        <td class="button" ontouchstart='sendButtonInput("Move
        <td class="button"></td>
        <td class="button" ontouchstart='sendButtonInput("Move
      </tr>
      <tr>
        <td></td>
        <td class="button" ontouchstart='sendButtonInput("Move
        <td></td>
      </tr>
      <tr/><tr/>
      <tr>
        <td style="text-align:left"><b>Speed:</b></td>
        <td colspan=2>
         <div class="slidecontainer">
            <input type="range" min="0" max="100" value="50" c
          </div>
        </td>
      </tr>

    </table>

    <script>
      var webSocketCameraUrl = "ws:\/\/" + window.location.hos
      var webSocketCarInputUrl = "ws:\/\/" + window.location.h
      var websocketCamera;
      var websocketCarInput;

      function initCameraWebSocket()
      {
        websocketCamera = new WebSocket(webSocketCameraUrl);
        websocketCamera.binaryType = 'blob';
        websocketCamera.onopen    = function(event){};
        websocketCamera.onclose   = function(event){setTimeout
        websocketCamera.onmessage = function(event)
        {
          var imageId = document.getElementById("cameraImage")
          imageId.src = URL.createObjectURL(event.data);
        };
      }

      function initCarInputWebSocket()
      {
        websocketCarInput = new WebSocket(webSocketCarInputUrl
        websocketCarInput.onopen    = function(event)
        {
          sendButtonInput("Speed", document.getElementById("Sp

        };
        websocketCarInput.onclose   = function(event){setTimeo
        websocketCarInput.onmessage = function(event){};
      }

      function initWebSocket()
      {
        initCameraWebSocket ();
        initCarInputWebSocket();
      }

      function sendButtonInput(key, value)
      {
        var data = key + "," + value;
        websocketCarInput.send(data);
      }

      window.onload = initWebSocket;
      document.getElementById("mainTable").addEventListener("t
        event.preventDefault()
      });
    </script>
  </body>
</html>
)HTMLHOMEPAGE";


void advance(uint8_t motorNumber,uint8_t speed)
{
```

```
  switch(motorNumber)
  {
    case 1:
      mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_G
      mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,sp
      break;

    case 2:
      mcpwm_set_signal_high(MCPWM_UNIT_0,MCPWM_TIMER_1,MCPWM_G
      mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_1,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_1,MCPWM_GEN_A,sp
      break;

    case 3:
      mcpwm_set_signal_high(MCPWM_UNIT_1,MCPWM_TIMER_0,MCPWM_G
      mcpwm_set_duty_type(MCPWM_UNIT_1,MCPWM_TIMER_0,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_1,MCPWM_TIMER_0,MCPWM_GEN_A,sp
      break;

    case 4:
      mcpwm_set_signal_high(MCPWM_UNIT_1,MCPWM_TIMER_1,MCPWM_G
      mcpwm_set_duty_type(MCPWM_UNIT_1,MCPWM_TIMER_1,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_1,MCPWM_TIMER_1,MCPWM_GEN_A,sp
      break;
  }
}

void retreat(uint8_t motorNumber,uint8_t speed)
{
  switch(motorNumber)
  {
    case 1:
      mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GE
      mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GEN_A,sp
      break;

    case 2:
      mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_1,MCPWM_GE
      mcpwm_set_duty_type(MCPWM_UNIT_0,MCPWM_TIMER_1,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_0,MCPWM_TIMER_1,MCPWM_GEN_A,sp
      break;

    case 3:
      mcpwm_set_signal_low(MCPWM_UNIT_1,MCPWM_TIMER_0,MCPWM_GE
      mcpwm_set_duty_type(MCPWM_UNIT_1,MCPWM_TIMER_0,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_1,MCPWM_TIMER_0,MCPWM_GEN_A,sp
      break;

    case 4:
      mcpwm_set_signal_low(MCPWM_UNIT_1,MCPWM_TIMER_1,MCPWM_GE
      mcpwm_set_duty_type(MCPWM_UNIT_1,MCPWM_TIMER_1,MCPWM_GEN
      mcpwm_set_duty(MCPWM_UNIT_1,MCPWM_TIMER_1,MCPWM_GEN_A,sp
      break;
  }
}

void breake(uint8_t motorNumber)
{
  switch(motorNumber)
  {
    case 1:
      mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_0,MCPWM_GE
      break;

    case 2:
      mcpwm_set_signal_low(MCPWM_UNIT_0,MCPWM_TIMER_1,MCPWM_GE
      break;

    case 3:
      mcpwm_set_signal_low(MCPWM_UNIT_1,MCPWM_TIMER_0,MCPWM_GE
      break;

    case 4:
      mcpwm_set_signal_low(MCPWM_UNIT_1,MCPWM_TIMER_1,MCPWM_GE
```

```
      break;
  }
}

void moveCar(uint8_t inputValue,uint8_t speed)
{
  switch(inputValue)
  {
    case UP:
      advance(1,speed);
      advance(2,speed);
      advance(3,speed);
      advance(4,speed);
      break;

    case DOWN:
      retreat(1,speed);
      retreat(2,speed);
      retreat(3,speed);
      retreat(4,speed);
      break;

    case LEFT:
      advance(2,speed);
      advance(4,speed);
      retreat(1,speed);
      retreat(3,speed);
      break;

    case RIGHT:
      advance(1,speed);
      advance(3,speed);
      retreat(2,speed);
      retreat(4,speed);
      break;

    case STOP:
      breake(1);
      breake(2);
      breake(3);
      breake(4);
      break;
  }
}

void handleRoot(AsyncWebServerRequest *request)
{
  request->send_P(200, "text/html", htmlHomePage);
}

void handleNotFound(AsyncWebServerRequest *request)
{
    request->send(404, "text/plain", "File Not Found");
}

void onCarInputWebSocketEvent(AsyncWebSocket *server,
                      AsyncWebSocketClient *client,
                      AwsEventType type,
                      void *arg,
                      uint8_t *data,
                      size_t len)
{
  switch (type)
  {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n"
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", cli
      moveCar(STOP,0);
      //ledcWrite(PWMLightChannel, 0);
      //panServo.write(90);
      //tiltServo.write(90);
      break;
    case WS_EVT_DATA:
      AwsFrameInfo *info;
```

```
      info = (AwsFrameInfo*)arg;
      if (info->final && info->index == 0 && info->len == len
      {
        std::string myData = "";
        myData.assign((char *)data, len);
        std::istringstream ss(myData);
        std::string key, value;
        std::getline(ss, key, ',');
        std::getline(ss, value, ',');
        Serial.printf("Key [%s] Value[%s]\n", key.c_str(), val
        int valueInt = atoi(value.c_str());
        if (key == "MoveCar")
        {
          moveCar(valueInt,Speed);
        }
        else if (key == "Speed")
        {
          Speed = valueInt;
        }
        /*else if (key == "Light")
        {
          //ledcWrite(PWMLightChannel, valueInt);
        }
        else if (key == "Pan")
        {
          //panServo.write(valueInt);
        }
        else if (key == "Tilt")
        {
          //tiltServo.write(valueInt);
        }  */
      }
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
    default:
      break;
  }
}

void onCameraWebSocketEvent(AsyncWebSocket *server,
                      AsyncWebSocketClient *client,
                      AwsEventType type,
                      void *arg,
                      uint8_t *data,
                      size_t len)
{
  switch (type)
  {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n"
      cameraClientId = client->id();
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", cli
      cameraClientId = 0;
      break;
    case WS_EVT_DATA:
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
    default:
      break;
  }
}

void setupCamera()
{
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
```

```
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG; // for streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for face detect
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;

    // if PSRAM IC present, init with UXGA resolution and high
  //                      for larger pre-allocated frame buffe
  if(config.pixel_format == PIXFORMAT_JPEG){
    if(psramFound()){
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
  }


  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK)
  {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  if (psramFound())
  {
    heap_caps_malloc_extmem_enable(20000);
    Serial.printf("PSRAM initialized. malloc to take memory fr
  }
}

void sendCameraPicture()
{
  if (cameraClientId == 0)
  {
    return;
  }
  unsigned long  startTime1 = millis();
  //capture a frame
  camera_fb_t * fb = esp_camera_fb_get();
  if (!fb)
  {
      Serial.println("Frame buffer could not be acquired");
      return;
  }

  unsigned long  startTime2 = millis();
  wsCamera.binary(cameraClientId, fb->buf, fb->len);
  esp_camera_fb_return(fb);
```

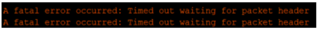
```
  //Wait for message to be delivered
  while (true)
  {
    AsyncWebSocketClient * clientPointer = wsCamera.client(cam
    if (!clientPointer || !(clientPointer->queueIsFull()))
    {
      break;
    }
    delay(1);
  }

  unsigned long  startTime3 = millis();
  Serial.printf("Time taken Total: %d|%d|%d\n",startTime3 - st
}

//Initialize the pins needed to generate the PWM signal
void mcpwm_init(void)
{
  //Configure mcpwm information
  mcpwm_config_t pwm_config;
  pwm_config.frequency = 1000;
  pwm_config.cmpr_a = 0;
  pwm_config.cmpr_b = 0;
  pwm_config.counter_mode = MCPWM_UP_COUNTER;
  pwm_config.duty_mode = MCPWM_DUTY_MODE_0;

  //A speed B direction
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0A,M1_EN);//Motor 1 GPIO
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM0B,M1_PN);
  mcpwm_init(MCPWM_UNIT_0,MCPWM_TIMER_0,&pwm_config);//Initial

  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM1A,M2_EN);//Motor 2 GPIO
  mcpwm_gpio_init(MCPWM_UNIT_0,MCPWM1B,M2_PN);
  mcpwm_init(MCPWM_UNIT_0,MCPWM_TIMER_1,&pwm_config);//Initial

  mcpwm_gpio_init(MCPWM_UNIT_1,MCPWM0A,M3_EN);//Motor 3 GPIO
  mcpwm_gpio_init(MCPWM_UNIT_1,MCPWM0B,M3_PN);
  mcpwm_init(MCPWM_UNIT_1,MCPWM_TIMER_0,&pwm_config);//Initial

  mcpwm_gpio_init(MCPWM_UNIT_1,MCPWM1A,M4_EN);//Motor 4 GPIO
  mcpwm_gpio_init(MCPWM_UNIT_1,MCPWM1B,M4_PN);
  mcpwm_init(MCPWM_UNIT_1,MCPWM_TIMER_1,&pwm_config);//Initial
}


void setup(void)
{
  Serial.begin(115200);
  while(axp.begin() != 0){
    Serial.println("init error");
    delay(1000);
  }
  axp.enableCameraPower(axp.eOV2640);//Set up camera power sup
  mcpwm_init();

  WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);

  server.on("/", HTTP_GET, handleRoot);
  server.onNotFound(handleNotFound);

  wsCamera.onEvent(onCameraWebSocketEvent);
  server.addHandler(&wsCamera);

  wsCarInput.onEvent(onCarInputWebSocketEvent);
  server.addHandler(&wsCarInput);

  server.begin();
  Serial.println("HTTP server started");

  setupCamera();
}
```

```
void loop()
{
  wsCamera.cleanupClients();
  wsCarInput.cleanupClients();
```

FAQ

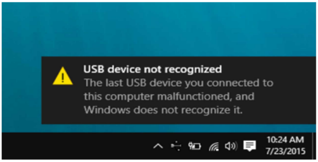
1. What will cause burning error?

- There is no delay or too short delay in Loop.



```
A fatal error occurred: Timed out waiting for packet header
A fatal error occurred: Timed out waiting for packet header
```

- The USB cannot be recognized by the PC as some functions are incorrectlly called.



USB device not recognized
The last USB device you connected to this computer malfunctioned, and Windows does not recognize it.

How to solve

- Press Boot, press RST and release both, then try burning again.

2. Data cannot be printed on serial port

- Check if the USB CDC is enabled
- Check print information using other serial debugger.

For any questions, advice or cool ideas to share, please visit the **DFRobot Forum**.

More Documents

- ESP32-S3-WROOM-1 Datasheet
- ESP32-S3 Chip Datasheet
- DFR0994-Schematics.pdf
- DFR0994-Dimension
- DFR0994-Rendergraph

Get **Romeo ESP32 S3** from DFRobot Store or **DFRobot Distributor**.

Turn to the Top

Downloaded from Arrow.com.