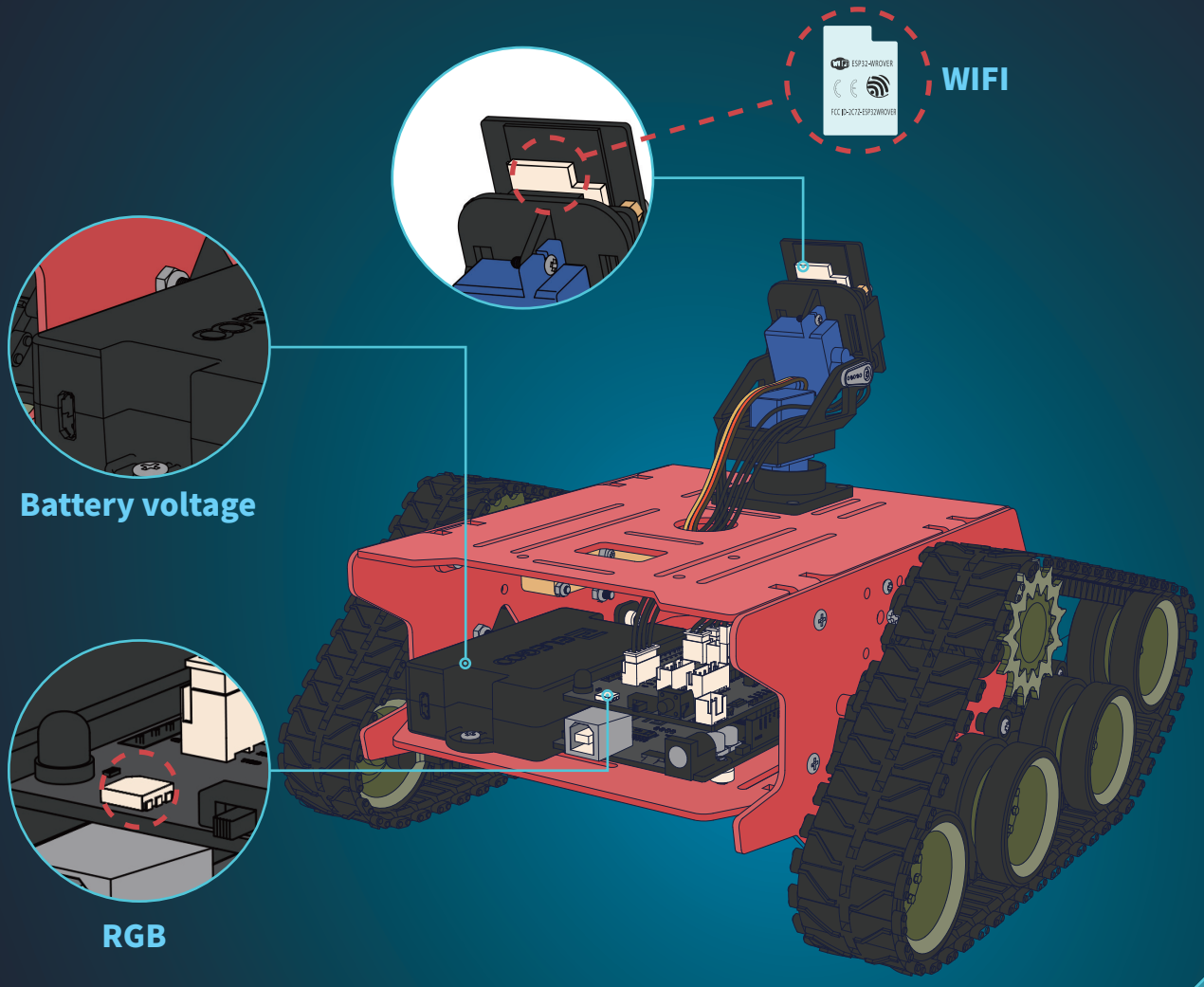
ELEGOO

7 Lesson

Others



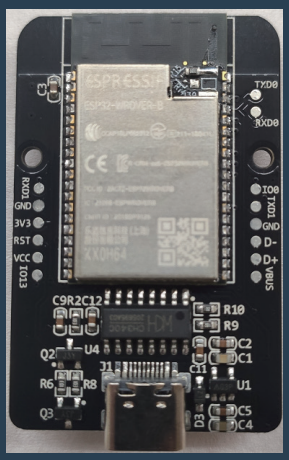
Introduction:

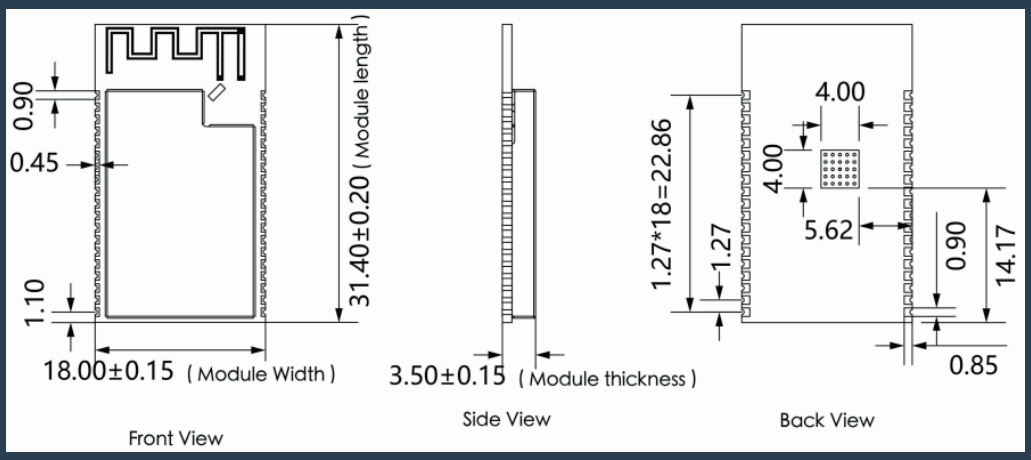
In this lesson, we will introduce you the function and the specific use of the remaining components.

Material Preparation:

A Conqueror Robot Car (with battery)
A USB Cable

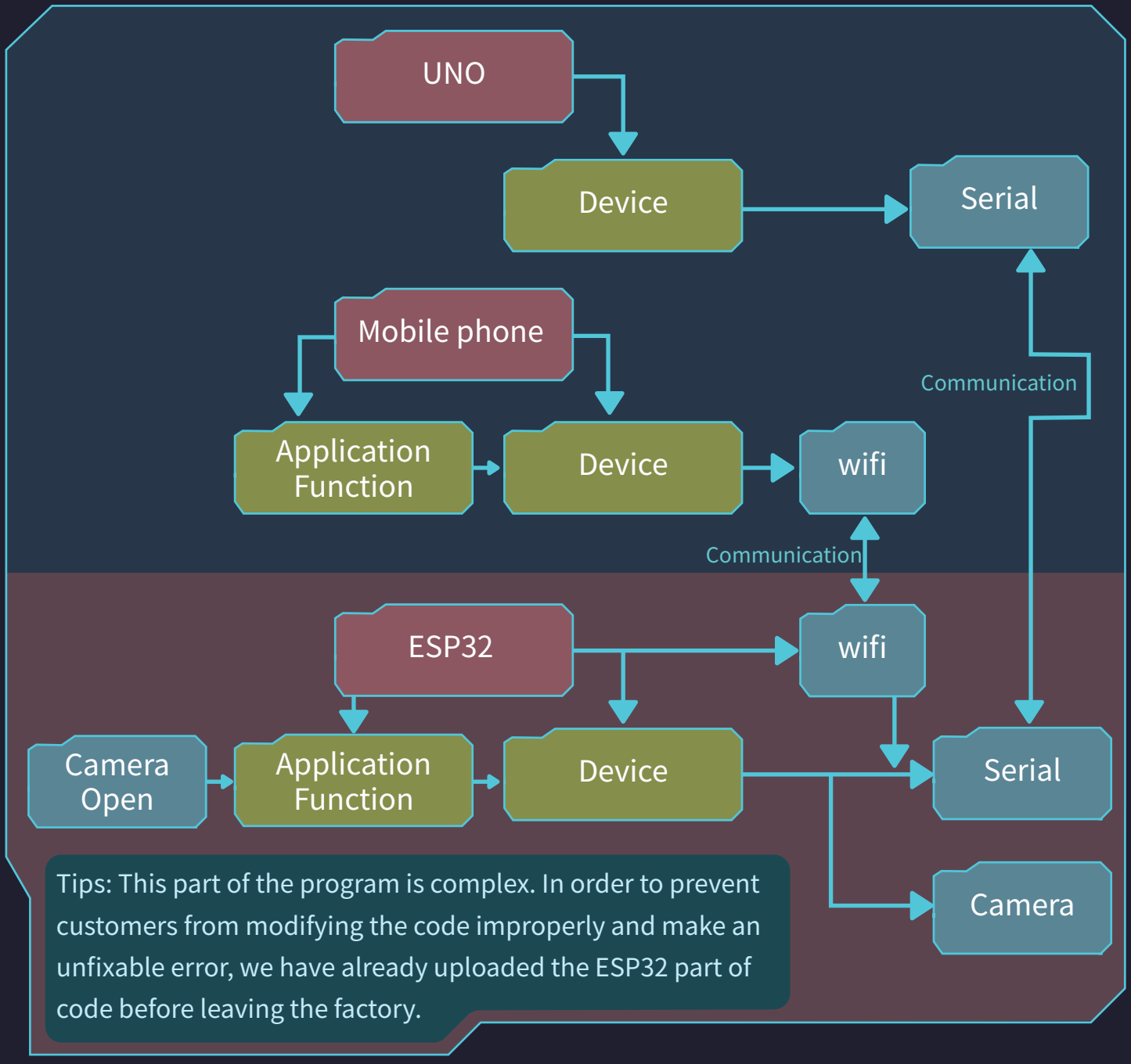
First, the use of Wi-Fi module.





Wi-Fi can simply be understood as wireless communication and is one of the most widely used wireless network transmission technologies today. Compared to Bluetooth, Wi-Fi has a wider coverage area, faster transmission speed, and is capable of many-to-many communication. In our kit, the Wi-Fi module is soldered on the ESP32 board. As the Wi-Fi module driver is complex to use, we are worried that users will have a bad experience due to improper operation, so we have already helped users configure the Wi-Fi module and camera module soldered on the ESP32 before leaving the factory, and we won't teach this part of the ESP32 for now.

The specific process is to use the Wi-Fi module of the ESP32 to open the Wi-Fi hotspot, and send the information through the APP after the phone has connected to the hotspot. The information was sent out through the Wi-Fi module of the phone and then received by the Wi-Fi module of the ESP32. Since we have set the Wi-Fi module to pass-through mode before leaving the factory, the acquired data received by the Wi-Fi module of the ESP32 is transferred to the serial port of the ESP32. And since the serial port on ESP32 is connected to the serial port on UNO, we just need to read the serial port on the UNO to get the data from the APP.



In the program on UNO, we obtain the string by reading the data received in the serial buffer, and parse the string. The key information is extracted through the fixed format of the string, and the flag bit is set according to different key information, and different functions are selected to be called to realize the corresponding function.

The format of the string (json data format) is rough as follows: {"N": 2, "D1": 1}, the parameters corresponding to "N" represent different functions, and "D1", "D2" and so on represent the parameters that need to be modified and adjusted when implementing different functions.

For specific communication instructions, please check out “Communication protocol for Conqueror” in the "04 Related chip information" folder.

Please open Demo1 in the current folder:

Let’ s first take a look at the definition of the relative functions of serial port data reading.

```
// in ApplicationFunctionSet_xxx0.h

class ApplicationFunctionSet
{
public:
  void ApplicationFunctionSet_Init(void);
  void ApplicationFunctionSet_SerialPortDataAnalysis(void);
  String CommandSerialNumber;
};
```

Then we need to add the JSON library to analyse the JSON data send from the APP.

```
// in ApplicationFunctionSet_xxx0.cpp
#include "ArduinoJson-v6.11.1.h" //ArduinoJson
```

Initial the serial port.

```
// in ApplicationFunctionSet_xxx0.cpp
void ApplicationFunctionSet::ApplicationFunctionSet_Init(void)
{
  Serial.begin(9600);
}
```

Access an complete instruction from the serial port.

```
// in ApplicationFunctionSet_xxx0.cpp
void
ApplicationFunctionSet::ApplicationFunctionSet_SerialPortDataAnalysis
(void)
{
  static String SerialPortData = "";
  uint8_t c = "";
  if (Serial.available() > 0)
  {
    while (c != '}' && Serial.available() > 0)
{
  c = Serial.read();
      SerialPortData += (char)c;
}
.....
  }
```

Serial.available(): determine if there is data in the buffer of the serial port

Serial.read(): read one byte at a time

Since the format of the string (JSON data format) is rough as follows: {" N ": 2," D1 ": 1}, we only need to determine whether the ending character is"} "to know whether we have received the entire command.

```
// in ApplicationFunctionSet_xxx0.cpp
void
ApplicationFunctionSet::ApplicationFunctionSet_SerialPortDataAnalysis(void)
{
.....
 if (c == '}')
 {
}
.......
  }
```

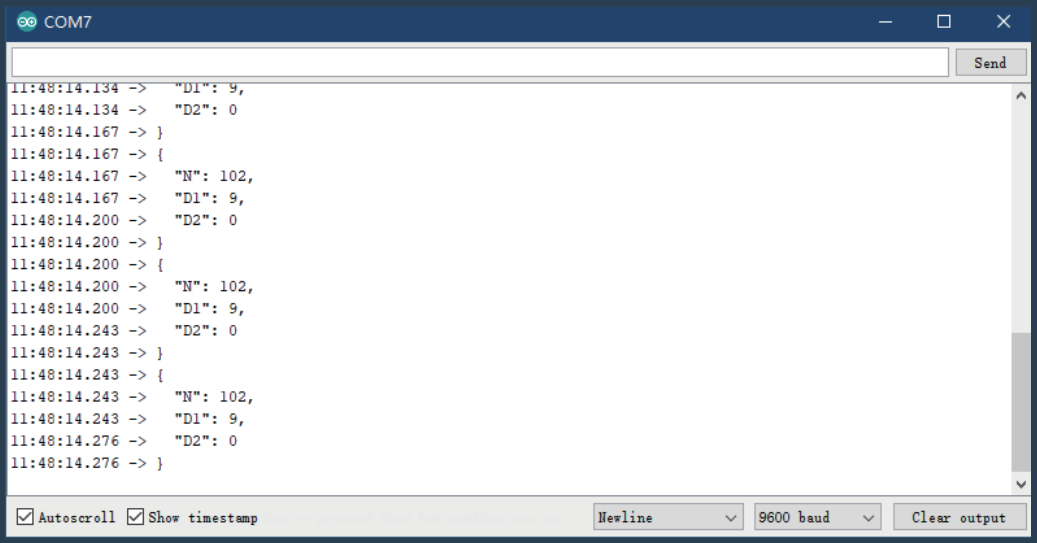
After obtaining the command successfully, the command needs to be parsed.

```
// in ApplicationFunctionSet_xxx0.cpp
void
ApplicationFunctionSet::ApplicationFunctionSet_SerialPortDataAnalysis
(void)
{
.....
else if (!error)
{
    int control_mode_N = doc["N"];
    char buf[3];
    uint8_t temp = doc["H"];
    sprintf(buf, "%d", temp);
    CommandSerialNumber = buf;
.......
 }
```

Each case represents one mode, we only need to put in the functions we have implemented in the previous few lessons to realize the APP switching function.

```
// in ApplicationFunctionSet_xxx0.cpp
void
ApplicationFunctionSet::ApplicationFunctionSet_SerialPortDataAnalysis(void)
{
.....
 switch (control_mode_N)
    {
   case 1: /*<命令:N 1> */
     break;
    case 2: /*<命令:N 2>*/
   .......
 }
```

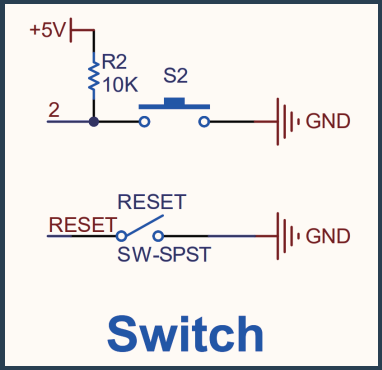
Upload program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program.) After the program has been uploaded successfully, please do not disconnect the USB cable. Turn toggle the “Upload-Cam” button to “Upload”, and then open mobile’ s APP and connect to the car. When the connection is complete, operate the button in the APP casually. By this time, open the serial port monitor in the Arduino IDE and you will find that every operation you do on the APP will send a string to the car, and the car will perform the corresponding function according to these instructions.



Second, the use of keys.

We can also switch functions by the key on our Conqueror Robot Car.

Please open the folder in the previous level for more details: Related chip information -> ConquerorCar-Shield-V1.0.pdf



We know that the key is connected to the D2 pin on the UNO.

Next, please open Demo2:

Let's first look at the definition of the relative pins and variables:

```
// in DeviceDriverSet_xxx0.h

/*Key Detection*/
class DeviceDriverSet_Key
{
public:
  void DeviceDriverSet_Key_Init(void);
#if _Test_DeviceDriverSet
  void DeviceDriverSet_Key_Test(void);
#endif
  void DeviceDriverSet_key_Get(uint8_t *get_keyValue);
public:
#define PIN_Key 2
#define keyValue_Max 4
public:
  static uint8_t keyValue;
};
```

In the following programming, we will use the interrupt function.

Usually our programs are executed sequentially, but interrupts are used when you have something more urgent to deal with.

Now, we’ ll need to add the corresponding library first:

```
// in DeviceDriverSet_xxx0.cpp

#include "PinChangeInt.h"
```

Then, define the key pin and add the interrupt function to it.

We set it as falling edge trigger. When the key is pressed, the level changes, and the falling edge will trigger the interrupt function.

```
// in DeviceDriverSet_xxx0.cpp

void DeviceDriverSet_Key::DeviceDriverSet_Key_Init(void)
{
  pinMode(PIN_Key, INPUT_PULLUP);
  attachPinChangeInterrupt
  (PIN_Key, attachPinChangeInterrupt_GetKeyValue, FALLING);
}
```

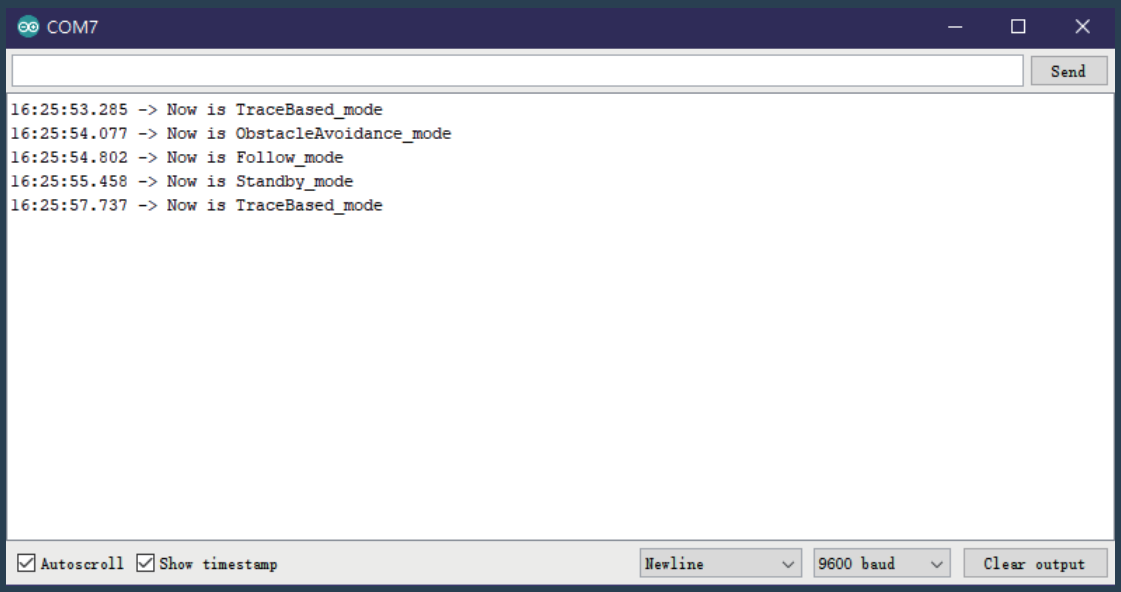
And we need to record how many times the key is pressed. Because we only have 4 modes, we only record the fourth time at most.

```
// in DeviceDriverSet_xxx0.cpp

static void attachPinChangeInterrupt_GetKeyValue(void)
{
  DeviceDriverSet_Key Key;
  static uint32_t keyValue_time = 0;
  static uint8_t keyValue_temp = 0;
  if ((millis() - keyValue_time) > 500)
  {
    keyValue_temp++;
    keyValue_time = millis();
    if (keyValue_temp > keyValue_Max)
    {
      keyValue_temp = 0;
    }
    Key.keyValue = keyValue_temp;
  }
}
```

It will trigger the interrupt function when we press the key and the corresponding function would be selected according to the times we press the key.

```
// in ApplicationFunctionSet_xxx0.cpp

void
ApplicationFunctionSet::ApplicationFunctionSet_KeyCommand
(void)
```

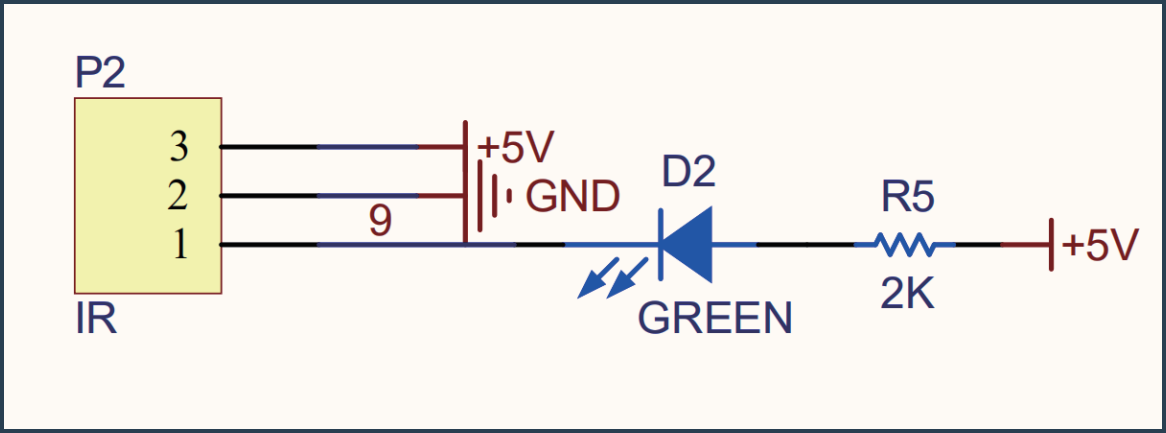
Upload program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program.) After the program has been uploaded successfully, please open the serial port and press the key, you will see the statement that switches to different functions is displayed on the serial port. Since our focus in this section is on the use of keys, we only print out the current switched mode statement on the serial port and do not implement the corresponding specific functions, but you can port different functions in according to the previous lessons.



Third, the use of infrared receiving and transmitting module.

Other than pressing keys to switch functions, we can also switch the functions of our Conqueror Robot Car through infrared remote control since we have applied the infrared receiving module.

Please open the folder in the previous level for more details: Related chip information -> ConquerorCar-Shield-V1.0.pdf



As shown in the figure, the infrared receiving module is connected to the D9 pin of UNO, and we also connect it to a green LED light. When we press the key of the remote control, if the infrared receiving module receives the signal, the LED light will turn on.

Infrared remote control is mainly composed of infrared transmitter and infrared receiver.

The signals transmitted and received by infrared are actually a series of binary pulse codes, and the high and low levels change according to a certain time rule to transmit corresponding information. In order to protect it from interference from other signals during wireless transmission, the signal is usually modulated on a specific carrier frequency (38K infrared carrier signal) and transmitted through an infrared emitting diode, while the infrared receiving end needs to decode and adjust the signal, and then restore it to binary pulse code for processing.

Next, please open Demo3:

Let’ s take a look at the definition of the relative pins and variables:

```
// in DeviceDriverSet_xxx0.h

class DeviceDriverSet_IRrecv
{
public:
  void DeviceDriverSet_IRrecv_Init(void);
  bool DeviceDriverSet_IRrecv_Get(uint8_t *IRrecv_Get /*out*/);
  void DeviceDriverSet_IRrecv_Test(void);

public:
  unsigned long IR_PreMillis;

private:
#define RECV_PIN 9
......
};
```

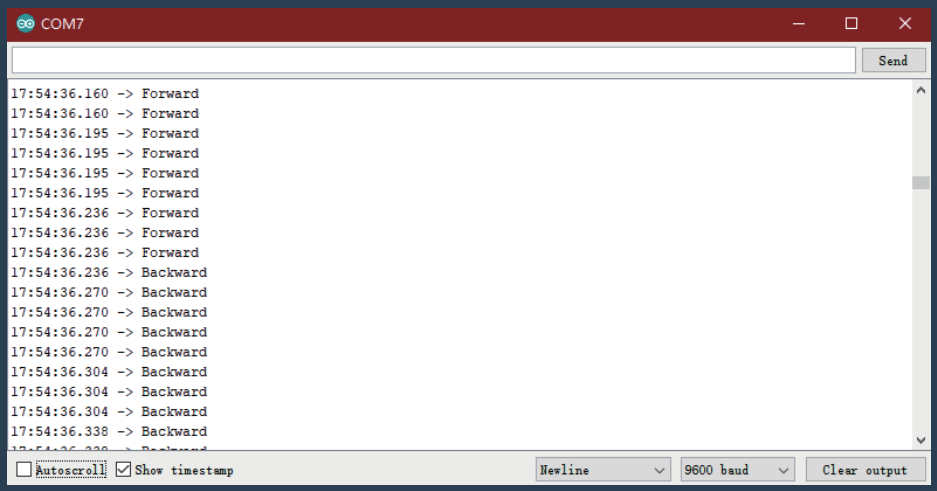
Then, distinguish the different keys according to the signal we received.

```
// in DeviceDriverSet_xxx0.cpp

bool DeviceDriverSet_IRrecv::DeviceDriverSet_IRrecv_Get
(uint8_t *IRrecv_Get /*out*/)
```

Finally, implement different functions according to the key we pressed.

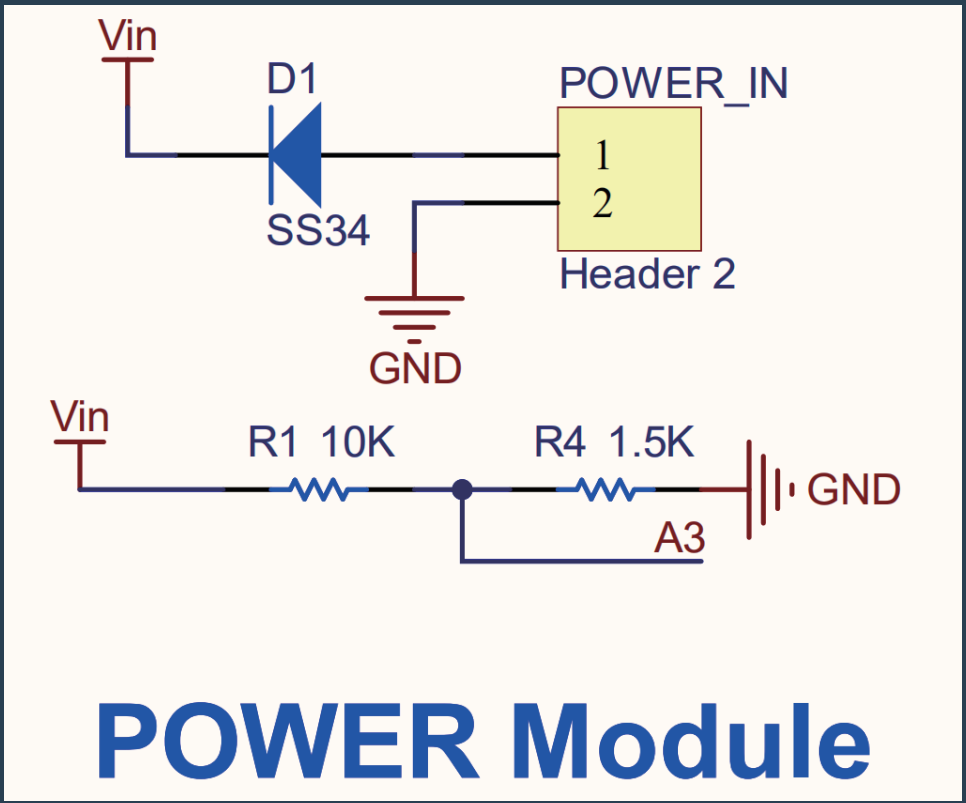
```
// in ApplicationFunctionSet_xxx0.cpp

void ApplicationFunctionSet::ApplicationFunctionSet_IRrecv(void)
```

Upload program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program.) After the program has been uploaded successfully, please open the serial port and press the key, you will see the statement that switches to different functions is displayed on the serial port. Since our focus in this section is on the use of keys, we only print out the current switched mode statement on the serial port and do not implement the corresponding specific functions, but you can port different functions in according to the previous lessons.



Forth, voltage detection.

In order to prevent damage to the battery caused by excessive discharge, voltage must be monitored. So we'll teach you how to measure the battery voltage.



We set the pin of Arduino to collect battery voltage as A3, because the internal reference voltage of the Arduino chip is 5V, that is to say, no matter how large the supply voltage is, the value of 0~1023 sampled by ADC corresponds to the voltage value of 0V~5V. The voltage over 5V is still 1023. The AVR document also pointed out that this benchmark value has a certain deviation. But the 5V reference voltage value is too low, and the measured voltage must be smaller than it. So let's divide the voltage that needs to be tested. Here we use a 10k and a 1.5k resistor in series, and then connect the A3 pin to the middle of the two resistors for measurement. At this time, the one-eleventh voltage value must be below 1.1V.

Now, please open Demo4:

Firstly, we need to initial the pin we used.

```
void voltageInit()
{
  pinMode(VOL_MEASURE_PIN, INPUT);
}
```

Then calculate the power supply voltage according to the formula.

Vout = (Vin × R2) / (R1 + R2)

Then collect the value of A3 through the analogRead() function, we know that the Arduino controller has multiple 10-digital modular conversion channels. This means that Arduino can map voltage input signals of 0-5V to a value of 0-1023. In other words, we can divide the 5V signal into 1024 equal parts, the 0V input signal corresponds to the value 0, and the 5Vt input signal corresponds to 1023. To achieve voltage monitoring, it is not enough to use the analogRead() function to read the A4 value. We also need to convert this read value into an actual voltage value, so we can use the following formula for conversion:

Battery voltage V = A3 read value * (5.00 / 1024.00) * multiplier

To sum up:

Battery voltage V = A3 read value * (5.00 / 1024.00) * ((R1 + R2) /R2)

Since the data might be overlooked by the software and cause errors if the decimal is too large when the data is dividing by 1024, we' d better divide the data by 1024 at the end.

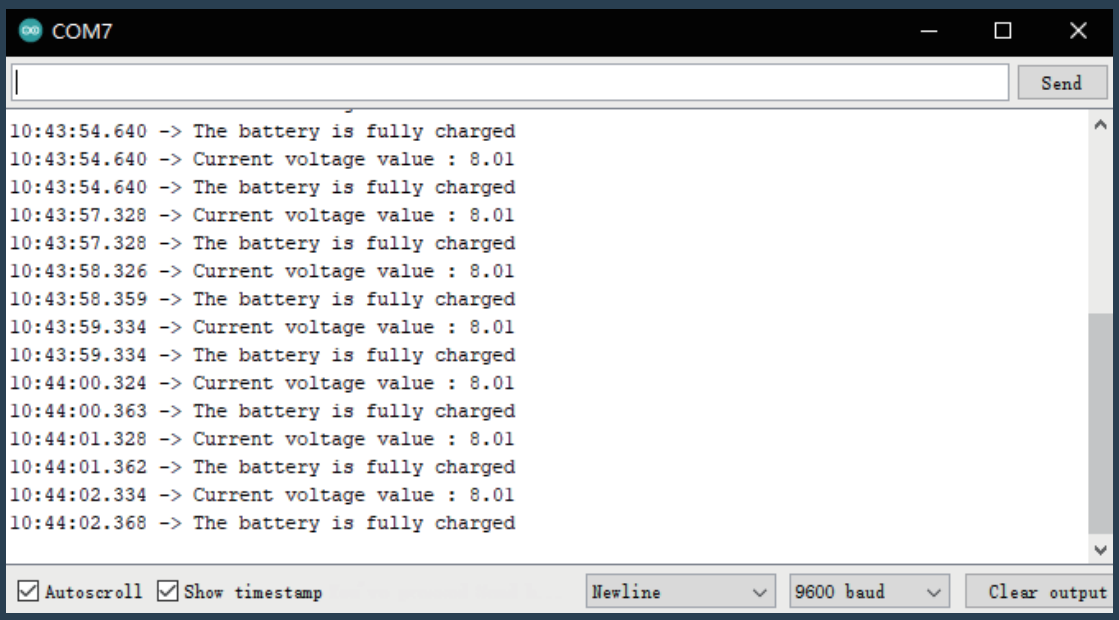
Battery voltage V = A3 read value * (5.00) * ((R1 + R2)/R2) / 1024.00

Finally, correct the error.

Battery voltage V =Battery voltage V+ (Battery voltage V* 0.08)

```
void Voltage_Measure()
{
  if (millis() - vol_measure_time > 1000)
  //Measured every 1000 milliseconds
 {
  vol_measure_time = millis();

  float voltage = (analogRead(VOL_MEASURE_PIN) * 5 )
                  * ((10 + 1.5) / 1.5)/ 1024; //Read voltage value

   //float voltage = (analogRead(VOL_MEASURE_PIN) * 0.0375);
  voltage = voltage + (voltage * 0.08);
  Serial.print("Current voltage value : ");
  Serial.println(voltage);
  if(voltage>7.8)
    Serial.println("The battery is fully charged");
  else
    Serial.println("Low battery");
 }
}
```

Upload program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program.) After the program has been uploaded successfully, please open the serial port and you will see the current voltage value.

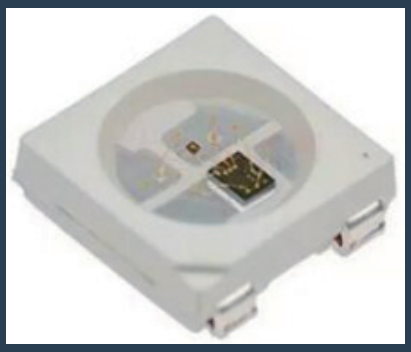


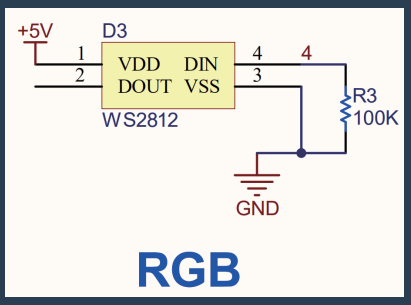
Fifth,the use of RGB colored light.

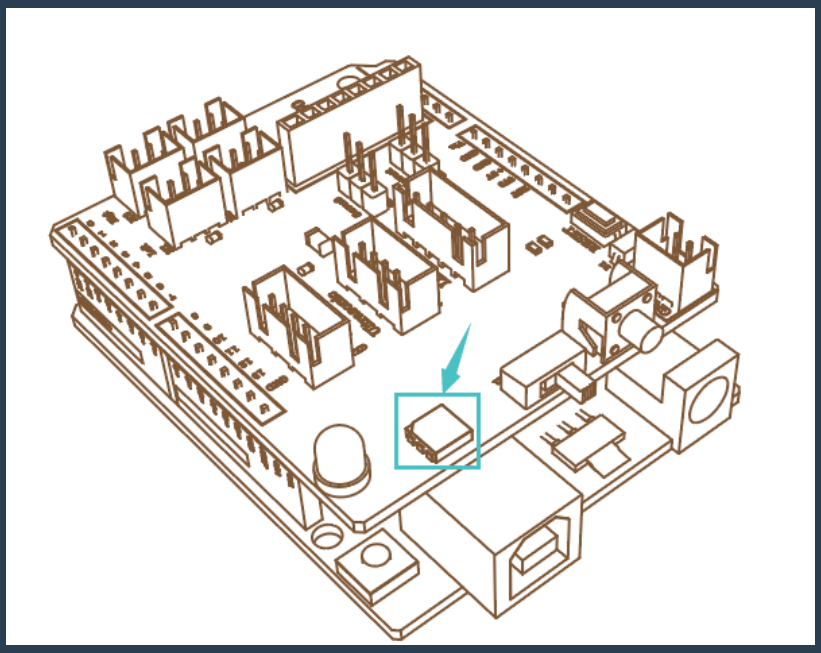
We will teach you how to turn on the colored light on the car and change its brightness and color.

In our kit, the realization of lighting effects uses the WS2812B module. WS2812B is an intelligent externally controlled LED light source integrating a control circuit and a light-emitting circuit. The data protocol adopts a single-wire return-to-zero code communication method. After the pixel is powered on and reset, the DIN terminal receives the data transmitted from the controller. The first 24bit data sent over is extracted by the first pixel and sent to the data latch inside the pixel, the remaining data is shaped and amplified by the internal shaping processing circuit and then forwarded through the DO port to start outputting to the next cascade of pixel, and the signal is reduced by 24bit for each pixel that passes through the transmission. The pixel adopts the automatic shaping and forwarding technology, so that the number of cascades of the pixel is not limited by the signal transmission, but only by the signal transmission speed requirements. LED has the advantages of low voltage drive, environmental protection and energy-saving, high brightness, large scattering angle, good consistency, ultra-low power, and ultra-long life. Integrating the control circuit on the LED makes the circuit simpler and smaller in size, and easier to install.

Please open the folder in the previous level for more details: Related chip information -> ConquerorCar-Shield-V1.0.pdf







It can be seen from the figure that the RGB colored light is connected to the D4 pin of UNO. Open Demo5 in the current folder:

In this lesson, we will use FastLED.h library.

```
#include "FastLED.h"
```

Define the relative variables.

```
#define PIN_RBGLED 4
#define NUM_LEDS 1
CRGB leds[NUM_LEDS];
```

Initial WS2812 and set the brightness of the LED light.

```
void setup() {
  Serial.begin(9600);
  FastLED.addLeds<NEOPIXEL, PIN_RBGLED>(leds, NUM_LEDS);
  FastLED.setBrightness(20);
}
```

Merges three 1-byte color data into a new color.

```
uint32_t Color(uint8_t r, uint8_t g, uint8_t b)
{
  return (((uint32_t)r << 16) | ((uint32_t)g << 8) | b);
}
```

Finally, change the color as you want.

```
Void myColor()
```

Upload program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program.) After the program has been upload successfully, please open the serial port and you will see the color of RGB colored light is changing gradually.