Internal use only

Infonique
iSEB RobotArm V1.0

Prepared by	Date	Version
Bing Ran	07/04/2024	1.0

Abstract

This document provides detailed of Infonique iSEB RobotArm specification.

Document History

Date	Rev	Modifier	Changes
07-April-2023	1.0	Bing Ran	First Draft

Contents

Table of Figures

Index of Tables

1 Introduction

This document will discuss the details of the iSEB RobotArm. ISEB RobotArm is sharing the same hardware with iSEB Crab which is iSEB Expansion Board 1200 0012 V1.0 It will control 4 servo motors. The figure below is showing the iSEB RobotArm.

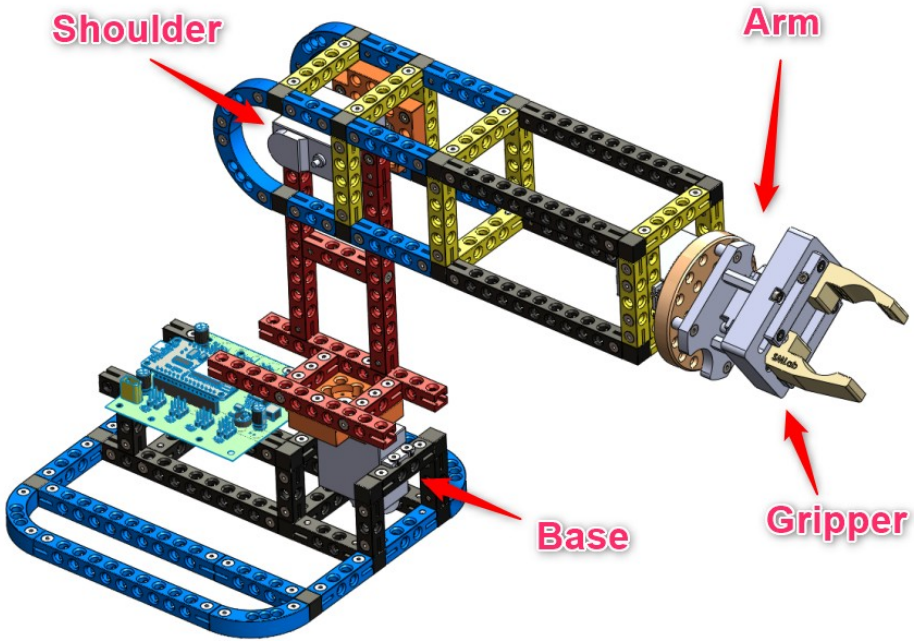


Figure 1: iSEB RobotArm

There are
for servo motors in iSEB Robot Arm which are base , shoulder , arm and gripper.

2 Hardware

2.1 Schematic

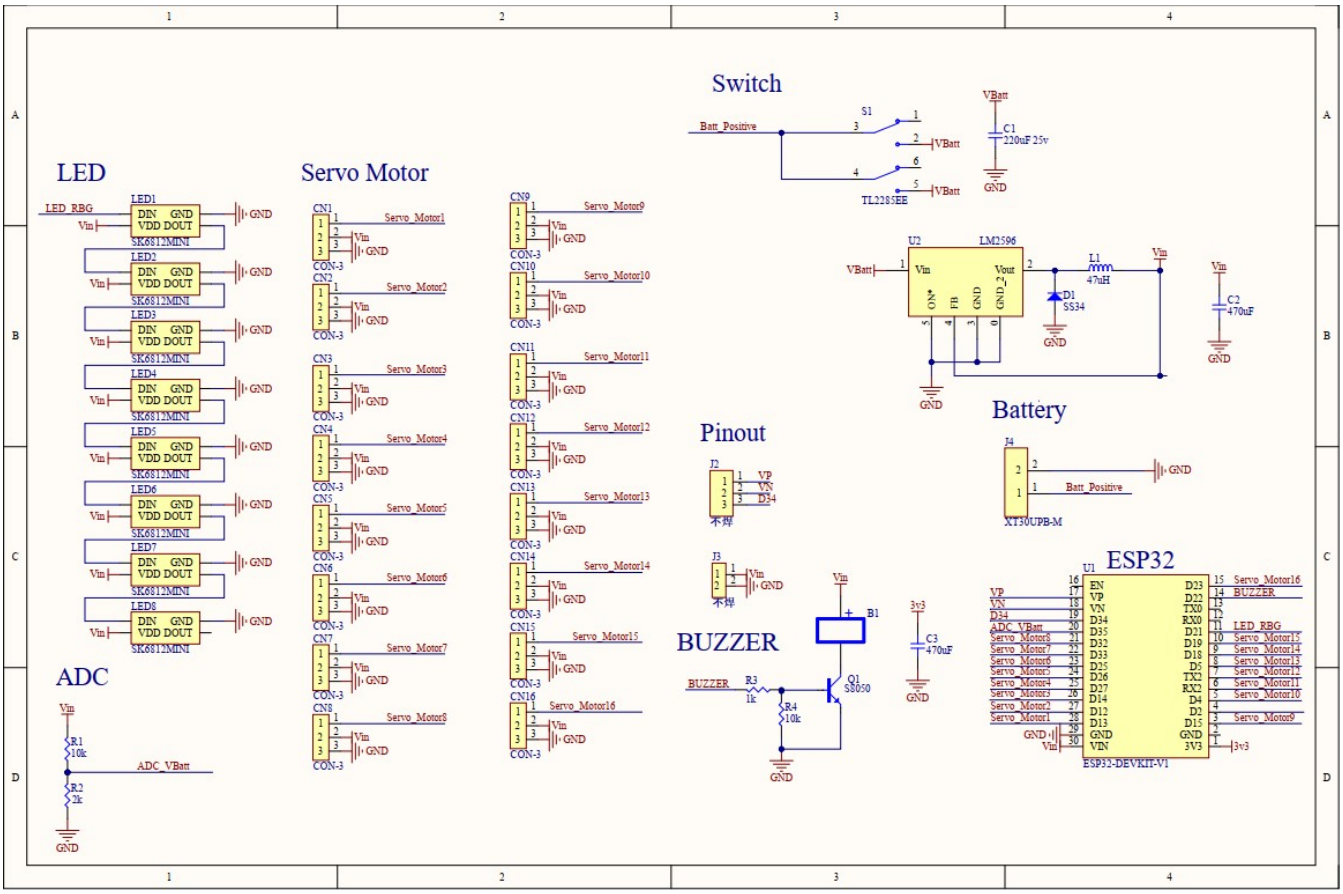


Figure 2: Schemaitc of iSEB Expansion Board 1200 0012 V1.3

2.2 Pinout

Pin	Function	Pin	Function
EN	Enable Pin	D23	CN16
VP	Unused	D22	Buzzer
VN	Unused	TX0	TX0
D34	Unused	RX0	RX0
D35	ADC Vbatt	D21	RGB Led
D32	CN8	D19	CN15
D33	CN7	D18	CN14
D25	CN6	D05	CN13
D26	CN5	D17	CN12
D27	CN4	D16	CN11
D14	CN3	D04	CN10
D12	CN2	D02	None
D13	CN1	D15	CN9

Table 1: Pinout

2.2 PCB Layout

The following is the figure of the iSEB Expansion Board 1200 0012 V1.0

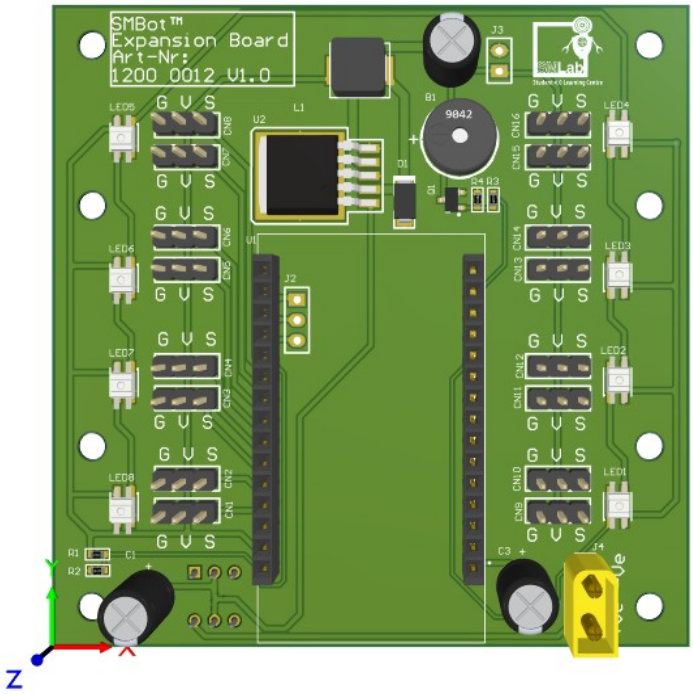


Figure 3: iSEB Expansion Board 1200 0012 V1.0 without ESP32 Module

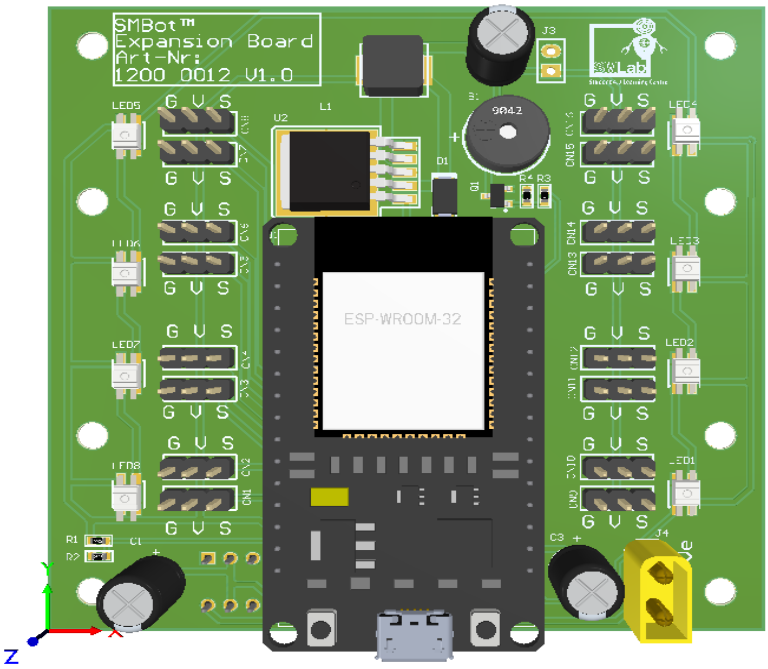


Figure 4: iSEB Expansion Board 1200 0012 V1.0 with ESP32 Module

2.2.1 Label of legs

The following figure is labeling the parts of the SMLab iSEB RobotArm.

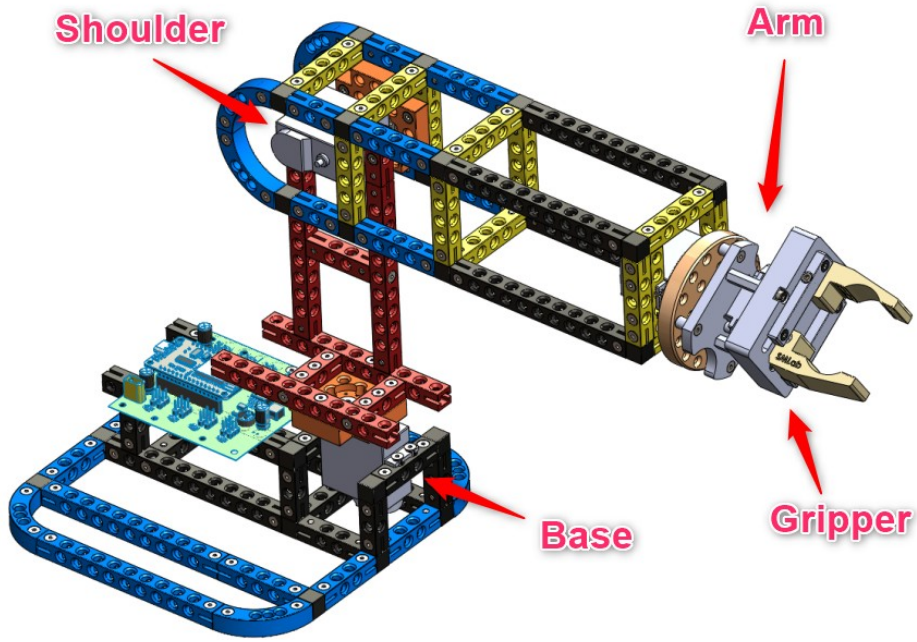


Figure 5: Labelling of iSEB Robot Arm

2.2.2 PWM control

There are 16 PWM control port in iSEB Expansion Board 1200 0012 V1.0. The figure below is showing the locaiton of the 16 PWM control port.

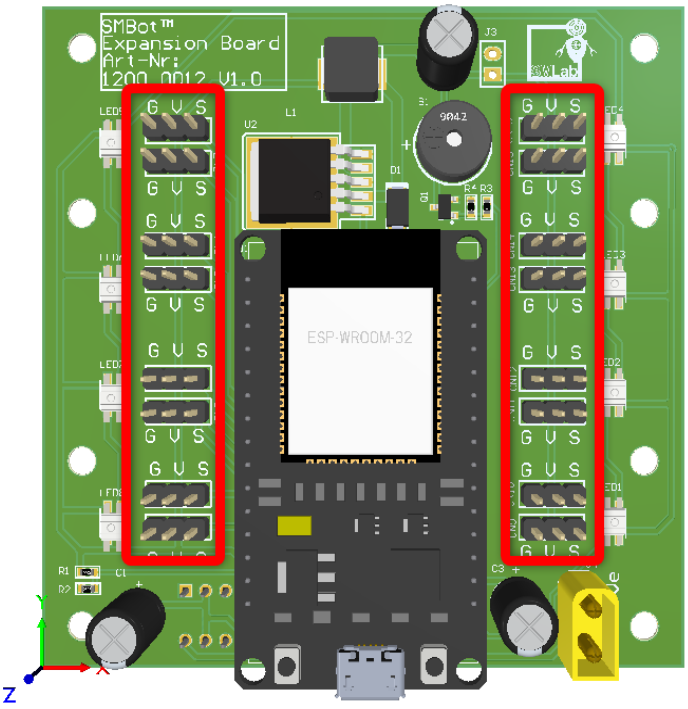
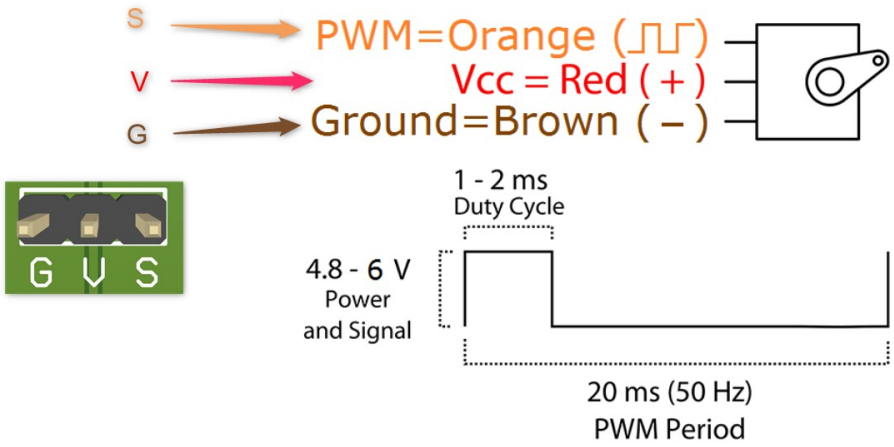


Figure 6: PWM control port

2.2.2.1 PWM Control Servo Motor Connection



Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

The figure below is specifying the port for each SMLab iSEB RobotArm

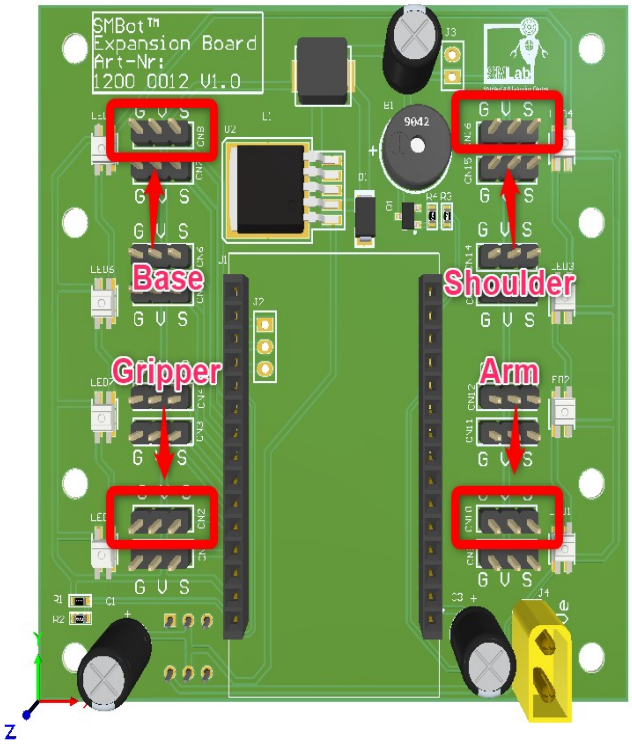


Figure 7: SMLab iSeb RobotArm

2.2.3 Battery Connector & RGB Led

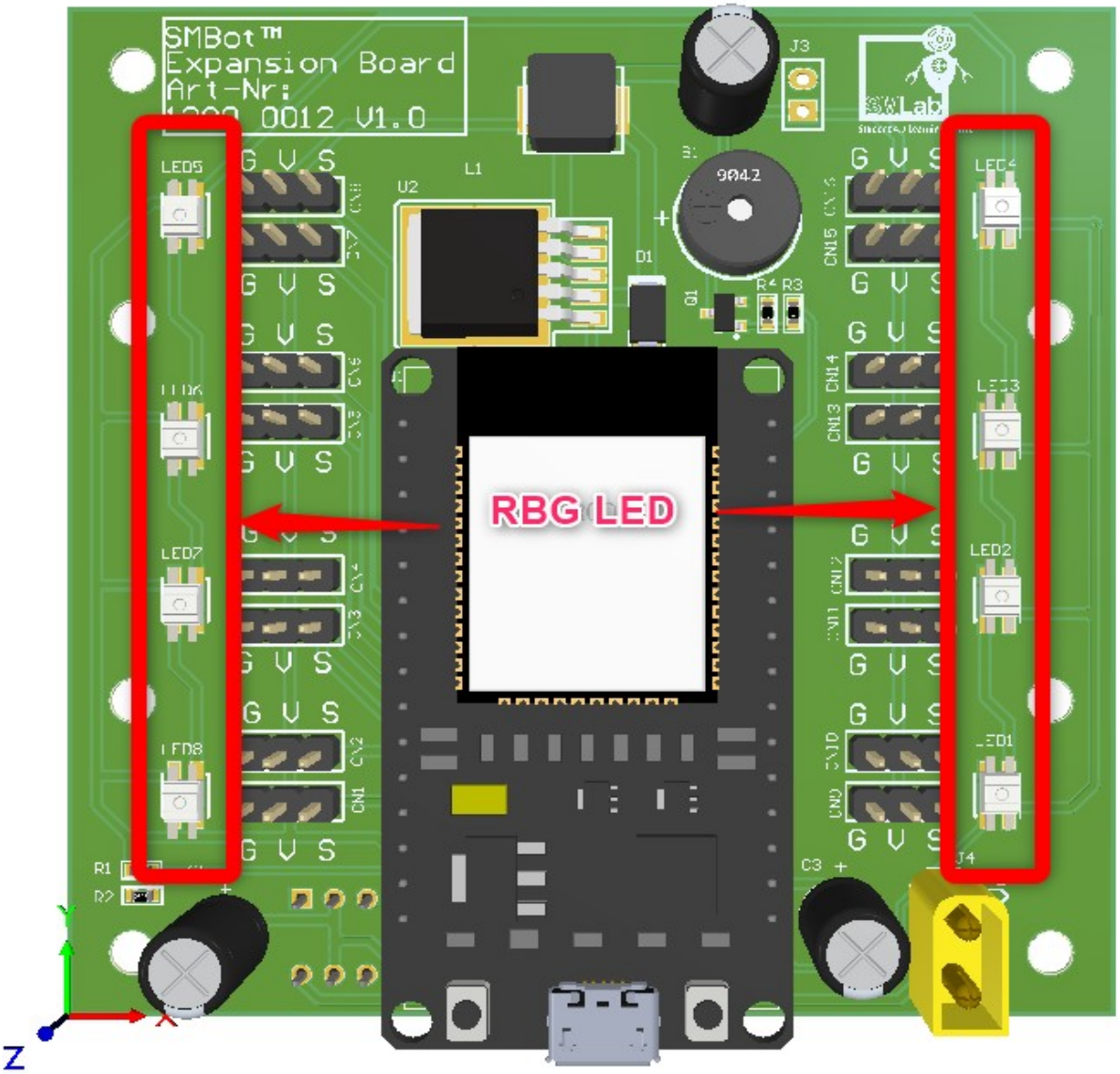


Figure 8: Battery connector & RGB led

The figure above showing Battery Connector and RGB Led. Battery connector is for battery to connect to provide power supply. The RBG led is SK6812MINI. It is a smart LD control circuit and light emitting circuit in one controller LED source. It able to display any color base on the combination of red , blue and green.

2.2.4 Switch

The figure below showing the switch .It is a latching switch. It able to cut of the battery supply.

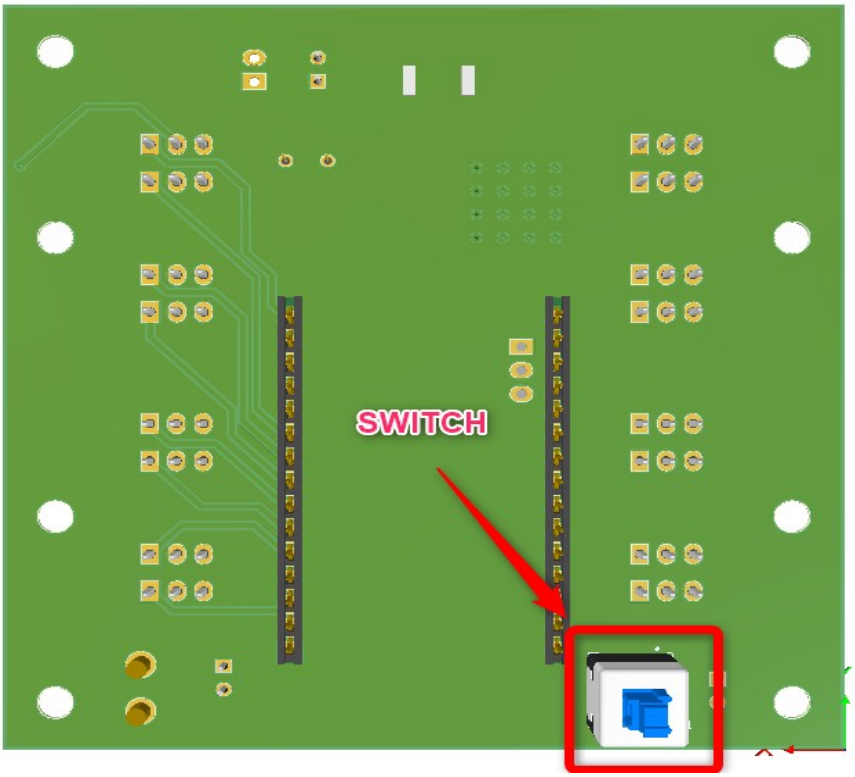


Figure 9: Battery Switch

2.2.5 LM2596 Voltage converter

LM2596 is a step down converter IC. It able to convert 3S lithium battery voltage with 12.6v to 5v. The figure below is showing the LM2596 step down converter circuit.

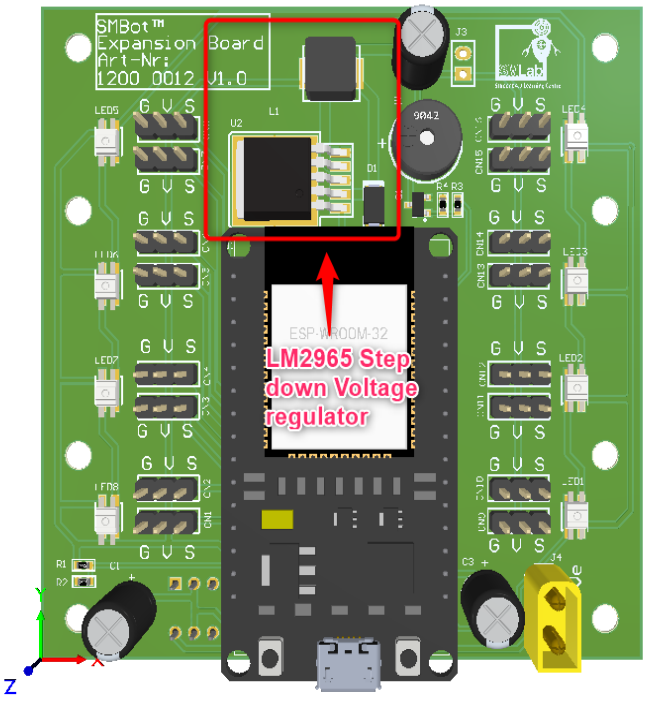


Figure 10: LM2596 voltage converter circuit

The figure below is showing the schematic of LM2596 step down converter circuit

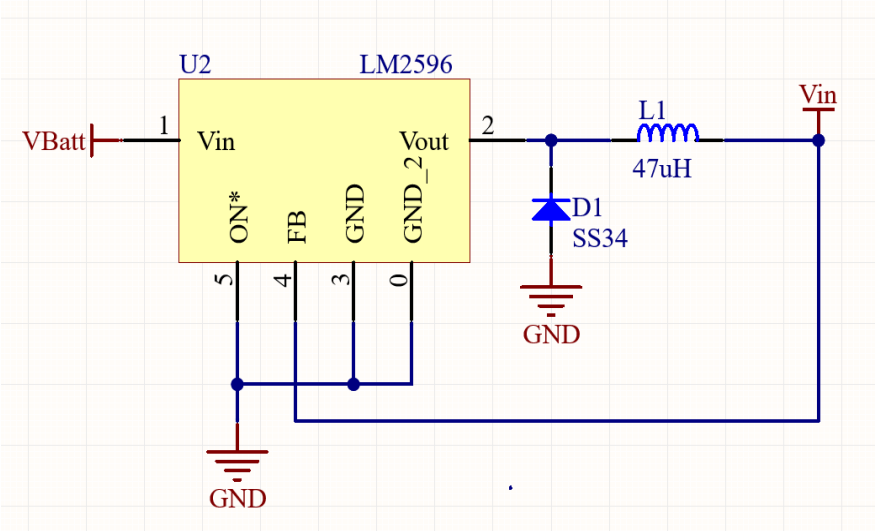


Figure 11: Schematic of LM2596 step down converter

2.2.5 Buzzer

The figur below showing the buzzer. It is a passive buzzer that able to have different tone with change the frequency of the PWM signal.

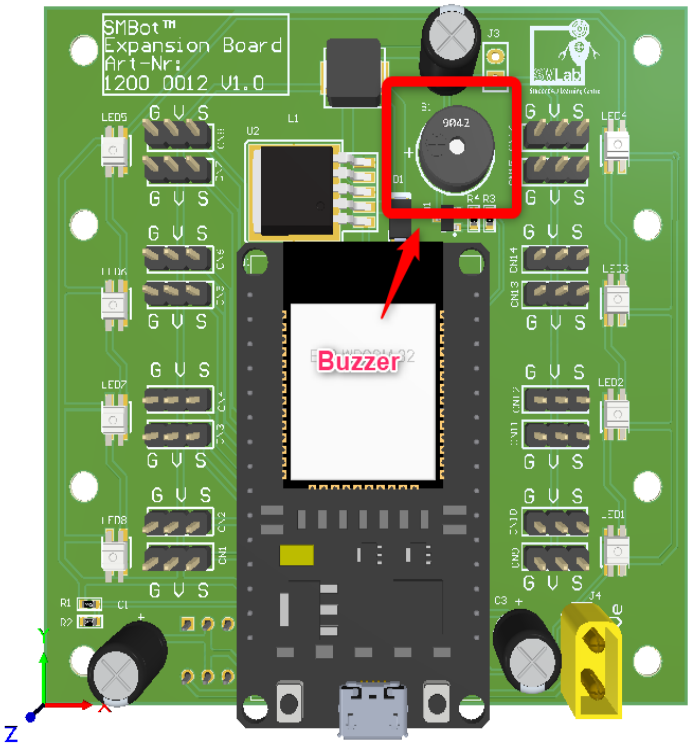


Figure 12: Buzzer

2.2.6 Capacitor and resistor

The figure below showing capacitor and resistor

The C1 is an electrolytic capacitor is a capacitor that uses an oxide film made of aluminum, tantalum or other oxidizable metal as a dielectric. The parameter of C1 is 10v 470uF . The C2 and C3 is a ceramic capacitor where the ceramic material acts as the dielectric The paramter of C2 and C3 are 25v 22uF. In this case capacitor is to prevent voltage drip and stablize the voltage.

The resistor R1 and R2 is acting as a voltage divider for ESP32 to measure the voltage of Battery through ADC. The value of R1 and R2 are 10k and 2k . We have to use resistor because ESP32 have a 12 bit ADC which only able to measure 0 to 3.3v (0 – 4095) . We add resistor to limit the current and also the voltage in order not to burn the esp32. The figure below is showin capacitors and resistors.

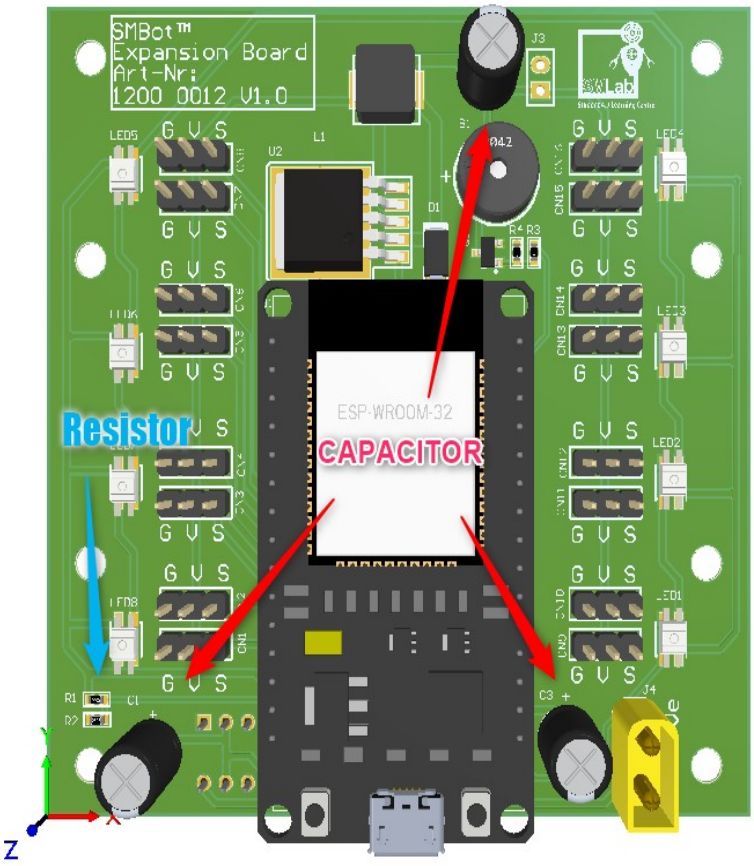


Figure 13: Capacitor and resistor

2.3 Bom list

- *iSEB Expansion Board 1200 0012 V1.0 with ESP32 Module x 1*
- TATTU 11.1V 3S 450mAh 75C LiPo Battery Pack with XT30 Plug
- *ESP32-DEVKIT-V1 x 1*
- *iSEB RobotArm Mechanical set x 1*

3 Firmware

The iSEB Expansion Board 1200 0012 V1.0 is using ESP32 DevKit V1. The figure is showing the pinout of ESP32 DevKit V1.The microcontroller is esp-wroom-32 module.

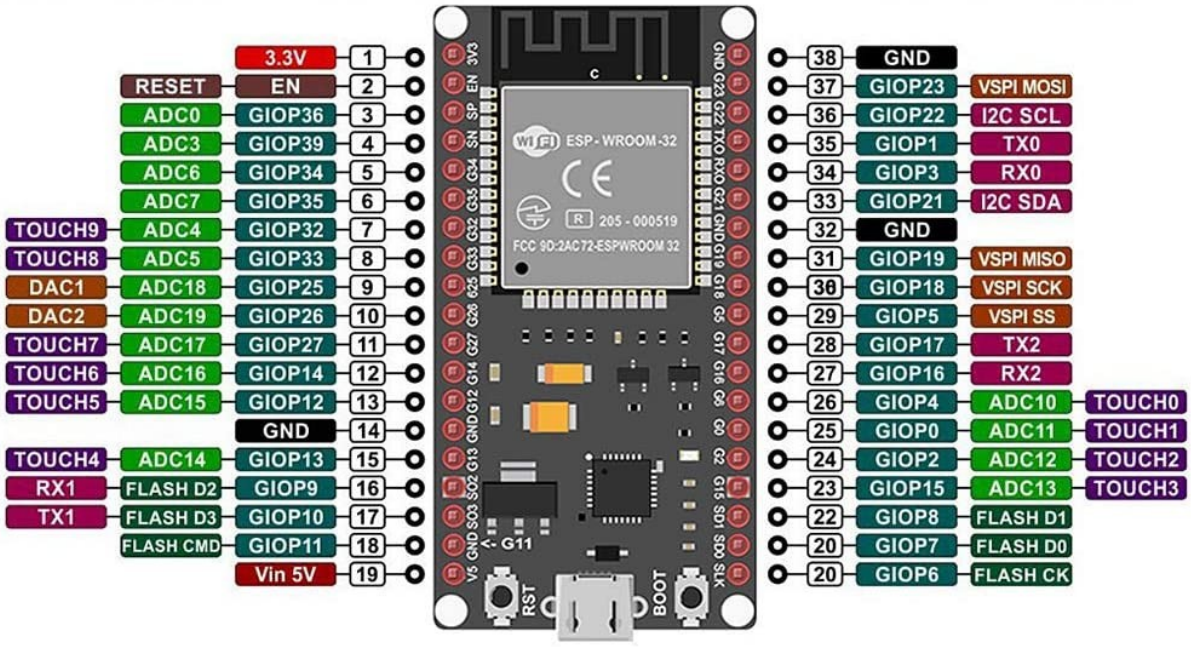


Figure 14: Pinout of ESP32 DevKit V1

3.1 Specification of the ESP32 DevKit V1

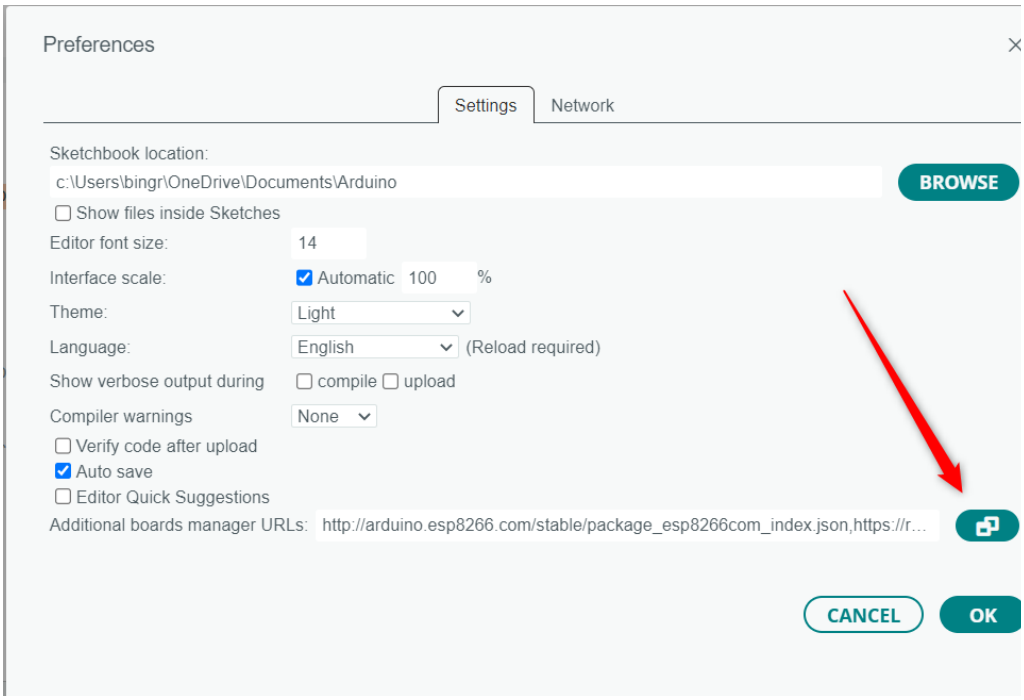
Microcontroller: Tensilica 32-bit Single-/Dual-core CPU Xtensa LX6

- Operating Voltage: 3.3V
- Input Voltage: 7-12V
- Digital I/O Pins (DIO): 25
- Analog Input Pins (ADC): 6
- Analog Outputs Pins (DAC): 2
- UARTs: 3
- SPIs: 2
- I2Cs: 3
- Flash Memory: 4 MB
- SRAM: 520 KB
- Clock Speed: 240 Mhz
- Wi-Fi: IEEE 802.11 b/g/n/e/i:
    - Integrated TR switch, balun, LNA, power amplifier and matching network
    - WEP or WPA/WPA2 authentication, or open networks
- Dimensions: 51.5x29x5mm

3.2 Environment set up

We need to set up the environment to flash the binary to ESP32 DevKit V1.

- Install Arduino IDE is requried to install. (Snapshot is base on Arduino IDE 2.2.0)
- Add https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json to Board Managers and install ESP32 libary.
  - The figure below showing how to update board managers



  - The figure showing after adding the Boards Manager URLs

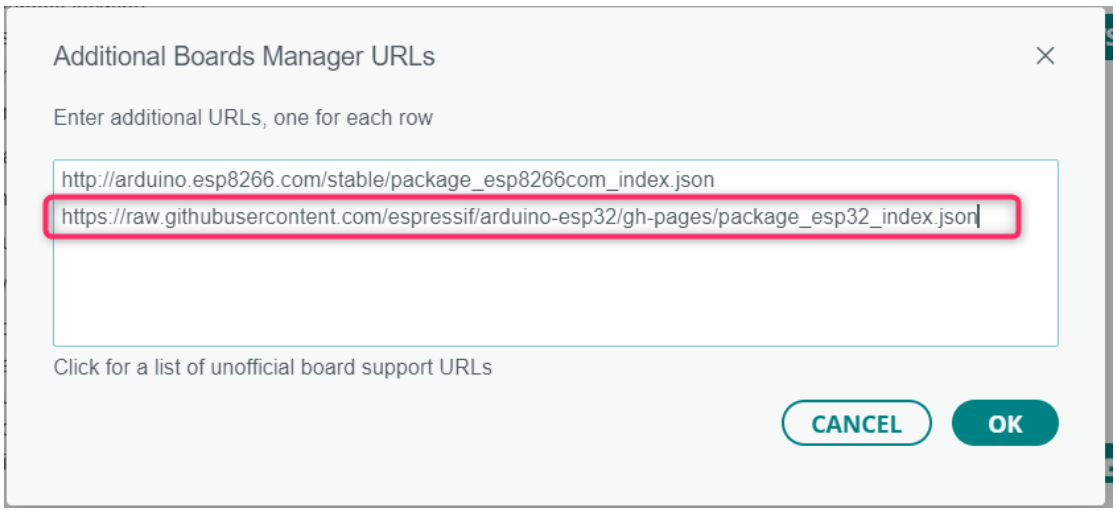


Figure 16: Adding board manager URLS

- The figure below showing how to install ESP32 by Espressif Systems at Board Manager.

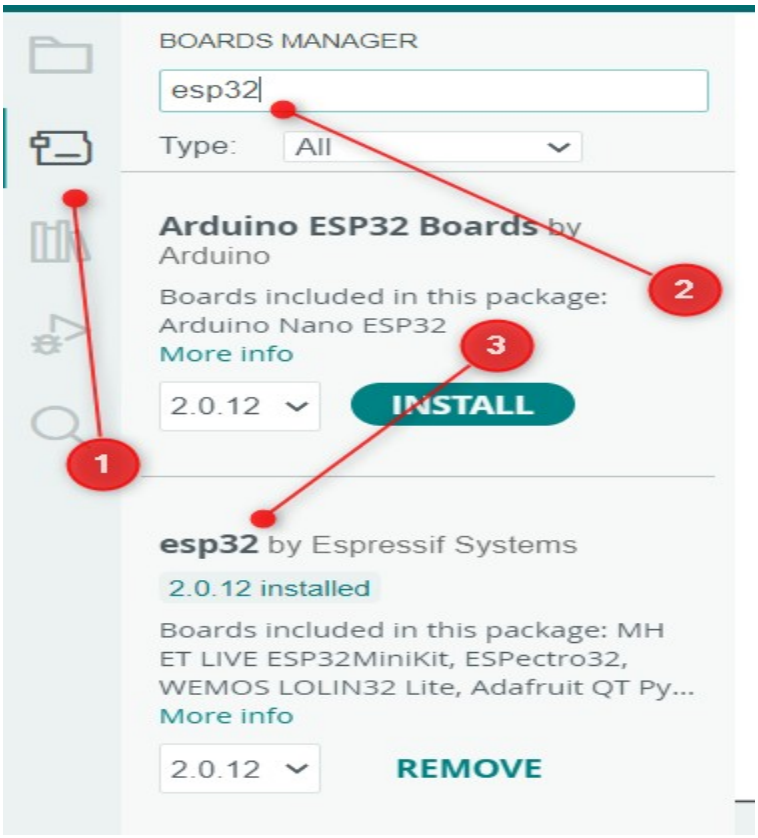


Figure 17: Install ESp32 by Espressif Systems at Board Managers

- Install WS2812FX by Harm Aldick (version 1.4.2) library.
- The figure below showing how to install WS2812FX libary

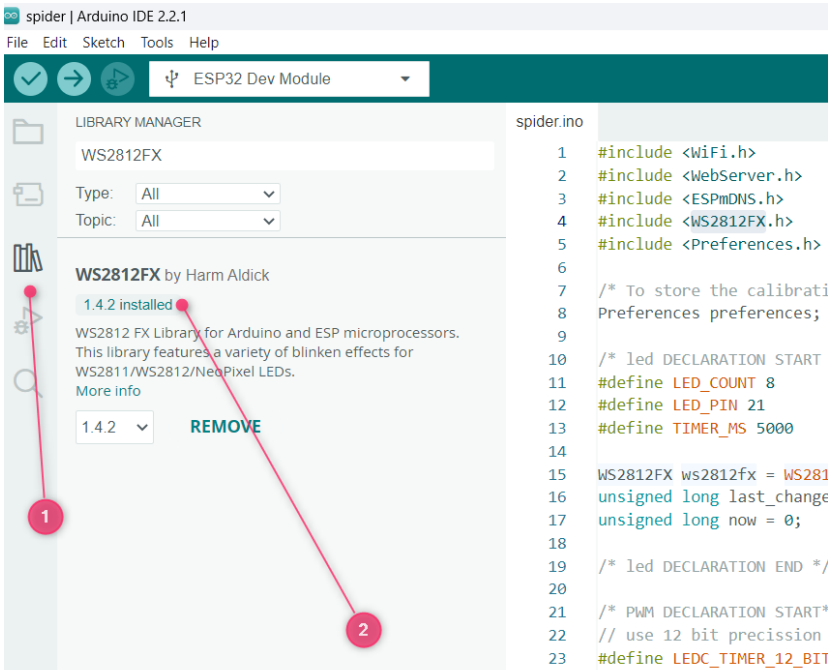


Figure 18: Instal WS2812FX library

- The figure below showing how to update the upload setting

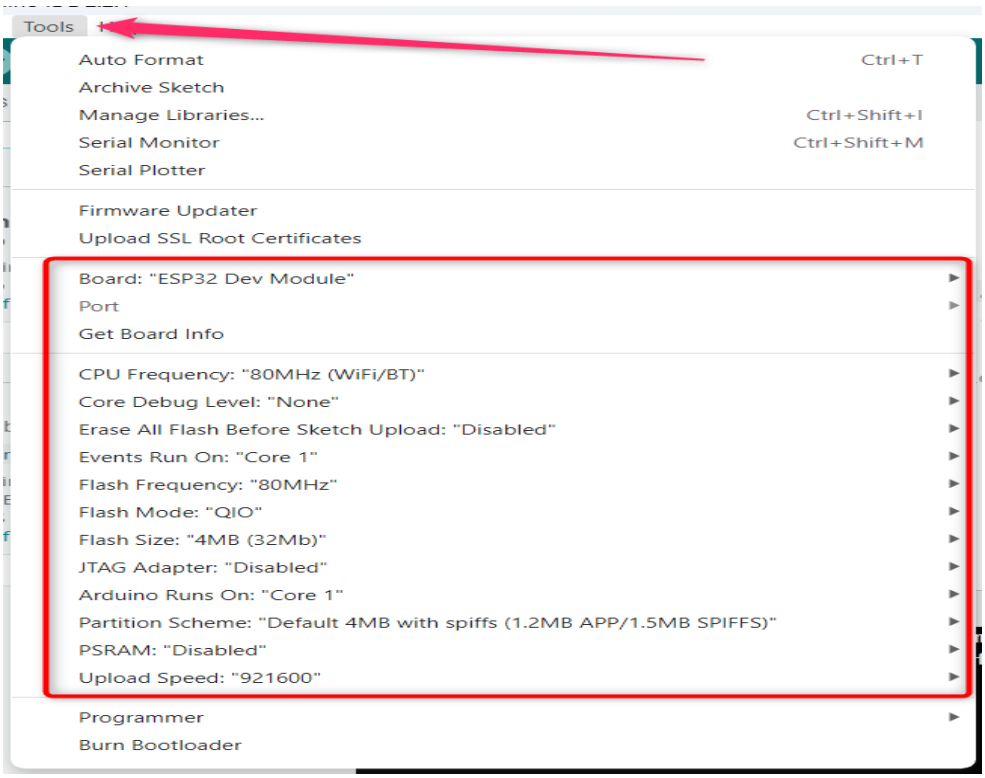


Figure 19: Upload setting

- Click upload button and the firmware will be flashed successfully if the snapshot below is seen.
- The figure below showing how to compile and upload the firmwrae

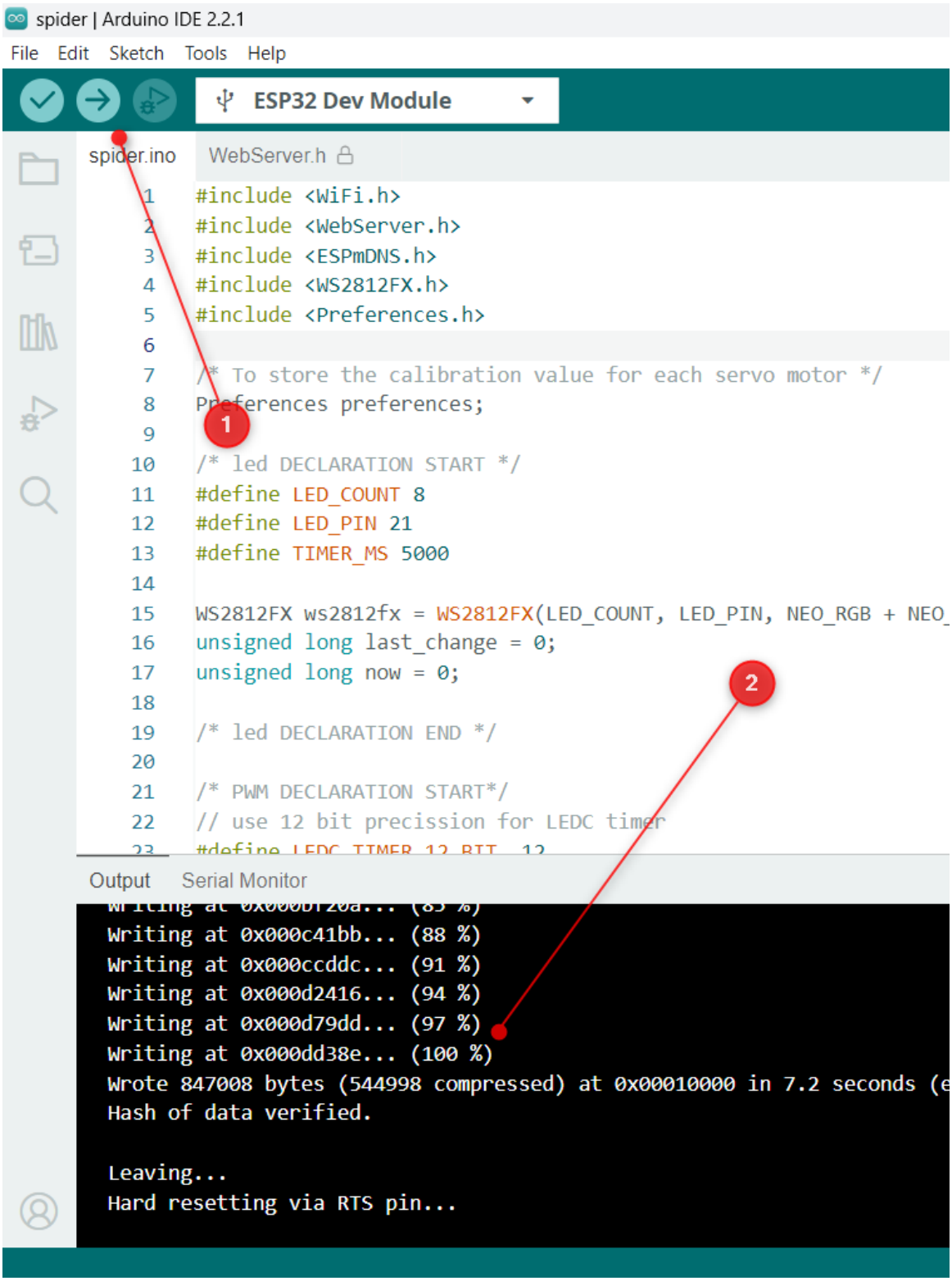


Figure 20: Compile and upload

- The environment set up is done if the binary able to flash to ESP32 DevKit V1

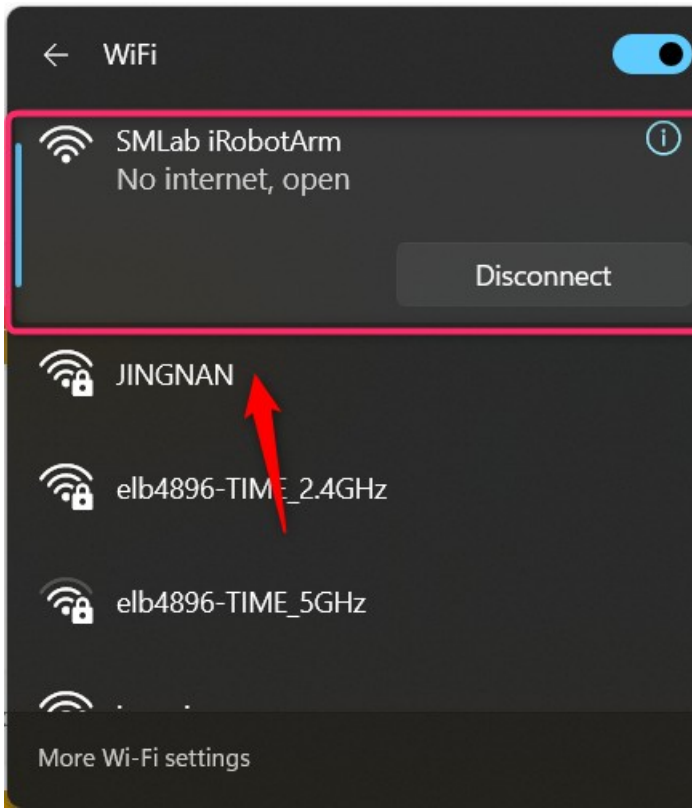
3.3 WiFi

3.3.1 How the WiFi Code works

- Firstly we need to include WiFi and WebServer library
  - `#include <WiFi.h>`
    - `WiFi.h - esp32 Wifi support.`
  - `#include <WebServer.h>`
    - WebServer.h - Dead simple web-server. Supports only one simultaneous client, knows how to handle GET and POST.
- Secondly we need to insert our ssid and password
  - `const char* ssid = "SMLab iRobotArm";  // Enter SSID here`
  - `const char* password = "12345678";  //Enter Password here`
- We able configure local ip , gateway and subnet.
  - `IPAddress local_ip(192,168,1,1);`
  - `IPAddress gateway(192,168,1,1);`
  - `IPAddress subnet(255,255,255,0);`
- Then we set our web server to port 80
  - `WebServer server(80);`
- We have to setup the WiFi in setup function
  - To start the Wi-Fi as an Access Point.
    - `WiFi.softAP(ssid);/* without password */`
    - `WiFi.softAP(ssid,password);/* with password */`
  - Function used to configure the IP as static (fixed) as well as the gateway and subnet.
    - `WiFi.softAPConfig(local_ip, gateway, subnet); /* to add exception to server */`
  - Set up handling of web page
    - `server.on("/",handleIndex);`
    - `server.on("/controller", handleController);`
  - Enable the server
    - `server.begin();`
- We have to handle the user request in loop funtion
    - `server.handleClient();`

3.3.2 WiFi server

- After flash succesfully, the iSEB RobotArm should be appear in the WiFi list. The figure below is showing the iSEB RobotArm is appeared in the WiFi list.



- The figure below showing how to access ISEB RobotArm through web broswer

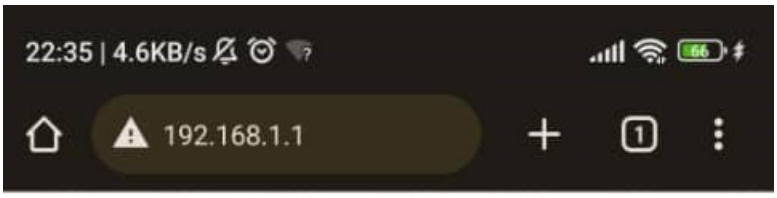


Figure 22: Access ISEB RobotArm through web broswer

3.3.2 Web Page

The figure below is showing iSEB Robot ARM webpage

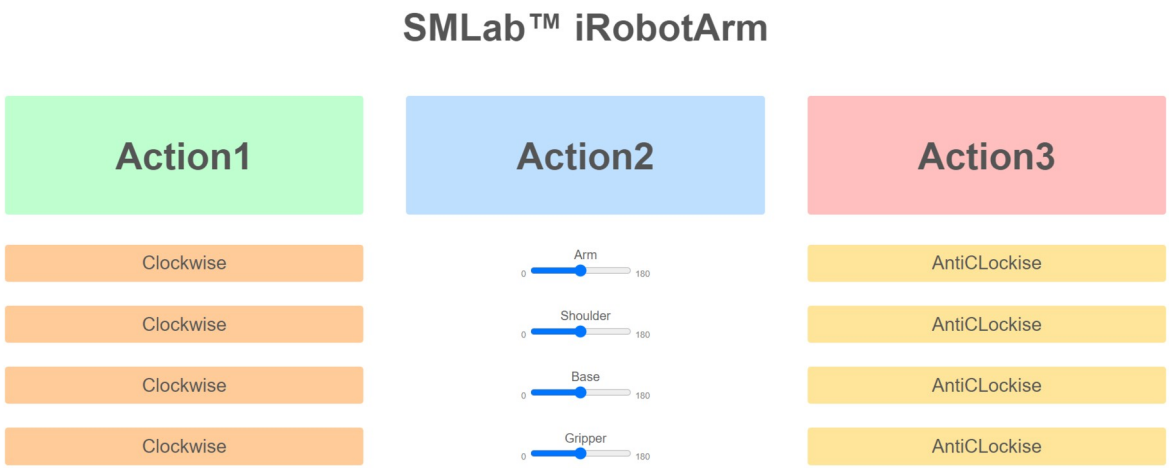


Figure 23: iSEB Robot Arm WebPage

ISEB Robot ARM's web page will have 3 action, 4 clockwise and 4 anticlockwise buttons. User can demo robot arm with 3 action buttons

For action 1 , the motion of robot arm is as below

- origin , go left 20, go down ,shake two times, grab stuff ,go up, go right 20, go down ,shake two times, relese stuff.

For action 2, the motion of robot arm is as below

- origin , go left 40, go down ,shake two times, grab stuff ,go up, go right 40, go down ,shake two times, relese stuff.

For action 3, the motion of robot arm is as below

- origin , go left 40, go down ,shake two times, grab stuff ,go up, go right 20, go down ,shake two times, relese stuff,shake two times, grab stuff ,go up, go right 20, go down ,shake two times, relese stuff,shake two times, grab stuff ,go up, go right 20, go down ,shake two times, relese stuff
-

User can manually control the 4 servo motors which are arm , shoulder , base and gripper manually though clockwise and anitclockwise buttons at the botto,.

3.3.3 How ESP32 server work?

The ESP32 module of iSEB Robot Arm is acting as a server and all the server operation is handled in server.handleClient that keep being called in loop function.

Base configuration mentioned in chapter 3.3.1, user can surf 192.168.1.1 in any web browser to call function hanldeIndex called because hanldleIndex is the default page. The string variable “content” is a html code that contructing the web page shown in Figure 23. Hence if we want to update the webpage, we need to update the variable content.

When user press action 1 , 2 and 3 button in the webpage , ESP32 will call function handleController with variable pm. User can also manually trigger function handleController by calling 192.168.1.1/controller?pm=“1” and changing the pm value to have differrent action.

When user press clock and anticlockwise button, ESP will call function handleController with variable servo and value. User can manually trigger function handleController to control specific servo motor with the 192.168.1.1/controller?servo="1"+value="90" . Servo is the identify which servo motor and value is the angle of servo motor.

3.3.4 Function handleIndex

```
void handleIndex() {

 String content = "";
 content += "<html>";
 content += "<head>";
 content += "<title>SMLab iRobotArm ™</title>";
 content += "<meta charset=UTF-8>";
 content += "<meta name=viewport content=width=device-width>";
 content += "<style type=text/css>";
 content += "body {";
 content += "margin:0px;";
 content += "backgound-color:#FFFFFF;";
 content += "font-family:helvetica,arial;";
 content += "font-size:100%;";
 content += "color: #555555;";
 content += "text-align: center;";
 content += "}";
 content += "td {";
 content += "text-align: center;";
 content += "}";
 content += "span {";
 content += "font-family:helvetica,arial;";
 content += "font-size:70%;";
 content += "color:#777777;";
 content += "}";
 content += ".button{";
 content += "width:90%;";
 content += "height:90%;";
 content += "font-family:helvetica,arial;";
 content += "font-size:100%;";
 content += "color:#555555;";
```

```
 content += "background:#BFDFFF;";
 content += "border-radius:4px;";
 content += "padding: 2px 2px 2px 2px;";
 content += "border:none;}";
 content += ".button:active{";
 content += "background-color:#999;";
 content += "color:white;}";
 content += ".button2{background-color:#BFFFCF;}";
 content += ".button3{background-color:#FFBFBF;}";
 content += ".button4{background-color:#FFCC99;}";
 content += ".button5{background-color:#FFE599;}";
 content += ".button6{background-color:#CFBFFF;}";
 content += "</style>";
 content += "</head>";
 content += "<body><h1>SMLab iRobotArm ™</h1>";
 content += "<table width=100% height=30%>";
 content += "<tr>";
 content += "<td width=33%><button class=\"button button2\"
onclick=controlPm(1)>Action1</button></td>";
 content += "<td width=33%><button class=\"button\" onclick=controlPm(2)>Action2</button></td>";
 content += "<td width=33%><button class=\"button button3\"
onclick=controlPm(3)>Action3</button></td>";
 content += "</tr>";
 content += "</table>";
 content += "<table width=100% height=50%>";
 content += "<tr><td colspan=4><span><br></span></td></tr>";
 content += "<tr>";
 content += "<td width=33%><button class=\"button button4\"
onclick=controlServo(0,'range_0',1)>Clockwise</button></td>";
 content += "<td width=33%>Arm <span><br>0 <input type=range id=range_0 min=0 max=180 value=90
onchange=controlServo(0,'range_0',0)> 180</span>";
 content += "<td width=33%><button class=\"button button5\"
onclick=controlServo(0,'range_0',2)>AntiCLockise</button></td>";
 content += "</tr>";
 content += "<tr><td colspan=4><span><br></span></td></tr>";
 content += "<tr>";
 content += "<td width=33%><button class=\"button button4\"
onclick=controlServo(1,'range_1',1)>Clockwise</button></td>";
 content += "<td width=33%>Shoulder <span><br>0 <input type=range id=range_1 min=0 max=180
value=90 onchange=controlServo(1,'range_1',0)> 180</span>";
  content += "<td width=33%><button class=\"button button5\"
onclick=controlServo(1,'range_1',2)>AntiCLockise</button></td>";
 content += "</tr>";
 content += "<tr><td colspan=4><span><br></span></td></tr>";
 content += "<tr>";
 content += "<td width=33%><button class=\"button button4\"
onclick=controlServo(2,'range_2',1)>Clockwise</button></td>";
 content += "<td width=33%>Base <span><br>0 <input type=range id=range_2 min=0 max=180 value=90
onchange=controlServo(2,'range_2',0)> 180</span>";
  content += "<td width=33%><button class=\"button button5\"
onclick=controlServo(2,'range_2',2)>AntiCLockise</button></td>";
 content += "</tr>";
 content += "<tr><td colspan=4><span><br></span></td></tr>";
```

```
 content += "<tr>";
 content += "<td width=33%><button class=\"button button4\"
onclick=controlServo(3,'range_3',1)>Clockwise</button></td>";
 content += "<td width=33%>Gripper <span><br>0 <input type=range id=range_3 min=0 max=180
value=90 onchange=controlServo(3,'range_3',0)> 180</span>";
 content += "<td width=33%><button class=\"button button5\"
onclick=controlServo(3,'range_3',2)>AntiCLockise</button></td>";
 content += "</tr>";
 content += "</table>";
 content += "</body>";
 content += "<script>";
 content += "function controlServo(id, textId,bfAdd) {";
 content += "var xhttp = new XMLHttpRequest();";
 content += "var value = document.getElementById(textId).value;";
 content += "if(1 == bfAdd) value = parseInt(value)-parseInt(\"10\");";
 content += "if(2 == bfAdd) value = parseInt(value)+parseInt(\"10\");";
 content += "if(parseInt(value) > 180 ) value = 180; ";
 content += "if(parseInt(value) < 0  ) value = 0; ";
 content += "document.querySelector('#range_' + id).value = value;";
 content += "xhttp.onreadystatechange = function() {";
 content += "if (xhttp.readyState == 4 && xhttp.status == 200) {";
 content += "}";
 content += "};";
 content += "xhttp.open(\"GET\",\"controller?servo=\"+id+\"&value=\"+value, true);";
 content += "xhttp.send();";
 content += "}";
 content += "function controlPm(id) {";
 content += "var xhttp = new XMLHttpRequest();";
 content += "xhttp.onreadystatechange = function() {";
 content += "if (xhttp.readyState == 4 && xhttp.status == 200) {";
 content += "}";
 content += "};";
 content += "xhttp.open(\"GET\", \"controller?pm=\"+id, true);";
 content += "xhttp.send();";
 content += "}";
 content += "</script>";
 content += "</html>";

 server.send(200, "text/html", content);
}
```

3.3.4 Function handlecontroller

```
void handleController()
{
 String pm = server.arg("pm");
 String servo = server.arg("servo");
 String value = server.arg("value");
 Serial.println("Controller pm: "+pm+" servo: "+servo +" value: "+value);
 if (pm != "") {
  Servo_PROGRAM = pm.toInt();
  server.send(200, "text/html", "(pm)=(" + pm + ")");
 }

 if (servo != "" && value!= "") {
  ConvertDegreeToPwmAndSetServo(servo.toInt(),value.toInt());
  server.send(200, "text/html", "servo =" + servo + " value =" + value);
 }
 server.send(200, "text/html", "Input invalid");
}
```

Variable Servo_PROGRAM is being update when function handlleController is called and variable pm is not equal to empty string. ESP32 will handle it in main loop function and will call function Servo_PROGRAM_Run to conduct a sequence of action that form action 1 , 2 and 3.

Function CovertDegreeToPwmAndSetServo is being called when handleController is called and variable servo and value is not equal to empty string. ESP will update the position of the specific servo motor base on variable servo and value.

3.4 Servo Motor

- The servo motor used in the iSEB RobotArm is TowerPro SG90 servo .
- The wire colors are Red = Battery(+) Brown = Battery(-) Orange = Signal
- The figure below show how the servo motor angle control by pwm
- Servo motor control with 50 Hz pulse width modulated (PWM) signal, which produces a pulse every 20ms.
-

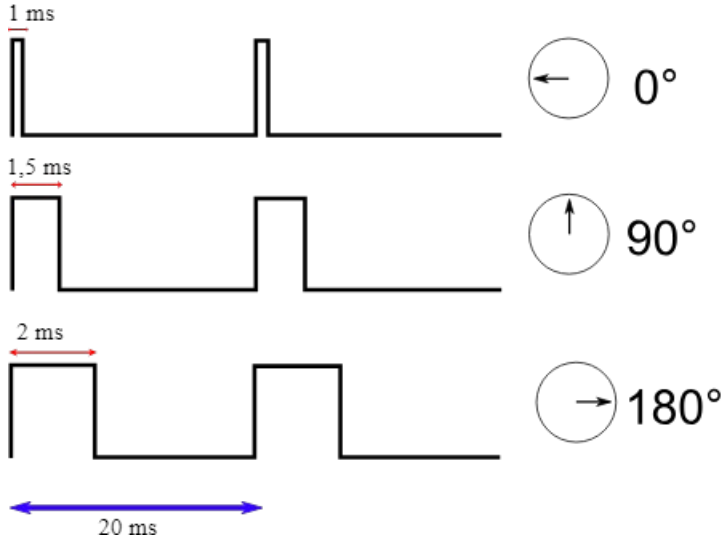


Figure 24: How servo's position controlled by PWM signal

3.4.1 How to control servo motor with ESP32

- We are using the LED Control library from ESP32 hal library to control servo motor.
- The LED control (LEDC) peripheral is primarly designed to control the intensity of LEDs, although it can also be used to generate PWM signals for other purposes. .
- For more details of the LEDC library can refer to the link
  - https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/api/ledc.html
- We able to generate PWM signals to control the servo motor.
- We have a motorInit funciton in the setup function to call the setup.
- We are calling funcition ledcSetupledc and ledcAttachPin in function motorInit.
- Function ledcSetupledc is used to setup the LEDC channel frequency and resolution.
  - uint32_t ledcSetup(uint8_t channel, uint32_t freq, uint8_t resolution_bits);
    - channel select LEDC channel to config.
      - ESP32 have 16 channels
    - freq select frequency of pwm.
    - resolution_bits select resolution for ledc channel.
      - range is 1-14 bits (1-20 bits for ESP32)
- Function ledcAttachPin is used to attach the pin to the LEDC channel.
  - void ledcAttachPin(uint8_t pin, uint8_t chan);
    - pin select GPIO pin.
    - chan select LEDC channel.
- The follow table is showing the GPIO vs Channel vs Connector in the example code

Part	GPIO	Channel	Connector
Arm	23	0	CN16
Shoulder	4	1	CN10
Gripper	32	2	CN8
Base	12	3	CN2
Buzzer	22	8	N/A

Table 2: Position vs GPIO vs Channel vs Connector matrix

3.4.2 Function motorInit

```
// use 12 bit precission for LEDC timer
#define LEDC_TIMER_12_BIT  12

// use 50 Hz as a LEDC base frequency
#define LEDC_BASE_FREQ     50

#define ARM_CHANNEL         0   /* Chanel 0 */
#define SHOULDER_CHANNEL    1   /* Chanel 1 */
#define GRIPPER_CHANNEL     2   /* Chanel 2 */
#define BASE_CHANNEL        3   /* Chanel 3 */
#define BUZZER_PWM          8 /* Channel 8 */

#define ARM_PIN             23  /* PIN 23 */
#define SHOULDER_PIN        4   /* PIN  4 */
#define GRIPPER_PIN         32  /* PIN 32 */
#define BASE_PIN            12  /* PIN 12 */
#define buzzerPin           22  /* PIN 22 */

void motorInit()

{
  // Set base frequency and resolution for all channels
 ledcSetup(0, LEDC_BASE_FREQ, LEDC_TIMER_12_BIT);
 ledcSetup(1, LEDC_BASE_FREQ, LEDC_TIMER_12_BIT);
 ledcSetup(2, LEDC_BASE_FREQ, LEDC_TIMER_12_BIT);
 ledcSetup(3, LEDC_BASE_FREQ, LEDC_TIMER_12_BIT);
 ledcSetup(BUZZER_PWM, LEDC_BASE_FREQ, LEDC_TIMER_12_BIT);
 // Attach each servo motor pin to a channel
 ledcAttachPin(ARM_PIN       , ARM_CHANNEL);       /* ARM */     /* CN0  *//* PIN 32*/
 ledcAttachPin(SHOULDER_PIN  , SHOULDER_CHANNEL);  /* SHOULDER *//* CN1 *//* PIN  4*/
 ledcAttachPin(BASE_PIN      , BASE_CHANNEL);      /* BASE */    /* CN2 *//* PIN 23*/
 ledcAttachPin(GRIPPER_PIN   , GRIPPER_CHANNEL);   /* GRIPPER */ /* CN3  *//* PIN 12*/
 ledcAttachPin(buzzerPin     , BUZZER_PWM);
  delay(50);
}
```

- From the code above we have set up pwm channel 0 to 7 to 50hz frequeny with resolution 12 bit with function ledcSetup
- We have assign GPIO pin to the pwm channel accordingly with function ledcAttachPin.

3.4.3 Function ConvertDegreeToPwmAndSetServo

```
void ConvertDegreeToPwmAndSetServo(int iServo, int iValue)
{
 Serial.print(F("iServo: "));
 Serial.print(iServo);
 Serial.print(F(" iValue: "));
 Serial.println(iValue);
 // Read from EEPROM to fix zero error reading
 iValue = (iValue*(MAX-MIN)/180.0)+MIN; /* convertion to pwm value */
 double NewPWM = iValue + preferences.getDouble((String(iServo)).c_str(),0);
 Serial.print(F(" NewPWM: "));
 Serial.println(NewPWM);
 /* 50 = zero degree 550 = 180 degree*/
 ledcWrite(iServo,NewPWM);
}
```

- ESP32 will output pwm signal after we configure the frequenc , resolutoin to the pwm channel and assign the GPIO pin to each pwm channel.
- We can call LEDCWrite to update the duty cycle of the particular pwm channel.
- By updating duty cycle we can control the positoin of servo motor mention chalter 3.4
- Functoin ledcWrite is used to set duty for the LEDC channel.
  - void ledcWrite(uint8_t chan, uint32_t duty);
    - chan select the LEDC channel for writing duty.
    - duty select duty to be set for selected channel.
- In the example code, we have set the resolution bit to 12 bit hence there are 4095 steps for the reoslution.
- By calculation we set 409 to acheive 1ms duty cycle and 819 to achieve 2ms duty cycle.
- However the example we set min to 50 min and maximum to 550 due to base on testing the servo motor only react between 50 and 550 (will further investigate on this issue suspect is due to servo motor but yet to confirm with scope).
- For the servo postion array such as Servo_Prg_X, the position is store as position therefore a positoin convert to duty cycle is needed.
- Function Set_PWM_to_Servo is to convert the position to duty cycle and update to the pwm channel
  - void Set_PWM_to_Servo(int iServo, int iValue)
    - iServo select the LEDC channel for writing duty.
    - Ivalue select the position to convert to duty tobe set for selected channel.
- We have printed the input parameter iServo , iValue and NewPWM for debug purpose.
- We have do conversion for iValue from position to duty cycle
- We have done the zero error calibration but currently not in use the value will always be zero.
- We will update the pwm channel value with ledcWrite.

3.4.4 Function Servo_PROGRAM_Run

```
// Action 1
int Servo_Prg_1_Step = 14;
int Servo_Prg_1 [][ALLMATRIX] PROGMEM = {
 //ARM,SHOULDER,BASE,GRIPPER, ms
 {  90,  90,  90,  90,   500  }, // origin
 {  90,  90,  60,  90,   1000  }, // go left 20
 {  90,  50,  60,  90,   1000  }, // go down
 {  50,  50,  60,  90,   1000  }, // shake 1
 { 130,  50,  60,  90,   1000  }, // shake 2
 { 130,  50,  60, 130,   1000  }, // grab
 {  90,  50,  60, 130,   1000  }, // arm go original
 {  90,  90,  60, 130,   1000  }, // go up
 {  90,  90,  90, 130,   1000  }, // go right 20
 {  90,  50,  90, 130,   1000  }, // go down
 { 130,  50,  90, 130,   1000  }, // shake 1
 {  50,  50,  90, 130,   1000  }, // shake 2
 {  50,  50,  90,  90,   1000  }, // release
 {  90,  90,  90,  90,   2000  }, // origin
};

const int BASEDELAYTIME = 20; // 10 ms

int Running_Servo_POS [ALLMATRIX] = {}; // servo motor current position

void Servo_PROGRAM_Run(int iMatrix[][ALLMATRIX], int iSteps)

{
 int INT_TEMP_A, INT_TEMP_B, INT_TEMP_C;

 for (int MainLoopIndex = 0; MainLoopIndex < iSteps; MainLoopIndex++) { // iSteps number of step
  Serial.print(F(" iSteps: "));
  Serial.println(iSteps);
  int InterTotalTime = iMatrix[MainLoopIndex][ALLMATRIX - 1]; // InterTotalTime - total time
needed

  int InterDelayCounter = InterTotalTime / BASEDELAYTIME; // InterDelayCounter time / step

  for (int InterStepLoop = 0; InterStepLoop < InterDelayCounter; InterStepLoop++) {

   for (int ServoIndex = 0; ServoIndex < ALLSERVOS; ServoIndex++) {

    INT_TEMP_A = Running_Servo_POS[ServoIndex]; // servo motor current position
    INT_TEMP_B = iMatrix[MainLoopIndex][ServoIndex]; // servo motor next position

    if (INT_TEMP_A == INT_TEMP_B) { // no update in servo motor position
     INT_TEMP_C = INT_TEMP_B;
    } else if (INT_TEMP_A > INT_TEMP_B) { // servo motor position position reduce
     INT_TEMP_C =  map(BASEDELAYTIME * InterStepLoop, 0, InterTotalTime, 0, INT_TEMP_A -
INT_TEMP_B);
     if (INT_TEMP_A - INT_TEMP_C >= INT_TEMP_B) {
      ConvertDegreeToPwmAndSetServo(ServoIndex, INT_TEMP_A - INT_TEMP_C);
```

```
      }
     } else if (INT_TEMP_A < INT_TEMP_B) { /// servo motor position position increase
      INT_TEMP_C =  map(BASEDELAYTIME * InterStepLoop, 0, InterTotalTime, 0, INT_TEMP_B -
INT_TEMP_A);
      if (INT_TEMP_A + INT_TEMP_C <= INT_TEMP_B) {
       ConvertDegreeToPwmAndSetServo(ServoIndex, INT_TEMP_A + INT_TEMP_C);
      }
     }

    }
    delay(BASEDELAYTIME);
   }

   // back of current servo motor position
   for (int Index = 0; Index < ALLMATRIX; Index++) {
    Running_Servo_POS[Index] = iMatrix[MainLoopIndex][Index];
   }
  }
}
```

- When user press button action 1 , ESP32 actually will call Servo_PROGRAM_Run and passing ptr of array Servo_Prg_1 and variable Servo_Prg_1_Step.
- The aray Servo_Prg_1 is a 5 x 14 2d array and Servo_Prg_1_Step indicate the number of the array in th array of the ptr of array (for this case Servo_Prg_1) which is 14.
- Servo_PROGRAM_Run will update the all the servo base on the value in the Servo_Prg_1 with number of step and BASEDELAYTIME delay.
- The number of step is define by time delay / BASEDELAYTIME.
- In the first array of Servo_Prg_1 the time delay is 500. So the number of steps is 500/20 = 25.
- The fucntion will adjust the position of servo motor to targer position 25 times gradually every BASEDELAYTIME instead of immediatly set the position of servo motor to the target position.
- This function can smoothen the motion of iSEB Robot Arm.