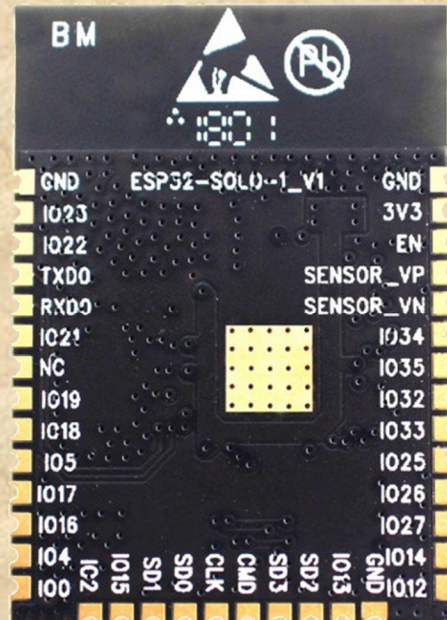


## ESP32를 이용한 BLE HID JOYSTICK





## - 목 차 -

■ BLE HID .....	3
■ ESP32 - HID 키보드.....	3
■ ESP32 - 버튼을 이용한 키보드 제어 .....	15
■ ESP32 - 조이스틱으로 인식 시키기.....	17
■ ESP32를 이용한 MATRIX KEYPAD 구동.....	21
■ ESP32를 이용한 조이스틱.....	24
■ ESP82 MCU PIN MAP .....	29
■ ESP82 MCU 사용 가능 핀 정리.....	29
■ ESP32 개발보드 만들기 .....	31
■ PCB설계.....	32
■ PCB도착 및 부속 실장.....	34
■ PS2형식 리모컨 테스트 하기 .....	36
■ PS2형식 리모컨 조이스틱으로 사용하기 .....	39

# ESP32를 이용한 BLE HID JOYSTICK

## ■ BLE HID

## ■ ESP32 - HID 키보드

ESP32는 블루투스를 이용한 HID Keyboard BLE를 지원을 합니다.

이와 관련된 테스트를 해보도록 하겠습니다.

ESP32의 BLE 예제에는 HID와 관련된 예제가 없습니다.

따라서 인터넷을 검색해서 아래 링크의 HID Keyboard BLE와 관련된 예제를 찾았습니다.

<https://github.com/nkolban/esp32-snippets/issues/230>

코드는 중간 정도에 있는 코드를 사용했습니다.

키보드로 Hello world from esp32 hid keyboard!!! 를 입력하는 예입니다.

```

123 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
45 67890

1
2 #include <BLEDevice.h>
3 #include <BLEUtils.h>
4 #include <BLEServer.h>
5 #include "BLE2902.h"
6 #include "BLEHIDDevice.h"
7 #include "HIDTypes.h"
8 #include "HIDKeyboardTypes.h"
9 #include <driver/adc.h>
10
11 BLEHIDDevice* hid;
12 BLECharacteristic* input;
13 BLECharacteristic* output;
14
15 uint8_t buttons = 0;
16 uint8_t button1 = 0;
17 uint8_t button2 = 0;
18 uint8_t button3 = 0;
19 bool connected = false;
20
21 class MyCallbacks : public BLEServerCallbacks {
22     void onConnect(BLEServer* pServer){
23         connected = true;
24         BLE2902* desc = (BLE2902*)input->getDescriptorByUUID(BLEUUID((uint16_t)0x2902));
25         desc->setNotifications(true);
26     }
27
28     void onDisconnect(BLEServer* pServer){
29         connected = false;
30         BLE2902* desc = (BLE2902*)input->getDescriptorByUUID(BLEUUID((uint16_t)0x2902));
31         desc->setNotifications(false);
32     }
33 };
34
35 /*

```



```

36      * This callback is connect with output report. In keyboard output report report special keys changes, like CAPSLOCK, NUMLOCK
37      * We can add digital pins with LED to show status
38      * bit 0 - NUM LOCK
39      * bit 1 - CAPS LOCK
40      * bit 2 - SCROLL LOCK
41      */
42      class MyOutputCallbacks : public BLECharacteristicCallbacks {
43      void onWrite(BLECharacteristic* me){
44          uint8_t* value = (uint8_t*)(me->getValue().c_str());
45          ESP_LOGI(LOG_TAG, "special keys: %d", *value);
46      }
47      };
48
49      void taskServer(void*){
50
51
52          BLEDevice::init("ESP32-keyboard");
53          BLEServer *pServer = BLEDevice::createServer();
54          pServer->setCallbacks(new MyCallbacks());
55
56          hid = new BLEHIDDevice(pServer);
57          input = hid->inputReport(1); // <-- input REPORTID from report map
58          output = hid->outputReport(1); // <-- output REPORTID from report map
59
60          output->setCallbacks(new MyOutputCallbacks());
61
62          std::string name = "chegewara";
63          hid->manufacturer()->setValue(name);
64
65          hid->pnp(0x02, 0xe502, 0xa111, 0x0210);
66          hid->hidInfo(0x00, 0x02);
67
68          BLESecurity *pSecurity = new BLESecurity();
69          // pSecurity->setKeySize();
70          pSecurity->setAuthenticationMode(ESP_LE_AUTH_BOND);
71
72          const uint8_t report[] = {
73              USAGE_PAGE(1),      0x01,      // Generic Desktop Ctrl's
74              USAGE(1),           0x06,      // Keyboard
75              COLLECTION(1),      0x01,      // Application
76              REPORT_ID(1),       0x01,      // Report ID (1)
77              USAGE_PAGE(1),      0x07,      // Kbrd/Keypad
78              USAGE_MINIMUM(1),   0xE0,
79              USAGE_MAXIMUM(1),   0xE7,
80              LOGICAL_MINIMUM(1), 0x00,
81              LOGICAL_MAXIMUM(1), 0x01,
82              REPORT_SIZE(1),     0x01,      // 1 byte (Modifier)
83              REPORT_COUNT(1),    0x08,
84              HIDINPUT(1),        0x02,      // Data, Var, Abs, No Wrap, Linear, Preferred State, No
II Position
85              REPORT_COUNT(1),    0x01,      // 1 byte (Reserved)
86              REPORT_SIZE(1),     0x08,
87              HIDINPUT(1),        0x01,      // Const, Array, Abs, No Wrap, Linear, Preferred State, No
Null Position
88              REPORT_COUNT(1),    0x06,      // 6 bytes (Keys)
89              REPORT_SIZE(1),     0x08,
90              LOGICAL_MINIMUM(1), 0x00,
91              LOGICAL_MAXIMUM(1), 0x65,      // 101 keys
92              USAGE_MINIMUM(1),   0x00,
93              USAGE_MAXIMUM(1),   0x65,
94              HIDINPUT(1),        0x00,      // Data, Array, Abs, No Wrap, Linear, Preferred State, No
Null Position

```

```

95      REPORT_COUNT(1),    0x05,    // 5 bits (Num Lock, Caps Lock, Scroll Lock, Compose, K
ana)
96      REPORT_SIZE(1),    0x01,
97      USAGE_PAGE(1),    0x08,    // LEDs
98      USAGE_MINIMUM(1),  0x01,    // Num Lock
99      USAGE_MAXIMUM(1),  0x05,    // Kana
100     HIDOUTPUT(1),        0x02,    // Data, Var, Abs, No Wrap, Linear, Preferred State, No Nu
ll Position, Non-volatile
101     REPORT_COUNT(1),    0x01,    // 3 bits (Padding)
102     REPORT_SIZE(1),    0x03,
103     HIDOUTPUT(1),        0x01,    // Const, Array, Abs, No Wrap, Linear, Preferred State, No
Null Position, Non-volatile
104     END_COLLECTION(0)
105 };
106
107     hid->reportMap((uint8_t*)report, sizeof(report));
108     hid->startServices();
109
110     BLEAdvertising *pAdvertising = pServer->getAdvertising();
111     pAdvertising->setAppearance(HID_KEYBOARD);
112     pAdvertising->addServiceUUID(hid->hidService->getUUID());
113     pAdvertising->start();
114     hid->setBatteryLevel(7);
115
116     ESP_LOGD(LOG_TAG, "Advertising started!");
117     delay(portMAX_DELAY);
118
119 };
120
121 void setup() {
122     Serial.begin(115200);
123     Serial.println("Starting BLE work!");
124
125     pinMode(38, INPUT_PULLDOWN);
126     attachInterrupt(digitalPinToInterrupt(38), clickNumLock, CHANGE); // Num Lock
127     pinMode(39, INPUT_PULLDOWN);
128     attachInterrupt(digitalPinToInterrupt(39), clickCapsLock, CHANGE); // Caps Lock
129     pinMode(37, INPUT_PULLDOWN);
130     attachInterrupt(digitalPinToInterrupt(37), clickScrollLock, CHANGE); // Scroll Lock
131
132     xTaskCreate(taskServer, "server", 20000, NULL, 5, NULL);
133 }
134
135 void loop() {
136
137     if(connected){
138
139         vTaskDelay(5000);
140         const char* hello = "Hello world from esp32 hid keyboard!!!\n";
141
142         while(*hello){
143             KEYMAP map = keymap[(uint8_t)*hello];
144             Serial.println(buttons);
145             uint8_t msg[] = {map.modifier | buttons, 0x0, map.usage, 0x0, 0x0, 0x0, 0x0}; // <---
fixed line
146             input->setValue(msg, sizeof(msg));
147             input->notify();
148             hello++;
149             uint8_t msg1[] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
150
151             input->setValue(msg1, sizeof(msg1));
152             input->notify();
153             delay(10);

```



```

154     }
155 }
156 delay(50);
157 }
158
159 #RAM_ATTR void clickNumLock(){
160     button1 = buttons&0x01;
161     button1 != button1;
162     buttons = button1<<0;
163 }
164 #RAM_ATTR void clickCapsLock(){
165     button2 = buttons&0x02;
166     button2 != button2;
167     buttons = button2<<1;
168 }
169 #RAM_ATTR void clickScrollLock(){
170     button3 = buttons&0x04;
171     button3 != button3;
172     buttons = button3<<2;
173 }

```

관련 자료 : esp32\_3 , original\_code

윗 자료를 자료를 업로드 하고 스마트폰등을 이용하여 블루투스를 검색하면 “ ESP32-keyboard” 가 검색이 되고 이를 페어링 하면 “ Hello world from esp32 hid keyboard!!!”

문자가 자동으로 반복해서 입력되는 것을 알 수 있습니다.

우선 구동이 되는 것 같기는 하지만 입력 내용을 보면 조금씩 오류도 있어 보이고 완벽해 보이지는 않습니다.

소스코드를 보면 관련 지식이 없는 사람은 알아보기 힘든 정도로 어렵게 되어 있습니다. 그래도 볼수 있는 부분만 체크해 봤습니다.

키보드 값을 전송하는 방법은 msg라는 배열에 전송할 데이터를 넣고 BLECharacteristic\* 허의 input class에 setValue로 값을 설정 후 notify를 하면 데이터를 전송하는 과정으로 보입니다.

내용중에 보면 실제 입력 키보드 값을 전송하는 부분이 있고 이후에 한번더 빈 데이터를 전송하는 걸 볼 수 있습니다.

```

1234 123456789012345678901234567890123456789012345678901234567890123456789012345678901234
5    567890
1      uint8_t msg1[] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
2      input->setValue(msg1, sizeof(msg1));
3      input->notify();
4

```

불필요한 작업이 아닌가 싶어 삭제해 보았더니 앞서 전송한 키를 계속 누르고 있는 것으로 인식이 되었습니다.

키보드를 누르는 데이터를 보내고 빈데이터를 하나 보내서 키를 Release시키는 것으로 보입니다.

다음으로 궁금한 부분은 전송할 데이터에 키의 값은 어떻게 되어 있는지 입니다.

과정을 보면 전송할 키의 문자를 keymap의 배열에서 다른 값으로 변환을 하고 있습니다.

이 keymap의 배열이 어떻게 정의되어 있을지 궁금해서 검색해 보니 아래와 같이 변환이 되고 있습니다.

1234	123456789012345678901234567890123456789012345678901234567890123456789012345678901234
5	567890
1	const KEYMAP keymap[KEYMAP_SIZE] = {
2	{0, 0}, /* NUL */
3	{0, 0}, /* SOH */
4	{0, 0}, /* STX */
5	{0, 0}, /* ETX */
6	{0, 0}, /* EOT */
7	{0, 0}, /* ENQ */
8	{0, 0}, /* ACK */
9	{0, 0}, /* BEL */
10	{0x2a, 0}, /* BS */ /* Keyboard Delete (Backspace) */
11	{0x2b, 0}, /* TAB */ /* Keyboard Tab */
12	{0x28, 0}, /* LF */ /* Keyboard Return (Enter) */
13	{0, 0}, /* VT */
14	{0, 0}, /* FF */
15	{0, 0}, /* CR */
16	{0, 0}, /* SO */
17	{0, 0}, /* SI */
18	{0, 0}, /* DEL */
19	{0, 0}, /* DC1 */
20	{0, 0}, /* DC2 */
21	{0, 0}, /* DC3 */
22	{0, 0}, /* DC4 */
23	{0, 0}, /* NAK */
24	{0, 0}, /* SYN */
25	{0, 0}, /* ETB */
26	{0, 0}, /* CAN */
27	{0, 0}, /* EM */
28	{0, 0}, /* SUB */
29	{0, 0}, /* ESC */
30	{0, 0}, /* FS */
31	{0, 0}, /* GS */
32	{0, 0}, /* RS */
33	{0, 0}, /* US */
34	{0x2c, 0}, /* */
35	{0x1e, KEY_SHIFT}, /* ! */
36	{0x34, KEY_SHIFT}, /* " */
37	{0x20, KEY_SHIFT}, /* # */
38	{0x21, KEY_SHIFT}, /* \$ */
39	{0x22, KEY_SHIFT}, /* % */
40	{0x24, KEY_SHIFT}, /* & */
41	{0x34, 0}, /* ' */
42	{0x26, KEY_SHIFT}, /* ( */
43	{0x27, KEY_SHIFT}, /* ) */
44	{0x25, KEY_SHIFT}, /* * */



45	{0x2e, KEY_SHIFT}, /* + */
46	{0x36, 0}, /* , */
47	{0x2d, 0}, /* - */
48	{0x37, 0}, /* . */
49	{0x38, 0}, /* / */
50	{0x27, 0}, /* 0 */
51	{0x1e, 0}, /* 1 */
52	{0x1f, 0}, /* 2 */
53	{0x20, 0}, /* 3 */
54	{0x21, 0}, /* 4 */
55	{0x22, 0}, /* 5 */
56	{0x23, 0}, /* 6 */
57	{0x24, 0}, /* 7 */
58	{0x25, 0}, /* 8 */
59	{0x26, 0}, /* 9 */
60	{0x33, KEY_SHIFT}, /* : */
61	{0x33, 0}, /* ; */
62	{0x36, KEY_SHIFT}, /* < */
63	{0x2e, 0}, /* = */
64	{0x37, KEY_SHIFT}, /* > */
65	{0x38, KEY_SHIFT}, /* ? */
66	{0x1f, KEY_SHIFT}, /* @ */
67	{0x04, KEY_SHIFT}, /* A */
68	{0x05, KEY_SHIFT}, /* B */
69	{0x06, KEY_SHIFT}, /* C */
70	{0x07, KEY_SHIFT}, /* D */
71	{0x08, KEY_SHIFT}, /* E */
72	{0x09, KEY_SHIFT}, /* F */
73	{0x0a, KEY_SHIFT}, /* G */
74	{0x0b, KEY_SHIFT}, /* H */
75	{0x0c, KEY_SHIFT}, /* I */
76	{0x0d, KEY_SHIFT}, /* J */
77	{0x0e, KEY_SHIFT}, /* K */
78	{0x0f, KEY_SHIFT}, /* L */
79	{0x10, KEY_SHIFT}, /* M */
80	{0x11, KEY_SHIFT}, /* N */
81	{0x12, KEY_SHIFT}, /* O */
82	{0x13, KEY_SHIFT}, /* P */
83	{0x14, KEY_SHIFT}, /* Q */
84	{0x15, KEY_SHIFT}, /* R */
85	{0x16, KEY_SHIFT}, /* S */
86	{0x17, KEY_SHIFT}, /* T */
87	{0x18, KEY_SHIFT}, /* U */
88	{0x19, KEY_SHIFT}, /* V */
89	{0x1a, KEY_SHIFT}, /* W */
90	{0x1b, KEY_SHIFT}, /* X */
91	{0x1c, KEY_SHIFT}, /* Y */
92	{0x1d, KEY_SHIFT}, /* Z */
93	{0x2f, 0}, /* [ */
94	{0x31, 0}, /* \ */
95	{0x30, 0}, /* ] */
96	{0x23, KEY_SHIFT}, /* ^ */
97	{0x2d, KEY_SHIFT}, /* _ */
98	{0x35, 0}, /* ` */
99	{0x04, 0}, /* a */
100	{0x05, 0}, /* b */
101	{0x06, 0}, /* c */
102	{0x07, 0}, /* d */
103	{0x08, 0}, /* e */
104	{0x09, 0}, /* f */
105	{0x0a, 0}, /* g */
106	{0x0b, 0}, /* h */



```

107 {0x0c, 0}, /* i */
108 {0x0d, 0}, /* j */
109 {0x0e, 0}, /* k */
110 {0x0f, 0}, /* l */
111 {0x10, 0}, /* m */
112 {0x11, 0}, /* n */
113 {0x12, 0}, /* o */
114 {0x13, 0}, /* p */
115 {0x14, 0}, /* q */
116 {0x15, 0}, /* r */
117 {0x16, 0}, /* s */
118 {0x17, 0}, /* t */
119 {0x18, 0}, /* u */
120 {0x19, 0}, /* v */
121 {0x1a, 0}, /* w */
122 {0x1b, 0}, /* x */
123 {0x1c, 0}, /* y */
124 {0x1d, 0}, /* z */
125 {0x2f, KEY_SHIFT}, /* { */
126 {0x31, KEY_SHIFT}, /* | */
127 {0x30, KEY_SHIFT}, /* } */
128 {0x35, KEY_SHIFT}, /* ~ */
129 {0, 0}, /* DEL */
130
131 {0x3a, 0}, /* F1 */
132 {0x3b, 0}, /* F2 */
133 {0x3c, 0}, /* F3 */
134 {0x3d, 0}, /* F4 */
135 {0x3e, 0}, /* F5 */
136 {0x3f, 0}, /* F6 */
137 {0x40, 0}, /* F7 */
138 {0x41, 0}, /* F8 */
139 {0x42, 0}, /* F9 */
140 {0x43, 0}, /* F10 */
141 {0x44, 0}, /* F11 */
142 {0x45, 0}, /* F12 */
143
144 {0x46, 0}, /* PRINT_SCREEN */
145 {0x47, 0}, /* SCROLL_LOCK */
146 {0x39, 0}, /* CAPS_LOCK */
147 {0x53, 0}, /* NUM_LOCK */
148 {0x49, 0}, /* INSERT */
149 {0x4a, 0}, /* HOME */
150 {0x4b, 0}, /* PAGE_UP */
151 {0x4e, 0}, /* PAGE_DOWN */
152
153 {0x4f, 0}, /* RIGHT_ARROW */
154 {0x50, 0}, /* LEFT_ARROW */
155 {0x51, 0}, /* DOWN_ARROW */
156 {0x52, 0}, /* UP_ARROW */
157 };
158

```

결국 각 키보드의 키에 대한 정의인것 같습니다.

다시 인터넷에 hid usage table를 검색하여 아래와 같은 자료를 찾았습니다.

윗 테이블은 2개의 값으로 구성이 되어 있고 앞부분의 값은 키에 대한 값인 듯 하며 뒷부분의 값은 특수키와 관련된 값으로 보입니다.

즉 키보드의 A키가 누르는 신호를 보내려면 A키의 ASCII 코드는 A는 65 , a는 97입니다.

A키가 눌려진 경우의 데이터를 전송하려면 67번째 줄의

```
67      {0x04, KEY_SHIFT},      /* A */
```

a키가 눌려진 경우의 데이터를 전송하려면 99번째 줄의

```
99      {0x04, 0},      /* a */
```

로 보이며 앞부분의 데이터는 키의 값이고 뒷부분의 데이터는 특수키에 해당하는 것으로 보입니다.

이 특수키에 해당하는 부분을 modifiers라고 부르며 아래와 같이 정의가 되어 있습니다.

```
1234      12345678901234567890123456789012345678901234567890123456789012345678901234
5      567890

1      /* Modifiers */
2      enum MODIFIER_KEY {
3          KEY_CTRL = 1,
4          KEY_SHIFT = 2,
5          KEY_ALT = 4,
6      };
7
8
9      enum MEDIA_KEY {
10         KEY_NEXT_TRACK,      /*!< next Track Button */
11         KEY_PREVIOUS_TRACK, /*!< Previous track Button */
12         KEY_STOP,            /*!< Stop Button */
13         KEY_PLAY_PAUSE,      /*!< Play/Pause Button */
14         KEY_MUTE,            /*!< Mute Button */
15         KEY_VOLUME_UP,       /*!< Volume Up Button */
16         KEY_VOLUME_DOWN,     /*!< Volume Down Button */
17     };
18
19     enum FUNCTION_KEY {
20         KEY_F1 = 128, /* F1 key */
21         KEY_F2,      /* F2 key */
22         KEY_F3,      /* F3 key */
23         KEY_F4,      /* F4 key */
24         KEY_F5,      /* F5 key */
25         KEY_F6,      /* F6 key */
26         KEY_F7,      /* F7 key */
27         KEY_F8,      /* F8 key */
28         KEY_F9,      /* F9 key */
29         KEY_F10,     /* F10 key */
30         KEY_F11,     /* F11 key */
31         KEY_F12,     /* F12 key */
32
33         KEY_PRINT_SCREEN, /* Print Screen key */
34         KEY_SCROLL_LOCK, /* Scroll lock */
35         KEY_CAPS_LOCK,   /* caps lock */
36         KEY_NUM_LOCK,    /* num lock */
37         KEY_INSERT,      /* Insert key */
38         KEY_HOME,        /* Home key */
39         KEY_PAGE_UP,     /* Page Up key */
40         KEY_PAGE_DOWN,   /* Page Down key */
41
42         RIGHT_ARROW,     /* Right arrow */
43         LEFT_ARROW,      /* Left arrow */
44         DOWN_ARROW,      /* Down arrow */
45         UP_ARROW,        /* Up arrow */
46     };
```



소스코드를 할수 있는 부분까지 분석한 내용은 위 내용과 같으며 간단하게 테스트 코드를 만들어 보도록 하겠습니다.

우선은 특수키는 제외하고 시리얼 모니터를 이용하여 문자열을 입력하면 그 값을 키보드 데이터로 전송하도록 하겠습니다.

코드에서 특이 사항은 xTaskCreate를 이용하여 멀티 태스킹을 구현하고 있는 것으로 보입니다.

하지만 실제 실행된 task인 taskServer의 경우 마지막에

```
117 delay(portMAX_DELAY);
```

로 인하여 진행이 멈춰버립니다.

이유는 portMAX\_DELAY값을 출력해 본 결과 4294967295로 확인이 되었으며 이는 약 49일에 해당하는 시간이었습니다.

이와 같이 작성된 이유를 알수는 없지만 의미가 없는 것 같아 task로 초기화를 하지 않고 일반 함수로 변경을 하여 초기화시 구동하도록 하고 delay(portMAX\_DELAY)를 제거하고 구동하여 보면 정상 작동하는 것을 확인 할 수 있었습니다.

또한 키보드 값을 연속입력할 경우 순서대로 입력이 안되고 앞뒤가 바뀌어 입력되는 현상이 가끔 발생하였습니다.

정확한 원인 연시 모르겠지만 블루투스를 이용한 신호 전송중 다른 신호의 전송명령으로 인하여 혼선이 오는 것같아 이전 신호가 모두 전달될 시간을 주기 위하여 delay명령을 notify 명령 이후에 넣어 주었습니다.

이와 같이 수정된 코드는 다음과 같으며

코드 업로딩 후 시리얼 모니터를 이용하여 글자를 입력하면 ESP32는 이를 블루투스 키보드로 전송이 되도록 하였습니다.

```
123 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
45 67890

1
2 #include <BLEDevice.h>
3 #include <BLEUtils.h>
4 #include <BLEServer.h>
5 #include "BLE2902.h"
6 #include "BLEHIDDevice.h"
7 #include "HIDTypes.h"
8 #include "HIDKeyboardTypes.h"
9 #include <driver/adc.h>
```

```

10
11 BLEHIDDevice* hid;
12 BLECharacteristic* input;
13 BLECharacteristic* output;
14
15 bool connected = false;
16
17 class MyCallbacks : public BLEServerCallbacks
18 {
19     void onConnect(BLEServer* pServer)
20     {
21         connected = true;
22         BLE2902* desc = (BLE2902*)input->getDescriptorByUUID(BLEUUID((uint16_t)0x2902));
23         desc->setNotifications(true);
24     }
25
26     void onDisconnect(BLEServer* pServer)
27     {
28         connected = false;
29         BLE2902* desc = (BLE2902*)input->getDescriptorByUUID(BLEUUID((uint16_t)0x2902));
30         desc->setNotifications(false);
31     }
32 };
33
34 /*
35  This callback is connect with output report. In keyboard output report report special keys
  changes, like CAPSLOCK, NUMLOCK
36  We can add digital pins with LED to show status
37  bit 0 - NUM LOCK
38  bit 1 - CAPS LOCK
39  bit 2 - SCROLL LOCK
40  */
41 class MyOutputCallbacks : public BLECharacteristicCallbacks
42 {
43     void onWrite(BLECharacteristic* me)
44     {
45         uint8_t* value = (uint8_t*)(me->getValue().c_str());
46         ESP_LOGI(LOG_TAG, "special keys: %d", *value);
47     }
48 };
49
50 void _taskServer()
51 {
52     BLEDevice::init("ESP32-keyboard");
53     BLEServer* pServer = BLEDevice::createServer();
54     pServer->setCallbacks(new MyCallbacks());
55
56     hid = new BLEHIDDevice(pServer);
57     input = hid->inputReport(1); // <-- input REPORTID from report map
58     output = hid->outputReport(1); // <-- output REPORTID from report map
59
60     output->setCallbacks(new MyOutputCallbacks());
61
62     std::string name = "chegewara";
63     hid->manufacturer()->setValue(name);
64
65     hid->pnp(0x02, 0xe502, 0xa111, 0x0210);
66     hid->hidInfo(0x00, 0x02);
67
68     BLESecurity* pSecurity = new BLESecurity();
69     // pSecurity->setKeySize();
70     pSecurity->setAuthenticationMode(ESP_LE_AUTH_BOND);

```



```

71
72     const uint8_t report[] =
73     {
74         USAGE_PAGE(1),      0x01,      // Generic Desktop Ctrl's
75         USAGE(1),           0x06,      // Keyboard
76         COLLECTION(1),      0x01,      // Application
77         REPORT_ID(1),        0x01,      // Report ID (1)
78         USAGE_PAGE(1),       0x07,      // Kbrd/Keypad
79         USAGE_MINIMUM(1),    0xE0,
80         USAGE_MAXIMUM(1),    0xE7,
81         LOGICAL_MINIMUM(1),  0x00,
82         LOGICAL_MAXIMUM(1),  0x01,
83         REPORT_SIZE(1),      0x01,      // 1 byte (Modifier)
84         REPORT_COUNT(1),     0x08,
85         HIDINPUT(1),         0x02,      // Data, Var, Abs, No Wrap, Linear, Preferred State, No
Null Position
86         REPORT_COUNT(1),     0x01,      // 1 byte (Reserved)
87         REPORT_SIZE(1),     0x08,
88         HIDINPUT(1),         0x01,      // Const, Array, Abs, No Wrap, Linear, Preferred State,
No Null Position
89         REPORT_COUNT(1),     0x06,      // 6 bytes (Keys)
90         REPORT_SIZE(1),     0x08,
91         LOGICAL_MINIMUM(1),  0x00,
92         LOGICAL_MAXIMUM(1),  0x65,      // 101 keys
93         USAGE_MINIMUM(1),    0x00,
94         USAGE_MAXIMUM(1),    0x65,
95         HIDINPUT(1),         0x00,      // Data, Array, Abs, No Wrap, Linear, Preferred State, N
o Null Position
96         REPORT_COUNT(1),     0x05,      // 5 bits (Num Lock, Caps Lock, Scroll Lock, Compose,
Kana)
97         REPORT_SIZE(1),     0x01,
98         USAGE_PAGE(1),       0x08,      // LEDs
99         USAGE_MINIMUM(1),    0x01,      // Num Lock
100        USAGE_MAXIMUM(1),     0x05,      // Kana
101        HIDOUTPUT(1),          0x02,      // Data, Var, Abs, No Wrap, Linear, Preferred State, No
Null Position, Non-volatile
102        REPORT_COUNT(1),      0x01,      // 3 bits (Padding)
103        REPORT_SIZE(1),       0x03,
104        HIDOUTPUT(1),         0x01,      // Const, Array, Abs, No Wrap, Linear, Preferred State,
No Null Position, Non-volatile
105        END_COLLECTION(0)
106    };
107
108    hid->reportMap((uint8_t*)report, sizeof(report));
109    hid->startServices();
110
111    BLEAdvertising *pAdvertising = pServer->getAdvertising();
112    pAdvertising->setAppearance(HID_KEYBOARD);
113    pAdvertising->addServiceUUID(hid->hidService()->getUUID());
114    pAdvertising->start();
115    hid->setBatteryLevel(7);
116    ESP_LOGD(LOG_TAG, "Advertising started!");
117    // delay(portMAX_DELAY);
118    };
119
120    void setup()
121    {
122        Serial.begin(115200);
123        Serial.println("Starting BLE work!");
124        _taskServer();
125        // xTaskCreate(taskServer, "server", 20000, NULL, 5, NULL);
126    }
127

```

```
128 char message[2];
129
130 void loop()
131 {
132     if ( Serial.available() )
133     {
134         char ch = Serial.read();
135         if ( connected == true )
136         {
137             KEYMAP map = keymap[(uint8_t) ch];
138             uint8_t msg[] = {map.modifier, 0x0, map.usage, 0x0, 0x0, 0x0, 0x0, 0x0};
139             input->setValue(msg, sizeof(msg));
140             input->notify();
141             delay(15);
142             uint8_t msg1[] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
143             input->setValue(msg1, sizeof(msg1));
144             input->notify();
145             delay(15);
146         }
147     }
148 }
149
```

이와 같이 정상적인 사용법인지 확인은 못했지만 구동가능한 무선 키보드에 대한 작성을 하였으며 정상작동하는 것을 확인하였습니다.

관련 자료 : esp32\_3

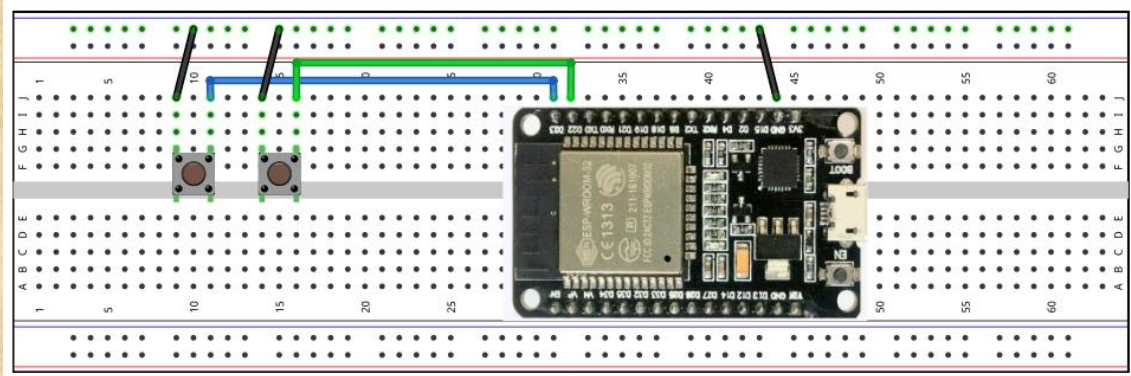
구동영상 :

<https://serviceapi.nmv.naver.com/flash/convertIframeTag.nhn?vid=390405821947D823E8DE3EEDE749FF3FC015&outKey=V12240d9d0f2c7e446cb85caa732847ba922e363058f88dadcca15caa732847ba922e&width=740&height=416>



## ■ ESP32 - 버튼을 이용한 키보드 제어

ESP32를 BLE HID 키보드로 인식시키고 두 개의 버튼을 설치하여 좌측과 우측 방향키로 사용하는 예입니다.



```

1234      123456789012345678901234567890123456789012345678901234567890123456789012345678901234
5      567890

1      void loop()
2      {
3
4      int left_status = digitalRead(23);
5      int right_status = digitalRead(22);
6
7      if ( ( left_status == LOW ) && ( prev_left_status == HIGH ) )
8      {
9          if ( connected == true )
10         {
11             KEYMAP map = {0x50, 0}; //keymap[(uint8_t) ch];
12             uint8_t msg[] = {map.modifier, 0x0, map.usage, 0x0, 0x0, 0x0, 0x0, 0x0};
13             input->setValue(msg, sizeof(msg));
14             input->notify();
15             delay(15);
16         }
17     }
18     if ( ( left_status == HIGH ) && ( prev_left_status == LOW ) )
19     {
20         if ( connected == true )
21         {
22             uint8_t msg1[] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
23             input->setValue(msg1, sizeof(msg1));
24             input->notify();
25             delay(15);
26         }
27     }
28
29     if ( ( right_status == LOW ) && ( prev_right_status == HIGH ) )
30     {
31         if ( connected == true )
32         {
33             KEYMAP map = {0x4F, 0}; //keymap[(uint8_t) ch];
34             uint8_t msg[] = {map.modifier, 0x0, map.usage, 0x0, 0x0, 0x0, 0x0, 0x0};
35             input->setValue(msg, sizeof(msg));

```

```

36         input->notify();
37         delay(15);
38     }
39 }
40
41 if ( ( right_status == HIGH ) && ( prev_right_status == LOW ) )
42 {
43     if ( connected == true )
44     {
45         uint8_t msg1[] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
46         input->setValue(msg1, sizeof(msg1));
47         input->notify();
48         delay(15);
49     }
50 }
51
52 prev_left_status = left_status;
53 prev_right_status = right_status;
54
55 if ( Serial.available() )
56 {
57     char ch = Serial.read();
58     if ( connected == true )
59     {
60         KEYMAP map = keymap[(uint8_t) ch];
61         uint8_t msg[] = {map.modifier, 0x0, map.usage, 0x0, 0x0, 0x0, 0x0, 0x0};
62         input->setValue(msg, sizeof(msg));
63         input->notify();
64         delay(15);
65         uint8_t msg1[] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
66         input->setValue(msg1, sizeof(msg1));
67         input->notify();
68         delay(15);
69     }
70 }
71 }
72

```

구동화면 :

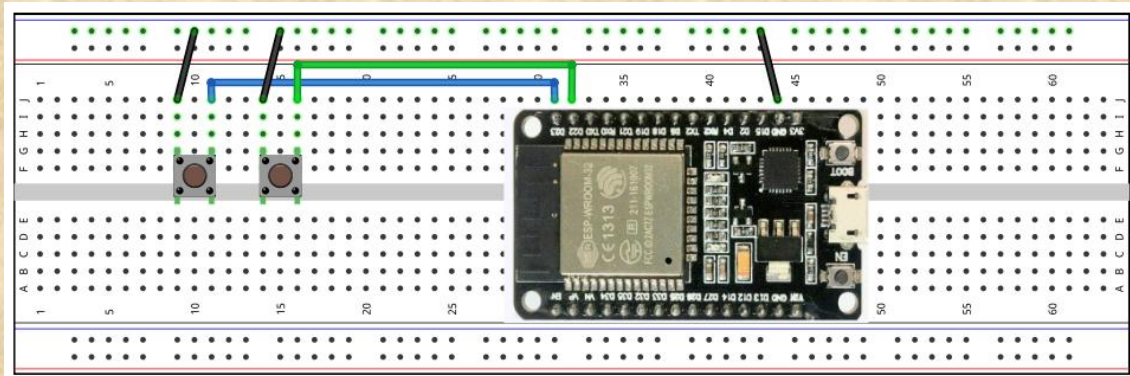
<https://servicapi.nmv.naver.com/flash/convertIframeTag.nhn?vid=216458B83D3936ADDAEC225D4408EE69B810&outKey=V125a9da02bff208a672881e760f310062bc7837eafef9549389681e760f310062bc7&width=740&height=416>

관련자료 : esp32\_5



## ■ ESP32 - 조이스틱으로 인식 시키기

ESP32를 BLE HID KEYPAD(JOYSTICK)로 인식시키고 두 개의 버튼을 설치하여 좌측과 우측 방향으로 사용하는 예입니다.



```

1234 123456789012345678901234567890123456789012345678901234567890123456789012345678901234
5    567890

1    #include <BLEDevice.h>
2    #include <BLEServer.h>
3    #include <BLEUtils.h>
4    #include <BLEHIDDevice.h>
5    #include <BLE2902.h>
6
7    static BLEHIDDevice *pHID;
8    BLEServer *pServer;
9    BLECharacteristic *input;
10
11    bool deviceConnected = false;
12    bool oldDeviceConnected = false;
13
14    struct gamepad_report_t
15    {
16        int8_t left_x;
17        int8_t left_y;
18        uint16_t buttons;
19    };
20
21    bool operator!=(const gamepad_report_t& lhs, const gamepad_report_t& rhs)
22    {
23        return (lhs.left_x != rhs.left_x)
24            || (lhs.left_y != rhs.left_y)
25            || (lhs.buttons != rhs.buttons);
26    }
27
28    class MyServerCallbacks: public BLEServerCallbacks
29    {
30    public:
31        void onConnect(BLEServer* pServer)
32        {
33            deviceConnected = true;
34            Serial.println("CONNECTED");
35        }

```

```

36         void onDisconnect(BLEServer* pServer)
37         {
38             deviceConnected = false;
39             Serial.println("DISCONNECTED");
40         }
41     };
42
43     void setup()
44     {
45         Serial.begin(115200);
46         pinMode(22, INPUT_PULLUP);
47         pinMode(23, INPUT_PULLUP);
48
49
50         // Create the BLE Device
51         BLEDevice::init("kProject Joystic"); // Give it a name
52
53         // Create the BLE Server
54         pServer = BLEDevice::createServer();
55         pServer->setCallbacks(new MyServerCallbacks());
56
57         // Instantiate HID Device
58         pHID = new BLEHIDDevice(pServer);
59         input = pHID->inputReport(1);
60
61         pHID->manufacturer()->setValue("kProject");
62         pHID->pnp(0x01, 0x02e5, 0xabcd, 0x0110);
63         pHID->hidInfo(0x00, 0x01);
64
65         BLESecurity *pSecurity = new BLESecurity();
66         pSecurity->setAuthenticationMode(ESP_LE_AUTH_BOND);
67
68         // Set Report Map
69         const uint8_t reportMap[] =
70         {
71             0x05, 0x01, // USAGE_PAGE (Generic Desktop)
72             0x09, 0x05, // USAGE (Game Pad)
73             0xa1, 0x01, // COLLECTION (Application)
74             0xa1, 0x02, // COLLECTION (Logical)
75             0x85, 0x01, // REPORT_ID (1)
76
77             0x75, 0x08, // REPORT_SIZE (8)
78             0x95, 0x02, // REPORT_COUNT (2)
79             0x05, 0x01, // USAGE_PAGE (Generic Desktop)
80             0x09, 0x30, // USAGE (X)
81             0x09, 0x31, // USAGE (Y)
82             0x15, 0x81, // LOGICAL_MINIMUM (-127)
83             0x25, 0x7f, // LOGICAL_MAXIMUM (127)
84             0x81, 0x02, // INPUT (Data, Var, Abs)
85
86             0x75, 0x01, // REPORT_SIZE (1)
87             0x95, 0x0b, // REPORT_COUNT (11)
88             0x15, 0x00, // LOGICAL_MINIMUM (0)
89             0x25, 0x01, // LOGICAL_MAXIMUM (1)
90             0x05, 0x09, // USAGE_PAGE (Button)
91             0x19, 0x01, // USAGE_MINIMUM (Button 1)
92             0x29, 0x0b, // USAGE_MAXIMUM (Button 11)
93             0x81, 0x02, // INPUT (Data, Var, Abs)
94             // PADDING for byte alignment
95             0x75, 0x01, // REPORT_SIZE (1)
96             0x95, 0x05, // REPORT_COUNT (5)
97             0x81, 0x03, // INPUT (Constant, Var, Abs)

```



```

98
99         0xc0,                // END_COLLECTION
100        0xc0                // END_COLLECTION
101    };
102
103    pHID->reportMap((uint8_t*)reportMap, sizeof(reportMap));
104    int numReport = sizeof(reportMap);
105    Serial.println(numReport);
106
107    // SetupGamepad();
108
109    // Start the service
110    pHID->startServices();
111
112    // Start advertising
113    BLEAdvertising *pAdvertising = pServer->getAdvertising();
114    pAdvertising->setAppearance(HID_GAMEPAD);
115    pAdvertising->addServiceUUID(pHID->hidService()->getUUID());
116    pAdvertising->start();
117
118    Serial.println("Waiting a client connection to notify...");
119    }
120
121    gamepad_report_t oldValue, newValue;
122
123    void loop()
124    {
125        if (deviceConnected)
126        {
127            // AXIS
128            {
129                newValue.left_y = 0;
130                newValue.left_x = 0;
131                if (digitalRead(23) == LOW)
132                {
133                    newValue.left_x = -100;
134                }
135                if (digitalRead(22) == LOW)
136                {
137                    newValue.left_x = 100;
138                }
139            }
140
141            // BUTTONS
142            {
143                newValue.buttons = 0;
144            }
145
146            if (newValue != oldValue)
147            {
148                uint8_t a[] = {newValue.left_x, newValue.left_y, newValue.buttons, (newValue.buttons >> 8)};
149                input->setValue(a, sizeof(a));
150                input->notify();
151                oldValue = newValue;
152            }
153            delay(5);
154        }
155
156        // Connecting
157        if (deviceConnected && !oldDeviceConnected)
158        {

```

```
159         oldDeviceConnected = deviceConnected;
160     }
161
162     // Disconnecting
163     if (!deviceConnected && oldDeviceConnected)
164     {
165         delay(500); // give the bluetooth stack the chance to get things ready
166         pServer->startAdvertising();
167         Serial.println("restart advertising");
168         oldDeviceConnected = deviceConnected;
169     }
170 }
171
172
173
```

관련자료 : esp32\_6



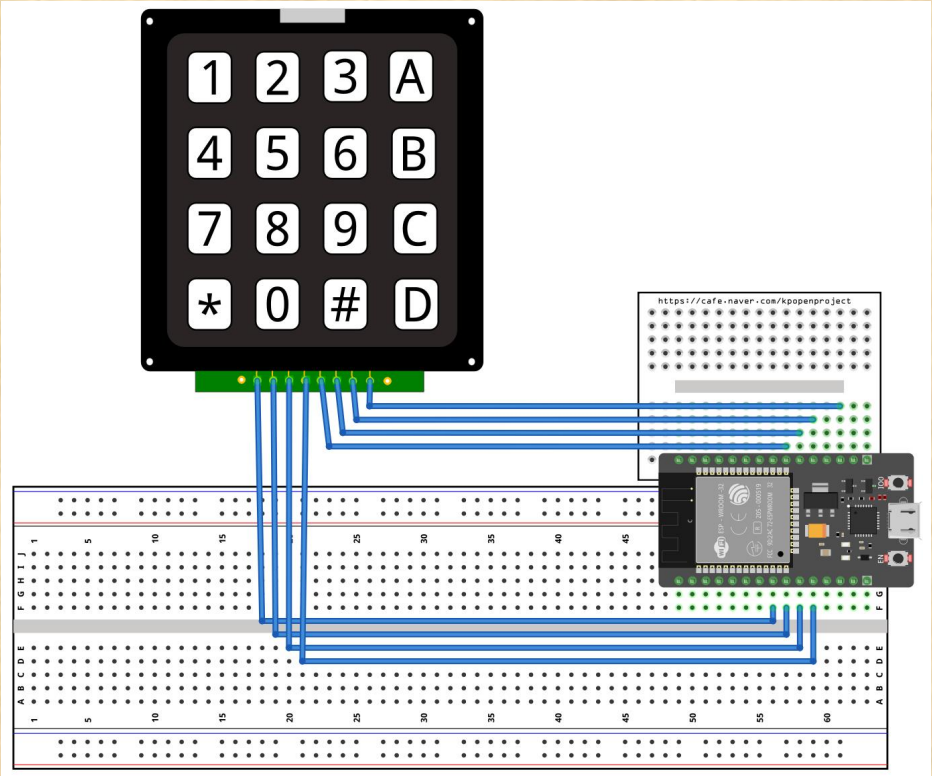
■ ESP32를 이용한 MATRIX KEYPAD 구동

이번에는 키패드 동작 테스트 입니다.  
KEYPAD는 PCB에 매트릭스 형식으로 구성을 했습니다.  
KEYPAD의 각 행, 열에 연결된 핀은 다음과 같습니다.

ESP32	부속(모듈)	모듈 핀
GPI 017	KEYPAD	C1
GPI 016	KEYPAD	C2
GPI 04	KEYPAD	C3
GPI 015	KEYPAD	C4
GPI 025	KEYPAD	R1
GPI 026	KEYPAD	R2
GPI 027	KEYPAD	R3
GPI 014	KEYPAD	R4

KEYPAD의 동작과 관련된 원리는 아래 링크의 글을 참조하시기 바랍니다.  
<https://cafe.naver.com/kpopenproject/333>

회로도



소스코드는 다음과 같습니다.

```

123 12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345
45 67890

1  #define ROWS 4
2  #define COLS 4
3
4  byte row_pins[ROWS] = {25, 26, 27, 14};
5  byte col_pins[COLS] = {17, 16, 4, 15};
6
7  uint16_t buttons;
8
9  void setup()
10 {
11     Serial.begin(9600);
12     buttons = 0;
13 }
14
15 void loop()
16 {
17     check_key2();
18     delay(100);
19 }
20
21 void check_key2()
22 {
23     buttons = 0;
24     for ( int i = 0; i < 4; i++)
25     {
26         // 1. ROW의 모든 핀을 모두 INPUT 상태로 변경합니다
27         for ( int i = 0; i < 4; i++)
28         {
29             pinMode(row_pins[i], INPUT_PULLUP);
30         }
31         // 2. 이제 COLUMN의 모든 핀을 GND로 설정을 합니다.
32         for ( int i = 0; i < 4; i++)
33         {
34             pinMode(col_pins[i], OUTPUT);
35             digitalWrite(col_pins[i], LOW);
36         }
37         /*
38         3. 그리고 나서 원하는 ROW의 값을 읽습니다.
39         4. ROW에 해당하는 버튼 중 1개의 버튼이라도 눌러진 버튼이 있으면 ROW핀의 값은 LOW가 됩니다.
        모두 HIGH일 경우 버튼은 하나도 눌러지지 않은 상태입니다.
        */
41         if ( digitalRead(row_pins[i]) == LOW )
42         {
43             for ( int j = 0; j < 4; j++)
44             {
45                 // 1. COL의 핀을 모두 INPUT 상태로 변경합니다.
46                 for ( int k = 0; k < 4; k++)
47                 {
48                     pinMode(col_pins[k], INPUT_PULLUP);
49                 }
50
51                 // 2. 이제 ROW의 모든 핀을 GND로 설정을 합니다.
52                 pinMode(row_pins[i], OUTPUT);
53                 digitalWrite(row_pins[i], LOW);
54
55

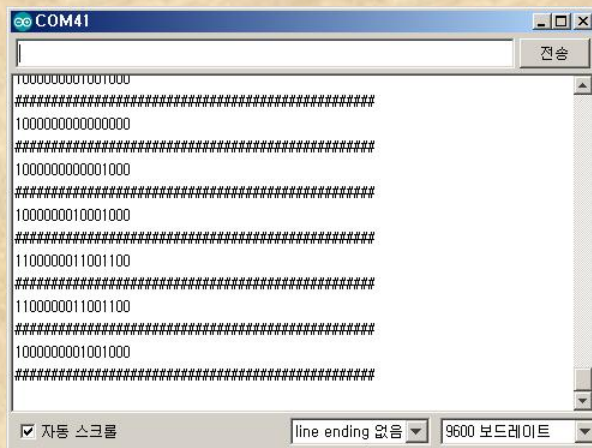
```



```

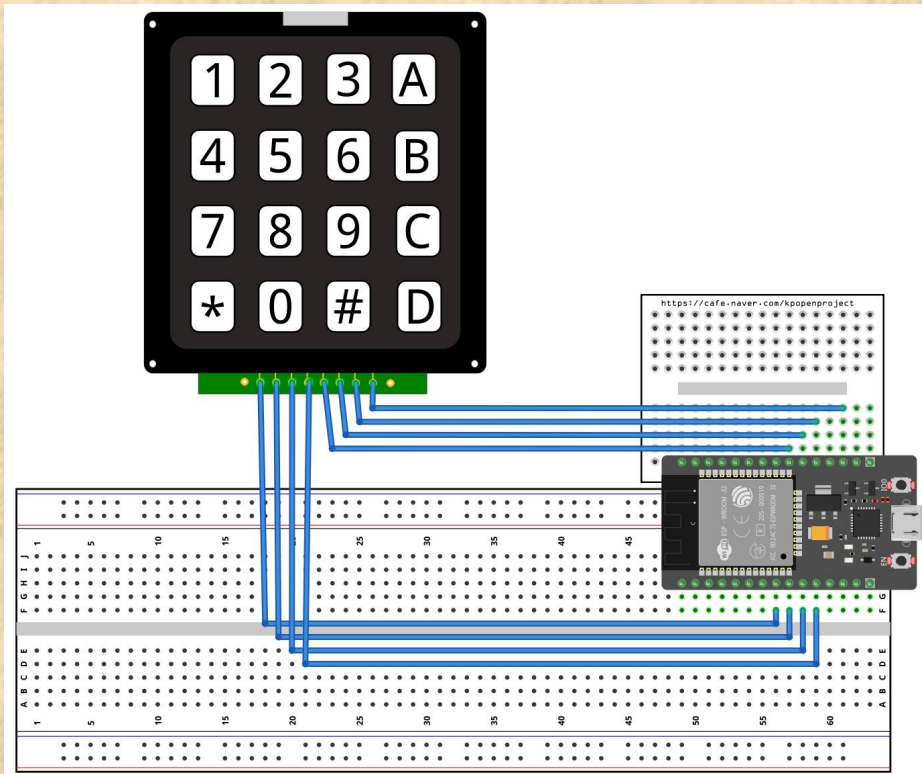
56         if ( digitalRead(col_pins[j]) == HIGH )
57         {
58             int index = i*4+j;
59             buttons = buttons | ( 0b00000001 << index );
60         }
61     }
62 }
63 else
64 {
65     int index = i * 4;
66     buttons = buttons | ( 0b00000001 << ( index + 0 ) );
67     buttons = buttons | ( 0b00000001 << ( index + 1 ) );
68     buttons = buttons | ( 0b00000001 << ( index + 2 ) );
69     buttons = buttons | ( 0b00000001 << ( index + 3 ) );
70 }
71 }
72 buttons = ~buttons;
73 Serial.println(buttons, BIN);
74 Serial.println("#####");
75 delay(100);
76 }
77
78

```



## ■ ESP32를 이용한 조이스틱

<회로도>



<소스코드>

```

123 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
45 67890

1  #define ROWS 4
2  #define COLS 4
3
4  byte row_pins[ROWS] = {25, 26, 27, 14};
5  byte col_pins[COLS] = {17, 16, 4, 15};
6
7  uint16_t buttons;
8
9  #include <BLEDevice.h>
10 #include <BLEServer.h>
11 #include <BLEUtils.h>
12 #include <BLEHIDDevice.h>
13 #include <BLE2902.h>
14
15 static BLEHIDDevice *pHID;
16 BLEServer *pServer;
17 BLECharacteristic *input;
18
19 bool deviceConnected = false;
20 bool oldDeviceConnected = false;
21

```



```

22 struct gamepad_report_t
23 {
24     int8_t left_x;
25     int8_t left_y;
26     uint16_t buttons;
27 };
28
29 gamepad_report_t oldValue, newValue;
30
31 bool operator!=(const gamepad_report_t& lhs, const gamepad_report_t& rhs)
32 {
33     return (lhs.left_x != rhs.left_x)
34           || (lhs.left_y != rhs.left_y)
35           || (lhs.buttons != rhs.buttons);
36 }
37
38 class MyServerCallbacks: public BLEServerCallbacks
39 {
40     void onConnect(BLEServer* pServer)
41     {
42         deviceConnected = true;
43         Serial.println("CONNECTED");
44     };
45
46     void onDisconnect(BLEServer* pServer)
47     {
48         deviceConnected = false;
49         Serial.println("DISCONNECTED");
50     }
51 };
52
53 void setup()
54 {
55     Serial.begin(115200);
56     buttons = 0;
57
58     BLEDevice::init("kProject Joystick"); // Give it a name
59
60     // Create the BLE Server
61     pServer = BLEDevice::createServer();
62     pServer->setCallbacks(new MyServerCallbacks());
63
64     // Instantiate HID Device
65     pHID = new BLEHIDDevice(pServer);
66     input = pHID->inputReport(1);
67
68     pHID->manufacturer()->setValue("kProject");
69     pHID->pnp(0x01, 0x02e5, 0xabcd, 0x0110);
70     pHID->hidInfo(0x00, 0x01);
71
72     BLESecurity *pSecurity = new BLESecurity();
73     pSecurity->setAuthenticationMode(ESP_LE_AUTH_BOND);
74
75     // Set Report Map
76     const uint8_t reportMap[] =
77     {
78         0x05, 0x01, // USAGE_PAGE (Generic Desktop)
79         0x09, 0x05, // USAGE (Game Pad)
80         0xa1, 0x01, // COLLECTION (Application)
81         0xa1, 0x02, // COLLECTION (Logical)
82         0x85, 0x01, // REPORT_ID (1)
83

```

```

84         0x75, 0x08,          // REPORT_SIZE (8)
85         0x95, 0x02,          // REPORT_COUNT (2)
86         0x05, 0x01,          // USAGE_PAGE (Generic Desktop)
87         0x09, 0x30,          // USAGE (X)
88         0x09, 0x31,          // USAGE (Y)
89         0x15, 0x81,          // LOGICAL_MINIMUM (-127)
90         0x25, 0x7f,          // LOGICAL_MAXIMUM (127)
91         0x81, 0x02,          // INPUT (Data, Var, Abs)
92
93         0x75, 0x01,          // REPORT_SIZE (1)
94         0x95, 0x11,          // REPORT_COUNT (11)
95         0x15, 0x00,          // LOGICAL_MINIMUM (0)
96         0x25, 0x01,          // LOGICAL_MAXIMUM (1)
97         0x05, 0x09,          // USAGE_PAGE (Button)
98         0x19, 0x01,          // USAGE_MINIMUM (Button 1)
99         0x29, 0x11,          // USAGE_MAXIMUM (Button 11)
100        0x81, 0x02,          // INPUT (Data, Var, Abs)
101        // PADDING for byte alignment
102        0x75, 0x01,          // REPORT_SIZE (1)
103        0x95, 0x05,          // REPORT_COUNT (5)
104        0x81, 0x03,          // INPUT (Constant, Var, Abs)
105
106        0xc0,                // END_COLLECTION
107        0xc0                // END_COLLECTION
108    };
109
110    pHID->reportMap((uint8_t*)reportMap, sizeof(reportMap));
111    int numReport = sizeof(reportMap);
112    Serial.println(numReport);
113
114    // SetupGamepad();
115
116    // Start the service
117    pHID->startServices();
118
119    // Start advertising
120    BLEAdvertising *pAdvertising = pServer->getAdvertising();
121    pAdvertising->setAppearance(HID_GAMEPAD);
122    pAdvertising->addServiceUUID(pHID->hidService->getUUID());
123    pAdvertising->start();
124
125    Serial.println("Waiting a client connection to notify...");
126    }
127
128    void loop()
129    {
130        check_key2();
131
132        if (deviceConnected)
133        {
134            // AXIS
135            {
136                newValue.left_y = 0;
137                newValue.left_x = 0;
138            }
139
140            if ( 0b1000000000000000 & buttons )
141            {
142                newValue.left_x = -100;
143            }
144            if ( 0b0100000000000000 & buttons )
145            {

```



```

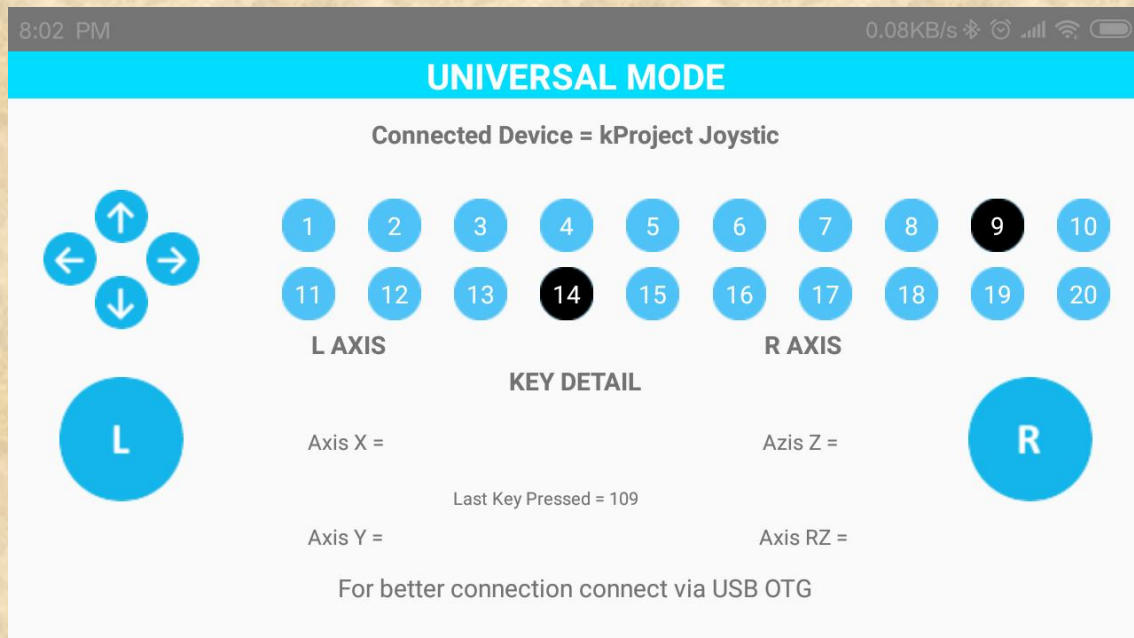
146         newValue.left_x = 100;
147     }
148
149     // BUTTONS
150     {
151         newValue.buttons = buttons;
152     }
153
154     if (newValue != oldValue)
155     {
156         uint8_t a[] = {newValue.left_x, newValue.left_y, newValue.buttons, (newValue.button
s >> 8), 0xFF};
157         input->setValue(a, sizeof(a));
158         input->notify();
159         oldValue = newValue;
160         Serial.println(buttons, DEC);
161     }
162     delay(5);
163 }
164
165 // Connecting
166 if (deviceConnected && !oldDeviceConnected)
167 {
168     oldDeviceConnected = deviceConnected;
169 }
170
171 // Disconnecting
172 if (!deviceConnected && oldDeviceConnected)
173 {
174     delay(500); // give the bluetooth stack the chance to get things ready
175     pServer->startAdvertising();
176     Serial.println("restart advertising");
177     oldDeviceConnected = deviceConnected;
178 }
179 }
180
181 void check_key2()
182 {
183     buttons = 0;
184     for (int i = 0; i < 4; i++)
185     {
186         // 1. ROW의 모든 핀을 모두 INPUT 상태로 변경합니다
187         for (int i = 0; i < 4; i++)
188         {
189             pinMode(row_pins[i], INPUT_PULLUP);
190         }
191         // 2. 이제 COLUMN의 모든 핀을 GND로 설정을 합니다.
192         for (int i = 0; i < 4; i++)
193         {
194             pinMode(col_pins[i], OUTPUT);
195             digitalWrite(col_pins[i], LOW);
196         }
197         /*
198         3. 그리고 나서 원하는 ROW의 값을 읽습니다.
199         4. ROW에 해당하는 버튼 중 1개의 버튼이라도 눌러진 버튼이 있으면 ROW 핀의 값은 LOW가 됩니다.
200         모두 HIGH일 경우 버튼은 하나도 눌러지지 않은 상태입니다.
201         */
202         if (digitalRead(row_pins[i]) == LOW)
203         {
204             for (int j = 0; j < 4; j++)
205             {
206                 // 1. COL의 핀을 모두 INPUT 상태로 변경합니다.

```

```

206         for ( int k = 0; k < 4; k++)
207         {
208             pinMode(col_pins[k] , INPUT_PULLUP);
209         }
210
211         // 2.   이제 ROW의 모든 핀을 GND로 설정을 합니다.
212         pinMode(row_pins[i], OUTPUT);
213         digitalWrite(row_pins[i], LOW);
214
215
216         if ( digitalRead(col_pins[j]) == HIGH )
217         {
218             int index = i*4+j;
219             buttons = buttons | ( 0b00000001 << index );
220         }
221     }
222 }
223 else
224 {
225     int index = i * 4;
226     buttons = buttons | ( 0b00000001 << ( index + 0 ) );
227     buttons = buttons | ( 0b00000001 << ( index + 1 ) );
228     buttons = buttons | ( 0b00000001 << ( index + 2 ) );
229     buttons = buttons | ( 0b00000001 << ( index + 3 ) );
230 }
231 }
232 buttons = ~buttons;
233 }
234
235

```






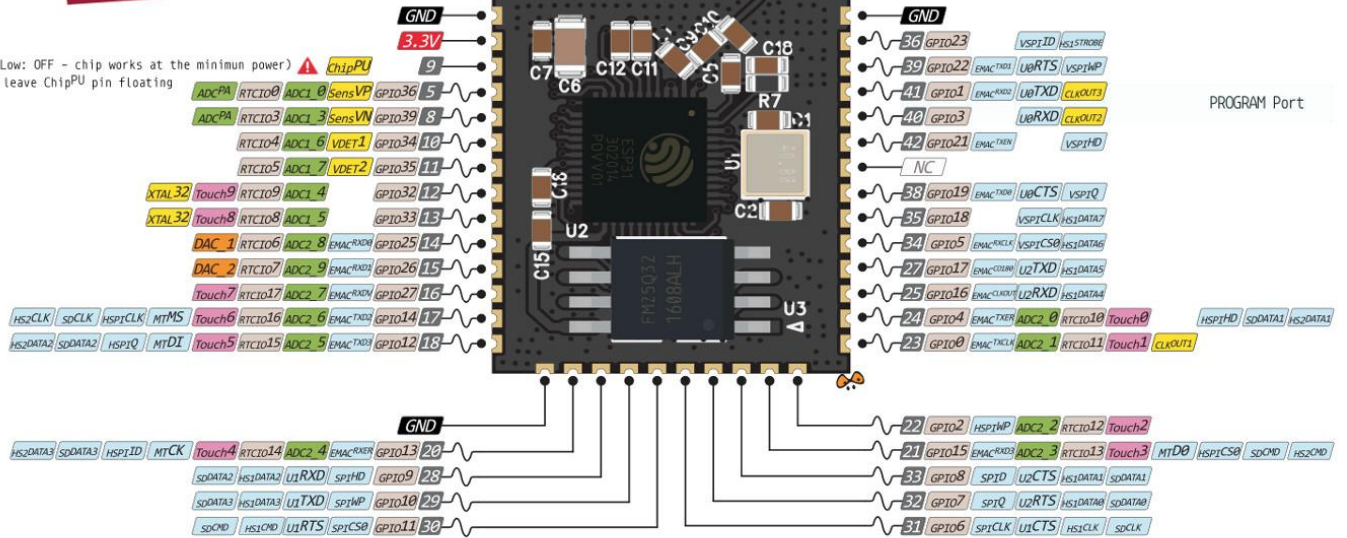
## ■ ESP82 MCU PIN MAP

# WROOM32

## PINOUT

Chip Enabled (High:ON, Low: OFF - chip works at the minimum power)  **ChipPU**  
Do not leave ChipPU pin floating

 Absolute MAX per pin 12mA  
recommended 6mA



-  Power
-  GND
-  Serial Pin
-  Analog Pin
-  Control
-  Physical Pin
-  Port Pin
-  Touch Pin
-  DAC Pin
-  PWM Pin



## ■ ESP82 MCU 사용 가능 핀 정리

ESP32 GPIO	아두이노	비고
GND		
3.3V		
CE	FLASH 핀	사용 주의
36	PULL-UP, PULL-DOWN 불가 (INPUT ONLY)	사용 주의(INPUT ONLY) ADC1_0
39	PULL-UP, PULL-DOWN 불가 (INPUT ONLY)	사용 주의(INPUT ONLY) ADC1_3
34	PULL-UP, PULL-DOWN 불가 (INPUT ONLY)	사용 주의(INPUT ONLY) ADC1_6
35	PULL-UP, PULL-DOWN 불가 (INPUT ONLY)	사용 주의(INPUT ONLY) ADC1_7



32		ADC1_4
33		ADC1_5
25		ADC2_8
26		ADC2_9
27		ADC2_7
14		ADC2_6
12	레귤레이터 전압 선택핀	사용 주의(부팅시 LOW) ADC2_5
GND		
13		
9	내부 플래쉬	사용 불가
10	내부 플래쉬	사용 불가
11	내부 플래쉬	사용 불가
6	내부 플래쉬	사용 불가
7	내부 플래쉬	사용 불가
8	내부 플래쉬	사용 불가
15		ADC2_3
2	부팅 모드 선택(UART download모드시, LOW)	사용 주의 부팅시 LOW ADC2_2
0	BOOT 핀	사용 주의 ADC2_1
4		ADC2_0
16		
17		
5		
18		
19		
NC		
21		
3	RX PIN	
1	TX PIN	
22		
23		
GND		



## ■ ESP32 개발보드 만들기

ESP32 MCU는 ESP8266보드와 마찬가지로 별도의 개별보드를 만들거나 개발보드에 장착된 모듈을 사용하여야 사용이 편리합니다.

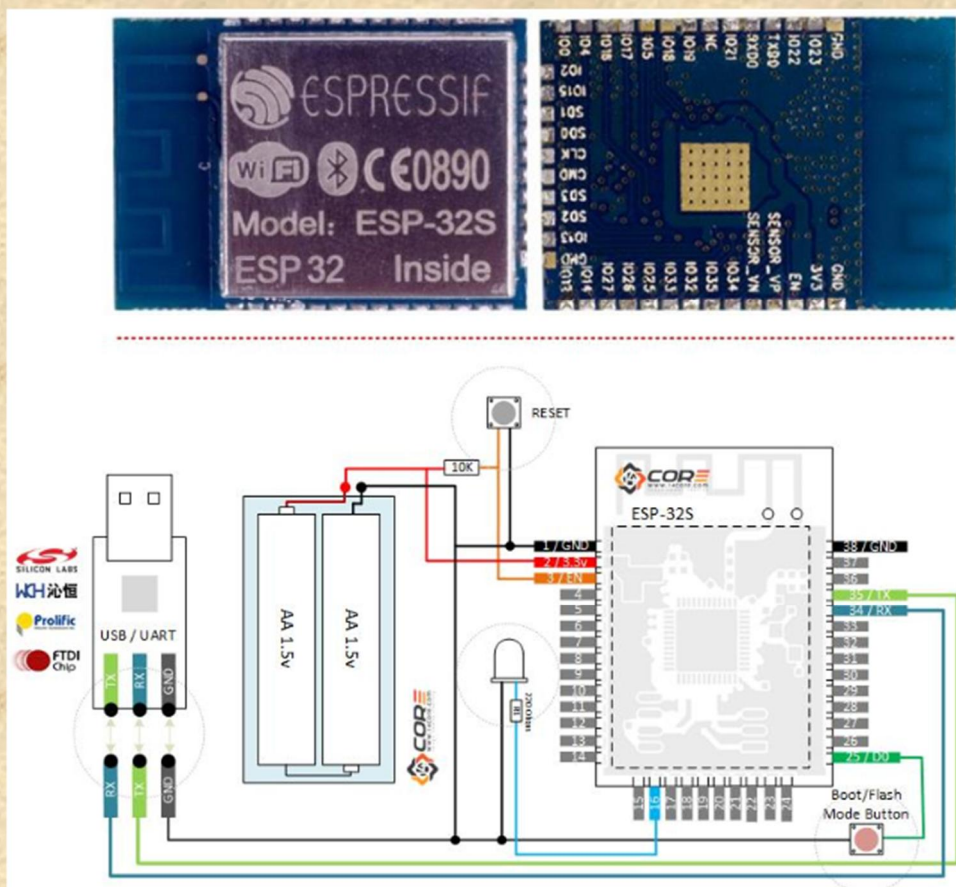
여기서는 ESP32를 이용하여 개발을 편리하도록 개발보드를 제작해 보도록 하겠습니다.

개발보드의 기능은 다음과 같습니다.

- 브레드보드에 사용할 수 있도록 2.0mm의 핀배치를 2.54mm의 배치로 변경하도록 합니다.
- RESET과 FLASH핀을 택트 버튼으로 제어할 수 있도록 합니다.
- USB\_TTL 단자를 연결할 수 있도록 합니다.
- AMS1117 3.3V 레귤레이터를 내장하여 5.0V전압으로 구동이 가능하도록 합니다.

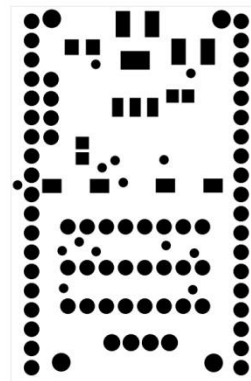
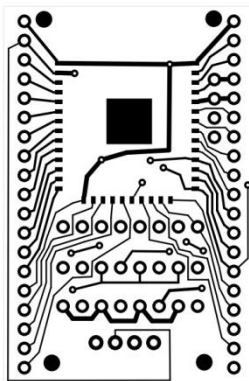
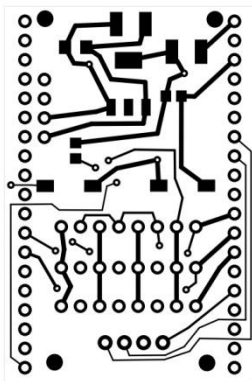
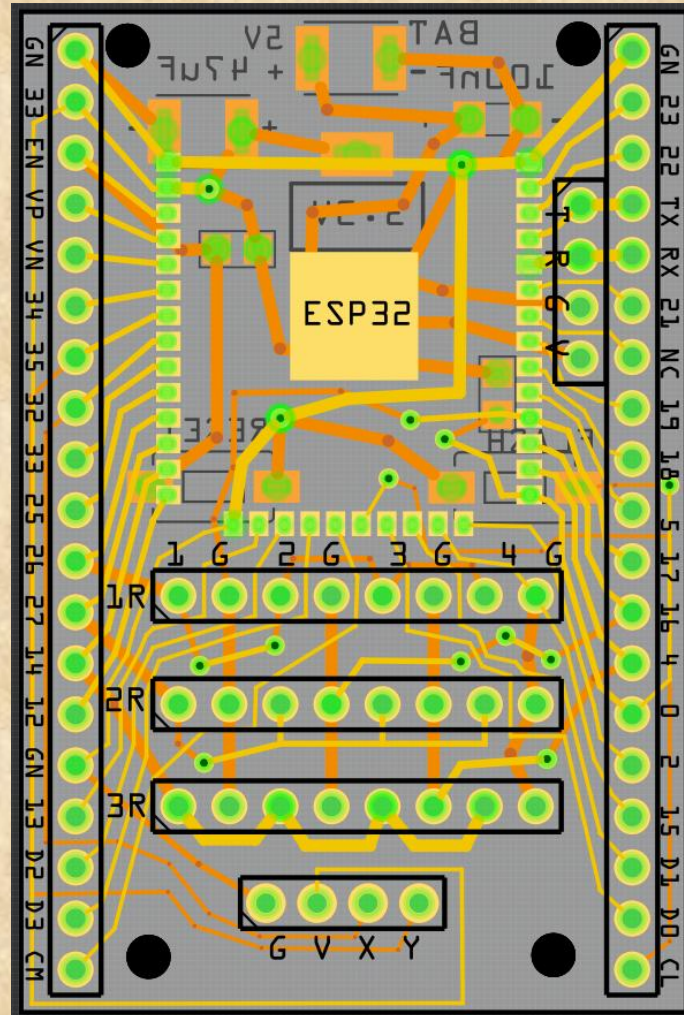
개발 보드를 위한 기본 회로는 아래와 같습니다.

RESET핀과 BOOT핀을 PULL-UP하여 연결하고 버튼을 이용하여 컨트롤 할 수 있도록 합니다.

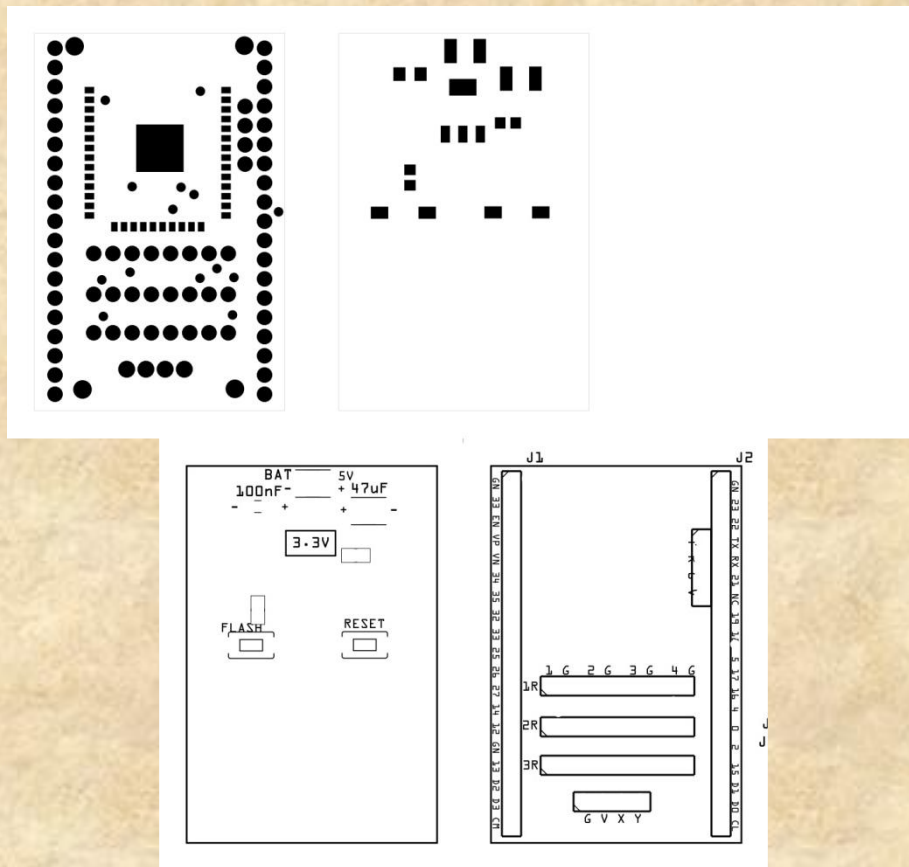




## ■ PCB설계







PCB의 설계는 위와 같이 33mm x 50mm의 기판으로 설계하였습니다.

설계상의 특징은 다음과 같습니다.

- 핀 간격을 브레드보드와 핀헤더 연결이 가능하도록 2.54mm로 변환
- AMS1117레귤레이터를 이용하여 5V 사용 가능하도록 수정
- RESET과 FLASH버튼 추가
- USB\_TTL 단자를 연결단자 추가
- 하단에 KEYPAD를 연결할 수 있는 4x3 버튼 배열
- 하단에 JOYSTICK 가변저항을 연결할 수 있는 단자 제작



## ■ PCB도착 및 부속 실장

PCB 가 도착하였습니다. 제작된 PCB 는 3.3cm x 5cm 사이즈로 아래와 같습니다.



기본적으로 납땜을 해야 하는 부속은 ESP32 와 2 개의 버튼(FLASH, RESET) 그리고 AMS1117 3.3V 와 10K 저항(SMD 타입), 핀헤더(USB TTL 연결 및 전원연결을 위한)를 납땜하면 ESP32 의 사용은 가능합니다.



펌웨어를 업로드 할 경우 USBTTL 을 연결하고 FLASH 버튼을 누른 상태에서 RESET 버튼을 눌렀다 떼면 펌웨어 업로드 모드로 변경이 되며 아두이노에서 업로딩을 하시면 됩니다.





## ■ PS2형식 리모컨 테스트 하기

PS2 형식의 리모컨은 X 축 및 Y 축이 GPIO32 와 35 번핀에 연결이 되어 있습니다.

이 핀에 조이스틱을 연결하고 ESP32 의 analogRead 를 이용하여 조이스틱의 값을 시리얼 플로터에 출력해 보겠습니다.

ESP8266 의 경우 ADC 의 기준전압이 1.0V 였습니다.

하지만 ESP32 의 경우 ADC 의 기준전압은 기본 1.1V 이외의 다른 전압을 사용할 수 있습니다.

이부분의 설명은 아래의 링크 자료를 참조하시기 바랍니다.

The default ADC full-scale voltage is 1.1V. To read higher voltages (up to the pin maximum voltage, usually 3.3V) requires setting >0dB signal attenuation for that ADC channel.

### Note

For any given channel, this function must be called before the first time `adc1_get_raw()` is called for that channel.

### Note

This function can be called multiple times to configure multiple ADC channels simultaneously. `adc1_get_raw()` can then be called for any configured channel.

When **VDD\_A** is 3.3V:

- 0dB attenuation (ADC\_ATTEN\_DB\_0) gives full-scale voltage 1.1V
- 2.5dB attenuation (ADC\_ATTEN\_DB\_2\_5) gives full-scale voltage 1.5V
- 6dB attenuation (ADC\_ATTEN\_DB\_6) gives full-scale voltage 2.2V
- 11dB attenuation (ADC\_ATTEN\_DB\_11) gives full-scale voltage 3.9V (see note below)

Due to ADC characteristics, most accurate results are obtained within the following approximate voltage ranges:

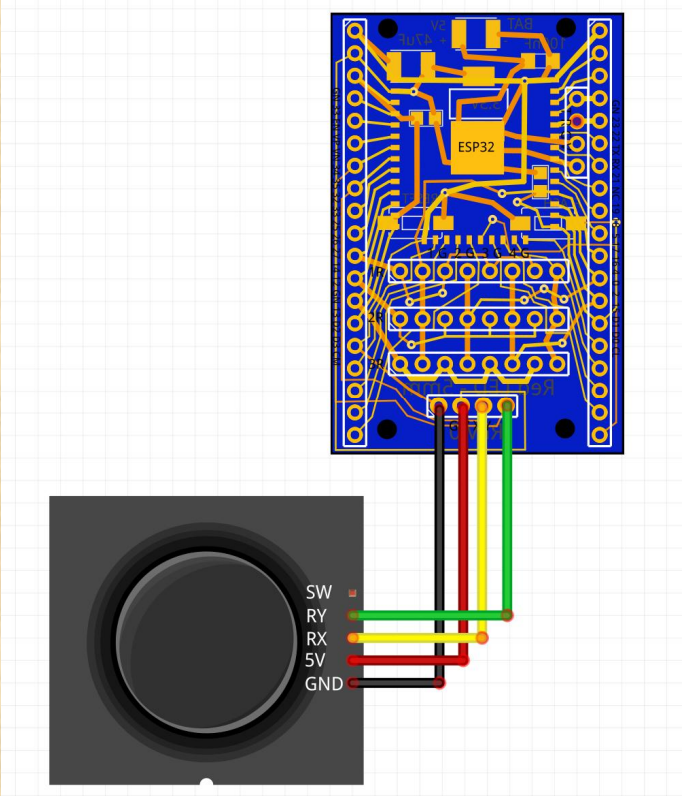
<https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/adc.html>

이를 이용하여 아두이노에서 테스트를 해보니 아두이노를 이용한 arduino esp32 의 경우 별다른 설정없이 analogRead 를 구동하게되면 1.1V 가 아닌 3.3V 를 기준으로 값이 측정되었습니다.



조이스틱의 연결회로도 는 다음과 같습니다.

회로도

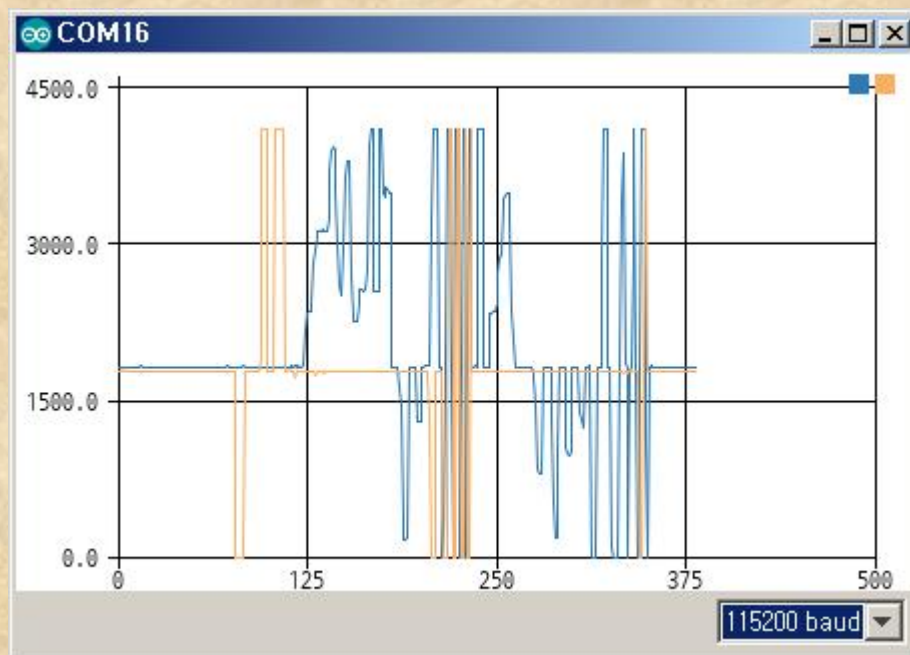


소스 코드

```
1234 1234567890123456789012345678901234567890123456789012345678901234
5 567890

1 void setup()
2 {
3     Serial.begin(115200);
4
5 }
6
7 void loop()
8 {
9     Serial.print(analogRead(32));
10    Serial.print(", ");
11    Serial.println(analogRead(35));
12    delay(100);
13 }
```

## 구동화면



조이스틱을 움직임에 따라 x 축의 값과 y 축의 값이 변경되는 걸 확인 할 수 있습니다.

관련자료 : joystick\_3

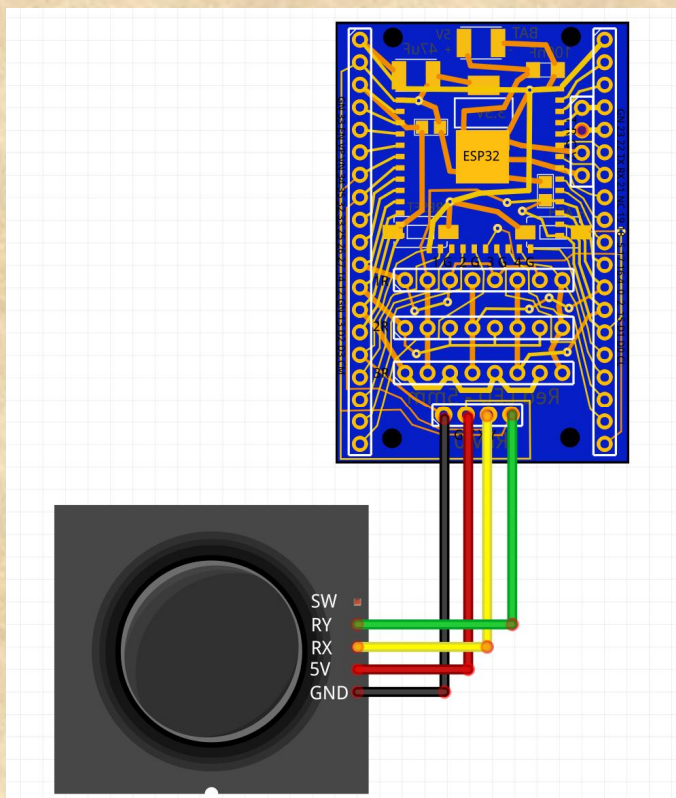


## ■ PS2형식 리모컨 조이스틱으로 사용하기

이번에는 위 PS2 형식 리모컨을 스마트폰의 조이스틱(게임패드)로 사용해 보도록 하겠습니다.

앞서 ESP32 를 BLE HID KEYPAD 로 사용하기를 이용하고 앞선 리모컨 테스트의 소스코드를 잘 조합하기만 하면 됩니다.

회로도



소스코드

1234 5	1234567890123456789012345678901234567890123456789012345678901234567890
1	#include <BLEDevice.h>
2	#include <BLEServer.h>
3	#include <BLEUtils.h>
4	#include <BLEHIDDevice.h>
5	#include <BLE2902.h>
6	
7	static BLEHIDDevice *pHID;
8	BLEServer *pServer;
9	BLECharacteristic *input;
10	
11	bool deviceConnected = false;
12	bool oldDeviceConnected = false;
13	

```

14 struct gamepad_report_t
15 {
16     int8_t left_x;
17     int8_t left_y;
18     uint16_t buttons;
19 };
20
21 bool operator!=(const gamepad_report_t& lhs, const gamepad_report_t& rhs)
22 {
23     return (lhs.left_x != rhs.left_x)
24         || (lhs.left_y != rhs.left_y)
25         || (lhs.buttons != rhs.buttons);
26 }
27
28 class MyServerCallbacks: public BLEServerCallbacks
29 {
30     void onConnect(BLEServer* pServer)
31     {
32         deviceConnected = true;
33         Serial.println("CONNECTED");
34     };
35
36     void onDisconnect(BLEServer* pServer)
37     {
38         deviceConnected = false;
39         Serial.println("DISCONNECTED");
40     }
41 };
42
43 void setup()
44 {
45     Serial.begin(115200);
46     pinMode(22, INPUT_PULLUP);
47     pinMode(23, INPUT_PULLUP);
48
49
50     // Create the BLE Device
51     BLEDevice::init("kProject Joystic"); // Give it a name
52
53     // Create the BLE Server
54     pServer = BLEDevice::createServer();
55     pServer->setCallbacks(new MyServerCallbacks());
56
57     // Instantiate HID Device
58     pHID = new BLEHIDDevice(pServer);
59     input = pHID->inputReport(1);
60
61     pHID->manufacturer()->setValue("kProject");
62     pHID->pnp(0x01, 0x02e5, 0xabcd, 0x0110);
63     pHID->hidInfo(0x00, 0x01);
64
65     BLESecurity *pSecurity = new BLESecurity();
66     pSecurity->setAuthenticationMode(ESP_LE_AUTH_BOND);
67
68     // Set Report Map
69     const uint8_t reportMap[] =
70     {
71         0x05, 0x01, // USAGE_PAGE (Generic Desktop)
72         0x09, 0x05, // USAGE (Game Pad)
73         0xa1, 0x01, // COLLECTION (Application)
74         0xa1, 0x02, // COLLECTION (Logical)
75         0x85, 0x01, // REPORT_ID (1)

```



```

76
77     0x75, 0x08,          // REPORT_SIZE (8)
78     0x95, 0x02,          // REPORT_COUNT (2)
79     0x05, 0x01,          // USAGE_PAGE (Generic Desktop)
80     0x09, 0x30,          // USAGE (X)
81     0x09, 0x31,          // USAGE (Y)
82     0x15, 0x81,          // LOGICAL_MINIMUM (-127)
83     0x25, 0x7F,          // LOGICAL_MAXIMUM (127)
84     0x81, 0x02,          // INPUT (Data, Var, Abs)
85
86     0x75, 0x01,          // REPORT_SIZE (1)
87     0x95, 0x0b,          // REPORT_COUNT (11)
88     0x15, 0x00,          // LOGICAL_MINIMUM (0)
89     0x25, 0x01,          // LOGICAL_MAXIMUM (1)
90     0x05, 0x09,          // USAGE_PAGE (Button)
91     0x19, 0x01,          // USAGE_MINIMUM (Button 1)
92     0x29, 0x0b,          // USAGE_MAXIMUM (Button 11)
93     0x81, 0x02,          // INPUT (Data, Var, Abs)
94     // PADDING for byte alignment
95     0x75, 0x01,          // REPORT_SIZE (1)
96     0x95, 0x05,          // REPORT_COUNT (5)
97     0x81, 0x03,          // INPUT (Constant, Var, Abs)
98
99     0xc0,                // END_COLLECTION
100    0xc0                  // END_COLLECTION
101    };
102
103    pHID->reportMap((uint8_t*)reportMap, sizeof(reportMap));
104    int numReport = sizeof(reportMap);
105    Serial.println(numReport);
106
107    // SetupGamepad();
108
109    // Start the service
110    pHID->startServices();
111
112    // Start advertising
113    BLEAdvertising *pAdvertising = pServer->getAdvertising();
114    pAdvertising->setAppearance(HID_GAMEPAD);
115    pAdvertising->addServiceUUID(pHID->hidService()->getUUID());
116    pAdvertising->start();
117
118    Serial.println("Waiting a client connection to notify...");
119    }
120
121    gamepad_report_t oldValue, newValue;
122
123    void loop()
124    {
125        if (deviceConnected)
126        {
127            // AXIS
128            {
129                int x_val = map(analogRead(32), 0, 4096, -125, 125);
130                int y_val = map(analogRead(35), 0, 4096, -125, 125);
131                newValue.left_x = x_val;
132                newValue.left_y = y_val;
133            }
134
135            // BUTTONS
136            {
137                newValue.buttons = 0;

```

```
138         }
139
140         if (newValue != oldValue)
141         {
142             uint8_t a[] = {newValue.left_x, newValue.left_y, newValue.buttons, (newValue.button
ns >> 8)};
143             input->setValue(a, sizeof(a));
144             input->notify();
145             oldValue = newValue;
146         }
147         delay(5);
148     }
149
150     // Connecting
151     if (deviceConnected && !oldDeviceConnected)
152     {
153         oldDeviceConnected = deviceConnected;
154     }
155
156     // Disconnecting
157     if (!deviceConnected && oldDeviceConnected)
158     {
159         delay(500); // give the bluetooth stack the chance to get things ready
160         pServer->startAdvertising();
161         Serial.println("restart advertising");
162         oldDeviceConnected = deviceConnected;
163     }
164 }
165
166
167
```

구동 영상

관련자료 : joystick\_4