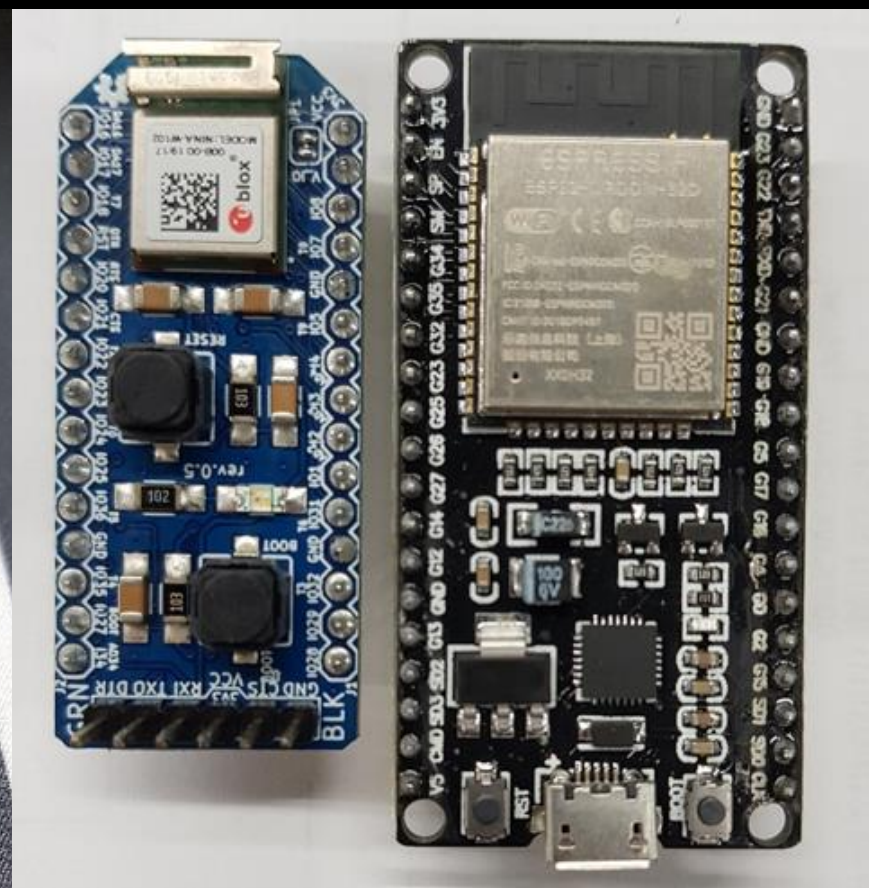


중급과정

ESP32

온라인 워크숍



과정 목차

❖ 2일차

1. Iwip + WIFI 간단 리뷰
2. ESP32 Bluetooth Architecture
3. ESP32 BT Classic A2DP
 - > I2S
 - > Codec
 - > A2DP Audio 재생 및 AVRC Volume Control 제어 실습
4. ESP32 BT Classic SPP
 - > ESP32 SPP acceptor demo
 - > ESP32 gpio LED
 - > SPP profile을 활용하여 Android App으로 LED 제어실습

시작하기전에...

2. ESP32 Bluetooth Architecture

- ✓ Bluetooth는 단거리 데이터 교환을 위한 무선 기술 표준으로 견고성, 저전력 소비 및 저렴한 비용 등의 이점이 있습니다. Bluetooth 시스템은 Classic Bluetooth와 Bluetooth Low의 두 가지 범주로 나눌 수 있습니다.
- ✓ ESP32는 듀얼 모드 블루투스를 지원합니다. 즉, 클래식 블루투스와 BLE 모두 ESP32에서 지원됩니다.
- ✓ 기본적으로 Bluetooth 프로토콜 스택은 "컨트롤러 스택"과 "호스트 스택"의 두 부분으로 나뉩니다. 컨트롤러 스택에는 PHY, 베이스 밴드, 링크 컨트롤러, 링크 관리자, 장치 관리자, HCI 및 기타 모듈이 포함되어 하드웨어 인터페이스에 사용됩니다.
- ✓ 관리 및 링크 관리. 호스트 스택에는 L2CAP, SMP, SDP, ATT, GATT, GAP 및 다양한 프로필이 포함되어 있으며 응용 프로그램 계층에 대한 인터페이스로 작동하므로 응용 프로그램 계층이 Bluetooth 시스템에 쉽게 액세스 할 수 있습니다.
- ✓ 블루투스 호스트는 컨트롤러와 동일한 장치 또는 다른 장치에서 구현됩니다. 두 가지 방법 모두 ESP32에서 지원됩니다.

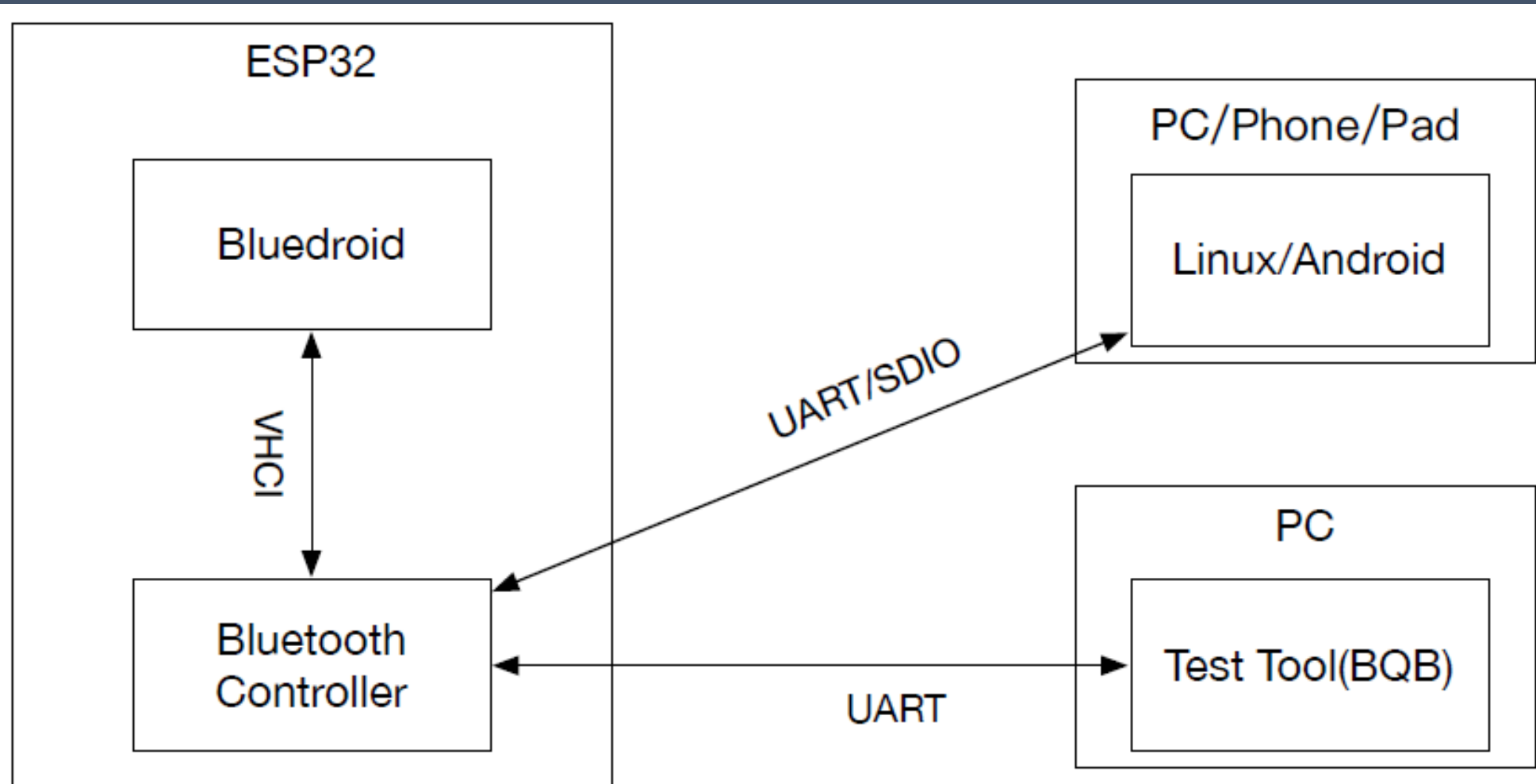
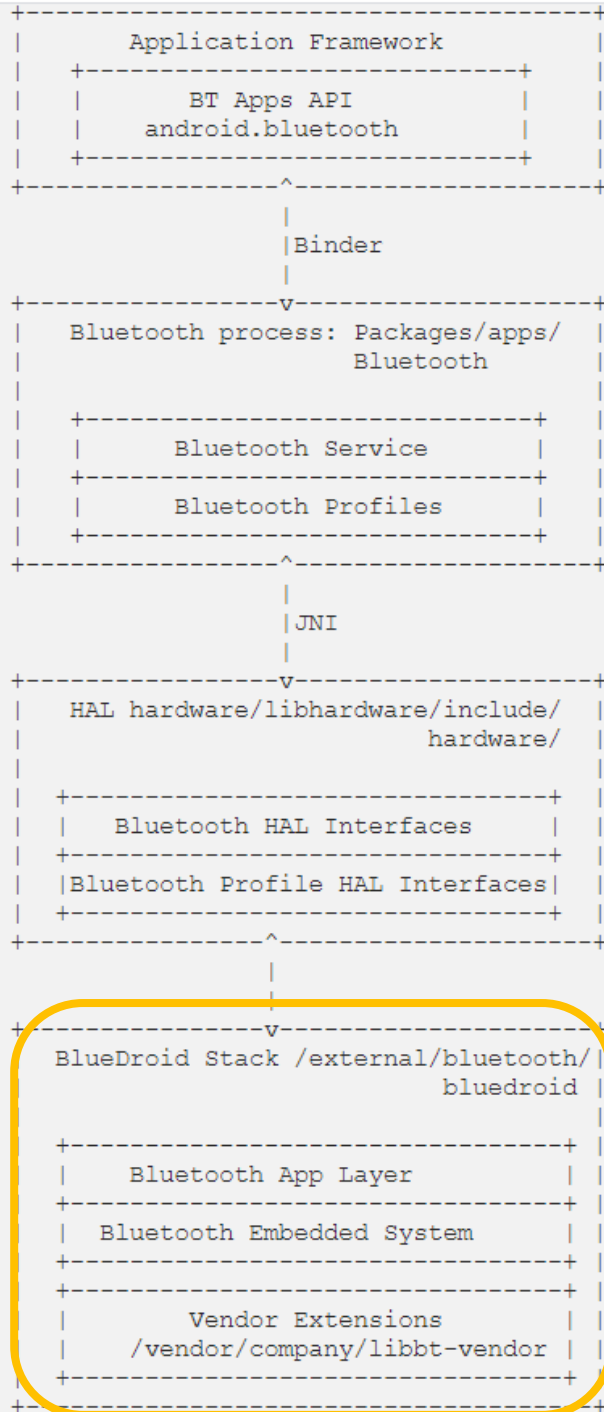


Figure 1-1.The architecture of Bluetooth host and controller in ESP-IDF

- 시나리오 1 (기본 ESP-IDF 설정) : BLUEDROID가 Bluetooth 호스트로 선택되고 VHCI (소프트웨어 구현 가상 HCI 인터페이스)가 Bluetooth 호스트와 컨트롤러 간의 통신에 사용됩니다. 이 시나리오에서 BLUEDROID와 컨트롤러는 모두 동일한 장치 (예 : ESP32 칩)에서 구현되므로 Bluetooth 호스트를 실행하는 추가 PC 또는 기타 호스트 장치가 필요합니다.
- 시나리오 2 : ESP32 시스템은 Bluetooth 컨트롤러로만 사용되며 Bluetooth 호스트를 실행하는 추가 장치 (예 : BlueZ를 실행하는 Linux PC 또는 BLUEDROID를 실행하는 Android 장치 등)가 필요합니다. 이 시나리오에서 컨트롤러와 호스트는 서로 다른 기기에서 구현되며 휴대 전화의 경우와 매우 유사합니다. 패드 또는 PC.
- 시나리오 3 : 이 시나리오는 시나리오 2와 유사합니다. 차이점은 BQB 컨트롤러 테스트 (또는 다른 인증)에서 ESP32를 테스트 도구에 연결하여 테스트 할 수 있으며 UART가 IO 인터페이스로 활성화 된 상태입니다.



Bluedroid는 Broadcom에서 제공하는 스택이며 현재 Android의 오픈 소스입니다. 안드로이드 4.2이상 부터 사용, 이전 버전은 BlueZ 스택 사용

<출처>

<https://www.it-swarm-ko.tech/ko/android/bluez-vs-bluedroid-%eb%b8%94%eb%a3%a8%ed%88%ac%ec%8a%a4-%ec%8a%a4%ed%83%9d/1042556430/>

<https://medium.com/@zpcat/bluedroid-stack-in-android-564c58b451f4>

1.1.2. Selection of the HCI Interfaces

ESP32 시스템에서 HCI는 한 번에 하나의 IO 인터페이스 만 사용할 수 있습니다. 즉, UART가 활성화되면 VHCI 및 SDIO와 같은 다른 인터페이스는 비활성화됩니다. ESP-IDF (V2.1 이상 버전)의 menuconfig 화면에서 다음과 같이 Bluetooth HCI IO 인터페이스를 VHCI 또는 UART로 구성 할 수 있습니다.

```
--- Bluetooth
[ ] Bluedroid Bluetooth stack enabled --->
[ ] HCI use UART as IO (NEW) ----
```

Figure 1-2. Configuration of the HCI IO interface

Bluedroid Bluetooth 스택 활성화 옵션을 선택하면 VHCI가 IO 인터페이스로 활성화되고 HCI가 UART를 IO (NEW)로 사용 옵션이 메뉴에서 사라집니다.

HCI가 UART를 IO로 사용 (신규) 옵션을 선택하면 UART가 IO 인터페이스로 활성화됩니다. 현재 다른 IO는 ESP-IDF에서 지원되지 않습니다. SPI와 같은 다른 IO를 사용하려면 SPI-VHCI 브리지가 필요합니다.

옵션 1:

Bluedroid Bluetooth 스택 사용 옵션을 선택하면 다음 화면이 표시됩니다.

```
-- Bluedroid Bluetooth stack enabled
(3072) Bluetooth event (callback to application) task stack size
[ ]   Bluetooth memory debug
[ ]   Classic Bluetooth
[ ]   Release DRAM from Classic BT controller
[*]   Include GATT server module(GATTS)
[*]   Include GATT client module(GATTC)
[*]   Include BLE security module(SMP)
[ ]   Close the bluedroid bt stack log print
(4)   BT/BLE MAX ACL CONNECTIONS(1~7)
```

Figure 1-3.VHCI configuration

여기에서 사용자는 다음 항목을 구성 할 수 있습니다.

- Bluetooth 이벤트 (응용 프로그램으로의 콜백) 작업 스택 크기 : BTC 작업의 크기를 설정합니다.
- Bluedroid 메모리 디버그 : BLUEDROID 메모리를 디버그합니다.
- 클래식 블루투스 : 클래식 블루투스를 활성화합니다.
- Classic BT Controller에서 DRAM 해제 : Classic Bluetooth Controller에서 DRAM을 해제합니다.
- GATTS (GATT 서버 모듈) 포함 : GATTS 모듈을 포함합니다. (GATT Server)
- GATTC (GATT 클라이언트 모듈) 포함 : GATTC 모듈을 포함합니다. (GATT Client)
- BLE 보안 모듈 (SMP) 포함 : SMP 모듈을 포함합니다.
- bluedroid bt 스택 로그 인쇄(print)를 닫습니다. BLUEDROID 인쇄를 닫습니다.
- BT / BLE MAX ACL CONNECTIONS (1 ~ 7) : 최대 ACL 연결 수를 설정합니다.

옵션 2 :

HCI가 UART를 IO로 사용 옵션을 선택하면 다음 화면이 표시됩니다.

-- HCI use UART as IO

(1) UART Number for HCI (NEW)

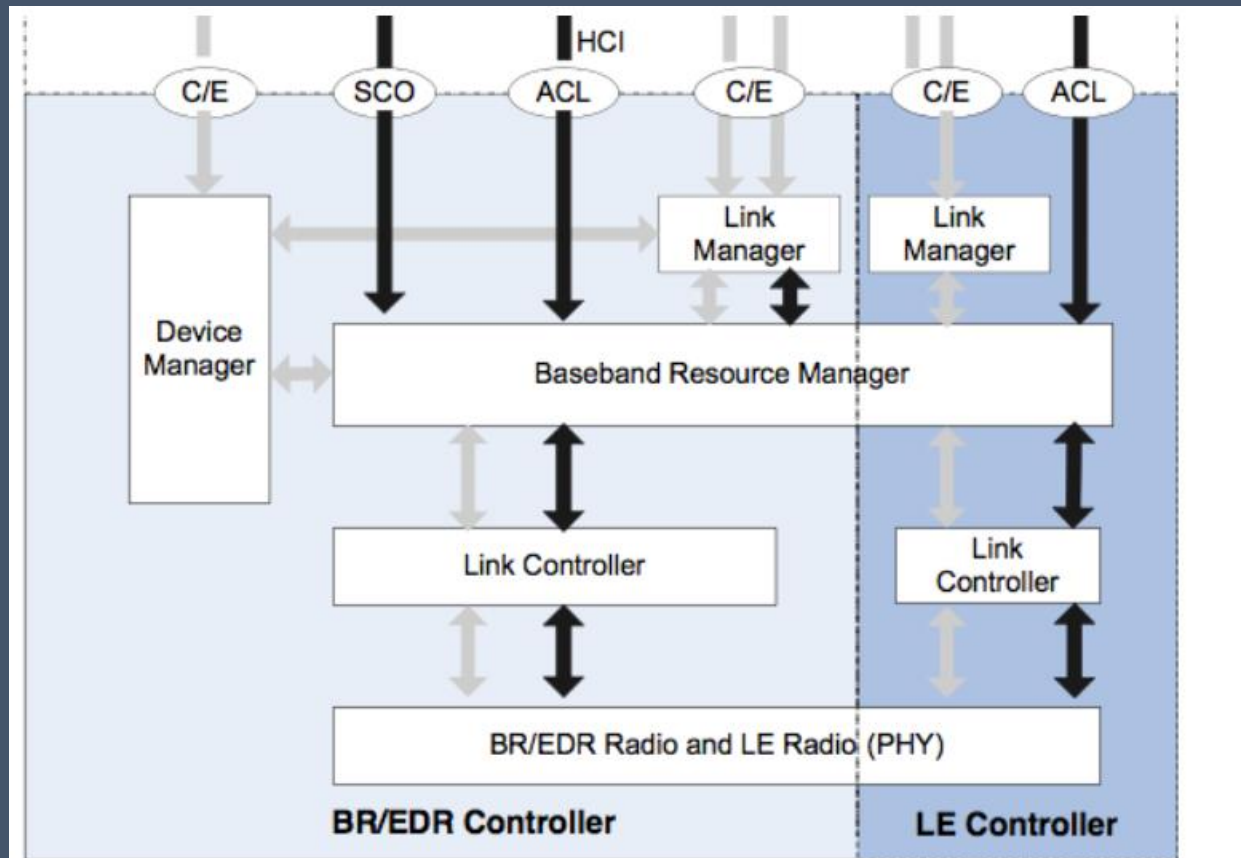
(921600) UART Baudrate for HCI (NEW)

Figure 1-4. UART configuration

사용자는 여기에서 HCI를 위한 UART 번호(UART Port Number)와 HCI를 위한 UART Baudrate를 설정할 수 있습니다. UART를 HCI IO 인터페이스로 사용하려면 CTS / RTS를 지원해야 합니다.

1.2.1 Controller

이러한 기능은 라이브러리 형태의 개발자, 컨트롤러에 액세스 할 수있는 일부 API도 제공됩니다. 자세한 내용은 설명서를 참조하십시오.



www.CodeZoo.co.kr

Figure 1-5. Architecture of the Classic BT & BLE controller (from the SIG BT CORE 4.2)

1.2.2. BLUEDROID

1.2.2.1. Overall Architecture

ESP-IDF에서 크게 수정 된 BLUEDROID가 Bluetooth 호스트 (Classic BT + BLE)로 사용됩니다. BLUEDROID는 완전한 기능 세트를 가지고 있으며 일반적으로 사용되는 표준과 아키텍처 디자인을 대부분 지원하고 비교적 복잡합니다.

그러나 수정 된 BLUEDROID는 대부분의 코드를 BTA 계층 아래에 유지하고 기본 사양 및 기타 제어 계층으로 더 얇은 BTC 계층을 사용하여 BTIF 계층 코드를 거의 완전히 삭제합니다.

수정 된 BLUEDROID의 아키텍처와 컨트롤러와의 관계는 아래 그림에 표시되어 있습니다.

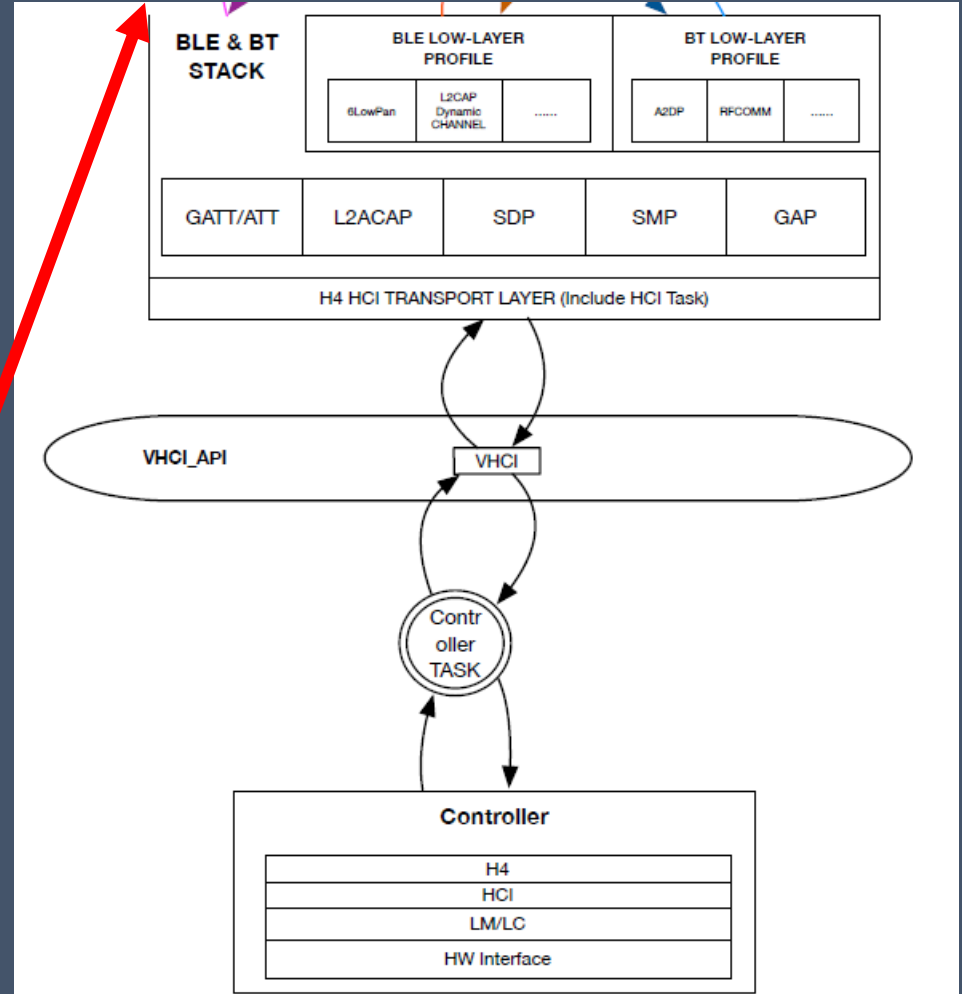
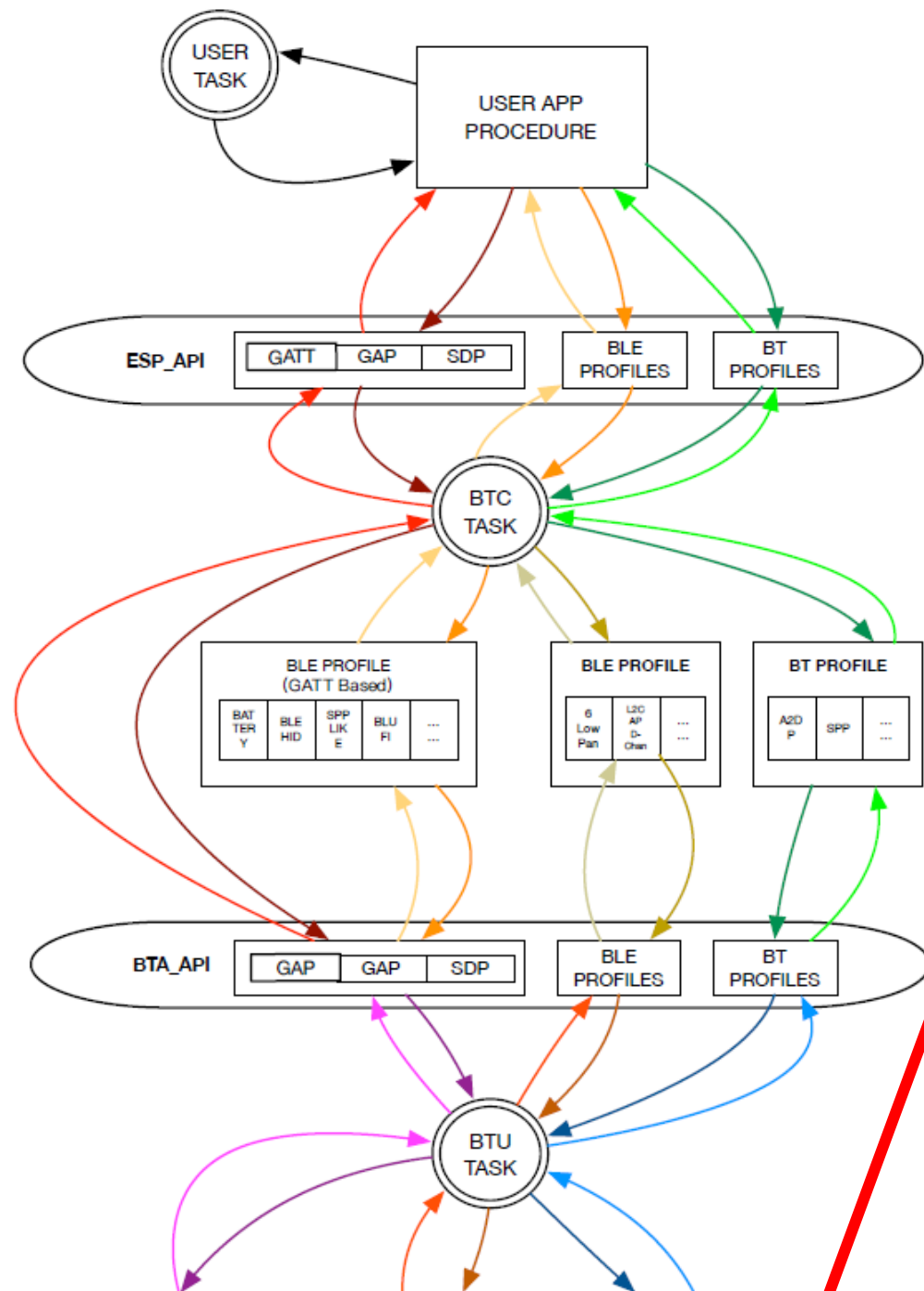


Figure 1-6.ESP32 BLUEDROID diagram

노트 :
이 다이어그램은 주로 HCI TASK와 같은 세부 정보가 아닌 BLUEDROID의 계층 구조를 설명합니다. 각 레이어에 대한 자세한 정보는 아래에서 확인할 수 있습니다.

- ✓ 위 그림과 같이 BLUEDROID는 주로 BTU 계층과 BTC 계층 (HCI 제외)의 두 계층으로 나눌 수 있습니다. 각 계층은 해당 작업을 담당합니다. 보다 구체적으로, **BTU 계층은 주로 처리를 담당**합니다.
- ✓ L2CAP, GATT / ATT, SMP, GAP 및 기타 프로파일을 포함하여 Bluetooth 호스트의 하위 계층 프로토콜 스택. BTU 계층은 "bta"접두사가있는 인터페이스를 제공합니다. **BTC 계층은 주로 "esp"접두어가있는 지원되는 인터페이스를 응용 프로그램 계층에 제공하고 GATT 기반 프로파일을 처리하고 기타 작업을 처리**합니다. 모든 API는 ESP_API 계층에 있습니다. **개발자는 접두사"esp"가있는 Bluetooth API를 사용해야합니다.**
- ✓ 위의 그림은 HCI 계층을 자세히 설명하지 않습니다. 실제로 HCI 계층에는 ESP-IDF V2.1 및 이전 버전의 설계에서 하향 및 상향 데이터를 처리하는 두 가지 작업이 있습니다.
- ✓ 이 아키텍처의 설계 논리는 사용자 관련 작업 부하를 최소화하고 Bluetooth 관련 작업을 BTC 계층으로 넘겨 Bluetooth 구조를 간소화하는 것입니다.
- ✓ 레거시 이유와 실제 수요로 인해 RFCOMM 및 A2DP와 같은 일부 클래식 Bluetooth 프로파일과 기타 하위 계층 프로토콜은 BTU 계층에서 구현되는 반면, 일부 프로토콜은 절차적 제어와 관련되거나 ESP-API는 BTC 계층에서 구현됩니다.
- ✓ 6LowPan 또는 Dynamic L2CAP Channel과 같은 Bluetooth Low Energy의 일부 프로파일 및 하위 계층 기능은 BTU 계층에서 구현되므로 BTC를 통해 애플리케이션 계층에 ESP-API를 제공합니다.

1.2.2.2. OS-related Adaptation

BLUEDROID의 시스템과 관련된 일부 인터페이스에는 OSI 조정이 필요합니다. 관련된 기능에는 타이머 (알람), task (스레드), Future Await / Ready (Semaphore) 및 Allocator / GKI (malloc / free)가 포함됩니다.

BLUEDROID의 FreeRTOS 타이머는 알람으로 패키징되어 있으며 특정 작업을 트리거하는 타이머를 시작하는 데 사용됩니다.

BLUEDROID에서 POSIX 스레드는 FreeRTOS task로 대체되었으며 FreeRTOS 대기열을 사용하여 task을 트리거합니다 (즉, 깨우기).

BLUEDROID에서 Future Await / Ready 기능은 차단을 달성하는 데 사용됩니다.

Future Lock은

freeRTOS의 xSemaphoreTake를 future await 함수로 패키징하고

xSemaphoreGive를 future ready 함수로 패키징합니다.

future await 및 future ready 함수를 동일한 task 컨텍스트에서 호출 할 수는 없습니다.

BLUEDROID에서 표준 라이브러리의 malloc / free는 메모리를 예약 (mallocs)하거나 해제하는 할당 자 함수로 패키징됩니다. 또한 GKI 함수는 malloc / free를 GKI getbuf / GKI freebuf의 핵심 함수로 사용합니다.

1.2.3. Bluetooth Directory Introduction

ESP-IDF의 component / bt 화면에서 다음 하위 폴더를 볼 수 있습니다.
서브 파일 :

```
├─ Kconfig
├─ bluebird
│   ├── api
│   ├── bta
│   ├── btc
│   ├── btcore
│   ├── btif
│   ├── device
│   ├── external
│   ├── gki
│   ├── hci
│   ├── include
│   ├── main
│   ├── osi
│   ├── stack
│   └── utils
├─ bt.c
├─ component.mk
├─ include
│   └── bt.h
└─ lib
    ├── LICENSE
    ├── README.rst
    └── libbtdm_app.a
```

Figure 1-7. Component/bt in ESP-IDF

Table 1-1. Description of component/bt in ESP-IDF

Dictionary	Description	Remarks
— <i>Kconfig</i>	Menuconfig files	—
— <i>bluedroid</i>	BLUEDROID home entry	—
— <i>api</i>	The API directory, which includes all the APIs (except for those that are related to the Controller)	—
— <i>bta</i>	The Bluetooth adaptation layer, which is suitable for the interface of some bottom layer protocols in the host.	—
— <i>btc</i>	The Bluetooth control layer, which controls the upper-layer protocols (including profiles) and miscellaneous items in the host.	—
— <i>btcore</i>	Some of the original <code>feature/bdaddr</code> conversion functions	To be abandoned
— <i>btif</i>	Some <code>call out</code> functions used by the BTA layer	To be abandoned
— <i>device</i>	Related to the device control of the Controller, e.g. the basic set of HCI CMD controller processes	—

모든 API를 포함하는 API 디렉토리
(컨트롤러와 관련된 API 제외)

더이상 개발하지 않음(포기)

www.CodeZoo.co.kr

Dictionary	Description	Remarks
— <i>external</i>	Codes that are not directly related to the Bluetooth, but are still usable, e.g. the SBC codec software programs	—
— <i>gki</i>	The management codes that are commonly used by the BLUEDROID memory, e.g. the buffer and queue.	—
— <i>hci</i>	HCI layer protocols	—
— <i>include</i>	The top-layer BLUEDROID directory	—
— <i>main</i>	Main program (mainly to start or halt the process)	—
— <i>osi</i>	OS interfaces (including semaphore/timer/thread, etc.)	—
— <i>stack</i>	The bottom layer protocol stacks in the Host (GAP/ATT/GATT/SDP/SMP, etc.)	—
— <i>utils</i>	Practical utilities	—
— <i>bt.c</i>	Controller-related processing files	—
— <i>component.mk</i>	makefile	—
— <i>include</i>	Controller-related header file directory	—
— <i>bt.h</i>	Header files that contain the controller-related APIs	—
— <i>lib</i>	Controller library directory	—
— <i>LICENSE</i>	License	—
— <i>README.rst</i>	Readme files	—
— <i>libbtadm_app.a</i>	Controller library	—

esp-idf/components/bt/host/bluedroid

```
api
bta
btc
common
device
external
hci
Kconfig.in
main
stack
```

```
a2dp
avct
avdt
avrc
btm
btu
gap
gatt
hcic
include
l2cap
rfcomm
sdp
smp
```

- api : 모든 API를 포함하는 API 디렉토리 (컨트롤러와 관련된 API 제외)
- bta : 호스트에서 일부 하위 계층 프로토콜의 인터페이스에 적합한 Bluetooth adaptation layer
- btc : Bluetooth 제어 계층-상위 계층 프로토콜 (프로파일 포함) 및 호스트의 기타 항목을 제어
- common
- device : 컨트롤러의 장치 제어와 관련이 있음 (예 : HCI CMD 컨트롤러 프로세스의 기본 세트)
- external : 블루투스와의 직접 관련이 없지만 여전히 사용할 수 있는 코드 (예 : SBC 코덱 소프트웨어 프로그램)
- hci : HCI layer 프로토콜
- common/include : The top-layer BLUEDROID directory
- main : Main program (mainly to start or halt the process)
- stack : Host의 최하위계층 프로토콜 스택 (a2dp, avct, avdt, avrc, btm, btu, gap, gatt, hcic, l2cap, rfcomm, sdp, smp)

2. Classic Bluetooth

ESP-IDF 내 Classic Bluetooth 소개

2.1 Overview (개요)

ESP-IDF의 Bluetooth 호스트 스택은 BLUEDROID에서 시작되었으며 임베디드 응용 프로그램에 맞게 조정되었습니다. 하위 계층에서 Bluetooth 호스트 스택은 가상 HCI 인터페이스를 통해 Bluetooth 이중 모드 컨트롤러와 통신합니다. 상위 계층에서 Bluetooth 호스트 스택은 스택 관리를 위한 프로파일 및 API를 사용자 응용 프로그램에 제공합니다.

프로토콜은 메시지 형식과 특정 기능을 달성하기위한 절차를 정의합니다 (예 : 데이터 전송, 링크 제어, 보안 서비스 및 서비스 정보 교환. 반면, Bluetooth 프로파일은 PHY에서 L2CAP에 이르기까지 Bluetooth 시스템의 각 계층에 필요한 기능과 특징 및 핵심 사양 이외의 다른 프로토콜을 정의합니다.

다음은 현재 호스트 스택에서 지원되는 Classic BT 프로파일 및 프로토콜입니다.

- Profiles: GAP, A2DP(SNK), AVRCP(CT)
- Protocols: L2CAP, SDP, AVDTP, AVCTP

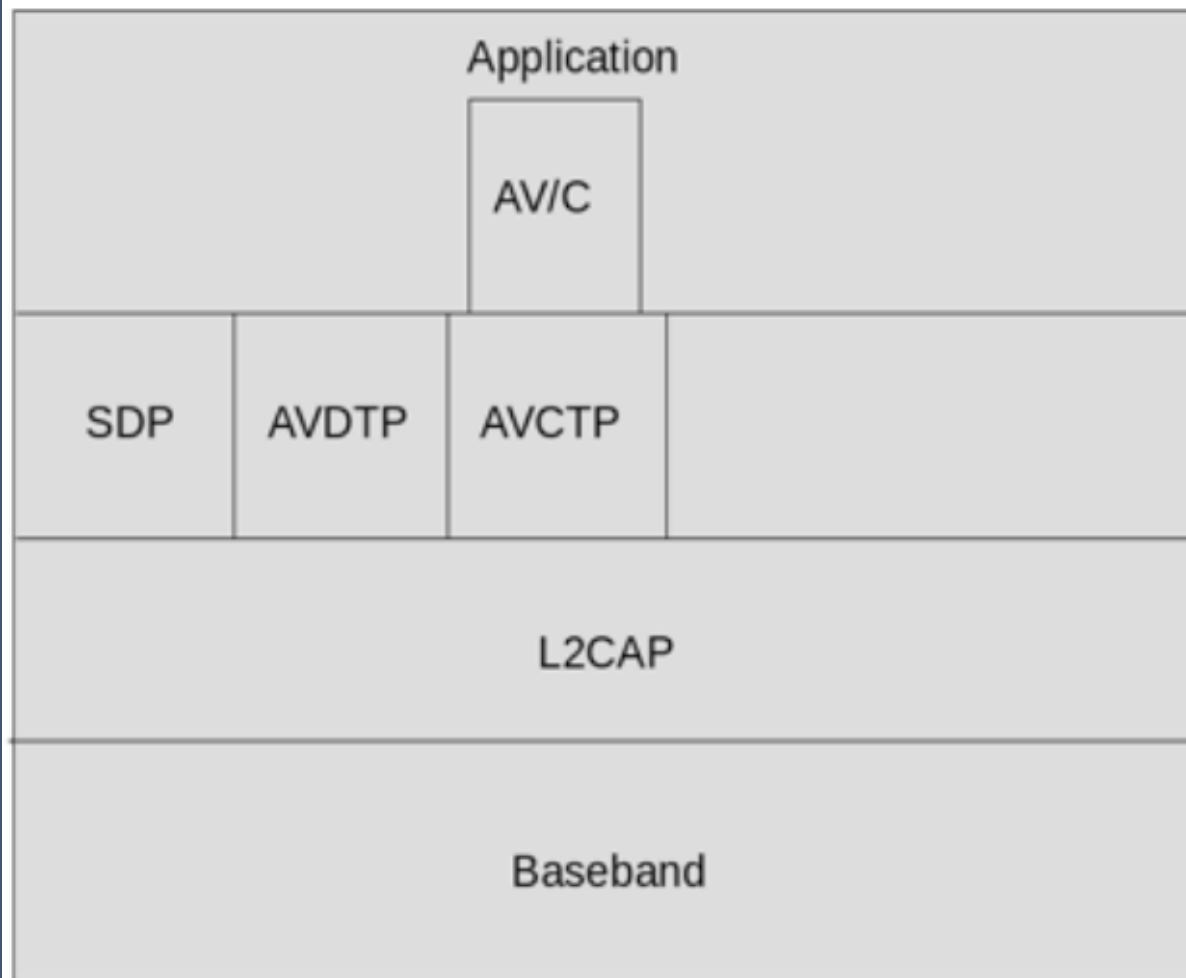


Figure 2-1. Profile Dependencies

그림 2-1에서 L2CAP 및 SDP는 Classic Bluetooth의 최소 호스트 스택에 필요합니다.

AVDTP, AV/C 및 AVCTP는 핵심 사양 외부에 있으며 특정 프로파일에서 사용됩니다.

2.1.1 L2CAP

OSI 계층 2 Bluetooth 프로토콜 인 Bluetooth L2CAP (Logical Link Control and Adaptation Protocol)는 더 높은 수준의 프로토콜 멀티플렉싱, 패킷 분할 및 재 조립은 물론 서비스 정보 품질 제공을 지원합니다. L2CAP를 사용하면 다른 응용 프로그램이 ACL-U 논리 링크를 공유 할 수 있습니다. 응용 프로그램 및 서비스 프로토콜은 채널 지향 인터페이스를 사용하여 L2CAP과 인터페이스하여 다른 장치의 동등한 엔터티에 대한 연결을 만듭니다.

L2CAP 채널은 L2CAP 채널 구성 절차를 통해 선택된 6 가지 모드 중 하나에서 작동 할 수 있습니다. 작동 모드는 이들이 제공 할 수있는 QoS와 구별되며 다른 애플리케이션 조건에서 활용됩니다. 이러한 모드는 다음과 같습니다.

- 기본 L2CAP 모드
- 흐름 제어 모드
- 재전송 모드
- 강화 된 재전송 모드
- 스트리밍 모드
- LE 크레딧 기반 흐름 제어 모드

ACL-U 논리 링크의 경우 지원되는 작동 모드는 기본 L2CAP 모드, 강화된 재전송 모드 및 스트리밍 모드입니다. 다른 기능의 경우 L2CAP 신호 채널이 지원되는 고정 채널이고 프레임 검사 시퀀스 (FCS)도 지원되는 옵션입니다.

2.1.2. SDP

SDP (Service Discovery Protocol)는 응용 프로그램이 피어 Bluetooth 장치가 제공하는 서비스를 검색하고 사용할 수 있는 서비스의 특성을 결정하는 수단을 제공합니다. SDP는 SDP 서버와 SDP 클라이언트. 서버는 서버와 관련된 서비스의 특성을 설명하는 서비스 레코드 목록을 유지 관리합니다. 클라이언트는 SDP 요청을 발행하여이 정보를 검색 할 수 있습니다.

SDP 클라이언트와 서버는 모두 호스트 스택에서 구현되며이 모듈은 A2DP 및 AVRCP와 같은 프로파일에서만 사용되며 현재 사용자 응용 프로그램에 대한 API를 제공하지 않습니다.

2.1.3. GAP

GAP (Generic Access Profile)는 장치 검색, 연결 및 보안의 모드와 절차에 대한 설명을 제공합니다.

당분간 Classic Bluetooth 호스트 스택에는 제한된 수의 GAP API 만 제공됩니다. 응용 프로그램은 이러한 API를 마치 "수동 장치"인 것처럼 피어 Bluetooth 장치에서 검색하고 연결할 수있는 것처럼 사용할 수 있습니다.

그러나 조회 절차를 시작하는 데 사용되는 API는 현재 고객 (사용자 애플리케이션)에게 제공되지 않습니다.

보안 측면에서 IO 기능은 "입력 없음, 출력 없음"으로 하드 코딩되므로 Secure Simple Pairing의 "Just Works"연관 모델 만 지원됩니다. 링크 키 저장은 자동으로 수행됩니다. 주인.

클래식 블루투스를 위한 더 많은 GAP API가 다음에 출시 될 예정입니다. 더 강력하고 다른 연관 모델을 지원하는 보안 API는 가까운 시일 내에 제공 될 것입니다. 장치 검색 및 링크 정책 설정을위한 API도 나중에 제공됩니다.

2.1.4. A2DP and AVRCP

A2DP (Advanced Audio Distribution Profile)는 ACL 채널에서 모노 또는 스테레오로 고품질 오디오 콘텐츠의 분배를 실현하는 프로토콜 및 절차를 정의합니다.

A2DP는 오디오 스트리밍을 처리하며 종종 오디오 / 비디오 제어 기능이 포함 된 AVRCP (Audio / Video Remote Control Profile)와 함께 사용됩니다. 그림 2-2는 프로파일의 구조와 종속성을 보여줍니다 [1] :

[1] : 고급 오디오 분배 프로파일 사양, 개정 1.3.1.

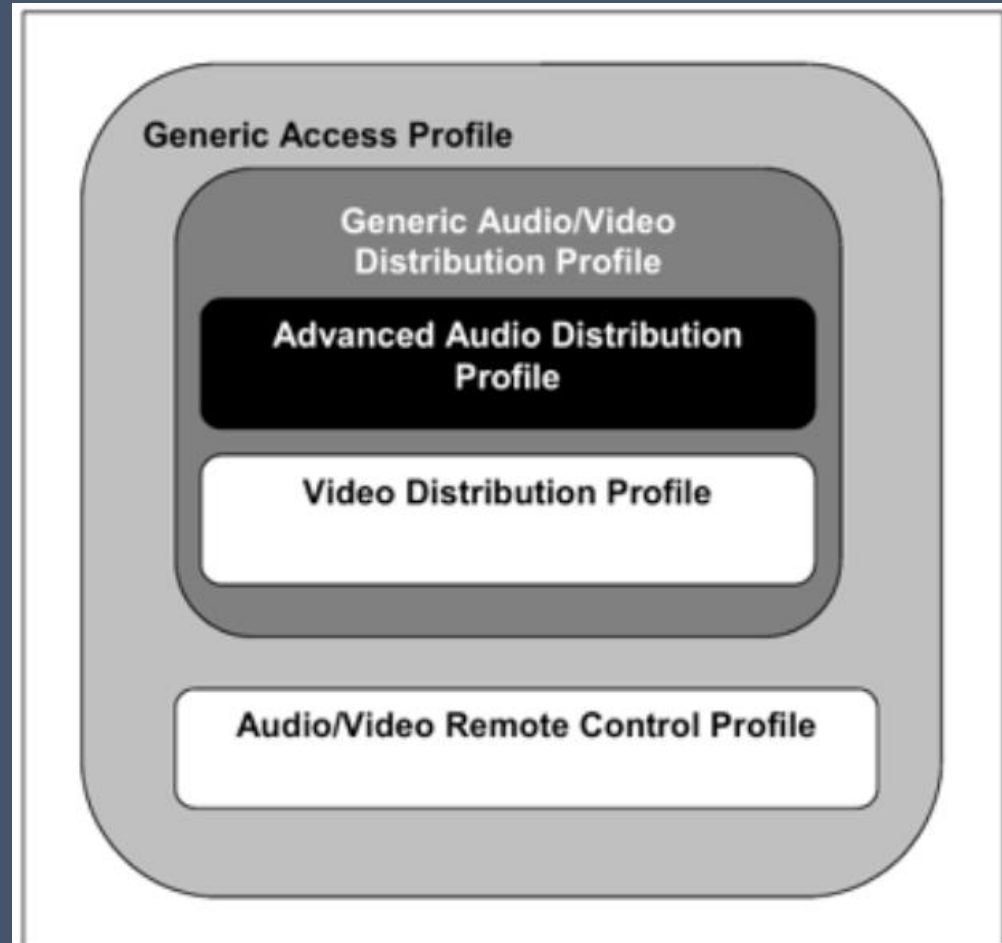


Figure 2-2. Profile Dependencies

그림 2-2에 표시된 것처럼 A2DP는 GAP 및 GVDP (Generic Audio / Video Distribution Profile)에 따라 달라지며 오디오 / 비디오 스트리밍을 설정하는 데 필요한 절차를 정의합니다.

A2DP에는 소스 (SRC)와 싱크 (SNK)의 두 가지 역할이 정의되어 있습니다. SRC는 디지털 오디오 스트림의 소스로 기능하고 SNK는 SRC에서 전달 된 디지털 오디오 스트림의 싱크로 기능합니다.

AVRCP에 정의 된 두 가지 역할은 컨트롤러 (CT)와 대상 (TG)입니다. CT는 명령 프레임을 대상에 전송하여 트랜잭션을 시작하는 장치입니다. CT의 예로는 개인용 컴퓨터, PDA 및 휴대폰이 있습니다. TG는 커맨드 프레임을 수신하여 응답 프레임을 생성하는 장치이다. 오디오 플레이어 또는 헤드폰이 TG의 예입니다. 당분간 A2DP (SRC) 및 AVRCP (CT)가 지원되며 장치는 라우드 스피커로 작동하여 원격 제어 메시지를 오디오 소스로 보낼 수도 있습니다.

현재 A2DP 솔루션에서 지원되는 유일한 오디오 코덱은 **SBC**이며 A2DP 사양에서 의무화되어 있습니다. A2DP 버전 1.2 및 AVDTP 버전 1.2가 구현되었습니다.

AVDTP (Audio / Video Distribution Transport Protocol)는 스트리밍 설정을위한 Bluetooth 장치와 L2CAP를 사용하는 오디오 및 비디오를 위한 미디어 스트리밍 간의 이진 트랜잭션을 정의합니다. A2DP의 기본 전송 프로토콜 인 AVDTP는 L2CAP 계층을 기반으로하며 스트리밍 매개 변수를 협상하기위한 신호 엔터티와 스트리밍 자체를 처리하는 전송 엔터티로 구성됩니다.

AVDTP 전송 기능의 기본 서비스는 A2DP 사양에 따라 요구됩니다. 현재 서비스 기능의 구성에 따라 기본 서비스 기능의 Media Transport 및 Media Codec이 제공됩니다.

AVRCP는 오디오 / 비디오 원격 제어 사용 사례 지원에 필요한 요구 사항을 정의합니다.

AVRCP에 사용되는 명령은 세 가지 주요 범주로 분류됩니다. 첫 번째는 AV / C 디지털 인터페이스 명령 세트로, 특정 경우에만 적용되며 AVCTP (Audio / Video Control Transport Protocol)로 전송됩니다. 탐색 명령은 두 번째 범주에 포함되어 있으며 AVCTP 탐색 채널이라는 다른 전송 채널을 통해 탐색 기능을 제공합니다. 세 번째 범주 인 표시 아트 명령은 미디어 항목과 관련된 이미지를 전송하는 데 사용되며 제공됩니다.

OBEX 프로토콜과 함께 BIP (Bluetooth Basic Imaging Profile)에 정의 된 프로토콜을 통해

AVRCP에는 두 세트의 AV / C 명령이 사용됩니다. 첫 번째 것은 PASS THROUGH 명령, UNIT INFO 명령 및 SUBUNIT INFO 명령을 포함하며 AV / C 사양에 정의되어 있습니다. 두 번째 세트에는 Bluetooth SIG 공급 업체 종속 확장으로 정의 된 AVRCP 특정 AV / C 명령이 포함됩니다. AV / C 명령은 AVCTP 제어 채널을 통해 전송됩니다. PASS THROUGH 명령은 버튼을 통해 사용자 조작을 컨트롤러에서 패널 서브 유닛으로 전송하는 데 사용되며, 이는 대상을 제어하는 단순하고 일반적인 메커니즘을 제공합니다. 예를 들어, PASS THROUGH의 작업 ID에는 Play, Pause, Stop, Volume Up 및 Volume down과 같은 일반적인 지침이 포함됩니다.

AVRCP는 A / V 기능을 4 가지 범주로 정렬하여 상호 운용성을 보장합니다.

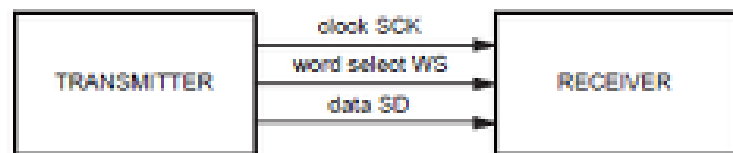
- 플레이어 / 레코더
- 모니터 / 앰프
- 튜너
- 메뉴

현재 구현에서는 AVRCP 버전 1.3 및 AVCTP 버전 1.4가 제공됩니다. AVRCP 지원 기능의 기본 구성은 카테고리 2 : 모니터 / 앰프입니다. 또한 PASS THROUGH 명령을 보내기 위한 API가 제공됩니다.

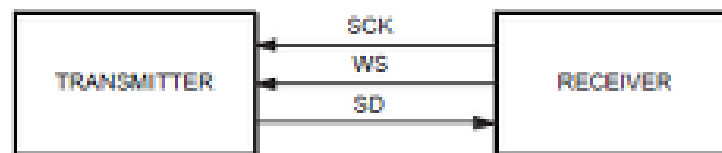
A2DP와 AVRCP는 종종 함께 사용됩니다. 현재 솔루션에서 하위 호스트 스택은 AVDTP 및 AVCTP 논리를 구현하면서 A2DP 및 AVRCP에 대한 인터페이스를 독립적으로 제공합니다. 그러나 스택의 상위 계층에서는 두 프로파일이 결합되어 구성됩니다.

"AV"모듈. 예를 들어 BTA 계층은 통합 된 "AV"인터페이스를 제공하며 BTC 계층에는 두 프로파일의 이벤트를 처리하는 상태 시스템이 있습니다. 그러나 API는 A2DP 및 AVRCP에 대해 별도로 제공됩니다.

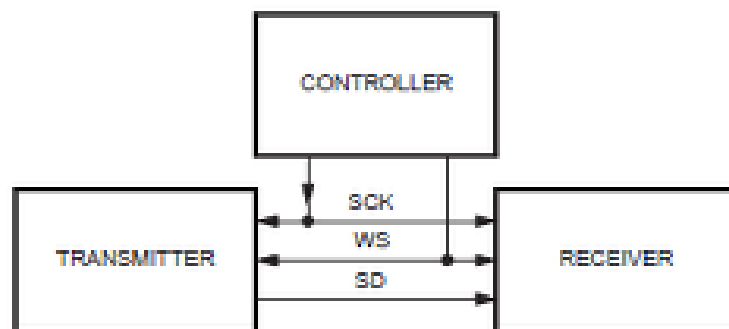
I2S (Intergrated Interchip Sound)



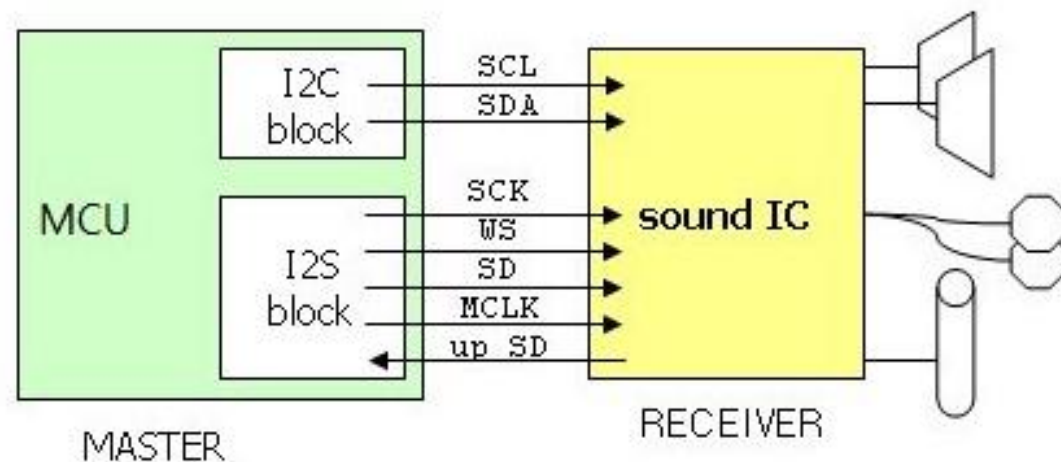
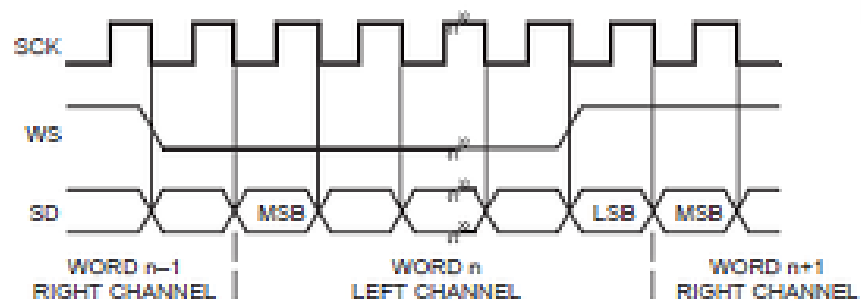
TRANSMITTER = MASTER



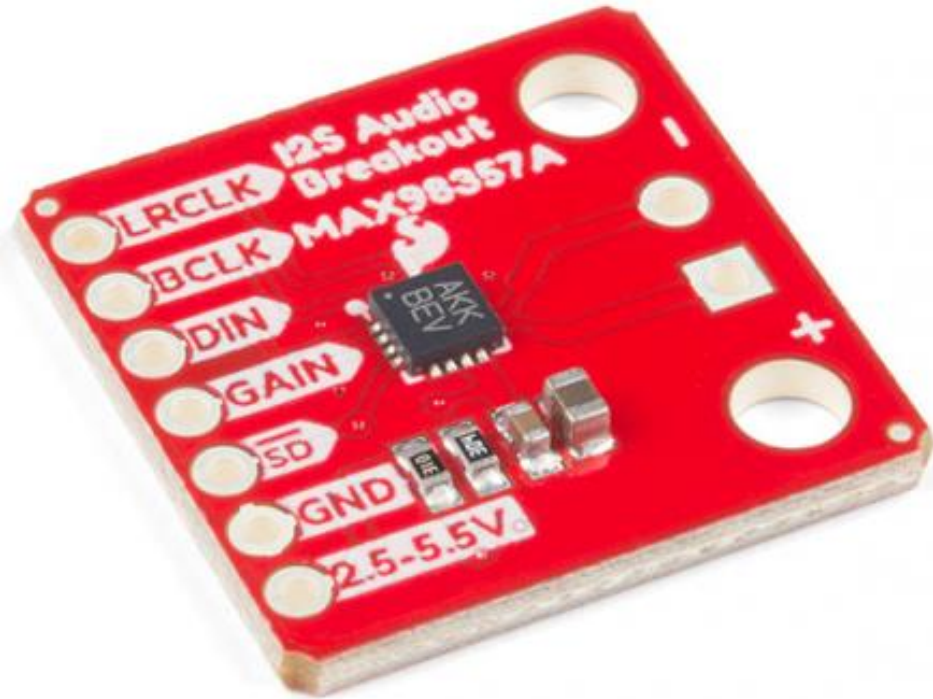
RECEIVER = MASTER



CONTROLLER = MASTER



SN00119



SparkFun I2S Audio Breakout - MAX98357A

DEV-14809

\$5.50

<https://learn.sparkfun.com/tutorials/i2s-audio-breakout-hookup-guide/all>


```

55     i2s_config_t i2s_config = {
56 #ifdef CONFIG_EXAMPLE_A2DP_SINK_OUTPUT_INTERNAL_DAC
57         .mode = I2S_MODE_MASTER | I2S_MODE_TX | I2S_MODE_DAC_BUILT_IN,
58 #else
59         .mode = I2S_MODE_MASTER | I2S_MODE_TX,           // Only TX
60 #endif
61         .sample_rate = 44100,
62         .bits_per_sample = 16,
63         .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,    //2-channels
64         .communication_format = I2S_COMM_FORMAT_I2S_MSB,
65         .dma_buf_count = 6,
66         .dma_buf_len = 60,
67         .intr_alloc_flags = 0,                          //Default interrupt priority
68         .tx_desc_auto_clear = true                       //Auto clear tx descriptor on underflow
69     };
70
71
72     i2s_driver_install(0, &i2s_config, 0, NULL);
73 #ifdef CONFIG_EXAMPLE_A2DP_SINK_OUTPUT_INTERNAL_DAC
74     i2s_set_dac_mode(I2S_DAC_CHANNEL_BOTH_EN);
75     i2s_set_pin(0, NULL);
76 #else
77     i2s_pin_config_t pin_config = {
78         .bck_io_num = CONFIG_EXAMPLE_I2S_BCK_PIN,
79         .ws_io_num = CONFIG_EXAMPLE_I2S_LRCK_PIN,
80         .data_out_num = CONFIG_EXAMPLE_I2S_DATA_PIN,
81         .data_in_num = -1                                //Not used
82     };
83
84     i2s_set_pin(0, &pin_config);
85 #endif

```


A2DP Example Configuration

- [?] Arrow keys navigate the menu. <Enter> selects submenus ---> (or
- [?] Highlighted letters are hotkeys. Pressing <Y> includes, <N> exc
- [?] features. Press <Esc><Esc> to exit, <?> for Help, </> for Search
- [?] excluded <M> module < > module capable

[illegible]

A2DP Sink Output (External I2S Codec) --->

(22) 2S LRCK (WS) GPIO


(26) | 2S BCK GPIO

(25) | 2S DATA GPIO

?

?

?



QUESTION

QUESTION

6th Grade

```
304 static void volume_change_simulation(void *arg)
305 {
306     ESP_LOGI(BT_RC_TG_TAG, "start volume change simulation");
307
308     for (;;) {
309         vTaskDelay(10000 / portTICK_RATE_MS);
310
311         uint8_t volume = (s_volume + 5) & 0x7f;
312         volume_set_by_local_host(volume);
313     }
314 }
315
316 static void bt_av_hdl_avrc_tg_evt(uint16_t event, void *p_param)
317 {
318     ESP_LOGD(BT_RC_TG_TAG, "%s evt %d", __func__, event);
319     esp_avrc_tg_cb_param_t *rc = (esp_avrc_tg_cb_param_t *) (p_param);
320     switch (event) {
321     case ESP_AVRC_TG_CONNECTION_STATE_EVT: {
322         uint8_t *bda = rc->conn_stat.remote_bda;
323         ESP_LOGI(BT_RC_TG_TAG, "AVRC conn_state evt: state %d, [%02x:%02x:%02x:%02x:%02x:%02x]",
324                 rc->conn_stat.connected, bda[0], bda[1], bda[2], bda[3], bda[4], bda[5]);
325         if (rc->conn_stat.connected) {
326             // create task to simulate volume change
327             xTaskCreate(volume_change_simulation, "vcsT", 2048, NULL, 5, &s_vcs_task_hdl);
328         } else {
329             vTaskDelete(s_vcs_task_hdl);
330             ESP_LOGI(BT_RC_TG_TAG, "Stop volume change simulation");
331         }
332         break;
333     }
```



» API Reference » Bluetooth API » CLASSIC BT » SPP API

SPP API

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp_spp.html

Application Example

Check [bluetooth/bluedroid/classic_bt](#) folder in ESP-IDF examples, which contains the following application:

- This is a SPP demo. This demo can discover the service, connect, send and receive SPP data
[bluetooth/bluedroid/classic_bt/bt_spp_acceptor](#),
[bluetooth/bluedroid/classic_bt/bt_spp_initiator](#)

Functions

```
esp_err_t esp_spp_register_callback(esp_spp_cb_t callback)
```

This function is called to init callbacks with SPP module.

Return

- ESP_OK: success
- other: failed

Parameters

- `[in] callback`: pointer to the init callback function.

```
esp_err_t esp_spp_init(esp_spp_mode_t mode)
```

This function is called to init SPP.

Return

- ESP_OK: success
- other: failed

Parameters

- `[in] mode`: Choose the mode of SPP, ESP_SPP_MODE_CB or ESP_SPP_MODE_VFS.

```
esp_err_t esp_spp_start_srv(esp_spp_sec_t sec_mask, esp_spp_role_t role, uint8_t local_scn, const char *name)
```

This function create a SPP server and starts listening for an SPP connection request from a remote Bluetooth device. When the server is started successfully, the callback is called with ESP_SPP_START_EVT. When the connection is established, the callback is called with ESP_SPP_SRV_OPEN_EVT.

Return

- ESP_OK: success
- other: failed

Parameters

- **[in] sec_mask**: Security Setting Mask. Suggest to use ESP_SPP_SEC_NONE, ESP_SPP_SEC_AUTHORIZE or ESP_SPP_SEC_AUTHENTICATE only.
- **[in] role**: Master or slave.
- **[in] local_scn**: The specific channel you want to get. If channel is 0, means get any channel.
- **[in] name**: Server's name.

이 기능은 SPP 서버를 생성하고 원격 Bluetooth 장치에서 SPP 연결 요청을 수신하기 시작합니다.

서버가 성공적으로 시작되면 콜백은 **ESP_SPP_START_EVT**와 함께 호출됩니다.

연결이 설정되면 콜백은 **ESP_SPP_SRV_OPEN_EVT**로 호출됩니다.

```
struct spp_data_ind_evt_param
```

```
#include <esp_spp_api.h>  
ESP_SPP_DATA_IND_EVT.
```

Public Members

```
esp_spp_status_t status
```

status

```
uint32_t handle
```

The connection handle

```
uint16_t len
```

The length of data

```
uint8_t *data
```

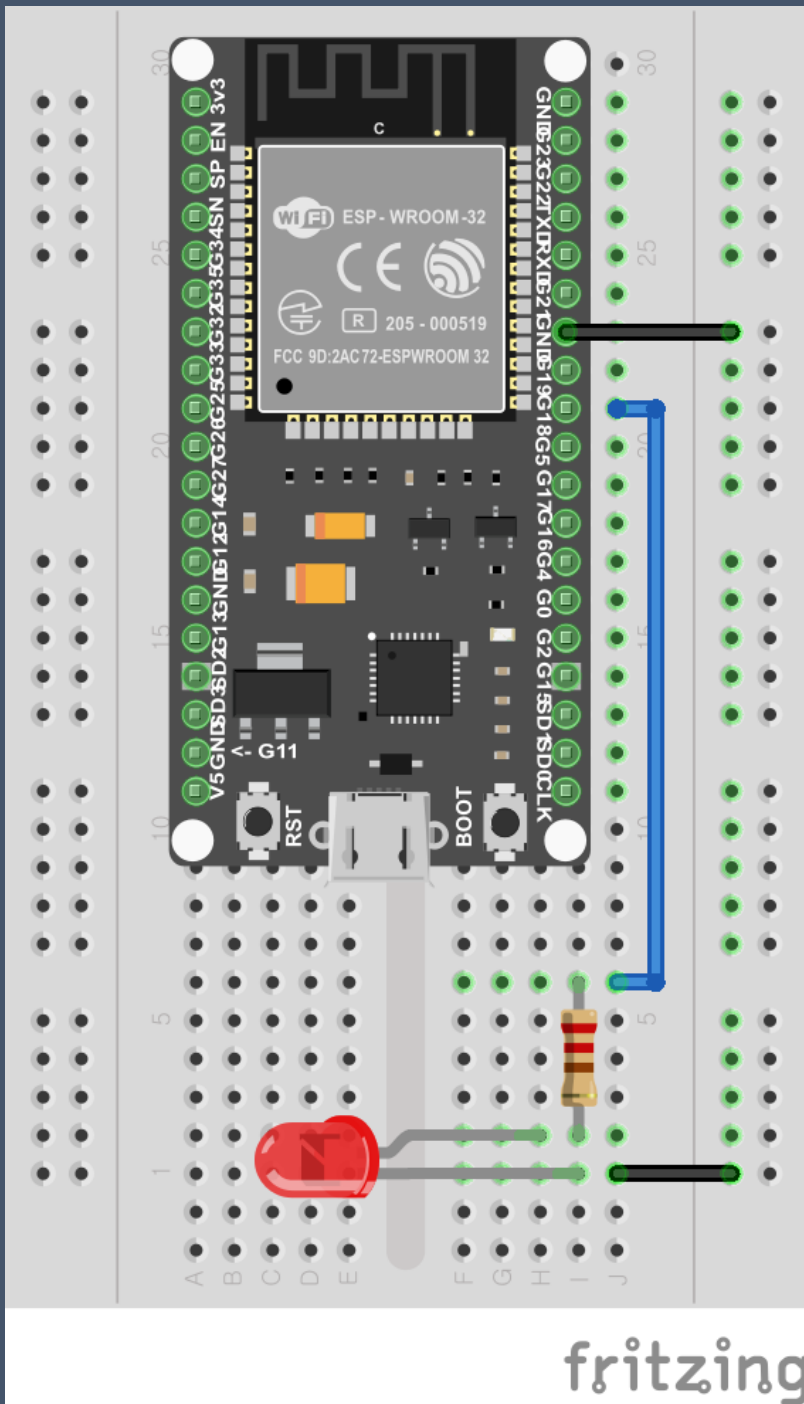
The data received

```
struct esp_spp_cb_param_t::spp_data_ind_evt_param data_ind
```

SPP callback param of ESP_SPP_DATA_IND_EVT

(base example)

esp-idf/examples/bluetooth/bluedroid/classic_bt/bt_spp_acceptor/main
example_spp_acceptor_demo.c



ESP32 18번 핀 --- 저항 연결 --- LED 연결

[esp-idf/examples/peripherals/gpio/main/gpio_example_main.c](https://github.com/espressystem/examples/blob/master/peripherals/gpio/main/gpio_example_main.c)

```
esp_err_t gpio_config(const gpio_config_t *pGPIOConfig)
```

GPIO common configuration.

Configure GPIO's Mode,pull-up,PullDown,IntrType

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pGPIOConfig`: Pointer to GPIO configure struct

```
/** espressif / esp-iot-solution
```

```
 * @brief Configuration parameters of GPIO pad for gpio_config function
```

```
 */
```

```
typedef struct {  
    uint64_t pin_bit_mask; /*!< GPIO pin: set with bit mask, each bit maps to a GPIO */  
    gpio_mode_t mode; /*!< GPIO mode: set input/output mode */  
    gpio_pullup_t pull_up_en; /*!< GPIO pull-up */  
    gpio_pulldown_t pull_down_en; /*!< GPIO pull-down */  
    gpio_int_type_t intr_type; /*!< GPIO interrupt type */  
} gpio_config_t;
```

```
esp_err_t gpio_set_level(gpio_num_t gpio_num, uint32_t level)
```

GPIO set output level.

Return

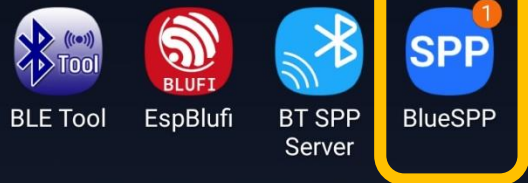
- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO number error

Parameters

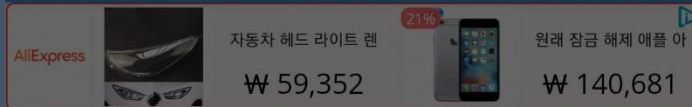
- `gpio_num`: GPIO number. If you want to set the output level of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- `level`: Output level. 0: low ; 1: high

```
/**  
 * @brief GPIO set output level  
 * <> Code  
 * @param hw Peripheral GPIO hardware instance address.  
 * @param gpio_num GPIO number. If you want to set the output level of e.g. GPIO16, gpio_num  
 * @param level Output level. 0: low ; 1: high  
 */  
static inline void gpio_ll_set_level(gpio_dev_t *hw, gpio_num_t gpio_num, uint32_t level)  
{  
    if (level) {  
        if (gpio_num < 32) {  
            hw->out_w1ts = (1 << gpio_num);  
        } else {  
            hw->out1_w1ts.data = (1 << (gpio_num - 32));  
        }  
    } else {  
        if (gpio_num < 32) {  
            hw->out_w1tc = (1 << gpio_num);  
        } else {  
            hw->out1_w1tc.data = (1 << (gpio_num - 32));  
        }  
    }  
}
```

파인더 검색



ESP_SPP_ACCEPTOR DISCONNECT



TERMINAL KEYBOARD SWITCH

Button Editor

Button Text

LED

Status Press

☒ Character ☐ HEX

Message

ledtoggle

Status Release

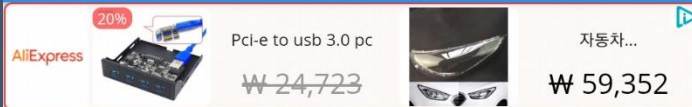
☒ Character ☐ HEX

Message

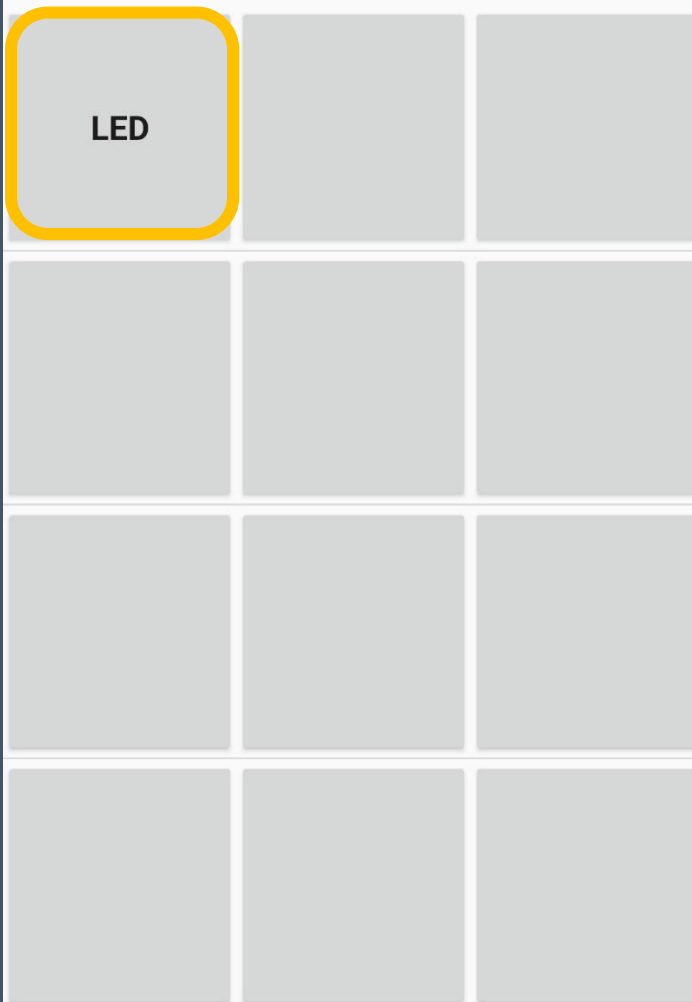
취소 확인

Edit Mode ☒

ESP_SPP_ACCEPTOR DISCONNECT



TERMINAL KEYBOARD SWITCH



Edit Mode ☐

감사합니다.