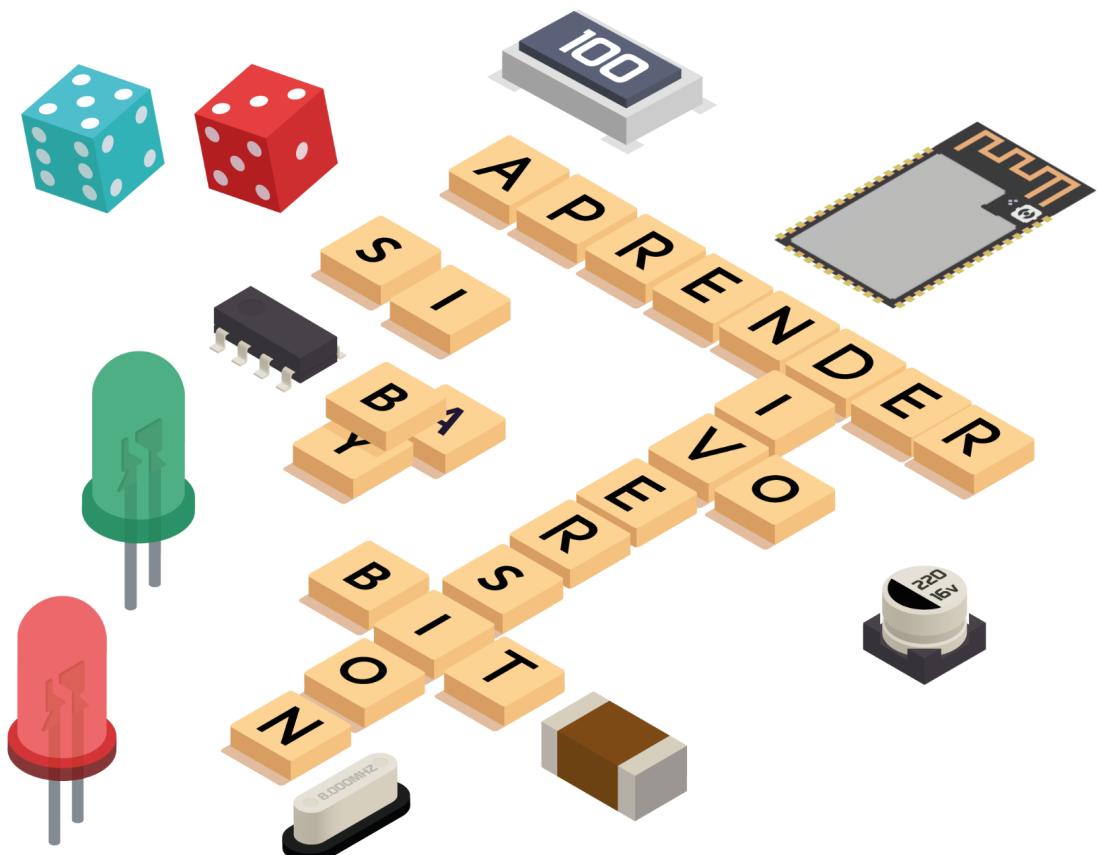


KIT: BIT one

1.0

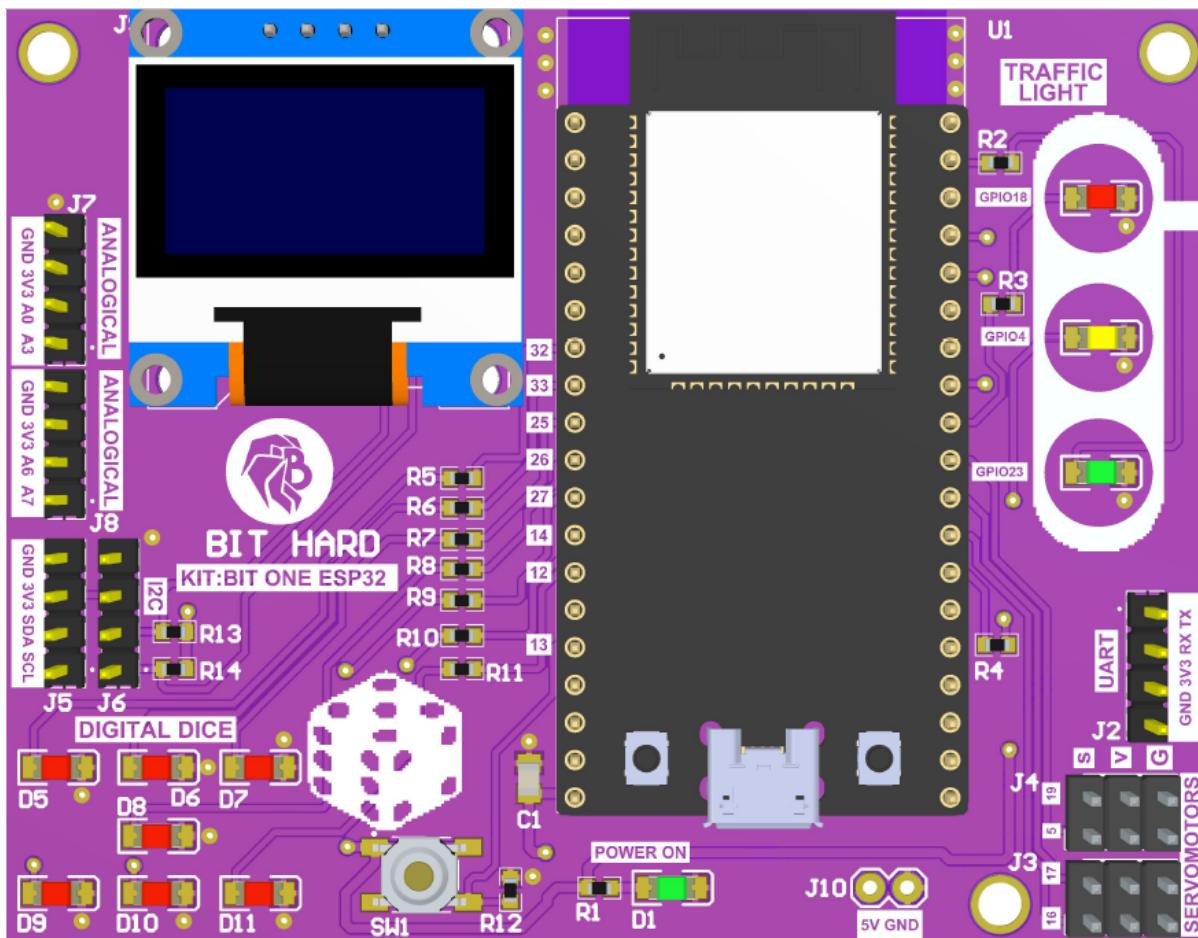
ESP32

VerSión



Design by:

 **BIT-HARD**
TECHNOLOGY



Kit Bit One ESP32

Copyright © Bit Hard SpA., 2022

Contáctanos

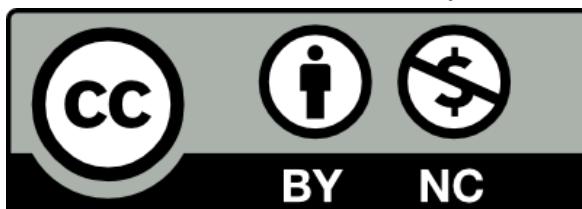
Bit Hard SpA

Santiago, Chile

Email: contacto@bit-hard.cl

www.bit-hard.cl

Atribución-NoComercial 4.0 Internacional (CC BY-NC 4.0)



Toda la información contenida en este manual cuenta con Licencia Creative Commons.

Índice

04

Introducción

05

Lista de componentes
electrónicos del kit.

06

Kit Bit One ESP32

09

Diagrama esquemático

11

Conceptos básicos

20

Inicio del Bit One ESP32.

25

Lección 1: Enciende un
led

28

Lección 2: Semáforo

30

Lección 3: Dado digital

32

Lección 4: Fade IN Fade
OUT

33

Lección 5: Uso del
sensor BMP280

36

Lección 6: Uso del
display Oled

38

Bibliografía

Introducción

Toda persona que quiera aprender electrónica puede hacerlo; y más en el siglo XXI; donde al menos un 80% de los objetos que nos rodean tienen un circuito eléctrico en su interior. Hoy podemos acceder fácilmente a recursos que nos permiten validar ideas y generar prototipos en cuestión de pocos días o incluso horas.

Enfocados en la posibilidad de tener a nuestro alcance una herramienta que nos permita desarrollar competencias de electrónica y prototipado rápido fue como surgió el **Kit Bit One ESP32**.

El **Kit Bit One ESP32** está compuesto por un original shield (escudo) para utilizar el fantástico **ESP32** (DevKitC); un sensor de humedad, presión y temperatura (**BMP280**); un display Oled de 0.96 pulgadas y varios cables dupont hembra-hembra para conectar diversos sensores.

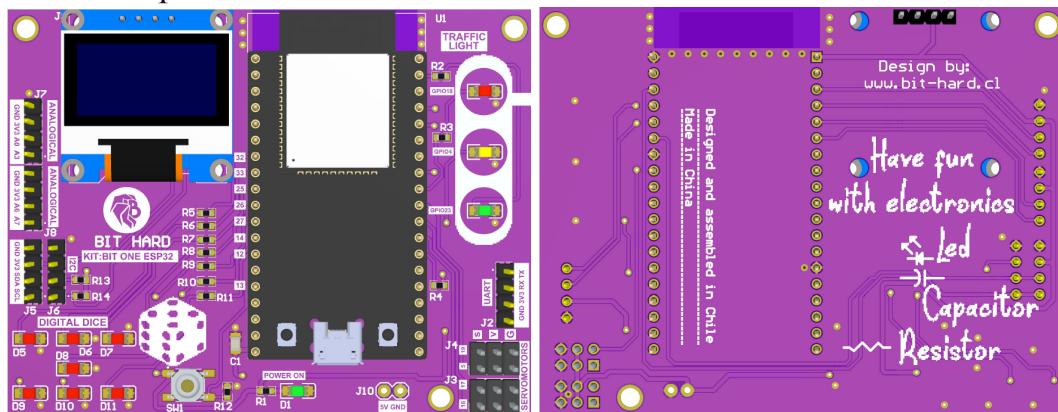


Imagen 1. PCB del kit Bit One ESP32.

El shield **Bit One ESP32** viene con 10 led programables para ser utilizados en la implementación de un semáforo y un dado digital; o pueden ser usados de acuerdo a las necesidades del usuario. Como uno de los ejemplos más cotidianos en las clases iniciales de programación es implementar un semáforo optamos por agregarlo al shield.

Con nuestro kit podrás aprender electrónica y programación básica al mismo tiempo. Utilizando algunos códigos de programación podrás validar las configuraciones básicas de diferentes componentes electrónicos. Tan sencillo como conectar un sensor o un servomotor a la placa, posteriormente cargar tu código a través de un PC; y ver en funcionamiento tu idea.

Ahora que has adquirido una nueva herramienta tendrás la oportunidad de validar tus ideas para desarrollar prototipos electrónicos. Crea, valida y experimenta con **Bit One ESP32**.

Disfrutemos aprendiendo electrónica juntos.

Lista de componentes electrónicos del Kit

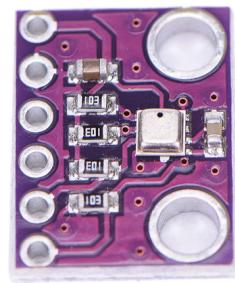
ESP32 DevKitC
x1



*Pantalla Oled 0.96
pulgadas, 128x64, I2C*
x1



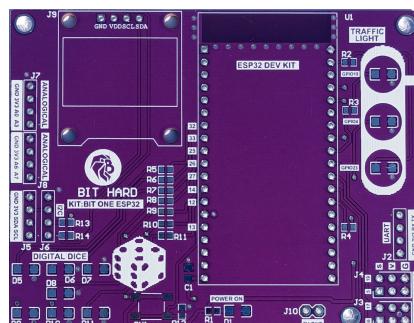
*Sensor BMP 280 (humedad,
presión y temperatura)*
x1

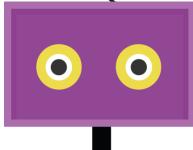


*Cable dupont
hembra-hembra*
x10



*PCB Bit One ESP32
(soldermask: morado,
surface finish: LeadFree
HASL - RoHS)*
x1





KIT BIT One ESP32

El kit Bit One *ESP32* es un original shield para aprender a utilizar el fantástico microcontrolador *ESP32*. Un shield es una tarjeta electrónica que proporciona nuevas capacidades y utilidades para las placas de desarrollo.

Con las configuraciones eléctricas utilizadas en Bit One *ESP32* expandimos las funcionalidades de la tarjeta de desarrollo *ESP32 DevKitC*. Dispondrás de un entorno de desarrollo modular con conectividad Wi-Fi y Bluetooth, compatible con el ambiente de programación y librerías de *Arduino IDE*.

Características

- Voltaje de alimentación: 5V DC.
- Representación de un dado digital a través de 7 leds y un pulsador.
- Representación de un semáforo a través de 3 leds.
- 4 puertos analógicos (pueden ser configurados como GPIO digitales también).
- 2 puertos de comunicación I2C.
- Un puerto de comunicación UART.
- 4 puertos GPIO para conectar servomotores.
- Una pantalla Oled de 0.96 pulgadas y 128x64 caracteres.
- Dimensión: 64x82 (mm).
- Diámetro de los orificios de fijación: 3mm (x3).

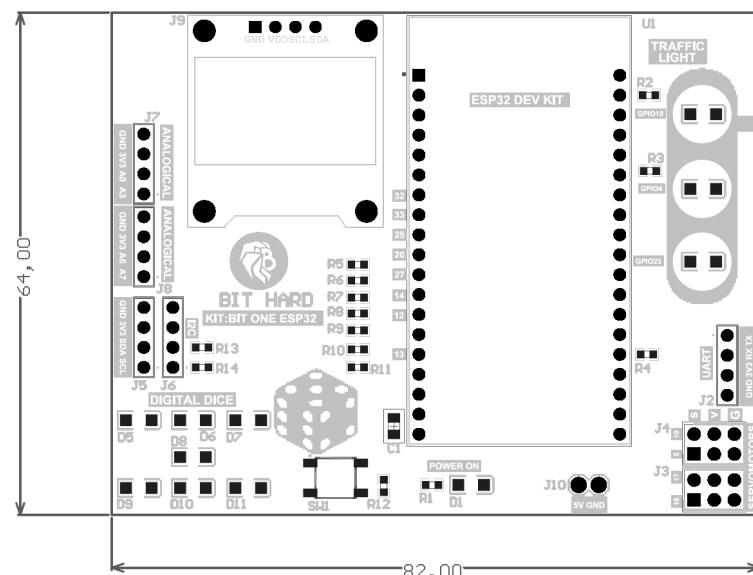


Imagen 2. Dimensiones físicas del PCB Bit One *ESP32*.

Descripción del hardware

El shield **Bit One ESP32** contiene los elementos básicos para permitir la iniciación de cualquier persona con interés en aprender electrónica. La programación del **ESP32** se puede realizar con el software **Arduino (IDE)**.

La placa **Bit One ESP32** se alimenta con **5V (DC)**. Tenemos dos formas de suministrar el voltaje de alimentación, la primera es a través del conector micro **USB** en el **ESP32 DevKitC**; y la segunda es a través del pin header **INPUT**, identificado en el **PCB** como **5V GND**. La placa no incluye el transformador de **5V**, pero puedes utilizar cualquier adaptador de **5V** con corriente de carga mayor a **1.5 Amp**, prácticamente todos los cargadores de teléfonos móviles son de **5V**, cualquiera de ellos funcionara; así evitas adquirir un nuevo adaptador de voltaje. En el caso de que dispongas de una fuente de alimentación **DC** solo suministra **5V** en el pin header específico y ajusta la corriente mayor a **1.5 Amp**.

Entre las funciones que agregan novedad y originalidad a la placa tenemos el dado digital y el semáforo. Utiliza los 7 led rojos para simular cualquiera de los dígitos en un dado, con cada toque en el pulsador se genera un número aleatorio; para activar esta función debes cargar el código de ejemplo: dado digital.

En la esquina superior derecha del **PCB** encontrarás tres leds identificados como traffic light, utiliza el código de ejemplo y podrás visualizar el funcionamiento clásico de un semáforo; realiza ajustes en los tiempos del encendido y visualiza la respuesta en los leds. Los leds utilizados para el dado y el semáforo pueden ser configurados como indicadores de salida, estos se encuentran conectados a varios **GPIO** y se adaptan al uso que necesites.

Tenemos 4 puertos **GPIO** para conectar servomotores, como cualquier servo viene con un cable de tres pines (señal, vcc y gnd) colocamos 2 pin header (2x3) para conectar los servomotores sin tener que utilizar múltiples cables.

Hay gran cantidad de proyectos que utilizan pantallas tipo Oled y pocos shield vienen con una integrada, nuestra placa utiliza un display Oled con comunicación **I2C**; podrás generar mensajes, configurar un medidor de temperatura o incluso implementar una estación meteorológica hecha a tu medida, todo dependerá de los sensores que incorpores, pero las variables a sensar podrás visualizarla en tu display.

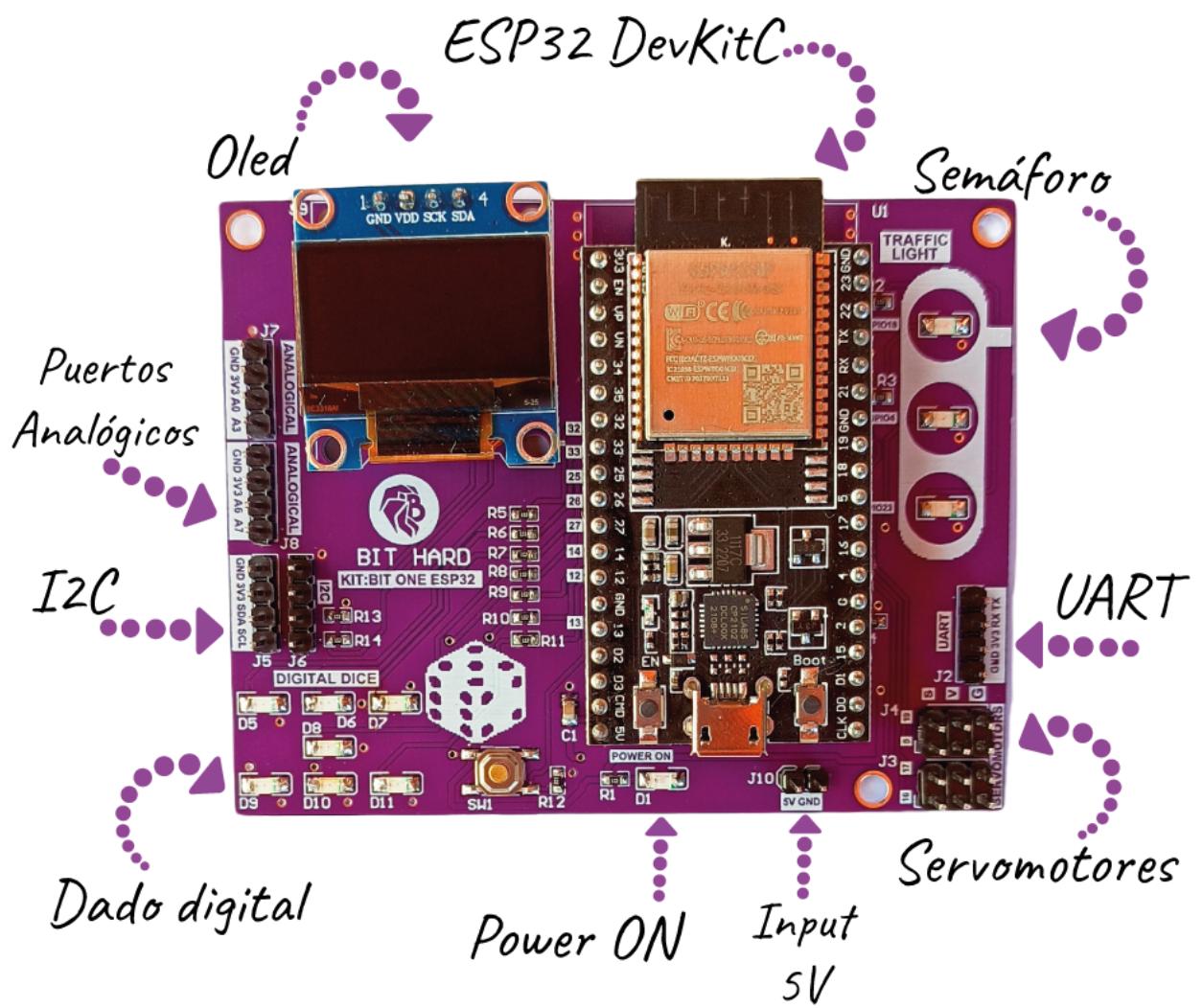


Imagen 3. Kit Bit One ESP32.

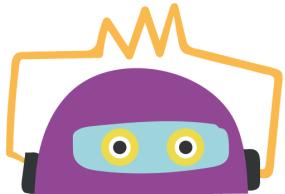


DIAGRAMA ESQUEMÁTICO

Un diagrama esquemático es la representación gráfica de un circuito electrónico o eléctrico. A través de diversos símbolos, líneas y otros elementos gráficos se representan las conexiones eléctricas que permiten el funcionamiento del circuito a implementar.

Un circuito eléctrico es un conjunto de elementos conectados entre sí por los que circula la corriente eléctrica. Tienen uno o más caminos cerrados por los cuales fluye la corriente eléctrica.

El diagrama esquemático de la placa electrónica Bit One ESP32 lo vemos a continuación.

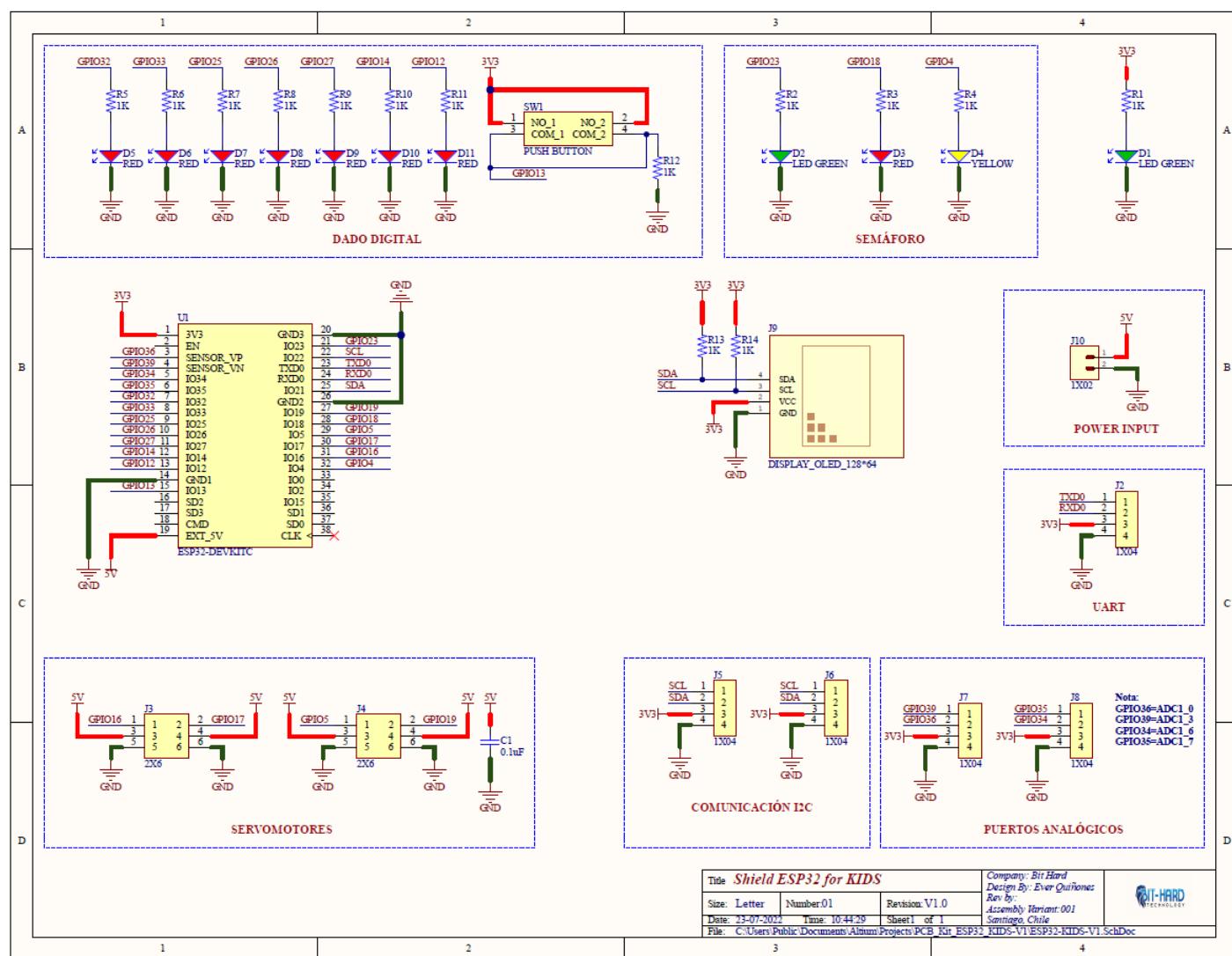


Imagen 4. Diagrama esquemático de la placa Bit One ESP32.

En el diseño de circuitos eléctricos se utiliza la letra **R** para representar a las resistencias, la referencia **D** corresponde a los diodos led utilizados. Con la **C** designamos a los condensadores y con la letra **J** hacemos referencia a los conectores que estén presentes en un circuito.

El **ESP32** es el cerebro de nuestra placa, todos los códigos e instrucciones serán cargados y almacenados aquí.

GPIO = Pin entrada/salida de propósito general.

Los pines **GPIO** pueden funcionar como entradas o salidas. En el caso del puerto analógico tenemos los **GPIO**: 36, 39, 34 y 35 que funcionan como entradas, aquí conectaremos hasta 4 sensores analógicos. La resolución del **ADC** del **ESP32** es de 12 bits. Si no queremos usar estos pines como analógicos podemos configurarlos como entradas/salidas digitales.

El kit viene con el sensor **BMP 280**, con él podrás sentir las variables físicas: temperatura, humedad y presión al mismo tiempo. Este sensor tiene un protocolo de comunicación **I2C**, quiere decir que a través de los pines **SDA** y **SCL** enviará la información capturada. La placa tiene conexión para dos I2C, lo que nos da la posibilidad de conectar dos sensores **I2C**.

El apartado servomotores muestra donde se conectarán los mismos. Todo servo viene con un cable de tres terminales (señal, vcc y gnd), este será conectado en cualquiera de los conectores pin header establecidos para ello.

Si queremos utilizar un sensor que tenga comunicación **UART** lo haremos en el pin header establecido para ello.

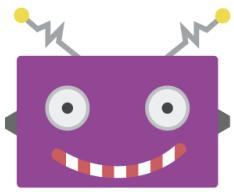
La representación del dato digital se conforma por siete leds de color rojo y un pulsador normalmente abierto. Se debe cargar el código dado digital para poder habilitar esta función. Cuando el pulsador es apretado el dato mostrará un número, con cada toque este número irá cambiando aleatoriamente.

El semáforo es representado con tres leds de color amarillo, rojo y verde. Se debe cargar el código semáforo para poder habilitar esta función. El tiempo del encendido de cada led puede ser modificado en el código.

La pantalla Oled se comunica vía **I2C**, aquí podemos ser creativos y probar cuantos códigos desarrollemos, todo dependerá de nuestra imaginación.

En el caso de tener una fuente de alimentación alimentaremos la placa a través de los pines **5V** y **GND**.

Este diagrama esquemático es bastante simple de comprender, más adelante veremos el símbolo de cada componente y con ello ya podrás comenzar a leer diagramas esquemáticos de nivel básico.



CONCEPTOS BÁSICOS

Carga Eléctrica.

La carga eléctrica es una de las características fundamentales de la materia, al igual que la masa. Se utiliza para justificar las fuerzas eléctricas que se observan experimentalmente. Dado que se observan dos tipos de fuerzas, de atracción y de repulsión, se considera que hay dos tipos de carga, cargas positivas y cargas negativas [1].

Podemos considerar a la carga eléctrica como la esencia de la electricidad, y aunque es un concepto un poco complejo para describir, podemos ver a la carga como las partículas constituyentes del átomo.

Corriente Eléctrica.

La intensidad de la corriente es la cantidad de cargas eléctricas que atraviesan una sección transversal de un conductor en la unidad de tiempo [1]. La corriente eléctrica existe en cualquier conductor eléctrico, el conductor más conocido y utilizado es el cobre y es el núcleo de los cables; en nuestro cuerpo también hay circulación de la corriente eléctrica, pero en la madera o en una hoja de papel no hay presencia de corriente eléctrica.

La corriente eléctrica se representa con la letra **I**, y su unidad de medida es el Amperio (**A**). En gran parte de circuitos eléctricos y en el caso de nuestro kit la corriente eléctrica se encuentra en el orden de los miliamperios. Un miliamperio es la milésima parte de un amperio.

$$1\text{Amper} = 1000\text{mA}$$

Ecuación 1.

Diferencia de Potencial: tensión eléctrica.

Para que las cargas eléctricas se muevan a través de los conductores, necesitan un “empujoncito”: la diferencia de potencial o la tensión eléctrica. Las cargas sólo se moverán cuando haya una diferencia de potencial; es decir, para que por un circuito pase corriente eléctrica, es totalmente necesario que existan diferencias de potencial entre los diferentes puntos del circuito [1].

Con la letra **V** representamos a la tensión eléctrica y su unidad de medida es el voltio.

Ley de Ohm

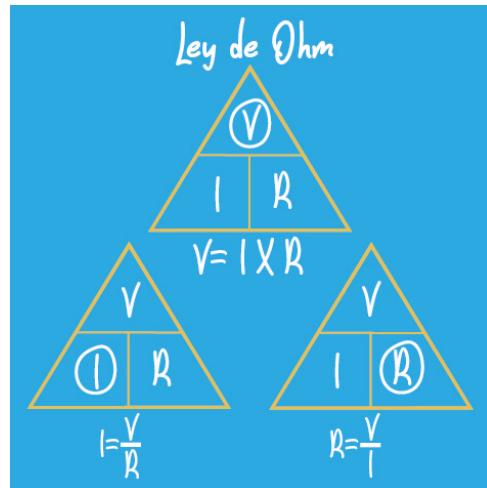


Imagen 5. Triángulo de la ley de Ohm.

Georg Simón Ohm en 1827 desarrolló una de las leyes más importantes en el análisis de circuitos eléctricos: la ley de Ohm.

Esta ley revela claramente qué para una resistencia fija, a mayor voltaje (o presión) en un resistor, mayor es la corriente, y a mayor resistencia para el mismo voltaje, menor es la corriente. En otras palabras, la corriente es proporcional al voltaje aplicado e inversamente proporcional a la resistencia [2].

$$I = \frac{V}{R} \quad (\text{amper, A})$$

Ecuación 2.

Aplicando manejos matemáticos se pueden conocer las otras dos variables, las cuales se pueden encontrar a través de las ecuaciones 3 y 4.

$$V = I \times R \quad (\text{volts, V})$$

Ecuación 3.

$$R = \frac{V}{I} \quad (\text{ohms, } \Omega)$$

Ecuación 4.

Una forma sencilla de recordar la ley de Ohm es a través del uso de un triángulo, se asigna cada variable a sus cuadrantes y cuando se tapa algunos de los cuadrantes queda expresada la relación matemática que se debe utilizar. En la imagen 5 se puede ver el triángulo de Ohm. Entonces podemos estar de acuerdo que la ley de Ohm es un principio matemático que necesita del uso de una fórmula, en electricidad o electrónica esta será la relación matemática más utilizada, es como el oxígeno que una persona necesita para poder respirar.

Resistencia eléctrica.

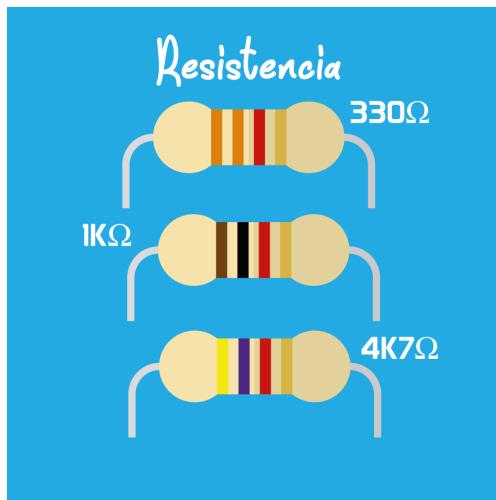


Imagen 6. Diferentes resistencias eléctricas.

El flujo de carga a través de cualquier material encuentra una fuerza opuesta que es similar en muchos aspectos a la fricción mecánica. A esta oposición, debida a las colisiones entre electrones y entre electrones y otros átomos en el material, que convierte la energía eléctrica en otra forma de energía como el calor, se le llama resistencia del material [2].

En materiales eléctricos la resistencia es la oposición a que la corriente eléctrica fluya con tranquilidad y libertad. La resistencia eléctrica es representada con la letra **R** y su unidad de medición es el ohm (Ω)



Imagen 7. Símbolo de la resistencia eléctrica.

En la imagen 7 podemos apreciar la simbología utilizada para representar a la resistencia. En los resistores se emplea un sistema de código de color para poder indicar su valor, esto sucede únicamente en el caso de los componentes tipo **THT** (Through-Hole Technology), y para las resistencias de tipo **SMD** (surface mount device) llevan impreso un código numérico.

En la imagen 8 podemos apreciar el código numérico utilizado en las resistencias **SMD**, este va impreso en el cuerpo de la resistencia eléctrica; en algunos casos se usan 3 dígitos y en otros 4 dígitos. Para el caso de un resistor con 3 dígitos los dos primeros números representan los dígitos y el último número la cantidad de ceros que se deben agregar; en el caso de tener impreso 4 números, los tres primeros son los dígitos y el último número es la cantidad de ceros por agregar.

Código de Resistencias SMD

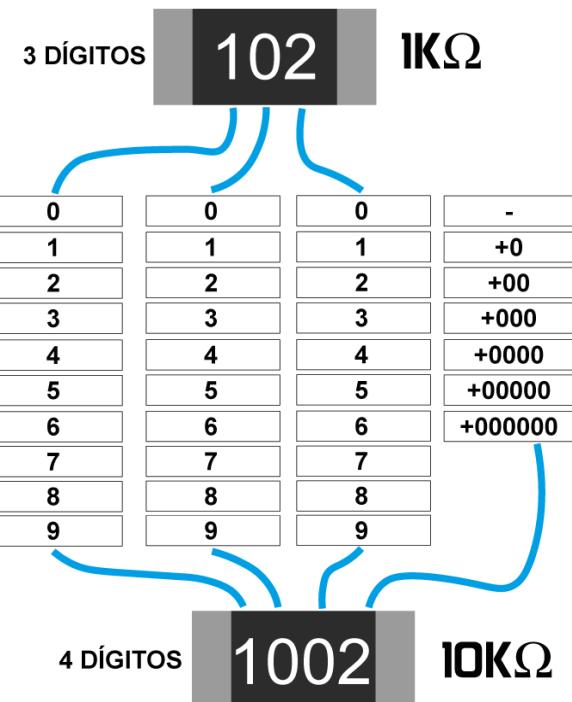


Imagen 8. Código de resistencias SMD.

Otro aspecto en las resistencias SMD es su tamaño, hay varios tipos de encapsulados y los más comunes son los siguientes: **0402**, **0603**, **0805** y **1206**. Estos códigos contienen el ancho y la altura del encapsulado, estas dimensiones están expresadas en pulgadas y corresponden al sistema imperial. En el caso de las medidas expresadas en milímetros el código cambia y se denomina sistema métrico. Lo usual es utilizar las resistencias del código imperial.

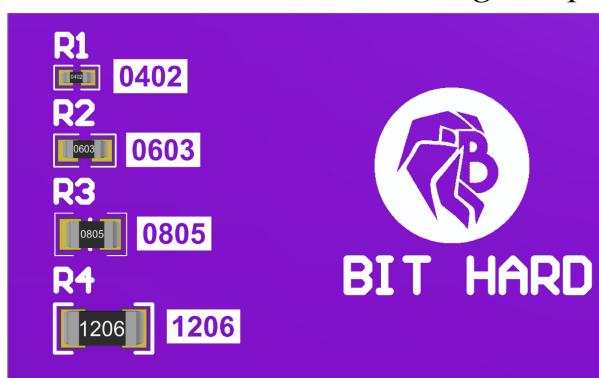


Imagen 9. Diferentes tamaños de encapsulados de resistencias SMD.

En la imagen 9 podemos ver los cuatro tamaños más usuales de utilizar; las resistencias utilizadas en el PCB del kit **Bit One ESP32** son del tamaño **0603**.

Capacitor.

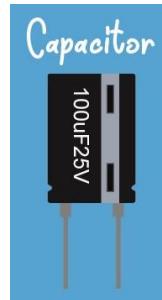


Imagen 10. Capacitor.

Los capacitores son dispositivos de almacenamiento de energía que son esenciales para los circuitos electrónicos tanto analógicos como digitales. El capacitor es un dispositivo electrónico que almacena energía en un campo eléctrico interno [3]. Para no complicarnos con el capacitor debemos verlo como un componente que se dedica a almacenar energía eléctrica; sería como una especie de tanque de agua.

El capacitor o condensador como también se le llama se representa con la letra C, su unidad de medición es el faradio, aunque frecuentemente los condensadores tienen un rango habitual entre picofaradio (pF) y micro faradio (uF). Mientras más grande sea su capacidad tendrán un mayor volumen, por ello no es común utilizar capacitores en el orden de milifaradio (mF) o incluso faradios, ya que ocupan un gran espacio y su costo sería muy elevado. En la imagen 11 y 12 podemos ver el símbolo esquemático para los condensadores polarizados y no polarizados.

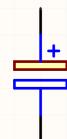


Imagen 11. Capacitor electrolítico.

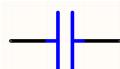


Imagen 12. Capacitor cerámico.

¿Qué es un led?



Imagen 13. Diodos led.

En términos sencillos **LED** es la abreviatura de diodo emisor de luz, probablemente has escuchado este término en algún momento, son muy populares en la actualidad y están presentes en cualquier dispositivo eléctrico, incluso hay ampolletas de bajo consumo eléctrico, cuya luz proviene de varios leds. El led es el típico elemento que ilumina prácticamente todos los aparatos presentes en nuestras vidas.

Como su nombre lo implica, el diodo emisor de luz es un diodo que emite luz visible o invisible (infrarroja) cuando se energiza [3]. Los diodos led corresponden a la categoría de componentes semiconductores, el material del cual están formados tiene la propiedad de conducir corriente eléctrica de acuerdo a varios factores, esto significa que el material puede comportarse como un conductor de corriente eléctrica o como un material aislante.

Para que un diodo led pueda emitir su luz debe polarizarse en directa. En la imagen 14 podemos ver el circuito básico para permitir el encendido de un led, pero debemos considerar el voltaje del led (V_f) que va desde los 1.2V hasta los 4V dependiendo del color.

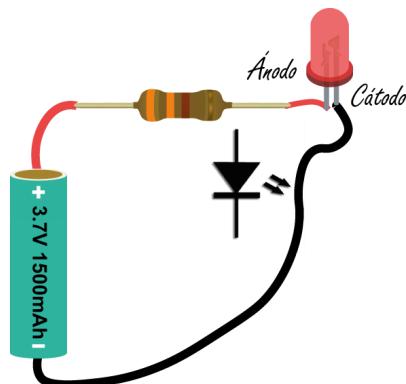


Imagen 14. Circuito básico para energizar un led.

No hay una letra específica que represente los leds, se pueden usar como identificador: D o led; cualquiera de las dos formas es válida. Su símbolo para utilizar en diagramas esquemáticos es el mostrado en la imagen 15.

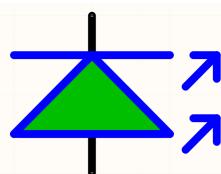


Imagen 15. Símbolo esquemático de un led.

Los leds utilizados en el **PCB** del kit **Bit One ESP32** son de tipo **SMD** y su encapsulado es 1206. En la imagen 16 tenemos la vista en 3D de algunos leds presentes en el PCB.

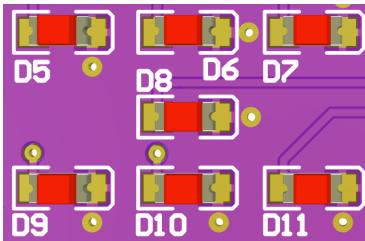


Imagen 16. Diodos led SMD.

¿Qué es el ESP32?

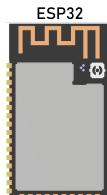


Imagen 17. Modulo ESP32.

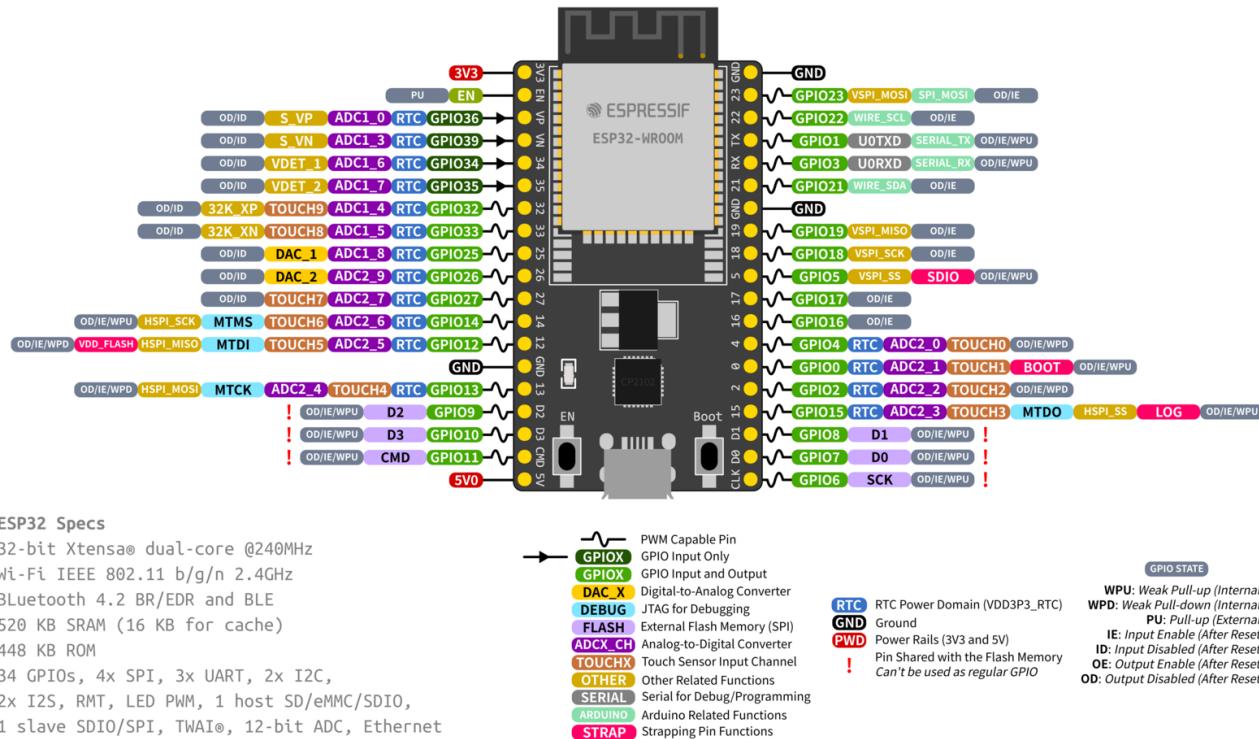
El **ESP32** es un **MCU** (unidad microcontrolador) rico en funciones con **Wi-Fi** integrado y conectividad Bluetooth para una amplia gama de aplicaciones [4]. Un microcontrolador es un circuito integrado que puede realizar varias funciones gracias a su naturaleza programable que permite configurar en su interior diferentes usos.

El **ESP32** es un microcontrolador diseñado por la compañía Espressif Systems, actualmente es muy utilizado en proyectos por su reducido precio y las prestaciones en su interior. La imagen 17 muestra el modelo 3D de una versión del módulo **ESP32**, este dispone de conectividad **Wi-Fi** y **Bluetooth**, su encapsulado es de tipo **SMD** y se puede soldar con facilidad en un **PCB**. Este dispositivo tiene las siguientes características:

- Frecuencia de operación: **240 MHz**
- Arquitectura 32 bit
- Wi-Fi y Bluetooth
- 34 GPIOs
- **ADC** de 12 bits

Otro detalle del **ESP32** es que tiene una variedad de modelos, los cuales se clasifican por series como: **ESP32-WROOM**, **ESP32-WROVER**, **ESP32-MINI**.

Así como hay diferentes tipos de módulos ESP32 también existen varias placas de desarrollo. Para el kit **Bit One ESP32** utilizamos la placa: DevKitC 4. En la imagen 18 se muestra el pinout y se señalan las funciones de cada pin. Las etiquetas color verde representan las funciones básicas y por defecto de los pines GPIO, los otros colores representan funciones particulares y registros especiales que se pueden configurar, pero solo a través de la programación.

Imagen 18. Pinout de la placa ESP32 DevKitC. Fuente: [Espressif](#)

¿Qué es un PCB?

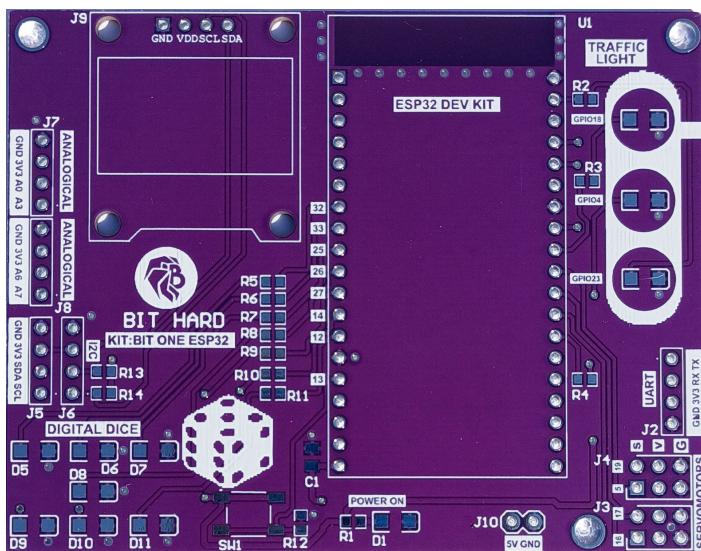


Imagen 19. PCB del kit Bit One ESP32.

Una placa de circuito impreso (PCB) es una placa hecha de material aislante eléctrico (laminado base, plástico de fibra de vidrio y dieléctricos similares) con delgadas tiras conductoras de electricidad metalizadas (conductores impresos) aplicadas en su superficie y pads para conectar complementos de elementos de radio, incluidos módulos y circuitos integrados [5].

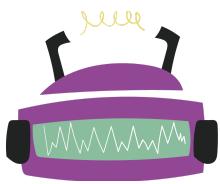
Los **PCB** se utilizan principalmente para proporcionar conexión eléctrica y soporte mecánico a los componentes eléctricos de un circuito.

El color verde que se encuentra en la mayoría de los **PCB** proviene de una máscara de soldadura. Las máscaras de soldadura también vienen en otros colores, como el azul o el rojo [6]. El soldermask del **PCB Boby One** es de color morado y a las letras blancas que contiene se le llama Silkscreen. En el silkscreen van todos los identificadores de los componentes electrónicos y también los detalles adicionales que se quieran agregar.

El principal material conductor utilizado en los circuitos impresos es la lámina de cobre [7]. Las pistas son las trazas o líneas que se pueden observar conectan cada pad, estas permiten la interconexión eléctrica entre cada uno de los componentes que estén presentes en un **PCB**.

Un **PCB** se conforma de diversos elementos entre los cuales encontramos: vias, pads, agujeros, pistas y los componentes electrónicos. Los PCB se conforman de una o varias capas, cuando un PCB es de una capa esto significa que las conexiones eléctricas están disponibles solo en un lado del sustrato aislante. El lado que contiene el layout del circuito se denomina el lado de la soldadura mientras que el otro lado se denomina el lado de los componentes.

Los PCB más comunes son los de dos capas, la interconexión de los elementos se hace en las dos caras del sustrato. A la capa superior se le llama Top layer y a la capa inferior Bottom layer. También existen **PCB** de más de 4 capas, estos se utilizan para aplicaciones especiales. El PCB de Boby One se conforma de 2 capas y tiene un espesor de 1.6mm.



INICIO DEL BIT ONE ESP32

Para comenzar a experimentar con tu kit necesitas lo siguiente:

- PCB Bit One ESP32.
- Cable micro USB tipo B.
- PC o Notebook.

Paso 1: Instalar el driver de la placa **ESP32 DevKitC**.

Dirígete a la página de [Silicon Labs](#) para descargar el driver USB que utiliza la placa ESP32; sin este driver tu PC no reconoce la placa y no podrás utilizarla. En el caso de que hayas utilizado anteriormente la placa **ESP32** deberías tener instalado el driver por lo que debes obviar el paso 1.

Paso 2: Instalación del **Arduino IDE**.

Descarga el último **Arduino IDE** en: www.arduino.cc/en/Main/Software, es software libre, no debes preocuparte por el pago de licencias. Elige el instalador adecuado para el sistema operativo de tu PC. Instálalo en tu equipo, debes seguir las instrucciones dadas por el instalador; es bastante fácil de instalar.

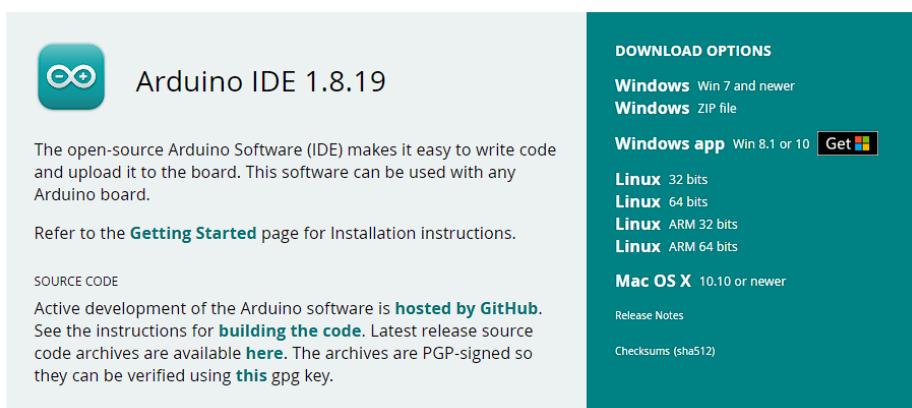


Imagen 20. Opciones de descarga del Arduino IDE.

Después de instalar el **Arduino IDE** debes hacer doble clic en el ícono del software, entonces deberías tener una pantalla similar a esta:

The screenshot shows the Arduino IDE interface. The title bar reads "sketch_aug22a Arduino 1.8.15". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". The toolbar has icons for save, run, upload, and download. The code editor window contains the following code:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
}
```

The status bar at the bottom displays "Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, Core 0, Core 0, None en COM61".

Imagen 21. Interfaz del Arduino IDE.

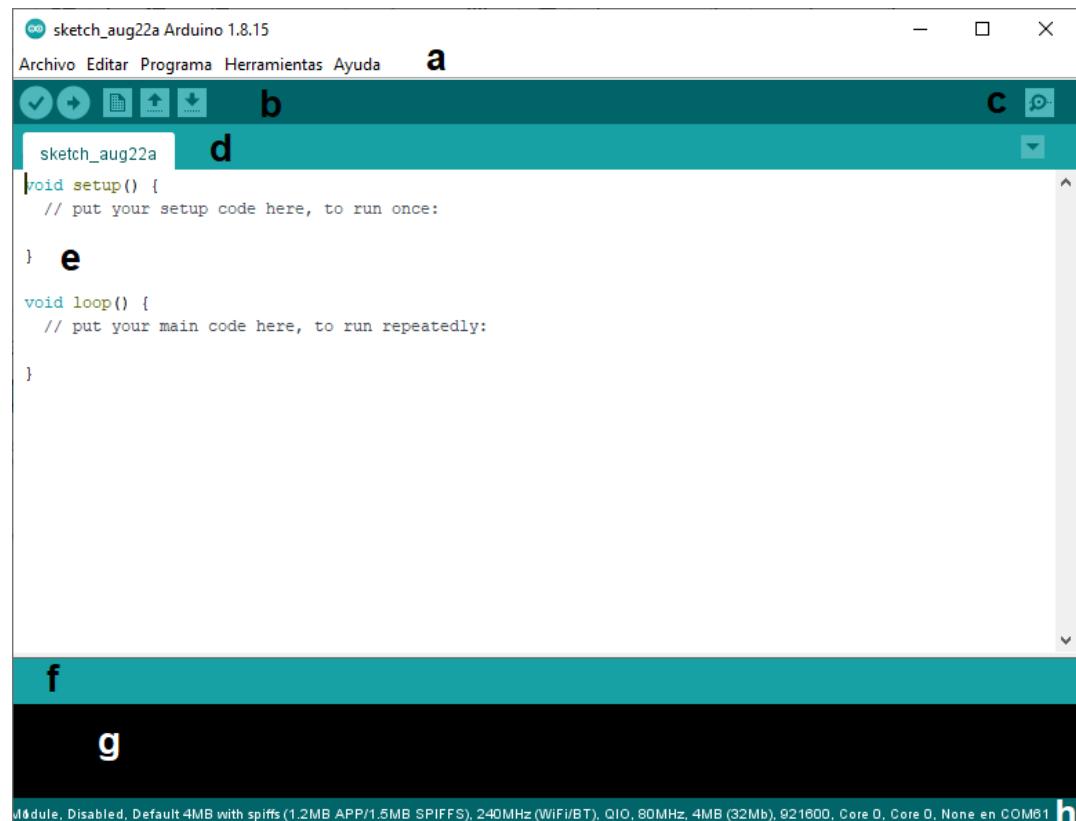


Imagen 22. Elementos de la interfaz del Arduino IDE.

Etiqueta	Descripción	Etiqueta	Descripción
a	Barra del menu	e	Area del código
b	Barra de botones de acción	f	Barra de estatus
c	Monitor serial	g	IDE output
d	Nombre del Sketch	h	Nombre de la placa y número de puerto COM

Tabla 1. Descriptores de la interfaz del Arduino IDE.

Verificar: Compila y verifica tu código. Aquí se detectan errores de sintaxis.

Subir: Sube tu código al ESP32. Cuando el sketch está siendo cargado debería parpadear el led de la placa ESP32.

Nuevo: Aquí abres una nueva pestaña para realizar un sketch nuevo.

Abrir: Aquí puedes abrir tus sketch.

Guardar: Aquí guardas tu sketch.

Paso 3: Instalación de la placa ESP32 en el Arduino IDE.

- En tu IDE de Arduino, ve a **Archivo > Preferencias**

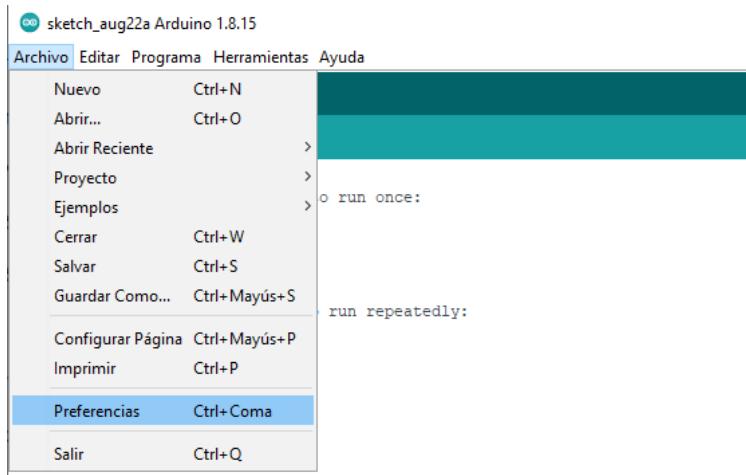


Imagen 23. Preferencias Arduino IDE.

- Ingrese lo siguiente en el campo "Gestor de URLs adicionales de tarjetas ": https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
Nota: si ya tiene agregada otras URL, puede separarlas a través de una coma.

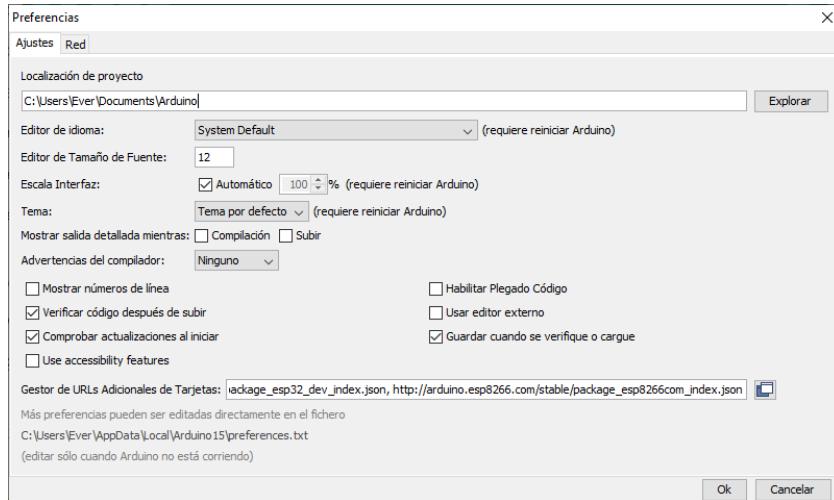


Imagen 24. Gestor de URLs adicionales de tarjetas.

- Abra el gestor de tarjetas. Vaya a Herramientas > Tablero > Administrador de tableros...

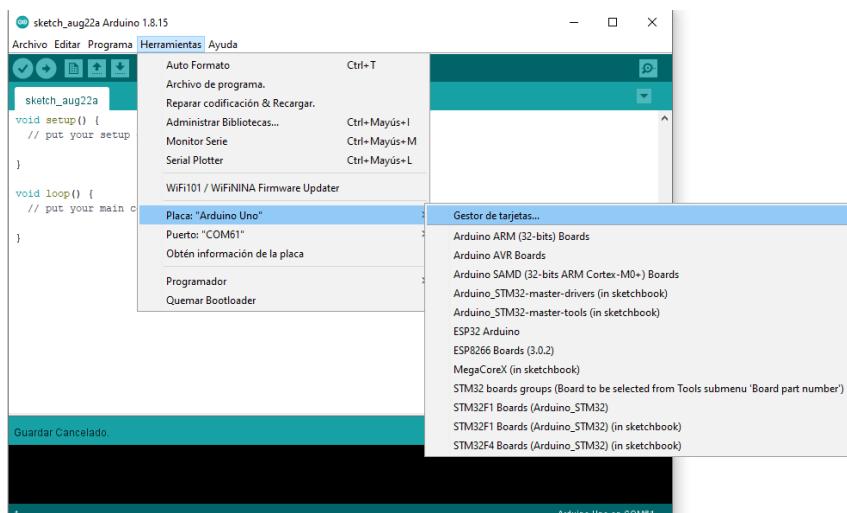


Imagen 25. Gestor de tarjetas.

- Busque **ESP32** y presione el botón de instalación para " **ESP32 by Espressif Systems** ":

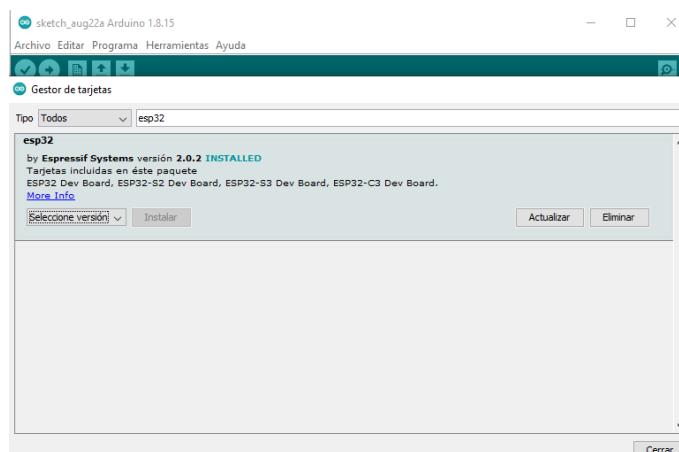


Imagen 26. Instalación de la placa ESP32.

- Eso es todo el procedimiento, debería tenerlo instalado luego de unos segundos.
- Paso 4:** Selección de la placa ESP32. Para ello debes ir a: **Herramientas > Gestor de Tarjetas > ESP32 Arduino > ESP32 Dev Module**

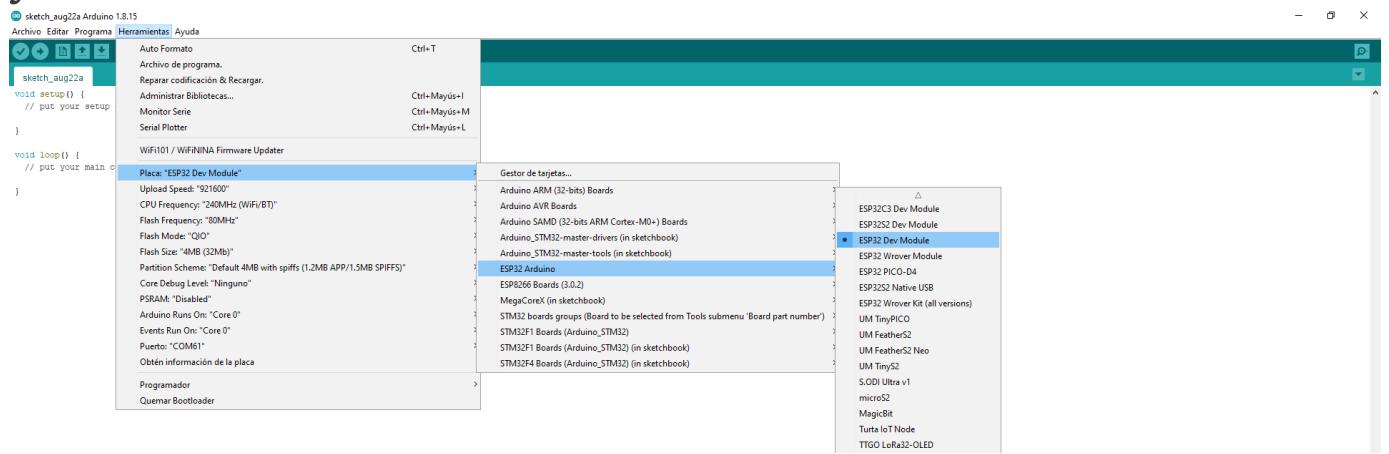
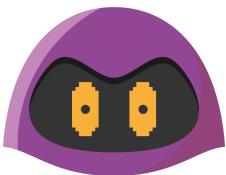


Imagen 27. Selección de la placa ESP32 Dev Module.



Lección 1: Enciende un Led

Para realizar nuestra primera práctica con el **ESP32** utilizaremos un código sencillo para lograr el parpadeo constante de un led; esto quiere decir encender y apagar el led de forma continua. Lo que necesitamos realizar es lo siguiente:

- Configurar el puerto **GPIO23** de nuestro shield como salida (output)
- Indicar que el led conectado a su salida se encienda (**HIGH**)
- Asignar un retraso (delay) de 1000 milisegundos. Este tiempo lo puedes variar, asigna varios tiempos para practicar.
- Indicar que el led conectado a su salida se apague (**LOW**)

Ahora para poder representar esas instrucciones en el entorno Arduino debemos convertirlas en el lenguaje que entiende el programa. Para ello necesitamos seguir unos principios básicos. La estructura básica de un programa en Arduino se compone de tres secciones:

- Declaración de variables globales: se ubica al principio del sketch.
- “**void setup()**”: se delimita por llaves de apertura y cierre.
- “**void loop()**”: se delimita por llaves de apertura y cierre.

Cualquier instrucción escrita en el “**void setup()**” se ejecuta una única vez, llevándose a cabo al momento de encender o reiniciar la placa.

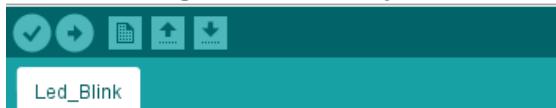
Todas las instrucciones escritas en el “**void loop()**” se ejecutarán inmediatamente del “**void setup()**” de manera infinita, hasta tener una interrupción de la alimentación de la placa o algún reinicio del sistema. En pocas palabras esto es un loop.

El lenguaje utilizado en la plataforma Arduino es “case-sensitive”, lo que quiere decir que importa la forma en que escribas una palabra; si escribes Void, tendrás un error de sintaxis; la forma correcta es: **void**; sin mayúsculas. Toma un poco de tiempo familiarizarse con este principio, pero luego será natural y errores de este tipo suelen no repetirse.

Cuando en un código aparece este símbolo: **//**, quiere decir que es un comentario, no una instrucción, esto se utiliza como guía o descripción de algunas partes del código. Todo lo que esté después de **//** no es considerado por el programa como instrucción.

Luego de comprender estos principios básicos continuamos con el diseño de nuestro código. Lo primero que realizamos es declarar como salida al led conectado al puerto **GPIO23**, y para ello debemos usar la función: **pinMode()**.

Para el loop escribimos en el puerto un estado alto con: **digitalWrite()**, asignamos un segundo de espera con **delay**, y luego debemos escribir un estado bajo y dar otro segundo de espera mientras el ciclo se repite de forma infinita.



```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(23, OUTPUT);  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
    digitalWrite(23, HIGH);  
    delay(1000);  
    digitalWrite (23, LOW);  
    delay(1000);  
}
```

01

Abre un nuevo sketch en el Arduino IDE

02

Escribe este código en tu sketch:

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(23, OUTPUT);  
}  
  
void loop() {  
    // put your main code here, to run  
    // repeatedly:  
    digitalWrite(23, HIGH);  
    delay(1000);  
    digitalWrite (23, LOW);  
    delay(1000);  
}
```



03

Compila el archivo

04

Conecta tu placa al PC

05

Sube el sketch a tu placa

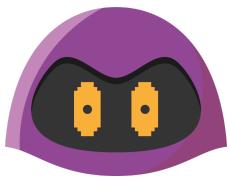
06

Mientras el código está siendo cargado en la placa, el IDE output mostrará una serie de mensajes, justo cuando muestre el mensaje: CONNECTING; deberás presionar el pulsador Boot del ESP32 DevKit por unos cinco segundos. Inmediatamente verás que aparece un porcentaje, al finalizar verás el mensaje: Carga finalizada. Inmediatamente verás encender el led del GPIO23 en tu placa.

```
Subido
Writing at 0x00020001... (71 %)
Writing at 0x00038231... (85 %)
Writing at 0x0003da6f... (100 %)
Wrote 207696 bytes (113042 compressed) at 0x00010000 in 2.1 seconds (effective 772.9 kbit/s)...
Unverified data
26
```

Imagen 28. Código cargado en la placa ESP32.

Después de seguir todas las instrucciones deberás ver el led verde conectado al puerto **GPIO23** encender y apagar constantemente. Si quieres encender los leds de los otros puertos GPIO solo deberás cambiar el número, en el diagrama esquemático encontrarás identificado cada puerto. Al momento de escribir las instrucciones no necesitas agregar la palabra GPIO, solo debes colocar el número. Si necesitas encender el led amarillo entonces cambiaras en el código el numero 23 por el 4, de esta forma veras como enciende el led amarillo en lugar del ver. El código implementado en este primer ejemplo práctico lo puedes descargar en el repositorio GitHub: [Led Blink](#).



Lección 2: Semáforo

sketch_Traffic_Light Arduino 1.8.15

Archivo Editar Programa Herramientas Ayuda

sketch_Traffic_Light

```
// variables
int GREEN = 23;
int YELLOW = 4;
int RED = 18;
int d14 = 14;
int DELAY_GREEN = 5000;
int DELAY_YELLOW = 2000;
int DELAY_RED = 5000;

// basic functions
void setup()
{
    pinMode(GREEN, OUTPUT);
    pinMode(YELLOW, OUTPUT);
    pinMode(RED, OUTPUT);
    pinMode(d14, INPUT);
}

void loop()
{
    green_light();
    delay(DELAY_GREEN);
    yellow_light();
    delay(DELAY_YELLOW);
    red_light();
    delay(DELAY_RED);
}

void green_light()
{
    digitalWrite(GREEN, HIGH);
    digitalWrite(YELLOW, LOW);
    digitalWrite(RED, LOW);
}

void yellow_light()
{
    digitalWrite(GREEN, LOW);
    digitalWrite(YELLOW, HIGH);
    digitalWrite(RED, LOW);
}

void red_light()
{
    digitalWrite(GREEN, LOW);
    digitalWrite(YELLOW, LOW);
    digitalWrite(RED, HIGH);
}
```

01

Abre un nuevo sketch en el Arduino IDE

02

Escribe este código en tu sketch:

```
// variables
int GREEN = 23;
int YELLOW = 4;
int RED = 18;
int d14 = 14;
int DELAY_GREEN = 5000;
int DELAY_YELLOW = 2000;
int DELAY_RED = 5000;

// basic functions
void setup()
{
    pinMode(GREEN, OUTPUT);
    pinMode(YELLOW, OUTPUT);
    pinMode(RED, OUTPUT);
    pinMode(d14, INPUT);
}

void loop()
{
    green_light();
    delay(DELAY_GREEN);
    yellow_light();
    delay(DELAY_YELLOW);
    red_light();
    delay(DELAY_RED);
}

void green_light()
{
    digitalWrite(GREEN, HIGH);
    digitalWrite(YELLOW, LOW);
    digitalWrite(RED, LOW);
}

void yellow_light()
{
    digitalWrite(GREEN, LOW);
    digitalWrite(YELLOW, HIGH);
    digitalWrite(RED, LOW);
}

void red_light()
{
    digitalWrite(GREEN, LOW);
    digitalWrite(YELLOW, LOW);
    digitalWrite(RED, HIGH);
}
```



03

Compila el archivo

04

Conecta tu placa al PC

05

Sube el sketch a tu placa

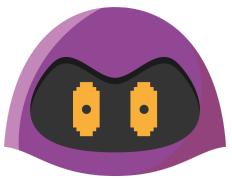
06

Mientras el código está siendo cargado en la placa, el IDE output mostrará una serie de mensajes, justo cuando muestre el mensaje: CONNECTING; deberás presionar el pulsador Boot del ESP32 DevKit por unos tres segundos. Inmediatamente verás que aparece un porcentaje, al finalizar verás el mensaje: Carga finalizada. Inmediatamente verás encender los leds del semáforo en tu placa.

En este código se utilizaron nuevas funciones que en el primer ejemplo no fueron consideradas. Tenemos el uso y declaración de variables.

Una variable es un lugar donde almacenar un dato, tiene un nombre, un valor y un tipo. Los nombres de las variables pueden tener letras, números y símbolos. En la variable el espacio entre dos palabras lo realizamos de la siguiente forma: **DELAY_GREEN**. Se asignó a cada color de led una variable, y para no complicarnos con los retardos entre cada color se establecieron variables para los delay de cada color del semáforo, de esta forma se asignan tiempos reales a cómo funciona un semáforo. La variable d14, aparece en el código porque el **D10** de la placa suele permanecer encendido, para poder desactivarlo debemos declararlo en el código como un input, de esta forma el led se apagará.

Recuerda que puedes descargar el código de ejemplo en el repositorio GitHub: [Semáforo](#).



Lección 3: DADO DIGITAL

```
Code_Digital_Dice Arduino 1.8.15
Archivo Editar Programa Herramientas Ayuda
Code_Digital_Dice
int button = 13; // specifying button

int led1 = 32; // specifying LEDs
int led2 = 33;
int led3 = 25;
int led4 = 26;
int led5 = 27;
int led6 = 14;
int led7 = 12;
long num;
int buttonstate;

void setup() {
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
    pinMode(led3, OUTPUT);
    pinMode(led4, OUTPUT);
    pinMode(led5, OUTPUT);
    pinMode(led6, OUTPUT);
    pinMode(led7, OUTPUT);
    pinMode(button, INPUT);
    randomSeed(analogRead(0));
}

void loop() {
    buttonstate = digitalRead(button);
    if(buttonstate == HIGH){
        num = random(1,7); // Generates random number between 1 and 7
        if (num == 1){
            digitalWrite(led4,HIGH);
            delay(2000); // provides time delay
        }
        if (num == 2){
            digitalWrite(led3,HIGH);
            digitalWrite(led5,HIGH);
            delay(2000);
        }
        if (num == 3){
            digitalWrite(led3,HIGH);
            digitalWrite(led4,HIGH);
            digitalWrite(led5,HIGH);
            delay(2000);
        }
        if (num == 4){
    }}
```

01

Abre un nuevo sketch en el Arduino IDE

02

Copia el código del sketch ubicado en el repositorio: [Code Digital Dice](#)

03

Compila el archivo

04

Conecta tu placa al PC

05

Sube el sketch a tu placa

06

Diviértete cambiando los números del dado digital (presiona el botón SW1)

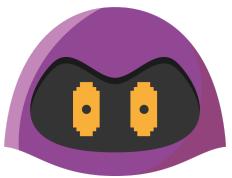
En este nuevo ejemplo utilizaremos un nuevo componente electrónico: un pulsador. Se identifica en la placa como **SW1**, este pulsador es normalmente abierto (**NA**), esto significa que su contacto interior permanece abierto a menos que sea pulsado, en lo que se mantiene presionado su contacto se cierra, pero al soltarlo se vuelve a abrir. Utilizamos 7 leds para representar los dígitos del dado, con cada toque del pulsador se genera un número aleatorio y luego de unos segundos los leds son apagados. El código puede resultar un poco complicado de entender, pero con la práctica podrás entender cada aspecto.

Sin embargo, comentamos las funciones nuevas que utilizamos:

- **long num.** Es un formato de variable numérica extendida, se refiere a números enteros.
- **randomSeed(analogRead(0))**. La función **randomSeed** inicializa el generador de números aleatorios, pero las computadoras suelen cometer errores en la generación de números aleatorios, para solucionar este problema se agrega la función **analogRead(0)**, lo que representa dar un punto de partida aleatorio. En este caso, el código lee el valor de una entrada analógica no conectada y la usa como un punto de partida.
- **if**. La instrucción **if** hace referencia a una condición que establecemos en nuestro código. El **if** utilizado en el código permite verificar el estado del pulsador y ejecutar la instrucción asignada; si el pulsador es presionado genera un número aleatorio, pero si no se encuentra presionado no genera ningún número.

En la web podrás encontrar diferentes formas de programar un dado digital, prueba con dos códigos distintos y evalúa con se adapta mejor a tus gustos.

El código de ejemplo lo puedes descargar en el repositorio GitHub: [Dado Digital](#).



Lección 4: FADE IN Y FADE OUT

Fade_In_Fade_Out Arduino 1.8.15
Archivo Editar Programa Herramientas Ayuda



```
// Digital pins with PWM for the LEDs
const int GPIO18 = 18;
const int GPIO4 = 4;
const int GPIO23 = 23;
const int GPIO14 = 14;

int brightness = 0;

void setup()
{
    // We define the digital pins as outputs, except gpio14
    pinMode(GPIO18, OUTPUT);
    pinMode(GPIO4, OUTPUT);
    pinMode(GPIO23, OUTPUT);
    pinMode(GPIO14, INPUT);
}

void loop()
{
    // Efect FADE-IN GPIO18
    for(brightness=0; brightness < 256; brightness++)
    {
        analogWrite(GPIO18, brightness);
        delay(7);
    }
    // Efect FADE_OUT GPIO18
    for(brightness=255; brightness >=0; brightness--)
    {
        analogWrite(GPIO18, brightness);
        delay(7);
    }

    // Efect FADE-IN GPIO4
    for(brightness=0; brightness < 256; brightness++)
    {
        analogWrite(GPIO4, brightness);
        delay(7);
    }
    // Efect FADE_OUT GPIO4
    for(brightness=255; brightness >=0; brightness--)
    {
        analogWrite(GPIO4, brightness);
        delay(7);
    }
}
```

01

Abre un nuevo sketch en el Arduino IDE

02

Copia el código del sketch ubicado en el repositorio: [Fade IN Fade Out](#)

03

Compila el archivo

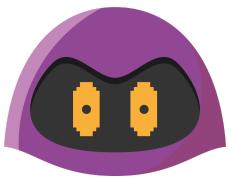
04

Conecta tu placa al PC

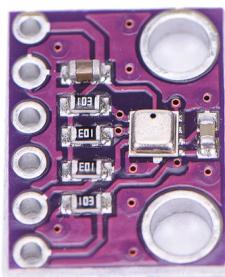
05

Sube el sketch a tu placa

En este nuevo ejemplo utilizaremos los leds de los puertos GPIO 18, 4 y 23; los cuales se utilizan para el traffic light. El Fade IN y Fade Out es un efecto utilizado para demostrar el uso de las salidas digitales como **PWM**. El **PWM** es una técnica de modulación de ancho de pulso, básicamente podemos configurar qué tan ancho será el pulso de una señal digital. En el ejemplo que utilizamos podemos hacer que los leds se vayan encendiendo y apagando, controlando la luminosidad y el tiempo en que tarda en variar. La secuencia del encendido y apagado puede ser más lenta o rápida, dependiendo del valor del `delay()` que introducimos en la programación.



Lección 5: USO DEL BMP280



01

Abre un nuevo sketch en el Arduino IDE

02

Copia el código del sketch ubicado en el repositorio: [BMP280 Sensor](#)

03

Compila el archivo

04

Conecta el sensor **BMP280** a cualquiera de los terminales I2C de la placa **Bit One ESP32**

05

Conecta tu placa al PC

06

Sube el sketch a tu placa

En esta oportunidad aprenderemos a utilizar el sensor **BMP280**, el cual permite sensar las variables físicas: temperatura, presión atmosférica y estimación de altitud sobre el nivel del mar. La tensión de alimentación del **BMP280** es 1.8V a 3.6V.

Este sensor tiene un rango de temperatura de -40º a 85ºC, con una precisión de ±1.0C. Para la presión atmosférica / altímetro es de 300 hPa(hecto pascal) a 1110 hPa, equivalente a una altitud de -500m a 9000m sobre el nivel del mar.

El **BMP280** utiliza los protocolos de comunicación: **I2C** y **SPI**. En nuestra placa utilizaremos el **I2C** para comunicarnos con el sensor.

El **I2C** es un protocolo de conexión de interfaz de bus incorporado en dispositivos para establecer comunicación serial.

En el protocolo I2C el sensor **BMP280** utiliza los siguientes pines:

SCK = SCL

SDI = SDA

Para poder realizar la adquisición de las variables físicas a través del sensor **BMP280**, debemos utilizar las librerías: [Adafruit BMP280](#) y [Adafruit Unified Sensor](#). Siga los siguientes pasos para instalar las bibliotecas en su **Arduino IDE**:

1. Abra el IDE de Arduino y diríjase a: **Programa > Incluir Libreria > Administrar bibliotecas**. El Administrador de la biblioteca abrirá en unos segundos.
2. En el cuadro de búsqueda escriba: "Adafruit BMP280" y proceda a instalar la biblioteca **BMP280** de Adafruit.

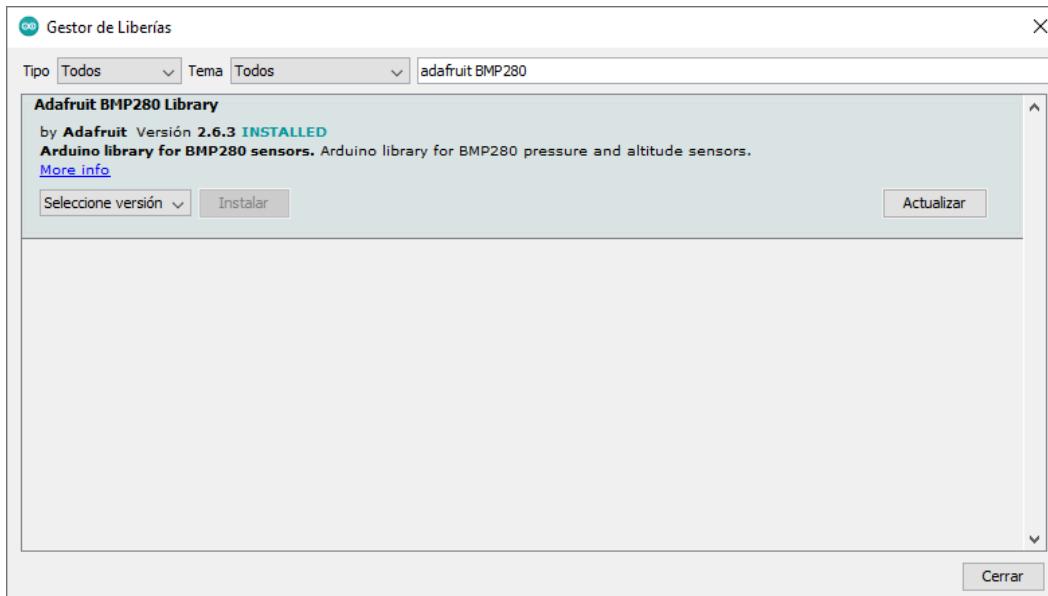


Imagen 29. Instalación de la librería BMP280 en el Arduino IDE .

3. Luego de instalar la librería BMP280 de Adafruit, coloca “Adafruit Unified Sensor” en el cuadro de búsqueda y procede a instalar la biblioteca.

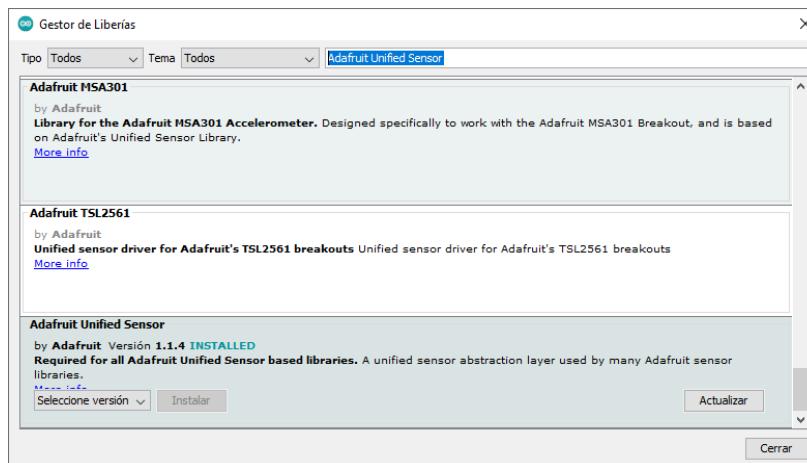


Imagen 30. Instalación de la librería Adafruit Unified Sensor en el Arduino IDE.

4. Después de instalar las dos librerías debes reiniciar el **Arduino IDE**.

Una librería es un conjunto de recursos diseñados para simplificar procesos. Los recursos pueden incluir: subrutinas, funciones, clases y valores. Las librerías básicamente son herramientas que permiten ahorrar tiempo y facilitar el proceso de programación para controlar un dispositivo. En el caso de las librerías: **BMP280** y **Adafruit Unified Sensor**, se ahorra una cantidad de tiempo considerable al utilizarlas; y también se cuenta con un código validado que permite evitar errores en la configuración del dispositivo. Si tuviéramos que desarrollar el código del sensor **BMP280** desde cero, se tendría un sketch bastante largo y complejo, y probablemente se presentaron errores que evitan obtener un resultado exitoso. Las librerías son un gran recurso en el trabajo de programación y configuración de sensores y otros dispositivos que utilicemos para los proyectos de electrónica con el ESP32.

Luego de tener las librerías instaladas no deberías tener problemas al compilar el código, ahora podrás utilizar el sensor **BMP280** y comenzar a sensar variables físicas. En la imagen 31 podrás observar el resultado que deberías ver en tu Monitor Serie de Arduino.

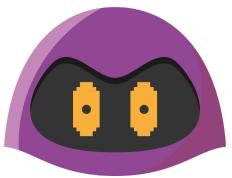


The screenshot shows the Arduino Serial Monitor window titled "COM61". The window displays a series of text messages from the serial port. The messages are timestamped and provide sensor readings:

```
19:48:49.665 -> Pressure = 960.17 hPa
19:48:49.665 -> Approx altitude = 504.25 m
19:48:49.710 ->
19:48:51.640 -> Temperature = 18.27 *C
19:48:51.640 -> Pressure = 960.17 hPa
19:48:51.687 -> Approx altitude = 504.27 m
19:48:51.687 ->
19:48:53.613 -> Temperature = 18.26 *C
19:48:53.660 -> Pressure = 960.16 hPa
19:48:53.660 -> Approx altitude = 504.30 m
19:48:53.706 ->
19:48:55.647 -> Temperature = 18.25 *C
19:48:55.647 -> Pressure = 960.17 hPa
19:48:55.693 -> Approx altitude = 504.28 m
19:48:55.693 ->
```

At the bottom of the monitor, there are several configuration options: "Autoscroll" (checked), "Mostrar marca temporal" (checked), "Nueva línea" (dropdown menu), "9600 baudio" (dropdown menu), and "Limpiar salida" (button).

Imagen 31. Datos mostrados en el Monitor Serie del Arduino IDE.



Lección 6: USO DEL DISPLAY OLED



01

Abre un nuevo sketch en el Arduino IDE

02

Copia el código del sketch ubicado en el repositorio: [Hello World](#)

03

Compila el archivo

04

Conecta tu placa al PC

05

Sube el sketch a tu placa

En este ejemplo utilizaremos el display **OLED** (diodo orgánico emisor de luz) que viene en nuestra placa; su modelo es el **SSD1306**, es un display de tipo monocolor y tiene unas dimensiones de 0,96 pulgadas con capacidad de 128x64 caracteres. El modelo utilizado dispone de 4 pines y se comunica con el **ESP32** a través del protocolo de comunicación **I2C**.

Para controlar el display Oled se necesita instalar una serie de librerías, las cuales son: [adafruit_SSD1306.h](#) y [adafruit_GFX.h](#). Siga las siguientes instrucciones para instalar las librerías.

1. Abra el **Arduino IDE** y diríjase a: **Programa > Incluir Librería > Administrar bibliotecas**. El Administrador de la biblioteca abrirá en unos segundos.
2. En el cuadro de búsqueda escriba: "**SSD1306**" y proceda a instalar la biblioteca **SSD1306** de **Adafruit**.

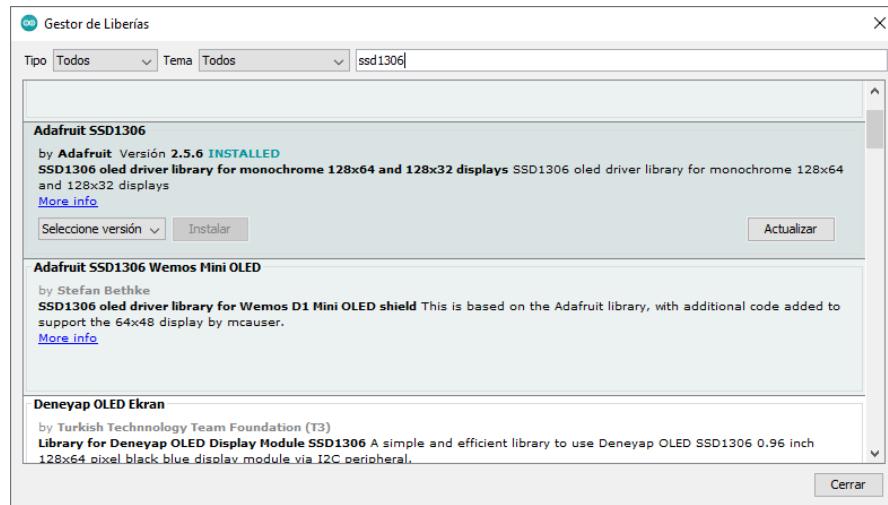


Imagen 32. Instalación de la librería SSD1306 en el Arduino IDE.

3. Luego de instalar la librería SSD1306 de Adafruit, coloca “GFX” en el cuadro de búsqueda y procede a instalar la biblioteca.

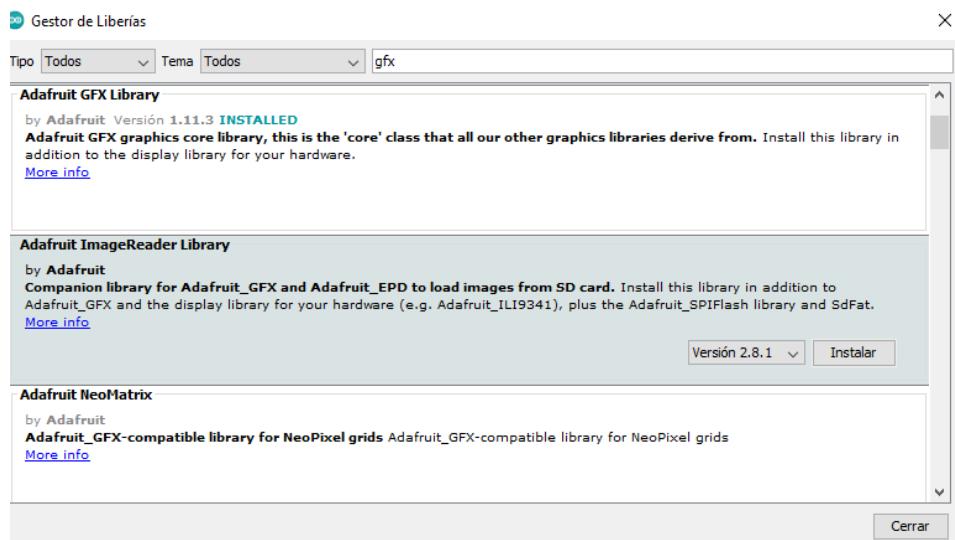


Imagen 33. Instalación de la librería GFX en el Arduino IDE.

4. Despues de instalar las dos librerías debes reiniciar el Arduino IDE.

Luego de tener las librerías instaladas no deberías tener problemas al compilar el código, ahora podrás utilizar el display Oled.

Como el display Oled es un poco complejo de utilizar te dejamos un [enlace](#) con ejemplos básicos sobre cómo sacar el mejor provecho posible del mismo. Agradecemos a [Randomnerdtutorials](#) por sus valiosos recursos para la programación.

BIBLIOGRAFÍA

- [1] T. Ruiz, O. Arbelaitz, I. Etxeberria y A. Ibarra, en: Análisis Básico de Circuitos Eléctricos y Electrónicos. Pearson Education S.A, Madrid 2004, pp 9, 25.
- [2] R. Boylestad, en: Introducción al Análisis de Circuitos. Pearson Educación, México 2004, pp 59, 75, 98.
- [3] R. Boylestad, L. Nashelsky, en: Electrónica: Teoría de Circuitos y Dispositivos Electrónicos. Pearson Educación, México 2009, pp 42, .
- [4] Serie de módulos ESP32. Visitado en: Julio 18, 2022.[Online]. Disponible: <https://www.espressif.com/en/products/modules/esp32>
- [5] A. Belous and V. Saladukha, in: High-Speed Digital System Design. Springer, Cham, 2020, pp 544, 575.
- [6] S. Bhunia and M. Tehranipoor (2018). In: Hardware Security. Elsevier Science & Technology, San Francisco, United States, pp 39, 82.
- [7] Clyde F. Coombs, Jr., Printed Circuits Handbook Sixth Edition, McGraw Hill, New York: 2008. pp 108, 117, 118, 156, 335
- [8] Guide for I2C OLED Display with Arduino. Visitado en: Agosto 28, 2022.[Online]. Disponible: <https://randomnerdtutorials.com/guide-for-oled-display-with-arduino/>