Prácticas con sensores digitales

Copyright 2026

Actividad-01. LED

> Página extraída de Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Enunciado

Realizar un programa que encienda y apague el LED rojo conectado al pin D12 DE LA PLACA STEAMMAKER.

Teoría

El diodo LED (Light Emitting Diode) es un diodo semiconductor capaz de emitir luz, lo mas usuales dentro del espectro visible aunque también pueden ser de infrarrojos, laser, etc. Su uso mas habitual es como indicador y, últimamente cada vez mas frecuentes en iluminación. Sus principales ventajas frente a luces incandescentes son:

- Menor consumo de energía
- Mayor vida útil
- Menor tamaño
- Gran durabilidad y fiabilidad

En la imagen siguiente vemos el aspecto físico que tiene y su símbolo electrónico.

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

El color de la cápsula es simplemente orientativo de la longitud de onda que define realmente el color de la luz emitida. Por ello el LED con la cápsula transparente puede emitir en cualquiera de los colores del espectro visible.

El LED es un dispositivo que tiene polaridad siendo su comportamiento el siguiente: En polarización directa (ánodo a positivo y cátodo a negativo) el LED emite luz y en polarización inversa (ánodo negativo y cátodo positivo) se comporta prácticamente como un interruptor abierto.

Para su correcto funcionamiento el diodo LED se polariza poniéndole en serie una resistencia que limita la corriente que pasa a través del mismo y, por tanto, determina el nivel de brillo de la luz emitida.

Sin entrar en detalles en la tabla siguiente se dan los valores de tensión directa (VF) y corriente directa (IF) para los colores mas habituales de LEDs. A partir de estos valores y el valor de tensión de alimentación de nuestro LED podemos calcular el valor de la resistencia serie sin mas que aplicar la formula indicada.

Color del LED	Tensión directa ($V_{F)}$	Corriente directa (I_F)	$R=\frac{Vcc-V_F}{I_F}$
Rojo	1,5 V	15 mA	
Verde	1,8 V	15 mA	
Amarillo	1,8 V	15 mA	
Azul	3 V	20 mA	
Blanco	2,7 V	20 mA	

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Si quieres saber más de cómo calcular la Resistencia para un led mira esta hoja de cálculo pero sustituye la Vcc por 5 V

En la TdR STEAM

La placa Imagina TDR STEAM dispone de un LED rojo conectado al pin D12 tal y como se indica en la serigrafía de la propia placa y que podemos ver en la imagen siguiente:

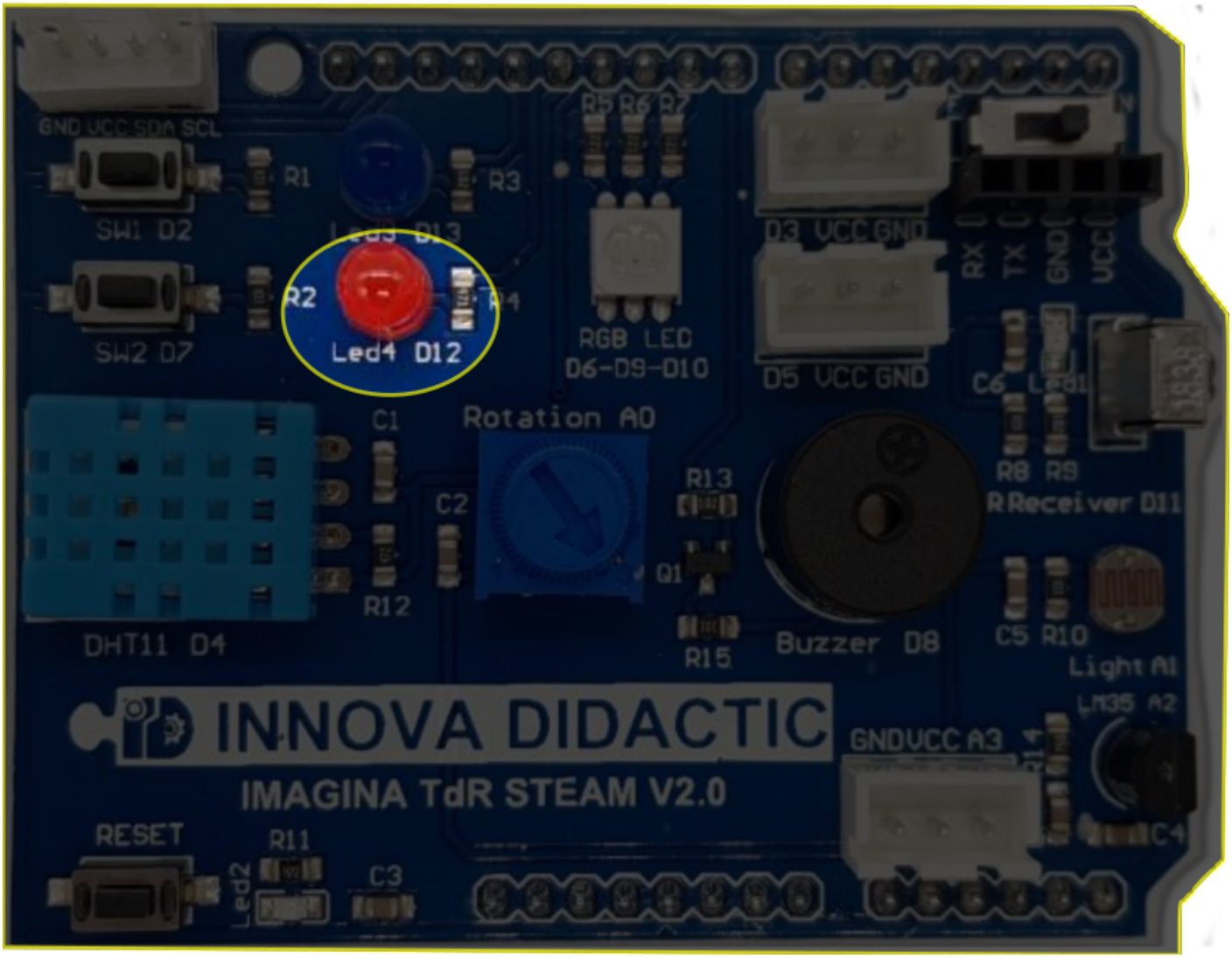


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Programando la Actividad

Entramos en ArduinoBlocks y nos identificamos convenientemente. Vamos a crear un nuevo proyecto para la placa ESP32 STEAMakers + TdR STEAM.

Escogemos el bloque LED de TDR STEAM y lo colocamos en el bucle, quedando algo similar a la imagen siguiente:

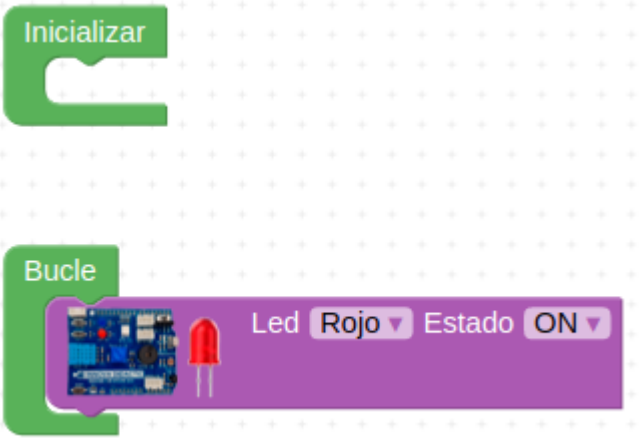


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Bloque LED de TDR STEAM

Podemos comprobar haciendo clic sobre las flechas como podemos cambiar de LED y también como podemos cambiar de estado al LED. Si solamente dejamos este bloque el LED permanecerá encendido de forma permanente y para que se acabe debemos ponerlo en estado OFF. En la imagen siguiente vemos el programa como quedaría.

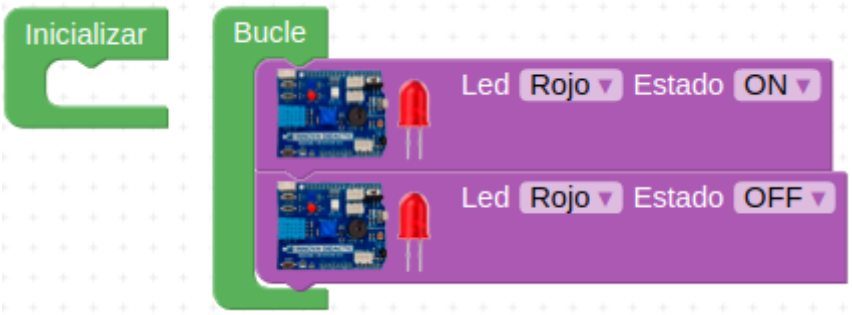


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

LED D12 a ON y OFF

Pero este programa no nos permite ver el efecto de encendido y apagado del LED debido a la velocidad de procesamiento que tiene la placa UNO que típicamente trabaja a 16 MHz. Esto supone que el micro tarda en ejecutar una instrucción algo mas de 0,06 microsegundos que es mucho menor que la persistencia visual humana de 0,1 segundo aproximadamente. Es decir, en realidad el LED se enciende y se apaga, pero nuestro ojo no puede apreciarlo y lo verá siempre encendido debido a la persistencia visual.

Tenemos por tanto que dejar el diodo un tiempo encendido y otro apagado (pueden ser el mismo tiempo) y para ello vamos a colocar un bloque Esperar desde el bloque de Tiempo. Si dejamos el tiempo por defecto en 100 milisegundos el diodo se encenderá y apagará cada segundo (1000 ms = 1s).

El programa final queda como vemos en la imagen siguiente y lo tenemos disponible en el enlace ESP32-SM-Actividad-01.

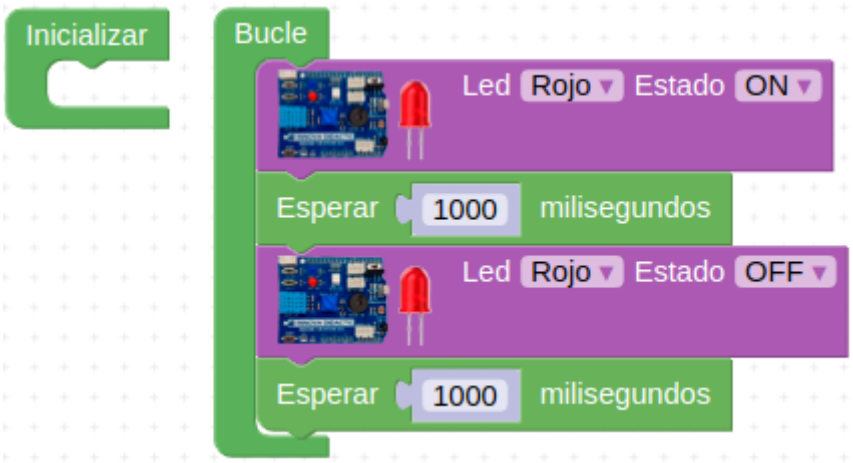


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Programa final para el LED rojo

> Coloca la placa Imagina TDR STEAM encima del ESP32

Conectamos nuestra placa a un puerto USB del ordenador, ponemos en marcha el programa Connector y cargamos el programa. Podemos observar como el diodo LED rojo parpadea con un intervalo de un segundo.

De esta forma el programa queda grabado en la memoria de programa del microcontrolador y el ciclo se repetirá por tiempo indefinido o hasta que quitemos la alimentación a la placa. Si alimentamos la placa externamente con una fuente de alimentación se ejecutará el programa en memoria.

Retos de ampliación

R1.A1. Cambiar los tiempos para que el parpadeo sea más rápido, mas lento y que los tiempos de encendido y apagado no coincidan.

R2.A1. Repetir el ejercicio A1 utilizando el LED azul conectado al pin D13.

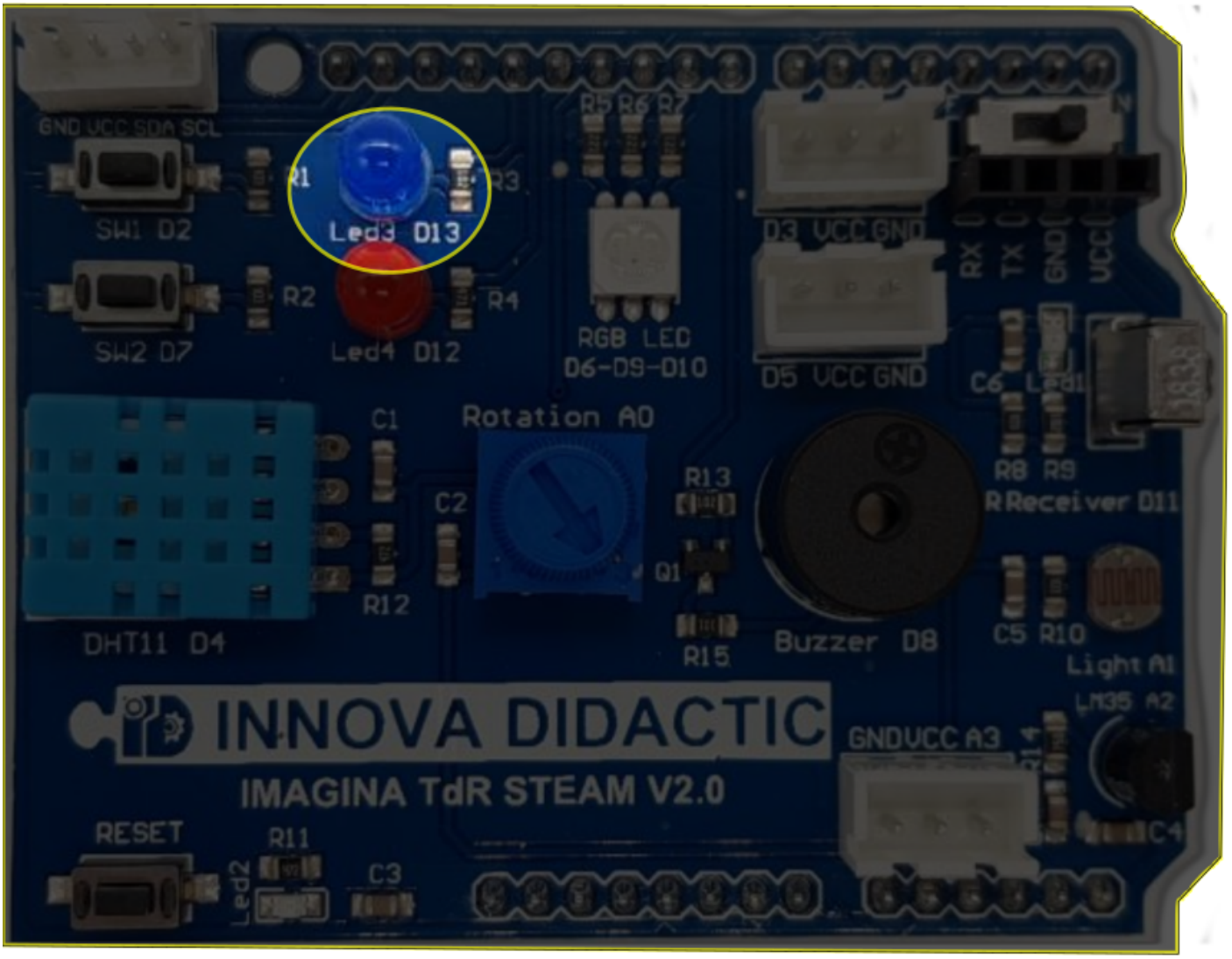


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Copyright 2026

Actividad-02. LEDs rojo y azul

> Página extraída de Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Enunciado

Como ya hemos visto anteriormente la placa dispone de dos LED (uno rojo y otro azul). Vamos a realizar un programa para que se vayan alternando en su encendido y apagado.

Teoría

Vamos a ver como se hacen los ciclos de repetición o bucles en ArduinoBlocks.

- **Repetir**. En el menú de Control hay un bloque 'Repetir (valor) veces hacer...', como el de la imagen siguiente:

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Lo que pongamos en hacer se va a repetir tantas veces como indiquemos en el número de veces, que por defecto estará a 10.
En realidad lo que estamos haciendo es lo que en programación se conoce como bucle for.

- **Repetir según condición.** En la imagen siguiente vemos dos bloques que repiten su interior mientras, o hasta, que se cumpla una condición.

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

- **Contador**. Realiza un bucle contando con un variable índice (normalmente i o j). Se define un valor de inicio, una valor de fin y los incrementos que se realizarán en cada iteración del bucle. Dentro del bucle podremos usar esta variable índice.

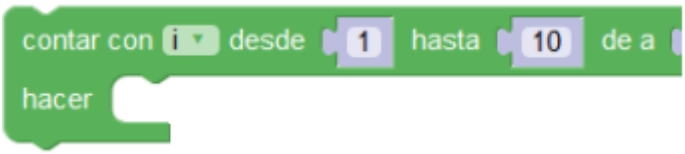


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

En la TdR STEAM

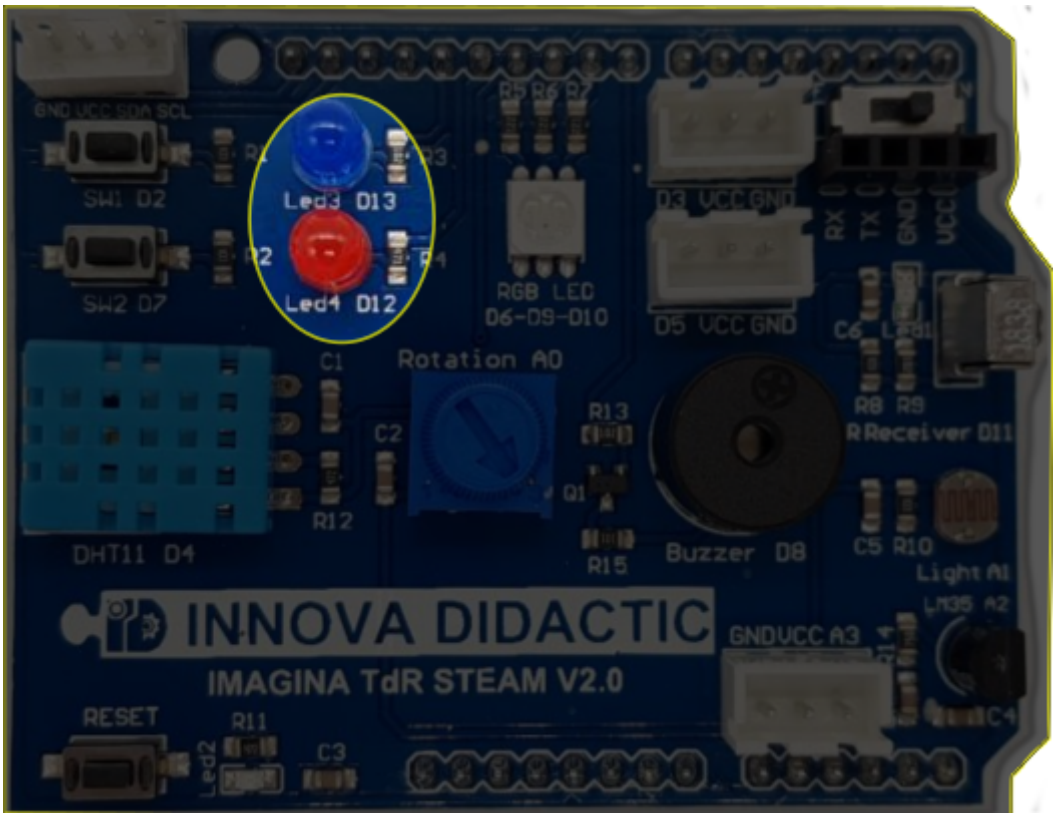


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Los dos LEDs de la TdR-STEAM

Programando la Actividad

Vamos a hacer que ambos diodos se enciendan y apaguen de forma simultanea con un programa como el siguiente, que lo tenemos disponible en Actividad-02. LED-rojo-azul

Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Intermitencia LEDs rojo y azul

Retos de ampliación

A2.R1. Hacer que los LEDs rojo y azul se enciendan simultáneamente con tiempos de espera de 300ms y 150ms respectivamente.

A2.R2. Realizar 4 intermitencias de 500ms con el LED azul y cuando estas acaben dejar el LED rojo encendido durante 1.5 segundos. Esperar un segundo para iniciar de nuevo el proceso.

A2.R3. Realiza 5 intermitencias de 500ms con el LED azul cada vez que el LED rojo lo hace 3 veces a intervalos de 150ms. Esperar un segundo para iniciar de nuevo el proceso.

Solución A22.R2
El Programa es el de la imagen siguiente:

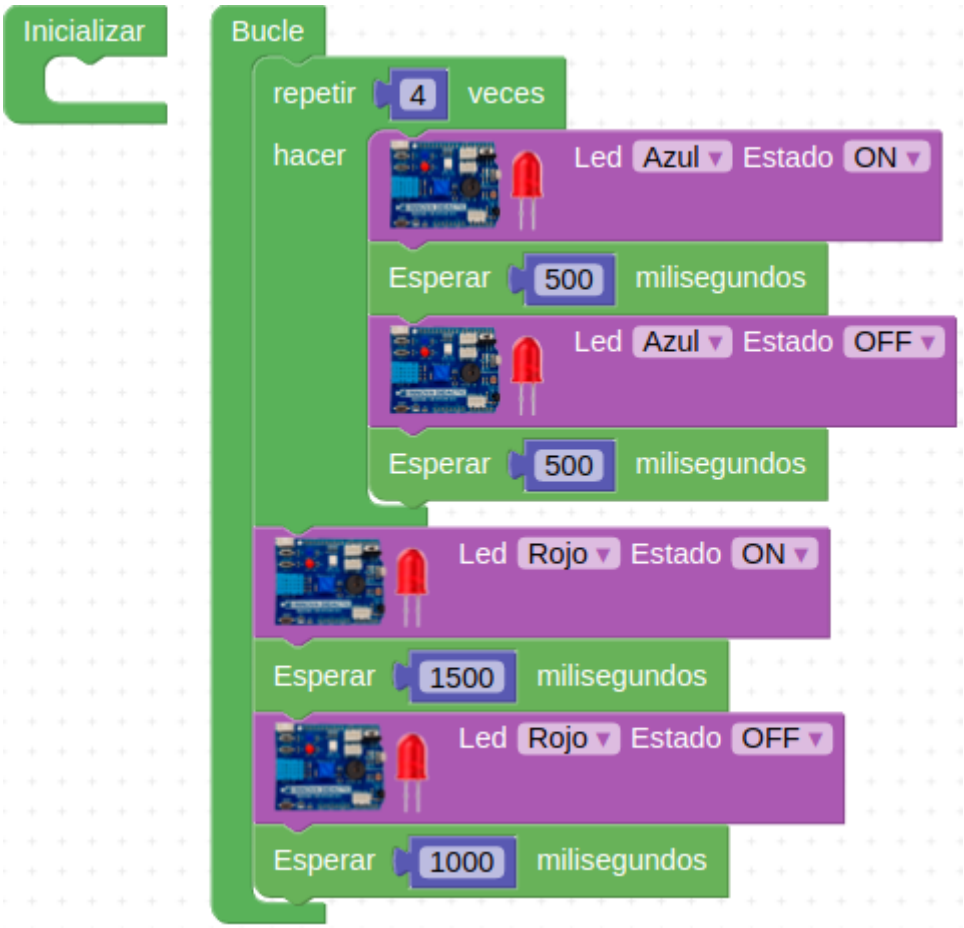


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Solución A2.R3
El Programa es el de la imagen siguiente:

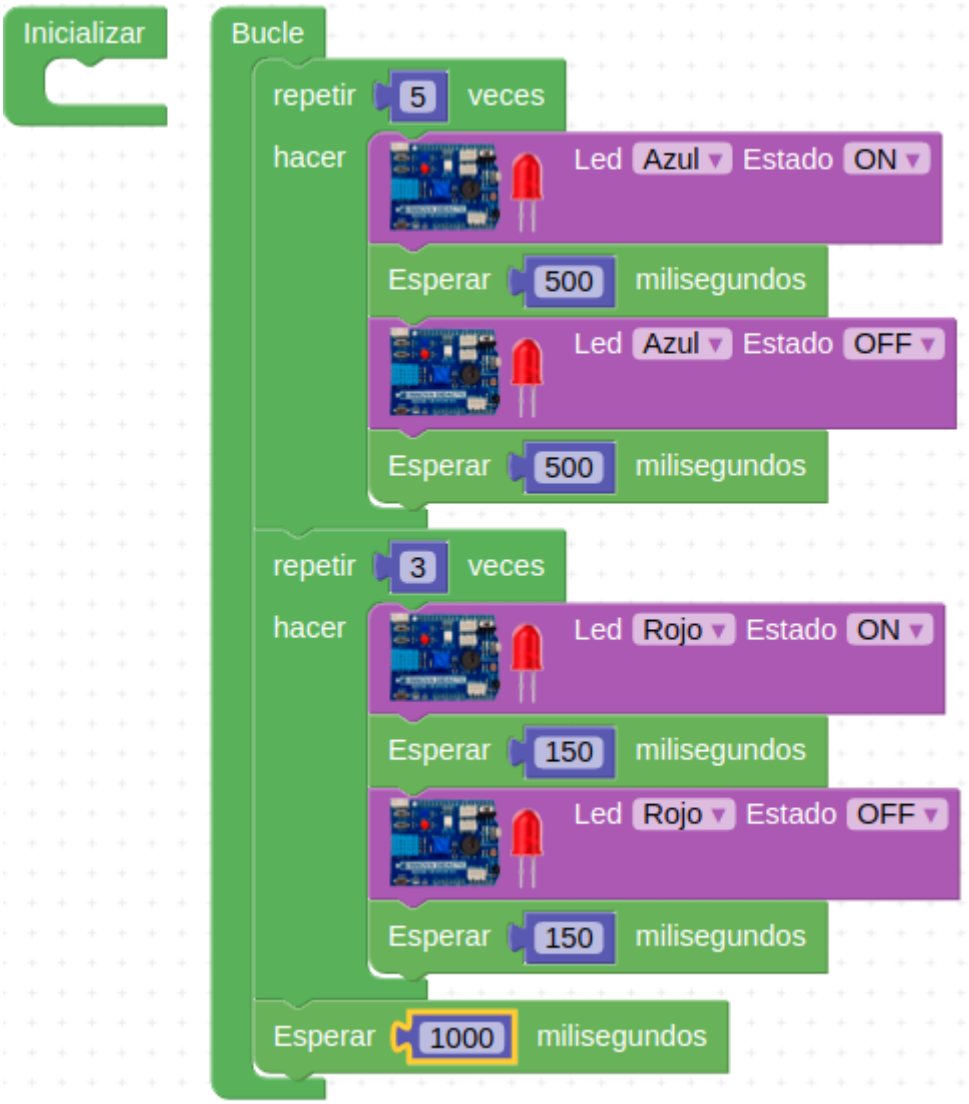


Imagen Federico Coca [Notas sobre ESP32 STEAMakers] CC-BY-SA

Actividad-03. Parpadeo intermitente. Multitarea

> Página extraída de Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Enunciado

Ya hemos visto como hacer funcionar a los LEDs de diferentes formas, pero ahora vamos a introducir un concepto que nos puede resultar bastante útil en el futuro, se trata de la multitarea en ArduinoBlocks.

Teoría

Este apartado se extrae de ArduinoBlocks - FreeBook disponible en Free Book (online & updated).

ArduinoBlocks nos permite utilizar una capa para implementar un sistema multitarea avanzado basado en FreeRTOS (del inglés Real Time Operating System), que es un sistema operativo de tiempo real kernel para dispositivos embebidos para plataformas de microcontrolador que se distribuye bajo licencia MIT. Este sistema permite crear tareas que se ejecutarán de forma paralela (virtualmente). En microcontroladores modestos como el Arduino UNO, Nano o incluso MEGA la multitarea con FreeRTOS es bastante limitada y consume gran parte de los recursos de nuestro Arduino, en caso de necesitar de un sistema multitarea más potente podemos optar por usarlo en placas basadas en ESP8266 o ESP32 con mucha más potencia y recursos (especialmente el ESP32 con doble núcleo y gran potencia de procesamiento y memoria interna)

Los sistemas software de multitarea utilizan un planificador o scheduler que se encarga de repartir el tiempo de procesamiento entre las distintas tareas, de forma que a cada una le toca un tiempo de microcontrolador para ejecutar un poquito de su parte de programa.

En las web de freeRTOS, en su entrada de menú Kernel podemos encontrar los conceptos básicos de multitarea y de programación que vamos a extractar seguidamente.

Conceptos básicos de multitarea

Un procesador convencional como el de Arduino UNO solo puede ejecutar una tarea a la vez, pero al cambiar rápidamente entre tareas, un sistema operativo multitarea puede hacer que parezca que cada tarea se ejecuta simultáneamente. Esto es lo que se representa en el diagrama de la

Figura siguiente que muestra el patrón de ejecución de tres tareas con respecto al tiempo. Los nombres de las tareas están codificados por colores y escritos a la izquierda. El tiempo se mueve de izquierda a derecha y las líneas de colores muestran qué tarea se está ejecutando en un momento determinado. El diagrama superior demuestra el patrón de ejecución concurrente percibido, y el inferior el patrón de ejecución multitarea real.

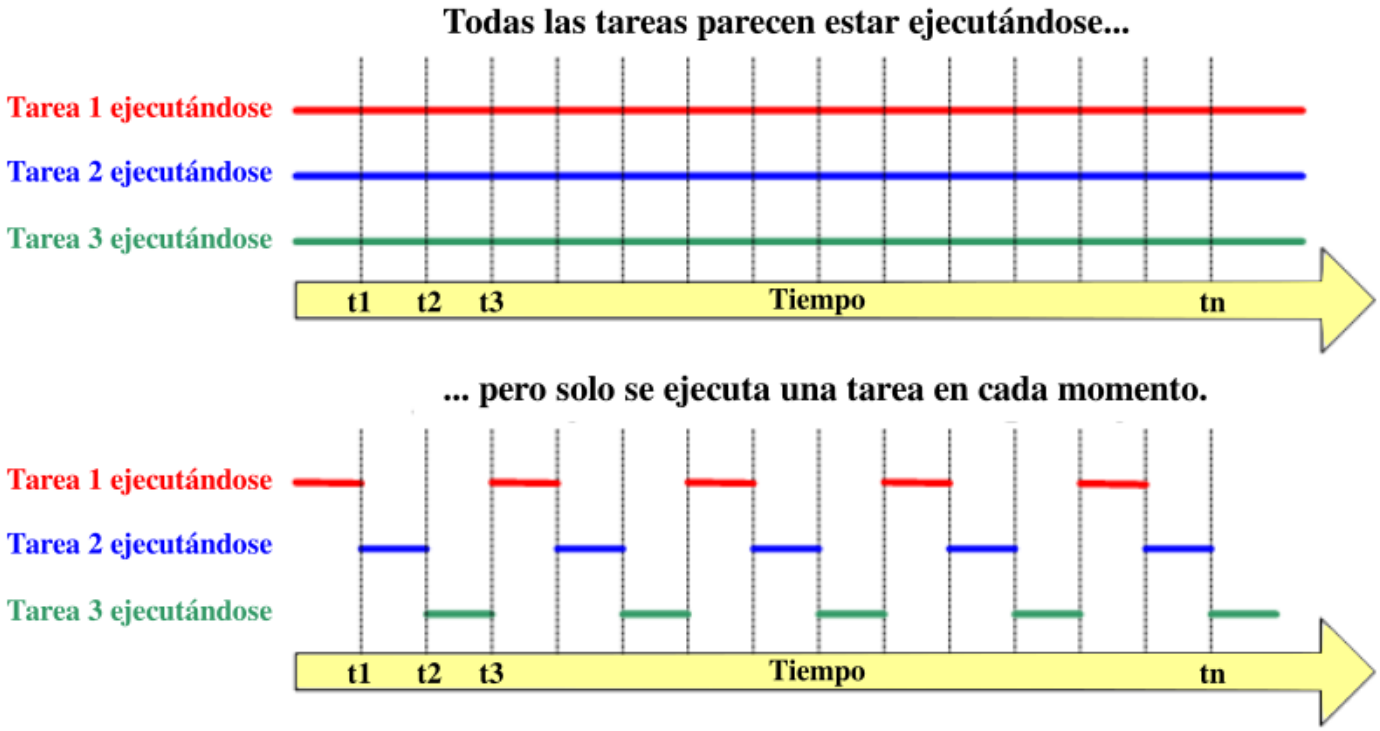


Imagen Federico Coca [Notas sobre ESP32 STEAMakers] CC-BY-SA

Patrón de ejecución de tres tareas con respecto al tiempo

Programación

El programador es quien debe decidir qué tarea debe ejecutarse en un momento determinado. El kernel o núcleo puede suspender y luego reanudar una tarea muchas veces durante el tiempo de vida de la tarea.

Además de ser suspendida involuntariamente por el núcleo o kernel, una tarea puede optar por suspenderse a sí misma. Hará esto si desea retrasar (**dormir**) por un período fijo o esperar (**bloquear**) a que un recurso (por ejemplo, un puerto serie) esté disponible, o que ocurra un evento (por ejemplo, presionar una pulsador). Una tarea bloqueada o inactiva no se puede ejecutar y no se le asignará ningún tiempo de procesamiento.

En la Figura siguiente vemos un posible diagrama de ejecución de tres tareas analizado punto por punto en distintos instantes de tiempo. En los círculos se representan los instantes de tiempo t1 a t10.

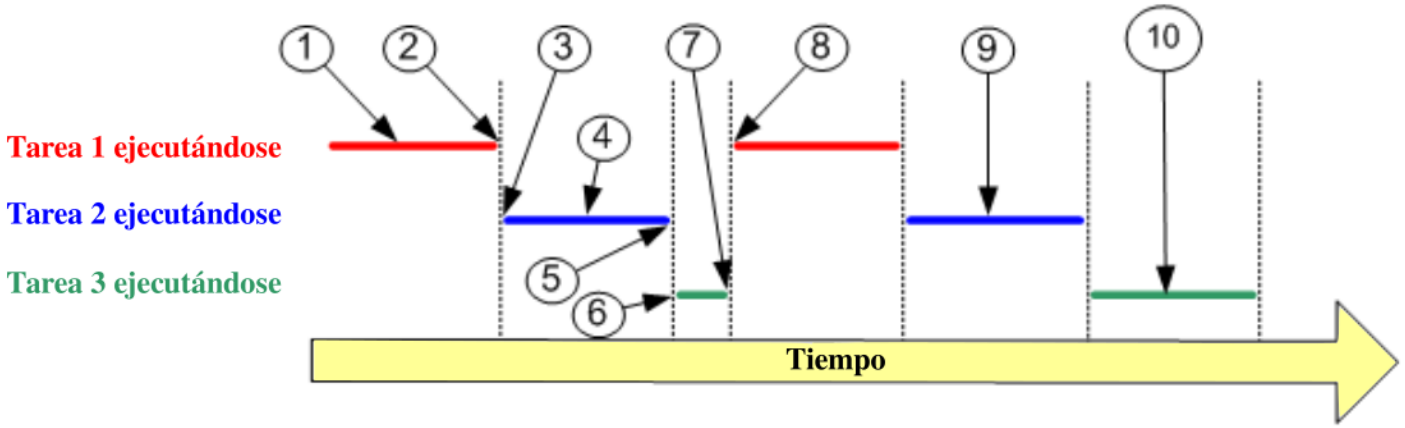


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Diagrama de ejecución de tres tareas en el tiempo

t1: la tarea 1 se está ejecutando.
t2: en el kernel se suspende, o mejor dicho se intercambia, la tarea 1 .
t3: se reanuda la tarea 2.
t4: mientras se ejecuta la tarea 2 el procesador bloquea el puerto serie para su acceso exclusivo.
t5: el kernel suspende la tarea 2.
t6: el kernel reanuda la tarea 3.
t1: la tarea 3 intenta acceder al puerto serie y lo encuentra bloqueado por lo que no puede continuar y se suspende.
t8: el kernel reanuda la tarea 1 .
t9: al ejecutarse de nuevo la tarea 2 se desbloquea el puerto serie.
t10: la tarea 3 ahora si puede acceder al puerto serie y se ejecuta al completo

Planificadores

Los planificadores de multitarea permiten asignar a cada tarea una prioridad, para así darle preferencia a las tareas más críticas o que necesitan más tiempo de procesamiento. Si creamos muchas tareas con “alta” prioridad puede que afectemos a las demás dejando poco tiempo de procesamiento para ellas. En la Figura siguiente vemos un esquema de varias tareas con distintas prioridades, variando así su tiempo de microprocesador asignado.

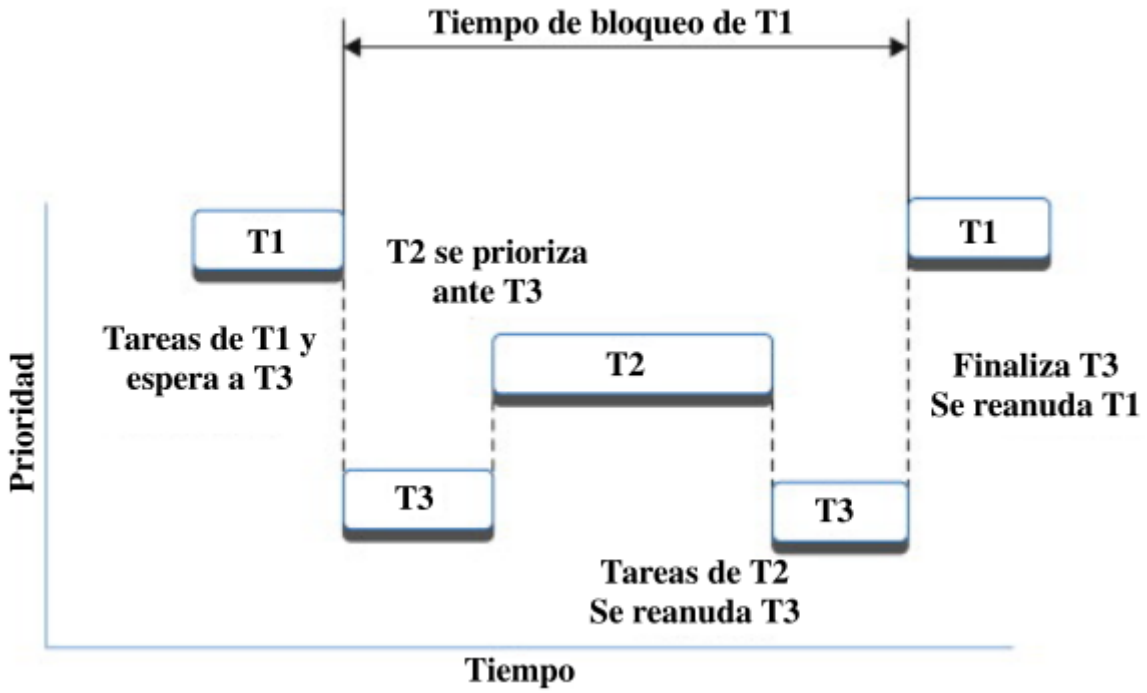


Imagen Federico Coca [Notas sobre ESP32 STEAMakers] CC-BY-SA

Distintas prioridades en tres tareas

Cada tarea tiene su propio espacio de memoria, por lo que crear demasiadas tareas también puede dejarnos el procesador sin memoria. Si la memoria asignada a las tareas tampoco es suficiente para almacenar los datos se podría reiniciar de forma inesperada el Arduino, o funcionar incorrectamente, es decir que como siempre, hay que ser consciente de los limitados recursos de los que disponemos.

Semáforos

Con la introducción teórica a la multitarea vista, debemos hacernos otra pregunta: ¿Qué pasa si una tarea accede a un recurso o variable, y el sistema multitarea le da el control a otra tarea y por tanto ese proceso falla o quizás otra tarea acceda al mismo recurso y se solapen?

Para ese problema de convivencia entre tareas se inventaron los “semáforos”, en concreto el que más nos interesa es el semáforo “mutex” o de exclusión mutua, que permite que bloqueemos el sistema multitarea, hagamos lo que tengamos que hacer crítico, y luego liberemos el control. Por supuesto estas tareas críticas deben ser lo más cortas y atómicas posibles: una escritura crítica en una variable, un envío de un dato, una actualización de una pantalla LCD,... siempre cosas simples. Los semáforos debemos usarlos en casos que tengamos claro que se pueden crear conflictos, pues su abuso puede hacer que el sistema multitarea empiece a fallar.

En la Figura siguiente vemos el esquema de acceso a un recurso desde dos tareas diferentes.

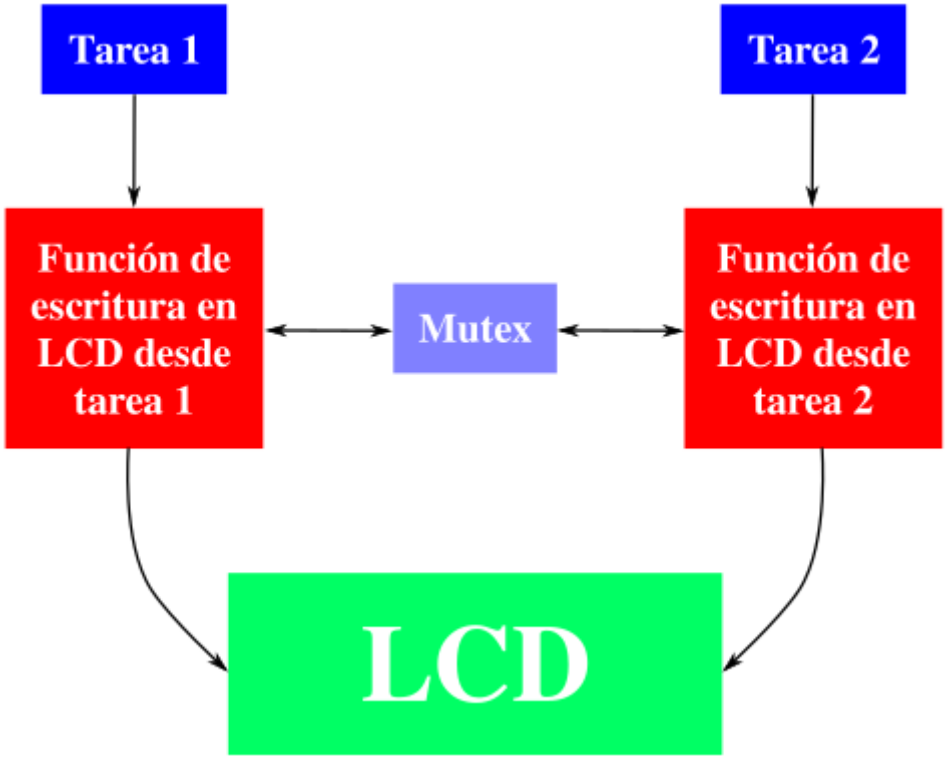


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Esquema de acceso a un mismo recurso por parte de 2 tareas diferentes

En la ESP32 STEAMakers

Una ventaja que posee la ESP32 Plus STEAMakers es que, al estar basado en un ESP32 con dos microcontroladores internos, podemos hacer que cada microcontrolador trabaje en una tarea y esto si sería multitarea real, o en varias tareas mezclando la multitarea real con la simulada. Esto también se puede hacer en las placas Arduino UNO, pero con el ESP32 tenemos más potencia para realizar estas tareas.

Esquema de acceso a un mismo recurso por parte de 2 tareas diferentes

Bloques esperar

¿Qué pasa con los bloques tipo “esperar” que estaban tan prohibidos en la programación de Arduino cuando queríamos simular una multitarea antes de tener estos bloques? Pues seguimos teniéndoles bastante tirria. Aunque en teoría podríamos usarlos, un bloque esperar hace pensar al microcontrolador que está haciendo algo útil, cuando en realidad no es así, por lo que el sistema multitarea querrá asignarle tiempo de procesamiento a la tarea, aunque sea para eso, ¡para no hacer nada!

Tenemos una solución, tenemos un nuevo bloque de esperar “task friendly” que en lugar de esperar sin hacer nada le dice al sistema: ¡voy a estar un rato sin hacer nada, permite ejecutar otras tareas mientras y luego vuelves! ….Mucho más “friendly”, claro que sí.

Bloques

Con toda esta información pasamos a ver los bloques disponibles para poner todo ésto en marcha.

Imagenes Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Bloque	Descripción
Bloque ejecutar tarea	Permite crear una nueva tarea con su bloque de “inicializar” y su “bucle” al igual que la tarea original de Arduino. Debemos asignar una prioridad a cada tarea, por defecto dejaría todas a “baja” y luego iría ajustando si hace falta. Para gestionar mejor las prioridades, es recomendable en algunos casos no utilizar el “inicializar” y “bucle” propio de Arduino que suele tener preferencia sobre todas estas tareas y es más difícil de equilibrar las prioridades.
Bloque esperar en esta tarea	El bloque esperar óptimo para tareas, pues deja funcionar al resto de tareas de forma más óptima mientras se espera en ésta. Este bloque tiene menos precisión que el bloque “esperar” original, si necesitamos hacer esperas muy precisas (o de menos de 20 ms) debemos usar el “esperar” tradicional. Pero nos servirá en la mayoría de casos.
Bloque bloqueo exclusivo	Si tenemos que hacer alguna acción crítica que no queremos que sea interrumpida internamente por el planificador del sistema multitarea podemos poner este bloque y dentro los bloques críticos. (no utilizar si no es estrictamente necesario)
Bloque establecer memoria	Cada tarea tiene su propio espacio de memoria reservado, esta es la cantidad por defecto para las tareas (192 bytes), si necesitamos ajustarla podemos utilizar este bloque en el “inicializar” principal y se ajustará para todas las tareas. Un mal ajuste puede provocar reinicios del microcontrolador o mal funcionamiento.
Bloque y destruir tarea	Las tareas en principio, igual que el bucle de Arduino, están pensadas para ejecutarse de forma indefinida, si en un caso una tarea deja de ser necesaria la forma de terminarla es con este bloque que parará la ejecución y liberará la memoria de la tarea en la que se ejecuta.

En la TdR STEAM

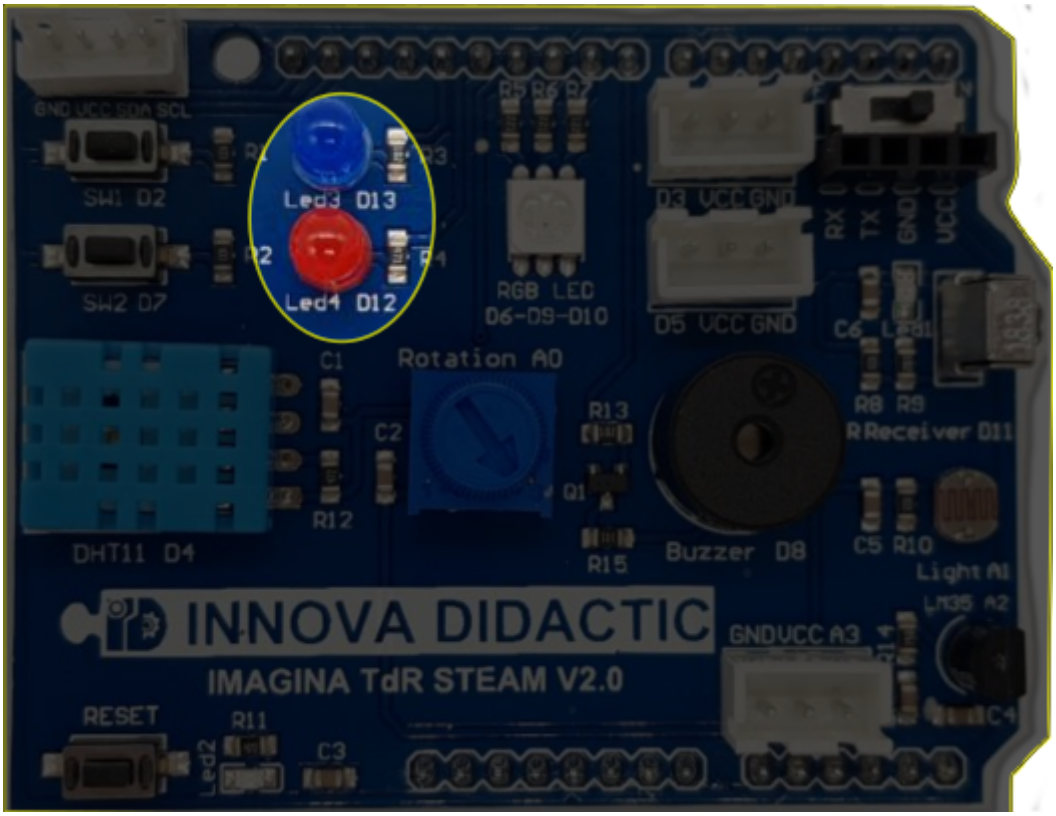


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Programando la Actividad

Vamos a hacer que los dos diodos parpadeen de forma independiente con un programa como el siguiente, que lo tenemos disponible en Actividad-03. Parpadeo intermitente. Multitarea

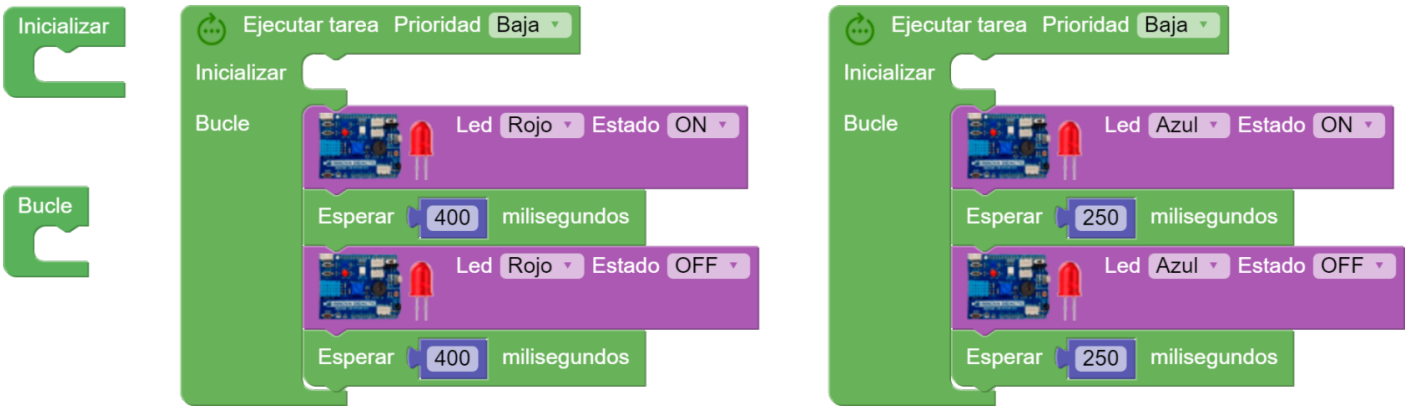


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Retos de ampliación

A3.R1. Hacer que los LEDs rojo y azul trabajen en multitarea modificando los tiempos.

Copyright 2026

Actividad-04. LED RGB

> Página extraída de Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Enunciado

La actividad básics consistirá en ver los colores primarios y su mezcla mediante el uso del LED RGB y también habrá una actividad secundaria sobre el control de intensidad mediante PWM.

Teoría

LED RGB

Un LED RGB es en realidad un encapsulado que incorpora tres diodos LED, uno por cada color fundamental. Los colores primarios en óptica son el rojo, el verde y el azul y la correcta combinación, en términos de intensidad, de ellos originará cualquiera de los colores secundarios. Las siglas RGB son el acrónimo de Red, Green y Blue. En el caso de la TDR-STEAM se utiliza un LED RGB con los tres LEDs unidos por su cátodo o terminal negativo, es decir es un LED RGB de cátodo común. En la imagen siguiente vemos el modelo aditivo de los colores rojo, verde y azul.

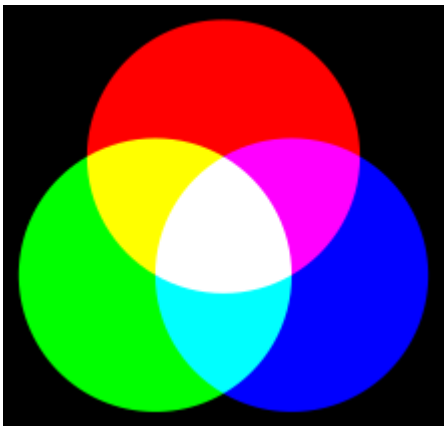


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

El símbolo y la representación mas comunes de este componente lo vemos en la imagen siguiente:

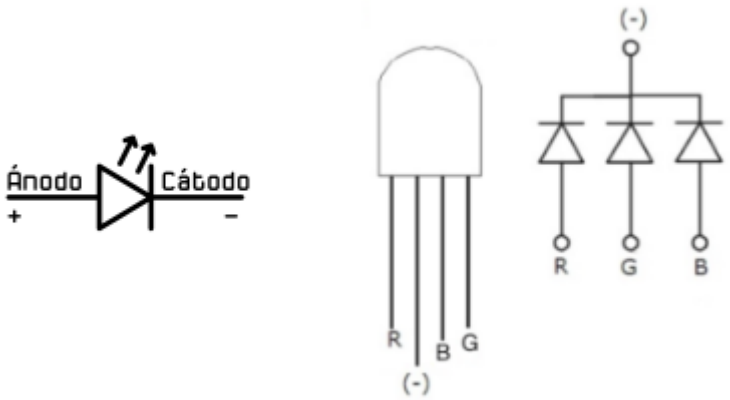


Imagen Federico Coca [Notas sobre ESP32 STEAMakers] CC-BY-SA

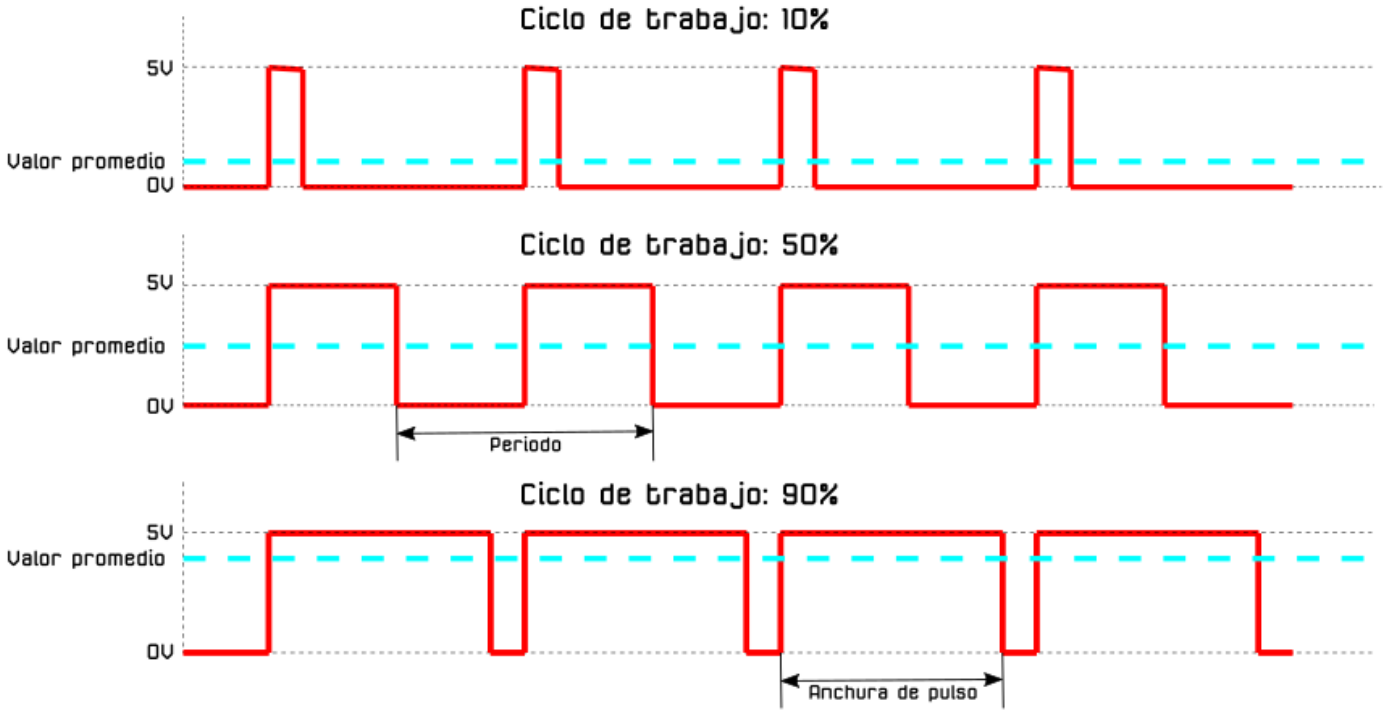
Teóricamente en Arduino, cada uno de esos LEDs podría adoptar 256 (valores entre 0 y 255) colores diferentes, es decir, un total de 16.777.216 posibles colores diferentes con un LED RGB.

PWM

PWM son siglas en inglés que significan Pulse Width Modulation y que lo podemos traducir a español como Modulación de ancho de pulso. Los pines PWM permiten generar una señal analógica mediante una salida digital mapeada con 8 bits, o lo que es lo mismo, valores del 0 al 255, es decir mediante una salida PWM podemos emular una señal analógica.

En realidad una placa tipo UNO no es capaz de generar una salida analógica y lo que se hace es emplear un truco que consiste en activar una salida digital durante un tiempo y el resto del tiempo del ciclo mantenerla desactivada. El valor promedio de la salida es el valor analógico. En el tipo de modulación PWM mantendremos constante la frecuencia, o lo que es lo mismo, el tiempo entre pulsos y lo que se hace es variar la anchura del pulso.

La proporción de tiempo que está encendida la señal, respecto al total del ciclo, se denomina ciclo de trabajo o Duty cycle, y generalmente se expresa en tanto por ciento. En la imagen siguiente vemos señales con distintos ciclos de trabajo.



Distintos Duty cicle Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Es importante recordar que en una salida PWM el valor de la tensión es 5V por lo que si alimentamos un dispositivo de 3V a partir de una salida de 5V lo dañaremos de forma irreversible.

Las señales PWM emula una señal analógica para aplicaciones como variar la luminosidad de un LED y variar la velocidad de motores de corriente continua.

La placa ESP32 Pus STEAMakers tiene muchas salidas PWM, pero en la placa Imagina TDR STEAM sólo se puede controlar por PWM el led RGB (pines 6, 9 y 10) y los tres pines que quedan libres para conectar elementos externos.

En la TdR STEAM

En la placa existe un LED RGB 5050 de 6 pines como el de la imagen siguiente conectado a los pines D6 (Red), D9 (Green) y D10(Blue). Estos tres pines son PWM y nos van a permitir regular su intensidad.

RGB 5050 *Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA*

En la placa TdR STEAM se localiza donde vemos en la imagen siguiente:

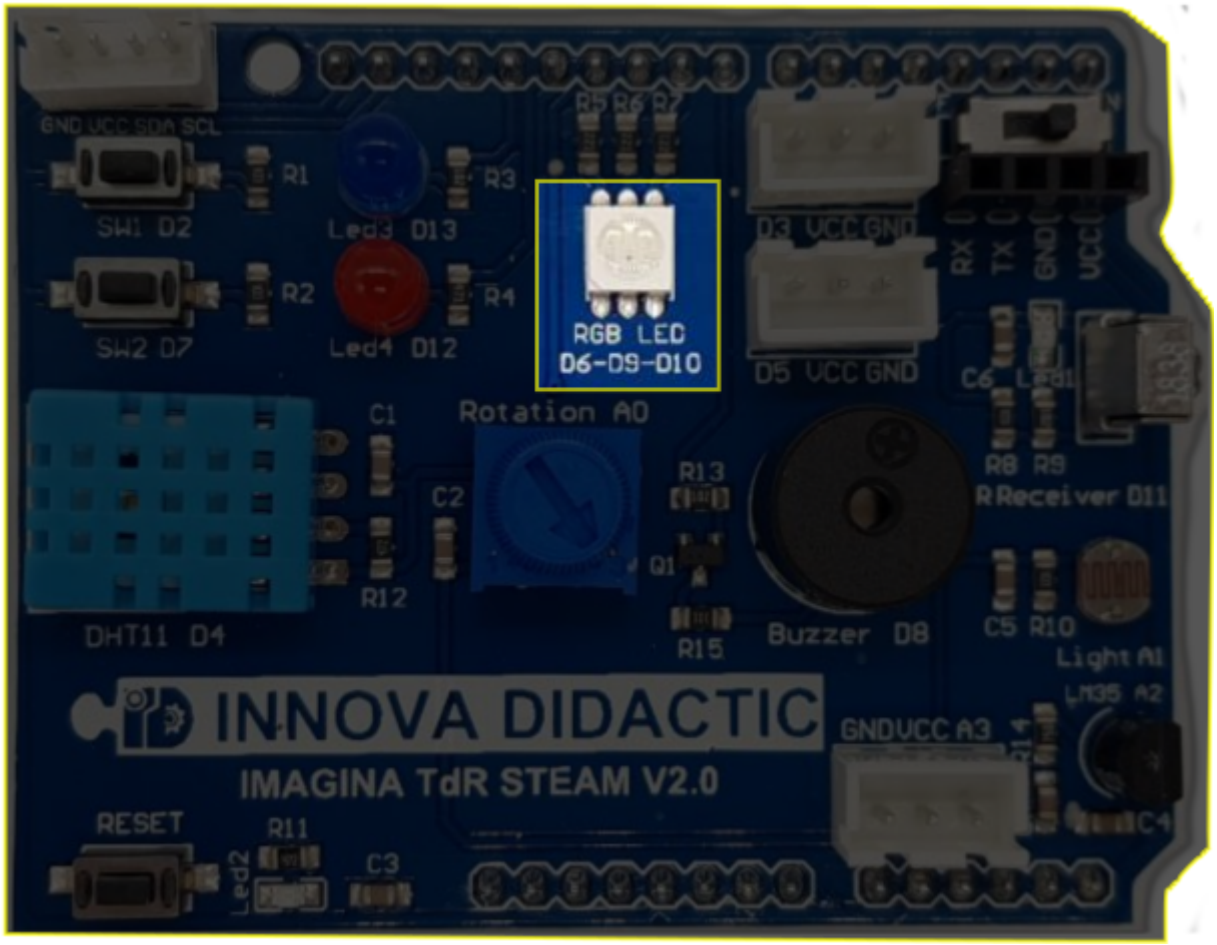


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Programando la actividad

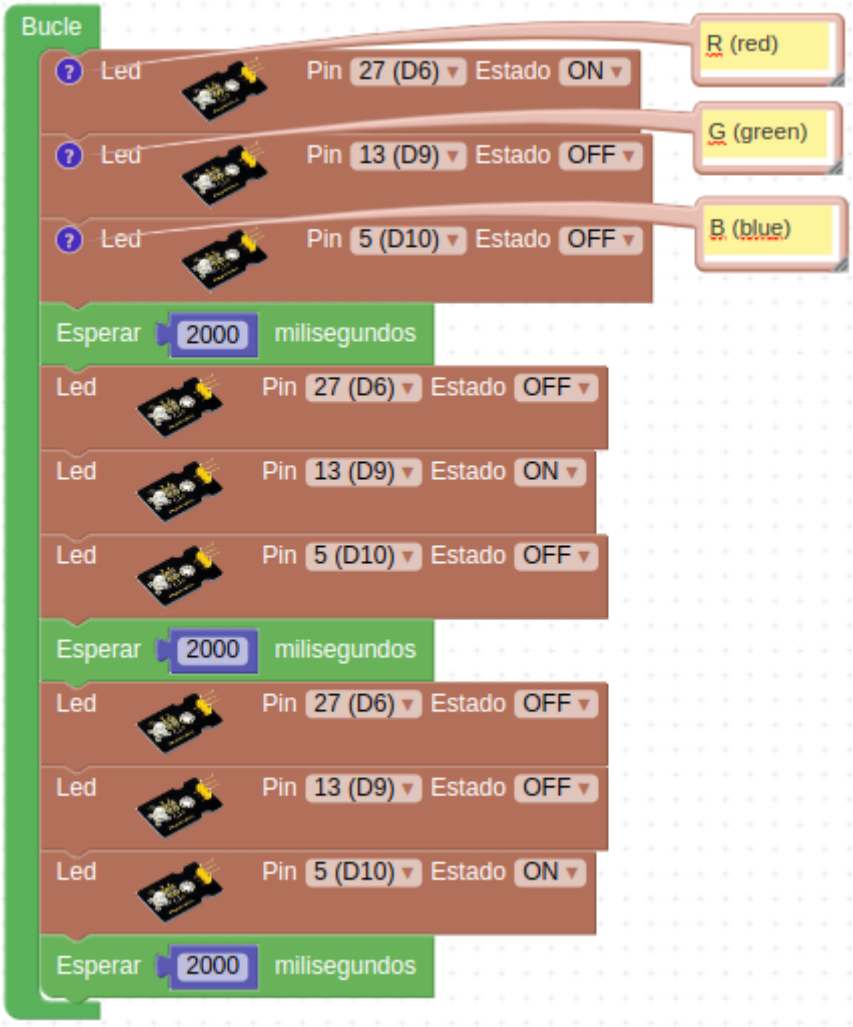
En ArduinoBlocks disponemos de los dos bloques que vemos en la imagen siguiente para el control PWM del LED RGB.

Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

El bloque superior permite asignar el color a partir de la paleta que se despliega al hacer clic sobre el cuadrado de color y en el inferior debemos introducir el valor numérico (entre 0 y 255) correspondiente a cada color primario en cada uno de los tres colores RGB. Con estos bloques no tenemos que preocuparnos por saber las conexiones de cada diodo ya que están asignadas internamente en el bloque.

También hemos visto que el LED RGB tiene asociados tres pines y por tanto podemos tratar a cada LED de forma individual. Pero para poder hacerlo debemos crear un tipo de proyecto "ESP32 STEAMakers" y no como hasta ahora "ESP32 STEAMakers + Imagina TdR STEAM". Esto nos va a permitir disponer del bloque LED que está dentro de "Actuadores" con todos los pines digitales configurables.

De esta forma un programa como el de la imagen siguiente nos va a permitir activar de forma individual cada diodo LED. La solución a esta actividad la tenemos disponible en Actividad-04.



Actividad 04 Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Retos de ampliación

A4.R1. A partir de la idea del reto realizar un programa que muestre sucesivamente los siguientes colores: magenta o violeta, cian o azul claro, amarillo y blanco. Ayúdate del gráfico que muestra el modelo aditivo de colores que hemos puesto al principio de la teoría.

A4.R2. Realizar un programa que nos muestre los tres colores primarios a partir de un proyecto tipo "ESP32 STEAMakers + Imagina TdR STEAM" y el bloque con la paleta de colores.

A4.R3. Realizar un programa que nos muestre los tres colores primarios a partir de un proyecto tipo "ESP32 STEAMakers + Imagina TdR STEAM" y el bloque con el valor numérico de cada color.

A4.R4. Realizar un programa que muestre de manera secuencial los colores del arcoiris en el orden que vemos en la imagen siguiente.

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Solución A4.R2

El Programa es el de la imagen siguiente:

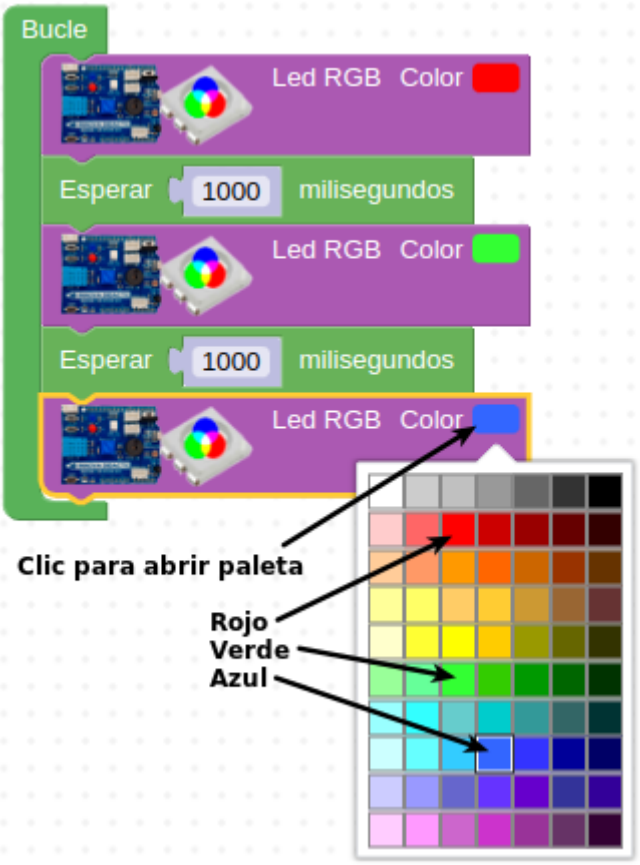


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Solución A4.R3

El Programa es el de la imagen siguiente:

Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Copyright 2026

Actividad-05. Zumbador

> Página extraída de Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Enunciado

Trabajaremos con el buzzer o zumbador partiendo de la reproducción de sonidos básicos hasta llegar a la reproducción de melodías completas.

Teoría

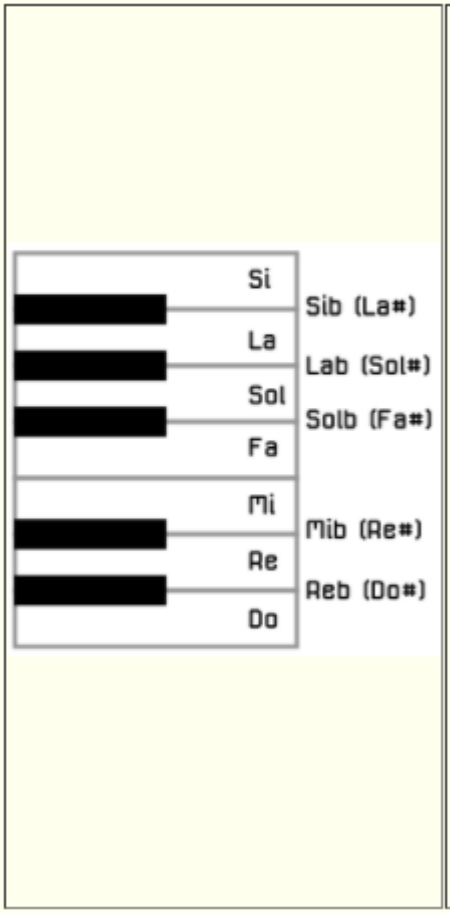
El buzzer, zumbador o altavoz es un transductor electroacústico (convierte una señal eléctrica en una onda de sonido) que produce un determinado zumbido o sonido. Existen de dos tipos:

- Pasivos: no disponen de electrónica interna, por lo que tenemos que proporcionar una señal eléctrica para conseguir el sonido deseado.
- Activos: disponen de un oscilador interno, por lo que únicamente tenemos que alimentar el dispositivo para que se produzca el sonido.

El zumbador que incorpora la placa TdR STEAM es de tipo pasivo y está conectado al pin D8.

Una de los parámetros que caracterizan a un sonido es su frecuencia de emisión, siendo la frecuencia el número de veces que se repite por unidad de tiempo (segundo). La transmisión del sonido se realiza por ondas a través en cualquier medio (sólido, líquido o gaseoso) excepto en el vacío. La frecuencia de un sonido nos indica cuantos ciclos por segundo tiene una onda.

En la imagen y la tabla siguientes vemos un dibujo con un fragmento de las teclas de un piano estando todo referido a una nota estándar, la nota "La" central que tiene una frecuencia de 440 Hz. Podemos ver la nota musical que reproduce, en las dos notaciones más comunes de los sonidos (Inglés: C D E F G A B, Alemán: C D E F G A H, Español, italiano y francés: Do Re Mi Fa Sol La Si) y además se encuentra la frecuencia que produce esa nota musical.



Si	B	493.88
La # (Si b)	A# (Bb)	466.16
La	A	440.00
Sol # (La b)	G# (Hb)	415.30
Sol	G	392.00
Fa # (Sol b)	F# (Gb)	369.99
Fa	F	349.23
Mi	E	329.63
Re # (Mi b)	D# (Eb)	311.13
Re	D	293.66
Do # (Re b)	C# (Db)	277.18
Do	C	261.63

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Bloques de programación

En ArduinoBlocks disponemos de un bloque que nos permite reproducir cualquier melodía RTTTL (del inglés, Ring Tone Text Transfer Language) y es el que vemos en la imagen siguiente:

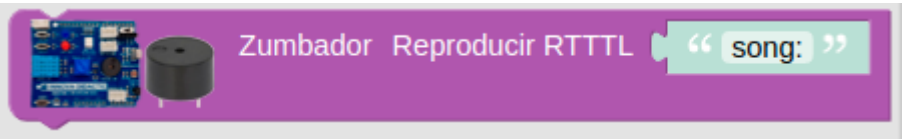


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Este bloque permite reproducir una melodía a partir de un texto con formato RTTTL, formato desarrollado por Nokia para ser usado para transferir tonos de llamada a teléfonos móviles. El formato RTTTL es una cadena dividida en tres secciones: nombre, valor predeterminado y datos. Por ejemplo, la siguiente cadena de texto se corresponde con la Intro de Donkey Kong:

Lo único que tenemos que hacer para reproducir la melodía es pegar esta cadena en la zona de texto del bloque.

```
d=4,o=5,b=140:8a#,8p,8d6,16p,16f.6,16g.6,16f.6,8a#,8p,8d6,16p,16f.6,16g.6,16f.6,8a#,8p,8d6,16p
,16f.6,16g.6,16f.6,8a#,8p,8d6,16p,16f.6,16g.6,16f.6
```

ArduinoBlocks nos suministra información y enlaces referentes al tema accediendo desde nuestro pr n siguiente:

Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

En la imagen siguiente tenemos desplegada la información que nos ofrece esta herramienta.

Información RTTTL *Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA*

También podemos acceder a esta información haciendo clic derecho sobre el bloque y escogiendo la opción "Ayuda" de entre las mostradas en la ventana emergente.

Zumbador activo

Existe otro tipo de zumbador que incluye un oscilador que genera una frecuencia audible fija y que se conoce como zumbador pasivo y en realidad es el que de forma correcta se puede denominar como zumbador. Este es mucho mas sencillo de usar ya que basta con conectarlo a un pin digital y cuando pongamos a nivel alto este pin el zumbador generará su zumbido característico durante el tiempo que establezcamos. Con este tipo de zumbador no se pueden generar melodías.

Cuando está nuevo se distingue del zumbador pasivo o altavoz por la pegatina de protección que lo acompaña, pero esta hay que quitarla para oir el zumbido y sin ella es difícil distinguir un tipo del otro, así que en este caso deberemos crear un programa con alguna melodía y si no se reproduce correctamente es que tenemos conectado el zumbador activo en lugar del pasivo. Lo mejor es marcar alguno de los dos cuando podemos distinguirlos. Una buena idea puede ser utilizar la propia pegatina en el lateral del mismo.

En la TdR STEAM

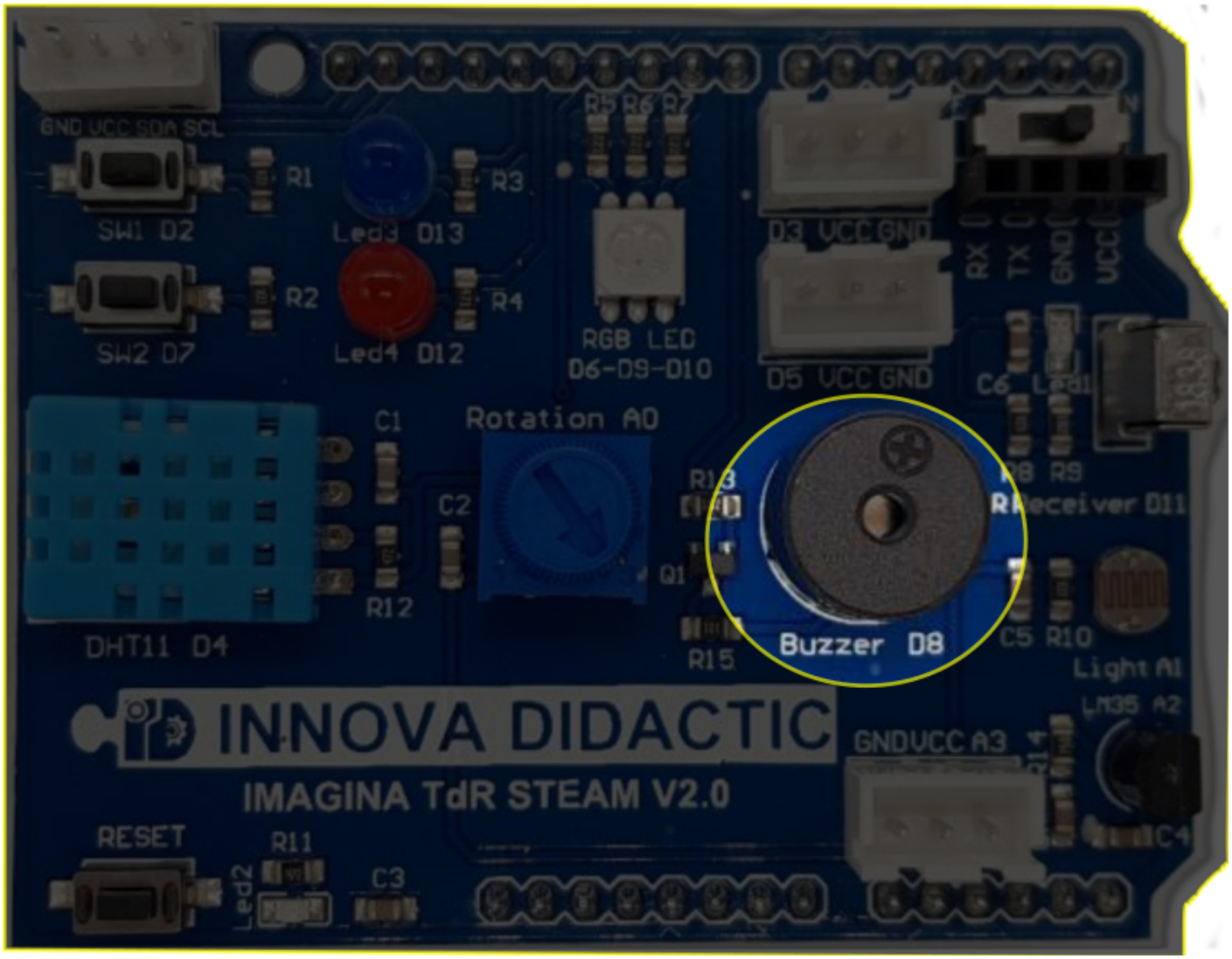


Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

Programando la actividad

Los cuatro bloques destinados a trabajar con el zumbador los vemos en la imagen siguiente:

Imagen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA

En el bloque Zumbador podemos modificar dos parámetros, el tiempo que dura cada sonido expresado en milisegundos (campo Ms) y la frecuencia en Hz a la que reproducirá el sonido el zumbador (campo Hz).

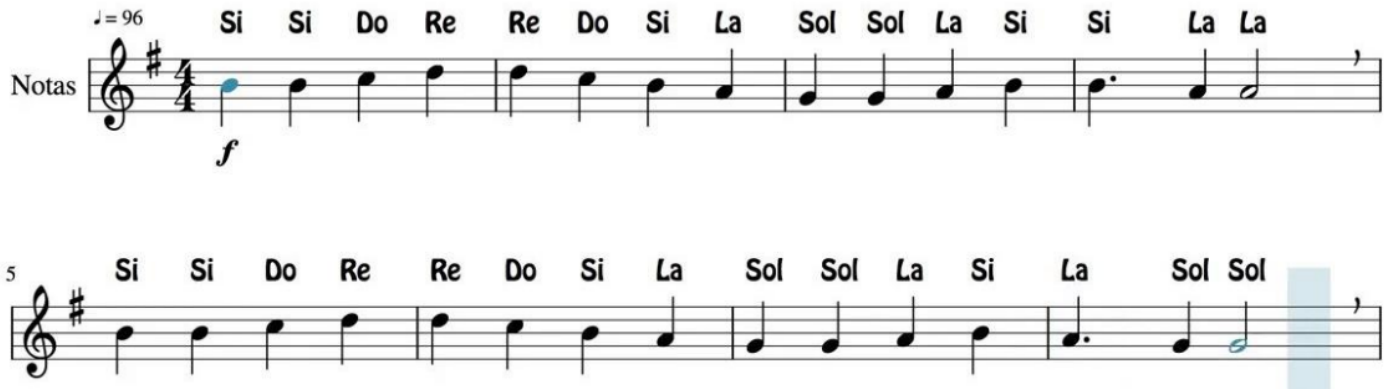
Vamos a comenzar haciendo un programa que reproduzca tres de las notas de la escala musical vista anteriormente. La solución al reto la tenemos disponible en [Actividad-05](#).

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Retos de ampliación

A5.R1. Reproducir la escala musical con las notas básicas utilizando los bloques Zumbador y Tono

A5.R2. Reproducir la melodía de la imagen siguiente sabiendo que las negras tienen una duración de 500ms, las negras con un puntito 750ms y las blancas 1000ms.



Himno a la alegría Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

A5.R3. Reproducir diferentes melodías a partir de los bloques RTTTL (Ring Tone Text Transfer Language o lenguaje de tonos de llamada).

A5.R4. Reproducir alguna otra melodía que descarguemos de las páginas web propuestas.

Actividad-06. Pulsadores

> Página extraída de Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

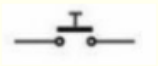
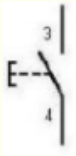
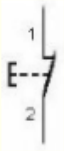
Enunciado

Utilizaremos uno, o ambos pulsadores, para llevar a cabo determinadas tareas como respuesta al accionamiento de los mismos.

Teoría

El pulsador

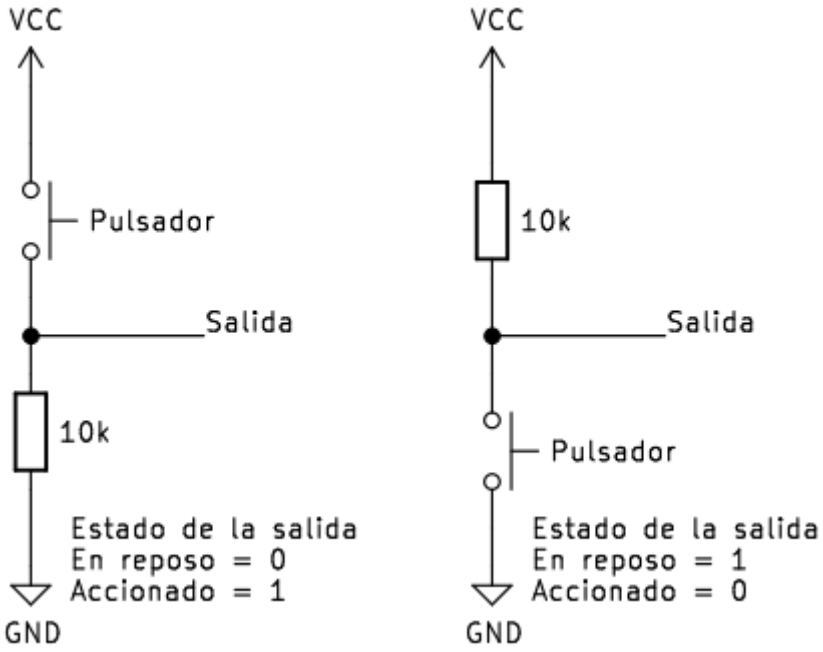
En la tabla siguiente vemos la simbología y algunos ejemplos del aspecto físico de estos elementos.

Tipo	Símbolo americano	Símbolo europeo	Aspecto
Normalmente abierto (NO)			
Normalmente cerrado (NC)			

Símbolos y aspecto real de un pulsador *Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA*

Se trata de un dispositivo que es capaz de abrir o cerrar el paso de la corriente eléctrica mientras permanece accionado, es decir, su función no queda anclada como por ejemplo en los interruptores de la luz de casa, en donde el accionamiento permanece hasta que no se vuelve a accionar. En un pulsador, por contra, su activación dura mientras lo mantenemos pulsado y vuelve a su estado de reposo en cuanto dejemos de pulsarlo.

Mediante la configuración adecuada podemos convertir un pulsador en un elemento de entrada a algún pin de nuestra placa UNO. Las configuraciones más básicas posibles con pulsadores las podemos ver en la imagen siguiente.



Configuración circuito elemental con pulsador *Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA*

> *Esto lo hemos dado en **Sensores**, son dos configuraciones: la primera PULL DOWN o lógica normal y la segunda PULL UP o lógica inversa. Los pulsadores Tdr Steam son PULL DOWN o lógica normal.*

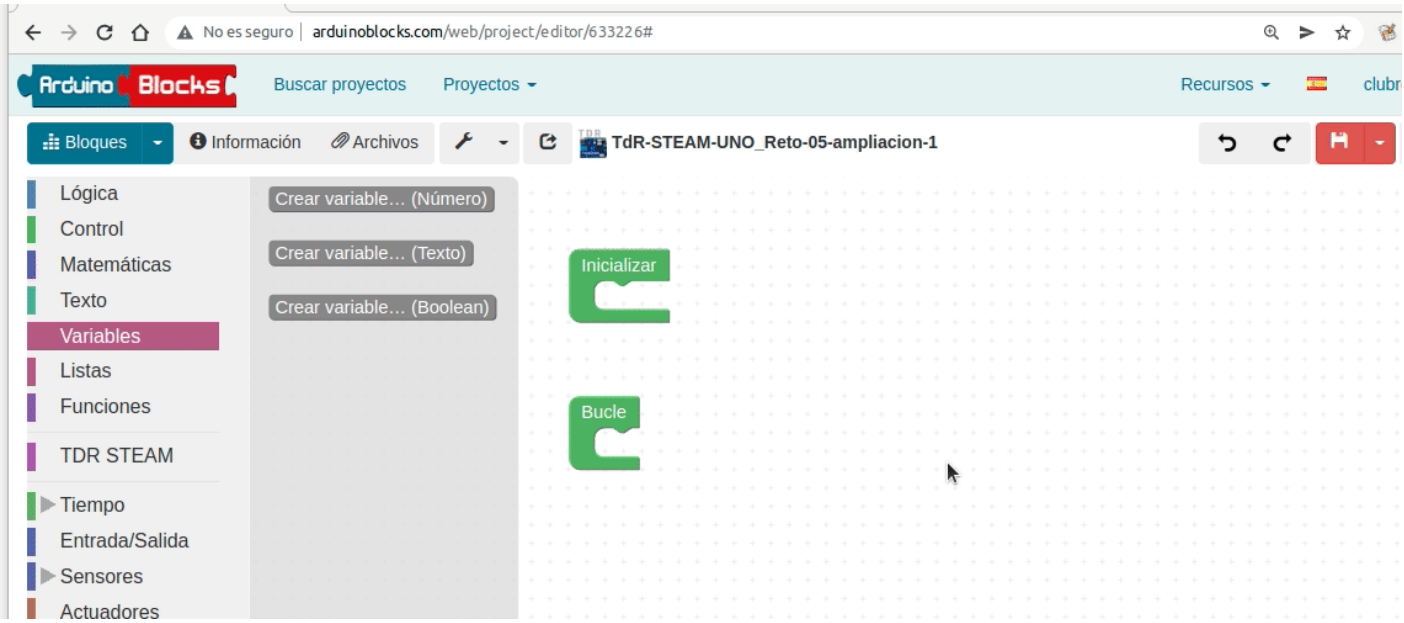
En el caso de la TdR STEAM los pulsadores se han configurado para que en reposo pongan a cero su entrada digital correspondiente y que se ponga a uno cuando son accionados.

Concepto de variable y de contador

El concepto de variable en programación consiste simplemente en asignarle un nombre significativo a un espacio de memoria donde almacenar determinada informa

ción durante la ejecución normal del programa. El concepto es muy amplio y complejo y en nuestro caso no vamos a entrar en detalles sobre el mismo, pero si indicar que no se debe confundir con el concepto de variable matemática, ya que una expresión como x = x + 1 que es una aberración en matemáticas tiene todo el sentido en programación. Lógicamente en matemáticas no se puede cumplir pero en programación significa que a la variable x se le sume uno y el resultado se vuelva a guardar en la misma variable.

En ArduinoBlocks podemos crear tres tipos de variables, numéricas, de texto o booleanas. En la animación siguiente podemos ver como se pueden crear, eliminar y renombrar variables.



Crear, renombrar y eliminar variables *Imagen Federico Coca* [*Notas sobre ESP32 STEAMakers*] *CC-BY-SA*

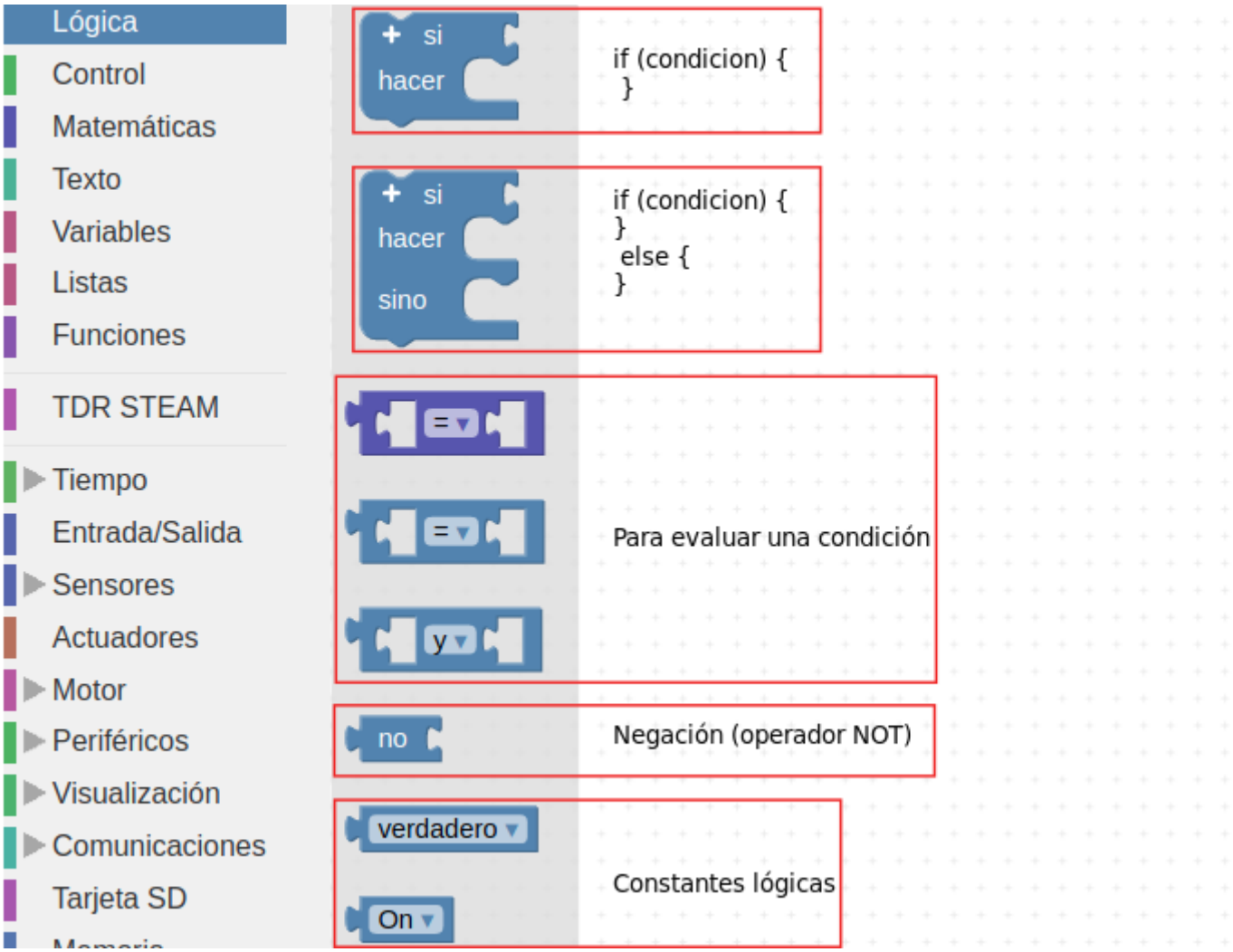
En programación, llamamos contador a una variable cuyo valor se incrementa o decrementa en un valor fijo para cada iteración del bucle para el que se ha definido. El uso habitual de un contador es simplemente contar el número de veces que itera un bucle en general o de forma mas extensa contar, solamente, aquellas iteraciones en las que se cumpla una determinada condición.

Por ejemplo, supongamos que tenemos una variable de nombre Estado de valor inicial cero y que se incremente cada vez que accionamos un pulsador, de esta forma si en un bucle vamos incrementando la variable de uno en uno, tenemos:

- Estado = 0 // valor inicial
- Estado = 1 // Estado = Estado + 1
- Estado = 2 // Estado = Estado + 1
- ...

Condicionales

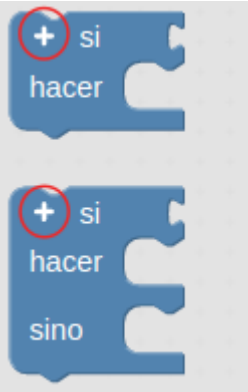
Las sentencias condicionales son aquellas que nos permiten tomar decisiones en función de si ocurre o no ocurre determinada cosa. En el caso de ArduinoBlocks estas las podemos encontrar en el bloque lógica. Este bloque contiene los elementos que vemos en la imagen siguiente:



Bloques de lógica *Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA*

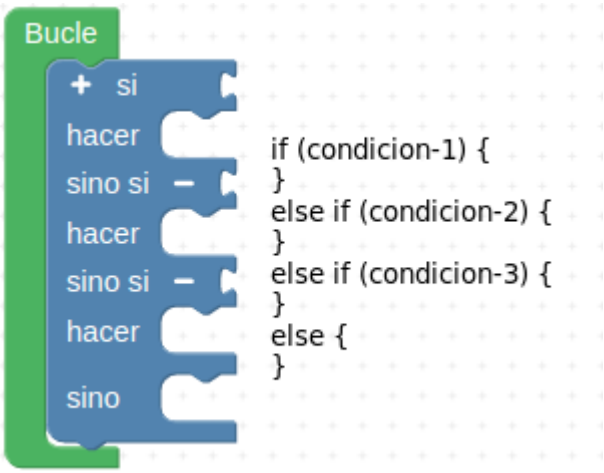
El funcionamiento es el siguiente: se evalúa la condición que ponemos en "si" y si el resultado es verdadero, o sea condición cierta, se realizan las acciones que pongamos en "hacer" y si no es cierta dichas acciones no se realizan. En el apartado condición se pueden poner infinidad de factores como pueden ser el estado de sensores, realizar comparaciones, hacer operaciones matemáticas, etc. Lógicamente el resultado de evaluar la condición debe ser verdadero o falso lo que se resuelve en el caso de la programación en Arduino diciendo que si el resultado es 0, el resultado de la evaluación es falso y si es 1 o cualquier otro valor es verdadero.

Si hemos sido observadores hemos visto en los bloques "si" de "Lógica" un signo mas (+) en la parte superior izquierda tanto del condicional "if" como la del "if ... else" tal y como se destaca en la imagen siguiente.



Añadir opciones else if Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Cada vez que pulsemos sobre el signo mas se añadirá una cláusula "else if" a la principal y podemos añadir tantas como necesitemos. Esta cláusula nos va a permitir establecer una nueva condición.En la imagen siguiente vemos añadidas dos para el caso de "if ... else". En la imagen observamos el código equivalente y las distintas condiciones que se pueden establecer.



Clausula if...else con dos else if *Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA*

El signo menos (-) que aparece en la imagen sirve para eliminar la correspondiente cláusula "else if".

En la TdR STEAM

La placa TdR STEAM dispone de dos pulsadores denominados SW1 y SW2 y conectados a los pines digitales D2 y D7 respectivamente.

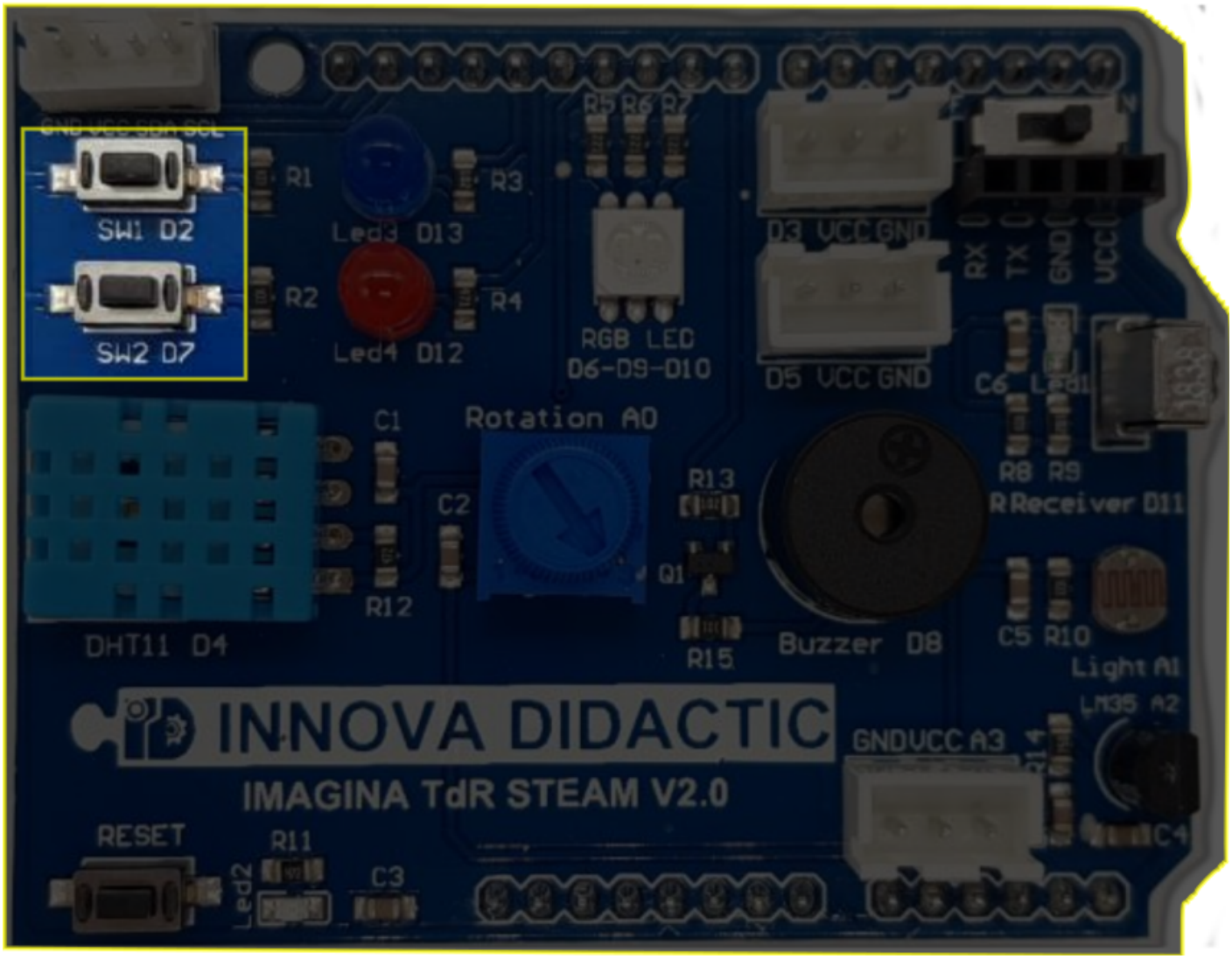


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Programando la actividad

La aplicación de un pulsador para hacer algo requiere saber si este esta pulsado o no lo está y para ello vamos a necesitar de las sentencias condicionales que hemos visto anteriormente.

Vamos a hacer un programa en el que preguntemos si el pulsador SW1 (D2) está o no pulsado y si lo está que se encienda el LED rojo (D12) y si no lo pulsamos que permanezca apagado. La solución al reto la tenemos disponible en Actividad-06. A modo de comentario se ha colocado otra forma de realizar la misma programación que podemos probar simplemente arrastrando el bloque que hay en bucle fuera del mismo y colocando el otro en su lugar.

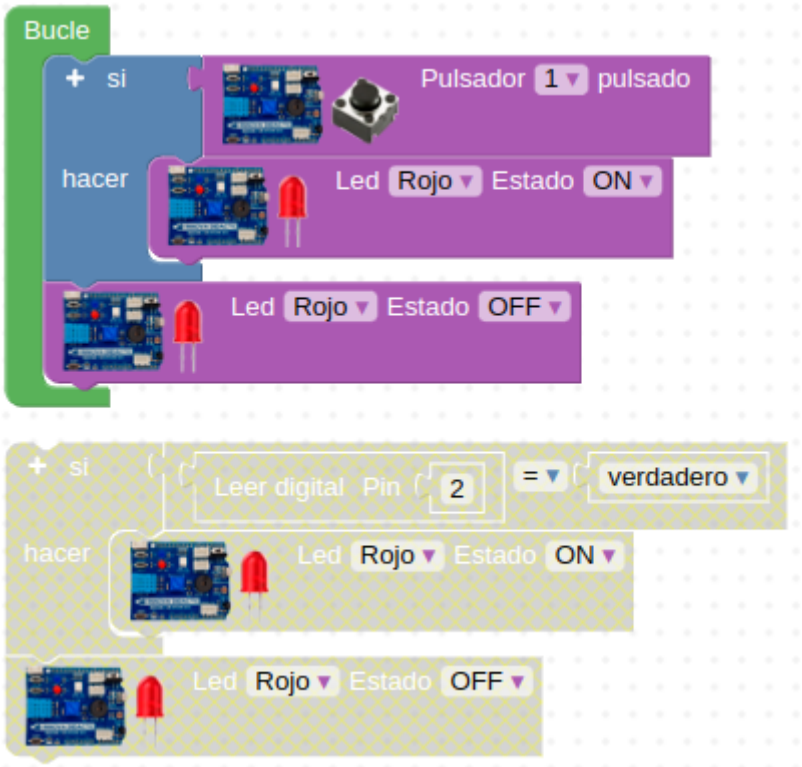
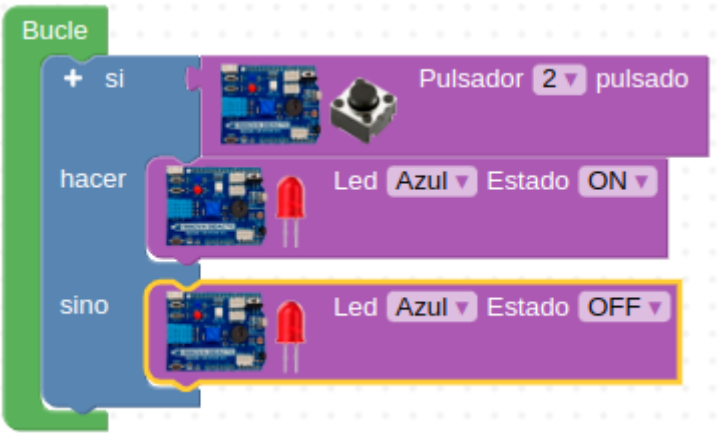


Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Ampliación sobre programación de la actividad

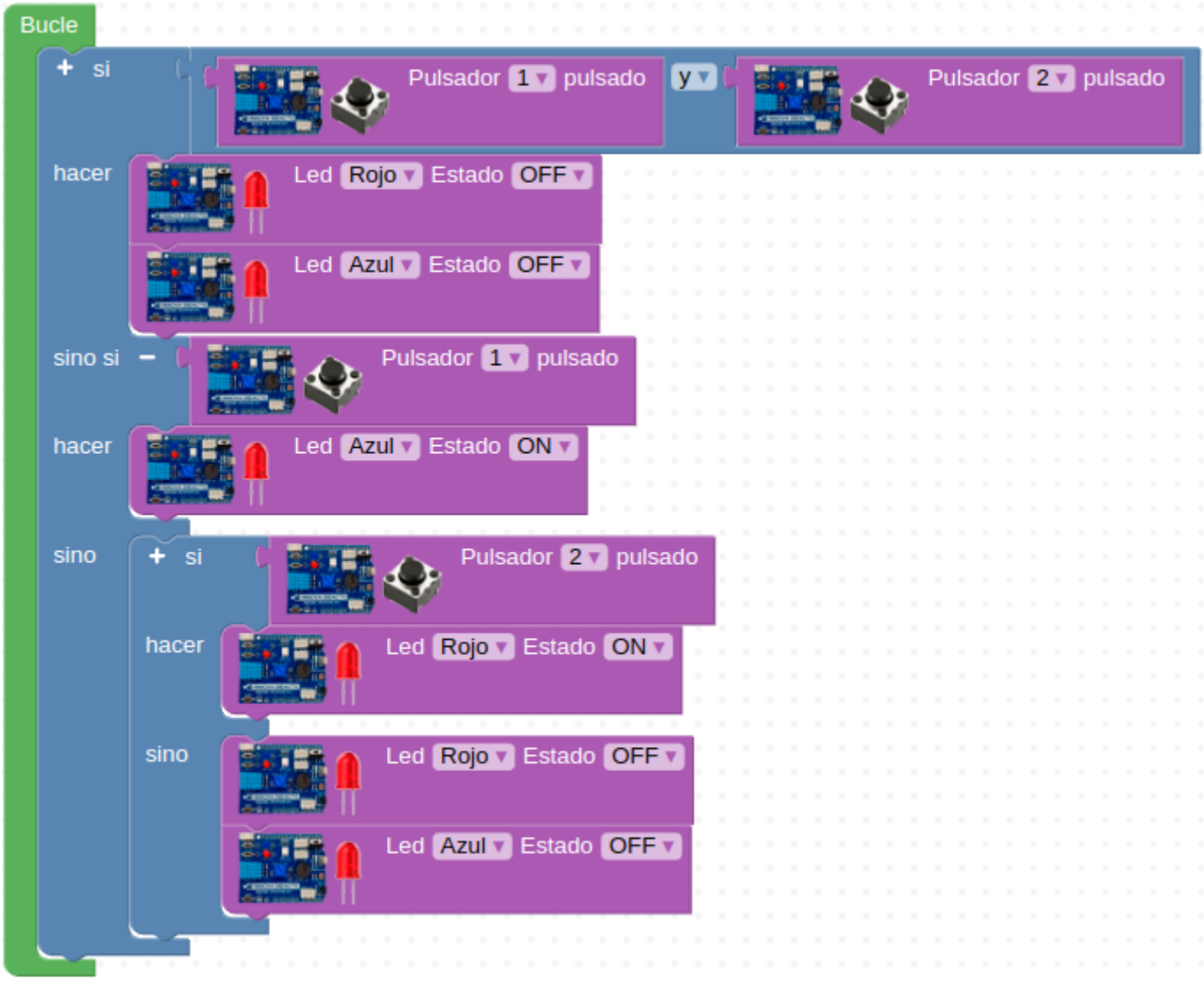
Ya se ha explicado el funcionamiento del signo mas (+) de la parte superior izquierda tanto del condicional "if" como la del "if ... else". Cada vez que pulsemos sobre el signo mas se añadirá una cláusula "else if" a la principal y podemos añadir tantas como necesitemos. Esta cláusula nos va a permitir establecer una nueva condición. El signo menos (-) que aparece en la imagen sirve para eliminar la correspondiente cláusula "else if".

Vamos a hace un programa similar a la Actividad-06 para hacer que si pulsamos SW2 se encienda el LED azul y si no está pulsado permanecerá apagado. La solución la tenemos disponible en Actividad-06-ampliación-1.



Actividad-06-ampliacion-1 Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

El siguiente programa hará que se encienda el LED azul al pulsar SW1 y el LED rojo al pulsar SW2 se encienda el rojo permaneciendo apagados si no están pulsados. Observese la primera condición AND (Y) que impide que si pulsamos ambos pulsadores al mismo tiempo se enciendan los LEDs. La solución la tenemos disponible en Actividad-06-ampliación-2.



Actividad-06-ampliacion-12 Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Retos de ampliación

A6.R1. Resolver el reto inicial de forma que el funcionamiento sea al contrario, es decir, que el LED rojo esté siempre encendido y al pulsar SW1 se apague.

A6.R2. Hacer un programa que al pulsar SW1 se encienda el LED azul y que al pulsar SW2 se apague. Como ampliación se sugiere modificar el programa para que encienda y apague los dos LEDs a un tiempo.

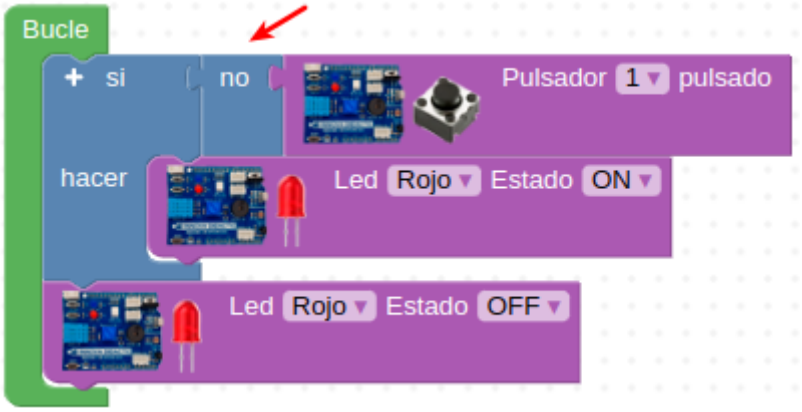
Copyright 2026

A6.R3. Hacer un programa que emita, usando SW1 como si fuese un telégrafo, el código Morse universal de solicitud de socorro, SOS.

A6.R4. Hacer un programa que al pulsar SW1 encienda el LED azul y que este permanezca encendido hasta que no pulsemos SW1 dos veces mas (3 pulsaciones en total), en cuyo caso se apagará.

Solución A6.R1

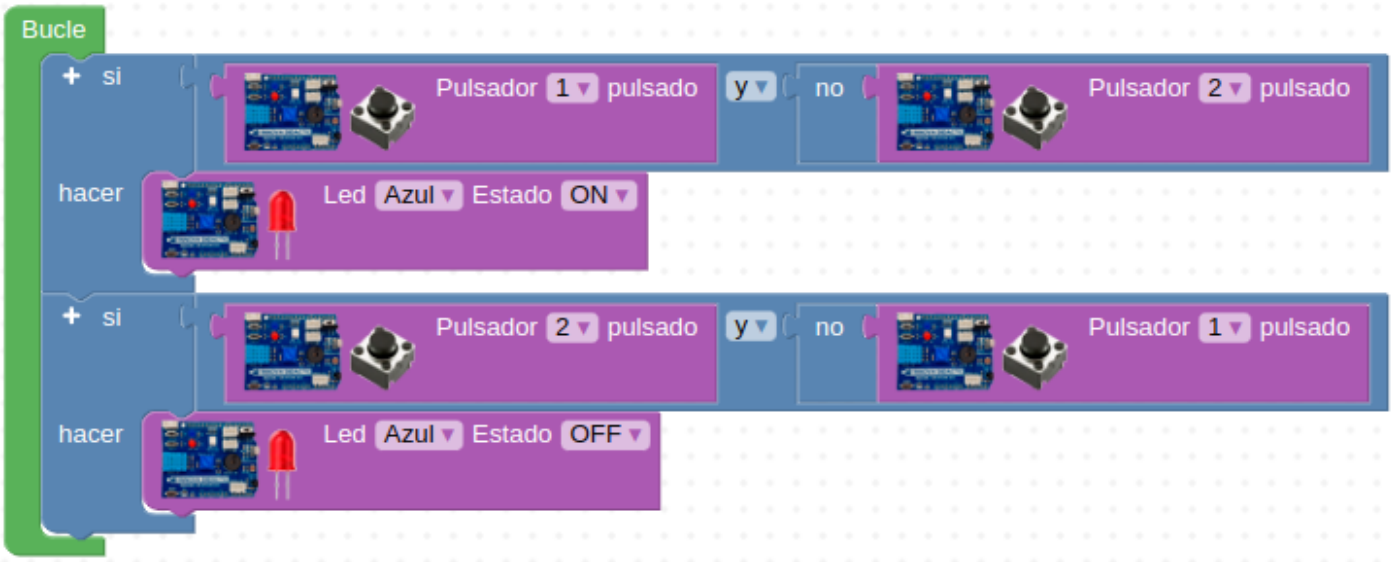
La solución está en utilizar el operador NOT. El Programa es el de la imagen siguiente:



Reto 1 de la actividad 6 *Imagen Federico Coca* *Notas sobre ESP32 STEAMakers* *CC-BY-SA*

Solución A6.R2

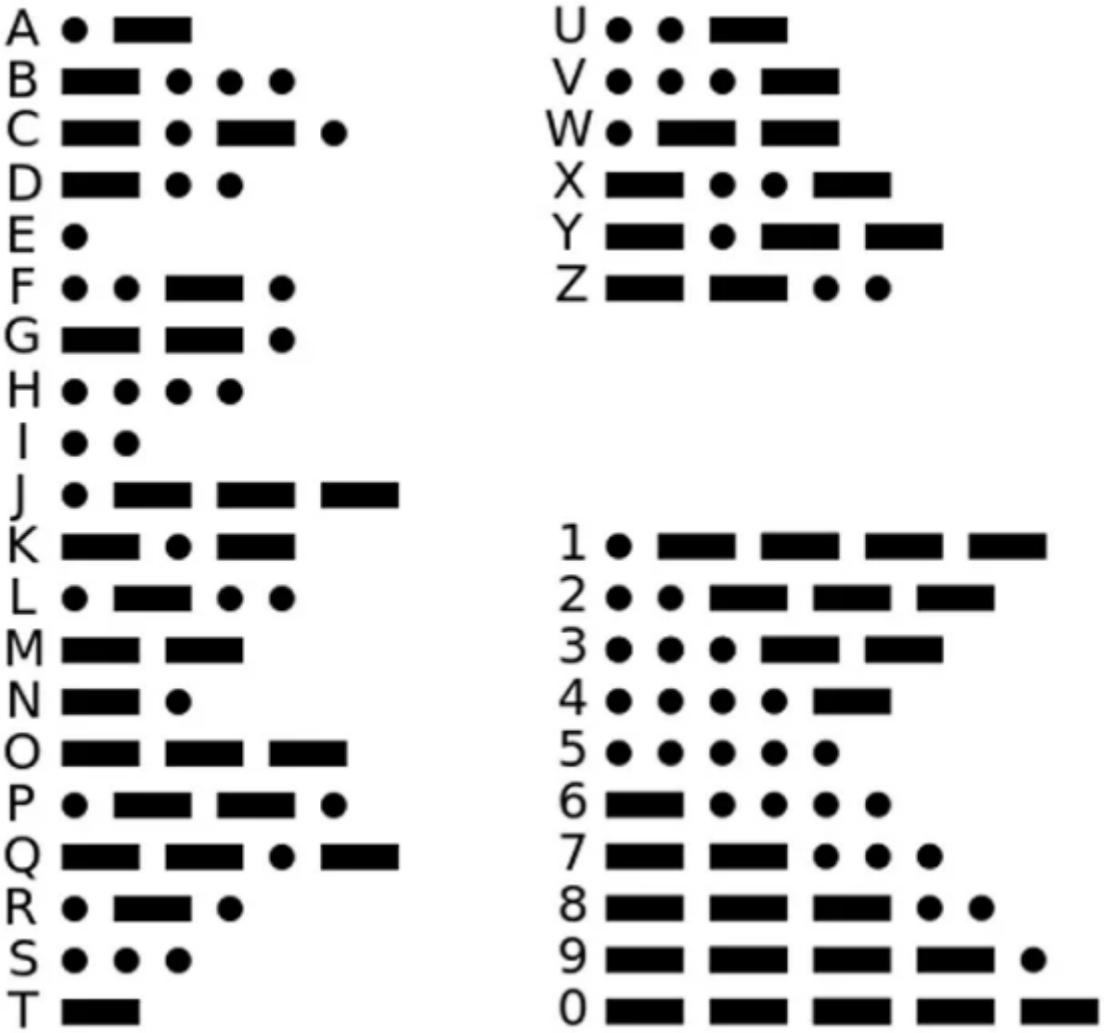
La solución en esta ocasión está en utilizar el operador NOT y el operador AND. El Programa es el de la imagen siguiente:



Reto 2 de la actividad 6 *Imagen Federico Coca* [Notas sobre ESP32 STEAMakers] *CC-BY-SA*

Solución A6.R3

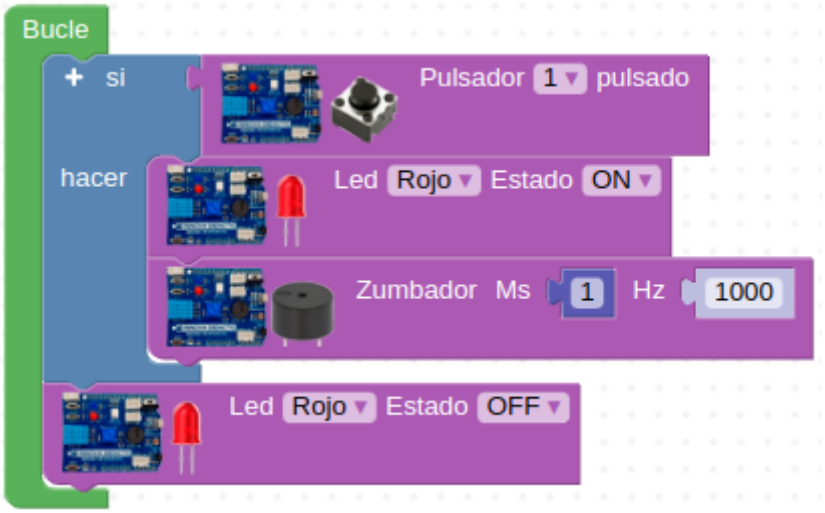
El alfabeto Morse lo vemos en la imagen siguiente, donde podemos observar que la S son tres puntos o pulsaciones cortas y la O son tres rayas o pulsaciones largas. Sin entrar en mas detalles daremos la solución considerando SOS como una palabra (es lo universalmente adoptado) de forma que no haremos pausas entre letras.



Alfabeto Morse *Imagen Federico Coca* [Notas sobre ESP32 STEAMakers] *CC-BY-SA*

Solución A6.R3

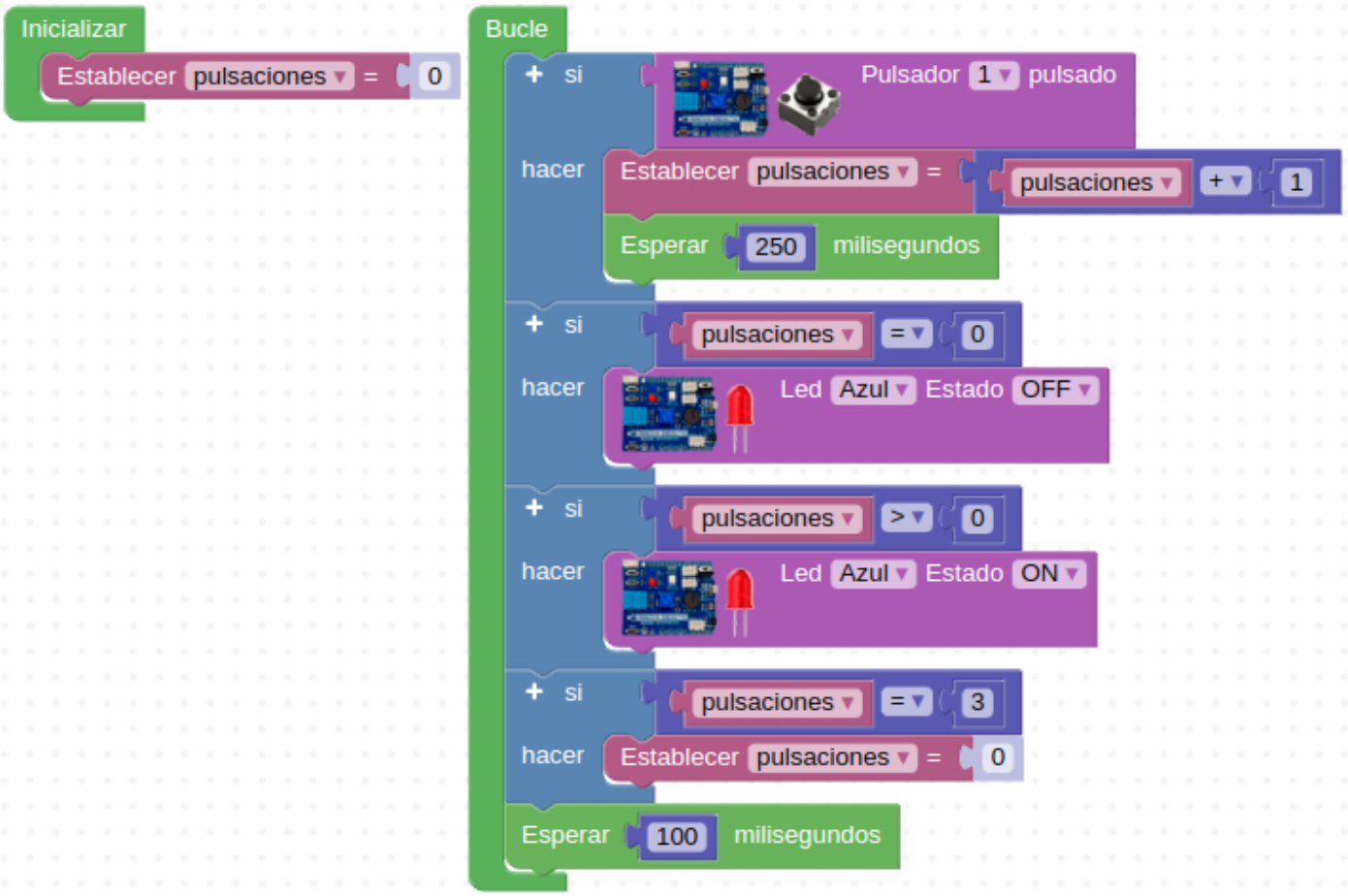
El Programa es el de la imagen siguiente:



Reto 3 de la actividad 6 I*magen Federico Coca [Notas sobre ESP32 STEAMakers](#) CC-BY-SA*

Solución A6.R4

Definimos una variable y la utilizamos para resolver el programa mediante un contador. El Programa es el de la imagen siguiente:



Reto 4 de la actividad 6 I*magen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA*

Copyright 2026