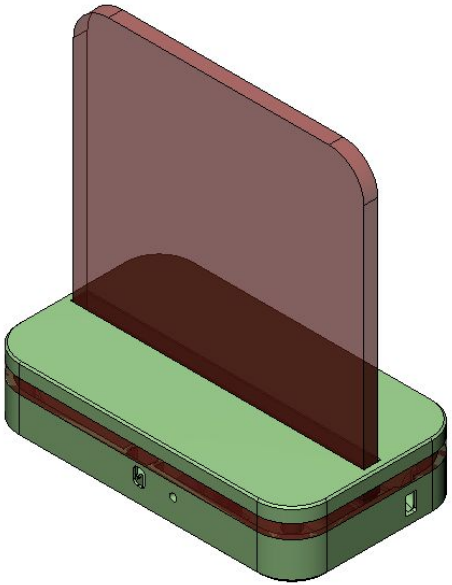
Cartel para distanciamiento social

Cartel de distanciamiento



Un sensor TOF (VL53L0x) lee la distancia del usuario, y modifica el color de leds inteligentes dependiendo de la zonas en la que se encuentre (peligro/precaucion/seguro).

Después de una condición de reset, el operador puede calibrar la distancia del punto de peligro, que además se usa para definir las dos zonas restantes que están separadas por 20 cm.

Mientras el equipo está controlando, se puede usar el pulsador para activar/desactivar el buzzer cuando el usuario está en la zona de peligro.

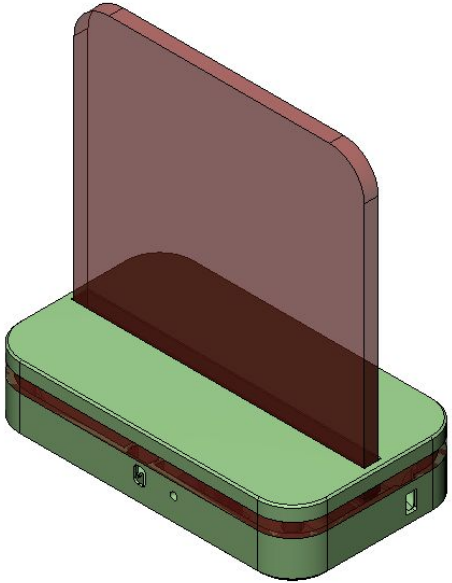
Para que la indicación de peligro desaparezca, el usuario tiene que salir de la zona por más de 1 segundo.

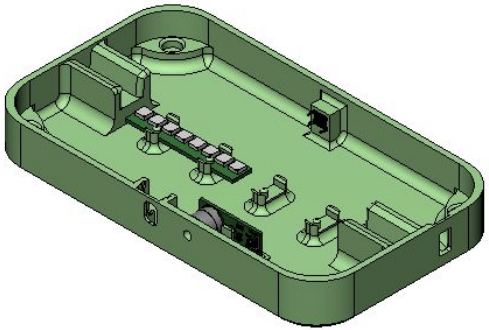
https://circuitdigest.com/news/breaking-barriers-espressifs-esp32-c5-brings-5-ghz-wi-fi-to-iot-devices

team

Gabineteria

Cartel de distanciamiento



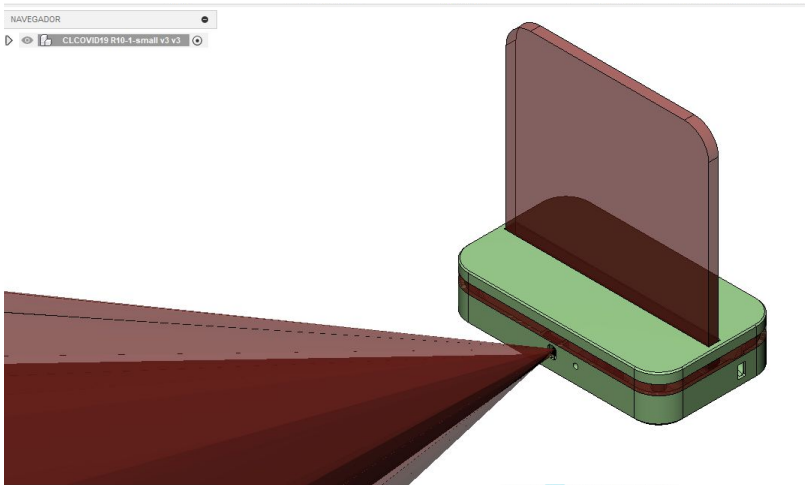


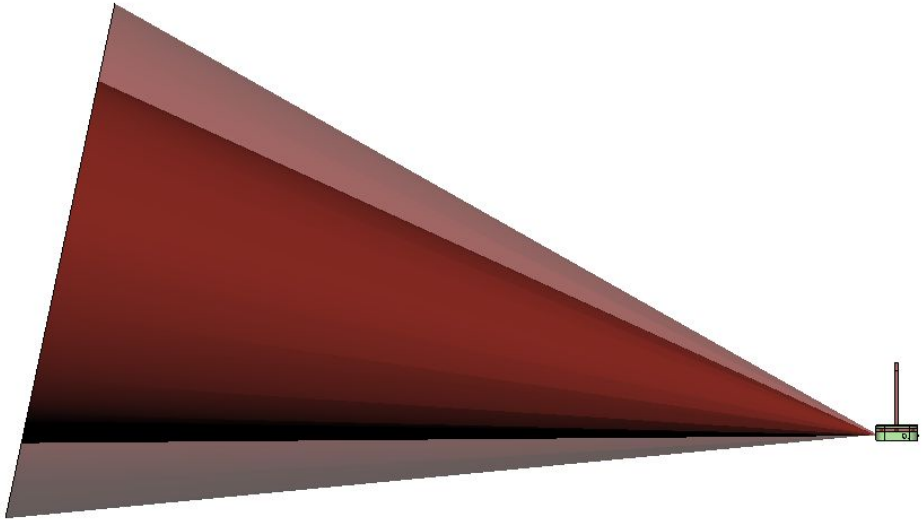
Fusion

team

Gabineteria

Cartel de distanciamiento

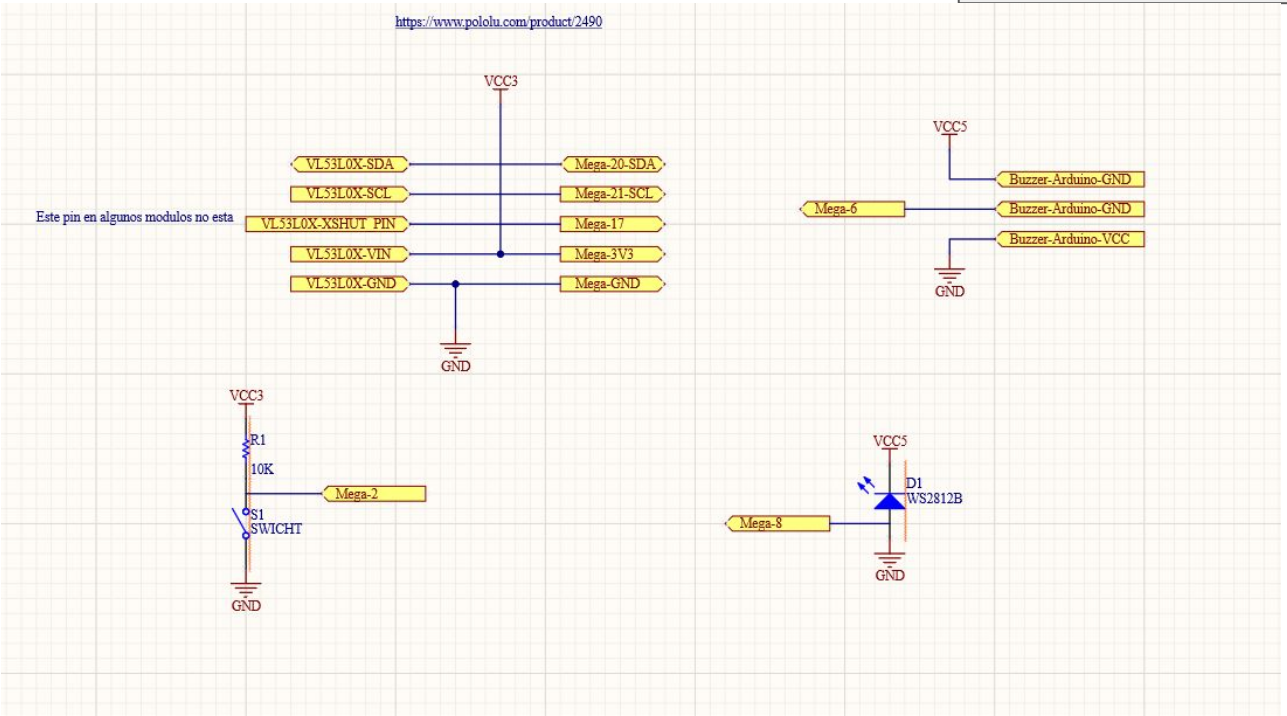




Fusion

Electrónica

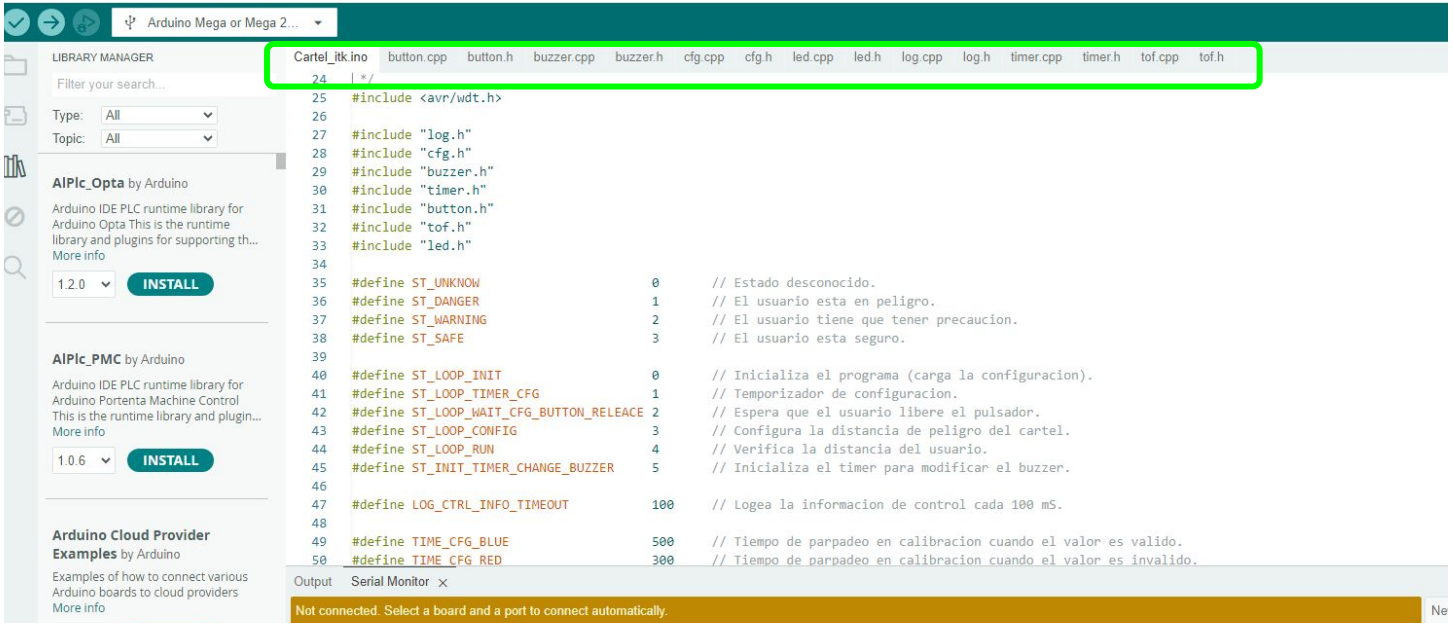
Cartel de distanciamiento



Altium

Firmware

Cartel de distanciamiento



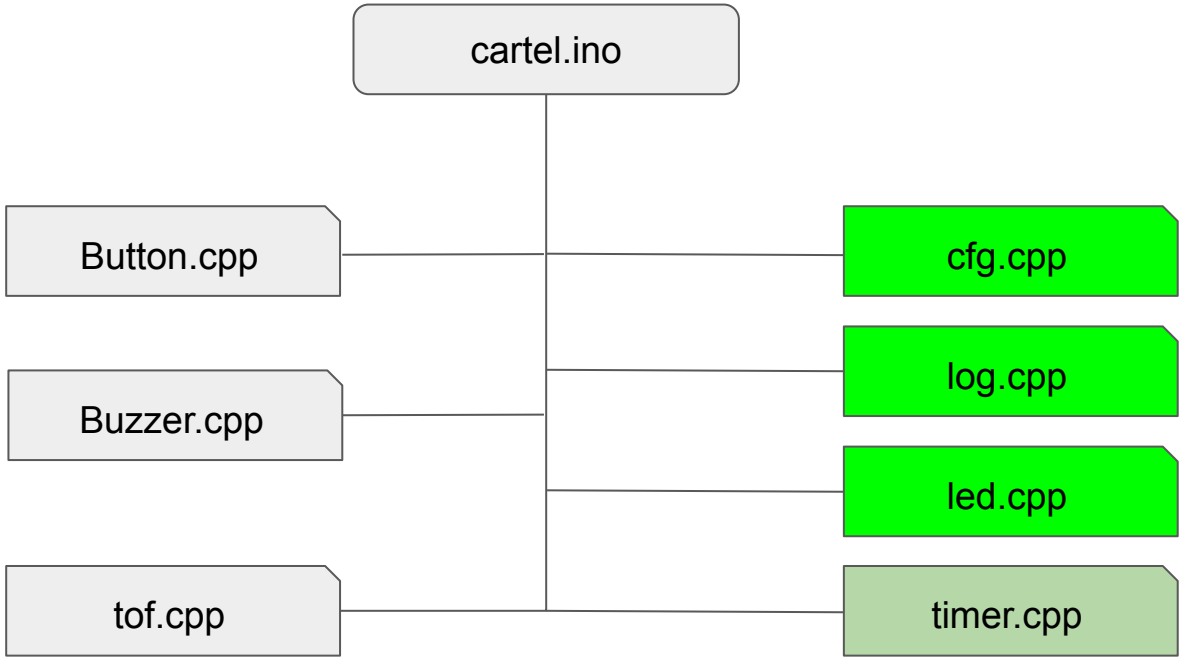
Arduino

https://github.com/jjsch-dev/vl53l0x-arduino

https://github.com/theinsideshine/cartel

Firmware

Cartel de distanciamiento

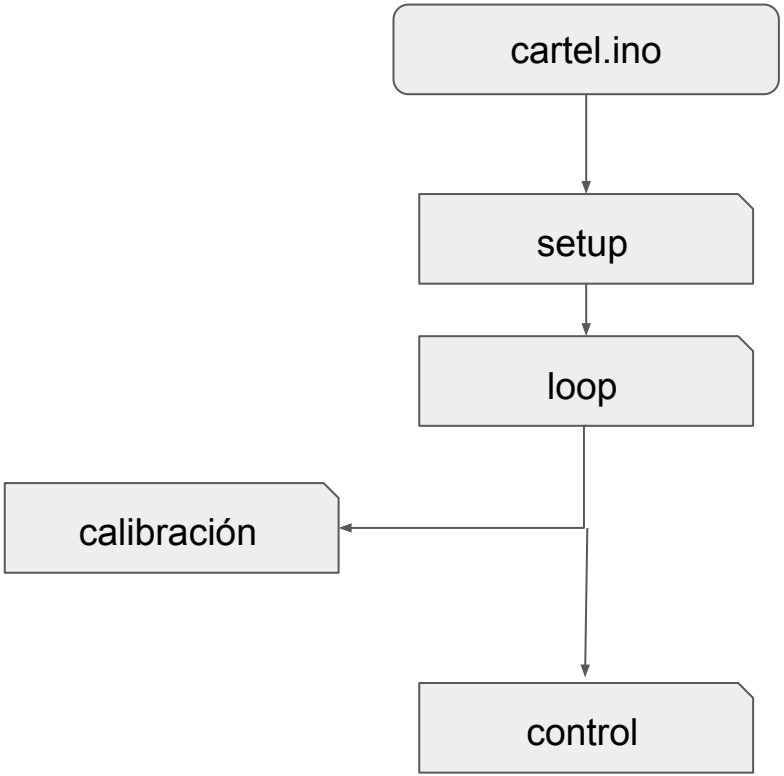


ESP32-Ejecución del ensayo-Debug vía puerto serie-Persistencia de datos
https://www.youtube.com/watch?v=BXJvdTQ8DYk

ESP32-logger serial-modos de ejecución
https://www.youtube.com/watch?v=TlgQHpessV4

Firmware

Cartel de distanciamiento



cartel-state Machine- Polulo

Button.cpp

Funcionalidad destacada:

- **Antirrebote:** La función `debounce()` evita que el botón registre múltiples pulsaciones por efecto del rebote mecánico.
- **Evitar disparos falsos:** La función `is_pressed()` garantiza que un evento de pulsación se registre solo una vez.
- **Temporización:** La lógica incluye un temporizador para asegurarse de que el botón esté liberado por al menos 500 ms entre pulsaciones.

Dependencias:

- `Timer`: Se utiliza para medir el tiempo de espera entre pulsaciones. No está definido en el código proporcionado, pero probablemente sea una clase o biblioteca adicional que ofrece funcionalidad de temporización.
- `PIN_CFG_BUTTON`: Es el pin del microcontrolador al que está conectado el botón. Debe estar definido en otro archivo o en una sección previa del código.

Buzzer.cpp

Funcionalidad destacada:

- **Control sencillo del buzzer:** Métodos dedicados (`on()`, `off()`) facilitan la operación del buzzer.
- **Alternar estado (`tgl()`):** Permite cambiar entre encendido y apagado de forma rápida, útil para patrones como parpadeos o alarmas sonoras.
- **Encendido inicial seguro:** El método `init()` apaga el buzzer al configurarlo para evitar que se active de forma no intencionada.

Métodos:

1. **Constructor (`CTimer::CTimer()`)**
   - Inicializa el temporizador llamando al método `start()` al momento de la creación del objeto.
2. **`start()`**
   - Registra el momento inicial del temporizador, almacenando el valor actual de `millis()` en la variable interna `timer`.
   - Este método reinicia el temporizador para comenzar una nueva medición de tiempo.
3. **`expired(uint32_t ms)`**
   - Comprueba si ha transcurrido un intervalo de tiempo específico (en milisegundos) desde que se inició el temporizador.
   - Retorna `true` si el tiempo transcurrido (`millis() - timer`) es mayor que el intervalo proporcionado (`ms`), de lo contrario, retorna `false`.

Funcionalidad destacada:

- **Temporización no bloqueante:** Utiliza `millis()` para medir el tiempo sin detener la ejecución del programa.
- **Flexibilidad:** Permite reiniciar el temporizador en cualquier momento con el método `start()`.
- **Simplicidad:** El método `expired()` facilita la verificación de si un intervalo de tiempo ha concluido, ideal para bucles principales (`loop`) en aplicaciones Arduino.