

INFORME DE PROYECTO FINAL: SISTEMA DE SEGURIDAD PARA DETECCIÓN DE INTRUSOS



Curso: Técnicas Digitales II

Profesores: Ing. Rubén Darío Mansilla, Ing. Lucas Abdala

Integrantes:

- Barrientos Lucas
- Cuellar Agustín
- Vera Monasterio Candela

Fecha de entrega: Febrero 2025

1. Consideraciones sobre el hardware del proyecto

1.1 Descripción del proyecto

La seguridad en el hogar es una necesidad fundamental en la sociedad actual. Con el avance de la tecnología, los sistemas de seguridad han evolucionado, permitiendo la implementación de dispositivos electrónicos que brindan una protección más efectiva y accesible. Este informe describe el desarrollo de un sistema de seguridad para la detección de intrusos utilizando una placa de desarrollo STM32 NUCLEO-F429ZI. El sistema utiliza sensores PIR para detectar movimiento y sensores magnéticos para monitorear la apertura de puertas o ventanas. Además, incluye un módulo de comunicación Bluetooth (HC-05) y una interfaz UART para enviar alertas a un dispositivo externo.

Objetivo del Proyecto

El propósito principal del proyecto es diseñar e implementar un sistema de alarma que permita monitorear el acceso a una vivienda mediante sensores de movimiento y contacto magnético, proporcionando alertas visuales y sonoras en caso de una intrusión.

Funcionamiento General del Sistema

El sistema de seguridad se activa mediante dos métodos:

- 1.** Presionando el pulsador azul de la placa STM32.
- 2.** Enviando un comando desde una aplicación móvil vía Bluetooth.

1. Inicialización

- Una vez activado, el sistema inicializa todos sus periféricos y muestra un mensaje de inicio en el display OLED.
- Luego, cambia al modo de monitoreo, donde permanece atento a cualquier actividad sospechosa.

2. Estado Activo

En este estado, el sistema supervisa dos sensores principales:

- Sensor PIR (detección de movimiento).
- Sensor magnético (detector de apertura de puertas/ventanas). Debido a la falta de stock del sensor magnético, se reemplazó por un final de carrera, que cumple la misma función.

3. Detección de Intrusión y Contador de Alerta

Si alguno de los sensores detecta una anomalía:

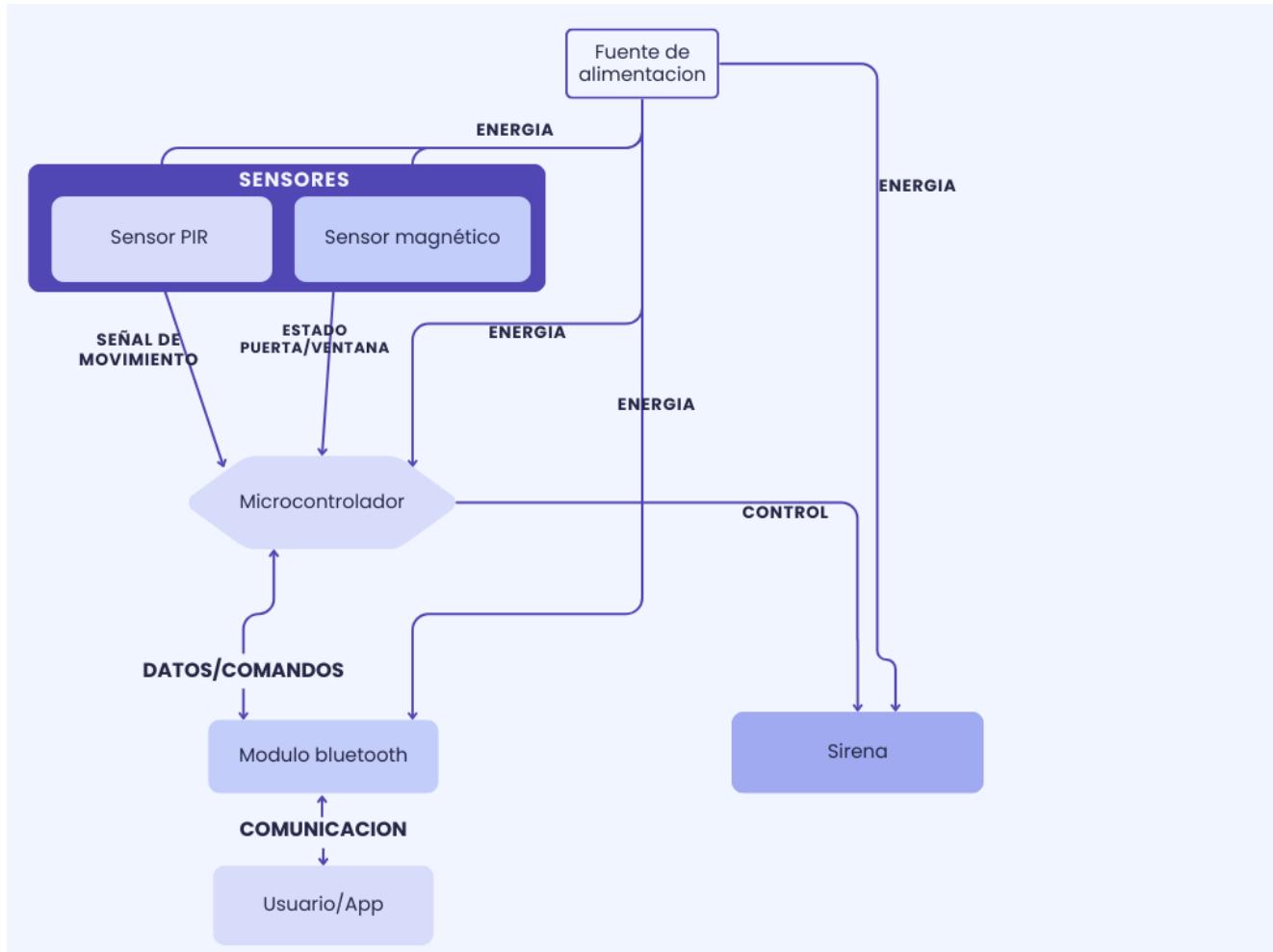
- Se muestra un mensaje de alerta en la pantalla OLED.
- Se activa un contador de 30 segundos, durante los cuales el usuario puede desactivar el sistema antes de que se active la alarma sonora.

4. Desactivación del Sistema

Para apagar la alarma, el usuario debe ingresar un código de seguridad en un teclado matricial 4x4.

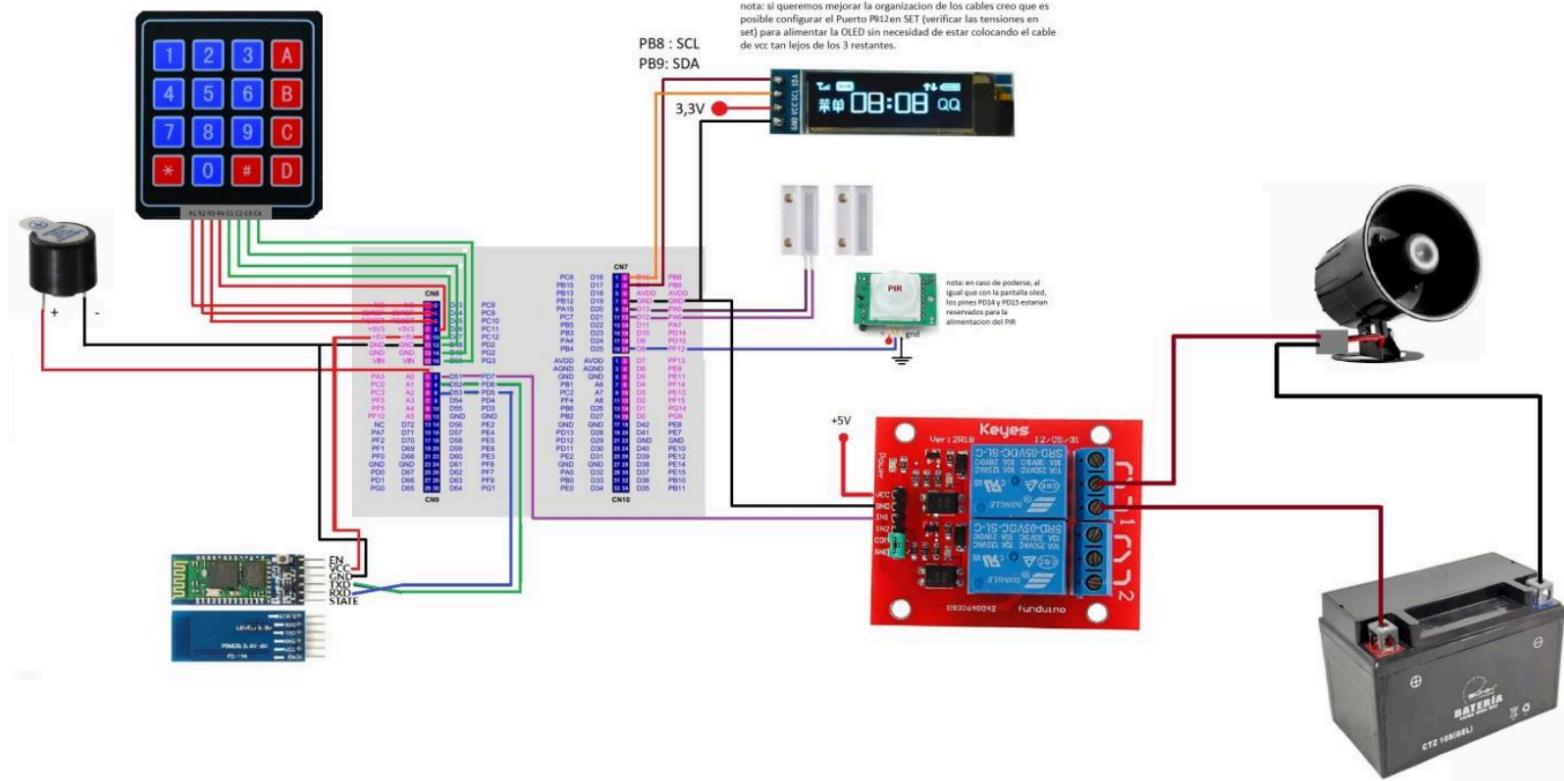
- Al presionar cada tecla, el buzzer emite un sonido como confirmación.
- Si ingresa el código correctamente, el sistema se apaga y vuelve al estado inicial.
- Si falla 3 veces, el teclado se bloquea y la sirena se activa automáticamente.

Diagrama en Bloques

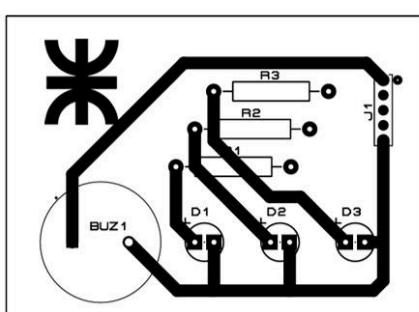


1.2 Circuito del proyecto

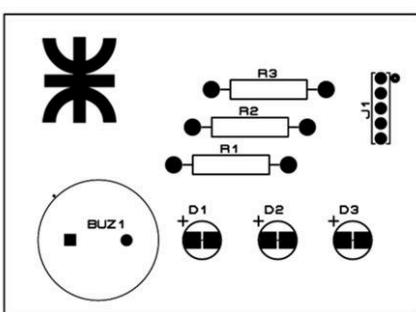
Conexiónado de placa con los periféricos



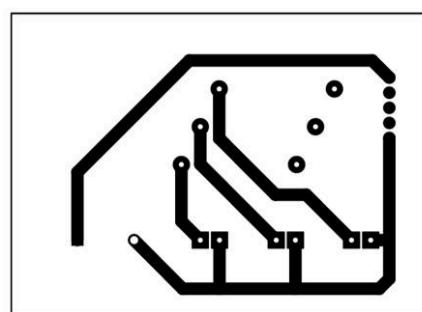
PCB de la placa externa utilizada



Diseño de la placa en PROTEUS8



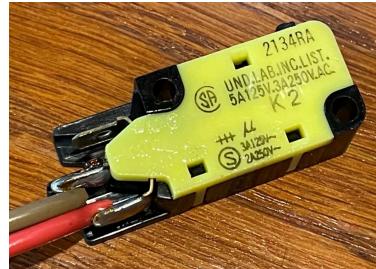
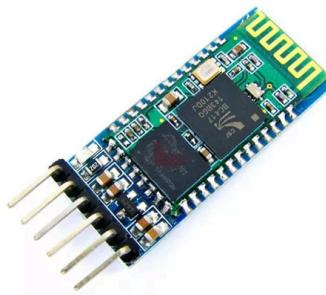
Plano del "Top silk" en PROTEUS8

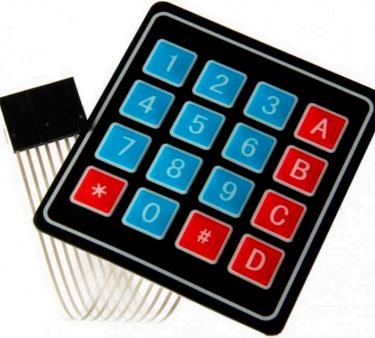


Plano del "Bottom Copper" en PROTEUS8

1.3 Listado de componentes

- Plataforma Embebida: NÚCLEO-F429ZI.
- Periféricos:

Componente	Imagen	Cantidad
Sensor PIR (SR-501)		1
Sensor final de carrera		1
Modulo Bluetooth HC-05		1

LEDs de usuario(rojo, verde y azul)		1 c/u
Buzzer para teclado		1
Display OLED 128x32 I2C		1
Teclado membrana matricial 4x4		1
Sirena de 12 V		

Selección de Hardware

1. Sensores

1.1. Sensor PIR (SR501)

Este sensor se encarga de detectar movimiento mediante la variación de radiación infrarroja en su entorno.

Características principales:

- Rango de detección: 3 a 7 metros.
- Alimentación: 3.3V a 5V.
- Salida digital HIGH (1) cuando detecta movimiento.

Problemas encontrados:

Inicialmente, el sensor detectaba movimiento sin razón aparente, lo que generaba falsas alarmas. Tras revisar las conexiones y la calibración del sensor, se logró estabilizar su funcionamiento.

1.2. Sensor de Puerta (Final de Carrera)

El diseño inicial contemplaba un sensor magnético para detectar aperturas de puertas o ventanas. Sin embargo, debido a la falta de stock, se reemplazó por un final de carrera que cumple una función similar.

Funcionamiento:

- Se instala en la puerta de la maqueta.
- Cuando la puerta se abre, el final de carrera se activa y envía una señal de alerta.

Razón del reemplazo:

La falta de disponibilidad del sensor magnético obligó a buscar una alternativa mecánica, asegurando que el sistema siguiera operando correctamente.

2. Dispositivos de Salida**2.1. Buzzer (Sonido de Confirmación en el Teclado)**

El buzzer se usa exclusivamente para generar un tono corto cada vez que se presiona una tecla en el teclado matricial 4x4.

Se utilizó como efecto de sonido decorativo, mejorando la interacción del usuario con el sistema.

2.2. Sirena de 12V

El sistema de seguridad incluye una sirena de 12V, que se activa cuando el usuario falla tres veces el código de desactivación.

• Activación mediante relé:

Dado que la placa STM32 trabaja con 3.3V y 5V, mientras que la sirena requiere 12V, se implementó un relé como etapa de potencia, permitiendo la activación de la sirena de manera segura.

3. Interfaz de Usuario**3.1. Display OLED 128x32 (I2C)**

Este display es utilizado para mostrar los mensajes de estado del sistema. Su comunicación se realiza a través del protocolo I2C.

Problemas encontrados:

Inicialmente, los textos se cortaban cuando eran demasiado largos (ejemplo: “ALERTA POSIBLE INTRUSO” se mostraba como “ALERTA POS”).

Se intentó solucionar con una función de scroll, pero esta generaba una ejecución bloqueante, lo que afectaba el temporizador de 30 segundos.

Finalmente, se decidió mostrar dos líneas de texto separadas, asegurando que el mensaje fuera completamente legible.

3.2. Teclado Matricial 4x4

El sistema cuenta con un keypad de 4x4 para que el usuario ingrese la clave de desactivación.

Características:

- Matriz de 16 teclas.
- Genera una señal digital para cada tecla presionada.
- Se integra con el buzzer para generar un feedback auditivo.

Condiciones de uso:

- El usuario tiene tres intentos para ingresar la clave correcta.
- Si falla tres veces, el teclado se bloquea y la sirena se activa automáticamente.

4. Indicadores Visuales (LEDs)

Para mejorar la visibilidad de los estados del sistema, se implementaron LEDs de mayor tamaño, replicando las funciones de los LEDs incorporados en la placa STM32.

LED azul → Indica que el sistema está inicializando.

LED verde → Indica que el sistema está monitoreando en estado activo.

LED rojo → Indica una alarma activa o que la sirena está sonando.

5. Conexiones y Alimentación

Para la correcta integración del hardware, fue necesario revisar el datasheet de la placa STM32 Nucleo-F429ZI, asegurando que cada componente estuviera conectado a los pines adecuados.

Principales conexiones:

- **USART** → Para la comunicación con Bluetooth.
- **I2C** → Para el display OLED.
- **5V y 3.3V** → Para la alimentación de sensores y periféricos.

6. Diseño y Construcción de la Maqueta

Además del desarrollo del software y la integración de los componentes electrónicos, una integrante del equipo se encargó del diseño y la construcción de la maqueta, la cual representa un hogar con el sistema de seguridad implementado.

6.1. Materiales Utilizados

Para la fabricación de la maqueta, se utilizó madera MDF debido a su facilidad de corte y ensamblaje, además de ofrecer una superficie adecuada para la instalación de los componentes electrónicos.

Materiales empleados:

- Madera MDF (base y paredes de la maqueta).
- Bisagras pequeñas para la puerta.
- Pegamento de contacto para fijar el teclado matricial y los sensores.
- Pintura acrílica para darle un acabado más realista.

6.2. Proceso de Construcción

Diseño inicial:

- Se realizó un boceto de la estructura, definiendo la ubicación de cada componente.
- Se determinó dónde iría la puerta, los sensores, el teclado matricial, el display OLED y los LEDs indicadores.

Corte y ensamblaje:

- Se cortaron las piezas de madera MDF con las medidas adecuadas.
- Se ensambló la estructura básica de la maqueta con pegamento.
- Se instaló la puerta con bisagras, permitiendo su apertura y cierre.

Integración de los componentes electrónicos:

- Se fijó el keypad 4x4 en la parte frontal de la maqueta para que el usuario pueda ingresar la clave.
- Se pegaron los LEDs indicadores en la parte superior para mejorar la visibilidad.
- Se colocaron los sensores PIR y final de carrera en puntos estratégicos para detectar intrusos.
- Se instaló el display OLED para mostrar mensajes de estado.

6.3. Desafíos y Ajustes Realizados

Ubicación del Keypad:

- Se tuvo que probar distintas ubicaciones hasta encontrar un lugar cómodo para el usuario.
- Se fijó con pegamento para evitar que se mueva al presionar las teclas.

Colocación de Sensores:

- El sensor PIR inicialmente detectaba movimiento incluso cuando no lo había, por lo que se ajustó su posición y sensibilidad.
- Al no encontrar el sensor magnético, se usó un final de carrera como alternativa.

Visibilidad del Display OLED:

- Se inclinó ligeramente el display para que los mensajes sean fáciles de leer desde distintos ángulos.

2. Consideraciones sobre el software

2.1. Link repositorio GRUPO 6:

https://github.com/codecuellar/Grupo_6_TDII_2024

Nombre del proyecto: Sistema de seguridad para detección de intrusos utilizando sensores.

Descripción de funcionamiento de la aplicación del proyecto

La aplicación se basa en una placa de desarrollo STM32 y utiliza un sistema de seguridad que incluye un teclado matricial 4x4, un sensor PIR (movimiento), un sensor magnético (apertura de puerta), y un display OLED para mostrar mensajes. Además, se utiliza un relé para activar una alarma y LEDs para indicar el estado del sistema.

La MEF tiene tres estados principales: **INICIALIZACIÓN**, **ESTADO ACTIVO**, y **ALARMA**. Cada estado tiene entradas y salidas específicas que dependen de las señales de los sensores, el teclado, y el tiempo.

Estados de la MEF:

- **Estado INICIALIZACIÓN:**

- 1. **Entradas:**

- Señal de activación del sistema (por botón físico o Bluetooth).
 - Reinicio manual del sistema (por código o botón).

- 2. **Salidas:**

- LED azul (LD1) parpadea cada 500 ms para indicar que el sistema está en espera.
 - Mensaje en el display: "Iniciando Sistema".

- 3. **Transiciones:**

- Si se recibe una señal de activación (botón o Bluetooth), el sistema pasa al estado ESTADO ACTIVO.

- **Estado ACTIVO:**

- 1. **Entradas:**

- Señal del sensor PIR (movimiento).
 - Señal del sensor magnético (apertura de puerta).
 - Reinicio manual del sistema (por código o botón).

- 2. **Salidas:**

- LED verde (LD2) encendido para indicar que el sistema está activo.
 - Si se detecta movimiento (PIR), se muestra un mensaje en el display: "ALERTA MOVIMIENTO".
 - Si se detecta apertura (sensor magnético), se muestra un mensaje en el display: "ALERTA APERTURA".
 - LED rojo (LD3) se enciende durante 30 segundos si se detecta una alerta.

- 3. **Transiciones:**

- Si no se desactiva la alerta en 30 segundos, el sistema pasa al estado ALARMA.
 - Si el usuario ingresa la contraseña correcta, el sistema vuelve al estado INICIALIZACIÓN.

- **Estado ALARMA:**

- 1. **Entradas:**

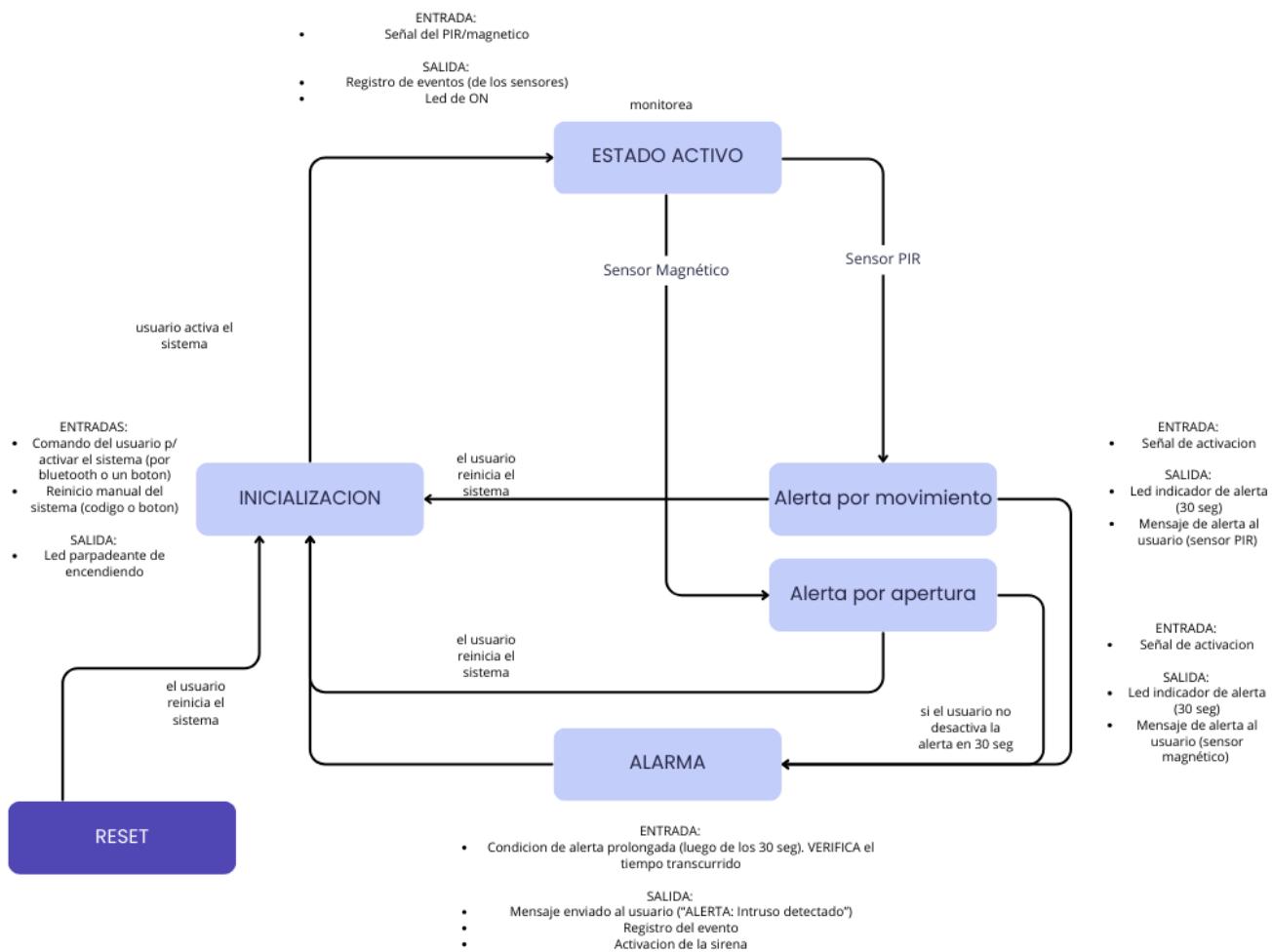
- Condición de alerta prolongada (luego de 30 segundos).
 - Reinicio manual del sistema (por código o botón).

2. Salidas:

- Mensaje en el display: "ALERTA: INTRUSO DETECTADO".
- Registro del evento en el sistema.
- Activación de la sirena (a través del relé).
- LED rojo (LD3) parpadea para indicar que la alarma está activa.

3. Transiciones:

- Si el usuario ingresa la contraseña correcta, el sistema vuelve al estado INICIALIZACIÓN.
- Si se superan los intentos permitidos, el sistema bloquea el teclado y permanece en estado de alarma hasta que se reinicie manualmente.



2.3. Listado de los módulos de software desarrollados en el proyecto

- **Módulo de Inicialización:**

- **Descripción:** Se encarga de inicializar todos los periféricos y configuraciones del sistema, como GPIOs, UART, I2C, y el display OLED.
- **Funciones principales:**
 - MX_GPIO_Init(): Inicializa los pines GPIO.
 - MX_I2C1_Init(): Configura el módulo I2C para la comunicación con el display OLED.
 - MX_USART2_UART_Init(): Configura la UART para la comunicación Bluetooth.
 - Display_Init(): Inicializa el display OLED.

- **Módulo de Control del Sistema (MEF):**

- **Descripción:** Implementa la Máquina de Estados Finitos (MEF) que controla el flujo principal del programa, gestionando los estados **INICIALIZACIÓN, ESTADO ACTIVO, y ALARMA**.
- **Funciones principales:**
 - SystemStateMachine(): Controla las transiciones entre los estados del sistema.
 - SystemON(): Activa el sistema y realiza la secuencia de inicio.
 - SistemaOFF(): Desactiva el sistema y realiza la secuencia de apagado.

- **Módulo de Monitoreo de Sensores:**

- **Descripción:** Se encarga de leer los sensores (PIR y magnético) y detectar eventos como movimiento o apertura de puerta.
- **Funciones principales:**
 - checkSensors(): Verifica el estado de los sensores y activa alertas si es necesario.

- `readPIR_GPIO()`: Lee el estado del sensor PIR.
 - `readMag_GPIO()`: Lee el estado del sensor magnético.
- **Módulo de Gestión de Alertas:**
 - **Descripción:** Gestiona las alertas generadas por los sensores, mostrando mensajes en el display y activando el LED rojo.
 - **Funciones principales:**
 - `mostrarAlerta()`: Muestra un mensaje de alerta en el display y enciende el LED rojo.
 - `activarAlarma()`: Activa la alarma (relé) y bloquea el sistema si no se desactiva en 30 segundos.
 - `desactivarAlarma()`: Desactiva la alarma y reinicia el sistema.
- **Módulo de Teclado:**
 - **Descripción:** Gestiona la entrada del usuario a través del teclado matricial 4x4, permitiendo la introducción de la contraseña.
 - **Funciones principales:**
 - `checkKeypad()`: Lee las teclas presionadas y verifica la contraseña ingresada.
 - `Keypad_Get_Char()`: Obtiene el carácter correspondiente a la tecla presionada.
- **Módulo de Comunicación Bluetooth:**
 - **Descripción:** Gestiona la comunicación con dispositivos externos a través de Bluetooth, permitiendo la activación/desactivación remota del sistema.
 - **Funciones principales:**
 - `HAL_UART_Transmit()`: Envía mensajes de alerta a través de Bluetooth.
 - `HAL_UART_Receive()`: Recibe comandos desde un dispositivo Bluetooth.

- **Módulo de Retardos No Bloqueantes:**

- **Descripción:** Implementa retardos no bloqueantes para controlar el tiempo de las transiciones y acciones en el sistema.
- **Funciones principales:**
 - delayInit(): Inicializa un retardo no bloqueante.
 - delayRead(): Verifica si ha transcurrido el tiempo configurado.

- **Módulo de Control de LEDs:**

- Descripción: Controla los LEDs que indican el estado del sistema (LED azul para espera, LED verde para activo, LED rojo para alerta).
- Funciones principales:
 - Turn_On(): Enciende un LED específico.
 - Turn_Off(): Apaga un LED específico.
 - LED_Toggle(): Cambia el estado de un LED (encendido/apagado).

- **Módulo de Registro de Eventos:**

- Descripción: Registra los eventos importantes en el sistema, como alertas de sensores o intentos fallidos de contraseña.
- Funciones principales:
 - registrarEvento(): Guarda un evento en la memoria del sistema para su posterior revisión.

- **Módulo de Reinicio del Sistema:**

- Descripción: Permite reiniciar el sistema manualmente, ya sea mediante un botón físico o un comando de Bluetooth.
- Funciones principales:
 - reiniciarSistema(): Reinicia el sistema y vuelve al estado de inicialización.

2.4 La estructura inicial del software incluye los siguientes **módulos**:

Drivers Utilizados:

API_GPIO

Encargado de la configuración y control de los pines de entrada y salida del microcontrolador. Permite manipular los estados de los LEDs, el teclado matricial, la sirena y otros periféricos conectados.

Funciones principales:

- Configuración de pines como entrada o salida.
- Control de niveles lógicos en los pines.
- Uso de pull-up y pull-down internos cuando es necesario.

API_Delay

Proporciona una forma precisa de manejar retardos no bloqueantes, permitiendo el control de tiempos sin afectar la ejecución del código principal.

Funciones principales:

- Generación de delays en milisegundos.
- Evita bloqueos en la ejecución del programa.

API_Display (OLED I2C)

Permite la comunicación con el display OLED 128x32 mediante el protocolo I2C, mostrando mensajes en pantalla.

Funciones principales:

- Envío de texto al OLED a través de I2C.

- Configuración del tamaño de fuente y posición del texto.
- Optimización para evitar superposición de mensajes.

API_keypad_4x4 (Teclado Matricial 4x4)

Encargado de la lectura del teclado matricial 4x4, utilizado para ingresar el código de desactivación del sistema.

Funciones principales:

- Escaneo de teclas presionadas. Integración con el buzzer para dar retroalimentación sonora.
- Control de intentos fallidos antes de bloquear el sistema.

API_I2C

Manejo del protocolo I2C para la comunicación con el display OLED y otros dispositivos que utilicen este protocolo.

Funciones principales:

- Inicialización del bus I2C.
- Envío y recepción de datos entre el STM32 y los dispositivos conectados.

API_Alarm

Control del sistema de seguridad, incluyendo monitoreo de sensores, activación de la sirena y manejo de intentos fallidos.

Funciones principales:

- Estado de monitoreo (espera de detección de movimiento o apertura de puerta).

- Activación de alertas en el display y LEDs.

Manejo del código de desactivación con intentos limitados.

Globals

Gestión de variables globales utilizadas en todo el sistema de seguridad, centralizando el acceso y la modificación de datos compartidos entre diferentes módulos.

Funciones principales:

- Declaración de variables globales: Centraliza la declaración de variables que son utilizadas en múltiples partes del sistema, como estados del sistema, contadores, y configuraciones.
- Acceso compartido: Permite que diferentes módulos del sistema accedan y modifiquen estas variables de manera controlada, evitando conflictos y mejorando la mantenibilidad del código.
- Modularidad: Facilita la modularización del código al separar la declaración de variables globales de la lógica de los módulos, permitiendo un desarrollo más organizado y escalable.

API_System

Gestión del estado general del sistema de seguridad, incluyendo el encendido y apagado del sistema, así como la inicialización de componentes clave.

Funciones principales:

- Encendido del sistema (SystemON): Inicializa el sistema de seguridad, reinicia variables clave y muestra mensajes de inicio en el display. También prepara los timers y otros periféricos necesarios para el funcionamiento del sistema.
- Apagado del sistema (SistemaOFF): Desactiva el sistema de seguridad, apaga los LEDs y muestra mensajes de apagado en el display. Reinicia los temporizadores y prepara el sistema para un nuevo ciclo de operación.

- Inicialización de componentes: Prepara los timers y otros periféricos necesarios para el funcionamiento del sistema, asegurando que todo esté listo para operar correctamente.

Primera Actualización

Inclusión del Display OLED

Se añade un display OLED 128x32 para mostrar mensajes y códigos ingresados por el usuario. Esto requirió la creación de dos nuevos módulos:

- API Display: Funciones de inicialización y escritura en el display.
- API OLED Driver: Controlador del SSD1306 para el display OLED.

Comunicación I2C

Para manejar el display OLED, se implementó la comunicación I2C, lo que llevó a la creación del módulo API I2c.

Segunda Actualización

Reemplazo del Buzzer por una Sirena

Para mejorar la potencia de la alarma, se reemplazó el buzzer por una sirena más potente, controlada mediante un relé. Esto requirió modificaciones en el código de API Alarm.

Tercera Actualización

Reconfiguración de Pines y Módulos

Hoy se descubrió que no se habían guardado ciertas configuraciones en el IOC de los pines, lo que requirió volver a configurar los pines. Además, el módulo API

Display.c y su correspondiente archivo de cabecera no se habían guardado correctamente, por lo que fue necesario reescribirlos.

Uso de Biblioteca Externa

Se descargó una biblioteca de GitHub para configurar el controlador del SSD1306, facilitando el manejo del display OLED.

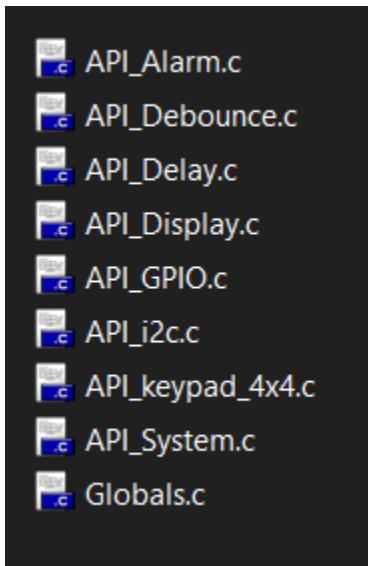
Evolución en el Desarrollo de Drivers

Durante el desarrollo del proyecto, se plantearon dos enfoques distintos para la organización del software:

1. Primer enfoque: Un integrante trabajó en una estructura modular con más de 10 drivers, separando cada funcionalidad en archivos específicos. Sin embargo, al probarlo físicamente con los componentes, el código no funcionaba correctamente. A pesar de varios intentos de corrección, no se logró obtener un sistema estable.

2. Segundo enfoque: Otro integrante implementó el código directamente en el main.c , sin utilizar tantos drivers. Este enfoque sí funcionó correctamente al ejecutarlo en la placa, pero el código no era modular ni fácil de mantener.

Al final, se adoptó una solución híbrida, en la que se optimizó el código del segundo enfoque, incorporando algunos drivers del primer enfoque, quedando la siguiente lista de drivers utilizados:



Lista final de drivers utilizados en el proyecto.

2.5. Listado de los periféricos que utiliza en el proyecto

- GPIO (General Purpose Input/Output):
 - Uso: Control de LEDs (azul, verde y rojo), buzzer, relé (alarma) y lectura de sensores (PIR y magnético).
- I2C1 (Inter-Integrated Circuit):
 - Uso: Comunicación con el display LCD para mostrar mensajes de estado y alertas.
- USART2 (Universal Synchronous/Asynchronous Receiver/Transmitter):
 - Uso: Comunicación Bluetooth para enviar alertas y recibir comandos remotos.

Desafíos Técnicos y Soluciones

Problema con la función bloqueante **HAL_UART_Read_String**

Durante el desarrollo del proyecto, se planteó la posibilidad de activar el sistema mediante un comando enviado por Bluetooth. Para ello, se implementó la función **HAL_UART_Read_String**, que lee una cadena de caracteres desde el módulo

Bluetooth (HC-05) a través de la UART. Sin embargo, se detectó que esta función es **bloqueante**, lo que significa que el sistema se detiene hasta que se recibe el comando completo. Esto generaba los siguientes problemas:

Bloqueo del sistema: Mientras el sistema esperaba recibir el comando por Bluetooth, no era posible interactuar con los pulsadores de usuario integrados en la placa STM32. Esto limitaba la usabilidad del sistema, ya que el usuario no podía activar o desactivar el sistema manualmente mientras se esperaba una señal Bluetooth.

Falta de retroalimentación: Al ser una función bloqueante, no había forma de indicar al usuario que el sistema estaba esperando un comando, lo que podía generar confusión.

Solución Implementada

Para resolver este problema, se decidió eliminar la función `HAL_UART_Read_String` y reemplazar su funcionalidad por una activación manual del sistema mediante los pulsadores integrados en la placa STM32. De esta manera, el sistema no se bloquea y el usuario puede interactuar con él en cualquier momento.

El código original que implementaba la lectura bloqueante era el siguiente:

```
main.c X PD-V2loc
01 /* USER CODE END 2 */
02
03 /* Infinite loop */
04 /* USER CODE BEGIN WHILE */
05 while (1) {
06
07     HAL_UART_Read_String(&huart2, cadena, 20); // si se lee el comando establecido se activa una variable
08
09     if(!strcmp(cadena, "systemON"))
10     {
11         BotonBT=true
12     }
13
14     // Verificar si el sistema está inactivo y se presiona el botón
15     if(!systemActive && (readButton_GPIO()|| botonBT)) {
16         SystemON();
17
18         if (systemActive) {
19             Display_Clear();
20
21             Display_Print("==>ESTADO ACTIVO",0,0);
22             Turn_On(LD2); // Encender LED verde
23         }
24
25         // Verifican sensores si el sistema está activo
26         if (systemActive && !sensoroff) {
27             checkSensors();
28         }
29     }
30
31 } //funcion para leer cadena de caracteres por uart
32 void HAL_UART_Read_String(UART_HandleTypeDef *huart, char *str, uint8_t size)
33 {
34     char caracter[1];
35     uint8_t cont_rx = 0;
36     do
37     {
38         HAL_UART_Receive(huart, (uint8_t*)caracter, 1, HAL_MAX_DELAY);
39         str[cont_rx++] = (char)caracter[0];
40         if(cont_rx >= size) break;
41     }while(caracter[0] != '\n');
42     str[cont_rx-2] = '\0';
43     cont_rx = 0;
44 }
```

Cambios Realizados

- **Eliminación de la función bloqueante:** Se eliminó la función `HAL_UART_Read_String` y su llamada en el lazo principal (while (1)).
 - **Activación manual:** Se simplificó la lógica para que el sistema se active únicamente mediante los pulsadores de la placa STM32, sin depender de la recepción de comandos por Bluetooth.

- **Mejora en la usabilidad:** Al eliminar la función bloqueante, el sistema ahora responde inmediatamente a las entradas del usuario, mejorando la experiencia de uso.

Con estos cambios, el sistema es más robusto y fácil de usar, ya que no depende de una comunicación Bluetooth que podría demorarse o fallar. Además, se mantiene la posibilidad de expandir el sistema en el futuro para incluir una implementación no bloqueante de la comunicación Bluetooth, utilizando interrupciones o DMA.

3. Buenas prácticas de programación

3.1. Ejemplo de variable global privada con un link al código fuente

[systemActive](#)

3.2. Link a la primera línea de la función de actualización de la MEF de elaboración propia

[checkSensors\(\)](#)

3.3. Reglas de Power of Ten aplicadas en el proyecto

- **Regla 9: Limitar el uso de punteros a una sola desreferencia. No usar punteros a funciones**

Descripción: Los punteros utilizados son manejados de manera simple y directa. En la configuración de periféricos, como en la inicialización de ETH, I2C, y UART, se utilizan punteros de manera controlada.

[Link \(linea 315\)](#)

- **Regla 1: Evitar recursión y control de flujo complejo.**

Descripción: No se utilizan recursiones ni sentencias goto en el código. Los bucles tienen límites fijos y no hay recursividad infinita, excepto el lazo infinito de main.c.

El lazo infinito while (1) en main.c es el único bucle infinito permitido, y no hay recursiones ni uso de goto.

[Link \(linea 155\)](#)

- **Regla 8: Usar el preprocesador moderadamente**

Descripción: El uso del preprocesador se limita a #include y #define, lo cual es adecuado y moderado.

[Link \(linea 43 a 50\)](#)

3.5. Link a un comentario en el código que explique el por qué y NO el cómo

En el proyecto, un ejemplo de un comentario que explica el por qué (y no el cómo) podría ser el que describe la razón detrás del uso de la variable `inputPassword`, que es un vector de 5 caracteres utilizado para almacenar la contraseña ingresada por el usuario.

```
static char inputPassword[5] = ""; // Almacena la contraseña ingresada por el usuario (4 dígitos + carácter nulo)
```

[Link:](#)

4. Anexos:

- [Sensor infrarrojo de movimiento PIR HC-SR501](#)
- [EL MÓDULO BLUETOOTH HC-05](#)
- [!\[\]\(baa5661a91c546ceb2ce86ae6c8926c4_img.jpg\) MB1137-EsquematicoPlacaSTM-F4XX.pdf](#)
- [!\[\]\(e70f45dc44932513538475d8faac3296_img.jpg\) STM32 Nucleo-144 board User Manual.pdf](#)
- [128 x 32 Graphic OLED](#)