**UNIVERSIDAD NACIONAL DE COLOMBIA**

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA


# Informe final del proyecto

## Electrónica Digital II

**Juan Felipe Gaitán Nocua - jgaitann@unal.edu.co**
**Samuel Mahecha Arango - samahechaa@unal.edu.co**
**Daniel Sepúlveda Suárez - dasepulvedas@unal.edu.co**

Profesor: Diego Alexander Tibaduiza Burgos

Bogotá D.C.
12 de Diciembre de 2025

# ÍNDICE

# 1. PROBLEMÁTICA

La exposición prolongada a campos magnéticos en entornos laborales e industriales constituye una preocupación creciente dentro de los ámbitos de la salud ocupacional y la seguridad. En sectores como la operación de subestaciones, líneas de transmisión, equipos industriales de alta potencia o maquinaria electromecánica, los trabajadores pueden verse sometidos a niveles de campo que, si no se monitorean adecuadamente, podrían generar efectos fisiológicos no deseados, como sensaciones de hormigueo, estimulación involuntaria de tejidos o malestar general en exposiciones prolongadas [1].

A pesar de la importancia del monitoreo, los sistemas comerciales disponibles presentan limitaciones relevantes, suelen ser costosos, poseen una integración limitada con plataformas digitales modernas y en muchos casos, no cuentan con procesamiento en tiempo real que permita reaccionar de manera inmediata ante niveles superiores a los umbrales recomendados.

A nivel internacional, la normativa evidencia la necesidad de sistemas de supervisión continua. Por ejemplo, el reglamento Control of Electromagnetic Fields at Work Regulations (CEMFAW, 2016) [2] del Reino Unido obliga a los empleadores a evaluar y comparar los niveles de campo con los Valores Límite de Exposición (ELVs) y a tomar medidas inmediatas si estos son superados. Aunque este marco normativo contempla tanto campos eléctricos como magnéticos, resalta la importancia de disponer de herramientas precisas, accesibles y permanentes para detectar superaciones de umbral en tiempo real, especialmente en ambientes donde predominan los campos magnéticos generados por equipos de alta corriente.

En este contexto surge la necesidad de diseñar un sistema de monitoreo de campos magnéticos que sea accesible, flexible, de bajo costo y capaz de integrarse fácilmente a entornos digitales modernos, permitiendo procesar la información en tiempo real y emitir alertas oportunas para la protección de los trabajadores y usuarios en instalaciones eléctricas e industriales.

# 2. OBJETIVOS

## 2.1. General

Diseñar e implementar un prototipo de un sistema digital capaz de monitorear campos electromagnéticos, procesar mediciones en tiempo real y activar mecanismos de alerta.

## 2.2. Objetivos específicos

- Activar una alarma cuando el campo magnético supere un valor de referencia o cuando se detecte movimiento en una zona restringida, asegurando que el sistema solo pueda restablecerse mediante una desactivación manual segura.

- Implementar una interfaz de autenticación mediante usuario y contraseña que limite el acceso al sistema exclusivamente a personal autorizado, proporcionando un nivel básico de seguridad.
- Visualizar en una pantalla LCD los valores instantáneos del campo magnético, facilitando la lectura y supervisión por parte de los operarios presentes en el entorno de riesgo.
- Analizar y representar gráficamente el comportamiento en tiempo real del campo magnético en el entorno monitorizado, permitiendo identificar cambios abruptos o condiciones anómalas.

# 3. PERIFÉRICOS

## 3.1. Entradas

- **Teclado matricial 4×4:** Permite la interacción directa del usuario con el sistema, posibilitando el ingreso de un nombre de usuario y contraseña para el control de acceso. Además, facilita la selección de funcionalidades tales como activar la medición del campo magnético, ejecutar pruebas de alarma, modificar credenciales o navegar por el menú del sistema. [3]
- **Sensor de campo magnético GY-271:** Módulo utilizado para la medición del campo magnético en el entorno. Sus lecturas permiten evaluar si los niveles detectados exceden un umbral configurado por el usuario, activando automáticamente el sistema de alarma en caso de sobrepasarlo. [4]
- **Sensor de movimiento PIR:** Dispositivo encargado de detectar la presencia de un individuo en una zona restringida. Su activación sirve como mecanismo adicional de seguridad, alertando sobre posibles exposiciones indebidas a campos magnéticos o ingresos no autorizados. [5]

## 3.2. Salidas

- **Buzzer KY-012:** Emite una alarma sonora cuando el sistema detecta que la magnitud del campo magnético supera el umbral definido por el usuario o cuando se registra movimiento dentro de una zona protegida. [6]
- **Pantalla LCD con interfaz I2C:** Permite visualizar información relevante del sistema en tiempo real, como la magnitud del campo magnético, el estado de las alarmas, mensajes del menú de usuario y notificaciones del sistema. [7]
- **Aplicación en computador:** Herramienta de monitoreo donde el usuario puede visualizar gráficas del comportamiento del campo magnético, revisar alertas generadas y consultar el historial de mediciones. Además, muestra información crítica como la magnitud del campo detectado en tiempo real.

# 4. ANÁLISIS PESTAL

## 4.1. Político

Organismos como la OMS y la ICNIRP establecen normas sobre la exposición a campos magnéticos, lo cual obliga a los gobiernos a revisar sus leyes de protección, medición y control. Estos lineamientos internacionales promueven políticas que exigen certificaciones y auditorías de seguridad en industrias con alta exposición a campos magnéticos, incentivando la adopción de tecnologías de monitoreo continuo. Además, la creciente preocupación gubernamental por la salud laboral impulsa la financiación de proyectos destinados a mejorar la vigilancia de riesgos físicos en entornos industriales. [8]

## 4.2. Ambiental

Los campos magnéticos intensos se encuentran en entornos como líneas de transmisión, subestaciones, estaciones de carga de vehículos eléctricos y equipos industriales de alta potencia. La implementación de sistemas que reduzcan la exposición a niveles potencialmente nocivos protege tanto la salud humana como el equilibrio ambiental en zonas afectadas. Asimismo, el uso de dispositivos capaces de detectar variaciones abruptas en el campo magnético permite identificar fallas, fugas eléctricas o sobrecargas que podrían representar riesgos ambientales. Al operar con bajo consumo energético y ser compatibles con estrategias de sostenibilidad, este tipo de soluciones contribuye a prácticas industriales más responsables. [9]

## 4.3. Legal

Existen normativas internacionales para establecer límites de exposición a campos magnéticos, emitidas por entidades como ICNIRP, IEEE y OMS, que cada país adapta según sus necesidades. En Colombia, instituciones como la Agencia Nacional del Espectro y organismos de salud ocupacional regulan los niveles permitidos en ambientes laborales. Esto obliga a los empleadores a garantizar ambientes seguros, ya que el incumplimiento puede generar sanciones, demandas laborales o indemnizaciones. Además, normativas de protección de datos como el Habeas Data requieren que la información recolectada por los sistemas de monitoreo sea manipulada de forma ética y segura, asegurando transparencia y trazabilidad en la gestión del riesgo. [10]

## 4.4. Económico

Los equipos especializados para medir campos magnéticos suelen ser costosos, limitando su accesibilidad para pequeñas y medianas empresas. El desarrollo de soluciones basadas en hardware reconfigurable, como FPGAs, reduce los costos frente a instrumentos comerciales, ofreciendo flexibilidad y escalabilidad en el monitoreo. Por otro lado, los efectos adversos derivados de exposiciones prolongadas generan costos adicionales para empresas y sistemas de salud, incluyendo incapacidades, tratamientos médicos y pérdidas de productividad. La implementación de un sistema económico

de alerta temprana contribuye a reducir estos gastos al prevenir accidentes, evitar daños en equipos y optimizar la gestión del riesgo laboral. [11]

## 4.5. Tecnológico

La medición de campos magnéticos requiere el desarrollo de tecnologías capaces de registrar señales de baja magnitud y alta variabilidad con suficiente precisión y estabilidad. Entre los principales retos tecnológicos se encuentran la sensibilidad y exactitud de los sensores, la mitigación del ruido y las interferencias generadas por otros dispositivos eléctricos, así como la necesidad de procesar la información en tiempo real.

Adicionalmente, la miniaturización de los equipos y su integración con plataformas digitales modernas, como sistemas embebidos y hardware reconfigurable, representa un desafío clave para garantizar soluciones compactas, eficientes y escalables. El uso de FPGAs permite implementar arquitecturas flexibles capaces de adquirir, procesar y transmitir datos de forma simultánea, facilitando el cumplimiento de normativas internacionales de exposición y la adaptación del sistema a distintos entornos industriales o laborales. [12]

## 4.6. Social

La implementación de un sistema de protección frente a niveles elevados de campos magnéticos contribuye directamente al bienestar de los trabajadores y de la sociedad en general, al reducir los riesgos asociados a exposiciones prolongadas en entornos industriales y eléctricos. Este tipo de soluciones promueve una mayor conciencia sobre la importancia de la seguridad laboral y la prevención de riesgos físicos, fortaleciendo la cultura de autocuidado.

Asimismo, la familiarización de los usuarios con sistemas de monitoreo y alerta facilita su aceptación e integración en la vida cotidiana, abriendo la posibilidad de su futura incorporación en dispositivos móviles, aplicaciones informáticas y plataformas de supervisión remota. De esta manera, la tecnología no solo actúa como una herramienta de protección, sino también como un medio educativo que fomenta prácticas más seguras y responsables frente a la exposición a campos magnéticos. [13]

# 5. TECLADO MATRICIAL 4X4


Figura 1: Teclado matricial 4x4 de 16 dígitos [3]

## 5.1. Teclado Matricial 4x4 e Integración en la Zybo Z7

Un teclado matricial de 4x4 está compuesto por 16 teclas organizadas en una red de 4 filas por 4 columnas, cada tecla funciona como un interruptor que, al presionarse, conecta una fila con una columna, y es gracias a esta estructura matricial que es posible manejar las 16 teclas utilizando únicamente 8 pines, lo que lo convierte en una solución eficiente para la integración de la contraseña de acceso al usuario que desactive el sistema de alarmas que se ha planteado para el proyecto..

### 5.1.1. Principio de funcionamiento

La lectura del teclado se realiza mediante un proceso de barrido scanning, este proceso consiste en activar una fila a la vez desde la FPGA y mientras dicha fila permanece activa, se lee el estado de las columnas, y si una columna cambia de su estado normal, se detecta que una tecla ha sido presionada en la intersección entre esa fila y esa columna.

Este procedimiento se repite rápidamente para las cuatro filas, permitiendo detectar cualquier tecla del teclado. Para facilitar la detección, las columnas suelen utilizar resistencias pull-up, de forma que su estado por defecto es alto y disminuye cuando se presiona una tecla (también se puede configurar el pull-up desde las salidas/entradas de la Zybo-Z7 el archivo de constrains).

### 5.1.2. Integración en Vivado

Para utilizar el teclado con la Zybo Z7, se ha intentado implementar en Vivado un módulo encargado del barrido y la detección de teclas, en donde las filas se configuran como señales de salida desde la FPGA hacia el teclado, mientras que las columnas se reciben como señales de entrada, este módulo genera un código digital que identifica la tecla presionada.

Para comunicar este código con el procesador ARM de la Zybo, se utiliza un bloque AXI-GPIO, que actúa como interfaz entre la lógica programable (PL) y el procesador del sistema (PS). Una vez configurada esta estructura, Vivado permite generar el bitstream y exportar el diseño hacia Vitis.

### 5.1.3. Lectura del teclado en Vitis

En Vitis, el procesador ARM ejecuta un programa que lee el valor entregado por el módulo AXI-GPIO, dicho valor corresponde al código de la tecla detectada y se interpreta utilizando una tabla de equivalencias, por ejemplo, fila 0 más columna 0 corresponde a la tecla 1. Esta información podría llegar a ser enviada por UART (lo hemos probado desde la consola) o utilizada internamente en el sistema.

### 5.1.4. Conexiones físicas

El teclado se conecta utilizando un conector PMOD disponible en la Zybo Z7, en donde se emplean cuatro pines para las filas (salidas) y cuatro pines para las columnas (entradas), por lo que en el archivo de restricciones (XDC) debe modificarse para asignar cada señal a un pin físico del conector Pmod.

# 6. SENSOR DE CAMPO MAGNÉTICO

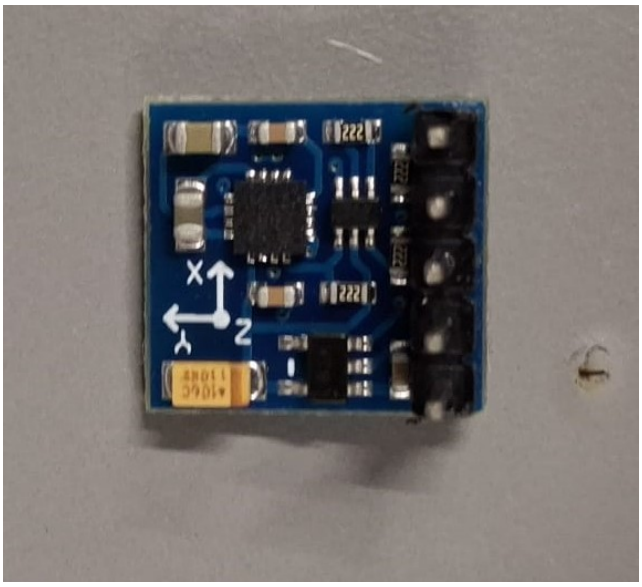


Figura 2: Magnetómetro de 3 ejes (HCM5883L) [4]

Una parte esencial del proyecto es la medición del campo magnético, existen diferentes métodos de medición, entre los que se encuentran:

- Efecto faraday.
- Magnetoresistencia (ampliamente utilizado en el desarrollo de discos de estado solido).
- Magneto inductancias.
- Efecto hall.
- Efecto de inducción.
- SQUID

Al existir tantas alternativas en el mercado es importante tener en cuenta varios factores como el precio, precisión, rango,etc.

Para el desarrollo del proyecto se usa el sensor HMC5883L magnetómetro, el cual cuenta con las siguientes características:

- Salida digital de 16 bits.
- Distintos modos de operación.
- Utiliza para la comunicación el protocolo I2C
- Internamente se controla por medio de 12 registros los cuales se escriben o leen.

## 6.1. Registros

A continuación se muestran las listas de registros cada de un tamaño de 8 bits:

| Address Location | Name | Access |
|---|---|---|
| 00 | Configuration Register A | Read/Write |
| 01 | Configuration Register B | Read/Write |
| 02 | Mode Register | Read/Write |
| 03 | Data Output X MSB Register | Read |
| 04 | Data Output X LSB Register | Read |
| 05 | Data Output Z MSB Register | Read |
| 06 | Data Output Z LSB Register | Read |
| 07 | Data Output Y MSB Register | Read |
| 08 | Data Output Y LSB Register | Read |
| 09 | Status Register | Read |
| 10 | Identification Register A | Read |
| 11 | Identification Register B | Read |
| 12 | Identification Register C | Read |

cada registro cumple la siguientes funciones:

- Registro 00: Configura los parámetros de la medición y el ritmo de salida de datos, Permite definir cuántas muestras se promedian por medición, el ritmo de salida de datos, y la configuración de la medición, como la incorporación de un sesgo (bias) en las mediciones.
- Registro 01: El registro B sirve para configurar la ganancia del dispositivo. La cual varia desde 230 hasta 1370.
- Registro 02: Este registro sirve para establecer en que modo funciona el sensor.
- Registro 03: Guarda los primeros 8 bits menos significativo de la posición x del campo magnético.
- Registro 04: Guarda los últimos 8 mas significativos de la posición x del campo magnético.
- Registros 05: Guarda los primeros 8 bits menos significativo de la posición z del campo magnético.
- Registro 06: Guarda los últimos 8 bits mas significativos de la posición z del campo magnético.
- Registro 07: Guarda los 8 bits menos significativos en la de la posición Y del campo magnético.
- Registro 08: Guarda los 8 mas significativos en la posición Y del campo magnético.
- Registro 09: Guarda el estado del sensor.
- Registro 11: Da información del fabricante.
- Registro 12: Da información del fabricante.

## 6.2. Funcionamiento

Para acceder a cada uno de los registros se necesitan tener en cuenta las siguientes consideraciones:

El maestro envía la dirección del dispositivo (7 bits) + el bit de lectura/escritura a continuación este envía el puntero de dirección del registro en el que quiere leer o escribir, si es una escritura, el maestro envía los datos que quiere que el sensor registre. Si se trata de una lectura, el maestro obtiene los datos del sensor a partir del registro al que apuntó.

El puntero de direcciones se actualiza cuando lee solo dentro del dispositivo teniendo en cuenta estas dos features , de la siguiente manera: Cuando la dirección 12 o mayor es accedida el puntero de address vuelve automáticamente a 0. A continuación cuando la dirección de address alcanzó el número 8, la dirección vuelve automáticamente al 03. (lee el ultimo bit de salida T lsb y vuelve al tercer bit de salida es decir X msb); Si se lee la dirección que no esta disponible vuelve a 0 y si se escribe algo incorrecto el dispositivo lo ignora. Para finalizar El punto de dirección no se puede leer.

# 7. SENSOR DE MOVIMIENTO PIR

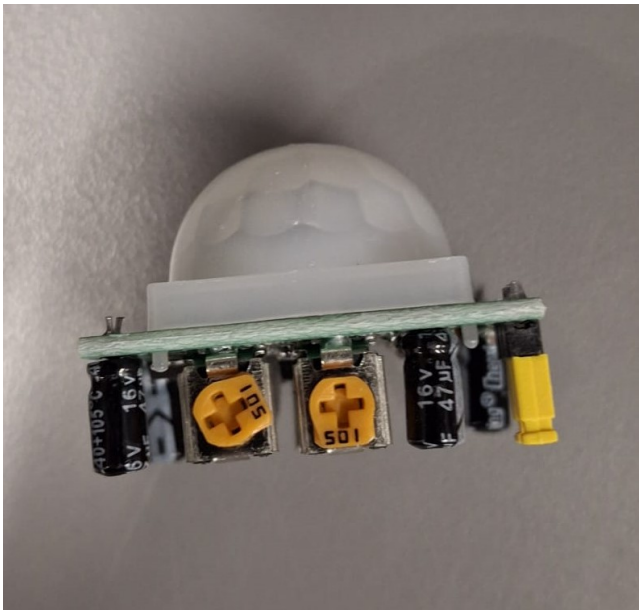

Figura 3: Sensor de movimiento PIR - HC-SR501 [5]

## 7.1. Sensor PIR HC-SR501 e integración en la Zybo Z7

El sensor de movimiento PIR HC-SR501 es un dispositivo utilizado para detectar la presencia de personas mediante la variación de la radiación infrarroja emitida por los cuerpos. Este sensor

resulta especialmente útil como mecanismo adicional de seguridad en el proyecto, ya que permite identificar si una persona ingresa a una zona de posible exposición a campos magnéticos elevados.

### 7.1.1. Principio de funcionamiento

El sensor PIR opera mediante un elemento piroeléctrico que detecta cambios en la radiación infrarroja del entorno. Cuando un cuerpo en movimiento atraviesa el campo de detección del sensor, se produce una variación que es procesada internamente y se traduce en una señal digital de salida. Esta señal adopta un nivel lógico alto cuando se detecta movimiento y permanece en bajo en ausencia de este. Este tiene un rango entre 120 y 100 grados, y un rango de medicion de 2 hasta 7 metros por lo que se tapo una partde del sensor para acotar el angulo de medicion.

### 7.1.2. Integración en Vivado y Vitis

La salida digital del sensor PIR se conecta a un pin de entrada de la FPGA a través de un bloque AXI-GPIO configurado como entrada. Desde el procesador ARM, esta señal es leída periódicamente para determinar la presencia de movimiento. En caso de activación, el sistema puede generar alertas, activar la alarma o registrar el evento como una condición de riesgo.

# 8. BUZZER ACTIVO


Figura 4: Modulo Buzzer Activo - KY-012 [6]

## 8.1. Módulo buzzer KY-012 e integración en la Zybo Z7

El módulo buzzer activo KY-012 es un dispositivo de salida utilizado para generar alertas sonoras de forma inmediata. En el proyecto, este módulo cumple la función de alarma acústica cuando se detectan condiciones de riesgo, como la superación del umbral de campo magnético o la presencia de movimiento en una zona restringida.

### 8.1.1. Principio de funcionamiento

A diferencia de los buzzers pasivos, el buzzer activo incorpora un oscilador interno, por lo que únicamente requiere una señal digital para su activación. Al recibir un nivel lógico alto en su entrada, el buzzer emite un sonido continuo sin necesidad de generar una señal PWM o una frecuencia externa.

### 8.1.2. Integración en Vivado y Vitis

El buzzer se conecta a un pin de salida de la FPGA controlado mediante un bloque AXI-GPIO configurado como salida. Desde el software en Vitis, el procesador ARM controla el estado del buzzer activándolo o desactivándolo según las condiciones del sistema, garantizando una respuesta rápida ante eventos críticos.

# 9. PANTALLA LCD

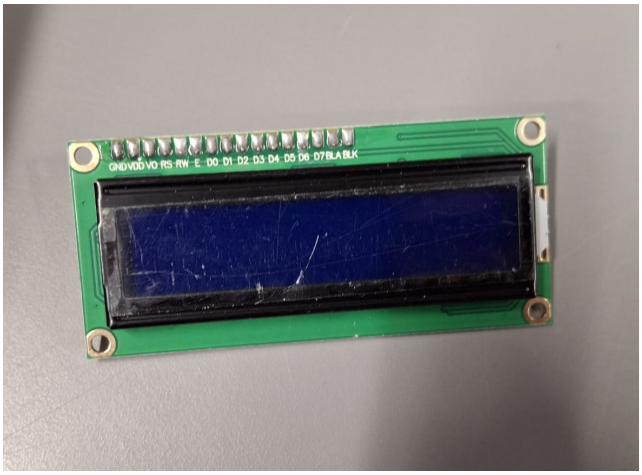


Figura 5: Pantalla LCD con módulo I2C [7]

## 9.1. Pantalla LCD 16x2 con interfaz I2C e integración en la Zybo Z7

La pantalla LCD de 16x2 con conversor I2C PCF8574 se utiliza como interfaz visual del sistema, permitiendo mostrar información relevante al usuario de forma clara y en tiempo real. Entre los datos presentados se incluyen los valores del campo magnético, el estado del sistema y mensajes del menú de usuario.

### 9.1.1. Principio de funcionamiento

La pantalla LCD funciona bajo el estándar HD44780, mientras que el módulo PCF8574 actúa como expansor de E/S, permitiendo la comunicación mediante el protocolo I2C. Esta configuración reduce significativamente el número de pines necesarios, utilizando únicamente dos líneas de comunicación: SDA y SCL.

### 9.1.2. Integración en Vivado y Vitis

La comunicación I2C se gestiona desde el procesador ARM utilizando el controlador I2C del sistema. En Vitis, se implementan rutinas para el envío de comandos y datos a la pantalla, permi-

tiendo controlar el cursor, borrar la pantalla y mostrar texto dinámico. Esta integración facilita una interacción intuitiva entre el usuario y el sistema.

# 10. IMPLEMENTACIÓN

Para la implementación del proyecto se utiliza la arquitectura Zynq creada por AMD, donde por un lado se tiene un procesador ARM (cortex A9) y por otro se puede utilizar la lógica programable, dando lugar al anteriormente mencionado soc (system on a chip). Por medio del procesador es programado, mientras que la parte lógica a través del lenguaje de descripción de hardware verilog, se utiliza para la creación de periféricos personalizados útil a nivel de rendimiento y libertad de la implementación.

Dicho proceso debe seguir los siguientes pasos:

- Creación del proyecto de Vivado, plataforma que usara para el diseño del hardware.
- Diseño del procesador por medio de la herramienta block design usando distinto bloques con el Zynq, o el gpiout, realizar las conexiones necesarias y establecer las salidas y entradas que la FPGA tendrá.
- Compilar y exportar el proyecto en formato .vsa
- Importar dicho archivo .vsa al software de vitis y crear la plataforma con base en ese archivo, escoger la configuración y donde va a correr (standalone en este caso).
- Crear un proyecto en el mismo software que se ejecute en la plataforma creada anteriormente.
- implementar el código en el lenguaje de programación c necesario por medio de las distintas librerías como plattform, necesarias para interactuar con los registros y los periféricos.
- Ejecutar el proyecto sobre la fpga, programarla en modo jtag

Como primera medida en la creación del sistema, es importante la implementaciones de una protocolo de comunicación, necesario para enviar las señales del sistema hacia otro dispositivo. Por lo se busco como primera implementación el funcionamiento del protocolo UART. A continuación se muestra su implementación ya realizados los pasos mencionados anteriormente.

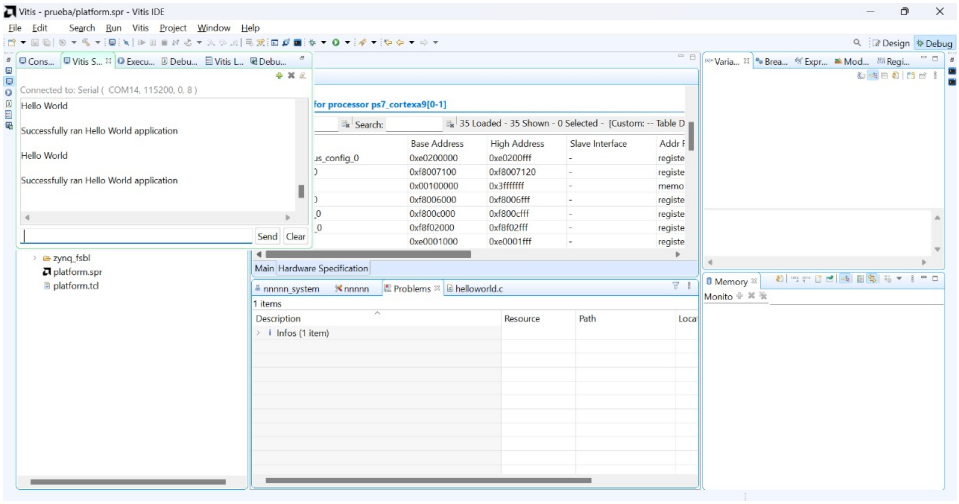


Figura 6: Implementación física del UART en la FPGA

Como se puede observar en la imagen en la parte superior izquierda en la terminal donde se visualiza un puerto del computador, un mensaje se envió por medio de UART a este puerto. Dicho modulo UART viene integrado con el procesador, por lo que en el diseño del block design es necesario activarlo y tendrá dos protocolos UART, uno por medio de USB aplicado en este caso y otro con pines de salida.

Por otro lado el diseño en el block design de la siguiente manera:

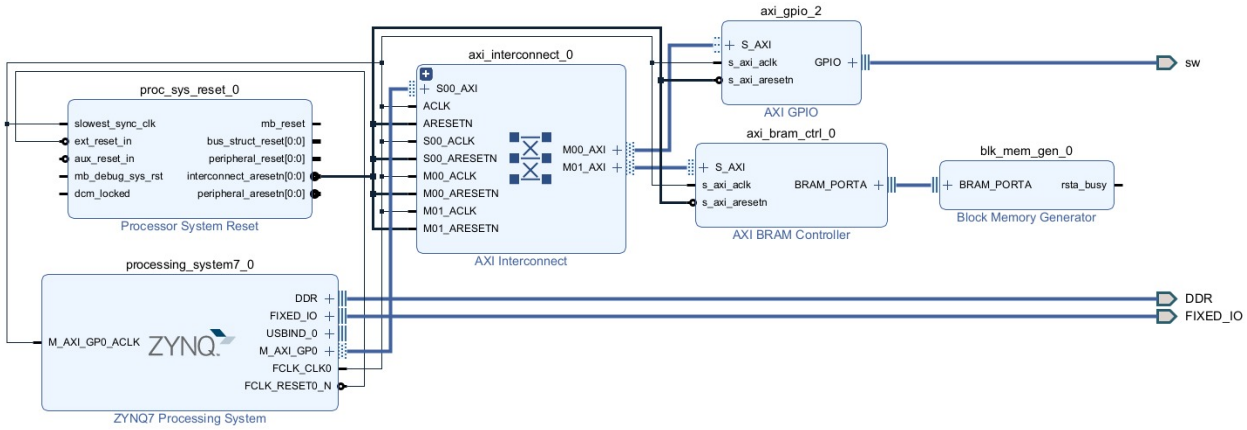


Figura 7: Block design del procesador

Como se observa el procesador es el bloque Zynq proccesor conectado a varios bloques gpio por medio de un axi connector y por ultimo los bloques de memorias bram y memgen

## 10.1. Diseño de bloques final - Procesador

Finalmente, el procesador implementado mediante Vivado 2023.1 permitió la integración y configuración de los distintos periféricos que conforman el sistema propuesto. La arquitectura resultante, que evidencia la interconexión entre el procesador, la lógica programable y los módulos de entrada y salida, se presenta en la figura a continuación.

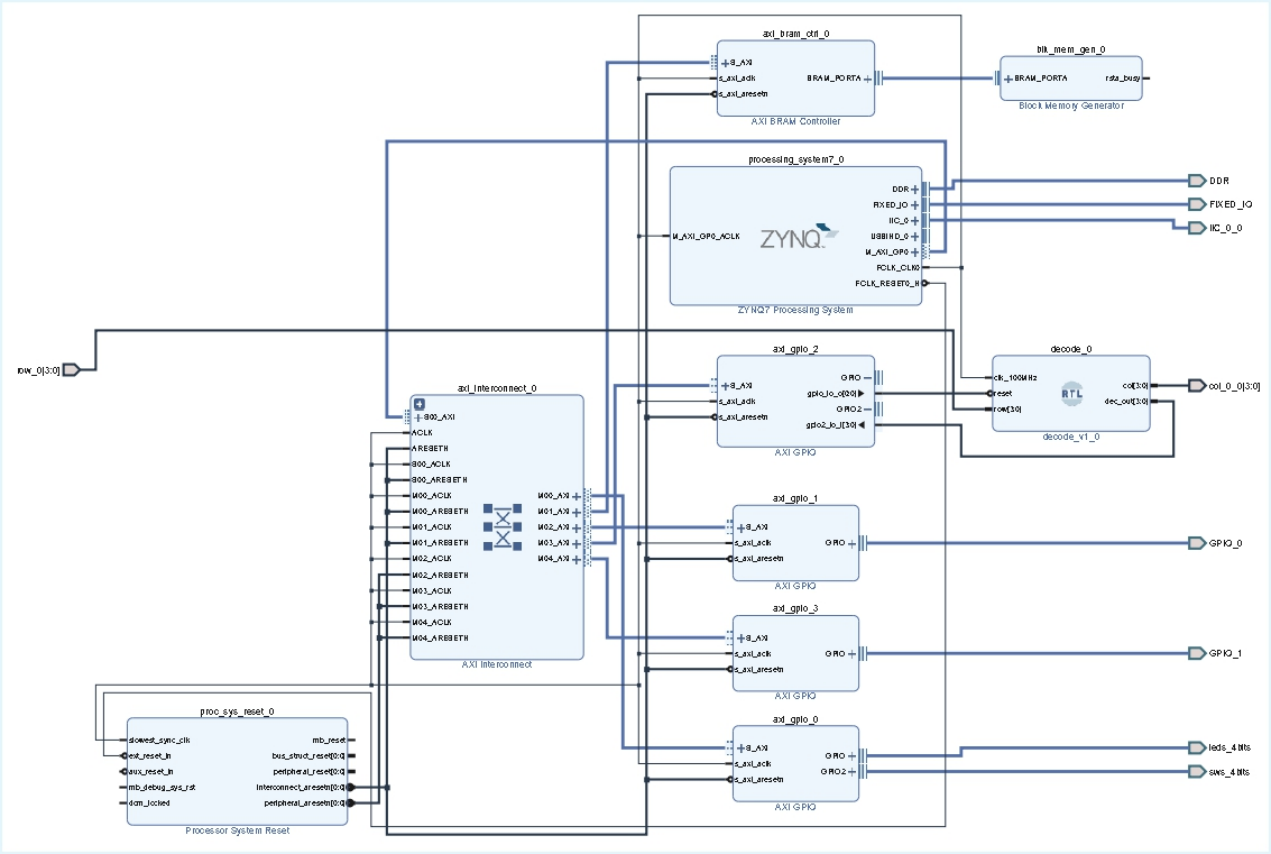


Figura 8: Diseño de bloques final - Vivado 2023-1

## 10.2. Processing System 7

El bloque *Processing System 7* corresponde al núcleo del sistema y contiene el procesador ARM Cortex-A9 de la arquitectura Zynq-7000. Este procesador es el encargado de ejecutar el software desarrollado en Vitis, desde donde se gestionan las lecturas de los sensores, la interacción con el usuario, el procesamiento de datos y la activación de las alarmas del sistema. [17]

Adicionalmente, este bloque proporciona acceso a la memoria DDR externa, interfaces de comunicación como I2C y UART, y genera las señales de reloj y reinicio necesarias para sincronizar los módulos implementados en la lógica programable.

## 10.3. AXI Interconnect

El bloque *AXI Interconnect* actúa como el bus de comunicación principal entre el procesador (PS) y los periféricos implementados en la lógica programable (PL). A través del protocolo AXI, este módulo permite la transferencia de datos, direcciones y señales de control de manera eficiente, garantizando que el procesador pueda acceder a cada uno de los periféricos conectados al sistema.

## 10.4. AXI GPIO

Los bloques *AXI GPIO* funcionan como interfaces entre el procesador y las señales digitales externas del sistema. Estos módulos permiten configurar pines como entradas o salidas y acceder a ellos mediante direcciones de memoria desde el software.

En el sistema desarrollado, los bloques AXI GPIO se emplean para:

- La lectura del teclado matricial, permitiendo la interacción del usuario.
- La lectura del sensor de movimiento PIR para detectar presencia en zonas restringidas.
- El control de dispositivos de salida como el buzzer de alarma.

Esta estructura facilita la comunicación entre la lógica programable y el procesador ARM de forma flexible y escalable.

En este diseño, el bloque `axi_gpio1` se utilizó para generar una salida de 1 bit desde los pines de la FPGA, configurando posteriormente el archivo `.xdc` correspondiente para asegurar la correcta integración del módulo con la placa.

El bloque `axi_gpio2` se empleó para recibir las entradas provenientes del teclado matricial, con un total de 4 bits de entrada. Este bloque trabaja en conjunto con un módulo Verilog denominado `decode_0`, el cual se encarga de decodificar los caracteres obtenidos al presionar una tecla, ya que el teclado por sí mismo no realiza dicha decodificación. Además, el módulo `decode_0` proporciona una salida de tamaño unitario.

Finalmente, el bloque `axi_gpio3` se utilizó para implementar la entrada del sensor de movimiento PIR al sistema, funcionando como una entrada directa de 1 bit hacia la FPGA.

## 10.5. Decode.v

A continuación, se muestra el archivo `decode.v` utilizado para la correcta impplementación del teclado matricial

Listing 1: Módulo decodificador del teclado matricial (decode.v)

```
1 `timescale 1ns / 1ps
2
3 module decode(
4     input clk_100MHz,
```

```
    input reset,              // reset activo en alto
    input [3:0] row,
    output reg [3:0] col,
    output reg [3:0] dec_out
);

    parameter LAG = 10;

    reg [19:0] scan_timer = 0;
    reg [1:0] col_select = 0;

    // --------------------------
    // SCAN TIMER + RESET
    // --------------------------
    always @(posedge clk_100MHz) begin
        if (reset) begin
            scan_timer <= 0;
            col_select <= 0;
            dec_out <= 4'h0;
            col <= 4'b1111;  // valor por defecto
        end
        else begin
            // avanzar temporizador
            if (scan_timer == 99_999) begin
                scan_timer <= 0;
                col_select <= col_select + 1;
            end
            else begin
                scan_timer <= scan_timer + 1;
            end

            // activar columnas
            case(col_select)
                2'b00: col <= 4'b0111;
                2'b01: col <= 4'b1011;
                2'b10: col <= 4'b1101;
                2'b11: col <= 4'b1110;
            endcase

            // decodificar tecla
            if (scan_timer == LAG) begin
                case(col_select)
                    2'b00: case(row)
                        4'b0111: dec_out <= 4'hD;
                        4'b1011: dec_out <= 4'hF;
                        4'b1101: dec_out <= 4'h0;
                        4'b1110: dec_out <= 4'hE;
                    endcase

                    2'b01: case(row)
                        4'b0111: dec_out <= 4'hC;
                        4'b1011: dec_out <= 4'h9;
                        4'b1101: dec_out <= 4'h8;
                        4'b1110: dec_out <= 4'h7;
                    endcase

                    2'b10: case(row)
                        4'b0111: dec_out <= 4'hB;
                        4'b1011: dec_out <= 4'h6;
                        4'b1101: dec_out <= 4'h5;
                        4'b1110: dec_out <= 4'h4;
                    endcase
```

```
67
68                      2'b11: case(row)
69                          4'b0111: dec_out <= 4'hA;
70                          4'b1011: dec_out <= 4'h3;
71                          4'b1101: dec_out <= 4'h2;
72                          4'b1110: dec_out <= 4'h1;
73                      endcase
74                  endcase
75              end
76          end
77      end
78
79  endmodule
```

El módulo `decode` implementa la lógica necesaria para la lectura y decodificación de un teclado matricial 4×4. Su función principal es realizar el barrido secuencial de las columnas del teclado y detectar la tecla presionada a partir del estado de las filas, generando como salida un código digital correspondiente a dicha tecla.

El barrido del teclado se realiza mediante un temporizador interno (`scan_timer`) que controla la activación periódica de cada columna a través de la señal `col`. De forma secuencial, se habilita una columna a la vez mientras se leen las señales de entrada `row`. Este proceso se repite continuamente, permitiendo detectar cualquier tecla presionada.

Cuando el temporizador alcanza un valor determinado (`LAG`), el módulo evalúa la combinación activa de fila y columna y asigna a la salida `dec_out` un valor hexadecimal que identifica la tecla presionada. En caso de reinicio, el módulo restablece todas sus señales internas y coloca las columnas en un estado inactivo.

De esta manera, el módulo permite realizar la decodificación del teclado directamente en hardware, reduciendo la carga de procesamiento del sistema y facilitando la lectura de teclas desde el procesador a través de los bloques AXI-GPIO.

## 10.6. AXI BRAM Controller y Block Memory Generator

El bloque *AXI BRAM Controller*, junto con el *Block Memory Generator*, implementa una memoria BRAM interna accesible desde el procesador a través del bus AXI. Esta memoria se utiliza para el almacenamiento temporal de datos, variables de control o información intercambiada entre el software y la lógica programable, ofreciendo un acceso rápido y determinístico.

## 10.7. Módulo personalizado *decode_v1_0*

El bloque *decode_v1_0* corresponde a un módulo personalizado implementado en la lógica programable. Su función principal es realizar el procesamiento específico de señales provenientes del teclado matricial, decodificando las combinaciones de filas y columnas y generando un código digital que identifica la tecla presionada.

Este procesamiento en hardware permite reducir la carga del procesador y mejorar el tiempo de respuesta del sistema.

## 10.8. Processor System Reset

El bloque *Processor System Reset* se encarga de gestionar las señales de reinicio de los distintos módulos del sistema. Este módulo garantiza que tanto el bus AXI como los periféricos y la lógica programable se inicialicen correctamente durante el arranque o ante un reinicio, evitando estados indeterminados en el sistema.

## 10.9. Conexiones externas (XDC)

Las señales de entrada y salida del sistema se conectan a los pines físicos de la Zybo Z7 mediante puertos externos definidos en el diseño y asociados en el archivo de restricciones (*XDC*). Esto permite la conexión directa de periféricos externos como el teclado matricial, el sensor PIR, el buzzer y la pantalla LCD, integrando el sistema con el entorno físico de operación.

Listing 2: Archivo .XDC utilizado para la implementación de los pines físicos de la FPGA Zybo Z7 Zybo-Z7-Master.xdc)

```
1
2  ##Pmod Header JC
3  set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33     } [get_ports { row_0[0] }]; #
       IO_L10P_T1_34 Sch=jc_p[1]
4  set_property PULLUP true [get_ports { row_0[0] }]
5  set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33     } [get_ports { row_0[1] }]; #
       IO_L10N_T1_34 Sch=jc_n[1]
6  set_property PULLUP true [get_ports { row_0[1] }]
7  set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33     } [get_ports { row_0[2] }]; #
       IO_L1P_T0_34 Sch=jc_p[2]
8  set_property PULLUP true [get_ports { row_0[2] }]
9  set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33     } [get_ports { row_0[3] }]; #
       IO_L1N_T0_34 Sch=jc_n[2]
10 set_property PULLUP true [get_ports { row_0[3] }]
11
12 set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {col_0_0[0]}]; #
       IO_L8P_T1_34 Sch=jc_p[3]
13 set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33 } [get_ports {col_0_0[1]}]; #
       IO_L8N_T1_34 Sch=jc_n[3]
14 set_property -dict { PACKAGE_PIN T12   IOSTANDARD LVCMOS33 } [get_ports {col_0_0[2]}];
15 set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports {col_0_0[3]}];
16
17 ##Pmod Header JD
18
19 set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports {GPIO_1_tri_i[0]}];
20 set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {IIC_0_0_scl_io}];
21 set_property PULLUP true [get_ports {IIC_0_0_scl_io}]
22 set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {IIC_0_0_sda_io}];
23 set_property PULLUP true [get_ports {IIC_0_0_sda_io}]
24 set_property -dict { PACKAGE_PIN V18   IOSTANDARD LVCMOS33     } [get_ports {GPIO_0_tri_o
       [0]}]; #IO_L2P_T0_34 Sch=jc_p[4]
```

# 11. SOFTWARE DEL PROCESADOR

En esta sección se presenta el código fuente desarrollado en lenguaje `C` para la programación del procesador del sistema. El software implementado se encarga de la comunicación con los periféricos, el procesamiento de las señales adquiridas, la gestión de la interfaz de usuario y la toma de decisiones del sistema, tales como la activación de alarmas y la visualización de información. Asimismo, se describen las principales funciones utilizadas para garantizar el correcto funcionamiento del sistema integrado. Se explicarán las funciones principales implementadas, con el fin de no alargar innecesariamente el presente documento.

El funcionamiento del software del sistema se basa en la interacción entre los archivos `Funciones.c`, `Funciones.h` y `main.c`. El archivo `Funciones.h` define la interfaz del sistema mediante la declaración de prototipos de funciones y constantes, mientras que `Funciones.c` contiene la implementación de dichas funciones, encargadas de la comunicación con los periféricos y el procesamiento de datos. Por su parte, el archivo `main.c` actúa como programa principal, inicializando el hardware y coordinando la ejecución del sistema a través de llamadas a las funciones implementadas, integrando de manera ordenada el software con la arquitectura del procesador.

## 11.1. Funciones.h

Listing 3: Archivo de cabecera Funciones.h

```
#include "xgpio.h"
#include "xiicps.h"
#include "xparameters.h"
#include "xil_printf.h"
#include "sleep.h"
#include <stdlib.h>
#include <string.h>

#define I2C_DEVICE_ID          XPAR_XIICPS_0_DEVICE_ID
#define QMC5883_ADDR           0x0D
#define LCD_ADDR               0x27

#define GPIO_VERILOG_ID        XPAR_AXI_GPIO_2_DEVICE_ID
#define VERILOG_CHANNEL        1

/* --- Funciones para magnetometro --- */
void mostrar_lectura_magnetometro(int *x, int *y, int *z);
int QMC_read6(u8 *buf);
int QMC_writeReg(u8 reg, u8 value);
int QMC_init();
```

```
int writeReg_sensor(u8 reg, u8 value);
int readBytes_sensor(u8 reg, u8 *buffer, int len);

/* --- Funciones para LCD --- */
void lcd_init();
void lcd_clear();
void lcd_set_cursor(int row, int col);
void lcd_print(const char *s);
void lcd_char(char c);
void lcd_cmd(u8 cmd);
void lcd_send_byte(u8 value, u8 mode);
void lcd_send_nibble(u8 nib, u8 mode);
void lcd_i2c_write(u8 data);
void lcd_set_brightness(int percent);

/* --- Funciones de teclado y control de usuario --- */
u32 leer_tecla(XGpio *GpioVerilog);
void crear_usuario_y_contrasena(XGpio *teclado);
int cambiar_contrasena(XGpio *teclado);
int comprobar_usuario(XGpio *teclado);
int mostrar_menu(XGpio *teclado);
int menu_con_login(XGpio *teclado);
void solicitar_usuario();

/* --- Comunicacion UART --- /
int check_uart_code_11();
void mostrar_fecha_desde_uart();

/* --- Funciones auxiliares --- */
char* addCharMalloc(char *buffer, int *size, u32 data);
int writeI2C(u8 addr, u8 *data, int len);
int readI2C(u8 addr, u8 reg, u8 *buffer, int len);
void welcome();
```

El archivo `Funciones.h` actúa como archivo de cabecera del proyecto y contiene las definiciones, constantes y prototipos de las funciones utilizadas para la interacción con los periféricos del sistema. En él se declaran las funciones encargadas de la comunicación con el magnetómetro mediante I2C, el control y visualización de información en la pantalla LCD, la lectura del teclado matricial, la gestión de usuarios y contraseñas, así como rutinas auxiliares para la comunicación por UART. Este archivo permite organizar el código de forma modular y facilita la reutilización de funciones dentro del programa principal.

## 11.2. Funciones.c

Luego, está el código Funciones.c, el cual cuenta con la logica de cada función implementada:

Listing 4: Inicialización y función auxiliar para manejo dinámico de cadenas en Funciones.c

```
#include "Funciones.h"
#include <math.h>

#define MAX_USER_LEN 20
#define UART_BUFFER_SIZE 10

XIicPs Iic;
char* addCharMalloc(char *buffer, int *size, u32 data)
{
    if (buffer == NULL) {
        buffer = (char*)malloc(2);
        buffer[0] = (char)data;
        buffer[1] = '\0';
        *size = 1;
        return buffer;
    }

    char *temp = (char*)realloc(buffer, (*size + 2));
    if (temp == NULL) {
        return buffer;
    }

    buffer = temp;
    buffer[*size] = (char)data;
    (*size)++;
    buffer[*size] = '\0';

    return buffer;
}
```

Esta primera sección del archivo `Funciones.c` incluye las librerías necesarias y define constantes utilizadas a lo largo del sistema, como el tamaño máximo del nombre de usuario y el buffer para la comunicación por UART. Además, se declara la variable global `XIicPs Iic`, que permite compartir el controlador I2C entre el programa principal y las funciones auxiliares.

La función `addCharMalloc` se encarga de gestionar dinámicamente la creación y ampliación de cadenas de caracteres. Su objetivo es agregar un carácter a un buffer de forma segura, utilizando memoria dinámica mediante `malloc` y `realloc`. Esta función resulta especialmente útil para construir

cadenas a partir de entradas secuenciales, como las teclas presionadas por el usuario, garantizando que el string permanezca correctamente terminado en todo momento.

Listing 5: Funciones de escritura y lectura I2C y definición de registros del sensor

```
int writeI2C(u8 addr, u8 *data, int len)
{
    int status = XIicPs_MasterSendPolled(&Iic, data, len, addr);
    while (XIicPs_BusIsBusy(&Iic));
    return status;
}

int readI2C(u8 addr, u8 reg, u8 *buffer, int len)
{
    int status;
    status = XIicPs_MasterSendPolled(&Iic, &reg, 1, addr);
    while (XIicPs_BusIsBusy(&Iic));
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XIicPs_MasterRecvPolled(&Iic, buffer, len, addr);
    while (XIicPs_BusIsBusy(&Iic));
    return status;
}

#define REG_X_LSB     0x00
#define REG_CTRL1     0x09
#define REG_CTRL2     0x0A
#define REG_RESET     0x0B
```

Estas funciones permiten la comunicación I2C con el sensor, realizando operaciones de escritura y lectura de registros de forma bloqueante. Además, se definen las direcciones de los registros principales utilizados para la configuración y lectura de datos del magnetómetro.

Listing 6: Funciones básicas de control y escritura en la pantalla LCD

```
u8 lcd_backlight = 0x08;

void lcd_clear()
{
    lcd_cmd(0x01);    // Limpia la pantalla
    usleep(2000);     // Tiempo necesario para completar el comando
}

void lcd_cmd(u8 cmd)
{
    lcd_send_byte(cmd, 0x00);
```

```
    usleep(2000);
}

void lcd_char(char c)
{
    lcd_send_byte(c, 0x01);
}

void lcd_print(const char *s)
{
    while (*s) lcd_char(*s++);
}

void lcd_set_cursor(int row, int col)
{
    const u8 offsets[2] = {0x00, 0x40};
    lcd_cmd(0x80 | (offsets[row] + col));
}
```

Este conjunto de funciones permite controlar la pantalla LCD, enviando comandos, caracteres y cadenas de texto, así como posicionar el cursor en filas y columnas específicas. Además, se gestiona el borrado de la pantalla respetando los tiempos requeridos por el controlador LCD.

Listing 7: Escritura de registros y lectura del magnetómetro

```
int writeReg_sensor(u8 reg, u8 value)
{
    u8 data[2] = {reg, value};
    int status;

    status = XIicPs_MasterSendPolled(&Iic, data, 2, QMC5883_ADDR);
    while (XIicPs_BusIsBusy(&Iic));

    return (status == XST_SUCCESS) ? XST_SUCCESS : XST_FAILURE;
}

void mostrar_lectura_magnetometro(int *x, int *y, int *z)
{
    u8 buf[6];
    s16 xx, yy, zz;

    if (QMC_read6(buf) == XST_SUCCESS) {
```

```
        xx = (buf[1] << 8) | buf[0];
        yy = (buf[3] << 8) | buf[2];
        zz = (buf[5] << 8) | buf[4];

        *x = xx;
        *y = yy;
        *z = zz;

        xil_printf("1,%d,%d,%d\r\n", xx, yy, zz);

        char sx[8], sy[8], sz[8];
        itoa(xx, sx, 10);
        itoa(yy, sy, 10);
        itoa(zz, sz, 10);

        lcd_set_cursor(0, 0);
        lcd_print("X:");
        lcd_print(sx);

        lcd_set_cursor(1, 0);
        lcd_print("Y:");
        lcd_print(sy);

        lcd_set_cursor(1, 8);
        lcd_print("Z:");
        lcd_print(sz);

    }
    else {
        xil_printf("ERROR␣leyendo␣QMC5883\r\n");
        lcd_set_cursor(0, 0);
        lcd_print("Error␣lectura");

        *x = 0;
        *y = 0;
        *z = 0;
    }
}
```

Estas funciones permiten configurar el magnetómetro mediante escritura I2C y leer sus valores de campo magnético en los ejes X, Y y Z. Los datos obtenidos se envían por UART, se muestran en la pantalla LCD y se retornan al programa principal mediante punteros.

Listing 8: Lectura del teclado y creación de usuario y contraseña

```
u32 leer_teclado(XGpio *GpioVerilog)
{
    XGpio_DiscreteWrite(GpioVerilog, 1, 0);

    u32 value = XGpio_DiscreteRead(GpioVerilog, 2);
    value &= 0xF;

    usleep(200200);

    if (value != 0) {
        XGpio_DiscreteWrite(GpioVerilog, 1, 1);
        return value;
    }

    usleep(20000);

    for (volatile int i = 0; i < 1000000; i++);

    return 0;
}

void crear_usuario_y_contrasena(XGpio *teclado)
{
    char usuario[20];
    char contrasena[20];
    int idx = 0;

    lcd_clear();
    lcd_set_cursor(0,0);
    lcd_print("Crear␣Usuario:");
    lcd_set_cursor(1,0);

    idx = 0;
    usuario[0] = '\0';

    while (1) {
        u32 tecla = leer_teclado(teclado);
        if (tecla == 0) continue;

        if (tecla == 0xD) {
```

```
            usuario[idx] = '\0';
            break;
        }

        if (idx < 19) {
            char c;
            sprintf(&c, "%X", tecla);
            usuario[idx++] = c;
            usuario[idx] = '\0';

            lcd_set_cursor(1,0);
            lcd_print(usuario);
        }

        usleep(150000);
    }

    lcd_clear();
    lcd_set_cursor(0,0);
    lcd_print("Crear␣Pass:");
    lcd_set_cursor(1,0);

    idx = 0;
    contrasena[0] = '\0';

    while (1) {
        u32 tecla = leer_teclado(teclado);
        if (tecla == 0) continue;

        if (tecla == 0xD) {
            contrasena[idx] = '\0';
            break;
        }

        if (idx < 19) {
            char c;
            sprintf(&c, "%X", tecla);
            contrasena[idx++] = c;
            contrasena[idx] = '\0';

            lcd_set_cursor(1,0);
```

```
            lcd_print(contrasena);
        }

        usleep(150000);
    }

    xil_printf("2,%s,%s\r\n", usuario, contrasena);

    lcd_clear();
    lcd_set_cursor(0,0);
    lcd_print("Guardado!");
    usleep(400000);
}
```

Estas funciones permiten leer las teclas del teclado matricial mediante GPIO y capturar un nombre de usuario y una contraseña ingresados por el usuario. La información se visualiza en la pantalla LCD y posteriormente se envía por UART para su almacenamiento o verificación.

Listing 9: Menú principal de selección de opciones

```
int mostrar_menu(XGpio *teclado)
{
    lcd_clear();
    lcd_set_cursor(0,0);
    lcd_print("1)Registrarse");
    lcd_set_cursor(1,0);
    lcd_print("2)Iniciar Sesion");

    while (1)
    {
        u32 tecla = leer_teclado(teclado);

        if (tecla == 0)
            continue;

        if (tecla == 0x1)
            return 1;
        if (tecla == 0x2)
            return 2;

        usleep(150000);
    }
}
```

Esta función muestra un menú en la pantalla LCD y espera la selección del usuario mediante el teclado matricial. Dependiendo de la tecla presionada, retorna la opción elegida para controlar el flujo del programa principal.

Listing 10: Menú de opciones con autenticación

```
int menu_con_login(XGpio *teclado)
{
    const char *opciones[] = {
        " Medir CampoMAG",
        " TEST Seguridad",
        " Info Usuario",
        " Cambiar Clave",
        " Ultimo Riesgo",
        " Cerrar Sesion"
    };

    int total = 6;
    int pos = 0;

    while (1)
    {
        lcd_clear();

        lcd_set_cursor(0,0);
        lcd_print(">");
        lcd_print(opciones[pos]);

        lcd_set_cursor(1,0);
        if (pos + 1 < total)
            lcd_print(opciones[pos + 1]);
        else
            lcd_print("                ");

        u32 tecla = leer_teclado(teclado);

        if (tecla == 0) {
            usleep(80000);
        }

        if (tecla == 0xA) {
            if (pos > 0)
```

```
                pos--;
            }
            else if (tecla == 0xB) {
                if (pos < total - 1)
                    pos++;
            }
            else if (tecla == 0xD) {
                return pos + 1;
            }

            usleep(150000);
        }
}
```

Esta función implementa un menú interactivo que permite al usuario navegar entre distintas opciones utilizando el teclado matricial. La selección se realiza mediante teclas de desplazamiento y confirmación, retornando la opción elegida al programa principal.

## 11.3. main.c

Los siguientes códigos pertenecen al archivo main.c, el cual cuenta con la interacción entre funciones mediante las entradas que de el usuario:

Listing 11: Inclusión de librerías y definiciones globales en main.c

```
#include "xgpio.h"
#include "xparameters.h"
#include "xil_printf.h"
#include "xiicps.h"
#include "sleep.h"
#include "Funciones.h"

extern XIicPs Iic;

#define I2C_DEVICE_ID          XPAR_XIICPS_0_DEVICE_ID
#define QMC5883_ADDR           0x0D

#define GPIO_VERILOG_ID        XPAR_AXI_GPIO_2_DEVICE_ID
#define GPIO_BUZZER            XPAR_AXI_GPIO_1_DEVICE_ID
#define GPIO_MOVIMIENTO        XPAR_AXI_GPIO_3_DEVICE_ID

#define VERILOG_CHANNEL        1
```

Este bloque incluye las librerías necesarias y define constantes globales para la identificación de los periféricos utilizados en el sistema. Además, declara el controlador I2C como variable externa para permitir su uso desde el programa principal.

Listing 12: Inicialización de GPIOs y periféricos principales

```
int main() {
    XGpio GpioMovimiento;
    XGpio GpioVerilog;
    XGpio buzzer_verilog;

    XGpio_Initialize(&buzzer_verilog, GPIO_BUZZER);
    XGpio_SetDataDirection(&buzzer_verilog,VERILOG_CHANNEL,0x00000000);

    XGpio_Initialize(&GpioMovimiento, GPIO_MOVIMIENTO);
    XGpio_SetDataDirection(&GpioMovimiento, 1, 0xFFFFFFFF);

    XGpio_Initialize(&GpioVerilog, GPIO_VERILOG_ID);
    XGpio_SetDataDirection(&GpioVerilog, 1, 0xFFFFFFFF);
```

Este bloque inicializa los GPIO utilizados para el buzzer, el sensor de movimiento PIR y el teclado matricial, configurando correctamente sus direcciones de entrada y salida.

Listing 13: Inicialización de I2C, LCD y magnetómetro

```
    XIicPs_Config *Cfg = XIicPs_LookupConfig(I2C_DEVICE_ID);
    XIicPs_CfgInitialize(&Iic, Cfg, Cfg->BaseAddress);
    XIicPs_SetSClk(&Iic, 100000);

    lcd_init();
    lcd_set_brightness(300);

    QMC_init();

    int logged = 0;
```

Aquí se configura el bus I2C, se inicializa la pantalla LCD y se pone en funcionamiento el magnetómetro, dejando el sistema listo para operar.

Listing 14: Control de registro e inicio de sesión

```
    while (1)
    {
        if (!logged)
        {
            int opcion = mostrar_menu(&GpioVerilog);
```

```
        if (opcion == 1)
            crear_usuario_y_contrasena(&GpioVerilog);
        else if (opcion == 2)
            logged = comprobar_usuario(&GpioVerilog);

        continue;
    }
```

Este bloque gestiona el registro y la autenticación del usuario antes de permitir el acceso a las funciones principales del sistema.

Listing 15: Medición del campo magnético y detección de peligro

```
int seleccion = menu_con_login(&GpioVerilog);

if (seleccion == 1)
{
    lcd_clear();

    while (1)
    {
        int x, y, z;

        mostrar_lectura_magnetometro(&x, &y, &z);
        u32 t = leer_teclado(&GpioVerilog);
        u32 valor = XGpio_DiscreteRead(&GpioMovimiento, 1);

        if (valor) {
            lcd_clear();
            lcd_print("  MOVIMIENTO");
            XGpio_DiscreteWrite(&buzzer_verilog, 1, 1);
            usleep(2000000);
            lcd_clear();
            XGpio_DiscreteWrite(&buzzer_verilog, 1, 0);
        }

        int peligro = 0;

        if (x < -4000 || (x > -10 && x < 10) || x > 4000)
        peligro = 1;
        if (y < -4000 || (y > -10 && y < 10) || y > 4000)
        peligro = 1;
```

```
            if (z < -4000 || (z > -10 && z < 10) || z > 4000)
            peligro = 1;

            if (peligro)
            {
                lcd_clear();
                lcd_print("PELIGRO");
                usleep(2000000);
                lcd_clear();
                XGpio_DiscreteWrite(&buzzer_verilog, 1, 1);
            }

            if (t == 0x2)
                XGpio_DiscreteWrite(&buzzer_verilog, 1, 0);

            if (t == 0xA)
                break;

            usleep(200000);
        }
    }
```

Este bloque realiza la medición continua del campo magnético, detecta condiciones de riesgo según rangos definidos y activa alarmas visuales y sonoras ante eventos peligrosos o movimiento detectado.

Listing 16: Opciones adicionales del menú y cierre de sesión

```
else if (seleccion == 2)
{
    XGpio_DiscreteWrite(&buzzer_verilog, 1, 1);
    lcd_clear();
    lcd_print("BUZZER ON");
    usleep(1000000);
    XGpio_DiscreteWrite(&buzzer_verilog, 1, 0);
}
else if (seleccion == 3)
{
    solicitar_usuario();
}
else if (seleccion == 4)
{
    cambiar_contrasena(&GpioVerilog);
}
```

```
        else if (seleccion == 5)
        {
            mostrar_fecha_desde_uart();
        }
        else if (seleccion == 6)
        {
            lcd_clear();
            lcd_set_cursor(0,0);
            lcd_print("Sesion␣cerrada");
            usleep(1500000);
            logged = 0;
            continue;
        }
    }

    return 0;
}
```

Este bloque implementa las funciones adicionales del sistema, como pruebas del buzzer, gestión de usuario, consulta de información y cierre seguro de sesión.

Finalmente, se presenta el diagrama de flujo simplificado del sistema implementado, el cual condensa la información explicada previamente

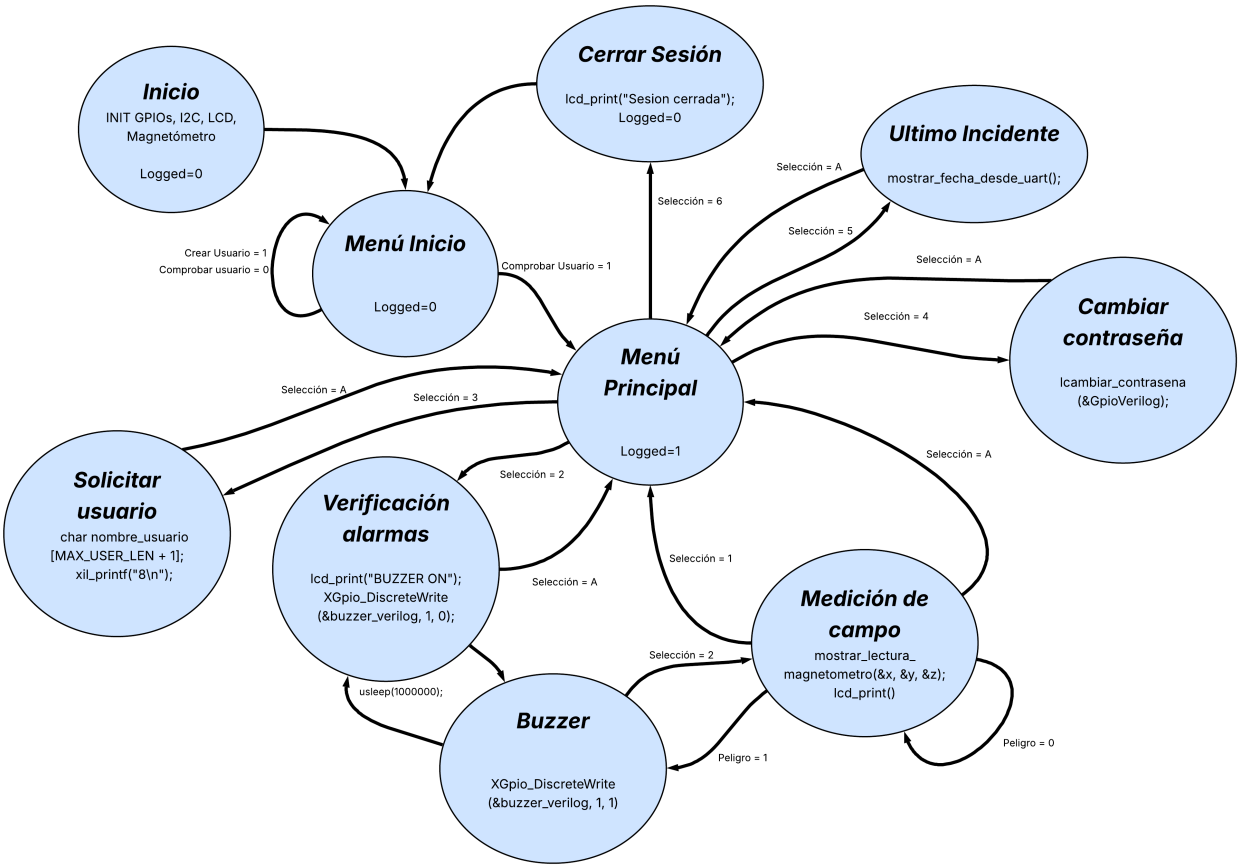


Figura 9: Diagrama de flujo del sistema de seguridad de campos magnéticos

## 11.4. Aplicación de Python

## 11.5. Funcionalidades

La aplicación desarrollada en Python constituye la interfaz de supervisión y análisis del sistema de seguridad para campos magnéticos, permitiendo la interacción directa entre el usuario y la información generada por el sistema embebido basado en FPGA. En primer lugar, la aplicación recibe en tiempo real las mediciones de los ejes $X$, $Y$ y $Z$ del campo magnético a través de comunicación serial, procesando estos valores para calcular la magnitud instantánea del campo, la cual se muestra de forma numérica y gráfica. Cuando dicha magnitud supera los umbrales de seguridad definidos, el sistema identifica la condición como un evento peligroso y lo registra automáticamente como un incidente.

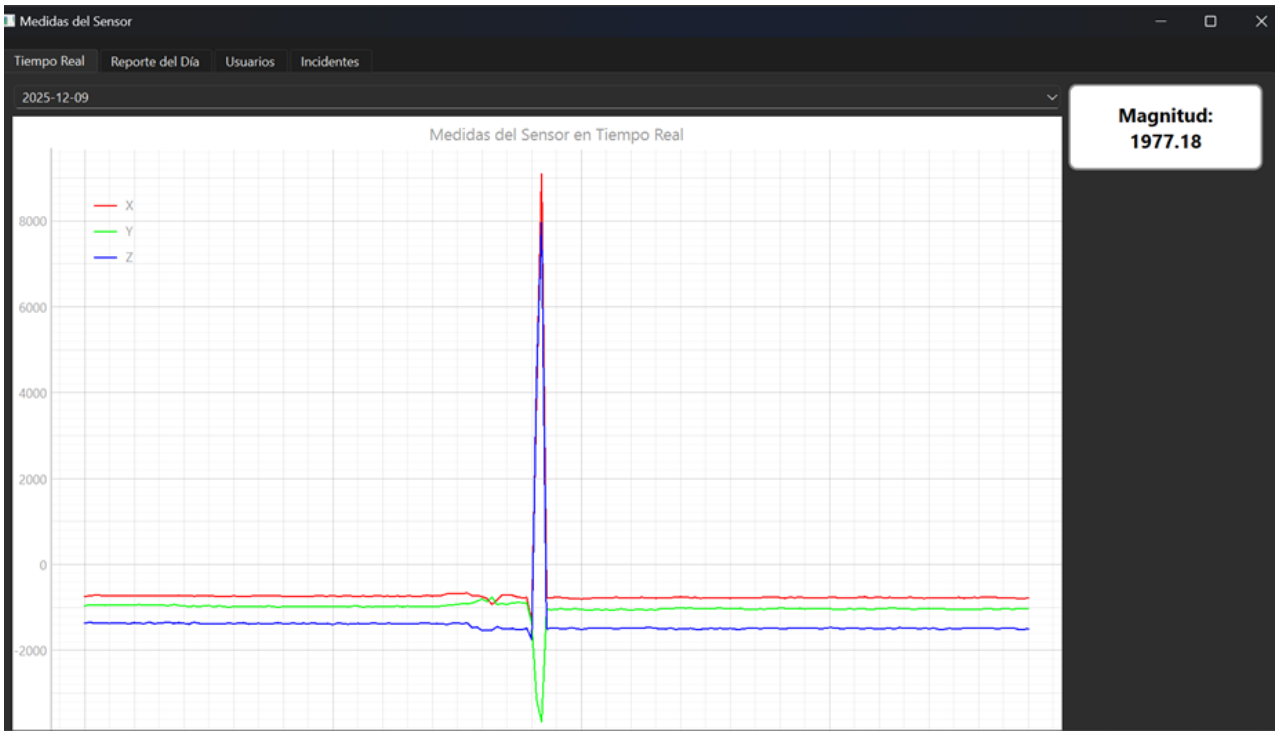


Figura 10: Medición del campo magnético en X Y Z durante el tiempo - Aplicación

Adicionalmente, la aplicación ofrece un módulo de visualización histórica organizado por días, en el cual se presenta el comportamiento temporal del campo magnético mediante gráficas que permiten evidenciar variaciones, picos y perturbaciones a lo largo del tiempo. De igual forma, se incorpora una gestión de usuarios basada en una base de datos, donde se almacenan los identificadores de usuario junto con sus contraseñas protegidas mediante encriptación *SHA-256*, garantizando un nivel básico de seguridad de la información. Finalmente, la interfaz incluye un registro detallado de incidentes, en el que se listan los eventos de campos magnéticos peligrosos detectados, indicando la fecha, los valores medidos y la condición de riesgo, consolidando así una herramienta integral de monitoreo, análisis y control del sistema.

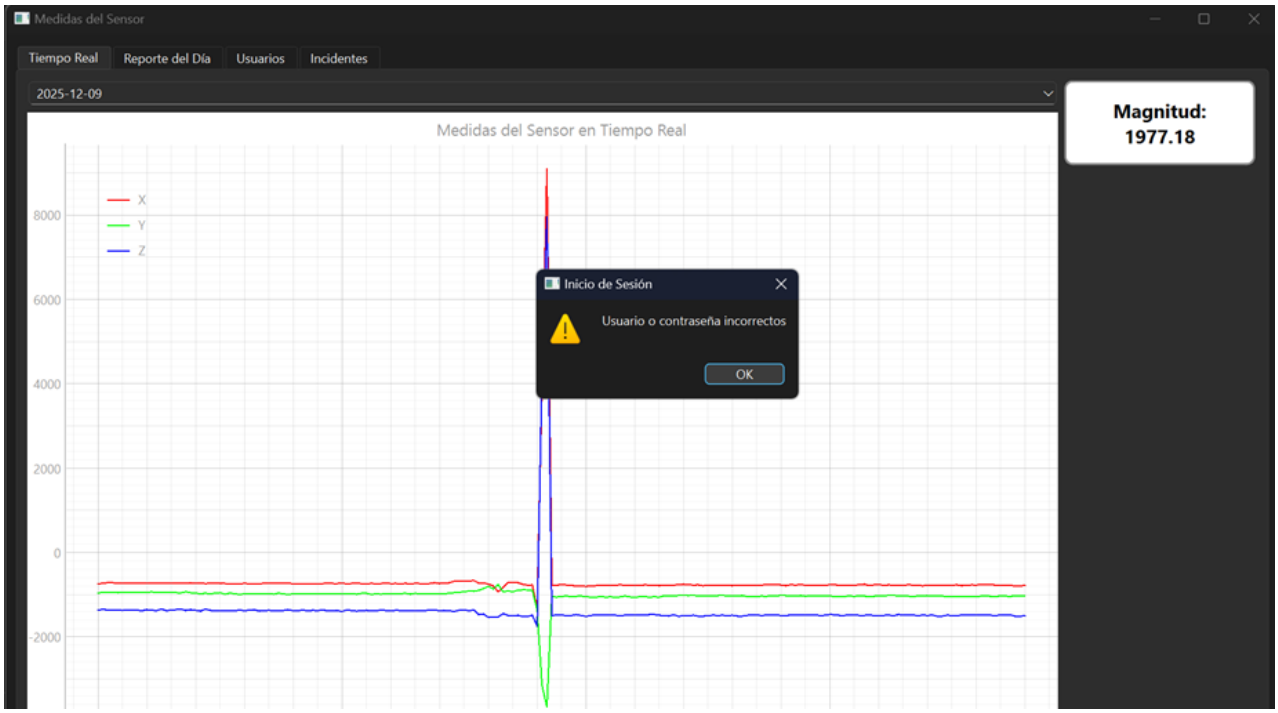


Figura 11: Alerta de nivel alto de campo magnético - Aplicación

## 11.6. Interfaz

La interfaz gráfica de la aplicación presenta de manera clara e intuitiva la información asociada a la medición del campo magnético. En la sección principal se muestra una gráfica temporal que representa la evolución de las componentes $X$, $Y$ y $Z$ del campo magnético, permitiendo al usuario observar su comportamiento a lo largo del tiempo e identificar variaciones abruptas o picos significativos. Cada componente se visualiza con un color distinto, facilitando su diferenciación y análisis comparativo.

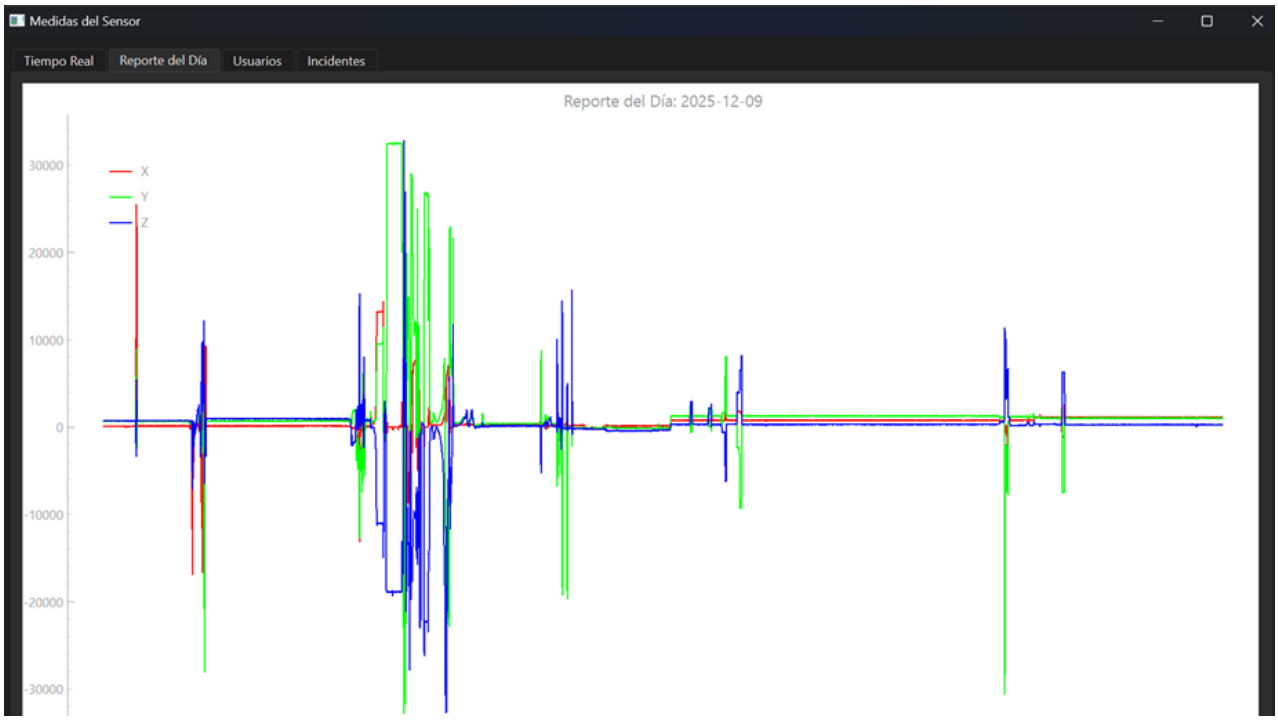


Figura 12: Registro histórico de niveles de campo magnético - Aplicación

Adicionalmente, la interfaz incluye un indicador numérico que muestra la magnitud instantánea del campo magnético, calculada a partir de las tres componentes medidas, lo que proporciona una referencia directa del nivel de exposición en tiempo real. Esta representación gráfica y numérica conjunta permite evaluar de forma inmediata tanto el valor absoluto del campo como su dinámica temporal, contribuyendo a una mejor interpretación de las condiciones del entorno monitoreado.

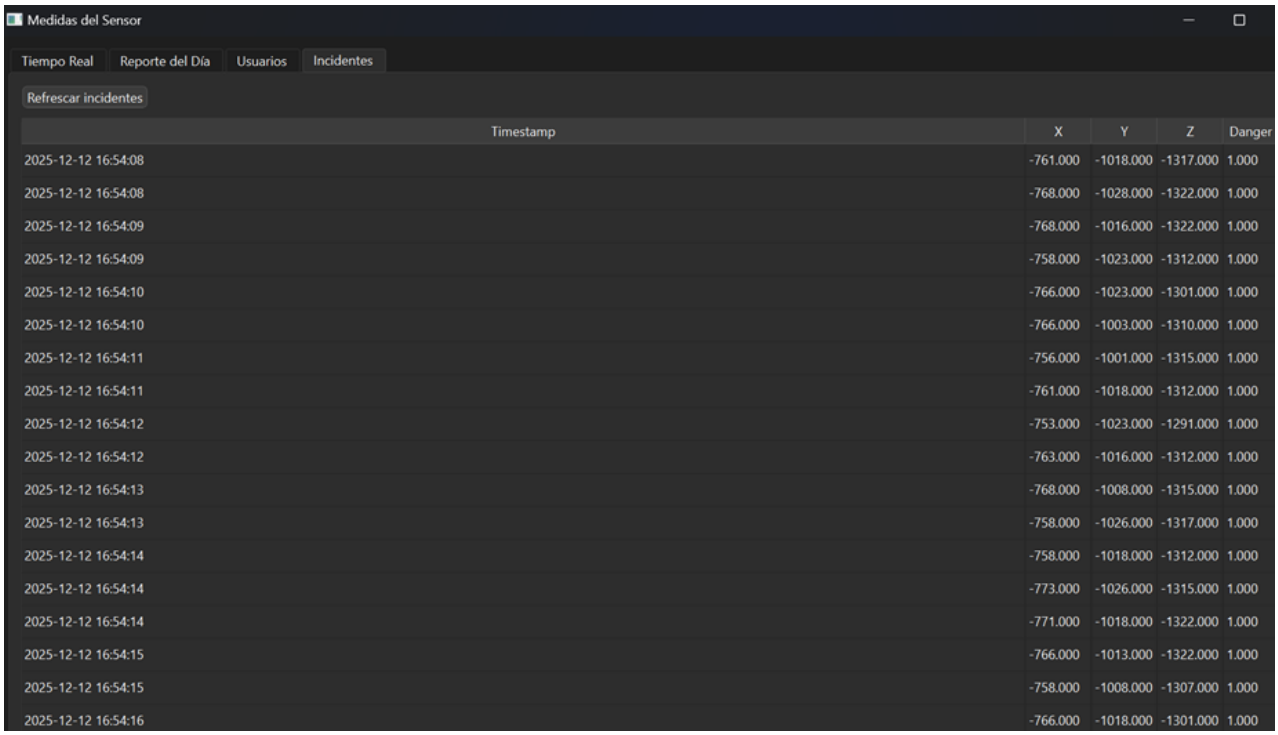


Figura 13: Lista de incidentes presentados - Aplicación

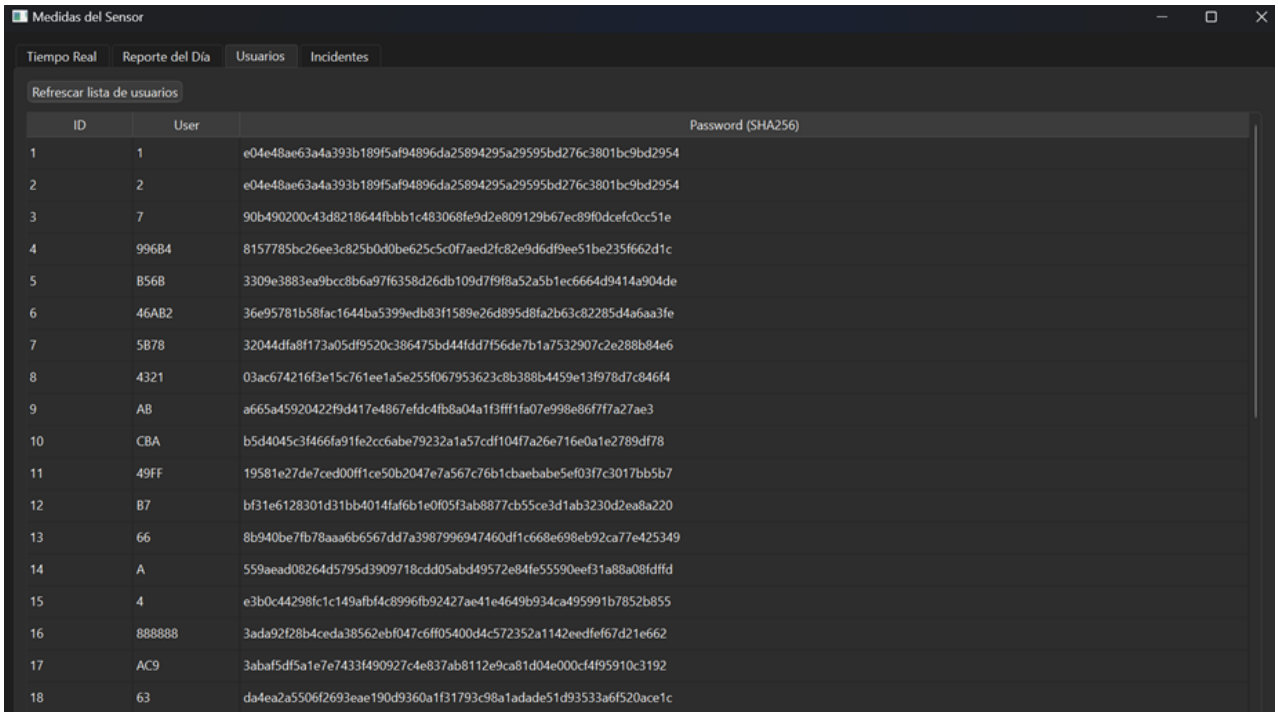


Figura 14: Base de datos con información de los usuarios - Aplicación

## 11.7. app.py

El archivo `app.py` implementa una interfaz gráfica en Python utilizando las librerías *PyQt6* y *PyQtGraph* para la visualización y análisis de los datos medidos por el sistema de monitoreo de campos magnéticos. La aplicación construye una ventana principal con una pestaña dedicada al reporte del sensor, en la cual se presenta una gráfica temporal de las componentes X, Y y Z del campo magnético, junto con dos tablas informativas: una tabla de incidentes registrados y una tabla de estadísticas. Los datos son obtenidos desde un módulo auxiliar (`Funciones`) y procesados mediante *NumPy* para el cálculo de valores estadísticos como el promedio, el valor máximo y el valor mínimo. Esta interfaz permite al usuario analizar tendencias, identificar eventos de riesgo y consultar información histórica de manera clara e intuitiva.

## 11.8. read_serial.py

El archivo `read_serial.py` implementa la comunicación serial entre el sistema embebido basado en FPGA y la aplicación de supervisión en PC, utilizando el protocolo UART. Este script se encarga de recibir en tiempo real las mediciones del campo magnético enviadas desde la FPGA, procesar los datos correspondientes a los ejes X, Y y Z, calcular la magnitud del campo y determinar condiciones de riesgo según umbrales definidos. Adicionalmente, el programa gestiona la interacción con una base de datos SQLite para almacenar mediciones, incidentes y usuarios, y presenta la

información mediante una interfaz gráfica desarrollada con *PyQt6* y *PyQtGraph*. El sistema también permite el intercambio bidireccional de comandos con la FPGA para funciones como autenticación de usuarios, activación de alertas y consulta de eventos, consolidando así un entorno completo de monitoreo, análisis y visualización de seguridad.

## 11.9. read_data.py

El archivo `read_data.py` se encarga de la consulta y visualización directa de la información almacenada en la base de datos del sistema. Mediante el uso de la librería *SQLite3*, el programa establece una conexión con la base de datos `data.db` y extrae todos los registros correspondientes a las mediciones del campo magnético almacenadas en la tabla `medidas`. Los datos obtenidos se presentan de forma estructurada en la consola utilizando la librería *tabulate*, lo que facilita su inspección y verificación. Adicionalmente, el script invoca funciones del módulo `Funciones.py` para mostrar la información de los usuarios registrados, constituyendo una herramienta de apoyo para depuración, validación de datos y análisis externo del sistema.

## 11.10. Funciones.py

El archivo `Funciones.py` implementa la lógica de gestión de datos y usuarios del sistema de monitoreo, actuando como intermediario entre la aplicación gráfica y la base de datos. En este módulo se definen funciones para la creación y verificación de usuarios mediante contraseñas cifradas con *hash SHA-256*, así como para el almacenamiento y consulta de las mediciones del campo magnético en una base de datos SQLite. Adicionalmente, se incluyen rutinas para identificar incidentes de riesgo en función de la magnitud del campo medido, generar estadísticas diarias de los ejes X, Y y Z, y mantener el estado del usuario autenticado durante la sesión. De esta manera, el módulo centraliza el manejo seguro de la información y facilita su visualización y análisis desde la interfaz gráfica.

## 11.11. Maqueta final

Finalmente, para la presentación del proyecto se desarrolló una maqueta con una estructura representativa de un entorno industrial, similar a una posible fábrica en la cual podrían presentarse niveles elevados de campo magnético. Esta maqueta permitió contextualizar el funcionamiento del sistema propuesto en un escenario realista, facilitando la demostración práctica de las capacidades de monitoreo, detección de riesgo y respuesta del sistema, y complementando de manera integral el ejercicio de diseño e implementación realizado a lo largo del proyecto.

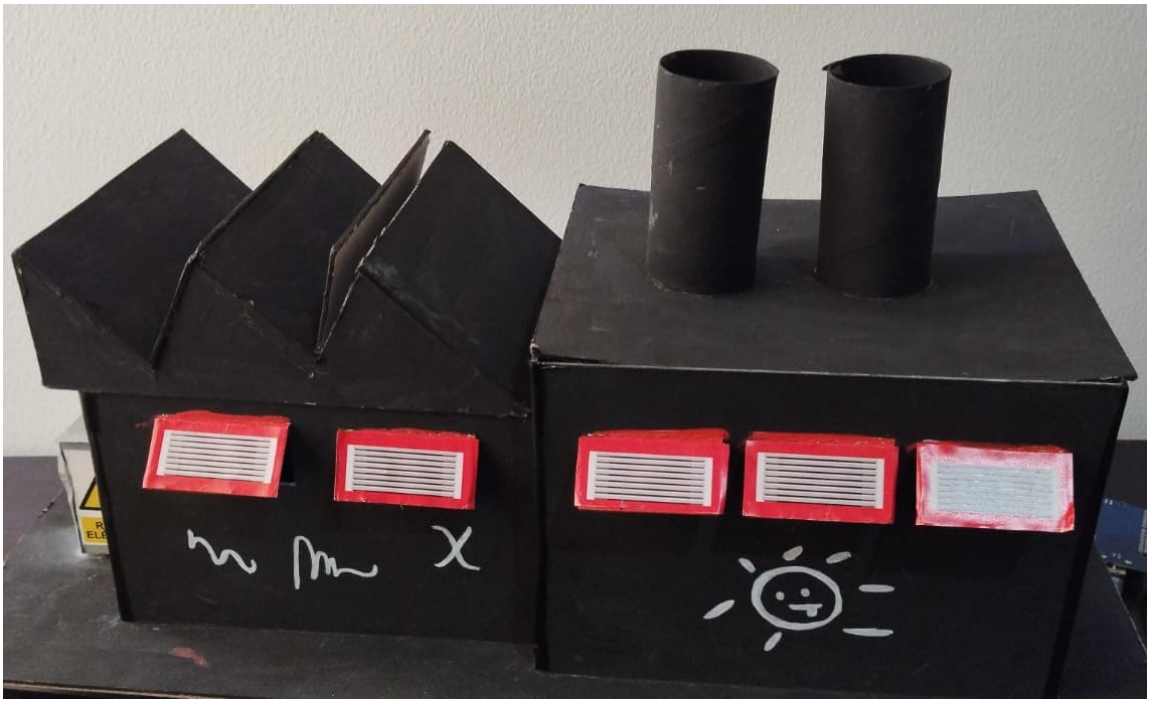

Figura 15: Maqueta a escala de una fabrica

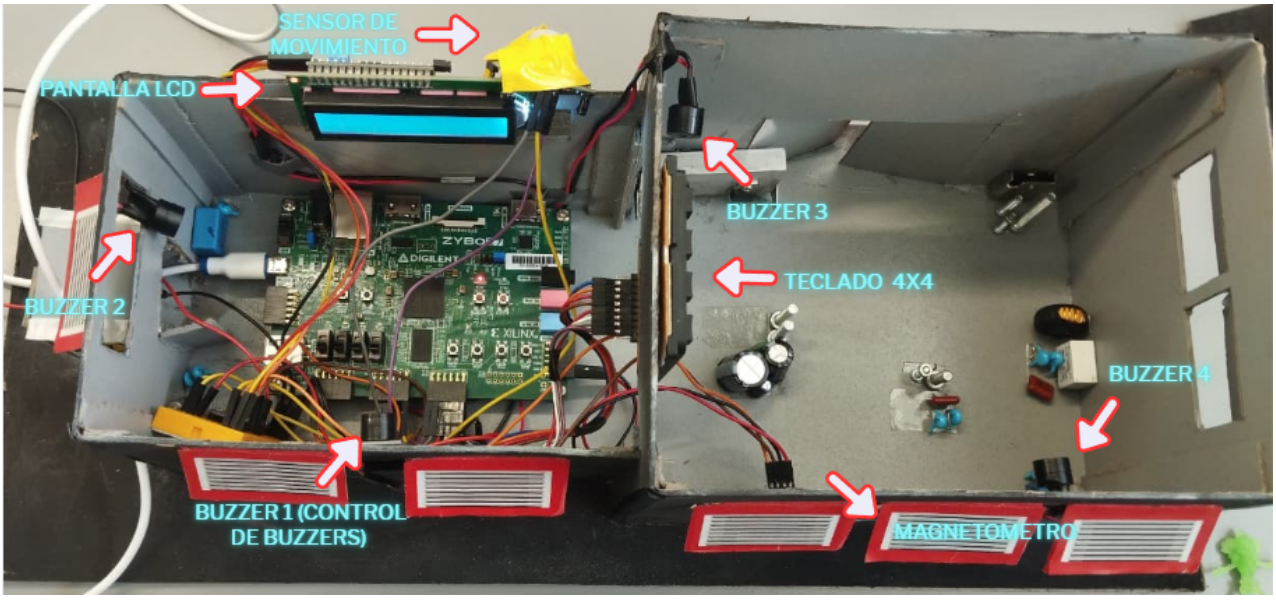


Figura 16: Montaje de los componentes internos de la fabrica a escala

# 12. CONCLUSIONES

- En el desarrollo del proyecto se logró diseñar e implementar un sistema digital funcional para el monitoreo de campos magnéticos, integrando sensores, periféricos de entrada y salida, y mecanismos de alerta sobre una plataforma basada en la arquitectura Zynq. El sistema permitió adquirir, procesar y visualizar información en tiempo real, cumpliendo con los objetivos propuestos inicialmente.

- La utilización del magnetómetro HMC5883L permitió medir de forma confiable las componentes del campo magnético en los ejes X, Y y Z, así como detectar condiciones de riesgo mediante la comparación con umbrales definidos. La integración del sensor mediante el protocolo I2C demostró ser eficiente y adecuada para aplicaciones de instrumentación y monitoreo continuo.

- La incorporación de un sensor de movimiento PIR como mecanismo adicional de seguridad fortaleció el sistema, permitiendo detectar la presencia de personas en zonas de posible exposición a campos magnéticos elevados. Esta redundancia mejora la protección del usuario y aumenta la confiabilidad del sistema ante escenarios reales de riesgo.

- El uso de la lógica programable para implementar el decodificador del teclado matricial permitió reducir la carga de procesamiento del procesador ARM, evidenciando las ventajas del enfoque hardware–software co-diseñado. La comunicación mediante bloques AXI-GPIO facilitó la interacción entre la lógica programable y el software desarrollado en Vitis.

- Finalmente, el sistema desarrollado constituye una solución flexible, escalable y de bajo costo frente a equipos comerciales de medición, demostrando que el uso de FPGAs y sistemas embebidos es una alternativa viable para aplicaciones de monitoreo de seguridad industrial. Este proyecto sienta las bases para futuras mejoras, como el almacenamiento histórico de datos

# BIBLIOGRAFÍA

[1] **World Health Organization (WHO)**, *Exposure to Extremely Low Frequency Fields (EHC Summary)*, WHO, 2024. [En línea]. Disponible en: `https://www.who.int/teams/environment-climate-change-and-health/radiation-and-health/non-ionizing/exposure`

[2] **Health and Safety Executive (HSE)**. *Control of Electromagnetic Fields at Work Regulations 2016 (CEMFAW)*. Disponible en: `https://www.hse.gov.uk/radiation/nonionising/emf-regulations.htm`

[3] **Electrónilab**, *Teclado matricial 4x4 de botones 16 dígitos*. [En línea]. Disponible en: `https://electronilab.co/tienda/teclado-matricial-4x4-de-botones-16-digitos/` [Accedido: 04-dic-2025].

[4] **Electrónilab**, *HMC5883L GY-273 Magnetómetro de 3 ejes brújula digital*. [En línea]. Disponible en: https://electronilab.co/tienda/hmc5883l-gy-273-magnetometro-de-3-ejes-brujula-digital/ [Accedido: 04-dic-2025].

[5] **Electrónilab**, *Sensor de movimiento PIR HC-SR501*. [En línea]. Disponible en: https://electronilab.co/tienda/sensor-de-movimiento-pir-hc-sr501/ [Accedido: 04-dic-2025].

[6] **Electrónilab**, *Módulo buzzer activo zumbador KY-012*. [En línea]. Disponible en: https://electronilab.co/tienda/modulo-buzzer-activo-zumbador-ky-012/ [Accedido: 04-dic-2025].

[7] **Electrónilab**, *Display LCD 16x2 con backlight azul y conversor I2C PCF8574*. [En línea]. Disponible en: https://electronilab.co/tienda/display-lcd-16x2-con-backlight-azul-conversor-i2c-pcf8574/ [Accedido: 04-dic-2025].

[8] **U.S. Occupational Safety and Health Administration (OSHA)**, *Extremely Low Frequency (ELF) Radiation — Health Effects*, OSHA, 2024. [En línea]. Disponible en: https://www.osha.gov/elf-radiation/health-effects

[9] **M. Tang, R. D'Andrea, R. Tell, et al.**, *Progress in Understanding Radiofrequency Heating and Burn Injuries*, 2022. [En línea]. Disponible en: https://pmc.ncbi.nlm.nih.gov/articles/PMC9849420/

[10] **L. J. D. Biberston et al.**, *Suspected Radiofrequency EMF Overexposure — Clinical Practice Guideline*, Joint Trauma System, 12 Jul. 2024. [En línea]. Disponible en: https://jts.health.mil/assets/docs/cpgs/Radio_Frequency_EMF_Overexposure_CPG_12_Jul_2024_ID98.pdf

[11] **S. M. Hussain y S. M. Ali**, *Assessment of Occupational Exposure to Extremely Low Frequency Magnetic Fields in Power Plants*, Journal of Occupational Health, vol. 58, no. 2, pp. 201–209, 2016. [En línea]. Disponible en: https://doi.org/10.1539/joh.15-0202-OA

[12] **Agencia Nacional del Espectro (ANE)**, *Resolución 773 de 2023 — Límites Máximos de Exposición a Campos Electromagnéticos*, Bogotá, Colombia, 2023. [En línea]. Disponible en: https://www.suin-juriscol.gov.co/clp/contenidos.dll/Resolucion/30050345

[13] **M. Piuzzi et al.**, *A Low-Cost Wireless System for the Measurement of Electric and Magnetic Fields in the Surroundings of Power Lines*, IEEE Transactions on Instrumentation and Measurement, vol. 69, no. 6, pp. 3195–3205, 2020. [En línea]. Disponible en: https://doi.org/10.1109/TIM.2019.2933879

[14] **Congreso de la República de Colombia**, *Ley Estatutaria 1581 de 2012 — Protección de Datos Personales*, Bogotá, Colombia, 2012. [En línea]. Disponible en: `https://cancilleria.gov.co/normograma/compilacion/docs/ley_1581_2012.htm`

[15] **Adafruit Industries**, *4×4 Matrix Keypad – Datasheet*, Adafruit Industries LLC, 2019, sec. 3, "Electrical Characteristics".

[16] **AMD – Xilinx**, *Zynq-7000 SoC Technical Reference Manual (UG585)*, AMD Inc., 2024, cap. 2, "Processing System Overview".

[17] **Digilent Inc.**, *FPGA Board Zybo Z7 – Zynq-7000 Development Board*, Digilent, 2024. [En línea]. Disponible en: `https://www.amazon.com/dp/B0797G4485`