



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE ALARMA DE PRESENCIA CON PLACA DIGILENT ZYBO Z7

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Pablo Jarque Zafra

TUTORIZADO POR

Patricia Balbastre Betoret

CURSO ACADÉMICO: 2020/2021

Agradecimientos

Quiero dar las gracias a mi tutora, Patricia Balbastre Betoret, por su ayuda, e implicación durante todo el proyecto. Ya que, sin ella no habría sido posible la realización de este trabajo.

A mis padres, Ana y Jesús, por su apoyo incondicional en todo lo que me propongo, por toda la ayuda que me han brindado durante esta faceta de mi vida y por ser el mejor ejemplo a seguir que podría tener.

A todos mis seres queridos, en especial, a mis abuelos, por creer en mí siempre, darme fuerzas y ser un pilar fundamental de mi vida.

Y, por último, a todos los compañeros y amigos que he conocido en la universidad y me han acompañado en esta etapa.

Gracias a todos.

Resumen

El sistema de alarma de presencia se ha desarrollado con la placa de Digilent Zybo Z7, una tarjeta de desarrollo con un procesador de la familia Xilinx Zynq-7000, que contiene un procesador ARM Cortex-A9 y una FPGA de Xilinx. A la cual se le han añadido los "Pmods" de Digilent KYPD, SSD y PIR, los cuales son un teclado alfanumérico, un doble visualizador de 7 segmentos y un sensor de movimiento infrarrojo.

El sistema se ha creado a través de los programas que la empresa Digilent proporciona para programar sus placas de desarrollo, se ha usado el programa Vivado para la parte de hardware, como la programación de la FPGA y las interconexiones de la placa y sus periféricos. Y el programa Xilinx Vitis para la parte de software desarrollando un programa modular en C con una máquina de estados para el control del sistema principal.

El sistema de alarma funciona detectando primero la presencia a través del sensor, después comienza una cuenta atrás en la que se espera a que el usuario introduzca la clave para desactivar la alarma. En el caso de que el usuario introduzca la clave correcta, el sistema se reinicia, pero en el caso de que el usuario falle repetidas veces la clave o la cuenta atrás llegue a su fin, la alarma se dispara. Este prototipo de alarma de presencia se podría instalar como medida de seguridad para la vigilancia de un hogar, ya sea cerca de una puerta o una ventana, o incluso en espacios más grandes como puede ser la entrada a un comercio, evitando así la entrada de intrusos.

Índice

<i>Agradecimientos</i>	1
Resumen	2
Índice de figuras:	4
Índice de tablas:	5
1. Introducción:	6
1.1. Objetivos:	7
2. Estudio de necesidades, factores a considerar: limitaciones y condicionantes:	8
2.1. Sistemas de alarma:.....	8
2.2. Sistemas embebidos:	9
2.3. Aplicaciones de los sistemas embebidos:	9
2.4. Limitaciones:	10
3. Planteamiento de soluciones alternativas y justificación de la solución adoptada:	11
3.1. Tipos de alarmas:.....	11
3.2. Hardware:.....	12
3.3. Software:	15
3.4. Lenguaje de programación:.....	16
4. Descripción detallada de la solución adoptada:	17
4.1. Especificaciones del funcionamiento de la alarma:	17
4.2. Programación del hardware:	18
4.3. Programación del microcontrolador:.....	27
5. Test y pruebas de funcionamiento:	38
5.1. Test de funcionamiento:	38
5.2. Pruebas de funcionamiento:.....	40
6. Presupuesto:.....	42
6.1. Desglose de costes en función de su naturaleza:	42
6.2. Costes totales:.....	43
7. Conclusiones:	44
7.1. Valoración personal:.....	44
7.2. Futuras ampliaciones y posibles mejoras:.....	44
8. Bibliografía:	46
9. Anexos:.....	48
9.1. Código del programa principal (Main.c):.....	48
9.2. Librería Pmod SSD:	55

Índice de figuras:

Figura 1: Alarma de Pope	6
Figura 2: Sistema de alarma	8
Figura 3: Alarma industrial y alarma de incendios	8
Figura 4: Microcontrolador ARM	9
Figura 5: Maquina de diálisis y robot industrial	10
Figura 6: Placa Zybo Z7-10	13
Figura 7: Pmod PIR	14
Figura 8: Pmod SSD	14
Figura 9: Pmod KYPD	15
Figura 10: Logo de los programas Vivado y Vitis	16
Figura 11: Flujograma de funcionamiento de la alarma	17
Figura 12: Bloque ID del procesador ZYNQ	19
Figura 13: Conexión de los periféricos GPIO	20
Figura 14: Diagrama de bloques con los periféricos led	20
Figura 15: Bloque GPIO triestado	21
Figura 16: Bloques ID de algunos Pmod	22
Figura 17: Bloque ID del timer	23
Figura 18: Activación de las interrupciones en el procesador	23
Figura 19: Diagrama de bloque completo	24
Figura 20: Archivo "Constrains"	25
Figura 21: Gráfico de utilización de la FPGA	26
Figura 22: Conexiones internas de la FPGA	26
Figura 23: Conexiones internas de una pequeña parte de la FPGA	27
Figura 24: Creación de estructuras para los periféricos GPIO	27
Figura 25: Inicialización de los periféricos GPIO	28
Figura 26: Configuración de los periféricos GPIO triestado	28
Figura 27: Configuración del Pmod PIR	28
Figura 28: Configuración del Pmod KYPD	29
Figura 29: Obtención de datos del Pmod KYPD	30
Figura 30: Configuración del Pmod SSD	30
Figura 31: Librerías de control del timer y el GIC	31
Figura 32: Estructura del timer y función manejadora	31
Figura 33: Estructuras y configuración del GIC	32
Figura 34: Configuración del timer	32
Figura 35: Asignación de prioridad de la interrupción	32
Figura 36: Diagrama de estados del funcionamiento	33
Figura 37: Flujograma del código principal	34
Figura 38: Flujograma del código de la interrupción	35
Figura 39: Parámetros de configuración de la comunicación serie	36
Figura 40: Mensajes de comunicación con el usuario	37
Figura 41: Estado de detección del sistema	38
Figura 42: Estado de reconocimiento del sistema	38
Figura 43: Estado después de introducir la clave	39
Figura 44: Estado de alerta	39

Índice de tablas:

Tabla 1: Recursos de la FPGA.....	13
Tabla 2: Utilización de la FPGA	25
Tabla 3: Prueba de funcionamiento 1	40
Tabla 4: Prueba de funcionamiento 2	40
Tabla 5: Prueba de funcionamiento 3	40
Tabla 6: Prueba de funcionamiento 4	41
Tabla 7: Prueba de funcionamiento 5	41
Tabla 8: Prueba de funcionamiento 6	41
Tabla 9: Costes del hardware	42
Tabla 10: Coste humano	43
Tabla 11: Costes totales	43

1. Introducción:

Una de las principales preocupaciones vitales del ser humano es su propia seguridad para garantizar su supervivencia. Es por ello, por lo que, con la ayuda de las herramientas de las que se disponía en cada época el ser humano ha intentado fabricar o construir los mejores sistemas de seguridad para protegerse o disuadir los peligros que pudieran afectarle, terminando así en el desarrollando sistemas de alarma para alertar a la población.

La primera instalación de una alarma electromagnética del mundo se patentó el 21 de junio de 1853. Augustus Rusell Pope de Sommerville, Boston, ideó un sistema simple, pero efectivo (Figura 1). Un dispositivo a pilas que reaccionaba al cerrar un circuito eléctrico en el cual la puerta o una de las ventanas estaban conectadas como unidad independiente a una conexión en paralelo. Así, cuando la puerta o ventana se abrían, hacían cerrar el circuito de forma que la corriente eléctrica creada dentro de los imanes del sistema producía una vibración. Estas oscilaciones se trasmitían a un martillo que golpeaba a una campana de latón. Y, es más, para desconectar la alarma no era suficiente con cerrar de nuevo la puerta o la ventana, ya que había un muelle instalado en la pueta o la ventana para que el circuito siguiese cerrado y continuase sonando.



Figura 1: Alarma de Pope Fuente: Seguridadcomparitda.mx

Hasta entonces la gente confiaba en sus perros guardianes o en las campanillas mecánicas para que sirvieran de sistema de detección ante intrusos. Por lo que la patente de Pope fue algo revolucionario. Pero solo sería el primer paso en el desarrollo de los sistemas de alarmas.

Tiempo después Edwin Holmes decidió aprovechar el telégrafo para que así sus alarmas pudiesen enviar un aviso a su oficina central y, por tanto, actuar de inmediato. Pero, quien realmente tuvo la mejor idea fue su hijo, Edwin T. Holmes, quien decidió aprovechar los sistemas telefónicos nocturnos de Boston, que en esa franja horaria estaban infrautilizados, para sus propios sistemas de alarmas.

Pero si alguien revoluciono por completo los sistemas de alarma del siglo XIX, ese fue Edward A. Calahan, el inventor de las cajas de alarma tipo Calahan y el que tuvo la idea de crear una oficina central de llamadas de emergencias. Estas cajas conectaban a la red eléctrica y no necesitaban de gran mantenimiento, proporcionando así un sistema de alarma de fácil adquisición.

A partir de estos avances, en el siglo XX, se continuaron instalando cada vez más cajas de tipo Calahan en más sitios, como emergencias médicas, comisarías y bomberos proporcionando así más seguridad a la población. Más tarde se integraron los primeros sistemas de detección de movimiento. Hasta que, finalmente, llegaron los primeros sistemas de alarma inalámbricos del mercado, que acabaron con el extenso cableado que era necesario hasta entonces.

Desde ese momento los sistemas de alarma han avanzado enormemente con los recientes avances tecnológicos. Proporcionando así, niveles de seguridad que parecían impensables en el pasado. Hoy en día se pueden encontrar sistemas de detección de innumerables tipos, cámaras de seguridad con análisis de video inteligente o incluso seguridad biométrica asociada a factores humanos prácticamente induplicables.

1.1. Objetivos:

El principal objetivo del presente proyecto es el desarrollo de un prototipo de sistema de alarma capaz de evitar y disuadir los intrusos que deseen entrar en una vivienda o un comercio, alertando de su presencia.

Para ello se desarrollará y se implementará un programa automático capaz de controlar el funcionamiento de la alarma. Se emplearán todos los conocimientos adquiridos sobre sistemas embebidos durante el grado y se ampliarán para la realización completa y eficiente del proyecto. Buscando así implementar de forma práctica los conocimientos sobre microcontroladores e introducirse el campo de las FPGA, que presentan grandes ventajas a la hora de la creación de proyectos gracias a su versatilidad y reprogramabilidad.

Por tanto, primero, se escogerá el hardware con el que se desea diseñar el sistema de alarma de presencia. Seguidamente se desarrollará el funcionamiento concreto que tendrá este y las distintas funcionalidades que incorporará.

Después le seguirá el objetivo de programar y designar que recursos del hardware se van a utilizar. Y, por último, se realizará la programación del software de control y funcionamiento de la alarma. Y su posterior revisión y puesta en marcha para comprobar su funcionamiento y sus posibles mejoras.

2. Estudio de necesidades, factores a considerar: limitaciones y condicionantes:

2.1. Sistemas de alarma:

Un sistema de alarma (Figura 2) es un elemento de seguridad pasivo. Lo que significa que no evitan directamente el peligro, pero si son capaces de advertir de ello. Su función es disuasoria frente a posibles problemas, como pueden ser el inicio de un fuego, la presencia de agentes tóxicos, el desbordamiento de un tanque o en el caso de este proyecto se tratará de la intrusión de personas en una propiedad. Y, por lo tanto, se disuadirá al intruso para que cese esta actividad y no pueda completar el allanamiento.



Figura 2: Sistema de alarma Fuente: RPP.pe

2.1.1. Aplicaciones de los sistemas de alarma:

Las aplicaciones de sistemas de alarma (Figura 3) son muy extensas, se utilizan para advertir de peligros en la industria, en laboratorios, en centrales energéticas, bancos y muchos más sitios con bienes de alto valor o peligrosos. Pero sobre todo en comercios y viviendas para preservar la seguridad de los bienes y personas que residen o trabajan allí.



Figura 3: Alarma industrial y alarma de incendios Fuente: alarmasacusticas.blogspot.com y chacarrex.com

2.2. Sistemas embebidos:

Los sistemas embebidos o empotrados (Figura 4) son herramientas de computación usadas para la ejecución de tareas de control. Son circuitos integrados programables, capaces de ejecutar las órdenes que se graban en su memoria. Suelen estar compuesto por distinto bloques funcionales: unidades de procesamiento, memoria y periféricos de entrada y salida.

La mayoría de los sistemas de alarma están diseñadas con sistemas embebidos en su interior que procesan la información recibida y controlan el funcionamiento del sistema.



Figura 4: Microcontrolador ARM Fuente: blog.330ohm.com

2.3. Aplicaciones de los sistemas embebidos:

Las aplicaciones de los sistemas embebidos son infinitas, hoy en día están presentes en todas las facetas de la humanidad, las podemos encontrar en campos como:

- La electrónica de consumo: electrodomésticos, consolas de videojuegos, control remoto, televisiones, etc.
- Sistemas de comunicación: sistemas de telefonía, contestadores, routers, Wi-Fi, infraestructuras de redes.
- Automóviles: inyección electrónica, frenos, elevadores, control de asientos, instrumentación, seguridad.
- Industria: monitorización, control, robótica, maquinaria específica.
- Medicina: monitores cardiacos, renales y de apnea, marcapasos, máquinas de diálisis...

Entre muchos otros, ya que los sistemas embebidos permiten la creación, el diseño y el control de tareas en un espacio muy reducido. Es por eso por lo que están presentes en todos los ámbitos de la vida (Figura 5).



Figura 5: Maquina de diálisis y robot industrial

Fuente: medicalexpo.es y revistaderobots.com

2.4. Limitaciones:

Las principales limitaciones que tiene el proyecto son debidas al propio hardware que, aparte de ser un hardware relativamente costoso para tratarse de un proyecto académico, no presenta gran escalabilidad.

En el caso de la tarjeta de desarrollo que se va a utilizar durante el proyecto, no posee más entradas adaptadas a periféricos con las que conectar otro tipo de sensores o actuadores que facilitarían la emisión de una señal de aviso más potente, como podría ser una luz con mayor luminosidad u otro tipo de detección de intrusos que hiciese el proyecto más completo.

Esta misma limitación es la que hace que el proyecto solo pueda cubrir una entrada del comercio o la vivienda, ya que, si pudiese ampliarse la instalación, la seguridad sería completa en toda la propiedad.

3. Planteamiento de soluciones alternativas y justificación de la solución adoptada:

Para la realización de este trabajo se han analizado diversas soluciones existentes de alarmas que hoy en día son las más populares del mercado. Y se ha valorado cual sería el mejor sistema posible y que herramientas serían las más eficientes para llevarlo a cabo.

3.1. Tipos de alarmas:

Existen muchos tipos, pero principalmente se pueden clasificar en función de dos características: el grado de seguridad y el sistema de detección que utilizan.

3.1.1. Grado de seguridad:

Según el grado de seguridad podemos encontrar alarmas desde el grado 1 hasta el grado 4, siendo estas las más seguras.

- Grado 1 (bajo riesgo): Son las alarmas que emiten cualquier tipo de señal disuasoria pero que no están conectadas con una Central Receptora de Alarmas.
- Grado 2 (riesgo bajo-medio): Son aquellas alarmas que si que disponen de un control remoto básico con el usuario y están conectadas con una Central Receptora de Alarmas.
- Grado 3 (riesgo medio-alto): Son muy parecidas a las de grado 2 pero con mayor sofisticación y con la obligatoriedad de tener doble vía de comunicación para la inhibición y la función anti-masking en las cámaras de vigilancia.
- Grado 4 (alto riesgo): Este grado de seguridad es prácticamente de uso exclusivo para lugares con material radioactivo, explosivos, bienes de muy alto valor e instalaciones militares. Y, se rige en función de la legislación de cada país.

3.1.2. Sistemas de detección:

Según el sistema de detección se pueden encontrar casi innumerables tipos de alarmas para cubrir un perímetro o una propiedad privada, pero los más comunes son los siguientes:

- Sistemas de vigilancia CCTV: son los también llamados circuitos cerrados de televisión y consta de un grupo de cámaras que posteriormente procesaran o grabaran la imagen obtenida.
- Sistemas de control de acceso: sirven para restringir la entrada solo a un grupo de personas determinado mediante un pin, huella o patrón biométrico.

- Sensores de movimiento: detectan el movimiento gracias a distintos métodos físicos, como pueden ser los rayos infrarrojos.
- Sensores de perturbación de campo electrostático: detectan la presencia cuando el campo eléctrico que se genera sufre una alteración.
- Sensores sísmicos: detectan las vibraciones dentro de su campo de acción.

Por tanto, después de saber cuál es la clasificación de los sistemas de alarma y teniendo en cuenta los objetivos y las limitaciones del proyecto, se ha decidido realizar un sistema de alarma de grado 1, basado en la detección de movimiento y presencia, con un sistema de control por clave de acceso. Lo que hará que sea un sistema de alarma muy versátil en cuanto a su uso y así tendrá buenas prestaciones.

3.2. Hardware:

Para la implementación de la alarma se requerirá de un sistema embebido que se encargue del control del sistema y otros periféricos para la obtención de datos y la comunicación con el usuario. Es por eso que la electrónica de la empresa Digilent es perfecta para este proyecto, puesto que sus placas de desarrollo presentan unas características muy versátiles y una sinergia excepcional con el resto de sus periféricos que cubren todos los aspectos necesarios del proyecto.

3.2.1. Placa Diligent Zybo Z7:

La placa Digilent Zybo Z7(Zyng Board) (Figura 6) es una plataforma de desarrollo de con muchas funcionalidades y fácilmente integrable. Diseñada alrededor del Zynq-7000 System-on-Chip (APSoC) de Xilinx, que contiene un procesador ARM Cortex-A9 de 667 MHz de doble núcleo y una FPGA de Xilinx de la serie 7.

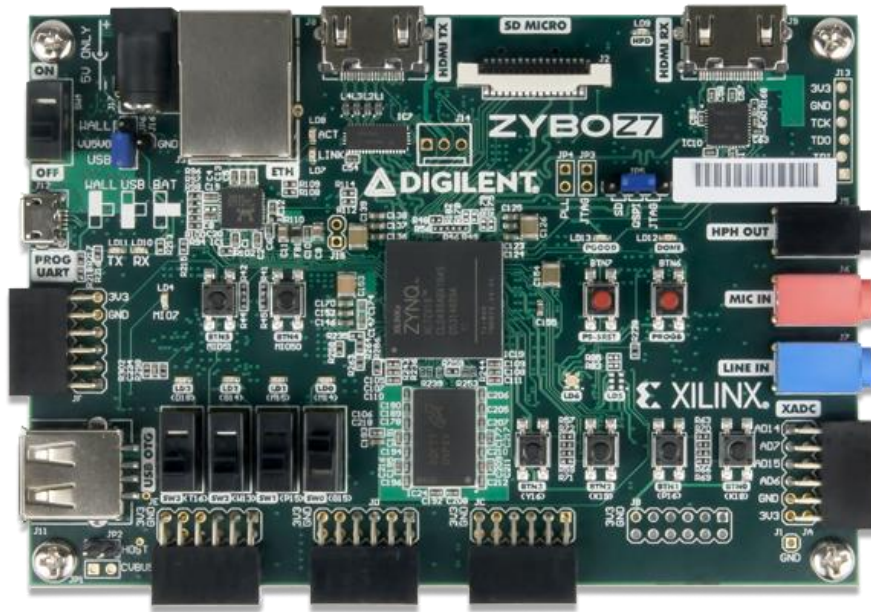


Figura 6: Placa Zybo Z7-10

Fuente: amazon.es

Sus principales características son:

- Memoria: 1 GB x32 DDR3L
- Memoria flash Quad-SPI de 16 Mb
- 2 conectores HDMI (input/output)
- 1 conector Ethernet de 1 GBit
- 1 puerto USB 2.0 PHY
- Ranura de tarjeta MicroSD, lugar en el que se guardan los archivos configurados
- Códec audio con salida de auriculares y micrófono
- GPIO: 6 pulsadores, 4 interruptores, 4 LEDs y un LED RGB.
- Micro-USB UART
- Micro-USB JTAG
- Micro-USB para fuente de alimentación de la placa 5V

Zybo Z7 presenta unos recursos totales limitados con los que se calcula una medición del costo del área de la FPGA. Para ello se utilizan BRAM (bloques RAM compuestos por una memoria pequeña interna y rápida a la que se puede acceder en cada ciclo de reloj), DSP, FF y LUT (Tabla 1).

Recursos	Total
BRAM	140
DPS	220
FF	106400
LUT	17600

Tabla 1: Recursos de la FPGA Fuente: Propia

3.2.2. Pmod PIR:

El Pmod PIR de Digilent (Figura 7) es un sensor infrarrojo piroeléctrico pasivo de bajo consumo. Construido con el EKMC1601111 puede detectar movimiento hasta 5 metros de distancia. Es ideal para aplicaciones de detección de movimiento a largo plazo y posee una sensibilidad excepcional.

Al tratarse de un sensor pasivo este no emite ningún tipo de señal, detecta la diferencia entre el calor del cuerpo humano y el espacio alrededor. Su componente principal son los sensores piroeléctricos, un componente diseñado para detectar cambios en la radiación infrarroja recibida. Con su bajo consumo de 170 μA y su conexión de 6 pines lo hacen idóneo para la detección de movimiento en el sistema de alarma con la Zybo Z7.



Figura 7: Pmod PIR

Fuente: digikey.es

3.2.3. Pmod SSD:

El Pmod SSD de Digilent (Figura 8) es un display de siete segmentos y dos dígitos (cátodo común). Se conecta en 2 entradas de 6 pines. Solo se puede visualizar un dígito a la vez, por tanto, es necesario alternar a una frecuencia de 50 Hz o superior entre los dígitos para proporcionar el efecto visual de ambos números encendidos.

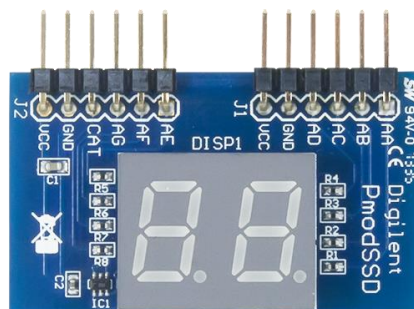


Figura 8: Pmod SSD

Fuente: digilentinc.com

3.2.4. Pmod KYPD:

El Pmod KYPD de Digilent (Figura 9) es un teclado alfanumérico matricial de 16 botones (números del 0 al 9 y letras de la “A” a la “F”), que puede detectar el tecleo simultáneo de botones y se conecta a la Zybo Z7 a través de un puerto de 12 pines.

Como cualquier teclado matricial tiene las filas y columnas aisladas, y para detectar que tecla esta activada es necesario leer las señales correspondientes a las filas y columnas, descifrando así la matriz y descubriendo la tecla que ha sido presionada. Es por eso por lo que el teclado permite ver si varias teclas están siendo pulsadas, pero no podrá saber con exactitud que teclas son.



Figura 9: Pmod KYPD Fuente: digilentinc.com

3.3. Software:

En cuanto a las herramientas de software necesarias para llevar a cabo el proyecto se utilizarán los programas gratuitos que proporciona Digilent para la programación de sus placas de desarrollo. Estos programas son: Vivado y Vitis (Figura 10).

3.3.1. Vivado:

Vivado Design Suite es una herramienta de Xilinx para el desarrollo de System on Chip (SoC). Es compatible con los procesadores de la marca Xilinx y con la mayoría de las placas de desarrollo que contienen estos, como lo es la Zybo Z7. Una de sus principales funciones es la programación de las FPGA y la gestión de los recursos de las placas de desarrollo y sus respectivos procesadores de una forma gráfica e intuitiva.

3.3.2. Xiling Vitis:

Vitis es un entorno de trabajo para el desarrollo de aplicaciones informáticas muy similar a otros como: Keil μ Vision o Code Composer Studio. La principal misión de este programa es ofrecer al usuario un entorno de trabajo donde desarrollar aplicaciones para después cargarlas a los procesadores de Xilinx.



Figura 10: Logo de los programas Vivado y Vitis

Fuente: wikiband.com y xilinx.com

3.4. Lenguaje de programación:

Para crear la aplicación que controlará este proyecto se utilizarán dos lenguajes de programación: Verilog y C.

- Verilog: es un lenguaje informático de descripción de hardware (HDL) usado para modelar sistemas electrónicos analógicos, digitales o mixtos. Fue creado con el objetivo de ser similar a C para resultar más familiar para los ingenieros, pero presenta algunas diferencias. No es un programa completamente secuencial, sino que consiste en una jerarquía de módulos. Algunas partes de los módulos se ejecutarán en paralelo de forma concurrente y otras partes serán ejecutadas de forma secuencial. Se empleará para la programación de la FPGA y la placa de desarrollo.
- C: es un lenguaje de programación secuencial de propósito general, muy utilizado para la programación de microcontroladores, aunque tiene infinidad de usos. Se empleará para la creación del programa principal de control y funcionamiento del sistema de alarma.

4. Descripción detallada de la solución adoptada:

El primer paso que se realizó para la creación del proyecto sabiendo el hardware que se iba a utilizar fue elegir unas especificaciones de funcionamiento. Para así, una vez escogidas estas, saber que recursos se iban a utilizar y que estrategia de programación se iba a seguir.

4.1. Especificaciones del funcionamiento de la alarma:

El funcionamiento del sistema de alarma debía de ser sencillo e intuitivo para el usuario, pero a la vez debía cumplir con la tarea de alertar de la presencia de intrusos de la manera correcta.

Es por ello por lo que se decidió seguir una metodología de diseño gráfica creando un flujograma de funcionamiento básico (Figura 11) para establecer las diferentes etapas por las que pasaría la alarma durante su funcionamiento.

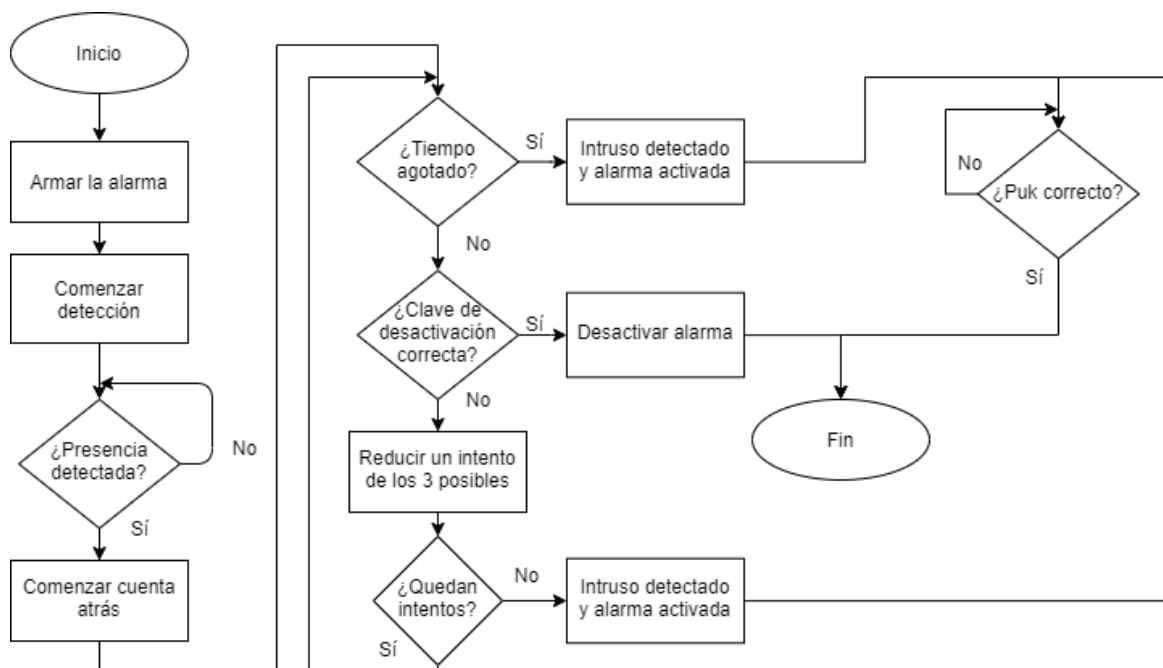


Figura 11: Flujograma de funcionamiento de la alarma

Fuente: Propia

En primer lugar, le aparecerá un mensaje por pantalla al usuario para que introduzca la clave de cuatro dígitos para armar la alarma. Entonces, el usuario pondrá la clave para armar la alarma, en ese momento el led de estado se volverá azul indicando que la alarma se ha armado y en breves momentos comenzará el proceso de detección. Habrá un pequeño periodo de espera para dotar de tiempo al usuario para que se aleje de la zona de detección y posteriormente la alarma empezará a recoger datos del sensor a la espera de detectar movimiento.

Una vez el sistema haya detectado movimiento, comenzará una cuenta atrás regresiva de 30 segundos. En ese momento el led de estado cambiara a rojo indicando que el sensor ha detectado movimiento. Esos 30 segundos será el tiempo del que dispondrá el usuario para introducir la contraseña.

En el caso de que el usuario introduzca correctamente la contraseña, la alarma se desactivará de inmediato y el led de estado cambiará a verde. En el caso de que el usuario introduzca erróneamente la clave perderá 1 de los 3 intentos de los que dispone para desactivar la alarma. En el caso de que falle los 3 intentos de desactivación o la cuenta atrás llegue a su final, los leds de alerta comenzarán a parpadear y el led de estado parpadeará alternado su color.

En ese momento la única forma de parar la alarma será con la introducción del PUK, una clave de desactivación total predefinida de seis dígitos. Una vez se haya introducido el PUK, el led de estado cambiará a verde. Y, al igual que cuando se introduce correctamente la contraseña, se le darán 2 opciones al usuario. La primera opción será volver a armar la alarma introduciendo la clave y la segunda será llevar la alarma a un estado de "Stand by" pulsando un interruptor. Para regresar de ese estado de "Stand by" el usuario solo deberá volver a pulsar el mismo interruptor.

De esta manera, se consigue un funcionamiento simple, en el cual el usuario solo tendrá que interactuar con la alarma cuando se le requiera para activarla y desactivarla con la clave. Es más, el usuario podrá escoger cada vez que active la alarma una clave nueva y esta se quedará guardada inmediatamente.

4.1.1. Herramientas necesarias para el desarrollo del sistema:

Una vez el funcionamiento del sistema de alarma ha sido diseñado se ha de ver que requisitos del hardware y el software van a requerirse para llevar el diseño del funcionamiento a la práctica.

Las herramientas del hardware escogido para la realización del proyecto que se requerirán para el desarrollo del sistema son las siguientes:

- Periféricos GPIO:
 - Leds y Led RGB: para la alerta y el led de estado.
 - Switches: para apagar la alarma y encenderla.
 - Las salidas de los Pmod con los Pmod seleccionados.

- Interrupción: para poder manipular el Pmod SDD y alternar la visibilidad de los dígitos como se ha comentado con anterioridad, mientras se ejecuta el código principal.

- Timer: para asociar la interrupción y así poder controlar el tiempo que transcurre entre las acciones.

Para preparar las herramientas de hardware correspondientes a la tarjeta de desarrollo Zybo Z7 será necesario comenzar con la programación de la FPG a través del programa Vivado.

4.2. Programación del hardware:

Una vez se tiene claro que funcionamiento tendrá el sistema de alarma y que recursos del hardware seleccionado se van a utilizar, el siguiente paso es crear

un proyecto en Vivado que permita programar y preparar correctamente la FPGA y la totalidad de la placa de desarrollo.

Para comenzar se deberá abrir Vivado y crear un nuevo proyecto, pero será necesario que el tipo de este proyecto sea “RTL_project”. Este tipo de proyecto permitirá que al terminar su realización se pueda analizar, implementar y exportar los archivos de configuración del hardware al programa Vitis. Donde se podrá programar el microcontrolador con las especificaciones de hardware que previamente se hayan diseñado.

4.2.1. Configuración de los periféricos GPIO:

El siguiente paso que será importante dar a la hora de realización del proyecto será el de seleccionar la placa de desarrollo que se va a utilizar. Así, gran parte de la configuración será automática. Ya que el programa reconocerá las especificaciones y componentes que tiene dicha placa, en este caso la Zybo Z7-10.

Una vez terminada la preparación del proyecto, y ya dentro del programa, se creará un “Block design”. Esta herramienta permitirá programar las conexiones que se quieren dentro de la placa de desarrollo de forma gráfica, basándose en una interfaz de IP’s.

Una vez se haya creado el “Block design”, se comenzarán a añadir los bloques IP necesarios para programar correctamente los recursos de la placa. En primer lugar, habrá que añadir el procesador Zynq-700 (Figura 12) e inmediatamente pulsar la opción de “Run Block Automation”. Esta opción, configurará de forma automática el procesador de la placa de desarrollo con las funciones y el hardware que esta trae predefinidas.

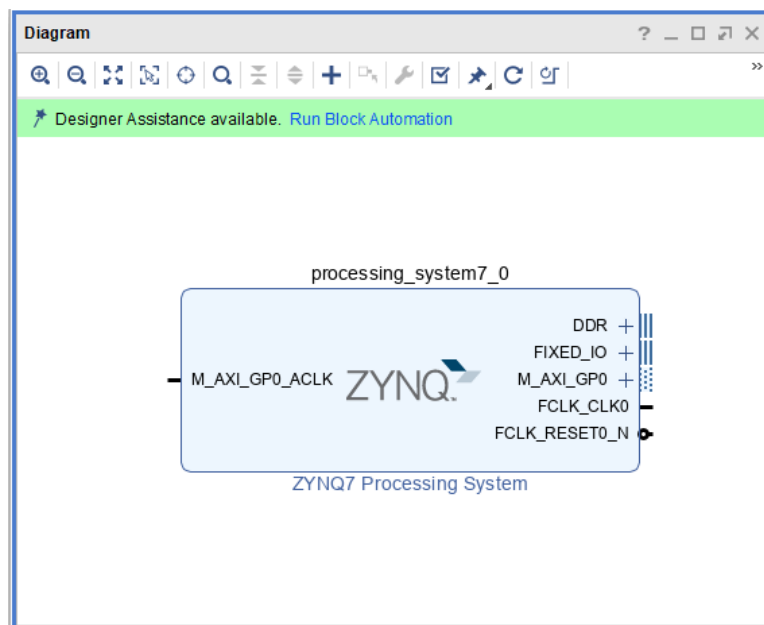


Figura 12: Bloque ID del procesador ZYNQ Fuente: Propia

Una vez añadido el procesador, se añadirán los periféricos GPIO de salida como son los Leds y el Led RGB. Al haber añadido la placa en la preparación del programa, estos se podrán añadir de forma automática, pulsando la opción de “Connect Board Component” (Figura 13). De forma automática el programa, a través de un bloque “AXI_GPIO_XXXXX” creará el bloque IP necesario.

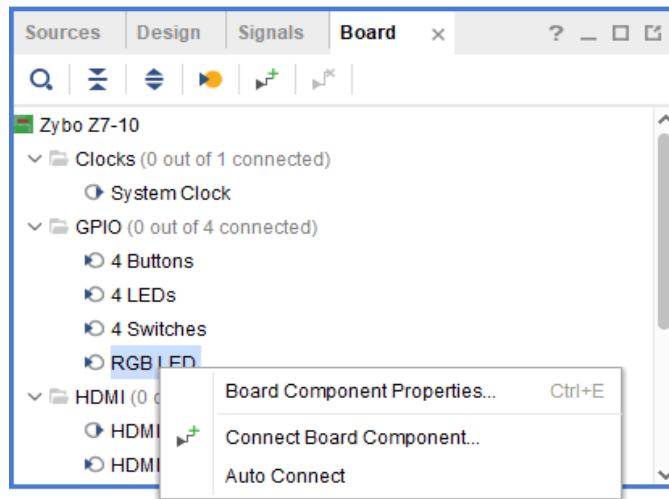


Figura 13: Conexión de los periféricos GPIO Fuente: Propia

Ya creado, utilizando la opción de “Run Connection Automation”, el programa conectará de forma automática el procesador con el periférico a través de un bloque IP de interconexión AXI. Este proceso se repetirá igual con los Leds y el Led RGB, quedando de la siguiente forma el diagrama (Figura 14).

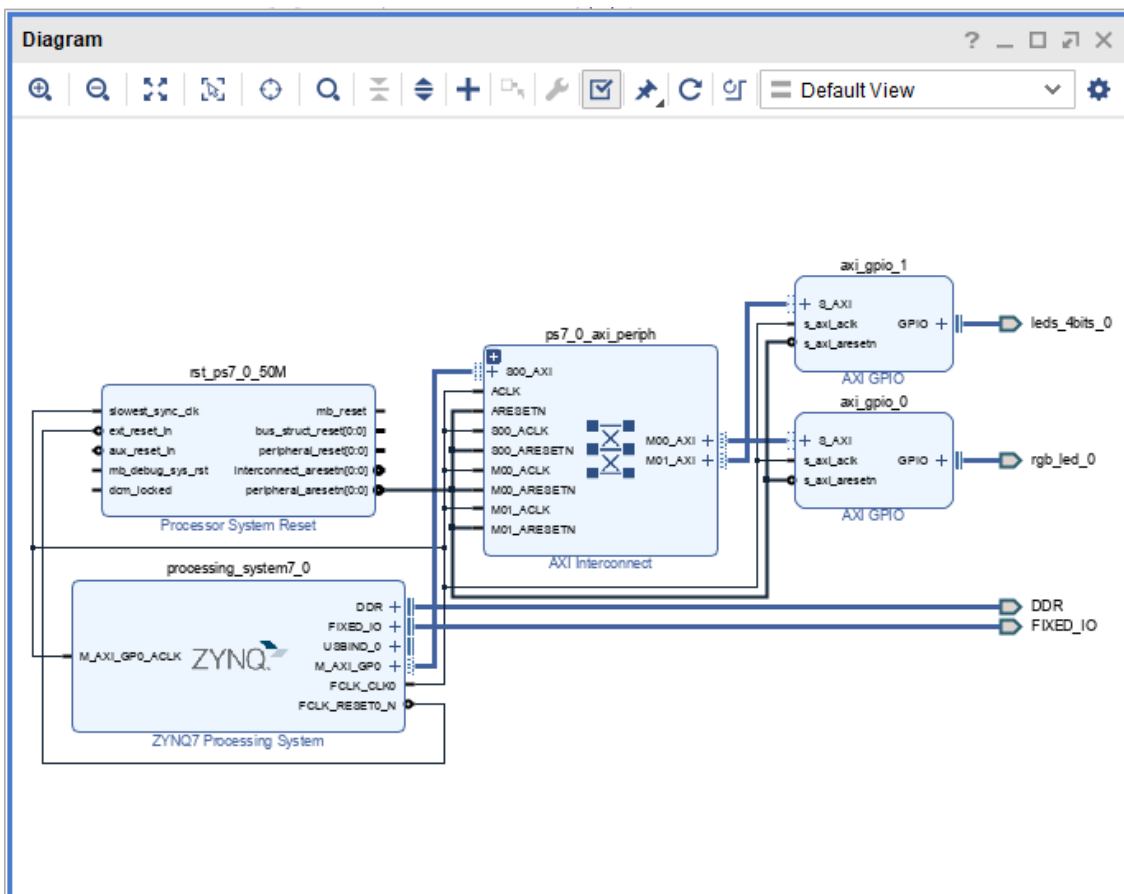


Figura 14: Diagrama de bloques con los periféricos led Fuente: Propia

Ya realizado este proceso con los Leds, habrá que hacer uno similar con los interruptores y los botones. Pero en este caso se quieren ajustar los perifericos de entrada de forma que esten en modo triestado (Figura 15). De esta forma sera más fácil programarlos más adelante.

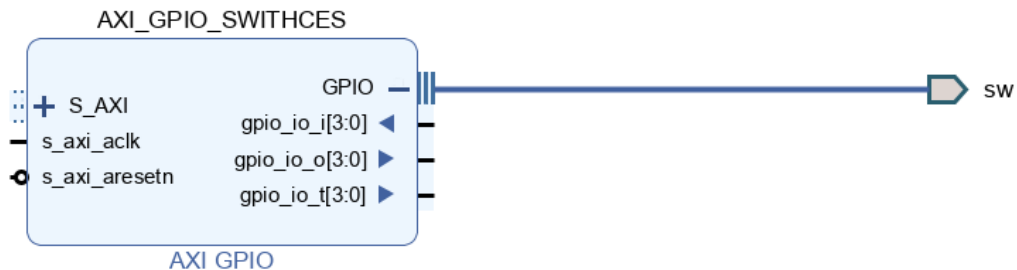


Figura 15: Bloque GPIO triestado Fuente: Propia

Por eso, en vez de adjuntarlos automáticamente, se introducirá un AXI_GPIO_IP de proposito general al cual se le cambiará el nombre por uno reconocible. Después sobre ese AXI_GPIO_IP se utilizará la opción de "Make External". Esto creará un puerto externo del cual se obtendrá la información de este GPIO, al cual se le llamara "sw" en el caso de los interruptores y "btn" en el caso de los botones. Gracias a este cambio, el programa reconocera este IP en modo triestado y habilitara 3 canales, aunque solo se utilizará el de entrada.

4.2.2. Configuración de los Pmods:

Antes de comenzar con la programación de los Pmod, será necesario introducir las IP de estos Pmod de forma manual, puesto que el programa no las incluye. Ha habido que descargarlas de la página oficial de Digilent con anterioridad. Y, una vez descargadas, en la sección de ajustes del proyecto. En repositorio de ID, habrá que pegar la ruta de donde se encuentran esto ID para que Vivado pueda acceder a ellos, esto facilitará la configuración.

En el caso de los Pmod su configuración también será automática. Solo será necesario ir al conector Pmod al cual se quiera conectar en la placa y volver a usar la opción de "Connect Board Component". En este caso se podrá escoger que Pmod se desea conectar a esa entrada y este se creará automáticamente. Cada vez que se crea un IP será necesario utilizar de nuevo la opción "Run Connection Automation", para que se generen las conexiones pertinentes con el procesador.

Si siguiendo estos pasos estarán conectados tanto el Pmod KYPD, como el Pmod PIR. Pero el Pmod SSD al conectarse a dos puertos distintos no tiene un IP dedicado. Por tanto habrá que crear dos Pmod_Gpio IP (Figura 16), que son IP dedicadas a Pmod de manera genérica y conectarlas con las salidas correspondientes, creando dos puertos de interfaz asociados a Pmod_Gpio y asociándolos a los puertos Pmod de la placa.

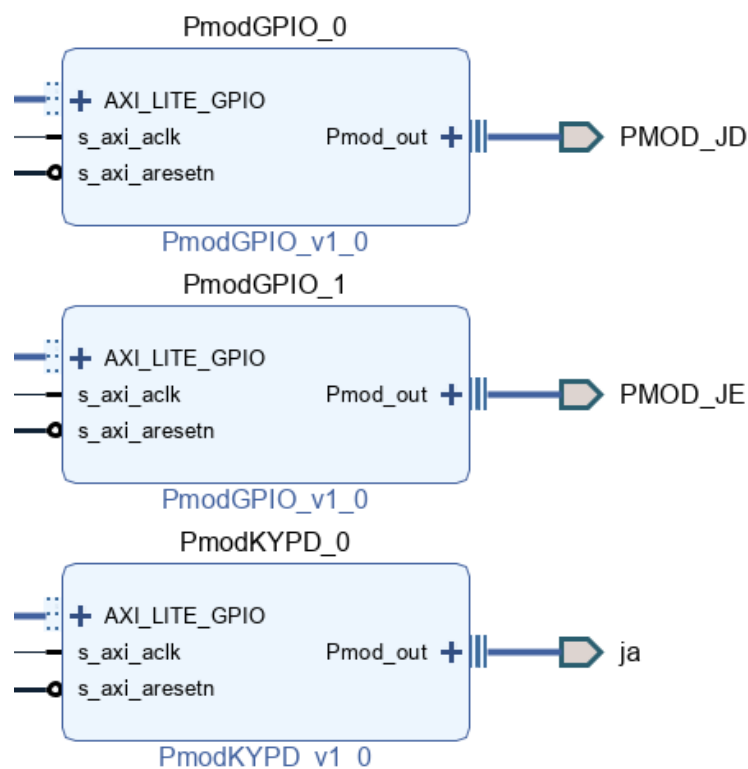


Figura 16: Bloques ID de algunos Pmod Fuente: Propia

4.2.3. Configuración del timer y la interrupción:

El objetivo es activar las interrupciones de propósito general IRQ y en el primer pin asociado a este puerto de las 16 posibles, conectar la interrupción asociada al timer.

Para ello será necesario crear un IP AXI_timer (Figura 17), activar las interrupciones en el procesador y conectar el pin asociado a la interrupción del timer al puerto IRQ. De esta forma ocupará el primer pin del puerto, en este caso el pin 61. El timer será de 32 bits ya que así viene configurado, aunque se pueda cambiar este parámetro.

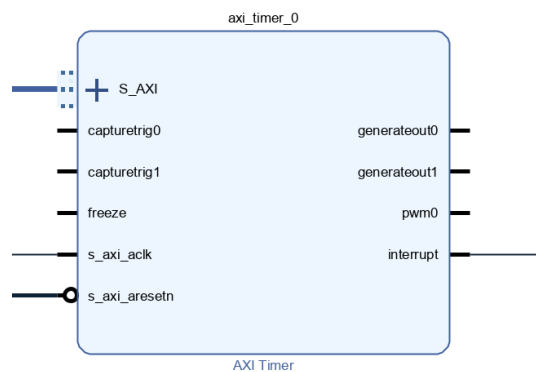


Figura 17: Bloque ID del timer Fuente: Propia

Las interrupciones se activan dentro del procesador como se muestra en la figura siguiente, activando el apartado de interrupciones del puerto “IRQ_F2P2 (Figura 18).

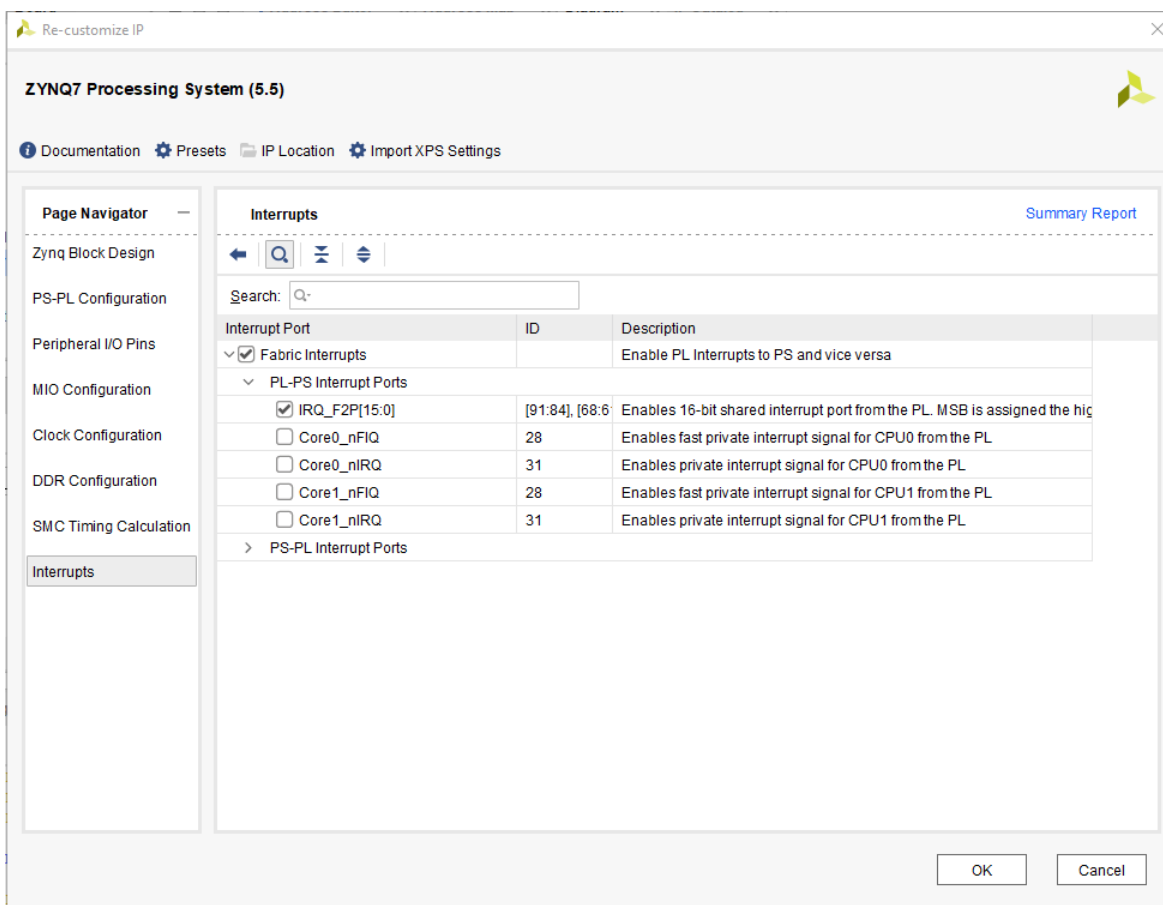


Figura 18: Activación de las interrupciones en el procesador Fuente: Propia

Una vez se hayan configurado todos los recursos de la placa de desarrollo el diagrama final sera el siguiente (Figura 19):

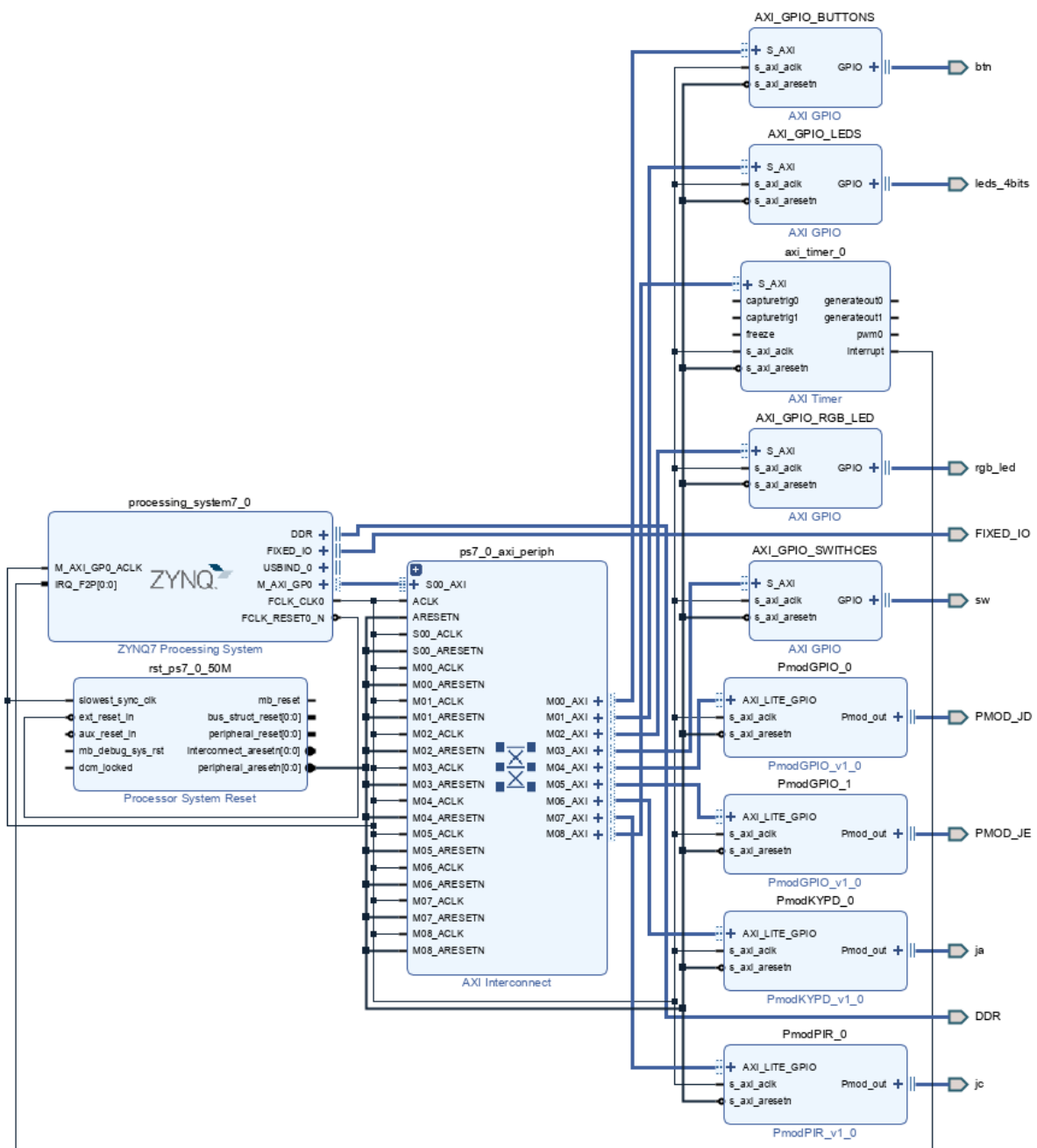


Figura 19: Diagrama de bloque completo Fuente: Propia

Antes de avanzar, con el objetivo de que cuando exportemos el proyecto a Vitis el microcontrolador reconozca todas las direcciones IP, asociadas a los periféricos, de las cuales estamos dotando a la placa y las cuales tendrán un espacio en la memoria, es necesario que adjuntemos un archivo que proporciona Digilent de “Constraints” de la placa Zybo Z7, llamado “Zybo-Z7-Master.xdc”. Para añadirlo será suficiente con ir al apartado de “Add sources”, elegir “Constraints” y marcar la dirección donde se encuentra el archivo.

En este archivo, dentro de Vivado, habrá que descomentar la línea de los pines que se van a utilizar, y cambiar el nombre de los registros en función del nombre que se le haya dado a las salidas.

Como se observa en la siguiente figura, las líneas de código correspondientes a los interruptores y botones esta no está comentada y se han ajustado el nombre de las salidas al nombre del triestado, “sw_tri_io” (Figura 20).

```
##Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_io[0] }]; #IO_L19N_T3_VREF_35 Sch=sw[0]
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_io[1] }]; #IO_L24P_T3_34 Sch=sw[1]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_io[2] }]; #IO_L4N_T0_34 Sch=sw[2]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_io[3] }]; #IO_L9P_T1_DQS_34 Sch=sw[3]

##Buttons
set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports { btn_tri_io[0] }]; #IO_L12N_T1_MRCC_35 Sch=btn[0]
set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn_tri_io[1] }]; #IO_L24N_T3_34 Sch=btn[1]
set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports { btn_tri_io[2] }]; #IO_L10P_T1_AD11P_35 Sch=btn[2]
set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn_tri_io[3] }]; #IO_L7P_T1_34 Sch=btn[3]
```

Figura 20: Archivo "Constraints"

Fuente: Propia

Se descomentarán las líneas de código del archivo “Constraints” pertenecientes a los Leds, el Led RGB, los botones, los interruptores y los puertos Pmod.

4.2.4. Configuración de la FPGA:

Por último, una vez se ha completado el diagrama de bloques y el archivo “Constraints” se ha sido añadido correctamente, es el momento de crear un “HDL Wrapper” de nuestro “Block design”. Esto convierte el diagrama de bloques, junto con el resto del proyecto, en una serie de archivos “.v” que las herramientas de Vivado pueden leer y procesar. Estos archivos serán archivos en lenguaje Verilog preparados para programar la FPGA con las conexiones que se han creado.

Ya realizado este paso, se va a generar un “Bitstream”, es el archivo necesario que se genera a partir del “HDL Wrapper” con los archivos en lenguaje Verilog para programar la FPGA. Primero el “HDL Wrapper” ha de pasar un proceso de análisis, otro de síntesis y un último de implementación. Para así, generar el “Bitstream”, que se comprimirá en un archivo “.xsa” y se exportará más adelante a Vitis para programar el Software del sistema de alarma.

Al haber realizado este proceso podemos obtener información sobre como quedarán las conexiones de la FPGA y los recursos que se van a utilizar (Figura 21 y tabla 2) después de la implementación.

Resource	Utilization	Available	Utilization %
LUT	1423	17600	8.09
LUTRAM	62	6000	1.03
FF	1680	35200	4.77
IO	47	100	47.00

Tabla 2: Utilización de la FPGA

Fuente: Propia

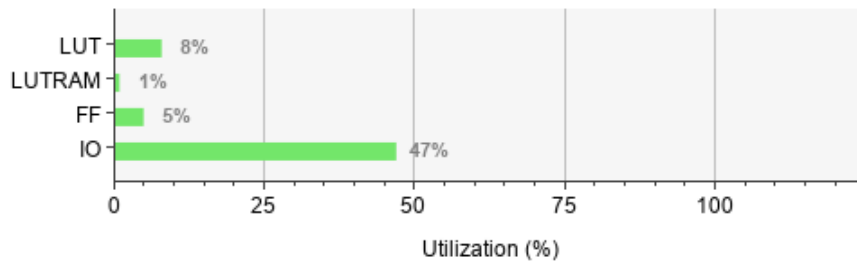


Figura 21: Gráfico de utilización de la FPGA Fuente: Propia

Las conexiones de la FPGA (Figura 22 y 23) serán las siguientes:

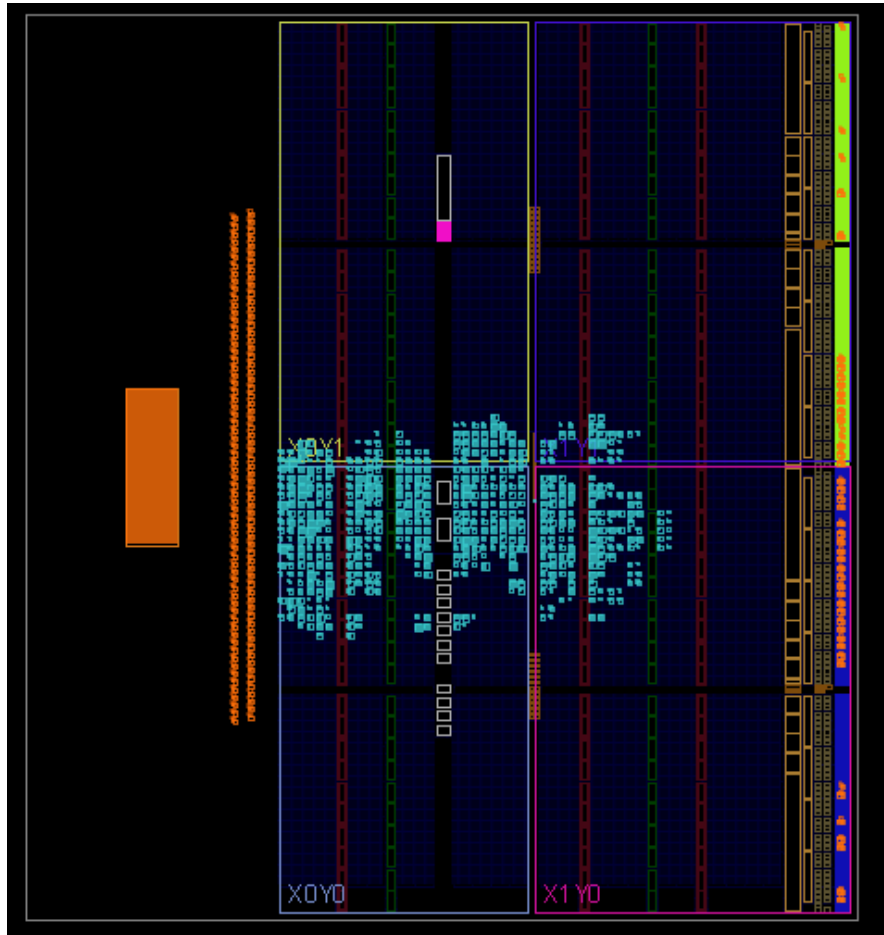


Figura 22: Conexiones internas de la FPGA Fuente: Propia

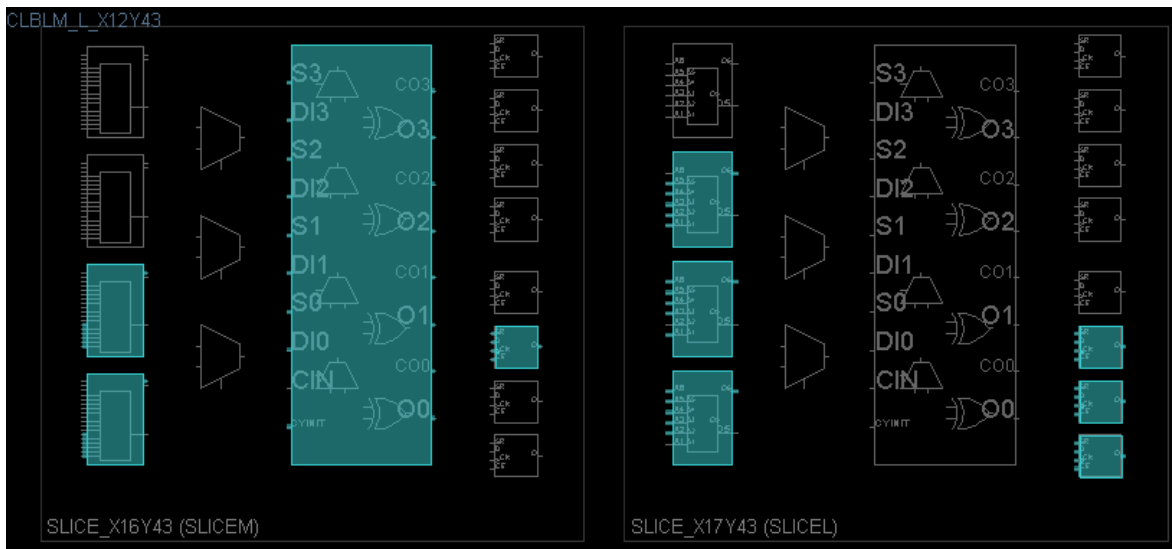


Figura 23: Conexiones internas de una pequeña parte de la FPGA Fuente: Propia

Después de la realización de todo el proceso, solo será necesario ir a Vitis para terminar el proyecto con la programación del software de control de la alarma. Pero la base de ese proyecto será el archivo ".xsa" que ha generado Vivado, donde se encuentra toda la información que programará la FPGA para configurar el hardware de la Zybo Z7, antes de que el software de control se cargue en el microcontrolador.

4.3. Programación del microcontrolador:

Para la programación del microcontrolador se utilizará el ya mencionado programa Vitis. En este, se creará un proyecto en C, eligiendo como plataforma del proyecto el archivo ".xsa" importado desde Vivado. Gracias a este proceso, el programa ya se creará incluyendo distintas carpetas, drivers, direcciones de memoria, etc. Que se podrán utilizar durante el desarrollo del programa. Y que, además, serán de gran utilidad puesto que estarán directamente relacionadas con el hardware que se está utilizando.

Para programar el sistema de control es necesario, al igual que se ha configurado el hardware de las herramientas que se van a utilizar, configurar el software de estas. Por ello antes de comenzar con la explicación del programa principal, habrá que hacer una configuración e inicialización de las distintas herramientas.

4.3.1. Programación de los periféricos GPIO:

Para comenzar será necesario inicializar los interruptores, botones, Leds y el Led RGB. Creando estructuras de tipo "GPIO" para cada uno y una estructura de configuración (Figura 24).

```
XGpio_Config *cfg_ptr;
XGpio led_device, btn_device, sw_device, led_rgb_device;
```

Figura 24: Creación de estructuras para los periféricos GPIO Fuente: Propia

Y haciendo uso de las funciones de configuración e inicialización de este tipo de periféricos (Figura 25).

```
// Initialize LED Device
cfg_ptr = XGpio_LookupConfig(LED_ID);
XGpio_CfgInitialize(&led_device, cfg_ptr, cfg_ptr->BaseAddress);
```

Figura 25: Inicialización de los periféricos GPIO Fuente: Propia

Para después, seleccionar si son de entrada o de salida, a través de una máscara, indicando con un 0 que periférico será de salida y con un 1 de entrada (Figura 26).

```
// Set Led Tristate
XGpio_SetDataDirection(&led_device, LED_CHANNEL, LED_MASK);
```

Figura 26: Configuración de los periféricos GPIO triestado Fuente: Propia

Este proceso será necesario repetirlo una vez por cada periférico antes de comenzar con el programa de control. Para posteriormente, poder acceder a ellos y utilizar las funciones de lectura y escritura.

4.3.2. Configuración de los Pmod:

Para programar e inicializar los Pmod se hará distinción entre los 3 que se van a utilizar: el Pmod KYPD, el Pmod PIR y, por último, el Pmod SSD. Ya que, la configuración de los Pmod será distinta en función del que se desea utilizar.

4.3.2.1. Pmod PIR:

El Pmod PIR será el más sencillo de configurar y de utilizar, puesto que este ya cuenta con librerías preparadas para utilizar en código C y su funcionamiento es simple.

Simplemente bastará con añadir al archivo “.c”, donde se esté realizando el programa las librerías de control de este periférico. Que ya vienen incluidas dentro del proyecto, al haber utilizado como plataforma el archivo importado de Vivado. En este caso la librería: PmodPIR.h.

Una vez añadida esta librería, para configurar el sensor de presencia será necesario crear una estructura de tipo “PmodPIR” y utilizar la función de inicialización del periférico que incluye la librería (Figura 27).

En esta función de inicialización habrá que poner una dirección de memoria para configurar el sensor. Simplemente, bastará con ir a la librería “xparameters.h” que incluye el proyecto. En esta librería todas las direcciones y registros de memoria están referenciados y definidos según los nombres que, previamente, se les ha otorgado en Vivado a los periféricos y sus controladores.

```
PmodPIR myPIR;

//Set PmodPIR
PIR_begin(&myPIR, XPAR_PMODPIR_0_AXI_LITE_GPIO_BASEADDR);
```

Figura 27: Configuración del Pmod PIR Fuente: Propia

Ya configurado el periférico, solo será necesario usar la función “PIR_getState(&myPIR)” para obtener los datos de lectura del sensor. Devolverá en una variable de tipo entero un 1 si detecta movimiento o un 0 en el caso de no detectarlo.

4.3.2.2. Pmod KYPD:

El proceso de configuración el Pmod KYPD es muy similar al del sensor infrarrojo. Al igual que para el Pmod PIR, para el teclado habrá que crear una estructura propia de tipo Pmod KYPD y habrá que añadir la librería “PmodKYPD.h”.

Para comenzar la configuración habrá que utilizar 3 funciones pertenecientes a la librería que inicializaran los aspectos principales del teclado alfanumérico. En primer lugar, una función que incluya la dirección de memoria asignada al periférico y lo inicialice.

Una segunda, que se utilizará para cargar una matriz alfanumérica correspondiente a los valores de cada una de las teclas del Pmod. Esta función servirá para poder relacionar el valor numérico que devolverá la fila y columna de cada tecla al ser pulsada y transformarlo en el número o letra que se desee. Por ejemplo, el teclado devolver un 48+ “el numero pulsado”. Gracias a esta matriz, el usuario solo recibirá el número que ha pulsado de manera directa.

Y, por último, habrá que utilizar una función para indicar que todas las salidas se comportaran como salidas de entrada de datos. Todas estas funciones agruparán dentro de una función creada de inicialización de Pmod KYPD (Figura 28) para esquematizar el programa.

```
void PmodKYPD_Begin() {
    KYPD_begin(&myKYPD, XPAR_PMODKYPD_0_AXI_LITE_GPIO_BASEADDR);
    KYPD_loadKeyTable(&myKYPD, (u8*) DEFAULT_KEYTABLE);
    Xil_Out32(myKYPD.GPIO_addr, 0xF);
}
```

Figura 28: Configuración del Pmod KYPD Fuente: Propia

Ya configurado el teclado, se explicará su funcionamiento (Figura 29). En primer lugar, se observa cual es el estado del teclado, es decir, que filas y columnas han sido pulsadas. Esta operación se realiza a través de la función “KYPD_getKeyStates”. Una vez se sabe si se ha pulsado una sola tecla o varias, mediante otra función se observa que tecla se ha pulsado. Para ello se utiliza la función “KYPD_getKeyPressed”. Esta función, en el caso de que solo se haya pulsado una tecla guardará el valor de esta tecla en una variable, para poder utilizarla más adelante.

```

// Capture state of each key
keystate = KYPD_getKeyStates(&myKYPD);
// Determine which single key is pressed, if any
status1 = KYPD_getKeyPressed(&myKYPD, keystate, &key);
// Print key detect if a new key is pressed or if status has changed
if (status1 == KYPD_SINGLE_KEY && (status1 != last_status || key != last_key)) {
    printf("Digito introducido: %c\r\n", (char) key);
}

```

Figura 29: Obtención de datos del Pmod KYPD Fuente Propia

Para que una misma tecla no se repita al mantenerse pulsada, se guardará el estado actual y el anterior del teclado. De esta forma, hasta que el estado no cambie, el teclado no hará nada. Todo este proceso de obtención de datos se repetirá cada vez que se quiera obtener información del teclado.

4.3.2.3. Pmod SSD:

El Pmod SSD es, con diferencia, el Pmod más difícil de configurar. Ya que, al tener que configurar su hardware con dos bloques ID para Pmod genéricos, el programa no incluirá ningún tipo de controlador o librería. Si que se dispondrá de una librería genérica para la utilización de los Pmod GPIO ya que es así como se han configurado las dos entradas Pmod a las que irá conectado.

Por tanto, para la utilización de este periférico fue necesaria la creación de una librería de control para hacer uso de él. El Pmod SSD como se ha mencionado con anterioridad es un doble visualizador de siete segmentos, que para visualizar ambos dígitos necesita alternar entre ambos a una frecuencia de 50 Hz.

Las librerías de control del Pmod se encuentran en el apartado de anexos de la memoria. Se crearon basándose en las funciones de la librería de Pmod GPIO genéricos, "PmodGPIO.h". Esta librería permite controlar las salidas de los conectores Pmod como salidas digitales.

El funcionamiento de los Pmod GPIO genéricos (Figura 30) es muy similar a los ya configurados. Primero, es necesario crear una estructura del tipo específico para cada conector y, después, inicializarlos asignando esa estructura a una dirección de memoria. Pero, en esta ocasión será necesario indicar si los pines del conector serán de salida o, de entrada, como es el caso.

```

PmodGPIO mySSD_J1;
PmodGPIO mySSD_J2;

void PmodSSD_Begin(){

    GPIO_begin(&mySSD_J1, XPAR_PMODGPIO_0_AXI_LITE_GPIO_BASEADDR, 0x00);
    GPIO_begin(&mySSD_J2, XPAR_PMODGPIO_1_AXI_LITE_GPIO_BASEADDR, 0x00);
}

```

Figura 30: Configuración del Pmod SSD Fuente: Propia

Una vez realizada la función de configuración, en la librería del Pmod SSD, se han creado dos funciones básicas para facilitar su uso. La primera de ellas es una función de visualización de un solo dígito y la segunda es una función de visualización general.

La primera de ellas es la función “display_single_number”. En esta función se han configurado las salidas digitales que están asociadas a cada segmento de los visualizadores para mostrar los números. Introduciendo una variable de tipo entero a la función con el número que se desea visualizar. Y, otra variable con el display donde se desea que se visualice. La función muestra el número por el display correspondiente que se haya asignado.

Por otro lado, se encuentra la función “display_PmodSSD”. Esta función es la que se usará en el código principal de control. Pues está diseñada para que, solamente introduciendo un número, este se muestre por el Pmod. Para lograr este proceso, la función se encarga de alternar la visualización entre ambos visualizadores y llama a la anterior función para que estos se muestren a través del periférico.

Ya explicada la librería de control junto a sus funciones, lo necesario para utilizar este periférico es hacer uso de ellas dentro del programa principal. Para ello será necesario utilizar la función de inicialización antes del comienzo del programa de control como con los otros periféricos. Y, después, la parte más compleja será la de programar una interrupción que cada 20 ms o menos llame a la función “display_PmodSSD”. Esto hará que se visualicen ambos números por el periférico.

4.3.3. Programación del timer y la interrupción:

Para configurar tanto el timer, como la interrupción asociada a este timer será necesario añadir una serie de librerías. Estas librerías permitirán que se usen las funciones específicas de configuración de estas herramientas. Son las librerías de control del timer y del “general interrupt controller” (Figura 31), encargado de controlar las interrupciones del procesador.

```
#include "xtmrctr.h"  
#include "xscugic.h"  
#include "xil_exception.h"
```

Figura 31: Librerías de control del timer y el GIC Fuente: Propia

El siguiente paso será crear la estructura de control del timer y la función que se va a utilizar como manejador de la interrupción (Figura 32).

```
XTmrCtr my_Timer;  
//-----  
//PROTOTYPE INTERRUPT HANDLER FUNCTION  
//-----  
void my_Interrupcion_Timer(void *CallbackRef);
```

Figura 32: Estructura del timer y función manejadora Fuente: Propia

Una vez añadidas estas dos cosas, se va a crear una función específica de configuración de la interrupción y el timer, llamada “interrupts_init”. Esta función contará con varias partes. La primera de ellas será definir estructuras para el “general interrupt controller” y su propia configuración.

Después se configurará el “general interrupt controller” (Figura 33) con sus funciones propias de configuración. Donde se le asignará la estructura y la dirección de memoria correspondiente.

```
XScuGic my_Gic;
XScuGic_Config *GicConfig;

//INICIALIZACIÓN DE GIC
GicConfig = XScuGic_LookupConfig(XPAR_PS7_SCUGIC_0_DEVICE_ID);
status = XScuGic_CfgInitialize(&my_Gic, GicConfig, GicConfig->CpuBaseAddress);
if(status != XST_SUCCESS) return XST_FAILURE;
```

Figura 33: Estructuras y configuración del GIC Fuente: Propia

Ya configurado en GIC, se configurará el timer. La configuración del timer es más compleja. Ya que, hay que especificar, de qué forma se quiere que actúe. El timer es un contador que hará una cuenta determinada a la velocidad del reloj del procesador. En este caso el reloj del procesador funciona a 50 MHz.

Por tanto, para configurar el timer primero habrá que asociar la función creada para manejar la interrupción a dicha interrupción. Seguidamente habrá que establecer el “Reset Value”, que es el valor que restado a 2^{32} contará el timer. Por ello, para que el timer cuente 10 ms, será necesario hacer el cálculo siguiente:

$$N = 10ms \cdot AXI \text{ Clock (50 MHz)} = 500.000$$

$$\text{Reset Value} = 2^{32} - N = 4.294.467.296$$

$$\text{Reset Value} = 0xFFF85EE0$$

Una vez obtenido este valor, las últimas configuraciones indicarán el comportamiento del timer. Se desea que se auto recargue una vez termine la cuenta descendente y llegue al 0, ejecutándose así la interrupción. Y por último se pondrá en marcha el timer (Figura 34).

```
//INICIALIZACIÓN DEL TIMER
status = XTmrCtr_Initialize(&my_Timer, XPAR_TMRCTR_0_DEVICE_ID);
if(status != XST_SUCCESS) return XST_FAILURE;

XTmrCtr_SetHandler(&my_Timer, (XTmrCtr_Handler)my_Interrupcion_Timer, (void *)&my_Timer);
XTmrCtr_SetResetValue(&my_Timer, 0, 0xFFF85EE0); //Tiempo= (2^bits_timer -Counter)*Periodo
XTmrCtr_SetOptions(&my_Timer, 0, XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);
XTmrCtr_Start(&my_Timer, 0);
```

Figura 34: Configuración del timer Fuente: Propia

Después se asignará la prioridad (Figura 35) que se quiere que tengas esta interrupción, en este caso carece de valor puesto que solo hay una, pero si en un futuro se añadiesen más, esto sería de vital importancia. La prioridad de las interrupciones se asigna en múltiplos de 4. Siendo 0 la prioridad más alta y 248 la más baja. Esta asignación hará que ante dos llamadas concurrentes de dos interrupciones, una se ejecute con prioridad respecto a otra. Al solo utilizar una interrupción, a esta se le ha dotado de la más alta prioridad.

```
//-----ASIGNACIÓN DE PRIORIDADES-----
// 0 is highest priority, 0xFB (248) is lowest, Step of 8 (0,8,16,...,248)

XScuGic_SetPriorityTriggerType(&my_Gic, XPS_FPGA0_INT_ID, 0x00, 0x1);
```

Figura 35: Asignación de prioridad de la interrupción Fuente: Propia

En el siguiente apartado se asignará la conexión entre el ID que genera la interrupción en el IRQ y la función asociada a la interrupción del timer. Y por último el GIC habilitará las interrupciones procedentes de IRQ y se habilitará el sistema de excepciones.

Una vez configurado por completo tanto el timer, como su interrupción, cuando se ejecute esta función en el código principal, cada 10 ms se accederá a la función manejadora de la interrupción. Y, seguidamente, se retomará el programa de control del sistema de alarma. Gracias a la velocidad del procesador, esto permitirá obtener datos, enviar mensajes al usuario, mostrar la cuenta atrás por el display, dentro de la interrupción sin detener el proceso de control de la alarma.

4.3.4. Programa de control del sistema de alarma:

Para la realización del programa de control del sistema de alarma se ha seguido un procedimiento de diseño por etapas. Primero, se ha diseñado un diagrama de estados (Figura 36), que la alarma irá atravesando en el transcurso del funcionamiento.

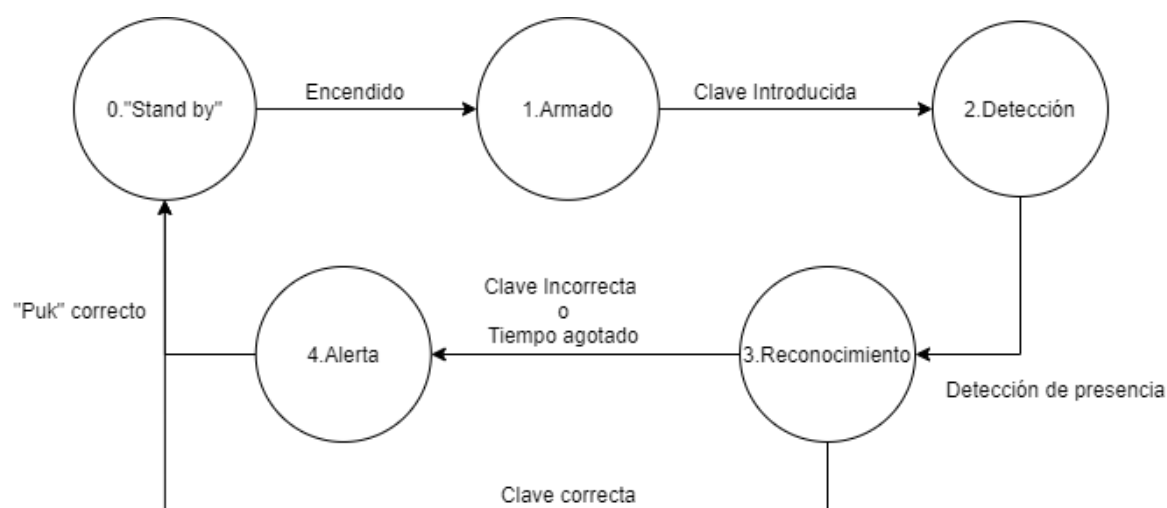


Figura 36: Diagrama de estados del funcionamiento

Fuente: Propia

En este diagrama de estados se puede ver por qué procesos pasará el sistema y que acciones harán que sistema cambie de estado. Dentro de cada estado, el programa seguirá una serie de pautas:

- 0. "Stand by": la primera vez el encendido del programa será automático, las siguientes veces habrá que pulsar el interruptor o introducir directamente la contraseña.
- 1. Armado: El programa esperará a que el usuario introduzca la clave de 4 dígitos para armar la alarma.
- 2. Detección: En esta fase se leerá la entrada del sensor hasta que se detecte presencia.

- 3. Reconocimiento: El usuario tendrá que introducir la clave de 4 dígitos correctamente antes de la finalización de una cuenta atrás. En caso de no hacerlo correctamente la alarma se activará.
- 4. Alerta: Una señal lumínica advertirá de la presencia de intrusos hasta que se introduzca el “PUK” de 6 dígitos correctamente.

Una vez decididos los estados que atravesará la alarma durante el proceso, se ha de hacer otra etapa del diseño. Y, esta será la realización de un flujograma (Figura 37) esquemático de cómo se va a estructurar el código.

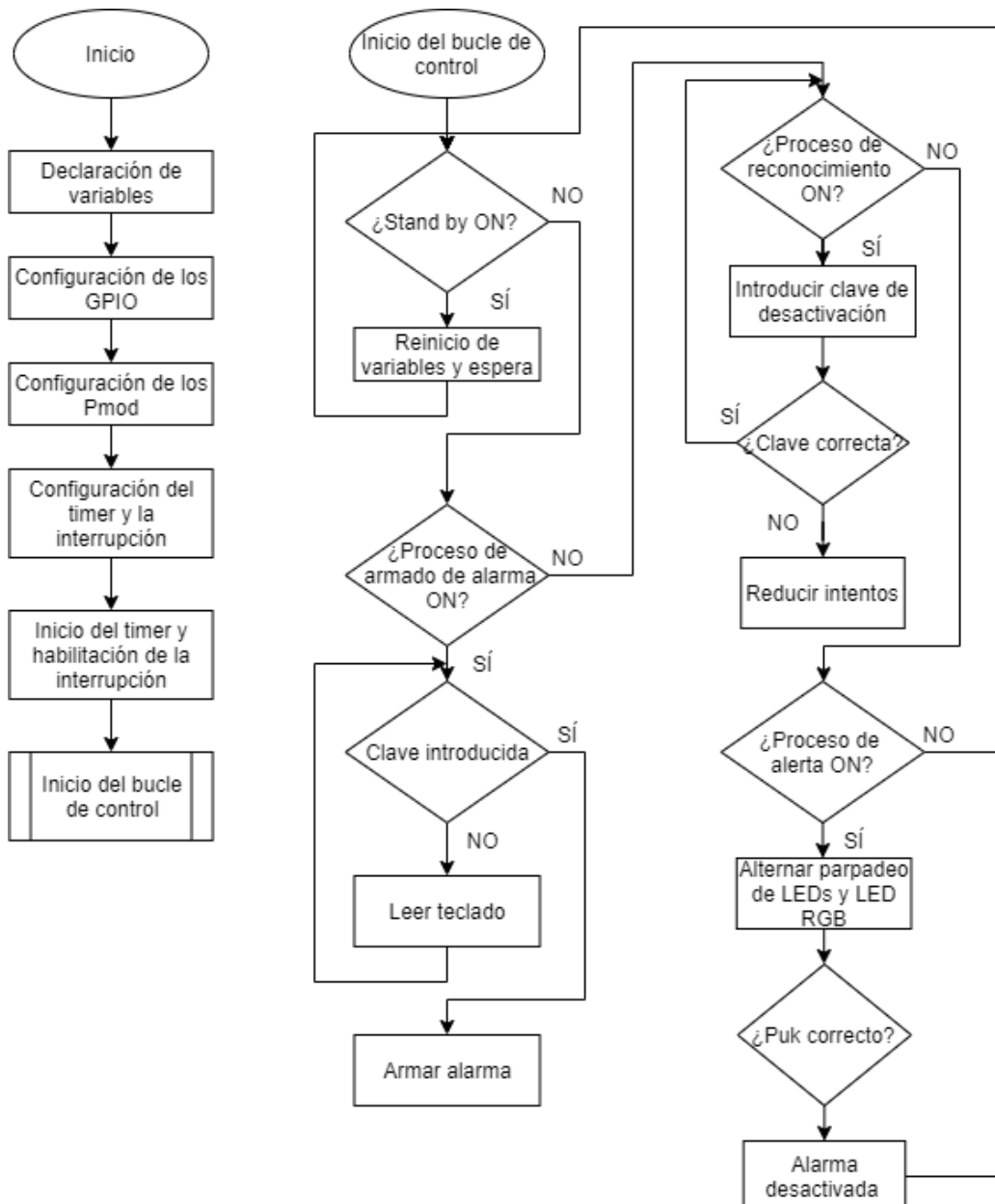


Figura 37: Flujograma del código principal

Fuente: Propia

Primero, se ha de inicializar todas las variables, periféricos y herramientas de software que se vayan a utilizar, como se ha mencionado en los pasos anteriores. Antes de comenzar con el bucle principal de control.

Ya dentro del bucle de control, se ha traducido la máquina de estados en un sistema controlado por estructuras "while". De esta forma, más adelante, se han asociado variables a las acciones que repercutían con un cambio de estado, para salir de los bucles "while" y cambiar de estado.

Dejando así el manejador de la interrupción (Figura 38) para los procesos de lectura y escritura continuos en los periféricos, y la comunicación directa con el usuario. De esta forma, el bucle principal de control no tiene que detenerse en ningún momento en procesos secundarios o concurrentes. Ya que este se ejecutará de forma recurrente cada 10 ms, y después se retomará el bucle principal.

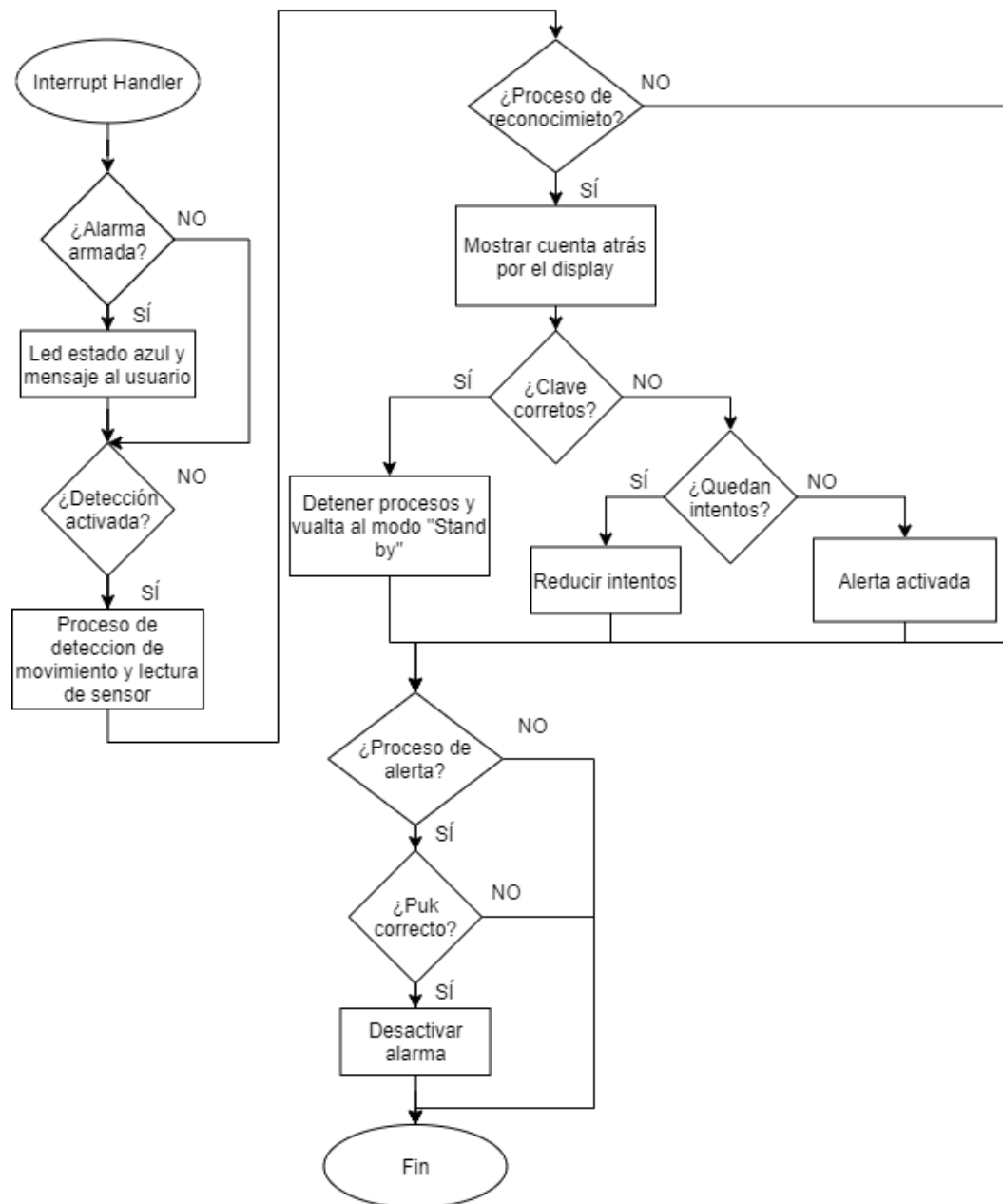


Figura 38: Flujograma del código de la interrupción

Fuente: Propia

El proceso de interrupción es completamente necesario ya que, en varios momentos del funcionamiento, el programa necesita de la realización de dos tareas simultaneas. Como, por ejemplo, mostrar la cuenta atrás en el display cada 10 ms y recoger los datos del teclado que introduce el usuario.

Una vez, detallado tanto el diagrama de estados, como el flujograma del código, solo será necesario materializar el código en C al completo. El código completo se encuentra en el apartado de anexos. Donde se habrán añadido todas las funciones y variables necesarias para el correcto desarrollo del sistema de alarma.

4.3.5. Comunicación serie con el usuario:

A lo largo del código, al usuario se le envían ciertos mensajes informativos o solicitándole información. La placa de desarrollo se comunica con el ordenador al cual está conectada a través del puerto Micro-USB. Esta comunicación necesita ser recibida y traducida por un programa compatible con comunicación serie en el ordenador.

En este proyecto se utilizará “Tera Term”, pero cualquier otro tipo de programa compatible con comunicación serie serviría, siempre que se configure con los parámetros adecuados (Figura 39).

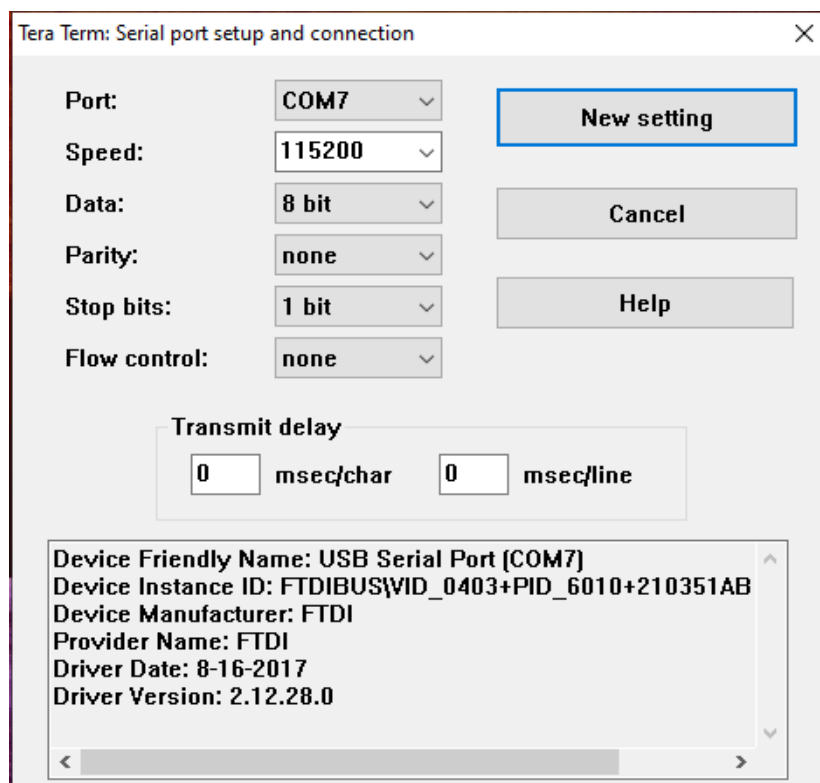
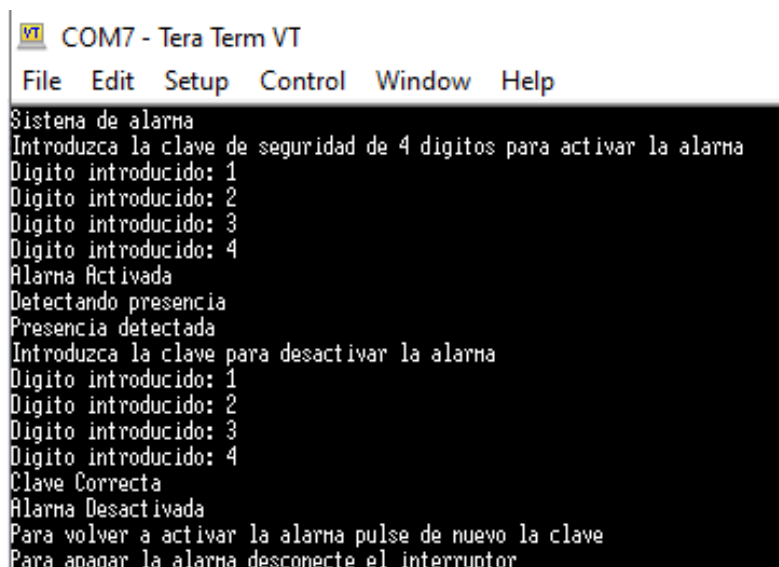


Figura 39: Parámetros de configuración de la comunicación serie

Fuente: Propia

El funcionamiento de “Tera Term” es muy básico, como el de la mayoría de los programas de comunicación similares. Para utilizarlo basta con encender el programa, elegir el tipo de comunicación y el puerto COM del ordenador al cual se ha conectado el dispositivo. Una vez se han seleccionado estos ajustes, habrá que configurar la comunicación serie y ejecutar el programa desde Vitis. Y, una vez ejecutado el programa, la placa se comunicará con el ordenador y los mensajes aparecerán por pantalla (Figura 40).



```
VT COM7 - Tera Term VT
File Edit Setup Control Window Help
Sistema de alarma
Introduzca la clave de seguridad de 4 digitos para activar la alarma
Digito introducido: 1
Digito introducido: 2
Digito introducido: 3
Digito introducido: 4
Alarma Activada
Detectando presencia
Presencia detectada
Introduzca la clave para desactivar la alarma
Digito introducido: 1
Digito introducido: 2
Digito introducido: 3
Digito introducido: 4
Clave Correcta
Alarma Desactivada
Para volver a activar la alarma pulse de nuevo la clave
Para apagar la alarma desconecte el interruptor
```

Figura 40: Mensajes de comunicación con el usuario Fuente: Propia

Este proceso, servirá para corroborar que la configuración automática de la comunicación de la placa funciona. Y, además, se podrá comprobar que el programa sigue el funcionamiento esperado y será útil para la realización de pruebas y test del sistema.

5. Test y pruebas de funcionamiento:

En el siguiente apartado comprobaremos si el funcionamiento de la placa es el deseado y se realizarán una serie de pruebas para verificar que en todos los casos posibles el programa funciona como se espera.

5.1. Test de funcionamiento:

El funcionamiento del sistema es el correcto. Cuando se enciende el programa al usuario le llegan los mensajes de arranque y de solicitud de armado programados en el código.

Seguidamente, cuando el usuario introduce la clave de acceso por el teclado, la alarma se activa y el led de estado cambia a color azul (Figura 41) (indicando que ha entrado en el proceso de detección). Mientras esta detectando, sigue enviando mensajes repetidamente para informar de que sigue detectando.

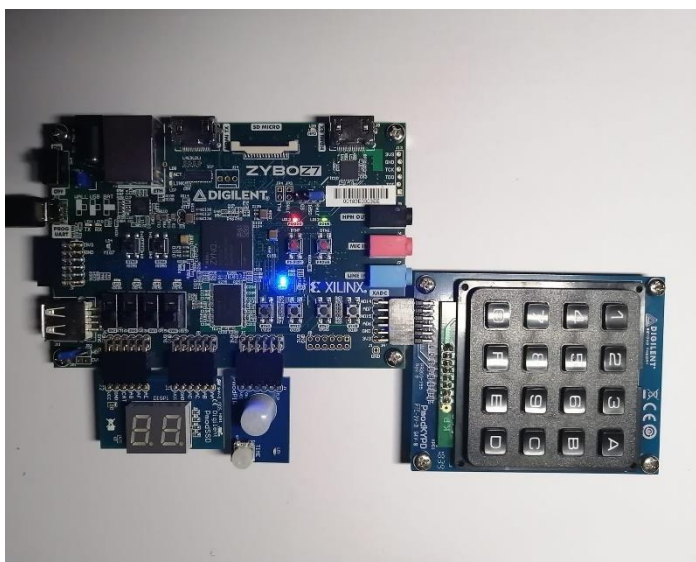


Figura 41: Estado de detección del sistema Fuente: Propia

Una vez detecta movimiento, lo comunica al usuario, a la vez que le pide la contraseña, cambia el led de estado a rojo (Figura 42) (indicando que ha detectado presencia) y empieza la cuenta atrás regresiva en el display.

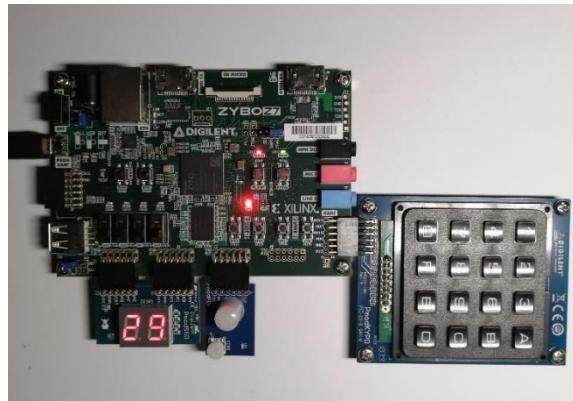
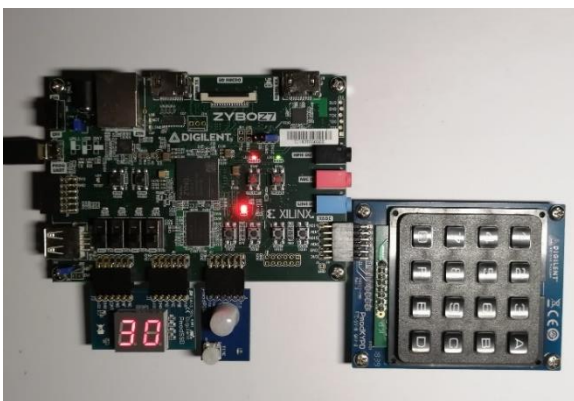


Figura 42: Estado de reconocimiento del sistema

Fuente: Propia

Si se introduce correctamente la contraseña, el led de estado cambia a verde (Figura 43), la cuenta atrás se detiene y muestra un 0 por el periférico y el sistema comunica al usuario que la alarma se ha desactivado. Además, le ofrece volver a activarla o apagarla con el interruptor.

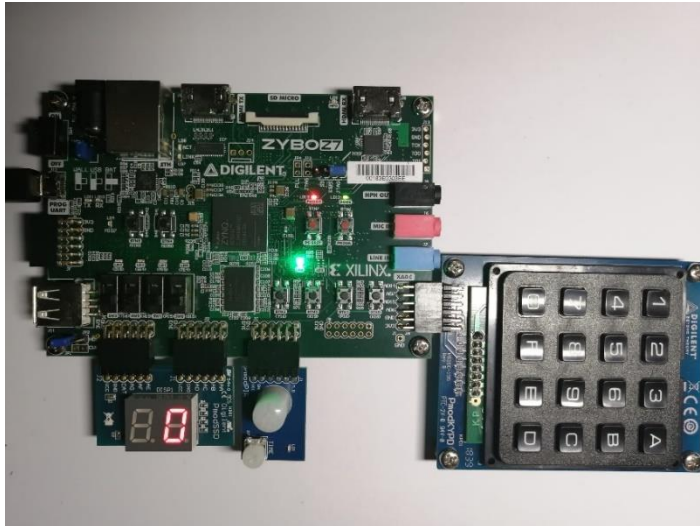


Figura 43: Estado después de introducir la clave Fuente: Propia

En el caso de errar la contraseña o terminar la cuenta atrás, la alarma comienza el modo de alerta. Los leds comienzan a parpadear y el led RGB alterna el parpadeo entre rojo y azul (Figura 44), y al usuario se le comunica por pantalla que se ha detectado a un intruso. Este modo solo se detiene si se introduce correctamente, por el teclado, el “PUK” de 6 dígitos preconfigurado (492560).

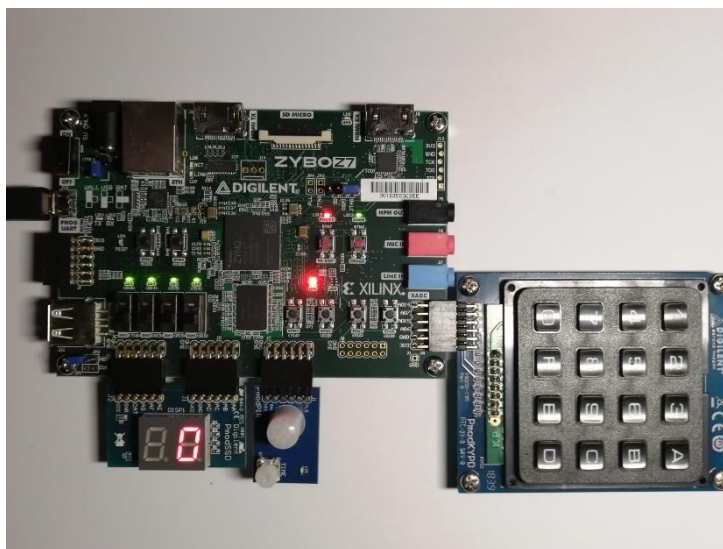


Figura 44: Estado de alerta Fuente: Propia

Por último, cabe destacar, que tanto el interruptor, como el sensor PIR, como el teclado, funcionan como era esperado. El sensor reacciona ante el movimiento de una forma sensible y correcta. Y, el teclado obtiene la información del usuario sin ningún error y de manera instantánea.

5.2. Pruebas de funcionamiento:

En el presente apartado se realizarán una serie de pruebas específicas para la comprobación de ciertos funcionamientos concretos del programa (Tablas de la 3 a la 8).

Prueba 1:

Prueba 1	Introducción de la clave
Objetivo	Comprobar que a través del teclado se guarda e introduce correctamente la contraseña.
Descripción	La contraseña se introduce de forma correcta. Solo permite la introducción de 4 dígitos. Y, se puede comprobar que se guarda de la forma esperada puesto que es necesario introducir la misma a la hora de la desactivación.
Resultado	Superada

Tabla 3: Prueba de funcionamiento 1 Fuente: Propia

Prueba 2:

Prueba 2	Fallo de clave
Objetivo	Comprobar que solo se disponen de 3 intentos. para acertar la clave.
Descripción	Efectivamente cuando se falla la clave por tercera vez, la alarma se activa inmediatamente.
Resultado	Superada

Tabla 4: Prueba de funcionamiento 2 Fuente: Propia

Prueba 3:

Prueba 3	Cuenta atrás
Objetivo	Comprobar cuenta atrás es regresiva unidad a unidad y al llegar a cero activa la alarma.
Descripción	La cuenta atrás funciona de la forma esperada, contando los 30 segundo de forma progresiva y al llegar a su fin activa la alarma.
Resultado	Superada

Tabla 5: Prueba de funcionamiento 3 Fuente: Propia

Prueba 4:

Prueba 4	Sensor de presencia
Objetivo	El sensor ha de detectar presencia en radio de acción de hasta 5 metros.
Descripción	El sistema detecta presencia a 5 metros siempre que sea en la dirección directa a la que apunta este.
Resultado	Superada

Tabla 6: Prueba de funcionamiento 4 Fuente: Propia

Prueba 5:

Prueba 5	"PUK"
Objetivo	Una vez activada la alarma, solo se desactivará introduciendo los 6 dígitos: 492560.
Descripción	Una vez ha saltado la alarma solo se desactiva introduciendo el "PUK" de forma consecutiva.
Resultado	Superada

Tabla 7: Prueba de funcionamiento 5 Fuente: Propia

Prueba 6:

Prueba 6	Reinicio del sistema
Objetivo	Comprobar que, una vez desactivada la alarma, haciendo uso del interruptor, el sistema se apagar y vuelve a iniciarse correctamente.
Descripción	El uso del interruptor deja la alarma en un estado de "Stand by", hasta que vuelve a activarse y el sistema se inicia de nuevo.
Resultado	Superada

Tabla 8: Prueba de funcionamiento 6 Fuente: Propia

6. Presupuesto:

En el presente apartado se encuentra el presupuesto necesario para realizar el trabajo que se ha mostrado a lo largo del documento. Para ello, primero se realizará un desglose de gastos en función de su naturaleza. Es decir, dividir los costes según si están asociados al uso de determinado hardware, software o provienen del coste del personal que ha realizado el trabajo. Y, por último, se realizará un sumario de los costes totales que ha supuesto el proyecto.

Para el cálculo correcto de los costes se tendrá en cuenta que la realización del trabajo final de grado representa 12 créditos ECTS (European Credit Transfer and Accumulation System). Considerando que cada crédito supone una inversión de 25 horas de trabajo según el Plan Bolonia, por tanto, el número total de horas necesarias para la realización del proyecto 300 horas.

6.1. Desglose de costes en función de su naturaleza:

El desglose de costes en función de su naturaleza se hará dividiendo los costes del proyecto en 3 ámbitos: costes de hardware, costes de software y coste humano.

6.1.1. Coste del hardware:

En la realización del trabajo han sido necesarios los materiales que previamente se han mencionado en el documento. Pero, también se ha requerido otro tipo de herramientas que han permitido desarrollar el proyecto y documentarlo (Tabla 9).

El hardware que compone el proyecto es el siguiente: la tarjeta de desarrollo Zybo Z7-10 y los Pmod SSD, KYPD y PIR de Digilent. Para la realización del proyecto se ha utilizado ordenador portátil modelo Asus Zenbook ux430u, con un procesador Intel Core i7-7500U, 8 GB de memoria RAM DDR3, almacenamiento de 256 GB de memoria SSD y tarjeta grafica Intel UHD 620.

Teniendo en cuenta, que para la realización del trabajo se ha utilizado el ordenador en todo momento, se habrán consumido 300 horas de vida útil de este, a una media de 8 horas diarias, serian 37 días y medio. Teniendo en cuenta que la vida de un portátil oscila entre 3 a 6 años, aproximadamente se ha utilizado un 2,3 % del coste total del dispositivo.

Coste del hardware	Coste total (€)	Coste amortizado (€)
Zybo Z7-10	194,63	194,63
Pmod SSD	5,86	5,86
Pmod KYPD	8,38	8,38
Pmod PIR	12,57	12,57
Ordenador portátil	1.049,00	24,13
Total	1.270,44	245,57

Tabla 9: Costes del hardware Fuente: Propia

6.1.2. Costes del software:

Para la realización del proyecto se han utilizado 3 programas específicos para el uso del hardware seleccionado. Como se ha mencionado a lo largo del documento estos programas han sido Vivado, Vitis y Tera Term. Las licencias para el uso de estos programas son completamente gratuitas, por tanto, el software utilizado durante el proyecto no ha supuesto coste alguno.

También se ha utilizado el paquete de programas Microsoft Office 365, cuya licencia proporciona la universidad a todos los estudiantes, aunque el valor de esta es de 69 € anuales.

6.1.3. Coste humano:

Para la realización del presente trabajo se han dedicado 300 horas de recursos humanos por el autor. Estimando un coste aproximado de 20€/h por el trabajo realizado por el autor del trabajo, se muestra un desglose asociado a la realización de las tareas requeridas para llevar a cabo el proyecto (Tabla 10).

Tareas	Coste temporal (h)	Coste (€)
Documentación e investigación	30	600
Aprendizaje de utilización del software	30	600
Aprendizaje de utilización del hardware	50	1.000
Programación del hardware	40	800
Programación del software	50	1.000
Validación y pruebas de funcionamiento	10	200
Redacción de la memoria	90	1.800
Total	300	6.000

Tabla 10: Coste humano Fuente: Propia

6.2. Costes totales:

Finalmente se resumen los costes totales para la realización del proyecto (Tabla 11).

Desglose de costes totales	
Costes de hardware	245,57€
Costes de software	0€
Coste humano	6.000€
Total	6.245,57€

Tabla 11: Costes totales

Por tanto, la realización del trabajo supone un coste total de 6.245,57€.

7. Conclusiones:

7.1. Valoración personal:

El proyecto ha conseguido su objetivo principal, el desarrollo de un sistema de alarma que alerte de la presencia de intrusos. Detectando el movimiento y permitiendo al usuario interactuar con él para su configuración y activación.

La principal ventaja de este proyecto no ha sido el desarrollo del sistema de alarma. Si no, el aprendizaje que ha conllevado desarrollar este sistema. Gracias a la utilización de las herramientas y materiales que se han requerido para la realización del proyecto. Puesto que muchas de ellas eran nuevas y no se había utilizado anteriormente.

Durante los estudios de grado, en las asignaturas de sistemas embebidos se ha trabajado con microcontroladores y se ha realizado la programación del software. Pero este trabajo ha sido la primera oportunidad que se me presentaba para aprender sobre las FPGA y la programación y personalización del hardware a este nivel. Para posteriormente poder implementar un programa en dicho hardware.

He podido aprender y practicar en otro microcontrolador, en el cual no había trabajado hasta ahora, todas las herramientas que he adquirido a lo largo de la carrera. Y me he introducido en el campo de las FPGA, unos sistemas que ofrecen infinitas posibilidades gracias a su adaptabilidad y reprogramabilidad.

Los sistemas embebidos son una parte fundamental de la electrónica. Puesto que, cualquier aparato electrónico necesita de uno de estos para controlar sus procesos. Y, cada uno de ellos, aunque similares, son muy diferentes a la hora de trabajar y configurarlos. Haber podido ampliar mis conocimientos en este campo y haber aprendido a ser más versátil y mejorar la faceta de autoaprendizaje sobre estos. Ha hecho que aprenda a utilizar toda la base de conocimientos adquiridos en el grado de una forma práctica.

7.2. Futuras ampliaciones y posibles mejoras:

Este proyecto se trata de un prototipo inicial de un sistema de alarma. Por lo que en un futuro se podrían realizar ampliaciones para mejorar el sistema. Estas mejoras se dividirán en dos partes: mejoras de instalación y mejoras en el sistema.

Para mejorar la instalación del sistema de alarma será necesario llevar a cabo las siguientes mejoras:

- Encapsular el sistema: para que no se pueda acceder de forma sencilla a la parte electrónica desde el exterior y solo queden visibles las parte manipulables.

- Cableado: será necesario alargar el cableado para colocar de forma efectiva el sensor y el resto de los periféricos.
- Alimentación: diseñar un regulador de tensión que alimente el sistema desde la red eléctrica:
- Interfaz de usuario: al no conectar el sistema a un ordenador, deberá comunicarse con el usuario a través de una pantalla instalada.

Estas son las mejoras que se requerirían para instalar el prototipo en una vivienda o un comercio.

En cuanto a las mejoras para el sistema, son las siguientes:

- Sistema de comunicación: conectar el sistema con el usuario a distancia a través de internet. La placa cuenta con una entrada Ethernet que le dotaría de conexión a internet para avisar al usuario de la intrusión a distancia. Mediante una app móvil o un sistema de mensajería gratuito.
- Videovigilancia: la placa también cuenta con una entrada compatible con cámara, además de un sistema de procesamiento de imagen excepcional. Por tanto, se podría conectar la cámara simplemente grabar cuando se detecte presencia. O, otra alternativa sería utilizar la cámara como sensor. Tratando la imagen con un programa de visión artificial que detectase movimiento.
- Ampliar el número de sensores: esta mejora también afectaría a la instalación, puesto que podría instalarse en lugares más grandes o completar la seguridad de un mismo recinto. Pero esto requeriría cambiar la placa a la Zybo Z7-20, que cuenta con otra entrada compatible con Pmod. Otra solución sería utilizar sensores que requieran de menos conexiones.
- Alerta sonora: la placa de desarrollo cuenta con salida de audio, a la cual se le podría conectar una red de altavoces que ejecutarán un sonido de alerta al hacer saltar la alarma.
- Seguridad biométrica: se podría cambiar la clave de seguridad por un lector de huellas dactilares. Hay varios modelos de lectores de huellas que transmiten su información de forma digital, compatibles con las entradas Pmod de la placa Zybo Z7.

8. Bibliografía:

Abus. (s. f.). *Historia de las alarmas*. Abus.com. Recuperado 14 de junio de 2021, de <https://www.abus.com/es/Guia/Proteccion-antirrobo/Sistemas-de-alarma/Historia-de-los-sistemas-de-alarma>

Prosegur S.A. (s. f.). *Tipos de sistemas de alarmas*. Blog.prosegur.es. Recuperado 14 de junio de 2021, de <https://blog.prosegur.es/alarmas-tipos/>

Pedre, S. (2017). *Sistemas embebidos. Laboratorio de Robótica y Sistemas Embebidos, Departamento de computación FCEN UBA.*

Aviles Salazar, A. D., & Cobeña Mite, K. (2015). *Diseño e implementación de un sistema de seguridad a través de cámaras, sensores y alarma, monitorizado y controlado teleméricamente para el centro de acogida "Patio mi Pana" perteneciente a la fundación proyecto salesiano* (Bachelor's thesis).

Vallejo, M. L., & Rodrigo, J. A. (2004). *FPGA: Nociones básicas e implementación. Laboratorio de Diseño Microelectrónico, 4º Curso, P94.*

Bobrowicz, S. (n.d.). Zybo Z7-20 SDSoC Platform. [online] GitHub. Disponible en: <https://github.com/Digilent/SDSoC-Zybo-Z7-20>

Digilent (2017). Seminario ZYBO Z7 Video Workshop

Xilinx.Inc. (s. f.). *Programación de sistemas embebidos con procesadores Xilinx*. Xilinx.com. Recuperado 14 de junio de 2021, de https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/embeddedplatforms.html

Xilinx.Inc. (s. f.). *Información sobre el programa Vivado*. Xilinx.com. Recuperado 15 de junio de 2021, de <https://www.xilinx.com/products/design-tools/vivado.html>

Digilent.Inc. (s. f.). *Manual de referencia de la placa Zybo Z7*. Reference.digilentinc.com. Recuperado 15 de abril de 2021, de <https://reference.digilentinc.com/programmable-logic/zybo-z7/reference-manual>

Digilent.Inc. (s. f.). *Manual de referencia del Pmod SSD*. Reference.digilentinc.com. Recuperado 15 de junio de 2021, de <https://reference.digilentinc.com/pmod/pmodssd/reference-manual?redirect=1>

Diligent.Inc. (s. f.). *Manual de referencia del Pmod KYPD*. Reference.digilentinc.com. Recuperado 15 de junio de 2021, de <https://reference.digilentinc.com/pmod/pmodkypd/start?redirect=1>

Diligent.Inc. (s. f.-b). *Manual de referencia del Pmod PIR*. Reference.digilentinc.com. Recuperado 14 de abril de 2021, de <https://reference.digilentinc.com/pmod/pmodpir/start>

Digilent.Inc. (s. f.). *Tutoriales de uso de las herramientas de Digilent*. Reference.digilentinc.com. Recuperado 14 de abril de 2021, de <https://reference.digilentinc.com/learn/tutorials/start>

Xilinx (2017). *SDSoC Environment User Guide (UG1027)*. [online] Disponible en: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1027-sdsoc-user-guide.pdf

Xilinx (2017). SDSoc Environment User Guide: Introduction (UG1028). [online]
Disponibile en:
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1028-sdsoc-intro-tutorial.pdf

9. Anexos:

9.1. Código del programa principal (Main.c):

```
#include "xparameters.h"
#include "xil_printf.h"
#include "stdio.h"
#include "sleep.h"

#include "xgpio.h"
#include "xil_types.h"
#include "xtmrctr.h"
#include "xscugic.h"
#include "xil_exception.h"

#include "PmodSSD.h"
#include "PmodKYPD.h"
#include "PmodPIR.h"

// Get device IDs from xparameters.h
#define BTN_ID XPAR_AXI_GPIO_BUTTONS_DEVICE_ID
#define LED_ID XPAR_AXI_GPIO_LEDS_DEVICE_ID
#define LED_RGB_ID XPAR_AXI_GPIO_RGB_LED_DEVICE_ID
#define SW_ID XPAR_AXI_GPIO_SWITCHES_DEVICE_ID

#define BTN_CHANNEL 1
#define LED_CHANNEL 1
#define SW_CHANNEL 1
#define LED_RGB_CHANNEL 1

#define BTN_MASK 0b1111
#define LED_MASK 0b0000
#define SW_MASK 0b1111
#define LED_RGB_MASK 0b0000

//Pmod KYPD keytable is determined as follows (indices shown in Keypad
position below)
// 12 13 14 15
// 8 9 10 11
// 4 5 6 7
// 0 1 2 3
#define DEFAULT_KEYTABLE "0FED789C456B123A"

XTmrCtr my_Timer;
PmodKYPD myKYPD;
PmodPIR myPIR;

XGpio_Config *cfg_ptr;
XGpio led_device, btn_device, sw_device, led_rgb_device;

int cuenta=0, segundo=30, countdown=0, presencia=0, deteccion=0,
alarma_activada=0,desactivacion=0, cambio=0,
posicion=0, clave_introducida=0, i=0, fallo_clave=5, fallo_puk=5, intentos=3,
alarma=0, reinicio=0, encendido=0, puk[6]={52,57,50,53,54,48};
```

```

//-----
//PROTOTYPE INTERRUPT HANDLER FUNCTION
//-----
void my_Interrupcion_Timer(void *CallbackRef);

//-----
//INITIAL INTERRUPT SETUP FUNCTION
//-----

int interrupts_init(){

    int status;

//-----CONFIGURACIÓN DE INTERRUPCIONES-----

    XScuGic my_Gic;
    XScuGic_Config *GicConfig;

    //INICIALIZACIÓN DE GIC
    GicConfig = XScuGic_LookupConfig(XPAR_PS7_SCUGIC_0_DEVICE_ID);
    status = XScuGic_CfgInitialize(&my_Gic, GicConfig, GicConfig-
>CpuBaseAddress); //Initialize the GIC using the config information
    if(status != XST_SUCCESS) return XST_FAILURE;

    //INICIALIZACIÓN DEL TIMER
    status = XTmrCtr_Initialize(&my_Timer, XPAR_TMRCTR_0_DEVICE_ID);
    if(status != XST_SUCCESS) return XST_FAILURE;

    XTmrCtr_SetHandler(&my_Timer, (XTmrCtr_Handler)my_Interrupcion_Timer,
(void *)&my_Timer);
    XTmrCtr_SetResetValue(&my_Timer, 0, 0xFFF85EE0); //Tiempo=
(2^bits_timer -Counter)*Tperiodo (Frecuencia AXI)
    XTmrCtr_SetOptions(&my_Timer, 0, XTC_INT_MODE_OPTION |
XTC_AUTO_RELOAD_OPTION);
    XTmrCtr_Start(&my_Timer, 0);

//-----ASIGNACIÓN DE PRIORIDADES-----
    // 0 is highest priority, 0xFB (248) is lowest, Step of 8
(0,8,16,...,248)

    XScuGic_SetPriorityTriggerType(&my_Gic, XPS_FPGA0_INT_ID, 0x00, 0x1);

//-----CONEXIÓN DE LAS INTERRUPCIONES DRIVERS/USUARIO-----

    status = XScuGic_Connect(&my_Gic, XPS_FPGA0_INT_ID,
(Xil_ExceptionHandler)my_Interrupcion_Timer, (void *)&my_Gic); //Connect our
interrupt handler for out IRQ (TIMER 0)
    if(status != XST_SUCCESS) return XST_FAILURE;
    XScuGic_Enable(&my_Gic, XPS_FPGA0_INT_ID); //Enable the interrupt
*input* on the GIC for the external IRQ

//-----CONFIGURACIÓN DEL SISTEMA DE INTERRUPCIONES-----

    Xil_ExceptionInit();
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT, (Xil_ExceptionHan
dler)XScuGic_InterruptHandler, &my_Gic); //Register ther interrupt controller
handler with the exception table
    Xil_ExceptionEnable(); //Enable exceptions.

    return XST_SUCCESS;
}

```

```

//-----
//INITIAL PModKYPD SETUP FUNCTION
//-----

void PmodKYPD_Begin() {
    KYPD_begin(&myKYPD, XPAR_PMODKYPD_0_AXI_LITE_GPIO_BASEADDR);
    KYPD_loadKeyTable(&myKYPD, (u8*) DEFAULT_KEYTABLE);
    Xil_Out32(myKYPD.GPIO_addr, 0xF);
}

int main() {

    u16 keystate;
    XStatus status1, last_status = KYPD_NO_KEY;
    u8 key, last_key = 'x';

    int status;

    int clave[4], clave1[4];

    printf("Sistema de alarma\r\n");
    printf("Introduzca la clave de seguridad de 4 digitos para activar la
alarma\r\n");

    // Initialize LED Device
    cfg_ptr = XGpio_LookupConfig(LED_ID);
    XGpio_CfgInitialize(&led_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Initialize Button Device
    cfg_ptr = XGpio_LookupConfig(BTN_ID);
    XGpio_CfgInitialize(&btn_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Initialize Switch Device
    cfg_ptr = XGpio_LookupConfig(SW_ID);
    XGpio_CfgInitialize(&sw_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Initialize LED RGB Device
    cfg_ptr = XGpio_LookupConfig(LED_RGB_ID);
    XGpio_CfgInitialize(&led_rgb_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Set Button Tristate
    XGpio_SetDataDirection(&btn_device, BTN_CHANNEL, BTN_MASK);

    // Set Led Tristate
    XGpio_SetDataDirection(&led_device, LED_CHANNEL, LED_MASK);

    // Set Button Tristate
    XGpio_SetDataDirection(&sw_device, BTN_CHANNEL, SW_MASK);

    // Set Led Tristate
    XGpio_SetDataDirection(&led_rgb_device, LED_RGB_CHANNEL, LED_RGB_MASK);

    //Set PmodSSD
    PmodSSD_Begin();

    //Set PmodKYPD
    PmodKYPD_Begin();

    //Set PmodPIR
    PIR_begin(&myPIR, XPAR_PMODPIR_0_AXI_LITE_GPIO_BASEADDR);

```

```

//Set Timer Interrupt
status = interrupts_init();
if(status != XST_SUCCESS) return XST_FAILURE;

countdown=0;
deteccion=0;

while (1) {
while(0b0001==XGpio_DiscreteRead(&sw_device,SW_CHANNEL)){
//Proceso de Stand by
XGpio_DiscreteWrite(&led_rgb_device, LED_RGB_CHANNEL, 0b000);
cuenta=0;
segundo=30;
countdown=0;
presencia=0;
deteccion=0;
alarma_activada=0;
desactivacion=0;
fallo_clave=5;
intentos=3;
alarma=0;
posicion=0;
clave_introducida=0;
i=0;
reinicio=1;
}
while(clave_introducida==0 &&
0b0000==XGpio_DiscreteRead(&sw_device,SW_CHANNEL)){ //Proceso de armado
//-----
// Capture state of each key
keystate = KYPD_getKeyStates(&myKYPD);
// Determine which single key is pressed, if any
status1 = KYPD_getKeyPressed(&myKYPD, keystate, &key);
// Print key detect if a new key is pressed or if status
has changed
if (status1 == KYPD_SINGLE_KEY && (status1 != last_status
|| key != last_key)) {
printf("Digito introducido: %c\r\n", (char) key);
last_key = key;
clave[posicion]= key;
posicion++;
}
last_status = status1;

if(posicion==4){
alarma_activada=1;
sleep(3);
deteccion=1;
posicion=0;
clave_introducida=1;
}
usleep(1000);
//-----
}
while(desactivacion==1 && alarma==0){ //Proceso de
reconocimiento
// Capture state of each key
keystate = KYPD_getKeyStates(&myKYPD);
// Determine which single key is pressed, if any
status1 = KYPD_getKeyPressed(&myKYPD, keystate, &key);

```

```

// Print key detect if a new key is pressed or if status
has changed
    if (status1 == KYPD_SINGLE_KEY && (status1 != last_status
|| key != last_key)) {
        printf("Digito introducido: %c\r\n", (char) key);
        last_key = key;
        clave1[posicion]= key;
        posicion++;
    }
    last_status = status1;

    if(posicion==4){
        for(i=0;i<4;i++){
            if(clave[i]!=clave1[i]){
                fallo_clave=1;
            }
        }
        if(fallo_clave==1){
            intentos--;
            fallo_clave=1;
            posicion=0;
        }else{
            fallo_clave=0;
            desactivacion=0;
        }
    }

    usleep(1000);
}
while(alarma==1){ //Proceso de alerta
    if(cambio==0){
        XGpio_DiscreteWrite(&led_rgb_device,
LED_RGB_CHANNEL, 0b001);
        XGpio_DiscreteWrite(&led_device, LED_CHANNEL, 0);
        usleep(250000);
        cambio=1;
    }else{
        XGpio_DiscreteWrite(&led_rgb_device,
LED_RGB_CHANNEL, 0b100);
        XGpio_DiscreteWrite(&led_device, LED_CHANNEL, 15);
        usleep(250000);
        cambio=0;
    }

    // Capture state of each key
    keystate = KYPD_getKeyStates(&myKYPD);
    // Determine which single key is pressed, if any
    status1 = KYPD_getKeyPressed(&myKYPD, keystate, &key);
    // Print key detect if a new key is pressed or if status
has changed
    if (status1 == KYPD_SINGLE_KEY && (status1 != last_status
|| key != last_key)) {
        printf("Digito introducido: %c\r\n", (char) key);
        last_key = key;
        clave1[posicion]= key;
        posicion++;
    }
    last_status = status1;

    if(posicion==6){

```

```

        for(i=0;i<6;i++){
            if(puk[i]!=clave1[i]){
                fallo_puk=1;
            }
        }
        if(fallo_puk==1){
            posicion=0;
        }else{
            fallo_clave=0;
            XGpio_DiscreteWrite(&led_device, LED_CHANNEL, 0);
            desactivacion=0;
            alarma=0;
        }
    }
}

}

}

}

//-----
// INTERRUPT HANDLER FUNCTION
// - called by the timer, interrupt, performs
//-----
void my_Interrupcion_Timer(void *Callbackref){
    if(XTmrCtr_IsExpired(&my_Timer, 0)){
        XTmrCtr_Stop(&my_Timer,0);

        //-----
        if(alarma_activada==1){
            printf("Alarma Activada\r\n");
            XGpio_DiscreteWrite(&led_rgb_device, LED_RGB_CHANNEL, 0b001);
            alarma_activada=0;
        }
        //-----
        if(deteccion==1){
            cuenta++;
            presencia = PIR_getState(&myPIR);
            if(cuenta==500){
                printf("Detectando presencia\r\n");
                cuenta=0;
            }
            if(presencia==1){
                printf("Presencia detectada \r\n");
                printf("Introduzca la clave para desactivar la alarma
\r\n");
                cuenta=0;
                deteccion=0;
                desactivacion=1;
                countdown=1;
            }
        }
        //-----
        if(countdown==1){
            XGpio_DiscreteWrite(&led_rgb_device, LED_RGB_CHANNEL, 0b100);
            if(segundo==0){
                display_PmodSSD(segundo);
                printf("Intruso Detectado \r\n");
                alarma=1;
                countdown=0;
            }
            display_PmodSSD(segundo);

```

```

        cuenta++;
        if(cuenta==100){
            segundo--;
            cuenta=0;
        }
    }
    //-----
    if(fallo_clave==1){
        if(intentos==0){
            printf("Clave Incorrecta \r\n");
            printf("Intruso Detectado \r\n");
            alarma=1;
            display_PmodSSD(0);
            countdown=0;
            fallo_clave=5;
        }else{
            printf("Clave Incorrecta \r\n");
            printf("%d Intentos restantes \r\n",intentos);
            fallo_clave=5;
        }
    }
    //-----
    if(fallo_clave==0){
        countdown=0;
        display_PmodSSD(0);
        XGpio_DiscreteWrite(&led_rgb_device, LED_RGB_CHANNEL, 0b010);
        printf("Clave Correcta \r\n");
        printf("Alarma Desactivada \r\n");
        printf("Para volver a activar la alarma pulse de nuevo la clave
\r\n");
        printf("Para apagar la alarma desconecte el interruptor \r\n");
        fallo_clave=5;
        encendido=0;
        cuenta=0;
        segundo=30;
        countdown=0;
        presencia=0;
        deteccion=0;
        alarma_activada=0;
        desactivacion=0;
        intentos=3;
        alarma=0;
        posicion=0;
        clave_introducida=0;
        i=0;
    }
    //-----
    if(reinicio==1 && 0b0000==XGpio_DiscreteRead(&sw_device,SW_CHANNEL)){
        printf("Sistema de alarma\r\n");
        printf("Introduzca la clave de seguridad de 4 digitos para
activar la alarma\r\n");
        reinicio=0;
    }
    //-----
    if(fallo_puk==1){
        printf("Puk Incorrecto \r\n");
        fallo_puk=5;
    }
    //-----

```

```

    XTmrCtr_Reset(&my_Timer, 0);
    XTmrCtr_Start(&my_Timer, 0);
}
}

```

9.2. Librería Pmod SSD:

9.2.1. Pmod SSD.c:

```

/*****
/*
/*     Archivo.c de configuración de funciones del PmodSSD
/*
/*
/*****

#include "PmodSSD.h"

PmodGPIO mySSD_J1;
PmodGPIO mySSD_J2;

int izquierda=0;

void PmodSSD_Begin(){

    GPIO_begin(&mySSD_J1, XPAR_PMODGPIO_0_AXI_LITE_GPIO_BASEADDR, 0x00);
    GPIO_begin(&mySSD_J2, XPAR_PMODGPIO_1_AXI_LITE_GPIO_BASEADDR, 0x00);
}

void display_PmodSSD(int numero){
    int decenas, unidades;
    if(numero<10){
        display_single_number(numero, 0);
    }
    else{
        decenas=numero/10;
        unidades=numero-decenas*10;
        if(izquierda==0){
            display_single_number(unidades, 0);
            izquierda=1;
        }
        else{
            display_single_number(decenas, 1);
            izquierda=0;
        }
    }
}

void display_single_number(int numero, int digitos){
    if(digitos==0){
        switch(numero){
            case 0:
                GPIO_setPins(&mySSD_J1, 0b1111);
                GPIO_setPins(&mySSD_J2, 0b0011);
                break;
            case 1:
                GPIO_setPins(&mySSD_J1, 0b0110);
                GPIO_setPins(&mySSD_J2, 0b0000);

```



```

        break;
    case 2:
        GPIO_setPins(&mySSD_J1, 0b1011);
        GPIO_setPins(&mySSD_J2, 0b0101);
        break;
    case 3:
        GPIO_setPins(&mySSD_J1, 0b1111);
        GPIO_setPins(&mySSD_J2, 0b0100);
        break;
    case 4:
        GPIO_setPins(&mySSD_J1, 0b0110);
        GPIO_setPins(&mySSD_J2, 0b0110);
        break;
    case 5:
        GPIO_setPins(&mySSD_J1, 0b1101);
        GPIO_setPins(&mySSD_J2, 0b0110);
        break;
    case 6:
        GPIO_setPins(&mySSD_J1, 0b1101);
        GPIO_setPins(&mySSD_J2, 0b0111);
        break;
    case 7:
        GPIO_setPins(&mySSD_J1, 0b0111);
        GPIO_setPins(&mySSD_J2, 0b0000);
        break;
    case 8:
        GPIO_setPins(&mySSD_J1, 0b1111);
        GPIO_setPins(&mySSD_J2, 0b0111);
        break;
    case 9:
        GPIO_setPins(&mySSD_J1, 0b1111);
        GPIO_setPins(&mySSD_J2, 0b0110);
        break;
    }
}
else{
    switch(numero){
    case 0:
        GPIO_setPins(&mySSD_J1, 0b1111);
        GPIO_setPins(&mySSD_J2, 0b1011);
        break;
    case 1:
        GPIO_setPins(&mySSD_J1, 0b0110);
        GPIO_setPins(&mySSD_J2, 0b1000);
        break;
    case 2:
        GPIO_setPins(&mySSD_J1, 0b1011);
        GPIO_setPins(&mySSD_J2, 0b1101);
        break;
    case 3:
        GPIO_setPins(&mySSD_J1, 0b1111);
        GPIO_setPins(&mySSD_J2, 0b1100);
        break;
    case 4:
        GPIO_setPins(&mySSD_J1, 0b0110);
        GPIO_setPins(&mySSD_J2, 0b1110);
        break;
    case 5:
        GPIO_setPins(&mySSD_J1, 0b1101);
        GPIO_setPins(&mySSD_J2, 0b1110);
        break;
    }
}

```

```

        case 6:
            GPIO_setPins(&mySSD_J1, 0b1101);
            GPIO_setPins(&mySSD_J2, 0b1111);
            break;
        case 7:
            GPIO_setPins(&mySSD_J1, 0b0111);
            GPIO_setPins(&mySSD_J2, 0b1000);
            break;
        case 8:
            GPIO_setPins(&mySSD_J1, 0b1111);
            GPIO_setPins(&mySSD_J2, 0b1111);
            break;
        case 9:
            GPIO_setPins(&mySSD_J1, 0b1111);
            GPIO_setPins(&mySSD_J2, 0b1110);
            break;
    }
}
}

```

9.2.2. Pmod SSD.h:

```

/*****
/*
/*     Archivo.h de configuración de funciones del PmodSSD
/*
/*
/*****

#ifndef PmodSSD_H
#define PmodSSD_H

/***** Include Files *****/
#include "PmodGPIO.h"
#include "xparameters.h"
#include "xil_types.h"
#include "xstatus.h"
#include <stdio.h>

/* ----- */
/*     Definitions
/* ----- */

void PmodSSD_Begin();
void display_PmodSSD(int numero);
void display_single_number(int numero, int digitos);

#endif // PmodSSD_H

```