UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Implementación de un sistema de alarma sobre un sistema particionado basado en hipervisor.

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Ortega Ayuso, Clara

Tutor/a: Guasque Ortega, Ana

CURSO ACADÉMICO: 2022/2023

Resumen

Este trabajo consiste en el desarrollo de un sistema de alarma convencional sobre un sistema particionado basado en hipervisor. Los sistemas particionados permiten el desarrollo de varias aplicaciones independientes ejecutándose sobre uno o varios procesadores. Para lograr esta independencia, se requiere una capa de virtualización (hipervisor) que aísle temporal y espacialmente las aplicaciones independizando su desarrollo y ejecución. El funcionamiento del sistema de alarma consiste en diferentes elementos de sensorización para la detección de movimiento. Todo ello se implementa sobre una placa de desarrollo Zybo Z7, que cuenta con dos núcleos ARM y una serie de conectores.

Palabras clave:

Sistema de alarma; Hipervisor; Lenguaje C; Diligent Zybo Z7; Sistema particionado; Aislamiento; Sensorización.

Resum

Aquest treball consisteix en el desenvolupament d'un sistema d'alarma convencional sobre un sistema particionat basat en hipervisor. Els sistemes particionats permeten el desenvolupament de diverses aplicacions independents executant-se sobre un o diversos processadors. Per a aconseguir aquesta independència, es requereix una capa de virtualització (hipervisor) que aïlle temporal i espacialment les aplicacions independitzant el seu desenvolupament i execució. El funcionament del sistema d'alarma consisteix en diferents elements de sensorització per a la detecció de moviment. Tot això s'implementa sobre una placa de desenvolupament Zybo Z7, que compta amb dos nuclis ARM i una sèrie de connectors.

Paraules clau:

Sistema d'alarma; Hipervisor; Llenguatge C; Diligent Zybo Z7; Sistema particionat, Aïllament; Sensorització.

Abstract

This work consists of the development of a conventional alarm system on a partitioned system based on hypervisor. Partitioned systems allow the development of several independent applications running on one or several cores. To achieve this independence, a virtualization layer (hypervisor) is required to temporally and spatially isolate the applications, making their development and execution independent. The operation of the alarm system consists of different sensorization elements for motion detection. All this is implemented on a Zybo Z7 development board, which has two ARM cores and several connectors.

Keywords:

Alarm system; Hypervisor; C language; Diligent Zybo Z7; Partitioned system; Isolation; Sensorization.

A Javi, Marc, Eme y Carlos por acompañarme todos estos años.

A Ana, por ayudarme y guiarme en este proyecto.

Y a todos aquellos que me han apoyado.

Índice general

I Memoria

II Planos

III Pliego de condiciones

IV Presupuesto

V Bibliografía

VI Anexos

Índice de figuras

Índice de tablas

Listado de siglas empleadas

FPGA Field Programmable Gate Array.

GPIO General Purpose Input/Output.

JTAG Joint Test Action Group.

KYPD Keypad.

LCD Liquid Crystal Display.

LED Light-Emitting Diode.

MIO Management Input/Output.

ODS Objetivos de Desarrollo Sostenible.

PIR Passive Infrared Motion Sensor.

Pmod Peripheral module.

RGB Red Green Blue.

SSD Seven-Segment Display.

UART Universal Asynchronous Receiver-Transmitter.

USB Universal Serial Bus.

XADC Xilinx analog mixed signal module.

Parte I

Memoria

Capítulo 1

Introducción

1.1. Objeto

En los sistemas *software* actuales la complejidad es cada vez mayor debido a las nuevas funcionalidades que se han incorporado. Los sistemas de tiempo real [1] también han visto incrementada su complejidad, pero las restricciones de seguridad y fiabilidad han de mantenerse a fin de proporcionar las garantías de criticidad y fiabilidad.

A nivel de *hardware*, el uso de los sistemas multinúcleo [2] [3] está totalmente extendido, siendo las técnicas de virtualización [4] un elemento clave para su uso en sistemas altamente críticos. Compartir recursos entre aplicaciones puede dar lugar a dificultades e interferencias que podrían obstaculizar el diseño lógico y el rendimiento de la ejecución del sistema. Desde este punto de vista, la protección entre aplicaciones es necesaria. Esto dio lugar a los sistemas particionados, desarrollados para abordar problemas de seguridad y protección. Esta técnica permite ejecutar varios entornos de ejecución (particiones) independientes en una misma plataforma *hardware*.

La aplicación de técnicas de virtualización basadas en hipervisores [4] permite que diferentes aplicaciones compartan el mismo software sin interferencia, proporcionando aislamiento espacial, temporal y de fallos entre particiones. Las propiedades de aislamiento espacial y temporal de las arquitecturas particionadas son aspectos muy relevantes en los sistemas particionados.

Unos sistemas de tiempo real que se benefician gratamente de estas técnicas, son las alarmas. Estas, se hallan cada vez más presentes en nuestro entorno, puesto que nos ayudan a mantener nuestro bienestar y nos avisan en caso de peligros (incendios, inundaciones, robos, ayuda sanitaria, etc.).

Entre ellas, destacan las alarmas de detección de intrusos, debido a que la protección de viviendas siempre ha sido objeto de interés poblacional y por lo tanto, siempre se está en constante crecimiento.

Es por ello que el presente proyecto tiene como objeto el diseño y desarrollo de un sis-

tema de alarma utilizando técnicas de virtualización, las cuales proporcionarán una mejor protección tanto del sistema como de los hogares.

Para conseguirlo, compararemos los diversos tipos de alarmas que hay en el mercado, así como su popularidad y características. Del mismo modo, investigaremos sobre los sistemas particionados y los multinúcleos, así como el uso de hipervisores.

Con este fin, queremos implementar el sistema en una placa de desarrollo Zybo Z7 [5]. En ella, podremos conectar los periféricos necesarios para crear nuestro sistema y sobre la que podemos volcar el tipo de programa que buscamos.

1.2. Antecedentes. Estado del arte.

1.2.1. Alarma

Una alarma es un sistema de tiempo real diseñado para prevenir, responder e interceptar peligros. Se utilizan para toda clase de emergencias: médicas, climáticas, incendios, intrusiones, robos etc. y aunque funcionan de una forma muy parecida, cada una está constituida de elementos muy diferentes en función de lo que el usuario necesite.

Las alarmas de protección contra intrusos han estado siempre presentes entre nosotros, puesto que siempre ha sido un tema de interés. Al principio, se utilizaban perros para poder ser alertados en caso de presencia.

Sin embargo, no fue hasta el siglo XVIII que se crearía la primera alarma, la cual consistía en un mecanismo de campanas conectado al cerrojo de una puerta. Cuando alguien trataba de abrirla, estas sonaban.

Más adelante en 1853, Augustus Pope, creó la primera alarma electromagnética [6]. El sistema estaba compuesto por un circuito eléctrico, imanes y campanas. Cuando una puerta o ventana se abría, el circuito se cerraba, provocando que una de las campanas sonase. Esto sentó las bases de las alarmas de detección de intrusos modernas, pues consiguió que la alarma continuase sonando aunque el intruso consiguiese entrar y cerrar la puerta o ventana.

Actualmente, la tecnología se ha desarrollado lo suficiente como para que el mismo usuario pueda observar desde su propio móvil el estado de su casa mediante cámaras, pueda tener sensores no solo en las puertas y ventanas, si no también en el mismo interior de la vivienda, así como, en los alrededores de la misma. Estas pueden integrarse junto con otros dispositivos de seguridad como detectores de humo, detectores de gas, etc. y hasta pueden enviar un aviso a la policía en casa de ser activada. Además, estas alarmas pueden ser inalámbricas, haciendo que su instalación sea mucho más simple y permitiendo que se agreguen más sensores.

1.2.2. Sistema de tiempo real y virtualización

Como se menciona en [1] y [2], unas de las características más importantes de los sistemas de tiempo real es su respuesta respecto a los estímulos externos mediante sensores o periféricos, la arquitectura de su sistema tanto a nivel de *hardware* como de *software*, así como los componentes de los mismos. Muchos sistemas de tiempo real son sistemas de control: aviónica, control de procesos, servicios de supervisión, robótica, etc.

La forma en la que se diseña estos sistemas es muy importante, pues los procesos que los forman suelen tener parámetros muy distintos en cuanto a criticidad, plazo o tiempo de ejecución. Es por ello que usualmente se organizan teniendo en cuenta estas características, ya que el propio sistema debe ser capaz de tratarlos en caso de que ocurran simultáneamente.

Sin embargo, en muchas situaciones esto no ha sido suficiente para asegurar su funcionamiento, siendo en algunos casos que no es posible su organización. Por ello, se pueden generar diversos problemas como son la coexistencia de tareas con niveles de criticidad diferentes sin ningún tipo de distinción de prioridad entre ellas, las averías en el sistema por fallo de una única tarea y las diferentes necesidades de hardware de cada una de las tareas [4].

Frente a estos problemas se planteó el uso de sistemas distribuidos [4], donde la separación de las tareas en mayor y menor criticidad se realizaba físicamente, dedicando dos *software* distintos a cada grupo de ellas. Sin embargo, esta solución es muy costosa ya que se necesita un *hardware* distinto para cada grupo de tareas.

Por esa misma razón, se desarrollaron varios métodos para mejorar su planificación y control, entre los cuales se encuentran la virtualización. Gracias a ella, se nos permite la división de un mismo *hardware* en diferentes particiones, creando un sistemas particionado [4][2] con el que podemos separar los diversos conjuntos de tareas según su criticidad mediante un hipervisor [4][3] el cual permite su aislamiento del resto. Las propiedades de aislamiento espacial y temporal [7] de las arquitecturas particionadas son aspectos muy relevantes en los sistemas particionados. El aislamiento espacial implica que cada aplicación sólo puede acceder a sus direcciones de memoria independientes.

En mononúcleo, el aislamiento temporal significa que sólo una aplicación en un momento dado tiene acceso a los recursos del sistema, por lo que no es posible que una aplicación se ejecute cuando otra aplicación ya se está ejecutando. Esto requiere el uso de eficientes técnicas de planificabilidad.

La tecnología de máquina virtual puede considerarse la forma más segura y eficiente para construir sistemas particionados [8]. Utiliza técnicas de virtualización para conseguir el deseado aislamiento temporal, espacial y de fallos antes mencionado. Un hipervisor o monitor de máquina virtual (VMM) es una capa de software (o una mezcla de *hardware* y capa de *software*) que ejecuta varias particiones en un único ordenador. La principal diferencia entre los hipervisores y otro tipo de virtualización es el el rendimiento. Las principales características de los hipervisores son una baja sobrecarga y un rendimiento

similar al que tendría si se ejecutara en el sistema nativo.

La virtualización también nos permite la incorporación de sistemas multinúcleo. Dichos sistemas van muy ligados a la determinación de las prioridades, puesto que nos permiten la ejecución de dos o más tareas, dependiendo de la cantidad de núcleos del sistema, de manera simultánea, permitiendo que la organización del programa sea más sencilla y eliminando una parte de la secuencialidad de los sistemas.

En la aviónica modular integrada [2], en concreto en las naves espaciales, se puede observar como afectó la utilización de estos métodos. Al inicio, permitía a las naves tener varias funcionalidades como el GPS o la radio, teniendo cada uno su propio *hardware*. Sin embargo, en la actualidad, esta ofrece muchas más funcionalidades por lo que si se continuara desarrollando de la misma forma, encarecería el producto en gran medida. Es por ello, que se usa la virtualización, ya que con un único *hardware*, se pueden tratar y organizar las múltiples tareas que componen el sistema.

Capítulo 2

Estudio de necesidades, factores a considerar: limitaciones y condicionantes

Para poder llevar a cabo este proyecto, debemos tener en cuenta, a parte de aquello que necesitamos, diversos factores que nuestra alarma debe cumplir.

Primero, necesitamos que nuestra alarma sea capaz de interactuar con el entorno y sepa responder a él con eficacia. En consecuencia, necesitamos un sistema que nos permita conectar los elementos oportunos para completarlo. Dichos componentes deben ser capaces de detectar la presencia indeseada, permitir la comunicación con el usuario para que conozca el estado de la alarma y permita configurarla a su gusto. Además, debe ser capaz de alertar al intruso para disuadirlo de entrar a la vivienda.

A continuación, debemos tener en cuenta el bienestar del usuario, que depende del correcto funcionamiento de nuestro sistema. Es por ello, que necesitamos que, en caso de fallo, la respuesta esté controlada, evitando que se propague por el resto del sistema. Esto implica que es necesario algún tipo de mecanismo que nos permita aislar los fallos y posteriormente recuperar dicha parte para conocer la causa.

Con respecto a las limitaciones de nuestro proyecto, debemos considerar que se trata de un sistema de alarma bastante sencillo a nivel de *hardware*: tan solo cuenta con un sensor de movimiento, una botonera, un display y los diversos leds para mostrar el estado en el que se encuentra. Debido a ello es un sistema que solo se puede utilizar en entradas de habitaciones o viviendas pues carece de los elementos necesarios para proteger en su totalidad una casa o edificio.

La siguiente limitación, vendría dada por la misma placa en la que buscamos implementarlo. Esto se debe, a que la Zybo Z7 tiene un número limitado de entradas para usar periféricos y estos conforman una gran parte del sistema, por lo que nos impide crear una alarma que pueda detectar intrusos con mayor precisión.

En cuanto al condicionamiento del proyecto, debemos tener en cuenta que se trata de

una alarma de uso general. Por lo tanto, lo que realmente afecta a la forma del sistema de alarma no es el lugar en el que se vaya a utilizar, si no el *hardware* en el que se implementa.

Al estar formado por una única placa, debemos de adaptar nuestro sistema utilizando alguna herramienta o método que nos permita añadir todo lo que hemos mencionado anteriormente sin que nos suponga un sobrecoste muy elevado y podamos implementarlo sin problemas.

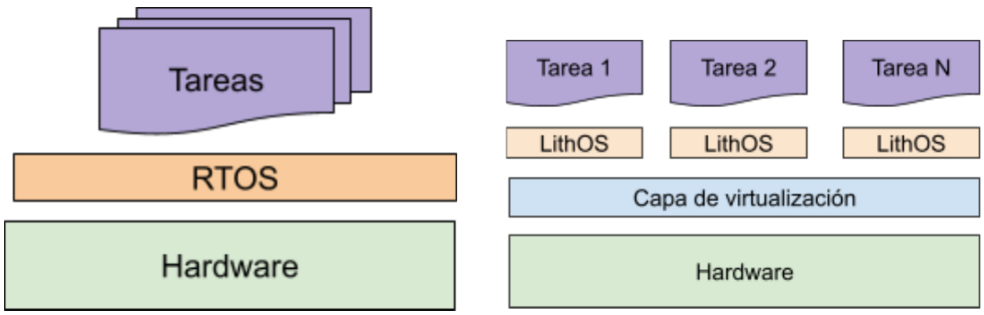
Capítulo 3

Planteamiento de soluciones alternativas y justificación de la solución adoptada

Teniendo en cuenta el apartado anterior, se han planteado las diversas soluciones que podríamos haber adoptado para desarrollar nuestro sistema. A la hora de entender la solución que vamos a escoger, debemos considerar el sistema que queremos, el tipo de alarma que necesitamos y los elementos que deseamos utilizar.

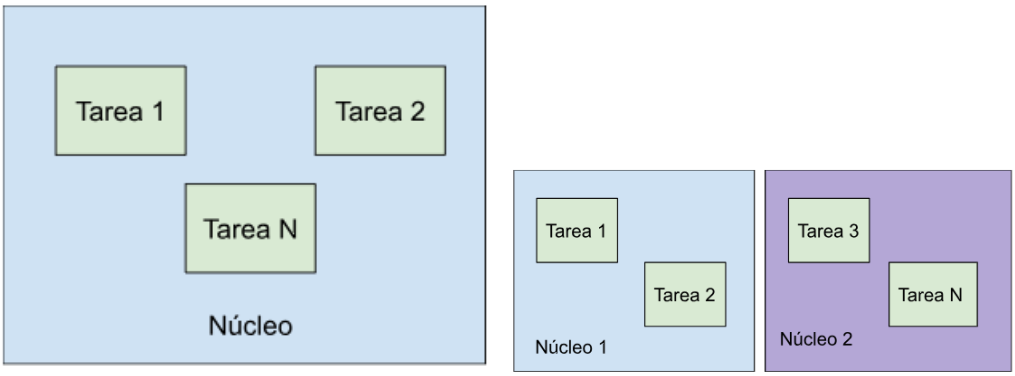
Primero elegiremos el tipo de sistema. Por un lado, tenemos los sistemas empotrados, donde, como podemos ver en la figura 3.1a, el funcionamiento de la aplicación depende del hardware al completo. Por otro lado, tenemos los sistemas particionados, con los que podemos separar las diversas tareas en particiones diferentes aislándose entre ellas tal y como se ve en la figura 3.1b. Esto evitaría que en caso de fallo, este no se propagara al resto de particiones, añadiendo un grado de complejidad en la implementación [2][3][9].

Esta última opción es la que nos resulta más interesante, debido a que buscamos que



(a) Sistema empotrado **(b) Sistema particionado**

Figura 3.1: Arquitecturas de un sistema empotrado y de un sistema particionado.



(a) Sistema mononúcleo **(b) Sistema multinúcleo**

Figura 3.2: Asignación de tareas en sistemas mononúcleo y multinúcleo

nuestro sistema sea fiable, proporcionando que continúe su funcionamiento a pesar de los fallos que pueda experimentar, además de ser capaces de conocer la causa.

A continuación, debemos decidir si queremos un sistema mononúcleo o multinúcleo. La diferencia entre ellos es que en las mononúcleo, como se ve en la figura 3.2a, todas las tareas son gestionadas por un solo núcleo, lo que provoca cierta secuencialidad en el sistema y provoca una respuesta más lenta. En cambio en los sistemas multinúcleo, como se puede observar en la figura 3.2b, las tareas se reparten entre varios núcleos, lo que genera una respuesta más rápida.

Nosotros buscamos un sistema que tenga una respuesta rápida frente a los estímulos del entorno, por lo que nos interesaría tener más de un núcleo.

Nuestro siguiente paso es elegir el tipo de alarma. Estas pueden clasificarse según el área a controlar, la función, el tipo de aviso y la instalación. Asimismo, según su complejidad estas adoptan un grado del 1 al 4, según el tipo de riesgo [10][11].

Primero seleccionaremos el área a controlar. Las alarmas pueden focalizarse en una única zona o en varias. En nuestro caso, buscamos proteger una única habitación o entrada, por lo que usaremos una alarma del primer tipo.

A continuación, debemos especificar la naturaleza de nuestra alarma. Esta nos puede alertar de: presencia de intrusos, incendios, inundaciones de agua o alarmas de emergencia para personas dependientes, como pueden ser personas mayores.

Como hemos mencionado, buscamos desalentar a los intrusos. En consecuencia, usaremos una alarma que nos alerte de su presencia. Para hacerlo, necesitaremos de algún aviso que indique que el sistema está en funcionamiento. Para ello se pueden hacer uso tanto de señales visuales (una luz), como acústicas (un pitido) o informativas (envía mensaje a una central que lo transmite a los propietarios). Teniendo en cuenta nuestras limitaciones, hemos hecho uso de avisos visuales y de la simulación de una llamada a las autoridades.

Otro factor a tener en cuenta, es la instalación de nuestra alarma. Pueden funcionar de

la siguiente manera:

- Mediante cableado: Lo cual supone un mayor coste y al mismo tiempo una mayor durabilidad y seguridad.
- Inalámbricas: lo que evita la realización de obras y permite la utilización de más sensores. Sin embargo, se pueden neutralizar utilizando inhibidores de frecuencia.
- Por monitorización: Permiten observar a tiempo real la zona controlada y pueden contactar directamente con las autoridades o la empresa a cargo en caso de detectar una amenaza.

Como queremos proteger a nuestro sistema de cualquier interferencia, nuestra alarma funciona mediante cableado sin hacer uso de la monitorización ya que sería necesario añadir más elementos a nuestro sistema.

Por último, según el grado de protección [10][11] que busquemos nuestra alarma puede ser:

- Grado 1: Defienden de un riesgo bajo, son sistemas centrados en la disuasión y que por lo tanto no protegen. No está conectada a ninguna central por lo que recae sobre el usuario la responsabilidad de dar el aviso.
- Grado 2: Para proteger frente riesgo medios. Son las más utilizadas tanto en viviendas como en negocios. A diferencia de las de grado 1, estas sí que avisan a una central receptora en caso de riesgo.
- Grado 3: Ofrecen un nivel de protección medio-alto y se utilizan principalmente en espacios propensos a ser robados. Este tipo de alarmas tienen una doble conexión para que no puedan ser saboteadas mediante inhibidores.
- Grado 4: Este tipo de alarmas son las que proporcionan el mayor nivel de protección y están destinadas a zonas consideradas como críticas según la legislación propia del país. Como puede ser el caso de instalaciones gubernamentales o militares, bancos o almacenes con elementos peligrosos.

Conociendo nuestras limitaciones y necesidades, hemos decidido desarrollar un sistema de alarma de grado 1. Esto se debe a que nuestro sistema solo se dedicaría a la detección de intrusos. Sin embargo, simula una llamada a las autoridades tras un determinado tiempo X sin insertar la clave, por lo que no se podría considerar de grado 2.

Una vez hemos elegido nuestro sistema y el tipo de alarma, debemos seleccionar el hardware sobre el que vamos a implementar el sistema particionado. Por lo cual, necesitamos un placa de desarrollo en la que a parte de realizar lo ya mencionado, se le puedan agregar los periféricos necesarios para completar la alarma.

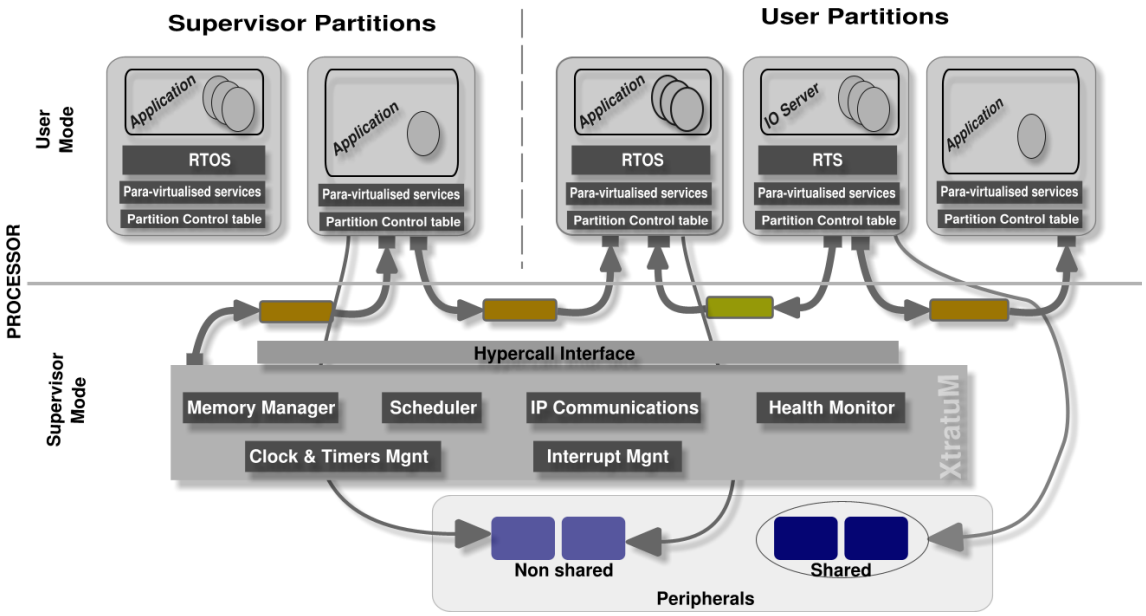


Figura 3.3: Arquitectura de Xtratum

La placa de desarrollo Zybo Z7 es una buena opción para llevar a cabo esta tarea. Ha sido desarrollada por la empresa Digilent y cuenta con una compatibilidad con unos periféricos de la misma empresa que podemos utilizar en nuestro proyecto. Dichos elementos son: un teclado numérico para permitir la entrada de datos por parte del usuario, una pantalla Liquid Crystal Display (LCD) para conocer la visualización de mensajes y un sensor de presencia para detectar movimiento.

Del mismo modo, debemos elegir de qué forma y con qué herramientas vamos a implementar nuestro sistema. Nosotros buscamos usar un hipervisor para dividir nuestro sistema en particiones mediante una capa de virtualización.

Xtratum [12][13][2] cumple este objetivo ya que se trata de un hipervisor diseñado para desarrollar sistemas de tiempo real que permite que varias aplicaciones funcionen en un mismo hardware sin interferencia. En la Figura 3.3 se representa un esquema general de la arquitectura de Xtratum. Xtratum ofrece una política de programación cíclica, a través de un plan temporal estático que asigna tiempo a las particiones. Cada partición planifica internamente sus tareas siguiendo un algoritmo (por prioridades, etc.). Este planificador cíclico sigue la norma ARINC-653 [14] y garantiza un fuerte aislamiento temporal.

Por lo tanto, la solución escogida será la de un sistema de alarma focalizado en una sola zona, de grado 2 cuya función es la de detectar presencia. Esta alarma estará implementada sobre un sistema particionado basado en el hipervisor Xtratum en una placa de desarrollo Zybo Z7.

Capítulo 4

Descripción detallada de la solución adoptada

Nuestro siguente paso, será detallar cada uno de los elementos que interviene en el sistema, así como la razón por la que los escogimos.

4.1. Arquitectura del sistema

Los sistemas particionados, tal y como indica su nombre, están constituidos por diversas particiones que se ejecutan simultáneamente sobre el hardware mediante una capa de virtualización. Como se puede ver en la Figura 4.1, estos sistemas están formados por: el hardware, la capa de virtualización y las particiones creadas a partir de esta.

Una partición es un conjunto de aplicaciones que trabajan en un mismo espacio. Estas se hallan aisladas tanto en tiempo como en espacio. Esto implica que cada partición tiene una cantidad de memoria limitada que debe repartirse entre las diversas aplicaciones y que no se pueden ejecutar dos al mismo tiempo. Esto último tan sólo ocurre en sistemas

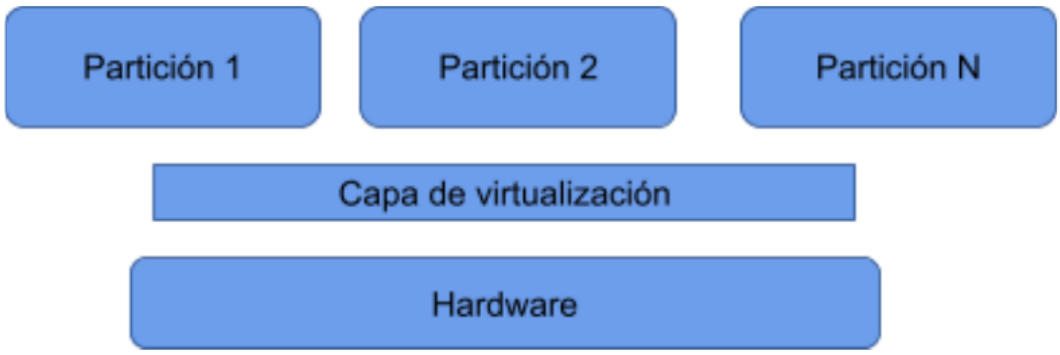


Figura 4.1: Arquitectura de un sistema particionado

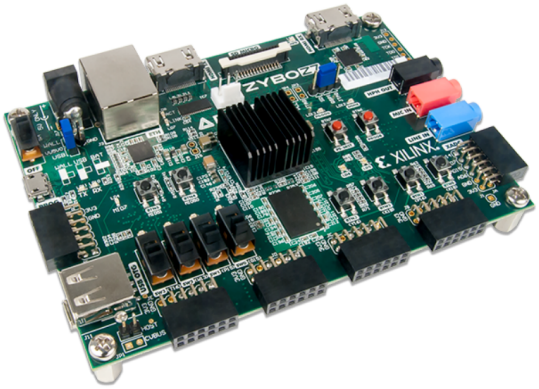


Figura 4.2: Plaza de desarrollo Zybo Z7.

mononúcleo, por lo que si este tuviese dos se podrían ejecutar ambas al mismo tiempo, funcionando cada una en un núcleo distinto.

Como hemos mencionado, se necesita de una capa de virtualización para crearla. Para lo cual, se hace uso de un hipervisor, o Virtual Machine Monitor, que es el término utilizado para referirse al software que lo realiza. Los hipervisores aíslan las diversas particiones en el tiempo y el espacio de la memoria y CPU. Esto permite protección en caso de fallo, así como, un aislamiento del mismo, generando una gran seguridad en el sistema. Se dividen en dos tipos: nativos y alojados [3].

Los hipervisores nativos, se ejecutan directamente sobre el hardware, haciendo el papel de un sistema operativo, y ofreciendo todo aquello que necesita el anfitrión. Por otro lado, los hipervisores alojados se ejecutan sobre el sistema operativo ya presente y dividen sus diversas funcionalidades.

En este caso, utilizaremos un hipervisor nativo, ya que estos tiene un mejor funcionamiento en sistemas de tiempo real críticos, debido a que permiten un mayor control y otorgan una mayor fiabilidad. En concreto, usaremos Xtratum, un hipervisor diseñado para funcionar directamente en el hardware y que permite la ejecución de diversos procesos sin interferencia.

4.2. Descripción del hardware

4.2.1. Zybo Z7

Para la realización de está alarma vamos a utilizar la placa de desarrollo Zybo Z7-10 (Figura 4.2) de la familia Xilix Zynq-7010 desarrollada por la empresa Digilent que, como todas las pertenecientes su familia, contiene un procesador de doble núcleo ARM Cortex-A9 y una Field Programmable Gate Array (FPGA) de la serie 7, también perteneciente a Xilinx [5].

Características destacables:

- Power Switch
- Power select jumper
- Universal Serial Bus (USB) de protocolo Joint Test Action Group(JTAG) /Universal Asynchronous Receiver-Transmitter (UART) port
- Peripheral Modules (Pmod) port de propósito general
- Management Input/Output (MIO) Pmod Port, al que se puede acceder directamente por programación.
- Xilinx analog mixed signal module (XADC) Pmod port, conexión directa a entradas/salidas digitales y analógicas.
- 4 Light-emitting diode (LED) monocromos y 1 LED RGB
- 4 interruptores y 4 pulsadores
- FPGA lógica programable

El desarrollo de la placa se realizará mediante el puerto multiplexado USB-JTAG/USB-UART. Siendo el primero para la programación y el segundo para la comunicación, perteneciendo al UART0.

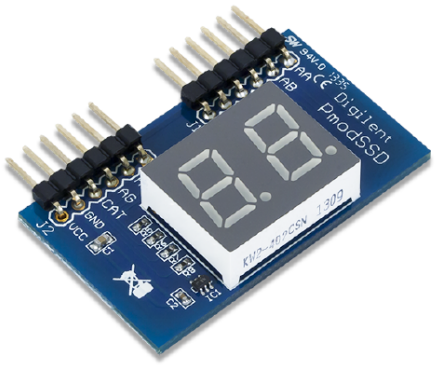
La comunicación con los dispositivos (LEDs, pulsadores, pmods, etc.) se realiza mediante la FPGA y por lo tanto, se debe programar.

Como hemos mencionado, la placa nos permite la conexión de hasta 5 periféricos para extender sus características. En nuestro caso, utilizaremos un total de 3.

4.2.2. PMOD DISPLAY DE SIETE SEGMENTOS (SSD)

Se trata de un display de dos dígitos de siete segmentos cuyo principal uso es el de mostrar contadores o temporizadores y que se comunica mediante General Purpose Input/Output (GPIO). Posee un pin para cada segmento, siendo estos compartidos entre los dos dígitos (Figura 4.3), lo que implica que no se pueden mostrar al mismo tiempo. Para poder mostrarlos simultáneamente, se deben alternar entre ellos con una frecuencia de 50 Hz o mayor [15].

El Pmod SSD nos permitirá conocer que la alarma ha sido activada y el tiempo restante para que la persona detectada se considere un intruso y se efectúe la llamada a las autoridades. Al ser un Pmod, está desarrollado específicamente para ser compatible con cualquier placa de Digilent.



PMOD SSD

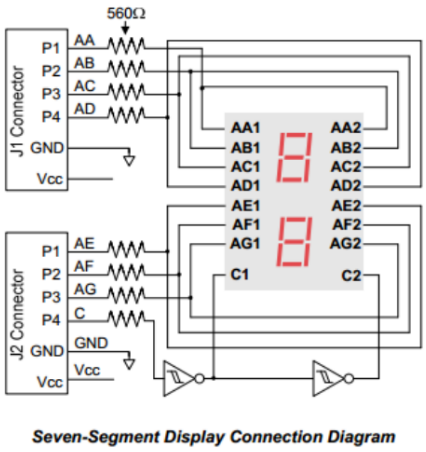


Figura 4.3: SSD

Figura 4.4: KYPD

4.2.3. PMOD TECLADO (KYPD)

Es un teclado matricial formado por 16 botones organizados de forma hexadecimal (Figura 4.4) y, que al igual que el display, se comunica mediante GPIO. Tanto sus filas como sus columnas están aisladas, permitiendo la pulsación simultánea de diferentes botones. Al mismo tiempo, esto provoca que para conocer qué tecla está siendo presionada, se deben leer las señales que emiten las filas y columnas, lo que puede dar lugar a error [16].

Un Pmod KYPD nos permitirá proveer una comunicación entre el sistema y el usuario, permitiendo el uso de una contraseña que añadirá una capa de protección al sistema, así como un manejo más cómodo de este, relegando el uso de los interruptores y pulsadores al reseteo de la alarma. Al igual que el SSD, es un Pmod por lo que la compatibilidad está asegurada.

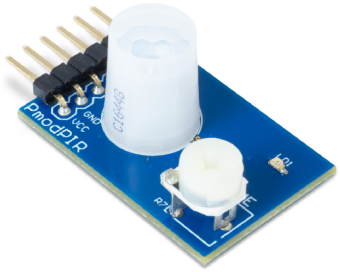


Figura 4.5: PIR

4.2.4. PMOD SENSOR INFRARROJOS PASIVO (PIR)

Sensor de Infrarrojos pasivo (PIR) de bajo consumo usado para la detección de movimiento hasta un total de 5 metros de distancia (Figura 4.5). Tiene una alta sensibilidad a movimientos y objetos pequeños, así como, una sensibilidad media para el caso contrario. Al igual que el resto de periféricos que vamos a utilizar, se comunica mediante GPIO [17].

Lo hemos escogido ya que, como hemos mencionado, nos permite detectar la presencia de personas. Esto permitirá que nuestro sistema esté completo y al igual que el resto de periféricos es compatible con nuestra placa.

4.3. Descripción del software

4.3.1. Lenguaje de desarrollo

Nuestro proyecto usa LithOS [18], un kernel en tiempo real para sistemas embebidos que sigue el estándar ARINC-653 APEX diseñado específicamente para Xtratum. Esto le permite ofrecer al sistema la posibilidad de tener diversas particiones, la comunicación entre ellas y la posibilidad de conocer su estado en todo momento. Todo esto también se ofrece en los procesos que funcionan dentro de dichas particiones, lo que enriquece en gran medida su valor.

Para poder implementarlo se necesita hacer uso del lenguaje de programación C puesto que es en el que se desarrolló.

LithOS no tiene la capacidad para crear las particiones, eso corresponde a Xtratum, el hipervisor. Sin embargo, LithOS puede pedir los servicios que necesita a Xtratum a través de la interfaz APEX que este mismo provee. Con esto se permite el control interno de las particiones dejando el aislamiento temporal y espacial a cargo del hipervisor.

En cuanto a los procesos, tanto su control como creación es completamente de LithOS ya que estos concurren dentro de cada una de las particiones que el sistema pueda tener.

Para que el sistema funcione correctamente se necesita de comunicación tanto entre las particiones, como entre los procesos dentro de cada una. Con ese fin, LithOS nos ofrece

```
<ProcessorTable>
 <Processor id="0" frequency="400Mhz">
  <CyclicPlanTable>
    <Plan id="0" majorFrame="1s">
      <Slot id="0" start="0ms" duration="1s" partitionId="0" flags="periodStart"/>
   </Plan>
  </CyclicPlanTable>
</Processor>
  <Processor id="1" frequency="400Mhz">
  <CyclicPlanTable>
    <Plan id="0" majorFrame="1s">
      <Slot id="0" start="0ms" duration="500ms" partitionId="1" flags="periodStart"/>
      <Slot id="1" start="500ms" duration="500ms" partitionId="2" flags="periodStart"/>
   </Plan>
  </CyclicPlanTable>
</Processor>
```

Figura 4.6: Particiones declaradas en cada uno de los núcleos de la placa

la posibilidad de usar:

- Sampling ports: lo que nos permite el envío de un único mensaje de partición a partición.
- Queuing ports: lo que nos permite el envío de varios mensajes mediante un buffer First In First Out (FIFO).

Para la comunicación entre procesos LithOS nos ofrece varios métodos:

- Blackboard: un mecanismo donde tan solo se puede escribir un único mensaje que se mantendrá hasta que este sea sobrescrito o borrado.
- Buffer: Un mecanismo similar al de una blackboard, pero con la diferencia de permitir el almacenaje de varios mensajes mediante una cola.

En nuestro caso solo ha sido necesario el uso de sampling ports y blackboards, ya que nuestro sistema no necesita colas para su funcionamiento.

Para diseñar nuestras particiones, tenemos un archivo de .xml llamado en nuestro caso "xm_cf.arm.xml". Se trata del fichero de configuración de XtratuM, donde determinamos las diferentes particiones, el espacio que utilizan, el tiempo en el que se ejecutan y el núcleo en el que funcionan, asegurando el aislamiento espacial y temporal.

Tal y como podemos ver en la figura 4.6, debemos declarar en cual de los dos núcleos de nuestra placa queremos que funcione y con que frecuencia. En nuestro caso, la partición 1 se encuentra sola para que la cuenta atrás no se vea afectada por el resto de tareas.

Por otro lado, en la figuras 4.7, 4.8 y 4.9, podemos observar como se declara el alcance de memoria de cada una de las particiones, declarando un área de la memoria en específico para cada una, así como las direcciones de los elementos que usa y los puertos de comunicación que utiliza.

Al final de este mismo archivo es donde se definen los canales de comunicación entre las diversas particiones que ya hemos mencionado. Estos son unidireccionales, por lo que

```
<PartitionTable>
    <Partition id="0" name="Partition0" flags="system boot fp" console="Uart">
        <PhysicalMemoryAreas>
            <Area start="0x10000000" size="256KB" />
        </PhysicalMemoryAreas>
       <PortTable>
        <Port name="Port1" type="sampling" direction="destination"/>
        <Port name="Port4" type="sampling" direction="destination"/>
       <Port name="Port7" type="sampling" direction="source"/>
       </PortTable>
        <HwResources>
         <IoPorts>
            <Range base="0x40000000" noPorts="32" />
            <Range base="0x40001000" noPorts="32" />
            <Range base="0x40002000" noPorts="32" />
            <Range base="0x41200000" noPorts="2" />
          <Range base="0xE000A040" noPorts="128"/>
         </IoPorts>
       </HwResources>
    </Partition>
```

Figura 4.7: Configuración partición 1

```
<Partition id="1" name="Partition1" flags="system boot fp" console="Uart">
  <PhysicalMemoryAreas>
    <Area start="0x14000000" size="256KB" />
  </PhysicalMemoryAreas>
 <PortTable>
 <Port name="Port2" type="sampling" direction="source"/>
 <Port name="Port5" type="sampling" direction="source"/>
 </PortTable>
  <HwResources>
    <IoPorts>
    <Range base="0x41210000" noPorts="4" />
    </IoPorts>
  </HwResources>
</Partition>
```

Figura 4.8: Configuración partición 2

```
    <Partition id="2" name="Partition2" flags="system boot fp" console="Uart">
 <PhysicalMemoryAreas>
    <Area start="0x16000000" size="256KB" />
 </PhysicalMemoryAreas>
<PortTable>
<Port name="Port3" type="sampling" direction="source"/>
<Port name="Port6" type="sampling" direction="destination"/>
<Port name="Port8" type="sampling" direction="destination"/>
 </PortTable>
  <HwResources>
    <IoPorts>
    <Range base="0x40003000" noPorts="32" />
    <Range base="0x41200008" noPorts="2" />
    </IoPorts>
 </HwResources>
</Partition>
```

Figura 4.9: Configuración partición 3

```
<SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
  <Source partitionId="1" portName="Port2"/>
  <Destination partitionId="0" portName="Port1"/>
</SamplingChannel>
<SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
  <Source partitionId="2" portName="Port3"/>
  <Destination partitionId="0" portName="Port4"/>
</SamplingChannel>
<SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
  <Source partitionId="1" portName="Port5"/>
  <Destination partitionId="2" portName="Port6"/>
</SamplingChannel>
    <SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
  <Source partitionId="0" portName="Port7"/>
  <Destination partitionId="2" portName="Port8"/>
</SamplingChannel>
```

Figura 4.10: Especificación sampling ports

como se puede observar en la figura 4.10, se debe especificar cual envía el mensaje y cual lo recibe.

4.4. Comunicación entre ordenador y placa de desarrollo

Para el desarrollo e implementación de este sistema se ha utilizado un ordenador que contiene Windows. Sin embargo, para realizar este proyecto nosotros necesitábamos Linux (Ubuntu 18.04) y por esa misma razón hemos hemos hecho uso de una máquina virtual con dicho sistema operativo. Para ello hemos usado VMWare Workstation Player, donde se nos permite tener una o más máquinas virtuales.

En cuanto a la forma en la que compilamos nuestro programa, contamos con un fichero Makefile. Este contiene las órdenes del comando *make* para crear y compilar las diversas particiones de nuestro proyecto. Cuando escribimos el comando *Make clean all* en la terminal, creamos todos los archivos que constituyen nuestro sistema.

Una vez hemos compilado nuestro programa, se creará el ejecutable llamado *resident_sw* y este se enviará a la placa para ponerla en funcionamiento mediante la conexión USB a través del comando xsdb, de Xilinx [19].

4.5. Implementación del software

En la figura 4.11 se puede observar el sistema que hemos diseñado, como se han repartido la diversas tareas entre las particiones y en que núcleo se ejecuta cada una.

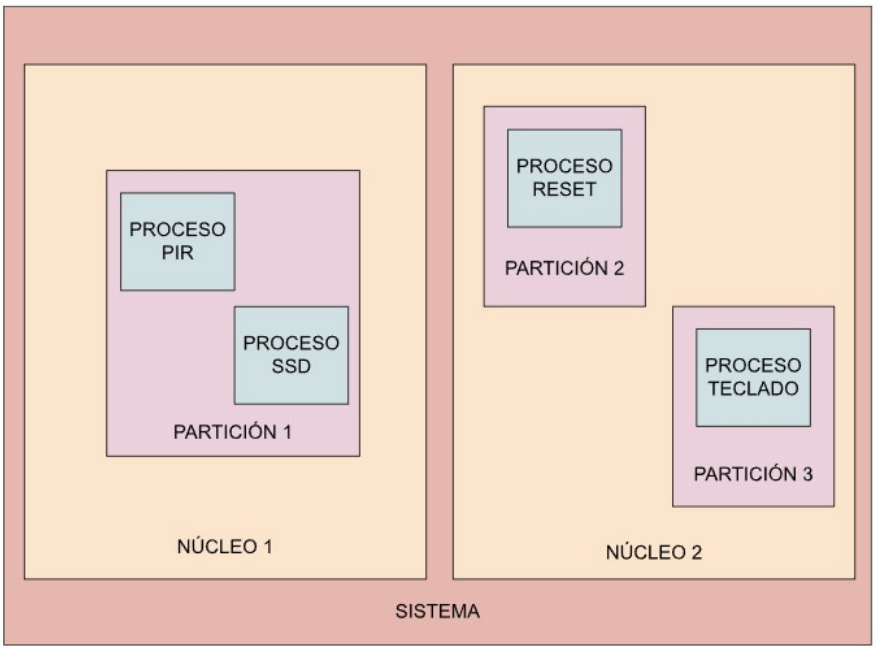


Figura 4.11: Esquema del sistema

El funcionamiento de nuestra alarma tal y como se puede observar en la figura 4.12, consta de dos modos: la puesta en marcha de la alarma, a la que se accede pulsando el botón 1, y el menú de configuración, al que se accede pulsando el botón 2.

Al iniciar el sistema, se encenderá el LED RGB en verde, como se ve en la figura 4.13 y se nos pedirá pulsar una de las teclas que hemos mencionado. Si presionamos otra, el sistema nos informará de que esta no hace nada.

Si pulsamos el botón 2, nos llevará al menú de configuración, encendiendo el LED RGB en azul como en la figura 4.14. Necesitaremos insertar la clave, si fallamos no pasa nada, pues no hay un número límite de intentos. Una vez introducida, se nos pedirá teclear la nueva clave y cuando la hayamos configurado, saldremos al menú principal.

Si se elige poner en marcha la alarma se encenderá el LED RGB en rojo como en la figura 4.15a y transmitirá mediante UART el mensaje “Esperando presencia” tal y como se ve en la figura 4.16, desde ese mismo instante, cuando el sensor de movimiento detecte presencia, se indicará encendiendo los 4 LEDs monocromos al mismo tiempo y transmitiendo el mensaje “Presencia!”, tras lo cual se pondrá en marcha el temporizador y se mostrará por el display la cuenta atrás como se muestra en la figura 4.15b.

Si se introduce la clave correcta, el temporizador se terminará y la luz roja se sustituirá por una verde. Sin embargo, si se termina el temporizador, la luz roja se mantendrá y simulará la llamada a las autoridades. Asimismo, si se falla un total de 4 veces en introducir el código, también se activará el protocolo de emergencia.

Para desactivar este protocolo, se debe introducir la secuencia de reseteo, la cual consiste en subir los cuatro switches y pulsar los cuatro botones al mismo tiempo.

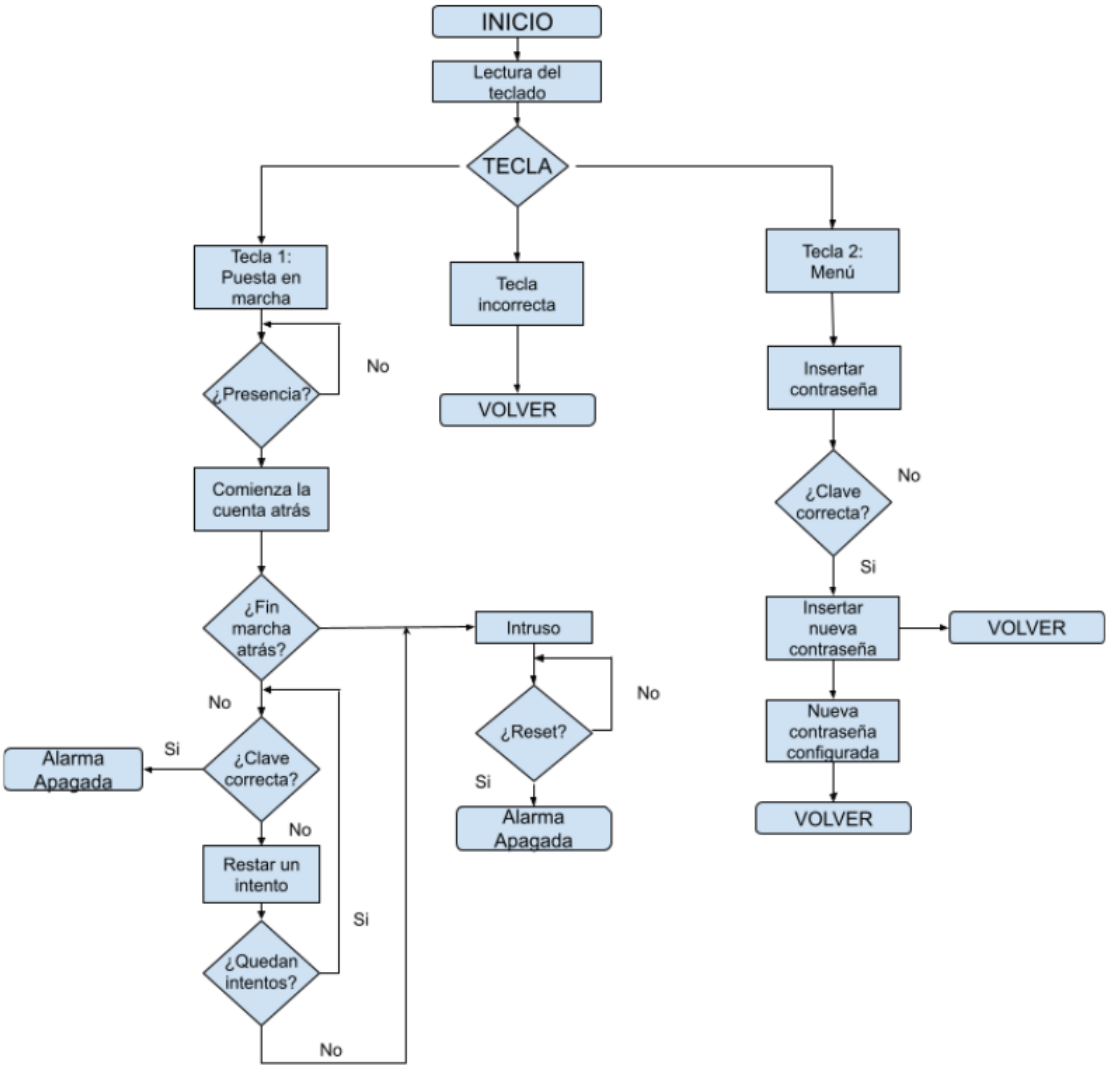


Figura 4.12: Diagrama de flujo

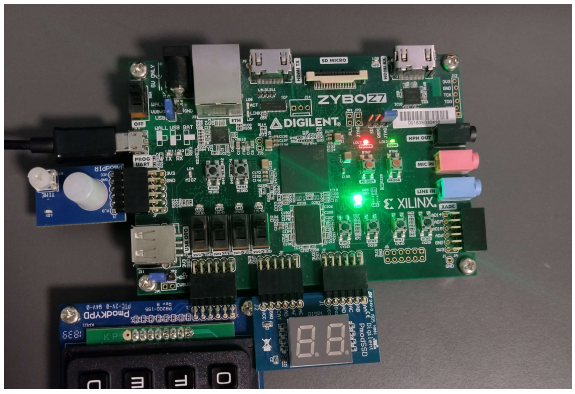


Figura 4.13: Alarma en funcionamiento

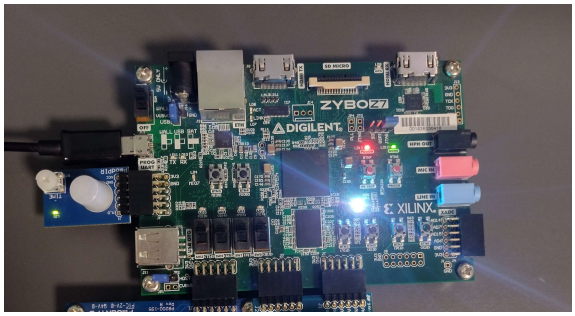
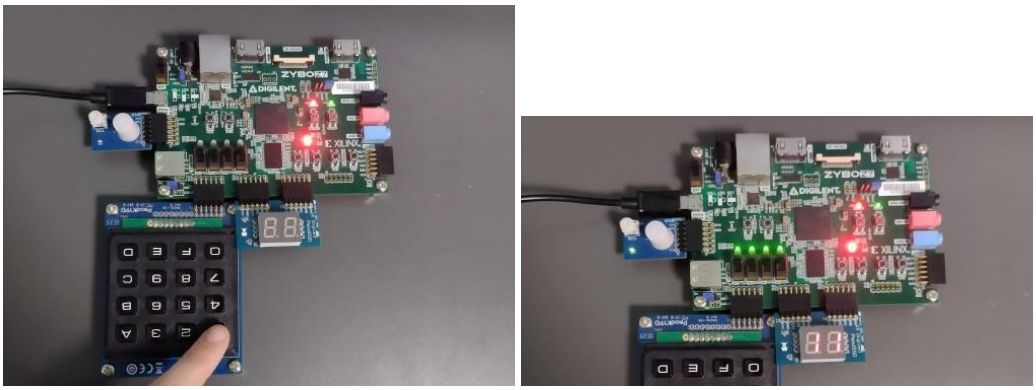


Figura 4.14: Menú de configuración



(a) Puesta en marcha **(b) Presencia detectada**

Figura 4.15: Estados alarma

```
[10:47:17:118] Esperando detectar algo
[10:47:17:166] Esperando detectar algo
[10:47:17:230] Esperando detectar algo
[10:47:17:294] Esperando detectar algo
[10:47:17:358] Esperando detectar algo
[10:47:17:406] Esperando detectar algo
[10:47:17:470] Esperando detectar algo
[10:47:17:534] Presencia!!!!!!!!!
[10:47:17:534] 0
[10:47:17:534] 20
[10:47:17:534] 19
```

Figura 4.16: Mensajes enviados por la placa

Capítulo 5

Resultados

5.1. Evaluación del funcionamiento de la alarma

En este capítulo se evaluará el correcto funcionamiento de la alarma. Para ello, realizaremos varias pruebas de servicio. Los códigos de estas pruebas se encuentran en los anexos.

1. En la primera prueba comprobaremos que la alarma se inicia correctamente. A continuación, entraremos en el menú de configuración y comprobaremos que podemos cambiar la clave correctamente. Para finalizar, confirmaremos que se puede volver al menú principal sin problema.

2. La segunda prueba consistirá en la puesta en marcha de la alarma. Pulsaremos la tecla adecuada, dando comienzo al sistema. Tras haberse puesto en marcha, comprobaremos que el sensor detecta nuestra presencia, muestra mediante el LCD la cuenta atrás para ingresar la clave y una vez que esta haya acabado, realizaremos un reset del sistema para comprobar que vuelve al menú y se puede volver a poner en marcha una vez más.

3. La última prueba servirá para determinar si la desactivación mediante clave funciona correctamente y que si introducimos una clave incorrecta más de 4 veces, el sistema pasa a considerarnos intrusos y debemos resetearlo.

5.2. Relación del trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2030

En la siguiente tabla se detalla la relación de este trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2030.

Objetivos de Desarrollo Sosteniles	Alto	Medio	Bajo	No procede
ODS1. Fin de la pobreza				X
ODS2. Hambre cero				X
ODS3. Salud y bienestar				X
ODS4. Educación de calidad				X
ODS 5. Igualdad de género				X
ODS 6. Agua limpia y saneamiento				X
ODS 7. Energía asequible y no contaminante				X
ODS 8. Trabajo decente y crecimiento económico				X
ODS 9. Industria, inovación e infraestructuras	X			
ODS 10. Reducción de las desigualdades				X
ODS 11. Ciudades y comunidades sostenibles				X
ODS 12. Producción y consumo responsables				X
ODS 13. Acción por el clima				X
ODS 14. Vida submarina				X
ODS 15. Vida de ecosistemas terrestres				X
ODS 16. Paz, justicia e instituciones sólidas		X		
ODS 17. Alianzas para lograr objetivos				X

Tabla 5.1: Objetivos de Desarrollo Sostenibles

El contenido de este proyecto corresponde con los sistemas de alarma, en concreto, se busca una mejora en la fiabilidad de los mismos sin la necesidad de añadir más hardware, lo cual se relaciona con el Objetivo de Desarrollo Sostenible (ODS) 9, en concreto con el 9.1 (“Desarrollar infraestructuras fiables, sostenibles, resilientes y de calidad”).

Nuestra alarma se trata de una de detección de intrusos, focalizada en viviendas, por lo que también se alinea con los objetivos del ODS 16, especialmente con el 16.1 (“Reducir significativamente todas las formas de violencia y las correspondientes tasas de mortalidad en todo el mundo”) y del 16.4 (“Fortalecer la recuperación y devolución de los activos robados y luchar contra todas las formas de delincuencia organizada”).

Capítulo 6

Conclusiones y valoración personal

Este proyecto tenía como objetivo el desarrollo e implementación de un sistema de alarma de detección de intrusos. Visto el resultado, podemos considerar que dicho objetivo ha sido cumplido.

Podríamos considerar que la parte más importante de este proyecto ha sido el aprendizaje sobre los métodos utilizados para desarrollarlos, ya que eran completamente nuevos. Esto ha sido posible gracias a los recursos proporcionados para estudiarlos y practicarlos antes de comenzar con el diseño y desarrollo.

Por otro lado, el uso del *hardware* no ha sido complicado, puesto que parte del mismo fue utilizado con anterioridad en una asignatura y por ello, ya estábamos familiarizados con él, lo que aligeró la implementación gratamente. Lo mismo sucede con el lenguaje de programación, al realizarse en C, era mucho más sencillo puesto que es el lenguaje que más hemos usado a lo largo del grado.

Este proyecto ha sido una una gran oportunidad para aprender a utilizar la placa Zybo con un objetivo y con un método de implementación diferente y nuevo. Esto ha permitido que la experiencia de desarrollarlo haya sido muy enriquecedora porque no solo he aplicado parte de los conocimientos obtenidos del grado de forma más directa, si no que además he adquirido nuevos.

6.1. Mejoras

A continuación, se hablará sobre las posibles mejoras que se podrían añadir en un futuro a nuestro proyecto. El sistema desarrollado es muy simple, por lo que en caso de querer mejorarlo se podría hacer lo siguiente:

- Añadir comunicación mediante Bluetooth o Internet con el usuario o las autoridades, para poder realizar un aviso verdadero en caso de activación de la alarma.
- Mejora física de la placa para poder permitir la conexión de más periféricos que ayu-

den a su funcionamiento como puede ser una cámara que detecte capte la presencia de personas o una señal sonora.

- Añadir algún tipo de cobertura para que la alarma pase desapercibida por parte del intruso y para permitir su protección del mismo.
- Aumentar la configuración de la alarma para permitir cambios en la duración de la marcha atrás, cambiar la secuencia de reseteo a gusto.
- Añadir la posibilidad de programar la alarma para su puesta en marcha automática en una hora determinada por el usuario.

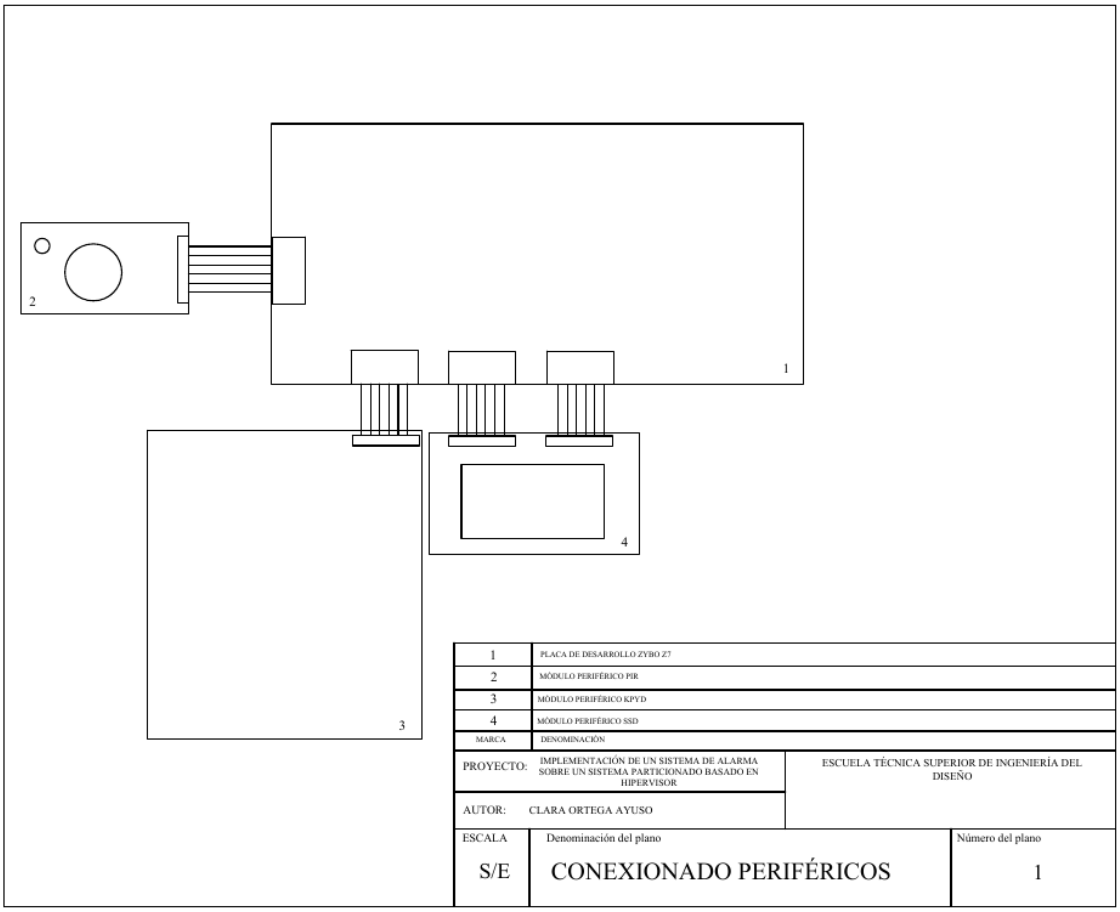
Parte II

Planos

Capítulo 7

Planos

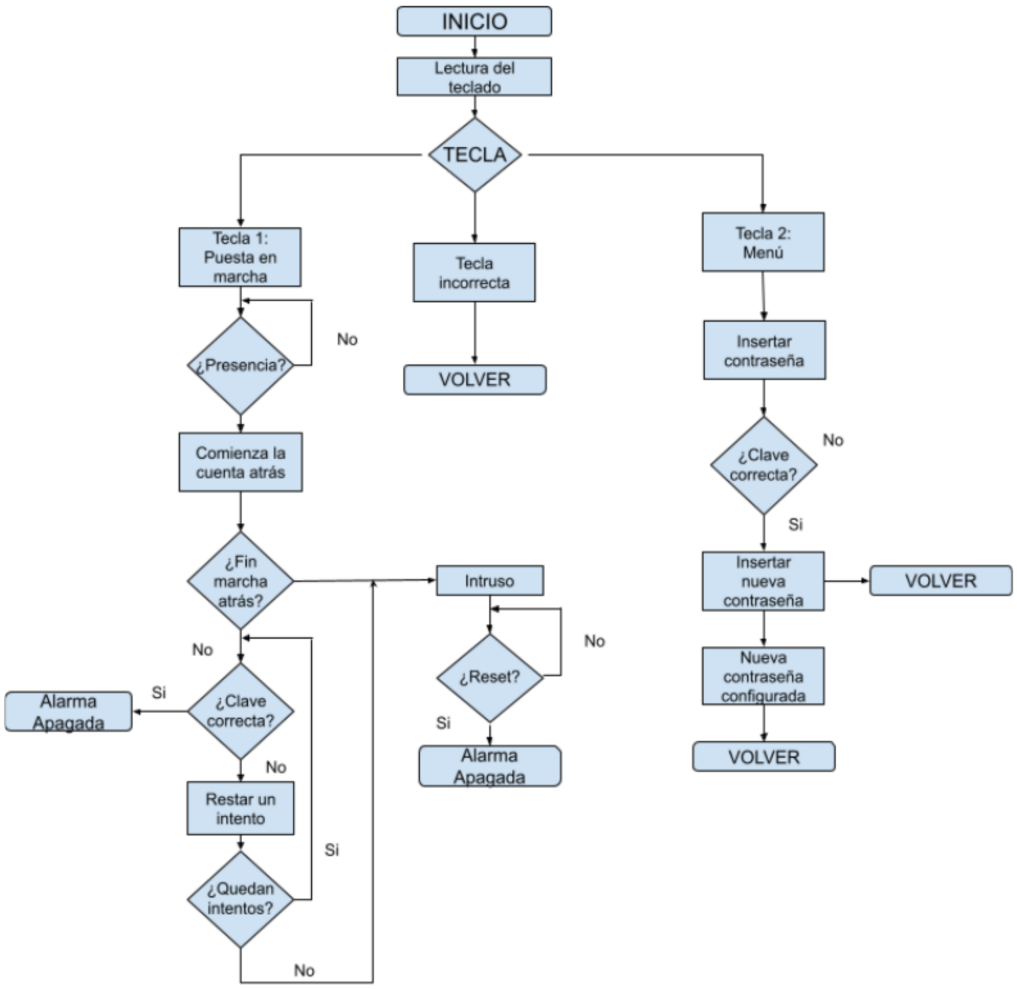
En este capítulo se incluye, por un lado, el esquema de conexiones entre la placa de desarrollo y los periféricos utilizados para el desarrollo del trabajo. Por otro lado, se muestra el diagrama de flujo del funcionamiento del sistema.



1	PLACA DE DESARROLLO ZYBO Z7
2	MÓDULO PERIFÉRICO PIR
3	MÓDULO PERIFÉRICO KPYD
4	MÓDULO PERIFÉRICO SSD
MARCA	DENOMINACIÓN

PROYECTO:	IMPLEMENTACIÓN DE UN SISTEMA DE ALARMA SOBRE UN SISTEMA PARTICIONADO BASADO EN HIPERVISOR	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
AUTOR:	CLARA ORTEGA AYUSO	

ESCALA	Denominación del plano	Número del plano
S/E	CONEXIONADO PERIFÉRICOS	1



PROYECTO: IMPLEMENTACIÓN DE UN SISTEMA DE ALARMA SOBRE UN SISTEMA PARTICIONADO BASADO EN HIPERVISOR		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
AUTOR: CLARA ORTEGA AYUSO		
ESCALA S/E	Denominación del plano DIAGRAMA DE FLUJO	Número del plano 2

Parte III

Pliego de condiciones

Capítulo 8

Pliego de condiciones

8.1. Objeto

La presente especificación técnica se refiere al diseño, desarrollo e implementación de un sistema de alarma para la detección y disuasión de intrusos en entradas a viviendas o habitaciones de uso habitual.

8.2. Condiciones de los materiales

8.2.1. Descripción

1. Placa Zybo Z7 Zynq-7010
    - Power Switch
    - Power select jumper
    - USB JTAG/UART port
    - MIO Pmod Port, se puede acceder directamente por programación.
    - Pmod port de propósito general
    - XADC Pmod port, conexión directa a entradas/salidas digitales y analógicas.
    - 4 LEDs monocromos y 1 LED RGB
    - 4 interruptores y 4 pulsadores
    - FPGA lógica programable
    - 630 KB de RAM
    - Procesador dual-core ARM Cortex-A9
    - Tamaño 8,8 x 12,2 cm

2. Periféricos:

   a) Pmod KPYD

   - 16 botones
   - Detección simultánea de botones
   - Columnas y filas aisladas
   - 12 pines Pmod con interfaz GPIO
   - Tamaño 8,64 x 6,86 cm

   b) Pmod SSD

   - Display de 7 segmentos con 2 dígitos
   - Fácil visualización de temporizadores y contadores
   - Configuración cátodo común
   - 2 conectores Pmod con 6 pines e interfaz GPIO
   - Tamaño 2,54 x 4,31 cm

   c) Pmod PIR

   - Bajo consumo
   - Alta sensibilidad para movimientos y objetos pequeños
   - Media sensibilidad para movimientos y objetos grandes
   - Detección hasta a 5 metros de distancia.
   - Conector Pmod con 6 pines e interfaz GPIO
   - Tamaño 2,032 x 3,048 cm

8.2.2. Control de calidad

Todos los elementos de este sistema deben realizar un control de funcionamiento por medio de testeo por multímetro y una prueba de uso, especificado en el reglamento electrotécnico de baja tensión.

Todos ellos siguen el reglamento UE 1025/2012 y vienen con marcado CE en función al Reglamento 765/2008, la Directiva 2014/30/UE, la Decisión 768/2008 y ISO 9001. Asimismo, siguen la directiva 2002/95/CE mediante la cual se corrobora que no se contienen sustancias peligrosas.

8.3. Condiciones de la ejecución

8.3.1. Descripción del proceso de ejecución

1. Placa Zybo Z7: Se colocará sobre una superficie plana que permita una correcta conexión de los periféricos y que no dificulte su funcionamiento.

2. Pmod KPYD: Se colocará en uno de los puertos Pmod de propósito general para poder comunicarse con la placa.
3. Pmod SSD: Se conectará a la placa mediante dos puertos Pmod de propósito general.
4. Pmod PIR: Se conectará en el puerto Pmod MIO para una mejor comunicación con la placa.

8.4. Control o controles a realizar

1. Placa Zybo Z7: Conexión al ordenador con el que se programa y del que se alimenta, comprobación de una comunicación UART con programa simple.
2. Pmod KPYD: Conexión a la placa y prueba de lectura de las teclas.
3. Pmod SSD: Conexión a la placa y prueba de funcionamiento con una pequeña cuenta regresiva.
4. Pmod PIR: Conexión a la placa y prueba de detección de movimiento.

8.5. Manual de usuario, pruebas y ajustes finales o de servicio

Para poder ser instalado, el sistema de alarma será entregado una vez se hayan superado las pruebas mencionadas en la memoria. La instalación de este proyecto incluye la alarma, junto con todos los componentes que lo conforman y el manual necesarios para la correcta puesta en marcha del mismo.

Para la utilización de la alarma se requiere:

- La entrada a una vivienda o habitación.
- Una superficie plana y libre de obstáculos.
- Prueba del menú de configuración.
- Prueba de la puesta en marcha de la alarma.
- Prueba de desactivación mediante clave.
- Prueba de reseteo del sistema.

Una vez se realizan las medidas de prueba y se asigna al supervisor competente del producto, se pasa a su integración. Su uso se comprueba mediante la detección de presencia desde lejos y la comprobación de falsas detecciones.

- Se pone en marcha la alarma y se sitúa un sujeto a 5 metros de distancia.
- Se comprueba la detección del mismo en caso de movimiento.
- Se desactiva la alarma y se vuelve al inicio del sistema.

Parte IV

Presupuesto

Capítulo 9

Presupuesto

El presupuesto se muestra dividido según su naturaleza en los materiales y la mano de obra.

9.1. Materiales

A continuación encontraremos tanto los materiales físicos, como es la placa o los periféricos, como materiales intangibles, como son las licencias utilizadas para el desarrollo del proyecto.

Como podemos observar en la Tabla 1.1, no aparece el precio del ordenador utilizado. Esto se debe a que hemos considerado que el precio tanto del ordenador, como de cualquier herramienta que podamos haber utilizado puntualmente, se ve reflejado en los medios auxiliares.

El software utilizado para desarrollar este proyecto ha sido: VMware Workstation 17 Player, el hipervisor Xtratum y el paquete de programas Microsoft Office 365. La versión de VMware que hemos utilizado es gratuita, sin embargo, en caso de querer utilizarla con fines comerciales deberíamos de comprar una. En cuanto a Xtratum, la versión empleada tiene una GNU General Public License (GNU GPL, Licencia Pública General de GNU en español) En el caso del paquete Office, su suscripción anual cuesta un total de 57,99 €. Sin embargo, al formar parte del alumnado de la UPV, esta nos proporciona una licencia, por lo que el coste por nuestra parte es de 0 €.

Ref	Ud	Descripción	Precio	Cantidad	Total
Materiales					
m1	ud	Plaza Zybo Z7-10	362.42 €	1	362.42€
m2	ud	Pmod KYPD	19.18 €	1	19.18 €
m3	ud	Pmod PIR	14.65 €	1	14.65 €
m4	ud	Pmod SSD	6.57 €	1	6.57 €
Medios Auxiliares					
	%	Medios auxiliares sobre costes directos	10 %	402,82 €	40,28 €
TOTAL PRESUPUESTO MATERIALES					**443,10 €**

Tabla 9.1: Coste de los materiales. Fuente: Propia

9.2. Mano de obra

Considerando únicamente la participación del autor, se estima un total de 360 h en la elaboración de este proyecto y se tiene en cuenta que el salario medio de un ingeniero electrónico es de 20 €/h. El coste del ingeniero desarrollador se ha extraído del salario medio de los ingenieros industriales según las estadísticas de Jobted. Disponible en `https://www.jobted.es/salario/ingeniero-industrial` o en páginas de colegios oficiales de ingenieros a veces dan esta información. A continuación, se muestra en la Tabla 9.2 un desglose de las tareas realizadas por el mismo, así como el tiempo dedicado a cada una.

9.3. Costes totales

En la Tabla 9.3 se muestra el coste total del proyecto, que incluye tanto los costes de materiales como el coste humano.

Ref	Ud	Descripción	Precio	Cantidad	Total
Mano de obra					
h1	h	Documentación e investigación	20 €	75	1.500 €
h2	h	Aprendizaje de utilización de software	20 €	70	1.400 €
h3	h	Diseño del sistema	20 €	30	600 €
h4	h	Implementación del sistema	20 €	80	1600 €
h5	h	Validación y pruebas de funcionamiento	20 €	7	300 €
h6	h	Redacción de los documentos	20 €	93	1800 €
Medios Auxiliares					
	%	Medios auxiliares sobre costes directos	10 %	7.200€	720 €
TOTAL PRESUPUESTO COSTES HUMANOS					**7.920 €**

Tabla 9.2: Coste humano. Fuente: Propia

Tipo de coste	Precios
Coste de los materiales	443,10 €
Coste humano	4098,60 €
TOTAL COSTES	**4541,70 €**

Tabla 9.3: Coste total. Fuente: Propia

Parte V

Bibliografía

Bibliografía

[1] Alfons Crespo y Alejandro Alonso. "Una Panorámica de los Sistemas de Tiempo Real". En: *Revista Iberoamericana de Automática e Informática industrial* 3.2 (sep. de 2010), págs. 7-18. URL: https://polipapers.upv.es/index.php/RIAI/article/view/8121.

[2] Tomás Terrizzano. *Control de la ejecución de sistemas particionados de criticidad mixta*. UPV, 2022. URL: http://hdl.handle.net/10251/179313.

[3] Ana Guasque. *Study, analysis and new scheduling proposals in partitioned real-time systems*. UPV, 2019. URL: 10.4995/Thesis/10251/135279.

[4] Alfons Crespo et al. "Sistemas de control con distintos niveles de criticidad". En: *Congreso Español de Informática*. Sep. de 2013, págs. 35-39.

[5] Diligent. Inc. (s.f.) *Zybo Z7*. https://digilent.com/reference/programmable-logic/zybo-z7/start.

[6] *Evolución de los sistemas de alarma: una breve historia de la seguridad doméstica moderna*. https://alarm.riscogroup.com/es/blog/evolucion-de-los-sistemas-de-alarma-una-breve-historia-de-la-seguridad-domestica-moderna.

[7] Alfons Crespo et al. "XtratuM: An Open Source Hypervisor for TSP Embedded Systems in Aerospace". En: (mayo de 2009), págs. 31-.

[8] Tomaso Poggi et al. "A Hypervisor Architecture for Low-Power Real-Time Embedded Systems". En: *2018 21st Euromicro Conference on Digital System Design (DSD)*. 2018, págs. 252-259. DOI: 10.1109/DSD.2018.00054.

[9] A. Crespo, I. Ripoll y M. Masmano. "Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach". En: *2010 European Dependable Computing Conference*. 2010, págs. 67-72. DOI: 10.1109/EDCC.2010.18.

[10] Mapfre S.L. (s.f.) *¿Qué tipos de alarmas hay?* https://www.mapfre.es/particulares/seguros-de-hogar/articulos/que-tipos-de-alarmas-hay/.

[11] Roams. *Grados de seguridad en alarmas y diferencias*. https://alarmas.roams.es/seguridad/grados-seguridad/. 2023.

[12] FentISS S.L. *Xtratum*. https://www.fentiss.com/xtratum/.

[13] M Masmano et al. “XtratuM: a Hypervisor for Safety Critical Embedded Systems.” En: *11th Real Time Linux Workshop*. Sep. de 2009, págs. 1-9.

[14] Miguel Masmano et al. “ARINC-653 APEX based on XtratuM”. En: 2011.

[15] Diligent. Inc. (s.f.) *Pmod SSD Reference Manual*. https://digilent.com/reference/pmod/pmodssd/reference-manual.

[16] Diligent. Inc. (s.f.) *Pmod KPYD*. https://digilent.com/reference/pmod/pmodkypd/start.

[17] Diligent. Inc. (s.f.) *Pmod PIR*. https://digilent.com/reference/pmod/pmodpir/start.

[18] FentISS S.L. *LithOS ARM Software User Manual (15-019.011.sum.01)*. https://www.fentiss.com/lithos/.

[19] *Xilinx*. https://www.xilinx.com/htmldocs/xilinx2019_1/SDK_Doc/SDK_concepts/concept_Xilinxsystemdebugger.html.

Parte VI

Anexos

Anexo J

Código fuente

J.1. Código de las particiones

J.1.1. Código de la partición 1

```
/*
 * $FILE: partition1.c
 */
#include <string.h>
#include <stdio.h>
#include <xm.h>
#include <apex_partition.h>
#include <apex_blackboard.h>
#include <apex_buffer.h>
#include <apex_sampling.h>

const struct xmImageHdr xmImageHdr __XMIHDR =
{
   .sSignature = XMEF_PARTITION_MAGIC,
   .compilationXmAbiVersion =
   XM_SET_VERSION(XM_ABI_VERSION, XM_ABI_SUBVERSION, XM_ABI_REVISION),
   .compilationXmApiVersion =
   XM_SET_VERSION(XM_API_VERSION, XM_API_SUBVERSION, XM_API_REVISION),
   .noCustomFiles = 0,
   .eSignature=XMEF_PARTITION_MAGIC,
};

#define GPIO_ADDRESS_JS1_DATA 0x40001000      //JC
#define GPIO_ADDRESS_JS1_CTR  0x40001004
#define GPIO_ADDRESS_JS2_DATA 0x40002000      //JD
#define GPIO_ADDRESS_JS2_CTR  0x40002004

#define GPIO_ADDRESS_MIO_IN  0xE000A060  // Input Data (GPIO Bank0, MIO)
#define GPIO_ADDRESS_MIO_OUT  0xE000A040  // Input Data (GPIO Bank0, MIO)
```

```
30 #define GPIO_ADDRESS_MIO_CTR 0xE000A204  // Direction mode (GPIO Bank0, MIO) 0: input  1:
       output
31
32 #define GPIO_ADDRESS_LED_DATA 0x41200000
33 #define GPIO_ADDRESS_LED_CTR  0x41200004
34
35 BLACKBOARD_NAME_TYPE Blackboard;
36 BLACKBOARD_ID_TYPE activar;
37 BLACKBOARD_ID_TYPE pantalla;
38 BLACKBOARD_ID_TYPE contador;
39 BLACKBOARD_ID_TYPE para;
40 SAMPLING_PORT_ID_TYPE port1,port4,port7;
41 int estado;
42
43 PROCESS_ATTRIBUTE_TYPE proc1;
44  PROCESS_ID_TYPE pid1;
45  PROCESS_ATTRIBUTE_TYPE proc2;
46  PROCESS_ID_TYPE pid2;
47  PROCESS_ATTRIBUTE_TYPE proc4;
48  PROCESS_ID_TYPE pid4;
49  RETURN_CODE_TYPE ret;
50
51
52 int encender_leds(int led){
53    xm_s32_t retCode;
54    retCode = XM_arm_outport(GPIO_ADDRESS_LED_DATA,  led);
55    return retCode;
56 }
57 int leer_pir(void){
58    xm_s32_t retCode;
59    int dato;
60    retCode = XM_arm_inport(GPIO_ADDRESS_MIO_IN,  &dato);
61    return dato;
62 }
63 void encender_7s_2(int led7sValue){
64  xm_s32_t retCode;
65
66
67    int d1,d2, valor1, valor2;
68
69    dosDigitos(led7sValue, &valor1, &valor2);
70    split(valor2, &d1, &d2);
71
72    d1 = d1 | 0x08;
73    retCode = XM_arm_outport(GPIO_ADDRESS_JS1_DATA, d2);
74      if (retCode < 0) printf("Error LCD2_1\n");
75    retCode = XM_arm_outport(GPIO_ADDRESS_JS2_DATA, d1);
76      if (retCode < 0) printf("Error LCD2_2\n");
77 }
78
79 int undigito(int v) {
80  int res;
81
```

```
82  res = 0x40;
83  if (v == 0) res = 0x3f;
84  else if (v == 1) res = 0x06;
85  else if (v == 2) res = 0x5b;
86  else if (v == 3) res = 0x4f;
87  else if (v == 4) res = 0x66;
88  else if (v == 5) res = 0x6d;
89  else if (v == 6) res = 0x7d;
90  else if (v == 7) res = 0x07;
91  else if (v == 8) res = 0x7f;
92  else if (v == 9) res = 0x6f;
93
94  return res;
95  }
96
97 void dosDigitos(int v, int *v1, int *v2) {
98  int unidades, decenas;
99
100 decenas = undigito(v / 10);
101 unidades = undigito(v  % 10);
102 *v1 = unidades;
103 *v2 = decenas;
104 }
105
106 void split(int v, int *v1, int *v2) {
107
108 *v1 = (v & 0xf0) / 16;
109 *v2 = (v & 0xF);
110 //printf("0x %x 0x %x 0x %x \n", v, *v1, *v2);
111 }
112    xm_s32_t retCode;
113
114 void encender_7s_1(int led7sValue){
115  xm_s32_t retCode;
116
117
118    int d1,d2, valor1, valor2;
119
120    dosDigitos(led7sValue, &valor1, &valor2);
121    split(valor1, &d1, &d2);
122
123    retCode = XM_arm_outport(GPIO_ADDRESS_JS1_DATA, d2);
124      if (retCode < 0) printf("Error LCD1_1\n");
125    retCode = XM_arm_outport(GPIO_ADDRESS_JS2_DATA, d1);
126      if (retCode < 0) printf("Error LCD1_2\n");
127
128 }
129
130
131
132 void init_ports(){
133    XM_arm_outport(GPIO_ADDRESS_LED_CTR, 0x00);
134    XM_arm_outport(GPIO_ADDRESS_LED_DATA,  0x00);
```

```
135
136   retCode = XM_arm_outport(GPIO_ADDRESS_JS1_CTR, 0x00);
137   if (retCode < 0) printf("Error 1\n");
138   retCode = XM_arm_outport(GPIO_ADDRESS_JS2_CTR, 0x00);
139    if (retCode < 0) printf("Error 2\n");
140
141   retCode = XM_arm_outport(GPIO_ADDRESS_MIO_CTR, 0x00);
142    if (retCode < 0) printf("Error 3  %d\n",retCode);
143    printf("Inicializado \n");
144 }
145
146 int contar(void){
147   RETURN_CODE_TYPE ret;
148   MESSAGE_SIZE_TYPE longitud;
149   xm_s32_t retCode;
150   int i = 40;
151   int intruso = 0,contar = 0,inicio = 0,intento = 0;
152
153   while(1)
154   {
155
156       //leo si se ha parado la tarea en algún momento para resetearla una vez vuelva a comenzar
157    CLEAR_BLACKBOARD(para,&ret);
158    READ_BLACKBOARD(para,INFINITE_TIME_VALUE,&contar,&longitud,&ret);
159    //leo la blackboard para saber si se ha pulsado 1 en el menú principal
160    CLEAR_BLACKBOARD(activar,&ret);
161    READ_BLACKBOARD(activar,INFINITE_TIME_VALUE,&intruso,&longitud,&ret);
162    printf(" %d\n", contar);
163
164
165    if(intruso == 1 &&  contar == 0) //si pulsa 1 y el contador no está activo, entonces lo activa y resetea el
         valor de iniciar el contador.
166    {
167
168      contar = 1;
169      i = 40;
170      intruso = 0;
171    }
172
173    while(contar == 1)
174    {
175      //comienza a contar y le pasa el valor de i al lcd
176      printf(" %d\n",i);
177
178      CLEAR_BLACKBOARD(pantalla,&ret);
179      DISPLAY_BLACKBOARD(pantalla,&i,sizeof(int),&ret);//con esto enviamos el valor de i al ssd
180      READ_BLACKBOARD(para,INFINITE_TIME_VALUE,&intento,&longitud,&ret); //si se supera el
           número máximo de intentos de introducir la clave se pasa directamente al fin de la cuenta
181
182      if(i != 0)
183      {
184        i--;
185        if(intento == 1)
```

```
186       {
187         printf("Intruso, llamando a la policia...\n");
188         contar = 2;
189       }
190     }
191     else
192     {
193       printf("Intruso, llamando a la policia...\n");
194       contar = 2; //si la marcha atrás llega a 0 se pone la alarma en solo para resetear
195     }
196
197
198
199     PERIODIC_WAIT(&ret);
200   }
201
202   while(contar == 2)
203   {
204     CLEAR_BLACKBOARD(contador,&ret);
205     READ_BLACKBOARD(contador,INFINITE_TIME_VALUE,&inicio,&longitud,&ret);//con esto se
        lee si se ha reseteado la alarma si Inicio = 1.
206     printf("reinicio contador: %d\n",inicio); //cuando salga esto en 1 es que se ha reiniciado
207     if(inicio == 1)
208     {
209       contar = 0; //se deja de contar
210       intruso = 0;
211       intento = 0;
212     }
213     PERIODIC_WAIT(&ret);
214
215   }
216  PERIODIC_WAIT(&ret);
217  }
218
219
220  return;
221
222 }
223
224 int LCD(void)
225 {
226  RETURN_CODE_TYPE ret;
227  MESSAGE_SIZE_TYPE longitud;
228  int dig;
229  while(1)
230  {
231    READ_BLACKBOARD(pantalla,INFINITE_TIME_VALUE,&dig,&longitud,&ret);
232    encender_7s_1(dig);
233    PERIODIC_WAIT(&ret);
234    encender_7s_2(dig);
235    PERIODIC_WAIT(&ret);
236  }
237
```

```
238 }
239
240 int detectar(void){
241  RETURN_CODE_TYPE ret, retport4;
242  int pir = 0;
243  int presencia = 0;
244  int recibo = 0,i = 0, clave = 0,espera = 0;
245  int reset = 0;
246  MESSAGE_SIZE_TYPE longitud;
247  VALIDITY_TYPE validez;
248
249  while(1)
250  {
251   READ_SAMPLING_MESSAGE(port4,&estado,&longitud,&validez,&retport4); //se lee el valor de
       iniciar la deteccion
252   if ((retport4 == NO_ERROR)&&(validez==longitud))
253    printf("Estado  %d\n",estado);
254
255
256   while(estado == 1 && reset == 0)
257   {
258     START(pid1, &ret);
259     clave = 0;
260     DISPLAY_BLACKBOARD(para,&clave,sizeof(int),&ret);
261     printf("Esperando detectar algo\n");
262     pir = leer_pir();
263     if(pir == 65025) //este es el valor del pir si se detecta
264     {
265      START(pid1, &ret);
266      WRITE_SAMPLING_MESSAGE(port7, &pir, sizeof(int),&ret);
267      printf("Presencia!!!!!!!!!\n");
268      encender_leds(15);
269      presencia = 1;
270      DISPLAY_BLACKBOARD(activar,&presencia,sizeof(int),&ret); //este es el valor que se manda al
       proceso contar para comenzar la marcha atrás
271      presencia = 0;  //se borra e valor de presencia
272      espera = 0;
273      estado = 2; //se pasa al siguiente estado
274     }
275
276    PERIODIC_WAIT(&ret);
277
278   }
279   while(estado == 2)
280   {
281    //recibo = 0;
282    READ_SAMPLING_MESSAGE(port1,&reset,&longitud,&validez,&ret); //se recibe el reseteo
283    READ_SAMPLING_MESSAGE(port4,&recibo,&longitud,&validez,&ret); //se recibe la clave de
       desactivación
284
285    if(reset == 1)
286    {
287     printf("reset\n");
```

```
288       if(recibo == 3)
289       {
290         printf("entro\n");
291         encender_leds(0);
292         i = 1;
293         DISPLAY_BLACKBOARD(contador,&i,sizeof(int),&ret); //la alarma se ha reseteado
294         recibo = 0;
295         estado = 0;
296         reset = 0;
297       }
298
299     }
300     else if(recibo == 2)
301     {
302       STOP(pid1,&ret);
303       encender_leds(0);
304       estado = 0;
305     }else if(recibo == 3)
306     {
307       i = 2;
308       DISPLAY_BLACKBOARD(para,&i,sizeof(int),&ret);
309     }
310     PERIODIC_WAIT(&ret);
311
312   }
313   PERIODIC_WAIT(&ret);
314 }
315 return;
316
317 }
318
319 void main(void)
320 {
321
322
323  //primero creamos la pizarra
324  CREATE_BLACKBOARD(Blackboard,sizeof(int),&activar,&ret);
325  if (ret==NO_ERROR)
326    printf("1rst blackboard created successfully\n");
327  else
328    printf("Error creating blackboard 1 (ret: %d)\n", ret);
329
330  CREATE_BLACKBOARD("Pantalla",sizeof(int),&pantalla,&ret);
331  if (ret==NO_ERROR)
332    printf("Pantalla created successfully\n");
333  else
334    printf("Error creating Pantalla (ret: %d)\n", ret);
335
336  CREATE_BLACKBOARD("Contador",sizeof(int),&contador,&ret);
337  if (ret==NO_ERROR)
338    printf("reset contador created successfully\n");
339  else
340    printf("Error creating reset contador (ret: %d)\n", ret);
```

```
341
342 CREATE_BLACKBOARD(”Para”,sizeof(int),&para,&ret);
343 if (ret==NO_ERROR)
344   printf(”para created successfully\n”);
345 else
346   printf(”Error creating para (ret: %d)\n”, ret);
347
348 CREATE_SAMPLING_PORT(”Port1”,32,DESTINATION,1000000000LL,&port1,&ret);
349
350 if (ret==NO_ERROR)
351   printf(”1rst port created successfully\n”);
352 else
353   printf(”Error creating port 1 (ret: %d)\n”, ret);
354
355 CREATE_SAMPLING_PORT(”Port4”,32,DESTINATION,1000000000LL,&port4,&ret);
356
357 if (ret==NO_ERROR)
358   printf(”4th port created successfully\n”);
359 else
360   printf(”Error creating port 4 (ret: %d)\n”, ret);
361
362 CREATE_SAMPLING_PORT(”Port7”,32,SOURCE,1000000000LL,&port7,&ret);
363
364 if (ret==NO_ERROR)
365   printf(”7th port created successfully\n”);
366 else
367   printf(”Error creating port 7 (ret: %d)\n”, ret);
368
369 //a continuación creamos los procesos
370
371 // proc1 data structure
372 memset(&proc1, 0, sizeof(PROCESS_ATTRIBUTE_TYPE));
373 strcpy(proc1.NAME, ”Contar”);
374 proc1.ENTRY_POINT=contar; //Name of the function that will execute the process
375 proc1.BASE_PRIORITY=MIN_PRIORITY_VALUE+2;
376 proc1.STACK_SIZE=8∗1024;
377 proc1.PERIOD=1000000000; //periodic process 1 s
378 proc1.TIME_CAPACITY=INFINITE_TIME_VALUE;
379
380 // proc2 data structure
381 memset(&proc2, 0, sizeof(PROCESS_ATTRIBUTE_TYPE));
382 strcpy(proc2.NAME, ”Detectar”);
383 proc2.ENTRY_POINT=detectar; //Name of the function that will execute the process
384 proc2.BASE_PRIORITY=MIN_PRIORITY_VALUE+2;
385 proc2.STACK_SIZE=8∗1024;
386 proc2.PERIOD=100000000; //periodic process 1 s
387 proc2.TIME_CAPACITY=INFINITE_TIME_VALUE;
388
389 memset(&proc4, 0, sizeof(PROCESS_ATTRIBUTE_TYPE));
390 strcpy(proc4.NAME, ”LCD”);
391 proc4.ENTRY_POINT=LCD; //Name of the function that will execute the process
392 proc4.BASE_PRIORITY=MIN_PRIORITY_VALUE+2;
393 proc4.STACK_SIZE=8∗1024;
```

```
394  proc4.PERIOD=10000000; //periodic process 1 s
395  proc4.TIME_CAPACITY=INFINITE_TIME_VALUE;
396
397  // creating the processes
398  CREATE_PROCESS(&proc1, &pid1, &ret);
399  if (ret==NO_ERROR)
400    printf("Contar created successfully\n");
401  else
402    printf("Error creating Contar(ret: %d)\n", ret);
403
404  CREATE_PROCESS(&proc2, &pid2, &ret);
405  if (ret==NO_ERROR)
406    printf("Detectar created successfully\n");
407  else
408    printf("Error creating Detectar (ret: %d)\n", ret);
409
410  CREATE_PROCESS(&proc4, &pid4, &ret);
411  if (ret==NO_ERROR)
412    printf("LCD created successfully\n");
413  else
414    printf("Error creating LCDs (ret: %d)\n", ret);
415
416  //starting the processes
417  START(pid1, &ret);
418  START(pid2, &ret);
419  START(pid4, &ret);
420
421  printf("System ready to execute the processes\n");
422
423
424  //Port initialization
425  init_ports();
426
427    SET_PARTITION_MODE (NORMAL, &ret);
428  return;
429
430 }
```

J.1.2. Código de la partición 2

```
1  /*
2  /*
3   * $FILE: partition2.c
4   *
5   */
6
7  #include <string.h>
8  #include <stdio.h>
9  #include <xm.h>
10 #include <apex_partition.h>
11 #include <apex_time.h>
```

```
12 #include <apex_sampling.h>
13
14 const struct xmImageHdr xmImageHdr __XMIHDR =
15 {
16    .sSignature = XMEF_PARTITION_MAGIC,
17    .compilationXmAbiVersion =
18    XM_SET_VERSION(XM_ABI_VERSION, XM_ABI_SUBVERSION, XM_ABI_REVISION),
19    .compilationXmApiVersion =
20    XM_SET_VERSION(XM_API_VERSION, XM_API_SUBVERSION, XM_API_REVISION),
21    .noCustomFiles = 0,
22    .eSignature=XMEF_PARTITION_MAGIC,
23 };
24
25 #define GPIO_ADDRESS_BTN_DATA 0x41210000
26 #define GPIO_ADDRESS_BTN_CTR  0x41210004
27 #define GPIO_ADDRESS_SWCH_DATA 0x41210008
28 #define GPIO_ADDRESS_SWCH_CTR  0x4121000C
29
30 SAMPLING_PORT_ID_TYPE port2,port5;
31
32 void init_ports(){
33    XM_arm_outport(GPIO_ADDRESS_SWCH_CTR, 0xFF);
34    XM_arm_outport(GPIO_ADDRESS_SWCH_DATA, 0x00);
35    XM_arm_outport(GPIO_ADDRESS_BTN_CTR, 0xFF);
36    XM_arm_outport(GPIO_ADDRESS_BTN_DATA, 0x00);
37 }
38
39
40
41 int leerSwitches(){
42    xm_s32_t retCode;
43    int dato;
44    retCode = XM_arm_inport(GPIO_ADDRESS_SWCH_DATA,  &dato);
45    return dato;
46 }
47
48 int leerButtons(){
49    xm_s32_t retCode;
50    int dato;
51    retCode = XM_arm_inport(GPIO_ADDRESS_BTN_DATA,  &dato);
52    return dato;
53 }
54
55
56 static void Reset(void) {
57    RETURN_CODE_TYPE ret;
58  int switches,buttons,suma,enviado = 0;
59  printf("Entro en Reset\n");
60
61  while(1)
62  {
63    //leer de los switches
64    switches = leerSwitches();
```

```
65    buttons = leerButtons();
66    suma = switches + buttons;
67    if(suma == 30)
68    {
69     printf("Reset\n");
70     enviado = 1;
71     WRITE_SAMPLING_MESSAGE(port2, &enviado, sizeof(int),&ret); //se envia al pir que se ha
         reseteado
72     WRITE_SAMPLING_MESSAGE(port5, &enviado, sizeof(int),&ret); //se envia al menú que se ha
         reseteado
73
74    }
75    else if(enviado == 1)
76    {
77     enviado = 0;
78     WRITE_SAMPLING_MESSAGE(port2, &enviado, sizeof(int),&ret); //se envia al pir que se ha
         terminado el reset
79     WRITE_SAMPLING_MESSAGE(port5, &enviado, sizeof(int),&ret); //se envia al menú que se ha
         terminado el reset
80    }
81    PERIODIC_WAIT(&ret);
82   }
83
84   return;
85  }
86
87  void main(void)
88  {
89   PROCESS_ATTRIBUTE_TYPE proc3;
90   PROCESS_ID_TYPE pid3;
91   RETURN_CODE_TYPE ret;
92   //SAMPLING_PORT_ID_TYPE port2;
93
94   CREATE_SAMPLING_PORT("Port2",32,SOURCE,1000000000LL,&port2,&ret);
95
96   if (ret==NO_ERROR)
97    printf("2nd port created successfully\n");
98   else
99    printf("Error creating port 2 (ret: %d)\n", ret);
100
101  CREATE_SAMPLING_PORT("Port5",32,SOURCE,1000000000LL,&port5,&ret);
102
103  if (ret==NO_ERROR)
104   printf("5th port created successfully\n");
105  else
106   printf("Error creating port 5 (ret: %d)\n", ret);
107  // proc3 data structure
108
109  memset(&proc3, 0, sizeof(PROCESS_ATTRIBUTE_TYPE));
110  strcpy(proc3.NAME, "Reset");
111  proc3.ENTRY_POINT=Reset; //Name of the function that will execute the process
112  proc3.BASE_PRIORITY=MIN_PRIORITY_VALUE+2;
113  proc3.STACK_SIZE=8*1024;
```

```
114  proc3.PERIOD=50000000; //periodic process 50 ms
115  proc3.TIME_CAPACITY=INFINITE_TIME_VALUE;
116
117  //Port initialization
118  init_ports();
119
120
121  // creating the processes
122
123  CREATE_PROCESS(&proc3, &pid3, &ret);
124  if (ret==NO_ERROR)
125    printf("Reset created successfully\n");
126  else
127    printf("Error creating reset (ret: %d)\n", ret);
128
129
130  //starting the processes
131  START(pid3, &ret);
132  printf("System ready to execute the processes\n");
133    SET_PARTITION_MODE (NORMAL, &ret);
134  return;
135 }
```

J.1.3. Código de la partición 3

```
1  /*
2   * $FILE: partition3.c
3   *
4   */
5
6  #include <string.h>
7  #include <stdio.h>
8  #include <xm.h>
9  #include <apex_partition.h>
10 #include <apex_time.h>
11 #include <apex_sampling.h>
12 #include <apex_blackboard.h>
13
14 const struct xmImageHdr xmImageHdr __XMIHDR =
15 {
16    .sSignature = XMEF_PARTITION_MAGIC,
17    .compilationXmAbiVersion =
18    XM_SET_VERSION(XM_ABI_VERSION, XM_ABI_SUBVERSION, XM_ABI_REVISION),
19    .compilationXmApiVersion =
20    XM_SET_VERSION(XM_API_VERSION, XM_API_SUBVERSION, XM_API_REVISION),
21    .noCustomFiles = 0,
22    .eSignature=XMEF_PARTITION_MAGIC,
23 };
24 #define GPIO_ADDRESS_JT 0x40003000
25 #define GPIO_ADDRESS_JT_CTR 0x40003004 //KPYD
26
```

```
27 #define GPIO_ADDRESS_RGB_DATA 0x41200008
28 #define GPIO_ADDRESS_RGB_CTR  0x4120000C
29
30 SAMPLING_PORT_ID_TYPE port3,port6,port8;
31
32 void init_ports(){
33    XM_arm_outport(GPIO_ADDRESS_RGB_CTR, 0x00);
34    XM_arm_outport(GPIO_ADDRESS_RGB_DATA,  0x00);
35    XM_arm_outport(GPIO_ADDRESS_JT_CTR,  0xF0);
36 }
37 int encender_rgb(int led){
38    xm_s32_t retCode;
39    retCode = XM_arm_outport(GPIO_ADDRESS_RGB_DATA,  led);
40    return retCode;
41 }
42
43 int leerTeclado(void){
44  xm_u32_t col,fila, retCode;
45      retCode = XM_arm_outport(GPIO_ADDRESS_JT_CTR, 0x0F);
46    retCode = XM_arm_outport(GPIO_ADDRESS_JT, 0xF0);
47    retCode = XM_arm_inport(GPIO_ADDRESS_JT, &col);
48    if (retCode < 0) printf("Error reading KYPD\n");
49     col = col & 0x0F;
50    retCode = XM_arm_outport(GPIO_ADDRESS_JT_CTR, 0xF0);
51    retCode = XM_arm_outport(GPIO_ADDRESS_JT, 0x00);
52    retCode = XM_arm_inport(GPIO_ADDRESS_JT, &fila);
53    fila = (~(fila >> 4)) & 0xF;
54   int ch;
55   if (col == 1) {
56    if (fila == 1) ch = 13;
57    if (fila == 2) ch = 12;
58    if (fila == 4) ch = 11;
59    if (fila == 8) ch = 10;
60   } else if (col == 2) {
61    if (fila == 1) ch = 14;
62    if (fila == 2) ch = 9;
63    if (fila == 4) ch = 6;
64    if (fila == 8) ch = 3;
65   } else if (col == 4) {
66    if (fila == 1) ch = 15;
67    if (fila == 2) ch = 8;
68    if (fila == 4) ch = 5;
69    if (fila == 8) ch = 2;
70   } else if (col == 8) {
71    if (fila == 1) ch = 0;
72    if (fila == 2) ch = 7;
73    if (fila == 4) ch = 4;
74    if (fila == 8) ch = 1;
75   } else ch = -1;
76   return ch;
77 }
78
79
```

```
80 static void Teclado(void) {
81   RETURN_CODE_TYPE ret;
82  int modo = 0,leo,reset = 0,falso = 0;
83  int clave1,clave2,clave3;
84  int cl1 = 1,cl2 = 2,cl3 = 3;
85  int cont = 0,clavenueva = 0;
86  int detener = 0,intento = 4;
87  MESSAGE_SIZE_TYPE longitud;
88  VALIDITY_TYPE validez;
89
90  printf("Comienzo el teclado\n");
91  while(1)
92  {
93
94     leo = leerTeclado();
95     if(leo == 1 && modo == 0) //se pulsa la tecla 1
96     {
97       reset = 0;
98             intento = 4;
99       printf("Puesta en marcha\n");
100      WRITE_SAMPLING_MESSAGE(port3, &leo, sizeof(int),&ret);//se envia la señal de comienzo al
         PIR
101      if (ret == NO_ERROR) printf("Envio bien port 3 (ret: %d)\n", ret);
102      encender_rgb(4);
103      modo = 1;
104    }
105    else if(leo == 2 && modo == 0) // se pulsa 2: configurar clave
106    {
107      printf("Menu de configuracion\n");
108      printf("Teclea la clave\n");
109      encender_rgb(15);
110      modo = 2;
111      PERIODIC_WAIT(&ret);
112    }
113    else if(leo != -1) //se presiona otra tecla distinta de 1 y 2
114    {
115      if(modo == 0)
116      {
117        printf("Tecla equivocada, pulse otra tecla\n");
118      }
119    }
120    else if(reset == 1)
121    {
122      falso = 3;
123      WRITE_SAMPLING_MESSAGE(port3, &falso, sizeof(int),&ret);
124      printf("Reinicio, pulse una tecla\n");
125      modo = 0;
126      reset = 0; //resetea el menú
127      encender_rgb(2);
128
129    }
130
131    if(modo == 0)
```

```
132     {
133      encender_rgb(2);
134     }
135     else if(modo == 1)
136     {
137      READ_SAMPLING_MESSAGE(port6,&reset,&longitud,&validez,&ret);
138      READ_SAMPLING_MESSAGE(port8,&detener,&longitud,&validez,&ret);
139     }else if(modo == 3)
140     {
141      READ_SAMPLING_MESSAGE(port6,&reset,&longitud,&validez,&ret);
142      falso = 2;
143      WRITE_SAMPLING_MESSAGE(port3, &falso, sizeof(int),&ret); //se le pide a la cuenta atrás que
         pare
144     }
145
146     if(modo == 2 || (modo == 1 && detener == 65025))
147     {
148
149      if(leo != -1)
150      {
151       if(cont == 0) //primer valor presionado
152       {
153        printf("valor 1: %d\n",leo);
154        clave1 = leo;
155        cont++;
156       }
157       else if(cont == 1) //segundo valor presionado
158       {
159        printf("valor 2: %d\n",leo);
160        clave2 = leo;
161        cont++;
162       }
163       else if(cont == 2)//tercer valor presionado
164       {
165        printf("valor 3: %d\n",leo);
166        clave3 = leo;
167        cont++;
168       }
169
170      }else
171      {
172       if(cont == 3)
173       {
174        if(modo == 2)
175        {
176         if(clave1 == cl1 && clave2 == cl2 && clave3 == cl3 && clavenueva == 0)
177         {
178          printf("clave: %d %d %d\n",clave1,clave2,clave3);
179          printf("clave correcta, teclee la nueva clave\n"); //clave corrrecta, introducir nueva
180          clavenueva = 1;
181          cont=0;
182
183         }
```

```
184          else if(clavenueva == 1)
185          {
186           cl1 = clave1;
187           cl2 = clave2;
188           cl3 = clave3;
189           printf("clave nueva:  %d %d %d\n",clave1,clave2,clave3); //nueva clave
190           cont=0;
191           clave1= clave2 = clave3 = 0;
192           clavenueva = 0;
193           printf("Saliste de configuracion\n"); //volver al menú principal
194           encender_rgb(2);
195           modo = 0;
196          }
197          else
198          {
199           printf("clave:  %d %d %d\n",clave1,clave2,clave3); //clave incorrecta
200           printf("clave incorrecta\n");
201           cont=0;
202          }
203         }
204         else if(modo == 1)
205         {
206          if(clave1 == cl1 && clave2 == cl2 && clave3 == cl3 && intento != 0)
207          {
208           printf("clave:  %d %d %d\n",clave1,clave2,clave3);
209           printf("clave correcta\n");
210           falso = 2;
211           WRITE_SAMPLING_MESSAGE(port3, &falso, sizeof(int),&ret); //se le pide a la cuenta atrás
    que pare
212           clave1= clave2 = clave3 = 0;
213           falso = 0;
214           modo = 0; //se vuelve al menu
215
216          }
217          else
218          {
219           if(intento == 0)
220           {
221            printf("clave incorrecta\n");
222            printf("Intentos agotados\n");
223            intento = 4;
224            modo = 3;
225           }
226           else
227           {
228            printf("clave incorrecta\n");
229            cont = 0;
230            intento--;
231           }
232
233
234          }
235         }
```

```
236         }
237
238       }
239     }
240   PERIODIC_WAIT(&ret);
241  }
242  return;
243 }
244
245 void main(void)
246 {
247  PROCESS_ATTRIBUTE_TYPE proc5;
248  PROCESS_ID_TYPE pid5;
249  RETURN_CODE_TYPE ret;
250
251
252  CREATE_SAMPLING_PORT("Port3",32,SOURCE,1000000000LL,&port3,&ret);
253
254  if (ret==NO_ERROR)
255   printf("3rd port created successfully\n");
256  else
257   printf("Error creating port 3 (ret: %d)\n", ret);
258
259  CREATE_SAMPLING_PORT("Port6",32,DESTINATION,1000000000LL,&port6,&ret);
260
261  if (ret==NO_ERROR)
262   printf("6th port created successfully\n");
263  else
264   printf("Error creating port 6 (ret: %d)\n", ret);
265
266  CREATE_SAMPLING_PORT("Port8",32,DESTINATION,1000000000LL,&port8,&ret);
267
268  if (ret==NO_ERROR)
269   printf("8th port created successfully\n");
270  else
271   printf("Error creating port 8 (ret: %d)\n", ret);
272
273  init_ports();
274  // proc1 data structure
275  memset(&proc5, 0, sizeof(PROCESS_ATTRIBUTE_TYPE));
276  strcpy(proc5.NAME, "Hello1");
277  proc5.ENTRY_POINT=Teclado; //Name of the function that will execute the process
278  proc5.BASE_PRIORITY=MIN_PRIORITY_VALUE+2;
279  proc5.STACK_SIZE=8*1024;
280  proc5.PERIOD=500000000; //periodic process 500 ms
281  proc5.TIME_CAPACITY=INFINITE_TIME_VALUE;
282
283
284
285  // creating the processes
286  CREATE_PROCESS(&proc5, &pid5, &ret);
287  if (ret==NO_ERROR)
288   printf("Teclao process created successfully\n");
```

```
289  else
290    printf("Error creating process teclao (ret: %d)\n", ret);
291
292
293  //starting the processes
294  START(pid5, &ret);
295
296
297  printf("System ready to execute the processes\n");
298    SET_PARTITION_MODE (NORMAL, &ret);
299  return;
300 }
```

J.2. Fichero de configuración de Xtratum

```
<SystemDescription xmlns="http://www.xtratum.org/xm-arm-2.x" version=
    "1.0.0" name="hello_world">
    <HwDescription>
        <MemoryLayout>
            <Region type="rom" start="0x0" size="1MB"/>
            <Region type="sdram" start="0x00100000" size="1023MB"/>
        </MemoryLayout>
        <ProcessorTable>
            <Processor id="0" frequency="400Mhz">
                <CyclicPlanTable>
                    <Plan id="0" majorFrame="1s">
                        <Slot id="0" start="0ms" duration="1s"
                            partitionId="0" flags="periodStart"/>
                    </Plan>
                </CyclicPlanTable>
            </Processor>
            <Processor id="1" frequency="400Mhz">
                <CyclicPlanTable>
                    <Plan id="0" majorFrame="1s">
                        <Slot id="0" start="0ms" duration="500ms"
                            partitionId="1" flags="periodStart"/>
                        <Slot id="1" start="500ms" duration="500ms"
                            partitionId="2" flags="periodStart"/>
                    </Plan>
                </CyclicPlanTable>
            </Processor>
        </ProcessorTable>
        <Devices>
            <Uart id="1" baudRate="115200" name="Uart"/>
        </Devices>
    </HwDescription>
    <XMHypervisor console="Uart">
        <PhysicalMemoryArea size="512KB"/>
    </XMHypervisor>
    <PartitionTable>
        <Partition id="0" name="Partition0" flags="system boot fp"
            console="Uart">
            <PhysicalMemoryAreas>
                <Area start="0x10000000" size="256KB"/>
            </PhysicalMemoryAreas>
            <PortTable>
                <Port name="Port1" type="sampling" direction="
```

```
            destination"/>
        <Port name="Port4" type="sampling" direction="
            destination"/>
        <Port name="Port7" type="sampling" direction="source"/>
    </PortTable>
    <HwResources>
        <IoPorts>
            <Range base="0x40000000" noPorts="32"/>
            <Range base="0x40001000" noPorts="32"/>
            <Range base="0x40002000" noPorts="32"/>
            <Range base="0x41200000" noPorts="2"/>
            <Range base="0xE000A040" noPorts="128"/>
        </IoPorts>
    </HwResources>
</Partition>
<Partition id="1" name="Partition1" flags="system boot fp"
    console="Uart">
    <PhysicalMemoryAreas>
        <Area start="0x14000000" size="256KB"/>
    </PhysicalMemoryAreas>
    <PortTable>
        <Port name="Port2" type="sampling" direction="source"/>
        <Port name="Port5" type="sampling" direction="source"/>
    </PortTable>
    <HwResources>
        <IoPorts>
            <Range base="0x41210000" noPorts="4"/>
        </IoPorts>
    </HwResources>
</Partition>
<Partition id="2" name="Partition2" flags="system boot fp"
    console="Uart">
    <PhysicalMemoryAreas>
        <Area start="0x16000000" size="256KB"/>
    </PhysicalMemoryAreas>
    <PortTable>
        <Port name="Port3" type="sampling" direction="source"/>
        <Port name="Port6" type="sampling" direction="
            destination"/>
        <Port name="Port8" type="sampling" direction="
            destination"/>
    </PortTable>
    <HwResources>
        <IoPorts>
```

```
                <Range base="0x40003000" noPorts="32"/>
                <Range base="0x41200008" noPorts="2"/>
            </IoPorts>
        </HwResources>
    </Partition>
</PartitionTable>
<Channels>
    <SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
        <Source partitionId="1" portName="Port2"/>
        <Destination partitionId="0" portName="Port1"/>
    </SamplingChannel>
    <SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
        <Source partitionId="2" portName="Port3"/>
        <Destination partitionId="0" portName="Port4"/>
    </SamplingChannel>
    <SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
        <Source partitionId="1" portName="Port5"/>
        <Destination partitionId="2" portName="Port6"/>
    </SamplingChannel>
    <SamplingChannel maxMessageLength="32B" refreshPeriod="1s">
        <Source partitionId="0" portName="Port7"/>
        <Destination partitionId="2" portName="Port8"/>
    </SamplingChannel>
</Channels>
</SystemDescription>
```

J.3. Fichero de configuración de LithOS para las particiones

J.3.1. Fichero de LithOS para la partición 1

```
#ifndef _LITHOS_CFG_H_
#define _LITHOS_CFG_H_

#define CONFIG_MAX_PROCESS_STACK_SIZE 8192
#define CONFIG_MAX_NO_PARTITIONS 1
#define CONFIG_MAX_NO_PROCESSES 3
#define CONFIG_MAX_NO_SAMPLING_PORTS 3
#define CONFIG_MAX_NO_QUEUING_PORTS 0
#define CONFIG_MAX_NO_BLACKBOARDS 4
#define CONFIG_MAX_NO_EVENTS 0
#define CONFIG_MAX_NO_BUFFERS 0
#define CONFIG_MAX_NO_MESSAGES 0
#define CONFIG_MAX_MESSAGE_SIZE 96
#define CONFIG_MAX_NO_SEMAPHORES 0
#define CONFIG_MAX_NO_IRQ_EVENTS 0
#define CONFIG_MAX_MESSAGE_POOL_SIZE 96
#define CONFIG_MAX_NO_LOGGED_HM_EVENTS 0
#define CONFIG_MAX_NO_MULTIPLE_SCHEDULES 1
#define CONFIG_PARTITION_PERIOD 200000LL
#define CONFIG_PARTITION_DURATION 200000LL

#define CONFIG_DFLT_HM_ACT_CORE ((AC_IDLE << EV_NUMERIC_ERROR) | \
                                 (AC_IDLE << EV_STACK_OVERFLOW) | \
                                 (AC_IDLE << EV_MEMORY_VIOLATION) | \
                                 (AC_IDLE << EV_HARDWARE_FAULT) | \
                                 (AC_IDLE << EV_INTERNAL_ERROR))

#define CONFIG_DFLT_HM_ACT_APP ((AC_IGNORE << EV_DEADLINE_MISSED) | \
                                (AC_IDLE << EV_APPLICATION_ERROR) | \
                                (AC_IDLE << EV_NUMERIC_ERROR) | \
                                (AC_IDLE << EV_STACK_OVERFLOW) | \
                                (AC_IDLE << EV_MEMORY_VIOLATION) | \
                                (AC_IDLE << EV_HARDWARE_FAULT))

#endif
```

J.3.2. Fichero de LithOS para la partición 2

```

#ifndef _LITHOS_CFG_H_
#define _LITHOS_CFG_H_

#define CONFIG_MAX_PROCESS_STACK_SIZE 8192
#define CONFIG_MAX_NO_PARTITIONS 1
```

```
#define CONFIG_MAX_NO_PROCESSES 2
#define CONFIG_MAX_NO_SAMPLING_PORTS 2
#define CONFIG_MAX_NO_QUEUING_PORTS 0
#define CONFIG_MAX_NO_BLACKBOARDS 0
#define CONFIG_MAX_NO_EVENTS 0
#define CONFIG_MAX_NO_BUFFERS 0
#define CONFIG_MAX_NO_MESSAGES 0
#define CONFIG_MAX_MESSAGE_SIZE 32
#define CONFIG_MAX_NO_SEMAPHORES 0
#define CONFIG_MAX_NO_IRQ_EVENTS 0
#define CONFIG_MAX_MESSAGE_POOL_SIZE 0
#define CONFIG_MAX_NO_LOGGED_HM_EVENTS 0
#define CONFIG_MAX_NO_MULTIPLE_SCHEDULES 1
#define CONFIG_PARTITION_PERIOD 200000LL
#define CONFIG_PARTITION_DURATION 200000LL

#define CONFIG_DFLT_HM_ACT_CORE ((AC_IDLE << EV_NUMERIC_ERROR) | \
                                 (AC_IDLE << EV_STACK_OVERFLOW) | \
                                 (AC_IDLE << EV_MEMORY_VIOLATION) | \
                                 (AC_IDLE << EV_HARDWARE_FAULT) | \
                                 (AC_IDLE << EV_INTERNAL_ERROR))

#define CONFIG_DFLT_HM_ACT_APP ((AC_IGNORE << EV_DEADLINE_MISSED) | \
                                (AC_IDLE << EV_APPLICATION_ERROR) | \
                                (AC_IDLE << EV_NUMERIC_ERROR) | \
                                (AC_IDLE << EV_STACK_OVERFLOW) | \
                                (AC_IDLE << EV_MEMORY_VIOLATION) | \
                                (AC_IDLE << EV_HARDWARE_FAULT))

#endif
```

J.3.3. Fichero de LithOS para la partición 3

```
#ifndef _LITHOS_CFG_H_
#define _LITHOS_CFG_H_

#define CONFIG_MAX_PROCESS_STACK_SIZE 8192
#define CONFIG_MAX_NO_PARTITIONS 1
#define CONFIG_MAX_NO_PROCESSES 1
#define CONFIG_MAX_NO_SAMPLING_PORTS 3
#define CONFIG_MAX_NO_QUEUING_PORTS 0
#define CONFIG_MAX_NO_BLACKBOARDS 0
#define CONFIG_MAX_NO_EVENTS 0
#define CONFIG_MAX_NO_BUFFERS 0
#define CONFIG_MAX_NO_MESSAGES 0
#define CONFIG_MAX_MESSAGE_SIZE 96
#define CONFIG_MAX_NO_SEMAPHORES 0
#define CONFIG_MAX_NO_IRQ_EVENTS 0
#define CONFIG_MAX_MESSAGE_POOL_SIZE 96
#define CONFIG_MAX_NO_LOGGED_HM_EVENTS 0
#define CONFIG_MAX_NO_MULTIPLE_SCHEDULES 1
```

```
19 #define CONFIG_PARTITION_PERIOD 200000LL
20 #define CONFIG_PARTITION_DURATION 200000LL
21
22 #define CONFIG_DFLT_HM_ACT_CORE ((AC_IDLE << EV_NUMERIC_ERROR) | \
23                                  (AC_IDLE << EV_STACK_OVERFLOW) | \
24                                  (AC_IDLE << EV_MEMORY_VIOLATION) | \
25                                  (AC_IDLE << EV_HARDWARE_FAULT) | \
26                                  (AC_IDLE << EV_INTERNAL_ERROR))
27
28 #define CONFIG_DFLT_HM_ACT_APP ((AC_IGNORE << EV_DEADLINE_MISSED) | \
29                                 (AC_IDLE << EV_APPLICATION_ERROR) | \
30                                 (AC_IDLE << EV_NUMERIC_ERROR) | \
31                                 (AC_IDLE << EV_STACK_OVERFLOW) | \
32                                 (AC_IDLE << EV_MEMORY_VIOLATION) | \
33                                 (AC_IDLE << EV_HARDWARE_FAULT))
34
35 #endif
```

J.4. Fichero con las órdenes de compilación

```
all: resident_sw

LTE_PATH=/opt/sdk-lithos-arm-zybo/
include ${LTE_PATH}/config.mk

# XM and toolchain
TARGET_CFLAGS+=-isystem ${XM_PATH}/include

# LTE variables
TARGET_CFLAGS+=-I${LTE_PATH}/include
LTCF=${LTE_PATH}/lib/ltcf.c
LTE_KERNEL=${LTE_PATH}/lib/lte_kernel.o
LIBC=../libc

# build partitions
main1.arm: main1.c
	${TARGET_CC} ${TARGET_CFLAGS} -c -o $@ $^
main2.arm: main2.c
	${TARGET_CC} ${TARGET_CFLAGS} -c -o $@ $^
main3.arm: main3.c
	${TARGET_CC} ${TARGET_CFLAGS} -c -o $@ $^

ltcf1.arm: system1.ltcf
	${TARGET_CC} ${TARGET_CFLAGS} --include $^ -c -o $@ ${LTCF}
ltcf2.arm: system2.ltcf
	${TARGET_CC} ${TARGET_CFLAGS} --include $^ -c -o $@ ${LTCF}
ltcf3.arm: system3.ltcf
	${TARGET_CC} ${TARGET_CFLAGS} --include $^ -c -o $@ ${LTCF}

partition1.arm: main1.arm ${LTE_KERNEL} ltcf1.arm
	${TARGET_LD} -T ${LTE_PATH}/lds/lte-xmarm.lds -Ttext=$(shell $
	   {LTE_PATH}/bin/xpathstart 0 xm_cf.arm.xml) -o $@ $^ -I${
	   LIBC}/include --start-group `${TARGET_CC} -print-libgcc-
	   file-name ${TARGET_CFLAGS_ARCH}` ${LIBC}/std_c.o ${XM_PATH
	   }/lib/libxm.a --end-group
partition2.arm: main2.arm ${LTE_KERNEL} ltcf2.arm
	${TARGET_LD} -T ${LTE_PATH}/lds/lte-xmarm.lds -Ttext=$(shell $
	   {LTE_PATH}/bin/xpathstart 1 xm_cf.arm.xml) -o $@ $^ -I${
	   LIBC}/include --start-group `${TARGET_CC} -print-libgcc-
	   file-name ${TARGET_CFLAGS_ARCH}` ${LIBC}/std_c.o ${XM_PATH
	   }/lib/libxm.a --end-group
partition3.arm: main3.arm ${LTE_KERNEL} ltcf3.arm
```

```
	${TARGET_LD} -T ${LTE_PATH}/lds/lte-xmarm.lds -Ttext=$(shell $
	   {LTE_PATH}/bin/xpathstart 2 xm_cf.arm.xml) -o $@ $^ -I${
	   LIBC}/include --start-group `${TARGET_CC} -print-libgcc-
	   file-name ${TARGET_CFLAGS_ARCH}` ${LIBC}/std_c.o ${XM_PATH
	   }/lib/libxm.a --end-group

# Use XM toolchain to prepare the binary image
partition1.xef: partition1.arm
	${XM_PATH}/bin/xmeformat build -o $@ $^
partition2.xef: partition2.arm
	${XM_PATH}/bin/xmeformat build -o $@ $^
partition3.xef: partition3.arm
	${XM_PATH}/bin/xmeformat build -o $@ $^

%.xmc: %.xml
	XTRATUM_PATH=${XM_PATH} TARGET_OBJCOPY=${TARGET_OBJCOPY}
	   TARGET_CC=${TARGET_CC} TARGET_CFLAGS_ARCH=$(
	   TARGET_CFLAGS_ARCH) ${XM_PATH}/bin/xmcparser $^ -o $@

%.xmc.xef: %.xmc
	${XM_PATH}/bin/xmeformat build -m -o $@ $^


container.bin: partition1.xef partition2.xef partition3.xef xm_cf.arm
    .xmc.xef
	${XM_PATH}/bin/xmpack build \
		-h ${XM_PATH}/lib/xm_core.xef:xm_cf.arm.xmc.xef \
		-p 0:partition1.xef \
		-p 1:partition2.xef \
		-p 2:partition3.xef $@

resident_sw: container.bin
	XTRATUM_PATH=${XM_PATH} ${XM_PATH}/bin/rswbuild $^ resident_sw

runZybo:
	xsdb -tcl ../zybo_xsdb.ini

clean:
	rm -f *.arm resident_sw resident_sw container.bin *.xef *.xmc
```