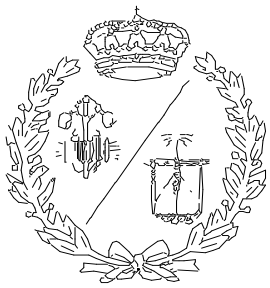
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Grado

SISTEMA PARA USO REMOTO DE PLACAS DE LABORATORIO BASADAS EN FPGA

SYSTEM FOR REMOTE USE OF FPGA-BASED LABORATORY BOARDS

Para acceder al Título de

GRADUADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Daniel Santana Coterillo
Tutor: Héctor Posadas Cobo

Septiembre-2021

RESUMEN

El objetivo del proyecto es el desarrollo de una infraestructura que permita la utilización de placas de laboratorio basadas en FPGA desde un ordenador remoto. Esta infraestructura tiene como finalidad conseguir crear un sistema mediante el cual los alumnos desde sus propios hogares obtengan acceso a las placas de prototipado con FPGA usadas en el laboratorio para facilitar su trabajo autónomo.

Así, por un lado, se desarrolla una aplicación de escritorio que ofrecerá una interfaz gráfica mediante la cual sea posible modificar el comportamiento de las placas de laboratorio de forma sencilla, intuitiva y atractiva para el alumnado. Por otro lado, en el laboratorio se emplazará un servidor que recibirá las ordenes enviadas por el ordenador y las redirigirá a la placa que se quiera controlar. Por motivos de relación calidad precio, velocidad y sencillez a la hora de su utilización el servidor será albergado por una Raspberry Pi.

Como dispositivo de ejemplo para mostrar el funcionamiento se utiliza una placa de laboratorio Basys 3, dispositivo del cual se podrá modificar el estado de sus interruptores, leds, botones, monitor de 7 segmentos, etc. así como leer los estados de dichos componentes. Esta placa esté siendo utilizada en la docencia de varias asignaturas de diseño electrónico digital de la Universidad de Cantabria.

Con la realización de este proyecto se pretende dotar a los alumnos de la capacidad de probar sus diseños con total autonomía sin necesidad de desplazarse hasta el laboratorio. Esto solventa el problema de la limitación horaria en que los laboratorios se encuentran abiertos pudiendo así continuar con el aprendizaje fuera de dichos horarios y de una forma cómoda. También es un recurso interesante en tiempos de Covid para gestionar el confinamiento de alumnos o la reducción del aforo de los laboratorios.

ABSTRACT

The objective of the project is the development of an infrastructure that allows the use of FPGA-based laboratory boards from a remote computer. The purpose of this infrastructure is to create a system that brings the students the opportunity of getting access to the prototyping boards with FPGA used in the laboratory without leaving their house facilitating their autonomous work.

The infrastructure is divided in two elements. On the one hand, a desktop application has been developed. This application presents a graphical interface that enables the interactions of the students with the laboratory boards equipment in a simple, intuitive and attractive way from the remote computer. On the other hand, a server will be located in the laboratory, which will receive the orders sent by the computer and redirect them to the device to be controlled. For reasons of value for money, speed and simplicity at the time of use, the server will be hosted by a Raspberry Pi.

As an example device to show the operations, a Basys 3 FPGA board is used. Demonstrations will show how it is possible to modify the status of its switches, LEDs, buttons, 7-segent display, etc. from the remote computer. This development board is currently use in several subjects related with digital electronics teaching in the University of Cantabria.

With the completion of this project, it is intended to provide students the ability to test their designs with total autonomy without having to travel to the laboratory. This solves the problem of limited hours in which the laboratories are open, thus being able to continue learning outside of those hours and in a comfortable way. Moreover, if provides a valuable resource in Covid times.

Documento Nº1: Memoria

ÍNDICE

ÍNDICE DE FIGURAS

Índice de tablas

1 INTRODUCCIÓN

Dentro de la enseñanza en laboratorio realizada en universidades, institutos, colegios y academias, surgen una serie de problemas, siendo las más notables la limitación horaria del uso del laboratorio y la limitación de puestos disponibles. Esto conlleva que los alumnos tengan que realizar el trabajo en horas limitadas, lo que en ocasiones resulta imposible para el alumno o supone la adaptación de los trabajos realizados en el laboratorio a el tiempo permitido, pudiendo llevar a la supresión de conocimientos por falta de tiempo. Estos problemas se han incrementado dramáticamente con la aparición del Covid, debido a la reducción de aforo en los laboratorios, y los confinamientos de estudiantes derivados de la enfermedad.

En la actualidad, debido a la existencia de una gran cantidad de formas de enviar datos de forma remota, se plantea la posibilidad de crear un sistema que permita realizar dichas prácticas a distancia, siendo innecesaria la presencia del alumno frente a la máquina para realizar su manipulación o para realizar las pruebas de diseño pertinentes (*Figura 1*).

Figura 1: Control a distancia

Aprovechando esta oportunidad se decide hacer frente a los problemas de manejo presencial para su aplicación en la docencia de asignaturas de electrónica digital. La idea es crear un

sistema de control remoto (*Figura 2*) que permita el acceso de los alumnos desde cualquier lugar a través de un ordenador a los equipos de laboratorio designados, que serán las placas de prototipado basadas en FPGA que habitualmente usan los alumnos de estas asignaturas en las prácticas.

Figura 2: Control remoto

Para que esta tecnología sea de utilidad, se dispone de las siguientes características:

- Comodidad: el alumno podrá realizar el trabajo desempeñado sin necesidad de desplazamiento ni limitación horaria lo que permite que el alumno pruebe sus diseños cuando él quiera.

- Coste reducido: la implementación de los sistemas de control se realiza mediante dispositivos con un precio atractivo en cuanto a relación calidad/precio.

- Flexibilidad: El sistema opta de la posibilidad de adaptarse fácilmente a cambios para dar la oportunidad de utilizarse para realizar trabajos diferentes con una modificación simple y rápida.

- Uso simple: La implementación del sistema se realiza mediante una interfaz visual que implicará el control remoto de forma sencilla e intuitiva para el alumno.

- Respuesta rápida: El sistema responde con gran velocidad lo que evita que el alumno tenga que esperar cada vez que ejecuta una orden.

- Desplazamiento innecesario: el control remoto permite que la modificación se realice sin necesidad de estar en el mismo lugar que el sistema que se quiere tratar, incluso permitiendo la modificación desde países o incluso continentes distintos.

- Acceso a información: al realizar la modificación mediante ordenador y a través de un servidor se puede realizar una gestión de la información que almacene un registro de las tareas realizadas.

Utilizando una infraestructura de este tipo, se posibilita el trabajo autónomo del alumno dotándole de la opción de probar y revisar sus diseños fuera del horario de laboratorio, permitiendo un uso ilimitado que elimina las restricciones a la hora de la realización de prácticas desde donde considere necesario (*Figura 3*).

Figura 3: Trabajo a distancia

Adicionalmente debido a la crisis sanitaria vivida actualmente debido al Coronavirus (Covid19), tras el estado de alarma provocado por la pandemia se suprimieron todas las actividades no esenciales que fueran llevadas a cabo de forma presencial lo que imposibilitó la realización de prácticas de laboratorio llevando a una adaptación o incluso supresión de estas. La implantación de una infraestructura que permita el control remoto sustituye la necesidad de desplazamiento fuera del hogar y por tanto una menor exposición a el virus y da la posibilidad a los alumnos de realizar las practicas originalmente diseñadas desde un entorno seguro.

Por estas razones, este proyecto ha sido integrado dentro de la última convocatoria de Proyectos de Innovación Docente de la Universidad de Cantabria, para su aplicación en los laboratorios docentes.

2 OBJETIVO

El objetivo del proyecto consiste en el desarrollo de una infraestructura que permita el control de placas de prototipado basadas en FPGA (*Figura 4*) desde un ordenador remoto. Dicho sistema deberá permitir la programación de dicha FPGA, así como par la posibilidad de modificar las salidas y entradas de dicha placa que podrá ir conectada a una máquina, robot, sensor, etc. El objetivo es que los alumnos puedan modificar su comportamiento y/o obtener información de las salidas generadas sin necesidad de desplazamiento hasta donde se encuentra la unidad.

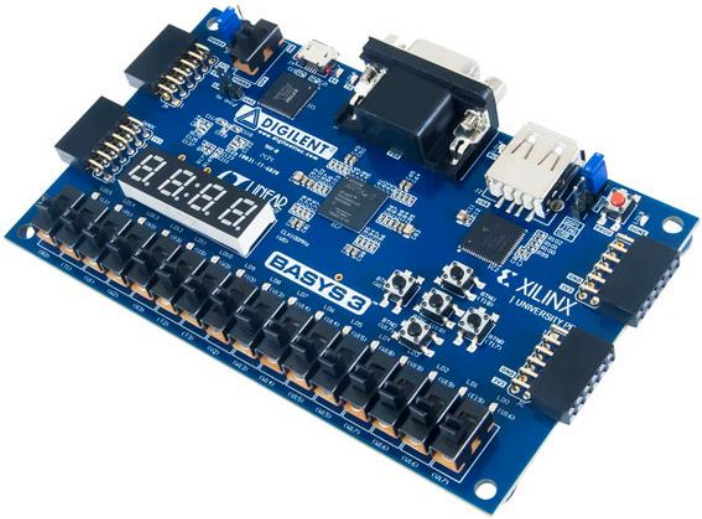


Figura 4: Placa de prototipado basada en FPGA

Para ello se realizará un sistema basado que permitirá al alumno, a través de una interfaz gráfica, modificar las entradas del sistema y visualizar sus salidas de una forma intuitiva y simple. Esta interfaz envíará las ordenes necesarias para permitir la interacción del alumno a través de la red del laboratorio a la palca de prototipado, que recibirá las ordenes enviadas, las analizará y en consecuencia modificará el estado de sus entradas y generará las salidas correspondientes.

De esta forma, el alumno podrá modificar las entradas recibidas por la placa sin necesidad de estar físicamente junto a ella, y así podrá modificar el comportamiento del dispositivo permitiendo cambios de estado dentro del circuito que repercutirán en la forma de trabajo de la máquina, robot, etc. y/o permitiendo la lectura de datos de salida emitidos por el dispositivo. Así el alumno podrá probar sus diseños y depurarlos.

De esta forma se solventa el problema planteado dando la posibilidad de realizar prácticas a distancia desde un lugar cómodo y seguro, sin necesidad de desplazamiento, liberando las restricciones horarias y dotando de la posibilidad de probar todos los diseños que se quieran analizar sin restricciones y con total autonomía.

2.1 PROPUESTA

La solución propuesta para cumplir el objetivo del proyecto se lleva a cabo en tres bloques claramente definidos: el primero está ligado al ordenador o lugar de trabajo desde donde se quiera controlar el sistema, el segundo al servidor que analizará la información enviada por el primer bloque y por último el dispositivo objetivo que se controlará mediante los dos bloques anteriores.

En el primer bloque se desarrolla una interfaz gráfica (*Figura 5*), mediante la creación de una aplicación de escritorio implementada usando el software Visual Studio desarrollado por Microsoft. Dicha interfaz debe resultar intuitiva y fácil de utilizar por el alumno. Por otra parte, dicha interfaz deberá contar además con un cliente TCP en cual conectará con el segundo bloque del proyecto enviando las ordenes que el usuario decida realizar modificando las diferentes opciones que brinda la interfaz. Para permitir el acceso remoto de los estudiantes, los equipos de laboratorio están integrados en el sistema UnicanLabs, que permite a los alumnos acceder remotamente a los PCs de los laboratorios, evitando las limitaciones de acceso impuestas por los sistemas de seguridad (Firewall) de la Universidad.

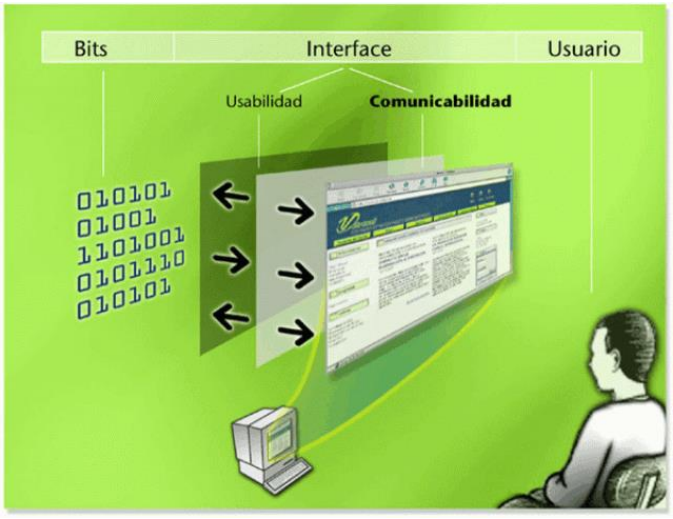


Figura 5:Interfaz gráfica

El segundo bloque consiste en el desarrollo de un sistema de control intermedio, que se encargue de la recepción de los datos enviados por el cliente TCP mediante la creación de un servidor TCP (*Figura 6*) e interactúe directamente con la palca de prototipado. Este sistema, por motivos de bajo coste y consumo (ya que deberá estar conectado en todo momento) se decide implementar en una Raspberry Pi 4, la cual, además de las características anteriores, brinda simpleza a la hora de desarrollar/modificar el software y velocidad de respuesta. Este servidor será el encargado de analizar la información que se envía desde el ordenador y aplicarla modificando los valores de entrada y salida de los pines GPIO en consecuencia. Estos pines irán conectados a la placa de prototipado que se quiera controlar pudiendo de tal manera modificar su comportamiento y permitiendo la obtención de datos generados por el dispositivo objetivo.

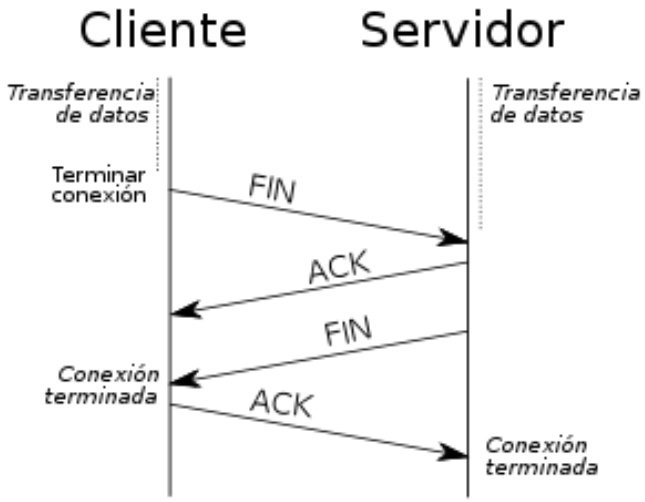


Figura 6: Comunicación TCP

El tercer y último bloque está formado por el propio dispositivo que se quiere controlar, que no será desarrollado en este proyecto. Este dispositivo recibirá el estado de cada uno de los pines GPIO de la Raspberry y modificará su comportamiento en consecuencia. Para el desarrollo del proyecto como dispositivo objetivo se utilizará como ejemplo una Basys 3, una tabla de inicio FPGA diseñada exclusivamente para Vivado Design Suite equipada con una arquitectura Xilinx Artix 7-FPGA. En dicho dispositivo se modificará el comportamiento de los interruptores, leds, lámpara de 7 segmentos, etc., demostrando de tal

manera como se puede modificar el estado de las diferentes partes del dispositivo y/o la obtención de información de este.

2.2 REQUISITOS DE DISEÑO

Para asegurar que la tecnología sea de utilidad y resulte atractivo para los alumnos debe cumplir las siguientes características:

- Atractivo visual: la interfaz debe de ser sencilla, atractiva e intuitiva para el alumno.

- Respuesta rápida: Se debe asegurar que la comunicación entre las distintas partes del proyecto sea lo más rápida posible, evitando así tiempos de espera.

- Flexibilidad: el sistema debe de poderse adaptar fácilmente para desempeñar diferentes tipos de trabajo realizando una modificación mínima.

- Fácil acceso a las herramientas: Utilizar en la medida de lo posible software gratuito y fácil de instalar para las partes que deban descargar los alumnos.

- Control visual: Disponer de un sistema para poder controlar visualmente lo que está ocurriendo en el otro extremo.

3 ESTADO DEL ARTE

Para desarrollar este trabajo, se ha revisado el estado del arte del acceso a placas basadas en FPGA, especialmente en entornos docentes. Por otro lado, en este capítulo también se describirá la tecnología base utilizada durante el proyecto se analizará dividiéndola en tres bloques, por un lado, el puesto de trabajo, por otro el servidor y por último el dispositivo objetivo que se desea controlar.

3.1 PROYECTOS RELACIONADOS

En este apartado se muestran proyectos realizados anteriormente que han servido como inspiración a la hora de plantear el proyecto y llevarle a cabo. Estos proyectos tienen como objetivo permitir el acceso de los alumnos a los recursos de laboratorio desde sus ordenadores. La solución de los proyectos se basa en la utilización de tecnología FPGA para realizar el control remoto. Algunos de estos proyectos son los siguientes:

3.1.1 Docencia semipresencial en laboratorios docentes para diseño de hardware digital en la ETSIT-UPM

Este proyecto plantea la primera opción que se barajó para llevar a cabo el control remoto donde se decidió utilizar una página web para albergar la interfaz visual y las funciones necesarias para realizar el control del dispositivo objetivo. El proyecto describe lo siguiente:

Desarrollado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad Politécnica de Madrid el proyecto plantea las medidas tomadas por la escuela para posibilitar el acceso de un mayor número de alumnos a los recursos de laboratorio mediante el empleo de las tecnologías de la información y las telecomunicaciones. El objetivo es mediante una aplicación web dotar a los alumnos de la capacidad de acceder a herramientas de software de desarrollo y a plataformas de desarrollo de prototipado reales destinadas al diseño de hardware digital de forma remota.

Para ello comienzan dotando a los alumnos de la posibilidad de acceso remoto con capacidad grafica a la red de ordenadores de laboratorio de forma segura y sencilla para el cliente. A continuación, se implementa una aplicación web asociada a una placa de prototipado con FPGA que permita configurar los elementos de la placa de prototipado, descarga de configuraciones en los dispositivos de hardware reconfigurable, inyección y extracción de datos para la prueba de módulos implementados, monitorización de elementos de la placa y accionamiento de interruptores y pulsadores virtuales. Por último, poner en marcha un portal web que permite acceder a la aplicación anteriormente descrita incluyendo funcionalidades como gestión de reservas, soporte de consultas online, foro, etc [1].

La idea de utilizar una interfaz basada en Web fue explorada por el tutor del trabajo fin de grado que se presenta en esta memoria, antes de su comeinzo. Sin embargo, esta solución presentó problemas de aplicación para su uso en plataformas de prototipado complejas, ya que la interfaz no podía usar servicios de las herramientas de desarrollo hospedadas en el PC, limitando sus capacidades. Por esta razón, en este trabajo se opta por el desarrollo de una aplicación de escritorio.

3.1.2 Laboratorio remoto FPGA: experiencia de un laboratorio compartido entre UPNA y UNIFESP

El proyecto sirvió como ejemplo para modificar la idea original del proyecto, decidiéndose crear un laboratorio remoto donde se concentre el servidor con todas las funciones que realizarán el proceso y crear una aplicación de escritorio que cada alumno instalará en su ordenador para enviar las ordenes necesarias. El proyecto propone lo siguiente:

Para evitar las limitaciones horarias en que los alumnos pueden acceder al laboratorio y dotar a estos de la posibilidad de acceder de manera remota en cualquier momento se decide crear un laboratorio remoto. Un laboratorio remoto es un sistema de hardware y software que permite a los estudiantes acceder a una FPGA real localizada en algún lugar de internet. En este artículo se presenta un laboratorio remoto internacional con diecisiete dispositivos conectados actualmente desarrollado por UPNA (España) y UNIFEST (Brasil) utilizando la tecnología y el mantenimiento de los laboratorios EdTech LabsLand en España de tal forma que los estudiantes de ambas instituciones pueden acceder.

Para que el laboratorio remoto sea de confianza permita la conexión de muchos usuarios concurrentes se recurre a sistemas relacionados con LabsLand. Uno de ellos es WebLabLib, el cual es de código libre y permite la integración de laboratorios dentro de la red. A través de él, el laboratorio obtiene acceso a la web de LabLands basada en sistema IDE, un entorno de compilación distribuida, LMS integración y otras características clave. Los alumnos pueden acceder a un IDE adaptado para lenguajes de definición de hardware que ofrece características como administración de archivos, sintaxis resaltada o complementación de código. De esta forma cuando el estudiante solicita que su código sea compilado se asigna la tarea a un controlador que puede estar colocado en cualquier lugar [2].

3.2 HERRAMIENTAS DEL PUESTO DE TRABAJO DE DESARROLLO

En este apartado se presenta el software utilizado para realizar todas las operaciones que se llevan a cabo desde el ordenador desde el que se ha desarrollado este trabajo. Entre ellos se encuentran los programas encargados para realizar el envío de datos al servidor mediante un cliente TCP albergado en una aplicación de escritorio, los programas para realizar modificaciones en el servidor de forma remota o los programas necesarios para completar la implementación del sistema:

3.2.1 Microsoft Visual Studio

La primera parte del proyecto consiste en realizar una interfaz visual desde la cual se controlará el comportamiento de las demás partes del proyecto, para ello se decide realizar una aplicación de escritorio utilizando el software Visual Studio.

Microsoft Visual Studio (*Figura 7*) es un entorno de desarrollo integrado para Windows y macOS compatible con múltiples lenguajes de programación. Visual Studio permite desarrollar sitios y aplicaciones web en entornos compatibles con la plataforma .NET, posibilitando la creación de aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, videoconsolas, etc [3].

Figura 7: Logo Microsoft Visual Studio

La aplicación de escritorio se diseña utilizando C++ y en ella se crea una interfaz gráfica que dispone de varios accionamientos, que envían una señal al servidor para modificar el estado de sus pines GPIO. Para manipular el dispositivo elegido, como receptor a controlar, se ha decidido realizar una interfaz con cuatro pulsadores, cuatro leds, dos interruptores y una opción para inicializar la placa y una ventana que mostrara en video la imagen capturada por el servidor. Además, se incluirá un cliente TCP que realizará las comunicaciones con el servidor. De esta manera cuando se pulse uno de los accionamientos el cliente TCP conectará con el servidor y enviará la señal que permitirá el cambio en el dispositivo receptor relativa a dicho accionamiento.

3.2.2 PuTTY

Para poder conectar con la Raspberry, el primer procedimiento utilizado es mediante comunicación SSH. Putty (*Figura 10*) es un cliente SSH y Telnet con el que podemos conectarnos a servidores remotos iniciando una sesión entre ellos que nos permite ejecutar comandos. El ejemplo más claro es cuando se emplea PuTTY para ejecutar comando en un servidor VPS y así poder instalar algún programa o configurar alguna parte del servidor [6].

Figura 8: Logo PuTTY

Este software permite grabar la información IP de la Raspberry pudiendo conectar con ella de manera sencilla y rápida. Una vez iniciada la comunicación se genera una conexión que nos permite crear un punto intermedio para probar las modificaciones realizadas en el servidor sin necesidad de lanzar comandos desde la interfaz visual.

3.2.3 Notepad++

Para facilitar la modificación de los programas desarrollados en la placa de control al utilizar acceso por SSH se ha usado Notepad++. Notepad++ (*Figura 8*) es un editor de código fuente gratuito y un reemplazo del Bloc de notas que admite varios idiomas. Notepad ++ se basa en el componente de edición Scintilla, está escrito en C++ y utiliza Win 32 API y STL, lo que garantiza una mayor velocidad de ejecución y un tamaño de programa más pequeño [4].

Figura 9: Logo Notepad++

De esta forma se pueden realizar modificaciones en el código que se quieran implementar sin necesidad de conectar con la Raspberry Pi, pudiéndolo hacer de manera cómoda desde el ordenador. Esta aplicación fue realmente útil para modificar códigos cuando el servidor aun no estaba configurado completamente.

3.2.4 RealVNC

Para poder tomar el control y modificar los archivos de la Raspberry Pi diretamente, se necesita conectarla a una serie de periféricos como un monitor, un teclado y un ratón. Para evitar la necesidad de utilizar dichos periféricos se utiliza RealVNC.

RealVCN (*Figura 9*) es un programa de software libre basado en una estructura cliente-servidor que permite observar las acciones del ordenador servidor remotamente a través de un ordenador cliente. Permite compartir la pantalla de una maquina con cualquier sistema operativo que admita VNC conectándose desde otro ordenador o dispositivo que disponga de un cliente VNC [5].

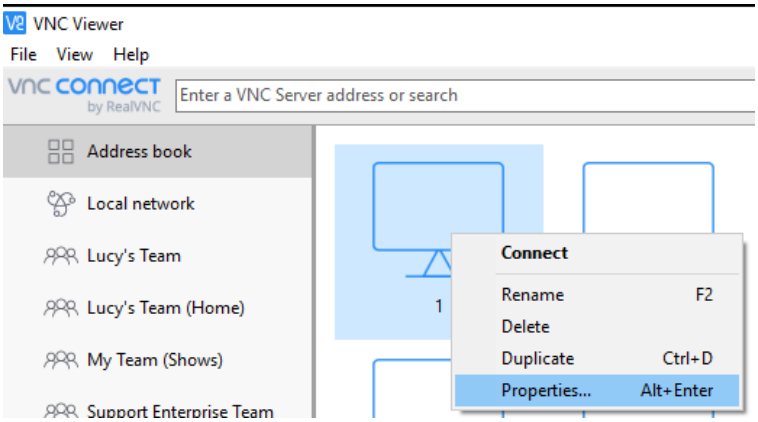


Figura 10: RealVNC

Esto permite clonar la imagen que la Raspberry enviaría al monitor y proyectarla en el ordenador, pudiendo además utilizar los propios periféricos del ordenador. De esta manera se permite realizar también modificaciones en la Raspberry utilizando el ordenador, pudiendo cambiar la configuración sin necesidad de desplazarte hasta donde se encuentre la Raspberry. Así se pueden realizar modificaciones no solo en la aplicación de escritorio sino también en el servidor desde un único puesto de trabajo. Esta solución se ha utilizado en etapas más avanzadas del diseño, donde era necesario una mayor interacción con la placa de control.

3.2.5 VLC Media Player

Para que Visual Studio muestre en directo un video de lo que observa la cámara colocada en el servidor y así poder tener un control visual de lo que está ocurriendo en el otro extremo se necesita de un reproductor multimedia. Debido a la compatibilidad con Visual Studio se decide usar VLC Media Player.

VLC Media Player (*Figura 11*) es un reproductor multimedia libre y de código abierto multiplataforma con capacidad para reproducir la mayoría de archivos multimedia además de capacidad de streaming [7].

Figura 11: Logo VLC Media Player

3.3 PLATAFORMA UTILIZADA COMO CONTROLADOR

La siguiente parte del proyecto es realizar un módulo de control que será el que recibirá las ordenes enviadas por la aplicación de escritorio, las analice y modifique el estado de sus salidas para interactuar con la placa de desarrollo. En este apartado se analiza el hardware donde se albergará el servidor y el software utilizado para la implementación:

3.3.1 Raspberry Pi

Debido a la relación calidad precio, a los requisitos del diseño y a la simpleza a la hora de realizar un servidor se decide utilizar una Raspberry Pi 4 como dispositivo para albergar dicho servidor.

Raspberry Pi (*Figura 12*) es una serie de ordenadores de placa reducida de bajo coste desarrolladas en el reino unido desarrolladas por la Raspberry Pi Foundation con el objetivo de promover la enseñanza en las escuelas, sin embargo, acabo de siendo más popular de lo esperado viéndose utilizada fuera del mercado, cumpliendo objetivos en el campo de la robótica, domótica o incluso utilizándose para transformar una televisión normal en una Smart TV o convertirla en un emulador de consolas antiguas [8].

Figura 12: Logo y eslogan de Raspberry Pi

- Raspberry Pi 4

  A diferencia de los modelos anteriores, la nueva Raspberry Pi 4 (*Figura 13*) presenta una mejora en cuanto a velocidad y rendimiento, creando una experiencia de escritorio completa de forma fluida con un formato más pequeño, más eficiente energéticamente y con una mejor relación calidad-precio [9].

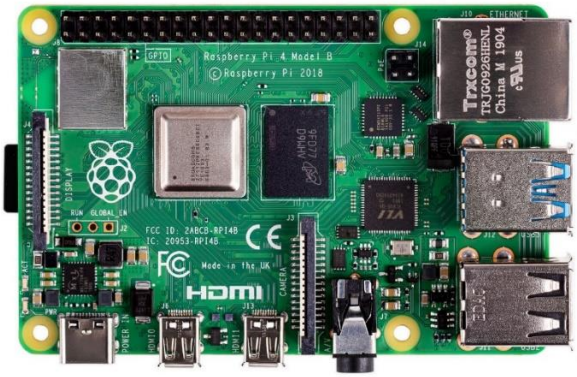


Figura 13: Raspberry Pi 4

En este último modelo se sustituyen los puertos HDMI por dos puertos microHDMI, por primera vez se introduce el USB 3.0 y se excluye la limitación de 300 Mbps del puerto Ethernet. Tiene un procesador Broadcom tres veces más eficiente que en modelos anteriores y está disponible en tres modelos diferentes, en los cuales varia la cantidad de memoria RAM siendo de 2GB, 4GB o 8GB [10].

- Raspberry Pi OS

Antiguamente conocido como Raspbian es una distribución del sistema operativo GNU/Linux basado en Debian, se trata de un software libre para la SBC Raspberry Pi (*Figura 14*) orientado en la enseñanza. Es el sistema operativo proporcionado actualmente por la Raspberry Pi Foundation como sistema operativo primario para la familia de placas de SBC de Rasberry Pi.

Figura 14: Raspberry Pi OS

Técnicamente es una adaptación de un programa a otra plataforma no oficial de Debian armhf para el procesador de Raspberry Pi, con soporte optimizado para cálculos en coma flotante por hardware [11].

Utilizando el sistema operativo de Rasberry Pi, se utiliza la consola de comandos para crear un servidor. Se introducen una serie de códigos que realizarán la función de servidor TCP, el cual recibirá la información enviada desde el ordenador y la analizará determinando en función de que accionamiento se pulso en la aplicación de escritorio que pin del puerto GPIO deberá cambiar su valor. El puerto GPIO irá conectado al dispositivo objetivo, de tal forma

que cuando se cambia el valor de uno de sus pines se modifica el comportamiento del componente electrónico al que va conectado.

Este servidor junto con los programas que analizan y envían la señal se implementan de forma que se inicien en el arranque de la Raspberry Pi y estén continuamente escuchando peticiones, de tal forma que nada más arrancar el dispositivo se pueda enviar peticiones en cualquier momento.

3.3.2 Motion

Motion es un programa altamente configurable que monitoriza las señales de video de muchos tipos de cámaras y permite trabajar con ellas. Este programa es capaz de crear videos o guarda imágenes de la actividad que se esté visualizando, además de ofrecer diversas capacidades, como grabación de paso a través de muchas cámaras IP, ver transmisión en vivo de cámara o invocar scripts cuando ocurren determinados eventos. El programa tiene soporte completo de TLS (https) con autenticación para control web y transmisiones. Eso permite usar Motion con muchos tipos de cámaras de red a través de protocolos como RTSP, RTMP y HTTP, utilizando codidificación V4L2. En este trabajo se ha utilizado para generar un servidor web que emita en directo las imágenes capturadas por la cámara conectada a la Raspberry, para que desde el ordenador remoto se puedan visualizar los efectos de los programas descargados en la placa de prototipado.

3.4 DISPOSITIVO OBJETIVO

Por último, se presenta la tecnología base utilizada como dispositivo a controlar:

3.4.1 Basys 3

Como ejemplo de dispositivo a controlar se decide utilizar una placa Basys 3.

Basys 3 (*Figura 15*) es una placa de desarrollo FPGA exclusivamente para Vivado Design Suite con arquitectura Xilinx Artix-7-FPGA. Basys 3 es la última incorporación a la línea de

Basys de placas de desarrollo FPGA para estudiantes o principiantes en tecnología FPGA [12].

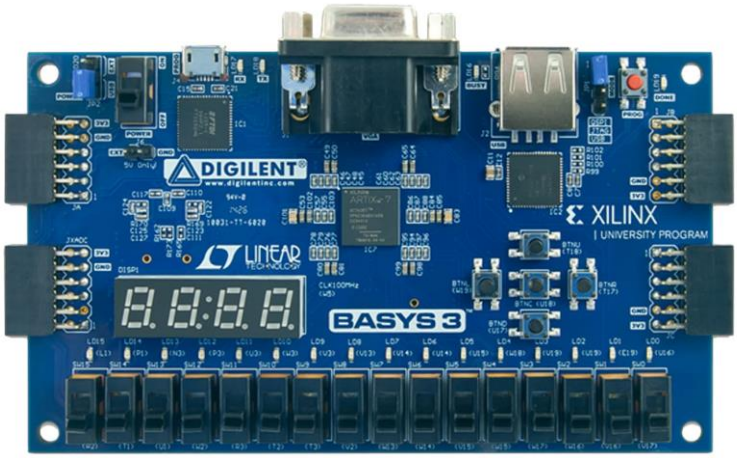


Figura 15: Basys 3

En ella se utilizarán los botones, los leds, la lampara de 7 segmentos y los interruptores para demostrar que el sistema funciona correctamente.

3.4.2 Vivado Design Suite

Para introducir la programación de la Basys se utiliza el software Vivado Design Suite.

Vivado Design Suite (*Figura 16*) proporciona las herramientas y la metodología necesarias para diseños basados en C. Se utiliza para optimizar la reutilización, reutilización del subsistema IP, automatización de la integración y cierre acelerado del diseño. Cuando se combina con la guía de metodología de diseño de productividad de alto nivel UltraFast permite a los diseñadores trabajar con un alto nivel de abstracción al tiempo que facilita la reutilización del diseño [13].

Figura 16: Logo Vivado Design Suite

4 SOLUCIÓN PROPUESTA

El proceso mediante el cual se ha llevado a cabo la solución del proyecto es el siguiente:

4.1 PLANTEAMIENTO INICIAL

La idea original del proyecto era realizar una página web que permitiese acceder a los alumnos desde casa. Para ello se planteó realizar un sistema de control con un PC del laboratorio donde se localizan habitualmente las placas. Este contaría con un servidor Apache donde desarrollar una aplicación que modificara, como ejemplo, las entradas y salidas de una Basys 3. Sin embargo, el servidor de la Universidad de Cantabria dispone de un cortafuegos que evita que alguien que no esté conectado a la red de la universidad pueda acceder a él. Debido a que no se puede acceder a dicho servidor sin estar en el campus universitario se decidió utilizar los PCs del laboratorio, conectados al sistema UnicanLabs, para poder superar las limitaciones del cortafuegos. El problema es que UnicanLabs independiza los ordenadores "virtuales" usados desde fuera de la universidad de los ordenadores físicos. Para poder resolver este problema se propuso modificar la ubicación del sistema de control, colocando un sistema junto a cada placa de prototipado de forma que puediese ser accedido independientemente de la gestión interna de UnicanLabs. Para este propósito, finalmente se decidió utilizar una Raspberry Pi debido a su bajo coste, simpleza y utilidad a la hora de funcionar como servidor.

Una vez decidido dónde se iba a colocar el servidor, el siguiente paso era decidir cómo realizar la interfaz visual. La idea inicial era realizarlo a través de una página web, pero debido a la simpleza que brinda Visual Studio se decidió elegirlo como software para diseñar una aplicación de escritorio basada en C++. Además, se comprobó que Xilinx solo proporciona un soporte mínimo para procesadores ARM (que son los que integra la placa Raspberry), de forma que para el control de placas de prototipado complejas (e.g. placas basadas en FGPAs para co-diseño) no podría realizarse completamente desde la Raspberry, y se necesitaba un control mixto PC/Raspberry, que no puede realizarse con una aplicación

Web hospedada en esta segunda. Esto hizo que finalmente se decidiese desarrollar una aplicación de escritorio para PC.

El siguiente paso era decidir que debía hacer la aplicación. Para ello como el dispositivo elegido para manejar en este Proyecto Fin de Grado era una Basys 3, se decidió que se podía comprobar el funcionamiento del sistema realizando un control que modifique el estado los interruptores, los botones, y leyendo los valores de los leds y el monitor de 7 segmentos. Además también se decidió integrar una webcam, para visualizar operaciones más complejas, como el control de un monitor VGA desde la Basys3.

4.2 DESARROLLO DEL CÓDIGO DE LA RASPBERRY PI

Para permitir el control de la placa Basys3 desde la raspberry PI, es necesario realizar dos actividades: abrir los pines de la Rasbperry adecuadamente (entradas/salidas) y luego generar/leer las señales correctamente.

4.2.1 APERTURA DE LOS PINES GPIO DE LA RASPBERRY PI

En primer lugar, se genera un código el cual estará incluido en la Raspberry que se encargue de abrir los puertos GPIO y definir cuáles son de entrada y cuáles de salida.

Para ello se comienza generando una función que eligiendo como argumento de entrada un pin GPIO se dirige a él y lo exporta (*Figura 17*):

```
int
GPIOExport(int pin)
{
#define BUFFER_MAX 3
    char buffer[BUFFER_MAX];
    ssize_t bytes_written;
    int fd;

    fd = open("/sys/class/gpio/export", O_WRONLY);
    if (-1 == fd) {
        fprintf(stderr, "Failed to open export for writing!\n");
        return(-1);
    }

    bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
    write(fd, buffer, bytes_written);
    close(fd);
    return(0);
}
```

Figura 17: Función GPIOExport

A continuación, se genera otra función que una vez exportado el pin lo configura como pin de entrada o de salida (*Figura 18*):

```
int
GPIODirection(int pin, int dir)
{
    static const char s_directions_str[]  = "in\0out";

#define DIRECTION_MAX 35
    char path[DIRECTION_MAX];
    int fd;

    snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/direction", pin);
    fd = open(path, O_WRONLY);
    if (-1 == fd) {
        fprintf(stderr, "Failed to open gpio %d direction for writing!\n",pin);
        return(-1);
    }

    if (-1 == write(fd, &s_directions_str[IN == dir ? 0 : 3], IN == dir ? 2 : 3)) {
        fprintf(stderr, "Failed to set direction!\n");
        return(-1);
    }

    close(fd);
    return(0);
}
```

Figura 18: Función GPIODirection

Si el GPIO es de entrada se debe realizar una función que lea si el estado de dicho pin está en alto (0) o en bajo (1) (*Figura 19*):

```
int
GPIORead(int pin)
{
#define VALUE_MAX 30
    char path[VALUE_MAX];
    char value_str[3];
    int fd;

    snprintf(path, VALUE_MAX, "/sys/class/gpio/gpio%d/value", pin);
    fd = open(path, O_RDONLY);
    if (-1 == fd) {
        fprintf(stderr, "Failed to open gpio %d value for reading!\n",pin);
        return(-1);
    }

    if (-1 == read(fd, value_str, 3)) {
        fprintf(stderr, "Failed to read value!\n");
        return(-1);
    }

    close(fd);

    return(atoi(value_str));
}
```

Figura 19: Función GPIORead

Si el GPIO es de salida se debe realizar una función que determine si el estado de dicho pin debe ser alto o bajo (*Figura 20*):

```
int
GPIOWrite(int pin, int value)
{
    static const char s_values_str[] = "01";

    char path[VALUE_MAX];
    int fd;

    snprintf(path, VALUE_MAX, "/sys/class/gpio/gpio%d/value", pin);
    fd = open(path, O_WRONLY);
    if (-1 == fd) {
        fprintf(stderr, "Failed to open gpio value for writing!\n");
        return(-1);
    }

    if (1 != write(fd, &s_values_str[LOW == value ? 0 : 1], 1)) {
        fprintf(stderr, "Failed to write value!\n");
        return(-1);
    }

    close(fd);
    return(0);
}
```

Figura 20: Función GPIOWrite

Por último, una vez se finaliza de utilizar el puerto GPIO el siguiente es paso dejar de exportar el puerto, para ello se realiza una función que va a el pin seleccionado y lo cierra (*Figura 21*):

```
int
GPIOUnexport(int pin)
{
    char buffer[BUFFER_MAX];
    ssize_t bytes_written;
    int fd;

    fd = open("/sys/class/gpio/unexport", O_WRONLY);
    if (-1 == fd) {
        fprintf(stderr, "Failed to open unexport for writing!\n");
        return(-1);
    }

    bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
    write(fd, buffer, bytes_written);
    close(fd);
    return(0);
}
```

Figura 21: Función GPIOUnexport

4.2.2 CONTROL DE LOS PINES GPIO DE LA RASPBERRY PI

El siguiente paso es definir qué deben hacer los puertos GPIO. Teniendo en cuenta las modificaciones que se quiere realizar en la Basys se genera una lista de las actividades a las que llame la interfaz del PC, dando lugar a diversas funciones dependiendo del argumento de entrada. Para comunicar la interfaz y la Raspberry, se opta por un procolo de texto, que se enviará sobre TCP-IP. Los textos de comunicación tiene como primer argumento una letra con la función que se quiere desempeñar y dependiendo la función uno o varios números para determinar el comportamiento de dicha función separados por espacios (estos espacios son importantes ya que se utilizaran más adelante como carácter delimitador para separar los argumentos) según el siguiente esquema:

Letra [numero1] [numero2]

Además, cuando haya que leer datos de la placa (típicamente para el monitor de 7 segmentos y los leds), la Raspberry enviará un texto del tipo Numero1Numero2Numero3. Estos números no tendrá espacios, sabiendo que en caso del monitor de 7 segmentos cada número tiene 2 dígitos, y en caso de los leds tienen 1.

La lista de letras utilizas es la siguiente:

- “i” para inicializar la placa.
- “d” para leer el display de la lampara de 7 segmentos.
- “b” para activar un botón y un numero (0,1,2,3) para determinar que botón activar.
- “l” para activar un led y un numero (0,1,2,3) para determinar que led iluminar.
- “s” para cambiar el estado de un interruptor, un numero (0,1) para determinar que interruptor se desea modificar y otro número (0,1) para elegir el estado de dicho interruptor.
- “r” para leer los puertos serie.
- “w” para escribir en los puertos serie.
- “n” para empezar un nuevo acceso (limpiar la cola los puertos serie).
- “e” para salir.

4.2.3 PROTOCOLO DE COMUNICACIÓN CON LA RASPBERRY PI

A continuación, se generan las funciones determinando qué debe hacer cada una. En primer lugar, se genera una la función que inicializa la placa. Esta función recoge los pines que se desean coger como pines de entrada y los exporta y los determina como pines de entrada utilizando las funciones descritas en el apartado anterior (*Figura 22*):

```
int i;
system("sudo djtgcfg prog -d Basys2 -i 0 -f /home/pi/digilent/download2.bit");
clear_fifo();
for(i=0;i<sizeof(input);i++){
    if (-1 == GPIOExport(input[i])){
        printf("Unable to open GPIO %d\n",input[i]);
        return(-1);
    }
    if (-1 == GPIODirection(input[i], IN)) {
        printf("unable to set GPIO %d as input\n",input[i]);
        return(-1);
    }
}
```

Figura 22: Función para abrir puertos

Se realiza lo mismo para los puertos de salida.

Después se genera una función que recoge el valor de cada uno de los segmentos de las lámparas de 7 segmentos y las muestra por pantalla. Los pines determinados para cada uno de los segmentos son el 23, 24, 6, 13, 19, 26 y 12 y tres pines más para determinar el carácter que se está leyendo que son el 16, 20 y 21. Todos estos pines se definen como pines de entrada quedando la función de la siguiente manera (*Figura 23*):

```
int read_display(){
    int i,val;
    int values[3]={0,0,0};
    int an_0=0,an_1=0,an_2=0,an_01=0,an_11=0,an_21=0;
    int seg_0,seg_1,seg_2,seg_3,seg_4,seg_5,seg_6;
    for(i=0;i<20;i++){
        an_0=GPIORead(16);        an_1=GPIORead(20);        an_2=GPIORead(21);
        seg_0=GPIORead(23); seg_1=GPIORead(24); seg_2=GPIORead(6); seg_3=GPIORead(13);
        seg_4=GPIORead(19); seg_5=GPIORead(26); seg_6=GPIORead(12);
        an_01=GPIORead(16);        an_11=GPIORead(20);        an_21=GPIORead(21);
        val = (seg_0 <<6) | (seg_1 << 5)| (seg_2 <<4) | (seg_3 <<3) | (seg_4 << 2)| (seg_5 <<1) | seg_6;
        if(!an_0 && !an_01) values[0] = val;
        if(!an_1 && !an_11) values[1] = val;
        if(!an_2 && !an_21) values[2] = val;
        usleep(300);
    }
    printf("%02X%02X%02X \n",convert_display(values[2]),convert_display(values[1]),convert_display(values[0]));
    return 0;
}
```

Figura 23: Función read_display

La siguiente función simula la pulsación de un botón poniendo el estado del pin GPIO asociado a valor alto durante un tiempo determinado y luego lo vuelve a valor bajo. Los pines GPIO determinados para realizar la función de botones son el 4, 17, 27 y 22 siendo todos de salida y representando al "Pulsador1", "Pulsador2", "Pulsador3" y "Pulsador4" respectivamente (*Figura 24*):

```
int button(int numbut){
    static const int id[4]={4,17,27,22};
    GPIOWrite(id[numbut], 1);
    usleep(300*1000);
    return GPIOWrite(id[numbut], 0);
}
```

Figura 24: Función button

El paso siguiente es realizar una función que leerá el estado de los leds de la Basys e imprimirá el resultado por pantalla. Los pines determinados para la lectura de los leds son el 10, 9, 11 y 7 siendo todos ellos de entrada y representando a "Led1", "Led2", "Led3" y "Led4" respectivamente (*Figura 25*):

```
int led(int numled){
    static const int id[4]={10,9,11,7};
    int val[3],i;
    for(i=0;i<3;i++)val[i]=GPIORead(id[i]);
    printf("%d%d%d%d \n",val[0],val[1],val[2],val[3]);
    return val;
}
```

Figura 25: Función led

A continuación, se implementa la función que simulara el comportamiento de un interruptor quedándose fijo en valor alto o bajo dependiendo del argumento de entrada que se introduzca al lanzar la función. Los pines GPIO definidos para cumplir la función de interruptores son el 2 y el 3 siendo ambos de salida y representando a "Interruptor1" e "Interruptor2" respectivamente (*Figura 26*):

```
int switches(int numSwt, int val){
    static const int id[2]={2,3};
    return GPIOWrite(id[numSwt], val);
}
```

Figura 26: Función switches

4.2.4 ASIGNACIÓN DE PINES DE LA RASPBERRY PI

Finalmente, los pines GPIO asociados a cada botón quedan recogidos en la siguiente tabla (*Tabla 1*):

Componente	Pin	Dirección
Lámpara 7 segmentos seg.1	23	IN
Lámpara 7 segmentos seg.2	24	IN
Lámpara 7 segmentos seg.3	6	IN
Lámpara 7 segmentos seg.4	13	IN
Lámpara 7 segmentos seg.5	19	IN
Lámpara 7 segmentos seg.6	26	IN
Lámpara 7 segmentos seg.7	12	IN
Lámpara 7 segmentos Carácter 1	16	IN
Lámpara 7 segmentos Carácter 2	20	IN
Lámpara 7 segmentos Carácter 3	21	IN
Pulsador1	4	OUT
Pulsador2	17	OUT
Pulsador3	27	OUT
Pulsador4	22	OUT
Led1	10	IN
Led2	9	IN
Led3	11	IN
Led4	7	IN
Interruptor1	2	OUT
Interruptor2	3	OUT

Tabla 1:Dirección pines GPIO

Adicionalmente el código está diseñado para leer y escribir en los puertos serie, sin embargo, finalmente no se implementaron para llevar a cabo el ejemplo de la Basys 3.

4.2.5 Servidor TCP

Por otro lado, en la Raspberry Pi se introduce el Código necesario para crear un servidor TCP que reciba los textos desde la interfaz gráfica. Para realizar el servidor TCP se comienza igual que en el cliente creando el socket y asignando la IP y el puerto (*Figura 39*):

```
int sockfd, connfd, len, i;
struct sockaddr_in servaddr, cli;

// socket create and verification
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
    printf("socket creation failed...\n");
    exit(0);
}
else
    printf("Socket successfully created..\n Binding ");
bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);
```

Figura 27: Creación de socket y asignación IP

A continuación, se vincula el socket creado con la dirección IP, lo que hace que el servidor esté listo para escuchar (*Figura 40*):

```
for (i=0;i<MAX_RETRY;i++){

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf(".");
        fflush(stdout);
        sleep(1);
    }
    else{
        printf("\nSocket successfully binded..\n");
        break;
    }
}
if (i== MAX_RETRY) {
        perror("socket bind failed...\n");
        exit(-1);
}
```

Figura 28: Vinculación IP

Por último, se recibe el paquete de datos enviado por el cliente y se incluye la función que permite chatear entre cliente y servidor (*Figura 41*):

```
while(1){
    pthread_t thread_id;
    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");

// Function for chatting between client and server
    pthread_create(&thread_id,NULL,func,(void*)connfd);
```

Figura 29: Recepción paquete de datos

Dicha función se ejecuta dentro de un bucle de forma infinita para que constantemente esté preparado para recibir información del cliente. Esta función en primer lugar recoge el mensaje enviado por el cliente y lo copia en el buffer (*Figura 42*):

```
// read the message from client and copy it in buffer
len = read(sockfd, buff, sizeof(buff));
buff[len] = 0;
printf("Readed message of size %d %s\n",len, buff);
```

Figura 30: Copiar mensaje en buffer

Luego se conecta al menú creado en el apartado 5.3 y tras recibir la cadena de caracteres generada por el cliente TCP la separa utilizando los espacios como delimitador y los ordena de tal forma que sea compatible con los argumentos de entrada de las funciones del menú (*Figura 43*):

```
char* trim(char* cadena){
    char *ptr=cadena;
    int len = strlen(cadena);
    char * end = cadena+len;
    while(isspace(*ptr)&&ptr<end) ptr++;
    char* back = end -1;
    while(isspace(*back) && back >= cadena){
        back[0]='\0';
        back--;
    }
    return cadena;
}
char *argv[10];//cadena;

char **ReadData(char *cadena, int*argc_ptr){
    int argc = 0, num = 0;
    char delimitador[] = " ";
    cadena=trim(cadena);
    char *tmp=trim(cadena);

    while((tmp=strchr(tmp,' '))!=NULL){
    tmp++;
    argc++;
    }
    if(argc>9) argc=9;
    //argv = (char**)malloc(argc*sizeof(char*));

    char *token = strtok(cadena, delimitador);
    while(token != NULL){
        if(num>argc){perror("More arrays found in 'cadena' than expected");}
        // Sólo en la primera se pasa la cadena; en las siguientes se pasa NULL
        //printf("Token: %s\n", token);
        argv[num++]=token;
        token = strtok(NULL, delimitador);
            // Añadir cada uno de los valores leidos en columnas
//      rbind(argv[],token);
    }
    *argc_ptr = num;
    return argv;
}
```

Figura 31: Adaptación de la cadena de caracteres

Una vez obtenidos los argumentos entra al menú y realiza la función correspondiente a la orden enviada por el cliente. Hecho esto el servidor copia la respuesta en el buffer y la envía de vuelta al cliente (*Figura 44*):

```
// print buffer which contains the client contents
printf("From client: %s\t To client : ", buff);
bzero(buff, MAX);
n = 0;
// copy server message in the buffer
  while ((buff[n++] = getchar()) != '\n');
  buff[n] = 0;
  printf("Readed message of size %d from console: %s\n",n, buff);
strcpy(buff, "done\n");
// and send that buffer to client
write(sockfd, buff, strlen(buff)+1);

// if msg contains "Exit" then server exit and chat ended.
if (strncmp("exit", buff, 4) == 0) {
    printf("Server Exit...\n");
    break;
}
```

Figura 32: Respuesta del servidor

Finalizada la conversación el servidor reinicia el bucle preparándose de nuevo para recibir una nueva orden.

Para asegurar que el servidor se inicia nada más arrancar la placa el código genera se debe incluir en el archivo rc.local. Esto hace que el programa se ejecute nada más iniciar la placa sin necesidad de hacerlo manualmente.

4.2.6 IMAGEN EN TIEMPO REAL

Para tener un control visual de lo que está ocurriendo en el dispositivo que se quiere controlar se coloca una cámara web conectada a la Raspberry Pi apuntando hacia dicho dispositivo. Dicha cámara deberá transmitir continuamente lo que está capturando. Para llevar esto a cabo se instala el programa “motion” en la Raspberry Pi introduciendo el siguiente comando en la ventana de comandos:

sudo apt-get install motion

Una vez instalado motion se accede al archivo de configuración principal situado en la ruta /etc/motion/motion.conf y se modifican sus propiedades. Para conseguir que motion se ejecute nada más arrancar la placa se edita la propiedad start_motion_daemon y se coloca en “yes”, la propiedad stream_localhost se coloca en off para poder acceder al stream de forma remota, el stream se genera a través del puerto 8081 y se realizan ciertas

modificaciones de las propiedades para conseguir que la imagen sea fluida y con una latencia lo más baja posible.

Instalado el motion y corriendo se comprueba que la imagen se está generando correctamente accediendo desde el navegador a la dirección IP de la Raspberry seguida del puerto 8081.

Cuando se compruebe que la imagen se envía correctamente se modifica la interfaz visual para que muestre la imagen en la ventana designada para ello. Dicha modificación consiste en insertar un control ActiveX utilizando un plugin de VLC Media Player el cual se utilizará como reproductor multimedia para mostrar la captura de la cámara en tiempo real.

Esta modificación conecta a través del reproductor VLC Media Player con la imagen generada por motion y la muestra en la ventana de la interfaz visual refrescando continuamente para conseguir una imagen en directo.

4.3 CREACION DE LA INTERFAZ VISUAL

Para llevar a cabo el control de una manera sencilla, cómoda, intuitiva y atractiva se genera una interfaz visual donde mediante el ordenador pulsando un clic se pueda decidir que función debe desempeñar la Raspberry.

Utilizando el software Visual Studio se genera una aplicación de escritorio basada en C++. Dentro de esta aplicación se crea un “Dialog”, una interfaz visual, donde se colocarán los botones que ejecuten las ordenes o muestren las lecturas. Para esta esta aplicación se añaden cuatro botones, cuatro botones radiales que simularan los leds, dos “checkbox” que simulen los interruptores, un botón para inicializar la Raspberry y una ventana donde se mostrará en vivo la captura de una cámara conectada a la Raspberry como muestra la figura (*Figura 27*):

Figura 33: Interfaz visual

Creada la interfaz el software Visual Studio genera automáticamente el código necesario para mostrar dicha interfaz. Accediendo a las funciones creadas por Visual estudio para cada una de las herramientas se modifica el contenido para que envíen una cadena de texto, la cual será recibida por la Raspberry Pi, con los argumentos necesarios para que el servidor desempeñe la función respectiva al botón que se ha pulsado.

En primer lugar, se rellena el código respectivo al botón "Inicializar" el cual enviará una "i" a la Raspberry para que inicie los puertos GPIO. Dicho botón deberá pulsarse al principio para que inicialice la placa y los demás botones funcionen correctamente. El código empleado es el siguiente (*Figura 28*):

```
// Boton de inicializar
void menu_inicio::OnBnClickedButton5()
{
    // TODO: Agregue aquí su código de controlador de notificación de control
    sendData("DontCare i");
}
```

Figura 34: Función boton de inicializar

A continuación, se rellena el código referente a las "checkbox" de tal manera que detecte si esta activada y por tanto envíe a el servidor los argumentos para que realice la función que simula un interruptor en estado alto y en caso contrario si esta desactivada que simule un estado bajo (*Figura 29*):

```
// Interruptor 1
void menu_inicio::OnBnClickedCheck1()
{
    int status = ((CButton*)GetDlgItem(IDC_CHECK1))->GetCheck();
    // TODO: Agregue aquí su código de controlador de notificación de control
    printf("Boton: state %d\n", status);
    if (status == 1) {
        sendData("DontCare s 0 1");
    }
    if (status == 0) {
        sendData("DontCare s 0 0");
    }
}
```

Figura 35: Función Interruptor

Después se rellena el apartado designado para los pulsadores en cual enviará los argumentos necesarios para que se simule que se ha pulsado un interruptor (*Figura 30*):

```
// Pulsador 1
void menu_inicio::OnBnClickedButton1()
{
    // TODO: Agregue aquí su código de controlador de notificación de control
    sendData("DontCare b 0");
}
```

Figura 36: Función Pulsador

Por último, se rellena la parte del código que muestra el estado de los leds. Esta función debe modificarse para que periódicamente envíe una señal a la Raspberry para que lea el estado de los leds y devuelva su estado. A continuación, en función de los estados que devuelve la lectura en la interfaz se iluminan los botones referentes a los leds que estaban iluminados (*Figura 31*):

```
void menu_inicio::OnTimer(UINT nID) {
    static int value = 0;
    ledButton->SetCheck(value);
    value = !value;

    UpdateData(false);
    CDialog::OnTimer(nID);
}
```

Figura 37

El control para la ventana que mostrara la imagen se explica más adelante.

4.3.1 CONEXIÓN TCP

Para que el ordenador y la Raspberry puedan comunicarse se implementa un cliente y un servidor TCP. Por un lado, en la aplicación de escritorio se añade un cliente TCP y en la Raspberry se introduce el servidor TCP de tal forma que se produzca un flujo de información entre ellos.

En primer lugar, en la aplicación de escritorio se añade una clase en la cual se generará un cliente TCP. Para ello se comienza inicializando el Winsock, una biblioteca dinámica con finalidad de implementar TCP/IP que incluye soporte para envío y recepción de paquetes de datos a través de sockets (*Figura 32*):

```
// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
    printf("WSAStartup failed with error: %d\n", iResult);
    return 1;
}
```

Figura 38: Inicializar Winsock

A continuación, se obtiene la información de la dirección del servidor y de los puertos (*Figura 33*):

```
// Resolve the server address and port
CStringA strA(addr);
CStringA strA2(port);
LPCSTR addrStr = strA;
LPCSTR portStr = strA2;
iResult = getaddrinfo(addrStr, portStr, &hints, &result);
if (iResult != 0) {
    printf("getaddrinfo failed with error: %d\n", iResult);
    WSACleanup();
    return 1;
}
```

Figura 39: Información de dirección y puertos

Después, se crea un socket, un hilo existente entre el servidor y el cliente para que lean y escriban la información y se conecta con el servidor (*Figura 34*):

```
// Attempt to connect to an address until one succeeds
for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {

    // Create a SOCKET for connecting to server
    ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
        ptr->ai_protocol);
    if (ConnectSocket == INVALID_SOCKET) {
        printf("socket failed with error: %ld\n", WSAGetLastError());
        WSACleanup();
        return 1;
    }

    // Connect to server.
    iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        closesocket(ConnectSocket);
        ConnectSocket = INVALID_SOCKET;
        continue;
    }
    break;
}
```

Figura 40: Creación de socket

A continuación, se genera un código que mantiene la recepción de datos hasta que se cierre la conexión (*Figura 35*):

```
// Receive until the peer closes the connection

iResult = recv(ConnectSocket, recbuf, reclen, 0);
if (iResult > 0)
    printf("Bytes received: %d\n", iResult);
else if (iResult == 0)
    printf("Connection closed\n");
else
    printf("recv failed with error: %d\n", WSAGetLastError());
```

Figura 41: Recepción de datos

Y se envía el buffer inicial (*Figura 36*):

```
// Send an initial buffer
iResult2 = send(ConnectSocket, sendbuf, sendlen, 0);
if (iResult2 == SOCKET_ERROR) {
    printf("send failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}
```

Figura 42: Buffer inicial

Por último, se cierra el socket y se finaliza la comunicación TCP (*Figura 37*):

```
int closeTCP(){
    // cleanup
    closesocket(ConnectSocket);
    WSACleanup();

    return 0;
}
```

Figura 43: Cerrar socket

Una vez generado el cliente TCP se modifica la aplicación de escritorio para que al iniciarse comience la comunicación y una vez conectado las cadenas de caracteres generadas por la interfaz visual serán enviadas al servidor (*Figura 38*):

```
menu_inicio::menu_inicio(CWnd* pParent /*= nullptr*/)
    : CDialogEx(MenuDeInicio, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    //if (connectTCP(CString("127.0.0.1"), CString("5000")) == 0) {
    if (connectTCP(CString("169.254.125.244"), CString("2345")) == 0) {
        ConnectSocket = getSocket();
        ledButton = NULL;
    }
}
```

Figura 44: Comunicación TCP

5 RESULTADOS EXPERIMENTALES

Para comprobar el funcionamiento del sistema desarrollado, primero se desarrolló un código que descargar en la placa Basys3, y luego se utilizó la infraestructura para probar su funcionamiento.

5.1 LOGICA DE EJEMPLO

Para realizar el control de la Basys3 se crea un nuevo proyecto en vivado. Una vez creado se definen los puertos I/O. Para que la aplicación funcione se crean buses para poder manipular tanto los componentes físicos de la propia basys como para simular la información que se recibe o es enviada por la Raspberry. Para ello se definen los siguientes puertos I/O (*Tabla 2*):

Nombre del puerto	Dirección	Bus	MSB	LSB
Boton	IN	X	3	0
BotonPi	IN	X	3	0
Caracteres	OUT	X	3	0
CaracteresPi	OUT	X	3	0
Interruptor	IN	X	1	0
InterruptorPi	IN	X	1	0
Led	OUT	X	3	0
LedPi	OUT	X	3	0
Segmentos	OUT	X	7	0
SegmentosPi	OUT	X	7	0
Reloj	IN	-	-	-
Reset	IN	-	-	-

Tabla 2: Asignación puertos I/O

Una vez definidos los puertos I/O se comienza a implementar el control de dichos puertos. Para realizar el ejemplo de funcionamiento se decide realizar el siguiente control:

- Los botones de la placa al ser pulsados activan los leds de la interfaz visual.

- Al pulsar los pulsadores de la interfaz visual se iluminan los leds de la Basys.
- Los interruptores de la placa conectan con el tercer y cuarto carácter de la lampara de siete segmentos de la Basys de tal manera que al accionar el primer interruptor en el primer carácter aparece un 1 y al activar el segundo interruptor en el segundo carácter aparece un 2.
- Los interruptores de la interfaz visual conectan con el primer y segundo carácter de la monitor de 7 segmentos de la Basys de tal forma que al pulsar el "Interruptor 1" en el tercer carácter aparece un 3 y al pulsar "Interruptor 2" aparece un 4 en el cuarto carácter.

Para ello se genera un código que cuando se pulsan los botones de la placa se redirige a la señal a los puertos PMOD de tal forma que la conectándolo a la Raspberry envíe una señal a la interfaz visual para que ilumine los apartados de los leds, que al pulsar los pulsadores de la interfaz visual se realice el mismo proceso a la inversa para encender los leds de la Basys, al activar los interruptores de la Basys se envía una señal a la lámpara de 7 segmentos para activar los caracteres correspondientes y a los puertos PMOD para dar a la Raspberry la posibilidad de leer dicha lámpara, al activar los interruptores de la interfaz visual la Raspberry envíe la señal a los puertos PMOD para que conecten con los caracteres de la lámpara de 7 segmentos. Además, se añade la cadena de bits de cada carácter de la lámpara de 7 segmentos para que salga el numero deseado (*Figura 45*):

```
begin
    LedPi<=Boton;
    Led<=BotonPi;
    Caracteres<=not(Interruptor&InterruptorPi);
    CaracteresPi<=not(Interruptor&InterruptorPi);

    process (Interruptor, InterruptorPi) begin
      if(Interruptor(0)='1') then
        Segmentos <= "00100101";
        SegmentosPi <= "00100101";
      elsif (Interruptor(1)='1') then
        Segmentos <= "10011111";
        SegmentosPi <= "10011111";
      elsif (InterruptorPi(0)='1') then
        Segmentos <= "10011001";
        SegmentosPi <= "10011001";
      else
        Segmentos <= "00001101";
        SegmentosPi <= "00001101";
      end if;
    end process;
```

Figura 45: Código vivado

El siguiente paso es definir los pines que va a tener cada puerto I/O para ello se asignan según la siguiente tabla (*Tabla 3*):

Nombre	Dirección	Pin	I/O Std
Boton[3]	IN	T18	LVCMOS33
Boton[2]	IN	T17	LVCMOS33
Boton[1]	IN	U17	LVCMOS33
Boton[0]	IN	W19	LVCMOS33
BotonPi[3]	IN	J1	LVCMOS33
BotonPi[2]	IN	L2	LVCMOS33
BotonPi[1]	IN	J2	LVCMOS33
BotonPI[0]	IN	G2	LVCMOS33
Caracteres[3]	OUT	W4	LVCMOS33
Caracteres[2]	OUT	V4	LVCMOS33
Caracteres[1]	OUT	U4	LVCMOS33
Caracteres[0]	OUT	U2	LVCMOS33
CaracteresPi[3]	OUT	K17	LVCMOS33
CaracteresPi[2]	OUT	M18	LVCMOS33
CaracteresPi[1]	OUT	N17	LVCMOS33
CaracteresPi[0]	OUT	P18	LVCMOS33
Interruptor[1]	IN	V16	LVCMOS33
Interruptor[0]	IN	V17	LVCMOS33
InterruptorPi[1]	IN	L17	LVCMOS33
InterruptorPi[0]	IN	M19	LVCMOS33
Led[3]	OUT	V19	LVCMOS33
Led[2]	OUT	U19	LVCMOS33
Led[1]	OUT	E19	LVCMOS33
Led[0]	OUT	U16	LVCMOS33
LedPi[3]	OUT	H1	LVCMOS33
LedPi[2]	OUT	K2	LVCMOS33
LedPi[1]	OUT	H2	LVCMOS33
LedPi[0]	OUT	G3	LVCMOS33

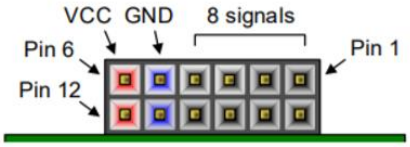
Segmentos[7]	OUT	W7	LVCMOS33
Segmentos[6]	OUT	W6	LVCMOS33
Segmentos[5]	OUT	U8	LVCMOS33
Segmentos[4]	OUT	V8	LVCMOS33
Segmentos[3]	OUT	U5	LVCMOS33
Segmentos[2]	OUT	V5	LVCMOS33
Segmentos[1]	OUT	U7	LVCMOS33
Segmentos[0]	OUT	V7	LVCMOS33
SegmentosPi[7]	OUT	A14	LVCMOS33
SegmentosPi[6]	OUT	A16	LVCMOS33
SegmentosPi[5]	OUT	B15	LVCMOS33
SegmentosPi[4]	OUT	B16	LVCMOS33
SegmentosPi[3]	OUT	A15	LVCMOS33
SegmentosPi[2]	OUT	A17	LVCMOS33
SegmentosPi[1]	OUT	C15	LVCMOS33
SegmentosPi[0]	OUT	C16	LVCMOS33
Reloj	IN	W5	LVCMOS33
Reset	IN	U18	LVCMOS33

Tabla 3: Asignación pines I/O

Los pines referentes a los puertos PMOD se seleccionan según la siguiente tabla (*Figura 46*), siendo asignados a los siguientes componentes (*Tabla 4*):

Componente	PMOD	Pines
Botones	JA	1-4
Leds	JA	7-10
Lámpara de 7 segmentos	JB	1-4 y 7-10
Carácter	JC	1-4
Interruptor	JC	7 y 8

Tabla 4: Asignación PMOD



Pmod JA	Pmod JB	Pmod JC	Pmod XDAC
JA1: J1	JB1: A14	JC1: K17	JXADC1: J3
JA2: L2	JB2: A16	JC2: M18	JXADC2: L3
JA3: J2	JB3: B15	JC3: N17	JXADC3: M2
JA4: G2	JB4: B16	JC4: P18	JXADC4: N2
JA7: H1	JB7: A15	JC7: L17	JXADC7: K3
JA8: K2	JB8: A17	JC8: M19	JXADC8: M3
JA9: H2	JB9: C15	JC9: P17	JXADC9: M1
JA10: G3	JB10: C16	JC10: R18	JXADC10: N1

Figura 46: Puertos PMOD

Por último, se ejecuta el código, se implementa en la Basys y se realizan las conexiones entre el puerto GPIO de la Raspberry Pi y los puertos PMOD de la Basys según la siguiente tabla (*Tabla 5*):

Componente	Puerto GPIO	Puerto PMOD
Lámpara 7 segmentos seg.1	23	A14
Lámpara 7 segmentos seg.2	24	A16
Lámpara 7 segmentos seg.3	6	B15
Lámpara 7 segmentos seg.4	13	B16
Lámpara 7 segmentos seg.5	19	A15
Lámpara 7 segmentos seg.6	26	A17
Lámpara 7 segmentos seg.7	12	C15
Lámpara 7 segmentos Carácter 1	16	K17
Lámpara 7 segmentos Carácter 2	20	M18
Lámpara 7 segmentos Carácter 3	21	N17
Pulsador1	4	J1
Pulsador2	17	L2
Pulsador3	27	J2

Pulsador4	22	G2
Led1	10	H1
Led2	9	K2
Led3	11	H2
Led4	7	G3
Interruptor1	2	L17
Interruptor2	3	M19

Tabla 5: Relación puertos GPIO y PMOD

Realizadas las conexiones se comprueba que el funcionamiento del sistema es el correcto y cumple con los requisitos esperados.

5.2 PRUEBA DEL DISPOSITIVO OBJETIVO

Para comprobar que el funcionamiento es el esperado se inicia la interfaz de escritorio y se comprueba que la Basys se comporta como es debido.

En primer lugar, tras iniciar la placa se pulsan los botones referentes a los pulsadores y se comprueba que los leds de la Basys se iluminan en el orden correcto (*Figura 47*):

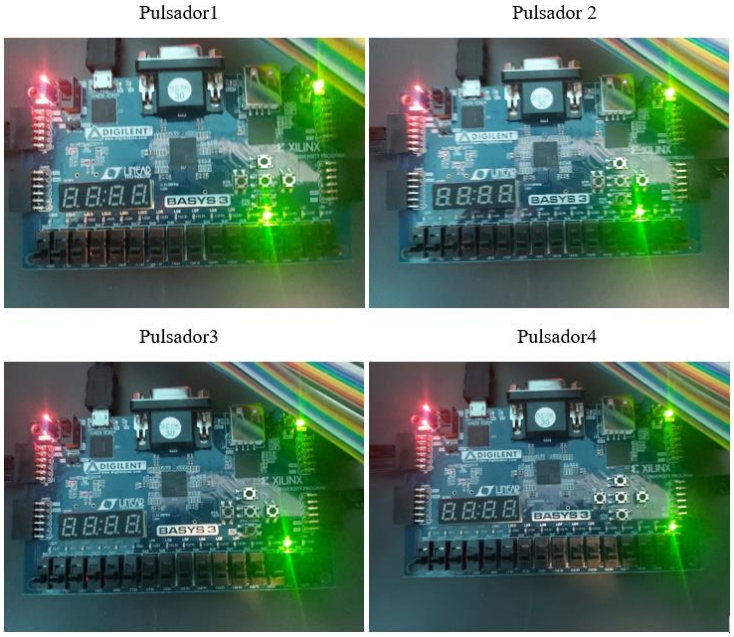


Figura 47: Comprobación pulsadores

Se comprueba que se ilumina correctamente el led correspondiente a cada pulsador.

A continuación, se comprueba que los interruptores tanto de la interfaz visual como los de la Basys encienden el carácter designado mostrando el número determinado para cada uno (*Figura 48*):

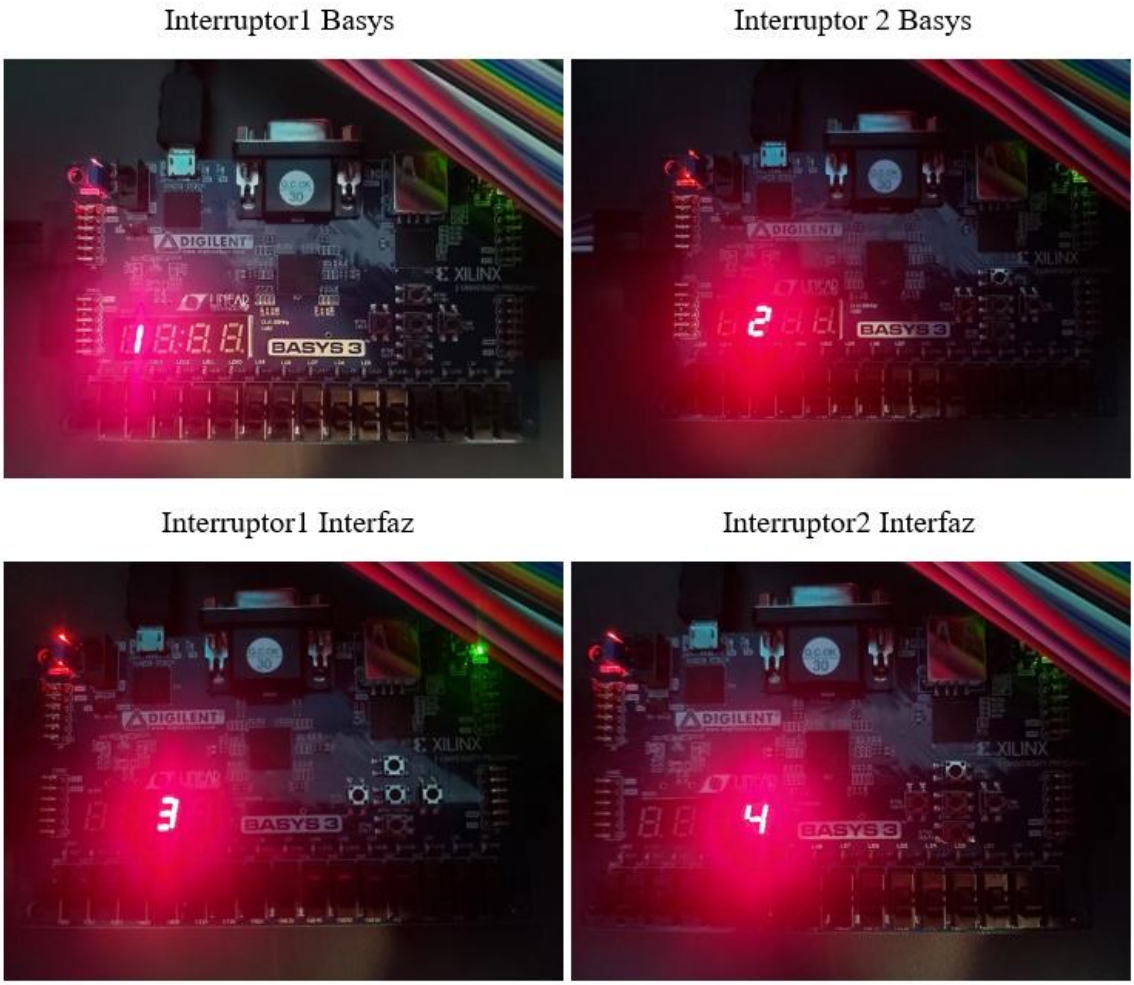


Figura 48: Comprobación interruptores

El resultado obtenido era el esperado siendo tanto la posición como los números mostrados los que se quería conseguir para cada interruptor.

La siguiente comprobación que se debe realizar es como al pulsar los botones de la Basys se iluminan los leds de la interfaz visual (*Figura 49*):

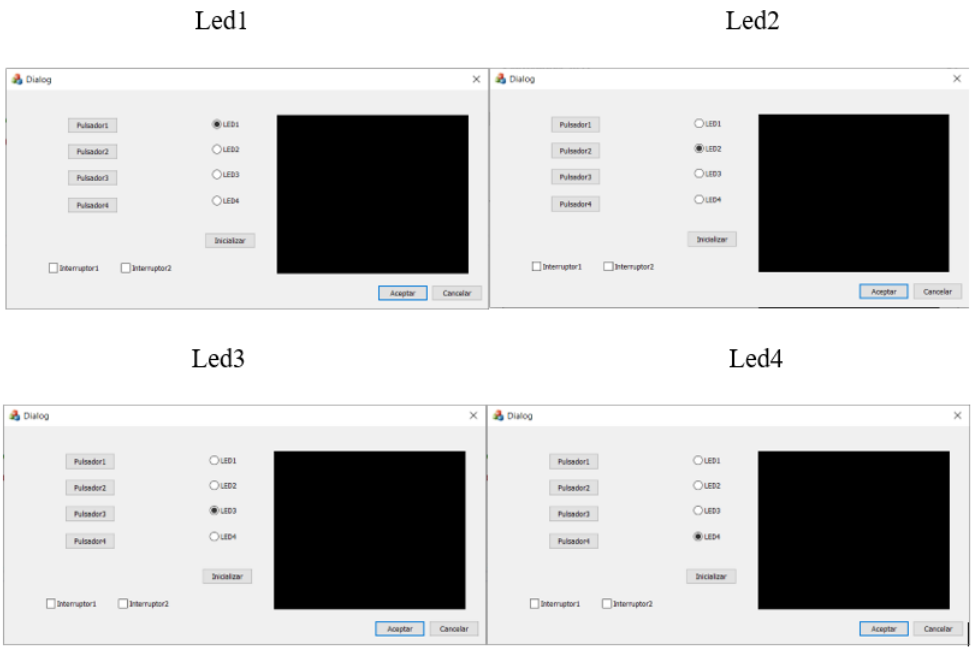


Figura 49: Comprobación leds

Por último, se comprueba que la ventana de la interfaz visual está mostrando la imagen en vivo capturada por la web cam conectada a la Raspberry Pi (*Figura 50*):

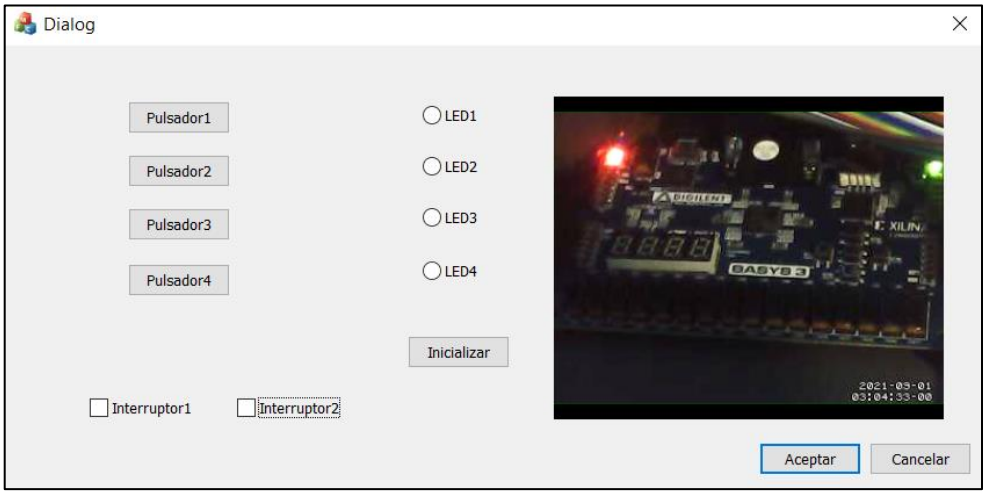


Figura 50: Comprobación cámara

Se compruebe que la transmisión funciona correctamente y que se puede observar la Basys remotamente.

6 CONCLUSIÓN

Los resultados obtenidos cumplen las expectativas, funcionando correctamente todas las partes del proyecto. La respuesta es rápida teniendo una repercusión inmediata desde que se manda una orden hasta que se ejecuta. La latencia recibida por la transmisión en vivo es más alta de lo esperado debido a que se utiliza como reproductor VLC Media Player el cual al conectar con una dirección IP produce un retraso apreciable de unos tres segundos aproximadamente.

El diseño permite que realizando pequeñas modificaciones en el código se adapte para aplicaciones distintas por lo que se puede utilizar el mismo diseño para realizar prácticas de laboratorio distintas con un simple cambio.

Los alumnos solo necesitarán descargar el software Visual Studio y la aplicación de escritorio para poder acceder al control remoto.

La interfaz visual resulta simple y muy intuitiva por lo que los alumnos pueden aprender rápidamente a utilizarla incluso sin necesidad de ayuda.

7 PROPUESTAS DE MEJORA FUTURAS

Existen múltiples mejoras de diseño que pueden mejorar el rendimiento del proyecto, algunas de las planteadas son las siguientes:

7.1 MEJORA LATENCIA CÁMARA

Actualmente la imagen en tiempo real se muestra utilizando el software VLC Media Player. Este reproductor al conectar con una IP produce una latencia que hace que la imagen se reproduzca en torno a tres segundos después de ser capturada. Una posible solución sería utilizar un reproductor distinto para conseguir una menor latencia.

7.2 CONTROL DE USUARIOS

Actualmente cualquiera se puede conectar a la placa y realizar modificaciones en ella, para evitar esto y tener un control más restringido y seguro de quien puede o no utilizar la placa y como se plantea introducir un control de usuarios en el cual se deba iniciar sesión con un usuario y contraseña antes de poder manipular nada.

7.3 CONTROL DE MÚLTIPLES DISPOSITIVOS

Actualmente el diseño esta implementado para conectar con un único servidor y aunque se puede modificar de manera simple a que servidor conectarse debe hacerse de forma manual. La propuesta para solventar este problema es crea un apartado en la interfaz gráfica que te permita añadir dispositivos de tal manera que te aparezcan todos los que tienes guardados y poder cambiar de uno a otro de manera simple.

8 BIBLIOGRAFIA

[1] *FPGA remote laboratory: experience of a shared laboratory between UPNA and UNIFESP*. (2020, 1 julio). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9163773 (accedido el 01-09-2021)

[2] Departamento de Ingeniería Electrónica, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid. http://oa.upm.es/4317/1/INVE_MEM_2008_59753.pdf (accedido el 01-09-2021)

[3] Colaboradores de Wikipedia. (2021, agosto 30). *Microsoft Visual Studio*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Microsoft_Visual_Studio (accedido el 10/8/2021)

[4] Notepad++. https://notepad-plus-plus.org/ (accedido el 12/08/2021)

[5] Colaboradores de Wikipedia. (2020, 10 abril). *VNC*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/VNC (accedido el 12/08/2021)

[6] Z. (2014, 28 marzo). *¿Qué es PuTTY y para qué sirve?* Vozidea.com. https://www.vozidea.com/que-es-putty-y-para-que-sirve (accedido el 12/08/2020)

[7] *Descarga oficial del Reproductor multimedia VLC, el mejor reproductor de código abierto - VideoLAN*. https://www.videolan.org/vlc/index.es.html (accedido el 02-09-2021)

[8] Colaboradores de Wikipedia. (2021, junio 5). *Raspberry Pi*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Raspberry_Pi (accedido el 10/8/2021)

[9] The Raspberry Pi Foundation. (446445). *Buy a 4 Model B –*. Raspberry Pi. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ (accedido el 10/8/2021)

[10] Colaboradores de Wikipedia. (2021, junio 5). *Raspberry Pi*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Raspberry_Pi (accedido el 10/8/2021)

[11] Colaboradores de Wikipedia. (2021, julio 8). *Raspberry Pi OS*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Raspberry_Pi_OS (accedido el 12/8/2021)

[12] Xilinx. https://www.mouser.es/new/xilinx/xilinx-vivado-design-suite-hlx/ (accedido el 30/8/2021)

[13] *Basys 3 Artix-7 FPGA Board*. Xilinx. https://www.xilinx.com/products/boards-and-kits/1-54wqge.html (accedido el 02-09-2021)

Documento Nº2: Anexos

ÍNDICE

1. CÓDIGO EMPLEADO

En este apartado se presentan los códigos más relevantes empleados en el desarrollo del proyecto:

1.1. GPIO.C

Programa albergado por la Raspberry Pi con las funciones necesarias para abrir y cerrar puertos GPIO, así como las funciones necesarias para leer y escribir en los puertos GPIO:

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "gpio.h"

#define LOW  0
#define HIGH 1

int
GPIOExport(int pin)
{
#define BUFFER_MAX 3
	char buffer[BUFFER_MAX];
	ssize_t bytes_written;
	int fd;

	fd = open("/sys/class/gpio/export", O_WRONLY);
	if (-1 == fd) {
		fprintf(stderr, "Failed to open export for writing!\n");
		return(-1);
	}

	bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
	write(fd, buffer, bytes_written);
	close(fd);
	return(0);
}

int
GPIOUnexport(int pin)
{
	char buffer[BUFFER_MAX];
	ssize_t bytes_written;
```

```
	int fd;

	fd = open("/sys/class/gpio/unexport", O_WRONLY);
	if (-1 == fd) {
		fprintf(stderr, "Failed to open unexport for writing!\n");
		return(-1);
	}

	bytes_written = snprintf(buffer, BUFFER_MAX, "%d", pin);
	write(fd, buffer, bytes_written);
	close(fd);
	return(0);
}

int
GPIODirection(int pin, int dir)
{
	static const char s_directions_str[] = "in\0out";

#define DIRECTION_MAX 35
	char path[DIRECTION_MAX];
	int fd;

	snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/direction", pin);
	fd = open(path, O_WRONLY);
	if (-1 == fd) {
		fprintf(stderr, "Failed to open gpio %d direction for writing!\n",
pin);
		return(-1);
	}

	if (-1 == write(fd, &s_directions_str[IN == dir ? 0 : 3], IN == dir ? 2 : 3))
{
		fprintf(stderr, "Failed to set direction!\n");
		return(-1);
	}

	close(fd);
	return(0);
}

int
GPIORead(int pin)
{
#define VALUE_MAX 30
	char path[VALUE_MAX];
	char value_str[3];
	int fd;

	snprintf(path, VALUE_MAX, "/sys/class/gpio/gpio%d/value", pin);
	fd = open(path, O_RDONLY);
	if (-1 == fd) {
		fprintf(stderr, "Failed to open gpio %d value for reading!\n", pin);
		return(-1);
	}

	if (-1 == read(fd, value_str, 3)) {
		fprintf(stderr, "Failed to read value!\n");
		return(-1);
	}

	close(fd);
```

```
	return(atoi(value_str));
}
int
GPIOWrite(int pin, int value)
{
	static const char s_values_str[] = "01";

	char path[VALUE_MAX];
	int fd;

	snprintf(path, VALUE_MAX, "/sys/class/gpio/gpio%d/value", pin);
	fd = open(path, O_WRONLY);
	if (-1 == fd) {
		fprintf(stderr, "Failed to open gpio value for writing!\n");
		return(-1);
	}

	if (1 != write(fd, &s_values_str[LOW == value ? 0 : 1], 1)) {
		fprintf(stderr, "Failed to write value!\n");
		return(-1);
	}

	close(fd);
	return(0);
}
```

1.2. TCPSERVER.C

Este código es el código principal que alberga la Raspberry Pi, en él se encuentran las funciones que inicializan la placa, leen la lampara de siete segmentos, simulan los botones, leds e interruptores y realiza la función de servidor TCP recibiendo los datos enviados por el cliente, analizándolos y realizando la función necesaria:

```
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <ctype.h>
//------------------
#include <time.h>
```

```
#include <sched.h>


#include "gpio.h"

#include "uart.h"

//------------------


#define MAX 80

#define PORT 2345

#define SA struct sockaddr

#define MAX_RETRY 100

//-------------------------------------------------------------------------------
-----------

char input[] = { 14,23,24,6,13,19,26,12,16,20,21 };
char output[] = { 2,3,4,17,27,22,10,9,11,7,15 };

int initialize() {

	int i;
	system("sudo djtgcfg prog -d Basys2 -i 0 -f
/home/pi/digilent/download2.bit");
	clear_fifo();
	for (i = 0; i < sizeof(input); i++) {
		if (-1 == GPIOExport(input[i])) {
			printf("Unable to open GPIO %d\n", input[i]);
			return(-1);
		}
		if (-1 == GPIODirection(input[i], IN)) {
			printf("unable to set GPIO %d as input\n", input[i]);
			return(-1);
		}
	}

	for (i = 0; i < sizeof(output); i++) {
		if (-1 == GPIOExport(output[i])) {
			printf("Unable to open GPIO %d\n", output[i]);
			return(-1);
		}
		if (-1 == GPIODirection(output[i], OUT)) {
			printf("Unable to set GPIO %d as output\n", output[i]);
			return(-1);
		}
	}
	printf("Initialization done. GPIO: %d inputs, %d outputs\n", sizeof(input),
sizeof(output));
	return 0;
}


int convert_display(int num) {
	int v = 0;

	if ((num & 1) == 0) v |= 0x40;
	if ((num & 2) == 0) v |= 0x20;
	if ((num & 4) == 0) v |= 0x10;
```

```
	if ((num & 8) == 0) v |= 0x8;
	if ((num & 0x10) == 0) v |= 0x4;
	if ((num & 0x20) == 0) v |= 0x2;
	if ((num & 0x40) == 0) v |= 0x1;
	//    return 0x1d;
	return v;
	/*
		switch(num){
			case 0x1: return '0';
			case 0x4f: return '1';
			case 0x12: return '2';
			case 0x6: return '3';
			case 0x4c: return '4';
			case 0x24: return '5';
			case 0x20: return '6';
			case 0xF: return '7';
			case 0x0: return '8';
			case 0xC: return '9';
			case 0x7E: return '-';
		}
		return 'E';*/
}


int read_caracter() {
	int i, val;
	int seg_0, seg_1, seg_2, seg_3, seg_4, seg_5, seg_6;
	for (i = 0; i < 20; i++) {
		seg_0 = GPIORead(23); seg_1 = GPIORead(24); seg_2 = GPIORead(6); seg_3
= GPIORead(13);
		seg_4 = GPIORead(19); seg_5 = GPIORead(26); seg_6 = GPIORead(12);
		val = (seg_0 << 6) | (seg_1 << 5) | (seg_2 << 4) | (seg_3 << 3) |
(seg_4 << 2) | (seg_5 << 1) | seg_6;
		printf("Detected %X %c \n", val, convert_display(val));
		printf("Detected %d%d%d%d%d%d%d \n", seg_0, seg_1, seg_2, seg_3,
seg_4, seg_5, seg_6);
		usleep(300);
	}
	return 0;
}


int read_display() {
	int i, val;
	int values[3] = { 0,0,0 };
	int an_0 = 0, an_1 = 0, an_2 = 0, an_01 = 0, an_11 = 0, an_21 = 0;
	int seg_0, seg_1, seg_2, seg_3, seg_4, seg_5, seg_6;
	for (i = 0; i < 20; i++) {
		an_0 = GPIORead(16);	an_1 = GPIORead(20);	an_2 =
GPIORead(21);
		seg_0 = GPIORead(23); seg_1 = GPIORead(24); seg_2 = GPIORead(6); seg_3
= GPIORead(13);
		seg_4 = GPIORead(19); seg_5 = GPIORead(26); seg_6 = GPIORead(12);
		an_01 = GPIORead(16);	an_11 = GPIORead(20);	an_21 =
GPIORead(21);
		val = (seg_0 << 6) | (seg_1 << 5) | (seg_2 << 4) | (seg_3 << 3) |
(seg_4 << 2) | (seg_5 << 1) | seg_6;
		if (!an_0 && !an_01) values[0] = val;
		if (!an_1 && !an_11) values[1] = val;
		if (!an_2 && !an_21) values[2] = val;
		usleep(300);
	}
```

```
	printf("%02X%02X%02X \n", convert_display(values[2]),
convert_display(values[1]), convert_display(values[0]));
	return 0;
}

int button(int numbut) {
	static const int id[4] = { 4,17,27,22 };
	GPIOWrite(id[numbut], 1);
	usleep(300 * 1000);
	return GPIOWrite(id[numbut], 0);
}

int led(int numled) {
	static const int id[4] = { 10,9,11,7 };
	GPIOWrite(id[numled], 1);
	usleep(300 * 1000);
	return GPIOWrite(id[numled], 0);
}


//---------------------------------------------
// int switches(int val){
	// return GPIOWrite(4, val);
// }
//---------------------------------------------

int switches(int numSwt, int val) {
	static const int id[2] = { 2,3 };
	return GPIOWrite(id[numSwt], val);
}

int set_prio() {
	struct sched_param p;
	p.sched_priority = sched_get_priority_max(SCHED_FIFO);
	return sched_setscheduler(0, SCHED_FIFO, &p);
}


//-------------------------------------------------------------------------------------
-------------

char* trim(char* cadena) {
	char* ptr = cadena;
	int len = strlen(cadena);
	char* end = cadena + len;
	while (isspace(*ptr) && ptr < end) ptr++;
	char* back = end - 1;
	while (isspace(*back) && back >= cadena) {
		back[0] = '\0';
		back--;
	}
	return cadena;
}
char* argv[10];//cadena;


char** ReadData(char* cadena, int* argc_ptr) {
	int argc = 0, num = 0;
	char delimitador[] = " ";
	cadena = trim(cadena);
	char* tmp = trim(cadena);

	while ((tmp = strchr(tmp, ' ')) != NULL) {
```

```
                tmp++;
                argc++;
        }
        if (argc > 9) argc = 9;
        //argv = (char**)malloc(argc*sizeof(char*));

        char* token = strtok(cadena, delimitador);
        while (token != NULL) {
                if (num > argc) { perror("More arrays found in 'cadena' than
expected"); }
                // Sólo en la primera se pasa la cadena; en las siguientes se pasa
NULL
                //printf("Token: %s\n", token);
                argv[num++] = token;
                token = strtok(NULL, delimitador);
                // Añadir cada uno de los valores leidos en columnas
//          rbind(argv[],token);
        }
        *argc_ptr = num;
        return argv;
}


//-------------------------------------------------------------------------------
-------------

int Control(char* cadena/*int argc, char *argv[]*/) {
        int repeat = 10;
        int i, argc;
        char** argv;

        argv = ReadData(cadena, &argc);

        if (argc < 2) {
                printf("Nothing to do. Add:\n 'i' for initializing,\n 'd' for reading
7 semgnet display,\n 'b' and a number (0,1,2) for pressing a a button,\n 'l' and a
number (0,1,2) to turn on a led,\n 's', the number of the switch (0,1) and a value
(0/1) to change the switch\n 'r' for reading from the serial port\n 'w' writing to
the serial port\n 'n' to start a new access (clear que serial queue)\n 'e' for
exiting\n");
                return -1;
        }
        //    printf(" Received: %d %s %s\n",argc, argv[0],argv[1]);

        if (argv[1][0] == 'i') {
                return initialize();
        }
        if (argv[1][0] == 'd') {
                set_prio();

                return read_display();
        }
        if (argv[1][0] == 'b') {
                printf("enciendo boton %s\n", argv[2]);
                return  button(atoi(argv[2]));
        }
        if (argv[1][0] == 'l') {
        printf("leyendo led %s\n", argv[2]);
        return  led(atoi(argv[2]));
        }
        if (argv[1][0] == 's') {
                return switches(atoi(argv[2]), atoi(argv[3]));
        }
```

```
	if (argv[1][0] == 'c') {
		return read_caracter();
	}
	if (argv[1][0] == 'n') {
		return clear_fifo();
	}
	if (argv[1][0] == 'r') {
		FILE* fp = popen("ps -ef | grep uart.x | grep 'grep uart' -v | grep
	sudo -v | wc -l", "r");
		char* ptr = fgets(buf, 100, fp);
		pclose(fp);
		int num = atoi(ptr);
		if (num < 2) {
			system("sudo /home/pi/digilent/listen7s/uart.x > /dev/null &");
		}
		return read_uart();
	}
	if (argv[1][0] == 'w') {
		return write_uart(argv[2], strlen(argv[2]));
	}
	if (argv[1][0] == 'e') {
		system("killall uart.x");
		system("echo 'closing' > /home/pi/tmp.txt");
	}

		printf("Nothing to do. Add 'i' for initializing, 'd' for reading 7
		semgnet display, 'b' and a number (0,1,2) for pressing a a button, and
		's' and a value (0/1) to change the switch\n");
		return -1;
}



//-------------------------------------------------------------------------------
-----------

// Function designed for chat between client and server.
void* func(void* arg)
{
	int sockfd = (int)arg;
	char buff[MAX] = "Connection ok\n\r";
	int n, len;

	write(sockfd, buff, strlen(buff) + 1);

	// infinite loop for chat
	for (;;) {
		bzero(buff, MAX);

		// read the message from client and copy it in buffer
		len = read(sockfd, buff, sizeof(buff));
		buff[len] = 0;
		printf("Readed message of size %d %s\n", len, buff);

		//-----------------------------------------------------------------
--
		Control(buff);
		//-----------------------------------------------------------------
--

		// print buffer which contains the client contents
		printf("From client: %s\t To client : ", buff);
		bzero(buff, MAX);
```

```
            n = 0;
            // copy server message in the buffer
  //      while ((buff[n++] = getchar()) != '\n');
  //      buff[n] = 0;
  //      printf("Readed message of size %d from console: %s\n",n, buff);
            strcpy(buff, "done\n");
            // and send that buffer to client
            write(sockfd, buff, strlen(buff) + 1);

            // if msg contains "Exit" then server exit and chat ended.
            if (strncmp("exit", buff, 4) == 0) {
                  printf("Server Exit...\n");
                  break;
            }
      }
}


// Driver function
int main()
{
      int sockfd, connfd, len, i;
      struct sockaddr_in servaddr, cli;

      // socket create and verification
      sockfd = socket(AF_INET, SOCK_STREAM, 0);
      if (sockfd == -1) {
            printf("socket creation failed...\n");
            exit(0);
      }
      else
            printf("Socket successfully created..\n Binding ");
      bzero(&servaddr, sizeof(servaddr));

      // assign IP, PORT
      servaddr.sin_family = AF_INET;
      servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
      servaddr.sin_port = htons(PORT);

      for (i = 0; i < MAX_RETRY; i++) {

            // Binding newly created socket to given IP and verification
            if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
                  printf(".");
                  fflush(stdout);
                  sleep(1);
            }
            else {
                  printf("\nSocket successfully binded..\n");
                  break;
            }
      }
      if (i == MAX_RETRY) {
            perror("socket bind failed...\n");
            exit(-1);
      }

      // Now server is ready to listen and verification
      if ((listen(sockfd, 5)) != 0) {
            printf("Listen failed...\n");
            exit(0);
      }
      else
```

```
		printf("Server listening..\n");
	len = sizeof(cli);

	while (1) {
		pthread_t thread_id;
		// Accept the data packet from client and verification
		connfd = accept(sockfd, (SA*)&cli, &len);
		if (connfd < 0) {
			printf("server acccept failed...\n");
			exit(0);
		}
		else
			printf("server acccept the client...\n");

		// Function for chatting between client and server
		pthread_create(&thread_id, NULL, func, (void*)connfd);

	}
	// After chatting close the socket
	close(sockfd);
	}
```

1.3. TCPCLIENT.CCP

Programa de Visual Studio que cumple la función que se encarga de realizar la conexión con el servidor TCP:

```
#define WIN32_LEAN_AND_MEAN

//#include "stdafx.h"
#include "pch.h"

#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include "tcpClient.h"

// Need to link with Ws2_32.lib, Mswsock.lib, and Advapi32.lib
#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")


#define DEFAULT_BUFLEN 512

SOCKET ConnectSocket = INVALID_SOCKET;

SOCKET getSocket() {
	return ConnectSocket;
}

int connectTCP(CString addr, CString port)
{
	WSADATA wsaData;
```

```
	struct addrinfo *result = NULL,
		*ptr = NULL,
		hints;
	int iResult;

//	CMyProgressBarTaskDialog dlg;
//	dlg.DoModal();
//	dlg.ShowDialog(_T("Hola"),_T("buenas"),_T("que tal"),0,0,TDCBF_CLOSE_BUTTON);

	// Initialize Winsock
	iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
	if (iResult != 0) {
		printf("WSAStartup failed with error: %d\n", iResult);
		return 1;
	}

	ZeroMemory(&hints, sizeof(hints));
	hints.ai_family = AF_UNSPEC;
	hints.ai_socktype = SOCK_STREAM;
	hints.ai_protocol = IPPROTO_TCP;

	// Resolve the server address and port
	CStringA strA(addr);
	CStringA strA2(port);
	LPCSTR addrStr = strA;
	LPCSTR portStr = strA2;
	iResult = getaddrinfo(addrStr, portStr, &hints, &result);
	if (iResult != 0) {
		printf("getaddrinfo failed with error: %d\n", iResult);
		WSACleanup();
		return 1;
	}

	// Attempt to connect to an address until one succeeds
	for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {

		// Create a SOCKET for connecting to server
		ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
			ptr->ai_protocol);
		if (ConnectSocket == INVALID_SOCKET) {
			printf("socket failed with error: %ld\n", WSAGetLastError());
			WSACleanup();
			return 1;
		}

		// Connect to server.
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
		if (iResult == SOCKET_ERROR) {
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
			continue;
		}
		break;
	}

	freeaddrinfo(result);

	if (ConnectSocket == INVALID_SOCKET) {
		printf("Unable to connect to server!\n");
		WSACleanup();
		return 1;
	}
```

```
	return 0;
}

int communicateTCP(char *sendbuf, UINT sendlen, char *recbuf, UINT reclen) {
	int iResult, iResult2;


	// Receive until the peer closes the connection

	iResult = recv(ConnectSocket, recbuf, reclen, 0);
	if (iResult > 0)
		printf("Bytes received: %d\n", iResult);
	else if (iResult == 0)
		printf("Connection closed\n");
	else
		printf("recv failed with error: %d\n", WSAGetLastError());

	// Send an initial buffer
	iResult2 = send(ConnectSocket, sendbuf, sendlen, 0);
	if (iResult2 == SOCKET_ERROR) {
		printf("send failed with error: %d\n", WSAGetLastError());
		closesocket(ConnectSocket);
		WSACleanup();
		return 1;
	}

	printf("Bytes Sent: %ld\n", iResult);


		return iResult;
}

int closeTCP(){
	// cleanup
	closesocket(ConnectSocket);
	WSACleanup();

	return 0;
}
```

1.4. MENU_INICIO.CCP

Programa de Visual Studio que se encarga de conectar con la IP del servidor utilizando la función tcpClient.ccp y en función de que botones se pulsan en la interfaz visual se encarga de mandar la orden necesaria:

```
// menu_inicio.cpp: archivo de implementación
//

#include "pch.h"
#include "MFCApplication1.h"
#include <string>
#include <thread>
#include <chrono>
```

```
#include "menu_inicio.h"
#include "afxdialogex.h"
#include "tcpClient.h"


// Cuadro de diálogo de menu_inicio

IMPLEMENT_DYNAMIC(menu_inicio, CDialogEx)

menu_inicio::menu_inicio(CWnd* pParent /*= nullptr*/)
	: CDialogEx(MenuDeInicio, pParent), ledButton(NULL)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	ledButton = NULL;
	//if (connectTCP(CString("127.0.0.1"), CString("5000")) == 0) {
//	if (connectTCP(CString("169.254.125.244"), CString("2345")) == 0) {
		ConnectSocket = getSocket();
//	}
}

DWORD WINAPI  menu_inicio::TimerThread(LPVOID arg) {
	static int value = 0;
	while (1) {
		std::this_thread::sleep_for(std::chrono::milliseconds(500) );
/*		ledButton.inv
		if (ledButton != NULL) {
			Label.Invoke((MethodInvoker)delegate {
				// Running on the UI thread
*/				ledButton->SetState(value);
				value = !value;
//			});
//		}
	}
}



menu_inicio::~menu_inicio()
{
}


void menu_inicio::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_VLCPLUGIN1, m_vlcVideo);
}


BEGIN_MESSAGE_MAP(menu_inicio, CDialogEx)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_TIMER()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDOK, &menu_inicio::OnBnClickedOk)
	ON_BN_CLICKED(IDC_BUTTON5, &menu_inicio::OnBnClickedButton5)
	ON_BN_CLICKED(IDC_CHECK1, &menu_inicio::OnBnClickedCheck1)
	ON_BN_CLICKED(IDC_CHECK2, &menu_inicio::OnBnClickedCheck2)
	ON_BN_CLICKED(IDC_BUTTON1, &menu_inicio::OnBnClickedButton1)
	ON_BN_CLICKED(IDC_BUTTON2, &menu_inicio::OnBnClickedButton2)
	ON_BN_CLICKED(IDC_BUTTON3, &menu_inicio::OnBnClickedButton3)
	ON_BN_CLICKED(IDC_BUTTON4, &menu_inicio::OnBnClickedButton4)
```

```
	ON_BN_CLICKED(IDC_RADIO1, &menu_inicio::OnBnClickedRadio1)
	ON_BN_CLICKED(IDC_RADIO2, &menu_inicio::OnBnClickedRadio2)
	ON_BN_CLICKED(IDC_RADIO3, &menu_inicio::OnBnClickedRadio3)
	ON_BN_CLICKED(IDC_RADIO4, &menu_inicio::OnBnClickedRadio4)
END_MESSAGE_MAP()


// Controladores de mensajes de menu_inicio

// Controladores de mensajes de CMFCApplication1Dlg

HANDLE gDoneEvent;

VOID CALLBACK TimerRoutine(PVOID lpParam, BOOLEAN TimerOrWaitFired)
{
	static int count = 0;
	count++;
	if (count == 100)
	{
		SetEvent(gDoneEvent);
	}
	if (lpParam == NULL)
	{
		printf("TimerRoutine lpParam is NULL\n");
	}
	else
	{
		// lpParam points to the argument; in this case it is an int

		printf("Timer routine called. Parameter is %d.\n",
			*(int*)lpParam);
		if (TimerOrWaitFired)
		{
			printf("The wait timed out.\n");
		}
		else
		{
			printf("The wait event was signaled.\n");
		}
	}

	SetEvent(gDoneEvent);
}


BOOL menu_inicio::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Agregar el elemento de menú "Acerca de..." al menú del sistema.

	// IDM_ABOUTBOX debe estar en el intervalo de comandos del sistema.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != nullptr)
	{
		BOOL bNameValid;
		CString strAboutMenu;
		bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
		ASSERT(bNameValid);
```

```
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Establecer el icono para este cuadro de diálogo.  El marco de trabajo
realiza esta operación
	//  automáticamente cuando la ventana principal de la aplicación no es un
cuadro de diálogo
	SetIcon(m_hIcon, TRUE);			// Establecer icono grande
	SetIcon(m_hIcon, FALSE);		// Establecer icono pequeño

	// TODO: agregar aquí inicialización adicional
	ledButton = ((CButton*)GetDlgItem(IDC_RADIO1));
	//std::thread timerThread(&menu_inicio::TimerThread, this);

	//::AfxBeginThread(&menu_inicio::TimerThread, NULL);

	SetTimer(1, 500, NULL);


#if 0
	HANDLE hTimer = NULL;
	HANDLE hTimerQueue = NULL;
	int arg = 123, x;
	hTimerQueue = CreateTimerQueue();

	// Use an event object to track the TimerRoutine execution
	gDoneEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
	if (NULL == gDoneEvent)
	{
		printf("CreateEvent failed (%d)\n", GetLastError());
		return 1;
	}

	if (NULL == hTimerQueue)
	{
		printf("CreateTimerQueue failed (%d)\n", GetLastError());
		return 2;
	}

	// Set a timer to call the timer routine in 10 seconds.
	if (!CreateTimerQueueTimer(&hTimer, hTimerQueue,
(WAITORTIMERCALLBACK)TimerRoutine, &arg, 50, 100, 0))
	{
		printf("CreateTimerQueueTimer failed (%d)\n", GetLastError());
		return 3;
	}
#endif

	return TRUE;  // Devuelve TRUE  a menos que establezca el foco en un control
}

void menu_inicio::OnTimer(UINT nID) {
	static int value = 0;
	ledButton->SetCheck(value);
	value = !value;

	UpdateData(false);
	CDialog::OnTimer(nID);
```

```
}

void menu_inicio::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
//		CAboutDlg dlgAbout;
//		dlgAbout.DoModal();
	}
	else
	{
		CDialogEx::OnSysCommand(nID, lParam);
	}
}

// Si agrega un botón Minimizar al cuadro de diálogo, necesitará el siguiente código
//  para dibujar el icono.  Para aplicaciones MFC que utilicen el modelo de
documentos y vistas,
//  esta operación la realiza automáticamente el marco de trabajo.

void menu_inicio::OnPaint()
{

	if (IsIconic())
	{
		CPaintDC dc(this); // Contexto de dispositivo para dibujo

		SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

		// Centrar icono en el rectángulo de cliente
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Dibujar el icono
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialogEx::OnPaint();
	}
}

// El sistema llama a esta función para obtener el cursor que se muestra mientras el
usuario arrastra
//  la ventana minimizada.
HCURSOR menu_inicio::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}

#include "tcpClient.h"

void menu_inicio::OnBnClickedOk()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	CDialogEx::OnOK();
}
```

```
// Enviar datos correspondientes a la funcion que debe realizar cada boton
void menu_inicio::sendData(const char* txt)
{
	int len = strlen(txt) + 1;
	// TODO: Agregue aquí su código de controlador de notificación de control
	//Inicializar la placa
	int ret = send(ConnectSocket, txt, len, 0);
	if (ret != len) {
		printf("Error: Data send error. Send: %d Return: %d \n", len, ret);
	}
}



// Boton de inicializar
void menu_inicio::OnBnClickedButton5()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	sendData("DontCare i");
	ledButton->SetState(1);
	ledButton->SetCheck(1);
}

// Interruptor 1
void menu_inicio::OnBnClickedCheck1()
{
	int status = ((CButton*)GetDlgItem(IDC_CHECK1))->GetCheck();
	// TODO: Agregue aquí su código de controlador de notificación de control
	printf("Boton: state %d\n", status);
	if (status == 1) {
		sendData("DontCare s 0 1");
	}
	if (status == 0) {
		sendData("DontCare s 0 0");
	}
}

// Interruptor 2
void menu_inicio::OnBnClickedCheck2()
{
	int status = ((CButton*)GetDlgItem(IDC_CHECK2))->GetCheck();
	// TODO: Agregue aquí su código de controlador de notificación de control
	printf("Boton: state %d\n", status);
	if (status == 1) {
		sendData("DontCare s 1 1");
	}
	if (status == 0) {
		sendData("DontCare s 1 0");
	}
}


// Pulsador 1
void menu_inicio::OnBnClickedButton1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	sendData("DontCare b 0");
}

// Pulsador 2
void menu_inicio::OnBnClickedButton2()
```

```
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	sendData("DontCare b 1");
}

// Pulsador 3
void menu_inicio::OnBnClickedButton3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	sendData("DontCare b 2");
}

// Pulsador 4
void menu_inicio::OnBnClickedButton4()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//---------------------------------
	sendData("DontCare b 3");
	//---------------------------------
}


// Led 1
void menu_inicio::OnBnClickedRadio1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	sendData("DontCare l 0");

}

// Led 2
void menu_inicio::OnBnClickedRadio2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	sendData("DontCare l 1");
}

// Led 3
void menu_inicio::OnBnClickedRadio3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	sendData("DontCare l 2");
}

// Led 4
void menu_inicio::OnBnClickedRadio4()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//---------------------------
	sendData("DontCare l 3");
	//---------------------------
}




void menu_inicio::OnAcnStartAnimate1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
}
BEGIN_EVENTSINK_MAP(menu_inicio, CDialogEx)
```

```
END_EVENTSINK_MAP()
```

1.5. TFG.VHD

Código generado en Vivado para realizar el control de la Basys 3:

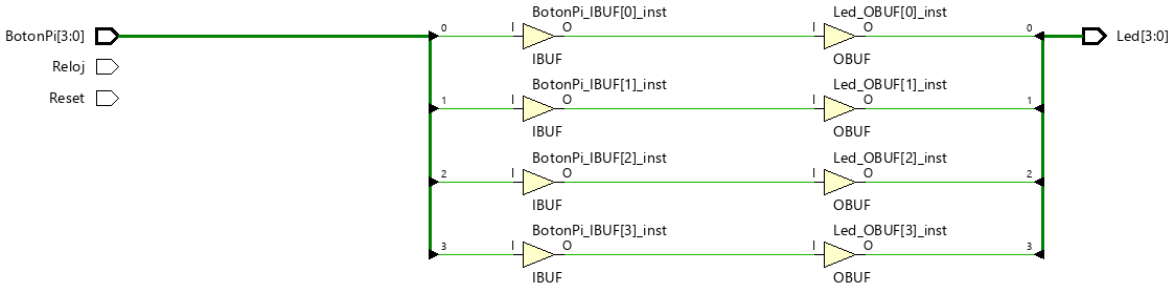
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Tfg is
    Port ( Boton : in STD_LOGIC_VECTOR (3 downto 0);
           BotonPi : in STD_LOGIC_VECTOR (3 downto 0);
           Led : out STD_LOGIC_VECTOR (3 downto 0);
           LedPi : out STD_LOGIC_VECTOR (3 downto 0);
           Interruptor : in STD_LOGIC_VECTOR (1 downto 0);
           InterruptorPi : in STD_LOGIC_VECTOR (1 downto 0);
           Segmentos : out STD_LOGIC_VECTOR (7 downto 0);
           SegmentosPi : out STD_LOGIC_VECTOR (7 downto 0);
           Caracteres : out STD_LOGIC_VECTOR (3 downto 0);
           CaracteresPi : out STD_LOGIC_VECTOR (3 downto 0);
           Reloj : in STD_LOGIC;
           Reset : in STD_LOGIC);
end Tfg;

architecture Behavioral of Tfg is

begin
```

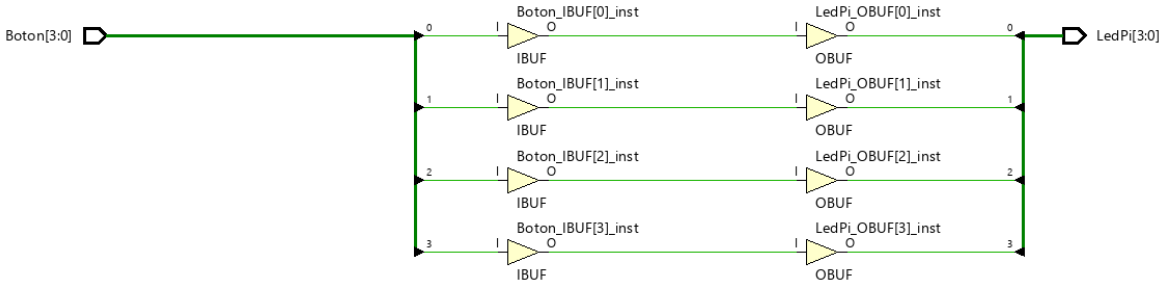
```
    LedPi<=Boton;
    Led<=BotonPi;
    Caracteres<=not(Interruptor&InterruptorPi);
    CaracteresPi<=not(Interruptor&InterruptorPi);

    process (Interruptor, InterruptorPi) begin
      if(Interruptor(0)='1') then
        Segmentos <= "00100101";
        SegmentosPi <= "00100101";
      elsif (Interruptor(1)='1') then
        Segmentos <= "10011111";
        SegmentosPi <= "10011111";
      elsif (InterruptorPi(0)='1') then
        Segmentos <= "10011001";
        SegmentosPi <= "10011001";
      else
        Segmentos <= "00001101";
        SegmentosPi <= "00001101";
      end if;
    end process;
end Behavioral;
```

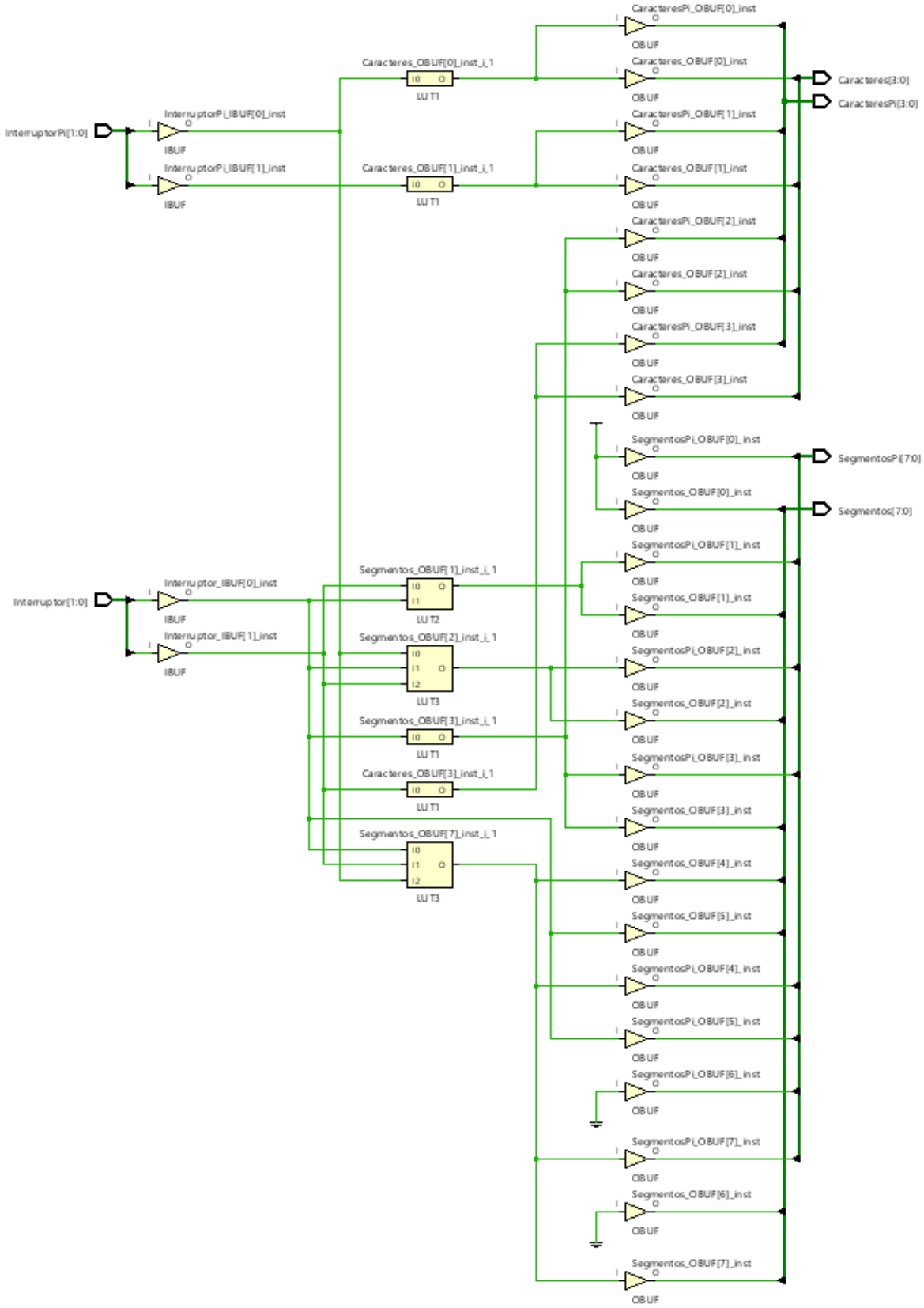
2. PLANOS CONEXIONES

2.1. ESQUEMÁTICO BASYS 3

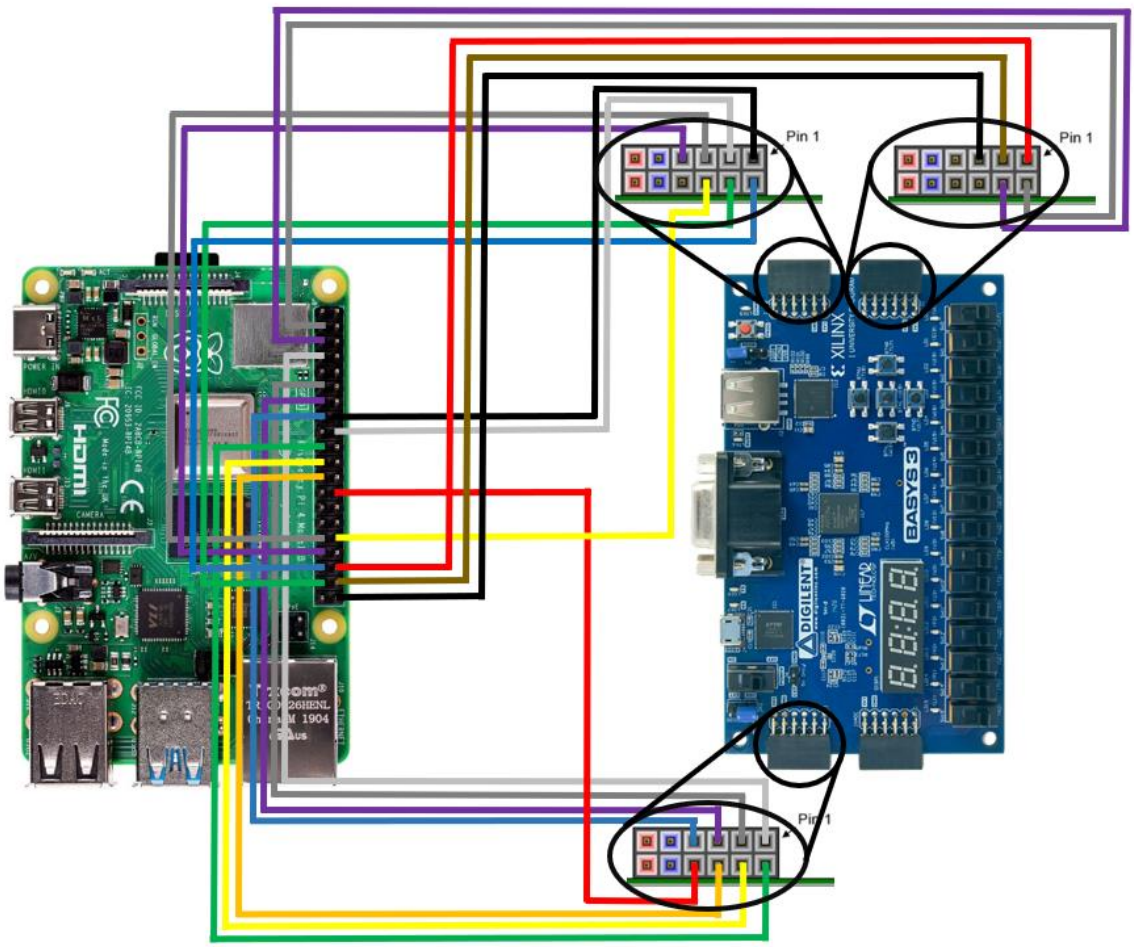
Esquemático con las conexiones que se forman en la Basys 3 para que cada puerto I/O realice la tarea desempeñada:






InterruptorPi[1:0]
InterruptorPi_IBUF[0]_inst
InterruptorPi_IBUF[1]_inst
IBUF
Caracteres_OBUF[0]_inst_i_1
Caracteres_OBUF[1]_inst_i_1
LUT1
CaracteresPi_OBUF[0]_inst
Caracteres_OBUF[0]_inst
CaracteresPi_OBUF[1]_inst
Caracteres_OBUF[1]_inst
CaracteresPi_OBUF[2]_inst
Caracteres_OBUF[2]_inst
CaracteresPi_OBUF[3]_inst
Caracteres_OBUF[3]_inst
OBUF
Caracteres[3:0]
CaracteresPi[3:0]
Interruptor[1:0]
Interruptor_IBUF[0]_inst
Interruptor_IBUF[1]_inst
Segmentos_OBUF[1]_inst_i_1
Segmentos_OBUF[2]_inst_i_1
Segmentos_OBUF[3]_inst_i_1
Caracteres_OBUF[3]_inst_i_1
Segmentos_OBUF[7]_inst_i_1
LUT2
LUT3
SegmentosPi_OBUF[0]_inst
Segmentos_OBUF[0]_inst
SegmentosPi_OBUF[1]_inst
Segmentos_OBUF[1]_inst
SegmentosPi_OBUF[2]_inst
Segmentos_OBUF[2]_inst
SegmentosPi_OBUF[3]_inst
Segmentos_OBUF[3]_inst
Segmentos_OBUF[4]_inst
Segmentos_OBUF[5]_inst
SegmentosPi_OBUF[4]_inst
SegmentosPi_OBUF[5]_inst
SegmentosPi_OBUF[6]_inst
SegmentosPi_OBUF[7]_inst
Segmentos_OBUF[6]_inst
Segmentos_OBUF[7]_inst
SegmentosPi[7:0]
Segmentos[7:0]

2.2. PLANO DE CONEXIONES PUERTOS GPIO Y PMOD



Documento Nº3: Pliego de condiciones

ÍNDICE

1. PLIEGO DE CONDICIONES ADMINISTRATIVAS

1.1. CONDICIONES GENERALES

En este pliego se ordenan las condiciones por las que se han de regir en el desarrollo y ejecución del sistema tratado en este proyecto: Vivienda inteligente y autosuficiente. El proyecto debe incluir los siguientes documentos: Portada e Índice general, Memoria, Anexo, Planos, Pliego de Condiciones y Presupuesto.

Se debe facilitar una copia firmada por los autores de todos los documentos que forman el proyecto al comprador de este; en caso de que el comprador encontrase alguna errata en algún documento, deberá indicarlo a la Dirección Facultativa para que se corrijan antes de iniciarse el trabajo. En caso contrario, no se podrá reclamar contra la orden de sustitución de la obra debidamente ejecutada.

1.2. OBJETO DEL PLIEGO DE CONDICIONES

El presente pliego trata los procesos especificados en los documentos que contiene el presente proyecto y, de forma excepcional, todos aquellos que deba llevar a cabo la empresa realizadora para conseguir terminar las actuaciones proyectadas y que se describen en la memoria.

Además, se incluirá cualquier cambio de diseño u obra que, por reforma, surja durante la realización del proyecto, así como aquellas acciones que, en el momento de redacción del proyecto, se omitieron y fuesen necesarias para la correcta terminación de los sistemas detallados en los documentos que componen el proyecto.

1.3. INTERPRETACIÓN DEL PROYECTO

La empresa realizadora del proyecto debe estar capacitada para la interpretación de éste con carácter global, o en su defecto, tiene personal cualificado a su servicio capaz de interpretar todos los documentos que forman el proyecto. La ejecución de todos los procesos debe estar estrictamente sujeta al pliego de condiciones y conforme a las directrices dadas en el proyecto.

1.4. NORMATIVA VIGENTE

Se ha elaborado el presente proyecto teniendo en cuenta con carácter global las siguientes normas y reglamentos:

- Real Decreto 1495/1986, de 26 de mayo, sobre el reglamento de seguridad en las máquinas.
- Real Decreto 2200/1995, de 28 de diciembre, por el que se aprueba el Reglamento de la Infraestructura para la Calidad y la Seguridad Industrial, en lo referente a la calidad industrial ISO 9001.
- Ley 31/1995, de 8 de noviembre, referente a la prevención de riesgos laborales.
- Real Decreto 2295/1985, de 9 de octubre de 1985, por el que se adiciona un nuevo párrafo al artículo 2 del Reglamento Electrotécnico de Baja Tensión, aprobado por Decreto 2413/1973, de 20 de septiembre.
- Real Decreto 1338/185, de 4 de julio, sobre medidas de seguridad en entidades y establecimientos públicos y privados.
- Real Decreto 485/1997, de 14 abril, sobre disposiciones mínimas en materia de señalización de seguridad y salud en el trabajo. Rango de Tensión continúa: 200mV, 2V, 20v, 2kV.
- Real Decreto 444/1994, de 11 marzo, por el que se establece los procedimientos de evaluación de la conformidad y los requisitos de protección relativos a compatibilidad electromagnética de los equipos, sistemas e instalaciones.
- Real Decreto 1950/1995, de 1 de diciembre, por el que se modifica el Real Decreto 444/1994, de 11 de marzo, por el que se establecen los procedimientos de evaluación de la conformidad y los requisitos de protección relativos a compatibilidad electromagnética de los equipos, sistemas e instalaciones.
- Orden de 19 de julio de 1999, de desarrollo del Real Decreto 444/1994, de 11 de marzo, por la que se publica la relación de normas españolas que transponen las normas europeas armonizadas, cuyo cumplimiento presume la conformidad con los requisitos de protección electromagnética.
- Directiva 89/336/CEE del Consejo de 3 de mayo de 1989 sobre la aproximación de las legislaciones de los Estados Miembros relativas a la compatibilidad electromagnética.
- Directiva 73/23/CEE del Consejo, de 19 febrero de 1973, relativa a la aproximación de las legislaciones de los Estados Miembros sobre el material eléctrico destinado a utilizarse con determinados límites de tensión.
- Orden de 30 de septiembre de 1980 por la que se dispone que las normas UNE que se citan sean consideradas como de obligado cumplimiento incluyéndolas en la instrucción MI-BT-044 del Reglamento Electrotécnico de Bata Tensión.

- Orden de 5 de junio de 1982, por la que se dispone la inclusión de las Normas UNE que se relacionan en las Instrucción MIE-BT-044 del Reglamento Electrotécnico de Bata Tensión.
- Orden de 22 de noviembre de 1995, por la que se adapta al progreso técnico la instrucción complementaria MI-BT-044 del Reglamento Electrotécnico de Bata Tensión.
- Reglamento Electrotécnico de Baja Tensión.
- Normas CEPREVEN, Centro Nacional de Prevención de Daños y Pérdidas.
- Directiva RoHS referente a la restricción de sustancias peligrosas en aparatos eléctricos y electrónicos.
- Directiva WEEE sobre el reciclaje de equipos eléctricos y electrónicos.
- ISO 9001 referente al sistema de gestión medioambiental.
- OHSAS 18001 referente a una serie de especificaciones sobre la salud y seguridad en el trabajo.
- UNE 157001:2001 mediante la cual se definen los criterios generales para la elaboración de proyectos.
- Se tendrán en cuenta las órdenes provistas por las Delegaciones Provinciales de Industria, así como las normas de obligado cumplimiento con la Legislación Eléctrica Española.

El conexionado del sistema se efectuará según las normativas de la compañía que realiza el contrato, del cual se responsabiliza esta misma. Para la elaboración del proyecto, se debe aplicar la revisión actualizada y aceptada de las normativas arribas descritas. En caso de discrepancia entre las normas y cualquier parte del pliego de condiciones, siempre se aplicará la norma más rigurosa. En caso de que el cliente lo desee, se podrá aplicar cualquier otra norma o recomendación reconocida internacionalmente.

1.5. CONDICIONES LEGALES

1.5.1. Comienzo del proceso de fabricación

El ingeniero responsable del proceso de fabricación será quien dé las instrucciones necesarias para la buena ejecución de este, de acuerdo siempre con los documentos que constituyen el proyecto. 1.5.2 Uso del dispositivo Todo usuario tiene derecho a consultar a la empresa proyectista sobre el funcionamiento del sistema, siendo el primero responsable de los daños que pudieran derivarse por una mala utilización de este, y/o por una falta de mantenimiento, conservación o cuidados necesarios que sean obligatorios o estén reglamentados.

1.6. CONDICIONES DE SEGURIDAD Y COMPROMISOS CONTRACTUALES

1.6.1. Del personal de fabricación

Todo operario involucrado en la fabricación del sistema tiene derecho a reclamar a la empresa todos aquellos elementos que garanticen su seguridad personal durante la preparación y ejecución de los trabajos que le sean encomendados, de acuerdo con la legislación vigente. La empresa encargada de la realización del proyecto tiene la obligación de poseer dichos elementos y facilitarlos en condiciones aptas para su uso, así como de exigir su utilización a los operarios.

1.6.2. De las empresas fabricantes de los componentes

Las empresas encargadas en la fabricación de los componentes del proyecto se comprometerán a ejecutar la fabricación de estos ajustándose a las disposiciones laborales hoy vigentes, recayendo sobre ellas la responsabilidad de cualquier daño personal o material que pudieran ocurrir si, por imprudencia, negligencia o impericia, dejaran de cumplir las condiciones descritas en este pliego, así como si evitan tomar cualquier precaución para la seguridad en el trabajo.

Además, todas y cada una de las partes de fabricación deben ser confiadas a personas cualificadas, bajo constante vigilancia del encargado de esta, y la supervisión del ingeniero responsable.

1.6.3. Del contrato

El contrato debe ser firmado por la parte interesada en la fabricación del dispositivo y por la empresa fabricante, suponiendo la firma de este, de mutuo acuerdo con las cláusulas que entre ambas partes queden estipuladas. Toda cláusula que se oponga a lo estipulado en los diversos apartados de este Pliego de Condiciones se considera como nula, así como toda cláusula que pueda generar la utilización de materiales de mala calidad o que no sean aceptados por la Dirección Técnica.

Las circunstancias no previstas en el proyecto y que requieran de la modificación de los costos de fabricación, necesitarán la aprobación del propietario intelectual del sistema. Cualquier modificación de los costos de fabricación tendrá que ser aprobada por el director del proyecto.

La forma de pago vendrá reflejada en el contrato establecido. Se podrá imponer multas de hasta el 20% del presupuesto total del costo del elemento en caso de incumplimiento en cuanto al plazo establecido.

El contrato puede ser prescindido por cualquier causa reconocida como válida en las cláusulas de este o en la legislación vigente.

1.6.4. Del fabricante

El fabricante tiene la obligación de facilitar al comprador un ejemplar completo del presente proyecto a fin de que pueda hacerse cargo de todas y cada una de las obligaciones que se especifican en este pliego.

1.6.5. Del proyecto

Los trabajos a realizar quedan especificados en los planos, anexos y memoria, así como en las condiciones técnicas, legales y contractuales. Se entiende en este pliego, que la empresa fabricante está capacitada para la interpretación del proyecto en todas sus partes, o en su defecto, tiene personal cualificado para interpretar correctamente todos los documentos de este.

1.6.6. Del presupuesto

Se entiende en este pliego que el presupuesto base de los componentes que componen el dispositivo es el que figura en el presente proyecto debiéndose tener en cuenta la existencia de múltiples unidades de este.

2. PLIEGO DE CONDICIONES TÉCNICAS

2.1. CONDICIONES TÉCNICAS GENERALES DEL PROYECTO

En este apartado se detallan los requisitos técnicos mínimos que deben cumplir los elementos y componentes que constituyen el prototipo fabricado para facilitar su mantenimiento y conservación.

2.1.1. Mantenimiento del sistema

El dispositivo debe de revisarse al menos una vez o dos anualmente mediante una breve inspección visual y testeo de los componentes, tanto internos como externos para comprobar el estado general del mismo. En caso de avería, el personal de mantenimiento deberá realizar las sustituciones pertinentes.

Además, las comprobaciones de funcionamiento, las realizará el propio usuario y este deberá ponerse en contacto con el instalador en caso de detectar problemas de funcionamiento.

2.1.2. Características de los componentes

Para asegurar el correcto funcionamiento del dispositivo se deben respetar las características de los componentes que se especifican en planos y memoria. Siendo estas de obligado cumplimiento por parte de la empresa encargada de la fabricación del sistema. Cualquier cambio o sustitución de componentes deberá cumplir las equivalencias de los componentes antiguos. La empresa proyectista no se hace responsable de los cambios de los valores y características que puedan sufrir los componentes sin su previa autorización o notificación por escrito.

La empresa encargada de fabricar el prototipo deberá comunicar al fabricante los posibles cambios realizados en el diseño y en las características de los componentes, siempre bajo la autorización de la empresa proyectista.

Todos los componentes electrónicos empleados en la construcción del sistema están reglamentados de acuerdo con las hojas de características del fabricante y deben cumplir los requisitos indicados en lo referente a potencia, tensión y corriente.

2.2. PRECAUCIONES DE USO

Se debe cumplir las siguientes medidas de seguridad debido al carácter eléctrico y electrónico del sistema:

- El dispositivo solo se empleará en las condiciones específicas y con fines para los cuales ha sido diseñado.
- El sistema deberá ser alimentado en los rangos indicados, nunca sobrepasar dichos valores.
- Deberán evitarse golpes y acciones abrasivas que pudieran dañar el dispositivo. La garantía no cubrirá desperfectos ocasionados por choques o deterioros por un uso indebido por parte del usuario.
- Durante la manipulación del aparato resultara de vital importancia seguir las precauciones de seguridad normalmente empleadas en laboratorios de electricidad y potencia, teniendo en cuenta en todo momento la normativa vigente sobre prevención de riesgos laborales.

2.3. GARANTÍA Y SERVICIO POSTVENTA

A partir de la recepción del sistema, la garantía será de dos o tres años, como mínimo. Esta incluirá durante todo el periodo, la reparación y sustitución de componentes, con intervención del personal técnico.

La garantía no se aplicará en ninguno de los siguientes casos:

- Averías producidas por el mal uso del dispositivo.
- Alimentación del sistema con una tensión incorrecta a la especificada.
- Manipulación o reparación del dispositivo realizada por terceros.

No existe ninguna otra garantía o seguro diferente de lo estipulado en este documento y todas las restantes garantías quedan excluidas y exentas de la responsabilidad por medio del presente documento.

3. PLIEGO DE CONDICIONES ECONÓMICAS

3.1. ABONO DE OBRA

En el contrato se deberá incluir, de manera detallada, las formas y plazos de pago. En caso de haber pagos parciales, tendrán carácter de documentos provisionales, sujetos a las certificaciones que resulten de la liquidación final. Terminada la obra, se procederá a la liquidación final, de acuerdo con los criterios establecidos en el contrato.

3.2. PRECIOS

Al formalizarse el contrato, la relación de precios de las unidades de obra será presentada con el Contratista. Estos precios, comprenderán la ejecución total de la obra, así como la parte proporcional de imposición fiscal, cargas laborales y otros gastos repercutibles. En caso de realizarse unidades de obra no previstas en el proyecto, se fijará su precio entre el Técnico, director y el Contratista antes de iniciar la obra, presentándosela al interesado para obtener su conformidad.

3.3. PREVISIÓN DE PRECIOS

En caso de que el Contratista tenga derecho a revisión de precios, se deberá indicar en el contrato, así como la fórmula a aplicar para calcular dicha revisión. Se aplicará a juicios del Técnico, director alguno de los criterios oficiales aceptados.

3.4. PENALIZACIONES

Se podrán aplicar tablas de penalización en caso de retraso en los plazos de entrega de la obra. Las cuantías y demoras se fijarán en el contrato.

Documento Nº4:

Presupuesto

ÍNDICE

ÍNDICE DE TABLAS

1. MANO DE OBRA DIRECTA

Teniendo en cuenta el precio estimado del ingeniero y la cantidad de horas trabajadas, se obtiene el precio de la mano de obra:

CONCEPTO	SUELDO (€/h)	HORAS	TOTAL (€)
Trabajo ingeniero	20	250	5000

Tabla 1: Coste mano de obra

De esta forma se obtiene un precio de mano de obra de 5000€.

2. MATERIAS PRIMAS

El presupuesto para todos los equipos utilizados para llevar a cabo el proyecto es el siguiente:

REFERENCIA	PRODUCTO	PRECIO UNITARIO (€)	PRECIO TOTAL (€)
RPI4-B-4GB	RASPBERRY PI 4 - MODELO B - 4GB	60,95	60,95
SDCS2/32GB	KINGSTON CANVAS SELECT PLUS MICROSDHC 32GB CLASS10 UHS-I A1 100MB/S	5,99	5,99
KSA-15E-051300HE-BL	ALIMENTADOR OFICIAL RASPBERRY PI 4 USB-C 5V 3A 15W BLANCO	8,95	8,95
T7690AX	CABLE MICRO HDMI (TIPO D) 2M. BLANCO OFICIAL RASPBERRY PI	6,95	6,95
USBWEBCAM5	CAMARA WEB USB 5MPX CABLE RETRACTABLE COMPATIBLE RASPBERRY PI	5,95	5,95
1286-1041-ND	BOARD FPGA BASYS3 FOR VIVADO	126,56	126,56
Total	-	-	215,35

Tabla 2: Coste materias primas

El precio total de los equipos es de 215,35€, teniendo en cuenta que el envío es de 5€ el coste total es de 220,35 €.

3. PUESTO DE TRABAJO

Para realizar las pruebas necesarias para asegurar el buen funcionamiento del proyecto se genera un consumo energético que deriva en el siguiente coste:

$$Coste\ energetico = 0{,}40\frac{€}{h} * 250h = 100€$$

4. COSTE TOTAL

Sumando todos los costes descritos anteriormente, el precio total es:

PRESUPUESTO FINAL DEL PROYECTO	
Mano de obra	5000€
Materias primas	215,35€
Puesto de trabajo	100€
Total	5315,35€

Tabla 3: Coste total

El precio final del proyecto es de 5315,35€.