UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería Electrónica

Desarrollo de un interfaz electrónico de señalización digital para el control rápido de cavidades aceleradoras de RF

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Fernández Ortega, Juan Carlos

Tutor/a: Torres Carot, Vicente

Cotutor/a externo: Boronat Arévalo, Marçà

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

DESARROLLO DE UN INTERFAZ ELECTRÓNICO DE SEÑALIZACIÓN DIGITAL PARA EL CONTROL RÁPIDO DE CAVIDADES ACELERADORAS DE RF

Autor: Juan Carlos Fernández Ortega

Tutor: Dr. Vicente Torres Carot

Tutor: Dr. Marçà Boronat Arevalo

Trabajo Fin de Máster presentado en el Departamento de Ingeniería Electrónica de la Universitat Politècnica de València para la obtención del Título de Máster Universitario en Ingeniería de Sistemas Electrónicos

Curso 2023-24

Valencia, Julio de 2024

Resumen

El presente trabajo de final de máster trata acerca de la implementación de un sistema de sincronización a través de señales de disparos (trigger) y señales de interbloqueos (interlock) basado en un SoC para la seguridad de una zona controlada. Esta zona puede variar, pero en este caso es una zona de radiación.

Los principales objetivos de este proyecto son programar un SoC diseñando varios core IP que generará las señales triggers y las señales interlock cuyos parámetros son variables y controlables.

Además, se estudia una pcb de entradas y salidas ya diseñada anteriormente. En concreto se realiza un estudio de disipación térmica y fiabilidad, y un estudio del ensamblaje donde irá conectado. Con este sistema se puede comprobar las señales de triggers e interlocks en un osciloscopio.

De forma definitiva, se configurará un sistema de control distribuido TANGO de código libre que controlará todas las señales del core IP ya que la duración, prioridad… de los pulsos puede variar, y con este sistema TANGO se podrá ir cambiando los parámetros sin tener que modificar el core ip. Finalmente se implementará una interfaz gráfica para que el usuario pueda realizar estos cambios directamente desde un PC de laboratorio.

El proyecto tiene como tarea principal el diseño de varios core IP para el funcionamiento sobre una FPGA, pero como se ha comentado se realizarán otras tareas de vital importancia como es el caso del estudio térmico de la PCB y el sistema de control distribuido que tiene una complejidad elevada y puede conllevar a una curva de aprendizaje larga.

Todo esto, formará parte de un sistema de seguridad de un laboratorio de radiación donde es imprescindible que el sistema responda de forma fiable y segura, pues de ello depende la seguridad del personal y la maquinaria. Se implementará en el laboratorio de RF del IFIC donde se estudian cavidades en aceleradores de partículas con la finalidad de mejorar la eficiencia en la radioterapia para el tratamiento contra el cáncer.

Resum

El present treball de final de màster tracta sobre la implementació d'un sistema de sincronització a través de senyals de dispar (trigger) i senyals d'interbloqueig (interlock) basat en un SoC per a la seguretat d'una zona controlada. Aquesta zona pot variar, però en aquest cas és una zona de radiació.

Els principals objectius d'aquest projecte són programar un SoC dissenyant diversos core IP que generaran les senyals triggers i les senyals interlock, els paràmetres dels quals són variables i controlables.

A més, s'estudia una PCB d'entrades i sortides ja dissenyada anteriorment. En concret es realitza un estudi de dissipació tèrmica i fiabilitat, i un estudi de l'assemblatge on anirà connectada. Amb aquest sistema es pot comprovar les senyals de triggers i interlocks en un oscil·loscopi.

De manera definitiva, es configurarà un sistema de control distribuït TANGO de codi lliure que controlarà totes les senyals del core IP ja que la durada, prioritat... dels polsos pot variar, i amb aquest sistema TANGO es podrà anar canviant els paràmetres sense haver de modificar el core IP. Finalment s'implementarà una interfície gràfica perquè l'usuari pugui realitzar aquests canvis

directament des d'un PC de laboratori.

El projecte té com a tasca principal el disseny de diversos core IP per al funcionament sobre una FPGA, però com s'ha comentat es realitzaran altres tasques de vital importància com és el cas de l'estudi tèrmic de la PCB i el sistema de control distribuït que té una complexitat elevada i pot portar a una corba d'aprenentatge llarga.

Tot això, formarà part d'un sistema de seguretat d'un laboratori de radiació on és imprescindible que el sistema respongui de forma fiable i segura, ja que d'això depèn la seguretat del personal i la maquinària. Es implementarà al laboratori de RF de l'IFIC on s'estudien cavitats en acceleradors de partícules amb la finalitat de millorar l'eficiència en la radioteràpia per al tractament contra el càncer.

Abstract

The present Master's thesis focuses on the implementation of a synchronization system using trigger and interlock signals based on a SoC for the safety of a controlled area. This area may vary, but in this case, it is a radiation zone.

The main objectives of this project are to program a SoC by designing various IP cores that will generate the trigger and interlock signals, whose parameters are variable and controllable.

Additionally, an already designed input and output PCB is studied. Specifically, a thermal dissipation and reliability study is conducted, as well as an assembly study of where it will be connected. With this system, the trigger and interlock signals can be checked on an oscilloscope.

Ultimately, a TANGO distributed control system, an open-source code, will be configured to control all the signals of the IP cores since the duration, priority, etc. of the pulses may vary. With this TANGO system, parameters can be changed without modifying the IP core. Finally, a graphical interface will be implemented so that the user can make these changes directly from a laboratory PC.

The main task of the project is the design of various IP cores for operation on an FPGA. However, as mentioned, other tasks of vital importance will be carried out, such as the thermal study of the PCB and the distributed control system, which has high complexity and may involve a long learning curve.

All of this will be part of a security system for a radiation laboratory where it is essential that the system responds reliably and safely, as the safety of personnel and machinery depends on it. It will be implemented in the RF laboratory of the IFIC where cavities in particle accelerators are studied with the aim of improving efficiency in cancer radiotherapy treatment.

Muestro mi agradecimiento de haber podido realizado la tesis gracias a esas personas que me han acompañado a lo largo de estos meses.

A Vicente Torres por los consejos y correcciones de cada capítulo. Aunque el trabajo ha sido realizado fuera de la UPV me ha ayudado mucho y se lo agradezco.

A Marçà Boronat por la ayuda diaria sobre el conocimiento del laboratorio de RF y la ayuda en la escritura del trabajo.

Al IFIMED ya que han aportado su granito de arena a este trabajo con recomendaciones y ayuda en algunas partes del trabajo.

A mi pareja por apoyarme cada día y buscar la positividad en todo momento.

A mis padres por confiar en mí y levantarme en los malos momentos, y sobre todo alegrarse por mí en los buenos.

Índice general

Índice de figuras

I Introducción

II Sistema. Hardware y software

III Desarrollo y resultados

Índice de tablas

Listado de siglas empleadas

AIO AXI Input/Output.

AMD Advanced Micro Devices, Inc..

ARM Advanced RISC Machine.

AXI Advanced eXtensible Interface.

CERN Centre Européen de Recherche Nucléaire.

CNC Computer Numerical Control.

CORBA Common Object Request Broker Architecture.

CPU Central Processing Unit.

DC Direct Current.

DDR Double Data Rate.

DESY Deutsches Elektronen Synchrotron.

EMIO Extended Multiplexed I/O.

eV Electronvoltio.

FPGA Field-Programmable Gate Array.

GHz Gigahertz.

GND Ground.

h coeficiente de transferencia de calor.

HG-RF High Gradient - Radio Frequency.

Hz Hertz.

I2C Inter-Integrated Circuit.

IFIC-UV Instituto de Física Corpuscular - Universidad de Valencia.

IOP Input/Output Processor.

IP Intellectual Property.

LED Light Emitting Diode.

LHC Large Hadron Collider.

LINAC Linear Accelerator.

MHz Megahertz.

MIO Multiplexed I/O.

NTC Negative Temperature Coefficient.

PC Personal Computer.

PCB Printed Circuit Board.

PETRA III Positron-Electron Tandem Ring Accelerator.

PL Programmable Logic.

PMOD Peripheral Module.

PS Processing System.

PYNQ Python Productivity for Zynq.

RAM Random Access Memory.

SCADA Supervisory Control And Data Acquisition.

SD Secure Digital.

SDK Software Development Kit.

SMD Surface-Mount Device.

SMT Surface-mount technology.

SoC System on Chip.

SPI Serial Peripheral Interface.

SQL Structured Query Language.

TANGO TAco Next Generation Objects.

UART Universal Asynchronous Receiver-Transmitter.

XDC Xilinx Design Constraints.

ZMQ ZeroMQ.

Parte I

Introducción

Capítulo 1

Objetivos

Ingeniería y física han trabajado conjuntamente para innovar y tratar de desarrollar equipamiento útil para la sociedad. Uno de los campos de aplicación es la medicina, donde la mayoría de instrumentos utilizados han sido diseñados anteriormente por un grupo de ingenieros y físicos para su correcto funcionamiento.

En el proyecto que se va a describir se tratará de realizar un sistema de control y seguridad para un laboratorio que emite radiación que es perjudicial para la salud. En cuanto al laboratorio, está dedicado al estudio de las cavidades aceleradoras que utilizan en los aceleradores de partículas para diversos ámbitos, pero en este caso en física médica, en concreto para terapias con radiación en el tratamiento del cáncer.

Figura 1.1: Laboratorio HG-RF del IFIC-UV. Fuente: propia

El desarrollo de la interfaz electrónica propuesta para este proyecto tiene unos objetivos claros.

- **Generación de la señalización digital** para activar los subsistemas del laboratorio de forma secuencial con unos parámetros ajustables, se utilizarían las señales de disparo (triggers). Por otra parte, tras diferentes combinaciones de entradas se deberá de mandar señales de interbloqueo (interlock) para detener parcialmente o totalmente los subsistemas de laboratorio del laboratorio, que vienen registradas de un bloque de registro o latch que permite capturar dichas señales. En la figura 1.2 se puede visualizar la generación de la señalización digital desde la FPGA (Field Programmable Gate Array) hacia los periféricos y la evaluación de las señales desde los periférios hacia el sistema de control para evaluar si hay que detener

el sistema. Es de vital importancia entender que este proyecto no trataremos los periféricos ni el sistema de control, sino que se diseñará el bloque de la FPGA para evaluar y mandar señales de disparo e interbloqueos. De hecho en la demostración final de este proyecto los periféricos y el sistema de control son reemplazados por el osciloscopio donde se visualizarán las señales que se han configurado para comprobar el correcto funcionamiento que se ha diseñado de forma correcta. La finalidad del proyecto es diseñar un sistema versátil sobre FPGA que se pueda utilizar en diferentes aplicaciones, como por ejemplo, empresas de telefonía, eléctricas, entorno sanitario o cualquier lugar que necesite de un sistema de seguridad y control de señales, aunque en este caso se aplicará al control de un laboratorio de HG-RF (High Gradient - Radio Frequency), distribuido por subsistemas.

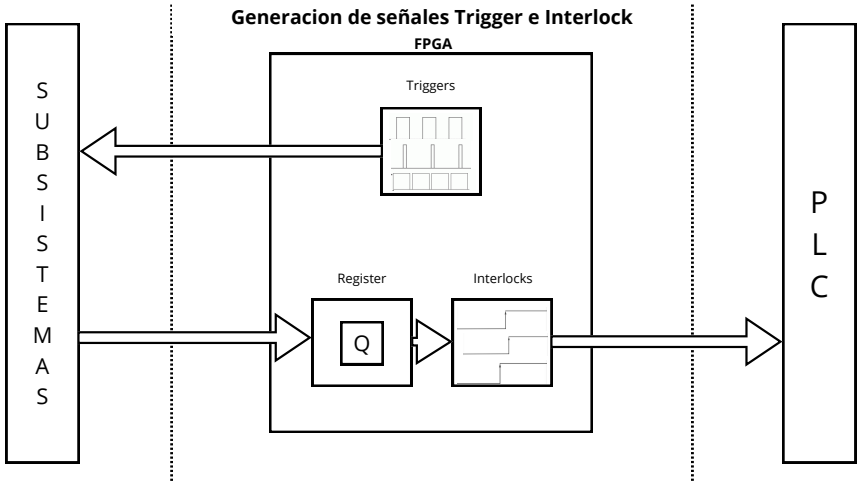


Figura 1.2: Diagrama de la implementación sobre la FPGA. Fuente: propia

- **Selección e interconexión de hardware y software** que permita que la aplicación tenga las herramientas para que el funcionamiento de los subsistemas se pueda centralizar en un único PC. Se necesitará una interfaz electrónica albergada por una PCB conectada a una placa de desarrollo de Xilinx que incluirá un (*System On Chip*) SoC y FPGA en la misma placa que permitirá interconectar directamente los subsistemas de laboratorio que se deben controlar desde el PC. Sobre el PC se desarrolla un sistema de control distribuido basado en TANGO (tango-controls de aquí en adelante) que permite ajustar los parámetros sobre las posiciones de memoria que se definirán más adelante.

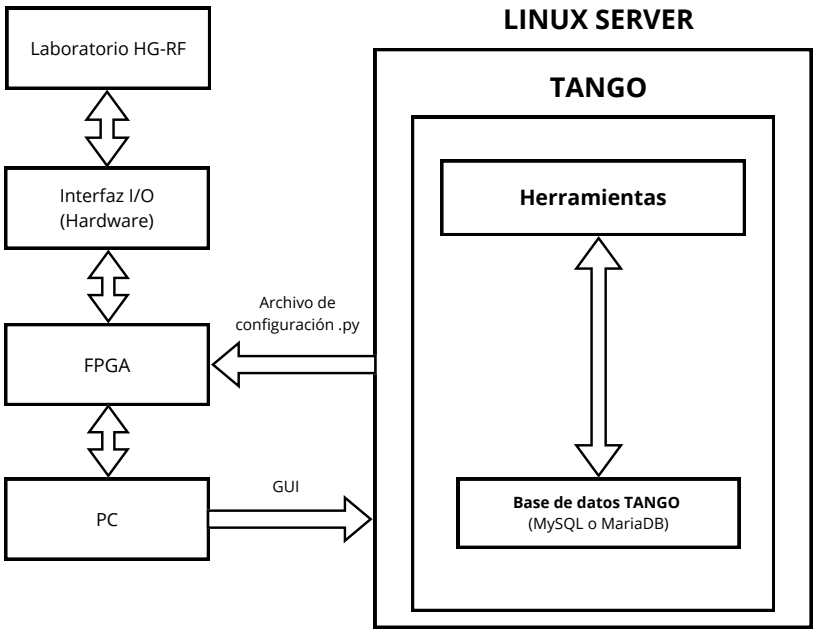


Figura 1.3: Esquema general del proyecto. Fuente: propia

- **Aprendizaje del software propuesto** tanto el software para la placa de Xilinx (Vivado) donde es importante conocer todas las herramientas que contiene. Y por otra parte, tango-controls es un sistema basado en C++, python o java y y se implementará en el lenguaje mas afín al proyecto. Además se deberá de aprender y utilizar las herramientas de tango-controls para el acceso a la base de datos.

- **Diseño de IPs** en Vivado para la activación e interbloqueo de los subsistemas de laboratorio. Se deberán integrar tres *Custom Core IP AXI* que permitan realizar estas funciones: señal de disparo, señal de interbloqueo y registro. El registro se utilizará para registrar señales de disparo que pudieran ser procesadas en un futuro.

- **Estudio de la Interfaz de entradas y salidas** que está gestionada por una PCB ya diseñada, pero que hay que caracterizar y estudiar en profundidad. Se realizará un estudio teórico y práctico sobre la disipación térmica, y cómo afecta el calor que generan los componentes al funcionamiento y vida útil de los componentes de la PCB. Se debe indicar que es de vital importancia que realice su funcionamiento correctamente para que no se pierda la interconexión entre FPGA y subsistemas del laboratorio. Además, se realizará una pequeña descripción del funcionamiento de la PCB que cuyo objetivo es cambiar la tensión y aumentar la corriente de las entradas y salidas gracias a los drivers de alta velocidad.

En resumen, los objetivos principales es realizar el hardware de control que pueda generar las señales de disparo con la frecuencia y distribución temporal deseada, para el correcto funcionamiento de los subsistemas del laboratorio. Así como la interfaz I/O para programar dichas señales sin perturbar el funcionamiento de la FPGA. Finalmente, como se ha explicado anteriormente esta información será controlada gracias al sistema de control distribuido (tango-controls) que proporcionará interfaz gráfica y acceso al servidor para modificar las especificaciones de las señales.

Capítulo 2

Antecedentes

En el año 1946 Robert Wilson planteó por primera vez el uso de protones para tratar pacientes con cáncer. En las décadas posteriores se comprobó su eficacia en un número reducido de pacientes y en patologías muy específicas. Desde entonces se han construido numerosos centros de tratamiento y investigación por todo el mundo. En el año 1970 el *Lawrence Berkeley Laboratory* empezó a estudiar el uso de otras partículas [1]. En las últimas décadas ha aumentado el estudio creando nuevos centros de investigación por todo el mundo, incluyendo España. Con ello, ha aumentado la eficacia de las técnicas para el tratamiento, mejorando las técnicas de tratamiento y la eficacia de las misma, aumentando notablemente los indices de supervivencia de personas con ciertos tipos de cáncer.

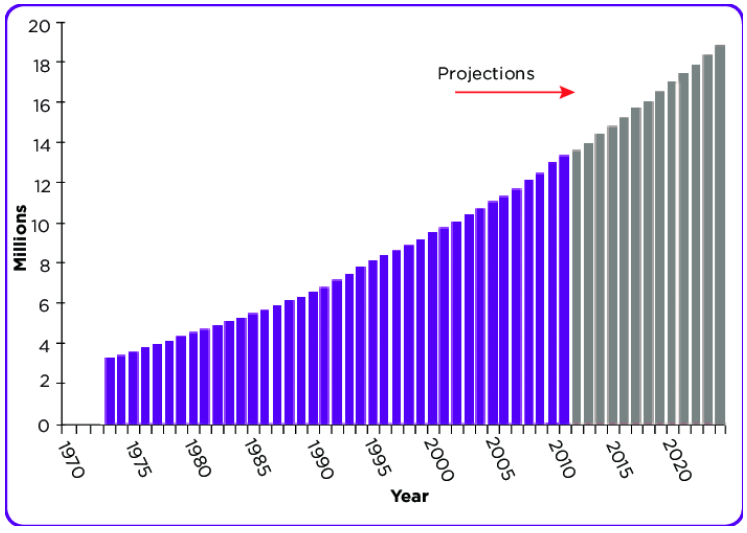


Figura 2.1: Número estimado de supervivientes de cáncer en los Estados Unidos. Información del Instituto Nacional del Cáncer. Fuente: [2]

Cada centro de investigación tiene líneas de investigación diferentes dependiendo de la línea de especialización del centro. En nuestro caso el IFIMED se centra en el desarrollo de aceleradores de partículas para aplicaciones médicas. Las cavidades aceleradoras se rigen por las ecuaciones básicas del electromagnetismo desarrolladas por Maxwell. Sin embargo, existe una ecuación muy sencilla que sirve para definir las fuerzas que actúan sobre las partículas, dentro del acelerador.

Esta es la ecuación o ecuaciones (cuando se usan de forma separada) de Lorentz.

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \tag{2.1}$$

Los usos de partículas cargadas para tratamiento de cancer, radioterapia, se pueden clasificar según la energía de las partículas (eV) y del tipo de partícula utilizada.

- **Radioterapia Convencional**: utiliza rayos X de alta energía para destruir las células cancerosas. Este tratamiento ha evolucionado significativamente desde la década de 1960 con el desarrollo de tecnologías como la Radioterapia de Intensidad Modulada (IMRT) y la Radioterapia Guiada por Imágenes (IGRT), que permiten una mayor precisión al dirigir las dosis de radiación.
- **Hadronterapia**: es un tipo avanzado de radioterapia que utiliza partículas pesadas (hadrones) como protones y iones de carbono en lugar de rayos X. Tiene la ventaja de poder depositar la dosis máxima de energía directamente en el tumor con menos daño a los tejidos circundantes [3]. La terapia de hadrones (HT) es una técnica que utiliza haces de protones o iones para tratamientos contra el cancer, obteniendo mayor eficacia y menor toxicidad. La principal ventaja de la HT radica en las propiedades físicas únicas de los hadrones cargados, especialmente protones e iones de carbono, que les permiten penetrar en los tejidos y depositar la máxima energía al final de su trayectoria según la curva de Bragg [4]. Dentro de la hadronterapia existen dos alternativas, protonterapia e ionterapia.

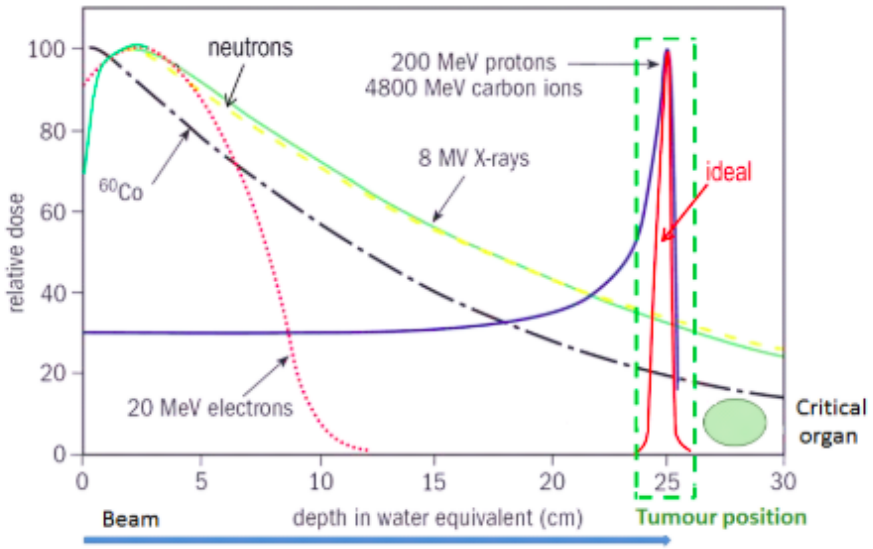


Figura 2.2: Distribución de dosis en la profundidad de agua de diferentes haces de partículas. Fuente: [5]

- **Protonterapia**: Usa protones y permite una mayor precisión en la administración de la radiación, adecuada para tumores cerca de estructuras críticas como el cerebro y la columna vertebral. En el año 2003 nuevos centros de protonterapia estaban en construcción: uno en Suiza, uno en Japón, uno en Alemania, uno en Corea, dos en China y

tres en EE.UU [6]. A día de hoy se ha expandido la construcción de centros de protonterapia de forma exponencial, por ejemplo, en España ya los hay en dos centros privados, mientras que los primeros diez centros en el ámbito público están actualmente en fase de construcción.

- **Ionterapia**: Utiliza iones de carbono que tienen una mayor masa que los protones, proporcionando una mayor eficacia biológica. Es efectiva contra tumores que son resistentes a otros tipos de radioterapia [3].

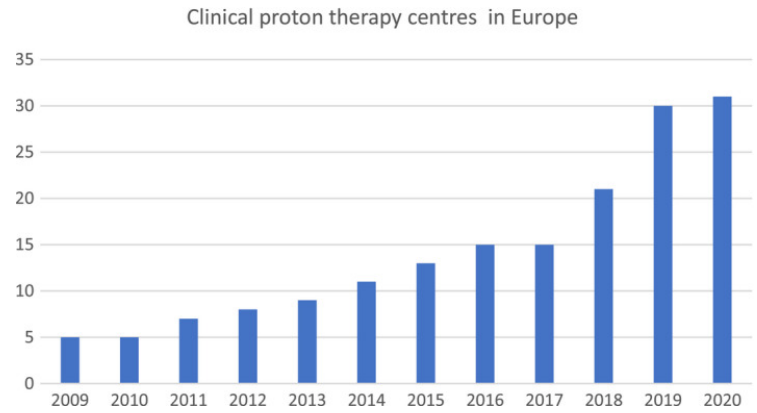


Figura 2.3: Número de clínicas de protonterapia en Europa en los años 2009–2020. Fuente: [7]

Hay varios tipos de aceleradores, como son los sincrotrones, ciclotrones y LINAC (*Linear accelerator*). Algunos ejemplos de grandes aceleradores que hay en España es el sincrotrón ALBA, en Barcelona. Por otra parte, otros ejemplos en Europa son el LHC (*Large Hadron Collider*) en el CERN (*Centre Européen de Recherche Nucléaire*) ubicado en Suiza, o el PETRA III en DESY (*Deutsches Elektronen Synchrotron*) ubicado en Alemania.

En el laboratorio de RF del IFIC-UV no hay un acelerador de partículas como tal, hay un laboratorio de RF para el estudio de las cavidades aceleradoras. Uno de los componentes que debe existir es el *Interlock system*.

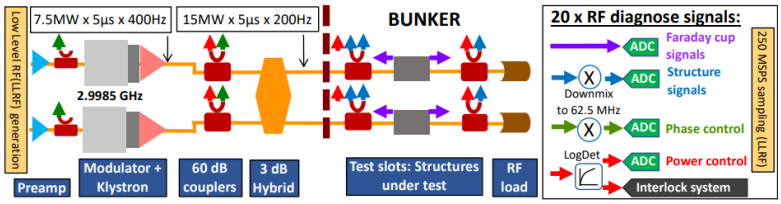


Figura 2.4: Esquema de la primera fase del sistema del laboratorio HG-RF, Valencia. Fuente: [8]

Una cavidad aceleradora de RF es un componente crucial en los aceleradores de partículas, como los utilizados en los colisionadores de partículas y en instalaciones de radioterapia. Estas cavidades utilizan campos eléctricos generados por señales de radiofrecuencia para acelerar partículas cargadas, como electrones, protones o iones, a altas energías.

1. **Generación de Radiofrecuencia (RF):** Un generador de RF produce ondas electromagnéticas a frecuencias específicas, generalmente en el rango de MHz a GHz.

2. **Cavidad Resonante:** La cavidad aceleradora está diseñada para ser resonante a cierta frecuencia de la señal de RF. Esto significa que la cavidad amplifica la señal de RF de manera eficiente.

3. **Campo Eléctrico Oscilante:** Dentro de la cavidad, el campo eléctrico oscila a la frecuencia de la señal de RF. Este campo eléctrico alternante es lo que acelera las partículas cargadas.

4. **Aceleración de Partículas:** Las partículas cargadas entran en la cavidad en sincronía con el campo eléctrico. Cuando las partículas se encuentran con la fase correcta del campo eléctrico, reciben un empuje adicional, acelerándose a medida que pasan por la cavidad.

Las cavidades aceleradoras están hechas de materiales conductores, como el cobre, y tienen formas geométricas específicas que permiten que la señal de RF forme un campo resonante. Pueden ser cavidades de forma cilíndrica.

En resumen, las cavidades aceleradoras de RF son fundamentales para la aceleración eficiente de partículas cargadas en una amplia variedad de aplicaciones científicas y médicas.

En las cavidades existen fenómenos físicos que perturban las cavidades. El uso de campos eléctricos elevados, puede producir violentas descargas eléctricas en el interior de las cavidades (arcos eléctricos), que pueden llegar a dañar el componente de manera permanente. Este fenómeno se conoce como breakdown, cuyas causas siguen siendo un tema de debate dentro de la comunidad científica.

Una perturbación de este estilo, como lo es el *breakdown*, es una de las señales que debe activar el sistema de interbloqueo ya que, si no se mitiga, puede dañar la cavidad de forma considerable. Además de fenómenos físicos dentro de la cavidad, pueden existir otros eventos que hay que monitorizar, como por ejemplo, insuficiente corriente por parte de los subsistemas de laboratorio, no activación de uno de los subsistemas de laboratorio, envío incorrecto de la comunicación o fallo de seguridad en el sistema. Generar una interfaz de comunicación estable de estas señales con el sistema de control distribuido es uno de los objetivos del presente trabajo.

Capítulo 3

Estructura del proyecto

La organización de este proyecto se ha estructurado en una serie de tareas. Se ha estimado el tiempo necesario para cada tarea y se ha previsto el resultado, con el objetivo de minimizar posibles inconvenientes. La gestión del proyecto ha proporcionado una visión clara de las actividades planificadas a lo largo de su duración y ofreciendo información detallada sobre la evolución del trabajo realizado.

En la figura 3.1 se muestra un diagrama de Gantt en el que se especifican las principales tareas y su duración. Las tareas que se muestran son bastante genéricas, ya que esto puede contribuir positivamente a la comprensión de su importancia. Por ello, se describirá detalladamente cada tarea para dar claridad a los distintos procesos.

Las tareas se han dividido de forma secuencial conforme se vaya adquiriendo y aprendiendolos conocimientos relevantes para continuar al siguiente paso, pero algunas de ellas se llevarán a cabo paralelamente.

1. **Tarea de planificación:** Es importante en cualquier proyecto ya que gracias a una buena planificación evita que el autor deba recurrir a reorganizar el proyecto en el futuro o incluso permite que el proyecto tenga una buena distribución temporal de las tareas siguientes
2. **Investigación:** Se ha caracterizado como la tarea en la que se explora información diferente a su campo de especialización, ya que el proyecto es de carácter investigativo con gran influencia en la física.
3. **Aprendizaje:** Es la tarea en la que se adquieren los conocimientos más afines al proyecto ingenieril, es decir, la formación y el uso de las herramientas necesarias para el funcionamiento del sistema digital.
4. **Diseño:** Se diseñará de forma paralela el sistema de control distribuido, sistema digital con lenguaje de descripción de hardware, diseño hardware y diseño software.
5. **Implementación y verificación:** Se realiza la implementación en el entorno y su verificación y proyecto futuro para evaluar si el entorno es adecuado tanto de forma simulada como real.
6. **Redacción:** No es una tarea que se hace al final del proyecto, sino que se empieza unos meses antes para que la carga de trabajo de redactar no se concentre en un solo mes. Del

mismo modo es una tarea muy relevante en cualquier proyecto o tesis, pues es la forma que tiene el autor de transmitir todo lo aprendido y conseguido en el proyecto.

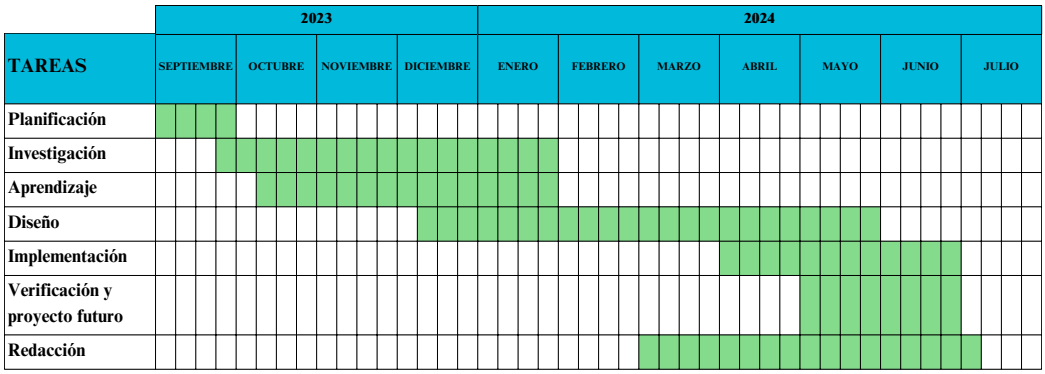


Figura 3.1: Diagrama de Gantt al principio del proyecto. Fuente: propia

El plan se ha seguido de la manera más exacta posible, con algún cambio debido a factores habituales, como desabastecimiento de componentes o disponibilidad de instrumentos del laboratorio.

Capítulo 4

Dificultades y retos

Como en todo proyecto existen problemas de menor peso, como pueden ser la compatibilidad entre sistemas, optimización de hardware o incluso un buen diseño hardware. En este proyecto en particular se trabajará sobre un amplio abanico de problemáticas, que hay que plantear y solucionar. Esta consideración es de vital importancia para la integridad operativa del sistema, dada la gran cantidad de componentes hardware interconectados, los cuales están en funcionamiento con un software que requiere una comunicación de datos eficiente y fiable para su correcto desempeño.

Una importante tarea es definir las necesidades de comunicación (interbloqueos y señales de disparo) que realmente son necesarias par el control efectivo del laboratorio de RF. La fiabilidad del sistema es un importante factor que se deberá tener en cuenta a la hora del diseño, por ello se deberá de diseñar bloques específicos en la programación digital que incluyan todas las posibilidades de la forma más sencilla, pero que sea flexible. Además, esta fiabilidad está relacionada con la sincronización del sistema ya que las señales si no están sincronizadas perfectamente el error de tiempo de respuesta aumenta.

La decisión de utilizar una FPGA se debe al gran volumen de datos que deben ser procesados, pero para este proyecto no se trabajará a la máxima tasa de datos, ni se implementaran todas las señales requeridas. De hecho esta parte es la parte previa al procesamiento de datos del sistema completo del laboratorio, por lo que es necesario mandar señales de disparo e interbloqueos a una frecuencia bastante baja siendo la máxima frecuencia 400 Hz. La decisión de utilizar una FPGA no es la frecuencia de operación sino la capacidad de tratar datos de una forma eficaz y rápida en un intervalo que dependerá de la frecuencia que llegan las señales de disparo, 400 Hz.

Por otro lado, integrar todo en un sistema conlleva ser capaz de simplificar lo máximo las conexiones entre subsistemas de laboratorio y realizar un estudio de posibles fallos que pueda haber por pérdidas en cables, características externas como la temperatura del ambiente... Por ejemplo, una tarea de este proyecto consiste en calcular la disipación térmica de un componente específico, pues será necesario conocer la temperatura del ambiente ya que influye en cómo se disipará el calor y afectará gravemente a su vida útil, y por ende, a la vida útil de todo el sistema.

Finalmente, se puede considerar un factor bastante complejo incluir un sistema SCADA que aborde todos los parámetros para que el usuario pueda controlar el sistema de una forma externa sin necesidad de tener conocimientos de codificación Verilog, C y C++, puesto que el objetivo final es tener una interfaz gráfica donde este usuario pueda cambiar los parámetros y que estos se transmitan a través a la FPGA, que modificará el modo de operación de los subsistemas del

laboratorio.

Parte II

Sistema. Hardware y software

Capítulo 1

Selección hardware

Este capítulo aborda la selección y configuración del hardware necesario para el desarrollo y funcionamiento del sistema. Se detallará la elección del servidor, el sistema operativo, las interfaces de entrada/salida, y las características específicas de las FPGA y SoC utilizadas. Además, se discuten las necesidades de disipación térmica y las posibles mejoras para optimizar el rendimiento del hardware. En resumen, este capítulo proporciona una guía detallada sobre la selección y configuración del hardware esencial para el sistema, destacando la importancia de cada componente y sus características específicas para optimizar el rendimiento y asegurar la fiabilidad del sistema en su conjunto.

1.1. Servidor linux

Se utilizará un único ordenador, con sistema operativo Ubuntu 20.04, para la gestión y programación del sistema. Sin embargo, existe la posibilidad de usar un ordenador secundario si fuera necesario. A continuación se describen las ventajas de Ubuntu, como su licencia gratuita, rendimiento, seguridad, personalización y control, y flexibilidad de la línea de comandos.

1. Licencia y Costo

Ubuntu es un sistema operativo de código abierto, lo que significa que es gratuito

2. Rendimiento

Ubuntu tiende a ser más eficiente en términos de recursos, lo que puede llevar a un mejor rendimiento en hardware más antiguo o menos potente.

3. Seguridad

Ubuntu es conocido por su robustez y seguridad, con usuarios que perciben menor vulnerabilidad a malware y virus.

4. Personalización y Control

Ubuntu ofrece un alto grado de personalización y control sobre el sistema operativo, permitiendo a los usuarios avanzados modificar y ajustar diversos aspectos según sus necesidades.

5. Terminal y Línea de Comandos

La flexibilidad y control que ofrece la línea de comandos en Ubuntu es apreciada por usuarios avanzados que están familiarizados con esta interfaz. En cuanto a las características hardware propias del ordenador, se ha utilizado el comando lscpu para mostrar esta información.

```
juancarlos@juancarlos-HP-EliteBook-640-14-inch-G9-Notebook-PC:~$ lscpu
Arquitectura:                         x86_64
modo(s) de operación de las CPUs:     32-bit, 64-bit
Orden de los bytes:                   Little Endian
Address sizes:                        46 bits physical, 48 bits virtual
CPU(s):                               16
Lista de la(s) CPU(s) en línea:       0-15
Hilo(s) de procesamiento por núcleo: 1
Núcleo(s) por «socket»:               12
«Socket(s)»                           1
Modo(s) NUMA:                         1
ID de fabricante:                     GenuineIntel
Familia de CPU:                       6
Modelo:                               154
Nombre del modelo:                    12th Gen Intel(R) Core(TM) i5-1250P
Revisión:                             3
CPU MHz:                              777.046
CPU MHz máx.:                         4400,0000
CPU MHz mín.:                         400,0000
BogoMIPS:                             4224.00
Virtualización:                       VT-x
Caché L1d:                            288 KiB
Caché L1i:                            192 KiB
Caché L2:                             7,5 MiB
```

Figura 1.1: Especificaciones hardware del servidor, comando lscpu. Fuente: propia

Dependiendo de los recursos utilizados, el software puede requerir una gran carga computacional, por lo que es posible que durante algunas pruebas se utilicen otros equipos, en caso de que se requiera comunicación inalámbrica entre ellos, o para verificar que el procesamiento de datos y el control distribuido están funcionando en el equipo anfitrión. La ventaja del sistema de control es la posibilidad de la conexión de forma remota sin necesidad de traslado ni interconexión cableada al equipo de trabajo.

1.2. Interfaz I/O

Como se mencionó en la Parte I, el sistema está compuesto por varios elementos que requieren interconexión física a través de una interfaz de entradas y salidas. Esta interfaz permite la comunicación entre los subsistemas de laboratorio y el resto del sistema, facilitando la conexión entre los subsistemas de laboratorio, tango-controls y la FPGA.

La relevancia de esta parte del sistema radica en las diferentes corrientes y tensiones a las que operan los subsistemas de laboratorio, así como las que puede soportar el sistema de adquisición de datos y control, que en este caso se trata de tensiones de 3.3 V y 5 V. La PCB diseñada debe integrarse perfectamente en el sistema completo, ya que existen otros desafíos, como la disipación térmica y la integridad de la señal, que pueden afectar la fiabilidad del sistema, un factor crucial a considerar.

El diseño de la PCB, que se detallará más adelante en este documento, debe cumplir con ciertas

especificaciones que serán estudiadas minuciosamente para garantizar la seguridad y la velocidad de la comunicación entre subsistemas. Además, uno de los factores más críticos que se analizarán en el diseño de la PCB es la disipación térmica.

1.2.1. Diseño y descripción PCB

El diseño de esta PCB ha sido realizado en el CERN por el Dr. Benjamin Woolley. Posteriormente, fue modificada por el Dr. Daniel Esperante, también con estancia en el CERN en ese momento.

En la parte posterior de la PCB, se encuentra un conector VHDCI de 68 pines al que se conectará la FPGA. La PCB incluye terminaciones serie de 50 ohmios y de alta impedancia, según las necesidades de los subsistemas de laboratorio y dependen del subsistema que se conecte. La alta impedancia se utiliza para minimizar las posibles reflexiones debidas a la longitud del cable empleado. En resumen, el comportamiento de esta PCB está diseñado para facilitar la comunicación entre los subsistemas de laboratorio a través de cableado, interconectándose con la FPGA mediante un conector VHDCI de 68 pines, actuando así como intermediario entre ambos a través de un carril de expansión.

Todos los archivos de fabricación y diseño han sido detallados en la memoria para facilitar su comprensión. Se ha realizado un estudio exhaustivo para entender las funciones de esta PCB. Es importante señalar que no se han realizado modificaciones al diseño original, a pesar de que el esquemático presenta algunas desviaciones de las buenas prácticas de diseño. A continuación, se presenta el esquemático de la PCB. Se observa la falta de un enfoque modular, ya que todos los circuitos están concentrados en una única página en lugar de estar distribuidos en varias. Esto incrementa la dificultad para entender el diseño.

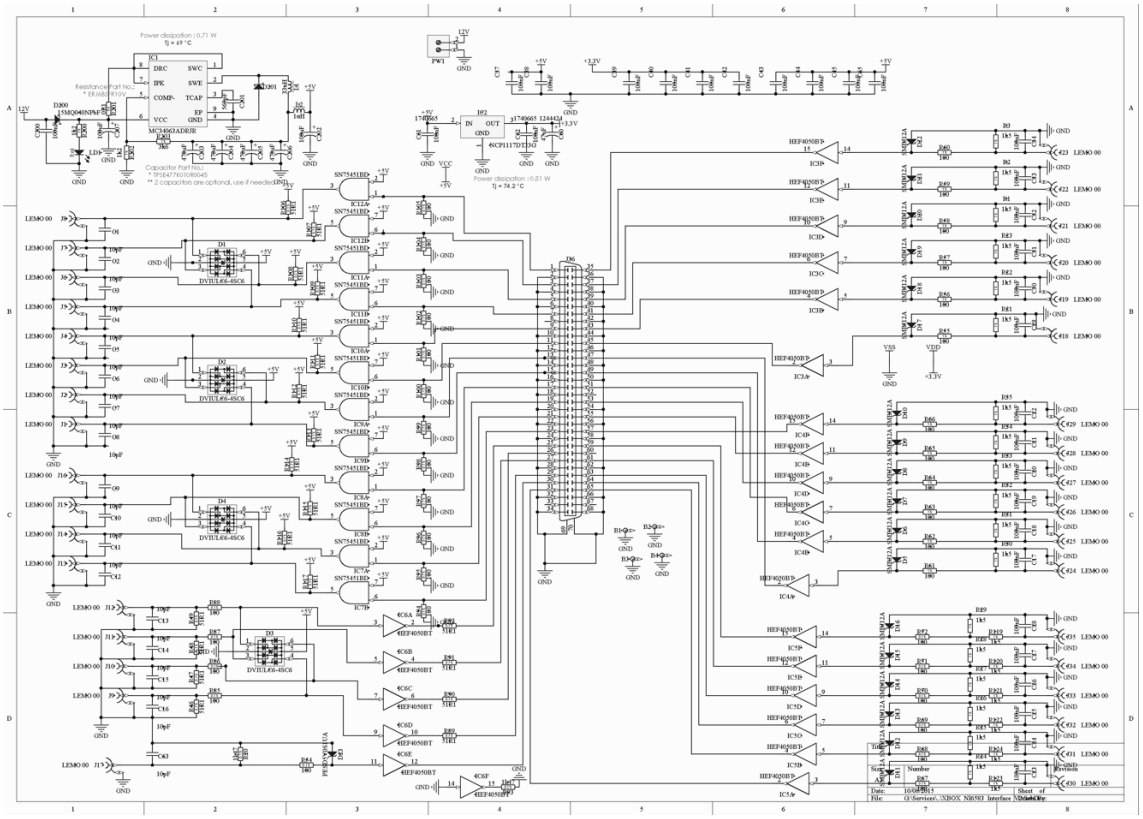
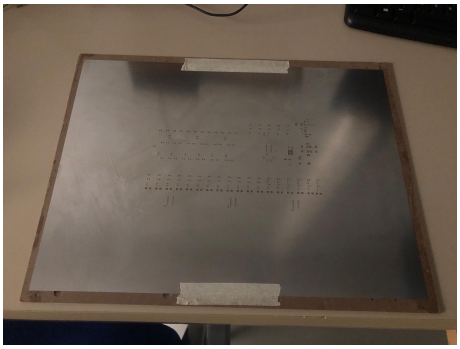


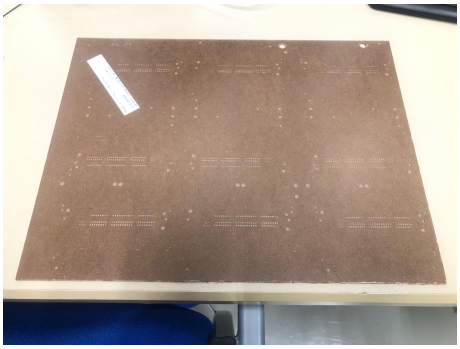
Figura 1.2: Esquemático de la PCB. Fuente: propia

1.2.2. Ensamblado PCB

La soldadura de la PCB se ha realizado en el Laboratorio de Electrónica del IFIC-UV por parte de Manuel López Redondo, técnico de la unidad de electrónica en la Nave Experimental. Para realizar el diseño se ha facilitado el *stencil*, que básicamente es una plantilla de PCB de acero inoxidable con aberturas cortadas con láser. Esta plantilla se utiliza para aplicar pasta de soldadura en ubicaciones específicas de una placa de PCB desnuda, permitiendo que los componentes se coloquen y alineen perfectamente sobre la placa para la colocación de componentes de montaje en superficie. Su función principal es depositar con precisión la cantidad adecuada de soldadura en pasta en los pads SMT para que la unión de soldadura entre el pad y el componente sea perfecta en términos de conexión eléctrica y resistencia mecánica [9].



(a) *Stencil* para la PCB



(b) *Drill* file

Figura 1.3: Stencil y drill files para la soldadura de componentes en PCB. Fuente: propia

1.2.3. Disipación térmica

La disipación térmica es un problema común en todo tipo de aplicaciones que requieren fiabilidad, ya que, con el tiempo, los componentes pueden acortar su vida útil prevista debido a la temperatura. Un claro ejemplo de esto es la CPU de un ordenador, que cuanto mayor temperatura trabaje menor rendimiento y menor vida útil tendrá conforme avance el tiempo.

El factor más importante de esta tarea es el estudio de la disipación térmica para averiguar si es un problema crítico o no. Tal como se puede observar en los esquemáticos, no existe un componente que requiera una frecuencia de operación alta, y por ende, tenga que disipar calor, de hecho no hay ningún tipo de microcontrolador ya que es una PCB para gestionar entradas y salidas con drivers.

Analizando los circuitos, se ha comprobado tanto teórica como experimentalmente la alta disipación térmica proveniente de las resistencias que limitan la corriente a la entrada y salida de los drivers SN7545BP. El funcionamiento principal de este driver es como una puerta AND.

Si en la salida hay un ’0’ lógico, se genera una potencia de 500 mW únicamente en la salida debido a la resistencia de 51 ohmios, por ejemplo, R106. Por otro lado, si en la salida hay un ’1’ lógico, significa que en la entrada (figura 1.4) hay un ’1’ lógico que corresponde a 3.3 V proveniente de la FPGA, lo que resulta en un consumo total de 100 mW debido a la resistencia de 100 ohmios, R105. Sin embargo, el driver consumirá los 100 mW o los 500 mW dependiendo del estado lógico. Cuando hay un ’1’ lógico en la salida, la entrada tiene un ’0’ lógico, lo que significa que no hay corriente a través de la resistencia R105. Por el contrario, si en la entrada hay un ’1’ lógico, fluye corriente por R105 y en la salida habrá un ’0’ lógico, lo que implica que no hay corriente en la resistencia R106.

El peor caso sería cuando se consume 500 mW en las patillas 3 y 5 del mismo componente, resultando en un consumo total de 1 W por driver.

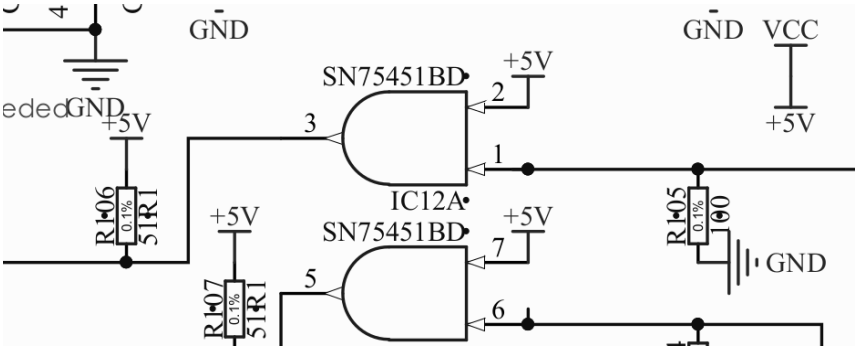


Figura 1.4: Driver SN7545BP. Fuente: propia

Conociendo los parámetros y características de este componente gracias al *datasheet* se pueden hacer estimaciones teóricas de la capacidad de disipación térmica que tiene el driver. Respecto al cálculo teórico, es cierto que existen diversas vías para realizar este cálculo, por ejemplo, potentes programas de cálculo o sencillamente realizar el cálculo a mano. El inconveniente principal del programa de cálculo es que suelen ser de pago, por lo que se tratará de realizar el cálculo de forma convencional y si hubiera complicaciones se hará uso de una hoja de cálculo que simplifique el proceso.

1.2.4. Medidas térmicas

Puesto que la PCB ha sido ensamblada ya, se puede partir con la ventaja de realizar mediciones de forma experimental. Se ha realizado una estimación de forma experimental del driver con el disipador escogido para confirmar que la temperatura a la que llega el componente no sobrepasa los límites absolutos dados por el fabricante.

Para realizar esta medida se ha hecho uso de un termopar tipo K conectado a un multímetro AM-550-EUR que tiene la opción de medir temperatura en ºC.

Figura 1.5: Termopar conectado a multímetro. Fuente: propia

Para realizar las mediciones, el termopar se ha fijado al driver utilizando cinta adhesiva de Kapton. Aunque este método tiene un error de medición considerable (4-5 ºC), resulta fácil de aplicar y retirar. No obstante, un aspecto negativo es que la cinta tiende a despegarse a altas temperaturas. Una ventaja adicional es la reducción de tensiones mecánicas en comparación con la opción de soldar el termopar a un pad del PCB [10]. La temperatura cerca del driver varía según el tiempo de funcionamiento del sistema, ya que la PCB disipa calor no solo de este componente, sino de todos los demás. La temperatura ambiente de la sala de laboratorio donde se encuentra la PCB es de 24 ºC.

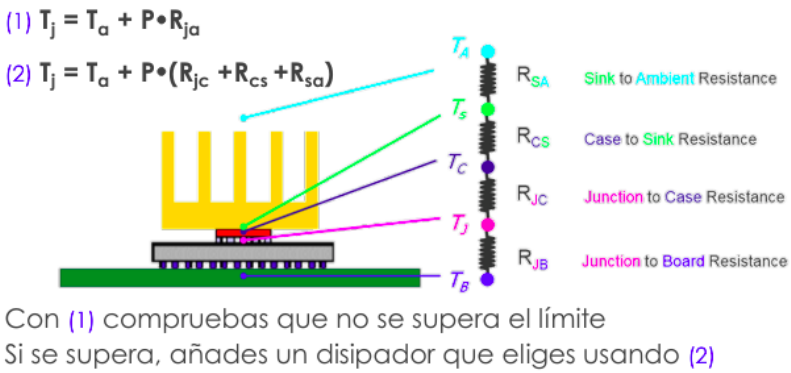


Figura 1.6: Representación de las resistencias térmicas en cada punto del componente electrónico. Fuente: [10]

Se han realizado dos pruebas, una con disipador y otra sin disipador, con el peor caso posible que es cuando el driver disipa la mayor potencia 1 W. Primero se ha realizado la prueba con disipador tras una hora de funcionamiento la temperatura máxima del componente Tc es de 55 ºC.

También se ha medido la temperatura sobre el PCB cerca del driver que es 49 ºC. Por otro lado, al quitar el disipador la temperatura máxima que alcanza el componente Tc es de 59 ºC, mientras que la temperatura del PCB ha disminuido a 46 ºC.

También se ha realizado medidas sin el componente para saber cómo afecta la temperatura de los demás componentes de la PCB al driver cuando éste no disipa nada de potencia. La medida ha resultado en 27 ºC (Ta).

Se puede observar que en el datasheet del componente la temperatura máxima del silicio que puede soportar es de 150 ºC, pero típicamente se suele considerar de 125 ºC, dándole un margen a la temperatura máxima.

1.2.5. Cálculo teórico a nivel de PCB

Se realiza un procedimiento teórico del driver SN7545BP ya que es el componente que se estima que más calor va a disipar, y más problemas de fiabilidad puede presentar. Además de las medidas tomadas en el apartado anterior 1.2.4, se deben de considerar las especificaciones iniciales siguientes:

- $Tjmax$ = 125 ºC
- Rja = 63.7 K/W
- Rjc = 53.6 K/W
- Rjb = 40.8 K/W
- ψ_{JT} = 31.1 K/W
- ψ_{jb} = 40.8 K/W
- $Pmax$ (Ta <25 ºC) = 1000 mW
- $Pmax$ (Ta = 70ºC) = 640 mW
- Rba = 5 - 25 K/W
- Potencia de la PCB = 21.6 W
- Espesor PCB 1.56 mm
- 4 capas Layer StackUp (1 oz de cobre)
- Potencia de driver en rendimiento máximo = 1 W/driver
- $A = 66 - 161\,\text{mm}^2$
- h_{conv} = 5 - 25 W/ m·K

Se va a analizar la disipación en forma de calor que desprenderá el driver para diferentes encapsulados, resistencias térmicas y evaluar si es necesario un disipador o si es necesario realizar un futuro ajuste en el diseño de las capas de la PCB para mejorar la conducción lateral y/o transversal. Se estudian varias alternativas para conocer el mejor resultado de disipación térmica y menor temperatura que alcance el componente para asegurar la mejor fiabilidad del mismo.

- **DIP con socket y con disipador**

  El driver va a desprender al ambiente una temperatura en funcionamiento de 53.6 K/W a 1 W serán 53.6 ºC. De las medidas térmicas que se han realizado en el apartado anterior la temperatura máxima a la que ha llegado el componente es de 55 ºC (Tc).

$$T_j = T_c + P \cdot R_{jc} = 55 + 1 \cdot 53.6 = 108.6\,^{\circ}\text{C} \tag{1.1}$$

  La temperatura en el silicio es de 108.6 ºC, lejos aún del máximo que es 125 ºC. Sin embargo, que haya un margen tan pequeño es negativo para la vida útil del componente. El driver como se puede visualizar en la figura irá en un *socket* soldado a la PCB. Esto perjudica de forma crucial su disipación de calor ya que no es un componente SMD que utilice la PCB como principal disipador sino que ahora tiene una capa de aire en su parte inferior.

  De forma analítica, considerando el peor caso donde el calor se disipa únicamente hacia arriba sin considerar que se disipe por la parte inferior hacia el PCB, la resistencia térmica entre encapsulado y ambiente es Rca = 87 K/W del disipador y se le suma Rjc = 53.6 K/W del driver siendo el resultado de 140.6 K/W.

$$T_j = T_c + P \cdot R_{jc} = 55 + 1 \cdot 140.6 = 195.6\,^{\circ}\text{C} \tag{1.2}$$

  Comparando el valor analítico y experimental se puede observar que hay una diferencia de 87 ºC entre 108.6 y 195.6, por lo que el calor no se está disipando únicamente hacia arriba, sino que habrá conducción lateral y hacia la PCB. A partir de las medidas experimentales se puede calcular Rja ya que los cálculos analíticos no son consistentes. Considerando que Tj es 108.6ºC y Ta es 27 ºC.

$$R_{ja} = T_j - T_a = 108.6 - 27 = 81.6K/W \tag{1.3}$$

  Por otro lado, se puede considerar que la resistencia térmica total de este componente se puede calcular conociendo la temperatura alcanzada a 1 W es de 55 ºC y cuando no hay componente es de 27 ºC (datos del apartado anterior 1.2.4). En condiciones iguales (solo es válido cuando las condiciones son las mismas para cualquier caso) hay una diferencia de 28 ºC, es decir, está aumentando la temperatura Ta en 28 ºC cuando está funcionando el driver. Es cierto que esta temperatura puede cambiar dependiendo de las condiciones del entorno, pero puede servir como referencia para entender cuánta temperatura está aumentando debido al funcionamiento de este driver.

(a) SN7545BP **(b)** Socket

Figura 1.7: Driver y socket donde irá encapsulado el componente. Fuente: mouser

- **DIP con socket y sin disipador**

  Otra opción sería considerar que no haya disipador encima del componente, la resistencia térmica entre encapsulado y ambiente Rca sería:

$$R_{CA} = \frac{1}{h_{conv} \cdot A} = \frac{1}{10 \cdot 0.000066} = 1515.15\,\text{K/W} \tag{1.4}$$

  Como se indica en las especificaciones iniciales, Rca puede cambiar dependiendo del área a considerar y de la convección natural del aire ($hconv$). Esto se debe a que si se considera el área del componente sin las partes laterales del mismo es de $66\,\text{mm}^2$, pero si se tiene en cuenta las caras laterales será de $161\,\text{mm}^2$ por lo que la resistencia térmica cambia. El resultado aplicando la misma ecuación 1.4 considerando las caras laterales es de 440 K/W. Esto se debe a que hay más superficie en la zona lateral que en la superficie, en la cara superior hay $66\,\text{mm}^2$, mientras que las caras laterales es de $95\,\text{mm}^2$. Incluso si ($hconv$) fuera mayor a 10 (el rango que se estima es de 5 a 25) en la circulación del aire de nuestro sistema tendría una resistencia térmica aún menor. Para salir de dudas se puede verificar con las medidas térmicas realizadas.

  ¿Se puede considerar que el calor se disipa únicamente hacia arriba desde el encapsulado al ambiente? ¿Hay otro camino de disipación de calor? ¿Qué otro camino hay y cómo afecta a la resistencia térmica?

  El driver va a desprender al ambiente una temperatura en funcionamiento de 53.6 K/W a 1 W serán 53.6 ºC. De las medidas térmicas que se han realizado en el apartado anterior la temperatura máxima a la que ha llegado el componente es de 59 ºC (Tc).

$$T_j = T_c + P \cdot R_{jc} = 59 + 1 \cdot 53.6 = 112.6\,^{\circ}\text{C} \tag{1.5}$$

  Se han realizado las medidas en el apartado anterior para confirmar que la temperatura máxima es aproximadamente de 112.6 ºC (Tj) en el peor caso, pero se esperan 2000 ºC o 440 ºC según los cálculos de forma analítica. Esto es debido a que no todo el calor se disipa de encapsulado al ambiente sino que hay una parte que se disipa hacia la PCB.

  Rja en la hoja de fabricante es de 63.7 K/W. Sin embargo, el mismo fabricante indica que está obsoleto este procedimiento de cálculo con las resistencias térmicas [11]. A partir de esto y de que los cálculos realizados de forma analítica no coinciden con las medidas experimentales, se consideran inconsistentes los cálculos realizados analíticamente y se valoran las medidas experimentales. Por lo tanto, se puede calcular Rja, considerando que Tj es 112.6 ºC, Ta es 27 ºC y la potencia es 1 W.

$$R_{ja} = T_j - T_a = 112.6 - 27 = 85.6K/W \tag{1.6}$$

  Por otro lado, la resistencia térmica total de este componente se puede calcular conociendo la temperatura alcanzada a 1 W es de 59 ºC y cuando no hay componente es de 27 ºC (datos del apartado anterior 1.2.4). En condiciones iguales (solo es válido cuando las condiciones son las mismas para cualquier caso) hay una diferencia de 32 ºC. Como se ha indicado anteriormente, es cierto que la temperatura puede cambiar, pero puede servir como referencia para entender cuánta temperatura está aumentando debido al funcionamiento de este driver.

La diferencia entre cuando hay disipador o no es de 4 ºC. Por cada 10 ºC de aumento de temperatura se duplica la probabilidad de fallo de un componente, luego cada ºC que se pueda mejorar es crucial para un buen diseño [10]. Añadir un disipador no es tan útil, pero ayuda en la disipación de calor ya que aumenta la superficie. El valor más óptimo de resistencia térmica del disipador debe ser menor que 10.1 K/W. Sin embargo, no hay disipadores con resistencias térmicas tan bajas para componentes tan pequeños.

- **SOIC (SMD)**

  Otra alternativa es el encapsulado de este driver en SMD, pero se puede visualizar en el propio *datasheet* del componente (figura 1.8) que las resistencias térmicas de esta versión son mayores que en DIP, esto significa que se calentará más. Aunque SMD tenga la ventaja de quitar esa capa de aire (el aire es un mal conductor del calor) que tiene el DIP entre encapsulado y PCB, las resistencias térmicas son considerablemente elevadas como para tener en cuenta esta opción. El valor Rba puede ser variable dependiendo del diseño, fabricación y materiales de la PCB, pero se estima un valor entre 5 y 25 K/W [12]. Se va a fijar en un valor de 15 K/W [13]. Las vías de disipación de calor se calculan en paralelo hacia la PCB (R_{ja1}) y hacia el ambiente, es decir, hacia arriba (R_{ja2}).

$$R_{ja1} = R_{jb} + R_{ba} = 62.4 + 15 = 77.4K/W \tag{1.7}$$

$$R_{ja2} = 122.2K/W \tag{1.8}$$

$$R_{ja} = \frac{R_{ja1} \cdot R_{ja2}}{R_{ja1} + R_{ja2}} = \frac{77.4 \cdot 122.2}{77.4 + 122.2} = 47.38K/W \tag{1.9}$$

7.3 Thermal Information

THERMAL METRIC[1]		SN7545xB			UNIT
		D (SOIC)	P (PDIP)	PS (SO)	
		8 PINS	8 PINS	8 PINS	
$R_{\theta JA}$	Junction-to-ambient thermal resistance	122.2	63.7	119.6	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	68.4	53.6	71.5	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	62.4	40.8	68.7	°C/W
ψ_{JT}	Junction-to-top characterization parameter	23.2	31.1	31.6	°C/W
ψ_{JB}	Junction-to-board characterization parameter	62.0	40.8	67.7	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report.

Figura 1.8: Resistencias térmicas de diferentes encapsulados. Fuente: [14]

- **DIP sin socket y sin disipador**

  Una alternativa es utilizar el encapsulado DIP directamente soldado a la PCB, sin *socket*, y evaluar la necesidad de un disipador térmico. Esta configuración ofrece la menor resistencia térmica entre el componente y la PCB al eliminar la capa de aire entre ellos y evitar el plástico del *socket*, que no es eficiente para la disipación de calor. Realizando cálculos preliminares sin disipador, se puede estimar la resistencia térmica total entre el encapsulado y el ambiente, considerando las vías de disipación de calor en paralelo, hacia la PCB (R_{ja1}) y hacia el ambiente, es decir, hacia arriba (R_{ja2}).

$$R_{ja1} = R_{jb} + R_{ba} = 40.8 + 15 = 55.8K/W \tag{1.10}$$

$$R_{ja2} = 63.7K/W \tag{1.11}$$

$$R_{ja} = \frac{R_{ja1} \cdot R_{ja2}}{R_{ja1} + R_{ja2}} = \frac{55.8 \cdot 63.7}{55.8 + 63.7} = 29.74K/W \tag{1.12}$$

- **DIP sin socket y con disipador**

  La resistencia térmica entre el silicio y ambiente es de 29.74 K/W aproximadamente, pero si se pone un disipador térmico se puede incluso mejorar. Hay que considerar que es un chip bastante pequeño y cuanto más pequeño es, mayor será la resistencia térmica del disipador implica a que el calor no se disipe por ahí. De hecho si se coloca una resistencia térmica muy grande el calor dejará de disiparse por arriba y tenderá a disiparse una mayor parte por la PCB.

  Los disipadores ayudan a la disipación del calor debido a que aumentar la superficie del componente contribuyen en una mayor área de disipación. Sin embargo, a partir de los datos del fabricante no se pueden realizar estimaciones ya que los resultados son inconsistentes puesto que cuando el disipador es mayor a 10.1 K/W la resistencia térmica total del componente será mayor que cuando no hay disipador.

  Para estimar la resistencia térmica real se deberían realizar medidas experimentales con un termopar como se realizó en el apartado 1.2.4.

1.2.6. Posibles mejoras y soluciones

Se han realizado los cálculos de disipación térmica con diferentes alternativas. La conclusión que se ha sacado es que el encapsulado deberá ser DIP ya que las resistencias térmicas del SMD (SOIC) son elevadas.

En relación al encapsulado DIP, se debe seleccionar aquel caso en el que la resistencia térmica total sea mínima, ya que de esta manera la temperatura alcanzada por el componente será menor dado un nivel de potencia aplicada. Según los cálculos realizados en la sección anterior (1.2.5), la opción de utilizar un encapsulado DIP sin disipador y sin socket resulta en la menor resistencia térmica total calculada.

No obstante, el uso de un disipador siempre contribuye a mejorar la disipación térmica. En este caso particular, no se ha podido concluir que el uso de un disipador sería beneficioso, debido a que la única manera de confirmar esta afirmación sería mediante pruebas experimentales. Además, actualmente no hay disponibles en el mercado disipadores con una resistencia térmica inferior a 10.1 K/W para las dimensiones requeridas.

Por consiguiente, existen algunas opciones para reducir aún más la temperatura del silicio del componente. Se deberá disipar el calor de manera más eficiente.

- Mejorar conducción lateral del PCB
- Mejorar ubicación componentes
- Disipador térmico

- Añadir chasis o estructura para disipar calor de la PCB hacia ambiente fuera del chasis
- Convección forzada

A pesar de que puedan ser opciones muy eficaces, se descartan la primera y segunda opción, ya que la PCB ha sido diseñada por otros ingenieros y no se trata de rediseñarla. Como ya se ha comentado, el disipador será efectivo si la resistencia térmica es menor de 10.1 K/W, que no es común para esas dimensiones. Por lo tanto, para una posible mejora en el diseño, se debería quitar el socket y, en cuanto al disipador hacer medidas experimentales para verificar si es necesario o no. Sin embargo, para este proyecto, se mantendrá el socket y el disipador, ya que funcionan correctamente y están soldados, aunque es posible que su vida útil se acorte. Otra alternativa a considerar en el futuro es añadir un chasis que mejore la conducción de calor y/o convección forzada.

1.3. FPGA y SoC

Xilinx desarrolló la primera FPGA en el año 1984, aunque no eran llamadas FPGAs hasta el año 1988 aproximadamente. A lo largo de 30 años, la FPGA ha aumentado la capacidad en un factor 10000. Por otro lado, tanto el coste y el consumo ha disminuido más de un factor 1000 [15].

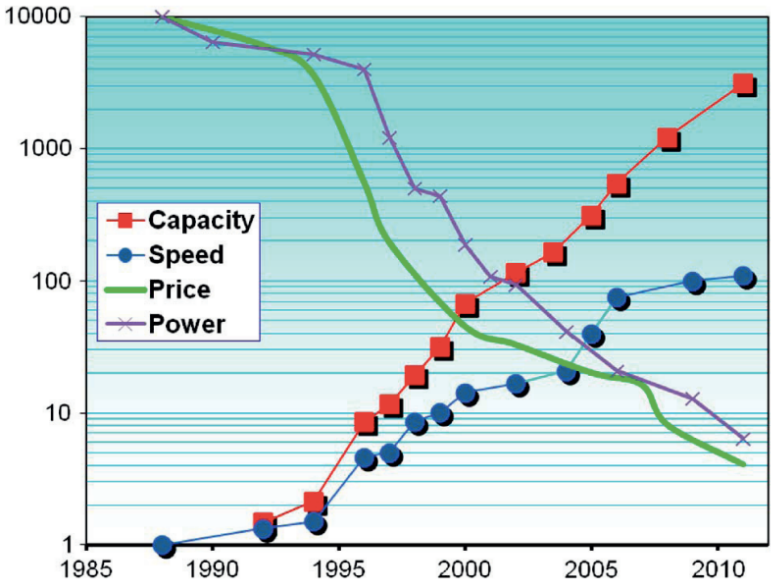


Figura 1.9: *Capacity* son el número de celdas lógicas. *Speed* es el rendimiento. *Price* por celda lógica. *Power* por celda lógica. *Price* y *power* son escalados en un factor 10000. Fuente: [15]

La producción de FPGAs se divide normalmente en dos grandes empresas, Altera y Xilinx. Cabe destacar que Altera fue comprada por Intel en 2015 y Xilinx por AMD en 2020, aunque Altera opera de forma independiente desde el 1 de marzo de 2024. En este proyecto se ha utilizado la electrónica de Xilinx.

Por otra parte, SoC (*System-on-Chip*) es un circuito integrado que integra todos los componentes de un sistema electrónico en un único chip. Esto puede incluir una CPU, memoria, interfaces

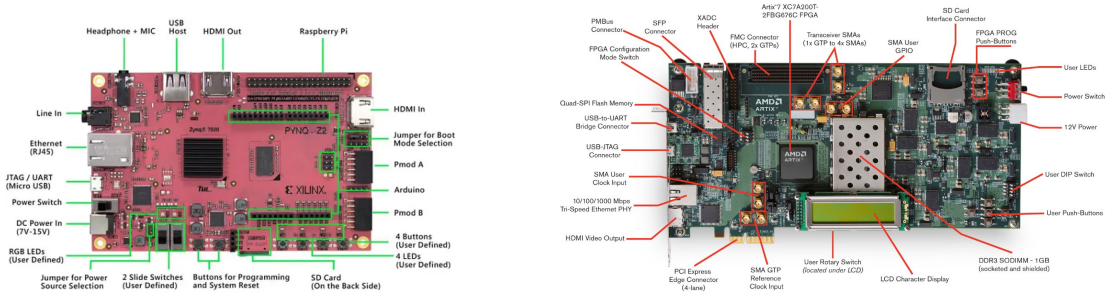
de entrada/salida, almacenamiento, y componentes especializados como aceleradores de hardware y controladores. Puede manejar tareas generales de procesamiento, ejecuta sistemas operativos como Linux, y gestiona la lógica de alto nivel. El SoC es la combinación de PS (procesador de propósito general) y PL (lógica programable) que integra un sistema embebido en una FPGA.

La unión de una FPGA con un SoC proporciona lo mejor de ambos mundos: la flexibilidad y potencia del procesamiento embebido y la capacidad de personalizar la lógica digital con la FPGA.

Debido a la rentabilidad económica y rendimiento de la FPGA se ha optado por utilizar una de ellas para este trabajo. A pesar de la gran variedad de FPGAs, tanto de marca como el propio modelo, se ha llevado un estudio exhaustivo para averiguar cuál se ajusta a nuestras necesidades.

No es necesaria una gran frecuencia de operación, ya que la máxima frecuencia que se trabajará es del orden de 400 Hz y en cuanto a carga computacional no se necesita una gran cantidad de recursos ya que no se realizará procesado de señal.

Como se ha comentado en la Parte I de Introducción, el sistema de control incluirá un el procesado de señales y la adquisición de datos que, aunque en este proyecto no se realice, se tiene en consideración para futuros proyectos en los que habrá que gestionar y procesar elementos como vector modulador o modulación IQ.



(a) Pynq-z2 **(b)** AC701

Figura 1.10: FPGAs utilizadas en este trabajo fin de máster. Fuente: Xilinx

Tras el estudio y disponibilidad de FPGAs y SoCs en el mercado, la velocidad y los requisitos para este proyecto pueden ser conseguidos con prácticamente cualquier FPGA, por lo que se ha decidido trabajar principalmente con la placa de desarrollo pynq-z2, aunque algunas pruebas se han realizado con la AC701 (Artix 7). La placa pynq-z2 tiene una frecuencia de reloj menor que la AC701, pero incluye un SoC XC7Z020-1CLG400C (Dual ARM® Cortex™-A9 MPCore™ with CoreSight™), generalmente denominado zynq-7020, que nos posibilita tener sistema operativo y otras funcionalidades que se verán en los próximos apartados. Con respecto a ambos datasheet, la máxima frecuencia de operación de la pynq-z2 es de 650 MHz, mientras que de la AC701 es de hasta 810 MHz.

Trabajar con distintas placas representa un desafío, no solo por las diferencias en las características hardware, sino también en los métodos de programación y configuración. En este caso, como se explicará y justificará más adelante, el software principal utilizado para el diseño de bloques y programación digital es Vivado.

Un ejemplo ilustrativo es que la pynq-z2 tiene un ARM incluido ya, mientras que como la AC701 no cuenta con esta integración, la configuración y diseño de bloques en Vivado será algo

más compleja ya que habrá que añadir el bloque Microblaze que sirve para crear una instancia de un procesador dentro de su diseño FPGA.

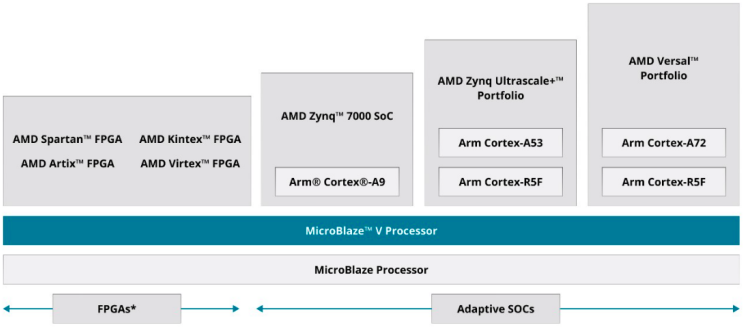


Figura 1.11: Esquema SoC-FPGA. Fuente:

En el caso de la pynq-z2, que incluye un SoC (System on Chip), presenta algunas características notables:

1. **Lógica programable (PL/FPGA):** Una matriz de bloques lógicos configurables que permiten a los diseñadores implementar lógica digital personalizada.
2. **Procesador(s) de propósito general (PS):** Uno o más núcleos de procesadores embebidos, como ARM Cortex-A9 o Cortex-A53, que proporcionan capacidad de procesamiento de propósito general.
3. **Memoria:** Puede incluir memoria RAM, memoria caché y otros tipos de memoria para soportar las necesidades de almacenamiento y acceso rápido a datos.
4. **Periféricos:** Diversos periféricos como interfaces de comunicación (UART, SPI, I2C), controladores de interrupciones, controladores de memoria, entre otros.

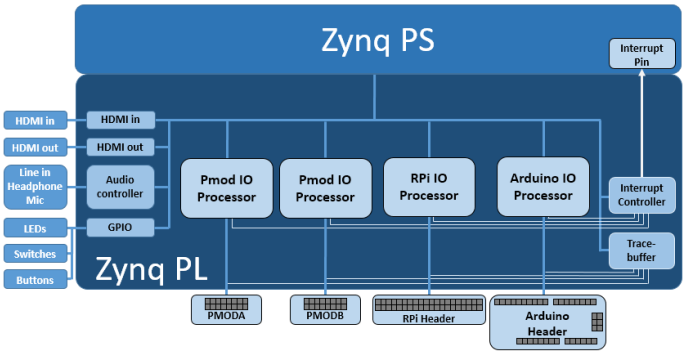


Figura 1.12: Esquema PL/PS Zynq (Pynq). Fuente: [16]

La combinación de estos elementos en un solo chip facilita la implementación en sistemas embebidos complejos. El procesador de propósito general puede ejecutar un sistema operativo, manejando tareas de control y gestión, mientras que la lógica programable de la FPGA permite la implementación de funciones específicas del usuario o aceleración de hardware. Esta arquitectura

SoC se utiliza en una variedad de aplicaciones, como sistemas de comunicación, vídeo, procesamiento de señales, redes, y otros sistemas embebidos de alto rendimiento. Proporciona un equilibrio entre flexibilidad y rendimiento, ya que combina la capacidad de reconfiguración de la FPGA con la potencia de procesamiento de un procesador embebido.

Como se ha comentado anteriormente, la pynq-z2 es la FPGA en la que se ha desarrollado todo el trabajo a excepción de alguna prueba que se realizó con la AC701, por ello de aquí en adelante, cuando se nombre FPGA será por defecto la pynq-z2.

Capítulo 2

Selección software

El propósito de este capítulo es detallar la elección y configuración del software necesario para el desarrollo del proyecto. Se aborda la selección de herramientas y entornos de desarrollo que permitirán la programación y gestión de las FPGAs, así como la implementación de sistemas embebidos y controladores. Además, se exploran las soluciones de software abierto para sistemas SCADA y DCS, destacando las capacidades y funcionalidades que optimizarán la integración del hardware con el software.

2.1. Vivado

AMD e Intel, como se ha mencionado anteriormente, son dos de las empresas más importantes en el mercado de FPGAs. Cada una de ellas tiene su propio software de diseño. Intel utiliza Quartus, mientras que AMD utiliza Vivado, que anteriormente era conocido como Xilinx ISE. En este proyecto específico, se ha utilizado Vivado debido a que las placas son de Xilinx.

La versión que se ha utilizado en este proyecto es Vivado 2021.2.1. Vivado es un entorno de diseño integrado que incluye herramientas de diseño a nivel de sistema electrónico para sintetizar y verificar IP basada en C. Tiene distintos componentes que facilitan y ayuda al diseñador a configurar su sistema, como por ejemplo Vivado Simulator, Vivado IP Integrator o Vivado Tcl Store.

La herramienta encargada del desarrollo de aplicaciones aceleradas en hardware y software integrado para una gran variedad de plataformas Xilinx es Vitis, una plataforma de software unificado. En versiones anteriores era SDK, pero en la versión de Vivado 2020.1 deja de utilizar SDK y lo reemplaza con Vitis Unified Software Development. Desde esta versión, Xilinx ha integrado Vitis como la herramienta principal para el desarrollo de software, unificando y extendiendo las capacidades del SDK anterior.

2.2. PYNQ y Sistema Operativo Embebido

PYNQ™ es un proyecto de código abierto de AMD® que facilita el uso de plataformas de Computación Adaptativa. De esta forma, utilizando el lenguaje Python, *Jupyter Notebooks* y el enorme

ecosistema de bibliotecas Python, se puede aprovechar los beneficios de la lógica programable y los microprocesadores para construir sistemas electrónicos más capaces [17].

PYNQ™ se puede utilizar para crear aplicaciones de alto rendimiento que incluyen:

- Ejecución paralela de hardware.
- Procesamiento de video de alta velocidad de fotogramas.
- Algoritmos acelerados por hardware.
- Procesamiento de señales en tiempo real.
- E/S de alto ancho de banda.
- Control de baja latencia.

Utilizando estas capacidades, PYNQ permite a los diseñadores desarrollar soluciones innovadoras y eficientes, aprovechando la sinergia entre la lógica programable y los microprocesadores, todo ello mediante un entorno de desarrollo accesible y flexible.

En este caso, se va a hacer uso de la herramienta PYNQ para utilizar únicamente el sistema operativo Ubuntu 22.04 sobre la placa de desarrollo pynq-z2. Se fusiona la configuración hardware creada en Vivado de la FPGA con el sistema operativo, que permitirá acceder a las posiciones de memoria de las variables desde un sistema SCADA (tango-controls) con interfaz gráfica.

La pynq-z2 tiene varios modos de funcionamiento, uno de ellos corresponde es para *boot* desde la tarjeta micro-SD, para ello se ha de cargar la imagen de la pynq-z2 en esta tarjeta y configurar la placa de forma que inicie por la tarjeta micro-SD, además de conectar cable de alimentación y cable ethernet que irá conectado al ordenador.

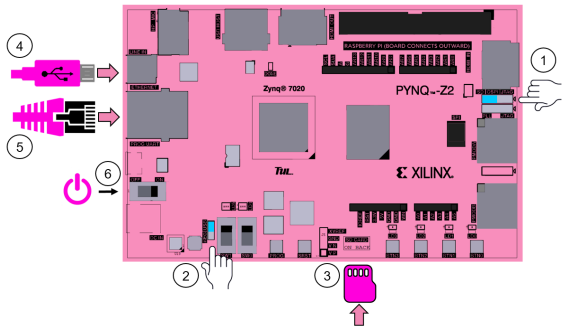


Figura 2.1: Setup de pynq-z2 para iniciar por la SD. Fuente: tango-controls

Se deberá de realizar la conexión de red y conexión a internet para actualizar la placa y descargar nuevos paquetes necesarios como python3-itango, python3-tango, tango-common y tango-db. Para establecer esta conexión se ha de añadir una dirección IP estática que corresponde con la IP de la pynq, en este caso se ha realizado a través de comandos en la terminal de linux.

```
sudo apt-get update
sudo nano /etc/netplan/01-netcfg.yaml
sudo nano netplan apply
```

Listing 2.1: Comandos utilizados

Tras realizar la conexión de forma física con la placa se abre PuTTY, un cliente SSH gratuito, que nos permite enviar y recibir datos a la pynq-z2 por el puerto serie. La ventaja de abrir Putty es que se puede visualizar todos los pasos de arranque de *boot* del Sistema Operativo y comprobar que finalmente, si todo ha ido bien, manda un mensaje de finalización del arranque.

El proyecto base que trae por defecto tiene una combinación de LEDs que indican que se ha cargado el bitstream de la configuración hardware y además se encenderá el LED de *DONE* que indica que el .bit (bitstream) se ha cargado de forma correcta. Más adelante se deberá realizar un cambio de fichero dentro de la imagen, específicamente el .bit. Es un fichero que generará el proyecto de vivado con el Diseño de Bloques y la configuración del sistema diseñado, incluyendo los *Custom Core IP* de las señales de disparo, interbloqueo y registro.

La principal ventaja de esta configuración es que PYNQ es de código abierto, igual que *Jupyter Notebook* permiten iniciar *Interactive Python (iPython)* kernel y un servidor web directamente en el procesador ARM del dispositivo Zynq. El servidor web gestiona el acceso al núcleo a través de un conjunto de herramientas basadas en el navegador que ofrecen un panel de control, terminal bash, editores de código y Jupyter notebooks.

2.3. Tango-Controls

2.3.1. Descripción general

Otro elemento necesario para el sistema completo es un sistema de control distribuido para gestionar tanto el hardware como el software, facilitando la adquisición y control de la información. Tango-controls es una solución de software abierto para sistemas SCADA (*Supervisory Control and Data Acquisition*) y DCS (*Distributed Control Systems*), que utiliza el protocolo de comunicación CORBA.

La dificultad de programación y entorno se ajustará en base al nivel de detalle de la aplicación. La curva de aprendizaje para la puesta en marcha y el dominio de las herramientas es algo complicado ya que hay una gran cantidad de herramientas que hay que conocer, siendo muchas de ellas independientes entre sí. Sin embargo, al poner el sistema en marcha, todas estas herramientas deben estar conectadas e integradas adecuadamente.

Dependiendo de la aplicación, lenguaje de programación, complejidad de la interfaz gráfica... se podrán utilizar distintas herramientas siendo algunas de ellas análogas entre sí o con pequeños cambios.

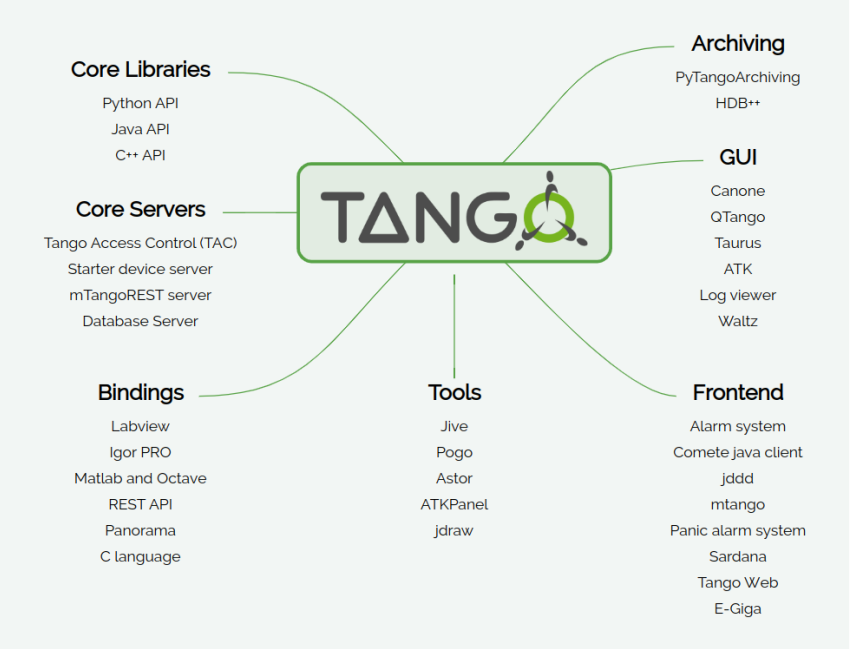


Figura 2.2: Herramientas de tango-controls. Fuente: tango-controls

Uno de los objetivos ha sido encontrar las herramientas más eficientes para el equipo de trabajo. Es por ello que se ha dedicado una gran parte del tiempo en la puesta en marcha del sistema para la elección de las herramientas.

El funcionamiento de tango-controls se fundamenta en una base de datos, una aplicación cliente y servidores de los subsistemas de laboratorio (en adelante, *Device Server*). Tango-controls se estructura en torno al concepto de *Device Server*, que son dispositivos implementados como objetos orientados y servicios orientados a software. Cada *device* opera mediante comandos, atributos, eventos y propiedades, pudiendo alcanzar el nivel de complejidad que la aplicación requiera.

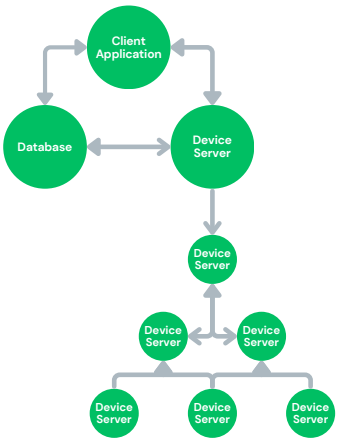


Figura 2.3: Todas las soluciones de tango-controls reducida a 3 simples procesos. Fuente: tango-controls

El sofware tango-controls es un sistema de control que deberá de proveer acceso de red y a internet. El acceso al propio hardware está gestionado por el *Device Server* que contiene *Devices* pertenecientes a diferentes *Device Classes* que implementan de forma directa el acceso al hardware. El proceso de adquisición y transferencia de información se basa en que el cliente importa los *Devices* mediante la base de datos y envían pedidos a los *Devices* usando tango-controls. La base de datos utilizada es MySQL en la que los *Devices* guardan la configuración y valores por defecto de forma permanente.

2.3.2. Herramientas

La instalación de las principales herramientas se han realizado en Linux ya que facilita el trabajo a la hora de compatibilidades de librerías ya que Linux ofrece mayor flexibilidad como sistema operativo. El proceso de instalación se debe seguir cautelosamente para poner en marcha la base de datos mariadb y su configuración. Un aspecto importante a tener en cuenta es establecer la variable de entorno correcta, por defecto se trabaja en *localhost*.

```
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
export TANGO_HOST=localhost:10000

^G Ver ayuda   ^O Guardar     ^W Buscar      ^K Cortar Texto ^J Justificar  ^C Posición
^X Salir       ^R Leer fich.  ^\ Reemplazar  ^U Pegar        ^T Ortografía ^_ Ir a línea
```

Figura 2.4: Exportación de la variable de entorno TANGO_HOST. Fuente: propia

La variable de entorno TANGO_HOST debe establecerse de la forma: TANGO_HOST = HOSTNAME:PORT, donde HOSTNAME es el nombre del ordenador y PORT es el puerto en el que el servidor esperará solicitudes. Esto se utilizará para enviar la solicitud de tango-controls.

En la figura 2.5 se pueden visualizar las herrameintas que Tango-controls proporciona. Están escalonadas en capas desde bajo a alto nivel.

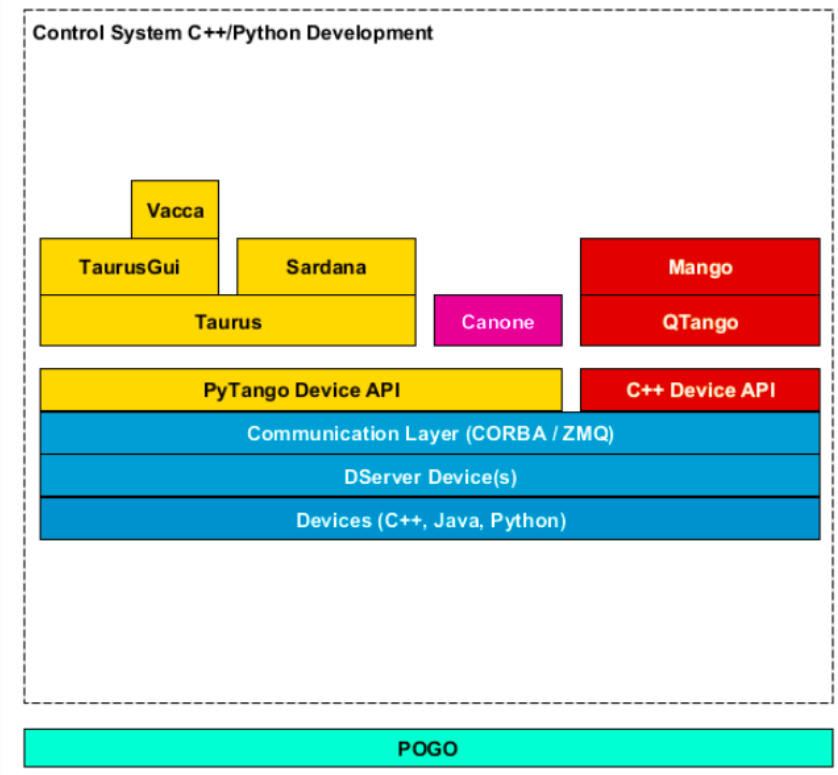


Figura 2.5: Herramientas de tango-controls. Fuente: [18]

Como se puede visualizar existen dos caminos que están condicionados por el lenguaje de programación a elegir: python o C++. También hay otro camino que es utilizar java como lenguaje de programación. La imagen muestra capas de bajo a alto nivel de programación, siendo lo más alto Vacca, TaurusGui, Sardana o Mango, son interfaces gráficas, mientras que Pogo o Devices es para programación a bajo nivel.

- **Pogo (*Class Generator*)** permite generar código generado en C++, Java o Python, es una opción que el usuario puede elegir. Proporciona propiedades, comandos, atributos, estados y tuberías (*pipes*) a modo de plantilla para que el usuario pueda realizar cambios.
- **Devices** se puede programar en diferentes lenguajes como java, C++ o Python. Realmente es un concepto abstracto, y en la realidad puede ser parte de hardware del sistema que se quiera controlar. Una forma de programar cada *Device* es elegir que cada uno de ellos es un componente del sistema.
- **DServer Device** es una aplicación web que hace puente desde http:// a tango://.

- **CORBA/ZMQ** es la capa de comunicación CORBA (Common Object Request Broker Architecture). CORBA permite la comunicación entre *software* ya que facilita la comunicación entre aplicaciones distribuidas en una red heterogénea. Sin embargo, no es tan necesario saber cómo funciona CORBA para diseñar un SCADA en tango-controls.
- **Device API** simplifica todo lo anterior en lenguaje Python, C++ o Java. Sustituye a CORBA y todas las herramientas de bajo nivel anteriores. Es muy útil, pero hay que tomar la decisión de qué lenguaje de programación utilizar.
- **Alto nivel** son todas las demás herramientas que están en la parte superior de la figura 2.5. Sirven principalmente para realizar interfaces gráficas. Puede parecer menos interesante ya que existen muchas formas de desarrollar una interfaz gráfica, pero en este caso va ligado a tango-controls.

En cada capa existen herramientas que son programas muy útiles con los que se va a trabajar y simplificarán el trabajo.

- **Astor** permite conectar varios servidores tango-controls a distancia, es decir, 2 servidores tango-controls con subsistemas de laboratorio independientes y a distancia puede ser controlado por un único ordenador sin que esté físicamente conectado.
- **Jive** es una aplicación en java diseñada para explorar y editar la base de datos estática de TANGO. En jive se gestionan y crean dispositivos, propiedades y clases. Jive también ofrece funciones avanzadas de búsqueda/selección. Para la puesta en marcha, se debe de crear una clase en Pogo para introducir en Jive. Además, Jive proporciona una interfaz gráfica llamada ATKPanel.
- **ATKPanel** es una aplicación de panel de control genérico. Puede ser utilizada para controlar cualquier dispositivo Tango. Soporta la mayoría de las características y tipos de datos de Tango. Con AtkPanel, puedes visualizar y configurar valores de atributos, ejecutar comandos, probar el dispositivo y acceder a datos de diagnóstico.
- **Taurus** es un entorno de trabajo en Python para el control y sistema de adquisición de datos CLIs y GUIs en entornos científicos o industriales.
- **Sardana** es un paquete para la supervisión, control y adquisición de información. Es un software más reciente por lo que no hay mucha documentación, pero las herramientas que se utilizan sobre Sardana deberán ser previamente configuradas desde Taurus.

En resumen, tango-controls permite utilizar distintas herramientas y lenguajes de programación. Para la puesta en marcha de un *Device Server* serían necesarias solamente dos herramientas (Pogo y jive), pero es recomendable una interfaz gráfica apropiada para nuestra aplicación ya que, aunque jive proporcione una pequeña interfaz gráfica, es bastante simple y no proporciona suficiente información para un sistema complejo. Por ello, la creación de la interfaz gráfica se realizará con Taurus.

2.3.3. Test

Tango-controls proporciona un ejemplo llamado TangoTest, es un ejemplo para visualizar que se ha instalado bien y ofrece ayuda para la puesta en marcha de un *Device Server*.

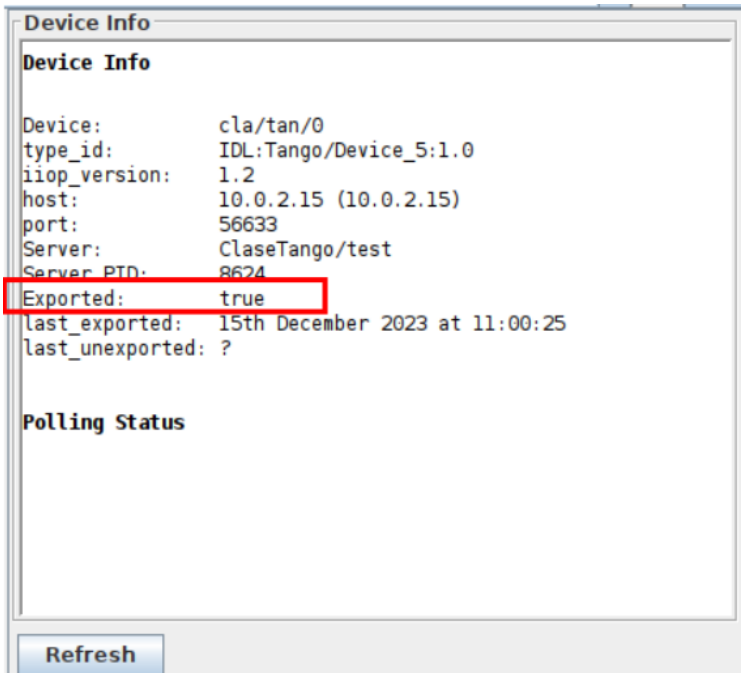


Figura 2.6: Prueba de puesta en marcha de *Device* en jive. Fuente: propia

No obstante, es un ejemplo sin aplicación real, es decir, no es posible controlar ni ver cambios tangibles, más que una simple simulación de una posible aplicación. Por consiguiente, se ha realizado una prueba midiendo la temperatura del ambiente con un sensor NTC. Está conectado a la placa pynq-z2 por medio de la salida analógica A0 y alimentado con los 3.3 V que proporciona la placa PYNQ.

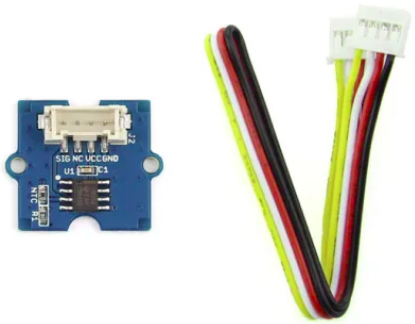


Figura 2.7: Sensor de temperatura Seeed Studio Grove-Temperature Sensor - 101020015. Fuente: rs-online

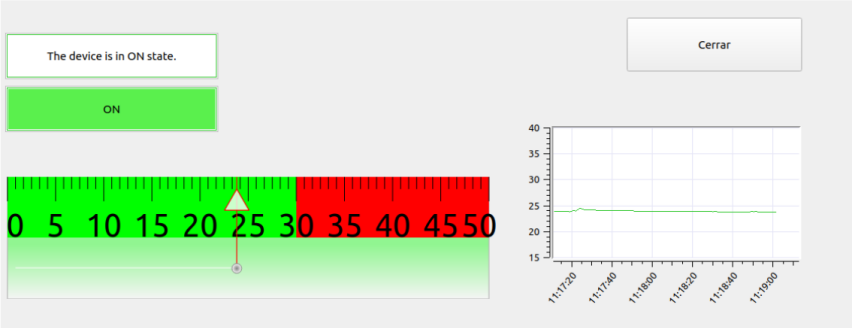
Los lenguajes de programación que se van a utilizar son C++ y Python. El lenguaje de programación que se ha utilizado para la prueba es Python ya que para este sensor existe gran variedad de librerías y es el lenguaje de programación más popular en estos últimos tiempos.

Tras la configuración de la pynq-z2 del apartado 2.2, se realiza el ejemplo de demostración en python, por lo que se ha de abrir Pogo para crear un *Device Server* siendo el atributo la temperatura

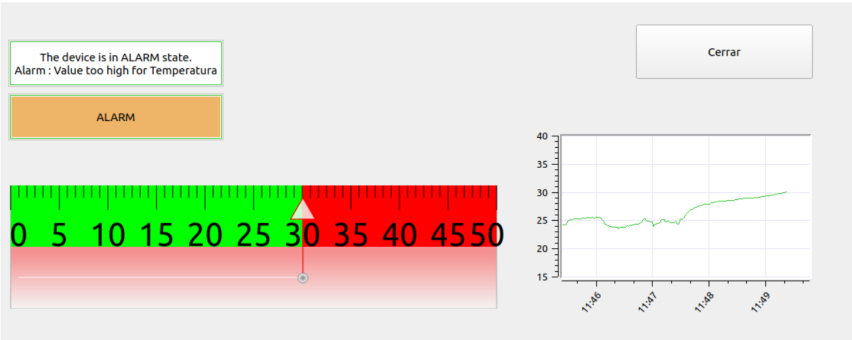
que es el valor que se deben obtener, además de añadir estados de ON, ALARM y de temperatura máxima y mínima. Se exporta y se genera la clase y se sube a jupyter notebooks donde se ha de añadir las líneas de comando necesarias para obtener la temperatura, en este caso al ser un sensor bastante común se ha encontrado el código en github. Además, se exporta el TANGO_HOST del ordenador en jupyter. Y finalmente se puede ejecutar la clase de Python.

Para ver el resultado, se crea un *Server* en jive para ver si se ha exportado correctamente el *Device Server*, y si está correctamente exportado pondra *true* y si no *false* como en la figura 2.6. En el *Monitor Device* se podrá ver la temperatura en el AtkPanel y ver las distinas propiedas, hacer modificaciones o incluso añadir una pequeña descripción.

Como se comentó anteriormente, jive proporciona una pequeña interfaz gráfica en la que se puede visualizar los atributos que se han configurado, pero para tener una interfaz más sofisticada se deberá de realizar por medio de Qt-Designer o Taurus. Si la temperatura aumenta hasta el límite superior, en este caso se ha puesto ese límite en 30 ºC, salta la alarma de que la temperatura es demasiado alta (figura 2.8b). En un simple ejemplo se ha podido configurar algunos parámetros, pero tango-controls ofrece muchos más parámetros, atributos, interrupciones, gráficas que serán útiles para el usuario, y además son modificables conforma avanza el tiempo o las necesidades del proyecto.



(a) Lectura de temperatura en la interfaz gráfica



(b) Alarma en la interfaz gráfica cuando supera los 30º

Figura 2.8: Interfaz gráfica realizada por Qt-Designer. Fuente: propia

Capítulo 3

Pruebas experimentales de la PCB

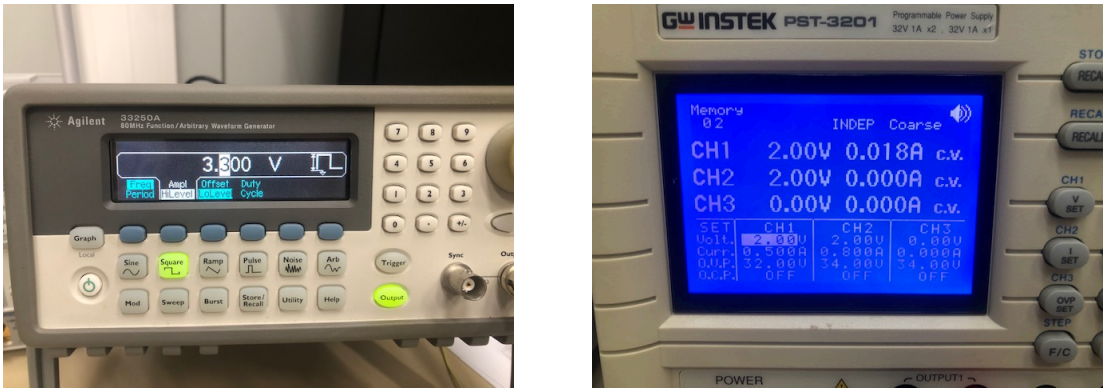
Para asegurar el buen funcionamiento de la PCB se han realizado pruebas aplicando alimentación y realizando medidas con el multímetro para comprobar que las tensiones que tiene en cada punto la PCB son correctas y no ha habido problemas ni errores en la soldadura. Las tensiones a evaluar son de 12, 5 y 3.3 V. También se hará uso del osciloscopio y generador de funciones inyectando una señal cuadrada con distintas frecuencias. Además se ha realizado un código de un simple PWM en VHDL para realizar un testeo más completo de la PCB y conocer tiempo de respuesta real de los componentes.

El motivo de utilizar un PWM para hacer las pruebas radica en que estos se utilizarán en el código final que se quiere utilizar, por lo que se puede comprobar si funciona bien todo el sistema montado y da la respuesta esperada. La verificación es tanto en hardware como en software, puesto que el código del PWM se está realizando en el entorno Vivado y se implementará un *Custom Core IP* creado por el usuario en un *Block Design*.

Debido a que se utiliza una FPGA de tipo Artix y un SoC Zynq 7020, el *Block Design* será diferente debido a que para la Artix es necesario un bloque microprocesador "software" llamado MicroBlaze diseñado para su implementación en FPGAs de Xilinx. Por otro lado, la placa de desarrollo pynq-z2 utiliza un bloque llamado *Zynq7 Proccessing System* que es para usar Zynq en su sistema de procesamiento (ARM). Cuando la integras al *Block Design* es como cualquier otro *Custom Core IP*. Significa que tienes un procesador ARM integrado en la FPGA y puedes aprovechar todas sus capacidades como PS DDR y USB externo. Esta parte se conoce como PS, y cuando la utilizas en el diseño de bloques en Vivado, todas las conexiones que ves van desde la PS al PL (donde PL significa FPGA).

3.1. Generador de funciones y fuente de alimentación

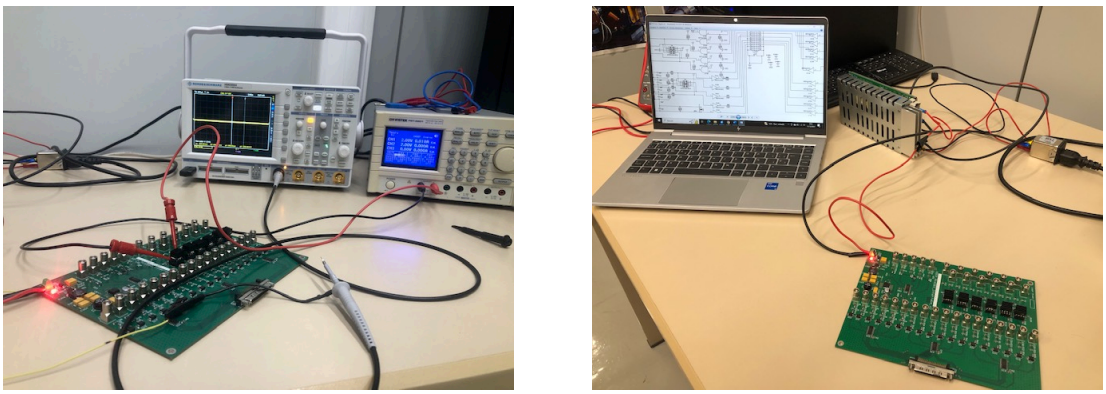
En esta primera prueba se utiliza el generador de funciones de una señal cuadrada unipolar con una frecuencia que irá variando y una amplitud pico a pico de 3.3 V sin componente negativa. Esta señal cuadrada representa la señal de disparo. También se ha utilizado una fuente de alimentación para señales continuas para representar las señales de interbloqueo.



(a) Generador de funciones **(b)** Fuente de alimentación

Figura 3.1: Instrumentación electrónica utilizada en el laboratorio para las pruebas. Fuente: propia

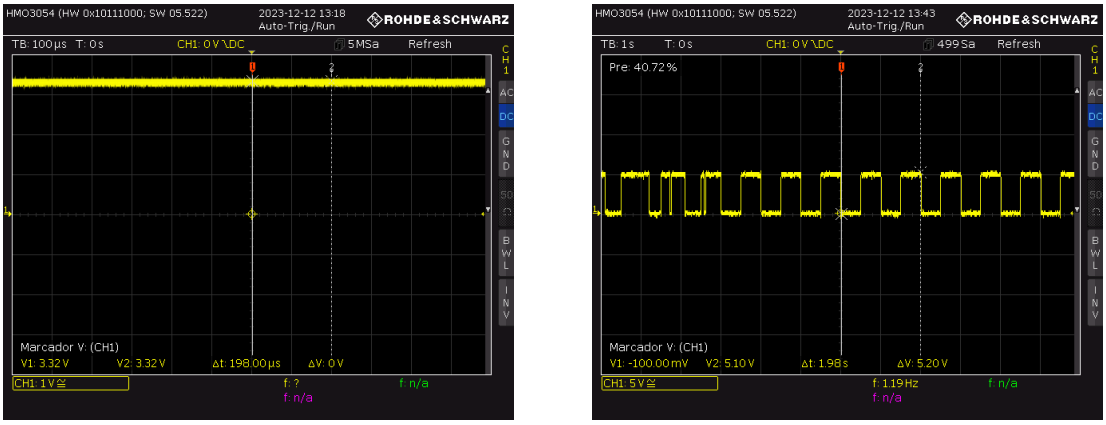
La instrumentación de la figura 3.1 genera la alimentación y la onda cuadrada configurada. A continuación, se puede visualizar la interconexión de la instrumentación a la PCB, la medida en osciloscopio y la alimentación de 12 V a la PCB.



(a) PCB e instrumentación electrónica **(b)** PCB y fuente de alimentación de la PCB

Figura 3.2: Medida en osciloscopio del sistema. Fuente: propia

Se comprueban que cada patilla de cada driver y punto de alimentación actúa como debe. En el caso del SN7545BP son salidas de la PCB que deberán de mandar señales de disparo, es decir, ondas cuadradas configurables a una tensión de salida de 5 V que han sido generadas con el generador de funciones, mientras que las entradas a la PCB están gestionadas por el driver HEF4050BT que deberá de recoger las señales de interbloqueo de los subsistemas de laboratorio. En esta prueba son señales continuas de 3.3 V que han sido generadas con la fuente de alimentación.



(a) Entradas de la señal continua del driver HEF4050BT

(b) Salidas de la señal cuadrada del driver SN7545B

Figura 3.3: Pruebas realizadas en los distintos drivers de la PCB. Fuente: propia

Además de estas pruebas, se ha verificado todos los puntos de alimentación de los convertidores DC/DC, aunque si no estuvieran bien soldados o no funcionaran correctamente la alimentación de los drivers no sería correcta y no habría señales en la Figura 3.3.

3.2. Pruebas en FPGA y SoC

Como se comentó en el apartado 1.3, el proyecto se realiza con la placa de desarrollo pynq-z2 que lleva incorporado un SoC, pero para las pruebas de la PCB se ha utilizado la AC701 también. El uso de ambas placas de desarrollo, pynq-z2 y AC701, ha permitido comparar y contrastar las capacidades de un SoC con procesador integrado frente a una FPGA con microprocesador implementado en su lógica programable. Esto ha proporcionado una perspectiva más amplia y un mayor entendimiento de las distintas arquitecturas y sus aplicaciones potenciales en el proyecto.

3.2.1. Codificación

El código que se ha utilizado es un PWM en código VHDL ya que es ligeramente parecido al funcionamiento final de las señales de disparo. La forma de crear el bloque AXI es igual en las dos placas, pero el diseño de bloques es ligeramente diferente como se comentó anteriormente. En FPGAs que incluyen un procesador hardcore como los Zynq-7020, se puede utilizar el procesador ARM integrado junto con bloques AXI, eliminando la necesidad de MicroBlaze.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.std_logic_unsigned.ALL;
4
5  entity pwm01 is
6      Port ( clk : in STD_LOGIC;
7          duty : in STD_LOGIC_VECTOR (7 downto 0);
8          pwm : out STD_LOGIC;
9          pwm_i : out STD_LOGIC
10     );
11 end pwm01;
```

```
12
13 architecture Behavioral01 of pwm01 is
14     signal counter : std_logic_vector (7 downto 0) := "00000000"; --initialize
           to 0
15     signal duty_sig : std_logic_vector (7 downto 0) := "00000000";
16     begin
17     process(clk)
18     begin
19         if rising_edge(clk) then
20             counter <= counter + 1;
21         end if;
22         if counter = 0 then
23             duty_sig <= duty;
24         end if;
25     end process;
26 pwm <= '1' when counter <= duty_sig else '0';
27 pwm_i <= '1' when counter > duty_sig else '0';
28 end Behavioral01;
```

Listing 3.1: Código VHDL del PWM

Además del PWM, se añade otra instanciación en el código top del bloque AXI de una entrada y salida digital para representar la salida del interbloqueo.

El código se ha realizado creando un *Custom AXI Core IP*, que aunque para esta prueba no sea necesaria el bus AXI, en el futuro sí que lo será debido a que las distintas variables serán cambiadas mapeando y cambiando las posiciones de memoria de cada registro. La realización del bloque IP se debe a que se pueda guardar en el repositorio y podría ser utilizado en cualquier otro proyecto sin necesidad de realizarlo de nuevo.

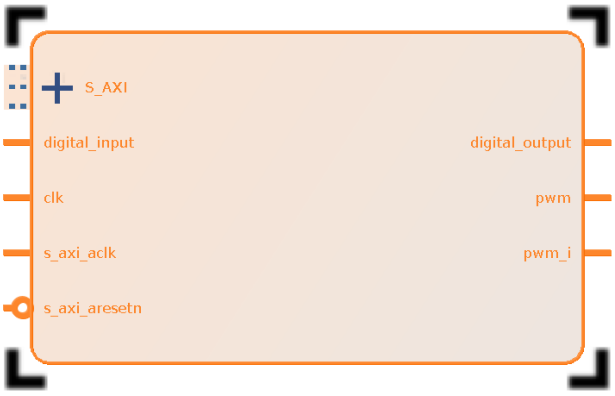
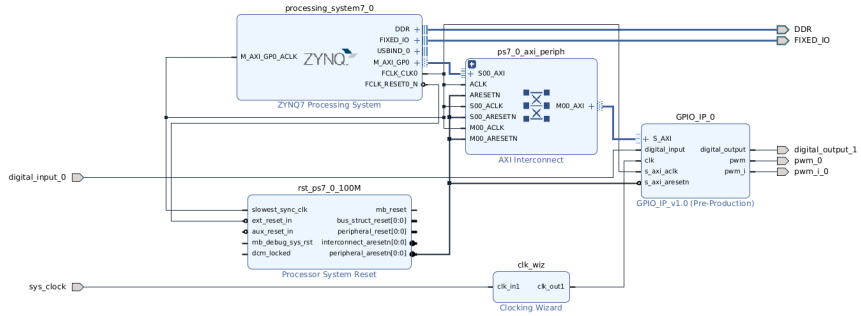
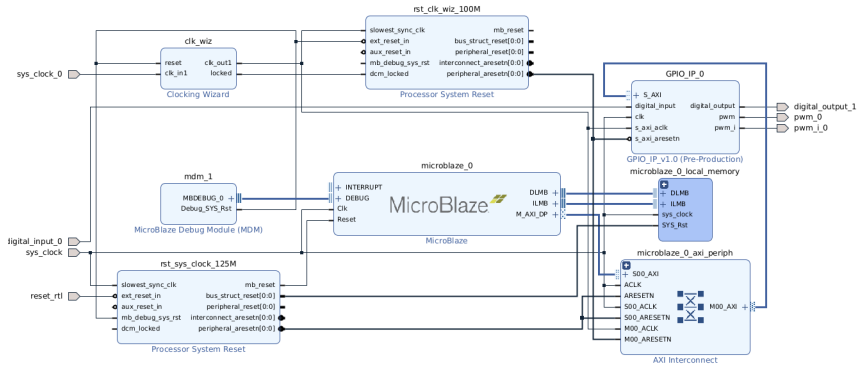


Figura 3.4: Bloque PWM en *IP Integrator*. Fuente: propia

Crear un *Custom Core IP* propio garantiza versatilidad de esa función que se ha programado el lenguaje de programación hardware VHDL (en este caso), ya que puedes utilizarlo en varios proyectos instanciándolo. Luego, es importante instanciar o realizar un correcto diseño de bloques para cumplir con las funcionalidades deseadas.



(a) Diseño de bloques de pynq-z2



(b) Diseño de bloques de AC701

Figura 3.5: Diseños de bloques en Vivado. Fuente: propia

En la parte superior de la figura 3.5 se puede visualizar el diseño de bloques para la pynq-z2, mientras que en la parte inferior se ha diseñado para la artix AC701. Como se ha ido comentando a lo largo de la memoria, es ligeramente diferente, pues ayuda a entender la versatilidad y diferencias de según que placa se esté utilizando.

Cabe indicar que realizar el diseño de bloques en Vivado es bastante intuitivo, ya que facilita las opciones de *Run Connection Automation* y *Block Connection Automation*. Estas funciones permiten realizar conexiones y añadir bloques de forma automática, utilizando configuraciones predeterminadas. En muchos casos, estas automatizaciones funcionan correctamente y de manera óptima, evitando la inclusión de bloques innecesarios que solo consumirían más recursos de hardware.

Sin embargo, cuando se trata de diseñar *Custom Core IP*, es necesario programar en un lenguaje de descripción de hardware (HDL). No es posible realizar estos diseños únicamente mediante bloques si la funcionalidad no es trivial. Vivado genera un archivo denominado HDL Wrapper en lenguajes como Verilog, VHDL o SystemVerilog, a partir del diseño de bloques. Este archivo permite instanciar bloques adicionales o modificar el código de manera más detallada, una tarea que puede ser complicada de realizar exclusivamente en el entorno de diseño de bloques. Este enfoque híbrido de diseño facilita tanto la creación rápida de prototipos como la posibilidad de realizar optimizaciones y personalizaciones avanzadas según sea necesario.

3.2.2. Simulaciones

Para realizar una comprobación de que el código funciona correctamente, en Vivado se pueden hacer simulaciones, comúnmente conocidas como *testbench*. Se debe definir las variables que se van a simular y se instancia el top de jerarquía. Se han verificado distintos escenarios con frecuencias diferentes, a continuación se han realizado dos casos.

En el caso 1, se configura con un *duty cycle* con valor numérico 254, siendo el rango máximo 255, es decir, 1 byte.

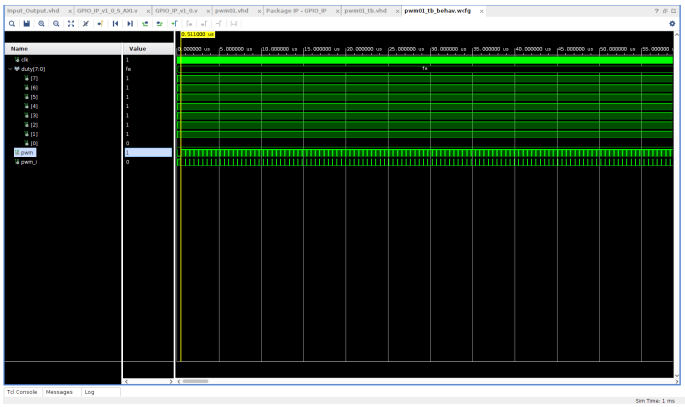


Figura 3.6: Bloque PWM en *IP Integrator*. Fuente: propia

Cambiando el *duty cycle* a 10 sobre 255, hace que el valor alto de la onda cuadrada sea menor y la parte baja se vuelve más grande. Al final es el funcionamiento de un PWM básico.

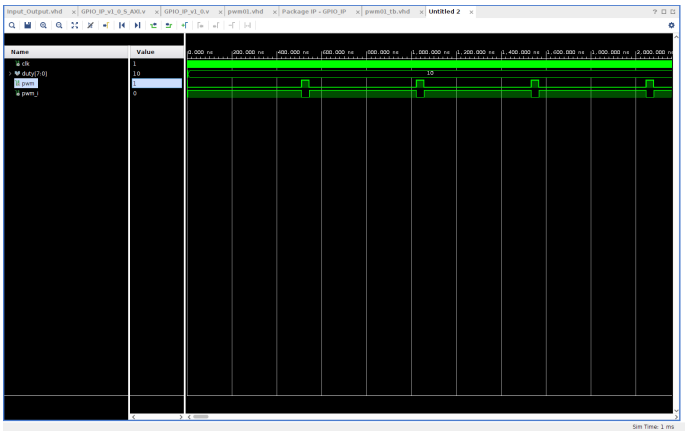
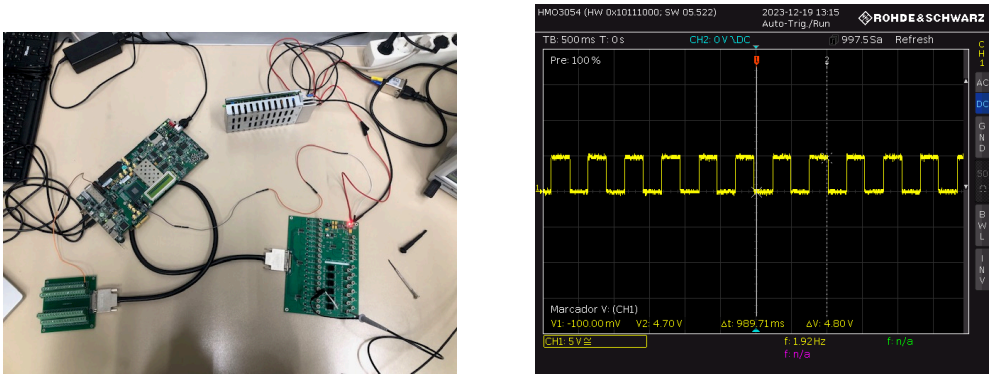


Figura 3.7: Bloque PWM en *IP Integrator*. Fuente: propia

3.2.3. Resultados

La verificación de la PCB se realizará como el apartado del generador de funciones y fuente de alimentación, pero la diferencia es que en este caso la señal de disparo es una onda PWM.



(a) Conexión PCB - AC701 **(b)** Visualización de la salida

Figura 3.8: Setup de instrumentación electrónica en el laboratorio para las pruebas. Fuente: propia

En este caso la frecuencia del PWM es de 1 Hz como se puede visualizar en la imagen 3.8b. A diferencia de las simulaciones se ha cambiado la frecuencia de forma drástica puesto que la PCB va a trabajar en frecuencias del orden de Hz, nunca llegaría a kHz, pues no está preparada para trabajar con frecuencias elevadas. Como se indicó antes se trabajará en el orden de 400 Hz, aunque puede variar dependiendo de la aplicación, pero el rango será de 100 a 800 Hz. Posteriormente, se realiza la conexión en la pynq-z2 de la misma forma que con la AC701.

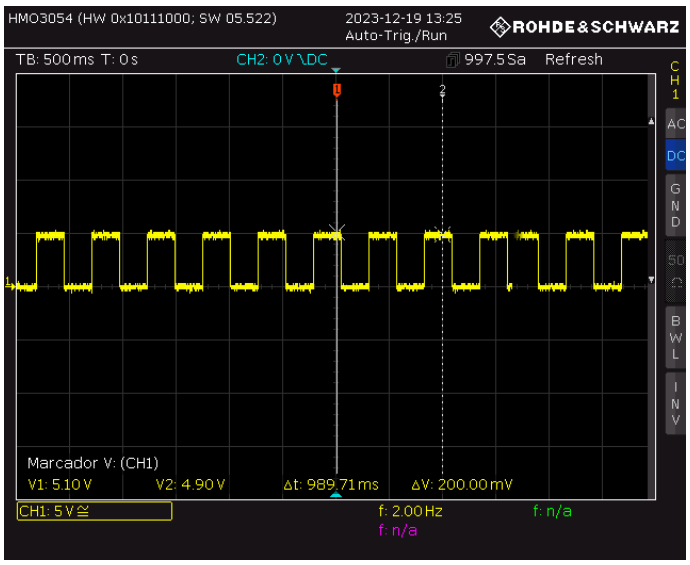


Figura 3.9: Visualización a la salida de la pynq-z2. Fuente: propia

3.3. Conclusiones de las pruebas

Se confirma que la PCB ha sido correctamente ensamblada y la creación del proyecto en Vivado ha sido suficiente para mostrar la señal a la salida en el osciloscopio. Se verifica que el funcionamiento del driver SN7545BP, HEF4050BT y que toda los convertidores DC/DC funcionan de

forma correcta. Se ha verificado todos los puntos y durante un tiempo de funcionamiento de varias horas para comprobar que cumple la fiabilidad estimada.

Finalmente, como se ha explicado con anterioridad, se deja de lado la FPGA AC701, y se decide continuar con la pynq-z2 a partir de aquí por las justificaciones que se han explicado en los primeros apartados.

Capítulo 4

Ensamblaje del sistema

La interfaz estará ensamblada en un chasis. El sistema se colocará en un armario de tipo *rack* cuyo tamaño dependerá del proyecto, capacidad de los demás componentes y funcionalidad de cada uno de ellos.

Un armario de tipo *rack* es un tipo de estante metálico cuya finalidad es almacenar equipamiento electrónico y de comunicaciones. Las medidas están normalizadas para conseguir compatibilidad de equipamiento con distintas marcas y fabricantes. En este tipo de armarios se colocan los chasis de aluminio normalizadas con entradas delanteras y traseras.

Figura 4.1: Armario rack. Fuente: propia

A parte de ofrecer facilidad en el montaje del armario, también ofrece versatilidad y comodidad a la hora del montaje, fabricación y diseño del chasis. La topología de este chasis se clasifican por xU, es decir, hay tamaños de 1U, 2U, 3U O 4U, a partir de 4U suelen ser directamente armarios ya que son dimensiones elevadas. Estos chasis cuentan con distintos componentes como guías o asas de transporte para facilitar la movilidad y poder hacer modificaciones del sistema de una forma rápida y fiable.

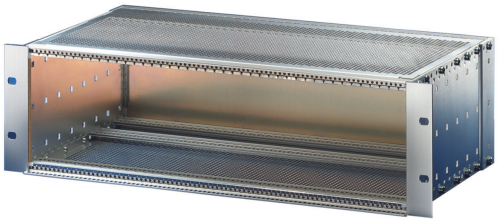
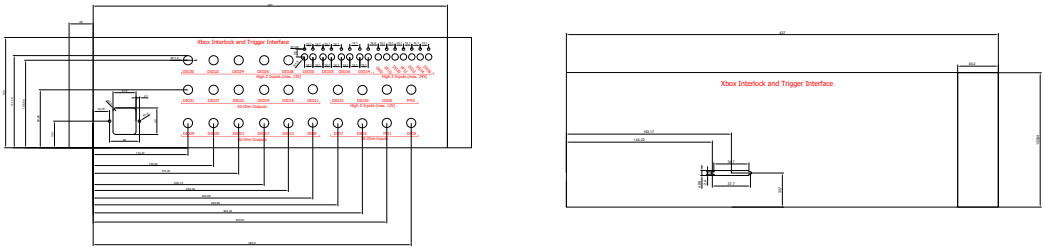


Figura 4.2: Chasis 3U. Fuente: propia

El chasis en el que se ha montado la PCB junto a los adaptadores ABD.00.250.NTM y ABB.00.250.NTM, conector de alimentación IEC C14 y conector VHDCI de 68 pines pasante, es un chasis 3U, suficientemente grande para que haya una distribución de cada componente y los cables no interfieran entre sí. Además, tiene unas rejillas que permite la circulación del flujo del aire, que aunque este chasis no tenga incorporado ningún sistema de refrigeración, es suficiente para que haya un mínimo de aire limpio y de menor temperatura en el sistema.

El chasis ha sufrido distintas modificaciones en la parte delantera y trasera. Las modificaciones traseras se realizan para el conector VHDCI de 68 pines y para una parte de la fuente de alimentacion 12V/4A. En cuanto a las modificaciones de la parte delantera son para los 25 adaptadores BNC, para 15 bornes pasamuros y el conector IEC C14. Las modificaciones se ha llevado a cabo a través del programa AutoCAD de Autodesk.



(a) Tapa delantera **(b)** Tapa trasera

Figura 4.3: Modificaciones AutoCAD del chasis. Fuente: propia

La fabricación del panel frontal y trasero ha sido realizado por el departamento mecánico del IFIC-UV de Valencia, ubicado en el Parque Científico. La elaboración ha sido realizada con una CNC (Control Numérico por Computadora), que permite la subida del archivo de AutoCAD y de esta forma realiza de forma precisa las operaciones que se han diseñado.

Tras las modificaciones de la tapa trasera y delantera se procede al montaje de la PCB dentro del chasis utilizando alzadores y y tornillos para fijar la PCB de la forma más estable posible. Además, se colocan los adaptadores de cable BNC para la conexión de los subsistemas de laboratorio. El cableado interno en el chasis del cable LEMO que proviene de la PCB a los adaptadores se colocan en canaletas. En cuanto a la fuente de 12 V se atornilla al propio chasis y se coloca en una esquina. Con estas modificaciones, se puede concluir que la interfaz I/O y la PCB están listas para su uso.

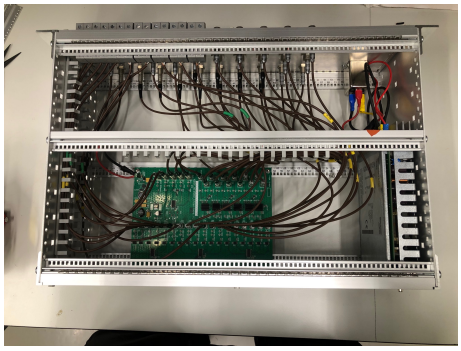
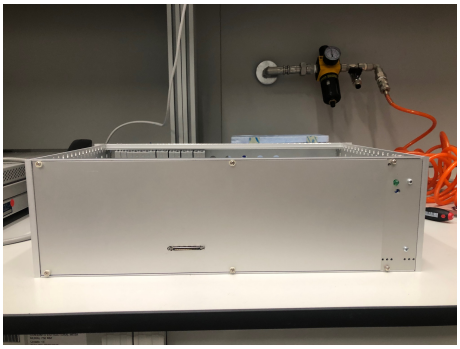


Figura 4.4: Chasis 3U. Fuente: propia

(a) Vista frontal



(b) Vista trasera

Figura 4.5: Vistas del chasis montado. Fuente: propia

Parte III

Desarrollo y resultados

Capítulo 1

Creación del proyecto

El principal objetivo del proyecto es la generación y sincronización de las señales de disparo e interbloqueo para la activación de un conjunto de periféricos gestionado por un sistema de control. La forma más adecuada para realizar un código fiable en entorno de programación digital es crear *Custom Core IP*.

Un *Custom Core IP* funciona como bloques de construcción que puedes integrar en implementaciones completas utilizando herramientas de diseño como Vivado™ IP Integrator de AMD. Se realizan a partir de código VHDL, Verilog o SystemVerilog. En este proyecto se ha ido alternando entre estos tres lenguajes de programación ya que Vivado permite que el lenguaje sea mixto, por lo que dependiendo del bloque o de la simulación se ha utilizado diferente lenguaje de programación dependiendo de las funciones utilizadas.

Se han realizado dos *Custom Core IP* para cada función y se incluye un tercero llamado registro que es para almacenar posibles señales de disparo.

Hay que indicar que en este apartado, los bloques no serán bloques AXI ya que solo se comprobará que el código hace lo que se requiere, y será más adelante donde se haga la integración con las direcciones de memoria AXI.

1.1. Descripción general de las señales

1.1.1. Señal de interbloqueo

Las señales de interbloqueo deben seguir unas reglas marcadas por una lógica de estados y se activarán, para interbloquear uno u otro subsistema, respondiendo a combinaciones de señales de entrada. La señal de salida dependerá de 5 señales de entrada.

Entradas	Salida
Input_Software (I) Enable (E) Nivel (N) Prioridad (P) Activación (A)	Interlock_out (OUT)

Tabla 1.1: Tabla general de entradas y salidas del sistema. Fuente: propia

La forma de expresar el comportamiento de la señal de salida en función de todas las posibles combinaciones de entradas, en una linea de código, es utilizar una tabla de verdad. De la siguiente tabla de verdad no se va a considerar la entrada Activación ya que es preferente y cuando es 1 la salida es 1 siempre. Se codificará como una puerta OR en toda la ecuación resultante.

I	E	N	P	OUT
0	0	0	0	0
			1	0
		1	0	0
			1	0
	1	0	0	0
			1	1
		1	0	0
			1	0
1	0	0	0	0
			1	0
		1	0	0
			1	0
	1	0	0	0
			1	0
		1	0	0
			1	1

Tabla 1.2: Tabla de entradas y salida del sistema. Fuente: propia

Para considerar todos los posibles casos, se analiza cada entrada respecto a la salida y se calcula la ecuación en una sola línea. Esta ecuación debe derivarse a partir de puertas lógicas. Es necesario visualizar cuándo la salida es 1. En este caso, hay 4 posibilidades como se muestra en la tabla 1.2:

$$S = (\overline{I} \wedge E \wedge \overline{N} \wedge P) \vee (I \wedge E \wedge N \wedge P) \tag{1.1}$$

Ahora se añade la entrada Activación que no se ha considerado en la tabla 1.2, añadiendo una puerta OR a toda la ecuación.

$$S = (E \wedge P \wedge [(\overline{I} \wedge \overline{N}) \vee (I \wedge N)]) \vee A \tag{1.2}$$

Se puede simplificar bastante la operacion que hay entre corchetes a una puerta XOR negada.

$$S = (E \wedge P \wedge \overline{[I \oplus N]}) \vee A \tag{1.3}$$

En cuanto a la salida tendrá un tamaño de dos bits debido a que el Input (I) recibido consta de 32 bits de los cuales la mitad son por hardware y la otra mitad por software. Los recibidos por hardware son los que provienen de los subsistemas de laboratorio, mientras que los de software pueden ser configurados o lanzados por el usuario. Sin embargo ese tamaño de dos bits se realizará en el apartado siguiente 2, ya que en este apartado solamente se realizó para un bit (software).

Finalmente, la ecuación se codifica en Verilog en la configuración del *Custom Core IP*. La implementación se debe realizar en varios ficheros indicando las salidas y entradas, si son registradas o no... y finalmente empaquetar ese bloque indicando las direcciones de memoria que utilizará e indicar que parámetros pueden ser configurables desde el diseño de bloques, todo esto se realiza en el fichero component.xml. Además se generará la interfaz gráfica del bloque que luego se instancia en el diseño de bloques.

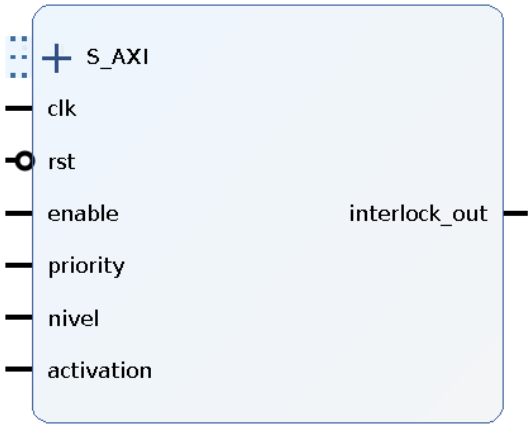


Figura 1.1: Interfaz gráfica de usuario del core Interlock. Fuente: propia

1.1.2. Señal de disparo

Además del bloque para la señal de interbloqueo se programa otro para la señal de disparo. Las señales de disparo obedecen a una lógica que se activa en consonancia con la configuración previa del usuario. La sincronización precisa de este módulo es crítica, dado que las señales de disparo activarán secuencialmente los componentes de los subsistemas de laboratorio donde las señales de activación tendrán unos tiempos de envío independiente.

El bloque funciona manteniendo un nivel bajo un tiempo determinado (offset) y activarse durante otro tiempo X. Ese disparo puede ser único, periódico o un número fijo de triggers. La frecuencia a la que trabaje este bloque es independiente a los demás bloques, pudiendo así cambiar la frecuencia de las señales de los disparos.

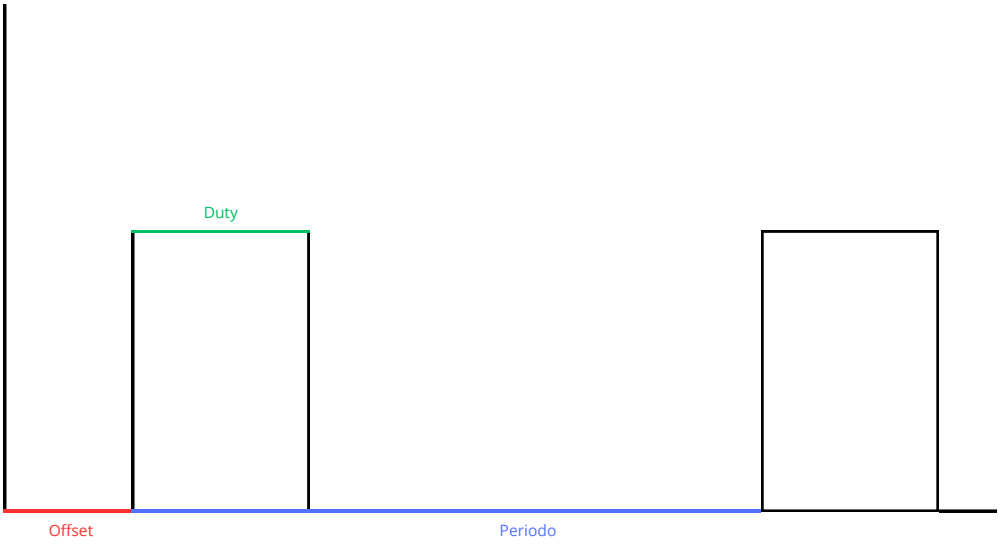


Figura 1.2: Funcionamiento básico de la señal de disparo. Fuente: propia

- **Control**
  - Modo 0 Apagado
  - Modo 1 Encendido modo continuo
  - Modo 3 Envío fijo de triggers
- **Offset**
  - Especificado en cuentas (1/clk_ref)
- **Duty**
  - Especificado en cuentas (1/clk_ref)
- **Periodo**
  - Especificado en cuentas (1/clk_ref)
  - Default: (FREQ -1)
- **Número fijo de triggers para enviar**
  - Control modo 3
  - Default: 0
- **Cont. Triggers enviados**
  - Modo 3
- **Valor ID identificativo del core**
  - Default: 0xAABB2022

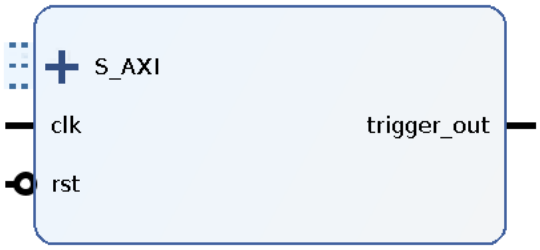


Figura 1.3: Interfaz gráfica de usuario del core Trigger. Fuente: propia

1.1.3. Registro

Se ha realizado el diseño de un tercer y último bloque. Las señales provenientes de los subsistemas de laboratorio podrían ser o no conocidas por lo que debido al desconocimiento de características de las señales como su tiempo de pulso o tiempo de respuesta se debe evitar la pérdida de un posible pulso almacenándolo en un registro. Por ello, se ha diseñado un bloque para almacenar señales para evitar que se pierdan algunas debido a las características específicas de esa señal.

Esta función no es necesaria para la aplicación en sí, pero se incluye para realizar prueba sobre ello y comprobar su validez en la casuística comentada, es decir, se añade a modo de prevención.

El funcionamiento es bastante simple, se trata de un código que registra una entrada, la cual estará conectada a la señal de salida de los subsistemas de laboratorio. Esta operación dependerá de un nivel lógico declarado como parámetro. En esencia, se implementa una puerta lógica XNOR, donde la salida será 1 únicamente cuando la señal proveniente de los subsistemas y el parámetro sean iguales. Además, se añade una señal de salida Qn, que es la negación de la salida Q. Esto se hace para manejar el caso donde la señal proporcionada por el componente x de los subsistemas de laboratorio tiene la polaridad invertida respecto al periodo definido: es decir, cuando un nivel bajo en el periodo se interpreta como alto y viceversa. Esta señal de salida Qn se conectará a la entrada del núcleo del interbloqueo.

Parámetro (fijo)	Signal Register	Q	Qn
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 1.3: Tabla de entradas y salidas del registro. Fuente: propia

Se desarrolla la ecuación para Q, es bastante simple debido a que es una puerta lógica XNOR.

$$Q = (\overline{Parametro \oplus SignalRegister}) \tag{1.4}$$

$$Qn = \overline{Q} \tag{1.5}$$

Finalmente, como en los anteriores bloques a partir del archivo component.xml se especifican parámetros, entradas, salidas, configuraciones de memoria y se crea la interfaz gráfica del bloque diseñado.

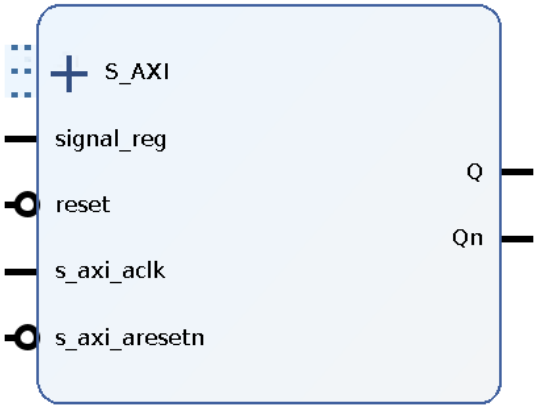


Figura 1.4: Interfaz gráfica de usuario del core registro. Fuente: propia

1.2. Verificación de bloques

La verificación de cada bloque se ha simulado en Vivado a partir de *testbenchs* indicando características diferentes a las entradas y tiempos. También se ha probado en placa cada uno de los bloques por separado en sencillos diseño de bloques para comprobar el funcionamiento sobre la pynq-z2. El bloque de registro no se verifica puesto que este bloque registra las señales de los subsistemas de laboratorio y para estas pruebas solamente se necesita verificar la funcionalidad de las señales de disparo e interbloqueo.

1.2.1. Señal de interbloqueo

Para un primer caso donde las señales de entrada input_interblock, enable, prioridad, nivel y activación son 11011, respectivamente, la señal de salida será 1 (interblock_out) debido a que la señal activation es 1 y como se ha comentado anteriormente, es preferente ante todas las demás.

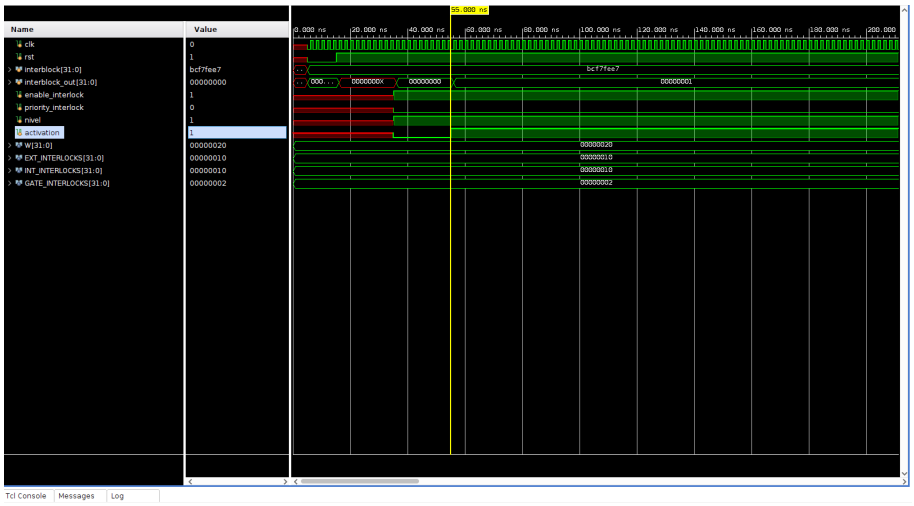


Figura 1.5: Primer caso para la verificación de la señal de interbloqueo. Fuente: propia

En este segundo caso, la salida cambia de 0 a 1 debido a una transición entre 50 y 60 ns. Activation es 0 y las demás señales (input_interblock, enable, priority, nivel) son 1111, resultando en una salida de 1 también.

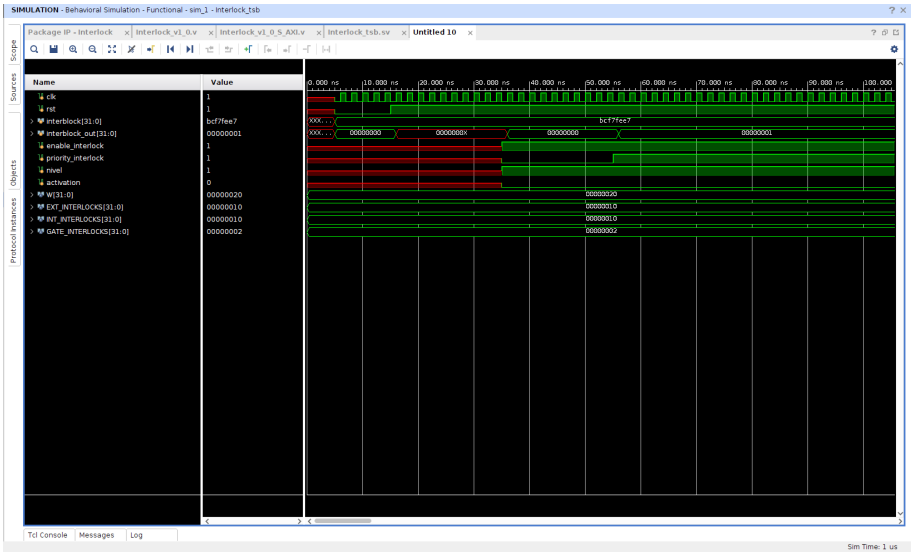


Figura 1.6: Segundo caso para la verificación de la señal de interbloqueo. Fuente: propia

Y un último caso en el que se ha parametrizado las variables a 32 bits para ver que funciona en muchos casos diferentes.

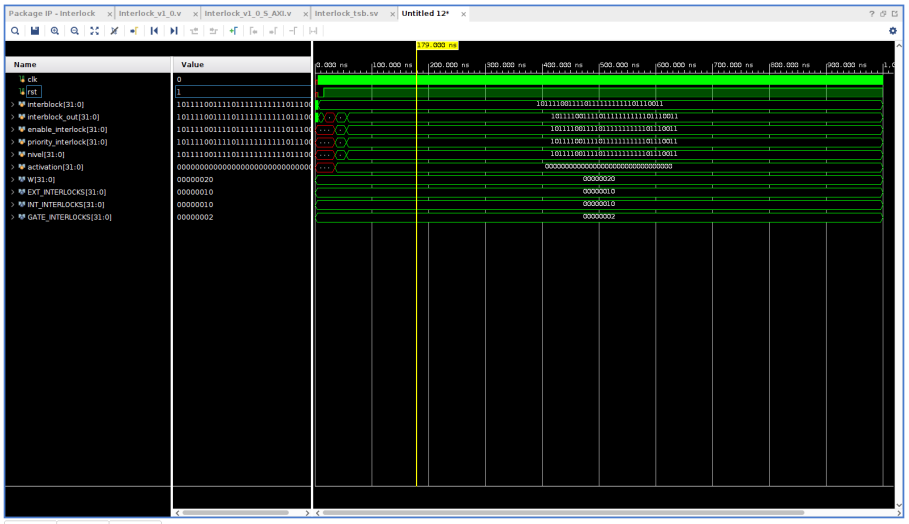


Figura 1.7: Tercer caso para la verificación de la señal de interbloqueo. Fuente: propia

Se puede ir verificando que son correctas las señales de salida de los distintos casos, por lo que la lógica de estados que se ha programado está bien diseñada.

1.2.2. Señal de disparo

En el primer caso que se verifica, se envían dos señales de disparo diferentes. Para la primera señal de disparo el valor bajo Li es de 64 y Hi es de 46, mientras que la segunda señal el valor bajo es de 96 y la alta es de 90 en decimal.

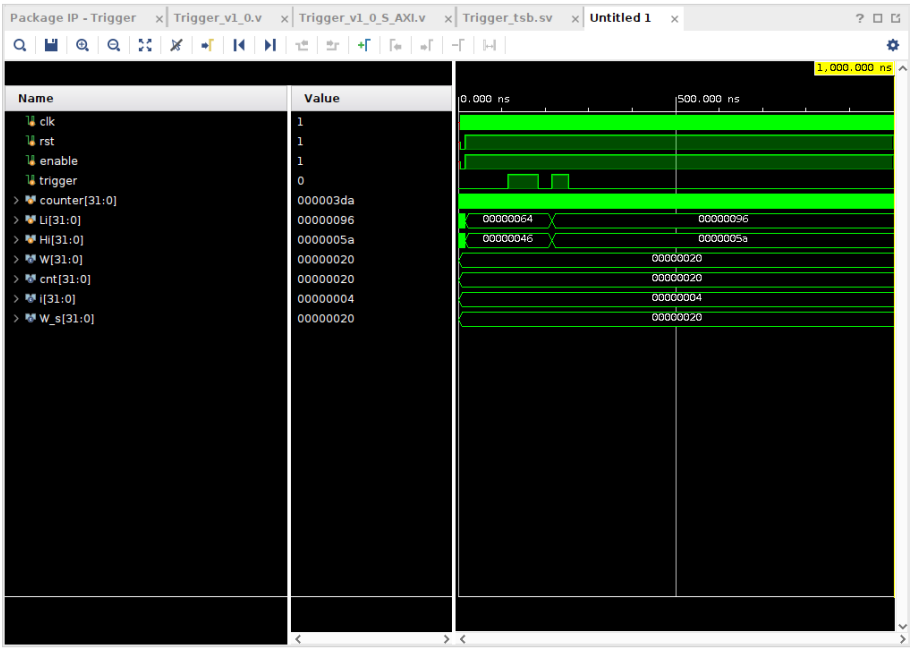


Figura 1.8: Interfaz gráfica de usuario del core registro. Fuente: propia

Y en un segundo caso, se envía un tren de pulsos periódicos con un nivel alto de 120 ns y un nivel bajo de 100 ns.

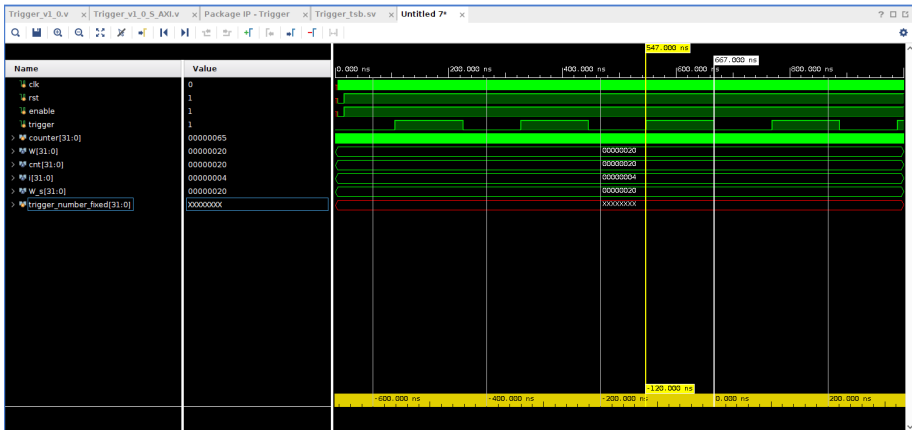


Figura 1.9: Interfaz gráfica de usuario del core registro. Fuente: propia

En definitiva, tanto la señal de disparo como de interbloqueo funciona como se esperaba por lo que el siguiente paso es la unión de todos los bloques en el *Block Design* de Vivado.

1.3. Diseño e integración de bloques IP

Una vez simulados los bloques por separado y verificados, se decide continuar a la interconexión de todos los bloques en el *IP INTEGRATOR* de Vivado en un único diagrama.

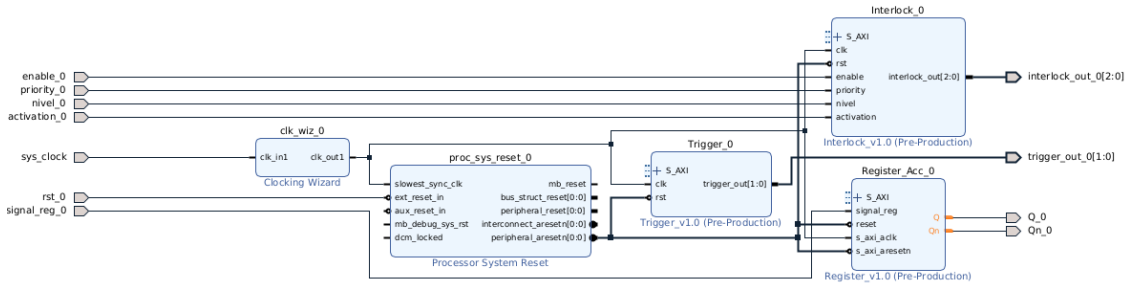


Figura 1.10: *Block design*. Fuente: propia

Se debe de incluir un reloj del sistema llamado *Clocking wizard* con una frecuencia de 100 MHz debido a que para estas pruebas no se va a utilizar el bloque AXI se utiliza un reloj definido por este bloque auxiliar y no es necesario el reloj AXI que lo genera *Zynq7 Processing System*. Además, se ha añadido un módulo de reinicio del sistema para utilizar el reset asíncrono que se le ha añadido en cada uno de los bloques, este bloque se llama *Processor System Reset* y se puede configurar para que funcione a nivel bajo o alto.

Para probar en placa y realizar la conexión en el sistema, se deben añadir las *constraints* de cada pin que se utiliza, en este caso se van a utilizar los dos PMOD de la pynq-z2 ya que nos proporciona 8 pines cada uno de ellos y es útil para visualizar las señales por el osciloscopio. Además, los LEDs podrán ser utilizados para realizar *debug* y conocer si alguna señal está fallando, pues esta placa contiene 4 botones y 2 interruptores adicionales. En Vivado se ha de añadir el XDC (*Xilinx Design Constraints file*) para indicar al software qué pines físicos se ha de rutar para el código HDL que se ha creado y se comporte como debe.

```
# Reset common Interruptor
# SW0
set_property PACKAGE_PIN M20 [get_ports rst_0]

# PMOD
# PINES GND ES PENULTIMA COLUMNA Y PINES VCC ES ULTIMA COLUMNA (DOS FILAS EN AMBAS).
# PIN 1 SUPERIOR PMOD A QUE ES DERECHA
set_property PACKAGE_PIN Y18 [get_ports trigger_out_0[0]]
set_property PACKAGE_PIN U18 [get_ports trigger_out_0[1]]
# PIN 2 SUPERIOR PMOD A QUE ES DERECHA
set_property PACKAGE_PIN Y19 [get_ports {interlock_out_0[0]}]
set_property PACKAGE_PIN U19 [get_ports interlock_out_0[1]]
set_property PACKAGE_PIN W18 [get_ports interlock_out_0[2]]
# PIN 3 SUPERIOR PMOD A QUE ES DERECHA
set_property PACKAGE_PIN Y16 [get_ports Q_0]
# PIN 4 SUPERIOR PMOD A QUE ES DERECHA
set_property PACKAGE_PIN Y17 [get_ports Qn_0]

# Configuración Voltaje 3.3
set_property IOSTANDARD LVCMOS33 [get_ports interlock_out_0[0]]
set_property IOSTANDARD LVCMOS33 [get_ports interlock_out_0[1]]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports interlock_out_0[2]]
set_property IOSTANDARD LVCMOS33 [get_ports activation_0]
set_property IOSTANDARD LVCMOS33 [get_ports nivel_0]
set_property IOSTANDARD LVCMOS33 [get_ports enable_0]
set_property IOSTANDARD LVCMOS33 [get_ports priority_0]
set_property IOSTANDARD LVCMOS33 [get_ports rst_0]
set_property IOSTANDARD LVCMOS33 [get_ports Q_0]
set_property IOSTANDARD LVCMOS33 [get_ports Qn_0]
set_property IOSTANDARD LVCMOS33 [get_ports trigger_out_0[0]]
set_property IOSTANDARD LVCMOS33 [get_ports trigger_out_0[1]]
set_property IOSTANDARD LVCMOS33 [get_ports trigger_out_0]

#Configuracion Pull Up, Corriente a 16 mA y velocidad
set_property PULLUP true [get_ports Q_0]
set_property PULLUP true [get_ports Qn_0]
set_property DRIVE 16 [get_ports trigger_out_0]
set_property SLEW FAST [get_ports trigger_out_0]
```

Listing 1.1: Fichero de Constraints

Hay que indicar que en las pruebas, solo se utilizan pines individuales, pero en la implementación final harán falta varias instanciaciones de cada bloque, por lo que aumentarán estos recursos dependiendo de cuántas señales de disparo, interbloqueo y registros se necesiten.

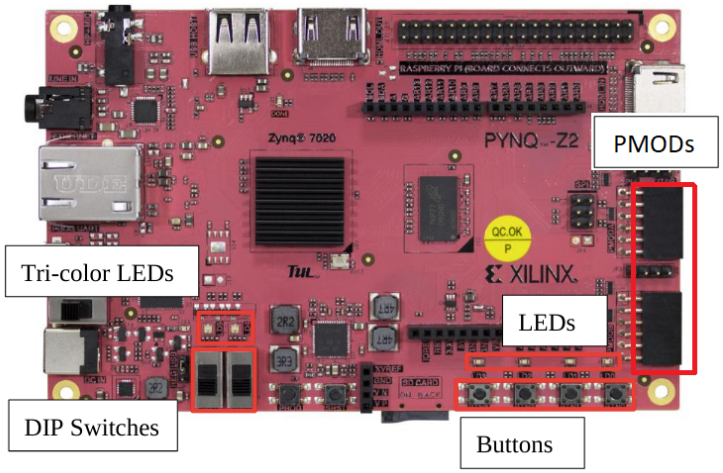


Figura 1.11: Leds, botones, interruptores y PMODs. Fuente: [19]

1.4. Verificación del diseño realizado

Se procede al montaje del sistema completo incluyendo instrumentación electrónica como el osciloscopio para la visualización de las entradas y salidas. El sistema se compone del PC que irá conectado a la pynq-z2 proporcionándole alimentación y programando el dispositivo. Desde los puertos PMOD irán conectados cablecillos hacia los distintos puntos del sistema para mandar las señales de disparo e interbloqueo, y además la señal de GND se conectará con GND de la PCB y de la instrumentación para que todo esté referenciado a la misma GND.

La Interfaz I/O estará conectada por la parte trasera al expansor GPIO de 68 pines (placa verde) para conectar osciloscopio o cablecillos desde la pynq-z2. Y por la parte delantera de la interfaz I/O estarán los conectores para ver las señales de salida por el osciloscopio o insertar las señales desde la pynq-z2.

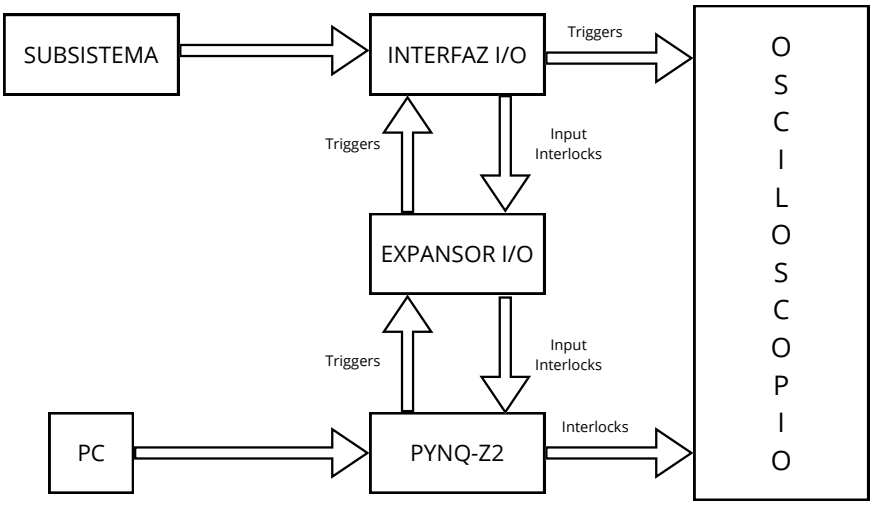


Figura 1.12: Esquema de la conexión realizada. Fuente: propia

Las señales de entrada a la interfaz I/O son las señales de interbloqueos y las señales de salida son las señales de disparo. Las primeras se van a insertar desde la interfaz I/O hacia el expansor de 68 pines y se visualizarán en el osciloscopio, mientras que las señales de disparo se insertan en el expansor de 68 pines y se visualizarán en los conectores BNC de la interfaz I/O.

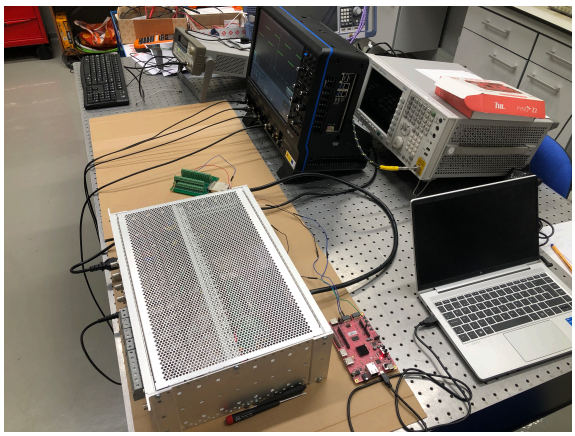


Figura 1.13: Sistema montado. Fuente: propia

Se recuerda que las salidas están gestionadas por el driver SN7545BP, mientras que las entradas estarán gestionadas por el HEF4050BT, esto significa que el nivel alto de las salidas será de 5 V y las entradas de 3.3 V. Una consideración importante a recordar es que la corriente mínima que es necesaria por los circuitos internos de la PCB es de 33 mA.

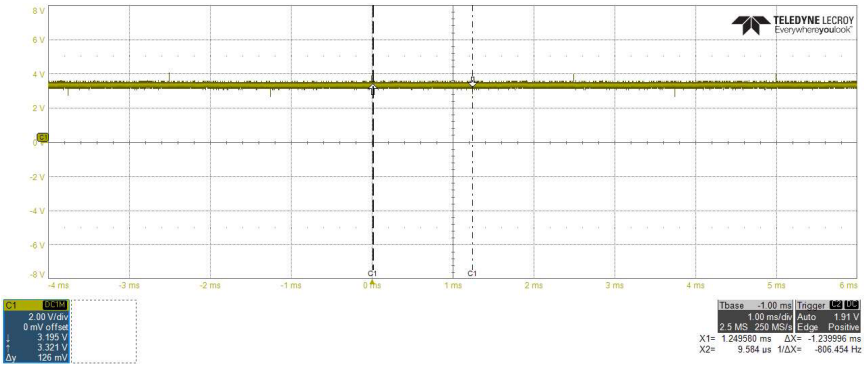


Figura 1.14: Visualización en el osciloscopio de la entrada a la interfaz I/O (DIO18). Fuente: propia

En cuanto a la señal de disparo, se deberá evaluar tiempos de subida y bajada de la onda de disparo y que no haya demasiadas perturbaciones. Estas pruebas se están realizando con cablecillos que no son los apropiados para evitar perturbaciones de ruido. La frecuencia de la onda es de 400 Hz (2.5 ms) con una tensión de salida de 5 V en la parte alta. La configuración de la onda se ha realizado para que sea periódica.

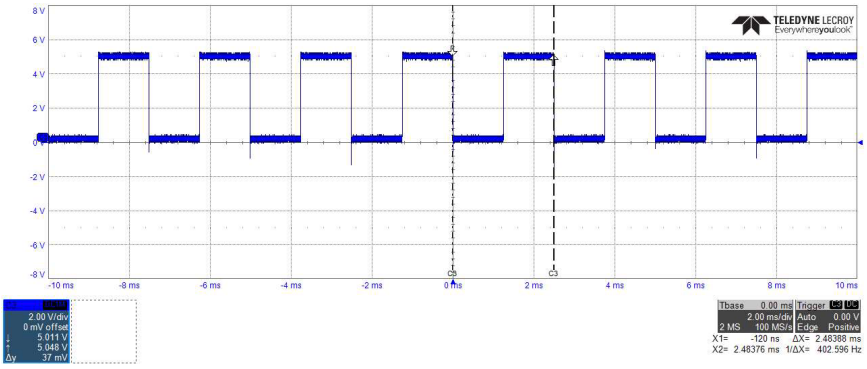
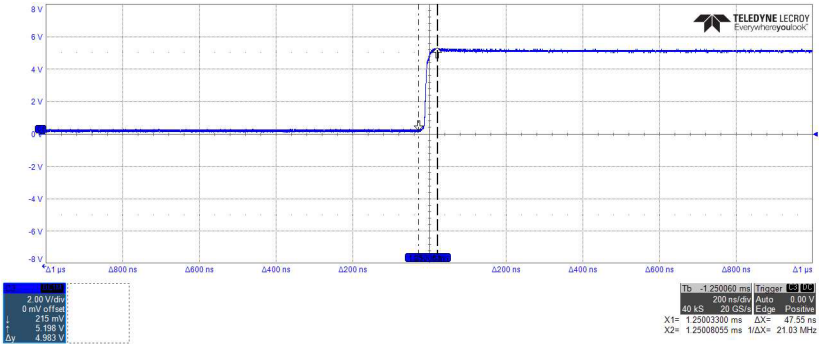
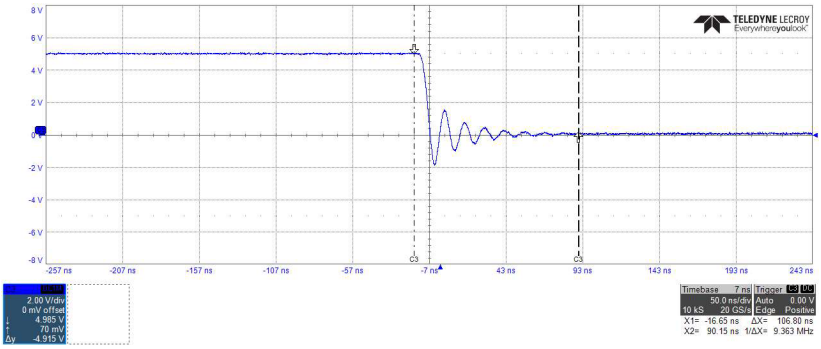


Figura 1.15: Visualización en el osciloscopio de salida de la interfaz I/O (DIO21). Fuente: propia

En el datasheet del fabricante del driver SN7545BP indica que el tiempo de subida es menor a 5 ns y el de bajada menor o igual a 10 ns. En nuestro caso, el tiempo de subida es de 47.55 ns. En cuanto al tiempo de bajada hay una sobremodulación hasta llegar al nivel bajo, es algo mayor al tiempo de subida ya que son 106.80 ns. Sin embargo son valores insignificantes para un periodo de 2.5 ms (frecuencia 400 Hz), es más de un factor 20000 respecto a las frecuencias que se van a trabajar.



(a) Tiempo de subida



(b) Tiempo de bajada

Figura 1.16: Visualización en el osciloscopio de la salida a la interfaz I/O (DIO21). Fuente: propia

Se han realizado las pruebas de forma conjunta. El objetivo real de este proyecto es que haya señales de interbloqueo y de disparo que funcionen de forma sincronizada entre unas y otras con tiempos predefinidos de los subsistemas de laboratorio y configurados por el operario. Al desconocer la magnitud de estos tiempos, se han configurado ondas periódicas de 400 Hz, que al menos sí se sabe que es la frecuencia a la que trabajarán.

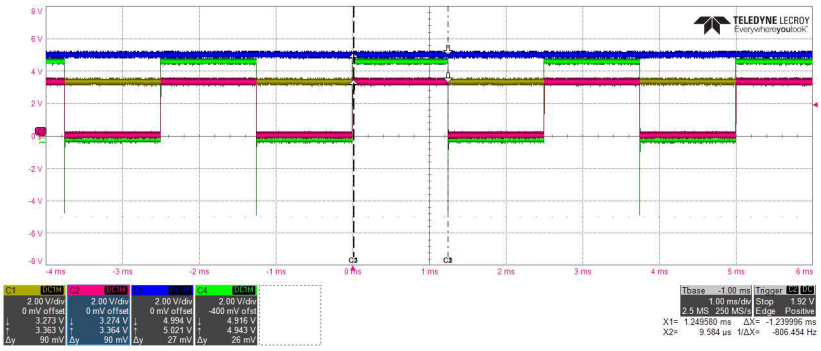


Figura 1.17: Visualización en el osciloscopio de varias entradas y salidas de la interfaz I/O. Fuente: propia

En la figura 1.17 se pueden visualizar dos salidas y dos entradas a diferente voltaje, esto se debe a que es posible utilizar las entradas de la interfaz como señales de disparo y las salidas como

señales de interbloqueo.

Capítulo 2

Integración bus AXI y Sistema Operativo

Este capítulo aborda la integración del bus AXI en el diseño de Vivado y el sistema operativo embebido en la FPGA. El propósito es proporcionar una guía detallada para combinar el hardware configurado en Vivado con un sistema operativo adecuado, como Ubuntu 22.04, que permitirá acceder a las posiciones de memoria de las variables desde un sistema SCADA con interfaz gráfica. Además, se explican las alternativas de implementación, como el uso de Petalinux o la imagen PYNQ, y se detalla el proceso de configuración y verificación del sistema embebido, así como la demostración práctica de su funcionamiento.

2.1. Integración AXI

La verificación de los *Custom Core IP* se ha realizado de forma positiva. El siguiente paso es realizar pequeñas modificaciones para que sean bloques AXI y de esta forma acceder a las posiciones de memoria desde la interfaz gráfica que se ha de crear en tango-controls y desde la pynq-z2.

En el caso de la señal de disparo se han escogido las siguientes posiciones de memoria para cada parámetro. Cada registro de configuración ocupa un rango de 4 bits. Al *Custom Core IP* se le asigna por defecto una posición en memoria base, por ejemplo 0x43C20000, y una posición en memoria final, por ejemplo 0x43C2FFFF. Hay 8 registros y 2 bits por registro serán 16 bits, es decir, 2 bytes en total.

- **slv_reg0 (reg 0x00)**
  - Control STATUS/FLAG
  - Modo 0 Apagado
  - Modo 1 Encendido modo continuo
  - Modo 3 Envío fijo de triggers
- **slv_reg1 (reg 0x04)**
  - Offset, especificado en cuentas (1/clk_ref)

- **slv_reg2 (reg 0x08)**
  - Dutty cycle, especificado en cuentas (1/clk_ref)
- **slv_reg3 (reg 0x0C)**
  - Periodo, especificado en cuentas (1/clk_ref)
  - Default: (FREQ -1)
- **slv_reg4 (reg 0x10)**
- **slv_reg5 (reg 0x14)**
  - Número fijo de triggers para enviar (modo 3)
  - Default: 0
- **slv_reg6 (reg 0x18)**
  - Cont. Triggers enviados (modo 3)
- **slv_reg7 (reg 0x1C)**
  - Valor ID identificativo del core
  - Default: 0xAABB2022

En el caso de la señal de interbloqueo:

- **slv_reg1 (reg 0x04)**
  - Enable
- **slv_reg2 (reg 0x08)**
  - Nivel
- **slv_reg3 (reg 0x0C)**
  - Prioridad
- **slv_reg4 (reg 0x10)**
  - Activación
- **slv_reg5 (reg 0x14)**
  - Input software [31:15]

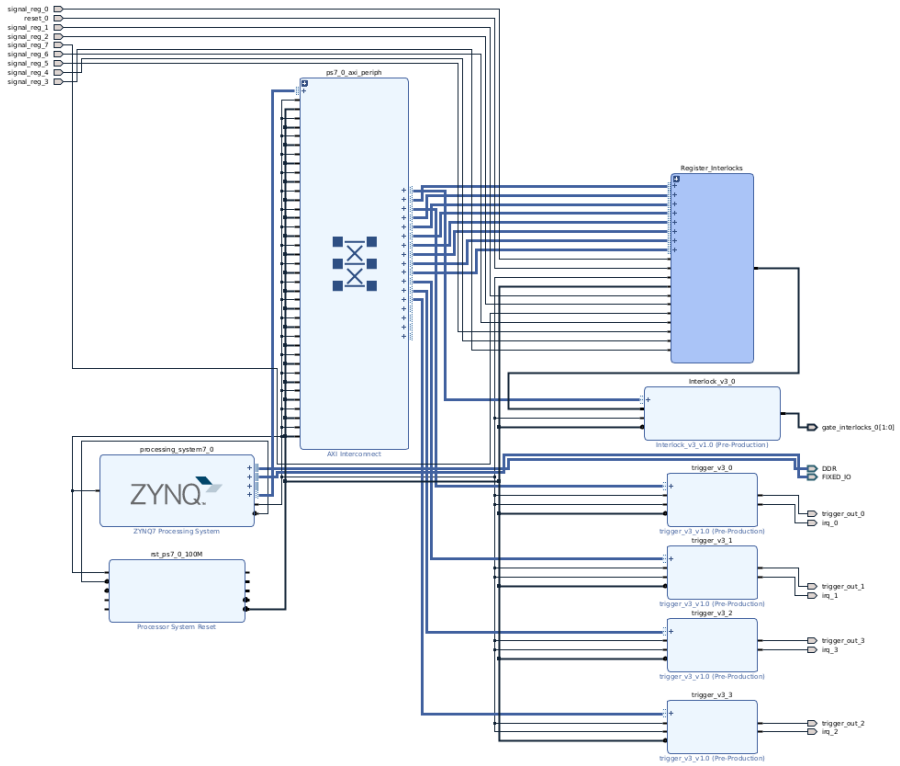


Figura 2.1: Diseño de bloques en Vivado. Fuente: propia

Se instanciarán en el diseño de bloques más señales de disparo y registros para tener varias salidas. En este caso como se puede visualizar no están las entradas ni salidas de los bloques como lo estaban en el apartado de 1.3 ya que son registros internos de cada bloque.

A diferencia del apartado 1.3, se han incluido bloques predefinidos de Xilinx, como lo son *ZYNQ7 Processing System* que sirve para gestionar la frecuencia de reloj del bus AXI que se requiere y otras características:

- Habilitar/Deshabilitar los subsistemas de laboratorio de E/S (IOP)
- Habilitar/Deshabilitar puertos AXI de E/S (AIO)
- Configuración de MIO
- Extensión de E/S de USO MÚLTIPLE (EMIO)
- Configuración de DDR
- Configuración de Seguridad y Aislamiento
- Lógica de Interconexión para la interfaz IP de Vivado - PS
- Relojes e Interrupciones del PL

Y en cuanto al AXI Interconnect IP permite la conexión entre los múltiples subsistemas de laboratorio maestros y esclavos con mapeo de memoria AXI, siguiendo las especificaciones. Este IP está diseñado exclusivamente para transferencias con mapeo de memoria.

2.2. Sistema Operativo Embebido

Para implementar el sistema de control Tango Controls en la placa Pynq-Z2, existen varias alternativas. La decisión crucial es si integrar un sistema operativo como Petalinux para aprovechar tango-controls en ese entorno, o si utilizar el firmware PYNQ que incluye Ubuntu 22.04 que por defecto es suficiente. Una complicación es que PYNQ trabaja sobre un bitstream predefinido que no incluye los Custom Core IP necesarios en este proyecto. Esto plantea diversas opciones para incorporar estos componentes personalizados dentro del bitstream existente.

Como se ha explicado en capítulos anteriores, se carga el fichero bitstream (.bit) que genera Vivado y se configura la placa para que arranque desde la SD. De esta forma se carga un Sistema Operativo Ubuntu 22.04 junto a la configuración hardware diseñada. Puedes verificar el éxito del arranque accediendo a la terminal del sistema desde Putty. Desde aquí, podrás confirmar mediante mensajes en la terminal si el Sistema Operativo se ha cargado correctamente o si ha surgido algún problema durante el proceso. Alternativamente, también puedes utilizar la terminal de Jupyter Notebooks, ya que el firmware PYNQ simplifica la programación con Python desde este entorno. Para verificar si el bitstream ha sido cargado se debe encender el led verde *DONE* de la placa pynq-z2.

2.3. Demostración final

El montaje del sistema es similar al realizado en la figura 1.13 del apartado anterior. Para comprobar su funcionamiento se puede ir cambiando las posiciones de memoria a las que apunta a los registros de cada *IP AXI* y configurar los parámetros que se explicaron anteriormente. El comando utilizado para el envío de señales a través de la terminal es devmem2.

La frecuencia de reloj a la que funciona por defecto el sistema es de 100 MHz. La señal de disparo verde tiene una frecuencia de 400 Hz siendo el ciclo de trabajo de 50 %, mientras que la señal de disparo roja tiene una frecuencia de 333.33 Hz con un ciclo de trabajo de 33.33 %.

```
# Comprobación de valor ID del Core
$ sudo devmem2 0x43C2001C

# Control Modo 1 (envío continuo de triggers)
$ sudo devmem2 0x43C20000 w 1

# Offset del trigger
$ sudo devmem2 0x43C20004 w 0

# Duty del trigger
$ sudo devmem2 0x43C20008 w 125000

# Periodo entre triggers
sudo devmem2 0x43C2000C w 250000
```

Listing 2.1: Señal trigger C1 (verde)

```
# Comprobación de valor ID del Core
$ sudo devmem2 0x43C2001C

# Control Modo 1 (envío fijo de 1 trigger)
$ sudo devmem2 0x43C20000 w 1

```

```
# Offset del trigger
$ sudo devmem2 0x43C20004 w 200000

# Duty del trigger
$ sudo devmem2 0x43C20008 w 100000

# Periodo entre triggers
sudo devmem2 0x43C2000C w 300000
```

Listing 2.2: Señal trigger C2 (rojo)

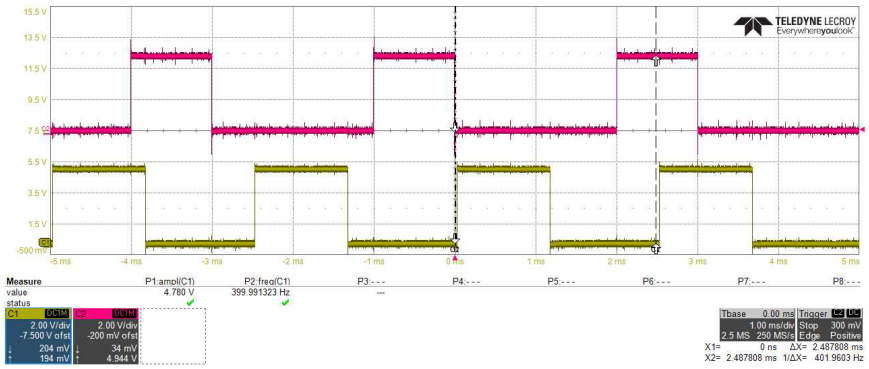


Figura 2.2: Señal de disparo a 400 Hz (verde) y a 333.33 Hz (rojo). Fuente: propia

A continuación se muestra otra prueba. Se inyectan dos señales de disparo con frecuencias de 333.33 Hz y 450 Hz, cuyos ciclos de trabajo del 20 % y 9 %, respectivamente. También se han configurado dos señales de interbloqueo, activada la señal del primer bit de la señal de interbloqueo y desactivada la del segundo bit. Se recuerda que para este proyecto se han configurado dos bits de interbloqueos generales.

```
# Comprobación de valor ID del Core
$ sudo devmem2 0x43C2001C

# Control Modo 1 (envío continuo de triggers)
$ sudo devmem2 0x43C20000 w 1

# Offset del trigger
$ sudo devmem2 0x43C20004 w 220000

# Duty cycle del trigger
$ sudo devmem2 0x43C20008 w 20000

# Periodo entre triggers
sudo devmem2 0x43C2000C w 240000
```

Listing 2.3: Señal trigger C1 (verde)

```
# Comprobación de valor ID del Core
$ sudo devmem2 0x43C2001C

# Control Modo 1 (envío fijo de 1 trigger)
$ sudo devmem2 0x43C20000 w 1

```

```
# Offset del trigger
$ sudo devmem2 0x43C20004 w 0

# Duty cycle del trigger
$ sudo devmem2 0x43C20008 w 60000

# Periodo entre triggers
sudo devmem2 0x43C2000C w 240000
```

Listing 2.4: Señal trigger C2 (rojo)

```
# Apagar la señal azul y encender la señal verde
$ sudo devmem2 0x43C00010 w 1
```

Listing 2.5: Señales de interbloqueo (verde y azul)

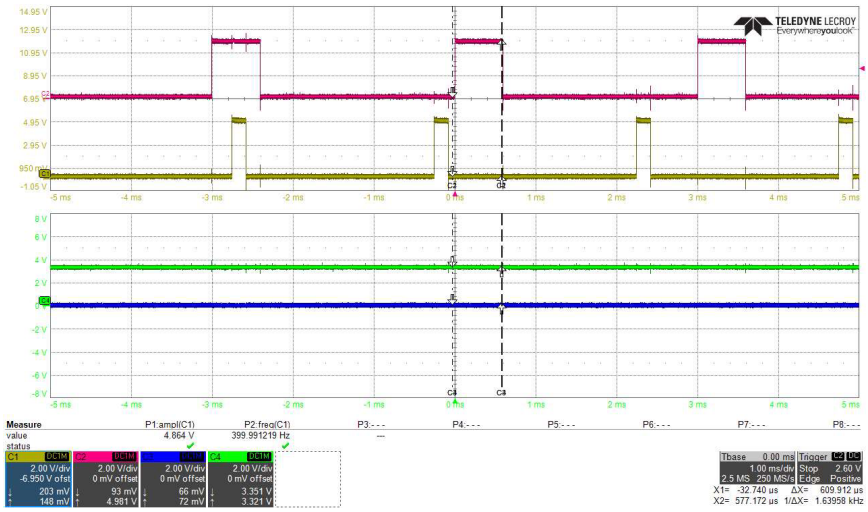


Figura 2.3: Señales de disparo a 450 Hz y 333.33 Hz, y señales de interbloqueo activadada y a desactivada. Fuente: propia

Acceder a las posiciones de memoria desde el sistema operativo embebido en la placa puede simplificar tanto las pruebas como la implementación final. En esta etapa, el siguiente paso crucial es desarrollar el sistema de control distribuido, donde se pueda escribir directamente los valores adecuados en los registros sin depender del comando devmem2 en la terminal. Esto permitirá una integración más directa y automatizada de las funcionalidades del sistema, mejorando la eficiencia y la fiabilidad del proceso.

Capítulo 3

Tango-Controls

Hasta este punto, se han alcanzado los objetivos principales de este proyecto. No obstante, es crucial que el ajuste de los parámetros configurables del sistema sea lo más intuitivo posible para el usuario. En el anterior apartado se deben cambiar las posiciones de memoria desde la terminal del sistema operativo embebido en la pynq-z2. Este método puede conllevar a cierta incertidumbre cuando se realizan pruebas y no se tiene a mano estos valores en hexadecimal. Por consiguiente, tango-controls no solo proporciona una interfaz gráfica intuitiva para el usuario, sino que también es un sistema controlado al cual se puede acceder desde varios PCs siempre que exista la conexión adecuada. Asimismo se podrían realizar varios proyectos en placas distintas siendo cada una de ellas un *Device Server* y controladas desde un único PC o varios dependiendo de la aplicación que se quiera dar. Las herramientas finalmente utilizadas han sido Pogo, Jive y Taurus.

3.1. Pogo

El primer paso que se debe realizar es crear una plantilla con algunos atributos que se van a necesitar en Pogo. El lenguaje de programación elegido es Python por lo que generará el archivo .py que se ha de modificar para añadir otros atributos y funciones para el acceso a memoria de la pynq-z2, tanto para escribir y para leer los registros tal y como se hacía en el apartado 2 de esta parte de la memoria.

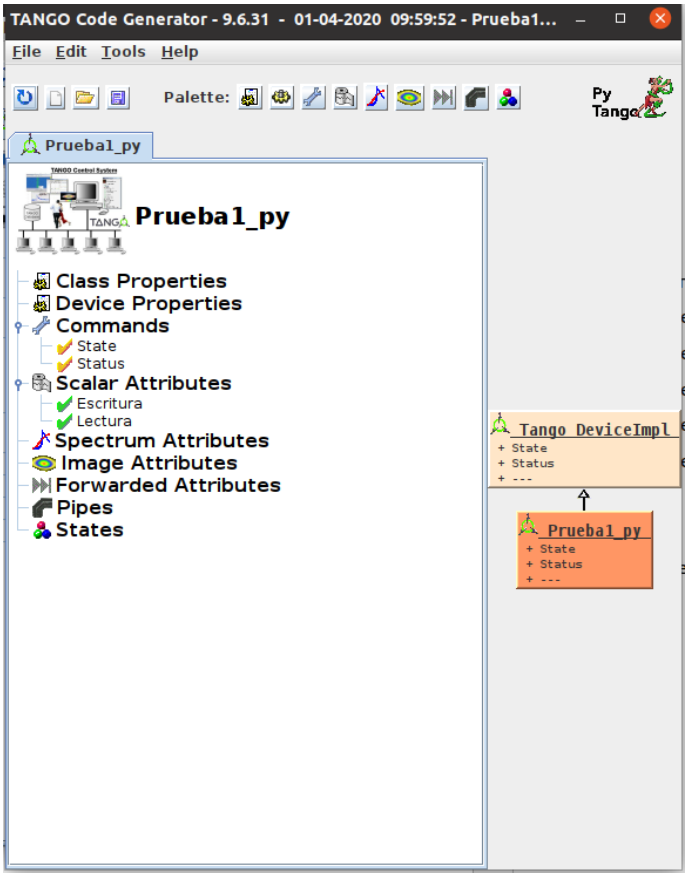


Figura 3.1: Creación de la plantilla con POGO. Fuente: propia

Se exporta la plantilla a un único archivo python (.py). Desde esta plantilla se han añadido otros atributos referidos a las 4 señales de disparo que se han diseñado para este proyecto y una única señal de interbloqueo. Todo esto se realiza accediendo a posiciones de memoria con la función mmap. Esta función está reemplazando al comando devmem2 del apartado 2. Un ejemplo de lectura del valor ID del registro 7 del trigger 1 se debe configurar de la siguiente manera y convertirlo a string.

```
#ID1
##Lectura de direccion de memoria de ID
# Dirección de memoria a la que deseamos acceder
memory_address = 0x43C2001C


# Tamaño de la región de memoria a mapear (puede variar según tus necesidades)
memory_size = 4096

# Desplazamiento dentro de la página de memoria
page_size = mmap.PAGESIZE
page_base = (memory_address // page_size) * page_size
page_offset = memory_address % page_size

try:
    # Abre /dev/mem con permisos de lectura/escritura
    with open("/dev/mem", "rb") as f:
```

```
        # Mapear la memoria
        mem = mmap.mmap(f.fileno(), memory_size, mmap.MAP_SHARED, mmap.
            PROT_READ, offset=page_base)

        # Leer 4 bytes desde la dirección especificada
        mem.seek(page_offset)
        data = mem.read(4)

        # Interpretar los bytes leídos como un entero de 32 bits
        value = struct.unpack('<I', data)[0]

        hex_value = hex(value)
        hex_string = format(value, 'x')
        global ID
        ID=str(hex_string)

        print(f"Valor leído desde 0x{memory_address:X}: {value} (hex: {
            hex_value}) (string: {hex_string}) (ID: {ID})")

        # Cerrar el mapeo de memoria
        mem.close()
except PermissionError as e:
    print("PermissionError:", e)
    print("Asegúrate de tener permisos adecuados o haber configurado
        correctamente los parámetros del kernel.")
```

Listing 3.1: Lectura ID del trigger 1

Para la escritura en una posición de memoria varía un poco en comparación con la lectura, pero la metodología es la misma. Un ejemplo de escritura del control (registro 1 de la señal de disparo) sería:

```
def write_Control1(self, attr):
    self.debug_stream("In write_Control1()")
    data_w = attr.get_write_value()

    #----- PROTECTED REGION ID(Prueba1_py.Escritura_write) ENABLED START -----#
     ##Escritura de direccion de memoria de triggers

    # Dirección de memoria a la que deseamos acceder
    memory_address_write = 0x43C20000
    # Tamaño de la región de memoria a mapear (puede variar según tus
        necesidades)
    memory_size_write = 4096

    # Desplazamiento dentro de la página de memoria
    page_size_write = mmap.PAGESIZE
    page_base_write = (memory_address_write // page_size_write) *
        page_size_write
    page_offset_write = memory_address_write % page_size_write


    try:
        # Abre /dev/mem con permisos de lectura/escritura
        with open("/dev/mem", "rb+") as f:
            # Mapear la memoria
            mem_write = mmap.mmap(f.fileno(), memory_size_write, mmap.
                MAP_SHARED, mmap.PROT_WRITE | mmap.PROT_READ, offset=
```

```
                page_base_write)
24
25              # Leer 4 bytes desde la dirección especificada
26              mem_write.seek(page_offset_write)
27              data_to_write = struct.pack('<I', data_w)  # Ejemplo de dato a
                    escribir
28
29              mem_write.write(data_to_write)
30
31
32              # Cerrar el mapeo de memoria
33              mem_write.close()
34      except PermissionError as e:
35          print("PermissionError:", e)
36          print("Asegúrate de tener permisos adecuados o haber configurado
                correctamente los parámetros del kernel.")
```

Listing 3.2: Escritura registro 1 del trigger 1

Para cada señal de disparo e interbloqueo se ha realizado una clase distinta en el archivo de python. Finalmente se debe subir al almacenamiento de la pynq-z2 para que se ejecute en la placa y acceder a las posiciones de memoria que se han cargado previamente con el bitstream.

3.2. Jive

Paralelamente se crea un servidor en Jive añadiendo el *ClassName* y en *Device* nombrándolo siguiendo la convención: dominio/familia/miembro. Se crea un servidor llamado Prueba1_py/test que debe coincidir con el nombre del fichero y con la instancia que se ha de ejecutar, en este caso test. En el apartado de Class se colocan las clases que se han declarado en el archivo de python, en nuestro caso Trigger1, Trigger2, Trigger3, Trigger4 e Interlock y finalmente los Dispositivos que se utilizan, en este caso es un dispositivo por clase únicamente.

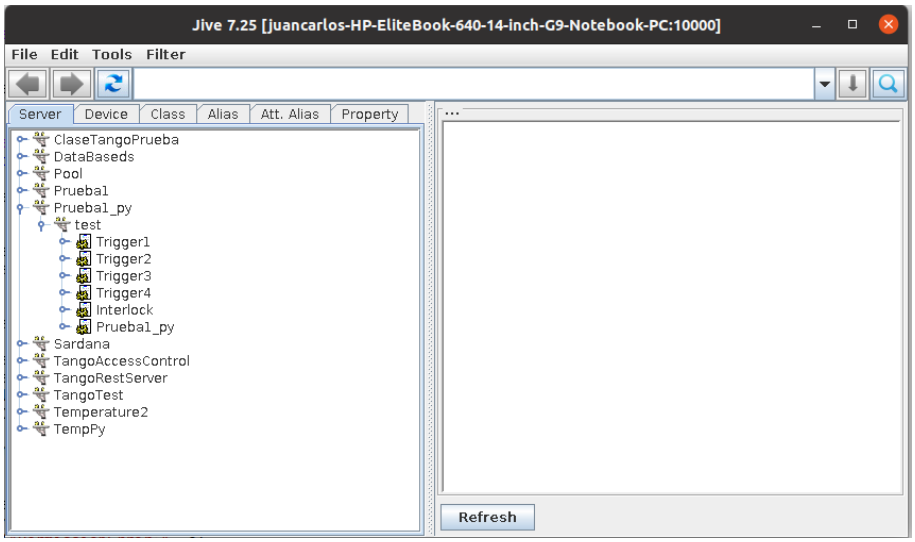


Figura 3.2: Creación de servidor con Jive. Fuente: propia

Jive proporciona la interfaz gráfica automática que genera llamada ATKPanel donde se puede visualizar e interactuar con los distintos parámetros. Como se puede ver en la figura 3.3 no es muy completa y a la hora de operar con esta interfaz tiene cierto delay al cambiar las variables.

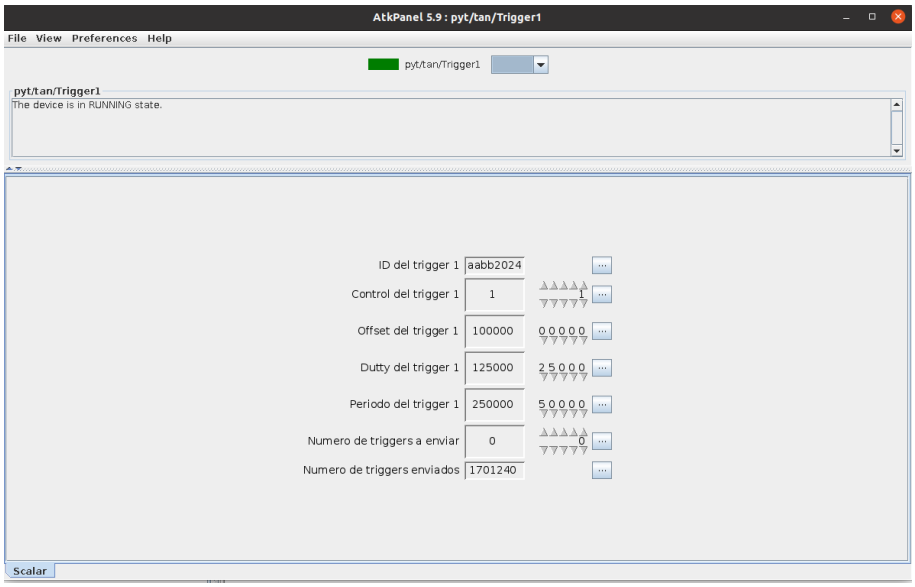


Figura 3.3: Interfaz gráfica automática generada por atkpanel y jive. Fuente: propia

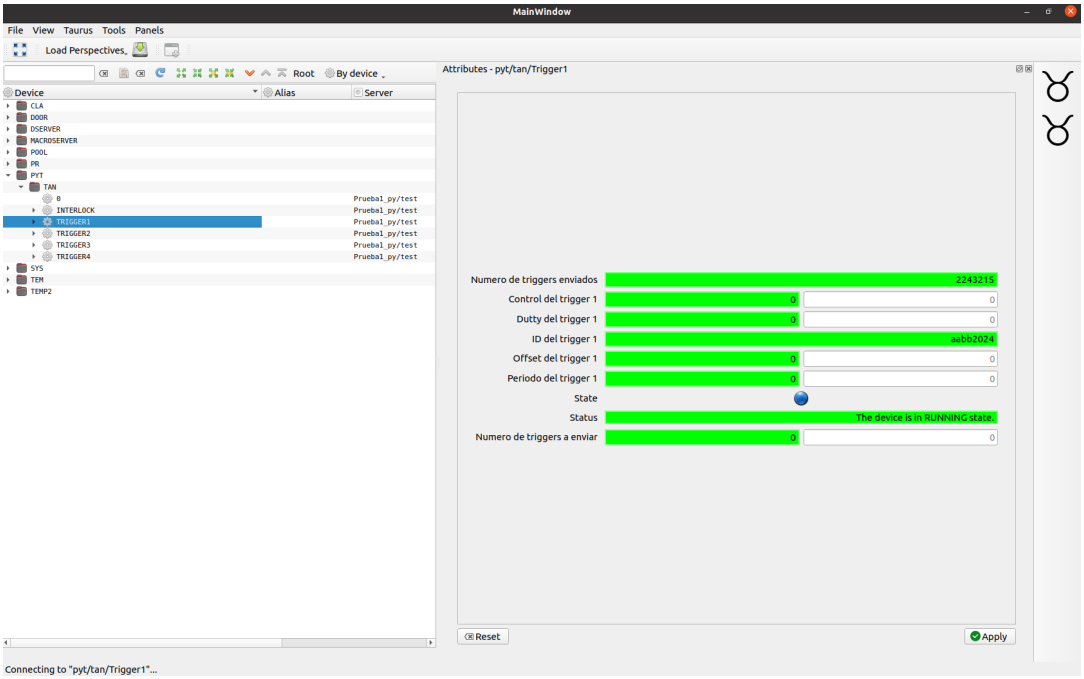
3.3. Taurus

Con el propósito de mejorar la calidad de la interfaz, es posible añadir *widgets* o incluso varios *Device Servers* en una única ventana gráfica. Taurus ofrece más posibilidades y es más versátil ya que se podría realizar un panel más complejo utilizando Qt Designer. En este caso, se ha realizado una interfaz gráfica con los registros a controlar de cada atributo.

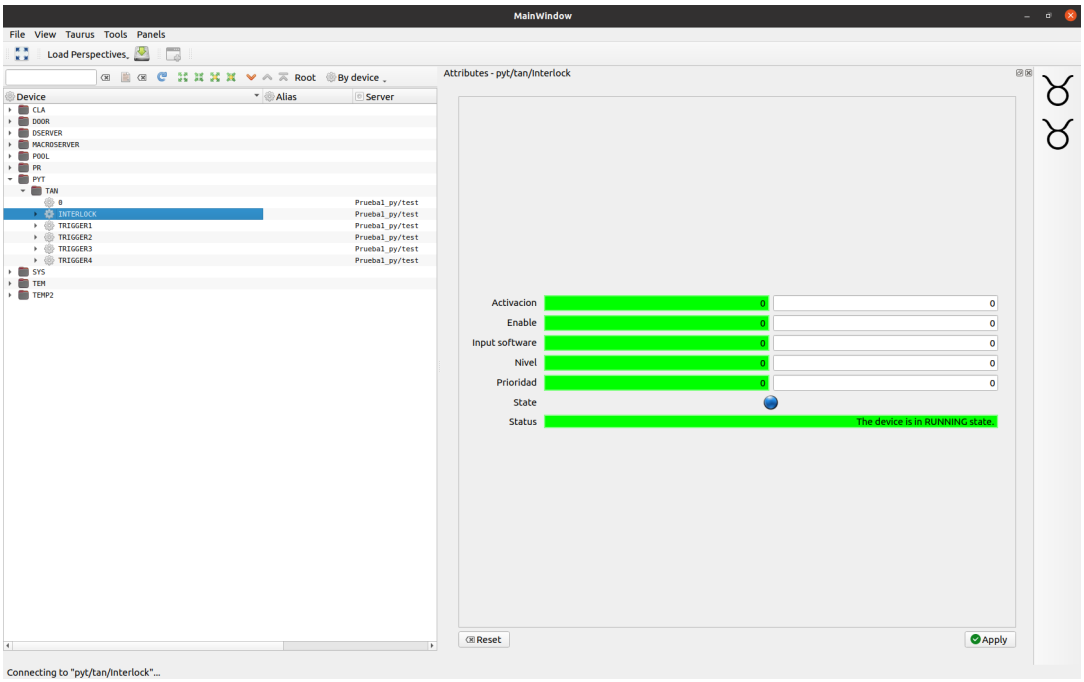
En la figura 3.4a se pueden observar los registros que se pueden modificar en la señal de disparo (trigger), un total de 7 registros, de los cuales 2 son de solo lectura: el ID y el número de señales enviadas. Los 5 restantes son de escritura: control, periodo, ciclo de trabajo (duty), offset y número de triggers a enviar. Además, se puede ver en la ventana adyacente en verde el valor que se ha escrito.

Por otro lado, en la figura 3.4b hay un total de 5 registros de escritura: activación, habilitación, entrada de software, nivel y prioridad. Tanto para los triggers como para el interlock, existe una señal llamada State que indica su estado actual, y otro atributo denominado Status que indica si está en modo *RUNNING* o *OFF*. De esta manera, es posible ajustar en tiempo real los parámetros de escritura y consultar los parámetros configurados previamente en Vivado.

En total, hay cuatro señales de disparo y un interlock que es global para todo el sistema. En la parte izquierda de la figura 3.4 se visualizan Trigger1 hasta Trigger4 y el Interlock.



(a) Trigger

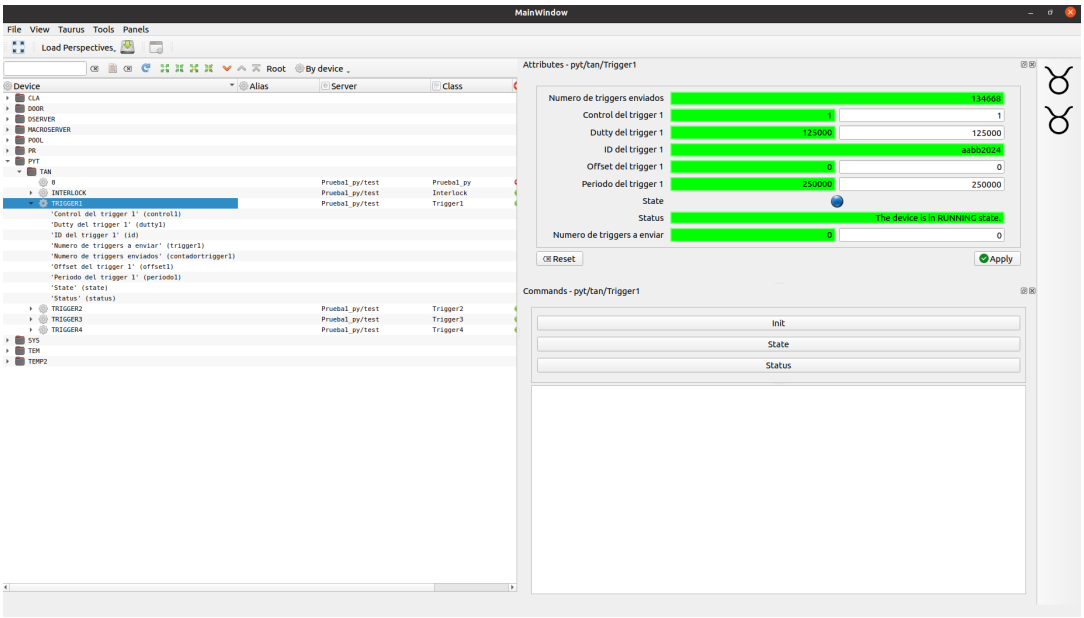


(b) Interlock

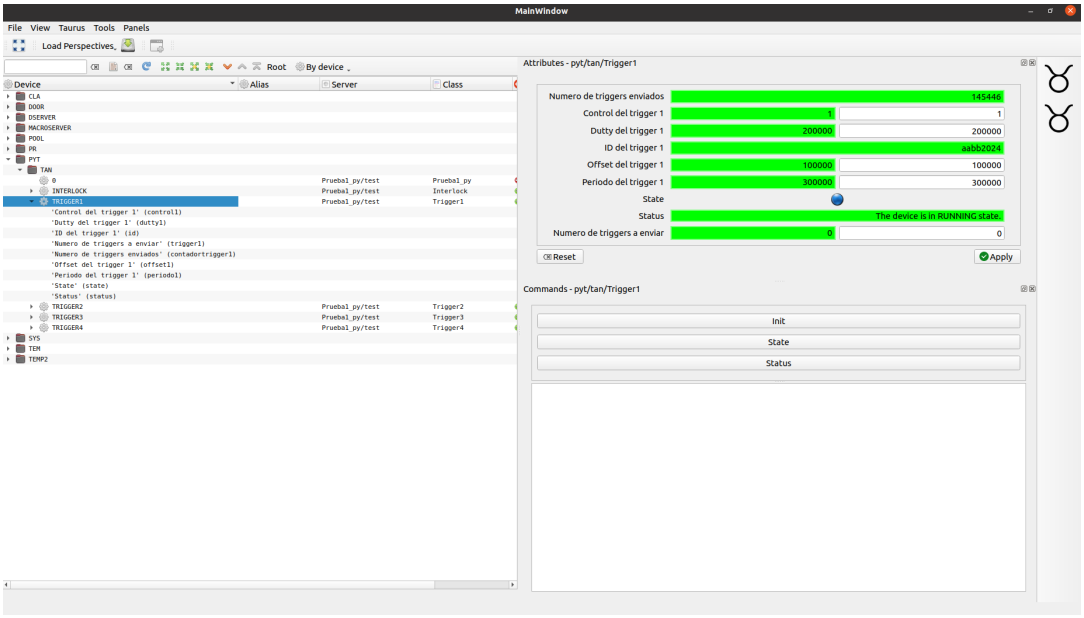
Figura 3.4: Interfaz gráfica generada por Taurus. Fuente: propia

Haciendo referencia a la demostración final realizada en el capítulo 1.3 del apartado 2.3, se ha llevado a cabo la configuración de las señales de disparo e interbloqueo con las características mencionadas en dicho apartado, utilizando esta vez la interfaz gráfica para dicha configuración. Las señales de la figura 2.2 (frecuencias de 400 Hz y 333.33 Hz) se configuran según lo indicado

en la figura 3.5. De igual manera, las señales de la figura 2.3 (frecuencias de 450 Hz y 333.33 Hz) se configuran conforme a lo mostrado en la figura 3.6.

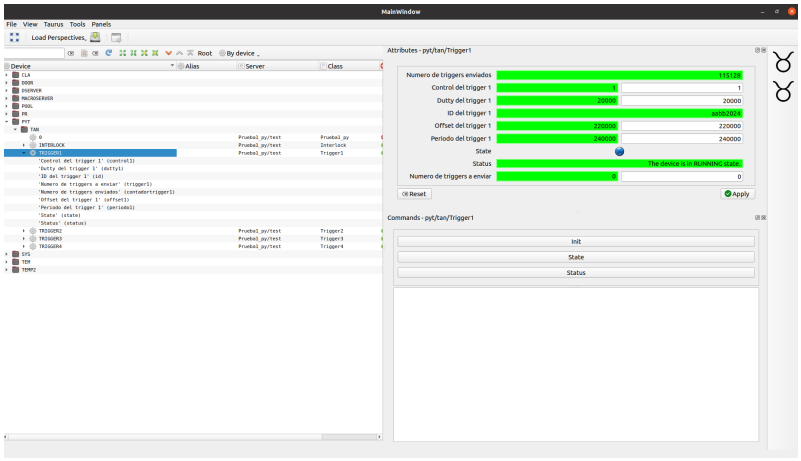


(a) Trigger 1 (señal verde)

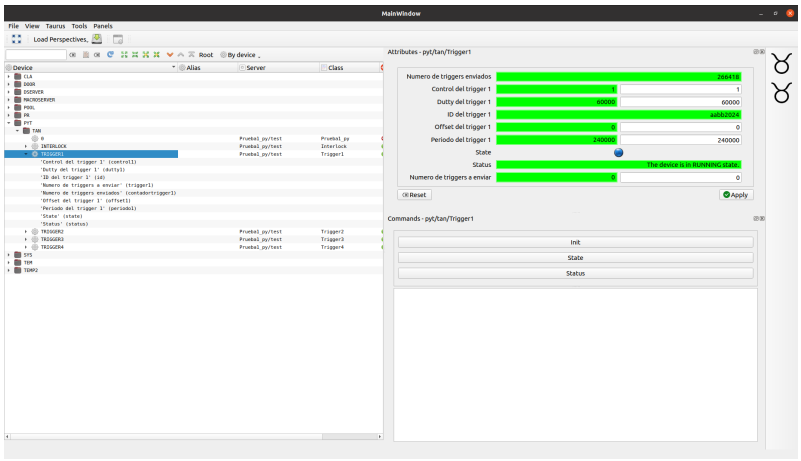


(b) Trigger 2 (señal roja)

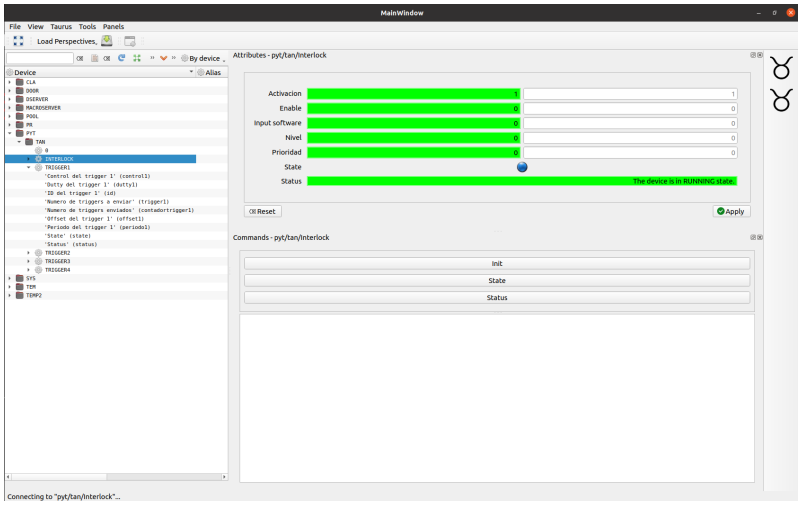
Figura 3.5: Interfaz gráfica final. Fuente: propia



(a) Trigger 3 (señal verde)



(b) Trigger 4 (señal roja)



(c) Interlocks

Figura 3.6: Interfaz gráfica final. Fuente: propia

Parte IV

Conclusiones y trabajo futuro

Capítulo 1

Conclusiones

En este trabajo de fin de máster, se ha desarrollado e implementado un sistema de control distribuido para un laboratorio de RF-HG utilizando la plataforma tango-controls. A lo largo del proyecto, se lograron los siguientes objetivos:

- **Estudio de la disipación térmica**

  Uno de los objetivos ha sido la verificación de la fiabilidad del sistema. Un factor a tener en cuenta en el diseño de PCBs es la correcta disipación térmica de los componentes electrónicos. En este proyecto había un problema con un driver que disipa una potencia bastante elevada.Como consecuencia, para periodos largos de consumo puede conllevar a la inutilización del driver. Además estos componentes son necesarios para el funcionamiento del sistema ya que todas las salidas de la Interfaz I/O pasan por ellos.

- **Desarrollo del Sistema Digital**

  El objetivo principal ha sido el desarrollo del código hardware para la generación de señales de disparo e interbloqueos. Se ha podido realizar de la forma más simplificada posible. Se han diseño los correspondientes *Custom Core IP AXI* para cada uno de ellos. La ventaja de realizarlo de esta manera es la versatilidad y ayuda en próximOs proyectos, o en la continuación del mismo.

- **Desarrollo del Sistema de Control**

  Se diseñó un sistema de control distribuido que permite la gestión eficiente de los componentes del laboratorio de RF. Este sistema asegura una comunicación efectiva entre los diferentes dispositivos y una supervisión adecuada de sus estados operativos. El uso de la plataforma tango-controls ha permitido integrar de manera coherente las diversas partes del sistema, facilitando su operación y mantenimiento.

- **Implementación de Interfaz Gráfica**

  A partir del sistema de control distrbuido, se ha creado una interfaz gráfica intuitiva, que facilita la interacción de los usuarios con el sistema de control. Permite una visualización clara y precisa de las señales y estados de los dispositivos, mejorando la eficiencia operativa y reduciendo el riesgo de errores humanos. La interfaz gráfica se desarrolló teniendo en cuenta las necesidades específicas del laboratorio, permitiendo a los usuarios realizar ajustes y monitorear el sistema en tiempo real.

- **Pruebas y Validación**

  El sistema fue sometido a rigurosas pruebas para asegurar su correcto funcionamiento en diversas condiciones operativas. Las pruebas incluyeron la verificación de la sincronización de señales, la capacidad de respuesta del sistema ante diferentes escenarios y la robustez de la comunicación entre los dispositivos. Los resultados de las pruebas demostraron que el sistema cumple con los requisitos establecidos, garantizando una operación fiable y segura.

- **Documentación y Transferencia de Conocimiento**

  Se generó una documentación detallada del sistema, esencial para asegurar la continuidad del proyecto y facilitar futuras actualizaciones. En la memoria se proporcionan ejemplos que permiten seguir en desarrollo, pero todo el código está disponible en los repositorios del laboratorio. Además, se llevarán a cabo sesiones de transferencia de conocimiento para futuros operadores del laboratorio, asegurando que comprendan completamente el funcionamiento del sistema y puedan operarlo de manera eficaz.

- **Impacto en la Eficiencia y Seguridad**

  El sistema de control distribuido desarrollado ha mejorado significativamente la eficiencia operativa del laboratorio de RF. Al automatizar y centralizar el control de los dispositivos, se ha reducido el tiempo necesario para configurar y monitorear los experimentos. Además, la implementación de este sistema ha incrementado la seguridad del laboratorio, protegiendo tanto al personal como a los equipos frente a posibles fallos operativos.

- **Contribución al Campo de la Radioterapia**

  Este proyecto contribuye al avance en el campo de la radioterapia con partículas pesadas. La precisión y fiabilidad del sistema de control son cruciales para la investigación en cavidades de aceleradores de partículas, cuyo objetivo es mejorar la eficiencia de los tratamientos de radioterapia para el cáncer. El desarrollo de este sistema no solo beneficia a la investigación actual, sino que también sienta las bases para futuras innovaciones en este ámbito.

Capítulo 2

Proyecto futuro

A partir de los logros y las conclusiones obtenidas en este trabajo, se proponen las siguientes líneas de trabajo futuro para mejorar y expandir el sistema desarrollado:

- **Optimización de la Comunicación**

  Investigar y aplicar técnicas avanzadas de optimización de la comunicación entre los componentes del sistema para mejorar la eficiencia y reducir la latencia. Esto puede incluir el uso de protocolos de comunicación más rápidos y fiables. Por otra parte, podría realizarse la mejora del código en Python para acceder a las posiciones en memoria de una forma más simple, ya que el código ha sido bastante extenso y se podría realizar de una forma más simplificada.

- **Integración de Nuevos Dispositivos**

  Incorporar nuevos dispositivos y sensores al sistema de control para ampliar su funcionalidad y adaptarlo a diferentes tipos de experimentos y necesidades del laboratorio. Esto permitirá que el sistema sea más versátil y pueda manejar una mayor variedad de situaciones operativas. Un ejemplo es implementar un sistema microTCA en el que estén integradas varias FPGAs y que cada una haga una función en específico. De esta forma el sistema tendrá mejor capacidad computacional que ayudará a la gestión de canales y procesado de señal.

- **Procesado de Señal**

  En algunos apartados se ha comentado que este proyecto abarca una pequeña parte del sistema completo del laboratorio de RF. El siguiente paso es realizar procesado de la señal utilizando algoritmo CORDIC, modulación IQ, gestionar el vector modulador del sistema... Este proyecto ayuda al entendimiento general del sistema y a dar ese primer paso.

- **Desarrollo de Algoritmos de Control Avanzados**

  Implementar algoritmos de control más sofisticados que permitan una gestión más precisa y adaptativa de los componentes del laboratorio. Estos algoritmos pueden considerar variables como la temperatura, la humedad y otros factores ambientales, ajustando automáticamente los parámetros del sistema para mantener un rendimiento óptimo.

- **Seguridad y Resiliencia**

Mejorar las medidas de seguridad del sistema para protegerlo contra posibles fallos y ataques cibernéticos. Esto incluye la implementación de mecanismos de autenticación y encriptación de datos, así como el desarrollo de procedimientos de recuperación ante desastres que aseguren una operación continua y segura.

- **Formación y Capacitación**

  Desarrollar programas de formación y capacitación para los nuevos usuarios del sistema, asegurando que el conocimiento y las mejores prácticas se transfieran efectivamente. Esto ayudará a mantener un alto nivel de competencia operativa y asegurar que el sistema sea utilizado de manera eficaz y segura.

Estas propuestas de trabajo futuro buscan no solo mejorar el sistema actual, sino también asegurar su adaptabilidad y sostenibilidad a largo plazo, permitiendo que continúe siendo una herramienta valiosa para la investigación y desarrollo en el campo de la radioterapia con partículas pesadas.

Bibliografía

[1] S. Pellejero S. Lozares F. Mañeru. “Radioterapia con partículas pesadas”. En: *Anales del Sistema Sanitario de Navarra* (2008).

[2] OCN, Melissa, AGN-BC, Patricia. “Optimizing the Teachable Moment for Health Promotion for Cancer Survivors and Their Families”. En: *Journal of the Advanced Practitioner in Oncology* (2016).

[3] K.PEACH (PhD, FInstP), P. WILSON (MSc, PhD) and B. JONES (MSc, MD). “Accelerator science in medical physics”. En: *The British Journal of Radiology* (2011).

[4] WH Wragg y R Kleeman. “On the ionization curves of Radium”. En: *Philosophical Magazine* 8 (1904), págs. 726-738.

[5] Anna Vnuchenko. “High-Gradient issues in S-band RF Acceleration Structure for Hadrontherapy accelerators and Radio Frequency Quadrupoles”. Supervisors: Dr. Angeles Faus Golfe and Dr. Benito Gimeno Martínez. Tesis doct. Doctorat en Física, dic. de 2019.

[6] U. Amaldi et al. “LIBO—a linac-booster for protontherapy: construction and tests of a prototype”. En: *TERA Foundation/CERN/Universita degli Studi di Milan/INFN Section of Milano/University Federico II of Naples/INFN Section of Naples* (2003).

[7] “Particle Therapy Cooperative Group”. En: *PTCOG* (2024).

[8] D. Esperante. “CONSTRUCTION AND COMMISSIONING OF THE S-BAND HIGH-GRADIENT RF LABORATORY AT IFIC”. En: *CERN, Geneva, Switzerland, IFIC (CSIC-Univ. Valencia), Valencia, Spain, Univ. Valencia, Valencia, Spain, LAL, Univ. Paris-Sud, CNRS/IN2P3, Univ. Paris-Saclay, Orsay, France* (2018).

[9] “What is a PCB Stencil?” En: (2021). URL: `https://www.pcbdirectory.com/community/what-is-a-pcb-stencil`.

[10] José F. Toledo Alarcón. “Diseño térmico en equipos electrónicos: una introducción”. En: *Departamento de Ingeniería Electrónica - Universitat Politècnica de València* (2022).

[11] Texas Instruments. “Thermal Design by Insight, Not Hindsight”. En: *Application Note SPRA953D* (Available online). URL: `https://www.ti.com/lit/an/spra953d/spra953d.pdf?ts=1720063018845`.

[12] Analog Devices. “Application Note AN-103”. En: *Analog Devices* (Available online). URL: `https://www.analog.com/media/en/technical-documentation/application-notes/an103fb.pdf`.

[13] Infineon Technologies. “Application Note AN-1030”. En: *Infineon Technologies* (Available online). URL: `https://www.infineon.com/dgdl/an-1030.pdf?fileId=5546d462533600a40153559132280f74`.

[14] "SN5545xB, SN7545xB Dual-Peripheral Drivers for High-Current, High-Speed Switching". En: (2017). URL: https://www.ti.com/lit/ds/symlink/sn75452b.pdf?ts=1717575846713&ref_url=https%253A%252F%252Fwww.mouser.es%252F.

[15] Stephen M. (Steve) Trimberger. "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology". En: *IEEE Solid-State Circuits Magazine* (2018). URL: https://ieeexplore.ieee.org/abstract/document/7086413.

[16] "Python productivity for Zynq (Pynq) v2.5". En: (2018). URL: https://pynq.readthedocs.io/en/v2.5/pynq_overlays/pynqz2/pynqz2_base_overlay.html.

[17] "PYNQ". En: (2024). URL: https://www.pynq.io/.

[18] "Tango Documentation Contents". En: (2023). URL: https://tango-controls.readthedocs.io/en/latest/development/overview.html#c-and-python.

[19] "PYNQ-Z2 Reference Manual v1.1". En: (2019). URL: https://dpoauwgwqsy2x.cloudfront.net/Download/PYNQ_Z2_User_Manual_v1.1.pdf.