Estudio, implementación sobre una FPGA y análisis de desempeño de un "Soft Processor" basado en la arquitectura RISC V

Llumiquinga Quelal, Richard Alexander

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Ingeniería en Electrónica y Telecomunicaciones

Trabajo de titulación, previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones

Ing. Ramos Vargas, Pablo Francisco, PhD.

26 de Julio del 2022

COPYLEAKS

Tesis_Llumiquinga_Richard_RISC-V_rev_final_PR.pdf

Scanned on: 13:21 July 28, 2022 UTC

Firmado electrónicamente por:
PABLO FRANCISCO
RAMOS VARGAS

2.0%	24	18359
Overall Similarity Score	Results Found	Total Words in Text

Identical Words	255
Words with Minor Changes	4
Paraphrased Words	110
Omitted Words	0

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Ingeniería en Electrónica y Telecomunicaciones

Certificación

Certifico que el trabajo de titulación: **Estudio, implementación sobre una FPGA y análisis de desempeño de un "Soft Processor" basado en la arquitectura RISC V,** fue realizado por el señor Llumiquinga Quelal Richard Alexander; el mismo que cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, además fue revisado y analizado en su totalidad por la herramienta de prevención y/o verificación de similitud de contenidos; razón por la cual me permito acreditar y autorizar para que se lo sustente públicamente.

Sangolquí, 26 de Julio del 2022

PABLO FRANCISCO
RAMOS VARGAS

.....

Ing. Ramos Vargas Pablo Francisco, PhD

C. C 1712447976

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Ingeniería en Electrónica y Telecomunicaciones

Responsabilidad de Autoría

Yo, **Llumiquinga Quelal, Richard Alexander**, con cédula de ciudadanía n° 1718516592, declaro que el contenido, ideas y criterios del trabajo de titulación: **Estudio, implementación sobre una FPGA y análisis de desempeño de un "Soft Processor" basado en la arquitectura RISC V** es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 26 de Julio del 2022

.....

Llumiquinga Quelal, Richard Alexander

C. C 1718516592

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Ingeniería en Electrónica y Telecomunicaciones

Autorización de Publicación

Yo, **Llumiquinga Quelal, Richard Alexander**, con cédula de ciudadanía n° 1718516592, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **Estudio, implementación sobre una FPGA y análisis de desempeño de un "Soft Processor" basado en la arquitectura RISC V** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 26 de Julio del 2022

.....

Llumiquinga Quelal, Richard Alexander

C. C 1718516592

Dedicatoria

Dedico mi trabajo de titulación a mis padres Segundo y Janeth, por su incondicional apoyo y su firme convicción en mí. Gracias por haberme guiado por el camino correcto no solo en mis estudios sino durante toda mi vida. Muchas gracias por creer en mí y por alentarme cada día a seguir adelante y nunca rendirme. Los amo y los aprecio con todo mi corazón.

A mis hermanos Franklin y Daniel, por su apoyo y ánimo para poder completar con éxito mi trabajo de titulación. Les agradezco por su preocupación y por siempre estar allí para mí, los quiero y los admiro por ser grandes personas y por el esfuerzo que veo cada día en ustedes.

A mis abuelitos Heriberto y Graciela que, a pesar de la distancia, siempre han estado atentos y se han preocupado por mí. Muchas gracias por todo su amor y cariño.

A mis amigos con los que viví buenos momentos durante mi carrera universitaria William, Mishelle y Andrés. Les agradezco por ser maravillosos amigos y por su apoyo en cada momento. Muchas gracias por su amistad, cariño y compañía.

A mis demás familiares y amigos de la iglesia. Muchas gracias por sus palabras y consejos, este logro alcanzado es para todos ustedes.

Richard Llumiquinga

Agradecimiento

Agradezco en primer lugar a Dios, el amigo que siempre estuvo allí para mí, en cada decisión, en cada paso, guió mi camino, me cuidó y me ayudó a superar cada obstáculo. A mis padres y hermanos, por cuidar de mí y por todo su apoyo y cariño. Muchas gracias por todo su esfuerzo y dedicación.

Agradezco de todo corazón al Doctor Pablo Ramos, Director del proyecto de Titulación, por haber confiado en mí, por su paciencia y por todo el conocimiento y ayuda que ha servido de guía para que el presente trabajo de titulación logre completarse de manera satisfactoria.

Agradezco a la Universidad de las Fuerzas Armadas ESPE por haberme permitido cursar mi carrera en una institución de prestigio. Muchas gracias a todos los docentes y amigos con los que tuve la oportunidad de compartir y que me sirvieron de inspiración para seguir formándome en mi carrera profesional.

Índice de Contenido

Índice de tablas

Índice de Figuras

Resumen

En este documento se realiza un estudio de la ISA de RISC-V, se implementan dos softcores sobre una FPGA Nexys A7-100T y se muestra la comparación de los mismos en términos de utilización de recursos, consumo de potencia y rendimiento. Para lograr este objetivo se hace uso de hardware y software libre con el fin de implementar los SoCs NEORV32 y RVfpga en la tarjeta objetivo. Una vez que los sistemas están en funcionamiento se ejecuta el benchmark llamado Coremark, el cual permite la obtención de datos referentes a rendimiento. Los datos de utilización de recursos y consumo de potencia se obtienen directamente desde el entorno de desarrollo Vivado. Adicionalmente se realizan modificaciones a cada sistema para verificar si las instrucciones por ciclo que ejecutan se optimizan. Las variaciones realizadas a cada sistema permiten comprobar que el rendimiento de NEORV32 se incrementa cuando se utilizan menos extensiones mientras que para RVfpga, el agregar la memoria DCCM es el factor que influenció de manera positiva en su rendimiento. Con respecto a los recursos utilizados, se verificó que RVfpga ocupó una mayor cantidad de elementos de la FPGA debido a que el sistema es más grande y complejo que NEORV32. Con repecto a la potencia consumida por cada sistema, también se verificó que RVfpga consumió mucho más que NEORV32 llegando a tener un valor cercano a 1 Watio.

Palabras clave: RISC-V, Procesadores, Arquitectura, FPGA, Rendimiento.

Abstract

The present document aims at studying the RISC-V ISA by means of the implementation of two soft-cores on a Nexys A7-100T FPGA and their comparison in terms of resources, power consumption and performance. To achieve this goal, free hardware and software are used in order to implement the NEORV32 and RVfpga SoCs on the target board. Once the systems are in operation, a benchmark called Coremark is executed for evaluating the performance of each softcore. Resource utilization and power consumption information is obtained directly from the Vivado development environment. In addition, modifications were performed to each system to verify the optimization of the instructions per cycle executed. On one hand, modifications allow checking that the performance of NEORV32 increases when fewer extensions are used. On the other hand, adding the DCCM memory improves the performance of RVfpga. Regarding the resources, it was confirmed that RVfpga uses a greater number of FPGA elements than NEORV32, because the system is larger and more complex. From the power consumption point of view, it was shown that RVfpga consumed much more power than NEORV32, reaching a value close to 1 Watt. It was expected due to the quantity of FPGA resources used.

Key words: RISC-V, Processors, ISA, FPGA, Performance.

Acrónimos

- AMD: Advanced Micro Devices
- ARM: Advanced RISC Machine
- ASIC: Circuito Integrado de Aplicación Especifica
- AXI: Interfaz extensible avanzada
- BSD: Berkeley Software Distribution
- CISC: Computación de conjunto de instrucciones complejas
- CLI: Interfaz de línea de comandos
- CPU: Unidad central de procesamiento
- CRC: Verificación de redundancia cíclica
- DCCM: Data Closely-Coupled Memory
- DDR: Double Data Rate
- DSP: Digital Signal Processor
- FPGA: Field Programmable Gate Array
- GPIO: General Purpose Input/Output
- GPS: Sistema de posicionamiento Global
- IBM: International Business Machines
- ICCM: Instruction Closely-Coupled Memory
- IPC: Instrucciones por ciclo
- IRQ: Petición de interrupción
- ISA: Arquitectura de set de instrucciones
- ISC: Internet Systems Consortium
- JTAG: Joint Test Action Group
- LUT: Lookup Table

- MIPS: Microprocessor without Interlocked Pipeline Stages
- PMOD: Peripheral module interface
- PWM: Pulse Width Modulation
- RISC: Reduced instruction set computer
- SISC: Simple Instruction Set Computing
- SoC: System on a Chip
- SPARC: Scalable Processor ARChitecture
- UART: Transmisor-receptor asíncrono universal
- USB: Bus Universal en Serie
- VGA: Video Graphics Array
- VHDL: Very High Speed Integrated Circuit Hardware Description Language

Capítulo I

Introducción

La tecnología de computadores ha progresado rápidamente en los últimos años como consecuencia de la miniaturización de los transistores que han permitido una mayor densidad dentro de los circuitos electrónicos (Hennessy & Patterson, 1993). Los computadores se han convertido en herramientas básicas y fundamentales que desde su aparición en la vida del hombre han sufrido múltiples cambios, lo cual ha permitido clasificar su desarrollo tecnológico en generaciones. Un sistema de este tipo, está compuesto por componentes físicos conocidos como hardware, los cuales interactúan a través de conjuntos de instrucciones establecidas (software) para alcanzar un objetivo específico. En sí, los computadores actuales han mejorado sus características con el pasar del tiempo haciéndolos dispositivos más llamativos que prestan una gran utilidad a los usuarios y que además poseen un gran desempeño. A nivel de hardware se han producido varias mejoras: los procesadores poseen varios núcleos, son más veloces y cada vez son más modernos, las memorias han aumentado su espacio de almacenamiento y son más confiables, las pantallas han aumentado su resolución, las baterías tienen mayor duración, entre otras cosas. A nivel de software, se han generado mejoras de los programas a partir de sus versiones anteriores y se han añadido más funcionalidades, permitiendo así generar miles de proyectos a través de diferentes lenguajes de programación. En las últimas décadas, el uso de software libre se ha convertido en una de las principales tendencias tecnológicas ya que ha permitido que los usuarios tengan libertad total para modificar, ejecutar y mejorar sus programas, mientras que el hardware permanecía cerrado principalmente debido a limitaciones como por ejemplo documentación y licencias para obtener los permisos necesarios para poder hacer uso de diagramas esquemáticos o especificaciones para creación de hardware. No obstante, en las dos últimas décadas ha proliferado significativamente el Hardware libre y ha comenzado a tomar fuerza. En la actualidad, la mayoría de dispositivos que se utilizan diariamente se basan en procesadores con arquitecturas como x86 fabricados por

empresas como INTEL, AMD, o en la arquitectura ARM fabricados principalmente por Qualcomm, Apple o MediaTek. Actualmente, considerando los avances que se han producido en el diseño de hardware libre, han comenzado a surgir sets de instrucciones libres de patentes que han motivado la innovación en el campo tecnológico y han permitido que cualquier universidad o compañía pueda acceder y usar libremente estos sets de instrucciones (Parejo Quirós, 2016). Una de estas alternativas prometedoras que ha comenzado una revolución en el mercado es RISC-V.

Durante años han estado presentes en el mercado proyectos que utilizan software libre (open-source). En la actualidad esta tendencia está siendo complementada con los progresos en la arquitectura RISC-V que contribuye significativamente a la masificación del hardware libre (Súarez Santamaría, 2019). El proyecto RISC-V empezó en el año 2010 en la Universidad de California en Berkeley. Se trataba del desarrollo derivado de varios proyectos académicos de diseño de computadoras con fines investigativos y de docencia. Los proyectos RISC ya habían pasado por cuatro generaciones y en 2015 se crea RISC-V Foundation para controlar la evolución de la arquitectura del conjunto de instrucciones (ISA) que se había desarrollado inicialmente (Rocha Pacheco, 2019). RISC-V, se estableció como libre y abierta con un repertorio de instrucciones basado en ideas ya existentes, basada en una arquitectura de tipo RISC (Reduced Instruction Set Computing), cuyo enfoque es convertirse en un ISA universal.

El proyecto RISC-V hace uso de dispositivos como las FPGAs para la implementación de hardware basado en procesadores. Las FPGAs son dispositivos semiconductores basados en matrices de bloques reconfigurables, que permiten el hardware de sistemas computacionales en un solo chip (SoC), y que gracias a su estructura, resultan muy convenientes para la implementación de procesadores RISC. Desde su invención en 1984, la industria ha visto un crecimiento considerable en la producción de FPGAs los cuales permiten implementar diseños hardware tanto combinacionales como secuenciales, y que en la

actualidad pueden contener cientos de miles o incluso millones de puertas lógicas para ejecución simultánea (Shaya Zamudio Vissuet, 2003). Considerando que las FPGAs son idóneas para la implementación de un procesador RISC-V, y que estas permiten generar diseños de manera accesible y rápida, se logra que los desarrolladores académicos y empresariales tengan la posibilidad de crear hardware personalizado.

En este trabajo se plantea un primer acercamiento con las arquitecturas RISC-V mediante la implementación de dos procesadores RISC-V existentes sobre una FPGA. Para ello, en primer lugar se realizará una exploración teórica de la arquitectura RISC-V y de los diferentes procesadores existentes que se pueden implementar sobre una FPGA. Luego se procederá a seleccionar dos de ellos que de manera que puedan ser implementados en la tarjeta de desarrollo Nexys A7 de Xilinx. Se estudiará su estructura, lenguaje de programación, funcionamiento y herramientas necesarias para su implementación. Una vez establecidos todos los parámetros teóricos se realizará la implementación de los procesadores en la FPGA. Finalmente se realizarán pruebas de desempeño y consumo de energía de los dos procesadores utilizando benchmarks genéricos para establecer una comparación cualitativa entre ambos.

Objetivos

General

Implementar un procesador existente basado en una arquitectura RISC-V sobre una FPGA.

Específicos

- Realizar un estudio del estado del arte de la arquitectura RISC-V y una revisión de las especificaciones técnicas del conjunto de instrucciones.
- Estudiar distintas implementaciones de núcleos basados en RISC-V implementados sobre FPGA.
- Seleccionar dos procesadores existentes basados en la arquitectura RISC-V que utilicen lenguaje de programación compatible con la FPGA.
- Utilizar herramientas de software para la síntesis e implementación de los procesadores sobre una FPGA disponible.
- Realizar verificación del funcionamiento y pruebas de desempeño de dos arquitecturas RISC-V utilizando un benchmark existente o programado en lenguaje C.
- Comparar el desempeño de las dos arquitecturas RISC-V implementadas y analizar los resultados obtenidos.

Justificación e Importancia

La ausencia de una tecnología de referencia propia en materia de procesadores acentúa la dependencia hacia otros mercados tales como Estados Unidos o Europa, lo cual lleva consigo condicionamientos y restricciones como medidas de protección al uso de arquitecturas y diseños ya creados. El proyecto colaborativo RISC-V nació como respuesta a esta dependencia tecnológica como un conjunto de instrucciones abierto y disponible para que todos lo puedan usar. En general, las empresas licencian sus diseños para que luego sean usados por otras marcas para la creación de sus propios procesadores. Estas licencias tienen un costo y además toman cierto tiempo hasta que las empresas interesadas puedan comenzar a diseñar sus propios chips. Sin embargo, gracias a RISC-V, ahora cualquier compañía o grupo de investigación podrán diseñar sus propios procesadores sin necesidad de pagar regalías por licencias de uso. Además, ya que no está sujeto a los intereses de una empresa en particular, las empresas serán las que se impongan su propio ritmo de desarrollo y precios según la necesidad del usuario (Waterman & Asanovic, The RISC-V Instruction Set Manual, 2017).

Gracias a la flexibilidad de diseño, RISC-V ha permitido crear tecnología más optimizada ya que se puede añadir y quitar componentes según la aplicación que se esté desarrollando, es decir, se puede evitar la sobre arquitectura, al implementar procesadores que contengan sólo los elementos necesarios; como resultado se podría reducir el consumo de energía o incluso se podría minimizar costos, por lo que se podrá diseñar procesadores personalizados y a la vez aceptados mundialmente gracias al respaldo del concepto RISC-V.

En referencia a lo ya mencionado, se ha escogido la arquitectura RISC-V puesto que es una de las principales referentes del hardware libre, su set de instrucciones y especificaciones es pública, legal y gratuita para fabricar y distribuir chips basados en esta arquitectura. Además, la creación de arquitecturas RISC-V permitiría ganar cierta independencia tecnológica.

Dado el interés que este estándar de repertorio de instrucciones abierto, libre y personalizable genera a nivel mundial, tanto en la academia como en las empresas, se estima indispensable introducir la investigación de la arquitectura RISC-V dentro del área de Sistemas Digitales del Departamento de Eléctrica, Electrónica y Telecomunicaciones (DEEL) de la ESPE, con el objetivo de comenzar a diseñar procesadores propios. Cabe mencionar que el DEEL ha sido líder en incursionar en el diseño de sistemas on chip basado en FPGA, tanto en investigación, como en sus asignaturas del área de sistemas digitales para las carreras afines. Por lo tanto, existe un fuerte conocimiento teórico y práctico para que un estudiante pueda abordar este tema que puede ser complejo sin la experiencia apropiada. Por tal motivo, el presente proyecto de titulación introduce la temática por medio de la implementación de dos procesadores existentes basados en arquitectura RISC-V sobre una FPGA. La realización del trabajo de investigación permitirá proporcionar beneficios desde el punto de vista académico y de investigación, tanto para el área de sistemas digitales, como para el DEEL y líneas de investigación asociadas, para que en futuros trabajos exista la posibilidad de diseñar e implementar procesadores personalizados para sistemas embebidos o incluso llegar a fabricar un chip propio en la Universidad de las Fuerzas Armadas ESPE.

Alcance del Proyecto

Para la realización del proyecto es necesario conocer a profundidad la arquitectura RISC-V. Por tal motivo es necesaria la recopilación de información: conocer su origen, especificaciones, núcleos que manejan la arquitectura, etc. Posteriormente, del universo de implementaciones de la arquitectura RISC-V, se elegirá dos para su estudio y análisis a través de la búsqueda de artículos científicos relacionados. Es importante conocer los procesadores a utilizar en este proyecto de titulación ya que cada uno de ellos dispone de una implementación lo cual involucra el uso de hardware y software dependiendo del diseño elegido. Un punto importante a considerar es la disponibilidad de una tarjeta de desarrollo basada en FPGA para la implementación de los procesadores objeto de prueba. En el caso del presente proyecto se dispone de la tarjeta Nexys A7-100T de la familia Xilinx Artix-7. Los procesadores a implementar deberán ser parte del repertorio RISC-V existente. Dentro de la plataforma GitHub se pueden encontrar algunos procesadores y Sistemas en Chip (SoC) que se pueden desarrollar. Como este proyecto persigue el objetivo de comenzar a introducir la arquitectura RISC-V, se desea seleccionar dos procesadores básicos de los cuales se disponga de la suficiente información. Una vez entendidas las bases teóricas y el funcionamiento de cada procesador, se procederá a su implementación en la FPGA. Con respecto al software, dependiendo de los procesadores seleccionados se manejará el lenguaje de programación principal el cual podría ser ensamblador, C, C++, Phyton entre otros, y como lenguajes para descripción de hardware se podría utilizar VHDL, SystemVerilog, Chisel, BlueSpec, SpinalHDL entre otras opciones más. Con respecto a la implementación, dentro de la plataforma GitHub se puede encontrar métodos de desarrollo de hardware y software que permiten construir los núcleos a través de herramientas como RISC-V tools, GNU Compiler Collection, makefile y algún software para depuración. Algunos también hacen uso del entorno de desarrollo Vivado de Xilinx para poder generar la síntesis e implementación de la programación sobre la FPGA y del kit de desarrollo de Software (SDK) para el desarrollo de aplicaciones integradas. Una vez

que los procesadores se encuentren funcionando sobre la FPGA Nexys A7, se realizarán comparaciones de desempeño y funcionamiento con la ayuda de un benchmark, para finalmente analizar los resultados obtenidos.

Por lo tanto, una vez definidos todos los puntos a tratar en el trabajo de investigación, en el siguiente capítulo se explican de manera resumida toda la teoría relevante sobre RISC-V y las herramientas de hardware y software necesarias para la implementación y medición de los núcleos.

Capítulo II

En el presente capítulo se realizará una breve descripción de los inicios de RISC hasta llegar a RISC-V, una introducción a otras ISAs importantes que han llegado a ser exitosas y que continúan funcionando al día de hoy, y herramientas de hardware y software que se utilizarán para el desarrollo de los núcleos y medición de los mismos.

Estado del Arte

Los procesadores

Antes que la tecnología tuviera un impacto visible en la sociedad, solo existían pocas personas capaces de poder manipular los primeros ordenadores que aparecieron en aquella época. Algunas de estas máquinas llegaban a ocupar cuartos enteros y debido a sus dimensiones, debían de ser manejadas por más de una persona. Luego, con los hechos sucedidos de la era industrial y la segunda guerra mundial, surge la arquitectura de Von Neumann como respuesta a la necesidad de las grandes potencias para poder resguardar toda la información que poseían. Los primeros ordenadores que salieron hacían uso del sistema binario, es decir, solamente con dos dígitos (0 y 1) construían la información que entraba al ordenador (VidaBytes, 2020). Es aquí donde aparece el procesador, el cual se ocuparía de admitir los pasos de los algoritmos en orden y ejecutarlos. Visto de otra manera, su función primordial sería la de actuar como el cerebro del sistema, ayudando al procesamiento de todo lo que ocurre dentro del computador y ejecutar las acciones que estén disponibles. Mientras más rápido era este, se ejecutaban de manera más veloz las ordenes que se le indicaba a la máquina.

Al igual que los demás componentes del computador, el procesador es uno de los que más ha evolucionado, permitiendo que las máquinas funcionen de manera más rápida y eficaz.

Es el componente más importante y más costoso, pero necesita de otras partes para poder servir y actuar (Etecé, 2021).

Arquitectura del conjunto de instrucciones (ISA)

Una ISA se refiere al conjunto de instrucciones que son programadas en un procesador y que son ejecutadas en el mismo (Morales, 2019).

Dentro del conjunto de instrucciones, se especifica las funcionalidades del procesador, es decir, de manera general se describen:

- Operaciones que soporta
- Mecanismos de almacenamiento y como son accesados.
- Como el programador comunica los programas al procesador.

Entre los principios básicos que los diseñadores deben de tomar en cuenta al momento de diseñar una ISA se encuentran (Waterman & Patterson, GUÍA PRÁCTICA DE RISC-V El Atlas de una arquitectura abierta, 2018):

- Costos
- Simplicidad
- Rendimiento
- Escalabilidad
- Aislamiento de arquitectura e implementación
- Dimensión del programa.
- Disposición para programar, compilar y linkear.

En resumen, desde la aparición de los primeros computadores han ido aparecido otras ISAs, algunas de ellas siguen funcionando hasta la actualidad y otras que han quedado en desuso pero que han servido como bases para crear otras ISAs. Conozcamos algunas de ellas en la siguiente sección.

Diferentes ISAs existentes

MIPS

Es una arquitectura de procesadores de tipo RISC (Hennessy & Patterson, 1993). Su origen se da en la universidad de Stanford, aproximadamente por el año de 1981. John Hennesy y su equipo se encontraban trabajando en lo que posteriormente será conocido como el primer procesador MIPS. Uno de los objetivos planteados para esta arquitectura fue el de mejorar el rendimiento a través del proceso de segmentación. Otro de los objetivos planeados fue el de que todas las tareas de las instrucciones se demoren un solo ciclo en completarse. En 1984, John Hennesy abandona Stanford y funda MIPS Computer Systems y comienza la creación de varios diseños: R2000, R3000, R4000, entre otros más. En 1999 se consolidad el sistema de licencias de MIPS y se lanzan MIPS32 y MIPS64, lo cual resultó ser un gran éxito en ventas y que desde ese entonces ha continuado mejorando hasta la actualidad, llegando a ser parte de dispositivos como ordenadores, decodificadores e incluso sintonizadores de televisión. (Hernández, Tejedor, & Iglesias, 2010)

SPARC

Es una arquitectura del tipo RISC, diseñada por Sun Microsystems. Se basó en diseños creados por la Universidad de California, RISC I y II. SPARC es una arquitectura abierta, que permite que otros desarrolladores creen sus propios diseños. Fue licenciada por Sun Microsystems para otras empresas como Texas Intruments, Fujitsu, entre otras más. Uno de los objetivos que se consiguió con la arquitectura fue el de la ventana de registros, la cual

permitió hacer compiladores de alto rendimiento y adicionalmente redujo de manera significativa la menoría en instrucciones como load/store. (Varona , López, & Martínez, 2015)

ARM

ARM inició como un proyecto de desarrollo de la empresa Acorn Computers en 1983. Dos años más tarde, Steve Furber y Roger Wilson desarrollarían el primer chip denominado ARM1 (Pastor, 2015). Más adelante, en el año 1991, después de haber trabajado en conjunto Acorn con Apple, crean un nuevo núcleo denominado ARM6. No fue después del desarrollo del núcleo ARM8 cuando se comenzó a utilizarse en otros aparatos como GPS, calculadoras o móviles. En la actualidad, existen una gran cantidad de dispositivos que hacen uso de la familia Cortex de ARM. Alrededor del 75% de procesadores que sean de 32 bits, utilizan este chip en su núcleo (Nuel, 2012). Si bien, ARM se encarga del diseño de sus arquitecturas, también vende la propiedad intelectual para que otras compañías puedan crear sus diseños, modificarlos en el caso de ser necesario para posteriormente realizar el proceso de fabricación final del procesador.

OpenRISC

El proyecto OpenRISC fue creado con la intención de permitir la creación de procesadores de código abierto para sistemas embebidos. En resumen, se buscaba lograr:

- Una ISA RISC libre y abierta con características DSP.
- Un conjunto de herramientas de desarrollo de software, aplicaciones gratuitas, bibliotecas, sistemas operativos de código abierto.
- Una variedad de simuladores para SoCs y sistemas.

El lenguaje de descripción de hardware utilizado es verilog. Ha sido utilizado para fabricar circuitos integrados ASIC e implementado en entornos de FPGA. Su uso estaba enfocado para implementaciones académicas, investigación e industriales (OpenRISC, 2021).

80x86

La arquitectura 80x86 en las últimas décadas se ha convertido en el conjunto de instrucciones más popular en el mercado. Las razones principales a las que se deben su popularidad han sido su tecnología de fabricación de vanguardia, sus implementaciones de microarquitectura, su disponibilidad y su enfoque a la compatibilidad (Waterman, Design of the RISC-V Instruction Set Architecture, 2016). Desarrollada por INTEL en base a los microprocesadores 8086 y 8088 a finales de los años 70. Comenzó como una CPU de memoria segmentada que permitiría procesar información de 16 bits o más. Desde que la arquitectura fue introducida al mercado, llego a ser el modelo para la construcción de otros CPUs ya que permitía crear conjuntos de instrucciones más amplios y complejos. La arquitectura x86 se basa en el proceso CISC (complex instruction set computer) a diferencia de ARM el cual se basa en SISC (Simple Instruction Set Computing) (Campos, 2021).

Antecedentes de RISC

Alan Turing en 1946 diseño un dispositivo con muchas características parecidas a las de una arquitectura RISC (Doran, 2005). Desde el año de 1960, varios sistemas basados en el enfoque de carga/almacenamiento han sido considerados como arquitecturas RISC. Michael Flynn ve al IBM 801 como el primer sistema RISC, en el cual se trató de construir un procesador que serviría de base para un conmutador telefónico digital. En 1975 el programa fue cancelado pero se activaría más tarde con una versión mejorada llamada IBM ROMP (Cocke & Markstein, 1990). El 801 sirvió como inspiración para el desarrollo de otros proyectos de investigación y se volvió como el más conocido en la década de 1970. Aparecieron chips más complejos que trataron de expandir el conjunto de instrucciones a través del uso de microcódigo (Starnes, 1983). Más tarde aparecería Dave Patterson con la creación del programa Berkeley RISC y se producirían mejoras para que los programas se ejecuten más rápido a través de la variación del microcódigo y posibles mejoras de rendimiento a través del

uso de canalizaciones y ventanas de registro (Patterson, 1980). En 1982 el programa Berkeley implementó el procesador RISC-I, el cual, a pesar de solo tener 44,420 transistores y 32 instrucciones, superó completamente a otros diseños de chips. En 1983 sale RISC-II con 40,760 transistores, 39 instrucciones pero con una velocidad triple que su antecesor (Sequin & Patterson, 1981). Cuando Silicon Valley llegó a conocer el proyecto RISC, se comenzó a desarrollar un proyecto similar en la Universidad de Stanford denominado MIPS, creado por John Hennessy el cual más tarde crearía la empresa MIPS Computer Systems que ofrecería una nueva arquitectura (Nurmi, 2007). En la década de 1980, alrededor del concepto RISC aparecieron muchas incertidumbres, pero en 1987 la empresa Sun Microsystems con ayuda de RISC II, lanza el procesador SPARC, el cual tuvo éxito y aceptación y despertó nuevamente el interés de IBM. El trabajo de los diseños originales terminó en RISC II, concepto diseñado por Berkeley. Más tarde, en el año 2010, saldría una nueva arquitectura denominada RISC-V con un conjunto de instrucciones abierto diseñado con fines investigativos y alternativas a ISAs patentadas (Waterman, Lee, Patterson, & Asanovi, 2014).

RISC-V es una nueva arquitectura de conjunto de instrucciones (ISA) desarrollado por la Universidad de California de Berkeley en el año 2010. Fue diseñada para apoyar a investigaciones de arquitectura de computadores y con fines educativos. El estándar llego a ser libre, con una arquitectura abierta para aplicaciones en la industria (Waterman & Asanovic, 2017).

El desarrollo de una nueva arquitectura de conjunto de instrucciones tenía por objetivo facilitar la extensibilidad y crear instrucciones que sean de código abierto. Así nació RISC-V, la cual incluía variantes para 32,64 y 128 bits y adicionalmente admitía extensiones opcionales brindando flexibilidad en el caso de ser necesario para alguna aplicación específica (Monteiro, 2019).

Proyecciones futuras de RISC-V

Desde el momento de su creación, RISC-V se planteó los siguientes planes a futuro:

- Llegar a ser una ISA completamente abierta para usos académicos e industriales que busca convertirse en universal.
- Llegar a ser una ISA adecuada para su implementación directa en hardware, es decir, debe acoplarse a procesadores, microcontroladores, incluso supercomputadoras. También deberá poder implementarse en otros tipos de tecnologías como FPGAs, ASICs, chips, etc.
- Debe de ser eficiente y evitar la sobre-arquitectura para una microarquitectura en particular.
- La ISA base debe ser estable y no descontinuarse.
- Una ISA que permita extensiones a nivel de usuario y variantes especializadas, que permita variantes de 32 y 64 bits para aplicaciones, núcleos de sistemas operativos e implementaciones en hardware.

Marco Teórico

Especificaciones de RISC-V

RISC-V es una ISA abierta de años recientes, que ha sido creada tomando en cuenta los errores de ISAs anteriores. Como se mencionó en los objetivos que desea alcanzar la arquitectura, la ISA debe ser eficaz para todos dispositivos, para lo cual se enfatiza en la simplicidad para continuar con costos reducidos, muchos registros y una velocidad de ejecución que ayude a los usuarios a resolver problemas de manera eficaz (Patterson & Waterman, 2018).

Dentro del manual de RISC-V versión 2.2 se describe en la tabla 1 las versiones de los módulos de la ISA RISC-V:

Tabla 1

Especificaciones congeladas y no congeladas de RISC-V Foundation

Base	**Versión**
RV32I	2.0
RV32E	1.9
RV64I	2.0
RV128I	1.7
Extensión	**Versión**
M	2.0
A	2.0
F	2.0
D	2.0
Q	2.0
L	0.0
C	2.0
B	0.0
J	0.0
T	0.0
P	0.1
V	0.2
N	1.1

Como se describe en la tabla 1, la notación RV de la base indica que se trata de un conjunto de instrucción de RISC-V, el cual dependiendo de la arquitectura puede ser de 32, 64 o 128 bits. Las extensiones, las cual se añaden a la base, permiten agregar más operaciones. A continuación se describen algunas de ellas:

Extensión I

La especificación comprende operaciones con números enteros, operaciones de memoria y saltos condicionales. En la tabla 1 se observó dos variantes para implementación, 32 y 64 bits. Según la variante escogida va a depender el ancho de los registros.

Extensión E

La especificación comprende operaciones con números enteros, fue reducida a la extensión I para sistemas embebidos de tamaño pequeño, por lo que se redujo el número de registros que posee.

Extensión M

La especificación incluye instrucciones para realizar la operación de multiplicación o división entre números con signo y sin signo. Esta especificación ayuda a comprender el concepto de arquitectura modular.

Extensión A

La especificación comprende a instrucciones atómicas, las cuales pueden ser útiles cuando se usa hilos sobre la unidad de procesamiento. Estas instrucciones se ejecutan antes de ejecutar las nuevas instrucciones.

Extensión F y D

Las especificaciones se refieren a operaciones con punto flotante. De manera general, suelen ser implementadas de manera conjunta para dar soporte a cualquier tipo de operación con punto flotante.

Extensión C

La especificación reduce el tamaño del código estático y dinámico al agregar codificaciones de instrucciones cortas de 16 bits para operaciones comunes. La extensión C se puede agregar a cualquiera de las ISA base (RV32, RV64, RV128) (Rocha, 2019).

Extensión Zicsr

La especificación se refiere al acceso de los registros de control y estado haciendo uso de instrucciones CSR (Control Status Registers).

Lenguaje ensamblador de RISC-V

RISC-V hace uso de instrucciones para programar al procesador. Entre los tipos de instrucciones más básicos usados se encuentran los computacionales, operaciones de memoria y saltos. Las instrucciones utilizan operandos los cuales pueden ser ubicados en registros, en la memoria o son decodificados como constantes.

Tabla 2

Instrucciones de lenguaje ensamblador de RISC-V más típicas

Lenguaje ensamblador	
Instrucciones	**Operación**
Computacionales	Aritméticas
	Lógicas
	Desplazamiento
Memoria	Carga y guardado
Branch	Saltos
Pseudoinstrucciones	Instrucciones compuestas

Entre las instrucciones de ensamblador computacionales más típicas que se describen en la tabla 2, se encuentran operaciones aritméticas (suma, resta, multiplicación, división), operaciones lógicas (AND, OR, XOR) y las de desplazamiento (SHIFT).

Entre las instrucciones de ensamblador de memoria, las más típicas son cargar (LOAD) y guardar (STORE).

Para las instrucciones de salto (BRANCH), se utiliza condicionales.

También existen las pseudoinstrucciones (MOVE, NOT, NOP), las cuales son implementadas con una o más instrucciones de RISC-V y que son más comúnmente utilizadas por los programadores.

Tabla 3

Algunos de los registros de RISC-V

Nombre	Uso
zero	Valor constante 0
ra	Dirección de retorno
sp	Puntero de pila
s1	Registro de almacenamiento
t0-2	Variables temporales

Con respecto a los registros mostrados en la tabla 3, ayudan a retener información según su tipo. Se pueden encontrar para almacenar datos, registros temporales para guardar datos de cálculos del tiempo, registros para guardar direcciones, entre otros más. Dentro de los registros más usado por los programadores se encuentra el zero (x0), el cual siempre posee el valor de 0.

Tabla 4

Algunas de las principales directivas utilizadas en ensamblador

Directiva	Descripción
.text	La información será almacenada en la sección de texto
.data	La información será almacenada en la sección de datos
.bss	La información será almacenada en la sección bss
.end	El ensamblador concluirá el código una vez alcance esta directiva.

Por último, se describe las directivas de la tabla 4, las cuales le dicen al ensamblador en donde ubicar el código y los datos. Estas también especifican las constantes de texto y datos que serán usadas en el programa. Estas directivas comienzan con un punto y seguido viene su nombre (.text, .data, .bss, .end).

Descripción de las herramientas de hardware y software

A continuación, se describe la tarjeta FPGA que se utilizó para la implementación de los dos núcleos y adicionalmente otras herramientas de software que fueron necesarias para poder programar, visualizar a través del monitor serie y realizar las mediciones de ambos núcleos:

FPGA Nexys A7

La tarjeta que se utilizó para la implementación es la Nexys A7, antes conocida como Nexys 4 DDR. Esta FPGA tiene un precio accesible y es una opción potente. Es una plataforma de desarrollo perteneciente a la familia de Xilinx Artix-7 que posee varias interfaces como 10/100 Ethernet, UART, USB, JTAG y VGA. Adicionalmente cuenta con un sensor de temperatura, micrófono, acelerómetro y 5 puertos Pmod para aplicaciones adicionales. Nexys A7 es compatible con la suite Vivado de Xilinx de la edición WebPack (Digilent, 2021).

Existen dos variantes de la tarjeta Nexys A7 las cuales se presentan en la tabla 5:

Tabla 5

Descripción de las características de las tarjetas Nexys A7-50T y Nexys A7-100T

	Nexys A7-50T	**Nexys A7-100T**
Circuito integrado	XC7A50T-1CSG324C	XC7A100T-1CSG324C
Logic Slices	8,150	15,850
Bloques RAM (Kbits)	2,700	4,860
Memoria DDR2 (MiB)	128	128
Clock Tiles (Con PLL)	5	6
DSP Slices	120	240

Nota. La tabla ha sido tomada de la página oficial de Digilent donde se muestran las características de ambas versiones de la FPGA.

Vivado 2019.2

Vivado Design Suite es una herramienta para el desarrollo de SoC. Cuenta con un entorno de desarrollo y herramientas que han contribuido a hacer frente a problemas que se puedan presentar en la productividad de sistemas digitales referente a integración e implementación. Vivado es compatible con varias familias de dispositivos como: UltraScale, Virtex-7, Kintex-7, Artix-7 y Zynq-7000 (Xilinx, 2019).

Vivado será el software que se utilizará para realizar la síntesis y la implementación de los núcleos. Adicionalmente, esta potente herramienta nos permitirá obtener algunos parámetros de utilización de la FPGA que servirán posteriormente para poder realizar la comparación de los núcleos.

Visual Studio Code

Visual Studio Code es un editor de código fuente disponible para Windows, Linux y MacOS. Posee soportes incorporados para Javascript, Typescript y Node.js. Adicionalmente

posee extensiones para otros lenguajes como Java, Python, PHP, C, C++, entre otros más y tiempos de ejecución como .NET y Unity (Studio, 2021).

PlatformIO

Es una IDE (entorno de desarrollo integrado) que trabaja en conjunto con Visual Studio Code para acelerar y simplificar la creación y entrega de dispositivos embebidos. Entre algunas de sus principales características, podemos encontrar que es un editor ligero multiplataforma que permite una fácil organización del flujo de trabajo con múltiples proyectos, además de facilitar la navegación por el código, posee una integración con PlatformIO Home lo cual introduce diferentes placas y bibliotecas, un terminal integrado con PlatformIO Core CLI, una variedad de proyectos ejemplo y un monitor de puerto serie (Labs, 2021).

Cutecom

Cutecom es un terminal serie gráfico que permite la comunicación con otros dispositivos, es de software libre y distribuido bajo GNU. Funciona en sistemas operativos Linux, FreeBSD y Mac OS X. Posee una interfaz fácil de usar, con diferentes secuencias de control, compatibilidad con xmodem, ymodem y zmodem, permite diferenciar entre el texto escrito y el eco, selección de modos solo lectura, solo escritura o lectura/escritura, entrada y salida hexadecimal, caracteres finales de línea configurables y retraso configurable entre caracteres (Neundorf, 2009).

Benchmark Coremark

Es un comparador de rendimiento diseñado específicamente para probar la funcionalidad de microcontroladores y unidades centrales de procesamiento utilizados en sistemas integrados. Su ejecución permite obtener puntuaciones de un solo número para

realizar comparaciones rápidas entre procesadores. Es un remplazo más actual del benchmark Dhrystone que posee implementaciones de algoritmos como:

- Procesamiento de listas.
- Manipulación de matrices.
- Máquinas de estado.
- CRC.

Está diseñado para ejecutarse en dispositivos de 8 bits hasta 64 bits (EEMBC, 2009).

Dentro de los parámetros que Coremark muestra una vez se ha efectuado la medición, se describen los siguientes:

- CoreMark/MHz: Es una medida de rendimiento de un solo subproceso por frecuencia de reloj.
- Iteraciones: Controla cuantas veces se ejecuta el kernel Coremark.
- TOTAL_DATA_SIZE: Define el tamaño total para los algoritmos de datos.
- CORE_TICKS: Define el tiempo de retorno de las funciones de temporización.
- TOTAL TICKS: Numero de ciclos requeridos para ejecutar el benchmark.
- Coremark Size: De manera predeterminada, Coremark asigna solamente 2 Kbytes para acomodar los datos.

RISC-V GNU Toolchain

El Toolchain, o también conocido como cadena de herramientas de software, que permite la creación de instrucciones y secuencias de ensamblaje que pueden ser ejecutadas en un simulador o también en una Fpga objetivo (MindChasers, 2021).

Un toolchain a través de un conjunto de herramientas, compila el código fuente en ejecutables para que pueda reproducirse en otro dispositivo destino. Se incluye un compilador,

un enlazador y varias librerías. Para el toolchain del compilador GNU de RISC-V se encuentra RISC-V C y el compilador cruzado C++ usado para el desarrollo de sistemas embebidos. (Simmonds, 2015)

Ahora que ya se dio a conocer un poco más sobre la ISA de RISC-V y adicionalmente todas las herramientas que han se han utilizado para el desarrollo del trabajo de titulación, tanto de hardware y software, es el momento de conocer que núcleos pueden ser implementados en la FPGA disponible, por lo que en el siguiente capítulo conoceremos algunos procesadores que han sido construidos haciendo uso de una tarjeta FPGA y se elegirán dos que sean compatibles y que puedan funcionar de manera correcta en la tarjeta objetivo.

Capítulo III

Selección de procesadores RISC-V

En el presente capítulo se revisarán algunas implementaciones de núcleos que hacen uso de una FPGA. Existen varios ejemplos desarrollados por la comunidad de GitHub, por lo que se elegirán dos de ellos para conocerlos más a profundidad y se describen sus características más relevantes antes de pasar a la implementación de los mismos.

Implementaciones en FPGAs

Existen varias implementaciones de núcleos RISC-V dentro de la plataforma Github. Actualmente la página que contenía todos los núcleos y SoCs ha sido archivada por la empresa RISC-V, por lo que, si se desea añadir algún núcleo de creación propia o añadir alguna modificación a los núcleos existentes, se debe de poner en contacto con la empresa RISC-V. Pasando nuevamente a la plataforma Github, dentro del ámbito de núcleos que hacen uso el set de instrucciones RISC-V, se puede encontrar de manera totalmente gratuita implementaciones de núcleos, SoCs y plataformas para SoCs.

Con el pasar del tiempo, han ido apareciendo nuevas creaciones basados en la ISA de RISC-V. Debido a que el trabajo de investigación se enfoca en la implementación de núcleos que hagan uso exclusivamente de una FPGA, se presentan a continuación algunos procesadores que han sido implementados haciendo uso de una tarjeta:

DarkRISCV core

DarkRISCV es un pequeño procesador el cual fue creado en un solo día. La idea inicial del proyecto comenzó como una prueba del conjunto de instrucciones RISC-V. A pesar de ser pequeño, posee algunas características destacables (Samsoniuk, 2018):

- Implementa la mayoría del conjunto de instrucciones RV32E y RV32I
- Puede soportar 1 reloj por instrucción la mayor parte del tiempo

- Utiliza una arquitectura Harvard muy flexible
- Funciona bien en Fpgas de Xilinx, altera y lattice
- Utiliza entre 850 a 1500 LUTS
- Posee una licencia del tipo BSD, lo cual implica que se lo puede usar sin restricciones.

PicoRV32

Es un núcleo de hardware libre y abierto licenciado bajo ISC (Internet Systems Consortium). Permite implementar el conjunto de instrucciones RV32IMC, es decir, puede ser configurado como RV32E, RV32I, RV32IC, RV32IM o RV32IMC. La CPU fue diseñada para ser utilizada como procesador auxiliar para diseños de FPGAs o ASICs. Entre algunas de sus características se encuentran (Wolf, 2015):

- Es pequeño, ocupa entre 750 a 2000 LUTs en arquitecturas Xilinx de la serie 7.
- Para FPGAs de Xilinx de la serie 7, la frecuencia máxima varía entre 250 a 450 MHz.
- La interfaz de memoria puede ser nativa seleccionable o maestro AXI4-Lite.
- Compatibilidad opcional con IRQ.
- Interfaz de coprocesador opcional

NeoRV32

Es un procesador de código abierto compatible con RISC-V el cual puede ser usado como un procesador auxiliar dentro de un sistema mucho más grande o como un microcontrolador independiente. El procesador puede ser configurado de varias maneras y adicionalmente permite la adición de periféricos adicionales como memorias integradas, puertos de E/S, interfaces serie, temporizadores entre otros más. Algunas de las características destacables de núcleo son (Nolting, 2022):

- Está escrito en lenguaje VHDL.

- De código abierto y compatibilidad con RISC-V.
- Amplio conjunto de opciones para personalizar con diferentes extensiones ISA. Las diferentes extensiones que se pueden configurar son: A, B, C, E, M, U, Zfinx, Zicsr, Zicntr, Zihpm, Zifencei, Zmmul, Zxcfu.

SwerVolf

SwerVolf es una plataforma de referencia basada en FuseSoc para la familia SweRV de núcleos RISC-V. Actualmente existen algunas variaciones: SweRV EH1, SweRV EL2 y SwerRV EH2. Basado en SweRV EH1, se ha creado una adaptación que permite la implementación en FPGAs llamada RVfpga. El uso del núcleo se centra en la portabilidad, extensibilidad y facilidad de uso, permitiendo a los usuarios ejecutar software, realizar modificaciones según las necesidades o portarlo a nuevos dispositivos. Algunas de sus características son las siguientes (Digital, 2022):

- De código abierto creado por Western Digital.
- Núcleo superescalar de 32 bits (RV32ICM) con canalización de 9 etapas.
- Su versión compleja tiene separada la memoria de instrucciones y datos (ICCM, DCCM).
- Controlador de interrupciones programable.
- Sistema Bus: AXI4 o AHB-Lite

PULPino

PULPino es un núcleo de código abierto basado en los núcleos RISC-V de 32 bits implementados en ETH Zurich. Permite ser configurado ya sea como núcleo RISCY o zero-riscy. Entre sus principales características se encuentran ((Platform, 2022)):

- En el caso de ser configurado como RISCY, cuenta con 4 fases de canalización y un IPC cercano a 1. Soporta el set de instrucciones RV32I, RV32C y RV32M.
- En el caso de ser configurado como zero-riscy, el núcleo contaría con dos fases de canalización y los conjuntos de instrucciones soportados serían RV32I, RV32C, RV32M y RV32E. Esta versión del núcleo fue diseñada para aplicaciones con potencia ultra baja.

Rocket

Rocket es un procesador escalar con cinco fases de canalización. Está formado por una ALU y un FPU opcional. Fue desarrollado en UC de Berkeley y actualmente SiFive continua dándole soporte. El procesador generalmente es utilizado como un componente dentro del generador SoC de Rocket Chip. El núcleo soporta las instrucciones RV64GC de RISC-V y está escrito en lenguaje Chisel. Tiene compatiblidad opcional con las extensiones M, F, D, A. Originalmente Rocket Chip fue diseñado para FPGAs Zynq y Virtex (Research, 2019).

VexRISCv

VexRiscv es un procesador amigable que hace uso del set de instrucciones de RISC-V y posee las siguientes características (SpinalHDL, 2022):

- Set de instrucciones RV32IM.
- Canalizado de 5 fases.
- Extensión MUL/DIV opcional.
- Caches opcionales para instrucciones y datos.
- Está escrito en lenguaje SpinalHDL.
- Esta optimizado para FPGAs Cyclone, ICE40.

Existen varios ejemplos más de procesadores que han sido implementados con la ayuda de alguna tarjeta FPGA, pero, para el desarrollo del trabajo de titulación se escogerán los núcleos NeoRV32 y la adaptación del SweRV EH1 que funciona en la tarjeta objetivo. Para la implementación de ambos núcleos, se construirán SoCs de ambos debido a que más adelante se necesitará de la interfaz UART para el envío del programa benchmark para realizar las mediciones correspondientes a cada core. Debido a que se dispone de una FPGA Nexys A7-100T para la implementación de ambos núcleos, uno de los criterios para la selección de los núcleos es el espacio que ocupará en la tarjeta. La FPGA dispone de 128 MB de RAM lo cual es lo suficientemente grande y es ideal para los procesadores. Otros criterios que son considerados para la selección de los núcleos es que ambos sean de código abierto, que tengan soporte para la tarjeta disponible y adicionalmente que posean una amplia cantidad de información sobre el proceso y herramientas necesarias para su implementación, lo cual ambos cumplen. A continuación, se realizará una presentación más completa de los dos núcleos elegidos respectivamente:

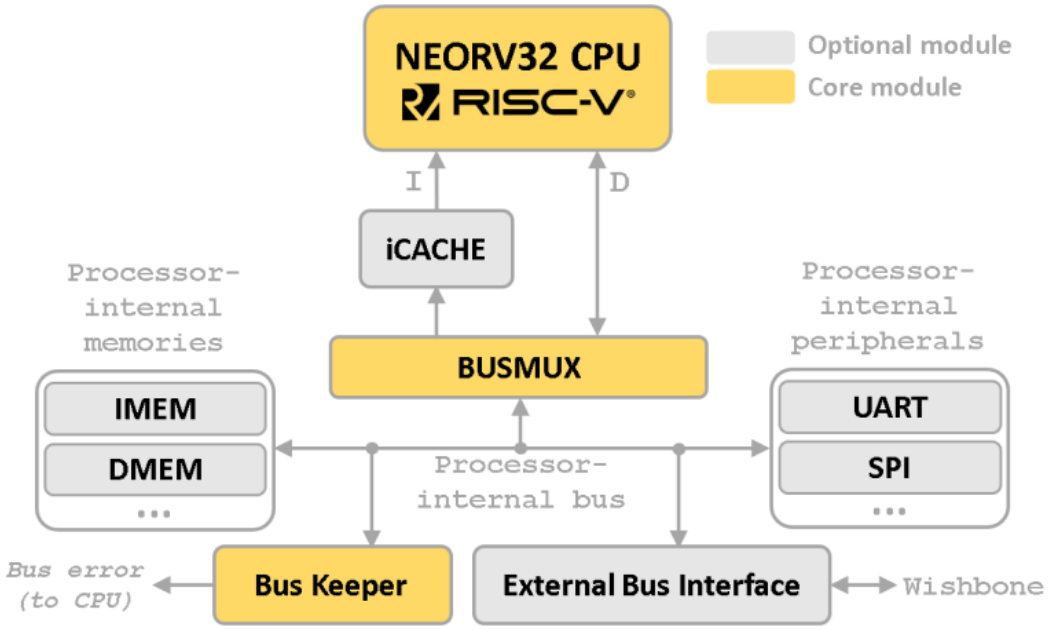
Procesador NeoRV32

El procesador permite la construcción de un SoC con todas las funciones en base al CPU NEORV32. Permite cambiar el diseño ya que es altamente configurable y además es flexible para adaptar su configuración dependiendo de las necesidades. De manera general está formado por:

- CPU
- Interconexiones (AXI4-Lite, WB b4)
- Periféricos (BOOTLDROM,GPIO, UART, SPI, memorias)

Figura 1

Arquitectura interna del procesador

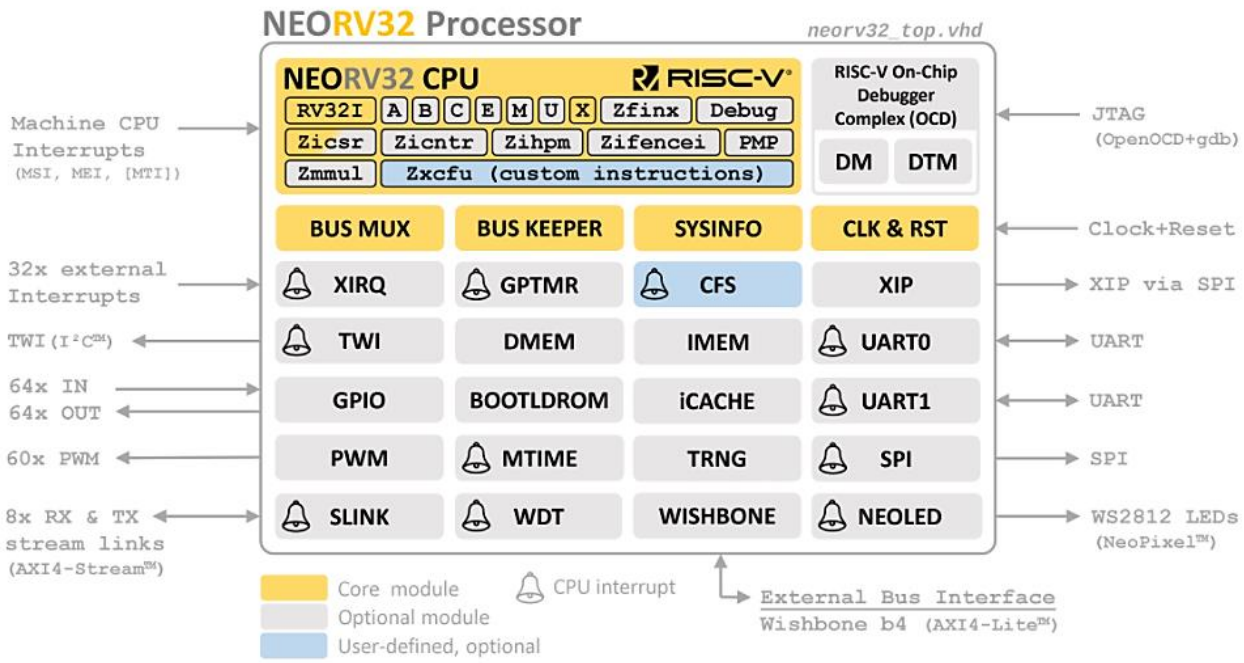


Nota. La figura muestra la arquitectura de bus interna del procesador. Tomado del Datasheet del procesador NEORV32, versión 1.7.2.

El procesador mostrado en la figura 2, está basado en el CPU NEORV32. En conjunto con interfaces periféricas comunes y memorias integradas, proporciona una plataforma SoC a escala completa.

Figura 2

Descripción gráfica del procesador NEORV32



Nota. La figura muestra al CPU NEORV32 junto con interfaces de periféricos comunes y memorias. Tomado del Datasheet del procesador NEORV32, versión 1.7.2.

Entre sus características principales se anuncian las siguientes:

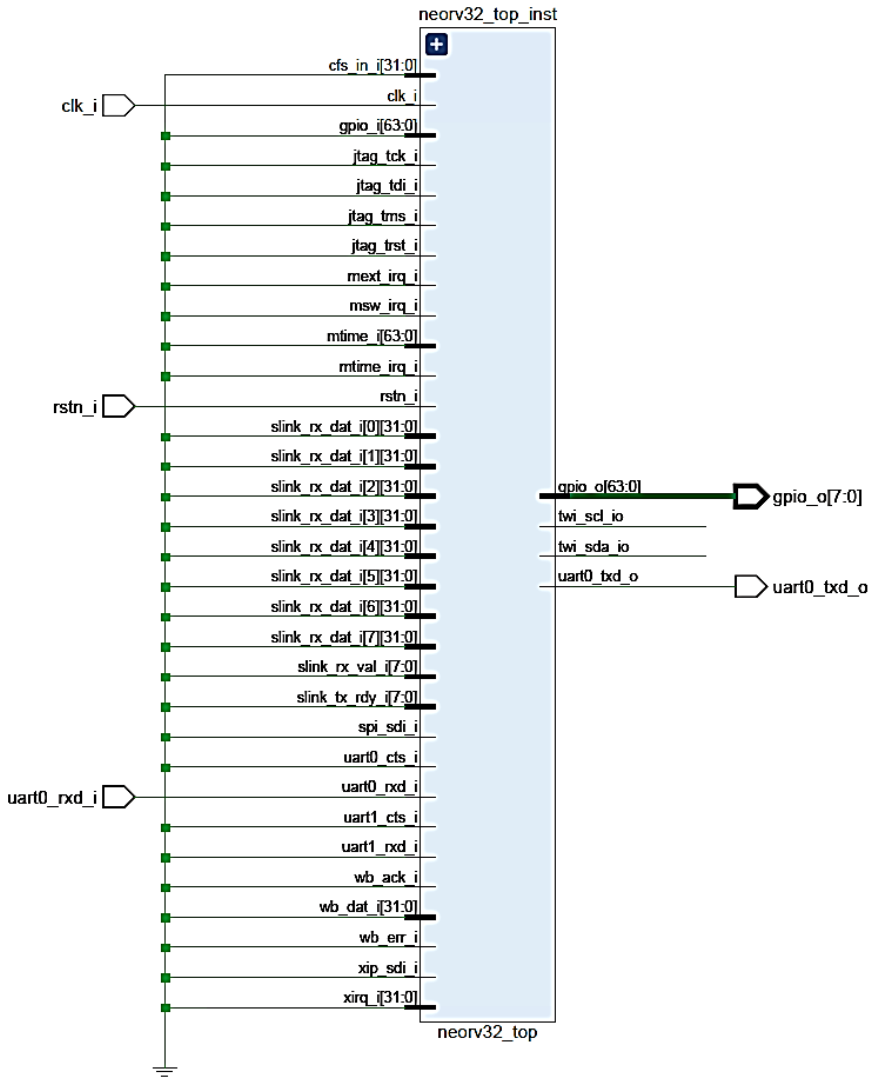
- Posee memorias de instrucciones y datos internos opcionales (DMEM/IMEM).
- Gestor de arranque interno (BOOTROM) opcional con UART y opción de arranque flash SPI.
- Temporizador opcional del sistema (MTIME) compatible con RISC-V.
- Dos receptores y transmisores universales opcionales (UART0, UART1) con control de flujo opcional (RTS/CTS) y FIFO.
- Controlador de interfaz periférica serie (SPI) opcional de 8/16/24/32 bits.
- Controlador de interfaz serie (TWI) opcional compatible con I2C.
- Puertos opcionales de E/S de uso general (GPIO).

- Interfaz de bus externo opcional de 32 bits.
- Controlador PWM con hasta 60 canales.
- Interfaz de LED inteligente compatible con NeoPixel (NEOLED).
- Controlador de interrupciones externo (XIRQ).
- Temporizador opcional de uso general de 32 bits (GPTMR).
- Depurador opcional en chip con JTAG TAP (OCD).
- Sistema de configuración de memoria de información para chequear el Hardware mediante Software (SYSINFO).

En la figura 3 se puede observar el esquemático de la entidad TOP del núcleo obtenido de Vivado:

Figura 3

Esquemático de la entidad neorv32_top



Una descripción más completa de los módulos y señales se encuentra en el apéndice F.

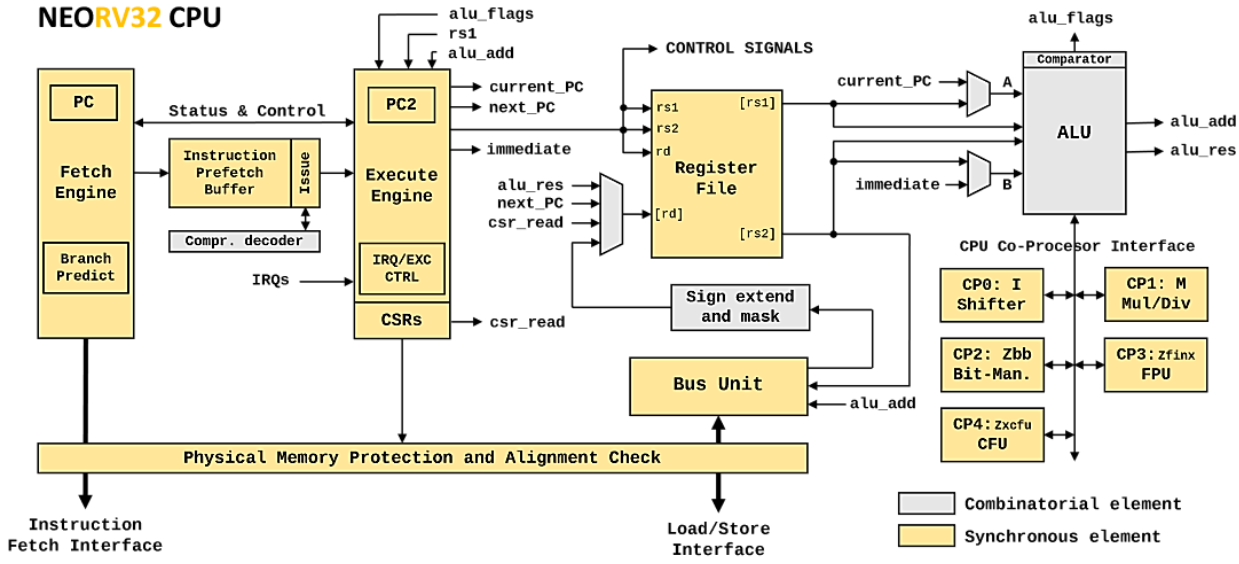
Arquitectura.

El CPU utiliza una arquitectura multiciclo, lo cual quiere decir que cada instrucción se ejecuta como una serie de microoperaciones consecutivas. Para explicar la figura 4 se describen dos interfaces; el Front-end y el Back-end.

El Front-end es el responsable de capturar instrucciones de 32 bits. Los datos de instrucción son almacenados en una cola FIFO.

Figura 4

Arquitectura Multiciclo del CPU



Nota. La figura muestra la arquitectura multiciclo del CPU NEORV32. Tomado del Datasheet del procesador NEORV32, versión 1.7.2.

El Back-end es el responsable de la ejecución de las instrucciones. Aquí se incluye lo que se conoce como “issue engine”, el cual se encarga de tomar datos que han sido capturados previamente por un buffer y ensambla palabras de 32 bits para que sean ejecutadas.

El Front-end y el Back-end operan en paralelo, por lo que el CPI óptimo del procesador es de 2. Como una máquina de Von Neumann, el CPU posee interfaces independientes para la captura de instrucciones y el acceso de datos. Estas dos interfaces se fusionan en un solo bus interno el cual funciona con un conmutador de priorización. Por lo tanto, todas las ubicaciones de la memoria incluyendo los dispositivos periféricos son mapeados a un único espacio de dirección unificado.

Procesador SweRV EH1

El procesador con el cual se va a trabajar está basado la versión compleja del núcleo SweRV EH1 que es el CPU. Los módulos que serán utilizados para la implementación del núcleo pueden ser clasificados en tres bloques grandes:

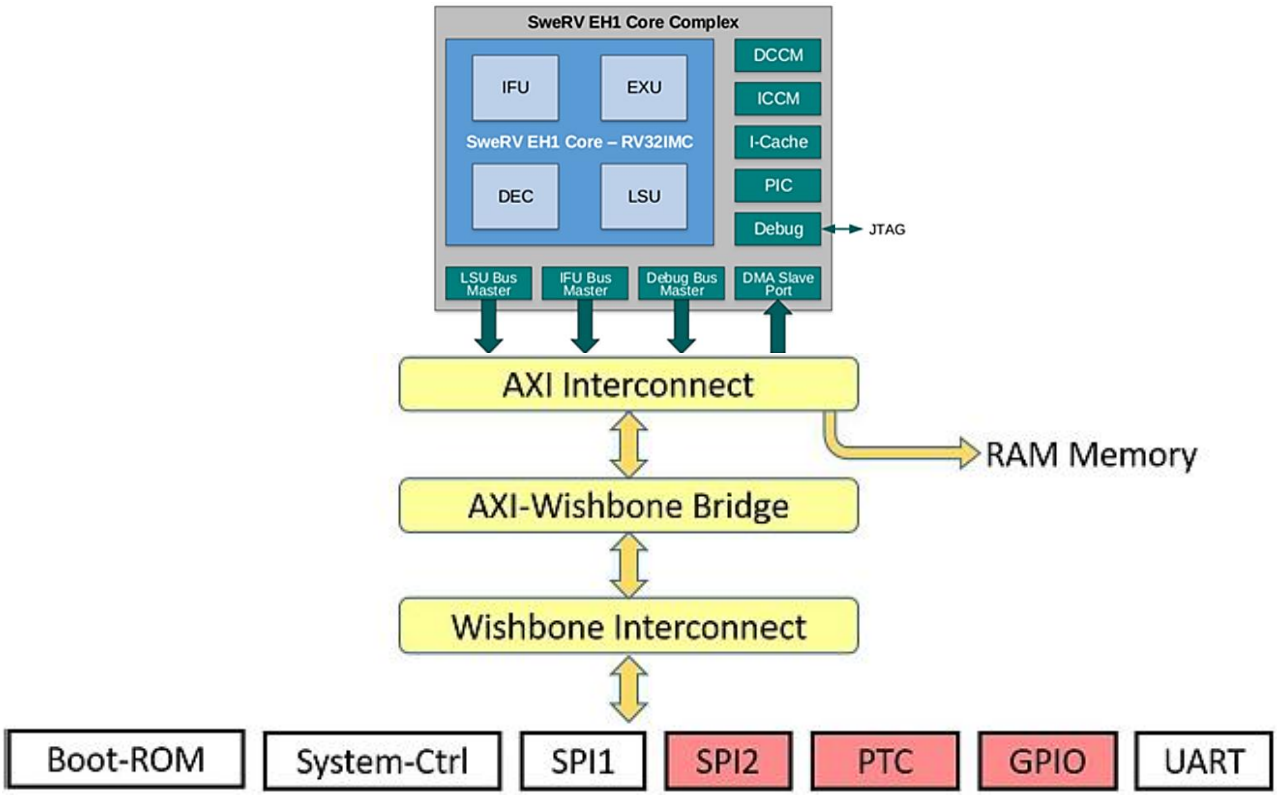
- CPU (SweRV EH1 Complejo)
- Interconexiones (AXI-Interconnect, AXI2WB y WB-Interconnect)
- Periféricos (Boot-ROM, Controlador GPIO y controlador del sistema)

Para la implementación del núcleo, existen varios módulos diferentes y no todos son necesarios para crear un SoC básico con RISC-V.

Por lo tanto, algunos módulos han sido omitidos con la finalidad de centrarse en la creación un SoC con la versión compleja del núcleo SweRV EH1. En la figura 5 se muestran los módulos que no fueron incluidos en el proyecto con color rojo:

Figura 5

Subconjuntos que forman SweRVolfX



Nota. La figura muestra de manera general los bloques que conforman al procesador. El bloque de color azul representa al núcleo SweRV EH1 básico. El bloque de color plomo abarca módulos adicionales para la versión compleja de SweRV EH1. La parte restante corresponde a interconexiones y periféricos que pueden ser añadidos para crear un SoC. Tomado del proyecto RVfpga de Imagination University Programme, versión 2021.

Entre los módulos no incluidos se encuentran: SPI2, PTC, GPIO.

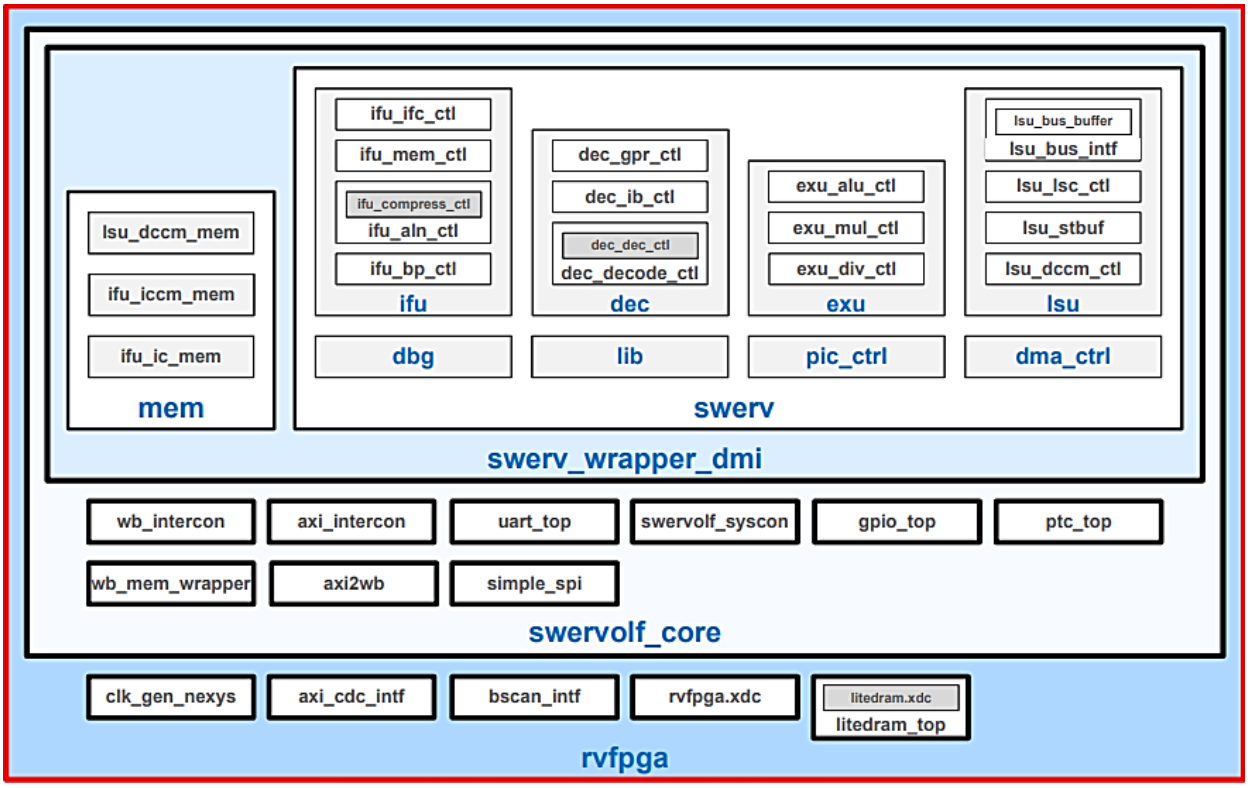
Por lo tanto, haciendo uso del SoC SweRVolfX (la X indica que se refiere a la versión extendida del núcleo SweRV) mostrado en la figura 5, su versión modificada para la tarjeta Nexys A7 se le denomida como RVfpga. Este último nombre es el que se utilizará de aquí en adelante.

Principales módulos y señales.

Los principales módulos y submódulos del sistema RVfpga con sus respectivos nombres en verilog se muestra en la figura 6. El núcleo consiste en el procesador SweRV, un controlador DRAM, un módulo generador de reloj y algunos módulos de interface.

Figura 6

Módulos y submodulos de RVfpga



Nota. La figura muestra los módulos y señales del sistema. Tomado del proyecto RVfpga de Imagination University Programme, versión 2021.

A continuación se describen los módulos principales:

- ***mem.-*** En este módulo se instancias las tres memorias disponibles en SweRV las cuales son ICCM, DCCM y $I. CCM quiere decir memorias acopladas

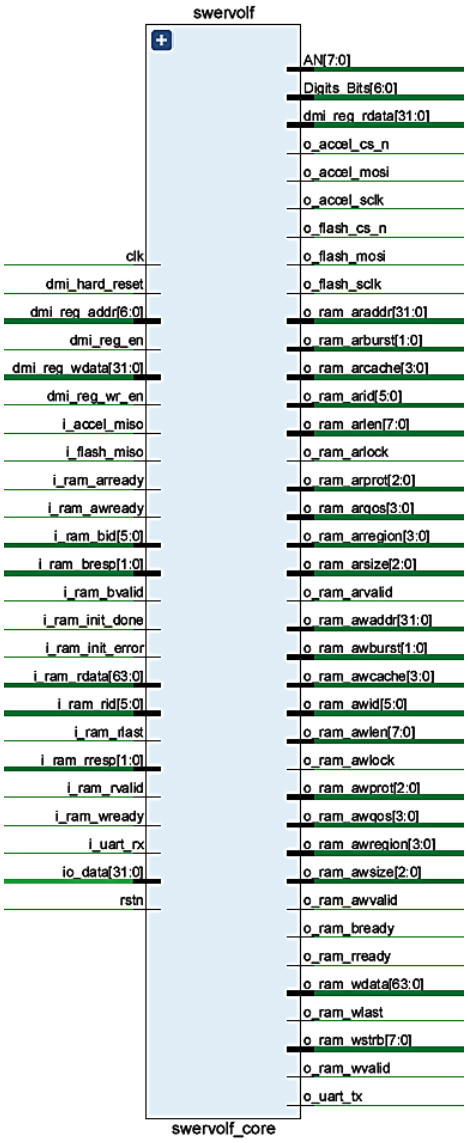
estrechamente, así entonces, ICCM es una memoria que trata con instrucciones mientras que DCCM trata con datos. La memoria $I corresponde a la cache.

- ***swerv.-*** Este módulo es el top del procesador SweRV EH1. Aquí se instancian los módulos principales del núcleo, entre los principales: ifu (Unidad de búsqueda de instrucciones), dec (Unidad de decodificación), exu (Unidad de ejecución) y lsu (Unidad de carga/almacenamiento).

En la figura 7 se muestra una parte del esquemático del núcleo correspondiente al core debido a que el tamaño es muy grande, por lo que si se desea ver el diagrama completo se sugiere verificar el material de Imagination University Programme:

Figura 7

Entidad Core del esquemático de SweRV obtenido desde Vivado



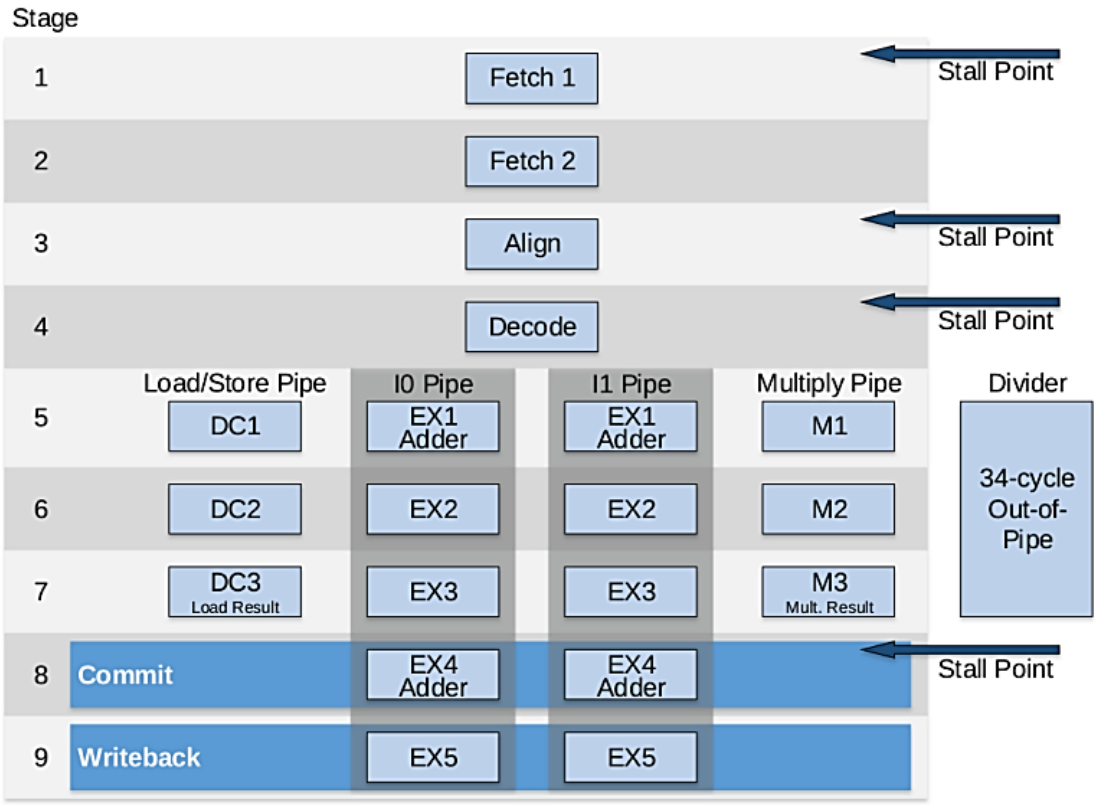
Una descripción más completa de los módulos y señales se encuentra en el apéndice G.

Microarquitectura

El procesador descrito en la figura 8 consta de 9 fases de canalización, las cuales se pueden resumir en: Fetch (Captura), Decode (Decodificación), Execute (Ejecución), Memory (Memoria) y Writeback (Reescritura).

Figura 8

Microarquitectura del procesador



Nota. La figura muestra una aproximación a la microarquitectura de SweRV EH1. Tomado del proyecto RVfpga de Imagination University Programme, versión 2021.

Las tres primeras fases mostradas corresponden a Fetch 1 (FC1), Fetch 2 (FC2) y Align.

Fetch es el responsable de la lectura de instrucciones desde la memoria de instrucciones. La dirección de la instrucción es computada por FC1 y esta es leída en FC2.

Align se encarga de realizar dos funciones. Proporciona dos instrucciones de 32 bits por ciclo a la etapa de decodificación; descomprimir instrucciones.

Las siguientes fases a analizar son Decode, Execution (EX1, EX2, EX3), Commit y Writeback.

Decode es encargada de realizar dos tareas importantes. Decodifica instrucciones y generar las señales de control; distribuye las instrucciones a los canales apropiados y proporciona los operandos.

La fase de ejecución está dividida en tres las cuales son llamadas EX1, EX2 y EX3. EX1 trabaja en conjunto con la ALU y realiza operaciones de suma o resta. EX2 y EX3 son indispensables para realizar sincronización con instrucciones de carga/almacenamiento (L/S), multiplicación (M), etc.

En la fase de Commit actúan dos multiplexores de 3:1 que ayudan a ubicar los resultados de las instrucciones a su canal correspondiente.

La última fase corresponde a Writeback, en la cual, se escriben los resultados a un archivo de registro.

Cada uno de los sistemas mostrados posee un diseño y funcionalidad diferente, pero, gracias a que ambos fueron escritos en su mayor parte en lenguaje VHDL o SystemVerilog, son perfectos para hacer uso de Vivado para su implementación. En el siguiente capítulo se muestra el proceso de construcción de los sistemas por separado y como se ejecutó el benchmark para la obtención de los parámetros de medición de los núcleos paso a paso.

Capítulo IV

Implementación de los procesadores RISC-V

Este capítulo presenta el procedimiento necesario que se utilizó para la implementación de los dos núcleos en la FPGA Nexys A7-100T, así como también la ejecución de los benchmarks una vez los núcleos se encuentren funcionando. Además, se explica cómo se obtuvo los parámetros de medición que servirán para el siguiente capítulo donde se analizarán los resultados y se compararán los procesadores.

Antes de proceder con la explicación de la implementación en la FPGA, primero se detalla de manera breve algunos puntos importantes a considerar con el fin de dejar listo el área de trabajo:

- El sistema operativo utilizado para el desarrollo del proyecto de investigación fue Ubuntu 20.04 LTS. Se puede trabajar con otras versiones de Linux, pero hay que considerar que algunos comandos que se ejecutan en el terminal están obsoletos para versiones posteriores, por lo que hay que tener cuidado con aquello.
- Se utilizó el software Vivado 2019.2 para la construcción de los núcleos. Existen otras versiones más actuales, pero, debido a que ambos núcleos fueron probados con el software indicado y funcionaron correctamente, se procedió a utilizar el mismo. Hay que considerar que, para cualquier versión de Vivado que se quiera utilizar, se deben de incluir los drivers de la tarjeta la cual se va a usar. Por ejemplo, para el presente caso, se instaló los drivers para la tarjeta Nexys A7-100T que corresponde a la FPGA que será usada para la implementación. También se puede revisar el apéndice D en donde se indica la instalación de los drivers de la tarjeta en el caso de ser necesario.
- Tener en cuenta las variables de entorno del sistema operativo e incluir al PATH la carpeta donde se instala el toolchain. Esto es importante debido a que el PATH del sistema generalmente siempre se reinicia cada vez que se enciende la PC, por lo que

las rutas que han sido añadidas con anterioridad no se guardarán y será necesario volver a introducirlas.

- Instalar Visual Studio Code con su extensión PlataformIO. Estas herramientas serán necesarias para programar la FPGA y para ejecutar/depurar programas.
- Instalar algún software que permita la comunicación serie entre la FPGA y la PC. Existen varios programas ya sea para el sistema operativo Windows o Linux. Para el presente caso se trabajó con Ubuntu, se utilizó el software Cutecom, el cual permitirá observar la comunicación a través de una interfaz gráfica amigable y adicionalmente permite enviar archivos a través del puerto serie.

Una vez vistas todas las herramientas de software necesarias, pasamos a la parte de la construcción de los procesadores. Como en el anterior capítulo, los núcleos ya fueron seleccionados, primero se detallará el proceso utilizado para la implementación del núcleo NeoRV32.

Implementación del SoC NEORV32

Instalación del toolchain

Existen varias maneras de obtener el toolchain necesario para el núcleo; la primera consiste en crear uno propio desde Scratch, la segunda en descargar e instalar el toolchain que el propio creador del núcleo ha desarrollado y la tercera consiste en utilizar un toolchain de terceros. Para este caso se utilizará la segunda opción, utilizar el toolchain que provee el propio desarrollador.

Para ello, a través del proyecto stnolting se accede a riscv-gcc-prebuilt dentro de la plataforma de Github. En la sección de toolchains disponibles se encuentran dos versiones, rv32i-2.0.0 y rv32e-1.0.0. Ambas cadenas de herramientas fueron construidas con las instrucciones de la página oficial de RISC-V GNU Compiler Toolchain en Ubuntu 20.04 LTS y

se encuentran comprimidos. Además, las dos cadenas de herramientas admiten extensiones de CPU estándar como la C (instrucciones comprimidas) y la M (Multiplicación y división).

Dentro de la misma página, en la sección de “Instalación” se deberán seguir todos los pasos y comandos descritos para instalar correctamente el toolchain. De forma resumida, en el apéndice A, se puede encontrar el proceso de instalación y los comandos ejecutados en el terminal.

Construcción del SoC con Vivado.

Para la implementación de NEORV32 y el acople para la FPGA disponible, se utilizó los archivos de configuración que han sido creados para la placa Nexys A7-100T.

Se obtiene el proyecto completo “neorv32-master” de la plataforma github y se descomprime. Dentro de la ruta rtl/test_setups se encuentran dos configuraciones minimalistas que permiten implementar un procesador muy básico con algunas características de una CPU:

- neorv32_testsetup_approm.vhd
- neorv32_testsetup_bootloader.vhd

La principal diferencia entre ambos procesos de booteo es el cómo se ejecutará el software en el procesador. Neorv32_testsetup_approm no requiere de una conexión vía UART y el software ejecutable es instalado en el bitstream para inicializar la memoria de lectura solamente. Por otro lado, neorv32_testsetup_bootloader hace uso del UART y además utiliza del gestor de arranque predeterminado para NEORV32 que permite ejecutar los ejecutables del software.

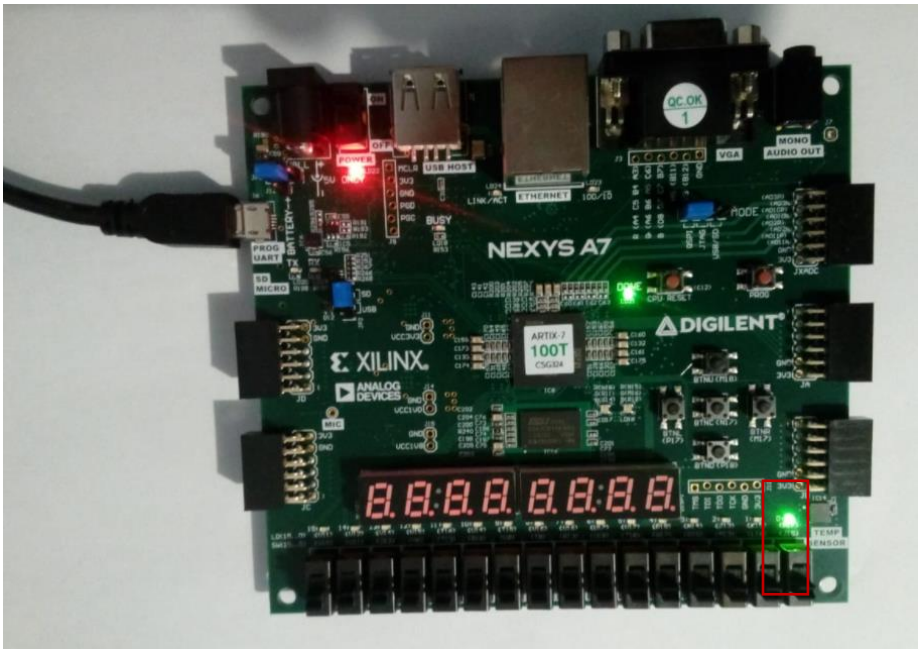
Como la segunda opción es más recomendable si la placa FPGA posee la interfaz UART, se utilizó esta configuración para cargar los archivos ejecutables al núcleo.

El siguiente paso consiste en construir el núcleo haciendo uso de la herramienta Vivado 2019.2, para lo cual se provee una guía en el apéndice B del procedimiento realizado.

NeoRV32 en funcionamiento una vez se inició dentro de la FPGA, muestra pulsaciones en uno de los leds, tal como se observa en la figura 9:

Figura 9

Procesador NEORV32 inicializándose



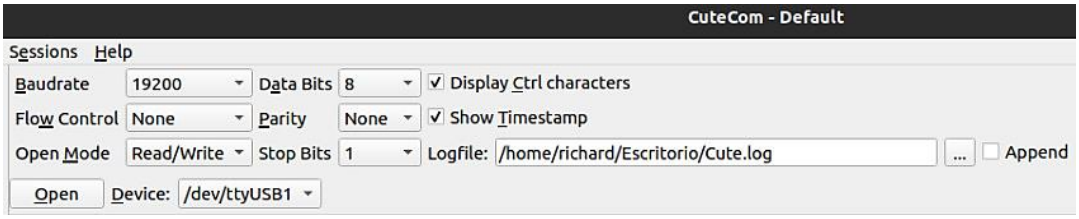
Este es un indicador de que el sistema está inicializando y que el archivo bit generado está funcionando en la fpga.

Ejecución del benchmark

Una vez se generó el archivo neorv32_test_setup_bootloader.bit, se abre Cutecom y se configuran los parámetros como se muestra en la figura 10:

Figura 10

Parámetros de configuración para el terminal serie CuteCom



Se vuelve a Vivado y se sube el archivo bit a la FPGA. Sé muestra un menú como se indica en la figura 11:

Figura 11

Menú de opciones de NEORV32

```
[23:01:07:984] << NEORV32 Bootloader >>
[23:01:08:027]
[23:01:08:027] BLDV: Nov 28 2021
[23:01:08:027] HWV:  0x01060507
[23:01:08:027] CLK:  0x05f5e100
[23:01:08:032] MISA: 0x40801104
[23:01:08:032] CPU:  0x00000081
[23:01:08:048] SOC:  0x0007000d
[23:01:08:048] IMEM: 0x00027000 bytes @0x00000000
[23:01:08:064] DMEM: 0x00027000 bytes @0x80000000
[23:01:08:096]
[23:01:08:096] Autoboot in 8s. Press key to abort.
[23:01:12:704] Aborted.
[23:01:12:704]
[23:01:12:704] Available CMDs:
[23:01:12:720]  h: Help
[23:01:12:720]  r: Restart
[23:01:12:736]  u: Upload
[23:01:12:736]  s: Store to flash
[23:01:12:752]  l: Load from flash
[23:01:12:752]  e: Execute
[23:01:12:768] CMD:>
```

Las opciones mostradas en el menú permiten visualizar instrucciones de ayuda con la letra h, resetear el sistema con la letra r, subir archivos con la letra u, guardar archivos en la memoria flash con la letra s, cargar dichos archivos con la letra l y por ultimo correr algún ejecutable que se haya enviado al núcleo con la letra e.

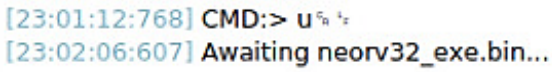
Para ejecutar el benchmark, el cual también se encuentra en la plataforma Github pero para el caso del núcleo ha sido modificado, se dirige a la ruta /sw/example/coremark del proyecto y se ejecuta el comando mostrado a continuación para generar el archivo binario:

`> *make USER_FLAGS+=-DRUN_COREMARK MARCH=rv32imc EFFORT=-O3 clean_all exe*`

Una vez generado el ejecutable binario, se exporta a NEORV32 con la ayuda de CuteCom, para lo cual se envía la letra “u” por el terminal serie, indicando que se requiere subir un archivo:

Figura 12

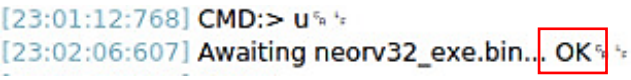
Envío de la letra "u" por CuteCom para indicar la subida de un ejecutable



Se observa que el núcleo entra en estado de espera, aguardando a que se suba el archivo binario ya generado en el paso previo. Se envía el archivo y una vez subido, aparecerá la siguiente indicación informando que la transferencia se ha completado de manera satisfactoria:

Figura 13

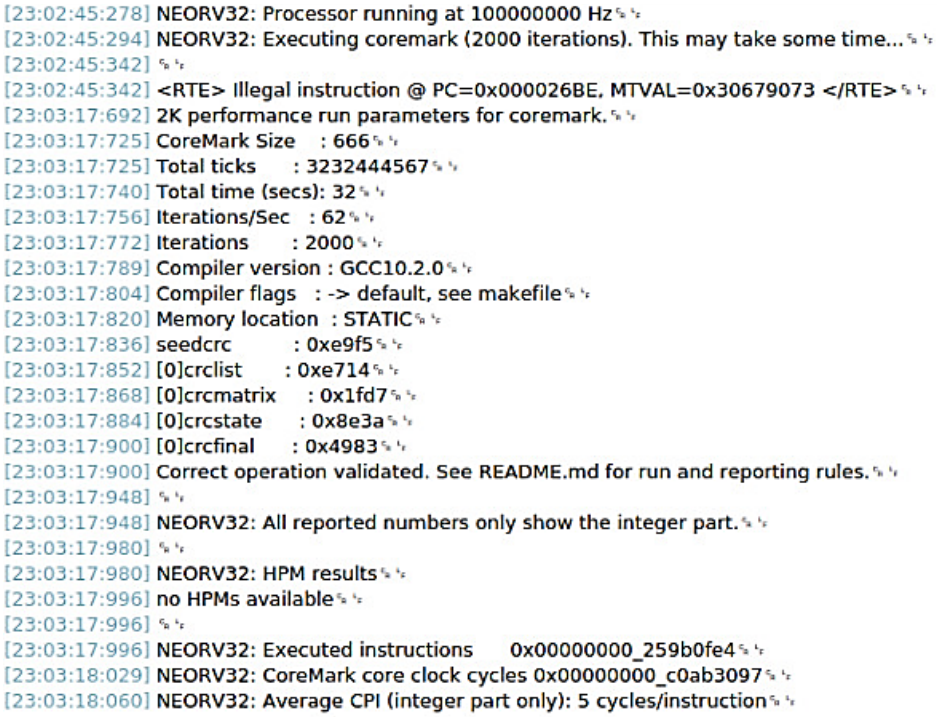
Indicación de NEORV32 de que el ejecutable ha sido cargado con éxito



Ahora se envía por el terminal serie la letra “e”, indicando que se requiere ejecutar el archivo subido, el cual corresponde al benchmark Coremark. Una vez terminado el proceso de ejecución se observan los siguientes resultados de medición:

Figura 14

Ejecución de Coremark con extensiones M, C y Zicsr



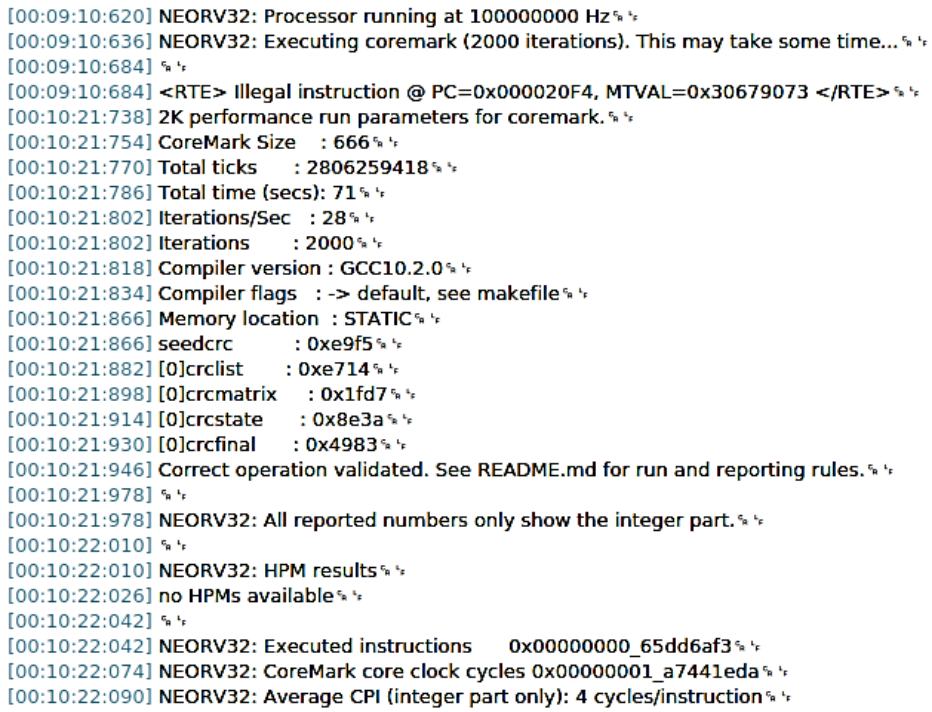
En la información provista por la plataforma de Github sobre el núcleo, se puede encontrar mediciones de rendimiento con variaciones en las extensiones. Es por esta razón que se realizaron dos mediciones diferentes.

Los parámetros mostrados en la figura 14 fueron tomados habilitando las extensiones M,C y Zicsr.

Ahora, se modifican las extensiones y solamente se habilita Zicsr. Se vuelve a ejecutar Coremark y los resultados que arroja el benchmark son los siguientes:

Figura 15

Ejecución de Coremark con extensión Zicsr



Todos los parámetros mostrados en la figura 14 y 15 en conjunto con los obtenidos con el software Vivado servirán posteriormente para realizar las comparaciones de los núcleos del siguiente capítulo.

NEORV32 ha sido implementado de manera correcta por lo que ahora pasaremos a detallar el proceso utilizado para la implementación del segundo núcleo.

Implementación del sistema RVfpga

Construcción del núcleo con Vivado.

Como ya se mencionó en el capítulo anterior, RVfpga es una adaptación del del SoC SweRVolfX para que funcione en FPGAs. Para su implementación, se usa la herramienta de

Vivado que permitirá construir todo el sistema y posteriormente se utiliza otro software para subir el archivo generado por Vivado a la FPGA. Como primer paso, se descargar el proyecto completo, el cual puede ser obtenido de manera totalmente gratuita en la página oficial de Imagination University Programme (IUP), dentro de la sección RVfpga. IUP ha desarrollado con la ayuda de varios profesionales y expertos, material educativo que permite aprender arquitectura de computadores haciendo uso de RISC-V (IUP, 2020).

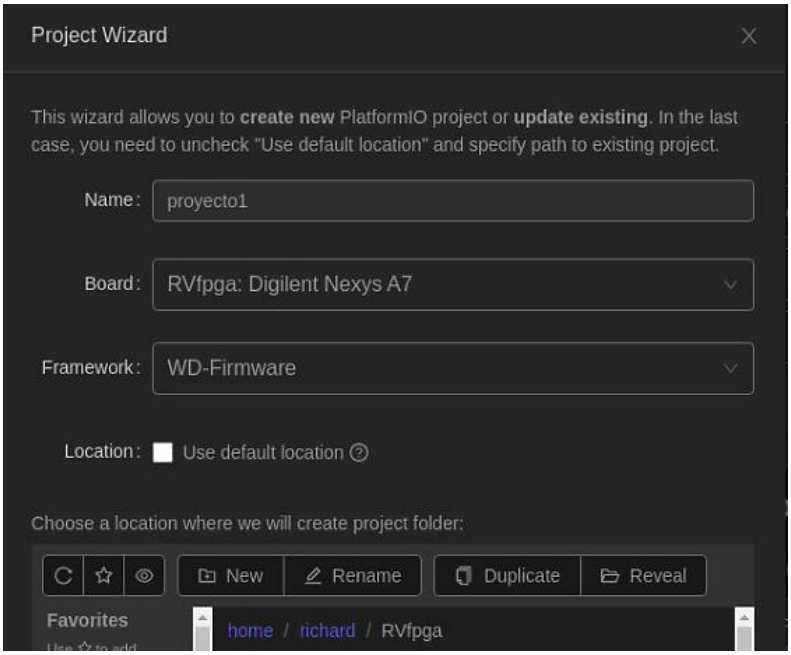
Una vez descargado el proyecto, se procede con la construcción del mismo. En la sección final, se adjunta el apéndice C, el cual indica los pasos seguidos para la implementación del núcleo en la tarjeta FPGA.

Una vez generado el archivo rvfpganexys.bit después del proceso realizado por Vivado, se pasa a hacer uso de PlatformIO. En el apéndice E, se puede encontrar la guía de instalación de Visual Studio Code y PlatformIO que puede ser usado como consulta.

Se abre PlatformIO, se crea un nuevo proyecto y se configura los parámetros nombre, placa y ruta donde se guardará, tal como se muestra en la figura 16:

Figura 16

Creación de un proyecto nuevo con PlatformIO



Una vez creado el proyecto, se modifica el fichero llamado “platformio.ini” aumentando las líneas de velocidad del monitor serie y la ruta en donde se ubica el archivo rvfpganexys.bit:

Figura 17

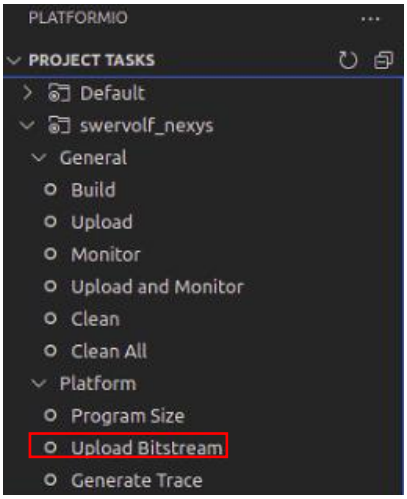
Adición de la velocidad del monitor serie y la ubicación del archivo bit generado por Vivado

```
[env:swervolf_nexys]
platform = chipsalliance
board = swervolf_nexys
framework = wd-riscv-sdk

monitor_speed = 115200

board_build.bitstream_file = /home/richard/RVfpga/src/rvfpganexys.bit
```

Como paso final, ahora se sube rvfpganexys.bit a la fpga haciendo uso de la opción “Upload Bitstream”:

Figura 18

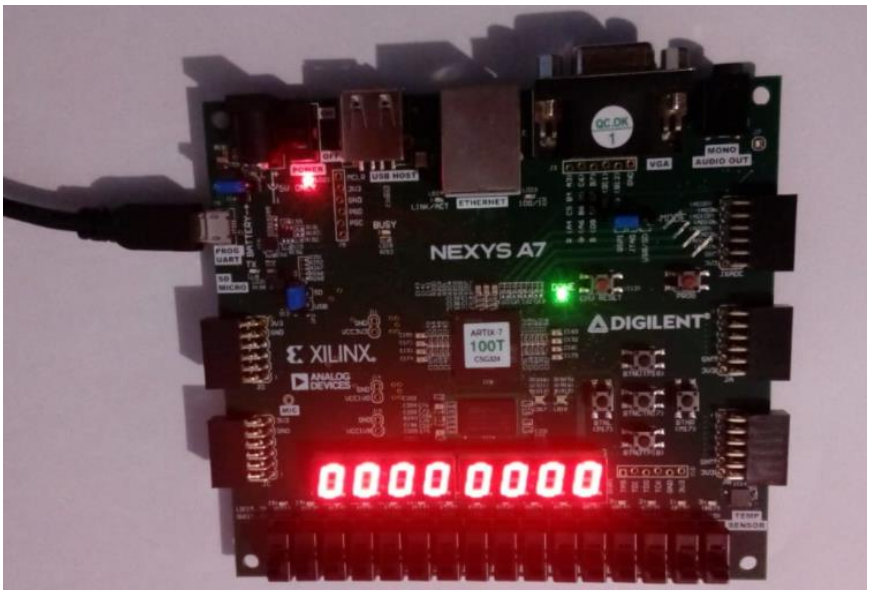
Subida del Bitstream a la FPGA



Como sugerencia, antes de realizar todo el proceso dentro de PlatformIO, se debe de encender con anterioridad la FPGA y esta debe estar conectada a la PC a través del cable USB. Si todo el proceso está correcto, se subirá el bitstream a la FPGA y esta estará lista para ejecutar programas en C. En la figura 19 se puede observar al núcleo corriendo en la fpga:

Figura 19

Sistema RVfpga inicializándose

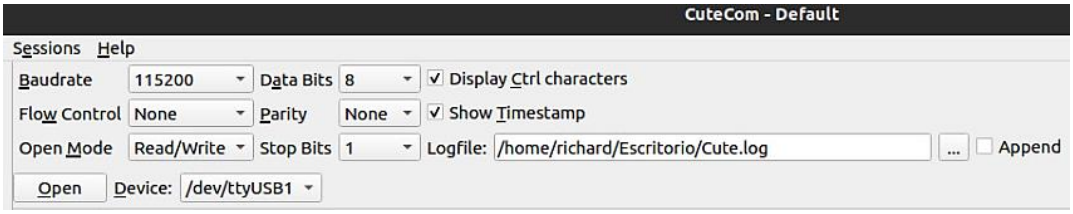


Ejecución del Benchmark

Ahora que el sistema ya está funcionando y antes de subir el benchmark programado en C, se abre CuteCom y se configuran sus parámetros tal como se describe en la figura 20:

Figura 20

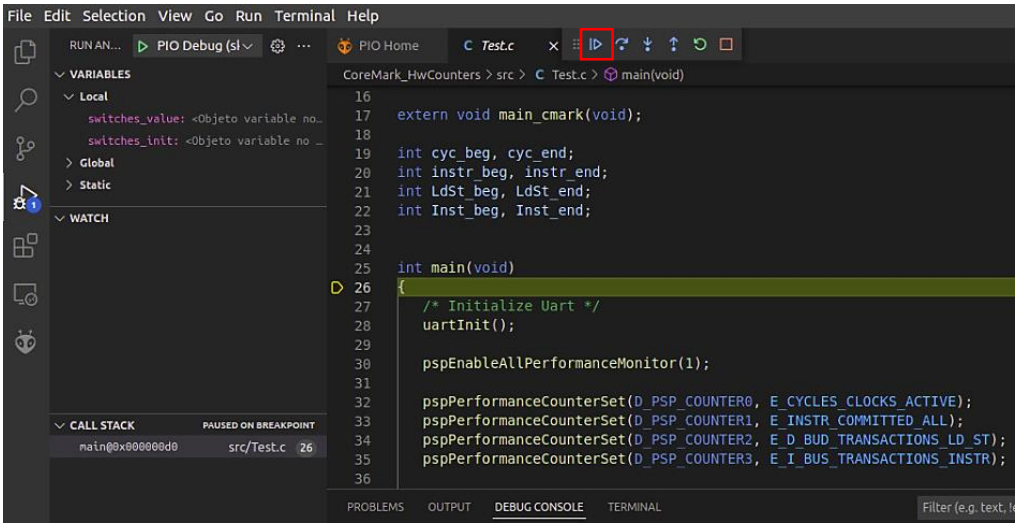
Parámetros de configuración para el terminal serie CuteCom



Ahora, se vuelve a PlatformIO, se carga el benchmark “Coremark”, el cual ha sido modificado previamente para que pueda medir el núcleo, y se lo ejecuta haciendo clic en el botón con la flecha que apunta hacia la derecha (botón play):

Figura 21

Ejecución de Coremark con PlatformIO



Se vuelve a CuteCom y se observa una indicación en el monitor serie especificando que se debe invertir cualquiera de los switches de la FPGA para parar la ejecución del benchmark y observar los resultados de medición:

Figura 22

Terminal CuteCom ejecutando el benchmark y solicitando una acción

```
[17:51:07:218] Invert any Switch to execute CoreMark
[17:51:07:234] Invert any Switch to execute CoreMark
[17:51:07:234] Invert any Switch to execute CoreMark
```

Se invierte uno de los switches y seguidamente se observan los siguientes resultados de la ejecución de Coremark en la placa Nexys A7:

Figura 23

Ejecución de Coremark en el núcleo sin optimizaciones

```
[17:51:07:410] 2K performance run parameters for coremark.
[17:51:07:410]
[17:51:07:410] CoreMark Size    : 666
[17:51:07:426]
[17:51:07:426] Total ticks      : 2100443
[17:51:07:426]
[17:51:07:426] Total time (secs): 2100
[17:51:07:426]
[17:51:07:426] Iterat/Sec/MHz   : 0.47
[17:51:07:426]
[17:51:07:426] Iterations       : 1
[17:51:07:426]
[17:51:07:426] Compiler version : GCC8.3.0
[17:51:07:426]
[17:51:07:426] Compiler flags   : -O2
[17:51:07:426]
[17:51:07:426] Memory location  : STATIC
[17:51:07:442]
[17:51:07:442] seedcrc          : 0xe9f5
[17:51:07:442]
[17:51:07:442] [0]crclist       : 0xe714
[17:51:07:442]
[17:51:07:442] [0]crcmatrix     : 0x1fd7
[17:51:07:442]
[17:51:07:442] [0]crcstate      : 0x8e3a
[17:51:07:442]
[17:51:07:442] [0]crcfinal      : 0xe714
[17:51:07:442]
[17:51:07:442] Correct operation validated. See readme.txt for run and reporting rules.
[17:51:07:461]
[17:51:07:461] Cycles = 2100204
[17:51:07:461] Instructions = 496678
[17:51:07:461] Data Bus Transactions = 133628
[17:51:07:461] Inst Bus Transactions = 392
```

Los datos obtenidos en la figura 23 fueron obtenidos con el núcleo sin optimizaciones, es decir sin usar alguna de las memorias DCCM o ICCM.

Ahora, así como se realizó el cambio para el primer núcleo y se obtuvieron dos mediciones, se incluye una de las memorias que para este caso será la DCCM.

Para incluirla, se deberá de modificar el fichero platform.ini mostrado en la figura 24 y se añade un conector para que se añada la modificación:

Figura 24

Adición de la memoria DCCM al núcleo

```
[env:swervolf_nexys]
platform = chipsalliance
board = swervolf_nexys
framework = wd-riscv-sdk


# DCCM/ICCM link scripts
board_build.ldscript = ld/link_DCCM.ld


monitor_speed = 115200

board_build.bitstream_file = /home/richard/RVfpga/src/rvfpganexys.bit
```

Debido a que se está usando el benchmark ubicado en la carpeta de RVfpga del proyecto de IUP, la conexión hacia la memoria DCCM se realiza añadiendo la línea enmarcada en el recuadro rojo de la figura 24.

Con el cambio realizado se volvió a ejecutar el benchmark y se obtuvo los siguientes resultados:

Figura 25

Ejecución de Coremark en el núcleo con memoria DCCM

```
[23:28:04:970] 2K performance run parameters for coremark.
[23:28:04:986]
[23:28:04:986] CoreMark Size    : 666
[23:28:04:986]
[23:28:04:986] Total ticks      : 529998
[23:28:04:986]
[23:28:04:986] Total time (secs): 529
[23:28:04:986]
[23:28:04:986] Iterat/Sec/MHz   : 1.89
[23:28:04:986]
[23:28:04:986] Iterations       : 1
[23:28:04:986]
[23:28:04:986] Compiler version : GCC8.3.0
[23:28:05:002]
[23:28:05:002] Compiler flags   : -O2
[23:28:05:002]
[23:28:05:002] Memory location  : STATIC
[23:28:05:002]
[23:28:05:002] seedcrc          : 0xe9f5
[23:28:05:002]
[23:28:05:002] [0]crclist       : 0xe714
[23:28:05:002]
[23:28:05:002] [0]crcmatrix     : 0x1fd7
[23:28:05:002]
[23:28:05:002] [0]crcstate      : 0x8e3a
[23:28:05:002]
[23:28:05:002] [0]crcfinal      : 0xe714
[23:28:05:018]
[23:28:05:018] Correct operation validated. See readme.txt for run and reporting rules.
[23:28:05:018]
[23:28:05:018] Cycles = 529882
[23:28:05:018] Instructions = 496678
[23:28:05:018] Data Bus Transactions = 0
[23:28:05:018] Inst Bus Transactions = 392
```

RVfpga ha sido implementado de manera correcta y con dos variaciones, por lo que ahora, en el siguiente capítulo se muestra un análisis de los parámetros de medición recogidos del benchmark y se adiciona otros más obtenidos del proceso realizado por Vivado.

Capítulo V

Análisis de resultados

Como último paso del proyecto de titulación, en este capítulo se muestra el análisis y la comparación de los distintos parámetros recogidos a través del benchmark para cada uno de los núcleos. Como se observó en el capítulo IV, se utilizaron dos maneras distintas para ejecutar el kernel de Coremark pero, al ser el mismo programa ejecutado para ambos, se pudo establecer una relación entre las variables obtenidas. A los datos adquiridos por el benchmark se le adiciona algunos de los parámetros que se obtuvo en Vivado y se verifica el rendimiento de los núcleos.

Por lo tanto, primero se procedió a analizar los recursos ocupados por cada sistema según los datos obtenidos de Vivado.

Para el caso de NEORV32, se puede apreciar que los recursos que ocupa en la FPGA son bajos. Esto también se puede observar en la tabla 6 en donde se muestran los porcentajes de utilización, los cuales no sobrepasan más del 6%. Debido a la poca cantidad de recursos que necesita, la FPGA no utilizó elementos más de los necesarios por lo que esto se verá reflejado en un consumo menor de potencia.

Figura 26

Porcentajes de utilización de la FPGA para NEORV32

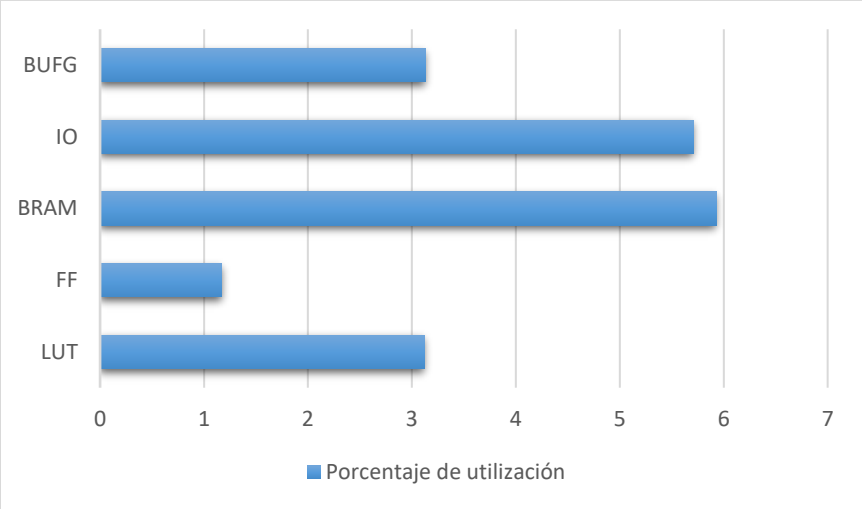


Tabla 6

Recursos de la FPGA para NEORV32 obtenida de Vivado

Recurso	Estimación	Disponible	Utilización [%]
LUT	1975	63400	3.12
FF	1479	126800	1.17
BRAM	8	135	5.93
IO	12	210	5.71
BUFG	1	32	3.13

Para el caso de RVfpga es diferente. Como se puede observar en la tabla 7, existen parámetros que sobrepasan más del 50% como es el caso de los LUTs e IO y esto se debe a que es un sistema más complejo y más grande. A pesar de ello, la FPGA sigue siendo más que suficiente para que el procesador pueda funcionar correctamente así como fue el caso para NEORV32 y debido a que se utilizan más recursos de la FPGA, esto reflejará un consumo de potencia superior.

Figura 27

Porcentajes de utilización de la FPGA para RVfpga

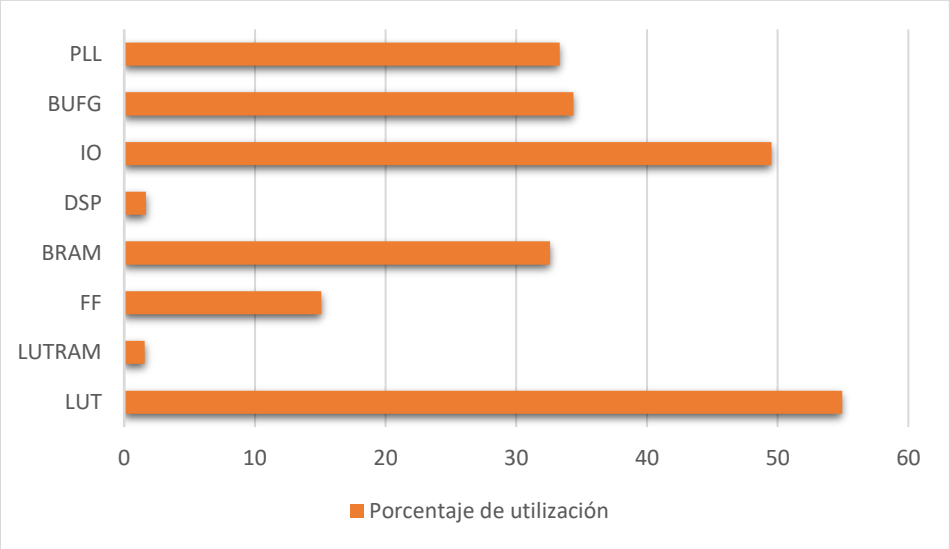


Tabla 7

Recursos de la FPGA para RVfpga obtenida de Vivado

Recurso	Estimación	Disponible	Utilización [%]
LUT	34824	63400	54.93
LUTRAM	295	19000	1.55
FF	19112	126800	15.07
BRAM	44	135	32.59
DSP	4	240	1.67
IO	104	210	49.52
BUFG	11	32	34.38
PLL	2	6	33.33

En la figura 28 se puede visualizar de manera más amplia los recursos de ambos núcleos y la capacidad máxima de cada caso según las características de la FPGA. Aquí se pude apreciar de mejor manera el porcentaje total de utilización y diferenciarlos. Se observa que NEORV32 utiliza muy poco de la tarjeta FPGA mientras que por otro lado RVfpga ocupa muchos más recursos. Pero, estos recursos utilizados son necesarios para que el sistema

completo pueda funcionar correctamente aunque evidentemente al hacer uso de más recursos, esto reflejará un consumo mayor de potencia.

Figura 28

Comparación de recursos de los procesadores y la Fpga

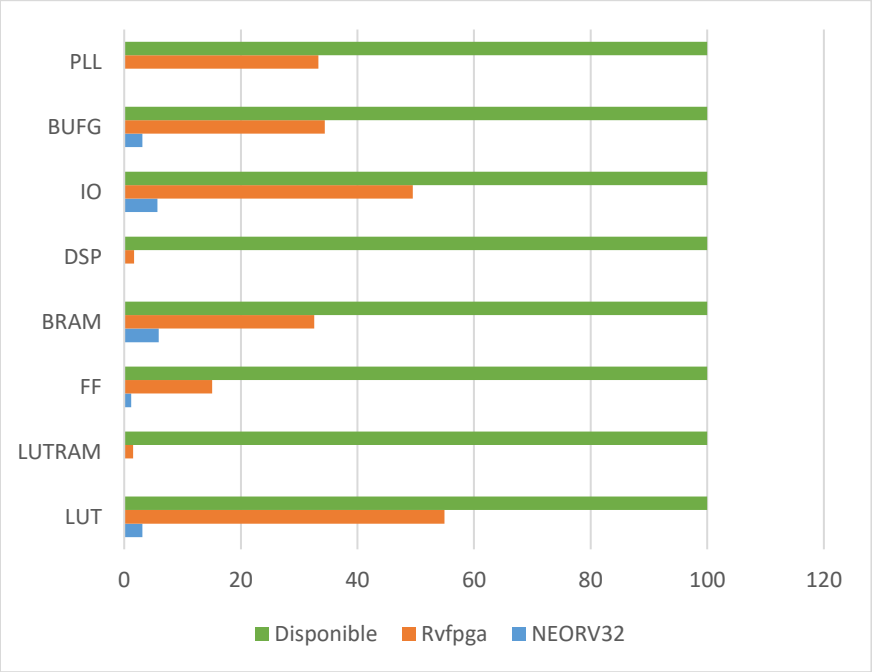


Tabla 8

Recursos que ocupan los núcleos en la FPGA obtenidos de Vivado

Parámetro/Núcleo	NEORV32	RVfpga	Disponible
LUT	1975	34261	63400
LUTRAM	0	295	19000
FF	1479	19112	126800
BRAM	8	44	135
DSP	0	4	240
IO	12	104	210
BUFG	1	11	32
PLL	0	2	6

En la tabla 9 se muestra un cuadro resumen con el dato de la potencia consumida por cada sistema después de su implementación en Vivado:

Tabla 9

Parámetros de potencia, canalización y extensiones de NEORV32 y RVfpga

Parámetro/Núcleo	NEORV32	RVfpga
Total On-Chip Power [W]	0,124	0,934
Core Power [W]	0.034	0.09
Pipeline	2	9
Extensiones	C,M, Zicsr	I,C,M

Evidentemente y como ya se observó en el párrafo anterior, los recursos de utilización de la fpga fueron distintos, siendo Rvfpga el que tiene un valor más elevado pero no sobrepasa 1 Watio. Aparte de los recursos usados para la implementación, los procesos que se realizan dentro de cada procesador dentro de cada arquitectura respectivamente también influye en el consumo de potencia. La entidad top de NEORV32, la cual corresponde solamente al núcleo sin considerar los demás elementos muestra que este consume 34 mW y la potencia total de todo el sistema es de 124 mW. Para SweRVolfX que es el núcleo para el sistema Rvfpga se muestra un consumo de 90 mW de un total de potencia en el sistema de 934 mW. Para ambos casos, la potencia consumida por los núcleos representa una pequeña parte del total, por lo que la demás potencia restante se da por los demás módulos que forman el SoC para cada sistema. Cada núcleo posee una arquitectura distinta lo cual conlleva la ejecución de procesos necesarios para que el procesador pueda operar de manera correcta.

Para NEORV32, el CPU utiliza una arquitectura canalizada básicamente en 2 fases: IF (Instruction Fetch) y EX (Execution). IF es la responsable de obtener nuevos datos de instrucción desde la memoria mientras que EX se encarga de su ejecución.

Para SweRV EH1, una aproximación inicial que ayuda a entender de mejor manera su arquitectura, se la puede apreciar verificando su canalización, la cual está compuesto por 9 fases: FC1 (Fetch1), FC2 (Fetch2), Align, Decode, EX1/DC1/M1, EX2/DC2/M2, EX3/DC3/M3, Commit y una etapa de writeback. Las fases de Fetch son las responsables de leer las

instrucciones desde la memoria. La fase de Align es responsable de dos tareas: proveer dos instrucciones de 32 bits por ciclo a la fase de decodificación y descomprimir instrucciones. La fase de decodificación también es responsable de dos tareas: ayuda a decodificar las instrucciones, genera señales de control y distribuye las instrucciones a sus respectivos canales. La fase de ejecución en la que se encuentran 2 canales para enteros, un canal para multiplicación, un canal para carga/guardado y un divisor no canalizado de 34 ciclos. En la fase de Commit, operan dos multiplexores de 3:1 los cuales seleccionan el resultado a retornar al archivo de registro. La última fase, Writeback, es la encargada de escribir el resultado al archivo de registro, el registro identifica y habilita las señales que fueron generadas en la fase de decodificación.

Por lo tanto, ahora que se conoce todos los procesos internos de cada procesador, se puede ver una clara diferencia entre ellos y por qué SweRVolfX consume más que NEORV32.

Ahora, pasando a los parámetros medidos por el Benchmark, la tablas 10 y 11 muestran un resumen de los datos mostrados por Cutecom de las figuras 14,15 para NEORV32 y 23,25 para SwerVolfX de las cuales se analizó el rendimiento a través de la puntuación arrojada por Coremark y las instrucciones por ciclo, parámetros que se explican a continuación.

Tabla 10

Parámetros de NEORV32 obtenidos a través de Coremark

Parámetro/Extensión	RV32imc_Zicsr	RV32i_Zicsr
Coremark Size	666	666
Total Ticks	32324445	28062594
Tiempo total (s)	32	71
Iteraciones/seg	62	28
Iteraciones	2000	2000
Versión del compilador	GCC10.2.0	GCC10.2.0
Localización de la memoria	STATIC	STATIC
CPI	5	4

Tabla 11

Parámetros de SweRVolfX obtenidos a través de Coremark

Parámetro/Variación	Sin optimización	Optimizado con DCCM
Coremark Size	666	666
Total Ticks	2100443	5299980
Tiempo total (s)	2,1	5,29
Iteraciones/seg	0,47	1,89
Iteraciones	1	1
Versión del compilador	GCC8.3.0	GCC8.3.0
Localización de la memoria	STATIC	STATIC
Ciclos	2100204	529882
Instrucciones	496678	496678
Transacciones del bus de datos	133628	0
Transacciones del bus de instrucciones	392	392

El número de iteraciones que maneja Coremark puede ser modificado ya que se ejecuta como un bucle. El número de iteraciones que se completan por segundo se lo conoce como CM (Coremark Score). El número de iteraciones por MHz es CM/MHz aunque también es llamado Iterat/Sec/MHz (Iteraciones/ segundos/ Megahercios).

Para el caso de NeoRV32, la puntuación CM/MHz se calcula aparte ya que el parámetro no fue incluido como si sucedió para el caso del núcleo SweRV. Para el primer caso del núcleo con las extensiones C, M y Zicsr, las cuales permiten implementar un CPU medio con características de compresión de instrucciones, multiplicación y división de enteros e implementación de instrucciones de acceso al registro de control y estado (CSR), el valor de Iterations/Sec según la tabla 10 es de 62 y la frecuencia en MHz es de 100, por lo que el CM/MHz corresponde a 0.62 lo cual corresponde a una puntuación de 62.0. Se realizan 5 ciclos por instrucción por lo que el IPC (Instrucciones por ciclo) calculado es de 0.20.

Para el otro caso que solamente utiliza la extensión Zicsr, se implementa un CPU pequeño con instrucciones CSR. Realizando el mismo procedimiento se obtiene el valor de

0.28 de CM/MHz lo cual equivale a una puntuación de 28. Se realizan 4 ciclos por instrucción por lo que el IPC calculado es de 0.25. El valor de rendimiento ha incrementado un poco.

El IPC óptimo obtenido de la hoja de especificaciones para el procesador corresponde a 0.5. El rendimiento obtenido es medio y esto se debe a que NEORV32 se basa en una arquitectura de varios ciclos, por lo que cada instrucción se ejecuta en una secuencia de varias microoperaciones consecutivas. El CPI depende directamente de la combinación de instrucciones de una aplicación en específico y de las extensiones de la CPU que se estén utilizando. Una de las formas en las que los arquitectos logran mejoras en el rendimiento de sus procesadores es a través de la reducción de la cantidad de pasos que necesita la unidad de procesamiento para ejecutar ciertas instrucciones específicas, de tal manera que el rendimiento para este caso, al usar menos extensiones mejoró.

Evidentemente también se pudo verificar que al utilizar más extensiones el rendimiento se redujo mientras que si se utiliza menos extensiones este puede incrementar aunque no fue una mejora significativa.

Para el caso de RVfpga se realizan los mismos cálculos para ambas variaciones. Para el primer caso en la que no se utilizaron optimizaciones, la puntuación CM/MHz es 0.47, la ejecución toma aproximadamente 2.1 millones de ciclos y se procesan 496678 instrucciones. Por lo tanto, el IPC corresponde al valor de 0.25.

Este rendimiento es considerado como pobre ya que el valor ideal de IPC de SweRV EH1 corresponde a 2. Sin embargo, el rendimiento se ve afectado debido a la gran cantidad de datos de lectura/escritura y una memoria DDR externa muy lenta. A pesar de ello, los resultados pueden mejorar si se usa el procesador ya sea con DCCM o con ICCM.

Para el segundo caso donde se incluye la memoria DCCM como optimización, la puntuación CM/MHz es 1.89, la ejecución toma aproximadamente medio millón de ciclos y se procesan 496678 instrucciones. Por lo tanto, el IPC corresponde al valor de 1.06.

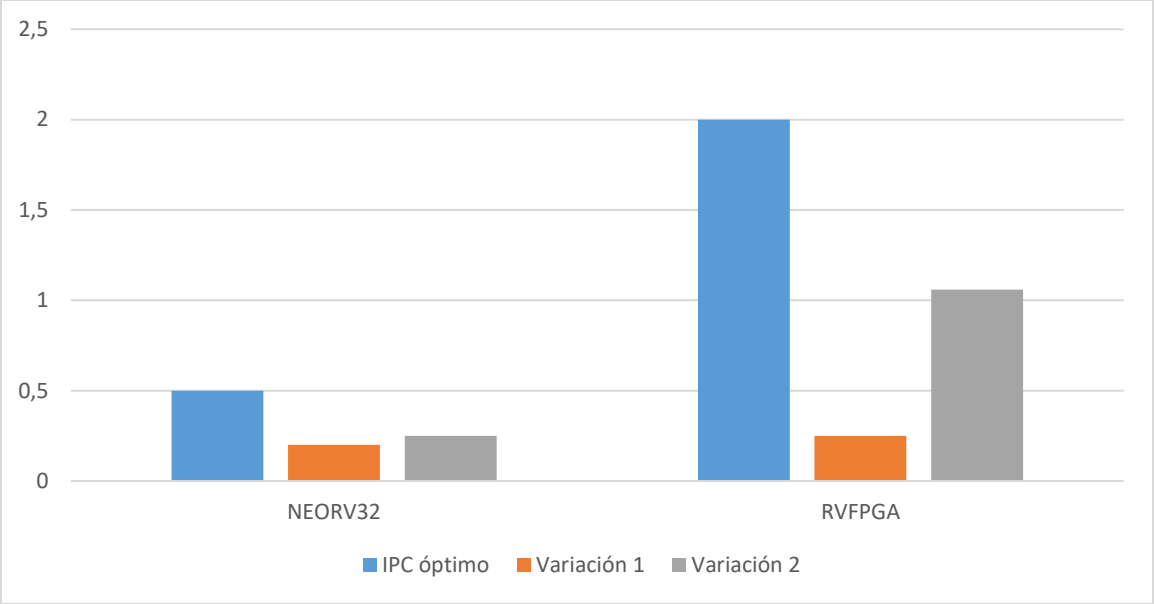
Al mapear secciones del núcleo a la memoria DCCM, el rendimiento aumento considerablemente en un factor de cuatro. Además, debido a que ahora los datos se almacenan en la memoria DCCM, el número de transacciones del bus de datos es 0, lo cual ayudó a optimizar al núcleo.

En la figura 29 se muestran los valores obtenidos de los cálculos para la obtención del IPC comparado con el valor óptimo para ambos núcleos, el cual corresponde a 0.5 para NEORV32 y 2 para SweRV.

Para el caso de NEORV32, se obtuvo mejoras pero son muy pequeñas y se mantienen en un nivel medio a su valor óptimo. Para el caso de Rvfpga sucedió algo parecido. El valor comenzó bajo pero una vez incluida la modificación de la memoria el IPC mejoró llegando a tener un valor medio al óptimo.

Figura 29

Comparación de valores de IPC obtenidos del benchmark



Por lo tanto, se pudo verificar que el rendimiento de los núcleos es influenciado por diferentes factores los cuales pueden ayudar a que el sistema mejore o que tenga un peor rendimiento. Para el caso de RVfpga, incluir optimizaciones como las memorias DCCM o ICCM ayudan a que su rendimiento mejore mientras que para NEORV32, el habilitar o deshabilitar extensiones fue lo que influenció en su rendimiento.

Capítulo VI

Conclusiones

Después de haber realizado el trabajo de investigación haciendo uso de distintas herramientas de hardware y software para la implementación de los núcleos, se logró hacer funcionar a los procesadores dentro de una sola FPGA, demostrando así la funcionalidad de un Core con arquitectura RISC-V.

RISC V es una ISA que sigue creciendo, mejorando y tiene aún mucho potencial por delante. En sus inicios solo fue utilizada con fines académicos, pero al día de hoy, se puede evidenciar un gran impacto en el área de procesadores, ofreciendo un amplio mundo de opciones para creación de núcleos. Ha despertado el interés de muchas compañías que han comenzado a utilizarlo para el desarrollo de sistemas embebidos y se prevé que en los siguientes años los núcleos sean más complejos y con un mayor rendimiento que sus antecesores.

Con respecto a los núcleos seleccionados para la implementación, ambos son de código abierto, fueron elegidos en base al tamaño del sistema dentro de la FPGA, y verificando si el procesador tenia soporte para su funcionamiento en la FPGA Nexys A7-100T. Aparte de los núcleos mostrados, existen muchas otras opciones que pueden ser exploradas pero, para el tema propuesto de titulación, NEORV32 y RVfpga fueron perfectos para poder realizar un primer acercamiento a la arquitectura RISC-V, la cual tiene mucho potencial y permite que los núcleos puedan ser diseñados o modificados para alterar la funcionabilidad y consumo de recursos.

Dentro de los implementos utilizados en la investigación, el uso de herramientas de carácter libre fueron esenciales para la implementación. Vivado en su versión gratuita permitió construir la arquitectura de los núcleos además que proveyó información importante sobre

algunos parámetros de los procesadores en cuanto a recursos y diagramas de ambos. Sin embargo, una desventaja es una alta exigencia de recursos de la computadora lo cual puede variar dependiendo de la velocidad de procesamiento del modelo que se esté usando.

Como se demostró en el análisis de resultados, la FPGA Nexys A7-100T fue más que suficiente para ambos SoCs hablando en términos de espacio y recursos utilizados. Ahora, por el lado de rendimiento de cada núcleo, ambos resultaron estar en un rango medio pero se logró verificar que el resultado puede ser mejorado si se modifica el diseño de los núcleos. Para NEORV32, el quitar o aumentar extensiones es lo que hizo que varié el rendimiento mientras que para RVfpga, el añadir la memoria DCCM fue lo que ayudo que el IPC mejore.

Con respecto a los resultados obtenidos de IPC de ambos sistemas, se pudo verificar que RVfpga, al ser un sistema mucho más grande y complejo, puedo manejar muchas más operaciones en cada ciclo que NEORV32, el cual es un sistema más pequeño. A pesar de ello, ambos están muy bien diseñados, pueden ser personalizados según la necesidad del usuario, son extensibles y pueden ser implementados en tarjetas FPGA si esta dispone de suficiente espacio.

RISC-V es sin duda una ISA que ha comenzado a causar un gran impacto en la industria de procesadores. Grandes empresas han comenzado a involucrarse debido a su éxito. Ha logrado captar la atención de varias compañías por ser una arquitectura libre, abierta y tener varios campos de aplicación, desde IoT, PCs, sistemas embebidos e incluso dispositivos móviles y en un futuro próximo se prevé que agencias espaciales también lo utilicen.

Recomendaciones

Se debe tener muy en cuenta las versiones de programas con las que se está trabajando ya que algunos diseños son sensibles a la versión que se use y no funcionan para todas. Así por ejemplo como sucedió con los núcleos implementados, ambos fueron probados en Vivado 2019.1 pero en las versiones más actuales se presentaron problemas de compilación al momento de construir los núcleos.

Es muy importante instalar todos los requerimientos mencionados para poder implementar las arquitecturas en la FPGA para que más adelante no se presenten problemas por falta de alguna herramienta o drivers que no estén en la máquina. Adicionalmente se recomienda utilizar el sistema operativo Linux ya que fue el sistema operativo utilizado para el correcto desarrollo del trabajo de investigación.

Al momento de la aplicación del benchmark y la visualización de los resultados a través de la comunicación serie, si se está trabajando con Linux es recomendable utilizar Cutecom ya que el programa es muy completo y adicionalmente permite enviar archivos a través del puerto serie para que la FPGA pueda ejecutarlos de manera correcta.

Trabajos futuros

- Con el uso de cualquiera de los dos núcleos implementados en la presente investigación, realizar algún programa de interés escrito en lenguaje C, ejecutar el software en el núcleo elegido, medir el rendimiento y adicionalmente realizar cálculos para poder obtener la potencia activa del procesador, es decir, la potencia del núcleo cuando está ejecutando el programa. Finalmente, realizar un análisis de los datos obtenidos de la medición.
- Creación de un núcleo de arquitectura RISC-V muy básico en base a alguno de los otros ejemplares que existen actualmente y que hacen uso de la arquitectura RISC-V. Una vez creado el núcleo, realizar las mediciones respectivas y analizar los resultados obtenidos.
- Hacer correr Linux en un núcleo y verificar el funcionamiento y rendimiento de la FPGA corriendo el sistema operativo. Se puede hacer uso de RVfpga para cumplir dicha tarea, ya que el núcleo si logra soportar un sistema operativo (SO) muy básico con algunos comandos usados en un terminal de Linux. En el caso de que no se quiera utilizar RVfpga se puede hacer uso de algún otro núcleo ya que existen otros más que logran soportar un SO.
- Añadir más periféricos a cualquiera de los núcleos y analizar su rendimiento con los periféricos adicionales. Como ya se explicó en el escrito, ambos núcleos son extensibles y permiten la adición de otros elementos más al sistema, por lo que sería interesante verificar el funcionamiento de los núcleos con la adición de los nuevos periféricos.

Bibliografía

Campos, F. (16 de Marzo de 2021). *códigoespagueti.com*. Obtenido de Qué es y cómo funciona la arquitectura x86: https://codigoespagueti.com/noticias/tecnologia/que-es-y-como-funciona-la-arquitectura-x86/

Cocke, J., & Markstein, V. (1990). *La evolución de la tecnología RISC en IBM.*

Digital, W. (2022). *EH1 RISC-V SweRV CoreTM 1.9 from Western Digital*. Obtenido de https://github.com/chipsalliance/Cores-SweRV

Doran, R. (2005). *Computer architecture and the ACE Computers.*

EEMBC. (2009). *CoreMark*. Obtenido de https://www.eembc.org/coremark/

Etecé, E. (05 de Agosto de 2021). *Procesador*. Obtenido de https://concepto.de/procesador/

Hennessy, J., & Patterson, D. (1993). *ARQUITECTURA DE COMPUTADORES Un enfoque cuantitativo.* España: McGraw-Hill.

Hernández, A., Tejedor, C., & Iglesias, A. (2010). *Aquitectura de MIPS.* Valladolid: 1era edición.

IUP. (2020). *TRAINING & TEACHING*. Obtenido de RVfpga: Understanding Computer Architecture: https://university.imgtec.com/resources/download/rvfpga-understanding-computer-architecture-rvfpga_v2-1_en/

Labs, P. (2021). *Professional collaborative platform for embedded development*. Obtenido de https://platformio.org/

MindChasers. (23 de Abril de 2021). *Introducción a la cadena de herramientas GNU de código abierto RISC-V*. Obtenido de https://mindchasers.com/dev/rv-getting-started

Monteiro, J. (2019). *Configurable RISC-V softcore processor for FPGA implementation*. Obtenido de https://www.semanticscholar.org/paper/Configurable-RISC-V-softcore-processor-for-FPGA-Rodrigues/ff2d07534cb1b3de41dfe25d56a52084e5fb008f

Morales, R. (2019). *Arquitectura del conjunto de instrucciones del microprocesador*. Obtenido de www.ticarte.com

Neundorf, A. (2009). *Bienvenido a CuteCom*. Obtenido de http://cutecom.sourceforge.net/

Nolting, S. (2022). *The NEORV32 RISC-V Processor*. Obtenido de https://github.com/stnolting/neorv32

Nuel, C. (2012). *¿Qés es la arquitectura ARM?* México.

Nurmi, J. (2007). *Processor design: system-on-chip computing for ASICs and FPGAs.*

OpenRISC. (2021). *Descripción general del proyecto OpenRISC*. Obtenido de https://openrisc.io/

Parejo Quirós, J. C. (2016). *Prototipado de sistemas basados en procesadores RISC-V.* Madrid.

Pastor, J. (2015). *Así empezó ARM, un gigante que cumple 25 años.*

Patterson, D. (1980). *The case for the reduced instruction set computer.*

Patterson, D., & Hennessy, J. (2004). *Estructura y diseño de computadoras.* reverte S.A.

Patterson, D., & Sequin, C. (1981). *RISC I : A Reduced Instruction Set VLSI Computer .* Minneapolis.

Platform, P. (2022). *PULP Features*. Obtenido de https://pulp-platform.org/

Research, B. A. (2019). *Rocket Core*. Obtenido de https://chipyard.readthedocs.io/en/latest/Generators/Rocket.html

Rocha Pacheco, N. (2019). *REVISIÓN DEL ESTADO DEL ARTE SOBRE EL CONJUNTO DE INSTRUCCIONES RISC-V.* Bogotá: Universidad de los Andes.

Samsoniuk, M. (2018). *darklife*. Obtenido de DarkRISCV: https://github.com/darklife/darkriscv

Sequin, C., & Patterson, D. (1981). *RISC I: A Reduced Instruction Set VLSI Computer.*

Shaya Zamudio Vissuet, S. A. (2003). *Diseño e implementación de un microprocesador RISC en VHDL.* Ciudad de México: Instituto Politécnico Nacional.

Simmonds, C. (2015). *Mastering Embedded Linux Programming*. Obtenido de https://subscription.packtpub.com/book/networking-and-servers/9781784392536/copyrightpg

SpinalHDL. (2022). *VexRiscv (RV32IM CPU)*. Obtenido de https://spinalhdl.github.io/SpinalDoc-RTD/v1.3.1/SpinalHDL/Libraries/vexriscv.html

Starnes, T. (1983). *Filosofía de diseño detrás del MC68000 de Motorola.*

Studio, V. (2021). *Docs*. Obtenido de https://code.visualstudio.com/docs

Súarez Santamaría, E. Z. (2019). *RISC-V UN ISA DE CÓDIGO ABIERTO.* Santander: Universidad de Cantabria.

Varona , R., López, O., & Martínez, J. (2015). *Arquitectura SPARC: Conceptos generales*. Obtenido de https://www.infor.uva.es/~bastida/OC/GeneralSPARC.pdf

Vega, J. (s.f). *Arquitectura SPARC: Conceptos generales.*

VidaBytes. (2020). *Historia de los procesadores ¡Así fue su gran origen!* Obtenido de https://vidabytes.com/historia-de-los-procesadores/

Waterman , A., & Patterson, D. (2018). *GUÍA PRÁCTICA DE RISC-V El Atlas de una arquitectura abierta.*

Waterman, A. (2016). *Design of the RISC-V Instruction Set Architecture.* California.

Waterman, A., & Asanovic, K. (2017). *The RISC-V Instruction Set Manual.* California.

Waterman, A., Lee, Y., Patterson, D., & Asanovi, K. (2014). *The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA version 2.* California.

Wolf, C. (2015). *PicoRV32*. Obtenido de https://github.com/YosysHQ/picorv32

Xilinx. (2019). *Vivado 2019.2 - Installation and Licensing*. Obtenido de https://www.xilinx.com/support/documentation-navigation/design-hubs/2019-2/dh0013-vivado-installation-and-licensing-hub.html

Apéndices