$DIFU_{100}ci@$
Revista de Difusión Científica, Ingeniería y Tecnologías
Vol. 17, No. 3, Septiembre - Diciembre 2023
ISSN:2007-3585
Artículo arbitrado
Publicado: 31 de diciembre de 2023

Development of a configurable soft microcontroller based on ISA RISC-V

Desarrollo de un microcontrolador soft configurable basado en el ISA RISC-V

Ismael Hurtado Martínez[1], **Remberto Sandoval Aréchiga**[1], **José Ricardo Gómez Rodríguez**[1], **Víktor Iván Rodríguez Abdalá**[1], **Oscar Osvaldo Ordaz García**[1], **Cristian Eduardo Boyain y Goytia Luna**[1], and **Salvador Ibarra Delgado**[*1]

[1] *Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica,*
Posgrado en Ingeniería para la Innovación Tecnológica,
Jardín Juárez, 148, Col. Centro, Zacatecas, Zacatecas, México, CP.98000
{36171075,rsandoval,jrgrodri,abdala,oscarordazg,cristian.boyain,sibarra}@uaz.edu.mx

Abstract

RISC-V is an open source architecture for the development of RISC type processors. In this work, based on the RISC-V 32I architecture, a configurable soft microcontroller is developed under the AMD-Xilinx Vivado development platform. The system allows the development of peripheral device blocks that can be added to the RISC-V processor using Vivado's IP Integrator tool, enabling rapid customization of the microcontroller. The way to integrate the peripherals to the processor is through the AXI-Lite interface, which in our proposal allows saving AXI slaves, especially in the integration of timers and PWMs.

***Keywords*—** RISC-V, microcontrollers, soft processor

Resumen

RISC-V es una arquitectura de uso libre para el desarrollo de procesadores tipo RISC. En este trabajo, tomando como base la arquitectura RISC-V 32I, se desarrolla un microcontrolador soft configurable bajo la plataforma de desarrollo Vivado de AMD-Xilinx. El sistema permite desarrollar bloques de dispositivos periféricos que pueden ser añadidos al procesador RISC-V utilizando la herramienta IP Integrator de Vivado, lo que permite una rápida personalización del microcontrolador. La forma de integrar los periféricos al procesador es por medio de la interfaz AXI-Lite, que en nuestra propuesta permite el ahorro de esclavos AXI, especialmente en la integración de los timers y PWMs.

***Palabras clave*—** RISC-V, microcontrolador, procesador suave

I. Introducción

En tan solo un poco mas de una década, el proyecto RISC-V (del inglés Reduced Instruction Set Computer V5) acuñado por la Universidad de California Berkley, se ha posicionado fuertemente como un competidor más en la arena del desarrollo de unidades de procesamiento. Dentro de las características que le han permitido este rápido posicionamiento se pueden mencionar: primera, proveer un conjunto de instrucciones ISA (del inglés, Instruction Set Architecture) libre y abierto, que le ha permitido ganar una gran cantidad de adeptos a nivel mundial; segunda, poseer un ISA modular a partir de uno de los núcleos principales base: RV32I, RV64I, RV128I o RV32E, lo cual permite estabilidad a ensambladores, compiladores, sisetemas operativos, etc. y además permitir extensiones de hardware que se pueden incorporar de acuerdo a las aplicaciones que serán implementadas en el procesador.

Las características mencionadas anteriormente han per-

[*]Autor de correspondencia

mitido por un lado, tener una comunidad muy amplia de colaboradores alrededor de la iniciativa, lo que ha llevado a grandes compañías a unirse activamente a esta iniciativa, solo por mencionar algunos: Google founding, Huawei, Intel, Qualcomm, ZTE, Bosh, NXP. Por otro lado, la modularidad que presenta la arquitectura permite que pueda ser utilizada y adoptada para diversas necesidades y aplicaciones que pueden ir desde: su uso para el desarrollo de microcontroladores de bajo consumo para aplicaciones del Internet de las cosas IoT (del inglés Internet of the Things), aplicaciones en la industria automotriz, aplicaciones para Inteligencia Artificial, hasta aplicaciones para su uso en dispositivos móviles, electrónica de consumo, infraestructura para centros de datos y computo en la frontera (Edge Computing, en inglés).

En este articulo presentamos el desarrollo de un microcontrolador basado en el núcleo RV32I que puede ser configurado bajo el entorno de desarrollo Vivado de AMD-Xilinx. Al procesador se le pueden integrar puertos paralelos de I/O de tamaño y mapa de direcciones personalizados. La misma flexibilidad se puede lograr con otros elementos como Timers y PWM.

El resto del artículo esta organizado de la siguiente manera: En la sección II se presenta una descripción del ISA RISC-V y por otro lado alternativas de microcontroladores desarrollados con procesadores basados en ISA RISC-V. En la sección III se muestra nuestra implementación del núcleo RISC-V 32I. En la sección IV presentamos como los periféricos son integrados el procesador y como son integrados por medio de las herramientas de Vivado. Posteriormente, en la sección V se muestran los resultados y finalmente, en la sección VI presentamos los resultados.

II. Marco Teórico

II.1. ISA RISC-V

Una de las características principales del ISA RISC-V es su flexibilidad. Aunque el conjunto de instrucciones básico es de 32 bits, existen diferentes longitudes de tamaño de instrucción: 32 bits, 64 bits, etc. Una revisión detallada se puede encontrar en [1]. En este trabajo nos enfocamos en describir el ISA RV32I.

RV32I es el núcleo básico del ISA RISC-V, consta de 47 instrucciones para realizar operaciones de tipo entero. Como se indica en [2] Se tienen seis formatos de instrucciones básicos: tipo-R para operaciones entre registros; tipo I para inmediatos cortos y loads; tipo-S para stores; tipo-B para bifurcaciones; tipo-U para inmediatos largos; y tipo-J para saltos incondicionales. El formato de estas instrucciones se puede observar en la Figura 1.

Además, en [2] se indica que dentro de las principales características que se pueden mencionar con respecto al formato de las instrucciones se encuentran:

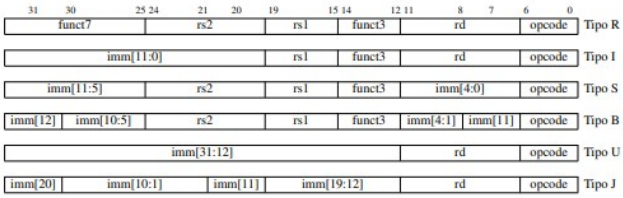

Figura 1: *Formato de Instrucciones RISC-V fuente: [1]*

- Solo existen seis formatos y todas las instrucciones son de 32 bits.
- Las instrucciones ofrecen operandos de tres registros, no tienen campos compartidos para origen y destino.
- En el formato de instrucción los bits de los registros a ser leídos y escritos siempre van en la misma posición para todas las instrucciones, esto implica que se puede acceder a dichos registros antes de la decodificación.
- Los campos inmediatos de estos formatos siempre son extendidos en signo, y el bit del signo siempre está en el bit más significativo de la instrucción.
- La arquitectura está diseñada para que *datapaths* similares compartan la mayor cantidad de bits posibles simplificando la lógica de control.
- Las direcciones de saltos siempre están corridas un bit a la izquierda para multiplicar la dirección por dos dando un mayor rango de memoria al RISC-V.
- El espacio de memoria es de 32 bits direccionables por byte.
- Todas las operaciones son entre registros.
- No hay instrucciones de multiplicación ni división.

RV32I contiene 32 registros de 32 bits cada uno (x0 - x31). El registro x0 siempre tiene el valor de 0. Esto último simplifica el ISA del RISC-V. Diferentes tipos de pseudoinstrucciones pueden ser ejecutadas tomando ventaja del registro x0. Un resumen de estas pseudoinstrucciones se puede encontrar en [2]

Una de las características más importantes de RISC-V son las extensiones que se pueden hacer del ISA, entre las que se pueden mencionar: RV32M, para multiplicación y división; RV32FD, para punto flotante de presición simple y doble; RV32A, para operaciones atómicas; RV32C, para instrucciones comprimidas; RV32V para instrucciones vectoriales. La posibilidad de crear estas extensiones se debe principalmente a que el espacio destinado el código de operación ocupa menos de la octava parte del formato de la instrucción.

II.2. Implementaciones soft RISC-V monociclo

Dadas las características del ISA RISC-V es común que la comunidad implemente procesadores con esta arquitectura sobre dispositivos lógicos programables. Estos

procesadores son denominados procesadores tipo *soft*, una de las principales características es que al ser implementados en esta tecnología los sistemas tienen la capacidad de ser reconfigurados, lo cual proporciona mucha flexibilidad. Existen diferentes formas de implementar estos núcleos de procesamiento desde monociclo que ejecutan una instrucción en un ciclo de reloj, hasta implementaciones pipeline que dividen la ejecución de las instrucciones en etapas con la finalidad de lograr en mayor rendimiento. En este trabajo el procesador utilizado como base del microcontrolador es un procesador monociclo, por lo cual centraremos nuestra investigación en este tipo de procesadores.

En [3] se realiza la implementación de un procesador RISC-V 32I sintetizado en una FPGA Spartan 3E XC3S500E para desarrollos de bajo costo. Los autores hacen uso de block RAM para generar la memoria de datos, no se especifica claramente bajo esta implementación como logran implementar las operaciones de *load* en un solo ciclo. Por su parte en [4] se hace la implementación del mismo núcleo de procesamiento en un SoC Zedboard. Al igual que el anterior utiliza memoria block RAM pero bajo una arquitectura Von Neumann, no queda claro como logran ejecutar las operaciones de *load* en un ciclo de reloj.

Por su parte en [5] presentan una plataforma que permite sintetizar diferentes núcleos RISC-V 32I. Uno de esto es el procesador monociclo. Los autores indican que para lograr que todas la operaciones se ejecuten en un ciclo de reloj, es necesario hacer uso de las LUT para implementar la memoria de datos, lo anterior además de limitar la frecuencia de trabajo, hace uso de un recurso muy valioso dentro del FPGA.

II.3. Microcontroladores con CPU RISC-V

Diversos autores han presentado alternativas de desarrollo de microcontroladores: Duran et. al. en [6] muestran un microcontrolador basado en RISC-V de 32 bits sintetizado para una tecnología de 132nm que tiene como buses de comunicación AXI4-Lite y APB. El microcontrolador contiene un ADC de 10 bits, un DAC de 12 bits y un GPIO de 8 bits. El sistema contiene 4kB de memoria RAM y una interfaz SPI AXI para verificación. Los autores indican que su arquitectura puede ser utilizada en algunas tareas reemplazando el microcontrolador M0 de ARM.

Por su parte en [7] presentan una plataforma para el Internet de las cosas basada en el RISC-V de 32 bits que se presento en [6], en esta propuesta integran protocolos de comunicación SPI, IIC, SDIO y JTAG. Su frecuencia máxima de operación es de 160MHz integran un ADC de 10bits y un DAC de 12 bits. Tienen un total de 8 puertos para GPIO.

Otra propuesta de un microcontrolador basado en el ISA RISC-V lo presenta en [8] ellos desarrollan un microcontrolador en tecnología de 28nm poniendo especial atención a sistemas tolerantes a fallos en ambientes hostiles como es el espacio. Ellos presentan un microcontrolador que soporta la triple redundancia con núcleos RISC-V en caso de que las condiciones de operación necesiten resultados altamente confiables, si no es así, los núcleos RISC-V pueden ejecutar tareas independientes que pueden ser configurados en tiempo de ejecución. La memoria la protegen por medio de un método de corrección de errores (ECC, error correction code, por sus siglas en inglés) en cada uno de los bancos de memoria SRAM que utiliza. Desde el punto de vista de conectividad del microcontrolador, el sistema provee SPI, UART, GPIO, timer, JTAG. Sus resultados muestran que pueden operar a una frecuencia de operación de 250MHz con un consumo de potencia de 19.7mw.

En el trabajo presentado en [9] se realiza la implementación de un núcleo de procesamiento basado en RISC-V destinado a ser utilizado en microcontroladores. La mayoría de sus instrucciones son realizadas en un ciclo de reloj exceptuando las instrucciones de *load* que se ejecuta en dos ciclos, lo anterior debido a que la implementación de la memoria RAM se hace sobre memorias BRAM lo que impide hacer la escritura en un solo ciclo de reloj.

En el campo comercial existen diversas propuestas de microcontroladores basados en ISA RISC-V en [10] se encuentra el procesador personalizable NEORV32, este procesador cuenta con UART, SPI, PWM, GPIO, timer, watchdog, IIC. Desde la misma construcción del procesador este sistema es altamente configurable y además provee múltiples opciones para la configuración de memorias, timers, Entrada/Salida, y Conectividad

III. Implementación RISC-V 32I monociclo

La base del microcontrolador desarrollado es un procesador monociclo con el ISA RV32I. en la 2 se muestra el diagrama principal de la arquitectura. Antes de describir nuestra implementación, es importante mencionar que el problema al que nos enfrentamos para lograr que el sistema pueda lograr la ejecución de todas las instrucciones en un ciclo de reloj es la dependencia de datos entre instrucciones. Es decir, cuando se ejecuta una instrucción de *load* la memoria BRAM tarda un ciclo de reloj en entregar el dato y otro ciclo en almacenarlo en al Banco de Registros, si la siguiente instrucción necesita el dato leído, este aun no se encuentra presente en el registro asociado en el banco de registros, por lo que sería necesario esperar un ciclo de reloj para poder ejecutar la siguiente instrucción.

Nuestra implementación consta de las siguientes unidades:

Memoria de Programa (MP): Esta es una memoria

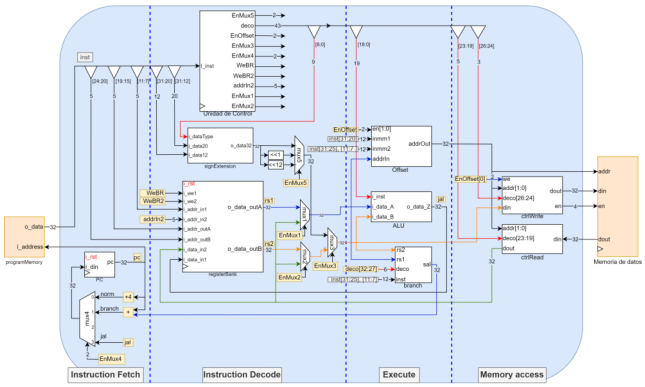

Figura 2: *Diagrama de bloques de la implementación RISC-V 32I*

ROM de 32 bits donde se encuentra almacenado el programa que va a ejecutar el núcleo RV32I. En nuestra implementación es necesario cargar desde el proceso de síntesis el programa que va a ser ejecutado por el procesador.

Program Counter (PC): Registro encargado de apuntar a la siguiente instrucción que será ejecutada. Este registro es modificado de acuerdo a la instrucción que se ha ejecutado. Esta modificación es controlada por un multiplexor cuyas líneas de control provienen de la Unidad de Control del sistema. Si el curso del programa sigue un flujo normal el PC es incrementado en 4 unidades, si el programa debe de ejecutar una instrucción de salto condicional o absoluta el PC es modificado con esta dirección.

La Unidad de Control (UC): Esta unidad es la encargada de decodificar la instrucción proveniente de la MP y generar la lógica necesaria para indicar por un lado la instrucción que debe de ser ejecutada, por otro lado establecer la fuente de los datos con los que se va a operar, determinar como será midificado el PC y finalmente generar la sincronización adecuada para que la ALU opere con los datos adecuados.

Extensión del Signo (SE): Este elemento es el encargado de extender el signo para aquellas instrucciones que así lo requieran.

Branch: Este elemento es el encargado de comparar registros y a partir de ello, generar operaciones de salto para aquellas instrucciones que así lo requieran.

Banco de Registros (BR): El banco de registros es implementado en base a LUTs y es de doble puerto, estos registros son la fuente de datos con los que se operan las instrucciones y además, el destino de las instrucciones ejecutadas. El control de este banco de registros depende de la UC.

Unidad de Aritmética y Lógica (ALU): Esta unidad es la encargada de realizar la instrucción que le indique la UC con la fuente de datos que también sean indicados por la UC. El resultado de estas operaciones es almacenado en el BR.

Existen otros elementos de apoyo en la arquitectura del RISC-V 32I que sirven para controlar la lectura y/o escritura de la memoria de datos, además de calcular el offset para las direcciones de memoria.

El problema principal que enfrentan los diseños monociclo cuando son implementados en FPGAs con memoria del tipo block RAM es que la lectura de un dato de memoria consume un ciclo de reloj, el dato almacenado en la dirección seleccionada esta disponible en el bus de salida pero será almacenado en el registro indicado por la instrucción un ciclo después. Esto no tendría inconveniente si no fuera porque este dato sea necesario para operar la siguiente instrucción en el flujo de programa. Si esto es así, la operación se llevaría a cabo con el dato que esta actualmente en el registro no actualizado del banco de registros.

La solución que proponemos para lograr ejecutar la instrucción con el valor del registro deseado es poder interceptar el dato proveniente de memoria y que por medio de la UC cuando lea la siguiente instrucción a ejecutar determine si en la fuente de los datos es necesario utilizar el dato del registro que aún no esta almacenado. Si esto es así, la UC puede manipular la señal de control de los multiplexores 1 y 2 en el diagrama de la Figura 2. Con esto tenemos de forma inmediata el valor actualizado del dato o datos con el que se quiere ejecutar la instrucción y además, mientras la instrucción se esta ejecutando el dato es almacenado en el BR. De este modo aseguramos que se cumpla el objetivo de ejecutar una operación por ciclo.

En la Figura 3 se muestra el diagrama general de la UC de nuestra implementación. En la parte inferior de la misma es donde se genera la lógica que permite: uno, detectar si la operación es de *load*; dos, retardar tanto la señal de escritura al BR como la dirección del registro que será escrito un ciclo después. Cuando la UC detecta que en la nueva instrucción a ejecutar, se ve involucrado al menos en un operando el registro de carga de la instrucción anterior, manipula las señales de control a los multiplexores 1 y 2 para que permitan pasar el dato directamente del bus de salida de la memoria block RAM y con esto asegurar que la instrucción se ejecuta con el dato correcto en el mismo ciclo de reloj.

IV. Integración de Periféricos

En nuestra propuesta es posible agregar una cantidad variable y configurable de periféricos el procesador RISC-V 32I implementado, para lograr esto, se integro al procesador una interfaz AXI-lite maestra como se muestra en la Figura 4. AMD-Xilinx permite la posibilidad de conectar múltiples esclavos a la interfaz maestra del procesador por medio del Núcleo de Propiedad Intelectual (Intellec-

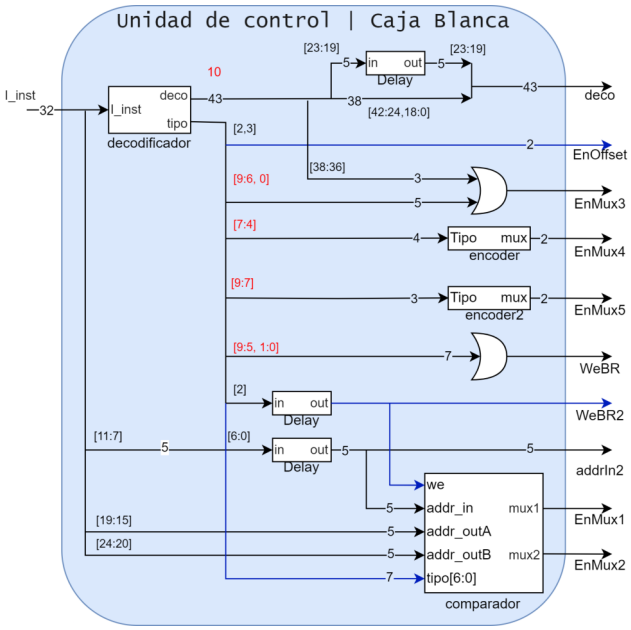

Figura 3: *Diagrama esquemático de la Unidad de Control de nuestra implementación RISC-V 32I*

tual Property Core, IP-Core, en inglés) que provee en su suite de desarrollo.

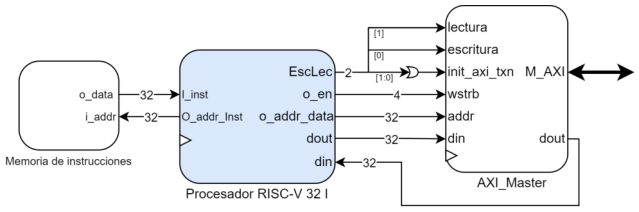

Figura 4: *Procesador RISC-V 32I con interfaz AXI-lite*

Los módulos que pueden ser agregados el sistema son: uno, puertos de entrada de propósito general (GPI); puertos de salida de propósito general (GPO); timers y moduladores de ancho de pulso (PWM). Los puertos GPI y GPO funcionan de manera similar, en la Figura 5 se muestra la forma de operación de estos puertos. En tiempo de diseño puede ser establecidos la cantidad de salidas que tendrá el puerto. Por cada puerto que es agregado se puede establecer las direcciones de memoria que permitirán operar el puerto. Se tiene un registro para habilitar individualmente cada uno de los pines del puerto y además se tiene un registro donde se encuentra el valor que se desea establecer a la salida del pin. Cada puerto de salida es habilitado por medio de una compuerta *and* entre el el bit asociado al pin de salida y el propio dato que e encuentra en el registro. El puerto de entrada funciona de manera similar.

El sistema de timers y PWM se muestra en la Figura 6. La forma de agregar timers/PWM al sistema es similar

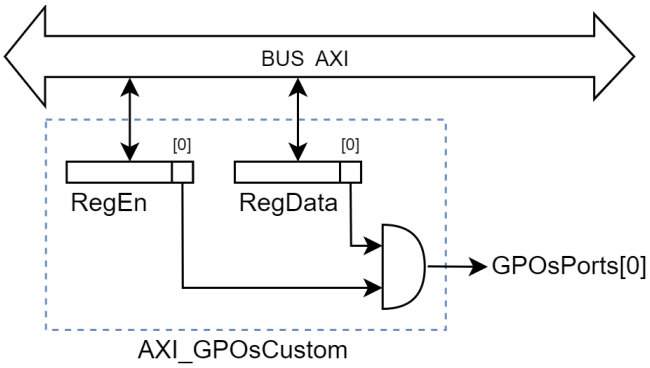

Figura 5: *Puerto de salida controlado por interfaz AXI-lite*

al de los GPI sin embargo, en estos elementos se tiene la ventaja que pueden ser agregadas varias instancias por medio de un solo elemento esclavo AXI-lite. Cada uno de los elementos pueden ser configurados y manipulados individualmente, el estatus de cada elemento es reportado por medio de una bandera que indica en el caso del timer que ha llegado al limite establecido, por lo tanto a transcurrido el tiempo de operación que se estableció, En el caso del PWM esta bandera es utilizada para regular el *duty cycle*.

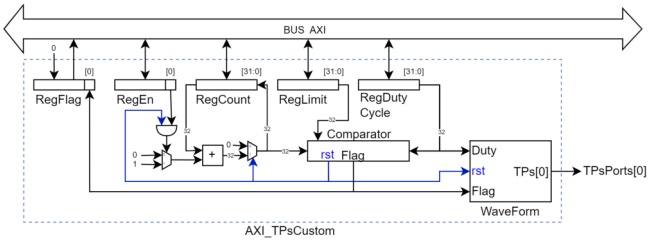

Figura 6: *Timers y PWM controlados por interfaz AXI-lite*

V. Pruebas y Resultados

La implementación del proyecto fue hecha sobre la plataforma de desarrollo del fabricante Digilent NEXYS A7, la cuál consta de un FPGA de Xilinx de la familia Artix-7. Los programas de prueba que se realizaron fueron ensamblados con la herramienta RARS que es un ensamblador y simulador para el ISA RISCV32 y RISCV64.

Además de probar individualmente la ejecución de cada instrucción, se realizó un banco de pruebas en función del programa de uso intensivo de memoria *torres de hanoi* proporcionado por la plataforma [5]. Las implementaciones que fueron probadas es la nuestra en 2 versiones: una, generando la memoria de datos en base a LUTs; dos, generando la memoria de datos con BRAM. Se contrasto contra la implementación realizada por [5] en versión LUT y BRAM, elaborando un recuento del número de ci-

clos de reloj sobre los distintos núcleos como se muestra en la Tabla 1, se puede observar que nuestra implementación BRAM puede mantener el ritmo de ejecución a 1 ciclo por reloj mientras que en [5] esto solo se logra en la implementación con LUTs.

Procesador	UAZv_L	UAZv_B	BRISCv_L	BRISCv_B
No. Ciclos	748	748	748	1688

Tabla 1: *Ciclos de reloj requeridos para ejecutar el programa torres de Hanoi el cual ejecuta 748 instrucciones*

En cuanto a los periféricos se comprobó que se pueden generar múltiples instancias de cada tipo, cada instancia con su propia cantidad de puertos, timers/PWM y que por medio de los mapas de memoria independientes pueden ser fácilmente agregados a un proyecto desarrollado en la plataforma de Vivado por medio del IP Integrator.

VI. Conclusiones

Las principales aportaciones que hemos realizado en el presente trabajo:

1. Nuestra implementación del ISA RISC-V 32I permite la ejecución sobre dispositivos lógicos programables, utilizando como memoria de datos Block RAM, de las instrucciones en un ciclo de reloj que es la característica principal de un procesador monociclo. Esto lo logramos descubriendo si en la siguiente instrucción a ejecutar es necesario utilizar el dato proveniente de memoria por una instrucción de *load*, e interceptándolo y desviándolo a la ALU antes de que sea almacenado en el banco de registros.
2. Con la implementación flexible de puertos, se pueden personalizar los pines de acuerdo a las necesidades específicas de cada proyecto.
3. En el caso de los timers/PWM si tiene la opción de integrar varias entidades de cada uno de ellos logrando con esto solo integrar una interfaz AXI-lite esclava, lo que redunda en el ahorro de recursos.

Una solución similar a la propuesta de múltiples timers/PWM puede ser desarrollada para los GPIOs del sistema y otros periféricos que se agreguen al sistema. Este desarrollo será parte de nuestro trabajo futuro.

Referencias

[1] Waterman A. y Asinovic K. *The RISC-V Instruction Set Manual*. 2rd edition. SiFive Inc. ISBN: 0137027419.

[2] Patterson D. y Waterman A. *Guía Práctica de RISC-V*. 1rd edition. Strawberry Canyon, 2018. ISBN: 978-0-9992491-2-3.

[3] Don Kurian Dennis et al. «Single cycle RISC-V micro architecture processor and its FPGA prototype». En: *2017 7th International Symposium on Embedded Computing and System Design (ISED)*. 2017, págs. 1-5. DOI: 10.1109/ISED.2017.8303926.

[4] Vineet Jain, Abhishek Sharma y Eduardo Augusto Bezerra. «Implementation and Extension of Bit Manipulation Instruction on RISC-V Architecture using FPGA». En: *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*. 2020, págs. 167-172. DOI: 10.1109/CSNT48778.2020.9115759.

[5] Sahan Bandara et al. «BRISC-V: An Open-Source Architecture Design Space Exploration Toolbox». En: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '19. Seaside, CA, USA: Association for Computing Machinery, 2019, pág. 306. ISBN: 9781450361378. DOI: 10.1145/3289602.3293991. URL: https://doi.org/10.1145/3289602.3293991.

[6] Ckristian Duran et al. «A 32-bit RISC-V AXI4-lite bus-based microcontroller with 10-bit SAR ADC». En: *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*. 2016, págs. 315-318. DOI: 10.1109/LASCAS.2016.7451073.

[7] Ckristian Duran et al. «A system-on-chip platform for the internet of things featuring a 32-bit RISC-V based microcontroller». En: *2017 IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS)*. 2017, págs. 1-4. DOI: 10.1109/LASCAS.2017.8126878.

[8] Michael Rogenmoser y Luca Benini. «Trikarenos: A Fault-Tolerant RISC-V-based Microcontroller for CubeSats in 28nm». En: *2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2023, págs. 1-4. DOI: 10.1109/ICECS58634.2023.10382727.

[9] Matthew Johns y Tom J. Kazmierski. «A Minimal RISC-V Vector Processor for Embedded Systems». En: *2020 Forum for Specification and Design Languages (FDL)*. 2020, págs. 1-4. DOI: 10.1109/FDL50818.2020.9232940.

[10] Nolting S. *The NEORV32 RISC-V pROCESSOR*. OpenCores, 2021. ISBN: https://opencores.org/projects/neorv32.