DISEÑO DE PROCESADOR RISC-V DE 32-BITS DE CICLO ÚNICO

DESIGN OF A SINGLE-CYCLE 32-BIT RISC-V PROCESSOR

Marco Antonio Gurrola Navarro
CUCEI – Universidad de Guadalajara, México
marco.gurrola@academicos.udg.mx

Josué Vladimir Quiroga Esparza
CUCEI – Universidad de Guadalajara, México
josue.quiroga4742@alumnos.udg.mx

Álvaro Emmanuel Avelar Huerta
CUCEI – Universidad de Guadalajara, México
alvaro.avelar@alumnos.udg.mx

Carlos Alberto Bonilla Barragán
CUCEI – Universidad de Guadalajara, México
alberto.bonilla@academicos.udg.mx

Iván Rodrigo Padilla Cantoya
CUCEI – Universidad de Guadalajara, México
ivan.padilla@academicos.udg.mx

Agustín Santiago Medina Vázquez
CUCEI – Universidad de Guadalajara, México
agustin.medina@academicos.udg.mx

Recepción: 28/junio/2022 **Aceptación:** 2/agosto/2022

Resumen

En este trabajo se presenta el diseño de una Unidad Central de Procesamiento (CPU) de 32-bits de ciclo único empleando la arquitectura de conjunto de instrucciones RISC-V. Como resultado, se generó un núcleo blando que consta de varios archivos en lenguaje de descripción de hardware VHDL. Esta descripción puede compilarse y cargarse en una tarjeta de desarrollo FPGA para usarse como microcontrolador, o bien, puede sintetizarse hasta obtener un plano fabricable de circuito integrado. Siendo RISC-V una arquitectura de uso libre, una de las motivaciones del presente trabajo es fomentar el uso académico del conjunto de

instrucciones RISC-V para realizar microarquitecturas de CPU, así como servir de apoyo para enseñar los conceptos básicos de Arquitectura de Computadoras.
Palabras Clave: FPGA, microarquitectura, procesador, RISC-V, VHDL.

Abstract

In this work, the design of a single-cycle 32-bit Central Processing Unit (CPU) using the RISC-V instruction set architecture is presented. As a result, a soft-core, consisting of several VHDL hardware description language files, was generated. This description can be compiled and loaded onto an FPGA development board for use as a microcontroller, or it can be synthesized into a fabricable integrated circuit layout. Being RISC-V a free-use architecture, one of the motivations of this work is to promote the academic use of the RISC-V instruction set to create CPU microarchitectures, as well as to support the teaching of the basic concepts of Computer Architecture.
Keywords**:** *FPGA, microarchitecture, processor, RISC-V, VHDL.*

1. Introducción

Para el diseño de un procesador se necesita, en primer lugar, contar con la definición de la Arquitectura de Conjunto de Instrucciones o ISA (*Instruction Set Architecture*). Hasta hace algunos años, para la selección de una ISA, solo se tenían un par de caminos: pagar la licencia de uso de una ISA propietaria (ARM, MIPS, etc) a fin de poder usarla y beneficiarse de toda una cadena existente de herramientas para el desarrollo del procesador y de sus aplicaciones software, o bien, crear una ISA propia que implicaría también un enorme esfuerzo para generar la cadena de herramientas de desarrollo de software.

En la actualidad RISC-V [1] se ofrece como una ISA donde el conjunto de instrucciones es libre y puede ser usado sin el pago de regalías para el propósito que se desee: diseño, fabricación, I+D, etc. A esta ventaja se le añade la existencia de un ecosistema creciente de herramientas de desarrollo de hardware y software relacionadas con esta arquitectura. Los orígenes de RISC-V [2] se remontan al 2010 en la Universidad de California en Berkeley. En ese entonces el profesor Krste

Asanovic, en colaboración con David Patterson, comenzó con este proyecto académico que inicialmente se ofrecía como parte de un curso de verano. Con el paso del tiempo el proyecto creció hasta llegar a ser la organización *RISC-V International* de hoy en día con sede en Suiza.

Actualmente la ISA RISC-V continúa en desarrollo con la participación de ingenieros de compañías privadas en conjunto con académicos de diversas instituciones. Según se muestra en las especificaciones oficiales, RISC-V es en realidad un conjunto de varias ISA. A la más sencilla de éstas se le denomina “base entera RV32I” y se emplea para procesar datos enteros de 32 bits. En la versión de especificaciones 20191213 esta ISA cuenta con 40 instrucciones incluyendo operaciones de memoria, comparaciones, ramificaciones, operaciones aritméticas y lógicas, entre otras [3]. Podemos considerar esta base entera como el conjunto mínimo de instrucciones necesarias para el diseño de procesadores o microcontroladores sencillos.

Además de la base de instrucciones RV32I, las especificaciones de RISCV contienen otros grupos de instrucciones denominados “extensiones” que pueden ser añadidos para trabajar en conjunto con la base entera.RISCV cuenta con extensiones para multiplicación y división (extensión M), punto flotante de precisión simple (extensión F), operaciones vectoriales (extensión V), etc. Con ellas se pueden diseñar procesadores de mayor desempeño a costa de un mayor costo de desarrollo y del uso de mayor cantidad de recursos hardware.

Para poner en contexto, en diseños de microcontroladores clásicos como los de la familia 6800 de Motorola, y concretamente en el M68HC11RM [4], se cuenta con poco más de 100 instrucciones entre las cuales hay operaciones de memoria, aritméticas, multiplicación, división, lógicas, rotaciones, desplazamientos, y de manipulación de bits. Estas son las instrucciones que el usuario debe utilizar para poder generar todos los programas y rutinas que necesite. Por desgracia, este diseño en concreto no cuenta con instrucciones de operaciones con punto flotante. Hay que realizar una adaptación para emular, mediante subrutinas de software, la capacidad para realizar este tipo de cálculos, o bien, emplear un diferente tipo de microcontrolador que sí cuente con el tipo de instrucciones requeridas.

Dentro de la academia y las universidades, el análisis y estudio de microcontroladores, como el mencionado anteriormente, así como de otros procesadores vendidos como componentes discretos, ayudan a entender las características principales y la arquitectura de una computadora: uso de memoria, conjunto de instrucciones, manipulación de datos, etc. Sin embargo, puesto que se trata de circuitos integrados sin capacidad de reconfiguración, éstos no se prestan para la modificación, optimización o el rediseño de la microarquitectura. Esto es cierto incluso para procesadores propietarios ofrecidos como núcleo blando (*soft-core*) tales como el Nios II [7] que, a pesar de implementarse en dispositivos reconfigurables FPGA, no son 5e código abierto.

En cambio, podemos usar las tarjetas de desarrollo FPGA como herramientas de verificación de descripciones hardware de procesadores elaboradas por nosotros mismos, que, aunado a la posibilidad de uso libre del conjunto de instrucciones de RISC-V, nos abre las puertas para aprender los rudimentos del diseño de microarquitecturas de procesador.

Podemos diseñar procesadores sencillos, como el procesador de ciclo único sugerido en el libro de Patterson y Hennessy [6], pero la complejidad puede extenderse hasta el nivel que se desee incorporando una o más extensiones de la ISA RISC-V a nuestro diseño. De esta manera, estudiantes, docentes e investigadores pueden experimentar desde los primeros pasos de la definición de la microarquitectura, la generación de los diferentes módulos básicos que lleva el procesador, la codificación del sistema en algún lenguaje HDL, portar el *soft-core* a una tarjeta FPGA, e incluso utilizar el sistema desarrollado de manera semejante a un microcontrolador comercial. Todo esto con la diferencia de poder manipular la descripción hardware para observar las ventajas y desventajas que implican las diferentes decisiones tomadas durante el proceso de diseño.

2. Métodos

Metodología para el diseño de la microarquitectura

Se describe el método seguido para diseñar una microarquitectura de procesador RISC-V de ciclo único para las instrucciones de la base entera RV32I, que se basa

en las sugerencias de los textos [6] y [7] y en la experiencia previa del autor principal de este trabajo en diseño de microarquitecturas:

- *Paso 1 Elección y estudio de instrucciones:* Para la arquitectura del procesador se eligió la base entera de 32 bits de la ISA RISC-V, llamada RV32I que abarca 40 instrucciones en total [3]. Procedimos a estudiar a detalle las partes correspondientes del documento de especificaciones: capítulo 1, *Introduction*; capítulo 2, *RV32I Base Integer Instruction Set, Version 2.1*; y parte del capítulo 24 que incluye una tabla con la codificación oficial de las instrucciones. En el documento se muestra que las 40 instrucciones de la base entera RV32I están todas codificadas con 32 bits. La arquitectura contiene un registro contador de programa PC (*Program Counter*) de 32 bits que permite manejar un espacio de memoria de hasta 2^{32} = 4 GiB. Contiene 32 registros de 32 bits de propósito general, nombrados x0, x1, …, x31, y conforman el módulo llamado *register file*. Todos estos registros son de lectura y escritura, excepto x0 que se encuentra alambrado de manera permanente al valor de cero. Se prefirió la ISA de 32 bits sobre la de 64 bits, por ser suficiente para aplicaciones sencillas de sistemas embebidos, requerir un esfuerzo de diseño ligeramente menor y por requerir una menor cantidad de recursos hardware en su implementación.
- *Paso 2 Elección entre estilo de diseño de ciclo único, segmentado, o ciclo múltiple:* Para nuestra propuesta de *soft-core* se optó por una implementación de ciclo único (*single cycle*). Esto implica que en la trayectoria de datos o *datapath*, cualquiera de las 40 instrucciones se ejecuta en un solo ciclo de reloj. Este tipo de procesadores tienen su campo de aplicación en sistemas embebidos con requerimientos de cómputo moderados o de bajo consumo de potencia. Ejemplos de implementación de un *datapath* de ciclo único se presentan en [6], para la base entera RV64I, y en [7], para una ISA MIPS de 32 bits. Sin embargo, los *datapath* ahí descritos sólo ejecutan 7 u 8 de las 40 instrucciones de sus respectivas ISA, pues el objetivo en dichos textos es ilustrar técnicas de diseño más que llegar a un *core* completo. En el presente trabajo, en cambio, se muestra cómo lidiar con la complejidad de un diseño que cubre todas las instrucciones de la ISA seleccionada. En tanto que otros

estilos de diseño como el segmentado (*pipeline*), y el estilo de ciclo múltiple (*multi cycle*), nos permiten obtener versiones con desempeño mayor o con menores requerimientos de recursos hardware, respectivamente, pero a costa de un mayor esfuerzo de diseño.

- *Paso 3 Elección entre control alambrado o por microcódigo:* En este trabajo las señales de control se generan con lógica combinacional, estilo que es conocido como unidad de control alambrada en hardware (*hard-wired*). Esta técnica se sigue en Patterson, David A., Hennessy, John L., 2018. Computer organization and design, The hardware software interface, RISC-V edition, Cambridge, United States, Morgan Kaufmann.[4] y [7], y es la técnica usual para microarquitecturas RISC (Reduced Instruction Set Computer). En contraparte en las arquitecturas tipo CISC (Complex Instruction Set Computer), debido a la complejidad requerida en la decodificación de sus instrucciones, es más sencillo diseñar arquitecturas empleando microcódigo, aunque su rendimiento suele ser menor.
- *Paso 4 Elección del criterio principal de diseño: velocidad o reducción de área:* El *soft-core* producido en este trabajo se puede implementar en FPGA o en circuito integrado VLSI. Se decidió que, durante el proceso de diseño, se aplicarían técnicas para reducir el tiempo de retardo de propagación de los módulos y multiplexores del *datapath*. Esto implicó, en algunos casos, emplear estructuras con mayores requerimientos de recursos hardware. Por ejemplo, se emplearon sumadores/restadores de acarreo de adelanto o CLA (Carry Look-Ahead) en lugar de sumadores de acarreo de rizo que consumen menos recursos, pero que son más lentos. En otro ejemplo, se diseñó un multiplexor con 9 datos de entrada de 32 bits. De manera que, sin que implicara un gasto extra de recursos hardware, se cuidó que aquella señal con el retardo más crítico en el resto del *datapath* fuera la que tuviera un tiempo de conmutación menor dentro del multiplexor.
- *Paso 5 Análisis de las instrucciones:* Para el análisis de las instrucciones elaboramos una tabla con ayuda de algún programa de hojas de cálculo. Primero se toman las 40 instrucciones de la base entera seleccionada, RV32I

[3], incluida su codificación binaria, y se organizan en grupos de instrucciones que realizan tareas similares. Por ejemplo, instrucciones tipo Load, Store, Arithmetic, Logic, etc. Después se elige la posición de cada grupo de instrucciones dentro de la tabla, teniendo cuidado de que los códigos binarios entre grupos vecinos compartan algunas porciones de bits que sean iguales o similares. Después de un proceso de prueba y error, en el presente trabajo se llegó al acomodo mostrado en la tabla 1 el cual facilita la deducción de las ecuaciones booleanas de las señales de control.

Tabla 1 Análisis de las 40 instrucciones de la base entera RV32I.

			Instrucciones y su Codificación						Operaciones Requeridas				Multiplexores			
			funct7 7 bits	rs2 5 bits	rs1 5 bits	funct3 3 bits	rd 5 bits	opcode 7 bits	bus inPC	bus inRF	bus Addr	Otras	muxA	muxB	muxinPC	muxinRF
	Instrucción	Descripción	$i_{31}:i_{25}$	$i_{24}:i_{20}$	$i_{19}:i_{15}$	$i_{14}:i_{12}$	$i_{11}:i_{7}$	$i_{6}:i_{2}$								
Load	LB rd,offset(rs1)	Load Byte	[11:5]	[4:0]	rs1	000	rd	00000	PC+4	L	D1+im	load	D1	im	PC4	L
	LH rd,offset(rs1)	Load Halfword	[11:5]	[4:0]	rs1	001	rd	00000	PC+4	L	D1+im	load	D1	im	PC4	L
	LW rd,offset(rs1)	Load Word	[11:5]	[4:0]	rs1	010	rd	00000	PC+4	L	D1+im	load	D1	im	PC4	L
	LBU rd,offset(rs1)	Load Byte Unsigned	[11:5]	[4:0]	rs1	100	rd	00000	PC+4	L	D1+im	load	D1	im	PC4	L
	LHU rd,offset(rs1)	Load Half Unsigned	[11:5]	[4:0]	rs1	101	rd	00000	PC+4	L	D1+im	load	D1	im	PC4	L
S. Store	SB rs2,offset(rs1)	Store Byte	[11:5]	rs2	rs1	000	[4:0]	01000	PC+4		D1+im	store	D1	im	PC4	—
	SH rs2,offset(rs1)	Store Halfword	[11:5]	rs2	rs1	001	[4:0]	01000	PC+4		D1+im	store	D1	im	PC4	—
	SW rs2,offset(rs1)	Store Word	[11:5]	rs2	rs1	010	[4:0]	01000	PC+4		D1+im	store	D1	im	PC4	—
F.	FENCE pred,succ	Synchroniz. Thread	fm Pred	Succ	00000	000	00000	00011	PC+4				-	-	PC4	—
Shift	SLLI rd,rs1,shamt	Shift Left Log. Imm.	0000000	shamt	rs1	001	rd	00100	PC+4	$Shift_{im}$ D1			D1	im	PC4	aluB
	SRLI rd,rs1,shamt	Shift Right Log. Imm.	0000000	shamt	rs1	101	rd	00100	PC+4	$Shift_{im}$ D1			D1	im	PC4	aluB
	SRAI rd,rs1,shamt	Shift Right Arith. Imm.	0100000	shamt	rs1	101	rd	00100	PC+4	$Shift_{im}$ D1			D1	im	PC4	aluB
	SLL rd,rs1,rs2	Shift Left Logical	0000000	rs2	rs1	001	rd	01100	PC+4	$Shift_{D2}$ D1			D1	D2	PC4	aluB
	SRL rd,rs1,rs2	Shift Right Logical	0000000	rs2	rs1	101	rd	01100	PC+4	$Shift_{D2}$ D1			D1	D2	PC4	aluB
	SRA rd,rs1,rs2	Shift Right Arithmetic	0100000	rs2	rs1	101	rd	01100	PC+4	$Shift_{D2}$ D1			D1	D2	PC4	aluB
Arithmet.	ADDI rd,rs1,imm	ADD Immediate	[11:5]	[4:0]	rs1	000	rd	00100	PC+4	D1+im			D1	im	PC4	aluAdSb
	ADD rd,rs1,rs2	ADD	0000000	rs2	rs1	000	rd	01100	PC+4	D1+D2			D1	D2	PC4	aluAdSb
	SUB rd,rs1,rs2	SUBtract	0100000	rs2	rs1	000	rd	01100	PC+4	D1−D2			D1	D2	PC4	aluAdSb
Logic	XORI rd,rs1,imm	XOR Immediate	[11:5]	[4:0]	rs1	100	rd	00100	PC+4	D1 xor im			D1	im	PC4	aluXOR
	ORI rd,rs1,imm	OR Immediate	[11:5]	[4:0]	rs1	110	rd	00100	PC+4	D1 or im			D1	im	PC4	aluOR
	ANDI rd,rs1,imm	AND Immediate	[11:5]	[4:0]	rs1	111	rd	00100	PC+4	D1 and im			D1	im	PC4	aluAND
	XOR rd,rs1,rs2	XOR	0000000	rs2	rs1	100	rd	01100	PC+4	D1 xor D2			D1	D2	PC4	aluXOR
	OR rd,rs1,rs2	OR	0000000	rs2	rs1	110	rd	01100	PC+4	D1 or D2			D1	D2	PC4	aluOR
	AND rd,rs1,rs2	AND	0000000	rs2	rs1	111	rd	01100	PC+4	D1 and D2			D1	D2	PC4	aluAND
Set	SLTI rd,rs1,imm	Set if < Immediate	[11:5]	[4:0]	rs1	010	rd	00100	PC+4	D1−im			D1	im	PC4	aluSet
	SLTIU rd,rs1,imm	Set if < Imm Unsigned	[11:5]	[4:0]	rs1	011	rd	00100	PC+4	D1−im			D1	im	PC4	aluSet
	SLT rd,rs1,rs2	Set if <	0000000	rs2	rs1	010	rd	01100	PC+4	D1−D2			D1	D2	PC4	aluSet
	SLTU rd,rs1,rs2	Set if < Unsigned	0000000	rs2	rs1	011	rd	01100	PC+4	D1−D2			D1	D2	PC4	aluSet
Up.Im.	AUIPC rd,imm	Add Upper Imm to PC	[31:25]	[24:20]	[19:15]	[14:12]	rd	00101	PC+4	PC+im			PC	im	PC4	aluAdSb
	LUI rd,imm	Load Upper Imm	[31:25]	[24:20]	[19:15]	[14:12]	rd	01101	PC+4	im			-	-	PC4	im
Branch	BEQ rs1,rs2,offset	Branch if =	[12|10:5]	rs2	rs1	000	[4:1|11]	11000	PC+4	PC+im		D1=D2?	PC	im	PC4 / aluAdSb	—
	BNE rs1,rs2,offset	Branch if !=	[12|10:5]	rs2	rs1	001	[4:1|11]	11000	PC+4	PC+im		D1=D2?	PC	im	PC4 / aluAdSb	—
	BLT rs1,rs2,offset	Branch if <	[12|10:5]	rs2	rs1	100	[4:1|11]	11000	PC+4	PC+im		D1<D2?	PC	im	PC4 / aluAdSb	—
	BGE rs1,rs2,offset	Branch if >=	[12|10:5]	rs2	rs1	101	[4:1|11]	11000	PC+4	PC+im		D1<D2?	PC	im	PC4 / aluAdSb	—
	BLTU rs1,rs2,offset	Branch if < Unsigned	[12|10:5]	rs2	rs1	110	[4:1|11]	11000	PC+4	PC+im		D1<D2?	PC	im	PC4 / aluAdSb	—
	BGEU rs1,rs2,offset	Branch if >= Unsigned	[12|10:5]	rs2	rs1	111	[4:1|11]	11000	PC+4	PC+im		D1<D2?	PC	im	PC4 / aluAdSb	—
Jump	JALR rd,offset(rs1)	Jump & Link Register	[11:5]	[4:0]	rs1	000	rd	11001	(D1+im) &~1	PC+4			D1	im	aluAdSb	PC4
	JAL rd,offset	Jump & Link	[20|10:5]	[4:1|11]	[19:15]	[14:12]	rd	11011	PC+im	PC+4			PC	im	aluAdSb	PC4
Amb.	ECALL	Call	0000000	00000	00000	000	00000	11100	PCtrap				-	-	4	—
	EBREAK	Break	0000000	00001	00000	000	00000	11100	PCtrap				-	-	4	—

De la tabla 1 se observan las siguientes características en la codificación de las instrucciones:

- ✓ Las instrucciones se codifican con 32 bits que aquí se denotan por $\mathbf{i}_{31}$, $\mathbf{i}_{30}$, $\mathbf{i}_{29}$, …, $\mathbf{i}_{1}$, $\mathbf{i}_{0}$. Pero en la tabla se omiten los bits $\mathbf{i}_{1}$ e $\mathbf{i}_{0}$ porque, en la ISA RV32I, éstos no pueden usarse para decodificar las señales de control debido a que mantienen el mismo valor “11” en todas las 40 instrucciones.
- ✓ La codificación de la instrucción se divide en seis campos: **opcode**, código principal de operación; **rd**, registro destino; **funct3** y **funct7**, auxiliares en la codificación de la operación; **rs1**, registro fuente 1; **rs2**, registro fuente 2.
- ✓ Los campos de instrucción sombreados en color gris incluyen bits de datos inmediatos (*immediate*). Se indican entre corchetes las posiciones que estos bits ocuparán dentro del dato inmediato.

- *Paso 6 Elaboración de microarquitectura:* La información vertida en la tabla 1, en conjunto con las especificaciones de las instrucciones, se utiliza para tomar decisiones sobre las partes que debe llevar la microarquitectura y su estructura de interconexión. La versión terminada de la microarquitectura se muestra en la figura 1 y a continuación se listan las partes principales de su estructura:
  - ✓ El sistema contiene 4 grandes bloques: **Core**, incluye *datapath* y control, y constituye al procesador propiamente dicho; **DataRAM&Ports**, incluye la memoria de datos y los puertos de entrada y salida; **InstructionROM**, incluye la memoria de programa; y **Exceptions**, incluye lógica combinacional para identificar eventos de excepción.
  - ✓ El sistema contiene tres elementos de estado o de memoria: **ProgramCounter**, guarda la dirección de la instrucción a ejecutar; **RegFile**, contiene los 32 registros de propósito general del *register file*; y **DataRAM&Ports**, ya mencionado. Los buses de datos y de control que determinan el estado siguiente de estos elementos de memoria se muestran en color gris.

- ✓ En la figura 1, rellenos en color gris oscuro, se muestran siete módulos que realizan operaciones aritméticas, lógicas o de reacomodo de datos.

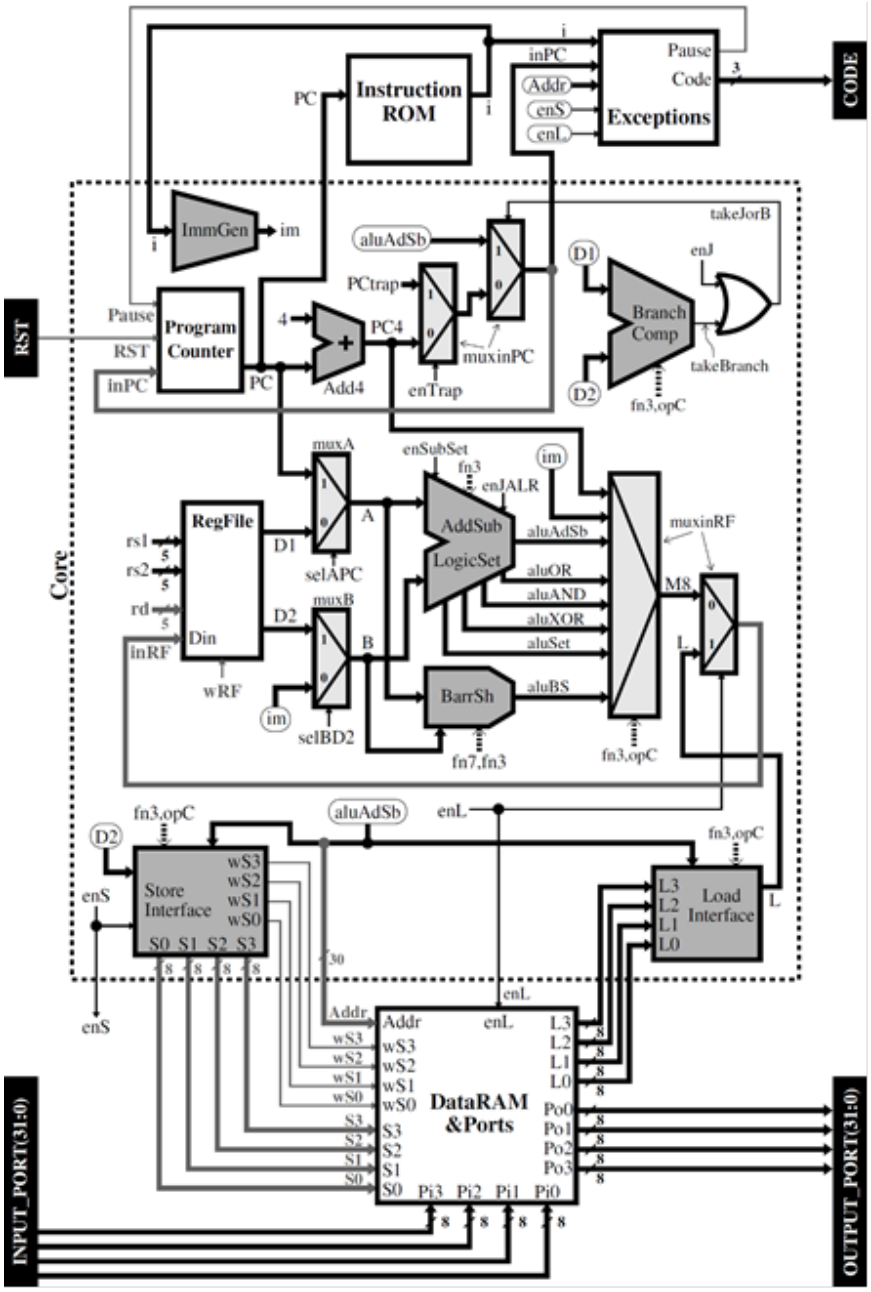

Figura 1 Microarquitectura del *soft-core* propuesto.

Rellenos en color gris claro, se muestran los multiplexores del *datapath*. Los buses se indican con línea continua gruesa y tienen un ancho de 32 bits a menos que se especifique otra cosa. Las líneas delgadas

representan señales de control individuales. Las líneas anchas punteadas denotan que algunos de los bits de los campos **funct7**, **funct3** y/o **opcode** (indicados como **fn7**, **fn3**, **opC**) se emplean para producir las señales de control del módulo al que entran.

Metodología para la codificación RTL del *soft-core*

Para la codificación del *soft-core* empleamos el lenguaje de descripción de hardware VHDL. Para la comprobación de sintaxis, simulación digital y para la síntesis (compilación) de este tipo de archivos empleamos las herramientas libres *Alliance VLSI CAD System* [8] [9] [10] [11].

La descripción de los módulos **Core**, **DataRAM&Ports**, **InstructionROM** y **Exceptions** mencionados en la sección previa, se capturó en sendos archivos empleando lenguaje VHDL de estilo comportamental. A este tipo de archivos se les conoce como descripción RTL (*Register Transfer Level*). Adicionalmente, se capturó otro archivo en VHDL, en este caso de estilo estructural, para codificar un quinto módulo de nivel jerárquico superior, **RVsystem**, que contiene e interconecta a los 4 módulos mencionados. Estos 5 módulos engloban la totalidad de la microarquitectura mostrada en la figura 1.

Con el empleo de las herramientas de síntesis y verificación de Alliance, los 5 módulos originales se sintetizaron hasta el punto en que el sistema se encontraba descrito en un único archivo en VHDL estructural compuesto por celdas digitales básicas, denominadas celdas estándar. Esta descripción estructural puede llevarse hasta la forma de un plano de circuito integrado fabricable empleando la metodología de posicionado e interconectado P&R (Place and Route) de las herramientas de Alliance. Por otra parte, éste mismo diseño se puede tomar para cargarse en una tarjeta de desarrollo FPGA con la ventaja de que los costos (el precio de una tarjeta de desarrollo FPGA) y los tiempos de implementación son mucho menores comparados con el costo y tiempo de fabricación del sistema en circuito integrado.

El *soft-core* de este trabajo se emuló en un FPGA Intel MAX 10 10M50DAF484C7G incluido en la tarjeta de desarrollo Terasic DE10-Lite, con un costo a la fecha de

USD$ 77 [12]. Se implementó un pequeño procedimiento para tomar las descripciones en VHDL estructural a nivel de celdas estándar producidas por Alliance y portarlas al software propietario Intel Quartus Prime, versión gratuita Lite Edition v18.1 [13]. Finalmente, se siguió el procedimiento indicado en el manual de la tarjeta para cargar el *soft-core* diseñado en el FPGA

Metodología para cargar programas escritos en lenguaje en ensamblador

La emulación del *soft-core* en la tarjeta de desarrollo FPGA permite tanto corroborar su correcto funcionamiento en tiempo real, como realizar aplicaciones embebidas completamente funcionales. Pero esto requiere de un procedimiento para preparar programas en lenguaje máquina y ejecutarlos en el *soft-core*.

Primero se escriben los programas empleando los mnemónicos del lenguaje ensamblador de RISC-V. Luego, se ensamblan estos programas con el software ensamblador “rv”, usado en [14], que genera un archivo de extensión *.bst, cuyo contenido es el programa ejecutable en código máquina. Con el contenido de este archivo de texto se crea la descripción VHDL del módulo **InstructionROM**. Después, el sistema completo, mostrado en la figura 1, se compila con ayuda de las herramientas de Alliance hasta obtener su descripción estructural en un solo archivo, el cual se porta a la herramienta Intel Quartus Prime Lite Edition v18.1. Finalmente, el sistema se carga en la tarjeta de desarrollo FPGA seleccionada y se verifica la correcta ejecución del programa en tiempo real.

Cabe mencionar que si se detecta algún mal funcionamiento durante la emulación FPGA que sea atribuible a la descripción VHDL de alguno de los 4 bloques del sistema, ésta se puede depurar empleando el simulador Asimut. Si, por el contrario, se sospecha que el error es debido al programa en lenguaje ensamblador, este se puede depurar con el simulador incluido en el programa “rv”.

3. Resultados

Pruebas iniciales de la descripción RTL

Para las pruebas iniciales, se prepararon varias versiones del módulo **InstructionROM** correspondientes a programas en lenguaje máquina de menos de

diez instrucciones. La ejecución de estos programas se simuló con Asimut, el simulador digital del conjunto de herramientas Alliance [8] [9]. Debido a su pequeña extensión, estos primeros programas se escribieron a mano directamente en código binario.

La idea era observar que en cada ciclo de reloj el sistema realizara el incremento del PC, la lectura correcta de la **InstructionROM**, entre otras funcionalidades básicas. Con estos intentos iniciales para poner en marcha el procesador se lograron depurar algunas fallas burdas del sistema como: contador de programa sin avanzar, o cálculo incorrecto de las direcciones para acceso a memoria. El origen de estas fallas fue: errores tipográficos o algunas líneas faltantes en el código RTL; negación faltante en alguna variable; y algunos errores de conexiones en el código VHDL estructural del archivo **RVsystem**.

Verificación del conjunto completo de instrucciones

Se escribió un programa en lenguaje ensamblador RISC-V y se guardó con el nombre **instr40.a**. El programa contiene 255 instrucciones en total que cubren las 40 instrucciones de la base entera RV32I. Las instrucciones tipo Set y Branch se ejecutan por lo menos dos veces para cubrir los casos donde se cumple y donde no se cumple la condición respectiva. La porción de código para probar cada instrucción se acompañó de instrucciones de apoyo adicionales, la última de las cuales siempre fue **sw** que envía el resultado de la prueba al puerto de salida.

El contenido de los 32 bits del puerto de salida se hace visible con ayuda de los *display* de 7 segmentos y de los LED que incluye la tarjeta de desarrollo. El programa hace una pausa para poder verificar el resultado de manera visual. La ejecución continúa al presionar un interruptor establecido para tal fin. La pausa se puede implementar con un módulo que detecte la instrucción sw y que active la señal **Pausa** (Figura 1), o mediante el llamado a una subrutina. Por brevedad no se ofrecen más detalles al respecto.

El programa incluye dos ciclos anidados para checar toda la memoria de datos: el primer ciclo guarda datos diferentes en cada byte de memoria; luego, el segundo ciclo lee y suma todos los datos guardados y envía el resultado al puerto de salida.

En la figura 2 se muestra el sistema durante pausas de ejecución. Como se puede observar, la tarjeta de desarrollo cuenta con dos interruptores de presión (*push-button*) y con 10 interruptores de dos posiciones. Uno de los *push-button* se emplea para activar la señal **RST** del sistema y el otro para salir de las pausas (señal **Next** en la figura 2). Ocho de los interruptores se emplearon para introducir datos, de byte en byte, durante las operaciones de lectura del puerto de entrada.

Tras depurar un puñado de errores, el sistema cargado en el FPGA pudo completar la ejecución del programa **instr40.a** de manera correcta.

Figura 2 Ejecución de programa instr40.a.

Ejemplo de aplicación: reloj digital

Se preparó un programa en lenguaje ensamblador que usa al *soft-core* cargado en la tarjeta FPGA para funcionar como un reloj digital. El reloj cuenta con las

siguientes características: muestra horas, minutos y segundos en 6 *displays* de 7 segmentos; emplea 7 LEDs para indicar el día de la semana; emplea 2 interruptores para variar la velocidad de avance con el propósito de facilitar la verificación del sistema; emplea 7 interruptores para programar la hora y los minutos de la alarma; si la hora actual coincide con la hora de alarma se enciende un LED; se emplea un *push-button* para apagar el LED de alarma. En la figura 3 se muestran algunas imágenes de las pruebas de emulación en FPGA, durante las cuales no se detectaron errores.

Figura 3 Aplicación de ejemplo: reloj digital.

4. Discusión

En la sección 2 se bosqueja el procedimiento seguido para el diseño de una microarquitectura que ejecute el conjunto de instrucciones seleccionado, RV32I. Este procedimiento es válido para arquitecturas de ciclo único, incluso cuando se añaden instrucciones de extensiones de la ISA RISC-V. En cambio, si lo que se desea es obtener un procesador *pipeline*, el procedimiento se debe modificar para llevar a cabo análisis adicionales para prevenir los riesgos de datos y riesgos de control. Por otro lado, si no se requiere de tanta velocidad de procesamiento, puede modificarse el diseño presentado para obtener un procesador de ciclos múltiples que implique un ahorro de recursos en ciertos bloques de hardware.

En la sección 3 se presentaron los resultados de la implementación de la microarquitectura diseñada en una tarjeta de desarrollo FPGA que corre con una frecuencia de señal de reloj de 50 MHz. Se elaboraron varios programas de prueba y una pequeña aplicación, y tras la depuración de una pequeña cantidad de errores, se demostró la correcta operación del procesador. Con esto podemos ver que la microarquitectura presentada puede bien ser usada como un microcontrolador, ya sea a través de su implementación en una tarjeta FPGA, o bien, ir más allá de lo logrado en el presente trabajo, e implementar la microarquitectura en circuito integrado.

Es de destacar que el esfuerzo de diseño de la microarquitectura aquí presentada fue mucho menor si se compara con implementaciones de procesadores de tamaño similar, pero de arquitectura tipo CISC. Esto se debe por un lado a que la cantidad de instrucciones en RISC-V es baja, obviamente por tratarse de una arquitectura RISC, pero también se debe a que el objetivo seguido por los diseñadores de esta ISA durante la selección de las instrucciones y de los códigos de instrucción tenía precisamente como uno de sus objetivos el simplificar el diseño de las microarquitecturas RISC-V. Un segundo objetivo de los diseñadores de la ISA era lograr procesadores económicos o ligeros en términos de hardware para que puedan alcanzar un elevado desempeño. Una forma de lograr esto, ha sido estableciendo códigos de instrucción que no son tan compactos, de manera que el esfuerzo de decodificación se mantenga bajo.

Hay que hacer notar que el campo de aplicación de procesadores es muy amplio, y que el nicho potencial de explotación de los procesadores como el descrito en el presente trabajo es en sistemas embebidos de complejidad baja o media incluyendo dispositivos de internet de las cosas.

Por otro lado, queremos mencionar algunas experiencias de trabajo con algunas ISA libres en diferentes instituciones académicas, lo cual les está permitiendo que estudiantes y académicos incursionen en el desarrollo de sus propias microarquitecturas. La primera es el proyecto preDRAC SoC [15] desarrollado por Barcelona Supercomputing Center (BSC), Centro de Investigación en Computación del Instituto Politécnico Nacional en México (CIC-IPN), Instituto de Microelectrónica

de Barcelona del Centro Nacional de Microelectrónica (IMB-CNM CSIC), y la Universitat Politècnica de Catalunya (UPC). Utiliza el *core* denominado Lagarto diseñado en el CIC-IPN de México. Para este diseño de este, se usó la ISA MIPS32 y después fue portado a RISC-V resultando en un *core pipeline* en-orden de 64-bit con las extensiones de multiplicación y división (M) e instrucciones atómicas (A). Se está llevando otro importante proyecto de un consorcio de red de universidades, centros de investigación y empresas europeas en donde participa también el BSC, el proyecto "European Processor Iniciative (EPI)" [16] [17] donde el cual se está desarrollando el procesador basado en RISC-V. En este caso el proyecto es considerado en la comunidad europea como un asunto de seguridad nacional y de soberanía tecnológica. Por otra parte, el Centro de Ingeniería y Desarrollo Industrial (CIDESI), en Querétaro, México, reporta el diseño y fabricación de un microprocesador de 24 bits propósito específico que ejecuta el algoritmo Kalman para aplicaciones en sistemas de navegación inerciales [18]. También tenemos la iniciativa *Jalisco on Chip* promovida por el Centro de Investigación y Estudios Avanzados (CINVESTAV) unidad Guadalajara [19], uno de cuyos principales objetivos es contar con un procesador de diseño propio basado en la ISA RISC-V. Por último, mencionar la experiencia del autor principal de este trabajo quién ha diseñado o asesorado en la Universidad de Guadalajara y en otras instituciones académicas el diseño de varias microarquitecturas de procesador, todas ellas empleando herramientas de síntesis VLSI de uso libre.

Estas experiencias nos permiten ver que: el desarrollo de microprocesadores es un asunto de soberanía tecnológica; que las ISA de uso libre, y en particular RISC-V, abren la puerta para realizar implementaciones de microarquitecturas exentas de pagos de regalías por el uso de la ISA; que los costos de llevar la fabricación en silicio de los diseños obtenidos puede verse reducido con el uso de licencias académicas de software propietario o el uso de software libre; que las plataformas de desarrollo en FPGA pueden servir como un excelente medio de verificación de la microarquitectura diseñada; y que, en suma, las universidades pueden ser capaces de crear conocimiento e investigaciones de alto impacto con costos moderados y además generar capital humano de alto nivel técnico.

5. Conclusiones

Se presentó una metodología práctica para el diseño de microarquitecturas RISC-V de ciclo único, que se basa en el análisis de las especificaciones oficiales. Los resultados del análisis se van vertiendo en una tabla. Esta tabla es el eje central del proceso de diseño que tiene como resultado final el diagrama esquemático del *datapath*, incluyendo todos sus módulos, buses de datos y señales de control.

La metodología presentada se probó exitosamente en el desarrollo de un procesador RISC-V de 32 bits. Una vez obtenida la microarquitectura se procedió a su codificación en VHDL para obtener un *soft-core*. Éste último puede ser compilado para la obtención de un plano de circuito integrado fabricable o para su emulación en FPGA.

Se desarrolló un ejemplo de aplicación, donde el sistema emulado en FPGA fue capaz de ejecutar correctamente un programa, enviar señales de actuación por sus puertos de salida, y recibir señales por sus puertos de entrada. Con esto se demostró la aplicabilidad del *soft-core* para ser usado como microcontrolador en el desarrollo de aplicaciones embebidas.

El presente trabajo puede usarse de dos maneras: emplear la microarquitectura presentada para ser implementada en FPGA o VLSI y poder usarse como microcontrolador en sistemas embebidos, o bien, basarse en la metodología aquí presentada para diseñar su propia microarquitectura.

6. Bibliografía y Referencias

[1] RISC-V International, (2021). Specifications: https://riscv.org/technical/ specifications/.

[2] RISC-V International, (2021). History of RISC-V: https://riscv.org/about/ history/.

[3] RISC-V Foundation, (2019). The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version: 20191213: https://github.com /riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20 191213.pdf.

[4] F. Semiconductor, (2007). M68HC11 Reference Manual, Semiconductor, Freescale: http://www.nxp.com/docs/en/reference-manual/M68HC11RM.pdf.

[5] Chu, P. P., 2011. Embedded SOPC design with NIOS II processor and VHDL examples. John Wiley & Sons.

[6] Patterson, David A., Hennessy, John L., 2018. Computer organization and design, The hardware software interface, RISC-V edition, Cambridge, United States, Morgan Kaufmann.

[7] Harris, David Money, Harris, Sarah L., 2013, Digital Design and Computer Architecture, Second Edition, Croydon, United Kindom, Morgan Kaufmann.

[8] Laboratoire d'Informatique de Paris 6, Systèmes Embarqués sur Puce (SoC), Alliance A Free VLSI CAD System, Sorbonne Université: https://www-soc.lip6.fr/equipe-cian/logiciels/alliance/.

[9] Laboratoire d'Informatique de Paris 6, Coriolis VLSI CAD Tools, Alliance VLSI/CAD System, Sorbonne Université, http://coriolis.lip6.fr/pages/alliance

[10] Chávez-Bracamontes, Ramón, Gurrola-Navarro, Marco Antonio, García-López, Reyna Itzel, Bandala-Sánchez, Manuel, “VLSI Design with Alliance Free CAD Tools: an Implementation Example”, Ingeniería Investigación y Tecnología, volumen 16, número 3, julio 2015, pag. 441-452: https://doi.org/10.1016/j.riit.2015.05.007

[11] Medina-Vázquez, Agustín Santiago, Gurrola-Navarro, Marco Antonio, Flores-Castillo, Pablo David, Meda-Campaña, María Elena, Bonilla-Barragán, Carlos Alberto, Villegas-González, José Martín, “Metodología de bajo costo para implementar circuitos electrónicos integrados, un ejemplo de aplicación”, Ingeniería Investigación y Tecnología, volumen 20, número 3, julio 2019, pag. 1-11, https://doi.org/10.22201/fi.25940732e.2019.20n3.029

[12] Terasic, DE10-Lite Board: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=218&No=1021.

[13] Intel Corporation, Quartus Prime Lite Edition v18.1: https://fpgasoftware.intel.com/18.1/?edition=lite.

[14] Dos Reis, Anthony J., 2019. RISC-V Assembly Language, Copell, USA, Independently Published.

[15] European Processor Initiative. Project: https://www.european-processor-initiative.eu/project/epi/.

[16] Abella, J., Bulla, C., Cabo, G., et Al, 2020. An academic risc-v silicon implementation based on open-source components. In 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS) (pp. 1-6). IEEE.

[17] Kovač, M., et Al, 2019. European processor initiative: the industrial cornerstone of EuroHPC for exascale era. In Proceedings of the 16th ACM International Conference on Computing Frontiers, pp. 319-319.

[18] Ramón Chávez-Bracamontes, Marco A. Gurrola-Navarro, Humberto J. Jiménez-Flores, Manuel Bandala-Sánchez, "VLSI architecture of a Kalman filter optimized for real-time applications", IEICE Electronics Express, Volume 13 (2016), Issue 6, pp. 20160043.

[19] Iniciativa Jalisco on Chip, 2021. Presentación rueda de prensa: https://www.youtube.com/watch?v=aNWk5zWTMD8&ab_channel=GobiernodeJalisco.