Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Caracterización y Comparación de las Arquitecturas de Unidades Centrales de Procesamiento Generadas en Rocketchip

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Edgar Solera Bolaños

Versión de 4 de diciembre de 2017

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado Caracterización y Comparación de las Arquitecturas de Unidades Centrales de Procesamiento Generadas en Rocketchip, realizado por Sr. Edgar Solera Bolaños y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

M.sc.Ing. Ronny García Ramírez

Profesor lector

Ing. Carlos Mauricio Segura Quiros

Profesor lector

M.sc Ing. Roberto Molina Robles

Profesor asesor

Cartago, 30 noviembre , 2017

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Edgar Solera Bolaños

Cartago, 3 de diciembre de 2017

Céd: 1-1584-0044

Resumen

En este documento se reporta la medición del poder de procesamiento de las arquitecturas Rocket generados por medio de la herramienta Rocketchip. La caracterización se llevo acabo por medio de los simuladores de VCS y Verilator, y por medio de una implementación practica en una Zedboard midiendo los ciclos de reloj e instrucciones contados en la ejeucución de programas escritos en C. Para medir la eficacia de las instrucciones agregadas por las extensiones se calcularon los CPI en los diferentes Benchmarks, para relacionar los ciclos promedio que necesita cada instrucción. Los programas utilizados fueron un suit de benchamrks RISC-V (encontrado en el repertorio de herramientas de Rocketchip), un suit aplicado al procesador MSP430 (encontrado en un documento que se encuentra en la bibliografía) y el programa Dhrystone. El suit de RISCV fue usado para medir el rendimiento en sumas, operaciones en memoria y la ejecución de multiplicación por software. El suit del MSP430 es usado para la comparación con este mismo procesador y medir su rendimiento en operaciones matemáticas, filtros, máquinas de estado y memoria.También se compararon los resultados prácticos con los obtenidos en los simuladores calculando el error. El programa dhrystone sirvió para comparar el rendimiento de los procesadores Rocket implementados con otras encontrados en la literatura con características similares. Por medio de un reporte de potencia hecho en vivado se obtuvo una estimación del consumo de potencia dinámica.

Palabras clave: Rocketchip, Benchmark, RISC-V, Poder de Procesamiento

Abstract

This paper reports the measurement power of the Rocket architectures generated by the Rocketchip tool. The characterization was carried out by the use of VCS and Verilator simulators and by the practical implementation of a Zedboard, measuring the clock cycles and instructions counted in the execution of programs written in C language. To measure the efficiency of the instructions added by the extensions, the CPI was calculated in the different Benchmarks to obtain the average cycles that take each instruction. The programs used were a set of RISC-V benchmarks (found in the Rocketchip tool repertoire), a suit applied to the MSP430 processor (found in the bibliography) and the Dhrystone program. The RISC-V suit was used to measure the performance of the additions, operations in memory and execution of multiplication by software. The suit of the MSP430 is used to compare the measure the performance of mathematical operations, filters, state machines and memory with this processor. Practical results were also compared with simulator's results. The Dhrystone program served to compare the performance of Rocket processors implemented with others found in the literature with similar characteristics. By a power report made on Vivado, it was estimated the dynamic power consumption.

Keywords: Processing Performance, Rocketchip, RISC-V, Benchmark

a mis queridos padres

Agradecimientos

El resultado de este trabajo no hubiese sido posible sin el apoyo de mi novia, familia, amigos y profesores.

Edgar Solera Bolaños

Cartago, 4 de diciembre de 2017

Índice general

Índice de figuras iii

Lista de símbolos y abreviaciones v

1 Introducción 1
- 1.1 Objetivos y estructura del documento . . . 3
  - 1.1.1 Objetivo General . . . 3
  - 1.1.2 Objetivos Específicos . . . 3

2 Marco Teórico 5
- 2.1 Velocidad de Procesadores . . . 5
  - 2.1.1 Tiempo de Ejecución y Velocidad de Procesamiento . . . 5
  - 2.1.2 Benchmarks . . . 6
  - 2.1.3 Amdahl's Law . . . 7
  - 2.1.4 Tiempo de CPU y Ciclos de Reloj por Instrucción . . . 8
  - 2.1.5 MIPS . . . 9
  - 2.1.6 Dhrystone benchmark . . . 9
- 2.2 Potencia . . . 10
  - 2.2.1 Potencias Dinámica . . . 10
  - 2.2.2 Consumo Estático . . . 11
- 2.3 RISC-V . . . 11
  - 2.3.1 Arquitectura RISC-V . . . 11
  - 2.3.2 Rocketchip y Toolchain . . . 12
  - 2.3.3 Compilador(-march y -mabi) . . . 12
  - 2.3.4 Timers and counters . . . 14

3 Caracterización de las Arquitecturas RISC-V 15
- 3.1 MSP430 Benchmark Suit . . . 15
- 3.2 RISC-V Benchmark Suit . . . 16
- 3.3 Dhrystone . . . 17
- 3.4 Función de medición y unidad de velocidad . . . 18
- 3.5 Construcción del compilador y de Rocket . . . 18
- 3.6 Simuladores : Emulador de Chisel y VCS . . . 19
- 3.7 Implementación FPGA . . . 20
- 3.8 Presentación de los Resultados . . . 21

4 Resultados y Análisis **23**
4.1 Benchmarking provisto por Rocketchip . . . 23
4.1.1 Emulador de Chisel Verilator . . . 23
4.1.2 VCS . . . 25
4.1.3 Implementación Zedboard . . . 26
4.2 MSP430 Benchmarking . . . 28
4.2.1 VCS . . . 28
4.2.2 Implementación Zedboard . . . 32
4.3 Dhrystone . . . 33
4.3.1 VCS . . . 34
4.3.2 Implementación Zedboard . . . 35
4.4 Reporte de Potencia Obtenido en Vivado . . . 37

5 Conclusiones **39**

Bibliografía **41**

A Resultados del MSP430 benchmark suit en Verilator **43**

B Suit de RISC-V **47**

C Dhrystone **51**

Índice de figuras

2.1 Código de Multiplicación de Dobles Ejemplo 13
2.2 Compilación de la Multiplicación Usando march Primer Caso 13
2.3 Compilación de la Multiplicación Usando march Segundo Caso 13

4.1 Ciclos por Instrucción del Benchmarking proveído por Rocketchip Simulado en Verilator 24
4.2 Instrucciones Contadas en el Benchmarking proveído por Rocketchip Simulado en Verilator 24
4.3 Ciclos de Reloj Contados del Benchmarking proveído por Rocketchip Simulado en VCS 25
4.4 Media Aritmética de los Resultados del Benchmarking proveído por Rocketchip Simulado en VCS 26
4.5 Ciclos de Reloj Contados del Benchmarking proveído por Rocketchip en la Zedboard 26
4.6 Instrucciones Contadas del Benchmarking proveído por Rockerchip en la Zedboard 27
4.7 Error de los Simuladores en el Benchmarking proveído por Rocketchip 27
4.8 Ciclos Contados en la ejecución Switch Case con Datos de 8 y 16 bits Simulado en VCS 29
4.9 Ciclos Contados en el Benchmark Switch Matrix con Datos de 8 y 16 bits simulado en VCS 29
4.10 Ciclos y Instrucciones Contadas en la Ejecución del Filtro FIR Simulado en VCS 30
4.11 Ciclos y Instrucciones Contadas en la Ejecución de la Matemática de Coma Flotante simulado en VCS 30
4.12 CC y IC del Programa de Multiplicación de Matrices en la simulación de VCS 31
4.13 Resultados del Benchmarking MSP430 Resumidos con la Media Aritmética simulado en VCS 32
4.14 Resultados del Benchmarking MSP430 Resumidos con la Media Aritmética Simulado en Zedboard 32
4.15 Errores de los Simuladores en el Suit del MSP430 33
4.16 Ciclos y Instrucciones Contadas en Dhrystone Simulado en VCS 34
4.17 DMIPS en Dhrystone Simulado en VCS 34
4.18 Comparación de la Capacidad de DMIPS/MHz con Diferentes Procesadores 35

4.19 Ciclos y Instrucciones Contadas del Benchmark en la Zedboard 36
4.20 Error de los Simuladores sobre el Benchmark Dhrystone 36
4.21 Error de los Simuladores sobre el Benchmark Dhrystone 37

A.1 Ciclos de Reloj Contadas en Switch Case Simulado en Verilator 43
A.2 Ciclos de Reloj Contadas en Switch Matrix Simulado en Verilator 44
A.3 Instrucciones y Ciclos Contadas en el Filtro FIR Simulado en Verilator . . . 44
A.4 Instrucciones y Ciclos Contadas en Floating Point Simulado en Verilator . . 45
A.5 Media aritmética del suit MSP430 Simulado en Verilator 45
A.6 Instrucciones y Ciclos Contadas en Matrix Mult Simulado en Verilator . . . 46

B.1 CPI en el Suit de RISC-V Simulado en VCS 47
B.2 IC en el Suit de RISC-V Simulado en VCS 48
B.3 CC en el Suit de RISC-V Simulado en VCS 48
B.4 Media Aritmética en el Suit de RISC-V Simulado en VCS 49

C.1 IC y CC del Benchmark Dhrystone en Verilator 51
C.2 DMIPS del Benchmark Dhrystone en Verilator 52
C.3 DMIPS/MHz del Benchmark Dhrystone en Verilator 52

Lista de símbolos y abreviaciones

Abreviaciones

ABI	Application binary interface
CC	Cycles Count
CPI	Cycles per Instruction
DMIPS	Dhrystone MIPS per second
FMA	Fused Multiply-add
FP	Floating Point
FPMM	Floating-Point Matrix Multiplication
FPU	Floating point Unit
GFLOPS	Giga Floating Operation per Second
GOPS	Giga operation per second
IC	Instruction Count
IEEE	Institute of Electrical and Electronics Engineers
IPC	Instruction per clock
ISA	Instruction set Architecture
MMU	Mapping Memory Unit
PCA	Análisis de componentes principales
SoC	System on Chip
TI	Texas Instruments
ULP	Ultra Low Power

Capítulo 1

Introducción

La microelectrónica y su relación con la industria de dispositivos médicos han dado origen a los implantables médicos; los cuales son todo aquel dispositivo electrónico capaz de realizar una función de medición o estimulación dentro del cuerpo humano con el fin de monitorear o dar tratamiento a un a padecimiento[22]. Estos dispositivos se componen generalmente de sensores, estimuladores, sistemas de alimentación, un control digital y un sistema de comunicación.

En la comunidad ingenieril existe un fuerte interés en la creación y mejoramiento de implantables. Basta con observar revistas de la IEEE que publican papers en base a estos temas como "Transaction on Biomedical Circuits and Systems" o "Transaction on Biomedical Engineering". Algunos de estos artículos ya se encuentran en el mercado como los marcapasos y otros continúan en investigación como los detectores de convoluciones. En ambos casos se observa un fuerte interés en mejorar e innovar estos dispositivos buscando disminuir su tamaño y consumo de potencia, aumentado su periodo de vida y alcance en el tratamiento de enfermedades. Algunos de ellos en fase de prototipo han sido probados con éxitos en animales; ya que es un largo camino el que deben recorrer para poder ser usado y probado en humanos. Cualquier innovación o avance en el diseño de un implantable que mejore alguna de sus características tiene un importante impacto en la investigación que se encuentra y en la salud de los pacientes que lo necesiten.

Algunos de estos dispositivos tienen un control digital que lee los periféricos, hace los cálculos necesarios con la información recolectada y generan una respuesta adecuada en las salidas del sistema. Se busca que este control digital ocupe el menor espacio físico posible manteniendo un consumo energético ultra bajo y poder computacional aceptable para la función que realiza. Un consumo ultra bajo de potencia se caracteriza por que los transistores operan en la zona sub-umbral, con el voltaje más bajo de activación posible. Esto tiene un gran efecto en el ahorro energía disminuyendo cuadráticamente el consumo dinámico como se observará en la ecuación 2.16; sin embargo repercute negativamente en la velocidad de procesamiento. Algunas técnicas para un diseño ultra-bajo consumo se observa en [16]. Un consumo ultra bajo prolonga la vida de la batería que comúnmente alimenta al sistema, un espacio físico reducido da versatilidad en la ubicación que tendrá dentro del cuerpo y un po-

der computacional alto permite al sistema de responder de manera pronta ante los estímulos recibidos. Los microprocesadores comerciales y hechos a la medida con propiedades de ultra bajo consumo son opciones para CPU de implantables, sin embargo ambos tienen desventajas. En el caso de los microprocesadores comerciales son productos hechos donde no todos sus recursos disponibles son requeridos por las aplicación desperdiciando área y energía. Los procesadores diseñados a la medida son construidos por el propio equipo de investigación del implantable y cumplen con todos los requisitos del implantable sin embargo, se incurre en una perdida de recursos y tiempo en diseñar un procesador adecuado a sus necesidades. Además los requisitos de un implantable van cambiando conforme sus proyectos de investigación avanzan necesitando muchas veces de más capacidad computacional, más memoria, más periféricos lo cual hace necesario que se opte por un procesador adecuado a los nuevos requerimientos. La opción comercial afecta el desempeño del implantable desperdiciando espacio y aumentando el consumo del implantable y la opción diseñada a la medida incurre en costos de tiempo para su modernización y adaptación a la evolución del implantable . De momento en el mercado una de las opciones de CPU usadas para aplicaciones médicas con características de bajo consumo es el MSP430 como se muestra en [23]. Otra opción son los procesadores Rocket los cuales cuentan con la herramienta rocketchip que permite parametrizar su diseño.

Los procesadores Rocket son otra opción en el mercado que por medio de su herramienta Rocketchip permiten parametrizar algunas características como el número de etapas de pipeline, espacio de memoria y las extensiones RISC-V en ellas. Esto permite a los usuarios crear un diseño de procesador parametrizado por ellos con las características deseadas para la aplicación. Sin embargo su herramienta no tiene documentación y es difícil de usar; además que su procesadores no pueden parametrizar característica de ultra bajo consumo, lo cual es un área de interés en implantables médicos. En otras palabras la herramienta no esta hecha para parametrizar variables del procesador de interés en implantables médicos.

El Laboratorio de Diseño de Circuitos Integrados (DCILAB) ha tomado como reto construir un procesador de ultra bajo consumo con una herramienta que permita parametrizar su diseño (como el Rocket), pero con uso enfocado en implantables médicos. Dicho proyecto recién inicia y se encuentra en etapa de investigación del estado del arte en temas de procesadores ultra bajo consumo y parametrizables. Se ha elegido una arquitectura de instrucciones RISC-V de Berkley para el proyecto, ya que la misma es de lenguaje abierto. De momento se busca obtener un marco de referencia para poder comparar los resultados del procesador a diseñar con otras arquitecturas que se encuentran en el ecosistema y además estudiar las características de quienes podrían ser la futura competencia como el MSP430 o la herramienta Rocketchip; reuniendo información sobre su desempeño y técnicas ultra-bajo consumo. En especial recopilar información de la características que hacen al MSP430 una opción de ultra-bajo consumo y las características que hacen a rocketchip una herramienta que parametriza el diseño de su procesador rocket.

Este documento se enfocará en la comprensión y caracterización de las arquiteturas generadas por la herramienta Rocketchip. Se generan diferentes arquitecturas del procesador, midiendo el poder computacional bajo diferentes benchmarks en diferentes simuladores y validando sus resultados al implementar un rocket en una FPGA. El fin del proyecto de graduación, plasmado en este documento, es recolectar datos sobre el poder de procesamiento de las diferentes arquitecturas del procesador rocket y compararlas entre ellas y con datos reunidos del MSP430 en [24].

Los datos generados servirán como marco de referencia del poder de procesamiento del chip en diseño y además dan un primer acercamiento al funcionamiento del compilador que usará generando documentación sobre su uso e instalación. También se recolectaron resultados de algunos papers sobre procesadores que comparten características con el que se encuentra en diseño, para incluirlos en la comparación.

Para llevar al cabo estas comparaciones se uso el benchmarking realizado al MSP430 presente en [24] y corriéndolo en las diferentes arquitecturas RISC-V; para comparar las microarquitecturas RISC-V se uso el benchmarking dado por la herramienta rocketchip y finalmente con el benchmark dhrystone, el cual es usado para medir el rendimiento en operaciones de enteros y usado en [15], [17], [14], [1], [2] y [9]

1.1 Objetivos y estructura del documento

1.1.1 Objetivo General

Desarrollar una metodología de verificación enfocada en la caracterización del poder de procesamiento y una estimación del consumo de potencia de las microarquitecturas de RISCV implementadas usando el generador Rocket

1.1.2 Objetivos Específicos

- Definir una metodología de verificación basada en benchmarks para obtener la velocidad de procesamiento de diferentes microarquitecturas RISC-V.
- Comparar el rendimiento de las arquitecturas RISC-V con otros procesadores similares, en especial el MSP430.
- Realizar una síntesis en Vivado para una placa Zedboard de una de las arquitecturas RISC-V donde se obtenga una estimación de su consumo de potencia.

Capítulo 2

Marco Teórico

La velocidad de procesamiento y consumo de potencia son puntos de interés para la caracterización de un procesador. Algunas aplicaciones como los marcapasos requieren un procesador con un consumo de potencia bajo para prolongar la vida útil de la batería, otras aplicaciones relacionadas al procesamiento de señales requieren una alta capacidad de procesamiento para llevar acabo operaciones muy precisas. La correcta caracterización de un procesador permite a los diseñadores elegir el que mejor se adapte las necesidades de su sistema. En el presente capítulo se pretende explicar las principales métricas usadas para medir el poder de procesamiento y el consumo de potencia en microprocesadores.

2.1 Velocidad de Procesadores

2.1.1 Tiempo de Ejecución y Velocidad de Procesamiento

Cuando se menciona que el procesador "X" es n veces más rápido que el Y en la ejecución de una tarea, significa que el tiempo de ejecución (*execution time*) es n veces menor en X que en Y. El tiempo de ejecución y la velocidad de procesamiento mantienen una relación inversa. Esto quiere decir que si un procesador tiene un tiempo de respuesta muy grande, su velocidad de procesamiento es muy pequeña.En la ecuación 2.1 muestra el cálculo de la relación n de procesadores con tiempo de ejecución o velocidad de procesamiento [20].

$$\frac{Executiontime_y}{Executiontime_x} = \frac{Performance_x}{Performance_y} = n \tag{2.1}$$

A la hora de comparar la velocidad de procesadores, una medida de peso es el tiempo de ejecución de aplicaciones reales, ya que esta reflejan resultados fiables del rendimiento del procesador. Los tiempos de ejecución de programas pequeños que intentan imitar cargas de trabajo reales son una mala métrica, ya que no reflejan el verdadero uso que tendrá en la aplicación.
Existen muchas maneras de definir el tiempo de ejecución, la manera más sencilla es conocida como *walk-clock, reponse time or elapsed time*. El *response time* es la tardanza que tiene el

sistema en completar un programa. Esta definición incluye los tiempos de acceso a memoria, actividades de I/O, operaciones del sistema operativo, en otras palabras considera todo. La actividad del I/O(Inputs and Outputs) son procesos largos en los cuales el procesador no realiza ningún trabajo, o bien, con la multiprogramación puede ejecutar otros programas en su espera. Ambas situaciones provocan que el tiempo de ejecución no refleje el poder computacional del procesador sobre una tarea. Para medir el poder computacional sobre un programa específico se usa el CPU time. El CPU time es el tiempo en el que el procesador se encuentra realizando cálculos de un solo programa, excluyendo la espera de I/O o la ejecución de otros programas. Cuando un usuario de computadora corre un programa, la tardanza que tiene en ejecutarse es el tiempo de ejecución y mientras que el CPU time es este mismo tiempo sin la espera de entradas y salidas; por ende el CPU time siempre va ser más pequeño.

2.1.2 Benchmarks

Un benchmark es un programa que mide, por medio de su ejecución y carga de trabajo, el rendimiento del compilador y del procesador. El mejor benchmark según [20], es una aplicación real, cualquier otro programa más pequeño que intente imitar la ejecución de una aplicación puede llevar a datos falsos de rendimiento. Existen 3 tipos de benchmark que intentan imitar programas:

- Kernels: Son pequeños programas con partes claves de aplicaciones reales.
- Toy programs: son programas como de 100 líneas, tipo quicksort.
- Synthetic benchmark: son programas falsos que tratan de imitar a un usuario dentro de una aplicación real, entre ellos esta el Dhrystone.

Estos tres tipos de benchmark están desacreditados individualmente, ya que el compilador puede conspirar para generar resultados que aparentan ser más rápidos de lo que en realidad son. Otro problema, con estos tipos de benchmark, es bajo que condiciones están corriendo, los compiladores utilizan diferentes banderas qué optimizan el código de maquina disminuyendo el CPU time. Diferentes banderas crean condiciones desiguales entre procesadores, además que muchas de estas banderas resultan ilegales en la ejecución de aplicaciones reales, por ende no reflejan resultados fiables. Seleccionar banderas específicas para la ejecución de un benchmark iguala las condiciones de ejecución y mejora su semejanza con una aplicación real.
Una medida de *performance* muy común para procesadores con gran variedad de aplicaciones son los llamados *benchmark suite*. Consisten en una agrupación de benchmarks, que individualmente explotan diferentes partes del procesador y juntos logran imitar la ejecución de una aplicación real. Por supuesto su eficacia depende de la consistencia individual de los programas que los componen. La variedad de benchmarks provoca que las banderas específicas de ejecución tengan efectos diferentes dentro del suite, ya que algunos benchmark correrán más rápido y otros más lento. Usando medidas estadísticas se reduce los efectos de

las optimizaciones, ya que se promedian los resultados por medio de una media geométrica. Por ende los benchmarks que optimiza las banderas del compilador se ven compensados por aquellos que no las optimizan.
Como cada benchmark dentro del suite da un resultado individual de *execute time*, sería bueno si se pudiera resumir todos los resultados dentro de un solo número. Para ello primero se debe referenciar cada uno de los resultados respecto al resultado de un procesador más lento, con ecuación 2.2.

$$\frac{Execution\_time_{reference}}{Execution\_time_x} = Ratio \tag{2.2}$$

Para obtener un dato numérico único que represente el desempeño del procesador ante otros que también corran el mismo suite se usa la media geométrica con la ecuación 2.3.

$$\sqrt[n]{\prod_{i=1}^{n} ratio_i} = media \tag{2.3}$$

Para relacionar la velocidad de procesadores corriendo el mismo benchmark suite se usa la ecuación 2.4 (donde un procesador es x veces más rápido).

$$\frac{Ratio_A}{Ratio_B} = \frac{Performance_A}{Performance_B} = \frac{execute\_time_B}{execute\_time_A} = x \tag{2.4}$$

Se puede buscar la variabilidad de los datos por medio de la desviación estándar geométrica. La cual se obtiene con la ecuación 2.5 :

$$desvi = exp\left(\sqrt{\frac{\sum_{i=1}^{n}(ln(sample_i) - ln(mean))^2}{n}}\right) \tag{2.5}$$

2.1.3 Amdahl's Law

La ley de Amdahl es usada para medir el efecto en *performance* al añadir una extensión o mejora al sistema. Esta ley relaciona la velocidad del procesador con y sin la mejora, para obtener un número que represente el desempeño de su adición. La ley de Amdahl define *speedup* como la relación que resulta de dividir la velocidad usando el módulo entre la velocidad sin el módulo, como se observa en la ecuación 2.6.

$$speedup = \frac{Performance\_task\_with\_module}{Performance\_task\_without\_module} \tag{2.6}$$

Esta ley dice cuanta veces más rápido es el procesador corriendo una tarea con el módulo agregado, para así hacer una valoración de su adición. Amdahl es dependiente de 2 valores :

- **Fracción de tiempo de ejecución de la porción del código mejorada (F)** : Hace referencia a cual fracción del código fue mejorada por la adición. Por ejemplo si un programa tarda 50 segundos ejecutándose, en una porción del código se realizan un grupo de sumas que tardan 30 segundos y si la mejora ataca el tiempo de ejecución de las sumas; entonces la fracción de tiempo mejorado (F) sería 30/50. Este número siempre es uno o menor.

- **Ganancia en tiempo de la porción mejorada(G)**: Es la ganancia resultante de tomar el tiempo de la porción de código sin la mejora y dividirla entre el nuevo tiempo obtenido con la mejora. Tomando el ejemplo anterior el tiempo original ejecutando las sumas es de 30 s, con la mejora es de 10 s. Por lo cual G sería 30/10. Este número puede ser uno o mayor.

En la ecuación 2.6 se rescribe en 2.7 pero en función de F y G.

$$speedup = \frac{1}{(1-F) + \frac{F}{G}} \tag{2.7}$$

Esta ley sirve para cuantificar la ganancia al agregar mejoras en la arquitectura y compararlas con el costo que estas incurran en su implementación, para saber que tan beneficiosa es.

2.1.4 Tiempo de CPU y Ciclos de Reloj por Instrucción

Una parte esencial en todas las computadoras y en la ejecución de un programa es la frecuencia de clock. El CPU time puede obtenerse con la cantidad de clocks ejecutados al correr un programa y dividiéndolo entra la frecuencia del clock. Tal como se muestra en la ecuación 2.8

$$CPU\_Time = \frac{CPU\_clocks\_cycles\_count\_for\_a\_program}{frecuencia\_de\_clock} \tag{2.8}$$

Comúnmente los diseñadores trabajan con el *clock per instruction*(CPI), el cual es el número promedio de ciclos de reloj que tarda cada instrucción en un programa. El CPI es un número que como mínimo es uno En algunos casos los diseñadores usan *instructions per clock*(IPC) para relacionar las instrucciones con la ejecución de una tarea. El IPC siempre es un número que como maximo es 1. Para calcular el CPI se necesita el *instruction count* (IC), que es el número de instrucciones ejecutadas, y el número de ciclos de reloj ejecutados en el programa. En la ecuación 2.9 se muestra el cálculo de CPI.

$$CPI = \frac{CPU\_clocks\_cycles\_count\_for\_a\_program}{IC} \tag{2.9}$$

Se puede reescribir la ecuación 2.8, dejándola en términos de IC, CPI y la frecuencia de clock como se muestra en la ecuación 2.10

$$CPU\_time = \frac{ICxCPI}{frecuencia} \tag{2.10}$$

De la ecuación 2.10 se puede observar que el tiempo de ejecución es dependiente de 3 factores:

- **Frecuencia** : Depende de la tecnología y la organización.
- **CPI**: Depende la organización y la arquitectura.
- **IC**: Depende del ISA y la tecnología del compilador.

Mejorar el rendimiento de uno de ellos en 10 % tendría un efecto en reducir un 10 % el tiempo de ejecución. Sin embargo un cambio en ellos afecta a los otros, ya que son sumamente interdependientes.

2.1.5 MIPS

Una medida común de velocidad en un programa es la cantidad de instrucciones que puede ejecutar el sistema en un segundo, esta unidad normalmente se expresa en millones de instrucciones por segundo (MIPS). Según [21] los MIPS se expresan con 2.11 donde Ic son las instrucciones de maquina contadas y T el tiempo de ejecución. También se pueden expresar en términos de CPI y de la frecuencia (f) como se muestra en 2.12.

$$MIPS = \frac{I_c}{Tx10^6} \tag{2.11}$$

$$MIPS = \frac{f}{CPIx10^6} \tag{2.12}$$

2.1.6 Dhrystone benchmark

Según [14] y [10] es un benchmark sintético creado en 1984 usado para evaluar el rendimiento del procesador y del compilador en la ejeución de operaciones de punto fija. Este benchmark implementa en un ciclo de su ejecución una serie de operaciones mixtas en lenguaje C. Su ventaja frente a otros benchmarks es que ha sido altamente usado por mucho tiempo y por muchos sistemas para medir velocidad. Es fácilmente portable, accesible y compilable, por lo cual resulta secillo encontrar datos de su ejecución en otros sistemas. Existen diferentes versiones y diferentes banderas de compilador para correrlo, sin embargo no existe una guía estandarizada para reportar sus resultados. Lo que desacredita muchos de sus resultados, ya que se desconoce el ambiente en el que se ejecutó.
Es benchmark mide cuantos ciclos de ejecución de su código principal puede realizar en un segundo y se obtiene con ecuación 2.13. El programa permite seleccionar cuantas repeticiones de su código principal se desean hacer.

$$Dhrystone/s = \frac{dhrystone\_loops}{tiempo\_ejecucion} \tag{2.13}$$

Los DMIPS son los Dhrystone entre segundo normalizados al procesador VAX, el cual logra ejecutar 1MIP a 1757 dhrystone. Para obtenerlos se usa la ecuación 2.14.

$$DMIPS = \frac{Dhrystones/s}{1757} \tag{2.14}$$

Otra unidad típica que relaciona la frecuencia de operación son los DMIPS/MHz que se obtienen al dividir los DMIPS resultantes entre la frecuencia de operación como se observa en 2.15.

$$DMIPS/MHz = \frac{DMIPS}{Frecuencia} \tag{2.15}$$

En [10] recomiendan correrlo múltiples veces como mínimo 10 variando el número de iteraciones y cada corrida debe durar más de 20 segundos para que este programa genere datos fiables.También antes de cada corrida se debe recetear el procesador y cada sistema que lo compone como la caches.

2.2 Potencia

Uno de los retos al diseñar procesadores es tener altas velocidades de procesamiento con un bajo consumo de potencia. Altos consumos de potencia generan calentamiento en los chips, afectando y dañando su funcionamiento; obligando a invertir recursos y tiempo en sistemas de enfriamiento. Además, un bajo consumo prolonga la vida útil de las baterias ahorrando su carga. En esta sección se platean métricas y consideraciones para la estimación potencia en procesadores.

2.2.1 Potencias Dinámica

En sistemas digitales la mayor parte de la energía es consumida por el *switching*. El *switching* hace referencia a los cambios de estado de 0 a 1 del transistor, que es cuando este absorbe energía de la fuente de alimentación. En sistemas grandes como procesadores, se dificulta medir el consumo transistor por transistor, ya que hay millones de transistores. Se usa el factor de conmutación para obtener la frecuencia de switching general del sistema. El factor de conmutación es la relación de como conmuta un dispositivo digital respecto a su clock (el cual siempre hace transiciones de 0 a 1 y tiene un factor de conmutación igual a 1). Este consumo lleva como nombre "potencia dinámica" y su comportamiento esta dado por la fórmula 2.16 [13].

$$potencia\_dinamica = \frac{Carga\_capacitiva * Voltaje^2 * \alpha * frecuencia_{clk}}{2} \tag{2.16}$$

Como se puede observar en 2.16 al disminuir el voltaje y la frecuencia, se disminuye el consumo dinámico. La carga capacitiva depende de la tecnología de fabricación, del número de transistores conectados en la salida y de las interconexiones en el sistema. Cuando se escala la tecnología por una más pequeñas se aumenta el número de transistores que hacen *switching* y la frecuencia con la que lo hacen, disminuye la carga capacitiva y el voltaje; sin embargo el primer efecto domina sobre los otros provocando que aumente el consumo de potencia. En otras palabras escalar en tecnología tiene como efecto aumentar el consumo dinámico del sistema [20].
El calculo del consumo energético esta dado por la fórmula 2.17.

$$Energia = Carga\_capacitiva * Voltaje. \tag{2.17}$$

Como se observa en 2.17 la frecuencia no tiene un efecto sobre la energía, por eso al disminuirla solo se obtiene un efecto beneficioso en la potencia.
Existen técnicas para disminuir el consumo de potencia demandada por el transistor como apagar el reloj de los módulos que no están en uso. El uso de estas técnicas en dispositivos ultra bajo consumo disminuyen a tal punto el consumo dinámico que el consumo estático comienza a tener relevancia.

2.2.2 Consumo Estático

El consumo estático es el consumo exigido por mantener encendido el procesador. A pesar el procesador no se encuentra ejecutando ningún programa, este consume energía debido a diferentes efectos como las corrientes de fuga. Este consumo se calcula con la formula 2.18 [20].

$$Potencia\_estatica = Corriente\_fuga * Voltaje. \tag{2.18}$$

2.3 RISC-V

2.3.1 Arquitectura RISC-V

Según [13], el *instruction set architecture* (ISA) es la interfaz entre el hardware y más bajo nivel de software en una computadora. Convirtiéndose en el primer nivel de abstracción en la creación de un programa. El ISA de un procesador define su hardware y normalmente es caracterizado por el ancho de bits de sus registros. RISC-V es un ISA de código abierto que originalmente fue diseñada para la investigación y educación sobre la arquitectura de computadoras y ahora se ha convertido en una arquitectura abierta estándar para implementaciones industriales. RISC-V fue creada por los investigadores de la división de ciencias de la computación de la universidad de California Berkeley [5]. RISC-V permite a los diseñadores personalizar su procesador al agregar extensiones especializadas en un cierto tipo de tarea. El set de instrucciones que da soporte a operaciones de enteros, es también la base de todas las arquitecturas, el resto de extensiones agregan instrucciones para acelerar cierto tipo de operaciones. Este set de instrucciones se representada con la nomenclatura: RV64I y RV32I, donde la "I" representa la extensión que da soporte a enteros y el número representa el ancho de los registros. Las extensiones están divididas en dos grupos: las estándares y las no estándares. Las estándares no son tan especializadas y no dependen ni afectan ninguna otra extensión al ser agregadas. En cambio las no estándares son más especializadas y pueden entrar en conflicto con otras extensiones o depender de estas en su inclusión [8].
El set de instrucciones base es llamada "I", como se mencionó anteriormente, y contiene las instrucciones que soportan las operaciones de enteros, operaciones básicas de almacenado y descarga de memoria y instrucciones para acceder a los registros de control(CSR). La extensión estándar "M" da soporte a operaciones de multiplicación y división en los registros de enteros. La extensión "A" agrega las instrucciones para hacer lectura, modificaciones y escrituras en memoria de manera sincronizada entre procesadores. La extensión denotada con "F" agrega los registros para almacenar números flotantes, operaciones de precisión simple de números flotantes y operaciones en memoria de números flotantes. La extensión D permite al procesador expander los registros de flotantes y operar números flotantes con una doble precisión. La arquitectura RV64G y RV32G son equivalente a tener todas la extensiones implementadas, en otra palabra son equivalentes a RV64IMAFD y RV32IMAFD.

2.3.2 Rocketchip y Toolchain

Rocketchip es un generador de *System-on-Chip* de código abierto capaz de crear un archivo RTL sintetizable del mismo. Fue creado por la universidad de California Berkeley con fines investigativos y industriales, esta construido en un lenguaje de descripción de hardware llamado Chisel creado también por ellos. Esta basado en una colección de librerías de construcción parametrizables que permiten crear diferentes versiones del rocket [11]. La herramienta permite crear diferentes arquitecturas del procesador variando las extensiones que se le agreguen; por defecto construye la RV64IMAFDC sin embargo se pueden adherir y quitar las extensiones a gusto. También la herramienta permite cambiar el ancho de los registros que por defecto son de 64 bit a 32 bits. El tamaño de la memoria y la cantidad de cache que tiene el procesador también son variables que se pueden parametrizar.

La herramienta permite realizar una prueba funcional donde cada uno de sus módulos escritos en lenguaje C son simulados en verilator, el cual es un simulador que usa código en chisel o verilog para generar su equivalente en system C y hacer las pruebas, para más información de verilator consulte [4]. La prueba funcional se concreta corriendo los benchmarks en lenguaje C y en ensamblador, que trae por defecto la herramienta, y midiendo el rendimiento en ciclos de maquina realizados en su ejecución. También es posible realizar otra prueba funcional con la misma dinámica pero en VCS, un simulador de synopsis que hace pruebas de comportamiento sobre archivos verilog. Otra capacidad de la herramienta es la de generar archivos verilog sintetizables en una FPGA con las herramientas necesarias para correr programas en ella.

Entre las carpetas de la herramienta se encuentra una llamada "riscv-tools", esta carpeta contiene entre sus archivos el toolchain y los benchmarks que la herramienta trae por defecto; junto con los scripts necesarios para la contrucción del compilador y la compilación de los benchmarks. En la siguiente sección se explica como compilar programas para una arquitectura específica.

2.3.3 Compilador(-march y -mabi)

GCC, el popular compilador para GNU/linux, soporta como backend el instruction set de RISC-V. En esta sección se pretende explicar como usarlo para generar código compatible con las diferentes extensiones de RISCV. Para ello se tiene que hacer uso de dos banderas:

- **-march**: La cual selecciona la arquitectura objetivo. Indicando que instrucciones y registros son permitidos usar para el compilador.

- **-mabi**: Controla que argumentos convecionales en el programa (tipo int,long, float) se mapean a cuales registros y layout de la memoria de datos.

-march

Esta bandera le indica al compilador el instruction set que podrá usado al traducir un programa. Según [12] las extensiones que tienen soporte en el toolchain son la M, la A, la F, la D y la C (Instrucciones para comprimir) para los casos de RV32I y RV64I. Para usar esta bandera solo se debe agregar "-march=" más la siglas de la extensión deseada, todo esto después de la instruccion de compilación. A continuación se muestra un ejemplo al compilar un codigo C de multiplicación. Sobre el siguiente código C que describe una función de multiplicación.

```
double dmul(double a, double b) {
  return a * b;
}
```

Figura 2.1: Código de Multiplicación de Dobles Ejemplo

A continuación se muestra la instrucción usada más el código de ensamblador generado. Como se observa se le indica al compilador que puede usar la extensiones de multiplicación, y en el código de ensamblador se observa que las usa.

```
$ riscv64-unknown-elf-gcc test.c -march=rv64imafdc -mabi=lp64d -o- -S -O3
dmul:
  fmul.d  fa0,fa0,fa1
  ret
```

Figura 2.2: Compilación de la Multiplicación Usando march Primer Caso

En cambio, en el ejemplo siguiente se compila el mismo código indicándole que no puede usar esa extensión. Como se observa el compilador resuelve la suma por otros métodos.

```
$ riscv64-unknown-elf-gcc test.c -march=rv64i -mabi=lp64 -o- -S -O3
dmul:
  add     sp,sp,-16
  sd      ra,8(sp)
  call    __muldf3
  ld      ra,8(sp)
  add     sp,sp,16
  jr      ra
```

Figura 2.3: Compilación de la Multiplicación Usando march Segundo Caso

Ambos ejemplos se obtuvieron de [12].

-mabi

Es un argumento que le indica al GCC sobre la especificaciones de las variables enteros y flotantes, en otra palabras define el *application binary interface*(ABI), para que compile un código compatilbe con la aquitectura del compilador.Según [13] el ABI es la combinación del ISA base y de la interfáz del sistema operativo. Tal como march especifica que tipo de código puede correr el hardware, mabi especifíca como el código de software puede vincularse con el sistema. Para Risc-V existen 2 ABI tipos enteros y 3 ABI tipos flotantes; los enteros son:

- **ilp32**: Define los tipo de dato int, long y punteros con extensiones de 32 bits, long long con extensiones de 64 bits, los char como de 8 bits y short como de 16 bits.
- **ilp64**: Define los tipo de dato long y punteros como de 64 bits, los int como de 32 bits y el resto mantienen el formato de 32ilp.

Las siguiente extensiones del ABI son para relacionar los datos tipo flotante con los registros en la microarquitectura especificamente para ellos:

- **""(string vacío)**: No pasan argumentos de coma flotante a los registros.
- **f**: Argumentos de coma flotante de 32 bits o más pequeños pasan a los registros. Requiere la extensión F ya que sin ella no hay registros de coma flotante.
- **d**: Argumentos de coma flotante de 64 bits o más pequeños pasan a los registros. Requiere la extensión D.

2.3.4 Timers and counters

Las aquitecturas RV64I y RV32I proveen de contadores de solo lectura de 64 bits que contienen el tiempo, ciclos de maquina ejecutados, instrucciones corridas, etc. Estos contadores son mapeados con direcciones de 12 bits y se puede acceder a ellos usando la instrucción CSRRS(*rd, csr, x0*) [8]. Esta instrucción copia el valor que tiene el registro solicitado en un registro de enteros. Para el caso de RV32I sus registros de enteros son muy pequeños para almacenar el dato de los contadores, por lo cual se usan 2 registros para que uno almacene los primeros 32 y el otro los últimos. Estos registros y contadores de control son una herramienta importante para rendimiento de los procesadores con ISA RISC-V.

Capítulo 3

Caracterización de las Arquitecturas RISC-V

Para caracterizar los procesadores RISC-V se recurrió a 2 grupos de programas: RISC-V benchmark suit y MSP430 benchmark suit, junto con la ejecución independiente del programa dhrystone que mide su desempeño en datos tipo entero. Cada uno de ellos contienen programas que explotan diferentes partes del procesador mientras que miden su rendimiento en ejecución; el primer suit es usado para caracterizar procesadores RISC-V y el segundo es un benchmarking usado para caracterizar el procesador de *Texas Instruments* el MSP430. En esta sección se pretende explicar como se eligió, compiló, ejecutó y validó los grupos de benchmarks para las diferentes arquitecturas en dos diferentes simuladores.

3.1 MSP430 Benchmark Suit

Este conjunto de benchmarks se usó con el objetivo de comparar el poder de procesamiento de las diferentes microarquitecturas RISC-V y el MSP430 de *Texas Instruments*. Dentro de [24] se encuentra los resultados en unidades de CC que obtuvo el MSP430 en cada programa, junto con el código en C de cada benchmark. En el documento se presentan dos tipos de resultado para el procesador MSP430 uno usando optimizaciones del compilador y sin ellas. Este benchmarking se conforma de los siguientes programas:

1. **Matemática de 8, 16 y 32 bits**: El objetivo de este programa es medir el rendimiento en la ejecución de operaciones matemáticas basicas en datos enteros de 8, 16 y 32 bits (datos tipo char, short y long). Cada uno de ellos inicia ejecutando una suma, una multiplicación y termina con la elaboración de una división.

2. **Matemática de Datos tipo Flotantes**: El objetivo de este programa es medir el rendimiento del procesador en la ejecución de operaciones matemáticas con datos del tipo flotantes. Inicia realizando una suma, luego una multiplicación y terminando con una división en punto flotante.

3. **Movimiento de matrices 8 y 16 bits**: Estos 2 programas copian varias veces una matriz de 4x16 con datos internos de 8 bits y 16 bits respectivamente. Este tipo de operaciones estresan la memoria ya que deben guardar los arreglos en ellas y moverlos.

4. **Operación Switch Case 8 y 16 bits**: Estos 2 programas usan la función *Switch/case* usada comúnmente para la construcción de maquinas de estado. Se pretende con este benchmark medir el desempeño de los procesadores en la implementación de maquinas de estado.

5. **Multiplicación de Matrices**: Este programa realiza un multiplicación de una matriz 3x4 con una 4x5 que da como resultado una matriz 4x4. La multiplicación de matrices en este caso se lleva acabo con un tipo de dato *short*; y ejecuta operaciones de multiplicación, suma y acumulación para su realización. Por tratarse de matrices el programa también hace bastante uso de memoria al guardar y cargar vectores.

6. **Filtro FIR**: Se realiza la implementación de un filtro digital FIR, el cual realiza muchas operaciones de multiplicación, suma y acumulación en su ejecución. Con él se mide el rendimiento en operaciones tipo punto flotante, comúnmente usadas en la implementación de cualquier filtro digital.

3.2 RISC-V Benchmark Suit

La herramienta Rocketchip ofrece un benchmarking que sus clientes pueden utilizar con el fin de caracterizar el procesador que ellos mismos construyen. Debido a que las implementaciones más básicas que se pudieron construir (RV64I y RV32I) no fueron capaces de ejecutar algunos programas dentro del benchmarking, este se limitó para el uso de aquellos que podían ser ejecutados en todas las arquitecturas. Los resultados se usaron para comparar y caracterizar la velocidad de las diferentes micro-arquitecturas y observar la bonificación en velocidad en la adición de distintas extensiones. Los programas que componen este benchmarking están escritos en C y son:

1. **Filtro Mediana**: Es un tipo de filtro comúnmente usado en procesamiento de imágenes, su funcionamiento consiste en tomar grupo de N datos y calcula la mediana de ellos. Luego se vuelven a tomar los datos pero sumando a n+1 y se vuelve a calcular la mediana y así consecutivamente [18]. Es un filtro que quita el ruido en imágenes removiendo los picos en ellos y suavizando la imagen. Con este benchmark se evalúa la ejecución del procesador en operaciones comparativas.

2. **Qsort**: Este programa ejecuta un algoritmo de ordenamiento llamado *Quicksort*, este tipo de programa estresa la memoria, ya que divide los datos en pequeños arreglos y luego los ordena. Muchos de estos subarreglos son guardados y cargados desde la memoria.

3. **Tower**: Este programa ejecuta un juego llamado torres de Hanoi; en donde el procesador estresa la memoria por medio de la recursividad de su algoritmo.

4. **Multiply**: Este programa ejecuta una multiplicación de software, realizando desplazamientos a nivel de bits. Multiplica dos vectores de datos y finalmente verifica el resultado retornando un 1 en caso de éxito.

5. **Vvadd**: Este programa ejecuta la suma de dos vectores guardando los resultados en un tercero. El programa mide la capacidad del procesador en ejecutar sumas de vectores de datos enteros.

3.3 Dhrystone

Este es un benchmark usado por muchos procesadores para medir el rendimiento en operaciones de strings y enteros. Se eligió por que es fácil encontrar resultados de su ejecución en otros procesadores, lo que facilita también su comparación. Los resultados de este benchmark sirvieron para comparar al procesador Rocket con otros procesadores similares y ubicar su rendimiento entre ellos.

El código de este programa venía junto al benchmarking de RISC-V, pero se separó del resto para independizar sus resultados. Se midieron los CC y las IC, además de los dhrystone entre segundo que podía procesar cada una de las arquitecturas Rocket. Con este último dato se calcularon los DMIPS referenciandolos a los resultados del procesador VAX. Como lo DMIPS son una unidad dependiente de la frecuencia la cual difiere entre procesadores se prefirió calcular los DMIPS/MHz, unidad que es independiente de la frecuencia. En este documento se recompilaron resultados de otros procesadores similares con el fin de realizar una comparacion, los resultados consultados se encuentran en [15], [17], [14], [1], [2] y [9].

En [15] se presenta una comparación entre el procesador su Rocket con arquitectura RISC-V y con MMU(Mapping Memory Unit) y el procesador ARM Cortex-A5 por medio de este benchmark. En [17] se expone un procesador RISC-V de 32 bits con una extensión llamada Oonlong que agrega instrucciones de aritmética con números complejos y la extensión M de multiplicación; este procesador es usa para aplicaciones de radio. En [1] y [2] son procesadores ARM diseñados para aplicaciones de internet de las cosas y con característica ultra-bajo consumo. En [14] se presenta el procesador leon-3 el cual es sintetizable en FPGA con una arquitectura de instrucciones SPARC-V8 de 32 bits.

El procesador MSP430 dentro de [24] corre el benchmark dhrystone, sin embargo solo reporta los ciclos contados en la ejecución. Para encontrar el dato de dhrystone entre segundo se calculó el tiempo de ejecución con la frecuencia de 8MHz (usada por el MSP430 en el benchmarking) y la cantidad de ciclos de máquina ejecutados. El tiempo calculado se dividió entre cantidad de veces que se corrió(*loops*) el programa y se obtuvo los dhrystone entre segundo del procesador. Este calculo se realizó para los resultados con y sin optimizaciones del compilador.

3.4 Función de medición y unidad de velocidad

Para medir y comparar el poder de procesamiento de cada benchmark se escogió CC (ciclos contados) y IC (Instrucciones contadas) de máquina ejecutados en cada simulación. No se usó el tiempo de ejecución por que depende de la frecuencia usada por el procesador la cual siempre difiere entre ellos; dándole ventaja a aquellos que la tienen más alta. Por ese motivo se prefirió usar CC, unidad de velocidad independiente de la frecuencia. Se contaron también las instrucciones con el propósito de medir el CPI de cada programa, que a pesar de no ser una unidad de velocidad relaciona la cantidad de ciclos promedio que ocupa una instrucción dando un referencia de la efectividad de las instrucciones.

Para encontrar este valor en las arquitecturas RISC-V, se incluyó en el código de cada benchmark una función que hace lectura de dos registros de control, los cuales cuentan las instrucciones y ciclos de máquina desde que se comienza a correr el programa. Se hacen dos lecturas en cada ejecución, la primera cuando ya se han declarado todas las variables y constantes del programa y la segunda lectura se realiza al terminar los cálculos; después de esto se restan ambos valores de las lecturas y se extrae el valor que representa los IC y CC de máquina realizados. El interés es medir la efectividad de las arquitecturas al momento de realizar los cálculos del programa con todas la constantes y variables ya declaradas, por ese motivo se excluyen todas las instrucciones y ciclos usados para declararlas y inicializar el dispositivo.

3.5 Construcción del compilador y de Rocket

Con la herramienta *Rocketchip* de la universidad de Berkeley se descargaron y instalaron todas las dependencias necesarias para el funcionamiento del compilador y del constructor de los procesadores Rocket. Se compiló cada uno de los benchmarks indicándole el ISA RISC-V objetivo para las diferentes implementaciones (por medio de las instrucciones -march y -mabi; se compilaron los test para el ISA objetivo):

1. RV64I: Rocket de 64 bits únicamente con la extensión de enteros.
2. RV32I: Rocket de 32 bits únicamente con la extensión de enteros.
3. RV64IM: Rocket de 64 bits con la extensión I de operaciones básicas de enteros y la extensión M de multiplicación y división de enteros.
4. RV32IM: Rocket de 32 bits con la extensión I de operaciones básicas de enteros y la extensión M de multiplicación y división de enteros.
5. RV64IMA: Rocket de 64 bits con la extensión I de operaciones básicas de enteros, la extensión M de multiplicación y división de enteros y A operaciones atómicas.
6. RV32IMA: Rocket de 32 bits con la extensión I de operaciones básicas de enteros, la extensión M de multiplicación y división de enteros y A operaciones atómicas.

7. RV64IMAFD: Rocket de 64 bits con la extensión I de operaciones básicas de enteros, la extensión M de multiplicación y división de enteros, extensión A de operaciones atómicas, extensión F de operaciones de coma flotante y la extensión D operaciones de doble presición de coma flotante.

8. RV64IMAFD: Rocket de 64 bits con la extensión I de operaciones básicas de enteros, la extensión M de multiplicación y división de enteros, extensión A de operaciones atómicas, extensión F de operaciones de coma flotante, la extensión D operaciones de doble presición de coma flotante y la extensión C especializada en operaciones de compresión y descompresión.

Para cada arquitectura se compilaron los benchmarks para su ISA objetivo, el cual le permite tener su máximo rendimiento aprovechando todos los recursos de su hardware. De esta manera se obtiene el mejor rendimiento de cada uno de ellos en la ejecución del benchmarking. Las arquitecturas armadas y los benchmarks compilados en esta herramientas son tomadas por los simuladores.

Se eligieron estas arquitecturas con el fin de comparar los rendimiento de la arquitecturas de 32 y 64 bits y observar la ganancia de velocidad al agregar las diferentes extensiones. Al medir las velocidades de las arquitecturas más básicas se encuentran los peores escenarios para el rendimiento de los procesadores Rocket implementados (RV64I y RV32I) y al usar la arquitectura con todas las extensiones se expone el mejor escenario.(RV64IMAFDC). Significa que cualquier otra construcción de Rocket debe tener un desempeño entre estos dos casos. Ahora el resto de arquitecturas construidas sirven para poner puntos intermedios que sirvan de referencia, además en sus adiciones se observan las bonificaciones de ir agregando cada extensión.

3.6 Simuladores : Emulador de Chisel y VCS

Para obtener los resultados se usaron dos simuladores de comportamiento, esto quiere decir que no asocian la construcción de Rocket con alguna tecnología. En otras palabras se simula únicamente el comportamiento de cada uno de los módulos en un caso ideal sin problemas de *timing*.

El primero de los simuladores es el emulador de Chisel basado en Verilator. Dentro del repositorio de rocketchip se encuentra una carpeta con el nombre de *emulator*; en ella se encuentran los scripts y pasos necesarios para correr una simulación con las arquitecturas construidas. El *script* construye cada uno de los módulos de la arquitectura y los ensambla para realizar un simulación comportamental en Verilator. Una vez ensamblado, el *script* toma un benchmark, lo carga a la memoria y lo corre. Al final de la ejecución despliega en la terminal del sistema anfitrión los IC y CC resultantes. Este *script* se ejecutó para todas las arquitecturas implementadas, corriendo todos los benchmarks descritos en secciones anteriores.

El segundo es VCS, un simulador de verificación funcional el cual es el mas usado por el top 20 de las empresas fabricadoras chips [7]. VCS toma el código de descripción de hardware(usualmente verilog) de un módulo para realizar una simulación funcional de él. Dentro del repertorio de Rocketchip se encuentra una carpeta llamada *Vsim* con las herramientas necesarias para generar la descripción del procesador en código verilog y simularlo en VCS usando los benchmarks como estimulación de entrada. Al igual que en el simulador anterior, el *script* inicia generando los archivos con la descripción del hardware en verilog, abriendo la herramienta VCS, cargando el procesador en ella e iniciando una simulación. La simulación carga un benchmark en la memoria, luego lo ejecuta y termina desplegando en la terminal del sistema anfitrión los IC y los CC. Este *script* se corrió para cada arquitectura con todos los benchmarks descritos en secciones anteriores y recompilando los datos generados por ellos.

Se usaron ambos simuladores para dar mayor firmeza a los resultados obtenidos, posteriormente se compararon los resultados con un caso practico obtenido por la implementación en una FPGA. Ambos simuladores fueron usados para observar el comportamiento funcional de las diferentes arquitecturas.

3.7 Implementación FPGA

Se programó la FPGA de la plataforma de desarrollo "Zedboard" [3] con la imagen de la arquitectura RV64IMAFD descargada en [6], la cual es una implementación generada por rocketchip. La Zedboard cuenta con un procesador ARM, el cual sirve como sistema anfitrión, siendo el encargado de cargar la memoria con los programas y desplegar los resultados. Por medio de un cable de ethernet se estableció una comunicación con el ARM para tomar control de él.

Es importante detallar que la imagen ofrecida en [6] difiere de las últimas versiones de Rocket teniendo un año de diferencia con las simuladas. Esto genera una diferencia entre ellas sumando error a los simuladores a la hora de la comparación. Otro factor que afecta los datos es que los simuladores corren versiones ideales de las arquitecturas sin considerar errores de timing, rompimiento de pipeline y retrasos por el acceso a la memoria.

En este caso práctico se corrieron todos los benchmarks descritos en las secciones anteriores y se extrajeron sus resultados, con el fin de compararlos con los casos ideales; de esta manera es posible calcular el error de los simuladores. Los resultados generados por este método sirvieron para validar los datos extraídos y observar que tanto se alejaban de ellos.El mejor resultado práctico posible es el obtenido por los simuladores. Para poder realizar la comparación se simuló la arquitectura RV64IMFD implementada en el caso práctico.

También se realizó una síntesis en Vivado de la cual se extrajo un reporte de potencia que estima el consumo dinámico hecho por la zedboard. Para obtener este reporte se uso un factor de actividad de 0.1, valor recomendado para una primera estimación de consumo de potencia dinámica [19].

3.8 Presentación de los Resultados

Como cada programas arroja números muy diferentes de IC y CC, los datos obtenidos se referenciaron respecto a los resultados del procesador RV32I. El número resultante de esta referencia representa el factor de menos CC e IC que tiene la arquitectura con respecto al RV32I. Como la velocidad es inversa a la cantidad de CC el número referenciado representa cuantas veces más rápido es con respecto al RV32I. Para calcular esta referencias se uso la ecuación 2.2 descrita en el marco teórico de este documento.

Para resumir los resultados de cada suit en un solo número se aplico la media aritmética representada en la ecuación 2.3 a cada uno de los resultados referenciados. Esta media aritmética resume el desempeño de cada procesador en los suits en un solo número que dice cuantas veces más rápido fue con respecto al RV32I. Esta media aritmética también se aplicó para las IC para resumir estos datos en un solo número.

Capítulo 4

Resultados y Análisis

4.1 Benchmarking provisto por Rocketchip

En esta sección se muestran los resultados obtenidos del benchmarking provisto por rocketchip simulado en VCS y verilator; también se incluyen lo resultados de la implementación en la Zedboard. Este benchmarking consiste en 5 programas llamados : vvadd (suma vectores), multiply (multiplicación de software), tower (programa recursivo), filtro de mediana y qsort (ordenamiento de listas).

4.1.1 Emulador de Chisel Verilator

En esta subsección se presentan los resultados obtenidos por verilator en unidades de CPI y IC bajo el benchmarking provisto por rocketchip. Los resultados de CC y la media geométrica se encuentran en los anexos de este documento con el fin de no recargar de imágenes esta sección. Como se observará posteriormente no hubo mayor diferencia entre las herramientas por ese motivo basta con analizar los resultados de un simulador.

En la gráfica de la figura 4.1 se ven los CPI resultante de cada arquitectura en el suit de RISC-V. Un CPI con valor de 1 muestra la mayor eficiencia de las instrucciones donde cada una de ellas tarda en promedio un ciclo de reloj en su ejecución, en cambio valores muy altos indican que la instrucciones tardan más ciclos para su realización. Como se observa en la gráfica 4.1 la efectividad de las instrucciones es estable sin importar que extensiones que se le agreguen. El programa vvadd obtuvo los valores más cercanos a 1, seguido por tower y multiply con valores aproximados de 1.1 , median obtuvo valores cercanos a 1.3 y el valor menos eficiente lo obtuvo qsort con 1.7. Este último tiene el valor más grande debido a que este programa de ordenamiento ejercita la memoria y las instrucciones que acceden a ella tardan en promedio más de un ciclo de reloj en su ejecución. Ninguna arquitectura sobresale con base a la otras en la efectividad de sus instrucciones en este caso, lo cual significa que ninguna de las extensiones implementadas mejora los resultados de CPI.

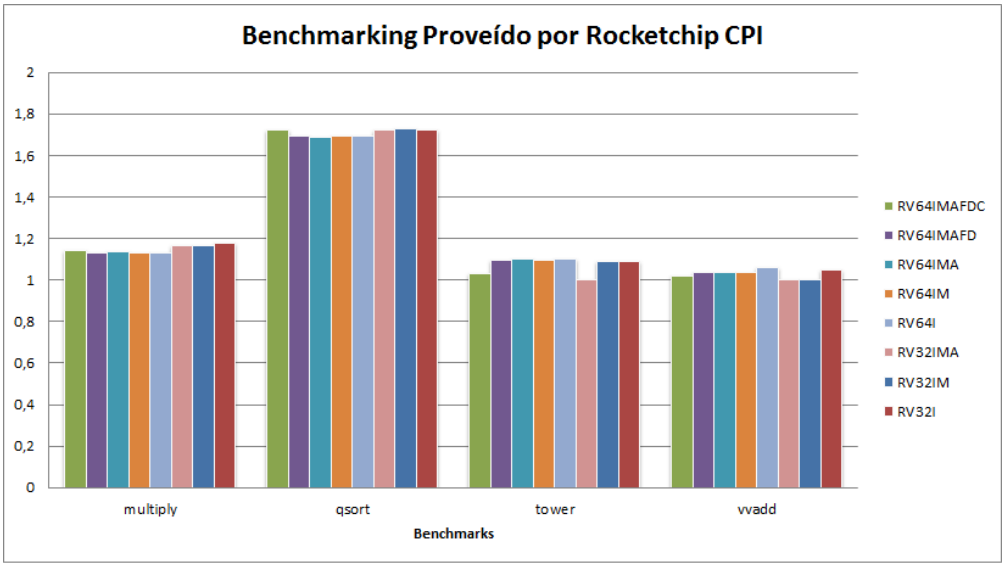

Figura 4.1: Ciclos por Instrucción del Benchmarking proveído por Rocketchip Simulado en Verilator

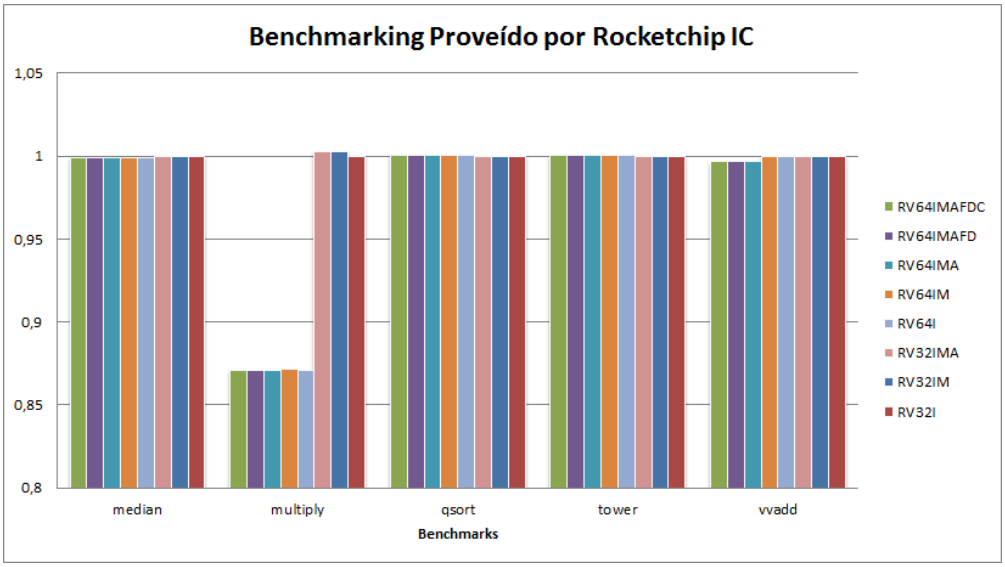

Figura 4.2: Instrucciones Contadas en el Benchmarking proveído por Rocketchip Simulado en Verilator

En la figura 4.2 se muestra la gráfica con los resultados de IC referenciados al RV32I en el benchmarking proveído por Rocketchip simulado en Verilator. La diferencia en IC es muy pequeña entre las implementaciones y la mas notable se observa en multiply donde los procesadores de 64 bits tienen aproximadamente 15 % más instrucciones que las arquitecturas de 32 bits. Solo en multiply se ve una mejora en la cantidad de instrucciones ejecutadas y es debida al ancho de palabra de los procesadores. La adición de extensiones no presenta una reducción en la IC, lo que significa que las instrucciones agregadas no mejoran la ejecución de estos programas.

4.1.2 VCS

En esta subsección se presentan los resultados obtenidos por la simulación del benchmarking proveído por Rocketchip en CC y la media aritmética en la herramienta VCS. Los resultados de este simulador de IC y CPI se encuentran en los anexos del documento.

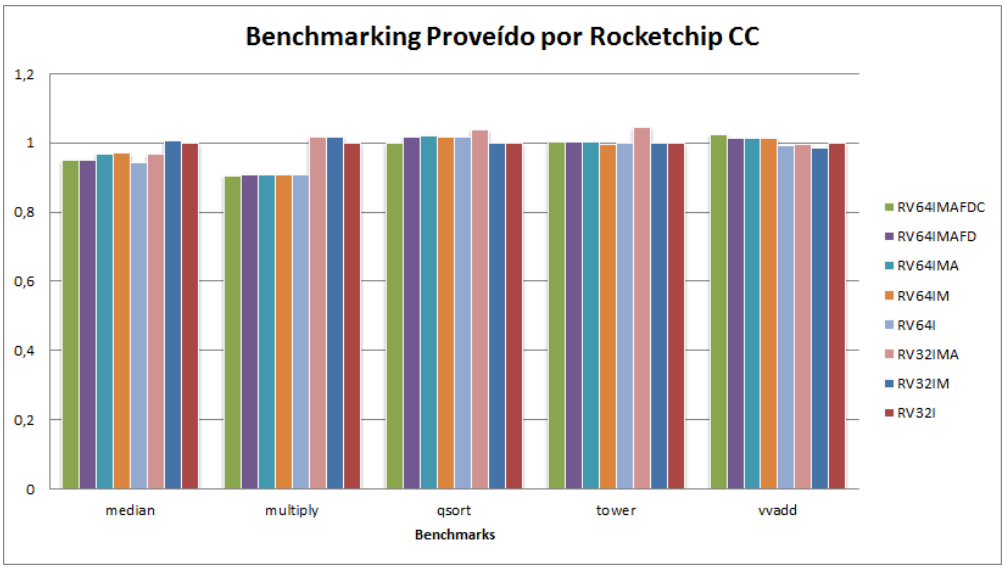

Figura 4.3: Ciclos de Reloj Contados del Benchmarking proveído por Rocketchip Simulado en VCS

En la figura 4.3 se observa los resultados de los CC referenciados respecto al RV32I. Al igual que los IC vistos en 4.2 los procesadores no se sacan mayor diferencia en velocidad de procesamiento, la ventaja más grande observada es en el programa multiply donde la multiplicación de software es ejecutada aproximadamente 15% más rápido para las arquitecturas de 32 bits con respecto a las de 64 bits. Las extensiones no tienen efecto en la velocidad por que no mejoran ni empeoran la ejecución de las sumas (Vvadd) o los tiempos de acceso a memoria (tower y qsort). Esto se reafirma al observar que en 4.2 la cantidad de IC ejecutadas no cambia entre arquitecturas, evidenciando que las instrucciones agregadas por las extensiones no mejoran el desempeño del procesador bajo este benchmarking.

En la figura 4.4 muestra la gráfica con los resultados del benchmarking de IC y CC por medio de la media aritmética. En IC los mejores resultados los tienen las arquitecturas de 32 bits empatando los 3, sin embargo ganan por muy poco tienen aproximadamente 0.08 menos instrucciones que las arquitecturas de 64 bits. En velocidad se observa que el modelo más rápido fue el RV32IMA, sin embargo al igual que IC las diferencias son muy pequeñas tanto así que el peor caso tiene solo una diferencia de 0.08 aproximadamente con el mejor desempeño. Como el número de instrucciones se mantienen constante al agregar y quitar extensiones, esta no agregan una bonificación en la ejecución del benchmarking. Esto indica que en la ejecución de es tipo de programas (multipliación de software, sumas y accesos a memoria) no mejora con las instrucciones agregadas por las extensiones.

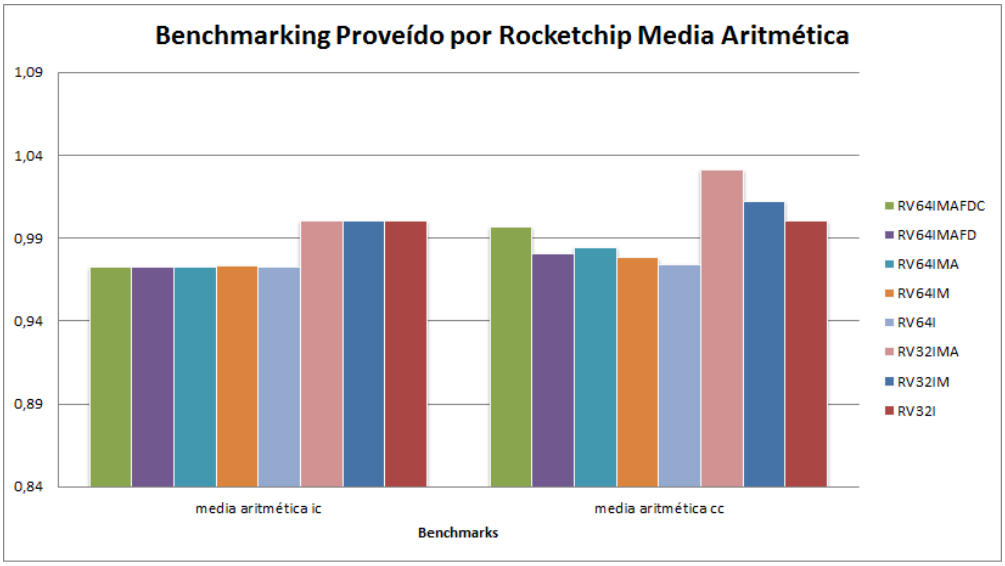

Figura 4.4: Media Aritmética de los Resultados del Benchmarking proveído por Rocketchip Simulado en VCS

4.1.3 Implementación Zedboard

En esta subsección se muestran los CC y IC de correr el suit en la implementación RV64IMFD en zedboard. También se exponen los errores de los simuladores al compararlos con los resultados prácticos. Los resultados CPI se encuentran en anexos de este documento.

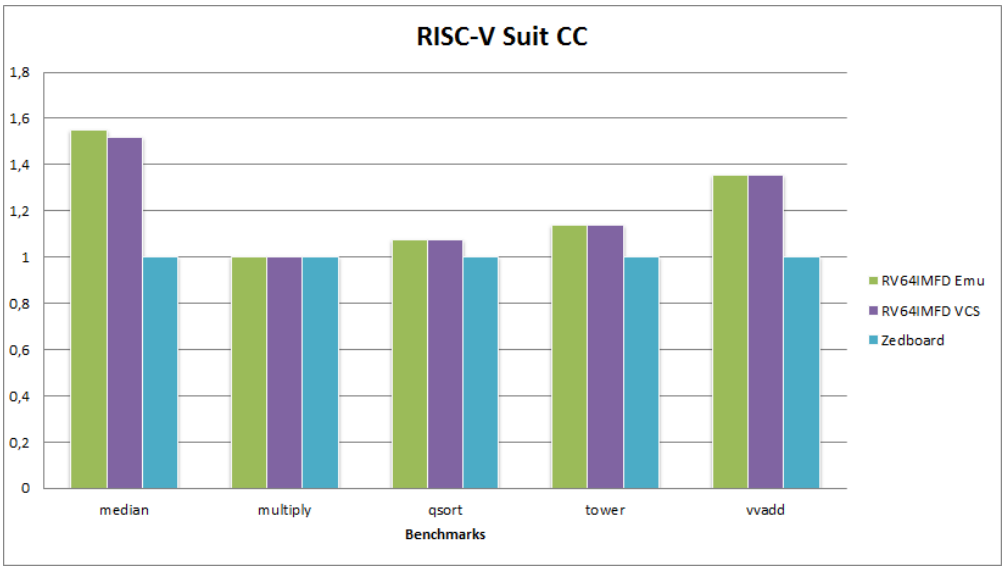

Figura 4.5: Ciclos de Reloj Contados del Benchmarking proveído por Rocketchip en la Zedboard

En la figura 4.5 se observan los resultados de los CC de los simuladores y de la implementación en Zedboard al referenciarlos a estos últimos. Se observa que en todos los casos los simuladores son más optimistas que en la prueba práctica. Los mayores errores se observan en el programa median y vvadd, donde los simuladores aseguran que el modelo es 1.5 y 1.3 veces más rápido respectivamente. Sin embargo estas diferencias se deben a que

los simuladores corren modelos ideales no referenciados a una tecnología, provocando que no consideren errores de una implementación práctica. Multiply fue el programa donde los simuladores fueron más precisos. Ambos simuladores como se observa en la gráfica dan respuesta muy similares entre ellos.

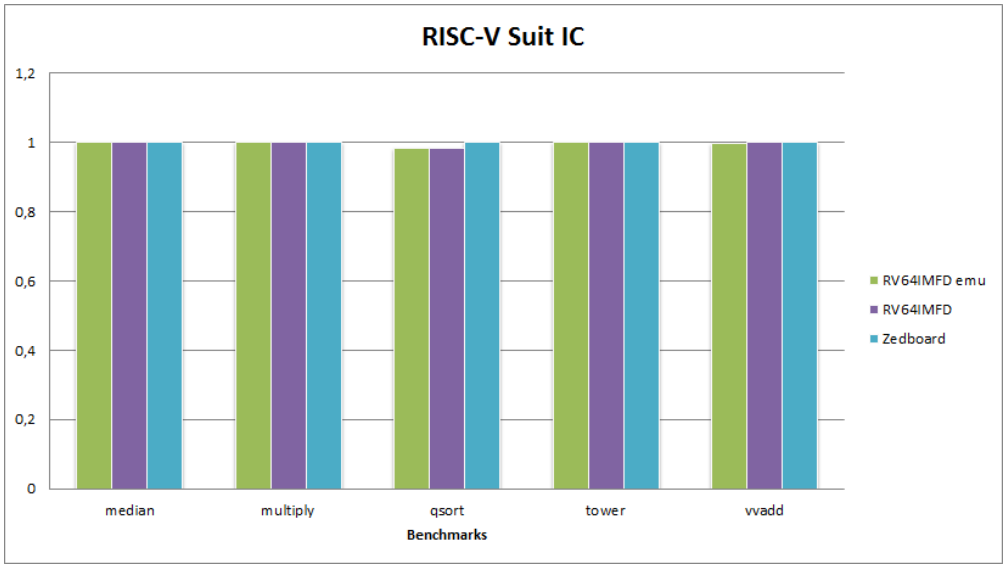

Figura 4.6: Instrucciones Contadas del Benchmarking proveído por Rockerchip en la Zedboard

En la figura 4.6 se observan las instrucciones contadas por los simuladores y por el modelo práctico. Como se observa las instrucciones contadas son todas cercanas a 1 lo que demuestra que los simuladores son muy precisos en contarlas.

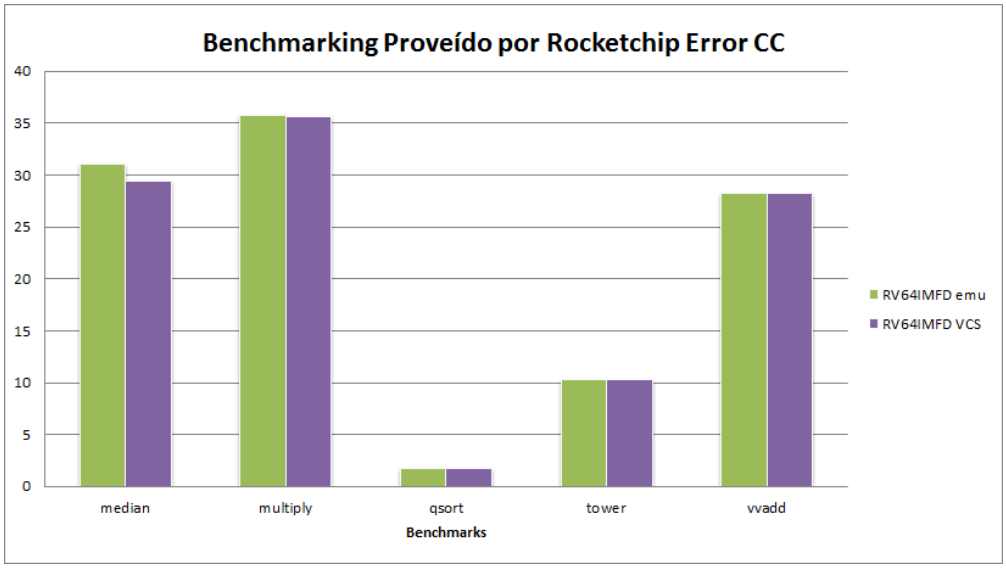

Figura 4.7: Error de los Simuladores en el Benchmarking proveído por Rocketchip

En la figura 4.7 se observan los errores de los simuladores en los CC calculados respecto a los resultados prácticos obtenidos con la Zedboard. Como se observa ambos simuladores tienen un error muy parecido debido a sus modelos ideales. Los errores más grandes son de 35 % y 25% aproximadamente y los obtienen al simular el filtro median y vvadd. Estos altos errores pueden deberse a que la versiones del Rocket simulado y práctico difieren por un año; además que el caso práctico considera errores de timing y de pipeline los cuales aumentan los CC en un programa.

En cuanto al error en predecir IC fueron calculados y se obtuvieron valores muy cercanos a 0 como se muestra en 4.6.

4.2 MSP430 Benchmarking

En esta sección se muestran los resultados obtenidos por el suit realizado al MSP430 en [24]. Se muestran los resultados de los CC en cada programa y se obtuvieron por medio de la herramienta VCS, mientras que los resultados del emulador de Verilator se encuentran en los anexos; esto para no rontundar en el análisis ya que ambos simuladores arrojan resultados muy similares. Al final se muestran los datos obtenidos de correr el suit en el modelo implementado en la Zedboard y se comparan con los datos obtenidos en los simuladores. Todos los datos del desempeño del procesador MSP430 se obtuvieron de [24] y este suit se compone de 10 programas.

4.2.1 VCS

En esta subsección se muestran los resultados de los CC ejecutados de algunos programas en las diferentes arquitecturas y termina con la media aritmética que resume el desempeño de los procesadores en el suit.

En la figura 4.8 se muestran los CC del programa switch case con datos de 8 y 16 bits; y se comparan con el desempeño del MSP430. Estos dos programas miden la ejecución de maquinas de estado y se observa que el MSP430 con y sin optimizaciones son aproximadamente 3 veces más rápidos que RV32I; además de ser también los más rapidos en esta tarea. Se observa que los los procesadores de 32 bits son levemente más veloces que los de 64 bits en la ejecución de esta tarea y que las extensiones no agregan una mejora en la velocidad de este programa.

En la figura 4.9 se muestran los CC en la ejecución del programa switch matrix, el cual ejercita la memoria al mover y copiar matrices. En este programa el procesador más rápido fue el MSP430 con optimizaciones de compilador, el cual es aproximadamente 1.6 veces más rápido que el RV32I. Mientras que el procesador más lento fue el MSP430 sin la optimizacio-

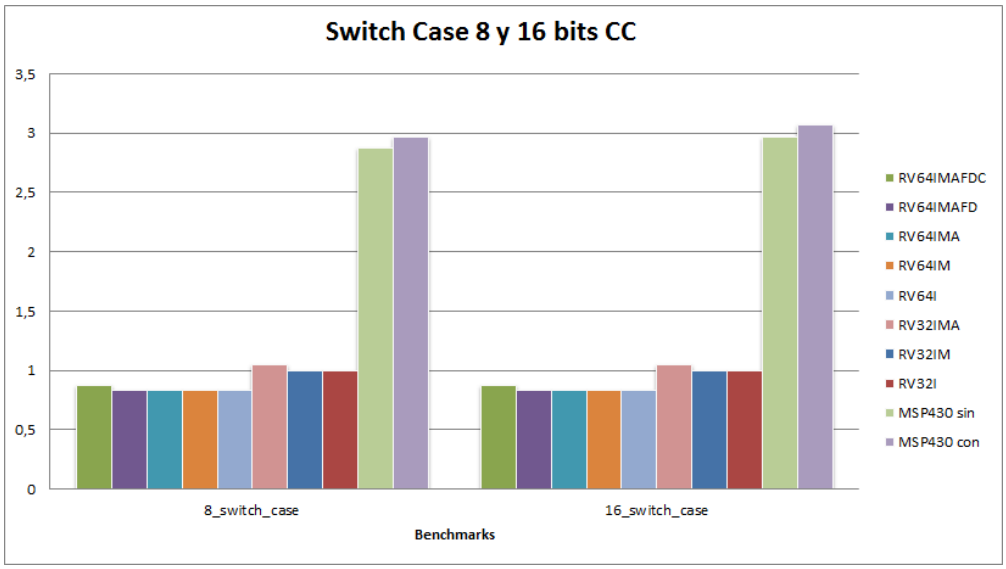

Figura 4.8: Ciclos Contados en la ejecución Switch Case con Datos de 8 y 16 bits Simulado en VCS

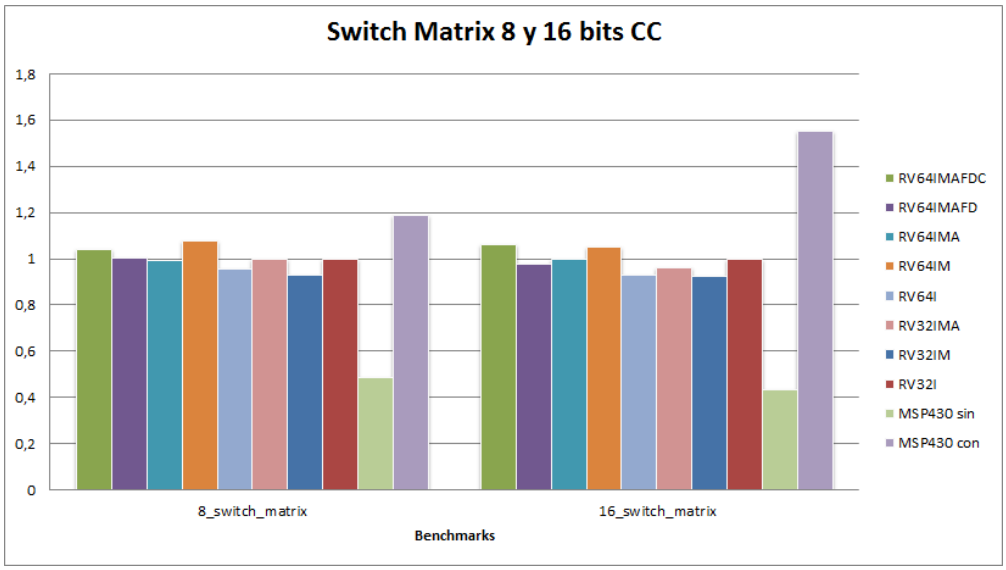

Figura 4.9: Ciclos Contados en el Benchmark Switch Matrix con Datos de 8 y 16 bits simulado en VCS

nes el cual tuvo una velocidad de 0.4 veces la velocidad del RV32I. Los procesadores rocket se mantuvieron con una velocidad muy similar a la del RV32I sin importar su ancho de palabras o las extensiones que los componían. Esto muestra que las extensiones implementadas no mejoran el rendimiento de los accesos a memoria.

En la figura 4.10 se muestra los CC y IC contados en la ejecución del filtro FIR. Este programa se separó del suit puesto que la herramienta de compilación para las arquitecturas de RV64I, RV64IM y RV64IMA fallaba realizando por un salto ilegal en la simulación. Este tipo de filtro ejecuta operaciones de multiplicación, suma y acumulación en coma flotante por lo cual se usa para medir el desempeño en este tipo de instrucciones matemáticas.

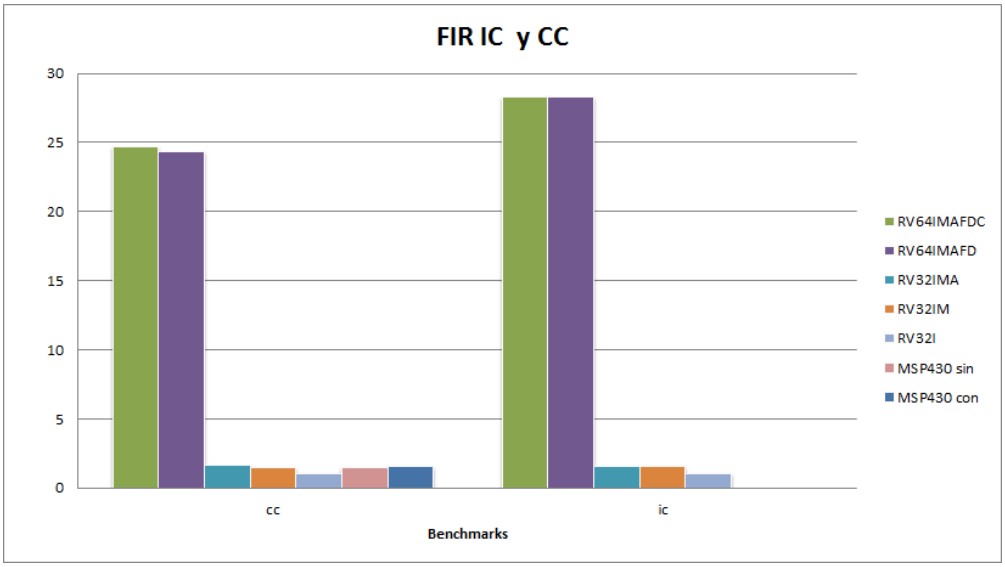

Figura 4.10: Ciclos y Instrucciones Contadas en la Ejecución del Filtro FIR Simulado en VCS

Como se observa en la gráfica los procesadores RISC-V implementados con las extensiones F y D son aproximadamente 25 veces más rápidos que el RV32I y también más veloces que los procesadores MSP430. Además que realizan 24 veces menos instrucciones que los procesadores Rocket sin esas extensiones.

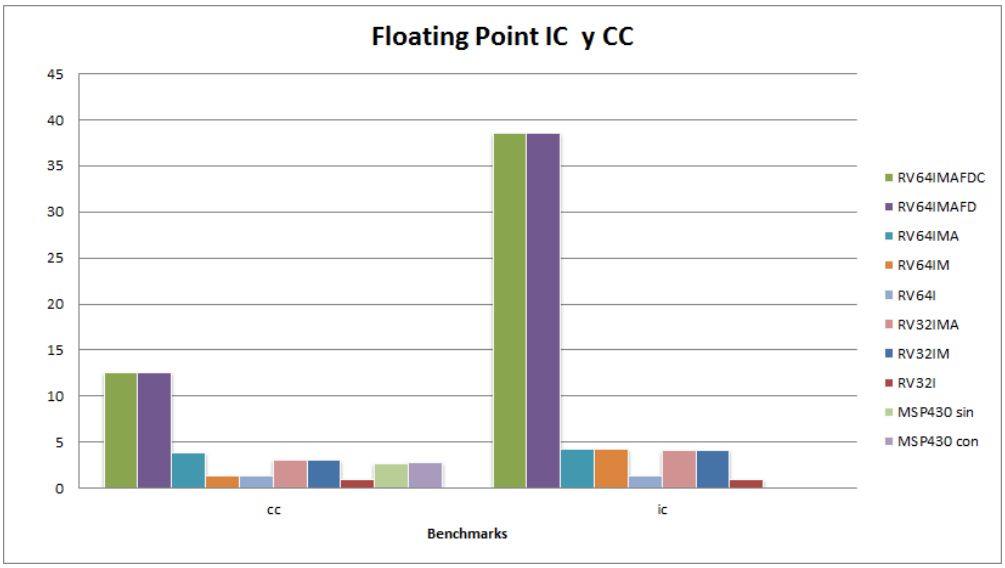

Figura 4.11: Ciclos y Instrucciones Contadas en la Ejecución de la Matemática de Coma Flotante simulado en VCS

En la figura 4.11 se muestran los CC y IC de la ejecución del programa Floating Point. Este programa ejecuta una suma, una multiplicación y una división en coma flotante. Se muestra que los procesadores más rápidos al igual que en el caso del filtro FIR son aquellos que tienen la extensión FD, la cual los hace aproximadamente 12 veces más rápido que el RV32I. También se ven una leves mejoras cuando los procesadores tienen la extensión M, esta mejora los hace similarmente igual de rápidos que los MSP430. Los procesadores RV32I

y RV64I tienen el peor desemepeño y similar entre ellos. A nivel de instrucciones mantiene un comportamiento igual que en CC donde las aquitecturas RV64IMAFDC y RV64IMAFD son las que tienen menos instrucciones, en la mitad estan las arquitecturas con extensión M y las que ejecutan un mayor número son los procesadores RV64I y RV32I.

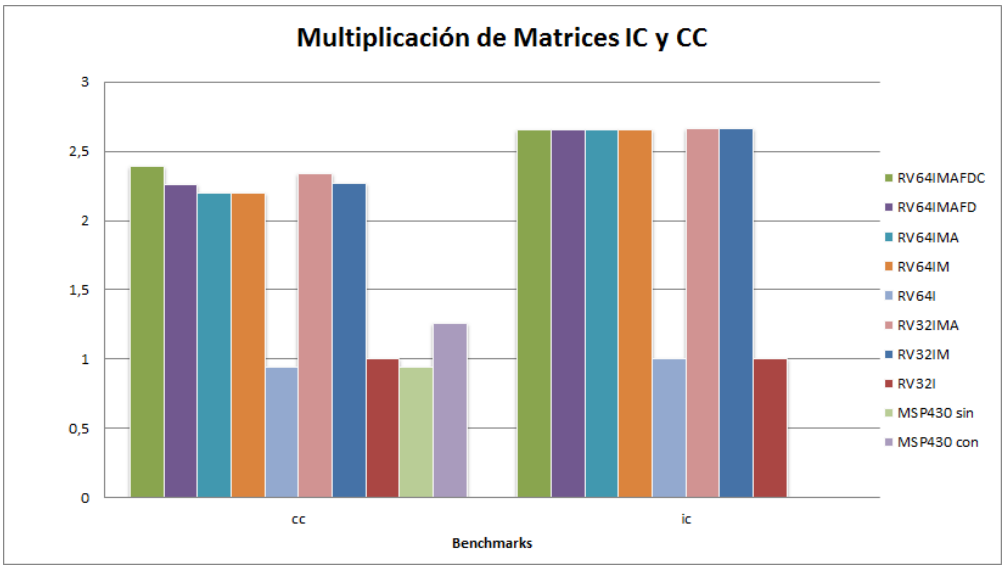

Figura 4.12: CC y IC del Programa de Multiplicación de Matrices en la simulación de VCS

En la figura 4.12 se muestran los CC y IC de la ejecución del programa llamado multiplicación de Matrices. Este programa como lo dice su nombre ejecuta la multiplicación de dos matrices la cual lleva acabo por medio de operaciones de multiplicación, suma y acumulación Como se observa los procesadores RISC-V con la extensión M son los más rápidos, siendo 2.2 veces más veloces que el procesador RV32I. Mientras que los procesadores MSP430 junto con los procesadores RV32I y RV64I tienen rendimientos similares. En términos de IC los que menos ejecutan instrucciones son los procesadores RISC-V con las extensiones M mientras.

En la figura 4.13 se muestran los CC y IC compactados por la media aritmética en la ejecución del suit MSP430. En 4.13 se observa que los procesadores más rápidos son los RISC-V con extensiones F y D, los cuales son aproximadamente 2 veces más veloces que el RV32I. Seguido de ellos se encuentran los MSP430 y los procesadores RISC-V con la extensión M; los cuales son aproximadamente 1.5 veces más rapidos que el RV32I. Y en último lugar estan los procesadores RV64I y RV32I con velocidades similares entre ellos. La media aritmética de las IC tiene un comporamiento similar donde los procesadores RV64IMAFDC y RV64IMAFD ejecutan 2.6 veces menos instrucciones que el procesadore RV32I y REV64I. Mientras que los procesadores con extensión M pero sin las extensiones F y D tienen la mitad de instrucciones que los RV32I y RV64I.

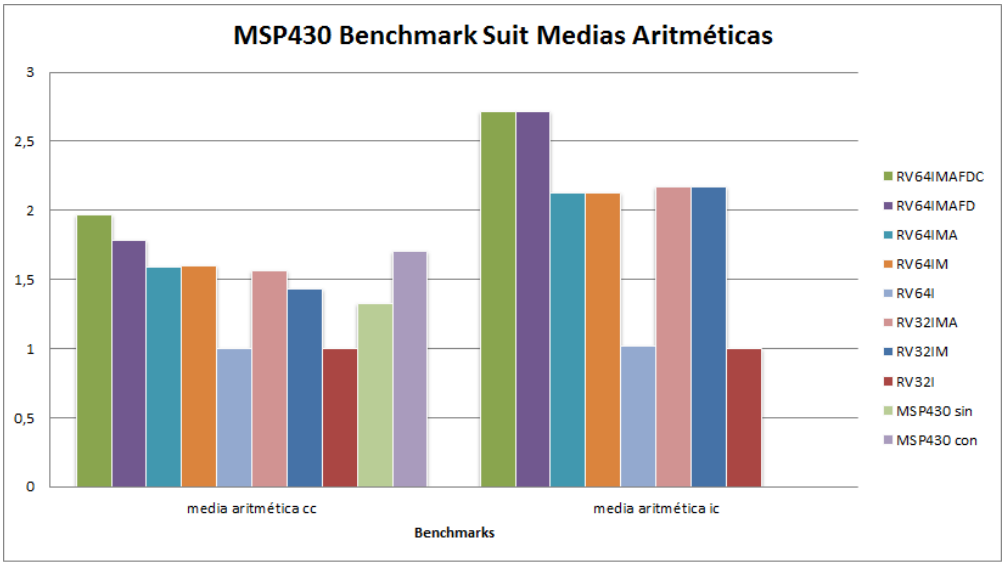

Figura 4.13: Resultados del Benchmarking MSP430 Resumidos con la Media Aritmética simulado en VCS

4.2.2 Implementación Zedboard

En esta subseccion se muestran los resultados prácticos obtenidos por la zedboard y la comparación de la misma con los resultados predichos por los simuladores. Se muestra una media aritmética con los CC e IC bajo el suit del MSP430. Los resultados usados para su cálculo están referenciados respecto a la implementación hecha en Zedboard.

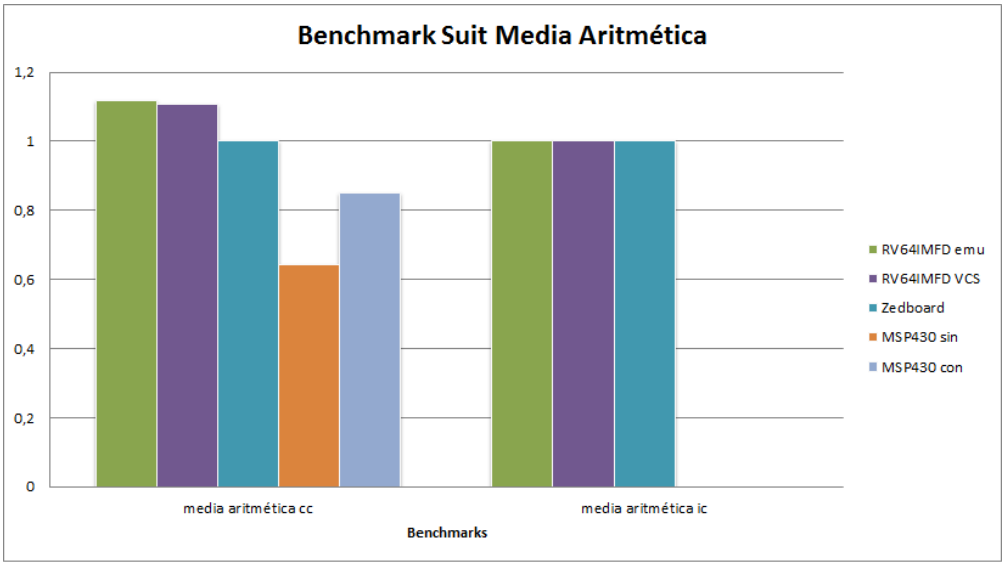

Figura 4.14: Resultados del Benchmarking MSP430 Resumidos con la Media Aritmética Simulado en Zedboard

En la figura 4.14 se observan los IC y CC obtenidos de la implementación en zedboard junto con los resultados de los simuladores y del MSP430. En ella se observa que los resultados de los simuladores y la implementación en Zedboard son más rápidos que los procesadores MSP430. Los resultados de los simuladores son levemente más optimistas y muy similares a los resultados prácticos. A nivel de IC los tres procesadores RISC-V ejecutaron la misma cantidad de instrucciones.

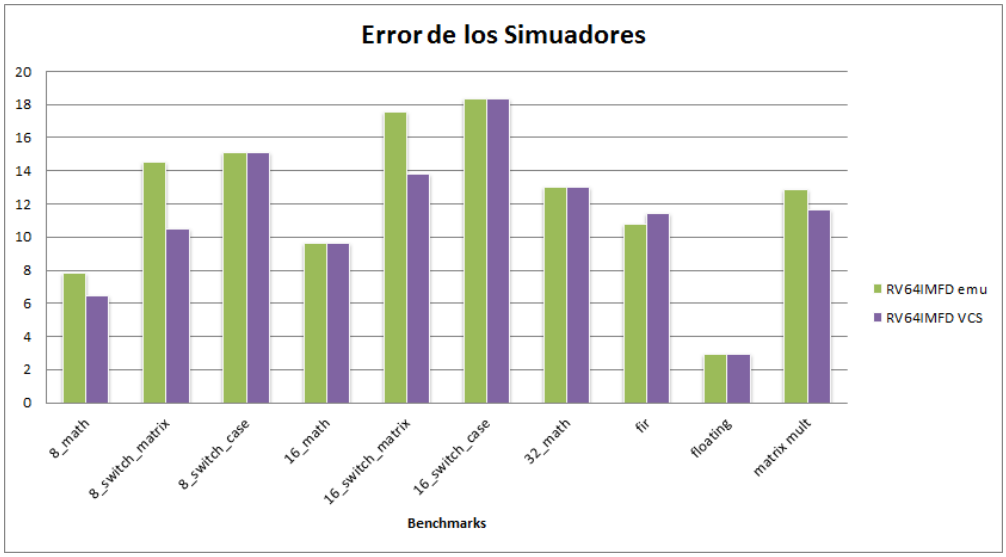

Figura 4.15: Errores de los Simuladores en el Suit del MSP430

En la figura 4.15 se observan los errores en el cálculo de los CC de los simuladores respecto a los datos obtenidos por la implementación en Zedboard. El error de los simuladores ronda entre 2% al 18%, este error es provocado a que las herramientas usan modelos ideales y no consideran detalles de la implementación.Los errores de timing y rompimiento de pipeline son tomados encuenta al momento de la simulación. También es importante detallar que las versiones del procesador practico y simulado difieren un año; actualizaciones reciente de la arquitectura pudieron haber provocado esta brecha entre ellos.

4.3 Dhrystone

En esta sección se muestran los diferentes resultados obtenidos al correr el benchmark dhrystone en el simulador VCS. Los resultados obtenidos en Verilator se encuentran en los anexos debido a su similitud con el otro simulador. Finalmente en la última subsección compara los resultados prácticos obtenidos con los resultados encontrados en los simuladores. También se hace una comparación con otros procesadores similares encontrados en la literatura en base a los DMIPS/MHz que puede ejecutar.

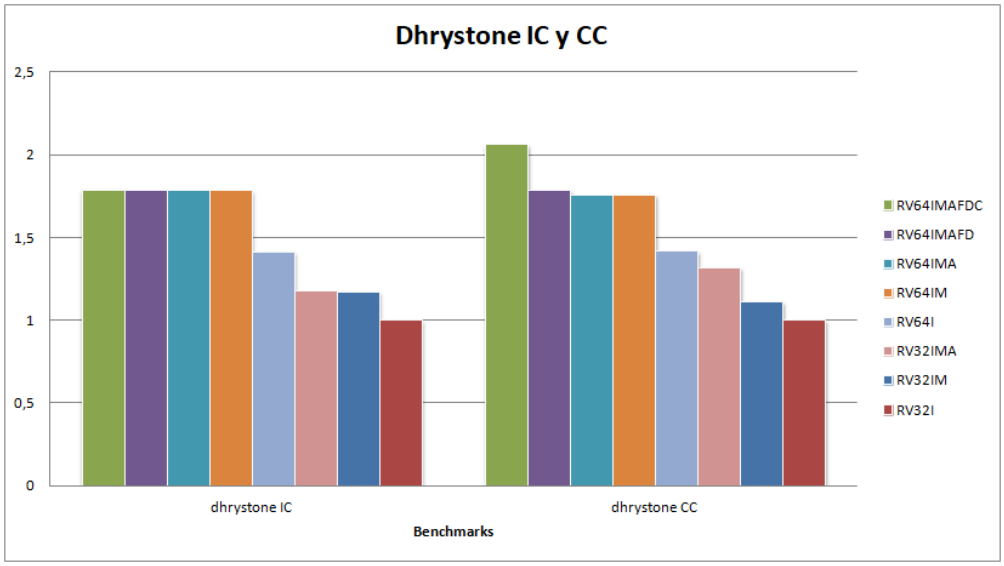

Figura 4.16: Ciclos y Instrucciones Contadas en Dhrystone Simulado en VCS

4.3.1 VCS

En la figura 4.16 se observan los IC y CC obtenidos en el simulador VCS en la realización del programa Dhrystone. Este programa tiene un código main que se ejecutó 500 veces (500 loops) para obtener estos datos y mide el desempeño en procesar strings y operaciones con datos tipos enteros. Como se observa en la gráfica las instrucciones contadas no varían para las arquitecturas que tienen la extensión M. También se observa que los procesadores con un ancho de palabra de 64 bits son más rápidos que sus equivalentes de 32 bits, y este efecto es el predominante. El segundo efecto predominante es la presencia o no de la extensión M el cual da una leves bonificaciones de velocidad en las arquitecturas. La arquitectura más rápida es la RV64IMAFD y la más lenta es RV32I.

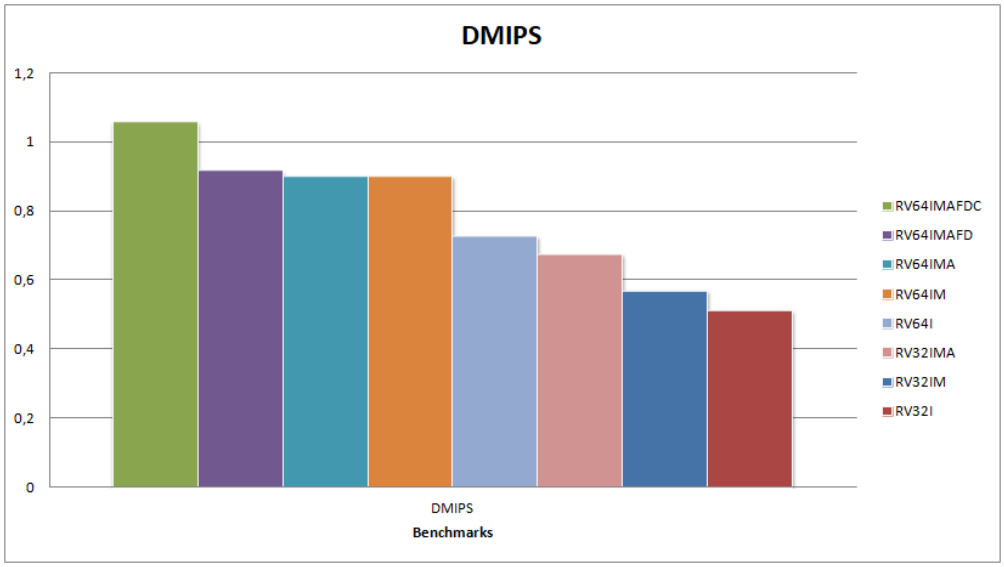

Figura 4.17: DMIPS en Dhrystone Simulado en VCS

En la figura 4.17 se observan los DMIPS hechos por cada una de las arquitecturas RISC-V en la simulación de VCS bajo el programa dhrystone. La mejor ejecución fue lograda por RV64IMAFDC gracias a la bonificación de la extensión C. Al igual que la gráfica 4.19 las arquitecturas de 64 bits tienen un mejor rendimiento que sus equivalentes de 32 bits, el segundo efecto dominante es la presencia de la extensión M la cual eleva 0.1 los DMIPS generados por el procesador. El peor rendimiento es por lo cual el del RV32I que carece de las extensiones M y C y que ejecuta 0.5 DMIPS aproximadamente.

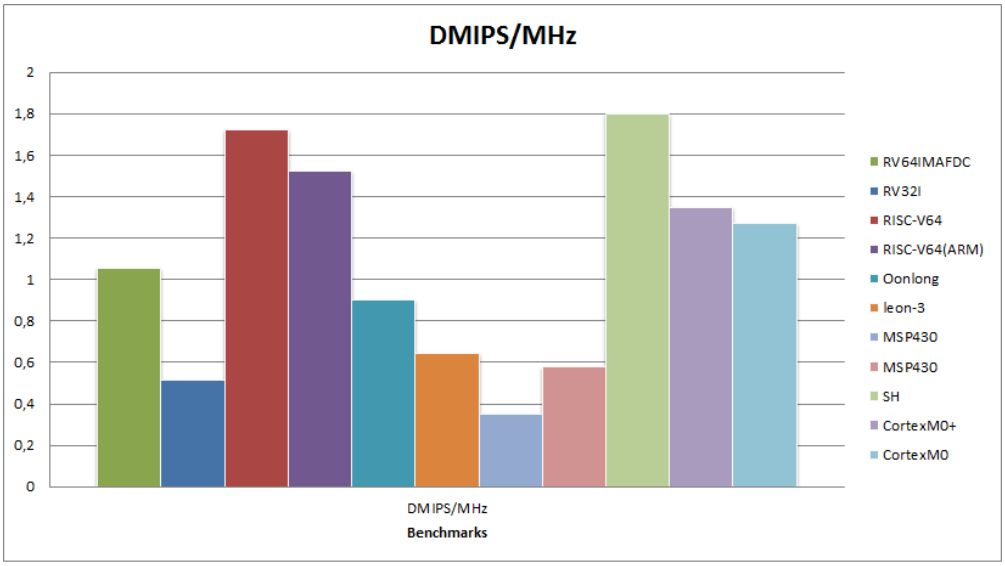

Figura 4.18: Comparación de la Capacidad de DMIPS/MHz con Diferentes Procesadores

En la figura 4.18 se muestra una comparación de los DMIPS/MHz generados por diferentes procesadores con alguna característica similar encontrados en la literatura. En la gráfica se expuso el mejor y el peor de los resultados obtenidos en la simulaciones que corresponden al RV64IMAFD y al RV32I. Se observa que el desempeño en el programa del procesador MSP430 con todas las optimizaciones es levemente mejor que el RV32I. En la gráfica también se observa que los procesadores ARM CortexM0 tienen un mejor desempeño que las arquitecturas RISCV implementadas. Los procesadores leon-3 y Oonlong tienen un rendimiento similar al de las arquitecturas RISC-V implementadas. Los procesadores Rocket simulados tienen un rendimiento intermedio al compararlos con la literatura consultada sobre procesadores que comparten similitudes con ellos.

4.3.2 Implementación Zedboard

En la figura 4.19 se observan los resultados de los IC y CC sobre el benchmark dhrystone en la implementación práctica sobre la zedboard junto con la de los simuladores. Como se observa el valor de los simuladores es bastante similar tanto para IC y CC. En IC el error es prácticamente cero. En CC el simulador VCS es levemente más optimista, mientras que verilator es levemente más fatalista.

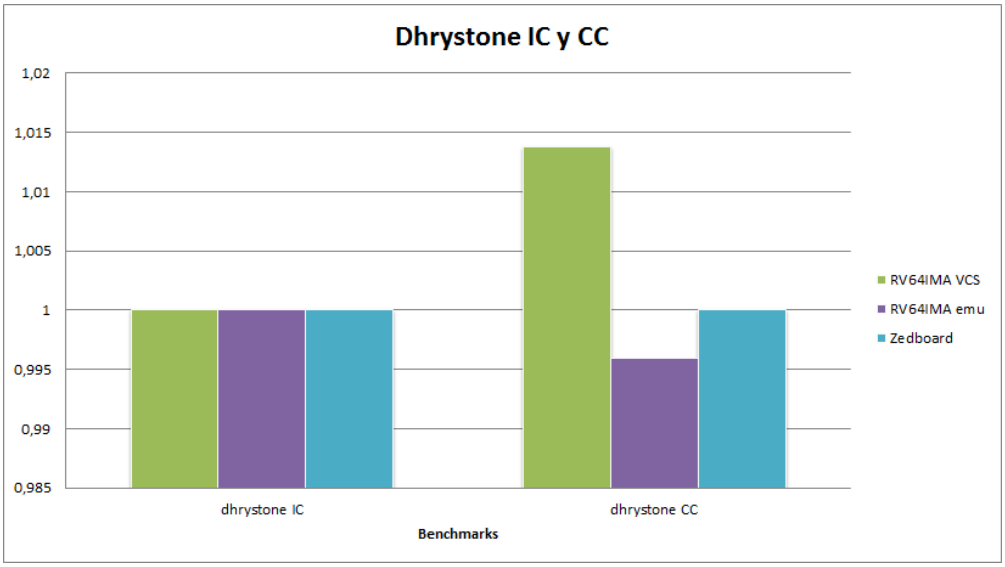

Figura 4.19: Ciclos y Instrucciones Contadas del Benchmark en la Zedboard

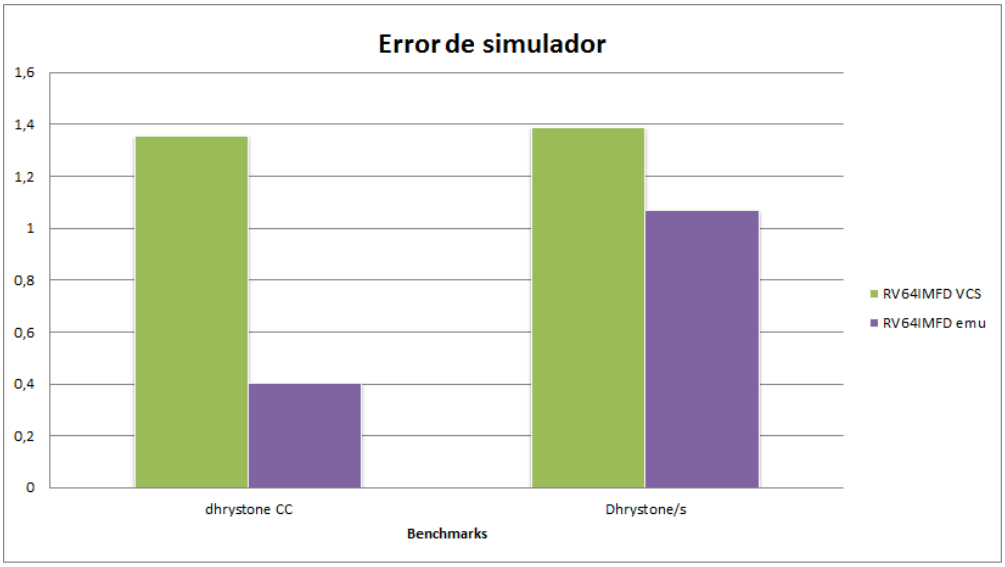

Figura 4.20: Error de los Simuladores sobre el Benchmark Dhrystone

En la figura 4.20 se observan los errores de los simuladores en benchmark dhrystone con el resultado práctico obtenido por medio de la zedboard. Los errores son sumamente pequeños y ambos simuladores son cercanos a predecir estos resultados.

4.4 Reporte de Potencia Obtenido en Vivado

En esta sección se resume por medio del gráfico 4.21 el análisis de potencia hecho por la herramienta Vivado con un factor de actividad de 0.1 [19]. De este análisis se excluyó la potencia dinámica consumida por el procesador ARM dentro Zedboard el cual sirve como sistema anfitrión para el procesador RISC-V. En la gráfica se muestra un consumo dividido por los recursos usados de la zedboard para implementar la arquitectura y la suma de ellos da un total de 0.173 W. Esta potencia dinámica obtenida sirve para hacer una primera estimación del consumo de la arquitectura RV64IMFD.

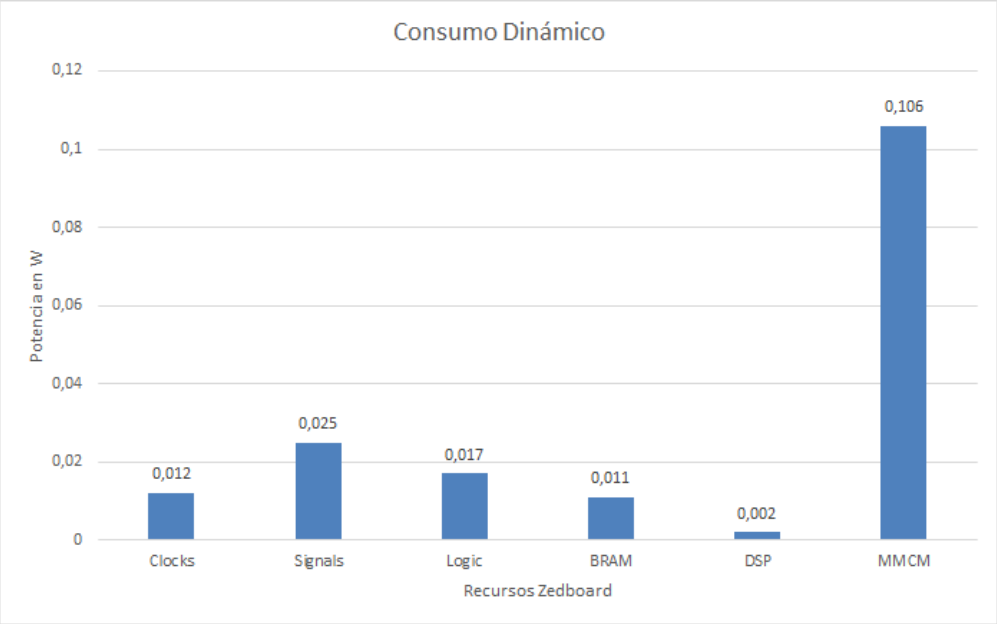

Figura 4.21: Error de los Simuladores sobre el Benchmark Dhrystone

Capítulo 5

Conclusiones

Por medio del suit de RISC-V se observó que los procesadores Rocket tienen un rendimiento similar bajo operaciones de memoria, multiplicaciones de software y suma de vectores. Ninguna de la extensiones agregan una bonificación en velocidad a las arquitecturas en este suit, mientras que un ancho de palabra pequeño beneficia levemente la ejecución del programa de multiplicación de software. Estos significa que las extensiones implementadas no aceleran este tipo de operaciones.

Se comparó las velocidades del procesador MSP430 con y sin optimizaciones con el rendimiento de cada una de las arquitecturas RISC-V implementadas en este documento, por medio de un suit de benchmarks aplicado al procesador de *Texas Instruments*. Los resultados mostraron que las arquitecturas de RISC-V con extensiones F y D son las más rápidas, debido a su alta velocidad en la ejecución de programas con punto flotante. En segundo lugar están las que tienen la extensión M y los procesadores MSP430, todos ellos con un rendimiento bastante similar. Por último están los procesadores con las extension básica I. Este suit ejecuta operaciones matemáticas básicas en punto fijo y flotante, junto con un programa que estresa la memoria y otro que imita a una máquina de estados. El MSP430 fue superior solo en los programas que acceden a memoria de manera concurrente y en la maquinas de estados; mientras que en las operaciones matemáticas los procesadores RISC-V con extensiones F y D tuvieron un mejor rendimiento.

La diferencia entre los CC medidos en los casos simulados y prácticos tuvieron una gran diferencia debido a que las versiones de la arquitectura Rocket del simulador y la práctica del zedboard difieren por un año, además que la implementación práctica toma en cuenta problemas de timing y rompimiento de pipeline. Se observó también que los simuladores fallan más cuando estiman la medición de programas que estresan la memoria; esta instrucciones en la practica son más lentas y consumen más ciclos de reloj. En el caso de la IC los simuladores tuvieron valores cercanos a cero en su estimación.

Por medio del reporte de potencia hecho en Vivado se obtuvo una estimación del consumo dinámico de la arquitectura implementada en la Zedboard. Se obtuvo un consumo dinámico de 173 mW.

Por medio del programa Dhrystone se compararon las velocidades de las arquitecturas con procesadores similares encontrados en la literatura. Los procesadores ARM Cortex M0 lideran en velocidad de procesamiento mientras que tiene un consumo ultra low-power. Sin embargo los procesadores RISC-V implementados no se alejan mucho del rendimiento de ellos, y le ganan al procesador MSP430 en DMIPS/MHz.

Recomendaciones

Para facilitar la simulación y compilación benchmarks se recomienda usar los scripts encontrados en las herramientas. Estos compilan los test con el set de instrucciones indicado, construye la arquitectura deseada y reportan los resultados en la terminal. También se recomienda que cualquier error obtenido sea preguntado en el git donde se descarga la herramienta, pues ellos la construyeron y tienen gran conocimiento en ella.

Se recomienda al inicio seguir el tutorial encontrado en `https://github.com/freechipsproject/rocket-chip` para entender rapidamente el funcionamiento y flujo herramienta Rocketchip.

En trabajos futuros se recomienda comprender a profundidad cada unos de los scripts, observar paso a paso que hacen para llamar a los simuladores y construir las arquitecturas. Además de correr el benchmark Coremark el cual es mucho mejor que el Dhrystone para medir rendimiento en procesadores y comparar sus resultados. Otro trabajo para un futuro es realizar la implementación de las arquitecturas a nivel de compuertas asociándola a una tecnología y establecer el consumo de ellas. No se pudo llevar la simulación a nivel de compuerta ya que la librerías con las que se cuentan no incluían memorias y el diseño del Rocket las necesita.

Bibliografía

[1] Cortexm0+. https://developer.arm.com/products/processors/cortex-m/cortex-m0-plus. Accessed: 2017-09-6.

[2] Cortexm0. https://developer.arm.com/products/processors/cortex-m/cortex-m0. Accessed: 2017-09-6.

[3] Zedboard. https://reference.digilentinc.com/_media/zedboard:zedboard_ug.pdf, 2012.

[4] Introduction to verilator. https://www.veripool.org/wiki/verilator, 2017.

[5] Risc-v foundation. https://riscv.org/risc-v-foundation/, 2017.

[6] Rocket chip on zynq fpgas. https://github.com/ucb-bar/fpga-zynq, 2017.

[7] Vcs. https://www.synopsys.com/verification/simulation/vcs.html, 2017.

[8] Krste Asanovic Andrew Waterman. *The RISC-V Instruction Set Manual*. SiFive Inc., May 2017.

[9] F. Arakawa. Multicore soc for embedded systems. In *2008 International SoC Design Conference*, volume 01, pages I–180–I–183, Nov 2008.

[10] ARM. Dhrystone benchmarking for arm cortex processors, 2011.

[11] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. The rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[12] Palmer Dabbelt. All aboard, part1: The -march, -mabi, -mtune arguments to risc-v compile. https://www.sifive.com/blog/2017/08/14/all-aboard-part-1-compiler-args/, 2017.

[13] John L. Hennessy David A. Patterson. *Computer Organization And Design*. Morgan Kaufmann Publishers, risc-v edition edition, 2017.

[14] A. Kchaou, W. El Hadj Youssef, and R. Tourki. Performance evaluation of multicore leon3 processor. In *2015 World Symposium on Computer Networks and Information Security (WSCNIS)*, pages 1–4, Sept 2015.

[15] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanović, and K. Asanović. A 45nm 1.3ghz 16.7 double-precision gflops/w risc-v processor with vector accelerators. In *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*, pages 199–202, Sept 2014.

[16] D. Markovic, C. C. Wang, L. P. Alarcon, T. T. Liu, and J. M. Rabaey. Ultralow-power design in near-threshold region. *Proceedings of the IEEE*, 98(2):237–252, Feb 2010.

[17] C. A. R. A. Melo and E. Barros. Oolong: A baseband processor extension to the risc-v isa. In *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 241–242, July 2016.

[18] Jorge Márquez. Procesamiento y análisis de señales e imágenes. http://www.academicos.ccadet.unam.mx/jorge.marquez/cursos/imagenes_neurobiomed/Mediana_filtro.pdf, 2017.

[19] David Money Harris Neil H.E. Weste. *CMOS VLSI DESIGN*, chapter 5, page 186. 4 edition, 2011.

[20] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[21] William Stallings. *Computer Organization and Architecture*. Pearson Education, ninth edition edition, 2013.

[22] Christos Strydis. Universal processor architecture for biomedical implants. 2011.

[23] Texas Instruments. *Consumer Medical Applications Guide*, 2010.

[24] Kripasagar Venkat William Goh. Msp430 competitive benchmarking. Technical report, Texas Instruments, 2006.

Apéndice A

Resultados del MSP430 benchmark suit en Verilator

En esta sección se exponen los resultados del suit del MSP410 en Verilator.

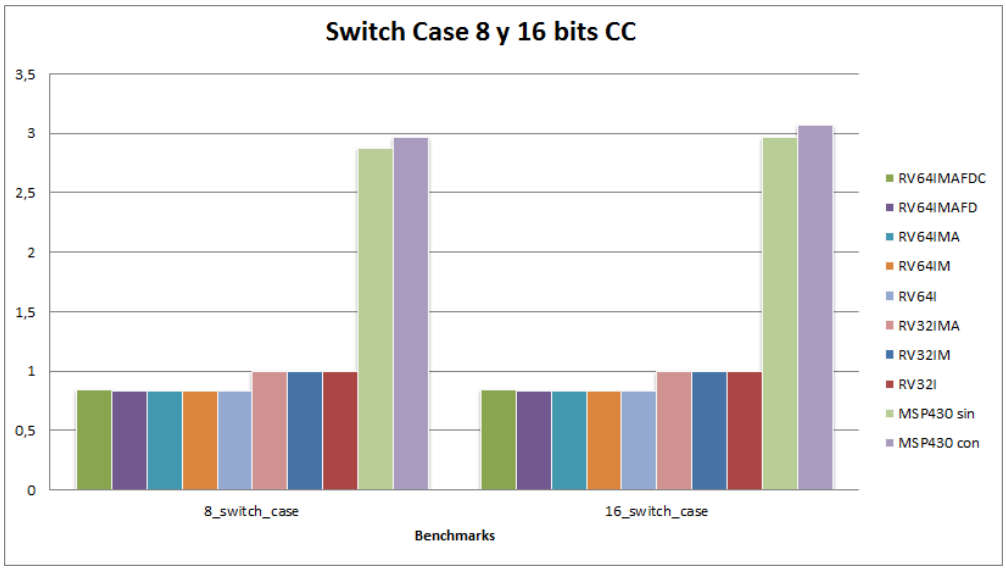

Figura A.1: Ciclos de Reloj Contadas en Switch Case Simulado en Verilator

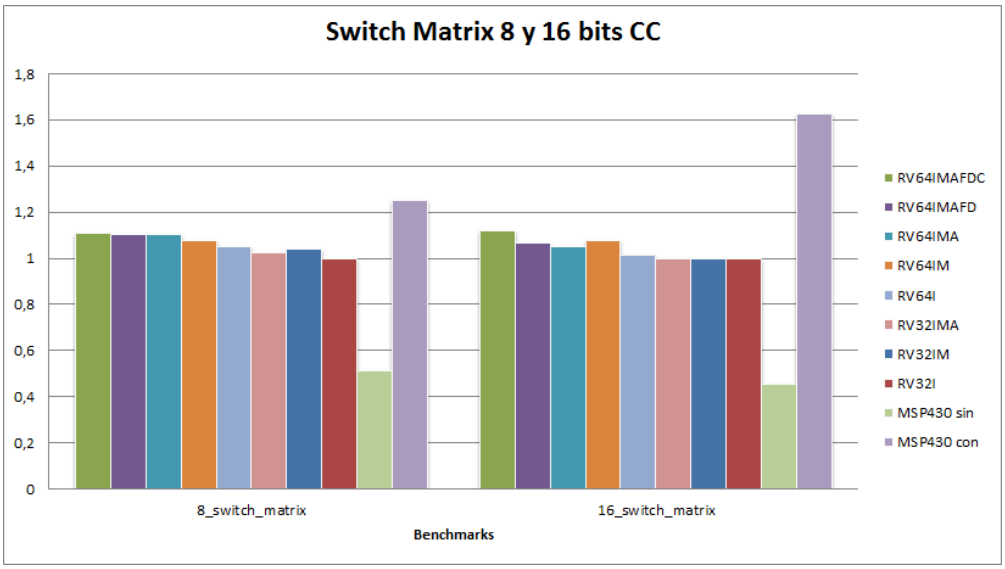

Figura A.2: Ciclos de Reloj Contadas en Switch Matrix Simulado en Verilator

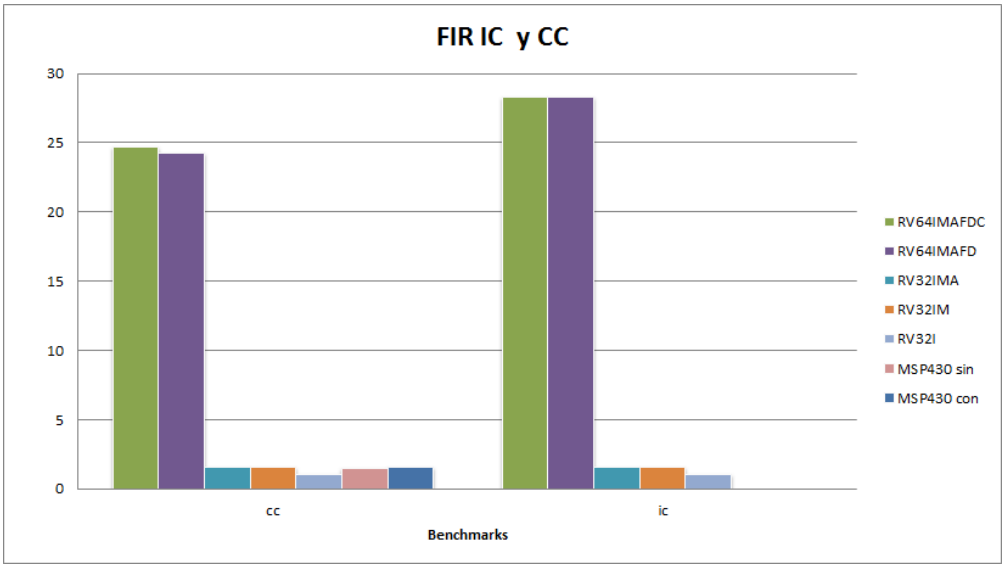

Figura A.3: Instrucciones y Ciclos Contadas en el Filtro FIR Simulado en Verilator

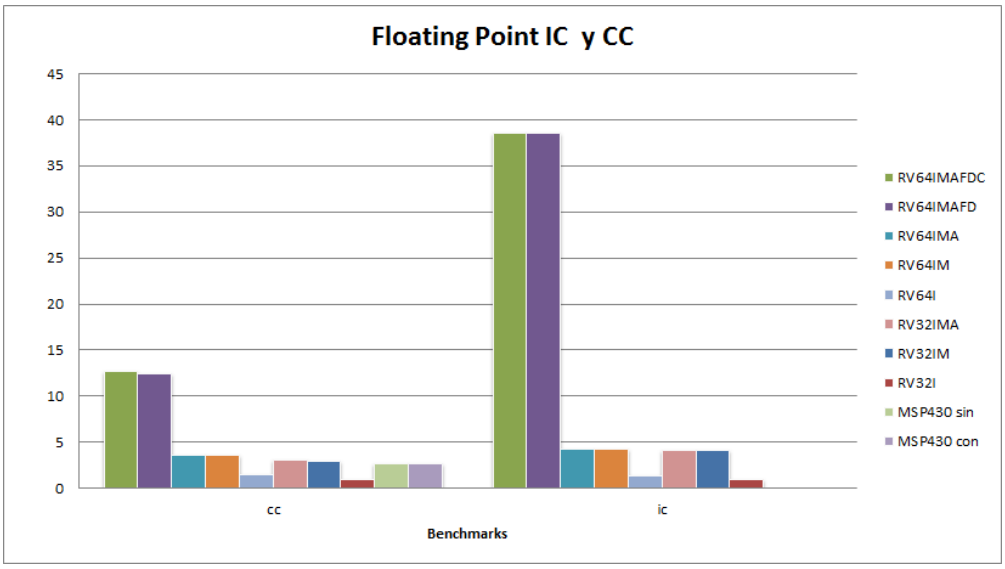

Figura A.4: Instrucciones y Ciclos Contadas en Floating Point Simulado en Verilator

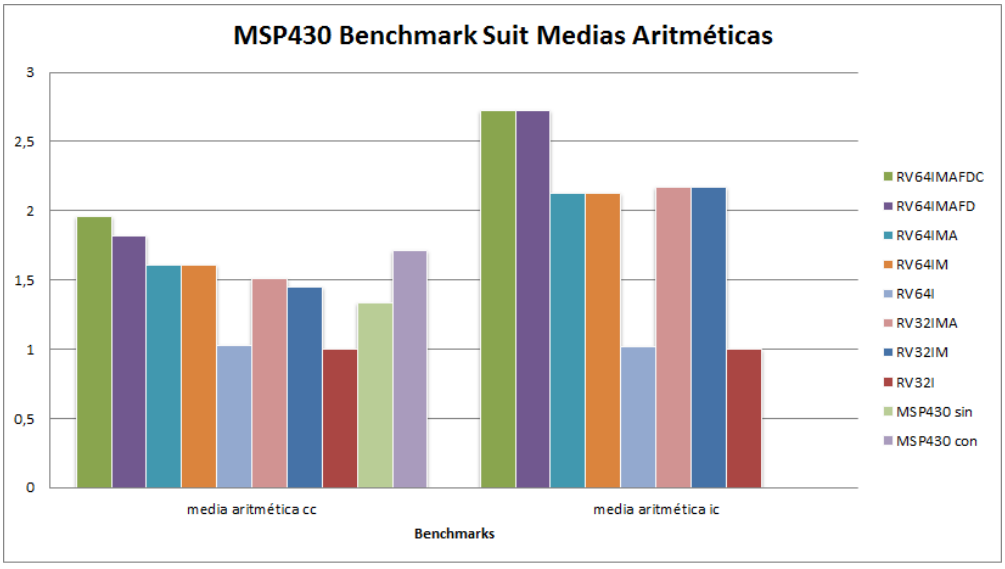

Figura A.5: Media aritmética del suit MSP430 Simulado en Verilator

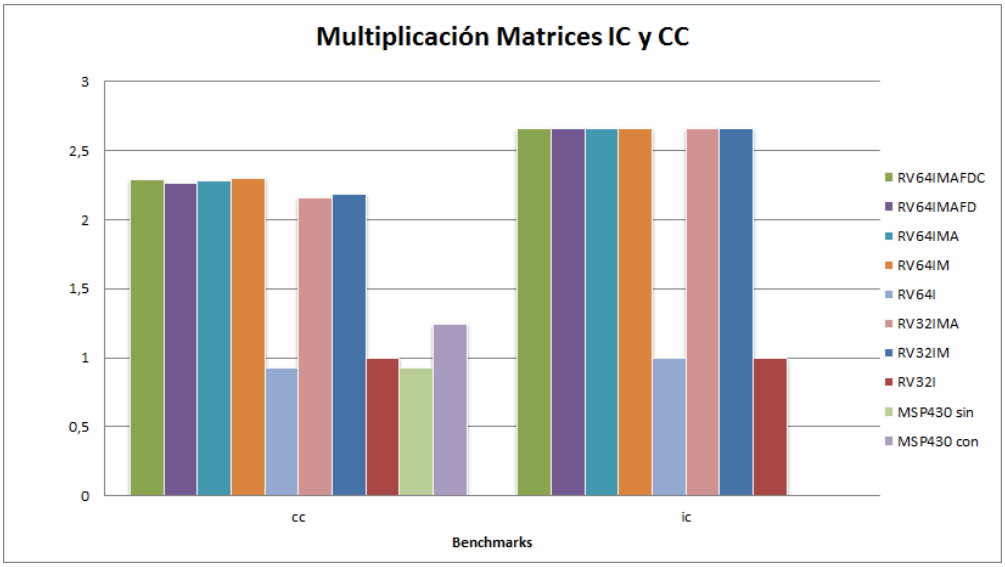

Figura A.6: Instrucciones y Ciclos Contadas en Matrix Mult Simulado en Verilator

Apéndice B

Suit de RISC-V

En esta sección se muestran los resultados en VCS y Verilator sobre el suit de RISC-V que no se incluyeron en los resultados.

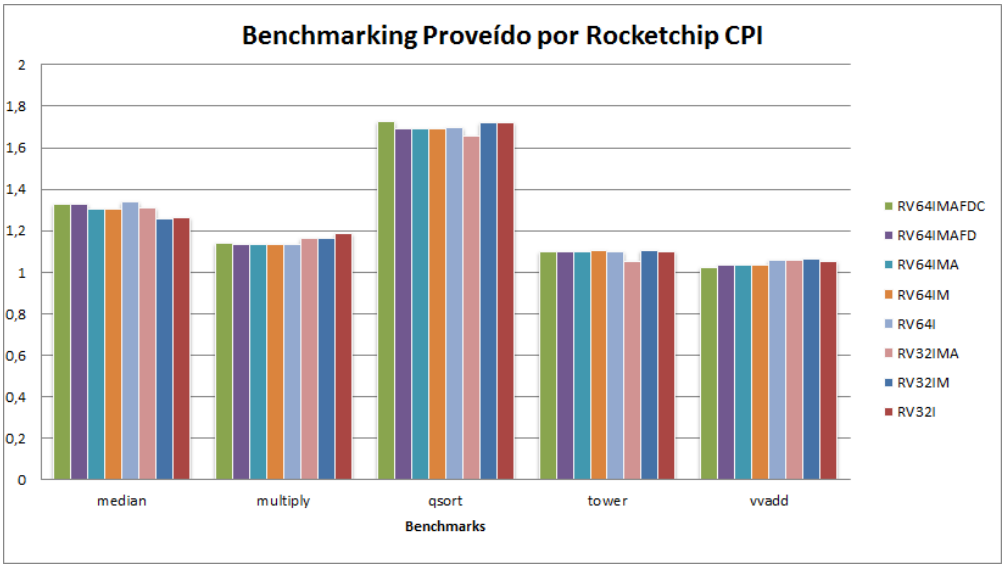

Figura B.1: CPI en el Suit de RISC-V Simulado en VCS

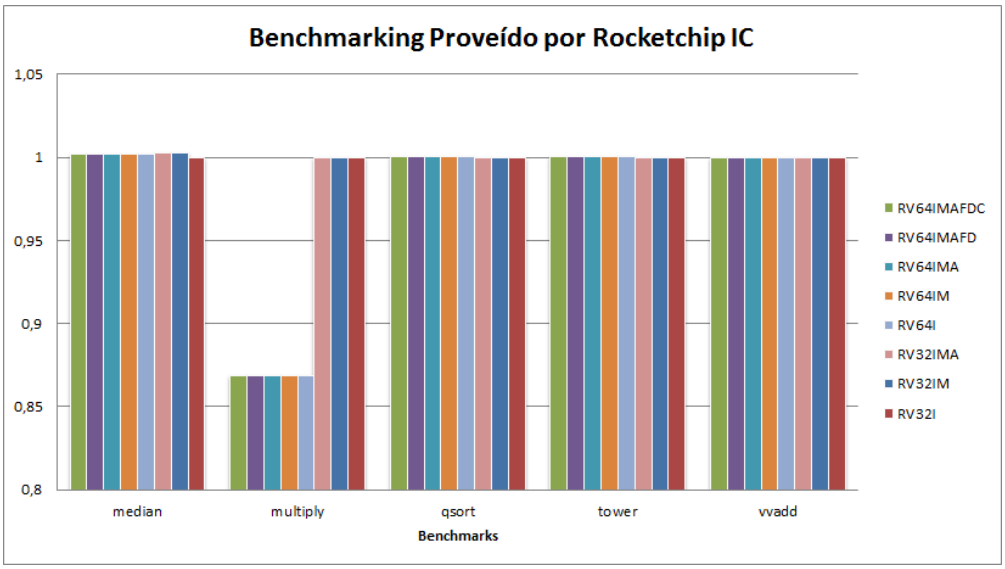

Figura B.2: IC en el Suit de RISC-V Simulado en VCS

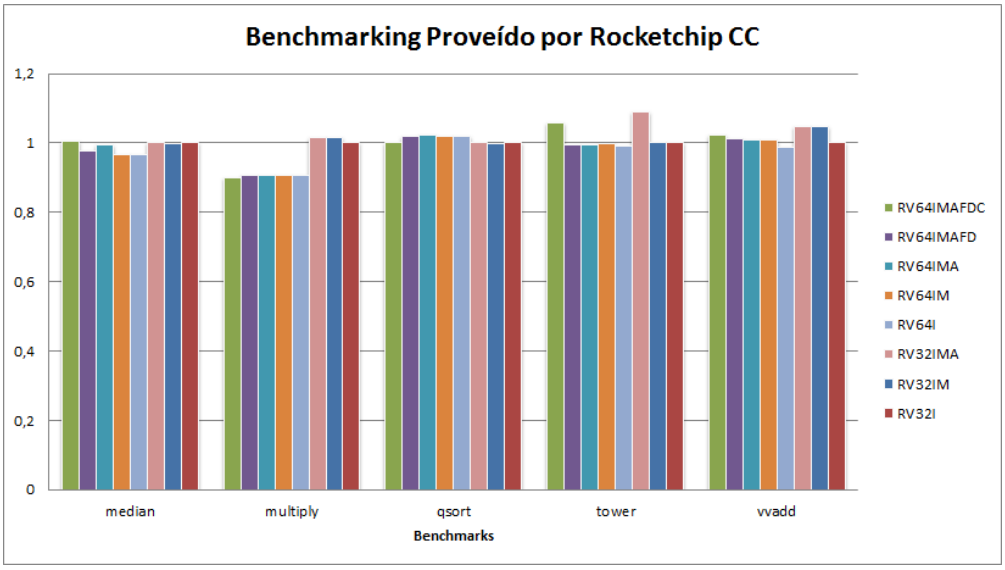

Figura B.3: CC en el Suit de RISC-V Simulado en VCS

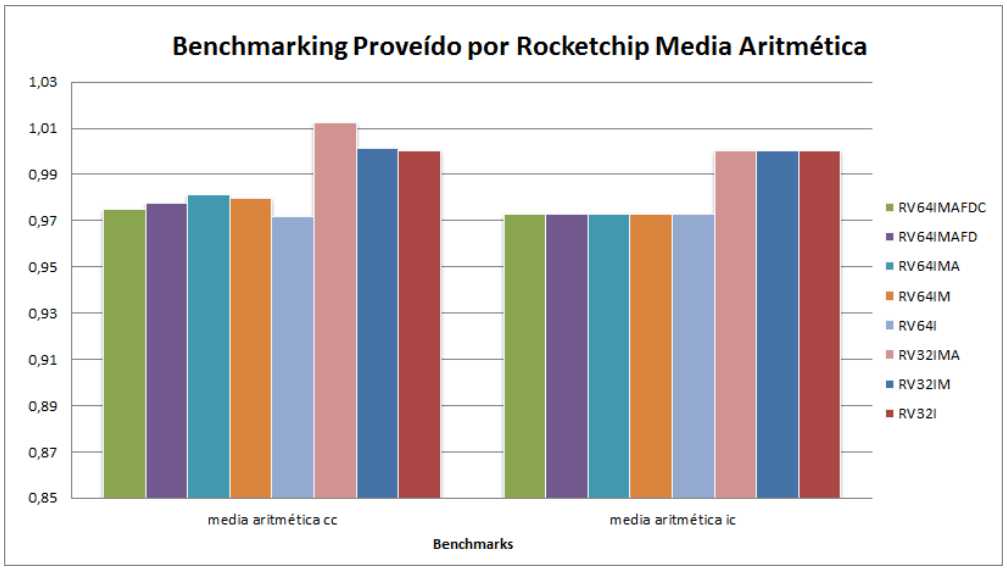

Figura B.4: Media Aritmética en el Suit de RISC-V Simulado en VCS

Apéndice C

Dhrystone

En esta parte se exponen los resultados complementarios del programa dhrystone simulado en Verilator.

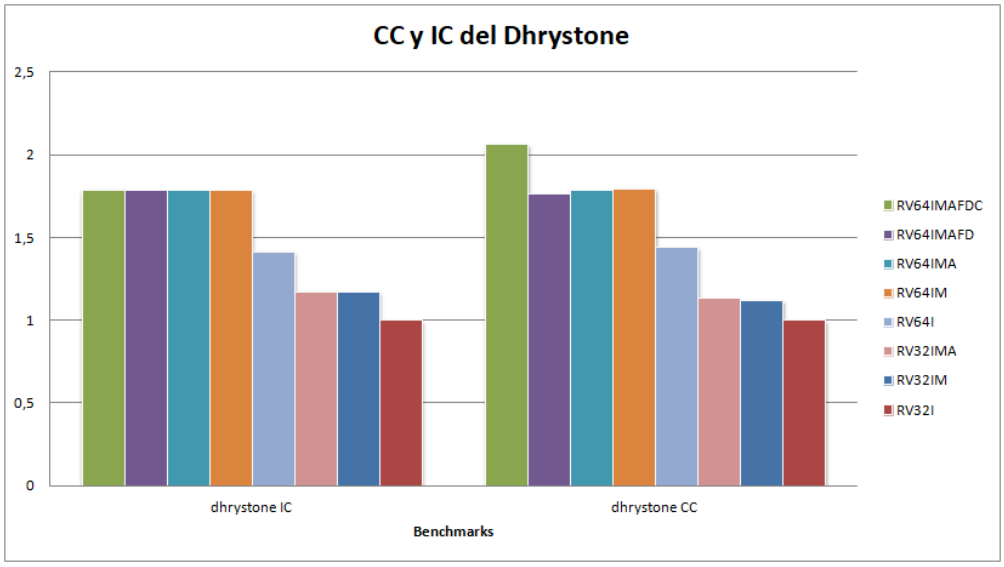

Figura C.1: IC y CC del Benchmark Dhrystone en Verilator

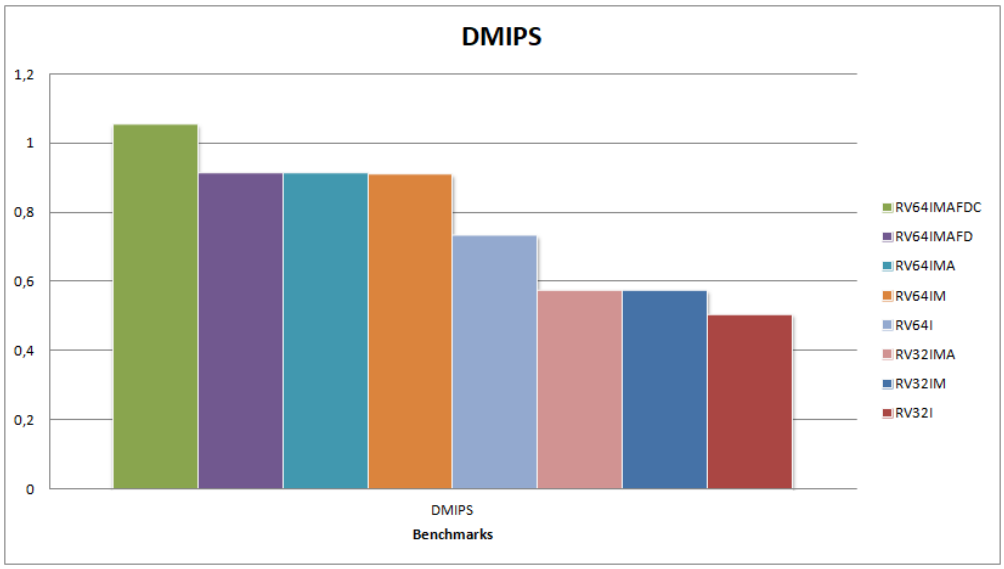

Figura C.2: DMIPS del Benchmark Dhrystone en Verilator

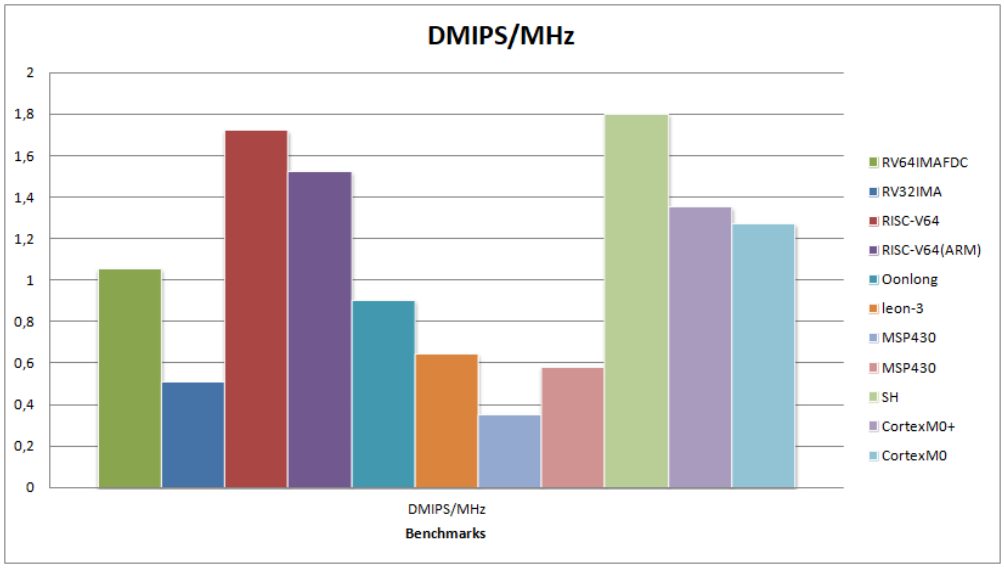

Figura C.3: DMIPS/MHz del Benchmark Dhrystone en Verilator