Facultad de Ciencias

RISC-V
UN ISA DE CÓDIGO ABIERTO
(RISC-V AN OPEN SOURCE ISA)

Trabajo de Fin de Máster
para acceder al

MÁSTER EN INFORMÁTICA

Autor: Elena Zaira Suárez Santamaría

Director: María del Carmen Martínez Fernández

Co-Director: Cristóbal Camarero Coterillo

Octubre – 2019

Agradecimientos

A mi familia, sin su ayuda no podría haber terminado este trabajo. En especial a mis padres por cómo me han educado y apoyado.

Y también quiero dar las gracias a Carmen por su ayuda y orientación durante la carrera.

RESUMEN

Desde 2010 se encuentra disponible un nuevo *Instruction Set Architecture* (ISA) conocido como RISC-V, que destaca principalmente por tratarse de una arquitectura abierta, cuyo principal objetivo es convertirse en universal y que persista. Dicho objetivo ha hecho que el desarrollo de los aspectos técnicos de RISC-V esté condicionado por su propósito de persistencia.

En esta tesis fin de máster se realizará un estudio de la arquitectura RISC-V, cuáles son sus características principales y qué recursos software y hardware podemos encontrar actualmente. Para ello, estudiaremos el ISA RISC-V, cómo está diseñado y cuáles son sus características e instrucciones, así como qué lo diferencia de otras arquitecturas existentes. También analizaremos cuáles son los *system on chip (*SoCs*)* que hay actualmente disponibles en el mercado y sus características. Por otro lado, estudiaremos qué tipo de software hay para RISC-V, tanto sistemas operativos como herramientas para desarrolladores.

Por último, aplicaremos parte de los conocimientos adquiridos y del estudio realizado anteriormente. Para ello, analizaremos la placa HiFive1, basada en la arquitectura RISC-V. Se realizará el desarrollo de un programa que nos permita la conexión de un dispositivo externo compuesto por varios leds (principalmente utilizado con Arduino o Raspberry Pi), y que nos ayudará a validar el correcto funcionamiento del hardware y software disponible, además de descubrir algunas de las muchas posibilidades que esta nueva arquitectura ofrece.

Palabras clave: RISC-V, arquitectura RISC-V, HiFive, ARM.

SUMMARY

Since 2010, a new Instruction Set Architecture (ISA) known as RISC-V is available. This ISA stands out mainly for being a free architecture, whose main objective is to become a universal ISA since it is an open source ISA. This objective has implied that the development of the technical aspects of RISC-V is conditioned by its purpose of persistence.

In this master thesis, we will study the RISC-V architecture, its main characteristics and which software and hardware is available. For this, we will study the ISA RISC-V and how it has been designed. Also, we will highlight the main differences from other existing architectures. In addition, we will analyze which SoCs and software, both in operating systems and developing tools, are available.

Finally, we will apply some of the knowledge acquired and studies previously performed by analyzing the HiFive1 board based on the RISC-V architecture. For this, we will develop a program to connect an external device consisting of several LEDs (mainly used with Arduino or Raspberry Pi). This will help us to validate the proper functioning of the hardware and the software and to discover some of the many possibilities that this new architecture can offer.

Keywords: RISC V, architecture RISC V, HiFive, ARM.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

DTIM: Data Tightly Integrated Memory subsystem, 19
harts: Hardware thread, 12
ISA: Instruction set architecture, 9
RISC: Reduced Instruction Set Computing, 9
SO: Sistema Operativo, 21
SoC: System on chip, 3
XLEN: Indica que el número de bits de un registro marcado con estas siglas, puede variar entre 32 o 64 bits, en función de si esta en RV32 o RV64., 14

1. INTRODUCCIÓN

En la actualidad nos encontramos con RISC-V una *arquitectura Reduced Instruction Set Computer* (RISC), libre y con una comunidad en constante crecimiento. Al tratarse de una arquitectura completamente libre, podemos diseñar y fabricar chips utilizando dicha arquitectura, al igual que podemos desarrollar software y posteriormente venderlo, todo ello sin tener ningún coste adicional por su uso.

El proyecto empezó en 2010 en la Universidad de California en Berkeley. RISC-V se trata de un desarrollo derivado de varios proyectos académicos de diseño de computadores, orientado a la investigación y la docencia. David Patterson, lideró en 1980 cuatro generaciones de proyectos RISC, esto inspiro el nombre del RISC más actual de Berkeley llamándole "RISC Cinco". David Patterson junto con Andrew Waterman son dos de los cuatro arquitectos de RISC-V, Andrew Waterman también es el Jefe de Ingeniería y cofundador de SiFive, una startup fundada por los creadores de la arquitectura RISC-V para proporcionar chips basados en RISC-V. La placa HiFive1 que hemos utilizado para nuestro caso de uso, se trata de una de las placas desarrollada por SiFive. Esta información ha sido extraída de (Patterson & Waterman, 2018) (Wikipedia, 2019).

Por lo tanto, estudiar qué es lo que nos puede ofrecer esta arquitectura es un buen comienzo para determinar sus posibilidades, tanto a nivel docente como empresarial, ya que disponer de un buen diseño en la arquitectura de nuestros computadores y sistemas embebidos, siendo además libre y gratuito, nos permitirá no solo tener programas más eficientes y estables, sino abaratar costes y disponer de sistemas mantenibles de forma fácil y estable a lo largo del tiempo.

Por lo tanto, viendo que se está tratando de impulsar esta "nueva" arquitectura, para que pueda llegar al mayor número de desarrolladores y empresas, creemos que sería muy interesante estudiar en que consiste y cuáles serían los beneficios que nos podremos encontrar en el futuro, al entrar esta arquitectura en juego.

1.1. OBJETIVOS

En este trabajo se pretende realizar un estudio de la arquitectura RISC-V, que información y documentación hay disponible, así como el hardware y software que nos podemos encontrar actualmente.

En concreto nos centraremos en los siguientes puntos:

1. Arquitectura RISC-V: ISA, filosofía de la arquitectura, comparación con otras arquitecturas RISC.
2. Soporte software existente a nivel del sistema: SOs, compiladores, debuggers…
3. Soporte hardware: implementaciones hardware actuales de RISC-V.
4. Caso de uso: análisis de la placa HiFive1 basada en RISC-V

No solo veremos en que consiste la arquitectura RISC-V, también compararemos la arquitectura RISC-V con otras arquitecturas ya consolidadas, para poder ver qué diferencias nos podemos encontrar y si realmente, sería interesante desarrollar nuevos sistemas y aplicaciones teniendo como base esta arquitectura.

El objetivo principal es disponer, no solo de un resumen de las características principales de la arquitectura RISC-V, sino de una amplia información de las principales herramientas para desarrolladores que nos podemos encontrar, como pueden ser los manuales, guías, foros, etc... Toda esta información nos permitirá determinar, no solo si RISC-V es una arquitectura competente a nivel comercial, sino si RISC-V sería una arquitectura apta para la docencia, fácil de enseñar, con amplitud de documentación y con las herramientas necesarias para su correcta enseñanza (depuradores, compiladores…) también es interesante ver cuál es la filosofía que impulsa y mueve el desarrollo, mantenimiento y mejora de RISC-V.

2. ARQUITECTURA RISC-V

El ISA RISC-V fue desarrollado originalmente en la división de informática en el departamento de EECS en Berkeley, Universidad de California. Inicialmente fue diseñado para educación e investigación, pero actualmente se está empezando a fomentar su uso comercial.

RISC-V surgió como consecuencia de la realización de varios proyectos académicos de diseño de computadores, orientados principalmente a la investigación y la docencia. En el desarrollo de RISC-V participo David Patterson, profesor de Ciencias de la computación en la Universidad de California, Berkeley desde 1977 (Wikipedia, 2018). David Patterson lideró en 1980 cuatro generaciones de proyectos RISC, siendo uno de los pioneros en la tecnología RISC de microprocesadores, también es autor de cinco libros, dos de los cuales son sobre arquitectura de computadores, los cuales escribió junto al profesor John L. Hennessy y que son utilizados como libros de texto desde 1990. El nombre de "RISC Cinco" está inspirado por pertenecer a la quinta generación de proyectos RISC en la que participa Patterson. Andrew Waterman es otro de los arquitectos de RISC-V, es Jefe de Ingeniería y cofundador de SiFive, una startup fundada por los creadores de la arquitectura RISC-V para proporcionar chips basados en RISC-V. Esta información ha sido extraída de (Patterson & Waterman, 2018) (Wikipedia, 2019).

RISC-V se trata de una arquitectura de tipo RISC, cuyo principal objetivo es convertirse en un ISA universal gracias a tratarse de un ISA abierto, centrado en mantener la estabilidad de RISC-V con una evolución lenta y cuidadosa, fundamentada principalmente en razones técnicas. La Fundación RISC-V fundada en 2015 cuenta ya con más de 325 miembros, en la Figura 2.1 podemos ver algunos de ellos.

Figura 2.1: Miembros pertenecientes al grupo Platino de la Fundación RISC-V (RISC-V Foundation, 2019)

Una de las novedades que aporta RISC-V, es que se trata de un ISA modular. El núcleo fundamental del ISA es el RV32I, el cual ejecuta un stack de software completo. El RV32I nunca cambiará, produciendo estabilidad para los programadores. La modularidad se produce gracias a las extensiones opcionales estándar que el hardware puede incorporar, en función de la necesidad de la aplicación a desarrollar.

Al compilador de RISC-V se le indica mediante las *flags* que extensiones existen en el hardware. La convención es concatenar las letras de las extensiones que son soportadas por el hardware. RV32IMFD está compuesto por las instrucciones base obligatorias RV32I más la multiplicación RV32M junto con punto flotante precisión simple RV32F y RV32D.

Por ejemplo, en la placa HiFive1 (apartado 3.1.1), tenemos un *core* que está compuesto por las instrucciones base obligatorias RV32I, dispone de multiplicación estándar RV32M, operaciones atómicas RV32A e instrucciones comprimidas RV32C, con lo que tenemos que nuestro hardware tiene un ISA RV32IMAC.

A continuación, realizaremos un pequeño análisis de la arquitectura RISC-V. En concreto del núcleo fundamental del ISA, que es el RV32I, los registros e instrucciones de los que dispone, así como los distintos modos en los que se puede encontrar el procesador. La información de este capítulo se trata de un resumen extraído de (Patterson & Waterman, 2018) y (Waterman & Asanovi´c, 2017).

2.1. FORMATO DE INSTRUCCIONES RV32I

A continuación, en la Figura 2.2 se muestran los seis formatos básicos de instrucciones que nos encontramos en RISC-V.

Tipo R: para operaciones entre registros.

Tipo I: para inmediatos cortos y loads.

Tipo S: para stores.

Tipo B: para branches.

Tipo U: para inmediatos largos.

Tipo J: para saltos incondionales.

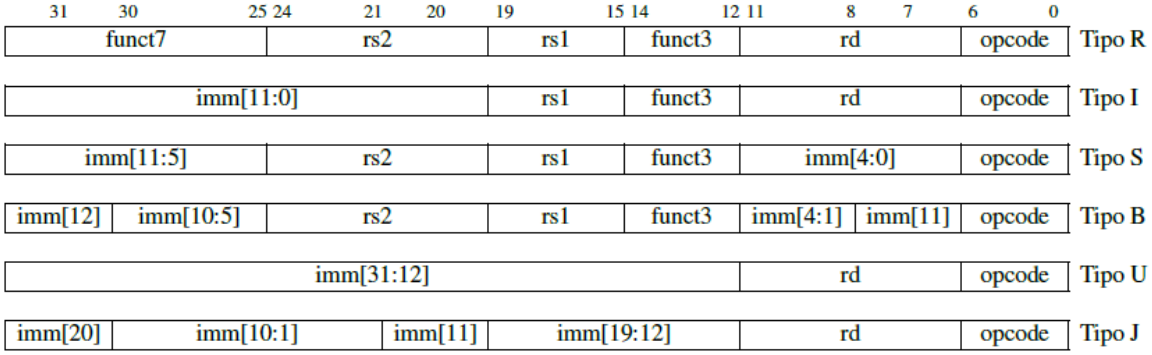

Figura 2.2: Formato de instrucciones de RISC-V RV32I.

En RV32I en las instrucciones se dispone de tres registros, esto está pensado para evitar que el compilador o programador de ensamblador tenga que usar una instrucción adicional para guardar el operando destino. En RV32I como se puede ver en la Figura 2.2 los bits de los registros al ser leídos y escritos están en la misma posición para todas las instrucciones, de esta forma se puede empezar a acceder a dichos registros antes de la decodificación.

2.1.1. Computación Entera

Las instrucciones aritméticas sencillas (add, sub), las instrucciones lógicas (and, xor,or) y las instrucciones de desplazamiento (sll, srl, sra) realizan la función esperada de ellas, leen dos valores de registros de 32 bits y escriben el resultado al registro destino (también hay versiones inmediatas de estas instrucciones).

2.1.2. Loads y Stores

En RV32I disponemos de *loads* y *stores word* de 32 bits (lw, sw) y también bytes y *halfwords*, tanto en versión *signed* o *unsigned* (lb, lbu, lh, lhu, sb, sh). Bytes y *halfwords* con signo hacen *sign-extensión* a 32 bits y son escritos en el registro destino. Esta extensión de datos permite que, aunque el dato sea más corto de 32 bits, las operaciones aritméticas posteriores operen correctamente. Bytes y *halfwords* sin signo, se extienden con cero a 32 bits.

2.1.3. Branches Condicionales

En RV32I se puede comparar dos registros y saltar si el resultado es igual, distinto, mayor o igual (beq, bne, bge, blt), también hay disponibles las versiones sin signo (bgeu, bltu).

2.1.4. Salto Incondicional

La instrucción *jump and link* (jal) en llamadas a funciones, almacena la dirección de la siguiente instrucción PC+4 en el registro destino, normalmente el registro *ra*. Para los saltos incondicionales utiliza el registro cero (x0) en lugar de *ra* ya que no cambia.

En RV32I está también la instrucción *jump and link* con registro (jalr) que puede hacer una llamada a función a una dirección de memoria calculada dinámicamente o retornar de la función usando *ra* como registro origen, y el registro cero como destino. La instrucción jalr es útil para operaciones de switch o case, que calculan la dirección a saltar.

2.2. MODOS DE DIRECCIONAMIENTO

En RISC-V nos encontramos principalmente con dos modos de direccionamiento.

En el caso de los *loads* y *stores* hay un único modo de direccionamiento, este consiste en sumar un valor inmediato de 12 bits a un registro (en x86-32 se denominaba *"modo de direccionamiento por desplazamiento"*).

El segundo modo de direccionamiento que nos encontramos se trata de *"direccionamiento relativo al PC"* y se usa en los *branches*. Este direccionamiento obtiene la dirección a la que tenemos que acceder, multiplicando el valor inmediato de 12 bits por 2 (las instrucciones de RISC-V deben ser múltiplos de dos bytes, debido a la extensión con instrucciones de 16 bytes), le extiende el

signo y lo suma al PC. En la instrucción *jal* se usa este mismo modo también, multiplicando la dirección de 20 bits por 2, se extiende el signo y se suma el resultado al PC para obtener la dirección a saltar.

En RISC-V no hay instrucciones de stack específicas, utilizando un registro como stack pointer, el modo de direccionamiento estándar obtiene la mayoría de los beneficios de las instrucciones *push* y *pop*.

2.3. BANCO DE REGISTROS

En RISC-V hay 31 registros de propósito general x1-x31, para el manejo de enteros. El registro x0 tiene siempre el valor 0. Siguiendo *the standard software calling convention* se usa el registro x1 para guardar la dirección de retorno en una llamada. Para RV32, los registros x son de 32 bits y para RV64 de 64 bits. En la Tabla 2.1 se pueden ver los diferentes registros para RV32I y en la Tabla 2.2. que registros debemos preservar y cuáles no, cuando se realiza una llamada a otra función.

31 … 0	
x0 / zero	Alambrado a cero
x1 / ra	Dirección de retorno
x2 / sp	Stack pointer
x3 / gp	Global pointer
x4 / tp	Thread pointer
x5 / t0	Temporal
x6 / t1	Temporal
x7 / t2	Temporal
x8 / s0 / fp	Saved register, frame pointer
x9 / s1	Saved register
x10 / a0	Argumento de función, valor de retorno
x11 / a1	Argumento de función, valor de retorno
x12 / a2	Argumento de función
x13 / a3	Argumento de función
x14 / a4	Argumento de función
x15 / a5	Argumento de función
x16 / a6	Argumento de función
x17 / a7	Argumento de función
x18 / s2	Saved register
x19 / s3	Saved register
x20 / s4	Saved register
x21 / s5	Saved register
x22 / s6	Saved register
x23 / s7	Saved register
x24 / s8	Saved register
x25 / s9	Saved register
x26 / s10	Saved register
x27 / s11	Saved register
x28 / t3	Temporal
x29 / t4	Temporal
x30 / t5	Temporal
x31 / t6	Temporal
32	
31 … 0	
pc	
32	

Tabla 2.1: Registros de RV32I (Patterson & Waterman, 2018).

Registro	Nombre ABI	Descripción	¿Preservado en llamadas?
x0	zero	Alambrado a cero	—
x1	ra	Dirección de retorno	No
x2	sp	Stack pointer	Sí
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Link register temporal/alterno	No
x6–7	t1–2	Temporales	No
x8	s0/fp	Saved register/frame pointer	Sí
x9	s1	Saved register	Sí
x10–11	a0–1	Argumentos de función/valores de retorno	No
x12–17	a2–7	Argumentos de función	No
x18–27	s2–11	Saved registers	Sí
x28–31	t3–6	Temporales	No
f0–7	ft0–7	Temporales, FP	No
f8–9	fs0–1	Saved registers, FP	Sí
f10–11	fa0–1	Argumentos/valores de retorno, FP	No
f12–17	fa2–7	Argumentos, FP	No
f18–27	fs2–11	Saved registers, FP	Sí
f28–31	ft8–11	Temporales, FP	No

Tabla 2.2: Convención de preservar a través de llamadas a funciones o no (Patterson & Waterman, 2018).

2.4. MODOS DEL PROCESADOR

La mayor parte de las instrucciones que utilizamos habitualmente están disponibles en *modo usuario*, pero hay situaciones en las que es necesario acceder a otros modos más privilegiados (gestionar excepciones, interrupciones y controlar I/O). En RISC-V al igual que en otras arquitecturas, nos encontramos con el *modo máquina* (que ejecuta el código más fiable), y *modo supervisor* (que da soporte a los sistemas operativos).

En la Tabla 2.3 se pueden ver las instrucciones privilegiadas que hay en RISC-V y en la Tabla 2.4 sus opcodes.

machine-mode	trap **ret**urn
supervisor-mode	
supervisor-mode **fence.v**irtual **m**emory **a**ddress	
wait **f**or **i**nterrupt	

Tabla 2.3: Instrucciones privilegiadas en RISC-V.

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
0001000		00010	00000	000	00000	1110011	R sret
0011000		00010	00000	000	00000	1110011	R mret
0001000		00101	00000	000	00000	1110011	R wfi
0001001		rs2	rs1	000	00000	1110011	R sfence.vma

Tabla 2.4. Opcodes de las instrucciones privilegiadas de RISC-V.

2.4.1. Modo Máquina

Se trata del modo más privilegiado, en este modo los *harts*, tienen acceso completo a la memoria, I/O y funcionalidades necesarias para configurar el sistema. Por lo tanto, se trata del único modo de privilegio que se implementa en todos los procesadores RISC-V estándar. Los microcontroladores simples de RISC-V solo soportan el modo máquina, como veremos en el apartado 3.1.1. el modelo HiFive1 dispone de un microcontrolador que solo soporta el modo máquina.

Este modo se caracteriza por que puede interceptar y manejar las excepciones. En la Tabla 2.5. se pueden ver las excepciones que nos podemos encontrar en RISC-V.

Interrupción / Excepción `mcause[XLEN-1]`	Código de Excepción `mcause[XLEN-2:0]`	Descripción
1	1	Interrupción de software de Supervisor
1	3	Interrupción de software de Máquina
1	5	Interrupción de temporizador de Supervisor
1	7	Interrupción de temporizador de Máquina
1	9	Interrupción externa de Supervisor
1	11	Interrupción externa de Máquina
0	0	Dirección de instrucción desalineada
0	1	Fallo de acceso en instrucción
0	2	Instrucción ilegal
0	3	Breakpoint
0	4	Dirección de Load desalineada
0	5	Fallo de acceso en Load
0	6	Dirección de Store desalineada
0	7	Fallo de acceso en Store
0	8	Llamada al entorno desde modo U
0	9	Llamada al entorno desde modo S
0	11	Llamada al entorno desde modo M
0	12	Fallo de página en instrucción
0	13	Fallo de página en Load
0	15	Fallo de página en Store

Tabla 2.5: Causas de excepciones e interrupciones en RISC-V.

En RISC-V hay ocho registros de control y estado (CSRs), a continuación, se muestran cuáles son y para que se utilizan:

mstatus: Contiene el habilitador global de interrupciones, así como otros estados. En la Figura 2.3 se puede ver este CSR. Los únicos campos presentes en procesadores simples, sólo con modo máquina y sin las extensiones F ni V, son el habilitador global de interrupciones MIE y MPIE, el cual después de una excepción contiene el valor anterior de MIE.

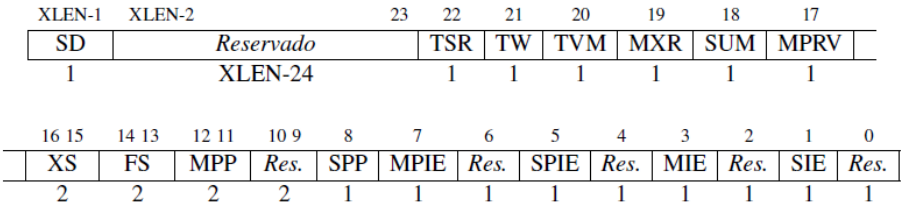

Figura 2.3: CSR mstatus. XLEN es 32 para RV32 o 64 para RV64.

mip: *Interrupciones de Máquina Pendientes*, lista las interrupciones actualmente pendientes. En la Figura 2.4 se puede ver el CSR.

mie: *Habilitador de Interrupciones de Máquina*. Lista que interrupciones puede tomar el procesador y cuáles no. En la Figura 2.4 se puede ver el CSR.

Los CSR de interrupciones máquina son registros de XLEN-bits de lectura y escritura que contienen los bits de interrupciones pendientes (mip) y habilitadores de interrupciones (mie). Solo es posible escribir en los bits correspondientes a las interrupciones del menor privilegio (SSIP), interrupciones de temporizador (STIP) e interrupciones externas (SEIP) en mip por medio de esta dirección CSR, el resto de los bits son de solo lectura.

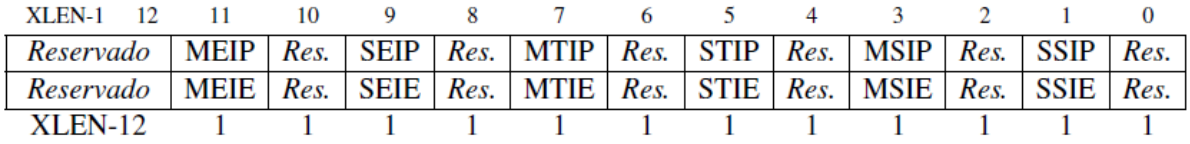

Figura 2.4: CSRs de interrupciones de máquina.
XLEN es 32 para RV32 o 64 para RV64.

mcause: *Causa de Excepción de Máquina*, indica que excepción ha tenido lugar. En la Figura 2.5 se pude ver el CSR, en este se escribe un código que indica el evento que causo la excepción (en el campo código de excepción). El bit de interrupción se pone a 1 si la excepción fue causada por una interrupción. La Tabla 2.5 mapea los valores de los códigos indicando por que se ha producido esa excepción.

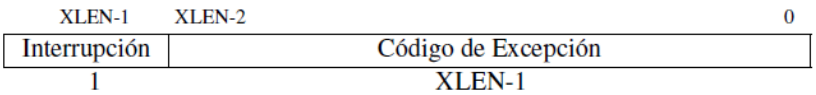

Figura 2.5: CSRs causa de la excepción producida en la máquina y supervisor (mcause y scause).

mtvec: *Vector de Interrupciones de Máquina*, contiene la dirección a la cual salta el procesador cuando ocurre una excepción. En la Figura 2.6 se puede ver el CSR, son registros de XLEN-bits de lectura y escritura que contienen la configuración de vectores de excepciones, que consiste en una dirección base del vector base (BASE) y un modo del vector (MODE). El valor del campo BASE siempre debe estar alineado en una frontera de 4 bytes. MODE = 0 significa que todas las excepciones escriben BASE en el PC. MODE = 1 escribe (BASE + (4 x causa)) en el PC en interrupciones asíncronas.

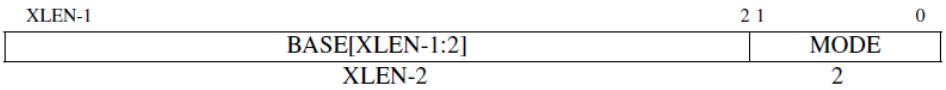

Figura 2.6: CSRs de direcciones base de vectores de excepciones para máquina y supervisor (mtvec y stvec)

mtval: *Valor de Excepción de Máquina*, contiene información adicional de excepciones: la dirección defectuosa para excepciones de direcciones, la instrucción misma para excepciones de instrucción ilegal y cero para otras excepciones. En la figura 2.7 se pueden ver los CSRs.

mepc: *PC de Excepción de Máquina, apunta a la instrucción donde ocurrió la excepción*. En la figura 2.7 se pueden ver los CSRs.

mscratch: *Scratch de Máquina,* contiene una palabra de datos para almacenamiento temporal de manejadores de excepciones. En la figura 2.7 se pueden ver los CSRs.

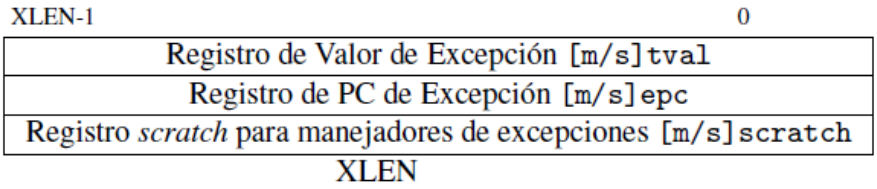

Figura 2.7: CSRs asociados con excepciones e interrupciones.

El modo de privilegio que había antes de la excepción es preservado en el campo MPP de mstatus, y cambiado posteriormente a M. En la Tabla 2.6 se muestra la codificación del campo MPP.

Codificación	Nombre	Abreviación
00	Usuario	U
01	Supervisor	S
11	Máquina	M

Tabla 2.6: Codificación de los niveles de privilegio en RISC-V.

2.4.2. Modo Usuario

Hay situaciones en las que no es adecuado confiar en todo el código de una aplicación, por lo que es necesario restringir el acceso del código que no se considera seguro a las instrucciones privilegiadas, para evitar al programa tener el control del sistema. Para conseguir estas restricciones tenemos disponible el *modo usuario* (Tabla 2.6), que niega el acceso generando una excepción de instrucción ilegal cuando se intenta usar una instrucción o CSR de modo máquina.

Además de no poder acceder a las instrucciones privilegiadas, también se debe impedir que el código no confiable acceda a partes de la memoria no deseadas. Para ello el procesador tiene implementada la funcionalidad PMP (Protección física de la memoria), que consiste en varios registros de dirección (Figura 2.8) y sus registros de configuración correspondiente, indicando si se tiene permiso de escritura, lectura o modificación para esa dirección. Con ella el modo máquina puede saber cuáles son las direcciones de memoria a las cuales puede acceder el modo usuario.

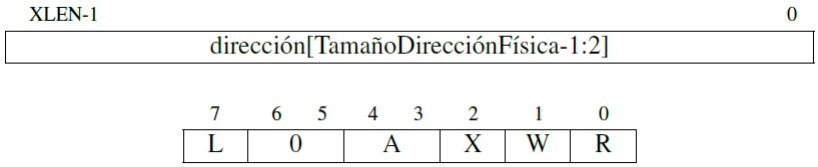

Figura 2.8: Registros de dirección y configuración PMP.
(Patterson & Waterman, 2018)

2.4.3. Modo Supervisor

En este modo nos encontramos con la memoria virtual basada en páginas. Estamos ante un modo de privilegio opcional diseñado para poder soportar sistemas operativos como Linux. El modo supervisor se encuentra a medio camino entre el modo máquina y el modo usuario. No puede utilizar CSRs ni

instrucciones del modo máquina y está sujeto a las restricciones establecidas por el modo PMP.

RISC-V dispone de un mecanismo por el cual las interrupciones y excepciones síncronas pueden ser delegadas al modo supervisor selectivamente, evitando software de modo máquina por completo, permitiendo reducir el tiempo que se invierte en manejar estas excepciones.

El CSR *mideleg* (Delegación de Interrupciones Máquina) indica que interrupciones son delegadas al modo supervisor (Figura 2.9).

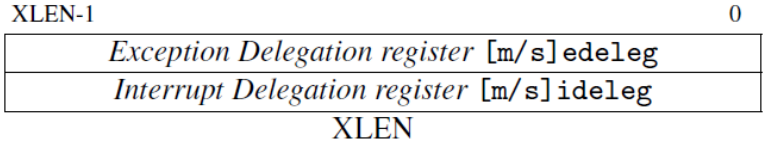

Figura 2.9: Los CSRs de delegación. CSRs de delegación de excepciones e interrupciones de máquina y supervisor (medeleg, sedeleg, mideleg, sideleg) (Patterson & Waterman, 2018)

Las interrupciones delegadas al modo supervisor pueden ser enmascaradas por software en modo supervisor. Los CSRs *sie* (Habilitador de Interrupciones de Supervisor) y *sip* (Interrupciones de Supervisor Pendientes) son CSRs de modo supervisor y subconjuntos de los CSRs *mie* y *mip* (Figura 2.10). Solo en los bits correspondientes a interrupciones que han sido delegadas en *mideleg* es posible leer y escribir con *sie* y *sip*. Los bits correspondientes a interrupciones que no han sido delegadas siempre son cero.

XLEN-1 10	9	8	7 6	5	4	3 2	1	0
Reservado	SEIP	*Res.*	*Res.*	STIP	*Res.*	*Res.*	SSIP	*Res.*
Reservado	SEIE	*Res.*	*Res.*	STIE	*Res.*	*Res.*	SSIE	*Res.*
XLEN-10	1	1	2	1	1	2	1	1

Figura 2.10: CSRs de interrupción de supervisor.

El modo máquina también puede delegar excepciones síncronas al modo supervisor, usando el CSR *medeleg*. Hay que tener en cuenta que las excepciones nunca transfieren el control a un modo menos privilegiado (Una excepción que ocurre en modo supervisor puede ser manejada por el modo máquina o por el modo supervisor, dependiendo de la configuración de delegación, pero nunca por el modo usuario).

El modo supervisor tiene varios CSRs de manejo de excepciones, *scause, stvec, sepc, stval, sscratch y sstatus*. La acción de tomar una excepción es muy parecida al modo máquina. Si un *hart* toma una excepción y ésta es delegada al modo supervisor, el hardware atómicamente pasa por varias transiciones de estados similares, usando CSRs de modo supervisor:

- El PC de la instrucción que causó la excepción es guardado en *sepc* y *stvec* es escrito al PC.
- *scause* recibe la causa de la excepción y *stval* recibe la dirección defectuosa o una palabra con información de la excepción.
- Las interrupciones son deshabilitadas escribiendo SIE=0 en el CSR *sstatus* (Figura 2.11) y el valor previo de SIE es preservado en SPIE.
- El modo de privilegio pre-excepción es preservado en el campo SPP de *sstatus* y el modo de privilegio es cambiado a S.

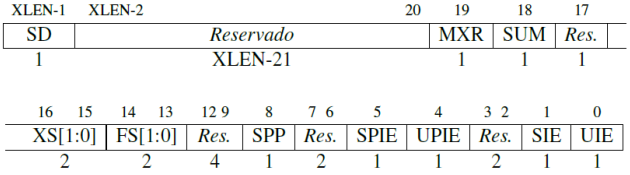

Figura 2.11: CSR sstatus. (Patterson & Waterman, 2018)

2.5. COMPARACIÓN DE RISC-V CON OTRAS ARQUITECTURAS

A continuación, comentaremos algunas de las diferencias que nos podemos encontrar en relación con otras arquitecturas conocidas, teniendo en cuenta las características del ISA para RV32I que hemos expuesto en los apartados 2.1, 2.2. y 2.3.

Para empezar en RV32I nos encontramos con que disponemos únicamente de seis formatos de instrucciones y todas las instrucciones son de 32 bits, esto permite simplificar la decodificación de las instrucciones. En x86-32 hay muchos formatos, dificultando la decodificación.

En cuanto a los registros, RV32I dispone de 31 registros, más x0 como ya vimos en el apartado 2.3. Sin embargo, ARM-32 tiene 16 registros y x86-32 dispone sólo de 8 registros.

En RV32I no se incluye multiplicación ni división, estas opciones están disponibles en las extensiones opcionales RV32M, lo que permite, en el caso de no necesitar disponer de estas utilidades, reducir el tamaño de los chips embebidos. No ocurre igual en x86-32 ni en ARM-32, este último no tiene división, pero si multiplicación.

Aunque RV32I disponga de muy pocos modos de direccionamiento, si puede imitar algunos modos existentes en el x86-32, si se deja el valor inmediato a cero, es igual al modo registro-indirecto. RISC-V tampoco tiene instrucciones de stack específicas como si ocurre en x86-32. No es necesario que los datos estén alineados en memoria algo que si es necesario en ARM-32 y MIPS-32.

Otra de las decisiones que tomaron en RISC-V, es no implementar el *branch retardado* que había disponible en MIPS-32 y Oracle SPARC, esto es debido a que en los *branches condicionales,* inicialmente se tardaba un ciclo en determinar si se tomaba el salto y se ejecutaba la instrucción indicada, o se ejecutaba la siguiente instrucción, en el caso de procesadores posteriores a MIPS-32, los cuales contaban con pipeline con más de 5 etapas, esta decisión tardana más de un ciclo de reloj en tomarse, por lo que la mejora que se aplicaba al utilizar el *branche retardado*, que consistía en ejecutar una instrucción útil justo después de un salto condicional y evitar ese retraso de un ciclo, ya no era efectiva. Tampoco se usan los códigos de condición de ARM-32 y x86-32 para *branches condicionales*, ya que complican el cálculo de dependencias en ejecución fuera de orden y no se han implementado las instrucciones de *loop* del x86-32. Tampoco se han implementado las llamadas a funciones complejas en RV32I, como las instrucciones *enter* y *leave* de x86-32.

3. SOPORTE HARDWARE RISC-V

En este capítulo mostraremos las diferentes placas basadas en RISC-V que nos podemos encontrar en el mercado actualmente. Realizaremos una breve descripción de cada una de ellas y hablaremos de las características técnicas que nos podemos encontrar, en función del tipo de placa y el microcontrolador o procesador que tenga implementado.

También en el último apartado, compararemos algunas de las placas más conocidas con sus competidoras más directas basadas en RISC-V.

3.1. SIFIVE

SiFive es la primera empresa encargada en desarrollar hardware basado en RISC-V, fue fundada por los creadores de la arquitectura RISC-V para proporcionar chips basados en RISC-V. Andrew Waterman, uno de los arquitectos de RISC-V, es el Jefe de Ingeniería y cofundador de SiFive, la empresa situada cerca de las instalaciones de la Universidad de Berkeley (como mostramos en el apartado 2) pertenece a la Fundación RISC-V.

En SiFive no solo podemos especificar nuestro propio procesador (SiFive, Core Designer, s.f.), sino que también ha sacado al mercado una serie de placas basadas integramente en la arquitectura RISC-V.

A continuación mostraremos las que se encuentran disponibles en la actualidad, así como sus principales caracterísiticas, la información detallada a continuación ha sido extraida de (SiFive, s.f.).

3.1.1. Modelo HiFive1

Se trata de una de las primeras placas disponibles en SiFive (actualmente no está disponible, sustituida por la nueva versión HiFive1 Rev B) y compatible con Arduino. Esta es la placa que utilizaremos posteriormente para realizar nuestro caso de uso.

Características:

- **Microcontrolador FE310-G000:** Dispone de un core SiFive E31 RISC-V de alto rendimiento y pipeline de una instrucción, ejecución en orden y una ejecución de 1 instrucción por ciclo. Soporta solo

modo máquina, multiplicación estándar, operaciones atómicas e instrucciones comprimidas de RISC-V (ISA RV32IMAC).

El sistema de memoria tiene DTIM con un tamaño de 16 KiB. Dispone de una cache de instrucciones de 2 vías y un tamaño de 16 KiB.

- Operating Voltage 3.3 V and 1.8 V
- Input Voltage 5 V USB or 7-12 VDC Jack
- IO Voltages Both 3.3 V or 5 V supported
- Digital I/O Pins 19
- PWM Pins 9
- SPI Controllers/HW CS Pins 1/3
- External Interrupt Pins 19
- External Wakeup Pins 1
- Flash Memory 128 Mbit Off-Chip (ISSI SPI Flash)
- Host Interface (microUSB) Program, Debug, and Serial Communication
- Peso 22 g

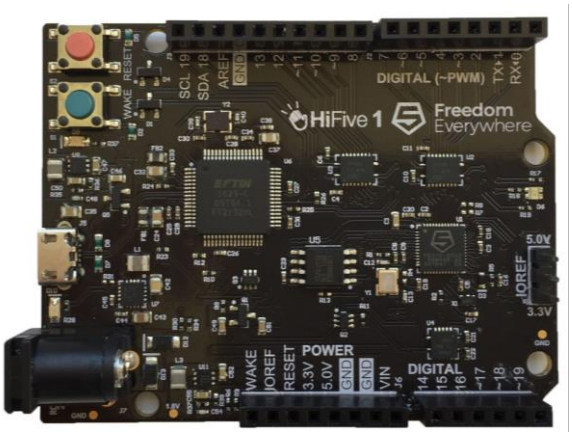

Figura 3.1: HiFive1 (SiFive, HiFive1, s.f.)

3.1.2. HiFive1 Rev B

Se trata de una revisión del modelo HiFive1 que incluye algunas mejoras como Wifi (Precio actual 59$ + 12$ de gastos de envío). A continuación, se detallan las características del nuevo modelo.

Características:

- **Microcontrolador FE310-G002:** Dispone de un core SiFive E31 RISC-V de alto rendimiento y pipeline de una instrucción, ejecución

en orden y una ejecución de 1 instrucción por ciclo. Soporta modo máquina y modo privilegiado, multiplicación estándar, operaciones atómicas e instrucciones comprimidas de RISC-V (ISA RV32IMAC).

El sistema de memoria tiene DTIM con un tamaño de 16 KiB. Dispone de una cache de instrucciones de 2 vías y un tamaño de 16 KiB.

- Operating Voltage 3.3 V and 1.8 V
- Input Voltage 5 V USB or 7-12 VDC Jack
- IO Voltage 3.3 V
- Digital I/O Pins 19
- PWM Pins 9
- SPI Controllers/HW CS Pins 1/3
- UART 2
- I2C 1
- Networking WiFi/BT (off-chip)
- External Interrupt Pins 19
- External Wakeup Pins 1
- Flash Memory 32 Mbit Off-Chip (ISSI SPI Flash)
- Host Interface (microUSB) Program, Debug, and Serial Communication
- Debug Segger J-Link, drag/drop code download
- Peso 22 g

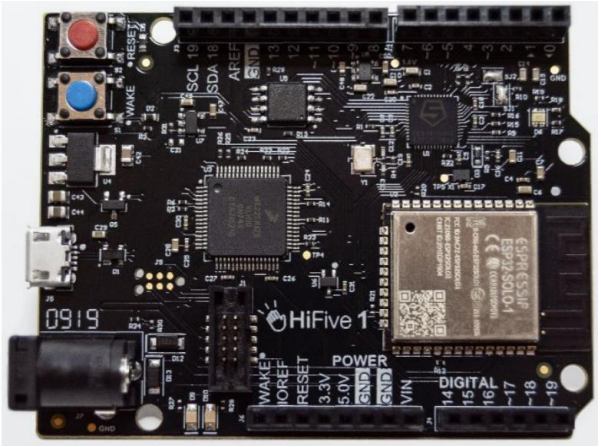

Figura 3.2: HiFive1 Rev B (SiFive, HiFive1 Rev B, s.f.)

3.1.3. HiFive Unleashed

Se trata de la última placa basada en RISC-V desarrollada por SiFive (precio actual 999$ + 40$ de gastos de envío). Contiene el procesador Freedom U540, el primer procesador RISC-V multicore y compatible con Linux (ya que este procesador soporta modo supervisor, lo que nos permite poder instalar un SO).

Características:

- **SoC SiFive Freedom U540 SoC:** Tienen 8 cores, 4 cores U54 RISC-V que soportan modo máquina, supervisor y usuario. Tienen L1 de cache de instrucciones de 32 KiB y 8 vías y L1 de cache de datos de 32 KiB y 8 vías. (ISA RV54IMAFDC)
  Los otros 4 cores son los E51 RISC-V con modo máquina y usuario, L1 I-cache de 16 KiB y 2 vías y 8 KiB DTIM. (ISA RV64IMAC).
  Tiene una cache L2 de 2 MiB con 16 vías.
- Memory 8GB DDR4 with ECC
- Gigabit Ethernet Port
- Flash Memory 32MB Quad SPI Flash from ISSI
- Removable Storage MicroSD Card
- Future Expansion FMC Connector

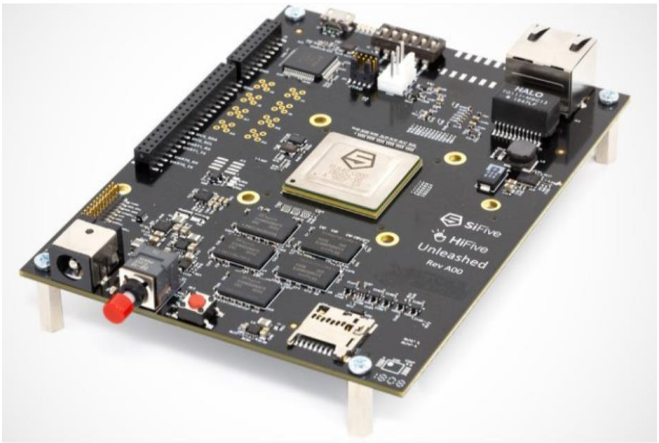

Figura 3.3: HiFive Unleashed (SiFive, HiFive Unleashed, s.f.) (Crowd supply, s.f.)

3.2. GROVE AI HAT PARA EDGE COMPUTING

Grove AI HAT para Edge Computing se basa en el módulo para AI Sipeed MAix M1 que utiliza el procesador Kendryte K210. Se trata de una extensión de bajo costo, pero potente para la Raspberry Pi 3 B+. También se pude utilizar de forma independiente para aplicaciones de *edge computing*. Información extraída de (Seeed, Grove AI HAT for Edge Computing, s.f.).

Características:

- Procesador: Sipeed MAIX-I module w/o WiFi (1st RISC-V 64 AI Module, K210)
- 1x USB 2.0, Tipo C
- 6x Grove Interface: 1x Digital IO, 1x PWM, 1x I2C, 1x UART, 2x ADC
- 1x Power LED, 1x Boot LED
- 1x Reset Button, 1x Boot Button
- 1x LCD Interface
- 1x Camera Interface
- 1x Digital Mic
- 1x Acelerómetro
- 1x JTAG & ISP UART Pin Header
- 2x 20 Pin Header with I2C, UART, SPI, I2S, PWM, GPIO

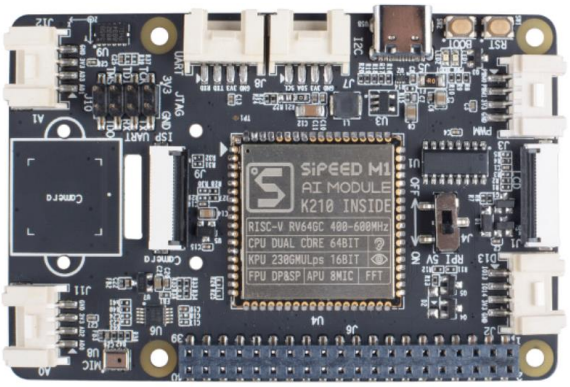

Figura 3.4: Grove AI Hat para Edge Computing
(Seeed, Grove AI HAT for Edge Computing, s.f.)

Esta placa ha sido utilizada para poder desarrollar un sistema que nos permite realizar conteo y reconocimiento facial. Para ello, además de esta placa que utiliza el procesador Kendryte k210, se necesita una Raspberry Pi 3 Model B+ una cámara y una pantalla LCD TFT. Toda la información de los dispositivos necesarios y los pasos para poder realizar este sistema de reconocimiento facial se encuentran disponibles en (Seeed, 2019).

3.3. COMPARACIÓN CON OTROS SOC

En este apartado mostraremos una comparación entre los SoC más conocidos de bajo coste y los últimos SoC basados en RISC-V, teniendo en cuenta que sean similares entre sí en cuanto a características y funcionalidades.

En la Tabla 3.1 podemos ver una comparación entre las características principales de HiFive1 y Arduino, al igual que podemos ver en la Tabla 3.2 una comparación entre HiFive Unleased y Raspberry Pi Model B.

Estos datos nos permitirán determinar si el uso de la arquitectura RISC-V en la docencia, es algo posible e incluso recomendable, teniendo en cuenta que ya hay en la actualidad diversos laboratorios y asignaturas en universidades españolas, que utilizan como herramientas para el aprendizaje tanto Arduino como Raspberry Pi.

	Arduino MEGA 2560 REV3	HiFive1
Memoria	• Flash Memory 256 KB of which 8 KB used by bootloader. • SRAM 8 KB • EEPROM 4 KB	• 16KB L1 Instruction Cache • 16KB Data SRAM Scratchpad • Flash Memory 128 Mbit Off-Chip (ISSI SPI Flash)
Microcon-trolador	• ATmega2560 (Arquitectura AVR)	• FE310-G000 (SiFive, SiFive FE310-G000 Preliminary Datasheet v1p5, 2016-2017)
Entrada/ Salida	• Digital I/O Pins 54 (15 provide PWM output) • Analog Input Pins 16	• Digital I/O Pins 19 • PWM Pins 9 • SPI Controllers/HW CS Pins 1/3
Voltaje	• Operating Voltage 5V • Input Voltage (recommended) 7-12V • Input Voltage (limite) 6-20V	• Operating Voltage 3.3 V and 1.8 V • Input Voltage 5 V USB or 7-12 VDC Jack • IO Voltages Both 3.3 V or 5 V supported
Frecuencia	• 16 MHz	• 320 MHz
Tamaño	• 101.52 mm x 53.3 mm	• 68 mm x 51 mm
Peso	• 37 g	• 22 g
Precio	• 35 € + 9 € de gastos de envío	• 59$ + 12$ de gastos de envío

Tabla 3.1: Comparación entre HiFive1 (SiFive, HiFive1, s.f.) y Arduino MEGA (Arduino, 2019)

	Raspberry Pi Model B+	HiFive Unleashed
Procesador	• Broadcom BCM2837B0 • Cortex-A53 64-bit SoC • 1.4GHz	• SiFive Freedom U540 SoC • 64-bit • 1,5GHz • 4+1 Multi-Core Coherent Configuration
Memoria	• 1GB LPDDR2 SDRAM	• 8GB DDR4 with ECC • 32MB Quad SPI Flash from ISSI
Conectividad	• 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac Wireless LAN, Bluetooth 4.2, BLE • Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) • 4 × USB 2.0 ports	• Gigabit Ethernet Port
Video & sonido	• 1 × full size HDMI • MIPI DSI display port • MIPI CSI camera port • 4 pole stereo output and composite video port	• (Required HiFive Unleashed Expansion Board)
Multimedia	• H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics	-
Tarjeta SD	• Micro SD port for loading your operating system and storing data	• MicroSD Card
Acceso	• Extended 40-pin GPIO header	• 16 General Purpose I/O pins
Potencia de entrada	• 5V/2.5ª DC via micro USB connector • 5V DC via GPIO header • Power over Ethernet (PoE)–enabled (requires separate PoE HAT)	• External 12V, 2ª power supply. • When not powering directly from the FMC expansion module, a 12V DC Wall Adapter with a 5.5 outer diameter, 2.5 mm center-positive barrel plug, must be plugged into DC power jack on the HiFive Unleashed.
Tamaño	• 85 mm x 56 mm	• 120 mm x 90 mm
Precio	• 32 €	• 999 $ (895€)

Tabla 3.2: Comparación entre Raspberry Pi model B (Foundation, Raspberry Pi, 2019) y HiFive Unleased (SiFive, HiFive Unleashed, s.f.)

4. SOPORTE SOFTWARE RISC-V

Disponer de paquetes para desarrolladores, así como de buenas herramientas para poder crear y depurar nuestros códigos, es algo fundamental para poder usar correctamente las nuevas arquitecturas que nos encontramos en el mercado. Es por ello, por lo que en el siguiente capítulo realizaremos un estudio del software disponibles para RISC-V, principalmente nos centraremos en los paquetes de desarrollo y los sistemas operativos más conocidos (para una información más detallada (RISC-V, Software Status, 2019)).

4.1. SOFTWARE PARA DESARROLLADORES

A continuación, hablaremos un poco de los paquetes para desarrolladores disponibles actualmente para RISC-V.

4.1.1. Paquete de desarrollo *Freedom E SDK Toolchain*.

Se trata de una de las herramientas imprescindibles para poder realizar nuestros programas basados en RISC-V.

Se puede encontrar disponible en (SiFive, Open Source Software for Developing on the Freedom E Platform, 2019) y nos permite depurar y compilar nuestro código para poder generar nuestros ejecutables y subirlos a la placa, en la sección *5.2.2.* indicaremos paso a paso como instalar y usar esta herramienta.

4.1.2. PlatformIO

Se trata de una extensión gratuita, disponible para Visual Studio Code, utiliza como base el SDK comentado anteriormente y nos permite también depurar y compilar nuestro código para distintas placas disponibles de SiFive, al igual que subirlo a la placa de forma rápida y sencilla. En el apartado *5.2.3.* se habla con más detalle de como instalarlo y trabajar con él.

4.2.3. GNU MCU Eclipse

Como se puede ver en el siguiente enlace (SiFive, Boards, 2019), hay disponible un plug-in en Eclipse basado en GNU para el desarrollo de sistemas embebidos y desarrollado para diferentes plataformas. RISC-V está disponible entre sus plataformas, y permite crear proyectos en C/C++.

4.2. SISTEMAS OPERATIVOS Y KERNELS

No solo es importante poder desarrollar nuestras aplicaciones para crear nuestros propios sistemas, sino que una buena arquitectura también se caracteriza por dar soporte a los sistemas operativos más utilizados y mejorar su rendimiento.

En la Tabla 4.1. se muestran algunos de los sistemas operativos más conocidos basados en Linux que hay disponibles para RISC-V (en el apartado 5.3 hablamos más en detalle del SO Fedora) y en la Tabla 4.2. los SO para tiempo real. La información que se presenta a continuación ha sido extraída de (RISC-V, OS and OS kernels, 2019).

Nombre	Encargados de su mantenimiento
Fedora (Fedora/RISC-V, 2018)	Richard WM Jones, Stefan O'Rear, David Abdurachmanov
Debian (Fernandez Montecelo & Merker, 2016) (Fernandez Montecelo & Keville, Debian for RISC-V 64-bit (riscv64), s.f.) (Fernandez Montecelo, Debian, 2017)	Manuel A. Fernandez Montecelo
OpenMandriva (OpenMandriva, 2016-2019)	Bernhard "Bero" Rosenkränzer
openSUSE (openSUSE, 2019)	Andreas Schwab (SUSE)

Tabla 4.1: Distribuciones de Linux

Nombre	Encargados de su mantenimiento
RTEMS (RTEMS, 2019) (RTEMS RISC-V, 2019)	Hesham Almatary
FreeRTOS (FreeRTOS Real Time Kernel (RTOS), 2019) (Quality RTOS & Embedded Software, s.f.)	AWS
Zephyr (Zephyr, 2019) (Zephyr Project, 2015-2019)	Karol Gugala (Antmicro), Peter Gielda (Antmicro), Nathaniel Graff (SiFive)
Apache Mynewt (Runtime, 2016)	James Pace, Runtime
OpenWrt (OpenWrt, 2018) (OpenWrt, 2019)	Zoltan Herpai
seL4 (sel4riscv-manifest, 2019) (Hesham M. , 2017)	Hesham Almatary and Data61/CSIRO

Tabla 4.2: SO de Tiempo Real

5. CASO DE USO: DESARROLLO EN HIFIVE 1

En este capítulo realizaremos un caso de uso en el que crearemos un driver sencillo para poder conectar un dispositivo con varios leds a HiFive1, esto nos permitirá comprobar el funcionamiento de la placa, probar el código, así como los paquetes para desarrolladores disponibles, su instalación y funcionamiento. Además de esto también ejecutaremos en HiFive1 algunos de los bechmarks más utilizados en microcontroladores y analizaremos sus resultados.

5.1. BENCHMARK

A continuación, ejecutaremos los benchmark Dhrystone y Coremark sobre HiFive1.

5.1.1. Dhrystone

Se trata de un benchmark gratuito utilizado para medir el rendimiento de los microcontroladores y CPUs utilizados en los sistemas embebidos. Mide la capacidad del procesador para trabajar con enteros y hay que tener en cuenta, que en función del compilador que se utilice, así como de las opciones utilizadas a la hora de compilar, pueden variar los resultados, ya que este benchmark está fuertemente influenciado por el compilador. Las medidas obtenidas nos indican cuantas veces se pudo ejecutar el programa en 1 segundo. Como podemos ver en los resultados obtenidos en HiFive1, se ha podido ejecutar 735.294 veces en 1 segundo.

Este benchmark ya se encontraba disponible en el propio paquete Freedom E SDK para ejecutar sobre HiFive1, solo es necesario compilarle y ejecutarle sobre la placa para obtener los resultados.

```
RESULTADOS OBTENIDOS: Dhrystone Benchmark, Version 2.1
272655974 Hz (core freq)
Microseconds for one run through Dhrystone: 1.3
Dhrystones per Second: 735294.1
```

Ahora una vez que ya tenemos calculados los Dhrystones por segundo, tenemos que calcular los "*Dhrystone VAX MIPs*" esto es debido a que necesitamos comparar el rendimiento de la placa que estamos midiendo (en nuestro caso la

placa HiFive1) con el rendimiento obtenido en la máquina de referencia. La máquina de referencia elegida fue VAX 11/780.

Para obtener los VAX MIPs, hay que dividir el número de Dhrystone por segundo obtenidos en la placa a medir, entre el número de Dhrystone por segundo alcanzados por la máquina de referencia. El VAX 11/780 alcanza 1757 Dhrystones por segundo. Información extraída de (arm, 2010)

Si dividimos los resultados obtenidos entre 1757 obtenemos por lo tanto los VAX MIPs (Dhrystone MIPS), y en nuestro caso de estudio tenemos 418,494 DMIPS (1,54 DMIPS/MHz).

Si comparamos con los datos obtenidos (0.45 DMIPS/MHz datos obtenidos de las siguientes web (Vak, 2017)) por el microcontrolador Atmega2560 que se encuentra instalado en Arduino MEGA, vemos que le triplica en velocidad, aunque hay que tener en cuenta que no son sistemas exactamente iguales y que estos valores dependen también del compilador que se utilice, pero teniendo en cuenta las características de ambos y sus precios, tenemos un rendimiento muy aceptable de la placa HiFive1 frente a sus posibles competidores.

5.1.2. Coremark

Coremark (EEMBC, s.f.) se trata de un benchmark bastante conocido para ver el rendimiento de los microcontroladores y CPUs que se utilizan en los sistemas embebidos, a diferencia del Dhrystone, Coremark se centra más en medir la capacidad de la MCU y CPU en vez de tener en cuenta la capacidad del compilador para optimizar la carga de trabajo, como sí ocurre con Dhrystone.

```
RESULTADOS OBTENIDOS:
core freq at 272787046 Hz
2K performance run parameters for coremark.CoreMark Size: 666
Total ticks      : 470695
Total time (secs): 14
Iterations/Sec   : 714
Iterations       : 10000
Compiler version : GCC8.1.0
```

```
Compiler flags   : -O2 -fno-common -funroll-loops -finline-
functions–param max-inline-insns-auto=20 -falign-functions=4
-falign-jumps=4 -falign-loops=4
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Divide the reported Iterations/Sec by the reported core
frequency in MHz to obtain a CoreMarks/MHz value.
714/272,787046 = 2,617426342 CoreMarks/MHz
```

Si vemos los resultados obtenidos por FE310-G000 y los comparamos con el resultado obtenido por Atmega2560 (Tabla 5.1.) disponible en la web de Coremarks (Coremark Scores, 2009-2019), podemos ver que seguimos teniendo un buen rendimiento por parte de HiFive, aunque hay que tener en cuenta la gran diferencia que tenemos de frecuencia del reloj, si esta fuese similar tendríamos unos resultados bastante ajustados teniendo en cuenta los resultados actuales, pero aun así con estos datos podemos ver que el rendimiento de FE310-G000 es un rendimiento acorde a los sistemas actuales.

Processor	Cert.	Compiler	Execution Memory	MHz	Cores	CoreMark	CoreMark / MHz	Threads	Date↓
Atmel ATmega2560		avr-gcc 4.3.2	Internal flash & RA…	8	1	4.25	0.53	1	2010-07-12

Tabla 5.1: Puntuación Coremarks de ATmega2560

5.2. DESARROLLO DE CÓDIGO

5.2.1. Conectarse a HiFive1

Antes de compilar un programa de prueba deberemos conectar nuestra placa y comprobar que todo funciona correctamente. Para ello podemos usar GNU screen, realizando los pasos que se muestran a continuación.

Instalamos screen en Linux:

```
sudo apt-get install screen
```

Para permitir la transferencia de información sin usar permisos en Ubuntu, seguir los pasos indicados en el capítulo 5 de la guía de inicio SiFive HiFive1 (SiFive, SiFive HiFive1 Getting Started Guide, 2017).

Con el siguiente comando podremos conectarnos a la placa que tengamos enlazada a nuestro PC:

```
sudo screen /dev/ttyUSB1 115200
```

Después de conectarnos por primera vez nos debería de aparecer el logotipo de SiFive (Figura 5.1, este logotipo esta generado por un programa cargado por defecto en nuestra placa HiFive1 y que posteriormente sustituiremos con un programa de nuestra elección), si no es así presionar el botón reset de la HiFive1, ya que en algunas ocasiones es necesario reiniciar el programa que tenemos cargado en la placa.

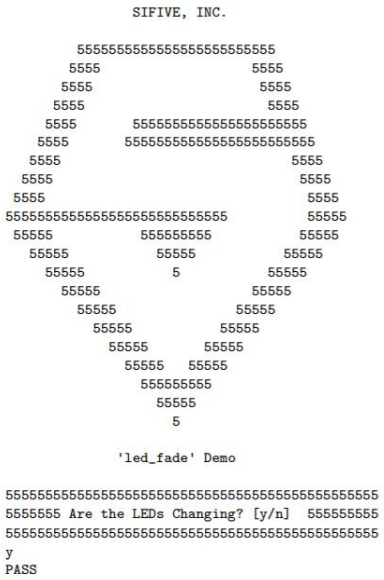

Figura 5.1: Programa cargado por defecto en HiFive1 (SiFive, SiFive HiFive1 Getting Started Guide, 2017)

5.2.2. SDK

En SiFive cuando nos descargamos su guía para comenzar a trabajar con HiFive1 (SiFive, SiFive HiFive1 Getting Started Guide, 2017), nos indica que tiene disponible el paquete de desarrollo Freedom E SDK Toolchain, el cual se pude descargar accediendo al siguiente enlace de github (SiFive, Open Source Software for Developing on the Freedom E Platform, 2019), este paquete de SDK nos permite compilar, depurar y posteriormente subir nuestro código, a

alguna de las placas basadas en RISC-V que vimos en el apartado 3.1. Toda la información de este apartado es un resumen extraído de la guía de (SiFive, SiFive HiFive1 Getting Started Guide, 2017).

Instalación de SDK en Ubuntu

Para instalar SDK desde ubuntu primero debemos instalar RISC-V GNU Toolchain y OpenOCD disponible en el siguiente enlace (SiFive, Boards, 2019), nos descargamos el paquete necesario para nuestra plataforma, desempaquetamos en el destino que deseemos guarda dichos archivos y después tendremos que tener en cuenta que cada vez que queramos utilizar GNU y openOCD hay que indicar los siguientes PATH:

```
export RISCV_OPENOCD_PATH=/carpeta/destino/openocd
export RISCV_PATH=/carpeta/destino/riscv64-unknown-elf-gcc-<date>-<version>
```

Una vez que ya tenemos instalados los dos paquetes anteriores ya podemos instalar Freedom E SDK, toda la información que aparece aquí está de forma más detallada en el fichero README de (SiFive, Open Source Software for Developing on the Freedom E Platform, 2019).

Después de descargarnos Freedom E SDK, realizamos los pasos que mostramos a continuación para terminar la instalación y a continuación ya podremos realizar alguna prueba para ver el funcionamiento:

```
cd freedom-e-sdk
make tools
```

Compilar y subir un programa a HiFive1

Una vez que ya tenemos todo instalado y funcionando, podemos compilar y subir un programa de prueba, para ello podemos utilizar uno de los disponibles en el paquete SDK.

Para compilar el programa demo_gpio:

```
cd freedom-e-sdk
make software PROGRAM=demo_gpio BOARD=freedom-e300-hifive1
```

Una vez compilado ya podemos subirle a nuestra placa y después presionar reset para ver la ejecución del nuevo programa.

```
make upload PROGRAM=demo_gpio BOARD=freedom-e300-hifive1
```

Depurar

Para depurar con GDB conectamos la placa y realizamos los siguientes pasos:

Abrimos dos terminales y nos situamos en el directorio freedom-e-sdk:

```
cd freedom-e-sdk
```

Primero ejecutamos openocd en un terminal:

```
make run_openocd BOARD=freedom-e300-hifive1
```

En la otra teminal ejecutamos gdb (en el manual pone depurar con run_debug pero no funciona correctamente, hay que poner en su lugar run_gdb)

```
make run_gdb PROGRAM=<tu programa> BOARD=freedom-e300-hifive1
```

5.2.3. PlatformIO (Visual Studio Code)

En este apartado mostraremos otra aplicación disponible de código abierto, para compilar, depurar y subir nuestro código creado para RISC-V. Toda la información de este apartado se trata de un resumen del tutorial que se puede ver en la siguiente dirección (Fink, s.f.), publicado por Western Digital Corporation.

Instalación

Lo primero que tenemos que hacer es instalar visual studio code (Visual Studio Code, 2019). Después podremos instalarnos en Visual Studio la extensión de PlatformIO, que dispone de las herramientas necesarias para trabajar con la placa HiFive.

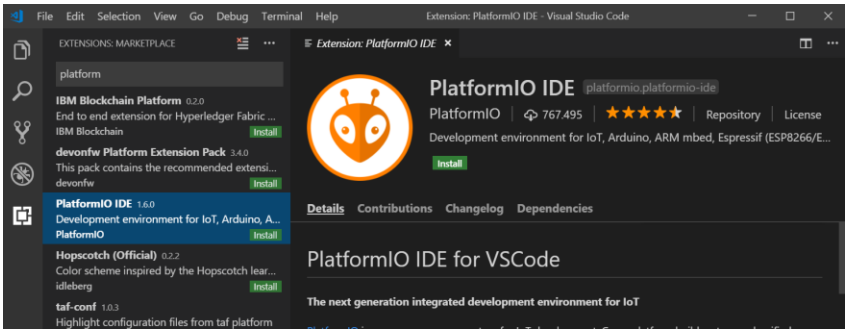

Figura 5.2: Extensión PlatformIO IDE.

Como se puede ver en la información disponible en su página oficial (PlarformIO, 2014-2019), PlatformIO se trata de un ecosistema de código abierto. Un entorno de desarrollo integrado multiplataforma y con un depurador unificado. Dentro de las plataformas que soporta se encuentra RISC-V.

Compilar y subir un programa a HiFive1

Una vez instalado PlarformIO ya podemos empezar a desarrollar nuestras aplicaciones, esta extensión disponible de forma gratuita para *Visual Studio Code,* nos permite compilar, ensamblar y enlazar nuestros programas de forma transparente, sin tener que modificar o crear un *makefile* específico para nuestro código, como sí ocurre si utilizásemos únicamente Freedom E SDK (hay que tener en cuenta que PlatformIO lo que hace es utilizar Freedom E SDK como base). Una vez que ya tenemos nuestro programa listo para subir a la placa, con PlatformIO podemos subirlo clicando únicamente un botón.

A continuación, realizaremos un pequeño programa (basado en el programa mostrado en (Fink, s.f.)) modificado para poder utilizar y conectar la placa de leds BerryClip a nuestra placa HiFive. Este programa utilizará código c y ensamblador, y nos permitirá ir mostrando los pasos necesarios para compilar, ensamblar y enlazar el código y posteriormente subirlo a HiFive1.

Lo primero que tendremos que hacer, será crearnos un nuevo proyecto asignándole la placa que utilizaremos (en este caso HiFive1) y el framework Freedom E SDK, como se muestra en la Figura 5.3.

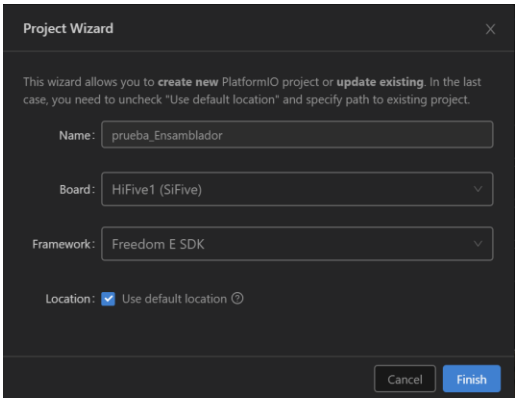

Figura 5.3: Creación de un nuevo proyecto.

Ahora, ya tenemos la base para crear nuestro proyecto, en el menú izquierdo podemos ver la plataforma, la carpeta para las librerías, y la carpeta src para introducir nuestro código.

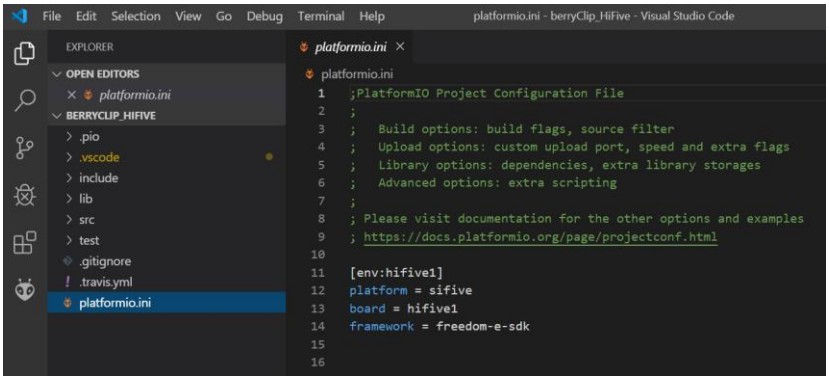

Figura 5.4: Nuevo proyecto.

A continuación, crearemos los ficheros ***berryClip.c*** y ***lib_berryClip.h*** donde programaremos nuestras librerías y nuestro main.

Para compilar, ensamblar y enlazar nuestro código y ver si no hay ningún error, clicamos en el icono del check que se encuentra en la parte inferior izquierda de Visual Studio Code.

Figura 5.5: Compilar el proyecto.

Ahora empezaremos a crear código en ensamblador. Para ello nos creamos los siguientes ficheros; **address_offset.inc** (donde guardaremos las direcciones

que usaremos en nuestros ficheros .S y los offset necesarios) **setLED.S y setupGPIO.S** (importante poner la extensión .S en mayúsculas para evitar errores). En este caso trabajaremos con la GPIO, por lo que necesitamos saber cuál es la dirección base de la GPIO, esta dirección la podemos encontrar indicada en el manual (SiFive, SiFive FE310-G000 Manual v2p3, 2016-2017) en el Capítulo 3 – Memory map (en la Tabla 5.2 se encuentra remarcada la dirección a utilizar) y en el Capítulo 17 tenemos listados todos los offset necesarios, para poder acceder a los registros que nos permitirán trabajar con la GPIO (Tabla 5.3).

FE310-G000 Memory Map

Base	Top	Attr.	Description	Notes
0x0000_0000	0x0000_00FF		*Reserved*	Debug Address Space
0x0000_0100	0x0000_0FFF	RWXC	Debug	
0x0000_1000	0x1000_1FFF	RXC	Mask ROM	On-Chip Non-Volatile Memory
0x0000_2000	0x0001_FFFF		*Reserved*	
0x0002_0000	0x0002_1FFF	RXC	OTP 8KiB	
0x0002_2000	0x01FF_FFFF		*Reserved*	
0x0200_0000	0x0200_FFFF	RW	CLINT	On-Chip Peripherals
0x0201_0000	0x0BFF_FFFF		*Reserved*	
0x0C00_0000	0x0FFF_FFFF	RW	PLIC	
0x1000_0000	0x1000_7FFF	RW	Always-On (AON)	
0x1000_8000	0x1000_FFFF	RW	PRCI	
0x1001_0000	0x1001_0FFF	RW	OTP Control	
0x1001_1000	0x1001_1FFF		*Reserved*	
0x1001_2000	0x1001_2FFF	RW	GPIO 0	
0x1001_3000	0x1001_3FFF	RW	UART 0	
0x1001_4000	0x1001_4FFF	RW	QSPI0 Control	
0x1001_5000	0x1001_5FFF	RW	PWM 0	
0x1001_6000	0x1002_2FFF		*Reserved*	
0x1002_3000	0x1002_3FFF	RW	UART 1	
0x1002_4000	0x1002_4FFF	RW	QSPI 1	
0x1002_5000	0x1002_5FFF	RW	PWM 1	
0x1002_6000	0x1003_3FFF		*Reserved*	
0x1003_4000	0x1003_4FFF	RW	QSPI 2	
0x1003_5000	0x1003_5FFF	RW	PWM 2	
0x1003_6000	0x1FFF_FFFF	RW	*Reserved*	
0x2000_0000	0x3FFF_FFFF	RXC	QSPI 0 XIP (512 MiB)	Off-Chip Non-Volatile Memory
0x4000_0000	0x7FFF_FFFF		*Reserved*	
0x8000_0000	0x8000_3FFF	RWXC	Data Tightly Integrated Memory (DTIM) 16 KiB)	On-Chip Volatile Memory
0x8000_4000	0xFFFF_FFFF		*Reserved*	

Table 3.1: FE310-G000 Memory Map.
Memory Attributes: **R** - Read **W** - Write **X** - Execute **C** - Cacheable

Tabla 5.2: Memory Map FE310-G000
(SiFive, SiFive FE310-G000 Manual v2p3, 2016-2017).

GPIO Peripheral Offset Registers		
Offset	**Name**	**Description**
0x000	value	pin value
0x004	input_en	* pin input enable
0x008	output_en	* pin output enable
0x00C	port	output port value
0x010	pue	* internal pull-up enable
0x014	ds	Pin Drive Strength
0x018	rise_ie	rise interrupt enable
0x01C	rise_ip	rise interrupt pending
0x020	fall_ie	fall interrupt enable
0x024	fall_ip	fall interrupt pending
0x028	high_ie	high interrupt enable
0x02C	high_ip	high interrupt pending
0x030	low_ie	low interrupt enable
0x034	low_ip	low interrupt pending
0x038	iof_en	* HW I/O Function enable
0x03C	iof_sel	HW I/O Function select
0x040	out_xor	Output XOR (invert)

Tabla 5.3: Offset registros GPIO FE310-G000

(SiFive, SiFive FE310-G000 Manual v2p3, 2016-2017)

En este tutorial tambien se ha creado un fichero llamado delay.S, que utiliza el timer, cuya información se ecuentra en la guía (SiFive, SiFive FE310-G000 Manual v2p3, 2016-2017) en el Capitulo 11 Core Local Interruptor (CLINT). En la Tabla 5.4 extraida de la guía anterior, podremos ver cual es la dirección de timer, la cual deberemos añadirla al fichero ***memory_map.inc,*** para posteriormente poder configurarle adecuadamente en el fichero ***delay.S***

CLINT Register Map				
Address	**Width**	**Attr.**	**Description**	**Notes**
0x0200_0000	4B	RW	`msip` for hart 0	MSIP Registers
0x0200_0004 ... 0x0200_3FFF			*Reserved*	
0x0200_4000	8B	RW	`mtimecmp` for hart 0	Timer compare register
0x0200_4008 ... 0x0200_BFF7			*Reserved*	
0x0200_BFF8	8B	RO	`mtime`	Timer register
0x0200_C000 ... 0x0200_FFFF			*Reserved*	

Tabla 5.4: SiFive E31 CLINT Memory Map.

Una vez que ya tenemos todos nuestro ficheros creados y programados, ya podemos compilar clicando en la opción mostrada en la Figura 5.5 y subir a la placa, clicando en la opción mostrada en la Figura 5.6.

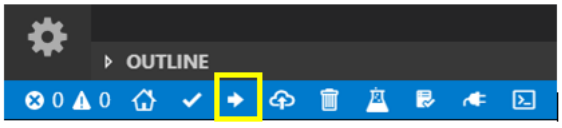

Figura 5.6: Subir el programa a la placa.

A continuación, se muestran todos los ficheros utilizados para realizar la prueba, con su código correspondiente y comentarios indicando lo que se realiza en cada momento. El programa desarrollado se encarga de encender los 6 leds que hay disponibles en la placa BerryClip, y realizar distintas esperas entre el encendido y apagado de los leds, así como probar distintas combinaciones al encender los diferentes leds.

```
//berryClip.c

#include <stdio.h>
#include "lib_berryClip.h"

int main()
{
    int fin = 0;
    int ledNum = 0;
    int led = 0;
    int colors[NUM_LEDS]={GPIO_00_LED1, GPIO_01_LED2, GPIO_02_LED3, GPIO_03_LED4,
                          GPIO_04_LED5, GPIO_05_LED6};

    setupGPIO();
    puts("*----------------------------HOLA----------------------------*\n");
    puts("*--ESTO ES UNA PRUEBA DE CONEXIÓN ENTRE BERRYCLIP Y HIFIVE 1--*\n");

    while(fin<=4){
        setLED(colors[ledNum],ON);
        delay(DELAY_a);
        setLED(colors[ledNum],OFF);
        delay(DELAY_a);
        ledNum++;
        if(ledNum >= NUM_LEDS){
            ledNum = 0;
            fin++;
        }
    }
    while(fin<=10){
        while(led<=5){
            setLED(colors[led],ON);
            led++;
        }
        delay(DELAY_b);
        led--;
        while(led>=0){
            setLED(colors[led],OFF);
            led--;
        }
        delay(DELAY_b);
        led = 0;
        fin++;
    }
    puts("*------------------------------------------------------------*\n");
    puts("*------------------------------------------------------------*\n");
    puts("*----------------------PRUEBA FINALIZADA---------------------*\n");
}
```

```
//lib_berryClip.h

#define DELAY_a 100
#define DELAY_b 300
#define ON 0x01
#define OFF 0x00
#define NUM_LEDS 0x06
#define GPIO_00_LED1 0x000001
#define GPIO_01_LED2 0x000002
#define GPIO_02_LED3 0x000004
#define GPIO_03_LED4 0x000008
#define GPIO_04_LED5 0x000010
#define GPIO_05_LED6 0x000020

void setupGPIO();
int watchBUTTON();
int setLED(int color, int state);
void delay(int milliseconds);
```

```
//address_offset.inc

# Dirección base MTIME y frecuencia
.equ MTIME, 0x0200BFF8
.equ MTIME_FREQUENCY, 33

# Dirección base GPIO
.equ GPIO_CTRL_ADDR, 0x10012000

# Offset GPIO OUTPUT
.equ GPIO_OUTPUT_EN,    0x008
.equ GPIO_OUTPUT_VAL,   0x00C
.equ GPIO_OUTPUT_XOR,   0x040
.equ GPIO_LEDS_PINS,    0x00003F
.equ GPIO_00_LED1,      0x000001
.equ GPIO_01_LED2,      0x000002
.equ GPIO_02_LED3,      0x000004
.equ GPIO_03_LED4,      0x000008
.equ GPIO_04_LED5,      0x000010
.equ GPIO_05_LED6,      0x000020
```

Las constantes GPIO_0<N>_LED<N> guardadas en el fichero *lib_berryClip.h*, estan definiendo el bit asociado con la GPIO que queremos trabajar. Por ejemplo en el caso del LED6 de nuestra placa berryClip, el cual queremos asociar con la GPIO5 de la placa HiFive, sabemos que esta indicado por el bit 5 menos significativo del registro **GPIO_OUTPUT_VAL(GPIO_CTRL_ADDR)**, ya que en este caso estamos estableciendo los pines del 0 al 5 como valores de salida.

```
#setupGPIO.S

.section .text
.align 2
.globl setupGPIO
.include "address_offset.inc"

setupGPIO:

  li t0, GPIO_CTRL_ADDR       # Cargamos en t0 la dirección base de la GPIO
  li t1, GPIO_LEDS_PINS       # Obtenemos el offeset de los leds a activar
  sw t1, GPIO_OUTPUT_EN(t0)   # Activamos de salida los pines de los LEDS para que sean
                                de escritura
  sw t1, GPIO_OUTPUT_XOR(t0)  # Configuramos para que los pines de los LEDS estén activos
  and t1, t1, 0xffffffff      # Ponemos a 1 los bits de la GPIO para apagar los leds
  sw t1, GPIO_OUTPUT_VAL(t0)  # Escribimos el valor anterior para limpiar los leds

  ret
```

```
#setLED.S

.section .text
.align 2
.globl setLED
.include "address_offset.inc"

.equ NOERROR,   0x0
.equ ERROR,     0x1

# En a0 tenemos el offset del led y en a1 si encedemon o apagamos el led

setLED:
    li t0, GPIO_CTRL_ADDR       # Cargamos en t0 la dirección de la GPIO
    lw t1, GPIO_OUTPUT_VAL(t0)  # Guardamos en t1 los valores actuales de
                                  los pins

    beqz a1, ledOff             # Si a1 == 0 saltamos a ledOn
    li t2, 1                    # Cargamos un 1 en t2
    beq a1, t2, ledOn           # Si a1 == 1 salta a ledOff
     li a0, ERROR                # Algo ha salido mal, retorna error
    j exit

ledOff:
    xor t1, t1, a0              # Hace xor para cambiar los bits
                                  indicados en a0
    sw t1, GPIO_OUTPUT_VAL(t0)  # Escribe el nuevo valor en GPIO output
    li a0, NOERROR
    j exit

ledOn:
    xor a0, a0, 0xffffffff      # Ivertimos los bits, todos a 1 menos los
                                  indicados en a0
    and t1, t1, a0              # And a0 y t1 para poner a 0 los bits de
                                  los leds que queremos encender
    sw t1, GPIO_OUTPUT_VAL(t0)  # Escribimos la nueva salida en el
                                  registro indicado
    li a0, NOERROR

exit:
    ret
```

```
#delay.S

.section .text
.align 2
.globl delay

.include "address_offset.inc"

# En a0 tenemos el tiempo en milisegundos que queremos esperar

delay:

    li t0, MTIME              # Cargamos el registro del timer
    lw t1, 0(t0)              # Cargamos el valor actual del timer
    li t2, MTIME_FREQUENCY    # Obtenemos la frecuencia del reloj
    mul t2, t2, a0            # Multiplicamos los milisegundos por
                                la frecuencia
    add t2, t1, t2            # El valor que debe alcanzar el
                                timer esta en t2
1:
    lw t1, 0(t0)              # leemos el valor de mtime
    blt t1, t2, 1b            # permanecemos en el bucle hasta
                                alcanzar el valor indicado
    ret
```

Depurar con PlatformIO

Mientras se estaba realizando este trabajo, la herramienta para depurar nuestro código estaba disponible únicamente en PlatformIO Plus la cual era una versión de pago, a mediados de este verano la plataforma ha pasado a ser Open Source.

En la Figura 5.7 se puede ver cuáles son las diferentes opciones disponibles para poder depurar nuestro código.

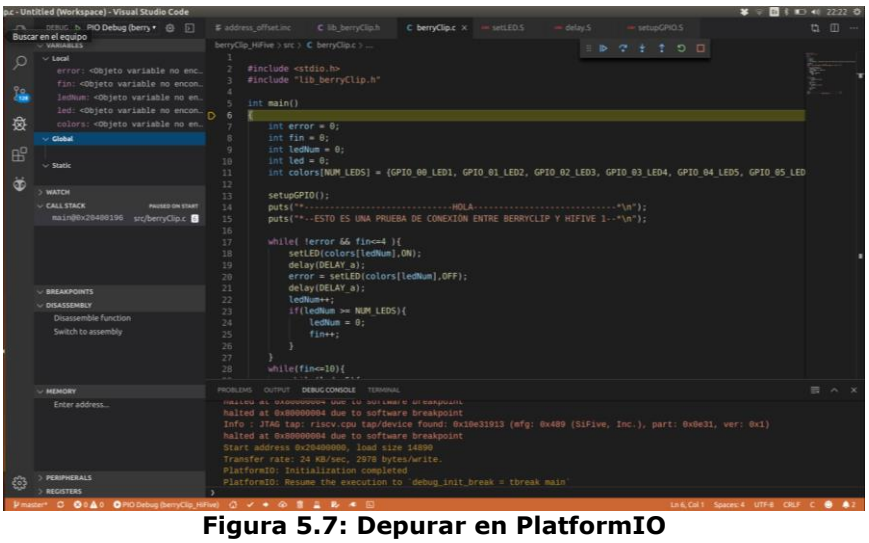

Figura 5.7: Depurar en PlatformIO

En el panel lateral izquierdo nos aparecen las variables disponibles, los puntos de ruptura que tengamos definidos, también podemos ver lo que tenemos en la memoria, indicando la dirección a inspeccionar en el apartado MEMORY. Para poder ir moviéndonos por el código, solo necesitamos el panel que nos aparece en la parte superior del código (Figura 5.8), con el podemos ir pasando de instrucción en instrucción, detener la ejecución del programa y reanudarla.

Figura 5.8: Menú depurardor PlatformIO

5.2.4. Conexión y programación dispositivo I/O

En este apartado hemos realizado una pequeña descripción de la conexión de un dispositivo formado por 6 leds, 1 buzzer y un interruptor, para comprobar el correcto funcionamiento del programa probado en el apartado 5.2.3 y la forma de conectar nuestros dispositivos, así como la información disponible para poder realizar las conexiones de forma adecuada y rápida.

A continuación, utilizaremos la placa BerryClip (Figura 5.11) y la conectaremos a HiFive1. Para poder conectarla será necesario utilizar la documentación disponible en (SiFive, Esquema placa HiFive, 2016) con el esquema de la placa HiFive1, para poder conectar de forma adecuada a los leds que deseamos utilizar en nuestro programa, y que si miramos en el esquema de la placa podemos ver (Figura 5.9) que se encuentran en J3.

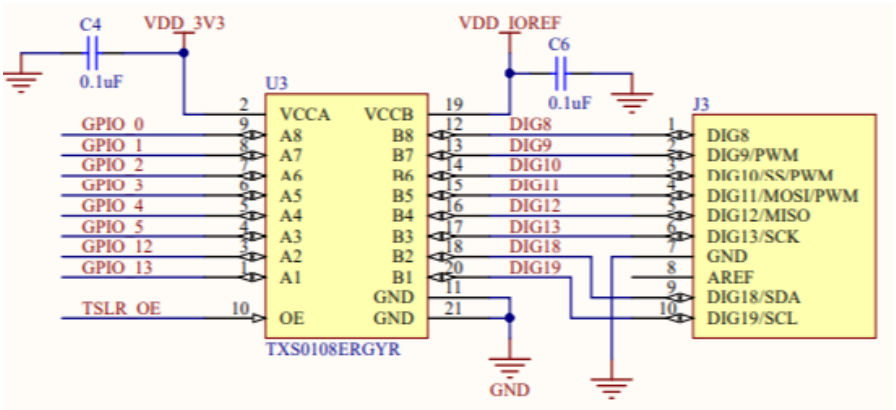

Figura 5.9: GPIO 0-13 (SiFive, Esquema placa HiFive, 2016)

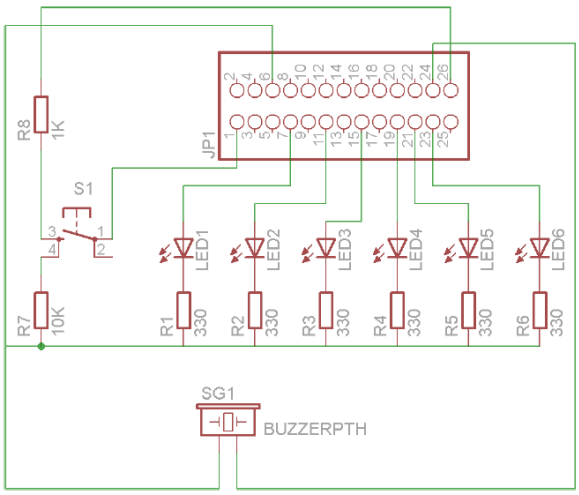

Figura 5.10: Circuito BerryClip (Berryclip instructions, 2019)

Una vez que ya tenemos localizadas las entradas donde queremos conectar nuestros dispositivos en el esquema (GPIO 0, GPIO 1, GPIO 2, GPIO 3, GPIO 4 Y GPIO 5) conectamos a nuestra placa **BerryClip** (Figura 5.11. Se vende desmontada, dispone de página web (Berryclip instructions, 2019) con amplia información para su montaje) siguiendo el esquema que se puede ver en la Figura 5.10, podremos así determinar cuáles son los pines pertenecientes a los leds que tenemos que conectar con sus correspondientes GPIO N. (Por ejemplo, el LED 1 que está en el pin 7 lo conectamos con el pin 8 que está en J3 de la placa HiFive)

En la Figura 5.12 se puede ver las conexiones realizadas entre HiFive1 y BerryClip.

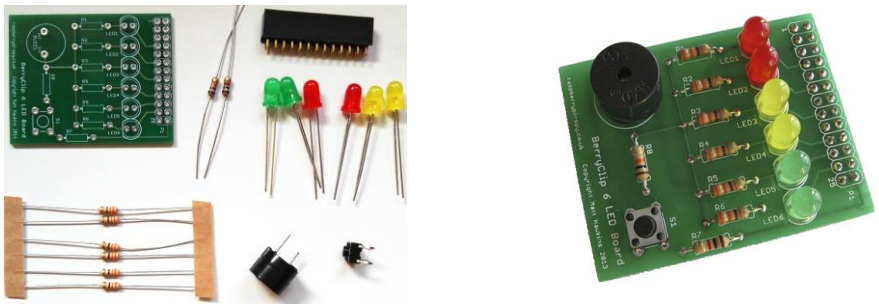

Figura 5.11: Placa BerryClip.

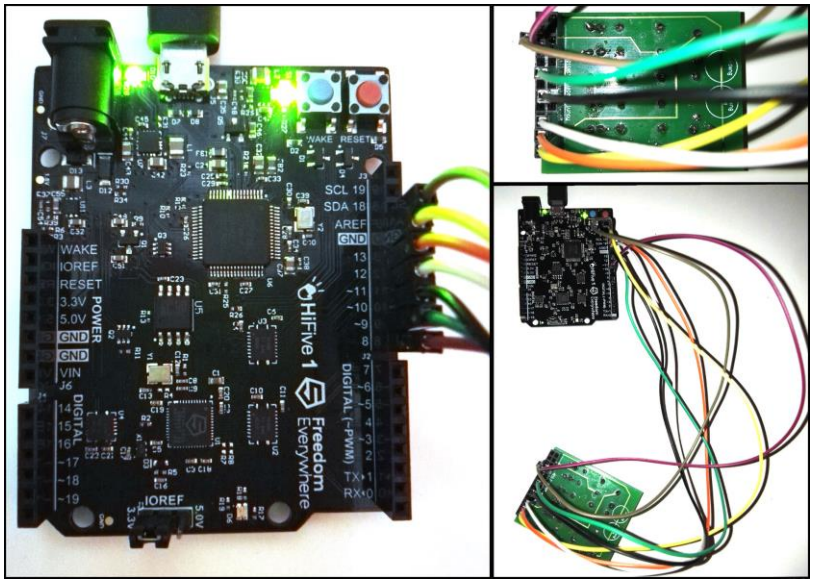

Figura 5.12: Conexión entre BerryClip y HiFive.

Una vez que ya tenemos todo conectado, compilaremos y subiremos el programa que hemos detallado en el apartado 5.2.3. como indicamos anteriormente.

5.3. FEDORA SOBRE RISCV

En el siguiente apartado veremos algunos de los Sistemas Operativos que hay disponibles para RISC-V. Principalmente la mayoría de ellos se han probado sobre máquinas virtuales, debido a la falta de placas con el hardware necesario para soportar la instalación completa de estos S.O., aunque si es cierto que ya se ha realizado la instalación del Sistema Operativo Fedora 29 GNOME sobre una placa con arquitectura RISC-V, como veremos a continuación en el punto 5.3.2.

Actualmente se sigue trabajando para tener más S.O. disponibles para RISC-V (información más detallada sobre el software disponible para RISC-V en (RISC-V, Software Status, 2019)) como Debian (Installing Debian on HiFive, 2018) y openSUSE (openSUSE: RISC-V, 2019), este último con imagen disponible para ejecutar sobre QEMU.

5.3.1. Fedora en QEMU

A continuación, mostraremos los pasos necesarios para poder instalar Fedora para RISC-V utilizando QEMU (QEMU, 2019), para una información más detallada visitar (Fedora wiki, 2019).

1. Instalar qemu:

```
git clone https://git.qemu.org/git/qemu.git
cd qemu
git submodule init
git submodule update –recursive
```

2. compilar riscv:

```
./configure–target-list=riscv64-softmmu && make
```

3. Si da problemas de librerías instalar:

```
apt-get install build-essential zlib1g-dev pkg-config libglib2.0-dev / binutils-dev libboost-all-dev autoconf libtool libssl-dev libpixman-1-dev / libpython-dev python-pip python-capstone virtualenv
```

4. Añadir el patch donde se encuentra qemu-system-riscv64:

```
export PATH=$PATH://home/elena/Escritorio/Fedora_RISCV/qemu/riscv64-softmmu
```

5. Descargar imagen de Fedora del siguiente enlace (Fedora Project, 2018).

6. Descomprimir imagen para arrancarla más adelante:

```
xzdec -d stage4-disk.img.xz > stage4-disk.img
```

7. Boot linux using RV64GC qemu:

```
qemu-system-riscv64 \
   -nographic \
   -machine virt \
   -smp 4 \
   -m 2G \
   -kernel bbl \
   -object rng-random,filename=/dev/urandom,id=rng0 \
   -device virtio-rng-device,rng=rng0 \
   -append “console=ttyS0 ro root=/dev/vda” \
   -device virtio-blk-device,drive=hd0 \
   -drive file=stage4-disk.img,format=raw,id=hd0 \
   -device virtio-net-device,netdev=usernet \
   -netdev user,id=usernet,hostfwd=tcp::10000-:22
```

8. Una vez arrancada la máquina usar el siguiente login y password

- login: root
- password: riscv

5.3.2. Fedora 29 GNOME en HiFive Unleashed

El Sistema Operativo Fedora ha sido instalado con éxito en la placa HiFive Unleashed (SiFive, HiFive Unleashed, s.f.), la primera y única placa actualmente, que dispone de un procesador RISC-V, multi-core y compatible con Linux.

Pero hay que tener en cuenta que para poder instalar Fedora es necesario disponer de hardware adicional además de la placa HiFive Unleashed. En la Tabla 5.5, se muestra todo lo necesario para poder crear un PC que utiliza la arquitectura RISC-V.

Hardware necesario
HiFive Unleashed
Microsemi HiFive Unleashed Expansion board
Radeon HD 6450 GPU card
PCIe to USB card
SATA Drive(HDD/SSD) or NVMe SSD

Tabla 5.5. Hardware necesario para crear PC con RISC-V

Toda la información detallada, junto con el software necesario y la instalación paso a paso se encuentra en (Build Fedora Gnome Desktop on RISC-V, 2018).

5.4. INSTALACIÓN DEL KERNEL ZEPHYR

Otra de las pruebas que hemos realizado con la placa HiFive1 es la instalación del kernel disponible Zephyr.

Zephyr se trata de un proyecto de código abierto, que consiste en un SO modular que admite múltiples arquitecturas. El núcleo de Zephyr viene del microkernel VxWorks comercial de Wind River para VxWorks. Más información en la siguiente dirección (Zephyr Project, 2019).

Para realizar la instalación hemos seguido los pasos indicados en los capítulos 3 y 5 de la guía "*RISC-V Getting Started Guide*" que podemos descargar en la siguiente dirección (RISC-V, Zephyr, RISC-V - Getting Started Guide, 2019).

Hay que tener en cuenta que debemos de tener correctamente configuradas y añadidas las siguientes variables:

```
export RISCV_OPENOCD_PATH=/<dirección_de_instalación>/freedom-e-sdk/openocd
export RISCV_PATH=/<dirección_de_instalación>/freedom-e-sdk/riscv-gnu-toolchain
export ZEPHYR_TOOLCHAIN_VARIANT=zephyr
export ZEPHYR_SDK_INSTALL_DIR="/opt/zephyr-sdk/"
export ZEPHYR_BASE=/<dirección_de_instalación>/Zephyr_INFO/zephyr
```

También deberemos ejecutar el siguiente script, que se encuentra en el código disponible de Zephyr:

```
. ./zephyr-env.sh
```

En el punto 5.4 Flashingl la línea correcta (si nos hemos descargado la última versión de freedom-e-sdk) sería:

```
sudo openocd/bin/openocd -f bsp/sifive-hifive1/openocd.cfg
```

Siguiendo los pasos indicados en la guía podremos ejecutar sobre el kernel Zephyr el programa de prueba *Hello World* que hay disponible.

Si vemos que no podemos acceder a la placa después de haber subido el programa de prueba *Hello World* en Zephyr, deberemos de activar el arranque seguro, para ello debemos presionar el botón RESET, cuando el led verde se ilumine hay que presionar otra vez el botón RESET, después de un segundo el led rojo parpadeará y nos podremos conectar a la placa. Esta información se encuentra disponible en la página 22 de (SiFive, SiFive HiFive1 Getting Started Guide, 2017).

6. CONCLUSIONES Y TRABAJO FUTURO

RISC-V se trata de una arquitectura RISC que se caracteriza por ser de código abierto, la cual tiene como base unos objetivos muy claros y sólidos en cuanto a su diseño y visión de futuro, centrados principalmente en el desarrollo de una arquitectura modular, bien estructurada, sencilla, versátil y eficiente.

RISC-V se trata de un desarrollo derivado de varios proyectos académicos de diseño de computadores, orientado a la investigación y la docencia. David Patterson, el cual lideró en 1980 cuatro generaciones de proyectos RISC, inspiró el nombre de "RISC Cinco" y es uno de los cuatro arquitectos de RISC-V. Andrew Waterman es otro de los arquitectos de RISC-V, además de ser el Jefe de Ingeniería y cofundador de SiFive, una startup fundada por los creadores de la arquitectura RISC-V para proporcionar chips basados en RISC-V.

En este trabajo no solo hemos estudiado la arquitectura, también la hemos comparado con otras arquitecturas existentes de tipo RISC y hemos realizado un estudio del soporte software disponible a nivel de sistema (SOs, compiladores, debuggers) y del hardware que podemos encontrar actualmente basado en RISC-V. Gran parte de estos conocimientos, los hemos podido aplicar posteriormente en una de las placas disponibles basadas en RISC-V, la placa HiFive1, desarrollada por la empresa que hemos comentado anteriormente llamada SiFive.

Esto solo es un inicio de lo que podemos encontrarnos con respecto a esta arquitectura, a partir de aquí son innumerables todos los trabajos que se pueden realizar, hay una gran cantidad de información disponible, gratuita y al alcance de todos, que nos permite aprender e investigar todo aquello que nos puede proporcionar una arquitectura de estas características.

Disponer de un ISA gratuito es algo muy positivo. Todo esto nos permite tener una herramienta muy potente no solo para la investigación, sino para la educación, permitiéndonos acceder a toda la información que necesitemos para estudiar el comportamiento de los computadores, su rendimiento, la creación de drivers y optimización de los programas. Para ello tenemos a nuestro alcance actualmente, la placa HiFive1 Rev B, la cual por un precio razonable (unos 60€)

nos permitirá disponer de una placa sobre la que poder trabajar, podremos enseñar de forma clara y directa un ISA sencillo y fácil de aprender.

Algo muy importante y fundamental a la hora de enseñar una arquitectura es disponer de herramientas para desarrolladores, las cuales nos permitan crear nuestros propios códigos, compilarlos y depurarlos y eso es lo primero que vemos que nos ofrece RISC-V, nos da acceso no solo a toda la información detallada de la arquitectura, sino que su comunidad pone a nuestro alcance una variedad más que aceptable de herramientas para desarrollar nuestro código, las cuales están en continua evolución, no solo son gratuitas y fáciles de instalar, sino que están bien documentadas y además hay foros muy activos en los cuales podemos encontrar respuestas a todas nuestras dudas.

En contraposición a lo anterior, actualmente no hay una placa disponible que nos permita instalar un SO con un coste reducido, al contrario de lo que pasa en el caso de la arquitectura ARM y la placa Raspberry Pi. Esto nos impediría la creación de un laboratorio basado únicamente en hardware RISC-V para la programación en ensamblador y el desarrollo de drivers

En mi opinión, estamos ante una iniciativa muy buena y apasionante, que nos permitirá disponer de información más detallada y de fácil acceso, para una educación más completa sobre la arquitectura de los computadores, permitirá desarrollar nuevos sistemas reduciendo su costo y dará acceso a un mayor número de desarrolladores. No solo eso, también será un aliciente, para que otras empresas propietarias de diferentes arquitecturas, se muestren más abiertas en cuanto a la información que muestran, permitiendo tanto a desarrolladores como a estudiantes y profesores, tener una visión más amplia de las arquitecturas actuales, su evolución y la mejor forma de trabajar con cada una de ellas.

Como trabajo futuro sería muy interesante poder estudiar la placa HiFive Unleashed (la cual apareció en el mercado mientras se desarrollaba este trabajo). Como hemos podido ver a lo largo de nuestro estudio, tiene un gran potencial y nos permitiría instalar un SO teniendo como base RISC-V, algo muy interesante para poder ver el comportamiento de esta arquitectura y ver qué posibilidades reales nos podría ofrecer a nivel educativo.

BIBLIOGRAFÍA

Arduino. (2019). *Arduino mega 2560 rev3*. Obtenido de https://store.arduino.cc/mega-2560-r3

arm. (28 de julio de 2010). Obtenido de Dhrystone and MIPs performance of ARM processors: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka3885.html

Berryclip instructions. (2019). Obtenido de http://www.raspberrypi-spy.co.uk/berryclip-6-led-add-on-board/berryclip-6-led-add-on-board-instructions/

Build Fedora Gnome Desktop on RISC-V. (26 de septiembre de 2018). Obtenido de https://github.com/westerndigitalcorporation/RISC-V-Linux

Coremark Scores. (2009-2019). Obtenido de https://www.eembc.org/coremark/scores.php

Crowd supply. (s.f.). Obtenido de HiFive Unleashed by SiFive: https://www.crowdsupply.com/sifive/hifive-unleashed

EEMBC. (s.f.). Obtenido de EEMBC's CoreMark.: https://www.eembc.org/coremark/

Fedora Project. (17 de octubre de 2018). Obtenido de Imagen Fedora para RISC-V: https://fedorapeople.org/groups/risc-v/disk-images/

Fedora wiki. (24 de junio de 2019). Obtenido de Architectures RISC-V Installing: https://fedoraproject.org/wiki/Architectures/RISC-V/Installing#Boot_under_QEMU

Fedora/RISC-V. (18 de agosto de 2018). Obtenido de https://fedoraproject.org/wiki/Architectures/RISC-V

Fernandez Montecelo, M. A. (22 de abril de 2017). *Debian*. Obtenido de Debian GNU/Linux port for RISC-V 64-bit (riscv64): https://people.debian.org/~mafm/posts/2017/20170422_debian-gnulinux-port-for-risc-v-64-bit-riscv64/

Fernandez Montecelo, M. A., & Keville, K. (s.f.). *Debian for RISC-V 64-bit (riscv64)*. Obtenido de http://riscv.mit.edu/

Fernandez Montecelo, M. A., & Merker, K. (19 de febrero de 2016). *Wiki Debian*. Obtenido de Debian for the RISC-V: https://wiki.debian.org/RISC-V

Fink, M. (s.f.). *RISC V ASM Tutorial*. Obtenido de https://www.youtube.com/watch?v=MnWI9qplfvA&index=4&list=PL6noQ0vZDAdh_aGvqKvxd0brXImHXMuLY

Foundation, Raspberry Pi. (2019). *Raspberry Pi 3 Model B+*. Obtenido de https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf

FreeRTOS Real Time Kernel (RTOS). (4 de septiembre de 2019). Obtenido de https://sourceforge.net/projects/freertos/

Hesham M. , A. (24 de junio de 2017). *seL4/RISC-V SMP support | seL4/RISC-V Tutorials Release*. Obtenido de http://heshamelmatary.blogspot.com/2017/06/update-sel4risc-v-smp-support-sel4.html

Installing Debian on HiFive. (12 de abril de 2018). Obtenido de https://wiki.debian.org/InstallingDebianOn/SiFive/HiFiveUnleashed

OpenMandriva. (2016-2019). Obtenido de openmandriva.org

openSUSE. (2019). *openSUSE Factory Port for RISC-V*. Obtenido de https://build.opensuse.org/project/show/openSUSE:Factory:RISCV

openSUSE: RISC-V. (13 de marzo de 2019). Obtenido de https://en.opensuse.org/openSUSE:RISC-V

OpenWrt. (17 de octubre de 2018). Obtenido de https://git.openwrt.org/?p=openwrt/staging/wigyori.git;a=shortlog;h=refs/heads/kitchensink-201810

OpenWrt. (26 de agosto de 2019). Obtenido de http://openwrt.uid0.hu/

Patterson, D., & Waterman, A. (2018). *Guía Práctica de RISC-V. El Atlas de una Arquitectura Abierta* (Primera Edición, 1.0.5 ed.). (A. Lemus, & E. Corpeño, Trads.) 1era. Obtenido de http://riscvbook.com/spanish/

PlarformIO. (2014-2019). Obtenido de PlatformIO is an open source ecosystem for IoT development: https://platformio.org/

QEMU. (27 de marzo de 2019). Obtenido de QEMU Documentation/Platforms/RISCV: https://wiki.qemu.org/Documentation/Platforms/RISCV#Description

Quality RTOS & Embedded Software. (s.f.). Obtenido de Using FreeRTOS on RISC-V Microcontrollers: https://www.freertos.org/Using-FreeRTOS-on-RISC-V.html

RISC-V. (2019). *OS and OS kernels*. Obtenido de https://riscv.org/software-status/#os-and-os-kernels

RISC-V. (2019). *Software Status*. Obtenido de https://riscv.org/software-status/

RISC-V. (31 de mayo de 2019). *Zephyr, RISC-V - Getting Started Guide*. Obtenido de https://buildmedia.readthedocs.org/media/pdf/risc-v-getting-started-guide/latest/risc-v-getting-started-guide.pdf

RISC-V Foundation. (2019). *RISC -V. Members at a Glance*. Obtenido de https://riscv.org/members-at-a-glance/

RTEMS. (2019). Obtenido de https://git.rtems.org/rtems/

RTEMS RISC-V. (2019). Obtenido de https://docs.rtems.org/branches/master/user/bsps/bsps-riscv.html#riscv

Runtime. (2016). *Apache Mynewt*. Obtenido de https://content.riscv.org/wp-content/uploads/2016/07/Wed930_riscv_apachemynewt_v1.2.pdf

Seeed. (3 de julio de 2019). *Face Count and Display: Using Grove AI HAT and Pi.* Obtenido de https://project.seeedstudio.com/SeeedStudio/face-count-and-display-using-grove-ai-hat-and-pi-3e100f

Seeed. (s.f.). *Grove AI HAT for Edge Computing*. Obtenido de https://www.seeedstudio.com/Grove-AI-HAT-for-Edge-Computing-p-4026.html

sel4riscv-manifest. (2019). Obtenido de https://github.com/heshamelmatary/sel4riscv-manifest/blob/master/sel4test-06032018.xml

SiFive. (s.f.). Obtenido de Página oficial: https://www.sifive.com

SiFive. (19 de diciembre de 2016). *Esquema placa HiFive*. Obtenido de https://sifive.cdn.prismic.io/sifive%2F080cdef9-4631-4c9b-b8f5-7937fbdec8a4_hifive1-a01-schematics.pdf

SiFive. (2016-2017). *SiFive FE310-G000 Manual v2p3*. Obtenido de https://sifive.cdn.prismic.io/sifive%2F4d063bf8-3ae6-4db6-9843-ee9076ebadf7_fe310-g000.pdf

SiFive. (2016-2017). *SiFive FE310-G000 Preliminary Datasheet v1p5*. Obtenido de https://sifive.cdn.prismic.io/sifive%2Ffeb6f967-ff96-418f-9af4-a7f3b7fd1dfc_fe310-g000-ds.pdf

SiFive. (3 de enero de 2017). *SiFive HiFive1 Getting Started Guide*. Obtenido de https://sifive.cdn.prismic.io/sifive%2F9c57065b-6d28-465b-b67d-f416894123a9_hifive1-getting-started-v1.0.2.pdf

SiFive. (2019). *Boards*. Obtenido de Paquetes RISC-V GNU Toolchain y openOCD: https://www.sifive.com/boards

SiFive. (2019). *Boards*. Obtenido de GNU MCU Eclipse: https://www.sifive.com/boards

SiFive. (24 de julio de 2019). *Open Source Software for Developing on the Freedom E Platform*. Obtenido de https://github.com/sifive/freedom-e-sdk

SiFive. (s.f.). *Core Designer*. Obtenido de Customize a Standard Core: https://scs.sifive.com/core-designer/

SiFive. (s.f.). *HiFive1*. Obtenido de https://www.sifive.com/boards/hifive1

SiFive. (s.f.). *HiFive1 Rev B*. Obtenido de https://www.sifive.com/boards/hifive1-rev-b

SiFive. (s.f.). *HiFive Unleashed*. Obtenido de https://www.sifive.com/boards/hifive-unleashed

Vak. (13 de febrero de 2017). *Datos benchmark Atmega2560*. Obtenido de https://vak.dreamwidth.org/440259.html

Visual Studio Code. (2019). Obtenido de https://code.visualstudio.com/

Waterman, A., & Asanovi´c, K. (7 de mayo de 2017). *The RISC-V Instruction Set Manual. Volume I: User-Level ISA. Version 2.2.* Obtenido de https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

Wikipedia. (4 de abril de 2018). Obtenido de David A. Patterson: https://es.wikipedia.org/wiki/David_A._Patterson

Wikipedia. (17 de septiembre de 2019). Obtenido de https://es.wikipedia.org/wiki/RISC-V

Zephyr. (2019). Obtenido de Primary Git Repository for the Zephyr Project: https://github.com/zephyrproject-rtos/zephyr/

Zephyr Project. (2015-2019). Obtenido de Supported Boards: RISCV32 Boards: https://docs.zephyrproject.org/latest/boards/riscv32/index.html

Zephyr Project. (2019). Obtenido de ¿What is the zephyr project?: https://www.zephyrproject.org/what-is-zephyr/

Todas las direcciones URL fueron comprobadas y validadas el 20 de septiembre de 2019.