



Part II

Overview of RISC-V SW Ecosystem

Bunnaroath Sou, SiFive



RISGV foundation now > 230 members.

RISC-V Free, open, extensible ISA for all computing devices





RISC-V Open Source Tools Status



Open Standards Foster Collaboration

- Palmer Dabbelt (SiFive): binutils, GCC, GDB, linux, glibc, QEMU
- Kito Cheng (SiFive): GCC and newlib
- Jim Wilson (SiFive): binutils and GCC
- Darius Rad (Bluespec): glibc
- Andrew Waterman (SiFive): binutils, GCC, and glibc
- DJ Delorie (RedHat): glibc
- Andrew Burgess (Embecosm)
- Alex Bradbury (lowRISC): LLVM



Berkeley
UNIVERSITY OF CALIFORNIA



redhat.



bluespec



lowRISC

On Compiler

- GCC and Binutils has been upstreamed
 - since 7.1 and 2.8 (May 2017)



On Libraries

- Newlib* has been upstreamed
 - since 2.50 (March 2018)
- Glibc has been upstreamed
 - since 2.27 (February 2018)

On Debugger

- GDB has been upstreamed
 - since 8.2 (March 2018)



- **`-march=ISA` (selects the architecture to target)**
 - Controls which instructions set and registers to use
 - Determines set of implementations a program run on
 - Toolchain support three base ISAs (v2.2)
 - RV32I: A load-store ISA with 32, 32-bit GP int registers.
 - RV32E: RV32I with only 16 int registers, for embedded
 - RV64I: 64-bit flavor of RV32I GP int registers are 64-bits
 - Plus these extensions
 - M: Integer Multiplication and Division
 - A: Atomic Instructions
 - F: Single-Precision Floating-Point
 - D: Double-Precision Floating-Point
 - C: Compressed Instructions
 - ISA strings defined by appending extensions to base ISA
 - For example `-march=rv32im`, `-march=rv64imafdc`



GCC RISC-V Emulation function

- Use when a particular operation support

- For example, this C code

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

- Compile directly to a FP multiplication instruction with D extension

```
$ riscv64-unknown-elf-gcc test.c -march=rv64imafdc -mabi=lp64  
dmul:  
    fmul.d   fa0,fa0,fa1  
    ret
```

- But compile to an emulation routine without the D extension

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



- **-mabi=ABI (selects the ABI to target)**
 - Controls the calling convention (which arguments are passed into which registers) and the layout of data in memory
 - Two integer ABIs and three floating-point ABIs, which together are treated as a single ABI string
 - Two integer ABIs follow the standard ABI naming scheme:
 - `ilp32`: `int`, `long`, and pointers are all 32-bits long. `long long` is a 64-bit type, `char` is 8-bit, and `short` is 16-bit
 - `lp64`: `long` and pointers are 64-bits long, while `int` is a 32-bit type. The other types remain the same as `ilp32`



- Three floating-point ABIs are RISC-V specific addition:
 - "" (the empty string): No floating-point arguments are passed in registers.
 - f: 32-bit and smaller floating-point arguments are passed in registers. This ABI requires the F extension, as without F there are no floating-point registers.
 - d: 64-bit and smaller floating-point arguments are passed in registers. This ABI requires the D extension.



GCC RISC-V Treatment of ISA and ABI String

- ISA and ABI treated as two separate arguments, eg:
- **-march=rv32imafdc -mabi=ilp32d:**
 - Hardware floating-point instructions can be generated and floating-point arguments are passed in registers.
 - Equivalent to ARM's GGC -mfloat-abi=hard argument
- **-march=rv32imac -mabi=ilp32:**
 - No floating-point instructions can be generated and no floating-point arguments are passed in registers.
 - Equivalent to ARM's GGC -mfloat-abi=soft argument
- **-march=rv32imafdc -mabi=ilp32:**
 - Hardware floating-point instructions can be generated, but no floating-point arguments will be passed in registers.
 - Equivalent to ARM's GGC -mfloat-abi=softfp argument
 - Used when interfacing with soft-float binaries on a hard-float system.
- **-march=rv32imac -mabi=ilp32d:**
 - Illegal, ABI requires floating-point arguments are passed in registers
 - ISA defines no floating-point registers to pass them in



GCC RISC-V Command Line `-mtune=`

- **`-mtune=CODENAME` selects the microarchitecture to target.**
 - Informs GCC about the performance of each instruction, for target-specific optimizations
 - SiFive have a number of tuning model



RISC-V Relocation: How it works

- **Concept due split between compiler and linker**

- Exist to pass information between the compiler and linker
- Compiler emit tags, eg R_RISCV_HI20 and R_RISCV_RELAX
- Allow linker to resolve address in the final output ELF exec

```

$ riscv64-unknown-linux-gnu-objdump -d -t -r relocation.o
$ riscv64-unknown-linux-gnu-objdump -d -t -r relocation
relocation:      file format elf64-littleriscv
SYMBOL TABLE:
0000000000000000 g      F .text
0000000000000010      O *(
SYMBOL TABLE:
0000000000012038 g      O .bss 0000000000000010      global
Disassembly of section .text:
int main()
return 0
}
0000000000000000 main:
0: 000007b7
Disassembly of section .text:
0:
0: 0000000000010330 main:
4: 0007b503      10330:      67c9      lui      a5,0x12
4: 10332:      0387b503   ld      a0,56(a5) # 12038 gl
4: 10336:      00a03533   snez   a0,a0
8: 00a03533      1033a:      8082      ret
c: 8082

```



RISC-V Linker Relaxation: How it works (1)

- Mechanism for optimizing programs at link-time, instead of at compile-time
 - Compiler emit tags, R_RISCV_CALL and R_RISCV_RELAX

```
$ cat test.c
int func(int a) __attribute__((noinline));
int func(int a) {
    return a + 1;
}

int _start(int a) {
    return func(a);
}

$ riscv64-unknown-linux-gnu-gcc test.c -o test -O3
$ riscv64-unknown-linux-gnu-objdump -d -r test.o
test.o:      file format elf64-littleriscv
Disassembly of section .text:

0000000000000000 <func>:
   0: 2505          addiw  a0,a0,1
   2: 8082          ret

0000000000000004 <_start>:
   4: 0000317      auipc  ra,0x0
                        4: R_RISCV_CALL func
                        4: R_RISCV_RELAX
   8: 00030067     jr     ra
```

```
$ riscv64-unknown-linux-gnu-objdump -d -r test
test:      file format elf64-littleriscv
```

Disassembly of section .text:

00000000000010078 <func>:

```
10078:      2505          addiw  a0,a0,1
1007a:      8082          ret
```

0000000000001007c <_start>:

```
1007c:      ffdff06f     j     10078 <func>
```



RISC-V Linker Relaxation: How it works (2)

- **Two unconditional control transfer instructions in RISC-V ISA, jalr and jal**
 - jalr, jumps to an absolute address as specified by an immediate offset from a register
 - jal, jumps to a pc-relative offset as specified by an immediate.
 - In this example, auipc+jalr pair can address a 32-bit signed offset from the current PC (0x1007c)
 - While jal can only address a 21-bit signed offset from the current PC (0x1007c)
 - Because linker knows the call from `_start` to `func` fits within the 21-bit offset of the jal instruction, it uses a single instruction.
 - A proxy for twice the speed



RISC-V Handling of Multilib

- **Mechanism for handling multiple sets of system libraries**
 - eg. 32bits application in 64bits architecture
 - Mixing soft-float and hard-float systems
 - Building a single compiler that targets many RISC-V systems
- **Modular ISA, allow multilib implementation to be cleaner**
 - Well known naming scheme for all ISA targets
- **Standard set of ABIs was already known for RISC-V**
 - C type sizes: ilp32 vs lp64
 - Floating-point registers: none, single, single and double
- **Derive a set of multilib by using script**
 - <https://www.sifive.com/blog/all-aboard-part-5-risc-v-multilib>



RISC-V Example Script to Derive Multilib List

```
#!/bin/bash
for abi in ilp32 ilp32f ilp32d lp64 lp64f lp64d; do
  for isa in rv32e rv32i rv64i; do
    for m in "" m; do
      for a in "" a; do
        for f in "" f fd; do
          for c in "" c; do
            readlink -f $(riscv64-unknown-elf-gcc -march=$isa$m$a$f$c -mabi=$abi -print-
search-dirs | grep ^libraries | sed 's:/:/g') | grep 'riscv64-unknown-elf/lib' | grep -
ve 'lib$' | sed 's@^.*\/lib/@@' | while read path; do
              echo "riscv64-unknown-elf-gcc -march=$isa$m$a$f$c -mabi=$abi => $path"
            done
          done
        done
      done
    done
  done
done
```




RISC-V Embedded Multilib List

riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 => rv32i/ilp32

riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 => rv32i/ilp32

riscv64-unknown-elf-gcc -march=rv32iac -mabi=ilp32 => rv32iac/ilp32

riscv64-unknown-elf-gcc -march=rv32im -mabi=ilp32 => rv32im/ilp32

riscv64-unknown-elf-gcc -march=rv32imc -mabi=ilp32 => rv32im/ilp32

riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 => rv32imac/ilp32

riscv64-unknown-elf-gcc -march=rv32imafc -mabi=ilp32f => rv32imafc/ilp32f

riscv64-unknown-elf-gcc -march=rv32imafdc -mabi=ilp32f => rv32imafc/ilp32f

riscv64-unknown-elf-gcc -march=rv64imac -mabi=lp64 => rv64imac/lp64

riscv64-unknown-elf-gcc -march=rv64imafdc -mabi=lp64d => rv64imafdc/lp64d



RISC-V Linux Multilib List

riscv64-unknown-linux-gnu-gcc -march=rv32ima -mabi=ilp32 => lib32/ilp32
riscv64-unknown-linux-gnu-gcc -march=rv32imac -mabi=ilp32 => lib32/ilp32
riscv64-unknown-linux-gnu-gcc -march=rv32imaf -mabi=ilp32 => lib32/ilp32
riscv64-unknown-linux-gnu-gcc -march=rv32imafc -mabi=ilp32 => lib32/ilp32
riscv64-unknown-linux-gnu-gcc -march=rv32imafd -mabi=ilp32 => lib32/ilp32
riscv64-unknown-linux-gnu-gcc -march=rv32imafdc -mabi=ilp32 => lib32/ilp32
riscv64-unknown-linux-gnu-gcc -march=rv32imafd -mabi=ilp32d => lib32/ilp32d
riscv64-unknown-linux-gnu-gcc -march=rv32imafdc -mabi=ilp32d => lib32/ilp32d
riscv64-unknown-linux-gnu-gcc -march=rv64ima -mabi=lp64 => lib64/lp64
riscv64-unknown-linux-gnu-gcc -march=rv64imac -mabi=lp64 => lib64/lp64
riscv64-unknown-linux-gnu-gcc -march=rv64imaf -mabi=lp64 => lib64/lp64
riscv64-unknown-linux-gnu-gcc -march=rv64imafc -mabi=lp64 => lib64/lp64
riscv64-unknown-linux-gnu-gcc -march=rv64imafd -mabi=lp64 => lib64/lp64
riscv64-unknown-linux-gnu-gcc -march=rv64imafdc -mabi=lp64 => lib64/lp64
riscv64-unknown-linux-gnu-gcc -march=rv64imafd -mabi=lp64d => lib64/lp64d
riscv64-unknown-linux-gnu-gcc -march=rv64imafdc -mabi=lp64d => lib64/lp64d



On Compiler/Assembler

- Upstreamed in experimental mode as of 8.0
- Non-experimental support planned for 9.0
- Compile and run GCC torture suite
 - {RV32I, RV32IM, RV32IFD} + 'C'
- Compile and run all torture suite tests
 - for RV64I at O1, O2, O3, and Os
- MC-layer now support
 - RV32IMAFDC + RV64IMAFDC,
- CodeGen support
 - RV32IMFDC and RV64I



Upstreamed since 2.12 (April 2018)

Sources

- <https://www.qemu.org/download/#source>

Build and Boot instructions

- <https://wiki.qemu.org/Documentation/Platforms/RISCV>

Contributors

- Sagar Karandikar (University of California, Berkeley), Bastian Koppelman (University of Paderborn), Alex Suykov, Stefan O'Rear, Michael Clark and Alistair Francis (Western Digital)



SiFive



Berkeley
UNIVERSITY OF CALIFORNIA



Open On -Chip Debugger

OpenOCD

- Ability to connect to embedded target for debugging
- Not yet Upstreamed
- Upstreaming is planned, but low priority
- <https://github.com/riscv/riscv-openocd>
- Tim Newsome, Megan Wachs, Palmer Dabbelt (SiFive)



Linux Kernel and FreeBSD Port



Upstream Kernel boots

- Boots on QEMU
 - Since v4.15 (December 2017)
- Well supported
- 340+ commits since
- Alan Kao (Andes), Anup Patel (WD), Aishh Patra (WD), Christoph Hellwig, David Abdurachmanov, Palmer Dabbelt, Paul Walmsley (SiFive), Zong Li (SiFive), Jim Wilson (SiFive)



Berkeley
UNIVERSITY OF CALIFORNIA



SiFive



Western
Digital®





Embedded

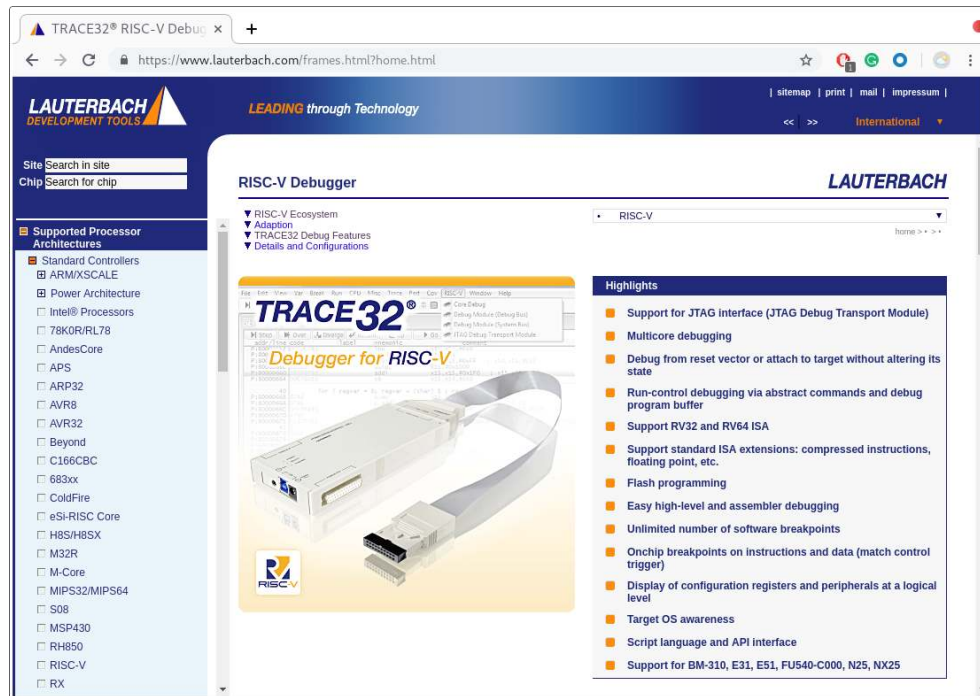


RISC-V Embedded Software Ecosystem

- **SiFive Freedom Studio**
 - Eclipse CDT, GNU MCU Eclipse, pre-built GCC, and OpenOCD
 - Built on Open Source technology
- **AndesSight** - Eclipse based IDE for RISC-V
- **SEGGER** - JLINK Probe and Embedded Studio RISC-V IDE
- **Lauterbach** - Lauterbach TRACE32 for silicon bring up and debug
- **IAR** - IAR Embedded Workbench with SiFive support in development
- **Ashling** - RiscFree C/C++ IDE for development and debug
- **Embedded Software and Operating Systems**
 - Bare Metal
 - FreeRTOS
 - Zephyr OS
 - RTEMS
 - Express Logic – Thread X
 - Micrium - μ COS
 - RIOT
 - NuttX
- **Imperas** - Simulation models and tools for early software development
- **UltraSoC** - IP and tooling supporting SiFive instruction trace



- **Commercial RISC-V Debug Hardware and Software**
 - Available in the market for over a year
 - TRACE32 for silicon bring up and debug
- **Support for all SiFive IP and platforms**
 - RV32/RV64
 - Multicore
 - Heterogeneous Multicore
- **Collaborate with UltraSoC**
 - Providing powerful debug, trace and logic analyzer tools

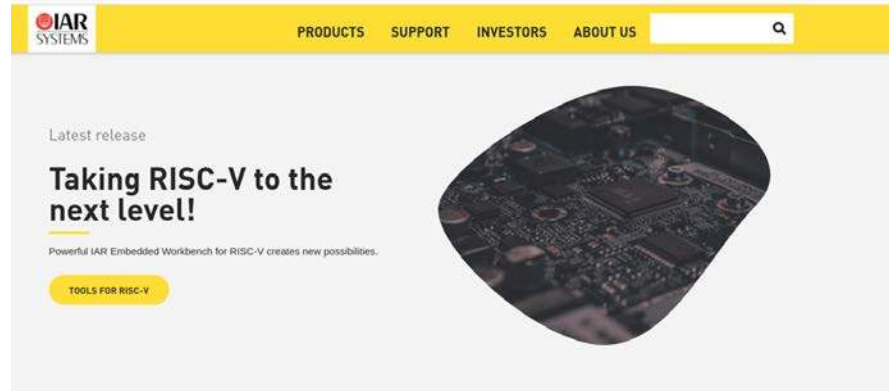


- **Commercial RISC-V Debug Hardware and Software**
 - Available since late 2017
 - JLINK Probe and Embedded Studio RISC-V IDE
- **Support for all SiFive IP and platforms**
 - Great support for single core RV32
 - RV64 and Multicore support in beta now
- **SEGGER RTT fully supported**
 - High Speed communication with host debugger

The screenshot shows the SEGGER website's RISC-V Support page. At the top, there is a navigation bar with the SEGGER logo and links for Products, Downloads, Purchase, Support, and About Us. Below the navigation bar, a large banner features the text "RISC-V Support" and a description: "The SEGGER Software Platform, including development tools, debug probes and middleware, provides a comprehensive one-stop solution for complete product development with microcontrollers based on the open RISC-V architecture. It simply works!". To the right of the text is the RISC-V logo. Below the banner, there are three product tiles under the heading "SEGGER Platform for RISC-V":

- J-Link RISC-V Support**: Represented by a stylized 'J' logo.
- Embedded Studio for RISC-V**: Represented by a 'C++' logo.
- embOS for RISC-V**: Represented by an 'OS' logo.

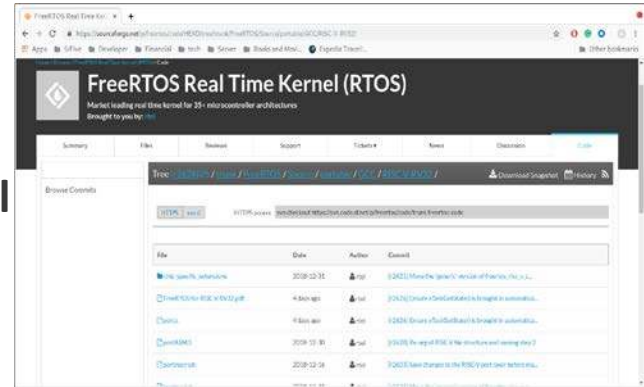
- **Very well known in the Embedded Space**
 - Have been around for a long time
20+ years
 - Support many different architectures
- **Very well known for their embedded toolchain**
 - The first proprietary RISC-V Compiler
 - Compiler is known for both code Density and Performance





FreeRTOS / AWS FreeRTOS

- **De facto real time operating system for small, low-power devices**
- **Amazon FreeRTOS which extends the FreeRTOS kernel with software libraries**
 - Through the effort of Richard Barry
- **RISC-V support now available in FreeRTOS Kernel 10.2 (Feb 2019)**
 - With 10.2.1 added support for RV64
 - QEMU emulation of SiFive Hifive1
 - OpenISA's [VEGAboard](#)
 - Antmicro's [Renode emulator](#) of Microchip M2GL025 Creative Board
- **SiFive will also port FreeRTOS to Freedom Metal for SiFive devices**
 - With FreeRTOS examples added to Freedom E SDK and Freedom Studio

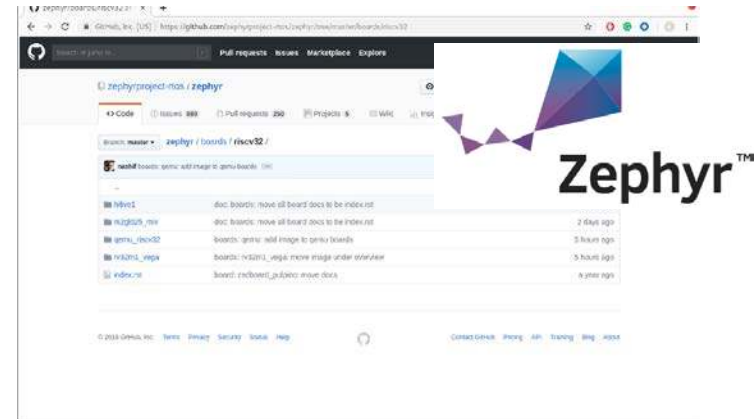


Amazon FreeRTOS



Zephyr - On RISC -V (HiFive1) is Upstream and Well Supported

- **Zephyr RTOS**
 - Open Source
 - Well defined development cycle
 - Performant and scalable
 - Strong software stacks
- **Zephyr already Support**
 - HiFive1 Rev B board supported in the latest Zephyr LTS release
- **Used in a real product**
 - <http://badge.antmicro.com/>
- **Zephyr SDK comes with RISC-V toolchains**
- **Contributors**
 - Karol Gugala (Antmicro), Peter Gielda (Antmicro), Nathaniel Graff (SiFive)





Getting Started with Zephyr on QEMU

Install Zephyr SDK

- Get Zephyr SDK, and follow the instructions in Zephyr's Documentation for your platform of choice (Linux, macOS, Windows)

Get the Zephyr Sources

- Zephyr RTOS is developed on GitHub. Get the sources by:
- `git clone https://github.com/zephyrproject-rtos/zephyr.git`
- `cd zephyr`

The RISC-V foundation published a getting started guide:

<https://buildmedia.readthedocs.org/media/pdf/risc-v-getting-started-guide/latest/risc-v-getting-started-guide.pdf>

Build a Zephyr example

- `source zephyr-env.sh`
- `cd samples/hello_world`
- `mkdir build && cd build`
- `cmake -DBOARD=qemu_riscv32 ..`
- `make -j$(nproc)`

The zephyr elf is in `zephyr/zephyr.elf`

Running Zephyr in QEMU

- `make run`



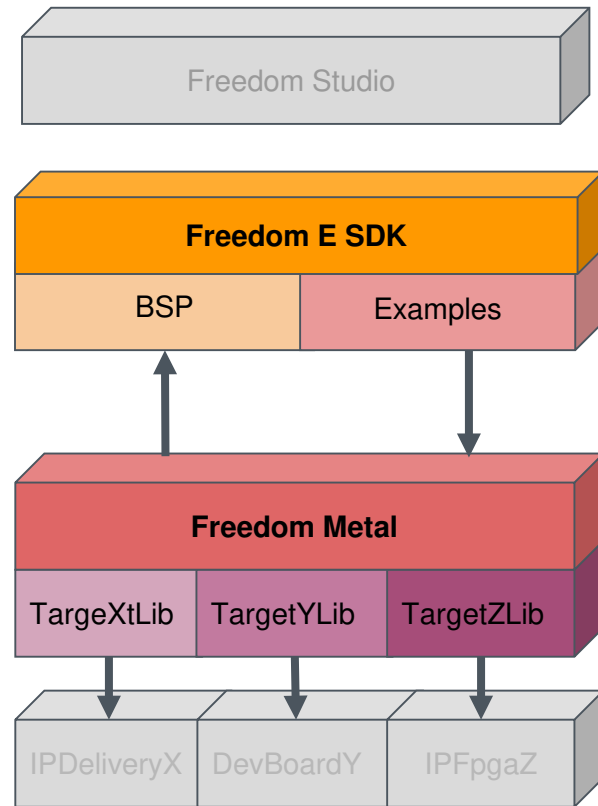
Open Source Bare Metal SW Stack from SiFive

What is Freedom E SDK

- Embedded development kit providing a command line driven workflow with Examples and Utilities including BSP's for SiFive targets
 - SiFive Qemu for E31, S51 CoreIP
 - Freedom E310 FPGA board
 - SiFive Development Boards
- Examples use Freedom Metal to provide portability
- Open source repository
 - <https://github.com/sifive/freedom-e-sdk>

What is Freedom Metal

- Library for writing Portable, Bare Metal SW for all SiFive devices
 - A Bare Metal C application environment
 - An API for controlling CPU features and peripherals
 - The ability to retarget to any SiFive RISC-V product
 - A RISC-V hardware abstraction layer (HAL)
- Uses BSP's to provide target adaptation
- Open source repository
 - <https://github.com/sifive/freedom-metal>





SiFive Freedom Studio

Freedom Studio provides an Eclipse based GUI for developing and debugging Freedom E SDK applications

SiFive Cloud Products Company Community Contact Sales Workspace B

Freedom Studio

Freedom Studio is the fastest way to get started programming with your SiFive hardware. Freedom Studio is built on top of the popular Eclipse IDE and packaged with a prebuilt toolchain and example projects from the Freedom E SDK. Freedom Studio is compatible with all SiFive RISC-V development boards.

We highly recommend downloading and reviewing the [Freedom Studio User Manual](#) before downloading and installing Freedom Studio. This manual has important information about preparing your host system to help you get up and running as quickly as possible.

Download Freedom Studio — v2019.05.0

[Windows](#)

[macOS](#)

[Linux](#)

local_interrupts Debug (GDB OpenOCD Debugging)
local_interrupts.elf
Thread #1 (Suspended : Breakpoint)
button_2_isr() at local_interrupts.c:115 0x4040
handle_trap() at init.c:75 0x404001da
trap_entry() at entry.S:50 0x40400280
openocd
riscv64-unknown-elf-gdb

<https://www.sifive.com>



Freedom E SDK New Project Wizard using QEMU

FreedomStudio File Edit Naviga

Create a Freedom E SDK Project

C Project
Create C project of selected type

Project name:

Use default location

Location:

Choose file system: default

Project type:

- GNU Autotools
- Executable
- Shared Library
- Static Library
- Makefile project
- Empty Project
- Freedom E SDK Project**

Show project types and toolchains only if they are

Select Target

- e21-arty
- e76-arty
- freedom-e310-arty
- qemu-sifive-e31**
- qemu-sifive-s51
- qemu-sifive-u54
- sifive-hifive-unleashed
- sifive-hifive1
- sifive-hifive1-rev01
- freedom-meta

Select Example Project

- hello

hello
A simple "Hello, World!" program.

Options

Project name:

(yo) Name

Create a debug toolchain

Project Tree:

- qemu-sifive-e31-hello
 - Build Targets
 - Binaries
 - Archives
 - Includes
 - freedom-metal/src
 - src
 - debug
 - hello.eif - (none/ie)
 - hello.hex
 - hello.lst
 - hello.map
 - hello.c
 - LICENSE
 - LICENSE.Apache2
 - LICENSE.MIT
 - Makefile
 - README.md
 - bsp
 - freedom-metal
 - debug.mk
 - Makefile
 - release.mk

Console:

```
qemu qemu-sifive-e31-hello [SiFive GDB QEMU Debugging] qemu-system-riscv64: QEMU emulator version 3.1.0 (SiFive QEMU 3.1.0-2019.05.1)
Copyright (c) 2003-2018 Fabrice Bellard and the QEMU Project team
Hello, World!
```

Debugger Console:

```
qemu qemu-sifive-e31-hello [SiFive GDB QEMU Debugging] $(qemu_gdb)
For help, type 'help'.
Type 'apropos word' to search for commands related to 'word'.
Warning: the current language does not match this frame.
Note: automatically using hardware breakpoints for read-only instructions.

Temporary breakpoint 1, main () at hello.c:7
7      printf("Hello, World!\n");
```

hello.c

```
/* Copyright 2019 SiFive, Inc */
/* SPDX-License-Identifier: Apache-2.0 */

#include <stdio.h>

int main() {
    printf("Hello, World!\n");
}
```

Outline:

- stdio.h
- main(): int



Freedom E SDK New Project Wizard using FPGA Board

The screenshot shows the FreedomStudio New Project Wizard interface. The main window is titled "C Project" and contains the following sections:

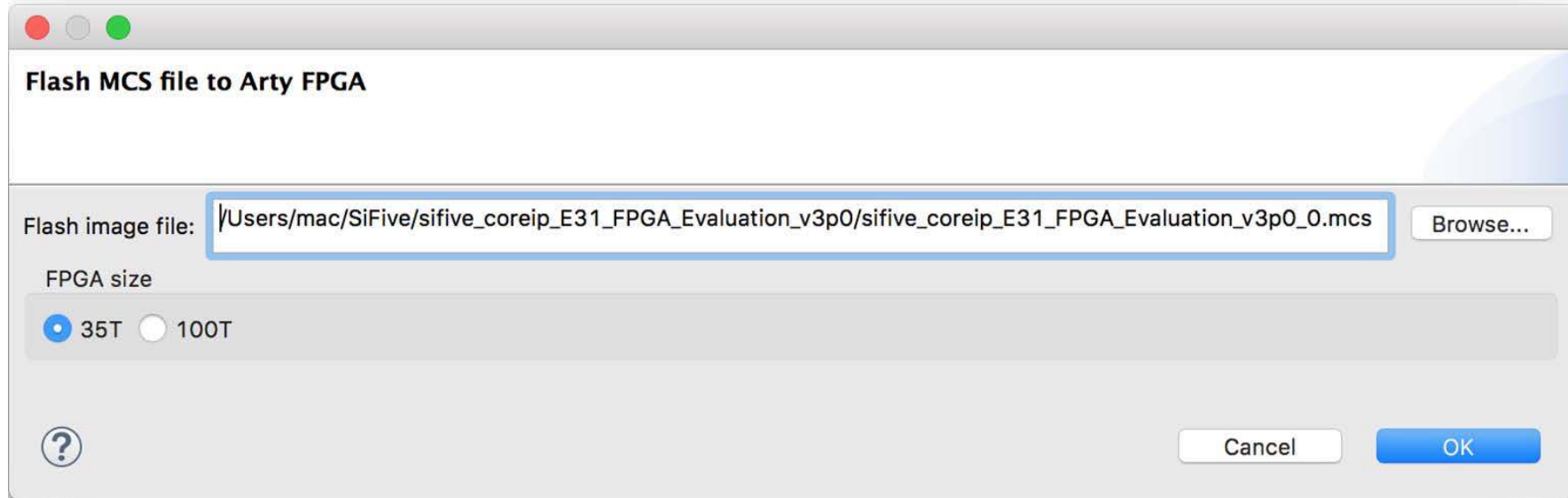
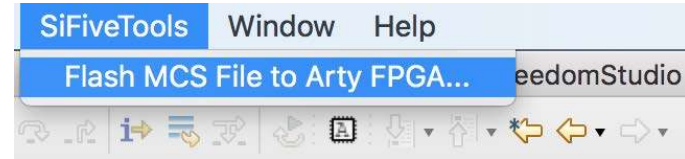
- Select Target:** A dropdown menu is set to "coreip-e31-arty". Below it, text describes the SiFive E31 Standard Core as the world's most deployed RISC-V core, co-designed alongside the RISC-V ISA, resulting in a power-efficient core for IoT, storage, and industrial applications. It also states, "This FPGA core target is ideal for m...".
- Project name:** A text field containing "my-project".
- Use default location:** A checked checkbox.
- Location:** A text field showing the path "/Users/mac/SiFive/FreedomStudio-2019-".
- Choose file system:** A dropdown menu set to "default".
- Project type:** A list of project types including GNU Autotools, Executable, Shared Library, Static Library, Makefile project, Empty Project, and Freedom E SDK Project. The "Freedom E SDK Project" is selected.
- Show project types and toolchains only if they are:** A checked checkbox.
- Select Example Program:** A list of example programs including hello, example-itim, software-interrupt, timer-interrupt, local-interrupt, return-fail, return-pass, example-pmp, example-spi, empty, sifive-welcome, and dhrystone. "sifive-welcome" is selected.
- Create an OpenOCD debug launch for this:** A checked checkbox.
- Create a freedom-e-sdk debug launch:** A section with a "Debug Launch" sub-section. It contains text explaining that a debug launch configuration dialog will be opened. A checked checkbox "Create a debug launch configuration for this project" is present. Below it, a "Debug launch type:" dropdown menu is open, showing "OpenOCD" (checked) and "J-Link".

At the bottom of the wizard, there are navigation buttons: "< Back", "Next >", "Cancel", and "Finish".



Flashing Arty FPGA Image - not using Vivado

- **Writes driver .bit file to FPGA via Xc3sprog (Digilent)**
 - E31 image in FPGA RAM (not Flash)
 - E31 image allows OpenOCD to access Flash
- **Writes MCS (ihex) file to Flash via OpenOCD (Olimex)**





Start debugging

my-workspace - my-project/src/sifive-welcome.c - FreedomStudio

Project Explorer: my-project, Binaries, Archives, Includes, freedom-metal/src, src, debug, sifive-welcome.elf - [none/le], sifive-welcome.hex, sifive-welcome.lst, sifive-welcome.map, release

Console: openocd my-project [SiFive GDB OpenOCD Debugging] openocd
(3289) hpmcounter24h (/32)
(3290) hpmcounter25h (/32)
(3291) hpmcounter26h (/32)
(3292) hpmcounter27h (/32)
(3293) hpmcounter28h (/32)
(3294) hpmcounter29h (/32)
(3295) hpmcounter30h (/32)
(3296) hpmcounter31h (/32)
(3922) mvendorid (/32)

Debug Console: Thread #1 (Suspended : Breakpoint)
main() at sifive-welcome.c:87 0x40400382
openocd
riscv64-unknown-elf-gdb

Code Editor: sifive-welcome.c

```
int main (void)
{
    int rc, up_cnt, dn_cnt;
    struct metal_led *led0_red, *led0_green, *led0_blue;

    // This demo will
    led0_red = metal_l
    led0_green = metal
    led0_blue = metal
    if ((led0_red == N
        printf("At led
    return 1;
}

// Enable each LED
metal_led_enable(l
metal_led_enable(l
metal_led_enable(led0_blue);

// All Off
```

Expression Watch Window:

Expression	Type	Value
led0_red	struct metal_led *	0x4040436a
vtable	const struct metal_led_vtable *	0x87aabfd9

Variable Watch Window:

Name	Type	Value
rc	int	0x0
led0_red	struct metal_led *	0x4040436a
vtable	const struct metal_led_vtable *	0x87aabfd9
led0_green	struct metal_led *	0x1
led0_blue	struct metal_led *	0x0

Details for led0_red:

Name : led0_red
Details: 0x4040436a <__libc_init_array+108>
Default: 0x4040436a <__libc_init_array+108>
Decimal: 1077953386
Hex: 0x4040436a
Binary: 1000000010000000100001101101010
Octal: 010020041552

Bottom Right Legend:

- timer_isr(int, void*) : void
- wait_for_timer(struct metal_led*)
- main(void) : int



Viewing elements and memory

(x)= Variables Expressions Memory

Expression	Type	Value
(x)=tmr_id	int	0x0
▼ \$pc;\$sp	Group-pattern	2 unique matches (Default)
➔ \$pc	void (*)()	0x4040048e
➔ \$sp	void *	0x80001490
▼ led0_red	struct metal_led *	0x80000fb8
▼ vtable	const struct metal_led_vtable *	0x40404b70
➔ led_exist	int (*)(struct metal_led *, char *)	0x40403874
➔ led_enable	void (*)(struct metal_led *)	0x404038b0
➔ led_on	void (*)(struct metal_led *)	0x404038b0
➔ led_off	void (*)(struct metal_led *)	0x404038b0
➔ led_toggle	void (*)(struct metal_led *)	0x404038b0

+ Add new expression

Name : led0_red
 Details:0x80000fb8 <__metal_dt_led_0
 Default:0x80000fb8 <__metal_dt_led_0
 Decimal:-2147479624
 Hex:0x80000fb8
 Binary:10000000000000000000000000111110111
 Octal:020000007670

(x)= Variables Expressions Memory

Monitors

0x80000000

Address	0 - 3	4 - 7	8 - B	C - F
0000000080000000	FFFFFFA8	800002EC	80000354	800003BC
0000000080000010	00000000	00000000	00000000	00000000
0000000080000020	00000000	00000000	00000000	00000000
0000000080000030	00000000	00000000	00000001	4040069C

Ox80000000 : Ox80000000 <Hex Integer>

Export Memory

Format: Plain Text

Start address: 0x80000000 End address: 0x80000010 Length: 16

File name: /Users/mac/SiFive/my-memory-export.hex



Viewing registers and terminal output

<project>/bsp/design.reglist file:

zero
ra
sp
gp
tp
t0
t1
t2
fp
s1
a0
a1
a2
a3
a4
a5
a6
a7
s2
...

Name	Value	Description
General Registers		General Purpose
zero	0x0	
ra	0x4040048e	
sp	0x80001490	
gp	0x80001840	
tp	0x0	
t0	0x40401212	
t1	0x40000	
t2	0xffffffff	
fp	0x800014b0	
s1	0x0	
a0	0xffffffff	
a1	0xa	
a2	0x22	
a3	0x22	
a4	0x22	
a5	0xffffffff	
a6	0x1f	
a7	0x0	
s2	0x40403b10	

Name : ra
Hex:0x4040048e
Decimal:1077937294
Octal:010020002216
Binary:1000000010000000000010010001110
Default:0x4040048e <main+276>

The screenshot shows an IDE window with a terminal output area displaying a boot sequence for SiFive, Inc. The output includes a series of '5555' characters forming a pattern, followed by 'Welcome to SiFive!'.

Overlaid on the terminal is a 'Launch Terminal' dialog box. The 'Choose terminal' dropdown is set to 'Serial Terminal'. The 'Serial port' dropdown is set to '/dev/cu.usbserial-210319A28D950'. Other settings include Baud rate: 115200, Data size: 8, Parity: None, Stop bits: 1, and Encoding: Default (ISO-8859-1). The 'OK' button is highlighted.



Viewing disassembly

```
.c sifive-welcome.c
    return 4;
}
cpu_intr = metal_cpu_interrupt_controller(cpu0);
if (cpu_intr == NULL) {
    printf("CPU interrupt controller is null.\n");
    return 3;
}
metal_interrupt_init(cpu_intr);

// display welcome banner
display_banner();

// Setup Timer and its interrupt so we can toggle LEDs on 1s cadence
tmr_intr = metal_cpu_timer_interrupt_controller(cpu0);
if (tmr_intr == NULL) {
    printf("TIMER interrupt controller is null.\n");
    return 4;
}
metal_interrupt_init(tmr_intr);
tmr_id = metal_cpu_timer_get_interrupt_id(cpu0);
rc = metal_interrupt_register_handler(tmr_intr, tmr_id, timer_isr, cpu0);
if (rc < 0) {
    printf("TIMER interrupt handler registration failed\n");
    return (rc * -1);
}

// Lastly CPU interrupt
if (metal_interrupt_enable(cpu_intr, 0) == -1) {
    printf("CPU interrupt enable failed\n");
    return 6;
}
}

Outline Disassembly Terminal 1
Enter location here
114     if (cpu_intr == NULL) {
40400478:  li    a5,3
4040047a:  j     0x4040057a <main+512>
116     return 3;
4040047c:  auipc a5,0x3fc01
40400480:  addi  a5,a5,-992 # 0x8000109c <cpu_intr>
40400484:  lw    a5,0(a5)
40400486:  mv    a0,a5
40400488:  jal  ra,0x40402132 <metal_interrupt_init>
119
4040048c:  jal  0x40400180 <display_banner>
122
4040048e:  auipc a5,0x3fc01
40400492:  addi  a5,a5,-1006 # 0x800010a0 <cpu0>
40400496:  lw    a5,0(a5)
40400498:  mv    a0,a5
4040049a:  jal  ra,0x40402092 <metal_cpu_timer_interrupt_controll
4040049e:  mv    a4,a0
404004a0:  auipc a5,0x3fc01
404004a4:  addi  a5,a5,-1032 # 0x80001098 <tmr_intr>
404004a8:  sw    a4,0(a5)
123     // Setup Timer and its interrupt so we can toggle LEDs
404004aa:  auipc a5,0x3fc01
404004ae:  addi  a5,a5,-1042 # 0x80001098 <tmr_intr>
404004b2:  lw    a5,0(a5)
404004b4:  bnez  a5,0x404004c4 <main+330>
124     tmr_intr = metal_cpu_timer_interrupt_controller(cpu0);
404004b6:  auipc a0,0x4
404004ba:  addi  a0,a0,1430 # 0x40404a4c
404004be:  jal  0x4040060e <puts>
```



Using Freedom E SDK

- **Required**

- GNU Make and Git
- Download RISC-V GNU Embedded Toolchain
- Download RISC-V OpenOCD or Install Segger J-Link Software*
- Download and build QEMU from its git repo

- **Clone Freedom -E-SDK**

- `git clone --recursive https://github.com/sifive/freedom-e-sdk.git`
- `cd freedom-e-sdk`
 - `set RISCV_PATH=/home/tools-path/riscv64-unknown-elf-gcc-8.2.0-2019`

- **Building an Example**

`make PROGRAM=hello TARGET=qemu-sifive-e31 software`

- **Run Example**

`make simulate PROGRAM=hello TARGET=qemu-sifive-e31`

- coremark
- dhrystone
- empty
- example-itim
- example-pmp
- example-spi
- example-user-mode
- example-user-syscall

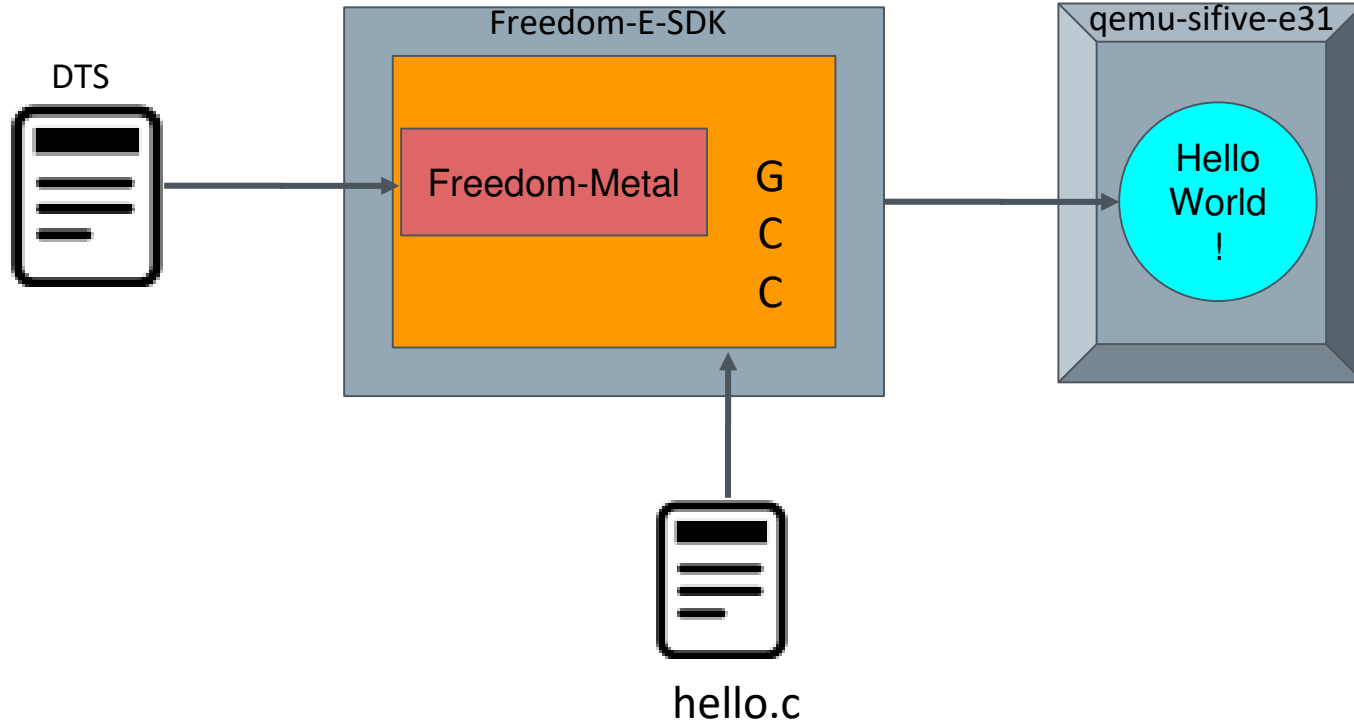
- ✓ hello

- local-interrupt
- multicore-hello
- return-fail
- return-pass
- sifive-welcome
- software-interrupt
- test-coreip
- timer-interrupt



How Freedom E SDK use Freedom Metal?

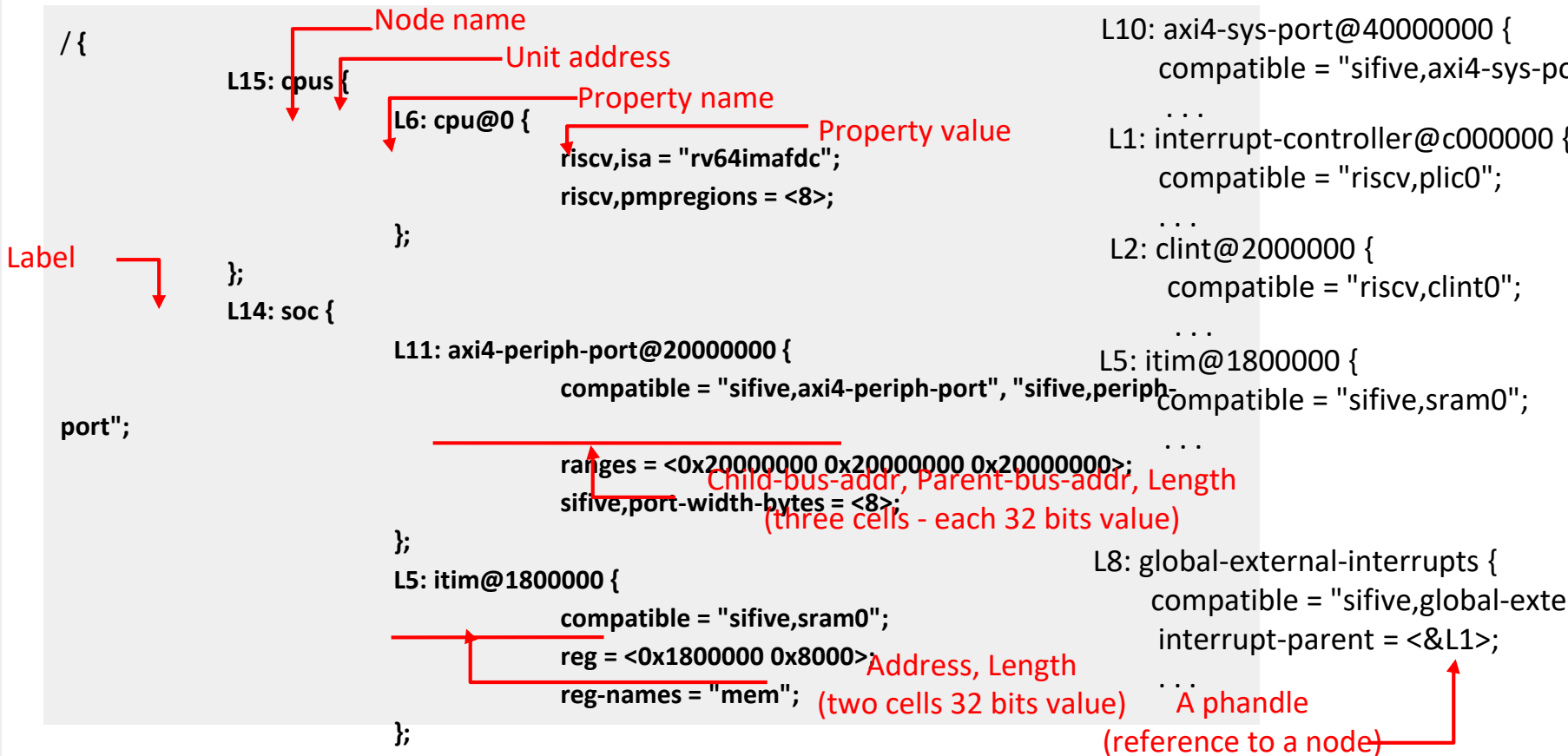
```
make simulate PROGRAM=hello TARGET=qemu-sifive-e31
```





DTS file - design.dts

Device Tree Source file (DTS) that describe the physical devices of a design or hardware.





DTS to BSP Files

```

L6: cpu@0 {
    compatible = "sifive,bullet0", "riscv";
    d-cache-size = <32768>;
    device_type = "cpu";
    riscv,isa = "rv64imafdc";
    riscv,pmpregions = <8>;
    L4: interrupt-controller {
        compatible = "riscv,cpu-
intc";
        Interrupt-controller;
L11: axi4-periph-port@20000000 {
    compatible = "sifive,axi4-periph-port";
    ranges = <0x20000000 0x20000000
0x20000000>;
    sifive,port-width-bytes = <8>;
L1: interrupt-controller@c000000 {
    compatible = "riscv,plic0";
    interrupts-extended = <&L4
1>;

    reg = <0xc000000 0x4000000>;
    riscv,max-priority = <7>;
    riscv,ndev = <127>;

```

settings.mk

```

RISCV_ARCH=rv64imafdc
RISCV_ABI=lp64d

COREIP_MEM_WIDTH=64

```

metal.default.lds

```

OUTPUT_ARCH("riscv")
MEMORY
{
    ram (wxa!ri) : ORIGIN = 0x80000000, LENGTH =
0x20000000
    flash (rxai!w) : ORIGIN = 0x20000000, LENGTH =
0x20000000

```

metal.h

```

#include <metal/memory.h>
#include <metal/drivers/riscv_cpu.h>
#include <metal/drivers/riscv_plic0.h>
#include <metal/pmp.h>
#define METAL_RISCV_PLIC0_C000000_RISCV_NDEV 128UL
static inline int __metal_driver_cpu_hartid(...)
static inline int __metal_driver_cpu_num_pmp_regions(...)
static inline int __metal_driver_sifive_plic0_num_interrupts(...)
{
    return METAL_RISCV_PLIC0_C000000_RISCV_NDEV;
}

```



Metal Library referencing BSP Files

Metal header files and settings.mk referenced by Freedom-Metal library during build for a design (TARGET).

See freedom-e-sdk/bsp/<target>/install/lib/release/libmetal.a

freedom-metal/src/pmp.c

```
static int _pmp_regions() {  
    ...  
    return __metal_driver_cpu_num_pmp_regions(current_cpu);  
}  
int metal_pmp_set_region(struct metal_pmp *pmp,  
                        unsigned int region, ...  
{  
    ...  
    if(region > _pmp_regions()) {
```

freedom-metal/src/driver/riscv_plic0.c

```
void __metal_driver_riscv_plic0_init (...)  
{  
    ...  
    for(int parent = 0; parent < __METAL_PLIC_NUM_PARENTS;  
    ...  
    num_interrupts =  
        __metal_driver_sifive_plic0_num_interrupts(...);  
    intc = __metal_driver_sifive_plic0_interrupt_parents(...);  
    line = __metal_driver_sifive_plic0_interrupt_lines(...);
```

metal.h & metal-platform.h

```
#include <metal/memory.h>  
#include <metal/drivers/riscv_cpu.h>  
#include <metal/drivers/riscv_plic0.h>  
#include <metal/pmp.h>  
  
#define __METAL_PLIC_NUM_PARENTS 1  
#define METAL_RISCV_PLIC0_C000000_RISCV_NDEV 128UL  
static inline int __metal_driver_cpu_hartid(...)  
static inline int __metal_driver_cpu_num_pmp_regions(...)  
static inline int __metal_driver_sifive_plic0_num_interrupts(...)  
{  
    return METAL_RISCV_PLIC0_C000000_RISCV_NDEV;  
}  
static inline struct metal_interrupt *  
__metal_driver_sifive_plic0_interrupt_parents(...)  
{  
    if (idx == 0) {  
        return (struct metal_interrupt  
*);  
    }  
    metal_dt_cpu_0_interrupt_controller_controller;
```

2



Allowing for Portable Software

```

int main (void)
{
    struct metal_led *led0_red;
    struct metal_cpu *cpu0;
    struct metal_interrupt *cpu_intr;
    int tmr_id;

    // Let's get start with getting LEDs and turn only RED ON
    led0_red = metal_led_get_rgb ("LD0", "red");
    metal_led_enable(led0_red);
    metal_led_on(led0_red);

    // Let's get the CPU and and its interrupt
    cpu0 = metal_cpu_get(0);
    cpu_intr = metal_cpu_interrupt_controller(cpu0);
    metal_interrupt_init(cpu_intr);

    // Setup Timer and its interrupt
    tmr_intr = metal_cpu_timer_interrupt_controller (cpu0);
    metal_interrupt_init(tmr_intr);

    tmr_id = metal_cpu_timer_get_interrupt_id (cpu0);
    rc = metal_interrupt_register_handler(tmr_intr, tmr_id,
    timer_isr, cpu0);
}

```

Only label!
No fix address

No knowledge of CLINT/CLIC
ID determine by target

design.dts

```

led@0red {
    compatible = "sifive,gpio";
    leds";
    label = "LD0red";
    gpios = <&L8 0>;
};

```

Machine Header - metal.h

```

static inline struct metal_gpio *
__metal_driver_sifive_gpio_led_gpio(struct metal_led *led)
{
    if ((uintptr_t)led == (uintptr_t)&__metal_dt_led_0red)
    {
        return (struct metal_gpio
        *)&__metal_dt_gpio_20002000;
    }
}

```

API Header Files

Thin wrapper that call/link to the selected "target" drivers (library)

libriscv mmachine coreip-e31-arty.a
metal.h

libriscv mmachine coreip-e21-arty.a
metal.h

riscv,cpu.c
sifive,clic0.c
sifive,gpio0.c
sifive,uart0.c

....
sifive,local-external-interrupt0.c
sifive,global-external-interrupt0.c



Linux





Fedora on RISC -V

Through the works of David Abdurachmanov

- ~20% of Fedora packages built for RISC-V
- Pre-build images available for Qemu and HiFive Unleashed
 - Build farm running, producing nightly images
- No signed RPM yet
- No images for Fedora Workstation/Server

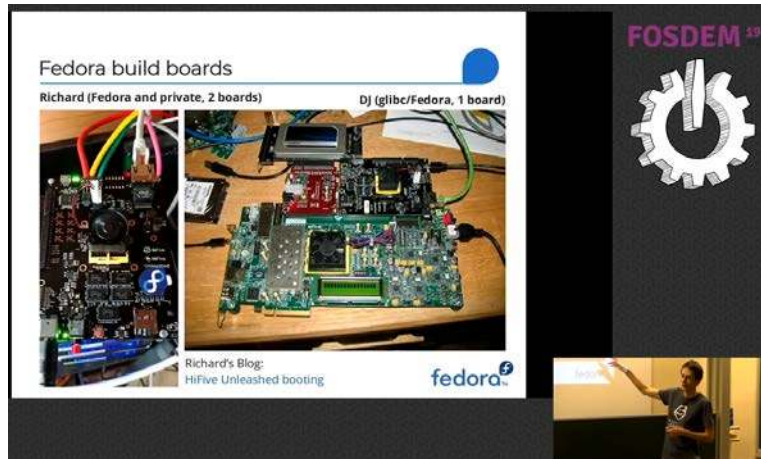


Want to know more/get involve, see

- <https://fedoraproject.org/wiki/Architectures/RISC-V>

Interest to try it out,

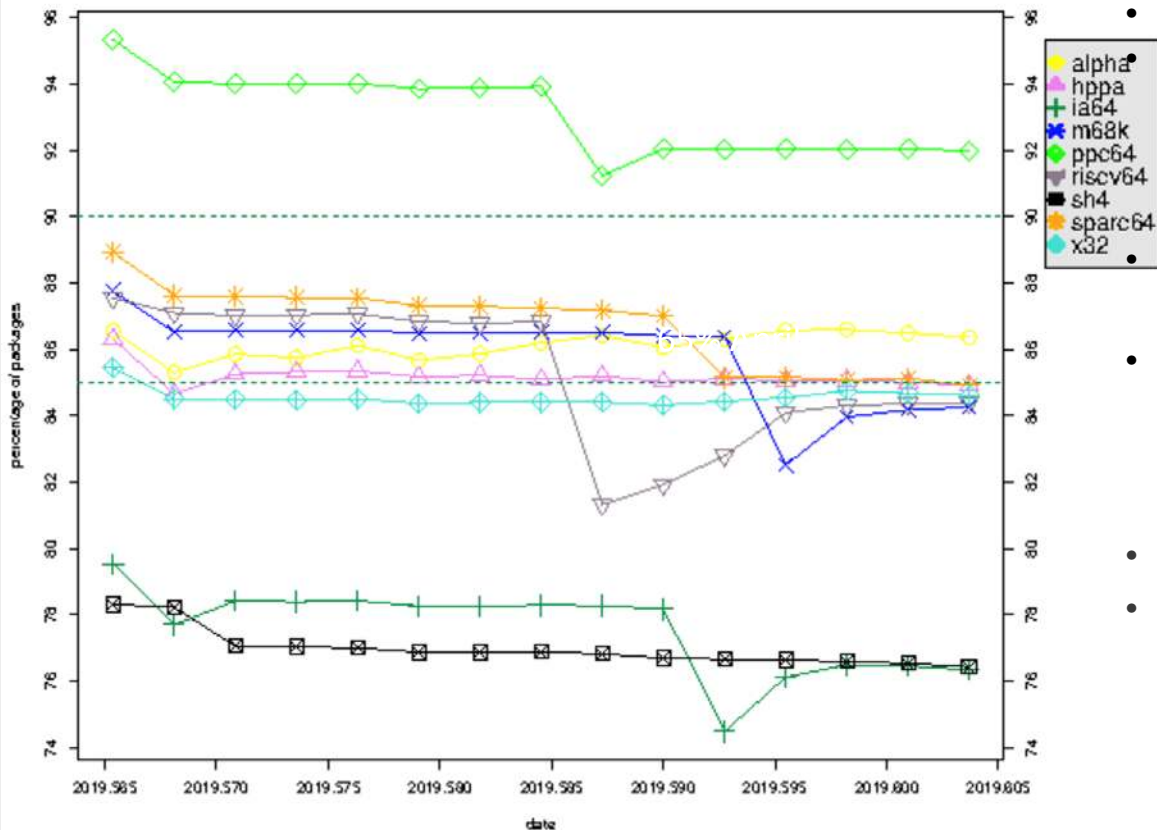
- Self Hosting images available
- <https://fedoraproject.org/wiki/Architectures/RISC-V/Installing>





Debian Distro Package Build Status

What percent is built for each architecture (past two weeks)



- **Open Software on Open Hardware RISC-V Debian Port Goals**

- Software-wise, this port will target the Linux kernel
- Hardware-wise, the port will target the 64-bit variant, little-endian

- **84% of all Debian packages**

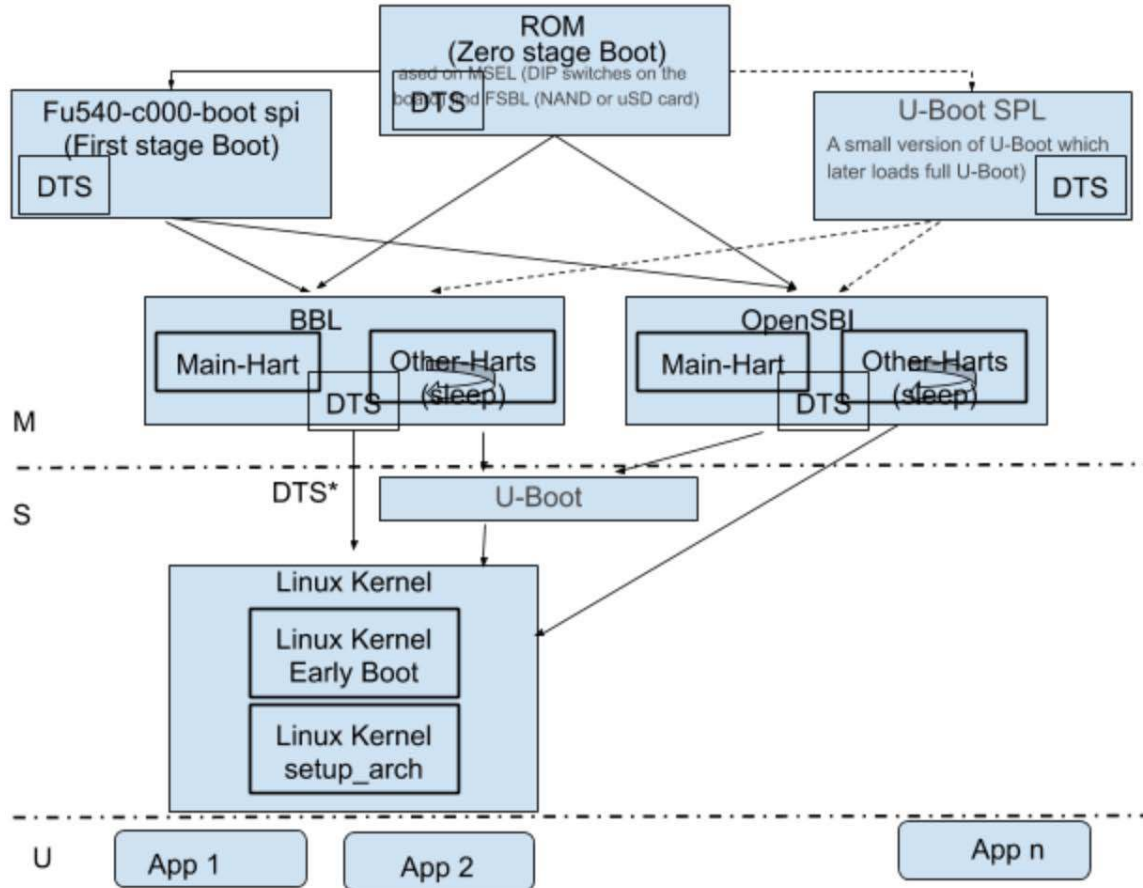
- **Debian Distro runs on HiFive Unleashed with SiFive Freedom U540 SoC**

- No Debian installer yet
- See Freedom-U-SDK for current, Buildroot based
<https://github.com/sifive/freedom-u-sdk>

- <https://wiki.debian.org/RISC-V>



Linux Boot Flow





Linux Boot on the FU540

Zero Stage Bootloader (ZSBL): Mask ROM

- Reset vector of all harts, baked into a mask ROM on die
- Looks at DIP switches and loads a partition from the SPI flash (or SD card in a recovery mode)

First Stage Bootloader (FSBL): SPI Flash

- Loaded from SPI flash into the L2 LIM
- Initializes the DRAM, Ethernet, and PRCI (clock controller)
- Loads a bootloader from the SD card

Berkeley Boot Loader (bbl - First stage bootloader)

- Runs in machine-mode
- Filters the device tree for Linux (disables the S51 hart)
- Contains trap handles for unimplemented instructions
 - rdttime, which traps to the CLINT
 - Floating-point on systems without hardware FP
- Contains an SBI implementation, which says resident during Linux
 - Handles remote fences and IPIs via the CLINT
 - Helper functions for an early debug console
- Loads a flat binary Linux image



RISC-V Linux Early Boot

Linux boots expects the system to be in the following state

- a0 contains a unique per-hart id
- a1 contains a pointer to device tree, as a binary flattened device tree (DTB)
- Memory is identity mapped
- The kernel's ELF image has been loaded correctly
- Handle impedance mismatch between RISC-V spec and what Linux expects
- Perform "hart lottery," which is a very short AMO-based sequence that picks the first hart to boot, while the rest spin, until they can proceed

Proceed with a fairly standard Linux early boot process:

- A linear mapping of all physical memory is set up, with PAGE_OFFSET as the offset
- Paging structures are initialized and then used (BBL boots with paging enabled)
- The C runtime is set up, which includes the stack and global pointers
- A spin-only trap vector is set up that catches any errors early in the boot process
- start_kernel is called to enter the standard Linux boot process



RISC-V Linux setup_arch

- Unconditionally enable the EARLY_PRINTK console via the SBI
- When kernel command line is parsed, and the early arch-specific options are dealt with, user can only control the amount of physical memory actually used by Linux
- The device tree's memory map is parsed in order to find the kernel image's memory block, which is marked as reserved. The rest of the device tree's memory is released to the kernel for allocation
- The memory management subsystem is initialized, only ZONE_NORMAL is support
- Any other hart in the system is woken up
- Processor's ISA is read from the device tree, which is used to fill out the HWCAP field in the ELF auxvec. This allows userspace programs to determine what the hardware they're executing on looks like. Homogeneous ISA is assume
- Return control back to the upstream kernel code



Getting Started

- Freedom Studio from SiFive.com
 - or Freedom E SDK & Metal
- RISC-V on Qemu
 - <https://risc-v-getting-started-guide.readthedocs.io/en/latest/index.html>
 - sw-dev@groups.riscv.org

To download and build QEMU from git:

- #riscv on freenode

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

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

