CEC Continuing Education Center

DesignNews

Embedded Software using RUST

DAY 2 : "Hello Rust!", using QEMU

Sponsored by

Digi-Key Electronics

Beningo Embedded Group

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

THE SPEAKER

Jacob Beningo

Visit 'Lecturer Profile'

Beningo Embedded Group - President

Focus: Embedded Software Consulting

An independent consultant who specializes in the design of real-time, microcontroller based embedded software.
He has published two books:

- Reusable Firmware Development
- MicroPython Projects
- Embedded Software Design

Writes a weekly blog for DesignNews.com focused on embedded system design techniques and challenges.

Visit www.beningo.com to learn more ...

Visit 'Lecturer Profile' in your console for more details.

Course Sessions

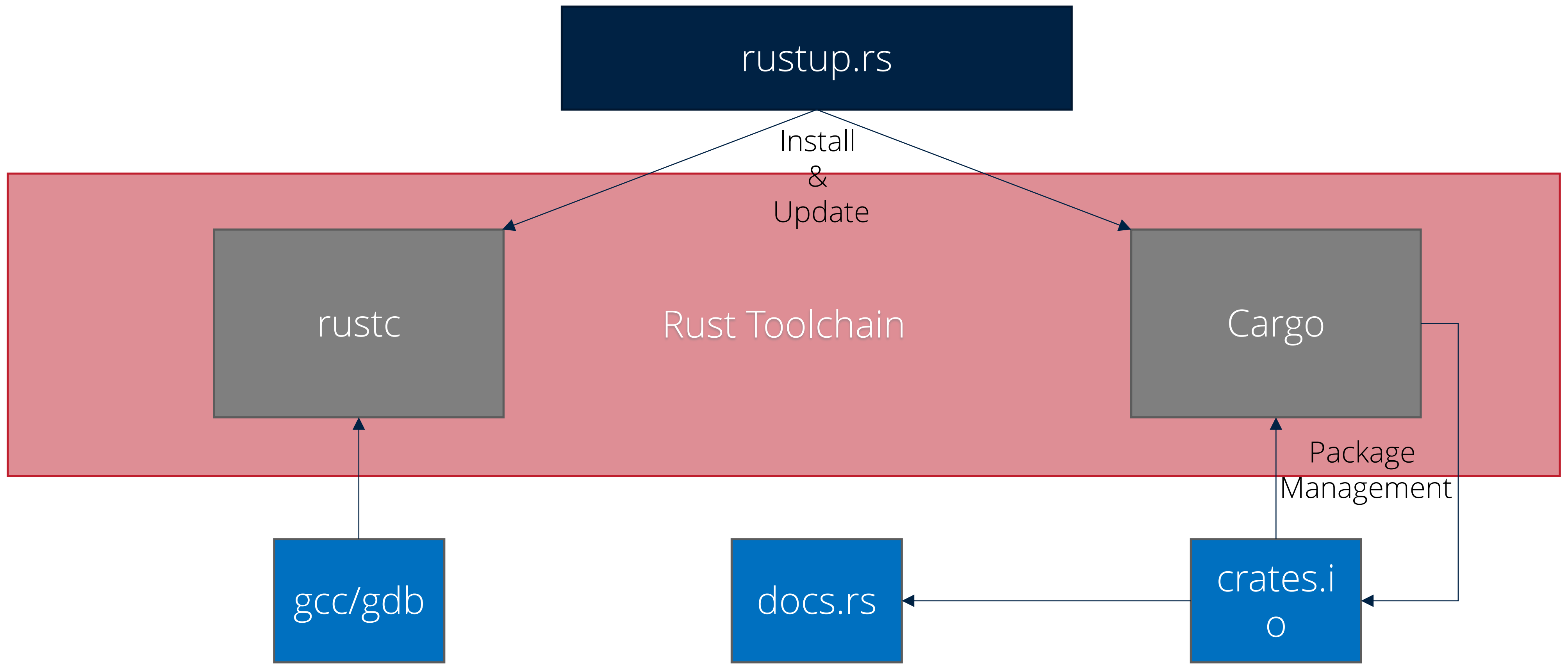
- Introduction to Rust for Embedded Systems
- **”Hello Rust!”, using QEMU**
- “Hello Rust!”, using the STM32F3
- Interfacing to Peripherals in Rust
- Becoming a Rust Expert

Sponsored By

1

Installing Rust

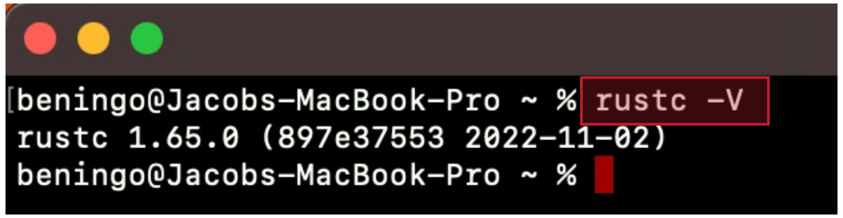
The Rust Toolchain



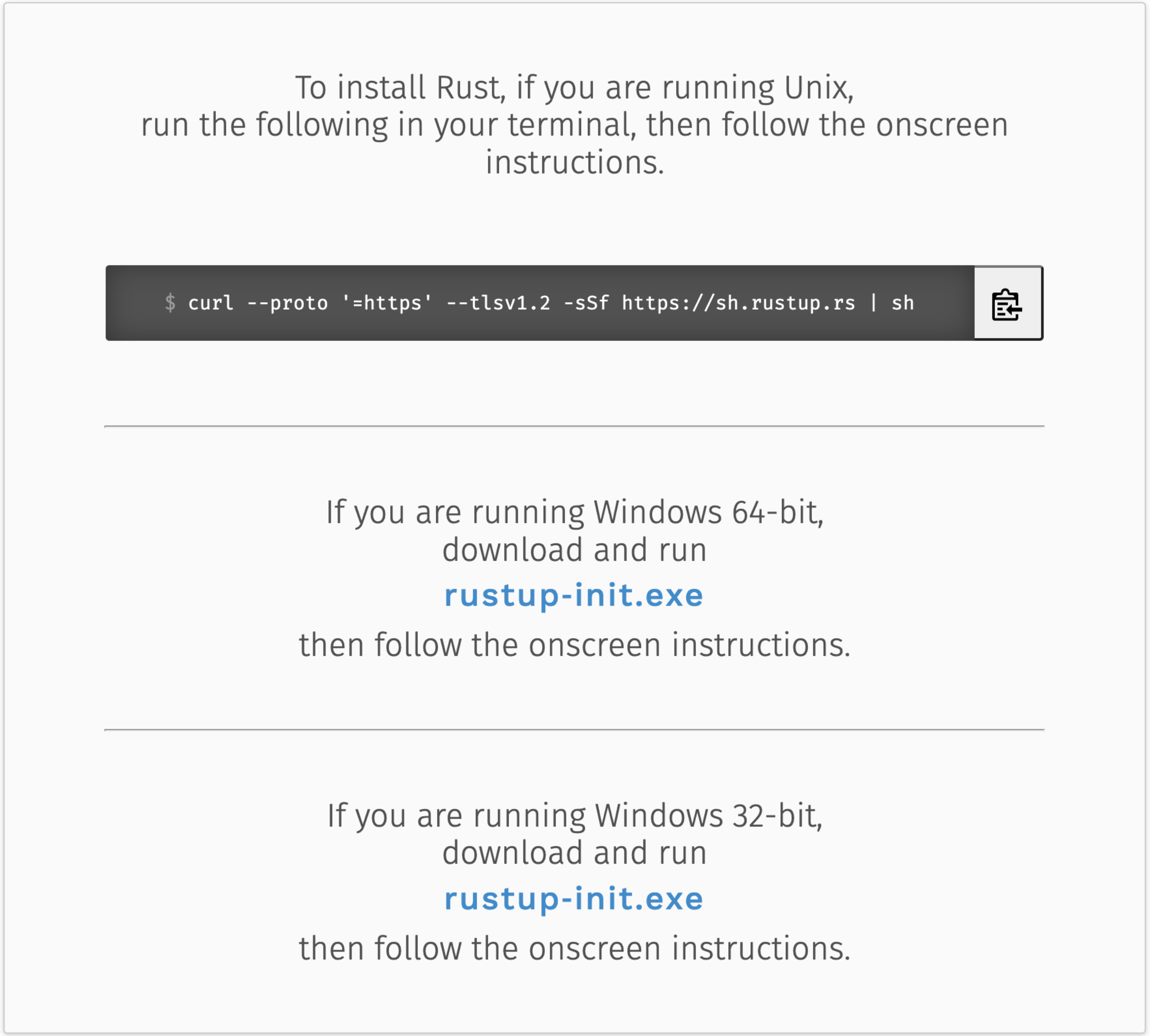
Rustup

Visit https://rustup.rs/

- Follow install instructions



```
[beningo@Jacobs-MacBook-Pro ~ % rustc -V
rustc 1.65.0 (897e37553 2022-11-02)
beningo@Jacobs-MacBook-Pro ~ %
```



To install Rust, if you are running Unix,
run the following in your terminal, then follow the onscreen instructions.

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

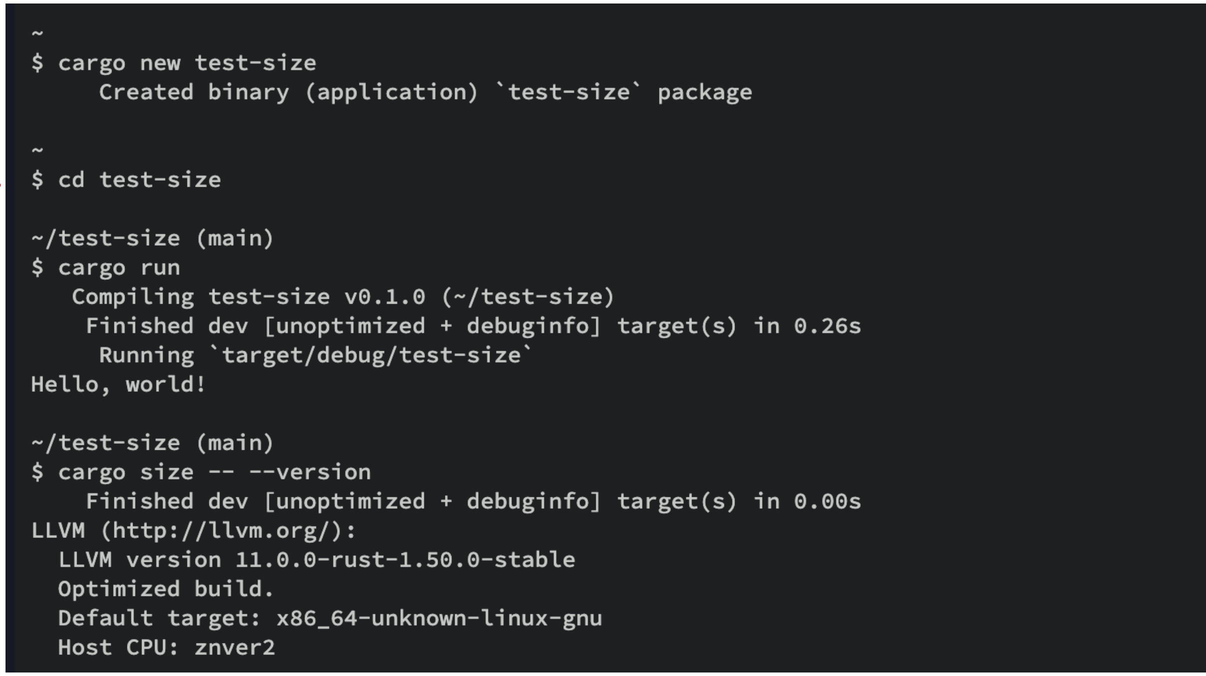
If you are running Windows 64-bit,
download and run
rustup-init.exe
then follow the onscreen instructions.

If you are running Windows 32-bit,
download and run
rustup-init.exe
then follow the onscreen instructions.

Additional Tools

Visit https://docs.rust-embedded.org/discovery/f3discovery/03-setup/index.html

- itmdump
- cargo-binutils
- arm-none-eabi-gdb
- OpenOCD



```
~
$ cargo new test-size
     Created binary (application) `test-size` package

~
$ cd test-size

~/test-size (main)
$ cargo run
   Compiling test-size v0.1.0 (~/test-size)
    Finished dev [unoptimized + debuginfo] target(s) in 0.26s
     Running `target/debug/test-size`
Hello, world!

~/test-size (main)
$ cargo size -- --version
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
LLVM (http://llvm.org/):
  LLVM version 11.0.0-rust-1.50.0-stable
  Optimized build.
  Default target: x86_64-unknown-linux-gnu
  Host CPU: znver2
```

Sponsored By

Will you be installing Rust to follow along?
- Yes
- No

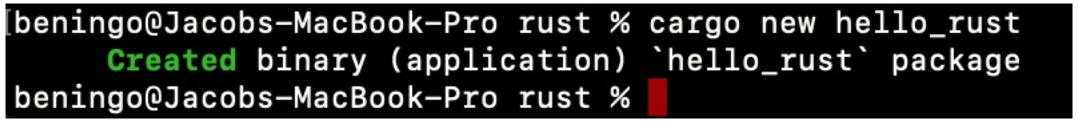
Sponsored By

2

Hello Rust!

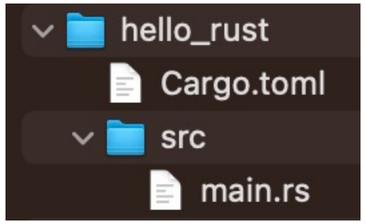
Creating a Hello Rust Application

Use cargo to create, manage, and build your projects:



```
beningo@Jacobs-MacBook-Pro rust % cargo new hello_rust
     Created binary (application) `hello_rust` package
beningo@Jacobs-MacBook-Pro rust %
```

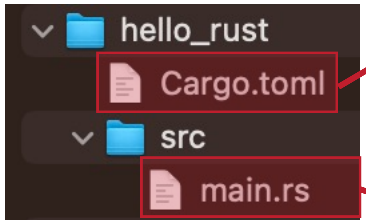
Created project:

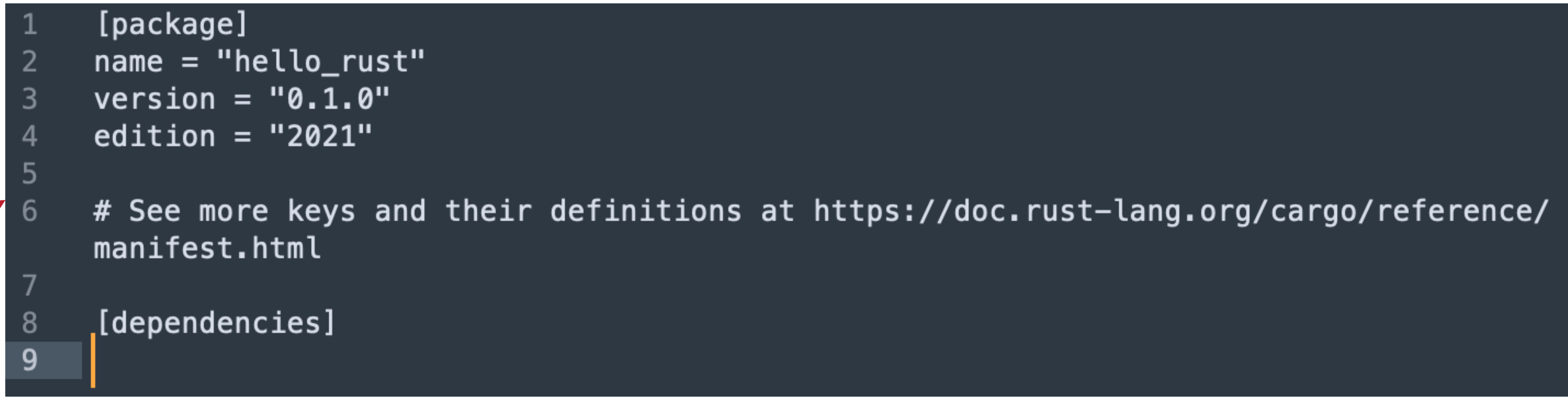


```
hello_rust
    Cargo.toml
    src
        main.rs
```

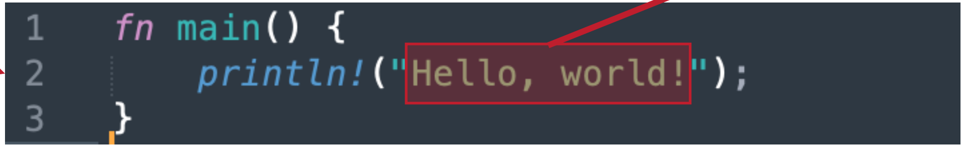
Creating a Hello Rust Application

Created project:





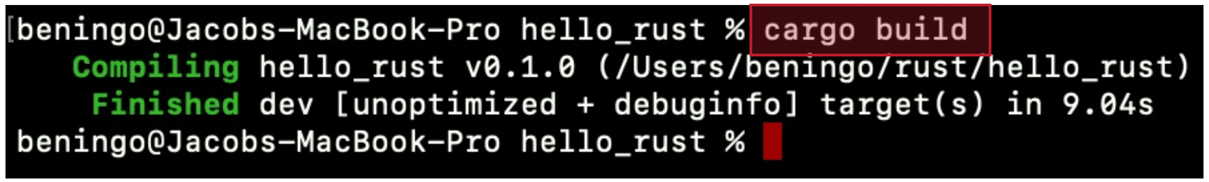
```toml
[package]
name = "hello_rust"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]

```



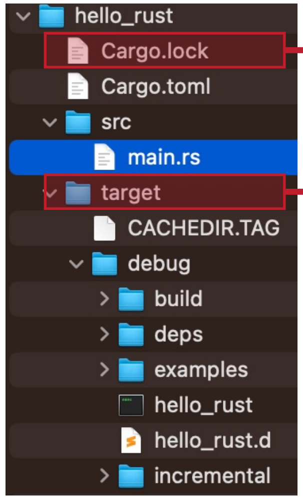
```rust
fn main() {
    println!("Hello, world!");
}
```

Change to: "Hello Rust!"

Building Hello Rust



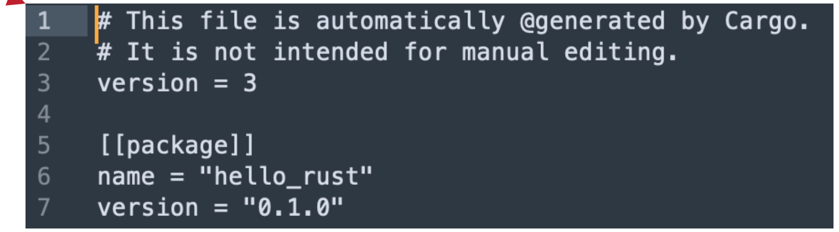
```
beningo@Jacobs-MacBook-Pro hello_rust % cargo build
   Compiling hello_rust v0.1.0 (/Users/beningo/rust/hello_rust)
    Finished dev [unoptimized + debuginfo] target(s) in 9.04s
beningo@Jacobs-MacBook-Pro hello_rust % 
```



```
hello_rust
  Cargo.lock
  Cargo.toml
  src
    main.rs
  target
    CACHEDIR.TAG
    debug
      build
      deps
      examples
      hello_rust
      hello_rust.d
      incremental
```

Contains exact information about our dependencies.
Automatically generated

Contains all the build information.

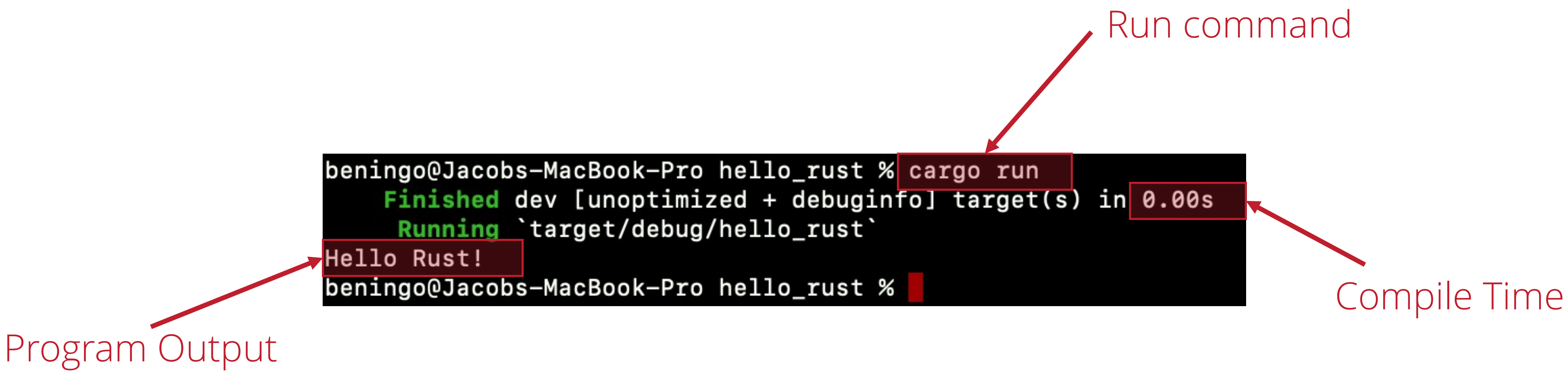


```
# This file is automatically @generated by Cargo.
# It is not intended for manual editing.
version = 3

[[package]]
name = "hello_rust"
version = "0.1.0"
```

Continuing Education Center

Sponsored By

Running Hello Rust



```
beningo@Jacobs-MacBook-Pro hello_rust % cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/hello_rust`
Hello Rust!
beningo@Jacobs-MacBook-Pro hello_rust %
```

What rust tool will you find yourself running the most when developing an application?

- rustup
- rustc
- cargo
- other

Sponsored By

3

Hello Rust! (QEMU Style)

Hello Rust! (QEMU Style)

Goal: Compile, Emulate, and debug a Rust application on an Arm Cortex-M3

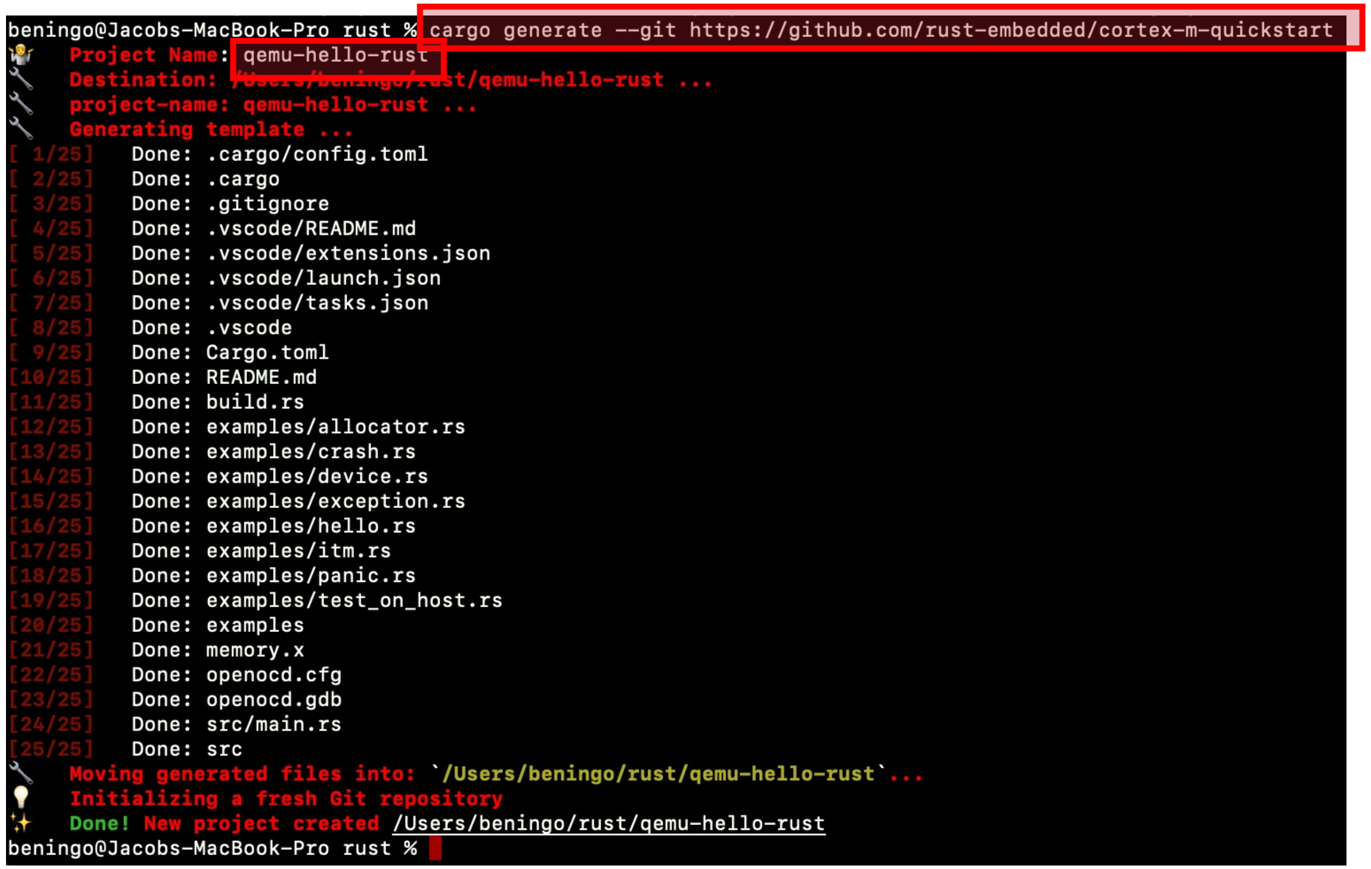
Make sure that you have cargo-generate installed:

```
cargo install cargo-generate
```

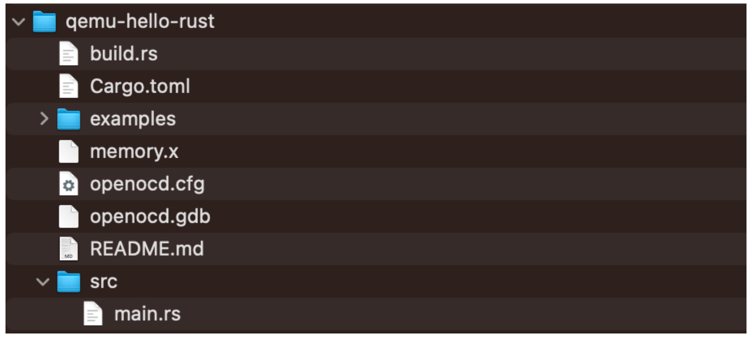
Create a new project:

```
cargo generate --git https://github.com/rust-embedded/cortex-m-quickstart
```

Hello Rust! (QEMU Style)

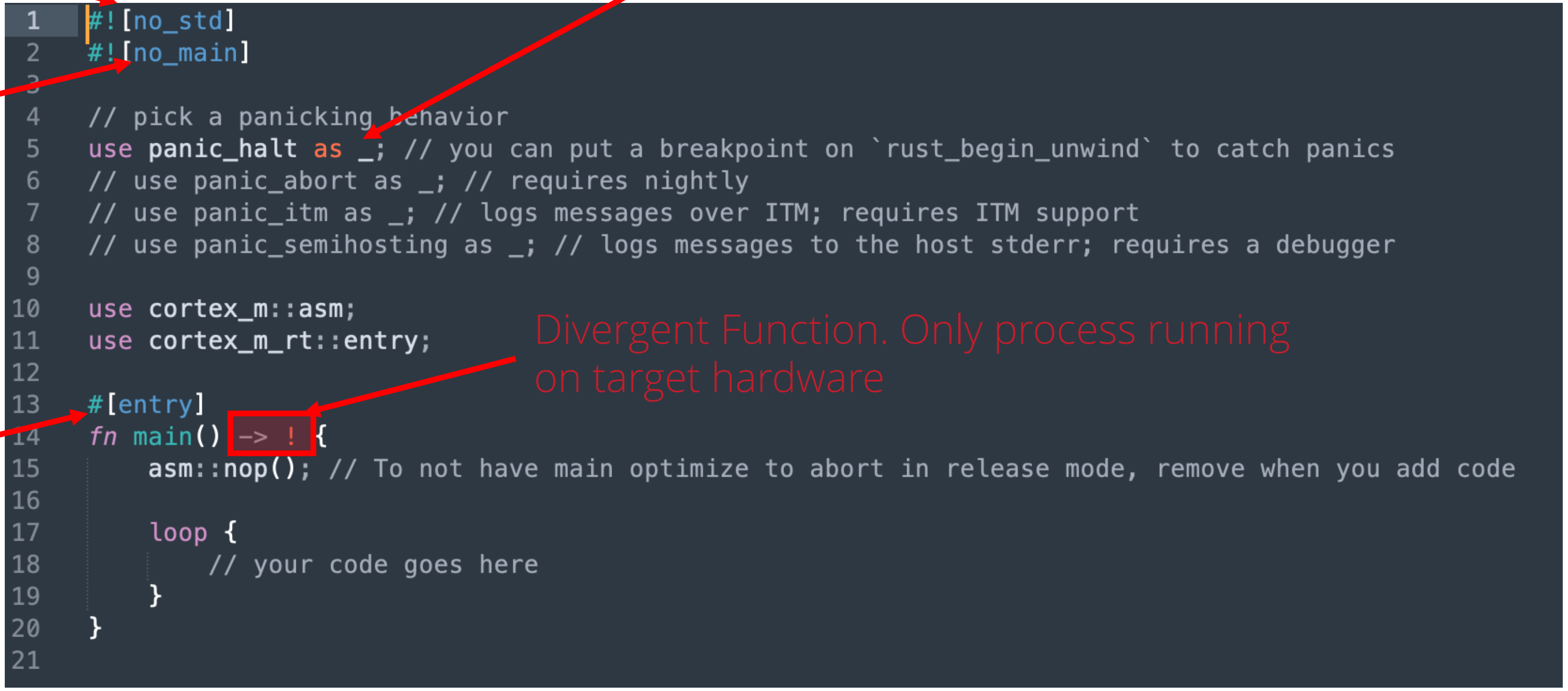


```
beningo@Jacobs-MacBook-Pro rust % cargo generate --git https://github.com/rust-embedded/cortex-m-quickstart
🤷   Project Name: qemu-hello-rust
🔧   Destination: /Users/beningo/rust/qemu-hello-rust ...
🔧   project-name: qemu-hello-rust ...
🔧   Generating template ...
[ 1/25]   Done: .cargo/config.toml
[ 2/25]   Done: .cargo
[ 3/25]   Done: .gitignore
[ 4/25]   Done: .vscode/README.md
[ 5/25]   Done: .vscode/extensions.json
[ 6/25]   Done: .vscode/launch.json
[ 7/25]   Done: .vscode/tasks.json
[ 8/25]   Done: .vscode
[ 9/25]   Done: Cargo.toml
[10/25]   Done: README.md
[11/25]   Done: build.rs
[12/25]   Done: examples/allocator.rs
[13/25]   Done: examples/crash.rs
[14/25]   Done: examples/device.rs
[15/25]   Done: examples/exception.rs
[16/25]   Done: examples/hello.rs
[17/25]   Done: examples/itm.rs
[18/25]   Done: examples/panic.rs
[19/25]   Done: examples/test_on_host.rs
[20/25]   Done: examples
[21/25]   Done: memory.x
[22/25]   Done: openocd.cfg
[23/25]   Done: openocd.gdb
[24/25]   Done: src/main.rs
[25/25]   Done: src
🔧   Moving generated files into: `/Users/beningo/rust/qemu-hello-rust`...
💡   Initializing a fresh Git repository
✨   Done! New project created /Users/beningo/rust/qemu-hello-rust
beningo@Jacobs-MacBook-Pro rust %
```



```
qemu-hello-rust
  build.rs
  Cargo.toml
  examples
  memory.x
  openocd.cfg
  openocd.gdb
  README.md
  src
    main.rs
```

Hello Rust! (QEMU Style)



```
#![no_std]
#![no_main]

// pick a panicking behavior
use panic_halt as _; // you can put a breakpoint on `rust_begin_unwind` to catch panics
// use panic_abort as _; // requires nightly
// use panic_itm as _; // logs messages over ITM; requires ITM support
// use panic_semihosting as _; // logs messages to the host stderr; requires a debugger

use cortex_m::asm;
use cortex_m_rt::entry;

#[entry]
fn main() -> ! {
    asm::nop(); // To not have main optimize to abort in release mode, remove when you add code

    loop {
        // your code goes here
    }
}
```

Don't link to the standard crate!

Don't want to link to nightly

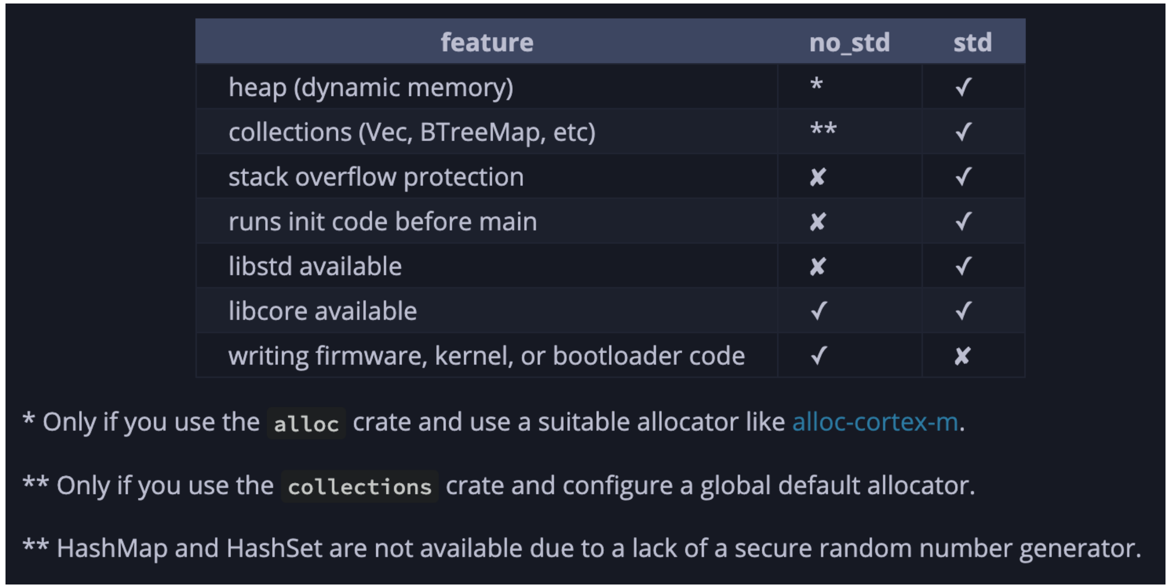
Define how panics are handled

Divergent Function. Only process running on target hardware

Define the entry function into application

Hello Rust! (QEMU Style)

More about #![no_std]



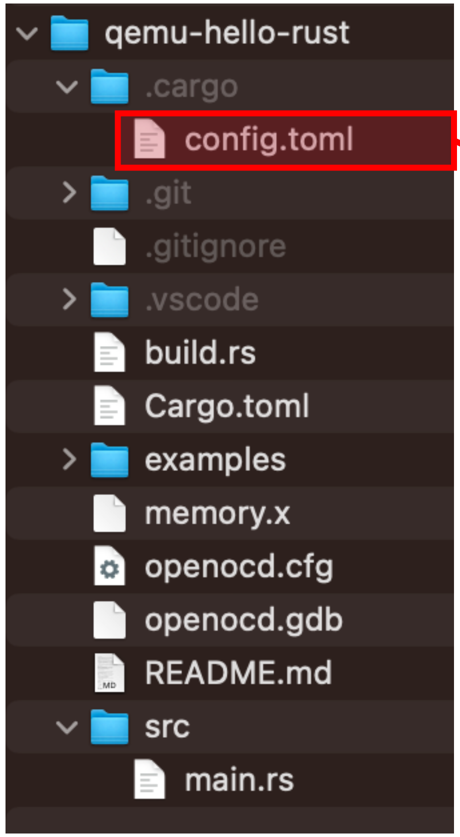
feature	no_std	std
heap (dynamic memory)	*	✓
collections (Vec, BTreeMap, etc)	**	✓
stack overflow protection	✘	✓
runs init code before main	✘	✓
libstd available	✘	✓
libcore available	✓	✓
writing firmware, kernel, or bootloader code	✓	✘

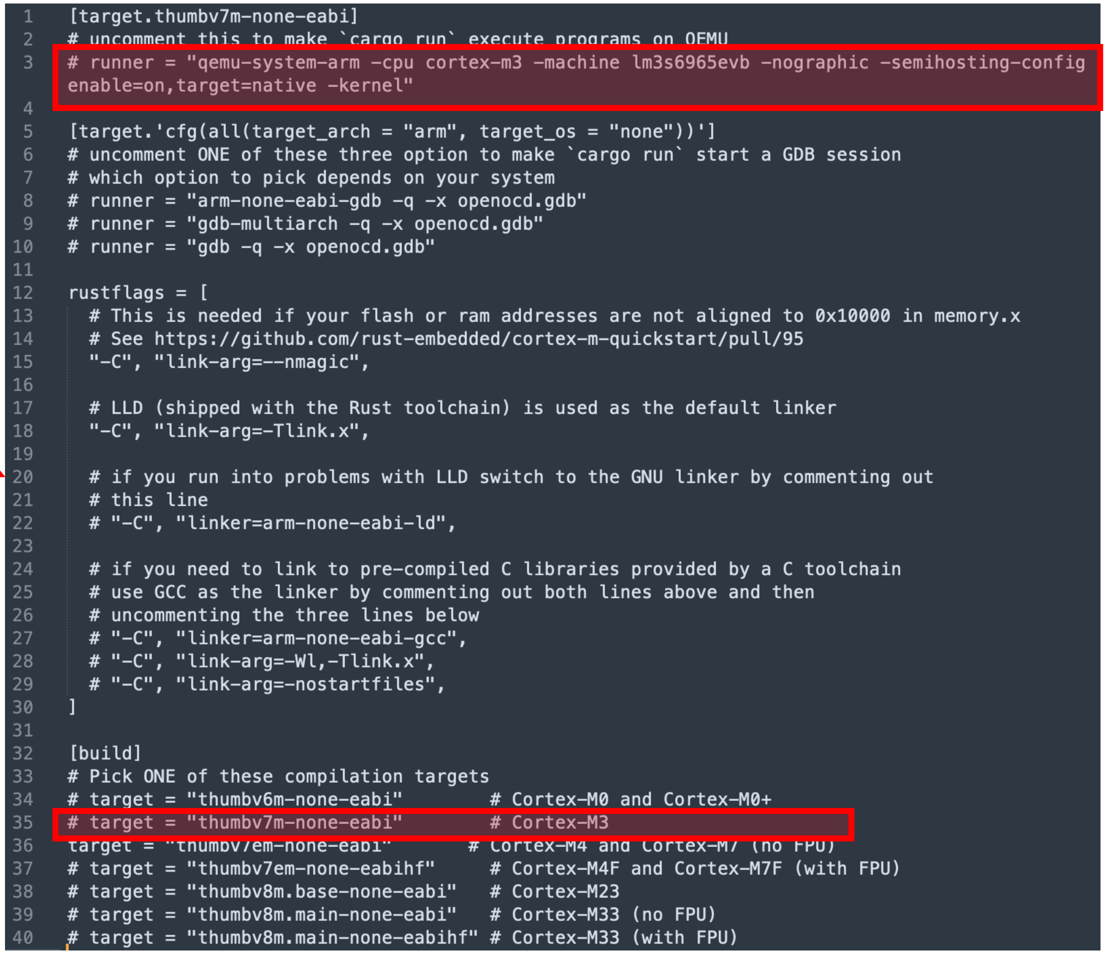
* Only if you use the `alloc` crate and use a suitable allocator like alloc-cortex-m.

** Only if you use the `collections` crate and configure a global default allocator.

** HashMap and HashSet are not available due to a lack of a secure random number generator.

Hello Rust! (QEMU Style)



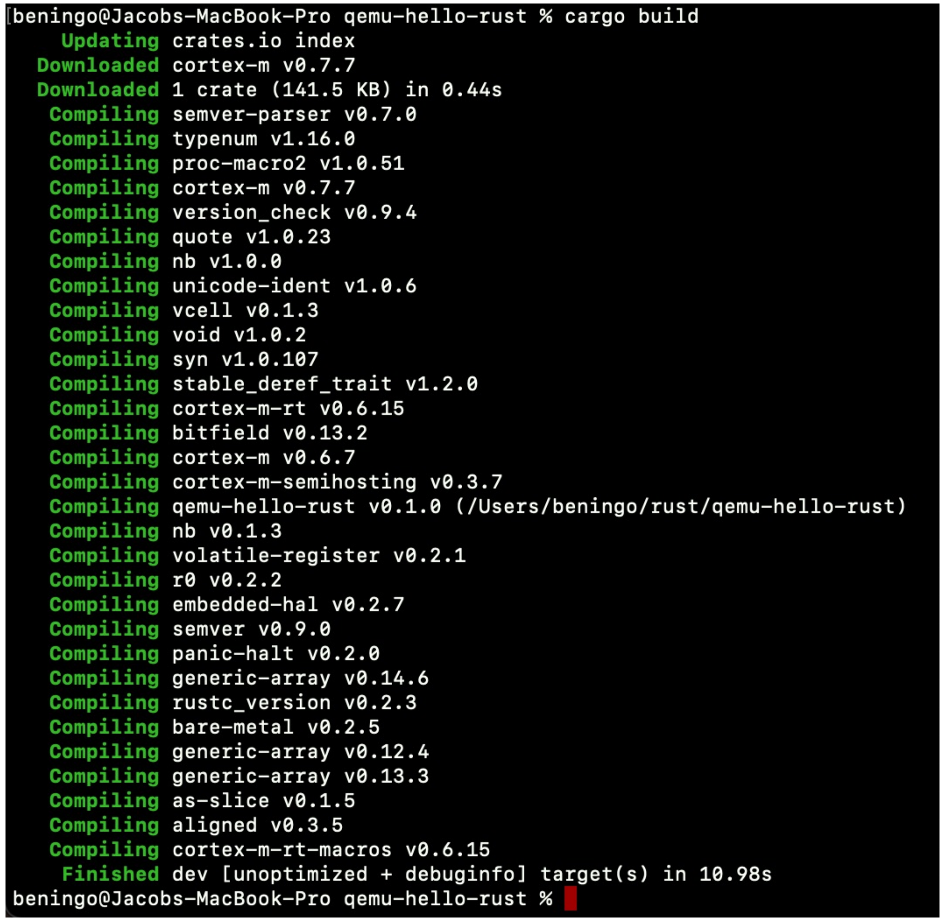


```
[target.thumbv7m-none-eabi]
# uncomment this to make `cargo run` execute programs on QEMU
# runner = "qemu-system-arm -cpu cortex-m3 -machine lm3s6965evb -nographic -semihosting-config
enable=on,target=native -kernel"

[target.'cfg(all(target_arch = "arm", target_os = "none"))']
# uncomment ONE of these three option to make `cargo run` start a GDB session
# which option to pick depends on your system
# runner = "arm-none-eabi-gdb -q -x openocd.gdb"
# runner = "gdb-multiarch -q -x openocd.gdb"
# runner = "gdb -q -x openocd.gdb"

rustflags = [
  # This is needed if your flash or ram addresses are not aligned to 0x10000 in memory.x
  # See https://github.com/rust-embedded/cortex-m-quickstart/pull/95
  "-C", "link-arg=--nmagic",

  # LLD (shipped with the Rust toolchain) is used as the default linker
  "-C", "link-arg=-Tlink.x",

  # if you run into problems with LLD switch to the GNU linker by commenting out
  # this line
  # "-C", "linker=arm-none-eabi-ld",

  # if you need to link to pre-compiled C libraries provided by a C toolchain
  # use GCC as the linker by commenting out both lines above and then
  # uncommenting the three lines below
  # "-C", "linker=arm-none-eabi-gcc",
  # "-C", "link-arg=-Wl,-Tlink.x",
  # "-C", "link-arg=-nostartfiles",
]

[build]
# Pick ONE of these compilation targets
# target = "thumbv6m-none-eabi"        # Cortex-M0 and Cortex-M0+
# target = "thumbv7m-none-eabi"        # Cortex-M3
target = "thumbv7em-none-eabi"       # Cortex-M4 and Cortex-M7 (no FPU)
# target = "thumbv7em-none-eabihf"     # Cortex-M4F and Cortex-M7F (with FPU)
# target = "thumbv8m.base-none-eabi"   # Cortex-M23
# target = "thumbv8m.main-none-eabi"   # Cortex-M33 (no FPU)
# target = "thumbv8m.main-none-eabihf" # Cortex-M33 (with FPU)
```

Hello Rust! (QEMU Style)

Compile the application for a Cortex-M4
- rustup target add thumbv7m-none-eabi
- cargo build

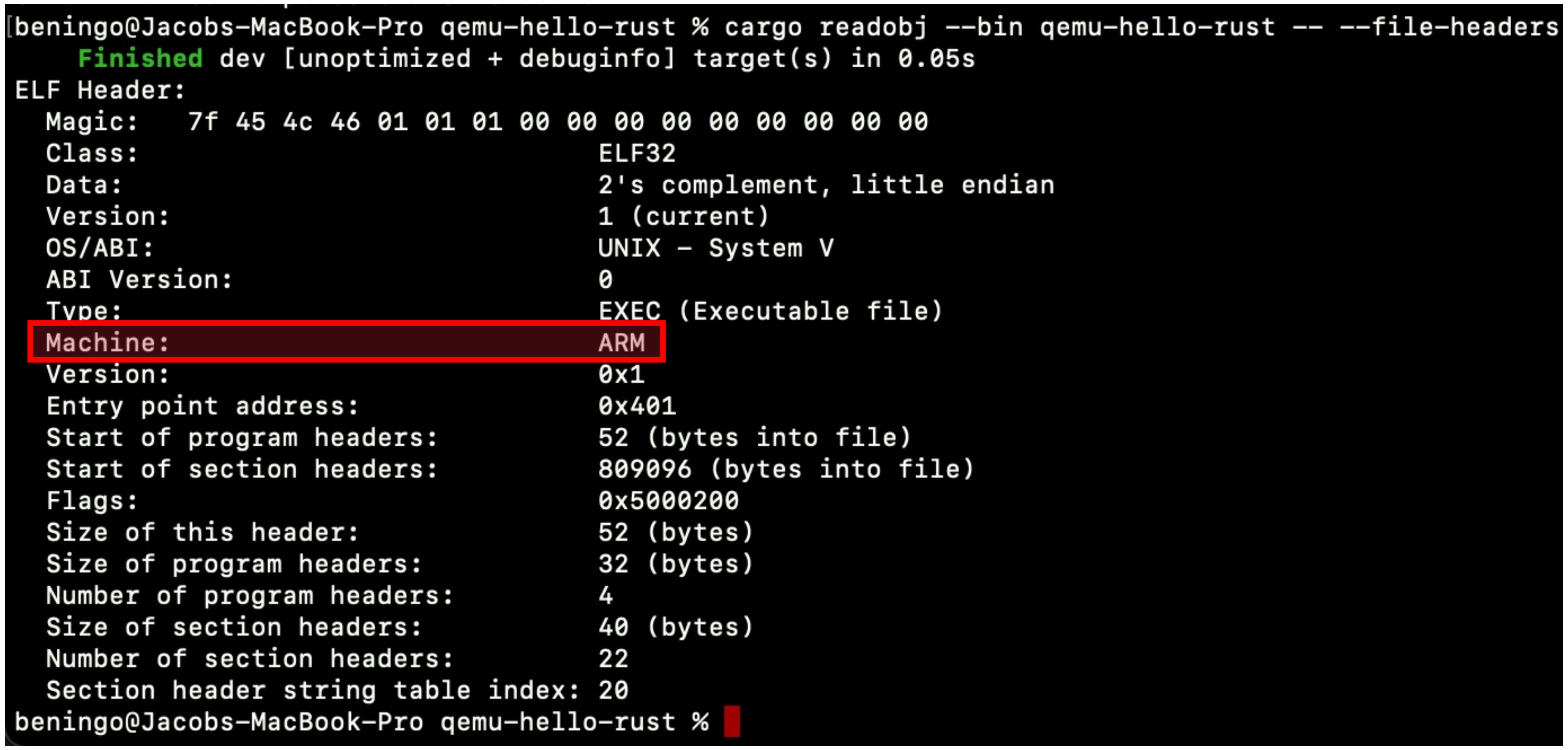
Use the following to verify the elf file is arm
- cargo readobj --bin qemu-hello-rust -- --file-headers



```
[beningo@Jacobs-MacBook-Pro qemu-hello-rust % cargo build
    Updating crates.io index
  Downloaded cortex-m v0.7.7
  Downloaded 1 crate (141.5 KB) in 0.44s
   Compiling semver-parser v0.7.0
   Compiling typenum v1.16.0
   Compiling proc-macro2 v1.0.51
   Compiling cortex-m v0.7.7
   Compiling version_check v0.9.4
   Compiling quote v1.0.23
   Compiling nb v1.0.0
   Compiling unicode-ident v1.0.6
   Compiling vcell v0.1.3
   Compiling void v1.0.2
   Compiling syn v1.0.107
   Compiling stable_deref_trait v1.2.0
   Compiling cortex-m-rt v0.6.15
   Compiling bitfield v0.13.2
   Compiling cortex-m v0.6.7
   Compiling cortex-m-semihosting v0.3.7
   Compiling qemu-hello-rust v0.1.0 (/Users/beningo/rust/qemu-hello-rust)
   Compiling nb v0.1.3
   Compiling volatile-register v0.2.1
   Compiling r0 v0.2.2
   Compiling embedded-hal v0.2.7
   Compiling semver v0.9.0
   Compiling panic-halt v0.2.0
   Compiling generic-array v0.14.6
   Compiling rustc_version v0.2.3
   Compiling bare-metal v0.2.5
   Compiling generic-array v0.12.4
   Compiling generic-array v0.13.3
   Compiling as-slice v0.1.5
   Compiling aligned v0.3.5
   Compiling cortex-m-rt-macros v0.6.15
    Finished dev [unoptimized + debuginfo] target(s) in 10.98s
beningo@Jacobs-MacBook-Pro qemu-hello-rust %
```

Sponsored By

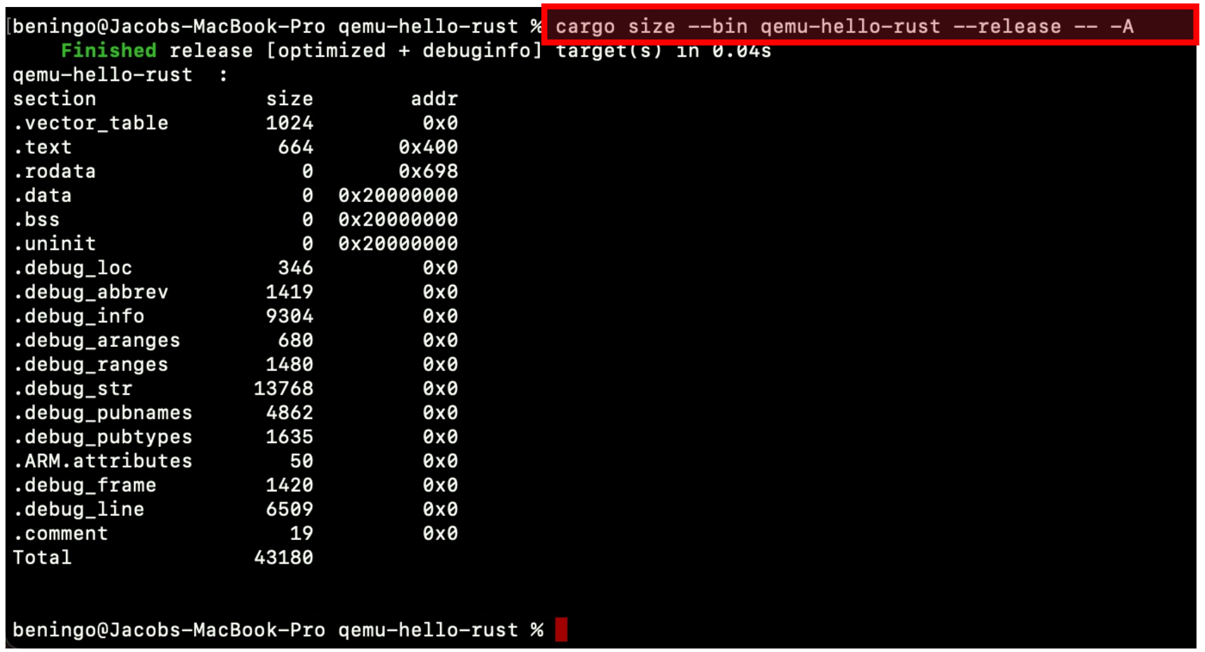
Hello Rust! (QEMU Style)



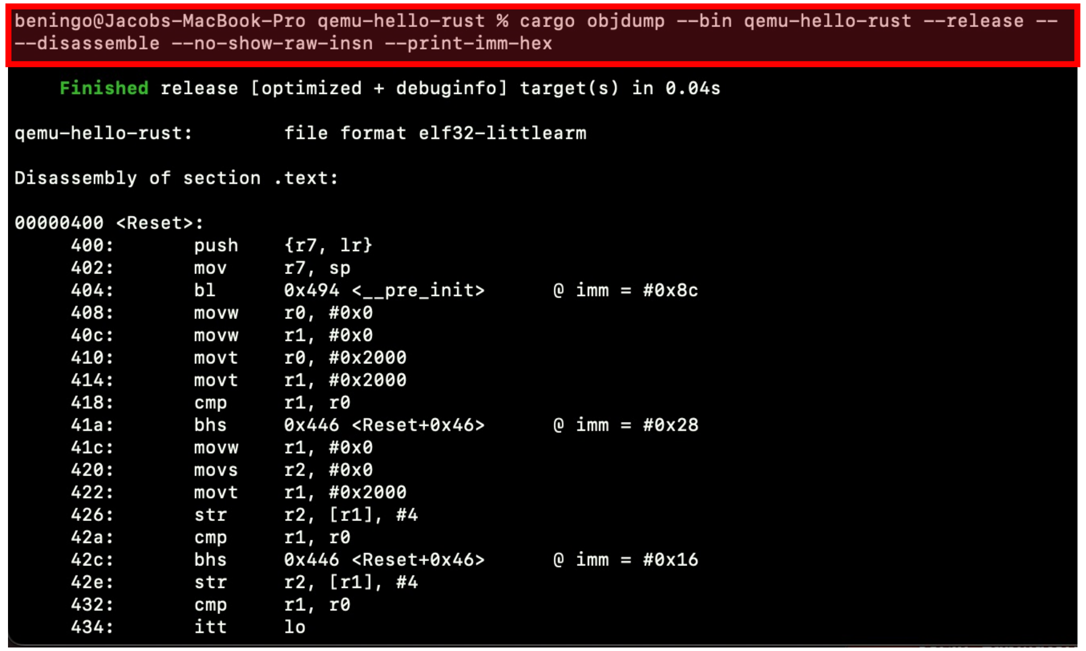
```
[beningo@Jacobs-MacBook-Pro qemu-hello-rust % cargo readobj --bin qemu-hello-rust -- --file-headers
    Finished dev [unoptimized + debuginfo] target(s) in 0.05s
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x401
  Start of program headers:          52 (bytes into file)
  Start of section headers:          809096 (bytes into file)
  Flags:                             0x5000200
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         4
  Size of section headers:           40 (bytes)
  Number of section headers:         22
  Section header string table index: 20
beningo@Jacobs-MacBook-Pro qemu-hello-rust %
```

Sponsored By

Hello Rust! (QEMU Style)

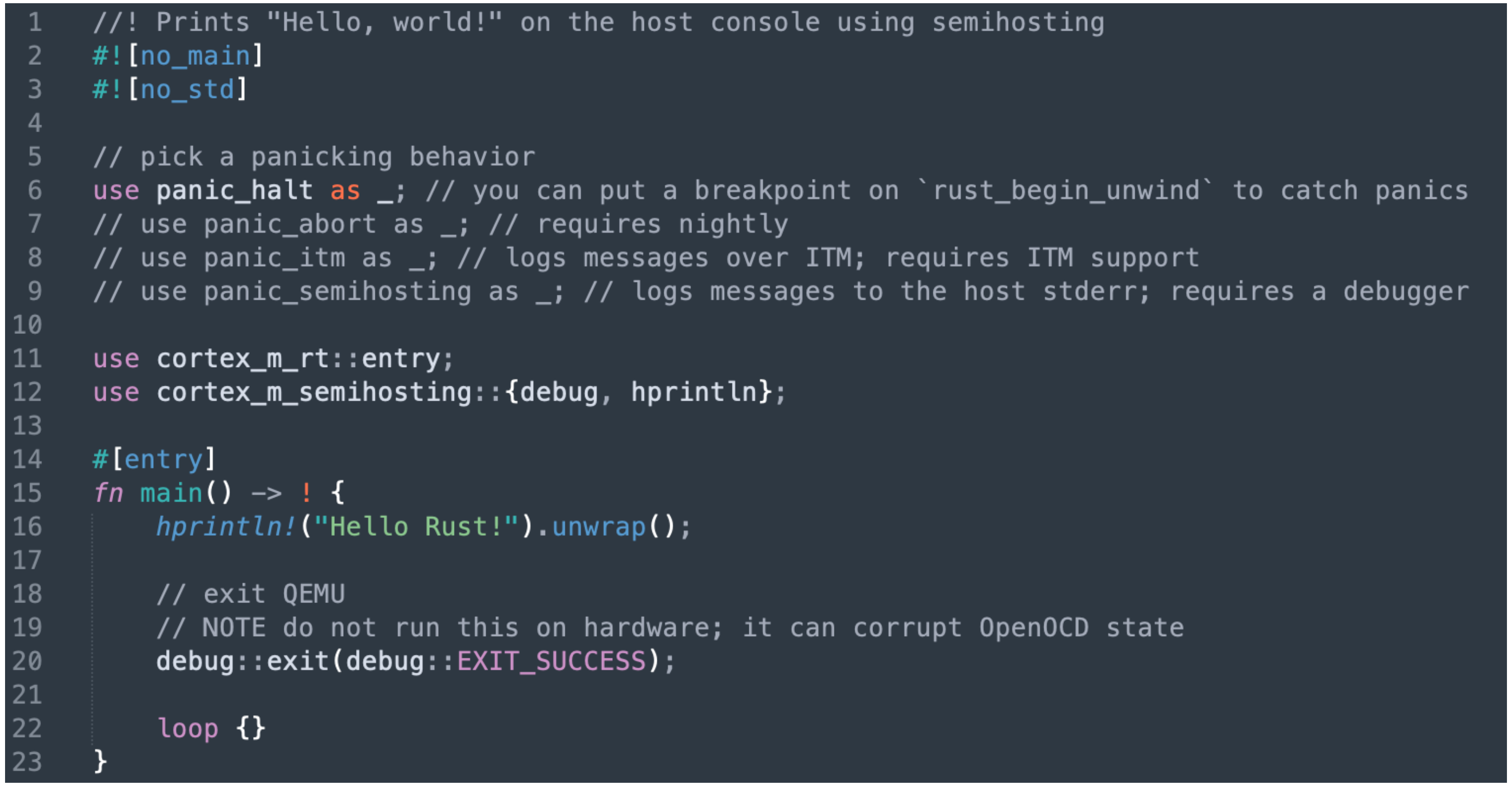


```
beningo@Jacobs-MacBook-Pro qemu-hello-rust % cargo size --bin qemu-hello-rust --release -- -A
    Finished release [optimized + debuginfo] target(s) in 0.04s
qemu-hello-rust  :
section                size        addr
.vector_table          1024         0x0
.text                   664       0x400
.rodata                   0       0x698
.data                     0  0x20000000
.bss                      0  0x20000000
.uninit                   0  0x20000000
.debug_loc              346         0x0
.debug_abbrev          1419         0x0
.debug_info            9304         0x0
.debug_aranges          680         0x0
.debug_ranges          1480         0x0
.debug_str            13768         0x0
.debug_pubnames        4862         0x0
.debug_pubtypes        1635         0x0
.ARM.attributes          50         0x0
.debug_frame           1420         0x0
.debug_line            6509         0x0
.comment                 19         0x0
Total                 43180


beningo@Jacobs-MacBook-Pro qemu-hello-rust %
```



```
beningo@Jacobs-MacBook-Pro qemu-hello-rust % cargo objdump --bin qemu-hello-rust --release --
--disassemble --no-show-raw-insn --print-imm-hex

    Finished release [optimized + debuginfo] target(s) in 0.04s

qemu-hello-rust:        file format elf32-littlearm

Disassembly of section .text:

00000400 <Reset>:
     400:       push    {r7, lr}
     402:       mov     r7, sp
     404:       bl      0x494 <__pre_init>      @ imm = #0x8c
     408:       movw    r0, #0x0
     40c:       movw    r1, #0x0
     410:       movt    r0, #0x2000
     414:       movt    r1, #0x2000
     418:       cmp     r1, r0
     41a:       bhs     0x446 <Reset+0x46>      @ imm = #0x28
     41c:       movw    r1, #0x0
     420:       movs    r2, #0x0
     422:       movt    r1, #0x2000
     426:       str     r2, [r1], #4
     42a:       cmp     r1, r0
     42c:       bhs     0x446 <Reset+0x46>      @ imm = #0x16
     42e:       str     r2, [r1], #4
     432:       cmp     r1, r0
     434:       itt     lo
```

Hello Rust! (QEMU Style)

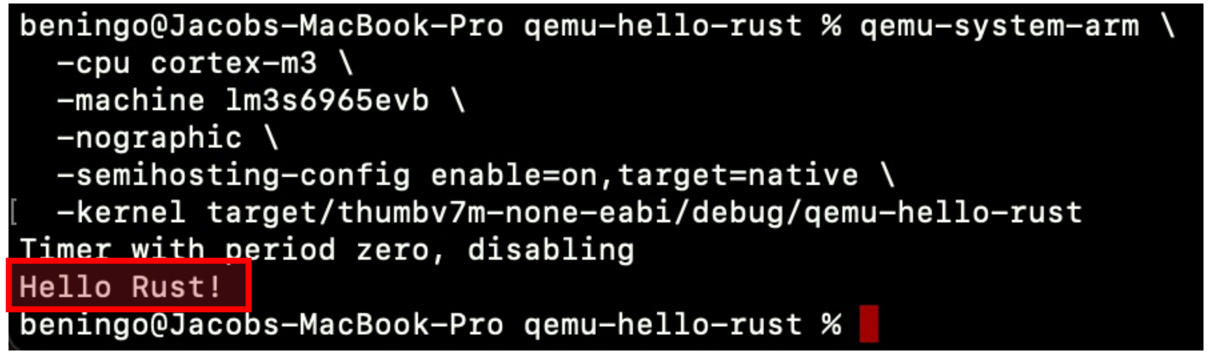


```rust
//! Prints "Hello, world!" on the host console using semihosting
#![no_main]
#![no_std]

// pick a panicking behavior
use panic_halt as _; // you can put a breakpoint on `rust_begin_unwind` to catch panics
// use panic_abort as _; // requires nightly
// use panic_itm as _; // logs messages over ITM; requires ITM support
// use panic_semihosting as _; // logs messages to the host stderr; requires a debugger

use cortex_m_rt::entry;
use cortex_m_semihosting::{debug, hprintln};

#[entry]
fn main() -> ! {
    hprintln!("Hello Rust!").unwrap();

    // exit QEMU
    // NOTE do not run this on hardware; it can corrupt OpenOCD state
    debug::exit(debug::EXIT_SUCCESS);

    loop {}
}
```

Hello Rust! (QEMU Style)

Run the application in QEMU:

```
qemu-system-arm \
  -cpu cortex-m3 \
  -machine lm3s6965evb \
  -nographic \
  -semihosting-config enable=on,target=native \
  -kernel target/thumbv7m-none-eabi/debug/examples/hello
```



```
beningo@Jacobs-MacBook-Pro qemu-hello-rust % qemu-system-arm \
  -cpu cortex-m3 \
  -machine lm3s6965evb \
  -nographic \
  -semihosting-config enable=on,target=native \
  -kernel target/thumbv7m-none-eabi/debug/qemu-hello-rust
Timer with period zero, disabling
Hello Rust!
beningo@Jacobs-MacBook-Pro qemu-hello-rust %
```

What method do you think you prefer?
- Running on the host without emulation
- Running on the host with emulation
- Running on the target hardware
- A combination of the above
- Other

Sponsored By

4

Going Further

Sponsored By

Rust Resources

- Rust Website
- Rust Book
- Rust for Embedded Book
- Learning Rust for Embedded Systems
- Rust By Example
- RTIC: Real-Time Interrupt Driven Concurrency

Thank you for attending

Please consider the resources below:

- www.beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm
  - Embedded Software Design
    - https://www.beningo.com/embedded-software-design/

From www.beningo.com under

- Blog > CEC – Embedded Software using Rust

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

BENINGO EMBEDDED GROUP

informa markets