



DesignNews

Writing Microcontroller Drivers in Rust

DAY 4 : Writing a Rust Application

Sponsored by

DigiKey



©2023 Beningo Embedded Group, LLC. All Rights Reserved.

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



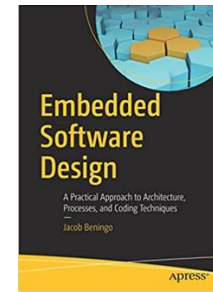
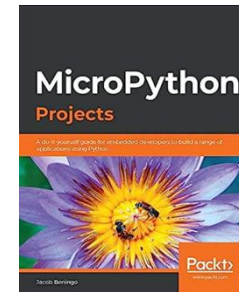
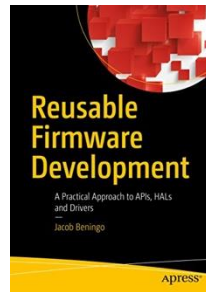
Jacob Beningo

Jacob@beningo.com

Beningo Embedded Group – CEO / Founder

Focus: Embedded Software Consulting and Training

Help teams deliver higher-quality embedded software faster. We specialize in creating and promoting embedded software excellence in businesses around the world.



Blogs for:

- DesignNews.com
- Embedded.com
- EmbeddedRelated.com
- MLRelated.com

Visit www.beningo.com to learn more

Embedded Rust Foundation

01

Embedded Rust Foundations

Options for Developing Embedded Rust Applications

- Frameworks
 - cortex-m-rt
 - embedded-hal
- Real-Time Interrupt-Driven Concurrency (RTIC)
- FreeRTOS
- Embassy
- RIOT OS



Embedded Rust Foundations

Frameworks: cortex_m_rt

Overview

- Provides a runtime for ARM Cortex-M microcontrollers, including startup code and exception handling.

Key Features

- Startup and reset handlers.
- Exception handling.
- Integration with other Cortex-M crates.

```

1  #![no_std]
2  #![no_main]
3
4  // pick a panicking behavior
5  use panic_halt as _, // you can put a breakpoint on `rust_begin_unwind` to catch panics
6  // use panic_abort as _; // requires nightly
7  // use panic_itm as _; // logs messages over ITM; requires ITM support
8  // use panic_semihosting as _; // logs messages to the host stderr; requires a debugger
9
10 use cortex_m::asm;
11 use cortex_m_rt::entry;
12
13 #[entry]
14 fn main() -> ! {
15     asm::nop(); // To not have main optimize to abort in release mode, remove when you add code
16
17     loop {
18         // your code goes here
19     }
20 }
21

```

Don't link to the standard crate!

Don't want to link to nightly

Define how panics are handled

Define the entry function into application

Divergent Function. Only process running on target hardware

Audience POLL Question

What does rt in cortex-m-rt stand for?

- a) Real-Time
- b) Run-Time
- c) RTOS
- d) None of the above

02

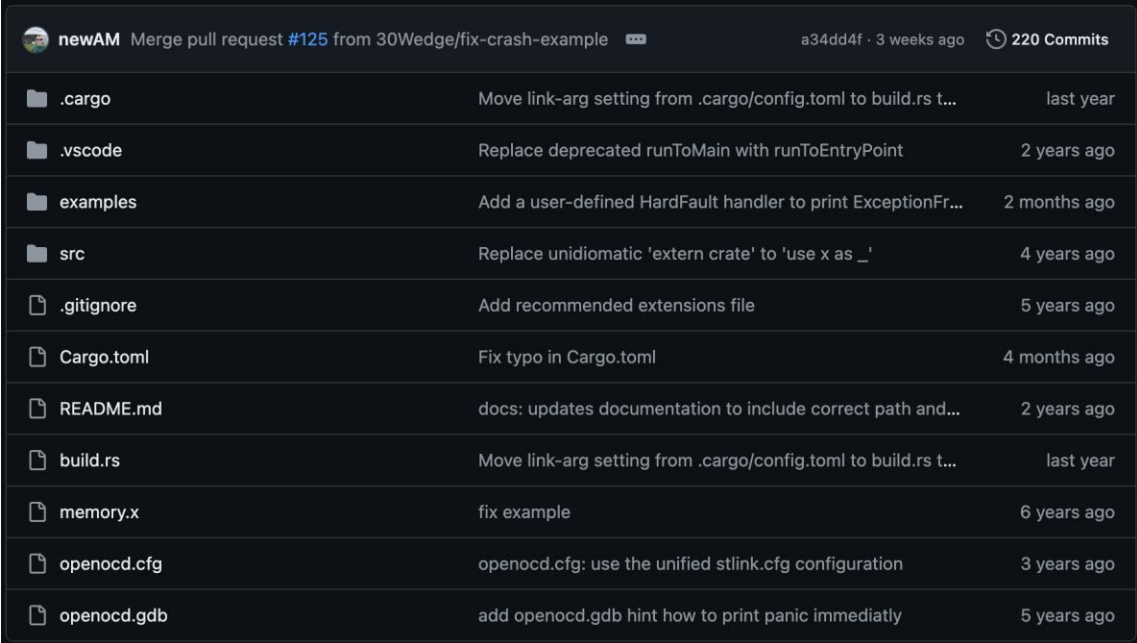
•• The Cortex-M
QuickStart Template

The Cortex-M QuickStart Template

Introduction

Template to develop bare metal applications for the Cortex-M

- Target configuration
- Linker
- Debug configurations
- Example projects
 - Simulation
 - ITM
 - Exceptions
 - Panics
 - Etc.



The screenshot shows a GitHub pull request interface. At the top, it says "newAM Merge pull request #125 from 30Wedge/fix-crash-example" with a commit hash "a34dd4f" and "3 weeks ago" and "220 Commits". Below this is a list of files and folders with their commit messages and dates:

File/Folder	Commit Message	Time Ago
.cargo	Move link-arg setting from .cargo/config.toml to build.rs t...	last year
.vscode	Replace deprecated runToMain with runToEntryPoint	2 years ago
examples	Add a user-defined HardFault handler to print ExceptionFr...	2 months ago
src	Replace unidiomatic 'extern crate' to 'use x as _'	4 years ago
.gitignore	Add recommended extensions file	5 years ago
Cargo.toml	Fix typo in Cargo.toml	4 months ago
README.md	docs: updates documentation to include correct path and...	2 years ago
build.rs	Move link-arg setting from .cargo/config.toml to build.rs t...	last year
memory.x	fix example	6 years ago
openocd.cfg	openocd.cfg: use the unified stlink.cfg configuration	3 years ago
openocd.gdb	add openocd.gdb hint how to print panic immediatly	5 years ago

The Cortex-M QuickStart Template

Creating a New Project

Embedded Applications are special:

- Linker files
- Target core support
- Semi-hosting / ITM
- Debug scripts
- SVD files
- Constrained memory environments
- Etc.

```
beningo@Jacobs-MacBook-Pro rust % cargo generate --git https://github.com/rust-embedded/cortex-m-quickstart
Project Name: stm32f3_hello
Renaming project called `stm32f3_hello` to `stm32f3-hello`...
Destination: /Users/beningo/rust/stm32f3-hello ...
project-name: stm32f3-hello ...
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/stm32f3-hello`...
Initializing a fresh Git repository
Done! New project created /Users/beningo/rust/stm32f3-hello
beningo@Jacobs-MacBook-Pro rust %
```

The Cortex-M QuickStart Template

std vs no_std

main.rs

```

1  #![no_std]
2  #![no_main]
3
4  // pick a panicking behavior
5  use panic_halt as _; // you can put a breakpoint on `rust_begin_unwind` to catch panics
6  // use panic_abort as _; // requires nightly
7  // use panic_itm as _; // logs messages over ITM; requires ITM support
8  // use panic_semihosting as _; // logs messages to the host stderr; requires a debugger
9
10 use cortex_m::asm;
11 use cortex_m_rt::entry;
12
13 #[entry]
14 fn main() -> ! {
15     asm::nop(); // To not have main optimize to abort in release mode, remove when you add code
16
17     loop {
18         // your code goes here
19     }
20 }
21

```

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	✓	✗

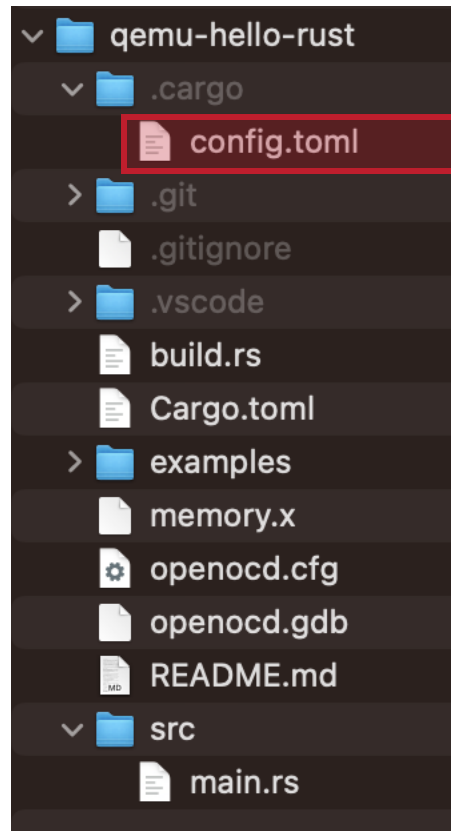
* Only if you use the `alloc` crate and use a suitable allocator like `alloc-cortex-m`.

** 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.

The Cortex-M QuickStart Template

Target Configuration



```
1 [target.thumbv7m-none-eabi]
2 # uncomment this to make `cargo run` execute programs on QEMU
3 # runner = "qemu-system-arm -cpu cortex-m3 -machine lm3s6965evb -nographic -semihosting-config
  enable=on,target=native -kernel"
4
5 [target.'cfg(all(target_arch = "arm", target_os = "none"))']
6 # uncomment ONE of these three option to make `cargo run` start a GDB session
7 # which option to pick depends on your system
8 # runner = "arm-none-eabi-gdb -q -x openocd.gdb"
9 # runner = "gdb-multiarch -q -x openocd.gdb"
10 # runner = "gdb -q -x openocd.gdb"
11
12 rustflags = [
13     # This is needed if your flash or ram addresses are not aligned to 0x10000 in memory.x
14     # See https://github.com/rust-embedded/cortex-m-quickstart/pull/95
15     "-C", "link-arg=--nmagic",
16
17     # LLD (shipped with the Rust toolchain) is used as the default linker
18     "-C", "link-arg=-Tlink.x",
19
20     # if you run into problems with LLD switch to the GNU linker by commenting out
21     # this line
22     # "-C", "linker=arm-none-eabi-ld",
23
24     # if you need to link to pre-compiled C libraries provided by a C toolchain
25     # use GCC as the linker by commenting out both lines above and then
26     # uncommenting the three lines below
27     # "-C", "linker=arm-none-eabi-gcc",
28     # "-C", "link-arg=-Wl,-Tlink.x",
29     # "-C", "link-arg=-nostartfiles",
30 ]
31
32 [build]
33 # Pick ONE of these compilation targets
34 # target = "thumbv6m-none-eabi" # Cortex-M0 and Cortex-M0+
35 # target = "thumbv7m-none-eabi" # Cortex-M3
36 target = "thumbv7em-none-eabi" # Cortex-M4 and Cortex-M7 (no FPU)
37 # target = "thumbv7em-none-eabihf" # Cortex-M4F and Cortex-M7F (with FPU)
38 # target = "thumbv8m.base-none-eabi" # Cortex-M23
39 # target = "thumbv8m.main-none-eabi" # Cortex-M33 (no FPU)
40 # target = "thumbv8m.main-none-eabihf" # Cortex-M33 (with FPU)
```

Audience POLL Question

#![no_std] has the following effects on a Rust application:

- Enables stack overflow protection
- Runs init code before main

a) false

b) true

Building the Template Project

03

Building the Template Project

Compiling an Application

Compile the application for a Cortex-M4

- `rustup target add thumbv7em-none-eabi`
- `cargo build`

Use the following to verify the elf file is arm

- `cargo readobj --bin blinky -- --file-headers`

```
root@20d054703a23:/home/app/blinkys# cargo build
  Compiling semver-parser v0.7.0
  Compiling proc-macro2 v1.0.86
  Compiling cortex-m v0.7.7
  Compiling unicode-ident v1.0.12
  Compiling nb v1.1.0
  Compiling syn v1.0.109
  Compiling cortex-m-rt v0.7.3
  Compiling void v1.0.2
  Compiling vcell v0.1.3
  Compiling bitfield v0.13.2
  Compiling critical-section v1.1.2
  Compiling cortex-m-semihosting v0.3.7
  Compiling stm32u575_pac v0.1.0 (/home/app/stm32u575_pac)
  Compiling blinky v0.1.0 (/home/app/blinkys)
  Compiling panic-halt v0.2.0
  Compiling nb v0.1.3
  Compiling semver v0.9.0
  Compiling volatile-register v0.2.2
  Compiling embedded-hal v0.2.7
  Compiling rustc_version v0.2.3
  Compiling bare-metal v0.2.5
  Compiling quote v1.0.36
  Compiling cortex-m-rt-macros v0.7.0
  Finished `dev` profile [unoptimized + debuginfo] target(s) in 45.23s
root@20d054703a23:/home/app/blinkys#
```

Building the Template Project

Reading an Object File

```
root@20d054703a23:/home/app/blinkys# cargo readobj --bin blinky -- --file-headers
  Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.37s
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 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:                   0x8000259
  Start of program headers:              52 (bytes into file)
  Start of section headers:              910444 (bytes into file)
  Flags:                                  0x5000400
  Size of this header:                    52 (bytes)
  Size of program headers:                32 (bytes)
  Number of program headers:              5
  Size of section headers:                40 (bytes)
  Number of section headers:              23
  Section header string table index:      21
root@20d054703a23:/home/app/blinkys#
```

Building the Template Project

Size Info and Objdump

```

root@20d054703a23:/home/app/blinky# cargo size --bin blinky --release -- -A
  Compiling cortex-m v0.7.7
  Compiling proc-macro2 v1.0.86
  Compiling syn v1.0.109
  Compiling cortex-m-rt v0.7.3
  Compiling bare-metal v0.2.5
  Compiling cortex-m-semihosting v0.3.7
  Compiling stm32u575_pac v0.1.0 (/home/app/stm32u575_pac)
  Compiling blinky v0.1.0 (/home/app/blinky)
  Compiling quote v1.0.36
  Compiling cortex-m-rt-macros v0.7.0
  Finished `release` profile [optimized + debuginfo] target(s) in 23.69s
blinky :
section          size      addr
.vector_table    600      0x8000000
.text            600      0x8000258
.rodata          16       0x80004b0
.data            0        0x20000000
.gnu.sgstubs     0        0x80004c0
.bss             12       0x20000000
.uninit          0        0x2000000c
.debug_loc       1104     0x0
.debug_abbrev    2210     0x0
.debug_info      26266    0x0
.debug_aranges   672      0x0
.debug_ranges    1656     0x0
.debug_str       21546    0x0
.comment         64       0x0
.ARM.attributes  58       0x0
.debug_frame     992      0x0
.debug_line      4668     0x0
.debug_pubnames  803      0x0
.debug_pubtypes  71       0x0
Total            61338

```

```

root@376d134fd775:/home/app/blinky# cargo objdump --bin blinky --release -- --disassemble --no-show-raw-insn --print-imm-hex
  Finished `release` profile [optimized + debuginfo] target(s) in 0.58s

blinky: file format elf32-littlearm

Disassembly of section .text:

08000258 <__stext>:
8000258:    bl      0x800047e <__pre_init> @ imm = #0x222
800025c:    ldr     r0, [pc, #0x38] @ 0x8000298 <__stext+0x40>
800025e:    ldr     r1, [pc, #0x3c] @ 0x800029c <__stext+0x44>
8000260:    movs   r2, #0x0
8000262:    cmp     r1, r0
8000264:    beq    0x800026a <__stext+0x12> @ imm = #0x2
8000266:    stm    r0!, {r2}
8000268:    b      0x8000262 <__stext+0xa> @ imm = #-0xa
800026a:    ldr     r0, [pc, #0x34] @ 0x80002a0 <__stext+0x48>
800026c:    ldr     r1, [pc, #0x34] @ 0x80002a4 <__stext+0x4c>
800026e:    ldr     r2, [pc, #0x38] @ 0x80002a8 <__stext+0x50>
8000270:    cmp     r1, r0
8000272:    beq    0x800027a <__stext+0x22> @ imm = #0x4
8000274:    ldm    r2!, {r3}
8000276:    stm    r0!, {r3}
8000278:    b      0x8000270 <__stext+0x18> @ imm = #-0xc
800027a:    ldr     r0, [pc, #0x30] @ 0x80002ac <__stext+0x54>
800027c:    mov.w  r1, #0xf00000
8000280:    ldr     r2, [r0]
8000282:    orr.w  r2, r2, r1
8000286:    str     r2, [r0]

```

Audience POLL Question

Have you ever used low-level gdb tools like this to debug your application?

- a) Yes
- b) No

•• Next Steps

04

Embedded Rust Docker Container

- https://mailchi.mp/beningo/embedded_rust_docker_container
- Rust Toolchain
- Embedded Tools

Beningo Rust Docker Container



Additional Resources

Please consider the resources below:

- [Jacob's Blogs](#)
- [Jacob's CEC courses](#)
- [Embedded Software Academy](#)
- Embedded Bytes Newsletter
 - <http://bit.ly/1BAHYXm>

www.beningo.com



Consulting

Coaching

Training



DesignNews

Thank You

Sponsored by

DigiKey

