Using non-C languages with Zephyr

David Brown
Linaro
2022-06-09
2022 Zephyr Developer Summit

Zephyr

- We're at the Zephyr Dev Summit, so I won't spend a lot of time here
- Zephyr is mostly C code, with a smattering of assembly
- Lots of python and CMake, but that runs at build time
- Complex and rich build system
  - Kconfig
  - CMake
  - Device Tree

A Zephyr App

- west creates a directory tree, with Zephyr and other sources
- My app can be elsewhere

```
cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(myapp)
...
```

- Zephyr's build is in charge
- It does a **lot**, not just build
  - determine configuration
  - compile files
  - figure out devices and such
  - builds numerous tables and such, e.g. syscalls
  - links it all together

Non-C Languages

- Why?
    - Safety/security, robustness in general
        - Pointer/array safety
        - Memory allocation/deallocation safety
        - Fun
    - Available software
        - Often the reason for C, but a lot is coming out in other languages
- The focus here
    - C++, not really, already mostly supported
    - Python/JS/Lua, some support, different focus
    - Focus: low-level languages
        - Compile to native code
        - Small runtime
- Cover 3 languages: 1 you care about, 1 you don't, and 1 you probably haven't heard of

The Languages

- Rust (https://www.rust-lang.org/)
    - Lots of activity, fairly stable
    - Some work on embedded, but more on standalone
    - Pointer safety through borrow checking
- Zig (https://ziglang.org/)
    - Newer, more active development and design
    - Simpler than Rust, still has checked arrays, pointer safety, etc
    - Very focused on low-level, no constructs that generate a bunch of code
- Ada (https://www.adaic.org/advantages/ada-overview/)
    - No, I'm not joking, sort of
    - Very mature (part of GCC since 1995)
    - Probably not relevant, esp with Rust, but there are things to learn

Rust: Borrow Checker

- Goal is full pointer safety, no null, no use after free
- Many languages handle this with a Garbage Collector (GC) (Lisp/Scheme, Java, .NET, Haskell, Go, etc)
- GC not really suited for embedded
- Rust Compiler sets to track and validate “ownership” of referenced data
- Restricts what you can point to, but the result can never be invalid
- Provides reference counting and other things for cases where this doesn’t work
- It worked better than expected
- Once passed learning experience, programming feels fairly free
- Get best of both worlds, no pointer problems, yet fairly freely written code
- Has deeper consequence than pointers, including thread safety and resource cleanup

Rust: Cargo and Ecosystem

- Software encapsulated into “crates” that are compiled separately
- Tools and ecosystem (crates.io) to manage dependencies
- Versioning “hell” solved mostly by allowing multiple version of a crate
- Will likely need to be more closely managed to be useful for small embedded
- crates.io has 84,436 crates registered, with over 16 billion served
- To a Rust developer, a magical ideal world would be to have Zephyr as just a crate that is brought in
- That’s not what I’m doing here

Zig: Philosophy

- A Simple Language
  - No hidden control flow
  - No hidden memory allocations
  - No preprocessor, no macros
- Compile time
  - Call any function at compile-time
  - Manipulate types as values without runtime overhead
  - Comptime emulates the target architecture

Zig: Practically

- Still < 1.0 release
- A lot of neat ideas
    - Things that seem kind of similar, struct/module, generic/function are actually the same in Zig
    - Embracing compile time evaluation, instead of ifdefs, it is just ordinary code flow
    - Some strong robustness ideas: error handling, option types
    - Allocation is passed into modules, not assumed, easily replaced with debugging version, or target specific
- Zephyr probably needs to do less to support Zig

Ada

- First version 1983, then 1995, 2005, 2012.
- Had grand ideas, many of which were deemed to expensive to implement in a compiler (like things similar to what Rust has done with the borrow checker)
- Was mandated by the US Military for a while, still used in some safety-critical applications
- Preference for lots of keywords instead of symbols makes it verbose
- Still has some good ideas, but I think Rust addresses more now
- Still has good ideas around multi-threading and real-time that seem to get neglected in most modern RTOSes
- Probably not practical or interesting on Zephyr

The code

- https://github.com/tangybbq/non-c-on-zephyr
- Feel free to follow along, look through the code as we go through it

- Let's start with just a bit of an overview of each language, without Zephyr

01 Rust Hello

- Rust's standard build is with Cargo
- Cargo does **everything** around the build
  - Fetch and maintain dependencies
  - Compile and cross compile code
- The file `Cargo.toml` directs this
- This code is in 'src'

02 Zig Hello

- Compiling managed by Zig
- Each source file is a module, modules reference other modules
- Outside dependencies and such is WIP
- Hello is a bit more complex, things like stdio, and an allocator aren't assumed
- Error handling is explicit, similar to Rust
- In Zig, many concepts are different, much more can be const, and even code can be run at compile time

03 Ada Hello

- gnatmake handles build compilation
- Not really any dependency management, wasn't really a thing
- Has "spec" files and "body" files, like headers but actually compiled
- Language designers aggressively thought through things many languages overlook (e.g., elaboration order, what would be static ctor in C++)
- Would have liked to do more, but computers of the time couldn't really do it
- In some ways, Rust is a continuation of some of these goals
- Fortunately, not the verbosity

Common among them

- Each language has a build tool that manages dependencies, compilation order, etc
- Normally, the build tool will produce an executable
- This conflicts what what Zephyr's build system wants
- For each, we'll take advantage of the language's ability to produce libraries that can be linked into a C program

A hidden difficulty

- Both Zig and Rust are built around LLVM
- Zephyr mostly builds with gcc. LLVM cross-compiling hasn't worked in a while. Arm-clang may help
- ABIs are largely compatible
- compiler support libs somewhat, but we will encounter conflicts

04 Rust Zephyr Hello

- Simple Hello world, but a Rust ‘main’ running within Zephyr
- A CMake file adds the rust code as an “external” dependency
- Use a handful of clumsy mechanism to tell rust that this code is “bare metal”, and being used as a library
- We have to implement panic, for now it can just hang, but should probably hook back into Zephyr
- We won’t have ‘std’, so many crates won’t work (but see later)
- Rust can call C functions, but prototypes have to be replicated in Rust declarations

Rust/Zephyr walkthrough

05 Zig Zephyr Hello

- Same approach, tell ‘zig’ we’re making a library (and cross compiling)
- Similar CMake file to invoke zig to build library
- Similar workarounds for the compiler library conflicts
- Zig runtime requirements are less, embedded seems to be a default assumption when our target is chosen. ‘std’ is still available, but will have far fewer things available, that are conditional based on the target
- For now, declare C functions from Zephyr directly with Zig declarations

Zig/Zephyr Walkthrough

06 Ada Zephyr Hello

- More work to setup dev environment, LLVM does cross compilation so much better than GCC
- Cross compiling for Arm with GNAT starts to push boundaries of what AdaCore wants to charge large amounts of money for
- Otherwise, fairly similar to the others
- But, Ada's builtin library is quite a bit more extensive, the compiler likes to call lots of things defined by it

Ada/Zephyr Walkthrough

(if I can get it working, otherwise sadness and moving on)

Let's Do More than "Hello World"

- Biggest frustration, much of Zephyr's public API aren't actually functions, but built-time-generated macros.
- A few solutions:
  - Write C wrappers, these can then be called
  - Replicate what the macros are doing in our language, awkward because what the macros do depends on configuration
- Mismatches:
  - Rust and Zig both have built-in logging frameworks. Not difficult to write a backend that forward to log (at least Zephyr v2 log), but it adds a lot of code
  - Zephyr's LOG system is heavily based around C macros and C-varargs, and C's unsafe string formatting, and preference for nul-terminated strings
- Mechanisms in these languages may not line up with Zephyr's

Logging

- Rust
  - Implement a log handler
  - Rust log calls package formatting data with borrowed data to be logged
  - Log handler can turn these to strings to send off to Zephyr, but can't really keep references as they are borrowed and will go away when we return
- Zig
  - Similar, we write our own handler
  - Similar issues having to do string formatting to buffers to give to Zephyr
- Ada
  - Doesn't have a standard logging mechanism
  - Could just wrap Zephyr's, except Ada eschews C-style string formatting, etc

Syscalls

- Lots of Zephyr's API are syscalls
- Generated macros translate to wrapper calls
- How this happens depends on configuration
- Do we replicate this configuration within Rust/Zig
- Many other APIs rely on inline functions, which are also problematic
    - Much of this wraps calls through pointers, simulating OO
    - Others are just attempts to get performance, or perceived performance
    - Could we just have this be regular functions in Zephyr, and use LTO to get better code?

Zig C interface

- Zig code can use C headers directly:

- This isn't working with Zephyr, as the headers are angry with the C environment it sees.
- This might be fixable with proper #defines to make it look like the environment we're coming from.
- Supports simple macros, external definitions, maybe some inline functions
- Result will still be a very "C" API, without pointer safety and such

Zephyr-Rust

- https://github.com/tylerwhall/zephyr-rust
- Is a port of Zephyr's 'std' library to Zephyr
- Would allow many more Rust programs to just run
- Unfortunately, only works with very specific versions of both Zephyr, and rustc
- And doesn't seem to be getting any recent work
- Something like this would probably only really work with a good bit of regular work on it, and it should probably be included in both project's CIs to keep things from constantly breaking

Where to from here?

- Rust: likely, Zig: maybe, Ada: unlikely
- Rust would be a lot of work, but fruitful if possible
- Zig is probably easier to interface, harder to see where the language is going
- Ada was more as a historical interest
- vlang, nim, crystal, ...? Maybe, but we should probably focus on one thing
- We don't expect a "rewrite it in Rust" for Zephyr. Zephyr is still a C RTOS
- But, some awareness of things that would make Rust easier wouldn't hurt

- What kind of interest is there?

Thank you

Linaro