Build Your App with Zig

Southeast Linux Fest 2025
Charlotte, NC

Cameron Conn

What?

- This is a talk about **Zig**
- Low-level programming language, like C
- Not a 30-minute tutorial on how to be a Zoomer Zig coder

This talk in 100 seconds

- Zig has no magic 💩, macros, or hidden flow
- Low level w/ manual memory management
- `zig` cross-compiles Zig *and C*
- Optional bounds-checked arrays
- `comptime`, error handling, defer cleanup, SIMD
- Zig is C, but *way* better

In this Talk

- Motivations
- Why/where use to Zig right now
- How to use Zig w/o programming
- Not an in-depth course on Zig
  - goto docs if you want a lesson

What your favorite programming language says about you

Don’t use Zig

- Zig is *alpha quality* software
- Frequently breaking changes
- Few resources or books online

is:issue state:open label:miscompilation

Labels | Milestones | New issue

Open 63 | Closed 168 | Author | Labels | Projects | Milestones | Assignees | Types | Newest

- **mips: Compiling for mips2 generates mips4 instructions** arch-mips backend-llvm bug miscompilation upstream
  #24089 · Sirius902 opened last week · unplanned — 1
- **mips{,64) f16 infinity works with explicitly inline function, but not normal function** arch-mips backend-llvm bug miscompilation upstream
  #24066 · rootbeer opened last week · 0.15.0 — 2
- **`vector reduce operation` fails on `sparc64-linux`** arch-sparc backend-llvm bug miscompilation upstream
  #23719 · alexrp opened on Apr 28 · unplanned
- **`LLVM ERROR: SPARCv8 does not handle f128 in calls; pass indirectly` for `sparc-*` targets** arch-sparc backend-llvm bug miscompilation upstream
  #23674 · alexrp opened on Apr 26 · unplanned — 2
- **LLVM compilation error when assigning enum to union at runtime** backend-llvm bug miscompilation
  #23577 · Abstract-Everything opened on Apr 15 · 0.15.0
- **@errorName misbehaves when global error set type has a size of 1 byte** bug frontend miscompilation
  #23533 · wooster0 opened on Apr 10 · 0.15.0
- **Sema: saturating left shift produces an incorrect safety check and invalid Air** bug frontend miscompilation
  #23033 · jacobly0 opened on Mar 1 · 0.15.0
- **Surprising aliasing despite explicit copy** backend-llvm bug miscompilation
  #22906 · jamii opened on Feb 15 · 0.15.0

Don’t use Zig (cont.)

Skipping Stuff

- SIMD Vectors (cool)
- Enum
- Tuples, varargs
- And much more...
- For better coverage, see ziglang.org

Part I: Crash Course In Zig

zig zen

Syntax

- `https://github.com/camconn/self-2025`
- See: `self-2025/src/main.zig`

Tastes Like C~~hicken~~

C	Zig
`uint8_t, …, int64_t`	`u8, ..., i64`
`bool, true, false`	`bool, true, false`
`float, double`	`f32, f64`
`char *`	`[]u8`
`char * (w/ null term)`	`[:0]u8`
`float, double`	`f32, f64`
`bool`	`bool`
`!, &&, ||, (Logical)`	`!, and, or`
`!, >>, <<, &, | (Bitwise)`	`!, >>, <<, &, |`
`func(foo)`	`func(foo)`

Numbers

- Integers are signed or unsigned
  - u8, i8, u32, i32, etc.
  - Have arbitrary width (1 up to $2^{16} - 1$)
    - i7, u2025, u1337
- IEEE 754 types:
  - f16, f32, f64, f80, f128

Numbers: To and From

- Convert int to float with explicit calls
- `@truncate`, `@intCast`, `@floatFromInt`, `@intFromFloat`

Numbers: Safety

- Overflow checks for number ops
- Explicit overflow-underflow permitting versions
- See: `src/overflow.zig`

Pointers

- Pointers are non-null addresses in memory
- Have an alignment and a type
- `var x: *i64 = &y;`

Control Flow

- `if`
- `while`
- `for`
  - Range syntax, multiple items
- `break :label`

Arrays & Slices

- Arrays have comptime-known length
- Slices have runtime-known length
  - Pointer + runtime-known length
  - Prefer slices over arrays
- Arrays → slices, but slices !→ arrays
- See: `src/slice.zig`

Union & Structs

- Unions can be optionally tagged w/ enums
  - Tagging prevents type confusion from C
- Structs are just like you think
- Unions & structs can be `extern` or `packed` for interop w/ C

Optional Types

- Optional types wrapping an inner type
  - Like Rust's `Option<T>`
- Unwrap with `.?` or `orelse` or `if`
- You can have null pointers with `?*usize`

defer and errdefer

- Need to close a file? Procrastinate!
  - Use `defer file.close()`
- Optionally free resources if there’s an error?
  - Use `errdefer alloc.free(my_var)`

Error Handling

- Funcs can return types beginning with !
  - Indicates an optional error
- To propagate the error to the caller and unwrap success, do `try func()`
- Coerce error union to some value with `catch`
- See: src/err.zig

Memory

- Want `malloc`? Too bad!
- See: `src/mem.zig`
- Pick an `Allocator`
  - `GeneralPurposeAllocator`, `HeapAllocator`, `FixedBufferAllocator`, `ArenaAllocator`
- `Allocator.alloc()`, `defer Allocator.free()`

Breakout to Assembly

- `asm volatile ( … )`
- In case you need it
- AT&T Syntax

Straight outta `comptime`

- `comptime` lets you run code at compile time
- Create look-up tables
- Hash input files
- Generics like `std.ArrayList(i32)`

comptime example

- See: `src/comp.zig`

Part 2: Unicode to machine code

Build w/ Zig

- Zig ships with the `zig` compiler
- `zig run hello_world.zig`
- `zig build [thing]` (defined in build.zig)
- See: `build.zig`

Cross Compilation

- Zig has all of the targets included in distribution
  - Listed with `zig targets` command.
- `zig build -Dtarget=<NAME>` or `zig build-exe -target <NAME>`
- Example Targets:
  - x86_64-linux-gnu, x86_64-windows-gnu
  - aarch64-linux-gnu, aarch64-freestanding
  - riscv64-linux-musl
  - nvptx64-cuda-none, amdgcn-amdhsa-none

Cross-compile demo

- Cross compile with `zig build -Dtarget=<NAME>` or `zig build-exe -target <NAME>`
- Cross compile for Arm64:
  - `zig build-exe src/main.zig -target aarch64-linux-gnu`
- RISC-V
  - `zig build-exe src/main.zig -target riscv64-linux-musl`

`zig cc`

- Zig compiles C too!
  - `zig` ships a C compiler (clang)
- Cross compiles w/ LLVM
- No more 21 GB toolchain downloads!
- Very useful for cross-compiling
  - Used in prod at Uber

`zig cc` demo

- Cross compile C code for riscv64

Zig Cross Limitations

- Some exotic targets and as well supported, like LLVM

Deprecate C with Zig

- `zig translate-c` converts C code to Zig
- Show before-and-after example
- Uses `std.libc` functions
  - I lied, you have `std.c.malloc`
- See: `wc.c`, `wc_fixed.zig`

Builds

- C uses Makefiles, CMake, Scripts
- Zig code is built in a project with a build.zig
- Defines module roots, dependencies, and created exes, libs
- zig does <build|build-exe|build-lib|build-obj>

Builds (cont.)

- Build w/ `zig build`
- See: `build.zig`

Testing

- With C, you need external test frameworks
  - Gtest, cppcheck
- With Zig, it’s built-in
  - `zig test path/to/file.zig`
  - Or in a project with a target, e.g. `zig build test`

Testing (demo)

- `zig test src/tester.zig`

Wrap-Up

- Zig does everything C does better
- C is old and yucky

Slides

camconn.cc/self-2025

Extras