EOSS EMBEDDED OPEN SOURCE SUMMIT

Zephyr Footprint – Where Are We and Where Are We Going

Ederson de Souza

OS Development Engineer

intel.

Zephyr® Project
Developer Summit 2023

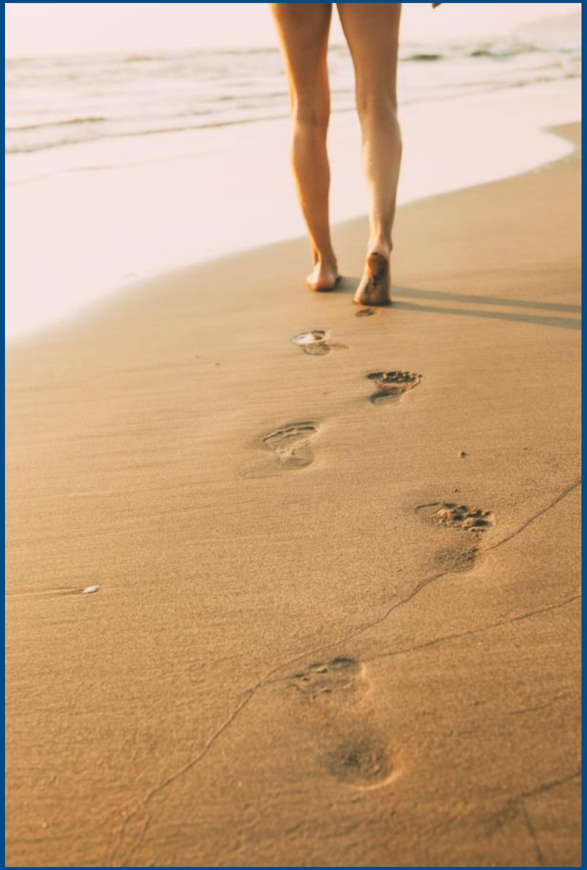
Agenda

- Footprint?
- Tools
- Common Hints
- Experiments
  - Function pointers considered harmful
    - Not really
  - What about LTO?
- Ending thoughts
- Q&!A

Footprint?

Where are we?

- How big is the impact on some resource
  - ROM/Disk/Flash
  - RAM
  - Power consumption
- Why do we care about it?
  - Save resources
    - Money
    - Energy



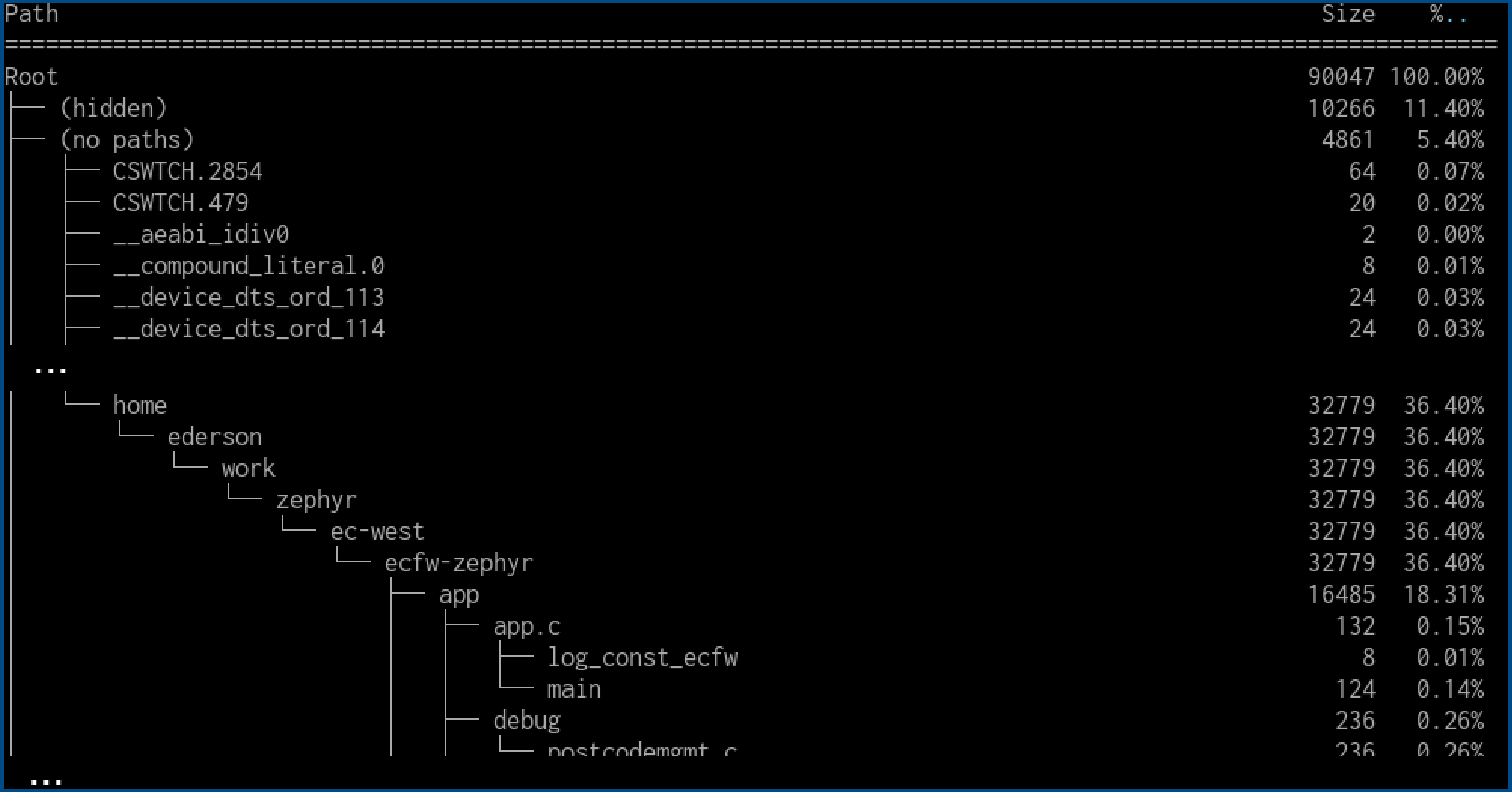
Tools

- But first, let's measure it
  - Otherwise, improvements can't be proven
  - Nor regressions detected
- Zephyr has a nice ROM report tool

```
west build -t rom_report
```

Tools (II)

Output from rom_report



```
Path                                                          Size     %..
==========================================================================
Root                                                         90047 100.00%
├── (hidden)                                                 10266  11.40%
├── (no paths)                                                4861   5.40%
│   ├── CSWTCH.2854                                             64   0.07%
│   ├── CSWTCH.479                                              20   0.02%
│   ├── __aeabi_idiv0                                            2   0.00%
│   ├── __compound_literal.0                                     8   0.01%
│   ├── __device_dts_ord_113                                    24   0.03%
│   ├── __device_dts_ord_114                                    24   0.03%
  ...
│   └── home                                                 32779  36.40%
│       └── ederson                                          32779  36.40%
│           └── work                                         32779  36.40%
│               └── zephyr                                   32779  36.40%
│                   └── ec-west                              32779  36.40%
│                       └── ecfw-zephyr                      32779  36.40%
│                           ├── app                          16485  18.31%
│                           │   ├── app.c                      132   0.15%
│                           │   │   ├── log_const_ecfw           8   0.01%
│                           │   │   └── main                   124   0.14%
│                           │   ├── debug                      236   0.26%
│                           │   │   └── postcodemgmt.c         236   0.26%
  ...
```

Common hints

- Disable unused features/subsystems
- Avoid holes in structs
  - `west build -t pahole`
- Limit number of threads
- Logging
- Power management
- Try different toolchains

Experiments

Where are we going?

Setting expectations

- Experimenting what we *could* do
  - So, we can ask if we *should* do
  - Not prescribing *how*
- Some test were not checked on runtime
  - So, things may be broken
- Used some open-source projects on the tests
  - Not implying anything
- Focus on application size

Function pointers considered harmful

- Compiler loses visibility on what's being used
  - Dead code elimination misses
- Zephyr APIs extensively use them

```
// Declaration
__subsystem struct kscan_driver_api {
    kscan_config_t config;
    kscan_disable_callback_t disable_callback;
    kscan_enable_callback_t enable_callback;
};

// Driver "instantiation"
static const struct kscan_driver_api kscan_npcx_driver_api = {
    .config = kscan_npcx_configure,
    .enable_callback = kscan_npcx_enable_interface,
    .disable_callback = kscan_npcx_disable_interface,
};
```

Function pointers considered harmful (II)

- Zephyr APIs extensively use them (II)

```
// API usage
__syscall int kscan_enable_callback(const struct device *dev);

static inline int z_impl_kscan_enable_callback(const struct device *dev)
{
    const struct kscan_driver_api *api =
            (const struct kscan_driver_api *)dev->api;

    if (api->enable_callback == NULL) {
        return -ENOSYS;
    }

    return api->enable_callback(dev);
 }
```

Function pointers considered harmful (III)

- What if we have something like C++ templates?
  - `` `kscan<npcx>` ``?
- Could `` `_Generic ` `` come to help?
  - How to have the "type" at coding time?
  - Maybe DTS can help here?
- Didn't explore this line further
  - But it could be interesting

Function pointers considered harmful (IV)

- But not all is lost
  - Maybe have a "static dispatcher" table?
- Some macros, regex and code generation can help
  - Let's try it!

Static dispatcher

- A script, "gen_static_dispatch.py"
  - Basically, greps for the API "instantiation" and generate some functions

```
int _static_kscan_npcx_enable_callback(const struct device *dev) {
    return kscan_npcx_enable_interface(dev);
}
```

  - Also, generates a "dispatcher" for them, that the API can use

```
static inline int z_impl_kscan_enable_callback(const struct device *dev)
{
#ifdef CONFIG_STATIC_DISPATCH_KSCAN
    return static_dispatch_kscan_enable_callback(dev);
#else
(...)
```

Static dispatcher (II)

- A script, "gen_static_dispatch.py" (II)
  - And the "dispatcher"

```
static inline int static_dispatch_kscan_enable_callback(const struct device *dev){
#ifdef CONFIG_KSCAN_NPCX
    extern int _static_kscan_npcx_enable_callback(const struct device *dev);
    return _static_kscan_npcx_enable_callback(dev);
#endif
    return -EINVAL;
}
```

  - Which we include in the driver code

```
#include "static_dispatch_kscan_npcx.c"
```

Static dispatcher (III)

Function pointers considered harmful? **Not really**

- What if there's more than one driver for the same subsystem enabled at the same time?
  - We'd need some way to know the "type" of a device in runtime
    - Another field on "dev" struct, some "pointer tag", etc
  - And a switch to chose the right API
  - On preliminary tests, this *increased* the footprint
    - So, experiments with "static dispatching" proceeded only when there was a single driver for a subsystem

Static dispatcher (IV)

Rom reports

- After some use of regex, got a few subsystems ready for testing
  - ADC, Clock control, Display, ESPI, Flash, GPIO, I^{2}C, Kscan, PS2, PWM, Regulator, Sensor, UART, Watchdog
- Some open source projects
  - ZSWatch (https://github.com/jakkra/ZSWatch)
    - Board: zswatch_nrf5340_cpuapp
  - Intel EC FW (https://github.com/intel/ecfw-zephyr)
    - Board: mec1501_mtl_p
  - ZMK* (https://zmk.dev)
    - Board: planck_rev6
- As static dispatch work was done on Zephyr's main branch, those projects were rebased on top of it
- Zephyr SDK 0.16.1

Static dispatcher (V)

ZSWatch and Intel EC FW

- ZSWatch

No static dispatch	Static dispatch	Difference	%
629354	628498	856	0.1

  - Not really impressive, given size of application
  - Still something, I guess

- Intel EC FW

No static dispatch	Static dispatch	Difference	%
91478	90047	1431	1.5

  - More interesting – smaller applications shall get more gains

Static dispatcher (VI)

The curious case of ZMK

- ZMK

No static dispatch	Static dispatch	Difference	%
34739	35411	-672	-1.9

  - It actually got bigger!
  - Looking at the report, it seems some dead code was activated instead
  - Not sure why, at the moment of writing this presentation
    - If you know what's going on, let me know!
  - But it gives some inspiration to go a bit further…

What about LTO?

Link Time Optimization

- "Ultimate" dead code elimination
- Old "dream"
  - https://github.com/zephyrproject-rtos/zephyr/issues/2112
  - Reports of downstream use
- Basically, add "-flto=auto -ffat-lto-objects"
  - `west build -b <board> -- -DEXTRA_CFLAGS="-flto=auto -ffat-lto-objects"`
- Need a few patches on Zephyr
  - Mainly, add `__used` to some functions and variables

What about LTO?(II)

Results (ZMK keeps on surprising...)

- ZSWatch

Before	Static dispatch (gain/%)	LTO (gain/%)	LTO + Static Dispatch (gain/%)
629354	628498 (856/0.1)	627465 (1889/0.3)	626553 (2801/0.4)

- Intel EC FW

Before	Static dispatch (gain/%)	LTO (gain/%)	LTO + Static Dispatch (gain/%)
91478	90047(1431/1.5)	81436 (10042/11.0)	78912 (12566/13.7)

- ZMK

Before	Static dispatch (gain/%)	LTO (gain/%)	LTO + Static Dispatch (gain/%)
34739	35411(-672/-1.9)	30092 (4647/13.4)	30728(4011/11.5)

Ending thoughts

Where shall we go?

- LTO provides biggest gains
- Is static dispatch really useful?
    - Interesting gains
    - How to implement it?
        - Not sure macros + code generation is the way
    - Could devicetree help something more like templates?
- Need to ensure things do work!
    - No subtle bugs

EOSS EMBEDDED OPEN SOURCE SUMMIT

Q&!A

Find me on Zephyr Discord: edersondisouza#9895

intel®

Zephyr® Project
Developer Summit 2023

Thank you!

*Other names and brands may be claimed as the property of others