

LLEX: Extending Zephyr at Runtime

Tom Burdick, Intel Corporation



What is LLEXT?

Linkable Loadable Extensions

Extensions are runtime changeable behavior

Set of tooling to build, manage, and link extensions

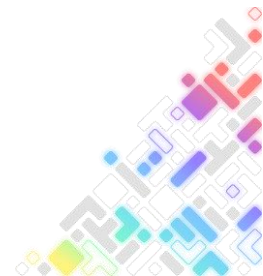
An ELF loader like...

- Linux Kernel Modules
- Linux Userspace Programs



Agenda

- Extensions and Memory Management
- Relocations
- Building Extensions
- Use cases
- Q/A



ELF¹⁰¹ a ~~Linux executable~~ walk-through

ANGE ALBERTINI
CORKAMI.COM



Zephyr Extension

DISSECTED FILE

```
-$ uname -m  
armv7l  
-$ ./simple.ARM  
Hello World!
```

SIMPLE.ARM

HEADER^{1/2}

TECHNICAL DETAILS FOR IDENTIFICATION AND EXECUTION

SECTIONS

CONTENTS OF THE EXECUTABLE

HEADER^{2/2}

TECHNICAL DETAILS FOR LINKING (IGNORED FOR EXECUTION)

ELF HEADER

IDENTIFY A FILE
SPECIFY THE ARCHITECTURE

PROGRAM HEADER TABLE

EXECUTION INFORMATION

CODE

EXECUTABLE INFORMATION

DATA

INFORMATION USED BY THE CODE

SECTIONS' NAMES

IDENTIFY TAB...
LINKAGE

SECTION HEADER TABLE

LINKING (CONNECTING PROGRAM OBJECTS) INFORMATION

HEXADECIMAL DUMP

ASCII DUMP

FIELDS

VALUES

EXPLANATION

1

e_ident

ELI_MAGIC

ELI_CLASS

ELI_DATA

ELI_VERSION

e_type

e_machine

e_version

e_entry

e_shoff

e_phoff

e_shsize

e_phsize

e_shentsize

e_phentsize

e_shstrndx

e_phstrndx

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

e_flags

0x7F, "ELF"

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

CONSTANT SIGNATURE

32 BITS LITTLE-ENDIAN

ALWAYS 1

EXECUTABLE

ARM PROCESSOR

ALWAYS 1

ADDRESS WHERE EXECUTION STARTS

PROGRAM HEADERS OFFSET

SECTION HEADERS OFFSET

ELF HEADERS SIZE

SIZE OF A SMALL PROGRAM HEADER

COUNT OF PROGRAM HEADERS

SIZE OF A SMALL SECTION HEADER

COUNT OF SECTION HEADERS

INDEX OF THE NAMES SECTION IN THE TABLE

THE SEPARATOR SHOULD BE LOADED IN MEMORY

OFFSET WHERE IT SHOULD BE READ

VIRTUAL ADDRESS WHERE IT SHOULD BE LOADED

PHYSICAL ADDRESS WHERE IT SHOULD BE LOADED

SIZE OF FILE

SIZE IN MEMORY

READABLE AND EXECUTABLE

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

ARM ASSEMBLY

EQUIVALENT C CODE

MOV R0, #1

ADD R0, R0, #20

MOV R0, #1

MOV R7, #0

SVC #0

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

WRITESTROUT, "hello world!\n", len("hello world!\n")

STRINGS

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

"hello world!\n", 0

SECTION NAMES

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

..shdrtab, .text, .rodata

SECTION HEADER TABLE

INDEX NAME TYPE FLAGS ADDR SIZE OFFSET SHN

0 .null 0 0 0 0 0 0

1 .text 1 0 0 0 0 0

2 .rodata 2 0 0 0 0 0

3 .shdrtab 3 0 0 0 0 0

4 .shdrtab 4 0 0 0 0 0

5 .shdrtab 5 0 0 0 0 0

6 .shdrtab 6 0 0 0 0 0

7 .shdrtab 7 0 0 0 0 0

8 .shdrtab 8 0 0 0 0 0

9 .shdrtab 9 0 0 0 0 0

10 .shdrtab 10 0 0 0 0 0

11 .shdrtab 11 0 0 0 0 0

12 .shdrtab 12 0 0 0 0 0

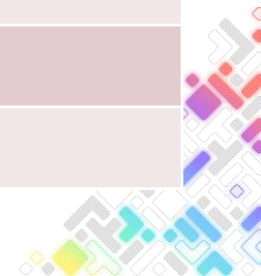
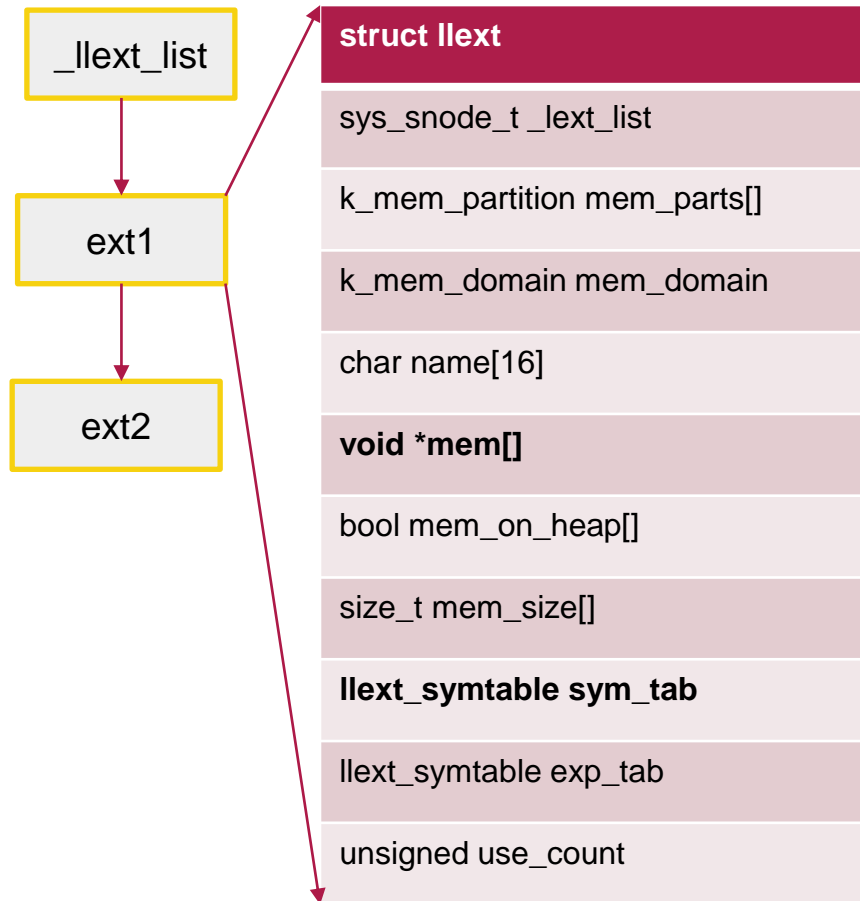
13 .shdrtab 13 0 0 0 0 0

14 .shdrtab 14 0 0 0 0 0

15 .shdrtab 15 0 0 0 0 0

LLEX: Extension manager

- `llex_load()`, `llex_unload()`, `llex_find_sym()`
- Maintains a list of loaded extensions
- Maintains a reference count for each extension
- Manages memory for extensions
- Manages symbol tables for the base firmware and each extension



LLEX: Memory Manager

- Extensions need memory
- Can be directly referenced in some cases or allocated on a LLEX dedicated heap
- Memory permissions need to be set with MPU/MMU in use
- Metadata for the extension
- .text needs read+execute
- .data, .bss needs read+write
- .rodata needs read

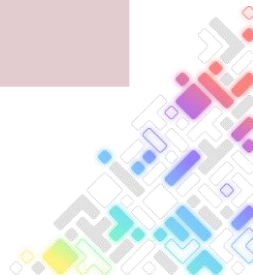
Metadata – read + write + supervisor

.text – read + execute + user + supervisor

.data – read + write + user + supervisor

.bss – read + write + user + supervisor

.rodata – read only + user + supervisor



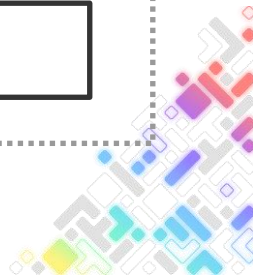
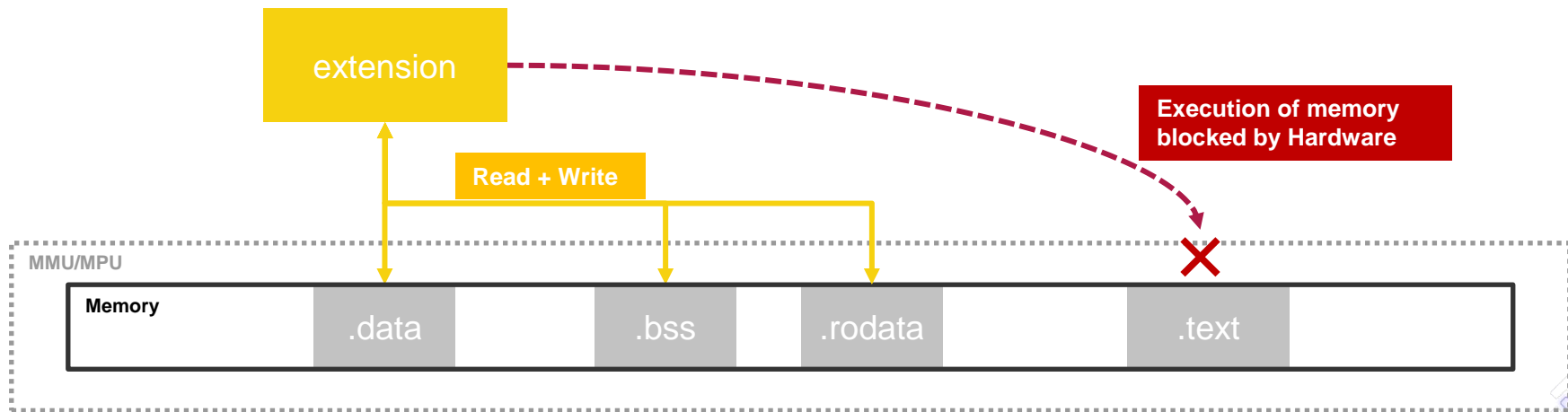
LLEXT: Memory Protection

CONFIG_MPU/CONFIG_MMU AND
CONFIG_USERSPACE

OR

NOT CONFIG_MPU/CONFIG_MMU

**Memory, with MPU/MMU, is not executable
outside of original .text in base firmware!**



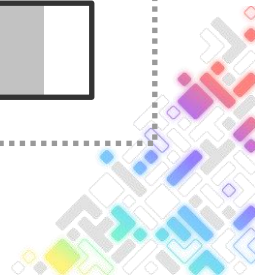
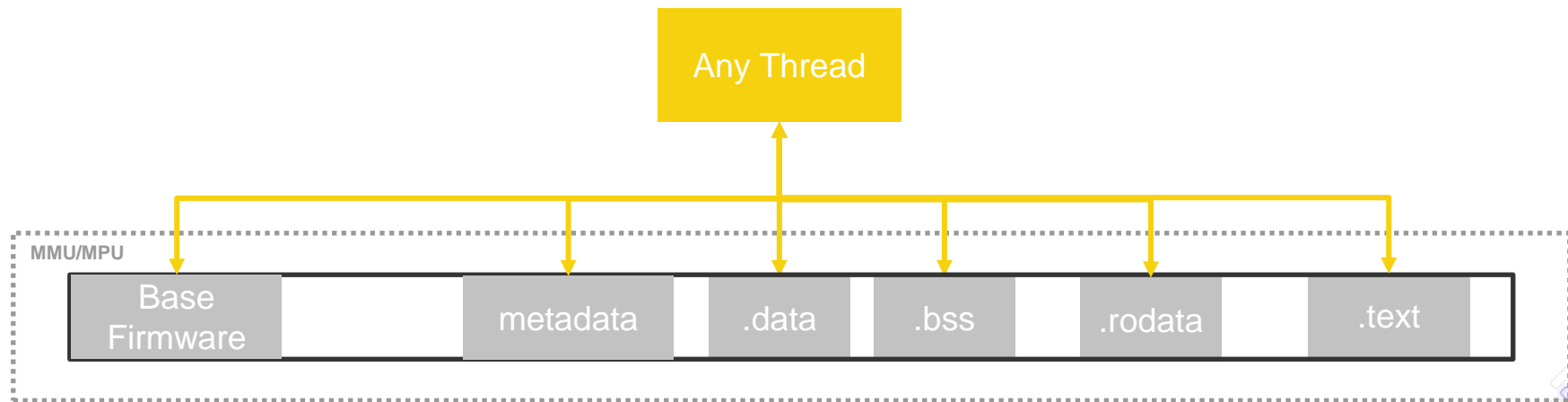
LLEX: Kernel space execution

NOT CONFIG_MPU/CONFIG_MMU

OR

All memory is accessible

Kernel mode thread calls into extension code



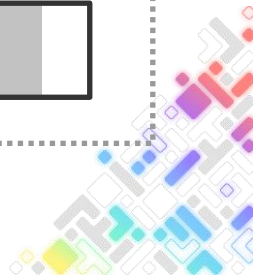
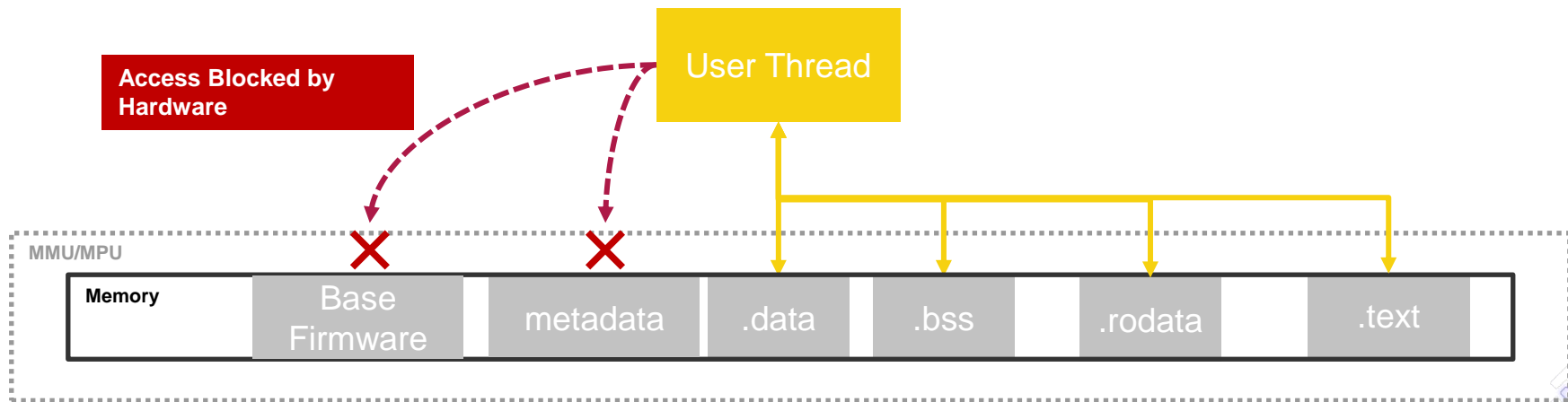
LLEXT: User space execution

CONFIG_MPU/CONFIG_MMU AND
CONFIG_USERSPACE

User thread assigned extension domain

A Process is Born!

**Only extension code, and
data is accessible! Syscalls
may be used.**



Relocations: Linking and Placement

arm-zephyr-eabi-objdump -r -d -x hello_world.elf

Disassembly of section .text:

```
00000000 <hello_world>:
 0: b580      push  {r7, lr}
 2: af00      add   r7, sp, #0
 4: 4b08      ldr   r3, [pc, #32] ; (28 <hello_world+0x28>)
```

```
28: 00000004      .word 0x00000004
      28: R_ARM_ABS32 .rodata
2c: 00000000      .word 0x00000000
      2c: R_ARM_ABS32 printk
30: 00000014      .word 0x00000014
      30: R_ARM_ABS32 .rodata
```

Rewrite with address of .rodata + 0x00000004

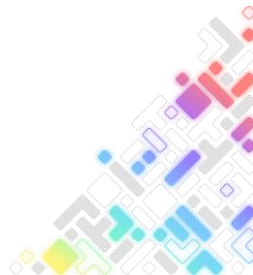
Rewrite with address of printk function

```
34: 4718      bx     r3
36: 46c0      nop                    ; (mov r8, r8)
```



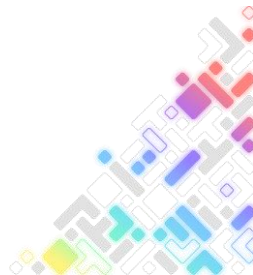
Relocations: Some key Points

- Kinds of Relocations that show up depend on
 - Architecture: x86, armv7m, xtensa lx7, etc
 - ELF linkage: shared, static, relocatable
 - Compiler flags: e.g. -mlong-calls
- May require opcode decoding, updating, and reencoding
- May require generation of a jump table
 - Opcodes are sometimes location dependent
 - Limits range of address accessibility for that opcode
 - E.g. PC relative call instructions encoded as 16bit instruction opcode



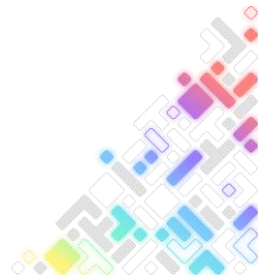
Relocations: Finding symbolic locations

- Sections may be interdependent (.text needs data from .data/.bss/.rodata)
- Symbolic linking requires a name and address pair to be found
- Relocation instructions are in ancillary ELF sections
 - .rel.text
 - .rel.bss
 - etc



Loading an ELF, High Level

1. ELF header and sections are read
2. Memory is Allocated or Referenced
3. Relocations are applied
4. Extension added to global list



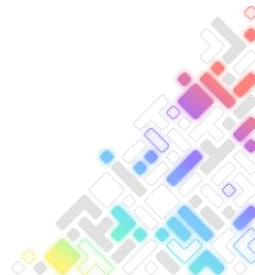
Building an Extension

Manually, great for exploring and tinkering

```
arm-zephyr-eabi-gcc -mlong-calls -mthumb -c -o hello_world.elf tests/subsys/llexthello_world/hello_world.c
```

Using Zephyr provided CMake Functions

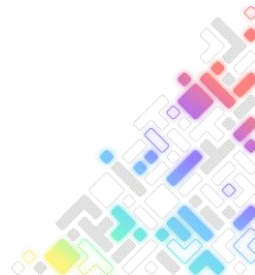
```
add_llexth_target(${ext_name}_ext  
    OUTPUT ${ext_bin}  
    SOURCES ${ext_src}  
)
```



Building an Extension: EDK

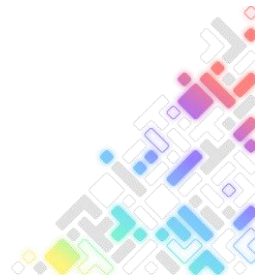
Extension Developer Kit – [PR 69831](#) – Base Application Source Not Needed

1. Application developer creates an application, with its own API in addition to that of Zephyr;
2. Application developer builds the EDK via `west build -t llex-edk`;
3. Application developer somehow makes the EDK available to extension developer;
4. Extension developer extracts EDK and includes it in its build system - the EDK provides some files to aid getting CFLAGS for cmake or make;
5. Extension developer can build the extension.



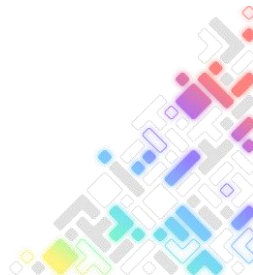
Use Cases – Today and Tomorrow

- Plugins; E.g. audio/video/sensing processing modules/plugins
- Isolated and Updatable Processes; E.g. multi-tenant firmware with base image
- Faster application build and load; Majority of application Logic in an Extension



Caveats Today, but maybe not Tomorrow

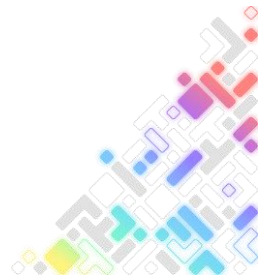
- Extension Signing is Needed; Untrusted ELF could do damage
- Debugging at an assembly level is tenable but slow
- Kernel objects need special care
- Dictionary logging; Missing some puzzle pieces
- Execute in Place (XIP) would require some additional work

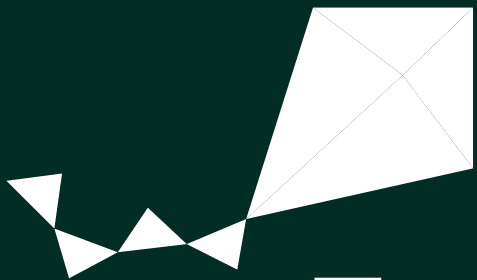


Credits

LLEXT was created by numerous people, and built on Zephyr infrastructure

- Chen Peng, Initial ELF Loader (Former Intel)
 - Lucas Burelli, Armv7m and CMake tooling (Arduino)
 - Guennadi Lyakh, Xtensa Support (Intel – Sound Open Firmware)
 - Ederson de Souza, EDK (Intel – Zephyr Team)
-
- Myself, Gluing it together (Intel – Zephyr Team)
 - A growing List of others!





Zephyr[®] Project

Developer Summit

