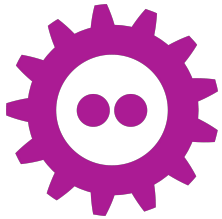


FOSDEM'24

Standardizing the generation and signing of boot images

2024-02-04

Neha Malcom Francis, TI
Simon Glass, Google
Vignesh Raghavendra, TI

Introduction to the Speakers

```
Neha-Malcom-Francis {
     software-engineer;
     Texas-Instruments;
     location = “Bangalore, KA, India”;
};
```

```
Simon-Glass {
     software-engineer;
     Google;
     location = “Boulder, CO, USA”;
};
```

```
Vignesh-Raghavendra {
     software-engineer;
     Texas-Instruments;
     location = “Bangalore, KA, India”;
};
```

Overview

- Quick intro to U-Boot
- Firmware packing: Modern Systems
- Boot flow of complex SoCs
- Binman!
- Extending Binman to support K3 boot-loaders
- Future development
- Extending to other devices

Das U-Boot: The Universal Boot-loader

- Open source bootloader for embedded devices
- Rich set of peripheral drivers and stacks
- Tightly integrated with the Linux kernel
- Multiple Archs: ARM, x86, RISC-V etc.

Device Tree and U-Boot

- U-Boot supports Device Tree
  - Popular way of describing platform in embedded world
- Device tree
  - Data structure describing the hardware
  - Acyclic graph, made of named nodes containing properties

```
#include <dt-bindings/soc/ti,sci_pm_domain.h

/ {
        model = "Texas Instruments K3 J721E SOC";
        compatible = "ti,j721e";
        interrupt-parent = <&gic500>;
        #address-cells = <2>;
        #size-cells = <2>;
[...]
        cpus {
                #address-cells = <1>;
                #size-cells = <0>;
                cpu-map {
                        cluster0: cluster0 {
                                core0 {
                                        cpu = <&cpu0>;
                                };
                                core1 {
                                        cpu = <&cpu1>;
                                };
                        };
                };
                cpu0: cpu@0 {
                        compatible = "arm,cortex-a72";
                        reg = <0x000>;
                        device_type = "cpu";
[...]
```

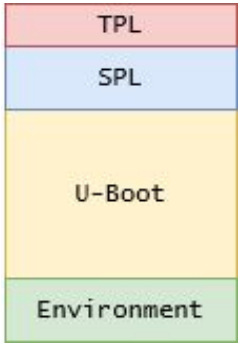
https://www.devicetree.org/specifications/

Firmware packing - Ancient

- Old approach: make
- Set location of environment: flash!

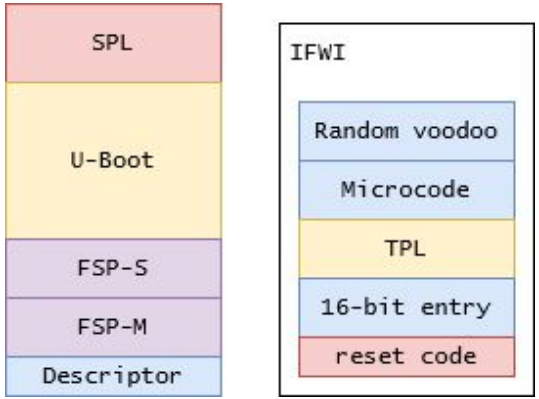
Firmware packing - Legacy

- SPL and TPL comes along
- cat them?
- ATF? FPGA FW?

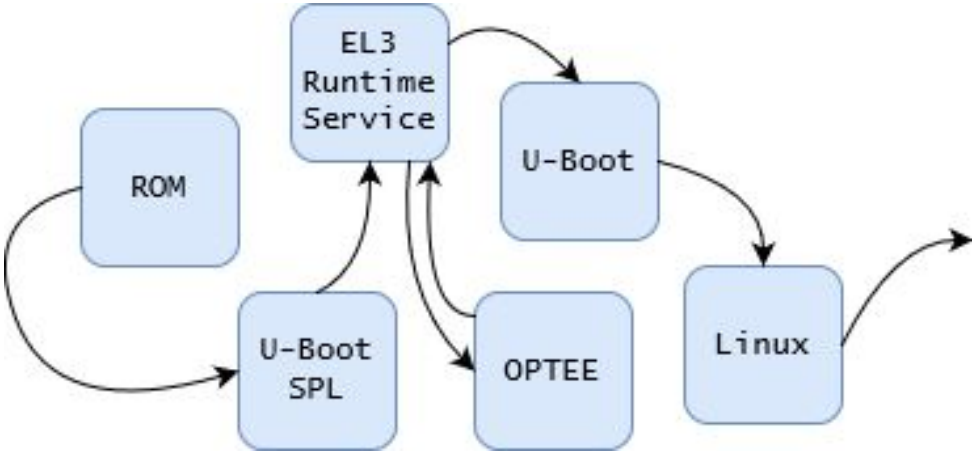


Firmware packing - Modern systems

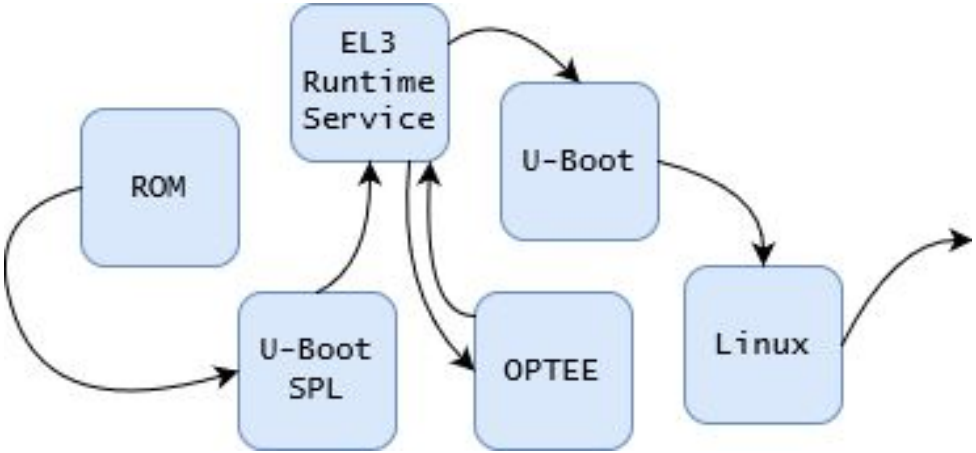
- x86
  - Various binary blobs
  - Needs 32-bit code to run FSP
  - IFWI



A typical ARM64 Boot Flow

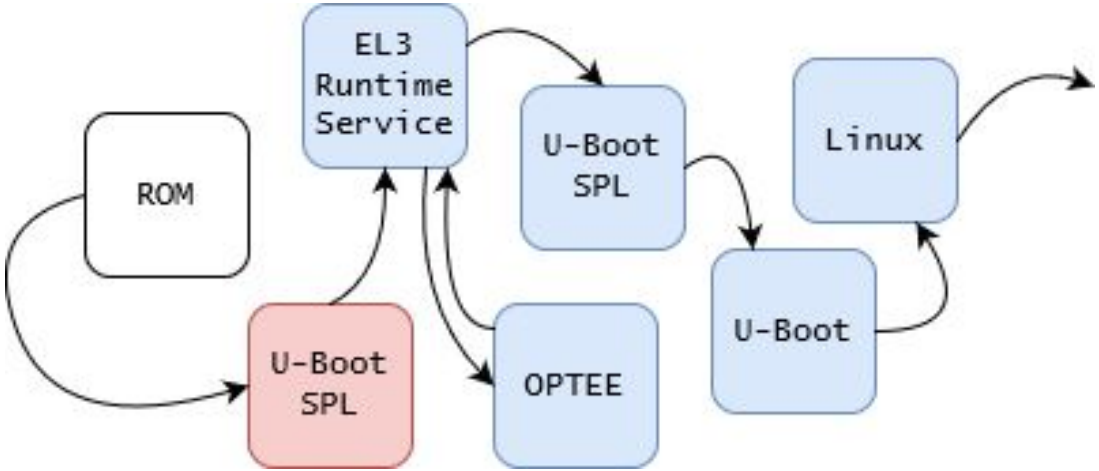


A typical ARM64 Boot Flow



Heterogenous SoCs?

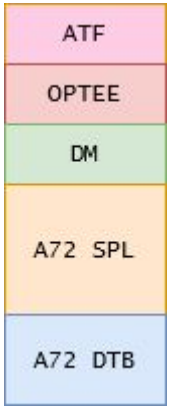
TI K3 Boot Flow - Heterogenous SoCs



- Need to boot R5 (32-bit) and A72 (64-bit) cores
- First SPL runs on boot master (R5)
- Another one on A72

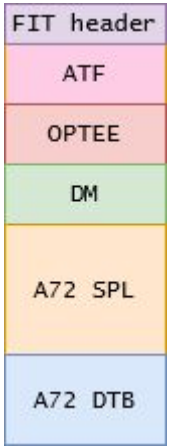
TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A72) Bootloader
  - Device Management Firmware (TI version of ARM SCP)
  - OPTEE, ATF → ARM standard FWs



TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A72) Bootloader
  - Device Management Firmware (TI version of ARM SCP)
  - OPTEE, ATF → ARM standard FWs
- FIT
- Security?

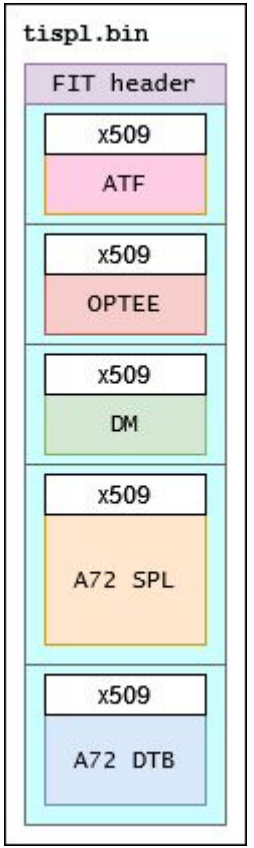


TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A72) Bootloader
  - Device Management Firmware (TI version of ARM SCP)
  - OPTEE, ATF → ARM standard FWs
- FIT
- Security?

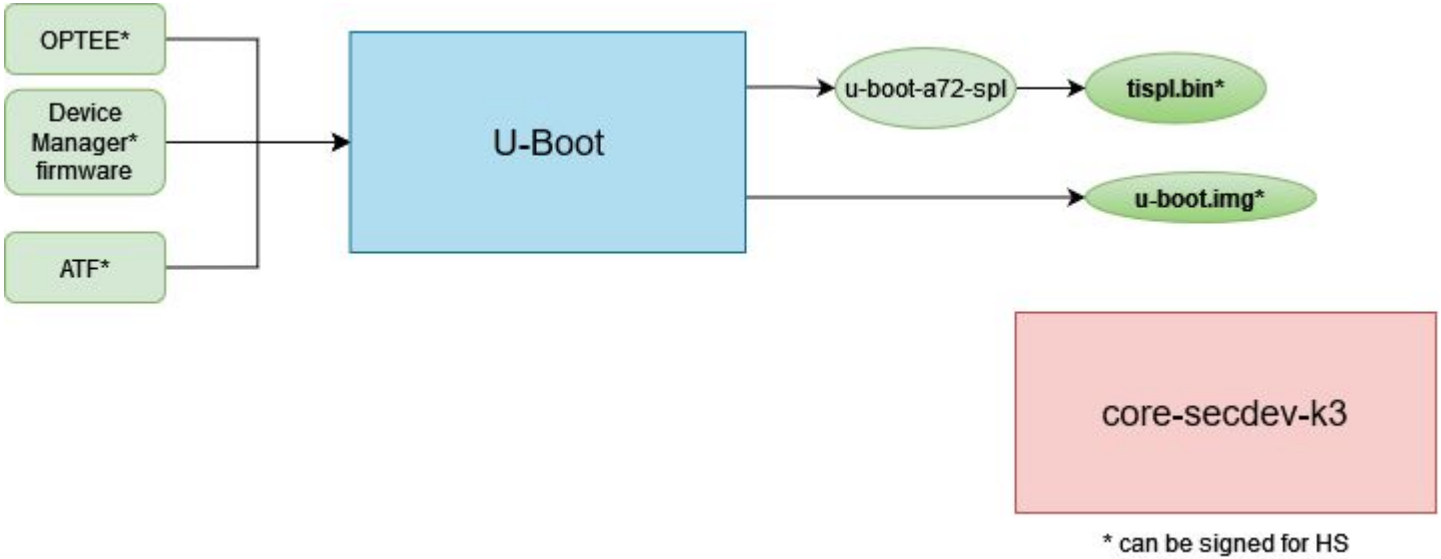
TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A72) Bootloader
  - Device Management Firmware (TI version of ARM SCP)
  - OPTEE, ATF → ARM standard FWs
- FIT
- Security?



Custom scripts

- Complex steps
- core-secdev-k3: signing tools



Custom scripts

- Issues:
  - Maintaining and scaling
    - Boot flow variation
    - Multiple device type (GP vs HS)
  - Moving away from TI signing process is difficult
  - Non std, non-distro friendly
  - No unit level testing

TI K3 Boot-loader binaries: a firmware tangle!

```
# build image

cmd_k3_mkits = \
        $(srctree)/tools/k3_fit_atf.sh \
        $(CONFIG_K3_ATF_LOAD_ADDR) \
        $(patsubst %,$(obj)/dts/%.dtb,$(subst ",,$(LIST_OF_DTB))) > $@

 # Get input file info
HS_SHA_VALUE=$(openssl dgst -sha512 -hex $INPUT_FILE | sed -e "s/^.*= //g")
HS_IMAGE_SIZE=$(cat $INPUT_FILE | wc -c)

# Get software revision info
HS_SWRV=$(cat ${PREFIX}/keys/swrv.txt)

# Parameters to get populated into the x509 template
HS_SED_OPTS="-e s/TEST_IMAGE_LENGTH/${HS_IMAGE_SIZE}/ "
HS_SED_OPTS+="-e s/TEST_IMAGE_SHA_VAL/${HS_SHA_VALUE}/ "
HS_SED_OPTS+="-e s/TEST_SWRV/${HS_SWRV}/ "
TMPX509=$(mktemp) || exit 1
cat ${PREFIX}/templates/x509-template.txt | sed ${HS_SED_OPTS} > ${TMPX509}

# Generate x509 certificate
TMPCERT=$(mktemp) || exit 1

openssl req -new -x509 -key ${PREFIX}/keys/custMpk.pem -nodes -outform DER -out ${TMPCERT} -config ${TMPX509}
-sha512

# Append x509 certificate
cat ${TMPCERT} $INPUT_FILE > $OUTPUT_FILE


#Sign image
cmd_k3secureimg = $(TI_SECURE_DEV_PKG)/scripts/secure-binary-image.sh \
        $< $@ \
        $(if $(KBUILD_VERBOSE:1=), >/dev/null)
```

Why is packaging so hard?

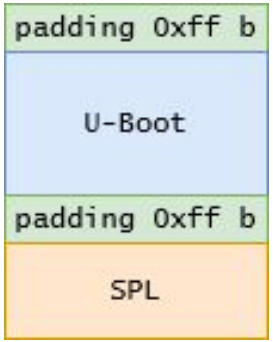
- Collection of inputs
- Dependencies between entries
- Alignment
- Signing
- Run-time discovery of content
- Code/data-size limitations
- Compression

- Formats (FIT, CBFS, FIP)
- Processing time
- SoC-specific tools
- Examining an image / map
- Changing an image later
- Size constraints on parts
- Split over several phases

Binman

- Reimagining the image as **described data**
- Use DT lang for image description

```
binman {
    size = <0x100000>;
    pad-byte = <0xff>;
    u-boot-spl {
    };
    u-boot {
        offset = <0x8000>;
    };
};
```



When does Binman run?

- Initially, as part of the U-Boot build
  - After all inputs have been built
  - It packages the inputs
- Binman can be also be run later
  - With the same inputs and description file
  - Allows signing with different keys
  - Allows setting a firmware ID

Binman - entries

- Images consist of 'entries'
  - Each entry holds a binary or some text or other data
  - Each entry has properties
- Entries are packed one after the other, in order
  - Unless entries have explicit offsets
- Entries cannot overlap

```
// U-Boot SPL entry
u-boot-spl {
};


// U-Boot entry
u-boot {
    offset = <0x8000>;
};
```

Binman - entries

```
class Entry(object):
        """An Entry in the section
        Properties:
            -    section: Section object containing this entry
            -    offset: Offset of entry within the section
            -    size: Entry size in bytes
            -    align_size: Entry size alignment, or None
            -    data: Contents of entry
```

```
class Entry_blob(Entry):
        """Arbitrary binary blob
        Properties:
            -    filename: Filename of file to read into entry
            -    compress: Compression algorithm to use
```

```
class Entry_u_boot_spl(Entry_blob):
        """U-Boot SPL binary
        Properties:
            -    filename: Filename of u-boot-spl.bin (default
                 'spl/u-boot-spl.bin')
```

```
// U-Boot SPL entry
u-boot-spl {
};
```

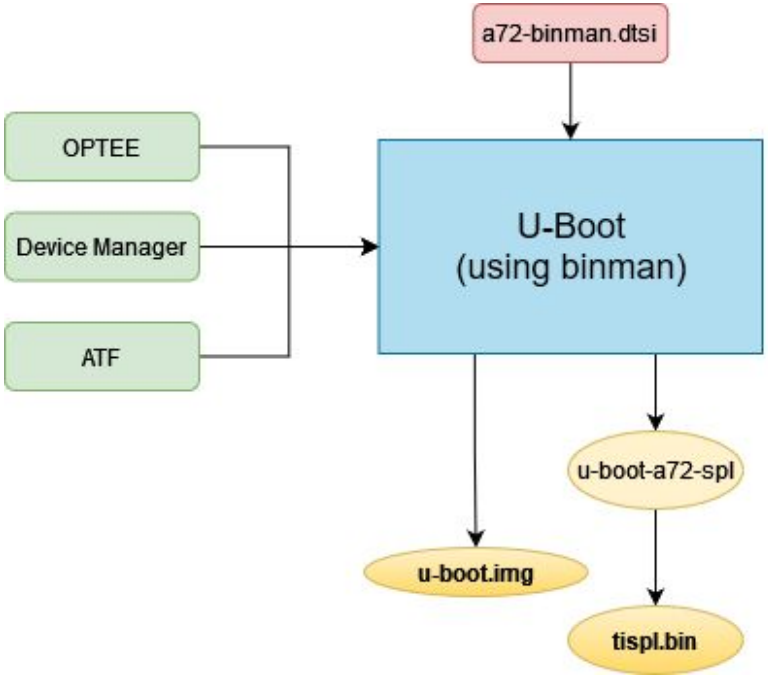
```
class Entry_u_boot(Entry_blob):
        """U-Boot flat binary
        Properties:
            -    filename: Filename of u-boot.bin (default
                 'u-boot.bin')
```

```
// U-Boot entry
u-boot {
        offset = <0x8000>;
};
```

Binman - adding an entry type

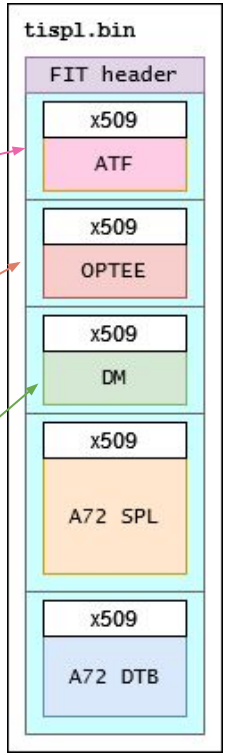
- Add a new ti-secure.py to the etype/ directory
  - Define Entry_ti_secure class
  - Binman will find it
- Entries can run command-line tools
  - tools.Run(...)
- Main logic is in control.py (ProcessImage())
- Code has lots of comments
- Look at other entries for ideas
- Make sure to add a test!

Extending Binman to support K3



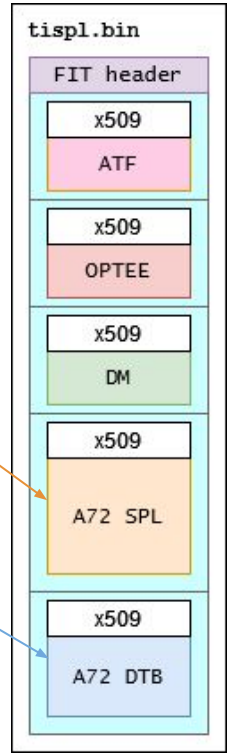
Binman representation - tispl.bin

```
&binman {
        tispl {
                filename = “tispl.bin”;
                pad-byte = <0xff>;
                fit {
                        description = “FIT configuration to load ATF and SPL”;
                        #address-cells = <1>;
                        images {
                                atf {
                                        ti-secure {
                                                content = <&atf>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        atf: atf-bl31 {
                                        };
                                };
                                tee {
                                        ti-secure {
                                                content = <&tee>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        tee: tee-os {
                                        };
                                };
                                dm {
                                        ti-secure {
                                                content = <&dm>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        dm: blob-ext {
                                                filename = “ti-dm.bin”;
                                        };
                                };
```



Binman representation - tispl.bin

```
                fdt-0 {
                        ti-secure {
                                content = <&u_boot_spl_nodtb>;
                                keyfile = “custMpk.pem”;
                        };
                        u_boot_spl_nodtb: u-boot-spl-nodtb {
                        };
                };
                fdt {
                        ti-secure {
                                content = <&spl_dtb>;
                                keyfile = “custMpk.pem”;
                        };
                        spl_dtb: u-boot-spl-dtb {
                        };
                };
        };
    };
};
```



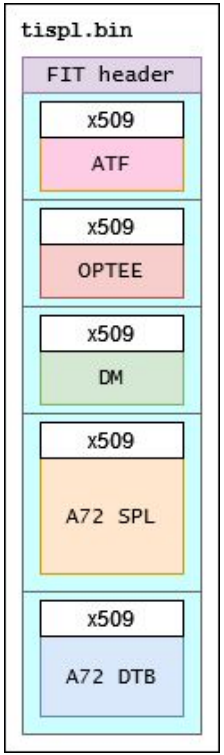
Binman representation - tispl.bin

```
&binman {
        tispl {
                filename = “tispl.bin”;
                pad-byte = <0xff>;
                fit {
                        description = “FIT configuration to load ATF and SPL”;
                        #address-cells = <1>;
                        images {
                                atf {
                                        ti-secure {
                                                content = <&atf>;
                                                keyfile = “custMpk.pem”;
                                                load = <CONFIG_K3_ATF_LOAD_ADDR>;
                                        };
                                        atf: atf-bl31 {
                                        };
                                };
                                tee {
                                        ti-secure {
                                                content = <&tee>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        tee: tee-os {
                                        };
                                };
                                dm {
                                        ti-secure {
                                                content = <&dm>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        dm: blob-ext {
                                                filename = “ti-dm.bin”;
                                        };
```

Custom etype

CONFIG option

Standard etype



ti-secure

```
class Entry_ti_secure(Entry_x509_cert):
      def __init__(self, section, etype, node):
            super().__init__(section, etype, node)
            self.openssl = None
            …
      def ReadNode(self):
            super().ReadNode()
            self.key_fname = self.GetEntryArgsOrProps([
                        EntryArg('keyfile', str)], required=True)[0]
            self.sha = fdt_util.GetInt(self._node, 'sha', 512)
            …
```

ti-secure

```
class Entry_ti_secure(Entry_x509_cert):
      …
      def ObtainContents(self):
            data = self.data
            if data is None:
                  data = self.GetCertificate(False)
            if data is None:
                  return False
            self.SetContents(data)
            return True

    def ProcessContents(self):
            # The blob may have changed due to WriteSymbols()
            data = self.data
            return self.ProcessContentsUpdate(data)
```

ti-secure

```
class Entry_ti_secure_rom(Entry_x509_cert):
    …
    def AddBintools(self, btools):
        super().AddBintools(btools)
        self.openssl = self.AddBintool(btools, 'openssl')
```

Future developments

- Binman DT node is not part of DT specification
  - This potentially is long term maintenance hurdle
- Ability to parse custom firmware paths via CLI argument
  - Today hardcoded in binman dts files
- x509 certificate template generation

Extending to other devices

- Current boards using custom scripts that can benefit from Binman:
  - DragonBoard 410c
  - imx8/9 (imx*_image.sh)
  - See the **tools/ folder** of U-Boot source for more such boards

References

- OSFC 2019 Binman Talk - Simon Glass
- K3 Migration to using Binman patch series
- U-Boot Open Source Project
- U-Boot Documentation - Binman
- Open Source Summit Europe 2022 Bootloaders-101 - Bryan Brattlof

Credits and Acknowledgments

- FOSDEM’24 organizers
- Texas Instruments
- U-Boot community working on Binman ;)

Q&A

- Contact information:
  - Vignesh Raghavendra <vigneshr@ti.com>
  - Neha Malcom Francis <n-francis@ti.com>
  - Simon Glass <sjg@chromium.com>
- Also on IRC @ libera.chat #u-boot #linux-ti

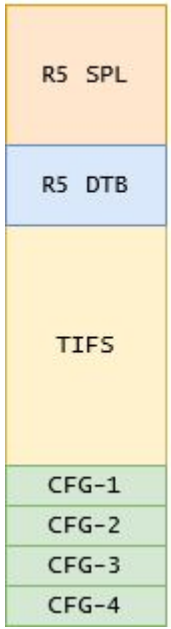
Thank you!

Bonus stuff

- R5 boot-loader image
- New in Binman - Templating
- Binman - Runtime symbol updation

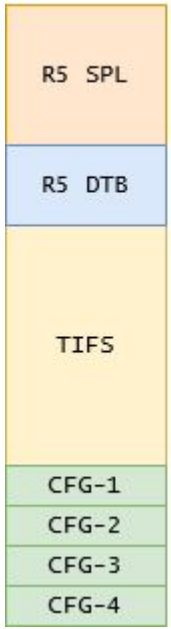
TI K3 Boot-loader binaries: a firmware tangle!

- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries



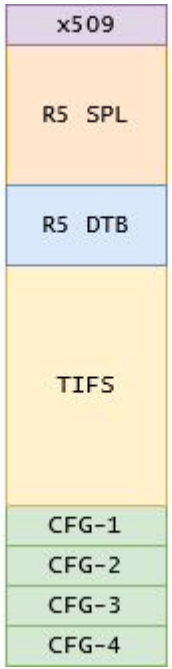
TI K3 Boot-loader binaries: a firmware tangle!

- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries
- Security?



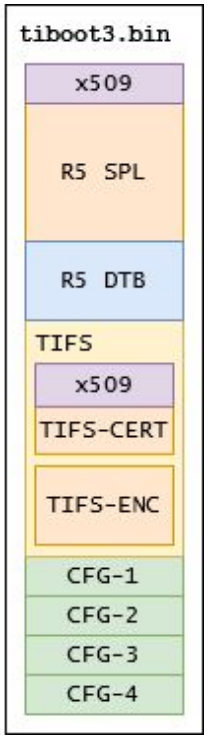
TI K3 Boot-loader binaries: a firmware tangle!

- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries
- Security?



TI K3 Boot-loader binaries: a firmware tangle!

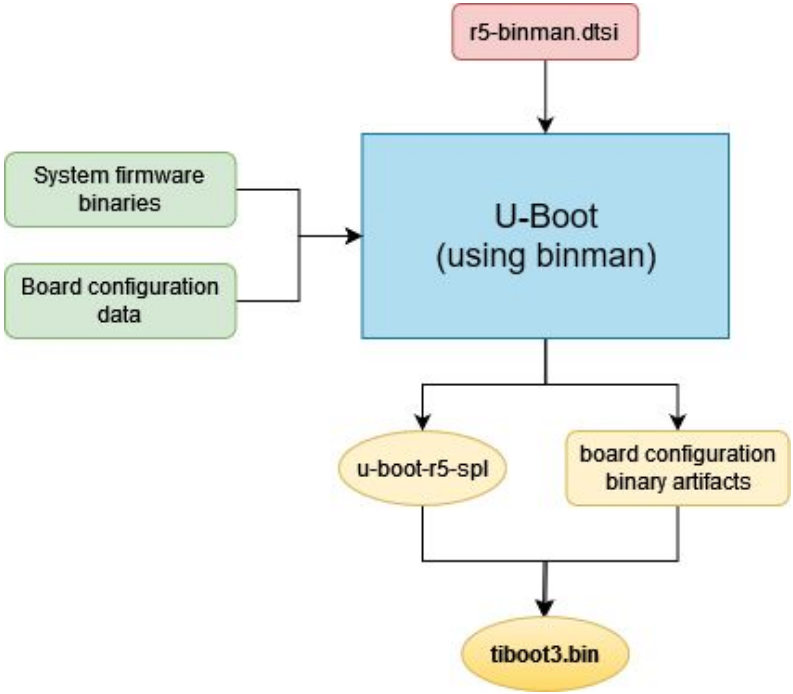
- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries
- Security? More?



TI K3 Boot-loader binaries: a firmware tangle!

```
...
ifneq ($(SOC_TYPE),gp)
$(SYSFW_HS_CERTS_PATH): $(SYSFW_HS_INNER_CERT_PATH)
	@echo "Signing the SYSFW inner certificate with $(KEY) key...";
	./gen_x509_cert.sh -d -c m3 -b $< -o $@ -l $(LOADADDR) -k $(KEY) -r $(SW_REV);

$(soc_objroot)/sysfw.bin-$(SOC_TYPE): $(SYSFW_HS_CERTS_PATH) $(SYSFW_PATH) | _objtree_build
	cat $^ > $@
else
$(soc_objroot)/sysfw.bin-$(SOC_TYPE): $(SYSFW_PATH) | _objtree_build
	@echo "Signing the SYSFW release image with $(KEY) key...";
	./gen_x509_cert.sh -c m3 -b $< -o $@ -l $(LOADADDR) -k $(KEY) -r $(SW_REV);
endif

$(ITS): | _objtree_build
	./gen_its.sh $(SOC) $(SOC_TYPE) $(CONFIG) $(SOC_BINS) > $@

$(ITB): $(ITS) $(SOC_BINS) | _bindir_build
	$(MKIMAGE) -f $< -r $@

.PHONY: sysfw.itb
sysfw.itb: $(ITB)
	@ln -sf $< $(BIN_DIR)/$@

$(COMBINED_SYSFW_BRDCFG): $(soc_objroot)/board-cfg.bin $(soc_objroot)/sec-cfg.bin $(soc_objroot)/pm-cfg.bin
$(soc_objroot)/rm-cfg.bin
	python3 ./scripts/sysfw_boardcfg_blob_creator.py -b $(soc_objroot)/board-cfg.bin -s
$(soc_objroot)/sec-cfg.bin -p $(soc_objroot)/pm-cfg.bin -r $(soc_objroot)/rm-cfg.bin -o $@

...
```

Extending Binman to support K3



New in Binman - Templates

- Common part in multiple images
- e.g. FIT with ATF, OPTEE etc.
- Reduce code redundancy

New in Binman - Templates

```
binman {
        u-boot.img {
                filename = “u-boot.img”;
                pad-byte = <0xff>;

                fit {
                        description = “U-Boot image”;

                        images {

                                uboot {

                                        type = “firmware”;
                                        os = “u-boot”;
                                        arch = “arm”;
                                        compression = “none”;
                                        load = <CONFIG_TEXT_BASE>;
                                        ti-secure {
                                                content = <&u_boot_nogtb>;
                                                Keyfile = “custMpk.pem”;
                                        };
                                        U_boot_nodtb: u-boot-nodtb {
                                        };
                                        Hash {
                                                Algoo = “crc32”;
                                        };
                                };

                                fdt-0 {
                                        description = “foo-board”;
                                        …
```

foo-board-binman.dtsi

New in Binman - Templates

```
binman {
        uboot_template: u-boot.img {
                filename = “u-boot.img”;
                pad-byte = <0xff>;

                fit {
                        description = “U-Boot image”;

                        images {

                                uboot {

                                        type = “firmware”;
                                        os = “u-boot”;
                                        arch = “arm”;
                                        compression = “none”;
                                        load = <CONFIG_TEXT_BASE>;
                                        ti-secure {
                                                content = <&u_boot_nodtb>;
                                                Keyfile = “custMpk.pem”;
                                        };
                                        u_boot_nodtb: u-boot-nodtb {
                                        };
                                        Hash {
                                                Algoo = “crc32”;
                                        };
```

common.dtsi

New in Binman - Templates

```
#include “common.dtsi”
…
&binman {
      u-boot {
            insert-template = <&uboot_template>;
            fit {
                  images {
                        fdt-0 {
                              description = “foo-board”;
                              …
                        };
                  };
            };
      };
};
```

foo-board-binman.dtsi

Binman - Runtime symbol updation

```
binman_sym_declare(ulong, u_boot, image_pos);
...
void spl_set_header_raw_uboot(struct spl_image_info *spl_image)
{
      ulong u_boot_pos = binman_sym(ulong, u_boot_any, image_pos);
      spl_image->size = CONFIG_SYS_MONITOR_LEN;
      if (u_boot_pos && u_boot_pos != BINMAN_SYM_MISSING) {
            spl_image->entry_point = u_boot_pos;
            spl_image->load_addr = u_boot_pos;
      } else {
            spl_image->entry_point = CONFIG_SYS_UBOOT_START;
            spl_image->load_addr = CONFIG_SYS_TEXT_BASE;
      }
      spl_image->os = IH_OS_U_BOOT;
      spl_image->name = "U-Boot";
      ...
```