LEDGER

From dusk till dawn
Toward an effective trusted UI

SSTIC 2023 - Patrice HAMEAU, Philippe THIERRY, Florent VALETTE

About Trusted UI

- A **Trusted UI** (Trusted User Interface aka. TUI) is
  - A trusted HW+SW path
  - Used in order to allow a secure environment (a smartcard, an secure administration control system, or any security-sensitive element) to communicate with the user
  - Through or beside an unsecure path

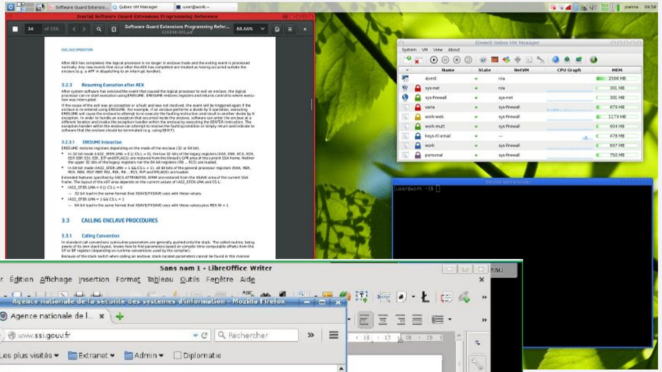
- It shall keep confidentiality, integrity, disponibility and imputability of the data it manipulates

- TUI implementation problematic is a very old need
  - Requiring **user presence**
  - Enforcing only **user-initiated** operation
  - Requiring **authentication** mechanism
    - something you know (PIN, passphrase...)
    - something you own (tag...)
    - who you are (fingerprinting...)
  - Providing **secured acknowledging** of the authentication sequence and secure operations

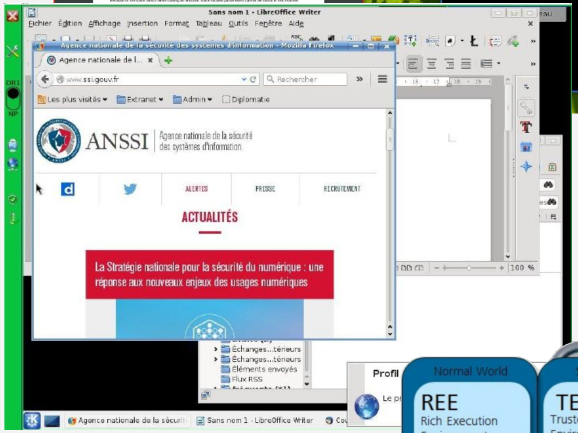
- TUI is required in various technical fields for different degrees of trust:
  - Applications (payment, ...) on mobile devices,
  - Credentials for administrative tasks,
  - Access control on workstations...

- In consumer electronics, it is mostly designed with a centralized execution model:
  - **Single** Application Processor (AP)
    - For both the normal/unsecured and secure worlds
    - Using virtualization or TEE for isolating both worlds
  - **Sharing** the peripherals
    - Peripherals dynamic switch (TEE)
    - Hardware virtualization

- In general TUI is hard to make **portable on different hardware**
  - Highly linked to specificities of used architecture technology (Virtualization architecture, TrustZone...)
  - Highly coupled to the (un)secure interface sources
  - Sensitive to other peripherals in the system (side channels...)



QubesOS



ClipOS

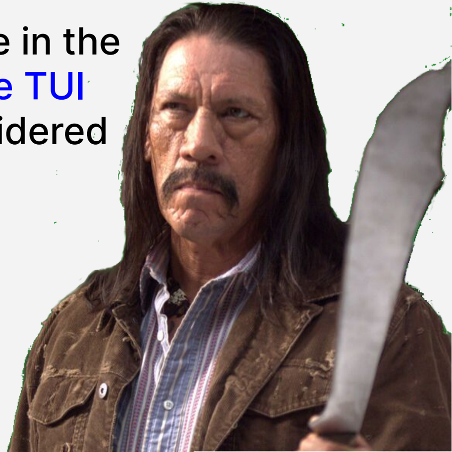
Mobiles (GlobalPlatform)

- Some consortiums (e.g. GlobalPlatform) have tried to address the requirements for providing trustful and high security TUI, resistant to different kind of attacks.
- In consumer electronics, the TUI architectures are imposed by:
  - The limited choices among SoC manufacturers, which provide similar architectures (mainly ARM(R) based)
  - And also mostly built in regards of consumer application needs, that is to:
    - **Counter logical attacks**: privilege escalation, data corruption, …
    - … but less to **resist to semi-invasive and side-channels attacks**!
    - Sharing power lines termination, clocks, memory hierarchy, and cpu cores with the unsecure domain imposes indeed **blockers** in the design of security architecture and thus on attacks path resistance.

- Some consortiums (e.g. GlobalPlatform) have tried to address the requirements for providing trustful and high security TUI, resistant to different kind of attacks.
- In consumer electronics, the TUI architectures are imposed by:
  - The limited choices among SoC manufacturers, which provide similar architectures (mainly ARM(R) based)
  - And also mostly built in regards of consumer application needs, that is to:
    - **Counter logical attacks**: privilege escalation, data corruption, …
    - … but less to **resist to semi-invasive and side-channels attacks**!
    - Sharing power lines termination, clocks, memory hierarchy, and cpu cores with the unsecure domain imposes indeed **blockers** in the design of security architecture and thus on attacks path resistance.

- Some consortiums (e.g. GlobalPlatform) have tried to address the requirements for providing trustful and high security TUI, resistant to different kind of attacks.
- In consumer electronics, the TUI architectures are imposed by:
  - The <u>limited choices among SoC</u> manufacturers, which provide similar architectures (mainly ARM(R) based)
  - And also mostly built in regards of <u>consumer application needs</u>, that is to:
    - **Counter logical attacks**: privilege escalation, data corruption, ...
    - ... but less to **resist to semi-invasive and side-channels attacks**!
    - Sharing power lines termination, clocks, memory hierarchy, and cpu cores with the unsecure domain imposes indeed **blockers** in the design of security architecture and thus on attacks path resistance.
- In the embedded (and industrial) markets, more choices are possible in the hardware and its architecture: considering a alternative and reusable TUI security architecture concepts is a feasible option that may be considered

In embedded systems, why not just....
move the global input/output data and control plane to an isolated trusted hardware component dedicated only to this task?



Extracting the graphic chain

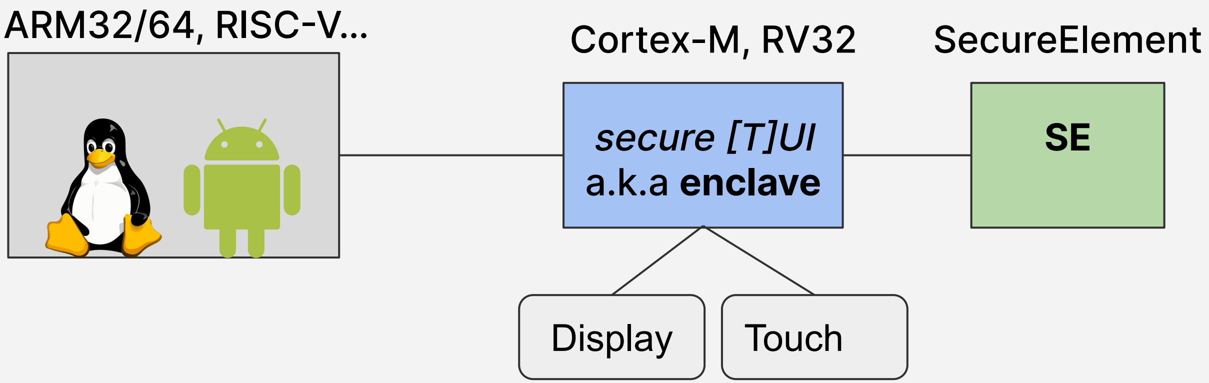
General principles

Let's make an arch & OS independent TUI mechanism

- Display interface technologies are based on standard protocols and encodings
  - SPI buses and MIPI-DSI bridges, pixel encoding formats (RGBA888, ARGB32...)
  - pixel format support negotiation already exist in standard protocols
- Display input sources (touchscreen, keyboards) are easy to intercept
  - standard 'slow' peripherals (I2C...) with simple protocol, interrupt based

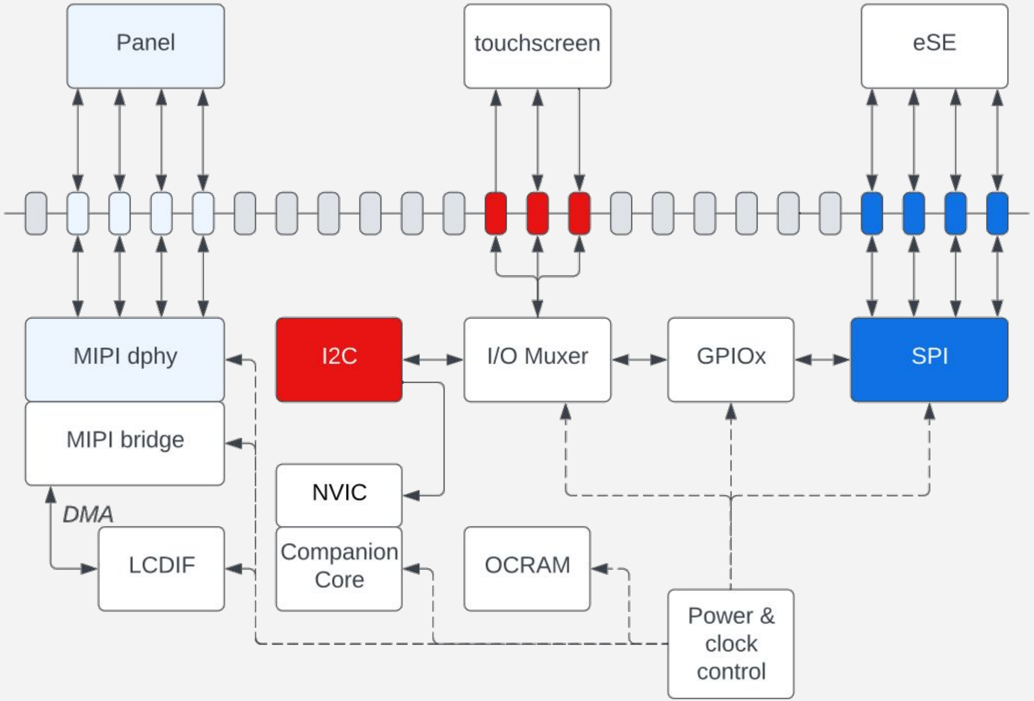
Let’s make an arch & OS independent TUI mechanism

- Display interface technologies are based on standard protocols and encodings
  - SPI buses and MIPI-DSI bridges, pixel encoding formats (RGBA888, ARGB32…)
  - pixel format support negotiation already exist in standard protocols
- Display input sources (touchscreen, keyboards) are easy to intercept
  - standard ‘slow’ peripherals (I2C…) with simple protocol, interrupt based
- For both of them:
  - **para-virtualization through a deported (even SoC-external) graphical controller with TUI capacity**



Let’s dive in reality: the i.MX 8 case

- The **enclave** is hosted in the SoC, in an isolated *companion core*, but…

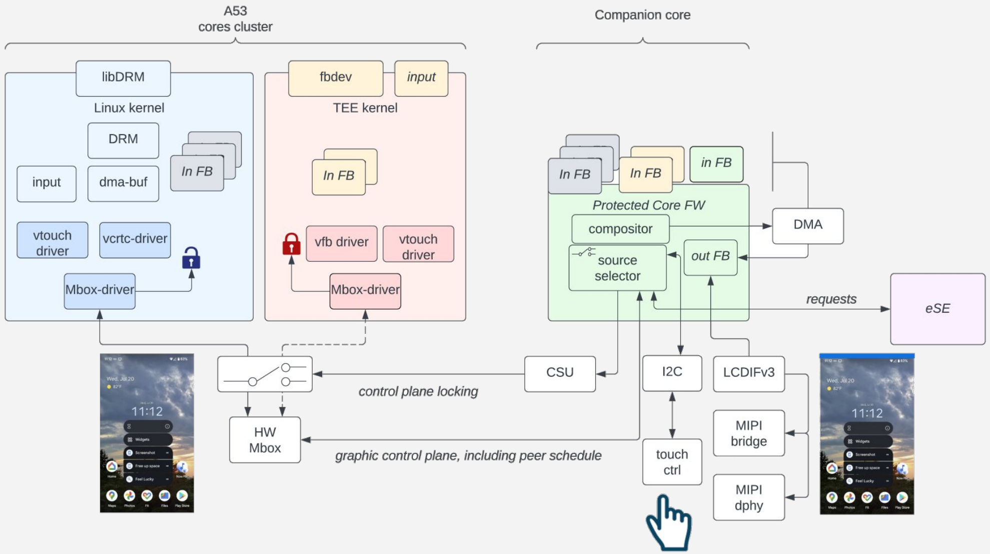


- A lot of indirect hardware elements impacting the global security also need to be virtualized
  - GPIO controller, I/O muxer, overall power and clock management
  - companion-core dedicated memory (ITCM/DTCM)

- All the SoC security components need to be locked and controlled by the enclave too

Let's dive in reality: the i.MX 8 case

- The control interface between unsecure worlds (Android, TEE) and the **enclave**
  - is reduced to a 4-registers set mailbox
  - has its access scheduled by the TUI enclave
  - is a medium for a basic protocol
  - use lightweight authenticated session-based principle



- Enclave manipulates its own, dedicated, framebuffers (FB) set for secure UI

- Overall layouting (framebuffer mapping, device assignation, etc.) is specified at build time

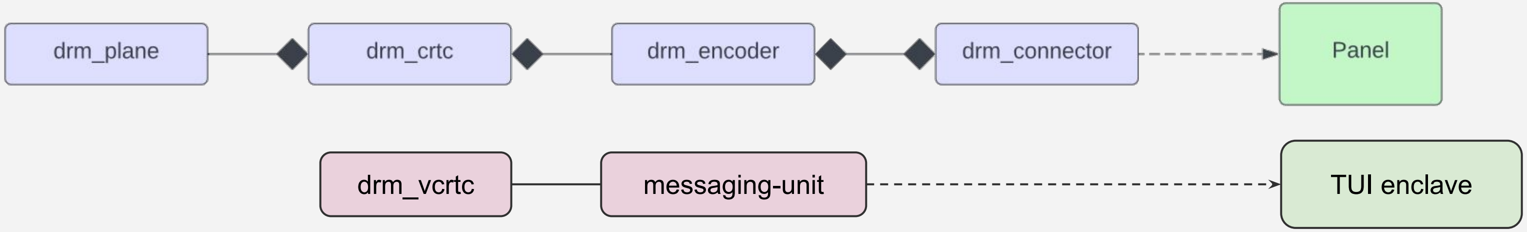
Para-virtualizing the Application Processor OSes

Para-virtualizing the output: Linux DRM to remote mailboxing

- Linux kernel has defined a standard graphical stack denoted DRM
    - all graphical drivers should declare themselves against the DRM framework
    - this allows a unified userspace interface to GPU rendering libraries

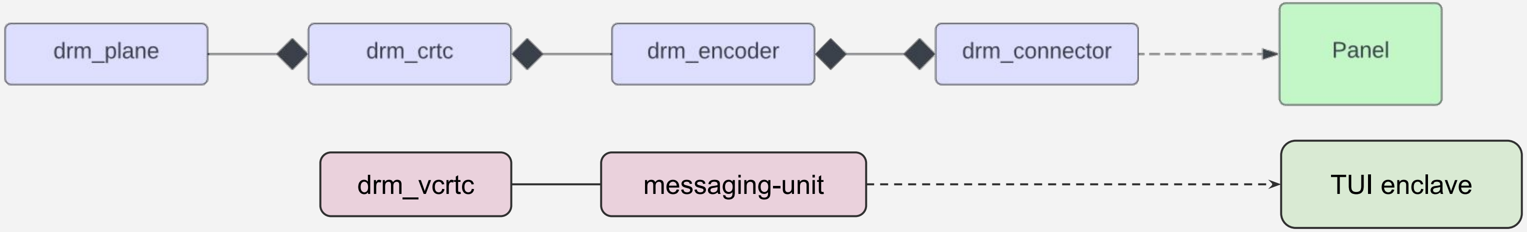
Para-virtualizing the output: Linux DRM to remote mailboxing

- Linux kernel has defined a standard graphical stack denoted DRM
  - all graphical drivers should declare themselves against the DRM framework
  - this allows a unified userspace interface to GPU rendering libraries



Para-virtualizing the output: Linux DRM to remote mailboxing

- Linux kernel has defined a standard graphical stack denoted DRM
  - all graphical drivers should declare themselves against the DRM framework
  - this allows a unified userspace interface to GPU rendering libraries



Para-virtualizing the input: standard Linux input device

- Linux input device framework is kept untouched. Easier to virtualize as only touch device interrupts need to be emulated through the messaging unit
- Any hotplugged input device (e.g. USB keyboard) is then only AP-dedicated input, unusable by the TUI

Switching to TUI mode : TUI session handling

- The TUI content being under the control of the Secure Element (SE), it is the sole master of the TUI session startup and releasing, when :
  - User authentication is requested
  - Specific user interactions with confidentiality/integrity/authenticity is requested
  - SE-specific UI control interface is required

- To enforce TUI contents isolation and protection, during the TUI session:
  - The enclave **ignores any graphical request** from other sources
  - The enclave **emulates hardware acknowledgement** toward unsecure sources as if the requested content have been displayed, even if discarded
  - The **touch display events are directed to the SE** (SoC has no access to them)

Securing the TUI firmware

Booting and protecting the TUI software

- The **enclave** behaves as a transparent graphic proxy, and must be started first
- Its boot sequence is controlled by the I.MX8 secureboot bootrom + SPL (Secondary Platform Loader), and:
  - is started **before** ATF, TEE, Android, etc. to guarantee very minimal TCB
  - is **ready in milliseconds**, even in a MCU, due to its very small footprint (~15KLOC)
  - on I.MX 8 its **integrity** is controlled by the **HAB secure boot** process, using the Boot-ROM startup check

Booting and protecting the TUI software

- The **enclave** immediately performs the following action when starting:
  - security domain controller, separating proxy domain from main compute node (and associated peripherals) domain
  - takes full control on the IOMUX, power and clock controllers (CCM, mediablock controllers, etc.) to hold an lock its own lines
  - initializes the graphical subsystem
  - initializes communication channel with the eSE
  - release hardware semaphore to acknowledge SPL for continuing AP boot sequence
  - … and wait for events in proxy mode

Demo time!

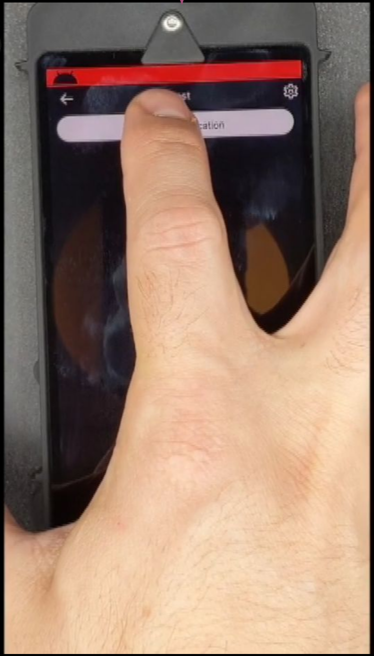
Trusted User Interface PoC demo - a video CLIP

Normal Android mode (non TUI)

Frame Buffer generated by Android
(captured with ADB)

Effective screen display managed by Enclave coprocessor (CM7), composed of:
- Frame Buffer generated by Android (as per left picture)
- Security bar (red) added by CM7

Display generated by Android



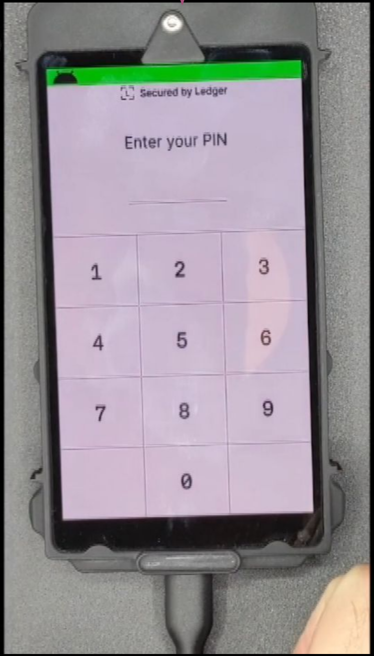
Effective display on screen

TUI mode (secure)

In TUI mode, no change in frame Buffer generated by Android, and Android is not aware that it is not displayed on the screen
(captured with ADB)

Effective screen display once in TUI, composed of:
- TUI rendered by CM7
- Security bar (green) added by CM7

Display generated by Android



Effective display on screen

Demo done on a Ledger development box (NXP iMX8; 720p display; OS Android; TUI on Cortex-M7 co-processor)

What’s next?

What's next...

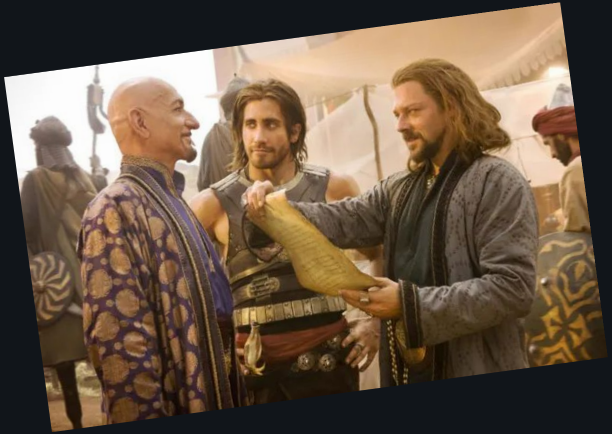
- **Our Proof Of Concept includes**
  - Linux kernel paravirtualized drivers fully developed, integrated as a DRM device, and operational in Android system
  - An enclave firmware (running on Cortex-M7) built from scratch with security and portability in mind

- **And now what?**
  - Continue to improve the enclave firmware implementation to be resilient and as much portable as possible:
    - Minimize dependency to the main AP architecture (ARM64, RISC-V…) and OS running on it (GNU/Linux, Android…)
    - Ease portability from ARMv7 to other architecture (e.g. RISC-V)
    - Increase as much as possible build-time (static) resources allocation and peripheral configurations (locking) versus runtime (dynamic) ones
  - Support architecture evolution to an external enclave coprocessor (e.g. a Secure Element companion outside of the SoC) that acts as a proxy (bridge) between the SoC and the display / touch
  - Open-source the design and firmware and maintain it as an open-source project (part of Ledger open-source plan)

Thank you !

Extra slides

Console boot extract



Build-time set memory layout: simplify the security domain configuration

```
[    0.000000][    T0] Reserved memory: created DMA memory pool at 0x0000000050000000, size 4 MiB
[    0.000000][    T0] OF: reserved mem: initialized node framebuffer@50000000, compatible id
shared-dma-pool
[    0.000000][    T0] Reserved memory: created DMA memory pool at 0x0000000050400000, size 4 MiB
[    0.000000][    T0] OF: reserved mem: initialized node framebuffer@50400000, compatible id
shared-dma-pool
[    0.000000][    T0] Reserved memory: created DMA memory pool at 0x0000000050800000, size 4 MiB
[    0.000000][    T0] OF: reserved mem: initialized node framebuffer@50800000, compatible id
shared-dma-pool
[...]
[    3.872296][    T1] init: Loading module /lib/modules/libmu.ko with args ""
[    3.880790][    T1] libmu initialized with success.
[    3.885930][    T1] init: Loaded kernel module /lib/modules/libmu.ko
[    3.892385][    T1] init: Loading module /lib/modules/cm7drm.ko with args ""
[    3.901070][    T1] cm7-drm cm7-drm: probe begin
[    3.905868][    T1]  cm7-drm-framebuffer-0: assigned reserved memory node framebuffer@50000000
[    3.914537][    T1]  cm7-drm-framebuffer-1: assigned reserved memory node framebuffer@50400000
[    3.923184][    T1]  cm7-drm-framebuffer-2: assigned reserved memory node framebuffer@50800000
[    3.932536][    T1] cm7-drm cm7-drm: cm7-plane: init
[    3.937559][    T1] cm7-drm cm7-drm: init begin
[    3.942299][    T1] [drm] Initialized cm7-drm 1.0.0 20220916 for cm7-drm on minor 0
[    3.950107][    T1] init: Loaded kernel module /lib/modules/cm7drm.ko
[    3.956667][    T1] init: Loading module /lib/modules/libmu-core.ko with args ""
[    3.978321][    T1] libmu-core initialized with success.
[    3.978328][    C0] libmu-core: ping received, ree is taking ownership of mu endpoints
[    3.983654][    C0] libmu-core: ping received, ree is taking ownership of mu endpoints
[    3.983744][    T1] libmu-core: libmu-core driver and sysctl registered.
[    4.006284][    T1] init: Loaded kernel module /lib/modules/libmu-core.ko
```

boot console

AP / Enclave library providing hardware isolated communication setup

Frame buffer statically allocated at absolute address.

'dtsi' file

```
&resmem {
        nwd_framebuffer_1: framebuffer@50000000 {
                compatible = "shared-dma-pool";
                reg = <0 0x50000000 0 0x400000>;
                no-map;
        };

        nwd_framebuffer_2: framebuffer@50400000 {
                compatible = "shared-dma-pool";
                reg = <0 0x50400000 0 0x400000>;
                no-map;
        };

        nwd_framebuffer_3: framebuffer@50800000 {
                compatible = "shared-dma-pool";
                reg = <0 0x50800000 0 0x400000>;
                no-map;
        };
};

&mu {
 compatible = "ledger,libmu";

 memory-region =
   <&nwd_framebuffer_1>,
   <&nwd_framebuffer_2>,
   <&nwd_framebuffer_3>;

 memory-region-names = "framebuffer1",
   "framebuffer2", "framebuffer3";

 status = "okay";
};
```

Boot sequence extract