Devicetree State of the Union

Linaro Connect
VIRTUAL • SPRING 2021

Agenda:

- Introduction
- System Devicetree in context, why we need it:
  - Andrei Kholodnyi, Wind River, Principal Technologist, ROS2 TSC
- System Devicetree
  - Stefano Stabellini, Xilinx, Principal Engineer
- Devicetree in Zephyr and other RTOS
  - Kumar Gala, Linaro IOT and Embedded Group Technical Lead
- Devicetree in Standardized Linux & other OS boot
  - Bill Mills, Linaro, Principal Technical Consultant
- Conclusion

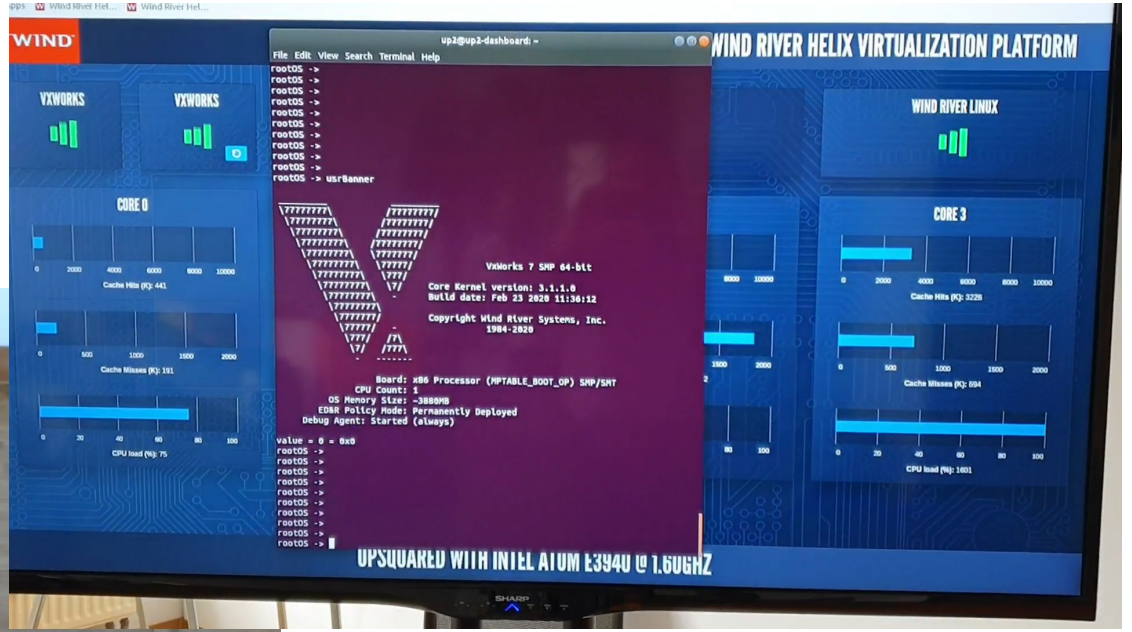
Introduction:

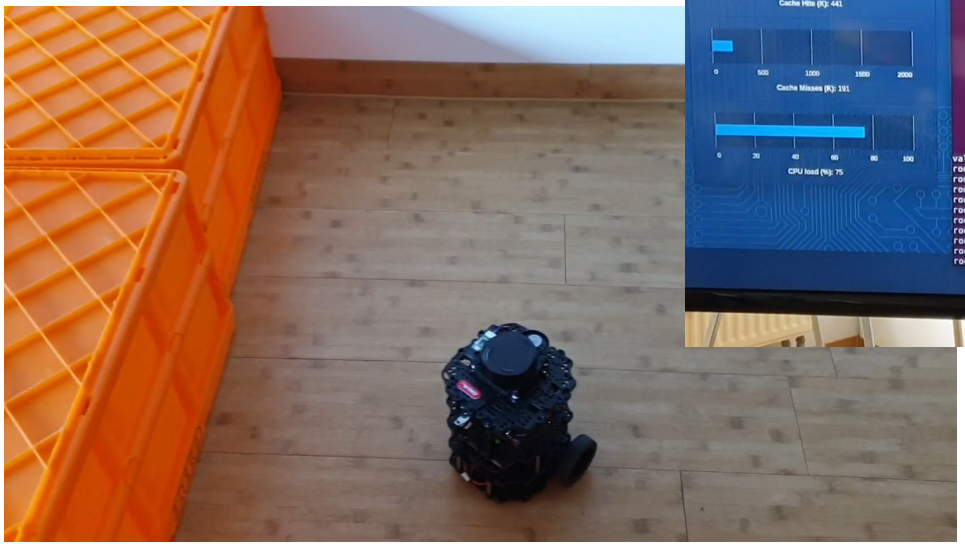
- What is Devicetree?
  - A structured data representation that describes SOC and board hardware
  - Started in the powerpc architecture, used for Arm in Linux since v3.1 in 2011
  - Supports many architectures:
    - arm, arm64, riscv, powerpc, mips, x86, microblaze, m68k, sh, arc, xtensa, nios, etc
  - Used in lots of projects:
    - Linux, U-Boot, TF-A, Xen, BSDs, Zephyr, VxWorks, Nucleus
- What is System Devicetree?
  - Describes the HW properties of the full system and how resources are allocated to different execution domains (e.g. Linux, XEN, RTOS)
- But ARM Server's use ACPI, why not just use that?
  - If ACPI meets your needs, use it
  - Devicetree is data only, ACPI is data + byte code. This kernel code remains in charge.
  - Devicetree is more flexible and nimble (blessing and a curse)
  - Devicetree can describe situations that ACPI cannot today
- Device Tree remains relevant today and in to the future
  - SystemReady IR is devicetree based to match current SOC practice
  - EBBR standardizes the boot flow for DT based systems

Need for System Devicetree

Linaro Connect
VIRTUAL • SPRING 2021

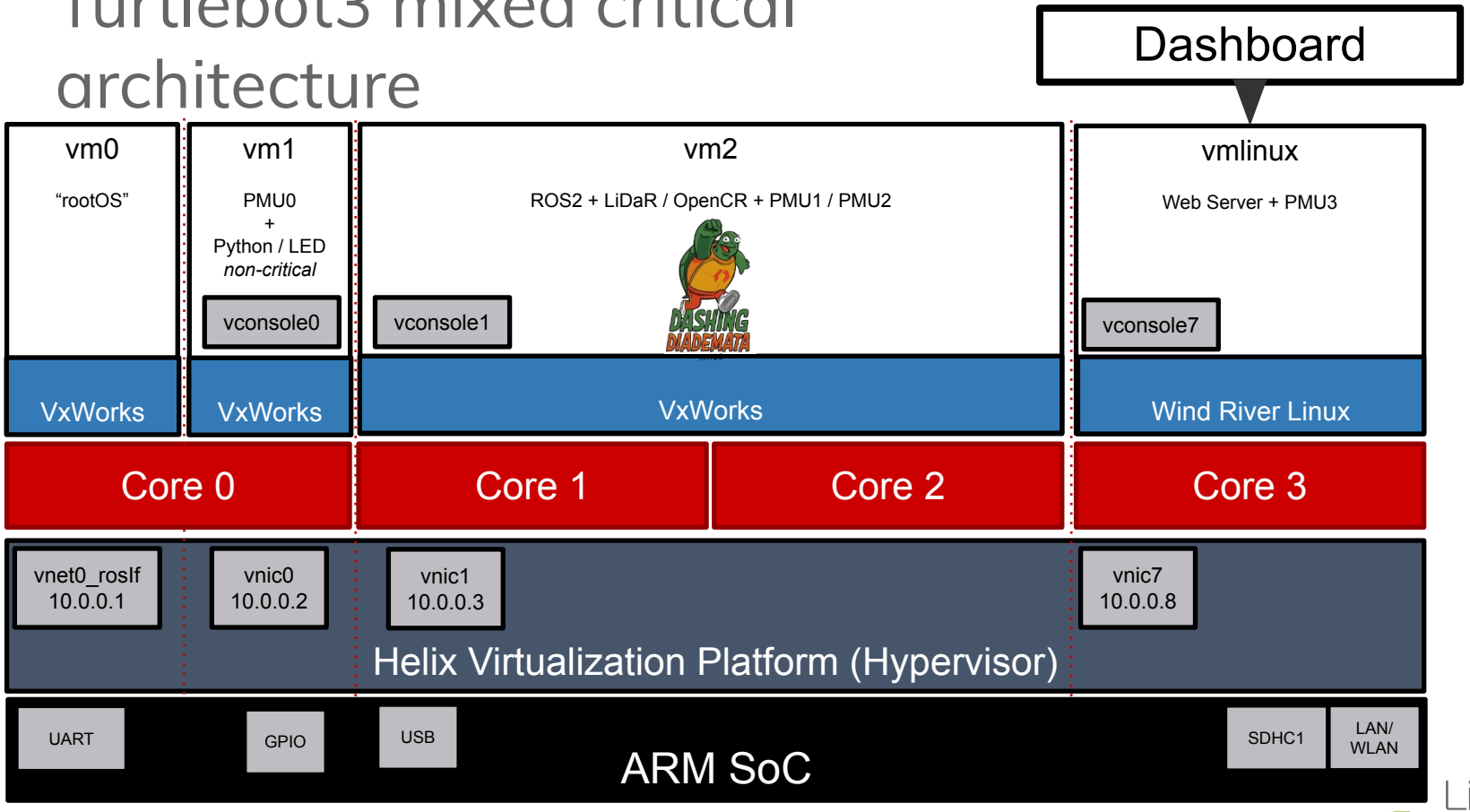
Example System Video





Linaro Connect VIRTUAL • SPRING 2021

Turtlebot3 mixed critical architecture



Challenges to configure system with a hypervisor

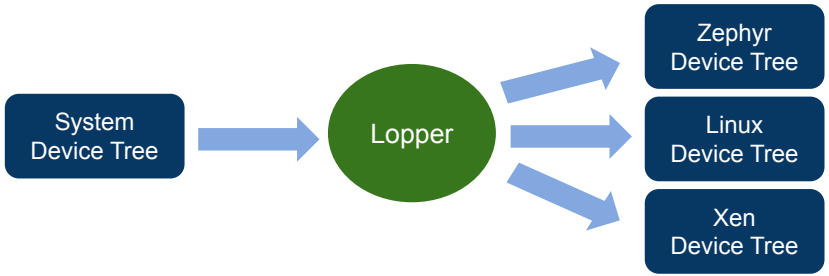
- System configuration is spreaded via different instances (hypervisor + RTOS + Linux)
  - CPUs
  - RAM
  - Network
  - Peripherals
- Configuration formats are not compatible
  - Proprietary Device Tree for the the hypervisor
  - VxWorks Device Tree (other RTOSes don‘t even have DT and values are hardcoded)
  - Linux Device Tree
- Difficult to check for consistency
- Modifications are needed in several places
- There are also AMP systems (e.g. with R5)

System Devicetree

Linaro Connect
VIRTUAL • SPRING 2021

System Device Tree

- Multi-View Multi-OS hardware description
  - *cpus,clusters*
  - *address-map*
  - *indirect-bus*
- AMP Configuration
  - execution domains (*openamp,domain*)
  - unique access to hardware resource (*access*)
- Lopper: prunes a S-DT down to traditional device trees




```
cpus_r5: cpus-cluster@0 {
        compatible = "cpus,cluster";

        address-map =
                <0xf1000000 &amba 0xf1000000 0xeb00000>,
                <0xf9000000 &amba_rpu 0xf9000000 0x10000>,
                <0x0 &memory 0x0 0x80000000>,
                <0x0 &tcm 0xFFE90000 0x10000>;

        cpu@0 {
        };

        cpu@1 {
        };
};


domains {
        domain@0 {
                compatible = "openamp,domain-v1";

                cpus = <&cpus_r5 0x3 0x80000001>;
                memory = <0x0 0x100000 0x0 0x400000>;

                access = <&can0 0x80000000>;
        };
```

OpenAMP RemoteProc

- Easier system-wide configuration in S-DT
  - dedicated resources (*access*)
  - shared resources (*include*)
- Lopper generates RemoteProc bindings from S-DT information
- Compatible with existing RemoteProc Bindings
  - No changes to existing software required

```
resource_group: resource_group@0 {
        compatible = "openamp,remoteproc-v1";
        memory = <0x0 0x3ed40000 0x0 0x4000
                  0x0 0x3ed44000 0x0 0x4000
                  0x0 0x3ed48000 0x0 0x100000
                  0x0 0x3ed00000 0x0 0x40000>;
        access = <&tcm_0_a 0x0>;
};

openamp_a53 {
        compatible = "openamp,domain-v1";
        memory = <0x0 0x80000000 0x0 0x78000000
                  0x8 0x0 0x0 0x80000000>;
        cpus = <&cpus_a53 0xf 0x2>;

        access = <&ipi_mailbox_rpu0 0x13>;
        include = <&resource_group 0x1>;
};

openamp_r5 {
        compatible = "openamp,domain-v1";
        memory = <0x0 0x0 0x0 0x20000000>;
        cpus = <&cpus_r5 0x2 0x80000000>;

        include = <&resource_group 0x0>;
};
```

Traditional RemoteProc Device Tree Output

```
r5fss@ff9a0000 {
        compatible = "xlnx,zynqmp-r5-remoteproc";
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;
        reg = <0x0 0xff9a0000 0x0 0x10000>;
        xlnx,cluster-mode = <0>;

        r5f_0 {
                compatible = "xilinx,r5f";
                memory-region = <&elf_load0>,
                                <&rpu0vdev0vring0>,
                                <&rpu0vdev0vring1>,
                                <&rpu0vdev0buffer>;
                sram = <&tcm_0a>, <&tcm_0b>;
                mboxes = <&ipi_mailbox_rpu0 0x0 &ipi_mailbox_rpu0 0x1>;
                mbox-names = "tx", "rx";
                power-domain = <0x7>;
        };
};
```

```
zynqmp_ipi1 {
        compatible = "xlnx,zynqmp-ipi-mailbox";
        interrupt-parent = <&gic>;
        interrupts = <0 33 4>;
        xlnx,ipi-id = <5>;
        #address-cells = <1>;
        #size-cells = <0>;

        ipi_mailbox_rpu0: mailbox@ff990600 {
                reg = <0xff990600 0x20>,
                        <0xff990620 0x20>,
                        <0xff9900c0 0x20>,
                        <0xff9900e0 0x20>;
                reg-names = "local_request_region",
                "local_response_region",
                "remote_request_region",
                "remote_response_region";
                #mbox-cells = <1>;
                xlnx,ipi-id = <3>;
        };
```

System Device Tree Example

Full System Device Tree example here, including:

- full hardware description of a Xilinx Versal board
- multiple domains

```
domains {

        #address-cells = <0x2>;
        #size-cells = <0x2>;

        linux: domain@0 {
                compatible = "openamp,domain-v1";
                cpus = <&cpus_a72 0x3 0x00000001>;
                memory = <0x0 0x500000 0x0 0x7fb00000>;

                access = <&mmc0 0x0 &ipi_mailbox_rpu0 0x13>;
                include = <&resource_group_0 0x1>;

                /* 1: block */
                firewallconfig-default = <1 0>;
        };

        zephyr: domain@1 {
                compatible = "openamp,domain-v1";
                cpus = <&cpus_r5 0x3 0x80000001>
                memory = <0x0 0x100000 0x0 0x400000>;

                access = <&can0 0x0>;
                include = <&resource_group_0 0x0>;

                /* 1: block */
                firewallconfig-default = <1 0>;
        };
}
```

Next 6-12 months

- System Device Tree bindings specification
  - Bus Firewalls / System MMU bindings
  - Hypervisor bindings
  - YAML

Devicetree usage for RTOS/microcontroller

Linaro Connect
VIRTUAL • SPRING 2021

Devicetree for RTOS/Microcontroller

- Same reasons devicetree utilized by Linux, BSDs, Xen, u-boot, VxWorks, TF-A, ...
  - Vendor neutral hardware description
  - Convey hardware configuration of system to different software environments (RTOS, middleware, etc)
  - Ability to describe hardware configurations to different software contexts:
    - Asymmetric Multiprocessing
    - Trusted Execution Environments
    - Heterogeneous Systems (Linux + RTOS / MPU + MCU in same SoC)

- Unique concern for microcontroller software is footprint in memory usage, code and data size
  - MCUs having 256K of flash, 64k of memory (BBC Microbit)
  - Devicetree blob - 6k
  - Libfdt - 48k

- Utilize devicetree source as a common data representation and generate code specific to software environment need

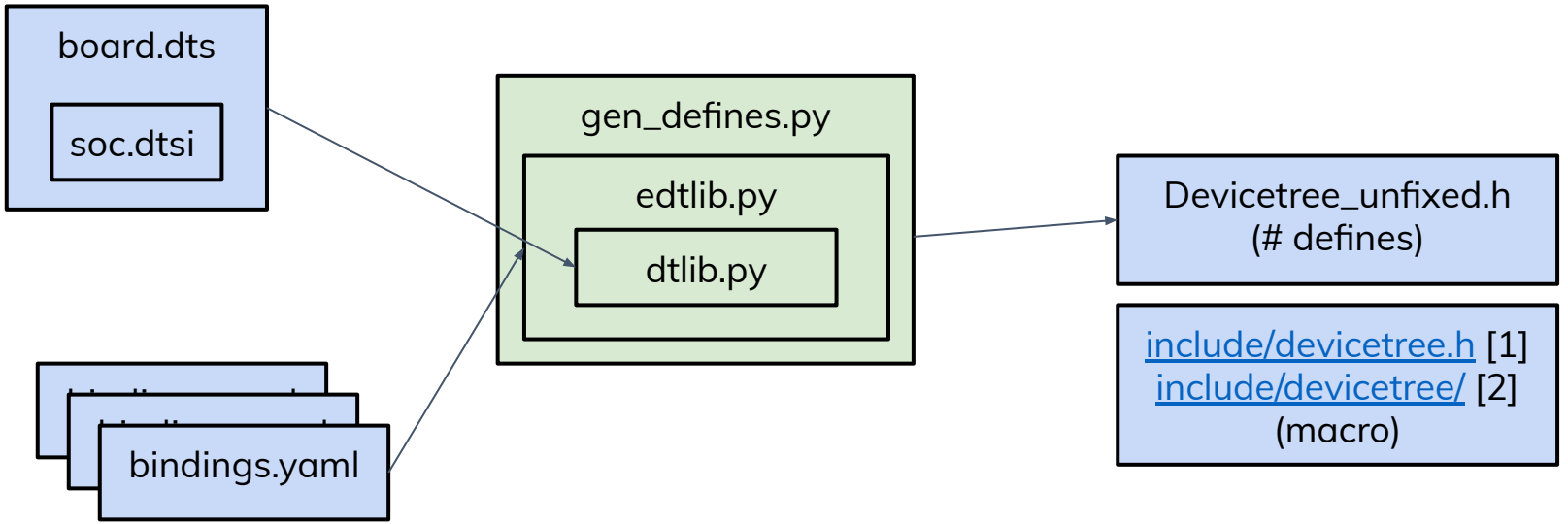
Devicetree codegen RTOS/Microcontroller

- Zephyr RTOS was utilizing Kconfig to describe hardware instance info
- Linaro introduced devicetree to Zephyr in version 1.7 / Jan 2017
- Tooling based around python - need portability across Linux/MacOS X/Windows
- Several iterations of tooling and what code generated looks like
- As of Zephyr 2.5 (Feb 12, 2021) - 3 python scripts for code generation:
  - scripts try not to be Zephyr specific:
  - dtlib.py - pure python devicetree source parser
  - edtlib.py - utilizes dtlib.py + devicetree bindings to provide device data model view (concepts like registers, interrupts, gpios, clocks, etc.)
  - gen_defines.py - utilizes edtlib.py to generate defines
- Generate a set of #defines + macros that are able to extract devicetree data

```
uart0: uart@4006a000 {
        compatible = "nxp,kinetis-uart";
        reg = < 0x4006a000 0x1000 >;
};
```

- Example - `DT_REG_ADDR(DT_NODELABEL(uart0)) /* 0x4006a000 */`

Zephyr code generation workflow



[1] https://github.com/zephyrproject-rtos/zephyr/blob/master/include/devicetree.h
[2] https://github.com/zephyrproject-rtos/zephyr/tree/master/include/devicetree

What's next for RTOS/MCU and devicetree

- Expanding into uses cases that need system devicetree:
  - TFM (Trusted Firmware-M) and ability to utilize system devicetree to describe secure and non-secure resource partitioning and have TFM and Zephyr utilize output

- Work with system devicetree group on ensuring bindings defined for defining contexts and partition handle MCU hardware and usecases
  - Various hardware protection units for MCUs, partitioning of device resources, memory, flash, OpenAMP/remoteproc/rpmsg, etc.

- Work with looper to utilize a common python scripts/libraries to build tools around:
  - Define a common data view of devicetree
  - Have a common python library that can read and write devicetree(s)

- Example system devicetree workflow for Trusted Firmware-M

Devicetree in Standard Linux Boot

Linaro Connect
VIRTUAL • SPRING 2021

Devicetree in Linux and other OS Boot

- EBBR and Arm SystemReady IR
  - Standardizing boot in traditional embedded Linux form factors
  - Allow Separation of OS/Hypervisors layer from Firmware layer
  - Devicetree is an accepted part of this profile
- Cortex-A profile Operating systems and Hypervisors
  - Linux, BSDs, Xen, etc
- The existing boot flow patterns are too varied and too platform specific
  - No clear distinction between Firmware layer and OS layer
  - Varies from SOC vendor to SOC vendor
  - Varies year to year for Firmware / OS pairs from the same SOC vendor
- DT must co-exist with Secure and Measured boot
  - If doing secure boot, all DTB data must be verified by signature
  - If doing measured boot, all DTB data must be measured

Who owns the DTB data?

- Everyone agrees the answer is obvious
  - They just disagree on which answer that is
- Two fundamental models for ownership of DTB used by OS
  - DTB is supplied by Firmware
    - **Simple and better in theory, more standardized**
    - **New HW should just work, no need for OS to know exact model**
    - **Harder to test and enforce**
    - **Not current practice**
  - DTB is supplied by OS
    - **Exact match between DTB and kernel version**
    - **Matches current common practice**
    - **OS works only with boards it knows**
  - Current Plan: Hybrid
    - DTB is supplied by firmware but OS boot manager can override if it knows better
    - **OS works great with boards it knows and OK or great for others based on ABI stability**
    - **Still need ABI and testing**

Device Tree Evolution: Current Activity

- Support for DT overlay source in the upstream kernel
  - Boot loader applied overlays are an existing pattern
  - ***However***, up till now, no place to call upstream
  - Fixing this is 5.12 (started) and rest in 5.13 (or later 5.12-rcN)
  - Build and test apply during kernel build
  - Future work to do more verification on overlay source
- UEFI DT Fixup API for use by OS Boot managers (Grub etc)
  - The DT from OS has a problem when using an OS Boot manager
    - Grub can load a new DTB but does not know how to fixup MAC addr's, SN, etc
  - New UEFI API defined so Grub can call back to firmware to do fixups
    - Implemented in U-boot master
- Proof of Concept build for DT from firmware
  - Firmware: TFA, U-boot, OP-TEE, DTB for OS
  - UEFI based boot w/ or w/o Grub
  - QEMU based but built in a way that does not rely on QEMU DTB magic
  - Example to be used for real HW

Device Tree Evolution: Next 6 to 12 months

- Enable DT from OS model in a standard compatible fashion
  - Complete the Grub based “DTB load w/ firmware fixup” boot model
- Begin testing DT ABI stability
  - Begin testing kernel tip against DTB from previous kernels
  - Start with select boards
  - Start with DTBs from previous 1 or 2 LTS’s
  - Later expand to more boards and maybe new DTB with old kernel
- Reporting tool to check cross project DT synchronization
  - One DT repo “to rule them all” has been rejected
  - Instead cross check project DT source synchronization
  - Tool for project maintainers
- *Define* new DTB file/memory format
  - Many suggestions for new FDT formats have accumulated over the past few years
  - Some or all of the below will make it:
    - Metadata: Source files hash, build date, ***signature of data***
    - Better overlay support: delete node & property, format clean-up
    - Direct support of segmented FDT (base + overlays + fixup)
    - Size reduction: string deduplication or elimination, compression

Conclusion

Linaro Connect
VIRTUAL • SPRING 2021

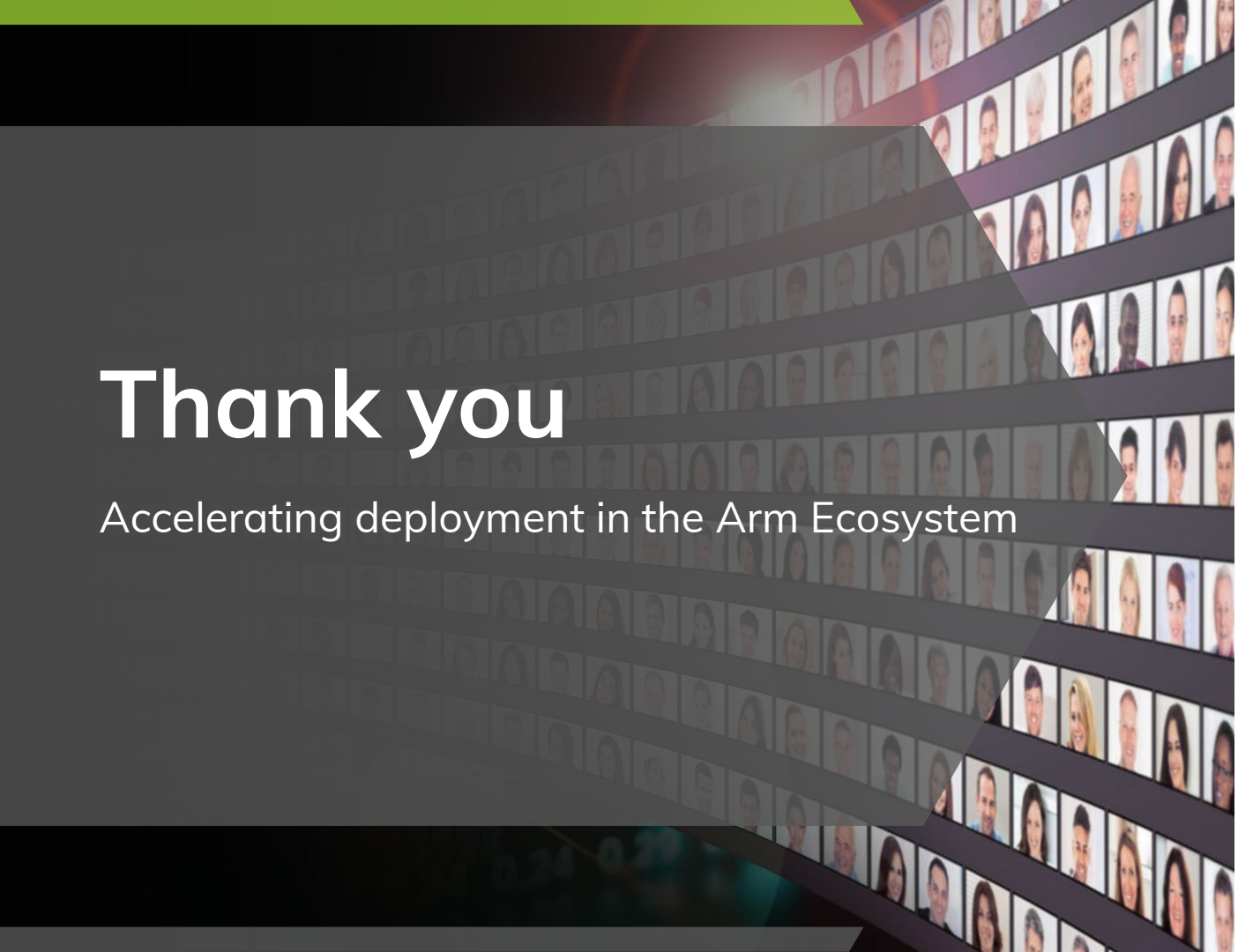
Conclusion:

- Devicetree is an important technology that is driving innovation in many areas
  - Heterogeneous Systems
  - RTOS and High Level OS (Linux etc)
  - Hypervisors
  - Secure OS and Firmware
- Devicetree is driving more standardization
  - Devicetree is more nimble and wider scope than other technologies
  - ***But*** we need stability to drive many use cases like OS boot (Linux, etc)

No Q&A in this session

Please stay for the Birds of a Feather session directly following this one

All speakers in this session and others will be available for questions



Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect
VIRTUAL • SPRING 2021