Designing Quick Starting Embedded Systems Training

Schuyler Patton – Catalog Processors, Linux Software Applications

Systems overview

- **What problem is being solved?**
  Providing a boot time background to enable users of the AM3x/AM4x/AM57x processors to make choices that reduce boot time during system design of a selected processor.
- **Why was this solution developed?**
  Users of the AM3x/AM4x/AM57x processors are concerned about boot time of the processor and seeking ways to minimize it.
- **Brief description of this system solution:**
  - Understand the boot time components of the catalog processors, system, and the Processor Linux Software Development Kit (PLSDK) for the AM335x, AM437x, and AM57x.
  - Learn first steps and capabilities to reduce boot time using the Processor SDK without doing significant customization. This presentation gives developers a look beyond just the initialization of the selected OS.
- **What steps are recommended to evaluate this solution?**
  This presentation shows steps that can be used to evaluate and design a boot time.

Detailed agenda

- Defining “boot time”
- Out-of-the-Box (OOB) boot times of the TI Processor Linux SDK
- Hardware elements of boot-time design in an embedded system
- Software elements of boot-time design in an embedded system
- Single user boot times on the Catalog EVMs
- Designing boot time
- Summary

Defining Boot Time

Defining boot time

- Does boot time matter?
- One common definition for boot time is that it is defined as the time from system power-on to full system availability. The distinction here is that this does not mean full application capability. This will be unique to the application.
- The boot time expectations can be different based on system complexity.
- The time required to achieve complete system availability can be more than product developers want.
- Often the common view is that the OS is the main contributor to boot time duration.
- Are there decisions that can be made during system design that can improve the product boot time?

Boot time: Does it matter?

- This is a question and a decision that should be made at the start of the system design due to potential impact on cost and development time.
- A boot time requirement may require hardware support, software, run-time, or unique application development eco-system for the product.
- Is the boot time requirement needed for user satisfaction, product domain expectation, or product build and test requirement?

Sitara™ ARM® Processors

Texas Instruments

Image Storage

Runtime Init

Application Development

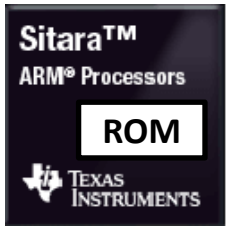
Boot Board Diagnostic

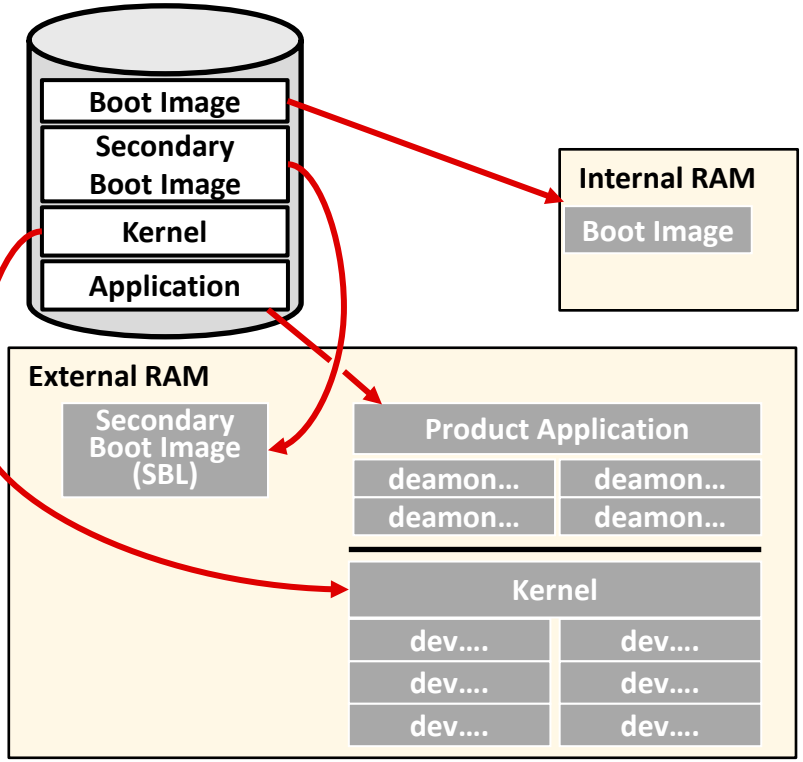
Boot Board Flashing

Boot Board Sys Cfg & Test

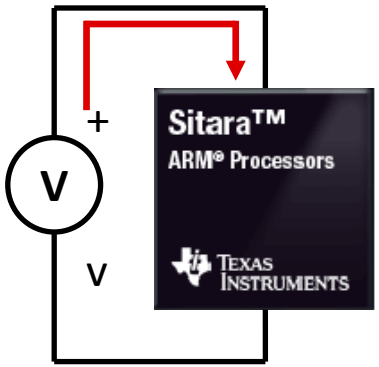
What is the boot process? … simplified

- Once the HW releases Sys_Reset, the ROM loads a boot image into internal RAM.
- The boot image may either load a secondary into external RAM or load the OS kernel into external RAM.
- The OS image needs time to initialize itself and drivers.
- The OS image starts the user environment and product application, which also needs time to initialize to full system availability.
- Completing application init is described as **full application availability**.
- All this takes time.





Looking at a system boot to application - Linux



AMxx

- Boot Mode Storage Interface Init
- Clocks
- DDR

Kernel

- Graphics: PowerVR SGX
- Display: LCD/HDMI, Touch Scr
- PRU-ICSS
- Kernel
- MMC/SD/SDIO
- GPIO
- EDMA
- UART
- SPI
- I²C
- McASP
- CAN
- USB
- EMAC
- GPMC
- Timers
- WDT
- RTC
- eHRPWM
- eQEP
- eCAP
- ADC

User Space

- udev
- systemD
- bluetooth
- udhcpd
- sound
- weston
- debugfs
- various FS
- telephony
- avahi
- telnetd
- thttpd
- dropbear
- lightpd
- login
- agetty

RBL **MLO/SPL** **U-Boot** **Kernel** **User Space Init**

What is the application?

- Obviously, embedded products have targeted or unique applications.
- The user interfaces and interactions are unique as well.
- One common method of determining application complexity is considering the user interface. If it has no display, this is considered “headless,” but should not be assumed to be less complicated.

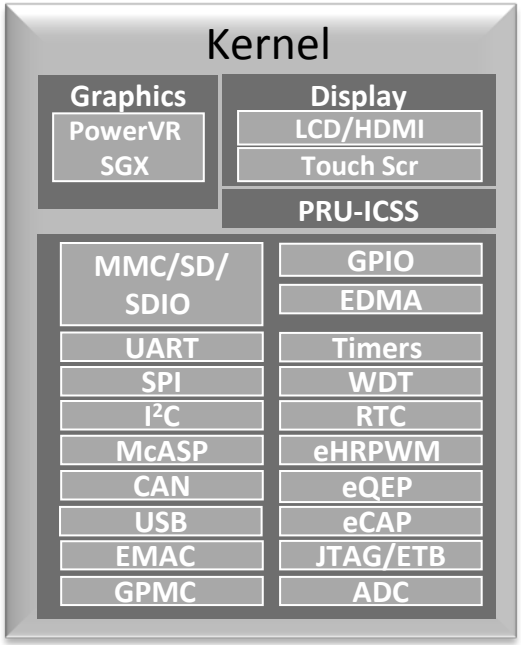
Examples of “headless” devices (no display).

Examples of display devices

Single-user systems

- Let's use the definition of "single user" to mean essentially a single-user thread/application.
- Single-user applications "may" use fewer resources.

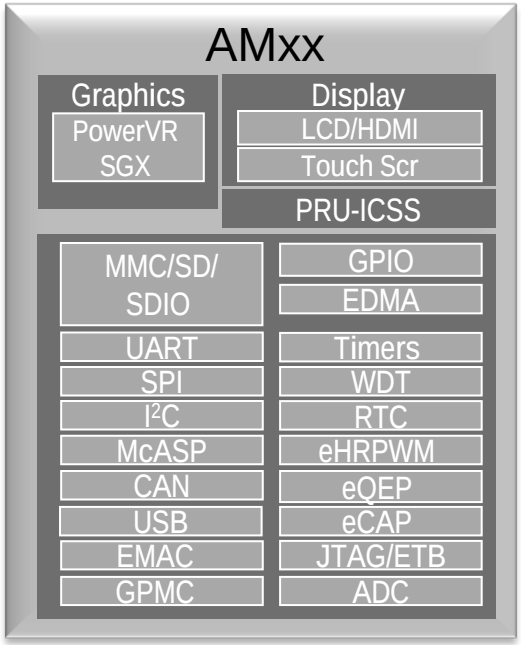
- Fewer resources can translate to less processor features to enable and therefore less boot time.



Multi-user systems

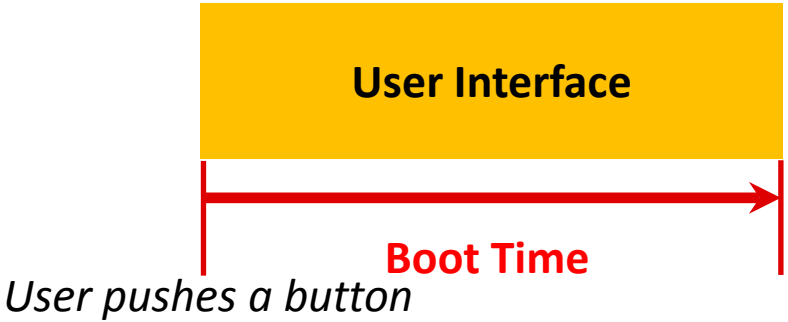
- Let's use the definition of "multi user" to mean essentially a multi-user thread/application that requires a support infra-structure that requires initialization.
- Multi-user applications "may" use most or all SOC resources.

- More resources can translate to more processor features to enable and this could mean an extended boot time

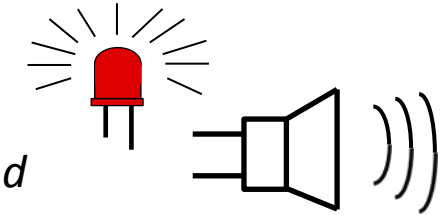


The user perspective of boot time

- A user wants to see some indication or “proof of life” from the device after power initiation through even the minimalist user interface.
- Embedded devices can have a variety of ways to indicate they are ready for operation.



Light, sound



Display

- The indication does not mean full system availability but that the boot process is proceeding.

Summary: Defining boot time

- Does boot time matter for the product?
- The boot process of an “embedded product.”
- There various stages of the Linux boot process.
- Product requirements can affect boot time, such as the difference between single-user and multi-user application.
- Proof-of-life indications to the user are important for extended boot times.
- When is the best time to consider the boot time requirements for the product?

Out-of-the-Box (OOB) boot times of the TI Processor Linux SDK

OOB boot times of the Processor Linux SDK

- What is the OOB boot time of the Processor Linux SDK (PLSDK) and where to find this information?
- Break down the components of the OOB boot time.
- Describe the multi-user space.
- Describe the single-user space.

Boot time measurements OOB PLSDK

- This data is from the performance data sheet released with the SDK and it shows two different timings per SOC.

	Boot Configuration	2016.06			
		am335x-evm	am43xx-gpevm	am57xx-evm	k2g-evm
		boot time (sec)	boot time (sec)	boot time (sec)	boot time (sec)
Multi-user	Kernel boot time test when bootloader, kernel and sdk-rootfs are in mmc-sd	35.31 (min 35.0, max 35.67)	32.2 (min 32.01, max 32.34)	19.51 (min 19.17, max 20.4)	56.34 (min 56.17, max 56.56)
Single-user	Kernel boot time test when init is /bin/sh and bootloader, kernel and sdk-rootfs are in mmc-sd	5.49 (min 5.47, max 5.52)	5.88 (min 5.17, max 6.23)	5.58 (min 5.53, max 5.59)	9.19 (min 9.08, max 9.44)

- Multi-user refers to an Initialization Run Time Target or Run Level. This is the initialization for the OOB PLSDK.
- Single-user refers to shell.

Break down the components of the OOB boot time

- The OOB SDK is booting a desktop-like environment, since this provides the necessary support for OOB product demos and development.
- Because of the support necessary for the OOB, user space initialization takes a long amount of time to achieve full system availability.
- Strongly emphasize that the OOB PLSDK environment was not intended for production.

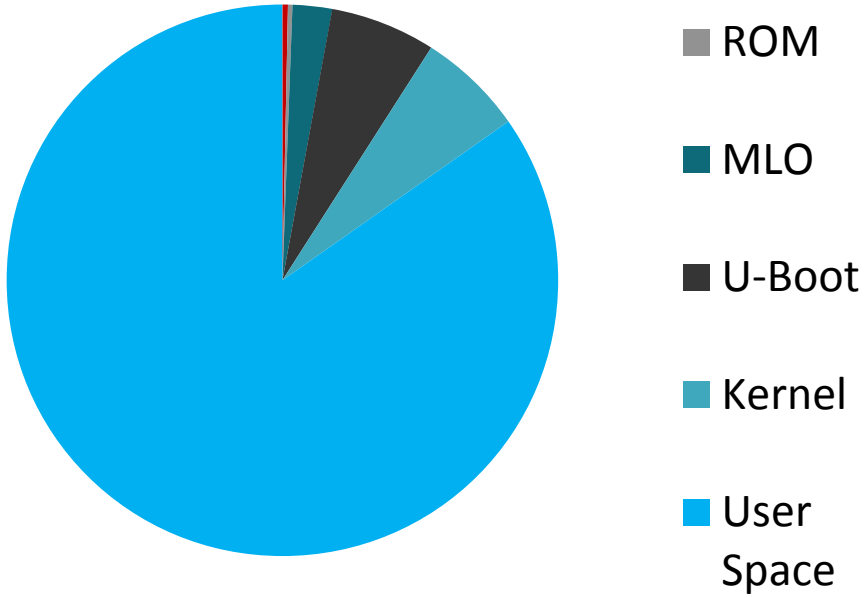
The multi-user space

- The OOB user space uses SystemD which is a centralized user level daemon for starting/stopping/controlling/configuring system services
- SystemD is called right after kernel initialization and – based on a target selection needed by user space – starts several services that have defined dependencies.
- SystemD uses udev and dbus to get events and communicate between services.
- SystemD replaced SysVinit, which has several examples on how to reduce boot time.
- Getting to the login prompt on the console from a cold boot with SystemD takes about 18s. Here is the abbreviated grabserial output:

```
[8.961440 ] systemd[1]: System time before build time, advancing clock.
[26.856867] am57xx-evm login:
```

AM437x OOB SDK boot time breakdown by stage

- This is the boot time by stage for an OOB PLSDK for the AM437x.
- The only change made was U-Boot boot delay was changed from a default of 2 to 0.
- The length of time to initialize user space is due to the required support for demos and a development work space in the OOB PLSDK.
- If the boot time requirement is significantly less than the user space time demonstrated, this would be a reason to think about another option.



The single user or thread user space

- The single-user or thread user space is initiated after the kernel has initialized.
- Single User or shell mode is initiated on the kernel command line by passing the location of the executable to run.

```
Kernel command line: console=... init=/bin/boot_app.sh ....
```

- The capabilities of the shell are completely up to the writer.
  Example shell script:

```
#!/bin/sh

mount -t proc proc /proc
mount -t sysfs sysfs /sys

/bin/sh
```

- The last line of the above script could be the command that launches the application rather than invoking another shell.

Summary: OOB boot times of the Processor Linux SDK

- Looked at the boot times for the OOB PLSDK.
- Discussed the components of the OOB PLSDK and why they contribute to the length of boot time user experience.
- The pie chart illustrated that the bulk of the OOB PLSDK boot time is the initialization of the user space.
- Discussed the single-user space, how to initiate it, and that it is a single shell or application that the user controls.

Hardware elements of boot time design in an embedded system

Hardware elements used to define boot time

- Boot Modes
- Power Management Integrated Circuits (PMIC)
- Processor Operating Performance Point (OPP)

Choosing processor boot mode

- Boot modes have a significant impact on boot time.
- Memory device boot modes such as MMC, QSPI, NAND, NOR should be first in the boot order.
- Example of Beagle Bone Black boot mode:
  - The first row is the default.
  - The second is the user switch which can select a different mode
- Example of the AM437 Starter Kit EVM boot mode:
  - The first is the default, MMC0.
  - The fourth is QSPI.
  - A significant timeout takes place between MMC0 to QSPI.
- Peripheral-based boot modes – such as EMAC, USB, UART – should be later in the boot order. These boot modes would be good for “in place” device programming.

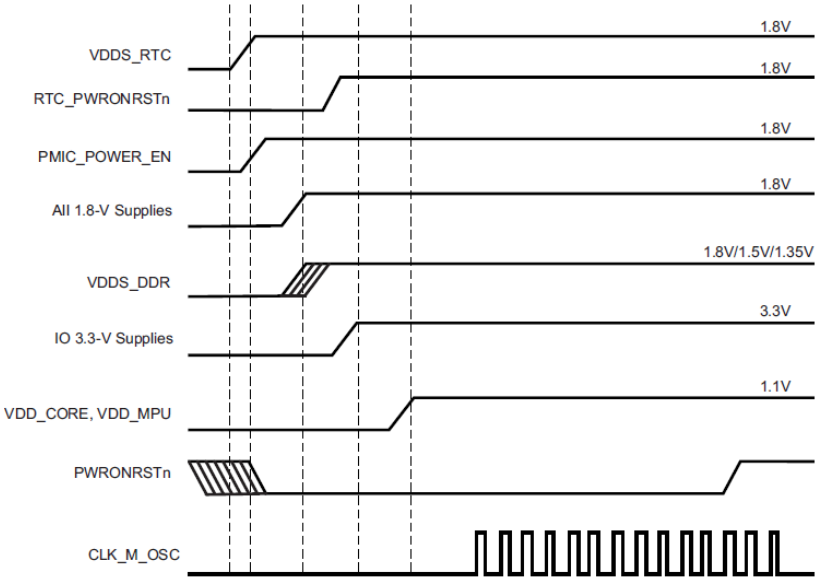
SYSBOOT[4:0]	Boot Sequence			
00101b	UART0	XIP (MUX1)[2]	SPI0	NANDI2C

SYSBOOT[4:0]	Boot Sequence			
11100b	MMC1	MMC0	UART0	USB0[5]
11000b	SPI0	MMC0	USB0[5]	UART0

11000b	MMC0	USB_MS (USB1)	USB_CL (USB0)	QSPI

PMICs – power sequencing the processor

- Later in the presentation there will be more details presented on the PMIC power up and how it can effect boot time.
- The reason for needing a PMIC is that the processor voltage rails need to sequenced on startup in a particular order and OPP voltages managed as part of DVFS support. This is done by the PMIC.
- Each Processor ROM is expecting the PMIC to be at a certain OPP mode defined by the datasheet.



Processor operating performance points

- AM335x from PORz runs ROM at 600MHz OPP100 voltage.
- AM4378 from PORz runs ROM at 300Mhz OPP50 voltage
- AM57x runs the ROM at 588MHz.
- After MLO loads and runs, the processor clock and voltages are set as needed.

FUNCTION	AM3351	AM3352
ARM Cortex-A8	Yes	Yes
Frequency(1)	300 MHz 600 MHz	300 MHz 600 MHz 800 MHz 1000 MHz
MIPS(2)	600 1200	600 1200 1600 2000

FUNCTION	AM4378
ARM Cortex-A9	Yes
Frequency	300 MHz 800 MHz 1000 MHz
MIPS	2000 2500

Summary: Hardware elements used to define boot time

- Boot modes – make sure the primary boot mode is from a memory boot device
- Power Management Integrated Circuits (PMIC) – this device can impact boot time
- Processor Operating Performance Point (OPP) – ROMs start at a lower OPP level that have will impact boot time.

Software elements of boot time design in an embedded system

Software elements for designing boot time

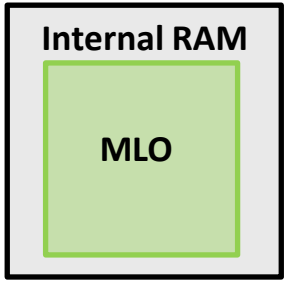
- MLO/SPL of the U-Boot code base has a mode called Falcon that allows skipping U-Boot to boot the kernel directly.
- There are several tutorials on the internet for reducing Linux kernel initialization time.
- For this discussion, a few common techniques are introduced that can be used to significantly reduce kernel initialization time. These techniques do not require code customization or configuration changes to the OOB PLSDK Linux kernel.
- Device Tree source files
- User Space choice

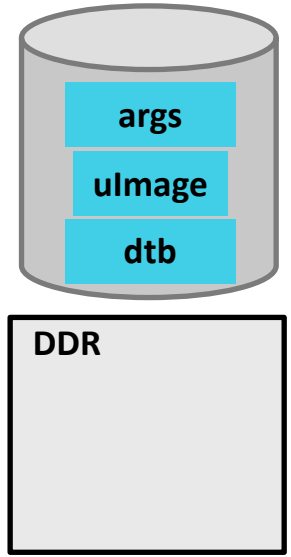
U-Boot Falcon mode

- Falcon mode is a configurable mode of U-Boot that allows MLO/SPL to load the kernel directly without having to load U-Boot.
- Boot sequence with U-Boot, 3 stages:

- Boot sequence with Falcon mode, 2 stages.

- To use Falcon mode, MLO/SPL must be able to read data that indicates where to find the kernel, kernel parameters, and the DTB, so that they can be loaded into DDR and started.





Linux kernel initialization time optimizations

- There are several very good articles on the web that describe these steps. So there is no need to go into detail on all of them here.
- Here are just a few of several suggestions that can be found:

  - Reducing kernel size through configuration
  - Kernel compression
  - Add quiet to the command line
  - Add loops per jiffy LPJ to the command line
  - Selecting which drivers are initialized

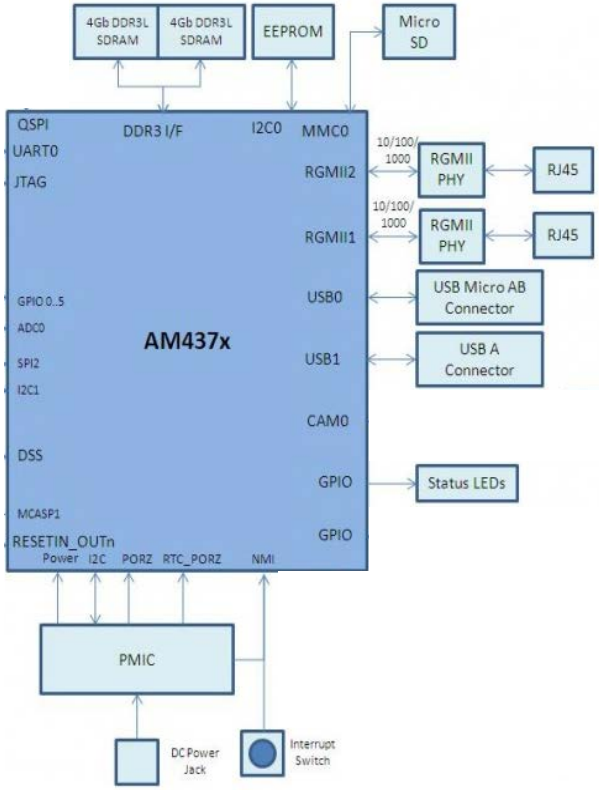
- One goal of this presentation was to show significant time can be saved with just these techniques of kernel compression, quiet boot, and driver initialization.

Linux kernel compression

- The default kernel compression of the SDK uses LZMA, which is a compression technique focused on size efficiency.
- By switching to LZ4, the compression changes from *size* efficient to *decompression* efficient. The kernel is larger than the kernel compressed with LZMA, but the decompression speed more than makes up for the additional transfer time from storage to DDR.
- One example using the AM437:
    - SDK pre-built zImage size compressed with LZMA: 3489392 bytes
    - SDK zImage compressed with LZ4: 5511632 bytes **(over 2MB larger)**
    - Yielded a decompression improvement of **1.80 seconds**
    - This demonstrates the possible **improvement on boot time by changing the kernel compression types.**

Device tree source file

- The default DTB files shipped with the SDK bind board features to the kernel.
- A product board DTB might have more or less features.
- The size of the DTB can impact the boot time.
- AM437 SK EVM OOB DTB kernel init time – 1.49s
- AM437 reduced feature example DTB kernel init time – 0.97s, .52S faster
- Key point is to only enable the kernel features necessary for the product.



Results were done with passing quiet and an LPJ value on the kernel command line. Times were gathered from dmesg.

User space

- As mentioned in an earlier section, the OOB SDK user space type (single/multi user) decides the init system which impacts boot time
- The developer is the owner and implementer of this decision
- Is the product application a single-thread type or does it require the infrastructure of a multi-threaded system?
  A single-threaded app may be more cumbersome to develop due to lack of accustomed infrastructure.
- The infrastructure required for the application may decide the init system.
- The user space choice will be based on the application infrastructure and boot time requirements

Summary: SW elements in designing boot time

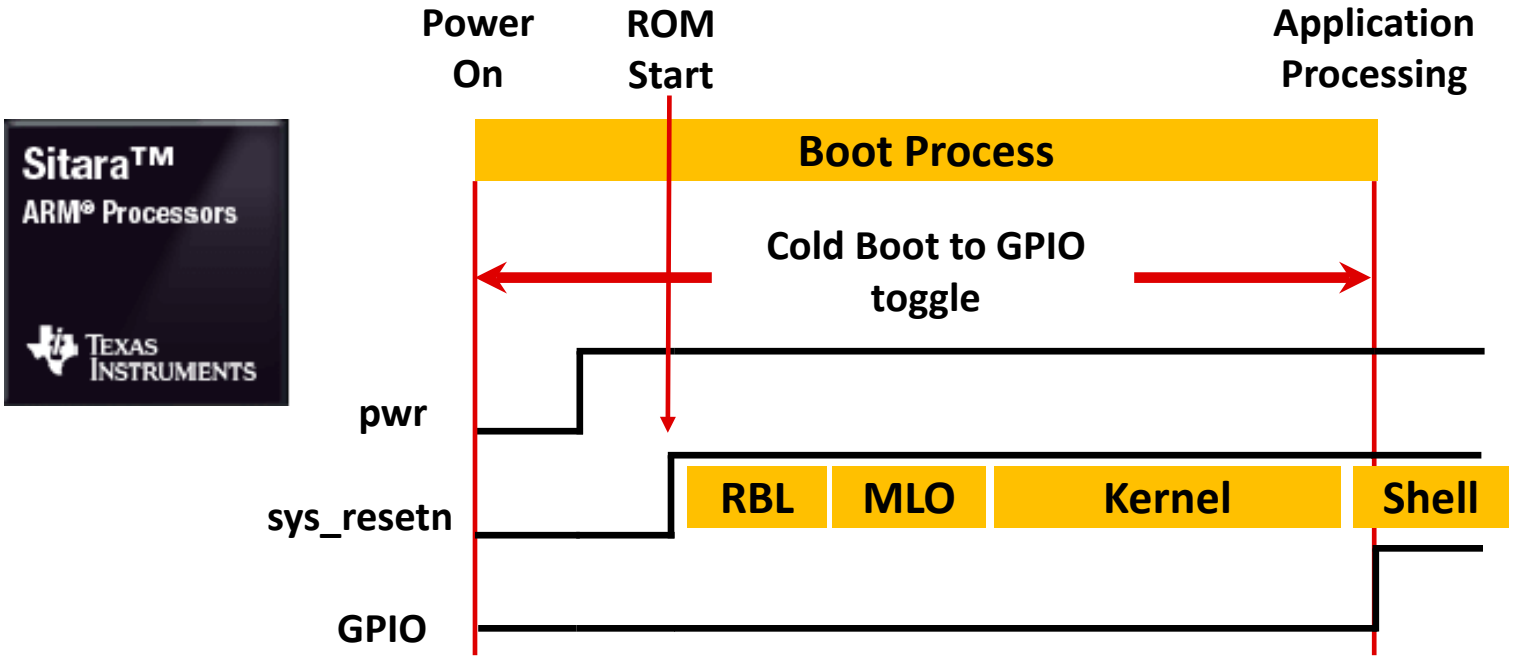
- The first topic discussed was how Falcon mode can boot the kernel directly without having to load U-Boot.
- Next the presentation discussed that there are several kernel initialization tutorials on the web and discussed which ones provide significant boot time improvement without having to customize the OOB SDK.
- A Demonstration how the size of the DTB file can affect boot time:
  - If the system only uses a few peripherals, then the DTS file should only enable those nodes that are used.
  - Do not use the example EVM DTS with minor modifications unless it matches the product exactly
- The last topic was about user space choice which is based on application and boot time requirements.

Single-user boot times on the Catalog EVMs

Single user boot times on catalog EVMs

- This section describes the boot time for a single user type environment
- Initial test setup environment
- Review boot time of a single user:
  - Beagle Bone Black
  - AM437 Starter Kit
  - AM572x Industrial Development Kit (IDK)
- Overview of PMIC processor interaction
- All numbers presented in this section are derived by experiment and are for reference purposes only.

How the single user boot time was measured



This diagram emphasizes that this is the timeline to the start of application processing.

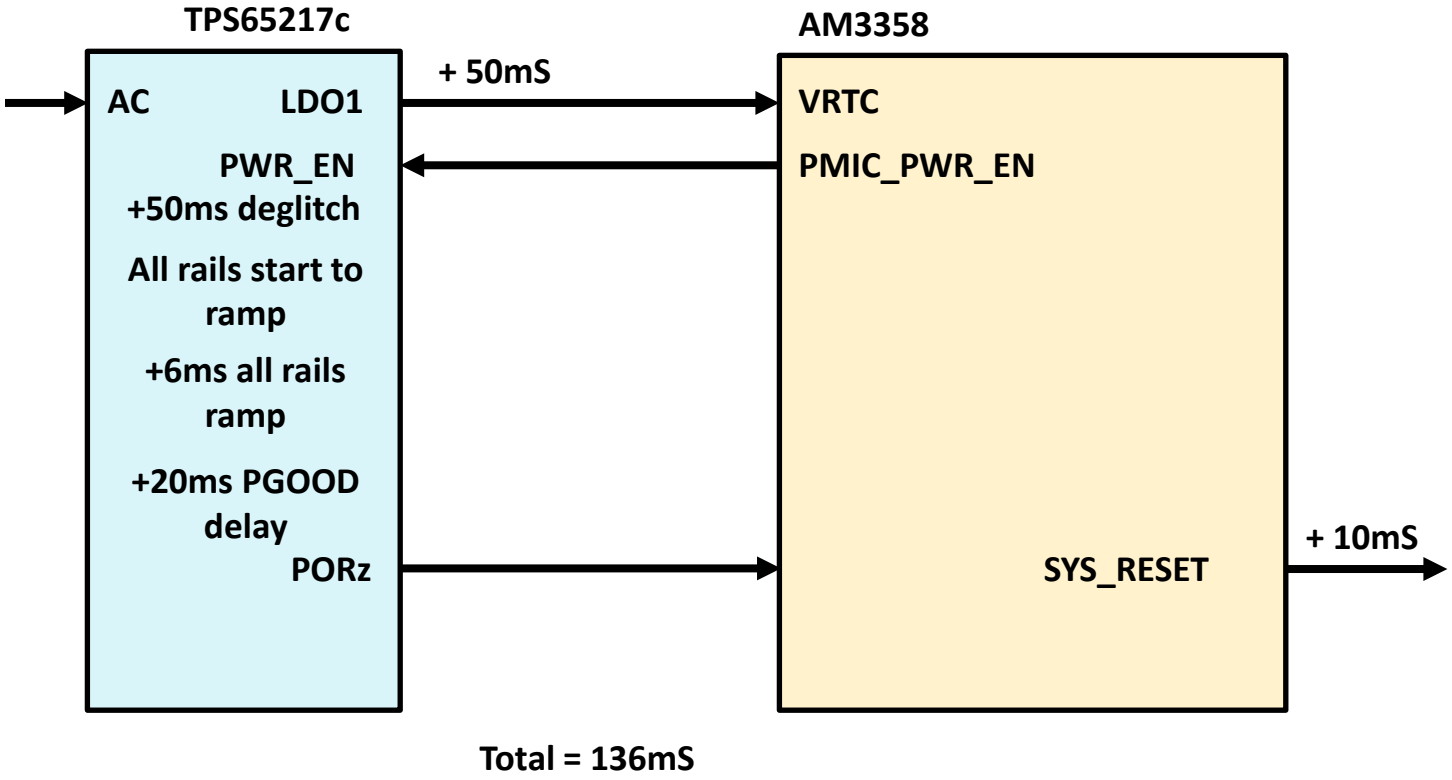
Closer look at components of a BBB boot time
using init=/bin/sh – eMMC

- From power on
- Reduced DTB / kernel compressed with LZ4

Kernel Decompress time

≈ 1.0S

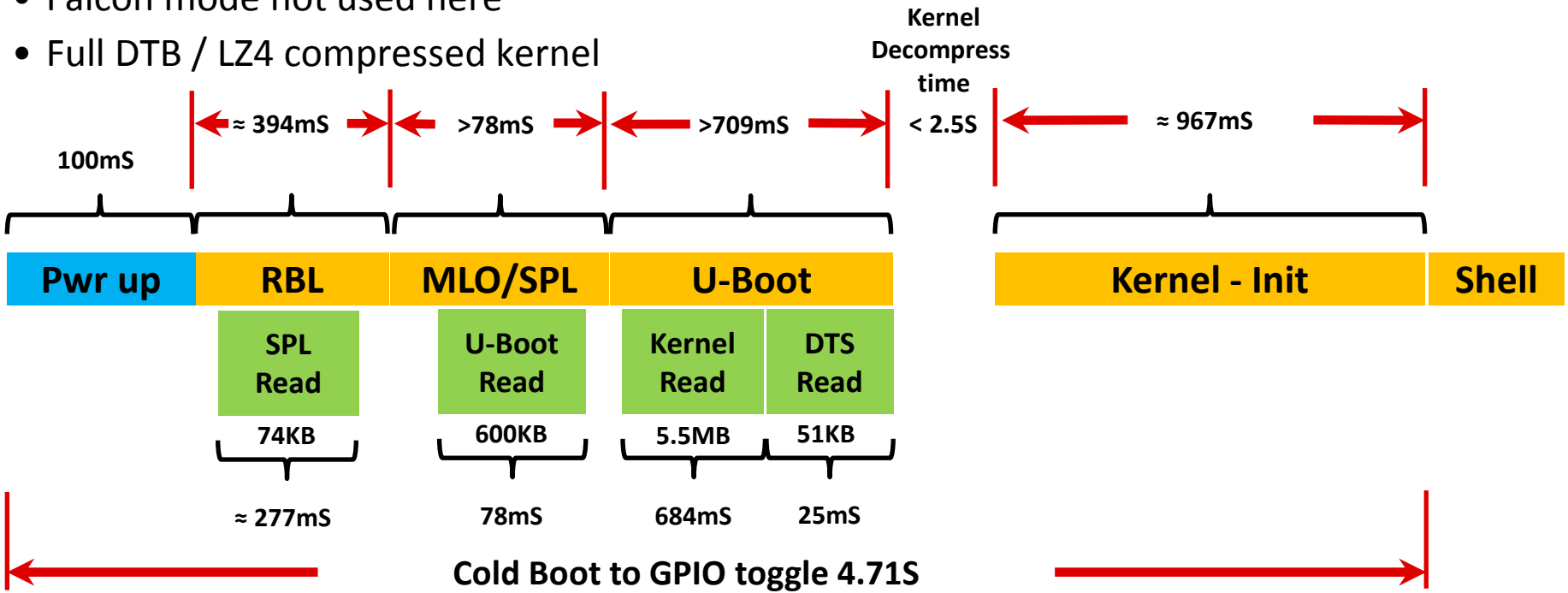
Breakdown of time to ramp TPS65271c



Closer look at components of AM437SK boot time

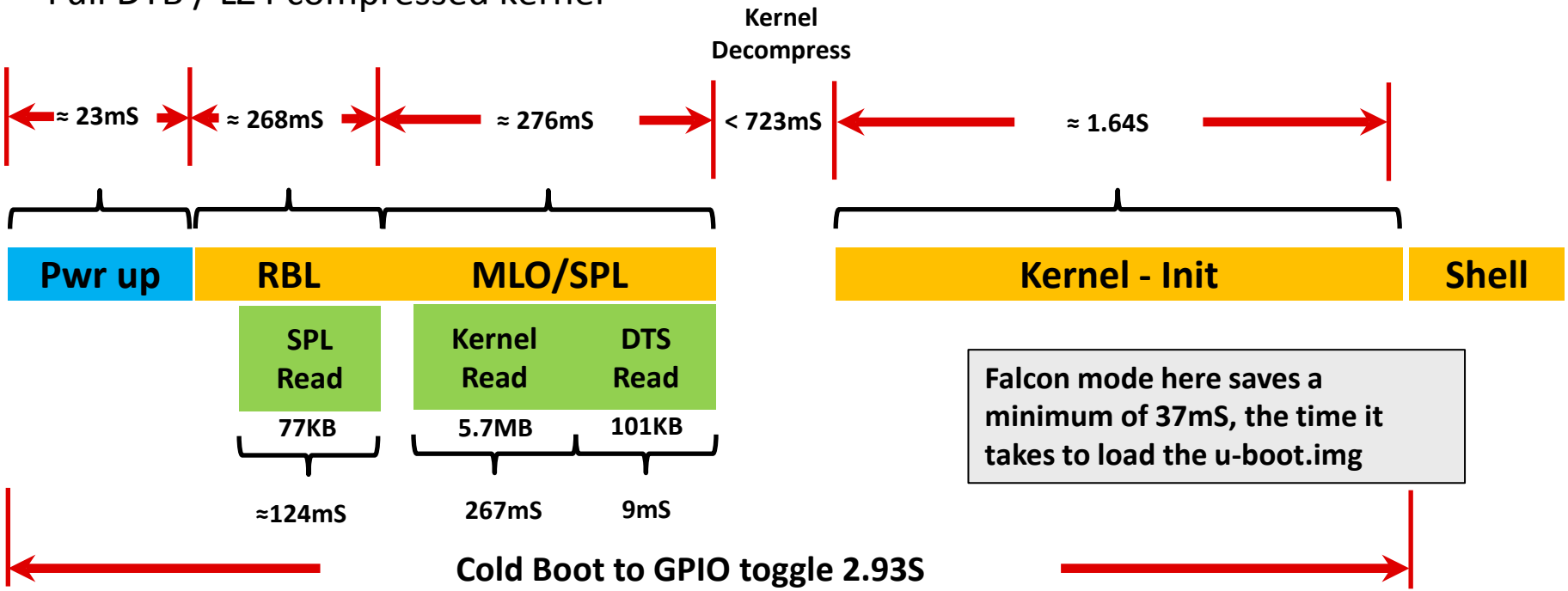
using init=/bin/sh – SD

- From power on
- Falcon mode not used here
- Full DTB / LZ4 compressed kernel



Closer look at components of AM572x-IDK boot time using Falcon mode & init=/bin/sh

- From power on
- Full DTB / LZ4 compressed kernel



Summary: Single user boot times on Catalog EVMs

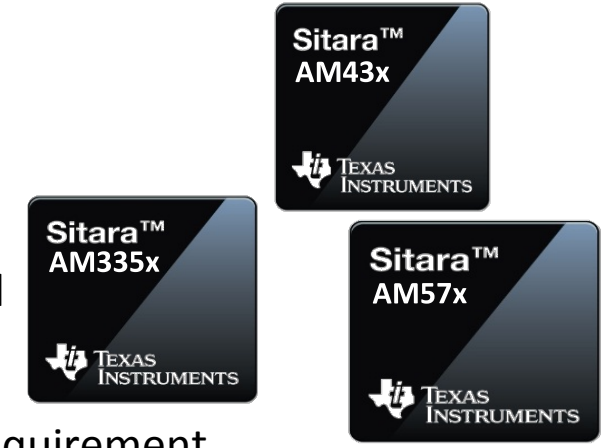
- Described the minimalistic elements for booting a single user system or shell.
- Defined how the boot to application time was measured.
- Discussed boot times for the AM3x/AM4x/AM5x.
- Demonstrated for each processor how power on through kernel init was affected by various factors such as:
  - PMIC
  - Processor OPP
  - Falcon mode
  - DTB size
- **The key takeaway here is that an embedded application boot time is very dependent on the processor features used and the support required to initialize them.**

Designing System Boot Time

Designing system boot time

- Putting together all the elements from this presentation
- When should the system developer set the boot time requirements
- Pick a boot mode that has local storage first in the boot order
- Review rough numbers of boot time
- Deciding on the user space
- Overall, achieving boot time is asymptotic in nature

Review: Designing boot time



- The processor was most likely picked for features rather than boot time. So the developer must be aware of the processor capabilities such as the OPP of the processor and the impact on boot time.
- System developer decides if the product has a boot time requirement.
- Choose memory storage as the primary boot mode, such as MMC, NAND, NOR, QSPI

11000b	MMC0	USB_MS (USB1)	USB_CL (USB0)	QSPI

Rough numbers to start analysis

- Some rough numbers presented in this application for quickly thinking about what the processor boot time potentially could be.
- The example here is based on the EVMs using MMC boot mode, the PMIC ramp times, and ROM to read in the OOB SDK MLO images:

Pwr up	**136mS – AM3x**	**101ms – AM4x**	**22ms – AM572x**
RBL	**237mS – AM3x**	**394ms – AM4x**	**268ms – AM572x**

MMC boot – includes reading in MLO

- The point here is that each processor has a minimum time that the it takes to get to the MLO image read in by ROM.
- One tradeoff is to consider XIP boot mode if the processor has support to avoid the ROM reading in MLO image.

Review: Falcon mode or full MLO/U-boot

- Falcon mode allows the direct loading of the kernel and DTB.

- This presentation demonstrated that bulk of the time saved by using Falcon mode is the time is takes to load U-Boot.

Review: Kernel initialization

- Kernel compression – switch from LZMA to LZ4
  - On AM437x with the released SDK kernel, this yielded an decompression improvement of **1.80 seconds**
- Kernel command line **quiet** saves about 500mS or more depending on kernel init.

```
Kernel command line: console=... quiet ....
```

- Kernel DTB sized for the system:
  - Board DTS should be sized for the system. Starting with an EVM DTS file and leaving nodes from the EVM that are not used in target system will have a negative impact on boot time.
  - On AM437x, a reduced DTB to emulate a smaller system initialized over 500mS faster.

Review: User space initialization

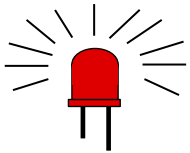
- The user space initialization is a big consideration. A quick review of the SDK performance datasheet indicates the boot times for multi-user and single-user.

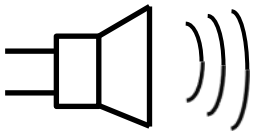
Boot Configuration	2016.06			
	am335x-evm	am43xx-gpevm	am57xx-evm	k2g-evm
	boot time (sec)	boot time (sec)	boot time (sec)	boot time (sec)
Kernel boot time test when bootloader, kernel and sdk-rootfs are in mmc-sd	35.31 (min 35.0, max 35.67)	32.2 (min 32.01, max 32.34)	19.51 (min 19.17, max 20.4)	56.34 (min 56.17, max 56.56)
Kernel boot time test when init is /bin/sh and bootloader, kernel and sdk-rootfs are in mmc-sd	5.49 (min 5.47, max 5.52)	5.88 (min 5.17, max 6.23)	5.58 (min 5.53, max 5.59)	9.19 (min 9.08, max 9.44)

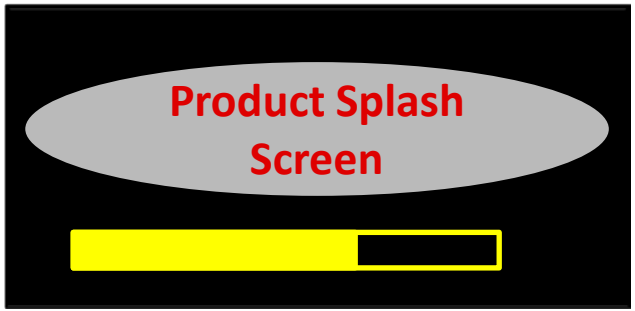
- The system developer decides if the product has a boot time requirement.
- User space choice has an impact on development eco-system, too.
- The system developer is the owner for the user space choice and development.
- Unfortunately, this choice often gets left until the end of a project, sometimes requiring the developer to address this issue with a user space change.

Review: User space initialization – (cont.)

- If the product requires a multi-user system, then a “proof of life” type boot stage might be a choice. If that is the case, then U-Boot has to be loaded and “customized” to provide the “proof of life” indication.





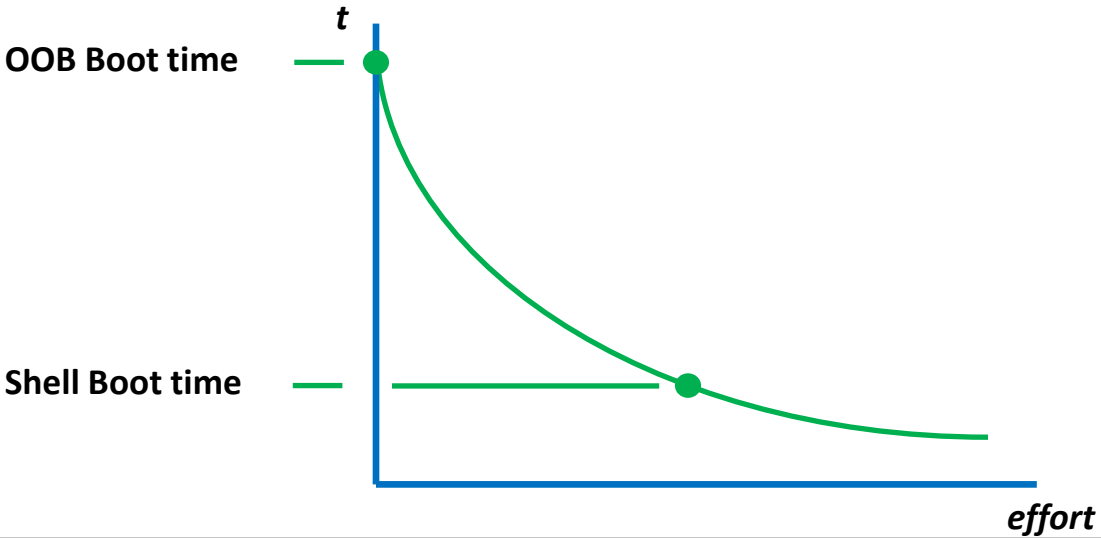


- The system boot time requirement is different from a fully running application. Examples shown here are only to a command prompt for the single user case.

- The system developer must do additional work to determine their unique application boot time.

Boot time reduction becomes an asymptotic effort

- Overall boot time design effort depends on the boot time requirement of the product
- OOB vs heavy customization
- This is where developers will be doing significant experimentation and development to achieve the desired boot time



Summary: Designing system boot time

The goal of this section was to review each element of determining boot time:

- Suggested choosing a primary boot mode that is memory-based
- Discussed rough numbers to guide determined boot time
- Suggested employing Falcon mode if supported by the processor
- Discussed a few effective steps to reduce the Linux kernel initialization time
- Examined user space initialization and the impact on boot time, development, and the need for proof-of-life indicators on extended boot times
- Discussed how boot time development can be asymptotic

Training summary

- The earlier that the developer decides the boot time in the design process, the better.
- Pick the right boot mode, not just for the product, but perhaps manufacturing as well.
- There are some HW elements – such as OPPs, PMIC – that impact boot times.
- The are SW elements that help in shortening a boot time, such as Falcon mode, DTB size, kernel compression, and boot time parameters.
- The developer's choice of user space for single user or multi-user can impact product development time.
- The techniques presented here are a way point on the way to the design the developer requires, there are not any one step actions to achieve a desired boot time.
- Be prepared for experimentation on boot time, as each product is unique and **actual results will vary**.

For more information

- Designing Quick Starting Embedded Systems Training Series: http://training.ti.com/designing-quick-starting-embedded-systems-training-series
- Sitara Processors Overview: http://www.ti.com/sitara
- Embedded Linux Wiki discussion of how to reduce boot time:
  - http://elinux.org/Boot-up_Time_Reduction_Howto
  - NOTE: A web search of “Linux Boot Time Reduction” yields several more presentations
- Embedded Linux Wiki discussing boot time measurement tools/techniques : http://elinux.org/Boot_Time
- For questions about this training, refer to the E2E Community Forums at http://e2e.ti.com